Spotify Exploratory Data Analysis

What makes a song popular?

Question – What makes a song popular?

- Spotify has a Python library that provides data on all its tracks – including audio features like tempo, key, ‘danceability’, etc.
- We will use this data to attempt to determine what features have the largest impact on the overall popularity of a song

Variables used for analysis

- “Danceability” – describes how suitable a track is for dancing
- “Energy” – represents a perceptual measure of intensity and activity (high energy tracks are loud and noisy)
- ”Loudness” – overall loudness of track in decibels
- “Valence” – describes musical positiveness (High valence tracks sound happy, low valence tracks sound sad)
- “Instrumentalness” – predicts whether a track contains vocals
- “Key” – musical key the track is in
- “Mode” – whether the track is in a major or minor key
- “Popularity” – overall popularity of track

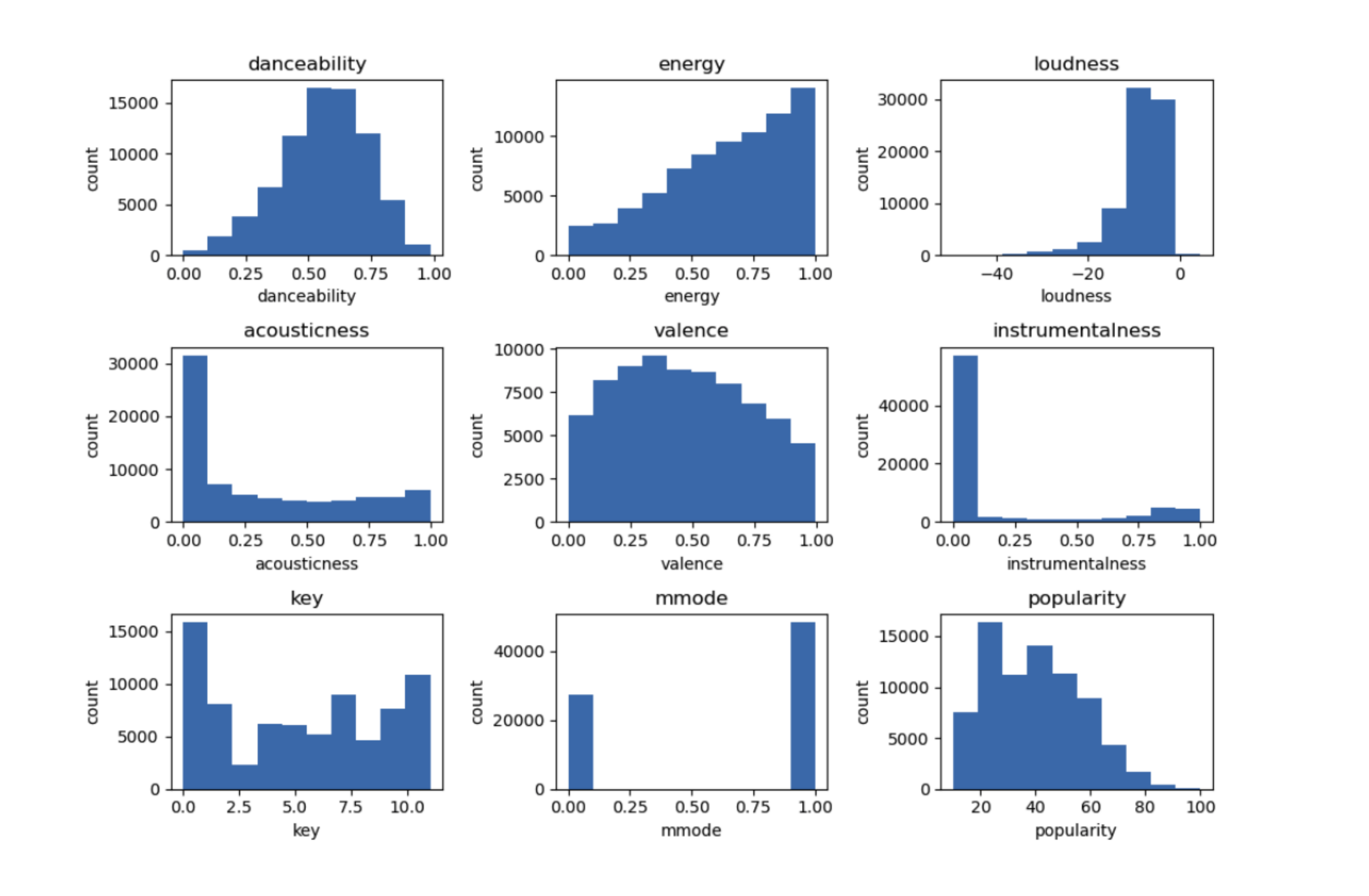
Histograms

```
df = df.dropna()
features = ['danceability', 'energy', 'loudness', 'acousticness', 'valence', 'instrumentalness', 'key', 'mmode', 'popularity']
df = df.rename(columns={'mode': 'mmode'}, )
df = df.drop(df[df['popularity'] < 10].index)
plt.figure(figsize=(10, 7))

for i, feature in enumerate(features, 1):
    plt.subplot(3, 3, i)
    plt.hist(df[feature])
    plt.title(f'{feature}')
    plt.xlabel(feature)
    plt.ylabel('count')
plt.tight_layout()
plt.show()
```

✓ 0.4s

Because there are so many tracks included, many of them are relatively unknown and have a popularity of 0, which was skewing the data. To get a more accurate model we dropped all tracks with a popularity less than 1, limiting the tracks to those with at least some exposure. There are not any other apparent outliers in the histograms. It is worth noting that the variables ”key” and “mode” are categorical variables, not continuous variables.

Histograms



danceability
energy
loudness
acousticness
valence
instrumentalness
key
mmode
popularity
count

Descriptive characteristics

```
count    75875.000000
mean         0.559842
std          0.173756
min          0.000000
25%          0.451000
50%          0.573000
75%          0.687000
max          0.985000
Name: danceability, dtype: float64
```

```
count    75875.000000
mean         0.642593
std          0.254161
min          0.000000
25%          0.467000
50%          0.684000
75%          0.860000
max          1.000000
Name: energy, dtype: float64
```

```
count    75875.000000
mean        -8.384520
std          5.118226
min        -49.531000
25%        -10.173000
50%         -7.150000
75%         -5.090000
max          4.532000
Name: loudness, dtype: float64
```

```
count    75875.000000
mean         0.323324
std          0.333905
min          0.000000
25%          0.015600
50%          0.187000
75%          0.614000
max          0.996000
Name: acousticness, dtype: float64
```

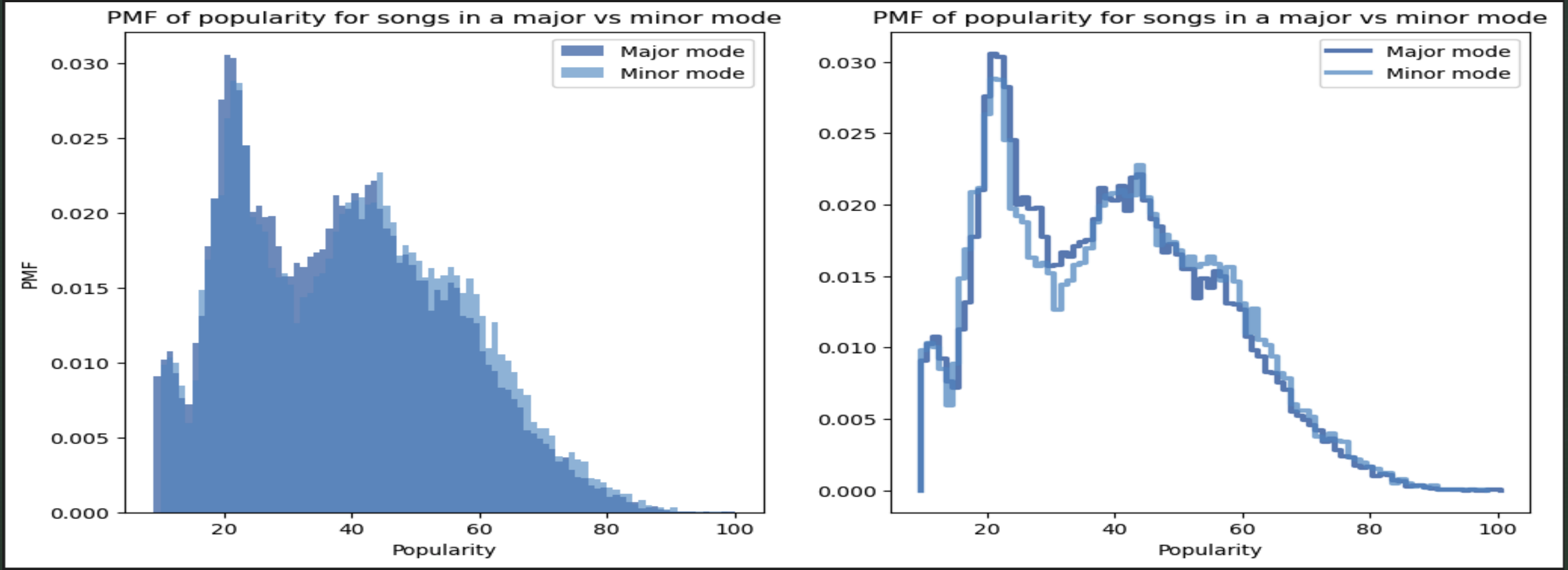
```
count    75875.000000
mean         0.466925
std          0.261832
min          0.000000
25%          0.248000
50%          0.453000
75%          0.678000
max          0.995000
Name: valence, dtype: float64
```

```
count    75875.000000
mean         0.170741
std          0.320839
min          0.000000
25%          0.000000
50%          0.000062
75%          0.091000
max          1.000000
Name: instrumentalness, dtype: float64
```

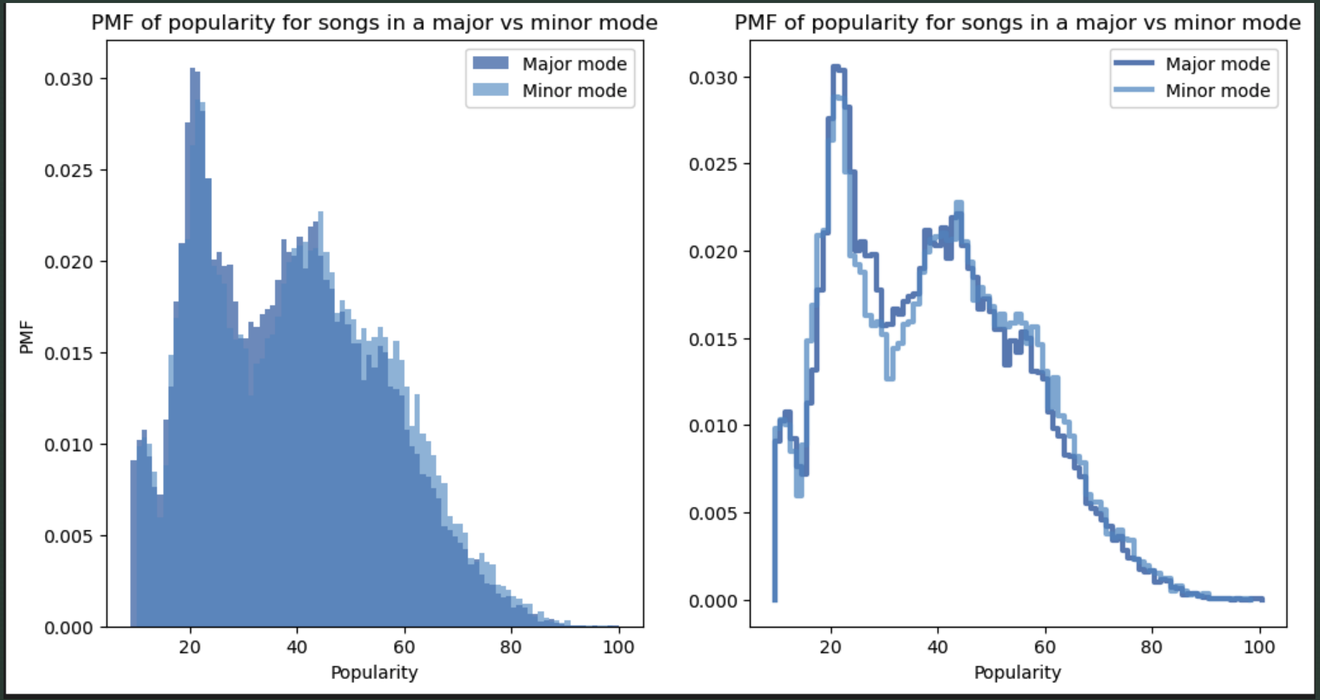
PMF comparison

```
major_key = df[df.mmode == 1]
minor_key = df[df.mmode == 0]
pmf = thinkstats2.Pmf(major_key.popularity, label='Major mode')
pmf2 = thinkstats2.Pmf(minor_key.popularity, label='Minor mode')
width = 1
thinkplot.PrePlot(2, cols=2)
thinkplot.Hist(pmf, align="right", width=width)
thinkplot.Hist(pmf2, align="left", width=width)
thinkplot.Config(xlabel="Popularity", ylabel="PMF", title="PMF of popularity for songs in a major vs minor mode")

thinkplot.PrePlot(2)
thinkplot.SubPlot(2)
thinkplot.Pmfs([pmf, pmf2], width=1)
thinkplot.Config(xlabel="Popularity", title="PMF of popularity for songs in a major vs minor mode")
```

✓ 0.1s



PMF comparison

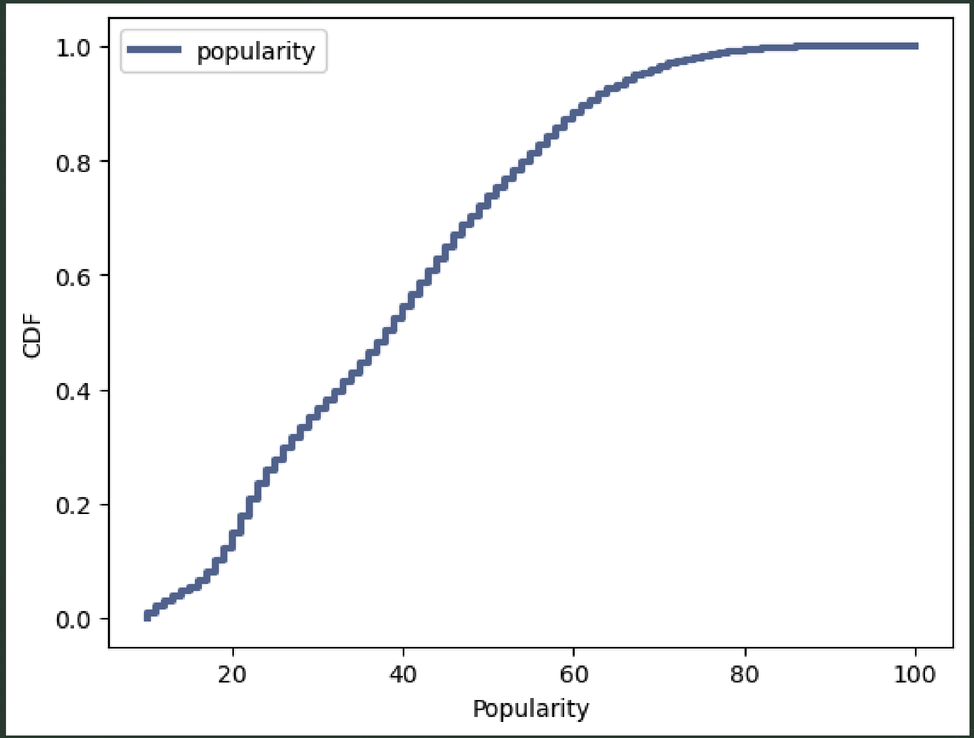


Looking at the charts for PMF of track popularity for songs in a major key vs minor key, it does not appear that there is a significant difference in probability for songs in a major or minor key to be more popular. We do see that the probability is higher for minor key songs at the right end of the graph, but not by much. This might suggest that songs in a minor key are slightly more likely to be popular.

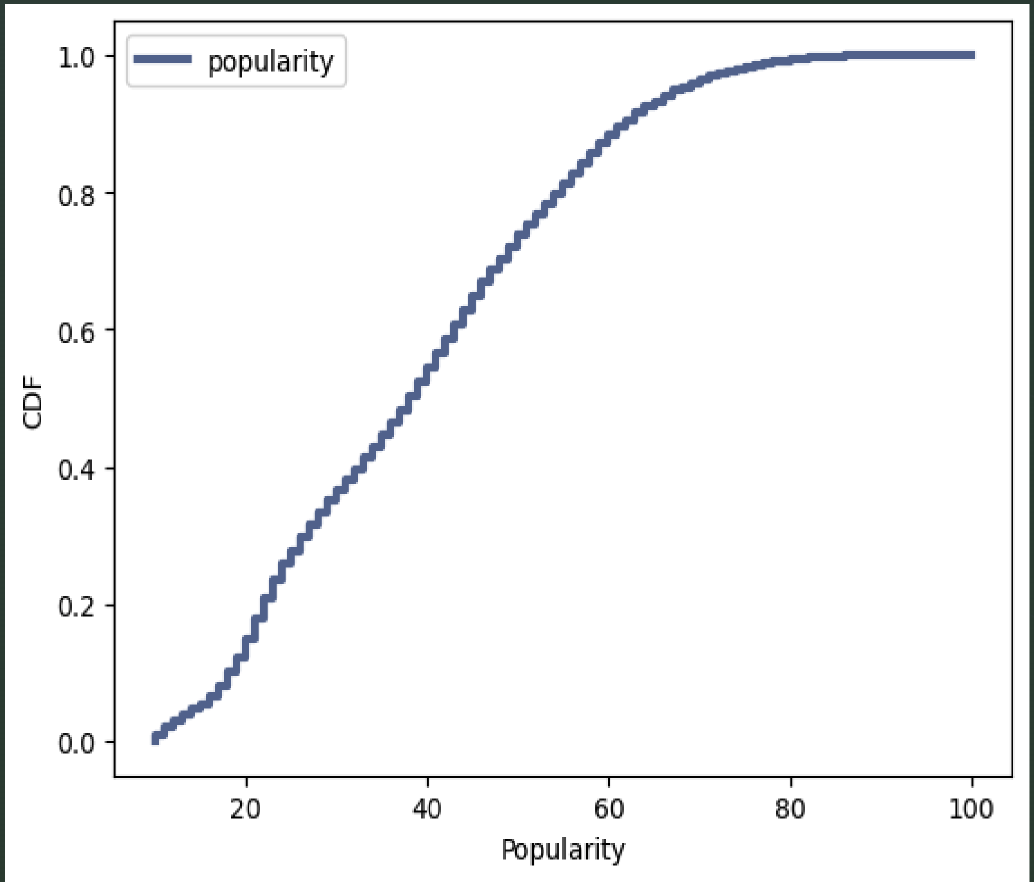
CDF

```
cdf = thinkstats2.Cdf(df.popularity, label='popularity')
thinkplot.Cdf(cdf)
thinkplot.Config(xlabel='Popularity', ylabel='CDF', loc='upper left')
```

✓ 0.0s



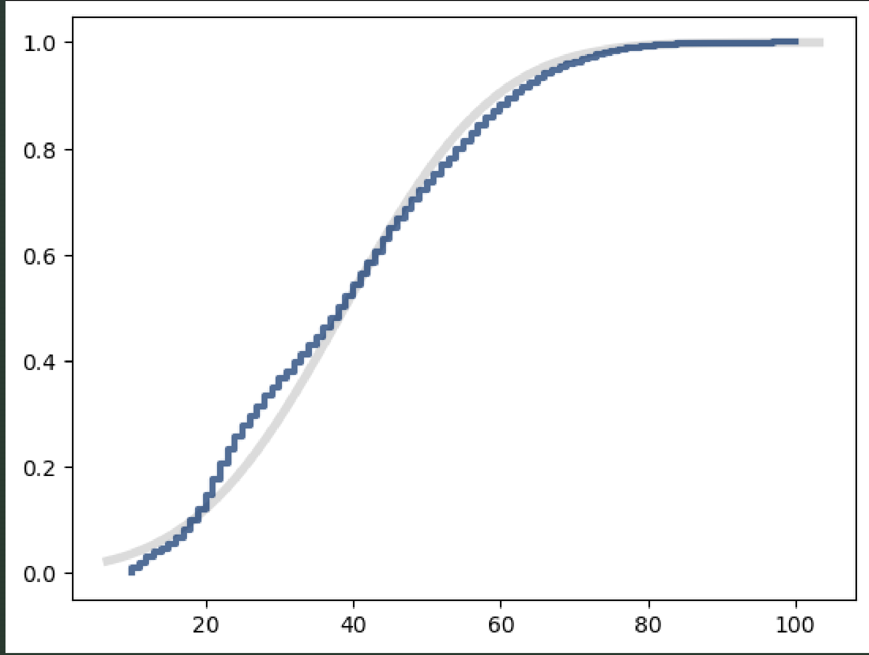
CDF



The CDF of track popularity shows us that over 90% of tracks have a popularity of less than around 70. This suggests that a popularity score of 70 might be a good cutoff to determine whether a song could be considered a ‘hit’.

Popularity vs normal distribution

```python
def MakeNormalModel(weights):
    """Plots a CDF with a Normal model.

    weights: sequence
    """
    cdf = thinkstats2.Cdf(weights, label="popularity")

    mean, var = thinkstats2.TrimmedMeanVar(weights)
    std = np.sqrt(var)
    print("n, mean, std", len(weights), mean, std)

    xmin = mean - 2 * std
    xmax = mean + 4 * std

    xs, ps = thinkstats2.RenderNormalCdf(mean, std, xmin, xmax)
    thinkplot.Plot(xs, ps, label="model", linewidth=4, color="0.8")
    thinkplot.Cdf(cdf)

MakeNormalModel(df.popularity)
```

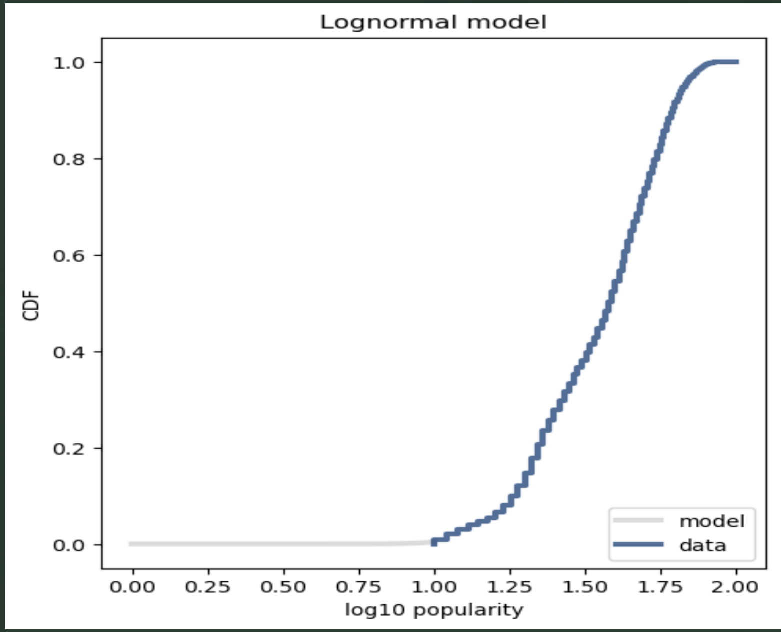


We can see in the chart that the normal distribution is not a very good fit for the distribution of track popularity, particularly in the left tail. The distribution matches better towards the right end of the curve.

Track popularity vs lognormal distribution

```
thinkplot.PrePlot(cols=2)

mu, sigma = log_pops.mean(), log_pops.std()
xs, ps = thinkstats2.RenderNormalCdf(mu, sigma, low=0, high=1)
thinkplot.Plot(xs, ps, label="model", color="0.8")

thinkplot.Cdf(cdf_log)
thinkplot.Config(xlabel="log10 popularity", ylabel="CDF", title="Lognormal model", loc="lower right")
```

✓ 0.0s

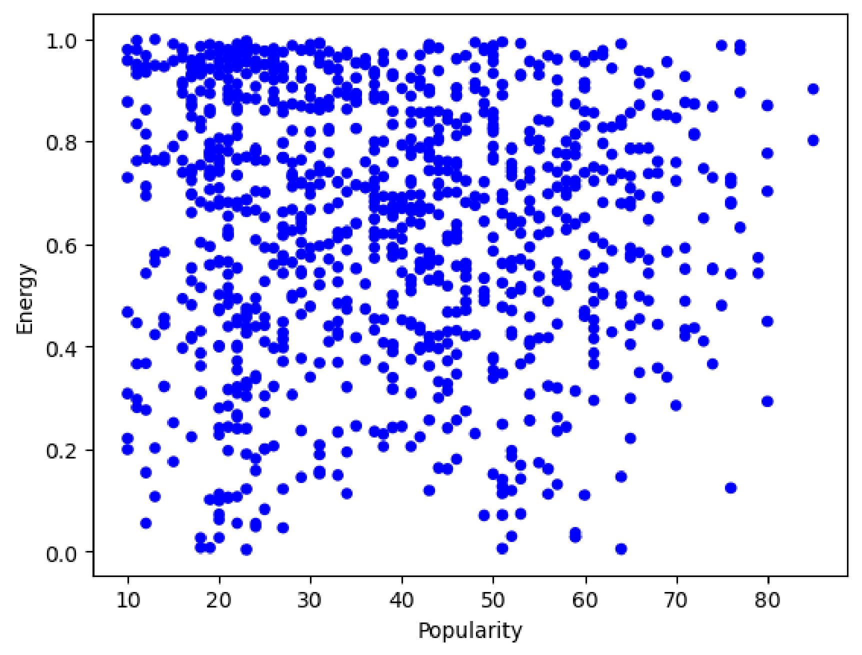


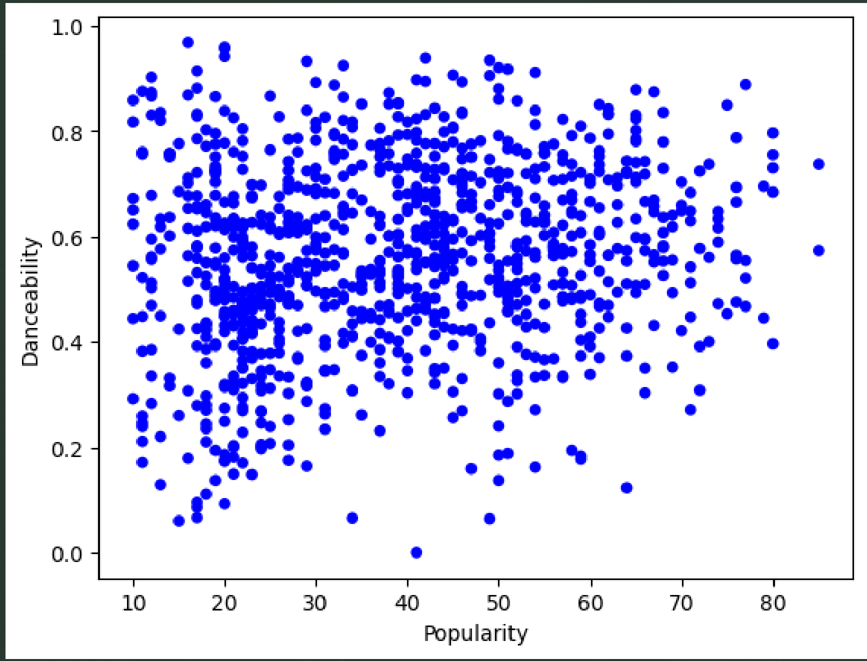
We can see in this chart that the lognormal distribution is a better fit for the distribution of track popularity. This makes sense as the lognormal model is usually a better fit for a pareto distribution, which we might expect track popularity to fit better than a normal distribution.

Scatterplots of Energy and Danceability vs Popularity

```python
def SampleRows(df, nrows, replace=False):
    indices = np.random.choice(df.index, nrows, replace=replace)
    sample = df.loc[indices]
    return sample

sample = SampleRows(df, 1000)
thinkplot.Scatter(sample.popularity, sample.energy, alpha=1)
thinkplot.Config(xlabel='Popularity',
                 ylabel='Energy',
                 legend=False)
```
✓ 0.0s

```python
thinkplot.Scatter(sample.popularity, sample.danceability, alpha=1)
thinkplot.Config(xlabel='Popularity',
                 ylabel='Danceability',
                 legend=False)
```
✓ 0.0s

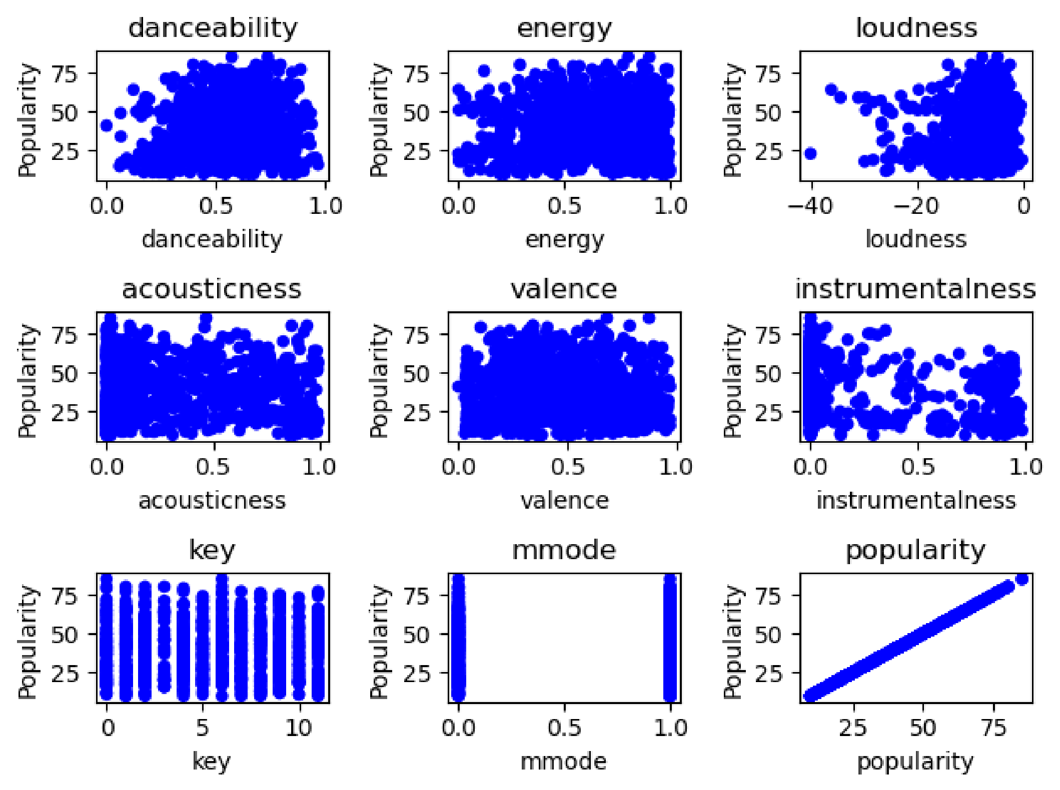




Scatterplots of other variables

```python
for i, feature in enumerate(features, 1):
    plt.subplot(3, 3, i)
    thinkplot.Scatter(sample[feature], sample.popularity, alpha=1 )
    plt.title(f'{feature}')
    plt.xlabel(feature)
    plt.ylabel('Popularity')
plt.tight_layout()
plt.show()
```

✓ 0.3s



Scatterplots analysis

Looking at the scatterplots does not reveal any obvious correlation between these variables and popularity. Perhaps the closest would be the scatterplot for danceability, which is confirmed by correlation analysis on the variables – The correlation between danceability and popularity is 0.12 which is low but still higher than many other variables.

Correlations

```
def Cov(xs, ys, meanx=None, meany=None):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    if meanx is None:
        meanx = np.mean(xs)
    if meany is None:
        meany = np.mean(ys)

    cov = np.dot(xs-meanx, ys-meany) / len(xs)
    return cov

def Corr(xs, ys):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    meanx, varx = thinkstats2.MeanVar(xs)
    meany, vary = thinkstats2.MeanVar(ys)

    corr = Cov(xs, ys, meanx, meany) / np.sqrt(varx * vary)
    return corr

for feature in features:
    print(f'Correlation between {feature} and popularity: {Corr(df[feature], df.popularity)}')
```

✓ 0.0s

```
Correlation between danceability and popularity: 0.1198779474897449
Correlation between energy and popularity: -0.05196339751893727
Correlation between loudness and popularity: 0.06050469500286372
Correlation between acousticness and popularity: -0.02042069320064255
Correlation between valence and popularity: 0.005644477360249906
Correlation between instrumentalness and popularity: -0.1797938830819091
Correlation between key and popularity: 0.008186558498638835
Correlation between mmode and popularity: -0.02402480863461855
Correlation between popularity and popularity: 1.0
```

Spearman Correlations

```python
def SpearmanCorr(xs, ys):
    xranks = pd.Series(xs).rank()
    yranks = pd.Series(ys).rank()
    return Corr(xranks, yranks)

for feature in features:
    print(f'Spearman Correlation between {feature} and popularity: {SpearmanCorr(df[feature], df.popularity)}')
```

✓ 0.0s

```
Spearman Correlation between danceability and popularity: 0.10942722123269469
Spearman Correlation between energy and popularity: -0.087530025648837
Spearman Correlation between loudness and popularity: 0.06663482983775926
Spearman Correlation between acousticness and popularity: 0.05303485313716811
Spearman Correlation between valence and popularity: 0.005423726587253576
Spearman Correlation between instrumentalness and popularity: -0.18849805477779994
Spearman Correlation between key and popularity: 0.008123680761793314
Spearman Correlation between mmode and popularity: -0.021904647934660744
Spearman Correlation between popularity and popularity: 1.0
```

Chi squared test – Does musical mode influence popularity?

```
class PopularityTest(thinkstats2.HypothesisTest):

    def MakeModel(self):
        firsts, others = self.data
        self.n = len(firsts)
        self.pool = np.hstack((firsts, others))

        pmf = thinkstats2.Pmf(self.pool)
        self.values = range(35, 44)
        self.expected_probs = np.array(pmf.Probs(self.values))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data

    def TestStatistic(self, data):
        firsts, others = data
        stat = self.ChiSquared(firsts) + self.ChiSquared(others)
        return stat

    def ChiSquared(self, lengths):
        hist = thinkstats2.Hist(lengths)
        observed = np.array(hist.Freqs(self.values))
        expected = self.expected_probs * len(lengths)
        stat = sum((observed - expected)**2 / expected)
        return stat

data = major_key.popularity.values, minor_key.popularity.values
ht = PopularityTest(data)
p_value = ht.PValue()
print('p-value =', p_value)
print('actual =', ht.actual)
print('ts max =', ht.MaxTestStat())
```

✓ 1.3s

```
p-value = 0.715
actual = 6.2109416340612205
ts max = 28.221264594720815
```

Chi squared test – Does musical mode influence popularity?

- The chi squared test gives a p value of 0.715 for the relationship between popularity and musical mode which suggests that there is not a significant correlation between mode and popularity.

Regression analysis

```
model = smf.ols('popularity ~ danceability + energy + instrumentalness + loudness', data=df)
results = model.fit()
results.summary()
```
✓ 0.0s

OLS Regression Results

Dep. Variable:	popularity	R-squared:	0.051
Model:	OLS	Adj. R-squared:	0.051
Method:	Least Squares	F-statistic:	1029.
Date:	Sat, 16 Nov 2024	Prob (F-statistic):	0.00
Time:	12:59:43	Log-Likelihood:	-3.1968e+05
No. Observations:	75875	AIC:	6.394e+05
Df Residuals:	75870	BIC:	6.394e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	45.5067	0.456	99.800	0.000	44.613	46.400
danceability	7.9655	0.356	22.386	0.000	7.268	8.663
energy	-10.7088	0.370	-28.933	0.000	-11.434	-9.983
instrumentalness	-7.8939	0.211	-37.396	0.000	-8.308	-7.480
loudness	0.3265	0.020	16.126	0.000	0.287	0.366

Omnibus:	4380.206	Durbin-Watson:	0.534
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2744.789
Skew:	0.338	Prob(JB):	0.00
Kurtosis:	2.359	Cond. No.	98.7

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

We were able to achieve the best fit for a model by doing a regression between popularity and danceability, energy, instrumentalness, and loudness. These variables has the highest correlations with popularity individually, and creating a model using all of them results in a R-squared value of 0.051 – which is still low. This suggests that these variables are not particularly effective in predicting track popularity.

Regression analysis

```
df['major_key'] = df.mmode == 1
df['minor_key'] = df.mmode == 0
model = smf.ols('popularity ~ minor_key', data=df)
results = model.fit()
results.summary()
```

✓ 0.0s

OLS Regression Results

Dep. Variable:	popularity	R-squared:	0.001
Model:	OLS	Adj. R-squared:	0.001
Method:	Least Squares	F-statistic:	43.82
Date:	Sat, 16 Nov 2024	Prob (F-statistic):	3.63e-11
Time:	12:59:43	Log-Likelihood:	-3.2166e+05
No. Observations:	75875	AIC:	6.433e+05
Df Residuals:	75873	BIC:	6.433e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	38.6955	0.076	507.111	0.000	38.546	38.845
minor_key[T.True]	0.8391	0.127	6.620	0.000	0.591	1.088

Omnibus:	4775.116	Durbin-Watson:	0.469
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2830.442
Skew:	0.335	Prob(JB):	0.00
Kurtosis:	2.333	Cond. No.	2.42

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

We also ran a regression with a categorical variable, musical mode. Our regression resulted in a R-squared value of 0.001, suggesting that the model created using musical mode is a poor fit for track popularity.