گزارش پروژه سیگنال‌ها و سیستم‌ها - بخش پردازش صوت

برای بخش پردازش صوت نیاز به کتابخونه‌های زیر داشتیم که به ترتیب علت استفاده‌شون رو توضیح می‌دم:

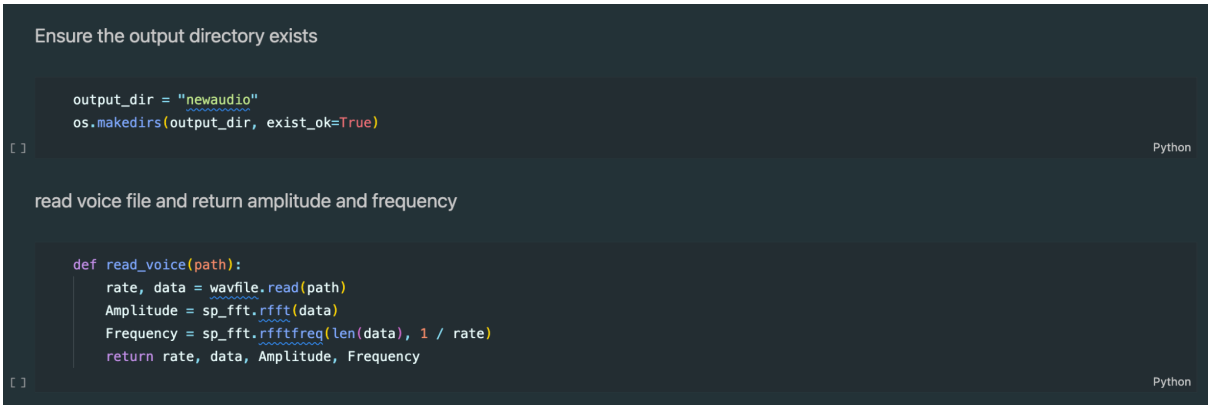
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wavfile
import scipy.fft as sp_fft
import scipy.signal as sp_signal
import os
```

کتابخونه نامپای که برای عملیات‌های عددی و محاسبات آرایه‌ها و در صورت نیاز بردارها استفاده می‌شه.

از متپلات‌لیب هم برای پلات‌کردن تصاویر مربوط به فرکانس ویس‌ها و یا نمودارهای مختلفی که نیاز داریم استفاده کردیم.

توی کتابخونه scipy، از wavfile برای خوندن و نوشتن فایل‌های ویس‌ی استفاده شد. از fft برای انجام تبدیل فوریه استفاده شد که تبدیل سیگنال از حوزه زمان به حوزه فرکانس (و در صورت نیاز برعکس) رو ممکن می‌کنه. از signal هم برای عملیات‌های مرتبط با پردازش سیگنال (مثلا ریسمپلینگ) استفاده می‌شه.

برای عملیات‌های مربوط به ورودی گرفتن و ایجاد دایرکتوری و ... هم از os استفاده کردیم.



توی کد بالا اول مطمئن شدیم که دایرکتوری خروجی وجود داره و بعد هم در صورت عدم وجود ساختیمش.

تابع readvoice هم مطابق خواست سوال، از مسیر مشخص‌شده فایل رو می‌خونه و با کمک fft از scipy دامنه فرکانسی و فرکانس‌های متناظر با دامنه رو برمی‌گردونه. از تابع rfft برای تبدیل فوریه از دیتای حقیقی استفاده می‌شه و برای محاسبه فرکانس هم اندازه کل داده و معکوس ریت رو به rfftreq ورودی دادیم.

پیاده‌سازی تک‌تک فانکشن‌های مورد نیاز برای پردازش ویس:

```
change voice speed by changing the rate

def change_voice_speed(data, rate, speed_factor):
    speed_factor = np.sqrt(speed_factor)
    new_rate = int(rate * speed_factor)
    new_length = int(len(data) / speed_factor)
    new_data = sp_signal.resample(data, new_length)
    return new_data, new_rate
```

تابع تغییر سرعت رو پیاده کردیم که rate جدید برابر با ضربش در میزان افزایش سرعت می‌شه و برای اینکه داده جدید رو بسازیم هم با استفاده از متود resample که داخل signal پیاده‌سازی شده، داده‌ها رو بر اساس طول جدید مجددا نمونه‌برداری می‌کنیم که باعث می‌شه دیتامون فشرده/گسترده بشه.

```
low pass filter to remove noise from voice

def low_pass_filter(Frequency, Amplitude, cutoff):
    filtered_amplitude = Amplitude.copy()
    filtered_amplitude[Frequency > cutoff] = 0
    filtered_amplitude[np.abs(Amplitude) > 1e8] = 0
    return filtered_amplitude
```

توی فیلتر پایین‌گذر هم در ابتدا یه کپی از دامنه فرکانسی می‌گیریم و بر اساس cutoff، هر جا فرکانس از فرکانس قطع بیشتر بود اونجارو صفر می‌کنیم که نویزهای با فرکانس بالا رو حذف کنیم. همچنین چک می‌کنیم که اگر دامنه‌ها بسیار بزرگ بودن (خط سوم) که بخشی از نویزهامون هستن، اون‌ها رو هم صفر می‌کنیم. نهایتا دامنه‌ای که از فیلتر گذشته رو خروجی می‌دیم.

ادامه توضیحات پیاده‌سازی توابع رو صفحه بعد دنبال می‌کنیم:

reverse the voice to play it backwards

```
def reverse_voice(data, rate):
    return data[::-1], rate  # reverse the data
```

برای معکوس‌کردن هم کار پیچیده‌ای نداشتیم، کافی بود داده‌هامون رو با سینتکس ساده پایتون تو یه خط به صورت برعکس برگردونیم.

mix voices

```
def mix_voices(Datas, Rates):
    min_rate = min(Rates)
    resampled_datas = [sp_signal.resample(data, int(len(data) * min_rate / rate)) for data, rate in zip(Datas, Rates)]
    max_length = max(len(data) for data in resampled_datas)
    normalized_datas = [np.pad(data, (0, max_length - len(data)), 'constant') for data in resampled_datas]
    mixed_data = np.sum(normalized_datas, axis=0)
    mixed_data = mixed_data / len(Datas)  # Normalize to avoid clipping
    return mixed_data.astype(np.int16), min_rate
```

برای میکس‌کردن ویس‌ها اول کمترین ریت رو بین ریت‌های ویس‌ها گرفتیم چون بر اون اساس باید نرخ نمونه‌برداری رو تنظیم کنیم که همه ویس‌ها به درستی میکس بشن. بعد همه داده‌ها رو بر اساس ریتی که به دست آوردیم ریسمپل می‌کنیم که همه داده‌ها ریت یکسانی داشته باشن. توی خط بعد چک می‌کنیم که بیشترین طول بین ویس‌ها چقدره و بعدش همه ویس‌ها رو نرمالایز می‌کنیم، یعنی چی؟ مثلا اگه طول ویس slow که از همه طولانی تره ۶۲ ثانیه‌ست و باقی ویس‌ها ماکزیموم تو ۳۱ ثانیه تموم می‌شن باید به اندازه اون باقی ۳۱ ثانیه ۰ اضافه کنیم آخر ویس‌ها که توی میکس مشکلی به وجود نیاد. نهایتا با numpy داده‌هامون رو با هم جمع می‌زنیم و بعد نرمالایزکردنش بر اساس طول داده‌ها خروجی می‌دیم. یه سری تابع هم پیاده شدن که توضیح خاصی ندارن و فقط برای اینکه بتونیم خروجی مورد نظر سوال رو پلات/ذخیره کنیم ایجاد شدن که همه رو اینجا و صفحه بعد قرار می‌دم و نهایتا روند ورودی/خروجی‌دادن رو چک می‌کنیم.

write the voice to a file

```
def write_voice(data, rate, path):
    wavfile.write(path, rate, data.astype(np.int16))
    return
```

plot and save the voice

```
def plot_voice(Frequency, Amplitude, title="Frequency Domain Representation", filename=None):
    plt.plot(Frequency, np.abs(Amplitude))
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.title(title)
    if filename:
        plt.savefig(filename)
    plt.close()
```

plot and save the spectrogram

```
def show_spectrogram(data, rate, title="Spectrogram", filename=None):
    plt.specgram(data, Fs=rate)
    plt.title(title)
    if filename:
        plt.savefig(filename)
    plt.close()
```

plot and save the waveform

```
def plot_waveform(data, rate, title="Waveform", filename=None):
    time = np.arange(len(data)) / rate
    plt.plot(time, data)
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.title(title)
    if filename:
        plt.savefig(filename)
    plt.close()
```

توابع کاملا مشخصه چیکار می‌کنن و برای اینکه کد شلوغ نشه یه جا پیاده‌شدن و از متپلات‌لیب استفاده می‌کنن برای نمایش اسپکتروگرام، شکل موج و نمودار دامنه فرکانسی.

```
# Original voice path
input_path = "audio/potc.wav"

# Cleaned voice path
cleanpotc = os.path.join(output_dir, "cleanpotc.wav")

# Read the voice file
rate, data, Amplitude, Frequency = read_voice(input_path)

# Save original voice plots
plot_voice(Frequency, Amplitude, title="Original Frequency Domain Representation", filename=os.path.join(output_dir, "original_Amplitude.pn
plot_waveform(data, rate, title="Original Waveform", filename=os.path.join(output_dir, "original_Data.png"))
show_spectrogram(data, rate, title="Original Spectrogram", filename=os.path.join(output_dir, "original_Spectrogram.png"))
```

توی مرحله اول نمودارهای خواسته‌شده توی صورت سوال از ویس اصلی رو ذخیره کردیم که می‌تونید داخل پیوست پروژه تمامشون مشاهده کنید.

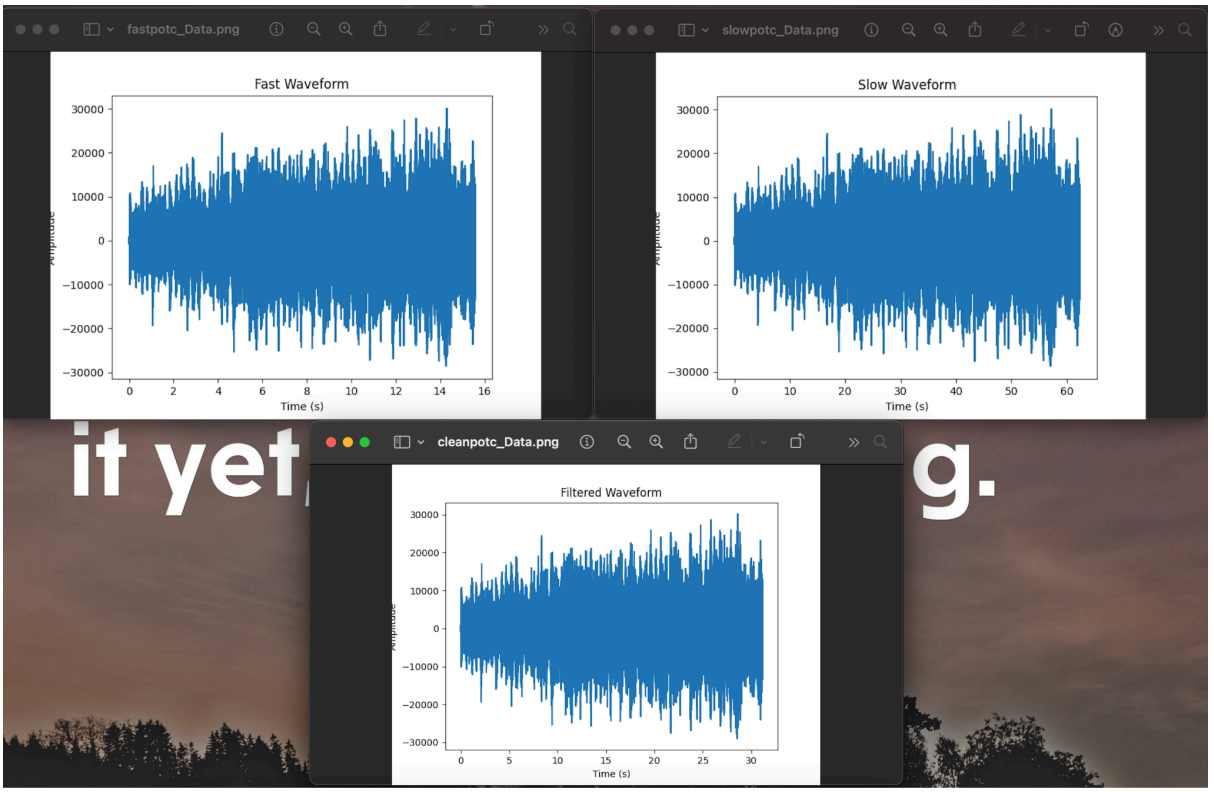
حذف نویزها:

```
# cutoff frequency for low pass filter
cutoff_frequency = 4000
# Apply low pass filter
filtered_Amplitude = low_pass_filter(Frequency, Amplitude, cutoff_frequency)

# Save filtered voice plots
plot_voice(Frequency, filtered_Amplitude, title="Filtered Frequency Domain Representation", filename=os.path.join(output_dir, "cleanpotc_Amp

# Inverse FFT to get back to time domain
filtered_data = sp_fft.irfft(filtered_Amplitude)

# Write the filtered voice to a new file
write_voice(filtered_data.astype(np.int16), rate, cleanpotc)

# Save cleaned voice plots
plot_waveform(filtered_data, rate, title="Filtered Waveform", filename=os.path.join(output_dir, "cleanpotc_Data.png"))
show_spectrogram(filtered_data, rate, title="Filtered Spectrogram", filename=os.path.join(output_dir, "cleanpotc_Spectrogram.png"))
```

توی این مرحله، بر اساس نموداری که توی مرحله قبل پلات کردیم بازه فرکانسی نویزها رو متوجه شدیم چون به طور واضحی نمودار رو به هم زده بود و بر اون اساس کاتاف رو تنظیم کردیم و فیلتر پایین‌گذر رو برای خالی از نویزکردن ویسمون پیاده کردیم. بعد از اعمال فیلتر هم مجددا نمودارهای مورد نیاز رو پلات کردیم و با فوریه ترانسفورم معکوس خروجی‌ای که تولید کردیم رو به تایم‌دامین برگردوندیم و با write voice داده رو داخل فایل نوشتیم.

تغییر سرعت ویس‌ها:

عکس کد این بخش از توضیحاتم رو توی صفحه بعد می‌تونید ببینید. اول اومدیم ویس‌ی که نویزش حذف شده رو به عنوان ورودی در نظر گرفتیم و بعدش سرعت رو به ترتیب ۲ برابر و ۱/۲ برابر کردیم و مجددا داخل فایل مربوطه نوشتیم. همونطور که داخل نمودارهای پلات‌شده از دیتای ویس‌ها مشخصه، دیتای هر سه مورد (بدون نویز، سریع و کند) کاملا یکسانه و فقط دامنه متفاوتی دارن. :)



Change the speed of the voice

```
# Consider the cleaned voice as the input voice
input_path = cleanpotc

# Speed changed voice path
fastpotc = os.path.join(output_dir, "fastpotc.wav") # 2x speed
slowpotc = os.path.join(output_dir, "slowpotc.wav") # 0.5x speed

# Read the voice file
rate, data, _, _ = read_voice(input_path)

# Change the speed of the voice
fast_data, fast_rate = change_voice_speed(data, rate, 2)
slow_data, slow_rate = change_voice_speed(data, rate, 0.5)

# Write the fast voice to a new file
write_voice(fast_data, fast_rate, fastpotc)

# Write the slow voice to a new file
write_voice(slow_data, slow_rate, slowpotc)

# Save fast voice plots
plot_waveform(fast_data, fast_rate, title="Fast Waveform", filename=os.path.join(output_dir, "fastpotc_Data.png"))
show_spectrogram(fast_data, fast_rate, title="Fast Spectrogram", filename=os.path.join(output_dir, "fastpotc_Spectrogram.png"))

# Save slow voice plots
plot_waveform(slow_data, slow_rate, title="Slow Waveform", filename=os.path.join(output_dir, "slowpotc_Data.png"))
show_spectrogram(slow_data, slow_rate, title="Slow Spectrogram", filename=os.path.join(output_dir, "slowpotc_Spectrogram.png"))
```

معکوس‌کردن ویس:

Reverse the voice

```
# Consider the cleaned voice as the input voice
input_path = cleanpotc

# Reversed voice path
revpotc = os.path.join(output_dir, "revpotc.wav")

# Read the voice file
rate, data, _, _ = read_voice(input_path)

# Reverse the voice
reversed_data, reversed_rate = reverse_voice(data, rate)

# Write the reversed voice to a new file
write_voice(reversed_data, reversed_rate, revpotc)

# Save reversed voice plots
plot_waveform(reversed_data, rate, title="Reversed Waveform", filename=os.path.join(output_dir, "revpotc_Data.png"))
show_spectrogram(reversed_data, rate, title="Reversed Spectrogram", filename=os.path.join(output_dir, "revpotc_Spectrogram.png"))
```

توی این مرحله هم ویس بدون نویز رو به عنوان ورودی به تابع ریورسمون دادیم که فقط دیتا رو برعکس می‌کرد و دیتای خروجی رو هم توی پیوست پروژه می‌تونید ببینید که دقیقا برعکس دیتای بدون نویزه.

میکس ویس‌ها:

توضیحات صفحه بعد

Mix voices

```
# inputs are all the voices in the previous steps
input_paths = [cleanpotc, fastpotc, slowpotc, revpotc]

# Mixed voice path
mixpotc = os.path.join(output_dir, "mixpotc.wav")

DataArray = []
RateArray = []

for path in input_paths:
    rate, data, _, _ = read_voice(path)
    DataArray.append(data)
    RateArray.append(rate)

# Mix the voices
mixed_data, mixed_rate = mix_voices(DataArray, RateArray)

# Write the mixed voice to a new file
write_voice(mixed_data, mixed_rate, mixpotc)

# Save mixed voice plots
plot_waveform(mixed_data, mixed_rate, title="Mixed Waveform", filename=os.path.join(output_dir, "mixpotc_Data.png"))
show_spectrogram(mixed_data, mixed_rate, title="Mixed Spectrogram", filename=os.path.join(output_dir, "mixpotc_Spectrogram.png"))

mixed_Amplitude = sp_fft.rfft(mixed_data)
mixed_Frequency = sp_fft.rfftfreq(len(mixed_data), 1 / mixed_rate)
plot_voice(mixed_Frequency, mixed_Amplitude, title="Mixed Frequency Domain Representation", filename=os.path.join(output_dir, "mixpotc_Amplitude.png"))
```

اول دیتا و ریت همه ویس‌هایی که تا الان ایجاد کرده بودیم رو داخل دو تا آرایه بر اساس ورودی که صورت سوال برای mixvoices خواسته بود ذخیره کردیم و بعد به تابع میکس‌ویس که بالاتر توضیحش داده شد ورودی دادیم. خروجیش هم تلفیقی از دیتای تمام ویس‌های قبلی شد: