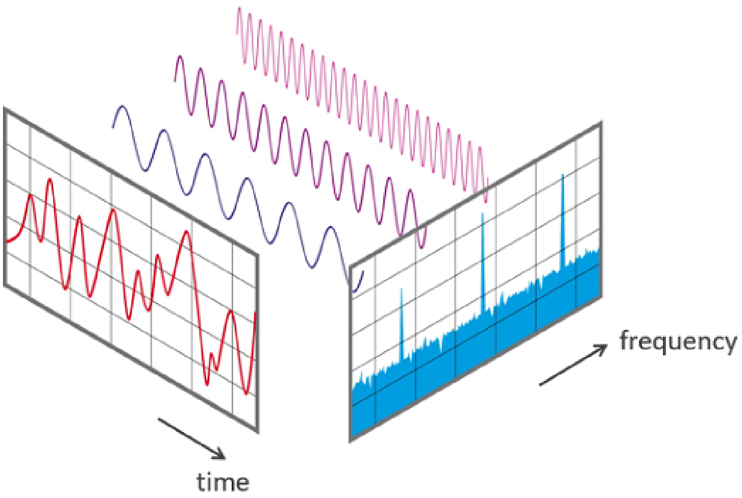
Audio classification

Классификация аудио с помощью python

Звук представлен в форме аудиосигнала с такими параметрами, как частота, полоса пропускания, децибел и т.д. Типичный аудиосигнал можно выразить в качестве функции амплитуды и времени.



Из спектрограмм я провела анализ аудиоданных и извлекла характеристики в виде среднего, дисперсии и др. значений с помощью библиотеки librosa.   Для классификации “живого” голоса (класс 1) и его отделению от синтетического/конвертированного/перезаписанного голоса (класс 2) я использовала ML алгоритм SVM (Support Vector Machines) / машины опорных векторов.

SVM работает путем сопоставления данных с многомерным пространством функций, чтобы точки данных можно было классифицировать, даже если данные не могут быть линейно разделены иным образом.

Для работы я использовала математическую функцию, используемой для преобразования (известна как функция ядра) - RBF (радиальную базисную функцию).
Результат: разработала модель классификации; точность классификатора составляет: Train set Accuracy: 0.979725
Test set Accuracy: 0.9713

```
%matplotlib inline
import librosa
import librosa.display
import IPython
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
```

```
audio_data = '../input/audioset/Training_Data/human/human_00004.wav'
y, sr = librosa.load(audio_data)
print(type(y), type(sr))
```

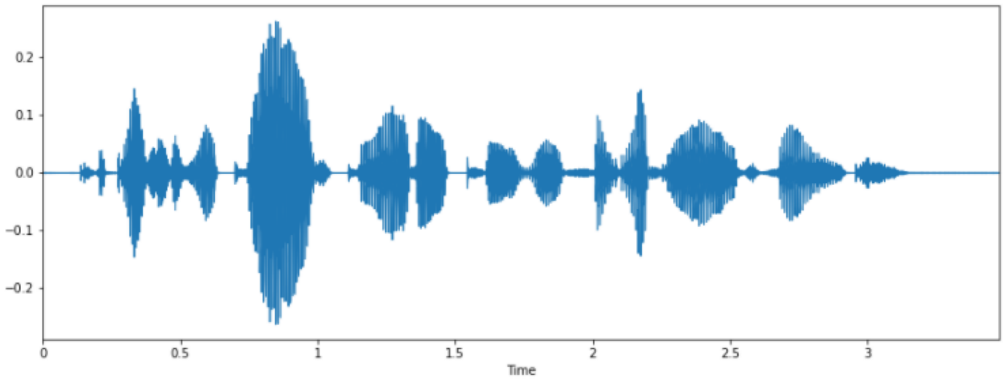
`<class 'numpy.ndarray'> <class 'int'>`

```
print(y.shape, sr)
```

`(76734,) 22050`

график управления амплитудой формы волны:

```
import IPython.display as ipd
plt.figure(figsize=(14, 5))
librosa.display.waveplot(y, sr=sr)
ipd.Audio(audio_data)
```

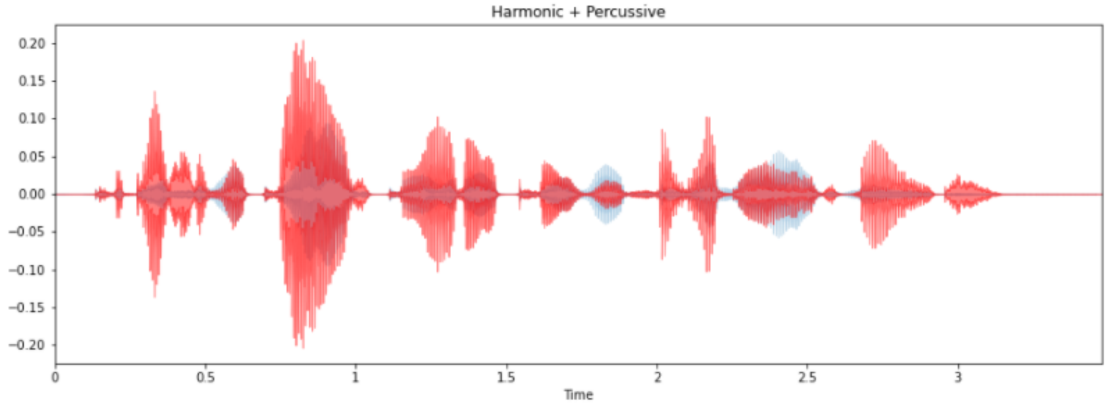



```
print(y, sr)
```

```
[-5.8037718e-04 -5.1912345e-04 -3.2173379e-04 ... -2.0331862e-04
 -5.4037344e-05  2.2379844e-04] 22050
```

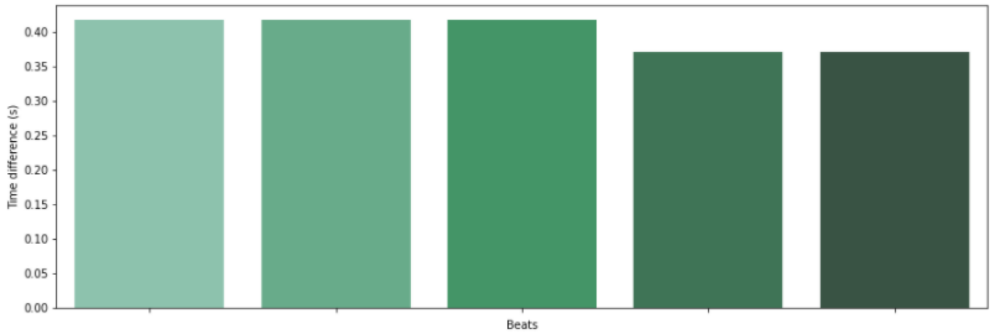
```
# Seperation of Harmonic and Percussive Signals
y_harmonic, y_percussive = librosa.effects.hpss(y)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(y_harmonic, sr=sr, alpha=0.25)
librosa.display.waveplot(y_percussive, sr=sr, color='r', alpha=0.5)
plt.title('Harmonic + Percussive')
```

```
Text(0.5, 1.0, 'Harmonic + Percussive')
```




```
#Beat Extraction
tempo, beat_frames = librosa.beat.beat_track(y=y_percussive,sr=sr)
print('Detected Tempo: '+str(tempo)+ ' beats/min')
beat_times = librosa.frames_to_time(beat_frames, sr=sr)
beat_time_diff=np.ediff1d(beat_times)
beat_nums = np.arange(1, np.size(beat_times))

fig, ax = plt.subplots()
fig.set_size_inches(15, 5)
ax.set_ylabel("Time difference (s)")
ax.set_xlabel("Beats")
g=sns.barplot(beat_nums, beat_time_diff, palette="BuGn_d",ax=ax)
g=g.set(xticklabels=[])
```

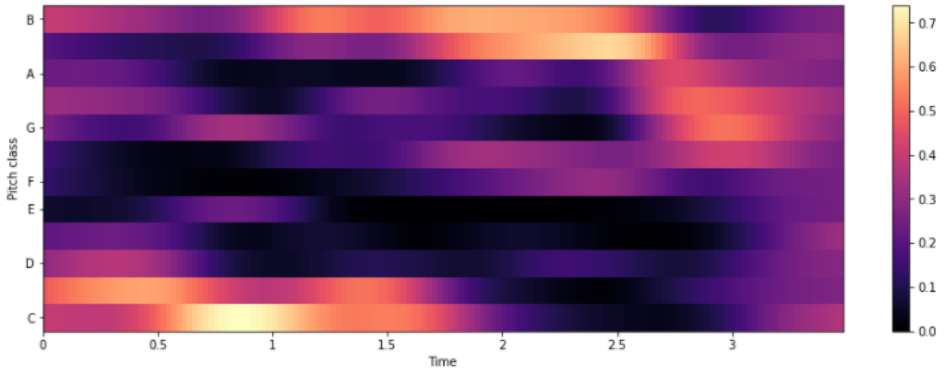
```
Detected Tempo: 143.5546875 beats/min
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misint
erpretation.
  FutureWarning
```



+ Code + Markdown

```
#Chroma Energy Normalized (CENS)
chroma=librosa.feature.chroma_cens(y=y_harmonic, sr=sr)
plt.figure(figsize=(15, 5))
librosa.display.specshow(chroma,y_axis='chroma', x_axis='time')
plt.colorbar()
```

: <matplotlib.colorbar.Colorbar at 0x7f5900163f10>

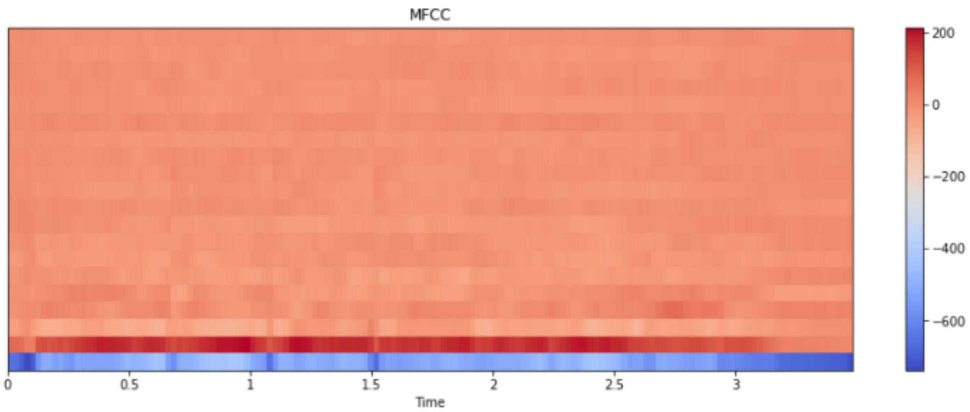


Мел-частотные кепстральные коэффициенты (MFCC)

Представляют собой набор признаков , которые описывают общую форму спектральной огибающей. Они моделируют характеристики человеческого голоса. MFCC - коэффициенты частотной капсулы, суммируют частотное распределение по размеру окна. Поэтому можно анализировать как частотные, так и временные характеристики звука.

```
# Calculate MFCCs
mfccs = librosa.feature.mfcc(y=y_harmonic, sr=sr, n_mfcc=20)
plt.figure(figsize=(15, 5))
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('MFCC')
```

Text(0.5, 1.0, 'MFCC')




```
mfccs
```

```
[10]: array([[-651.3048   , -658.4189   , -690.20197  , ..., -692.6444   ,
        -697.09656  , -724.9138   ],
       [  75.17549  ,   78.24612  ,   67.94123  , ...,   19.951931 ,
          17.157658 ,   12.479238 ],
       [ -40.15209  ,  -32.193527 ,  -13.942684 , ...,   -4.436577 ,
          -7.216289 ,   -1.5706065],
       ...,
       [   7.723238 ,    6.256642 ,    6.141851 , ...,    1.1357243,
          -2.9157412,   -3.1209354],
       [   6.408819 ,    4.8222485,    3.368474 , ...,    5.3537865,
          10.618406 ,   11.36906  ],
       [   2.8844805,    3.5840774,    6.452325 , ...,    2.6758838,
           6.5814176,    9.842529 ]], dtype=float32)
```

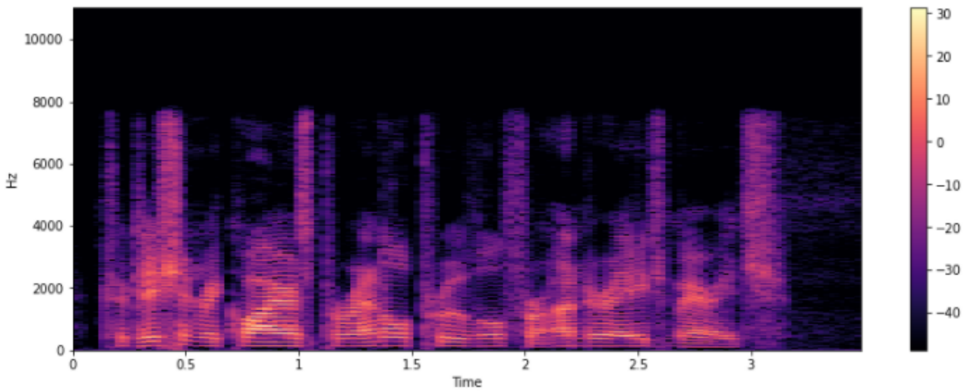
Спектрограмма

Спектрограмма — это визуальный способ представления уровня или “громкости” сигнала во времени на различных частотах, присутствующих в форме волны. Обычно изображается в виде тепловой карты.

`.stft()` преобразует данные в кратковременное преобразование Фурье. С помощью STFT можно определить амплитуду различных частот, воспроизводимых в данный момент времени аудиосигнала.

```
X = librosa.stft(y)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```

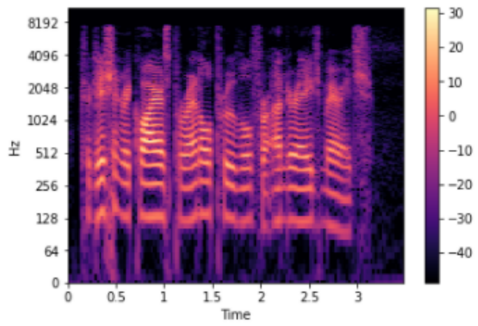
<matplotlib.colorbar.Colorbar at 0x7f59000fd710>



Поскольку все действие происходит в нижней части спектра, мы можем преобразовать ось частот в логарифмическую.

```
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
plt.colorbar()
```

<matplotlib.colorbar.Colorbar at 0x7f59003a6650>



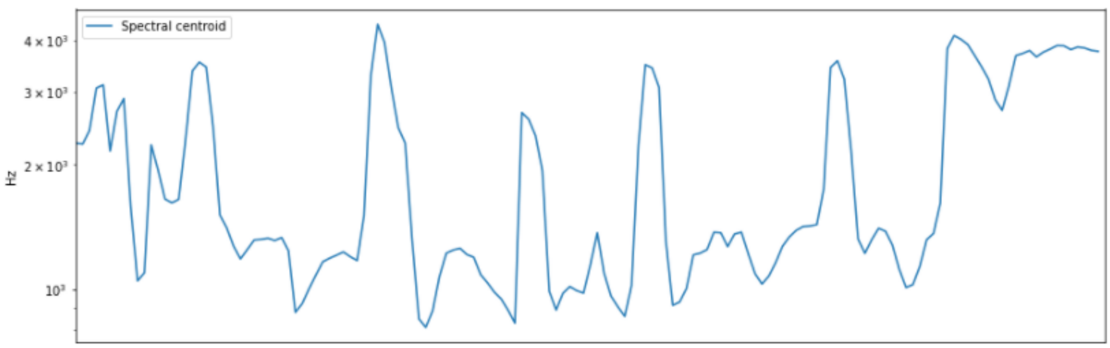
Спектральный центроид

Указывает, на какой частоте сосредоточена энергия спектра или, другими словами, указывает, где расположен “центр масс” для звука.

`librosa.feature.spectral_centroid` вычисляет спектральный центроид для каждого фрейма в сигнале:

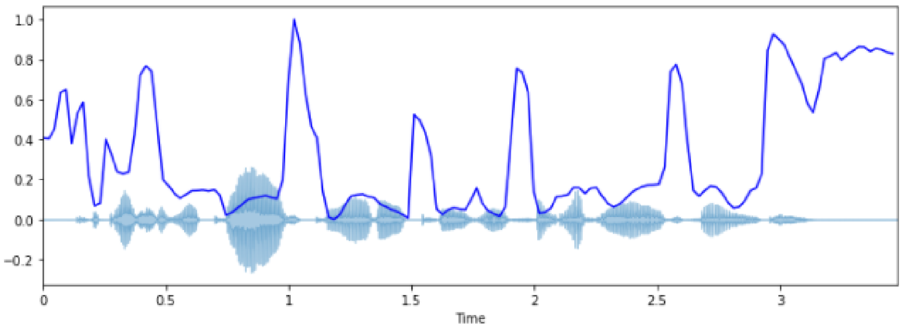
```
# Spectral Centroid
cent = librosa.feature.spectral_centroid(y=y, sr=sr)
plt.figure(figsize=(15,5))
plt.subplot(1, 1, 1)
plt.semilogy(cent.T, label='Spectral centroid')
plt.ylabel('Hz')
plt.xticks([])
plt.xlim([0, cent.shape[-1]])
plt.legend()
```

: <matplotlib.legend.Legend at 0x7f59005b2750>



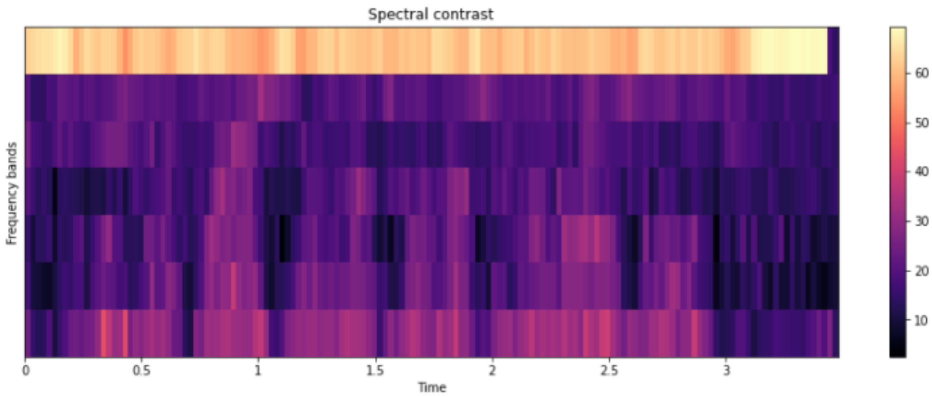

```
import sklearn
spectral_centroids = librosa.feature.spectral_centroid(y, sr=sr)[0]
spectral_centroids.shape

# Вычисление временной переменной для визуализации
plt.figure(figsize=(12, 4))
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)
# Нормализация спектрального центроида для визуализации
def normalize(y, axis=0):
    return sklearn.preprocessing.minmax_scale(y, axis=axis)
# Построение спектрального центроида вместе с формой волны
librosa.display.waveplot(y, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='b')
```

```
[<matplotlib.lines.Line2D at 0x7f59017b3a10>]
```




```
# Spectral Contrast
contrast=librosa.feature.spectral_contrast(y=y_harmonic,sr=sr)
plt.figure(figsize=(15,5))
librosa.display.specshow(contrast, x_axis='time')
plt.colorbar()
plt.ylabel('Frequency bands')
plt.title('Spectral contrast')
```

```
Text(0.5, 1.0, 'Spectral contrast')
```



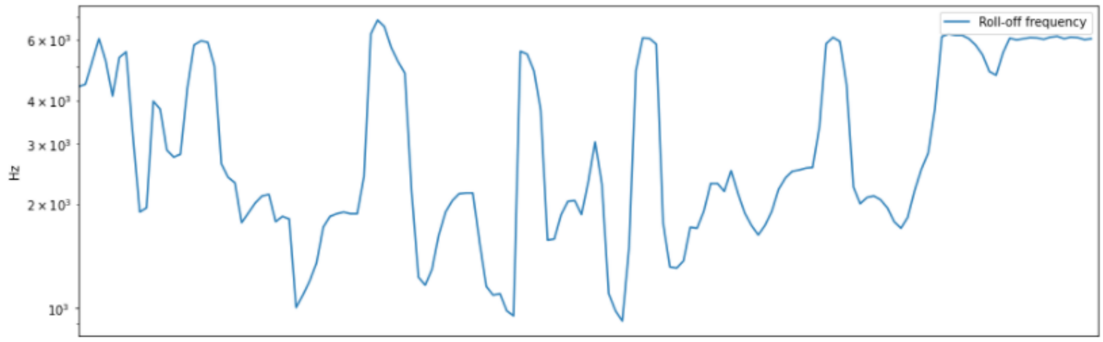
Спектральный спад

Это мера формы сигнала, представляющая собой частоту, в которой высокие частоты снижаются до 0. Чтобы получить ее, нужно рассчитать долю элементов в спектре мощности, где 85% ее мощности находится на более низких частотах.

`librosa.feature.spectral_rolloff` вычисляет частоту спада для каждого фрейма в сигнале:

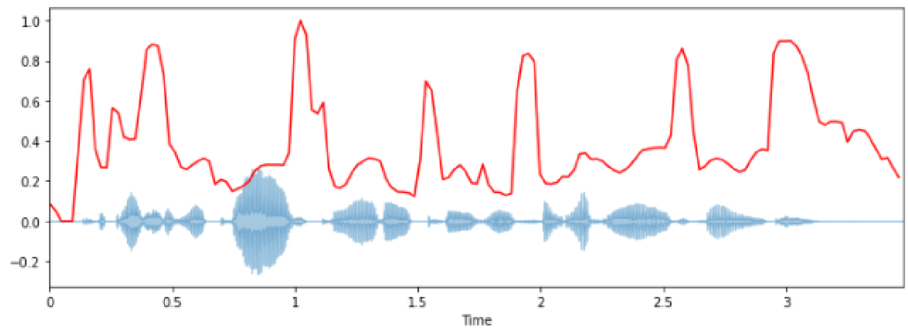
```
# Spectral Rolloff
rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
plt.figure(figsize=(15,5))
plt.semilogy(rolloff.T, label='Roll-off frequency')
plt.ylabel('Hz')
plt.xticks([])
plt.xlim([0, rolloff.shape[-1]])
plt.legend()
```

<matplotlib.legend.Legend at 0x7f5901824810>




```
spectral_rolloff = librosa.feature.spectral_rolloff(y+0.01, sr=sr)[0]
plt.figure(figsize=(12, 4))
librosa.display.waveplot(y, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_rolloff), color='r')
```

[<matplotlib.lines.Line2D at 0x7f58f2c37a90>]



Спектральная ширина

Спектральная ширина определяется как ширина полосы света на половине максимальной точки (или полная ширина на половине максимума [FWHM]) и представлена двумя вертикальными красными линиями и λSB на оси длин волн.

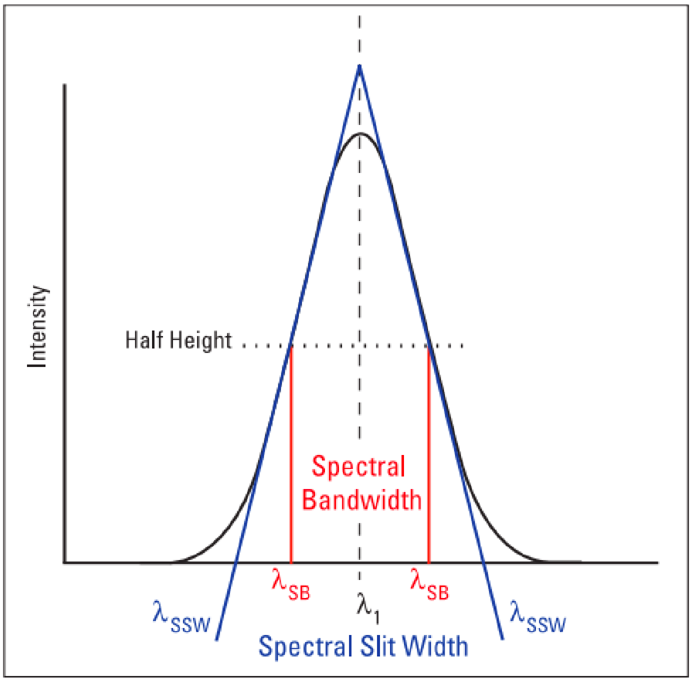
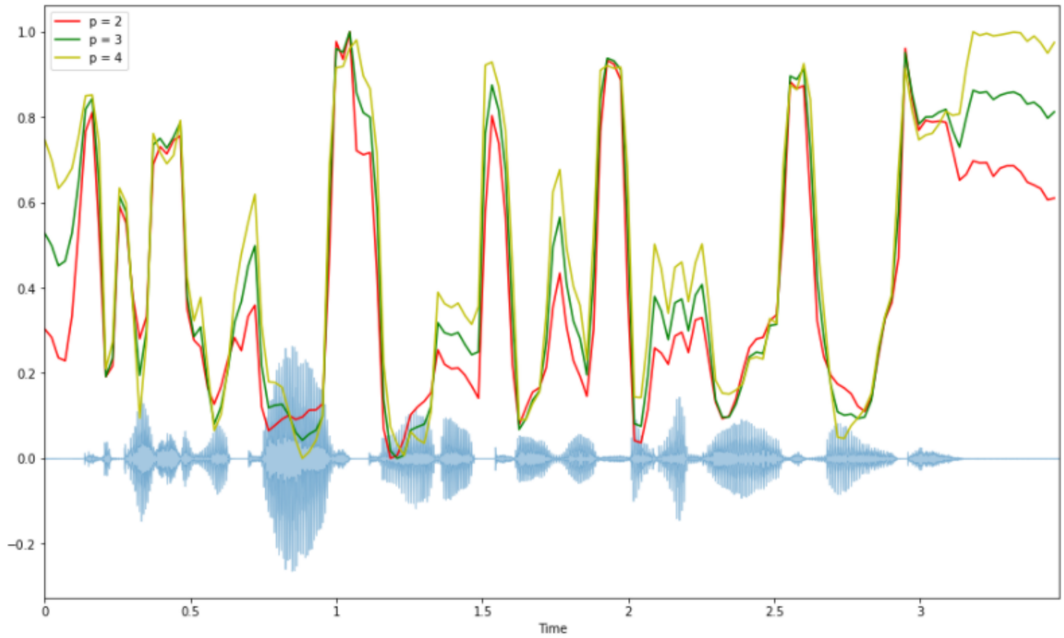


Figure 1: Gaussian intensity distribution of wavelengths emerging from the monochromator. The spectral bandwidth is defined by the red boundaries and λ_{SB}. The spectral slit width is depicted by the blue boundaries and λ_{SSW}.

```
spectral_bandwidth_2 = librosa.feature.spectral_bandwidth(y+0.01, sr=sr)[0]
spectral_bandwidth_3 = librosa.feature.spectral_bandwidth(y+0.01, sr=sr, p=3)[0]
spectral_bandwidth_4 = librosa.feature.spectral_bandwidth(y+0.01, sr=sr, p=4)[0]
plt.figure(figsize=(15, 9))
librosa.display.waveplot(y, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_bandwidth_2), color='r')
plt.plot(t, normalize(spectral_bandwidth_3), color='g')
plt.plot(t, normalize(spectral_bandwidth_4), color='y')
plt.legend(('p = 2', 'p = 3', 'p = 4'))
```

<matplotlib.legend.Legend at 0x7f58f2b6b3d0>

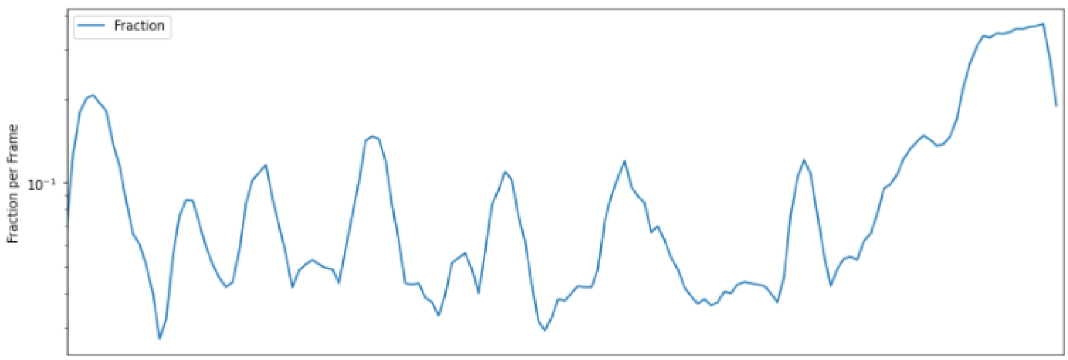


Скорость пересечения нуля

Простой способ измерения гладкости сигнала — вычисление числа пересечений нуля в пределах сегмента этого сигнала. Голосовой сигнал колеблется медленно. Например, сигнал 100 Гц будет пересекать ноль 100 раз в секунду, тогда как “немой” фрикативный сигнал может иметь 3000 пересечений нуля в секунду.

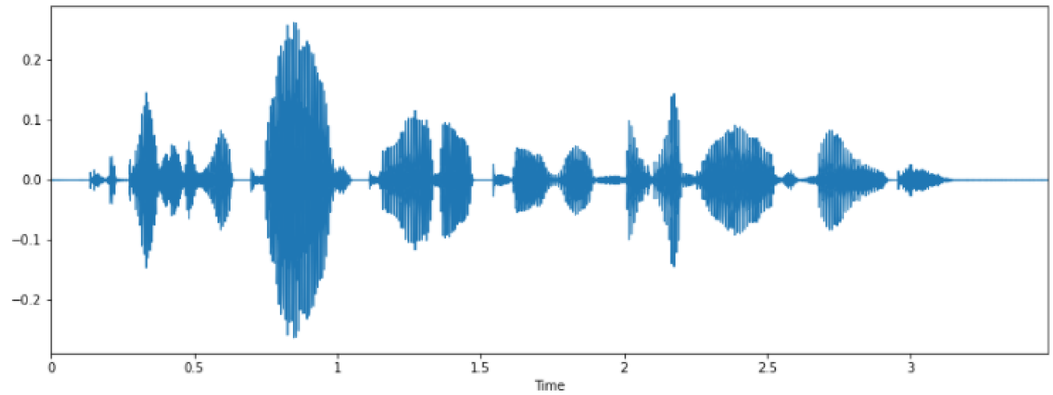
```
# Zero Crossing Rate
zrate=librosa.feature.zero_crossing_rate(y_harmonic)
plt.figure(figsize=(14,5))
plt.semilogy(zrate.T, label='Fraction')
plt.ylabel('Fraction per Frame')
plt.xticks([])
plt.xlim([0, rolloff.shape[-1]])
plt.legend()
```

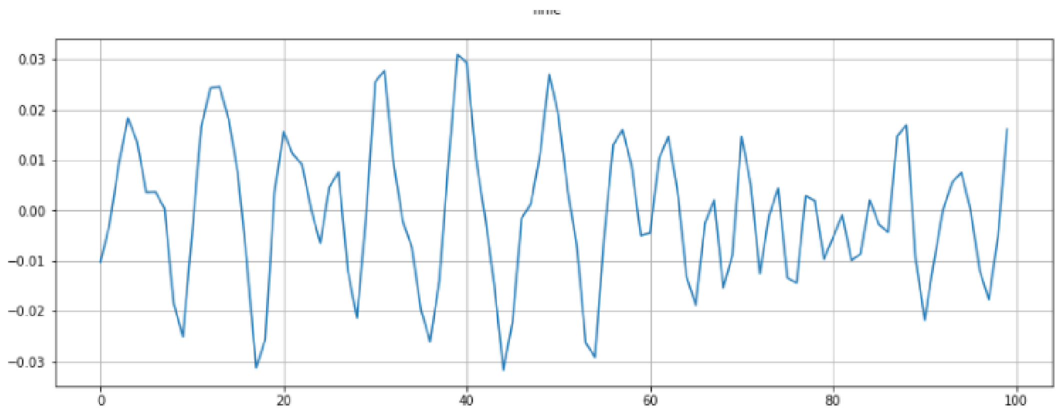
<matplotlib.legend.Legend at 0x7f58f2b54490>




```
# Построение графика сигнала:
plt.figure(figsize=(14, 5))
librosa.display.waveplot(y, sr=sr)
```

```
# Увеличение масштаба:
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(y[n0:n1])
plt.grid()
```






```
zero_crossings = librosa.zero_crossings(y[n0:n1], pad=False)
print(sum(zero_crossings))
```

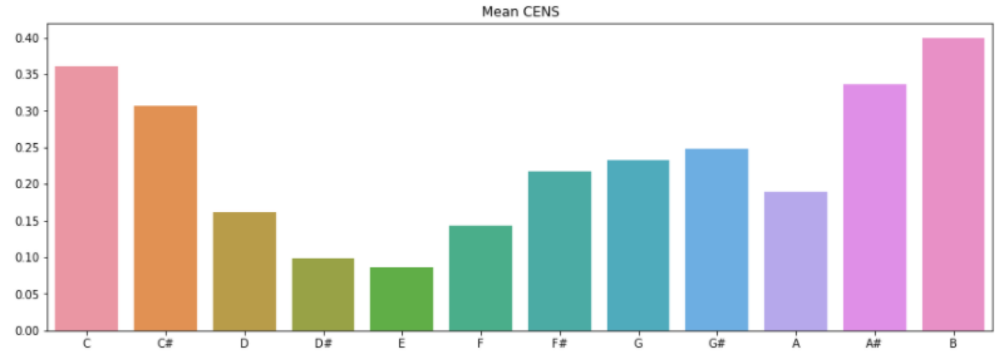
33

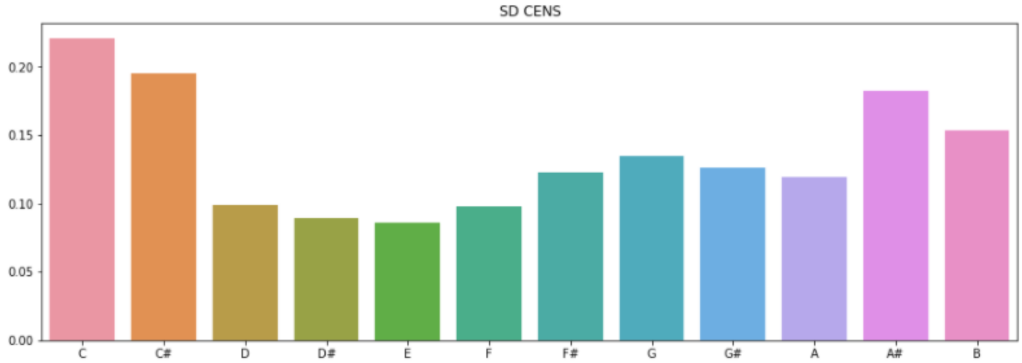
```
#Feature Generation. Chroma Energy Normalized

chroma_mean=np.mean(chroma,axis=1)
chroma_std=np.std(chroma,axis=1)
#plot the summary
octave=['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
plt.figure(figsize=(15,5))
plt.title('Mean CENS')
sns.barplot(x=octave,y=chroma_mean)

plt.figure(figsize=(15,5))
plt.title('SD CENS')
sns.barplot(x=octave,y=chroma_std)
#Generate the chroma Dataframe
chroma_df=pd.DataFrame()
for i in range(0,12):
    chroma_df['chroma_mean_'+str(i)]=chroma_mean[i]
for i in range(0,12):
    chroma_df['chroma_std_'+str(i)]=chroma_mean[i]
chroma_df.loc[0]=np.concatenate((chroma_mean,chroma_std),axis=0)
```

```
chroma_df
```

	chroma_mean_0	chroma_mean_1	chroma_mean_2	chroma_mean_3	chroma_mean_4	chroma_mean_5	chroma_mean_6	chroma_mean_7	chroma_mean_8	chroma_mean_9	.
0	0.360248	0.306567	0.162005	0.097873	0.086152	0.143012	0.216519	0.233423	0.247545	0.18933	

1 rows × 24 columns

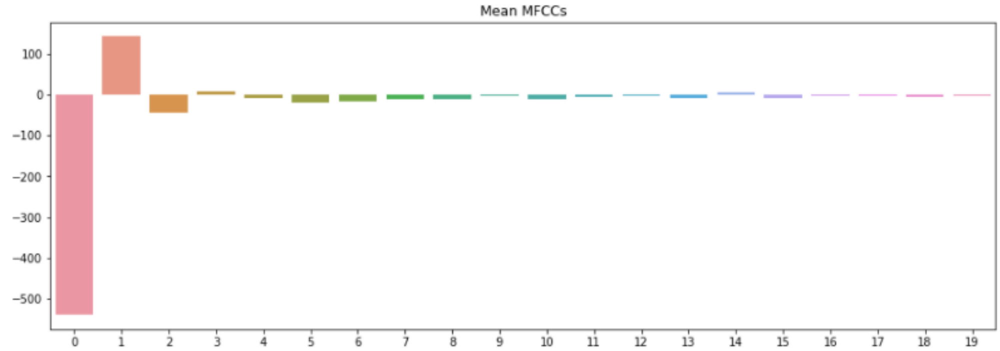


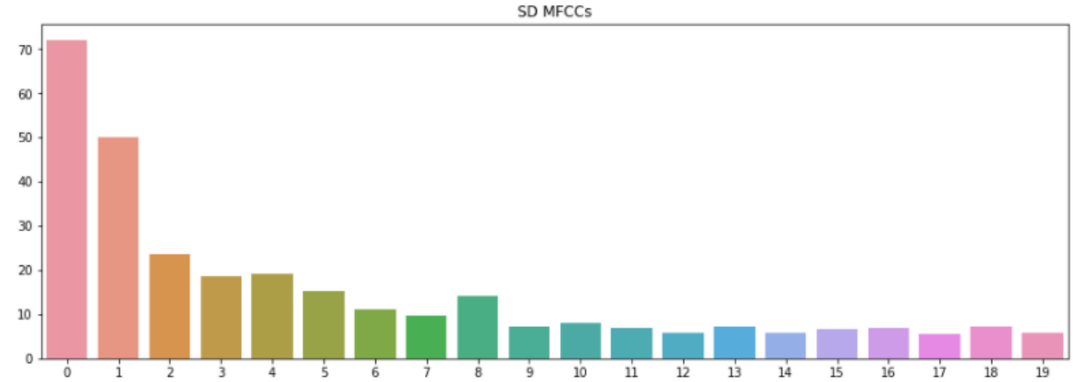



```
# Feature Generation
# MFCCs
mfccs_mean=np.mean(mfccs,axis=1)
mfccs_std=np.std(mfccs,axis=1)

coeffs=np.arange(0,20)
plt.figure(figsize=(15,5))
plt.title('Mean MFCCs')
sns.barplot(x=coeffs,y=mfccs_mean)

plt.figure(figsize=(15,5))
plt.title('SD MFCCs')
sns.barplot(x=coeffs,y=mfccs_std)
#Generate the chroma Dataframe
mfccs_df=pd.DataFrame()
for i in range(0,20):
    mfccs_df['mfccs_mean_'+str(i)]=mfccs_mean[i]
for i in range(0,20):
    mfccs_df['mfccs_std_'+str(i)]=mfccs_mean[i]
mfccs_df.loc[0]=np.concatenate((mfccs_mean,mfccs_std),axis=0)
mfccs_df
```

	mfccs_mean_0	mfccs_mean_1	mfccs_mean_2	mfccs_mean_3	mfccs_mean_4	mfccs_mean_5	mfccs_mean_6	mfccs_mean_7	mfccs_mean_8	mfccs_mean_9	...	mfccs_std_10
0	-539.370178	142.519714	-44.240788	8.094377	-7.475252	-19.327667	-16.648611	-10.093089	-11.561587	-2.548789	...	8.075235

1 rows × 40 columns

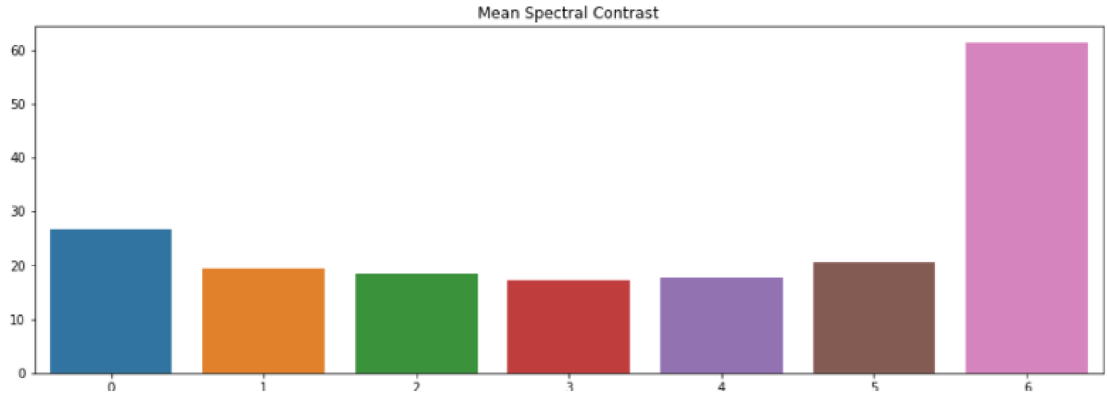


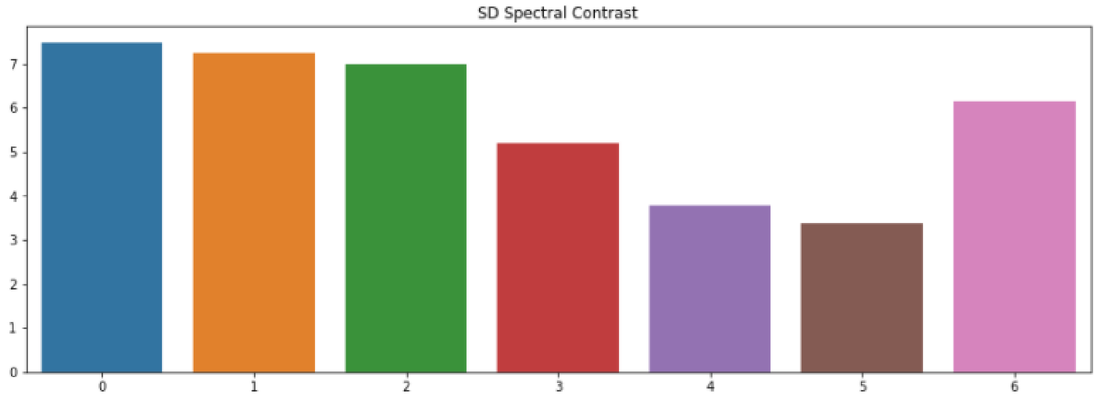



```
# Spectral Features
# Spectral Centroid

cent_mean=np.mean(cent)
cent_std=np.std(cent)
cent_skew=scipy.stats.skew(cent,axis=1)[0]
print('Mean: '+str(cent_mean))
print('SD: '+str(cent_std))
print('Skewness: '+str(cent_skew))
```

```
Mean: 1932.3248618681969
SD: 1047.7706155489461
Skewness: 0.830979185714757
```

```
# Spectral Contrast
contrast_mean=np.mean(contrast,axis=1)
contrast_std=np.std(contrast,axis=1)

conts=np.arange(0,7)
plt.figure(figsize=(15,5))
plt.title('Mean Spectral Contrast')
sns.barplot(x=conts,y=contrast_mean)

plt.figure(figsize=(15,5))
plt.title('SD Spectral Contrast')
sns.barplot(x=conts,y=contrast_std)
#Generate the chroma Dataframe
contrast_df=pd.DataFrame()
```






```
# Spectral Rolloff
rolloff_mean=np.mean(rolloff)
rolloff_std=np.std(rolloff)
rolloff_skew=scipy.stats.skew(rolloff,axis=1)[0]
print('Mean: '+str(rolloff_mean))
print('SD: '+str(rolloff_std))
print('Skewness: '+str(rolloff_skew))
```

```
Mean: 3284.74658203125
SD: 1850.2706548960884
Skewness: 0.5133480526070657
```

```
spectral_df=pd.DataFrame()
collist=['cent_mean','cent_std','cent_skew']
for i in range(0,7):
    collist.append('contrast_mean_'+str(i))
for i in range(0,7):
    collist.append('contrast_std_'+str(i))
collist=collist+['rolloff_mean','rolloff_std','rolloff_skew']
for c in collist:
    spectral_df[c]=0
data=np.concatenate(([cent_mean,cent_std,cent_skew],contrast_mean,contrast_std,[rolloff_mean,rolloff_std,rolloff_std]),axis=0)
```

```
spectral_df.loc[0]=data
spectral_df
```

	cent_mean	cent_std	cent_skew	contrast_mean_0	contrast_mean_1	contrast_mean_2	contrast_mean_3	contrast_mean_4	contrast_mean_5	contrast_mean_6	contrast_s
0	1932.324862	1047.770616	0.830979	26.800425	19.344108	18.518281	17.337795	17.76409	20.498698	61.331761	7.48

```
# Zero Crossing Rate
zrate_mean=np.mean(zrate)
zrate_std=np.std(zrate)
zrate_skew=scipy.stats.skew(zrate,axis=1)[0]
print('Mean: '+str(zrate_mean))
print('SD: '+str(zrate_std))
print('Skewness: '+str(zrate_skew))
```

```
Mean: 0.09929361979166666
SD: 0.08465011647434162
Skewness: 1.971980706672974
```

```
zrate_df=pd.DataFrame()
zrate_df['zrate_mean']=0
zrate_df['zrate_std']=0
zrate_df['zrate_skew']=0
zrate_df.loc[0]=[zrate_mean,zrate_std,zrate_skew]
zrate_df
```

	zrate_mean	zrate_std	zrate_skew
0	0.099294	0.08465	1.971981

```
# Beat and Tempo
beat_df=pd.DataFrame()
beat_df['tempo']=tempo
beat_df.loc[0]=tempo
beat_df
```

	tempo
0	143.554688

```
# Generate the  audio_df
audio_df=pd.concat((chroma_df,mfccs_df,spectral_df,zrate_df,beat_df),axis=1)
audio_df.head()
```

	chroma_mean_0	chroma_mean_1	chroma_mean_2	chroma_mean_3	chroma_mean_4	chroma_mean_5	chroma_mean_6	chroma_mean_7	chroma_mean_8	chroma_mean_9
0	0.360248	0.306567	0.162005	0.097873	0.086152	0.143012	0.216519	0.233423	0.247545	0.18933

```
import librosa
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
import pathlib
import csv

import warnings
warnings.filterwarnings('ignore')
```

```
    # y_harmonic, y_percussive = librosa.effects.hpss(y)

    # tempo, beat_frames = librosa.beat.beat_track(y=y_harmonic, sr=sr)
    # chroma = librosa.feature.chroma_cens(y=y_harmonic, sr=sr)
    # mfccs = librosa.feature.mfcc(y=y_harmonic, sr=sr, n_mfcc=13)
    # cent = librosa.feature.spectral_centroid(y=y, sr=sr)
    # contrast = librosa.feature.spectral_contrast(y=y_harmonic, sr=sr)
    # rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
    # zrate = librosa.feature.zero_crossing_rate(y_harmonic)

    # mfccs_std = np.std(mfccs, axis=1)
    # cent_mean = np.mean(cent)
    # cent_std = np.std(cent)
    # cent_skew = scipy.stats.skew(cent, axis=1)[0]
    # contrast_mean = np.mean(contrast,axis=1)
    # contrast_std = np.std(contrast,axis=1)

    #rolloff_mean=np.mean(rolloff)
    #rolloff_std=np.std(rolloff)
    #zrate_mean = np.mean(zrate)
    #zrate_std = np.std(zrate)
    #zrate_skew = scipy.stats.skew(zrate,axis=1)[0]
```

```
def extract_features(directory, file):
    name = f'{directory}/{file}'
    y, sr = librosa.load(name, mono=True, duration=5)

    features = []
    features.append(file) # filename
    features.extend([np.mean(e) for e in librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)]) # mfcc_mean<0..20>
    features.extend([np.std(e) for e in librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)])  # mfcc_std
    features.append(np.mean(librosa.feature.spectral_centroid(y=y, sr=sr).T, axis = 0)[0])  # cent_mean
    features.append(np.std(librosa.feature.spectral_centroid(y=y,sr=sr).T, axis = 0)[0])   # cent_std
    features.append(scipy.stats.skew(librosa.feature.spectral_centroid(y=y,sr=sr).T, axis = 0)[0]) # cent_skew
    features.append(np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr).T, axis = 0)[0])  # rolloff_mean
    features.append(np.std(librosa.feature.spectral_rolloff(y=y, sr=sr).T, axis = 0)[0])  # rolloff_std


    features.append(directory.split('/')[-1])
    return features
```

```
#Списки файлов
human_dir, _, human_files = next(os.walk('../input/audioset/Training_Data/human'))
spoof_dir, _, spoof_files = next(os.walk('../input/audioset/Training_Data/spoof'))
print(f"Human files: {len(human_files)}\nSpoof files: {len(spoof_files)}")
```

```
Human files: 10322
Spoof files: 39678
```

```
buffer = []
buffer_size = 5000
buffer_counter = 0

#Создание заголовка для файла CSV.
header = ['filename']
header.extend([f'mfcc_mean{i}' for i in range(1, 21)])
header.extend([f'mfcc_std{i}' for i in range(1, 21)])
header.extend(['cent_mean', 'cent_std', 'cent_skew', 'rolloff_mean', 'rolloff_std',  'label'])

with open('dataset.csv', 'w', newline='') as file:
    writer = csv.writer(file, delimiter=',')
    writer.writerow(header)
    for directory, files in [(human_dir, human_files), (spoof_dir, spoof_files)]:
        for file in files:
            features = extract_features(directory, file)
            if buffer_counter + 1 == buffer_size:
                buffer.append(features)
                writer.writerows(buffer)
                print(f"- [{directory.split('/')[-1]}] Write {len(buffer)} rows")
                buffer = []
                buffer_counter = 0
            else:
                buffer.append(features)
                buffer_counter += 1
        if buffer:
            writer.writerows(buffer)
            print(f"- [{directory.split('/')[-1]}] Write {len(buffer)} rows")
        print(f"- [{directory.split('/')[-1]}] Writing complete")
        buffer = []
        buffer_counter = 0
```

```
data = pd.read_csv('../input/datatrain/dataset.csv')
data
```

]:

	filename	mfcc_mean1	mfcc_mean2	mfcc_mean3	mfcc_mean4	mfcc_mean5	mfcc_mean6	mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std17	r
0	human_09165.wav	-570.01830	149.887220	-24.901587	60.922962	-12.673909	7.521337	-4.852243	1.571152	4.497556	...	5.337415	
1	human_02634.wav	-514.01434	142.528900	-17.022379	44.921906	5.200197	13.416348	1.655196	3.223727	-7.332282	...	9.586616	
2	human_04952.wav	-472.95145	108.426730	-36.866430	33.368526	-36.298060	4.874664	-24.286007	-0.044528	-11.721227	...	9.245657	
3	human_02581.wav	-340.25310	124.741165	0.145759	56.383804	-27.436000	14.926692	-11.114817	-8.257433	-10.501668	...	6.046317	
4	human_04093.wav	-286.76663	142.639470	-40.818730	30.503517	-22.006926	3.530504	-30.791471	2.681680	-24.251282	...	8.346058	
...	...	...	...	...	...	...	...	...	...	...	...	...	
49995	spoof_24803.wav	-289.30206	113.028740	-66.136120	28.582876	-54.378902	-15.437898	-21.146448	-25.286797	-13.655656	...	6.999405	
49996	spoof_07481.wav	-356.77463	119.486210	-15.072644	52.494940	-9.573299	15.345481	4.927555	-4.417617	3.744606	...	8.014559	
49997	spoof_23235.wav	-284.19960	94.524950	-40.940483	51.611300	-14.198608	6.771314	-15.215896	8.838337	-20.167315	...	11.460436	
49998	spoof_24962.wav	-497.06060	160.509540	-6.562953	34.603947	-3.608054	17.226690	2.268985	-12.877869	-14.338764	...	11.389318	
49999	spoof_34164.wav	-490.97134	165.186500	-11.087096	45.722050	9.384361	12.521290	-11.607862	-10.304213	-4.657143	...	8.917649	

50000 rows × 47 columns

mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std17	mfcc_std18	mfcc_std19	mfcc_std20	cent_mean	cent_std	cent_skew	rolloff_mean	rolloff_std	label
-4.852243	1.571152	4.497556	...	5.337415	6.178011	5.065423	5.297127	1071.430854	754.369980	3.276589	2116.482519	1428.790189	human
1.655196	3.223727	-7.332282	...	9.586616	7.806422	5.891542	6.204598	1004.950466	563.432582	1.735601	1912.068685	1268.295235	human
-24.286007	-0.044528	-11.721227	...	9.245657	6.507904	6.851704	6.136752	2179.472415	1040.904297	1.007797	3842.600098	1671.209878	human
-11.114817	-8.257433	-10.501668	...	6.046317	9.178984	6.305259	4.464326	1312.326079	414.879780	0.459081	3154.614258	1201.035814	human
-30.791471	2.681680	-24.251282	...	8.346058	7.480830	6.613024	5.334562	1695.547157	1001.886957	1.391572	3055.521647	1679.586495	human
...	...	...	...	...	...	...	...	...	...	...	...	...	...
-21.146448	-25.286797	-13.655656	...	6.999405	5.920071	5.365368	5.823417	2040.704878	857.159446	1.110293	3563.246663	1018.126036	spoof
4.927555	-4.417617	3.744606	...	8.014559	6.151502	4.985438	5.889870	1350.941452	639.808203	2.537737	3221.985468	1175.317793	spoof
-15.215896	8.838337	-20.167315	...	11.460436	6.068170	5.646905	6.682852	2117.193404	1435.380378	0.507224	3704.059855	2003.513404	spoof
2.268985	-12.877869	-14.338764	...	11.389318	6.435936	8.575619	8.593656	1334.613405	1116.990417	2.105450	2386.885232	1821.254058	spoof
-11.607862	-10.304213	-4.657143	...	8.917649	10.322688	6.432636	6.567556	981.910137	503.939778	2.235306	1999.846395	1230.095025	spoof

```
print(data.dtypes)
```

```
filename         object
mfcc_mean1      float64
mfcc_mean2      float64
mfcc_mean3      float64
mfcc_mean4      float64
mfcc_mean5      float64
mfcc_mean6      float64
mfcc_mean7      float64
mfcc_mean8      float64
mfcc_mean9      float64
mfcc_mean10     float64
mfcc_mean11     float64
mfcc_mean12     float64
mfcc_mean13     float64
mfcc_mean14     float64
mfcc_mean15     float64
mfcc_mean16     float64
mfcc_mean17     float64
mfcc_mean18     float64
mfcc_mean19     float64
mfcc_mean20     float64
mfcc_std1       float64
mfcc_std2       float64
mfcc_std3       float64
mfcc_std4       float64
mfcc_std5       float64
mfcc_std6       float64
mfcc_std7       float64
mfcc_std8       float64
mfcc_std9       float64
mfcc_std10      float64
mfcc_std11      float64
mfcc_std12      float64
mfcc_std13      float64
mfcc_std14      float64
mfcc_std15      float64
mfcc_std16      float64
mfcc_std17      float64
mfcc_std18      float64
mfcc_std19      float64
mfcc_std20      float64
cent_mean       float64
cent_std        float64
cent_skew       float64
rolloff_mean    float64
rolloff_std     float64
label            object
dtype: object
```

```
data['label'].value_counts()
```

```
spoof    39678
human    10322
Name: label, dtype: int64
```

```
y = data['label'].values
y[0:5]
```

```
array(['human', 'human', 'human', 'human', 'human'], dtype=object)
```

```
data.columns
```

```
Index(['filename', 'mfcc_mean1', 'mfcc_mean2', 'mfcc_mean3', 'mfcc_mean4',
       'mfcc_mean5', 'mfcc_mean6', 'mfcc_mean7', 'mfcc_mean8', 'mfcc_mean9',
       'mfcc_mean10', 'mfcc_mean11', 'mfcc_mean12', 'mfcc_mean13',
       'mfcc_mean14', 'mfcc_mean15', 'mfcc_mean16', 'mfcc_mean17',
       'mfcc_mean18', 'mfcc_mean19', 'mfcc_mean20', 'mfcc_std1', 'mfcc_std2',
       'mfcc_std3', 'mfcc_std4', 'mfcc_std5', 'mfcc_std6', 'mfcc_std7',
       'mfcc_std8', 'mfcc_std9', 'mfcc_std10', 'mfcc_std11', 'mfcc_std12',
       'mfcc_std13', 'mfcc_std14', 'mfcc_std15', 'mfcc_std16', 'mfcc_std17',
       'mfcc_std18', 'mfcc_std19', 'mfcc_std20', 'cent_mean', 'cent_std',
       'cent_skew', 'rolloff_mean', 'rolloff_std', 'label'],
      dtype='object')
```

```
X = data[['mfcc_mean1', 'mfcc_mean2', 'mfcc_mean3', 'mfcc_mean4',
       'mfcc_mean5', 'mfcc_mean6', 'mfcc_mean7', 'mfcc_mean8', 'mfcc_mean9',
       'mfcc_mean10', 'mfcc_mean11', 'mfcc_mean12', 'mfcc_mean13',
       'mfcc_mean14', 'mfcc_mean15', 'mfcc_mean16', 'mfcc_mean17',
       'mfcc_mean18', 'mfcc_mean19', 'mfcc_mean20', 'mfcc_std1', 'mfcc_std2',
       'mfcc_std3', 'mfcc_std4', 'mfcc_std5', 'mfcc_std6', 'mfcc_std7',
       'mfcc_std8', 'mfcc_std9', 'mfcc_std10', 'mfcc_std11', 'mfcc_std12',
       'mfcc_std13', 'mfcc_std14', 'mfcc_std15', 'mfcc_std16', 'mfcc_std17',
       'mfcc_std18', 'mfcc_std19', 'mfcc_std20', 'cent_mean', 'cent_std',
       'cent_skew', 'rolloff_mean', 'rolloff_std']]
X[0:5]
```

	mfcc_mean1	mfcc_mean2	mfcc_mean3	mfcc_mean4	mfcc_mean5	mfcc_mean6	mfcc_mean7	mfcc_mean8	mfcc_mean9	mfcc_mean10	...	mfcc_std16	mfcc_std17	n
0	-570.01830	149.887220	-24.901587	60.922962	-12.673909	7.521337	-4.852243	1.571152	4.497556	1.335568	...	7.249393	5.337415	
1	-514.01434	142.528900	-17.022379	44.921906	5.200197	13.416348	1.655196	3.223727	-7.332282	7.746022	...	8.754921	9.586616	
2	-472.95145	108.426730	-36.866430	33.368526	-36.298060	4.874664	-24.286007	-0.044528	-11.721227	-7.771946	...	7.794386	9.245657	
3	-340.25310	124.741165	0.145759	56.383804	-27.436000	14.926692	-11.114817	-8.257433	-10.501668	-1.994703	...	5.696082	6.046317	
4	-286.76663	142.639470	-40.818730	30.503517	-22.006926	3.530504	-30.791471	2.681680	-24.251282	-14.366329	...	6.970195	8.346058	

5 rows × 45 columns

+ Code   + Markdown

```
from sklearn import preprocessing
X= preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[-2.03441182e+00,  1.36068245e+00,  5.19513326e-02,
         1.03487763e+00,  1.59238747e-01,  1.02319152e-02,
         7.98033745e-01,  3.19250906e-01,  1.68637946e+00,
         6.08115933e-01,  1.41484830e+00,  2.03695085e+00,
         1.16005545e+00,  2.22263007e+00,  7.41908425e-01,
         1.05955200e+00,  1.40218507e+00,  1.30807216e+00,
         5.35491141e-01,  6.69477251e-01, -7.39146967e-01,
        -6.67514585e-01,  2.47018075e-01, -1.58217572e-01,
        -9.03718770e-01,  1.06973801e-01, -1.21835578e-01,
         3.85330086e-01,  2.18404523e-01,  3.25089821e-01,
        -9.49725778e-01,  4.72384164e-01, -9.15550721e-02,
        -8.21230760e-01,  6.11987254e-01, -5.85780356e-01,
        -1.47628011e+00, -7.31976666e-01, -1.23072453e+00,
        -7.99303696e-01, -1.50113203e+00, -5.81236278e-01,
         2.77806216e+00, -1.48020868e+00, -2.71697740e-01],
       [-1.50622857e+00,  1.02863531e+00,  3.50791881e-01,
         1.77512097e-01,  1.07264136e+00,  3.90847673e-01,
         1.35091976e+00,  4.78619695e-01,  5.11339323e-01,
         1.48481072e+00,  5.61401914e-01,  1.28023962e+00,
         2.27159311e+00,  8.51444382e-01,  1.84187358e+00,
         2.15612891e+00,  6.78589295e-01,  2.31651399e+00,
         1.59100160e-01,  1.58133079e+00,  7.29097843e-01,
        -3.47861718e-02,  1.70119448e+00,  5.86791737e-01,
         4.26877424e-01, -2.55931023e-01,  9.93437053e-02,
         3.91374382e-01,  1.10818790e+00,  4.91536877e-01,
         1.23421979e+00,  9.20836287e-01, -3.69009295e-02,
         1.79991828e+00,  3.30013379e-01,  1.69149918e-01,
         6.10977611e-01,  9.01750312e-02, -7.23028349e-01,
        -3.29762138e-01, -1.65199013e+00, -1.14209683e+00,
         5.96249217e-01, -1.75093973e+00, -5.95428516e-01],
       [-1.11895718e+00, -5.10238785e-01, -4.01848115e-01,
        -4.41538905e-01, -1.04800268e+00, -1.60652537e-01,
        -8.53100493e-01,  1.63440003e-01,  7.53920408e-02,
        -6.37429355e-01,  2.10055461e-01, -7.36992917e-01,
        -6.00089905e-01,  7.31707275e-01, -4.39080815e-01,
        -4.09195804e-01, -1.02226958e-01, -2.17987446e-01,
         6.64190011e-01, -1.18139899e+00,  2.52439149e-02,
         3.67797989e-01,  2.37859838e-01,  2.05916720e-01,
         7.26508127e-01,  5.05156824e-02,  1.92177795e-01,
        -1.96627988e-01, -1.07536406e-01,  2.68676425e-01,
        -6.92150150e-01,  6.26737759e-01,  6.49067697e-01,
         4.60501531e-01, -3.74063572e-01, -3.12499677e-01,
         4.43494535e-01, -5.65420358e-01, -1.32955634e-01,
        -3.64866859e-01,  1.01324937e+00,  2.60431189e-01,
        -4.34213909e-01,  8.05906766e-01,  2.17281581e-01],
       [ 1.32544437e-01,  2.25956634e-01,  1.00194053e+00,
         7.91661328e-01, -5.95133664e-01,  4.88364161e-01,
         2.65952059e-01, -6.28585420e-01,  1.96529009e-01,
         1.52667393e-01,  1.02709440e-01,  5.68347570e-01,
         3.00445517e-01,  4.38768423e-01,  2.03155301e-01,
        -2.59441474e-01, -1.00343574e-01, -1.16872473e+00,
        -4.43382233e-01, -4.31769672e-01, -2.77232991e-01,
        -1.67728066e+00,  1.58673360e-01, -1.18290897e+00,
         3.51923381e-02, -9.46602955e-01,  1.00625283e+00,
         4.70407991e-01, -5.32966807e-01, -1.02464932e+00,
         2.22594671e-01, -7.67288283e-01, -2.45172227e-01,
        -6.73122618e-01, -3.35212807e-01, -1.36467088e+00,
        -1.12805915e+00,  7.83153942e-01, -4.68776467e-01,
        -1.23020947e+00, -9.54489651e-01, -1.57845664e+00,
        -1.21111469e+00, -1.05279745e-01, -7.31095987e-01],
       [ 6.36984801e-01,  1.03362482e+00, -5.51749920e-01,
        -5.95051270e-01, -3.17697111e-01, -2.47439251e-01,
        -1.40581867e+00,  4.26346559e-01, -1.16919955e+00,
        -1.53927827e+00, -2.83658739e-01, -1.48706683e+00,
        -6.03457687e-01, -2.26398191e+00, -6.04756320e-01,
        -9.73800654e-01, -1.57138358e+00, -8.83915668e-01,
        -2.71952781e+00, -1.68520203e+00, -4.78045277e-01,
         5.94569760e-02,  1.31769560e-01,  1.14395764e-01,
        -3.22881834e-02, -1.52163255e-01, -5.58080603e-01,
        -4.17836927e-02, -6.16027851e-01, -6.65961253e-01,
        -4.88069233e-01, -2.19351078e-01, -7.24236824e-02,
        -6.02953557e-01, -4.63686000e-01, -7.25781089e-01,
         1.60091155e-03, -7.42098387e-02, -2.79637833e-01,
```

```
        -7.79934164e-01, -8.48797177e-02,  1.45821442e-01,
         1.09155629e-01, -2.36520601e-01,  2.34177868e-01]])
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=17)
print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)

from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
yhat = clf.predict(X_test)
yhat [0:10]
```

```
Train set: (40000, 45) (40000,)
Test set: (10000, 45) (10000,)
]: array(['spoof', 'spoof', 'spoof', 'human', 'spoof', 'spoof', 'human',
          'spoof', 'spoof', 'human'], dtype=object)
```

```
print("Prediction:", yhat[0:20])
print("Real Value:", y_test[0:20])
```

```
Prediction: ['spoof' 'spoof' 'spoof' 'human' 'spoof' 'spoof' 'human' 'spoof' 'spoof'
 'human' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof'
 'spoof' 'human']
Real Value: ['spoof' 'spoof' 'spoof' 'human' 'spoof' 'spoof' 'human' 'spoof' 'spoof'
 'human' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof' 'spoof'
 'spoof' 'human']
```

```
# accuracy evaluation
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, clf.predict(X_train)))
print("Test set Accuracy: ",metrics.accuracy_score(y_test, yhat) )
```

```
Train set Accuracy:  0.979725
Test set Accuracy:  0.9713
```

```
from sklearn.metrics import classification_report,confusion_matrix

print('CONFUSION_MATRIX :\n')
print(confusion_matrix(y_test,yhat))
print('\n')
print('REPORT :\n')
print(classification_report(y_test,yhat))
```

```
CONFUSION_MATRIX :

[[1897  208]
 [  79 7816]]


REPORT :

              precision    recall  f1-score   support

       human       0.96      0.90      0.93      2105
       spoof       0.97      0.99      0.98      7895

    accuracy                           0.97     10000
   macro avg       0.97      0.95      0.96     10000
weighted avg       0.97      0.97      0.97     10000
```

```
#Списки файлов
testing_dir, _, testing_files  = next(os.walk('../input/testingset/Testing_Data'))
print(f"Testing Data files: {len(testing_files)}")
```

```
Testing Data files: 5000
```

```
buffer = []
buffer_size = 1000
buffer_counter = 0

#Создание заголовка для файла CSV.
header = ['filename']
header.extend([f'mfcc_mean{i}' for i in range(1, 21)])
header.extend([f'mfcc_std{i}' for i in range(1, 21)])
header.extend(['cent_mean', 'cent_std', 'cent_skew', 'rolloff_mean', 'rolloff_std',  'label'])

directory = testing_dir
with open('testset.csv', 'w', newline='') as file:
    writer = csv.writer(file, delimiter=',')
    writer.writerow(header)
    for file in testing_files:
        features = extract_features(testing_dir, file)
        if buffer_counter + 1 == buffer_size:
            buffer.append(features)
            writer.writerows(buffer)
            print(f"- [{directory.split('/')[-1]}] Write {len(buffer)} rows")
            buffer = []
            buffer_counter = 0
        else:
            buffer.append(features)
            buffer_counter += 1
    if buffer:
        writer.writerows(buffer)
        print(f"- [{directory.split('/')[-1]}] Write {len(buffer)} rows")
    print(f"- [{directory.split('/')[-1]}] Writing complete")
    buffer = []
    buffer_counter = 0
```

```
test_data = pd.read_csv('../input/testset/testset.csv')
test_data
```

	filename	mfcc_mean1	mfcc_mean2	mfcc_mean3	mfcc_mean4	mfcc_mean5	mfcc_mean6	mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std17	mfcc_std18
0	sample_2964.wav	-421.31635	158.14297	3.762873	0.339285	-20.170176	15.607149	-1.892440	-14.770078	-19.657373	...	7.618616	5.422542
1	sample_2226.wav	-416.44632	118.61284	-16.785368	51.839878	11.913440	13.446784	-6.196104	8.669424	-25.191850	...	8.669201	9.111779
2	sample_4401.wav	-477.58630	131.04564	-71.929794	-10.960677	0.577396	-45.477943	-23.479357	-0.222770	-20.360256	...	7.342184	5.159128
3	sample_3306.wav	-263.90262	199.49097	-29.954306	11.072295	-10.356196	-27.164942	2.328327	-7.718163	-26.647543	...	5.943689	3.845689
4	sample_1200.wav	-276.36260	132.21123	-17.764063	38.506527	-33.138515	9.466157	-19.513160	7.568661	-19.955858	...	10.855307	6.059436
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	sample_3167.wav	-294.17407	92.88919	-72.425735	69.098800	-25.029654	25.481490	-17.788597	6.090208	-7.980536	...	8.291508	7.199494
4996	sample_2405.wav	-174.62381	97.25732	-22.827950	33.976852	-63.088493	13.791640	-48.362650	-19.745262	-24.421131	...	7.644339	6.798235
4997	sample_0854.wav	-297.61664	114.18568	-35.433723	38.724453	-16.152746	0.787063	-14.684649	-7.068820	-13.637604	...	6.780140	7.252308
4998	sample_4273.wav	-327.96155	113.74922	-32.699190	34.492245	-26.412588	15.847356	-24.972742	-13.348909	-18.514206	...	8.254047	7.606890
4999	sample_3642.wav	-393.42680	114.87638	-31.839302	44.534110	-24.959799	28.009907	-21.089777	8.194910	-16.981745	...	8.025428	9.943584

5000 rows × 47 columns

6	mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std17	mfcc_std18	mfcc_std19	mfcc_std20	cent_mean	cent_std	cent_skew	rolloff_mean	rolloff_std	label
9	-1.892440	-14.770078	-19.657373	...	7.618616	5.422542	7.296792	6.131514	1142.540335	630.348327	1.632423	1925.009364	1271.735685	Testing_Data
4	-6.196104	8.669424	-25.191850	...	8.669201	9.111779	6.911362	7.285794	1704.155492	1210.310028	1.972223	3175.983085	1620.373868	Testing_Data
3	-23.479357	-0.222770	-20.360256	...	7.342184	5.159128	5.992408	5.915099	1647.501356	349.099047	0.161218	2788.288796	637.652969	Testing_Data
2	2.328327	-7.718163	-26.647543	...	5.943689	3.845689	3.651412	3.736007	975.734965	348.523682	0.862979	1805.337175	881.565686	Testing_Data
7	-19.513160	7.568661	-19.955858	...	10.855307	6.059436	7.148800	6.658545	1789.201407	849.743733	1.938799	3589.515177	1498.972909	Testing_Data
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	-17.788597	6.090208	-7.980536	...	8.291508	7.199494	6.383205	6.060671	2617.182254	615.625500	1.494126	4411.654000	849.795947	Testing_Data
0	-48.362650	-19.745262	-24.421131	...	7.644339	6.798235	6.570068	5.740641	2735.314620	1403.122506	0.814622	4989.305579	1711.641166	Testing_Data
3	-14.684649	-7.068820	-13.637604	...	6.780140	7.252308	6.860434	5.302031	1641.550469	1372.568333	1.103572	2860.476685	2152.021818	Testing_Data
6	-24.972742	-13.348909	-18.514206	...	8.254047	7.606890	6.976487	6.367769	2054.700602	1002.288786	0.771484	3983.742269	1780.246041	Testing_Data
7	-21.089777	8.194910	-16.981745	...	8.025428	9.943584	5.098617	5.935461	1917.240892	246.376257	-0.028781	4380.494173	320.715965	Testing_Data

```
print(test_data.dtypes)
```

```
filename         object
mfcc_mean1      float64
mfcc_mean2      float64
mfcc_mean3      float64
mfcc_mean4      float64
mfcc_mean5      float64
mfcc_mean6      float64
mfcc_mean7      float64
mfcc_mean8      float64
mfcc_mean9      float64
mfcc_mean10     float64
mfcc_mean11     float64
mfcc_mean12     float64
mfcc_mean13     float64
mfcc_mean14     float64
mfcc_mean15     float64
mfcc_mean16     float64
mfcc_mean17     float64
mfcc_mean18     float64
mfcc_mean19     float64
mfcc_mean20     float64
mfcc_std1       float64
mfcc_std2       float64
mfcc_std3       float64
mfcc_std4       float64
mfcc_std5       float64
mfcc_std6       float64
mfcc_std7       float64
mfcc_std8       float64
mfcc_std9       float64
```

```
mfcc_std10      float64
mfcc_std11      float64
mfcc_std12      float64
mfcc_std13      float64
mfcc_std14      float64
mfcc_std15      float64
mfcc_std16      float64
mfcc_std17      float64
mfcc_std18      float64
mfcc_std19      float64
mfcc_std20      float64
cent_mean       float64
cent_std        float64
cent_skew       float64
rolloff_mean    float64
rolloff_std     float64
label            object
dtype: object
```

[49]:

```
test_data['test_predict'] = ""
test_data
```

[49]:

5	mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std18	mfcc_std19	mfcc_std20	cent_mean	cent_std	cent_skew	rolloff_mean	rolloff_std	label	test_predict
9	-1.892440	-14.770078	-19.657373	...	5.422542	7.296792	6.131514	1142.540335	630.348327	1.632423	1925.009364	1271.735685	Testing_Data	
4	-6.196104	8.669424	-25.191850	...	9.111779	6.911362	7.285794	1704.155492	1210.310028	1.972223	3175.983085	1620.373868	Testing_Data	
3	-23.479357	-0.222770	-20.360256	...	5.159128	5.992408	5.915099	1647.501356	349.099047	0.161218	2788.288796	637.652969	Testing_Data	
2	2.328327	-7.718163	-26.647543	...	3.845689	3.651412	3.736007	975.734965	348.523682	0.862979	1805.337175	881.565686	Testing_Data	
7	-19.513160	7.568661	-19.955858	...	6.059436	7.148800	6.658545	1789.201407	849.743733	1.938799	3589.515177	1498.972909	Testing_Data	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	-17.788597	6.090208	-7.980536	...	7.199494	6.383205	6.060671	2617.182254	615.625500	1.494126	4411.654000	849.795947	Testing_Data	
0	-48.362650	-19.745262	-24.421131	...	6.798235	6.570068	5.740641	2735.314620	1403.122506	0.814622	4989.305579	1711.641166	Testing_Data	
3	-14.684649	-7.068820	-13.637604	...	7.252308	6.860434	5.302031	1641.550469	1372.568333	1.103572	2860.476685	2152.021818	Testing_Data	
5	-24.972742	-13.348909	-18.514206	...	7.606890	6.976487	6.367769	2054.700602	1002.288786	0.771484	3983.742269	1780.246041	Testing_Data	
7	-21.089777	8.194910	-16.981745	...	9.943584	5.098617	5.935461	1917.240892	246.376257	-0.028781	4380.494173	320.715965	Testing_Data	

[50]:

```
test_X = test_data[['mfcc_mean1', 'mfcc_mean2', 'mfcc_mean3', 'mfcc_mean4',
       'mfcc_mean5', 'mfcc_mean6', 'mfcc_mean7', 'mfcc_mean8', 'mfcc_mean9',
       'mfcc_mean10', 'mfcc_mean11', 'mfcc_mean12', 'mfcc_mean13',
       'mfcc_mean14', 'mfcc_mean15', 'mfcc_mean16', 'mfcc_mean17',
       'mfcc_mean18', 'mfcc_mean19', 'mfcc_mean20', 'mfcc_std1', 'mfcc_std2',
       'mfcc_std3', 'mfcc_std4', 'mfcc_std5', 'mfcc_std6', 'mfcc_std7',
       'mfcc_std8', 'mfcc_std9', 'mfcc_std10', 'mfcc_std11', 'mfcc_std12',
       'mfcc_std13', 'mfcc_std14', 'mfcc_std15', 'mfcc_std16', 'mfcc_std17',
       'mfcc_std18', 'mfcc_std19', 'mfcc_std20', 'cent_mean', 'cent_std',
       'cent_skew', 'rolloff_mean', 'rolloff_std']]
test_X[0:5]
```

[50]:

	mfcc_mean1	mfcc_mean2	mfcc_mean3	mfcc_mean4	mfcc_mean5	mfcc_mean6	mfcc_mean7	mfcc_mean8	mfcc_mean9	mfcc_mean10	...	mfcc_std16	mfcc_std17	mfcc
0	-421.31635	158.14297	3.762873	0.339285	-20.170176	15.607149	-1.892440	-14.770078	-19.657373	-17.111542	...	8.171947	7.618616	5.4
1	-416.44632	118.61284	-16.785368	51.839878	11.913440	13.446784	-6.196104	8.669424	-25.191850	-1.254284	...	6.989859	8.669201	9.
2	-477.58630	131.04564	-71.929794	-10.960677	0.577396	-45.477943	-23.479357	-0.222770	-20.360256	-23.043287	...	7.428599	7.342184	5.
3	-263.90262	199.49097	-29.954306	11.072295	-10.356196	-27.164942	2.328327	-7.718163	-26.647543	-5.702496	...	5.164096	5.943689	3.
4	-276.36260	132.21123	-17.764063	38.506527	-33.138515	9.466157	-19.513160	7.568661	-19.955858	-2.004704	...	12.068744	10.855307	6.

5 rows × 45 columns

```
from sklearn import preprocessing
test_X= preprocessing.StandardScaler().fit(test_X).transform(test_X)
test_X[0:5]
```

```
array([[-0.59846779,  1.27830468,  1.14231646, -1.15106973, -0.32067778,
         0.55749275,  1.11575784, -1.16784287, -0.60197061, -1.49493694,
        -0.07910548,  0.24673079, -0.11489797,  0.56681922, -0.6070711 ,
         1.32602459, -0.75774024,  0.1837474 ,  0.28787611, -0.35809035,
         0.34688452,  0.34783412, -1.00154936,  0.25086903, -0.05942185,
        -1.49464097,  0.58579216, -0.49713181, -1.47735148, -0.01959043,
        -1.37752971,  0.05965666, -0.27927721,  0.94240222,  0.56926599,
         0.20151779,  0.11979675, -0.94658232,  0.47488627, -0.0077104 ,
        -1.22479858, -0.3145266 ,  0.71837478, -1.4755317 , -0.01035188],
       [-0.54944387, -0.13552207,  0.5407441 ,  0.82781666,  1.18618105,
         0.45356848,  0.79104977,  0.62404784, -1.14579921,  0.15336684,
         1.29234813,  1.62088192,  0.9240488 , -0.54841206,  0.37804976,
         0.89703314,  0.8873653 ,  0.17493124,  0.32784135,  2.80504812,
        -0.25407709,  0.35861961,  0.72473371, -0.07340256,  0.87188611,
         1.28193095,  0.72795952,  0.57614686,  0.65036339,  0.63025118,
        -0.56946338,  1.75842202,  0.41960601,  1.250077  ,  1.81823437,
        -0.42688312,  0.68554672,  1.25458423,  0.22924242,  0.78605776,
        -0.11507444,  1.29735391,  1.15646558, -0.09546369,  0.64668019],
       [-1.16490648,  0.30914698, -1.07366959, -1.58526553,  0.65376543,
        -2.38100313, -0.51295798, -0.05573794, -0.67103727, -2.11152007,
        -0.36498685, -1.34489837, -3.16049246,  0.50487381, -0.01170415,
        -0.60488339, -2.60566372,  0.6656744 ,  0.90978226,  0.27437817,
         0.56422189,  0.49706365,  1.59824503,  0.44207057, -0.08301018,
         0.3316861 , -0.66258809, -0.68922109,  0.56487188,  0.189666  ,
        -1.33637401,  0.09915929,  0.89838229, -0.25539576,  0.6150189 ,
        -0.19364771, -0.0290645 , -1.10374709, -0.35642832, -0.1565335 ,
        -0.22702025, -1.09619931, -1.17838976, -0.52316611, -1.20532333],
       [ 0.9861298 ,  2.75714901,  0.15520897, -0.7386588 ,  0.14025164,
        -1.50005696,  1.43421137, -0.62874179, -1.28883847, -0.30900843,
        -0.05124398,  0.17484657, -1.60850562, -1.64309466, -0.61579392,
        -0.0701881 , -2.06524944,  0.12358266, -1.54834922, -2.44546235,
        -1.10449764, -0.1862609 , -1.08841723, -0.35738512, -0.75386486,
        -1.30512815, -1.20026655, -1.04205793, -0.85731322, -0.98746658,
        -1.10851851, -0.983164  , -0.93835773, -1.40443791, -1.41009598,
        -1.3974625 , -0.78216729, -1.88740464, -1.84840076, -1.65503732,
        -1.55439785, -1.09779842, -0.27363762, -1.60755347, -0.74565344],
       [ 0.86070202,  0.35083524,  0.51209173,  0.31548884, -0.92975675,
         0.26208058, -0.21371155,  0.53989727, -0.63130033,  0.07536346,
        -0.62463545, -0.8784294 , -0.3228627 , -1.245192  ,  0.31112551,
        -1.43810945, -0.53399371, -0.60816803, -0.37519578, -0.22215011,
         0.54246002, -0.52882619, -1.22241913,  1.11748677, -0.25969748,
        -0.35257688,  0.13949161,  0.4140448 ,  0.19114205,  0.93844944,
        -0.73220792,  0.13578073,  1.38923419,  0.70367237,  1.31311575,
         2.27306403,  1.86278548, -0.5665824 ,  0.38056772,  0.35471494,
         0.05297214,  0.29523638,  1.11337369,  0.36074287,  0.41789193]])
```

```
from sklearn import svm
test_yhat = clf.predict(test_X)
test_yhat [0:10]
```

```
from sklearn import svm
test_yhat = clf.predict(test_X)
test_yhat [0:10]
```

```
[52]: array(['human', 'spoof', 'spoof', 'spoof', 'spoof', 'spoof', 'spoof',
       'spoof', 'human', 'human'], dtype=object)
```

```
test_data['test_predict'] = test_yhat
test_data
```

[53]:

	filename	mfcc_mean1	mfcc_mean2	mfcc_mean3	mfcc_mean4	mfcc_mean5	mfcc_mean6	mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std18	mfcc_std19
0	sample_2964.wav	-421.31635	158.14297	3.762873	0.339285	-20.170176	15.607149	-1.892440	-14.770078	-19.657373	...	5.422542	7.296792
1	sample_2226.wav	-416.44632	118.61284	-16.785368	51.839878	11.913440	13.446784	-6.196104	8.669424	-25.191850	...	9.111779	6.911362
2	sample_4401.wav	-477.58630	131.04564	-71.929794	-10.960677	0.577396	-45.477943	-23.479357	-0.222770	-20.360256	...	5.159128	5.992408
3	sample_3306.wav	-263.90262	199.49097	-29.954306	11.072295	-10.356196	-27.164942	2.328327	-7.718163	-26.647543	...	3.845689	3.651412
4	sample_1200.wav	-276.36260	132.21123	-17.764063	38.506527	-33.138515	9.466157	-19.513160	7.568661	-19.955858	...	6.059436	7.148800
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	sample_3167.wav	-294.17407	92.88919	-72.425735	69.098800	-25.029654	25.481490	-17.788597	6.090208	-7.980536	...	7.199494	6.383205
4996	sample_2405.wav	-174.62381	97.25732	-22.827950	33.976852	-63.088493	13.791640	-48.362650	-19.745262	-24.421131	...	6.798235	6.570068
4997	sample_0854.wav	-297.61664	114.18568	-35.433723	38.724453	-16.152746	0.787063	-14.684649	-7.068820	-13.637604	...	7.252308	6.860434
4998	sample_4273.wav	-327.96155	113.74922	-32.699190	34.492245	-26.412588	15.847356	-24.972742	-13.348909	-18.514206	...	7.606890	6.976487
4999	sample_3642.wav	-393.42680	114.87638	-31.839302	44.534110	-24.959799	28.009907	-21.089777	8.194910	-16.981745	...	9.943584	5.098617

5000 rows × 48 columns

[53]:

5	mfcc_mean7	mfcc_mean8	mfcc_mean9	...	mfcc_std18	mfcc_std19	mfcc_std20	cent_mean	cent_std	cent_skew	rolloff_mean	rolloff_std	label	test_predict
9	-1.892440	-14.770078	-19.657373	...	5.422542	7.296792	6.131514	1142.540335	630.348327	1.632423	1925.009364	1271.735685	Testing_Data	human
4	-6.196104	8.669424	-25.191850	...	9.111779	6.911362	7.285794	1704.155492	1210.310028	1.972223	3175.983085	1620.373868	Testing_Data	spoof
3	-23.479357	-0.222770	-20.360256	...	5.159128	5.992408	5.915099	1647.501356	349.099047	0.161218	2788.288796	637.652969	Testing_Data	spoof
2	2.328327	-7.718163	-26.647543	...	3.845689	3.651412	3.736007	975.734965	348.523682	0.862979	1805.337175	881.565686	Testing_Data	spoof
7	-19.513160	7.568661	-19.955858	...	6.059436	7.148800	6.658545	1789.201407	849.743733	1.938799	3589.515177	1498.972909	Testing_Data	spoof
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	-17.788597	6.090208	-7.980536	...	7.199494	6.383205	6.060671	2617.182254	615.625500	1.494126	4411.654000	849.795947	Testing_Data	human
0	-48.362650	-19.745262	-24.421131	...	6.798235	6.570068	5.740641	2735.314620	1403.122506	0.814622	4989.305579	1711.641166	Testing_Data	spoof
3	-14.684649	-7.068820	-13.637604	...	7.252308	6.860434	5.302031	1641.550469	1372.568333	1.103572	2860.476685	2152.021818	Testing_Data	spoof
5	-24.972742	-13.348909	-18.514206	...	7.606890	6.976487	6.367769	2054.700602	1002.288786	0.771484	3983.742269	1780.246041	Testing_Data	spoof
7	-21.089777	8.194910	-16.981745	...	9.943584	5.098617	5.935461	1917.240892	246.376257	-0.028781	4380.494173	320.715965	Testing_Data	human

[54]:

```
testpredict = test_data[['filename','test_predict']]
testpredict
```

[54]:

	filename	test_predict
0	sample_2964.wav	human
1	sample_2226.wav	spoof
2	sample_4401.wav	spoof
3	sample_3306.wav	spoof
4	sample_1200.wav	spoof
...	...	...
4995	sample_3167.wav	human
4996	sample_2405.wav	spoof
4997	sample_0854.wav	spoof
4998	sample_4273.wav	spoof
4999	sample_3642.wav	human

5000 rows × 2 columns

```
testpredict[0:50]
```

[55]:

	filename	test_predict
0	sample_2964.wav	human
1	sample_2226.wav	spoof
2	sample_4401.wav	spoof
3	sample_3306.wav	spoof
4	sample_1200.wav	spoof
5	sample_3626.wav	spoof
6	sample_2555.wav	spoof
7	sample_3942.wav	spoof
8	sample_0381.wav	human
9	sample_4931.wav	human
10	sample_1839.wav	spoof
11	sample_4649.wav	spoof
12	sample_1168.wav	spoof
13	sample_1811.wav	spoof
14	sample_0437.wav	human
15	sample_3298.wav	spoof
16	sample_1880.wav	human

17	sample_2148.wav	spoof
18	sample_0624.wav	spoof
19	sample_0098.wav	spoof
20	sample_1773.wav	spoof
21	sample_4936.wav	spoof
22	sample_2086.wav	spoof
23	sample_4522.wav	spoof
24	sample_0093.wav	spoof
25	sample_3563.wav	spoof
26	sample_0655.wav	spoof
27	sample_0317.wav	human
28	sample_0384.wav	spoof
29	sample_2832.wav	spoof
30	sample_2817.wav	human
31	sample_3609.wav	spoof
32	sample_0443.wav	spoof
33	sample_4817.wav	spoof
34	sample_1795.wav	spoof
35	sample_4393.wav	spoof
36	sample_4112.wav	human
37	sample_1912.wav	spoof
38	sample_2553.wav	spoof
39	sample_3685.wav	spoof
40	sample_4457.wav	spoof
41	sample_1958.wav	spoof
42	sample_3208.wav	human
43	sample_3799.wav	human
44	sample_1876.wav	spoof
45	sample_2690.wav	spoof
46	sample_1251.wav	spoof
47	sample_4768.wav	spoof
48	sample_2836.wav	human
49	sample_4253.wav	spoof

[56]:

```
testpredict.to_csv('output.csv', index=False)
```