In [22]:
```python
# Create a scaled version of the data so that all values are between -1 and 1
mm = preprocessing.MinMaxScaler()
df_mm = df_raw.copy()
num_cols = df_mm.drop(columns='track_genre').columns
df_mm[num_cols] = mm.fit_transform(df_mm.drop(columns='track_genre'))
df_mm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19955 entries, 5000 to 108999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   popularity        19955 non-null  float64
 1   duration_ms       19955 non-null  float64
 2   explicit          19955 non-null  float64
 3   danceability      19955 non-null  float64
 4   energy            19955 non-null  float64
 5   key               19955 non-null  float64
 6   loudness          19955 non-null  float64
 7   mode              19955 non-null  float64
 8   speechiness       19955 non-null  float64
 9   acousticness      19955 non-null  float64
 10  instrumentalness  19955 non-null  float64
 11  liveness          19955 non-null  float64
 12  valence           19955 non-null  float64
 13  tempo             19955 non-null  float64
 14  time_signature    19955 non-null  float64
 15  track_genre       19955 non-null  object 
dtypes: float64(15), object(1)
memory usage: 2.6+ MB
```

II. Exploratory Data Analysis

In [24]:
```python
df.sample(10)
```

Out[24]:

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
54384	-1.412613	0.316920	-0.181294	0.976702	0.599810	1.061682	-0.819268	-1.349109	-0.442136	-0.690465	2.234677	-0.249663	-0.256109	1.196783	0.241705	idm
55210	1.650196	1.244444	-0.181294	-0.971826	0.111061	1.625871	0.179987	-1.349109	-0.361902	0.790679	-0.542043	0.218154	-0.180384	-1.195361	0.241705	indian
74026	1.139728	0.059853	-0.181294	-0.277663	-0.897968	-0.912980	-0.103597	0.741230	-0.540667	0.672792	-0.542049	-0.593647	-0.905745	-1.062472	-2.113381	mpb
12824	1.139728	0.147428	-0.181294	0.696601	0.970313	-0.912980	0.648232	-1.349109	0.374272	-0.636056	-0.542049	-0.753255	0.564904	-1.046021	0.241705	cantopop
69101	0.437834	0.440066	-0.181294	0.501748	0.682582	0.779587	0.606388	0.741230	-0.292930	-0.563510	-0.542049	-0.737661	0.796063	-0.371189	-2.113381	malay
50085	0.054983	-0.497352	-0.181294	-1.726880	1.135857	1.061682	0.993446	0.741230	-0.108535	-1.130767	-0.541642	0.264018	0.843889	-1.036955	0.241705	heavy-metal
42170	-1.157379	-0.741845	5.515904	-1.288461	1.399939	-1.195074	1.223362	0.741230	0.768399	-1.128732	-0.173831	0.135598	-0.036906	-0.397892	0.241705	grindcore
97389	0.884494	-0.646145	-0.181294	1.518636	0.934839	-0.630885	0.855406	-1.349109	0.472804	-0.046621	-0.542049	2.126113	0.871788	0.137524	0.241705	sertanejo
6034	-0.519294	-0.877322	-0.181294	-2.031337	1.151623	0.215398	0.733740	0.741230	0.080084	-1.131743	2.285986	0.470408	1.138816	1.330332	0.241705	black-metal
26933	-0.902145	-0.314015	-0.181294	-1.178857	-1.248764	0.497493	-1.004383	-1.349109	-0.613863	1.615888	1.766859	-0.368911	-1.160817	-1.410998	0.241705	disney

In [25]:
```
# Examine the descriptive statistics of the numerical data
df.describe()
```

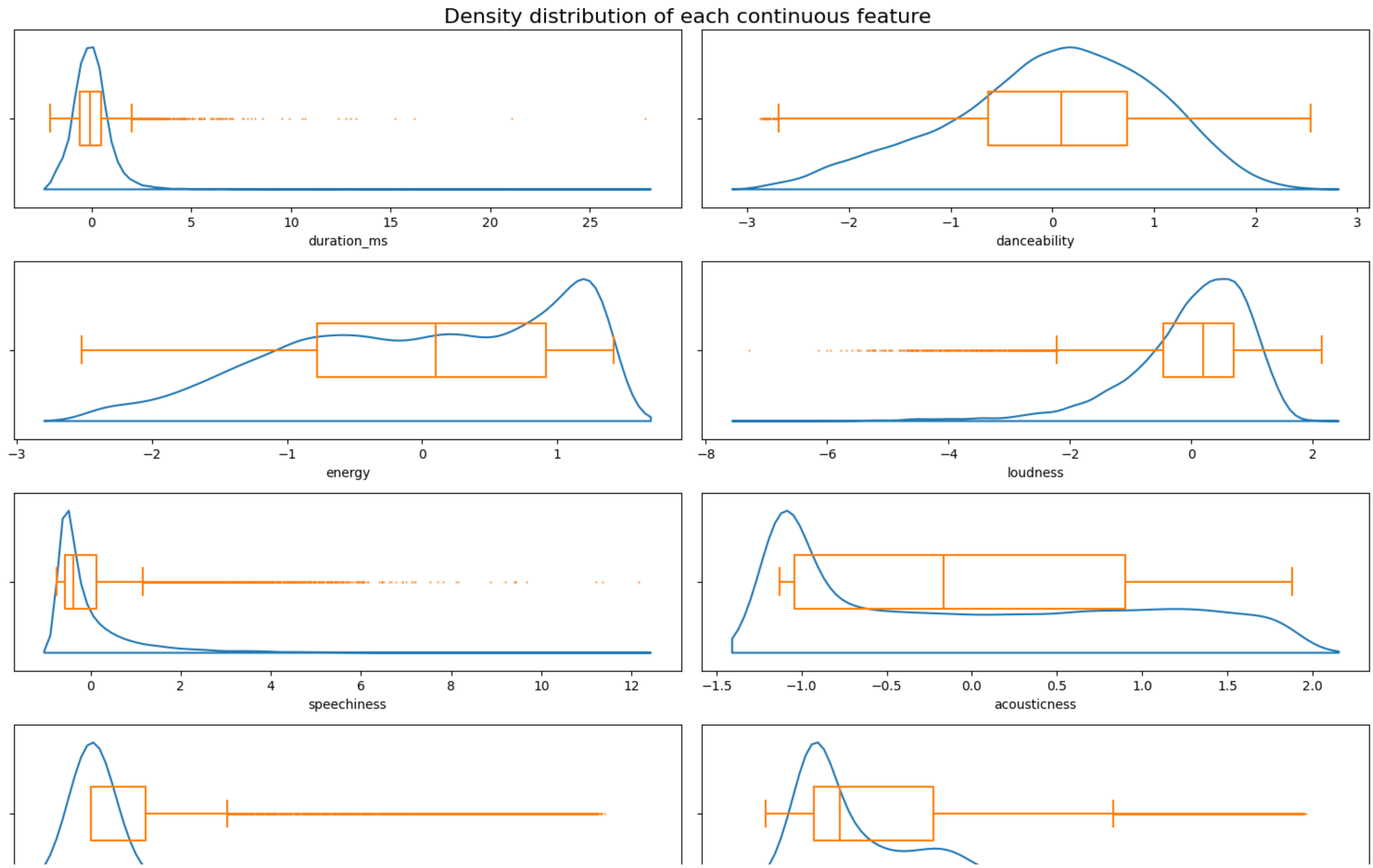
Out[25]:

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	tin
count	1.995500e+04	19955.000000	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	19955.000000	1.995500e+04	1.995500e+04	1
mean	2.050978e-16	0.000000	-3.988012e-17	-3.304353e-16	2.050978e-16	8.759384e-17	-4.557728e-17	8.830599e-17	-5.412302e-17	-9.115457e-17	5.127444e-17	0.000000	1.709148e-16	4.158927e-16	-
std	1.000025e+00	1.000025	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025	1.000025e+00	1.000025e+00	1
min	-2.178315e+00	-2.097496	-1.812939e-01	-2.872858e+00	-2.521597e+00	-1.477168e+00	-7.289416e+00	-1.349109e+00	-7.503996e-01	-1.131785e+00	-5.420494e-01	-1.063757	-1.786541e+00	-3.078958e+00	-9
25%	-9.021446e-01	-0.583862	-1.812939e-01	-6.338779e-01	-7.797228e-01	-9.129795e-01	-4.655263e-01	-1.349109e+00	-5.660042e-01	-1.043219e+00	-5.420494e-01	-0.657857	-8.300205e-01	-7.976322e-01	[illegible]
50%	1.826001e-01	-0.062605	-1.812939e-01	8.768604e-02	9.923667e-02	-6.669615e-02	1.902211e-01	7.412297e-01	-3.830164e-01	-1.645077e-01	-5.419570e-01	-0.442294	-8.871751e-02	7.469095e-03	[illegible]
75%	7.568767e-01	0.461311	-1.812939e-01	7.392248e-01	9.190733e-01	7.795872e-01	7.005375e-01	7.412297e-01	1.293495e-01	8.994976e-01	-2.221223e-01	0.337401	7.801215e-01	7.138510e-01	[illegible]
max	3.436834e+00	27.786392	5.515904e+00	2.541613e+00	1.419647e+00	1.625871e+00	2.155076e+00	7.412297e-01	1.215587e+01	1.878866e+00	2.473113e+00	3.433249	2.139177e+00	3.191446e+00	2

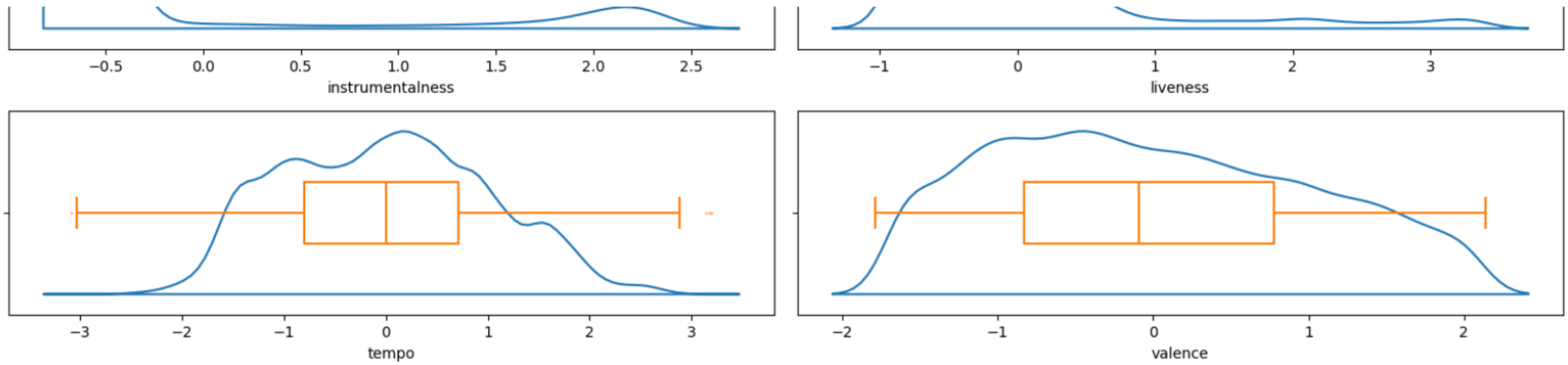
Most of the features ranges between [0,1] while the rest vary drastically: some ranges betweem [0,100] (e.g. `popularity`), some [0,11] (e.g. `key`), or even to negative values (e.g. `loudness`). Thus, it will be important to standardize and/or mean center these features accordingly for the remainder of the project.

In [ ]:
```
# Plot the density distribution for each continuous variable
cts_cols = ['duration_ms','danceability','energy','loudness','speechiness','acousticness','instrumentalness','liveness','tempo','valence']

fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 13))
col = 0
row = 0

for i in cts_cols:
  sns.violinplot(data=df, x=i, ax=axes[row][col], inner=None, fill=False, split=True) #inner_kws=dict(box_width=15, whis_width=2)
```

```
    sns.boxplot(data=df, x=i, fill=False, width=0.3, fliersize=0.5, boxprops={'zorder': 2}, ax=axes[row][col])
    axes[row][col].set_xlabel(i, fontsize=10)
    col+=1
    if col==2:
      col=0
      row+=1

plt.suptitle('Density distribution of each continuous feature', fontsize=16)
plt.tight_layout()
```



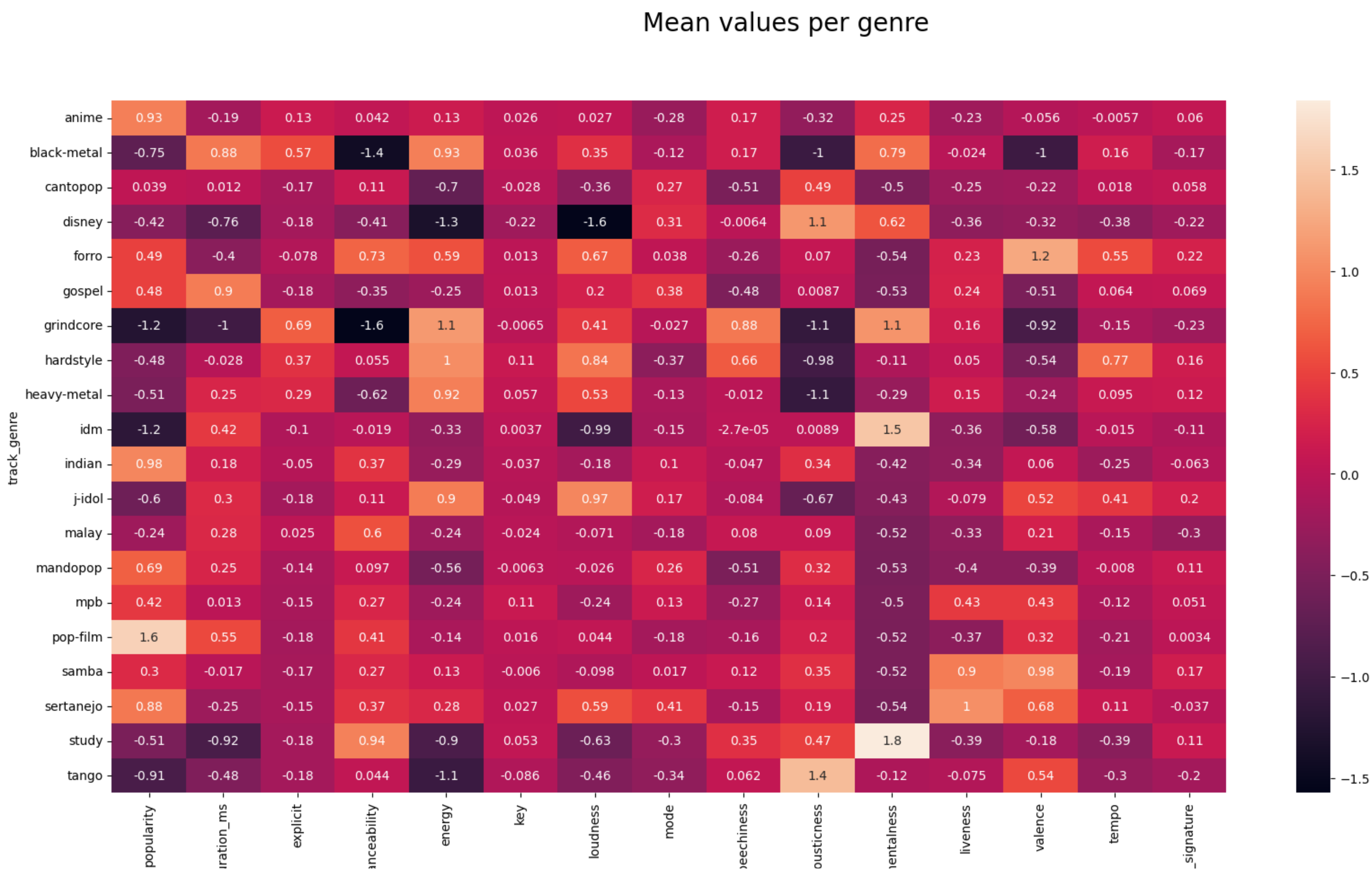
Density distribution of each continuous feature
duration_ms
danceability
energy
loudness
speechiness
acousticness



Like some of the categorical predictor variables, many of the continuous predictor variables are heavily skewed with a handful of outliers. This should be addressed depending on the assumptions made by different modeling techiques (e.g. distance-based clustering methods may be sensitive to outliers, linear regression assumes normality and homoscedasticity).

In [ ]:
```
# Examine the predictor variable means per genre
fig, ax = plt.subplots(figsize=(20, 10))
sns.heatmap(df.groupby('track_genre').mean(),annot=True)
plt.suptitle('Mean values per genre', fontsize=20)
```

Out[ ]:
```
Text(0.5, 0.98, 'Mean values per genre')
```



Mean values per genre
track_genre
anime
black-metal
cantopop
disney
forro
gospel
grindcore
hardstyle
heavy-metal
idm
indian
j-idol
malay
mandopop
mpb
pop-film
samba
sertanejo
study
tango
popularity
ıration_ms
explicit
ınceability
energy
key
loudness
mode
ɔeechiness
ousticness
ıentalness
liveness
valence
tempo
_signature

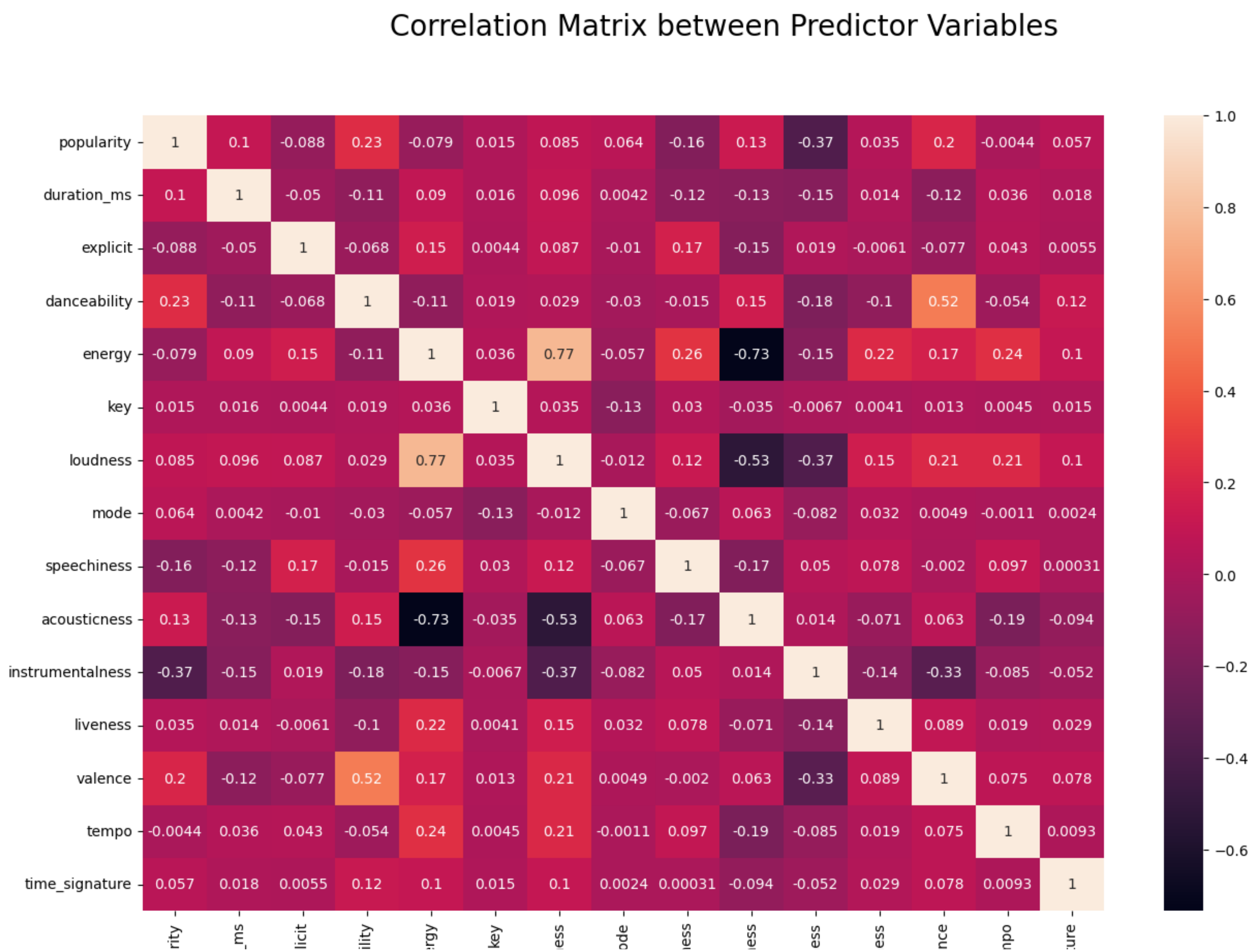
As illustrated by the mean values per feature per genre, some of the class imbalance in the predictor variables are associated with specific classes in the response variable `track_genre`. For instance, `explicit`-ness is expecially present in genres like grindcore, black-metal, an hardstyle while genres like sertanejo, gospel, and disney lean toward being on a major scale.

In [ ]:

```
# Plot the correlation matrix between predictor variables
fig, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(df.drop(columns=['track_genre']).corr(),annot=True)
plt.suptitle("Correlation Matrix between Predictor Variables", fontsize=20)
```

Out[ ]:

```
Text(0.5, 0.98, 'Correlation Matrix between Predictor Variables')
```



Correlation Matrix between Predictor Variables
popularity
duration_ms
explicit
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
time_signature

It is important to note the following highly-correlated features since modeling techinques (e.g. logistic regression) can make assumptions about the absence of multicollinearity.

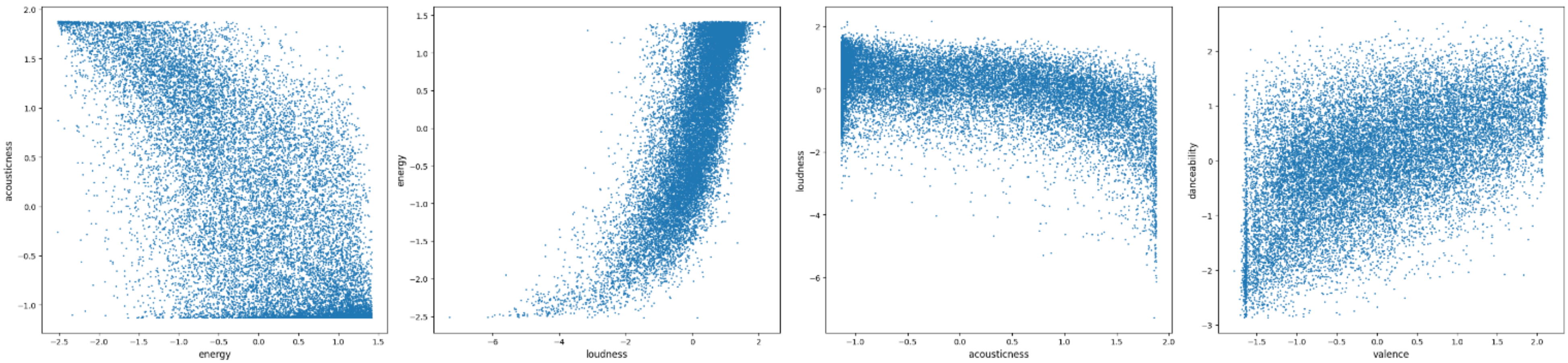
- valence and danceability: 0.52
- energy and loudness: -0.73
- energy and acousticness: -0.73
- loudness and acousticness: -0.53

The scatterplots below illutrates how their relationships are non-linear.

In [ ]:

```
fig, axs = plt.subplots(1,4, figsize=(28,7))
axs[0].scatter(df['energy'], df['acousticness'], s=1)
axs[0].set_xlabel('energy', fontsize=12)
axs[0].set_ylabel('acousticness', fontsize=12)
axs[1].scatter(df['loudness'], df['energy'], s=1)
axs[1].set_xlabel('loudness', fontsize=12)
axs[1].set_ylabel('energy', fontsize=12)
axs[2].scatter(df['acousticness'], df['loudness'], s=1)
axs[2].set_xlabel('acousticness', fontsize=12)
axs[2].set_ylabel('loudness', fontsize=12)
axs[3].scatter(df['valence'], df['danceability'], s=1)
axs[3].set_xlabel('valence', fontsize=12)
axs[3].set_ylabel('danceability', fontsize=12)

plt.suptitle("Relationship between strongly correlated variables", y=1.01, fontsize=20)
plt.tight_layout()
plt.show()
```

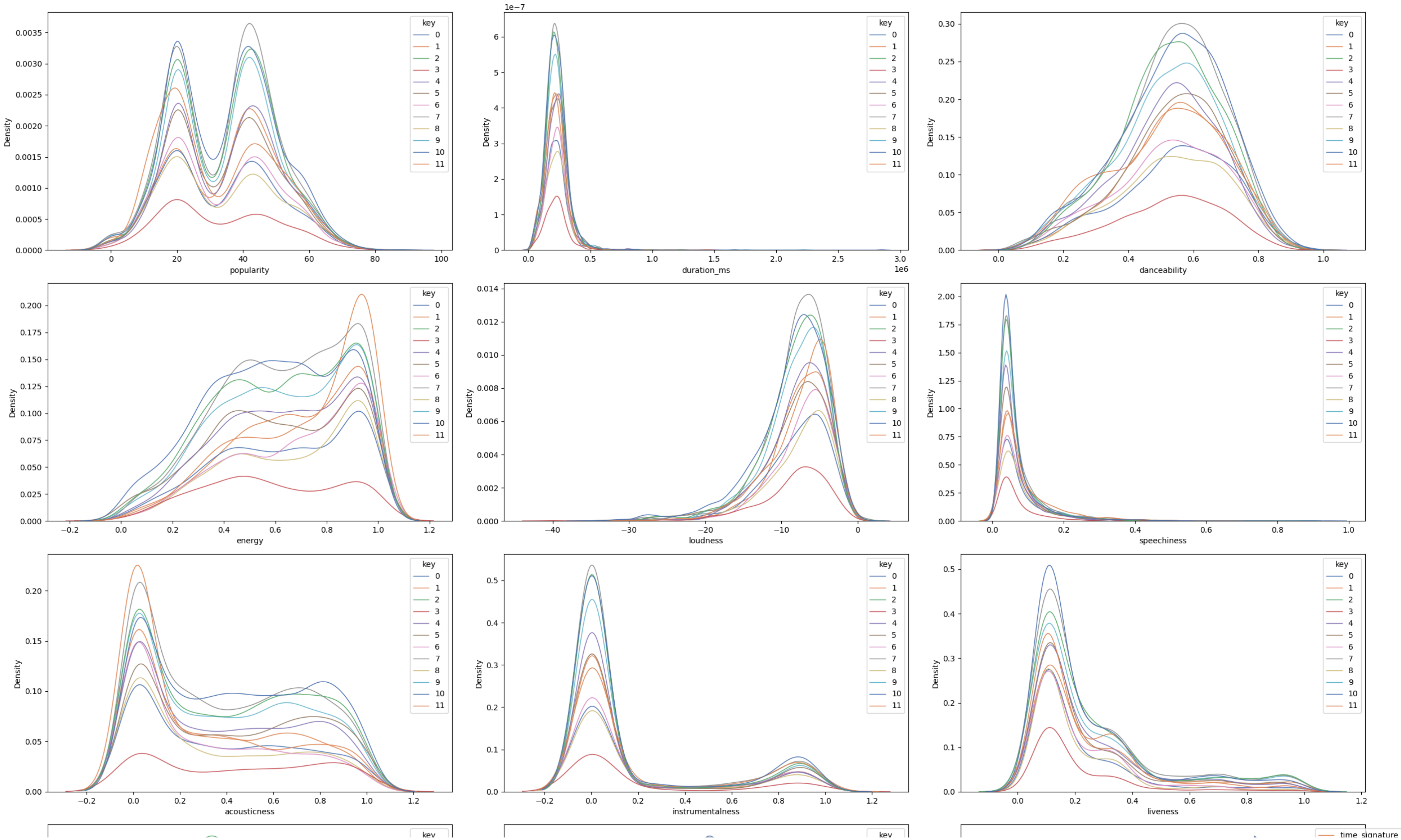
Relationship between strongly correlated variables

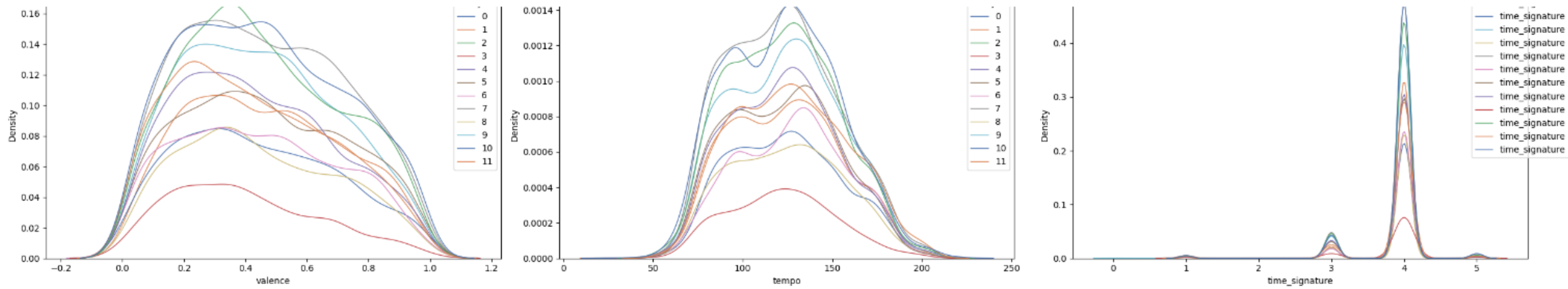


We can further explore the relationship between each numerical feature's distribution and each categorical feature via the density plots below.

In [ ]:
```python
# Create a figure and axes
fig, ax = plt.subplots(4,3, figsize=(25,20))
i=0
j=0
# Loop through the columns and create a KDE plot for each
for col in df_raw.drop(columns=['track_genre','mode','explicit','key']).columns:
    sns.kdeplot(df_raw[[col,'key']], x=col, ax=ax[i][j], label=col, lw=1, palette='deep', hue='key')
    j+=1
    if j==3:
      j=0
      i+=1

# Add a legend and title
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1))
plt.suptitle("Density Plot for Each Numerical Feature Categorized by 'Key'", y=1.01, fontsize=20)
plt.tight_layout()
plt.show()
```

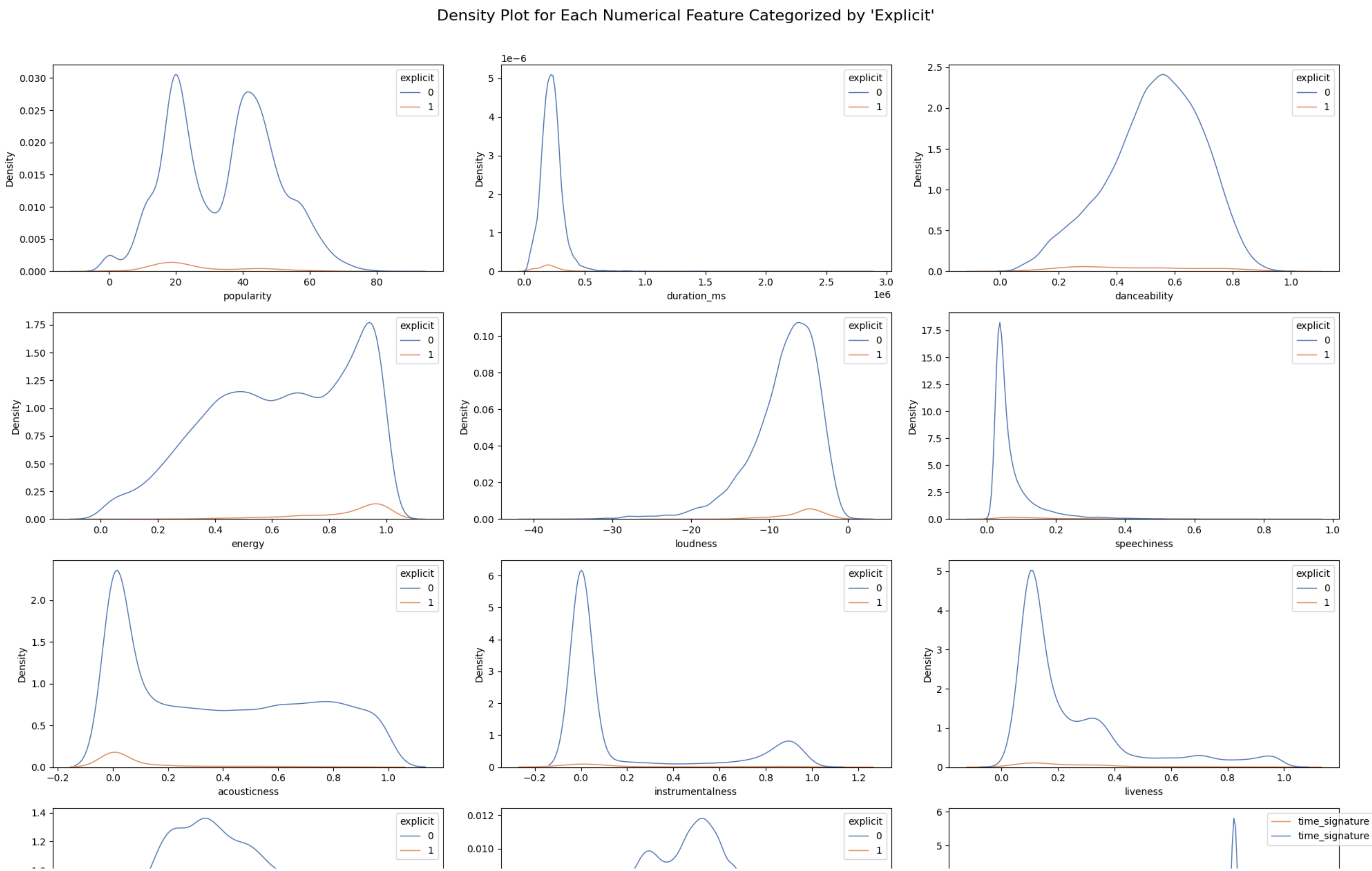
Density Plot for Each Numerical Feature Categorized by 'Key'

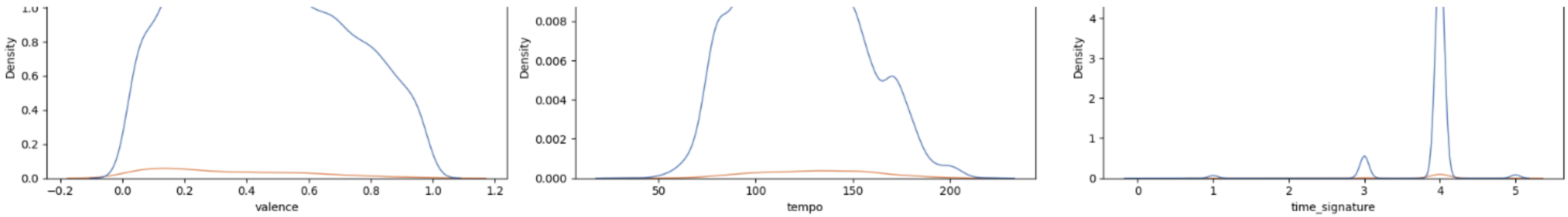




In [ ]:

```
# Create a figure and axes
fig, ax = plt.subplots(4,3, figsize=(20,15))
i=0
j=0
# Loop through the columns and create a KDE plot for each
for col in df_raw.drop(columns=['track_genre','mode','explicit','key']).columns:
    sns.kdeplot(df_raw[[col,'explicit']], x=col, ax=ax[i][j], label=col, lw=1, palette='deep', hue='explicit')
    j+=1
    if j==3:
      j=0
      i+=1

# Add a legend and title
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1))
plt.suptitle("Density Plot for Each Numerical Feature Categorized by 'Explicit'", y=1.01, fontsize=16)
plt.tight_layout()
plt.show()
```

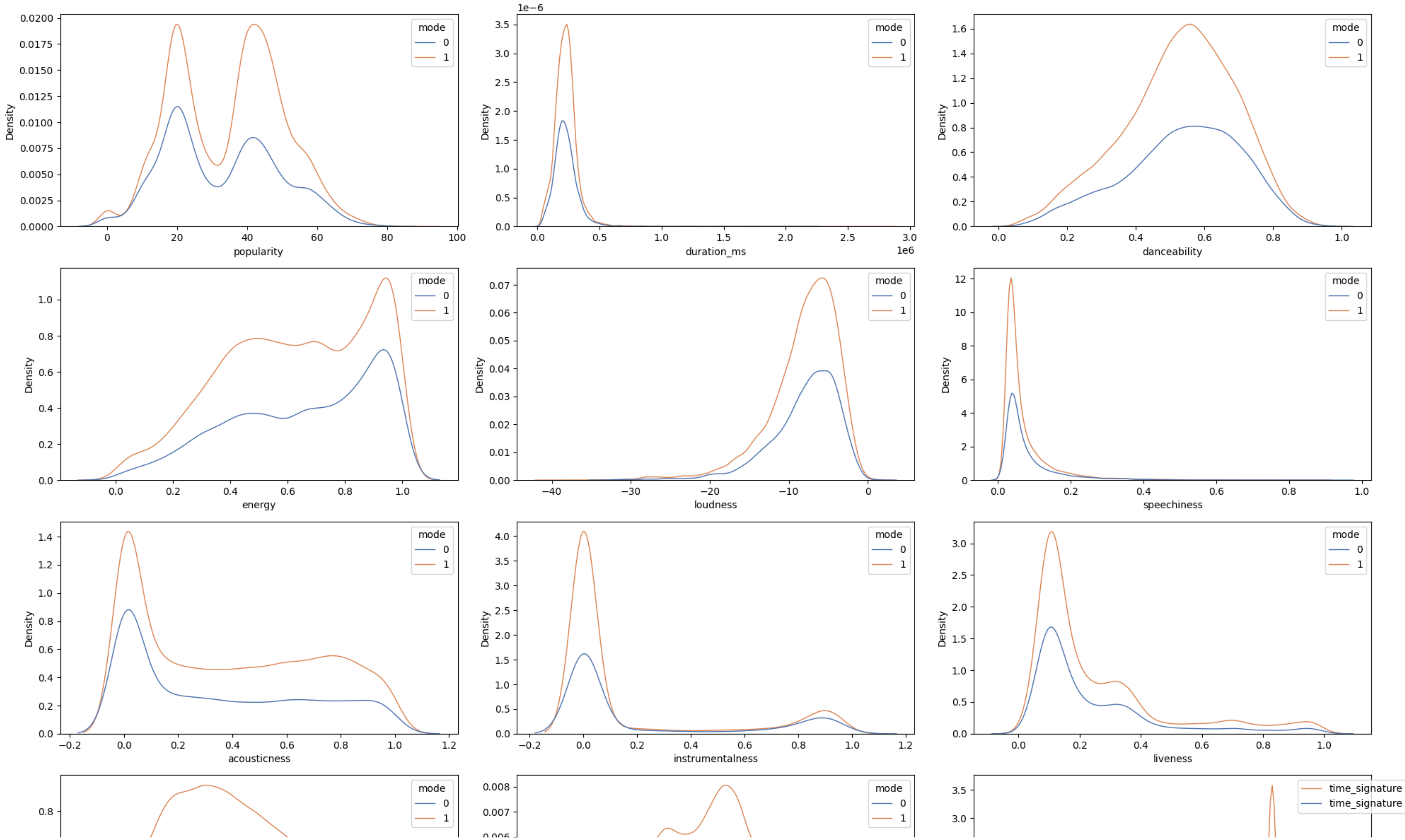

Density Plot for Each Numerical Feature Categorized by 'Explicit'
explicit
0
1
Density
popularity
duration_ms
1e−6
1e6
danceability
energy
loudness
speechiness
acousticness
instrumentalness
liveness
time_signature
time_signature

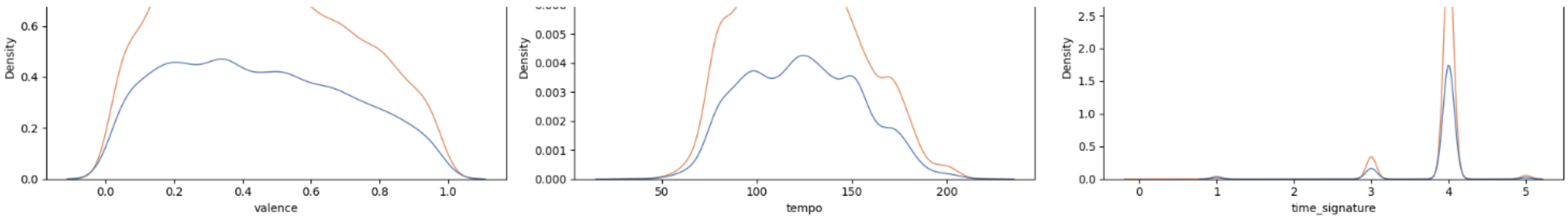


In [ ]:

```
# Create a figure and axes
fig, ax = plt.subplots(4,3, figsize=(20,15))
i=0
j=0
# Loop through the columns and create a KDE plot for each
for col in df_raw.drop(columns=['track_genre','mode','explicit','key']).columns:
    sns.kdeplot(df_raw[[col,'mode']], x=col, ax=ax[i][j], label=col, lw=1, palette='deep', hue='mode')
    j+=1
    if j==3:
      j=0
      i+=1

# Add a legend and title
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1))
plt.suptitle("Density Plot for Each Numerical Feature Categorized by 'Modality'", y=1.01, fontsize=16)
plt.tight_layout()
plt.show()
```

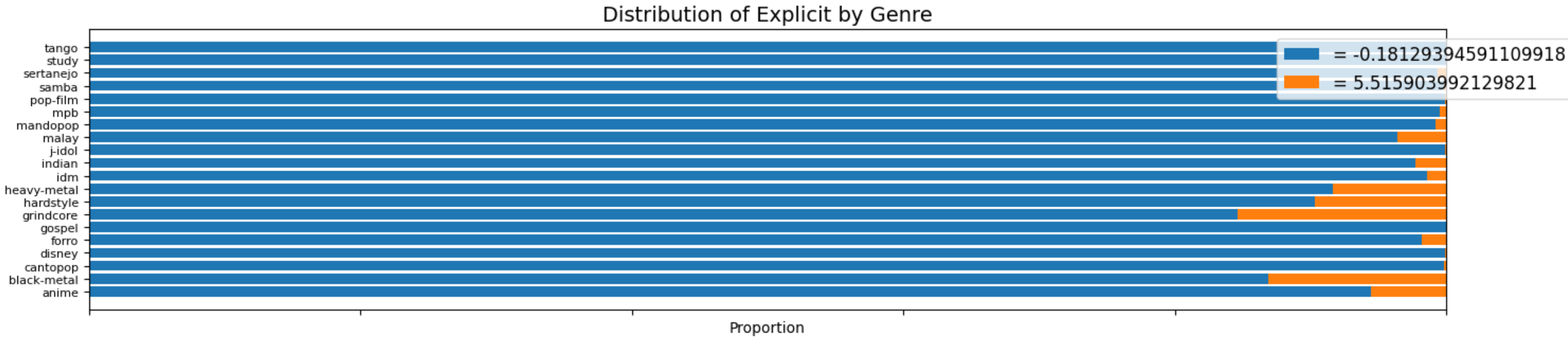
Density Plot for Each Numerical Feature Categorized by 'Modality'



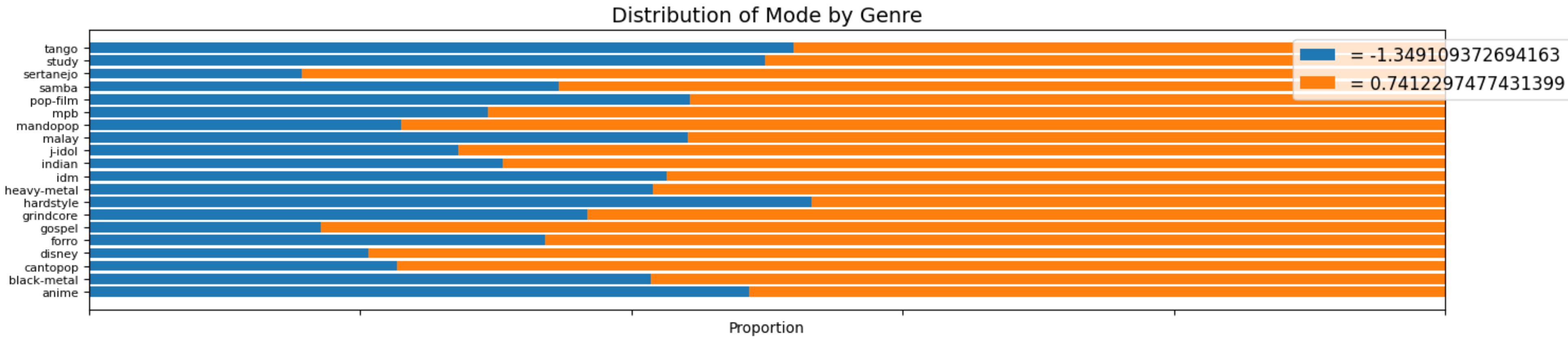


We can also visualize the class distribution per genre for every categorical feature.

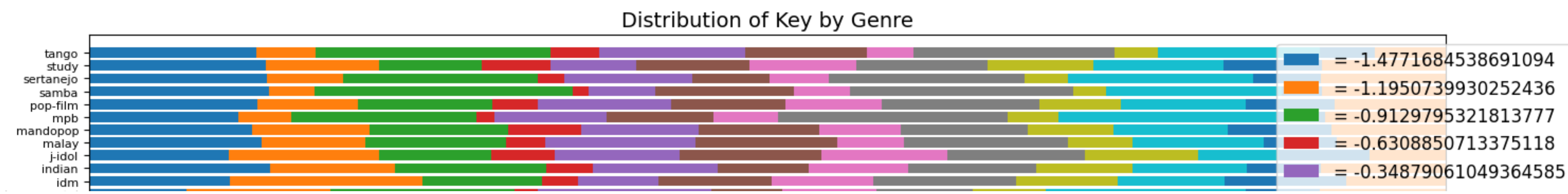
```
In [ ]: # calculate the class proportions for each categorical feature
proportions = {}
for col in ['explicit', 'mode', 'key', 'time_signature']:
    proportions[col] = (df.groupby('track_genre')[col].value_counts(normalize=True).unstack(fill_value=0))


fig, axes = plt.subplots(4, 1, figsize=(15, 15), sharex=True)
genres = df['track_genre'].unique()  # 20 unique genres

# plot each category in a separate subplot
for ax, (col, prop_df) in zip(axes, proportions.items()):
    left_pos = np.zeros(20) # bars intialized at leftmost position

    # plot each class as a stacked bar
    for col_class in prop_df.columns:
        ax.barh(genres,
            prop_df[col_class].reindex(genres, fill_value=0),  # index replaced with genres
            left=left_pos,  # stack this class's bars on top of previous bars
            label=f"= {col_class}") # legend label
        left_pos += prop_df[col_class].reindex(genres, fill_value=0)

    ax.set_title(f"Distribution of {col.capitalize()} by Genre", fontsize=14)
    ax.set_xlabel("Proportion", fontsize=10)
    ax.set_xlim(0, 1)
    ax.tick_params(axis='both', which='major', labelsize=8)
    ax.legend(loc='upper right',bbox_to_anchor=(1.1, 1),fontsize=12)

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



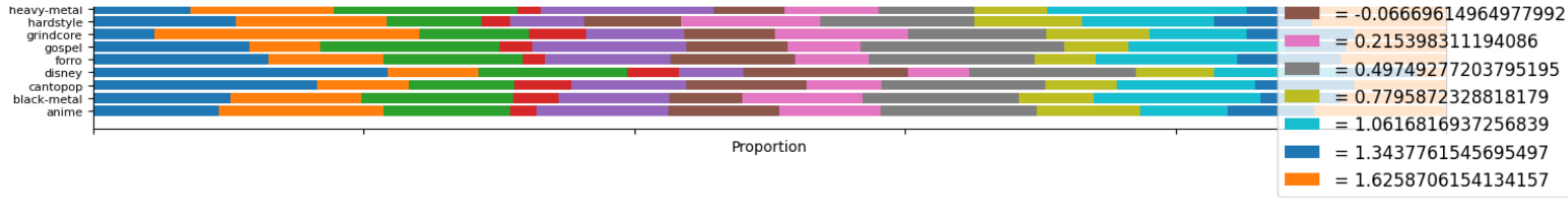
Distribution of Explicit by Genre
tango
study
sertanejo
samba
pop-film
mpb
mandopop
malay
j-idol
indian
idm
heavy-metal
hardstyle
grindcore
gospel
forro
disney
cantopop
black-metal
anime
= -0.18129394591109918
= 5.515903992129821
Proportion

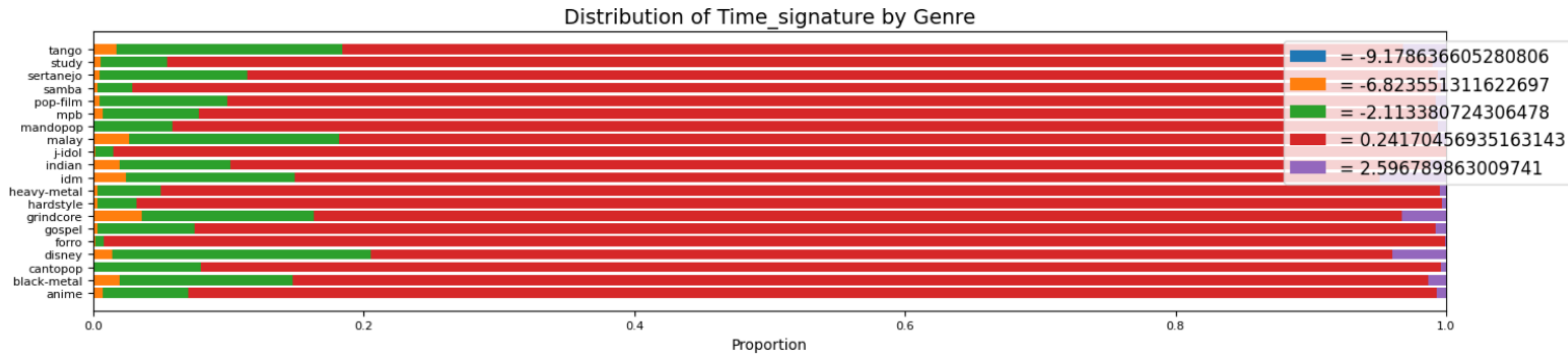


Distribution of Mode by Genre
tango
study
sertanejo
samba
pop-film
mpb
mandopop
malay
j-idol
indian
idm
heavy-metal
hardstyle
grindcore
gospel
forro
disney
cantopop
black-metal
anime
= -1.349109372694163
= 0.7412297477431399
Proportion



Distribution of Key by Genre
tango
study
sertanejo
samba
pop-film
mpb
mandopop
malay
j-idol
indian
idm
= -1.4771684538691094
= -1.1950739930252436
= -0.9129795321813777
= -0.6308850713375118
= -0.34879061049364585





iii. Regression Analysis

Since our main problem to address involves the classification of track genres, there is not much use in performing linear regression. However, in exploring the data and conducting some preliminary analyses before we settled on the primary topic of our project, we did use linear regression to predict the loudness of a song from its energy, seeing as they were fairly correlated from EDA. We also compared the fit to a ridge regression model, though the results indicate that ridge regression had minimal effect on the regression. The original model was simple enough to not have overfitted to the training data, as indicated by the similar mean-squared errors of the training and testing sets. Therefore, applying ridge regression did not impact the model, as no regularization was really needed.

In [26]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
```

In [27]:

```
X = df[['energy']]
y = df[['loudness']]

# create the training and testing (validation) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# fit a linear regression model regressing loudness on energy
```