EEE3017 – Predicting Pigs Behavioural Changes in the Pork Industry and using machine learning with VOCs.

Alex Dowsett

URN: 6585580

Final Report | Final Year Project

A dissertation submitted to the Department of Electronic Engineering in partial fulfilment of the Degree of Bachelor of Engineering in Electronic Engineering with Computer Systems.

GitLab Repository:

https://gitlab.surrey.ac.uk/ad01326/voc

UNIVERSITY OF SURREY

Department of Electrical and Electronic Engineering

Faculty of Engineering and Physical Sciences

University of Surrey

9 August 2023

ABSTRACT

The project's vision is to develop a computer system that can foresee behavioural changes in pigs used in pork production. Using regularly monitored Volatile Organic Compounds (VOCs) data released from the pigs, we train the program to associate patterns of changes within the data with certain changes in the environment. Using pigs under a strict controlled environment on our SRUC farm we can record data and train our network to associate these changes in VOCs to certain behavioural changes. We discuss challenges encountered during the project and results from test data during production of the software.

ACHKNOWLEDGEMENTS

I would like to thank my project supervisor Dr Kevin Wells for providing me with constant support with the project and for extending his support during the LSA period. He was invaluably helpful for helping clear up any questions during the project development.

I would also like to thank Tim Gibson from RoboScientifc for help understanding data collection methods discussing methods to analyse the data.

Thank you to Dr Anjun Dutta for the valuable feedback from the midterm report.

Thank you to Miroslaw Bober for sharing their knowledge of pattern recognition with large, complex data.

Thank you to Martin Fitzpatrick for his many educational python tutorials revolving around PyQt6 and PyPlot.

CONTENTS

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

LIST OF FIGURES

LIST OF TABLES

No table of figures entries found.

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

1 INTRODUCTION

The aim of the project is to develop a software system capable of predicting behavioural changes in the pigs within the pork industry. Over the years, the pork industry has been impacted by disease outbreaks such as swine flu and African swine fever, which can lead to reduced pork production and trade disruptions. The central idea is just like any other animal, when pigs experience pain or stress they emit their volatile organic compounds (VOCs) will be altered. Just like how specially trained medical dogs can smell altercations in VOCs in humans they can predict if a change in a human before are aware themselves. By using these VOC sensors on the farm, we can detect trends in the data that can recognise early disease onset, tail biting and abnormal eating and drinking habits so they can be handled on the farms as early as possible. This report documents the current state of the VOC software in the context of an undergraduate Bachelor of Engineering project. The motivation for this work is due to the rising cost of pork in the UK but more importantly the sustainability of the pork industry with the rising concern of the environmental impacts from meat farming. Whilst research has been established using VOCs since the 1960s.Since 1998 the UK has saw a decline in the number of farms producing pigs, although this has been slowly rising slowly the last few years, it fell once again significantly in 2022.

The main objectives for this project are as follows:

1) To automate renaming, sorting, and restructuring raw data files from the VOC analysers ready to be imported into the software.
2) Create backend software that extracts sensor data from a large amount of text files and stores the data efficiently to the hard disk to eliminate the need to import the data every time the software is executed.
3) Pre-process data by normalisation and standardization and detect and eliminate corrupt data points and abnormal data spikes.
4) Create frontend software with a user-friendly GUI framework to visually plot the data display test information allowing a user to recognize and spot visual trends.
5) Use dimensionality reduction techniques such as PCA and t-SNE to effectively visualize clusters or groups of data points and their relative proximities and display results.
6) Recognize data patterns using this software and pair them with conditions altered in each pen.

3 DESIGN CONSIDERATIONS

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

I have decided to group the design considerations into one section to explain how each part of the project synergise with one another to understand the overall scope of the project more easily. Each requirement of this project was split into a modular design where each python script represents a module. The way each module communicates with one another can be described visually in Figure 3.

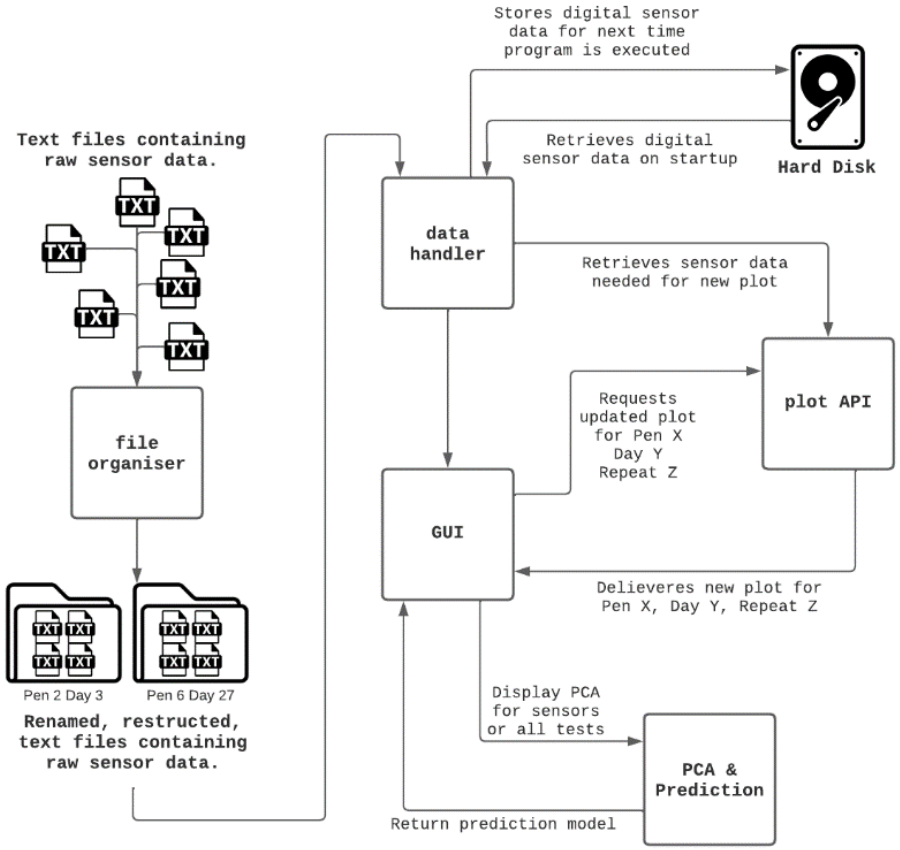


Figure 1: Communication Diagram detailing the basic communications between each module. Data handler imports text files and stores it in a more digitally friendly manner. The GUI is the master program and enables everything to work together while giving an interface for the user to interact with. The plot API is the middleman between the data handler and GUI for visualising the data. The file organiser restructures raw data and is independent of the rest of the system and does not communication with it.

3.1 Back-end data framework

3.2 Data plotting API

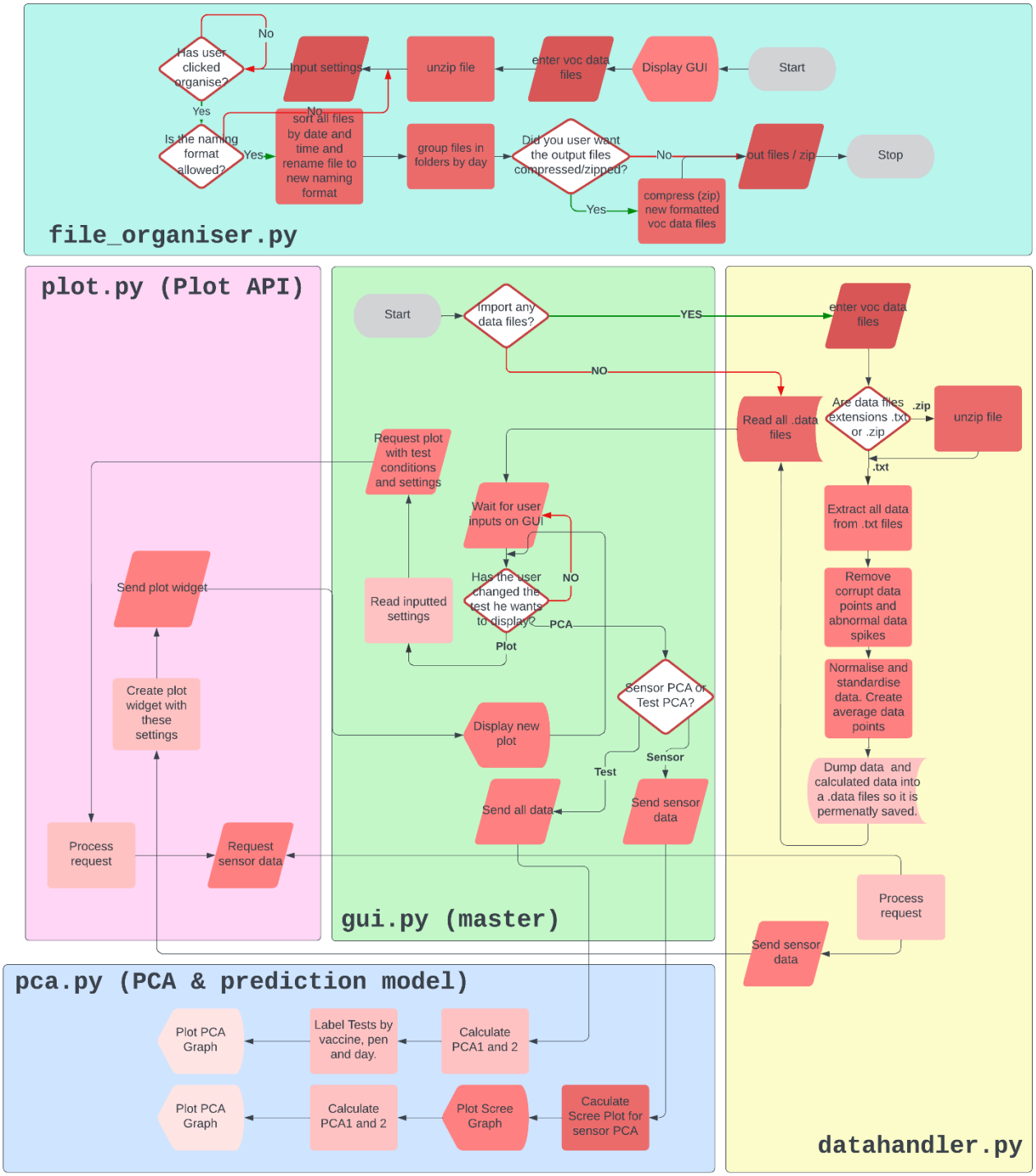


Figure 2: A complete algorithm flowchart describing the simplified tasks and communications of the system. Each coloured section represents a module and its tasks. In reality, each module has more complex duties which can be found detailed in each sections system overview, or alternatively in the python code itself found in Appendix A to E.

3.3 Front-end Graphical User Interface

3.4 Prediction Model

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

4 BACK-END DATA FRAMEWORK

4.1 System overview

A class, `Open()`, was created to take a text file as an argument. This text file is then scanned to find test details such as, time, date, repeats and sensory variables such as baseline, absorb, flow values. Then the actual sensory data is read and put into a 2-dimensional list.

The object then can then be dumped using the classes' function, `.dump()`, into a data file using the python pickle module. These data files are contained within the /data/ folder in the local directory of the python(.py) or executable(.exe) file. On the startup of the program the /data/ folder is scanned and all the sensory data within this folder is loaded into the program automatically. These `Open()` objects are stored in a 3-dimensional dictionary. Sensory data files can then be accessed using three dictionary keys; pen, day, repeat.

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

For example, if we wanted to print the test name, time and the baseline used on the VOC test imported from the Pen5a60_Day 10_14_11_08.43.txt file, we use 'Pen5a60' as the room name, the day is day 10 and the repeat is 4 as this is the 4th repeat of the test.

```
print(self.data['Pen5a60'][10][4].name)
print(self.data['Pen5a60'][10][4].time)
print(self.data['Pen5a60'][10][4].baseline)
```

```
> Pen5a60_Day 10
> 14:11:50
> 2.0
```

If we want to access the sensory data for sensor 3 (index 2) at data point 7 use:

```
print(self.data['Pen5a60'][10][4].data[7][2])
```

```
> 311.819
```

If we cross reference this to our text file, we can see this is the expected value.

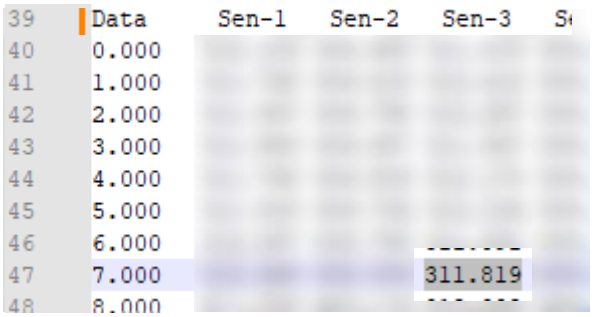


Figure 5: Sensory data snippet from an example text file named 'Pen5a60_Day 10_14_11_08.43.txt'

These data files can be deleted manually by using the Delete Imported Data button in the toolbar which uses the python os module.

4.X Results and discussion

Talk about load up time on import and on second load up

Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

3.3 Data restructuring

3.4 Data cleansing

3.2 Software framework (GUI)

3.5 Data plotting API

3.6 Data standardization

3.7 Sensor weighting

3.8 Descriptor calculation

3.9 Data analysis

4 DATA PLOTTING API

4.1 Design Considerations

4.2 System overview

3.X Results and discussion

Results and discussion

5 FRONT-END GRAPHICAL USER INTERFACE

3.1 Design Considerations

3.2 System overview

3.X Results and discussion

6 PREDICTION MODEL

3.1 Design Considerations

3.2 System overview

dimensional reduction techniques

3.X Results and discussion

Gigo

Limited data

Poor quality data

No way to replicate data.

However promising results (trends starting to form)

6 CONCLUSION AND FUTURE WORK

Linear regression machine learning (least squares regression)

https://www.infoworld.com/article/3695208/14-popular-ai-algorithms-and-their-uses.html#:~:text=Linear%20regression%2C%20also%20called%20least,gradient%20descent%20(see%20below).

7 BIBLIOGRAPHY

[1] A. Chmielowiec-Korzeniowska, L. Tymczyna, Ł. Wlazło, B. Trawińska, and M. Ossowski, De Gruyter Poland, "Emissions of Gaseous Pollutants from Pig Farms and Methods for their Reduction," *Department of Animal Hygiene and Environmental Hazards, University of Life Sciences, Akademicka 13, 20-950Lublin, Poland.*

[2] Bin Yuan, Matthew M. Coggon, Abigail R. Koss, Carsten Warneke, Scott Eilerman, Jeff Peischl, Kenneth C. Aikin, Thomas B. Ryerson, and Joost A. de Gouw, "Emissions of volatile organic compounds (VOCs) from concentrated animal feeding operations (CAFOs): chemical compositions and separation of sources," Atmospheric Chemsitry and Physics, 18 April 2017. [Online]. Available: https://acp.copernicus.org/articles/17/4945/2017/. [Accessed 3 January 2023].

Reference all python modules and GitHub for help on them

8 APPENDICIES

8.1 Appendix A | Python file, file_organiser.py

This python file is a standalone program requested by Tim Gibson at RoboScientifc to rename, sort and restructure a batch of raw data files received from the VOC analysers into a more manageable state. The files are renamed to the day they were sampled, organised by time they sampled and repeats of each test are grouped into individual folders. This saved Tim doing it manually in Windows file explorer. This also reduced the stress of my main software having to perform these tasks before importing.

```
VERSION =      1.1
# DO NOT EDIT
# Impletmented by Alex Dowsett
#################################################################

HELP    = ['\nThis software is designed to organise sensory data files into seperate',
                  'folders where each folder contains the repeats of a test.',
                  '\nHelp:',
                  '\n\u2022 The source folder should contain all the textfiles (.txt) of sensory
data.',
                  '\n\u2022 The destination folder is where the the organised files are
exported.',
                  "\n\u2022 The 'Rename test name' box can be used to change a file's name or\n
experiment's name. The following text substitutions can be used:\n   '<number>', '<time>',
<date>' which will be substitued appropriately\n   depending on the data within the particular
text file.",
                  "\n    - If you wish to keep the original file name untick the 'Apply name
change to\n       file name' checkbox.",
                  "    - If you wish to keep the original experiment name untick the 'Apply
name\n       change to experiment name' checkbox.",
                  "\n\u2022 If you wish to compress the output folder as a .zip file tick the
'Compress\n    output folder' checkbox.\n"
                 ]

AUTHOR  = ['\t\t\t   University of Surrey            ',
                  '\t\t               Created by Alex Dowsett             ',
                 ]


#################################################################

from PyQt6 import QtCore, QtGui, QtWidgets
import PyQt6
import sys
import tempfile
```

```
from warnings import warn
from datetime import datetime
import os
from distutils.dir_util import copy_tree
from shutil import make_archive
import ctypes


#####################################################################

images_dir = 'images/'
icon_path = images_dir+'icon.ico'
logo_path = images_dir+'logo.png'
folder_im_path = images_dir+'folder-import.png'
folder_ex_path = images_dir+'folder-export.png'
question_path = images_dir+'question.png'

#####################################################################

try: # If run through PyInstaller
        icon_path = os.path.join(sys._MEIPASS, icon_path)
        logo_path = os.path.join(sys._MEIPASS, logo_path)
        folder_im_path = os.path.join(sys._MEIPASS, folder_im_path)
        folder_ex_path = os.path.join(sys._MEIPASS, folder_ex_path)
        question_path = os.path.join(sys._MEIPASS, question_path)
        myappid = u'mycompany.myproduct.subproduct.version' # arbitrary string
        ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(myappid)

except AttributeError:
        pass


#####################################################################

def main():

        global win
        app = QtWidgets.QApplication(sys.argv)
        app.setWindowIcon(QtGui.QIcon(icon_path))
        win = Window()
        win.show()
        sys.exit(app.exec())

#####################################################################

class Window(QtWidgets.QMainWindow):
    '''Main Window.'''
    def __init__(self, parent=None):
        '''Initializer.'''
        try:
            self.isVisible()
        except RuntimeError:
            super().__init__(parent)

        # Window Config
        self.setWindowTitle("Sensor File Organiser")
        self.resize(350, 500)
        self.setFixedSize(350, 500)
        self.setWindowIcon(QtGui.QIcon(logo_path))

        self.statusBar = QtWidgets.QStatusBar(self)
        self.setStatusBar(self.statusBar)
        self.statusBar.showMessage('')

        # Fonts
        self.font = QtGui.QFont()
        self.font.setBold(True)
        self.font.setPointSize(11)

        self.font2 = QtGui.QFont()
        self.font2.setItalic(True)
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
        # Confirmation Button
        self.confirmB = QtWidgets.QPushButton("Convert files", self)
        self.confirmB.setGeometry(QtCore.QRect(25, 210, 300, 40))
        self.confirmB.clicked.connect(self.confirm)
        self.confirmB.setStatusTip('Confirm and export organised files to destination folder.')

               # Source Folder Widgets
        self.fileL = QtWidgets.QLabel("Select source folder:", self)
        self.fileL.setGeometry(QtCore.QRect(25, 25, 325, 25))
        self.fileL.setFont(self.font)

        self.fileE = QtWidgets.QLineEdit("Select folder containing sensor data files...", self)
        self.fileE.setGeometry(QtCore.QRect(25, 50, 300, 25))
        self.fileE.setReadOnly(True)
        self.fileE.defaultMousePressEvent = self.fileE.mousePressEvent
        self.fileE.mousePressEvent = self.file
        self.fileE.setFont(self.font2)
        self.fileE.setStatusTip('Source folder directory.')

        self.fileB = QtWidgets.QPushButton("Select Folder", self)
        self.fileB.setGeometry(QtCore.QRect(201, 75, 125, 25))
        self.fileB.clicked.connect(self.file)
        self.fileB.setStatusTip('Select a source folder containing the sensor data files.')

        self.fileI = QtWidgets.QLabel(self)
        self.fileI.setGeometry(QtCore.QRect(6, 51, 25, 25))
        self.filePixMap = QtGui.QPixmap(folder_im_path)
        self.fileI.setPixmap(self.filePixMap)

        # Destination Folder Widgets
        self.fileL2 = QtWidgets.QLabel("Select destination folder:", self)
        self.fileL2.setGeometry(QtCore.QRect(25, 110, 325, 25))
        self.fileL2.setFont(self.font)

        self.fileE2 = QtWidgets.QLineEdit(os.path.expanduser('~/Downloads').replace('\\', '/'),
self)
        self.fileE2.setGeometry(QtCore.QRect(25, 135, 300, 25))
        self.fileE2.setStatusTip('Destination folder directory.')

        self.fileB2 = QtWidgets.QPushButton("Select Folder", self)
        self.fileB2.setGeometry(QtCore.QRect(201, 160, 125, 25))
        self.fileB2.clicked.connect(self.file2)
        self.fileB2.setStatusTip('Select a destination where the organised files will be
exported.')

        self.fileI2 = QtWidgets.QLabel(self)
        self.fileI2.setGeometry(QtCore.QRect(6, 136, 25, 25))
        self.file2PixMap = QtGui.QPixmap(folder_ex_path)
        self.fileI2.setPixmap(self.file2PixMap)

        # Options
        self.fileL = QtWidgets.QLabel("Settings (optional):", self)
        self.fileL.setGeometry(QtCore.QRect(25, 260, 325, 25))
        self.fileL.setFont(self.font)

        self.compressC = QtWidgets.QCheckBox("Compress output folder", self)
        self.compressC.setGeometry(QtCore.QRect(30, 400, 170, 25))
        self.compressC.setStatusTip('Check this box to compress the output as a .zip file.')

        self.font3 = QtGui.QFont()
        self.font3.setPointSize(8)

        self.changeNameE = QtWidgets.QLineEdit("Day <number> <date> <time>", self)
        self.changeNameE.setGeometry(QtCore.QRect(25, 313, 300, 25))
        self.changeNameE.setStatusTip('Enter name format here.')
        self.changeNameE.setFont(self.font3)

        self.changeNameL = QtWidgets.QLabel("Rename test name:", self)
        self.changeNameL.setGeometry(QtCore.QRect(25, 290, 300, 25))
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
        self.changeFileNameC = QtWidgets.QCheckBox("Apply name change to file name", self)
        self.changeFileNameC.setGeometry(QtCore.QRect(50, 340, 230, 25))
        self.changeFileNameC.setStatusTip('Check to rename the text files with this format.')
        self.changeFileNameC.toggled.connect(self.checkbox)
        self.changeFileNameC.setChecked(True)

        self.changeExperimentNameC = QtWidgets.QCheckBox("Apply name change to experiment name",
self)
        self.changeExperimentNameC.setGeometry(QtCore.QRect(50, 363, 250, 25))
        self.changeExperimentNameC.setStatusTip('Check to rename the experiment names with this
format.')
        self.changeExperimentNameC.toggled.connect(self.checkbox)
        self.changeExperimentNameC.setChecked(True)

        # Information
        self.helpI = QtWidgets.QLabel(self)
        self.helpI.setGeometry(QtCore.QRect(328, 469, 16, 16))
        self.helpPixMap = QtGui.QPixmap(question_path)
        self.helpI.mousePressEvent = self.help
        self.helpI.setStatusTip('Click here for help and information.')
        self.helpI.setPixmap(self.helpPixMap)

        self.font4 = QtGui.QFont()
        self.font4.setPointSize(7)

        self.verL = QtWidgets.QLabel("Version: "+str(VERSION), self)
        self.verL.setGeometry(QtCore.QRect(275, 467, 50, 20))
        self.verL.setStatusTip('Software Version.')
        self.verL.setFont(self.font4)

################################################################

    def msgbox(self, title, message, type=''):
        msg = QtWidgets.QMessageBox(parent=self, text=message+'\n')

        if type.casefold() == 'warning':
                msg.setIcon(QtWidgets.QMessageBox.Icon.Warning)
        elif type.casefold() == 'error':
                msg.setIcon(QtWidgets.QMessageBox.Icon.Critical)
        elif type.casefold() == 'information':
                msg.setIcon(QtWidgets.QMessageBox.Icon.Information)
        else:
                msg.setIcon(QtWidgets.QMessageBox.Icon.NoIcon)
        if type != '':
                title = ': ' + title
        msg.setWindowTitle(type.capitalize() + title)
        msg.exec()

    def checkbox(self):
        if self.changeFileNameC.isChecked() or self.changeExperimentNameC.isChecked():
                self.changeNameE.setStyleSheet("color: black;")
                self.changeNameE.setReadOnly(False)
        else:
                self.changeNameE.setStyleSheet("color: gray;")
                self.changeNameE.setReadOnly(True)


    def confirm(self):
        dirname = self.fileE.text()
        if dirname == "Select folder containing sensor data files...":
                self.msgbox("Source folder not found", "Please select the source folder that
contain the sensor data files.", 'warning')
                return

        if not os.path.exists(dirname):
                self.msgbox("Source folder not found", "Source folder does not exist.\nPlease
check if the folder directory is correct.", 'error')
                return

        if not os.path.exists(self.fileE2.text()):
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                self.msgbox("Destination folder not found", "Destination folder does not
exist.\nPlease check if the folder directory is correct.", 'error')
                return

        name = self.changeNameE.text()

        if name.strip() == '':
                self.changeFileNameC.setChecked(False)
                self.changeExperimentNameC.setChecked(False)

        if (self.changeFileNameC.isChecked() or self.changeExperimentNameC.isChecked()) and not
(( '<number>' in name) or  ( ('<date>' in name) and ('<time>' in name)     )       ):
                print('heelo)')
                self.msgbox('Rename format not valid', "The new name must meet one of the
following conditions to the avoid file names clashing:\n \u2022 contain '<number>'\n \u2022
contain '<time>'' and '<date>'", 'error')
                return

        temp = name.replace('<number>', '').replace('<date>','').replace('<time>','')
        illegalChars = set('\/:*?"<>|')

        if '<' in temp or '>' in temp:
                self.msgbox('Incorrect use of angle brackets', 'Angle brackets ('<' and '>') can
only be used with the following text substitution: \n\t\u2022  <number>\n\t\u2022
<time>\n\t\u2022  <date>', 'error')
                return

        if any((c in illegalChars) for c in temp):
                self.msgbox('File name contains illegal characters', 'A file name cannot contain
any of the following characters:\n\t\u2022  \\/:*?"<>|', 'error')
                return


        files = os.listdir(dirname)

        dataFiles = []
        for file in files:
                dataFiles.append([DataFile(file, dirname)])
                try:
                        dataFiles[-1].append(dataFiles[-1][0].timestamp)
                except AttributeError:
                        return

        dataFiles = sorted(dataFiles, key=lambda x: x[1])


        with tempfile.TemporaryDirectory() as tmpdirname:
                print('Created temporary directory at ', tmpdirname)
                tmpdirname = tmpdirname.replace('\\', '/')
                container = datetime.now().strftime("/sensor files %d-%m-%Y %H.%M.%S/")


                lastRepeat = 9999
                group = 0
                for dataFile in dataFiles:

                        if dataFile[0].repeat <= lastRepeat:
                                group += 1

                                if self.changeFileNameC.isChecked():
                                        n = name.replace('<number>', str(group)).replace('<date>',
dataFile[0].date.replace('/', '-')).replace('<time>', dataFile[0].time.replace(':', '.'))
                                else:
                                        n = dataFile[0].fileName

                                dirname = tmpdirname + container + n + '/'
                                os.makedirs(dirname, exist_ok=True)


                        if self.changeFileNameC.isChecked():
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                                n = name.replace('<number>', str(group)).replace('<date>',
dataFile[0].date.replace('/', '-')).replace('<time>', dataFile[0].time.replace(':', '.'))
                        else:
                                n = dataFile[0].fileName

                        if self.changeFileNameC.isChecked():
                                destinationFile = dirname + n + ' Repeat
{}.txt'.format(dataFile[0].repeat)
                        else:
                                destinationFile = dirname + n

                        if self.changeExperimentNameC.isChecked():
                                        experimentName = name.replace('<number>',
str(group)).replace('<date>', '').replace('<time>', '').strip()
                                        dataFile[0].lines[dataFile[0].experimentNameIndex] =
dataFile[0].experimentNamePrefix + '= "' + experimentName + '"\n'
                        f = open(destinationFile, 'w')
                        f.writelines(dataFile[0].lines)
                        f.close()

                        lastRepeat = dataFile[0].repeat

                if self.compressC.isChecked():
                        make_archive(self.fileE2.text()+'/'+container.replace('/',''), 'zip',
tmpdirname+container)
                else:
                        copy_tree(tmpdirname, self.fileE2.text())

        del dataFiles
        print('Conversion complete.')
        self.msgbox("Export complete", "Success!\n\nThe organised files have been exported to the
destination folder:\t\n{}{}".format(self.fileE2.text(), container), 'information')


    def file(self, *args):
        folder = str(QtWidgets.QFileDialog.getExistingDirectory(self, "Select Source Folder
Directory"))
        if not folder:
                return

        self.fileE.setText(folder)
        self.fileE.setReadOnly(False)
        self.fileE.mousePressEvent = self.fileE.defaultMousePressEvent
        self.font2.setItalic(False)
        self.fileE.setFont(self.font2)

    def file2(self, *args):
        folder = str(QtWidgets.QFileDialog.getExistingDirectory(self, "Select Destination Folder
Directory"))
        if not folder:
                return

        self.fileE2.setText(folder)

    def help(self, *args):

        self.msgbox("Help and information", ( '\n'.join(HELP)        )+'\n\n----------------------
------------------------------------------------------------\n\n'+(    '\n'.join(AUTHOR)
        )       +'\n')

#################################################################

class DataFile:
        def __init__(self, filename, dirname):

                self.fileName = filename
                self.longFileName = dirname + '/' + filename


                try:
                        f = open(self.longFileName, 'r')
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                        lines = f.readlines()
                        f.close()

                        err = True
                        i = 0
                        for line in lines:
                                if 'name' in line.casefold() and 'experiment' in line.casefold():
                                        try:
                                                temp = line.split('=')
                                                self.experimentNameIndex = i
                                                self.experimentNamePrefix = temp[0]
                                                err = False
                                        except:
                                                pass
                                        break
                                i += 1

                        if err and (win.changeFileNameC.isChecked() or
win.changeExperimentNameC.isChecked()):
                                warn('File ' + self.fileName + ': Experiment name not found.')
                                win.msgbox('Experiment name not found', "Experiment name not found
for file '{}'.\nAdd 'Name of the experiment =' field to this sensor text file\nin order to rename
this file.\nAlternatively turn the rename options off in Settings.".format(self.fileName),
'error')
                                return


                        err = True
                        for line in lines:
                                if 'repeat' in line.casefold() and ('no' in line.casefold() or
'number' in line.casefold()):
                                        try:
                                                temp = line.split('=').pop().strip().split('/')
                                                self.repeat = int(temp[0])
                                                self.noRepeats = int(temp[1])
                                                err = False
                                        except:
                                                pass
                                        break

                        if err:
                                warn('File ' + self.fileName + ': Repeats not found.')
                                win.msgbox('Repeats not found', "Repeats value not found for file
'{}'.\nAdd 'Repeats =' field in the format 'X/Y' to this sensor text file in order to include
this file.".format(self.fileName), 'error')
                                return

                        err = True
                        for line in lines:
                                if 'time' in line.casefold() and '=' in line.casefold():
                                        try:
                                                self.time = line.split('=').pop().strip()
                                                err = False
                                        except:
                                                pass
                                        break

                        if err:
                                warn('File ' + self.fileName + ': Time not found.')
                                win.msgbox('Time not found', "Time value not found for file
'{}'.\nAdd 'Time =' field in the format 'HH:MM:SS' to this sensor text file in order to include
this file.".format(self.fileName), 'error')
                                return

                        err = True
                        for line in lines:
                                if 'date' in line.casefold() and '=' in line.casefold():
                                        try:
                                                self.date = line.split('=').pop().strip()
                                                err = False
                                        except:
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                                        pass
                                    break

                    if err:
                            warn('File ' + self.fileName + ': Date not found.')
                            win.msgbox('Date not found', "Date value not found for file
'{}'.\nAdd 'Date =' field in the format 'DD/MM/YYYY' to this sensor text file in order to include
this file.".format(self.fileName), 'error')
                            return

                    self.datetime = datetime.strptime((self.date + ' | ' + self.time),
'%d/%m/%Y | %H:%M:%S')
                    self.timestamp = self.datetime.timestamp()
                    self.lines = lines


            except FileNotFoundError as e:
                    warn('File ' + self.fileName + ': Unreadable file / file not found.')

#################################################################

if __name__ == "__main__":
        main()
```

8.2 Appendix B | Python file, gui.py

This python file is the executable and scheduler for the software and incorporates all the other python files/modules excluding the file_organiser.py file. It also controls all the front-end programming such as Graphical User Interface elements.

```
from PyQt6 import QtCore, QtGui, QtWidgets

from datahandler import open_files, load, avgload
from plot import MplCanvas, MplWidget
from PCA import Calculate_PCA, Calculate_PCA_2
from time import sleep
import pandas as pd
import numpy as np
import re
import os

def main():

    #profiling();return

    import sys
    app = QtWidgets.QApplication(sys.argv)
    global win
    win = Window()
    win.show()

    sys.exit(app.exec())


def profiling():
    import sys
    import cProfile
    import pstats

    with cProfile.Profile() as pr:
        app = QtWidgets.QApplication(sys.argv)
        global win
        win = Window()
        win.show()
```

```
        app.exec()

    stats = pstats.Stats(pr)
    stats.sort_stats(pstats.SortKey.TIME)
    #stats.print_stats()
    stats.dump_stats(filename='stats.prof')


class Window(QtWidgets.QMainWindow):
    '''Main Window.'''
    def __init__(self, parent=None):
        '''Initializer.'''
        try:
            self.isVisible()
        except RuntimeError:
            super().__init__(parent)

        self.setWindowTitle("VOC Tool")
        self.resize(800, 600)
        self.setFixedSize(800, 600)
        self.setWindowIcon(QtGui.QIcon('logo.png'))

        self.centralwidget = QtWidgets.QWidget(self)
        self.centralwidget.setObjectName("centralwidget")

        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
        self.tabWidget.setGeometry(QtCore.QRect(0, 0, 800, 600))
        self.tabWidget.setObjectName("tabWidget")
        self.tabWidget.setMovable(True)
        #self.tabWidget.setDocumentMode(True)
        #self.tabWidget.tabBarClicked(self.show_wait)
        self.tabWidget.blockSignals(True)
        self.tabWidget.currentChanged.connect(lambda x: self.update_plot())

        self.setCentralWidget(self.centralwidget)

        self.menubar = QtWidgets.QMenuBar(self)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 20))
        self.menubar.setObjectName("menubar")

        self.menuFile = QtWidgets.QMenu(self.menubar)
        self.menuFile.setObjectName("menuFile")

        self.menuEdit = QtWidgets.QMenu(self.menubar)
        self.menuEdit.setObjectName("menuEdit")

        self.menuSettings = QtWidgets.QMenu(self.menubar)
        self.menuSettings.setObjectName("menuSettings")

        self.menuWindow = QtWidgets.QMenu(self.menubar)
        self.menuWindow.setObjectName("menuWindow")

        self.menuHelp = QtWidgets.QMenu(self.menubar)
        self.menuHelp.setObjectName("menuHelp")

        self.menuView = QtWidgets.QMenu(self.menubar)
        self.menuView.setObjectName("menuView")

        self.setMenuBar(self.menubar)

        self.statusbar = QtWidgets.QStatusBar(self)
        self.statusbar.setObjectName("statusbar")
        self.setStatusBar(self.statusbar)
        self.statusbar.showMessage('')

        self.actionOpen = QtGui.QAction(self)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("icons/icons/folder-open-document.png"),
QtGui.QIcon.Mode.Normal, QtGui.QIcon.State.Off)
        self.actionOpen.setIcon(icon)
        self.actionOpen.setObjectName("actionOpen")
```

```
        self.actionOpen.triggered.connect(self.openfile)

        self.actionOpen_folder = QtGui.QAction(self)
        icon3 = QtGui.QIcon()
        icon3.addPixmap(QtGui.QPixmap("icons/icons/folder-open.png"), QtGui.QIcon.Mode.Normal,
QtGui.QIcon.State.Off)
        self.actionOpen_folder.setIcon(icon3)
        self.actionOpen_folder.setObjectName("actionOpen_folder")
        self.actionOpen_folder.triggered.connect(self.openfolder)

        self.actionQuit = QtGui.QAction(self)
        self.actionQuit.setObjectName("actionQuit")
        self.actionQuit.triggered.connect(self.close)

        self.actionAverage = QtGui.QAction(self)
        self.actionAverage.setObjectName("actionAverage")
        self.actionAverage.setCheckable(True)
        self.actionAverage.triggered.connect(self.show_avg)
        self.avg = False

        self.actionNorm = QtGui.QAction(self)
        self.actionNorm.setObjectName("actionNorm")
        self.actionNorm.setCheckable(True)
        self.actionNorm.triggered.connect(self.show_norm)
        self.shownorm = False

        self.actionMinimise = QtGui.QAction(self)
        self.actionMinimise.setObjectName("actionMinimise")
        self.actionMinimise.triggered.connect(self.showMinimized)

        self.actionAnnotate = QtGui.QAction(self)
        self.actionAnnotate.setObjectName("actionAnnotate")
        self.actionAnnotate.setCheckable(True)
        self.actionAnnotate.setChecked(True)
        self.actionAnnotate.triggered.connect(self.show_annotate)
        self.annotate = True


        self.actionShowwait = QtGui.QAction(self)
        self.actionShowwait.setObjectName("actionShowwait")
        self.actionShowwait.setCheckable(True)
        self.actionShowwait.triggered.connect(self.show_wait)
        self.showwait = False

        #self.actionExport_to_excel = QtGui.QAction(self)
        #icon1 = QtGui.QIcon()
        #icon1.addPixmap(QtGui.QPixmap("icons/icons/document-excel-table.png"),
QtGui.QIcon.Mode.Normal, QtGui.QIcon.State.Off)
        #self.actionExport_to_excel.setIcon(icon1)
        #self.actionExport_to_excel.setObjectName("actionExport_to_excel")

        self.actionDelete_all_imports = QtGui.QAction(self)
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap("icons/icons/table--minus.png"), QtGui.QIcon.Mode.Normal,
QtGui.QIcon.State.Off)
        self.actionDelete_all_imports.setIcon(icon2)
        self.actionDelete_all_imports.setObjectName("actionDelete_all_imports")
        self.actionDelete_all_imports.triggered.connect(self.deleteimports)

        self.actionPreferences = QtGui.QAction(self)
        self.actionPreferences.setObjectName("actionPreferences")

        self.actionHelp = QtGui.QAction(self)
        self.actionHelp.setObjectName("actionHelp")

        self.actionLicense = QtGui.QAction(self)
        self.actionLicense.setObjectName("actionLicense")

        self.noneLabel = QtWidgets.QLabel(self, text='No data found.\nGo to File -> Import... to
get started.')
        self.noneLabel.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
```

```
self.noneLabel.setGeometry(QtCore.QRect(300, 200, 200, 200))
self.noneLabel.setObjectName("None Label")

#self.avgCheckBox = QtWidgets.QCheckBox(self, text="Average")
#self.avgCheckBox.setGeometry(QtCore.QRect(705, 548, 100, 20))
#self.avgCheckBox.setChecked(True)
#self.avgCheckBox.stateChanged.connect(self.show_avg)
#self.avgCheckBox.setStatusTip('If enabled, data is averaged for each day')
#self.showavg = True

self.menuFile.addAction(self.actionOpen)
self.menuFile.addAction(self.actionOpen_folder)
self.menuFile.addSeparator()
#self.menuFile.addAction(self.actionExport_to_excel)
self.menuFile.addAction(self.actionDelete_all_imports)
self.menuFile.addSeparator()
self.menuFile.addAction(self.actionQuit)

self.menuEdit.addAction(self.actionAverage)
self.menuEdit.addAction(self.actionNorm)

self.menuView.addAction(self.actionAnnotate)
self.menuView.addAction(self.actionShowwait)

self.menuSettings.addAction(self.actionPreferences)

self.menuWindow.addAction(self.actionMinimise)
self.menuHelp.addAction(self.actionHelp)
self.menuHelp.addSeparator()
self.menuHelp.addAction(self.actionLicense)

self.menubar.addAction(self.menuFile.menuAction())
self.menubar.addAction(self.menuEdit.menuAction())
self.menubar.addAction(self.menuView.menuAction())
self.menubar.addAction(self.menuSettings.menuAction())
self.menubar.addAction(self.menuWindow.menuAction())
self.menubar.addAction(self.menuHelp.menuAction())

self.data = load()

if not hasattr(self, 'recalculate_average'):
    self.recalculate_average = None

if self.data != {}:
    self.noneLabel.setText('')
else:
    self.tabWidget.hide()

self.avgdata = avgload(self.data, self.recalculate_average)


#self.data = {}
self.tabs = {}
self.tabSliders = {}
self.tabDaySpin = {}
self.tabRepeatSpin = {}
self.tabPlot = {}
self.vals = {}
self.avgvals = {}
self.tabLabel = {}
self.tabButton = {}
self.tabButton2 = {}
self.detailsButton = {}
self.min_repeats = {}
self.max_repeats = {}
self.sensorLabel = {}

for key in self.data:
    self.tabs[key] = QtWidgets.QWidget()
    self.tabs[key].setObjectName(key)
    self.tabs[key].pen = key
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
            self.tabWidget.addTab(self.tabs[key], "")

        self.showdetails = True

        for key, tab in self.tabs.items():

            title = f"{key} sensor data"

            self.tabPlot[key] = MplWidget(tab)
            self.tabPlot[key].canvas.ax.plot([0,1,2,3,4], [10,1,20,3,40])
            self.tabPlot[key].setGeometry(QtCore.QRect(-30, 0, 680, 500))
            self.tabPlot[key].setObjectName(key + " plot")
            self.tabPlot[key].canvas.setTitle(title)

            self.tabSliders[key] = QtWidgets.QSlider(tab)
            self.tabSliders[key].setGeometry(QtCore.QRect(50, 500, 500, 20))
            self.tabSliders[key].setOrientation(QtCore.Qt.Orientation.Horizontal)
            self.tabSliders[key].setObjectName(key + " slider")
            self.tabSliders[key].setStatusTip('Change the test displayed')

            self.vals[key] = []

            max_day = 0
            temp = list(self.data[key].keys())
            for i in range(len(temp)):
                if temp[i] == 'water':
                    temp[i] = -1
            self.avgvals[key] = sorted(temp)

            for day in range(len(self.avgvals[key])):
                if self.avgvals[key][day] == -1:
                    self.avgvals[key][day] = 'water'
                    day = 'water'

            for day in range(len(self.avgvals[key])+1):

                for repeat in range(0, 5):
                    if self.data[key].get(day):
                        if self.data[key][day].get(repeat):
                            self.vals[key].append([day, repeat])


            self.min_repeats[key] = {}
            self.max_repeats[key] = {}
            for day, repeat in self.vals[key]:
                if not self.min_repeats[key].get(day):
                    self.min_repeats[key][day] = repeat
                if not self.max_repeats[key].get(day) or self.max_repeats[key].get(day) < repeat:
                    self.max_repeats[key][day] = repeat

            #for day in self.min_repeats[key].keys():
            #    print(day, self.min_repeats[key][day], self.max_repeats[key][day])



            self.tabSliders[key].setRange(0, (len(self.vals[key])-1))
            self.tabSliders[key].setSingleStep(1)
            self.tabSliders[key].setValue(0)
            self.tabSliders[key].setTickPosition(QtWidgets.QSlider.TickPosition.TicksAbove)
            self.tabSliders[key].setTickInterval(1)
            self.tabSliders[key].valueChanged.connect(self.slider_change)

            self.tabDaySpin[key] = QtWidgets.QSpinBox(tab)
            self.tabDaySpin[key].setGeometry(QtCore.QRect(560, 500, 60, 20))
            self.tabDaySpin[key].setObjectName(key + " daySpin")
            self.tabDaySpin[key].setRange(0, self.avgvals[key][-1:][0])
            self.tabDaySpin[key].setPrefix('Day ')

            temp = self.vals[key][0][0]
            if temp == 'water':
                temp = -1
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
            self.tabDaySpin[key].setValue(temp)
            self.tabDaySpin[key].setSpecialValueText('Water')
            self.tabDaySpin[key].valueChanged.connect(self.day_spin_change)
            self.tabDaySpin[key].setStatusTip('Change the test displayed by day')

            self.tabRepeatSpin[key] = QtWidgets.QSpinBox(tab)
            self.tabRepeatSpin[key].setGeometry(QtCore.QRect(630, 500, 70, 20))
            self.tabRepeatSpin[key].setObjectName(key + " repeatSpin")
            self.tabRepeatSpin[key].setRange(self.min_repeats[key][self.vals[key][0][0]],
self.max_repeats[key][self.vals[key][0][0]])
            self.tabRepeatSpin[key].setPrefix('Repeat ')
            self.tabRepeatSpin[key].setValue(self.vals[key][0][1])
            self.tabRepeatSpin[key].valueChanged.connect(self.repeat_spin_change)
            self.tabRepeatSpin[key].setStatusTip('Change the test displayed by repeat')

            self.tabLabel[key] = QtWidgets.QLabel(tab)
            self.tabLabel[key].setGeometry(QtCore.QRect(585, 50, 200, 450))
            self.tabLabel[key].setObjectName(key + " label")
            self.tabLabel[key].setAlignment(QtCore.Qt.AlignmentFlag.AlignTop |
QtCore.Qt.AlignmentFlag.AlignLeft)

            self.sensorLabel[key] = {}
            self.sensorlabels =
self.data[key][list(self.data[list(self.data)[0]])[0]][list(self.data[key][list(self.data[key])[0
]])[0]].sensorlabels
            y = 15
            n = 0
            self.showsensor = {}
            for sensorlabel in self.sensorlabels:
                self.sensorLabel[key][sensorlabel] = QtWidgets.QCheckBox(self, text=sensorlabel)
                self.showsensor[sensorlabel] = True
                self.sensorLabel[key][sensorlabel].setGeometry(QtCore.QRect(642, 156+(y*n), 100,
y))
                n += 1
                self.sensorLabel[key][sensorlabel].setChecked(True)
                self.sensorLabel[key][sensorlabel].stateChanged.connect(lambda state,
x=sensorlabel: self.show_sensor(state, x))
                self.sensorLabel[key][sensorlabel].setStatusTip('Show/hide ' + sensorlabel)
                self.sensorLabel[key][sensorlabel].setStyleSheet("font-size : 10px")
                #self.showavg = True

            self.detailsButton[key] = QtWidgets.QPushButton("Show details", tab)
            self.detailsButton[key].setGeometry(QtCore.QRect(625, 20, 100, 25))
            self.detailsButton[key].setObjectName(key + " detailsButton")
            self.detailsButton[key].clicked.connect(self.show_details)
            self.detailsButton[key].setStatusTip('Show further test details')
            self.showdetails = False

            self.tabButton[key] = QtWidgets.QPushButton("Show Sensor PCA", tab)
            self.tabButton[key].setGeometry(QtCore.QRect(60, 463, 110, 25))
            self.tabButton[key].setObjectName(key + " pcabutton")
            self.tabButton[key].clicked.connect(self.show_pca)

            self.tabButton2[key] = QtWidgets.QPushButton("Analyse", tab)
            self.tabButton2[key].setGeometry(QtCore.QRect(477, 463, 100, 25))
            self.tabButton2[key].setObjectName(key + " analysebutton")
            self.tabButton2[key].clicked.connect(self.analyse)


            self.update_plot(key, self.vals[key][0][0], self.vals[key][0][1])


        self.retranslateUi()
        self.tabWidget.setCurrentIndex(0)
        self.tabWidget.blockSignals(False)
        QtCore.QMetaObject.connectSlotsByName(self)
        self.show_details()


        return
        ####
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
            try:
                self.Display_PCA()
            except ValueError:
                pass

        def analyse(self):
            self.w = AnalyseWindow()
            self.setDisabled(True)
            self.w.show()
            self.setDisabled(False)



        def Display_PCA(self, options=None):

            type = 'a'

            if options != None:
                if options[0] == 'Amplitude':
                    type = 'a'
                elif options[0] == 'Decay Value':
                    type = 'd'
                elif options[0] == 'Time to return to baseline':
                    type = 'r'
                elif options[0] == 'Maximum gradient':
                    type = 'g'
                elif options[0] == 'Amplitude - baseline':
                    type = 'ab'


            if options[2]: # If exclude first repeat
                pass

            dfs = {}
            sensorlabels = ['Sen-' + str(x) for x in range(1, 25)]
            sensorlabels.append('Humidity')
            vp = {}

            for pen in self.avgdata.keys():

                descriptors = []
                column_names = []
                for day in self.avgdata[pen]:
                    if type == 'g':
                        descriptors.append(self.avgdata[pen][day]['avg'].max_gradient)
                    elif type == 'd':
                        descriptors.append(list(self.avgdata[pen][day]['avg'].decayb.values()))
                    elif type == 'r':

descriptors.append(list(self.avgdata[pen][day]['avg'].bl_return_point.values()))
                    elif type == 'a':
                        descriptors.append(self.avgdata[pen][day]['avg'].amplitude)
                    elif type == 'ab':
                        descriptors.append(self.avgdata[pen][day]['avg'].ampbl)

                    vacprot = self.avgdata[pen][day]['avg'].vacprot

                    column_names.append(str(self.avgdata[pen][day]['avg'].room) + '_' +
str(self.avgdata[pen][day]['avg'].day))
                    if vacprot == 'No treatment':
                        colour = 'gray'
                    elif 'Lawsonia' in vacprot:
                        colour = 'orange'
                    elif 'Circoflex' in vacprot:
                        colour = 'purple'
                    elif 'Saline' in vacprot:
                        colour = 'green'
                    elif vacprot == 'Water Removed':
                        colour = 'blue'
                    else:
```

```
                    colour = 'red'


                vp[(str(self.avgdata[pen][day]['avg'].room) + '_' +
str(self.avgdata[pen][day]['avg'].day))] = colour


            descriptors = list(zip(*descriptors))

            descdf = pd.DataFrame(descriptors, columns=column_names, index=sensorlabels)

            descdf = descdf.replace(0, np.NaN)
            if type == 'g':
                descdf.type = 'Gradient'
            elif type == 'd':
                descdf.type = 'Decay Value'
            elif type == 'r':
                descdf.type = 'Return to Baseline Time'
            elif type == 'a':
                descdf.type = 'Amplitude'
            elif type == 'ab':
                descdf.type = 'Amplitude - baseline'


            dfs[pen] = descdf.astype(float)

        #for pen in self.avgdata.keys():
                #Calculate_PCA_2(dfs[pen])

        result = pd.concat(list(dfs.values()), axis=1)


        #print(result)


        if not options[1]:  #If include only selected sensors
            for k, v in self.showsensor.items():
                #print(k, v)
                if not v:
                    result = result.drop(k)

        #print(result)
        result.type = descdf.type
        result.pen = pen
        del descdf

        #print(result[['2_19 Circoflex+0']].to_string(index=False))
        #print(result[['2_20 Circoflex+1']].to_string(index=False))
        Calculate_PCA_2(result, vp)




    def show_sensor(self, state, sensor):
        if state:
            self.showsensor[sensor] = True
        else:
            self.showsensor[sensor] = False

        self.update_plot()

    def show_details(self):
        if self.showdetails:
            self.showdetails = False
        else:
            self.showdetails = True

        for key, tab in self.tabs.items():
            for sensorlabel in self.sensorlabels:
                if self.showdetails:
                    self.sensorLabel[key][sensorlabel].hide()
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                else:
                    self.sensorLabel[key][sensorlabel].show()

            if self.showdetails:
                self.detailsButton[key].setText("Edit sensors")
                self.detailsButton[key].setStatusTip('Edit visible sensors')
            else:
                self.detailsButton[key].setText("Show details")
                self.detailsButton[key].setStatusTip('Show further test details')
        if self.data != {}:
            self.update_plot()

    def show_pca(self):
        pen = self.tabWidget.currentWidget().pen
        day = self.tabDaySpin[pen].value()
        if day < 1:
            day = 'water'
        repeat = self.tabRepeatSpin[pen].value()

        pca = Calculate_PCA(self.data[pen][day][repeat])


    def slider_change(self):
        val = self.sender().value()
        pen = self.tabWidget.currentWidget().pen
        if self.avg:
            day, repeat = [self.avgvals[pen][val], 'avg']
        else:
            day, repeat = self.vals[pen][val]
            self.tabRepeatSpin[pen].setValue(repeat)
            self.tabRepeatSpin[pen].setRange(self.min_repeats[pen][day],
self.max_repeats[pen][day])

        #print('Slider calculated day:{0}, repeat:{1}'.format(day, repeat))

        if day == 'water':
            temp = -1
        else:
            temp = day
        self.tabDaySpin[pen].setValue(temp)
        self.update_plot(pen, day, repeat)


    def day_spin_change(self):
        day = self.sender().value()
        if day < 1:
            day = 'water'
        #print('Day Spinbox value: ' + str(day))
        pen = self.tabWidget.currentWidget().pen

        if self.avg:
            val = self.avgvals[pen].index(day)
            self.tabSliders[pen].setValue(val)
        else:
            repeat = self.tabRepeatSpin[pen].value()
            if self.min_repeats[pen].get(day) and self.max_repeats[pen].get(day):
                self.tabRepeatSpin[pen].setRange(self.min_repeats[pen][day],
self.max_repeats[pen][day])
            try:
                val = self.vals[pen].index([day, repeat])
                self.tabSliders[pen].setValue(val)
            except ValueError:
                pass

    def repeat_spin_change(self):
        if not self.avg:
            repeat = self.sender().value()
            pen = self.tabWidget.currentWidget().pen
            day = self.tabDaySpin[pen].value()
            try:
                val = self.vals[pen].index([day, repeat])
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                self.tabSliders[pen].setValue(val)
            except ValueError:
                pass


    def update_plot(self, pen=None, day=None, repeat=None):
        if pen == None:
            pen = self.tabWidget.currentWidget().pen
        if day == None:
            day = self.tabDaySpin[pen].value()
            if day < 1:
                day = 'water'
        if repeat == None:
            if self.avg:
                repeat = 'avg'
            else:
                repeat = self.tabRepeatSpin[pen].value()

        if repeat == 'avg':
            avg = True
            test = self.avgdata[pen][day][repeat]
        else:
            avg = False
            test = self.data[pen][day][repeat]


        text = f"Test: {test.name}\n"
        if avg:
            text += f"Date: {test.date}\nRepeat: Average\n\n"
        else:
            text += f"Time & Date: {test.time} {test.date}\n"
            text += f"Repeat: {test.repeats[0]}/{test.repeats[1]}\n\n"

        if self.showdetails:
            #text += "Details:\n"
            text += ''.join(test.details)
            text += f"\nBaseline: {test.baseline}\n"
            text += f"Absorb: {test.absorb}\nPause: {test.pause}\nDesorb: {test.desorb}\n"
            text += f"Flush: {test.flush}\nWait: {test.wait}\nHigh Flow: {test.hflow}\n"
            text += f"Medium Flow: {test.mflow}\nLow Flow: {test.lflow}\n\nProfile Time:
{test.profiletime}\n"
            text += f"Data Rate: {test.datarate}\nData Total: {test.datatotal}\nNumber of
Sensors: {test.sensors}\n"
            if day == 'water':
                age = 28
            else:
                age = test.day + 28

            text += f"Pig's Age: {age} days old\nTreatment/Vaccine: {test.vacprot}"
            #text += f"Max Gradient {test.max_gradient}\n Baseline Return Point:
{test.bl_return_point}"

        self.tabLabel[pen].setText(text)
        self.tabPlot[pen].update_plot(test, self.showsensor,
            self.annotate, self.showwait, self.showdetails, self.shownorm)

        #self.tabPlot[pen].savefigure()

    def retranslateUi(self):
        ''
        _translate = QtCore.QCoreApplication.translate
        self.setWindowTitle(_translate("MainWindow", "VOC Tool"))

        for key, tab in self.tabs.items():
            self.tabWidget.setTabText(self.tabWidget.indexOf(tab), _translate("MainWindow",
(key)))

        self.menuFile.setTitle(_translate("MainWindow", "File"))
        self.menuEdit.setTitle(_translate("MainWindow", "Edit"))
        self.menuSettings.setTitle(_translate("MainWindow", "Settings"))
        self.menuWindow.setTitle(_translate("MainWindow", "Window"))
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
        self.menuHelp.setTitle(_translate("MainWindow", "Help"))
        self.menuView.setTitle(_translate("MainWindow", "View"))

        self.actionOpen.setText(_translate("MainWindow", "Import file(s)..."))
        self.actionOpen.setToolTip(_translate("MainWindow", "Open a .txt or .zip file."))
        self.actionOpen.setShortcut(_translate("MainWindow", "Ctrl+O"))

        self.actionQuit.setText(_translate("MainWindow", "Quit"))
        self.actionQuit.setShortcut(_translate("MainWindow", "Ctrl+Q"))

        self.actionMinimise.setText(_translate("MainWindow", "Minimize"))
        self.actionMinimise.setShortcut(_translate("MainWindow", "Ctrl+M"))

        self.actionAnnotate.setText(_translate("MainWindow", "Annotate Graph"))
        self.actionAnnotate.setShortcut(_translate("MainWindow", "Ctrl+A"))

        self.actionShowwait.setText(_translate("MainWindow", "Show Wait Period"))

        self.actionAverage.setText(_translate("MainWindow", "Average Repeats"))

        self.actionNorm.setText(_translate("MainWindow", "Standardize Data"))

        #self.actionExport_to_excel.setText(_translate("MainWindow", "Export to Excel (WIP)"))
        self.actionDelete_all_imports.setText(_translate("MainWindow", "Delete Imported Data"))

        self.actionPreferences.setText(_translate("MainWindow", "Preferences..."))
        self.actionPreferences.setShortcut(_translate("MainWindow", "Ctrl+P"))

        self.actionOpen_folder.setText(_translate("MainWindow", "Import folder..."))
        self.actionOpen_folder.setShortcut(_translate("MainWindow", "Ctrl+Shift+O"))

        self.actionHelp.setText(_translate("MainWindow", "Help"))
        self.actionHelp.setShortcut(_translate("MainWindow", "Ctrl+?"))

        self.actionLicense.setText(_translate("MainWindow", "About"))

    def openfile(self):
        self.statusbar.showMessage('Importing file(s)...')
        fname = QtWidgets.QFileDialog.getOpenFileNames(
            self,
            'Select file(s)',
            "",
            "Data files (*.txt *.zip)",
        )
        if not fname[0]:
            self.statusbar.showMessage('')
            return

        self.recalculate_average = open_files(fname[0])

        #self.close()
        self.__init__()
        self.show()

    def openfolder(self):
        self.statusbar.showMessage('Importing file(s)...')
        folder = QtWidgets.QFileDialog.getExistingDirectory(self, 'Select folder')
        if not folder:
            self.statusbar.showMessage('')
            return

        folder += '/'
        fnames = os.listdir(folder)
        longfnames = [folder + file for file in fnames]

        self.recalculate_average = open_files(longfnames)

        #self.close()
        self.__init__()
        self.show()
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
    def deleteimports(self):
        text = f"Are you sure you want to continue and remove all imported data?\nThis includes
{len(os.listdir('data'))} data files."
        msg = QtWidgets.QMessageBox(text=text, parent=self)
        msg.setIcon(QtWidgets.QMessageBox.Icon.Warning)
        msg.setStandardButtons(QtWidgets.QMessageBox.StandardButton.Yes |
QtWidgets.QMessageBox.StandardButton.Cancel)
        msg.buttonClicked.connect(self.output)
        ret = msg.exec()

    def output(self, button):
        if button.text() == '&Yes':
            files = os.listdir('data')
            for file in files:
                file = 'data/' + file
                if os.path.isfile(file):
                    os.remove(file)

            msg = QtWidgets.QMessageBox(text="Data files deleted successfully.", parent=self)
            msg.setIcon(QtWidgets.QMessageBox.Icon.Information)
            ret2 = msg.exec()

            self.data = {}
            self.avgdata = {}
            self.recalculate_average = None
            #self.close()
            self.__init__()
            self.show()

    def show_wait(self, action):
        self.showwait = action
        self.update_plot()

    def show_annotate(self, action):
        self.annotate = action
        self.update_plot()

    def show_norm(self, action):
        self.shownorm = action
        self.update_plot()

    def show_avg(self, action):
        if action:
            self.avg = True
        else:
            self.avg = False

        for key, tab in self.tabs.items():
            self.tabRepeatSpin[key].setReadOnly(self.avg)
            self.tabSliders[key].setValue(0)
            if self.avg:
                self.tabRepeatSpin[key].hide()
                self.tabSliders[key].setRange(0, (len(self.avgdata[key])-1))

            else:
                self.tabRepeatSpin[key].show()
                self.tabRepeatSpin[key].setValue(self.min_repeats[key][self.vals[key][0][0]])
                self.tabRepeatSpin[key].setRange(self.min_repeats[key][self.vals[key][0][0]],
self.max_repeats[key][self.vals[key][0][0]])
                self.tabSliders[key].setRange(0, (len(self.vals[key])-1))
        self.update_plot()

class AnalyseWindow(QtWidgets.QWidget):
    """

    Window for confirming analyse and allow configuration of settings before analysing.

    """

    def __init__(self):
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
        super().__init__()

        win.analyseConfig = None


        self.setWindowFlags(QtCore.Qt.WindowType.WindowStaysOnTopHint)

        self.firstRepeatCheckbox = QtWidgets.QCheckBox("  Do not include first\n  repeat in
test", self)
        self.firstRepeatCheckbox.setGeometry(QtCore.QRect(20, 110, 180, 50))
        self.firstRepeatCheckbox.setChecked(True)

        self.setWindowTitle("VOC Tool: Anaylse data")
        size = [200, 190]
        self.resize(size[0], size[1])
        self.setFixedSize(size[0], size[1])
        self.setWindowIcon(QtGui.QIcon('logo.png'))

        self.descriptorLabel = QtWidgets.QLabel("Descriptor Type:", self)
        self.descriptorLabel.setGeometry(QtCore.QRect(20, 10, 180, 25))

        self.descriptorCombobox = QtWidgets.QComboBox(self)
        self.descriptorCombobox.setGeometry(QtCore.QRect(20, 32, 160, 25))
        self.descriptorCombobox.addItems(['Amplitude', 'Decay Value', 'Time to return to
baseline', 'Maximum gradient', 'Amplitude - baseline'])

        self.confirmButton = QtWidgets.QPushButton("Analyse Data", self)
        self.confirmButton.setGeometry(QtCore.QRect(20, 160, 160, 25))
        self.confirmButton.clicked.connect(self.confirm)

        self.sensorLabel = QtWidgets.QLabel("Sensors used:", self)
        self.sensorLabel.setGeometry(QtCore.QRect(20, 58, 180, 25))

        self.sensorCombobox = QtWidgets.QComboBox(self)
        self.sensorCombobox.setGeometry(QtCore.QRect(20, 80, 160, 25))
        self.sensorCombobox.addItems(['Visible sensors', 'All sensors'])

    def confirm(self):
        options = []

        options.append(str(self.descriptorCombobox.currentText()))

        if str(self.sensorCombobox.currentText()) == 'All sensors':
            options.append(True)
        else:
            options.append(False)

        options.append(self.firstRepeatCheckbox.isChecked())

        #print(options)

        self.close()
        win.Display_PCA(options)









if __name__ == "__main__":
    main()
```

8.3 Appendix C | Python file, datahandler.py

This python file is responsible for most of the back end such as importing the raw text files received from the VOC analysers. It converts the text files into data files which are loaded into the software on start-up to avoid the requirement to import the raw text files every time the software is executed. It performs validation on the data, removing corrupt data points and abnormal data spikes. It also performs processing including standardizing and normalising the data, calculating descriptor values, and partial pre-processing for the PCA module. Most of this is done when importing the raw text files to greatly improve efficiency when the software is run multiple times.

```
import pickle
import os
import re
from zipfile import ZipFile
import pandas as pd
import numpy as np


class Open:
    def __init__(self, file=None, data=None):
        self.error = False

        if file == None and data != None:

            data = list(data)
            filename = data[0][1].filename
            filename1 = re.split("day", filename, flags=re.IGNORECASE, maxsplit=1)
            if len(filename1) == 1:
                filename2 = filename1[0].split("_", 1)
                self.filename = filename2[0] + '_average'
            else:
                filename2 = filename1[1].split("_", 1)
                if len(filename2) != 1:
                    self.filename = filename1[0] + 'Day' + filename2[0] + '_average'
                else:
                    self.filename = filename + '_average'

            self.details = data[0][1].details
            self.date = data[0][1].date
            self.time = None
            self.name = data[0][1].name
            self.smartname = data[0][1].smartname
            self.day = data[0][1].day
            self.room = data[0][1].room
            self.repeats = ['avg', data[0][1].repeats[1]]
            self.baseline = data[0][1].baseline
            self.absorb = data[0][1].absorb
            self.pause = data[0][1].pause
            self.desorb = data[0][1].desorb
            self.flush = data[0][1].flush
            self.wait = data[0][1].wait
            self.hflow = data[0][1].hflow
            self.mflow = data[0][1].mflow
            self.lflow = data[0][1].lflow
            self.profiletime = data[0][1].profiletime
            self.datarate = data[0][1].datarate
            self.datatotal = data[0][1].datatotal
            self.sensors = data[0][1].sensors
            self.triggers = data[0][1].triggers
            self.vacprot = data[0][1].vacprot
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
            self.data = [[0.0 for x in range(len(data[0][1].data[0]))] for y in
range(len(data[0][1].data))]
            for x in range(len(self.data)):
                for y in range(len(self.data[0])):
                    sum_ = 0.0
                    for i in range(len(data)):
                        sum_ += data[i][1].data[x][y]
                    self.data[x][y] = sum_ / (i+1)

            self.create_data_frame()


        else:
            try:
                f = open(file, 'r')
                lines = f.readlines()
                f.close()
                file = file.split('.')
                file.pop()
                file = '.'.join(file)

                file = file.split('/')
                self.filename = file.pop()


                self.read(lines)
            except FileNotFoundError as e:
                self.error = True

    def read(self, lines):

        if re.search('Test', self.filename):
                print('Sampler test')
                self.sampler = True

        line = 1
        self.details = []
        while not lines[line].startswith('---'):
            self.details.append(lines[line])
            line += 1

        self.date = lines[line+1][7:17]
        self.time = lines[line+2][7:15]

        self.name = lines[line+5][25:].replace('"', '').replace('\n', '')

        temp = re.split("day", self.name, flags=re.IGNORECASE)



        self.smartname = temp[0].replace('_', ' ').strip()
        if len(temp) > 1:
            self.day = int(temp[1])
        elif self.smartname == 'Water end':
            self.day = 'water'
            self.smartname = 'Pen3A 60'

        elif self.smartname == 'water f':
            self.day = 'water'
            self.smartname = 'Pen5a60'
        elif self.smartname == 'water d':
            self.day = 'water'
            self.smartname = 'Pen 6A'
        elif 'water' in temp[0].casefold():
            temp2 = re.split("water", temp[0], flags=re.IGNORECASE)
            if len(temp2) > 1:
                if int(temp2[1]) == 1:
                    self.day = -2
                if int(temp2[1]) == 2:
                    self.day = -1
                if int(temp2[1]) == 3:
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                self.day = 0
            else:
                self.day = -999
        else:
            self.day = -999

        if 'pen' in self.smartname.casefold():
            temp = re.split("pen", self.smartname, flags=re.IGNORECASE)
        else:
            temp = re.split("room", self.smartname, flags=re.IGNORECASE)

        self.room = int(temp[1].strip()[0])

        self.repeats = [int(lines[line+7][13]), int(lines[line+7][15])]
        self.baseline = float(lines[line+11][11:])
        self.absorb = float(lines[line+12][9:])
        self.pause = float(lines[line+13][8:])
        self.desorb = float(lines[line+14][9:])
        self.flush = float(lines[line+15][8:])
        self.wait = float(lines[line+16][7:])
        self.hflow = float(lines[line+17][18:])
        self.mflow = float(lines[line+18][18:])
        self.lflow = float(lines[line+19][15:])

        y = 0
        try:
            self.profiletime = int(lines[line+21][21:23])
        except ValueError:
            y = 1
            line += 11
            self.profiletime = int(lines[line+21][21:23])

        self.datarate = int(lines[line+23][33:])
        self.datatotal = int(lines[line+25][(27-y):])
        self.sensors = int(lines[line+27][31:33])

        self.triggers = [0.0, self.baseline, self.absorb, self.pause, self.desorb, self.flush,
self.wait]
        time = 0
        for i in range(len(self.triggers)):
            time += (self.triggers[i]) * self.datarate
            self.triggers[i] = time
        line += 31
        self.data = []
        while not line == len(lines):
            strs = lines[line].split()
            #floats = [float(val) for val in strs]
            floats = []
            for val in strs:
                temp = float(val)
                if temp > 9999 or temp < -9999:
                    temp = 0.0
                floats.append(temp)
            self.data.append(floats[1:9999])

            line += 1

        if self.smartname == 'Pen2A 60' or 'Room 2':
            order = ['Saline', 'Lawsonia', 'Circoflex']

        elif self.smartname == 'Pen3A 60' or 'Room 3':
            ''

        elif self.smartname == 'Pen5a60' or 'Room 5':
            order = ['Lawsonia', 'Circoflex', 'Saline']

        elif self.smartname == 'Pen 6A' or 'Room 6':
            order = ['Circoflex', 'Saline', 'Lawsonia']

        else:
            print('Warning: Failed to allocating vaccine protocol (1).')
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
            self.vacprot = 'N/A'
            self.create_data_frame()
            return

        if self.day == 'water':
            self.vacprot = 'Water Removed'

        elif (self.day <= 4) | (self.smartname == 'Pen3A 60' or self.smartname == 'Room 3'):
            self.vacprot = 'No treatment'

        elif self.day <= 11:
            self.vacprot = order[0] + '+' + str(self.day-5)

        elif self.day <= 18:
            self.vacprot = order[1] + '+' + str(self.day-12)

        elif self.day <= 24:
            self.vacprot = order[2]  + '+' + str(self.day-19)
        else:
            print('Warning: Failed to allocating vaccine protocol (2).')
            self.vacprot = 'N/A'


        self.create_data_frame()



        #import matplotlib.pyplot as plt
        #Fs = 250
        #tstep = 1 / Fs

        #humidity = self.dataframe['Humidity']

        #print(humidity)
        #n = len(humidity)



        #t = np.linspace(0, (n-1)*tstep, n)
        #fstep = Fs / n
        #f = np.linspace(0, (n-1)*fstep, n)

        #humidity = 1 * np.sin(2* np.pi * f0 * t)

        #x = np.fft.fft(humidity)
        #xmag = np.abs(x) / n

        #f_plot = f[0:int(n/2+1)]
        #xmagplot = 2 * xmag[0:int(n/2+1)]
        #xmagplot[0] = 0#xmagplot[0] /2

        #fig, [ax1, ax2] = plt.subplots(nrows=2, ncols=1)
        #ax1.plot(t, humidity, '.-')
        #ax2.plot(f_plot, xmagplot, '.-')
        #plt.show()


    def create_data_frame(self):

        self.timelabels = [x for x in range(0, len(self.data))]
        self.sensorlabels = ['Sen-' + str(x) for x in range(1, len(self.data[0]))]
        self.sensorlabels.append('Humidity')

        self.dataframe = pd.DataFrame(self.data, columns=self.sensorlabels,
index=self.timelabels)

        for i in self.dataframe.index:
            self.dataframe.loc[i] = self.data[i]

        self.dataframe = self.dataframe.replace(0, np.NaN)
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
        baseline = range(round(self.triggers[0]), round(self.triggers[1]))
        baseline_values = []
        for index in baseline:
            baseline_values.append(list(self.dataframe.iloc[index]))

        from scipy.optimize import curve_fit
        self.decaya = {}
        self.decayb = {}
        for label in self.sensorlabels:
            df = self.dataframe[label][round(self.triggers[4]):round(self.triggers[5])]
            df = df.dropna()
            try:
                (a, b), *_ = curve_fit(self.model, range(0, len(df)), df.values /
df.values.max())

                a *= df.values.max()

                self.decaya[label] = a
                self.decayb[label] = b
            except ValueError:
                pass


        #print(self.decay)

        self.max_gradient = [(self.dataframe[l] / self.dataframe[l].shift(-
1)).sort_values(ascending = False).iloc[0] for l in self.sensorlabels]
        #print(self.max_gradient)

        #print(baseline_values)
        average = []
        for i in zip(*baseline_values):
            sum_ = 0
            count = 0
            for v in i:
                if not np.isnan(v):
                    sum_ += v
                    count += 1
            average.append(sum_ / count)
        self.normdf = self.dataframe.sub(average, axis='columns')
        self.bl_average = dict(zip(self.sensorlabels, average))
        self.bl_return_point = {}
        self.ampbl = []
        k = 2
        for label in self.sensorlabels:
            df = self.dataframe[label].between(self.bl_average[label]-k,
self.bl_average[label]+k)

            for i in range(round(self.triggers[3]), round(self.triggers[5])-2):
                self.bl_return_point[label] = round(self.triggers[5])
                try:
                    if df[i] and df[i+1] and df[i+2] and df[i+3] and df[i+4]:
                        self.bl_return_point[label] = self.dataframe[label].index[i]
                        break
                except KeyError:
                    break
        self.amplitude = []
        for label in self.sensorlabels:
            df = self.dataframe[label][round(self.triggers[2])+1:round(self.triggers[3])]
            self.amplitude.append(df.mean(axis=0))

        for i in range(len(self.amplitude)):
            self.ampbl.append(self.amplitude[i] - list(self.bl_average.values())[i])



    def model(self, t, a, b):
        return a * np.exp(-b * t)

    def dump(self):
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
        if not os.path.exists('data'):
            os.mkdir('data')


        file = open('data/' + self.filename + '.data', 'wb')
        pickle.dump(self, file)
        file.close()




def open_files(files):
    if files == []:
        return


    for i in range(len(files)):

        if files[i].endswith('.zip'):
            with ZipFile(files[i], 'r') as zip:

                temp = zip.namelist()

                files += ['temp/' + x for x in temp]

                #print('Extracting all the files now from ' + files[0])

                zip.extractall(path ='temp/')

                #zip.printdir()

                #print('Done!')

    #print(files)

    recalculate_average = []


    for i in range(len(files)):

        #if re.search('Sampler|Test', files[i]):
        #    print('Sampler test')

        if files[i].endswith('.txt'):
            print("Opening " + files[i])
            f = Open(files[i])
            if not recalculate_average.count([f.smartname, f.day]):
                recalculate_average.append([f.smartname, f.day])
            f.dump()




    for file in files:
        if file.startswith('temp/') and os.path.isfile(file):
            os.remove(file)

    if os.path.isdir('temp') and not os.listdir('temp'):
        os.rmdir('temp')

    return recalculate_average

def load(avg=False):

    if not os.path.exists('data'):
        return {}

    files = os.listdir('data')
```

```
    data = {}

    for i in range(len(files)):
        if 'Test' in files[i]:
            break
        if ((not avg and files[i].endswith('.data') and not files[i].endswith('average.data')) or
(avg and files[i].endswith('average.data'))):
            temp = open(('data/' + files[i]), 'rb')
            file = pickle.load(temp)

            new_room = True
            keys = list(data.keys())
            for j in range(len(data)):
                if keys[j] == file.smartname:
                    new_room = False
                    break

            if new_room:
                data[file.smartname] = {}
                data[file.smartname][file.day] = {}

            else:
                new_day = True
                keys = list(data[file.smartname].keys())
                for j in range(len(data[file.smartname])):
                    if keys[j] == file.day:
                        new_day = False
                        break

                if new_day:
                    data[file.smartname][file.day] = {}

            data[file.smartname][file.day][file.repeats[0]] = file
            #print(data[file.smartname][file.day][file.repeats[0]].smartname + ', ' +
str(data[file.smartname][file.day][file.repeats[0]].day) + ', ' +
str(data[file.smartname][file.day][file.repeats[0]].repeats[0]))

    return data

def avgload(data, recalculate_average):
    if recalculate_average != None:

        for keys in recalculate_average:
            pen = keys[0]
            day = keys[1]
            f = Open(data=data[pen][day].items())
            f.dump()

    avgdata = load(avg=True)
    return avgdata



def main():

    test_filename = 'Room 2 Day 1 31-10-2022 09.43.17 Repeat 1 Test'
    #test_filename = 'Room 2 Water 1 31-10-2022 09.14.59 Repeat 1 Test'

    test = Open(test_filename + '.txt')
    if test.error == True:
        print("FileNotFoundError")
    test.dump()

    file = open('data/' + test_filename + '.data', 'rb')
    data = pickle.load(file)
    file.close()
    #print(vars(data))
    #print(data.triggers)

    file = open('data/' + 'Pen2A 60 Day 1_15_05_46.44.data', 'rb')
    data = pickle.load(file)
```

```
    file.close()
    #print(vars(data))
    #print(data.triggers)


    #print('\n\n------------AVERAGE--------------\n\n')
    #file = open('data/Room 2 Day 1 31-10-2022 09.43.17 Repeat 1.data', 'rb')
    #data = pickle.load(file)
    #file.close()
    #print(vars(data))

    #print(data.data)


if __name__ == '__main__':
    main()
```

8.4 Appendix D | Python file, plot.py

The python file is an API (application programming interface) between the data handler and the GUI. It creates live plots of the sensor data as a PyPlot element and adds backwards support for PyQt6 to display it as a GUI element.

```
# Imports
from PyQt6 import QtCore, QtWidgets
from matplotlib.figure import Figure
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as Canvas
import matplotlib.pyplot as plt
import matplotlib

matplotlib.use('Qt5Agg')


class MplCanvas(Canvas):
    def __init__(self):
        self.fig = Figure(figsize=(6,4), dpi=100)
        self.ax = self.fig.add_subplot(111)


        Canvas.__init__(self, self.fig)
        #Canvas.setSizePolicy(self, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Expanding)
        Canvas.updateGeometry(self)

    def setTitle(self, title):
        self.fig.suptitle(f"{title}\n", fontweight ="bold")

# Matplotlib widget
class MplWidget(QtWidgets.QWidget):
    def __init__(self, parent=None):
        QtWidgets.QWidget.__init__(self, parent)    # Inherit from QWidget
        self.canvas = MplCanvas()                   # Create canvas object
        self.vbl = QtWidgets.QVBoxLayout()          # Set box for plotting
        self.vbl.addWidget(self.canvas)
        self.setLayout(self.vbl)

        self.smartname = None
        self.day = None
        self.repeats = None
        self.showsensor = {}
        self.annotate = None
        self.showwait = None
        self.showdetails = None
        self.shownorm = None
```

```
    def update_plot(self, data, showsensor, annotate=True, showwait=True, showdetails=False,
shownorm=False):
        change = False
        if self.smartname != data.smartname:
            self.smartname = data.smartname
            change = True

        if self.day != data.day:
            self.day = data.day
            change = True

        if self.repeats != data.repeats:
            self.repeats = data.repeats
            change = True

        temp = list(showsensor.values())
        if self.showsensor != temp:
            self.showsensor = temp
            change = True

        if self.annotate != annotate:
            self.annotate = annotate
            change = True

        if self.showwait != showwait:
            self.showwait = showwait
            change = True

        if self.shownorm != shownorm:
            self.shownorm = shownorm
            change = True

        if self.showdetails != showdetails:
            self.showdetails = showdetails
            change = 2

        if not change:
            return

        self.canvas.ax.cla()
        #self.canvas.ax.plot((range(len(data.data))), data.data, label=data.sensorlabels)

        if annotate:
            if showwait:
                r = 6
                triggers=data.triggers
            else:
                r = 5
                triggers=data.triggers[:-1]

            trigger_names = ['Baseline', 'Absorb', 'Pause', 'Desorb', 'Flush', 'Wait']
            trigger_length = [len(n) * 1.1 for n in trigger_names]

            if shownorm:
                y = -107.5
            else:
                y = +14

            if not 'pen' in data.smartname.casefold():
                y += 5

            for i in range(r):
                self.canvas.ax.text(
                    ((data.triggers[i]+data.triggers[i+1]-trigger_length[i])/2),
                    y, trigger_names[i], fontsize = 8)

        if shownorm:
            self.canvas.ax.plot(data.normdf, label=data.sensorlabels)
            if annotate:
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                self.canvas.ax.vlines(x=triggers, ymin=-100, ymax=100, colors='black', ls='--',
lw=1)
        else:
            self.canvas.ax.plot(data.dataframe, label=data.sensorlabels)
            if annotate:
                if 'pen' in data.smartname.casefold():
                    self.canvas.ax.vlines(x=triggers, ymin=25, ymax=320, colors='black', ls='--',
lw=1)
                else:
                    self.canvas.ax.vlines(x=triggers, ymin=35, ymax=500, colors='black', ls='--',
lw=1)



        #humidity = [x[-1:] for x in data.data]
        #self.canvas.ax.plot(humidity, label="Humidity (%r.h.)")

        if change == 2:
            if showdetails:
                self.canvas.ax.leg.remove()
            else:
                self.canvas.ax.leg = self.canvas.fig.legend(loc='right',
                bbox_to_anchor=(1.08, 0.40),
                frameon=False,
                prop={'size': 7}
                )

        n = 0
        for sensor in showsensor.items():
            if not sensor[1]:
                self.canvas.fig.gca().lines[n].set_alpha(0)
            n += 1

        self.canvas.ax.set_xlabel('Datapoint /{0}ms'.format(int(round(1000/data.datarate))),
               fontweight ='bold')
        self.canvas.ax.set_ylabel('Response /V',
               fontweight ='bold')
        self.canvas.draw()

    #def savefigure(self):
    #    plt.savefig('file.jpeg', edgecolor='black', dpi=400, facecolor='black',
transparent=True)

if __name__ == "__main__":
    ''#import sys

    #app = QtWidgets.QApplication(sys.argv)
    #win = MplCanvas()
    #win.show()

    #sys.exit(app.exec())
```

8.5 Appendix E | Python file, PCA.py

The python file performs PCA (principal component analysis) and scree plot calculations for each sensor to visualize sensor weight and for each test to visualize trends in sensor data.

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

```
#https://www.youtube.com/watch?v=Lsue2gEM9D0

class Calculate_PCA:
	def __init__(self, rawdata):

		data = rawdata.dataframe

		for i in data.index:
			data.loc[i] = rawdata.data[i]


		scaled_data = preprocessing.scale(data.T)
		#StandardScaler().fit_transform(data.T)

		pca = PCA()
		pca.fit(scaled_data)
		pca_data = pca.transform(scaled_data)

		per_var = np.round(pca.explained_variance_ratio_* 100, decimals=1)
		labels = ['PC' + str(x) for x in range(1, len(per_var)+1)]

		#plt.bar(x=range(1,len(per_var)+1), height=per_var, tick_label=labels)
		#plt.plot()
		#plt.ylabel('Percentage of Explained Variance')
		#plt.xlabel('Principal Component')
		#plt.title(f'Scree Plot - {rawdata.smartname} Day {rawdata.day} Repeat
{rawdata.repeats[0]}')
		#plt.xticks(rotation=45)
		#plt.show()

		pca_df = pd.DataFrame(pca_data, index=rawdata.sensorlabels, columns=labels)

		plt.scatter(pca_df.PC1, pca_df.PC2)
		plt.title(f'PCA Graph - {rawdata.smartname} Day {rawdata.day} Repeat
{rawdata.repeats[0]}')
		plt.xlabel('PC1 - {0}%'.format(per_var[0]))
		plt.ylabel('PC2 - {0}%'.format(per_var[1]))

		for sample in pca_df.index:
			plt.annotate(sample, (pca_df.PC1.loc[sample], pca_df.PC2.loc[sample]))

		plt.show()

		loading_scores = pd.Series(pca.components_[0], index=rawdata.timelabels)
		sorted_loading_scores = loading_scores.abs().sort_values(ascending=False)

		top_10_sensors = sorted_loading_scores.iloc[0:10].index.values

		print("Top 10 sensors:")
		print(loading_scores[top_10_sensors])


class Calculate_PCA_2:
	def __init__(self, data, vp):

		scaled_data = preprocessing.scale(data.T)
		#StandardScaler().fit_transform(data.T)
		pca = PCA()
		pca.fit(scaled_data)
		pca_data = pca.transform(scaled_data)

		per_var = np.round(pca.explained_variance_ratio_* 100, decimals=1)
		labels = ['PC' + str(x) for x in range(1, len(per_var)+1)]

		if False:
			plt.bar(x=range(1,len(per_var)+1), height=per_var, tick_label=labels)
			plt.plot()
			plt.ylabel('Percentage of Explained Variance')
			plt.xlabel('Principal Component')
			plt.title(f'Scree Plot - '+ str(data.pen))
			plt.xticks(rotation=45)
```


Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY

```
                        plt.show()

                pca_df = pd.DataFrame(pca_data, index=list(data.columns.values), columns=labels)

                for i,j in enumerate(pca_df.PC1):
                        plt.scatter(pca_df.PC1.values[i], pca_df.PC2.values[i], color=
vp.get(pca_df.index[i]))

                foo = data.pen
                plt.title(f'PCA Graph (Descriptor = {data.type})')

                #plt.title(f'PCA Graph - {data.pen} (Descriptor = {data.type})')
                plt.xlabel('PC1 - {0}%'.format(per_var[0]))
                plt.ylabel('PC2 - {0}%'.format(per_var[1]))

                for sample in pca_df.index:
                        plt.annotate(sample, (pca_df.PC1.loc[sample], pca_df.PC2.loc[sample]))

                plt.show()

                loading_scores = pd.Series(pca.components_[0], index=list(data.index.values))
                sorted_loading_scores = loading_scores.abs().sort_values(ascending=False)

                top_10_sensors = sorted_loading_scores.iloc[0:10].index.values


if __name__ == '__main__':
        import pickle
        file = open('data/Pen 6A_Day 18_15_15_25.46.data', 'rb')
        data = pickle.load(file)
        file.close()

        Calculate_PCA(data)
```

8.6 Appendix F | Progress Report Presentation

This presentation was performed for Kevin Wells and Tim Gibson in March to discuss current and future development. It can be downloaded if this report is in DOCX format and not PDF.

UNIVERSITY OF SURREY
UNDERGRADUATE PROJECT

Progress Report

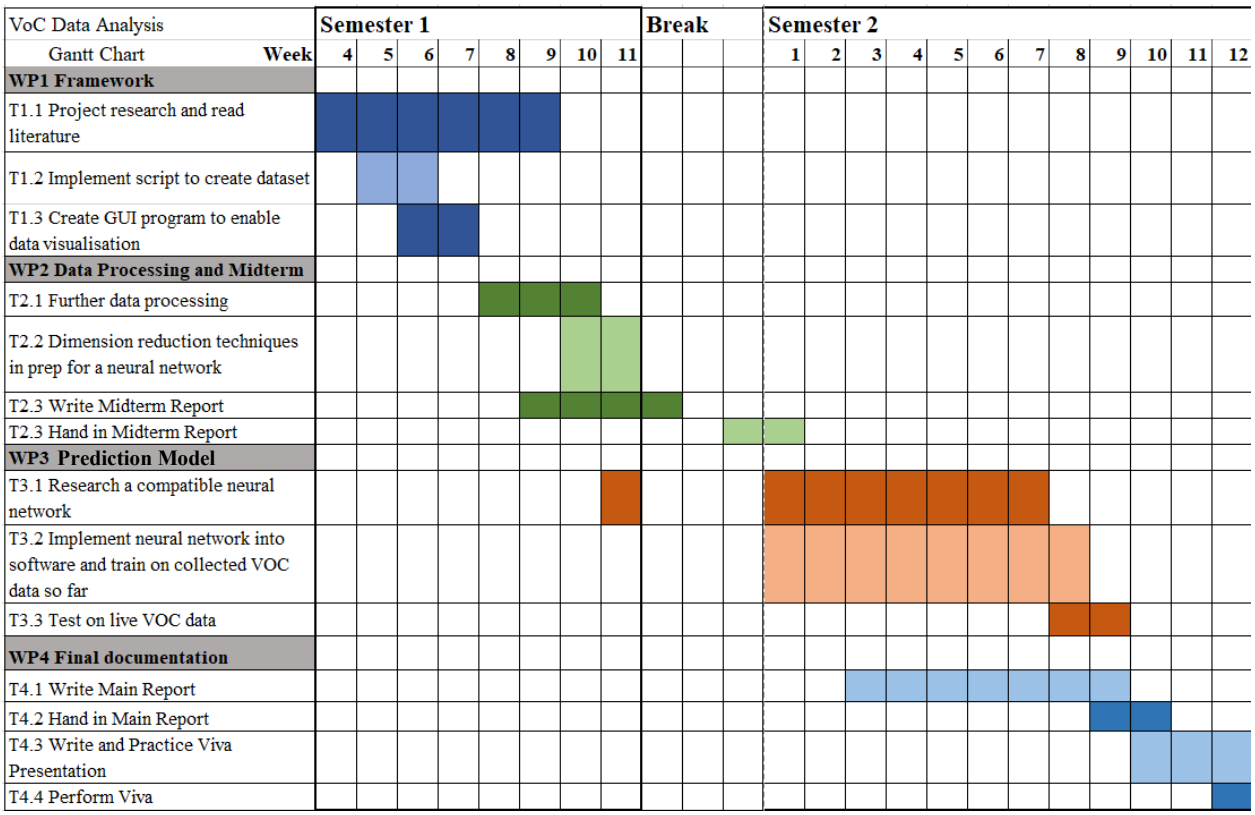
Predicting Pigs Behavioural Changes in the Pork Industry using Violate Organic Compounds with Machine Learning

By Alex Dowsett

Progress Report.pptx

8.7 Appendix G | Gantt Chart

Figure 3: Gantt Chart detailing project plan by each week



Alex Dowsett (UG - Comp Sci & Elec Eng)
UNIVERSITY OF SURREY