Exploration-Old-Songs

November 27, 2021

1 EDA for old song list output vars

[1]:
```
import re
import string
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy
%matplotlib inline

import pylab
import scipy.stats as stats
from fitter import Fitter, get_common_distributions, get_distributions

#nltk imports
import nltk
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

import warnings
import scipy.stats as st
import statsmodels.api as sm
from scipy.stats._continuous_distns import _distn_names
import matplotlib
import matplotlib.pyplot as plt

matplotlib.rcParams['figure.figsize'] = (16.0, 12.0)
matplotlib.style.use('ggplot')
```

[2]:
```
%%javascript
IPython.OutputArea.auto_scroll_threshold = 9999;
```

```
<IPython.core.display.Javascript object>
```

[3]:
```
#create a df from csv
```

```
filename = "/Users/gautham/Documents/Documents - gBookPro/Berkeley MIMS/
↪Semester 1/256 - ANLP/anlp21-project/TopicModeling/data/full_df.csv"
lyrics_df= pd.read_csv(filename)
lyrics_df.rename(columns={ 'Unnamed: 0': 'track_name'}, inplace=True)
lyrics_df.drop('Unnamed: 0.1', inplace=True, axis=1)
lyrics_df.head()
```

[3]:

```
            track_name  playlist_name            playlist_id playlist_genre  \
0         Back In Black  Rock Classics  37i9dQZF1DWXRqgorJj26U           Rock
1         Paradise City  Rock Classics  37i9dQZF1DWXRqgorJj26U           Rock
2              Dream On  Rock Classics  37i9dQZF1DWXRqgorJj26U           Rock
3                 Creep  Rock Classics  37i9dQZF1DWXRqgorJj26U           Rock
4  Don't Stop Believin'  Rock Classics  37i9dQZF1DWXRqgorJj26U           Rock

                 track_id track_artist_name         track_artist_id  \
0  08mG3Y1vljYA6bvDt4Wqkj             AC/DC  711MCceyCBcFnzjGY4Q7Un
1  3YBZIN3rekqsKxbJc9FZko     Guns N' Roses  3qm84nBOXUEQ2vnTfUTTFC
2  5MxNLUsfh7uzROypsoO5qe         Aerosmith  7Ey4PD4MYsKc5I2dolUwbH
3  70LcF31zb1H0PyJoS1Sx1r         Radiohead  4Z8W4fKeB5YxbusRsdQVPb
4  4bHsxqR3GMrXTxEPLuK5ue           Journey  0rvjqX7ttXeg3mTy8Xscbt

   danceability  energy  key  loudness  mode  acousticness  valence   tempo  \
0         0.310   0.700    9    -5.678     1        0.0110    0.763  188.386
1         0.273   0.952   11    -8.762     1        0.0169    0.472  100.271
2         0.307   0.433    1   -10.057     1        0.3880    0.224  160.900
3         0.515   0.430    7    -9.935     1        0.0097    0.104   91.844
4         0.500   0.748    4    -9.072     1        0.1270    0.514  118.852

                                                tabs  \
0  E,D,A/C#,E,D,A/C#,E,D,A/C#,E,D,A/C#,E,D,A/C#,A…
1  G,C,F,C,G,G5,F5,C5,Bb5,C5,C5,Bb5,G,F,G,G,G,C,C…
2  Fm,Fm6,Bbm6,Fm,C7sus,Fm,Fm,Fm7,Fm6,Bbm6,Fm,Fm7…
3  G,B,C,Cm,G,B,C,Cm,G,B,C,Cm,G,B,C,Cm,G,B,C,Cm,G…
4  E,B,C#m,A,E,B,G#m,A,E,B,C#m,A,E,B,G#m,A,E,B,C#…

                                        dirty_lyrics
0  [Verse 1]\nBack in black, I hit the sack\nI've…
1  [Chorus]\nTake me down to the Paradise City\nW…
2  [Verse 1]\nEvery time that I look in the mirr…
3  [Verse 1]\nWhen you were here before\nCouldn't…
4  [Verse 1]\nJust a small-town girl\nLivin' in a…
```

[4]:

```
#use length to filter out non-lyrics
lyrics_df['len'] = None
lyrics_df['len'] = lyrics_df['dirty_lyrics'].apply(lambda x: len(str(x)))
lengths = list(lyrics_df['len'])
percentiles = [90, 91, 92, 93, 94, 95, 99]
```

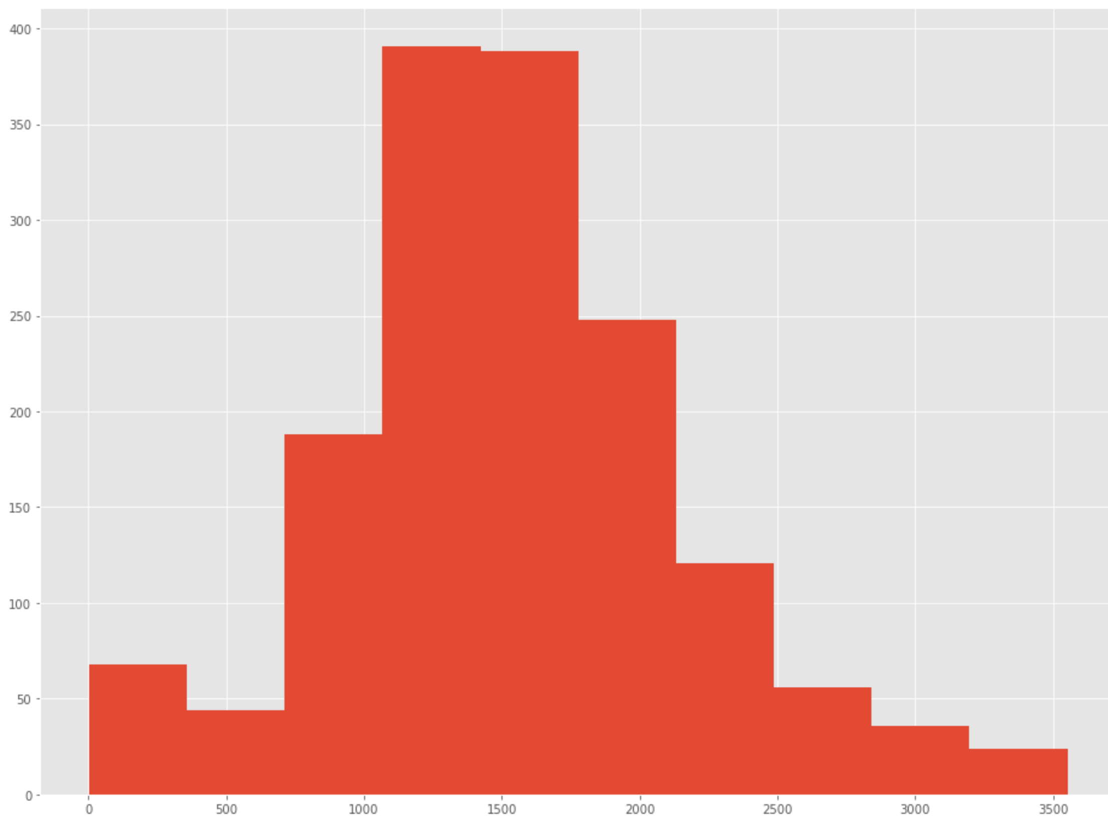
```python
for p in percentiles:
    print(np.percentile(lengths, p))
```

```
3092.0
3246.0
3580.0
3853.0
4322.0
5001.0
89514.0
```

[5]:
```python
ldf = lyrics_df.loc[lyrics_df.len < np.percentile(lengths, 92)]
print(len(ldf))
print(len(lyrics_df))
plt.hist(ldf.len, bins=10)
```

```
1564
1701
```

[5]:
```
(array([ 68.,  44., 188., 391., 388., 248., 121.,  56.,  36.,  24.]),
 array([3.0000e+00, 3.5780e+02, 7.1260e+02, 1.0674e+03, 1.4222e+03,
        1.7770e+03, 2.1318e+03, 2.4866e+03, 2.8414e+03, 3.1962e+03,
        3.5510e+03]),
 <BarContainer object of 10 artists>)
```



[6]:
```
#drop NAs from lyrics col
print(ldf.shape)
ldf = ldf[ldf['dirty_lyrics'].notna()]
ldf.reset_index(drop=True, inplace=True)
print(ldf.shape)
ldf = ldf.drop(ldf.index[776])
print(ldf.shape)
```

```
(1564, 18)
(1496, 18)
(1495, 18)
```

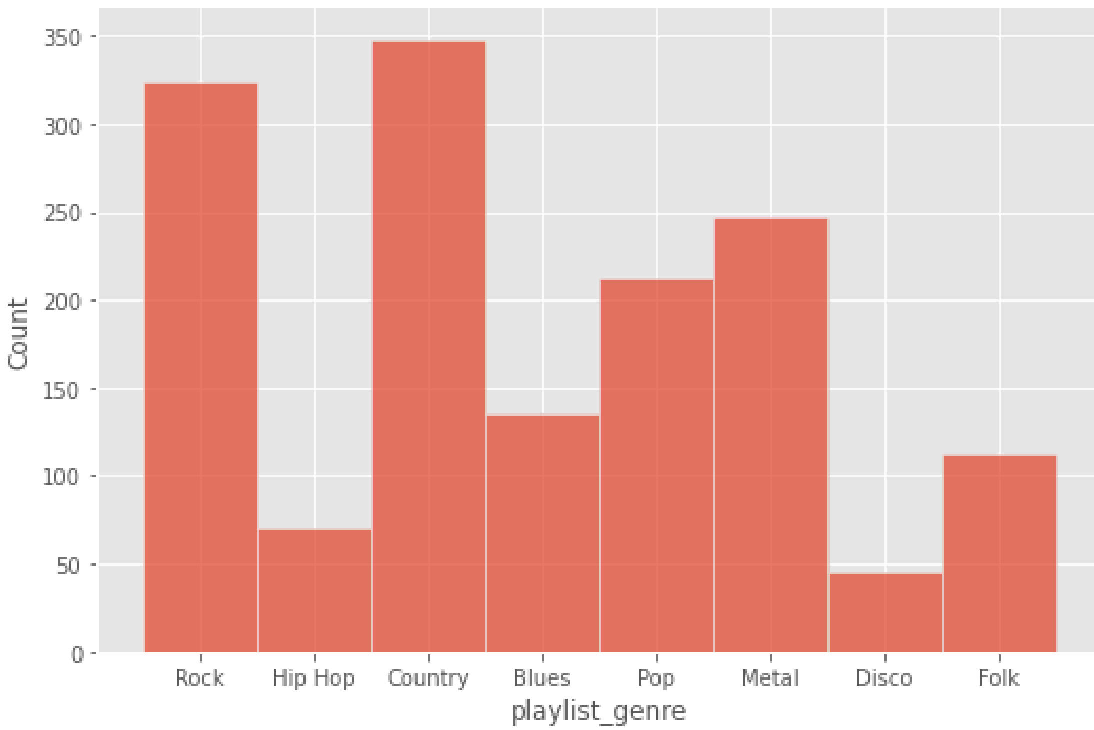
2 Plotting Genre Distribution in Dataset

[7]:
```
ldf.columns
```

[7]:
```
Index(['track_name', 'playlist_name', 'playlist_id', 'playlist_genre',
       'track_id', 'track_artist_name', 'track_artist_id', 'danceability',
       'energy', 'key', 'loudness', 'mode', 'acousticness', 'valence', 'tempo',
       'tabs', 'dirty_lyrics', 'len'],
      dtype='object')
```

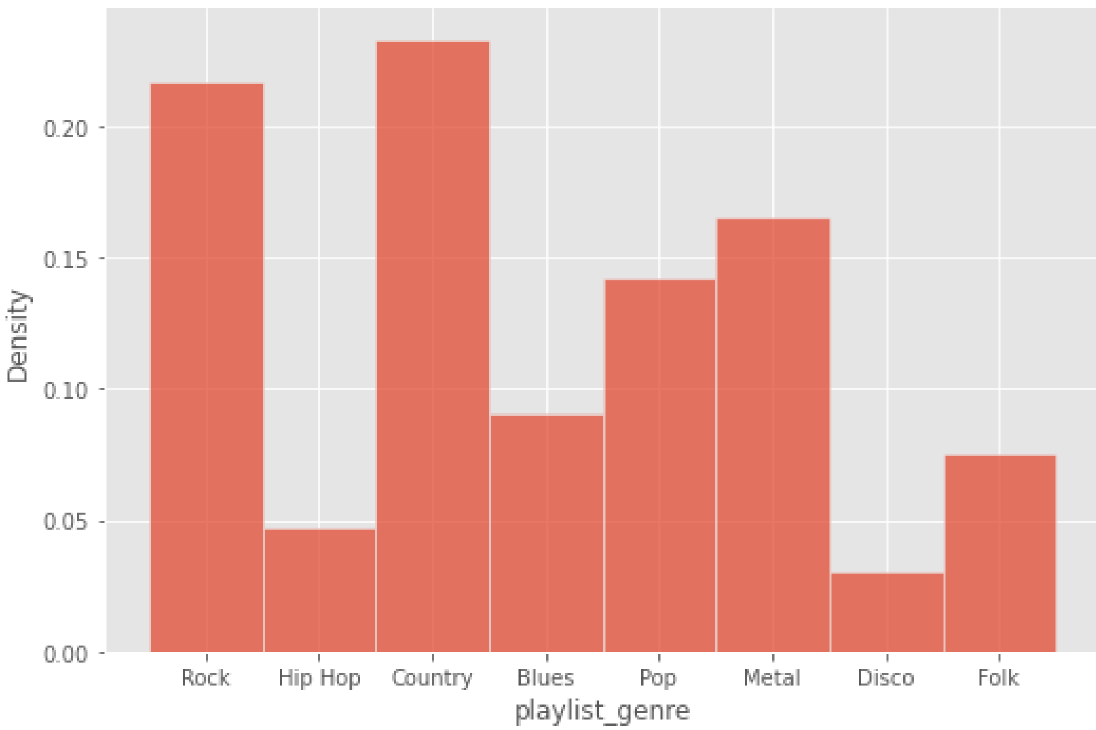
[8]:
```
sns.displot(data=ldf, x='playlist_genre', aspect=1.5, stat='count')
```

[8]:
```
<seaborn.axisgrid.FacetGrid at 0x7f97b1d299a0>
```



[9]:
```
sns.displot(data=ldf, x='playlist_genre', aspect=1.5, stat='density')
```

[9]: <seaborn.axisgrid.FacetGrid at 0x7f97d0917070>



3 Plotting predicted variable distributions

Source: 1. https://towardsdatascience.com/10-examples-to-master-distribution-plots-with-python-seaborn-4ea2ceea906a 2. https://www.c-sharpcorner.com/article/a-complete-python-seaborn-tutorial/

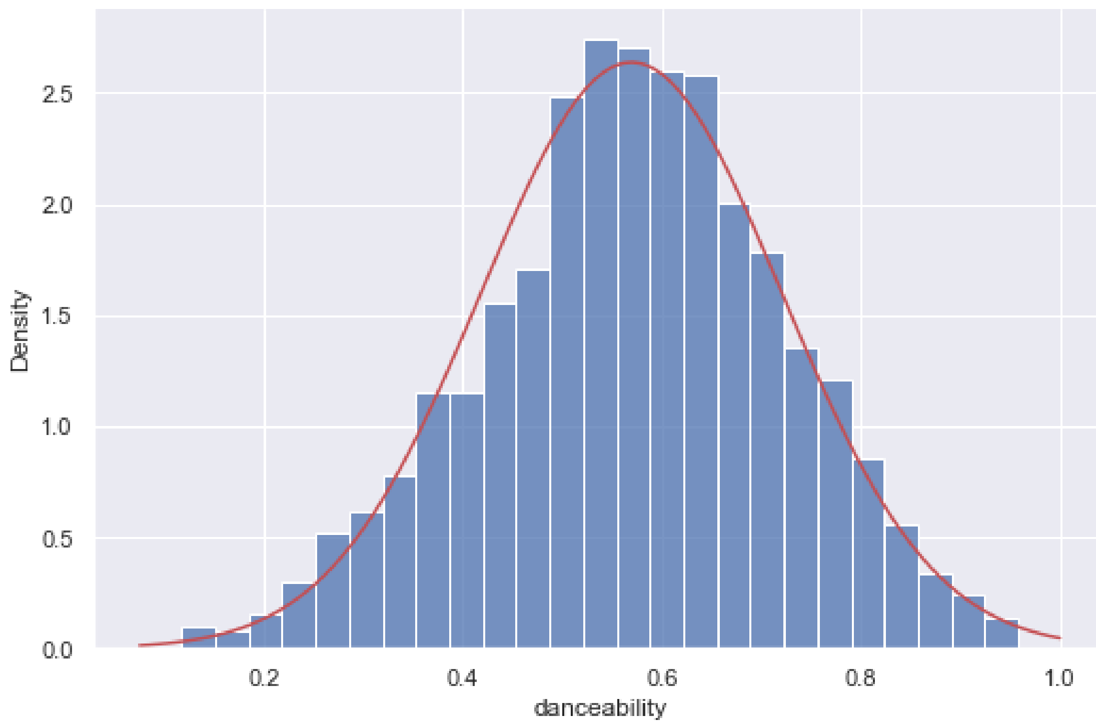
[10]:
```
vars = ldf.columns.to_list()
vars = vars[7:15]
vars
```

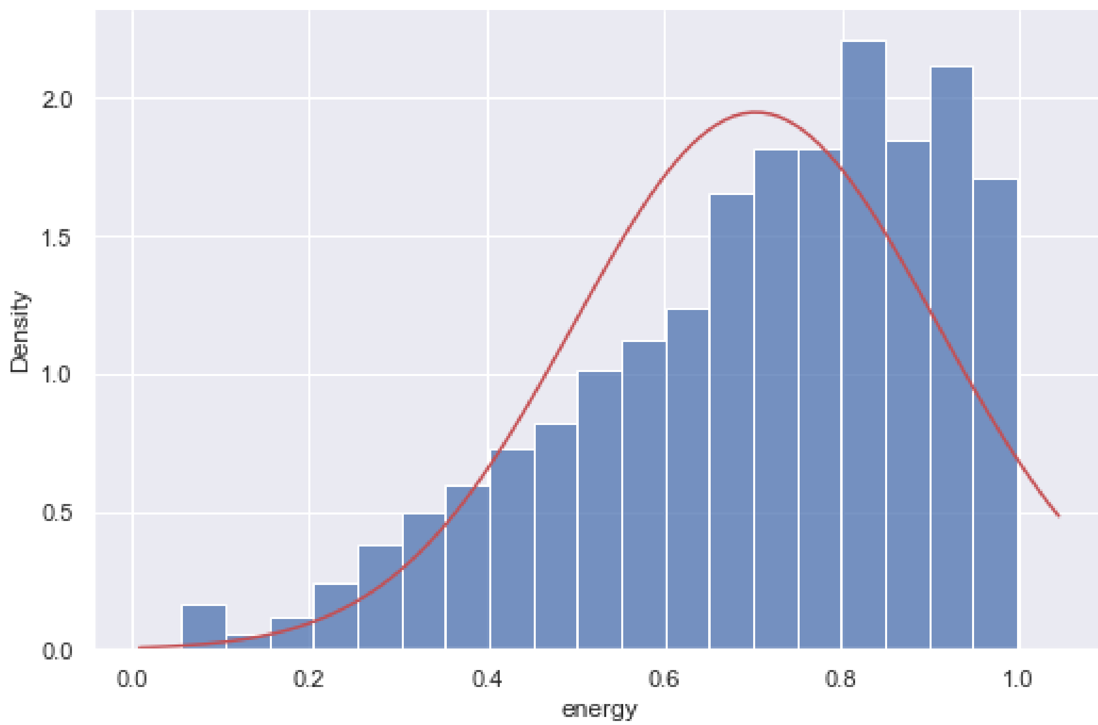
[10]:
```
['danceability',
 'energy',
 'key',
 'loudness',
 'mode',
 'acousticness',
 'valence',
 'tempo']
```

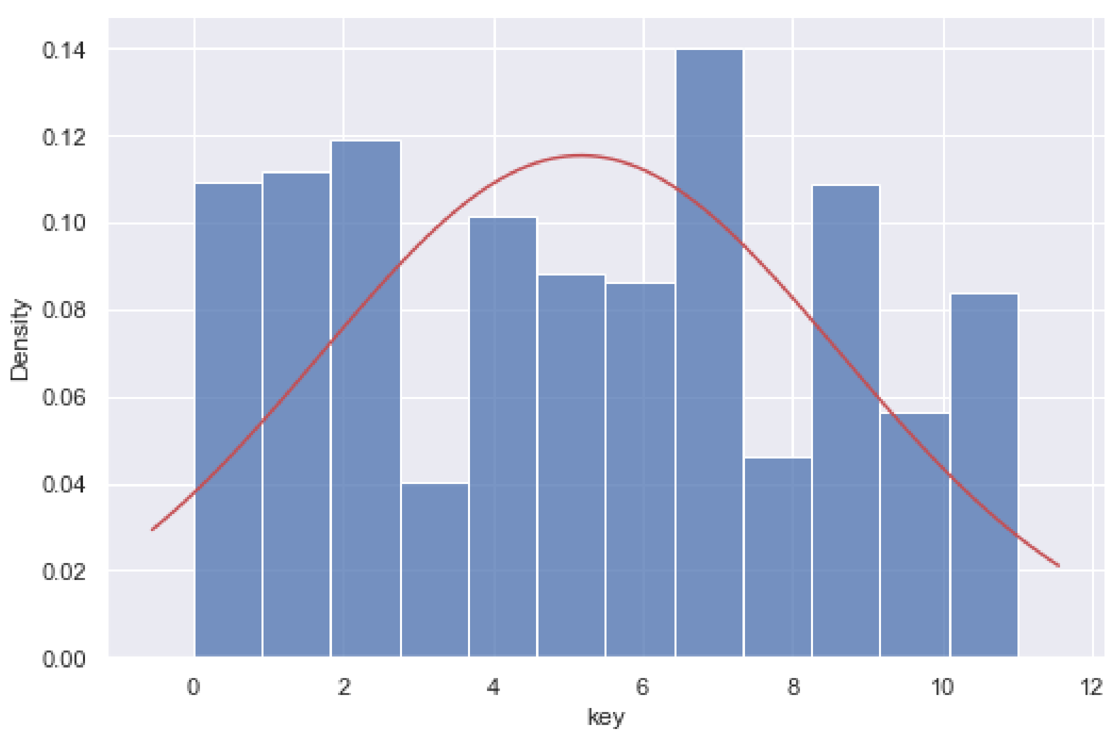
3.0.1 Histogram

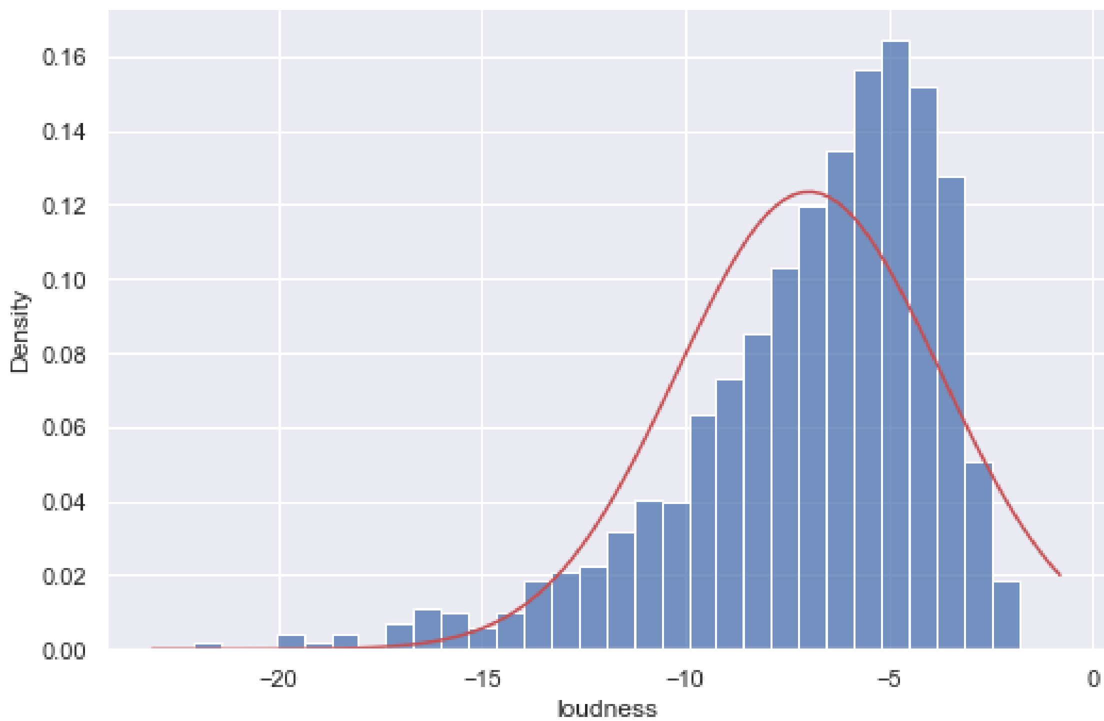
[11]:
```
#https://stackoverflow.com/questions/69059121/
↪how-to-draw-a-normal-curve-on-seaborn-displot
def map_pdf(x, **kwargs):
    mu, std = scipy.stats.norm.fit(x)
    x0, x1 = p1.axes[0][0].get_xlim()  # axes for p1 is required to determine␣
↪x_pdf
    x_pdf = np.linspace(x0, x1, 100)
    y_pdf = scipy.stats.norm.pdf(x_pdf, mu, std)
    plt.plot(x_pdf, y_pdf, c='r')
```

[12]:
```
sns.set_theme()
for var in vars:
    p1 = sns.displot(data=ldf, x=var, aspect=1.5, stat='density')
    p1.map(map_pdf, var)
```

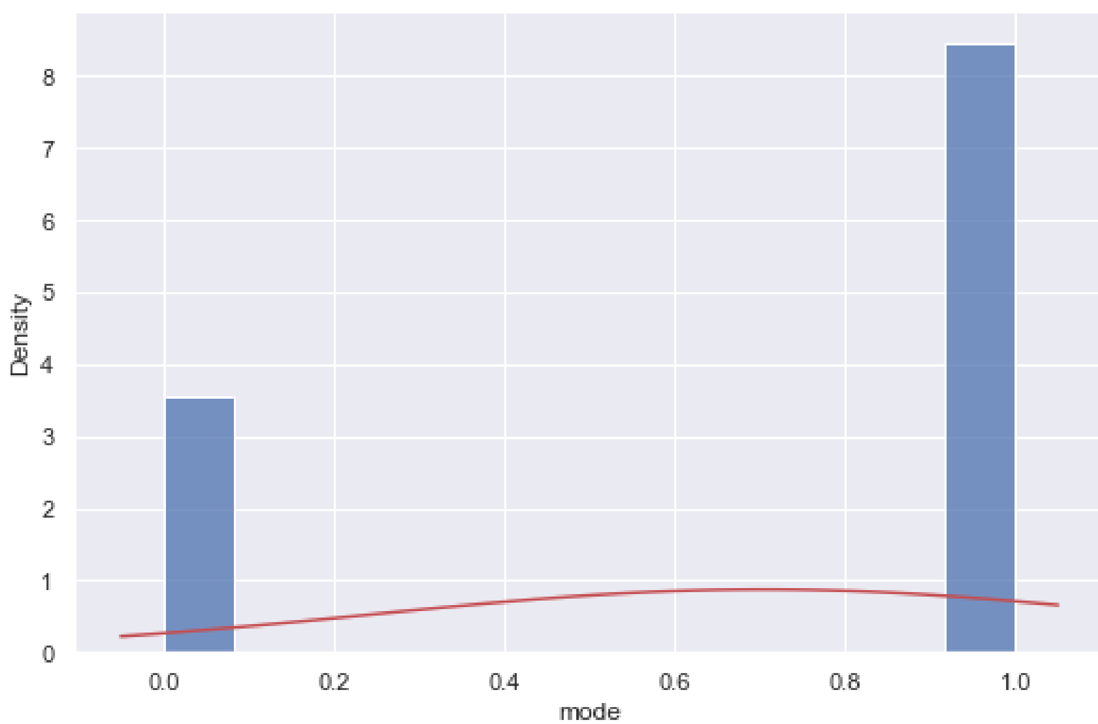



2.0
1.5
1.0
0.5
0.0
Density
0.0
0.2
0.4
0.6
0.8
1.0
energy

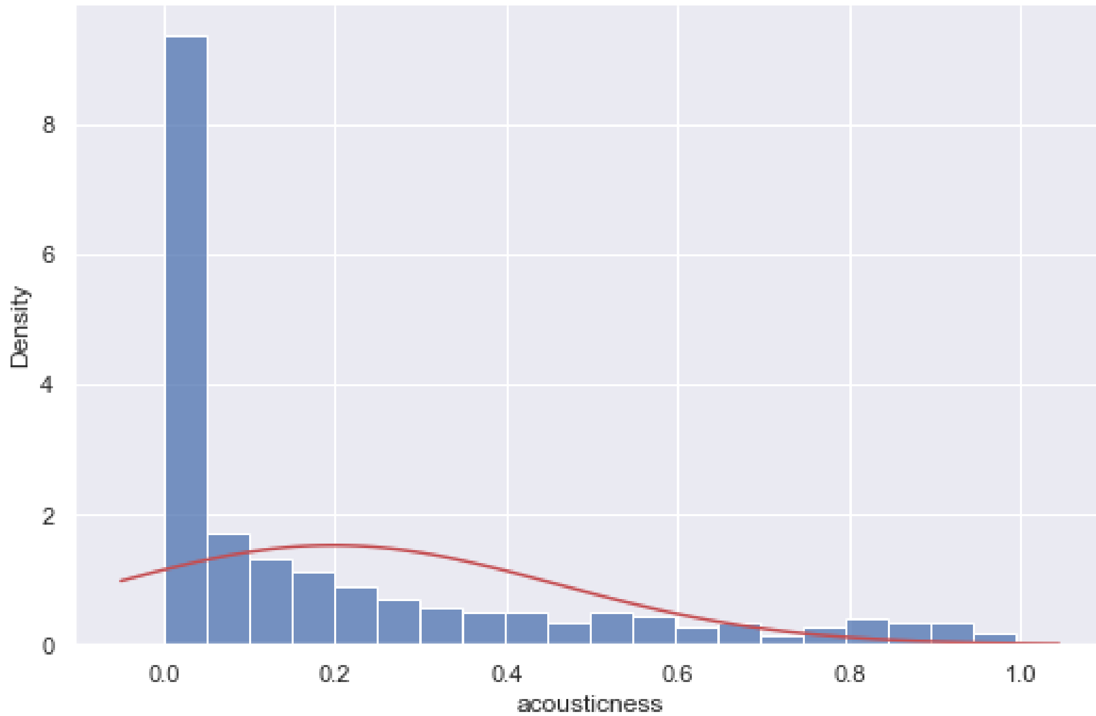

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
0
2
4
6
8
10
12
key

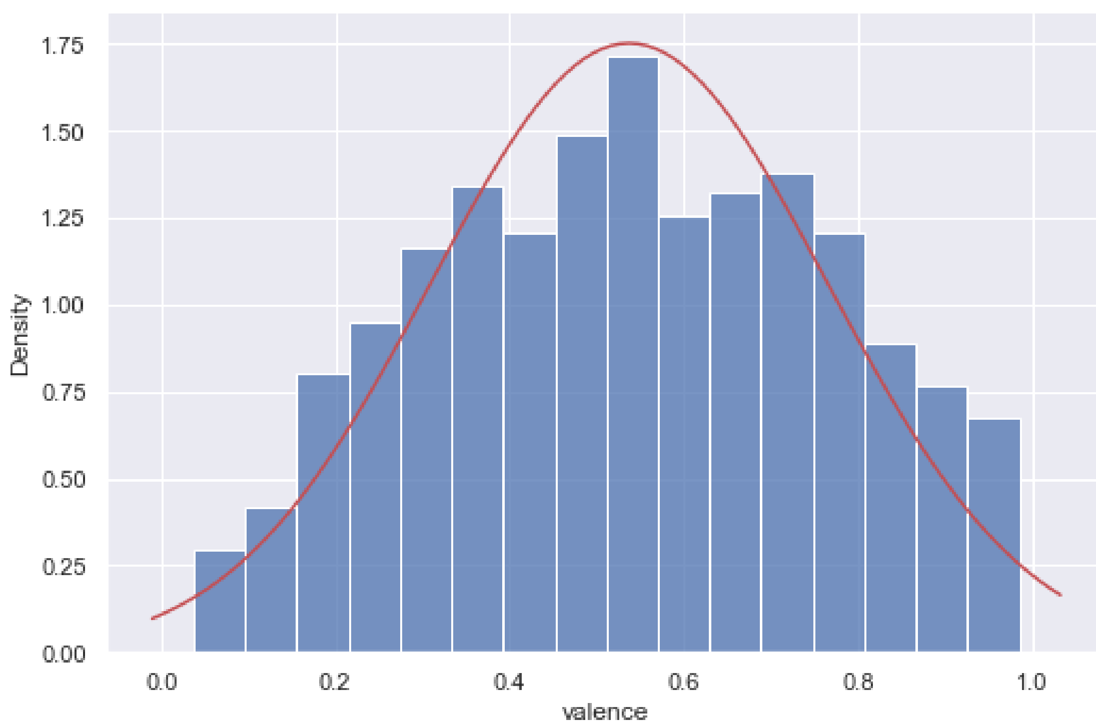


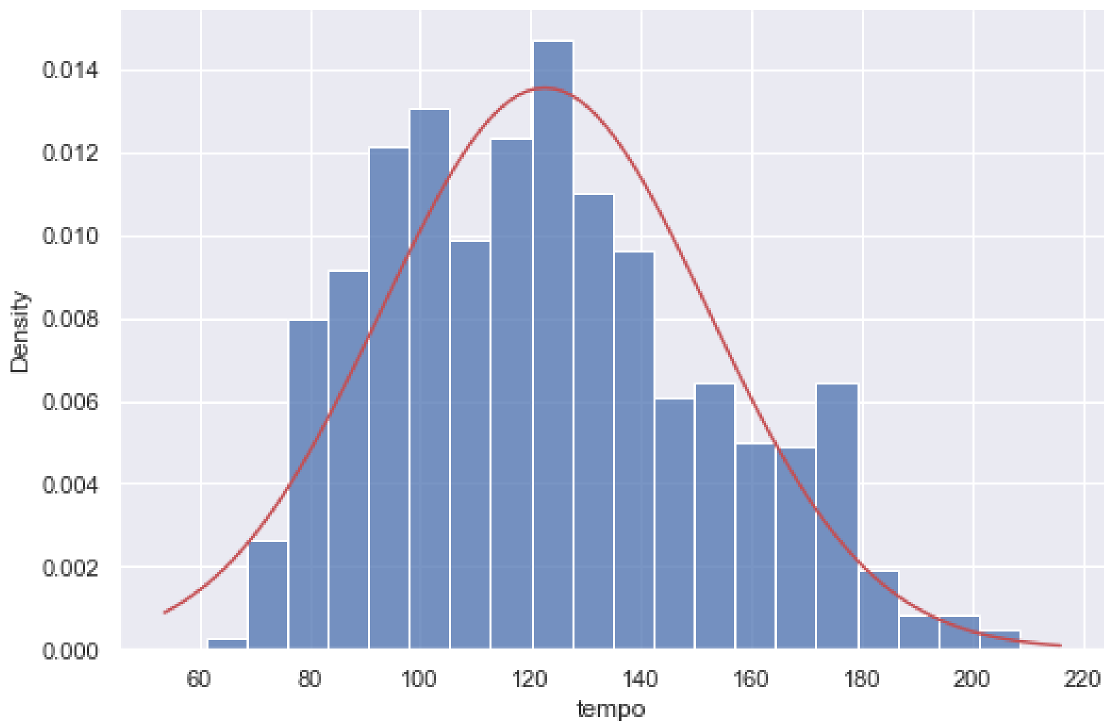
Density
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−20
−15
−10
−5
0
loudness



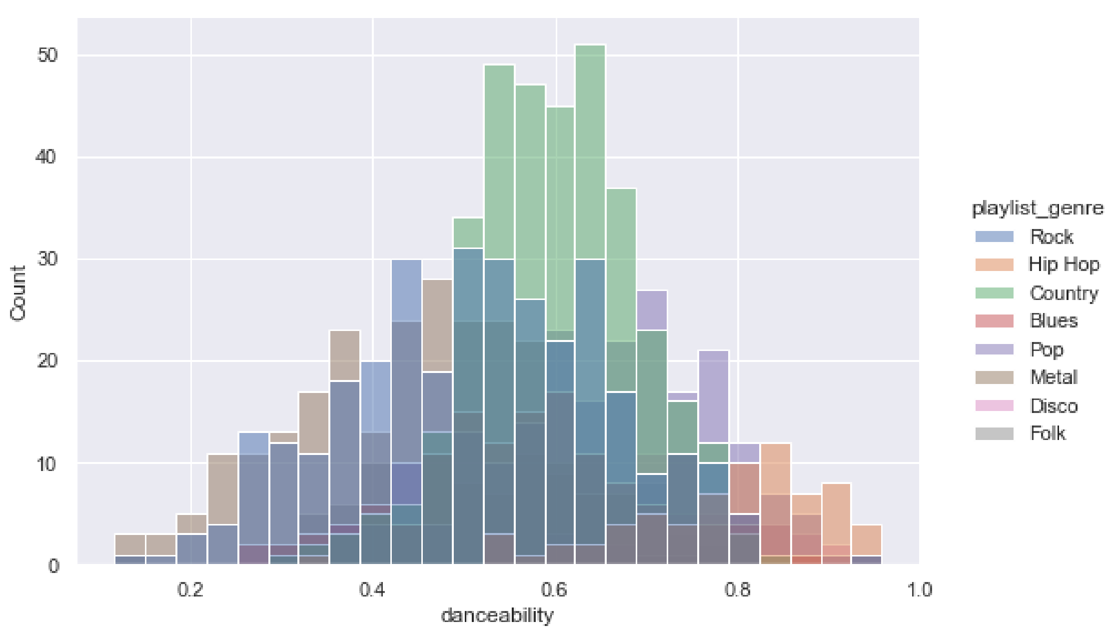
Density
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
mode

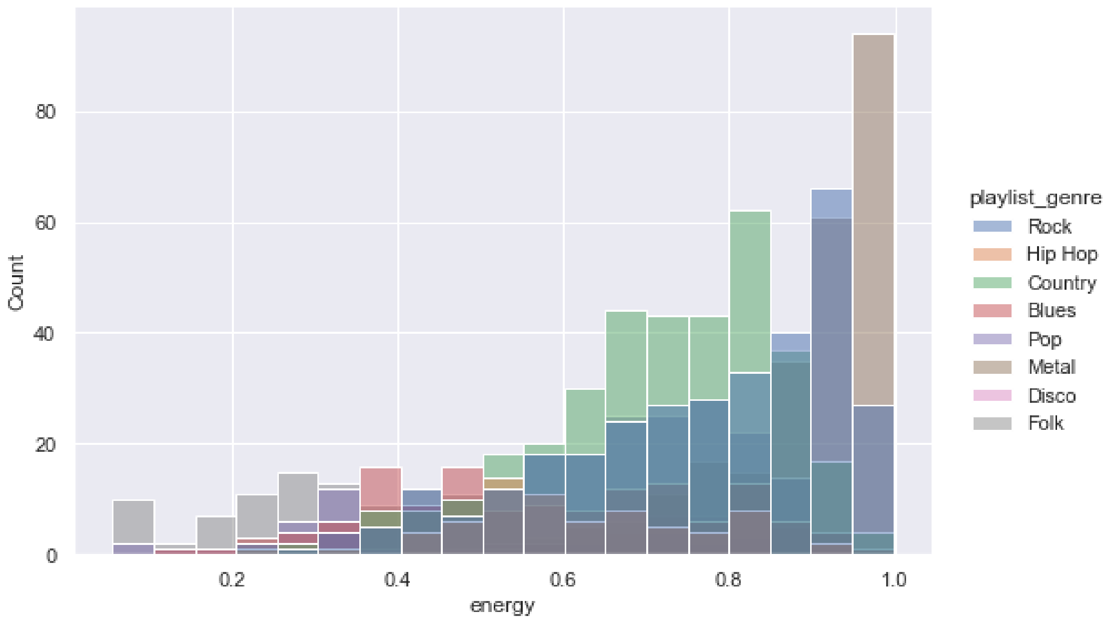

Density
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0
acousticness


Density
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
valence

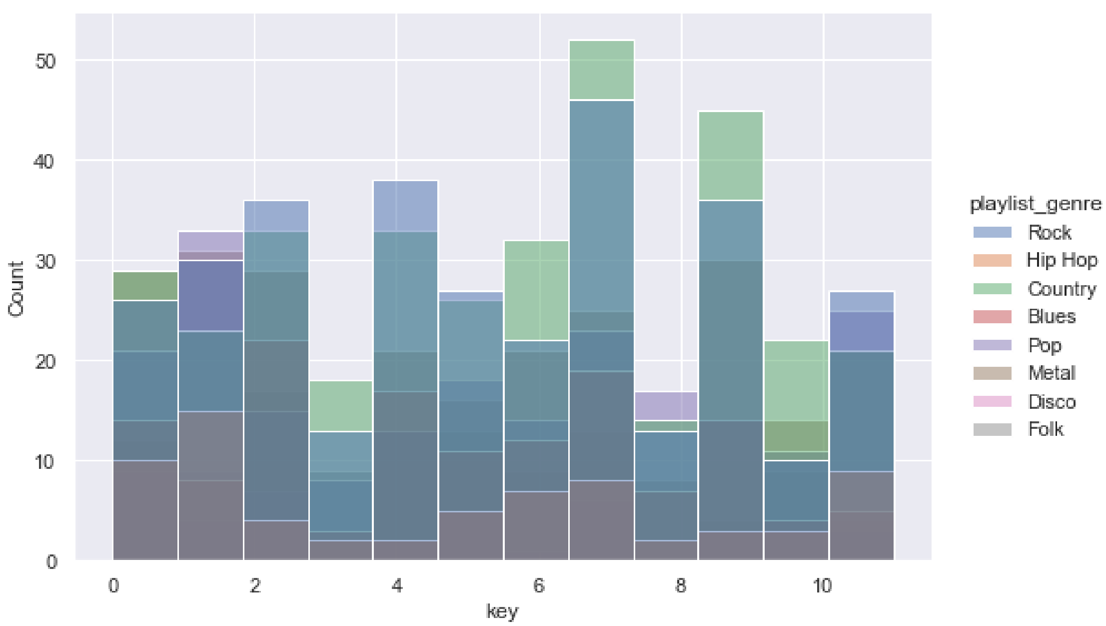


[13]:
```
for var in vars:
    sns.displot(data=ldf, x=var, hue='playlist_genre', kind='hist', aspect=1.5)
```

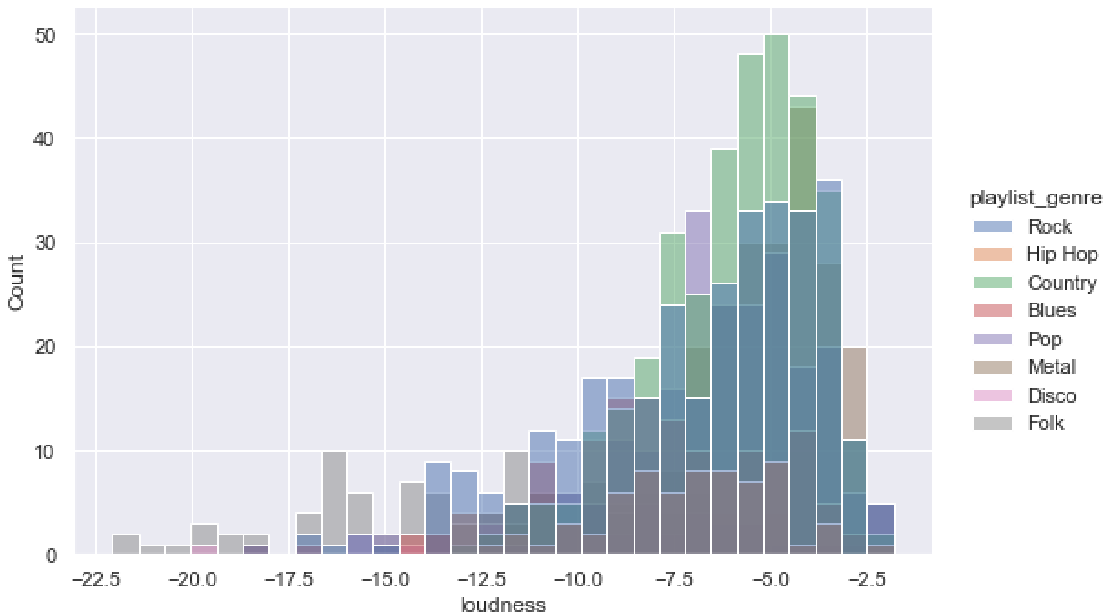




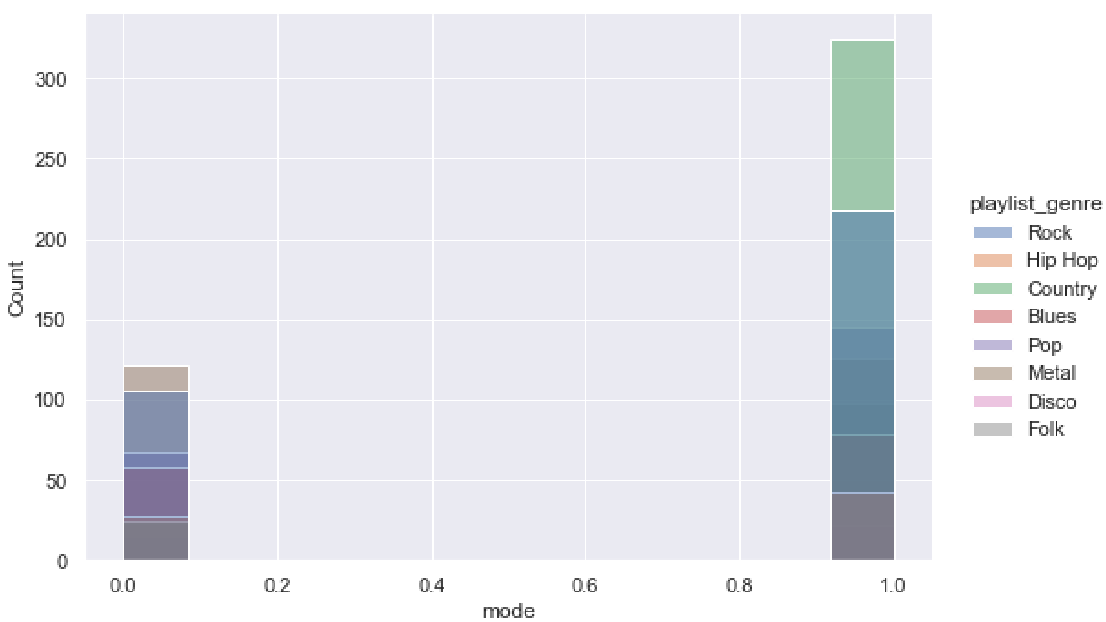
Count
energy
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk



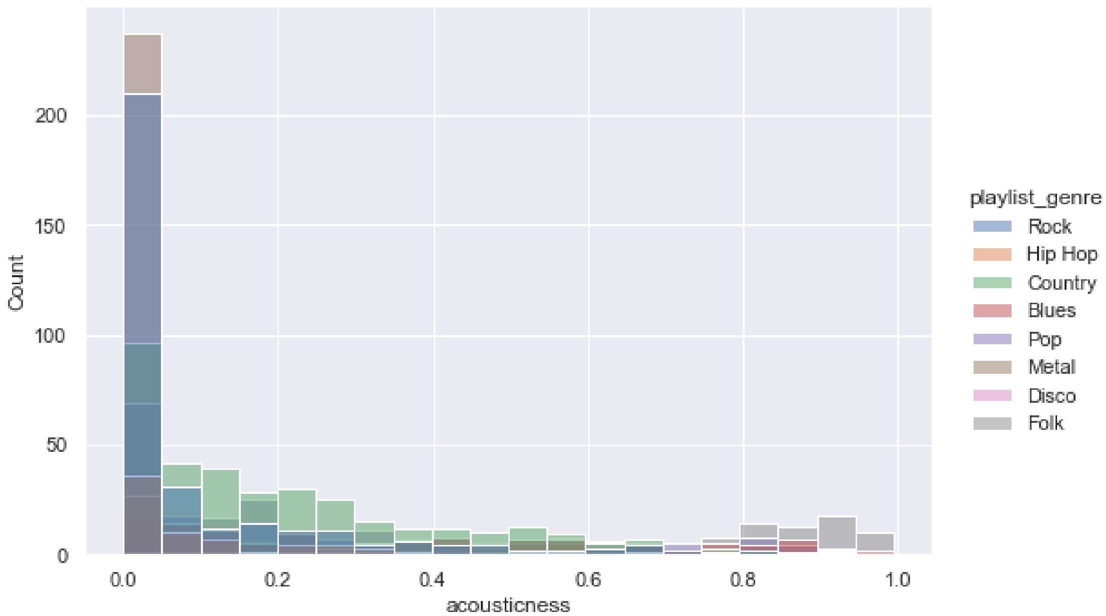
Count
key
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

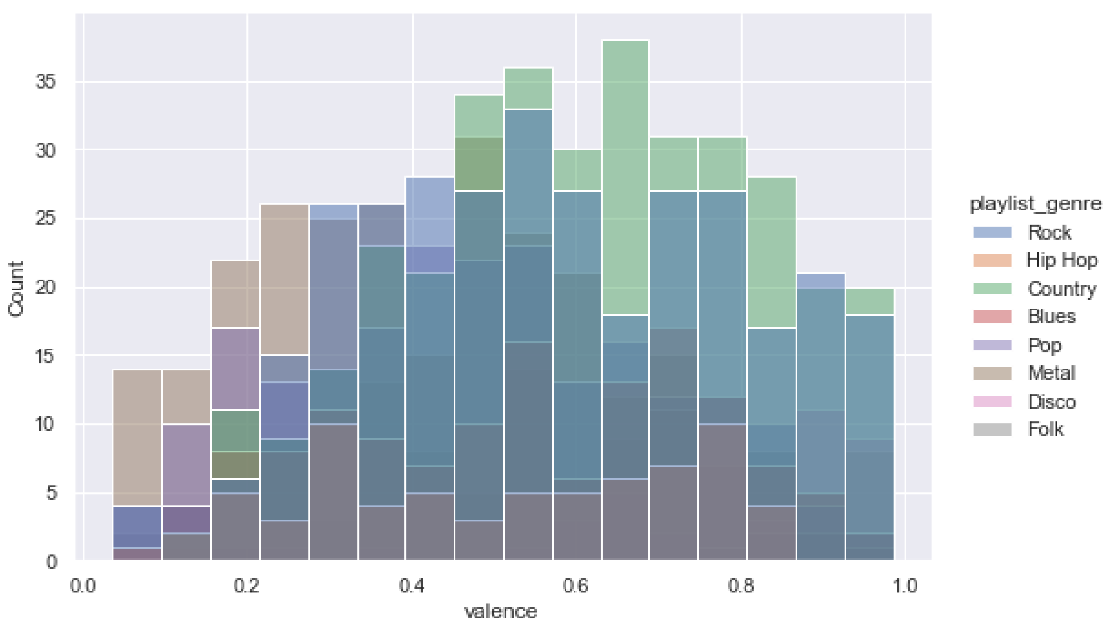


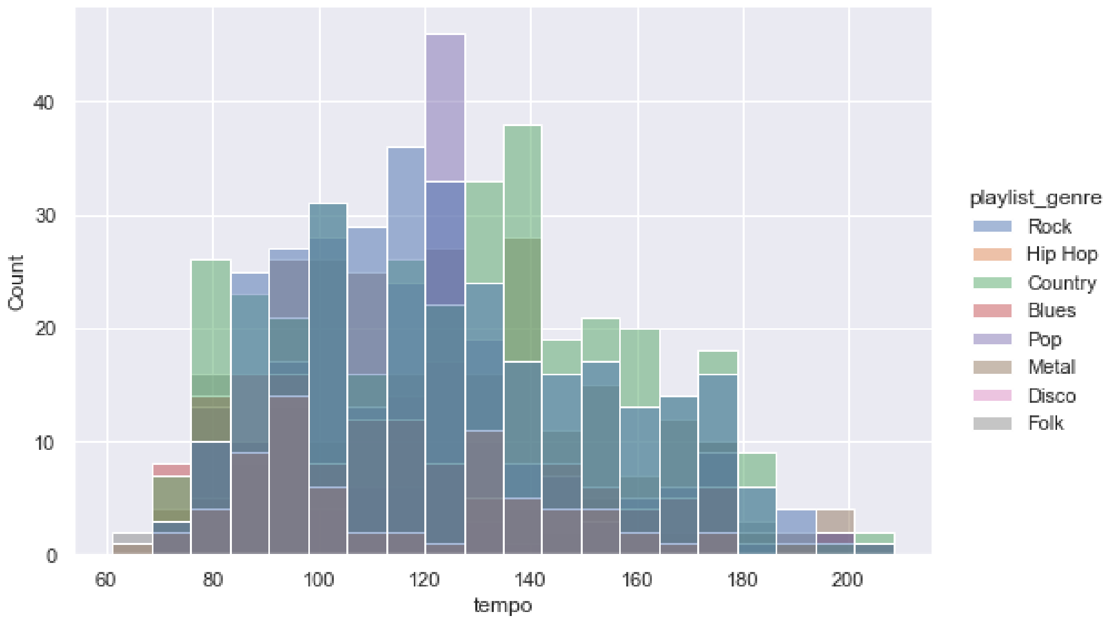
Count
loudness
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk



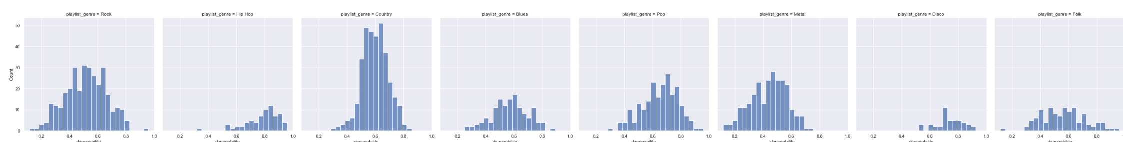
Count
mode
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

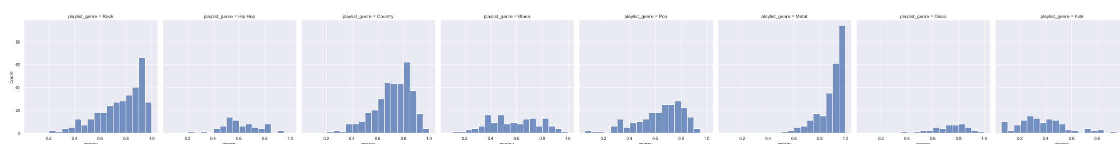

Count
acousticness
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

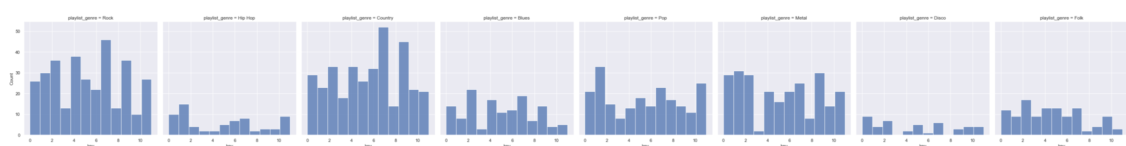

Count
valence
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

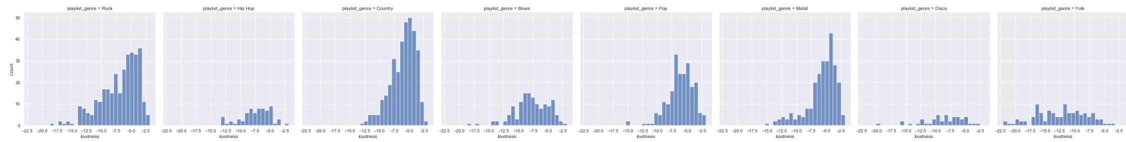


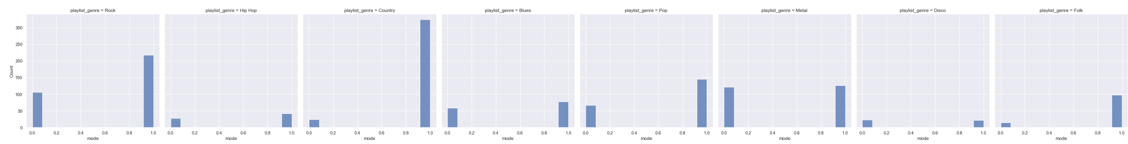
[14]:
```
for var in vars:
    sns.displot(data=ldf, x=var, col='playlist_genre', kind='hist', aspect=1)
```

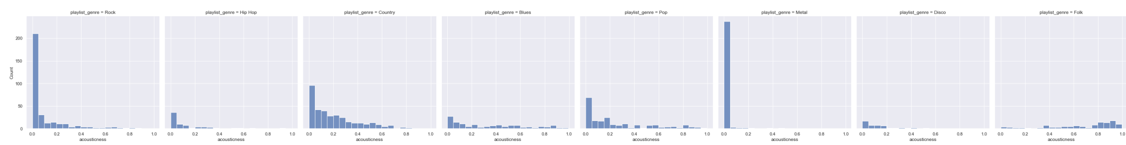


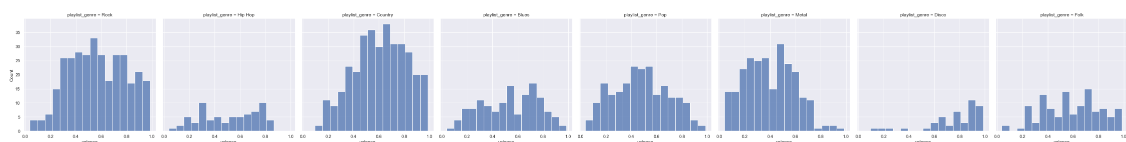


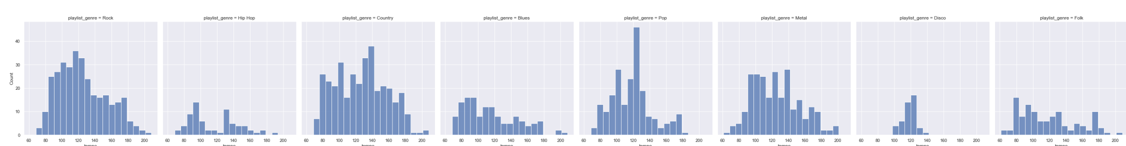




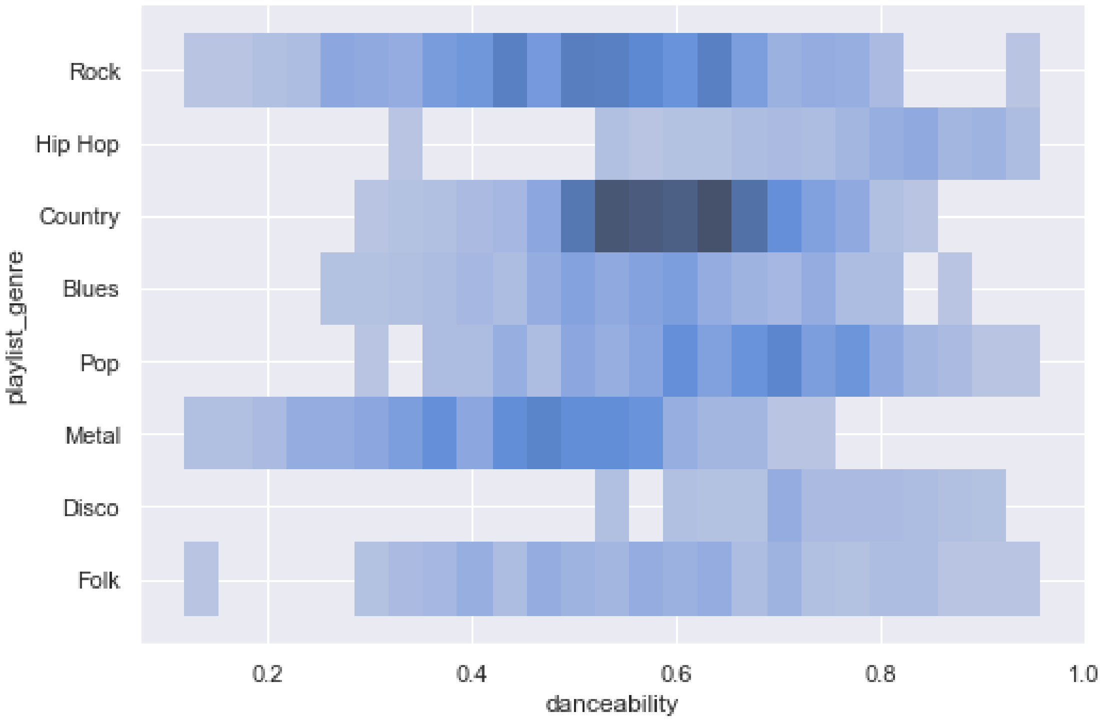




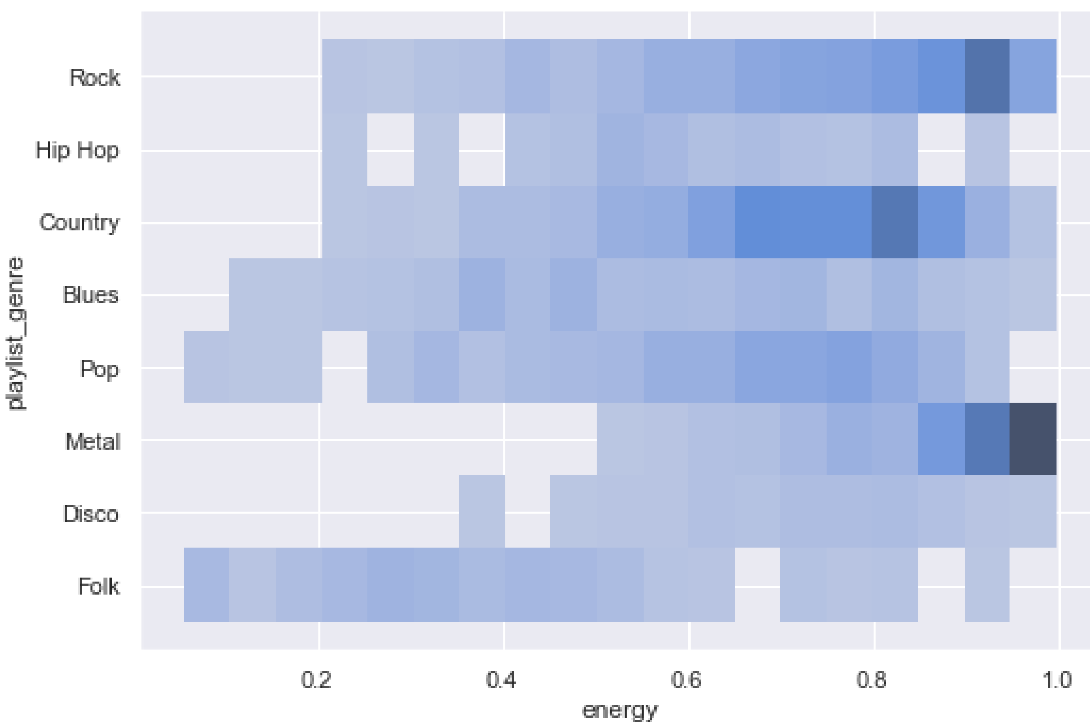




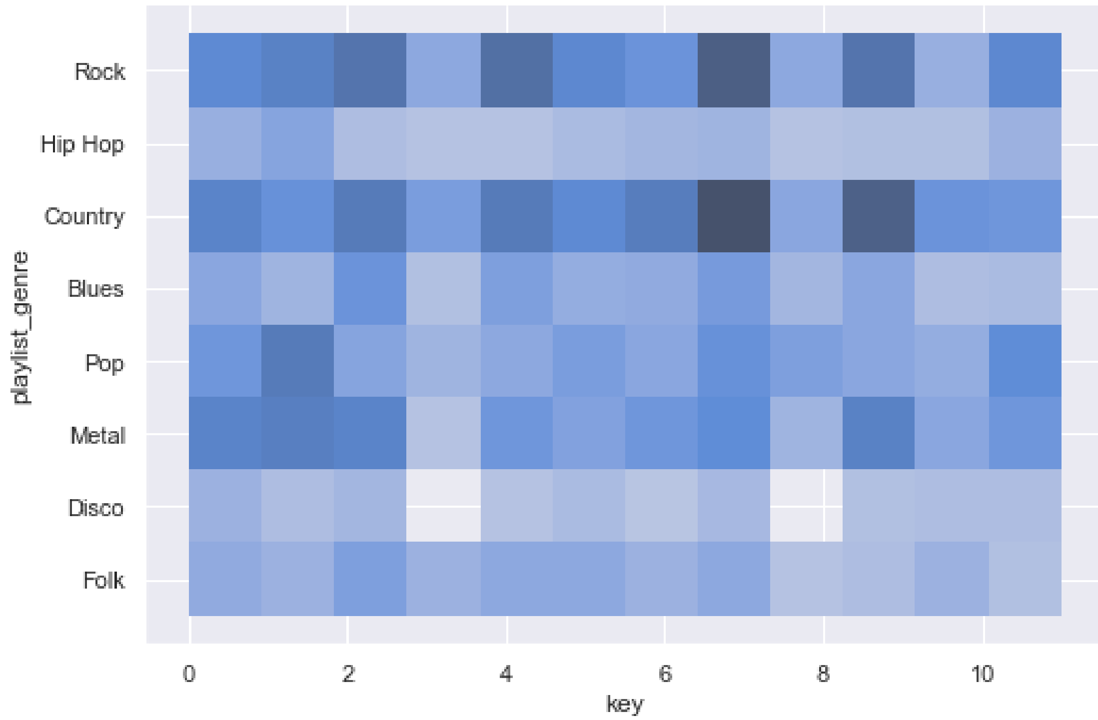
[15]:
```
for var in vars:
    sns.displot(data=ldf, x=var, y='playlist_genre', kind='hist', aspect=1.5)
```



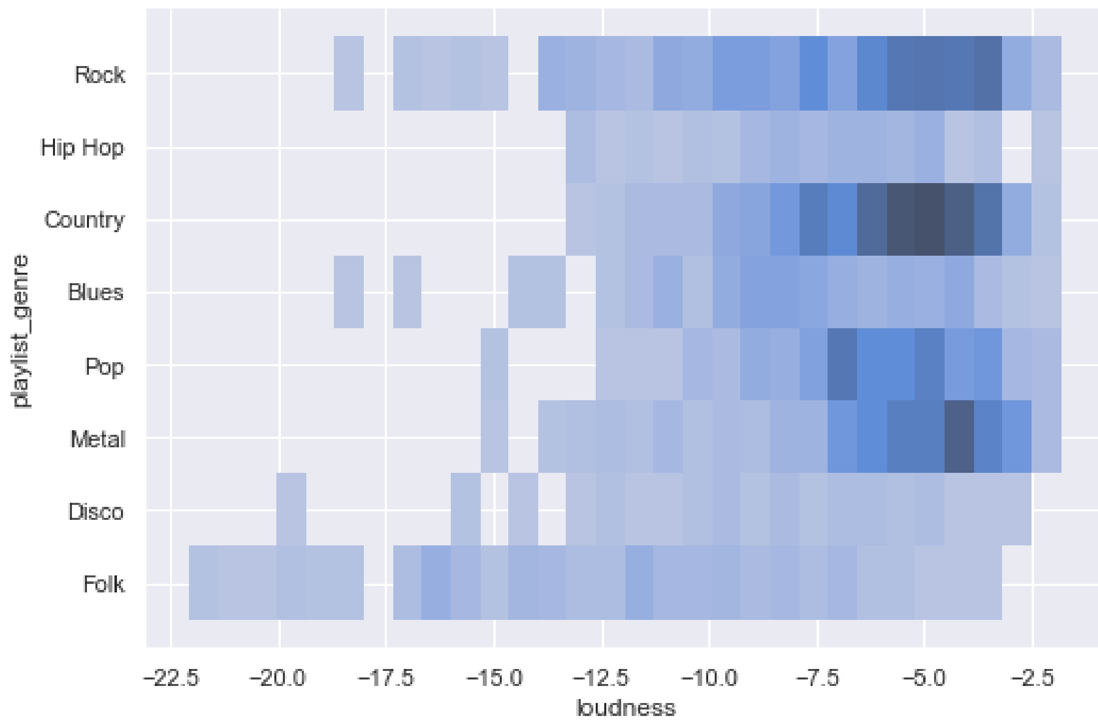
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
0.2
0.4
0.6
0.8
1.0
danceability



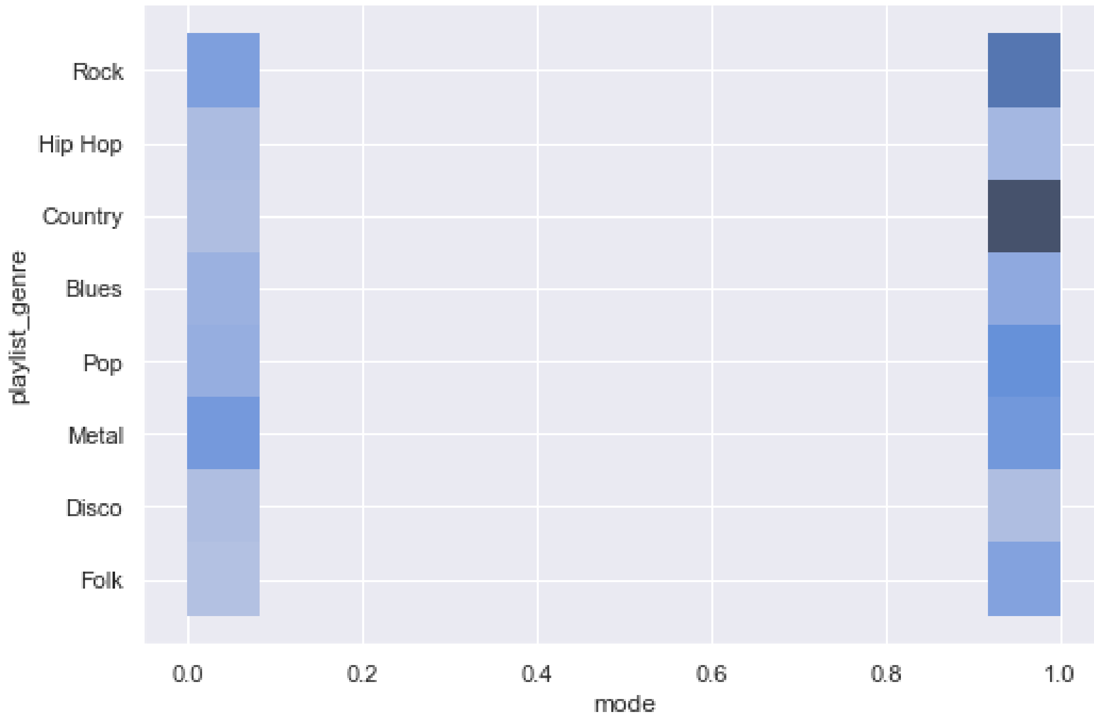
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
0.2
0.4
0.6
0.8
1.0
energy

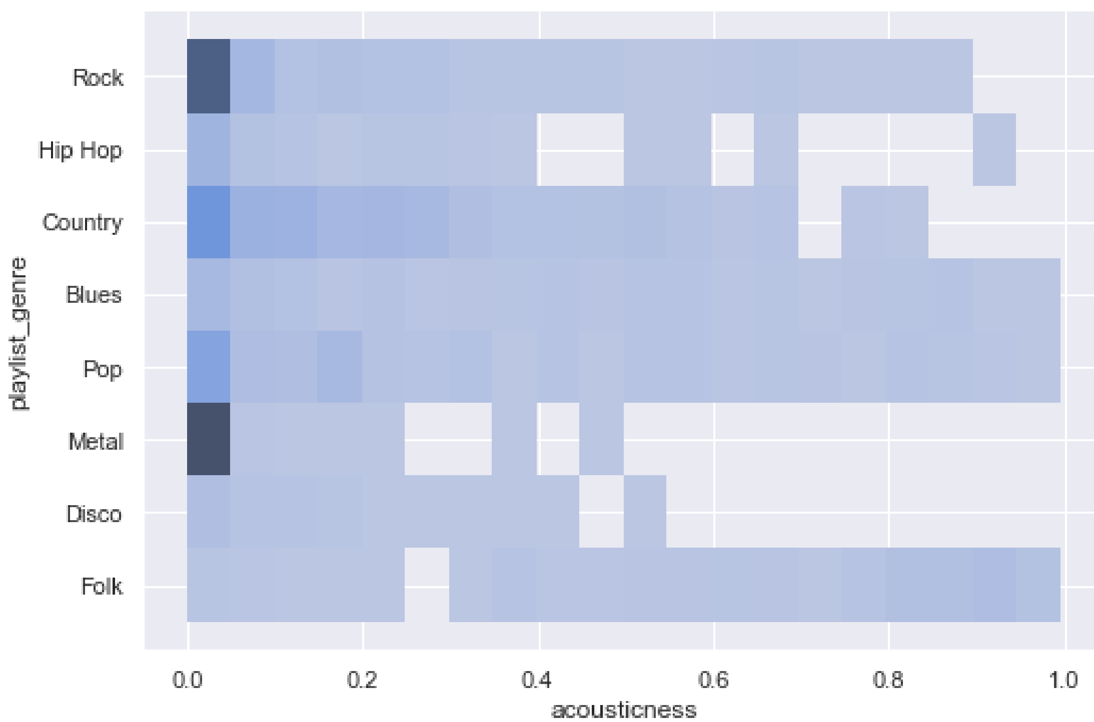


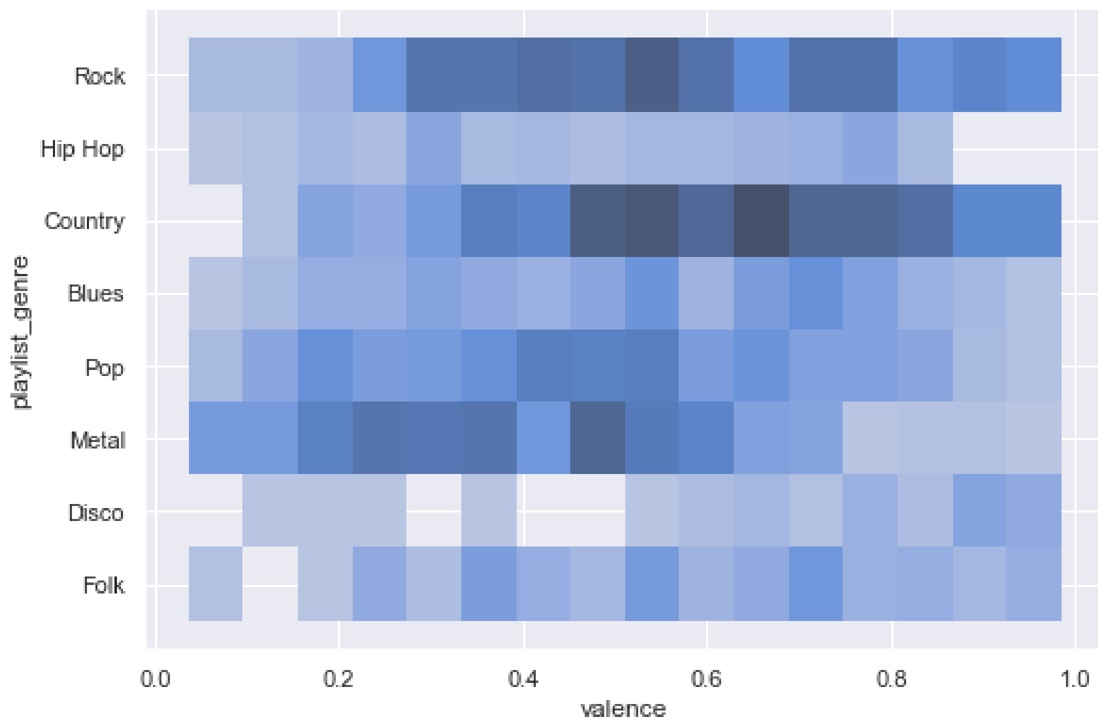
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
0
2
4
6
8
10
key

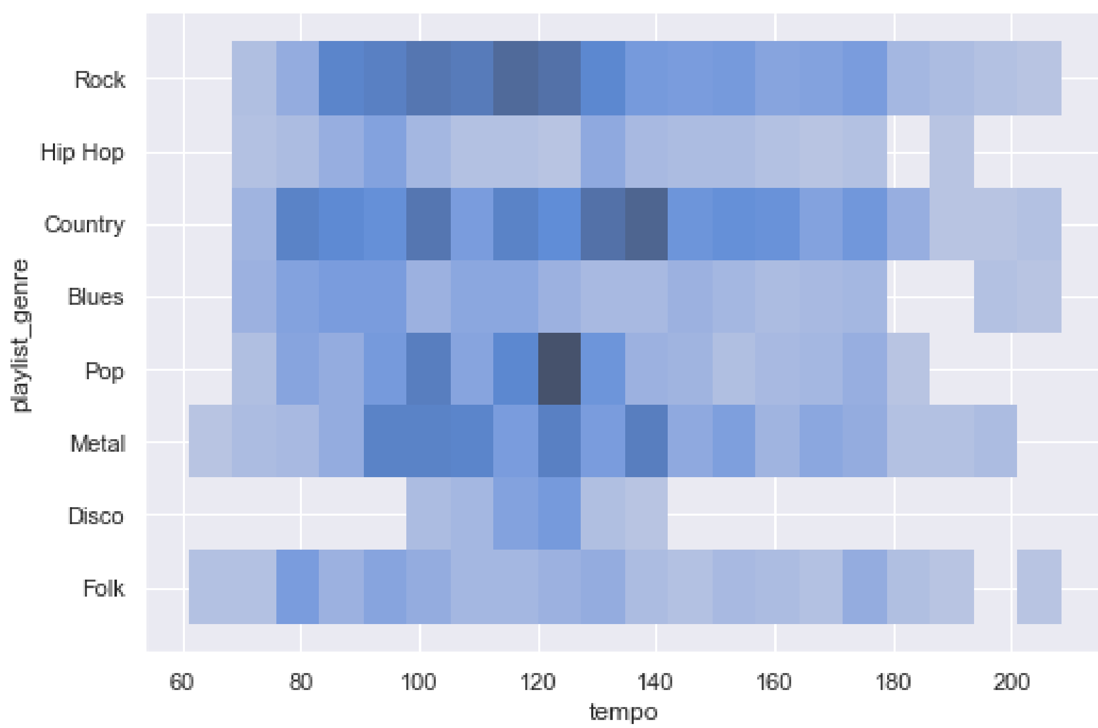


Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
−22.5
−20.0
−17.5
−15.0
−12.5
−10.0
−7.5
−5.0
−2.5
loudness


Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
0.0
0.2
0.4
0.6
0.8
1.0
mode

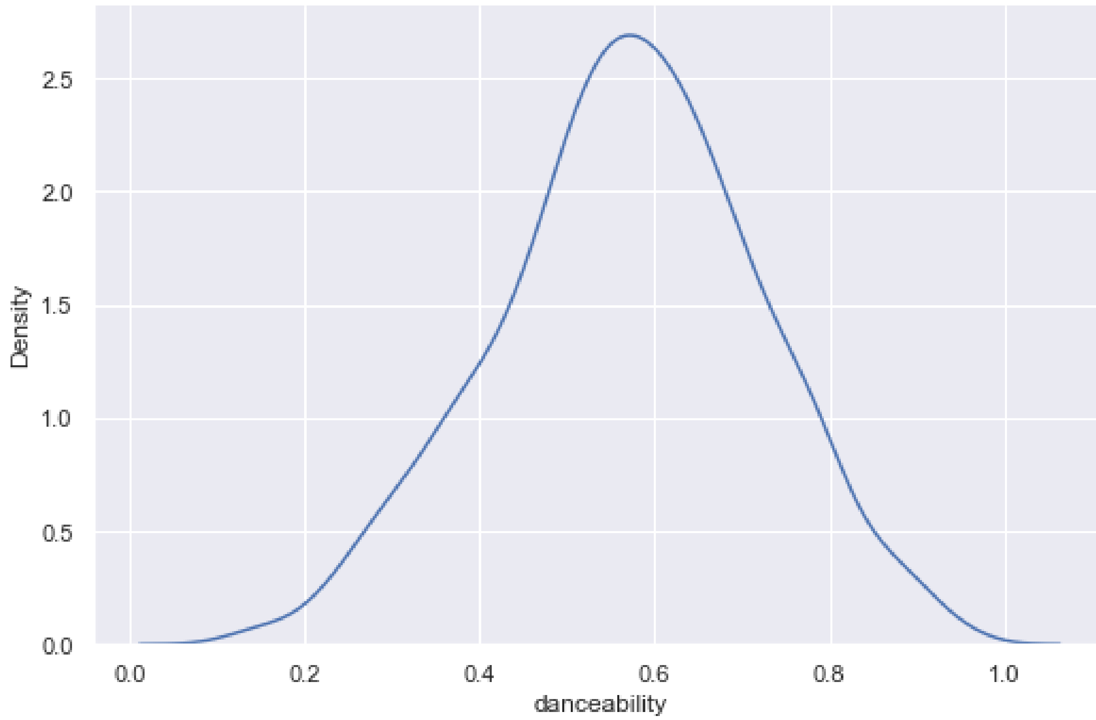

Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
0.0
0.2
0.4
0.6
0.8
1.0
acousticness

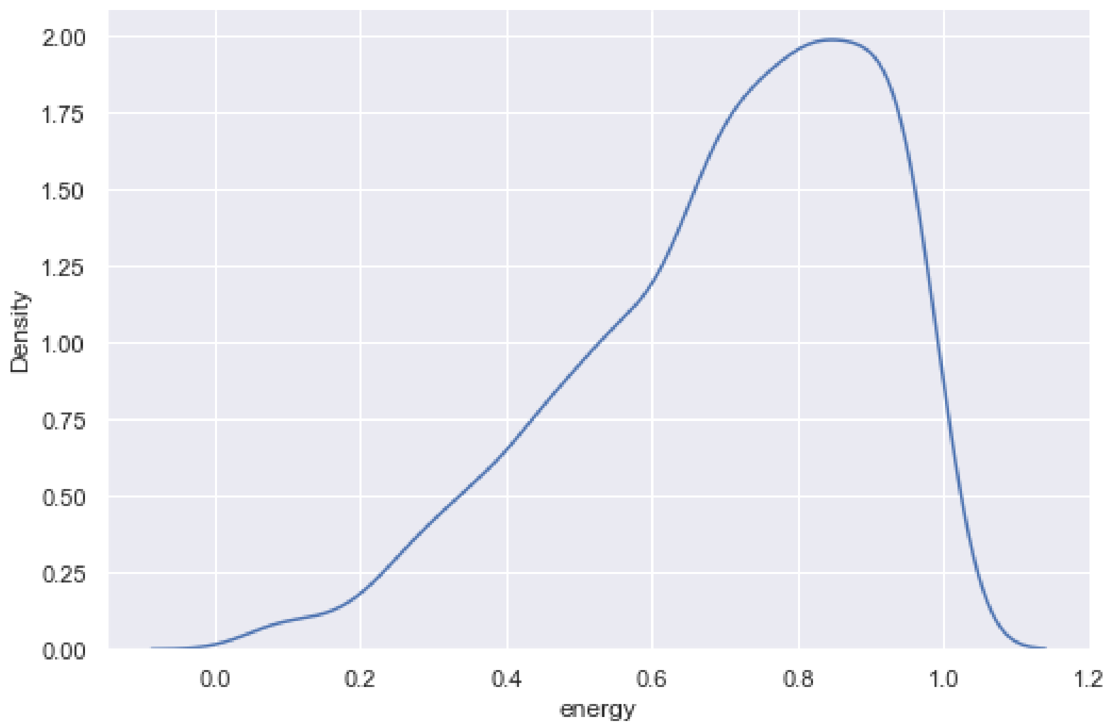

Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
0.0
0.2
0.4
0.6
0.8
1.0
valence

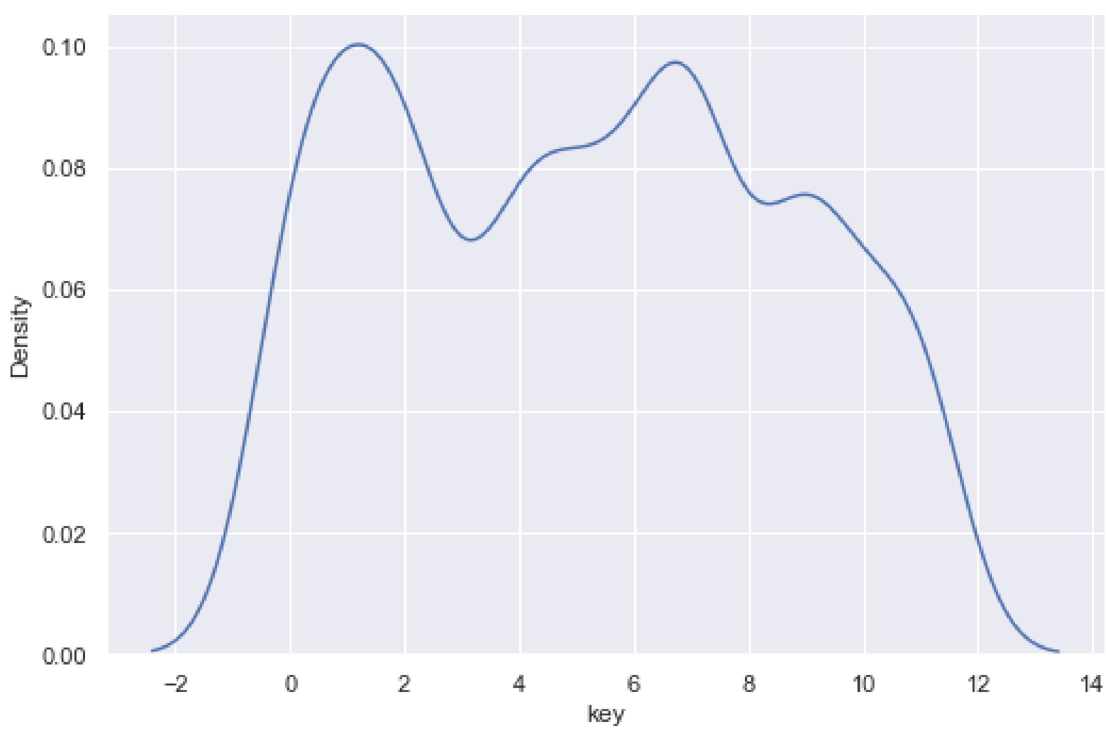

Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk
playlist_genre
60
80
100
120
140
160
180
200
tempo

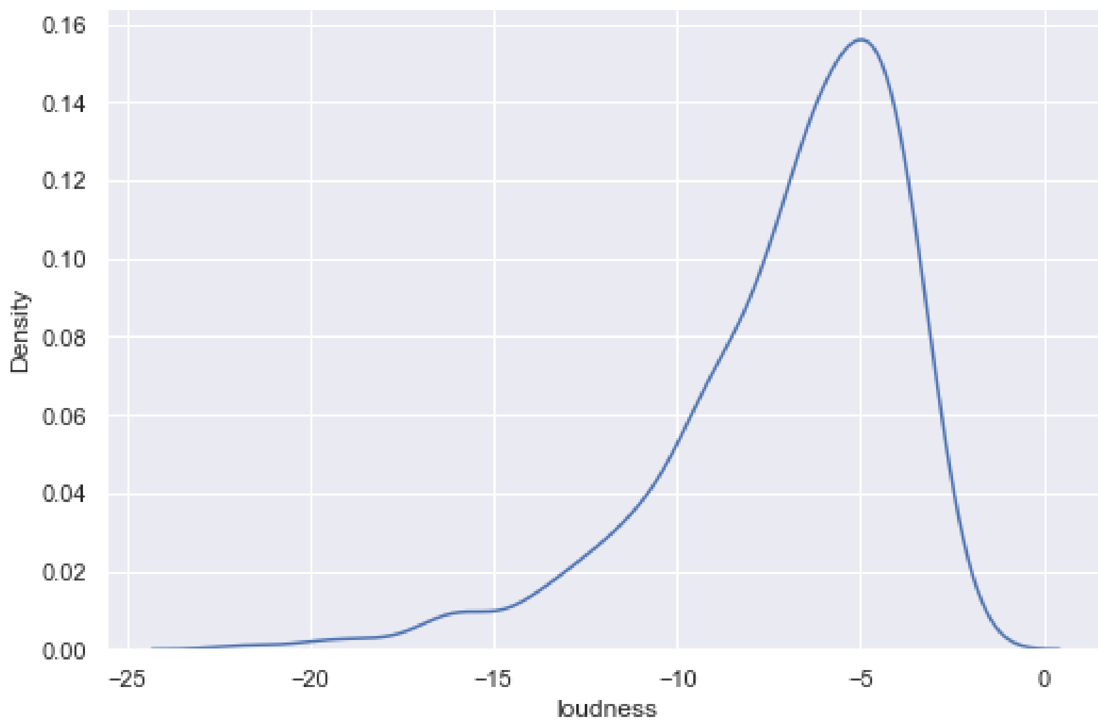
3.0.2 KDE

[16]:
```
for var in vars:
    sns.displot(data=ldf, x=var, kind='kde', aspect=1.5)
```

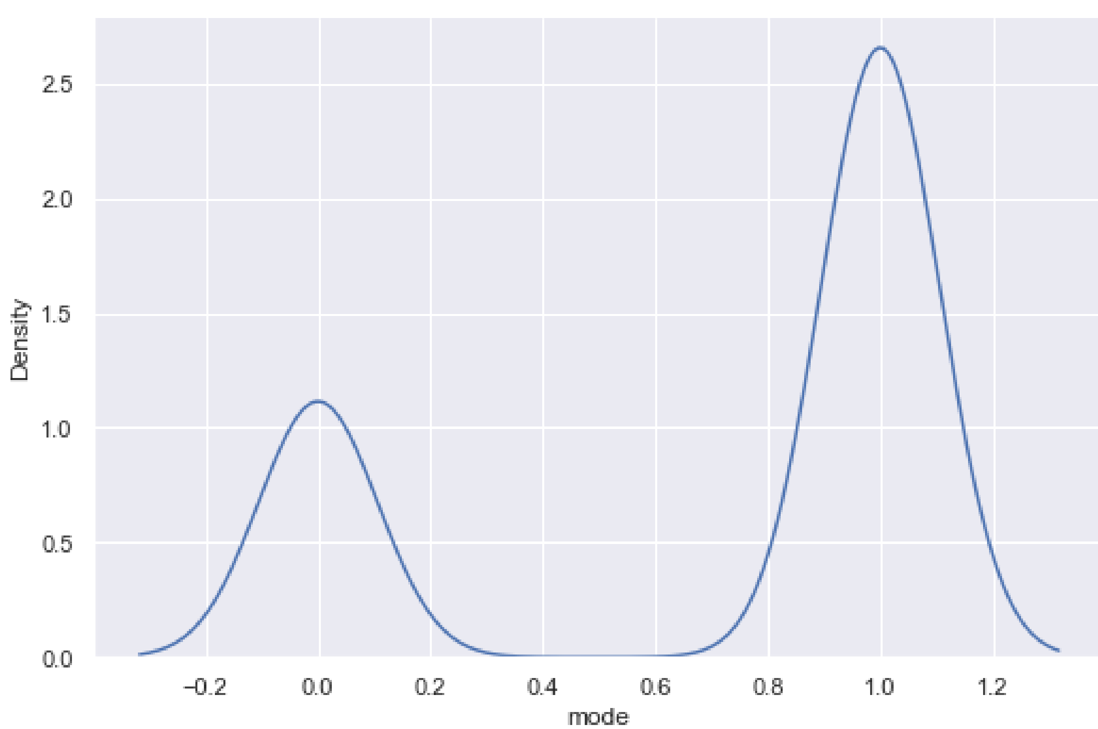



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
energy

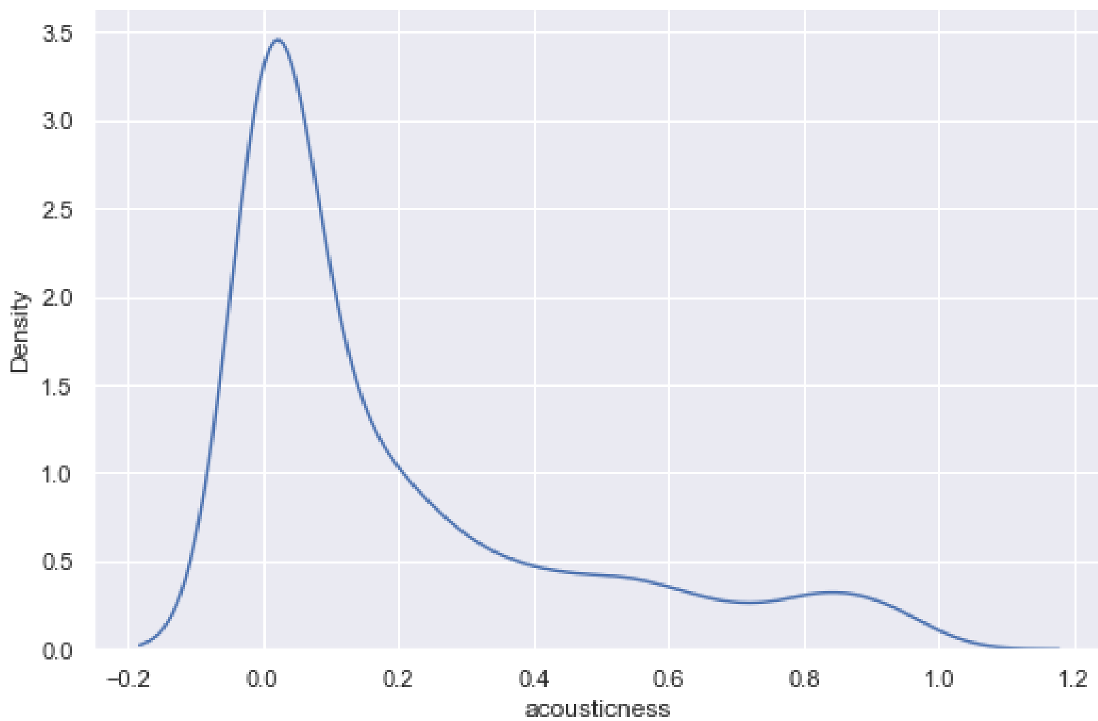

0.10
0.08
0.06
0.04
0.02
0.00
Density
−2
0
2
4
6
8
10
12
14
key

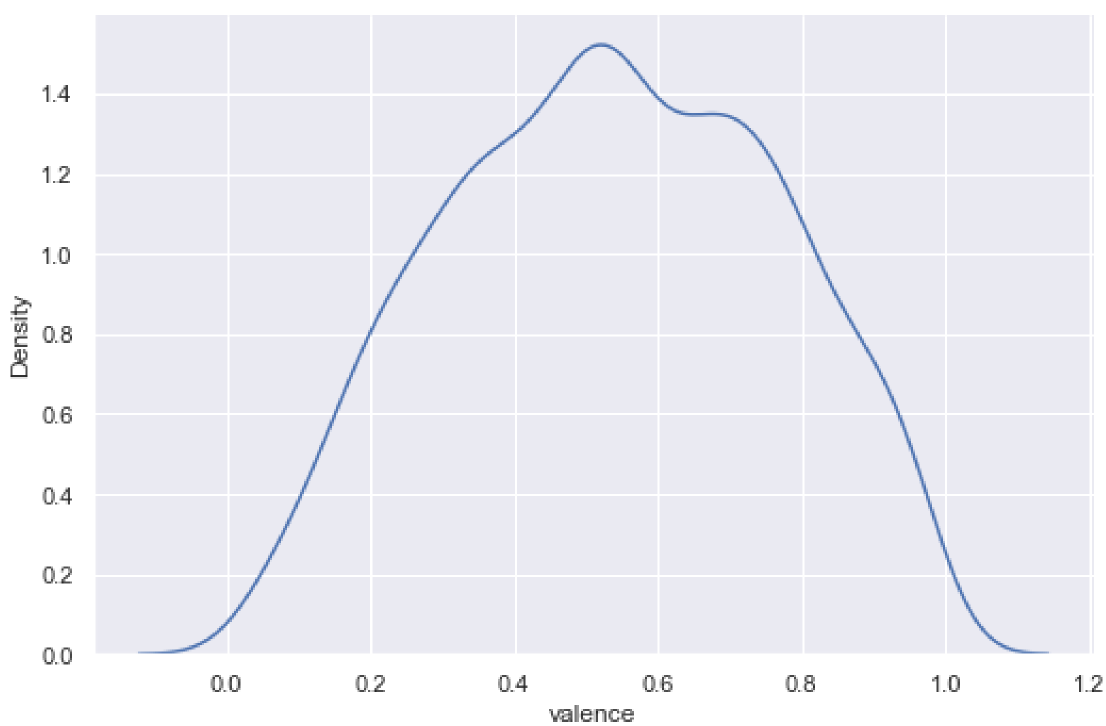


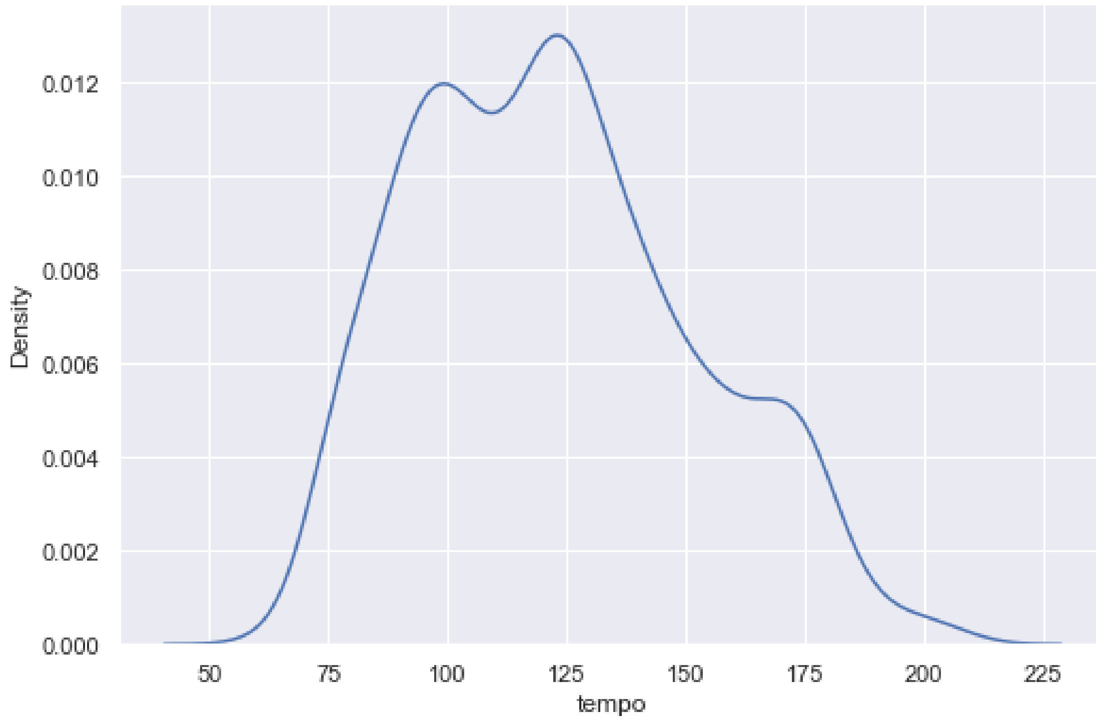
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
−25
−20
−15
−10
−5
0
loudness



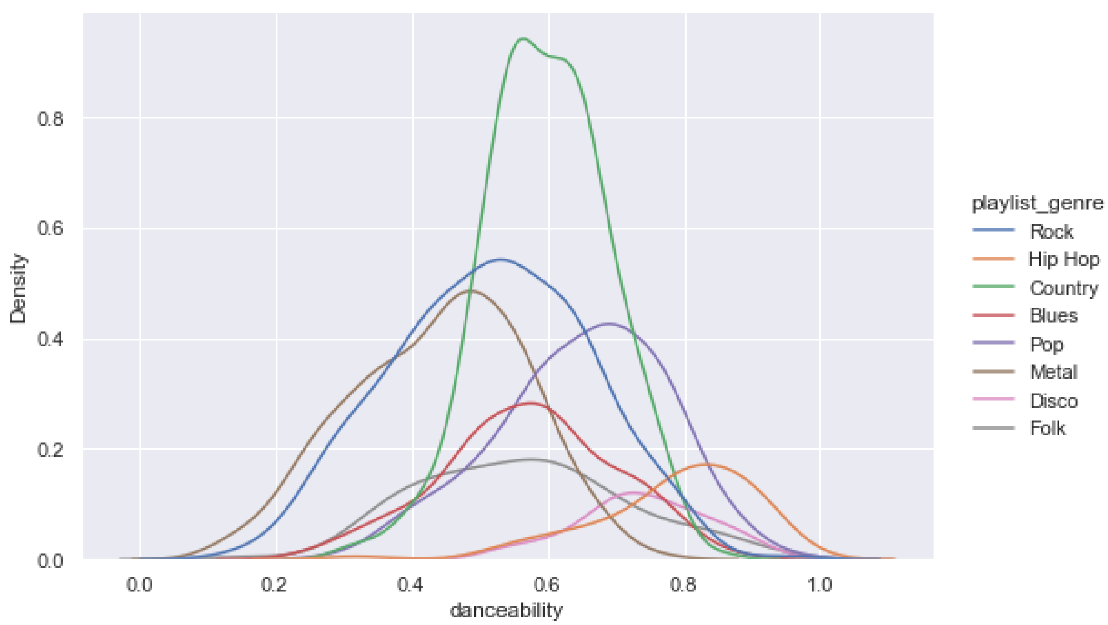
2.5
2.0
1.5
1.0
0.5
0.0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
mode

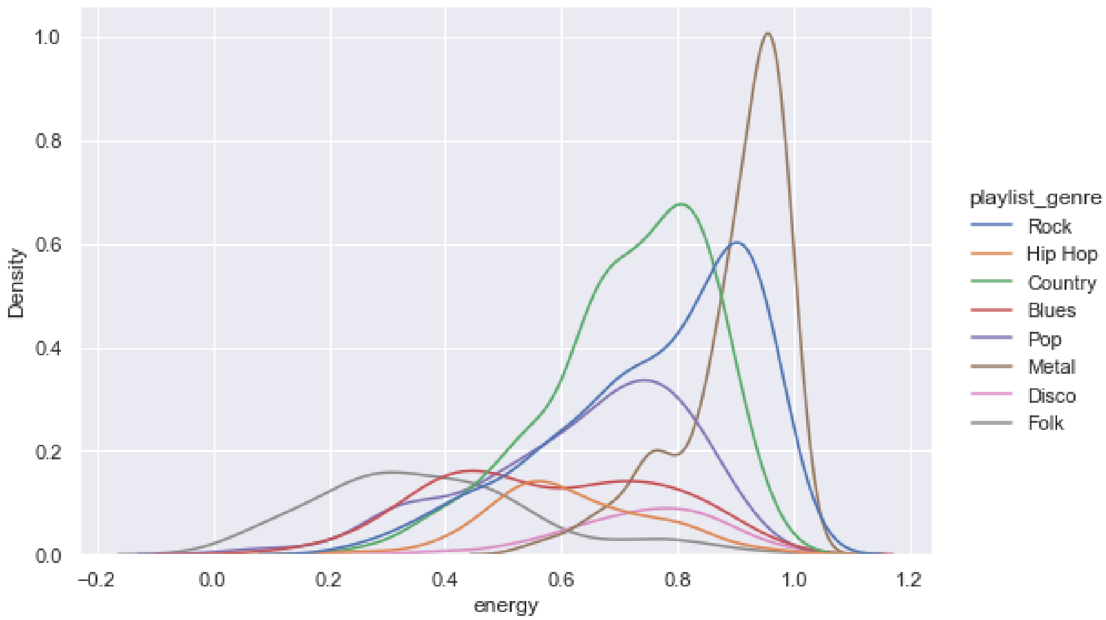

3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
acousticness


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
valence

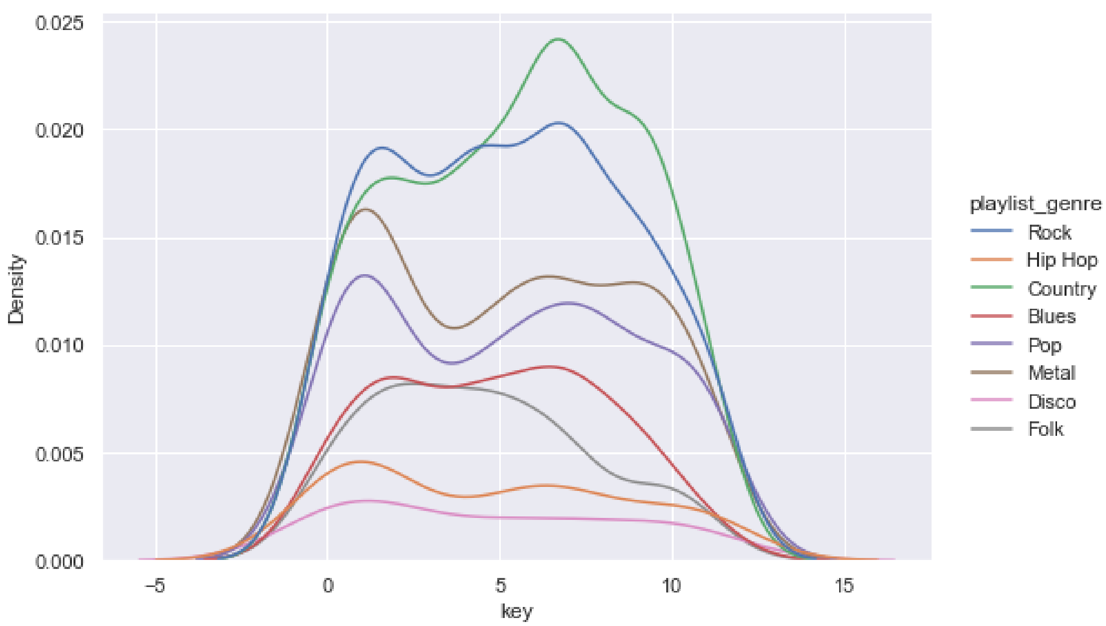


[17]:
```
for var in vars:
    sns.displot(data=ldf, x=var, hue='playlist_genre', kind='kde', aspect=1.5)
```

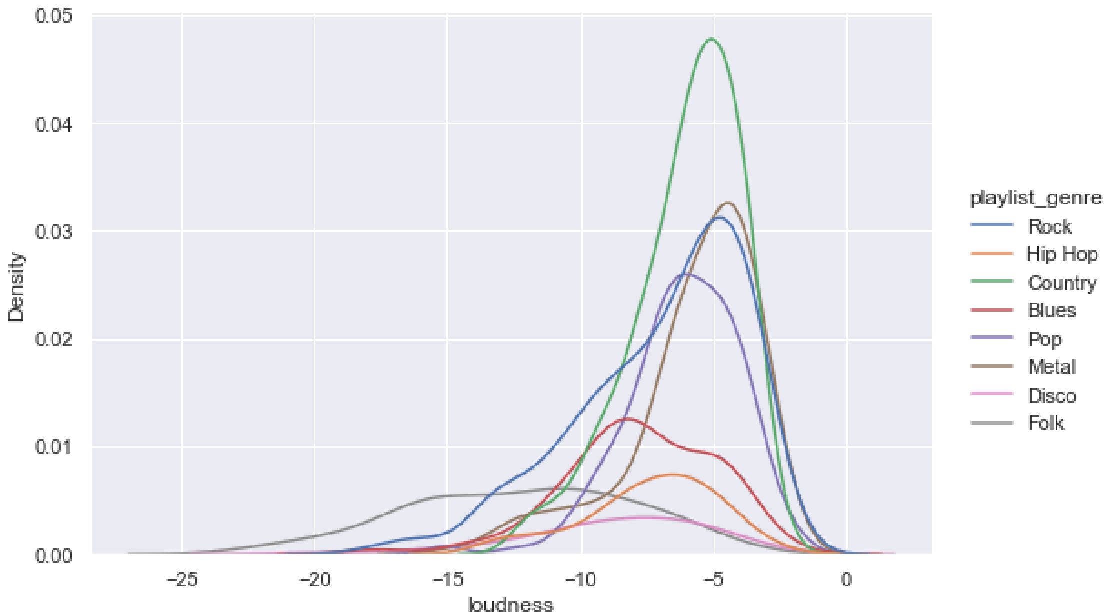


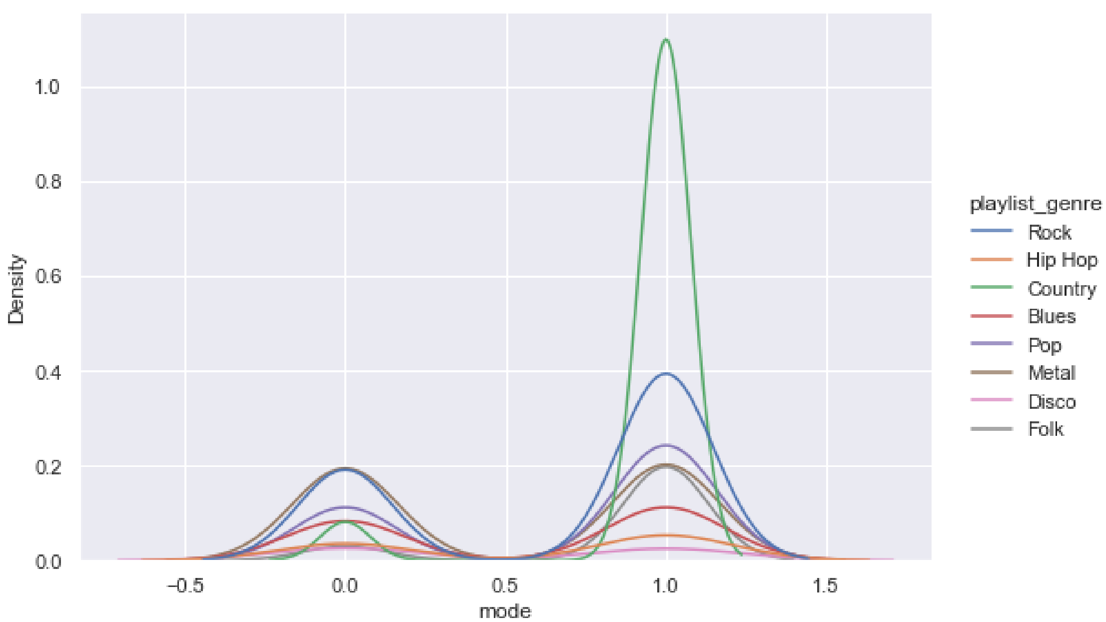


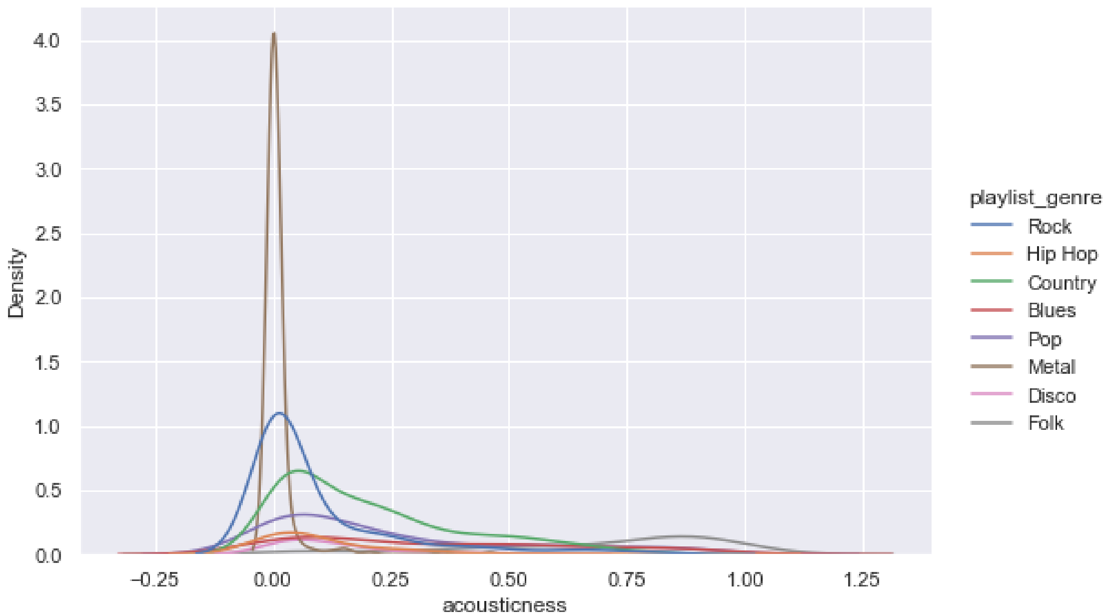
1.0
0.8
0.6
0.4
0.2
0.0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
energy
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

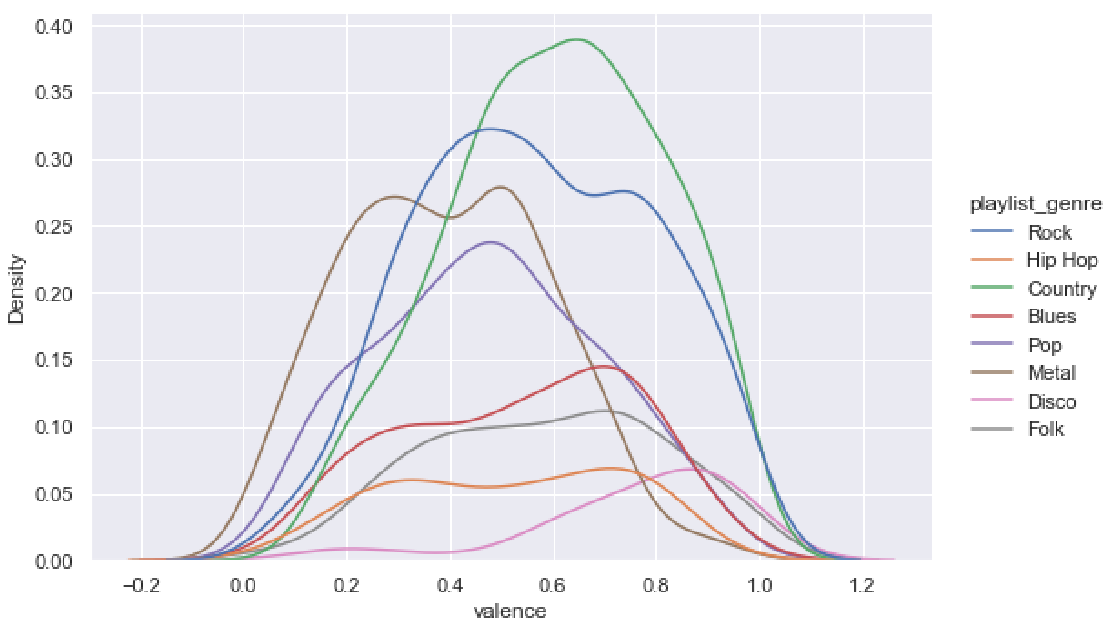


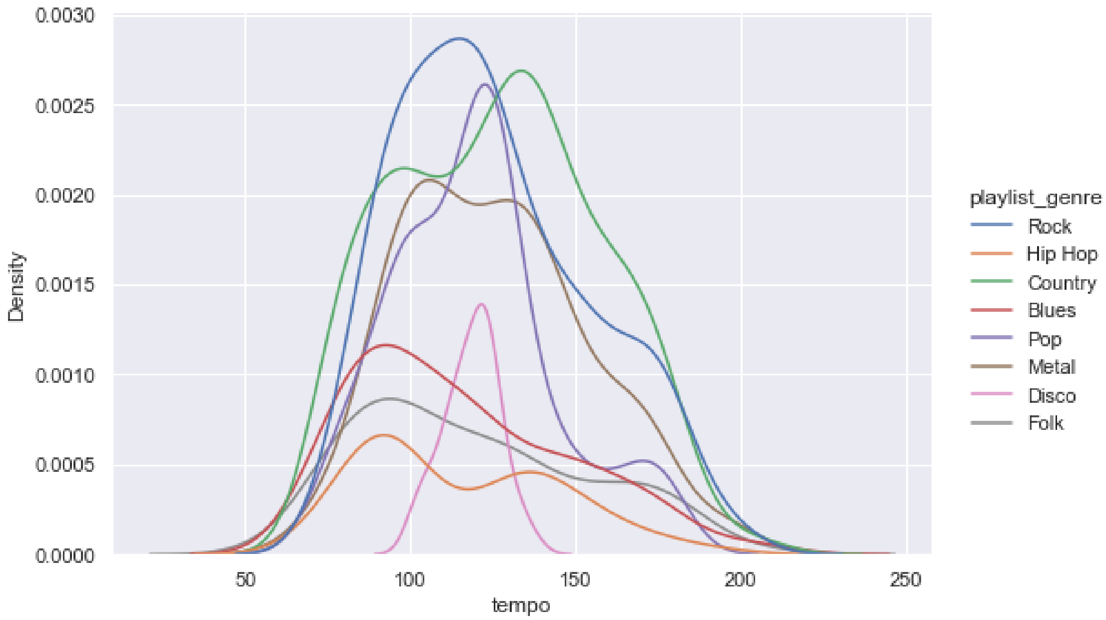
0.025
0.020
0.015
0.010
0.005
0.000
Density
−5
0
5
10
15
key
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk


0.05
0.04
0.03
0.02
0.01
0.00
Density
−25
−20
−15
−10
−5
0
loudness
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

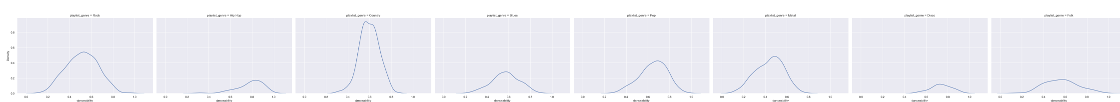

1.0
0.8
0.6
0.4
0.2
0.0
Density
−0.5
0.0
0.5
1.0
1.5
mode
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

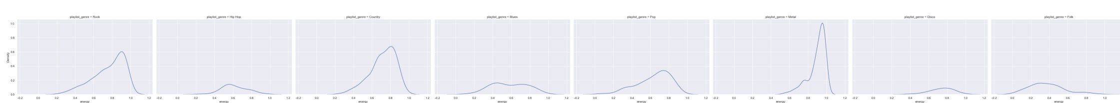

Density
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
acousticness
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

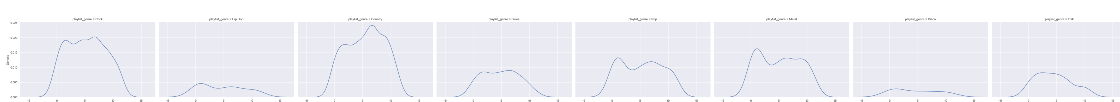

Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
valence
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

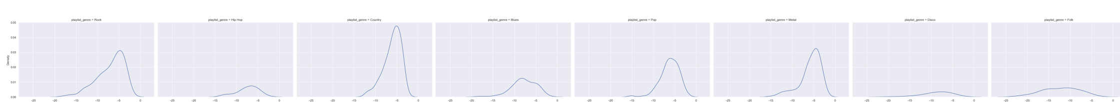


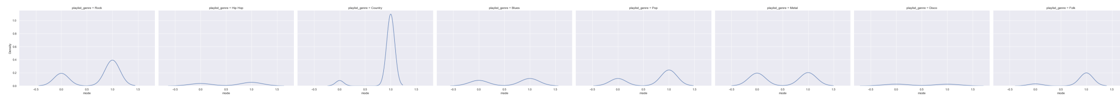
[18]:
```python
for var in vars:
    sns.displot(data=ldf, x=var, col='playlist_genre', kind='kde', aspect=1.5)
```

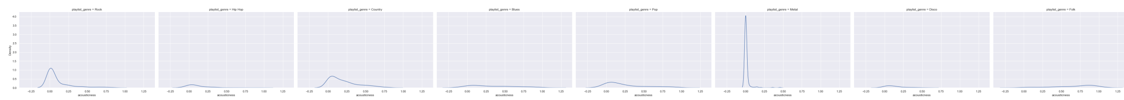


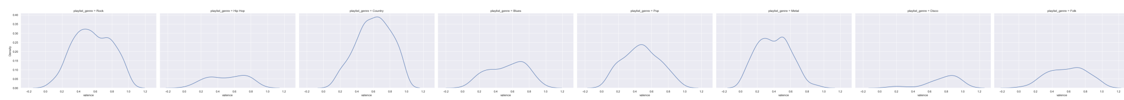


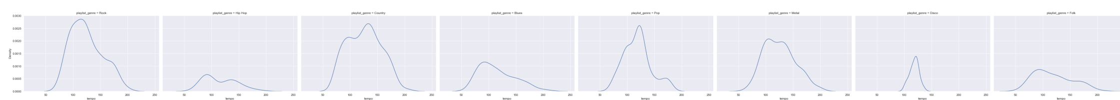






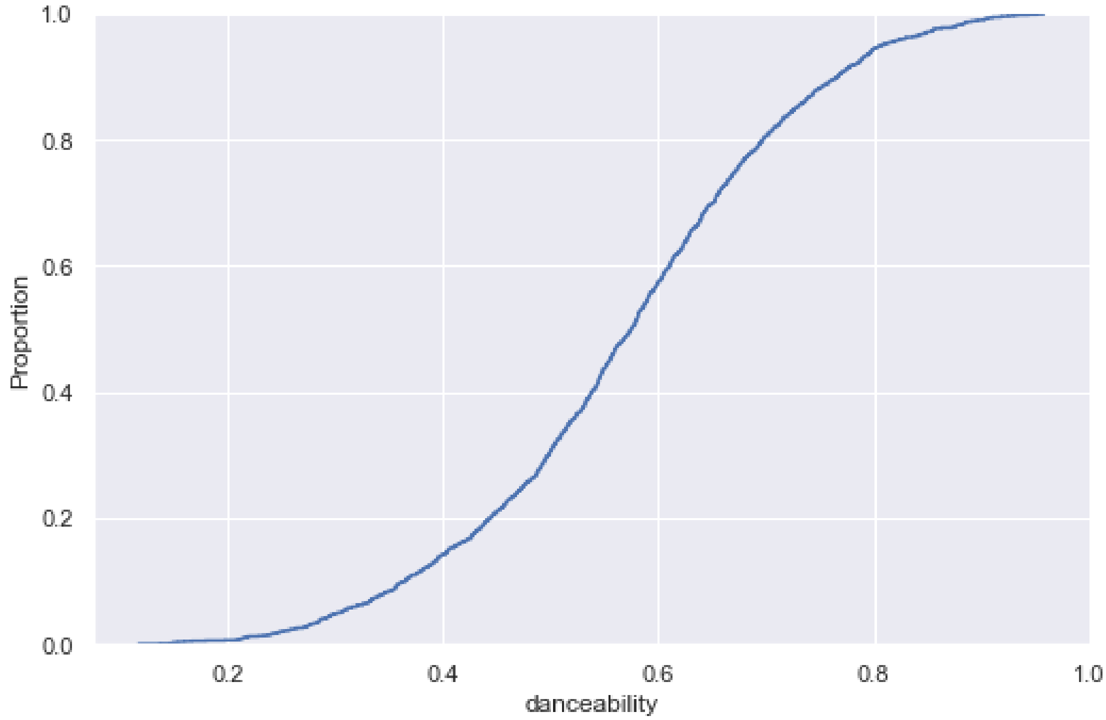


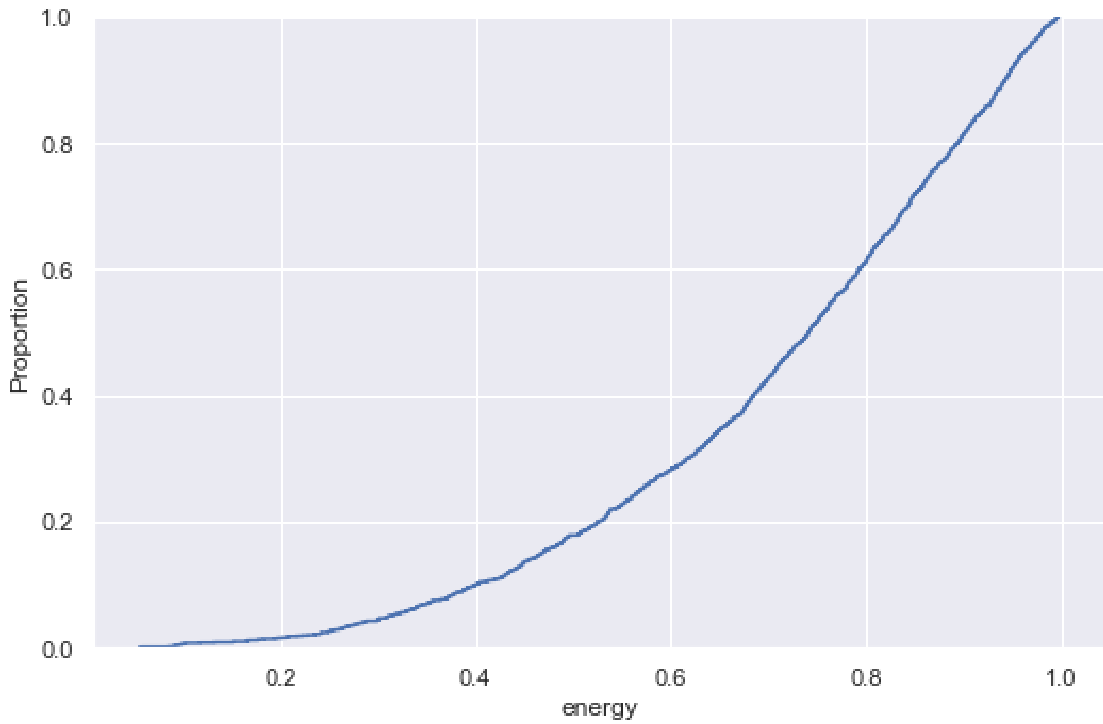


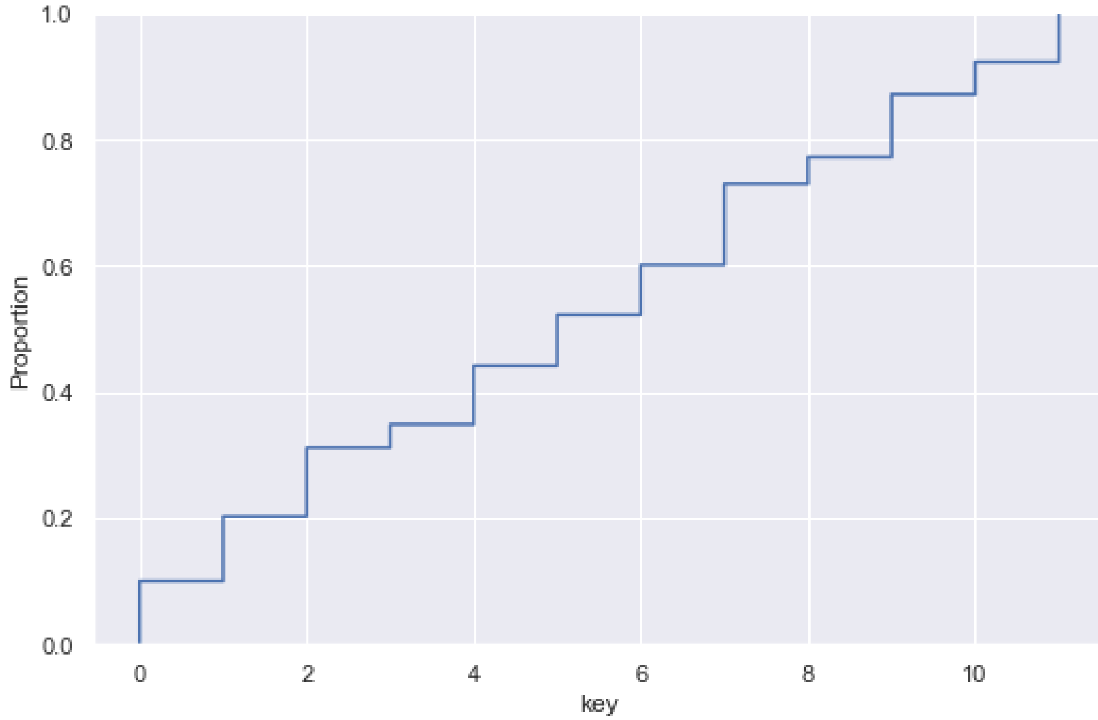


3.0.3 ECDF

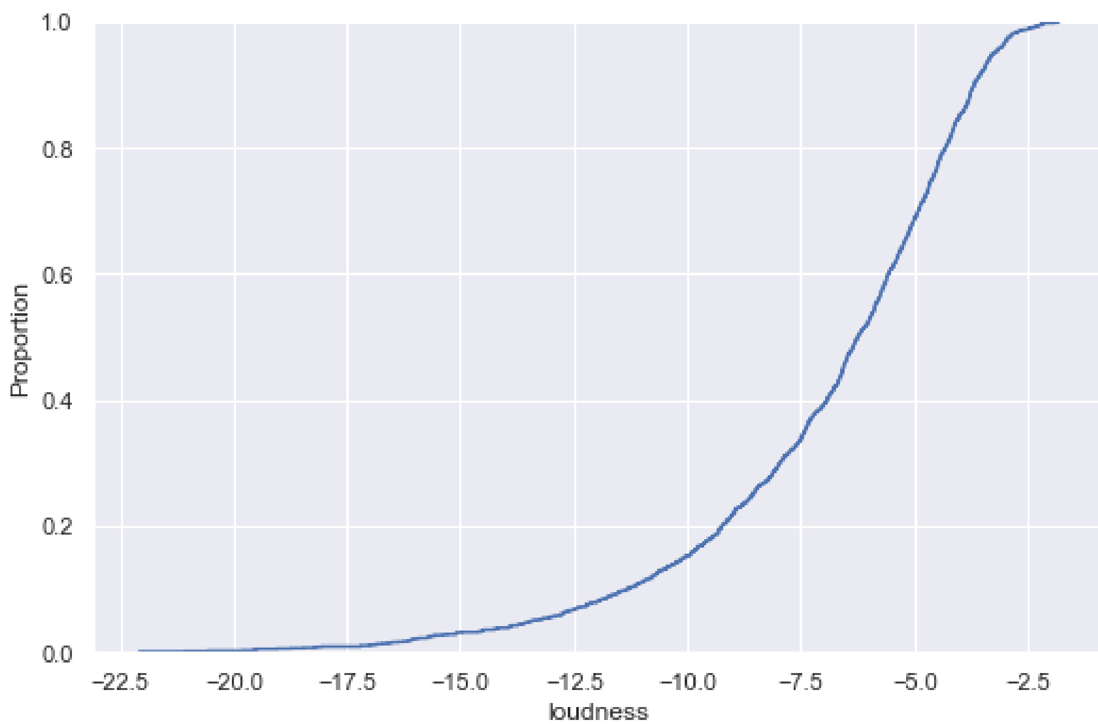
[19]:
```
for var in vars:
    sns.displot(data=ldf, x=var, kind='ecdf', aspect=1.5)
```


1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.2
0.4
0.6
0.8
1.0
danceability

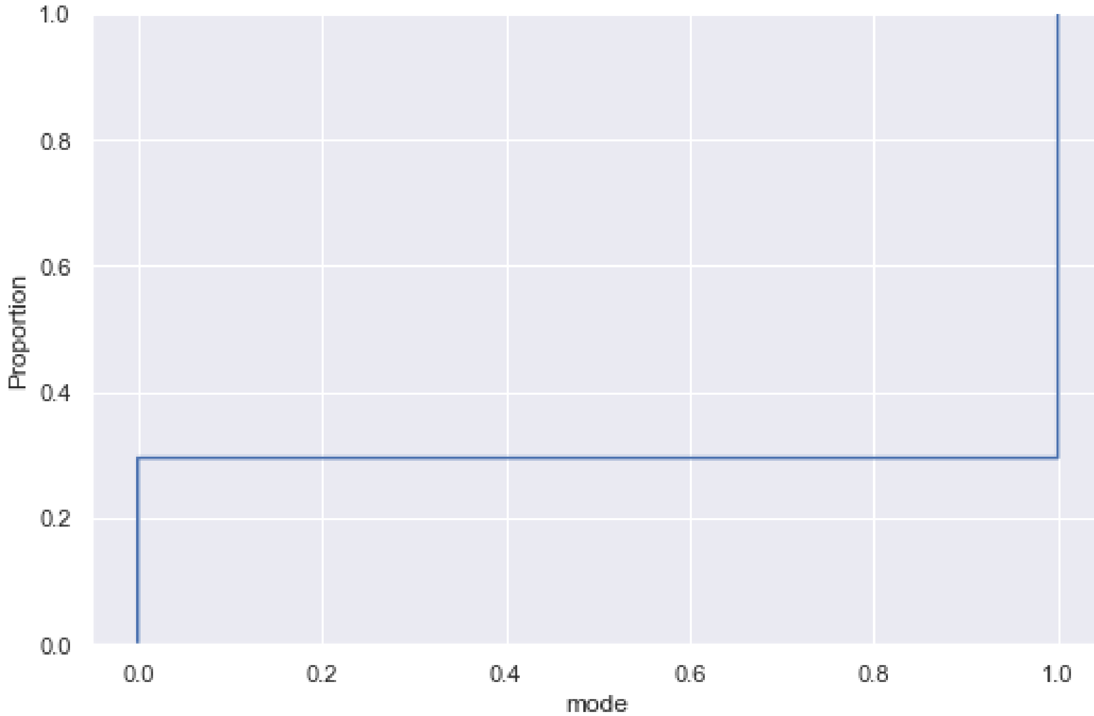

1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.2
0.4
0.6
0.8
1.0
energy



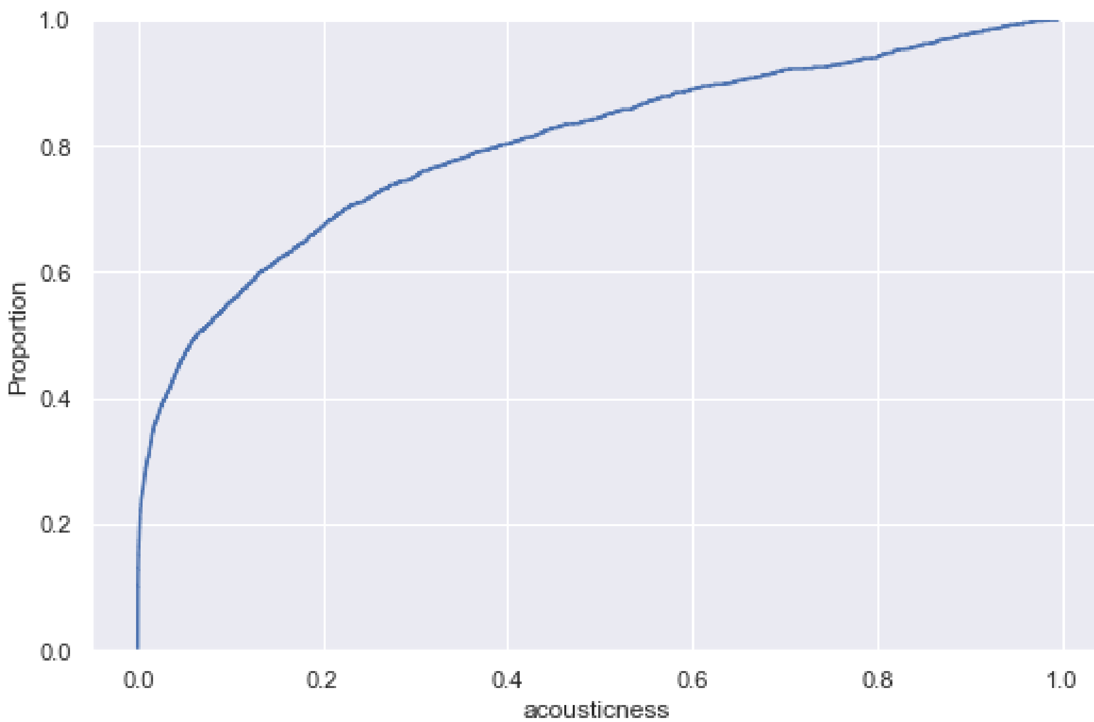
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0
2
4
6
8
10
key



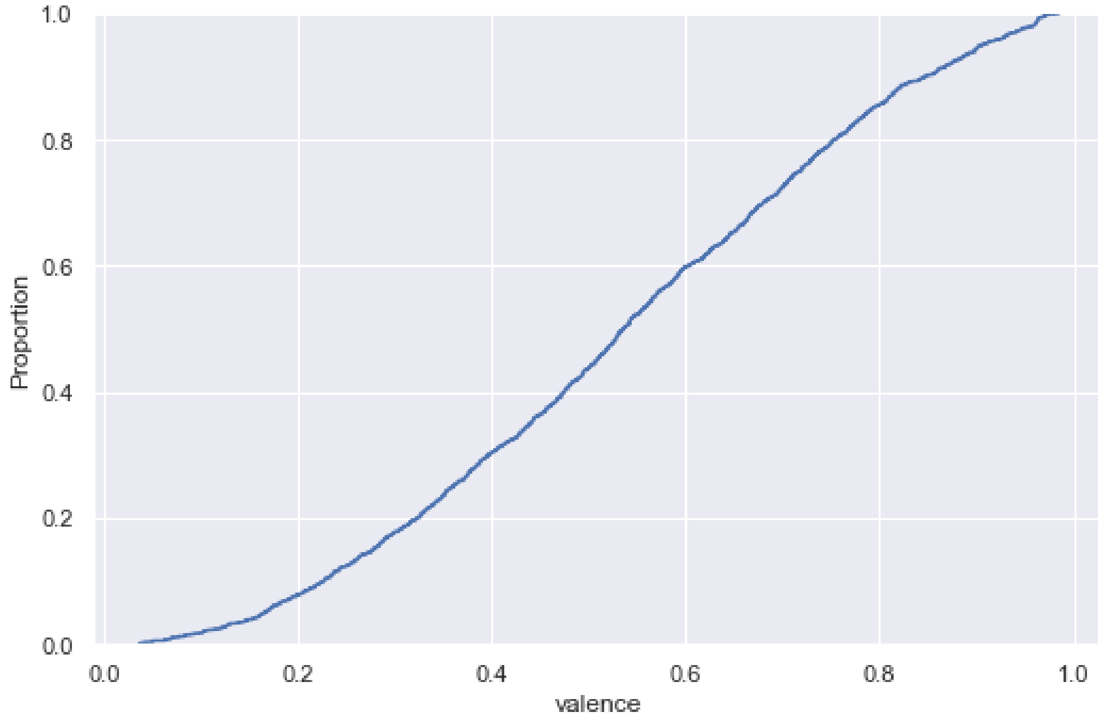
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
−22.5
−20.0
−17.5
−15.0
−12.5
−10.0
−7.5
−5.0
−2.5
loudness



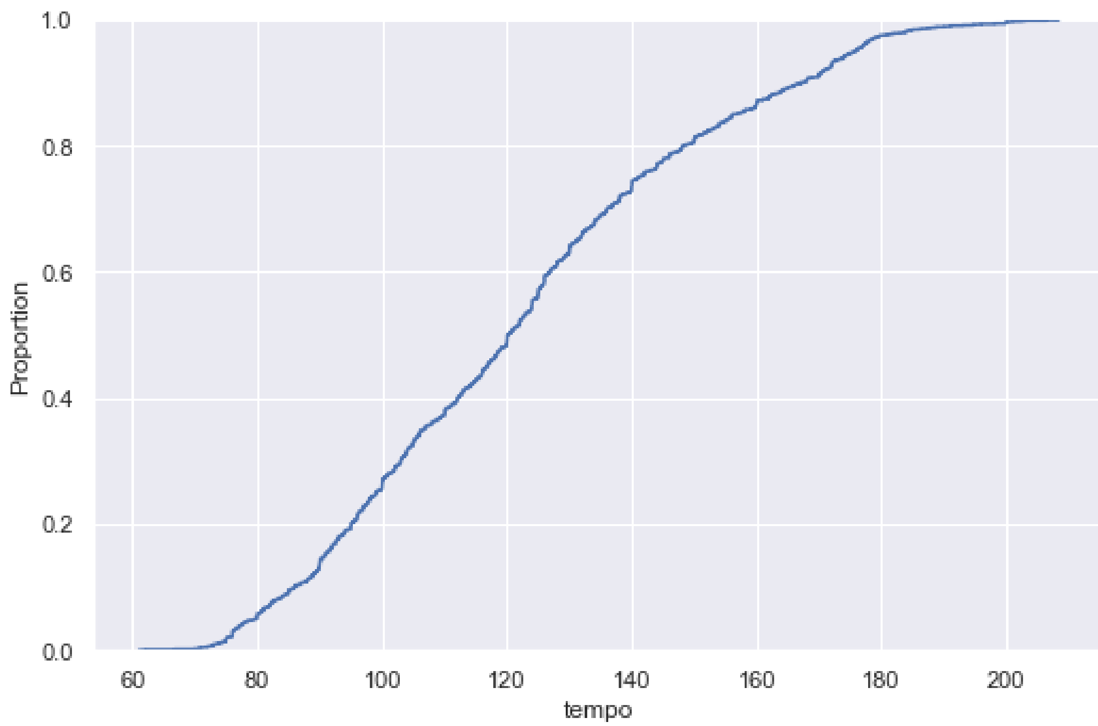
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
mode



1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
acousticness

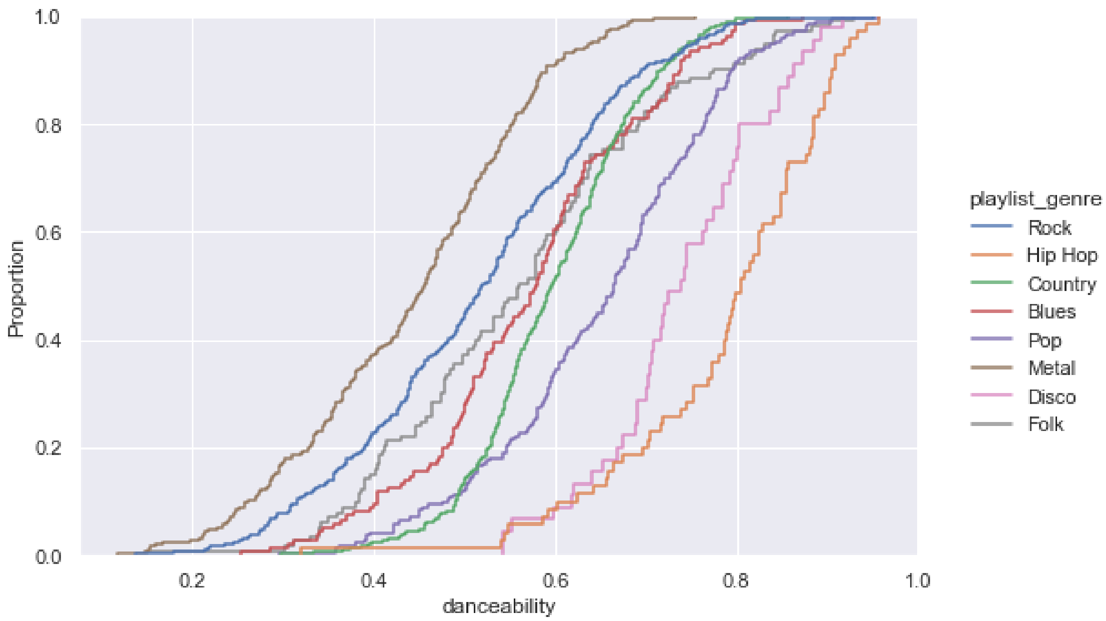


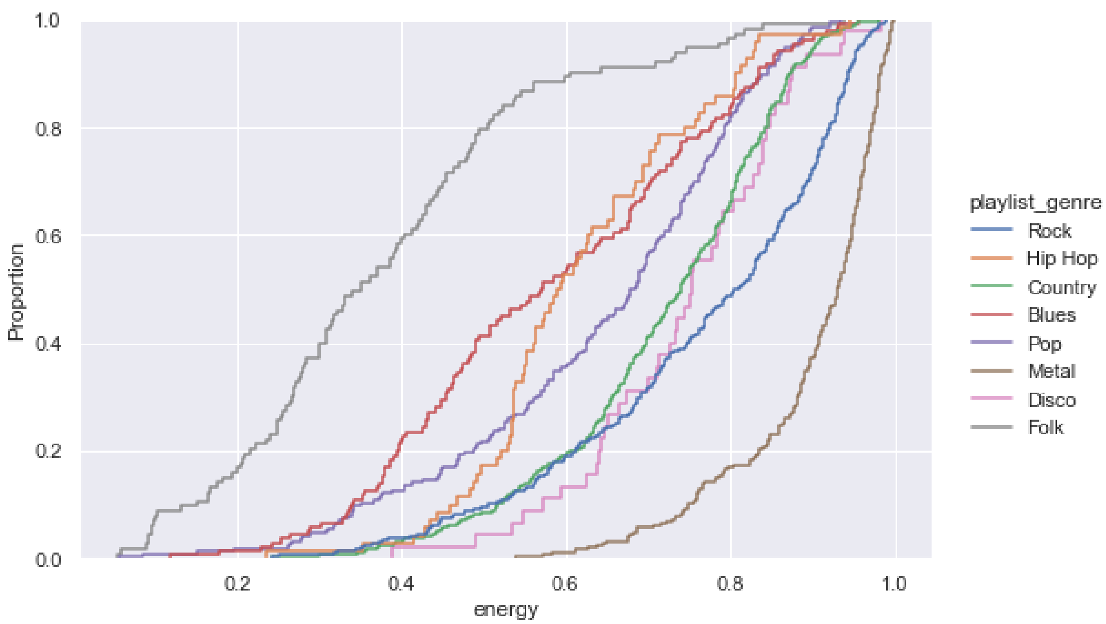
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
valence

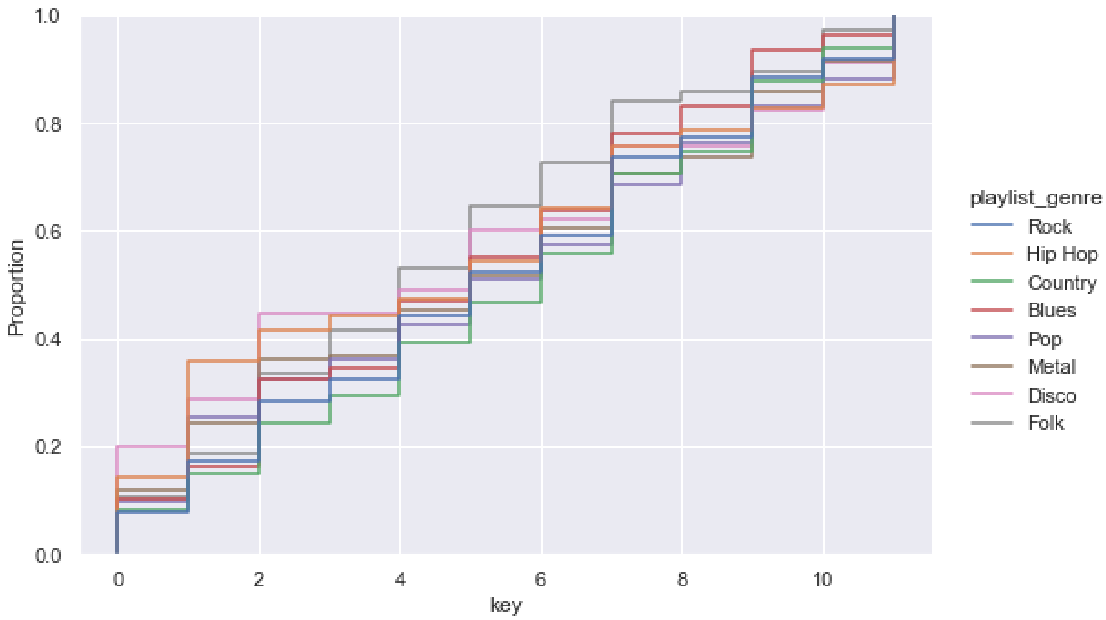


1.0
0.8
0.6
0.4
0.2
0.0
Proportion
60
80
100
120
140
160
180
200
tempo

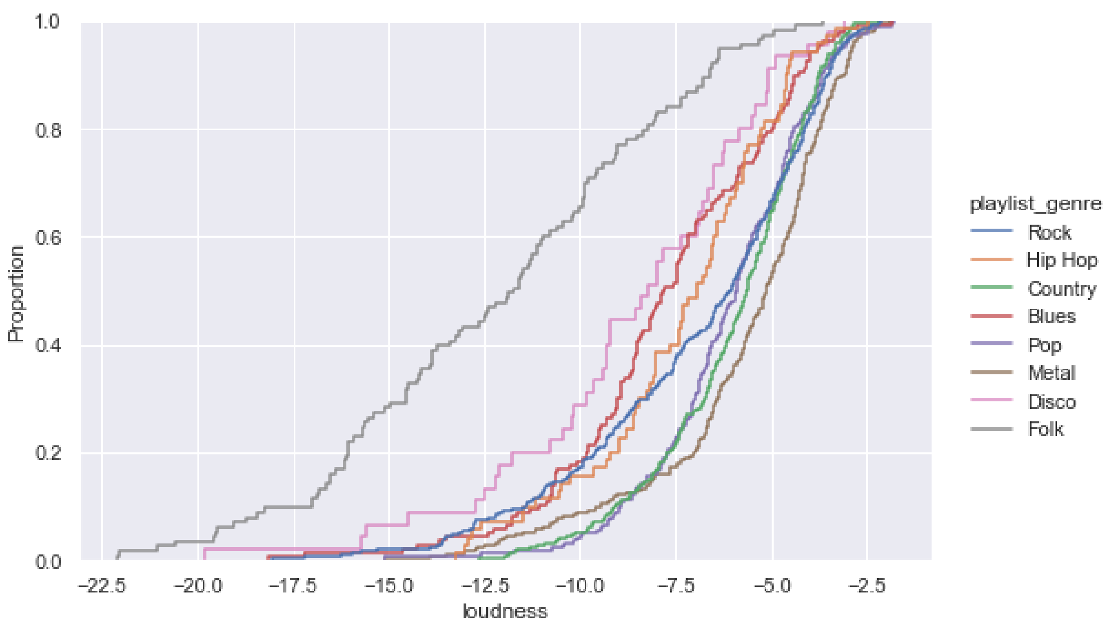
[20]:
```python
for var in vars:
    sns.displot(data=ldf, x=var, hue='playlist_genre', kind='ecdf', aspect=1.5)
```



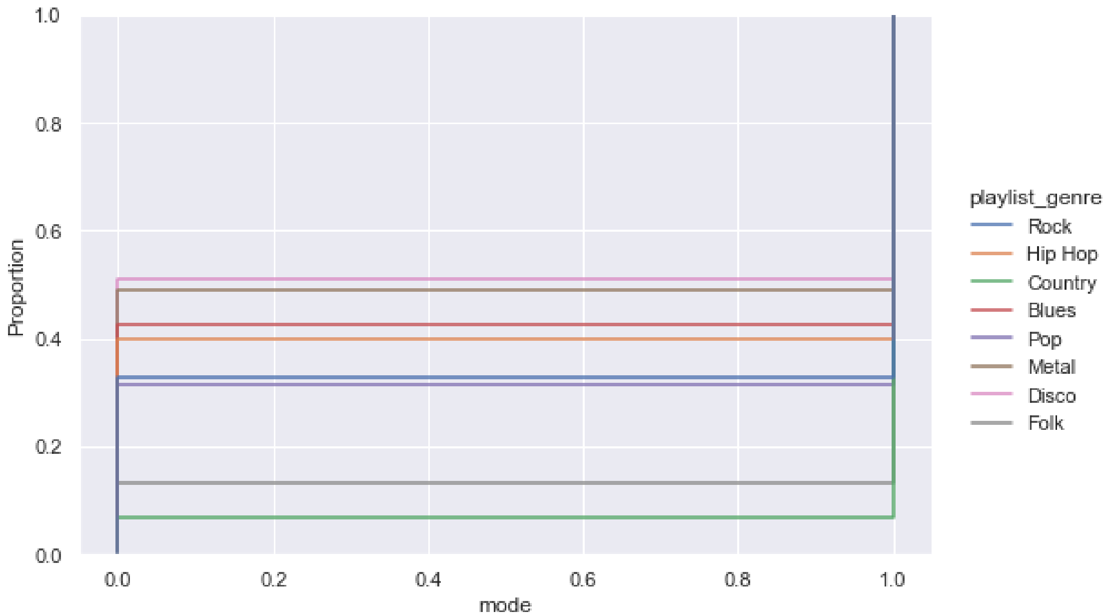


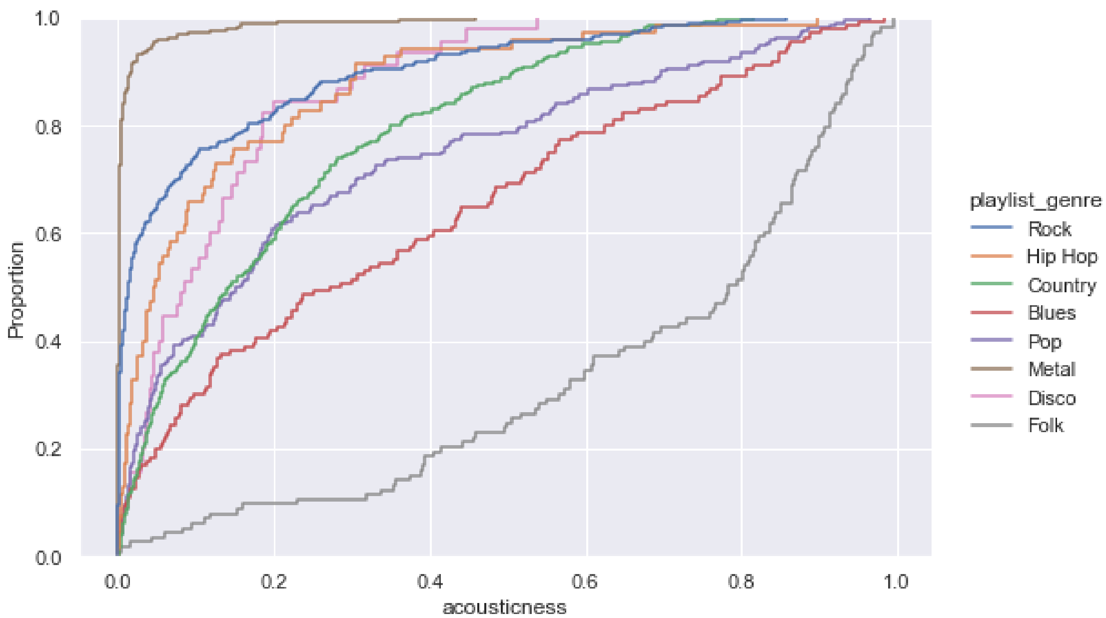


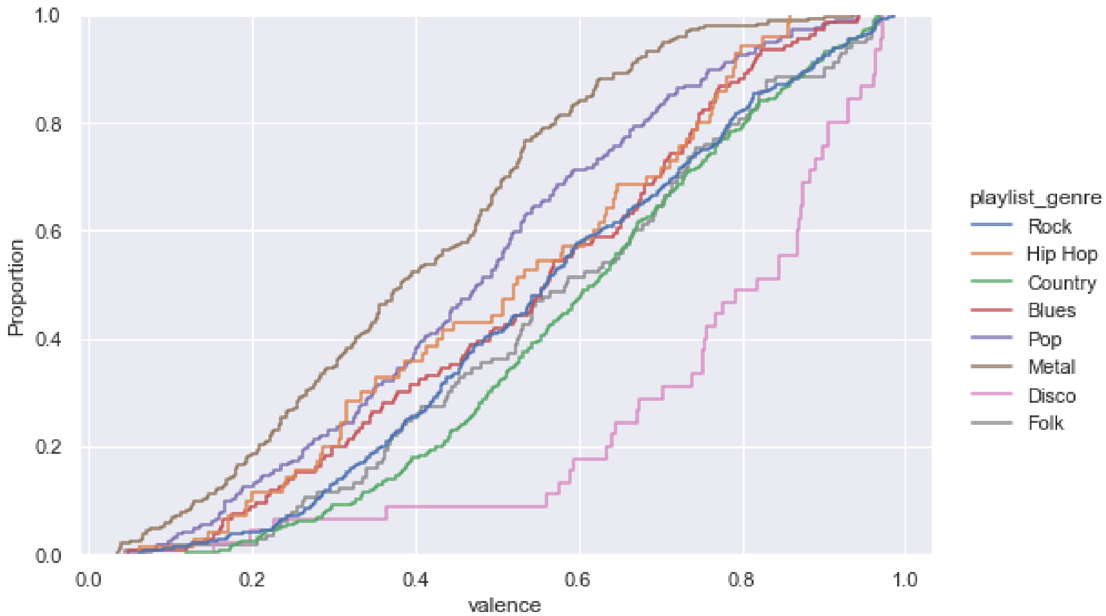
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0
2
4
6
8
10
key
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

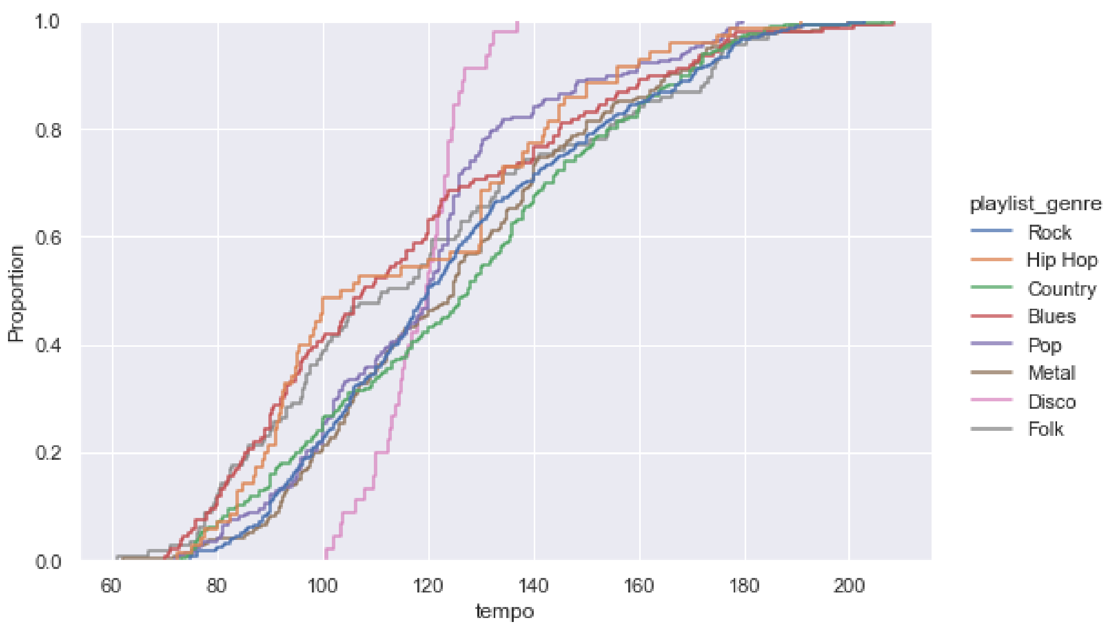


1.0
0.8
0.6
0.4
0.2
0.0
Proportion
−22.5
−20.0
−17.5
−15.0
−12.5
−10.0
−7.5
−5.0
−2.5
loudness
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

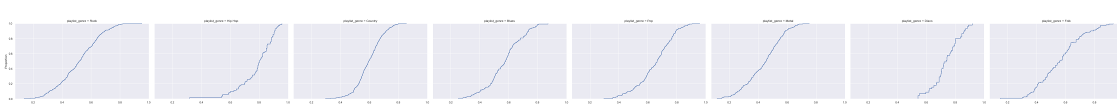

1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
mode
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

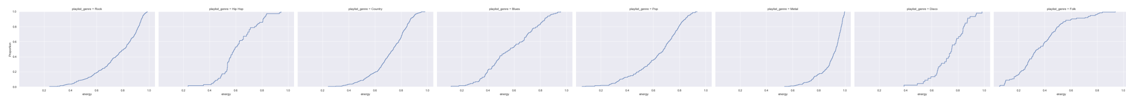

1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
acousticness
playlist_genre
Rock
Hip Hop
Country
Blues
Pop
Metal
Disco
Folk

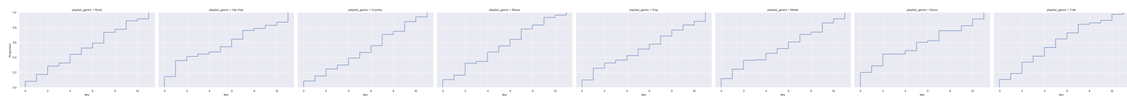


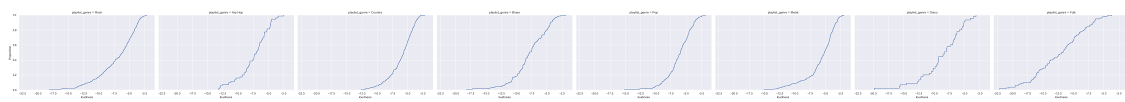


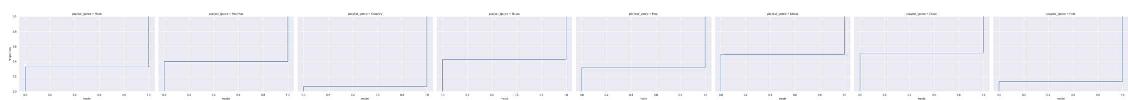
[21]:
```
for var in vars:
    sns.displot(data=ldf, x=var, col='playlist_genre', kind='ecdf', aspect=1.5)
```

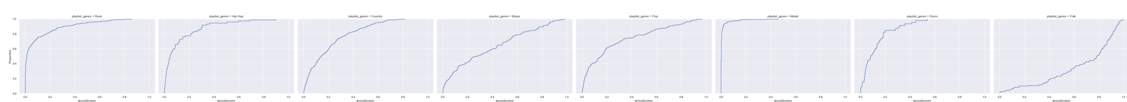


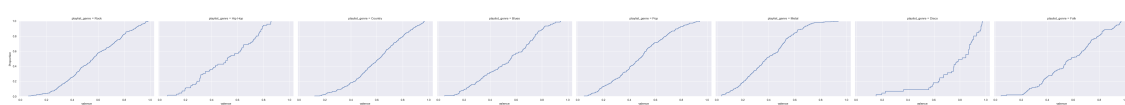


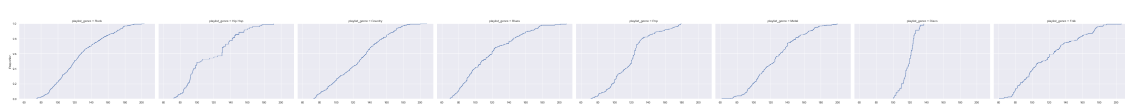










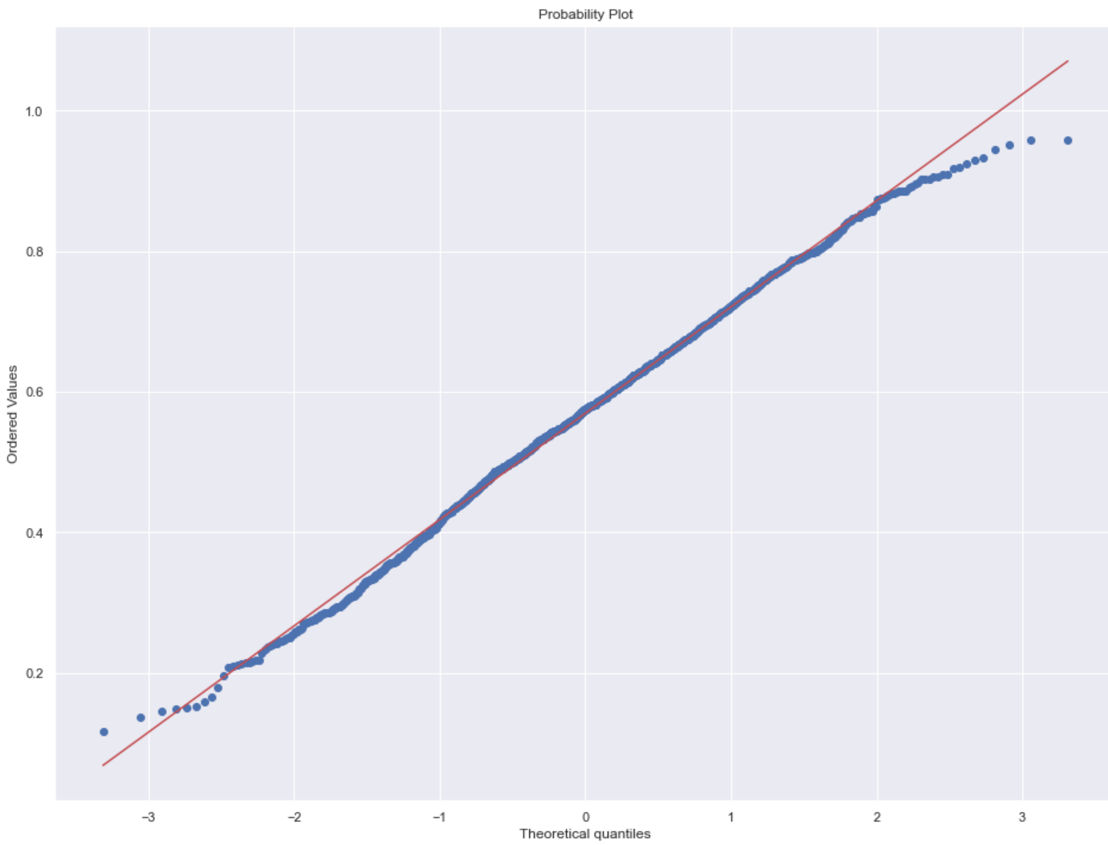


3.0.4 QQ plot for normality

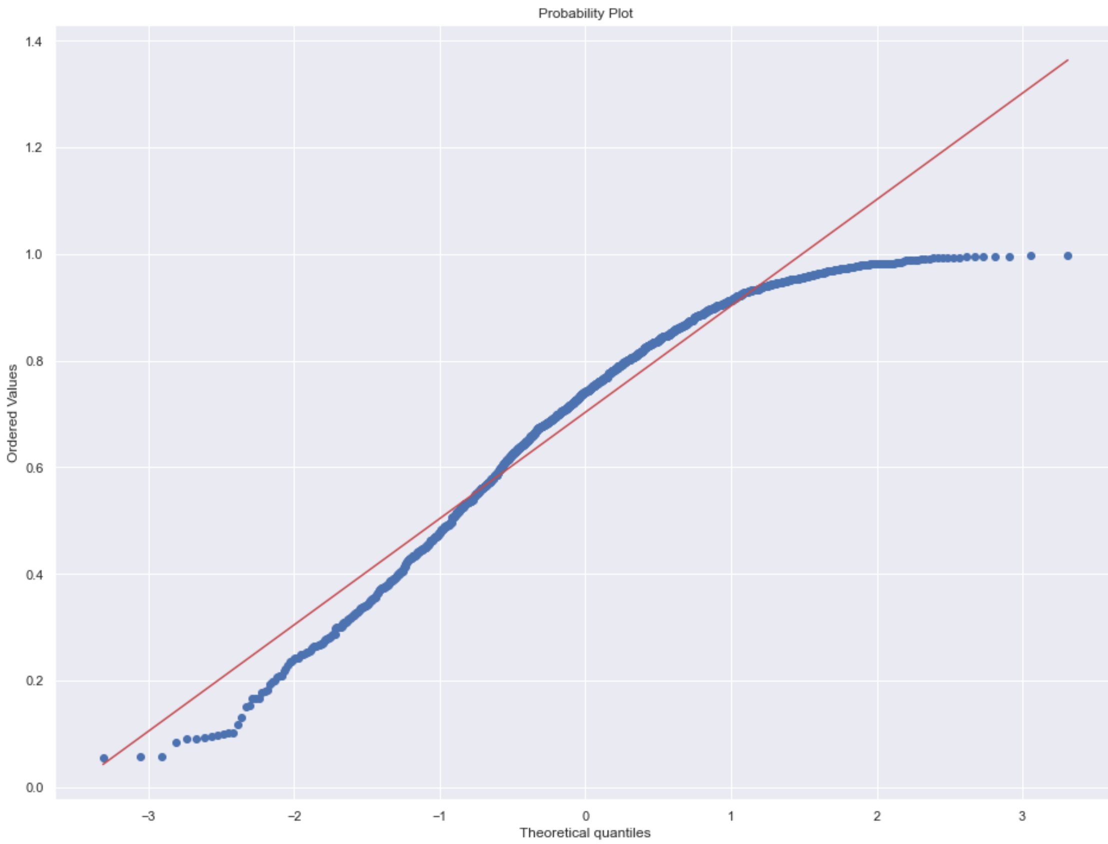
Source - https://stackoverflow.com/questions/13865596/quantile-quantile-plot-using-scipy

[22]:
```
for var in vars:
    print(var)
    stats.probplot(ldf[var], dist="norm", plot=pylab)
    pylab.show()
```

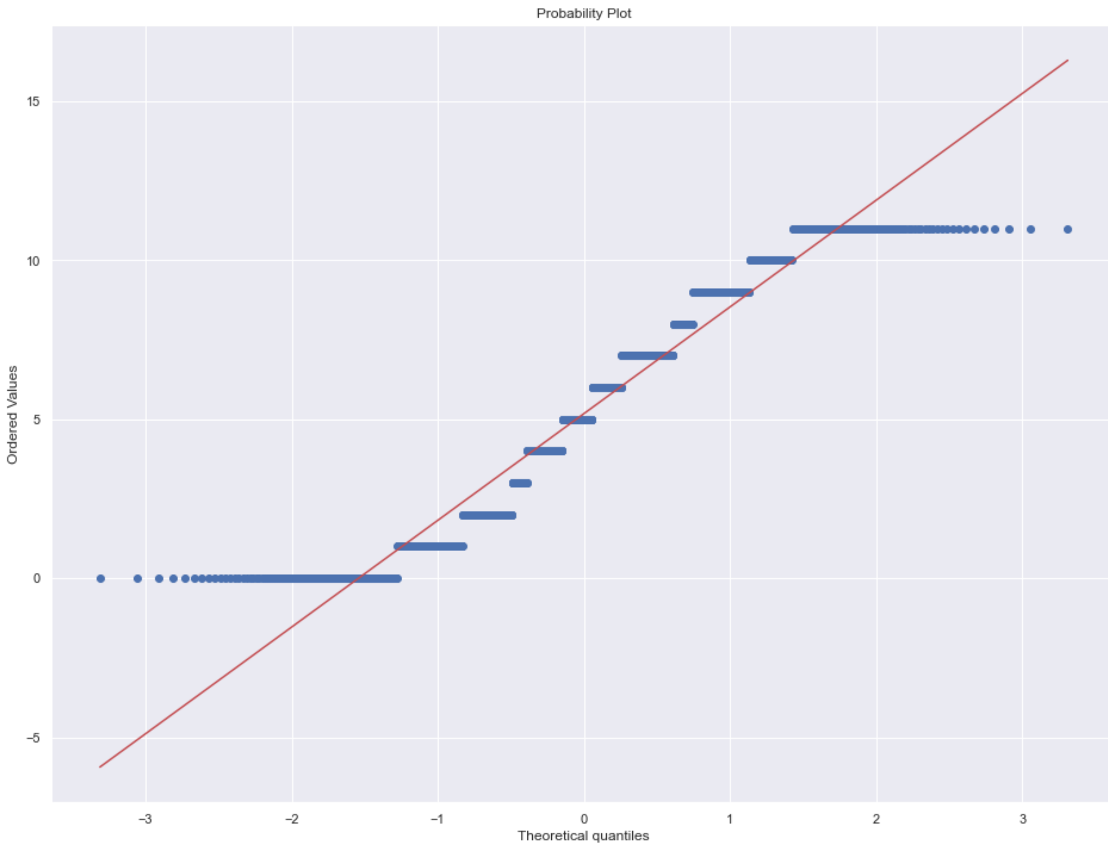
danceability



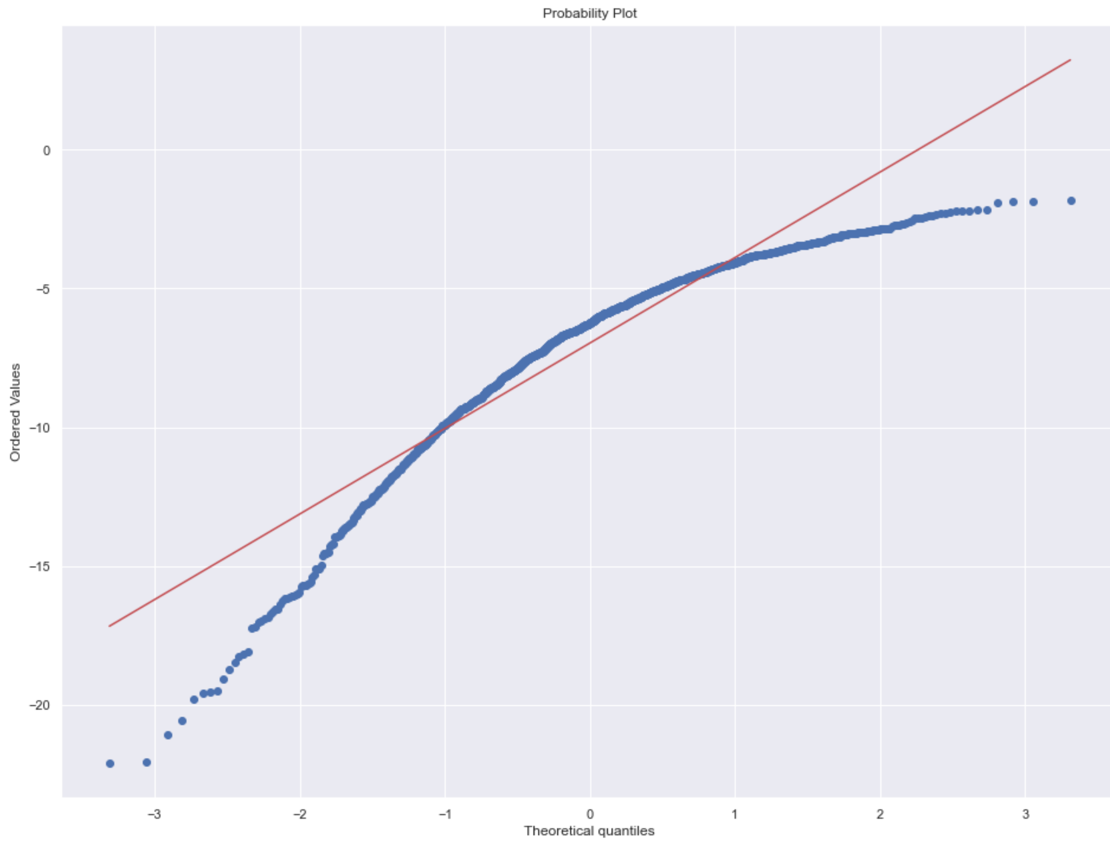
energy


Probability Plot
Ordered Values
Theoretical quantiles
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3

key

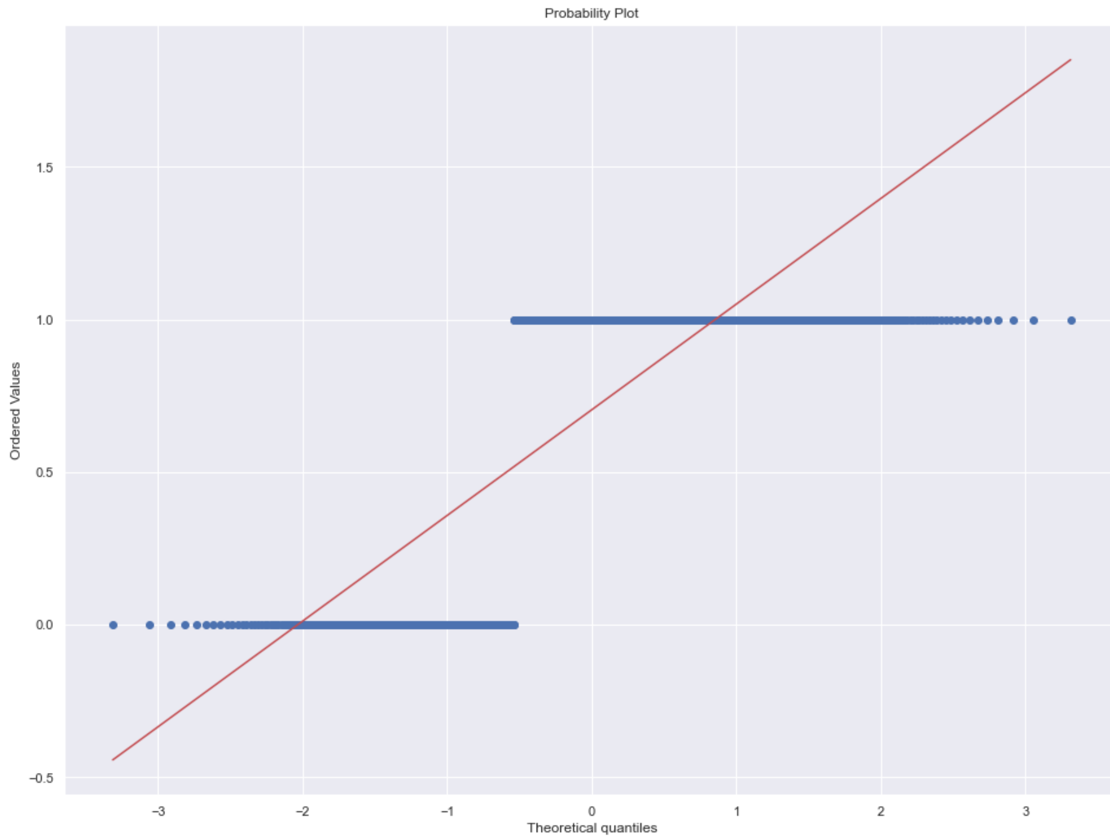

Probability Plot
15
10
5
0
−5
Ordered Values
−3
−2
−1
0
1
2
3
Theoretical quantiles

`loudness`

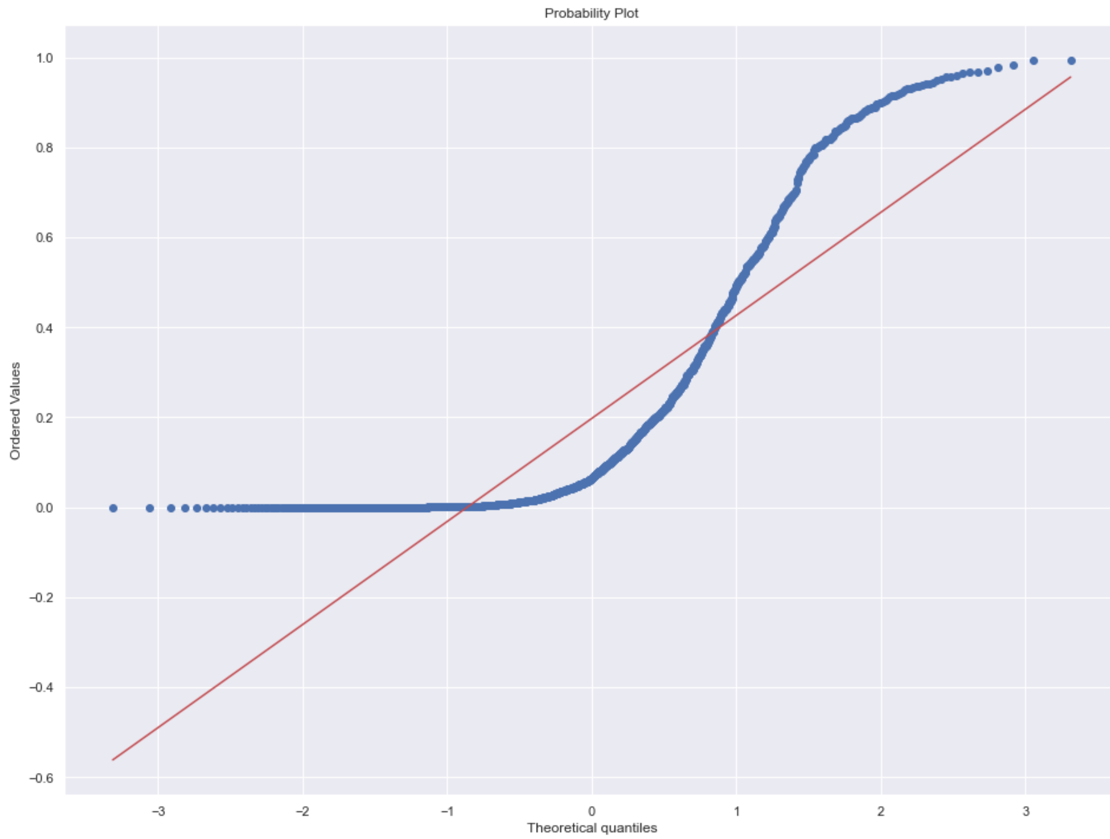


Probability Plot
Ordered Values
Theoretical quantiles
0
−5
−10
−15
−20
−3
−2
−1
0
1
2
3

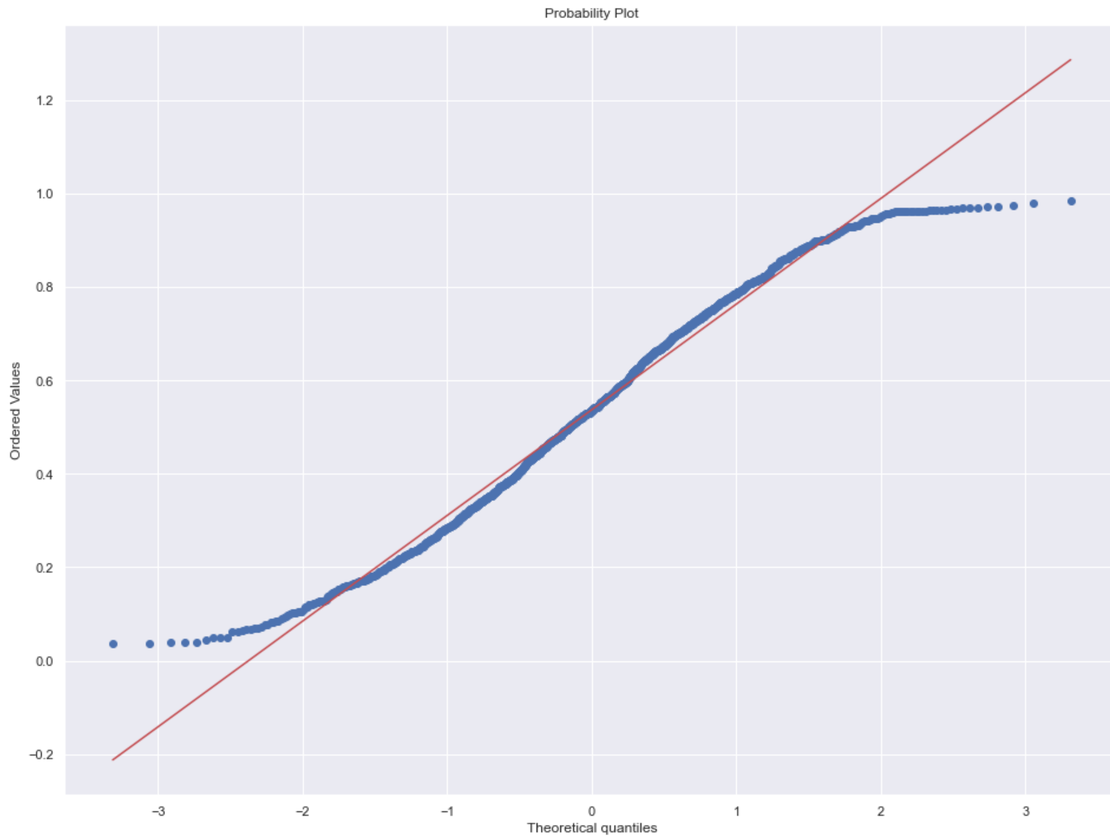
`mode`



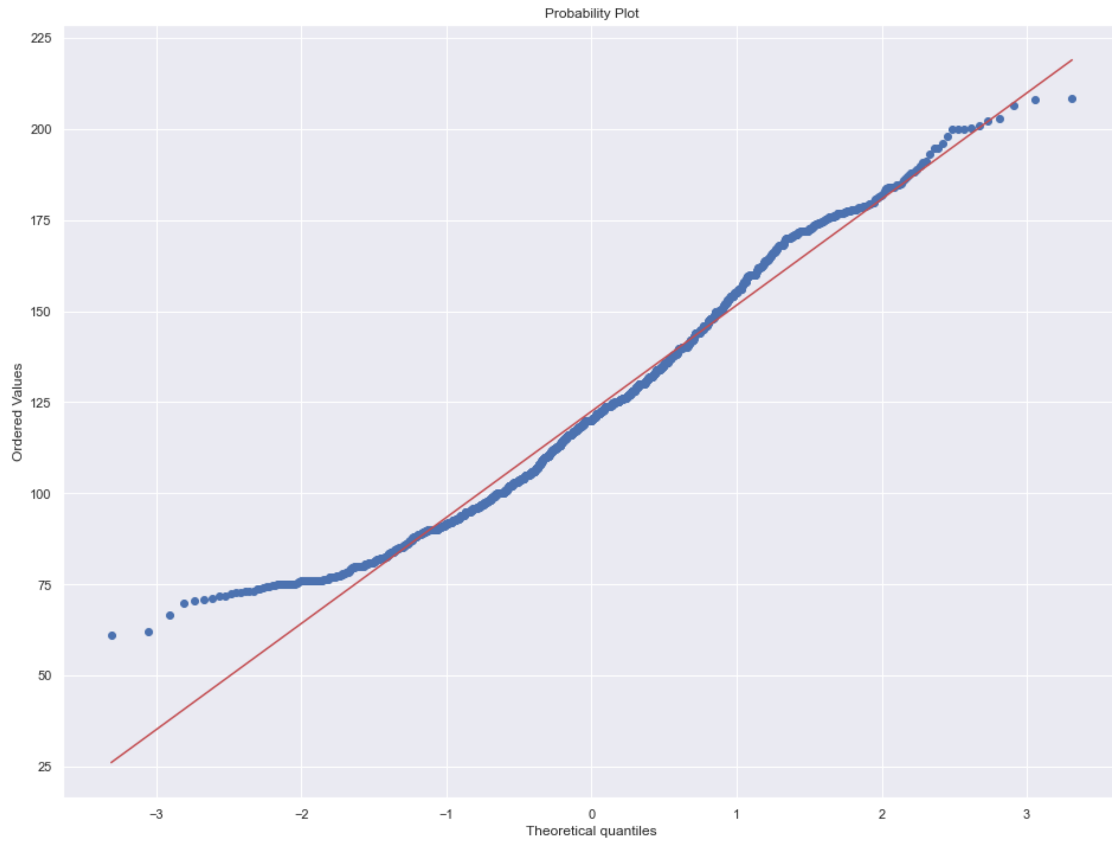
acousticness


Probability Plot
Ordered Values
Theoretical quantiles
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−3
−2
−1
0
1
2
3

`valence`


Probability Plot
Ordered Values
Theoretical quantiles
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−3
−2
−1
0
1
2
3

`tempo`



3.0.5 Fitting Distributions

Source:

1. https://stackoverflow.com/questions/6620471/fitting-empirical-distribution-to-theoretical-ones-with-scipy-python

[23]:
```
# Create models from data
def best_fit_distribution(data, bins=200, ax=None):
    """Model data by finding best fit distribution to data"""
    # Get histogram of original data
    y, x = np.histogram(data, bins=bins, density=True)
    x = (x + np.roll(x, -1))[:-1] / 2.0

    # Best holders
    best_distributions = []

    # Estimate distribution parameters from data
    for ii, distribution in enumerate([d for d in _distn_names if not d in␣
 ↪['levy_stable', 'studentized_range']]):
```

```
        print("{:>3} / {:<3}: {}".format( ii+1, len(_distn_names), distribution␣
↪))

        distribution = getattr(st, distribution)

        # Try to fit the distribution
        try:
            # Ignore warnings from data that can't be fit
            with warnings.catch_warnings():
                warnings.filterwarnings('ignore')

                # fit dist to data
                params = distribution.fit(data)

                # Separate parts of parameters
                arg = params[:-2]
                loc = params[-2]
                scale = params[-1]

                # Calculate fitted PDF and error with fit in distribution
                pdf = distribution.pdf(x, loc=loc, scale=scale, *arg)
                sse = np.sum(np.power(y - pdf, 2.0))

                # if axis pass in add to plot
                try:
                    if ax:
                        pd.Series(pdf, x).plot(ax=ax)
                    end
                except Exception:
                    pass

                # identify if this distribution is better
                best_distributions.append((distribution, params, sse))

        except Exception:
            pass


    return sorted(best_distributions, key=lambda x:x[2])

def make_pdf(dist, params, size=10000):
    """Generate distributions's Probability Distribution Function """

    # Separate parts of parameters
    arg = params[:-2]
    loc = params[-2]
    scale = params[-1]
```

```
    # Get sane start and end points of distribution
    start = dist.ppf(0.01, *arg, loc=loc, scale=scale) if arg else dist.ppf(0.
 ↪01, loc=loc, scale=scale)
    end = dist.ppf(0.99, *arg, loc=loc, scale=scale) if arg else dist.ppf(0.99,␣
 ↪loc=loc, scale=scale)

    # Build PDF and turn into pandas Series
    x = np.linspace(start, end, size)
    y = dist.pdf(x, loc=loc, scale=scale, *arg)
    pdf = pd.Series(y, x)

    return pdf
```

[24]:
```
def plot_distributions(var):

    # Load data from statsmodels datasets
    data = ldf[var]

    # Plot for comparison
    plt.figure(figsize=(12,8))
    ax = data.plot(kind='hist', bins=50, density=True, alpha=0.5,  ␣
 ↪color=list(matplotlib.rcParams['axes.prop_cycle'])[1]['color'])

    # Save plot limits
    dataYLim = ax.get_ylim()

    # Find best fit distribution
    best_distibutions = best_fit_distribution(data, 200, ax)
    best_dist = best_distibutions[0]

    # Update plots
    ax.set_ylim(dataYLim)
    ax.set_title(var+u'All Fitted Distributions')
    ax.set_xlabel(var)
    ax.set_ylabel('Frequency')

    # Make PDF with best params
    pdf = make_pdf(best_dist[0], best_dist[1])

    # Display
    plt.figure(figsize=(12,8))
    ax = pdf.plot(lw=2, label='PDF', legend=True)
    data.plot(kind='hist', bins=50, density=True, alpha=0.5, label='Data',␣
 ↪legend=True, ax=ax)
```

```
    param_names = (best_dist[0].shapes + ', loc, scale').split(', ') if␣
↪best_dist[0].shapes else ['loc', 'scale']
    param_str = ', '.join(['{}={:0.2f}'.format(k,v) for k,v in zip(param_names,␣
↪best_dist[1])])
    dist_str = '{}({})'.format(best_dist[0].name, param_str)

    ax.set_title(var + u'with best fit distribution \n' + dist_str)
    ax.set_xlabel(var)
    ax.set_ylabel('Frequency')
```

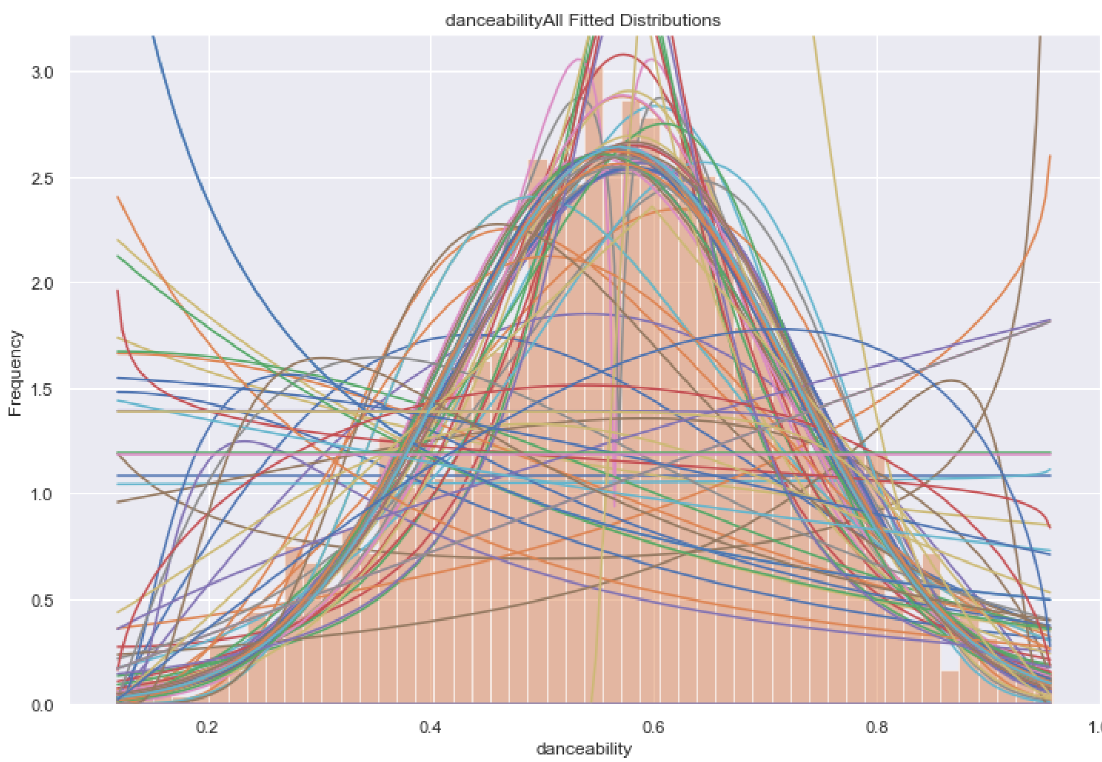
plotting for continuous vars

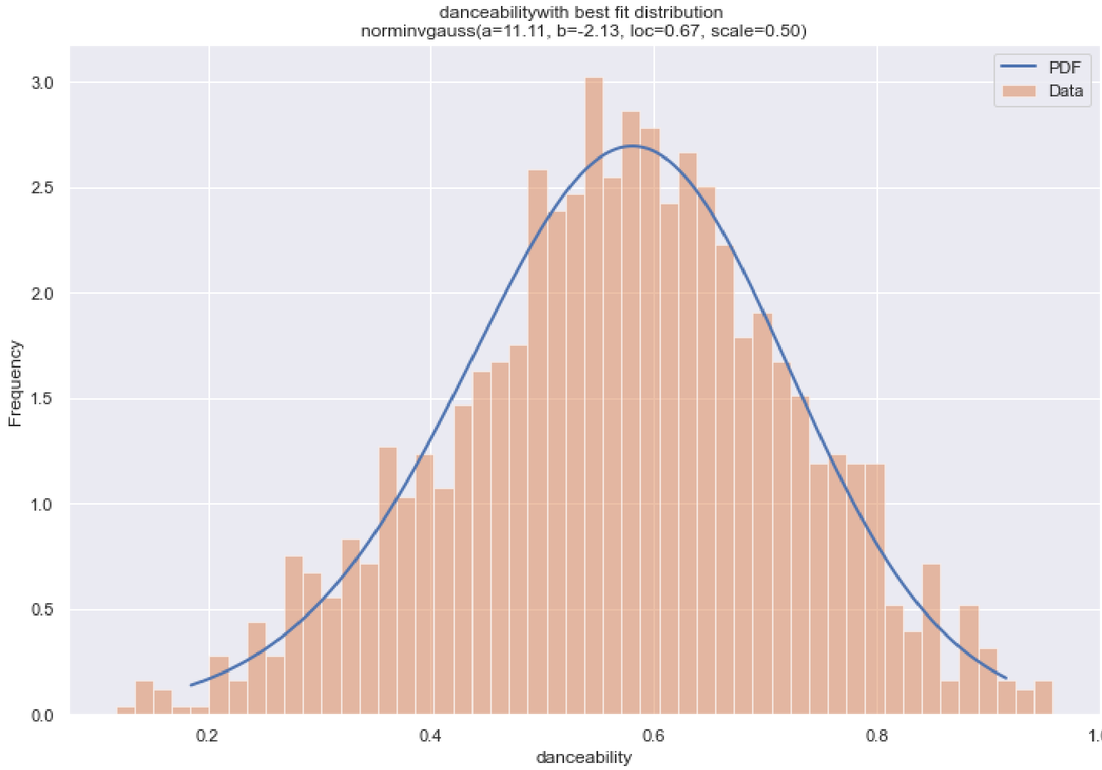
[25]:
```
print(vars[0])
plot_distributions(vars[0])
```

```
danceability
  1 / 104: ksone
  2 / 104: kstwo
  3 / 104: kstwobign
  4 / 104: norm
  5 / 104: alpha
  6 / 104: anglit
  7 / 104: arcsine
  8 / 104: beta
  9 / 104: betaprime
 10 / 104: bradford
 11 / 104: burr
 12 / 104: burr12
 13 / 104: fisk
 14 / 104: cauchy
 15 / 104: chi
 16 / 104: chi2
 17 / 104: cosine
 18 / 104: dgamma
 19 / 104: dweibull
 20 / 104: expon
 21 / 104: exponnorm
 22 / 104: exponweib
 23 / 104: exponpow
 24 / 104: fatiguelife
 25 / 104: foldcauchy
 26 / 104: f
 27 / 104: foldnorm
 28 / 104: weibull_min
 29 / 104: weibull_max
 30 / 104: genlogistic
 31 / 104: genpareto
 32 / 104: genexpon
```

```
33 / 104: genextreme
34 / 104: gamma
35 / 104: erlang
36 / 104: gengamma
37 / 104: genhalflogistic
38 / 104: genhyperbolic
39 / 104: gompertz
40 / 104: gumbel_r
41 / 104: gumbel_l
42 / 104: halfcauchy
43 / 104: halflogistic
44 / 104: halfnorm
45 / 104: hypsecant
46 / 104: gausshyper
47 / 104: invgamma
48 / 104: invgauss
49 / 104: geninvgauss
50 / 104: norminvgauss
51 / 104: invweibull
52 / 104: johnsonsb
53 / 104: johnsonsu
54 / 104: laplace
55 / 104: laplace_asymmetric
56 / 104: levy
57 / 104: levy_l
58 / 104: logistic
59 / 104: loggamma
60 / 104: loglaplace
61 / 104: lognorm
62 / 104: gilbrat
63 / 104: maxwell
64 / 104: mielke
65 / 104: kappa4
66 / 104: kappa3
67 / 104: moyal
68 / 104: nakagami
69 / 104: ncx2
70 / 104: ncf
71 / 104: t
72 / 104: nct
73 / 104: pareto
74 / 104: lomax
75 / 104: pearson3
76 / 104: powerlaw
77 / 104: powerlognorm
78 / 104: powernorm
79 / 104: rdist
80 / 104: rayleigh
```

```
 81 / 104: loguniform
 82 / 104: reciprocal
 83 / 104: rice
 84 / 104: recipinvgauss
 85 / 104: semicircular
 86 / 104: skewcauchy
 87 / 104: skewnorm
 88 / 104: trapezoid
 89 / 104: trapz
 90 / 104: triang
 91 / 104: truncexpon
 92 / 104: truncnorm
 93 / 104: tukeylambda
 94 / 104: uniform
 95 / 104: vonmises
 96 / 104: vonmises_line
 97 / 104: wald
 98 / 104: wrapcauchy
 99 / 104: gennorm
100 / 104: halfgennorm
101 / 104: crystalball
102 / 104: argus
```



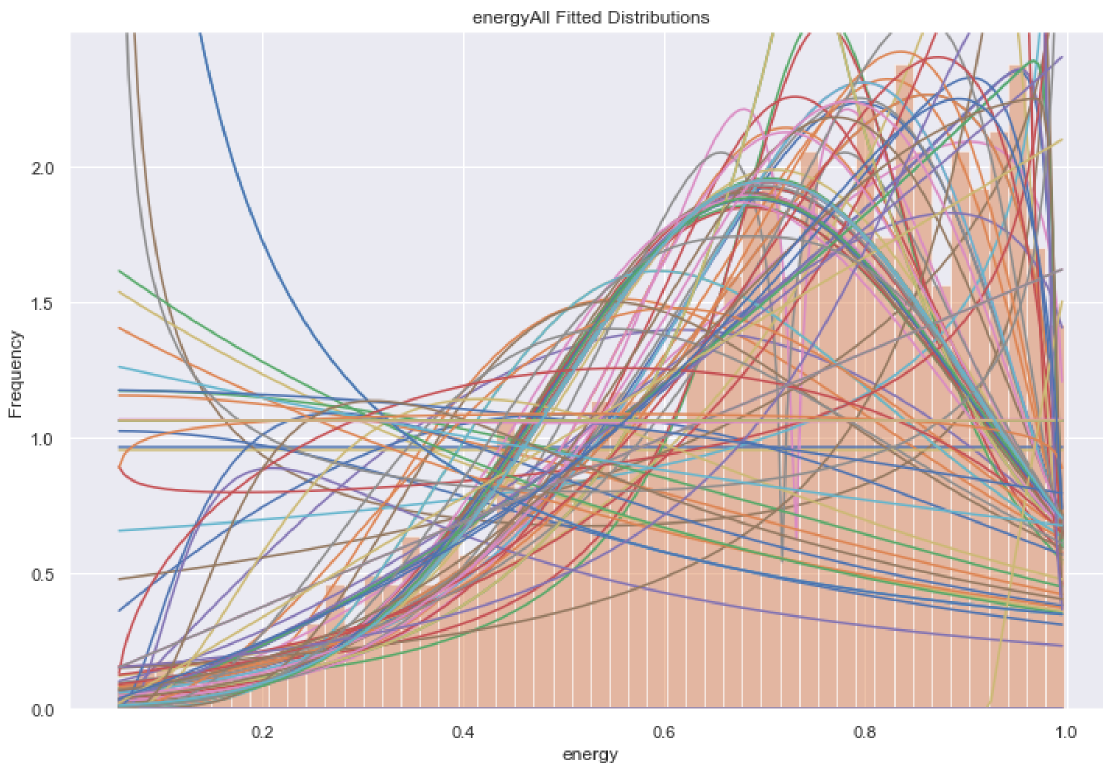


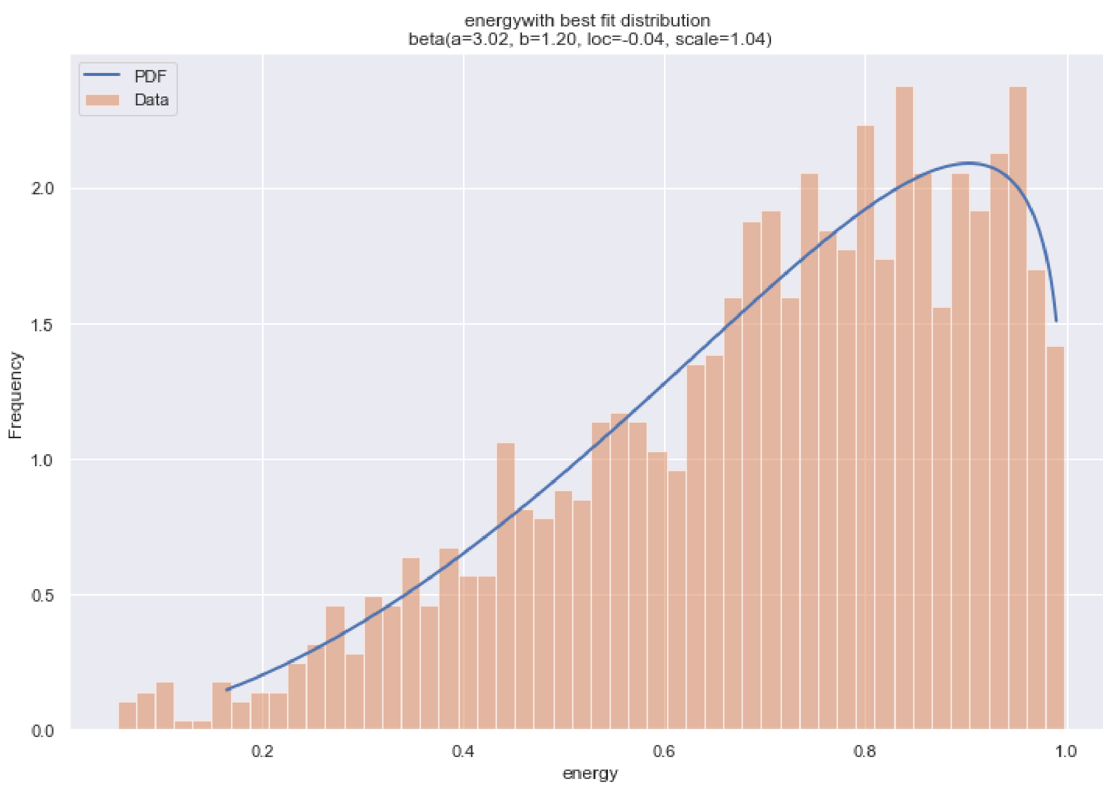
[26]:
```
print(vars[1])
plot_distributions(vars[1])
```

```
energy
  1 / 104: ksone
  2 / 104: kstwo
  3 / 104: kstwobign
  4 / 104: norm
  5 / 104: alpha
  6 / 104: anglit
  7 / 104: arcsine
  8 / 104: beta
  9 / 104: betaprime
 10 / 104: bradford
 11 / 104: burr
 12 / 104: burr12
 13 / 104: fisk
 14 / 104: cauchy
 15 / 104: chi
 16 / 104: chi2
 17 / 104: cosine
 18 / 104: dgamma
 19 / 104: dweibull
```

```
20 / 104: expon
21 / 104: exponnorm
22 / 104: exponweib
23 / 104: exponpow
24 / 104: fatiguelife
25 / 104: foldcauchy
26 / 104: f
27 / 104: foldnorm
28 / 104: weibull_min
29 / 104: weibull_max
30 / 104: genlogistic
31 / 104: genpareto
32 / 104: genexpon
33 / 104: genextreme
34 / 104: gamma
35 / 104: erlang
36 / 104: gengamma
37 / 104: genhalflogistic
38 / 104: genhyperbolic
39 / 104: gompertz
40 / 104: gumbel_r
41 / 104: gumbel_l
42 / 104: halfcauchy
43 / 104: halflogistic
44 / 104: halfnorm
45 / 104: hypsecant
46 / 104: gausshyper
47 / 104: invgamma
48 / 104: invgauss
49 / 104: geninvgauss
50 / 104: norminvgauss
51 / 104: invweibull
52 / 104: johnsonsb
53 / 104: johnsonsu
54 / 104: laplace
55 / 104: laplace_asymmetric
56 / 104: levy
57 / 104: levy_l
58 / 104: logistic
59 / 104: loggamma
60 / 104: loglaplace
61 / 104: lognorm
62 / 104: gilbrat
63 / 104: maxwell
64 / 104: mielke
65 / 104: kappa4
66 / 104: kappa3
67 / 104: moyal
```

```
 68 / 104: nakagami
 69 / 104: ncx2
 70 / 104: ncf
 71 / 104: t
 72 / 104: nct
 73 / 104: pareto
 74 / 104: lomax
 75 / 104: pearson3
 76 / 104: powerlaw
 77 / 104: powerlognorm
 78 / 104: powernorm
 79 / 104: rdist
 80 / 104: rayleigh
 81 / 104: loguniform
 82 / 104: reciprocal
 83 / 104: rice
 84 / 104: recipinvgauss
 85 / 104: semicircular
 86 / 104: skewcauchy
 87 / 104: skewnorm
 88 / 104: trapezoid
 89 / 104: trapz
 90 / 104: triang
 91 / 104: truncexpon
 92 / 104: truncnorm
 93 / 104: tukeylambda
 94 / 104: uniform
 95 / 104: vonmises
 96 / 104: vonmises_line
 97 / 104: wald
 98 / 104: wrapcauchy
 99 / 104: gennorm
100 / 104: halfgennorm
101 / 104: crystalball
102 / 104: argus
```


energyAll Fitted Distributions
Frequency
energy
0.0
0.5
1.0
1.5
2.0
0.2
0.4
0.6
0.8
1.0

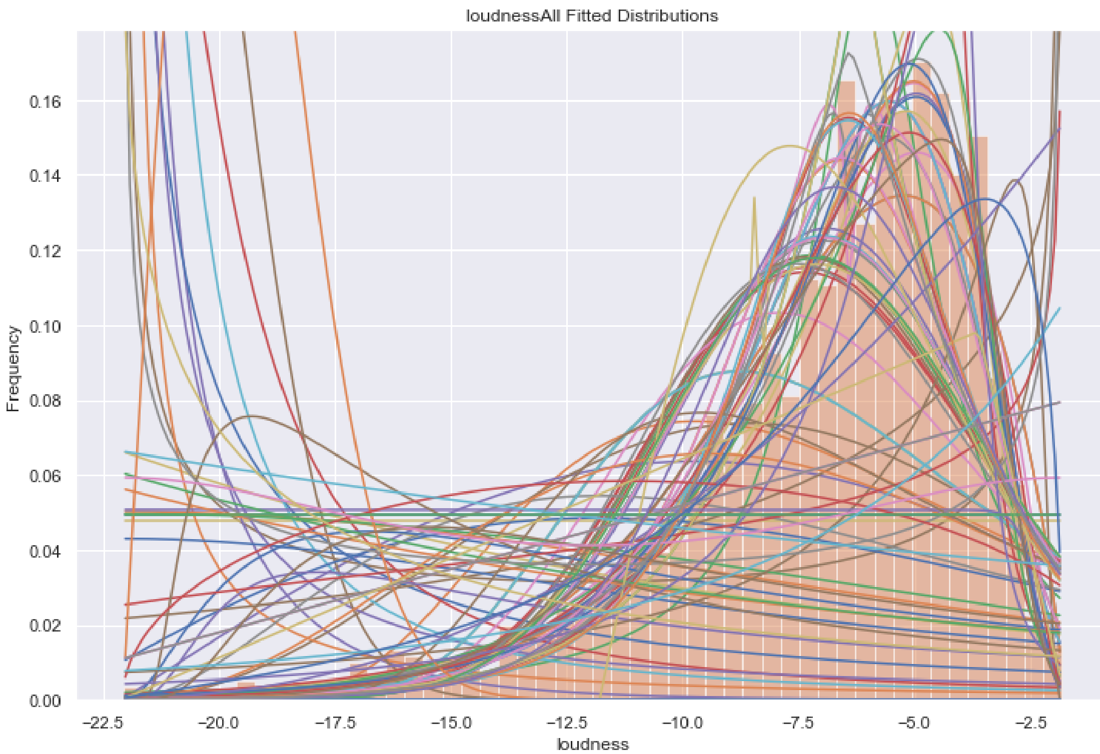

energywith best fit distribution
beta(a=3.02, b=1.20, loc=-0.04, scale=1.04)
PDF
Data
Frequency
energy
0.0
0.5
1.0
1.5
2.0
0.2
0.4
0.6
0.8
1.0

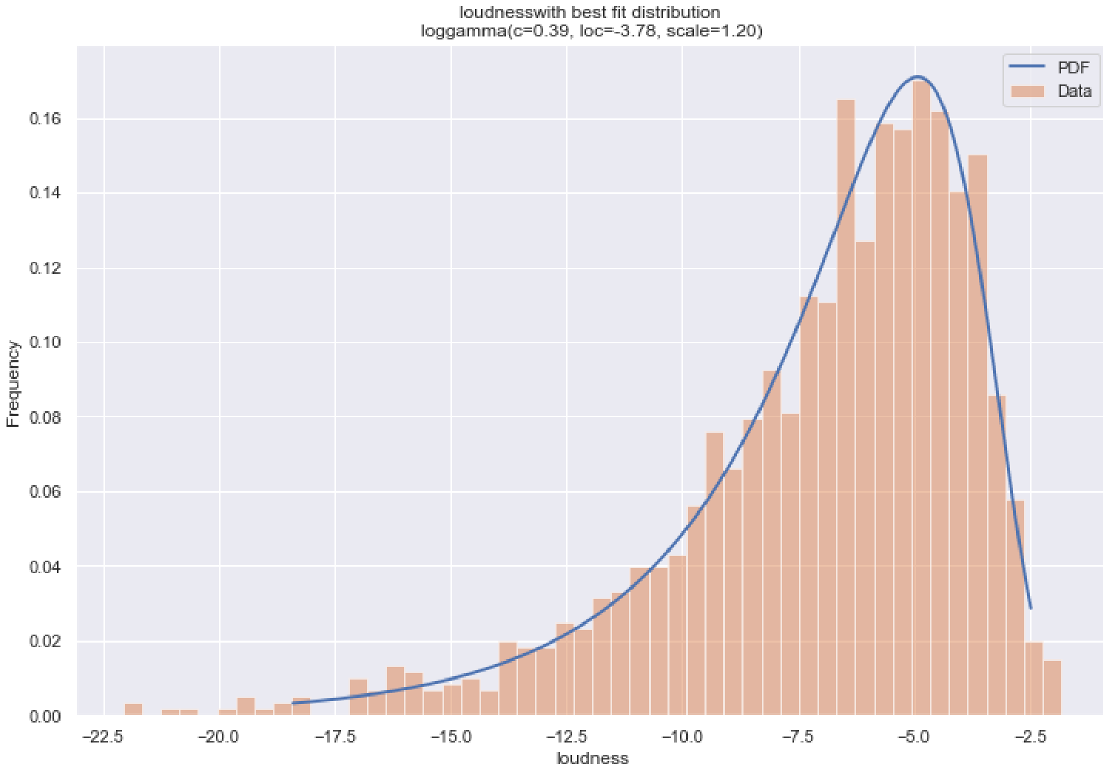
[27]:
```
print(vars[3])
plot_distributions(vars[3])
```

```
loudness
  1 / 104: ksone
  2 / 104: kstwo
  3 / 104: kstwobign
  4 / 104: norm
  5 / 104: alpha
  6 / 104: anglit
  7 / 104: arcsine
  8 / 104: beta
  9 / 104: betaprime
 10 / 104: bradford
 11 / 104: burr
 12 / 104: burr12
 13 / 104: fisk
 14 / 104: cauchy
 15 / 104: chi
 16 / 104: chi2
 17 / 104: cosine
 18 / 104: dgamma
 19 / 104: dweibull
 20 / 104: expon
 21 / 104: exponnorm
 22 / 104: exponweib
 23 / 104: exponpow
 24 / 104: fatiguelife
 25 / 104: foldcauchy
 26 / 104: f
 27 / 104: foldnorm
 28 / 104: weibull_min
 29 / 104: weibull_max
 30 / 104: genlogistic
 31 / 104: genpareto
 32 / 104: genexpon
 33 / 104: genextreme
 34 / 104: gamma
 35 / 104: erlang
 36 / 104: gengamma
 37 / 104: genhalflogistic
 38 / 104: genhyperbolic
 39 / 104: gompertz
 40 / 104: gumbel_r
 41 / 104: gumbel_l
 42 / 104: halfcauchy
```

```
43 / 104: halflogistic
44 / 104: halfnorm
45 / 104: hypsecant
46 / 104: gausshyper
47 / 104: invgamma
48 / 104: invgauss
49 / 104: geninvgauss
50 / 104: norminvgauss
51 / 104: invweibull
52 / 104: johnsonsb
53 / 104: johnsonsu
54 / 104: laplace
55 / 104: laplace_asymmetric
56 / 104: levy
57 / 104: levy_l
58 / 104: logistic
59 / 104: loggamma
60 / 104: loglaplace
61 / 104: lognorm
62 / 104: gilbrat
63 / 104: maxwell
64 / 104: mielke
65 / 104: kappa4
66 / 104: kappa3
67 / 104: moyal
68 / 104: nakagami
69 / 104: ncx2
70 / 104: ncf
71 / 104: t
72 / 104: nct
73 / 104: pareto
74 / 104: lomax
75 / 104: pearson3
76 / 104: powerlaw
77 / 104: powerlognorm
78 / 104: powernorm
79 / 104: rdist
80 / 104: rayleigh
81 / 104: loguniform
82 / 104: reciprocal
83 / 104: rice
84 / 104: recipinvgauss
85 / 104: semicircular
86 / 104: skewcauchy
87 / 104: skewnorm
88 / 104: trapezoid
89 / 104: trapz
90 / 104: triang
```

```
 91 / 104: truncexpon
 92 / 104: truncnorm
 93 / 104: tukeylambda
 94 / 104: uniform
 95 / 104: vonmises
 96 / 104: vonmises_line
 97 / 104: wald
 98 / 104: wrapcauchy
 99 / 104: gennorm
100 / 104: halfgennorm
101 / 104: crystalball
102 / 104: argus
```



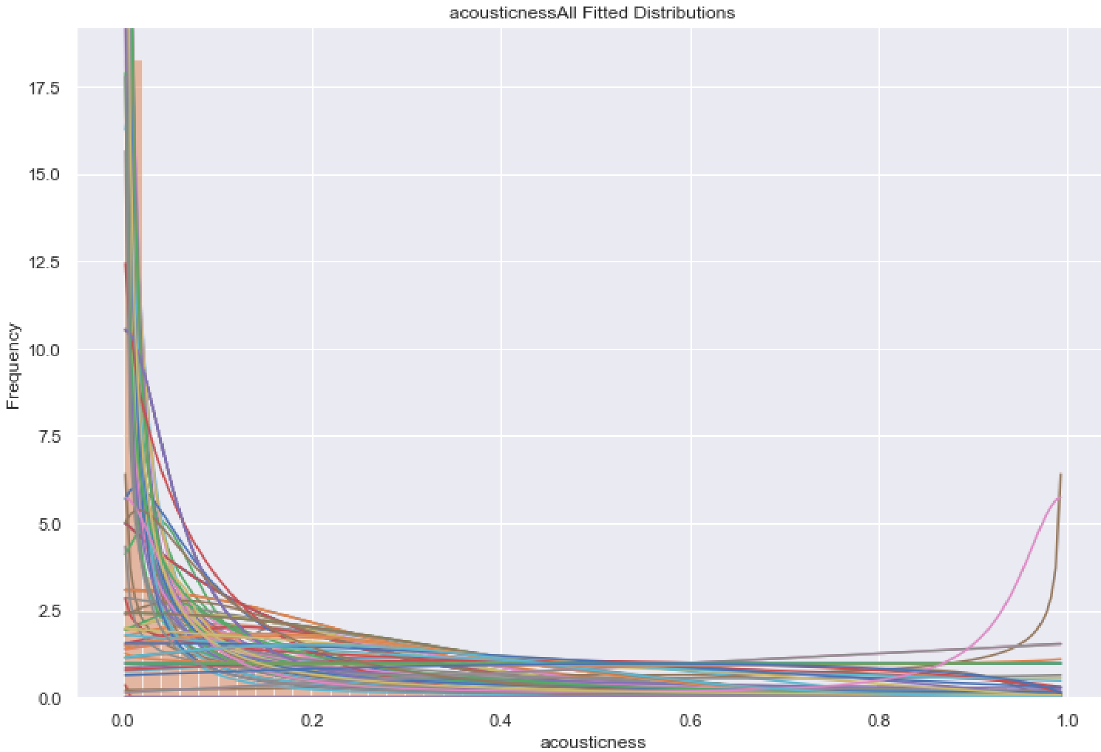


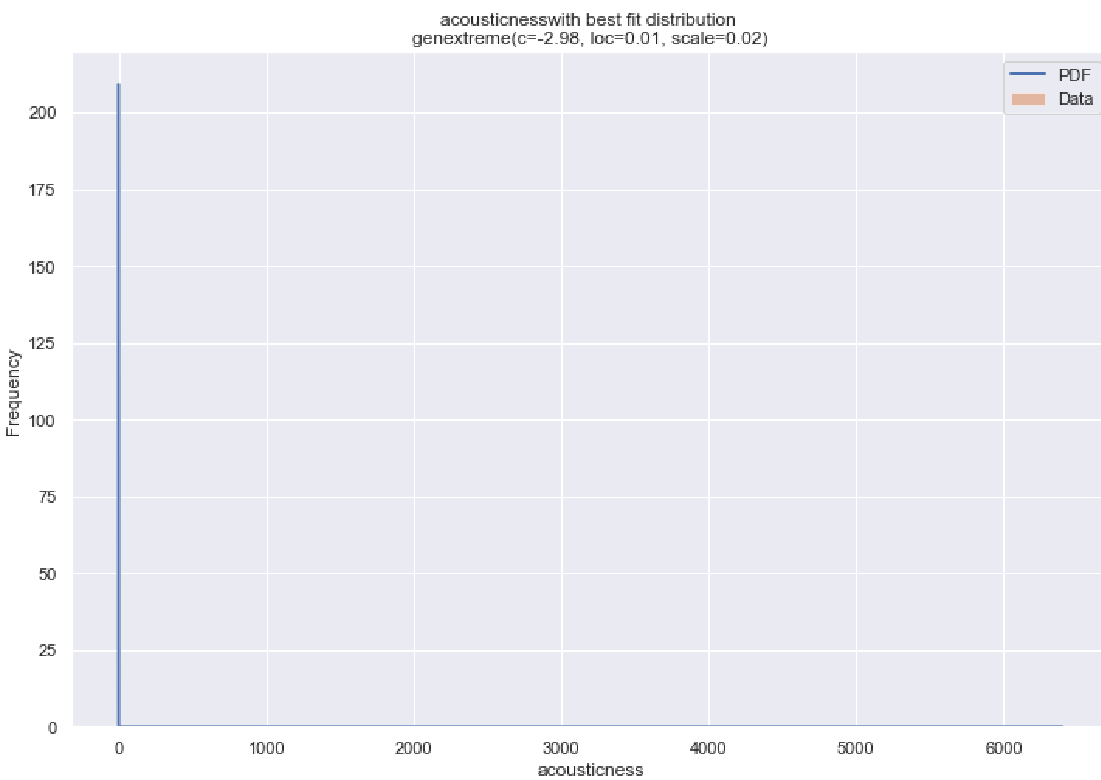
[28]:
```
print(vars[5])
plot_distributions(vars[5])
```

```
acousticness
  1 / 104: ksone
  2 / 104: kstwo
  3 / 104: kstwobign
  4 / 104: norm
  5 / 104: alpha
  6 / 104: anglit
  7 / 104: arcsine
  8 / 104: beta
  9 / 104: betaprime
 10 / 104: bradford
 11 / 104: burr
 12 / 104: burr12
 13 / 104: fisk
 14 / 104: cauchy
 15 / 104: chi
 16 / 104: chi2
 17 / 104: cosine
 18 / 104: dgamma
 19 / 104: dweibull
```

```
20 / 104: expon
21 / 104: exponnorm
22 / 104: exponweib
23 / 104: exponpow
24 / 104: fatiguelife
25 / 104: foldcauchy
26 / 104: f
27 / 104: foldnorm
28 / 104: weibull_min
29 / 104: weibull_max
30 / 104: genlogistic
31 / 104: genpareto
32 / 104: genexpon
33 / 104: genextreme
34 / 104: gamma
35 / 104: erlang
36 / 104: gengamma
37 / 104: genhalflogistic
38 / 104: genhyperbolic
39 / 104: gompertz
40 / 104: gumbel_r
41 / 104: gumbel_l
42 / 104: halfcauchy
43 / 104: halflogistic
44 / 104: halfnorm
45 / 104: hypsecant
46 / 104: gausshyper
47 / 104: invgamma
48 / 104: invgauss
49 / 104: geninvgauss
50 / 104: norminvgauss
51 / 104: invweibull
52 / 104: johnsonsb
53 / 104: johnsonsu
54 / 104: laplace
55 / 104: laplace_asymmetric
56 / 104: levy
57 / 104: levy_l
58 / 104: logistic
59 / 104: loggamma
60 / 104: loglaplace
61 / 104: lognorm
62 / 104: gilbrat
63 / 104: maxwell
64 / 104: mielke
65 / 104: kappa4
66 / 104: kappa3
67 / 104: moyal
```

```
 68 / 104: nakagami
 69 / 104: ncx2
 70 / 104: ncf
 71 / 104: t
 72 / 104: nct
 73 / 104: pareto
 74 / 104: lomax
 75 / 104: pearson3
 76 / 104: powerlaw
 77 / 104: powerlognorm
 78 / 104: powernorm
 79 / 104: rdist
 80 / 104: rayleigh
 81 / 104: loguniform
 82 / 104: reciprocal
 83 / 104: rice
 84 / 104: recipinvgauss
 85 / 104: semicircular
 86 / 104: skewcauchy
 87 / 104: skewnorm
 88 / 104: trapezoid
 89 / 104: trapz
 90 / 104: triang
 91 / 104: truncexpon
 92 / 104: truncnorm
 93 / 104: tukeylambda
 94 / 104: uniform
 95 / 104: vonmises
 96 / 104: vonmises_line
 97 / 104: wald
 98 / 104: wrapcauchy
 99 / 104: gennorm
100 / 104: halfgennorm
101 / 104: crystalball
102 / 104: argus
```


acousticnessAll Fitted Distributions
Frequency
acousticness
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
0.2
0.4
0.6
0.8
1.0

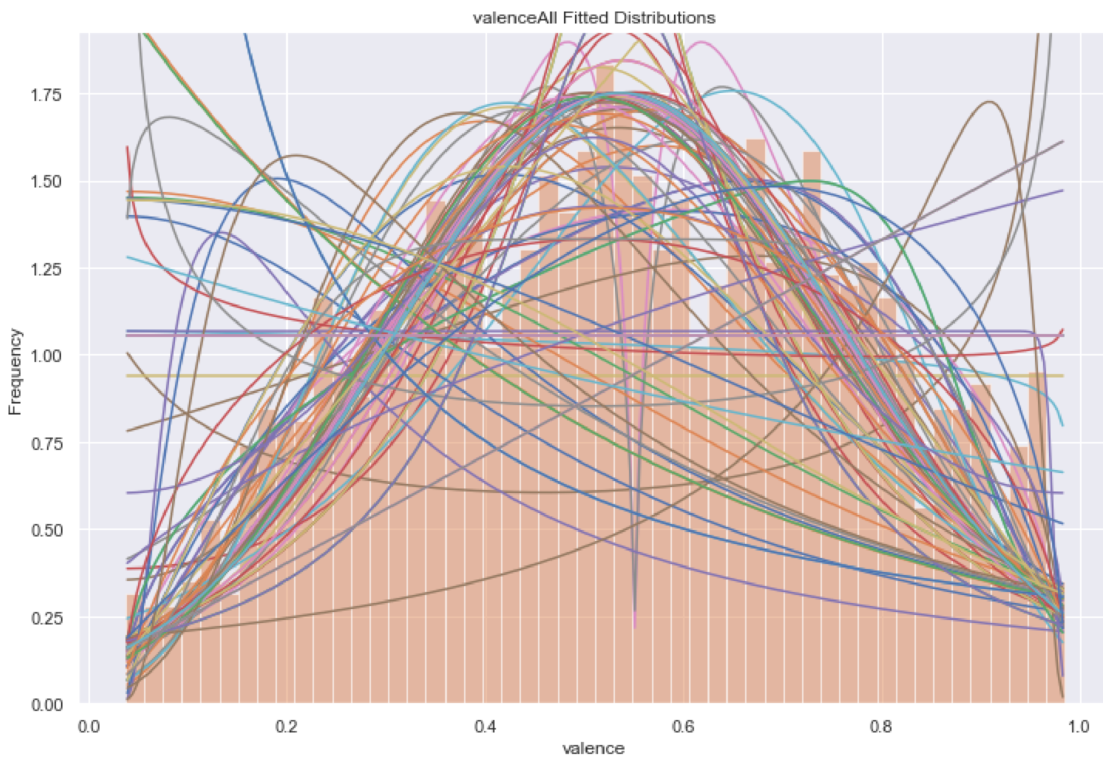

acousticnesswith best fit distribution
genextreme(c=-2.98, loc=0.01, scale=0.02)
PDF
Data
Frequency
acousticness
0
25
50
75
100
125
150
175
200
0
1000
2000
3000
4000
5000
6000

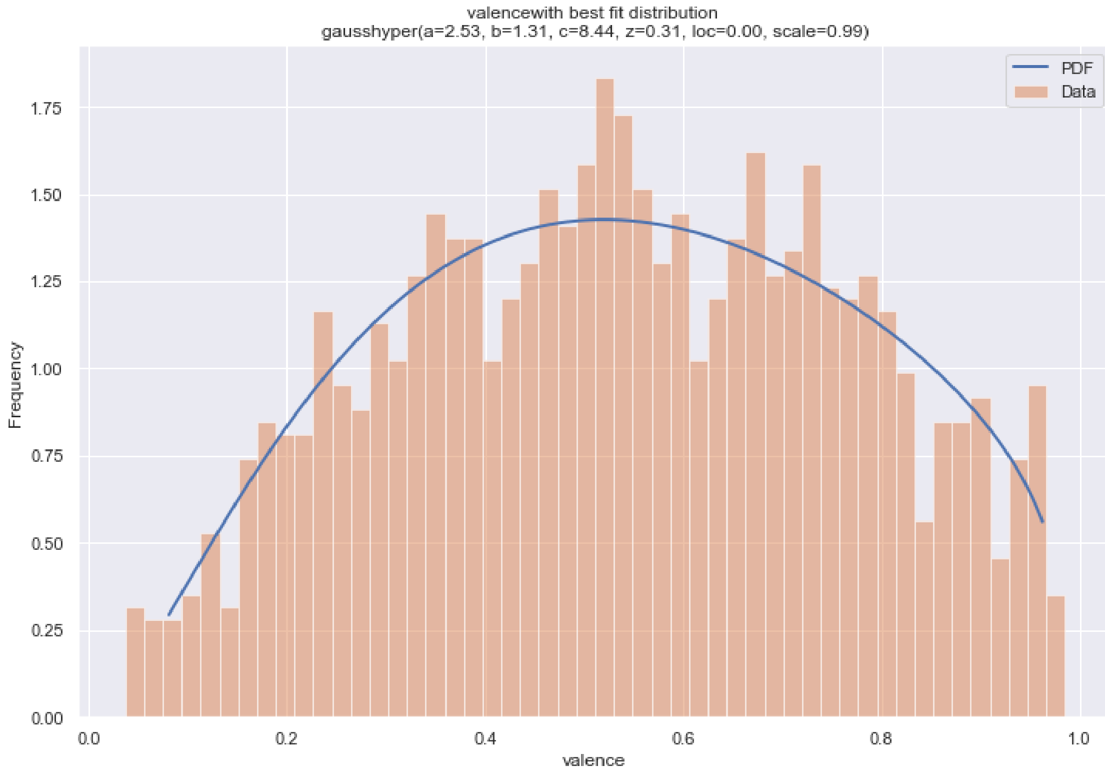
[29]:
```
print(vars[6])
plot_distributions(vars[6])
```

```
valence
  1 / 104: ksone
  2 / 104: kstwo
  3 / 104: kstwobign
  4 / 104: norm
  5 / 104: alpha
  6 / 104: anglit
  7 / 104: arcsine
  8 / 104: beta
  9 / 104: betaprime
 10 / 104: bradford
 11 / 104: burr
 12 / 104: burr12
 13 / 104: fisk
 14 / 104: cauchy
 15 / 104: chi
 16 / 104: chi2
 17 / 104: cosine
 18 / 104: dgamma
 19 / 104: dweibull
 20 / 104: expon
 21 / 104: exponnorm
 22 / 104: exponweib
 23 / 104: exponpow
 24 / 104: fatiguelife
 25 / 104: foldcauchy
 26 / 104: f
 27 / 104: foldnorm
 28 / 104: weibull_min
 29 / 104: weibull_max
 30 / 104: genlogistic
 31 / 104: genpareto
 32 / 104: genexpon
 33 / 104: genextreme
 34 / 104: gamma
 35 / 104: erlang
 36 / 104: gengamma
 37 / 104: genhalflogistic
 38 / 104: genhyperbolic
 39 / 104: gompertz
 40 / 104: gumbel_r
 41 / 104: gumbel_l
 42 / 104: halfcauchy
```

```
43 / 104: halflogistic
44 / 104: halfnorm
45 / 104: hypsecant
46 / 104: gausshyper
47 / 104: invgamma
48 / 104: invgauss
49 / 104: geninvgauss
50 / 104: norminvgauss
51 / 104: invweibull
52 / 104: johnsonsb
53 / 104: johnsonsu
54 / 104: laplace
55 / 104: laplace_asymmetric
56 / 104: levy
57 / 104: levy_l
58 / 104: logistic
59 / 104: loggamma
60 / 104: loglaplace
61 / 104: lognorm
62 / 104: gilbrat
63 / 104: maxwell
64 / 104: mielke
65 / 104: kappa4
66 / 104: kappa3
67 / 104: moyal
68 / 104: nakagami
69 / 104: ncx2
70 / 104: ncf
71 / 104: t
72 / 104: nct
73 / 104: pareto
74 / 104: lomax
75 / 104: pearson3
76 / 104: powerlaw
77 / 104: powerlognorm
78 / 104: powernorm
79 / 104: rdist
80 / 104: rayleigh
81 / 104: loguniform
82 / 104: reciprocal
83 / 104: rice
84 / 104: recipinvgauss
85 / 104: semicircular
86 / 104: skewcauchy
87 / 104: skewnorm
88 / 104: trapezoid
89 / 104: trapz
90 / 104: triang
```

```
 91 / 104: truncexpon
 92 / 104: truncnorm
 93 / 104: tukeylambda
 94 / 104: uniform
 95 / 104: vonmises
 96 / 104: vonmises_line
 97 / 104: wald
 98 / 104: wrapcauchy
 99 / 104: gennorm
100 / 104: halfgennorm
101 / 104: crystalball
102 / 104: argus
```





valencewith best fit distribution
gausshyper(a=2.53, b=1.31, c=8.44, z=0.31, loc=0.00, scale=0.99)
PDF
Data
Frequency
valence
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0.0
0.2
0.4
0.6
0.8
1.0


```
[ ]:
```