Il dataset analizzato è stato ottenuto da un'elaborazione del 'Ryerson Audio-Visual Database of Emotional Speech and Song' (RAVDESS), che contiene le registrazioni di 24 attori professionisti che pronunciano due differenti frasi in un accento neutrale nordamericano.

Il dataset contiene 2452 record che sono caratterizzati da 38 attributi.

L'identificativo dell'attore ('actor') ha range tra 01 e 24 con numeri pari per le donne e dispari per gli uomini; il genere degli attori è rappresentato dall'attributo 'sex' (M, F). La feature 'modality' ha un unico valore ('audio_only'), 'vocal_channel' invece indica se la frase è stata pronunciata come 'speech' o come 'song'. 'Emotion' indica l'emozione che l'attore doveva rappresentare durante la registrazione dell'audio e ha 8 possibili valori (neutral, calm, happy, sad, angry, fearful, disgust, surprised) di cui due (surprise e disgust) non sono presenti con la frase cantata. Ogni espressione è stata prodotta a due livelli di 'emotional intensity': normal e strong (quest'ultima non è presente nel caso di emozione neutral). La feature 'statement' ha due possibili valori: "Kids are talking by the door" e "Dogs are sitting by the door" mentre 'repetition' indica se la registrazione è relativa alla prima o alla seconda ripetizione della frase.

Altri attributi sono relativi a caratteristiche della registrazione audio, come 'channels' (numero di canali, 1 for mono e 2 for stereo audio), 'sample_width' (numero di bytes per campione, 1 significa 8-bit, 2 significa16-bit), 'frame_rate' (la frequenza di campionatura usata espressa in Hertz) e 'frame_width' (numero di bytes per ogni frame). Sono poi presenti altre feature numeriche che contengono informazioni sulla lunghezza del file audio espresso in millisecondi ('length_ms'), sul numero di frame estratti dal campione ('frame_count'), sul volume in dBFS ('intensity') e sulla somma dei zero-crossing rate ('zero_crossings_sum').

Infine, i record sono caratterizzati da variabili statistiche del segnale audio originale ( 'mean', 'std', 'min', 'max', 'kur', 'skew' ) e da statistiche dei Mel-Frequency Cepstral Coefficients, dello spectral centroid e del cromagramma stft.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('ravdess_features.csv', sep =',', skipinitialspace = True)
```

In [3]:
```
df.head()
```

Out[3]:

| | modality | vocal_channel | emotion | emotional_intensity | statement | repetition | actor | sex | channels | sample_width | ... | stft_min |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | audio-only | speech | fearful | normal | Dogs are sitting by the door | 2nd | 2.0 | F | 1 | 2 | ... | 0.0 |
| **1** | audio-only | speech | angry | normal | Dogs are sitting by the door | 1st | 16.0 | F | 1 | 2 | ... | 0.0 |
| **2** | audio-only | NaN | happy | strong | Dogs are sitting by the door | 2nd | 16.0 | F | 1 | 2 | ... | 0.0 |
| **3** | audio-only | NaN | surprised | normal | Kids are talking by the door | 1st | 14.0 | F | 1 | 2 | ... | 0.0 |
| **4** | audio-only | song | happy | strong | Dogs are sitting by the door | 2nd | 2.0 | F | 1 | 2 | ... | 0.0 |

5 rows × 38 columns

In [4]:
```
df.describe()
```

Out[4]:

| | actor | channels | sample_width | frame_rate | frame_width | length_ms | frame_count | intensity | zero_crossings |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 1326.000000 | 2452.000000 | 2452.0 | 2452.0 | 2452.000000 | 2452.000000 | 2452.000000 | 1636.000000 | 2452.0 |
| **mean** | 12.582202 | 1.002447 | 2.0 | 48000.0 | 2.004894 | 4092.151305 | 193587.188010 | -37.625332 | 12885.3 |
| **std** | 6.916240 | 0.049416 | 0.0 | 0.0 | 0.098833 | 598.321526 | 36825.369056 | 8.451982 | 3665.3 |
| **min** | 1.000000 | 1.000000 | 2.0 | 48000.0 | 2.000000 | 2936.000000 | -1.000000 | -63.864613 | 4721.0 |
| **25%** | 7.000000 | 1.000000 | 2.0 | 48000.0 | 2.000000 | 3604.000000 | 172972.000000 | -43.539869 | 10362.5 |
| **50%** | 13.000000 | 1.000000 | 2.0 | 48000.0 | 2.000000 | 4004.000000 | 190591.000000 | -37.072745 | 12383.5 |
| **75%** | 19.000000 | 1.000000 | 2.0 | 48000.0 | 2.000000 | 4538.000000 | 217817.000000 | -31.591309 | 14966.0 |
| **max** | 24.000000 | 2.000000 | 2.0 | 48000.0 | 4.000000 | 6373.000000 | 305906.000000 | -16.353953 | 30153.0 |

8 rows × 31 columns

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

Out[5]:
```
modality                0
vocal_channel         196
emotion                 0
emotional_intensity     0
statement               0
repetition              0
actor                1126
sex                     0
channels                0
sample_width            0
frame_rate              0
frame_width             0
length_ms               0
frame_count             0
intensity             816
zero_crossings_sum      0
mfcc_mean               0
mfcc_std                0
mfcc_min                0
mfcc_max                0
sc_mean                 0
sc_std                  0
sc_min                  0
sc_max                  0
sc_kur                  0
sc_skew                 0
stft_mean               0
stft_std                0
stft_min                0
stft_max                0
stft_kur                0
stft_skew               0
mean                    0
std                     0
min                     0
max                     0
kur                     0
skew                    0
dtype: int64
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 38 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   modality             2452 non-null   object 
 1   vocal_channel        2256 non-null   object 
 2   emotion              2452 non-null   object 
 3   emotional_intensity  2452 non-null   object 
 4   statement            2452 non-null   object 
 5   repetition           2452 non-null   object 
 6   actor                1326 non-null   float64
 7   sex                  2452 non-null   object 
 8   channels             2452 non-null   int64  
 9   sample_width         2452 non-null   int64  
 10  frame_rate           2452 non-null   int64  
 11  frame_width          2452 non-null   int64  
 12  length_ms            2452 non-null   int64  
 13  frame_count          2452 non-null   float64
 14  intensity            1636 non-null   float64
 15  zero_crossings_sum   2452 non-null   int64  
 16  mfcc_mean            2452 non-null   float64
 17  mfcc_std             2452 non-null   float64
 18  mfcc_min             2452 non-null   float64
 19  mfcc_max             2452 non-null   float64
 20  sc_mean              2452 non-null   float64
 21  sc_std               2452 non-null   float64
 22  sc_min               2452 non-null   float64
 23  sc_max               2452 non-null   float64
 24  sc_kur               2452 non-null   float64
 25  sc_skew              2452 non-null   float64
 26  stft_mean            2452 non-null   float64
 27  stft_std             2452 non-null   float64
 28  stft_min             2452 non-null   float64
 29  stft_max             2452 non-null   float64
 30  stft_kur             2452 non-null   float64
 31  stft_skew            2452 non-null   float64
 32  mean                 2452 non-null   float64
 33  std                  2452 non-null   float64
 34  min                  2452 non-null   float64
 35  max                  2452 non-null   float64
 36  kur                  2452 non-null   float64
 37  skew                 2452 non-null   float64
dtypes: float64(25), int64(6), object(7)
memory usage: 728.1+ KB
```

In [7]:
```
for col in df.columns:
    print(df[col].value_counts())
```

```
modality
audio-only    2452
Name: count, dtype: int64
vocal_channel
speech    1335
song       921
Name: count, dtype: int64
emotion
fearful      376
angry        376
happy        376
calm         376
sad          376
surprised    192
disgust      192
neutral      188
Name: count, dtype: int64
emotional_intensity
normal    1320
strong    1132
Name: count, dtype: int64
statement
Dogs are sitting by the door    1226
Kids are talking by the door    1226
Name: count, dtype: int64
repetition
2nd    1226
1st    1226
Name: count, dtype: int64
actor
22.0    65
12.0    63
14.0    62
20.0    61
8.0     61
```

```
19.0    60
13.0    60
2.0     58
16.0    58
24.0    58
5.0     58
10.0    56
21.0    55
11.0    55
6.0     55
17.0    55
4.0     52
3.0     51
1.0     51
7.0     51
23.0    51
9.0     51
15.0    44
18.0    35
Name: count, dtype: int64
sex
M    1248
F    1204
Name: count, dtype: int64
channels
1    2446
2       6
Name: count, dtype: int64
sample_width
2    2452
Name: count, dtype: int64
frame_rate
48000    2452
Name: count, dtype: int64
frame_width
2    2446
4       6
Name: count, dtype: int64
length_ms
3737    82
3604    69
3570    69
3504    67
3537    67
        ..
5472     1
5973     1
3003     1
5906     1
5806     1
Name: count, Length: 95, dtype: int64
frame_count
168168.0    65
179379.0    62
176176.0    58
171371.0    57
184184.0    54
            ..
296296.0     1
145745.0     1
142542.0     1
145746.0     1
278679.0     1
Name: count, Length: 158, dtype: int64
intensity
-47.386438    7
-46.787174    7
-45.012642    7
-46.172481    7
-36.522822    7
              ..
-29.910065    1
-29.132889    1
-34.559789    1
-45.649076    1
-29.512788    1
Name: count, Length: 989, dtype: int64
zero_crossings_sum
10667    4
12913    4
9531     3
14005    3
12769    3
```

```
       ..
11266    1
10147    1
17605    1
13830    1
9716     1
Name: count, Length: 2176, dtype: int64
mfcc_mean
-27.674932    2
-33.485947    1
-32.751840    1
-30.872604    1
-30.884867    1
             ..
-29.027380    1
-30.782503    1
-30.988361    1
-32.051650    1
-29.019236    1
Name: count, Length: 2451, dtype: int64
mfcc_std
139.45705    2
138.63795    2
149.35985    2
128.14398    1
102.01124    1
            ..
171.00209    1
161.13136    1
168.47473    1
169.90286    1
149.18895    1
Name: count, Length: 2449, dtype: int64
mfcc_min
-844.52500    2
-755.22345    1
-729.23010    1
-784.70020    1
-747.66090    1
              ..
-839.23114    1
-891.49050    1
-890.57623    1
-925.34860    1
-799.51010    1
Name: count, Length: 2451, dtype: int64
mfcc_max
227.80377    2
182.00595    2
183.27990    2
171.69092    1
206.03415    1
            ..
228.63307    1
221.48242    1
237.42914    1
204.68369    1
219.52780    1
Name: count, Length: 2449, dtype: int64
sc_mean
4923.917032    2
5792.550744    1
5279.163053    1
5898.306578    1
4223.683498    1
               ..
4889.582524    1
6715.377639    1
4612.975400    1
5575.586720    1
6082.676123    1
Name: count, Length: 2451, dtype: int64
sc_std
2699.056226    2
3328.055457    1
3801.315824    1
3706.576710    1
3081.617761    1
               ..
3942.528932    1
4279.151195    1
3767.339065    1
3841.913477    1
```

```
3963.725117    1
Name: count, Length: 2451, dtype: int64
sc_min
0.000000       1021
929.402795        2
988.012034        1
1196.107458       1
1612.972969       1
                ...
1031.733709       1
1218.763756       1
996.950540        1
901.738627        1
760.822547        1
Name: count, Length: 1431, dtype: int64
sc_max
12000.000120    3
12000.000156    3
12000.000219    3
12000.000034    3
11999.999899    3
                ..
9658.847190     1
11124.098748    1
12000.000018    1
9040.816085     1
12199.773419    1
Name: count, Length: 2423, dtype: int64
sc_kur
-1.183524    2
-1.120769    1
-1.444946    1
-1.556946    1
-0.790181    1
             ..
-1.486379    1
-1.741240    1
-1.384221    1
-1.380264    1
-1.501387    1
Name: count, Length: 2451, dtype: int64
sc_skew
 0.136688    2
 0.250940    1
 0.519839    1
 0.168139    1
 0.598204    1
            ..
 0.499113    1
-0.242630    1
 0.296817    1
 0.172072    1
 0.147574    1
Name: count, Length: 2451, dtype: int64
stft_mean
0.533131    2
0.415250    1
0.430852    1
0.496167    1
0.340012    1
           ..
0.456941    1
0.632981    1
0.436762    1
0.522623    1
0.554946    1
Name: count, Length: 2451, dtype: int64
stft_std
0.298766    2
0.335533    1
0.359215    1
0.319616    1
0.352159    1
           ..
0.356365    1
0.298679    1
0.353029    1
0.316908    1
0.320787    1
Name: count, Length: 2451, dtype: int64
stft_min
0.000000    1021
0.002504       2
```

```
0.003353       1
0.000243       1
0.000705       1
            ...
0.003993       1
0.003008       1
0.003193       1
0.004626       1
0.001565       1
Name: count, Length: 1431, dtype: int64
stft_max
1.0    2452
Name: count, dtype: int64
stft_kur
-1.169462    2
-1.215025    1
-1.478790    1
-1.334803    1
-1.070674    1
            ..
-1.506614    1
-0.997259    1
-1.420758    1
-1.255202    1
-1.257666    1
Name: count, Length: 2451, dtype: int64
stft_skew
-0.007618    2
 0.403514    1
 0.225088    1
 0.067770    1
 0.635253    1
            ..
 0.153232    1
-0.550228    1
 0.150741    1
-0.070329    1
-0.237757    1
Name: count, Length: 2451, dtype: int64
mean
-2.111432e-08    2
 7.334097e-07    2
 1.577629e-06    1
 4.042853e-08    1
 1.588445e-05    1
                ..
-7.093227e-07    1
 1.198948e-06    1
 1.436278e-07    1
-5.845742e-07    1
 6.342640e-07    1
Name: count, Length: 2450, dtype: int64
std
0.006793    2
0.014482    1
0.023513    1
0.009701    1
0.017741    1
           ..
0.007897    1
0.003108    1
0.004464    1
0.002861    1
0.010001    1
Name: count, Length: 2451, dtype: int64
min
-0.113922    4
-0.069153    4
-0.998810    3
-0.102325    3
-0.027527    3
            ..
-0.074799    1
-0.062347    1
-0.024017    1
-0.210236    1
-0.081512    1
Name: count, Length: 2148, dtype: int64
max
0.998810    14
0.055176     4
0.062256     4
0.054779     3
```

```
0.053070     3
            ..
0.148468     1
0.185303     1
0.396362     1
0.872070     1
0.103027     1
Name: count, Length: 2166, dtype: int64
kur
9.427334     2
9.406061     1
6.529379     1
21.931220    1
4.295619     1
            ..
2.949691     1
12.947434    1
8.559832     1
15.014362    1
12.973181    1
Name: count, Length: 2451, dtype: int64
skew
 0.388334    2
 0.273153    1
 0.499449    1
 0.478787    1
 0.048384    1
            ..
 0.013932    1
-0.085400    1
 0.047113    1
 0.009214    1
 1.032081    1
Name: count, Length: 2451, dtype: int64
```

In [12]:
```
idx = df.loc[df['channels'] == 2].index
idx
```

Out[12]:
```
Index([287, 778, 1045, 1336, 1348, 1809], dtype='int64')
```

In [13]:
```
idx = df.loc[df['frame_width'] == 4].index
idx
```

Out[13]:
```
Index([287, 778, 1045, 1336, 1348, 1809], dtype='int64')
```

In [16]:
```
idx_sc = df.loc[df['sc_min'] == 0].index
idx_sc
idx_stft = df.loc[df['stft_min'] == 0].index
if idx_sc.equals(idx_stft):
    print(True)
else:
    print(False)
```

```
True
```

da una preliminare analisi noto che stft_max, sample_width, frame_rate e modality presentano un solo valore pertanto li posso droppare.

stft_min e sc_min presentano sono valori numerici continui, presentano circa il 45% di record = 0, sono sicuramente errori essendo la percentuale così alta decido di dropparle

infine noto che channels e frame_width coincidono, contengono solo due valori di cui uno presente 0.3%

actor ha molti valori nulli e non la trovo utile per l'analisi

In [22]:
```
df2 = df.drop(['sc_min','stft_min','sample_width','frame_rate','modality','channels','frame_width','actor','stf
```

In [23]:
```
df2
```

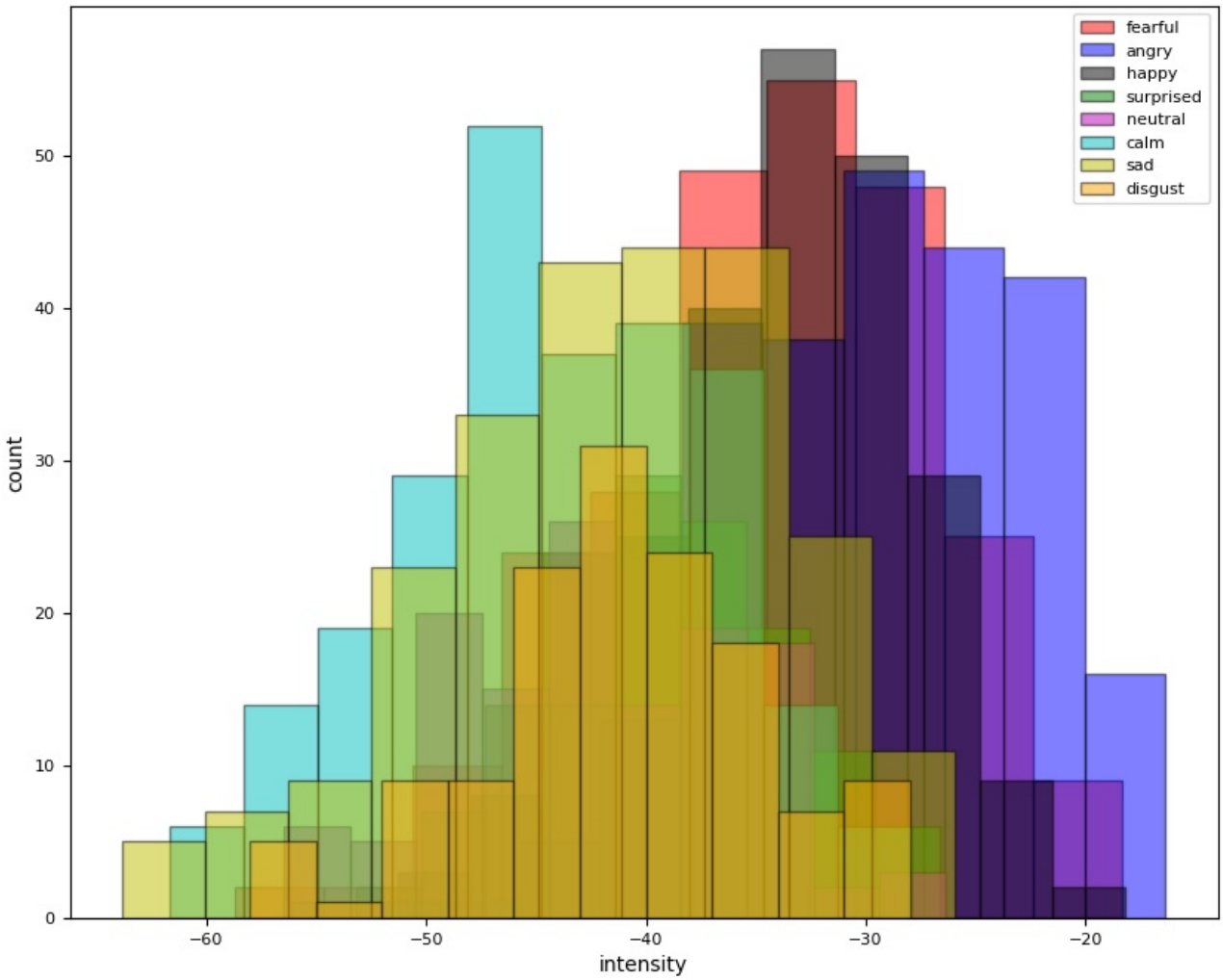
Out[23]:

| | vocal_channel | emotion | emotional_intensity | statement | repetition | sex | length_ms | frame_count | intensity | zero_crossings |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | speech | fearful | normal | Dogs are sitting by the door | 2nd | F | 3737 | 179379.0 | -36.793432 | |
| **1** | speech | angry | normal | Dogs are sitting by the door | 1st | F | 3904 | 187387.0 | NaN | |
| **2** | NaN | happy | strong | Dogs are sitting by the door | 2nd | F | 4671 | 224224.0 | -32.290737 | |
| **3** | NaN | surprised | normal | Kids are talking by the door | 1st | F | 3637 | 174575.0 | -49.019839 | |
| **4** | song | happy | strong | Dogs are sitting by the door | 2nd | F | 4404 | 211411.0 | -31.214503 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2447** | speech | calm | strong | Kids are talking by the door | 1st | M | 4605 | 221021.0 | NaN | |
| **2448** | speech | calm | normal | Dogs are sitting by the door | 1st | M | 4171 | 200200.0 | -43.342901 | |
| **2449** | song | sad | strong | Dogs are sitting by the door | 2nd | M | 5239 | 251451.0 | NaN | |
| **2450** | speech | surprised | normal | Kids are talking by the door | 1st | M | 3737 | 179379.0 | -45.751265 | |
| **2451** | NaN | neutral | normal | Dogs are sitting by the door | 2nd | M | 3837 | 184184.0 | -40.018044 | |

2452 rows × 29 columns

In [25]:
```
emozioni = list(df['emotion'].unique())
colors = ['r', 'b', 'k', 'g', 'm', 'c', 'y', 'orange']
```

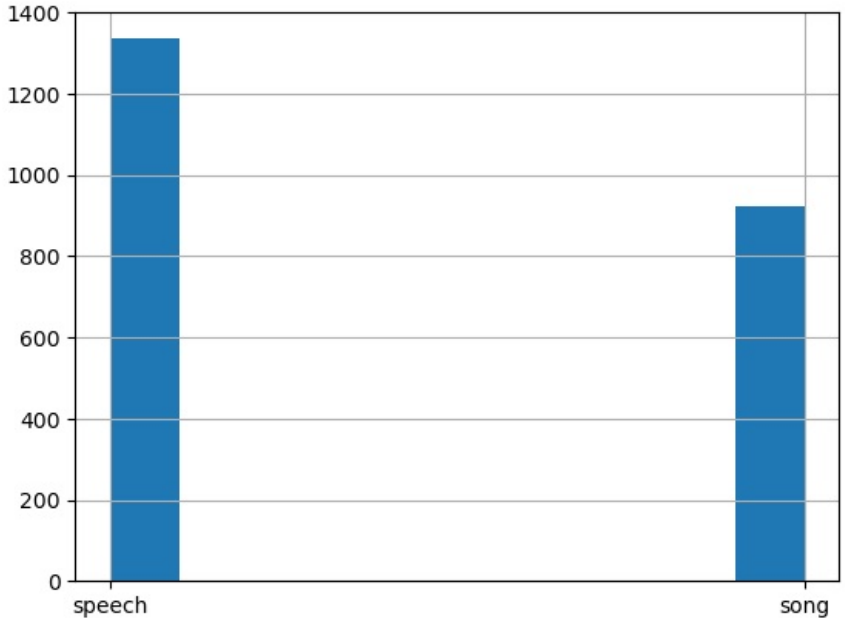
In [26]:
```
plt.figure(figsize=(10,8))
for i, val in enumerate(emozioni):
    plt.hist(df[df['emotion']==val]['intensity'], bins=10, color=colors[i], label=val, alpha=0.5,edgecolor='bla
plt.legend( fontsize=8)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.xlabel('intensity',fontsize=10)
plt.ylabel('count',fontsize=10)
plt.show()
```




provo a vedere se c'è qualche relazione tra intensity (che ha alcuni valori nulli) e le emozioni

inzio a gestire i valori nulli

In [28]:
```
df2['vocal_channel'].hist()
plt.show()
```




In [29]:
```
df2['vocal_channel'].value_counts(normalize=True)
```

Out[29]:
```
vocal_channel
speech    0.591755
song      0.408245
Name: proportion, dtype: float64
```

In [39]:
```
def missing():
    scelta = np.random.choice(['speech', 'song'], p=[0.591, 0.409])
```

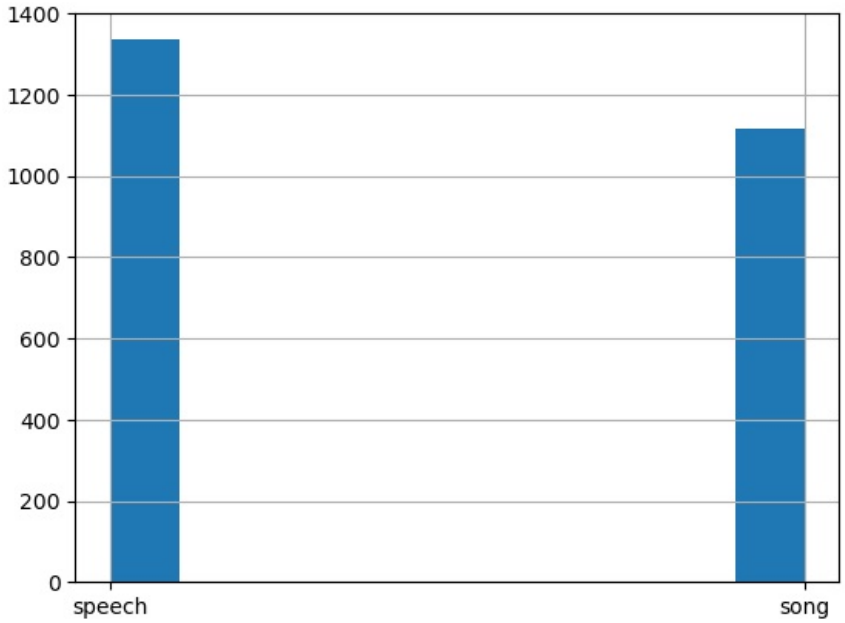
```
    return scelta
```

In [40]:
```
df2['vocal_channel'].fillna(missing(), inplace=True)
```

In [41]:
```
df2['vocal_channel'].value_counts(normalize=True)
```

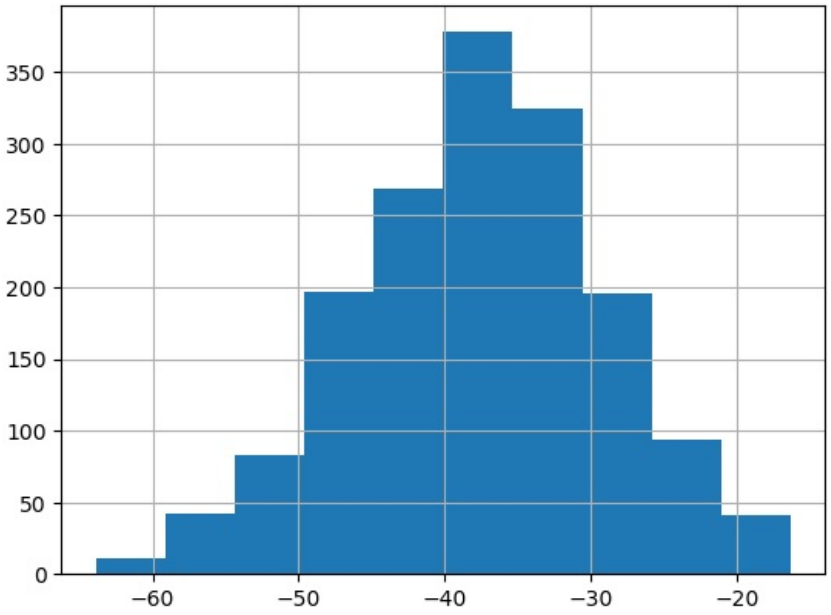
Out[41]:
```
vocal_channel
speech    0.544454
song      0.455546
Name: proportion, dtype: float64
```

In [42]:
```
df2['vocal_channel'].hist()
plt.show()
```




decido di sostituire i valori mancanti con valori generati casualmente pesati rispetto alla percentuale dentro il dataset

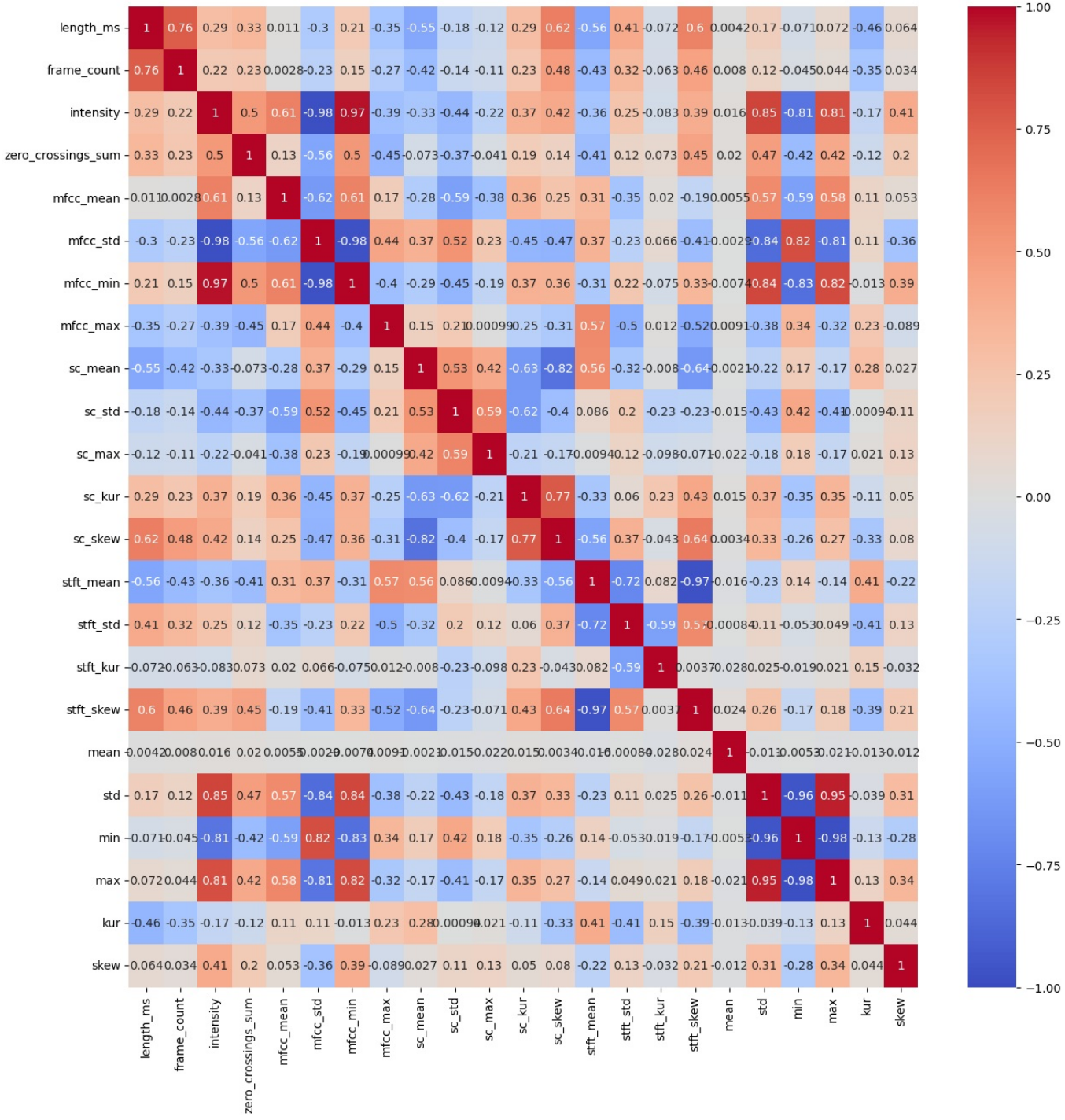
In [43]:
```
df2['intensity'].hist()
plt.show()
```




In [47]:
```
df_numerico = df2.select_dtypes(include=['number'])
correlazione = df_numerico.corr()
```

In [49]:
```
plt.figure(figsize=(15,15))
sns.heatmap(correlazione,cmap='coolwarm', vmin=-1, vmax=1, annot=True)
```

Out[49]:
```
<Axes: >
```




noto che l'intensity è fortemente correlata a mfcc_std e mfcc_min, decido dunque di effettuare una regressione lineare per calcolare i valori mancanti

In [50]:
```
from sklearn.linear_model import LinearRegression
```

In [51]:
```
not_na_mask=~df2['intensity'].isnull().values
not_na_idx = np.argwhere(not_na_mask).ravel()# fa il contrario prende i not na

na_mask = df2['intensity'].isnull().values #trovo indici dove intensity è null
na_idx = np.argwhere(na_mask).ravel()
```

In [52]:
```
X_train = df2[['mfcc_std','mfcc_min']].values
X_train = X_train[not_na_idx]
```

In [54]:
```
y_train = df2['intensity'].values[not_na_idx]
```

In [55]:
```
regg = LinearRegression()
regg.fit(X_train,y_train)
```

Out[55]:
```
▾ LinearRegression
LinearRegression()
```

In [56]:
```
X_test = df2[['mfcc_std','mfcc_min']].values
X_test = X_test[na_idx]
```

In [57]:
```
y_test = regg.predict(X_test)
```

In [61]:
```
df_numerico['intensity'][na_idx] = y_test
```

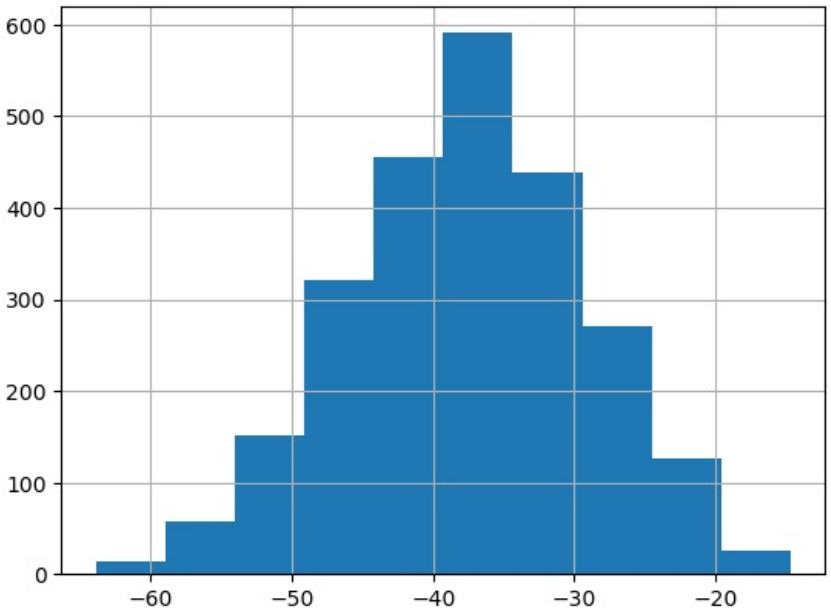
```
C:\Users\dimit\AppData\Local\Temp\ipykernel_10572\620500613.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_numerico['intensity'][na_idx] = y_test
```

In [64]:
```
df_numerico['intensity']
```

Out[64]:
```
0      -36.793432
1      -34.546284
2      -32.290737
3      -49.019839
4      -31.214503
          ...
2447   -46.264888
2448   -43.342901
2449   -38.245412
2450   -45.751265
2451   -40.018044
Name: intensity, Length: 2452, dtype: float64
```

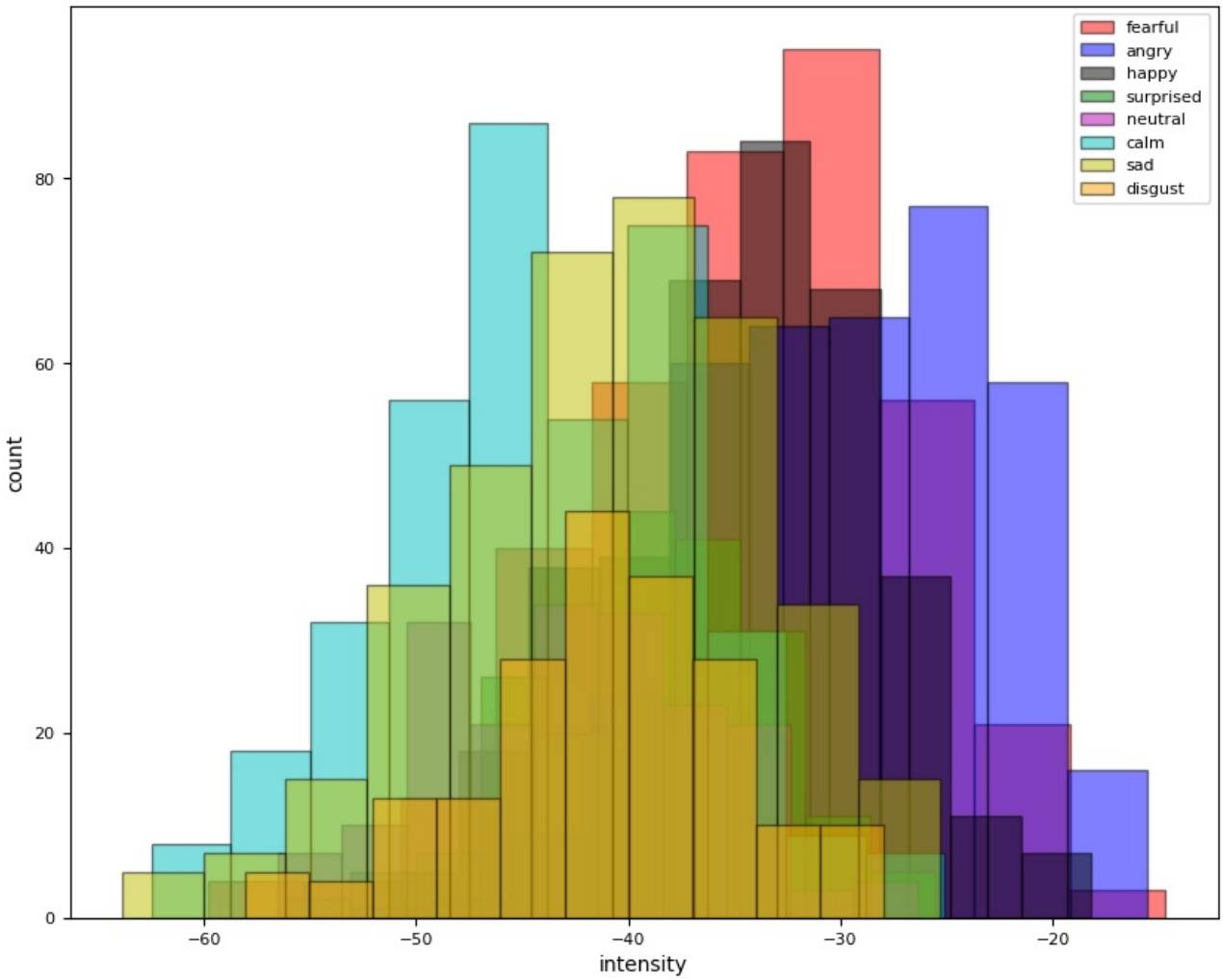
In [63]:
```
df_numerico['intensity'].hist()
```

Out[63]:
```
<Axes: >
```




l'histogramma rimane molto simile, si alzano i valori prima del bin che rappresenta il maggior numero di record e calano quelli dopo

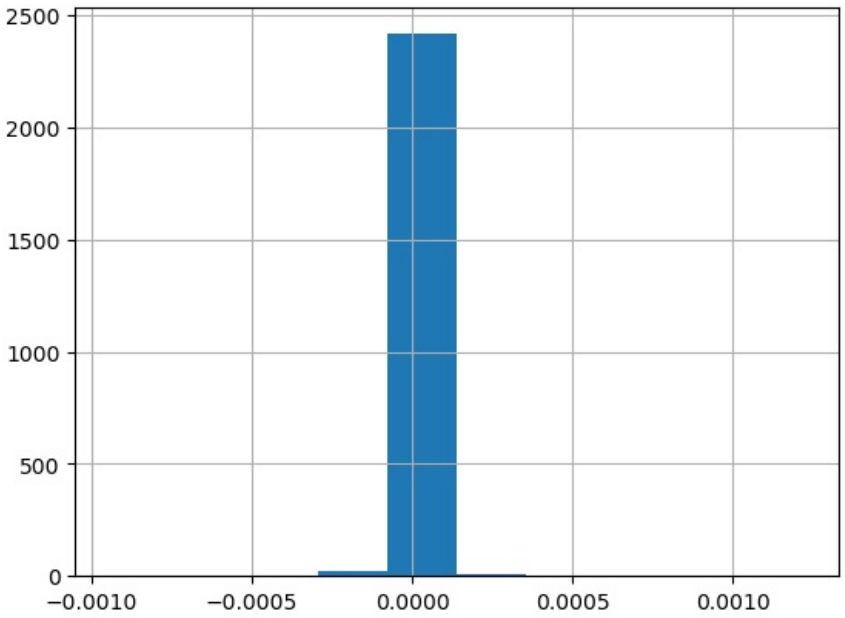
In [66]:
```
plt.figure(figsize=(10,8))
for i, val in enumerate(emozioni):
    plt.hist(df2[df2['emotion']==val]['intensity'], bins=10, color=colors[i], label=val, alpha=0.5,edgecolor='b
plt.legend( fontsize=8)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.xlabel('intensity',fontsize=10)
plt.ylabel('count',fontsize=10)
plt.show()
```



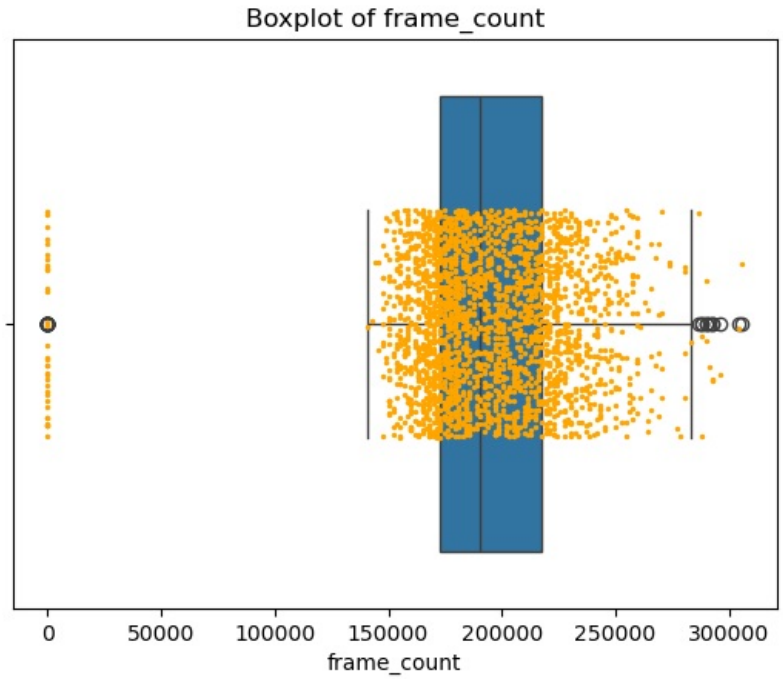

inizio ora lo studio per valutare quali variabili utilizzare successivamente per il clustering

In [67]:
```
df2['mean'].hist()
```

Out[67]: <Axes: >




In [3]:
```
ax = sns.boxplot(data= df, x ='frame_count')
ax = sns.stripplot(data=df, x='frame_count', color="orange", jitter=0.2, size=2.5)
plt.title('Boxplot of frame_count')
plt.show()
```

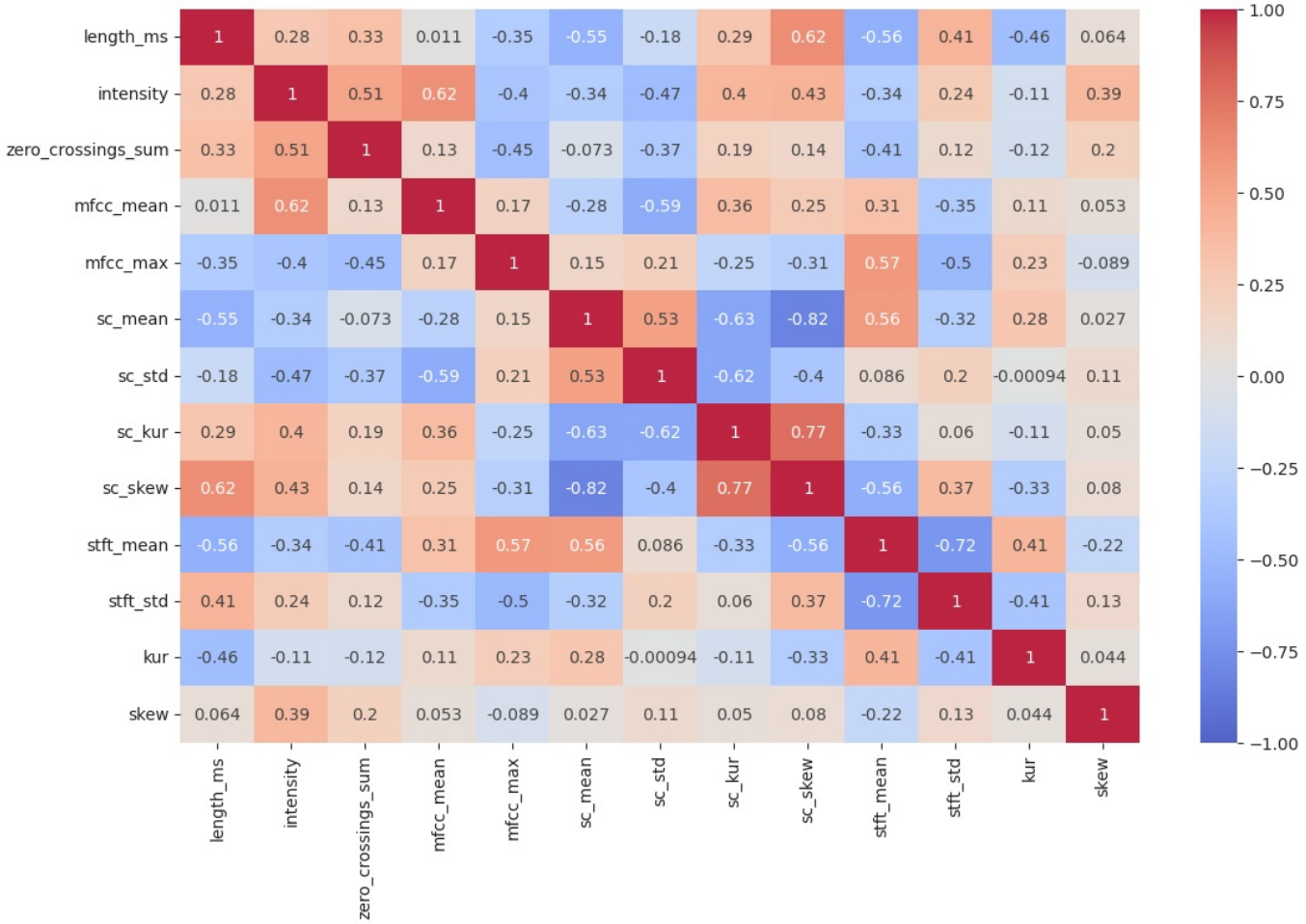



mean non ha correlazioni e sono valori prossimo allo zero dentro unico bin. std, min, e max valori correlazione molto simili nel df, terrei solo la std perchè tra le tre è quella che mi pare avere valori di correlazione leggermente più alti elimino mfcc_min perchè simile a mfcc std mfcc_std simile a std quindi lo tolgo std e intensity molto simili, tengo intensity sc_max simile a sc_std, elimino il primo skew correlato solo intensià e std ma decido di tenerlo stft_min simile stft_mean ma meno evidente frame_count simile lenght_ms ma meno evidente. inoltre con boxplot si vedono valori errati in frame_count, infatti alcuni valori vengono negativi mentre gli altri sono nell'ordine delle centinaia di migliaia tengo stft_mean e droppo stft_skew per andamento simile. stft_kur poche correlazioni.

In [69]:
```
df_numerico2 = df.select_dtypes(include=['number'])
df_numerico2.drop(['sc_min','actor','channels','stft_max', 'frame_rate','sample_width','frame_width','stft_skew
correlazione = df_numerico2.corr()
```

In [70]:
```
plt.figure(figsize=(13,8))
sns.heatmap(correlazione,cmap='coolwarm', vmin=-1, vmax=1, annot=True)
```

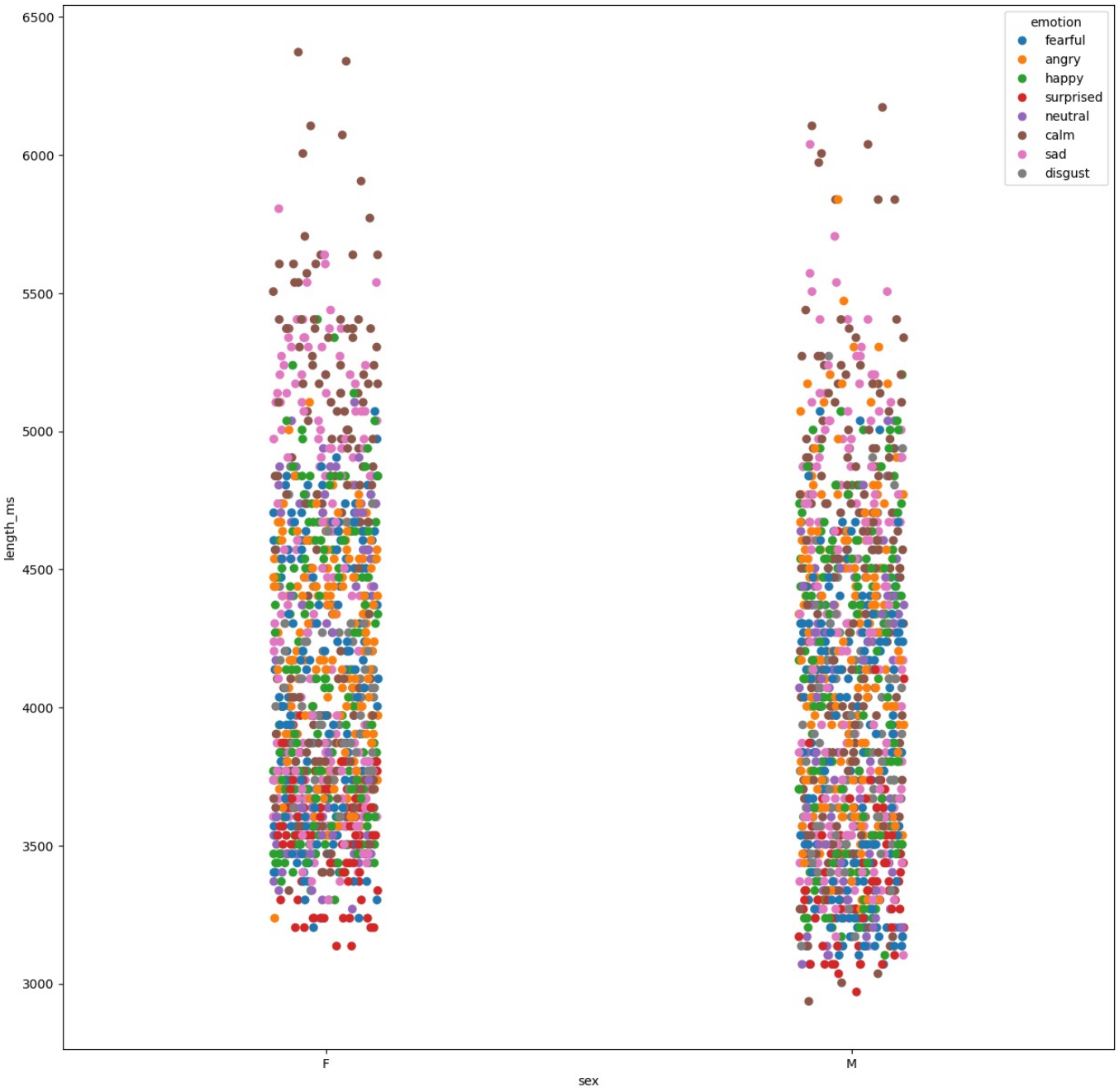
Out[70]: <Axes: >

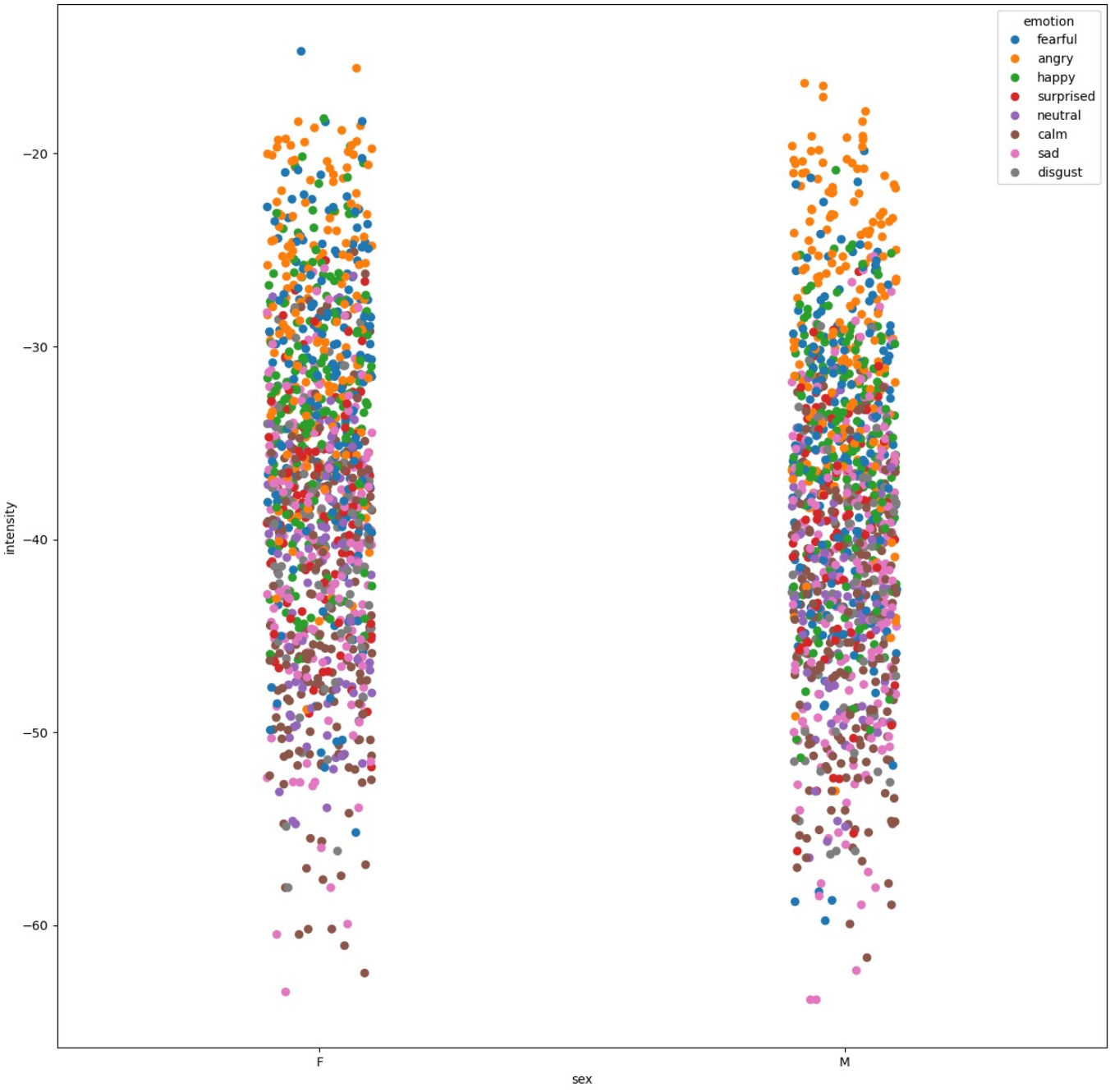


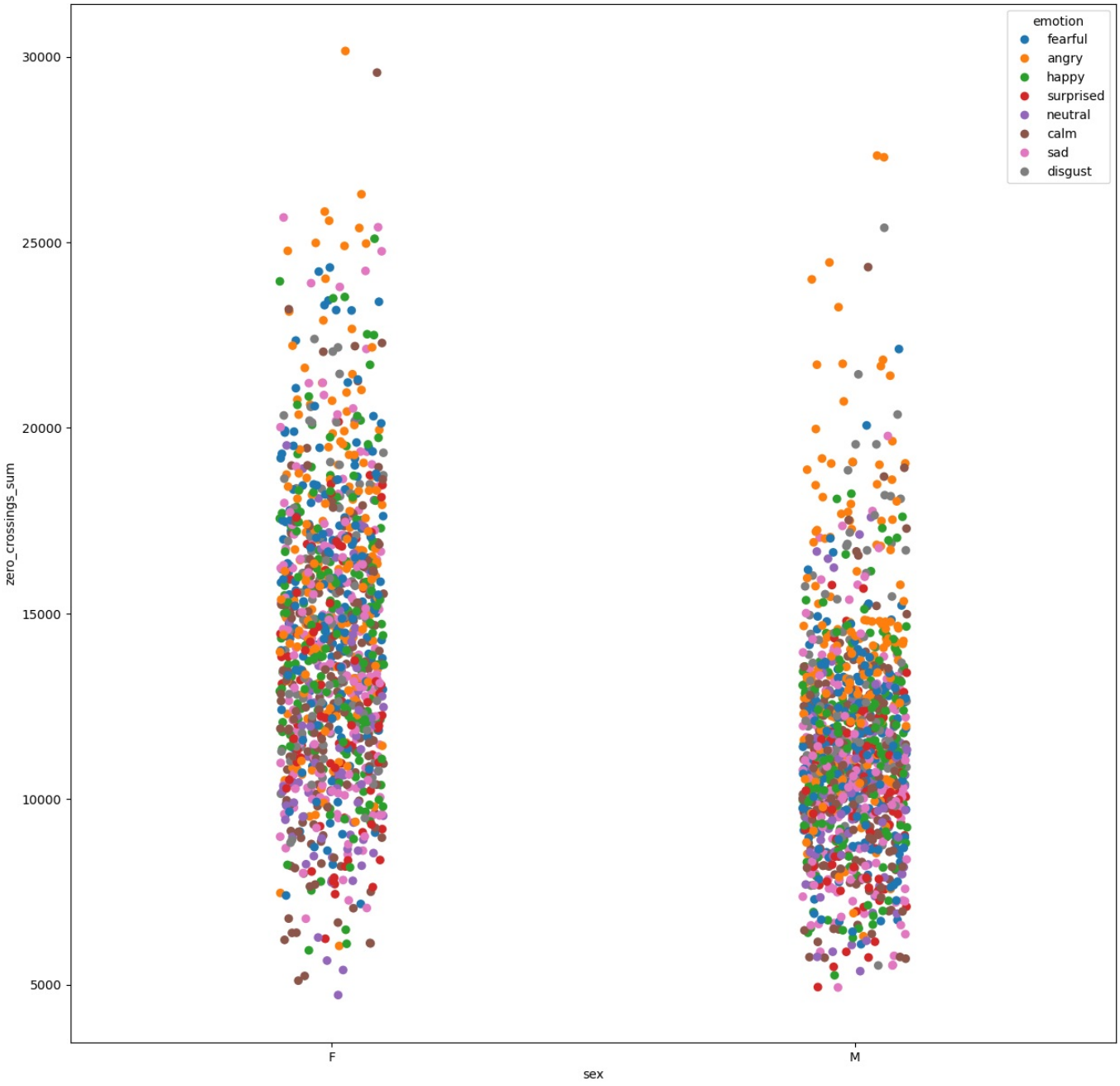

i grafici delle variabili che mi sono rimaste per capire se posso ridurle ancora

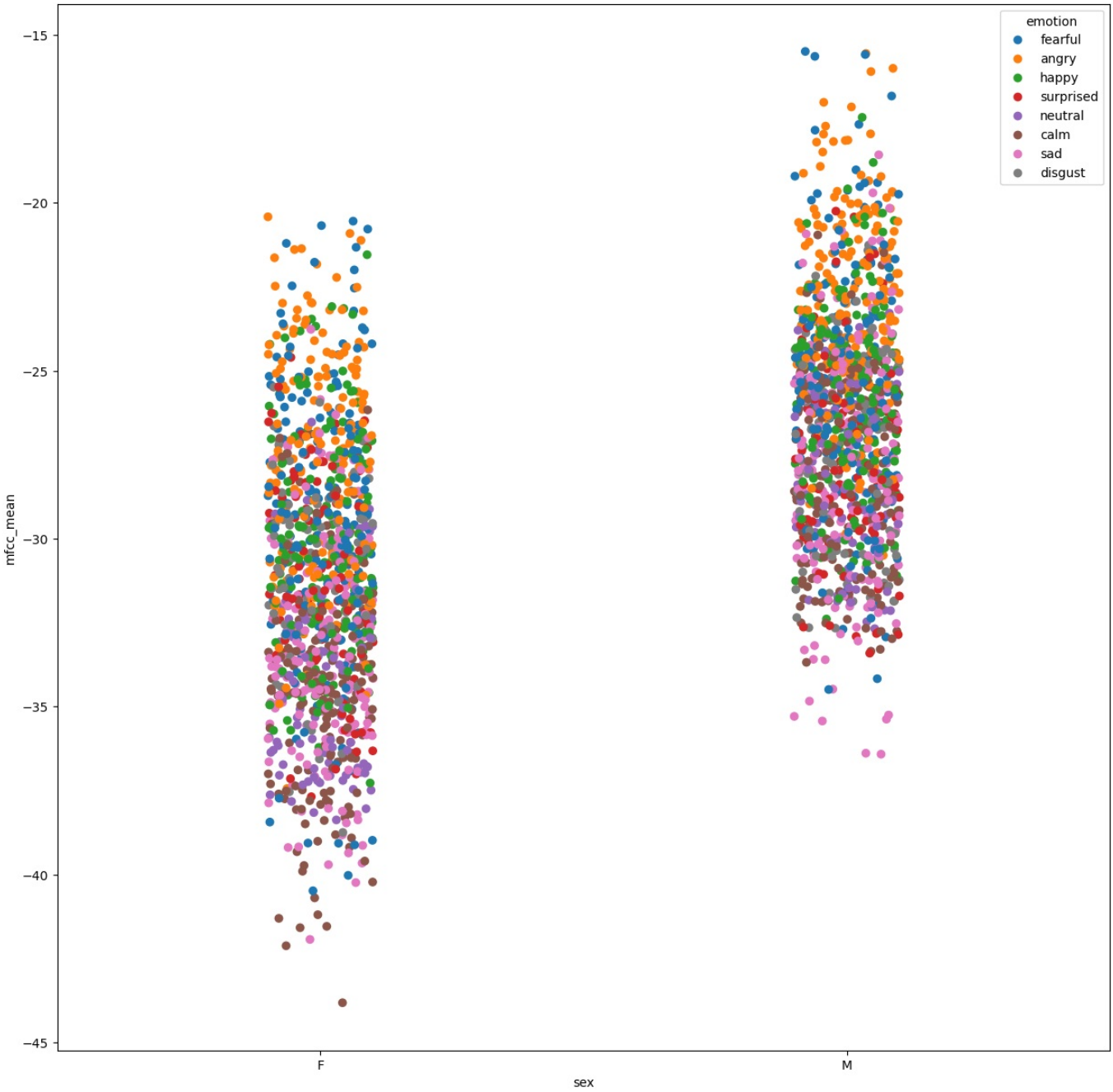
In [72]:

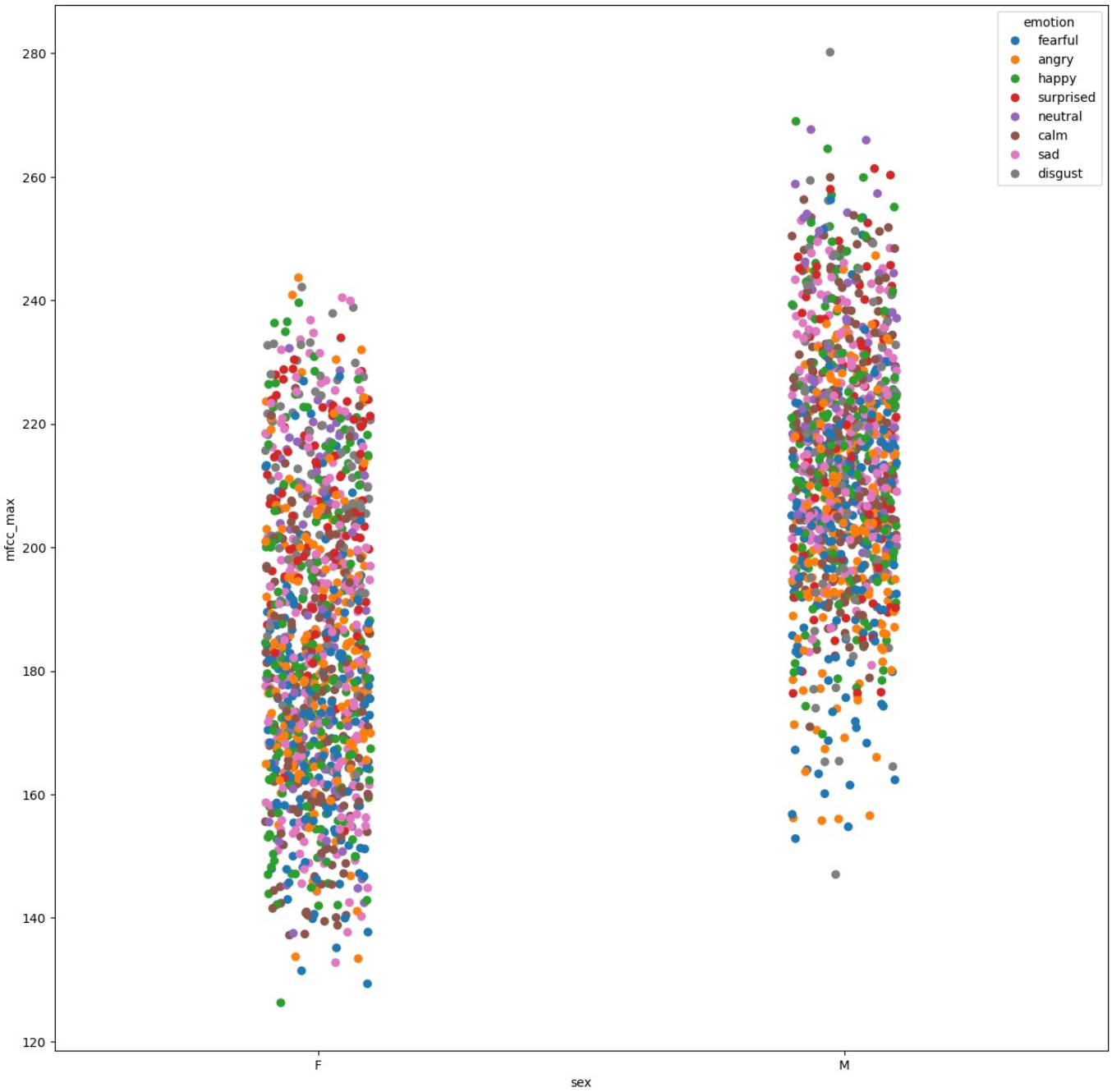
```
for i in df_numerico2:
    plt.figure(figsize=(15,15))
    sns.stripplot(data=df, x='sex', y=i ,hue='emotion',size=7)
```

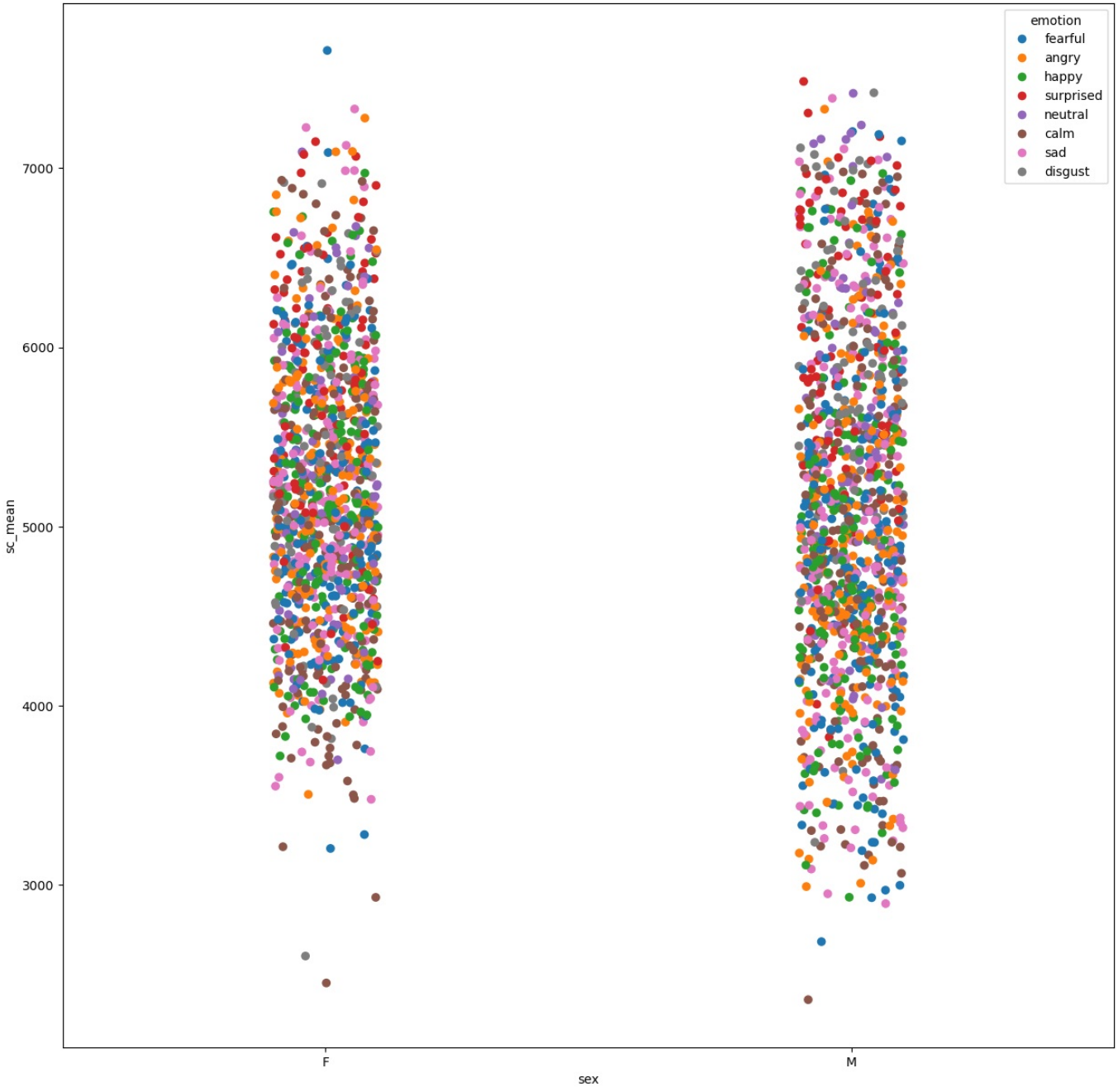


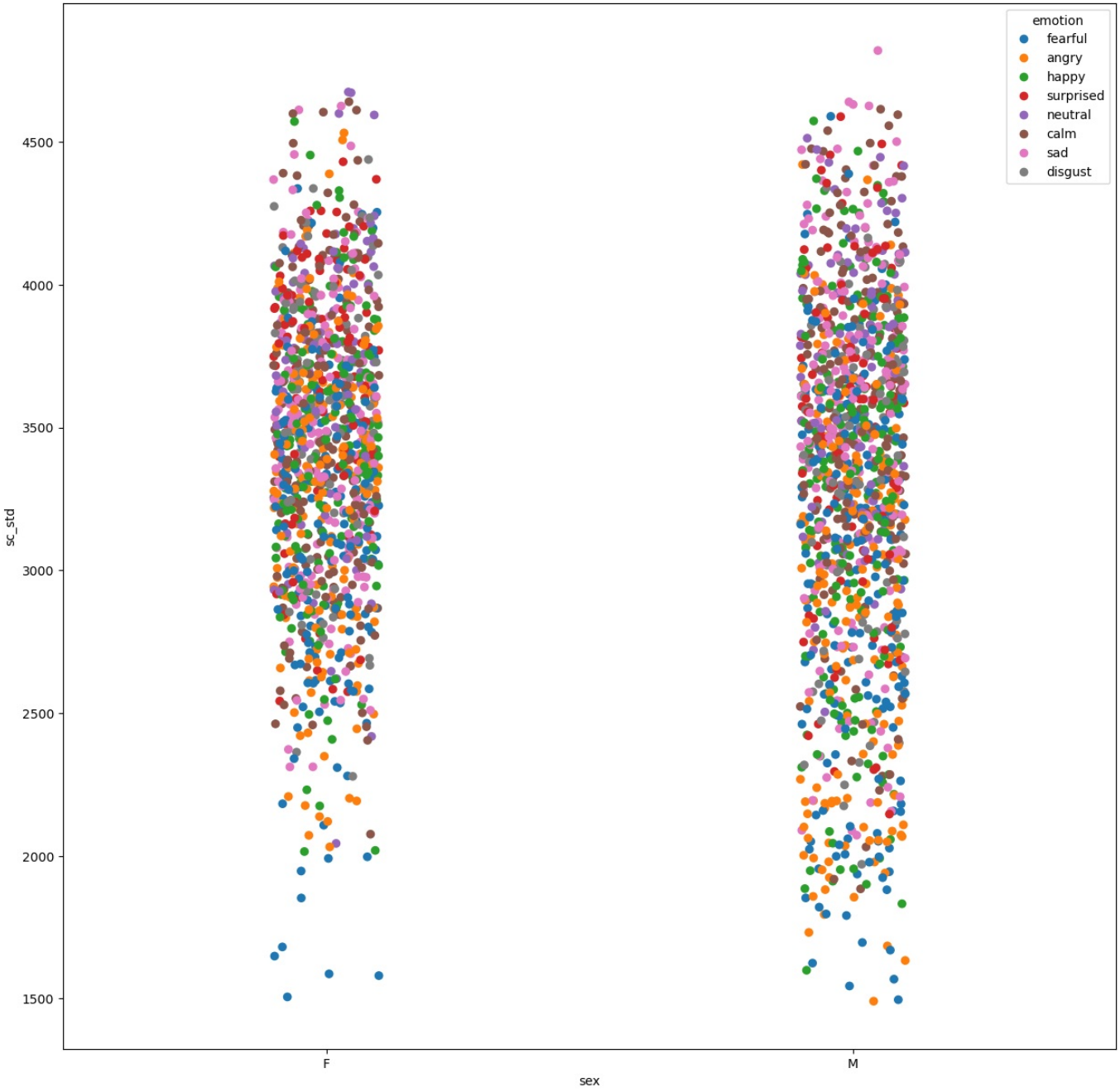

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
intensity
−20
−30
−40
−50
−60
F
M
sex

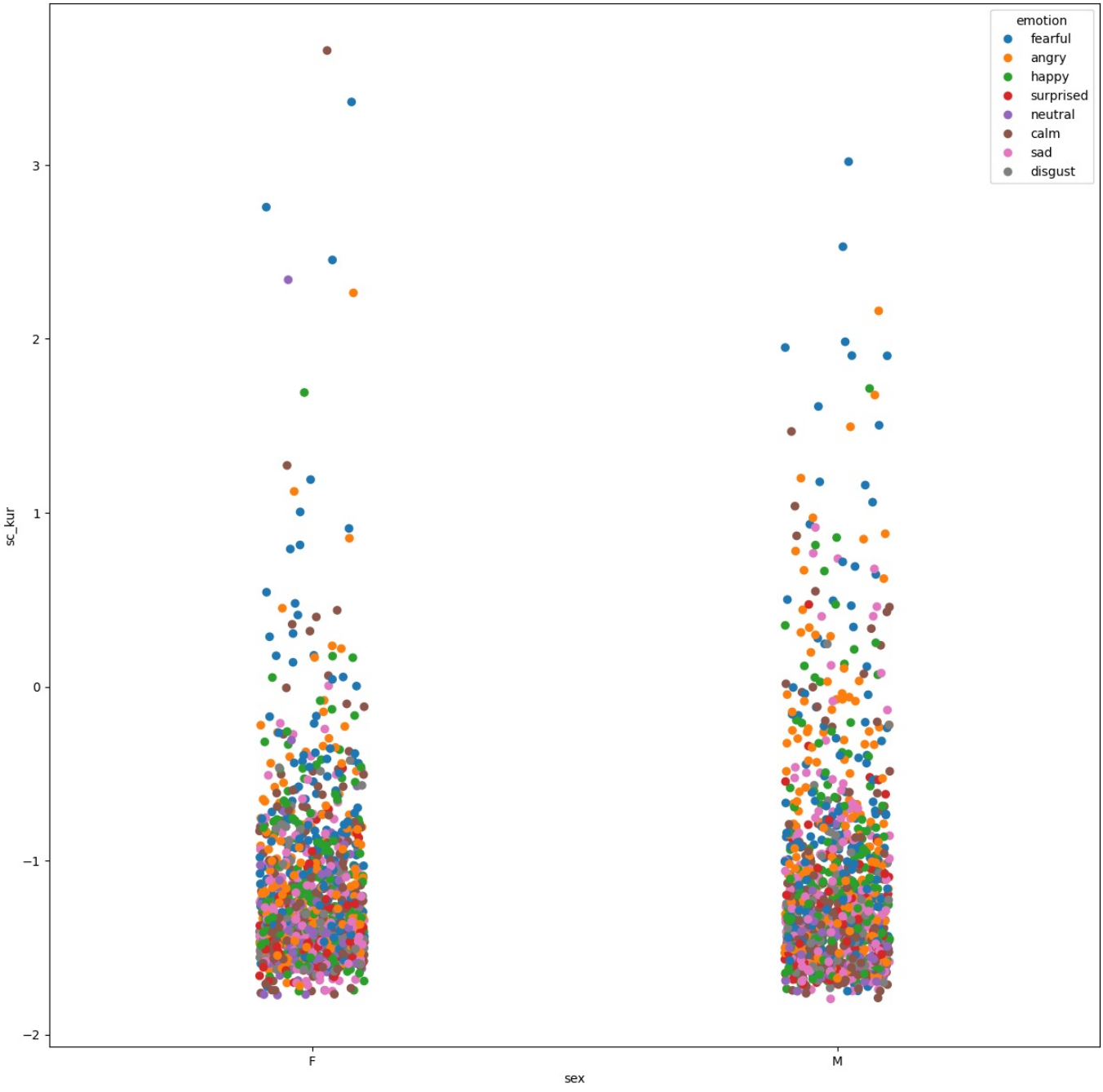

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
30000
25000
20000
15000
10000
5000
zero_crossings_sum
F
M
sex

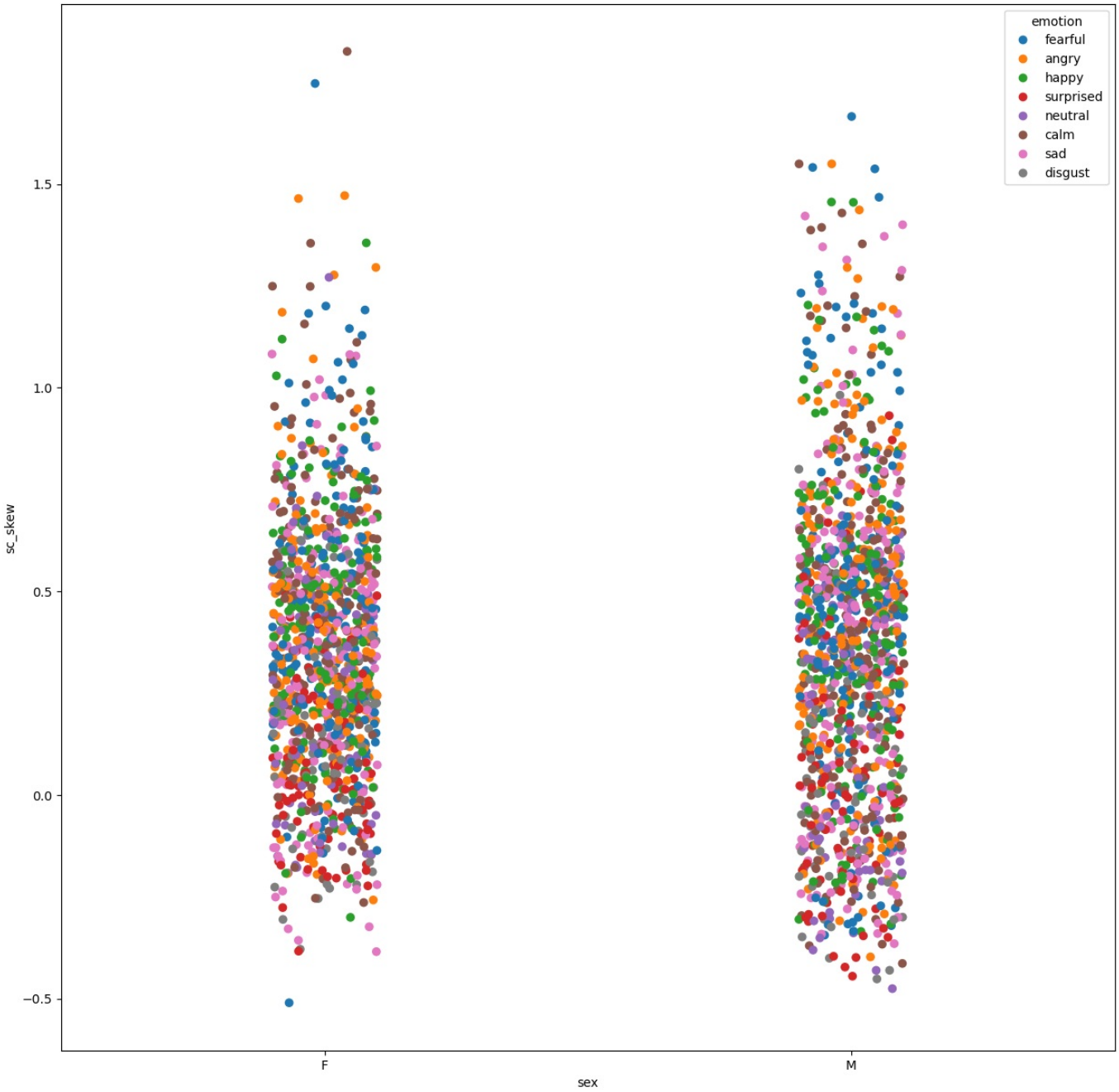

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
−15
−20
−25
−30
−35
−40
−45
mfcc_mean
F
M
sex

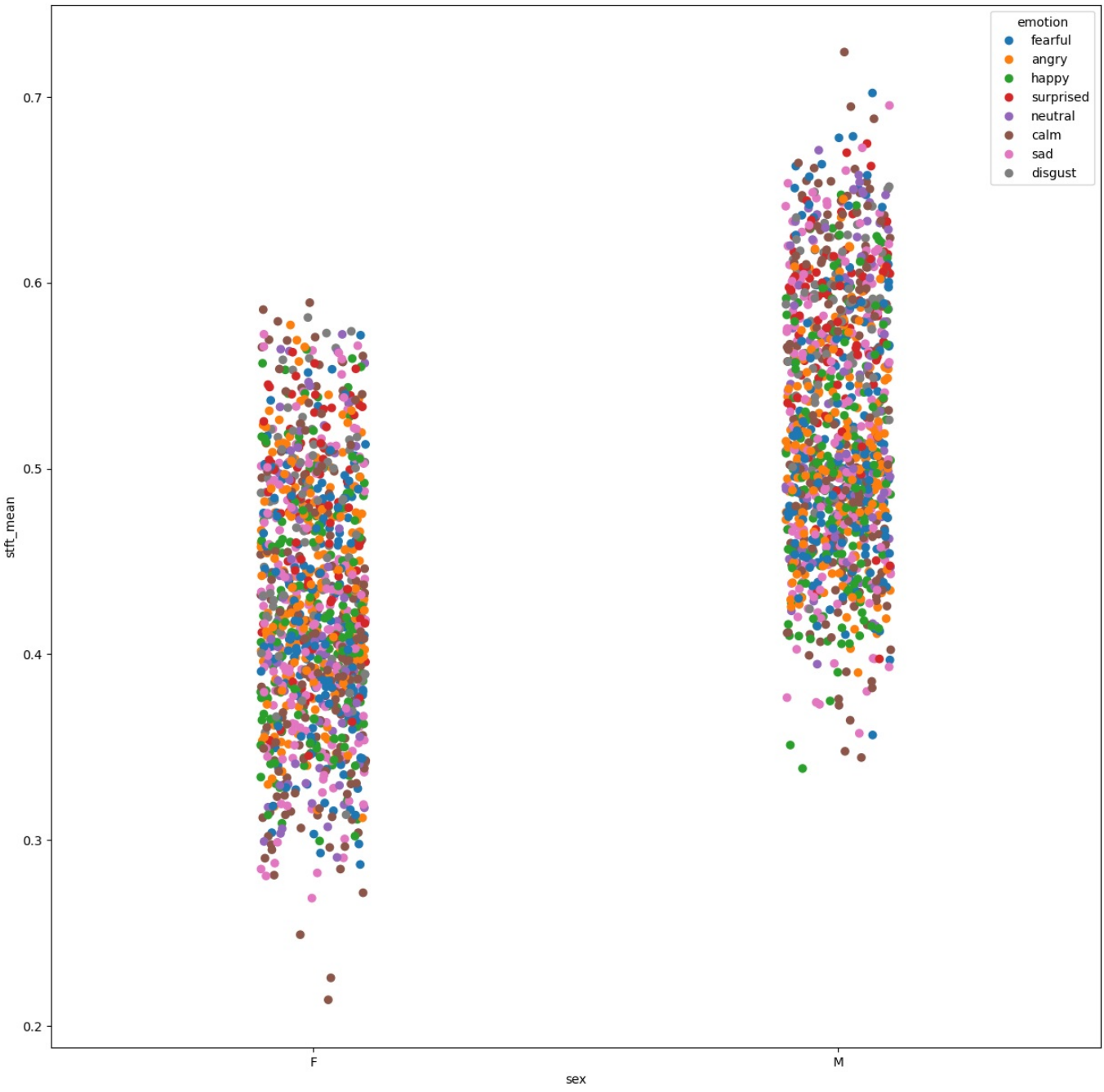

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
mfcc_max
280
260
240
220
200
180
160
140
120
F
M
sex

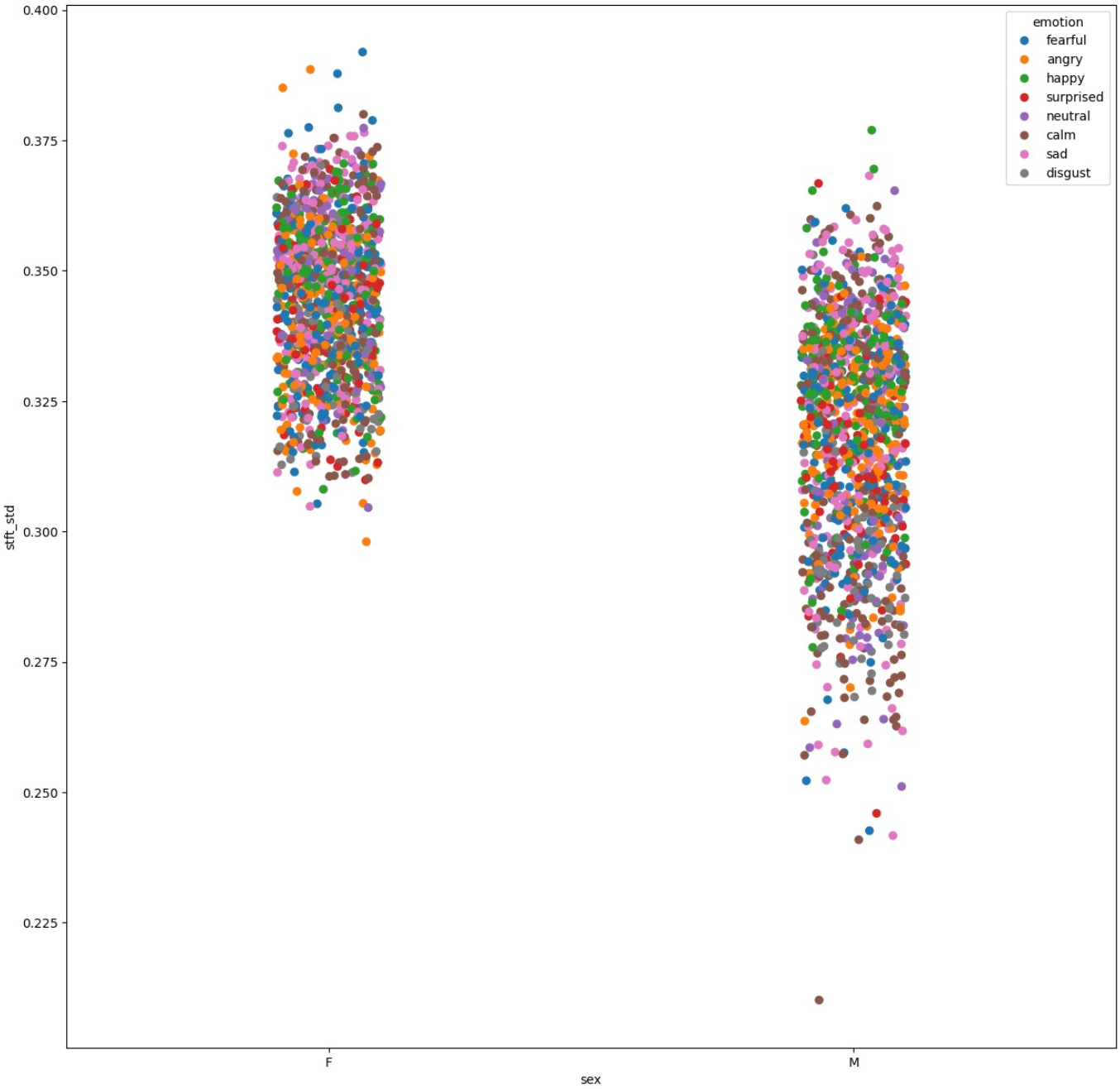

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
sc_mean
7000
6000
5000
4000
3000
F
M
sex

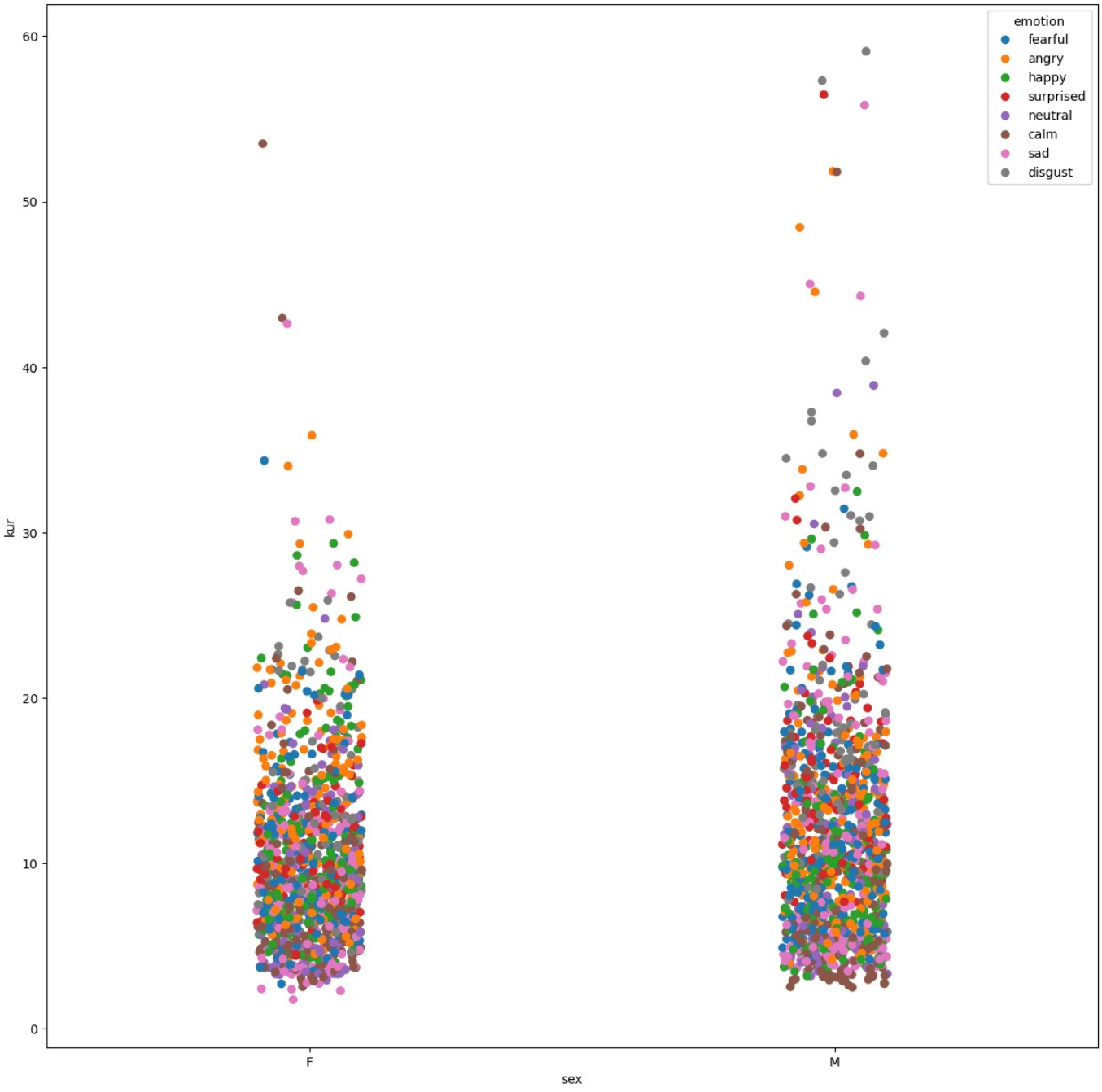

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
sc_std
4500
4000
3500
3000
2500
2000
1500
F
M
sex

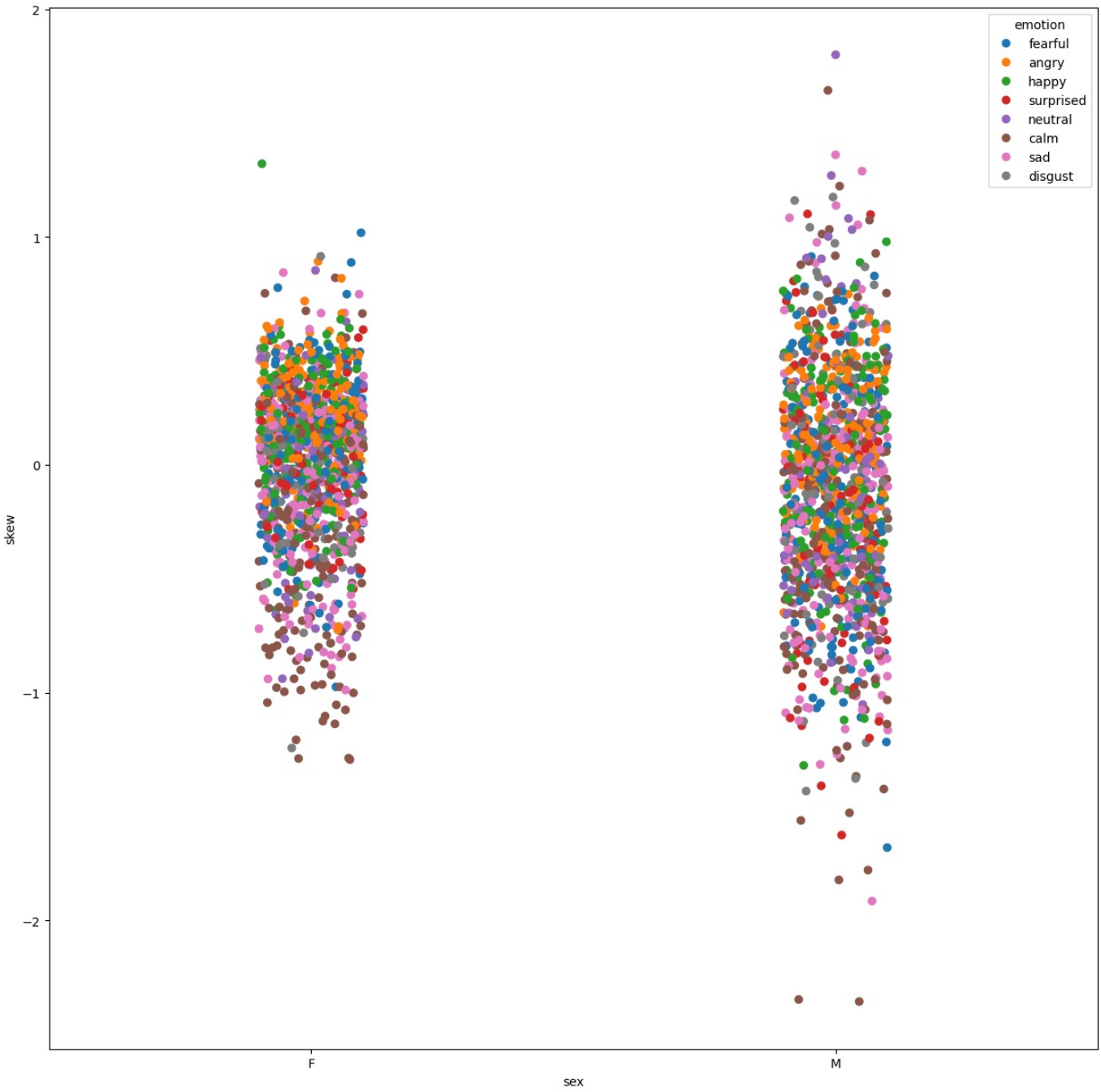

emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
sc_kur
3
2
1
0
−1
−2
F
M
sex


emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
1.5
1.0
0.5
0.0
−0.5
sc_skew
F
M
sex


emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
stft_mean
0.7
0.6
0.5
0.4
0.3
0.2
F
M
sex


emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
stft_std
F
M
sex


emotion
fearful
angry
happy
surprised
neutral
calm
sad
disgust
kur
60
50
40
30
20
10
0
F
M
sex

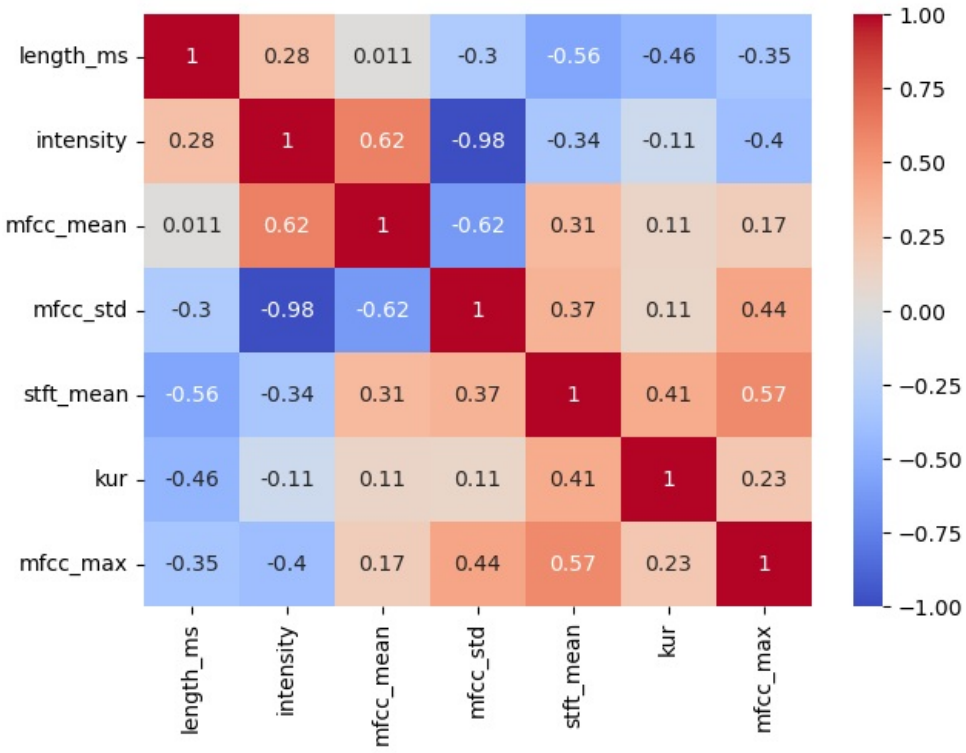



Lenght_ms uomini in media valori leggermente più bassi surprised hanno valori quasi esclusivamente bassi valori alti quasi solo calm e sad, che però si concentrano anche nella parte con valori bassi INTENSITY per m nei valori più alti angry sad e calm hanno valori bassi i valori alti sono principalmente angry, in secondo luogo fearful e happy ZERO CROSSING SUM leggermente più bassa per uomini per le emozioni non sembra troppo utile, si vede leggermente angry alta e calm bassa MFCC MEAN più alta per uomini i valori alti sono principalmente angry e fearful, quest'ultima però uniforme nei vari livelli, happy simile a fearful calm valori bassi MFCC MAX alta per uomini fearful valori bassi happy alta negli uomini ma comunque uniforme neutral alto per uomini per le donne surprised alto SC MEAN molto uniforme sc std angry fearful basse poi happy (ma sembra solo per uomini) calm, sad alte, poi neutral SCKUR tutte molto basse si vedono alcuni valori alti di calm, fearful(w) seguite da angry(m), però la maggior parte delle emozioni si trova a bassi valori sc skew disgust andamento medio basso stft_mean alta per men fearful forte picco a metà calm andamento continuo con picco a metà sad simile a calm ma picco più forte stdt_std valori bassi principalmetne calm KUR disgust abbastanza alto soprattutto uomini SKEW sembra non indicare nulla in particolarela maggior parte delle variabili sembrano differenziarsi al loro interno per i valori di angry fearful happy sad e calm. come si vedeva anche nel grafico di distribuzione delle emozioni al variare dell'intensità, sad e calm si concentrano su valori di intensità minori di -35 dBFS. mentre angry fearful ed happy su valori di intensità maggiori per queste ragioni le variabili che mi sembrano buoni indicatori delle emozioni disgust, neutral e surprised decido di tenerle. per quanto riguarda le altre variabili decido di tenere quelle maggiormente rappresentative.

In [74]:
```
df3= df2[['length_ms','intensity','mfcc_mean','mfcc_std','stft_mean','kur','mfcc_max']]
```

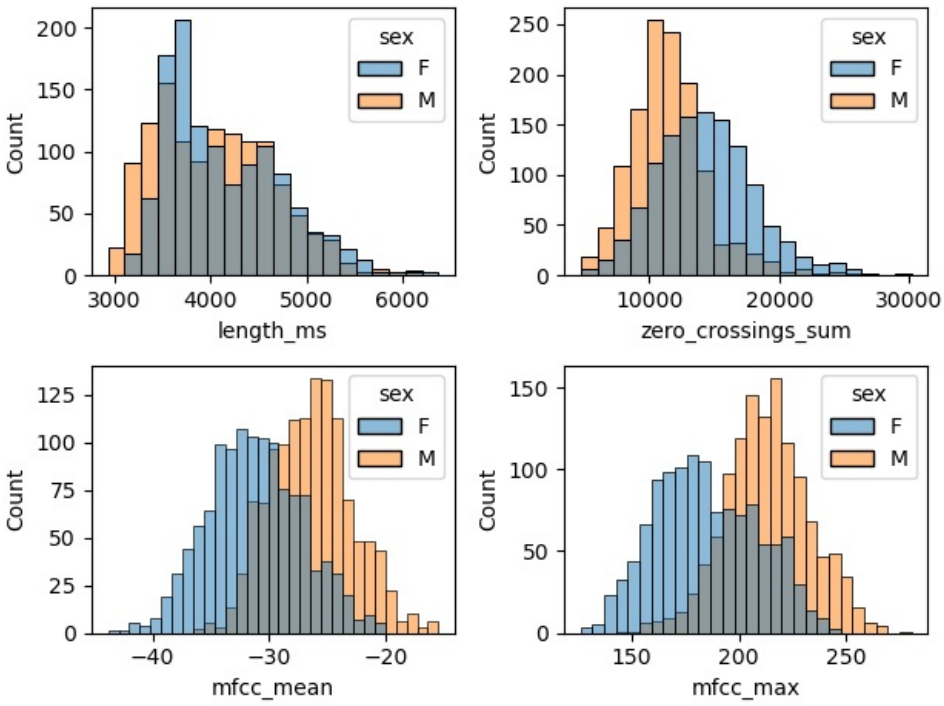
In [75]:
```
plt.figure(figsize=(7,5))
sns.heatmap(df3.corr(),cmap='coolwarm', vmin=-1, vmax=1, annot=True)
```

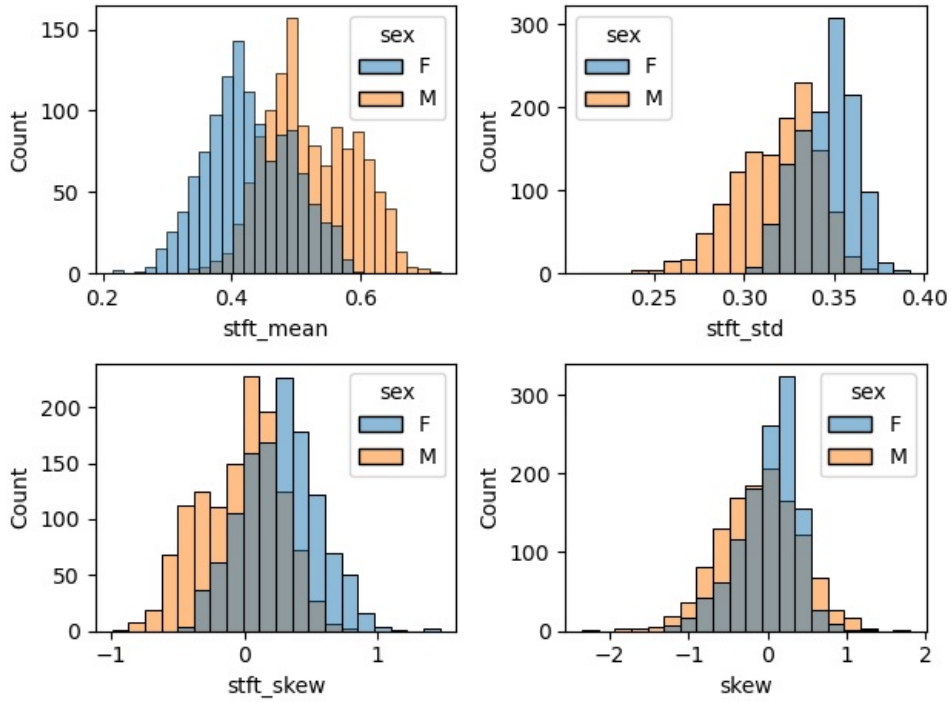
Out[75]: <Axes: >




In [4]:

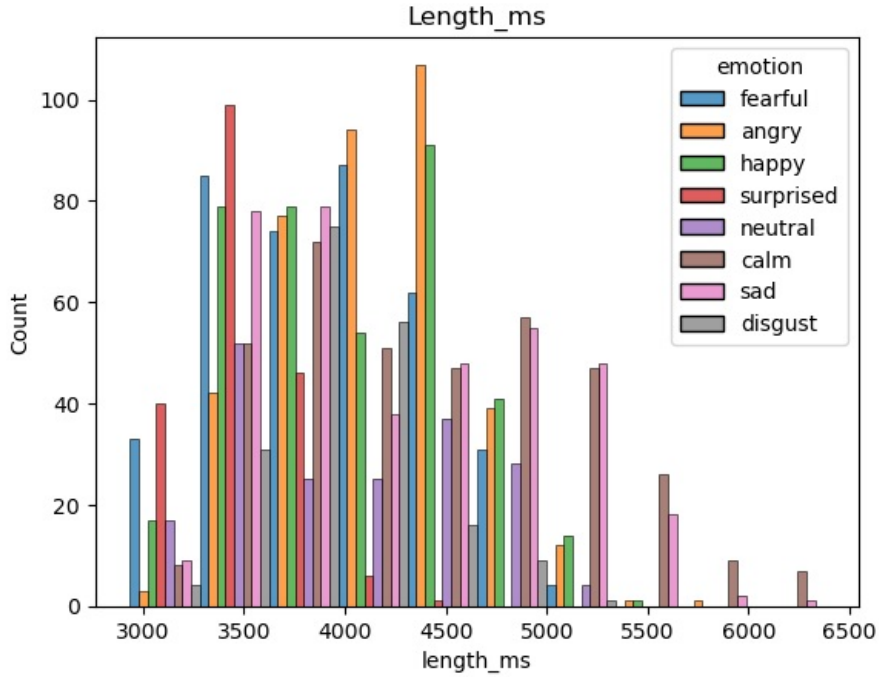
```
fig, axs = plt.subplots(2,2)
sns.histplot(data=df, x='length_ms', hue='sex',bins=20,ax=axs[0][0])
sns.histplot(data=df, x='zero_crossings_sum', hue='sex',bins=20,ax=axs[0][1])
sns.histplot(data=df, x='mfcc_mean', hue='sex',ax=axs[1][0])
sns.histplot(data=df, x='mfcc_max',  hue='sex',ax=axs[1][1])
plt.tight_layout()

fig, axs = plt.subplots(2,2)
sns.histplot(data=df, x='stft_mean', hue='sex',bins=30,ax=axs[0][0])
sns.histplot(data=df, x='stft_std',  hue='sex',bins=20,ax=axs[0][1])
sns.histplot(data=df, x='stft_skew',  hue='sex',bins=20,ax=axs[1][0])
sns.histplot(data=df, x='skew',  hue='sex',bins=20,ax=axs[1][1])
plt.tight_layout()
#plottate tutte, tengo solo queste che sono quelle che si vedono meglio
```





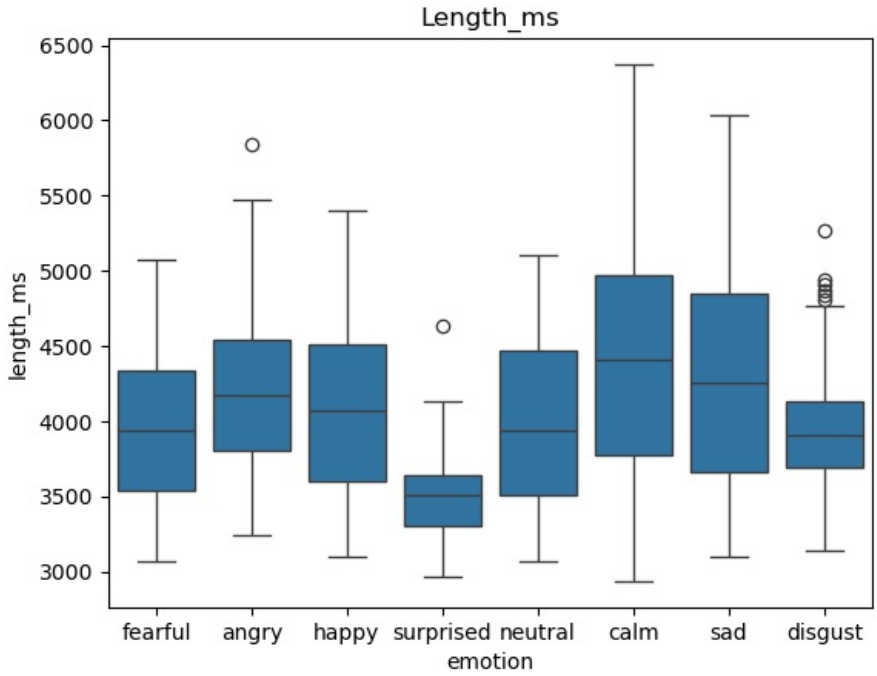

In [5]:
```
sns.histplot(data=df, x='length_ms', hue='emotion',multiple='dodge', bins=10).set(title='Length_ms')
```

Out[5]:
```
[Text(0.5, 1.0, 'Length_ms')]
```



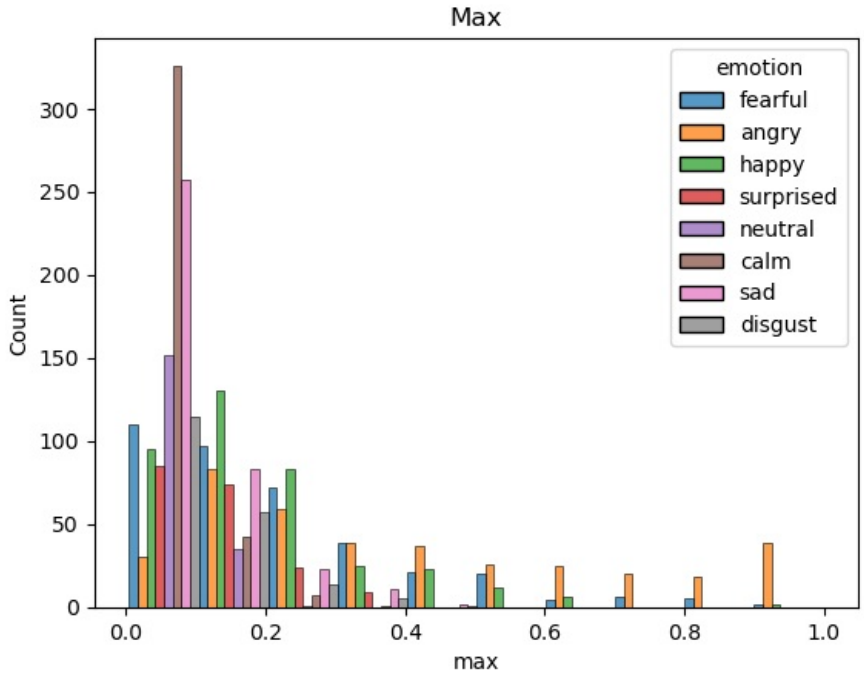

In [7]:
```
sns.boxplot(data=df, y='length_ms', x='emotion').set(title='Length_ms')
```

Out[7]:
```
[Text(0.5, 1.0, 'Length_ms')]
```



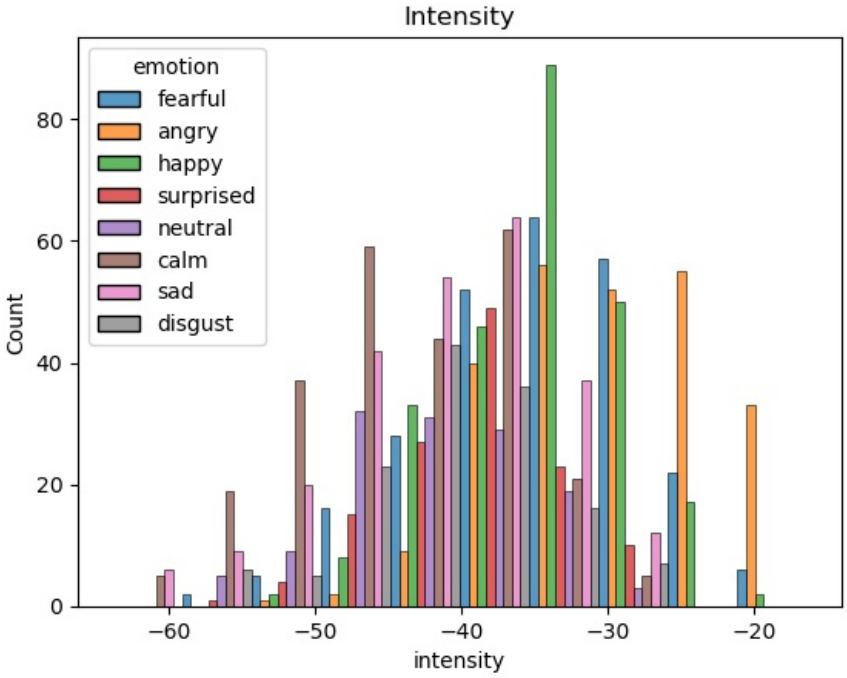

In [8]:
```
sns.histplot(data=df, x='max', hue='emotion',bins=10,multiple='dodge').set(title='Max')
```

Out[8]:
```
[Text(0.5, 1.0, 'Max')]
```



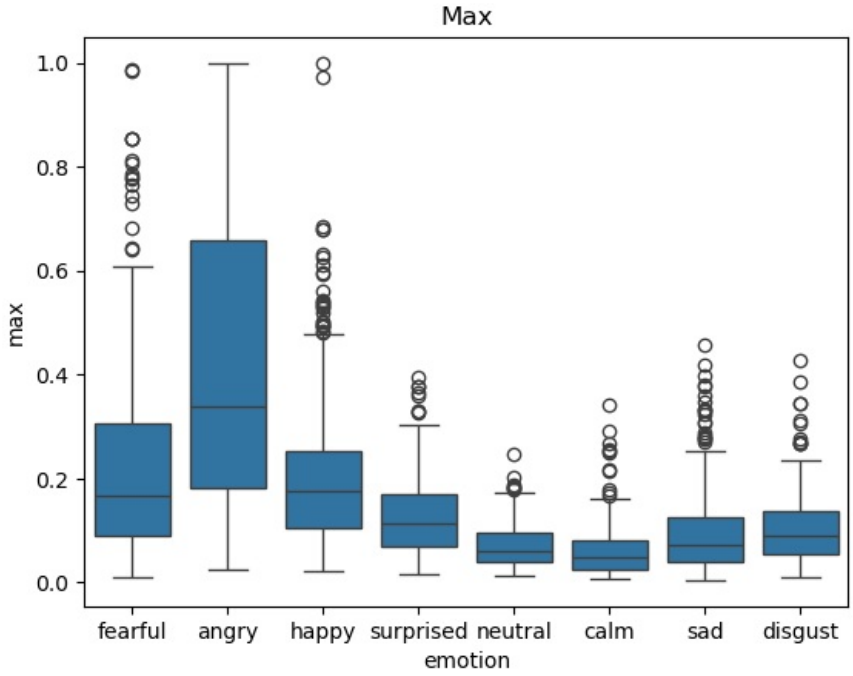

In [13]:
```
sns.histplot(data=df, x='intensity', hue='emotion',bins=10,multiple='dodge').set(title='Intensity')
```

Out[13]:
```
[Text(0.5, 1.0, 'Intensity')]
```



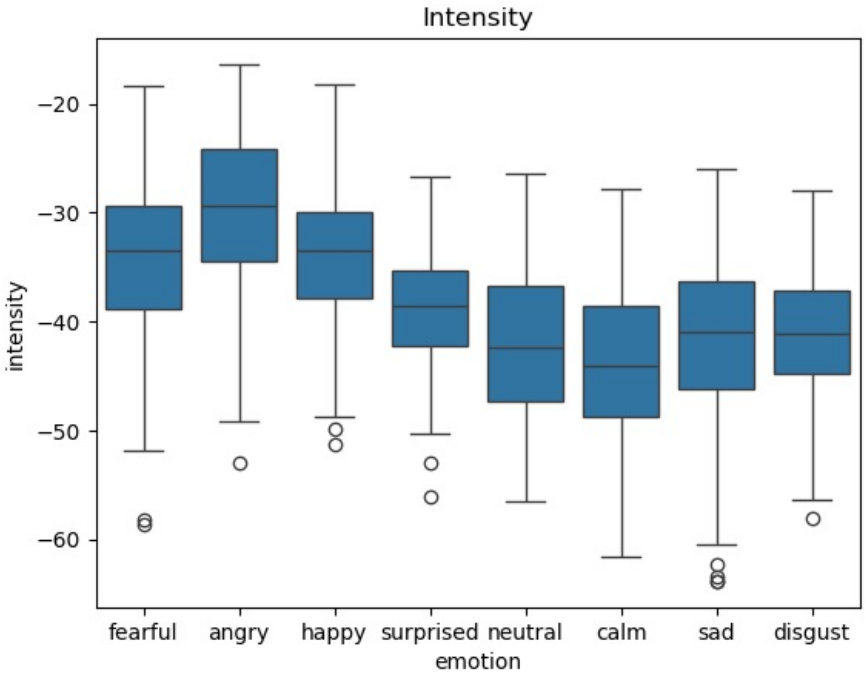

In [9]:
```
sns.boxplot(data=df, y='max', x='emotion').set(title='Max')
```

Out[9]:
```
[Text(0.5, 1.0, 'Max')]
```



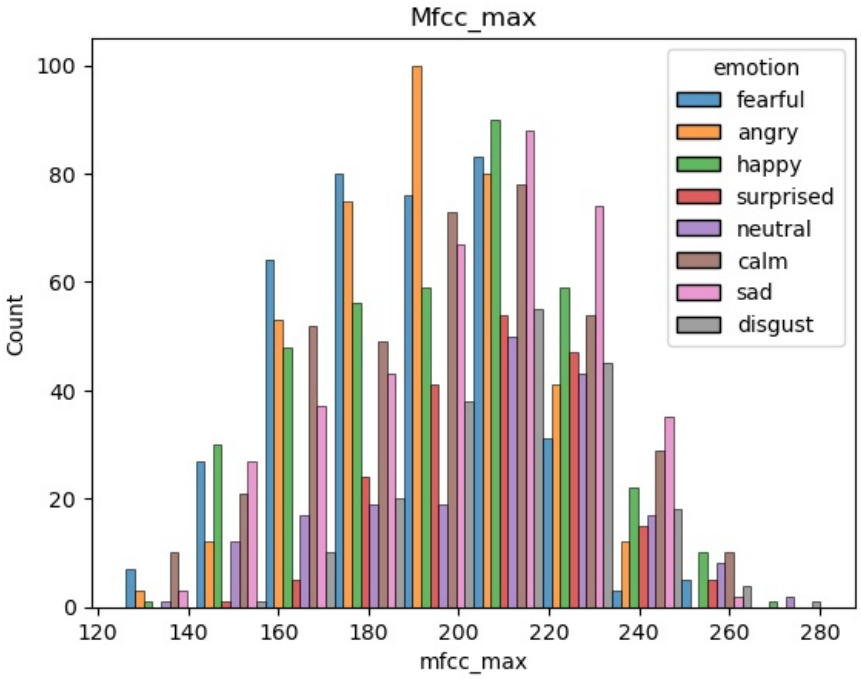

In [14]:
```
sns.boxplot(data=df, y='intensity', x='emotion').set(title='Intensity')
```

Out[14]:
```
[Text(0.5, 1.0, 'Intensity')]
```



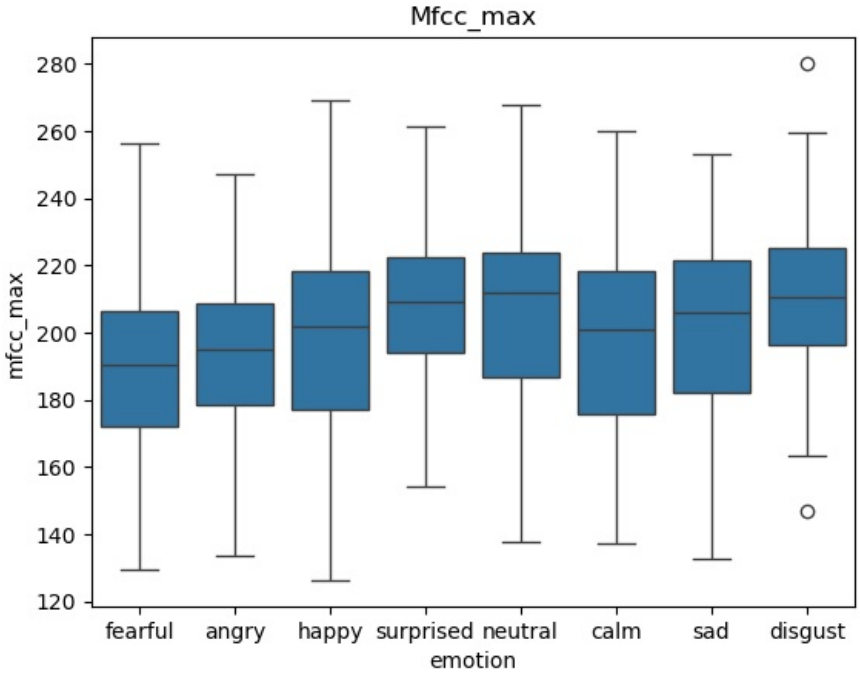

In [10]:
```
sns.histplot(data=df, x='mfcc_max', hue='emotion',bins=10,multiple='dodge').set(title='Mfcc_max')
```

Out[10]:
```
[Text(0.5, 1.0, 'Mfcc_max')]
```




In [11]:
```
sns.boxplot(data=df, y='mfcc_max', x='emotion').set(title='Mfcc_max')
```

Out[11]:
```
[Text(0.5, 1.0, 'Mfcc_max')]
```




In [ ]:

# K-MEANS

Inizialmente l'algoritmo k_means è stato eseguito con le variabili continue mantenute dopo la fase di data understanding.

In [6]:
```
X = df.values
```

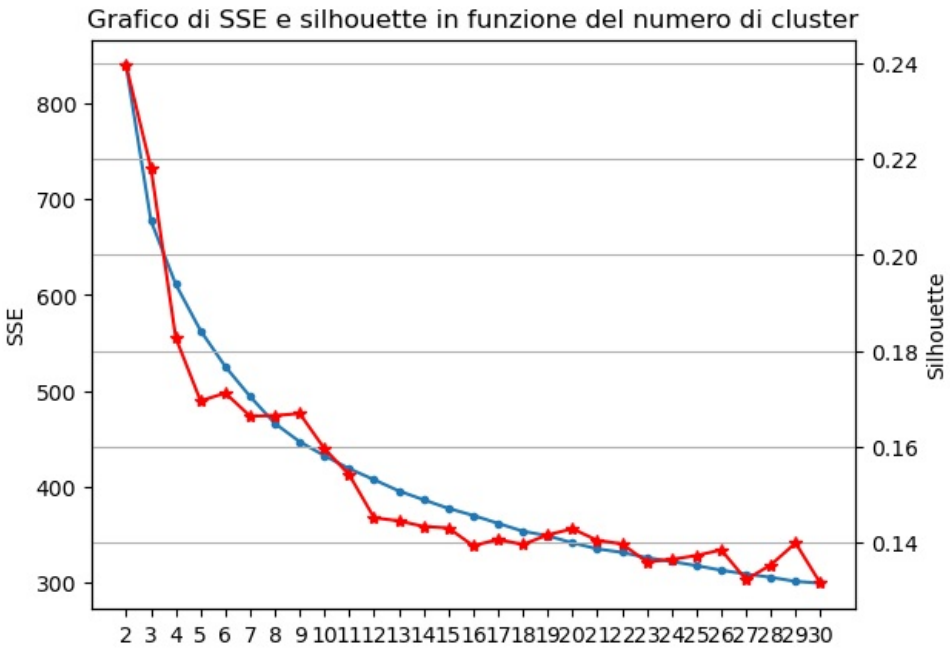
In [7]:
```
minscaler = MinMaxScaler()
X_minmax = minscaler.fit_transform(X)
```

In [8]:
```
D = squareform(pdist(X_minmax))
```

In [9]:
```
sse_list = list()
sil_list = list()
max_k = 30
for k in range(2,max_k + 1):
    kmeans = KMeans(n_clusters = k, n_init = 10, random_state = 0)
    kmeans.fit(X_minmax)
    sse_list.append(kmeans.inertia_)
    sil_list.append(silhouette_score(D, kmeans.labels_,metric='precomputed'))
```

In [10]:
```
plt.figure(figsize=(7,3))
fig, ax1 = plt.subplots()

ax1.plot(range(2, max_k + 1), sse_list, marker='.')
ax1.set_ylabel('SSE')

ax2 = ax1.twinx()
ax2.plot(range(2, max_k + 1), sil_list, marker='*', c='r')
ax2.set_ylabel('Silhouette')

plt.xlabel('k')
plt.xticks(range(2, max_k + 1))
plt.grid()
plt.title('Grafico di SSE e silhouette in funzione del numero di cluster')
plt.show()
```

```
<Figure size 700x300 with 0 Axes>
```




Per determinare il numero di cluster da impostare nell'algoritmo K-Means, cerchiamo un valore di k per cui aggiungendo un cluster la diminuzione della Sum of Squared Errors non sia significativa. Identifichiamo quindi il valore k=6, in corrispondenza del quale si ha anche un aumento del coefficiente di silhouette.

## K-means con k=6

In [11]:
```
kmeans = KMeans(n_clusters = 6, n_init = 100, max_iter = 300, random_state = 0)
```

In [12]:
```
kmeans.fit(X_minmax)
```

Out[12]:
```
KMeans(n_clusters=6, n_init=100, random_state=0)
```

In [13]:
```
print('SSE', kmeans.inertia_)
print('Silhouette', silhouette_score(X_minmax, kmeans.labels_))
```

```
SSE 525.33409528883
Silhouette 0.16411471733375485
```

## K-means con k=6 e sottoinsieme delle feature

L'algortimo k-means è stato eseguito con un sottoinsieme delle variabili numeriche. La riduzione del numero di feature permette di diminuire il valore di SSE e incrementare il coefficiente di silhouette, indicando la realizzazione di un clustering migliore. Le feature selezionate assumono valori differenti nei sei cluster ottenuti, come si nota nel parallel coordinates plot.

In [14]:
```
colonne_num = ['length_ms','intensity','mfcc_mean', 'stft_mean', 'kur','mfcc_max','mfcc_std']
```

In [15]:
```
df = df0[colonne_num]
```

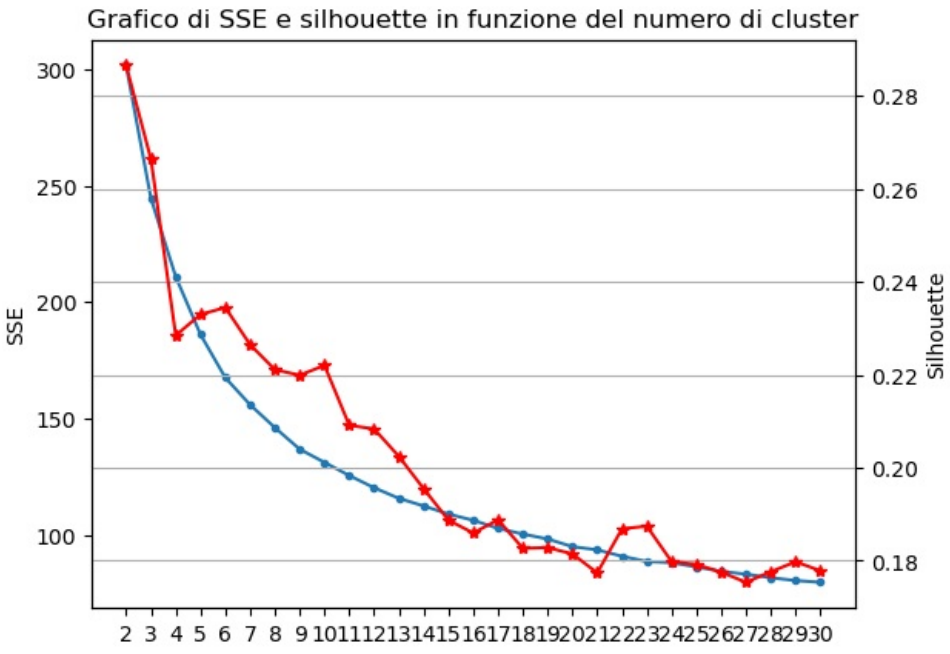
In [16]:
```
X = df.values
minscaler = MinMaxScaler()
X_minmax = minscaler.fit_transform(X)
```

In [17]:
```
D = squareform(pdist(X_minmax))
```

In [18]:
```
sse_list = list()
sil_list = list()
max_k = 30
for k in range(2,max_k + 1):
    kmeans = KMeans(n_clusters = k, n_init = 10, random_state = 0)
    kmeans.fit(X_minmax)
    sse_list.append(kmeans.inertia_)
    sil_list.append(silhouette_score(D, kmeans.labels_,metric='precomputed'))
```

In [19]:
```
plt.figure(figsize=(7,3))
fig, ax1 = plt.subplots()

ax1.plot(range(2, max_k + 1), sse_list, marker='.')
ax1.set_ylabel('SSE')

ax2 = ax1.twinx()
ax2.plot(range(2, max_k + 1), sil_list, marker='*', c='r')
ax2.set_ylabel('Silhouette')

plt.xlabel('k')
plt.xticks(range(2, max_k + 1))
plt.grid()
plt.title('Grafico di SSE e silhouette in funzione del numero di cluster')
plt.show()
```

```
<Figure size 700x300 with 0 Axes>
```



In [20]:
```
kmeans = KMeans(n_clusters = 6, n_init = 100, max_iter = 1000, random_state = 0)
```

In [21]:
```
kmeans.fit(X_minmax)
```

Out[21]:
```
KMeans(max_iter=1000, n_clusters=6, n_init=100, random_state=0)
```

In [22]:
```
df['kmeans_labels'] = kmeans.labels_
```

```
C:\Users\giorg\AppData\Local\Temp\ipykernel_32180\3286506214.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df['kmeans_labels'] = kmeans.labels_
```

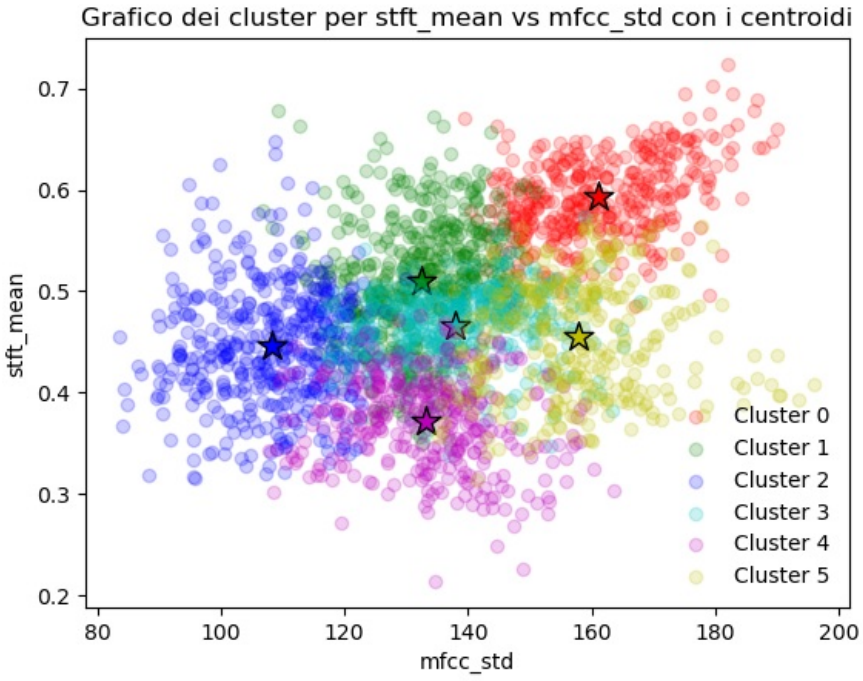
In [23]:
```
centers = kmeans.cluster_centers_
centers = minscaler.inverse_transform(centers)
```

In [24]:
```
dfc = pd.DataFrame(data = centers, columns=colonne_num)
dfc
```

Out[24]:

| | length_ms | intensity | mfcc_mean | stft_mean | kur | mfcc_max | mfcc_std |
|---|---|---|---|---|---|---|---|
| **0** | 3567.067183 | -47.137073 | -29.085647 | 0.592565 | 16.161347 | 226.873837 | 161.175377 |
| **1** | 3704.236791 | -35.529235 | -27.114533 | 0.509650 | 13.528483 | 207.023102 | 132.451719 |
| **2** | 4218.690832 | -26.268282 | -24.364330 | 0.446246 | 11.903542 | 179.473335 | 108.224632 |
| **3** | 4632.321429 | -38.017256 | -26.887183 | 0.465857 | 6.398225 | 213.151544 | 137.782136 |
| **4** | 4813.997207 | -36.287756 | -33.381296 | 0.371236 | 6.027901 | 166.976853 | 133.185009 |
| **5** | 3709.814925 | -46.414596 | -34.367784 | 0.454370 | 12.099731 | 200.896807 | 157.826548 |

In [25]:
```
x = 'mfcc_std'
y = 'stft_mean'
colors = ['r', 'g', 'b', 'c', 'm', 'y']

for l in sorted(df['kmeans_labels'].unique()):
    plt.scatter(df[df['kmeans_labels'] == l][x], df[df['kmeans_labels'] == l][y], label='Cluster ' + str(l), al
    plt.scatter(dfc.iloc[l][x], dfc.iloc[l][y], marker='*', s=200, c=colors[l], edgecolors='k')

plt.legend(frameon=False)
plt.title('Grafico dei cluster per stft_mean vs mfcc_std con i centroidi')
plt.xlabel(x)
plt.ylabel(y)
plt.show()
```

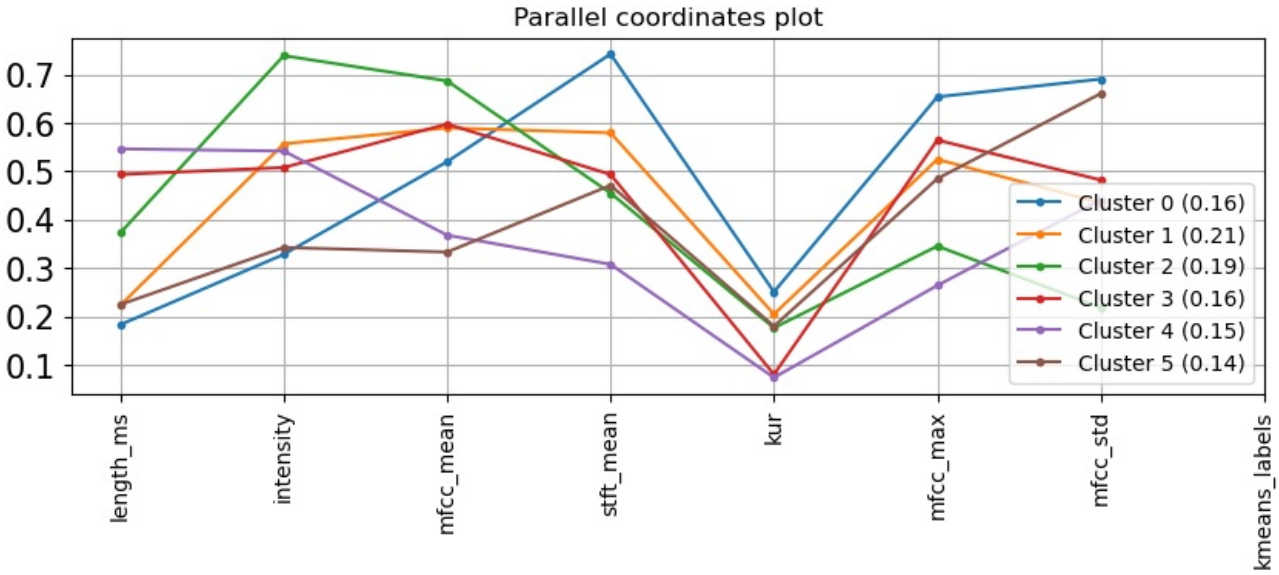



I cluster ottenuti hanno una dimensione simile tra loro e dal grafico è possibile vederne la distribuzione in uno spazio bidimensionale.

In [26]:
```
labels, sizes = np.unique(kmeans.labels_, return_counts = True)
for l,s in zip(labels, sizes):
    print('Cluster %s, size %s (%.2f)' % (l,s, s/len(X)))
```

```
Cluster 0, size 387 (0.16)
Cluster 1, size 511 (0.21)
Cluster 2, size 469 (0.19)
Cluster 3, size 392 (0.16)
Cluster 4, size 358 (0.15)
Cluster 5, size 335 (0.14)
```

In [27]:
```
plt.figure(figsize=(10,3))
for l in np.unique(kmeans.labels_):
    plt.plot(dfc.columns, kmeans.cluster_centers_[l], marker='.', label='Cluster %s (%.2f)' % (l, sizes[l]/len(X
plt.xticks(range(0,len(df.columns)), df.columns, rotation=90, fontsize=10)
plt.yticks(fontsize=15)
plt.title('Parallel coordinates plot')
plt.legend()
plt.grid()
plt.show()
```




In [28]:
```
print('SSE', kmeans.inertia_)
print('Silhouette', silhouette_score(X_minmax, kmeans.labels_))
```

```
SSE 167.5064165698241
Silhouette 0.2344437275407549
```

Il valore di SSE è notevolemnte diminuito limitando il numero di feature e il coefficiente di silhouette è superiore a 0.23.

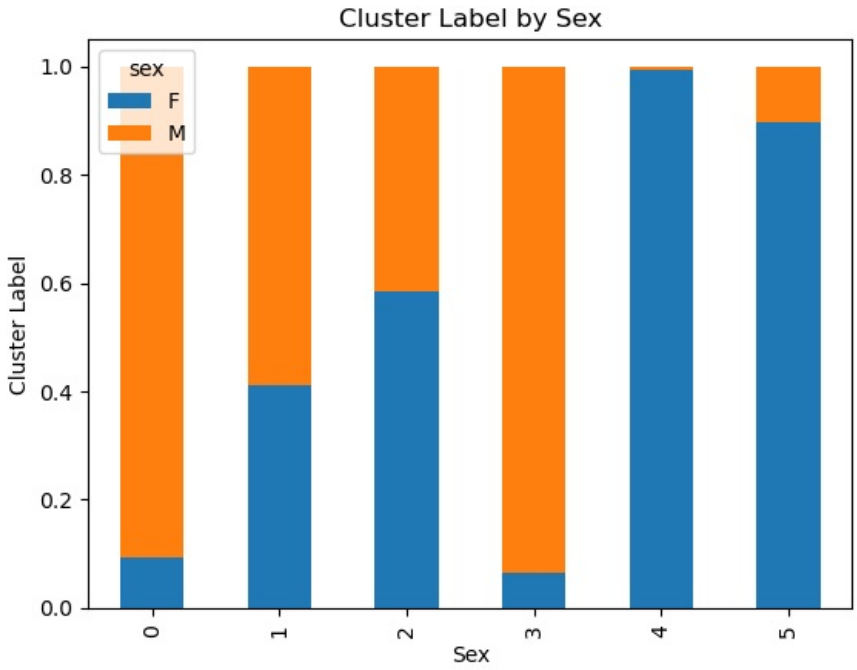
In [29]:
```
df_clusters = df.copy()
df_clusters['Labels'] = kmeans.labels_
```

In [30]:
```
p = pd.crosstab(df_clusters['Labels'],df0['sex']); p
```

Out[30]:

| sex | F | M |
|---|---|---|
| Labels | | |
| 0 | 36 | 351 |
| 1 | 211 | 300 |
| 2 | 274 | 195 |
| 3 | 26 | 366 |
| 4 | 356 | 2 |
| 5 | 301 | 34 |

In [31]:
```
pct_sex = p.div(p.sum(1).astype(float), axis=0)
pct_sex.plot(kind='bar', stacked=True, title='Cluster Label by Sex')
plt.xlabel('Sex')
plt.ylabel('Cluster Label')
plt.show()
```



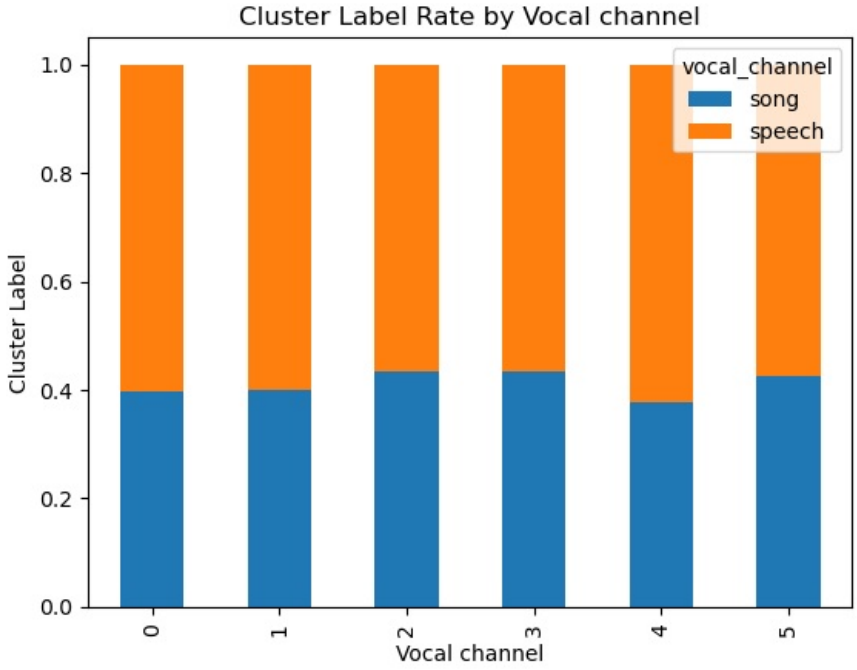

Le feature selezionate riescono a distinguere in modo piuttosto significativo il genere degli attori, infatti i cluster 4 e 5 sono costituiti prevalentemente da record relativi ad attrici mentre i cluter 0 e 3 sono a maggioranza di attori uomini.

In [32]:
```
p1 = pd.crosstab(df_clusters['Labels'],df0['vocal_channel']);p1
```

Out[32]:

| vocal_channel | song | speech |
|---|---|---|
| Labels | | |
| 0 | 154 | 233 |
| 1 | 204 | 307 |
| 2 | 203 | 266 |
| 3 | 170 | 222 |
| 4 | 135 | 223 |
| 5 | 143 | 192 |

In [33]:
```
pct_vocal = p1.div(p.sum(1).astype(float), axis=0)
pct_vocal.plot(kind='bar', stacked=True, title='Cluster Label Rate by Vocal channel')
plt.xlabel('Vocal channel')
plt.ylabel('Cluster Label')
plt.show()
```



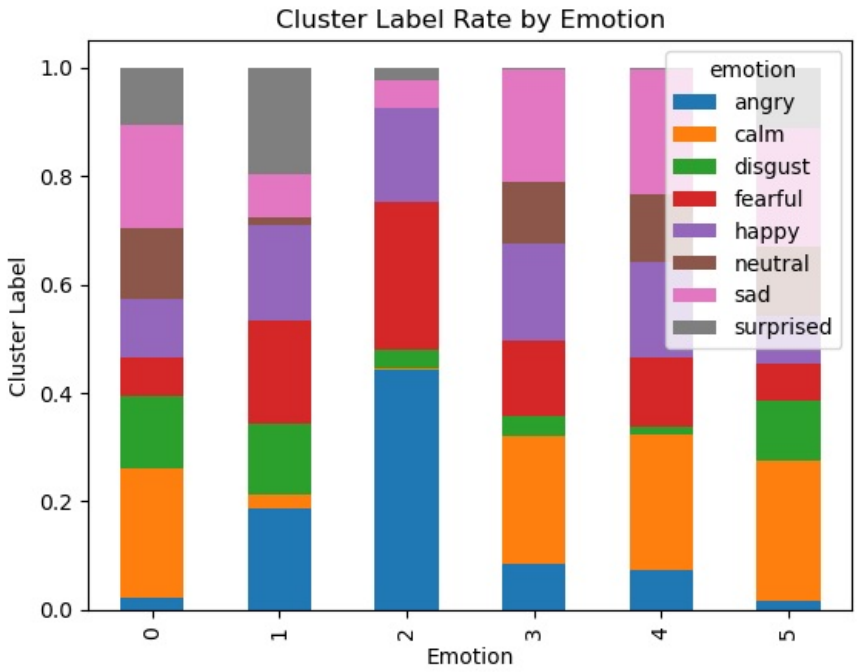

Nel caso di 'vocal channel', le feature selezionate non caratterizzano i cluster in modo differente.

In [34]:
```
p2 = pd.crosstab(df_clusters['Labels'],df0['emotion']);p2
```

Out[34]:

| emotion | angry | calm | disgust | fearful | happy | neutral | sad | surprised |
|---|---|---|---|---|---|---|---|---|
| Labels | | | | | | | | |
| **0** | 9 | 92 | 52 | 27 | 42 | 51 | 73 | 41 |
| **1** | 95 | 13 | 68 | 97 | 90 | 7 | 40 | 101 |
| **2** | 207 | 2 | 16 | 128 | 81 | 0 | 24 | 11 |
| **3** | 33 | 93 | 14 | 55 | 70 | 44 | 82 | 1 |
| **4** | 26 | 90 | 5 | 46 | 63 | 44 | 83 | 1 |
| **5** | 6 | 86 | 37 | 23 | 30 | 42 | 74 | 37 |

In [35]:
```
pct_emotion = p2.div(p.sum(1).astype(float), axis=0)
pct_emotion.plot(kind='bar', stacked=True, title='Cluster Label Rate by Emotion')
plt.xlabel('Emotion')
plt.ylabel('Cluster Label')
plt.show()
```



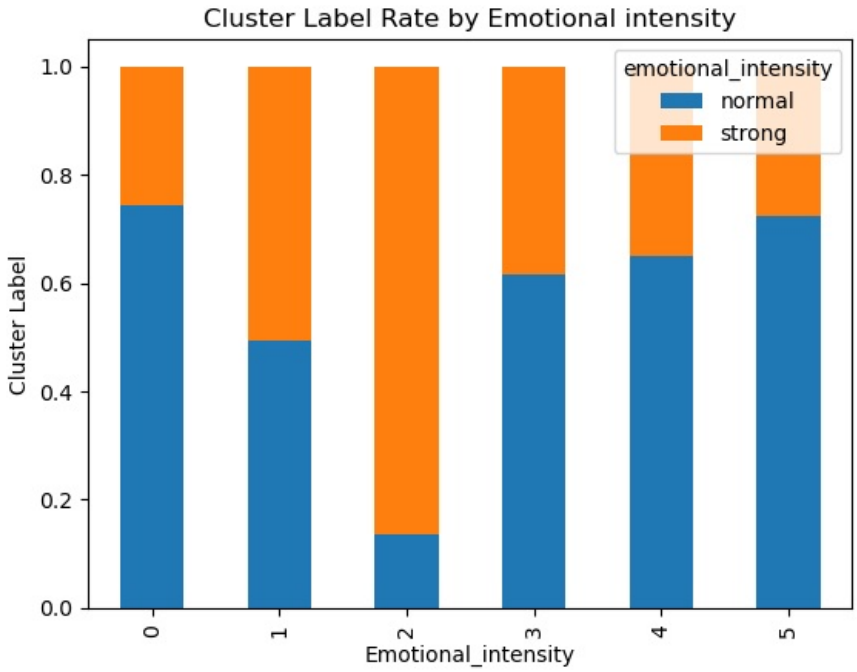

Dal grafico si osserva come le emozioni siano presenti nei sei cluster in proporzioni differenti. Ad esempio, 'surprised' è quasi del tutto assente nei cluster 3 e 4. L'emozione 'angry' si trova principalmente nei cluster 1 e 2, in corrispondenza di un numero ridotto di record relativi all'emozione 'calm'.

In [36]:
```
p3 = pd.crosstab(df_clusters['Labels'],df0['emotional_intensity']); p3
```

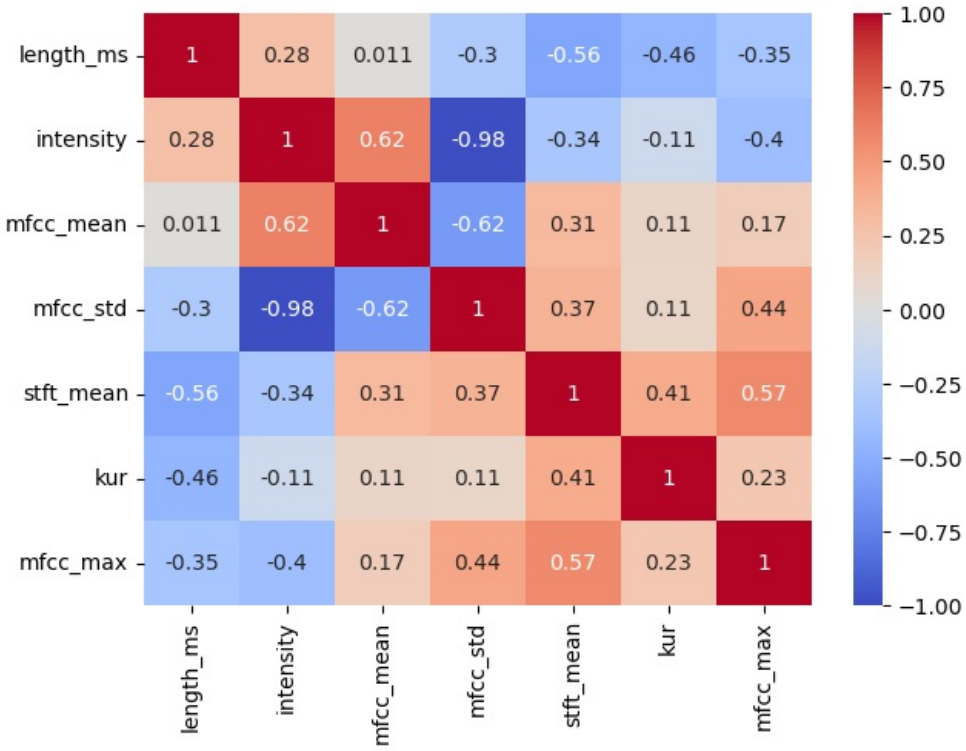
Out[36]:

| emotional_intensity | normal | strong |
|---|---|---|
| Labels | | |
| **0** | 288 | 99 |
| **1** | 252 | 259 |
| **2** | 64 | 405 |
| **3** | 241 | 151 |
| **4** | 233 | 125 |
| **5** | 242 | 93 |

In [37]:
```
pct_emotion = p3.div(p.sum(1).astype(float), axis=0)
pct_emotion.plot(kind='bar', stacked=True, title='Cluster Label Rate by Emotional intensity')
plt.xlabel('Emotional_intensity')
plt.ylabel('Cluster Label')
plt.show()
```




Infine, i cluster 0, 3, 4 e 5 mostrano una maggioranza di record relativi ad 'emotional intensity' 'normal', mentre il cluster 2 ha una predominanza di righe con 'emotional intensity' 'strong'.




In [18]:
```
X_hier = df3
```

In [19]:
```
scaler = MinMaxScaler(feature_range=(0,1))
```

In [20]:
```
scaler.fit(X_hier)
```

Out[20]:
```
▾ MinMaxScaler
MinMaxScaler()
```

In [21]:
```
X1_hier =scaler.transform(X_hier)
X1_hier
```

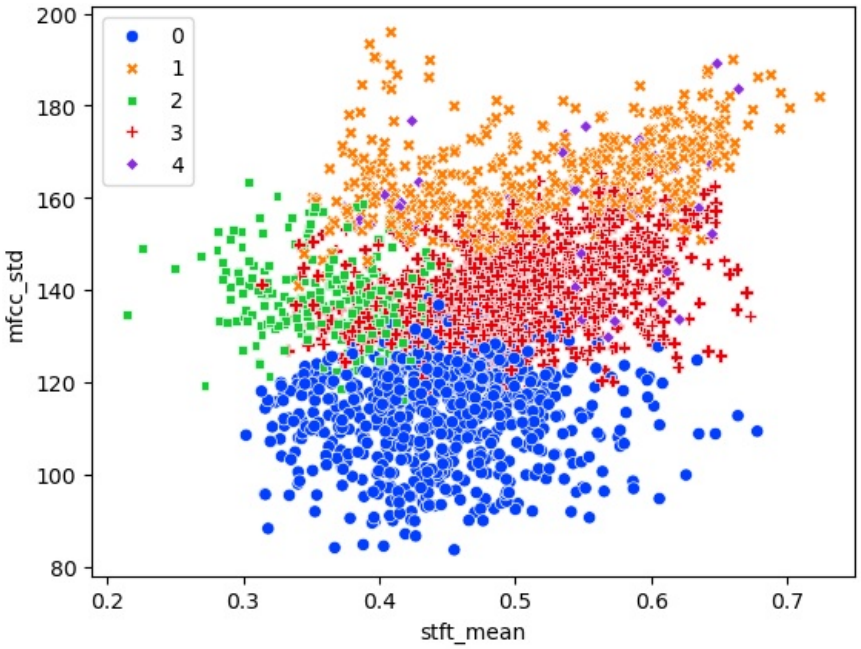
Out[21]:
```
array([[0.23305208, 0.5505861 , 0.36463414, ..., 0.39444307, 0.13341265,
        0.29521347],
       [0.28164097, 0.59628963, 0.50529911, ..., 0.39883034, 0.31060606,
        0.51166457],
       [0.5048007 , 0.64216397, 0.46891841, ..., 0.32484738, 0.05448389,
        0.25471013],
       ...,
       [0.6700611 , 0.52105507, 0.62417809, ..., 0.49786435, 0.05593306,
        0.71002156],
       [0.23305208, 0.36839759, 0.54976335, ..., 0.71524277, 0.21102804,
        0.6080492 ],
       [0.26214722, 0.48500247, 0.52234878, ..., 0.66836228, 0.19563576,
        0.60599817]])
```

In [22]:
```
hier_agg = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='complete')
hier_agg.fit(X1_hier)
```

