In [ ]:
```
#Spotify song popularity year wise prediction
#group 24
#Siriwardhana T.D.R.D   EG/2020/4219
#Rishinath    K.        EG/2020/4165
```

In [364…
```
# @title Import Libraries

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

In [365…
```
# @title Load data
df = pd.read_csv('top10s.csv',encoding='ISO-8859-1')

df.head()
```

Out[365]:

	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	p
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	80	217	19	4	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	64	263	24	23	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	71	200	10	14	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	71	295	0	4	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	43	221	2	4	

In [366…
```
df.shape
```

Out[366]:
```
(603, 15)
```

In [367…
```
df.isna().sum()
```

Out[367]:
```
Unnamed: 0    0
title         0
artist        0
top genre     0
year          0
bpm           0
nrgy          0
dnce          0
dB            0
live          0
val           0
dur           0
acous         0
spch          0
pop           0
dtype: int64
```

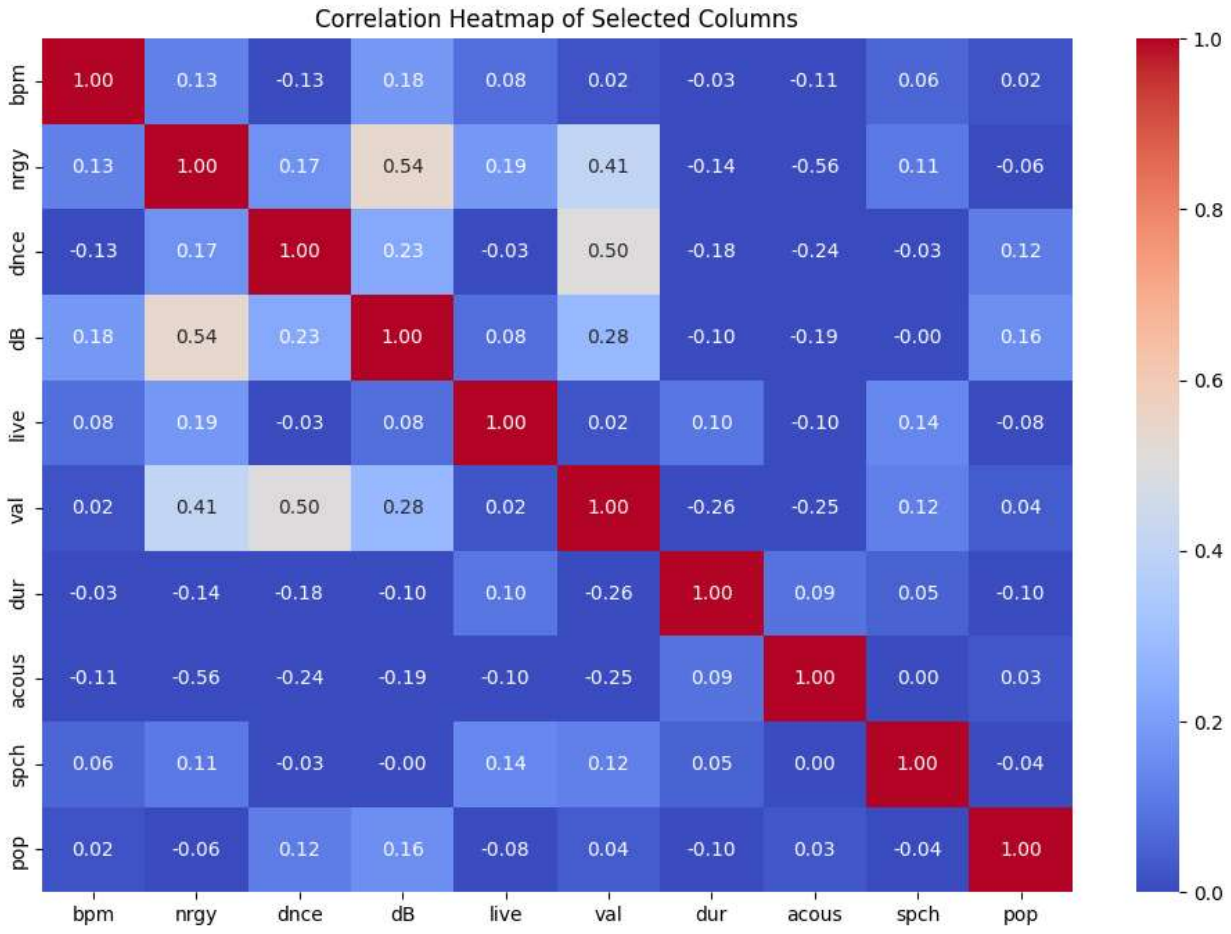
In [368…
```
df.columns
```

Out[368]:
```
Index(['Unnamed: 0', 'title', 'artist', 'top genre', 'year', 'bpm', 'nrgy',
       'dnce', 'dB', 'live', 'val', 'dur', 'acous', 'spch', 'pop'],
      dtype='object')
```

In [369…
```
print(df['title'].nunique())
print(df['artist'].nunique())
print(df['top genre'].nunique())
```

```
584
184
50
```

In [370…
```
# @title checking the correlation
selected_columns = ['bpm', 'nrgy', 'dnce', 'dB', 'live', 'val', 'dur', 'acous', 'spch'

# Extracting the selected columns
df_selected = df[selected_columns]

# Calculating the correlation matrix
correlation_matrix = df_selected.corr()

# Creating a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=0, vmax=1
plt.title('Correlation Heatmap of Selected Columns')
plt.show()
```

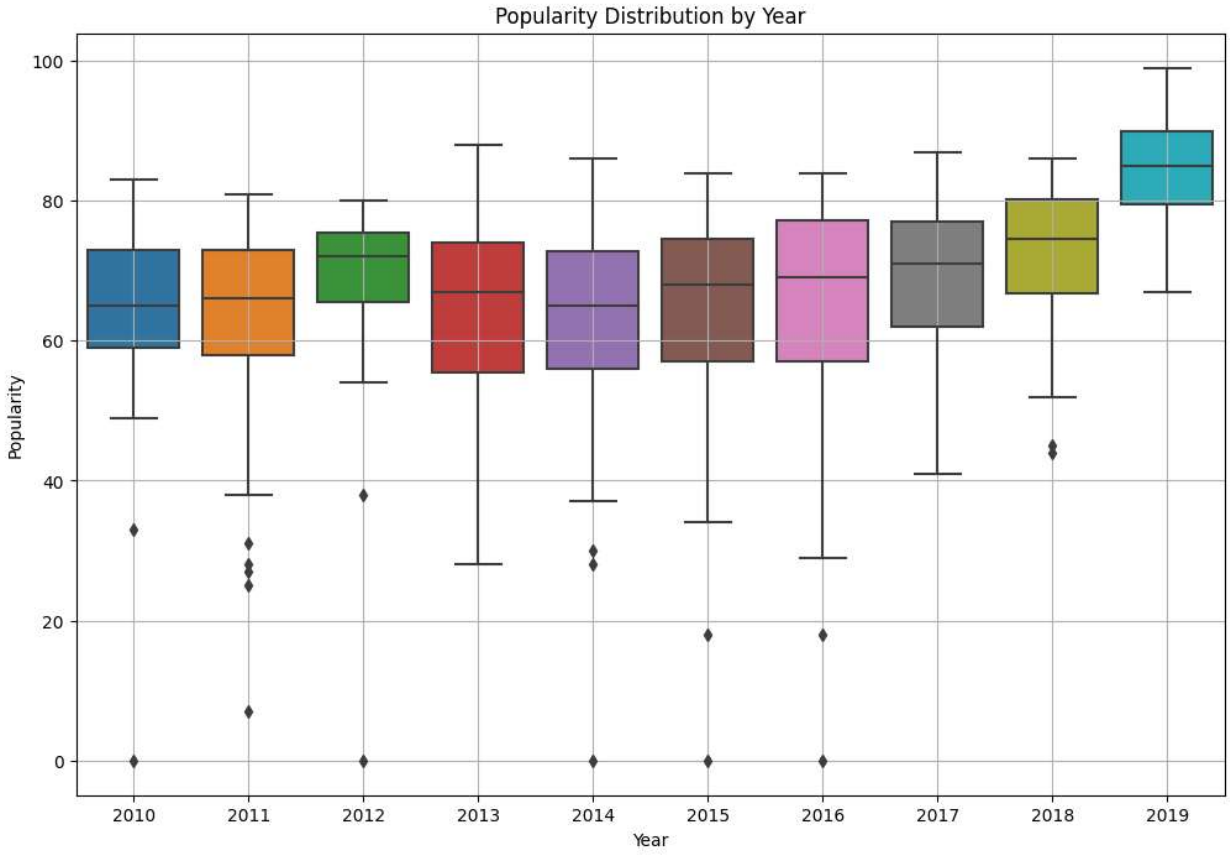


In [371…

```
# @title Checking for the the outliers

popularity_by_year = df.groupby('year')['pop'].mean().reset_index()
```

In [372…

```
#plot the data

plt.figure(figsize=(12, 8))
sns.boxplot(x='year', y='pop', data=df)
plt.title('Popularity Distribution by Year')
plt.xlabel('Year')
plt.ylabel('Popularity')
plt.grid(True)
plt.show()
```



In [373…

```
#statistics of the dataset

popularity_stats_by_year = df.groupby('year')['pop'].describe()
print(popularity_stats_by_year)
df.head()
```

```
      count       mean        std   min    25%   50%    75%   max
year
2010   51.0  64.254902  13.227007   0.0  59.00  65.0  73.00  83.0
2011   53.0  61.867925  16.058831   7.0  58.00  66.0  73.00  81.0
2012   35.0  67.771429  14.528037   0.0  65.50  72.0  75.50  80.0
2013   71.0  63.985915  12.864673  28.0  55.50  67.0  74.00  88.0
2014   58.0  62.706897  15.545077   0.0  56.00  65.0  72.75  86.0
2015   95.0  64.568421  14.352443   0.0  57.00  68.0  74.50  84.0
2016   80.0  64.162500  16.237512   0.0  57.00  69.0  77.25  84.0
2017   65.0  69.015385  10.982219  41.0  62.00  71.0  77.00  87.0
2018   64.0  72.437500   9.870390  44.0  66.75  74.5  80.25  86.0
2019   31.0  84.354839   8.292761  67.0  79.50  85.0  90.00  99.0
```

Out[373]:

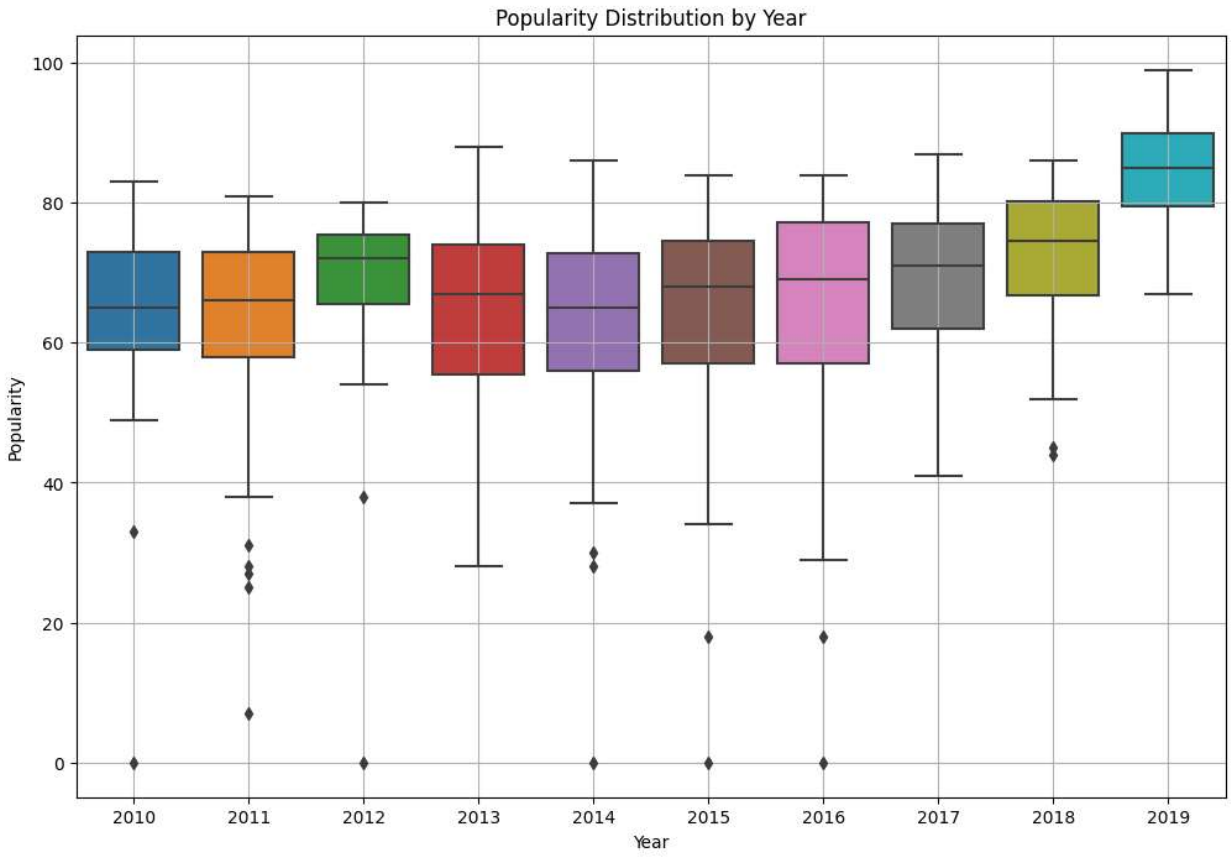
	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	p
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	80	217	19	4	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	64	263	24	23	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	71	200	10	14	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	71	295	0	4	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	43	221	2	4	

In [374…

```
# way to remove the outliers just by changing the range

for i in range(0):
    Q1 = df['pop'].quantile(0.25)
    Q3 = df['pop'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df = df[(df['pop'] >= lower_bound) & (df['pop'] <= upper_bound)]
    df.head()
```

In [375…

```
#if we remove the outliers, the codeblock for again plotting the data without outliers
plt.figure(figsize=(12, 8))
sns.boxplot(x='year', y='pop', data=df)
plt.title('Popularity Distribution by Year')
plt.xlabel('Year')
plt.ylabel('Popularity')
plt.grid(True)
plt.show()
df.head()
```



Out[375]:

	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	p
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	80	217	19	4	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	64	263	24	23	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	71	200	10	14	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	71	295	0	4	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	43	221	2	4	

In [376…

```
#statistics just after removing the outliers

popularity_stats_no_outliers = df.groupby('year')['pop'].describe()
print(popularity_stats_no_outliers)
```

year	count	mean	std	min	25%	50%	75%	max
2010	51.0	64.254902	13.227007	0.0	59.00	65.0	73.00	83.0
2011	53.0	61.867925	16.058831	7.0	58.00	66.0	73.00	81.0
2012	35.0	67.771429	14.528037	0.0	65.50	72.0	75.50	80.0
2013	71.0	63.985915	12.864673	28.0	55.50	67.0	74.00	88.0
2014	58.0	62.706897	15.545077	0.0	56.00	65.0	72.75	86.0
2015	95.0	64.568421	14.352443	0.0	57.00	68.0	74.50	84.0
2016	80.0	64.162500	16.237512	0.0	57.00	69.0	77.25	84.0
2017	65.0	69.015385	10.982219	41.0	62.00	71.0	77.00	87.0
2018	64.0	72.437500	9.870390	44.0	66.75	74.5	80.25	86.0
2019	31.0	84.354839	8.292761	67.0	79.50	85.0	90.00	99.0

In [377…

```
# @title Encoding the string features

from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Assuming df is your DataFrame
vectorizer = TfidfVectorizer()

# Combine text from 'title', 'artist', and 'top genre' into a single column
#combined_text = df['title'] + ' ' + df['artist'] + ' ' + df['top genre']
combined_text = df['top genre']


# Transform the combined text using the vectorizer
text_matrix = vectorizer.fit_transform(combined_text)

# Extract feature names
feature_names = vectorizer.get_feature_names_out()

# Create DataFrames for each text column
#title_matrix = pd.DataFrame(vectorizer.transform(df['title']).toarray(), columns=[f't
#artist_matrix = pd.DataFrame(vectorizer.transform(df['artist']).toarray(), columns=[f
top_genre_matrix = pd.DataFrame(vectorizer.transform(df['top genre']).toarray(), colum

# Concatenate the matrices with the original DataFrame
#df = pd.concat([df, title_matrix, artist_matrix, top_genre_matrix], axis=1)
df = pd.concat([df, top_genre_matrix], axis=1)
#df = pd.concat([df, title_matrix, top_genre_matrix], axis=1)
```

In [378…

```
# @title Classified the target variable

# Calculate the median popularity for each year
median_popularity_by_year = df.groupby('year')['pop'].median()

# Create labels based on median popularity values
labels = ['Miss', 'Hit']

# Assign labels to each year based on median popularity
df['pop_classification'] = df['year'].map(median_popularity_by_year)
df['pop_classification'] = pd.cut(df['pop_classification'], bins=[df['pop_classificati

# Display the count of each label
classification_counts = df['pop_classification'].value_counts()
print(classification_counts)

# Display statistical summary of popularity for each year without outliers
```

```
popularity_stats_no_outliers = df.groupby('year')['pop'].describe()
print(popularity_stats_no_outliers)
```

```
Hit     339
Miss    233
Name: pop_classification, dtype: int64
      count       mean        std   min    25%   50%    75%   max
year
2010   51.0  64.254902  13.227007   0.0  59.00  65.0  73.00  83.0
2011   53.0  61.867925  16.058831   7.0  58.00  66.0  73.00  81.0
2012   35.0  67.771429  14.528037   0.0  65.50  72.0  75.50  80.0
2013   71.0  63.985915  12.864673  28.0  55.50  67.0  74.00  88.0
2014   58.0  62.706897  15.545077   0.0  56.00  65.0  72.75  86.0
2015   95.0  64.568421  14.352443   0.0  57.00  68.0  74.50  84.0
2016   80.0  64.162500  16.237512   0.0  57.00  69.0  77.25  84.0
2017   65.0  69.015385  10.982219  41.0  62.00  71.0  77.00  87.0
2018   64.0  72.437500   9.870390  44.0  66.75  74.5  80.25  86.0
2019   31.0  84.354839   8.292761  67.0  79.50  85.0  90.00  99.0
```

In [379…
```
df.head()
```

Out[379]:

	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	...	top_genre_rap	top_g
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	...	0.0	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	...	0.0	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	...	0.0	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	...	0.0	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	...	0.0	

5 rows × 71 columns

In [380…
```
#plt.scatter(df['Unnamed: 0'],df['pop_classification'])
df.isna().sum()
```

Out[380]:
```
Unnamed: 0              0
title                   0
artist                  0
top genre               0
year                    0
                      ..
top_genre_soul          0
top_genre_trap          0
top_genre_tropical      0
top_genre_wave          0
pop_classification     31
Length: 71, dtype: int64
```

In [381…
```
df = df.dropna()
df.isna().sum()
```

Out[381]:
```
Unnamed: 0            0
title                 0
artist                0
top genre             0
year                  0
                     ..
top_genre_soul        0
top_genre_trap        0
top_genre_tropical    0
top_genre_wave        0
pop_classification    0
Length: 71, dtype: int64
```

In [382…
```
#drop the unwanted columns

df=df.drop('Unnamed: 0',axis=1)
df=df.drop('title',axis=1)
df=df.drop('artist',axis=1)
df=df.drop('top genre',axis=1)
df=df.drop('pop',axis=1)
```

In [383…
```
df.head()
```

Out[383]:

	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	...	top_genre_rap	top_genre_rock	top_g
0	2010	97	89	67	-4	8	80	217	19	4	...	0.0	0.0	
1	2010	87	93	75	-5	52	64	263	24	23	...	0.0	0.0	
2	2010	120	84	76	-3	29	71	200	10	14	...	0.0	0.0	
3	2010	119	92	70	-4	8	71	295	0	4	...	0.0	0.0	
4	2010	109	84	64	-5	9	43	221	2	4	...	0.0	0.0	

5 rows × 66 columns

In [384…
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

In [385…
```
#feature engineering

X = df.drop(['pop_classification'], axis=1)
y = df['pop_classification']
```

In [386…
```
# splitting the training and test datasets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [387…

```
#Normalizing the data

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [388…

```
def HyperPara(c,g):
    # Create and train the SVM classifier
    #svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
    # Example with additional hyperparameter (gamma) for RBF kernel
    svm_classifier = SVC(kernel='rbf', C=c, gamma=g, random_state=42)
    svm_classifier.fit(X_train_scaled, y_train)

    # Make predictions on the test set
    predictions = svm_classifier.predict(X_test_scaled)
    predictions = svm_classifier.predict(X_test_scaled)

    # Evaluate the classifier
    return [predictions,svm_classifier]
```

In [389…

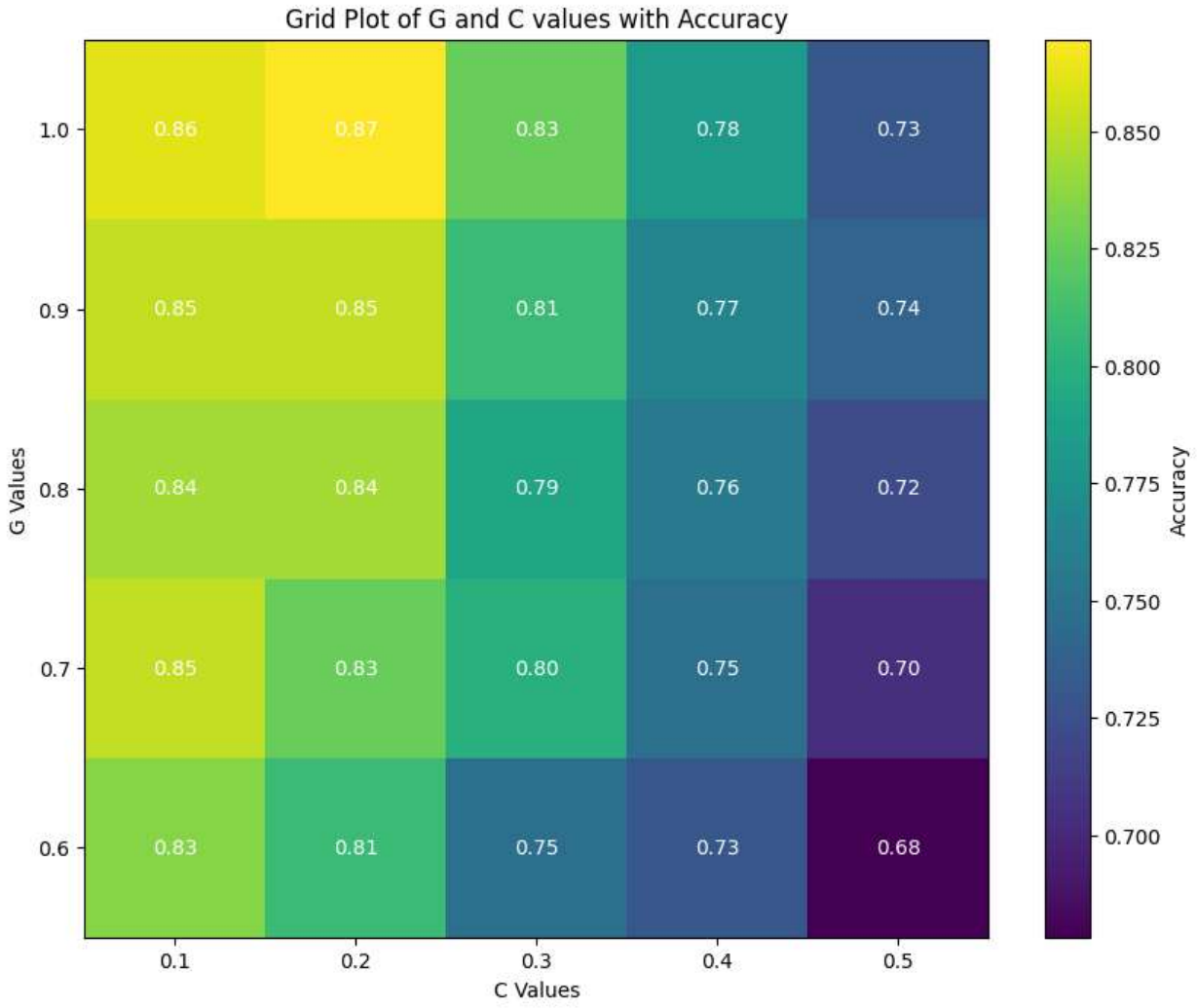
```
#Hyperparameter optimization

global predictions
global svm_classifier
accuracy=0
G=[0.6,0.7,0.8,0.9,1.0]
C=[0.1,0.2,0.3,0.4,0.5]
AC=[]
GM=0
CO=0
for i in G:
    for j in C:
        H = HyperPara(i,j)
        predict =H[0]
        ac= accuracy_score(y_test, predict)
        AC.append(ac)
        if ac>accuracy:
            accuracy=ac
            GM=i
            CO=j
            predictions = predict
            svm_classifier = H[1]
print(AC)
print(GM,CO)
```

```
[0.8347826086956521, 0.808695652173913, 0.7478260869565218, 0.7304347826086957, 0.678
2608695652174, 0.8521739130434782, 0.8260869565217391, 0.8, 0.7478260869565218, 0.704
3478260869566, 0.8434782608695652, 0.8434782608695652, 0.7913043478260869, 0.75652173
91304347, 0.7217391304347827, 0.8521739130434782, 0.8521739130434782, 0.8086956521739
13, 0.7652173913043478, 0.7391304347826086, 0.8608695652173913, 0.8695652173913043,
0.8260869565217391, 0.782608695652174, 0.7304347826086957]
1.0 0.2
```

In [390…

```
# @title accuracy grid for hyperparameters

# Assuming AC is a 1D array of accuracy values, and G, C are 1D arrays of parameter va
AC_reshaped = np.array(AC).reshape(len(G), len(C))

# Creating a grid plot
```

```
plt.figure(figsize=(10, 8))
heatmap = plt.pcolor(AC_reshaped, cmap='viridis')

# Adding color bar
cbar = plt.colorbar(heatmap, label='Accuracy')

# Adding G and C labels
plt.xticks(np.arange(len(C)) + 0.5, C)
plt.yticks(np.arange(len(G)) + 0.5, G)
plt.xlabel('C Values')
plt.ylabel('G Values')

# Adding text annotations in each cell
for i in range(len(G)):
    for j in range(len(C)):
        plt.text(j + 0.5, i + 0.5, f'{AC_reshaped[i, j]:.2f}', ha='center', va='center

plt.title('Grid Plot of G and C values with Accuracy')
plt.show()
```

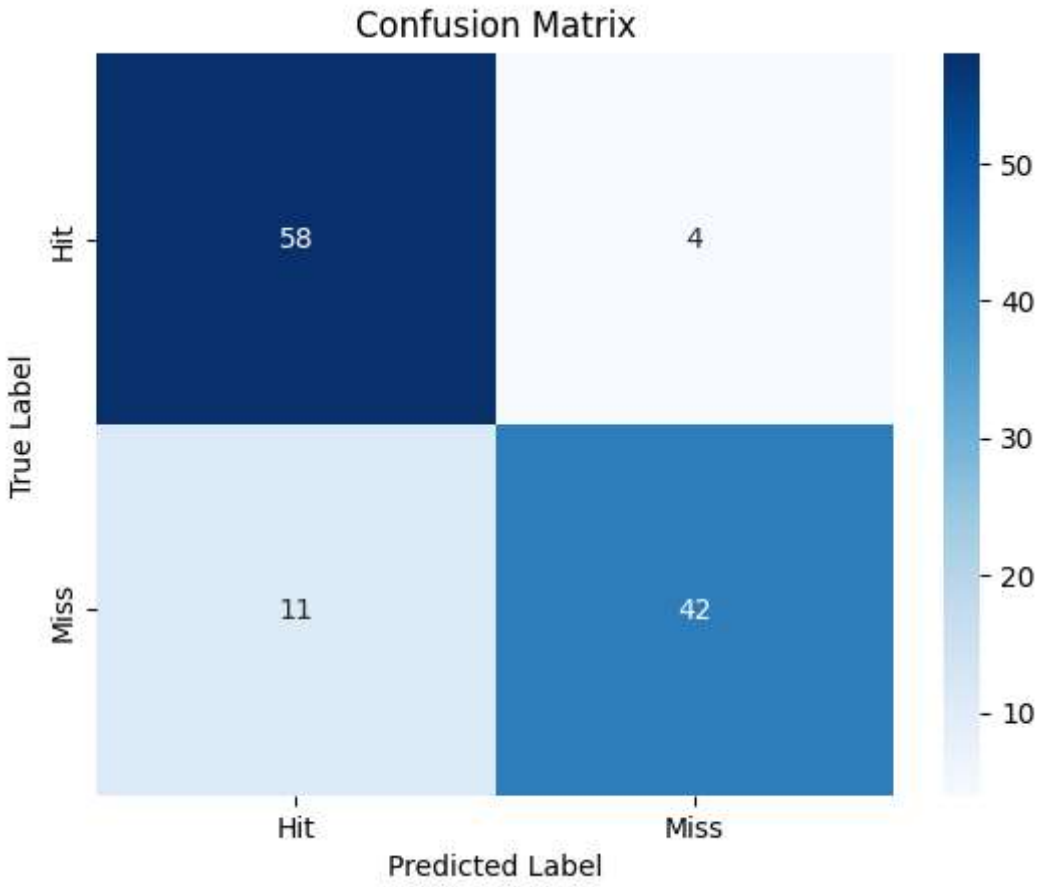


In [391…

```
#visualize the output

conf_matrix = confusion_matrix(y_test, predictions)
classification_rep = classification_report(y_test, predictions)

# Display evaluation metrics
print(C,G)
```

```
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)

# Visualize the confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=svm_classifier
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
[0.1, 0.2, 0.3, 0.4, 0.5] [0.6, 0.7, 0.8, 0.9, 1.0]
Accuracy: 0.8695652173913043
Confusion Matrix:
[[58  4]
 [11 42]]
Classification Report:
              precision    recall  f1-score   support

         Hit       0.84      0.94      0.89        62
        Miss       0.91      0.79      0.85        53

    accuracy                           0.87       115
   macro avg       0.88      0.86      0.87       115
weighted avg       0.87      0.87      0.87       115
```



In [392…

```
#Hyperparameter optimization for training data

def Train_HyperPara(c,g):
```

```
    # Create and train the SVM classifier
    #svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
    # Example with additional hyperparameter (gamma) for RBF kernel
    svm_classifier = SVC(kernel='rbf', C=c, gamma=g, random_state=42)
    svm_classifier.fit(X_train_scaled, y_train)

    # Make predictions on the test set
    predictions = svm_classifier.predict(X_train_scaled)
    predictions = svm_classifier.predict(X_train_scaled)

    # Evaluate the classifier
    return [predictions,svm_classifier]
```

In [393…

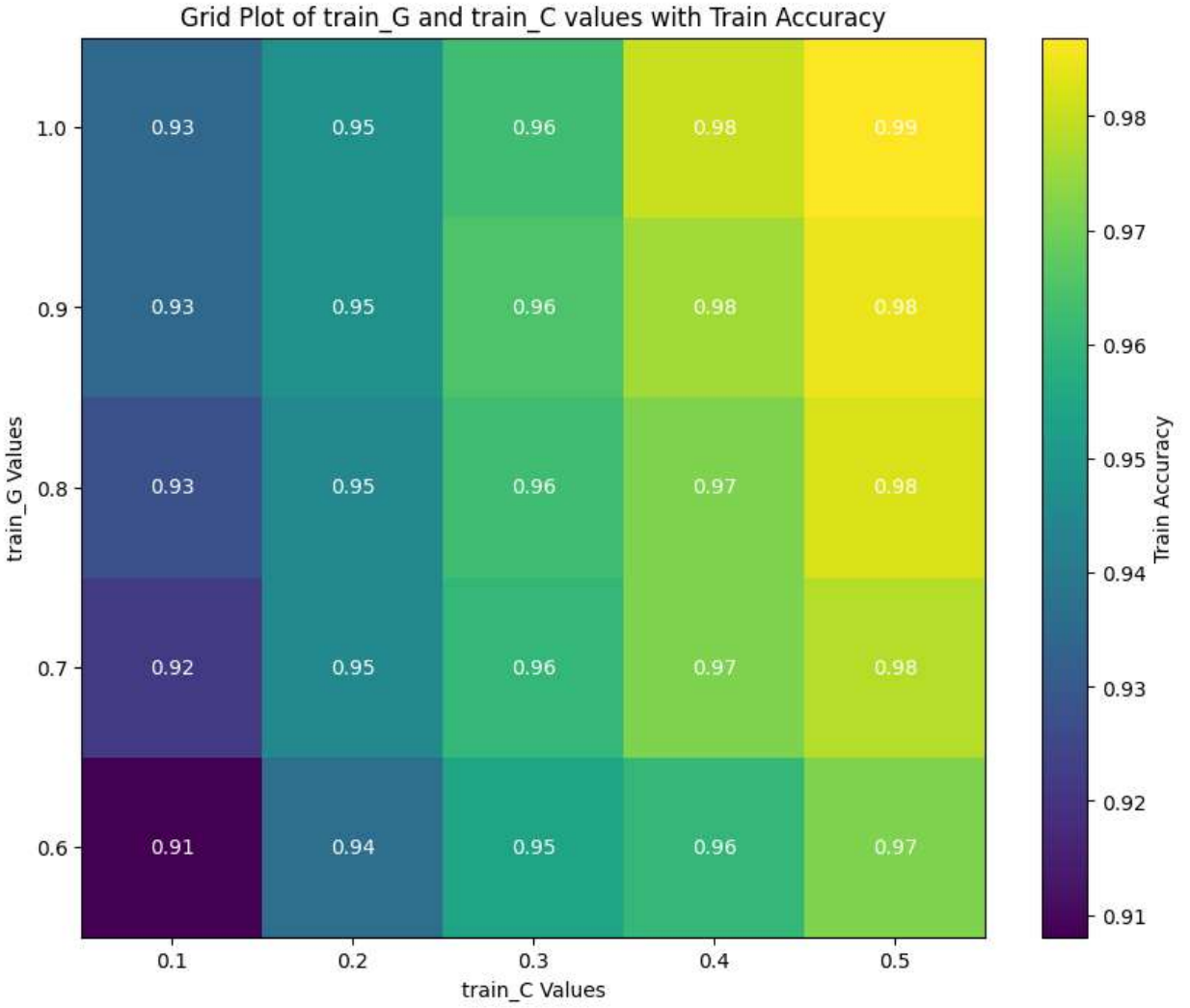
```
global train_predictions
global train_svm_classifier
train_accuracy=0
train_G=[0.6,0.7,0.8,0.9,1.0]
train_C=[0.1,0.2,0.3,0.4,0.5]
train_AC=[]
train_GM=0
train_CO=0
for i in train_G:
    for j in train_C:
        train_H = Train_HyperPara(i,j)
        train_predict =train_H[0]
        train_ac= accuracy_score(y_train, train_predict)
        train_AC.append(train_ac)
        if train_ac>train_accuracy:
            train_accuracy=train_ac
            train_GM=i
            train_CO=j
            train_predictions = train_predict
            train_svm_classifier = train_H[1]
print(train_AC)
print(train_GM,train_CO)
```

```
[0.9080962800875274, 0.936542669584245, 0.9540481400437637, 0.9606126914660832, 0.971
5536105032823, 0.9212253829321663, 0.9452954048140044, 0.9606126914660832, 0.97155361
05032823, 0.9781181619256017, 0.9277899343544858, 0.9452954048140044, 0.9628008752735
23, 0.9715536105032823, 0.9824945295404814, 0.9343544857768052, 0.9474835886214442,
0.9649890590809628, 0.975929978118162, 0.9846827133479212, 0.9343544857768052, 0.9474
835886214442, 0.962800875273523, 0.9803063457330415, 0.986870897155361]
1.0 0.5
```

In [394…

```
# @title accuracy grid for hyperparameters

# Assuming AC is a 1D array of accuracy values, and G, C are 1D arrays of parameter va
train_AC_reshaped = np.array(train_AC).reshape(len(train_G), len(train_C))

# Creating a grid plot
plt.figure(figsize=(10, 8))
heatmap = plt.pcolor(train_AC_reshaped, cmap='viridis')

# Adding color bar
train_cbar = plt.colorbar(heatmap, label='Train Accuracy')

# Adding G and C labels
plt.xticks(np.arange(len(train_C)) + 0.5, C)
plt.yticks(np.arange(len(train_G)) + 0.5, G)
plt.xlabel('train_C Values')
```

```
plt.ylabel('train_G Values')

# Adding text annotations in each cell
for i in range(len(train_G)):
    for j in range(len(train_C)):
        plt.text(j + 0.5, i + 0.5, f'{train_AC_reshaped[i, j]:.2f}', ha='center', va='

plt.title('Grid Plot of train_G and train_C values with Train Accuracy')
plt.show()
```

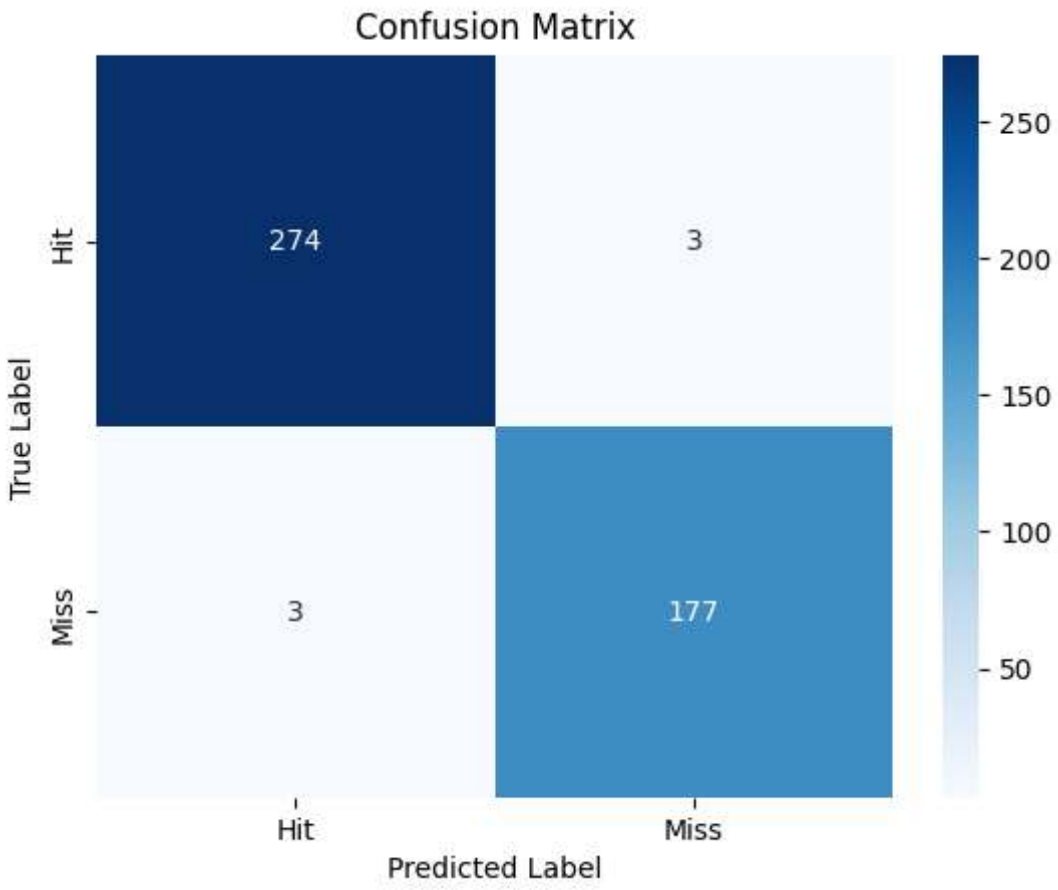


In [395…

```
train_conf_matrix = confusion_matrix(y_train, train_predictions)
train_classification_rep = classification_report(y_train, train_predictions)

# Display evaluation metrics
print(train_C,train_G)
print(f"Train Accuracy: {train_accuracy}")
print("Confusion Matrix:")
print(train_conf_matrix)
print("Classification Report:")
print(train_classification_rep)

# Visualize the confusion matrix
sns.heatmap(train_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=train_sv
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.title('Confusion Matrix')
plt.show()
```

```
[0.1, 0.2, 0.3, 0.4, 0.5] [0.6, 0.7, 0.8, 0.9, 1.0]
Train Accuracy: 0.986870897155361
Confusion Matrix:
[[274   3]
 [  3 177]]
Classification Report:
              precision    recall  f1-score   support

         Hit       0.99      0.99      0.99       277
        Miss       0.98      0.98      0.98       180

    accuracy                           0.99       457
   macro avg       0.99      0.99      0.99       457
weighted avg       0.99      0.99      0.99       457
```



In [316…
```
# @title Import Libraries

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

In [317…
```
# @title Load data
df = pd.read_csv('top10s.csv',encoding='ISO-8859-1')
df.head()
```

Out[317]:

	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	p
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	80	217	19	4	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	64	263	24	23	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	71	200	10	14	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	71	295	0	4	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	43	221	2	4	

In [318…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 603 entries, 0 to 602
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  603 non-null    int64
 1   title       603 non-null    object
 2   artist      603 non-null    object
 3   top genre   603 non-null    object
 4   year        603 non-null    int64
 5   bpm         603 non-null    int64
 6   nrgy        603 non-null    int64
 7   dnce        603 non-null    int64
 8   dB          603 non-null    int64
 9   live        603 non-null    int64
 10  val         603 non-null    int64
 11  dur         603 non-null    int64
 12  acous       603 non-null    int64
 13  spch        603 non-null    int64
 14  pop         603 non-null    int64
dtypes: int64(12), object(3)
memory usage: 70.8+ KB
```

In [319…
```
df.columns
```

Out[319]:
```
Index(['Unnamed: 0', 'title', 'artist', 'top genre', 'year', 'bpm', 'nrgy',
       'dnce', 'dB', 'live', 'val', 'dur', 'acous', 'spch', 'pop'],
      dtype='object')
```

In [320…
```
df.isna().sum()
```

Out[320]:
```
Unnamed: 0    0
title         0
artist        0
top genre     0
year          0
bpm           0
nrgy          0
dnce          0
dB            0
live          0
val           0
dur           0
acous         0
spch          0
pop           0
dtype: int64
```

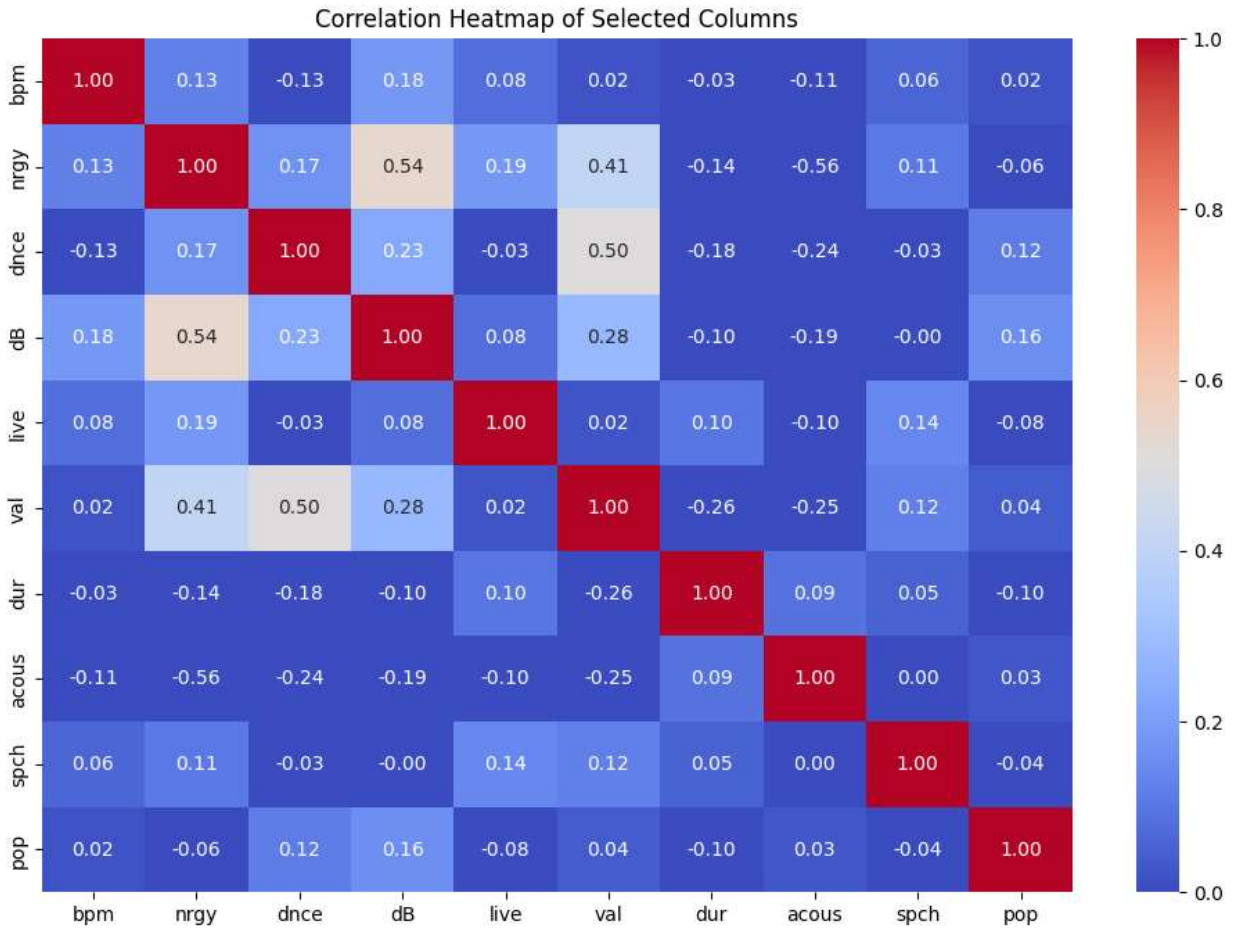
In [321…
```
df.duplicated().sum()
```

Out[321]:
```
0
```

In [322…
```
print(df['title'].nunique())
print(df['artist'].nunique())
print(df['top genre'].nunique())
```

Out[322]:
```
50
```

In [323…
```
# @title checking the correlation
selected_columns = ['bpm', 'nrgy', 'dnce', 'dB', 'live', 'val', 'dur', 'acous', 'spch'

# Extracting the selected columns
df_selected = df[selected_columns]

# Calculating the correlation matrix
correlation_matrix = df_selected.corr()

# Creating a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=0, vmax=1
plt.title('Correlation Heatmap of Selected Columns')
plt.show()
```

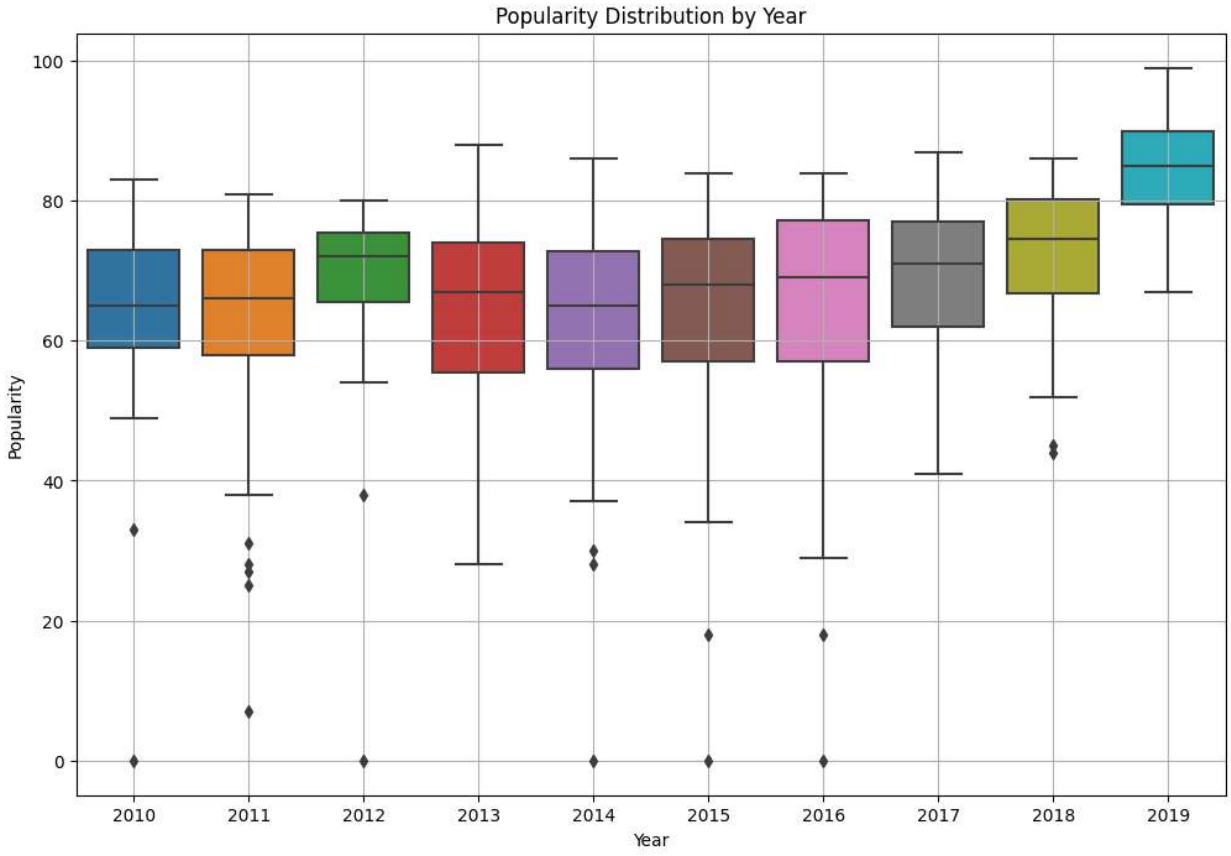


In [324…

```
# @title Checking for the the outliers

popularity_by_year = df.groupby('year')['pop'].mean().reset_index()
```

In [325…

```
#plot the data

plt.figure(figsize=(12, 8))
sns.boxplot(x='year', y='pop', data=df)
plt.title('Popularity Distribution by Year')
plt.xlabel('Year')
plt.ylabel('Popularity')
plt.grid(True)
plt.show()
df.head()
```



Out[325]:

	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	p
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	80	217	19	4	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	64	263	24	23	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	71	200	10	14	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	71	295	0	4	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	43	221	2	4	

In [326…

```
#statistics of the dataset

popularity_stats_by_year = df.groupby('year')['pop'].describe()
print(popularity_stats_by_year)
df.head()
```

year	count	mean	std	min	25%	50%	75%	max
2010	51.0	64.254902	13.227007	0.0	59.00	65.0	73.00	83.0
2011	53.0	61.867925	16.058831	7.0	58.00	66.0	73.00	81.0
2012	35.0	67.771429	14.528037	0.0	65.50	72.0	75.50	80.0
2013	71.0	63.985915	12.864673	28.0	55.50	67.0	74.00	88.0
2014	58.0	62.706897	15.545077	0.0	56.00	65.0	72.75	86.0
2015	95.0	64.568421	14.352443	0.0	57.00	68.0	74.50	84.0
2016	80.0	64.162500	16.237512	0.0	57.00	69.0	77.25	84.0
2017	65.0	69.015385	10.982219	41.0	62.00	71.0	77.00	87.0
2018	64.0	72.437500	9.870390	44.0	66.75	74.5	80.25	86.0
2019	31.0	84.354839	8.292761	67.0	79.50	85.0	90.00	99.0

Out[326]:

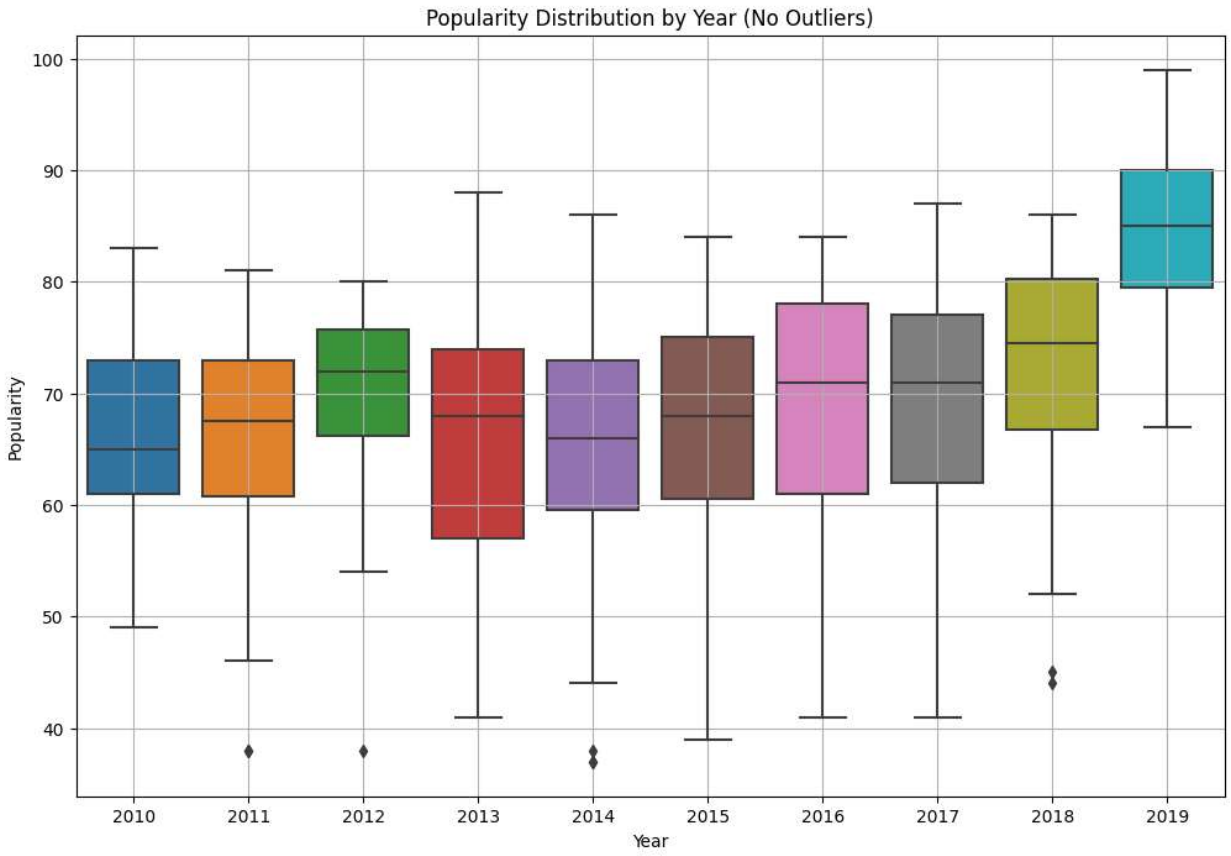
	Unnamed: 0	title	artist	top genre	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	p
0	1	Hey, Soul Sister	Train	neo mellow	2010	97	89	67	-4	8	80	217	19	4	
1	2	Love The Way You Lie	Eminem	detroit hip hop	2010	87	93	75	-5	52	64	263	24	23	
2	3	TiK ToK	Kesha	dance pop	2010	120	84	76	-3	29	71	200	10	14	
3	4	Bad Romance	Lady Gaga	dance pop	2010	119	92	70	-4	8	71	295	0	4	
4	5	Just the Way You Are	Bruno Mars	pop	2010	109	84	64	-5	9	43	221	2	4	

In [327…

```
# way to remove the outliers just by changing the range

for i in range(2):
    Q1 = df['pop'].quantile(0.25)
    Q3 = df['pop'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df = df[(df['pop'] >= lower_bound) & (df['pop'] <= upper_bound)]
    df.head()
```

In [328…

```
#if we remove the outliers, the codeblock for again plotting the data without outliers

plt.figure(figsize=(12, 8))
sns.boxplot(x='year', y='pop', data=df)
plt.title('Popularity Distribution by Year (No Outliers)')
plt.xlabel('Year')
plt.ylabel('Popularity')
plt.grid(True)
plt.show()
```



In [329…

```
#statistics just after removing the outliers

popularity_stats_no_outliers = df.groupby('year')['pop'].describe()
print(popularity_stats_no_outliers)
```

```
      count       mean        std   min    25%   50%    75%   max
year
2010   49.0  66.204082   8.485231  49.0  61.00  65.0  73.00  83.0
2011   48.0  65.854167  10.274922  38.0  60.75  67.5  73.00  81.0
2012   34.0  69.764706   8.613094  38.0  66.25  72.0  75.75  80.0
2013   68.0  65.352941  11.289626  41.0  57.00  68.0  74.00  88.0
2014   55.0  65.072727  11.594786  37.0  59.50  66.0  73.00  86.0
2015   91.0  66.439560  11.033493  39.0  60.50  68.0  75.00  84.0
2016   73.0  67.794521  11.046567  41.0  61.00  71.0  78.00  84.0
2017   65.0  69.015385  10.982219  41.0  62.00  71.0  77.00  87.0
2018   64.0  72.437500   9.870390  44.0  66.75  74.5  80.25  86.0
2019   31.0  84.354839   8.292761  67.0  79.50  85.0  90.00  99.0
```

In [330…

```
# @title Encoding the string features

from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Assuming df is your DataFrame
vectorizer = TfidfVectorizer()

# Combine text from 'title', 'artist', and 'top genre' into a single column
#combined_text = df['title'] + ' ' + df['artist'] + ' ' + df['top genre']
combined_text = df['top genre']


# Transform the combined text using the vectorizer
```

```
text_matrix = vectorizer.fit_transform(combined_text)

# Extract feature names
feature_names = vectorizer.get_feature_names_out()

# Create DataFrames for each text column
#title_matrix = pd.DataFrame(vectorizer.transform(df['title']).toarray(), columns=[f't
#artist_matrix = pd.DataFrame(vectorizer.transform(df['artist']).toarray(), columns=[f
top_genre_matrix = pd.DataFrame(vectorizer.transform(df['top genre']).toarray(), colum

# Concatenate the matrices with the original DataFrame
#df = pd.concat([df, title_matrix, artist_matrix, top_genre_matrix], axis=1)
df = pd.concat([df, top_genre_matrix], axis=1)
#df = pd.concat([df, title_matrix, top_genre_matrix], axis=1)
```

In [331…

```
#drop the unwanted columns

df_old=df
df=df.drop('title',axis=1)
df=df.drop('artist',axis=1)
df=df.drop('top genre',axis=1)
df.head()
```

Out[331]:

	Unnamed: 0	year	bpm	nrgy	dnce	dB	live	val	dur	acous	...	top_genre_pop	top_genre
0	1.0	2010.0	97.0	89.0	67.0	-4.0	8.0	80.0	217.0	19.0	...	0.000000	
1	2.0	2010.0	87.0	93.0	75.0	-5.0	52.0	64.0	263.0	24.0	...	0.000000	
2	3.0	2010.0	120.0	84.0	76.0	-3.0	29.0	71.0	200.0	10.0	...	0.613664	
3	4.0	2010.0	119.0	92.0	70.0	-4.0	8.0	71.0	295.0	0.0	...	0.613664	
4	5.0	2010.0	109.0	84.0	64.0	-5.0	9.0	43.0	221.0	2.0	...	1.000000	

5 rows × 66 columns

In [332…

```
# @title Classified the target variable


# Calculate the median popularity for each year
median_popularity_by_year = df.groupby('year')['pop'].median()

# Create labels based on median popularity values
labels = ['Miss', 'Hit']

# Assign labels to each year based on median popularity
df['pop_classification'] = df['year'].map(median_popularity_by_year)
df['pop_classification'] = pd.cut(df['pop_classification'], bins=[df['pop_classificati

# Display the count of each label
classification_counts = df['pop_classification'].value_counts()
print(classification_counts)

# Display statistical summary of popularity for each year without outliers
```

```
popularity_stats_no_outliers = df.groupby('year')['pop'].describe()
print(popularity_stats_no_outliers)
```

```
Hit     395
Miss    152
Name: pop_classification, dtype: int64
        count       mean        std   min    25%   50%    75%   max
year
2010.0   49.0  66.204082   8.485231  49.0  61.00  65.0  73.00  83.0
2011.0   48.0  65.854167  10.274922  38.0  60.75  67.5  73.00  81.0
2012.0   34.0  69.764706   8.613094  38.0  66.25  72.0  75.75  80.0
2013.0   68.0  65.352941  11.289626  41.0  57.00  68.0  74.00  88.0
2014.0   55.0  65.072727  11.594786  37.0  59.50  66.0  73.00  86.0
2015.0   91.0  66.439560  11.033493  39.0  60.50  68.0  75.00  84.0
2016.0   73.0  67.794521  11.046567  41.0  61.00  71.0  78.00  84.0
2017.0   65.0  69.015385  10.982219  41.0  62.00  71.0  77.00  87.0
2018.0   64.0  72.437500   9.870390  44.0  66.75  74.5  80.25  86.0
2019.0   31.0  84.354839   8.292761  67.0  79.50  85.0  90.00  99.0
```

In [333…
```
df.isna().sum()
```

Out[333]:
```
Unnamed: 0            25
year                  25
bpm                   25
nrgy                  25
dnce                  25
                      ..
top_genre_soul        25
top_genre_trap        25
top_genre_tropical    25
top_genre_wave        25
pop_classification    56
Length: 67, dtype: int64
```

In [334…
```
df = df.dropna()
df.isna().sum()
```

Out[334]:
```
Unnamed: 0            0
year                  0
bpm                   0
nrgy                  0
dnce                  0
                     ..
top_genre_soul        0
top_genre_trap        0
top_genre_tropical    0
top_genre_wave        0
pop_classification    0
Length: 67, dtype: int64
```

In [335…
```
#dB values are negative so, let shift them
min_dB = df['dB'].min()
df['dB'] = df['dB'] + abs(min_dB)
min_dB
```

```
<ipython-input-335-1f5d4bd188b0>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df['dB'] = df['dB'] + abs(min_dB)
```

Out[335]:
```
-15.0
```

In [336…
```
#feature engineering

X = df.drop(['pop', 'pop_classification'], axis=1)
y = df['pop_classification']
```

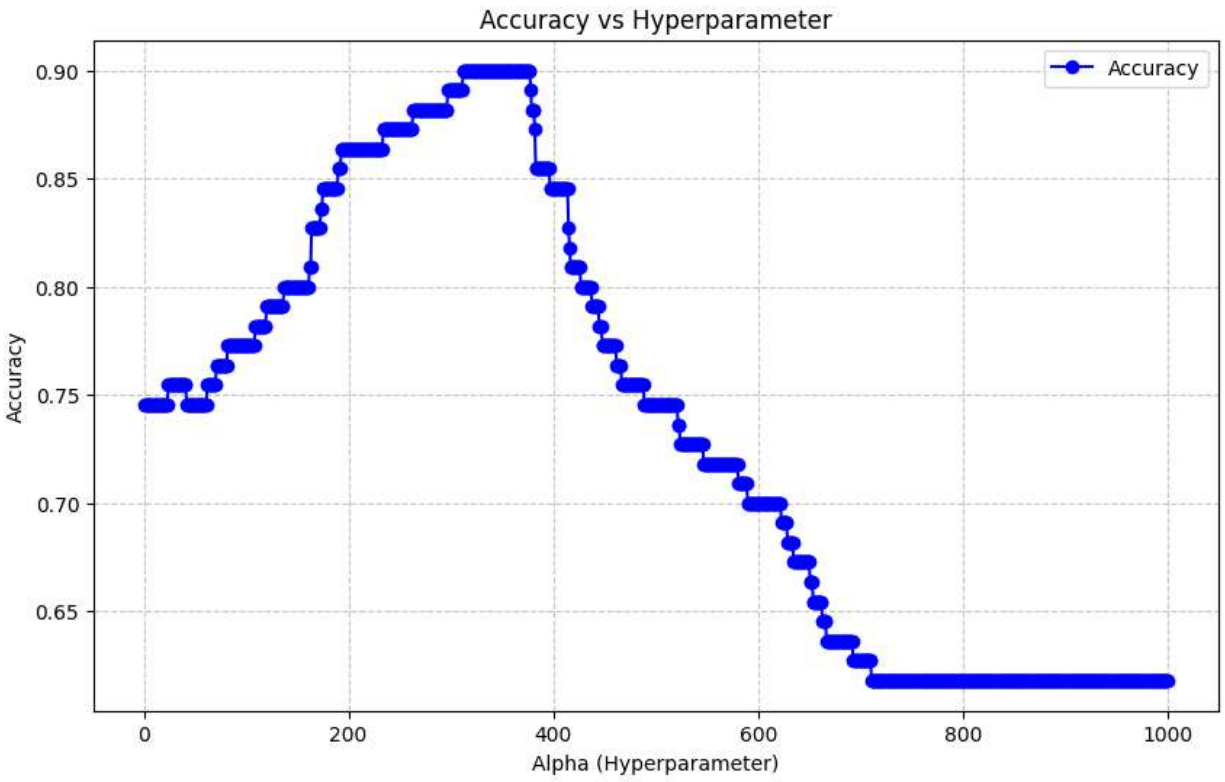
In [337…
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# splitting the training and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Create a Naive Bayes classifier (MultinomialNB is commonly used for text data)
global predictions
global accuracy
global naive_bayes_classifier

accuracy=0
def Hyper(Alpha):
  tmp_naive_bayes_classifier = MultinomialNB(alpha=Alpha)

  # Train the classifier
  tmp_naive_bayes_classifier.fit(X_train, y_train)

  # Make predictions on the test set
  tmp_predictions = tmp_naive_bayes_classifier.predict(X_test)

  # Evaluate the classifier
  tmp_accuracy = accuracy_score(y_test, tmp_predictions)
  return [tmp_accuracy,tmp_predictions,tmp_naive_bayes_classifier]
```

In [338…
```
#Hyperparameter optimization

A=[]
AC=[]
for a in range(1,1001):
  H = Hyper(a)
  ac = H[0]
  A.append(a)
  AC.append(ac)
  if accuracy<ac:
    accuracy = ac
    predictions = H[1]
    naive_bayes_classifier = H[2]
    print(a,':',accuracy)
```

```
1 : 0.7454545454545455
24 : 0.7545454545454545
71 : 0.7636363636363637
82 : 0.7727272727272727
109 : 0.7818181818181819
120 : 0.7909090909090909
137 : 0.8
162 : 0.8090909090909091
164 : 0.8272727272727273
173 : 0.8363636363636363
175 : 0.8454545454545455
190 : 0.8545454545454545
193 : 0.8636363636363636
234 : 0.8727272727272727
263 : 0.8818181818181818
297 : 0.8909090909090909
312 : 0.9
```

In [339…

```python
# accuracy plot for hyperparameter value

plt.figure(figsize=(10, 6))
plt.plot(A, AC, marker='o', linestyle='-', color='b', label='Accuracy')
plt.title('Accuracy vs Hyperparameter')
plt.xlabel('Alpha (Hyperparameter)')
plt.ylabel('Accuracy')
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()
```

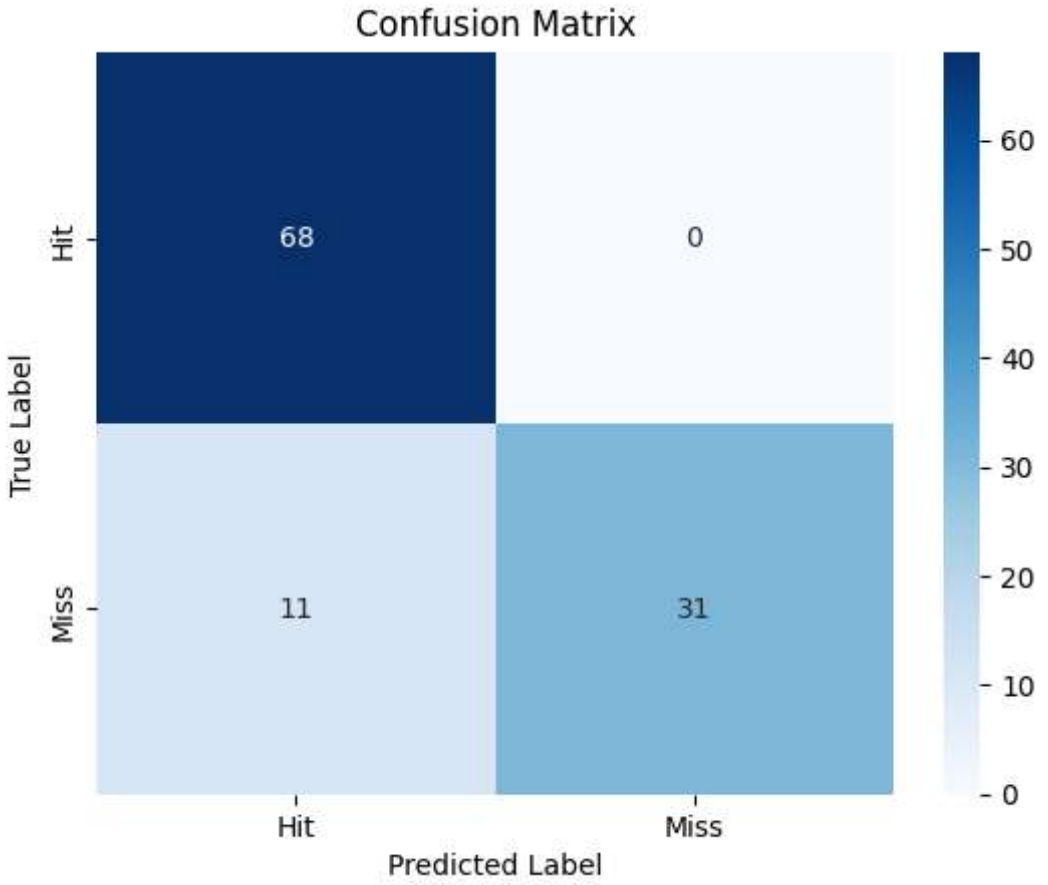


In [340…

```python
#visualize the output

conf_matrix = confusion_matrix(y_test, predictions)
classification_rep = classification_report(y_test, predictions)

print(f"Accuracy: {accuracy}")
```

```
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=naive_bayes_cl
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.9
Confusion Matrix:
[[68  0]
 [11 31]]
Classification Report:
              precision    recall  f1-score   support

         Hit       0.86      1.00      0.93        68
        Miss       1.00      0.74      0.85        42

    accuracy                           0.90       110
   macro avg       0.93      0.87      0.89       110
weighted avg       0.91      0.90      0.90       110
```



In [341…

```
# @title check the accuracy for training dataset
global train_predictions
global train_accuracy
global train_naive_bayes_classifier

train_accuracy=0
def train_Hyper(Alpha):
```

```
  train_tmp_naive_bayes_classifier = MultinomialNB(alpha=Alpha)

  # Train the classifier
  train_tmp_naive_bayes_classifier.fit(X_train, y_train)

  # Make predictions on the test set
  train_tmp_predictions = train_tmp_naive_bayes_classifier.predict(X_train)

  # Evaluate the classifier
  train_tmp_accuracy = accuracy_score(y_train, train_tmp_predictions)
  return [train_tmp_accuracy,train_tmp_predictions,train_tmp_naive_bayes_classifier]
```

In [342…

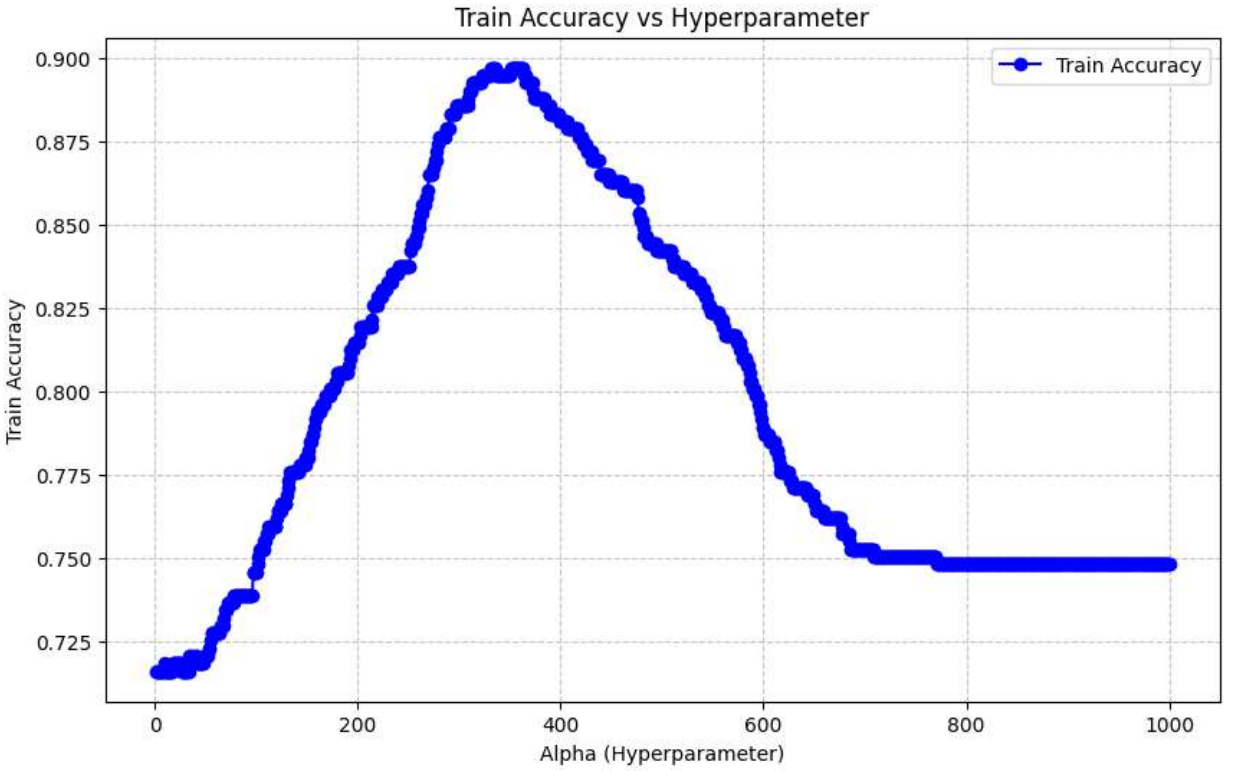
```
train_A=[]
train_AC=[]
for a in range(1,1001):
  train_H = train_Hyper(a)
  train_ac = train_H[0]
  train_A.append(a)
  train_AC.append(train_ac)
  if train_accuracy<train_ac:
    train_accuracy = train_ac
    train_predictions = train_H[1]
    train_naive_bayes_classifier = train_H[2]
    print(a,':',train_accuracy)
```

1 : 0.7162471395881007
9 : 0.7185354691075515
34 : 0.7208237986270023
53 : 0.7231121281464531
55 : 0.7254004576659039
56 : 0.7276887871853547
65 : 0.7299771167048055
68 : 0.7322654462242563
69 : 0.7345537757437071
72 : 0.7368421052631579
78 : 0.7391304347826086
97 : 0.7459954233409611
101 : 0.7482837528604119
102 : 0.7505720823798627
103 : 0.7528604118993135
108 : 0.7551487414187643
110 : 0.7574370709382151
112 : 0.7597254004576659
120 : 0.7620137299771167
121 : 0.7643020594965675
125 : 0.7665903890160183
130 : 0.7688787185354691
131 : 0.7711670480549199
132 : 0.7734553775743707
133 : 0.7757437070938215
143 : 0.7780320366132724
149 : 0.7803203661327232
152 : 0.782608695652174
153 : 0.7848970251716247
155 : 0.7871853546910755
157 : 0.7894736842105263
158 : 0.7917620137299771
160 : 0.7940503432494279
164 : 0.7963386727688787
168 : 0.7986270022883295
173 : 0.8009153318077803
178 : 0.8032036613272311
180 : 0.8054919908466819
191 : 0.8077803203661327
192 : 0.8100686498855835
193 : 0.8123569794050344
196 : 0.8146453089244852
202 : 0.816933638443936
203 : 0.8192219679633868
214 : 0.8215102974828375
215 : 0.8260869565217391
220 : 0.8283752860411899
224 : 0.8306636155606407
229 : 0.8329519450800915
234 : 0.8352402745995423
240 : 0.8375286041189931
252 : 0.8421052631578947
253 : 0.8443935926773455
257 : 0.8466819221967964
259 : 0.8489702517162472
261 : 0.851258581235698
262 : 0.8535469107551488
264 : 0.8558352402745996
267 : 0.8581235697940504
269 : 0.8604118993135011

```
270 : 0.8649885583524027
274 : 0.8672768878718535
276 : 0.8695652173913043
278 : 0.8718535469107551
279 : 0.8741418764302059
280 : 0.8764302059496567
287 : 0.8787185354691075
291 : 0.8832951945080092
297 : 0.88558352402746
309 : 0.8878718535469108
310 : 0.8901601830663616
313 : 0.8924485125858124
323 : 0.8947368421052632
332 : 0.897025171624714
```

In [343…

```
plt.figure(figsize=(10, 6))
plt.plot(train_A, train_AC, marker='o', linestyle='-', color='b', label='Train Accurac
plt.title('Train Accuracy vs Hyperparameter')
plt.xlabel('Alpha (Hyperparameter)')
plt.ylabel('Train Accuracy')
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()
```



In [344…

```
train_conf_matrix = confusion_matrix(y_train, train_predictions)
train_classification_rep = classification_report(y_train, train_predictions)

print(f"Train Accuracy: {train_accuracy}")
print("Confusion Matrix:")
print(train_conf_matrix)
print("Classification Report:")
print(train_classification_rep)
sns.heatmap(train_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=train_na
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.title('Confusion Matrix')
plt.show()
```

```
Train Accuracy: 0.897025171624714
Confusion Matrix:
[[326   1]
 [ 44  66]]
Classification Report:
              precision    recall  f1-score   support

         Hit       0.88      1.00      0.94       327
        Miss       0.99      0.60      0.75       110

    accuracy                           0.90       437
   macro avg       0.93      0.80      0.84       437
weighted avg       0.91      0.90      0.89       437
```