University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

BACHELOR'S THESIS

Study programme/specialisation: Bachelor's in computer science	Spring/ Autumn semester, 2021 Open / ~~Confidential~~
Author: Ingrid Nesbø Isene, Withya Wijeyaraj	
Programme coordinator: Trygve Christian Eftestøl Supervisor(s): Trygve Christian Eftestøl	
Title of bachelor's thesis: Development of python-based graphical user interface for handling data from COVID19 patients Title in Norwegian: Utvikling av pythonbasert grafisk brukergrensesnitt for håndtering av data fra COVID19-pasienter	
Credits: 2x10	
Keywords: Python programming File processing GUI Interactive Data analysis	Number of pages: 48 + supplemental material/other: 55 Stavanger, 15.05.2021 date/year

Title page for bachelor's thesis
Faculty of Science and Technology

University of Stavanger
Stavanger, Spring 2021

Development of python-based graphical user interface for handling data from COVID19 patients

Ingrid Nesbø Isene, Withya Wijeyaraj

Supervisor: Trygve Christian Eftestøl

Bachelor thesis, Department of Electrical Engineering and Computer Science

Major: Computer Science

University of Stavanger

Abstract

One of the most significant threats associated with COVID-19 is the increase in hospital admission. The primary study gathering the data was integral as the basis of this thesis. The study investigates whether using biosensors to monitor patients infected by COVID-19 will impact outcomes compared to patients who are not monitored electronically. The biosensors are used to monitor vitals such as heart rate, respiration rate, heart rate variability, and relative stroke volume of a patient group at home.

An interactive and user-friendly GUI was developed to visualize data from patients that participate in the study. The user can navigate the plot and change settings from a menu bar. A vital functionality implemented is the 'Extract data' function. By extracting data, health care professionals can use the data for further data analysis. Data analysis was performed to test the 'Extract Data' functionality. The analysis compared the respiration rate of two patients by calculating the average, median, and standard deviation. The results show that patient 1 has a higher average and median, while patient 2 has a higher standard deviation.

The GUI is implemented using Python and Qt as a framework; this has been used because it is open-source and easy to install or import. Python is an object-oriented language with clear syntax.

Acknowledgements

This bachelor thesis is completed at the Department of Electrical Engineering and Computer Science at the University of Stavanger during the spring of 2021. Thanks are given to our supervisor Trygve Christian Eftestøl for close follow-ups and helpful conversations. In addition to this, a big thank you to Omsyn AS for providing the necessary equipment to gather the data needed for the thesis.

University of Stavanger

Stavanger, May 2021

Ingrid Nesbø Isene

Withya Wijeyaraj

Glossary

- **FHI** - Folkehelseinstituttet
- **GUI** - Graphical user interface
- **CSV** - Comma-separated values
- **QT** - Widget toolkit for creating GUIs
- **NaN** - 'Not A Number', is a numeric data type used to represent any undefined or unpresentable values.
- **Ticks** - The amount of datapoints plotted. One tick represents one datapoint.
- **Backend** - Refers to the server side of an application

Contents

List of Figures

Listings

1 Introduction

COVID-19 was first discovered in Wuhan, China, in December 2019. Since then, the virus has spread across the whole world, making it a pandemic. According to WHO the virus spreads primarily through droplets of saliva from the nose when an infected person coughs or sneezes. Most patients that have tested positive for the COVID-19 virus will experience mild to moderate respiratory symptoms (World Health Organization, 2020). It is estimated that a person who has tested positive for COVID-19 can infect around 2.5 people on average(Gavi the vaccine alliance, 2020). One of the biggest challenges associated with this is the number of hospitalizations. The increasing number of positive COVID-19 tests has put significant pressure on the health system in many countries. In the present situation, it is essential to relieve the stress on health service capacity by identifying those with serious illness without transferring all infected patients to the hospitals. The goal of developing new tools for monitoring the well-being of patients has become essential during the COVID-19 pandemic(Arnulf et al., 2021). The study which gathered the data used in this thesis uses biosensors to collect data from patients who tested positive for COVID-19. The sensors are helpful, both for treatment and safety of care given to these patients. This thesis aims to build a GUI in Python using the QT application framework. To process and visualize CSV files containing the data from patients participating in the study, modules such as Pandas and Matplotlib have been used.

The sensors provided by Omsyn AS measure heart rate, respiration rate, heart rate variability, relative stroke volume, and sleep status at 1-minute intervals. A python-based tool will be developed that will enable the user to:

1. Process incoming files
2. List available patient episodes, where an episode can be selected for further handling.
3. Navigate an episode and visualize a section
4. Edit the plot by using functions in a menu bar
5. Save changes

2 Background

One of the biggest challenges associated with COVID-19 is the increase in hospital admission and prolonged hospital stays. When the virus turned into a global pandemic, hospitals quickly needed to increase their capacity to receive a growing number of patients requiring constant supervision and intensive care. There was an increasing concern that due to the nature of the virus, patients would require long-term intensive care with respirators (Larsen et al., 2020).

The primary study and data collection process is designed to sample biosensor data from patients treated and observed at home due to mild and moderate symptoms caused by COVID-19. This is useful because patients may be referred to hospitalization earlier if their vitals worsen. The local health system may benefit from the feedback of a monitoring system because the system can detect changes in respiration rate, heart rate, sleep status, and relative stroke volume. The main research question is whether using biosensors to monitor patients infected by COVID-19 will impact outcomes compared to patients who are not monitored electronically (Arnulf et al., 2021).

There are two groups in the study. The control group follows the guidelines from FHI on self-care. This patient group reports symptoms, temperature, whether they are taking painkillers and whether they notice a decline in breathing or general health. The intervention group reports the same as the control group. In addition to this, biosensors that measure heart rate, respiration rate, sleep status, heart rate variability, and relative stroke volume at 1-minute intervals are used. These measures will automatically be transmitted to a medical control center for review (Arnulf et al., 2021).

Omsyn AS is the provider of the health care technology used to collect the data for this thesis. The GUI being developed is employed to provide health care professionals with the ability to monitor patients from home that has been infected with COVID-19. This may help determine when patient's vitals worsen to the point in which they would require intensive care.

3 Theory

This chapter aims at presenting the theoretical background related to the thesis. It starts with a short explanation of Omsyn, the company that provides the necessary data and equipments. Then, subsections of the technologies used to develop the GUI is presented. In chapter 4, the development of the GUI and the relevant functions is explained more in-depth.

3.1 Omsyn

Omsyn is a Norwegian-based company that specializes in welfare technology. They produce sensors that measure values pertaining to the individual patient. The values can be customized to fit the healthcare professional's needs. In this study, the essential values determining a patients need of intensive care are heart rate, heart rate variability, respiration rate, sleep status and relative stroke volume. The sensor registers data at 1-minute intervals and forwards the data to the hub through WiFi, as seen in figure 3.1. The location of the sensors is placed at the patient's bed or chair as seen in figure 3.2. When the sensor detects that the patient is sitting or lying down, it will start making measurements. The sensor registers the vibrations and sends the data to the hub 3.3, which is later stored in CSV files (Omsyn, 2021). These are the CSV files being processed and later visualized in the developed GUI.

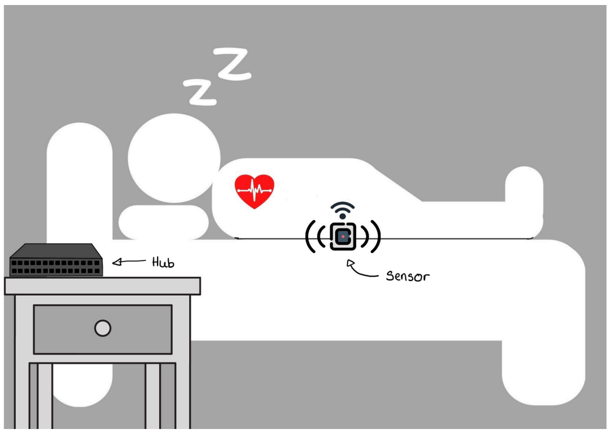


Figure 3.1: Demonstration of how sensor and hub works.

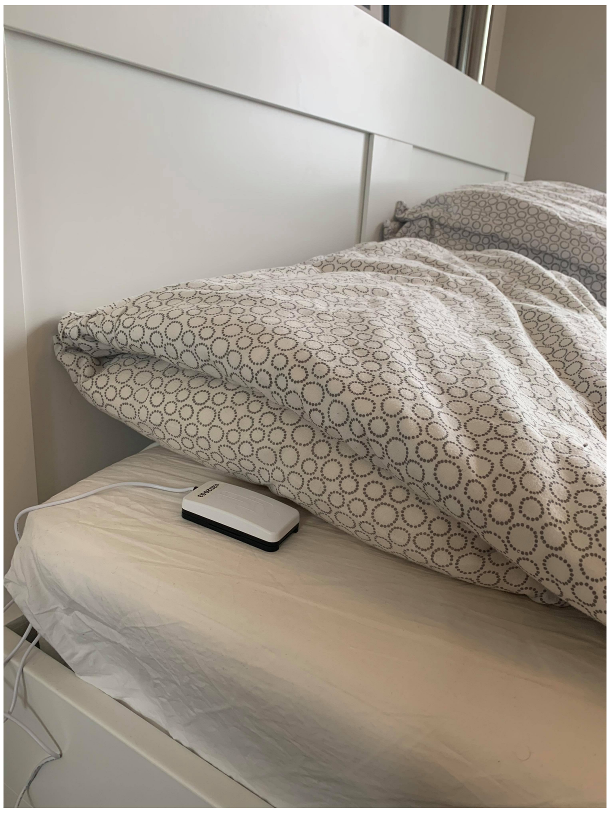


Figure 3.2: Sensor that is used to measure the patients vitals.

Figure 3.3: The sensor sends signal to a hub.

3.2 Python

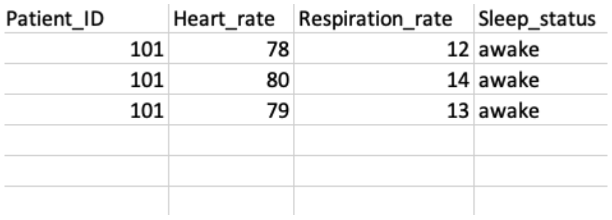
Python is an object-oriented programming language created by Guido van Rossum in 1991 and further developed by the Python Software Foundation (GeeksforGeeks, 2020). One of many characteristics related to Python is that it is an open-source, interactive, and high-level programming language(Mindfire Solutions, 2017). The Python language was preferred for the development of this GUI because of Python classes. Classes provides re-usability and fewer code lines which makes the programming language object-oriented.

3.3 Matplotlib

Matplotlib is an open-source Python library. It provides several ways to create static, animated, and interactive visuals of data, which is why this library is considered essential. One of the benefits of using Matplotlib is that it works well with other Python libraries such as Pandas. In this thesis, Matplotlib is used to visualize the data from the CSV files.

3.4 Pandas

Pandas is a python package that is used for data analysis. The package allows importing data from numerous file formats such as CSV files. One of many benefits of using Pandas is that it allows for data manipulation operations such as merging, selecting, and reshaping data (Wikipedia, 2021). The data that is used in the study is stored in various CSV files. By using Pandas data frames, the data from the CSV files are made more accessible. It enables the program to decide the exact values to retrieve for visualization. To demonstrate the potential of Pandas, an example of data retrieval is shown below. The example illustrates the conversion of a CSV file to a Pandas data frame, and demonstrates different ways to retrieve data.



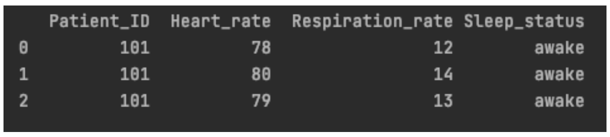
Patient_ID	Heart_rate	Respiration_rate	Sleep_status
101	78	12	awake
101	80	14	awake
101	79	13	awake

Figure 3.4: CSV file.

By calling Pandas built-in read_csv() function, the CSV file is converted to a Pandas data frame.

```
import pandas as pd
df = pd.read_csv('Pandas.csv')
row = df.loc[1]
column = df["Heart_rate"]
```

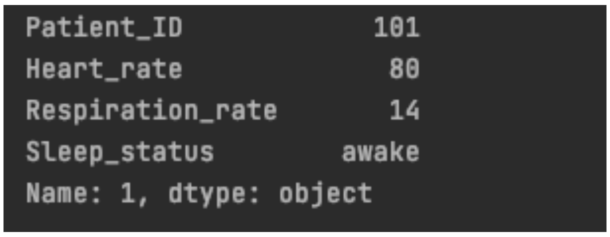
Listing 1: Conversion of CSV file to Pandas data frame



```
   Patient_ID  Heart_rate  Respiration_rate Sleep_status
0         101          78                12        awake
1         101          80                14        awake
2         101          79                13        awake
```

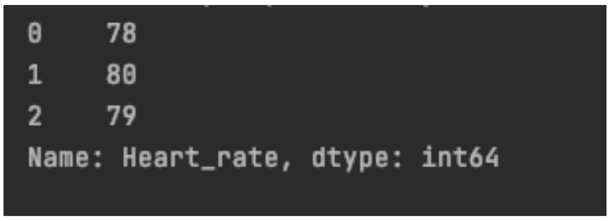
Figure 3.5: CSV file converted to a Pandas data frame.

As mentioned, using Pandas enables the program to decide the exact values to retrieve for visualization. The code snippet above 1 illustrates how Pandas enables the program to retrieve specific rows and columns. Pandas makes data processing easy and simple, which is why this package is used for developing the GUI.



```
Patient_ID            101
Heart_rate             80
Respiration_rate       14
Sleep_status        awake
Name: 1, dtype: object
```

Figure 3.6: Data retrieved from Row, index 1



```
0    78
1    80
2    79
Name: Heart_rate, dtype: int64
```

Figure 3.7: Data retrieved from column, Heart Rate

3.5 Pickle

Python's pickle module is used for serialization. This means that objects can be saved to a file and loaded into a program again later on. It is possible to pickle different objects like Pandas data frames, plot figures, lists, dictionaries, integers, etc. To Pickle, one must specify the name of the file the objects will be saved in (Théo Vanderheyden, 2018).

To store the objects created to develop the GUI, Pickle is used. This allows re-usability of the objects which means that the objects do not need to be recreated every time the GUI is run.

The open() function is used to open the file for writing or reading. It takes on two arguments, the file and the "wb" or "rb" argument, where 'w' means writing to the file and 'r' means reading from the file. 'b' refers to binary mode. To write and store objects, one must use the pickle.dump() function. To read and load objects, one must use the pickle.load()

An illustration of how Pickle is used in Python is shown in the listing 2 below.

```
import pickle
my_pickle_data = {
    'Python': 'open-source',
    'MatLab': 'license',
    'Javascript': 'open-source'
}
with open('myfilename.pickle', 'wb') as file:
    pickle.dump(my_pickle_data, file)

with open('myfilename.pickle', 'rb') as file:
    unserialized_data = pickle.load(file)
```

Listing 2: Pickle example

3.6 GUI

One sub-sector of the software industry is the development of GUIs. One of the main advantages of GUIs is that it allows users to interact with electronic devices through icons and buttons. It was introduced to make computers more user-friendly compared to text-based command-line interfaces. There are several frameworks used to develop a GUI in Python. One of the most common frameworks that are used is PyQt (Editorial Team, 2020).

3.7 PyQt5 Modules

PyQt allows the developers to use much of the functionality of QT in Python. Qt is not a programming language itself, but a framework written in C++. PyQt5 is used to develop this GUI because it is compatible with different operating systems as MacOS, Windows, Android and Unix. It also provides several essential modules that is used to build the GUI. Some of these elements are:

- Widgets
- Layout managers
- Signals and slots

Most of these building blocks are represented as Python classes. The module that provides most of these classes are PyQt5.QtWidgets (Leodanis Pozo Ramos, nd).

3.8 Widgets

Widgets are the base elements for creating user interfaces in QT. Widgets are helpful because they can receive user input, display data, and provide a container for other widgets that should be grouped together (The Qt Company Ltd, 2021d). Widgets are components that can receive key-presses and mouse clicks. When the widget gets an event from a user, it emits a signal to announce its state change. PyQt5 has a vast collection of widgets that can be used for different purposes (Leodanis Pozo Ramos, nd). A group of different widgets used to build this GUI are:

1. Buttons
2. Labels
3. Combo boxes
4. Menu Bars
5. Scroll bars
6. Dialogues

7. Checkboxes

3.8.1 Buttons

It is possible to create a button by importing the QPushButton() class from the QWidget module. When a button is clicked, a signal is sent to the computer to perform concrete actions (Leodanis Pozo Ramos, nd). Several instances of the QPushButton() are initialized when developing this GUI, mainly for the purpose of submitting concrete actions such as selecting an episode for visualization.

3.8.2 Labels

The label module, QLabel(), is used to display text and images (Leodanis Pozo Ramos, nd). To show the user which patient and episode is selected for visualization, the QLabel() class is used.

3.8.3 Combo boxes

Combo boxes can be created in Python by using QComboBox(). A combo box will allow the user to choose options in a drop-down list of data (Leodanis Pozo Ramos, nd). For the user to select patient ID and date, combo boxes are used at the start of the GUI.

3.8.4 MenuBars

Menu bars display options that are relevant to the application. QMenuBar() is displayed differently depending on the operating systems that are being used. It allows the user to choose between different options and impact the GUI according to their wishes. The GUI that is developed holds several menu bar options, and the list below shows some of the functions implemented in the menu bar:

- Change date
- Set Interval
- Show/Hide Plot

The menu bar is the main contributor to making the GUI interactive.

3.8.5 Scrollbars

Implementing a QScrollBar() will enable the user to access parts of the plot that is larger than the plot figure used to display it. The QScrollBar() class can be implemented to provide a vertical or a horizontal scroll bar (The Qt Company Ltd, 2021b).

3.8.6 Dialogues

Dialogues are important to maintain good usability. It allows the user to communicate with the GUI directly. The QInputDialog() lets the user specify an input that is later executed by a function (Martin Fitzpatrick, nd). A window with an input field is displayed when calling the QInputDialog(). These dialogues are used for events such as setting x, y-axis limitations in the developed GUI.

3.8.7 Checkboxes

The QTreeWidget() class provides checkboxes using a tree model. Each checkbox is a QTreeWidgetItem() of the main tree, QTreeWidget(). This class is implemented when creating checkboxes for user selection in this GUI.

3.9 Layout Managers

To position various widgets in a GUI, layout management classes from Qt are used. These classes are used to describe how widgets are positioned in a GUI. One of several benefits of using layout classes is that they automatically place and resize widgets when the number of available spaces changes. This is important because it makes the GUI usable at all times. To give the widgets a clean layout, built-in layout classes such as QHBoxLayout(), QVBoxLayout(), QGridLayout() can be used (The Qt Company Ltd, 2021a).

In the listing 3 below, a ButtonClass() has been created to illustrate a layout example. Three variables have been defined, where layout1 takes on three widgets and places them horizontally, while layout2 does the same but places them vertically. Since one window can only hold one main layout, the MainLayout variable is created and takes layout1 and layout2 as arguments. In the listing below, the mainLayout variable is set to QVBoxLayout().

```
class ButtonClass(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Layout Example")
        layout1 = QHBoxLayout()
        layout2 = QVBoxLayout()
        MainLayout = QVBoxLayout()

        layout1.addWidget(QPushButton("Left"))
        layout1.addWidget(QPushButton("Center"), 1)
        layout1.addWidget(QPushButton("Right"), 2)

        layout2.addWidget(QPushButton("Top"))
        layout2.addWidget(QPushButton("Middle"))
        layout2.addWidget(QPushButton("Bottom"))

        MainLayout.addLayout(layout1)
        MainLayout.addLayout(layout2)
        self.setLayout(MainLayout)
```

Listing 3: Layout managers example

The result is seen in the figure below 3.8.

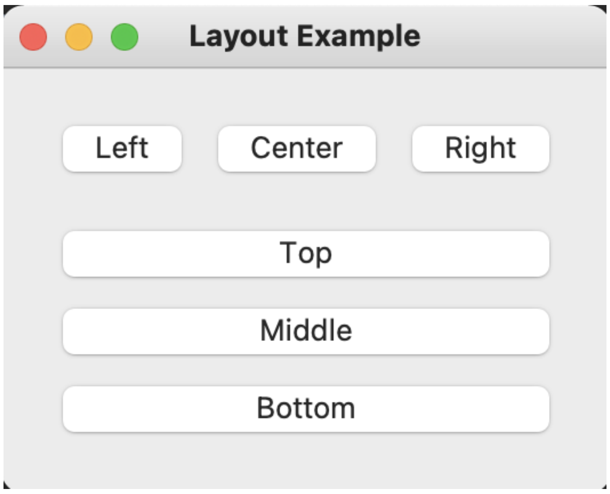


Figure 3.8: Placing widgets using Layout.

A combination of QVBoxLayout() and QHBoxLayout() is used to build the GUI in this thesis.

3.10 Signals and Slots

GUIs are user-friendly because it allows the user to interact with the application through icons and buttons. When a user clicks on a button or types a message in a dialog box, functions or methods are executed in response. The widgets are designed to release a 'signal' in response. A signal does not perform any action alone. For a signal to be executed, it must be connected to a function or a 'slot.' A slot is a callable Python function that contains some action and is called when a signal is emitted. A slot is marked in a Python code by calling the pyqtSlot() function decorator above the executing function (Tutorialspoint, nd).

As mentioned, the menu bar is the main contributor to making the GUI interactive. To connect each menu to a function, signal and slots are used. The listing4 below demonstrates how a signal is connected to a slot in PyQt5.

```

@QtCore.pyqtSlot()
def showDialog(self):
    text,ok = QInputDialog.getText(self, 'input dialog', 'write
text')
    if ok:
        print(text)
```

Listing 4: signal connected to a slot

The slot is connected to an QInputDialog() as seen in the figure3.9 below.

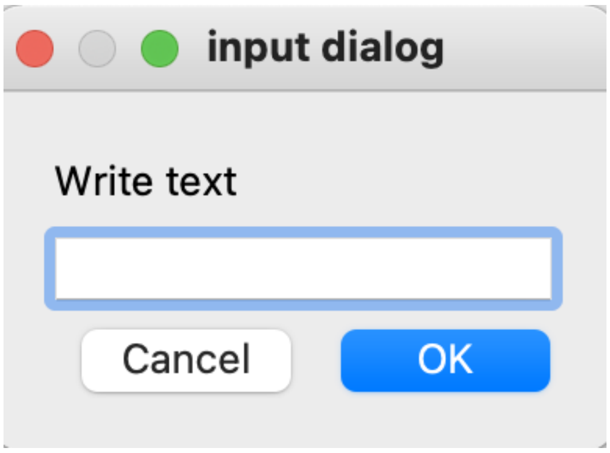


Figure 3.9: Simple example of use of signals and slots input dialog

The slot function is executed when an input in the QInputDialog() is submitted. In return, a print statement is displayed in the terminal window. The listing 4 illustrates how few lines of code are required to create a widget connected to a callable Python function.

4 Methods

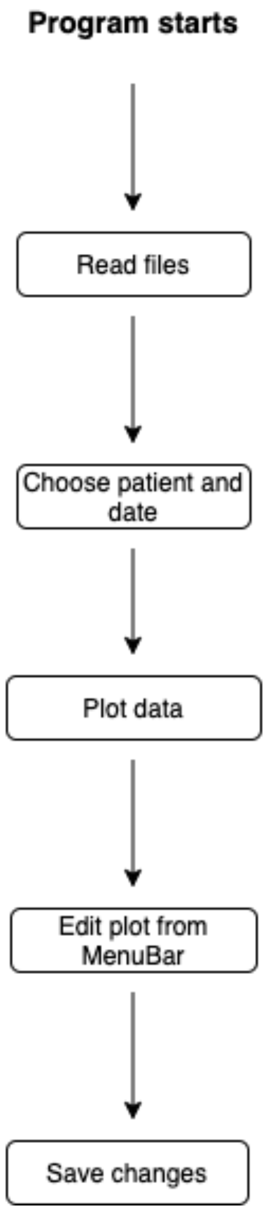


Figure 4.1: Flowchart of the program

The flowchart in figure 4.1 illustrates how the program should work. The idea is that the program should be able to:

1. Process the incoming files
2. List available patient episodes, where an episode can be selected for further handling.
3. Navigate an episode and visualize a section

4. Edit the plot by using functions in a menu bar
5. Save changes

In this chapter, a review of the different methods, functions, and classes will be presented as they appear in the source code. The text is structured so that one begins at the beginning of the program. It starts by explaining how the program processes the incoming CSV files; further, the class and functions that allow the user to choose a patient and date to go forward with are explained. Next, the functions in the menu bar that will let the user edit the plot are presented. At last, the functions used to store the settings in the plot are discussed. Chapter 5 shows the finished result.

4.1 Main.py file

The main.py file is the file that runs the program. Before the program can run, several steps need to be handled. The CSV files must be processed, pickle files must be created, and the main event loop must be called.

4.1.1 File processing

To process the incoming CSV files, two necessary functions have been implemented in the function.py file. The function.py file contains all functions used to process the incoming files. The functions extract_paths() and read _files() are called on in the main.py file. The extract_path() function locates all CSV files within a given directory and returns a list of all paths, as seen below, in listing 5.

```
['/Users/eier/PyCharm_Projects/DATA/covid1.csv', '/Users/eier/
 PyCharm_Projects/DATA/TEexc.csv', '/Users/eier/
 PyCharm_Projects/DATA/covid4.csv', '/Users/eier/
 PyCharm_Projects/DATA/covid3.csv', '/Users/eier/
 PyCharm_Projects/DATA/COVID5.csv', '/Users/eier/
 PyCharm_Projects/DATA/covid2.csv', '/Users/eier/
 PyCharm_Projects/DATA/NAKOS-20200928-20200930.csv']
```

Listing 5: The extract path () function returns the path of each CSV file

The read_files function() takes the path list returned by the extract_path() function as an argument. The read_files() function reads each CSV file and then converts it to a Pandas data frame. The function also returns a Python dictionary containing patient ID as keys and the corresponding Pandas data frames as values, as seen in the listing 6 below.

```
dict_keys(['bee41650-b699-4cac-a838-75f01b2656ae', '6deb3c48-
    c0c7-4567-b6e9-e3b9f1adfb74', '863c5c1b-98a6-4856-b21a-2
    a0c3c898263', 'be3dbf1e-f30e-438c-b8af-532095d50e65', '76
    b29cf7-149b-4533-9a91-d2864761496f', 'e662d866-0594-456f-b5c8
    -795191882330', 'Unknown patient #1'])


                                       patient_id   ...
    date
     0      bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-09
     1      bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-09
     2      bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-09
     3      bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-09
     4      bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-09
     ...                                     ...   ...          ...
     3255   bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-14
     3256   bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-14
     3257   bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-14
     3258   bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-14
     3259   bee41650-b699-4cac-a838-75f01b2656ae   ...   2020-12-14
```

Listing 6: Dictionary with Pandas data frame as value.

The sensors provided by Omsyn AS measure values such as heart rate, respiration rate, sleep status, relative stroke volume, and heart rate variability at 1-minute intervals. These values are set as column names in the CSV files. However, some of the files can have missing values or different column names. For example: if a CSV file does not contain a heart rate column, a column with the same name must be added to the Pandas data frame with corresponding values set to 'NaN'. The read _file function() is implemented to process all CSV files the same way. If a CSV file contains different column names, errors can occur. A predefined list is therefore created to compare the incoming files column names. The known _headers list includes the column names that should be present in each CSV file.

```
known_headers = [
    "patient_id", "time",
    "heart_rate_min","heart_rate_median","heart_rate_max",
    "respiration_rate_min","respiration_rate_median","
   respiration_rate_max",
    "heart_rate_variability_min","heart_rate_variability_median"
   ,"heart_rate_variability_max",
    "relative_stroke_volume_min","relative_stroke_volume_median"
   ,"relative_stroke_volume_max",
    "status","sleep_status"]
```

Listing 7: Predefined list of headers

In listing 6 the read_file function() returns a fully processed CSV file as a Pandas data frame. However, before converting the file to a Pandas data frame, the file must be checked for errors. This occurs in the listing 8 below.

```
df.columns = df.columns.str.lower()
new_headers = (set(list(df.columns)).difference(known_headers))
missing_headers =(set(known_headers).difference(list(df.columns)
   ))
```

Listing 8: Python code from function.py file

The df.columns variable shows which column names are present for the incoming CSV file. The new_headers variable identifies the column names that exist in df.columns, but are not present in the known_headers list. The new_headers variable then checks if the new list consists of column names similar to the columns in known_headers list. An example of this might be if a CSV file contains the column name "Time Stamp," but in known_headers list, that column name is set to "Time". The "Time Stamp" column is then converted to "Time". At last, the missing_headers variable returns a list of the column names that should be present in the df.columns list, but is not there.

Listing 9 below illustrates how the read_files() function works. The CSV file in the current example are missing the column names 'sleep_status', 'patient_id', 'time' and 'status'. First, the function replaces the name 'timestamp' with 'time' from the known_headers list. The function then checks if some of the columns are still missing and prints a 'still missing' list. The column names that are still missing are added to the Pandas data frame with 'NaN' values. If a file has a missing patient ID, a default ID is given. At last a Final list is printed. The final list is identical to the known_headers list.

```
New:{'timestamp'}
Missing:{'sleep_status', 'patient_id', 'time', 'status'}
Merged:['time', 'heart_rate_min', 'heart_rate_max', '
    heart_rate_median', 'respiration_rate_min', '
    respiration_rate_max', 'respiration_rate_median', '
    heart_rate_variability_min', 'heart_rate_variability_max', '
    heart_rate_variability_median', 'relative_stroke_volume_min',
     'relative_stroke_volume_max', 'relative_stroke_volume_median
    ']
Still missing:{'status', 'sleep_status', 'patient_id'}
File:/Users/eier/PyCharm_Projects/NAKOS-20200928-20200930.csv
    does not contain "patient_id. Default ID is given: Unknown
    patient # 1
Final:['time', 'heart_rate_min', 'heart_rate_max', '
    heart_rate_median', 'respiration_rate_min', '
    respiration_rate_max', 'respiration_rate_median', '
    heart_rate_variability_min', 'heart_rate_variability_max', '
    heart_rate_variability_median', 'relative_stroke_volume_min',
     'relative_stroke_volume_max', 'relative_stroke_volume_median
    ', 'status', 'sleep_status', 'patient_id']
```

Listing 9: Processing files with missing columns

4.1.2 Storing files

To avoid re-processing the files the next time the program is run, the dictionary returned by the read_files() function and the path list returned by the extract_paths() is pickled. As mentioned in the theory part of this thesis, pickle is used to store objects. In listing 10 below, the 'pickle.pkl' file is created. The files are then processed and the objects returned by the extract_paths() and read_files() are dumped into the pickle file.

```
directory = DATA_PATH
pathList = extract_paths(directory)
DICT_ID_df = read_files(pathList)

objects = [pathList, DICT_ID_df]
file = open(PICKLE_PATH+'/pickle.pkl', "wb")
pickle.dump(objects, file)
file.close()
```

Listing 10: Python Pickle

The users can edit the plot by using functions in a menu bar. To store these settings, two more pickle files are created in the main.py file as seen in listing 11 below. 'plot_settings.pkl' for standard settings such as setting intervals, and 'patient_settings.pkl' for storing settings such as markers and texts. The empty dictionaries dumped into these pickle files are later filled when settings are saved. This is discussed later in this chapter.

```
plot_settings = {}
with open(PICKLE_PATH+'/plot_settings.pkl', "wb") as f:
    pickle.dump(plot_settings, f)

patient_settings = {}
with open(PICKLE_PATH+'/patient_settings.pkl', "wb") as
f:
    pickle.dump(patient_settings, f)
```

Listing 11: Pickle files being created

An illustration of how the main.py file operates is shown in the figure 4.2 below.

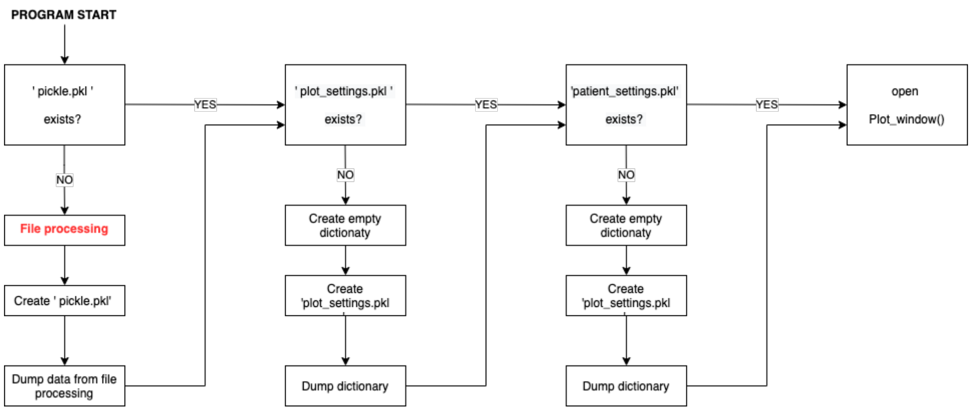


Figure 4.2: Flowchart of main.py

4.1.3 Application control

A GUI can consist of several windows. For the windows to appear when running the program, a QApplication() object is called. The QApplication() is the main event loop, meaning it controls the application's flow control and primary settings. For a GUI using Qt, a QApplication() object is needed. After calling this class, an instance of the Choice_window() class is created, which is the first window being displayed.

4.2 Choice _window class

After the files are processed, a window listing available patient episodes should appear. The application must be able to visualize data from a specific patient for a particular date. To select which patient episode to visualize, the user must first choose a patient ID and a corresponding date to go forward with. This occurs in the Choice_window() class.

```
class Choice_window(QWidget):


    def __init__(self):
        objects = pickle.load(open(PICKLE_PATH+"/pickle.pkl", "
  rb"))
        self.DICT_ID_df = objects[1]


        super().__init__()
        self.model = QStandardItemModel()

        self.comboID = QComboBox()
        self.comboID.setModel(self.model)

        self.comboDate = QComboBox()
        self.comboDate.setModel(self.model)

        self.submit = QPushButton('SUBMIT')
        self.submit.clicked.connect(self.open_plot)
```

Listing 12: Python class used to create the first window.

The 'pickle.pkl' created in the main.py file, containing the dictionary holding the patient IDs and dates, is opened in this class. The Choice_window() class extracts the keys and values from the dictionary and lists the data in a QComboBox(). Each time a user selects a different patient ID, the list of corresponding dates should be updated. When the user submits the patient ID and date from the QComboBox(), the open_plot() function is called as seen in listing 13 below. This function creates an instance of the Plot_window() class that displays the plots.

```
    def open_plot(self):
        ID = self.comboID.currentText()
        DATE = self.comboDate.currentText()
        self.plotWin = Plot_window(ID,DATE)
        self.plotWin.showFullScreen()
        self.close()
```

Listing 13: open plot() function

4.3 Plot_window() class

The Plot_window() class is responsible for visualizing the selected episode. It initializes all objects and variables necessary for creating the plot window. This implies objects such as plot figures and Pandas data frames. To initialize objects, four functions are implemented.

The functions being used depend on whether previous configurations are stored in pickle files, as shown in the flowchart, figure 4.3 below. The very first time the program is run, the first_plot_setting() function is called. Due to no previous plot settings, the 'plot_settings.pkl' will contain an empty dictionary and will therefore initialize the needed objects by creating them. The next time the program is run, the Plot_window() class calls the stored_config() function, passing the dictionary from 'plot_settings.pkl' as argument. Since the dictionary holds data from the last session, it initializes the needed objects by retrieving them from the pickle file. This way, the program does not have to recreate the objects.

The same goes for creating objects for patient episodes. If an episode is opened for the first time, the new_date_setting() function is called. This function retrieves the data for the specific patient ID and date, and creates the plot objects. If an episode have been opened before, the stored_date_setting() function is called. Similar to the stored_config(), it retrieves stored objects from the pickle file and initializes them.

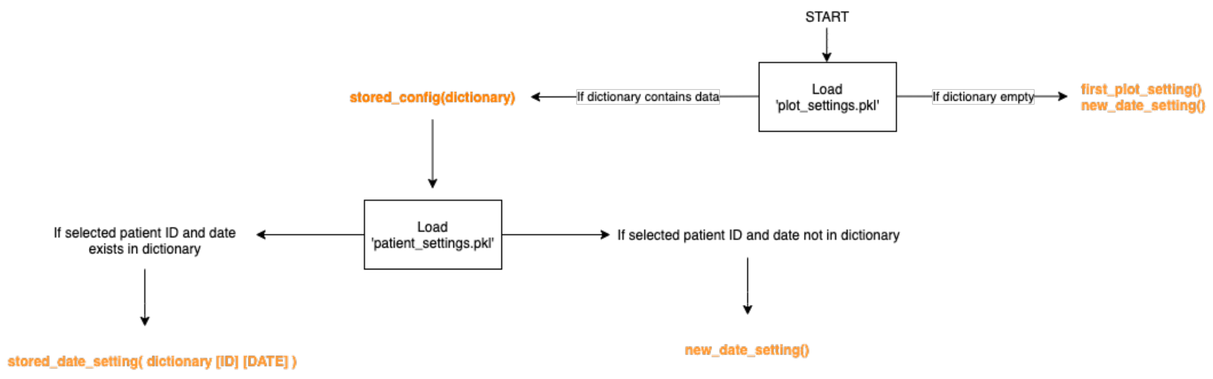

START
Load 'plot_settings.pkl'
If dictionary contains data
stored_config(dictionary)
If dictionary empty
first_plot_setting()
new_date_setting()
Load 'patient_settings.pkl'
If selected patient ID and date exists in dictionary
stored_date_setting(dictionary [ID] [DATE])
If selected patient ID and date not in dictionary
new_date_setting()

Figure 4.3: Initialization in Plot_window() class

```
def new_date_setting(self):
        self.start_tick= 0
        self.SLEEP= plotClass(False , None, None)
        self.HR = plotClass(False , None, None)
        self.RESP = plotClass(False , None, None)
        self.HRV = plotClass(False , None, None)
        self.REL= plotClass(False , None, None)
        self.FULL= plotClassFull()
        self.df, self.markerList= GET_DF_FOR_DATE(self.ID, self.
    DATE, self.DICT_ID_df)
```

Listing 14: Creating objects

```
    def stored_date_setting(self, patient_s):
        self.start_tick = patient_s['start_tick']
        self.SLEEP= plotClass(False, None, None)
        self.HR = plotClass(True, patient_s['HR']['fig'],
    patient_s['HR']['axes'])
        self.RESP= plotClass(True,patient_s['RESP']['fig'],
    patient_s['RESP']['axes'])
        self.HRV = plotClass(True, patient_s['HRV']['fig'],
    patient_s['HRV']['axes'])
        self.REL = plotClass(True, patient_s['REL']['fig'],
    patient_s['REL']['axes'])
        self.FULL= plotClassFull()
```

Listing 15: Retrieving objects from pickle file.

During the process described above, in figure 4.3, the Pandas data frame for the patient episode is initialized. This is done by calling the GET_DF _FOR _DATE() function as seen in listing 14. The function returns the Pandas data frame for the selected ID and date, which holds the data that will be visualized. Another important functionality is that the function adds missing timestamps between the first and last timestamp in the Pandas data frame. The timestamps are added to envision the full-time course of the episode. It also returns a list of timestamps that are located between 'NaN' values.

```
def GET_DF_FOR_DATE(ID,DATE,DICT_ID_df):
    df = DICT_ID_df[ID]
    df = df[df['date'] == DATE]
```

Listing 16: Python function

Other important objects that are initialized during this process are the figures that will visualize the Pandas data frame. The plotClass() creates these figures. Five instances of this class are created as seen in listing 14. Each figure holds one subplot and one ax, and the canvas these figures renders into is set up using the 'FigureCanvasQTAgg' widget.

```
class plotClass(FigureCanvasQTAgg):
    def __init__(self, pickle_state,fig, ax):
            self.fig = fig
            self.axes = ax
            super(plotClass, self).__init__(self.fig)
            self.fig.tight_layout()
```

Listing 17: plotClass()

The plot class creates the figures that will contain the data from the patients. However, to plot the data, the function plotData() is used. This function is located in the Plot _window() class.

```
def plotData(self):
    if self.pickle_state == False:
        for i in epi_dict.keys():
            ax = self.get_widget(i).axes
            ax.set_title(i, loc='left', fontsize=7)

            ax.plot(self.df['time'], self.df[episode], linestyle
   ='-', color=color, label=string)
```

Listing 18: plotData()

Each patient ID plots four plots by default; these four are heart rate, respiration rate, heart rate variability, and relative stroke volume. The biosensors from Omsyn AS measure these values in min, max, and median. These values are added to a Python dictionary. The dictionary takes the four main plots as keys, and min, max, median as values. This is done in epi_dict dictionary as seen in listing 19. Looping through this dictionary, the plotData() function plots three lines representing the min, max, and median value, in each plot.

```
epi_dict = {
    'Heart Rate': ["heart_rate_min","heart_rate_median","
   heart_rate_max"],
    'Respiration Rate' : ["respiration_rate_min","
  respiration_rate_median","respiration_rate_max"],
    'Heart Rate Variability ': ["heart_rate_variability_min","
  heart_rate_variability_median","heart_rate_variability_max"],
    'Relative Stroke Volume': ["relative_stroke_volume_min","
  relative_stroke_volume_median","relative_stroke_volume_max"]
}
```

Listing 19: Episode dictionary

At last, the Plot_window() class sets the main layout for the application window. This layout is set to be vertical using the QVBoxLayout(). Along with the figures created, other widgets are also added to the main layout to make the GUI more interactive. The functionality of these widgets is discussed in the next section.

```
self.mainLayout = QVBoxLayout()
self.mainLayout.addWidget(self.menuBar)
self.mainLayout.addLayout(Label, 0.5)
self.mainLayout.addLayout(PlotLay, 8)
self.mainLayout.addLayout(self.scrollBar.Scroll_Lay, 1)
self.setLayout(mainLayout)
```

Listing 20: Layout

4.4 Interactive Widgets

To make the GUI more interactive, widgets such as QScrollBar() and QMenuBar() have been added. These widgets respond to events caused by user actions. By adding these widgets to the GUI, the user is able to edit and navigate an episode.

4.4.1 QScrollBar()

QScrollBar() allows the user to navigate back and forth in the plot. Large datasets can be challenging to read when displayed in one window. The plot is therefore set to show a specific interval at a time. The scroll bar represents the full episode, while the grey area within the scroll bar represents the exact interval being displayed as seen in figure 5.4.

When navigating, the scroll_action() function is triggered. To know which fraction of the plot is being shown, a color patch is created to mark the specific area. This color patch is also updated for every scroll_action(). At last, the function redraws the plot.

```
def scroll_action(self):
    self.start_tick = self.scrollBar.scroll.value()
    self.setInterval()
    self.updateColorPatch()
    self.redrawAll()
```

Listing 21: Scroll function

4.4.2 QMenuBar()

QMenuBar() allows the user to edit the plot and specify their settings. The menu bar consists of several settings, each connected to a Python function. For a user to receive a response to the event requested when editing, the signals are connected to slots using pyqtSlot() function decorator. An example of functions that allows the users to edit the plot is setting intervals.

The set_x _interval() function is one of many functions connected to the QMenuBar(). When this action is called, a QInputDialog() window is displayed. This allows the user to specify a new interval as seen in figure 5.8. Pressing the submit button will update the self.interval variable. This variable decides the length of the interval. The interval is then updated as seen in figure 5.9, and the plot is redrawn.

```
@QtCore.pyqtSlot()
def set_x_interval(self):
    text, ok = QInputDialog.getText(self, 'Interval Settings', '
   Set x-interval:')
    if ok:
        self.interval = int(text)
        self.setInterval()
        self.updateColorPatch()
        self.redrawAll()
```

Listing 22: Setting X-interval

Another essential functionality added to the menu bar is the 'Extract Data' function. This allows the user to mark specific areas of interest and retrieve the data within the area. This is useful because it will enable the user to keep areas of interest and further use the data retrieved in data analysis.

To retrieve data from a specific area, the user must specify an interval name. This is achieved through a QInputDialog() as seen in listing 23. When submitting the input, a 'button_press_event' is activated, allowing the user to mark two specific areas within the plot. The two markers represents the start and end of the area in interest as seen in figure 5.10.

```
def activate_MarkInterval(self, bool, prev):
    if bool == True:
        text, ok = QInputDialog.getText(self, 'INTERVAL', '
ENTER NAME FOR INTERVAL:')
        if ok:
            self.cid = {}
            for i in self.get_all_main_wigets():
                cid = i.fig.canvas.mpl_connect('
button_press_event', partial(self.MarkInterval,text))
                self.cid[i] = cid
```

Listing 23: Input dialog

The function triggered by the 'button_press_event' is MarkInterval(). This function takes the input from the QInputDialog() as an argument. When the user clicks on the plot, points and annotations are added. After two clicks, the marked area is stored, and the event is disconnected.

```
def MarkInterval(self,text,event):
    ax = event.inaxes
    self.p1 = ax.plot(event.xdata, event.ydata, marker='.',
color='purple',ms=10, ls="", label='_'+text)
    self.a1 = ax.annotate(text+' START', xy=(event.xdata -
0.3, event.ydata + 0.1),weight='bold')
```

```
        self.start_time = self.df['time'].loc[round(event.xdata)
    ]
        self.redrawAll()
        self.count+=1
        self.MarkedIntervals.append([text, self.start_time, self.
    end_time])

        for i in self.get_all_main_wigets():
            i.fig.canvas.mpl_disconnect(self.cid[i])
          self.cid = {}
```

The marked area is stored in the self.MarkedIntervals list, which holds the starting- and ending timestamp of the interval. For a user to retrieve the data between these timestamps, the create_Excel() function is implemented. As shown in listing 24, the function extracts the data from the Pandas data frame and stores it by creating an Excel file. Whenever the user marks new intervals, each interval is added as a sheet within the Excel file. A table showing an overview of all marked areas is also added to this file as seen in figure 5.12.

```
@QtCore.pyqtSlot()
def create_Excel(self):
    writer = pd.ExcelWriter(Date_path, engine='xlsxwriter')
    df = DataFrame(self.MarkedIntervals, columns=['INTERVAL', '
  START TIME', 'END TIME'])
    df.to_excel(writer, sheet_name='Intervals')

    for interval in self.MarkedIntervals:
        index_start = self.df[self.df['time']==start_t].index[0]
        end_start = self.df[self.df['time'] == end_t].index[0]

        interval_df.to_excel(writer, sheet_name=name)

    writer.save()
```

Listing 24: Store data in Excel

4.4.3 HideClass()

To let the user decide which plot figures and lines to display, the HideClass() is implemented. The class creates checkboxes by using the QTreeWidget(). Each checkbox value represents either the plot figure or the lines within the plot, as seen in figures 5.5 and 5.6. These checkboxes are made by iterating through both keys and values in the epi_dict dictionary, listing 19. The reason both keys and values are iterated through is to put parent items and child items to the QTreeWidget(). The listing 25 below, demonstrates how the checkboxes are created.

```
class HideClass(QWidget):
    def __init__(self,child_status,uncheck_status,unselectedList
    , selectedList):
        super().__init__()
        self.tree = QTreeWidget()
        for header in epi_dict.keys():
            parent = QTreeWidgetItem(self.tree)
            parent.setText(0, header)
            parent.setCheckState(0, Qt.Checked)
            parent.setFlags(parent.flags() | Qt.ItemIsTristate |
    Qt.ItemIsUserCheckable)

            for x in epi_dict[header]:
                child = QTreeWidgetItem(parent)
                child.setFlags(child.flags() | Qt.
    ItemIsUserCheckable)
                child.setText(0, x)
                child.setCheckState(0, Qt.Checked)
```

Listing 25: HideClass()

Three instances of the HideClass() is initialized when the first_plot_setting() function is called inside the Plot_window() class, listing 26. The functionalities of the checkboxes in each instance vary. The self.hidePlot instance decides how many plots to display in one, as seen in figure5.5. By default, the program starts with four plots, but the 'Show/Hide

Plot' functionality added to the menu bar allows the users to decide this through the HideClass(). The self.hideLines instance decides which values to show in the plot, as seen in figure5.6. By default, min, max, and median values appear in the plot. The 'Show/Hide Line' functionality added to the menu bar handles the user's choice of lines. The last instance, the self.Vline_win allows the user to decide whether the plots should show a vertical line between the min and max values. This functionality is added as 'Show/Hide V-lines' to the menu bar.

```
def first_plot_setting(self):
    self.interval        = 30
    self.current_full    = 'heart_rate_min'
    self.hidePlot        = HideClass(False, False, [], [])
    self.hiddenPlots     = []
    self.hideLine        = HideClass(True, False, [], [])
    self.hiddenLines     = []
    self.Vline_win       = HideClass(False, True, [], [])
```

Listing 26: HideClass() instances initialized.

4.5 Data Storage

Under the Main subsection it was described how pickle files were created and how stored data was opened and used. In this subsection it is described how data is stored before the program is closed. To save the settings from the menu bar, the settings are dumped into pickle files. Whenever the user changes a date, a patient ID or closes the application, a call to the SAVE() function is made.

```
def SAVE(self):
    plot_settings, patient_settings = self.__getstate__()

    with open(PICKLE_PATH+'/plot_settings.pkl', "wb") as f:
        pickle.dump(plot_settings, f)

    with open(PICKLE_PATH+'/patient_settings.pkl', "rb") as
  f:
```

```
            patient_s = pickle.load(f)

        if self.ID not in patient_s.keys():
            patient_s[self.ID] = {}
        patient_s[self.ID][self.DATE] = patient_settings

        with open(PICKLE_PATH+'/patient_settings.pkl', "wb") as
    f:
            pickle.dump(patient_s, f)
```

Listing 27: The SAVE() function.

The SAVE() function retrieves two dictionaries of objects returned by the __getstate__() function, where objects in interest are stored. As seen in the flowchart below, figure 4.4, the dictionary containing objects related to the plot setting is dumped into the 'plot_settings.pkl' file. Then it opens the 'patient_settings.pkl' file, which returns a dictionary containing all patient IDs as keys and corresponding date and data as key-value pairs. The second dictionary returned by the __getstate__() function is then added and finally dumped back into the 'patient_settings.pkl' file.

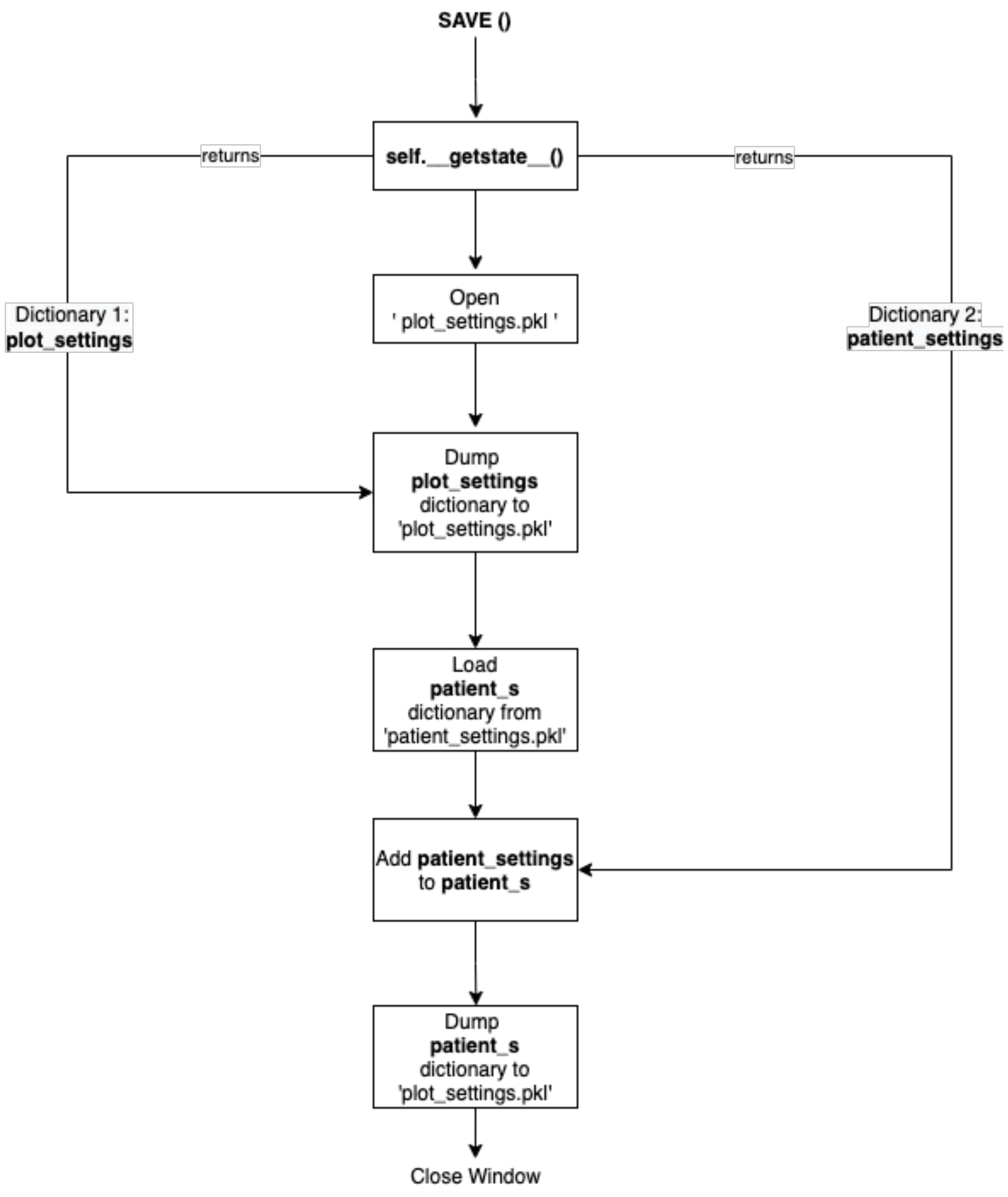

SAVE ()
self.__getstate__()
returns
returns
Dictionary 1: plot_settings
Dictionary 2: patient_settings
Open ' plot_settings.pkl '
Dump plot_settings dictionary to 'plot_settings.pkl'
Load patient_s dictionary from 'patient_settings.pkl'
Add patient_settings to patient_s
Dump patient_s dictionary to 'plot_settings.pkl'
Close Window

Figure 4.4: Flowchart of the SAVE() function.

5 Result

After the program has processed the incoming CSV files, the Choice_window() will appear as demonstrated in figure 5.1. This window consists of combo boxes, which allows the user to select a patient ID and a corresponding date to go forward with. Choosing a patient ID and date allows the user to follow up with individual patients easily and straightforwardly. In figure 5.2 the combo box lists all patient IDs and in figure 5.3 the corresponding dates to the patient ID are listed.

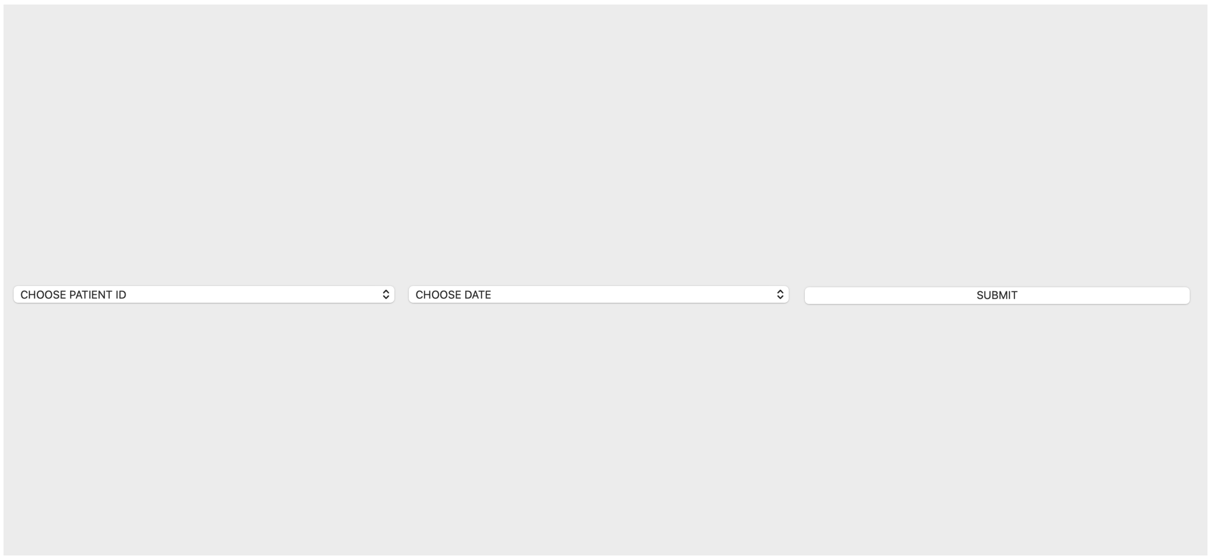


Figure 5.1: Combo box containing patient IDs and dates

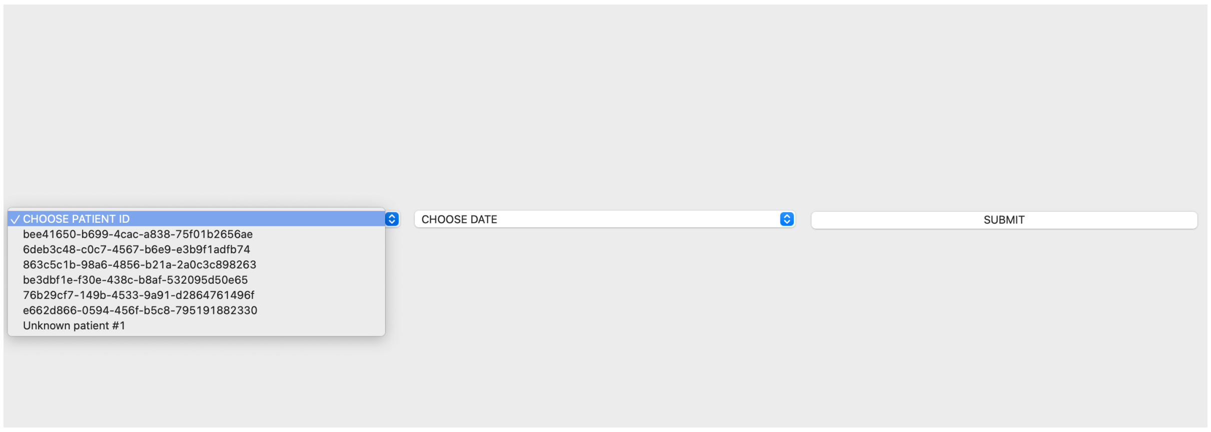

✓ CHOOSE PATIENT ID
bee41650-b699-4cac-a838-75f01b2656ae
6deb3c48-c0c7-4567-b6e9-e3b9f1adfb74
863c5c1b-98a6-4856-b21a-2a0c3c898263
be3dbf1e-f30e-438c-b8af-532095d50e65
76b29cf7-149b-4533-9a91-d2864761496f
e662d866-0594-456f-b5c8-795191882330
Unknown patient #1
CHOOSE DATE
SUBMIT

Figure 5.2: List of patient IDs

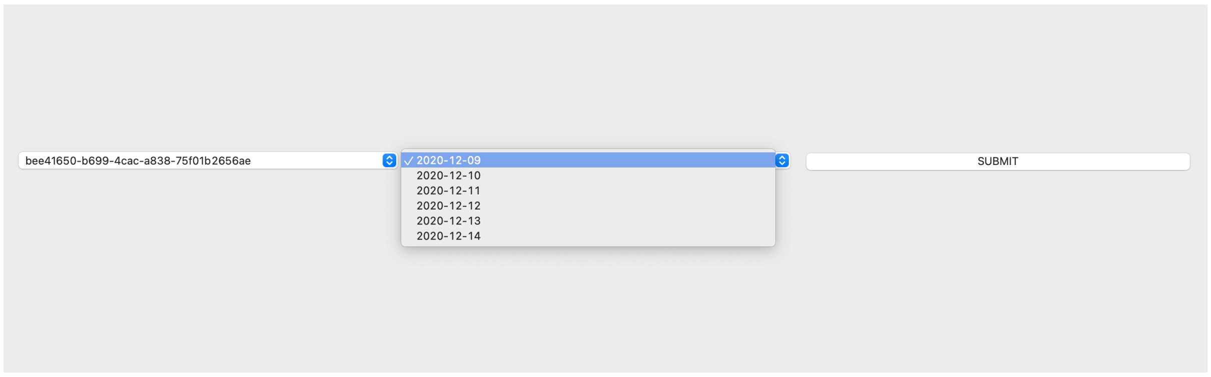
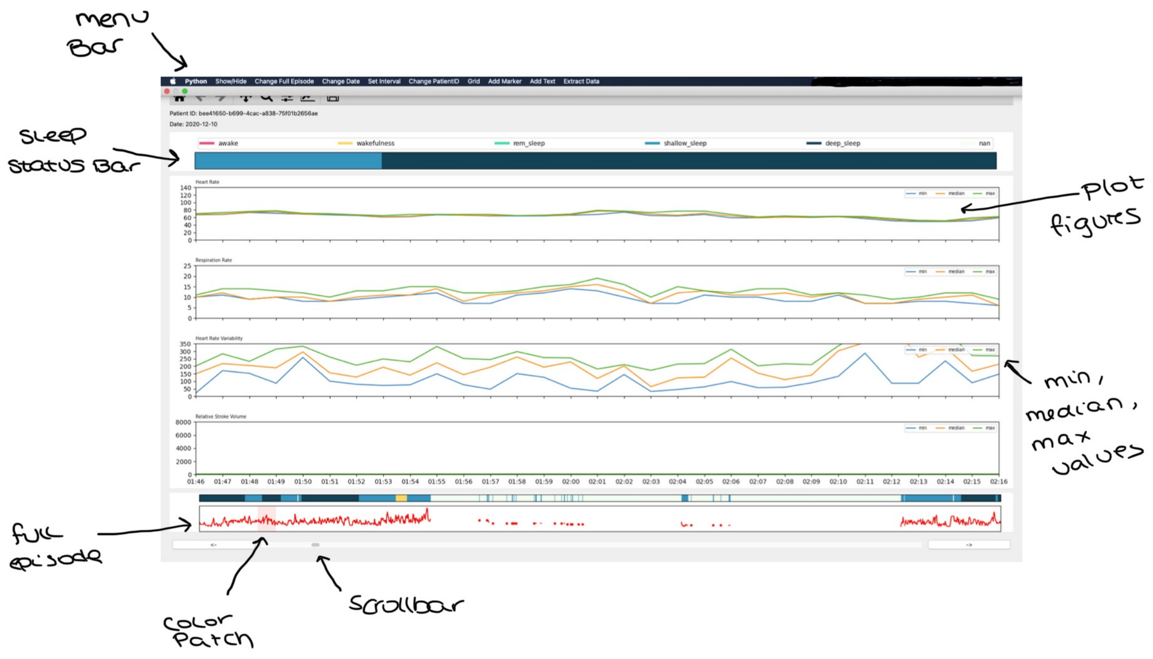


Figure 5.3: List of dates corresponding to a specific patient ID

After selecting a patient ID and a date, the user is redirected to the plot as shown in figure 5.4 below. By default, all four plots containing heart rate, respiration rate, heart rate variability, and relative stroke volume are shown in the window. An overview of the patient's sleep status is also displayed at the top, next to labels displaying the patient ID and date. The four plot figures and the sleep status bar shows the data plotted for a given interval. Therefore, another plot and sleep status bar displaying the full time course of a full episode is displayed at the bottom of the GUI as seen in figure 5.4. By default, the heart rate episode is shown, but the full episode can be changed from the menu bar by clicking on 'Change full episode'. The pink color patch attached to the full episode represents the exact area displayed in the four plots above. When navigating with the scroll bar, the color patch will follow.

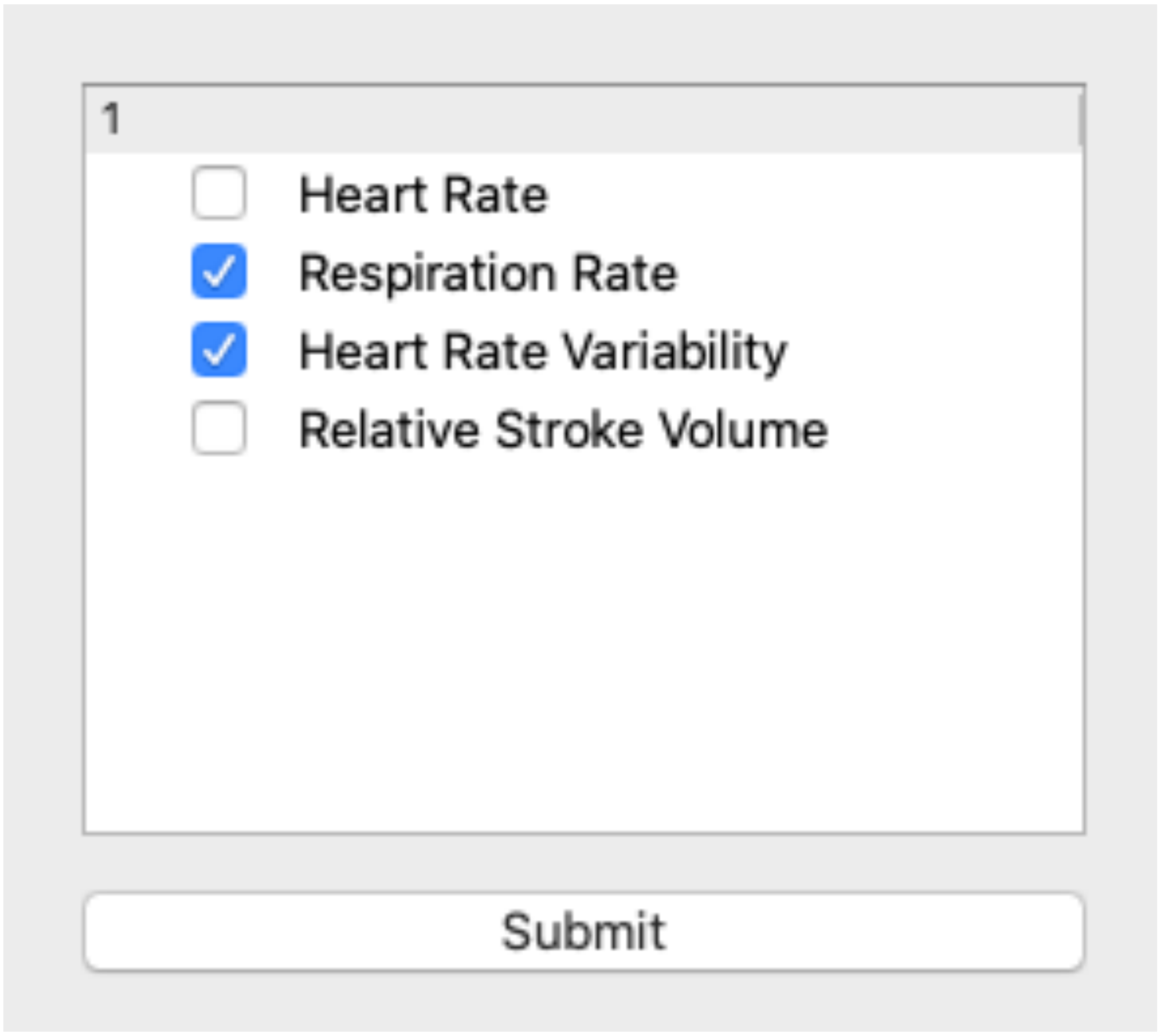


Patient ID: bee41650-b699-4cac-a838-75f01b2656ae

Date: 2020-12-10

Figure 5.4: Figure of Plot Window

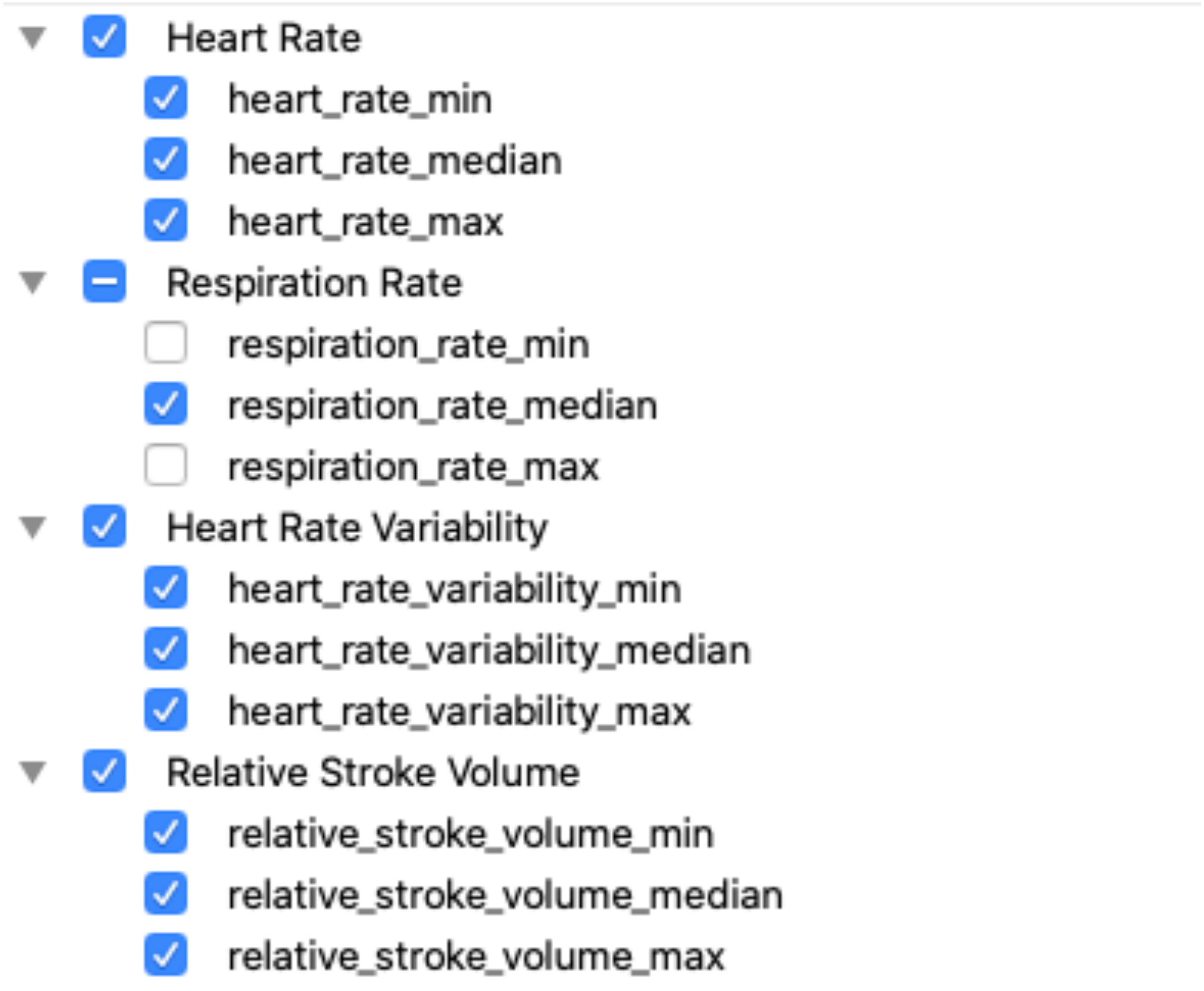
The menu bar is displayed at the top of the window. There are several features associated with the menu bar that allows the user to make changes to the plot. The user will have the option to select how many plots to display in the window. This can be changed in the menu bar by clicking on 'Show/Hide'. The user can also choose which values to display in the plot. In the figure 5.7 below, the plot figures for respiration rate and heart rate variability are displayed. In the respiration rate plot, only the median value is displayed. The minimum and max values have been hidden. This can easily be changed back again if desired. Figures 5.5 and 5.6 demonstrates the check boxes used to select plots and lines.



1

- [ ] Heart Rate
- [✓] Respiration Rate
- [✓] Heart Rate Variability
- [ ] Relative Stroke Volume

Figure 5.5: Selection of plots to display



- [✓] Heart Rate
  - [✓] heart_rate_min
  - [✓] heart_rate_median
  - [✓] heart_rate_max
- [–] Respiration Rate
  - [ ] respiration_rate_min
  - [✓] respiration_rate_median
  - [ ] respiration_rate_max
- [✓] Heart Rate Variability
  - [✓] heart_rate_variability_min
  - [✓] heart_rate_variability_median
  - [✓] heart_rate_variability_max
- [✓] Relative Stroke Volume
  - [✓] relative_stroke_volume_min
  - [✓] relative_stroke_volume_median
  - [✓] relative_stroke_volume_max

Figure 5.6: Selection of lines to display

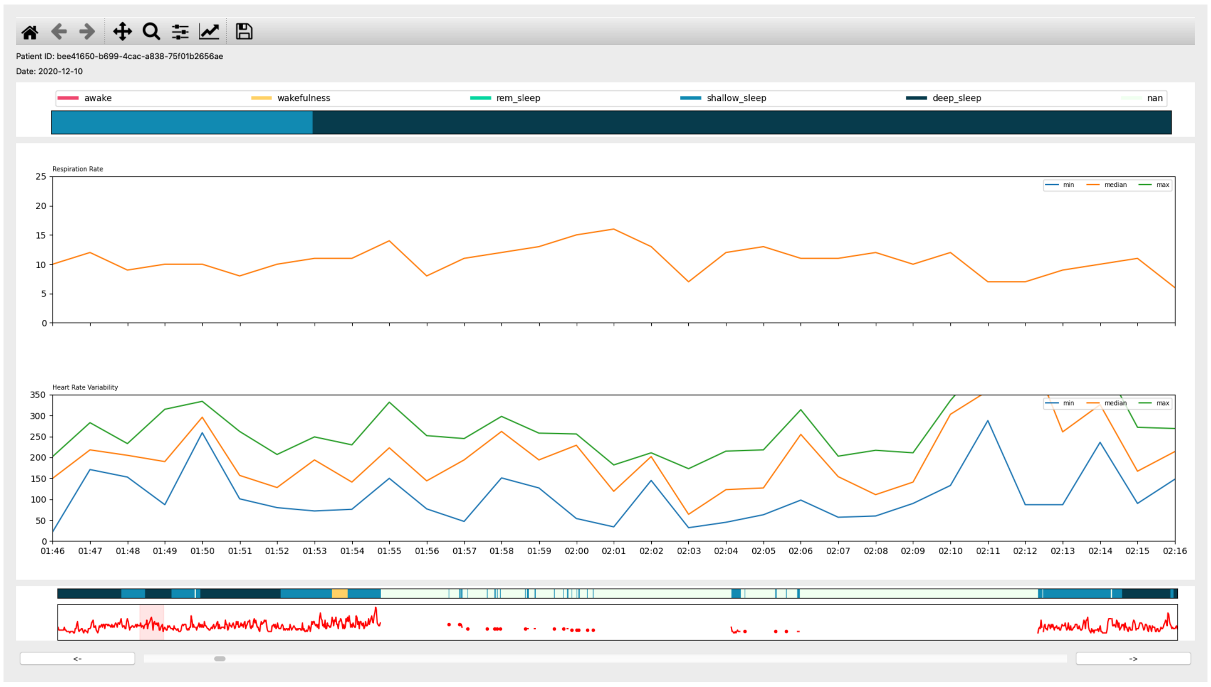


Figure 5.7: The plot after change in settings.

Another feature associated to the menu bar is to set intervals on the x and y axes. By default, 30 'ticks' are displayed on the x axis. This value can be changed in the menu bar by selecting 'Set Interval'. By clicking on 'Set Interval' a input dialog is displayed as seen in figure 5.8.

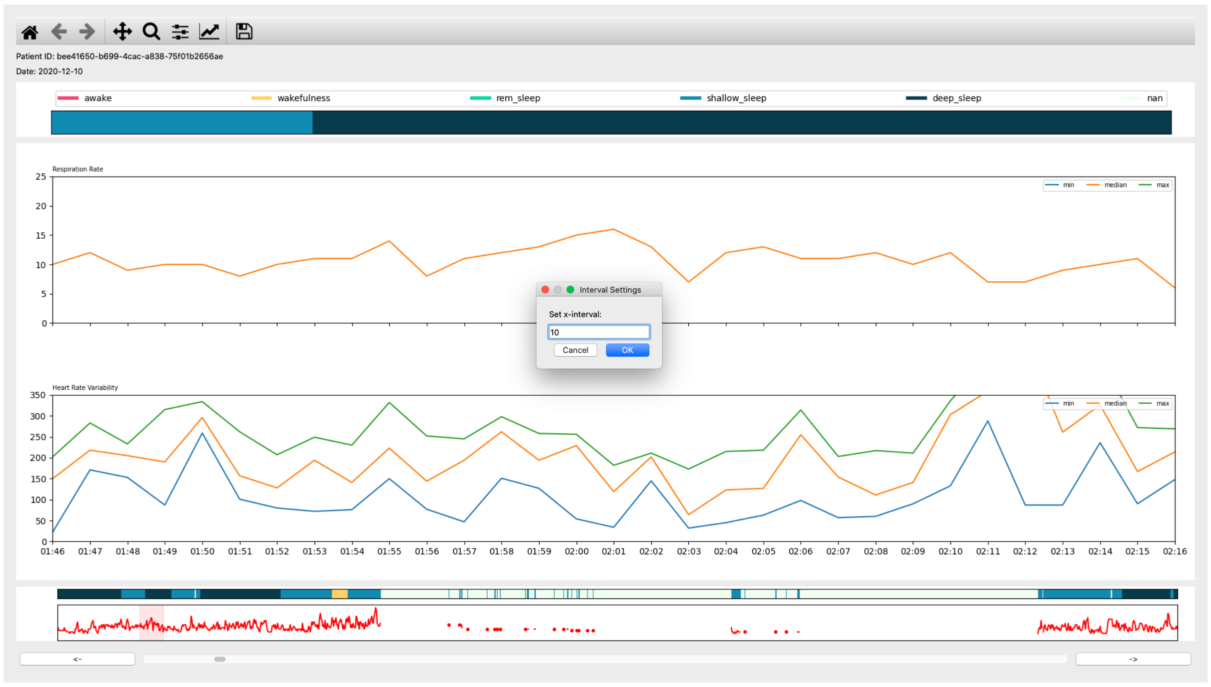
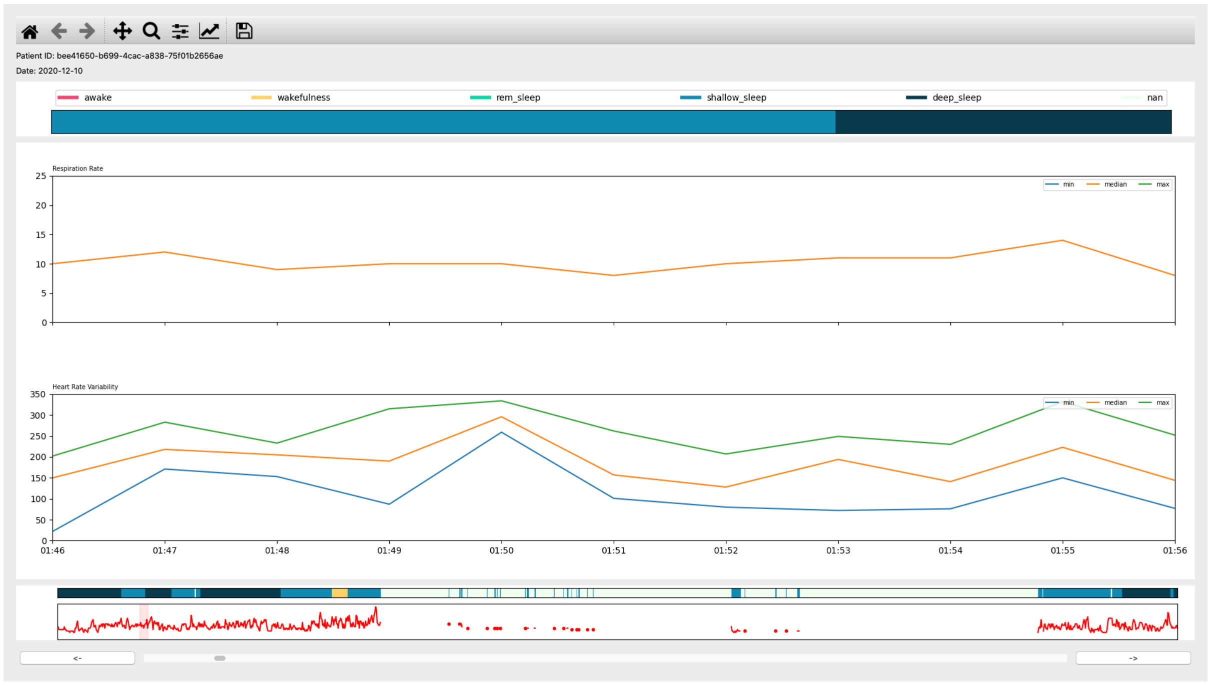


Figure 5.8: Setting ticks on x-axis

After submitting the input from the input dialog box, the number of ticks on the x interval will update from the default value to 10 ticks, seen in figure 5.9.


Patient ID: bee41650-b699-4cac-a838-75f01b2656ae

Date: 2020-12-10

Figure 5.9: Plot after updating ticks on x-axis

Other features to make the GUI more interactive are also implemented in the menu bar.

- Show / Hide Plots
- Show / Hide Lines
- Show / Hide V-lines (Vertical lines)
- Change Full Episode
- Change Date
- Set X-interval
- Set Y-interval
- Change Patient ID
- Add Grid
- Add Marker
- Add Text

- Extract Data (Excel)

5.1 Data analysis

One of the most important factors in the thesis was to develop a GUI to visualize the data from patients being monitored from home. Data analysis can be used to determine if the patient's vitals worsen to the point in which they require intensive care. Therefore, one of the most important features associated with the menu bar is the 'Extract Data' option. This option allows the user to mark an area of interest and extract all data within. The marked intervals and their data are saved into Excel files. With this feature, health care professionals are able to use the Excel files to perform data analysis.

The figures below illustrates how the 'Extract Data' functionality works. When areas of interests are marked as seen in figures 5.10 and 5.11, an interval name is given, and the 'Import to Excel' function is clicked. This creates an Excel file showing an overview of marked intervals and its data, as seen in figures 5.12 and 5.13.

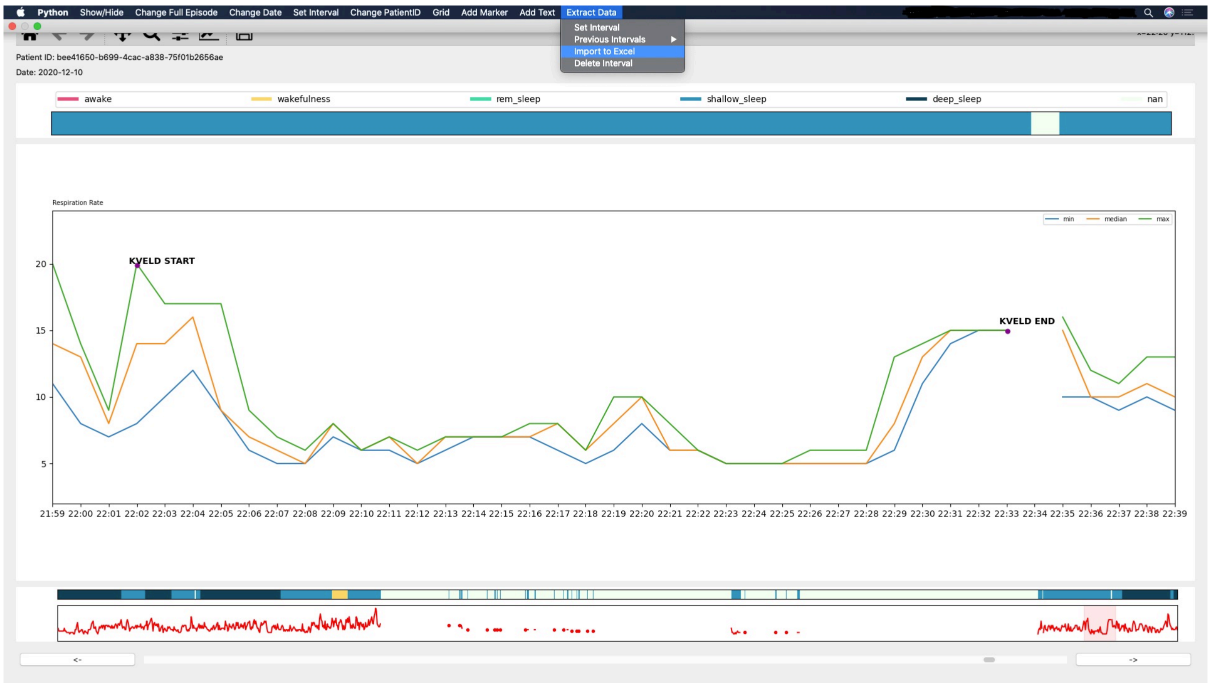

Patient ID: bee41650-b699-4cac-a838-75f01b2656ae
Date: 2020-12-10

Figure 5.10: Extract data function, morning

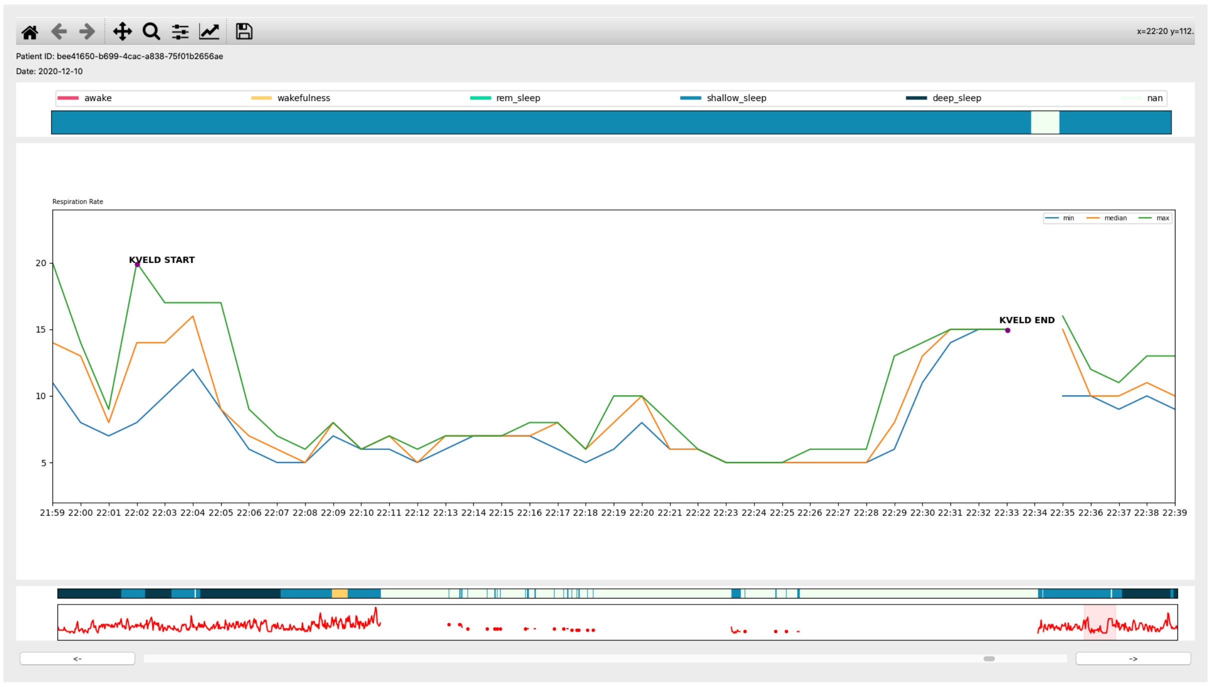
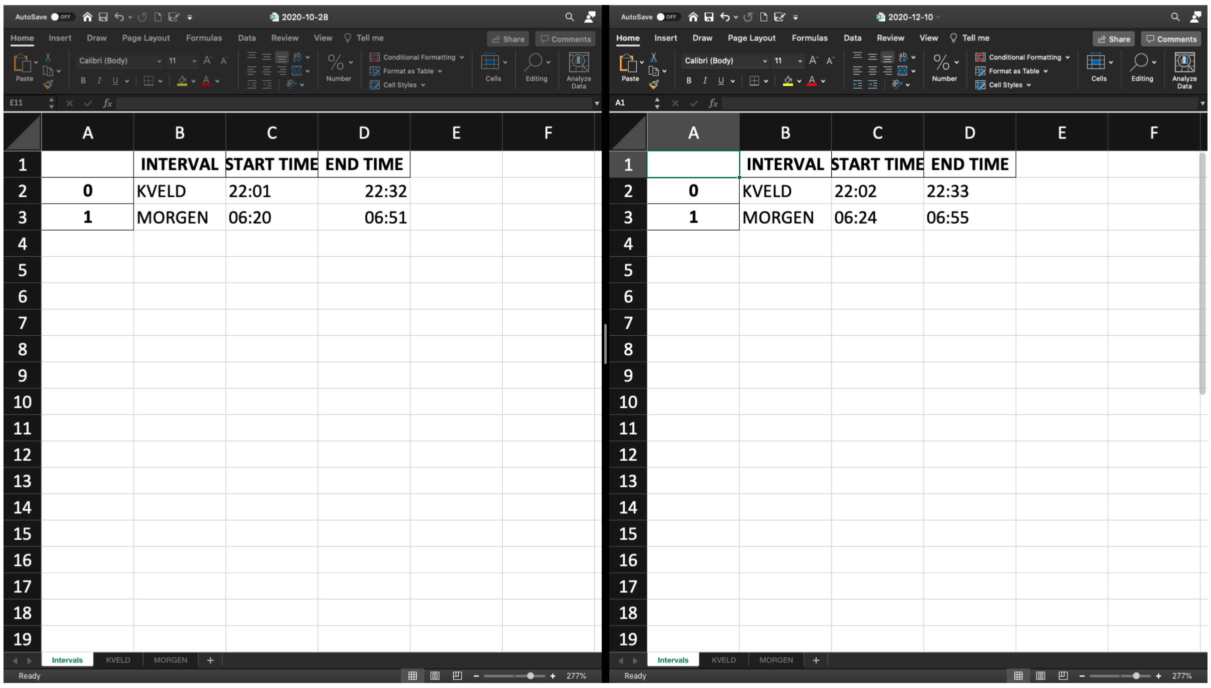

Patient ID: bee41650-b699-4cac-a838-75f01b2656ae
Date: 2020-12-10

Figure 5.11: Extract data function, night

By using built-in Excel functions such as median, average and standard deviations, the patients data can be analyzed. To give an example of how this analysis can be performed, two similar intervals of two different patients are extracted for the purpose of comparison. Patient 1 to the left, and patient 2 to the right, shown in figure 5.13. In the figures below, the excel functions median, average and standard deviation have been used on the respiration rate data set.



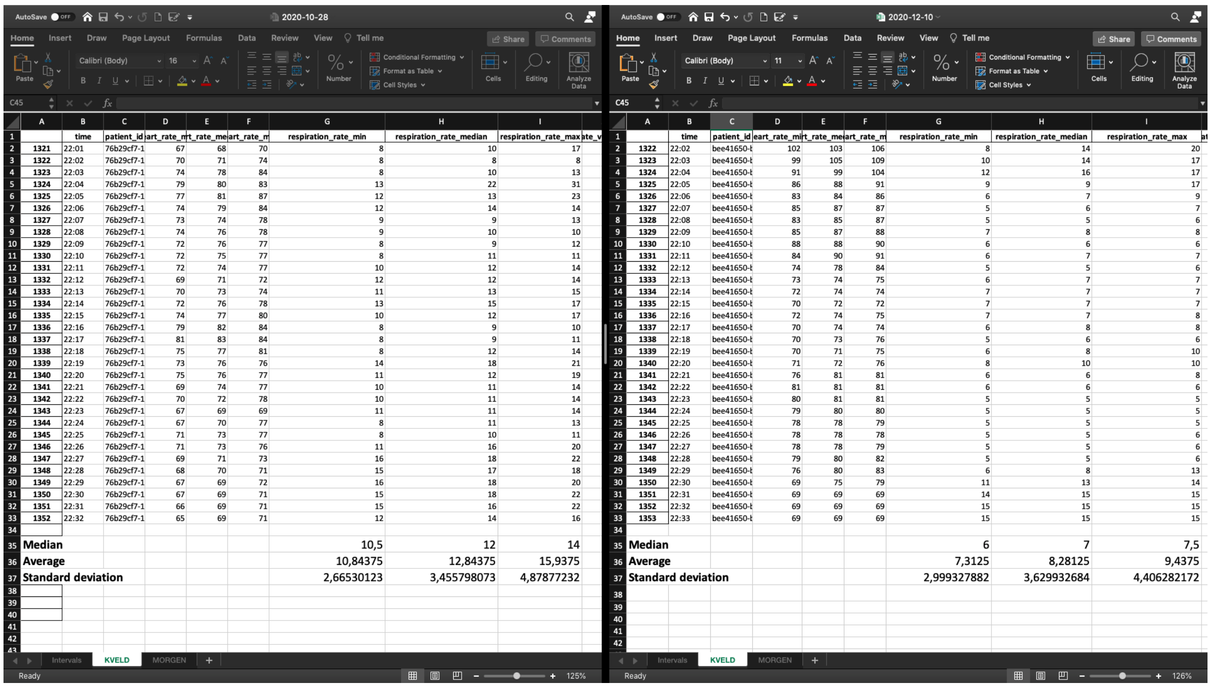
2020-10-28 (Intervals)

	A	B	C	D
1		INTERVAL	START TIME	END TIME
2	0	KVELD	22:01	22:32
3	1	MORGEN	06:20	06:51

2020-12-10 (Intervals)

	A	B	C	D
1		INTERVAL	START TIME	END TIME
2	0	KVELD	22:02	22:33
3	1	MORGEN	06:24	06:55

Figure 5.12: Table showing extracted intervals.



2020-10-28 (KVELD)

	A	B	C	D	E	F	G	H	I	J
1		time	patient_id	art_rate_m	rt_rate_me	art_rate_m	respiration_rate_min	respiration_rate_median	respiration_rate_max	ate_v
2	1321	22:01	76b29cf7-1	67	68	70	8	10	17	
3	1322	22:02	76b29cf7-1	70	71	74	8	8	8	
4	1323	22:03	76b29cf7-1	74	78	84	8	10	13	
5	1324	22:04	76b29cf7-1	79	80	83	13	22	31	
6	1325	22:05	76b29cf7-1	77	81	87	12	13	23	
7	1326	22:06	76b29cf7-1	74	79	84	12	14	14	
8	1327	22:07	76b29cf7-1	73	74	78	9	9	13	
9	1328	22:08	76b29cf7-1	74	76	78	9	10	10	
10	1329	22:09	76b29cf7-1	72	76	77	8	9	12	
11	1330	22:10	76b29cf7-1	72	75	77	8	11	11	
12	1331	22:11	76b29cf7-1	72	74	77	10	12	14	
13	1332	22:12	76b29cf7-1	69	71	72	12	12	14	
14	1333	22:13	76b29cf7-1	70	73	74	11	13	15	
15	1334	22:14	76b29cf7-1	72	76	78	13	15	17	
16	1335	22:15	76b29cf7-1	74	77	80	10	12	17	
17	1336	22:16	76b29cf7-1	79	82	84	8	9	10	
18	1337	22:17	76b29cf7-1	81	83	84	8	9	11	
19	1338	22:18	76b29cf7-1	75	77	81	8	12	14	
20	1339	22:19	76b29cf7-1	73	76	76	14	18	21	
21	1340	22:20	76b29cf7-1	75	76	77	11	12	19	
22	1341	22:21	76b29cf7-1	69	74	77	10	11	14	
23	1342	22:22	76b29cf7-1	70	72	78	10	11	14	
24	1343	22:23	76b29cf7-1	67	69	69	11	11	14	
25	1344	22:24	76b29cf7-1	67	70	77	8	11	13	
26	1345	22:25	76b29cf7-1	71	73	77	8	10	11	
27	1346	22:26	76b29cf7-1	71	73	76	11	16	20	
28	1347	22:27	76b29cf7-1	69	71	73	16	18	22	
29	1348	22:28	76b29cf7-1	68	70	71	15	17	18	
30	1349	22:29	76b29cf7-1	67	69	72	16	18	20	
31	1350	22:30	76b29cf7-1	67	69	71	15	18	22	
32	1351	22:31	76b29cf7-1	66	69	71	15	16	22	
33	1352	22:32	76b29cf7-1	65	69	71	12	14	16	
34										
35	Median						10,5	12	14	
36	Average						10,84375	12,84375	15,9375	
37	Standard deviation						2,66530123	3,455798073	4,87877232	

2020-12-10 (KVELD)

	A	B	C	D	E	F	G	H	I	J
1		time	patient_id	eart_rate_mi	rt_rate_me	art_rate_m	respiration_rate_min	respiration_rate_median	respiration_rate_max	at
2	1322	22:02	bee41650-t	102	103	106	8	14	20	
3	1323	22:03	bee41650-t	99	105	109	10	14	17	
4	1324	22:04	bee41650-t	91	99	104	12	16	17	
5	1325	22:05	bee41650-t	86	88	91	9	9	17	
6	1326	22:06	bee41650-t	83	84	86	6	7	9	
7	1327	22:07	bee41650-t	85	87	87	5	6	7	
8	1328	22:08	bee41650-t	83	85	87	5	5	6	
9	1329	22:09	bee41650-t	85	87	88	7	8	8	
10	1330	22:10	bee41650-t	88	88	90	6	6	6	
11	1331	22:11	bee41650-t	84	90	91	6	7	7	
12	1332	22:12	bee41650-t	74	78	84	5	5	6	
13	1333	22:13	bee41650-t	73	74	75	6	7	7	
14	1334	22:14	bee41650-t	72	74	74	7	7	7	
15	1335	22:15	bee41650-t	70	72	72	7	7	7	
16	1336	22:16	bee41650-t	72	74	75	7	7	8	
17	1337	22:17	bee41650-t	70	74	74	6	8	8	
18	1338	22:18	bee41650-t	70	73	76	5	6	6	
19	1339	22:19	bee41650-t	70	71	75	6	8	10	
20	1340	22:20	bee41650-t	71	72	76	8	10	10	
21	1341	22:21	bee41650-t	76	81	81	6	6	8	
22	1342	22:22	bee41650-t	81	81	81	6	6	6	
23	1343	22:23	bee41650-t	80	81	81	5	5	5	
24	1344	22:24	bee41650-t	79	80	80	5	5	5	
25	1345	22:25	bee41650-t	78	78	79	5	5	5	
26	1346	22:26	bee41650-t	78	78	78	5	5	6	
27	1347	22:27	bee41650-t	78	78	79	5	5	6	
28	1348	22:28	bee41650-t	79	80	82	5	5	6	
29	1349	22:29	bee41650-t	76	80	83	6	8	13	
30	1350	22:30	bee41650-t	69	75	79	11	13	14	
31	1351	22:31	bee41650-t	69	69	69	14	15	15	
32	1352	22:32	bee41650-t	69	69	69	15	15	15	
33	1353	22:33	bee41650-t	69	69	69	15	15	15	
34										
35	Median						6	7	7,5	
36	Average						7,3125	8,28125	9,4375	
37	Standard deviation						2,999327882	3,629932684	4,406282172	

Figure 5.13: Median, average and standard deviations for 'Respiration rate'.

6 Discussion

Figure 5.13 shows the median, average, and standard deviation that has been calculated on the respiration rate dataset for two different patients. The data extracted from the plot are in the same time interval with the same number of measurements. The respiration rate is the number of breaths a person takes per minute. When comparing the calculated results for both patients, it is clear that patient 1 (patient to the left) has a slightly higher average respiration rate than patient 2 (patient to the right). Patient 1 also has a higher median value compared to patient 2. However, patient 2 has a higher standard deviation.

6.1 User-friendliness

The goal was to create a user interface that was intuitive, easy to use, and straightforward to understand for the targeted group. The measured data is currently visualized on a web application developed by Omsyn AS, requiring user login. The visualization is not interactive as it is a static plot that updates for each new incoming data. One wish was to make the GUI more user-friendly and interactive. It was essential to develop a GUI where the users could use a menu bar to specify their settings. The user can make several changes from the menu bar to the GUI, such as changing the values on the axis, marking areas with text and shapes, hiding lines, and changing patient ID. The fact that users can easily specify changes is an important factor in user-friendliness. Being able to store these changes is also an important feature as the users are able to continue where they last left off.

One of the advantages of using Python is that it is an open-source library which means that no license is required to download it. As a result, it is easy to manage and accessible for the users. Installation of Python and other modules is the first point of contact for the users. Therefore, it was essential to use a programming language that was easy and straightforward. Some of the Python modules are dependent on other modules to work. A solution to this problem is to create a simple, and well-documented installation guide as seen in appendix A.

One advantage of using Omsyn's web application is that no modules need to be installed, but it requires the user to have a username and password.

6.2 Choices

6.2.1 Python

Python exists in several versions, and there are many IDE-es available. In this thesis, Python version 3.9 has been used to develop the GUI, as this version is the latest release. PyCharm has been used as the IDE during the development as it provides consistent experience on operating systems like Windows, macOS, and Linux. The PyCharm version used for this thesis was PyCharm Edu. This version is open-source and is free to download.

6.2.2 PyQt

PyQt5 allows the developer to create GUIs by importing various modules to help with the development. Using PyQt, the developer has to implement the layout, design, and content of the GUI. This should be very easy and straightforward if a simple GUI is developed. However, it can be more complicated in advanced GUIs. The GUI in this thesis is built by using widgets and layout managers. However, QT Designer may have been a more manageable choice for developing the GUI in this thesis. This program is a tool for building, designing, and placing QT widgets in GUIs (The Qt Company Ltd, 2021c). Using signals and slots mechanisms, the developer can assign behavior to the widgets developed in Qt Designer. Qt Designer lets the developer build a GUI in fewer lines of programming code. Therefore, Qt Designer could have made the GUI in this thesis less complicated to develop.

6.2.3 Pickle

Pickle was used to implement the functions related to data storage. Although pickle is used to store any objects, it only supports certain types of QT objects to be pickled. This means that all unsupported objects such as QComboBox(), QVBoxLayout(), QInputDialog(), etc., must be saved by storing the values used to initialize the class. Such a method of storing data is time-consuming and makes the program slower. But, pickling dictionaries, Pandas data frames, and integers, pickle was a good option.

6.3 Early development

Good design is fundamental because it creates a link between the users and the program being developed. One of the first drafts of the GUI is drawn in the figure 6.1. The idea was to open a new window for every decision made by the user. The user could only select a patient ID in the first window and then select an episode next. The third window displayed the optional dates for the specific patient, and finally, the fourth window would appear, showing the episode.

There are several weaknesses with the first draft. Developing multiple windows makes it more difficult for the users and could potentially make the program slower. The idea behind the first draft was also only to display the chosen episode, meaning it would either show heart rate, relative stroke volume, heart rate variability, or respiration rate. The idea was only to display a chosen min, max, or median value. This intention would not have created an interactive user experience as it would have been difficult for the user to compare measurements and analyze. Hence, only two different windows were developed. One for selecting episodes and one for visualizations.

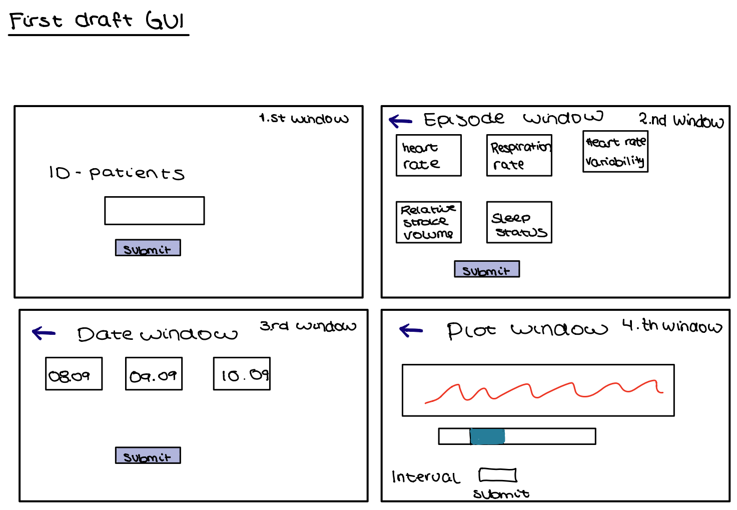


Figure 6.1: First draft of GUI

6.4 Further Development

Further development of this program should include several aspects. Some improvements can make the program faster and better visually. The GUI is developed using the macOS operating system and has therefore adapted to MAC computer screens. A more flexible GUI can be programmed for future developments by testing it on other operating systems before launching. This way, the GUI may adapt to different sizes of computer screens.

One change that can make the GUI more user-friendly is to remove the menu bar and create a configuration window with all the settings. By adding a configuration window, operations will be more visible to the user, and the GUI may be easier to use.

If Omsyn AS were to change the settings on the sensors to measure at 1-second intervals instead of 1-minute intervals, retrieving information about the patient's health condition would be easier. The relevant data can then be used further for more secure data analysis. This change in settings relies on the company providing the sensors, and if this was to be achieved, the backend implementation of file processing must be changed.

For further development, adding a search field would make the GUI more user-friendly. A search field will allow the user to search for a patient ID and a corresponding date instead of being redirected to the first window and going through a combo box drop-down list.

7 Conclusion

The thesis focuses on two main parts; the first part was to process the incoming CSV files. These CSV files contained the data from the patients who participated in the study. The biosensor measured the values of the patients at 1-minute intervals. Furthermore, the data was handled using Pandas data frames. By using Pandas data frames, all CSV files were handled equally, which made the program easier to run.

After handling the data, the data was visualized in four different plots. Another important aspect was to create a GUI where the user could mark areas on the plot that was of interest. Based on the markings, the user could retrieve the data between the marked areas. The data that was retrieved could be used further for data analysis. One of the challenges associated with COVID-19 is the number of hospital admissions. By retrieving data related to the patients and following the patients from home, one can prevent the hospitals from being overloaded.

Part two was to build a GUI that was interactive and user-friendly. The goal was to develop a GUI where the user could change the settings to make the plots more readable. Various functionalities were added to a menu bar. The user can change values on axes, show and hide selected episodes, add grids, mark areas, etc. However, some improvements can make the program run faster and look better visually, such as creating a configuration window or a separate search field to search for an episode.

References

Arnulf, V., Eftestøl, T., Aramendi, E., Zarandona, U. I., Bjorland, H., and Wik, L. (2021). Sensor based vital signs monitoring of patients with clinical manifestation of covid 19 disease during home isolation, a randomized feasibility study. Accessed: 14.05.2021 12:00.

Editorial Team (2020). Best python gui framework. https://techsore.com/best-python-gui/. Accessed: 16.04.2021.

Gavi the vaccine alliance (2020). Why is coronavirus lockdown necessary? https://www.gavi.org/vaccineswork/why-coronavirus-lockdown-necessary. Accessed: 01-03-2021.

GeeksforGeeks (2020). Python language introduction. https://www.geeksforgeeks.org/python-language-introduction/. Accessed: 03.03.2021.

Larsen, B. H., Magelssen, M., Dunlop, O., Pedersen, R., and Førde, R. (2020). Etiske dilemmaer i sykehusene under covid-19-pandemien. *Tidskriftet Den Norske Legeforening*, 18.

Leodanis Pozo Ramos (n.d.). Python and pyqt: Building a gui desktop calculator. https://realpython.com/python-pyqt-gui-calculator/. Accessed: 14.04.2021.

Martin Fitzpatrick (n.d.). Dialogs and alerts. https://www.learnpyqt.com/tutorials/dialogs/. Accessed: 19.04.2021.

Mindfire Solutions (2017). Advantages and disadvantages of python programming language. https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121. Accessed: 14.04.2021.

Omsyn (2021). Sengesensoren redder liv. https://www.omsyn.no/blogg. Accessed: 11.04.2021.

The Qt Company Ltd (2021a). Layout management. https://doc.qt.io/qtforpython/overviews/layout.html. Accessed: 19.04.2021.

The Qt Company Ltd (2021b). Qscrollbar class. https://doc.qt.io/archives/qt-4.8/qscrollbar.html. Accessed: 30.04.2021.

The Qt Company Ltd (2021c). Qt designer manual. https://www.overleaf.com/project/603c9dc204cbe70e1678ec95. Accessed: 26.04.2021.

The Qt Company Ltd (2021d). Qt widgets. https://doc.qt.io/qt-6/qtwidgets-index.html. Accessed: 14.04.2021.

Théo Vanderheyden (2018). Pickle in python: Object serialization. https://www.datacamp.com/community/tutorials/pickle-python-tutorial?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=229765585183&utm_targetid=aud-299261629574:dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=1010917&gclid=Cj0KCQjwyN-DBhCDARIsAFOELTl5LBpi-nxjgwkq7orna642uKrRqaAim5nejeoCVnGG7TpJ0A3xnaoaAjQkE
wcB. Accessed: 15.04.2021.

Tutorialspoint (n.d.). Pyqt5 - signals and slots. https://www.tutorialspoint.com/pyqt5/pyqt5_signals_and_slots.htm. Accessed: 19.04.2021.

Wikipedia (2021). Pandas (software). https://en.wikipedia.org/wiki/Pandas_(software). Accessed: 15.04.2021.

World Health Organization (2020). Coronavirus disease 2019 (covid-19) situation report – 94. https://www.who.int/docs/default-source/coronaviruse/situation-reports/20200423-sitrep-94-covid-19.pdf. Accessed: 01-03-2021.

Appendix

A How to install the programs

To be able to start using the GUI, installation of some programs is required. 'Python' and 'git' must be installed before downloading the source code. The source code is published on GitHub and a README file demonstrating how the program is downloaded through the terminal. The link to where Python and git are downloaded is also posted here.

Link to GitHub repository: https://github.com/COVID-GUI/COVID19

B How to use the program

1. When running the program, a window will be displayed as seen in figure A0.1. The user can then choose which patient ID and date to go forward with.

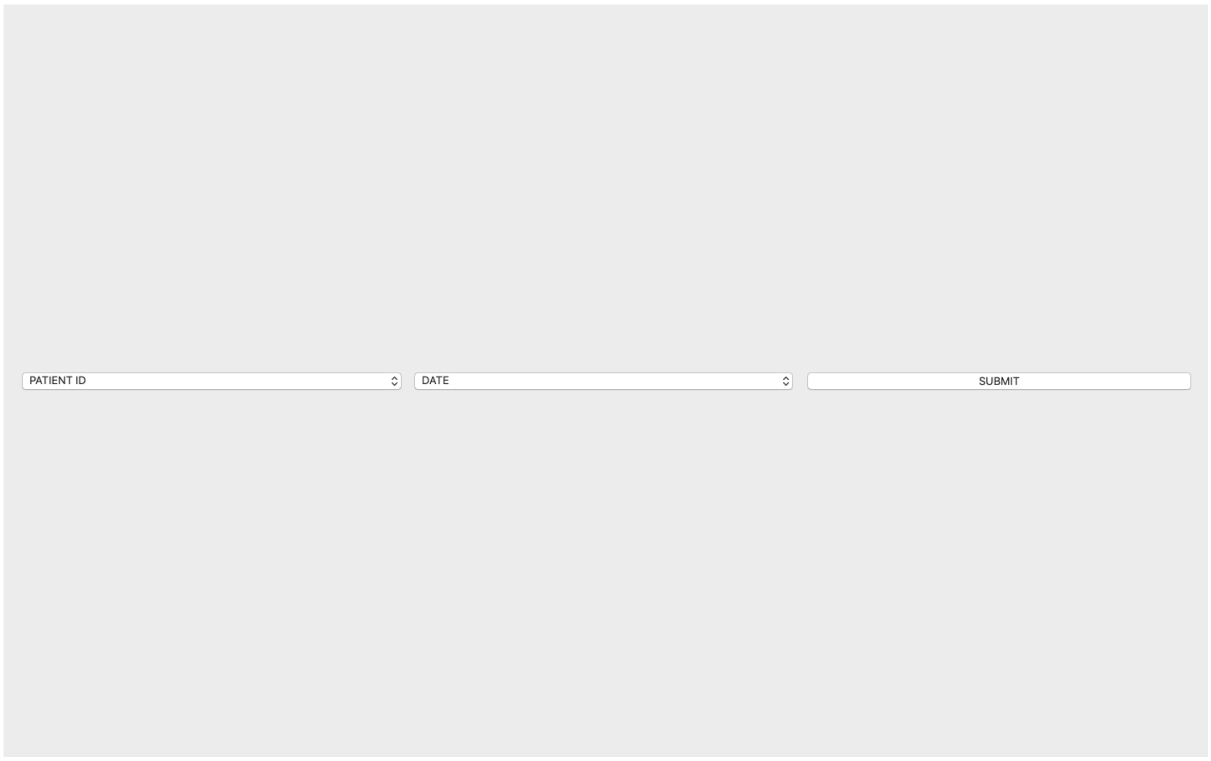


Figure A0.1: Combobox containing patient IDs and dates

2.After choosing a patient ID and a date, a new window with the plotted data will be displayed as seen in figure A0.2.

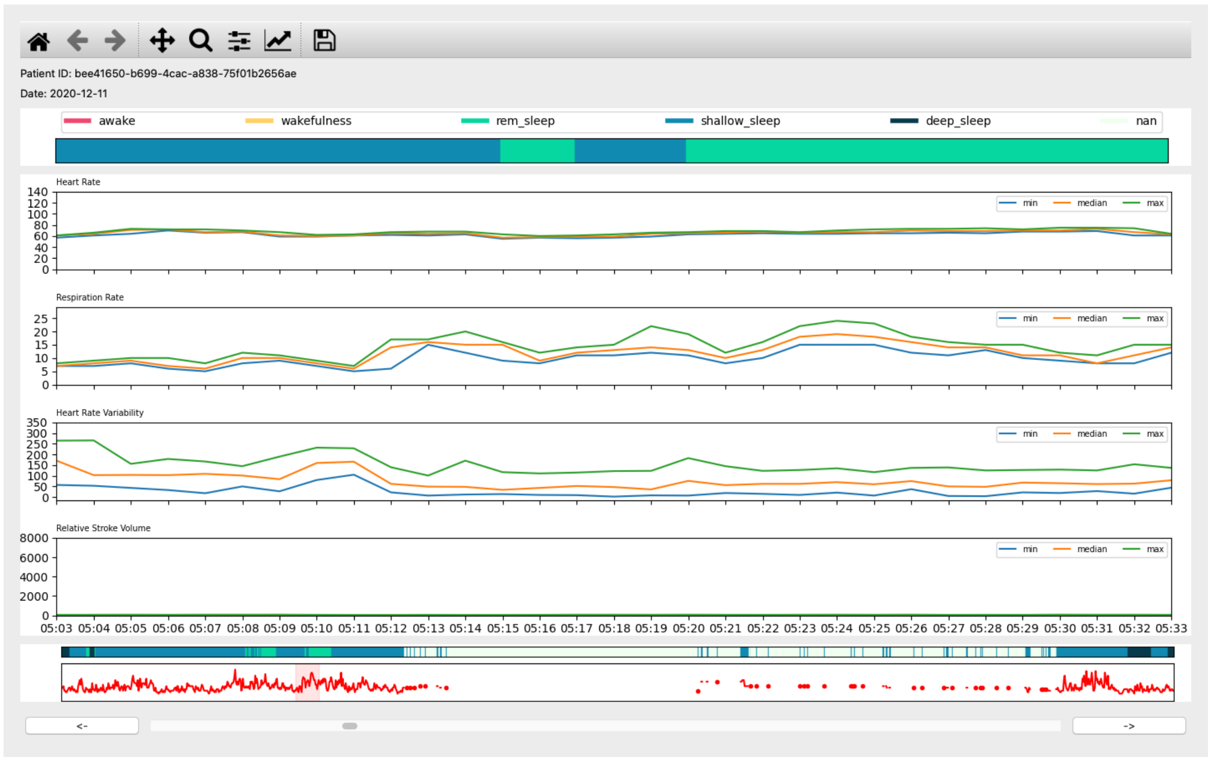


Figure A0.2: The plotted data

3. From the plot, the user has the opportunity to navigate in the episode using the scroll bar. The pink area in the full episode indicates the exact area being showed in the four plots.

4. The user can also select functions from QMenuBar(). The menu bar is displayed differently depending on the operating system used. The functions added to the QMenuBar() are seen in the figure A0.3 below.

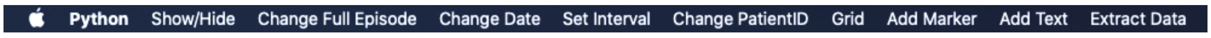


Figure A0.3: Full menu bar

5. From the QMenuBar(), it is possible to Show/Hide plots.

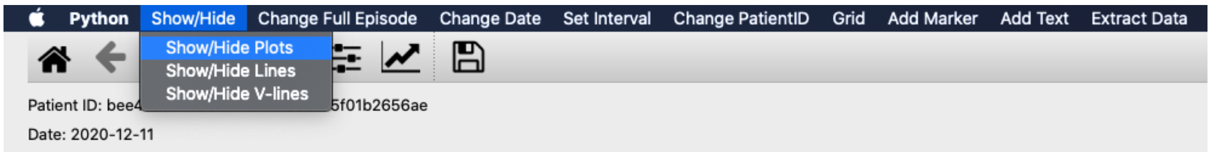


Figure A0.4: The user is able to show/hide plots

6. It is also possible to Change Full Episode

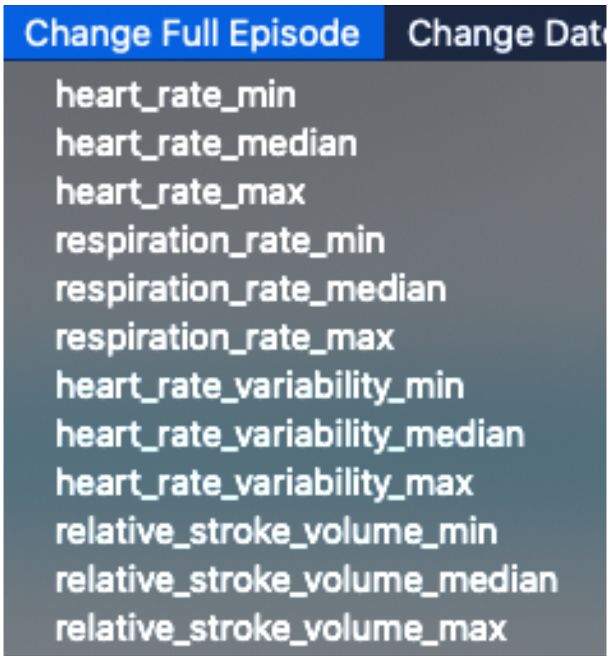

Change Full Episode
Change Dat
heart_rate_min
heart_rate_median
heart_rate_max
respiration_rate_min
respiration_rate_median
respiration_rate_max
heart_rate_variability_min
heart_rate_variability_median
heart_rate_variability_max
relative_stroke_volume_min
relative_stroke_volume_median
relative_stroke_volume_max

Figure A0.5: Change full episode

7. Date and patient ID can be changed by selecting the options in the QMenuBar()

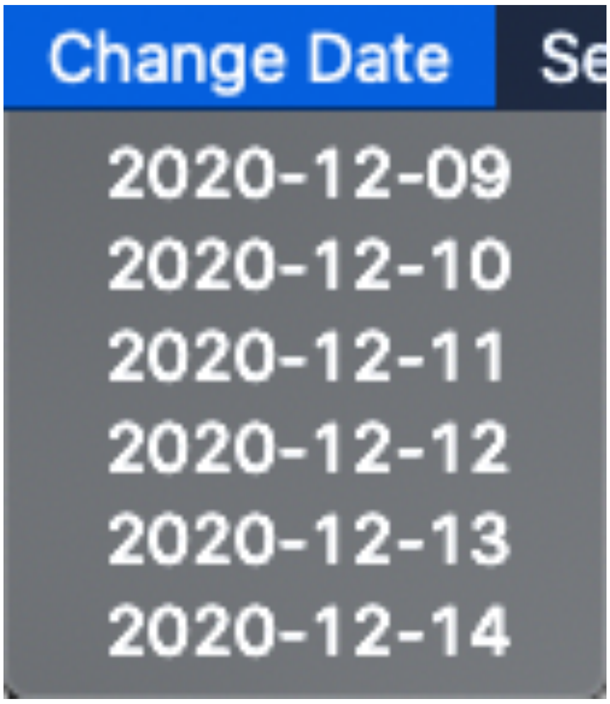


Figure A0.6: Change date

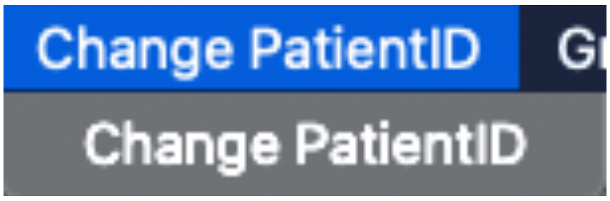


Figure A0.7: Change patient ID

8. By click on set Interval, one can specify new values on both x-axis and y-axis.

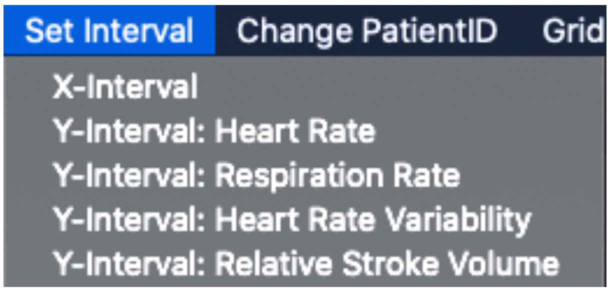

Set Interval
Change PatientID
Grid
X-Interval
Y-Interval: Heart Rate
Y-Interval: Respiration Rate
Y-Interval: Heart Rate Variability
Y-Interval: Relative Stroke Volume

Figure A0.8: Change interval

9. In order to mark areas on the plot with markers or text, click on Add Marker or Add Text in QMenuBar()

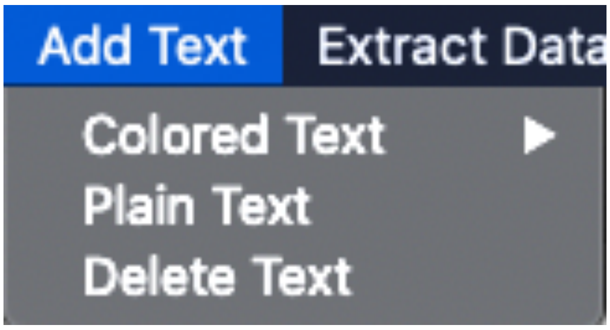


Figure A0.9: Add text

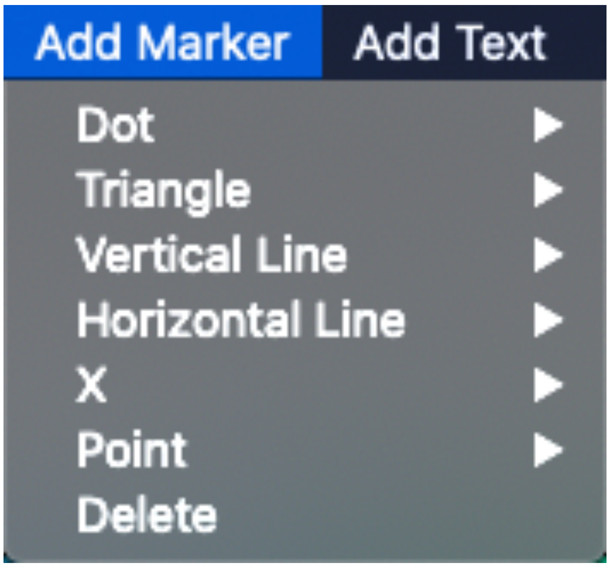


Figure A0.10: Add markers

10. At last, adding grids can be done by pressing the Grid option in the QMenuBar()

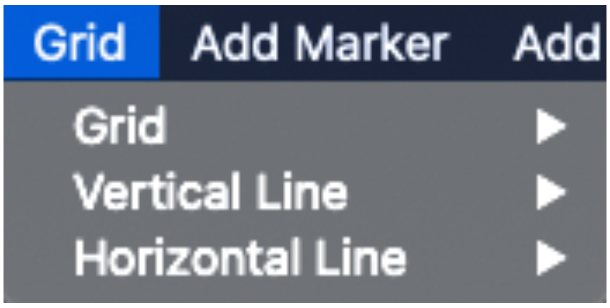


Figure A0.11: Add grid