```
C:\Users\dimit\anaconda3\envs\env_master\Lib\site-packages\sklearn\cluster\_agglomerative.py:1006: FutureWarning
: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Out[22]:
```
AgglomerativeClustering(affinity='euclidean', linkage='complete', n_clusters=5)
```

In [23]:
```
sns.scatterplot(data=df, x="stft_mean", y="mfcc_std", hue=hier_agg.labels_,
                style=hier_agg.labels_, palette="bright")
plt.show()
```




In [24]:
```
silhouette_score(X1_hier, hier_agg.labels_)
```

Out[24]:
```
0.19305791223185037
```

la silhouette sembra essere buona con 5 cluster, ma sembra esserci un cluster di popolazione molto bassa

In [25]:
```
hier_agg2 = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='complete')
hier_agg2.fit(X1_hier)
```

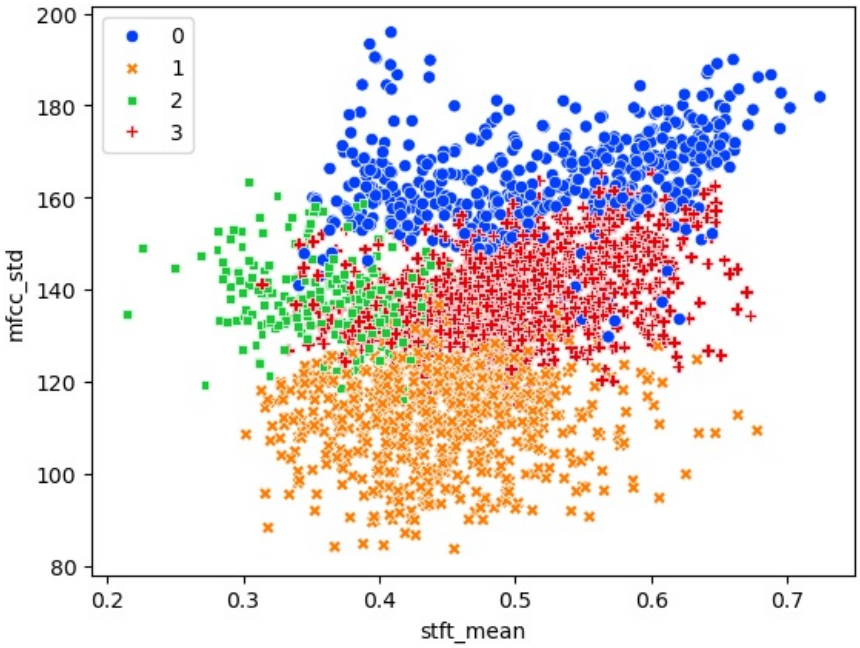
```
C:\Users\dimit\anaconda3\envs\env_master\Lib\site-packages\sklearn\cluster\_agglomerative.py:1006: FutureWarning
: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Out[25]:
```
AgglomerativeClustering(affinity='euclidean', linkage='complete', n_clusters=4)
```

In [26]:
```
silhouette_score(X1_hier, hier_agg2.labels_)
```

Out[26]:
```
0.1922548462689763
```

In [27]:
```
sns.scatterplot(data=df, x="stft_mean", y="mfcc_std", hue=hier_agg2.labels_,
                style=hier_agg2.labels_, palette="bright")
plt.show()
```




riprovo con 4 cluster, la silhouette rimane praticamente uguale però la proporzione tra i cluster è migliore

In [28]:
```
labels, sizes = np.unique(hier_agg.labels_, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster0, size 733 (29.893964110929854)%
Cluster1, size 459 (18.71941272430669)%
Cluster2, size 259 (10.562805872756934)%
Cluster3, size 958 (39.07014681892333)%
Cluster4, size 43 (1.7536704730831976)%
```

In [29]:
```
labels, sizes = np.unique(hier_agg2.labels_, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster0, size 502 (20.473083197389887)%
Cluster1, size 733 (29.893964110929854)%
Cluster2, size 259 (10.562805872756934)%
Cluster3, size 958 (39.07014681892333)%
```

si conferma quanto visto ad occhio sulle popolazioni dei cluster

In [84]:
```
hier_agg3 = AgglomerativeClustering(n_clusters=7, affinity='euclidean', linkage='complete')
hier_agg3.fit(X1_hier)
```

```
C:\Users\dimit\anaconda3\envs\env_master\Lib\site-packages\sklearn\cluster\_agglomerative.py:1006: FutureWarning
: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Out[84]:
```
AgglomerativeClustering(affinity='euclidean', linkage='complete', n_clusters=7)
```

In [85]:
```
silhouette_score(X1_hier, hier_agg3.labels_)
```

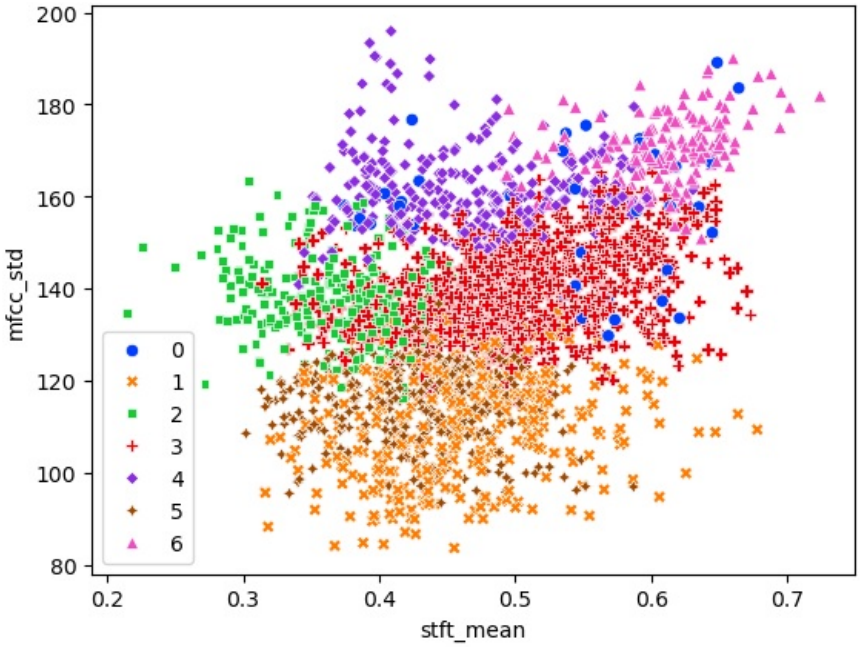
Out[85]:
```
0.1442506324988574
```

In [87]:
```
labels, sizes = np.unique(hier_agg3.labels_, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster0, size 43 (1.7536704730831976)%
Cluster1, size 372 (15.171288743882544)%
Cluster2, size 259 (10.562805872756934)%
Cluster3, size 958 (39.07014681892333)%
Cluster4, size 291 (11.867862969004895)%
Cluster5, size 361 (14.722675367047309)%
Cluster6, size 168 (6.851549755301795)%
```

In [86]:
```
sns.scatterplot(data=df, x="stft_mean", y="mfcc_std", hue=hier_agg3.labels_,
```

```
                style=hier_agg3.labels_, palette="bright")
plt.show()
```




faccio ultimo tentativo con 7 cluster però la silhouette si abbassa sensibilmente e sembrano essere cluster più sparpagliati

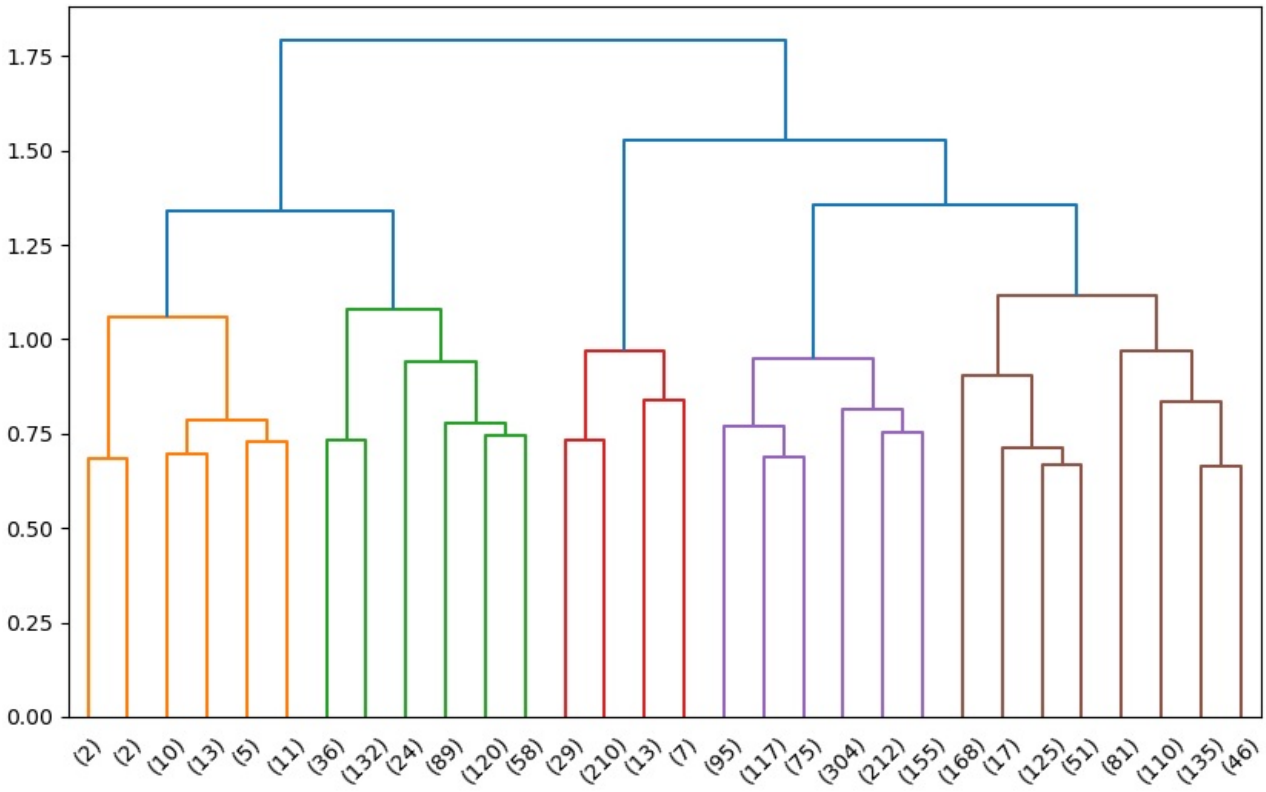
In [34]:
```
D = squareform(pdist(X1_hier))
D
```

Out[34]:
```
array([[0.        , 0.32230724, 0.33315884, ..., 0.67002674, 0.57005101,
        0.47144019],
       [0.32230724, 0.        , 0.43764915, ..., 0.53787544, 0.49434516,
        0.3681678 ],
       [0.33315884, 0.43764915, 0.        , ..., 0.56028076, 0.73847091,
        0.62313079],
       ...,
       [0.67002674, 0.53787544, 0.56028076, ..., 0.        , 0.58243402,
        0.4971502 ],
       [0.57005101, 0.49434516, 0.73847091, ..., 0.58243402, 0.        ,
        0.16371065],
       [0.47144019, 0.3681678 , 0.62313079, ..., 0.4971502 , 0.16371065,
        0.        ]])
```

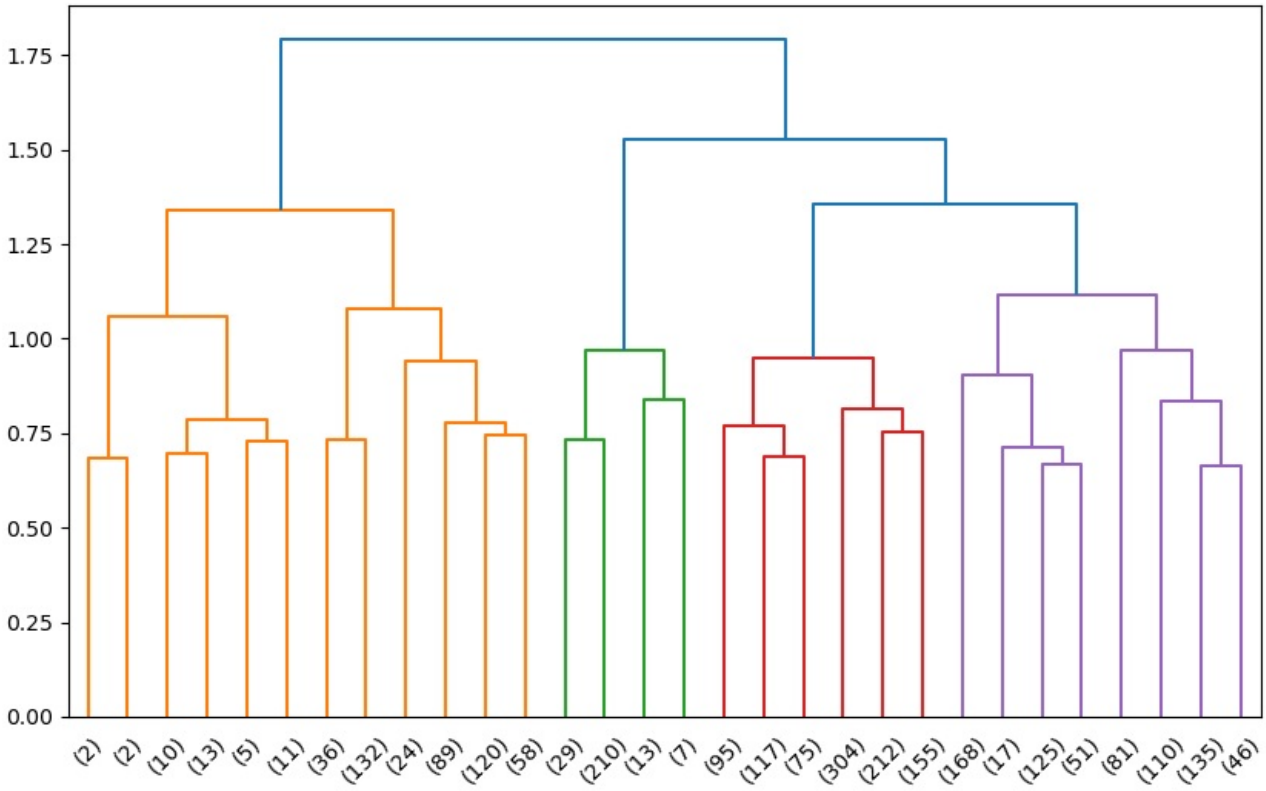
In [35]:
```
Ds = pdist(X1_hier)
```

In [36]:
```
links_complete = linkage(Ds, method='complete')
```

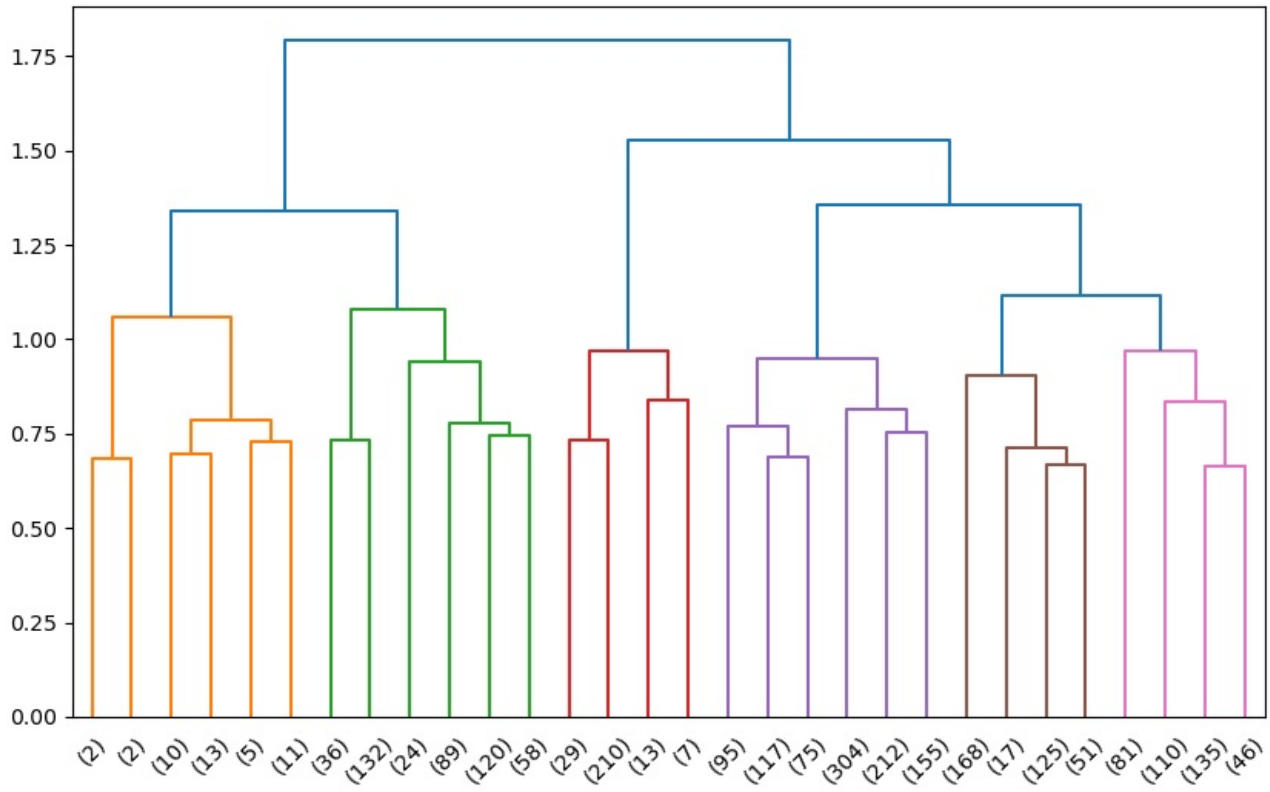
In [59]:
```
plt.figure(figsize=(10,6))
res = dendrogram(links_complete,truncate_mode='lastp',color_threshold=1.13)
```




In [63]:
```
plt.figure(figsize=(10,6))
res = dendrogram(links_complete,truncate_mode='lastp',color_threshold=1.35)#sceglie cosa non far vedere, ed è p
```




In [83]:
```
plt.figure(figsize=(10,6))
res2 = dendrogram(links_complete,truncate_mode='lastp',color_threshold=1.1)#sceglie cosa non far vedere, ed è p
```




provo anche con 6 cluster per vedere se sono presenti differenze

In [46]:
```
labels_res1= fcluster(links_complete, t=1.13, criterion='distance')
labels_res1
```

Out[46]:
```
array([4, 4, 5, ..., 4, 2, 4], dtype=int32)
```

In [64]:
```
labels_res3= fcluster(links_complete, t=1.35, criterion='distance')
labels_res3
```

Out[64]:
```
array([3, 3, 4, ..., 3, 1, 3], dtype=int32)
```

In [52]:
```
labels_res2= fcluster(links_complete, t=1.1, criterion='distance')
labels_res2
```

Out[52]:
```
array([4, 4, 5, ..., 4, 2, 4], dtype=int32)
```

In [65]:
```
labels, sizes = np.unique(labels_res3, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster1, size 502 (20.473083197389887)%
Cluster2, size 259 (10.562805872756934)%
Cluster3, size 958 (39.07014681892333)%
Cluster4, size 733 (29.893964110929854)%
```

In [53]:
```
labels, sizes = np.unique(labels_res1, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster1, size 43 (1.7536704730831976)%
Cluster2, size 459 (18.71941272430669)%
Cluster3, size 259 (10.562805872756934)%
Cluster4, size 958 (39.07014681892333)%
Cluster5, size 733 (29.893964110929854)%
```

In [54]:
```
labels, sizes = np.unique(labels_res2, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster1, size 43 (1.7536704730831976)%
Cluster2, size 459 (18.71941272430669)%
Cluster3, size 259 (10.562805872756934)%
Cluster4, size 958 (39.07014681892333)%
Cluster5, size 361 (14.722675367047309)%
Cluster6, size 372 (15.171288743882544)%
```

In [55]:
```
silhouette_score(X1_hier, labels_res1)
```

Out[55]: 0.19305791223185037

In [56]:
```
silhouette_score(X1_hier, labels_res2)
```

Out[56]: 0.1519533905404676

anche con 6 cluster la silhouette si abbassa e popolazione sbilanciata

In [67]:
```
silhouette_score(X1_hier, labels_res3)
```

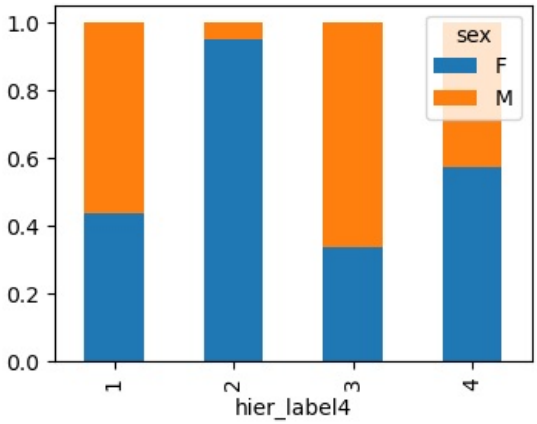
Out[67]: 0.1922548462689763

con 5 cluster la silhouette risulta essere migliore che con 4, tuttavia si crea un cluster piccolo: 1,75%, anche se a livello grafico non parrebbe, con 6 più proporzione nei cluster rispetto al 5 però silhouette più bassa. faccio analisi rispetto variabili categoriche solo per n.cluster = 4 e n.cluster = 5

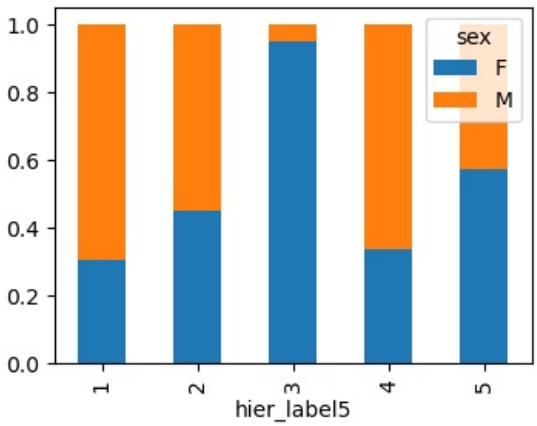
In [88]:
```
df['hier_label4'] = labels_res3
```

In [89]:
```
df['hier_label5'] = labels_res1
```

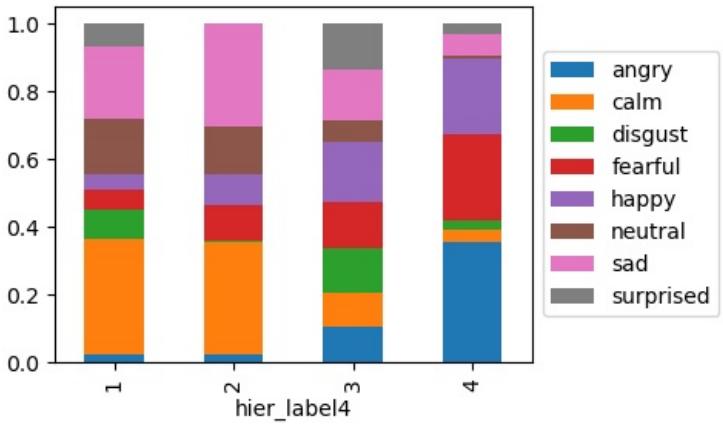
In [91]:
```
ct=pd.crosstab(df['hier_label4'], df['sex'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.show()
```



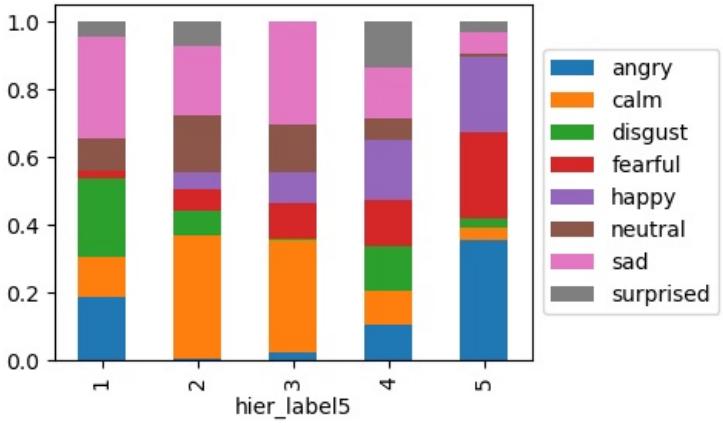

In [92]:
```
ct=pd.crosstab(df['hier_label5'], df['sex'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.show()
```



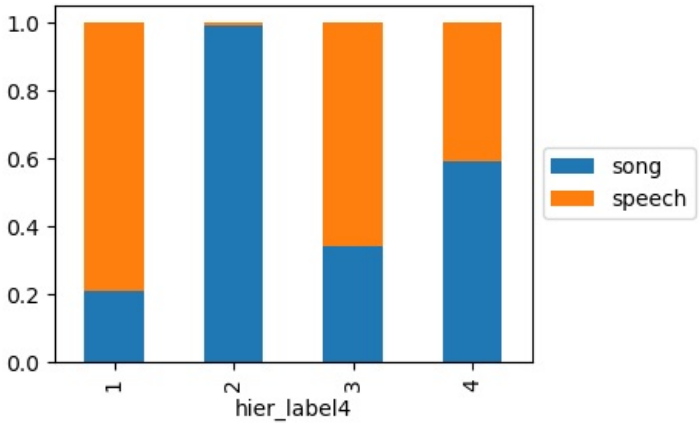

In [94]:
```
ct=pd.crosstab(df['hier_label4'], df['emotion'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



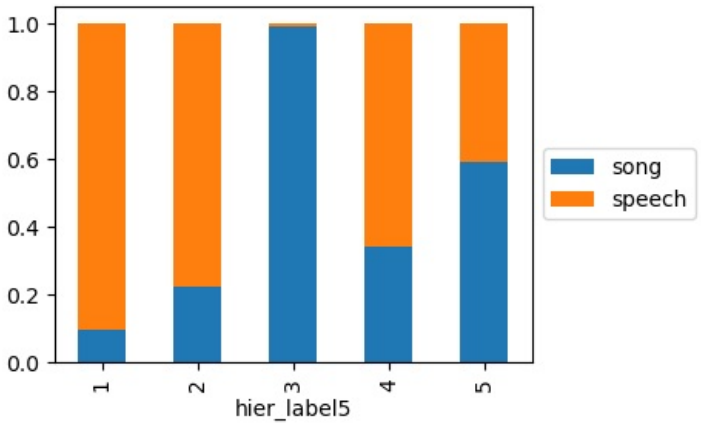

In [95]:
```
ct=pd.crosstab(df['hier_label5'], df['emotion'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```




In [96]:
```
ct=pd.crosstab(df['hier_label4'], df['vocal_channel'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```




In [97]:
```
ct=pd.crosstab(df['hier_label5'], df['vocal_channel'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```




dopo aver effettuato un confronto con 4 e 5 cluster mi ritengo soddisfatto dal cluster con k=4, sembra dividere per vocal channel a differenza del k-means

inoltre anche per le emozioni ad eccezion fatta del cluster 3 si vedono differenze. cluster 1 e 2 maggiromente sad e calm, cluster 4 happy fearful angry

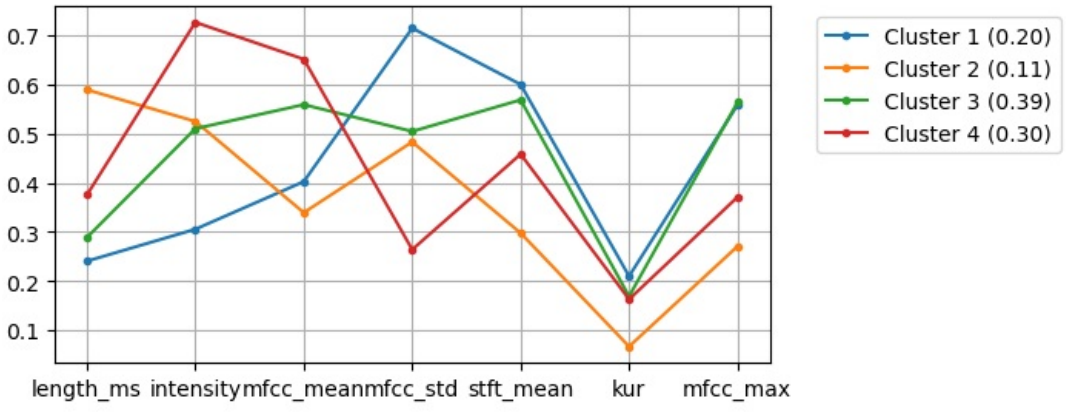
silhouette gerarchico (0.192) inferiore al k-means (0.234)

In [100…
```
hier_centers = list()
for l in sorted(np.unique(labels_res3)):
    if l == -1:
        continue
    hier_centers.append(np.mean(X1_hier[labels_res3 == l], axis=0))
hier_centers = np.array(hier_centers)
```

In [101…
```
hier_centers
```

Out[101…
```
array([[0.24050264, 0.30546208, 0.40230042, 0.71427618, 0.59983895,
        0.2098164 , 0.55815954],
       [0.58911482, 0.52464449, 0.33918135, 0.48297222, 0.29764403,
        0.06694237, 0.27067756],
       [0.28867938, 0.50994295, 0.55878607, 0.50437322, 0.56841735,
        0.16894678, 0.56341768],
       [0.37509591, 0.72615994, 0.65119859, 0.26406035, 0.45806974,
        0.16298599, 0.37073709]])
```

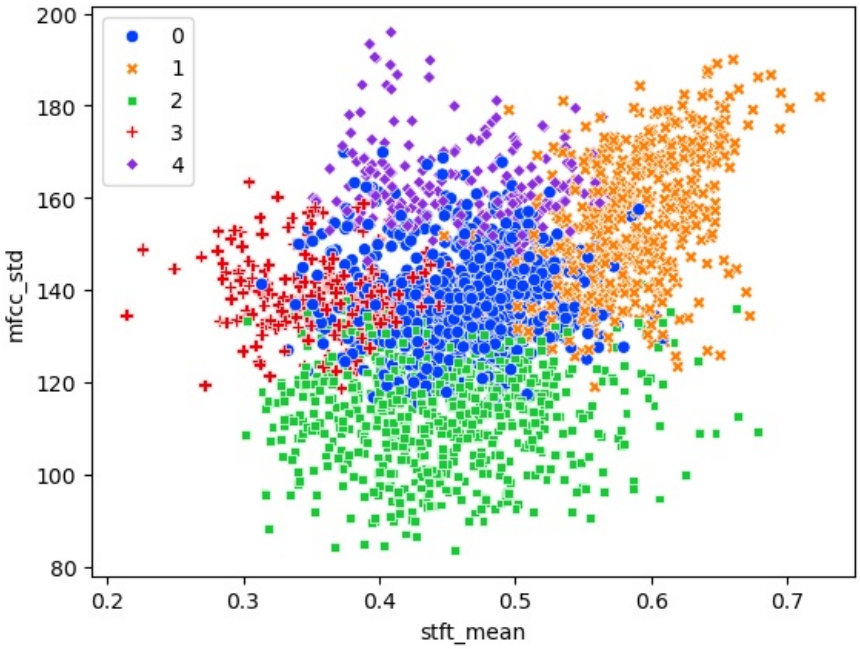
In [107…
```
labels, sizes = np.unique(labels_res3, return_counts=True)
plt.figure(figsize=(6,3))
for l in np.unique(labels_res3):
    if l == -1:
        continue
    plt.plot(df2.columns, hier_centers[l-1], marker='.', label='Cluster %s (%.2f)' % (l, sizes[l-1]/len(X1_hier

plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```




In [ ]:

provo ora a fare con linkage = ward

In [23]:
```
sns.scatterplot(data=df, x="stft_mean", y="mfcc_std", hue=hier_agg.labels_,
                style=hier_agg.labels_, palette="bright")
plt.show()
```




In [24]:
```
silhouette_score(X1_hier, hier_agg.labels_)
```

Out[24]:
```
0.18112735270685082
```

i cluster sembrano ben separati e anche la silhouette è buona, leggermente inferiore a complete

In [25]:
```
hier_agg2 = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
hier_agg2.fit(X1_hier)
```

```
C:\Users\dimit\anaconda3\envs\env_master\Lib\site-packages\sklearn\cluster\_agglomerative.py:1006: FutureWarning
: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Out[25]:
```
AgglomerativeClustering
AgglomerativeClustering(affinity='euclidean', n_clusters=4)
```

In [26]:
```
silhouette_score(X1_hier, hier_agg2.labels_)
```

Out[26]:
```
0.16768657155496042
```

sns.scatterplot(data=df, x="stft_mean", y="mfcc_std", hue=hier_agg2.labels_, style=hier_agg2.labels_, palette="bright") plt.show()

con n-cluster = 4 cala la silhouette

In [28]:
```
labels, sizes = np.unique(hier_agg.labels_, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster0, size 729 (29.73083197389886)%
Cluster1, size 522 (21.28874388254486)%
Cluster2, size 751 (30.62805872756933)%
Cluster3, size 225 (9.176182707993474)%
Cluster4, size 225 (9.176182707993474)%
```

In [29]:
```
labels, sizes = np.unique(hier_agg2.labels_, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster0, size 747 (30.464926590538337)%
Cluster1, size 729 (29.73083197389886)%
Cluster2, size 751 (30.62805872756933)%
Cluster3, size 225 (9.176182707993474)%
```

In [30]:
```
hier_agg3 = AgglomerativeClustering(n_clusters=7, affinity='euclidean', linkage='average')
hier_agg3.fit(X1_hier)
```

```
C:\Users\dimit\anaconda3\envs\env_master\Lib\site-packages\sklearn\cluster\_agglomerative.py:1006: FutureWarning
: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Out[30]:
```
AgglomerativeClustering(affinity='euclidean', linkage='average', n_clusters=7)
```

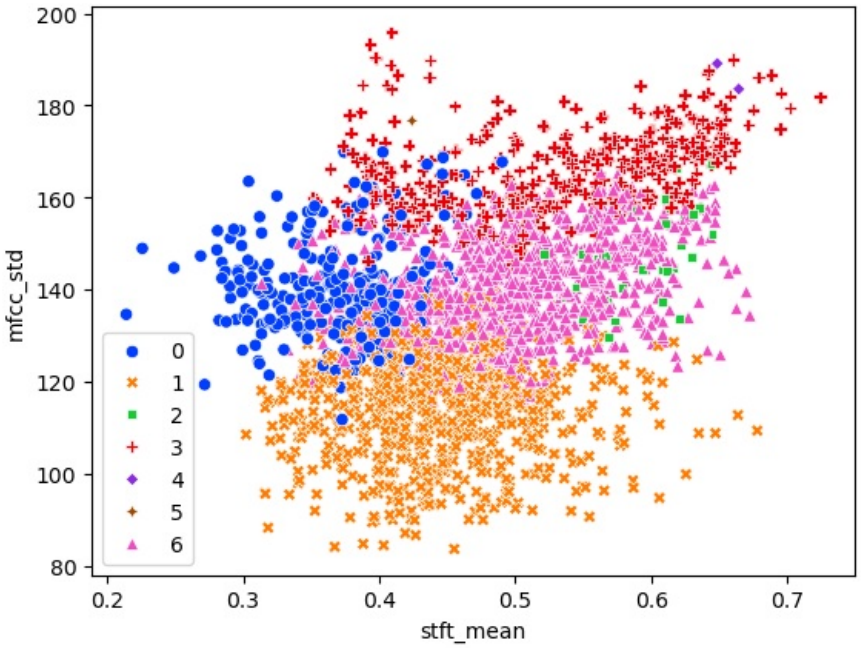
In [31]:
```
silhouette_score(X1_hier, hier_agg3.labels_)
```

Out[31]:
```
0.1960942738420164
```

In [32]:
```
labels, sizes = np.unique(hier_agg3.labels_, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster0, size 278 (11.33768352365416)%
Cluster1, size 736 (30.0163132137031)%
Cluster2, size 35 (1.4274061990212072)%
Cluster3, size 403 (16.43556280587276)%
Cluster4, size 2 (0.08156606851549755)%
Cluster5, size 2 (0.08156606851549755)%
Cluster6, size 996 (40.61990212071778)%
```

In [33]:
```
sns.scatterplot(data=df, x="stft_mean", y="mfcc_std", hue=hier_agg3.labels_,
                style=hier_agg3.labels_, palette="bright")
plt.show()
```




con n.cluster = 7 silhouette alta ma ho 2 cluster che rappresentano 2 valori ciascuno

In [34]:
```
D = squareform(pdist(X1_hier)) #calcolo la matrice delle distanze per le variabile numeriche selezione normaliz
D
```

Out[34]:
```
array([[0.        , 0.32230724, 0.33315884, ..., 0.67002674, 0.57005101,
        0.47144019],
       [0.32230724, 0.        , 0.43764915, ..., 0.53787544, 0.49434516,
        0.3681678 ],
       [0.33315884, 0.43764915, 0.        , ..., 0.56028076, 0.73847091,
        0.62313079],
       ...,
       [0.67002674, 0.53787544, 0.56028076, ..., 0.        , 0.58243402,
        0.4971502 ],
       [0.57005101, 0.49434516, 0.73847091, ..., 0.58243402, 0.        ,
        0.16371065],
       [0.47144019, 0.3681678 , 0.62313079, ..., 0.4971502 , 0.16371065,
        0.        ]])
```

In [35]:
```
Ds = pdist(X1_hier)#in questo modo viene come un vettore solo, tutto su una riga, evita ripetizione di informaz
```

In [36]:
```
links_complete = linkage(Ds, method='ward')#matrice dei numeri da passare poi al dendo
```

In [41]:
```
labels_res3= fcluster(links_complete, t=7, criterion='distance')
labels_res3
```

Out[41]:
```
array([5, 5, 3, ..., 5, 2, 2], dtype=int32)
```

In [42]:
```
labels_res2= fcluster(links_complete, t=8, criterion='distance')
labels_res2
```

Out[42]:
```
array([4, 4, 2, ..., 4, 1, 1], dtype=int32)
```

In [43]:
```
labels, sizes = np.unique(labels_res3, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster1, size 225 (9.176182707993474)%
Cluster2, size 522 (21.28874388254486)%
Cluster3, size 751 (30.62805872756933)%
Cluster4, size 225 (9.176182707993474)%
Cluster5, size 729 (29.73083197389886)%
```

In [45]:
```
labels, sizes = np.unique(labels_res2, return_counts =True)
for l, s in zip(labels,sizes):
    perc = s/len(X1_hier)*100
    print(f'Cluster{l}, size {s} ({perc})%')
```

```
Cluster1, size 747 (30.464926590538337)%
Cluster2, size 751 (30.62805872756933)%
Cluster3, size 225 (9.176182707993474)%
Cluster4, size 729 (29.73083197389886)%
```

In [47]:
```
silhouette_score(X1_hier, labels_res2)
```

Out[47]:
```
0.16768657155496042
```

In [48]:
```
silhouette_score(X1_hier, labels_res3)
```

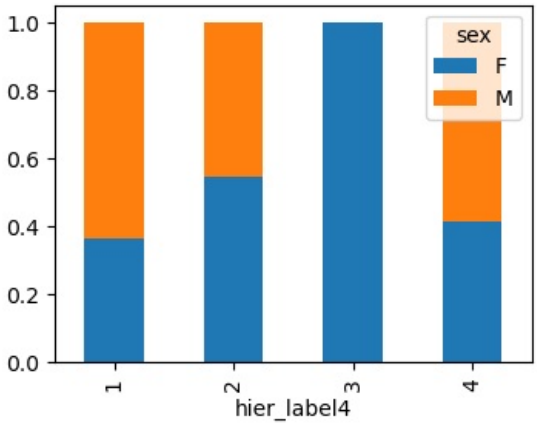
Out[48]:
```
0.18112735270685082
```

silhouette leggermente più bassa rispetto a metodo complete, con ward preferisco n.cluster =5 al momento (silhouette 0.181 vs 0.167)

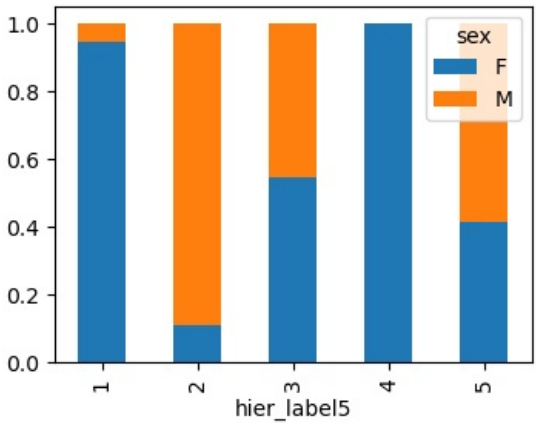
In [49]:
```
df['hier_label5'] = labels_res3
```

In [50]:
```
df['hier_label4'] = labels_res2
```

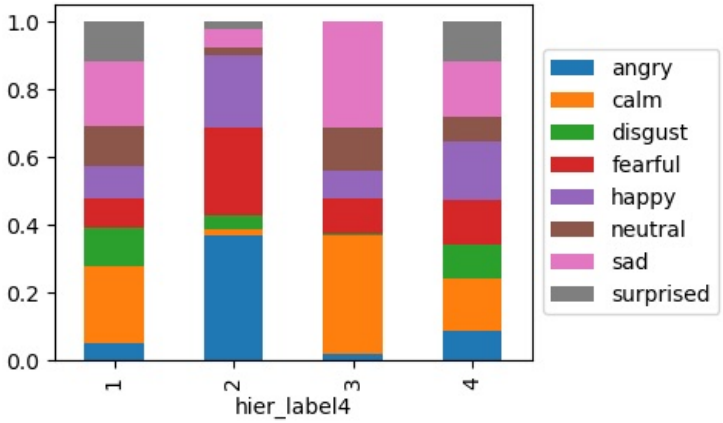
In [51]:
```
ct=pd.crosstab(df['hier_label4'], df['sex'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.show()
```



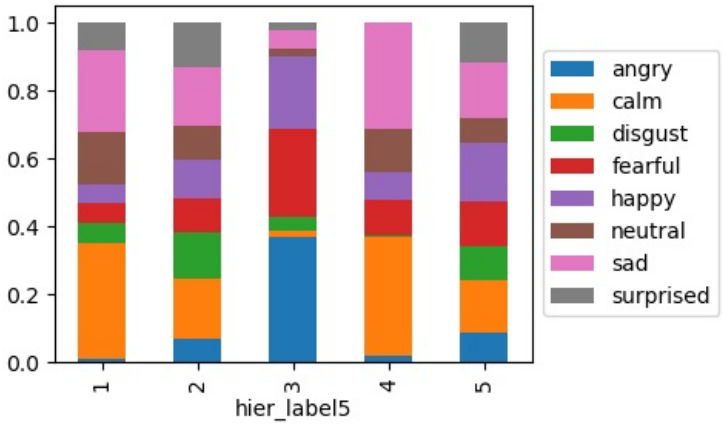

In [52]:
```
ct=pd.crosstab(df['hier_label5'], df['sex'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.show()
```



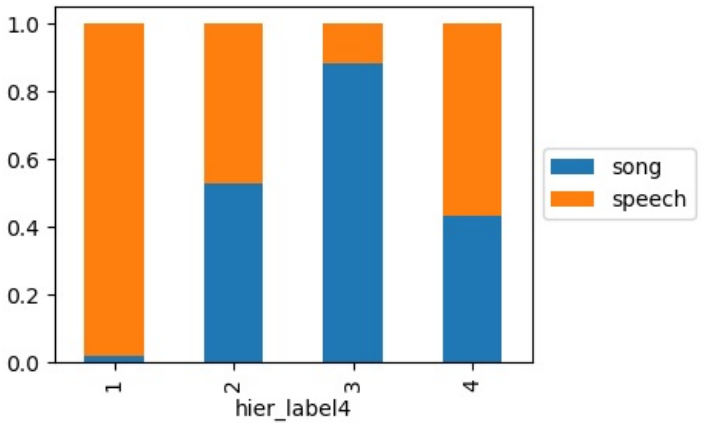

In [53]:
```
ct=pd.crosstab(df['hier_label4'], df['emotion'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



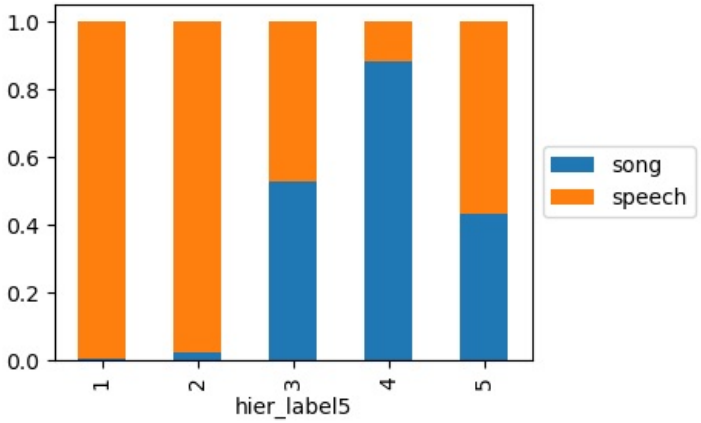

In [54]:
```
ct=pd.crosstab(df['hier_label5'], df['emotion'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```




In [55]:
```
ct=pd.crosstab(df['hier_label4'], df['vocal_channel'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```




In [56]:
```
ct=pd.crosstab(df['hier_label5'], df['vocal_channel'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

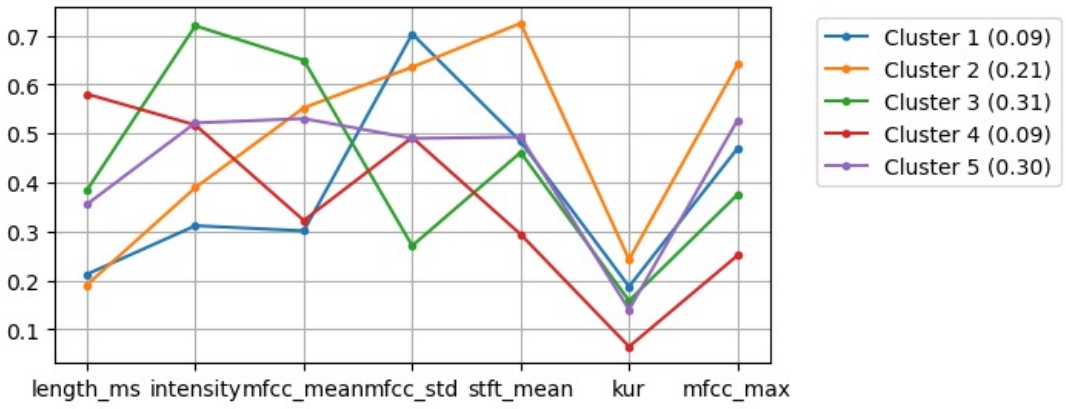



In [57]:
```
hier_centers = list()
for l in sorted(np.unique(labels_res3)):
    if l == -1:
        continue
    hier_centers.append(np.mean(X1_hier[labels_res3 == l], axis=0))
hier_centers = np.array(hier_centers)
```

In [58]:
```
hier_centers
```

Out[58]:
```
array([[0.21216177, 0.31136855, 0.30091595, 0.7029754 , 0.48332849,
        0.18710099, 0.46909751],
       [0.19049068, 0.38918924, 0.55215069, 0.63523015, 0.72481514,
        0.24244055, 0.64052536],
       [0.38369828, 0.71960518, 0.64928435, 0.27040705, 0.46105119,
        0.15807951, 0.37484299],
       [0.58022694, 0.51706133, 0.32202709, 0.49157505, 0.29374312,
        0.06434088, 0.25065069],
       [0.35518821, 0.5213285 , 0.53011113, 0.48966055, 0.49255671,
        0.14010913, 0.52675079]])
```

In [59]:
```
labels, sizes = np.unique(labels_res3, return_counts=True)
plt.figure(figsize=(6,3))
for l in np.unique(labels_res3):
    if l == -1:
        continue
    plt.plot(df2.columns, hier_centers[l-1], marker='.', label='Cluster %s (%.2f)' % (l, sizes[l-1]/len(X1_hier

plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

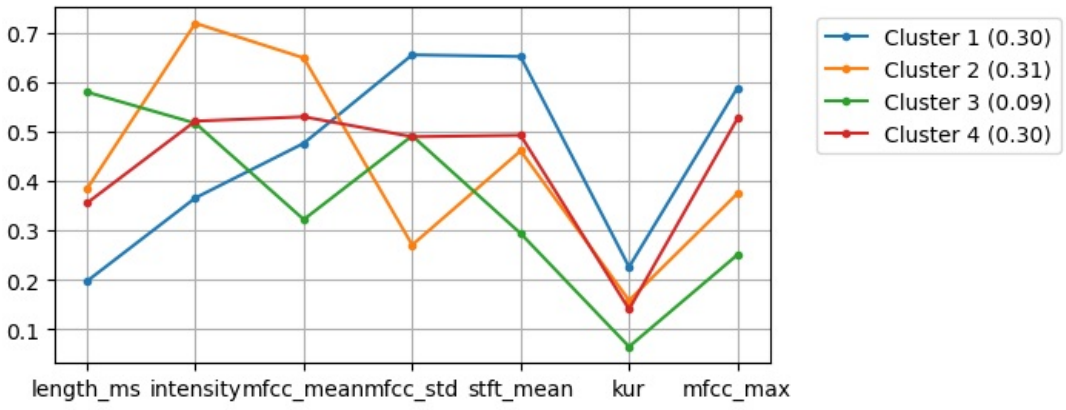



In [60]:
```
hier_centers4 = list()
for l in sorted(np.unique(labels_res2)):
    if l == -1:
        continue
    hier_centers4.append(np.mean(X1_hier[labels_res2 == l], axis=0))
hier_centers4 = np.array(hier_centers4)
```

In [61]:
```
hier_centers4
```

Out[61]:
```
array([[0.19701812, 0.36574927, 0.47647758, 0.65563535, 0.6520782 ,
        0.225772  , 0.58889047],
       [0.38369828, 0.71960518, 0.64928435, 0.27040705, 0.46105119,
        0.15807951, 0.37484299],
       [0.58022694, 0.51706133, 0.32202709, 0.49157505, 0.29374312,
        0.06434088, 0.25065069],
       [0.35518821, 0.5213285 , 0.53011113, 0.48966055, 0.49255671,
        0.14010913, 0.52675079]])
```

In [62]:
```
labels, sizes = np.unique(labels_res2, return_counts=True)
plt.figure(figsize=(6,3))
for l in np.unique(labels_res2):
    if l == -1:
        continue
    plt.plot(df2.columns, hier_centers4[l-1], marker='.', label='Cluster %s (%.2f)' % (l, sizes[l-1]/len(X1_hie

plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```




ward: n.cluster = 5 silhouette = 0.181

complete: n.cluster = 4 silhouette = 0.193

entrambi i metodi risultano ugualmente validi con tuttavia una leggera preferenza per il metodo complete che fornisce una silhouette maggiore

In [ ]:

In [5]:
```
from sklearn.cluster import DBSCAN
```

In [ ]:
```
eps_l=[]
min_samples_l=[]

eps_values = [round(i,2) for i in np.arange(0.13, 0.28, 0.1)]
min_samples_values = [i for i in range(8, 40, 1)]
metric_values = ["euclidean", 'manhattan']

parametri = product(eps_values, min_samples_values, metric_values)

for eps, min_samples, metric in parametri:
    dbscan = DBSCAN(eps=eps, min_samples=min_samples, metric=metric)

    dbscan.fit(dbscan_scaled)
    labels = dbscan.labels_

    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise = list(labels).count(-1)

    if 50<=n_noise<=500 and n_clusters >=2:
        print(f"Eps={eps}, min_samples={min_samples}, metric={metric}")
        print(f"Cluster: {n_clusters}")
        print(f"Noise: {n_noise}")
        print("-------------------------")
        eps_l.append(eps)
        min_samples_l.append(min_samples)
```

Nel corso dell'analisi del DBSCAN sono state eseguite diversi cicli for per cercare valori ottimali di eps e min_samples. Tuttavia quando venivano rilevati cluster con popolazioni accettabili la silhouette veniva negativa. In alcuni casi è stata trovata silhouette positiva ma con cluster inutilizzabili (elevato numero di punti di rumore, popolazioni sbilanciate ecc)

In [6]:
```
dbscan = DBSCAN(min_samples=25, eps=0.3, metric='manhattan')
```

In [7]:
```
dbscan.fit(dbscan_scaled)
```

Out[7]:
```
DBSCAN(eps=0.3, metric='manhattan', min_samples=25)
```

In [8]:
```
labels, sizes = np.unique(dbscan.labels_, return_counts=True)
for l, s in zip(labels, sizes):
    print('Cluster %s, size %s (%.2f)' % (l, s, s/len(dbscan_scaled)))
```

```
Cluster -1, size 1096 (0.45)
Cluster 0, size 1323 (0.54)
Cluster 1, size 9 (0.00)
Cluster 2, size 24 (0.01)
```

In [23]:
```
df0['dbscan'] = dbscan.labels_
```

In [24]:
```
df0
```

Out[24]:

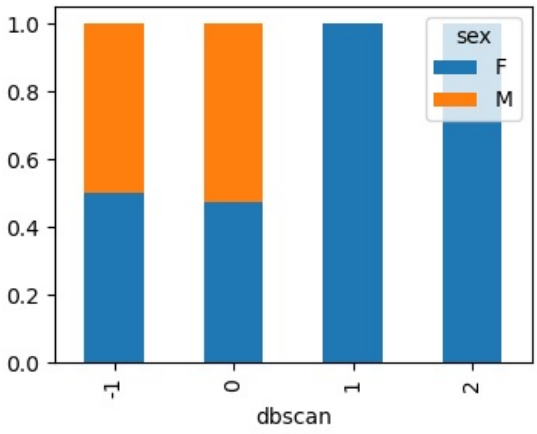
| | modality | vocal_channel | emotion | emotional_intensity | statement | repetition | actor | sex | channels | sample_width | ... | stft_m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | audio-only | speech | fearful | normal | Dogs are sitting by the door | 2nd | 2.0 | F | 1 | 2 | ... | 1 |
| **1** | audio-only | speech | angry | normal | Dogs are sitting by the door | 1st | 16.0 | F | 1 | 2 | ... | 1 |
| **2** | audio-only | NaN | happy | strong | Dogs are sitting by the door | 2nd | 16.0 | F | 1 | 2 | ... | 1 |
| **3** | audio-only | NaN | surprised | normal | Kids are talking by the door | 1st | 14.0 | F | 1 | 2 | ... | 1 |
| **4** | audio-only | song | happy | strong | Dogs are sitting by the door | 2nd | 2.0 | F | 1 | 2 | ... | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2447** | audio-only | speech | calm | strong | Kids are talking by the door | 1st | 23.0 | M | 1 | 2 | ... | 1 |
| **2448** | audio-only | speech | calm | normal | Dogs are sitting by the door | 1st | 23.0 | M | 1 | 2 | ... | 1 |
| **2449** | audio-only | song | sad | strong | Dogs are sitting by the door | 2nd | 23.0 | M | 1 | 2 | ... | 1 |
| **2450** | audio-only | speech | surprised | normal | Kids are talking by the door | 1st | NaN | M | 1 | 2 | ... | 1 |
| **2451** | audio-only | NaN | neutral | normal | Dogs are sitting by the door | 2nd | 23.0 | M | 1 | 2 | ... | 1 |

2452 rows × 39 columns

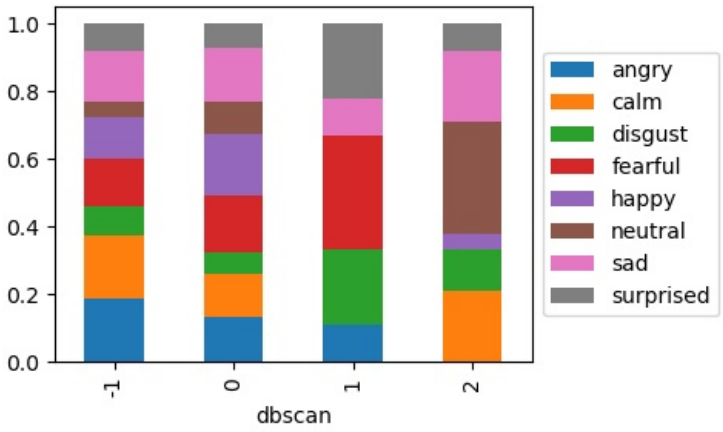
In [16]:
```
silhouette_score(dbscan_scaled, dbscan.labels_)
```

Out[16]:
```
-0.17736833433489774
```

In [25]:
```
ct=pd.crosstab(df0['dbscan'], df0['sex'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.show()
```




In [27]:
```
ct=pd.crosstab(df0['dbscan'], df0['emotion'])
ct.div(ct.sum(axis=1), axis=0).plot(kind='bar', stacked=True, figsize=(4,3))
plt.legend(fontsize=10, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```




In [ ]:

In [14]:
```python
from itertools import product
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import numpy as np

eps_values = [round(i, 2) for i in np.arange(0.1, 0.21, 0.01)]
min_samples_values = range(8, 50)
metric_values = ["euclidean", "manhattan"]
parameter_combinations = product(eps_values, min_samples_values, metric_values)

for eps, min_samples, metric in parameter_combinations:
    dbscan = DBSCAN(eps=eps, min_samples=min_samples, metric=metric)
    dbscan.fit(dbscan_scaled)
    labels, sizes = np.unique(dbscan.labels_, return_counts=True)
    n_clusters = len(labels) - (1 if -1 in labels else 0)
    n_noise = sizes[labels == -1][0] if -1 in labels else 0

    if n_clusters >= 2 and n_clusters <= 4 and n_noise <= 1500:
        sil = silhouette_score(dbscan_scaled, dbscan.labels_)
        if sil >= 0:
            print(f"Parametri: eps={eps}, min_samples={min_samples}, metric={metric}")
            print(f"Numero di cluster: {n_clusters}")
            print(f"Numero di punti rumorosi: {n_noise}")
            print(f"Coefficiente silhouette: {sil:.2f}")
            for l, s in zip(labels, sizes):
                print(f'Cluster {l}, size {s} ({s/len(dbscan_scaled)*100:.1f}%)')
            print("-------------------------")
```

```
Parametri: eps=0.13, min_samples=11, metric=euclidean
Numero di cluster: 3
Numero di punti rumorosi: 807
Coefficiente silhouette: 0.04
Cluster -1, size 807 (32.9%)
Cluster 0, size 1626 (66.3%)
Cluster 1, size 11 (0.4%)
Cluster 2, size 8 (0.3%)
-------------------------
Parametri: eps=0.14, min_samples=8, metric=euclidean
Numero di cluster: 4
Numero di punti rumorosi: 421
Coefficiente silhouette: 0.01
Cluster -1, size 421 (17.2%)
Cluster 0, size 2010 (82.0%)
Cluster 1, size 9 (0.4%)
Cluster 2, size 4 (0.2%)
Cluster 3, size 8 (0.3%)
-------------------------
Parametri: eps=0.15, min_samples=8, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 286
Coefficiente silhouette: 0.17
Cluster -1, size 286 (11.7%)
Cluster 0, size 2157 (88.0%)
Cluster 1, size 9 (0.4%)
-------------------------
Parametri: eps=0.15, min_samples=9, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 324
Coefficiente silhouette: 0.07
Cluster -1, size 324 (13.2%)
Cluster 0, size 2120 (86.5%)
Cluster 1, size 8 (0.3%)
-------------------------
```

```
Parametri: eps=0.15, min_samples=39, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 1338
Coefficiente silhouette: 0.00
Cluster -1, size 1338 (54.6%)
Cluster 0, size 300 (12.2%)
Cluster 1, size 814 (33.2%)
-------------------------
Parametri: eps=0.16, min_samples=8, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 195
Coefficiente silhouette: 0.19
Cluster -1, size 195 (8.0%)
Cluster 0, size 2248 (91.7%)
Cluster 1, size 9 (0.4%)
-------------------------
Parametri: eps=0.16, min_samples=49, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 1175
Coefficiente silhouette: 0.02
Cluster -1, size 1175 (47.9%)
Cluster 0, size 322 (13.1%)
Cluster 1, size 955 (38.9%)
-------------------------
Parametri: eps=0.17, min_samples=8, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 133
Coefficiente silhouette: 0.21
Cluster -1, size 133 (5.4%)
Cluster 0, size 2308 (94.1%)
Cluster 1, size 11 (0.4%)
-------------------------
Parametri: eps=0.17, min_samples=9, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 146
Coefficiente silhouette: 0.21
Cluster -1, size 146 (6.0%)
Cluster 0, size 2295 (93.6%)
Cluster 1, size 11 (0.4%)
-------------------------
Parametri: eps=0.18, min_samples=11, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 126
Coefficiente silhouette: 0.22
Cluster -1, size 126 (5.1%)
Cluster 0, size 2315 (94.4%)
Cluster 1, size 11 (0.4%)
-------------------------
Parametri: eps=0.18, min_samples=12, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 130
Coefficiente silhouette: 0.22
Cluster -1, size 130 (5.3%)
Cluster 0, size 2311 (94.2%)
Cluster 1, size 11 (0.4%)
-------------------------
Parametri: eps=0.19, min_samples=12, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 90
Coefficiente silhouette: 0.24
Cluster -1, size 90 (3.7%)
Cluster 0, size 2349 (95.8%)
Cluster 1, size 13 (0.5%)
-------------------------
Parametri: eps=0.19, min_samples=13, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 102
Coefficiente silhouette: 0.23
Cluster -1, size 102 (4.2%)
Cluster 0, size 2338 (95.4%)
Cluster 1, size 12 (0.5%)
-------------------------
Parametri: eps=0.2, min_samples=13, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 71
Coefficiente silhouette: 0.25
Cluster -1, size 71 (2.9%)
Cluster 0, size 2369 (96.6%)
Cluster 1, size 12 (0.5%)
-------------------------
Parametri: eps=0.2, min_samples=14, metric=euclidean
Numero di cluster: 2
Numero di punti rumorosi: 75
```

```
Coefficiente silhouette: 0.25
Cluster -1, size 75 (3.1%)
Cluster 0, size 2367 (96.5%)
Cluster 1, size 10 (0.4%)
-------------------------
```

il DBSCAN risulta quindi inadatto per il clustering del nostro DF

In [ ]:

In [13]:
```
df2 = df.drop(['actor','sc_min','mean'], axis=1)
```

In [14]:
```
df2.drop(['modality', 'sample_width', 'frame_rate','frame_count','frame_width','stft_max'], axis=1, inplace=Tru
```

In [15]:
```
df2
```

Out[15]:

| | vocal_channel | emotion | emotional_intensity | statement | repetition | sex | channels | length_ms | intensity | zero_crossings_su |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | speech | fearful | normal | Dogs are sitting by the door | 2nd | F | 1 | 3737 | -36.793432 | 169 |
| **1** | speech | angry | normal | Dogs are sitting by the door | 1st | F | 1 | 3904 | -34.546284 | 139 |
| **2** | song | happy | strong | Dogs are sitting by the door | 2nd | F | 1 | 4671 | -32.290737 | 187 |
| **3** | song | surprised | normal | Kids are talking by the door | 1st | F | 1 | 3637 | -49.019839 | 116 |
| **4** | song | happy | strong | Dogs are sitting by the door | 2nd | F | 1 | 4404 | -31.214503 | 151 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2447** | speech | calm | strong | Kids are talking by the door | 1st | M | 1 | 4605 | -46.264888 | 98 |
| **2448** | speech | calm | normal | Dogs are sitting by the door | 1st | M | 1 | 4171 | -43.342901 | 89 |
| **2449** | song | sad | strong | Dogs are sitting by the door | 2nd | M | 1 | 5239 | -38.245412 | 97 |
| **2450** | speech | surprised | normal | Kids are talking by the door | 1st | M | 1 | 3737 | -45.751265 | 97 |
| **2451** | song | neutral | normal | Dogs are sitting by the door | 2nd | M | 1 | 3837 | -40.018044 | 94 |

2452 rows × 29 columns

In [17]:
```
df2['sex'] = df2['sex'].map({'M': 0, 'F': 1})
df2
```

Out[17]:

| | vocal_channel | emotion | emotional_intensity | statement | repetition | sex | channels | length_ms | intensity | zero_crossings_su |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | speech | fearful | normal | Dogs are sitting by the door | 2nd | 1 | 1 | 3737 | -36.793432 | 169 |
| **1** | speech | angry | normal | Dogs are sitting by the door | 1st | 1 | 1 | 3904 | -34.546284 | 139 |
| **2** | song | happy | strong | Dogs are sitting by the door | 2nd | 1 | 1 | 4671 | -32.290737 | 187 |
| **3** | song | surprised | normal | Kids are talking by the door | 1st | 1 | 1 | 3637 | -49.019839 | 116 |
| **4** | song | happy | strong | Dogs are sitting by the door | 2nd | 1 | 1 | 4404 | -31.214503 | 151 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2447** | speech | calm | strong | Kids are talking by the door | 1st | 0 | 1 | 4605 | -46.264888 | 98 |
| **2448** | speech | calm | normal | Dogs are sitting by the door | 1st | 0 | 1 | 4171 | -43.342901 | 89 |
| **2449** | song | sad | strong | Dogs are sitting by the door | 2nd | 0 | 1 | 5239 | -38.245412 | 97 |
| **2450** | speech | surprised | normal | Kids are talking by the door | 1st | 0 | 1 | 3737 | -45.751265 | 97 |
| **2451** | song | neutral | normal | Dogs are sitting by the door | 2nd | 0 | 1 | 3837 | -40.018044 | 94 |

2452 rows × 29 columns

In [18]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   vocal_channel        2452 non-null   object 
 1   emotion              2452 non-null   object 
 2   emotional_intensity  2452 non-null   object 
 3   statement            2452 non-null   object 
 4   repetition           2452 non-null   object 
 5   sex                  2452 non-null   int64  
 6   channels             2452 non-null   int64  
 7   length_ms            2452 non-null   int64  
 8   intensity            2452 non-null   float64
 9   zero_crossings_sum   2452 non-null   int64  
 10  mfcc_mean            2452 non-null   float64
 11  mfcc_std             2452 non-null   float64
 12  mfcc_min             2452 non-null   float64
 13  mfcc_max             2452 non-null   float64
 14  sc_mean              2452 non-null   float64
 15  sc_std               2452 non-null   float64
 16  sc_max               2452 non-null   float64
 17  sc_kur               2452 non-null   float64
 18  sc_skew              2452 non-null   float64
 19  stft_mean            2452 non-null   float64
 20  stft_std             2452 non-null   float64
 21  stft_min             2452 non-null   float64
 22  stft_kur             2452 non-null   float64
 23  stft_skew            2452 non-null   float64
 24  std                  2452 non-null   float64
 25  min                  2452 non-null   float64
 26  max                  2452 non-null   float64
 27  kur                  2452 non-null   float64
 28  skew                 2452 non-null   float64
dtypes: float64(20), int64(4), object(5)
memory usage: 555.7+ KB
```

In [19]:
```
df2['vocal_channel'] = df2['vocal_channel'].map({'speech': 0, 'song': 1})
df2['emotional_intensity'] = df2['emotional_intensity'].map({'normal': 0, 'strong': 1})
df2['statement'] = df2['statement'].map({'Dogs are sitting by the door': 0, 'Kids are talking by the door': 1})
```

```
df2['repetition'] = df2['repetition'].map({'1st': 0, '2nd': 1})
```

In [20]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   vocal_channel        2452 non-null   int64  
 1   emotion              2452 non-null   object 
 2   emotional_intensity  2452 non-null   int64  
 3   statement            2452 non-null   int64  
 4   repetition           2452 non-null   int64  
 5   sex                  2452 non-null   int64  
 6   channels             2452 non-null   int64  
 7   length_ms            2452 non-null   int64  
 8   intensity            2452 non-null   float64
 9   zero_crossings_sum   2452 non-null   int64  
 10  mfcc_mean            2452 non-null   float64
 11  mfcc_std             2452 non-null   float64
 12  mfcc_min             2452 non-null   float64
 13  mfcc_max             2452 non-null   float64
 14  sc_mean              2452 non-null   float64
 15  sc_std               2452 non-null   float64
 16  sc_max               2452 non-null   float64
 17  sc_kur               2452 non-null   float64
 18  sc_skew              2452 non-null   float64
 19  stft_mean            2452 non-null   float64
 20  stft_std             2452 non-null   float64
 21  stft_min             2452 non-null   float64
 22  stft_kur             2452 non-null   float64
 23  stft_skew            2452 non-null   float64
 24  std                  2452 non-null   float64
 25  min                  2452 non-null   float64
 26  max                  2452 non-null   float64
 27  kur                  2452 non-null   float64
 28  skew                 2452 non-null   float64
dtypes: float64(20), int64(8), object(1)
memory usage: 555.7+ KB
```

speech, normal, dogs, 1st, male == 0

song, strong, kids, 2nd, female == 1

In [26]:
```
df2.drop(['stft_min','channels'], axis=1, inplace=True)
```

In [27]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 27 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   vocal_channel        2452 non-null   int64  
 1   emotion              2452 non-null   object 
 2   emotional_intensity  2452 non-null   int64  
 3   statement            2452 non-null   int64  
 4   repetition           2452 non-null   int64  
 5   sex                  2452 non-null   int64  
 6   length_ms            2452 non-null   int64  
 7   intensity            2452 non-null   float64
 8   zero_crossings_sum   2452 non-null   int64  
 9   mfcc_mean            2452 non-null   float64
 10  mfcc_std             2452 non-null   float64
 11  mfcc_min             2452 non-null   float64
 12  mfcc_max             2452 non-null   float64
 13  sc_mean              2452 non-null   float64
 14  sc_std               2452 non-null   float64
 15  sc_max               2452 non-null   float64
 16  sc_kur               2452 non-null   float64
 17  sc_skew              2452 non-null   float64
 18  stft_mean            2452 non-null   float64
 19  stft_std             2452 non-null   float64
 20  stft_kur             2452 non-null   float64
 21  stft_skew            2452 non-null   float64
 22  std                  2452 non-null   float64
 23  min                  2452 non-null   float64
 24  max                  2452 non-null   float64
 25  kur                  2452 non-null   float64
 26  skew                 2452 non-null   float64
dtypes: float64(19), int64(7), object(1)
memory usage: 517.3+ KB
```

## ho creato un df con mappati i valori delle variabili binarie categoriche. non mappo i valori di emotion perchè sono quelli che andrò a predirre

In [28]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
```

In [29]:
```
target = 'emotion'
columns = [c for c in df2.columns if c != target]
```

In [30]:
```
X = df2[columns].values
y = df2[target].values
```

In [31]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [32]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.15, stratify=y, random_state=9)
```

In [33]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, stratify=y_train_va
```

In [34]:
```
len(X_train), len(X_val), len(X_test)
```

Out[34]:
```
(1771, 313, 368)
```

## ho usato 15% per test e per validation, ora knn

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
```

In [36]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_val)
y_pred_train = knn.predict(X_train)
```

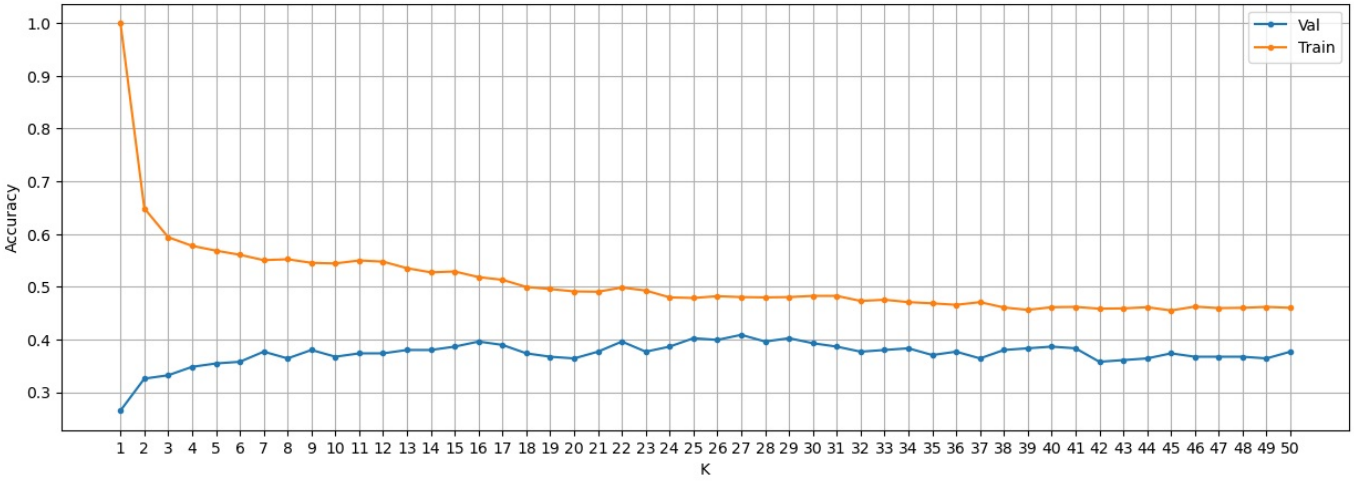
In [37]:
```
accuracy_score(y_val, y_pred), accuracy_score(y_train, y_pred_train)
```

Out[37]:
```
(0.3546325878594249, 0.5686053077357425)
```

## inizialmente ottengo 0.35 di accuracy sul validation set

In [38]:
```
acc_val_list = list()
acc_train_list = list()
for k in np.arange(1, 50+1, 1):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_val)
    y_pred_train = knn.predict(X_train)
    acc_val_list.append(accuracy_score(y_val, y_pred))
    acc_train_list.append(accuracy_score(y_train, y_pred_train))
```

In [39]:
```
plt.figure(figsize=(15,5))
plt.plot(np.arange(1, 50+1, 1), acc_val_list, label='Val', marker='.')
plt.plot(np.arange(1, 50+1, 1), acc_train_list, label='Train', marker='.')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.xticks(np.arange(1, 50+1, 1))
plt.grid()
plt.legend()
plt.show()
```

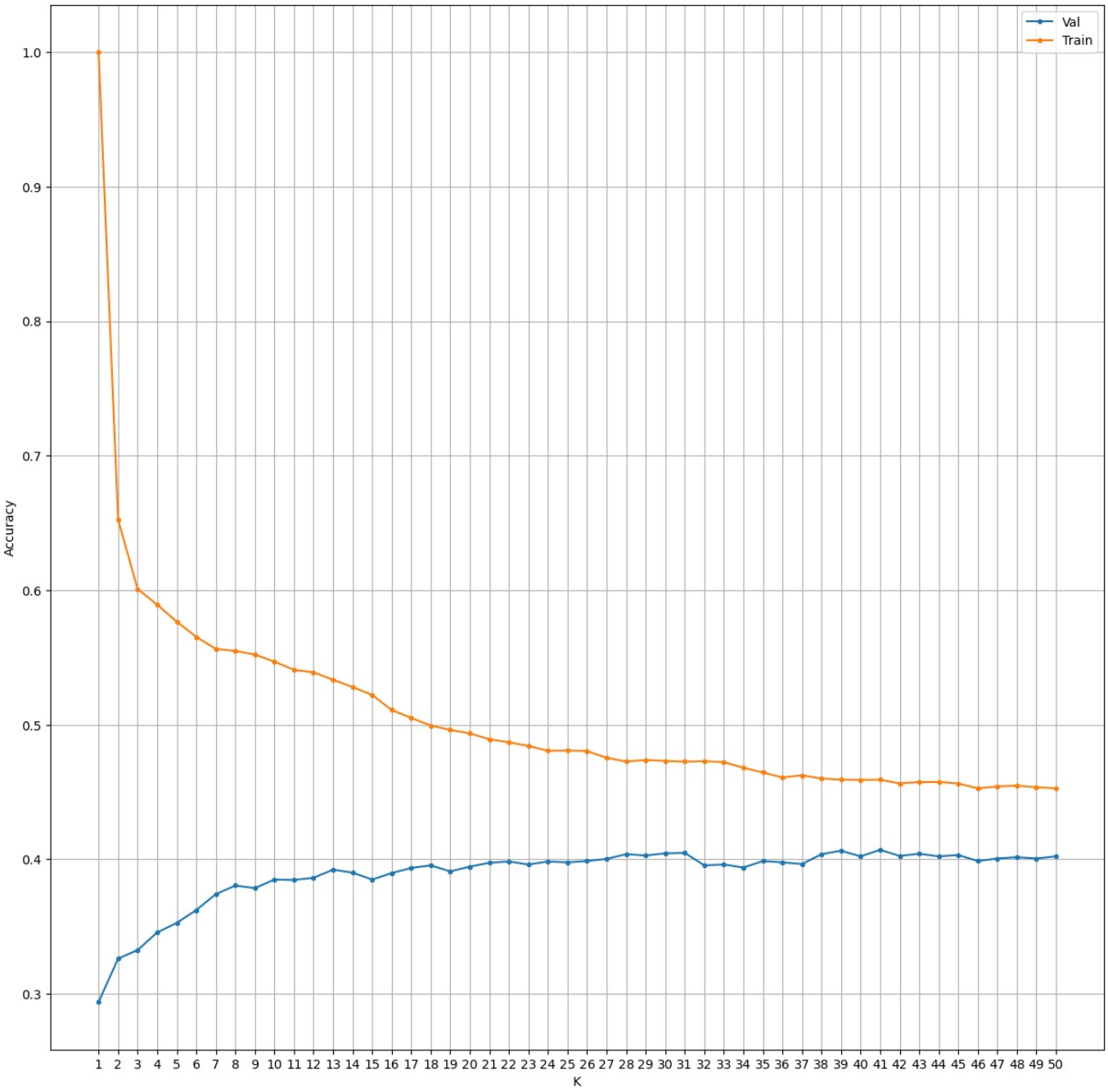



ora faccio ciclo for per eseguire più volte la scelta random del train e test, prendo media e vedo quale numero k è meglio

In [40]:
```
nbr_holdout = 10
acc_val_list_all = list()
acc_train_list_all = list()
for i in range(nbr_holdout):

    X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,
                                                      test_size=0.15,
                                                      stratify=y_train_val,
                                                     random_state=i)

    acc_val_list = list()
    acc_train_list = list()
    for k in np.arange(1, 50+1, 1):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_val)
        y_pred_train = knn.predict(X_train)
        acc_val_list.append(accuracy_score(y_val, y_pred))
        acc_train_list.append(accuracy_score(y_train, y_pred_train))

    acc_val_list_all.append(acc_val_list)
    acc_train_list_all.append(acc_train_list)
```

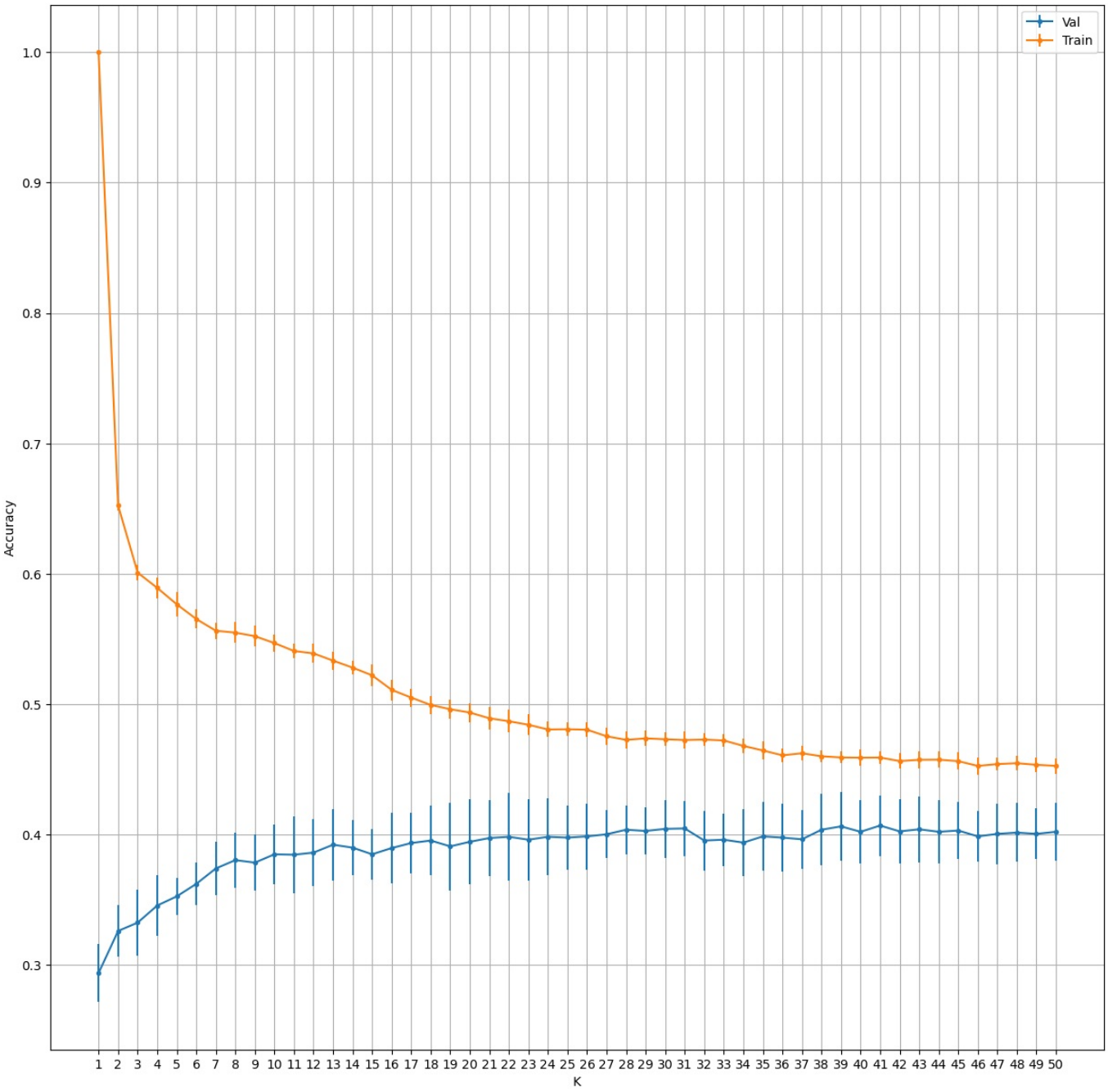
In [41]:
```
acc_val_list_all = np.array(acc_val_list_all)
acc_train_list_all = np.array(acc_train_list_all)
```

In [42]:
```
plt.figure(figsize=(15,15))
plt.plot(np.arange(1, 50+1, 1), np.mean(acc_val_list_all, axis=0), label='Val', marker='.')
plt.plot(np.arange(1, 50+1, 1), np.mean(acc_train_list_all, axis=0), label='Train', marker='.')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.xticks(np.arange(1, 50+1, 1))
plt.grid()
plt.legend()
plt.show()
```




prendo in considerazione std per vedere quali sono i k che hanno meno varianza

In [43]:
```
plt.figure(figsize=(15,15))
plt.errorbar(x=np.arange(1, 50+1, 1),
             y=np.mean(acc_val_list_all, axis=0),
             yerr=np.std(acc_val_list_all, axis=0),
             label='Val', marker='.')

plt.errorbar(x=np.arange(1, 50+1, 1),
             y=np.mean(acc_train_list_all, axis=0),
             yerr=np.std(acc_train_list_all, axis=0),
             label='Train', marker='.')

plt.xlabel('K')
plt.ylabel('Accuracy')
plt.xticks(np.arange(1, 50+1, 1))
plt.grid()
plt.legend()
plt.show()
```




da qua sembra buono 28 come k. ora cross validation.

In [45]:
```
nbr_repetitions = 10
acc_list_all = list()
for i in range(nbr_repetitions):
    acc_list = list()
    for k in np.arange(1, 60+1, 1):
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X_train_val, y_train_val, cv=10, #k-fold
                                 scoring='accuracy')
        acc_list.append(scores)

    acc_list_all.append(acc_list)
```

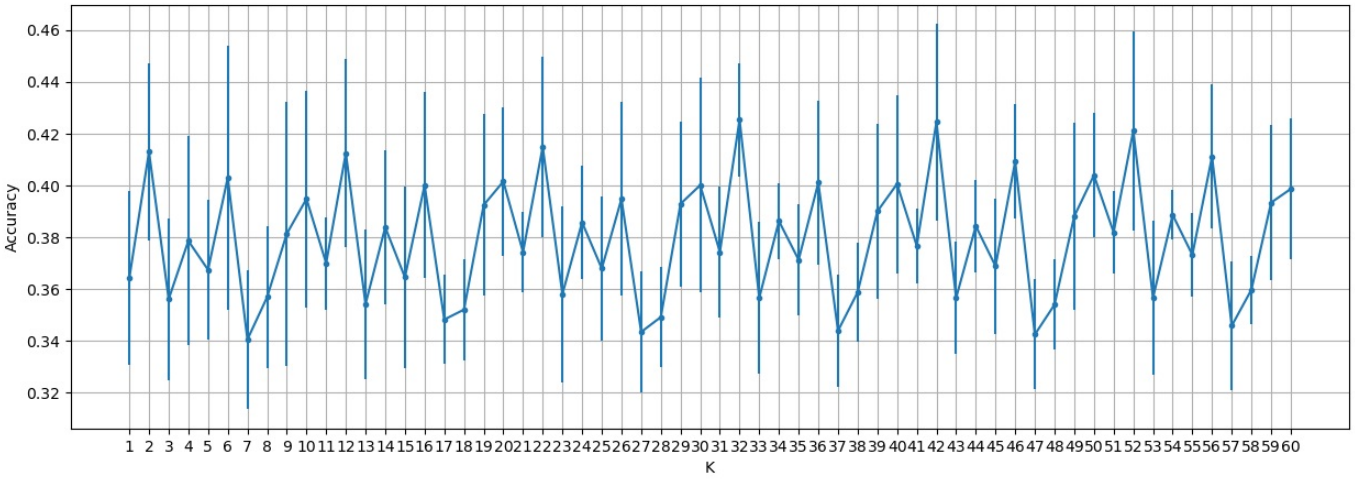
In [46]:
```
acc_list_all = np.array(acc_list_all)
```

In [47]:
```
acc_list_all.reshape(100, 60).shape
```

Out[47]:
```
(100, 60)
```

In [48]:
```
plt.figure(figsize=(15,5))
plt.errorbar(x=np.arange(1, 60+1, 1),
             y=np.mean(acc_list_all.reshape(100, 60), axis=0),
             yerr=np.std(acc_list_all.reshape(100, 60), axis=0),
             marker='.')

plt.xlabel('K')
plt.ylabel('Accuracy')
plt.xticks(np.arange(1, 60+1, 1))
```

```
plt.grid()
plt.show()
```




con la cross validation direi k migliore = 32, ha una varianza non trascurabile però l'accuratezza più bassa che posso raggiungere con k=32 rimane comunque un valore valido (>0.40)

In [68]:
```
knn = KNeighborsClassifier(n_neighbors=32)
knn.fit(X_train_val, y_train_val)

y_pred = knn.predict(X_test)
print(classification_report(y_pred, y_test))

y_pred_trainval = knn.predict(X_train_val)
print(classification_report(y_pred_trainval, y_train_val))
```

```
              precision    recall  f1-score   support

       angry       0.62      0.55      0.58        64
        calm       0.45      0.38      0.41        65
     disgust       0.34      0.34      0.34        29
     fearful       0.47      0.46      0.47        59
       happy       0.30      0.30      0.30        56
     neutral       0.29      0.27      0.28        30
         sad       0.23      0.38      0.29        34
   surprised       0.45      0.42      0.43        31

    accuracy                           0.40       368
   macro avg       0.39      0.39      0.39       368
weighted avg       0.42      0.40      0.41       368

              precision    recall  f1-score   support

       angry       0.71      0.66      0.69       344
        calm       0.63      0.55      0.59       370
     disgust       0.45      0.44      0.44       167
     fearful       0.43      0.46      0.45       300
       happy       0.40      0.39      0.39       332
     neutral       0.42      0.40      0.41       171
         sad       0.26      0.45      0.33       181
   surprised       0.52      0.38      0.44       219

    accuracy                           0.48      2084
   macro avg       0.48      0.47      0.47      2084
weighted avg       0.50      0.48      0.49      2084
```

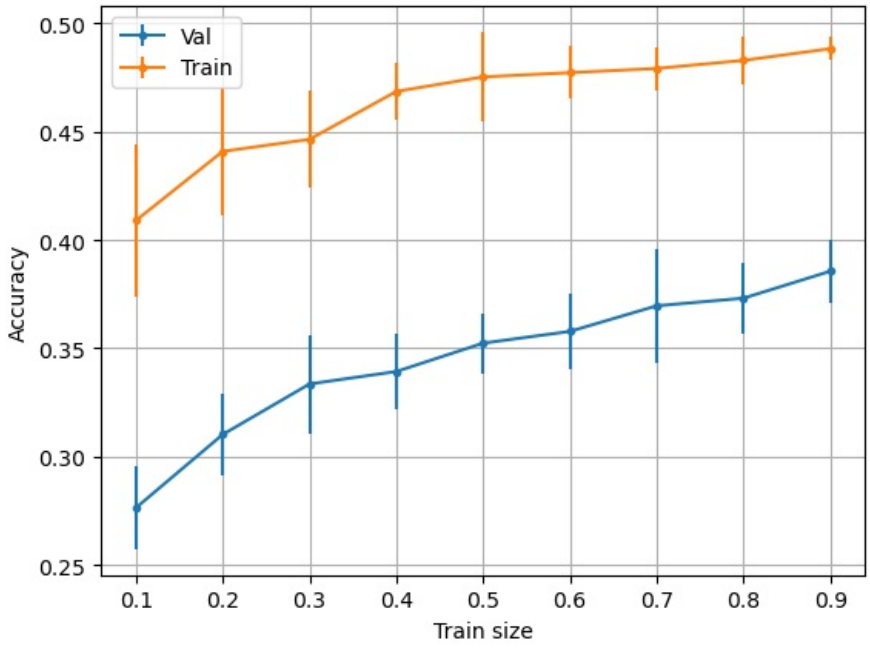
In [52]:
```
nbr_repetitions = 10
acc_val_list_all = list()
acc_train_list_all = list()
for p in np.arange(0.1, 1.0, 0.1):
    acc_val_list = list()
    acc_train_list = list()
    for i in range(nbr_repetitions):
        index = np.random.choice(np.arange(len(X_train)), int(len(X_train) * p), replace=False)
        knn = KNeighborsClassifier(n_neighbors=21)
        knn.fit(X_train[index], y_train[index])
        y_pred = knn.predict(X_val)
        y_pred_train = knn.predict(X_train[index])
        acc_val_list.append(accuracy_score(y_val, y_pred))
        acc_train_list.append(accuracy_score(y_train[index], y_pred_train))

    acc_val_list_all.append(acc_val_list)
    acc_train_list_all.append(acc_train_list)
```

In [53]:
```
acc_val_list_all = np.array(acc_val_list_all)
acc_train_list_all = np.array(acc_train_list_all)
```

In [54]:
```
np.mean(acc_val_list_all, axis=1)
```

Out[54]:
```
array([0.27635783, 0.31022364, 0.33354633, 0.33929712, 0.35239617,
       0.35782748, 0.36964856, 0.37316294, 0.385623  ])
```

In [55]:
```
plt.errorbar(x=np.arange(0.1, 1.0, 0.1),
             y=np.mean(acc_val_list_all, axis=1),
             yerr=np.std(acc_val_list_all, axis=1),
             label='Val', marker='.')

plt.errorbar(x=np.arange(0.1, 1.0, 0.1),
             y=np.mean(acc_train_list_all, axis=1),
             yerr=np.std(acc_train_list_all, axis=1),
             label='Train', marker='.')

plt.xlabel('Train size')
plt.ylabel('Accuracy')
plt.xticks(np.arange(0.1, 1.0, 0.1))
plt.grid()
plt.legend()
plt.show()
```




in generale l'accuratezza aumenta aumentando la train size. con train size di 0.5 l'accuratezza sul validation test è quasi il 90% dell'acuratezza considerando tutto il train size

In [71]:
```
knn = KNeighborsClassifier(n_neighbors=32)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_val)
y_pred_train = knn.predict(X_train)
y_pred_test = knn.predict(X_test)
```

In [72]:
```
accuracy_score(y_val, y_pred), accuracy_score(y_train, y_pred_train), accuracy_score(y_test,y_pred_test)
```

Out[72]:
```
(0.3769968051118211, 0.4731789949181254, 0.39945652173913043)
```

In [73]:
```
print(classification_report(y_pred_test, y_test))
```

```
              precision    recall  f1-score   support

       angry       0.64      0.59      0.62        61
        calm       0.45      0.47      0.46        53
     disgust       0.28      0.29      0.28        28
     fearful       0.42      0.38      0.40        63
       happy       0.30      0.28      0.29        61
     neutral       0.39      0.31      0.35        35
         sad       0.23      0.36      0.28        36
   surprised       0.45      0.42      0.43        31

    accuracy                           0.40       368
   macro avg       0.39      0.39      0.39       368
weighted avg       0.41      0.40      0.40       368
```

ottengo un'accuracy del 0.4

le emozioni più difficili da predirre sono: sad, disgust

sad nonostante bassa precision ha alta recall più alta.

angry ha i valori più alti di precision e recall seguita da calm

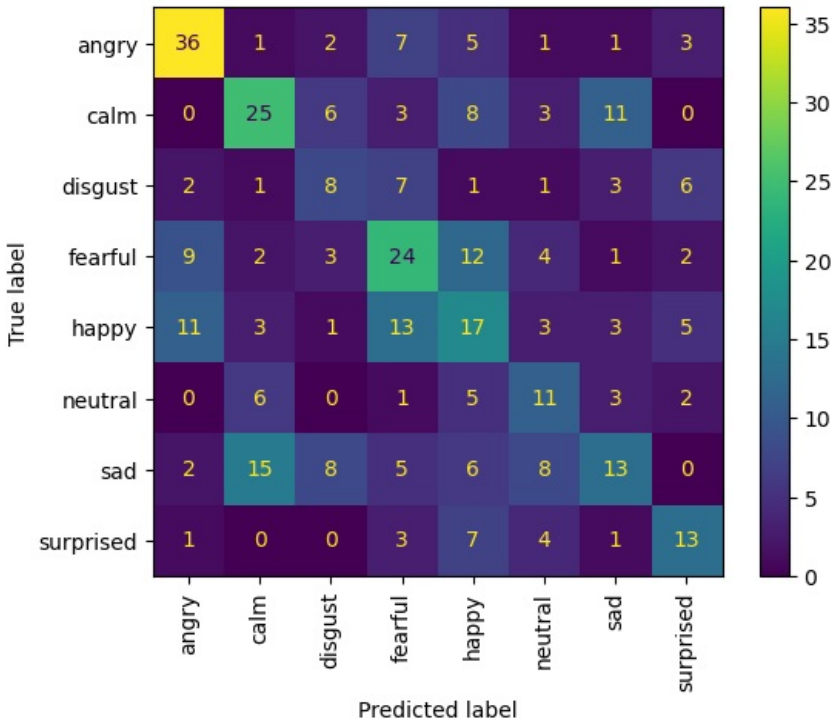
In [60]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [74]:
```
cm = confusion_matrix(y_test, y_pred_test)
cm
```

Out[74]:
```
array([[36,  1,  2,  7,  5,  1,  1,  3],
       [ 0, 25,  6,  3,  8,  3, 11,  0],
       [ 2,  1,  8,  7,  1,  1,  3,  6],
       [ 9,  2,  3, 24, 12,  4,  1,  2],
       [11,  3,  1, 13, 17,  3,  3,  5],
       [ 0,  6,  0,  1,  5, 11,  3,  2],
       [ 2, 15,  8,  5,  6,  8, 13,  0],
       [ 1,  0,  0,  3,  7,  4,  1, 13]], dtype=int64)
```

In [75]:
```
plt.figure(figsize=(10,10))
disp_cm = ConfusionMatrixDisplay(cm,display_labels=knn.classes_)
disp_cm.plot()
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```




dalla confusion matrix si nota come sad venga confusa per calm e viceversa

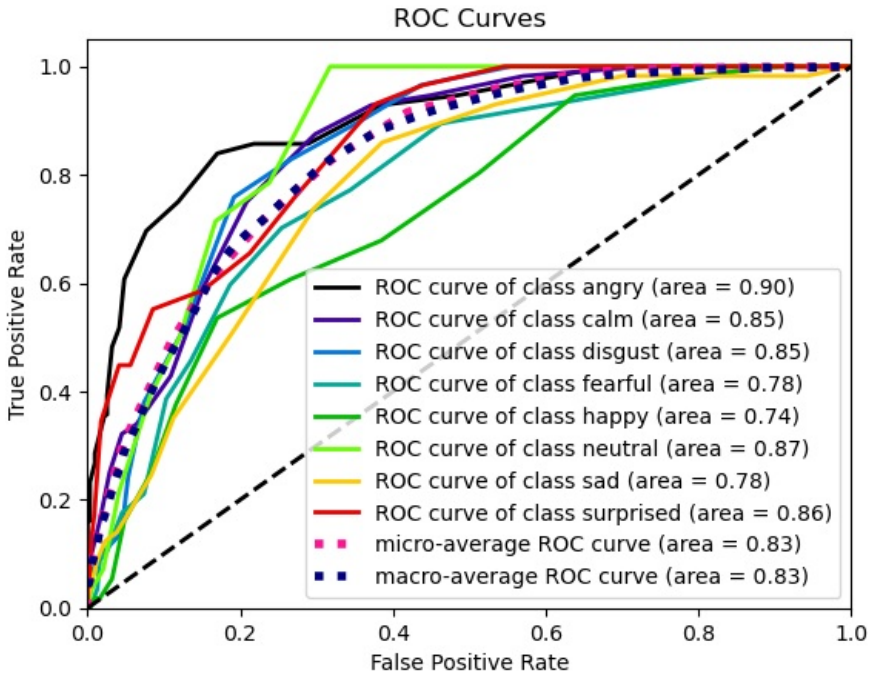
a volte happy viene confusa con fearful, anche viceversa ma con proporzione inferiore

angry risulta essere quella riconosciuta correttamente la maggior parte delle volte, come emerso anche in precedenza

In [63]:
```
import scikitplot
```

In [76]:
```
y_pred_prob = knn.predict_proba(X_test)
```

In [77]:
```
scikitplot.metrics.plot_roc(y_test, y_pred_prob)
plt.show()
```




la roc curve conferma quanto notato finora: area di angry prossima ad 1. la più bassa risulta essere happy (0.74)

sono tutte ben sopra la diagonale quindi in definitiva i risultati ottenuti dalla classificazione possono dirsi soddisfacenti

In [ ]:

# DECISION TREE

In [3]:
```
categorical_cols = ["sex","vocal_channel","repetition","statement", "emotional_intensity"]
```

In [22]:
```
df = pd.get_dummies(df, columns=categorical_cols)
X = df.values
```

In [9]:
```
target = 'emotion'
columns = [c for c in df.columns if c != target]
```

In [10]:
```
X = df[columns].values
y = df[target].values
```

In [11]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
```

In [12]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.3, stratify=y_train_val
```

## Decision tree con parametri default

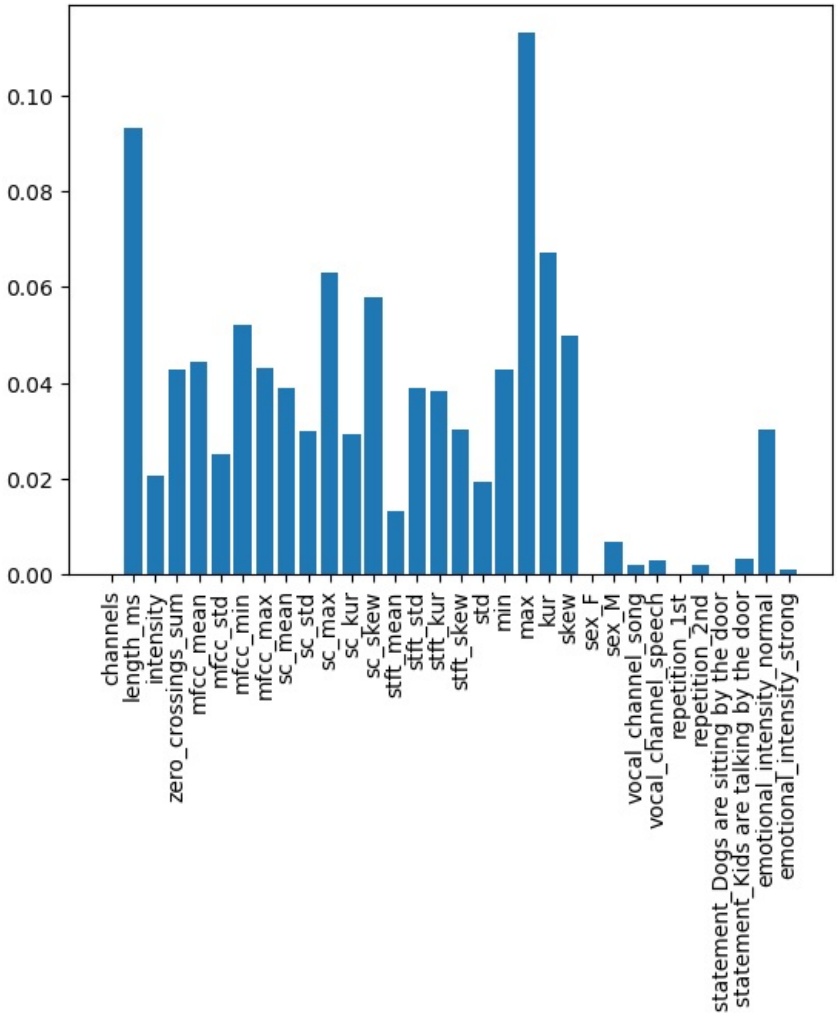
In [13]:
```
clf = DecisionTreeClassifier(
    criterion='gini',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    ccp_alpha=0.0,
    random_state=0
)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)
print(accuracy_score(y_val, y_pred))
```

```
0.39805825242718446
```

In [14]:
```
confusion_matrix(y_val, y_pred)
```

Out[14]:
```
array([[41,  1,  3, 15, 12,  0,  4,  3],
       [ 2, 38,  6,  6,  3,  5, 18,  1],
       [ 7,  2, 11,  1,  6,  1, 10,  2],
       [10,  8,  3, 28, 12,  4,  8,  6],
       [10,  3,  6, 18, 26,  5,  6,  5],
       [ 1,  6,  1,  5,  4, 18,  5,  0],
       [ 5, 12,  6,  4,  6,  8, 34,  4],
       [ 6,  0,  1,  3,  5,  5, 11,  9]], dtype=int64)
```

In [15]:
```
plt.bar(columns, clf.feature_importances_)
plt.xticks(rotation=90)
plt.show()
```




## Variazione parametri

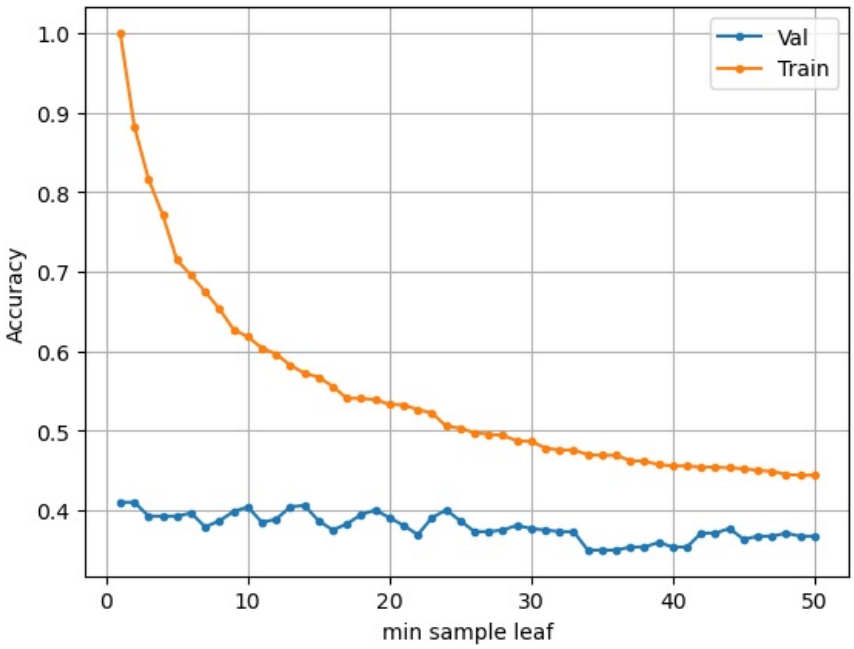
In [23]:
```
target = 'emotion'
columns = [c for c in df.columns if c != target]
```

In [24]:
```
X = df[columns].values
y = df[target].values
```

In [25]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
```

In [26]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.3, stratify=y_train_val
```

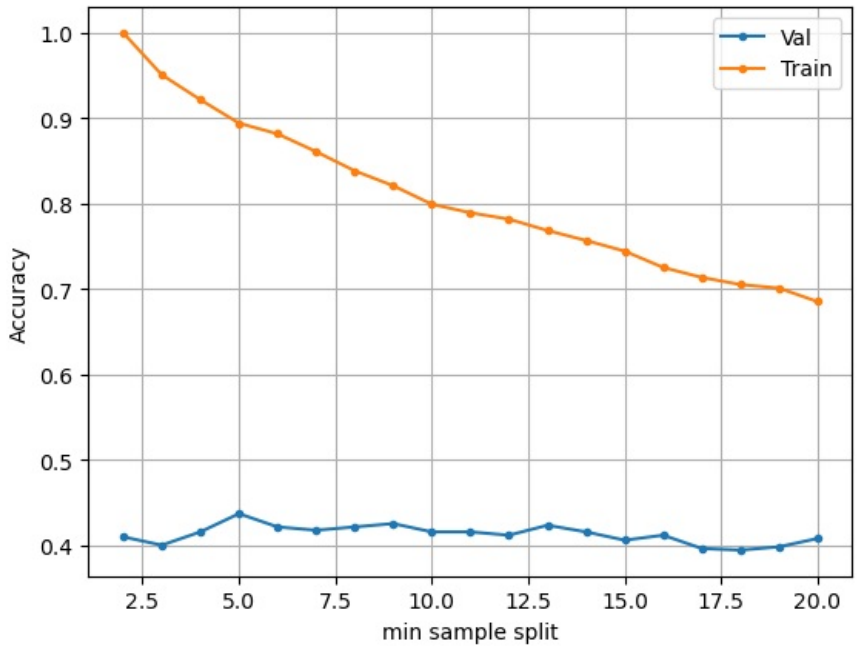
In [27]:
```
min_sample_leaf_list = np.arange(1, 50+1, 1)
accuracy_val_list = list()
accuracy_train_list = list()
for min_sample_leaf in min_sample_leaf_list:
    clf1 = DecisionTreeClassifier(
        criterion='gini',
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=min_sample_leaf,
        ccp_alpha=0.0,
        random_state=0
    )
    clf1.fit(X_train, y_train)

    y_pred = clf1.predict(X_val)
    accuracy_val_list.append(accuracy_score(y_val, y_pred))

    y_pred_train = clf1.predict(X_train)
    accuracy_train_list.append(accuracy_score(y_train, y_pred_train))

plt.plot(min_sample_leaf_list, accuracy_val_list, label='Val', marker='.')
plt.plot(min_sample_leaf_list, accuracy_train_list, label='Train', marker='.')
plt.ylabel('Accuracy')
plt.xlabel('min sample leaf')
plt.grid()
plt.legend()
plt.show()
```




Si ha overfitting, il valore maggiore di accuracy sul validation set si ha per min sample leaf = 1.

In [182…

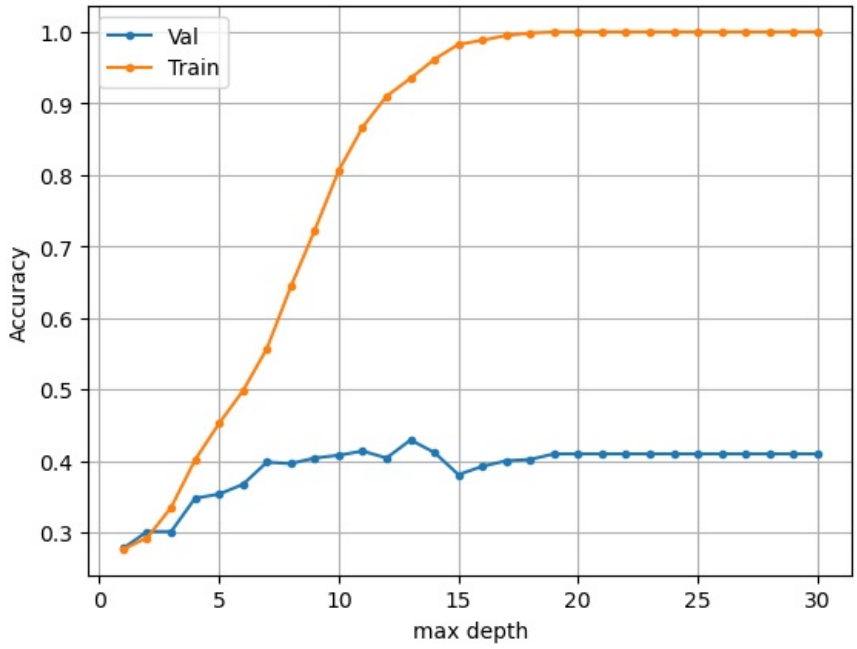
```
min_sample_split_list = np.arange(2, 20+1, 1)
min_sample_split_list

accuracy_val_list = list()
accuracy_train_list = list()
for min_sample_split in min_sample_split_list:
    clf2 = DecisionTreeClassifier(
        criterion='gini',
        max_depth=None,
        min_samples_split=min_sample_split,
        min_samples_leaf=1,
        ccp_alpha=0.0,
        random_state=0
    )
    clf2.fit(X_train, y_train)

    y_pred = clf2.predict(X_val)
    accuracy_val_list.append(accuracy_score(y_val, y_pred))

    y_pred_train = clf2.predict(X_train)
    accuracy_train_list.append(accuracy_score(y_train, y_pred_train))

plt.plot(min_sample_split_list, accuracy_val_list, label='Val', marker='.')
plt.plot(min_sample_split_list, accuracy_train_list, label='Train', marker='.')
plt.ylabel('Accuracy')
plt.xlabel('min sample split')
plt.grid()
plt.legend()
plt.show()
```




Si ha overfitting, il valore maggiore di accuracy sul test set si ha per min sample split = 5.

In [183…

```
max_depth_list = np.arange(1, 30+1, 1).tolist() + [None]

accuracy_val_list = list()
accuracy_train_list = list()
for max_depth in max_depth_list:
    clf3 = DecisionTreeClassifier(
        criterion='gini',
        max_depth=max_depth,
        min_samples_split=2,
        min_samples_leaf=1,
        ccp_alpha=0.0,
        random_state=0
    )
    clf3.fit(X_train, y_train)

    y_pred = clf3.predict(X_val)
    accuracy_val_list.append(accuracy_score(y_val, y_pred))

    y_pred_train = clf3.predict(X_train)
    accuracy_train_list.append(accuracy_score(y_train, y_pred_train))

plt.plot(max_depth_list, accuracy_val_list, label='Val', marker='.')
plt.plot(max_depth_list, accuracy_train_list, label='Train', marker='.')
plt.ylabel('Accuracy')
plt.xlabel('max depth')
plt.grid()
plt.legend()
plt.show()
```



Per max depth = 2 l'accuratezza sul validation set è leggermente maggiore a quella sul train set, il valore migliore per max depth è 13.

## Decision tree con parametri ottimizzati

In [226…
```
clf = DecisionTreeClassifier(
    criterion='gini',
    max_depth=13,
    min_samples_split=5,
    min_samples_leaf=1,
    ccp_alpha=0.0,
    random_state=0
)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)
accuracy_score(y_val, y_pred)
```

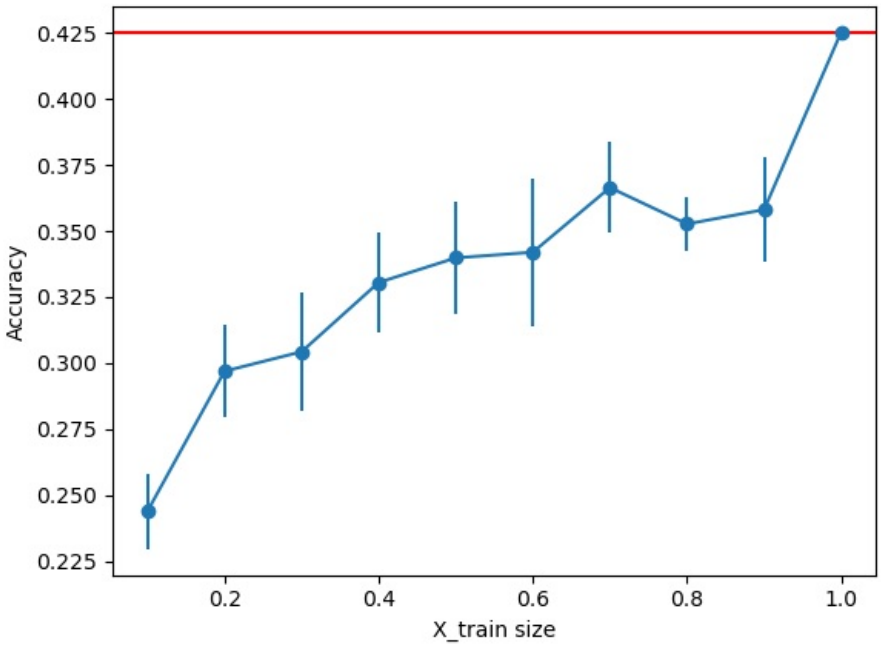
Out[226…
```
0.42524271844660194
```

L'accuracy sul validation set è leggermente migliore del modello con i parametri default.

## Learning curve

In [232…
```
accuracy_mean_list = list()
accuracy_std_list = list()
for p in np.arange(0.1, 1.0, 0.1):
    accuracy_list_p = list()
    for i in range(0, 10):
        index = np.random.choice(np.arange(0, len(X_train)), int(len(X_train) * p), replace=False)
        # print(p, i, len(index), index[:5])
        clf = DecisionTreeClassifier(
            criterion='gini',
            max_depth=13,
            min_samples_split=5,
            min_samples_leaf=1,
            ccp_alpha=0.0,
            random_state=0
        )
        clf.fit(X_train[index], y_train[index])

        y_pred = clf.predict(X_val)
        accuracy_list_p.append(accuracy_score(y_val, y_pred))
    #print(p, len(index), accuracy_list_p)
    accuracy_mean_list.append(np.mean(accuracy_list_p))
    accuracy_std_list.append(np.std(accuracy_list_p))
```

In [233…
```
accuracy_mean_list.append(0.42524271844660194)
accuracy_std_list.append(0.0)

plt.errorbar(x=np.arange(0.1, 1.1, 0.1), y=accuracy_mean_list,
             yerr=accuracy_std_list, marker='o')
plt.axhline(y=0.42524271844660194, color='r')
plt.ylabel('Accuracy')
plt.xlabel('X_train size')
plt.show()
```




In generale, l'accuratezza migliora aumentando X_train. Con un valore di X_train pari al 70% si ha un'accuracy di circa l'85%.

## Randomized search

Si esegue una randomized search per determinare i parametri migliori limitando le feature a quelle che hanno mostrato avere la maggiore importanza nella classificazione.

In [234…
```
target = 'emotion'
columns = ['length_ms','zero_crossings_sum','max','kur','sc_skew','mfcc_max','stft_min']
```

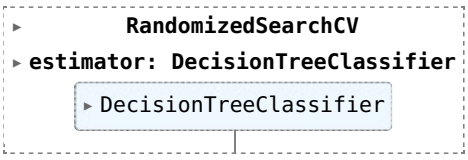
In [235…
```
X = df[columns].values
y = df[target].values
```

In [236…
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
```

In [237…
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.3, stratify=y_train_val
```

In [242…
```
param_dict4 = {
    'max_depth': [1,2,3,4, 6, 8,10,12,16,20,24, None],
    'min_samples_split': [2, 4,6, 8, 10,12, 16, 20, 24,28,32],
    'min_samples_leaf': [1, 2,4,6, 8, 10,12, 16, 20, 24,28,32],
    'ccp_alpha': np.arange(0.0, 0.1, 0.01)
}
```

In [244…
```
clf_rand = DecisionTreeClassifier(
    criterion='gini',
    max_depth=max_depth,
    min_samples_split=2,
    min_samples_leaf=1,
    ccp_alpha=0.0,
    random_state=0
)
rands = RandomizedSearchCV(clf_rand, param_dict4, cv=5, scoring='accuracy', refit=True, n_iter=1000)
rands.fit(X_train_val, y_train_val)
```

Out[244…



In [245…
```
rands.best_params_
```

Out[245…
```
{'min_samples_split': 20,
 'min_samples_leaf': 24,
 'max_depth': 20,
 'ccp_alpha': 0.0}
```

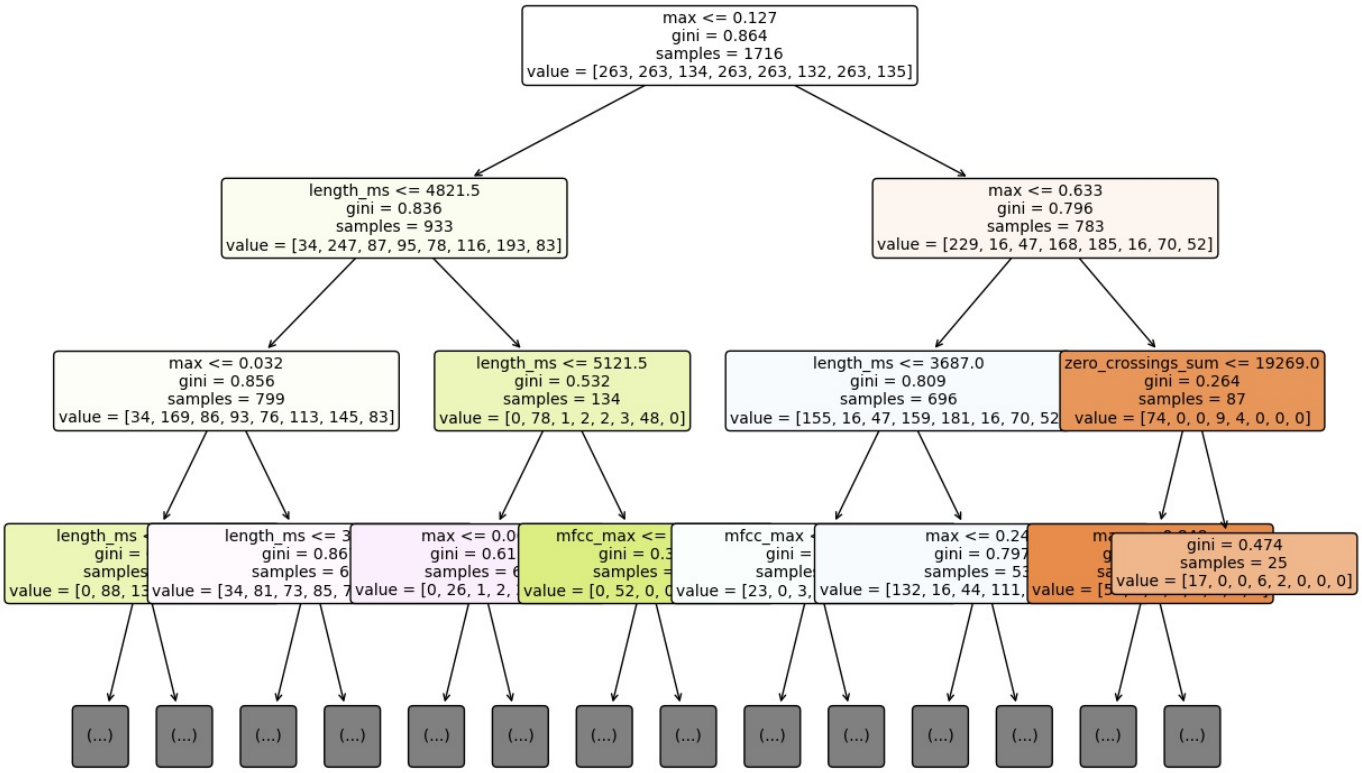
In [246…
```
clf = rands.best_estimator_

y_pred = clf.predict(X_test)
```

```
accuracy_score(y_test, y_pred)
```

Out[246… 0.40217391304347827

Con i parametri ottenuti dalla randomized search, l'accuracy sul test set risulta essere 0.40

In [196…
```
plt.figure(figsize=(14, 10))
plot_tree(clf,
          feature_names=columns,
          #class_names=clf.classes_,
          filled=True,
          rounded=True,
          fontsize=10,
          max_depth=3
          )
plt.show()
```




Dal decision tree, si osserva che le prime feature ad essere usata per la classificazione sono 'max', 'length_ms', 'zero_crossings_sum' e 'mfcc_mean'. Il primo split divide i record in modo piuttosto omogeneo, mentre i successivi producono una foglia con un numero elevato di campioni e un'altra con pochi record.

Inoltre, è possibile vedere che a profondità 3 si ottiene una foglia che non viene ulteriormente divisa in base ai parametri impostati e che è principalmente costituita da record con emotion 'angry'.

Performance evaluation

In [197…
```
y_train_pred = clf.predict(X_train)
print(accuracy_score(y_train, y_train_pred))
```

0.5045795170691091

In [199…
```
y_test_pred = clf.predict(X_test)
print(accuracy_score(y_test, y_test_pred))
print(f1_score(y_test, y_test_pred, average="macro"))
```

0.41847826086956524
0.40138735887379406

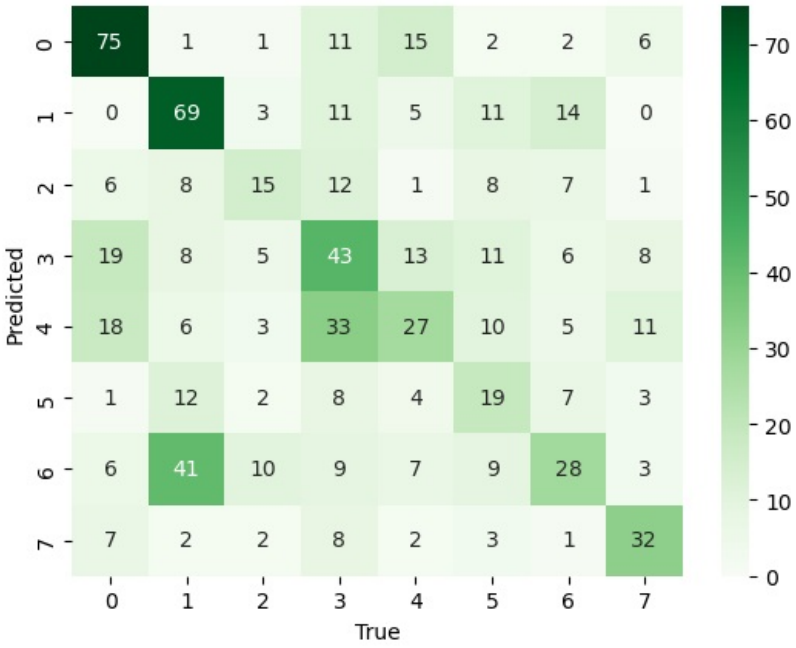
Si verifica un overfitting in quanto l'accuracy sul train set è maggiore a quella sul test set

In [200…
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

       angry       0.57      0.66      0.61       113
        calm       0.47      0.61      0.53       113
     disgust       0.37      0.26      0.30        58
     fearful       0.32      0.38      0.35       113
       happy       0.36      0.24      0.29       113
     neutral       0.26      0.34      0.29        56
         sad       0.40      0.25      0.31       113
   surprised       0.50      0.56      0.53        57

    accuracy                           0.42       736
   macro avg       0.41      0.41      0.40       736
weighted avg       0.41      0.42      0.41       736
```

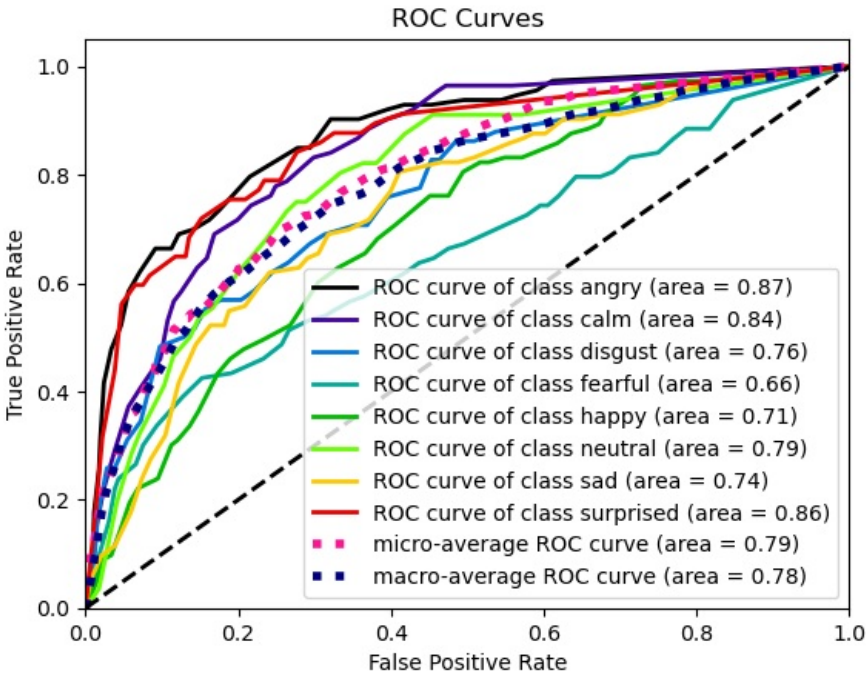
L'emozione 'angry' ha il valore più alto di precision e recall, seguita da 'surprised'. A differenza del knn, 'sad' ha una precision maggiore della recall. La precision in generale ha una distribuzione meno variabile della recall.

In [201…
```
cf = confusion_matrix(y_test, y_test_pred)
sns.heatmap(cf, annot=True, cmap="Greens")
plt.xlabel("True")
plt.ylabel("Predicted")
plt.show()
```




In [202…
```
y_test_pred_proba = clf.predict_proba(X_test)
```

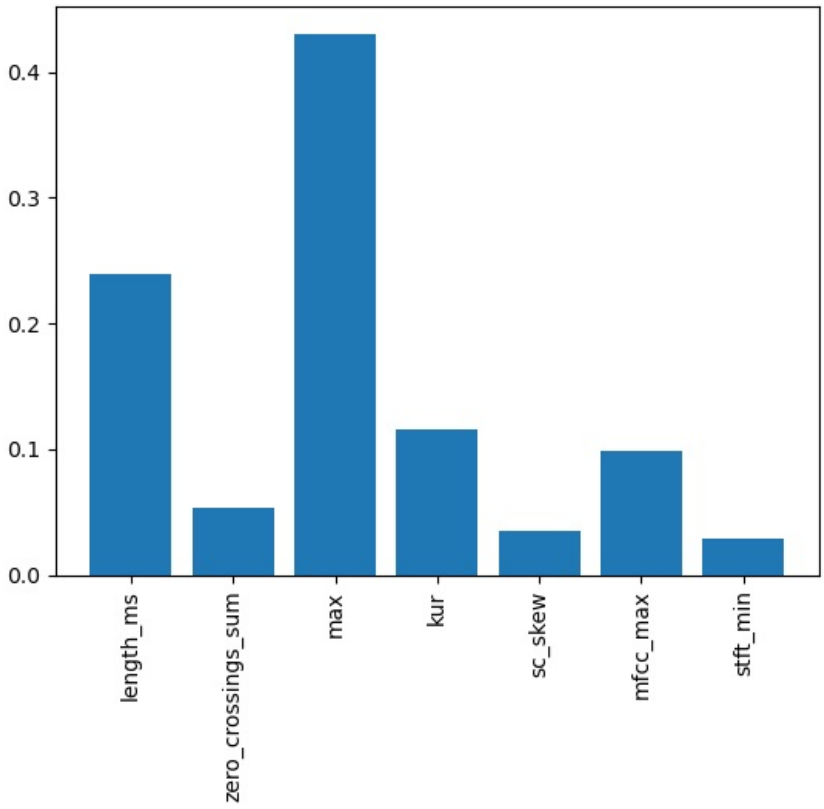
In [203…
```
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
plot_roc(y_test, clf.predict_proba(X_test))
plt.show()
print(roc_auc_score(y_test, y_test_pred_proba, multi_class="ovr", average="macro"))
```




```
0.7791533054397486
```

Dalla confusion matrix e dalla roc curve si nota che, equivalentemente all'algoritmo knn, 'angry' è l'emozione classificata in modo più accurato mentre, a differenza del knn, 'fearful' ha l'area minore nel grafico.

In [205…
```
plt.bar(columns, clf.feature_importances_)
plt.xticks(rotation=90)
plt.show()
```




La feature importance maggiore è attribuita alla variabile max, infatti come osservato nella fase di data understanding, le emozioni assumono valori di max differenti. La seconda variabile per importanza è la lunghezza della registrazione audio, a indicare che pronunciare le frasi interpretando emozioni differenti risulta in una durata maggiore o minore del file audio.

# SVM

In [82]:
```
from sklearn.svm import LinearSVC, SVC
```

In [274…
```
target = 'emotion'
columns = ['length_ms','zero_crossings_sum','max','kur','sc_skew','mfcc_max','stft_min']
```

In [275…
```
X = df[columns].values
y = df[target].values
```

In [276…
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [277…
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
```

In [278…
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.3, stratify=y_train_val
```

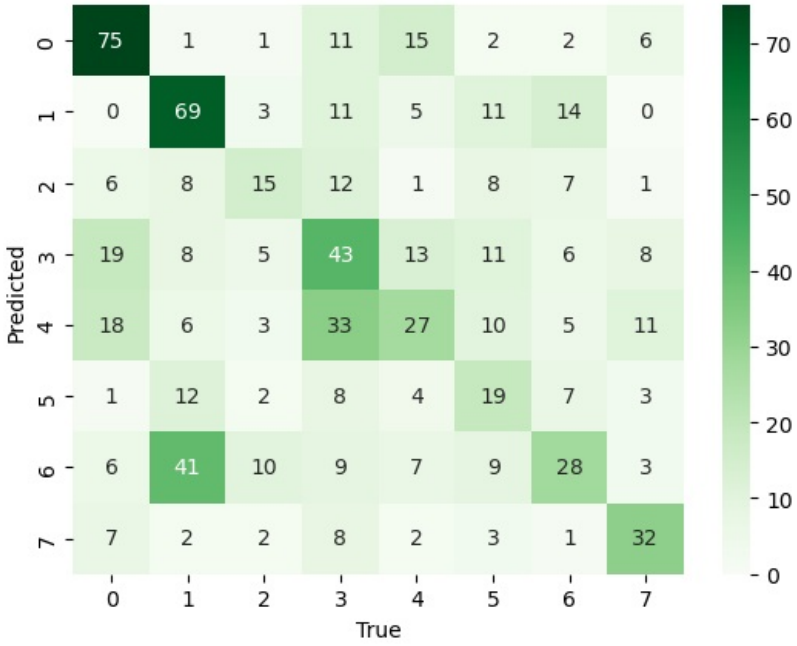
In [ ]:
```
param_dict = {
    'C': [0, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 100, 1000],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'degree': [1, 2, 3, 4, 5],
    'gamma': ['scale', 'auto'],
    'coef0': [0.0, 0.1, 0.5, 1, 10],
}

clf = SVC()

rands = RandomizedSearchCV(clf, param_dict, cv=5, scoring='accuracy', refit=True, n_iter=100)
rands.fit(X_train_val, y_train_val)

print(rands.best_params_)

clf = rands.best_estimator_

clf.fit(X_train_val, y_train_val)

y_pred = clf.predict(X_test)

accuracy_score(y_test, y_pred)
```

La randomized search effettuata indica che i parametri migliori sono: kernel function Radial basis, con C pari a 10 e gamma 'scale'. Ciò fornisce un'accuracy sul test set di 0.45.

In [279…
```
svm = SVC(C=10, kernel='rbf', gamma='scale',probability=True)
svm.fit(X_train_val, y_train_val)
y_pred = svm.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[279…
```
0.44972826086956524
```

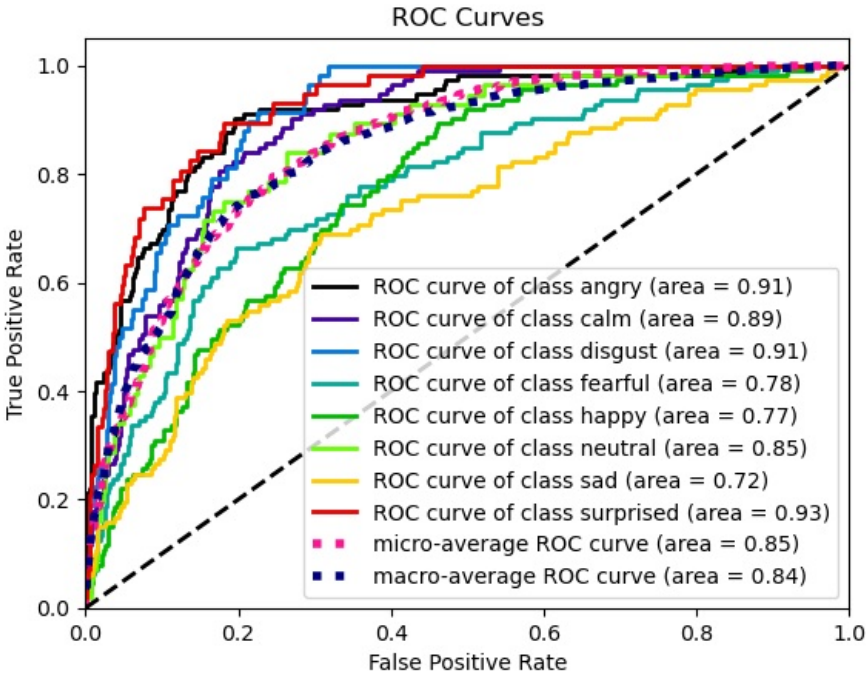
In [270…
```
svm = confusion_matrix(y_test, y_test_pred)
sns.heatmap(svm, annot=True, cmap="Greens")
plt.xlabel("True")
plt.ylabel("Predicted")
plt.show()
```




I valori dei True Positive sono piuttosto buoni, l'emozione 'angry' risulta essere quella con la classificazione più appropriata.

In [157…
```
predicted_proba = svm.predict_proba(X_test)
roc_auc = roc_auc_score(y_test, predicted_proba, multi_class='ovr')
```

In [159…
```
plot_roc(y_test, svm.predict_proba(X_test))
plt.show()
print(roc_auc_score(y_test, y_test_pred_proba, multi_class="ovr", average="macro"))
```




```
0.8337185155366571
```

La curva roc mostra un andamento simile a knn, infatti l'area sottesa alla curva delle emozione 'angry' e 'surprised' è prossima a 1.

In conclusione, il metodo svm opera una classificazione simile a knn ma con un'accuracy leggermente migliore.

# Random Forest

In [40]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=4)
```

In [41]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, stratify=y_train_val
```

In [24]:
```
len(X_train), len(X_val), len(X_test)
```

Out[24]:
```
(1568, 393, 491)
```

In [35]:
```
param_dict2 = {
    'max_depth': np.arange(1, 35+1, 1).tolist() + [None],
    'min_samples_split': np.arange(2, 30+1, 1),
    'min_samples_leaf': np.arange(1, 25+1, 1),
    'ccp_alpha': np.arange(0.0, 0.1, 0.01)
}
```

In [37]:
```
clf2 = RandomForestClassifier(
    n_estimators=20,
    criterion='gini',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    ccp_alpha=0.0,
    max_features='sqrt',
    random_state=0
)

rands = RandomizedSearchCV(clf2, param_dict2, cv=5, scoring='accuracy', refit=True, n_iter=5000)
rands.fit(X_train_val, y_train_val)

clf2 = rands.best_estimator_

clf2.fit(X_train_val, y_train_val)

y_pred = clf2.predict(X_test)
```

In [38]:
```
accuracy_score(y_test, y_pred)
```

Out[38]:
```
0.515274949083503
```

ottengo un'acuracy buona con i paramentri di clf2 indicati sotto

In [39]:
```
rands.best_params_
```

Out[39]:
```
{'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_depth': 25,
 'ccp_alpha': 0.0}
```

In [40]:
```
clf2
```

Out[40]:
```
RandomForestClassifier(max_depth=25, n_estimators=20, random_state=0)
```

clf2 è calcolato con i migliori valori ricavati da una randomize search

In [49]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [53]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
```

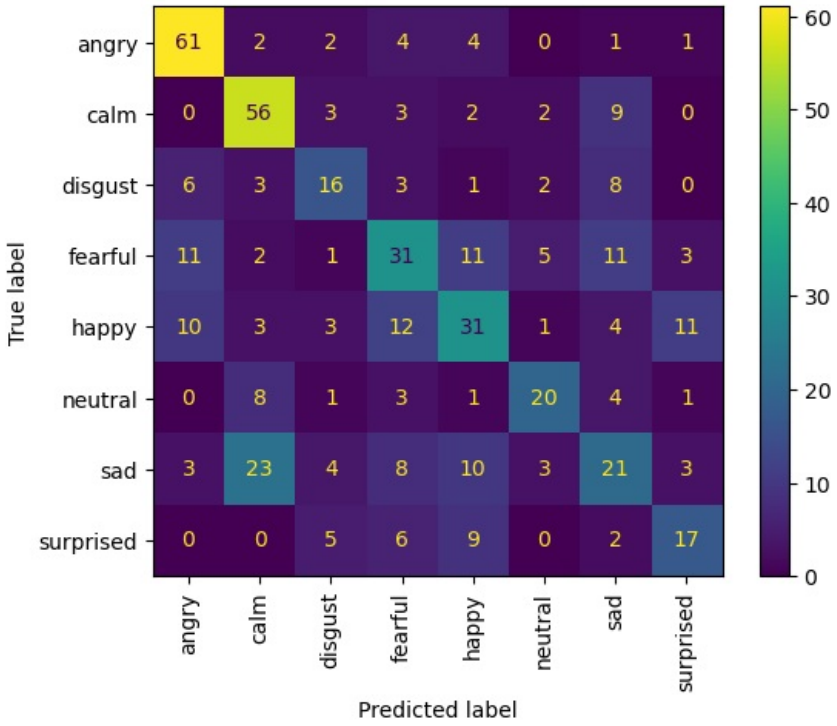
In [50]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[50]:
```
array([[61,  2,  2,  4,  4,  0,  1,  1],
       [ 0, 56,  3,  3,  2,  2,  9,  0],
       [ 6,  3, 16,  3,  1,  2,  8,  0],
       [11,  2,  1, 31, 11,  5, 11,  3],
       [10,  3,  3, 12, 31,  1,  4, 11],
       [ 0,  8,  1,  3,  1, 20,  4,  1],
       [ 3, 23,  4,  8, 10,  3, 21,  3],
       [ 0,  0,  5,  6,  9,  0,  2, 17]], dtype=int64)
```

In [51]:
```
plt.figure(figsize=(10,10))
```

```
disp_cm = ConfusionMatrixDisplay(cm,display_labels=clf2.classes_)
disp_cm.plot()
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```




dalla confusion matrix si nota sempre l'errore tra sad e calm, mentre per il resto delle emozioni sembra essere più solida

In [54]:
```
print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

       angry       0.81      0.67      0.73        91
        calm       0.75      0.58      0.65        97
     disgust       0.41      0.46      0.43        35
     fearful       0.41      0.44      0.43        70
       happy       0.41      0.45      0.43        69
     neutral       0.53      0.61      0.56        33
         sad       0.28      0.35      0.31        60
   surprised       0.44      0.47      0.45        36

    accuracy                           0.52       491
   macro avg       0.50      0.50      0.50       491
weighted avg       0.55      0.52      0.53       491
```

angry come nelle altre classificazioni risulta essere la più alta, in questo caso lo si nota ancora di più, stesso discorso per calm

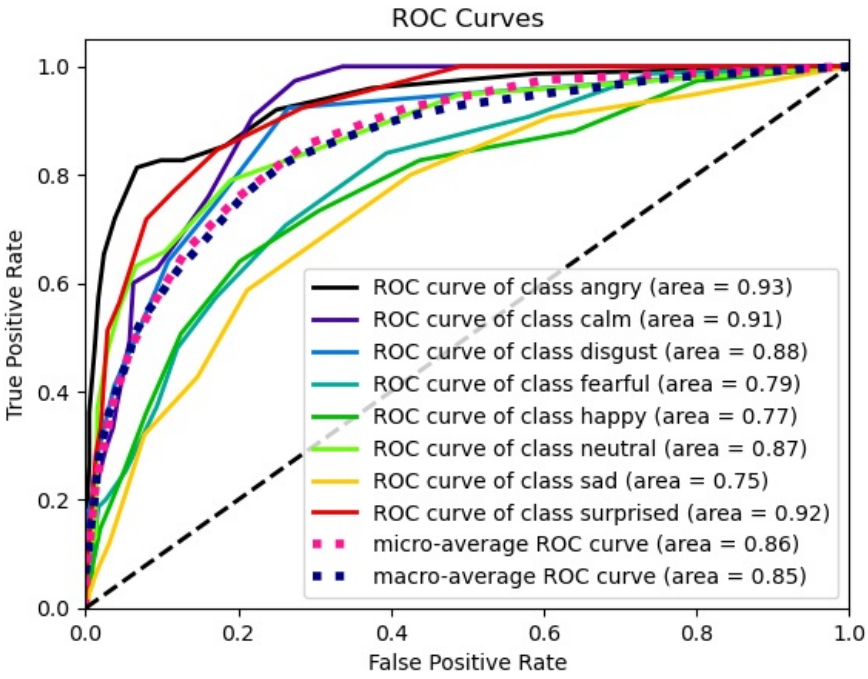
le altre mediamente migliorano ad eccezione di sad che rimane più o meno con la precisione trovata dalle precedenti classificazioni.

sad ha recall maggiore di precision come in knn, a differenza del decision tree dove avveniva il contrario

In [55]:
```
import scikitplot
```

In [57]:
```
y_pred_prob = clf2.predict_proba(X_test)
```

In [58]:
```
scikitplot.metrics.plot_roc(y_test, y_pred_prob)
plt.show()
```




le più basse risultano essere sad happy e fearful. angry calm e surprised molto alte

# CONCLUSIONI

## Clustering

La tecnica di clustering migliore risulta essere il K-means in quanto da esso risulta una silhouette maggiore (0.234) con numero di cluster totali = 6 a differenza del clustering gerarchico dove la silhouette migliore risultava essere 0.192 con numero di cluster totali = 4. Nel clustering gerarchico è stato adottato il metodo di separazione completo, delle prove con metodo di separazione ward sono state effetuate ma davano silhouette minore e cluster meno uniformi a livello di numero di popolazione. Interessante notare come nel clustering gerarchico si riescano a distinguere meglio le distribuzione della variabile vocal_channel tra i vari cluster. Questo ci suggerisce che nel caso in cui come focus di analisi sia la divisione in cluster della variabile vocal_channel allora il clustering gerarchico è da preferirsi. Il DBSCAN non raggiunge valori di silhouette soddisfacenti inoltre la divisione della popolazione all'interno dei clsuter non è numericamente omogenea.

## Classificazione

Tra le varie tipologie di classificazione adottate per la previsione della variabile emotion quella con risultati migliori risulta essere la random forest raggiungendo un'accuracy di 0.52 sul test_set, tuttavia c'è un consistente overfitting da tenere in considerazione. Il secondo classificatore per ordine di accuracy rilevata è SVM con 0.45. Ciò indica come le tecniche di ensemble risultano essere migliori per questo dataset. A seguire decision tree con 0.42 e knn con 0.40.

In generale le roc curve di tutte le tecniche di classificazione sembrano avere lo stesso andamento. Angry risulta sempre essere quella più facile da individuare, le altre variavano invece a seconda del metodo adottato, solitamente sono calm e surprised. Mentre sad risulta essere quelal più difficile da individuare.

In [ ]: