

DMML 2024: Final Project

Francesco Peria, Maria Paola Sforza Fogliani, Andrea Vitali

```
In [2]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt
```

```
In [3]: df = pd.read_csv('ravdess_features.csv', sep=',', skipinitialspace=True)  
pd.set_option('display.float_format', '{:.5f}'.format)
```

A) Data Understanding & Preparation

1. Data Semantics

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

	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	2.00000	F	1	2	...	0.00000
1	audio-only	speech	angry	normal	Dogs are sitting by the door	1st	16.00000	F	1	2	...	0.00000
2	audio-only	Nan	happy	strong	Dogs are sitting by the door	2nd	16.00000	F	1	2	...	0.00000
3	audio-only	Nan	surprised	normal	Kids are talking by the door	1st	14.00000	F	1	2	...	0.00000
4	audio-only	song	happy	strong	Dogs are sitting by the door	2nd	2.00000	F	1	2	...	0.00000

5 rows × 38 columns

```
In [5]: 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

```

Il dataset è composto da **2452 record e 38 attributi**. In esso sono descritte le caratteristiche di diversi file audio in cui alcuni attori (uomini o donne) pronunciano una di due frasi ("Kids are talking by the door", "Dogs are sitting by the door"), parlando o cantando, ed esprimendo una particolare emozione; per ogni record, sono inoltre riportate alcune statistiche acustiche (relative a: *Mel-Frequency Cepstral Coefficients*, *spectral centroid*, e *stft chromagram*).

Eploriamo più nel dettaglio la semantica delle variabili, iniziando con quelle **non numeriche**:

```
In [6]: non_numerical_columns = df.select_dtypes(exclude='number')
non_numerical_columns
```

	modality	vocal_channel	emotion	emotional_intensity	statement	repetition	sex
0	audio-only	speech	fearful	normal	Dogs are sitting by the door	2nd	F
1	audio-only	speech	angry	normal	Dogs are sitting by the door	1st	F
2	audio-only	Nan	happy	strong	Dogs are sitting by the door	2nd	F
3	audio-only	Nan	surprised	normal	Kids are talking by the door	1st	F
4	audio-only	song	happy	strong	Dogs are sitting by the door	2nd	F
...
2447	audio-only	speech	calm	strong	Kids are talking by the door	1st	M
2448	audio-only	speech	calm	normal	Dogs are sitting by the door	1st	M
2449	audio-only	song	sad	strong	Dogs are sitting by the door	2nd	M
2450	audio-only	speech	surprised	normal	Kids are talking by the door	1st	M
2451	audio-only	Nan	neutral	normal	Dogs are sitting by the door	2nd	M

2452 rows × 7 columns

Valori unici per ogni feature:

```
In [7]: for column in non_numerical_columns:
```

```
print(column, ":", non_numerical_columns[column].unique(), sep="")  
modality: ['audio-only']  
vocal_channel: ['speech' 'nan' 'song']  
emotion: ['fearful' 'angry' 'happy' 'surprised' 'neutral' 'calm' 'sad' 'disgust']  
emotional_intensity: ['normal' 'strong']  
statement: ['Dogs are sitting by the door' 'Kids are talking by the door']  
repetition: ['2nd' '1st']  
sex: ['F' 'M']
```

In [8]: `non_numerical_columns.nunique()`

Out[8]:

modality	1
vocal_channel	2
emotion	8
emotional_intensity	2
statement	2
repetition	2
sex	2
dtype:	int64

Vediamo che la maggior parte delle variabili categoriche sono **binarie**; tra queste, alcune (e.g. 'sex') sono **simmetriche**, e altre (e.g. 'emotional_intensity') **ordinali**. La variabile 'modality' ha un unico valore in tutto il dataset, mentre 'vocal_channel' presenta dei valori mancanti; entrambe queste caratteristiche saranno trattate in fase di data preparation.

Ad eccezione di 'modality', le variabili categoriche – e soprattutto quelle binarie – sono ben **bilanciate** (c'è la stessa proporzione tra i due statement, 'M' ed 'F' sono quasi egualmente distribuiti, etc.):

In [9]: `for column in non_numerical_columns:
 print(non_numerical_columns[column].value_counts())
 print("\n")`

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

sex
M 1248
F 1204
Name: count, dtype: int64

Analizziamo ora le variabili **numeriche**:

In [10]: `numerical_columns = df.select_dtypes(include='number')`

numerical_columns

	actor	channels	sample_width	frame_rate	frame_width	length_ms	frame_count	intensity	zero_crossings_sum	mfcc_
0	2.00000	1	2	48000	2	3737	179379.00000	-36.79343	16995	-33.
1	16.00000	1	2	48000	2	3904	187387.00000	NaN	13906	-29.
2	16.00000	1	2	48000	2	4671	224224.00000	-32.29074	18723	-30.
3	14.00000	1	2	48000	2	3637	174575.00000	-49.01984	11617	-36.
4	2.00000	1	2	48000	2	4404	211411.00000	-31.21450	15137	-31.
...
2447	23.00000	1	2	48000	2	4605	221021.00000	NaN	9871	-30.
2448	23.00000	1	2	48000	2	4171	200200.00000	-43.34290	8963	-31.
2449	23.00000	1	2	48000	2	5239	251451.00000	NaN	9765	-26.
2450	NaN	1	2	48000	2	3737	179379.00000	-45.75126	9716	-28.
2451	23.00000	1	2	48000	2	3837	184184.00000	-40.01804	9427	-29.

2452 rows × 31 columns

Valori unici:

In [11]: numerical_columns.nunique()

```
Out[11]: actor          24
channels        2
sample_width    1
frame_rate      1
frame_width     2
length_ms       95
frame_count     158
intensity       989
zero_crossings_sum  2176
mfcc_mean       2451
mfcc_std        2449
mfcc_min        2451
mfcc_max        2449
sc_mean         2451
sc_std          2451
sc_min          1431
sc_max          2423
sc_kur          2451
sc_skew         2451
stft_mean       2451
stft_std        2451
stft_min        1431
stft_max        1
stft_kur        2451
stft_skew       2451
mean            2450
std             2451
min             2148
max             2166
kur             2451
skew            2451
dtype: int64
```

Conteggio dei valori:

In [12]:

```
for column in numerical_columns:
    print(numerical_columns[column].value_counts())
    print("\n")
```

actor	
22.00000	65
12.00000	63
14.00000	62
20.00000	61
8.00000	61
19.00000	60
13.00000	60
2.00000	58
16.00000	58
24.00000	58
5.00000	58
10.00000	56
21.00000	55

```
11.00000 55
6.00000 55
17.00000 55
4.00000 52
3.00000 51
1.00000 51
7.00000 51
23.00000 51
9.00000 51
15.00000 44
18.00000 35
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
168168.00000 65
179379.00000 62
176176.00000 58
171371.00000 57
184184.00000 54
..
296296.00000 1
145745.00000 1
142542.00000 1
145746.00000 1
278679.00000 1
Name: count, Length: 158, dtype: int64
```

```
intensity
-47.38644 7
-46.78717 7
-45.01264 7
-46.17248 7
-36.52282 7
..
-29.91007 1
-29.13289 1
-34.55979 1
-45.64908 1
-29.51279 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
-27.67493    2
-33.48595    1
-32.75184    1
-30.87260    1
-30.88487    1
...
-29.02738    1
-30.78250    1
-30.98836    1
-32.05165    1
-29.01924    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
4923.91703    2
5792.55074    1
5279.16305    1
5898.30658    1
4223.68350    1
...
4889.58252    1
6715.37764    1
4612.97540    1
```

```
5575.58672    1  
6082.67612    1  
Name: count, Length: 2451, dtype: int64
```

```
sc_std  
2699.05623    2  
3328.05546    1  
3801.31582    1  
3706.57671    1  
3081.61776    1  
...  
3942.52893    1  
4279.15119    1  
3767.33907    1  
3841.91348    1  
3963.72512    1  
Name: count, Length: 2451, dtype: int64
```

```
sc_min  
0.00000      1021  
929.40279     2  
988.01203     1  
1196.10746     1  
1612.97297     1  
...  
1031.73371     1  
1218.76376     1  
996.95054     1  
901.73863     1  
760.82255     1  
Name: count, Length: 1431, dtype: int64
```

```
sc_max  
12000.00012    3  
12000.00016    3  
12000.00022    3  
12000.00003    3  
11999.99990    3  
...  
9658.84719     1  
11124.09875     1  
12000.00002     1  
9040.81609     1  
12199.77342     1  
Name: count, Length: 2423, dtype: int64
```

```
sc_kur  
-1.18352      2  
-1.12077      1  
-1.44495      1  
-1.55695      1  
-0.79018      1  
...  
-1.48638      1  
-1.74124      1  
-1.38422      1  
-1.38026      1  
-1.50139      1  
Name: count, Length: 2451, dtype: int64
```

```
sc_skew  
0.13669      2  
0.25094      1  
0.51984      1  
0.16814      1  
0.59820      1  
...  
0.49911      1  
-0.24263      1  
0.29682      1  
0.17207      1  
0.14757      1  
Name: count, Length: 2451, dtype: int64
```

```
stft_mean  
0.53313      2  
0.41525      1
```

```
0.43085    1  
0.49617    1  
0.34001    1  
..  
0.45694    1  
0.63298    1  
0.43676    1  
0.52262    1  
0.55495    1  
Name: count, Length: 2451, dtype: int64
```

```
stft_std  
0.29877    2  
0.33553    1  
0.35921    1  
0.31962    1  
0.35216    1  
..  
0.35636    1  
0.29868    1  
0.35303    1  
0.31691    1  
0.32079    1  
Name: count, Length: 2451, dtype: int64
```

```
stft_min  
0.00000    1021  
0.00250    2  
0.00335    1  
0.00024    1  
0.00070    1  
..  
0.00399    1  
0.00301    1  
0.00319    1  
0.00463    1  
0.00156    1  
Name: count, Length: 1431, dtype: int64
```

```
stft_max  
1.00000    2452  
Name: count, dtype: int64
```

```
stft_kur  
-1.16946    2  
-1.21502    1  
-1.47879    1  
-1.33480    1  
-1.07067    1  
..  
-1.50661    1  
-0.99726    1  
-1.42076    1  
-1.25520    1  
-1.25767    1  
Name: count, Length: 2451, dtype: int64
```

```
stft_skew  
-0.00762    2  
0.40351    1  
0.22509    1  
0.06777    1  
0.63525    1  
..  
0.15323    1  
-0.55023    1  
0.15074    1  
-0.07033    1  
-0.23776    1  
Name: count, Length: 2451, dtype: int64
```

```
mean  
-0.00000    2  
0.00000    2  
0.00000    1  
0.00000    1  
0.00002    1
```

```
..  
-0.00000    1  
0.00000    1  
0.00000    1  
-0.00000    1  
0.00000    1  
Name: count, Length: 2450, dtype: int64
```

```
std  
0.00679    2  
0.01448    1  
0.02351    1  
0.00970    1  
0.01774    1  
..  
0.00790    1  
0.00311    1  
0.00446    1  
0.00286    1  
0.01000    1  
Name: count, Length: 2451, dtype: int64
```

```
min  
-0.11392    4  
-0.06915    4  
-0.99881    3  
-0.10233    3  
-0.02753    3  
..  
-0.07480    1  
-0.06235    1  
-0.02402    1  
-0.21024    1  
-0.08151    1  
Name: count, Length: 2148, dtype: int64
```

```
max  
0.99881    14  
0.05518    4  
0.06226    4  
0.05478    3  
0.05307    3  
..  
0.14847    1  
0.18530    1  
0.39636    1  
0.87207    1  
0.10303    1  
Name: count, Length: 2166, dtype: int64
```

```
kur  
9.42733    2  
9.40606    1  
6.52938    1  
21.93122   1  
4.29562    1  
..  
2.94969    1  
12.94743   1  
8.55983    1  
15.01436   1  
12.97318   1  
Name: count, Length: 2451, dtype: int64
```

```
skew  
0.38833    2  
0.27315    1  
0.49945    1  
0.47879    1  
0.04838    1  
..  
0.01393    1  
-0.08540   1  
0.04711    1  
0.00921    1  
1.03208    1  
Name: count, Length: 2451, dtype: int64
```

Statistiche descrittive preliminari sulle colonne numeriche (le riprenderemo più nel dettaglio nella prossima sezione):

In [13]: `numerical_columns.describe()`

	actor	channels	sample_width	frame_rate	frame_width	length_ms	frame_count	intensity	zero_crossings_sur
count	1326.00000	2452.00000	2452.00000	2452.00000	2452.00000	2452.00000	2452.00000	1636.00000	2452.00000
mean	12.58220	1.00245	2.00000	48000.00000	2.00489	4092.15131	193587.18801	-37.62533	12885.3140
std	6.91624	0.04942	0.00000	0.00000	0.09883	598.32153	36825.36906	8.45198	3665.3195
min	1.00000	1.00000	2.00000	48000.00000	2.00000	2936.00000	-1.00000	-63.86461	4721.0000
25%	7.00000	1.00000	2.00000	48000.00000	2.00000	3604.00000	172972.00000	-43.53987	10362.5000
50%	13.00000	1.00000	2.00000	48000.00000	2.00000	4004.00000	190591.00000	-37.07274	12383.5000
75%	19.00000	1.00000	2.00000	48000.00000	2.00000	4538.00000	217817.00000	-31.59131	14966.0000
max	24.00000	2.00000	2.00000	48000.00000	4.00000	6373.00000	305906.00000	-16.35395	30153.0000

8 rows × 31 columns

Alcune osservazioni:

- la variabile '**actor**' non è da considerarsi continua, bensì un identificativo; non è quindi utile ai fini dell'analisi (vedremo, inoltre, che presenta valori mancanti);
- '**sample_width**', '**frame_rate**' e '**stft_max**' non variano mai in tutto il dataset;
- '**mean**' ha una varianza prossima allo 0;
- '**channels**' e '**frame_width**' hanno 2446 record con un certo valore, e 6 con un altro; possiamo inoltre verificare che si tratta degli stessi:

In [14]: `df[(df['channels'] == 2) & (df['frame_width'] == 4)]`

	modality	vocal_channel	emotion	emotional_intensity	statement	repetition	actor	sex	channels	sample_width	...	stf
287	audio-only	speech	fearful	normal	Kids are talking by the door	2nd	NaN	F	2	2	...	0.0
778	audio-only	speech	calm	normal	Kids are talking by the door	2nd	1.00000	M	2	2	...	0.0
1045	audio-only	speech	surprised	normal	Dogs are sitting by the door	2nd	1.00000	M	2	2	...	0.0
1336	audio-only	song	neutral	normal	Kids are talking by the door	1st	24.00000	F	2	2	...	0.0
1348	audio-only	speech	happy	normal	Dogs are sitting by the door	1st	20.00000	F	2	2	...	0.0
1809	audio-only	speech	calm	normal	Dogs are sitting by the door	2nd	5.00000	M	2	2	...	0.0

6 rows × 38 columns

Tutte queste variabili potranno quindi essere **eliminate** in fase di data preparation. Vediamo inoltre che gli altri attributi hanno scale differenti; per compararli sarà perciò necessaria una normalizzazione.

2. Distribution of the variables and statistics

Statistiche descrittive sul dataset – in particolare: **media**, **mediana**, **deviazione standard**, **moda**:

In [15]: `df.describe()`

Out[15]:

	actor	channels	sample_width	frame_rate	frame_width	length_ms	frame_count	intensity	zero_crossings_sum
count	1326.00000	2452.00000	2452.00000	2452.00000	2452.00000	2452.00000	2452.00000	1636.00000	2452.0000
mean	12.58220	1.00245	2.00000	48000.00000	2.00489	4092.15131	193587.18801	-37.62533	12885.3140
std	6.91624	0.04942	0.00000	0.00000	0.09883	598.32153	36825.36906	8.45198	3665.3195
min	1.00000	1.00000	2.00000	48000.00000	2.00000	2936.00000	-1.00000	-63.86461	4721.0000
25%	7.00000	1.00000	2.00000	48000.00000	2.00000	3604.00000	172972.00000	-43.53987	10362.5000
50%	13.00000	1.00000	2.00000	48000.00000	2.00000	4004.00000	190591.00000	-37.07274	12383.5000
75%	19.00000	1.00000	2.00000	48000.00000	2.00000	4538.00000	217817.00000	-31.59131	14966.0000
max	24.00000	2.00000	2.00000	48000.00000	4.00000	6373.00000	305906.00000	-16.35395	30153.0000

8 rows × 31 columns

Media:

In [16]: `df.mean(numeric_only=True)`

```
Out[16]: actor           12.58220
channels          1.00245
sample_width      2.00000
frame_rate        48000.00000
frame_width       2.00489
length_ms         4092.15131
frame_count       193587.18801
intensity        -37.62533
zero_crossings_sum 12885.31403
mfcc_mean         -28.76918
mfcc_std          136.77723
mfcc_min          -758.90938
mfcc_max          199.18251
sc_mean           5170.10140
sc_std             3365.45339
sc_min             551.83412
sc_max             11830.46186
sc_kur             -1.14264
sc_skew            0.34844
stft_mean          0.47585
stft_std           0.33137
stft_min           0.00227
stft_max           1.00000
stft_kur           -1.24793
stft_skew          0.11289
mean               0.00000
std                0.02050
min                -0.16487
max                0.17984
kur                11.20300
skew               -0.04825
dtype: float64
```

Mediana:

In [17]: `df.median(numeric_only=True)`

```
Out[17]: actor           13.00000
channels          1.00000
sample_width      2.00000
frame_rate        48000.00000
frame_width       2.00000
length_ms         4004.00000
frame_count       190591.00000
intensity         -37.07274
zero_crossings_sum 12383.50000
mfcc_mean         -28.68111
mfcc_std          136.52381
mfcc_min          -760.98307
mfcc_max          201.69718
sc_mean           5122.71226
sc_std             3433.83537
sc_min             707.31926
sc_max             12000.29265
sc_kur             -1.30894
sc_skew            0.34762
stft_mean          0.47574
stft_std           0.33422
stft_min           0.00019
stft_max           1.00000
stft_kur           -1.29211
stft_skew          0.12606
mean               -0.00000
std                0.01388
min                -0.10378
max                0.10973
kur                9.82869
skew               0.00426
dtype: float64
```

Deviazione standard:

```
In [18]: df.std(numeric_only=True)
```

```
Out[18]: actor           6.91624
channels          0.04942
sample_width      0.00000
frame_rate        0.00000
frame_width       0.09883
length_ms         598.32153
frame_count       36825.36906
intensity         8.45198
zero_crossings_sum 3665.31958
mfcc_mean         4.46189
mfcc_std          20.45169
mfcc_min          99.94545
mfcc_max          26.00211
sc_mean           875.18544
sc_std             580.47903
sc_min             508.02589
sc_max             1004.95598
sc_kur             0.57265
sc_skew            0.35301
stft_mean          0.08255
stft_std           0.02377
stft_min           0.00483
stft_max           0.00000
stft_kur           0.21178
stft_skew          0.33076
mean               0.00004
std                0.02102
min                0.17544
max                0.19554
kur                6.61486
skew               0.45492
dtype: float64
```

La moda ha al massimo 5 parimeriti:

```
In [19]: df.mode(numeric_only=True).iloc[:10]
```

Out[19]:	actor	channels	sample_width	frame_rate	frame_width	length_ms	frame_count	intensity	zero_crossings_sum	mfcc_n
0	22.00000	1.00000	2.00000	48000.00000	2.00000	3737.00000	168168.00000	-47.38644	10667.00000	-27.6
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-46.78717	12913.00000	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-46.17248		NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-45.01264		NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-36.52282		NaN

5 rows × 31 columns



Conteggio dei parimeriti:

```
In [20]: df.mode(numeric_only=True).count()
```

```
Out[20]: actor           1
channels          1
sample_width      1
frame_rate        1
frame_width       1
length_ms         1
frame_count       1
intensity         5
zero_crossings_sum  2
mfcc_mean         1
mfcc_std          3
mfcc_min          1
mfcc_max          3
sc_mean           1
sc_std            1
sc_min            1
sc_max            5
sc_kur             1
sc_skew            1
stft_mean         1
stft_std          1
stft_min          1
stft_max          1
stft_kur          1
stft_skew          1
mean              2
std               1
min              2
max              1
kur              1
skew              1
dtype: int64
```

Visualizzazione più compatta delle **statistiche descrittive**:

```
In [21]: stat_df = pd.DataFrame({'Mean': df.mean(numeric_only=True),
                               'Median': df.median(numeric_only=True),
                               'Mode': df.mode(numeric_only=True).iloc[0],
                               'Standard Deviation': df.std(numeric_only=True)})
```

stat_df

Out[21]:

	Mean	Median	Mode	Standard Deviation
actor	12.58220	13.00000	22.00000	6.91624
channels	1.00245	1.00000	1.00000	0.04942
sample_width	2.00000	2.00000	2.00000	0.00000
frame_rate	48000.00000	48000.00000	48000.00000	0.00000
frame_width	2.00489	2.00000	2.00000	0.09883
length_ms	4092.15131	4004.00000	3737.00000	598.32153
frame_count	193587.18801	190591.00000	168168.00000	36825.36906
intensity	-37.62533	-37.07274	-47.38644	8.45198
zero_crossings_sum	12885.31403	12383.50000	10667.00000	3665.31958
mfcc_mean	-28.76918	-28.68111	-27.67493	4.46189
mfcc_std	136.77723	136.52381	138.63795	20.45169
mfcc_min	-758.90938	-760.98307	-844.52500	99.94545
mfcc_max	199.18251	201.69718	182.00595	26.00211
sc_mean	5170.10140	5122.71226	4923.91703	875.18544
sc_std	3365.45339	3433.83537	2699.05623	580.47903
sc_min	551.83412	707.31926	0.00000	508.02589
sc_max	11830.46186	12000.29265	11999.99990	1004.95598
sc_kur	-1.14264	-1.30894	-1.18352	0.57265
sc_skew	0.34844	0.34762	0.13669	0.35301
stft_mean	0.47585	0.47574	0.53313	0.08255
stft_std	0.33137	0.33422	0.29877	0.02377
stft_min	0.00227	0.00019	0.00000	0.00483
stft_max	1.00000	1.00000	1.00000	0.00000
stft_kur	-1.24793	-1.29211	-1.16946	0.21178
stft_skew	0.11289	0.12606	-0.00762	0.33076
mean	0.00000	-0.00000	-0.00000	0.00004
std	0.02050	0.01388	0.00679	0.02102
min	-0.16487	-0.10378	-0.11392	0.17544
max	0.17984	0.10973	0.99881	0.19554
kur	11.20300	9.82869	9.42733	6.61486
skew	-0.04825	0.00426	0.38833	0.45492

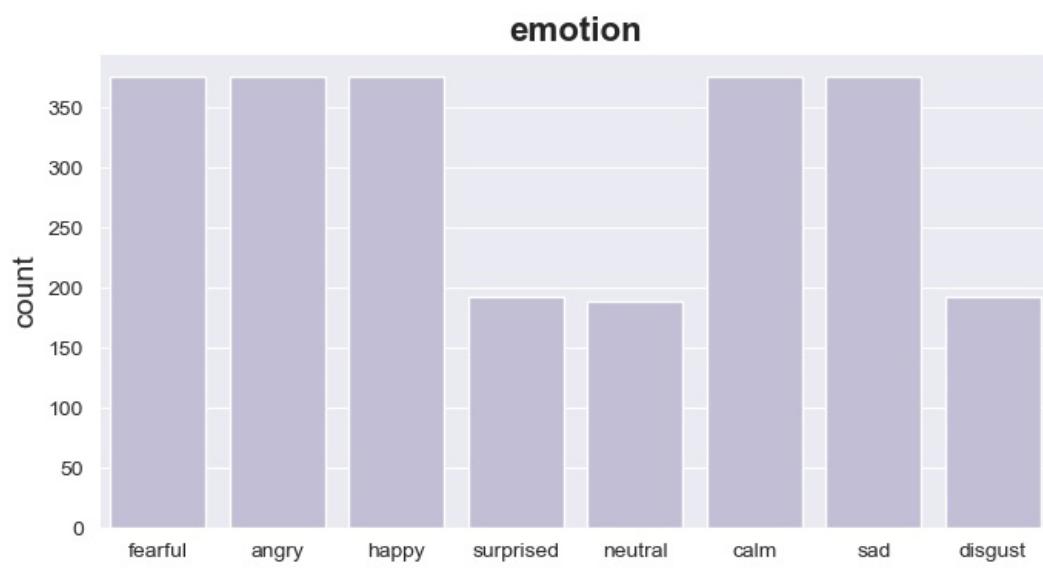
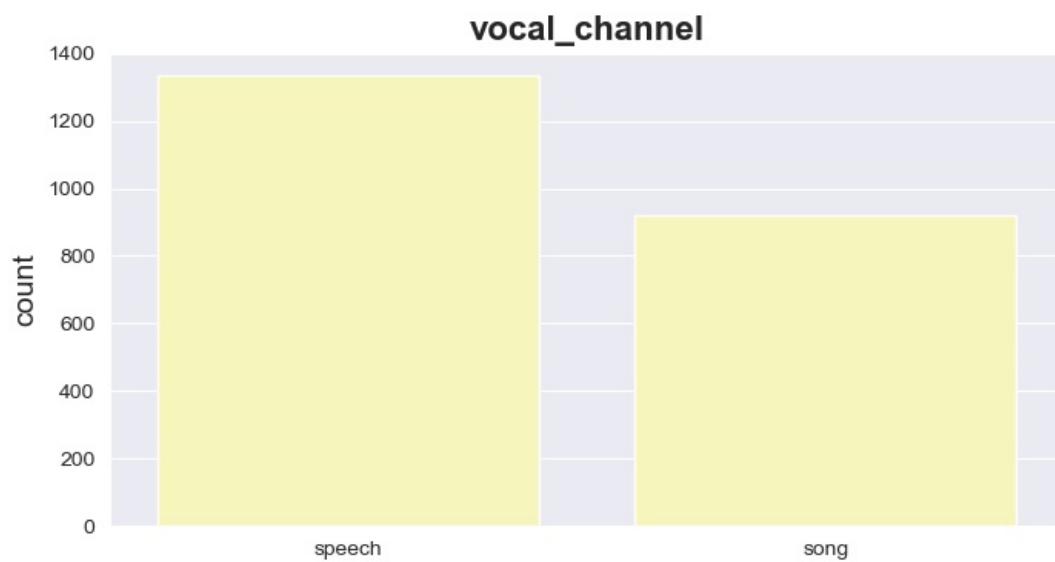
Usiamo alcune **visualizzazioni** per capire meglio le distribuzioni – prima esplorando le variabili **singolarmente**, e focalizzandoci poi su alcune delle loro **interazioni**.

In [22]: `import seaborn as sns`

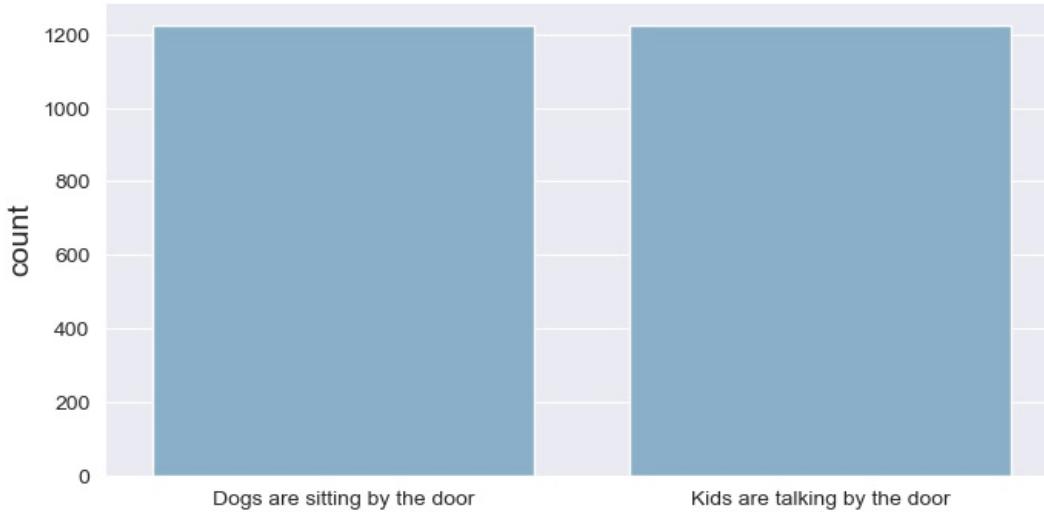
Iniziamo dalle variabili **categoriche**, usando quindi dei **bar chart**; per le ragioni viste in precedenza, escludiamo dalla visualizzazione 'modality':

In [23]: `palette = sns.color_palette("Set3")
sns.set_style("darkgrid")`

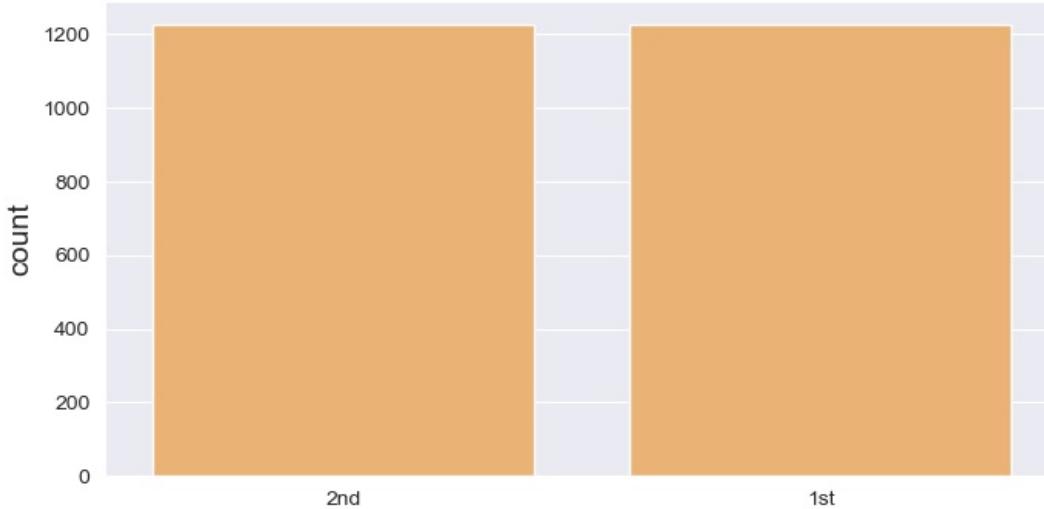
```
for i, col in enumerate(non_numerical_columns):
    if col != 'modality':
        plt.figure(figsize=(8, 4))
        color = palette[i]
        sns.countplot(x=col, data=df, color=color)
        plt.title(col, fontsize=16, fontweight='bold')
        plt.ylabel('count', fontsize=14)
        plt.xlabel('')
        plt.show()
```



statement



repetition



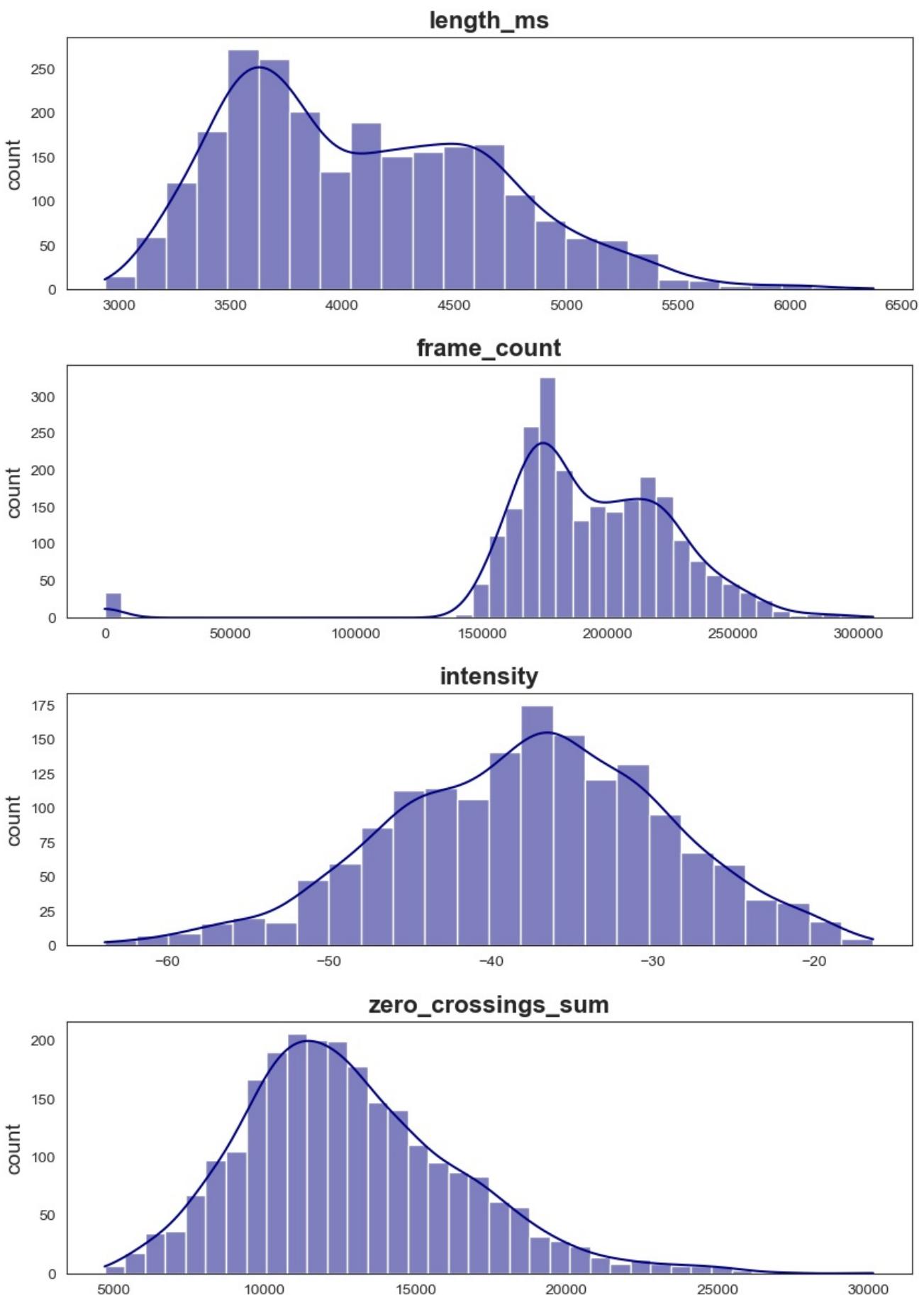
sex

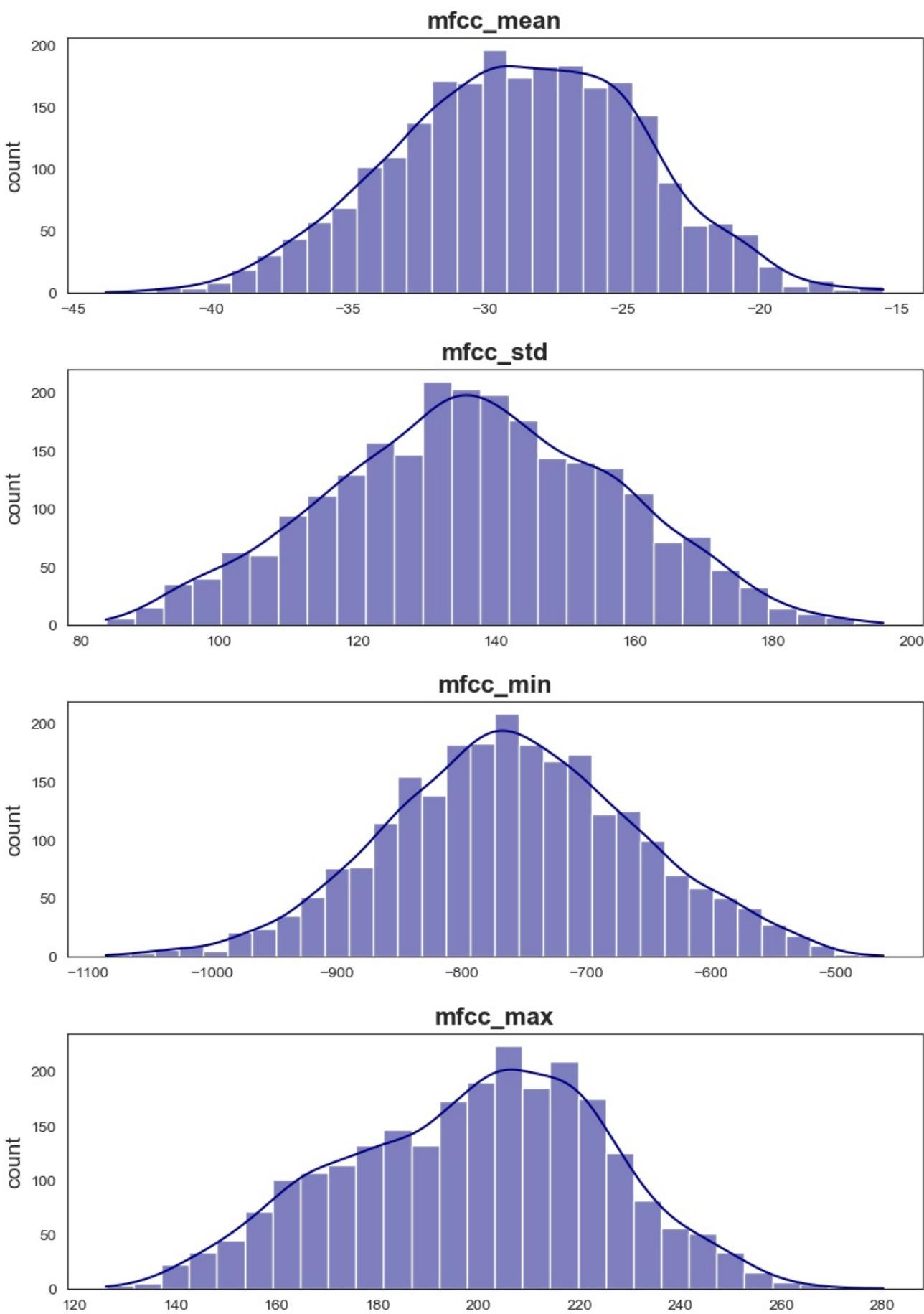


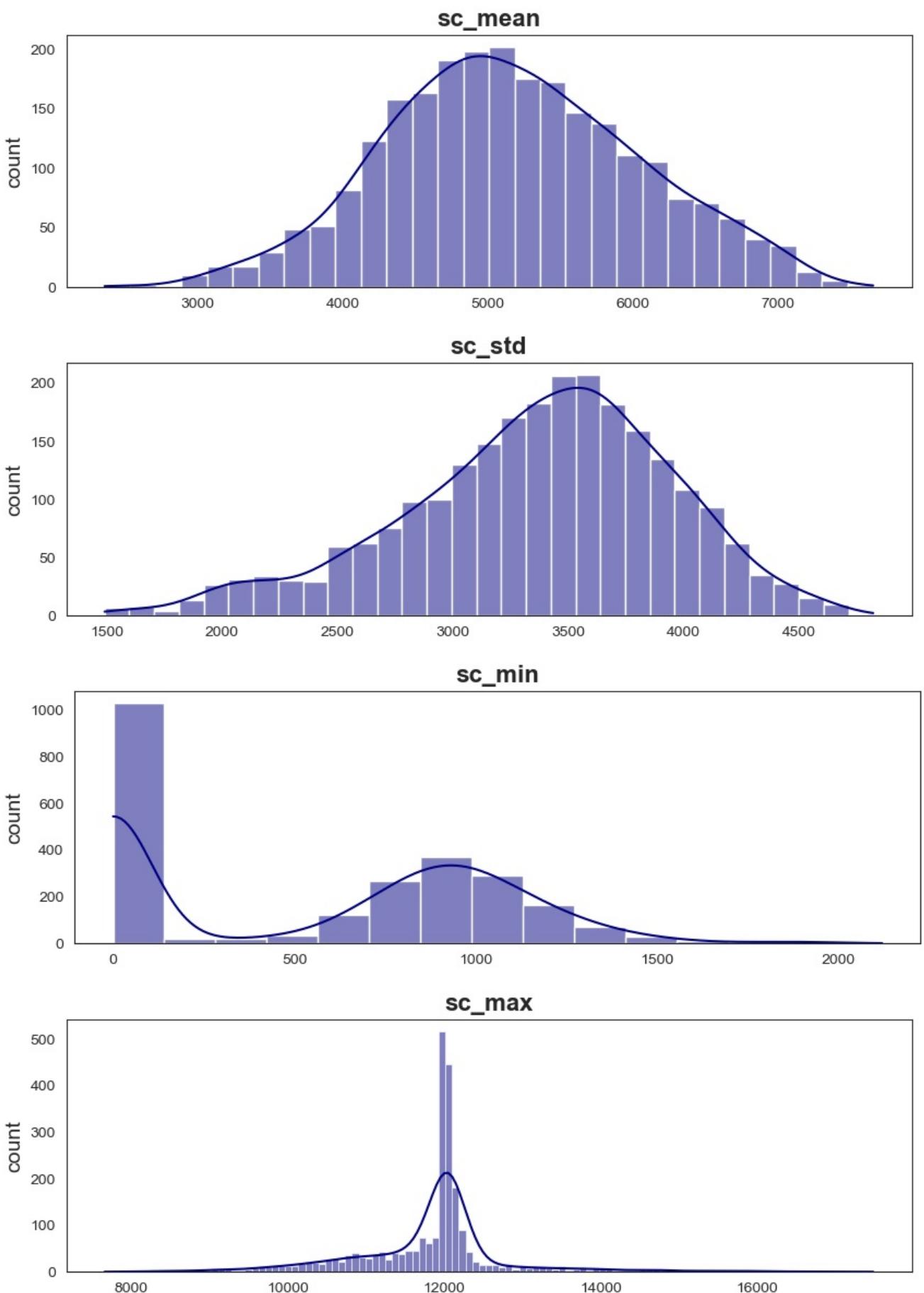
Per la visualizzazione delle variabili **continue** usiamo invece degli **istogrammi**; sempre a causa delle ragioni viste in precedenza, escludiamo dalla visualizzazione gli attributi 'actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max':

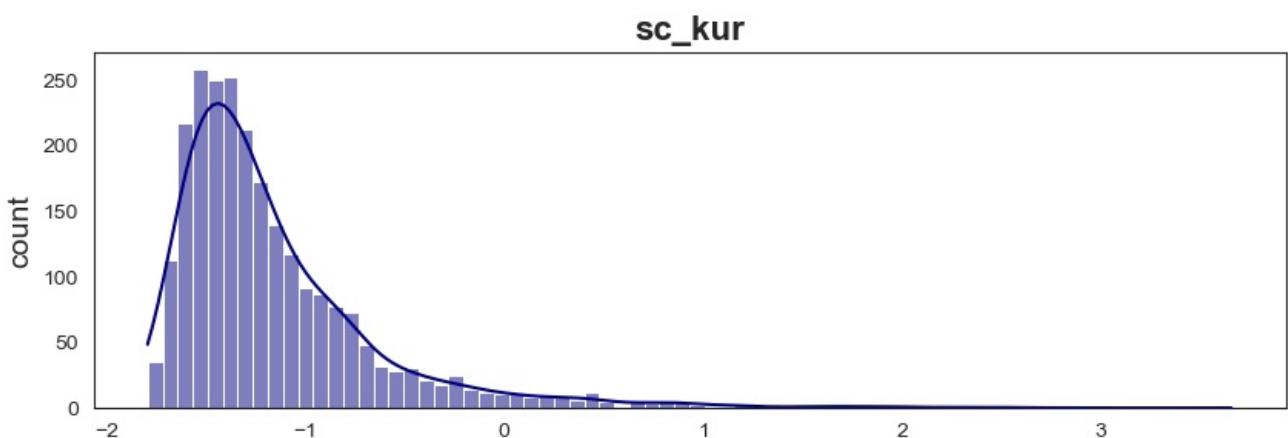
```
In [24]: sns.set_style("white")

for col in numerical_columns:
    if col not in ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max']:
        plt.figure(figsize=(10, 3))
        sns.histplot(df[col], kde=True, color='navy')
        plt.title(col, fontsize=16, fontweight='bold')
        plt.ylabel('count', fontsize=14)
        plt.xlabel('')
        plt.show()
```

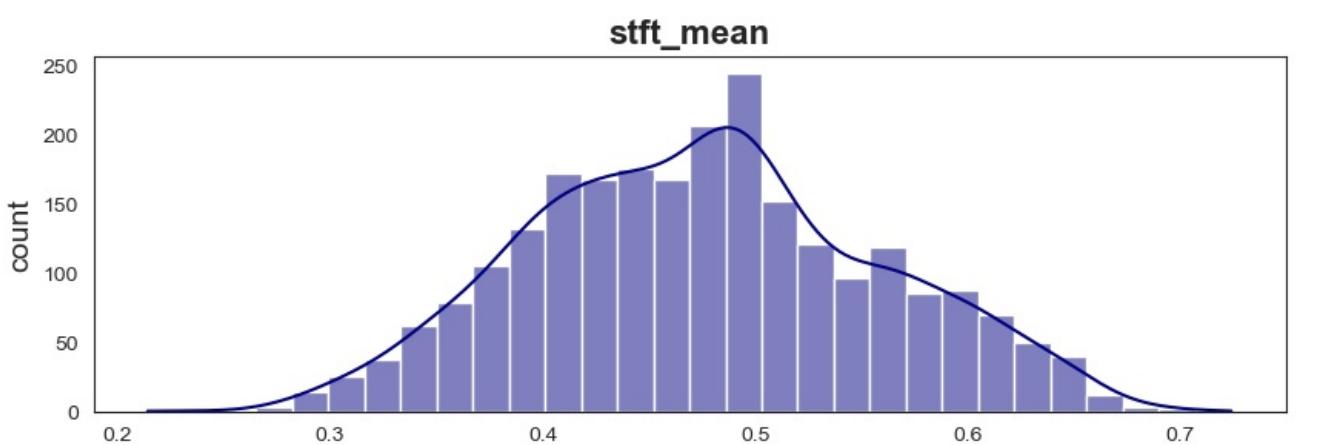




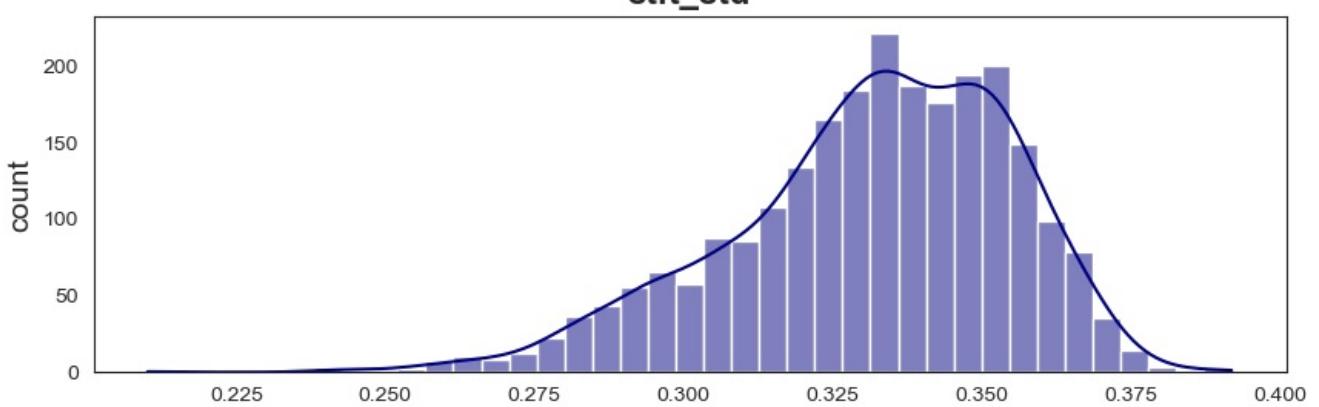


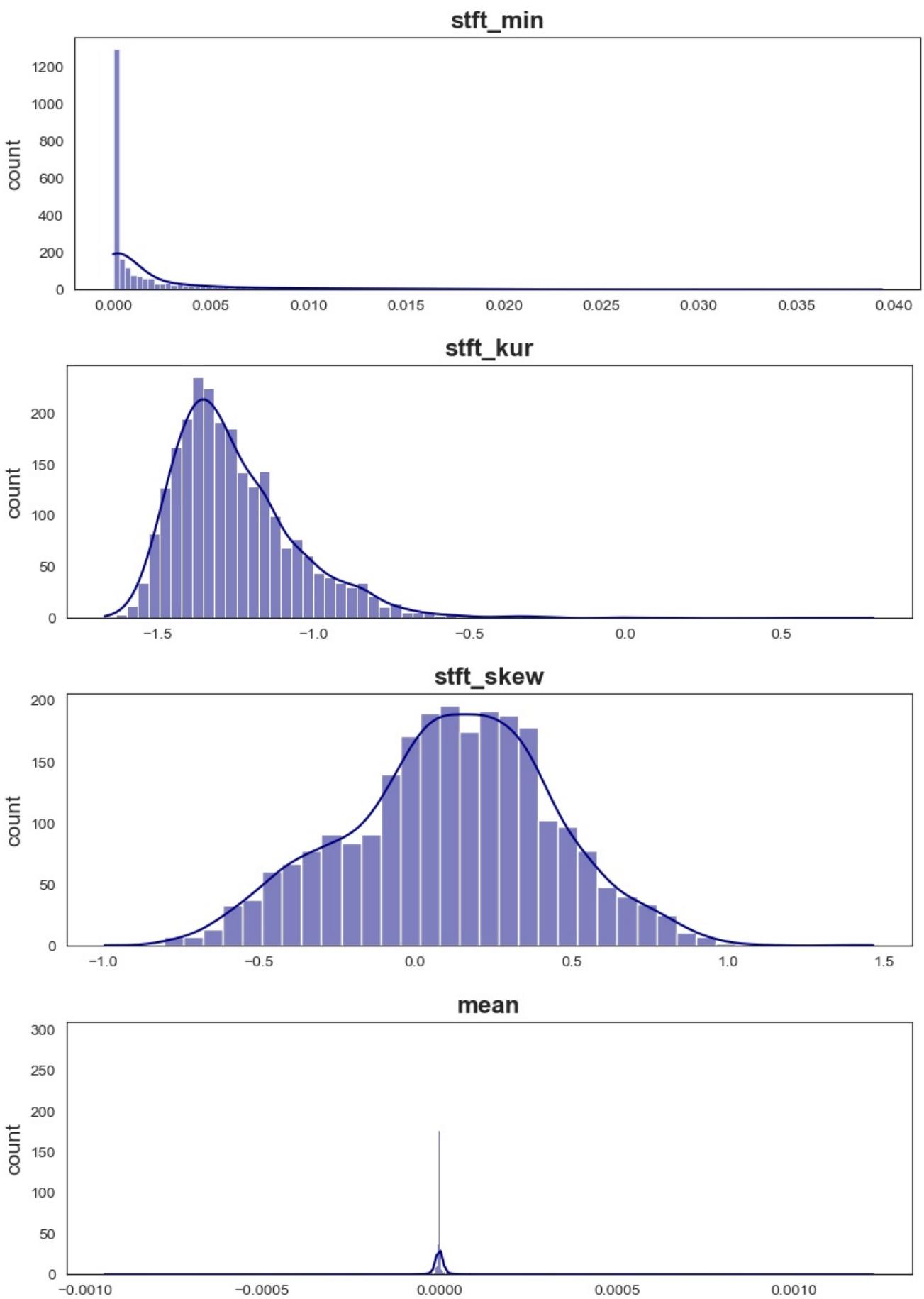


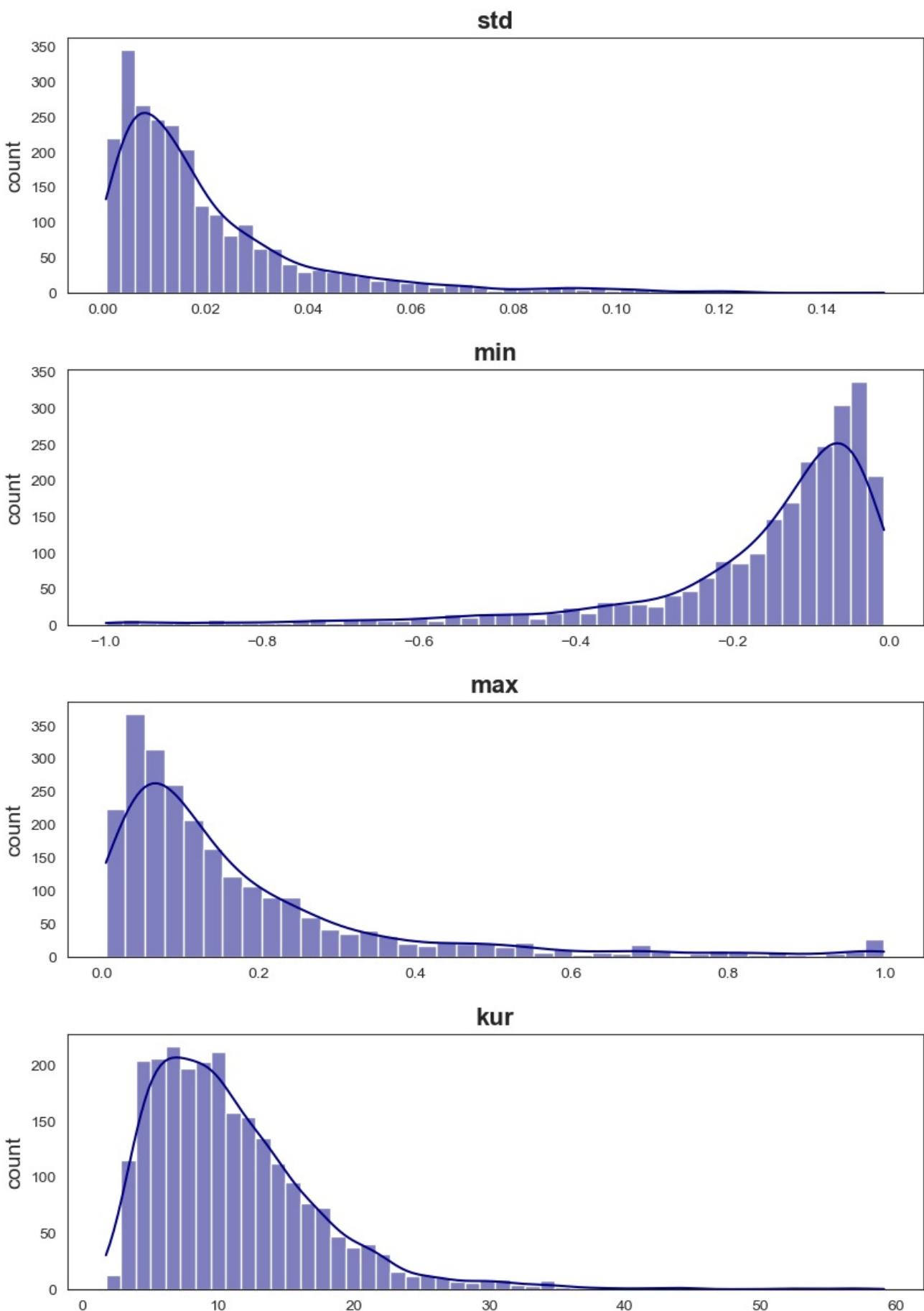
sc_skew.....

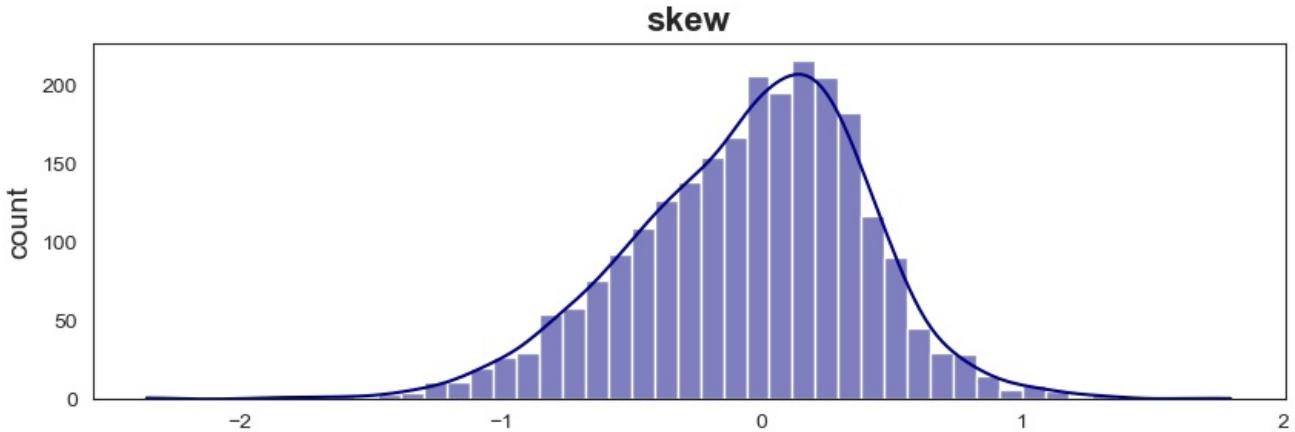


stft_std





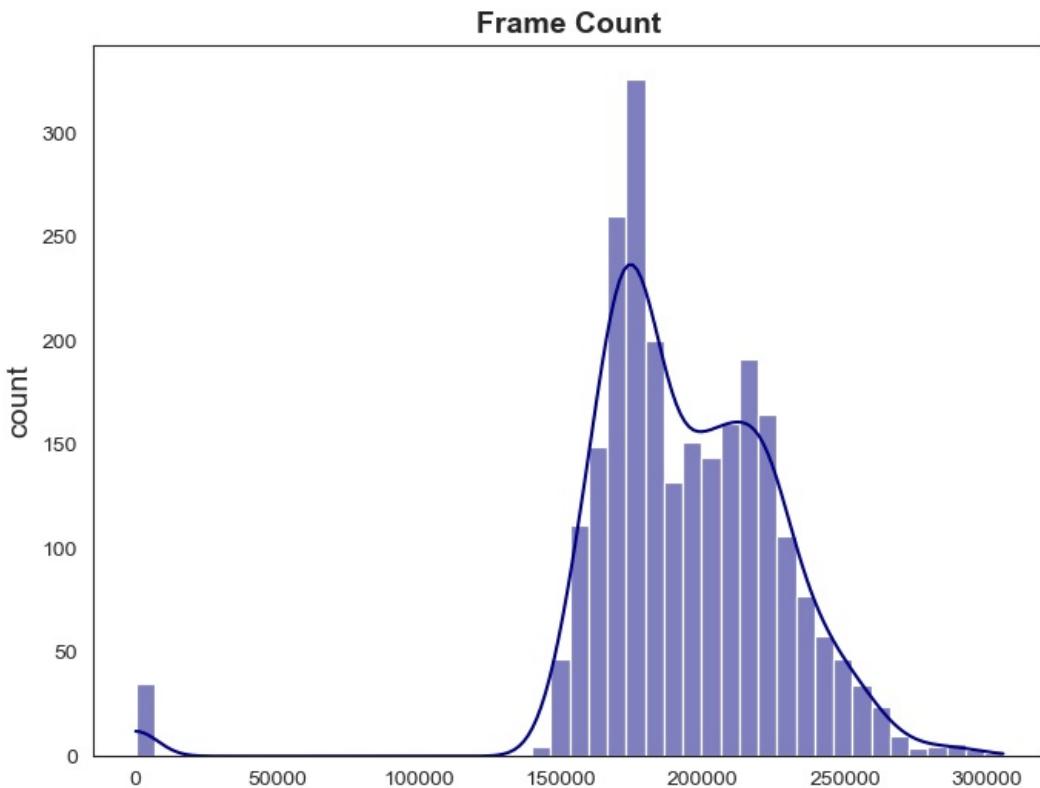




Analizziamo più nel dettaglio alcuni degli istogrammi:

```
In [25]: plt.figure(figsize=(8, 6))
sns.histplot(df['frame_count'], color='navy', kde=True)
plt.ylabel('count', fontsize=14)
plt.xlabel('')
plt.title('Frame Count', fontsize=14, fontweight='bold')

plt.show()
```



L'attributo 'frame_count' presenta alcuni **valori inferiori a 0**; data la semantica con cui è presentato nella descrizione del dataset ("numero di frame nel sample"), non dovrebbe averne:

```
In [26]: len(df[df['frame_count'] < 0])
```

```
Out[26]: 35
```

Anche aumentando di molto la soglia, solo questi 35 record hanno valori così piccoli:

```
In [27]: len(df[df['frame_count'] < 100000])
```

```
Out[27]: 35
```

Infatti, il secondo valore unico più piccolo è 140941:

```
In [28]: unique_fc = df['frame_count'].unique()
sorted_unique_fc = sorted(unique_fc)
sorted_unique_fc[:10]
```

```
Out[28]: [-1.0,
 140941.0,
 142542.0,
 144144.0,
 145745.0,
 145746.0,
 147347.0,
 147348.0,
 148948.0,
 148949.0]
```

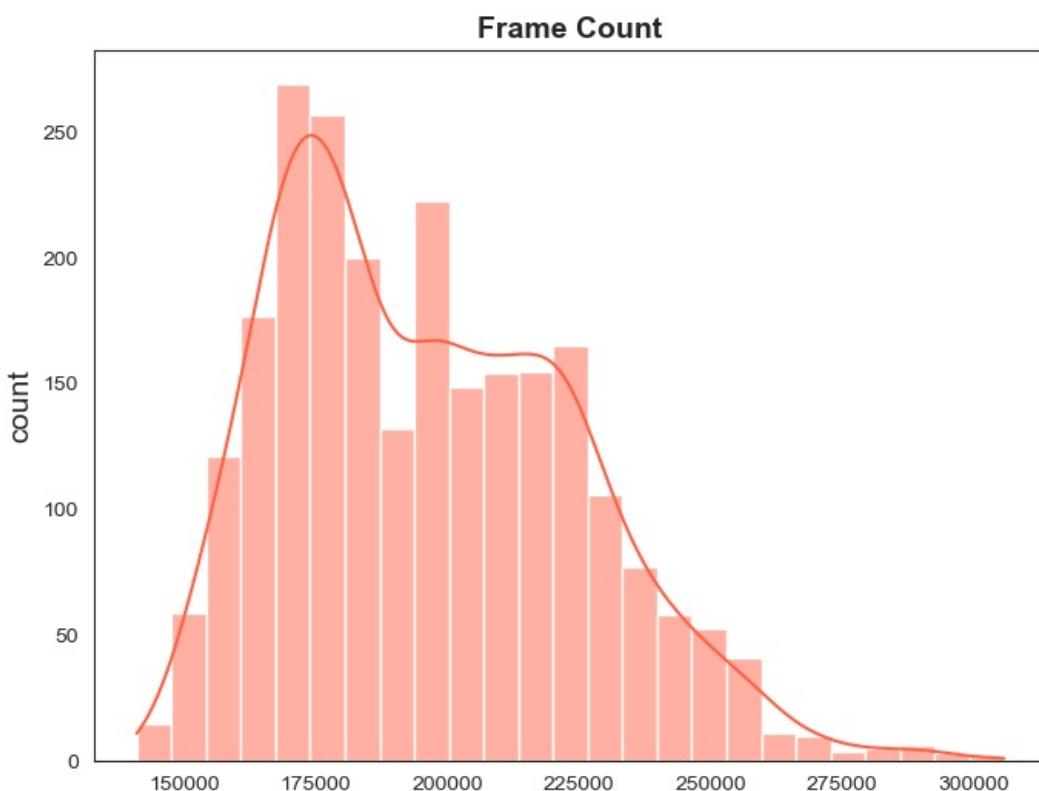
Pensiamo quindi che si tratti di **errori**; li rimpiazziamo con la **media** che la variabile avrebbe se non ci fossero i valori < 0:

```
In [29]: mean_fc = df['frame_count'][df['frame_count'] > 0].mean()
df.loc[df['frame_count'] < 0, 'frame_count'] = mean_fc
```

E ne plottiamo l'istogramma aggiornato:

```
In [30]: plt.figure(figsize=(8, 6))
sns.histplot(df['frame_count'], color='tomato', kde=True)
plt.ylabel('count', fontsize=14)
plt.xlabel('')
plt.title('Frame Count', fontsize=14, fontweight='bold')

plt.show()
```



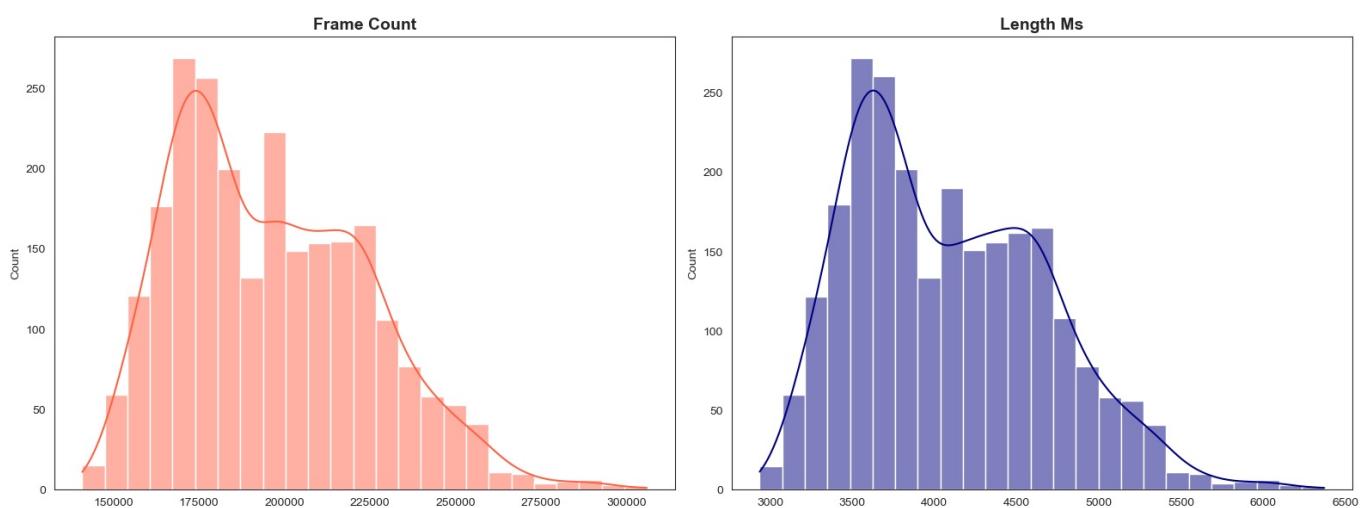
La nuova distribuzione della variabile è **molti simile a quella di 'length_ms'**; semanticamente, questo è infatti un risultato che ci aspetteremmo:

```
In [31]: fig, axes = plt.subplots(1, 2, figsize=(16, 6))

sns.histplot(df['frame_count'], ax=axes[0], color='tomato', kde=True)
axes[0].set_title('Frame Count', fontsize=14, fontweight='bold')
axes[0].set_xlabel('')

sns.histplot(df['length_ms'], ax=axes[1], color='navy', kde=True)
axes[1].set_title('Length Ms', fontsize=14, fontweight='bold')
axes[1].set_xlabel('')

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



Le due variabili **correlano** quasi perfettamente:

```
In [32]: df['length_ms'].corr(df['frame_count'])
```

```
Out[32]: 0.9929036807367501
```

Torneremo nel seguito al trattamento di queste e di diverse delle altre variabili. Per ora, possiamo fare alcune osservazioni a partire dagli istogrammi:

- 'sc_min' e 'stft_min' hanno entrambe 1021 valori a 0; si tratta in effetti degli stessi record:

```
In [33]: len(df[(df['sc_min'] == 0) & (df['stft_min'] == 0)])
```

```
Out[33]: 1021
```

- Molte delle altre variabili (e.g. 'sc_kur', 'stft_std', 'stft_kur', 'std', 'min', 'max', 'kur') presentano **distribuzioni asimmetriche**; ad esempio, 'sc_kur' ha un massimo di 3.65, ma meno del 10% dei valori è > -0.5:

```
In [34]: df['sc_kur'].max()
```

```
Out[34]: 3.65795320733888
```

```
In [35]: len(df[df['sc_kur'] > -0.5]) / len(df['sc_kur'])
```

```
Out[35]: 0.09787928221859707
```

- La variabile 'mean' ha una varianza vicina allo 0:

```
In [36]: df['mean'].var()
```

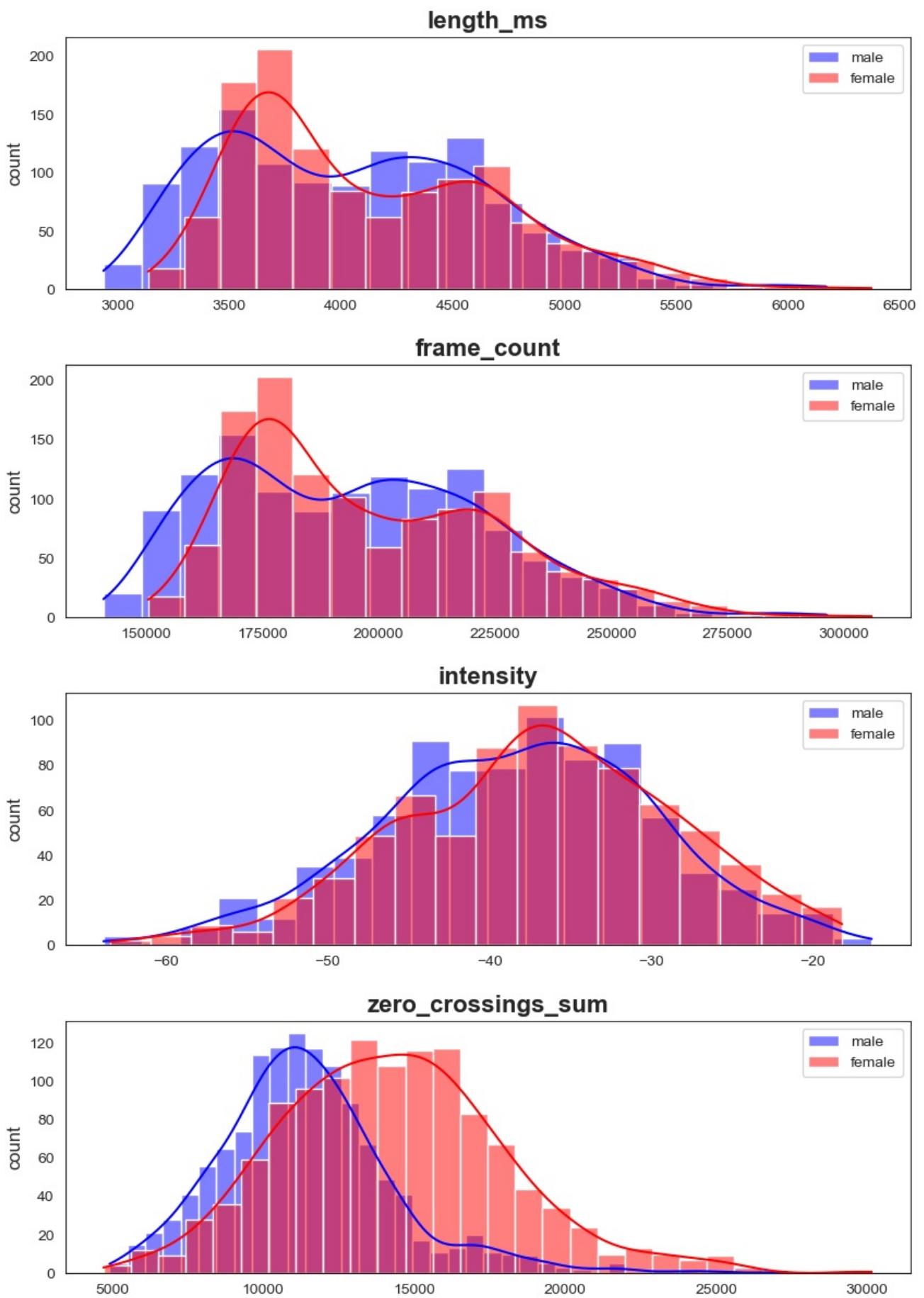
```
Out[36]: 1.8212297990049587e-09
```

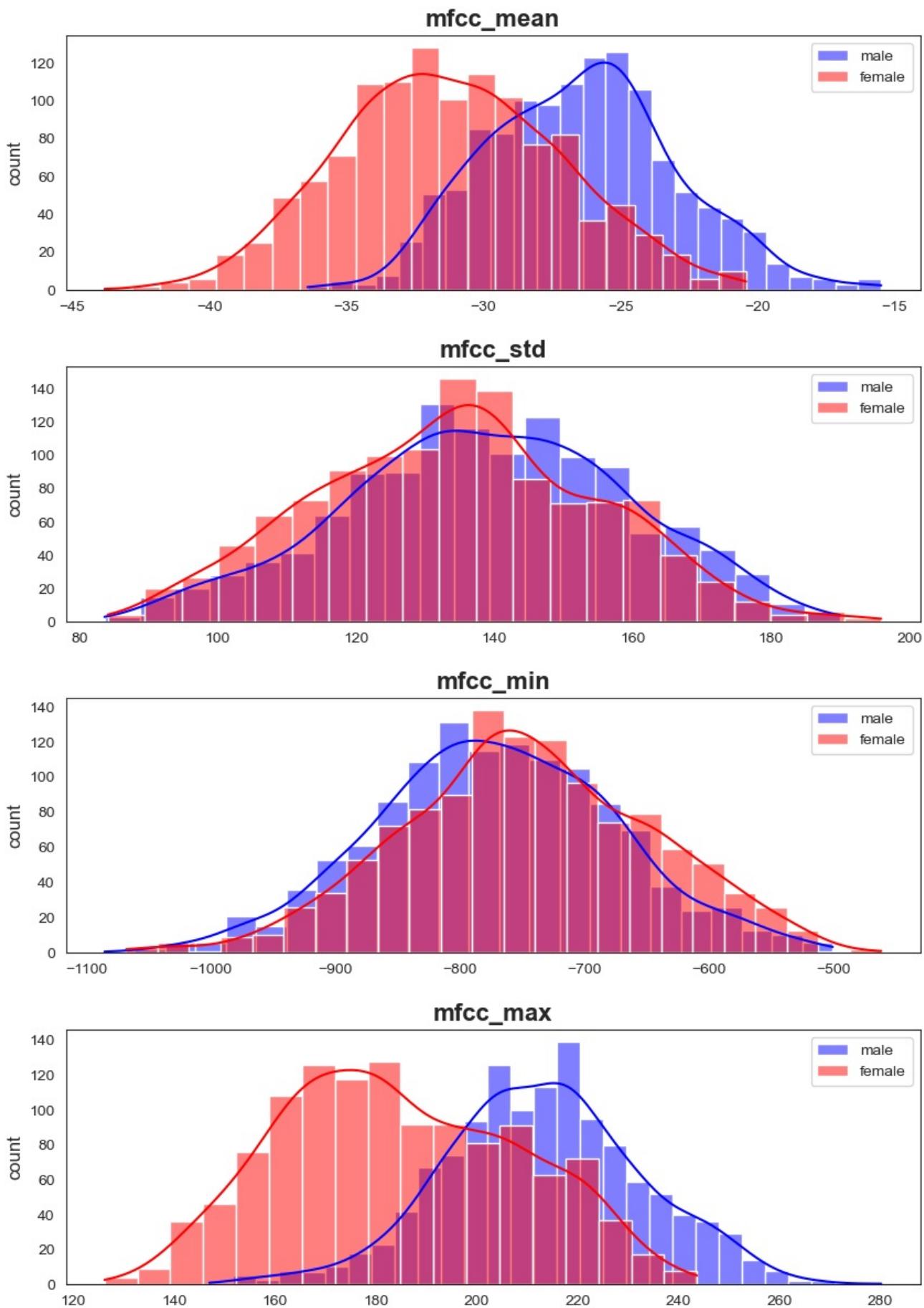
Dopo aver esplorato le variabili singolarmente, analizziamo ora alcune delle loro **interazioni**. In particolare, ci focalizziamo su 'sex' ed 'emotion', perché saranno i target della nostra classificazione.

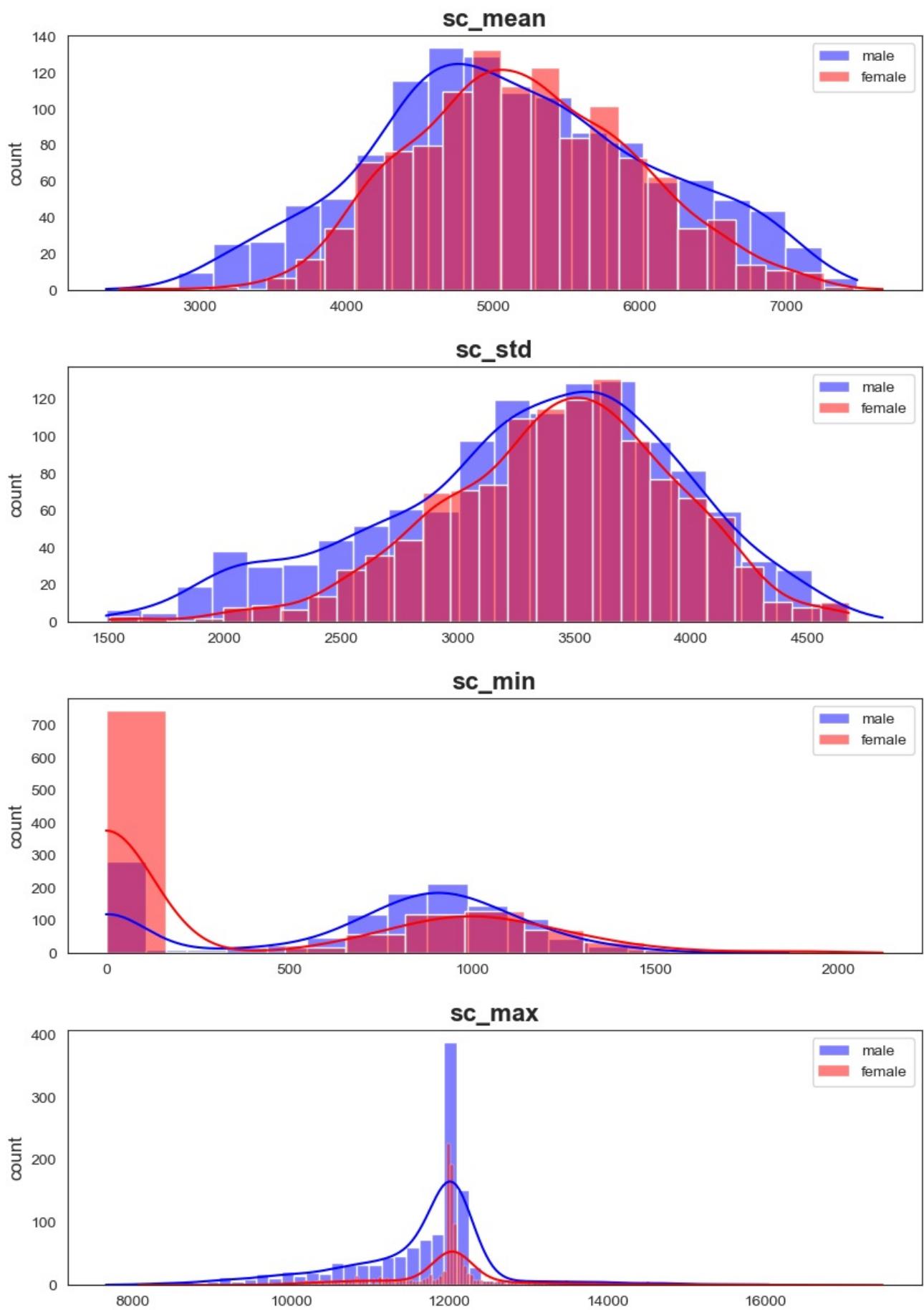
Sex:

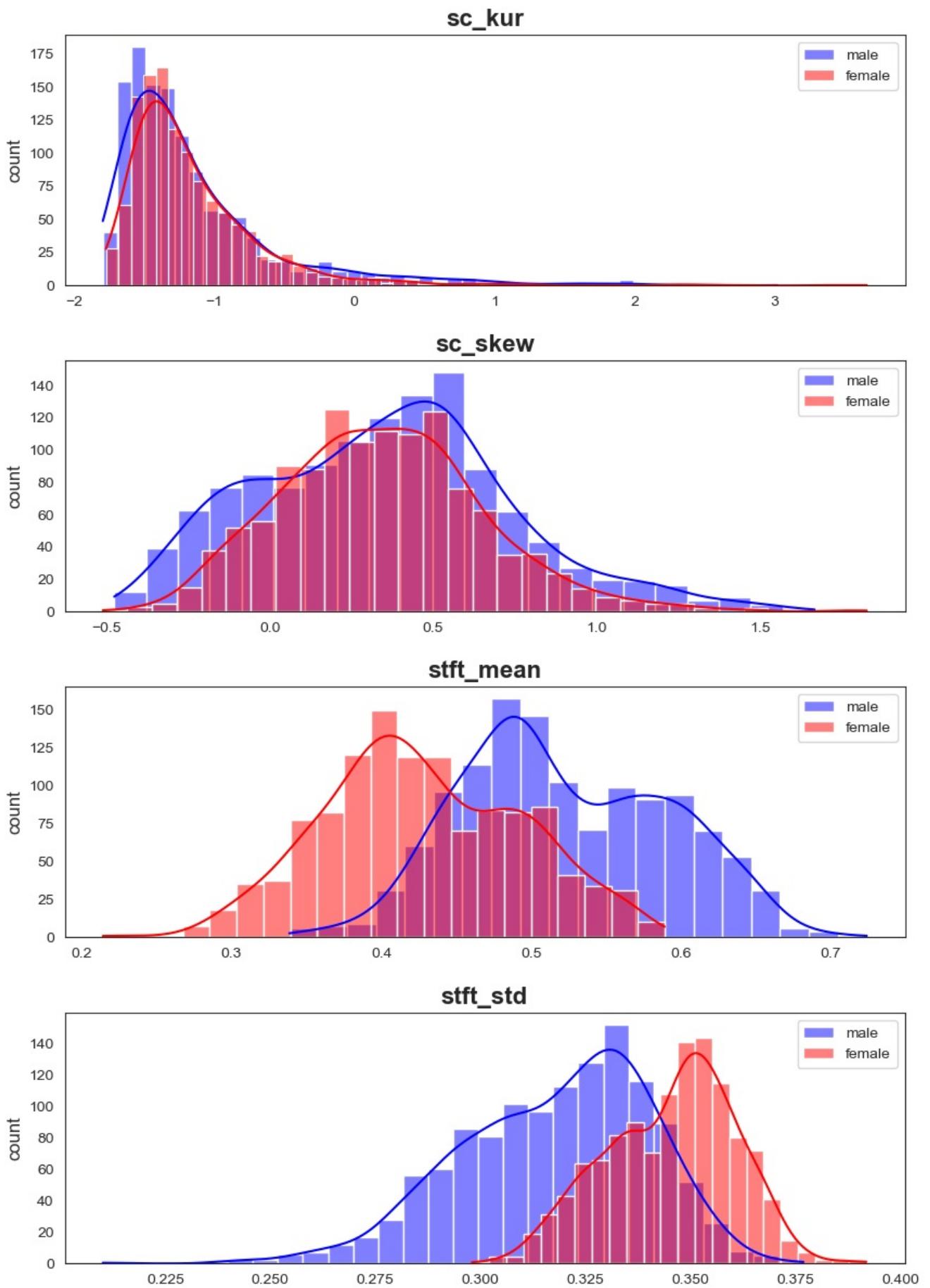
```
In [37]: male = df['sex'] == 'M'
female = df['sex'] == 'F'
```

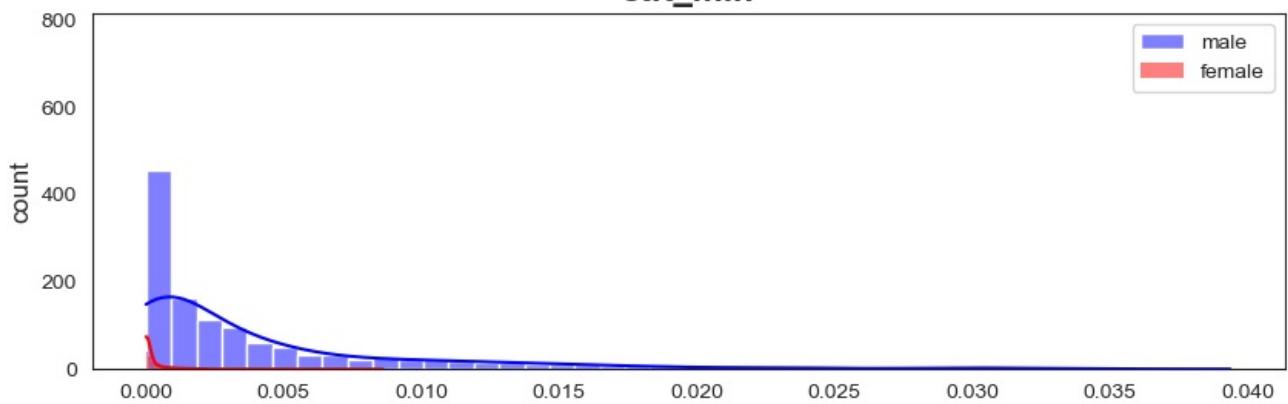
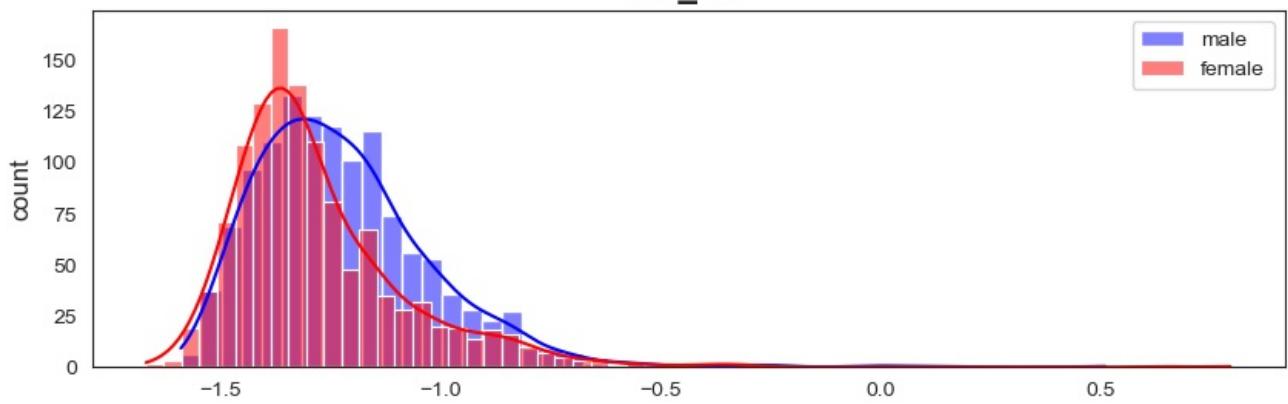
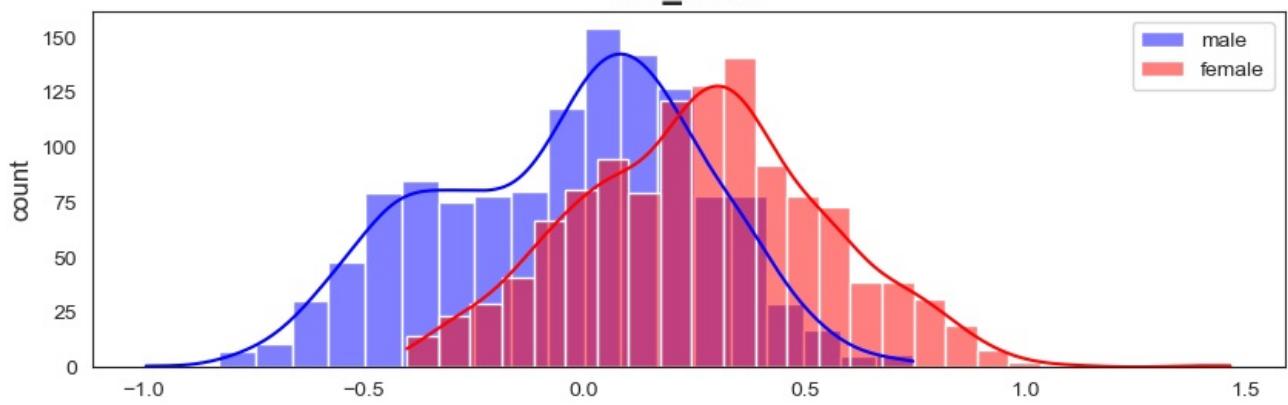
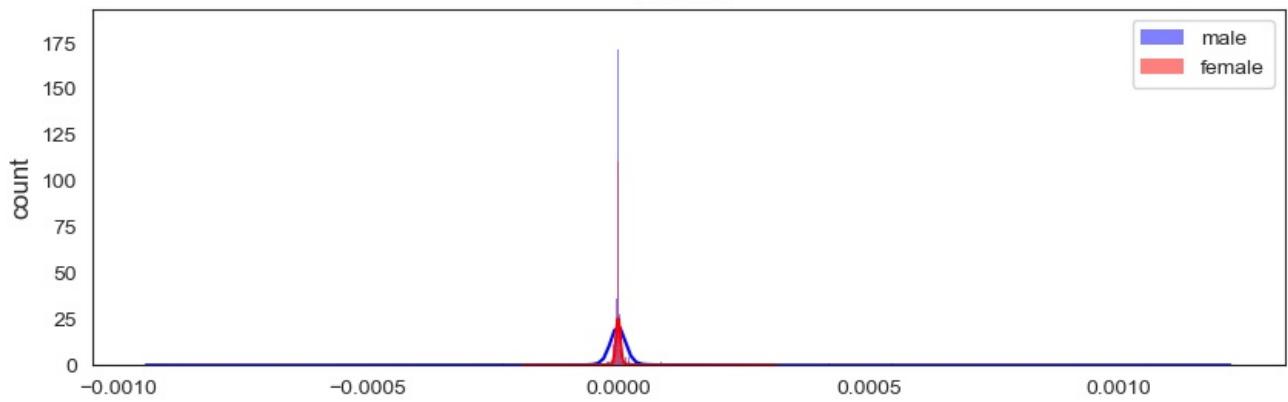
```
In [38]: for col in numerical_columns:
    if col not in ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max']:
        plt.figure(figsize=(10, 3))
        sns.histplot(df[male][col], label='male', kde=True, alpha=0.5, color='b')
        sns.histplot(df[female][col], label='female', kde=True, alpha=0.5, color='r')
        plt.title(col, fontsize=16, fontweight='bold')
        plt.ylabel('count', fontsize=12)
        plt.xlabel('')
        plt.legend()
        plt.show()
```

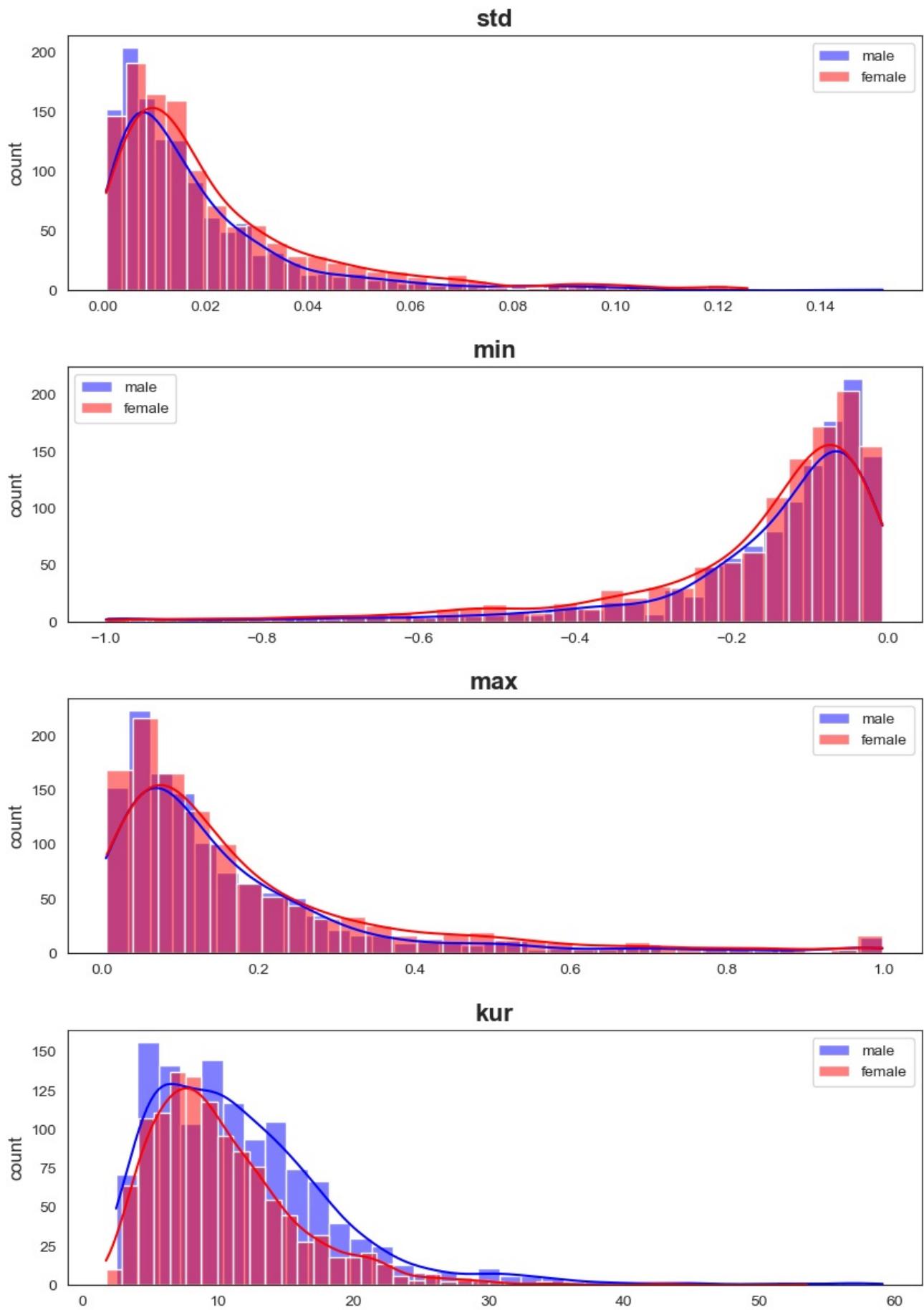


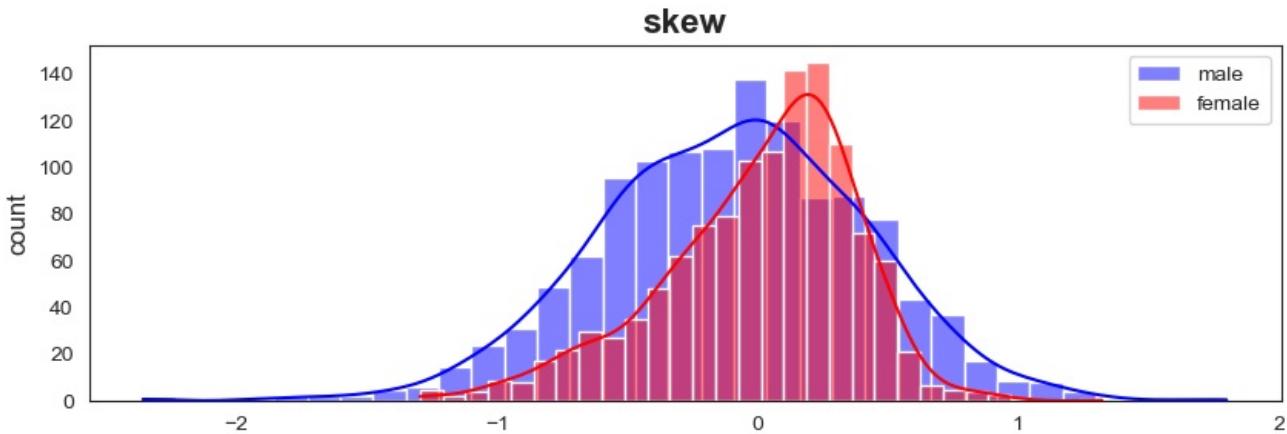






stft_min**stft_kur****stft_skew****mean**





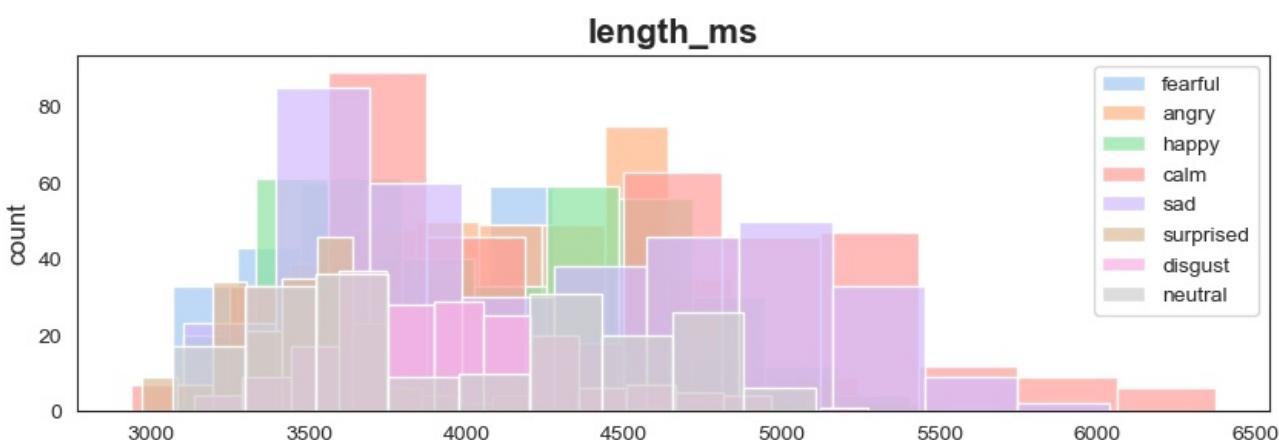
Vediamo che 'M' e 'F' sembrano avere distribuzioni **abbastanza distinte** rispetto ad alcuni degli attributi (ad esempio: 'stft_min', 'stft_std', 'zero_crossings_sum', 'mfcc_max').

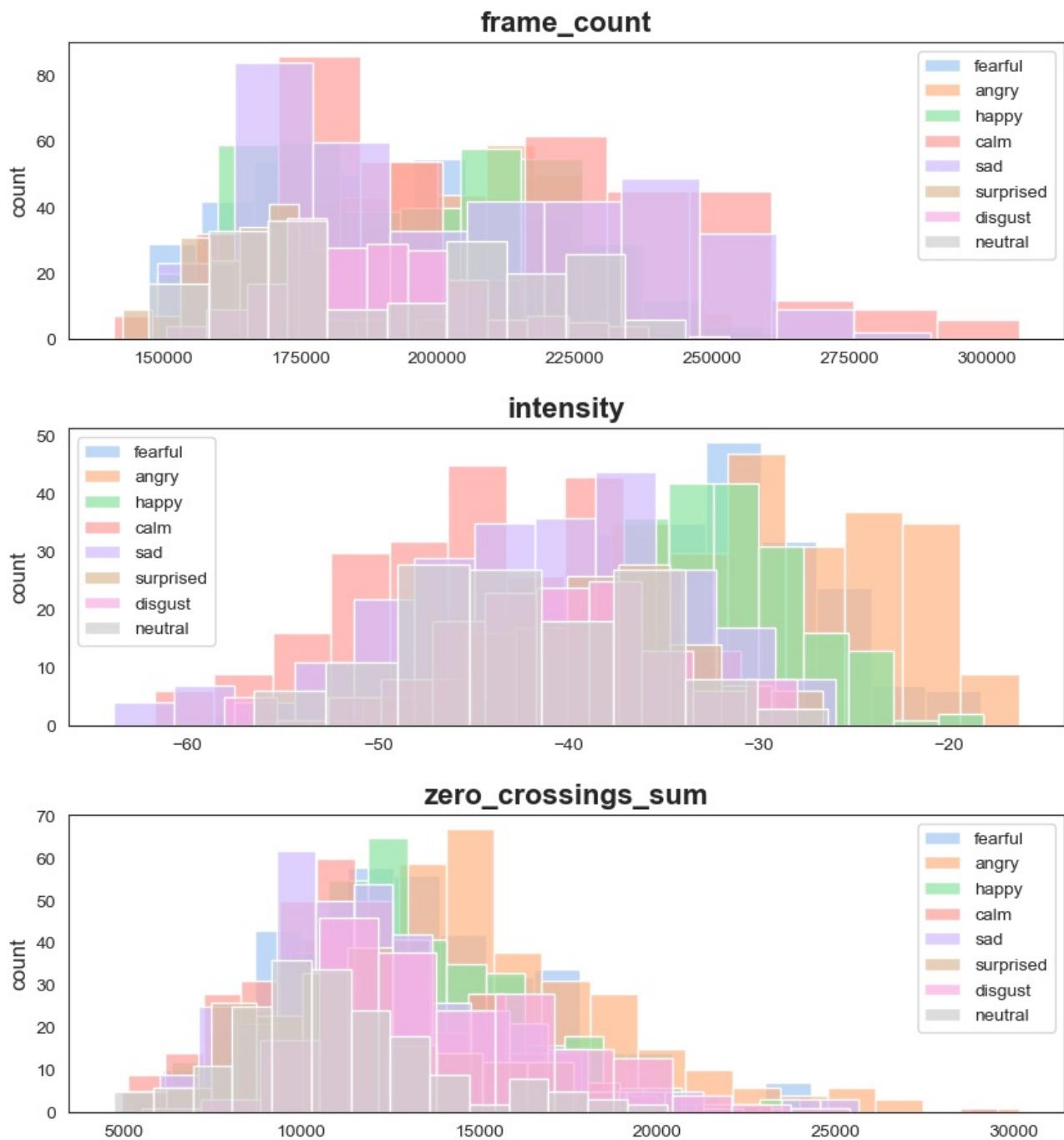
Emotion:

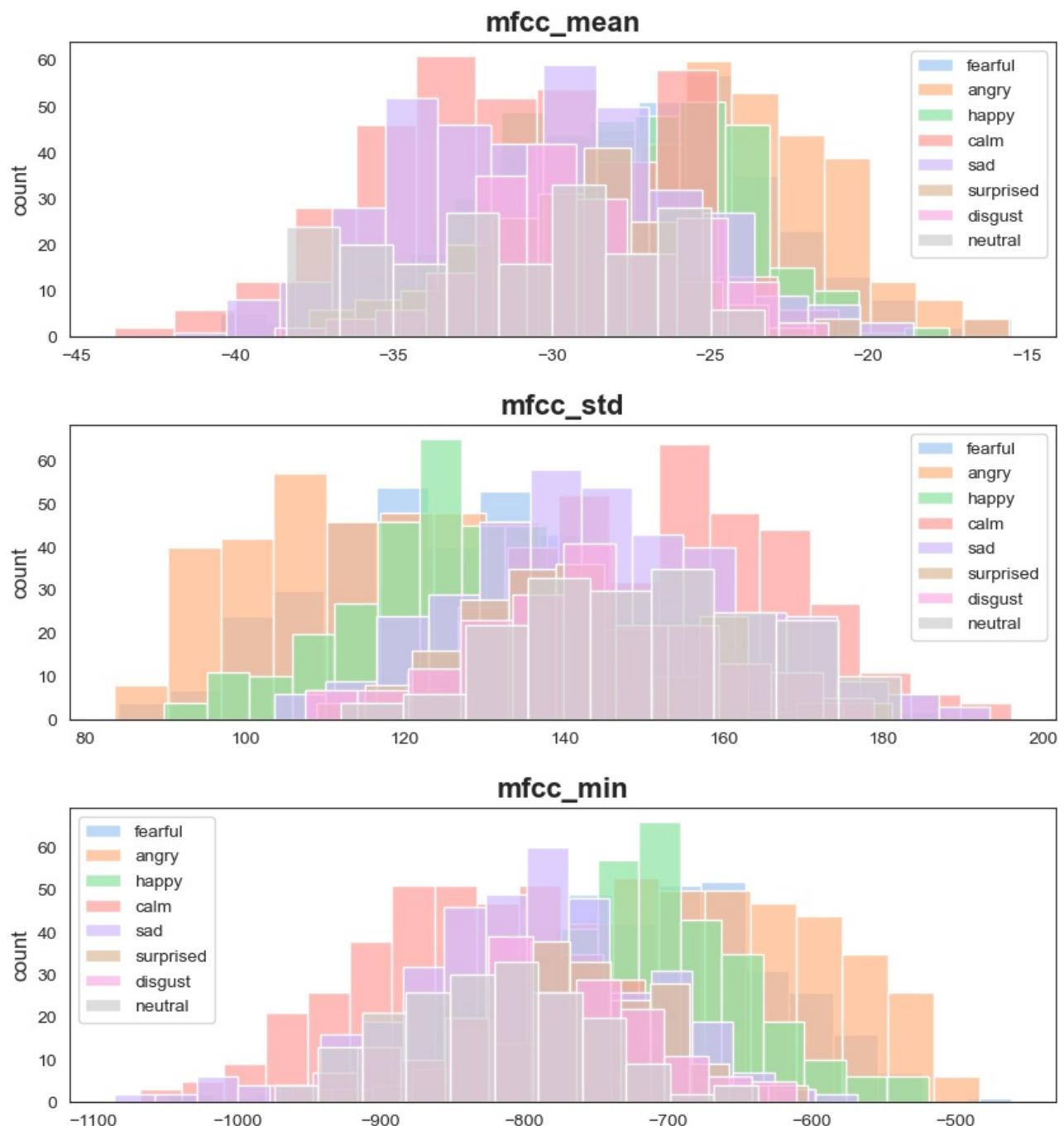
```
In [39]: fearful = df['emotion'] == 'fearful'
angry = df['emotion'] == 'angry'
happy = df['emotion'] == 'happy'
calm = df['emotion'] == 'calm'
sad = df['emotion'] == 'sad'
surprised = df['emotion'] == 'surprised'
disgust = df['emotion'] == 'disgust'
neutral = df['emotion'] == 'neutral'
```

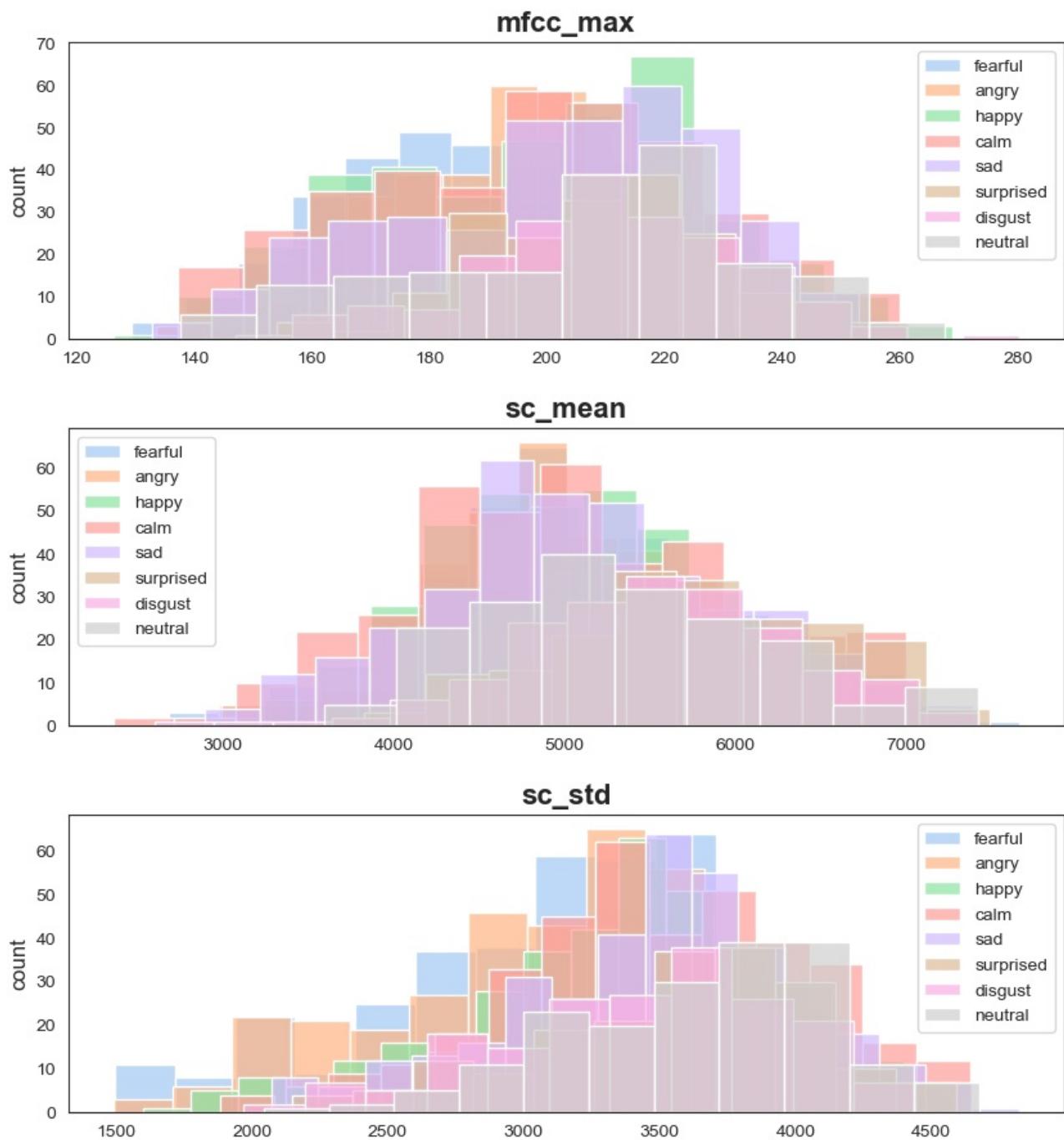
```
In [40]: pastel_colors = sns.color_palette("pastel", 8)

for col in numerical_columns:
    if col not in ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max']:
        plt.figure(figsize=(10, 3))
        sns.histplot(df[fearful][col], label='fearful', alpha=0.7, color=pastel_colors[0])
        sns.histplot(df[angry][col], label='angry', alpha=0.7, color=pastel_colors[1])
        sns.histplot(df[happy][col], label='happy', alpha=0.7, color=pastel_colors[2])
        sns.histplot(df[calm][col], label='calm', alpha=0.7, color=pastel_colors[3])
        sns.histplot(df[sad][col], label='sad', alpha=0.7, color=pastel_colors[4])
        sns.histplot(df[surprised][col], label='surprised', alpha=0.7, color=pastel_colors[5])
        sns.histplot(df[disgust][col], label='disgust', alpha=0.7, color=pastel_colors[6])
        sns.histplot(df[neutral][col], label='neutral', alpha=0.7, color=pastel_colors[7])
        plt.title(col, fontsize=16, fontweight='bold')
        plt.ylabel('count', fontsize=12)
        plt.xlabel('')
        plt.legend(facecolor = 'white')
        plt.show()
```

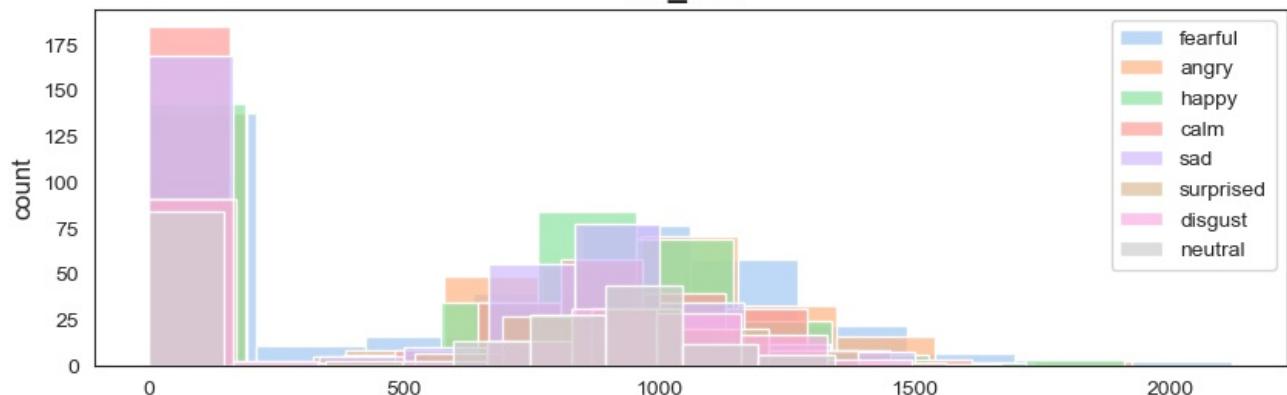




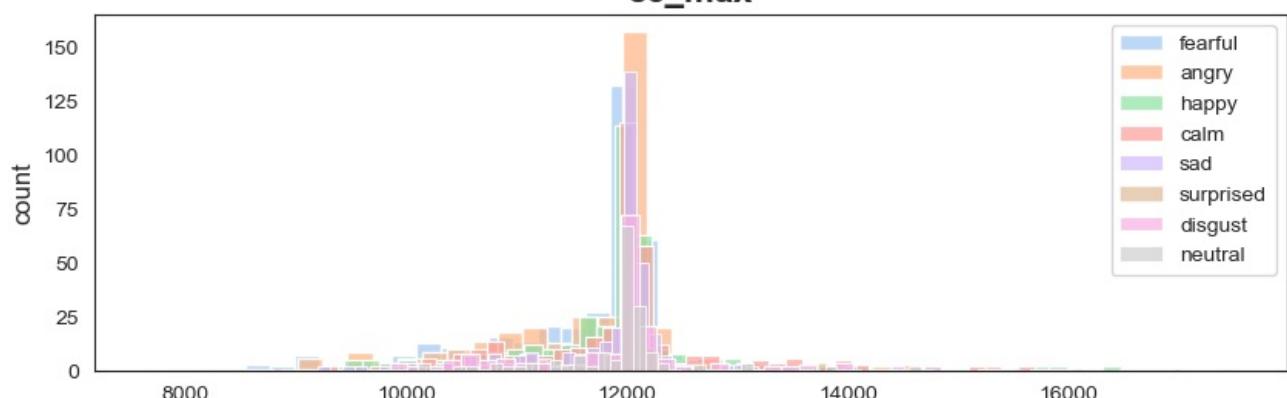




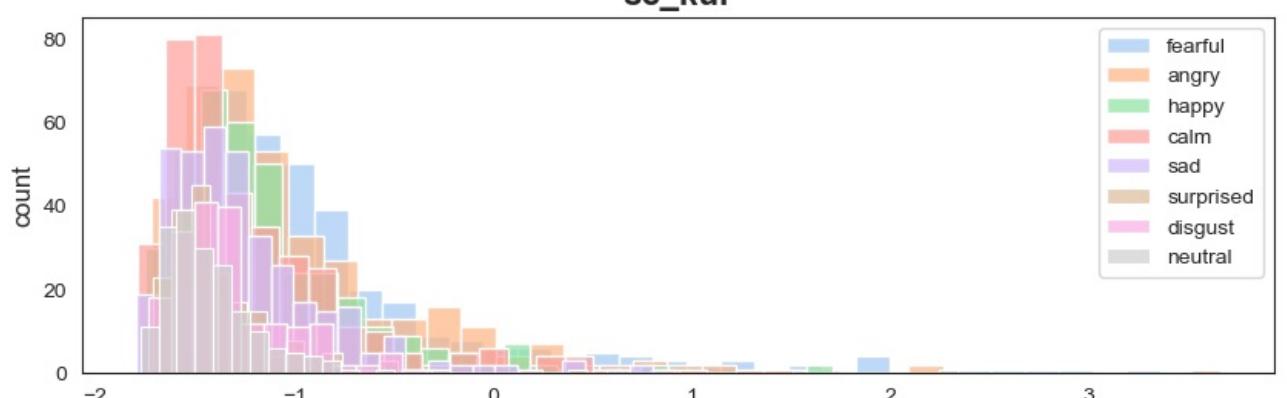
sc_min

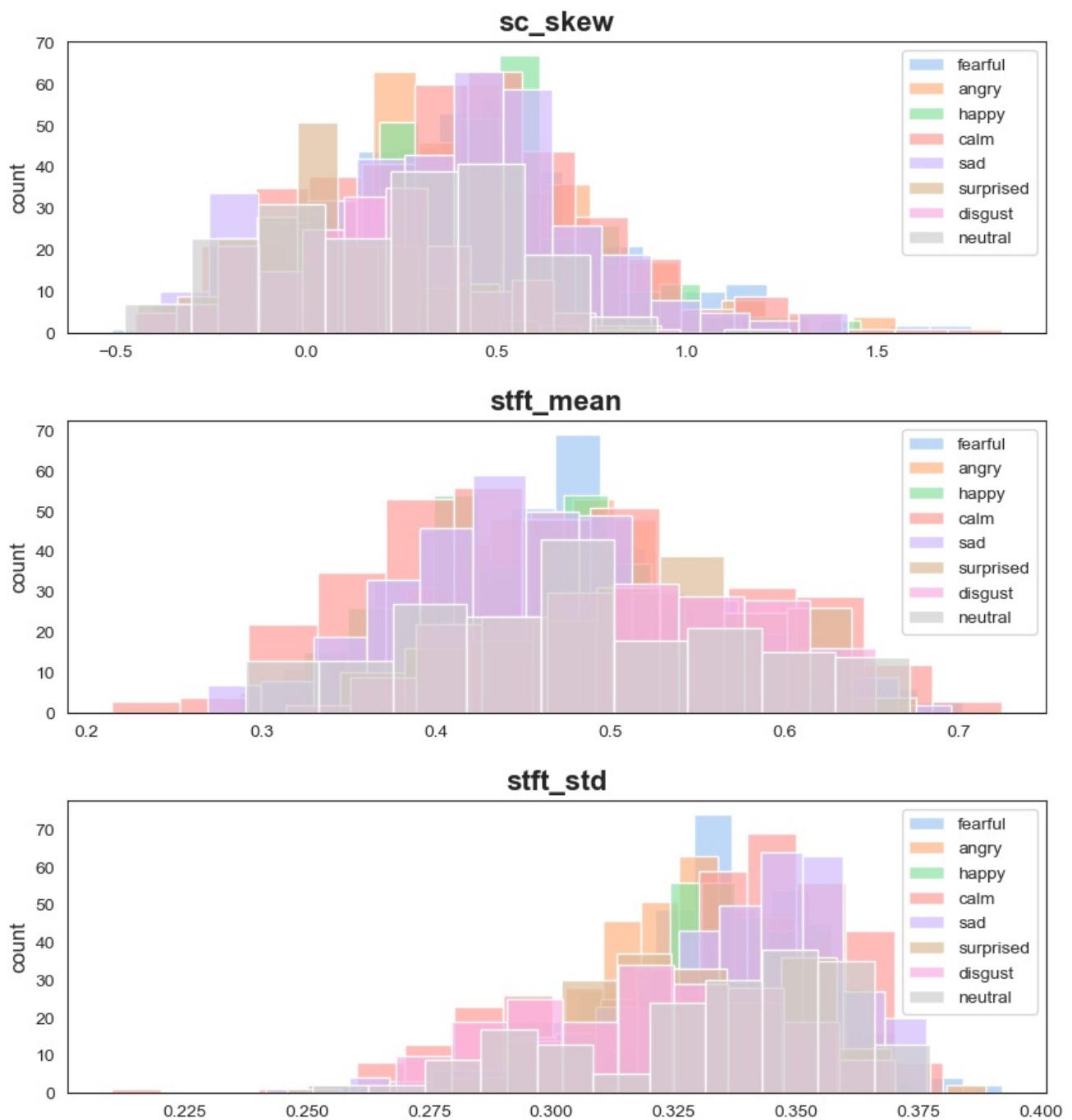


sc_max

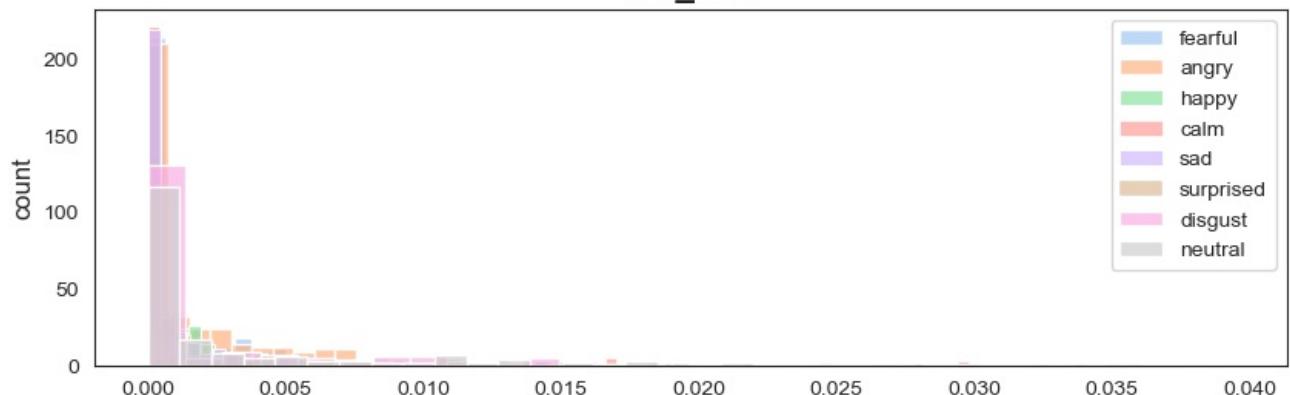


sc_kur

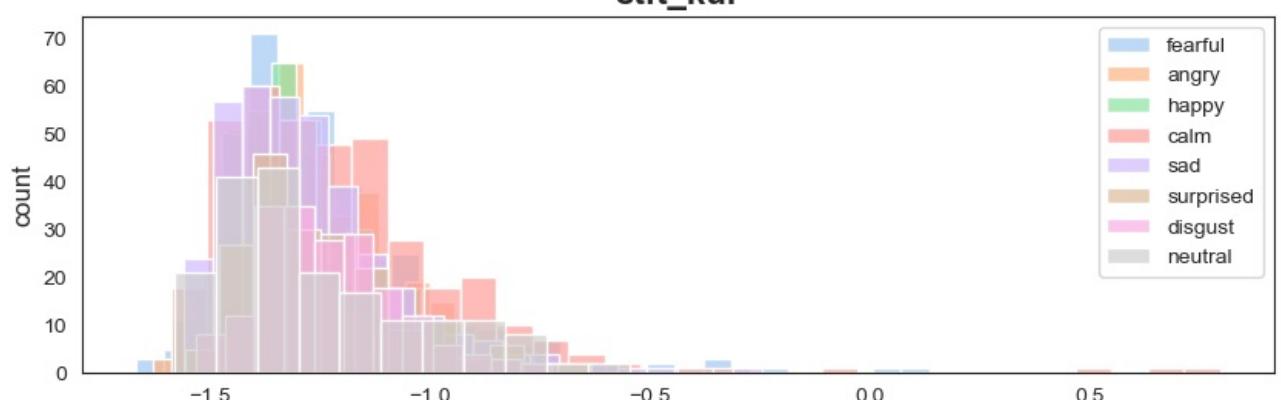




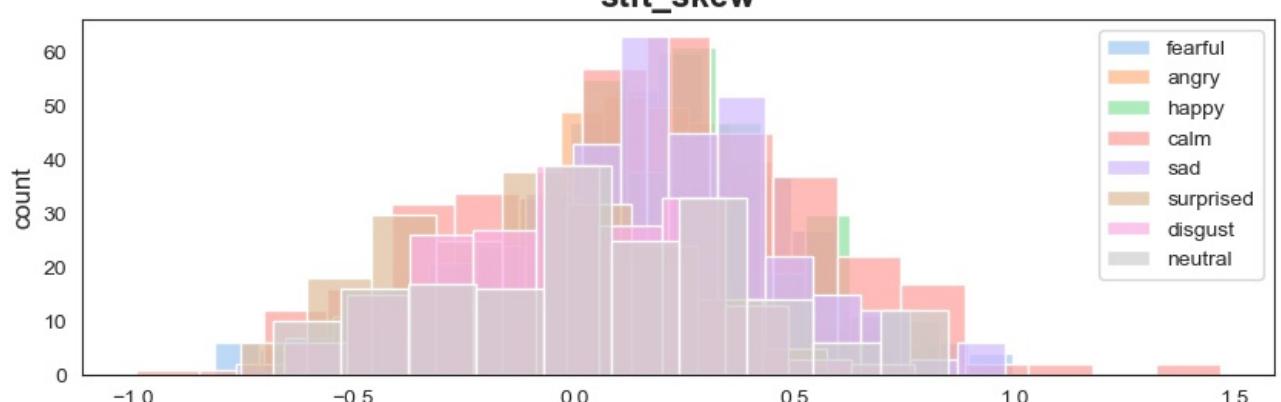
stft_min

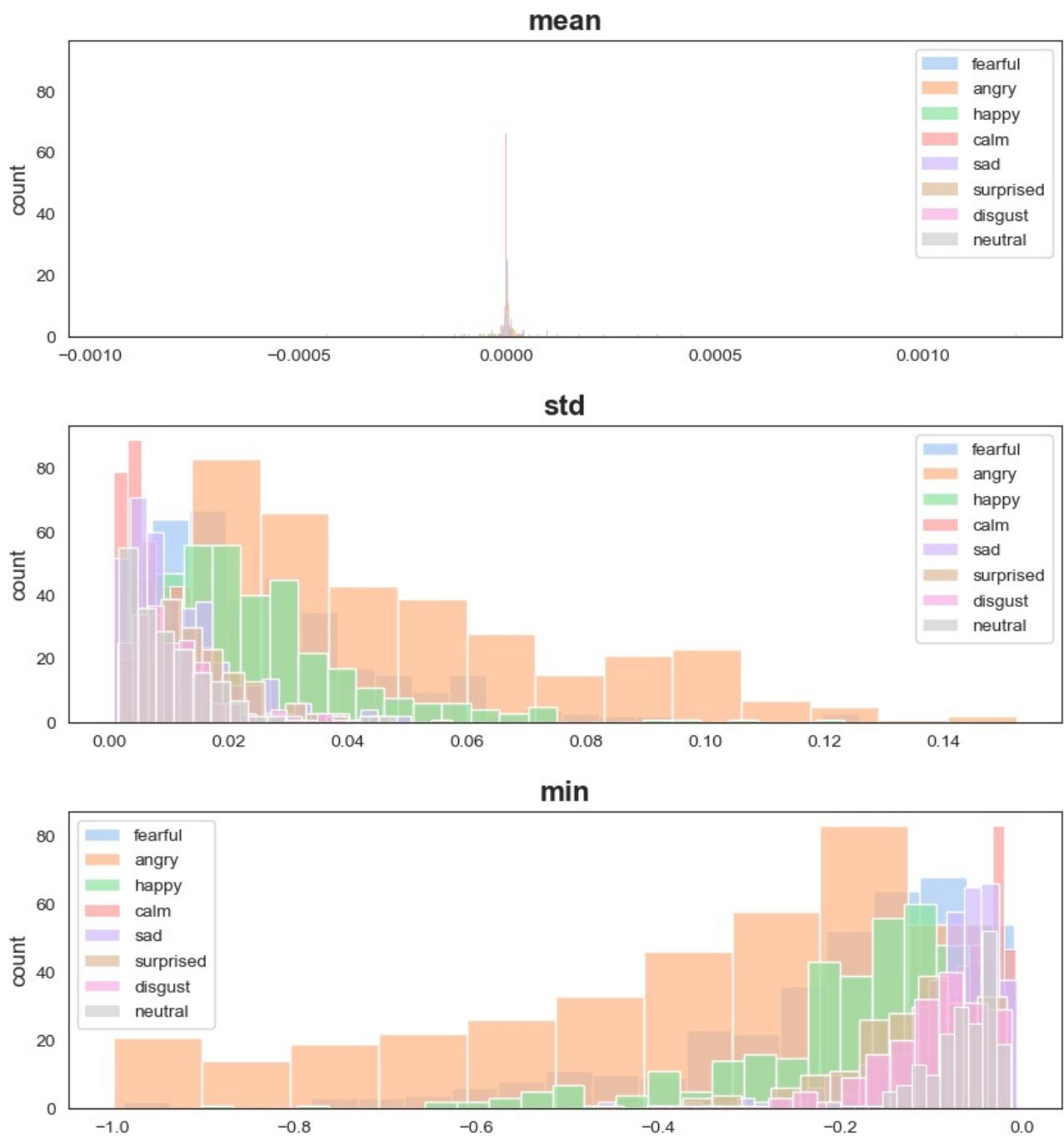


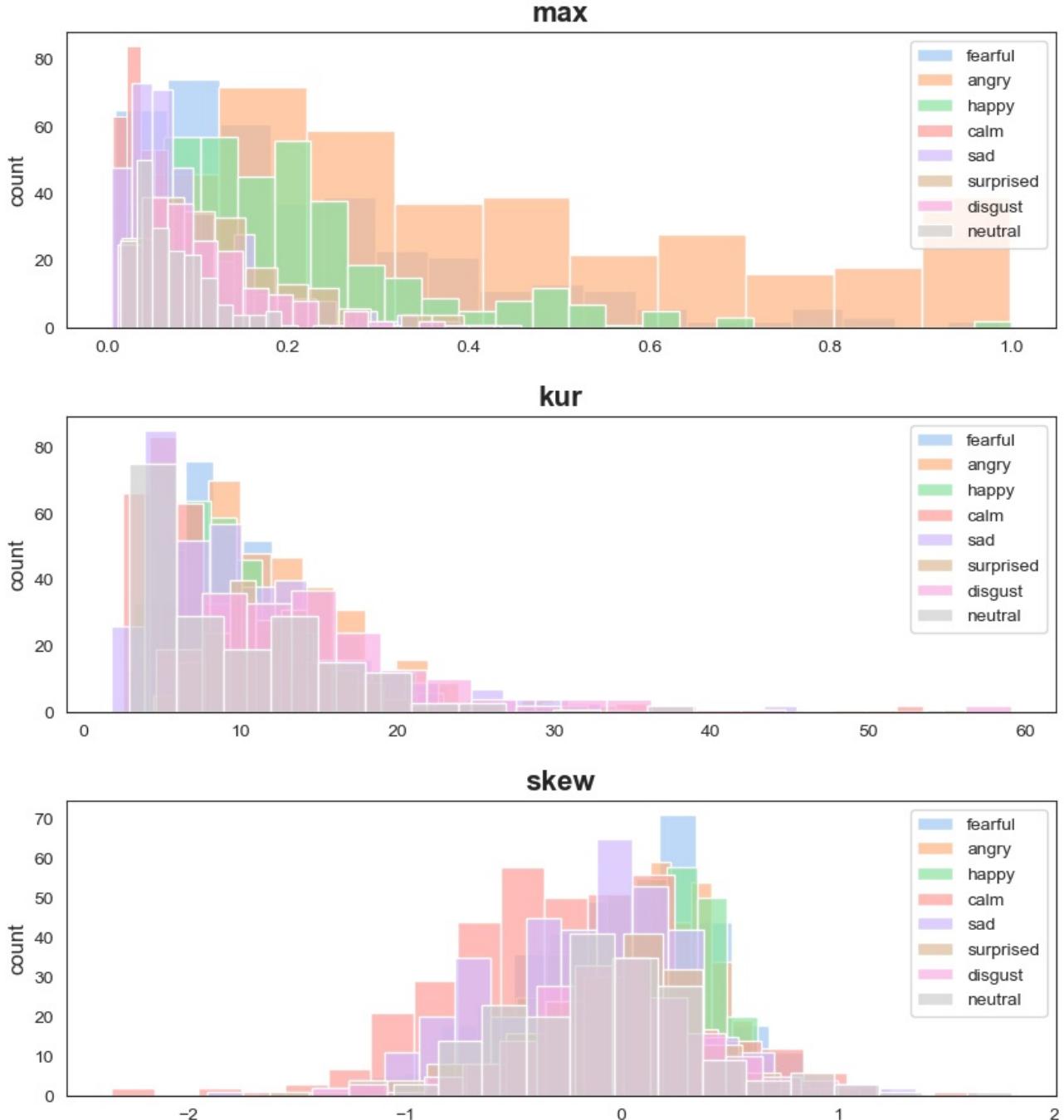
stft_kur



stft_skew





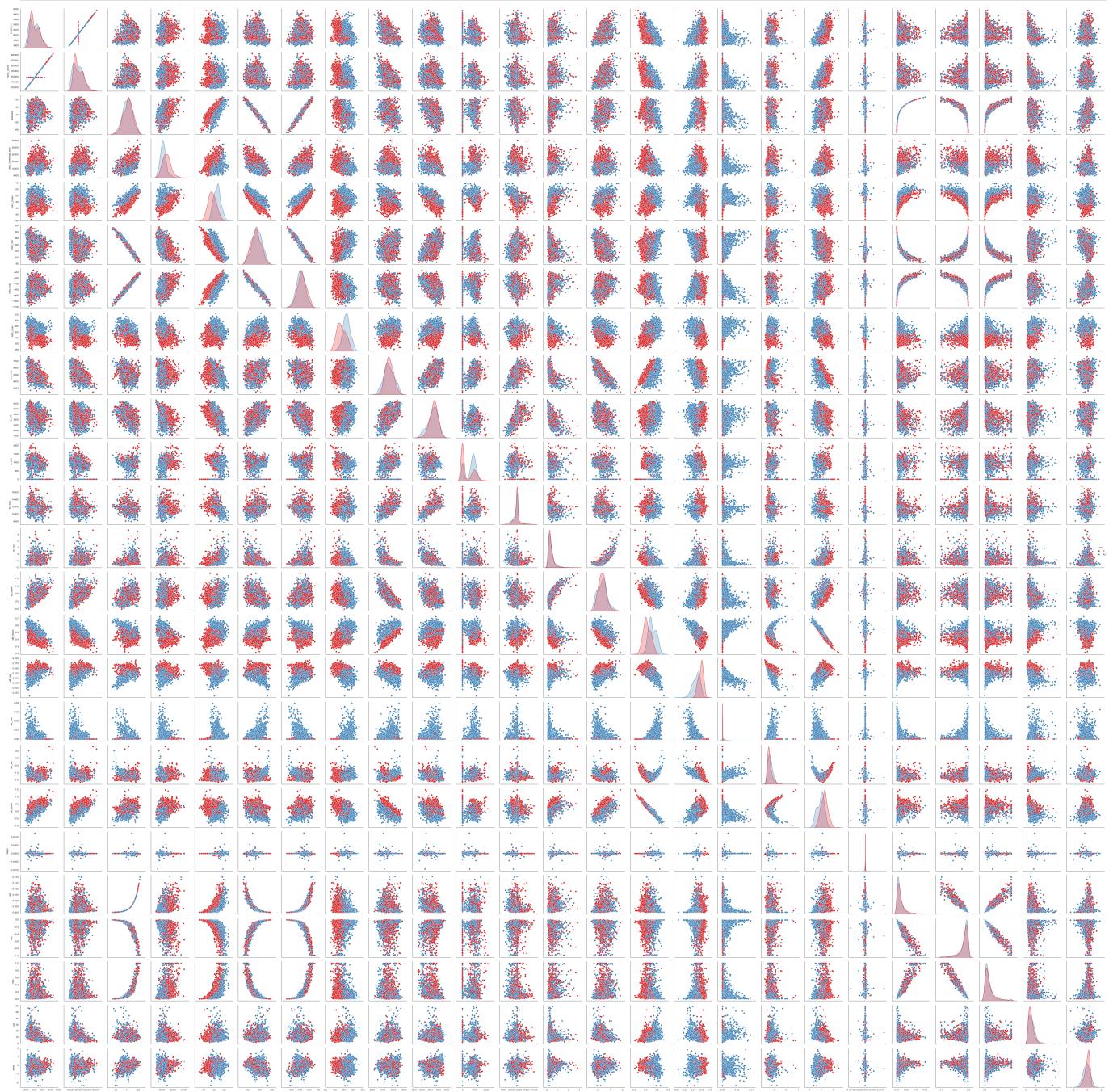


Le distribuzioni sono prevedibilmente **meno nette** nel caso delle emozioni, anche se alcuni sentimenti semanticamente opposti – come 'angry' e 'calm' – sembrano distinguersi in alcune delle variabili (ad esempio: 'std', 'min', 'max').

Queste considerazioni sulle interazioni di 'sex' ed 'emotion' con le variabili continue risultano evidenti anche attraverso la visualizzazione generale fornita dalle **scatter matrix**. Da esse possiamo inoltre notare che esistono **forti legami** tra molti degli attributi; alcuni di questi, come ad esempio quello tra 'intensity' e 'std', verranno esaminati in dettaglio nel seguito.

```
In [41]: num_col_list = [c for c in numerical_columns if c not in
                  ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max']]
sns.pairplot(df, palette='Set1', vars=num_col_list, hue='sex')
```

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



```
In [42]: num_col_list = [c for c in numerical_columns if c not in  
                  ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max']]  
  
sns.pairplot(df, palette='Set1', vars=num_col_list, hue='emotion')  
plt.tight_layout()  
plt.show()
```



Ed attraverso un **plot a coordinate parallele** ('emotion'):

```
In [43]: par_col_list = [c for c in numerical_columns if c not in
                  ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max', 'mean']]

emotions_list = list(df['emotion'].unique())

colormap = plt.colormaps['Set1']
colors_list = [colormap(i) for i in range(len(emotions_list))]

colors_dict = dict(zip(emotions_list, colors_list))
```

```
In [44]: from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[par_col_list].values)

df_numeric_scaled = pd.DataFrame(X_scaled, columns = par_col_list, index = df.index)
df_numeric_scaled['emotion'] = df['emotion']
```

```
In [45]: plt.figure(figsize = (18, 8))

x = range(len(par_col_list))

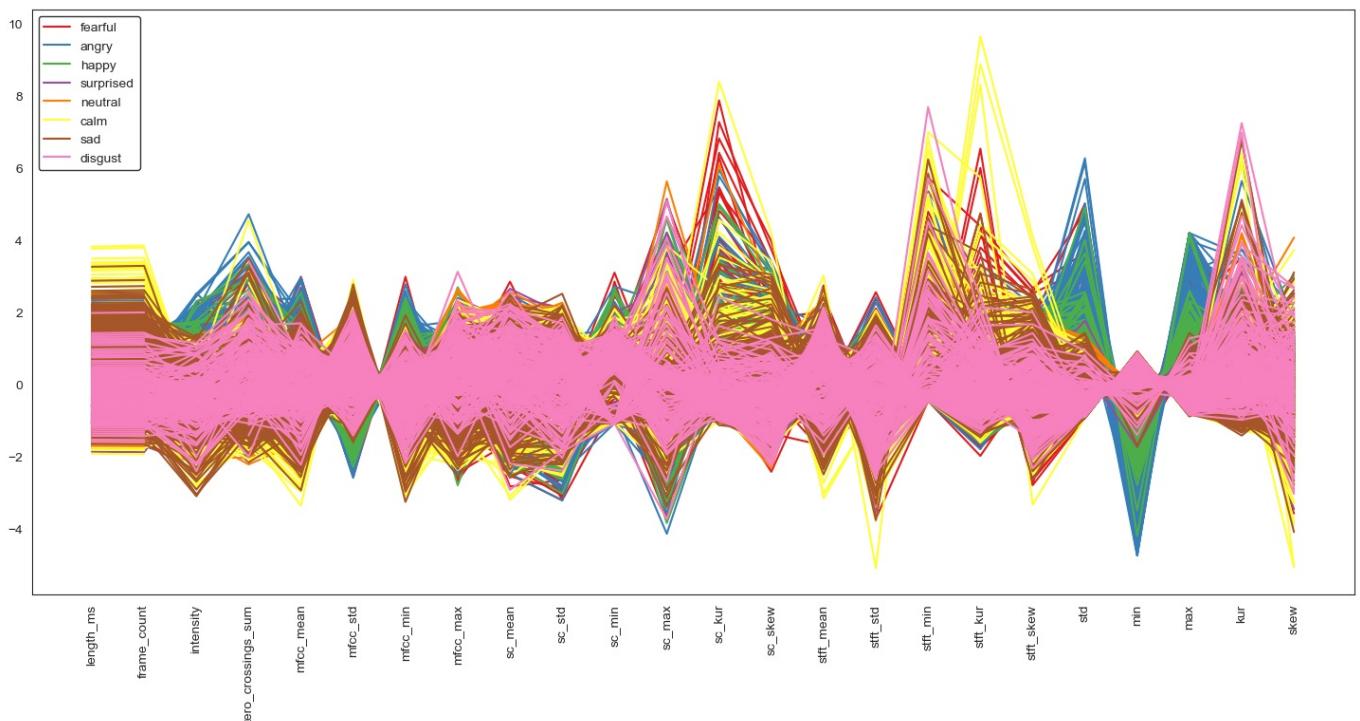
for j in emotions_list:
    df_temp = df_numeric_scaled[df_numeric_scaled['emotion'] == j]
    for i in range(df_temp.shape[0]):
        if i == 0:
            plt.plot( x, df_temp[par_col_list].iloc[i].values, c = colors_dict[j], label = j )
        else:
```

```

plt.plot( x, df_temp[par_col_list].iloc[i].values, c = colors_dict[j] )

plt.xticks(x, par_col_list, rotation = 90)
plt.legend(facecolor = 'white', edgecolor = 'black')
plt.show()

```



Molte delle variabili numeriche rappresentano **famiglie di metriche acustiche**: mfcc (Mel-Frequency Cepstral Coefficients), sc (spectral centroid), stft (stft chromagram); le statistiche riportate nel dataset ne mostrano i minimi, i massimi, le medie, etc. registrati nel corso dei vari sample. Raggruppiamo perciò queste metriche, e ne mostriamo le interazioni con le emozioni tramite dei **box plot**:

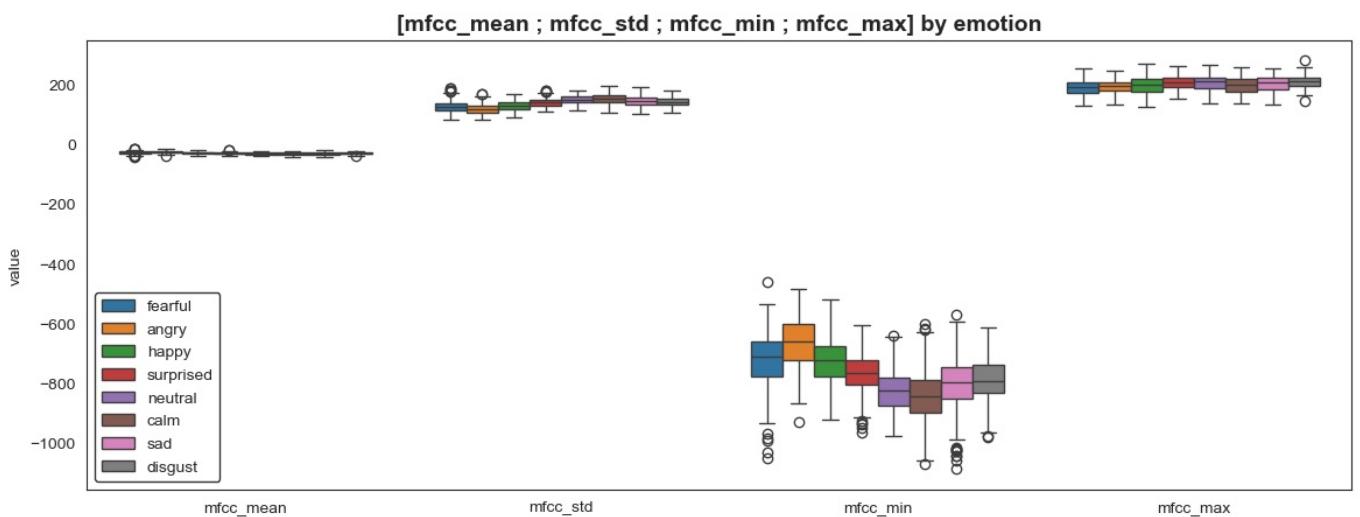
```

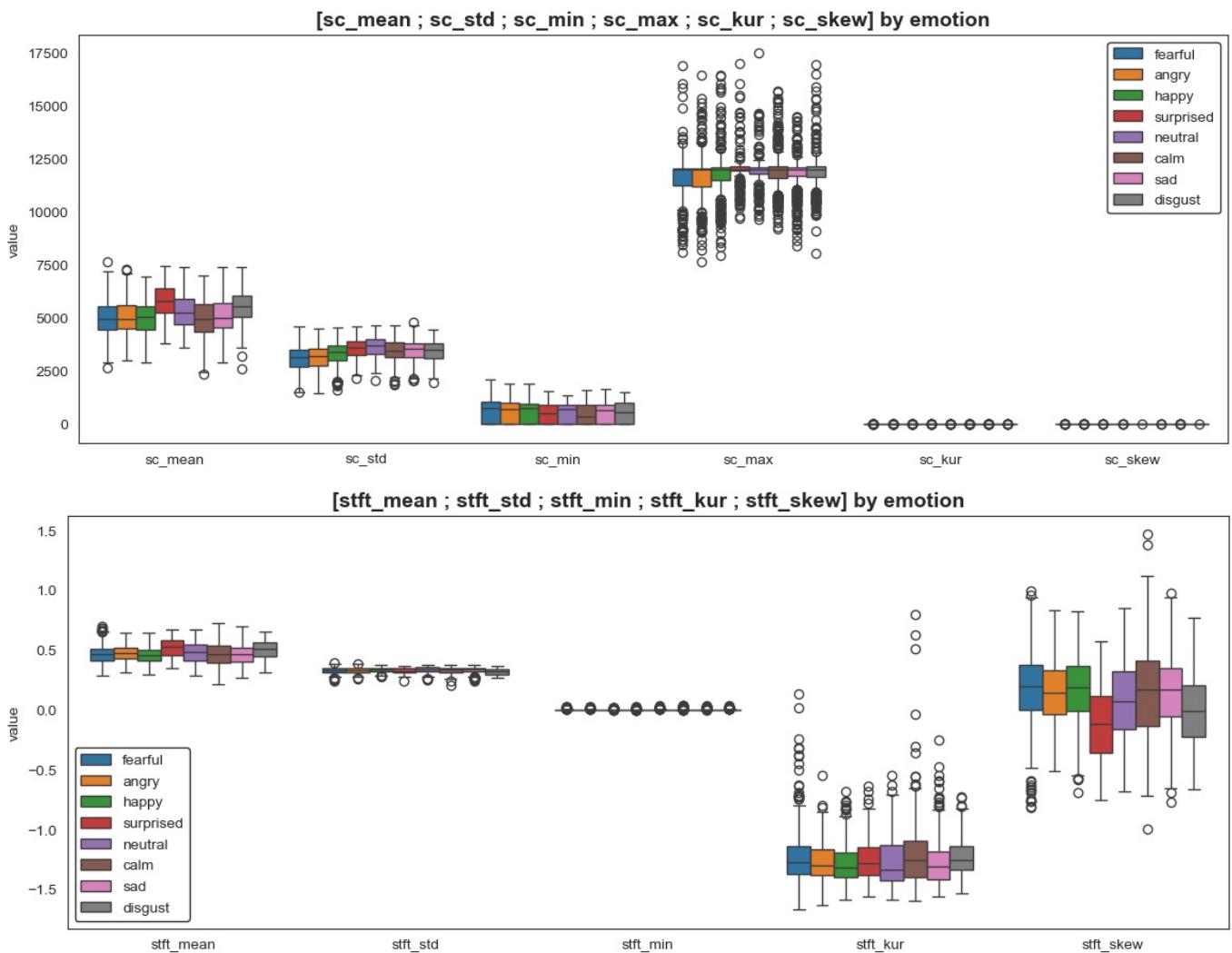
In [46]: target = 'emotion'

mfcc_metrics = ['mfcc_mean', 'mfcc_std', 'mfcc_min', 'mfcc_max']
sc_metrics = ['sc_mean', 'sc_std', 'sc_min', 'sc_max', 'sc_kur', 'sc_skew']
stft_metrics = ['stft_mean', 'stft_std', 'stft_min', 'stft_kur', 'stft_skew']

for metrics in [mfcc_metrics, sc_metrics, stft_metrics]:
    fig, ax = plt.subplots(figsize = (14, 5))
    df_temp = df[metrics + [target]].melt(id_vars = target)
    sns.boxplot(x = 'variable', y = 'value', hue = target, data = df_temp, ax = ax)
    ax.set_title('[' + ' ; '.join(metrics) + '] by ' + target, fontsize = 14, fontweight='bold')
    plt.legend(facecolor = 'white', edgecolor = 'black', fontsize = 10)
    plt.xlabel('')
    plt.show()

```





Come visualizzazione conclusiva di questa sezione, usiamo dei **violin plot** per mostrare le relazioni tra 'emotion' e 'sex', rispetto a tutte le variabili continue:

```
In [47]: hue = 'sex'
target = 'emotion'

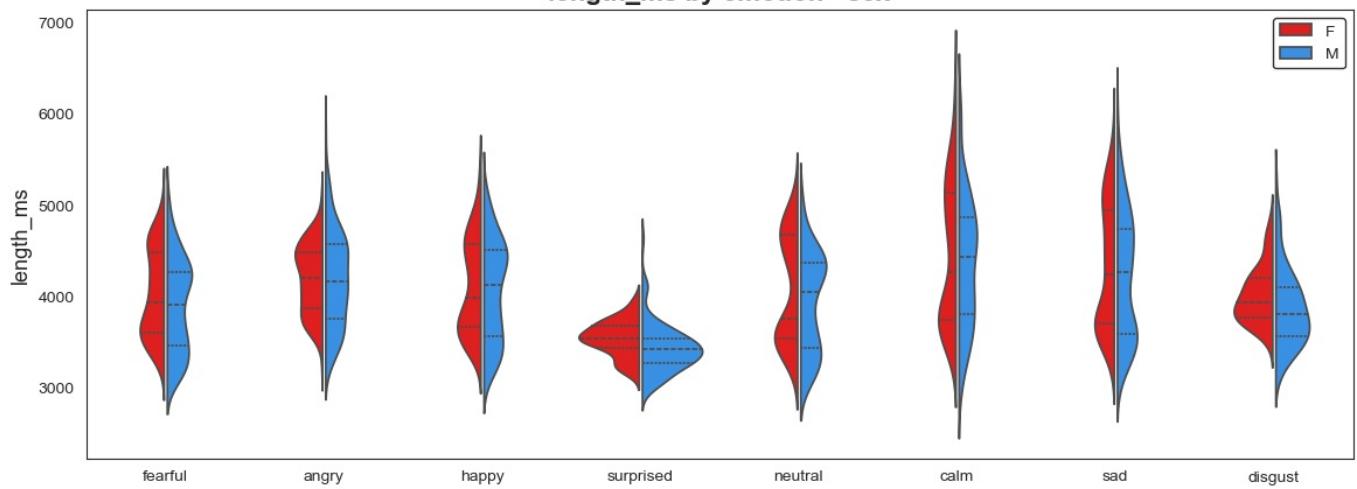
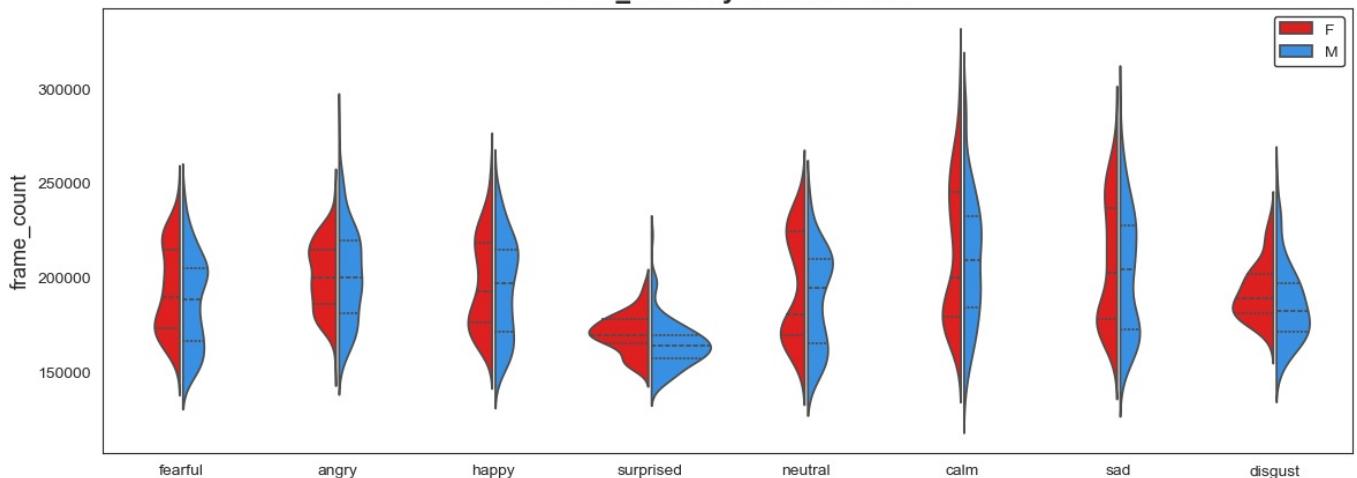
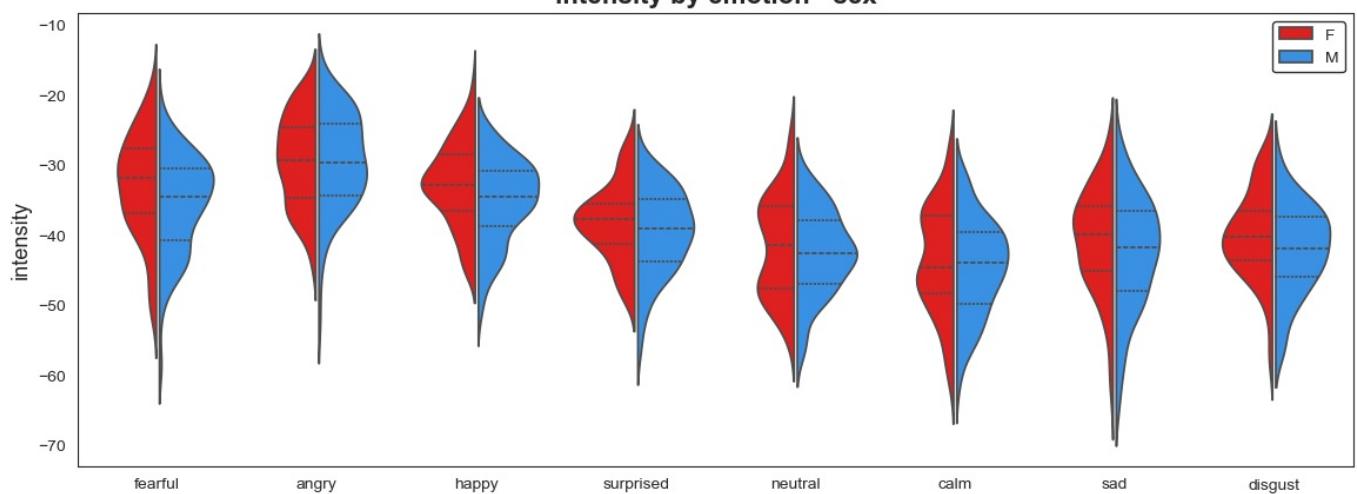
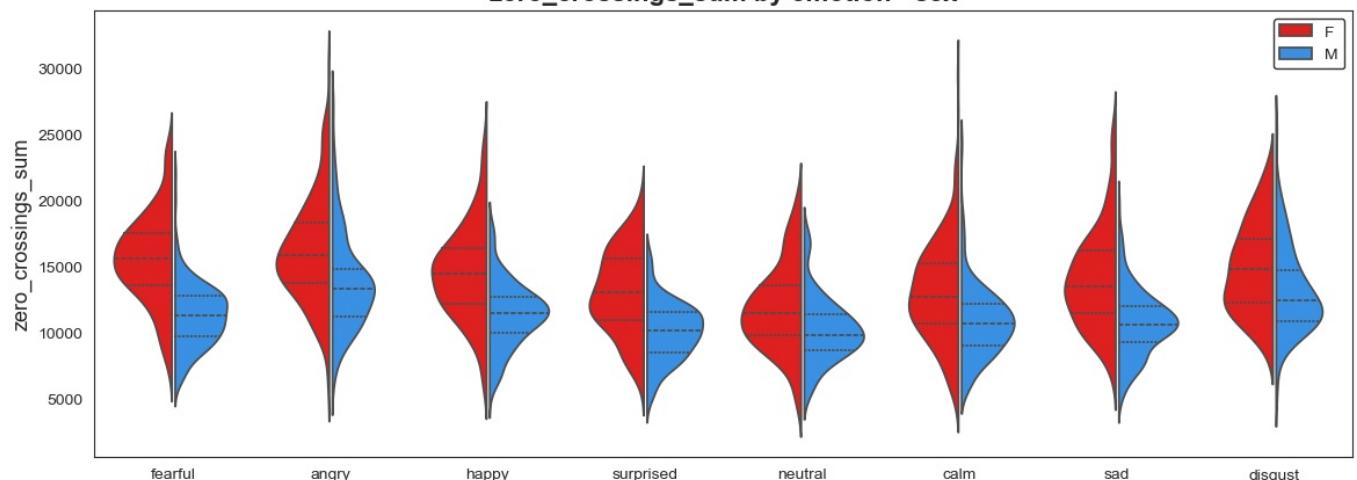
palette = {'M': 'dodgerblue', 'F': 'red'}

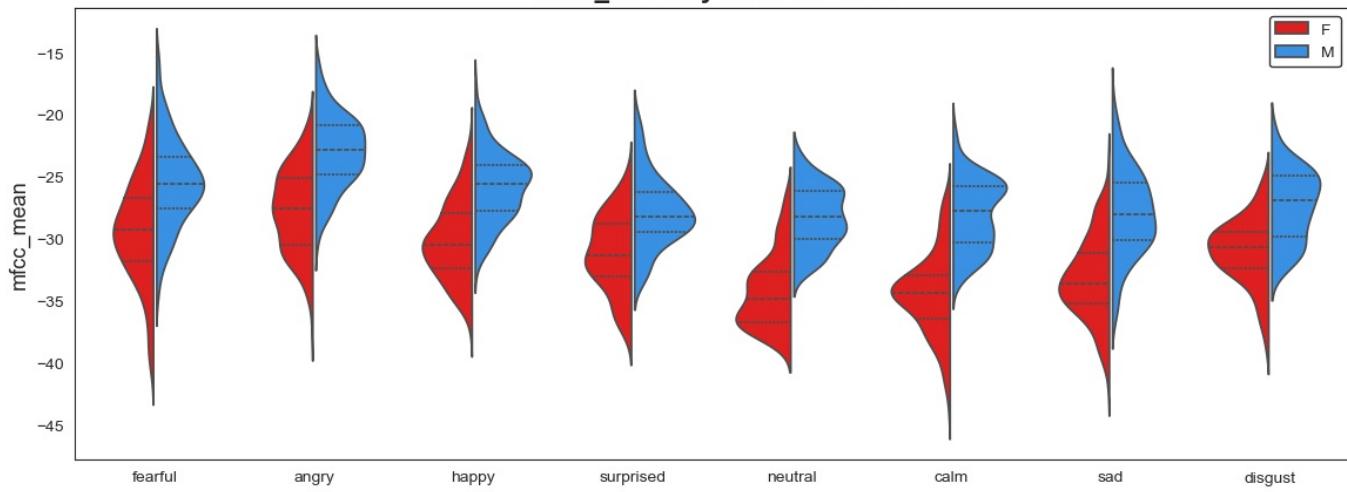
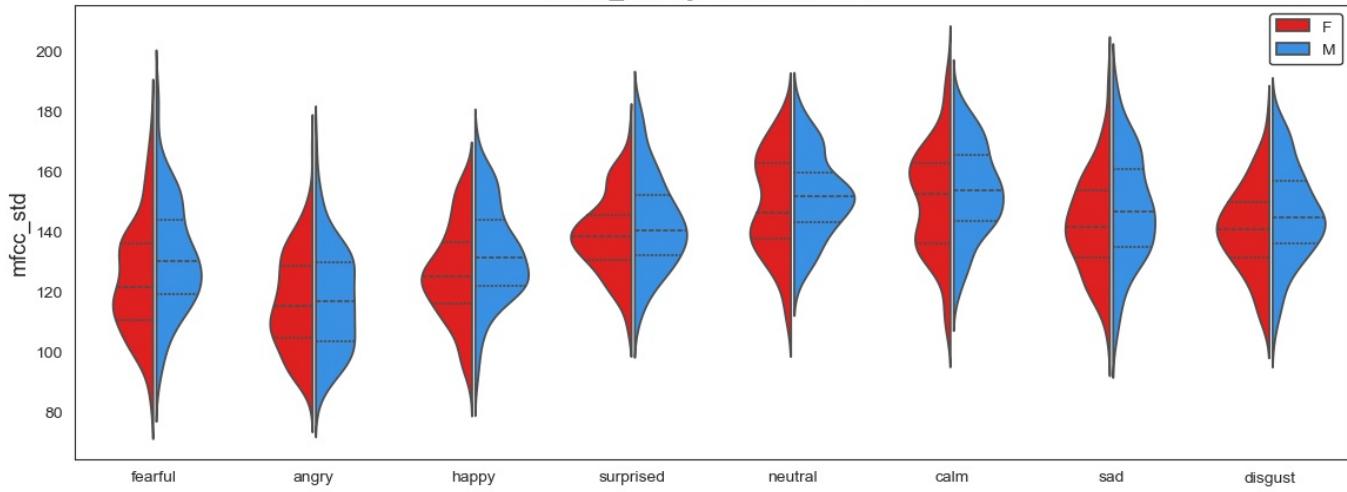
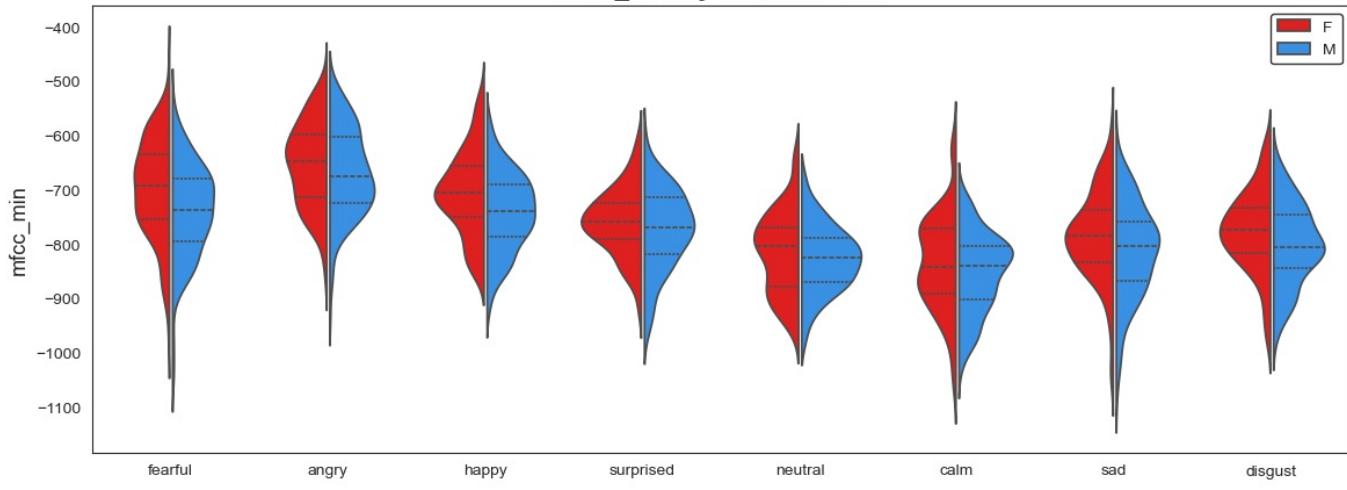
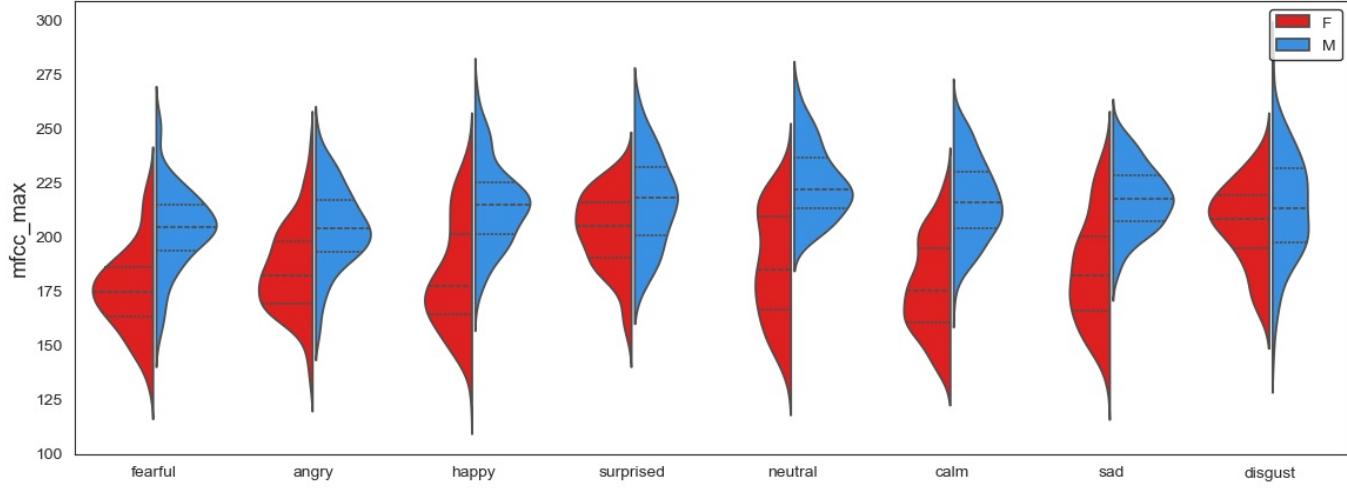
for col in num_col_list:

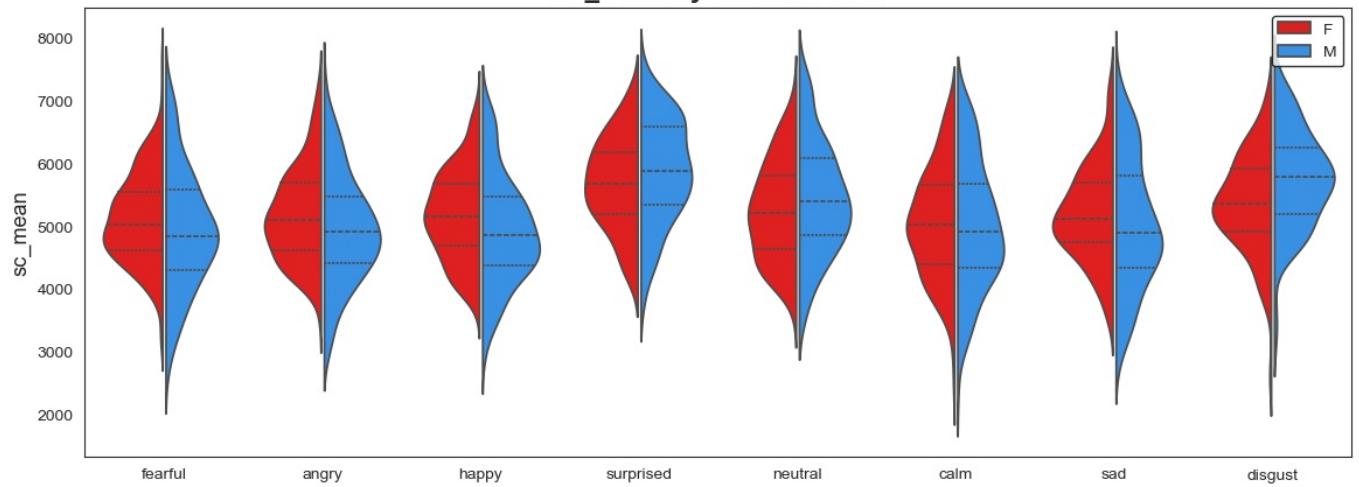
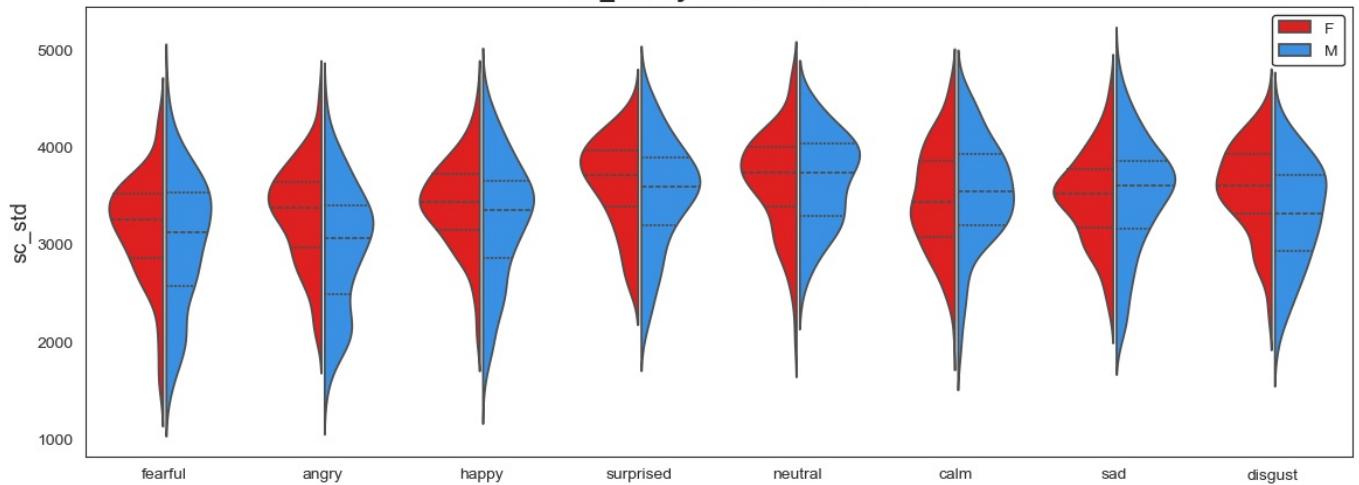
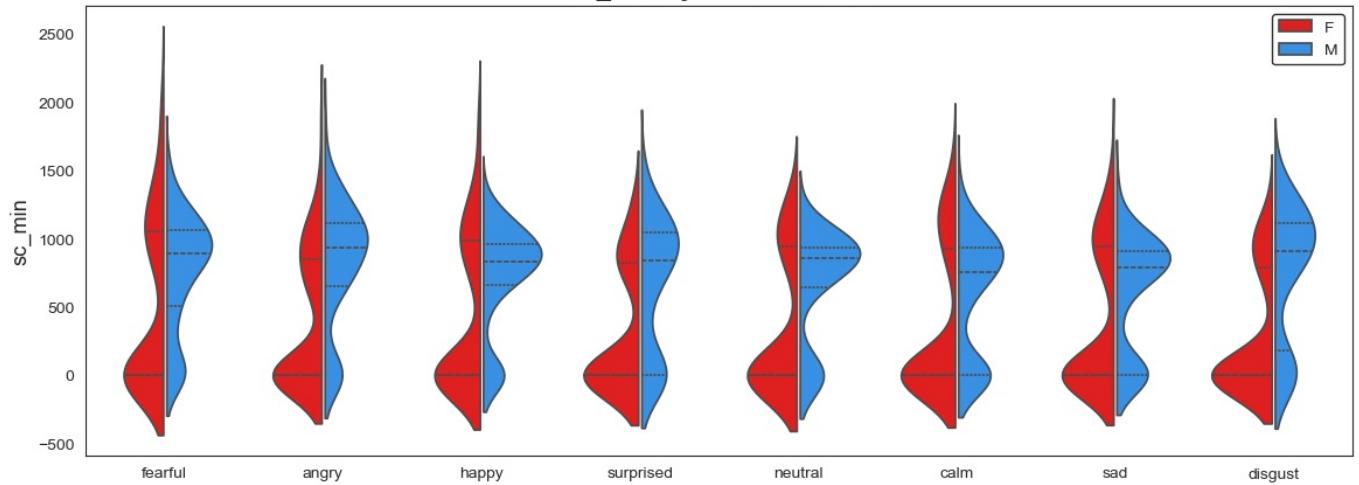
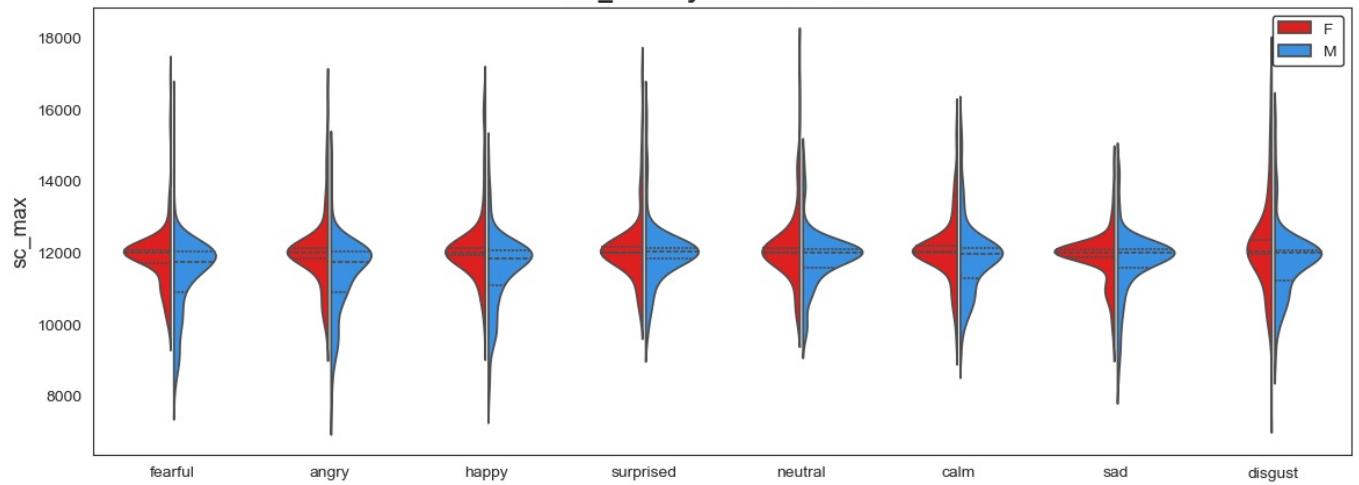
    fig, ax = plt.subplots(figsize = (14, 5))

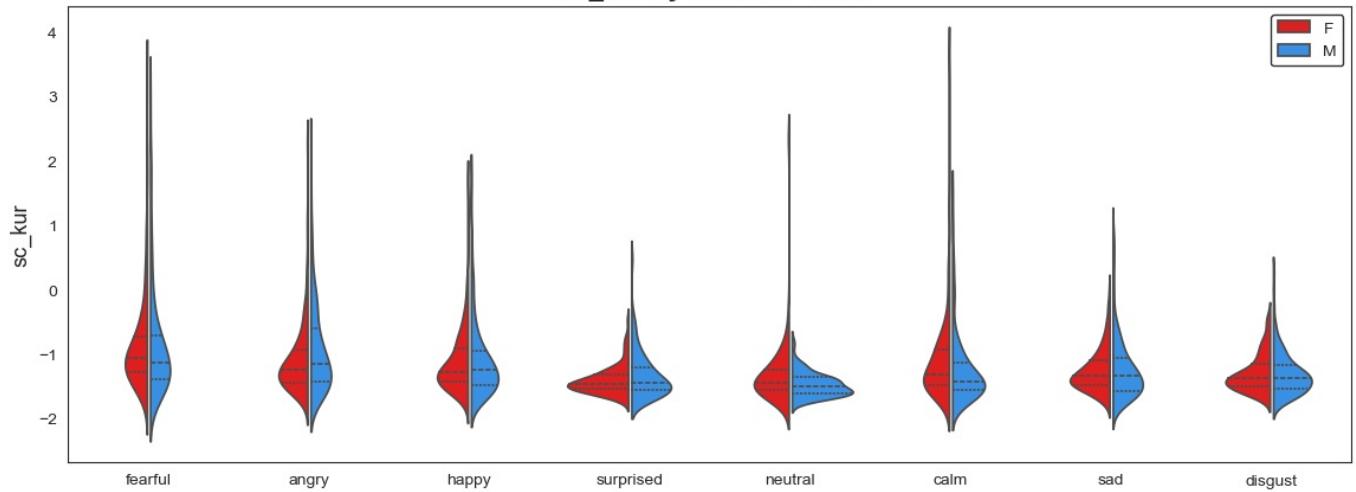
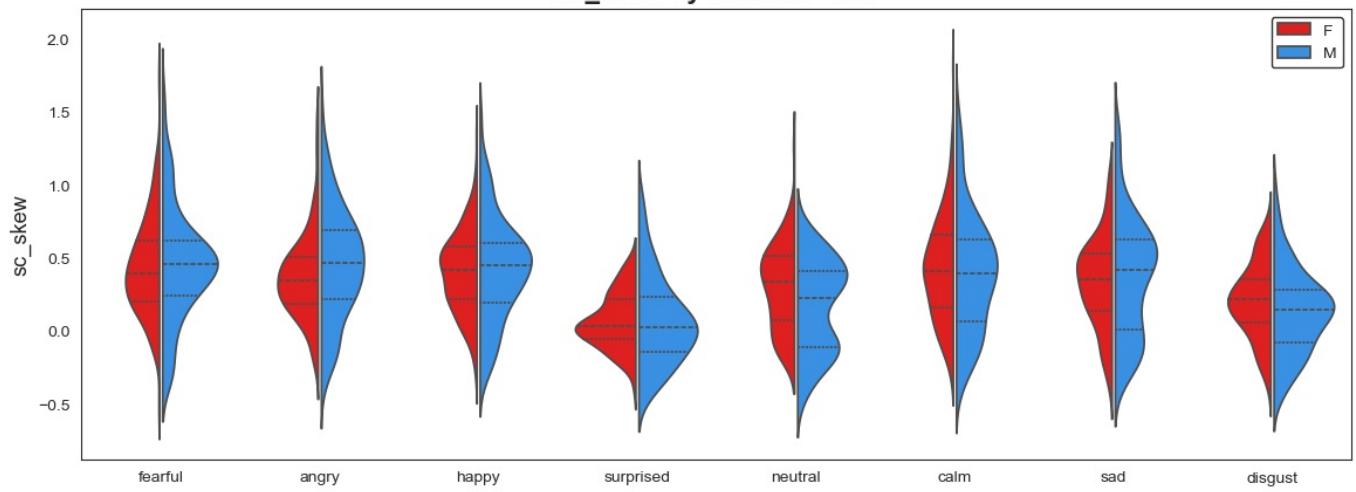
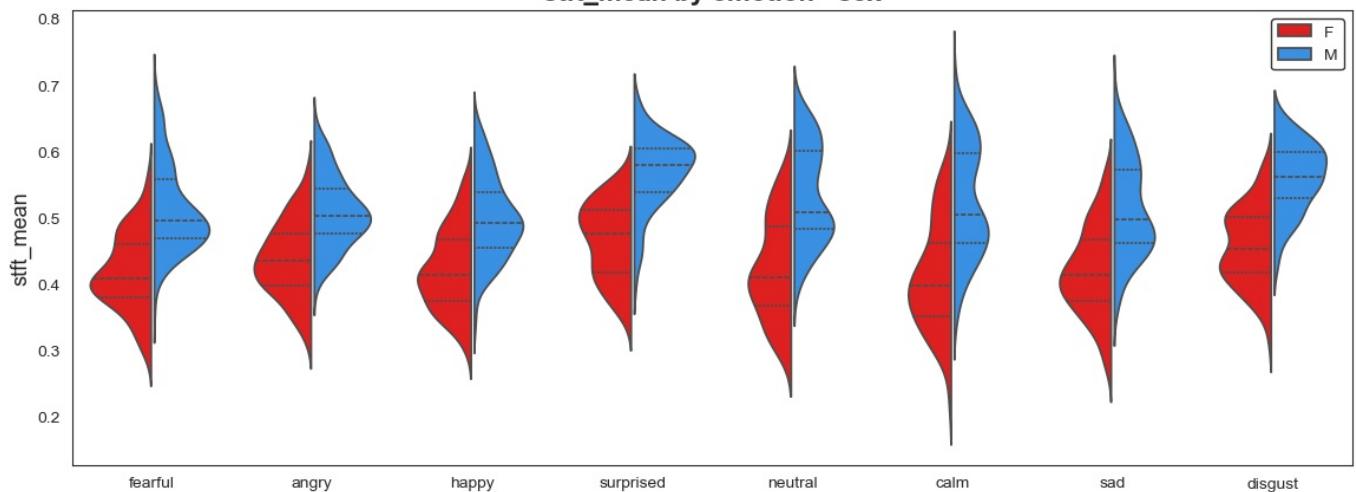
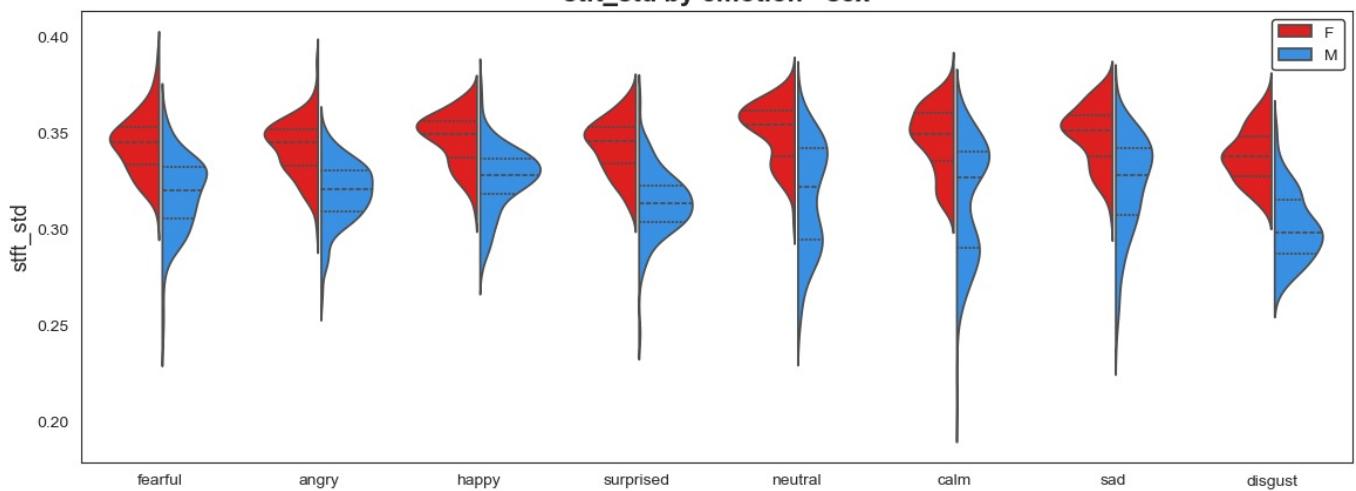
    df_temp = df[[col, target, hue]].melt(id_vars = [target, hue])

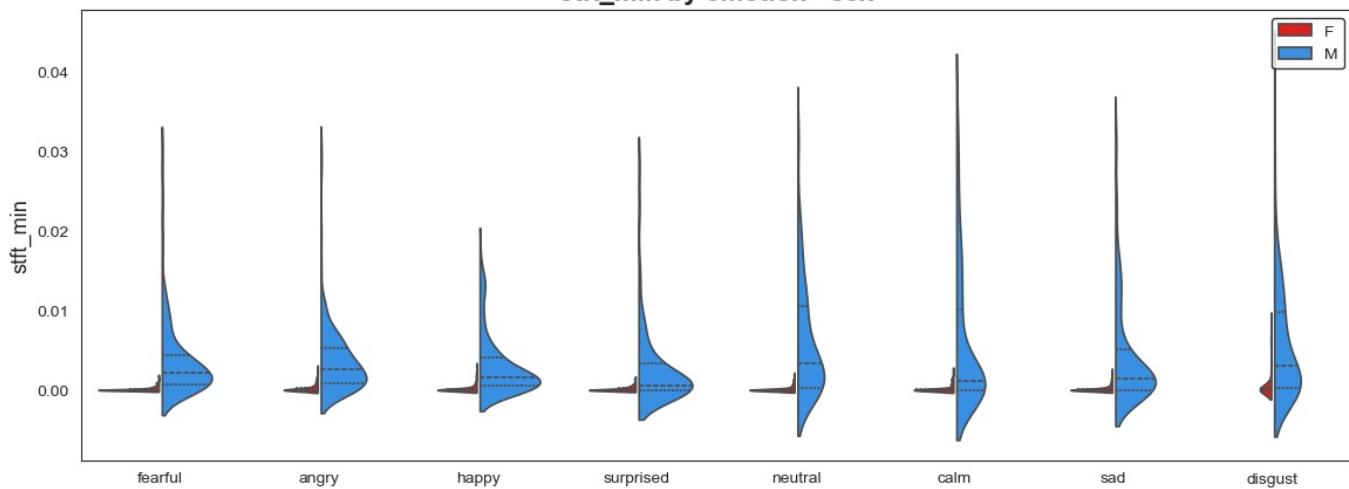
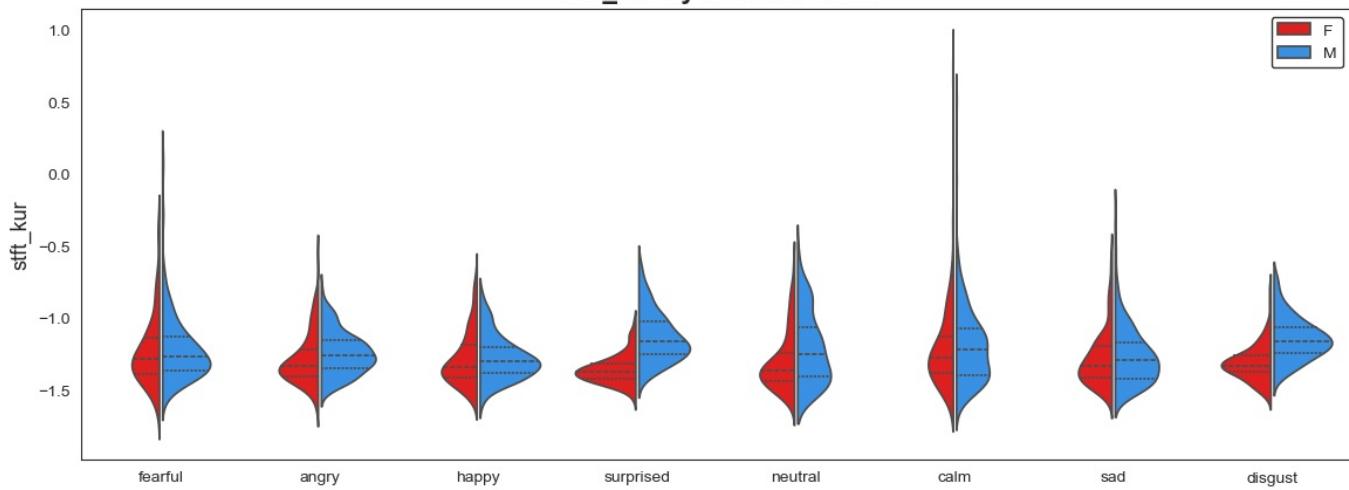
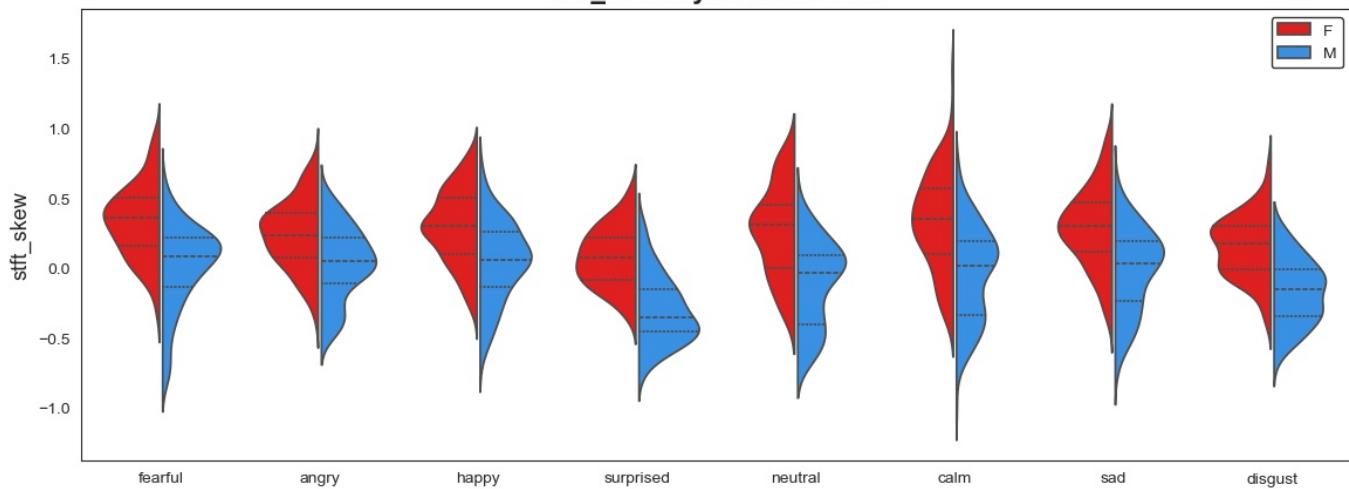
    sns.violinplot(x = target, y = 'value', hue = hue, data = df_temp, split = True,
                    inner = 'quart', gap = 0.05, palette=palette, ax = ax)
    ax.set_title(f'{col} by {target} - {hue}', fontsize = 16, fontweight='bold')
    plt.legend(facecolor = 'white', edgecolor = 'black', fontsize = 10)
    ax.set_ylabel(col, fontsize = 13)
    plt.xlabel('')
    plt.show()
```

length_ms by emotion - sex**frame_count by emotion - sex****intensity by emotion - sex****zero_crossings_sum by emotion - sex**

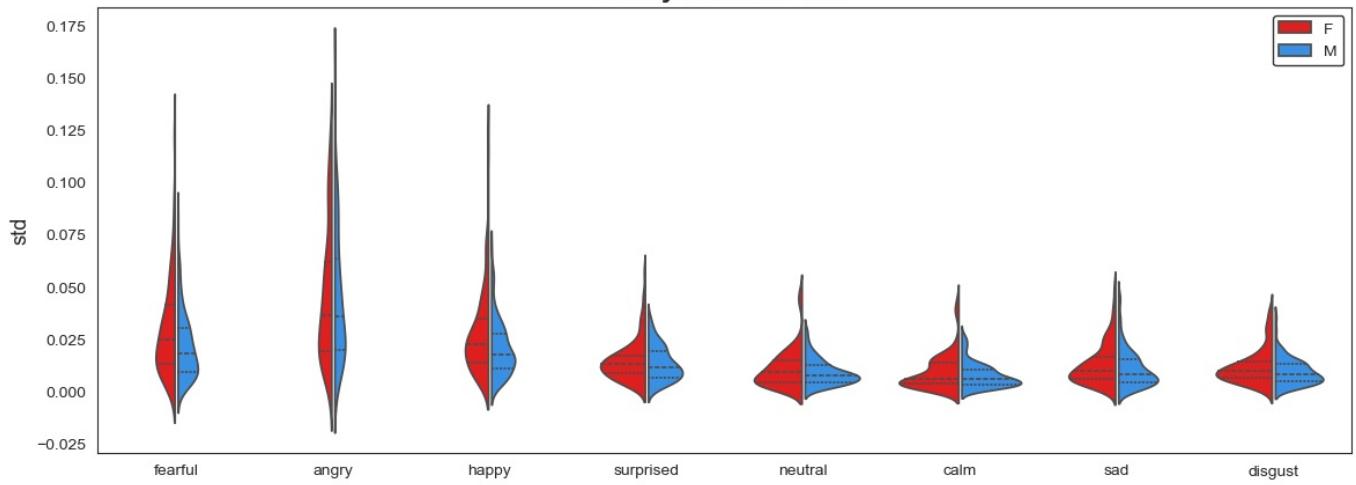
mfcc_mean by emotion - sex**mfcc_std by emotion - sex****mfcc_min by emotion - sex****mfcc_max by emotion - sex**

sc_mean by emotion - sex**sc_std by emotion - sex****sc_min by emotion - sex****sc_max by emotion - sex**

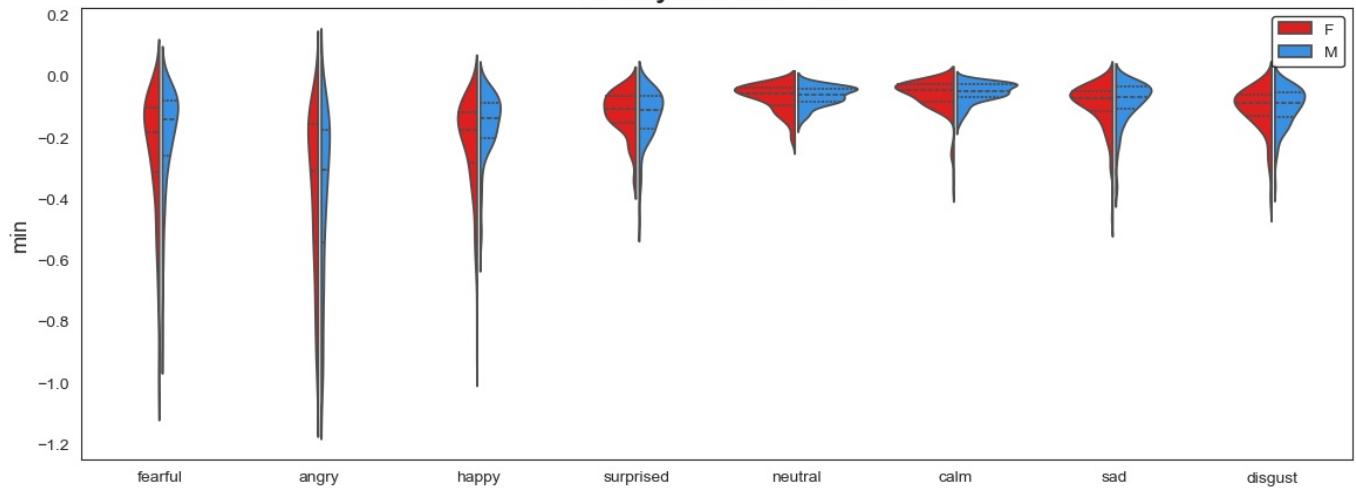
sc_kur by emotion - sex**sc_skew by emotion - sex****stft_mean by emotion - sex****stft_std by emotion - sex**

stft_min by emotion - sex**stft_kur by emotion - sex****stft_skew by emotion - sex****mean by emotion - sex**

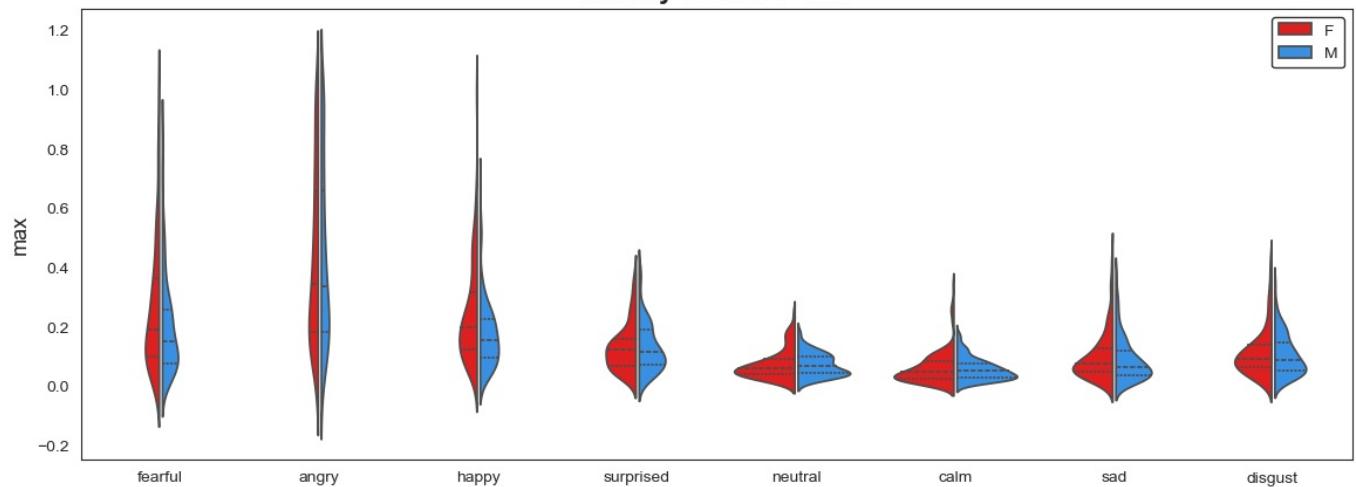
std by emotion - sex



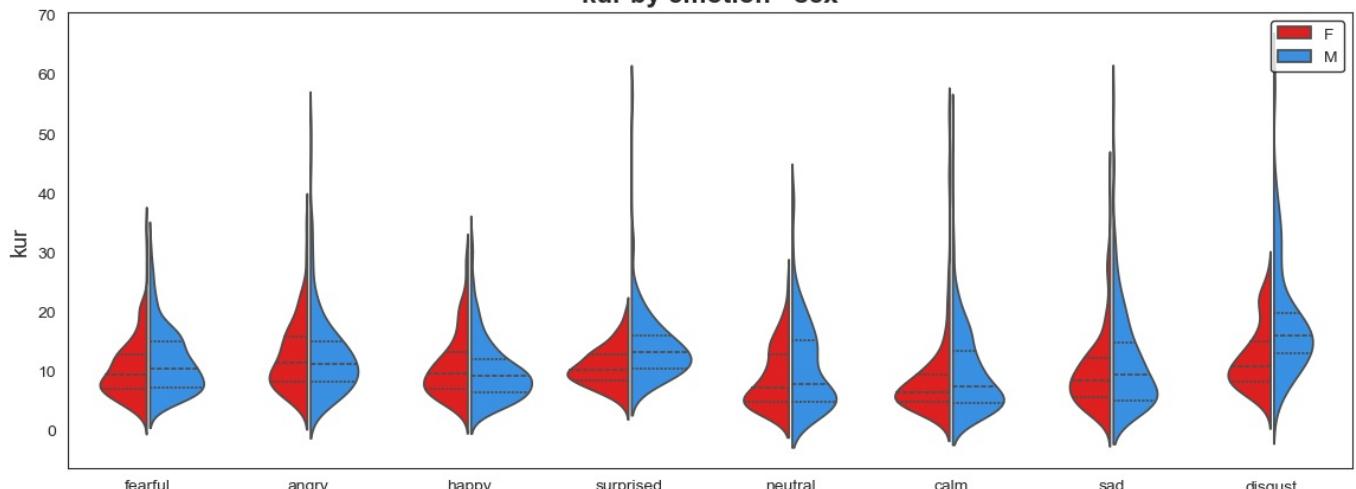
min by emotion - sex

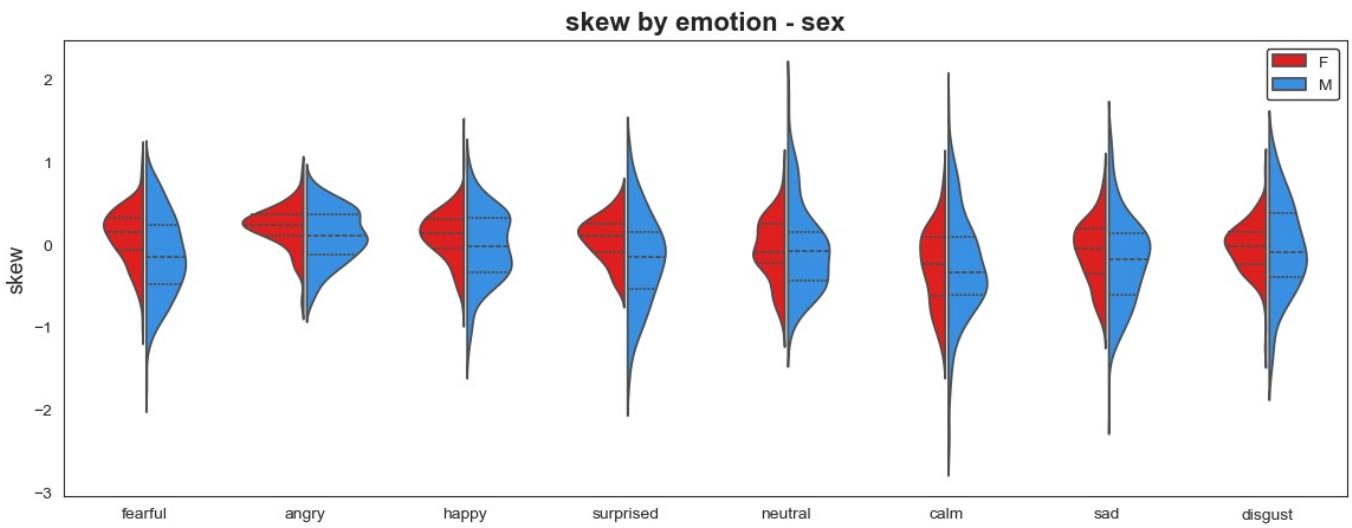


max by emotion - sex



kur by emotion - sex





3. Assessing data quality & variable transformations

Errori nel dataset: come mostrato sopra, la variabile 'frame_count' presentava alcuni valori inferiori < 0; dato il suo significato, e il resto della sua distribuzione, questi possono essere considerati errori.

Inaccuratezze sintattiche: nessuna delle variabili categoriche sembra presentare inaccuratezze sintattiche; possiamo verificarlo mostrandone i valori unici (ovvero, ad esempio, nessuna emozione ha refusi nel nome):

```
In [48]: for column in non_numerical_columns:
    print(column, ":", non_numerical_columns[column].unique(), sep="")
```

```
modality: ['audio-only']
vocal_channel: ['speech' nan 'song']
emotion: ['fearful' 'angry' 'happy' 'surprised' 'neutral' 'calm' 'sad' 'disgust']
emotional_intensity: ['normal' 'strong']
statement: ['Dogs are sitting by the door' 'Kids are talking by the door']
repetition: ['2nd' '1st']
sex: ['F' 'M']
```

Le inconsistenze semantiche sono difficili da individuare in questo dataset (non avendo, ad esempio, il nome degli attori). Allo stesso modo risulta arduo esprimersi sulla completezza dei dati; possiamo però notare, come già accennato in precedenza, che le variabili categoriche sembrano ben bilanciate ('M' vs. 'F', 'speech' vs. 'song', etc.). Inoltre, possiamo assicurarsi che il dataset non presenti **duplicati**:

```
In [49]: len(df)
```

```
Out[49]: 2452
```

```
In [50]: df = df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=True)
```

```
In [51]: len(df)
```

```
Out[51]: 2452
```

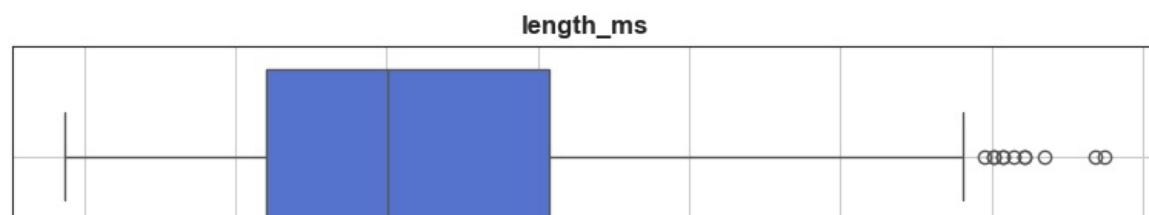
Attraverso l'uso di boxplot, possiamo mostrare che molte delle variabili continue presentano degli **outliers**:

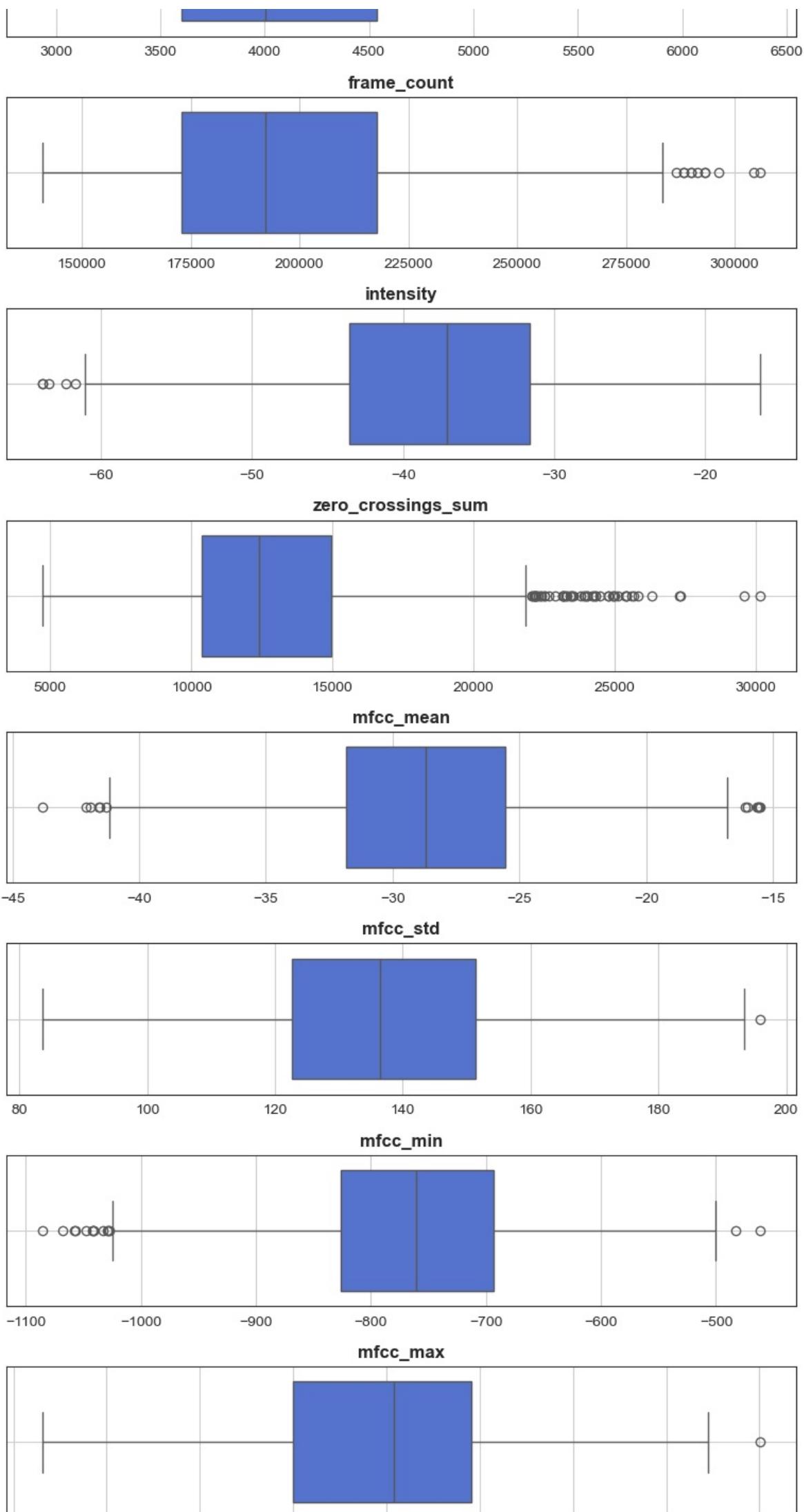
```
In [52]: box_col = [c for c in numerical_columns.columns if c not in
              ['actor', 'channels', 'sample_width', 'frame_rate', 'frame_width', 'stft_max']]

fig, axs = plt.subplots(len(box_col), 1, figsize=(8, len(box_col)*2))

for i, col in enumerate(box_col):
    ax = sns.boxplot(data=df, x=col, ax=axs[i], color='royalblue')
    ax.set_title(f'{col}', fontweight='bold')
    ax.set_xlabel('')
    ax.grid(True)

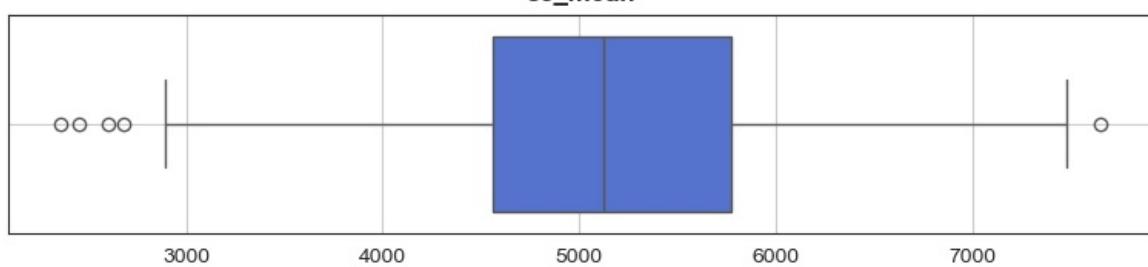
plt.tight_layout()
plt.show()
```



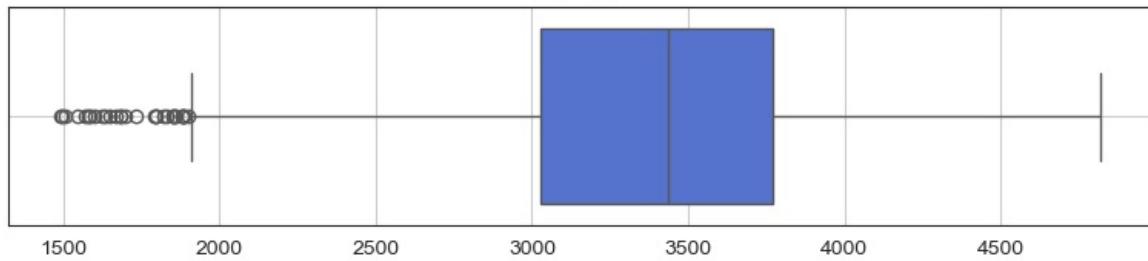


120 140 160 180 200 220 240 260 280

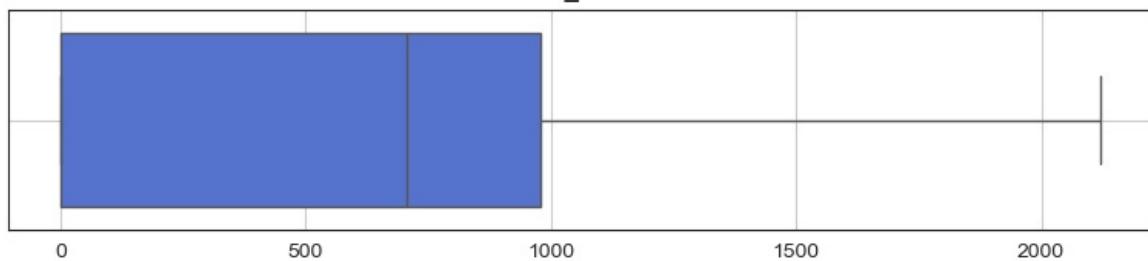
sc_mean



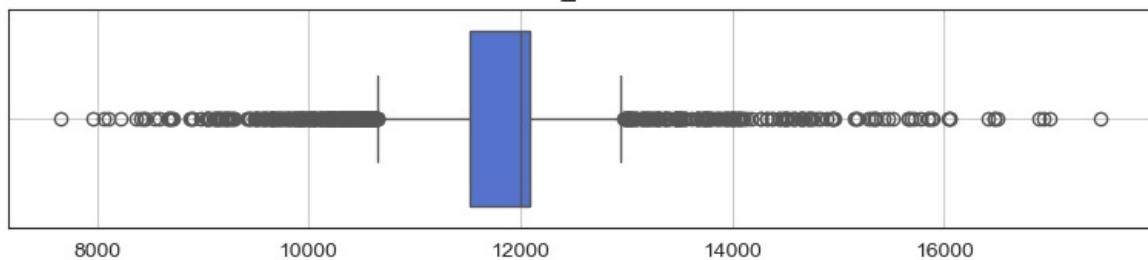
sc_std



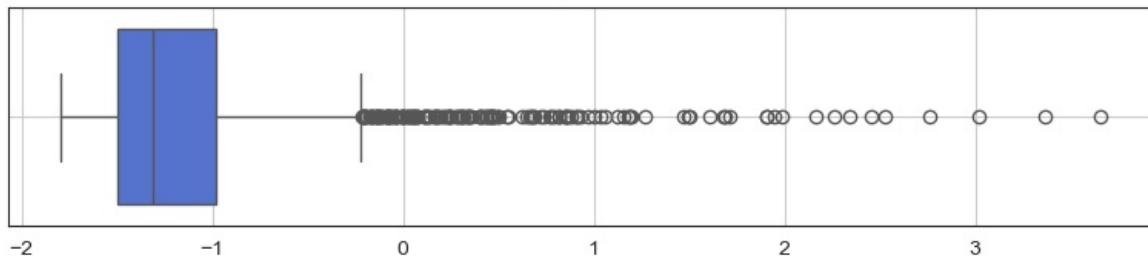
sc_min



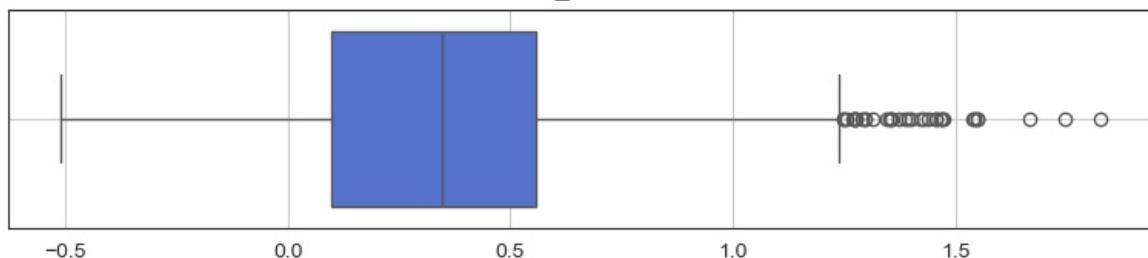
sc_max



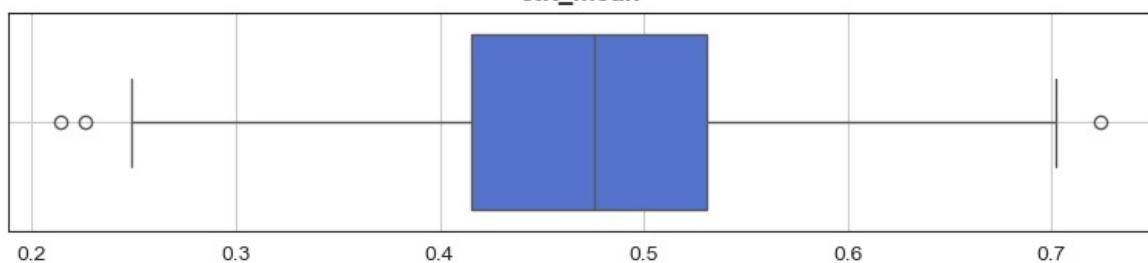
sc_kur



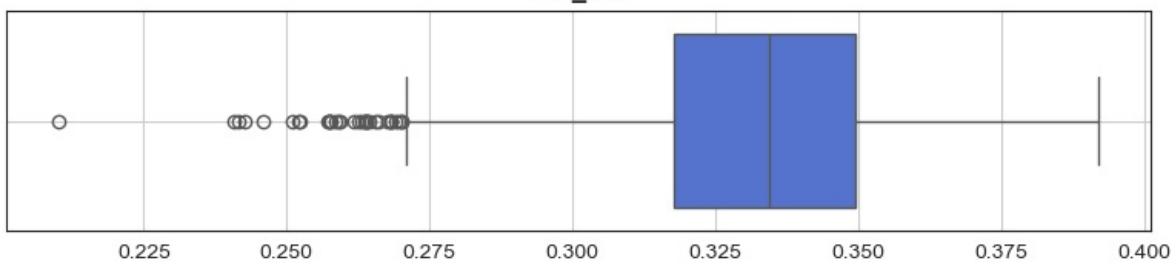
sc_skew



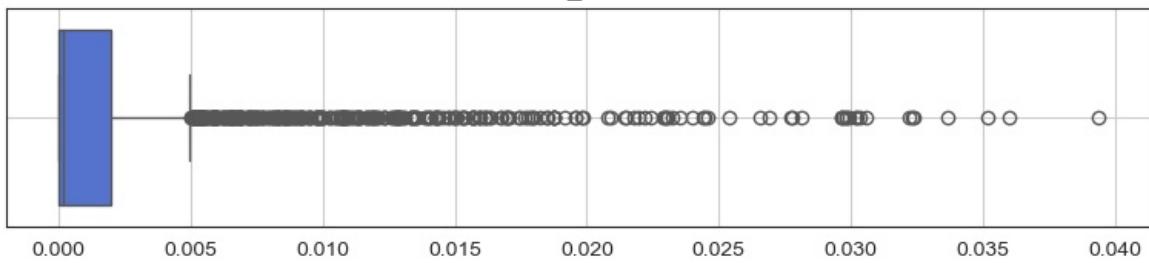
stft_mean



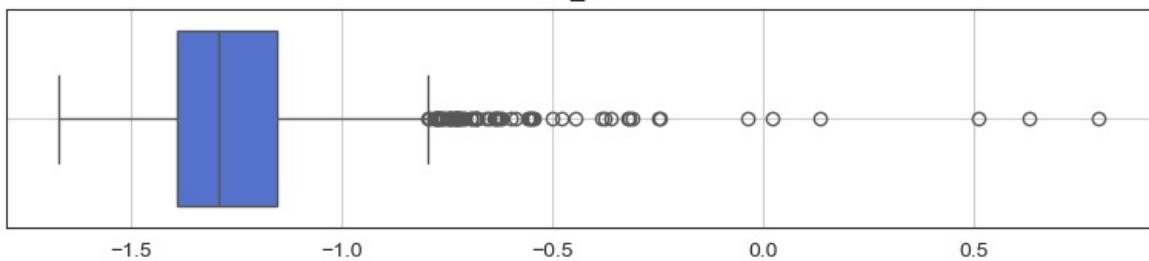
stft_std



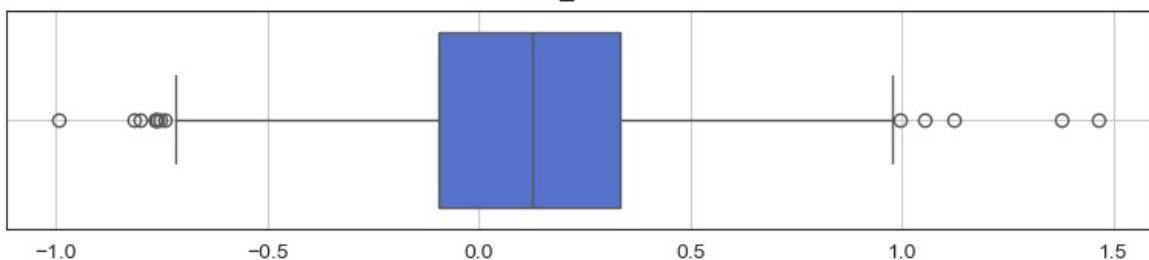
stft_min



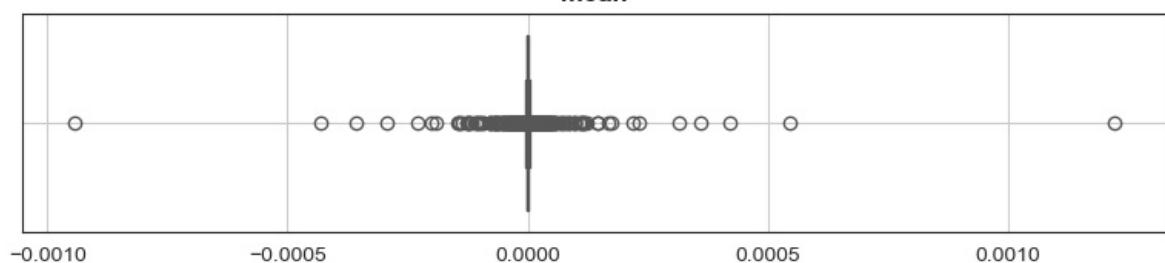
stft_kur



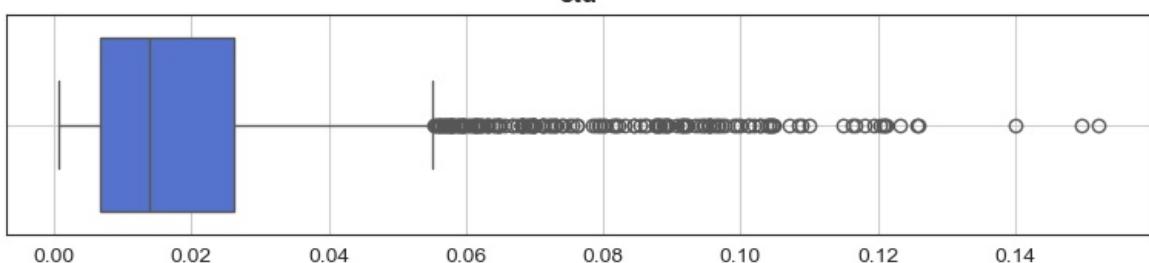
stft_skew



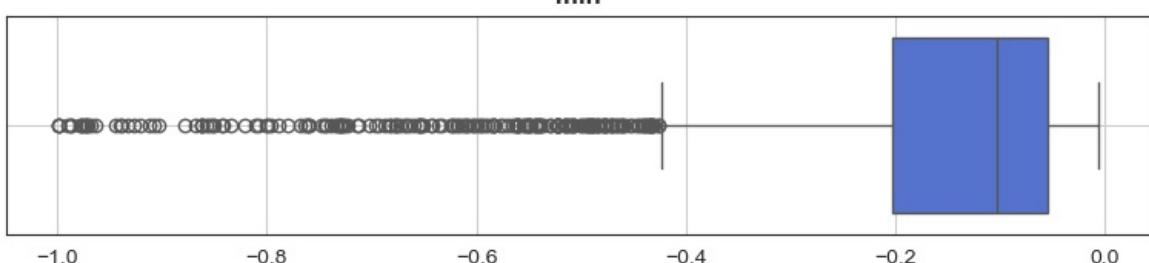
mean



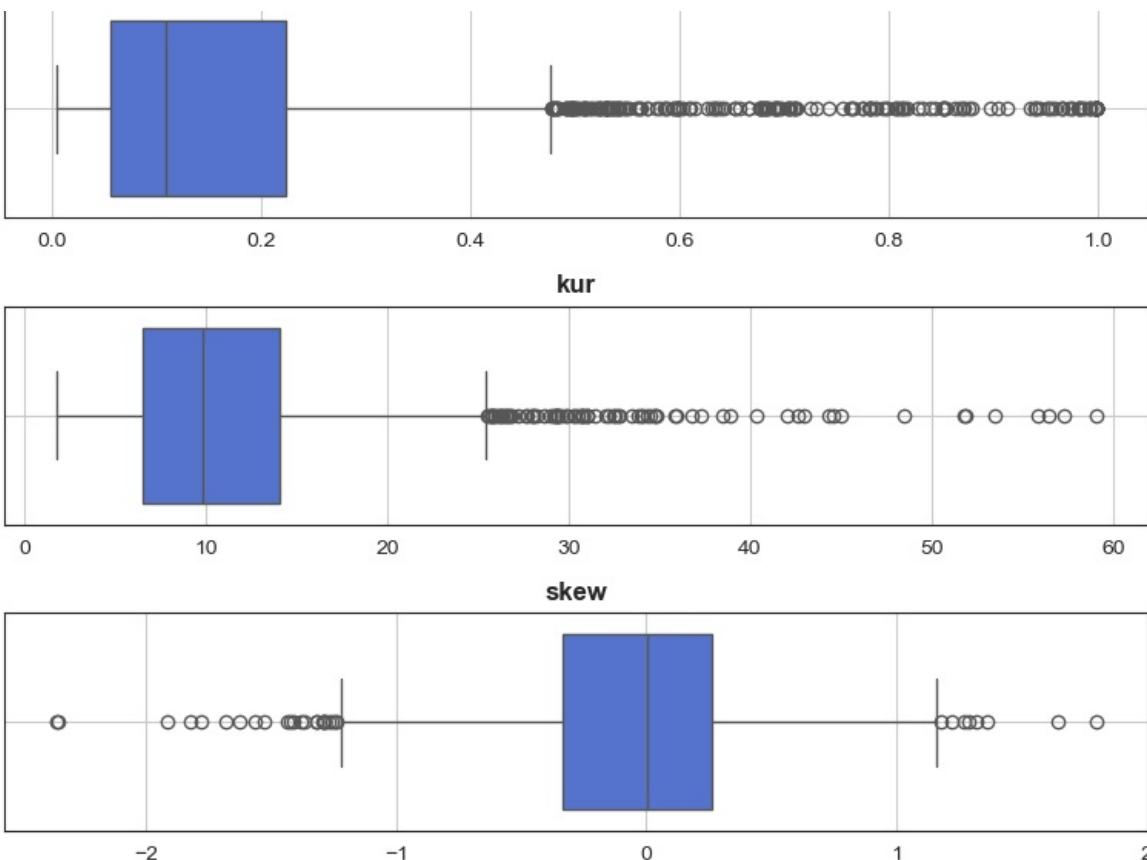
std



min



max



Troviamo gli outlier individuando i dati che sono più di **1.5 volte l'IQR sotto al primo quartile o sopra al terzo**, rispetto a ogni variabile continua:

```
In [53]: outliers_dict = {}

for column in numerical_columns.columns:
    feature = numerical_columns[column]

    q1 = np.quantile(feature, 0.25)
    q3 = np.quantile(feature, 0.75)
    iqr = q3 - q1

    outliers = []
    for v in feature:
        if v < q1 - 1.5 * iqr or v > q3 + 1.5 * iqr:
            outliers.append(True)
        else:
            outliers.append(False)

    outliers_dict[column] = outliers
```

Conteggio degli outlier per ogni feature:

```
In [54]: for column, outlier_value in outliers_dict.items():
    counter = sum(outlier_value)
    print(column, ":", counter, sep="")
```

```
actor: 0
channels: 6
sample_width: 0
frame_rate: 0
frame_width: 6
length_ms: 11
frame_count: 46
intensity: 0
zero_crossings_sum: 52
mfcc_mean: 12
mfcc_std: 1
mfcc_min: 14
mfcc_max: 1
sc_mean: 5
sc_std: 30
sc_min: 0
sc_max: 439
sc_kur: 152
sc_skew: 35
stft_mean: 3
stft_std: 33
stft_min: 336
stft_max: 0
stft_kur: 69
stft_skew: 14
mean: 536
std: 172
min: 207
max: 207
kur: 85
skew: 32
```

Non conoscendo nel dettaglio la semantica delle variabili, non siamo in grado di discernere se gli outlier siano errori nel dataset o data point genuini di cui dobbiamo tenere conto; per evitare di scartare record che dovremmo invece prendere in considerazione, e puntando a un'analisi più completa possibile, decidiamo quindi di tenerli, e valutare poi il loro impatto attraverso tecniche che risentono in diversa misura della loro presenza. Gli outlier individuati sopra sono comunque solo relativi alle variabili prese singolarmente; servirà quindi un'analisi multidimensionale per trovare i data point anomali rispetto a più attributi.

Per quanto riguarda i **missing values**, invece, questi sembrano essere effettive mancanze nel dataset, e non semplicemente dati a cui l'attributo in questione non si applica. Le colonne che presentano valori mancanti sono tre:

```
In [55]: df.isnull().sum()
```

```
Out[55]: 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 [56]: df.isnull().sum() / len(df)
```

```
Out[56]: modality      0.00000  
vocal_channel  0.07993  
emotion        0.00000  
emotional_intensity 0.00000  
statement      0.00000  
repetition     0.00000  
actor          0.45922  
sex            0.00000  
channels        0.00000  
sample_width    0.00000  
frame_rate      0.00000  
frame_width     0.00000  
length_ms       0.00000  
frame_count     0.00000  
intensity       0.33279  
zero_crossings_sum 0.00000  
mfcc_mean       0.00000  
mfcc_std        0.00000  
mfcc_min        0.00000  
mfcc_max        0.00000  
sc_mean          0.00000  
sc_std           0.00000  
sc_min           0.00000  
sc_max           0.00000  
sc_kur           0.00000  
sc_skew          0.00000  
stft_mean        0.00000  
stft_std         0.00000  
stft_min         0.00000  
stft_max         0.00000  
stft_kur         0.00000  
stft_skew        0.00000  
mean             0.00000  
std              0.00000  
min              0.00000  
max              0.00000  
kur              0.00000  
skew             0.00000  
dtype: float64
```

Come già accennato, la variabile 'actor' è un identificativo; non ha alcun ruolo nei nostri modelli, quindi possiamo accantonarla per il

momento. Passiamo invece a 'vocal_channel':

```
In [57]: df['vocal_channel'].value_counts(normalize=True)
```

```
Out[57]: vocal_channel
speech    0.59176
song      0.40824
Name: proportion, dtype: float64
```

I valori mancanti della variabile sono relativamente pochi, quindi non c'è un grosso rischio di modificarne la distribuzione originaria. Abbiamo perciò deciso di sostituire i missing values sfruttando la **proporzione dei suoi due valori unici**:

```
In [58]: vc_proportion = df['vocal_channel'].value_counts(normalize=True)
vc_null_num = df['vocal_channel'].isnull().sum()
replacements = np.random.choice(vc_proportion.index, size=vc_null_num, p=vc_proportion.values)

df.loc[df['vocal_channel'].isnull(), 'vocal_channel'] = replacements
```

La situazione è molto diversa per 'intensity'; la variabile riporta un totale di 816 missing values, che corrispondono al 33,28% delle osservazioni iniziali. Pertanto, data la numerosità del campione mancante, abbiamo cercato di effettuare il replacing attraverso un **modello di regressione**. La scelta è supportata dal fatto che altre strategie, come ad esempio il sampling da una distribuzione di probabilità, avrebbero potuto portare ad una sostituzione imprecisa.

Come features del regressore abbiamo deciso di utilizzare tutti gli attributi del dataset **ad eccezione di 'sex' ed 'emotion'**, in quanto queste variabili saranno successivamente predette, quindi utilizzarle in fase di replacing dei missing potrebbe introdurre un bias all'interno del dataset.

Dato che le features: ['vocal_channel', 'sex', 'emotional_intensity', 'repetition', 'statement'] sono **categoriche binarie**, possiamo effettuare la sostituzione dei valori sfruttando un **mapping 0 - 1**. Di fatto, nel caso di variabili binarie, questa strategia non è dissimile dall'effettuare un ONE-HOT-ENCODING, con l'unica differenza che in quest'ultimo si generano due colonne dummy che sono una il complemento a 1 dell'altra. Questo fa sì che entrambe abbiano lo stesso contenuto informativo, e che quindi una delle due possa essere scartata; il risultato è perciò identico a fare un semplice mapping 0 - 1.

```
In [59]: dict_map = {'speech': 0,
                 'song': 1,
                 'F': 0,
                 'M': 1,
                 'normal': 0,
                 'strong': 1,
                 '1st': 0,
                 '2nd': 1,
                 'Dogs are sitting by the door': 0,
                 'Kids are talking by the door': 1}
```

```
In [60]: for col in ['vocal_channel', 'sex', 'emotional_intensity', 'repetition', 'statement']:
    df[col] = df[col].apply(lambda x: dict_map[x])

df[['vocal_channel', 'sex', 'emotional_intensity', 'repetition', 'statement']]
```

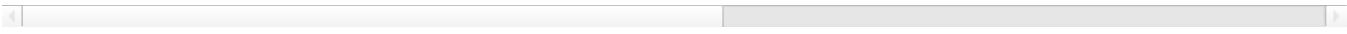
```
Out[60]:   vocal_channel  sex  emotional_intensity  repetition  statement
0             0     0                  0           1         0
1             0     0                  0           0         0
2             0     0                  1           1         0
3             0     0                  0           0         1
4             1     0                  1           1         0
...
2447          0     1                  1           0         1
2448          0     1                  0           0         0
2449          1     1                  1           1         0
2450          0     1                  0           0         1
2451          0     1                  0           1         0
```

2452 rows × 5 columns

```
In [61]: df.head()
```

	modality	vocal_channel	emotion	emotional_intensity	statement	repetition	actor	sex	channels	sample_width	...	stft_m
0	audio-only	0	fearful	0	0	1	2.00000	0	1	2	...	0.00000
1	audio-only	0	angry	0	0	0	16.00000	0	1	2	...	0.00000
2	audio-only	0	happy	1	0	1	16.00000	0	1	2	...	0.00000
3	audio-only	0	surprised	0	1	0	14.00000	0	1	2	...	0.00000
4	audio-only	1	happy	1	0	1	2.00000	0	1	2	...	0.00000

5 rows × 38 columns



Come detto in precedenza, isoliamo le colonne ['emotion', 'sex'], che non verranno utilizzate come predittori al fine di non creare un bias nell'operazione di resampling; escludiamo inoltre gli attributi che verranno poi eliminati dal dataset per le ragioni discusse nel data understanding.

```
In [62]: list_col_reg = [col for col in df.columns.to_list() if col not in ['emotion', 'intensity', 'sex', 'actor', 'sample_width', 'frame_rate', 'frame_width', 'modality', 'channels', 'stft_max', 'mean']]  
df[list_col_reg].head()
```

	vocal_channel	emotional_intensity	statement	repetition	length_ms	frame_count	zero_crossings_sum	mfcc_mean	mfcc_std
0	0	0	0	1	3737	179379.00000	16995	-33.48595	134.65486
1	0	0	0	0	3904	187387.00000	13906	-29.50211	130.48563
2	0	1	0	1	4671	224224.00000	18723	-30.53246	126.57711
3	0	0	1	0	3637	174575.00000	11617	-36.05956	159.72516
4	1	1	0	1	4404	211411.00000	15137	-31.40600	122.12582

5 rows × 27 columns



A questo punto andiamo a creare gli array **X** e **y**, che corrisponderanno alle sole osservazioni che non presentano missing values.

```
In [63]: # Valori not NaN su cui addestrare e valutare il regressore:  
mask_not_nan = df['intensity'].notna()  
  
# Valori NaN da rimpiazzare attraverso la previsione del regressore precedentemente addestrato e valutato:  
mask_nan = df['intensity'].isna()  
  
X = df.loc[mask_not_nan, list_col_reg].values  
y = df.loc[mask_not_nan, 'intensity'].values  
  
X[0:3, :], y[0:3]
```

```
Out[63]: (array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
       1.00000000e+00,  3.73700000e+03,  1.79379000e+05,
       1.69950000e+04, -3.34859470e+01,  1.34654860e+02,
      -7.55223450e+02,  1.71690920e+02,  5.79255074e+03,
       3.32805546e+03,  0.00000000e+00,  1.35419590e+04,
      -1.12076852e+00,  2.50940162e-01,  4.15250450e-01,
       3.35533140e-01,  0.00000000e+00, -1.21502497e+00,
       4.03514030e-01,  1.44816360e-02, -1.28631590e-01,
       1.38946530e-01,  9.40606124e+00,  2.73153374e-01],
      [ 0.00000000e+00,  1.00000000e+00,  0.00000000e+00,
       1.00000000e+00,  4.67100000e+03,  2.24224000e+05,
       1.87230000e+04, -3.05324630e+01,  1.26577110e+02,
      -7.26060360e+02,  1.65456530e+02,  4.83074304e+03,
       3.33213130e+03,  0.00000000e+00,  1.20077512e+04,
      -1.13015274e+00,  4.36699155e-01,  3.79757520e-01,
       3.52269680e-01,  0.00000000e+00, -1.24294652e+00,
       4.70350013e-01,  2.43171750e-02, -1.37481690e-01,
       1.66351320e-01,  4.88124056e+00,  3.02658999e-01],
      [ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00,
       0.00000000e+00,  3.63700000e+03,  1.74575000e+05,
       1.16170000e+04, -3.60595550e+01,  1.59725160e+02,
      -8.42946350e+02,  1.90036090e+02,  5.37644648e+03,
       4.05366307e+03,  0.00000000e+00,  1.20482239e+04,
      -1.49776486e+00,  9.88022841e-02,  4.07276720e-01,
       3.60552070e-01,  0.00000000e+00, -1.44531811e+00,
       2.74755933e-01,  3.56098640e-03, -2.73742680e-02,
       2.40783700e-02,  1.30402594e+01, -8.10143267e-02]]),
 array([-36.79343187, -32.29073734, -49.01983891]))
```

Dopo aver creato gli array X e y, passiamo a suddividere il dataset in **training set** e **test set** con proporzione 70% - 30%.

```
In [64]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

L'idea è quella di addestrare un **Decision Tree Regressor** sul training set, andando ad ottimizzare i parametri con una **Random Search Cross Validation**. Impostiamo un numero di fold della CV pari a 10 e reiteriamo la CV per 10 volte.

I parametri che abbiamo deciso di ottimizzare sono:

- massima profondità dell'albero;
- minimo numero di samples che devono essere presenti all'intero di un nodo per giustificare uno split;
- minimo numero di samples che possono stare all'interno di una foglia;
- parametro di regolarizzazione alpha che introduce una penalizzazione sul numero di foglie generate a seguito dei vari split, e dunque permette di regolare il trade-off tra bias (albero troppo poco dettagliato) e variance (albero in completo overfitting e quindi troppo ricco di foglie).

La metrifica utilizzata in fase di validazione del modello è il Mean Squared Error (MSE).

```
In [65]: from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeRegressor

# definisci la hyperparameter grid:
dict_params = {'max_depth': list(np.arange(1, 10, 3)),
               'min_samples_split': np.arange(5, 21, 3),
               'min_samples_leaf': np.arange(5, 21, 3),
               'ccp_alpha': np.arange(0, 0.2, 0.02)}

# crea il DecisionTreeRegressor:
reg = DecisionTreeRegressor()

rand = RandomizedSearchCV(reg, dict_params, cv = 10, scoring = 'neg_mean_squared_error', refit = True, n_iter = rand.fit(X_train, y_train))
```

```
Out[65]: >     RandomizedSearchCV
> estimator: DecisionTreeRegressor
    >     DecisionTreeRegressor
```

Di seguito osserviamo in ordine:

- migliori parametri, ovvero quelli che hanno mediamente permesso di ottenere un MSE più basso durante le varie iterazioni della 10-fold CV;
- miglior MSE ottenuto durante la 10-fold CV;
- miglior Tree ottenuto durante la 10-fold CV.

```
In [66]: rand.best_params_
```

```
Out[66]: {'min_samples_split': 11,
           'min_samples_leaf': 17,
           'max_depth': 7,
           'ccp_alpha': 0.04}
```

```
In [67]: best_validation_metric = rand.best_score_
best_validation_metric
```

```
Out[67]: -0.554803985356809
```

```
In [68]: reg = rand.best_estimator_
reg
```

```
Out[68]: ▾ DecisionTreeRegressor
DecisionTreeRegressor(ccp_alpha=0.04, max_depth=7, min_samples_leaf=17,
                      min_samples_split=11)
```

A questo punto possiamo fare il miglior Tree sull'intero dataset di training e usare il modello addestrato per andare a prevedere le osservazioni di test e vedere quanto la previsione **y_pred** si discosta dal reale valore **y_test**.

```
In [69]: reg.fit(X_train, y_train) # fit su intero training
y_pred = reg.predict(X_test) # predizione sul test set
```

```
In [70]: y_test[0:10], y_pred[0:10]
```

```
Out[70]: (array([-51.92743685, -44.37569489, -20.01804365, -30.79117597,
                  -29.77833341, -32.83096674, -21.15136076, -20.40644107,
                  -36.1405807 , -38.07453254]),
array([-51.36697177, -44.07733557, -20.36167671, -31.38662567,
      -29.44401329, -33.475995 , -20.36167671, -20.36167671,
      -36.40105338, -37.92074047]))
```

Di seguito riportiamo alcune metriche che per i task di regressione aiutano a capire se il modello è accurato:

- Mean Absolute Error;
- Mean Squared Error --> loss function scelta per il modello;
- BIAS --> permette di capire se il modello sta facendo overshooting oppure il contrario;
- R2 --> varianza spiegata del modello.

```
In [71]: # Mean Absolute Error
MAE = np.mean(np.abs(y_pred - y_test))
MSE = np.mean((y_pred - y_test)**2)
BIAS = np.mean((y_pred - y_test))
R2 = 1 - np.var(y_pred - y_test)/np.var(y_test)

print(f'Metriche di errore ottenute in fase di test:\nMAE: {MAE}\nMSE: {MSE}\nBIAS: {BIAS}\nR^2: {R2}')
```

Metriche di errore ottenute in fase di test:
MAE: 0.5693526351795175
MSE: 0.5259450230307586
BIAS: -0.014702059445458344
R^2: 0.9927433899392226

Il modello ha dei risultati talmente buoni da risultare anomali. Con una varianza spiegata del 99% circa, abbiamo quasi la certezza che **sussista una relazione esatta, deterministica** tra la variabile **intensity** e almeno una delle variabili scelte in fase di training.

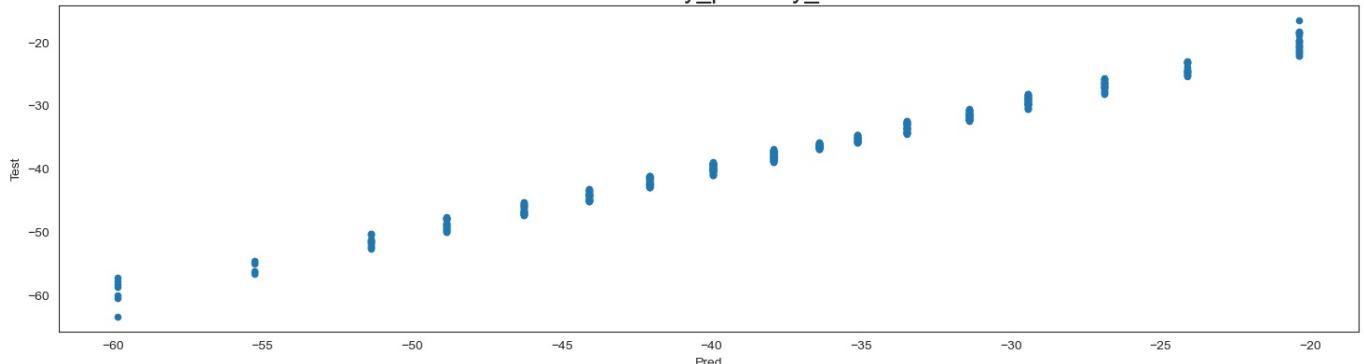
Questa affermazione è avvalorata dai seguenti due plot, che mettono in relazione **y_pred** e **y_test**:

```
In [72]: df_temp = pd.DataFrame({'Pred': y_pred, 'Test': y_test})

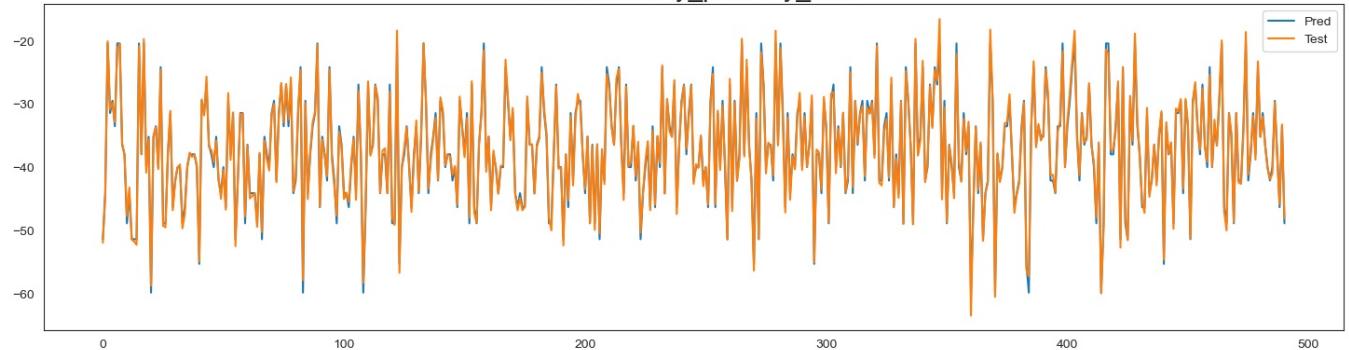
fig, ax = plt.subplots(2, figsize = (18, 10))

df_temp.plot(kind = 'scatter', x = 'Pred', y = 'Test', ax = ax[0])
df_temp.plot(ax = ax[1])
ax[0].set_title('Scatter tra y_pred e y_test', fontsize = 20)
ax[1].set_title('Andamento y_pred e y_test', fontsize = 20)
plt.legend()
plt.show()
```

Scatter tra y_pred e y_test



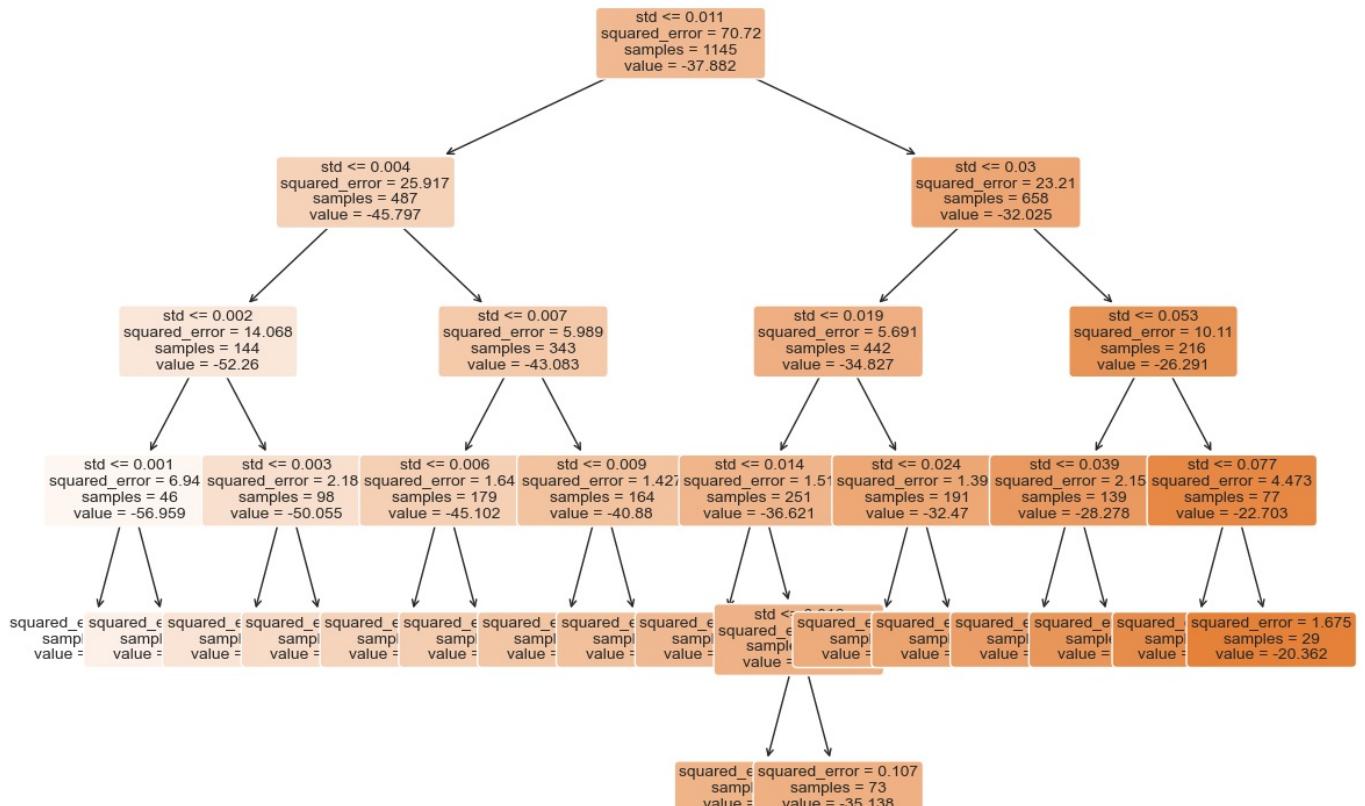
Andamento y_pred e y_test



Cerchiamo di capire il motivo di questo comportamento andando ad osservare su quali attributi l'albero esegue i primi split.

```
In [73]: from sklearn.tree import plot_tree

plt.figure(figsize=(14, 10))
plot_tree(reg,
          feature_names=list_col_reg,
          filled=True,
          rounded=True,
          fontsize=10,
          #max_depth=3
         )
plt.show()
```



Vediamo chiaramente che tutti gli split sono in corrispondenza dello stesso attributo **std** e quindi la relazione deve essere attribuita a tale variabile.

Dal seguente grafico osserviamo che la **relazione supposta esiste** ed è **non lineare**, più specificamente del tipo $f(x) = x^b$ con $b < 1$.

Dato che l'implementazione con l'algoritmo **CART** del Decision Tree esegue split axis parallel, notiamo che l'approssimazione della funzione da parte del Decision Tree è "**a scalini**" e quindi non adatta al problema.

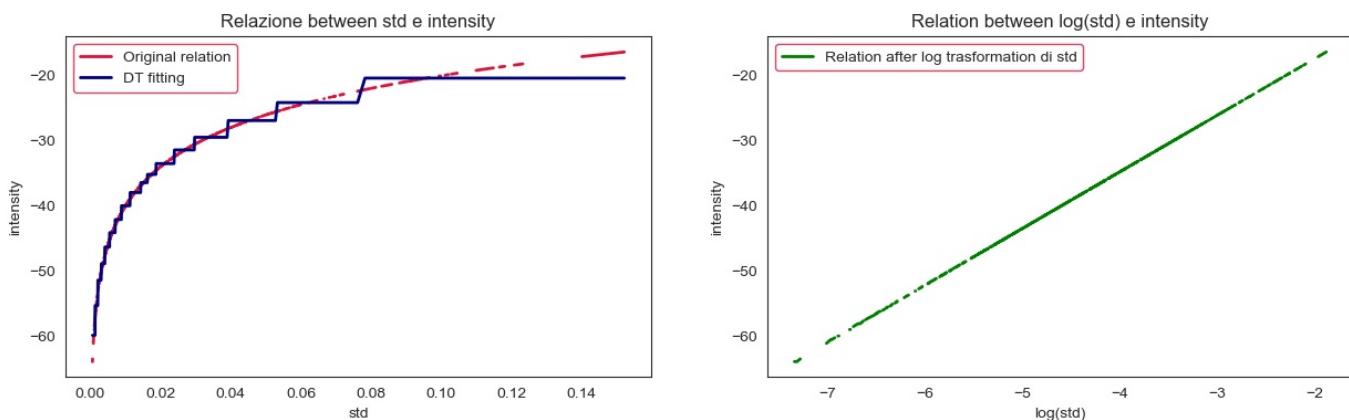
La strategia migliore sarebbe quella di catturare tale relazione attraverso una **trasformazione log della variabile std**, in modo da visualizzare una relazione lineare tra $\log(\text{std})$ e **intensity**.

```
In [76]: temp = df[['std', 'intensity']].copy(deep = True)
temp['DT_fitting'] = reg.predict(df.loc[:, list_col_reg].values)
temp['log_std'] = np.log(df['std'])
temp = temp.sort_values(by = ['std'])

fig, ax = plt.subplots(1, 2, figsize = (15, 4))
ax[0].plot(temp['std'], temp['intensity'], c = 'crimson', label = 'Original relation', lw = 2, linestyle = '-.')
ax[0].plot(temp['std'], temp['DT_fitting'], c = 'navy', label = 'DT fitting', lw = 2, linestyle = '-.')
ax[0].set_title('Relazione between std e intensity')
ax[0].set_xlabel('std')
ax[0].set_ylabel('intensity')
ax[0].legend(facecolor = 'white', edgecolor = 'crimson', fontsize = 10)

ax[1].plot(temp['log_std'], temp['intensity'], c = 'green', label = 'Relation after log trasformation di std',
           ax[1].set_title('Relation between log(std) e intensity')
ax[1].set_xlabel('log(std)')
ax[1].set_ylabel('intensity')
ax[1].legend(facecolor = 'white', edgecolor = 'crimson', fontsize = 10)

plt.show()
```



Come dimostrato nel plot esiste una relazione lineare esatta del tipo $\text{intensity} = m \cdot \log(\text{std}) + q$.

Dunque andiamo ad estrarre i parametri m e q di questa retta e proviamo a ricostruire la relazione originaria tra **std** e **intensity**.

```
In [77]: from sklearn.linear_model import LinearRegression

In [78]: X = np.log(df.loc[mask_not_nan, 'std'].values.reshape(-1, 1))
y = df.loc[mask_not_nan, 'intensity'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```
In [79]: linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
print(f'Intercetta: {linear_model.intercept_}\nCoefficiente Angolare: {linear_model.coef_[0]}')

Intercetta: 0.0613349472312521
Coefficiente Angolare: 8.703817022740086
```

Otteniamo che: $\text{intensity} = 8.7 \cdot \log(\text{std}) + 0.06$

Ovviamente calcolando le metriche di accuratezza, il bias e la varianza spiegata del modello, otteniamo i massimi valori ottenibili.

```
In [80]: y_pred = linear_model.predict(X_test)

# Mean Absolute Error
MAE = np.mean(np.abs(y_pred - y_test))
MSE = np.mean((y_pred - y_test)**2)
BIAS = np.mean((y_pred - y_test))
R2 = 1 - np.var(y_pred - y_test)/np.var(y_test)

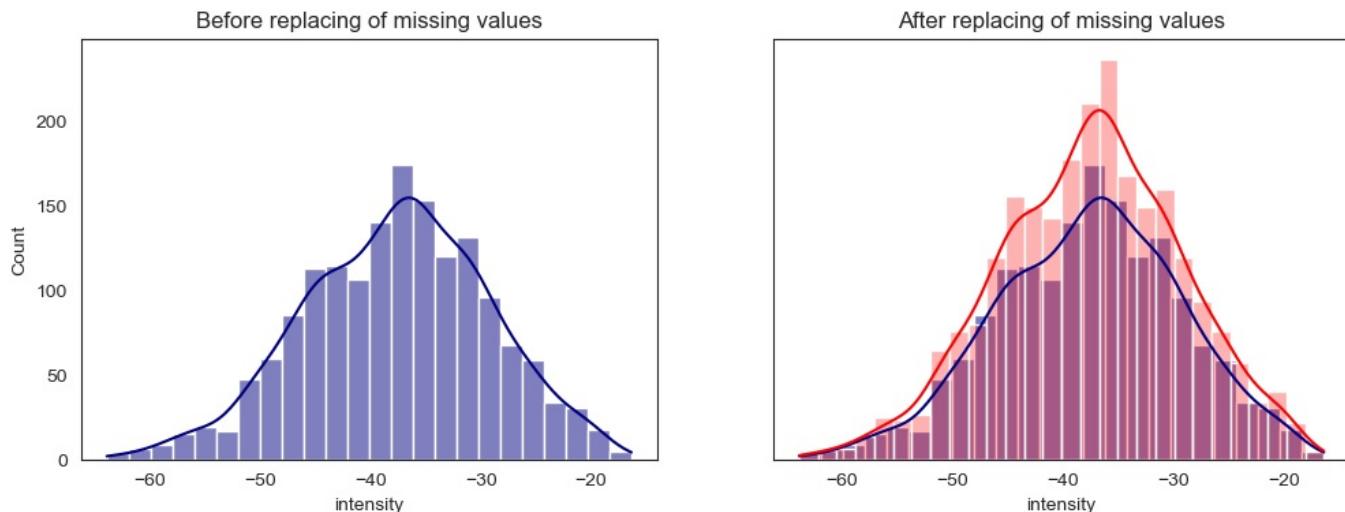
print(f'Metriche di errore ottenute in fase di test:\nMAE: {MAE}\nMSE: {MSE}\nBIAS: {BIAS}\nR^2: {R2}' )
```

```
Metriche di errore ottenute in fase di test:  
MAE: 0.011410262462256057  
MSE: 0.000377189979039145  
BIAS: -0.00039563557075306214  
R^2: 0.9999947958256035
```

A questo punto è evidente che non abbia molto senso proseguire l'analisi portandosi dietro entrambe le variabili **intensity** e **std**, dato che abbiamo visto che l'una è esattamente ottenibile a partire dall'altra. Quindi possiamo tranquillamente eliminare una delle due variabili e tenere l'altra: il contenuto informativo dei dati non subirà variazioni.

Tuttavia per **completezza** si preferisce completare la procedura di **replacing dei missing values**, al termine della quale **una delle due variabili verrà eliminata**.

```
In [81]: import copy  
  
old_intensity = df['intensity'].copy(deep = True)  
df.loc[mask_nan, 'intensity'] = linear_model.predict( np.log(df.loc[mask_nan, 'std']).values.reshape(-1, 1) )  
  
In [82]: fig, ax = plt.subplots(1, 2, figsize = (12, 4), sharey = True)  
  
sns.histplot(old_intensity, ax = ax[0], kde = True, color = 'navy')  
sns.histplot(old_intensity, ax = ax[1], kde = True, color = 'navy')  
sns.histplot(df['intensity'], ax = ax[1], alpha = 0.3, kde = True, color = 'red')  
ax[0].set_title('Before replacing of missing values')  
ax[1].set_title('After replacing of missing values')  
  
plt.show()
```



Come accennato in precedenza, procediamo all'eliminazione della variabile **intensity**:

```
In [83]: df.drop(columns = ['intensity'], inplace = True)
```

4. Pairwise correlations and eventual elimination of variables

Continuando con la fase di data preparation, possiamo notare che non sembra necessario un sampling, dato che il numero di record non è troppo elevato. D'altra parte, è invece utile una riduzione del numero degli attributi al fine di mitigare la 'curse of dimensionality'; utilizzeremo a questo scopo tecniche di **feature selection** e **feature projection**.

Iniziamo rimuovendo gli attributi che non apportano contributi informativi, viste le ragioni discusse nel data understanding. Ricapitolandole:

- tra le variabili categoriche: '**modality**' ha un solo valore in tutto il dataset:

```
In [84]: df['modality'].nunique()  
  
Out[84]: 1
```

- tra quelle continue: '**sample_width**', '**frame_rate**', '**stft_max**' hanno varianza pari a 0:

```
In [85]: df[['sample_width', 'frame_rate', 'stft_max']].var()  
  
Out[85]: sample_width    0.000000  
frame_rate     0.000000  
stft_max       0.000000  
dtype: float64
```

- 'actor' è solo un identificativo:

```
In [86]: df['actor'].unique()
```

```
Out[86]: array([ 2., 16., 14., nan, 12., 6., 22., 20., 8., 23., 7., 13., 3.,
       17., 21., 1., 15., 5., 19., 9., 11., 24., 18., 4., 10.])
```

- 'channels' e 'frame_width' hanno lo stesso valore in tutto il dataset, fatta eccezione per 6 record, gli stessi per entrambe le variabili:

```
In [87]: len(df[(df['channels'] == 2) & (df['frame_width'] == 4)])
```

```
Out[87]: 6
```

```
In [88]: len(df[(df['channels'] != 2) & (df['frame_width'] != 4)])
```

```
Out[88]: 2446
```

Procediamo quindi al drop:

```
In [89]: df = df.drop(['modality', 'sample_width', 'frame_rate', 'stft_max', 'actor', 'channels', 'frame_width'], axis=1)
```

- a differenza delle variabili eliminate in precedenza, 'mean' ha 2450 valori unici; tuttavia, la sua varianza è molto prossima allo 0:

```
In [90]: df['mean'].nunique()
```

```
Out[90]: 2450
```

```
In [91]: df['mean'].var()
```

```
Out[91]: 1.8212297990049588e-09
```

La eliminiamo quindi con un **Variance Threshold**; sceglioamo come soglia 1.8212297990049588e-06 perché non vogliamo escludere altre feature, come ad esempio 'stft_min':

```
In [92]: df.var(numeric_only=True)
```

```
Out[92]: vocal_channel          0.24183
emotional_intensity          0.24863
statement                      0.25010
repetition                     0.25010
sex                            0.25002
length_ms                      357988.64825
frame_count                     813199784.87234
zero_crossings_sum             13434567.65492
mfcc_mean                       19.90843
mfcc_std                        418.27157
mfcc_min                        9989.09344
mfcc_max                        676.10957
sc_mean                          765949.56097
sc_std                           336955.90887
sc_min                           258090.30540
sc_max                           1009936.51318
sc_kur                           0.32793
sc_skew                          0.12461
stft_mean                        0.00681
stft_std                         0.00057
stft_min                         0.00002
stft_kur                          0.04485
stft_skew                        0.10940
mean                            0.00000
std                             0.00044
min                            0.03078
max                            0.03824
kur                            43.75636
skew                           0.20696
dtype: float64
```

```
In [93]: from sklearn.feature_selection import VarianceThreshold
```

```
numerical_columns = df.select_dtypes(include='number')
selector = VarianceThreshold(threshold=1.8212297990049588e-06)
```

```
numerical_reduced = selector.fit_transform(numerical_columns)
```

```
numerical_reduced_df = pd.DataFrame(numerical_reduced, columns=numerical_columns.columns[selector.get_support()])
```

```
In [94]: numerical_reduced_df.var()
```

```
Out[94]: vocal_channel          0.24183
emotional_intensity        0.24863
statement                  0.25010
repetition                 0.25010
sex                         0.25002
length_ms                  357988.64825
frame_count                813199784.87234
zero_crossings_sum         13434567.65492
mfcc_mean                  19.90843
mfcc_std                   418.27157
mfcc_min                   9989.09344
mfcc_max                   676.10957
sc_mean                    765949.56097
sc_std                      336955.90887
sc_min                      258090.30540
sc_max                      1009936.51318
sc_kur                      0.32793
sc_skew                     0.12461
stft_mean                  0.00681
stft_std                   0.00057
stft_min                   0.00002
stft_kur                   0.04485
stft_skew                  0.10940
std                          0.00044
min                          0.03078
max                          0.03824
kur                          43.75636
skew                         0.20696
dtype: float64
```

```
In [95]: non_numerical_columns = df.select_dtypes(exclude='number')

df = pd.concat([non_numerical_columns, numeric_reduced_df], axis=1)
```

Il dataframe è quindi passato da **38 a 29 colonne**, nessuna di questi con valori mancanti:

```
In [96]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 29 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   emotion          2452 non-null    object 
 1   vocal_channel    2452 non-null    float64
 2   emotional_intensity  2452 non-null    float64
 3   statement        2452 non-null    float64
 4   repetition       2452 non-null    float64
 5   sex              2452 non-null    float64
 6   length_ms        2452 non-null    float64
 7   frame_count      2452 non-null    float64
 8   zero_crossings_sum  2452 non-null    float64
 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_min            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(28), object(1)
memory usage: 555.7+ KB
```

Esploriamo ora le **correlazioni** tra le variabili:

```
In [100...]: corr_matrix = df.corr(numeric_only=True)
corr_matrix.style.background_gradient(cmap='coolwarm')
```

Out[100...]	vocal_channel	emotional_intensity	statement	repetition	sex	length_ms	frame_count	zero_crossings_
vocal_channel	1.000000	-0.003409	-0.004147	0.000829	0.025720	0.735581	0.730971	0.150
emotional_intensity	-0.003409	1.000000	0.000000	0.000000	-0.000256	0.114027	0.112906	0.254
statement	-0.004147	0.000000	1.000000	-0.000000	0.000000	0.029536	0.031539	-0.162
repetition	0.000829	0.000000	-0.000000	1.000000	-0.000000	0.015751	0.014837	0.016
sex	0.025720	-0.000256	0.000000	-0.000000	1.000000	-0.071584	-0.071873	-0.389
length_ms	0.735581	0.114027	0.029536	0.015751	-0.071584	1.000000	0.992904	0.329
frame_count	0.730971	0.112906	0.031539	0.014837	-0.071873	0.992904	1.000000	0.326
zero_crossings_sum	0.150020	0.254869	-0.162901	0.016961	-0.389077	0.329517	0.326649	1.000
mfcc_mean	0.041158	0.305904	0.019976	0.011494	0.539941	0.011321	0.010193	0.130
mfcc_std	-0.273710	-0.416725	0.022503	-0.018511	0.108791	-0.302735	-0.301194	-0.559
mfcc_min	0.189327	0.403562	-0.010737	0.013579	-0.126288	0.205996	0.205063	0.500
mfcc_max	-0.272968	-0.213957	-0.002525	-0.002847	0.545416	-0.347253	-0.344543	-0.448
sc_mean	-0.490620	-0.118081	-0.083286	-0.010039	-0.042652	-0.546234	-0.539614	-0.072
sc_std	-0.139776	-0.256610	-0.075070	-0.013379	-0.100327	-0.182306	-0.178340	-0.373
sc_min	-0.044178	0.076052	-0.012920	0.008414	0.301492	-0.077001	-0.077952	0.190
sc_max	-0.119548	-0.090582	0.003062	-0.010509	-0.203118	-0.118781	-0.114388	-0.040
sc_kur	0.231057	0.233992	0.075559	0.005785	0.019193	0.292127	0.290117	0.186
sc_skew	0.537546	0.197154	0.076676	0.009429	0.007658	0.621148	0.616003	0.140
stft_mean	-0.461034	-0.081052	-0.001038	-0.012592	0.574827	-0.561395	-0.556985	-0.400
stft_std	0.375834	0.031236	-0.057050	0.007671	-0.578974	0.409380	0.407534	0.119
stft_min	-0.184459	-0.085916	-0.000847	0.005653	0.422499	-0.245511	-0.243661	-0.195
stft_kur	-0.098856	0.004877	0.079564	-0.004754	0.129566	-0.072309	-0.072402	0.072
stft_skew	0.489754	0.107097	0.011607	0.014438	-0.462716	0.602057	0.596986	0.449
std	0.132173	0.400001	-0.025725	0.018927	-0.083554	0.167134	0.166655	0.470
min	-0.030327	-0.416768	0.005634	-0.009472	0.052830	-0.070644	-0.071574	-0.420
max	0.037932	0.417563	0.001904	0.008832	-0.053374	0.072309	0.072823	0.411
kur	-0.486250	0.038873	0.168459	-0.028316	0.118022	-0.460723	-0.456365	-0.118
skew	0.036377	0.123895	0.032291	-0.005024	-0.122251	0.064473	0.067136	0.201

Come osservato in precedenza, dopo la rimozione degli errori da **'frame_count'** (ovvero i record con valori < 0), la variabile ha una forte correlazione positiva con **'length_ms'**; ciò risulta sensato anche dal punto di vista semantico. Possiamo quindi eliminare una delle due variabili – naturalmente, sceglieremo di mantenere la feature a cui non abbiamo apportato modifiche:

```
In [101...]: df = df.drop('frame_count', axis=1)
```

```
In [103...]: df.columns
```

```
Out[103...]: Index(['emotion', 'vocal_channel', 'emotional_intensity', 'statement', 'repetition', 'sex', 'length_ms', 'zero_crossings_sum', 'mfcc_mean', 'mfcc_std', 'mfcc_min', 'mfcc_max', 'sc_mean', 'sc_std', 'sc_min', 'sc_max', 'sc_kur', 'sc_skew', 'stft_mean', 'stft_std', 'stft_min', 'stft_kur', 'stft_skew', 'std', 'min', 'max', 'kur', 'skew'], dtype='object')
```

5. Dimensionality Reduction: Principal Component Analysis (PCA)

Proseguiamo adesso l'analisi con la **Principal Component Analysis**, una tecnica che consente di **ridurre la dimensionalità del problema**, andando a proiettare i dati nello spazio delle componenti principali PC_1, \dots, PC_n , ovvero delle **combinazioni lineari** degli attributi originari, generatrici dello spazio che permette di osservare i dati dalla **prospettiva di massima variabilità**. Questo perché le componenti principali sono le direzioni di massima variabilità dei dati.

Gli **attributi da usare nella PCA** sono i **continui**, opportunamente **standardizzati** per evitare che le differenze di scala inficino la bontà del risultato.

```
In [104...]: numeric_cols = ['length_ms', 'zero_crossings_sum', 'mfcc_mean', 'mfcc_std', 'mfcc_min', 'mfcc_max', 'sc_mean']
```

```

'sc_std',
'sc_min',
'sc_max',
'sc_kur',
'sc_skew',
'stft_mean',
'stft_std',
'stft_min',
'stft_kur',
'stft_skew',
'std',
'min',
'max',
'kur',
'skew']

```

Dalla teoria sappiamo che le direzioni di massima variabilità dei dati sono gli **autovettori della matrice di covarianza**, quindi un buon punto di partenza per comprendere se l'approccio può risultare fruttuoso è quello di calcolare la **matrice di correlazione** (che altro non è che la **matrice di covarianza scalata** rispetto alle deviazioni standard dei predittori coinvolti).

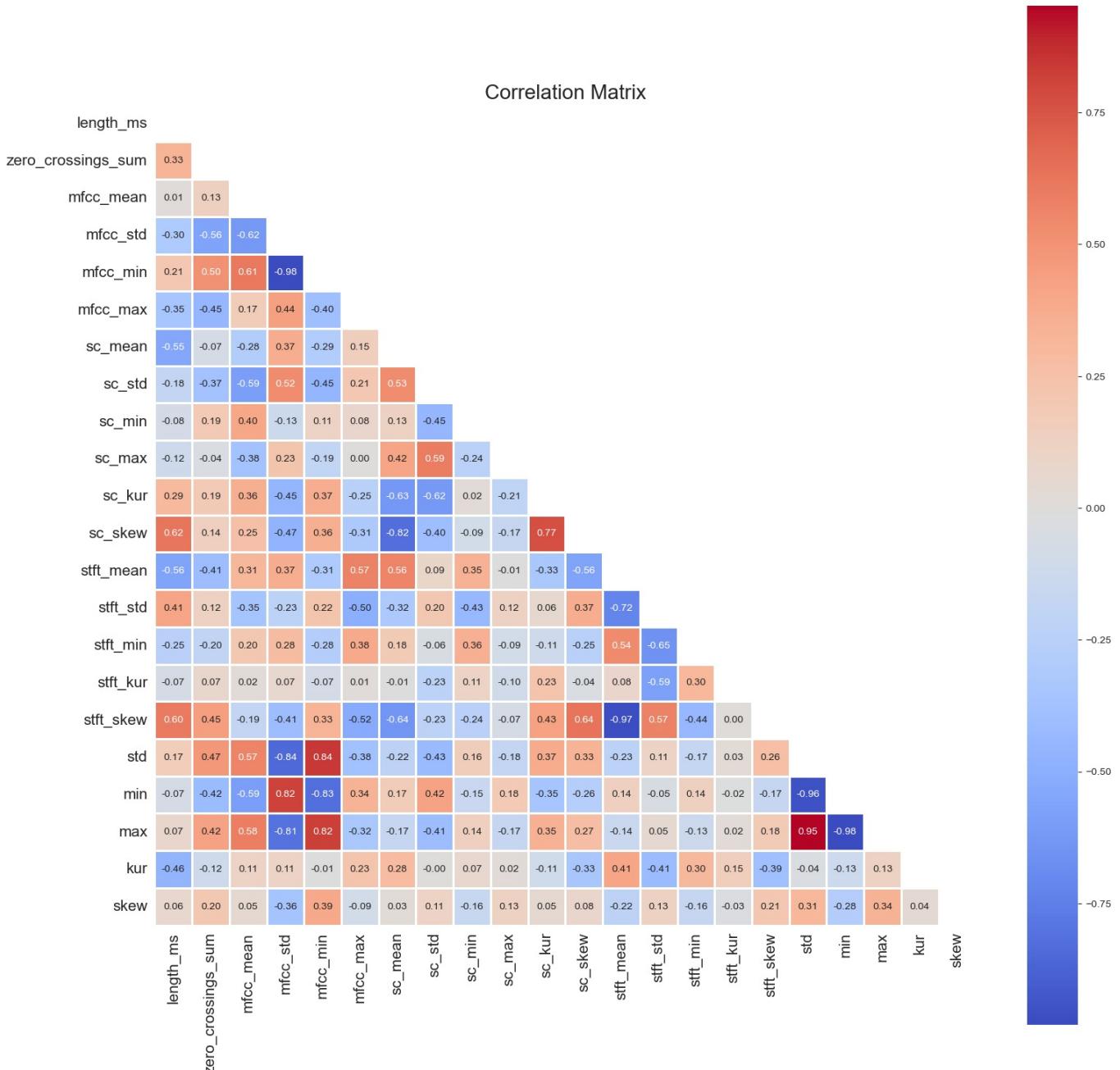
In [108]:

```

plt.figure(figsize=(18, 18))
corr_matrix = df[numeric_cols].corr(method = 'pearson', numeric_only=True)
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, mask=mask, cmap="coolwarm", annot=True, fmt=".2f", square = True, linewidths = 0.8)
plt.title('Correlation Matrix', fontsize = 20)
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)

plt.show()

```



Notiamo diverse correlazioni significative che risultano un buon presupposto per capire di più sulla struttura dei dati.

```
In [109]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Nella cella sottostante prima **standardizziamo** i fattori considerati con uno **standard scaler**, che semplicemente esegue questa operazione:

$$Z = \frac{X - \mu}{\sigma}, \text{ per ogni predittore } X, \text{ con } (\mu, \sigma) \text{ rispettivamente media e deviazione standard del vettore } X$$

Dopodiché proiettiamo ogni osservazione, nello spazio delle componenti principali:

```
In [110]: X = df[numeric_cols].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
In [111]: n_comp = 6

pca = PCA(n_components = n_comp)
pca.fit(X_scaled)

# Direzioni principali / eigenvectors della matrice di covarianza
prin_dir = pca.components_
prin_dir.shape
```

```
Out[111]: (6, 22)
```

Notiamo che abbiamo ottenuto una matrice con un numero di righe pari a 6 (valore da noi scelto e che giustificheremo in seguito), che sono le direzioni di variabilità principali e un numero di colonne che corrisponde ai fattori originari coinvolti.

Quindi questa matrice ci consente di leggere la relazione tra:

- **Fattori originari**
- **Componenti principali**

in termini di:

- **Segno** --> se un fattore originario cresce nella stessa direzione di crescita della componente principale o meno
- **Magnitudo** --> la grandezza della singola entry del DataFrame sottostante mi consente di capire quanto ciascun fattore originario "entra", ovvero è allineato a ciascuna componente principale

```
In [113]: df_principal_directions = pd.DataFrame(prin_dir.T,
                                              columns = ['EigenV' + str(j+1) for j in range(prin_dir.shape[0])],
                                              index = numeric_cols)

print('Principal directions explored:\n')
df_principal_directions
```

```
Principal directions explored:
```

Out[113...]

	EigenV1	EigenV2	EigenV3	EigenV4	EigenV5	EigenV6
length_ms	0.19029	-0.21519	0.18030	0.04122	-0.13937	-0.38954
zero_crossings_sum	0.21040	0.02007	-0.11875	0.44598	-0.27742	-0.13222
mfcc_mean	0.16502	0.34770	0.10739	-0.29785	-0.02479	-0.18165
mfcc_std	-0.33141	-0.12438	0.12550	0.05246	0.03933	0.07157
mfcc_min	0.30943	0.14586	-0.19008	-0.07664	-0.00769	-0.02527
mfcc_max	-0.20746	0.15073	0.12800	-0.31461	0.19061	-0.32657
sc_mean	-0.22772	0.12132	-0.37850	0.20325	-0.15599	-0.02400
sc_std	-0.21519	-0.20219	-0.32494	-0.08270	0.15333	-0.19501
sc_min	0.01141	0.27641	0.14201	0.15005	-0.47337	-0.22436
sc_max	-0.10371	-0.13615	-0.30882	0.12776	0.21301	-0.30724
sc_kur	0.23154	0.02090	0.29676	0.00857	0.30816	0.07954
sc_skew	0.25723	-0.13657	0.27984	-0.16248	0.19563	-0.13124
stft_mean	-0.22879	0.33444	0.00033	-0.17093	-0.05735	-0.05541
stft_std	0.14564	-0.36033	-0.17317	-0.24186	-0.13931	0.14996
stft_min	-0.14326	0.25343	0.19584	0.13316	0.06885	-0.26322
stft_kur	-0.01292	0.14233	0.21000	0.57841	0.34849	0.03963
stft_skew	0.24714	-0.28390	0.08646	0.20483	0.07113	-0.01392
std	0.29564	0.18874	-0.19044	-0.01207	0.00786	0.00873
min	-0.27570	-0.22873	0.20942	0.03984	-0.05469	-0.11358
max	0.27463	0.22649	-0.21861	-0.03659	0.08803	0.06890
kur	-0.09192	0.23934	-0.10974	0.05303	0.33820	0.38774
skew	0.11059	-0.00331	-0.28802	0.04213	0.36797	-0.46239

proiettiamo ogni osservazione X_j , nello spazio delle componenti principali:

In [114...]

```
# Osservazioni nel SR delle componenti principali
X_pca = pca.transform(X_scaled)
```

Nella cella sottostante troviamo giustificazione al numero di componenti principali sino ad ora scelto, ovvero 6.

Infatti ogni **direzione di variabilità (autovettore della matrice di covarianza)** si porta dietro un certo **autovalore λ** . Dividendo ciascun autovalore per la somma di tutti i possibili autovalori associati a tutte le direzioni di variabilità, otteniamo la **proporzione di varianza spiegata** da ciascuna componente principale:

$$\text{Explained - Variance - Ratio}_j = \frac{\lambda_j}{\lambda_1 + \dots + \lambda_d}$$

In [115...]

```
# percentage of explained variance (lambda_k) / (lambda_1 + ... + lambda_d)
prop_explained_variance = pca.explained_variance_ratio_
print('Proportion of explaine variance of each component:\n', prop_explained_variance, '\n\n')
```

Proportion of explaine variance of each component:
[0.33710647 0.19594283 0.11604689 0.06379365 0.05722639 0.04114389]

Considerando le prime 6 componenti principali totalizziamo una **proporzione di varianza spiegata cumulata** pari all'80%.

Questo significa che abbiamo notevolmente semplificato il problema in quanto **da oltre 20 fattori originari**, siamo passati a **sole 6 componenti principali, perdendo un contenuto informativo del solo 20%** circa.

$$\text{Explained - Variance - Ratio}_{\text{First-}k-\text{Components}} = \frac{\lambda_1 + \dots + \lambda_6}{\lambda_1 + \dots + \lambda_d} = 0.81$$

In [117...]

```
# cumulative percentage of explained variance (lambda_1 + lambda_k) / (lambda_1 + ... + lambda_d)
cum_prop_explained_variance = np.cumsum(prop_explained_variance)
print('Proportion of explaine variance of the first k-component:\n', cum_prop_explained_variance)
```

Proportion of explaine variance of the first k-component:
[0.33710647 0.5330493 0.64909619 0.7128983 0.77011622 0.81126011]

Di seguito visualizziamo lo **Scree Plot** che permette di percepire il gain di contenuto informativo che ciascuna componente principale si

porta dietro.

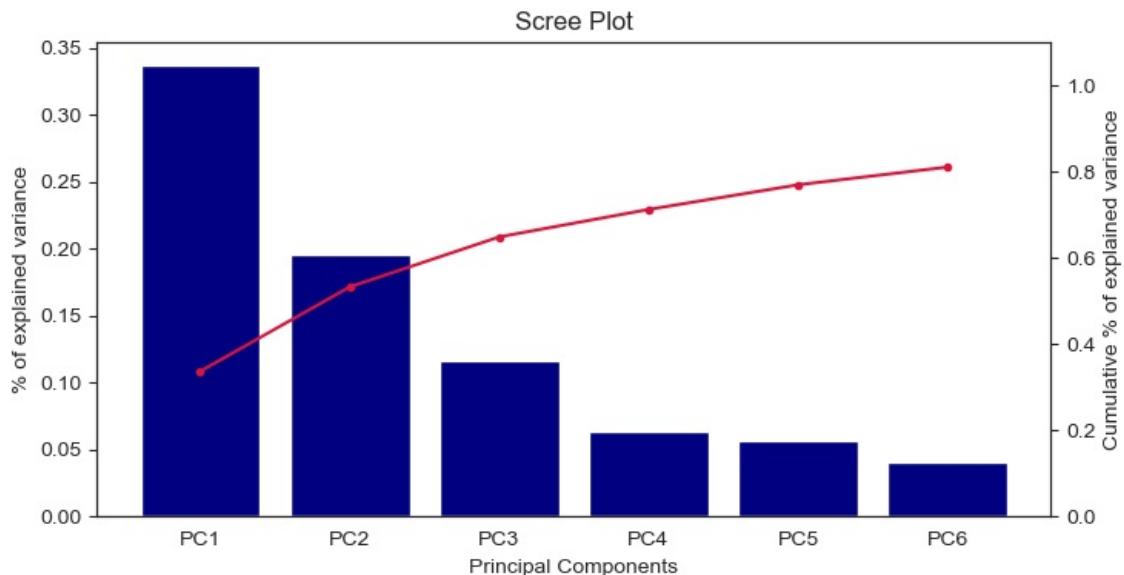
```
In [118]: fig, ax = plt.subplots(figsize = (8, 4))

axt = ax.twinx()

labels = ['PC' + str(j) for j in range(1, len(prop_explained_variance) + 1)]
ax.bar(labels, prop_explained_variance, color = 'navy')

axt.plot(labels, cum_prop_explained_variance, color = 'crimson', marker = '.')
ax.set_xlabel('Principal Components')
ax.set_ylabel('% of explained variance')
axt.set_ylabel('Cumulative % of explained variance')
ax.set_title('Scree Plot')

axt.set_ylim([0, 1.1])
axt.grid(False)
plt.show()
```



Cerchiamo adesso di dare un'interpretazione alle **Componenti Principali**, osservando la disposizione dei punti all'interno dello spazio che esse generano, con particolare focus sulla particolare **emotion** di ogni osservazione

```
In [121]: from mpl_toolkits.mplot3d import Axes3D

x = 0
y = 1
z = 2

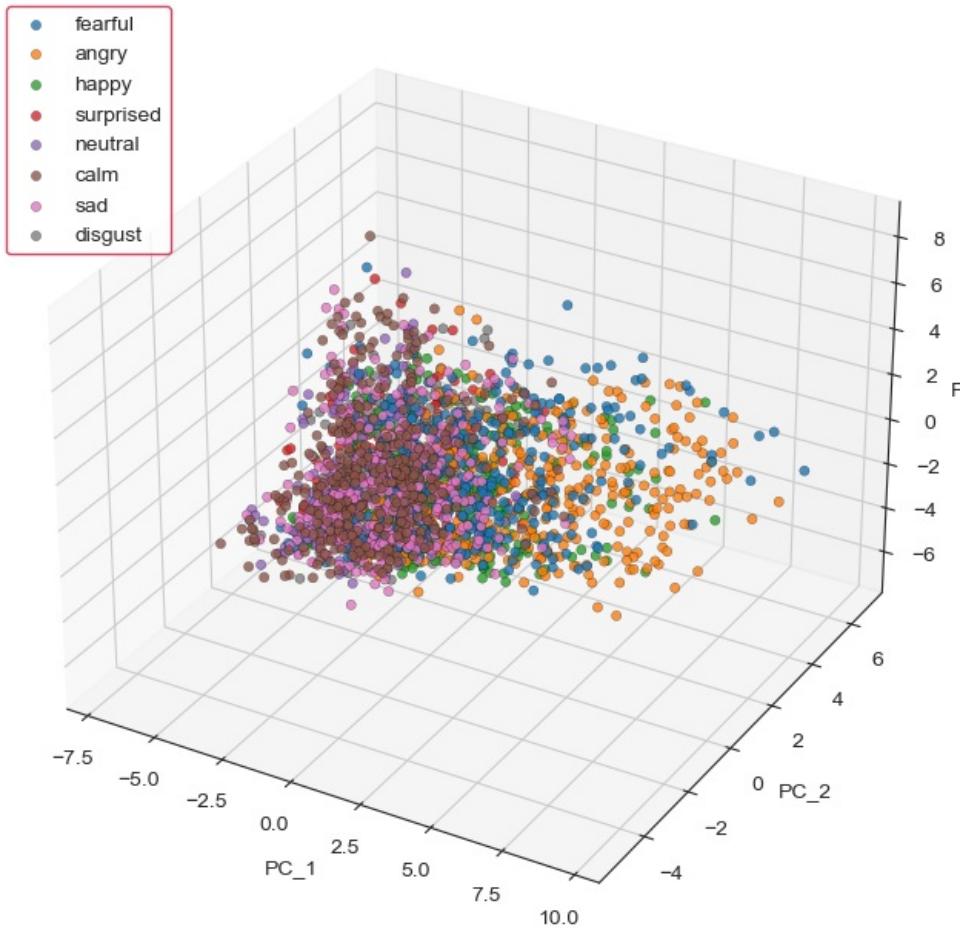
fig = plt.figure(figsize = (30, 8))
ax = fig.add_subplot(111, projection = '3d')

# X_pca e df hanno gli stessi indici
for emotion in df['emotion'].unique():
    mask = df['emotion'] == emotion
    scatter = ax.scatter(X_pca[mask, x],
                         X_pca[mask, y],
                         X_pca[mask, z],
                         label = emotion,
                         s = 20, alpha = 0.8, edgecolor = 'black', linewidth = 0.2, marker = 'o')

ax.set_xlabel(f'PC_{x + 1}')
ax.set_ylabel(f'PC_{y + 1}')
ax.set_zlabel(f'PC_{z + 1}')
ax.set_title(f'3D scatter of PC_{x+1} - PC_{y+1} - PC_{z+1}', fontsize = 14)
ax.legend(facecolor = 'white', edgecolor = 'crimson', fontsize = 10, loc = 'upper left')

plt.show()
```

3D scatter of PC_1 - PC_2 - PC_3



Dal precedente plot osserviamo che:

- Emozioni che corrispondono ad una **forte intensità vocale** come **angry**, **fearful** ed **happy** sono in corrispondenza di **valore alti e positivi di PC_1**
- L'emozione **fearful** in realtà ha una forte dispersione in quanto la troviamo anche per **valori centrali di PC_1**. Volendo azzardare un'interpretazione potremmo dire che durante gli stati di paura, le persone modulano la voce passando da un registro vocale molto basso, tipico degli **stati calmi** a un registro vocale molto alto, tipico degli **stati di rabbia / euforia**. La predominanza dell'uno o dell'altro nell'interpretazione dell'attore potrebbe aver generato questo effetto
- Emozioni caratterizzate da **bassa intensità vocale** come **calm**, **sad**, **neutral** si trovano per valori bassi della PC_1, come anche **surprised**
- Emozioni caratterizzate da **alta intensità vocale** come **happy** ed **angry** sembrano prediligere valori alti di PC_2 mentre **neutral** e **sad** valori diametralmente opposti della PC_2

Per validare queste prime intuizioni, facciamo uno step aggiuntivo e cerchiamo di capire *come i fattori originari si dispongono rispetto alle componenti principali*. Per farlo realizziamo il **biplot**, un grafico che permette di visualizzare i **predittori originari** come vettori nello spazio delle **componenti principali**.

In conclusione, come le componenti principali possono essere viste come combinazione lineare dei **predittori originari**, anche i predittori originari possono essere espressi come combinazione lineare delle **componenti principali**. Questo è molto utile a tracciare il link tra i due spazi vettoriali e **aggiungere contenuto semantico alle componenti principali**.

In [122]:

```
# %%BIPLOT

# PCs = X @ V, quindi X = PCs @ V_T. Considering only 2 PCs PC1 and PC2:
# PC1 @ V1 + PC2 @ V2 = X and for each factor Xj
# Xj = y1 * V1_j + y2 * V2_j
# So we can write the original factors as linear combinations of the principal components basis,
# having the principal_directions' coefficients as weights

def biplot(PC, principal_directions, list_variable_names, by_class_values = None, idx_1 = 0, idx_2 = 1):

    plt.figure(figsize=[12, 8])

    # idx_1, idx_2 are indexes of the PCs to be plotted (es. PC1 and PC_3)
    PC_1 = PC[:, idx_1] # first principal component
    PC_2 = PC[:, idx_2] # second principal component
```

```

PC_1 = PC_1 / (max(PC_1) - min(PC_1))
PC_2 = PC_2 / (max(PC_2) - min(PC_2))

for j in np.unique(by_class_values):
    mask = by_class_values == j
    plt.scatter(PC_1[mask], PC_2[mask], s = 20, label = str(j), edgecolor = 'black', linewidth = 0.2)

n = principal_directions.shape[0] # it's equal to the number of original factors

for j in range(n):
    plt.arrow(x = 0, y = 0, # coordinates of the arrow base
              dx = principal_directions[j, idx_1], # PC1 coordinate of arrow
              dy = principal_directions[j, idx_2], # PC2 coordinate of arrow
              color='black', head_width = 0.01, lw = 0.2)

    plt.text(principal_directions[j, idx_1] * 1.1, principal_directions[j, idx_2] * 1.1,
             list_variable_names[j], color='black', fontsize = 10)

plt.title("Biplot of factors in the principal components' space", fontsize = 16)
plt.xlabel('PC_1' + str(idx_1 + 1))
plt.ylabel('PC_2' + str(idx_2 + 1))
plt.legend(facecolor = 'white', edgecolor = 'crimson', fontsize = 10)

```

Semantica PC_1

Nei precedenti plot osserviamo che i fattori più allineati (con verso positivo) a PC_1 sono:

- zero_crossings_sum
- skew
- mfcc_min
- sc_skew
- sc_kur

Mentre con verso negativo:

- mfcc_std
- mfcc_max
- sc_mean

Inoltre confermiamo che solo **angry**, **happy** e **fearful** arrivano a toccare elevati valori di PC_1, mentre dal lato diametralmente opposto abbiamo le emozioni "deboli" come **sad**, **neutral** e **calm**.

Semantica PC_2

Nei precedenti plot osserviamo che i fattori più allineati (con verso positivo) a PC_2 sono:

- mfcc_mean
- sc_min
- stft_kur

Mentre con verso negativo:

- stft_std

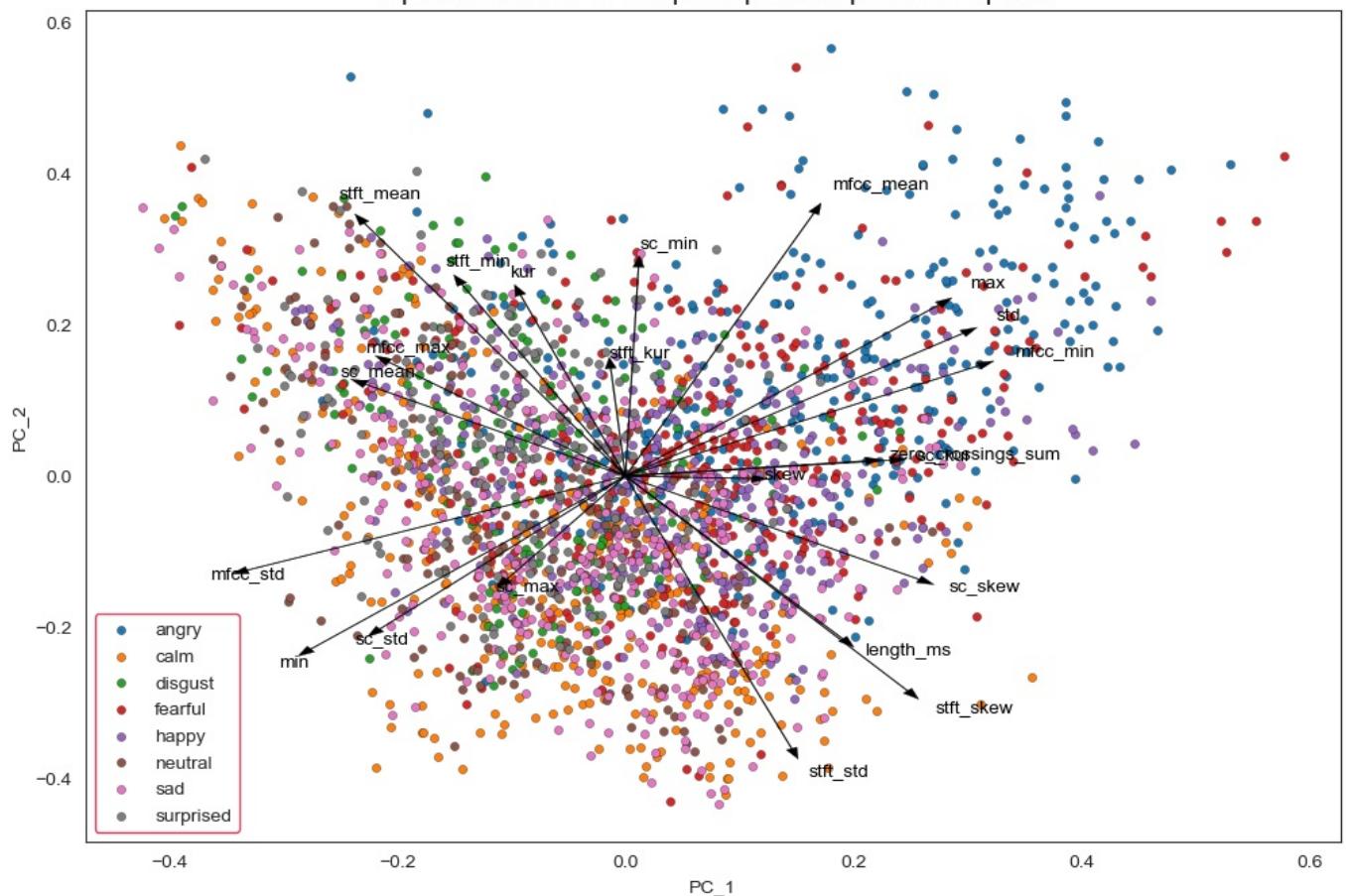
Ancora una volta solo **angry**, **happy** e **fearful** arrivano a toccare elevati valori di PC_2, mentre dal lato diametralmente opposto abbiamo le emozioni "deboli" come **sad**, **neutral** e **calm**, a cui si aggiunge anche **disgust**.

```

In [124]: biplot(PC = X_pca,
            # with the transposition the eigenvectors V are on columns
            principal_directions = prin_dir.T,
            list_variable_names = numeric_cols,
            by_class_values = df['emotion'].values,
            # insert index of principal direction
            idx_1 = 0, idx_2 = 1)

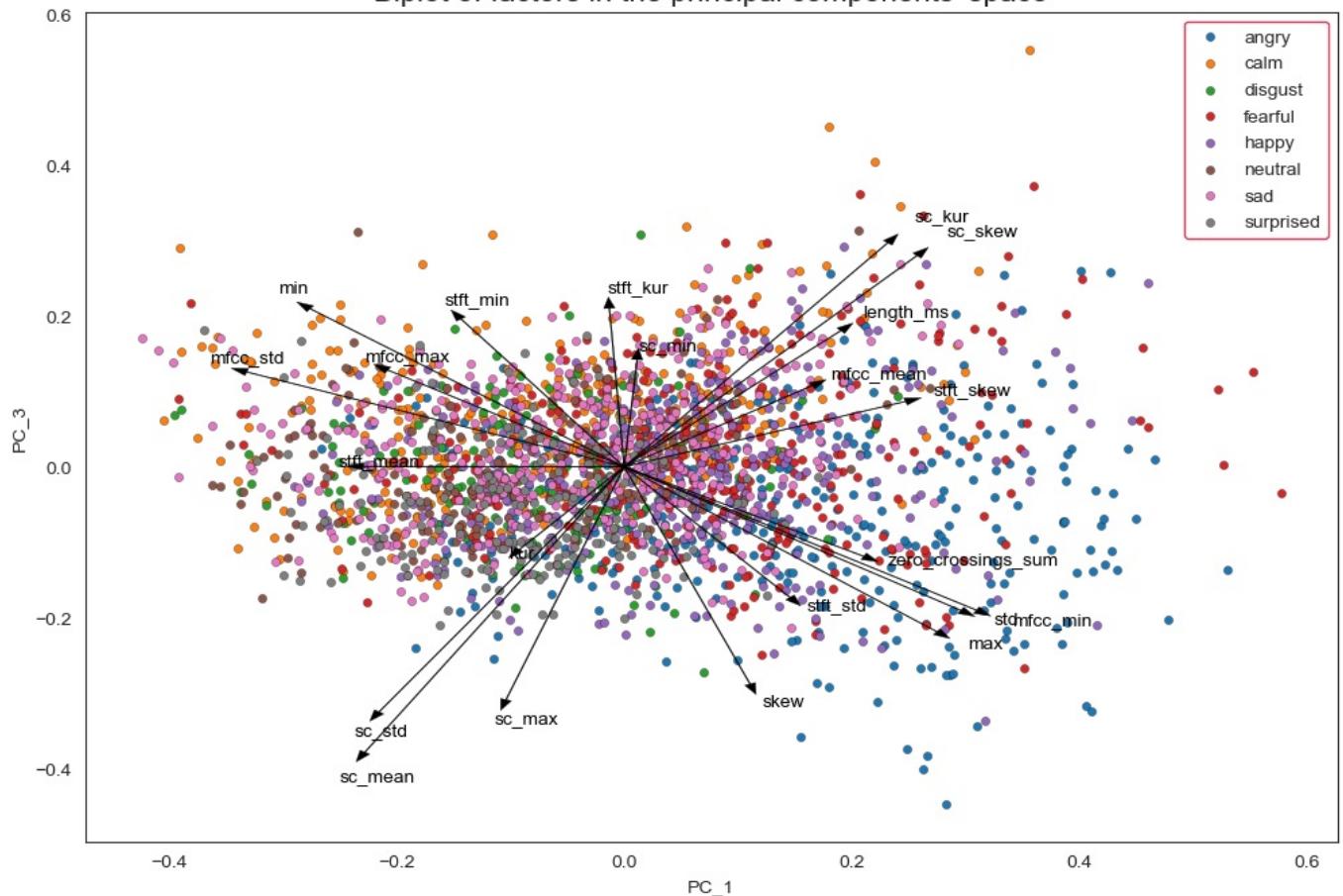
```

Biplot of factors in the principal components' space



```
In [125]: biplot(PC = X_pca,
           # with the transposition the eigenvectors V are on columns
           principal_directions = prin_dir.T,
           list_variable_names = numeric_cols,
           by_class_values = df['emotion'].values,
           idx_1 = 0, idx_2 = 2)
```

Biplot of factors in the principal components' space



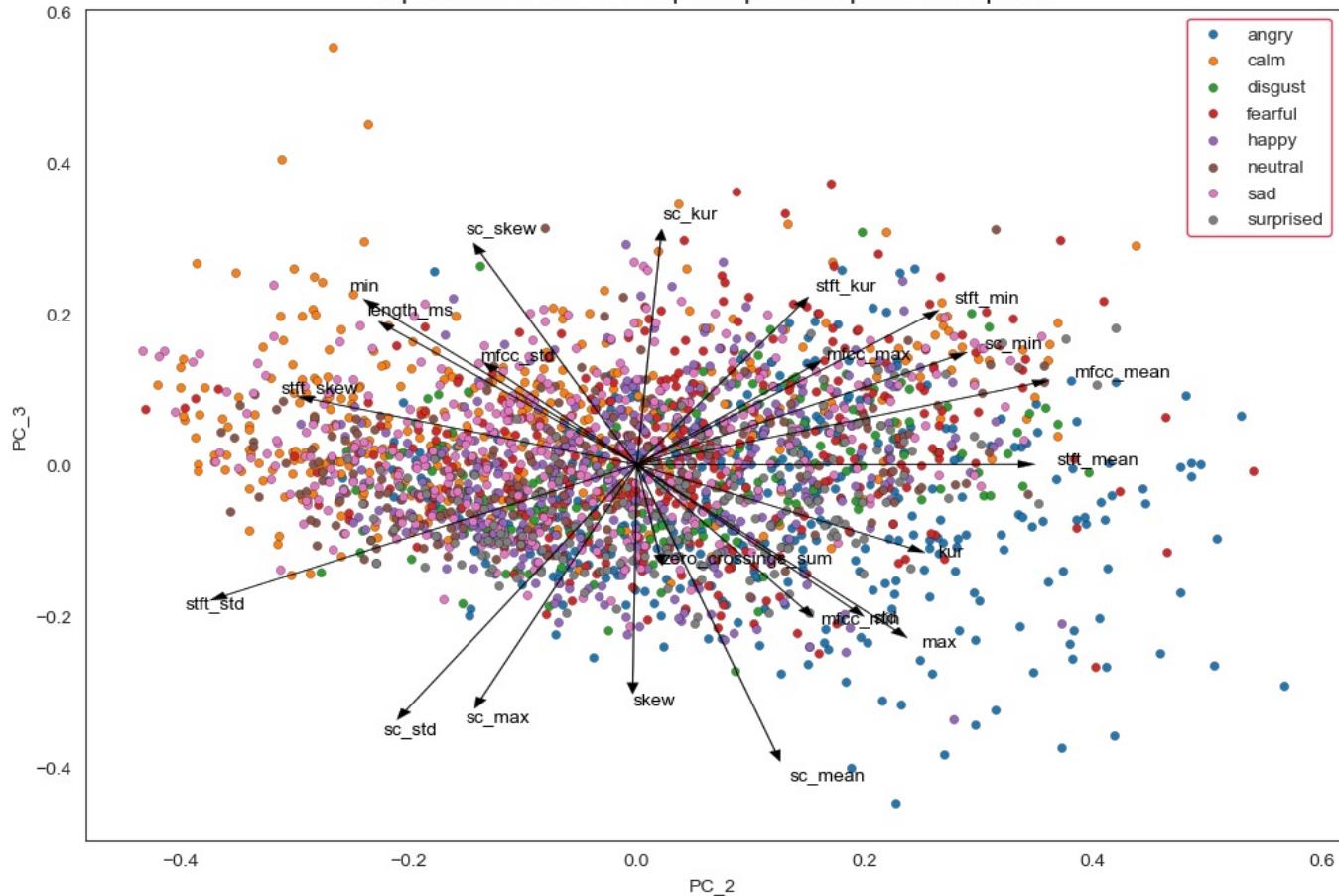
```
In [133]: biplot(PC = X_pca,
           # with the transposition the eigenvectors V are on columns
```

```

principal_directions = prin_dir.T,
list_variable_names = numeric_cols,
by_class_values = df['emotion'].values,
idx_1 = 1, idx_2 = 2)

```

Biplot of factors in the principal components' space



Possiamo usare quanto sopra per visualizzare un'altra dimensione, ovvero il **sex** (0 = 'F' e 1 = 'M').

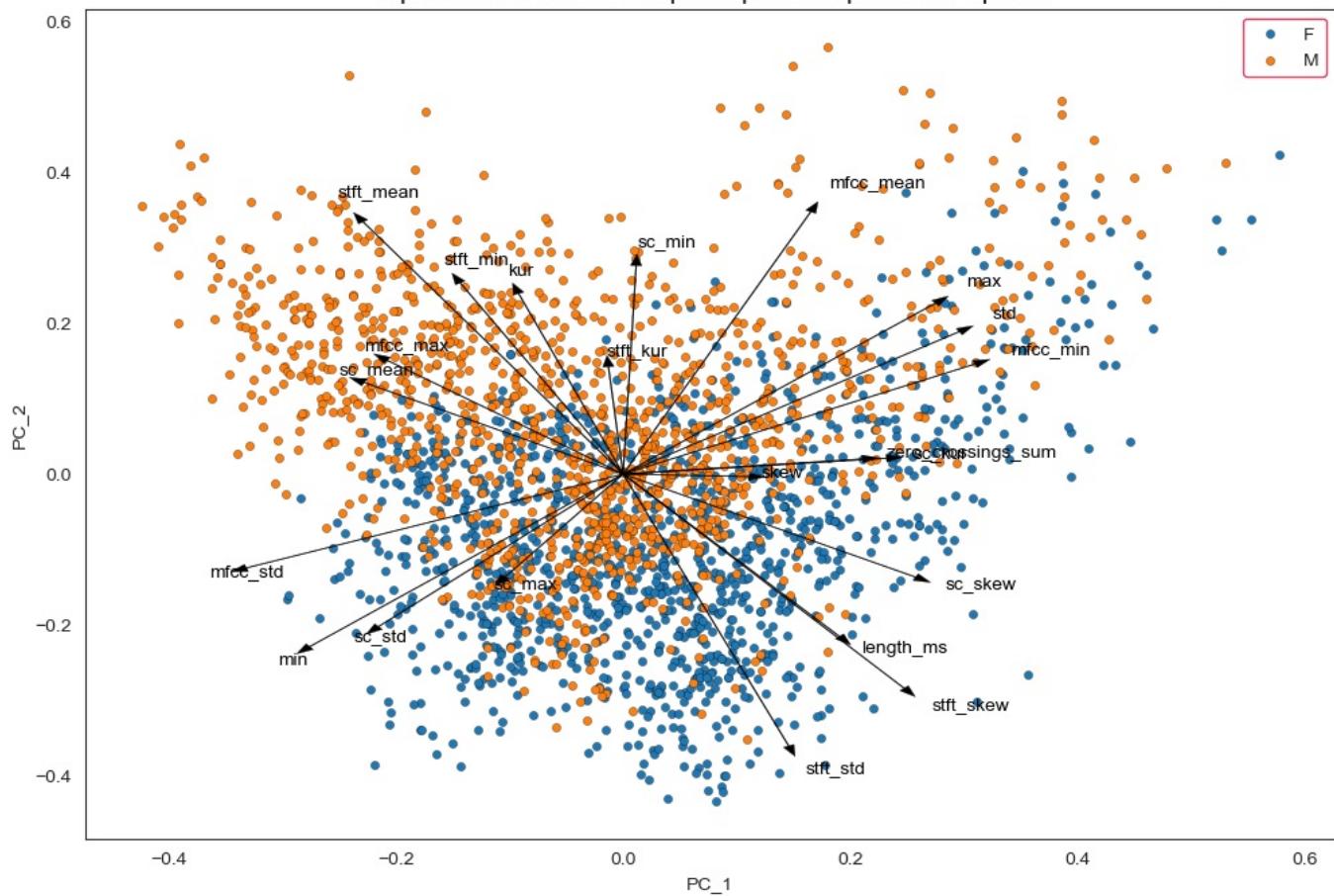
Sembra che PC_2 e PC_3 siano le dimensioni in cui possiamo meglio distinguere il **sex** e gli attributi che dominano questa separazione sono:

- stft_mean, mfcc_mean, mfcc_max, sc_min, stft_min --> valori elevati per uomini e bassi per le donne
- stft_skew, stft_std --> l'opposto

Questo lo avevamo già osservato in precedenza nei **violin plot**.

```
In [134]: biplot(PC = X_pca,
           # with the transposition the eigenvectors V are on columns
           principal_directions = prin_dir.T,
           list_variable_names = numeric_cols,
           by_class_values = df['sex'].apply(lambda x: {0: 'F', 1: 'M'}[x]).values,
           # insert index of principal direction
           idx_1 = 0, idx_2 = 1)
```

Biplot of factors in the principal components' space



6. Exporting Data Understanding & Preparation Output

```
In [132]: df.to_csv('A) Data Understanding & Preparation Output.csv', index = False)
```

```
In [ ]:
```

Processing math: 100%