Music taste prediction: Spotify dataset

In [1]:
```
# importing all required module for development

# Librbaries for dataframe and numerical operations
import numpy as np
import pandas as pd

# Librabries for visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Analysis of Dataset

First of all, will read dataset with the help of pandas.

In [2]:
```
# Reading the data from the CSV file
spotify_data = pd.read_csv('Spotify_data.CSV')
```

In [3]:
```
# Reteriving first few data
spotify_data.head(7)
```

Out[3]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness
0	0.803	0.6240	7	-6.764	0	0.0477	0.4510	0.000734
1	0.762	0.7030	10	-7.951	0	0.3060	0.2060	0.000000
2	0.261	0.0149	1	-27.528	1	0.0419	0.9920	0.897000
3	0.722	0.7360	3	-6.994	0	0.0585	0.4310	0.000001
4	0.787	0.5720	1	-7.516	1	0.2220	0.1450	0.000000
5	0.778	0.6320	8	-6.415	1	0.1250	0.0404	0.000000
6	0.666	0.5890	0	-8.405	0	0.3240	0.5550	0.000000

Let's check for datatype of each column.

In [4]:
```
# Checking info regarding loaded dataset
spotify_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   danceability      195 non-null    float64
 1   energy            195 non-null    float64
 2   key               195 non-null    int64  
 3   loudness          195 non-null    float64
 4   mode              195 non-null    int64  
 5   speechiness       195 non-null    float64
 6   acousticness      195 non-null    float64
 7   instrumentalness  195 non-null    float64
 8   liveness          195 non-null    float64
 9   valence           195 non-null    float64
 10  tempo             195 non-null    float64
 11  duration_ms       195 non-null    int64  
 12  time_signature    195 non-null    int64  
 13  liked             195 non-null    int64  
dtypes: float64(9), int64(5)
memory usage: 21.5 KB
```

From above output we can see that all of the variables present are numerical.

Now, will check for any missing value present in data.

In [5]:
```
# Checking missing values
spotify_data.isna().sum()
```

Out[5]:
```
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_ms         0
time_signature      0
liked               0
dtype: int64
```

By looking at above values we can say their is no missing value found in current dataset. Will print the overall statistics of columns present in dataset before that will check if the dataset contain any duplicates.

In [6]:
```
# Making sum of all duplicates rows found
sum(spotify_data.duplicated())
```

Out[6]:
```
0
```

From above infromation count it can be seen that there are no duplicate data found. Creating new column named as duration_mins by converting existing column duration_ms value milliseconds to minutes and droping of the column duration_ms.

In [7]:
```
# Converting milliseconds to minutes
spotify_data["duration_mins"] = spotify_data["duration_ms"]/60000
spotify_data.drop(columns="duration_ms", inplace=True)
```

In [8]:
```
# reteriving description of dataset
spotify_data.describe()
```

Out[8]:

	danceability	energy	key	loudness	mode	speechiness	acoustic
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.00
mean	0.636656	0.638431	5.497436	-9.481631	0.538462	0.148957	0.31
std	0.216614	0.260096	3.415209	6.525086	0.499802	0.120414	0.32
min	0.130000	0.002400	0.000000	-42.261000	0.000000	0.027800	0.00
25%	0.462500	0.533500	2.000000	-9.962000	0.000000	0.056800	0.04
50%	0.705000	0.659000	6.000000	-7.766000	1.000000	0.096200	0.21
75%	0.799000	0.837500	8.000000	-5.829000	1.000000	0.230500	0.50
max	0.946000	0.996000	11.000000	-2.336000	1.000000	0.540000	0.99

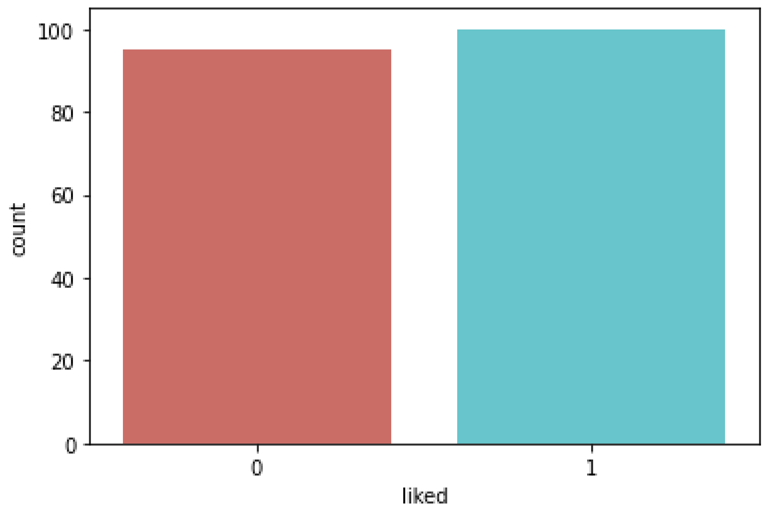
By looking at above statistic values of columns we can inferred that

- Column 'time_signature' conatin same values for first, second and third quartile.
- For column 'liked' it seems to have mean nearly equal to 0.5 which is nothing but the even distribution.
- 'loudness', 'tempo', 'time_signature', and 'duration_ms' columns contain scale of different values.

In [9]:
```
# getting count of particular column values
spotify_data['liked'].value_counts()
```

Out[9]:
```
1    100
0     95
Name: liked, dtype: int64
```

In [10]:
```
# Plotting the graph for values in liked column
sns.countplot(x='liked', data=spotify_data, palette='hls')
plt.show()
plt.savefig('count_plot')
```




```
<Figure size 432x288 with 0 Axes>
```

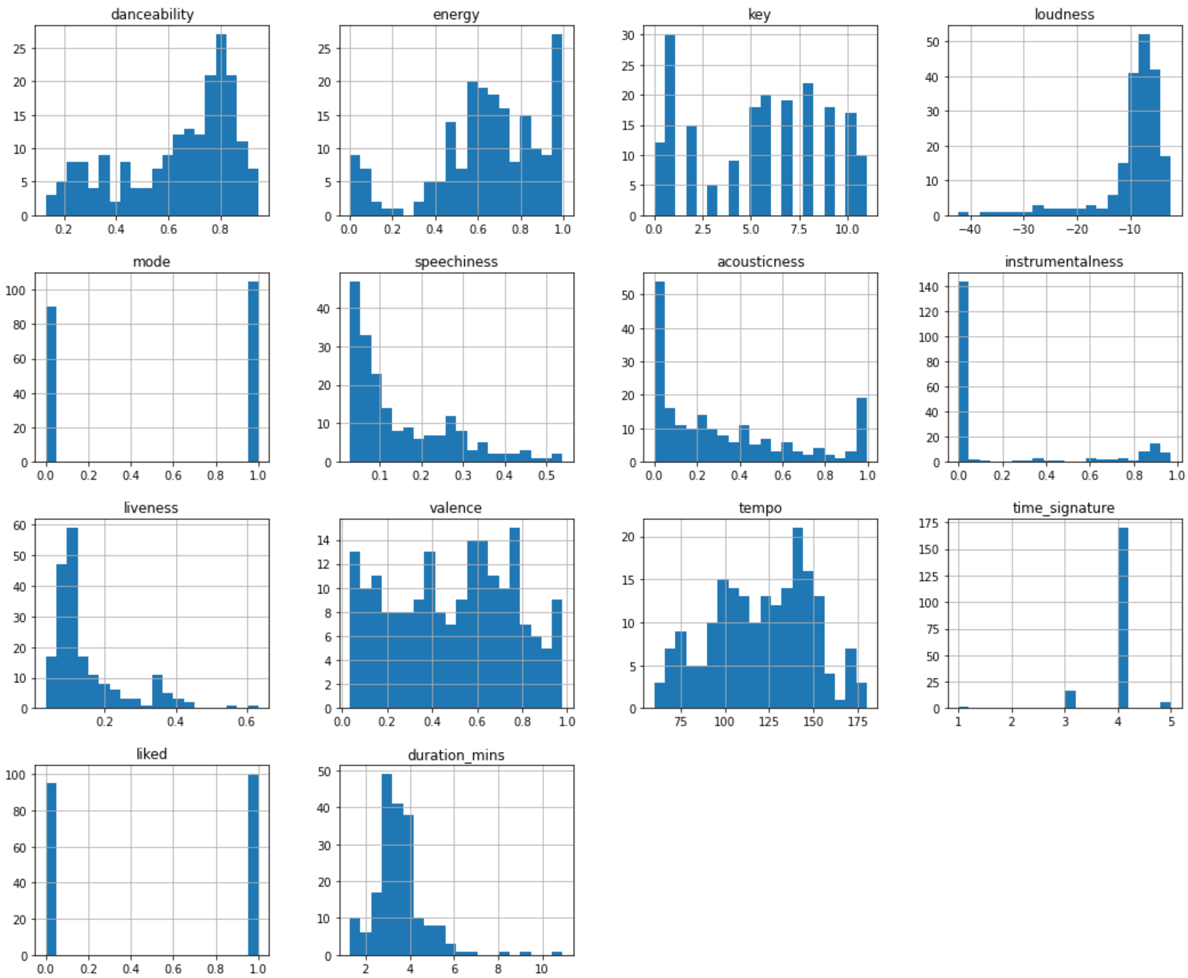
In [11]:
```
# with the grouby clause on liked column calculating mean for all paramaters
spotify_data.groupby('liked').mean()
```

Out[11]:

liked	danceability	energy	key	loudness	mode	speechiness	acousticness	i
0	0.510432	0.591538	5.652632	-12.224537	0.526316	0.076069	0.377977	
1	0.756570	0.682980	5.350000	-6.875870	0.550000	0.218201	0.263154	

In [12]:

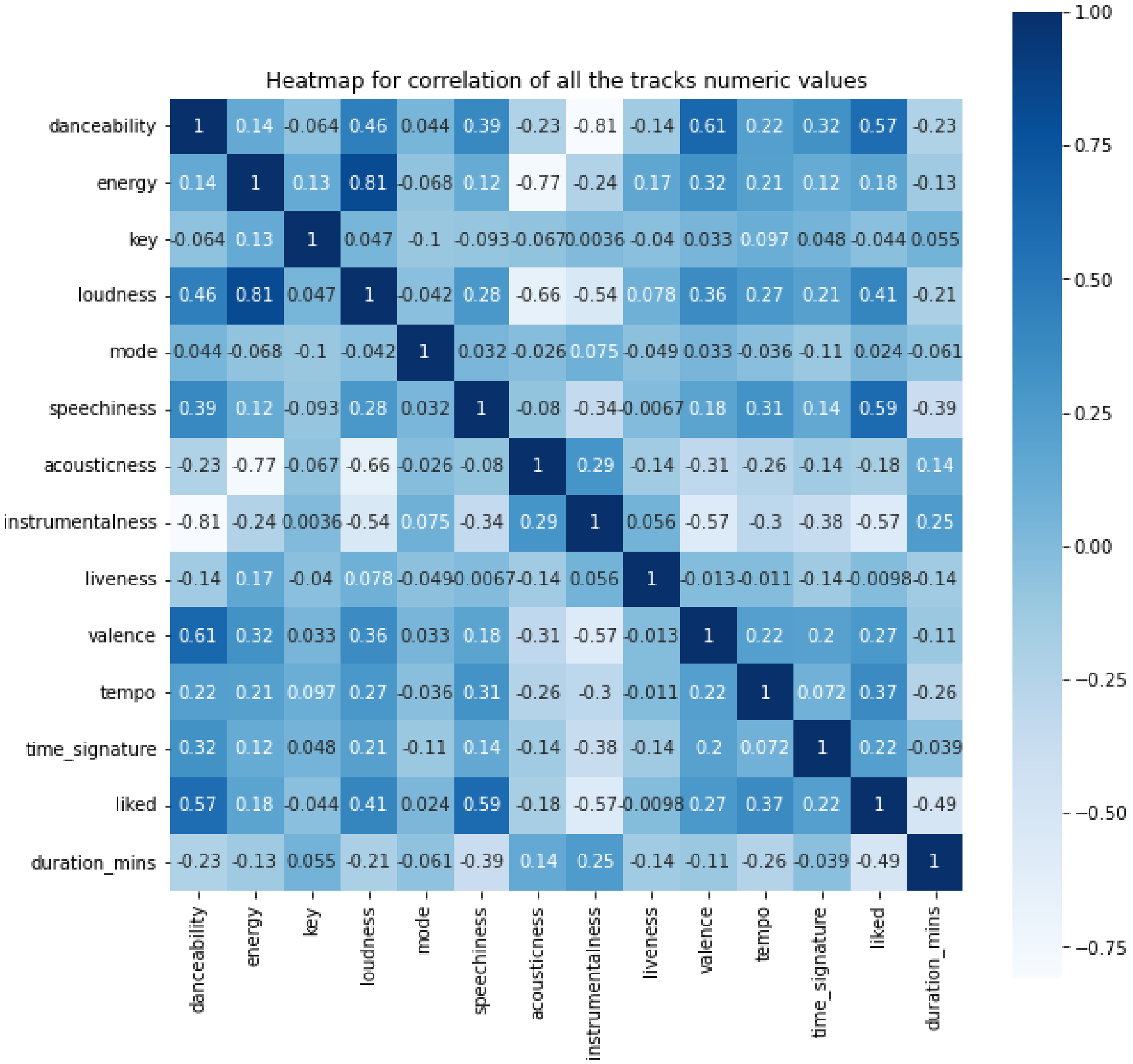
```
# Plotting histogram for all the parameters in dataset
histplot = spotify_data.hist(bins=20, figsize=(18,15))
```



This histogram is created for checking the relationship between the variables from dataset.

In [13]:

```
# Plotting correlation map to see relationship between parameters
# corr() function finds pairwise correlation of all columns
correlation = spotify_data.corr()
# Creating a heatmap of correlation using seaborn library
fig, axes = plt.subplots(figsize=(10,10))
sns.heatmap(correlation, cmap='Blues', square=True, annot=True)
# Giving the title to the heatmap
plt.title('Heatmap for correlation of all the tracks numeric values')
# Displaying the heatmap
plt.show()
```



After plotting heat map we got the idea of relation between the parameters of dataset.

- white color represent negative correlation
- blue represents medium correlation
- Dark blue represents high correlation

Now, Splitting up dataset into training data and test data

In [14]:
```
# Shuffling data to avoid prediction error
temp_data = spotify_data.sample(frac=1)

# calucating size for splitting up the data in two groups one called as
# training data other called as testing data.
sizeOfdata = int(len(spotify_data) * 0.75)

# splitting the dataset in two parts
training_dataset = temp_data[ : sizeOfdata]
testing_dataset = temp_data[sizeOfdata : ]

# assigning the training and testing dataset values by taking appropriate co
# as the columns are in series type we need to make them as np.array type so
# so it will directly convert them to np.arrays
x_training, y_training = training_dataset.drop('liked',axis=1).values, trair
x_testing, y_testing = testing_dataset.drop('liked',axis=1).values, testing_

# making transpose of matrix to make it in n by m size
x_training = x_training.T
```

```
# reshaping the matrix in shape 1 by m
y_training = y_training.reshape(1, x_training.shape[1])

# making transpose of matrix to make it in n by m size
x_testing = x_testing.T
# reshaping the matrix in shape 1 by m
y_testing = y_testing.reshape(1, x_testing.shape[1])

# printing the order of matrix for above transformation
print(x_training.shape)
print(y_training.shape)
print(x_testing.shape)
print(y_testing.shape)
```

```
(13, 146)
(1, 146)
(13, 49)
(1, 49)
```

In [15]:
```
# defining activation function which is also known as sigmoidal function
# function takes input as any number and convert it to number between 0 to 1
def sigmoid_function(z):
    return 1/(1+np.exp(-1*z))
```

In [16]:
```
# defining the prediction function which takes argument as weight matrix and
def probabilistic_prediction_function(weight_matrix, x_matrix, b):
    return np.dot(weight_matrix.T, x_matrix )+ b
```

In [17]:
```
# defining the cost function which takes argument as target martix, no of el
# and sigma value which is nothing but the predicted value from sigmoid func
def cost_function(y_matrix, size, sigma_value):
    return -(1/size)*np.sum(y_matrix*np.log(sigma_value)+(1-y_matrix)*np.log
```

In [18]:
```
# gradient descent to minimize the cost/error takes input as data matrix, pr
def gradient_descent(x_matrix, y_matrix, sigma_value, size):
    delta_w = (1/size)*np.dot(sigma_value-y_matrix, x_matrix.T)
    delta_b = (1/size)*np.sum(sigma_value+y_matrix)
    return delta_w, delta_b
```

In [19]:
```
# This code is referred from below link
# https://github.com/Jaimin09/Coding-Lane-Assets/blob/main/Logistic%20Regres
# defining algorithm for logistic regression with input as data matrix , ler
# and the number of iteration
def binar_logistic_regression_algorithm(X, Y, alpha, no_of_iterations):
    # getting the size of input matrix
    m = X.shape[1]
    n = X.shape[0]
    # Preparing weight matrix with order n by 1 and all values will be zero
    W = np.zeros((n,1))
    # bias value assigned as zero
    B = 0
    # list which will be used to store the value calculated for each iterati
    cost_list = []
    # loop which will run till n number of iterations given
    for iterate in range(no_of_iterations):
        # Calling predction function
        P = probabilistic_prediction_function(W, X, B)
        # calling Sigmoidal function with paramater as value return by predi
        sigma = sigmoid_function(P)
        # calling the cost function to calculate the error which conatin inp
        # value return from sigmoidal function
        cost = cost_function(Y, m, sigma)
        # calling gradient descent function which will return 2 values dw ar
```

```
        dW, dB = gradient_descent(X, Y, sigma, m)
        # calculating value of wight matrix with learning rate and derivativ
        W = W - alpha*dW.T
        # calculating value of bias with learning rate and derivative calcul
        B = B - alpha*dB
        # addng the cost value to cost list
        cost_list.append(cost)
        # if the current iteration number is divisible by a certain value (i
        # it prints the current iteration number and the current cost value.
        if iterate % (no_of_iterations/10) == 0:
            print(f"iteration {iterate}, objective: {cost}")
    # returning the weights, bias and cost list derived in above function
    return W, B, cost_list
```

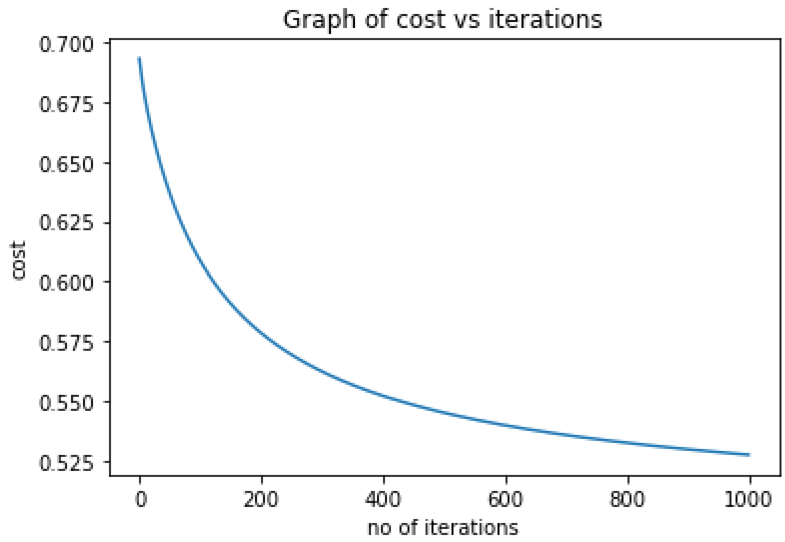
In [20]:

```
# calling the algorithm defined above with
# training datasets, learning rate as 0.0005 and iterations as 1000
W, B, cost_list = binar_logistic_regression_algorithm(x_training, y_training
```

```
iteration 0, objective: 0.6931471805599453
iteration 100, objective: 0.6084943912399452
iteration 200, objective: 0.5784217058267062
iteration 300, objective: 0.5623082386228897
iteration 400, objective: 0.5521156398602929
iteration 500, objective: 0.5450313729941813
iteration 600, objective: 0.5397891092607254
iteration 700, objective: 0.535726829017172
iteration 800, objective: 0.5324632775422579
iteration 900, objective: 0.5297636214271099
```

In [21]:

```
# This code is referred from below link
# https://github.com/Jaimin09/Coding-Lane-Assets/blob/main/Logistic%20Regres
# plotting the graph to visualize if the cost is reducing or not for given r
plt.plot(np.arange(1000),cost_list)
plt.xlabel('no of iterations')
plt.ylabel('cost')
plt.title('Graph of cost vs iterations')
plt.show()
```



Testing Model Accuracy

In [22]:

```
# defining function which us used to find accuracy of algorithm return above
# This code is referred from below link
# https://github.com/Jaimin09/Coding-Lane-Assets/blob/main/Logistic%20Regres
def accuracy(X, Y, W, B):
```

```
    # Calling prediction function
    Z = probabilistic_prediction_function(W, X, B)
    # calling sigmoidal function with value calculated for prediction functi
    A = sigmoid_function(Z)
    # checking for value present in A is greater than 0.5 just to check
    A = A > 0.5
    # Creating new numpy array from A matrix with datatype as int64
    A = np.array(A, dtype = 'int64')
    # calculating the accuracy of a model or prediction by calculating
    # the mean absolute error and then converting it to percentage accuracy.
    acc = (1 - np.sum(np.absolute(A - Y))/Y.shape[1])*100
    # printing the value of accuracy with last two decimal point
    print("Accuracy of the model is : ", round(acc, 2), "%")
```

In [23]:
```
# Calling accuracy function by sending input as testing dataset with weight
# binary logistic regression algorithm define above
accuracy(x_testing, y_testing, W, B)
```

```
Accuracy of the model is :  87.76 %
```

Testing the algorithm's performance by excluding certain columns

In this case removed instrumentalness and mode parameter from dataset to see how algorithm behaves and output result

- Here i removed instrumentalness and mode column to check accuracy

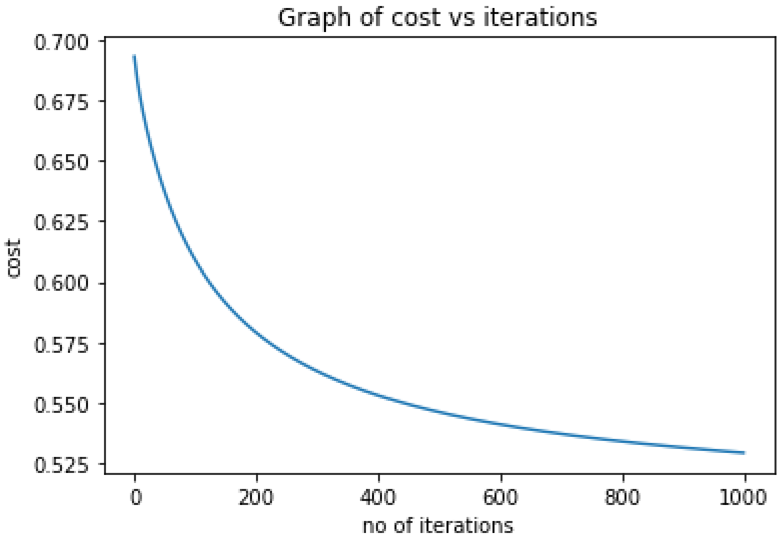
In [24]:
```
# Removing some columns to measure the accuracy of algorithm
temp_data1 = temp_data.drop(['instrumentalness','mode'],axis=1)

# calucating size for splitting up the data in two groups one called as
# training data other called as testing data.
sizeOfdata = int(len(spotify_data) * 0.75)

# splitting the dataset in two parts
training_dataset = temp_data1[ : sizeOfdata]
testing_dataset = temp_data1[sizeOfdata : ]

# assigning the training and testing dataset values by taking appropriate cc
# as the columns are in series type we need to make them as np.array type sc
# so it will directly convert them to np.arrays
x_training, y_training = training_dataset.drop('liked',axis=1).values, train
x_testing, y_testing = testing_dataset.drop('liked',axis=1).values, testing_

# making transpose of matrix to make it in n by m size
x_training = x_training.T
# reshaping the matrix in shape 1 by m
y_training = y_training.reshape(1, x_training.shape[1])

# making transpose of matrix to make it in n by m size
x_testing = x_testing.T
# reshaping the matrix in shape 1 by m
y_testing = y_testing.reshape(1, x_testing.shape[1])

# printing the order of matrix for above transformation
print(x_training.shape)
print(y_training.shape)
print(x_testing.shape)
print(y_testing.shape)
```

```
(11, 146)
(1, 146)
(11, 49)
(1, 49)
```

In [25]:
```
#Calling the function to test the newly created dataset after  remobing the
W, B, cost_list = binar_logistic_regression_algorithm(x_training, y_training
```

```
iteration 0, objective: 0.6931471805599453
iteration 100, objective: 0.6088187800517987
iteration 200, objective: 0.5789341554433725
iteration 300, objective: 0.5629840320275862
iteration 400, objective: 0.552945926513415
iteration 500, objective: 0.5460114059160153
iteration 600, objective: 0.5409156811502827
iteration 700, objective: 0.5369974747552252
iteration 800, objective: 0.5338759610184421
iteration 900, objective: 0.5313165890483339
```

In [26]:
```
# plotting the graph to visualize if the cost is reducing or not for given n
plt.plot(np.arange(1000),cost_list)
plt.xlabel('no of iterations')
plt.ylabel('cost')
plt.title('Graph of cost vs iterations')
plt.show()
```



In [27]:
```
#Calling the accuracy function which will return the result
accuracy(x_testing, y_testing, W, B)
```

```
Accuracy of the model is :  87.76 %
```

- Now, removing loudness, valence and instrumentalness to check the accuracy of algorithm

In [28]:
```
# Removing some columns to measure the accuracy of algorithm
temp_data2 = temp_data.drop(['loudness','valence','instrumentalness'],axis=1

# calucating size for splitting up the data in two groups one called as
# training data other called as testing data.
sizeOfdata = int(len(spotify_data) * 0.75)

# splitting the dataset in two parts
training_dataset = temp_data2[ : sizeOfdata]
testing_dataset = temp_data2[sizeOfdata : ]
```

```
# assigning the training and testing dataset values by taking appropriate co
# as the columns are in series type we need to make them as np.array type so
# so it will directly convert them to np.arrays
x_training, y_training = training_dataset.drop('liked',axis=1).values, train
x_testing, y_testing = testing_dataset.drop('liked',axis=1).values, testing_

# making transpose of matrix to make it in n by m size
x_training = x_training.T
# reshaping the matrix in shape 1 by m
y_training = y_training.reshape(1, x_training.shape[1])

# making transpose of matrix to make it in n by m size
x_testing = x_testing.T
# reshaping the matrix in shape 1 by m
y_testing = y_testing.reshape(1, x_testing.shape[1])

# printing the order of matrix for above transformation
print(x_training.shape)
print(y_training.shape)
print(x_testing.shape)
print(y_testing.shape)
```

```
(10, 146)
(1, 146)
(10, 49)
(1, 49)
```

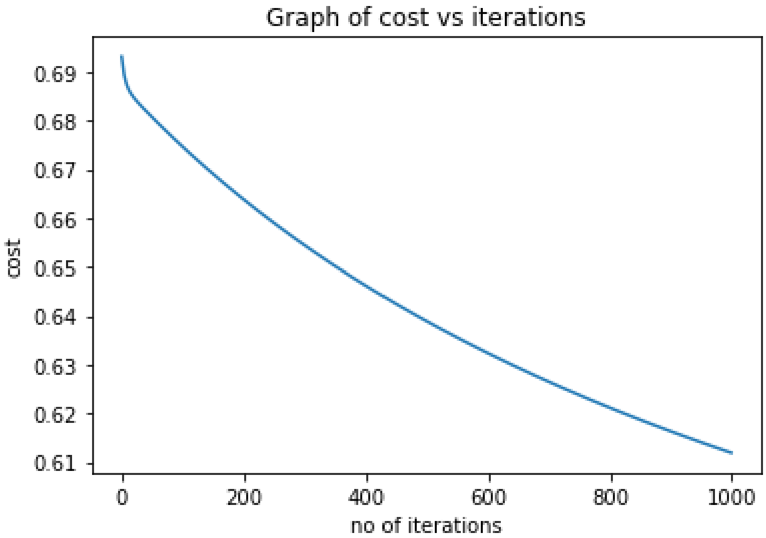
In [29]:

```
#Calling the function to test the newly created dataset after  remobing the
W, B, cost_list = binar_logistic_regression_algorithm(x_training, y_training
```

```
iteration 0, objective: 0.6931471805599453
iteration 100, objective: 0.674644964101805
iteration 200, objective: 0.663830066873527
iteration 300, objective: 0.6544375987802056
iteration 400, objective: 0.6461925340415315
iteration 500, objective: 0.6388853025717947
iteration 600, objective: 0.6323552153301287
iteration 700, objective: 0.6264779530948529
iteration 800, objective: 0.6211562771832443
iteration 900, objective: 0.6163131933387173
```

In [30]:

```
# plotting the graph to visualize if the cost is reducing or not for given n
plt.plot(np.arange(1000),cost_list)
plt.xlabel('no of iterations')
plt.ylabel('cost')
plt.title('Graph of cost vs iterations')
plt.show()
```



In [31]:
```
#Calling the accuracy function which will return the result
accuracy(x_testing, y_testing, W, B)
```

```
Accuracy of the model is :  75.51 %
```

In [ ]: