Spotify Song Popularity Prediction

In a Spotify song popularity prediction project, the goal is to build a model that can accurately predict how popular a given song will be on the platform. This will involve analyzing features of the song itself such as tempo, key, and duration etc

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import warnings
warnings.filterwarnings('ignore')
#Model
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import seaborn as sns
```

In [2]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import init_notebook_mode
import plotly.express as px
```

In [3]:
```
df = pd.read_csv("song_data.csv")
df
```

Out[3]:

	track	artist	uri	danceability	energy	key	loudne
0	Wild Things	Alessia Cara	spotify:track:2ZyuwVvV6Z3XJaXIFbspeE	0.741	0.626	1	-4.8
1	Surfboard	Esquivel!	spotify:track:61APOtq25SCMuK0V5w2Kgp	0.447	0.247	5	-14.6
2	Love Someone	Lukas Graham	spotify:track:2JqnpexlO9dmvjUMCaLCLJ	0.550	0.415	9	-6.5
3	Music To My Ears (feat. Tory Lanez)	Keys N Krates	spotify:track:0cjfLhk8WJ3etPTCseKXtk	0.502	0.648	0	-5.6
4	Juju On That Beat (TZ Anthem)	Zay Hilfigerrr & Zayion McCall	spotify:track:1lltf5ZXJc1by9SbPeljFd	0.807	0.887	1	-3.8
...	...	...	...	...	...	...	
6393	Lotus Flowers	Yolta	spotify:track:4t1TljQWJ6ZuoSY67zVvBI	0.172	0.358	9	-14.4
6394	Calling My Spirit	Kodak Black	spotify:track:2MShy1GSSgbmGUxADNIao5	0.910	0.366	1	-9.9
6395	Teenage Dream	Katy Perry	spotify:track:55qBw1900pZKfXJ6Q9A2Lc	0.719	0.804	10	-4.5
6396	Stormy Weather	Oscar Peterson	spotify:track:4o9npmYHrOF1rUxxTVH8h4	0.600	0.177	7	-16.0
6397	Dust	Hans Zimmer	spotify:track:2khIaVUkbMmDHB596lyMG3	0.121	0.123	4	-23.0

6398 rows × 19 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['track', 'artist', 'uri', 'danceability', 'energy', 'key', 'loudness',
       'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'duration_ms', 'time_signature', 'chorus_hit',
       'sections', 'target'],
      dtype='object')
```

Data Cleaning

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
track               0
artist              0
uri                 0
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
chorus_hit          0
sections            0
target              0
dtype: int64
```

Duplicate Check

In [6]:
```
df.drop_duplicates()
df.count()
```

Out[6]:
```
track               6398
artist              6398
uri                 6398
danceability        6398
energy              6398
key                 6398
loudness            6398
mode                6398
speechiness         6398
acousticness        6398
instrumentalness    6398
liveness            6398
valence             6398
tempo               6398
duration_ms         6398
time_signature      6398
chorus_hit          6398
sections            6398
target              6398
dtype: int64
```

In [7]:
```
df.rename(columns ={'target':'popularity'},inplace = True)
df
```

Out[7]:

	track	artist	uri	danceability	energy	key	loudne
0	Wild Things	Alessia Cara	spotify:track:2ZyuwVvV6Z3XJaXIFbspeE	0.741	0.626	1	-4.8
1	Surfboard	Esquivel!	spotify:track:61APOtq25SCMuK0V5w2Kgp	0.447	0.247	5	-14.6
2	Love Someone	Lukas Graham	spotify:track:2JqnpexlO9dmvjUMCaLCLJ	0.550	0.415	9	-6.5
3	Music To My Ears (feat. Tory Lanez)	Keys N Krates	spotify:track:0cjfLhk8WJ3etPTCseKXtk	0.502	0.648	0	-5.6
4	Juju On That Beat (TZ Anthem)	Zay Hilfigerrr & Zayion McCall	spotify:track:1lltf5ZXJc1by9SbPeljFd	0.807	0.887	1	-3.8
...	...	...	...	...	...	...	
6393	Lotus Flowers	Yolta	spotify:track:4t1TljQWJ6ZuoSY67zVvBI	0.172	0.358	9	-14.4
6394	Calling My Spirit	Kodak Black	spotify:track:2MShy1GSSgbmGUxADNIao5	0.910	0.366	1	-9.9
6395	Teenage Dream	Katy Perry	spotify:track:55qBw1900pZKfXJ6Q9A2Lc	0.719	0.804	10	-4.5
6396	Stormy Weather	Oscar Peterson	spotify:track:4o9npmYHrOF1rUxxTVH8h4	0.600	0.177	7	-16.0
6397	Dust	Hans Zimmer	spotify:track:2khIaVUkbMmDHB596lyMG3	0.121	0.123	4	-23.0

6398 rows × 19 columns

Analyzing the data

In [8]:
```
df.describe()
```

Out[8]:

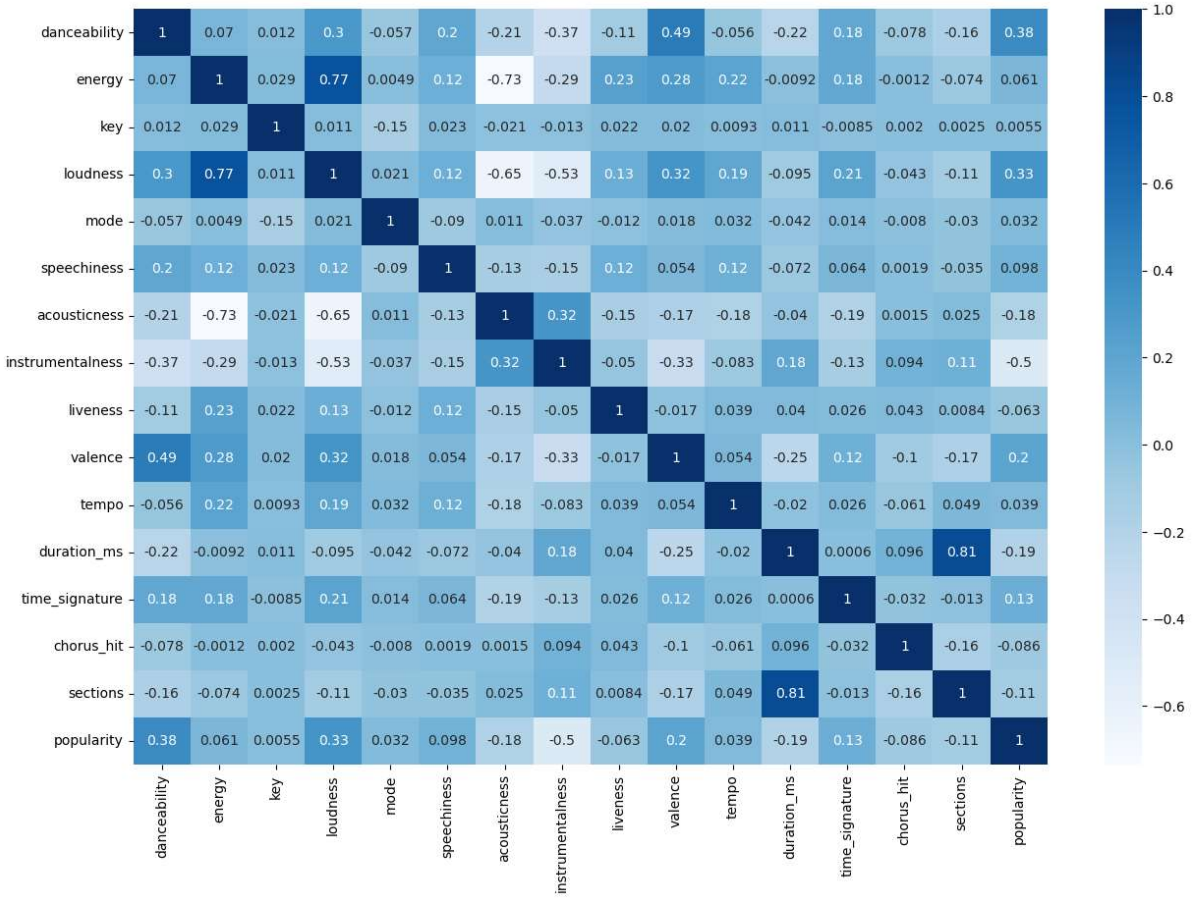
	danceability	energy	key	loudness	mode	speechiness	acousticness
count	6398.000000	6398.000000	6398.000000	6398.000000	6398.000000	6398.000000	6398.000000
mean	0.568163	0.667756	5.283526	-7.589796	0.645514	0.098018	0.216928
std	0.191103	0.240721	3.606216	5.234592	0.478395	0.097224	0.296835
min	0.062200	0.000251	0.000000	-46.655000	0.000000	0.022500	0.000000
25%	0.447000	0.533000	2.000000	-8.425000	0.000000	0.038825	0.008533
50%	0.588000	0.712500	5.000000	-6.096500	1.000000	0.057200	0.067050
75%	0.710000	0.857000	8.000000	-4.601250	1.000000	0.112000	0.311000
max	0.981000	0.999000	11.000000	-0.149000	1.000000	0.956000	0.996000

In [9]:
```
df.corr()
```

Out[9]:

	danceability	energy	key	loudness	mode	speechiness	acousticness
danceability	1.000000	0.069645	0.012429	0.300576	-0.057280	0.200090	-0.206865
energy	0.069645	1.000000	0.028703	0.774536	0.004929	0.119194	-0.734853
key	0.012429	0.028703	1.000000	0.010824	-0.146063	0.022796	-0.021271
loudness	0.300576	0.774536	0.010824	1.000000	0.021064	0.122028	-0.648717
mode	-0.057280	0.004929	-0.146063	0.021064	1.000000	-0.090107	0.011424
speechiness	0.200090	0.119194	0.022796	0.122028	-0.090107	1.000000	-0.134226
acousticness	-0.206865	-0.734853	-0.021271	-0.648717	0.011424	-0.134226	1.000000
instrumentalness	-0.371334	-0.288263	-0.013049	-0.533671	-0.037132	-0.148649	0.315563
liveness	-0.107581	0.231393	0.021785	0.126807	-0.011590	0.121075	-0.149926
valence	0.494136	0.281031	0.019572	0.324985	0.018198	0.053836	-0.166253
tempo	-0.056197	0.216886	0.009259	0.194467	0.032180	0.117813	-0.182050
duration_ms	-0.224803	-0.009228	0.011028	-0.094598	-0.042125	-0.071826	-0.039567
time_signature	0.178671	0.175984	-0.008462	0.207436	0.014125	0.063656	-0.191607
chorus_hit	-0.078254	-0.001224	0.001960	-0.042665	-0.007967	0.001857	0.001477
sections	-0.162908	-0.074466	0.002512	-0.111469	-0.030129	-0.035077	0.024583
popularity	0.384486	0.060701	0.005548	0.327471	0.032021	0.097783	-0.184479

In [10]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), cmap="Blues", annot=True)
plt.show()
```

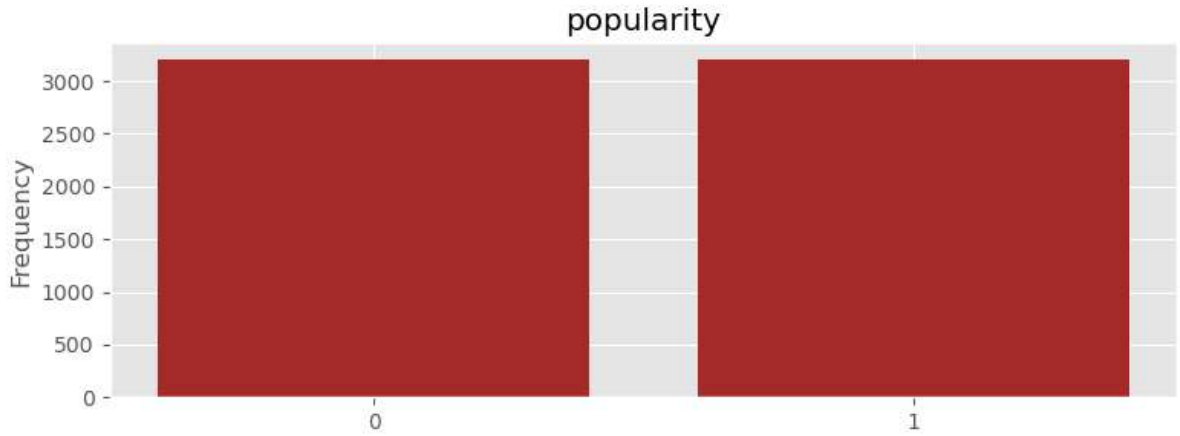


The above correation chart depicts that the target variable is highly correated with danceability, loudness and valence.

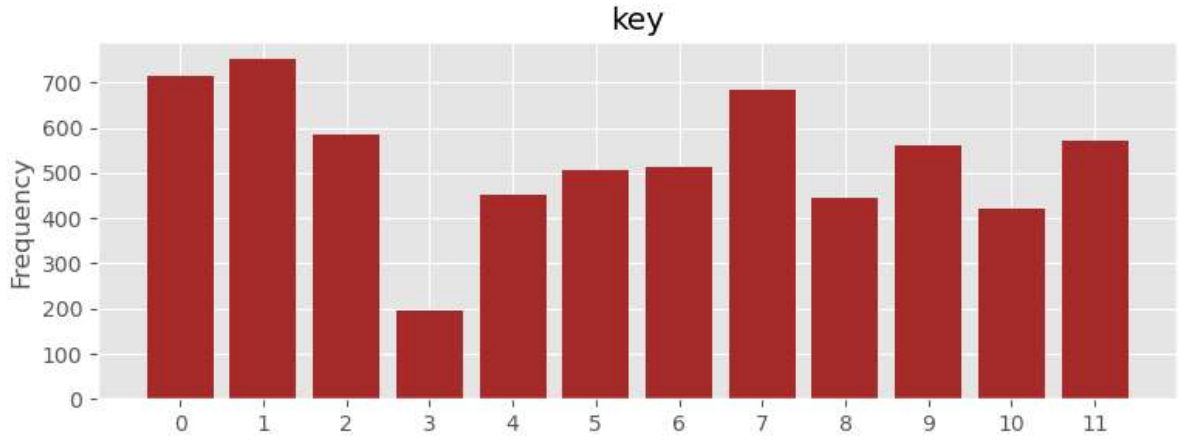
Analyzing categorical data

In [11]:
```
def bar_plot(variable):

    var=df[variable]
    var_value= var.value_counts()

    #visualize
    plt.figure(figsize=(9,3))
    plt.style.use('ggplot')
    plt.bar(var_value.index,var_value,color="Brown")
    plt.xticks(var_value.index,var_value.index.values)
    plt.ylabel("Frequency")
    plt.title(variable)
    plt.show()
    print("{}:\n{}".format(variable,var_value))
```

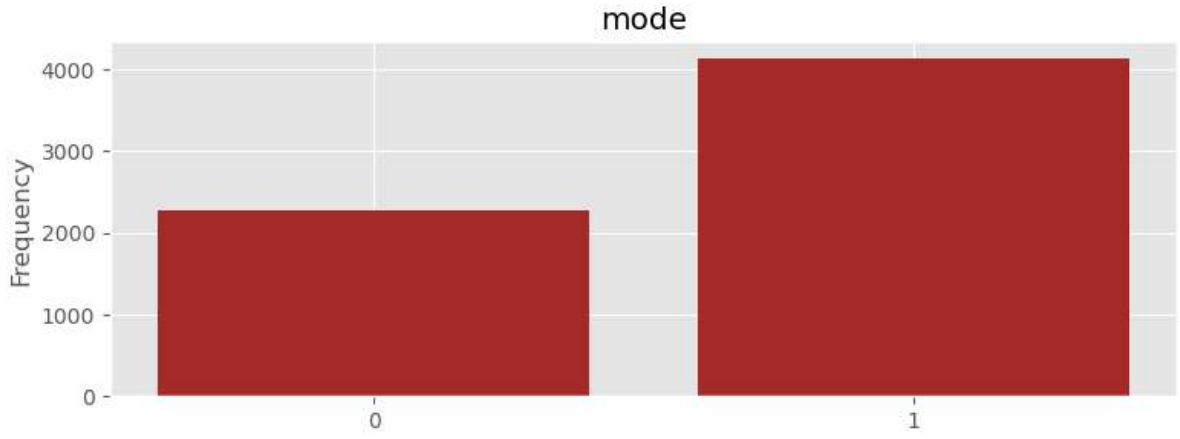
In [12]:
```
category1 = ["popularity","key","mode","time_signature"]
for c in category1:
    bar_plot(c)
```



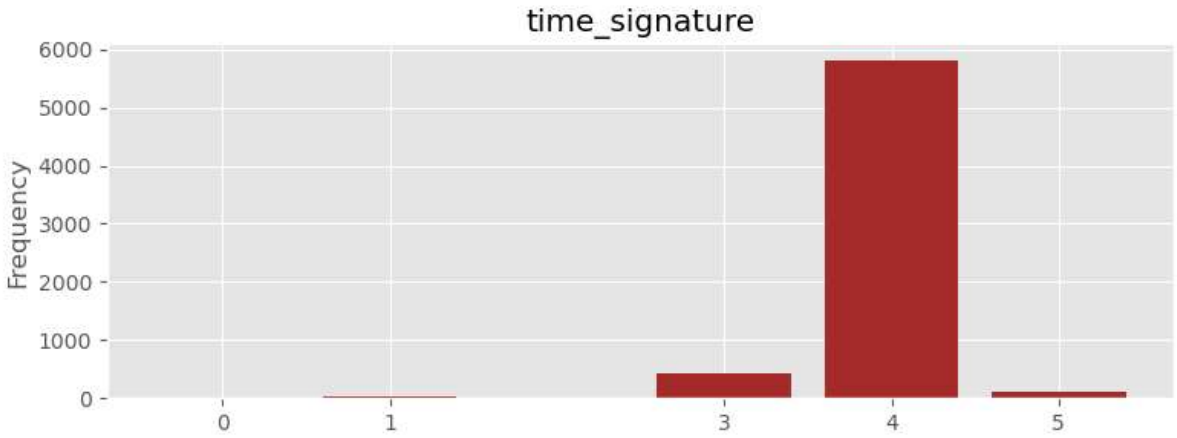

```
popularity:
1    3199
0    3199
Name: popularity, dtype: int64
```




```
key:
1     751
0     715
7     682
2     584
11    572
9     560
6     513
5     507
4     452
8     445
10    421
3     196
Name: key, dtype: int64
```



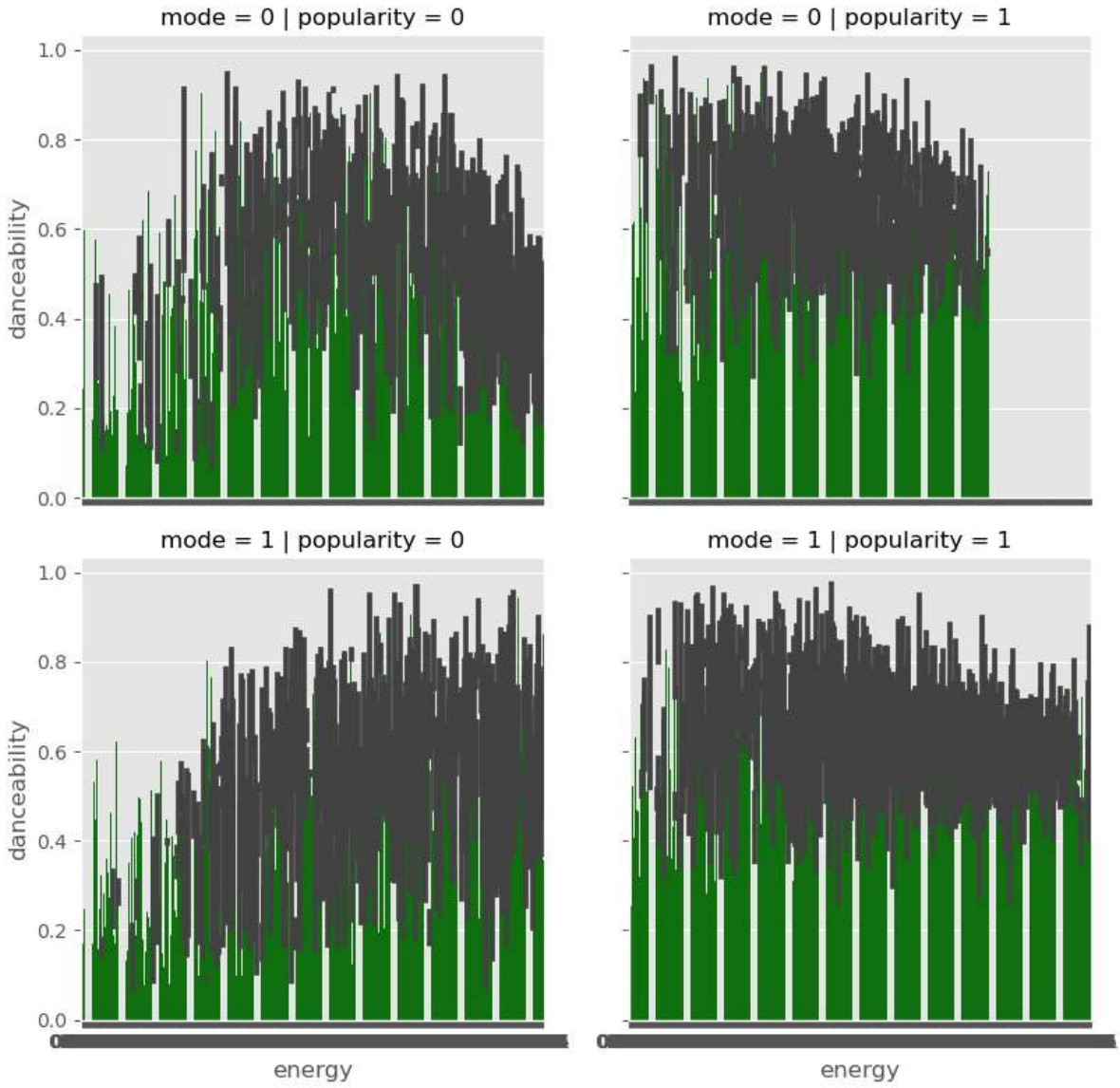
```
mode:
1    4130
0    2268
Name: mode, dtype: int64
```




```
time_signature:
4    5799
3     436
5     121
1      41
0       1
Name: time_signature, dtype: int64
```

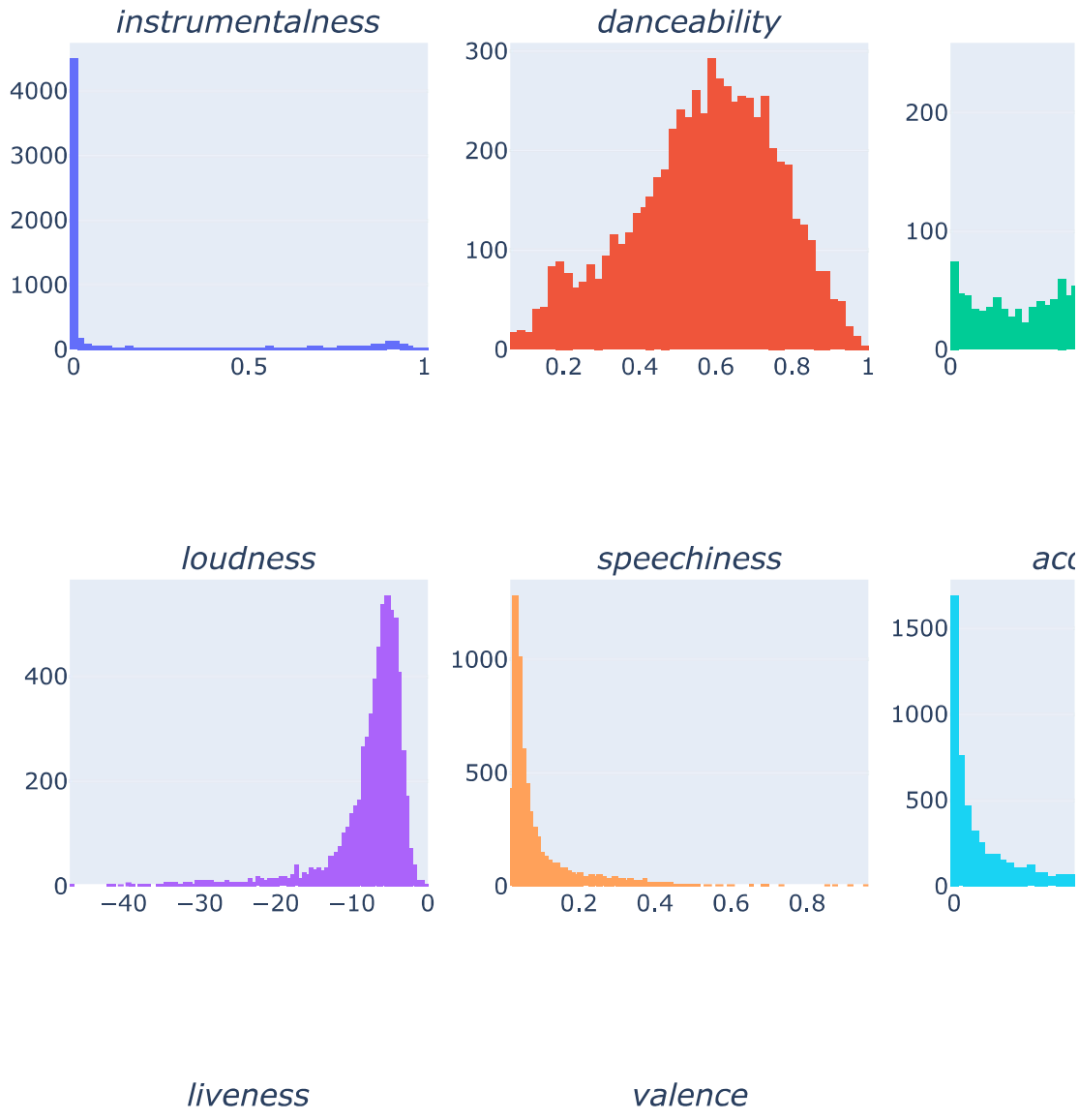
Numerical data

In [13]:
```
g = sns.FacetGrid(df, row = "mode", col = "popularity", height = 4)

g.map(sns.barplot, "energy", "danceability",color="Green")
g.add_legend()
plt.show()
```



In [14]:
```
ax=make_subplots(rows=3,cols=3,subplot_titles=('<i>instrumentalness', '<i>danceabili
ax.add_trace(go.Histogram(x=df['instrumentalness'],name='instrumentalness'),row=1,co
ax.add_trace(go.Histogram(x=df['danceability'],name='danceability'),row=1,col=2)
ax.add_trace(go.Histogram(x=df['energy'],name='energy'),row=1,col=3)
ax.add_trace(go.Histogram(x=df['loudness'],name='loudness'),row=2,col=1)
ax.add_trace(go.Histogram(x=df['speechiness'],name='speechiness'),row=2,col=2)
ax.add_trace(go.Histogram(x=df['acousticness'],name='acousticness'),row=2,col=3)
ax.update_layout(height=900,width=900,title_text='<b>Feature Distribution')
```

Feature Distribution

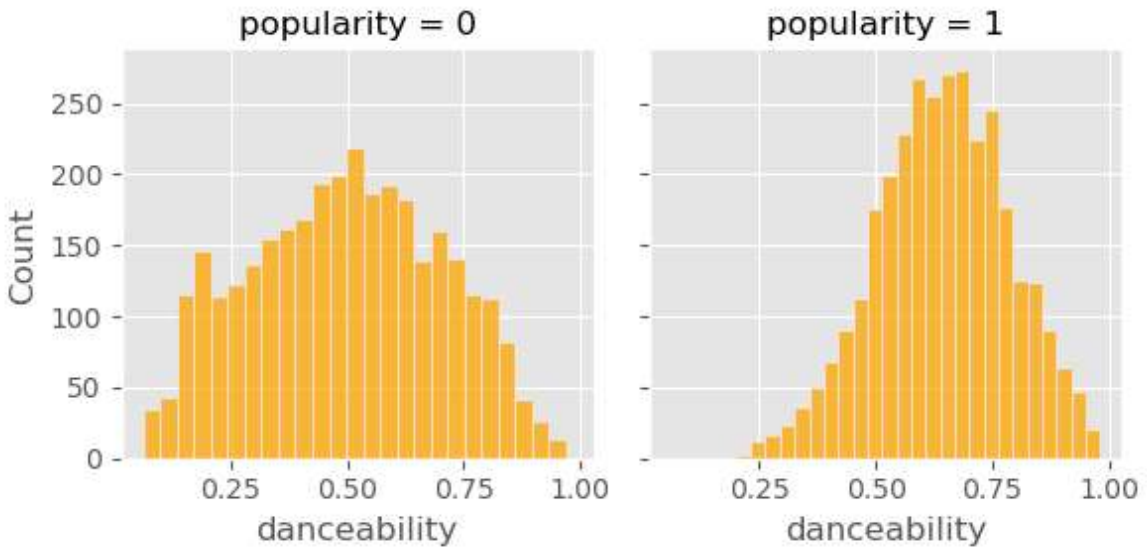


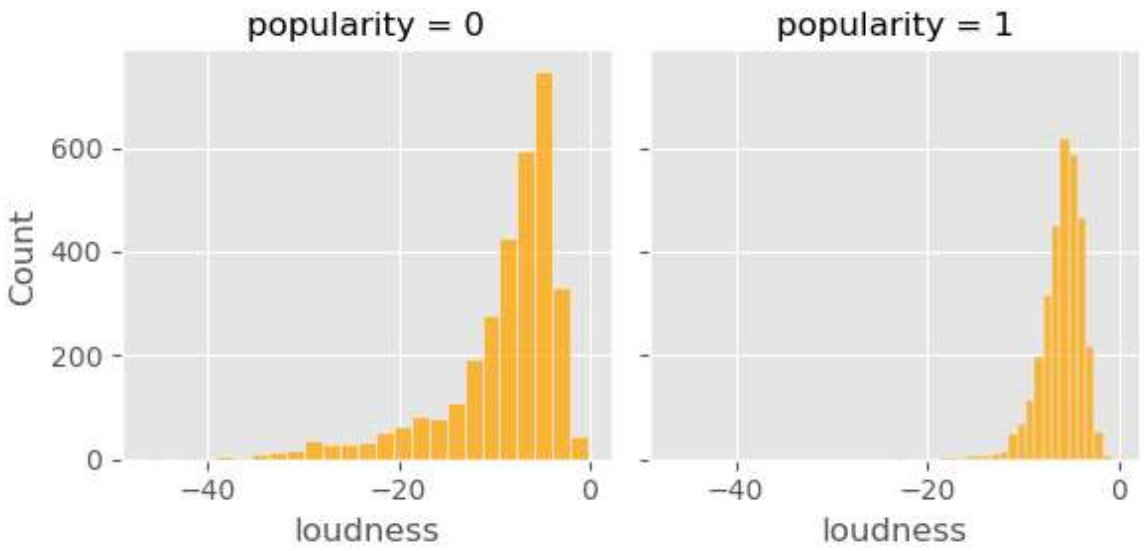

```
In [15]: px.scatter(df,x='valence',y='danceability',color='tempo',color_continuous_scale=px.c
```

Valence Vs Danceability

As we have seen that the valence and danceability are highly correlated, the graph depicts that valence is proportional to danceability

In [16]:
```
g = sns.FacetGrid(df, col = "popularity")
g.map(sns.histplot, "danceability", bins = 25,color='orange')
plt.show()
g = sns.FacetGrid(df, col = "popularity")
g.map(sns.histplot, "loudness", bins = 25,color="orange")
plt.show()
```





Feature Validation

In [17]:
```
for col in df.columns:
    miss = df[col].isnull().sum()
    if miss>0:
        print("{} has {} missing value(s)".format(col,miss))
    else:
        print("{} has NO missing value!".format(col))
```

```
track has NO missing value!
artist has NO missing value!
uri has NO missing value!
danceability has NO missing value!
energy has NO missing value!
key has NO missing value!
loudness has NO missing value!
mode has NO missing value!
speechiness has NO missing value!
acousticness has NO missing value!
instrumentalness has NO missing value!
liveness has NO missing value!
valence has NO missing value!
tempo has NO missing value!
duration_ms has NO missing value!
time_signature has NO missing value!
chorus_hit has NO missing value!
sections has NO missing value!
popularity has NO missing value!
```

Feature Engineering

In [18]:
```
x = df.drop(['track','artist','uri','popularity'],axis = 1)
y = df['popularity']
```

In [19]:
```
x
```

Out[19]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	live
0	0.741	0.626	1	-4.826	0	0.0886	0.02000	0.000000	0.(
1	0.447	0.247	5	-14.661	0	0.0346	0.87100	0.814000	0.(
2	0.550	0.415	9	-6.557	0	0.0520	0.16100	0.000000	0.
3	0.502	0.648	0	-5.698	0	0.0527	0.00513	0.000000	0.
4	0.807	0.887	1	-3.892	1	0.2750	0.00381	0.000000	0.
...	...	...	...	...	...	...	...	...	
6393	0.172	0.358	9	-14.430	1	0.0342	0.88600	0.966000	0.
6394	0.910	0.366	1	-9.954	1	0.0941	0.09960	0.000000	0.
6395	0.719	0.804	10	-4.581	1	0.0355	0.01320	0.000003	0.
6396	0.600	0.177	7	-16.070	1	0.0561	0.98900	0.868000	0.
6397	0.121	0.123	4	-23.025	0	0.0443	0.96400	0.696000	0.

6398 rows × 15 columns

In [20]:
```
y
```

Out[20]:
```
0       1
1       0
2       1
3       0
4       1
       ..
6393    0
6394    1
6395    1
6396    0
6397    0
Name: popularity, Length: 6398, dtype: int64
```

Splitting and scaling of the data

Here we are trying to split and scale the data accordingly to validate the model and to get good accuracy

In [21]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.7, shuffle
```

In [22]:
```
x_train
```

Out[22]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	live
5590	0.598	0.157	0	-15.138	1	0.0290	0.959000	0.681000	0.
1224	0.378	0.891	10	-4.735	1	0.0428	0.000010	0.013600	0.
4692	0.709	0.824	1	-8.824	1	0.0453	0.207000	0.000307	0.
3917	0.573	0.876	6	-7.644	0	0.0482	0.000024	0.000519	0.
1071	0.558	0.814	2	-5.378	1	0.0645	0.012900	0.000000	0.
...	...	...	...	...	...	...	...	...	
3772	0.715	0.655	0	-6.425	1	0.1370	0.052500	0.000000	0.
5191	0.869	0.846	4	-9.360	0	0.1390	0.233000	0.000013	0.
5226	0.610	0.893	9	-3.681	1	0.0921	0.154000	0.000044	0.
5390	0.747	0.553	8	-7.121	1	0.3720	0.192000	0.000000	0.
860	0.695	0.762	0	-3.497	1	0.0395	0.192000	0.002440	0.

4478 rows × 15 columns

Model Selection

In [23]:
```
models = {
    "                    Linear Regression": LinearRegression(),
    "                  Logistic Regression": LogisticRegression(),
    "                  K-Nearest Neighbors": KNeighborsClassifier(),
    "                        Decision Tree": DecisionTreeClassifier(),
    "Support Vector Machine (Linear Kernel)": LinearSVC(),
    "   Support Vector Machine (RBF Kernel)": SVC(),
    "                       Neural Network": MLPClassifier(),
    "                        Random Forest": RandomForestClassifier(),
    "                    Gradient Boosting": GradientBoostingClassifier()
}

for name, model in models.items():
    model.fit(x_train, y_train)
    print(name + " trained.")
```

```
                    Linear Regression trained.
                  Logistic Regression trained.
                  K-Nearest Neighbors trained.
                        Decision Tree trained.
Support Vector Machine (Linear Kernel) trained.
   Support Vector Machine (RBF Kernel) trained.
                       Neural Network trained.
                        Random Forest trained.
                    Gradient Boosting trained.
```

In [24]:
```
for name, model in models.items():
    print(name + ": {:.2f}%".format(model.score(x_test, y_test) * 100))
```

```
                     Linear Regression: 34.64%
                   Logistic Regression: 64.11%
                   K-Nearest Neighbors: 63.07%
                         Decision Tree: 77.66%
Support Vector Machine (Linear Kernel): 49.69%
   Support Vector Machine (RBF Kernel): 64.79%
                       Neural Network: 50.31%
                         Random Forest: 83.70%
                     Gradient Boosting: 83.44%
```

After testing the models with our data, we can see it's inferior idea to go with regression model, we chose the Randomforest Classifier as it stood over other models in terms of accuracy with 83.70% without any hyperparameter tuning. We are going to tune the randomforest furthermore.

In [25]:
```
import sklearn.metrics as sm
# creating function to evaluate model at one run
def evaluate(model,model_name):
    performance[model_name]=[sm.mean_absolute_error(y_test, model.predict(x_test)),
    sm.mean_squared_error(y_test, model.predict(x_test))**0.5,]
pd.options.display.float_format = '{:.3f}'.format
```

In [26]:
```
performance=pd.DataFrame(data=None, index=['MAE','RMSE'])
performance
```

Out[26]:

MAE	
RMSE	

In [27]:
```
for name, model in models.items():
    model.fit(x_train, y_train)
    evaluate(model, name)
    performance[[name]]
```

In [28]:
```
performance
```

Out[28]:

	Linear Regression	Logistic Regression	K-Nearest Neighbors	Decision Tree	Support Vector Machine (Linear Kernel)	Support Vector Machine (RBF Kernel)	Neural Network	Random Forest	Gradie Boosti
MAE	0.357	0.359	0.369	0.218	0.497	0.352	0.497	0.164	0.1
RMSE	0.404	0.599	0.608	0.467	0.705	0.593	0.705	0.405	0.4

In [29]:
```
rd = RandomForestClassifier(n_estimators=10,max_depth = 20,random_state =5,criterior
rd.fit(x_train,y_train)
rd.score(x_train,y_train)
```

Out[29]:
```
0.9738722644037516
```

In [30]:
```
rd.score(x_test,y_test)
```

Out[30]:
```
0.8359375
```

In [31]:
```
y_pred = rd.predict(x_test)
y_pred
```

```
y_pred = pd.DataFrame(y_pred)
y_pred
```

Out[31]:

	0
0	0
1	0
2	1
3	1
4	0
...	...
1915	1
1916	1
1917	1
1918	0
1919	0

1920 rows × 1 columns

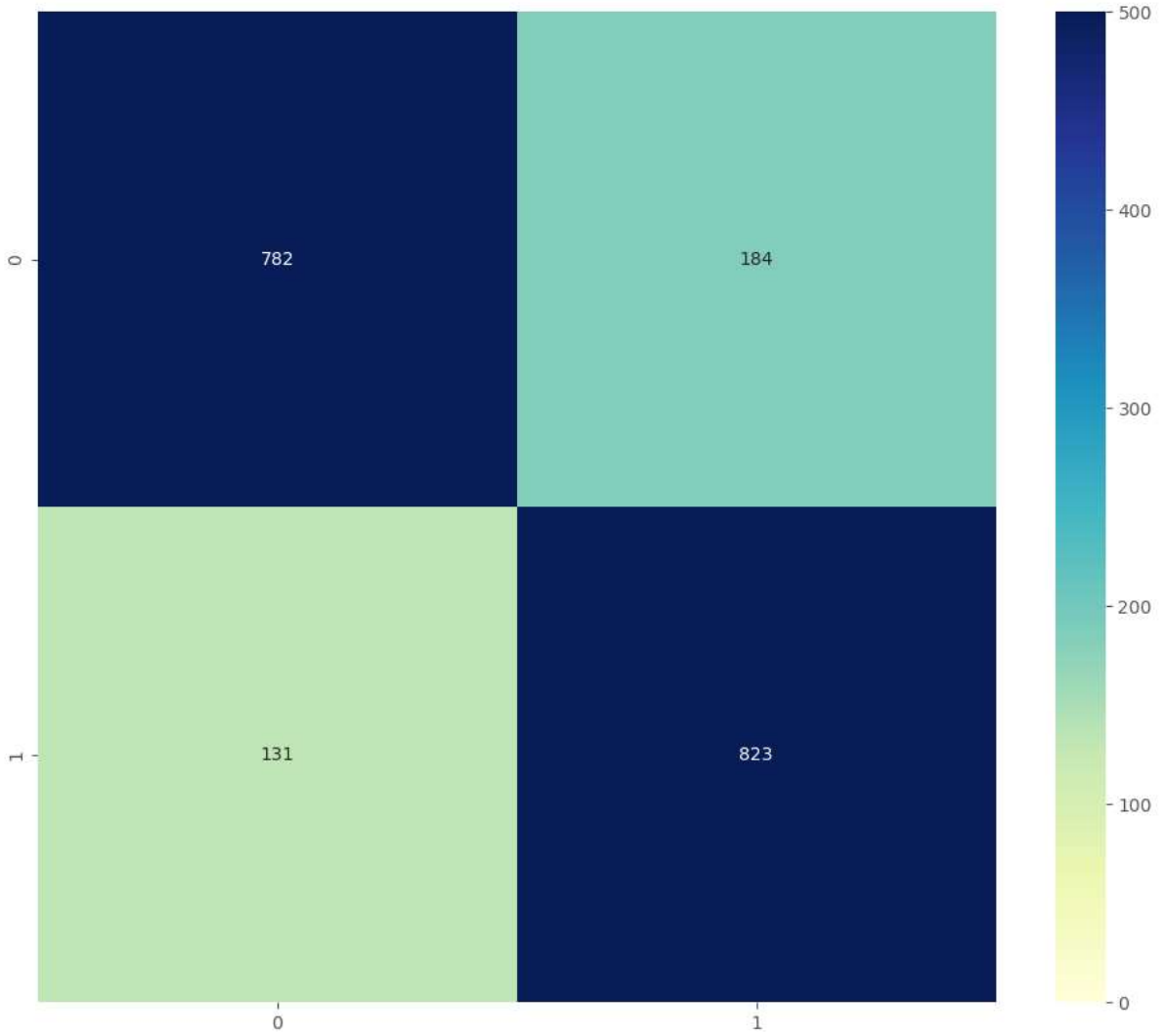
Metrics

In [32]:
```
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',cm)
print('Classification report: \n',classification_report(y_test,y_pred))
```

```
Confusion matrix:
 [[782 184]
 [131 823]]
Classification report:
              precision    recall  f1-score   support

           0       0.86      0.81      0.83       966
           1       0.82      0.86      0.84       954

    accuracy                           0.84      1920
   macro avg       0.84      0.84      0.84      1920
weighted avg       0.84      0.84      0.84      1920
```

In [33]:
```
f,ax1=plt.subplots(figsize=(12,10))
cm = confusion_matrix(y_test,y_pred)
sns.heatmap(cm, annot = True, fmt = 'd', vmin = 0, vmax = 500,cmap = 'YlGnBu', ax =
```

Out[33]:
```
<AxesSubplot:>
```



In [34]:
```
y_preddd = rd.predict(x_train)
y_preddd
```

Out[34]:
```
array([0, 0, 1, ..., 1, 1, 1], dtype=int64)
```

In [35]:
```
dff = pd.read_csv("song_validate.csv")
x1 = df.drop(['track','artist','uri','popularity'],axis = 1)
```

In [36]:
```
dfff=x1.head(10)
```

In [37]:
```
y1_pred = rd.predict(x1)
```

In [38]:
```
y1_pred
```

Out[38]:
```
array([1, 0, 1, ..., 1, 0, 0], dtype=int64)
```

UI Code

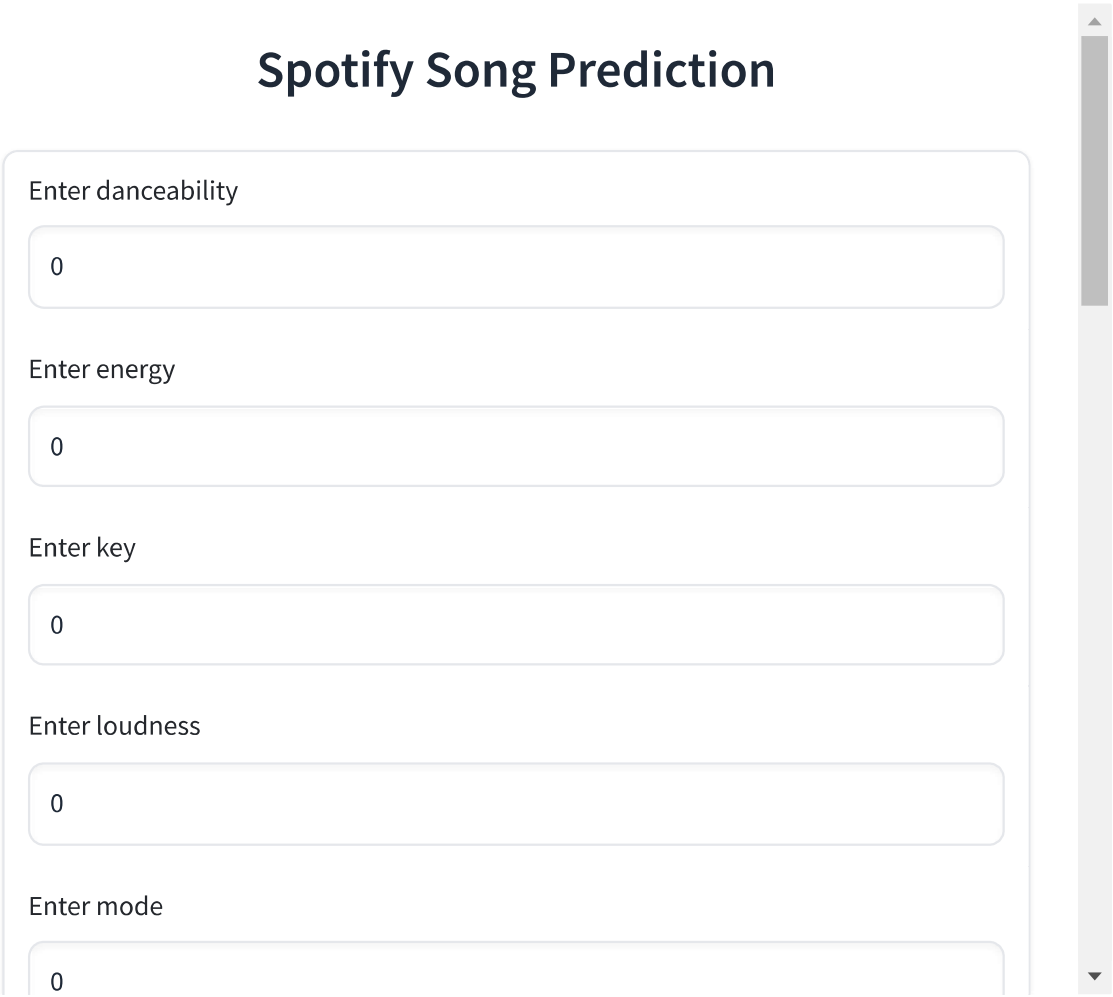
We are developing the UI by using gradio library in Python, this will display the results whether song is popular or not based on inputs

Note: After running the Code, you can able to see the gradio link, please copy and paste in the google or you can use it in here also for seeing the results.

In [39]:
```
import pickle

with open("predict.pkl", "wb") as f:
    pickle.dump(rd, f)
```

In [40]:
```
import gradio as gr
def make_prediction(danceability,energy,key,loudness,mode,speechiness,acousticness,i
    with open("predict.pkl", "rb") as f:
        clf  = pickle.load(f)
        preds = clf.predict([[danceability,energy,key,loudness,mode,speechiness,acou
    if preds == 1:
        return "Song Popular"
    return "Song Not popular"

#Create the input component for Gradio since we are expecting 4 inputs

age_input = gr.Number(label = "Enter danceability")
employment_input = gr.Number(label= "Enter energy")
bank_input = gr.Number(label = "Enter key")
account_input = gr.Number(label = "Enter loudness")
mode_input = gr.Number(label = "Enter mode")
speechiness_input = gr.Number(label = "Enter the speechiness")
acousticness_input = gr.Number(label= "Enter acousticness")
instrumentalness_input = gr.Number(label = "Enter instrumentalness")
liveness_input = gr.Number(label = "Enter liveness")
valence_input = gr.Number(label = "Enter valence")
tempo_input = gr.Number(label = "Enter tempo")
duration_ms_input = gr.Number(label = "Enter duration_ms")
time_signature_input = gr.Number(label = "Enter time_signature")
chorus_hit_input = gr.Number(label = "Enter chorus_hit")
sections_input = gr.Number(label = "Enter sections")

# We create the output
output = gr.Textbox()


app = gr.Interface(fn = make_prediction, inputs=[age_input, employment_input, bank_i

app.launch()
```

```
Running on local URL:  http://127.0.0.1:7867

To create a public link, set `share=True` in `launch()`.
```


Spotify Song Prediction
Enter danceability
0
Enter energy
0
Enter key
0
Enter loudness
0
Enter mode
0

Out[40]:

In [ ]: