spotify_songs

September 10, 2023

1 *******Beacon Nicolas Mkhabele*******

2 *******Creacting Cohorts of Songs Project Machine Learning Project*******

import pandas as pd import numpy as np import matplotlib.pyplot as plt

2.1 Initial data inspection and data cleaning:Check whether the data has duplicates, missing values, irrelevant (erroneous entries) values, or outliers.

[2]:
```
dataframe = pd.read_csv("1673873388_rolling_stones_spotify.csv")
```

[3]:
```
# printing the available columns in the csv file
print(dataframe.columns.tolist())
```

```
['Unnamed: 0', 'name', 'album', 'release_date', 'track_number', 'id', 'uri',
'acousticness', 'danceability', 'energy', 'instrumentalness', 'liveness',
'loudness', 'speechiness', 'tempo', 'valence', 'popularity', 'duration_ms']
```

[4]:
```
# printing the duplicates in a csv file
duplicate_rows = dataframe[dataframe.duplicated()]
print(duplicate_rows)
```

```
Empty DataFrame
Columns: [Unnamed: 0, name, album, release_date, track_number, id, uri,
acousticness, danceability, energy, instrumentalness, liveness, loudness,
speechiness, tempo, valence, popularity, duration_ms]
Index: []
```

[5]:
```
# Removing the duplicates in a csv file
dataframe = dataframe.drop_duplicates()
print(dataframe)
```

```
      Unnamed: 0                          name               album  \
0              0    Concert Intro Music - Live  Licked Live In NYC
1              1    Street Fighting Man - Live  Licked Live In NYC
2              2            Start Me Up - Live  Licked Live In NYC
3              3  If You Can't Rock Me - Live  Licked Live In NYC
```

```
4             4           Don't Stop - Live  Licked Live In NYC
…          …                          …                 …
1605        1605                          Carol  The Rolling Stones
1606        1606                        Tell Me  The Rolling Stones
1607        1607            Can I Get A Witness  The Rolling Stones
1608        1608   You Can Make It If You Try  The Rolling Stones
1609        1609                Walking The Dog  The Rolling Stones

     release_date  track_number                      id  \
0       6/10/2022             1  2IEkywLJ4ykbhi1yRQvmsT
1       6/10/2022             2  6GVgVJBKkGJoRfarYRvGTU
2       6/10/2022             3  1Lu761pZ0dBTGpzxaQoZNW
3       6/10/2022             4  1agTQzOTUnGNggyckEqiDH
4       6/10/2022             5  7piGJR8YndQBQWVXv6KtQw
…             …            …                       …
1605    4/16/1964             8  08l7M5UpRnffGl0FyuRiQZ
1606    4/16/1964             9  3JZllQBsTM6WwoJdzFDLhx
1607    4/16/1964            10  0t2qvfSBQ3Y08lzRRoVTdb
1608    4/16/1964            11  5ivIs5vwSj0RChOIvlY3On
1609    4/16/1964            12  43SkTJJ2xleDaeiE4TIM70

                                uri  acousticness  danceability  \
0     spotify:track:2IEkywLJ4ykbhi1yRQvmsT        0.0824         0.463
1     spotify:track:6GVgVJBKkGJoRfarYRvGTU        0.4370         0.326
2     spotify:track:1Lu761pZ0dBTGpzxaQoZNW        0.4160         0.386
3     spotify:track:1agTQzOTUnGNggyckEqiDH        0.5670         0.369
4     spotify:track:7piGJR8YndQBQWVXv6KtQw        0.4000         0.303
…                                       …           …           …
1605  spotify:track:08l7M5UpRnffGl0FyuRiQZ        0.1570         0.466
1606  spotify:track:3JZllQBsTM6WwoJdzFDLhx        0.0576         0.509
1607  spotify:track:0t2qvfSBQ3Y08lzRRoVTdb        0.3710         0.790
1608  spotify:track:5ivIs5vwSj0RChOIvlY3On        0.2170         0.700
1609  spotify:track:43SkTJJ2xleDaeiE4TIM70        0.3830         0.727

      energy  instrumentalness  liveness  loudness  speechiness    tempo  \
0      0.993          0.996000    0.9320   -12.913       0.1100  118.001
1      0.965          0.233000    0.9610    -4.803       0.0759  131.455
2      0.969          0.400000    0.9560    -4.936       0.1150  130.066
3      0.985          0.000107    0.8950    -5.535       0.1930  132.994
4      0.969          0.055900    0.9660    -5.098       0.0930  130.533
…        …               …      …       …         …          …
1605   0.932          0.006170    0.3240    -9.214       0.0429  177.340
1606   0.706          0.000002    0.5160    -9.427       0.0843  122.015
1607   0.774          0.000000    0.0669    -7.961       0.0720   97.035
1608   0.546          0.000070    0.1660    -9.567       0.0622  102.634
1609   0.934          0.068500    0.0965    -8.373       0.0359  125.275

      valence  popularity  duration_ms
```

```
0       0.0302          33        48640
1       0.3180          34       253173
2       0.3130          34       263160
3       0.1470          32       305880
4       0.2060          32       305106
…        …          …          …
1605    0.9670          39       154080
1606    0.4460          36       245266
1607    0.8350          30       176080
1608    0.5320          27       121680
1609    0.9690          35       189186

[1610 rows x 18 columns]
```

[6]:

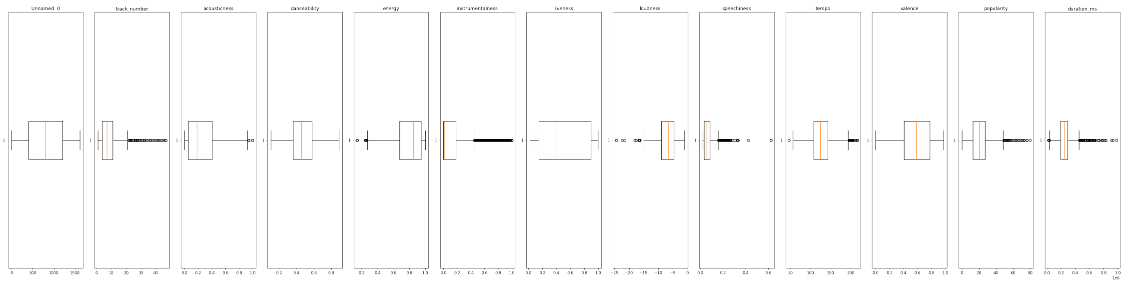
```
# Checking for missing values in a csv
missing_values = dataframe.isnull().sum()
print(missing_values)
```

```
Unnamed: 0          0
name                0
album               0
release_date        0
track_number        0
id                  0
uri                 0
acousticness        0
danceability        0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
valence             0
popularity          0
duration_ms         0
dtype: int64
```

[7]:

```
# Graphical representation of outliers using box and whisker plots
numeric_columns = dataframe.select_dtypes(include=['number'])

plt.figure(figsize=(40, 10))
for column in numeric_columns.columns:
    plt.subplot(1, len(numeric_columns.columns), numeric_columns.columns.
 ↪get_loc(column) + 1)
    plt.boxplot(dataframe[column], vert=False)
    plt.title(column)
```

```
plt.tight_layout()
plt.show()
```



2.2 Depending on your findings, clean the data for further processing.

[8]:
```
dataframe.fillna(dataframe.mean())
dataframe.dropna(axis=1)
```

```
/tmp/ipykernel_236/3033724010.py:1: FutureWarning: Dropping of nuisance columns
in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future
version this will raise TypeError.  Select only valid columns before calling the
reduction.
  dataframe.fillna(dataframe.mean())
```

[8]:
```
      Unnamed: 0                          name               album  \
0              0   Concert Intro Music - Live   Licked Live In NYC
1              1   Street Fighting Man - Live   Licked Live In NYC
2              2           Start Me Up - Live   Licked Live In NYC
3              3  If You Can't Rock Me - Live   Licked Live In NYC
4              4            Don’t Stop - Live   Licked Live In NYC
…            …                           …                  …
1605        1605                        Carol  The Rolling Stones
1606        1606                      Tell Me  The Rolling Stones
1607        1607            Can I Get A Witness  The Rolling Stones
1608        1608   You Can Make It If You Try  The Rolling Stones
1609        1609             Walking The Dog  The Rolling Stones

     release_date  track_number                      id  \
0       6/10/2022             1  2IEkywLJ4ykbhi1yRQvmsT
1       6/10/2022             2  6GVgVJBKkGJoRfarYRvGTU
2       6/10/2022             3  1Lu761pZ0dBTGpzxaQoZNW
3       6/10/2022             4  1agTQzOTUnGNggyckEqiDH
4       6/10/2022             5  7piGJR8YndQBQWVXv6KtQw
…             …           …                       …
1605    4/16/1964             8  0817M5UpRnffGl0FyuRiQZ
```

```
1606   4/16/1964           9  3JZllQBsTM6WwoJdzFDLhx
1607   4/16/1964          10  0t2qvfSBQ3Y08lzRRoVTdb
1608   4/16/1964          11  5ivIs5vwSj0RChOIvlY3On
1609   4/16/1964          12  43SkTJJ2xleDaeiE4TIM70

                                        uri  acousticness  danceability  \
0     spotify:track:2IEkywLJ4ykbhi1yRQvmsT        0.0824         0.463
1     spotify:track:6GVgVJBKkGJoRfarYRvGTU        0.4370         0.326
2     spotify:track:1Lu761pZ0dBTGpzxaQoZNW        0.4160         0.386
3     spotify:track:1agTQzOTUnGNggyckEqiDH        0.5670         0.369
4     spotify:track:7piGJR8YndQBQWVXv6KtQw        0.4000         0.303
…                                     …           …            …
1605  spotify:track:08l7M5UpRnffGl0FyuRiQZ        0.1570         0.466
1606  spotify:track:3JZllQBsTM6WwoJdzFDLhx        0.0576         0.509
1607  spotify:track:0t2qvfSBQ3Y08lzRRoVTdb        0.3710         0.790
1608  spotify:track:5ivIs5vwSj0RChOIvlY3On        0.2170         0.700
1609  spotify:track:43SkTJJ2xleDaeiE4TIM70        0.3830         0.727

      energy  instrumentalness  liveness  loudness  speechiness   tempo  \
0      0.993          0.996000    0.9320   -12.913       0.1100  118.001
1      0.965          0.233000    0.9610    -4.803       0.0759  131.455
2      0.969          0.400000    0.9560    -4.936       0.1150  130.066
3      0.985          0.000107    0.8950    -5.535       0.1930  132.994
4      0.969          0.055900    0.9660    -5.098       0.0930  130.533
…      …              …       …       …         …        …
1605   0.932          0.006170    0.3240    -9.214       0.0429  177.340
1606   0.706          0.000002    0.5160    -9.427       0.0843  122.015
1607   0.774          0.000000    0.0669    -7.961       0.0720   97.035
1608   0.546          0.000070    0.1660    -9.567       0.0622  102.634
1609   0.934          0.068500    0.0965    -8.373       0.0359  125.275

      valence  popularity  duration_ms
0      0.0302          33        48640
1      0.3180          34       253173
2      0.3130          34       263160
3      0.1470          32       305880
4      0.2060          32       305106
…       …          …           …
1605   0.9670          39       154080
1606   0.4460          36       245266
1607   0.8350          30       176080
1608   0.5320          27       121680
1609   0.9690          35       189186

[1610 rows x 18 columns]
```

[9]:
```
# Removing the outliers
Q1 = dataframe.quantile(0.25)
Q3 = dataframe.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
dataframe[(dataframe >= lower_bound) & (dataframe <= upper_bound)].
 ↪dropna(axis=1)
```

```
/tmp/ipykernel_236/787857994.py:7: FutureWarning: Automatic reindexing on
DataFrame vs Series comparisons is deprecated and will raise ValueError in a
future version. Do `left, right = left.align(right, axis=1, copy=False)` before
e.g. `left == right`
  dataframe[(dataframe >= lower_bound) & (dataframe <=
upper_bound)].dropna(axis=1)
```

[9]:

	Unnamed: 0	danceability	liveness	valence
0	0	0.463	0.9320	0.0302
1	1	0.326	0.9610	0.3180
2	2	0.386	0.9560	0.3130
3	3	0.369	0.8950	0.1470
4	4	0.303	0.9660	0.2060
…	…	…	…	…
1605	1605	0.466	0.3240	0.9670
1606	1606	0.509	0.5160	0.4460
1607	1607	0.790	0.0669	0.8350
1608	1608	0.700	0.1660	0.5320
1609	1609	0.727	0.0965	0.9690

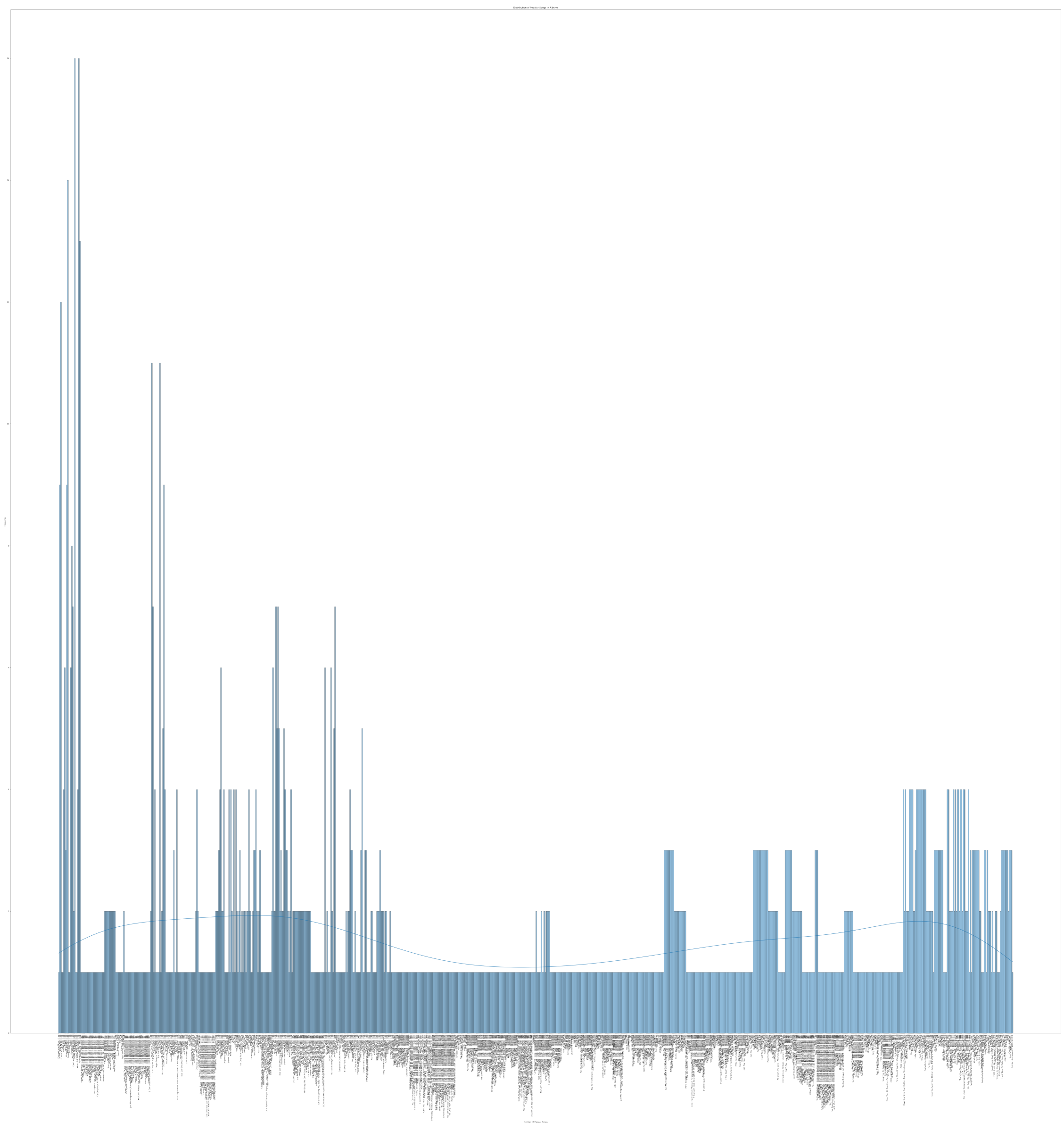
[1610 rows x 4 columns]

2.3 Perform exploratory data analysis to dive deeper into different features of songs and identify the pattern.

[24]:
```
import matplotlib.pyplot as plt
import seaborn as sns


# Summary statistics
print(dataframe.describe())

# Distribution of popular songs
plt.figure(figsize=(100, 100))
sns.histplot(data=dataframe.dropna(axis=1), x='name', bins=100, kde=True)
plt.xlabel('Number of Popular Songs')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
```

```python
plt.title('Distribution of Popular Songs in Albums')
plt.show()
```

```
       Unnamed: 0  track_number  acousticness  danceability       energy  \
count  1610.000000   1610.000000   1610.000000   1610.000000  1610.000000
mean    804.500000      8.613665      0.250475      0.468860     0.792352
std     464.911282      6.560220      0.227397      0.141775     0.179886
min       0.000000      1.000000      0.000009      0.104000     0.141000
25%     402.250000      4.000000      0.058350      0.362250     0.674000
50%     804.500000      7.000000      0.183000      0.458000     0.848500
75%    1206.750000     11.000000      0.403750      0.578000     0.945000
max    1609.000000     47.000000      0.994000      0.887000     0.999000

       instrumentalness    liveness     loudness  speechiness        tempo  \
count       1610.000000  1610.00000  1610.000000  1610.000000  1610.000000
mean           0.164170     0.49173    -6.971615     0.069512   126.082033
std            0.276249     0.34910     2.994003     0.051631    29.233483
min            0.000000     0.02190   -24.408000     0.023200    46.525000
25%            0.000219     0.15300    -8.982500     0.036500   107.390750
50%            0.013750     0.37950    -6.523000     0.051200   124.404500
75%            0.179000     0.89375    -4.608750     0.086600   142.355750
max            0.996000     0.99800    -1.014000     0.624000   216.304000

           valence   popularity    duration_ms
count  1610.000000  1610.000000    1610.000000
mean      0.582165    20.788199  257736.488199
std       0.231253    12.426859  108333.474920
min       0.000000     0.000000   21000.000000
25%       0.404250    13.000000  190613.000000
50%       0.583000    20.000000  243093.000000
75%       0.778000    27.000000  295319.750000
max       0.974000    80.000000  981866.000000
```

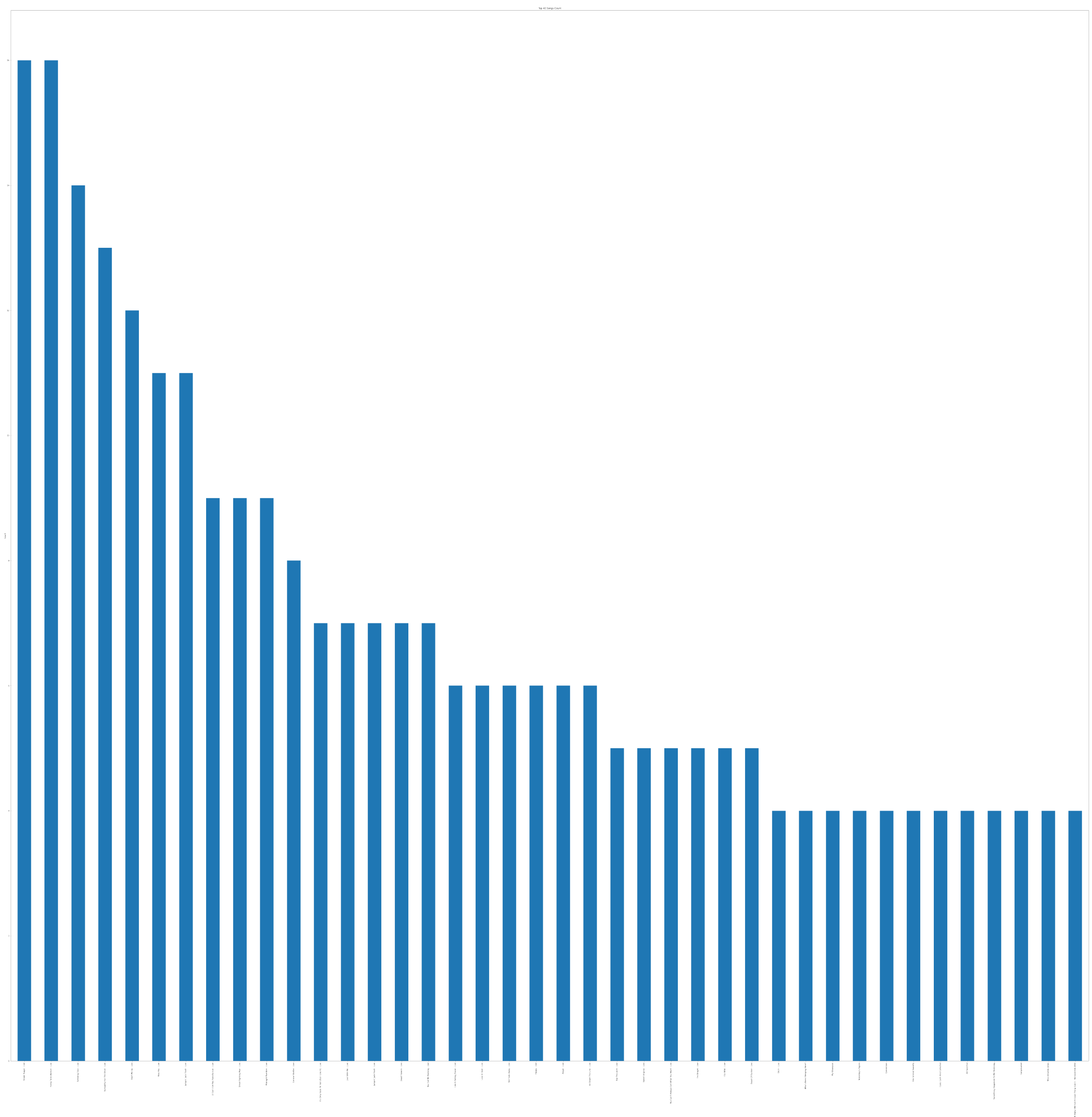


2.4 Perform exploratory data analysis to dive deeper into different features of songs and identify the pattern.

2.5 Bivariate and Multivariate Analysis

[36]:
```
# Bar Plots data analysis

plt.figure(figsize=(100, 100))

artist_counts = dataframe['name'].value_counts()
artist_counts[:40].plot(kind='bar')
```

```
plt.xlabel('Artist')
plt.ylabel('Count')
plt.title('Top 40 Songs Count')
plt.show()
```



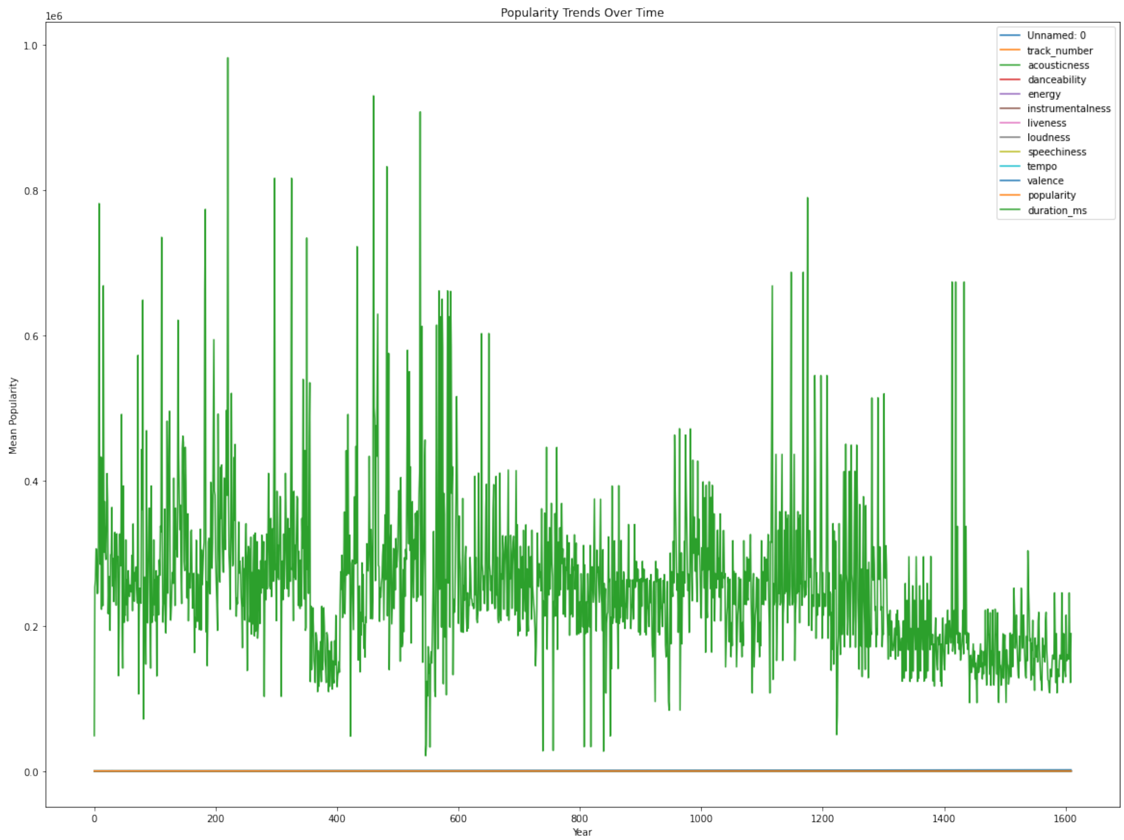
2.6 ****Linear regression to predicting the popularity of a song based on danceability and energy*****

[38]:
```
from statsmodels.formula.api import ols


model = ols('popularity ~ danceability + energy', data=dataframe).fit()
```

```python
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             popularity   R-squared:                       0.020
Model:                            OLS   Adj. R-squared:                  0.019
Method:                 Least Squares   F-statistic:                     16.55
Date:                Sun, 10 Sep 2023   Prob (F-statistic):           7.69e-08
Time:                        13:23:25   Log-Likelihood:                -6324.6
No. Observations:                1610   AIC:                         1.266e+04
Df Residuals:                    1607   BIC:                         1.267e+04
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept       16.0793      2.035      7.902      0.000      12.088      20.070
danceability    11.9473      2.269      5.265      0.000       7.496      16.398
energy          -1.1266      1.788     -0.630      0.529      -4.635       2.381
==============================================================================
Omnibus:                      180.283   Durbin-Watson:                   0.609
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              316.500
Skew:                           0.744   Prob(JB):                     1.87e-69
Kurtosis:                       4.582   Cond. No.                         13.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[48]:
```python
# Visualization
dataframe.plot(kind='line', figsize=(20, 15))
plt.xlabel('Year')
plt.ylabel('Mean Popularity')
plt.title('Popularity Trends Over Time')
plt.show()
```



2.7 ****Comment on the importance of dimensionality reduction techniques, share your ideas and explain your observations.****

2.8 Reduced-dimensional representations are often more interpretable, making it easier to extract meaningful insights from the data.

2.9 ****Perform Cluster Analysis: Identify the right number of clusters , Use appropriate clustering algorithm, Define each cluster based on the features****

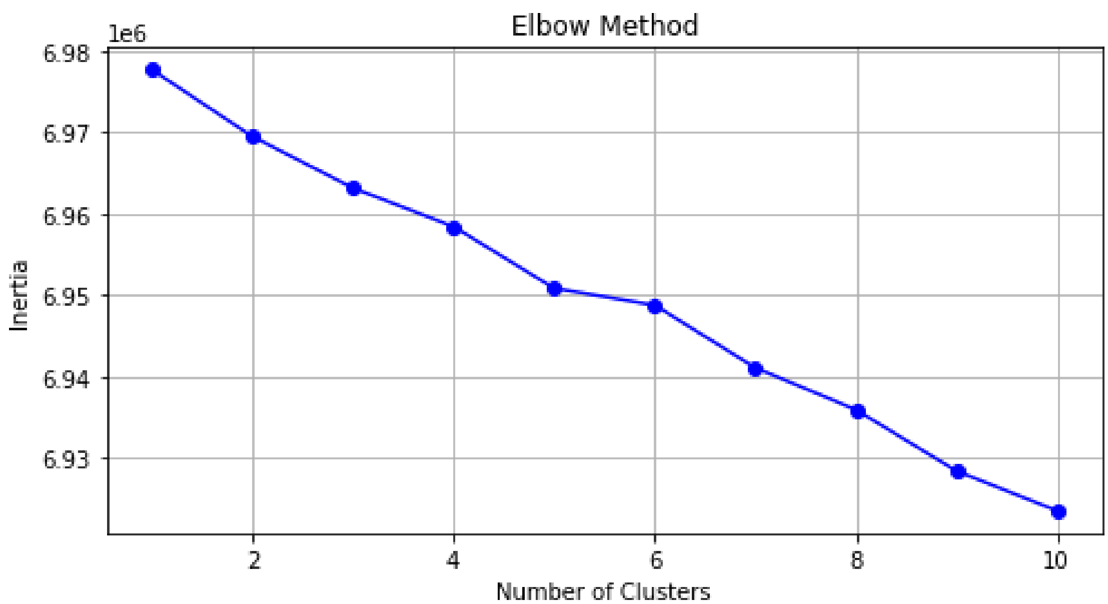
[64]:
```
df_encoded = pd.get_dummies(dataframe, column)
```

[65]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


numeric_columns = dataframe.select_dtypes(include=['number'])
categorical_columns = dataframe.select_dtypes(exclude=['number'])

categorical_encoded = pd.get_dummies(categorical_columns)
```

```python
preprocessed_data = pd.concat([numeric_columns, categorical_encoded], axis=1)


scaler = StandardScaler()
scaled_data = scaler.fit_transform(preprocessed_data)


inertia = []


for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    inertia.append(kmeans.inertia_)


plt.figure(figsize=(8, 4))
plt.plot(range(1, 11), inertia, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
```

```
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



[ ]: