Case Report

Healthc Inform Res. 2018 July;24(3):242-246.
https://doi.org/10.4258/hir.2018.24.3.242
pISSN 2093-3681 • eISSN 2093-369X

HIR
Healthcare Informatics Research

Construction of an Electrocardiogram Database Including 12 Lead Waveforms

Dahee Chung, PhD[1], Junggu Choi, BE[1], Jong-Hwan Jang, BA[1], Tae Young Kim, BE[1], JungHyun Byun, BS[1], Hojun Park, BS[1], Hong-Seok Lim, MD, PhD[2], Rae Woong Park, MD, PhD[1,3], Dukyong Yoon, MD, PhD[1,3]

[1]Department of Biomedical Informatics, Ajou University School of Medicine, Suwon, Korea; [2]Department of Cardiology, Ajou University School of Medicine, Suwon, Korea; [3]Department of Biomedical Sciences, Ajou University Graduate School of Medicine, Suwon, Korea

Objectives: Electrocardiogram (ECG) data are important for the study of cardiovascular disease and adverse drug reactions. Although the development of analytical techniques such as machine learning has improved our ability to extract useful information from ECGs, there is a lack of easily available ECG data for research purposes. We previously published an article on a database of ECG parameters and related clinical data (ECG-ViEW), which we have now updated with additional 12-lead waveform information. **Methods:** All ECGs stored in portable document format (PDF) were collected from a tertiary teaching hospital in Korea over a 23-year study period. We developed software which can extract all ECG parameters and waveform information from the ECG reports in PDF format and stored it in a database (meta data) and a text file (raw waveform). **Results:** Our database includes all parameters (ventricular rate, PR interval, QRS duration, QT/QTc interval, P-R-T axes, and interpretations) and 12-lead waveforms (for leads I, II, III, aVR, aVL, aVF, V1, V2, V3, V4, V5, and V6) from 1,039,550 ECGs (from 447,445 patients). Demographics, drug exposure data, diagnosis history, and laboratory test results (serum calcium, magnesium, and potassium levels) were also extracted from electronic medical records and linked to the ECG information. **Conclusions:** Electrocardiogram information that includes 12 lead waveforms was extracted and transformed into a form that can be analyzed. The description and programming codes in this case report could be a reference for other researchers to build ECG databases using their own local ECG repository.

Keywords: Electrocardiogram, Waveform, QT Interval, Database, Adverse Drug Reaction

Submitted: June 30, 2018
Revised: July 19, 2018
Accepted: July 24, 2018

Corresponding Author
Dukyong Yoon, MD, PhD
Department of Biomedical Sciences, Ajou University Graduate School of Medicine, 206 World cup-ro, Yeongtong-gu, Suwon 16499, Korea.
Tel: +82-31-219-4476, E-mail: d.yoon.ajou@gmail.com

I. Introduction

Electrocardiogram (ECG) has been widely used to diagnose various cardiovascular diseases including arrhythmia and acute coronary syndrome [1-3] because it is a non-invasive and convenient tool for measuring the continuous wave sequence characterizing the heart activity [2,4].

Information from ECGs is also used to detect a prolonged QT interval, which is one of the life-threatening adverse drug reactions (ADRs). A prolonged QT interval leads to an irregular heart beat and can result in various types of cardiac arrest including ventricular fibrillation, ventricular tachyarrhythmia, Torsades de Pointes, and sudden death [5-7]. Due to its importance as a drug-induced adverse reaction, a prolonged QT interval is strictly monitored and regulated [5].

To address the needs of ECG data analysis, we previously constructed the ECG databases, Electrocardiogram Vigilance with Electronic data Warehouse I (ECG-ViEW I) and ECG-ViEW II, using ECG data measured in a tertiary teaching hospital located in Korea [8,9]. However, these previous versions of the database had limitations because they did not provide the ECG waveform. To overcome these limitations, we designed an update of the ECG database.

II. Case Description

This study was a retrospective review of Electronic Health Records and was approved by the Ajou University Hospital Institutional Review Board (No. AJIRB-MED-MDB-18-075), which also waived the requirement for informed consent.

1. Data Resources and Patient Characteristics

ECG-ViEW I and II covered three data sources: scanned images of paper-based ECGs, ECGs in portable document format (PDF) from the MUSE system (GE Healthcare, Waukesha, WI, USA), and image files stored in the hospital's Electronic Medical Records (EMRs). In contrast, we used only the PDF ECGs from the MUSE system and no images in this study due to the following reasons. The images scanned from paper-based ECGs or image files from the EMRs are saved as pixels images; the quality of information extracted by optical character recognition (OCR) is dependent on the quality of image, and there is no appropriate way to extract waveforms with high accuracy. On the other hand, the quality of data extracted from PDF files from the MUSE system is stable and well controlled. Moreover, because the waveforms in PDF files are saved in scalable vector graphics (SVG) format, it is possible to export the waveforms and maintain the quality of the raw data [10].

2. ECG Data Extraction

An ECG report typically contains both alphanumeric values and waveform graphs (Figure 1). The upper part of the ECG report is a list of alphanumeric values including demographic information, patient ID, evaluation date, ECG parameters, and interpretations (e.g., normal sinus rhythm). Demographic information refers to basic patient information including name, age, sex, and ethnicity. ECG parameters include ventricular rate, PR interval, QRS duration, QT/QTc, and P-R-T axes. The waveform graphs, which typically cover the middle and bottom part of the ECG, are time series of graphs representing the sensor measurement data.

The alphanumeric data were converted from PDF to eXtensible Markup Language (XML) format to increase the accuracy of the parsing results. The main difference between parsing in PDF and XML is related to the handling of irrelevant data. In PDF format, data are saved and parsed according to object type, and deletion of unnecessary data requires careful manual revision. In contrast, the XML format enables conditional parsing, and thus, relevant data can be extracted using automated code. Second, the XML format provides x- and y-coordinates for each piece of data. The axes of the coordinates start from the upper left corner of the ECG report. Thus, alphanumeric information can be extracted based on the position information (i.e., the location of each piece of

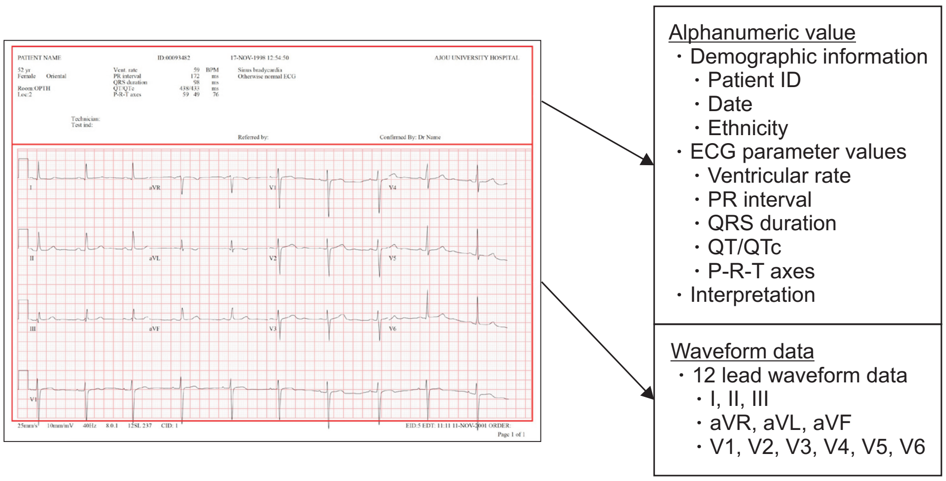

Figure 1. Example of an electrocardiogram (ECG) report. Alphanumeric values (demographic information, ECG parameter values, and interpretation) are located in the upper part of the report. Waveform data are given as time-series graphs with a grid, covering the middle and lower part of the report. One grid unit (1 mm × 1 mm square) corresponds to 0.1 mV × 0.04 seconds.

data on the ECG report).

The part of the PDF file containing the waveform data is stored as metadata of the image in SVG format. To extract the image data, we used INKSCAPE (open-source software, https://inkscape.org) in Linux and converted the waveforms in PDF to SVG format images. The waveforms in the SVG format were processed using svgpathtools from the Python library to classify the parsed data into three categories: path, attribute, and svg_attribute. We used the 'path' data to convert the waveform into a numeric series. 'path' can be considered to be composed of real and imaginary numbers describing the starting and ending points (i.e., a complex plane). After transforming this complex plane into a Cartesian coordinate system (Figure 2A), the starting point of each waveform was adjusted to '(0,0)' (Figure 2B). It was required to reset the baseline of each waveform because the starting point of the waveforms in raw data corresponds to a certain position on the ECG report. Then, we adjusted the values of the time series data to indicate the units of millivolt (mV), which corresponds to 10 times the height of the grid unit (square) used in the ECG report (Figure 2C). Finally, we converted the x- and y-coordinates of the vector images to an equidistant time series similar to that obtained from the sensor (Figure 2D). We set the frequency of data at 500 Hz and identified data points on the waveform using linear interpolation between points with known coordinates (the frequency of the raw data was varied from about 200 to 420 Hz).

There are 13 waveforms in a single ECG report: 3-second strips for each of the 12 leads, plus a 10-second strip, usually for lead II. Each waveform was saved in a separate file in comma-separated value (CSV) format, resulting in 13 waveform CSV files per ECG report. The waveform data were saved as a compressed CSV format using gzip, while their metadata were stored in a database to link the waveform data with the corresponding alphanumeric values from the ECG report as well as with the clinical data from the EMR.

3. Software Tools

We used a Java programming tool to extract the PDF files from the MUSE system, and a Linux-based program (pdftohtml) to convert PDF to the XML format. INKSCAPE for Linux was used to convert PDF to the SVG format. The parsing of XML and SVG files was performed using Python, and the svgpathtools library was used to extract the waveform data. The ElementTree library was used to parse the XML formats. All software tools and codes used during the parsing process are available in Supplementary 1.

4. Data Validation and Quality Control

The accuracy of the ECG data extraction was validated according to the correlation between the extracted and calculated QTc values. The parameter QTc can be calculated based on the QT intervals and RR intervals using Bazett formula. Approximately 99.94% of the extracted QTc values and calculated QTc values matched within ± 2 ms. We assume that the cause of this difference between the two values is due to the process of rounding off. The QT and RR values that we used to calculate QTc were already rounded as integers; thus, there might be some difference from the original

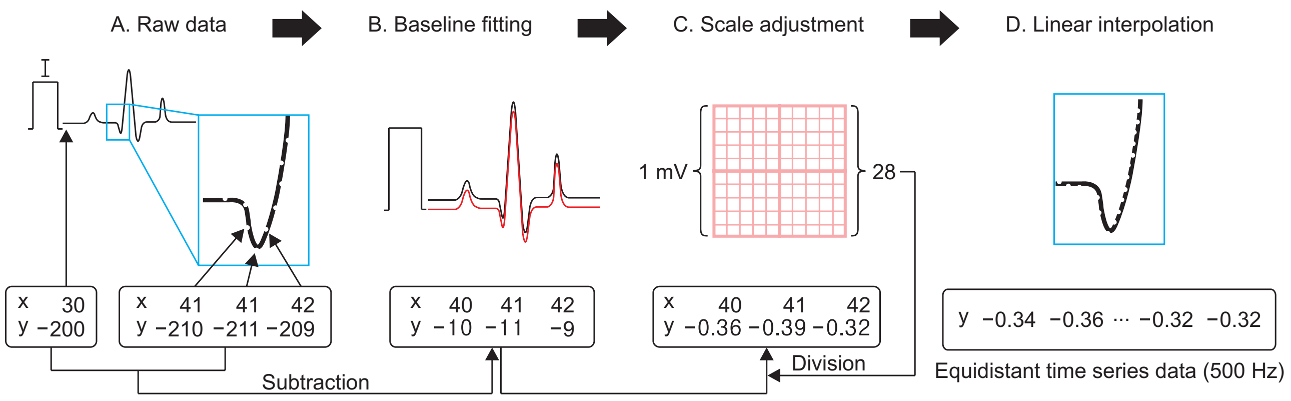

Figure 2. Data transformation process for the waveform data. (A) Raw data in SVG format contains information regarding the exact position of the data point on the electrocardiogram. (B) The start point of all 12-lead waveforms was set to 0. (C) Because 1 mV in the raw data corresponds to y-axis values of 28, we divided all y-axis values by 28 to adjust the scale of the y-axis to mV. (D) Using linear interpolation (500 Hz), vector image data given as x- and y-coordinates were converted into equidistant time series data. Although the resulting waveform data consist of a series of values without timestamps, the timestamps could be calculated by counting the data points from the starting point because the starting point was provided in the file name or database, and one data point corresponds to 1/500 seconds.

value used when calculating QTc in the ECG machine. For the extracted results in which the difference with the calculated QTc value was relatively high (> 2 mm), we manually reviewed the results and confirmed that there was no error in the data extraction process.

Because the frequency of the waveforms was adjusted to 500 Hz, there could be little difference in the converted waveform with the raw data at locations where the time series data point does not coincide with the x- and y-coordinates. To validate the quality of the linear interpolation, we compared the original waveform to the converted waveform for randomly chosen waveforms. The difference was not noticeable shown in Figure 3.

5. Data in the Database

The ECG database contains a total of 1,039,550 ECGs from 447,445 patients (Table 1). The mean follow-up period per person was 717 ± 1,534 days.

6. Software Availability

All programming codes for extracting both the alphanumeric and waveform data are provided in Supplementary 1.

III. Discussion

The update of the ECG databases includes a complete dataset in which the relevant data (ECG values, ECG waveforms, and demographic, diagnosis, medication, laboratory, and any other information related to the hospital visit) are provided for all patients covered by the database. Therefore, the database could be used as a data source in various studies including comprehensive clinical evaluation to determine the potential associations between ECG values or patterns and specific diagnoses, medications, or hospital visit characteristics.

We are currently working on collecting biosignal data from patient monitoring devices from more than 100 beds in an emergency room, intensive care units, and an operating room [11]. All biosignals including ECG lead II, peripheral

Table 1. Summary of demographic and ECG data covered by ECG-ViEW III (n = 447,445)

Characteristic	Value
Number of ECGs according to department	
Outpatient	544,859
Inpatient	190,125
Emergency room visit	304,566
Electrocardiograms	1,039,550
Year of birth	1964.0 ± 21.0
Sex, male	220,173 (49.2)
ECG parameter	
Ventricular rate (bpm)	76.5 ± 19.6
RR interval (ms)	828.2 ± 201.1
PR interval (ms)	149.2 ± 44.9
QRS duration (ms)	91.3 ± 15.9
QT interval (ms)	389.3 ± 45.8
QTc interval (ms)	430.9 ± 34.9
P axis (°)	46.2 ± 26.2
QRS axis (°)	43.6 ± 39.2
T axis (°)	46.7 ± 39.9
Interpretation	
Normal ECG	420,194
Normal sinus rhythm	709,334
Abnormal ECG	348,968
Prolonged QT	50,705

Values are presented as mean ± standard deviation or number (%).

ECG: electrocardiogram; ECG-ViEW: Electrocardiogram Vigilance with Electronic data Warehouse.

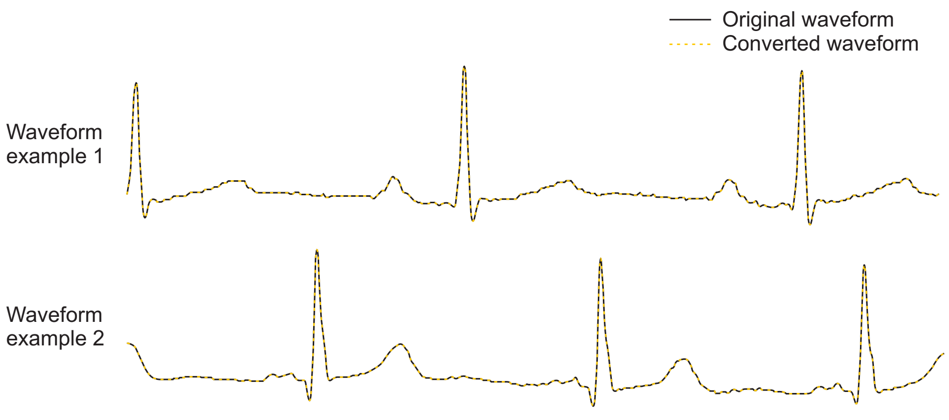

Figure 3. Comparison between the raw waveform and converted waveform data. Due to high-density interpolation, there was no significant difference between the raw data (which are based on x- and y-coordinates) and time series data.

capillary oxygen saturation, respiration, arterial blood pressure, central venous pressure, and end-tidal CO2 data are collected onto a local server. By constructing the ECG database from the ECG reports, we could expand coverage of the biosignal collection into general wards.

The ECG database, described in this article, is one of the largest ECG databases linked to relevant clinical data. This database has integrated all 12 lead ECG waveforms and not just only the numeric parameters of ECG, patient demographics, diagnosis data and drug prescription data. Although the full dataset cannot be made publicly available due to legal restrictions imposed by the Korean government in relation to the Personal Information Protection Act, we expect that the description of the process for constructing the database and the programming codes provided in Supplementary 1 could be a good reference for other researchers to build their own ECG databases using their own local ECG repository.

Conflict of Interest

No potential conflict of interest relevant to this article was reported.

Acknowledgments

This research was supported by a grant of the Korea Health Technology R&D Project through the Korea Health Industry Development Institute (KHIDI), funded by the Ministry of Health & Welfare, Republic of Korea (No. HI16C0982, HI17C0970, and HI6C0992).

Supplementary Materials

Supplementary materials can be found via http://doi.org.10.4258/hir.2018.24.3.242. Supplement 1. Software for extracting information from an ECG report in PDF format.

References

1. Ovreiu M, Simon D. Biogeography-based optimization of neuro-fuzzy system parameters for diagnosis of cardiac disease. Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation; 2010 Jul 7-11; Portland, OR. p. 1235-42.
2. Martis RJ, Acharya UR, Adeli H. Current methods in electrocardiogram characterization. Comput Biol Med 2014;48:133-49.
3. O'Neil BJ, Hoekstra J, Pride YB, Lefebvre C, Diercks D, Frank Peacock W, et al. Incremental benefit of 80-lead electrocardiogram body surface mapping over the 12-lead electrocardiogram in the detection of acute coronary syndromes in patients without ST-elevation myocardial infarction: results from the Optimal Cardiovascular Diagnostic Evaluation Enabling Faster Treatment of Myocardial Infarction (OCCULT MI) trial. Acad Emerg Med 2010;17(9):932-9.
4. Laguna P, Jane R, Caminal P. Automatic detection of wave boundaries in multilead ECG signals: validation with the CSE database. Comput Biomed Res 1994;27(1):45-60.
5. Lynch DR Jr, Washam JB, Newby LK. QT interval prolongation and torsades de pointes in a patient undergoing treatment with vorinostat: a case report and review of the literature. Cardiol J 2012;19(4):434-8.
6. Li XB, Tang YL, Zheng W, Wang CY, de Leon J. QT interval prolongation associated with intramuscular ziprasidone in Chinese patients: a case report and a comprehensive literature review with meta-analysis. Case Rep Psychiatry 2014;2014:489493.
7. Tarapues M, Cereza G, Arellano AL, Montane E, Figueras A. Serious QT interval prolongation with ranolazine and amiodarone. Int J Cardiol 2014;172(1):e60-1.
8. Park MY, Yoon D, Choi NK, Lee J, Lee K, Lim HS, et al. Construction of an open-access QT database for detecting the proarrhythmia potential of marketed drugs: ECG-ViEW. Clin Pharmacol Ther 2012;92(3):393-6.
9. Kim YG, Shin D, Park MY, Lee S, Jeon MS, Yoon D, et al. ECG-ViEW II, a freely accessible electrocardiogram database. PLoS One 2017;12(4):e0176222.
10. Ortigosa N, Gimenez VM. Raw data extraction from electrocardiograms with Portable Document Format. Comput Methods Programs Biomed 2014;113(1):284-9.
11. Yoon D, Lee S, Kim TY, Ko J, Chung WY, Park RW. System for collecting biosignal data from multiple patient monitoring systems. Healthc Inform Res 2017;23(4):333-7.

Supplementary Materials

Healthc Inform Res. 2018 July;24(3):242-246.
https://doi.org/10.4258/hir.2018.24.3.242
pISSN 2093-3681 • eISSN 2093-369X

HIR
Healthcare Informatics Research

Software for extracting information from an ECG report of PDF format

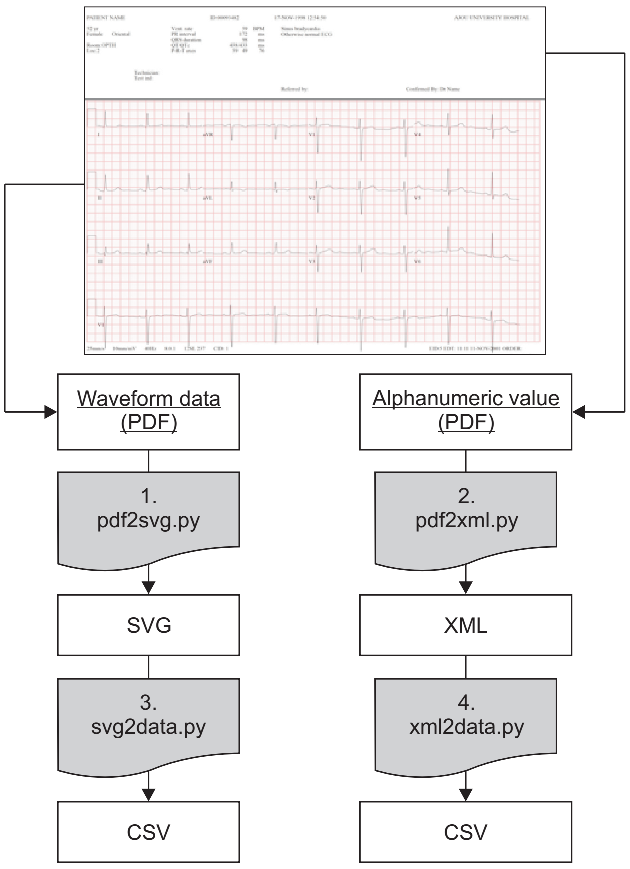

Figure 1. A flow chart for the Python code.

1. pdf2svg.py

A code that converts a PDF file into a SVG file.
Input: The folder where the PDF file is located. A list of all the files in that folder will be inputted into the execute function.
Output: Folder where converted SVG files will be stored. The SVG file will be created in that folder.
Because multiprocessing is used, the speed may vary depending on your CPU configuration.

```
###This code is for converting PDF file to SVG file.

##import library for converting ##
from multiprocessing import Process
from multiprocessing import Pool, cpu_count
from os import listdir
import time
import os

## execute function for converting ##
def execute(filelist):
    os.chdir("/directory/pdf")
    command = "inkscape -z -f /directory/pdf/"+filelist+" -l /directory/svg/"+filelist+".svg"
    os.system(command)
    os.chdir("/directory/pdf")
    # directory is the physical directory where the file is located



if __name__ == '__main__':
    print("Start convert ECG data!")

    os.chdir("/directory/")
    search_directory = "pdf"
    filelist = listdir(search_directory)

    processor = cpu_count()
    proc = os.getpid()

    print("proc_id",proc)
    print(os.fork())
    print("Number of processor:",processor)

    print("Number_of_pdf_file :", len(filelist))

    pool = Pool(processes = cpu_count())

    startTime = int(time.time())
    print(startTime)
    pool.map(execute, filelist)
    endTime = int(time.time())
    print("Total converting time", (endTime - startTime))
```

2. pdf2xml.py

A code that converts a PDF file into an XML file.
Input: The folder where the PDF file is located. A list of all the files in that folder will be inputted into the execute function.
Output: Folder where converted XML files will be stored. The XML file will be created in that folder.
Because multiprocessing is used, the speed may vary depending on your CPU configuration.

```
###This code is for converting PDF to XML

##import library for converting ##
from multiprocessing import Process
from multiprocessing import Pool, cpu_count
from os import listdir
import time
import os

## execute function for converting ##
def execute(filelist):
  os.chdir("/directory/pdf")
  command = "pdftohtml -xml /directory/pdf/"+filelist+" /directory/xml/"+filelist+".xml"
  os.system(command)
  os.chdir("/directory/pdf")
  # directory is the physical directory where the file is located



if __name__ == '__main__':
  print("Start convert ECG data!")

  os.chdir("/directory/")
  search_directory = "pdf"
  filelist = listdir(search_directory)

  processor = cpu_count()
  proc = os.getpid()

  print("proc_id",proc)
  print(os.fork())
  print("Number of processor:",processor)

  print("Number_of_pdf_file :", len(filelist))

  pool = Pool(processes = cpu_count())

  startTime = int(time.time())
  print(startTime)
  pool.map(execute, filelist)
  endTime = int(time.time())
  print("Total converting time", (endTime - startTime))
```

3. svg2data.py

A code converts an SVG file into a CSV file containing the waveform data.
Input: SVG file.
This code includes the following functions: (1) extract the waveform data by parsing the SVG file, (2) change the coordinates to Cartesian coordinates, (3) adjust the baseline and scale, and (4) linear interpolation
Because multiprocessing is used, the speed may vary depending on your CPU configuration.

```
###This code is for converting SVG to data(waveform)

## Module import (for parsing and multiprocessing)
from multiprocessing import Process
from multiprocessing import Pool, cpu_count
from svgpathtools import svg2paths2
from svgpathtools import wsvg
from os import listdir
import matplotlib.pyplot as plt
import scipy.interpolate
import scipy as sp
import pandas as pd
import numpy as np
import cmath
import os
import sys
import re
import hashlib
import time


### Define functions for converting
def pol2cart(polar):
  rho = polar[0]
  phi = polar[1]
  x = rho * np.cos(phi)
  y = rho * np.sin(phi)
  return(x, y)

def pars_info(file):
  paths, attributes, svg_attributes = svg2paths2(file)
  pth_tmp = paths[411:]

  return pth_tmp

def chang_corrd(pth_tmp):
  lst_tmp = []
  n = 0
  for i in pth_tmp:
    #print(i)
    path_tmp = [pth_tmp[n][0::2]]
```

```
        lst_tmp = lst_tmp + path_tmp
        n = n + 1
        path_tmp = []

    return lst_tmp

def corrd2data(lst_tmp):
    x_list = []
    y_list = []
n = 0
    for i in lst_tmp:
        x = []
        y = []
        for i[n] in i:

            #[0] is first part!
            real_num_s = i[n][0].real
            imag_num_s = i[n][0].imag

            #[1] is end part!
            real_num_e = i[n][1].real
            imag_num_e = i[n][1].imag

            #convert to complex number!
            start_cn = complex(real_num_s , imag_num_s)
            end_cn = complex(real_num_e, imag_num_e)

           #convert to polar coordinates!
            start_pol = cmath.polar(start_cn)
            end_pol = cmath.polar(end_cn)

            #convert to cartesian coordinate!
            start_poi = pol2cart(start_pol)
            end_poi = pol2cart(end_pol)

            start_poi_x = [start_poi[0]]
            end_poi_x = [end_poi[0]]
            start_poi_y = [start_poi[1]]
            end_poi_y = [end_poi[1]]

            #collect!
            x = x + start_poi_x + end_poi_x
            start_poi_x = []
            end_poi_x = []

            y = y + start_poi_y + end_poi_y
            start_poi_y = []
            end_poi_y = []
        x_list.append(x[2:])
        y_list.append(y[2:])

    return x_list, y_list

####Convert file name for DB
# 1)Parsing patient ID from name of pdf file and anonymous!
def gt_id(name):
    p_id = re.findall('ID=\d+',name)
    p_id = ''.join(p_id).replace("=","")
    hashSHA = hashlib.sha256()
    hashSHA.update(p_id)
    hexSHA256 = hashSHA.hexdigest()
    p_id= hexSHA256[0:20]

    return p_id

# 2)Parsing date from file name
def gt_date(name):
    date = re.findall('Date=\d+-\d+-\d+',name)
    date = ''.join(date).replace('Date=','').split("-")
    date[0],date[1],date[2] = date[2],date[1],date[0]
    date = ''.join(date)

    return date

# 3)Parsing time and merge with date for filename
def gt_time(name):
    time = re.findall('Time=\d+%\d+\w+%\d+\w\d+',name)
    time = ''.join(time)
    time = time.replace('Time=','').replace('%3a','')

    return time

# 4)Parsing end number from file name
def gt_endnum(name):
    endnum = re.findall('num\d+',name)
    endnum = ''.join(endnum)

    return endnum

### change Hz of ECG data!(500Hz/s)
def inter(x_list,y_list):
    #ytmp = [float(i) for i in y_list]
    #xtmp = [float(i) for i in x_list]
    y_array = np.array(y_list)
    x_array = np.array(x_list)
    new_length = 1500
    new_x = np.linspace(x_array.min(), x_array.max(),new_length)
    new_y = sp.interpolate.interp1d(x_array,y_array)(new_x)
    x_list = new_x.tolist()
    y_list = new_y.tolist()

    return x_list, y_list

def inter2(x_list,y_list):
    #ytmp = [float(i) for i in y_list]
    #xtmp = [float(i) for i in x_list]
    y_array = np.array(y_list)
    x_array = np.array(x_list)
    new_length = 5000
    new_x = np.linspace(x_array.min(), x_array.max(),new_length)
    new_y = sp.interpolate.interp1d(x_array,y_array)(new_x)
    x_list = new_x.tolist()
    y_list = new_y.tolist()

    return x_list, y_list



### fixing data to fit baseline of ECG!
def fix_data(x_list, y_list):
    ## calculate final y value of baseline!
    f_bl = y_list[0][-1]
    s_bl = y_list[1][-1]
    t_bl = y_list[2][-1]
    fr_bl = y_list[15][-1]

    ## first, fix y value of first row!
    dis_fn3 = f_bl - y_list[3][0]
    if dis_fn3 > 0:
        y_list[3] = list(map(lambda y: y + dis_fn3, y_list[3]))
        x_list[3], y_list[3] = inter(x_list[3],y_list[3])
        y_list[3] = list(map(lambda y: y - y_list[3][0], y_list[3]))
    else:
        y_list[3] = list(map(lambda y: y - dis_fn3, y_list[3]))
        x_list[3], y_list[3] = inter(x_list[3],y_list[3])
        y_list[3] = list(map(lambda y: y - y_list[3][0], y_list[3]))

    dis_fn6 = f_bl - y_list[6][0]
    if dis_fn6 > 0:
        y_list[6] = list(map(lambda y: y + dis_fn6, y_list[6]))
        x_list[6], y_list[6] = inter(x_list[6],y_list[6])
        y_list[6] = list(map(lambda y: y - y_list[6][0], y_list[6]))
    else:
        y_list[6] = list(map(lambda y: y - dis_fn6, y_list[6]))
        x_list[6], y_list[6] = inter(x_list[6],y_list[6])
        y_list[6] = list(map(lambda y: y - y_list[6][0], y_list[6]))

    dis_fn9 = f_bl - y_list[9][0]
    if dis_fn9 > 0:
        y_list[9] = list(map(lambda y: y + dis_fn9, y_list[9]))
        x_list[9], y_list[9] = inter(x_list[9],y_list[9])
        y_list[9] = list(map(lambda y: y - y_list[9][0], y_list[9]))
    else:
        y_list[9] = list(map(lambda y: y - dis_fn9, y_list[9]))
        x_list[9], y_list[9] = inter(x_list[9],y_list[9])
        y_list[9] = list(map(lambda y: y - y_list[9][0], y_list[9]))

    dis_fn12 = f_bl - y_list[12][0]
    if dis_fn12 > 0:
        y_list[12] = list(map(lambda y: y + dis_fn12, y_list[12]))
        x_list[12], y_list[12] = inter(x_list[12],y_list[12])
        y_list[12] = list(map(lambda y: y - y_list[12][0], y_list[12]))
    else:
        y_list[12] = list(map(lambda y: y - dis_fn12, y_list[12]))
        x_list[12], y_list[12] = inter(x_list[12],y_list[12])
        y_list[12] = list(map(lambda y: y - y_list[12][0], y_list[12]))

    ## second, fix y value of second row!
    dis_fn4 = s_bl - y_list[4][0]
    if dis_fn4 > 0:
        y_list[4] = list(map(lambda y: y + dis_fn4, y_list[4]))
        x_list[4], y_list[4] = inter(x_list[4],y_list[4])
        y_list[4] = list(map(lambda y: y - y_list[4][0], y_list[4]))
    else:
        y_list[4] = list(map(lambda y: y - dis_fn4, y_list[4]))
        x_list[4], y_list[4] = inter(x_list[4],y_list[4])
        y_list[4] = list(map(lambda y: y - y_list[4][0], y_list[4]))

    dis_fn7 = s_bl - y_list[7][0]
    if dis_fn7 > 0:
        y_list[7] = list(map(lambda y: y + dis_fn7, y_list[7]))
        x_list[7], y_list[7] = inter(x_list[7],y_list[7])
        y_list[7] = list(map(lambda y: y - y_list[7][0], y_list[7]))
    else:
        y_list[7] = list(map(lambda y: y - dis_fn7, y_list[7]))
        x_list[7], y_list[7] = inter(x_list[7],y_list[7])
        y_list[7] = list(map(lambda y: y - y_list[7][0], y_list[7]))

    dis_fn10 = s_bl - y_list[10][0]
    if dis_fn10 > 0:
        y_list[10] = list(map(lambda y: y + dis_fn10, y_list[10]))
        x_list[10], y_list[10] = inter(x_list[10],y_list[10])
        y_list[10] = list(map(lambda y: y - y_list[10][0], y_list[10]))
    else:
        y_list[10] = list(map(lambda y: y - dis_fn10, y_list[10]))
        x_list[10], y_list[10] = inter(x_list[10],y_list[10])
        y_list[10] = list(map(lambda y: y - y_list[10][0], y_list[10]))

    dis_fn13 = s_bl - y_list[13][0]
    if dis_fn13 > 0:
        y_list[13] = list(map(lambda y: y + dis_fn13, y_list[13]))
        x_list[13], y_list[13] = inter(x_list[13],y_list[13])
        y_list[13] = list(map(lambda y: y - y_list[13][0], y_list[13]))
    else:
        y_list[13] = list(map(lambda y: y - dis_fn13, y_list[13]))
        x_list[13], y_list[13] = inter(x_list[13],y_list[13])
        y_list[13] = list(map(lambda y: y - y_list[13][0], y_list[13]))

    ## thrid, fix y value of third row!
    dis_fn5 = t_bl - y_list[5][0]
    if dis_fn5 > 0:
        y_list[5] = list(map(lambda y: y + dis_fn5, y_list[5]))
        x_list[5], y_list[5] = inter(x_list[5],y_list[5])
        y_list[5] = list(map(lambda y: y - y_list[5][0], y_list[5]))
    else:
        y_list[5] = list(map(lambda y: y - dis_fn5, y_list[5]))
        x_list[5], y_list[5] = inter(x_list[5],y_list[5])
        y_list[5] = list(map(lambda y: y - y_list[5][0], y_list[5]))

    dis_fn8 = t_bl - y_list[8][0]
    if dis_fn8 > 0:
        y_list[8] = list(map(lambda y: y + dis_fn8, y_list[8]))
        x_list[8], y_list[8] = inter(x_list[8],y_list[8])
        y_list[8] = list(map(lambda y: y - y_list[8][0], y_list[8]))
    else:
        y_list[8] = list(map(lambda y: y - dis_fn8, y_list[8]))
        x_list[8], y_list[8] = inter(x_list[8],y_list[8])
        y_list[8] = list(map(lambda y: y - y_list[8][0], y_list[8]))

    dis_fn11 = t_bl - y_list[11][0]
    if dis_fn11 > 0:
        y_list[11] = list(map(lambda y: y + dis_fn11, y_list[11]))
        x_list[11], y_list[11] = inter(x_list[11],y_list[11])
        y_list[11] = list(map(lambda y: y - y_list[11][0], y_list[11]))
    else:
        y_list[11] = list(map(lambda y: y - dis_fn11, y_list[11]))
        x_list[11], y_list[11] = inter(x_list[11],y_list[11])
        y_list[11] = list(map(lambda y: y - y_list[11][0], y_list[11]))

    dis_fn14 = t_bl - y_list[14][0]
```

```
    if dis_fn14 > 0:
        y_list[14] = list(map(lambda y: y + dis_fn14, y_list[14]))
        x_list[14], y_list[14] = inter(x_list[14],y_list[14])
        y_list[14] = list(map(lambda y: y - y_list[14][0], y_list[14]))
    else:
        y_list[14] = list(map(lambda y: y - dis_fn14, y_list[14]))
        x_list[14], y_list[14] = inter(x_list[14],y_list[14])
        y_list[14] = list(map(lambda y: y - y_list[14][0], y_list[14]))

    ## fourth, fix y value of fourth row!
    dis_fn16 = fr_bl - y_list[16][0]
    if dis_fn16 > 0:
        y_list[16] = list(map(lambda y: y + dis_fn16, y_list[16]))
        x_list[16], y_list[16] = inter2(x_list[16],y_list[16])
        y_list[16] = list(map(lambda y: y - y_list[16][0], y_list[16]))
    else:
        y_list[16] = list(map(lambda y: y - dis_fn16, y_list[16]))
        x_list[16], y_list[16] = inter2(x_list[16],y_list[16])
        y_list[16] = list(map(lambda y: y - y_list[16][0], y_list[16]))

    return x_list, y_list


#Adjust scale of ECG data!
def adj(x_list,y_list):
    candid_1 = y_list[0]
    max_value_1 = max(candid_1)
    min_value_1 = min(candid_1)
    base_1 = abs(max_value_1 - min_value_1)

    y_list[3] = [x/base_1 for x in y_list[3]]
    y_list[6] = [x/base_1 for x in y_list[6]]
    y_list[9] = [x/base_1 for x in y_list[9]]
    y_list[12] = [x/base_1 for x in y_list[12]]

    candid_2 = y_list[1]
    max_value_2 = max(candid_2)
    min_value_2 = min(candid_2)
    base_2 = abs(max_value_2 - min_value_2)

    y_list[4] = [x/base_2 for x in y_list[4]]
    y_list[7] = [x/base_2 for x in y_list[7]]
    y_list[10] = [x/base_2 for x in y_list[10]]
    y_list[13] = [x/base_2 for x in y_list[13]]

    candid_3 = y_list[2]
    max_value_3 = max(candid_3)
    min_value_3 = min(candid_3)
    base_3 = abs(max_value_3 - min_value_3)

    y_list[5] = [x/base_3 for x in y_list[5]]
    y_list[8] = [x/base_3 for x in y_list[8]]
    y_list[11] = [x/base_3 for x in y_list[11]]
    y_list[14] = [x/base_3 for x in y_list[14]]

    candid_4 = y_list[15]
    max_value_4 = max(candid_4)
    min_value_4 = min(candid_4)
    base_4 = abs(max_value_4 - min_value_4)

    y_list[15] = [x/base_4 for x in y_list[15]]

    return x_list, y_list

#make each dataframe for each part of ECG!
def mk_pECG(x_list,y_list):
    p_df_3 = pd.DataFrame(y_list[3])
    p_df_4 = pd.DataFrame(y_list[4])
    p_df_5 = pd.DataFrame(y_list[5])
    p_df_6 = pd.DataFrame(y_list[6])
    p_df_7 = pd.DataFrame(y_list[7])
    p_df_8 = pd.DataFrame(y_list[8])
    p_df_9 = pd.DataFrame(y_list[9])
    p_df_10 = pd.DataFrame(y_list[10])
    p_df_11 = pd.DataFrame(y_list[11])
    p_df_12 = pd.DataFrame(y_list[12])
    p_df_13 = pd.DataFrame(y_list[13])
    p_df_14 = pd.DataFrame(y_list[14])
    p_df_16 = pd.DataFrame(y_list[16])


    return p_df_3, p_df_4, p_df_5, p_df_6, p_df_7, p_df_8, p_df_9, p_df_10, p_df_11, p_df_12, p_df_13, p_df_14, p_df_16


def execute(file):
    try:
        #first, change file to data list of ECG!
        pth_tmp = pars_info(file)
        lst_tmp = chang_corrd(pth_tmp)
        x_list, y_list = corrd2data(lst_tmp)
        x_list, y_list = fix_data(x_list, y_list)
        x_list, y_list = adj(x_list, y_list)


        #second, parsing filename and change saving filename!
        name = str(file)
        p_id= gt_id(name)
        date = gt_date(name)
        time = gt_time(name)
        endnum = gt_endnum(name)

        ###third, make subfolder for each patient!
        os.getcwd()

        try:
          # 1)subfolder each patient_id!
          os.chdir("/directory/csv")
          os.mkdir(p_id)

        except OSError:
          os.chdir("/directory/csv"+"/"+p_id)

          try:
            os.mkdir(date)
            os.chdir("/directory/csv"+"/"+p_id+"/"+date)

          except OSError:
            os.chdir("/directory/csv"+"/"+p_id+"/"+date)

              p_df_3,p_df_4,p_df_5, p_df_6,p_df_7,p_df_8,p_df_9, p_df_10,p_df_11,p_df_12,p_df_13,p_df_14, p_df_16 = mk_
pECG(x_list,y_list)

            #change to csv file for each part_ECG!
            p_df_3.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGI"+".gz",index=False,header=False,compression='
gzip')
            p_df_4.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGII"+".gz",index=False,header=False,compression=
'gzip')
            p_df_5.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGIII"+".gz",index=False,header=False,compression
='gzip')
            p_df_6.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGaVR"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_7.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGaVL"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_8.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGaVF"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_9.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGV1"+".gz",index=False,header=False,compression
='gzip')
            p_df_10.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGV2"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_11.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGV3"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_12.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGV4"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_13.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGV5"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_14.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGV6"+".gz",index=False,header=False,compressio
n='gzip')
            p_df_16.to_csv(date + time +"_" + date +  str(int(time)+10)+"_" +"ECGII"+".gz",index=False,header=False,compressio
n='gzip')




            else:

                    p_df_3,p_df_4,p_df_5, p_df_6,p_df_7,p_df_8,p_df_9, p_df_10,p_df_11,p_df_12,p_df_13,p_df_14, p_df_16 = mk_
pECG(x_list,y_list)

            #change to csv file for each part_ECG!
            p_df_3.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGI"+".gz",index=False,header=False,compression='
gzip')
            p_df_4.to_csv(date + time +"_" + date +  str(int(time)+3)+ "_" +"ECGII"+".gz",index=False,header=False,compression=
'gzip')

        #exception handling
        except IndexError:
            print(file)
            os.system("mv /directory/svg/"+file+" /directory/unusual_svg/")
            None




if __name__ == '__main__':
        print("Start converting ECG data!")

        os.chdir("/directory/")
        search_directory = "svg"
        filelist = listdir(search_directory)

        os.chdir("/directory/svg")
        print("Number_of_svg_file :",len(filelist))
        processor = cpu_count()
        pool = Pool(processes = cpu_count())
        print("Number of processor:",processor)
        startTime = int(time.time())

        pool.map(execute, filelist)

        endTime = int(time.time())
        print("Total converting time", (endTime - startTime))
```

4. xml2data.py

A code converts the XML file into a CSV file containing alphanumeric data.
Input: XML file.
Output: The alphanumeric data contained in the XML file is classified according to the item and stored in a CSV file

```
###This code is for converting XML to data(alphanumeric)

## Module import (for parsing and multiprocessing)
from multiprocessing import Process
from multiprocessing import Pool, cpu_count
import multiprocessing
import xml.etree.cElementTree as ET
from itertools import chain
from collections import defaultdict
import pandas as pd
from os import listdir
import re
import time
import os
import signal
import numpy as np
import sys



### Define functions for converting
## Parsing and collect element in XML file!
def par_elem(i):
    tree = ET.ElementTree(file = i)
    inf = []
    for elem in tree.iter():
        elem.attrib['text'] = elem.text
        y = [elem.attrib]
        inf = inf + y
    inf_tmp = inf[4:-2]

    return inf_tmp,i

def par_elem_2(i):
    tree = ET.ElementTree(file = i)
    inf = []
    for elem in tree.iter():
        x = [[elem.tag, elem.attrib, elem.text]]
        inf = inf + x
    inf_tmp_2 = inf[4:-2]

    return inf_tmp_2

#Collect element of XML file by each row!
```

```
def col_elem(inf_tmp,filename):
    all_dict_2 = []
    all_dict = []
    for n in inf_tmp:
        top_info = n['top']
        text_info = n['text']
        left_info = n['left']

        dic_tmp = {top_info : text_info}
        dic_tmp_2 = {left_info : text_info}

        dic_tmp = [dic_tmp]
        dic_tmp_2 = [dic_tmp_2]

        all_dict = all_dict + dic_tmp
        all_dict_2 = all_dict_2 + dic_tmp_2

        dic_tmp =[]
        dic_tmp_2 =[]

        super_dict = defaultdict(set)


        for d in all_dict:
            for k, v in d.items():
                super_dict[k].add(v)


    r_40 = super_dict['40']
    l_40 = list(r_40)
    r_67 = super_dict['67']
    l_67 = list(r_67)
    r_81 = super_dict['81']
    l_81 = list(r_81)
    r_94 = super_dict['94']
    l_94 = list(r_94)
    r_95 = super_dict['95']
    l_95 = list(r_95)
    r_108 = super_dict['108']
    l_108 = list(r_108)
    r_109 = super_dict['109']
    l_109 = list(r_109)
    r_122 = super_dict['122']
    l_122 = list(r_122)
    r_123 = super_dict['123']
    l_123 = list(r_123)

    return l_40,l_67, l_81, l_94,l_95,l_108,l_109,l_122,l_123
```

```
#Find suitable value in elements!
def find_values(l_40, l_67, l_81, l_94, l_95, l_108, l_109, l_122, l_123):
    #Use regex
    pattern1 = r'\d+-\w+-\d+  \d\d:\d\d:\d\d'
    pattern2 = r'\d+'
    pattern3 = r'\d+/\d+'
    pattern4 = r'\w+, \w+'

    ##'40' row!
    # 1)patient ID!
    p_id = [ x for x in l_40 if "ID" in x ]
    if len(p_id) > 1:
        p_id = [ x for x in p_id if "ID:" in x ]
    p_id = ''.join(p_id).replace('ID:','')
    # 2)Patient name!
    p_name = [ x for x in l_40 if re.findall(pattern4, x)]
    p_name = ''.join(p_name)
    # 3)Hospital name!
    hp_name = [ x for x in l_40 if "AJOU" in x ]
    hp_name = ''.join(hp_name)
    # 4)Date!
    date = [ x for x in l_40 if re.findall(pattern1, x)]
    date = ''.join(date)

    ##'67' row!
    # 1)Patient Age!
    p_yr = [ x for x in l_67 if 'yr' in x ]
    p_yr = ''.join(p_yr)
    p_yr = re.sub(r'\d+-\w+-\d+','',p_yr)
    p_yr = p_yr.replace('(','').replace(')','').replace('yr','')
    # 2)Vent.rate!
    vr_u = [ x for x in l_67 if 'BPM' in x ]
    vr_vl = [ x for x in l_67 if re.findall(pattern2, x) ]
    vr_vl = [ x for x in vr_vl if "yr" not in x ]
    vr_vl = [ x for x in vr_vl if "inus" not in x ]
    vr_vl = [ x for x in vr_vl if "bradycardia" not in x ]
    vr_vl = [ x for x in vr_vl if "Atrial " not in x ]
    vr_vl = [ x for x in vr_vl if "atrial" not in x ]
    vr_vl = [ x for x in vr_vl if "Mon" not in x ]
    vr_vl = [ x for x in vr_vl if "mon" not in x ]
    vr_vl = [ x for x in vr_vl if "days" not in x ]
    vr_vl = [ x for x in vr_vl if "complexes" not in x ]
    vr_vl = [ x for x in vr_vl if "ATRIAL" not in x ]
    vr_vl = [ x for x in vr_vl if "Unusual" not in x ]
    vr_vl = [ x for x in vr_vl if "-" not in x ]
    vr_vl = [ x for x in vr_vl if "AV BLOCK" not in x ]
    vr_vl = [ x for x in vr_vl if "AV" not in x ]
    vr_vl = [ x for x in vr_vl if "inversion" not in x ]
    vr_vl = [ x for x in vr_vl if "ECG" not in x ]
    vr_vl = [ x for x in vr_vl if "RATIO" not in x ]
    vr_vl = [ x for x in vr_vl if "hrs" not in x ]
    vr_vl = [ x for x in vr_vl if "wks" not in x ]
    vr_vl = [ x for x in vr_vl if "LEAD" not in x ]
    vr_vl = [ x for x in vr_vl if "*" not in x ]
    vr_vl = [ x for x in vr_vl if "waves" not in x ]
    vr_vl = [ x for x in vr_vl if "CHO" not in x ]
    vr_vl = [ x for x in vr_vl if "GX" not in x ]
    vr_vl = [ x for x in vr_vl if "$" not in x ]
    vr_vl = [ x for x in vr_vl if "SINUS" not in x ]
    vr_vl = [ x for x in vr_vl if "LEADS" not in x ]
    vent_rate = vr_vl
    if len(vent_rate) > 1:
        vent_rate = vent_rate[1]
    vent_rate = ''.join(vent_rate)
    # 3)Diagnosis in first row!
    dia1 = [item for item in l_67 if len(item)>=9]
    dia1 = [ x for x in dia1 if "Vent" not in x ]
    dia1 = [ x for x in dia1 if "yr" not in x ]
    dia1 = [x for x in dia1 if "Confirmed by" not in x]
    dia1 = [x for x in dia1 if "Reconfirmed by" not in x]



    ##'81' row!
    # 1)Sex of patient
    p_sex = [ x for x in l_81 if 'ale' in x ]
    p_sex = ''.join(p_sex)
    # 2)ethnicity of patient
    p_eth = [ x for x in l_81 if 'Oriental' in x ]
    p_eth_2 = [ x for x in l_81 if 'Asian' in x ]
    p_ethn = p_eth + p_eth_2
    p_ethn = ''.join(p_ethn)
    # 3)PR interval
    pr_i_u = [ x for x in l_81 if 'ms' in x ]
    pr_i_vl = [ x for x in l_81 if re.findall(pattern2, x)]
    pr_i_vl = [ x for x in pr_i_vl if "RSR' or QR pattern " not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "Atrial" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "ATRIAL" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "atrial" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "Increased" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "R/S" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "Sinus" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "sinus" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "rhythm" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "bundle" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "branch" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "because" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "and" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "by" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "RSR'" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "-" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "AV" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "mon" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "RATIO" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "QTc" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "QT" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "ms" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "r/s" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "CONSIDER" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "ventricular" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "RATIO" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "EARLY" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "PROBABLE" not in x ]
    pr_i_vl = [ x for x in pr_i_vl if "LEAD" not in x ]
    pr_interval = pr_i_vl
    pr_interval = ''.join(pr_interval)
    # 4)Diagnosis in second row!
    dia2 = [item for item in l_81 if len(item)>=9]
    dia2 = [ x for x in dia2 if "interval" not in x ]
    dia2 = [x for x in dia2 if "Confirmed by" not in x]
    dia2 = [x for x in dia2 if "Reconfirmed by" not in x]



    ##'94' row!
    # 1)Patient Height & Weight
    p_heigt = [ x for x in l_94 if 'cm' in x ]
    p_weight = [ x for x in l_94 if 'kg' in x ]
    p_heigt = ''.join(p_heigt)
    p_weight = ''.join(p_weight)
    p_heigt = str(p_heigt)
    p_weight = str(p_weight)
    p_heigt = p_heigt.replace('cm','')
    p_weight = p_weight.replace('kg','')
    # 2)QRS duration
    qrs_d_u = [ x for x in l_94 if 'ms' in x ]
    qrs_d_vl = [ x for x in l_94 if re.findall(pattern2, x)]
    qrs_d_vl = [ x for x in qrs_d_vl if "cm" not in x ]
    qrs_d_vl = [ x for x in qrs_d_vl if "kg" not in x ]
    qrs_duration = qrs_d_vl
    qrs_duration = ''.join(qrs_duration)
    # 3)Diagnosis in third row!
```

```
dia3 = [item for item in l_94 if len(item)>=9]
dia3 = [ x for x in dia3 if "duration" not in x ]
dia3 = [x for x in dia3 if "Confirmed by" not in x]
dia3 = [x for x in dia3 if "Reconfirmed by" not in x]



## top='95' row!(this row is unusual row for diagnosis information)
dia4 = [item for item in l_95 if len(item)>=10]
dia4 = [x for x in dia4 if "Confirmed by" not in x]
dia4 = [x for x in dia4 if "Reconfirmed by" not in x]

##'108' row!
# 1)QT/QTc
qtqtc_u = [ x for x in l_108 if 'ms' in x ]
qtqtc_vl = [ x for x in l_108 if re.findall(pattern3, x)]
qtqtc_vl_tmp = ''.join(qtqtc_vl)
qt_vl = [i.split('/', 1)[0] for i in qtqtc_vl]
qtc_vl = [i.split('/', 1)[1] for i in qtqtc_vl]
qt = qt_vl
qtc = qtc_vl
qt = ''.join(qt)
qtc = ''.join(qtc)
# 2)Room information
room_info  = [ x for x in l_108 if 'Room' in x ]
room_info = ''.join(room_info).replace('Room:','')
# 3)Diagnosis in fourth row!
dia5 = [item for item in l_108 if len(item)>=9]
dia5 = [ x for x in dia5 if "Room" not in x ]
dia5 = [x for x in dia5 if "Confirmed by" not in x]
dia5 = [x for x in dia5 if "Reconfirmed by" not in x]

##'109' row!(this row also unusual row for diagnosis information)
dia6 = [item for item in l_109 if len(item)>=10]
dia6 = [x for x in dia6 if "Confirmed by" not in x]
dia6 = [x for x in dia6 if "Reconfirmed by" not in x]

##'122' row!
# 1)Location information
loc_info = [ x for x in l_122 if "Loc" in x ]
loc_info = ''.join(loc_info).replace('Loc:','')
# 2) Diagnosis information in fifth row!
dia7 = [item for item in l_122 if len(item)>=9]
dia7 = [ x for x in dia7 if "P-R-T" not in x ]
dia7 = [x for x in dia7 if "Confirmed by" not in x]
dia7 = [x for x in dia7 if "Reconfirmed by" not in x]
```

```
    ##'123' row!(This row also unusual row for Diagnosis information!)
    dia8 = [item for item in l_123 if len(item)>=9]
    dia8 = [x for x in dia8 if "Confirmed by" not in x]
    dia8 = [x for x in dia8 if "Reconfirmed by" not in x]




    ## Collect Diagnosis information!
    diag = dia1+dia2+dia3+dia4+dia5+dia6+dia7+dia8
    diag = '/'.join(diag)


     return p_id, date,p_ethn, p_yr, p_heigt,p_weight, vent_rate, p_sex, pr_interval, qrs_duration, qt, qtc, room_info, loc_info,
diag, qtqtc_vl_tmp

#Find P-R-T axes value!
def fnd_prt(inf_tmp_2,qtqtc_vl_tmp):
    init_index = 0
    initial_index = 0
    tmp = []
    for n in inf_tmp_2:
        n[0]=''
        n[1]=''
        tmp = tmp + [n]
        n=[]

    for i,j in enumerate(tmp):
        if j[2] == 'P-R-T axes':
            final_index = i
        elif j[2] == qtqtc_vl_tmp:
            init_index = init_index + i
            initial_index = initial_index + init_index + 1
        else:
            pass

    prt_tmp = tmp[initial_index : final_index]
    p_axes = list(prt_tmp[2][2])
    p_axes = [ x for x in p_axes if "*" not in x ]
    if len(p_axes) > 3:
        p_axes = []

    p_axes = ''.join(p_axes)
    r_axes = list(prt_tmp[1][2])
    r_axes = [ x for x in r_axes if "*" not in x ]
    if len(r_axes) > 3:
        r_axes = []
    r_axes = ''.join(r_axes)
    t_axes = list(prt_tmp[0][2])
    t_axes = [ x for x in t_axes if "*" not in x ]
    t_axes = [ x for x in t_axes if "sinus" not in x ]
    if len(t_axes) > 3:
        t_axes = []


    t_axes = ''.join(t_axes)

    return p_axes, r_axes, t_axes


def execute(filelist):
    os.chdir("/directory/xml")
    try:
        inf_tmp, filename = par_elem(filelist)
        inf_tmp_2 = par_elem_2(filelist)

        l_40,l_67, l_81, l_94,l_95,l_108,l_109,l_122,l_123 = col_elem(inf_tmp,filename)

          p_id, date, p_ethn,p_yr, p_heigt,p_weight, p_sex, vent_rate, pr_interval, qrs_duration, qt, qtc, room_info, loc_info, diag,
qtqtc_vl_tmp = find_values(l_40,l_67, l_81, l_94,l_95,l_108,l_109,l_122,l_123)

        p_axes, r_axes, t_axes = fnd_prt(inf_tmp_2, qtqtc_vl_tmp)

         dt_lst = [p_id,date,p_ethn,p_yr,p_heigt,p_weight,vent_rate,p_sex,pr_interval,qrs_duration,qt, qtc, room_info, loc_info, p_
axes, r_axes,t_axes,diag]


        dt_lst = pd.DataFrame([dt_lst])

        return dt_lst

    except KeyError :
        print(filename)
        os.system("mv /directory/xml/" + filename + " /directory/unusual_xml/")
        return None

def main(filelist):
    processor = cpu_count()
    proc = os.getpid()

    print("proc_id",proc)
    print(os.fork())
    pool = Pool(processes = cpu_count())
    print("Number of processor:",processor)
    startTime = int(time.time())
    counter = 0

    dt_tmp = pool.map(execute,filelist)

    if dt_tmp is not None :
          df_tmp =pd.DataFrame({"Patient_info":["Patient_ID","Date","Ethnicity","Age(yr)","Height(cm)","Weight(kg)","Sex","Vent.
rate(BPM)","PR_interval(ms)","QRS_duration(ms)", "QC(ms)","QTc(ms)","Room","Location number","P_axes","R_axes",'T_
axes',"Diagnosis"]})

        df_tmp = df_tmp.T



        df = pd.concat(dt_tmp)

        df_tmp = pd.concat([df_tmp, df], axis=0,ignore_index=True)
        os.chdir("/directory/text/")
        df_tmp.to_csv("Result.txt",header=None, index=None,sep=',',encoding='utf-8')
        df_tmp.to_csv("Result.csv",encoding='utf-8')
        print("Complete")

        endTime = int(time.time())
        print("Total converting time", (endTime - startTime))


if __name__ == "__main__":
    print("Start extract ECG text data!")

    os.chdir("/directory/")
    search_directory = "xml"
    filelist = listdir(search_directory)
    main(filelist)
```