توضیحات پیاده سازی مقاله درس بینایی ماشین

عنوان مقاله:

Multimodal Neural Network for Recognition of Cardiac Arrhythmias Based on 12-Load Electrocardiogram Signal

عنوان مقاله به فارسی:

شبکه عصبی چندوجهی برای تشخیص آریتمی های قلبی بر اساس سیگنال های الکتروکاردیوگرام ۱۲ لید قلبی

نگارنده:

مبینا حسینی مقدم

شماره دانشجویی:

40214140111001

بهار 1403

فصل1:

1-1:دیتاست ها و داده های مقاله:

دیتاست های این مقاله به طورعمومی در دسترس نبود.

به این دلیل که در رفرنس های این مقاله از چندین مقاله متفاوت برای پژوهش و دیتاسازی استفاده شده است, به همین دلیل یک دیتاست کامل برای این مقاله تنظیم نشده است.

به همین دلیل من از چندین دیتا ست متفاوت برای بررسی و پیاده سازی این مقاله استفاده کردم.

1-2:رفرنس ها یا مراجع

با توجه به رفرنس های مقاله دیتاهایی که استفاده شده ترکیبی از چند مقاله است.

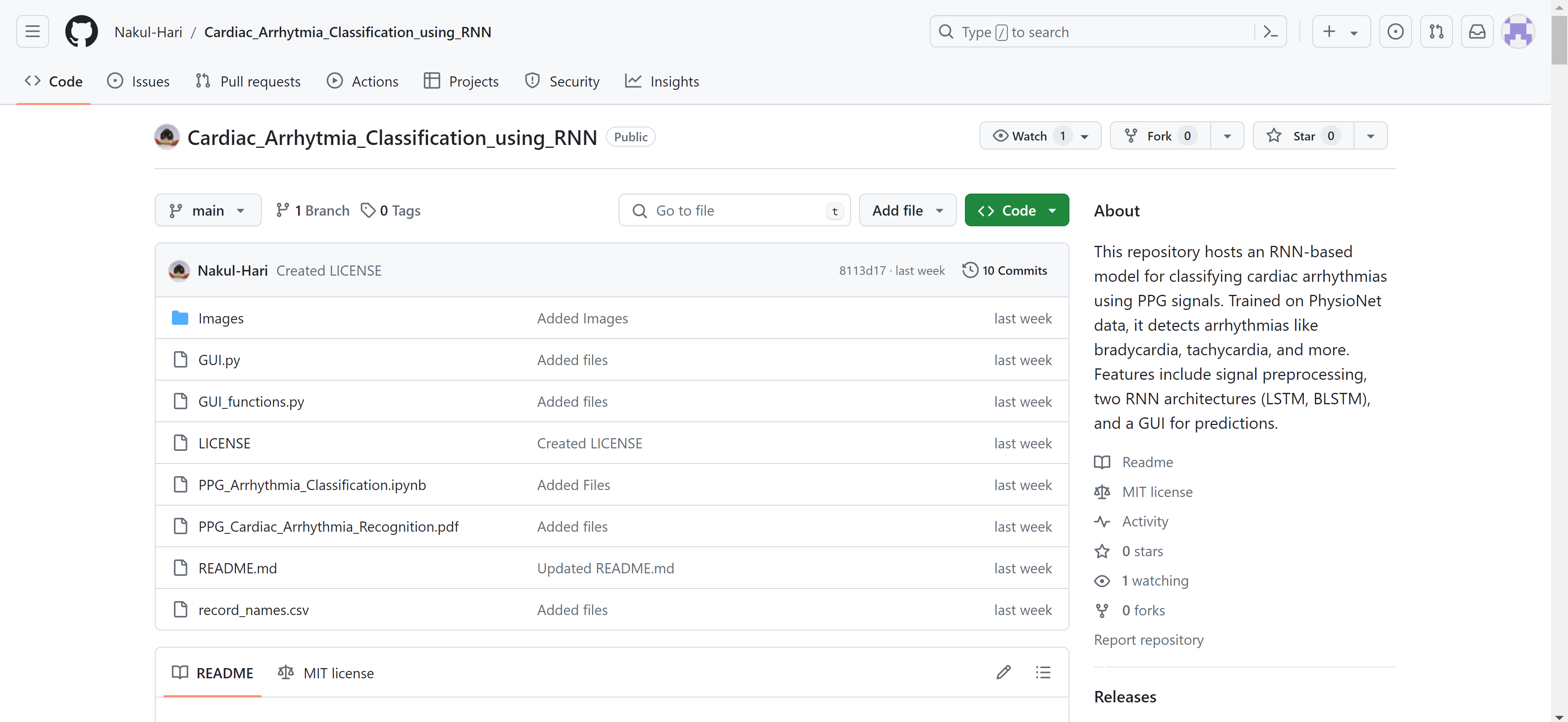
بنابراین دیتاست مورد استفاده به شکل یکسان مورد استفاده نیست.

1-3:روش جست و جوی دیتاست ها در گیت هاب:

1\_ابتدا کلمات متناسب با مقاله و عنوان مقاله در گیت هاب سرچ شد.

2\_در مرحله دوم کلمات کلیدی مقاله در گیت هاب سرچ شد.

3\_موضوعی که مرتبط با مقاله بودند انتخاب شد که به شرح عکس زیر است:



شکل1\_1: کد نویسی مرتبط اول.

1-3-1:کد نویسی قسمت اول:

**Part A: Installing Packages and Basic Visualization of PPG**

**A1: Installing Packages**

A بخش: نصب بسته ها و تجسم اولیه PPG

A۱: نصب بسته ها

# Importing packages

**import** tensorflow **as** tf

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** LSTM, Bidirectional, Dense

**import** os

**import** scipy.io

**from** scipy.signal **import** butter, filtfilt, find\_peaks

#from google.colab import drive

**import** datetime

**import** wfdb

**import** requests

**import** pywt

**import** seaborn

**import** keras

**import** random

**import** numpy **as** np

**import** csv

**import** tensorflow **as** tf

**import** matplotlib.pyplot **as** plt

**from** sklearn.metrics **import** confusion\_matrix

**from** os.path **import** join **as** osj

**import** pandas **as** pd

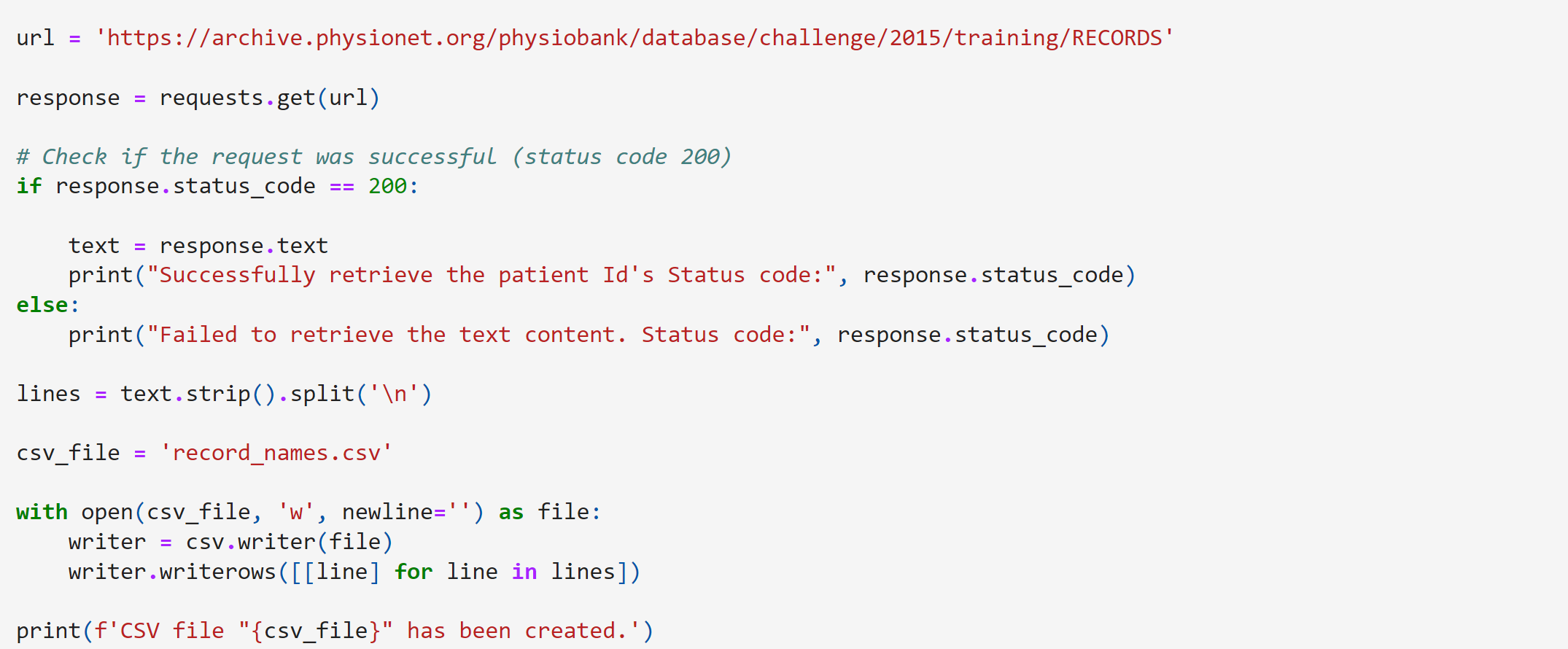


**a. Getting Recordings' IDs**

The PPG recordings are named after Patients' IDs , We'll get the Id from the physionet website records section and store it in a csv file

آ. دریافت شناسه های ضبط شده

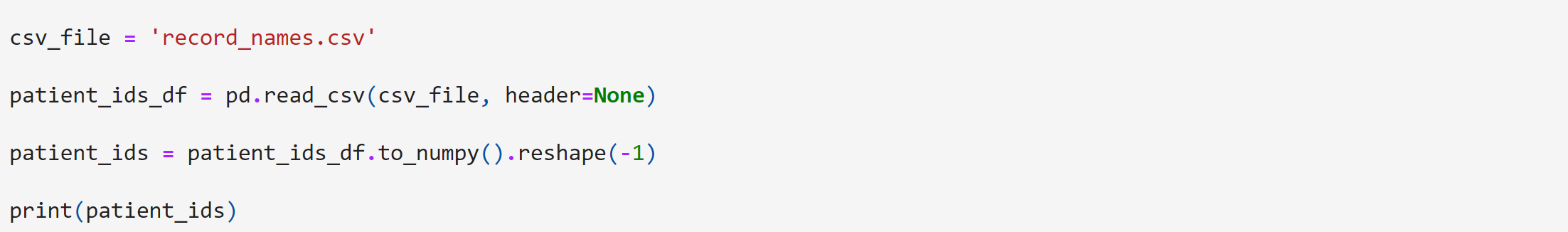
ضبط‌های PPG بر اساس شناسه‌های بیماران نامگذاری شده‌اند، ما شناسه را از بخش سوابق وب‌سایت physionet دریافت می‌کنیم و آن را در یک فایل csv ذخیره می‌کنیم.



Successfully retrieve the patient Id's Status code: 200

CSV file "record\_names.csv" has been created.

We'll read the csv file and then store the records Id in the array for easy access



['v100s' 'v101l' 'v102s' 'a103l' 'a104s' 'a105l' 't106s' 't107l' 't108s'

'a109l' 't110s' 'v111l' 't112s' 'v113l' 't114s' 'v115l' 't116s' 't117l'

't118s' 'v119l' 'f120s' 'f121l' 'v122s' 'a123l' 'b124s' 'b125l' 'b126s'

'v127l' 'v128s' 'f129l' 'f130s' 'v131l' 'v132s' 'v133l' 'a134s' 'v135l'

'v136s' 'f137l' 'f138s' 'v139l' 'v140s' 'v141l' 'a142s' 'v143l' 'f144s'

'a145l' 'v146s' 'v147l' 'v148s' 't149l' 't150s' 't151l' 'a152s' 'v153l'

'v154s' 'v155l' 't156s' 't157l' 'v158s' 'v159l' 'v160s' 'a161l' 'v162s'

'a163l' 'v164s' 'a165l' 'v166s' 'a167l' 'v168s' 'v169l' 'a170s' 'a171l'

'a172s' 't173l' 't174s' 't175l' 'v176s' 'v177l' 'a178s' 'v179l' 'v180s'

'v181l' 'v182s' 'b183l' 'b184s' 'a185l' 'a186s' 'b187l' 'v188s' 'f189l'

'f190s' 't191l' 't192s' 't193l' 'v194s' 't195l' 'f196s' 'v197l' 'v198s'

'v199l' 'v200s' 'v201l' 'v202s' 'a203l' 'v204s' 'v205l' 'v206s' 'v207l'

't208s' 't209l' 'v210s' 'v211l' 'v212s' 't213l' 't214s' 'b215l' 'b216s'

'v217l' 'v218s' 'a219l' 'b220s' 'v221l' 'v222s' 'a223l' 'v224s' 'a225l'

'a226s' 'b227l' 'b228s' 'b229l' 'v230s' 'b231l' 'v232s' 'v233l' 't234s'

't235l' 'f236s' 'f237l' 't238s' 'a239l' 't240s' 'v241l' 'v242s' 'v243l'

'v244s' 'v245l' 'v246s' 'v247l' 'v248s' 't249l' 'v250s' 't251l' 't252s'

'v253l' 'v254s' 'v255l' 'v256s' 'v257l' 'v258s' 'v259l' 'f260s' 'f261l'

'v262s' 't263l' 't264s' 'b265l' 'a266s' 'a267l' 'b268s' 'b269l' 't270s'

'v271l' 'a272s' 'a273l' 'v274s' 'v275l' 't276s' 't277l' 'a278s' 'a279l'

'v280s' 'f281l' 'v282s' 'v283l' 't284s' 'b285l' 'b286s' 'a287l' 'a288s'

'v289l' 'v290s' 'v291l' 'v292s' 'v293l' 'v294s' 'v295l' 'v296s' 'a297l'

'v298s' 'b299l' 't300s' 'a301l' 'a302s' 'v303l' 'f304s' 't305l' 'a306s'

'v307l' 'b308s' 'v309l' 'a310s' 'a311l' 'v312s' 'b313l' 'b314s' 'a315l'

'v316s' 'v317l' 'v318s' 'v319l' 't320s' 'f321l' 'v322s' 'v323l' 'v324s'

'v325l' 'v326s' 'v327l' 'v328s' 'v329l' 'b330s' 'b331l' 'b332s' 't333l'

'v334s' 't335l' 'v336s' 'v337l' 'v338s' 'b339l' 'b340s' 'b341l' 't342s'

't343l' 't344s' 'a345l' 'f346s' 'v347l' 'v348s' 'b349l' 't350s' 't351l'

'f352s' 'v353l' 'v354s' 'v355l' 't356s' 't357l' 't358s' 'v359l' 'v360s'

'v361l' 'f362s' 'a363l' 'v364s' 'v365l' 'v366s' 'v367l' 'v368s' 'v369l'

'v370s' 'v371l' 'a372s' 'v373l' 'v374s' 'v375l' 'a376s' 'a377l' 'a378s'

'b379l' 'v380s' 'v381l' 'a382s' 't383l' 't384s' 'a385l' 'a386s' 'b387l'

'b388s' 'b389l' 'v390s' 'a391l' 'v392s' 't393l' 't394s' 'v395l' 'a396s'

'a397l' 'v398s' 'v399l' 'v400s' 'v401l' 'v402s' 'v403l' 'v404s' 'v405l'

't406s' 'f407l' 'f408s' 't409l' 't410s' 't411l' 't412s' 't413l' 'f414s'

'f415l' 't416s' 't417l' 't418s' 'v419l' 'a420s' 'v421l' 'a422s' 'v423l'

't424s' 't425l' 'v426s' 'v427l' 'b428s' 'a429l' 't430s' 'v431l' 'v432s'

'v433l' 't434s' 'a435l' 'a436s' 'v437l' 'v438s' 'a439l' 'f440s' 'f441l'

'a442s' 'a443l' 't444s' 't445l' 'a446s' 't447l' 'v448s' 'a449l' 'f450s'

'b451l' 'v452s' 'v453l' 'v454s' 'b455l' 'b456s' 'a457l' 't458s' 'v459l'

'v460s' 'a461l' 'a462s' 'v463l' 'v464s' 'a465l' 'v466s' 't467l' 't468s'

't469l' 'v470s' 'v471l' 'v472s' 'v473l' 'f474s' 'v475l' 'v476s' 't477l'

't478s' 'v479l' 'v480s' 'v481l' 'v482s' 'v483l' 'b484s' 'b485l' 'b486s'

'b487l' 'b488s' 'v489l' 'a490s' 'v491l' 'v492s' 'f493l' 'b494s' 'b495l'

't496s' 'b497l' 'v498s' 'f499l' 'f500s' 'v501l' 'v502s' 't503l' 't504s'

'v505l' 't506s' 't507l' 't508s' 't509l' 'v510s' 'v511l' 'a512s' 'v513l'

'a514s' 'b515l' 'b516s' 'b517l' 'v518s' 'v519l' 't520s' 't521l' 'v522s'

'v523l' 't524s' 'v525l' 'a526s' 'a527l' 'b528s' 'f529l' 'f530s' 'v531l'

'v532s' 'v533l' 'v534s' 'v535l' 'v536s' 'b537l' 'b538s' 'a539l' 'v540s'

'v541l' 'v542s' 'f543l' 'f544s' 'f545l' 't546s' 't547l' 'v548s' 'v549l'

'a550s' 'v551l' 'v552s' 'b553l' 'b554s' 'a555l' 'a556s' 'v557l' 'a558s'

'v559l' 'b560s' 'b561l' 'b562s' 'f563l' 'v564s' 't565l' 'v566s' 't567l'

'v568s' 'v569l' 'v570s' 'v571l' 'f572s' 'v573l' 'v574s' 'v575l' 'f576s'

't577l' 'b578s' 'v579l' 't580s' 'v581l' 'a582s' 'v583l' 'a584s' 'v585l'

'f586s' 'b587l' 'b588s' 't589l' 'v590s' 'a591l' 'f592s' 'f593l' 't594s'

't595l' 'v596s' 'v597l' 'v598s' 'a599l' 'b600s' 'v601l' 'f602s' 'a603l'

'a604s' 'f605l' 'a606s' 'v607l' 'a608s' 'v609l' 'f610s' 'v611l' 'v612s'

'f613l' 't614s' 'v615l' 'v616s' 'b617l' 'f618s' 'v619l' 'v620s' 'v621l'

't622s' 'v623l' 'a624s' 'v625l' 'v626s' 'v627l' 'v628s' 'v629l' 'v630s'

'a631l' 'v632s' 'v633l' 'v634s' 'v635l' 'v636s' 'f637l' 'v638s' 'a639l'

'v640s' 'v641l' 'f642s' 'v643l' 'v644s' 'a645l' 'v646s' 'v647l' 'v648s'

'v649l' 'a650s' 'a651l' 'v652s' 'a653l' 'a654s' 'v655l' 'b656s' 'f657l'

'v658s' 'b659l' 'v660s' 'a661l' 't662s' 'v663l' 'b664s' 't665l' 'v666s'

'a667l' 'a668s' 'b669l' 'a670s' 'v671l' 'b672s' 'a673l' 'v674s' 'a675l'

'v676s' 't677l' 't678s' 't679l' 't680s' 'b681l' 'v682s' 't683l' 'b684s'

'b685l' 'v686s' 'v687l' 't688s' 't689l' 't690s' 'f691l' 'v692s' 't693l'

'a694s' 'b695l' 'v696s' 'f697l' 't698s' 'a699l' 't700s' 'v701l' 't702s'

'b703l' 'v704s' 'a705l' 'b706s' 't707l' 'b708s' 't709l' 'v710s' 'v711l'

'a712s' 'v713l' 'v714s' 'a715l' 't716s' 't717l' 'v718s' 't719l' 'v720s'

'v721l' 'b722s' 'a723l' 'v724s' 'v725l' 'v726s' 'v727l' 'v728s' 'v729l'

'b730s' 't731l' 'v732s' 'v733l' 'b734s' 'a735l' 'v736s' 't737l' 'v738s'

't739l' 'a740s' 't741l' 't742s' 'v743l' 't744s' 't745l' 'a746s' 't747l'

'v748s' 'v749l' 'a750s' 'f751l' 't752s' 'b753l' 'a754s' 't755l' 'v756s'

'b757l' 'v758s' 'v759l' 't760s' 'v761l' 't762s' 'v763l' 'b764s' 'v765l'

'v766s' 'v767l' 'f768s' 'v769l' 'v770s' 't771l' 'v772s' 'v773l' 'v774s'

'v775l' 'a776s' 't777l' 'a778s' 'v779l' 'a780s' 'v781l' 'v782s' 'v783l'

'v784s' 'a785l' 't786s' 't787l' 'v788s' 'f789l' 't790s' 'v791l' 'f792s'

'v793l' 'b794s' 'v795l' 'a796s' 'v797l' 'a798s' 'f799l' 't800s' 't801l'

'a802s' 'v803l' 'v804s' 'v805l' 'v806s' 'a807l' 'v808s' 'v809l' 'a810s'

'v811l' 't812s' 'v813l' 'v814s' 'v815l' 't816s' 't817l' 'v818s' 'a819l'

'b820s' 't821l' 'a822s' 'v823l' 'b824s' 'a825l' 'v826s' 'v827l' 'v828s'

'f829l' 'v830s' 'v831l' 'b832s' 'v833l' 'v834s' 'b835l' 'v836s' 'v837l'

'b838s' 'b839l' 'b840s' 'b841l' 'v842s' 'v843l' 'v844s' 'v845l' 'v846s'

'a847l' 'v848s' 'b849l']

**b. Patient PPG signal and info extracting**

Downloading the content fromt he pysionet website

ب سیگنال PPG بیمار و استخراج اطلاعات

دانلود مطالب از وب سایت physionet





{'file\_name': 'a103l',

'num\_signals': 3,

'frequency': 250,

'num\_samples': 82500,

'ADC\_resolution': '16+24',

'ADC\_gain': '1.253e+04/NU',

'ADC\_bits': 16,

'ADC\_zero': 0,

'initial\_value': 6042,

'check\_sum': -17391,

'block\_size': 0,

'description': 'PLETH',

'electrical\_activity': 'Asystole',

'validity': 'False alarm'}



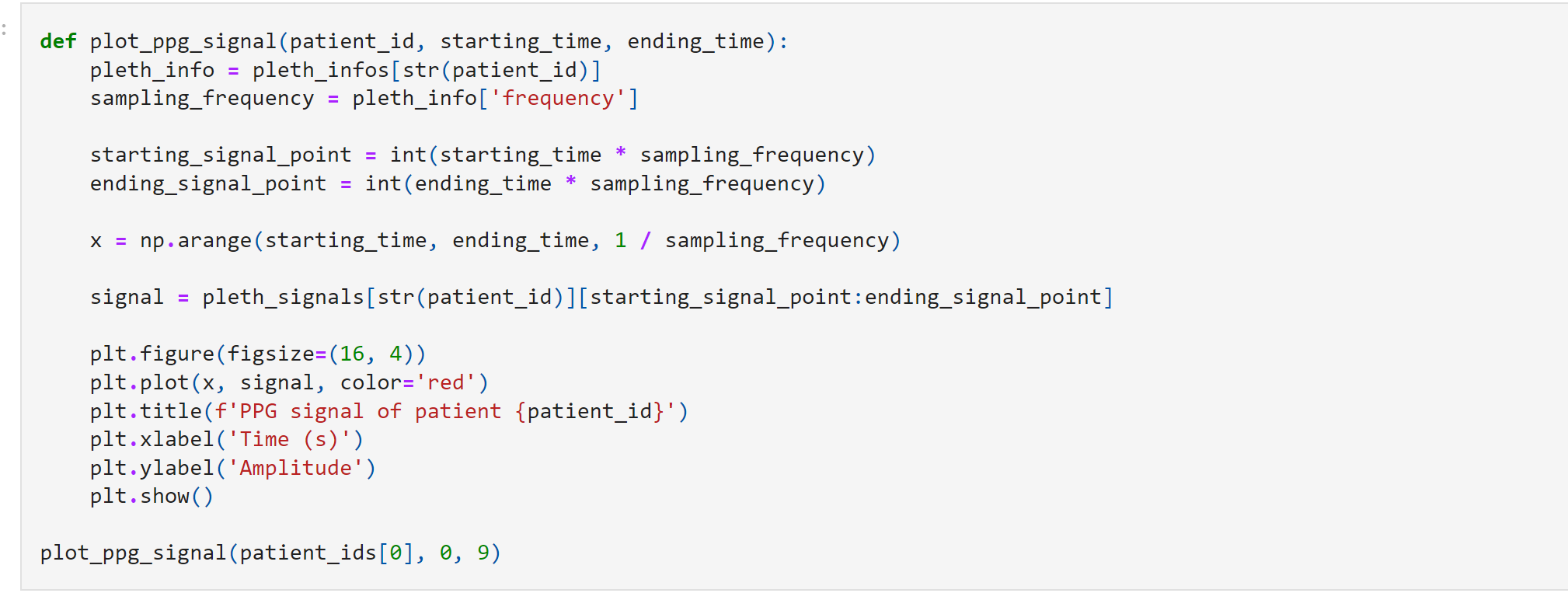
**c. All patients' PPG loading**

ج. بارگیری PPG همه بیماران



در این قسمت بارگیری بیماران از تصاویر انجام شد اما به دلیل زیاد و طولانی بودن کد های دستوری در این گزارش گنجانده نشده است.





**Part B: Pre-Processing the Signal**

**B1: Bandpass Filter**

Filtration of the signal utilizing a bandpass filter between (0.05 Hz to 30 Hz).

Args:

* signal: The input signal.
* lowcut: The low cutoff frequency of the bandpass filter.
* highcut: The high cutoff frequency of the bandpass filter.
* fs: The sampling frequency of the signal.

Returns:

* filtered\_signal: The filtered signal.

بخش B: پیش پردازش سیگنال

B۱: فیلتر باند گذر

فیلتر کردن سیگنال با استفاده از فیلتر گذر باند بین (۰.۰۵ هرتز تا ۳۰ هرتز).

ارگ:

سیگنال: سیگنال ورودی.

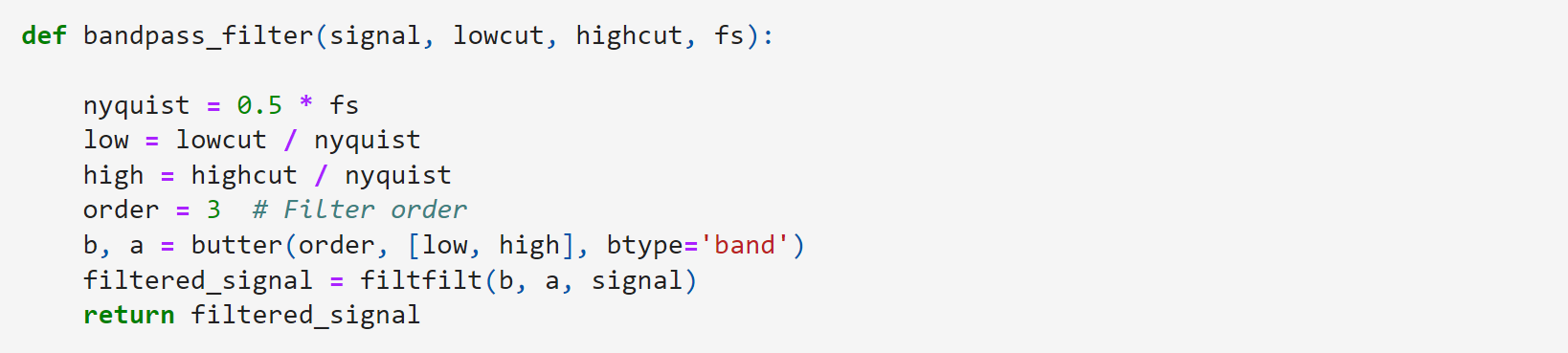
Lowcut: فرکانس قطع پایین فیلتر باند گذر.

highcut: فرکانس قطع بالای فیلتر باند گذر.

fs: فرکانس نمونه برداری سیگنال.

برمی‌گرداند:

filtered\_signal سیگنال فیلتر شده.



**B2: Moving Average Filter**

It works by averaging a window of adjacent samples in the signal, shifting the window along the signal, and replacing each sample with the average value of the samples within the window

Args:

* signal: The input signal.
* window\_size: The size of the moving average window.

Returns:

* smoothed\_signal: The smoothed signal.

B۲: فیلتر میانگین متحرک

با میانگین گیری پنجره ای از نمونه های مجاور در سیگنال، جابجایی پنجره در امتداد سیگنال، و جایگزینی هر نمونه با مقدار متوسط ​​نمونه های داخل پنجره کار می کند.

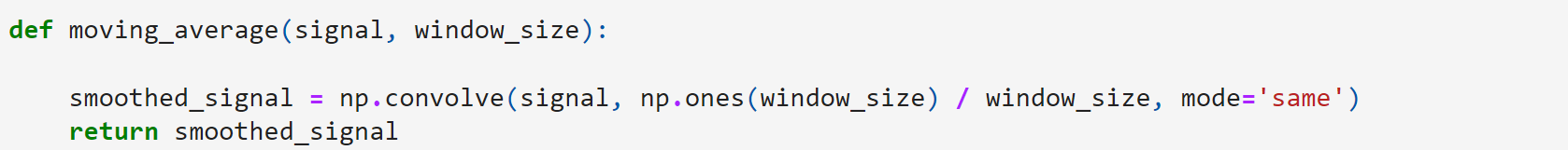
ارگ:

سیگنال: سیگنال ورودی.

window\_sizeاندازه پنجره میانگین متحرک.

برمی‌گرداند:

smoothed\_signalسیگنال صاف شده.



**B3: Baseline Wandering Removal**

Baseline wandering refers to the low-frequency fluctuation or drift observed in physiological signals, such as electrocardiogram (ECG) or photoplethysmogram (PPG), caused by various factors including respiration, body movement, electrode or sensor contact issues, and muscle activity. It appears as a slow, periodic variation superimposed on the actual signal of interest.

Baseline wandering removal is the process of eliminating or reducing this low-frequency fluctuation from the signal while preserving the underlying information.

Args:

* signal: The input signal.

Returns:

* denoised\_signal: The denoised signal.

B۳: حذف سرگردان در خط پایه

سرگردانی پایه به نوسان یا رانش کم فرکانس مشاهده شده در سیگنال های فیزیولوژیکی، مانند الکتروکاردیوگرام (ECG) یا فوتوپلتیسموگرام (PPG) اشاره دارد که توسط عوامل مختلفی از جمله تنفس، حرکت بدن، مشکلات تماس الکترود یا حسگر و فعالیت عضلانی ایجاد می شود. به‌عنوان یک تغییرات آهسته و دوره‌ای که بر سیگنال واقعی مورد علاقه قرار می‌گیرد، ظاهر می‌شود.

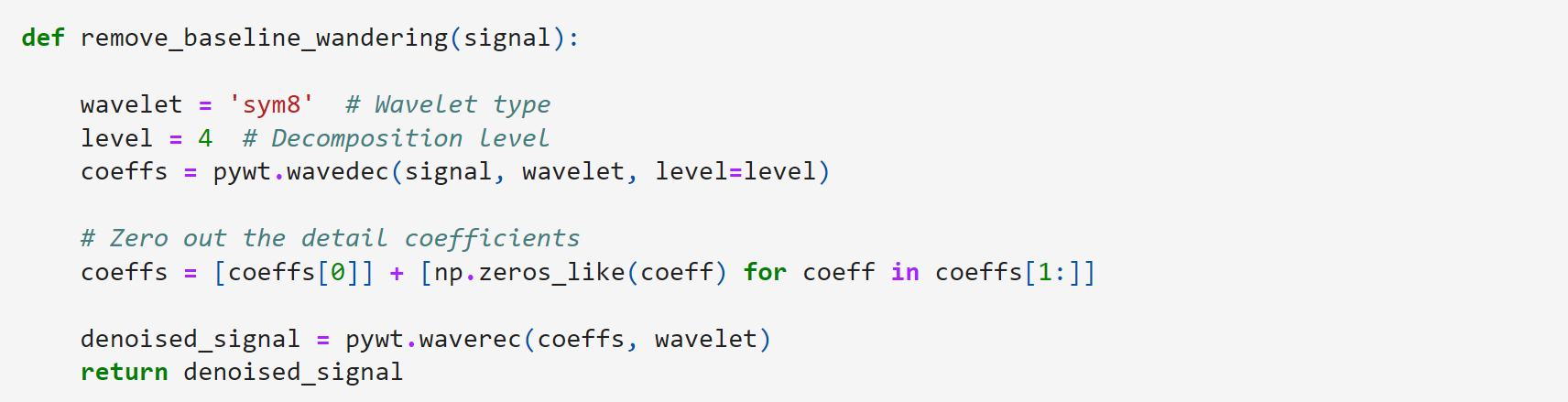
حذف سرگردانی در خط پایه فرآیند حذف یا کاهش این نوسانات فرکانس پایین از سیگنال با حفظ اطلاعات اساسی است.

ارگ:

سیگنال: سیگنال ورودی.

برمی‌گرداند:

denoised\_signalسیگنال حذف شده.



**B4: Normalisation**

The goal of normalization is to bring the data within a similar scale, typically between 0 and 1 or -1 and 1, to ensure that all features contribute equally to the analysis and to prevent certain features from dominating others due to differences in their scales.

Args:

* signal: The input signal.

Returns:

* normalized\_signal: The normalized signal.

B۴: عادی سازی

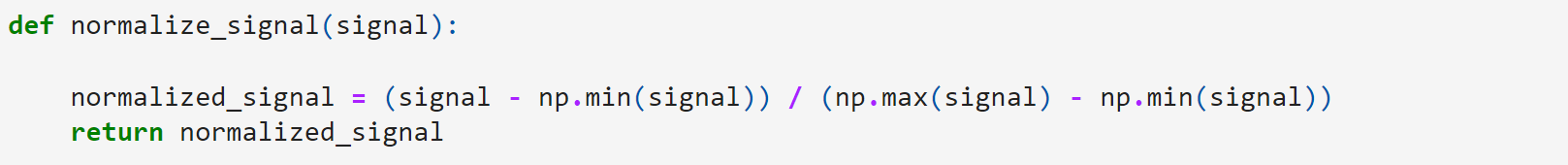
هدف نرمال سازی، آوردن داده ها در یک مقیاس مشابه، معمولاً بین ۰ و ۱ یا ۱- و ۱ است تا اطمینان حاصل شود که همه ویژگی ها به طور مساوی در تجزیه و تحلیل نقش دارند و از تسلط برخی ویژگی ها به سایر ویژگی ها به دلیل تفاوت در مقیاس آنها جلوگیری می کند.

ارگ:

سیگنال: سیگنال ورودی.

برمی‌گرداند:

normalized\_signal سیگنال نرمال شده.



**B5: Segmentation**

Each waveform was segmented into 10-s

Args:

* signal: The input signal.
* segment\_length: The length of each segment in seconds.
* fs: The sampling frequency of the signal.

Returns:

* segmented\_signal: A list of segmented signals.

B۵: تقسیم بندی

هر شکل موج به ۱۰ ثانیه تقسیم شد

ارگ:

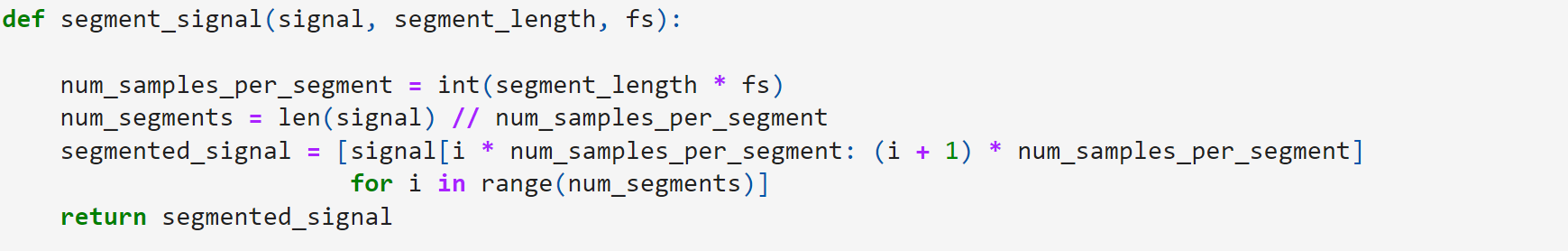
سیگنال: سیگنال ورودی.

segment\_length طول هر بخش بر حسب ثانیه.

fs: فرکانس نمونه برداری سیگنال.

برمی‌گرداند:

segmented\_signalفهرستی از سیگنال‌های تقسیم‌بندی شده.



* Firstly, Each waveform was segmented into 10-s
* The preprocessing includes filtration of the signal utilizing a bandpass filter between (0.05 Hz to 30 Hz).
* The signal smoothing using moving average filter
* Then, the baseline wandering was removed using wavelet transform
* Followed with signal normalization.

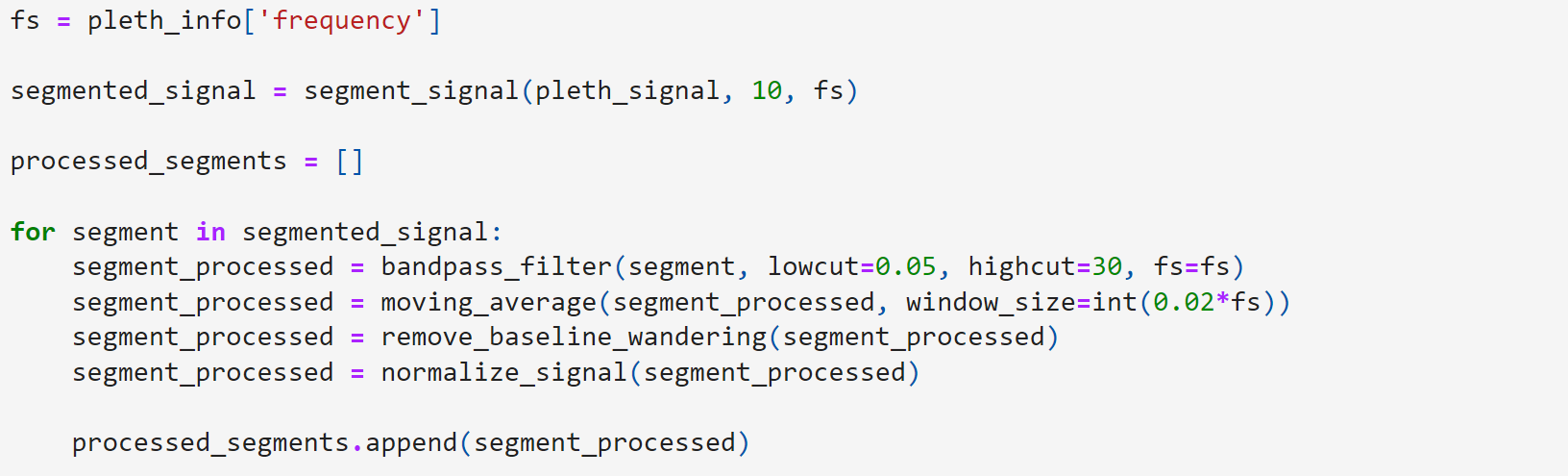
ابتدا هر شکل موج به ۱۰ ثانیه تقسیم شد

پیش پردازش شامل فیلتر کردن سیگنال با استفاده از فیلتر باند گذر بین (۰.۰۵ هرتز تا ۳۰ هرتز) است.

صاف کردن سیگنال با استفاده از فیلتر میانگین متحرک

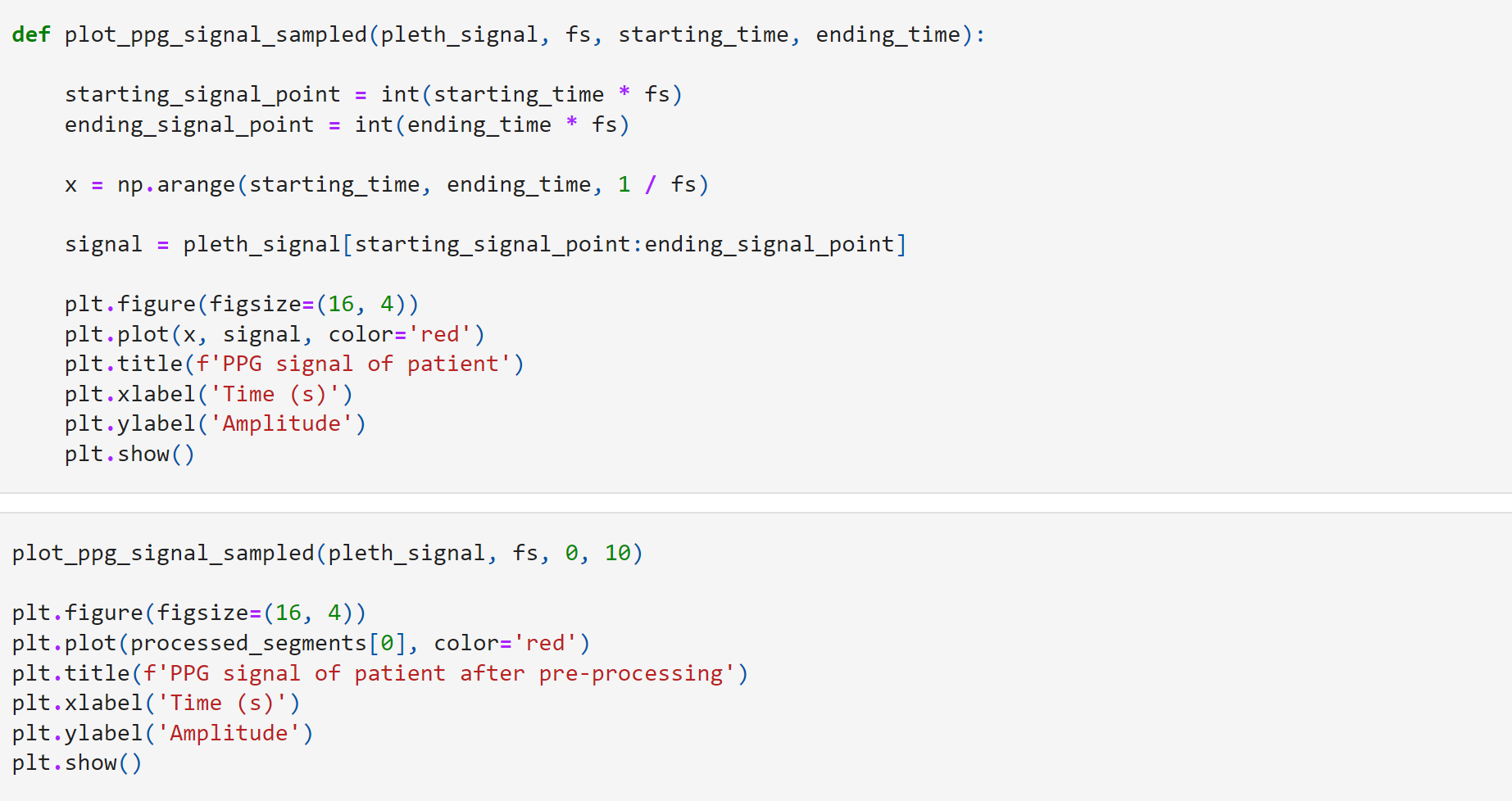
سپس، سرگردانی خط پایه با استفاده از تبدیل موجک حذف شد

به دنبال عادی سازی سیگنال.



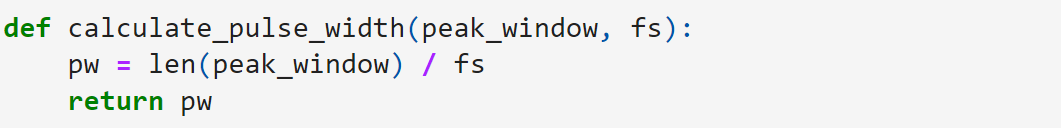
Plotting the Signal Before and After processing

ترسیم سیگنال قبل و بعد از پردازش



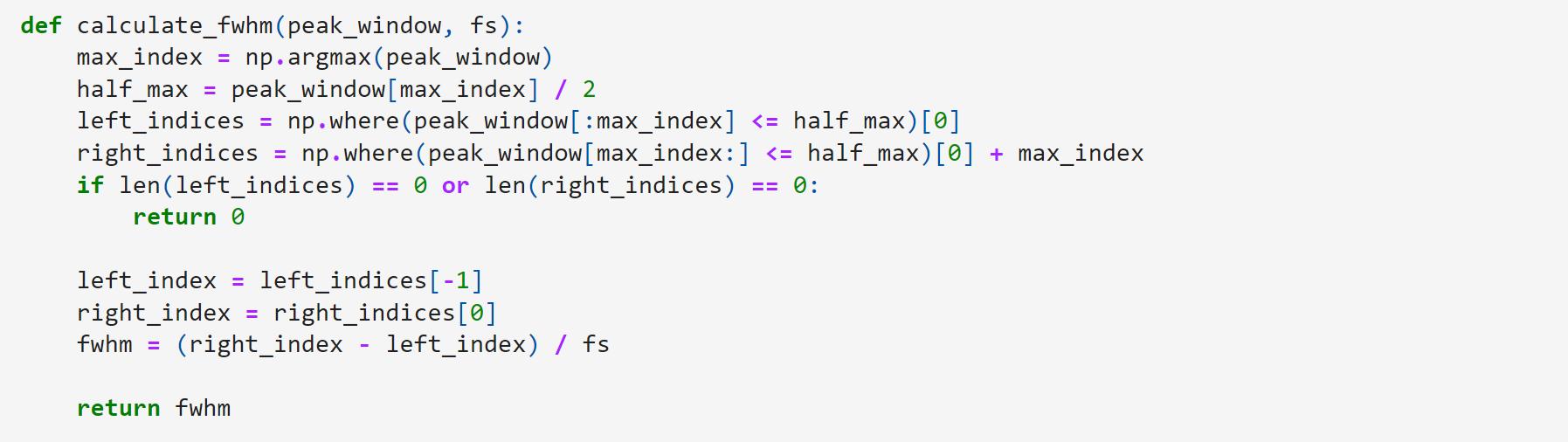
**Pulse Width Extraction**

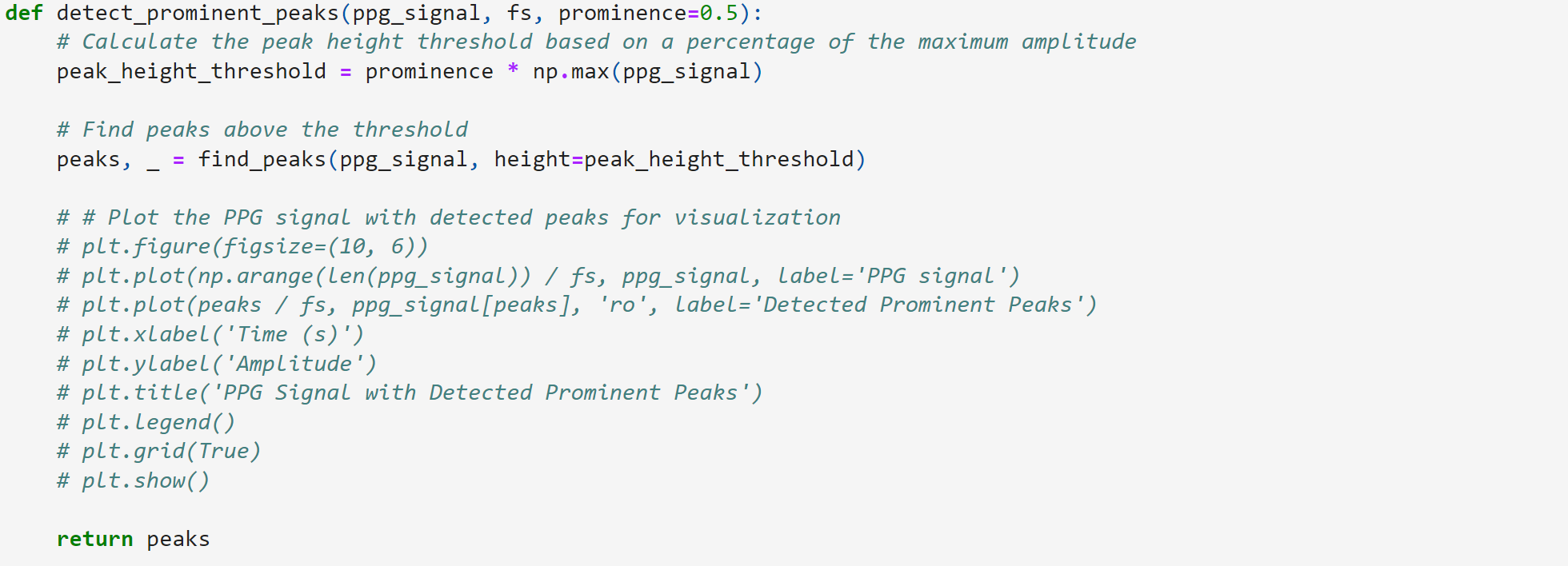
استخراج عرض پالس



**Full width at half maximum Extraction**

عرض کامل در نصف حداکثر استخراج





**import** numpy **as** np

**from** scipy.signal **import** find\_peaks

**from** scipy.integrate **import** simps

**from** scipy.fft **import** fft

**from** scipy **import** stats

**def** extract\_ppg\_features(ppg\_signal, fs):

*# Peak Detection with higher threshold*

peak\_height\_threshold **=** 0.6 *# Might have to adjust accordingly*

peaks, \_ **=** find\_peaks(ppg\_signal, height**=**peak\_height\_threshold)

peak\_features **=** []

prev\_peak\_index **=** **None**

**for** peak\_index **in** peaks:

window\_size **=** int(fs **\*** 2) *# Example: 0.5 seconds window*

peak\_window **=** ppg\_signal[max(0, peak\_index **-** window\_size):min(len(ppg\_signal), peak\_index **+** window\_size)]

*# Morphological Features*

sa **=** np**.**max(peak\_window)

Da **=** np**.**min(peak\_window)

SA **=** np**.**trapz(peak\_window, dx**=**1**/**fs)

DA **=** np**.**trapz(ppg\_signal, dx**=**1**/**fs) **-** SA

St **=** peak\_index **/** fs

Dt **=** (len(ppg\_signal) **-** peak\_index) **/** fs

*# Frequency Domain Features*

fft\_result **=** fft(peak\_window)

magnitude\_spectrum **=** np**.**abs(fft\_result)

frequency\_spectrum **=** np**.**fft**.**fftfreq(len(peak\_window), d**=**1**/**fs)

dominant\_frequency **=** frequency\_spectrum[np**.**argmax(magnitude\_spectrum)]

spectral\_entropy **=** **-**np**.**sum(magnitude\_spectrum **\*** np**.**log2(magnitude\_spectrum))

*# Additional features*

**if** prev\_peak\_index **is** **not** **None**:

ppi **=** (peak\_index **-** prev\_peak\_index) **/** fs

**else**:

ppi **=** 0

pi **=** ppi

pw **=** calculate\_pulse\_width(peak\_window, fs)

fwhm **=** calculate\_fwhm(peak\_window, fs)

signal\_area **=** np**.**trapz(ppg\_signal, dx**=**1**/**fs)

rise\_time **=** St **-** (max(0, peak\_index **-** window\_size)) **/** fs

fall\_time **=** (min(len(ppg\_signal), peak\_index **+** window\_size) **-** peak\_index) **/** fs

amplitude\_modulation\_depth **=** sa **-** Da

energy **=** np**.**sum(peak\_window **\*\*** 2)

zero\_crossing\_rate **=** calculate\_zero\_crossing\_rate(peak\_window)

*# Statistical Features*

mean **=** np**.**mean(peak\_window)

median **=** np**.**median(peak\_window)

std\_deviation **=** np**.**std(peak\_window)

skewness **=** stats**.**skew(peak\_window)

kurtosis **=** stats**.**kurtosis(peak\_window)

min\_value **=** np**.**min(peak\_window)

max\_value **=** np**.**max(peak\_window)

variance **=** np**.**var(peak\_window)

*# Additional features*

slope **=** calculate\_slope(peak\_window, fs)

peak\_count **=** len(peaks)

*# Handle exceptions for divide by zero errors*

**try**:

amplitude\_ratio **=** sa **/** Da

area\_ratio **=** SA **/** DA

interval\_ratio **=** pi **/** ppi **if** ppi **!=** 0 **else** 0

**except** ZeroDivisionError:

amplitude\_ratio **=** 0

area\_ratio **=** 0

interval\_ratio **=** 0

*# Append all features to the list*

peak\_features**.**append({

'sa': sa, 'Da': Da, 'SA': SA, 'DA': DA, 'St': St, 'Dt': Dt,

'PI': pi, 'PPI': ppi, 'PW': pw, 'FWHM': fwhm,

'Dominant\_frequency': dominant\_frequency,

'Spectral\_entropy': spectral\_entropy,

'Signal\_area': signal\_area,

'Rise\_time': rise\_time,

'Fall\_time': fall\_time,

'Amplitude\_modulation\_depth': amplitude\_modulation\_depth,

'Energy': energy,

'Zero\_crossing\_rate': zero\_crossing\_rate,

'Mean': mean,

'Median': median,

'Standard\_deviation': std\_deviation,

'Skewness': skewness,

'Kurtosis': kurtosis,

'Min\_value': min\_value,

'Max\_value': max\_value,

'Variance': variance,

'Slope': slope,

'Peak\_count': peak\_count,

'Amplitude\_ratio': amplitude\_ratio,

'Area\_ratio': area\_ratio,

'Interval\_ratio': interval\_ratio

*# Add more features as needed*

})

prev\_peak\_index **=** peak\_index

**return** peak\_features, peaks

**def** calculate\_slope(signal, fs):

time **=** np**.**arange(0, len(signal)) **/** fs

slope, \_ **=** np**.**polyfit(time, signal, 1)

**return** slope

**def** calculate\_pulse\_width(peak\_window, fs):

*# Calculate pulse width as the width at half of the maximum amplitude*

half\_max\_amplitude **=** 0.5 **\*** np**.**max(peak\_window)

above\_half **=** peak\_window **>** half\_max\_amplitude

**return** fs **\*** np**.**sum(above\_half) **/** fs

**def** calculate\_fwhm(peak\_window, fs):

*# Calculate full width half maximum (FWHM)*

half\_max\_amplitude **=** 0.5 **\*** np**.**max(peak\_window)

peaks, \_ **=** find\_peaks(peak\_window, height**=**half\_max\_amplitude)

**if** len(peaks) **<** 2:

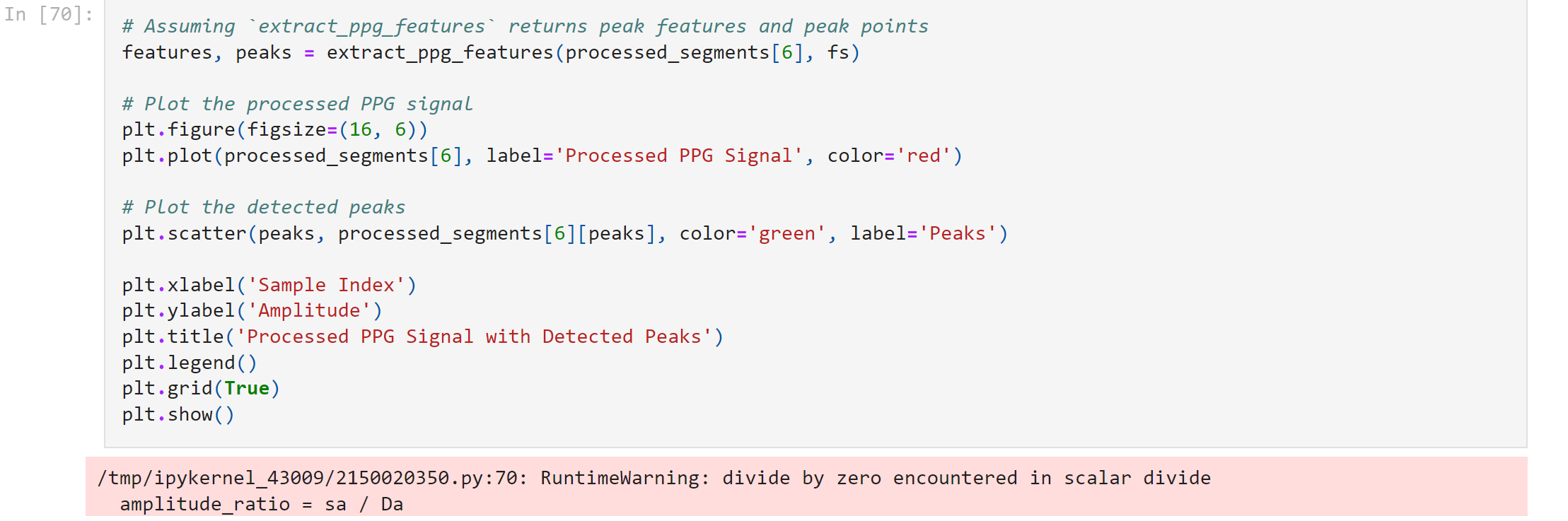
**return** 0

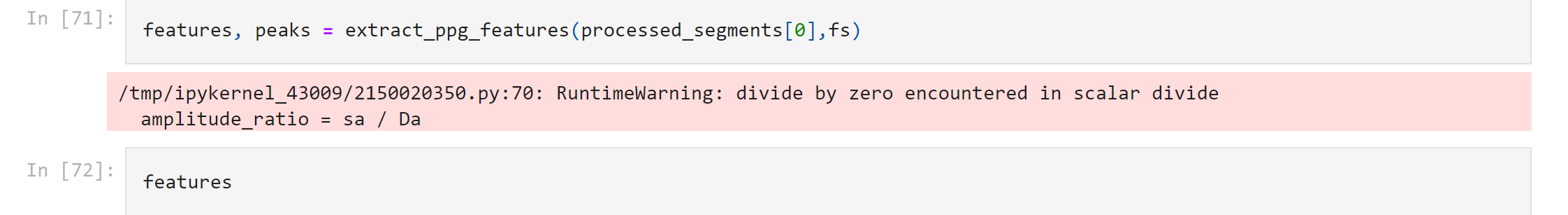
**return** (peaks[**-**1] **-** peaks[0]) **/** fs

**def** calculate\_zero\_crossing\_rate(signal):

*# Calculate zero crossing rate*

**return** np**.**sum(np**.**diff(np**.**sign(signal)) **!=** 0) **/** len(signal)





خروجی به شکل زیر است:

[{'sa': 0.9809083922120247,

'Da': 0.2538622235257515,

'SA': 1.4902409706653952,

'DA': 4.117623943672259,

'St': 0.316,

'Dt': 9.684,

'PI': 0,

'PPI': 0,

'PW': 432.0,

'FWHM': 1.956,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -5055.633095414664,

'Signal\_area': 5.607864914337654,

'Rise\_time': 0.316,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.7270461686862733,

'Energy': 264.41985044906573,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.644543900952092,

'Median': 0.6344146709971253,

'Standard\_deviation': 0.20309317880599792,

'Skewness': -0.05069995784665806,

'Kurtosis': -1.0898213727502677,

'Min\_value': 0.2538622235257515,

'Max\_value': 0.9809083922120247,

'Variance': 0.041246839277525045,

'Slope': 0.10055302139755666,

'Peak\_count': 15,

'Amplitude\_ratio': 3.8639399694398504,

'Area\_ratio': 0.36191769599443796,

'Interval\_ratio': 0},

{'sa': 0.9809083922120247,

'Da': 0.2538622235257515,

'SA': 1.9554015435249248,

'DA': 3.652463370812729,

'St': 0.976,

'Dt': 9.024,

'PI': 0.66,

'PPI': 0.66,

'PW': 597.0,

'FWHM': 2.612,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -6910.659436829061,

'Signal\_area': 5.607864914337654,

'Rise\_time': 0.976,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.7270461686862733,

'Energy': 349.88723733409415,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.6578964616531929,

'Median': 0.649209004426722,

'Standard\_deviation': 0.19352206734142446,

'Skewness': -0.10737214485225534,

'Kurtosis': -0.9737869723941666,

'Min\_value': 0.2538622235257515,

'Max\_value': 0.9809083922120247,

'Variance': 0.03745079054809882,

'Slope': 0.0696726221929586,

'Peak\_count': 15,

'Amplitude\_ratio': 3.8639399694398504,

'Area\_ratio': 0.5353651344324962,

'Interval\_ratio': 1.0},

{'sa': 0.9809083922120247,

'Da': 0.2538622235257515,

'SA': 2.412067861435483,

'DA': 3.195797052902171,

'St': 1.624,

'Dt': 8.376,

'PI': 0.648,

'PPI': 0.648,

'PW': 759.0,

'FWHM': 3.248,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -8770.586498132394,

'Signal\_area': 5.607864914337654,

'Rise\_time': 1.624,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.7270461686862733,

'Energy': 433.0483795416349,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.6662322899964052,

'Median': 0.6591121169061478,

'Standard\_deviation': 0.18469672339022442,

'Skewness': -0.15599137472472505,

'Kurtosis': -0.8574658281795156,

'Min\_value': 0.2538622235257515,

'Max\_value': 0.9809083922120247,

'Variance': 0.03411287963108507,

'Slope': 0.0504198176151689,

'Peak\_count': 15,

'Amplitude\_ratio': 3.8639399694398504,

'Area\_ratio': 0.7547625276282901,

'Interval\_ratio': 1.0},

{'sa': 0.9809083922120247,

'Da': 0.33218964958015423,

'SA': 2.6616543654389333,

'DA': 2.9462105488987205,

'St': 2.272,

'Dt': 7.728,

'PI': 0.648,

'PPI': 0.648,

'PW': 815.0,

'FWHM': 3.92,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9354.60239981084,

'Signal\_area': 5.607864914337654,

'Rise\_time': 1.9999999999999998,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.6487187426318706,

'Energy': 476.50315897689757,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.6662910342231478,

'Median': 0.6514831769043912,

'Standard\_deviation': 0.18044228077350852,

'Skewness': 0.033510678422975046,

'Kurtosis': -1.158357437611988,

'Min\_value': 0.33218964958015423,

'Max\_value': 0.9809083922120247,

'Variance': 0.03255941669074568,

'Slope': -0.00690773792490554,

'Peak\_count': 15,

'Amplitude\_ratio': 2.952856578920412,

'Area\_ratio': 0.9034162091483404,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.33884836344283636,

'SA': 2.6792995323675712,

'DA': 2.9285653819700825,

'St': 2.928,

'Dt': 7.072,

'PI': 0.656,

'PPI': 0.656,

'PW': 796.0,

'FWHM': 3.928,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9367.37458611151,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.6611516365571637,

'Energy': 483.77852131379717,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.6707818656650832,

'Median': 0.6557325241403626,

'Standard\_deviation': 0.18392990515048763,

'Skewness': 0.06936961410315892,

'Kurtosis': -1.1714992670176014,

'Min\_value': 0.33884836344283636,

'Max\_value': 1.0,

'Variance': 0.03383021000866738,

'Slope': -0.032053643975791385,

'Peak\_count': 15,

'Amplitude\_ratio': 2.951172583038607,

'Area\_ratio': 0.914884656105978,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.33884836344283636,

'SA': 2.6672149921567714,

'DA': 2.9406499221808824,

'St': 3.564,

'Dt': 6.436,

'PI': 0.636,

'PPI': 0.636,

'PW': 784.0,

'FWHM': 3.908,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9340.69328951432,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.6611516365571637,

'Energy': 480.3756279974275,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.6676985357281815,

'Median': 0.6548580453644702,

'Standard\_deviation': 0.18588785163068028,

'Skewness': 0.04010510660784447,

'Kurtosis': -1.1966301778636026,

'Min\_value': 0.33884836344283636,

'Max\_value': 1.0,

'Variance': 0.034554293383869804,

'Slope': -0.03326662684357364,

'Peak\_count': 15,

'Amplitude\_ratio': 2.951172583038607,

'Area\_ratio': 0.9070154771019725,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.2592579789957532,

'SA': 2.5732492681888246,

'DA': 3.034615646148829,

'St': 4.236,

'Dt': 5.764,

'PI': 0.672,

'PPI': 0.672,

'PW': 735.0,

'FWHM': 3.928,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9499.548463257555,

'Signal\_area': 5.607864914337654,

'Rise\_time': 1.9999999999999996,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.7407420210042468,

'Energy': 450.7633709234228,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.6441364582714125,

'Median': 0.6337961145997836,

'Standard\_deviation': 0.18934517170760837,

'Skewness': 0.06841472568664539,

'Kurtosis': -1.0835711895879256,

'Min\_value': 0.2592579789957532,

'Max\_value': 1.0,

'Variance': 0.0358515940489837,

'Slope': -0.03691150779885744,

'Peak\_count': 15,

'Amplitude\_ratio': 3.8571619044225467,

'Area\_ratio': 0.8479654652326348,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.2592579789957532,

'SA': 2.480984076415888,

'DA': 3.1268808379217656,

'St': 4.904,

'Dt': 5.096,

'PI': 0.668,

'PPI': 0.668,

'PW': 654.0,

'FWHM': 3.852,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9808.28265331695,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.7407420210042468,

'Energy': 424.3111929322455,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.621101199477929,

'Median': 0.606908543347864,

'Standard\_deviation': 0.19632751447345204,

'Skewness': 0.16901595005097705,

'Kurtosis': -1.1861236354635425,

'Min\_value': 0.2592579789957532,

'Max\_value': 1.0,

'Variance': 0.038544492939323514,

'Slope': -0.028820063608410933,

'Peak\_count': 15,

'Amplitude\_ratio': 3.8571619044225467,

'Area\_ratio': 0.7934373597891363,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.12267344708857572,

'SA': 2.3008897036672487,

'DA': 3.306975210670405,

'St': 5.532,

'Dt': 4.468,

'PI': 0.628,

'PPI': 0.628,

'PW': 554.0,

'FWHM': 3.904,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9764.050415972564,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.8773265529114243,

'Energy': 377.32925814519217,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.5759877429237044,

'Median': 0.5461501601834868,

'Standard\_deviation': 0.2134651684627935,

'Skewness': 0.18303604924296868,

'Kurtosis': -0.9879384587721258,

'Min\_value': 0.12267344708857572,

'Max\_value': 1.0,

'Variance': 0.04556737814684881,

'Slope': -0.04416800040557653,

'Peak\_count': 15,

'Amplitude\_ratio': 8.151723325080734,

'Area\_ratio': 0.6957686577882759,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.12267344708857572,

'SA': 2.2466690055187133,

'DA': 3.3611959088189405,

'St': 5.852,

'Dt': 4.148,

'PI': 0.32,

'PPI': 0.32,

'PW': 534.0,

'FWHM': 3.232,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9850.413855724895,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0000000000000004,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.8773265529114243,

'Energy': 359.1565007431734,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.5620508156745917,

'Median': 0.5198623020506175,

'Standard\_deviation': 0.20797928104188526,

'Skewness': 0.29268097780695684,

'Kurtosis': -0.8426194472974227,

'Min\_value': 0.12267344708857572,

'Max\_value': 1.0,

'Variance': 0.043255381342699496,

'Slope': -0.04450741480685553,

'Peak\_count': 15,

'Amplitude\_ratio': 8.151723325080734,

'Area\_ratio': 0.668413584469746,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.12267344708857572,

'SA': 2.2423960305368897,

'DA': 3.365468883800764,

'St': 6.2,

'Dt': 3.8,

'PI': 0.348,

'PPI': 0.348,

'PW': 548.0,

'FWHM': 3.912,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9663.857605113502,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.8773265529114243,

'Energy': 358.066173048781,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.561387965881315,

'Median': 0.5272805241819167,

'Standard\_deviation': 0.2071466263602193,

'Skewness': 0.25667462290584503,

'Kurtosis': -0.8210800007384234,

'Min\_value': 0.12267344708857572,

'Max\_value': 1.0,

'Variance': 0.0429097248124203,

'Slope': -0.06115935997572786,

'Peak\_count': 15,

'Amplitude\_ratio': 8.151723325080734,

'Area\_ratio': 0.6662952794870172,

'Interval\_ratio': 1.0},

{'sa': 1.0,

'Da': 0.12267344708857572,

'SA': 2.1553895827542355,

'DA': 3.4524753315834182,

'St': 6.78,

'Dt': 3.22,

'PI': 0.58,

'PPI': 0.58,

'PW': 511.0,

'FWHM': 3.244,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -9654.36959473671,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.8773265529114243,

'Energy': 331.63534113620466,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.5394971656607399,

'Median': 0.5065602170806158,

'Standard\_deviation': 0.2014401880962009,

'Skewness': 0.365156665850577,

'Kurtosis': -0.6688035019989553,

'Min\_value': 0.12267344708857572,

'Max\_value': 1.0,

'Variance': 0.040578149380232804,

'Slope': -0.07323088428292303,

'Peak\_count': 15,

'Amplitude\_ratio': 8.151723325080734,

'Area\_ratio': 0.6243026743845562,

'Interval\_ratio': 1.0},

{'sa': 0.9225169767263066,

'Da': 0.12267344708857572,

'SA': 1.971193189949853,

'DA': 3.636671724387801,

'St': 7.468,

'Dt': 2.532,

'PI': 0.688,

'PPI': 0.688,

'PW': 528.0,

'FWHM': 3.908,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -8343.300679871641,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 2.0,

'Amplitude\_modulation\_depth': 0.7998435296377309,

'Energy': 277.3839907260321,

'Zero\_crossing\_rate': 0.0,

'Mean': 0.4934655321469487,

'Median': 0.47321814610304347,

'Standard\_deviation': 0.18405368594233817,

'Skewness': 0.4025070921561108,

'Kurtosis': -0.5598295517443255,

'Min\_value': 0.12267344708857572,

'Max\_value': 0.9225169767263066,

'Variance': 0.033875759308960855,

'Slope': -0.058438056815692814,

'Peak\_count': 15,

'Amplitude\_ratio': 7.5201031569627945,

'Area\_ratio': 0.5420322039877505,

'Interval\_ratio': 1.0},

{'sa': 0.89898974326452,

'Da': 0.0,

'SA': 1.677453327933662,

'DA': 3.9304115864039915,

'St': 8.148,

'Dt': 1.852,

'PI': 0.68,

'PPI': 0.68,

'PW': 430.0,

'FWHM': 3.24,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -7641.762790829474,

'Signal\_area': 5.607864914337654,

'Rise\_time': 2.0,

'Fall\_time': 1.852,

'Amplitude\_modulation\_depth': 0.89898974326452,

'Energy': 219.1657913237154,

'Zero\_crossing\_rate': 0.0020768431983385254,

'Mean': 0.43595945573728917,

'Median': 0.4215533999001723,

'Standard\_deviation': 0.19371588593302336,

'Skewness': 0.15448196573864245,

'Kurtosis': -0.27563007835440256,

'Min\_value': 0.0,

'Max\_value': 0.89898974326452,

'Variance': 0.037525844462816114,

'Slope': -0.08207278722474991,

'Peak\_count': 15,

'Amplitude\_ratio': inf,

'Area\_ratio': 0.42678821061292366,

'Interval\_ratio': 1.0},

{'sa': 0.895170060113185,

'Da': 0.0,

'SA': 1.3226226644530517,

'DA': 4.285242249884602,

'St': 8.796,

'Dt': 1.204,

'PI': 0.648,

'PPI': 0.648,

'PW': 334.0,

'FWHM': 1.972,

'Dominant\_frequency': 0.0,

'Spectral\_entropy': -5638.439458933469,

'Signal\_area': 5.607864914337654,

'Rise\_time': 1.9999999999999991,

'Fall\_time': 1.204,

'Amplitude\_modulation\_depth': 0.895170060113185,

'Energy': 165.56553267603078,

'Zero\_crossing\_rate': 0.0024968789013732834,

'Mean': 0.41350627014167024,

'Median': 0.3997904275706776,

'Standard\_deviation': 0.18897382681268085,

'Skewness': 0.11994273090148866,

'Kurtosis': -0.30367458611001785,

'Min\_value': 0.0,

'Max\_value': 0.895170060113185,

'Variance': 0.0357111072202291,

'Slope': -0.09257242409396978,

'Peak\_count': 15,

'Amplitude\_ratio': inf,

'Area\_ratio': 0.30864594982667987,

'Interval\_ratio': 1.0}]



**Classifying the Dataset**

طبقه بندی مجموعه داده

electrical\_activity\_counts **=** {}

**for** patient\_info **in** pleth\_infos**.**values():

electrical\_activity **=** patient\_info**.**get('electrical\_activity')

electrical\_activity\_counts[electrical\_activity] **=** electrical\_activity\_counts**.**get(electrical\_activity, 0) **+** 1

print("Unique electrical activities and their counts:")

**for** activity, count **in** electrical\_activity\_counts**.**items():

print(f"{activity}: {count}")

Unique electrical activities and their counts:

Ventricular\_Tachycardia: 341

Asystole: 122

Tachycardia: 140

Ventricular\_Flutter\_Fib: 58

Bradycardia: 89

In [76]:

*# Initialize a dictionary to store patient IDs for each unique electrical activity*

patient\_ids\_by\_activity **=** {}

*# Iterate through the nested dictionaries in pleth\_infos*

**for** patient\_id, patient\_info **in** pleth\_infos**.**items():

*# Extract the electrical activity for the current patient*

electrical\_activity **=** patient\_info**.**get('electrical\_activity')

*# Extract the validity of the electrical activity*

validity **=** patient\_info**.**get('validity')

*# Check if the validity is 'True alarm'*

**if** validity **==** 'True alarm':

*# Add the patient ID to the list associated with the electrical activity*

**if** electrical\_activity:

**if** electrical\_activity **not** **in** patient\_ids\_by\_activity:

patient\_ids\_by\_activity[electrical\_activity] **=** []

patient\_ids\_by\_activity[electrical\_activity]**.**append(patient\_id)

*# Print the list of patient IDs for each unique electrical activity*

print("List of patient IDs for each unique electrical activity with validity 'True alarm':")

**for** activity, patient\_ids **in** patient\_ids\_by\_activity**.**items():

print(f"{activity}: {patient\_ids}")

List of patient IDs for each unique electrical activity with validity 'True alarm':

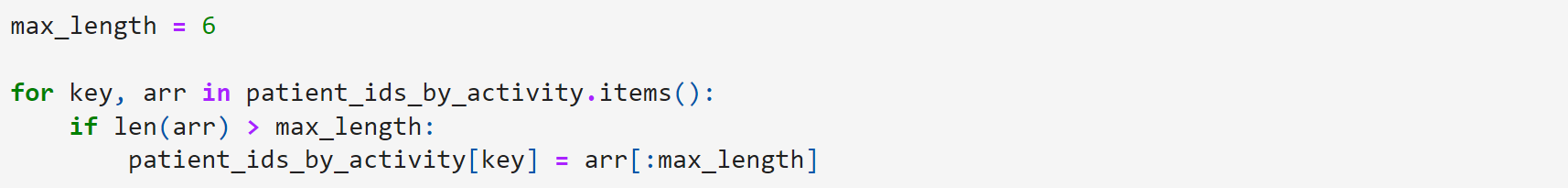
Tachycardia: ['t106s', 't107l', 't108s', 't110s', 't112s', 't114s', 't117l', 't118s', 't149l', 't150s', 't151l', 't156s', 't157l', 't173l', 't174s', 't175l', 't191l', 't192s', 't193l', 't195l', 't208s', 't209l', 't213l', 't214s', 't234s', 't235l', 't238s', 't240s', 't249l', 't251l', 't252s', 't263l', 't264s', 't270s', 't276s', 't277l', 't284s', 't300s', 't305l', 't320s', 't333l', 't335l', 't342s', 't343l', 't344s', 't350s', 't351l', 't356s', 't357l', 't358s', 't393l', 't394s', 't406s', 't410s', 't411l', 't412s', 't413l', 't416s', 't417l', 't418s', 't424s', 't425l', 't430s', 't434s', 't444s', 't445l', 't447l', 't458s', 't467l', 't468s', 't477l', 't478s', 't506s', 't507l', 't508s', 't509l', 't520s', 't521l', 't524s', 't546s', 't547l', 't565l', 't567l', 't577l', 't580s', 't589l', 't594s', 't595l', 't614s', 't622s', 't662s', 't665l', 't677l', 't678s', 't679l', 't680s', 't683l', 't688s', 't689l', 't690s', 't693l', 't698s', 't700s', 't702s', 't707l', 't709l', 't716s', 't717l', 't719l', 't731l', 't737l', 't739l', 't741l', 't742s', 't744s', 't745l', 't747l', 't752s', 't755l', 't760s', 't762s', 't771l', 't777l', 't786s', 't787l', 't790s', 't800s', 't801l', 't812s', 't816s', 't821l']

Bradycardia: ['b124s', 'b125l', 'b126s', 'b183l', 'b187l', 'b220s', 'b227l', 'b228s', 'b229l', 'b265l', 'b268s', 'b269l', 'b299l', 'b313l', 'b379l', 'b455l', 'b456s', 'b494s', 'b495l', 'b497l', 'b515l', 'b516s', 'b517l', 'b537l', 'b538s', 'b560s', 'b561l', 'b562s', 'b578s', 'b588s', 'b656s', 'b659l', 'b664s', 'b672s', 'b708s', 'b722s', 'b730s', 'b734s', 'b757l', 'b764s', 'b794s', 'b820s', 'b832s', 'b838s', 'b839l', 'b840s']

Ventricular\_Tachycardia: ['v131l', 'v132s', 'v133l', 'v139l', 'v143l', 'v158s', 'v159l', 'v188s', 'v194s', 'v197l', 'v199l', 'v206s', 'v221l', 'v253l', 'v254s', 'v255l', 'v275l', 'v290s', 'v309l', 'v318s', 'v328s', 'v329l', 'v334s', 'v348s', 'v368s', 'v369l', 'v404s', 'v448s', 'v471l', 'v522s', 'v523l', 'v525l', 'v534s', 'v541l', 'v542s', 'v559l', 'v564s', 'v571l', 'v573l', 'v574s', 'v579l', 'v596s', 'v597l', 'v607l', 'v616s', 'v625l', 'v626s', 'v628s', 'v629l', 'v630s', 'v632s', 'v635l', 'v636s', 'v638s', 'v646s', 'v648s', 'v652s', 'v696s', 'v701l', 'v714s', 'v724s', 'v726s', 'v728s', 'v729l', 'v733l', 'v748s', 'v758s', 'v761l', 'v765l', 'v769l', 'v772s', 'v773l', 'v783l', 'v788s', 'v793l', 'v797l', 'v803l', 'v805l', 'v806s', 'v813l', 'v815l', 'v818s', 'v823l', 'v828s', 'v831l', 'v836s', 'v837l', 'v842s', 'v844s']

Asystole: ['a142s', 'a161l', 'a167l', 'a172s', 'a185l', 'a203l', 'a345l', 'a372s', 'a385l', 'a386s', 'a442s', 'a443l', 'a446s', 'a449l', 'a604s', 'a639l', 'a653l', 'a654s', 'a670s', 'a754s', 'a776s', 'a796s']

Ventricular\_Flutter\_Fib: ['f450s', 'f543l', 'f544s', 'f545l', 'f563l', 'f697l']



**Dataset Formation**

تشکیل مجموعه داده

dataset **=** []

i**=**0

**for** label, patient\_ids **in** patient\_ids\_by\_activity**.**items():

**for** patient\_id **in** patient\_ids:

**if** patient\_id **not** **in** pleth\_signals:

print(f"Patient ID {patient\_id} not found in pleth\_signal dictionary")

**continue**

fs **=** pleth\_infos[patient\_id]['frequency']

segmented\_signal **=** segment\_signal(pleth\_signals[patient\_id], 10, fs)

**for** segment **in** segmented\_signal:

segment\_processed **=** bandpass\_filter(segment, lowcut**=**0.05, highcut**=**10, fs**=**fs)

segment\_processed **=** moving\_average(segment\_processed, window\_size**=** 3)

segment\_processed **=** remove\_baseline\_wandering(segment\_processed)

**if** **not** any(segment\_processed):

**continue**

*# segment\_processed = normalize\_signal(segment\_processed)*

*# dataset.append((label,segment\_processed))*

features, peaks **=** extract\_ppg\_features(segment\_processed,fs)

**for** feature **in** features:

print(i)

peak\_array **=** [feature[key] **for** key **in** feature**.**keys()]

dataset**.**append((label, peak\_array))

i **+=** 1

In [79]:

len(dataset)

Out[79]:

15949

In [80]:

**from** tensorflow.keras.utils **import** to\_categorical

Separate labels and signals: Extract labels and signals from your array of tuples.

In [81]:

labels **=** [label **for** label, \_ **in** dataset]

signals **=** [signal **for** \_, signal **in** dataset]

**Convert labels to one-hot encoding**: Convert your labels into one-hot encoded vectors to match the output shape expected by the models.

تبدیل برچسب‌ها به کدگذاری یک‌طرفه: برچسب‌های خود را به بردارهای کدگذاری شده تک داغ تبدیل کنید تا با شکل خروجی مورد انتظار مدل‌ها مطابقت داشته باشند.

*# Convert labels to one-hot encoding*

unique\_labels **=** set(labels)

label\_to\_index **=** {label: i **for** i, label **in** enumerate(unique\_labels)}

index\_to\_label **=** {i: label **for** label, i **in** label\_to\_index**.**items()}

encoded\_labels **=** [to\_categorical(label\_to\_index[label], num\_classes**=**len(unique\_labels)) **for** label **in** labels]

**Prepare input sequences**: Convert your signals into sequences that can be fed into the LSTM or BiLSTM models. Assuming each signal is of fixed length and you want to pad them to a maximum length.

دنباله های ورودی را آماده کنید: سیگنال های خود را به دنباله هایی تبدیل کنید که می توانند به مدل های LSTM یا BiLSTM وارد شوند. با فرض اینکه طول هر سیگنال ثابت است و شما می خواهید آنها را به حداکثر طول اضافه کنید.

*# Prepare input sequences*

*# Assuming each signal is of fixed length and you want to pad them to a maximum length*

max\_length **=** max(len(signal) **for** signal **in** signals)

padded\_signals **=** [np**.**pad(signal, (0, max\_length **-** len(signal))) **for** signal **in** signals]

**Convert to numpy arrays:** Convert the labels and padded signals into numpy arrays.

تبدیل به آرایه‌های کم‌رنگ: برچسب‌ها و سیگنال‌های پرشده را به آرایه‌های کم‌رنگ تبدیل کنید.

*# Convert to numpy arrays*

X **=** np**.**array(padded\_signals)

y **=** np**.**array(encoded\_labels)

**Split data into train and test sets**: Split your data into training and testing sets. Replace this with your actual splitting logic.

In [85]:

*# Split data into train and test sets (replace with your actual splitting logic)*

**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**=**42)

In [86]:

X\_train **=** X\_train**.**reshape(X\_train**.**shape[0], X\_train**.**shape[1], 1)

X\_test **=** X\_test**.**reshape(X\_test**.**shape[0], X\_test**.**shape[1], 1)

**Define input shape and number of classes:** Define the input shape and the number of classes based on your data.

In [87]:

*# Define input shape*

input\_shape **=** X\_train**.**shape[1:] *# Shape of input features*

*# Define number of classes*

num\_classes **=** len(unique\_labels)

**Create and compile models**: Create and compile your LSTM and BiLSTM models.

In [88]:

**def** create\_lstm\_model(input\_shape, num\_classes):

model **=** Sequential([

LSTM(64, return\_sequences**=True**, input\_shape**=**input\_shape),

LSTM(64),

Dense(32, activation**=**'relu'),

Dense(num\_classes, activation**=**'softmax')

])

**return** model

In [89]:

**def** create\_bilstm\_model(input\_shape, num\_classes):

model **=** Sequential([

Bidirectional(LSTM(64, return\_sequences**=True**), input\_shape**=**input\_shape),

Bidirectional(LSTM(64)), *# Second layer of Bidirectional LSTM*

Dense(32, activation**=**'relu'),

Dense(num\_classes, activation**=**'softmax')

])

**return** model

In [167]:

*# Create and compile LSTM model*

lstm\_model **=** create\_lstm\_model(input\_shape, num\_classes)

lstm\_model**.**compile(loss**=**'categorical\_crossentropy', optimizer**=**'adam', metrics**=**['accuracy'])

lstm\_model**.**summary()

*# Create and compile BiLSTM model*

bilstm\_model **=** create\_bilstm\_model(input\_shape, num\_classes)

bilstm\_model**.**compile(loss**=**'categorical\_crossentropy', optimizer**=**'adam', metrics**=**['accuracy'])

bilstm\_model**.**summary()

Model: "sequential\_4"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_8 (LSTM) (None, 31, 64) 16896

lstm\_9 (LSTM) (None, 64) 33024

dense\_8 (Dense) (None, 32) 2080

dense\_9 (Dense) (None, 5) 165

=================================================================

Total params: 52165 (203.77 KB)

Trainable params: 52165 (203.77 KB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Model: "sequential\_5"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

bidirectional\_4 (Bidirecti (None, 31, 128) 33792

onal)

bidirectional\_5 (Bidirecti (None, 128) 98816

onal)

dense\_10 (Dense) (None, 32) 4128

dense\_11 (Dense) (None, 5) 165

=================================================================

Total params: 136901 (534.77 KB)

Trainable params: 136901 (534.77 KB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Train the models:** Train your LSTM and BiLSTM models.

In [168]:

*# Train the LSTM model*

history\_lstm **=** lstm\_model**.**fit(X\_train, y\_train, epochs**=**50, batch\_size**=**64, validation\_data**=**(X\_test, y\_test))

*# Train the BiLSTM model*

history\_bilstm **=** bilstm\_model**.**fit(X\_train, y\_train, epochs**=**50, batch\_size**=**64, validation\_data**=**(X\_test, y\_test))

Epoch 1/50

200/200 [==============================] - 5s 18ms/step - loss: 1.4583 - accuracy: 0.3397 - val\_loss: 1.3988 - val\_accuracy: 0.3655

Epoch 2/50

200/200 [==============================] - 3s 16ms/step - loss: 1.2339 - accuracy: 0.4687 - val\_loss: 1.1449 - val\_accuracy: 0.5119

Epoch 3/50

200/200 [==============================] - 3s 16ms/step - loss: 1.0925 - accuracy: 0.5518 - val\_loss: 1.0466 - val\_accuracy: 0.5850

Epoch 4/50

200/200 [==============================] - 3s 16ms/step - loss: 1.0034 - accuracy: 0.6008 - val\_loss: 0.9875 - val\_accuracy: 0.6082

Epoch 5/50

200/200 [==============================] - 3s 16ms/step - loss: 0.9480 - accuracy: 0.6157 - val\_loss: 0.9579 - val\_accuracy: 0.6144

Epoch 6/50

200/200 [==============================] - 3s 16ms/step - loss: 0.8839 - accuracy: 0.6494 - val\_loss: 0.8940 - val\_accuracy: 0.6361

Epoch 7/50

200/200 [==============================] - 3s 16ms/step - loss: 0.8415 - accuracy: 0.6626 - val\_loss: 0.8595 - val\_accuracy: 0.6599

Epoch 8/50

200/200 [==============================] - 3s 16ms/step - loss: 0.7991 - accuracy: 0.6833 - val\_loss: 0.8373 - val\_accuracy: 0.6690

Epoch 9/50

200/200 [==============================] - 3s 16ms/step - loss: 0.7681 - accuracy: 0.6956 - val\_loss: 0.8087 - val\_accuracy: 0.6871

Epoch 10/50

200/200 [==============================] - 3s 16ms/step - loss: 0.7257 - accuracy: 0.7131 - val\_loss: 0.7973 - val\_accuracy: 0.6947

Epoch 11/50

200/200 [==============================] - 3s 16ms/step - loss: 0.7053 - accuracy: 0.7236 - val\_loss: 0.7452 - val\_accuracy: 0.7078

Epoch 12/50

200/200 [==============================] - 3s 16ms/step - loss: 0.6667 - accuracy: 0.7431 - val\_loss: 0.7396 - val\_accuracy: 0.7257

Epoch 13/50

200/200 [==============================] - 3s 17ms/step - loss: 0.6531 - accuracy: 0.7459 - val\_loss: 0.7114 - val\_accuracy: 0.7232

Epoch 14/50

200/200 [==============================] - 3s 17ms/step - loss: 0.6362 - accuracy: 0.7545 - val\_loss: 0.7288 - val\_accuracy: 0.7194

Epoch 15/50

200/200 [==============================] - 3s 16ms/step - loss: 0.6081 - accuracy: 0.7660 - val\_loss: 0.6880 - val\_accuracy: 0.7392

Epoch 16/50

200/200 [==============================] - 3s 17ms/step - loss: 0.5948 - accuracy: 0.7703 - val\_loss: 0.6969 - val\_accuracy: 0.7345

Epoch 17/50

200/200 [==============================] - 3s 16ms/step - loss: 0.5586 - accuracy: 0.7824 - val\_loss: 0.7037 - val\_accuracy: 0.7332

Epoch 18/50

200/200 [==============================] - 3s 16ms/step - loss: 0.5406 - accuracy: 0.7924 - val\_loss: 0.6478 - val\_accuracy: 0.7558

Epoch 19/50

200/200 [==============================] - 3s 16ms/step - loss: 0.5201 - accuracy: 0.8005 - val\_loss: 0.6703 - val\_accuracy: 0.7423

Epoch 20/50

200/200 [==============================] - 3s 16ms/step - loss: 0.4889 - accuracy: 0.8100 - val\_loss: 0.6537 - val\_accuracy: 0.7574

Epoch 21/50

200/200 [==============================] - 3s 16ms/step - loss: 0.4836 - accuracy: 0.8133 - val\_loss: 0.6058 - val\_accuracy: 0.7683

Epoch 22/50

200/200 [==============================] - 3s 16ms/step - loss: 0.4438 - accuracy: 0.8287 - val\_loss: 0.6300 - val\_accuracy: 0.7608

Epoch 23/50

200/200 [==============================] - 3s 16ms/step - loss: 0.4475 - accuracy: 0.8283 - val\_loss: 0.5967 - val\_accuracy: 0.7837

Epoch 24/50

200/200 [==============================] - 3s 16ms/step - loss: 0.4265 - accuracy: 0.8342 - val\_loss: 0.6164 - val\_accuracy: 0.7762

Epoch 25/50

200/200 [==============================] - 3s 17ms/step - loss: 0.4087 - accuracy: 0.8429 - val\_loss: 0.6461 - val\_accuracy: 0.7649

Epoch 26/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3987 - accuracy: 0.8500 - val\_loss: 0.5882 - val\_accuracy: 0.7881

Epoch 27/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3829 - accuracy: 0.8571 - val\_loss: 0.5920 - val\_accuracy: 0.7881

Epoch 28/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3643 - accuracy: 0.8624 - val\_loss: 0.6059 - val\_accuracy: 0.7850

Epoch 29/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3504 - accuracy: 0.8639 - val\_loss: 0.5986 - val\_accuracy: 0.7978

Epoch 30/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3488 - accuracy: 0.8667 - val\_loss: 0.5770 - val\_accuracy: 0.8031

Epoch 31/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3284 - accuracy: 0.8792 - val\_loss: 0.5961 - val\_accuracy: 0.7962

Epoch 32/50

200/200 [==============================] - 3s 17ms/step - loss: 0.3152 - accuracy: 0.8849 - val\_loss: 0.5465 - val\_accuracy: 0.8085

Epoch 33/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2996 - accuracy: 0.8893 - val\_loss: 0.5915 - val\_accuracy: 0.8028

Epoch 34/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2948 - accuracy: 0.8910 - val\_loss: 0.5660 - val\_accuracy: 0.8066

Epoch 35/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2717 - accuracy: 0.9003 - val\_loss: 0.5930 - val\_accuracy: 0.8016

Epoch 36/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2805 - accuracy: 0.8973 - val\_loss: 0.5441 - val\_accuracy: 0.8204

Epoch 37/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2552 - accuracy: 0.9063 - val\_loss: 0.5977 - val\_accuracy: 0.8034

Epoch 38/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2641 - accuracy: 0.9038 - val\_loss: 0.5929 - val\_accuracy: 0.8138

Epoch 39/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2609 - accuracy: 0.9055 - val\_loss: 0.5879 - val\_accuracy: 0.8163

Epoch 40/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2356 - accuracy: 0.9167 - val\_loss: 0.6291 - val\_accuracy: 0.8113

Epoch 41/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2395 - accuracy: 0.9138 - val\_loss: 0.6347 - val\_accuracy: 0.7940

Epoch 42/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2355 - accuracy: 0.9115 - val\_loss: 0.5932 - val\_accuracy: 0.8116

Epoch 43/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2098 - accuracy: 0.9230 - val\_loss: 0.5720 - val\_accuracy: 0.8238

Epoch 44/50

200/200 [==============================] - 3s 17ms/step - loss: 0.1791 - accuracy: 0.9364 - val\_loss: 0.6171 - val\_accuracy: 0.8179

Epoch 45/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2008 - accuracy: 0.9249 - val\_loss: 0.6242 - val\_accuracy: 0.8176

Epoch 46/50

200/200 [==============================] - 3s 17ms/step - loss: 0.2013 - accuracy: 0.9252 - val\_loss: 0.5911 - val\_accuracy: 0.8122

Epoch 47/50

200/200 [==============================] - 3s 17ms/step - loss: 0.1942 - accuracy: 0.9276 - val\_loss: 0.6087 - val\_accuracy: 0.8207

Epoch 48/50

200/200 [==============================] - 3s 16ms/step - loss: 0.1877 - accuracy: 0.9321 - val\_loss: 0.5788 - val\_accuracy: 0.8273

Epoch 49/50

200/200 [==============================] - 3s 15ms/step - loss: 0.1645 - accuracy: 0.9432 - val\_loss: 0.6312 - val\_accuracy: 0.8163

Epoch 50/50

200/200 [==============================] - 3s 16ms/step - loss: 0.1842 - accuracy: 0.9308 - val\_loss: 0.5797 - val\_accuracy: 0.8310

Epoch 1/50

200/200 [==============================] - 8s 24ms/step - loss: 1.4592 - accuracy: 0.3574 - val\_loss: 1.3165 - val\_accuracy: 0.4110

Epoch 2/50

200/200 [==============================] - 5s 23ms/step - loss: 1.1728 - accuracy: 0.5020 - val\_loss: 1.1088 - val\_accuracy: 0.5451

Epoch 3/50

200/200 [==============================] - 5s 23ms/step - loss: 1.0583 - accuracy: 0.5632 - val\_loss: 1.0310 - val\_accuracy: 0.5828

Epoch 4/50

200/200 [==============================] - 5s 23ms/step - loss: 0.9648 - accuracy: 0.6116 - val\_loss: 1.0188 - val\_accuracy: 0.5784

Epoch 5/50

200/200 [==============================] - 5s 25ms/step - loss: 0.9057 - accuracy: 0.6414 - val\_loss: 0.8842 - val\_accuracy: 0.6564

Epoch 6/50

200/200 [==============================] - 5s 24ms/step - loss: 0.8467 - accuracy: 0.6627 - val\_loss: 0.9481 - val\_accuracy: 0.6097

Epoch 7/50

200/200 [==============================] - 5s 25ms/step - loss: 0.7911 - accuracy: 0.6878 - val\_loss: 0.8378 - val\_accuracy: 0.6806

Epoch 8/50

200/200 [==============================] - 5s 23ms/step - loss: 0.7515 - accuracy: 0.7034 - val\_loss: 0.7837 - val\_accuracy: 0.6956

Epoch 9/50

200/200 [==============================] - 4s 22ms/step - loss: 0.7141 - accuracy: 0.7175 - val\_loss: 0.7657 - val\_accuracy: 0.7075

Epoch 10/50

200/200 [==============================] - 4s 22ms/step - loss: 0.6792 - accuracy: 0.7319 - val\_loss: 0.7258 - val\_accuracy: 0.7122

Epoch 11/50

200/200 [==============================] - 5s 23ms/step - loss: 0.6420 - accuracy: 0.7457 - val\_loss: 0.7048 - val\_accuracy: 0.7216

Epoch 12/50

200/200 [==============================] - 5s 23ms/step - loss: 0.6235 - accuracy: 0.7539 - val\_loss: 0.6975 - val\_accuracy: 0.7288

Epoch 13/50

200/200 [==============================] - 5s 23ms/step - loss: 0.5757 - accuracy: 0.7775 - val\_loss: 0.6767 - val\_accuracy: 0.7376

Epoch 14/50

200/200 [==============================] - 5s 23ms/step - loss: 0.5452 - accuracy: 0.7865 - val\_loss: 0.6378 - val\_accuracy: 0.7517

Epoch 15/50

200/200 [==============================] - 5s 23ms/step - loss: 0.5170 - accuracy: 0.7972 - val\_loss: 0.6291 - val\_accuracy: 0.7608

Epoch 16/50

200/200 [==============================] - 5s 23ms/step - loss: 0.4941 - accuracy: 0.8081 - val\_loss: 0.6126 - val\_accuracy: 0.7665

Epoch 17/50

200/200 [==============================] - 4s 22ms/step - loss: 0.4788 - accuracy: 0.8139 - val\_loss: 0.6402 - val\_accuracy: 0.7511

Epoch 18/50

200/200 [==============================] - 5s 23ms/step - loss: 0.4509 - accuracy: 0.8263 - val\_loss: 0.5948 - val\_accuracy: 0.7696

Epoch 19/50

200/200 [==============================] - 5s 23ms/step - loss: 0.4192 - accuracy: 0.8352 - val\_loss: 0.5524 - val\_accuracy: 0.7890

Epoch 20/50

200/200 [==============================] - 5s 23ms/step - loss: 0.3938 - accuracy: 0.8519 - val\_loss: 0.5784 - val\_accuracy: 0.7828

Epoch 21/50

200/200 [==============================] - 5s 23ms/step - loss: 0.3717 - accuracy: 0.8588 - val\_loss: 0.5700 - val\_accuracy: 0.7840

Epoch 22/50

200/200 [==============================] - 5s 23ms/step - loss: 0.3768 - accuracy: 0.8569 - val\_loss: 0.6020 - val\_accuracy: 0.7846

Epoch 23/50

200/200 [==============================] - 5s 24ms/step - loss: 0.3396 - accuracy: 0.8747 - val\_loss: 0.5676 - val\_accuracy: 0.7940

Epoch 24/50

200/200 [==============================] - 4s 22ms/step - loss: 0.3237 - accuracy: 0.8769 - val\_loss: 0.5755 - val\_accuracy: 0.7991

Epoch 25/50

200/200 [==============================] - 5s 23ms/step - loss: 0.3142 - accuracy: 0.8828 - val\_loss: 0.5484 - val\_accuracy: 0.8050

Epoch 26/50

200/200 [==============================] - 4s 22ms/step - loss: 0.2817 - accuracy: 0.8961 - val\_loss: 0.5169 - val\_accuracy: 0.8191

Epoch 27/50

200/200 [==============================] - 5s 26ms/step - loss: 0.2704 - accuracy: 0.8977 - val\_loss: 0.5370 - val\_accuracy: 0.8182

Epoch 28/50

200/200 [==============================] - 5s 27ms/step - loss: 0.2615 - accuracy: 0.9011 - val\_loss: 0.5451 - val\_accuracy: 0.8176

Epoch 29/50

200/200 [==============================] - 6s 29ms/step - loss: 0.2704 - accuracy: 0.8982 - val\_loss: 0.5098 - val\_accuracy: 0.8248

Epoch 30/50

200/200 [==============================] - 5s 26ms/step - loss: 0.2334 - accuracy: 0.9154 - val\_loss: 0.4901 - val\_accuracy: 0.8436

Epoch 31/50

200/200 [==============================] - 5s 25ms/step - loss: 0.2371 - accuracy: 0.9124 - val\_loss: 0.5098 - val\_accuracy: 0.8279

Epoch 32/50

200/200 [==============================] - 5s 26ms/step - loss: 0.1977 - accuracy: 0.9262 - val\_loss: 0.5765 - val\_accuracy: 0.8207

Epoch 33/50

200/200 [==============================] - 5s 26ms/step - loss: 0.2001 - accuracy: 0.9263 - val\_loss: 0.5458 - val\_accuracy: 0.8332

Epoch 34/50

200/200 [==============================] - 5s 26ms/step - loss: 0.1972 - accuracy: 0.9270 - val\_loss: 0.5491 - val\_accuracy: 0.8379

Epoch 35/50

200/200 [==============================] - 5s 27ms/step - loss: 0.1884 - accuracy: 0.9313 - val\_loss: 0.5523 - val\_accuracy: 0.8276

Epoch 36/50

200/200 [==============================] - 5s 26ms/step - loss: 0.2056 - accuracy: 0.9242 - val\_loss: 0.5900 - val\_accuracy: 0.8103

Epoch 37/50

200/200 [==============================] - 5s 27ms/step - loss: 0.1743 - accuracy: 0.9370 - val\_loss: 0.5749 - val\_accuracy: 0.8257

Epoch 38/50

200/200 [==============================] - 5s 27ms/step - loss: 0.1564 - accuracy: 0.9450 - val\_loss: 0.5583 - val\_accuracy: 0.8364

Epoch 39/50

200/200 [==============================] - 5s 26ms/step - loss: 0.1456 - accuracy: 0.9471 - val\_loss: 0.5553 - val\_accuracy: 0.8392

Epoch 40/50

200/200 [==============================] - 6s 28ms/step - loss: 0.1331 - accuracy: 0.9539 - val\_loss: 0.5467 - val\_accuracy: 0.8476

Epoch 41/50

200/200 [==============================] - 5s 27ms/step - loss: 0.1120 - accuracy: 0.9621 - val\_loss: 0.6247 - val\_accuracy: 0.8339

Epoch 42/50

200/200 [==============================] - 5s 25ms/step - loss: 0.1676 - accuracy: 0.9384 - val\_loss: 0.6554 - val\_accuracy: 0.8207

Epoch 43/50

200/200 [==============================] - 6s 28ms/step - loss: 0.1426 - accuracy: 0.9480 - val\_loss: 0.5334 - val\_accuracy: 0.8555

Epoch 44/50

200/200 [==============================] - 5s 27ms/step - loss: 0.1061 - accuracy: 0.9635 - val\_loss: 0.5530 - val\_accuracy: 0.8408

Epoch 45/50

200/200 [==============================] - 5s 24ms/step - loss: 0.0965 - accuracy: 0.9690 - val\_loss: 0.5628 - val\_accuracy: 0.8476

Epoch 46/50

200/200 [==============================] - 5s 24ms/step - loss: 0.0994 - accuracy: 0.9664 - val\_loss: 0.5715 - val\_accuracy: 0.8549

Epoch 47/50

200/200 [==============================] - 5s 24ms/step - loss: 0.0800 - accuracy: 0.9733 - val\_loss: 0.5819 - val\_accuracy: 0.8470

Epoch 48/50

200/200 [==============================] - 5s 23ms/step - loss: 0.0956 - accuracy: 0.9667 - val\_loss: 0.5396 - val\_accuracy: 0.8589

Epoch 49/50

200/200 [==============================] - 5s 23ms/step - loss: 0.0983 - accuracy: 0.9654 - val\_loss: 0.5814 - val\_accuracy: 0.8561

Epoch 50/50

200/200 [==============================] - 5s 25ms/step - loss: 0.0992 - accuracy: 0.9675 - val\_loss: 0.6257 - val\_accuracy: 0.8382

In [92]:

lstm\_evaluation **=** lstm\_model**.**evaluate(X\_test, y\_test)

print("LSTM Evaluation (Untrained Dataset):", lstm\_evaluation)

bilstm\_evaluation **=** bilstm\_model**.**evaluate(X\_test, y\_test)

print("BiLSTM Evaluation (Untrained Dataset):", bilstm\_evaluation)

100/100 [==============================] - 0s 4ms/step - loss: 0.5888 - accuracy: 0.8279

LSTM Evaluation (Untrained Dataset): [0.5887748003005981, 0.827899694442749]

70/100 [====================>.........] - ETA: 0s - loss: 0.6438 - accuracy: 0.8375

100/100 [==============================] - 1s 6ms/step - loss: 0.6291 - accuracy: 0.8395

BiLSTM Evaluation (Untrained Dataset): [0.6290794014930725, 0.8394984602928162]

In [93]:

lstm\_evaluation\_trained **=** lstm\_model**.**evaluate(X\_train, y\_train)

print("LSTM Evaluation (Trained Dataset):", lstm\_evaluation\_trained)

bilstm\_evaluation\_trained **=** bilstm\_model**.**evaluate(X\_train, y\_train)

print("BiLSTM Evaluation (Trained Dataset):", bilstm\_evaluation\_trained)

399/399 [==============================] - 2s 4ms/step - loss: 0.1493 - accuracy: 0.9482

LSTM Evaluation (Trained Dataset): [0.1493038684129715, 0.9481934309005737]

389/399 [============================>.] - ETA: 0s - loss: 0.0956 - accuracy: 0.9687

399/399 [==============================] - 2s 5ms/step - loss: 0.0948 - accuracy: 0.9689

BiLSTM Evaluation (Trained Dataset): [0.09480977803468704, 0.9688847064971924]

In [94]:

*# Save the trained models*

lstm\_model**.**save('/home/nakul/Documents/Python/Arrhythmia\_Classification\_PPG/lstm\_model.h5')

bilstm\_model**.**save('/home/nakul/Documents/Python/Arrhythmia\_Classification\_PPG/bilstm\_model.h5')

print("Models saved yaaaaay")

Models saved yaaaaay

/home/nakul/.local/lib/python3.10/site-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(

In [171]:

epochs **=** [i **for** i **in** range(50)]

fig , ax **=** plt**.**subplots(1,2)

train\_acc **=** history\_lstm**.**history['accuracy']

train\_loss **=** history\_lstm**.**history['loss']

test\_acc **=** history\_lstm**.**history['val\_accuracy']

test\_loss **=** history\_lstm**.**history['val\_loss']

fig**.**set\_size\_inches(20,6)

ax[0]**.**plot(epochs , train\_loss , label **=** 'Training Loss')

ax[0]**.**plot(epochs , test\_loss , label **=** 'Testing Loss')

ax[0]**.**set\_title('Training & Testing Loss')

ax[0]**.**legend()

ax[0]**.**set\_xlabel("Epochs")

ax[0]**.**set\_ylim(0, 1)

ax[1]**.**plot(epochs , train\_acc , label **=** 'Training Accuracy')

ax[1]**.**plot(epochs , test\_acc , label **=** 'Testing Accuracy')

ax[1]**.**set\_title('Training & Testing Accuracy for LSTM')

ax[1]**.**legend()

ax[1]**.**set\_xlabel("Epochs")

ax[1]**.**set\_ylim(0, 1)

plt**.**show()

In [170]:

epochs **=** [i **for** i **in** range(50)]

fig , ax **=** plt**.**subplots(1,2)

train\_acc **=** history\_bilstm**.**history['accuracy']

train\_loss **=** history\_bilstm**.**history['loss']

test\_acc **=** history\_bilstm**.**history['val\_accuracy']

test\_loss **=** history\_bilstm**.**history['val\_loss']

fig**.**set\_size\_inches(20,6)

ax[0]**.**plot(epochs , train\_loss , label **=** 'Training Loss')

ax[0]**.**plot(epochs , test\_loss , label **=** 'Testing Loss')

ax[0]**.**set\_title('Training & Testing Loss')

ax[0]**.**legend()

ax[0]**.**set\_xlabel("Epochs")

ax[0]**.**set\_ylim(0, 1)

ax[1]**.**plot(epochs , train\_acc , label **=** 'Training Accuracy')

ax[1]**.**plot(epochs , test\_acc , label **=** 'Testing Accuracy')

ax[1]**.**set\_title('Training & Testing Accuracy for BLSTM')

ax[1]**.**legend()

ax[1]**.**set\_xlabel("Epochs")

ax[1]**.**set\_ylim(0, 1)

plt**.**show()

In [97]:

lstm\_predictions **=** lstm\_model**.**predict(X\_test)

bilstm\_predictions **=** bilstm\_model**.**predict(X\_test)

*# Convert predictions to class labels*

lstm\_pred\_labels **=** np**.**argmax(lstm\_predictions, axis**=**1)

bilstm\_pred\_labels **=** np**.**argmax(bilstm\_predictions, axis**=**1)

100/100 [==============================] - 1s 4ms/step

100/100 [==============================] - 1s 6ms/step

In [98]:

lstm\_predictions\_train **=** lstm\_model**.**predict(X\_train)

bilstm\_predictions\_train **=** bilstm\_model**.**predict(X\_train)

*# Convert predictions to class labels*

lstm\_pred\_labels\_train **=** np**.**argmax(lstm\_predictions\_train, axis**=**1)

bilstm\_pred\_labels\_train **=** np**.**argmax(bilstm\_predictions\_train, axis**=**1)

399/399 [==============================] - 2s 4ms/step

399/399 [==============================] - 2s 5ms/step

In [99]:

y\_test\_labels **=** np**.**argmax(y\_test, axis**=**1)

In [100]:

y\_train\_labels **=** np**.**argmax(y\_train, axis**=**1)

In [213]:

class\_labels **=** [ 'Bradycardia' , 'Asystole', 'V. Flutter Fib', 'V. Tachycardia' ,'Tachycardia']

*# Compute confusion matrix*

conf\_matrix **=** confusion\_matrix(y\_test\_labels, lstm\_pred\_labels)

*# Plot confusion matrix*

plt**.**figure(figsize**=**(10, 8))

seaborn**.**heatmap(conf\_matrix, annot**=True**, fmt**=**'d', cmap**=**'Reds',

xticklabels**=**class\_labels,

yticklabels**=**class\_labels)

plt**.**xlabel('Predicted labels')

plt**.**ylabel('True labels')

plt**.**title('Confusion Matrix (Untrained Data) - LSTM Model')

plt**.**show()

In [214]:

*# Calculate True Positives, True Negatives, False Positives, and False Negatives*

TP **=** conf\_matrix[1, 1]

TN **=** conf\_matrix[0, 0]

FP **=** conf\_matrix[0, 1]

FN **=** conf\_matrix[1, 0]

*# Calculate accuracy*

accuracy **=** (TP **+** TN) **/** (TP **+** TN **+** FN **+** FP)

*# Calculate sensitivity*

sensitivity **=** TP **/** (TP **+** FN)

*# Calculate specificity*

specificity **=** TN **/** (TN **+** FP)

print("Accuracy (Untrained Dataset):", accuracy)

print("Sensitivity (Untrained Dataset):", sensitivity)

print("Specificity (Untrained Dataset):", specificity)

Accuracy (Untrained Dataset): 0.9274553571428571

Sensitivity (Untrained Dataset): 0.9440203562340967

Specificity (Untrained Dataset): 0.9145129224652088

In [215]:

*# Compute confusion matrix*

conf\_matrix **=** confusion\_matrix(y\_test\_labels, bilstm\_pred\_labels)

*# Plot confusion matrix*

plt**.**figure(figsize**=**(10, 8))

seaborn**.**heatmap(conf\_matrix, annot**=True**, fmt**=**'d', cmap**=**'Reds',

xticklabels**=**class\_labels,

yticklabels**=**class\_labels)

plt**.**xlabel('Predicted labels')

plt**.**ylabel('True labels')

plt**.**title('Confusion Matrix (Untrained Data) - BLSTM Model')

plt**.**show()

In [216]:

*# Calculate True Positives, True Negatives, False Positives, and False Negatives*

TP **=** conf\_matrix[1, 1]

TN **=** conf\_matrix[0, 0]

FP **=** conf\_matrix[0, 1]

FN **=** conf\_matrix[1, 0]

*# Calculate accuracy*

accuracy **=** (TP **+** TN) **/** (TP **+** TN **+** FN **+** FP)

*# Calculate sensitivity*

sensitivity **=** TP **/** (TP **+** FN)

*# Calculate specificity*

specificity **=** TN **/** (TN **+** FP)

print("Accuracy (Untrained Dataset):", accuracy)

print("Sensitivity (Untrained Dataset):", sensitivity)

print("Specificity (Untrained Dataset):", specificity)

Accuracy (Untrained Dataset): 0.9399773499433749

Sensitivity (Untrained Dataset): 0.9373368146214099

Specificity (Untrained Dataset): 0.942

In [217]:

*# Compute confusion matrix*

conf\_matrix\_train **=** confusion\_matrix(y\_train\_labels, lstm\_pred\_labels\_train)

*# Plot confusion matrix*

plt**.**figure(figsize**=**(10, 8))

seaborn**.**heatmap(conf\_matrix\_train, annot**=True**, fmt**=**'d', cmap**=**'Blues',

xticklabels**=**class\_labels,

yticklabels**=**class\_labels)

plt**.**xlabel('Predicted labels')

plt**.**ylabel('True labels')

plt**.**title('Confusion Matrix (Trained Data) - LSTM Model')

plt**.**show()

In [218]:

*# Calculate True Positives, True Negatidef collectInput(self):*

TP **=** conf\_matrix\_train[1, 1]

TN **=** conf\_matrix\_train[0, 0]

FP **=** conf\_matrix\_train[0, 1]

FN **=** conf\_matrix\_train[1, 0]

*# Calculate accuracy*

accuracy **=** (TP **+** TN) **/** (TP **+** TN **+** FN **+** FP)

*# Calculate sensitivity*

sensitivity **=** TP **/** (TP **+** FN)

*# Calculate specificity*

specificity **=** TN **/** (TN **+** FP)

print("Accuracy (Trained Dataset):", accuracy)

print("Sensitivity (Trained Dataset):", sensitivity)

print("Specificity (Trained Dataset):", specificity)

Accuracy (Trained Dataset): 0.9748227817159618

Sensitivity (Trained Dataset): 0.9801821103374397

Specificity (Trained Dataset): 0.9703237410071942

In [221]:

*# Compute confusion matrix*

conf\_matrix\_train **=** confusion\_matrix(y\_train\_labels, bilstm\_pred\_labels\_train)

*# Plot confusion matrix*

plt**.**figure(figsize**=**(10, 8))

seaborn**.**heatmap(conf\_matrix\_train, annot**=True**, fmt**=**'d', cmap**=**'Blues',

xticklabels**=**class\_labels,

yticklabels**=**class\_labels)

plt**.**xlabel('Predicted labels')

plt**.**ylabel('True labels')

plt**.**title('Confusion Matrix (Trained Data) - BLSTM Model')

plt**.**show()

In [220]:

*# Calculate True Positives, True Negatidef collectInput(self):*

TP **=** conf\_matrix\_train[1, 1]

TN **=** conf\_matrix\_train[0, 0]

FP **=** conf\_matrix\_train[0, 1]

FN **=** conf\_matrix\_train[1, 0]

*# Calculate accuracy*

accuracy **=** (TP **+** TN) **/** (TP **+** TN **+** FN **+** FP)

*# Calculate sensitivity*

sensitivity **=** TP **/** (TP **+** FN)

*# Calculate specificity*

specificity **=** TN **/** (TN **+** FP)

print("Accuracy (Trained Dataset):", accuracy)

print("Sensitivity (Trained Dataset):", sensitivity)

print("Specificity (Trained Dataset):", specificity)

Accuracy (Trained Dataset): 0.9891438440661239

Sensitivity (Trained Dataset): 0.9951403887688985

Specificity (Trained Dataset): 0.9840981372103589

**Model Testing**

Now that the Model is trained its time to test the Model by giving data from the datset and outside the dataset

In [156]:

*# Assuming you have already trained your model*

lstm\_model\_test **=** keras**.**models**.**load\_model('/home/nakul/Documents/Python/Arrhythmia\_Classification\_PPG/lstm\_model.h5')

bilstm\_model\_test **=** keras**.**models**.**load\_model('/home/nakul/Documents/Python/Arrhythmia\_Classification\_PPG/bilstm\_model.h5')

In [161]:

data **=** dataset[random**.**randint(0,len(dataset))]

input **=** data[1]

label **=** data[0]

print(label , input)

Ventricular\_Flutter\_Fib [1754.2313029955646, -3282.1048897207684, -1960.7128531600256, 972.6767297642912, 4.772, 5.228, 0.624, 0.624, 80.0, 2.964, 1.25, -123595294.74794412, -988.0361233957344, 2.0000000000000004, 2.0, 5036.336192716333, 1333047137.430105, 0.009, -489.64726263183167, -357.94613390851134, 1045.6063770020048, -0.36463463364166293, -0.11497016045357977, -3282.1048897207684, 1754.2313029955646, 1093292.6956272589, -11.426519051014022, 19, -0.5344836200968609, -2.015790851329573, 1.0]

In [162]:

input\_data **=** np**.**array(input)**.**reshape(1, 31, 1)

raw\_predictions **=** lstm\_model\_test**.**predict(input\_data)

predicted\_label\_index **=** np**.**argmax(raw\_predictions)

predicted\_percentage **=** raw\_predictions[0][predicted\_label\_index]

class\_labels **=** [ 'Bradycardia' , 'Asystole', 'Ventricular Flutter Fib', 'Ventricular Tachycardia' ,'Tachycardia']

predicted\_label **=** class\_labels[predicted\_label\_index]

prediction\_dict **=** {}

**for** label, raw\_prediction **in** zip(class\_labels, raw\_predictions[0]):

prediction\_dict[label] **=** f"{raw\_prediction**\***100}%"

prediction\_dict

1/1 [==============================] - 0s 16ms/step

Out[162]:

{'Bradycardia': '1.117078838319685e-05%',

'Asystole': '0.023479669471271336%',

'Ventricular Flutter Fib': '99.44393634796143%',

'Ventricular Tachycardia': '0.414813868701458%',

'Tachycardia': '0.11776362080127001%'}

**GUI Generation**

**Generating Random Patient Id from the Database**

In [207]:

record\_id **=** patient\_ids[random**.**randint(0,len(patient\_ids)**-**1)]

record\_id

Out[207]:

'f544s'

**Extracting their Header and Signal Location**

In [208]:

**def** extracting\_database(record\_name, data\_dir**=**'/home/nakul/Documents/Python/Arrhythmia\_Classification\_PPG/training'):

os**.**makedirs(data\_dir, exist\_ok**=True**)

base\_url **=** 'https://archive.physionet.org/physiobank/database/challenge/2015/training/'

*#!wget -N {base\_url + record\_name + '.mat'} -P $data\_dir*

*#!wget -N {base\_url + record\_name + '.hea'} -P $data\_dir # Uncomment these if you dont have the dataset in your drive*

record\_path **=** os**.**path**.**join(data\_dir, record\_name **+** '.mat')

hea\_file\_path **=** os**.**path**.**join(data\_dir, record\_name **+** '.hea')

**return** record\_path, hea\_file\_path

*# Example case*

record\_path\_test, hea\_file\_path\_test **=** extracting\_database(record\_id)

**Extracting their Cardiac Arrhytnmia Info**

In [209]:

**def** extract\_pleth\_info(file\_path):

pleth\_info **=** {}

**with** open(file\_path, 'r') **as** file:

lines **=** file**.**readlines()

first\_line\_info **=** lines[0]**.**split()

pleth\_info['frequency'] **=** int(first\_line\_info[2])

last\_three\_lines **=** lines[**-**3:]

pleth\_info['electrical\_activity'] **=** last\_three\_lines[1]**.**replace("#", "")**.**replace("\n", "")

pleth\_info['validity'] **=** last\_three\_lines[2]**.**replace("#", "")**.**replace("\n", "")

**return** pleth\_info

file\_path **=** hea\_file\_path\_test

pleth\_info\_test **=** extract\_pleth\_info(file\_path)

pleth\_info\_test

Out[209]:

{'frequency': 250,

'electrical\_activity': 'Ventricular\_Flutter\_Fib',

'validity': 'True alarm'}

* This funtion take a '.mat' file from the physionet Cardiac Challenge 2015 training database
* Extracts the **PPG Signal** from it
* Converts it into an array

In [210]:

**def** extract\_pleth\_signal(record\_path):

mat\_data **=** scipy**.**io**.**loadmat(record\_path)

signals **=** mat\_data['val']

pleth\_signal **=** signals[2] *# PLETH is the third signal*

**return** pleth\_signal

*# Example case*

pleth\_signal\_test **=** extract\_pleth\_signal(record\_path\_test)

pleth\_signal\_test

Out[210]:

array([11467, 15559, 16025, ..., 17028, 17055, 17078], dtype=int16)

* This is the function that has to be called in the GUI so that an the Cardiac Arrhythmia Prediction can be made from the signal when inputed as an array

In [211]:

**def** predict\_arrhythmia(fs, signal, lstm\_model\_test, class\_labels):

input\_features **=** []

segmented\_signal **=** segment\_signal(signal, 10, fs)

**for** segment **in** segmented\_signal:

segment\_processed **=** bandpass\_filter(segment, lowcut**=**0.05, highcut**=**10, fs**=**fs)

segment\_processed **=** moving\_average(segment\_processed, window\_size**=**3)

segment\_processed **=** remove\_baseline\_wandering(segment\_processed)

**if** **not** any(segment\_processed):

**continue**

features, peaks **=** extract\_ppg\_features(segment\_processed, fs)

**for** peak **in** features:

peak\_array **=** [peak[key] **for** key **in** peak**.**keys()]

input\_features**.**append(peak\_array)

input\_data **=** np**.**array(input\_features)

input\_data **=** input\_data**.**reshape(input\_data**.**shape[0], input\_data**.**shape[1], 1)

raw\_predictions **=** lstm\_model\_test**.**predict(input\_data)

average\_prediction **=** np**.**mean(raw\_predictions, axis**=**0)

prediction\_dict **=** {}

**for** label, percentage **in** zip(class\_labels, average\_prediction):

prediction\_dict[label] **=** f"{percentage**\***100:.2f}%"

**return** prediction\_dict

*# Example Usecase*

fs **=** 250

class\_labels **=** [ 'Bradycardia' , 'Asystole', 'Ventricular Flutter Fib', 'Ventricular Tachycardia' ,'Tachycardia']

prediction\_dict **=** predict\_arrhythmia(fs, pleth\_signal\_test, lstm\_model\_test, class\_labels)

prediction\_dict

16/16 [==============================] - 0s 6ms/step

Out[211]:

{'Bradycardia': '5.24%',

'Asystole': '2.31%',

'Ventricular Flutter Fib': '88.95%',

'Ventricular Tachycardia': '2.24%',

'Tachycardia': '1.26%'}