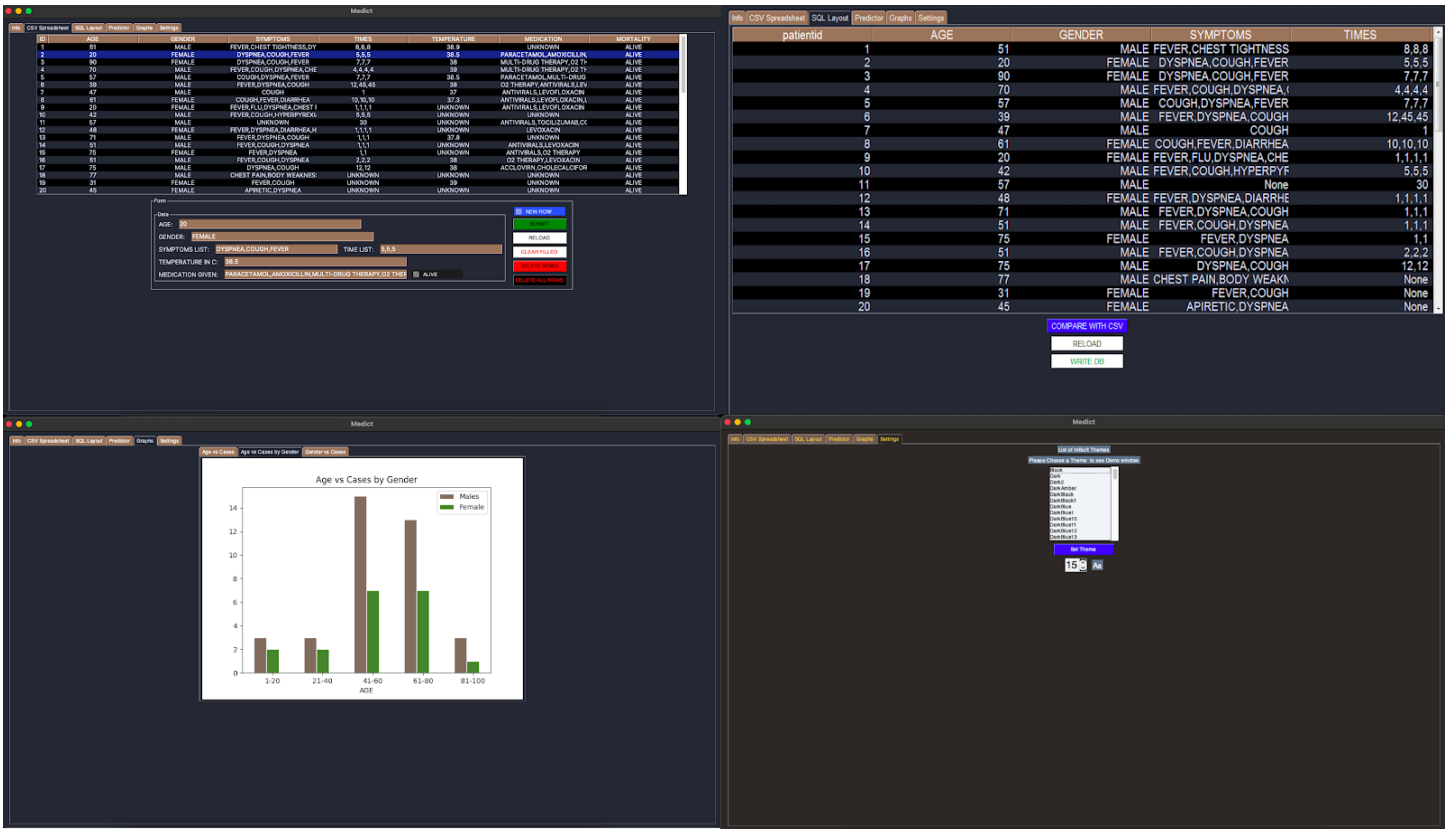
Computer Science Project
Medical Data Utilities

10 December 2020



Name :

Roll Number :

BONAFIDE CERTIFICATE

Register No. Internal Assessment

Certified to be the Bonafide Investigatory Project in Computer Science, done by _____ of Class XII, Section D of D.A.V Boy's Senior Secondary School, Gopalapuram during the year 2020 to 2021.

_____ _____

Signature of Principal
School Seal

Signature of Subject Teacher
Designation:

Submitted for the Practical Examination held on _____
at _____.

Internal Examiner	External Examiner

Chief Superintendent

Date :

Acknowledgement

Apart from the contribution and efforts of the team members, we would like to express our sincere gratitude to our teachers who have been instrumental in the approval of this project for their guidance and encouragement

We take this opportunity to thank our teacher Dr. H.Vidhya and other staff members. We cannot thank them enough for their tremendous support and help. Without their motivation and advice, we would not have been able to finish this project.

The contribution and support received from all team members including Aathish.S, Naveen.M.K, Rajesh Kumar.S and Naresh.S was vital. The team spirit shown by each team member has made this project successful.

We would like to express our gratitude to the Central Board of Secondary Education, as well as our School Management and Principal for giving us this valuable opportunity.

Table Of Contents

A Brief Overview:

The main objective of this project is to provide an easy to use, feature-rich Patient Data Management System for hospital and clinic use in the form of a compact, simple and user-friendly software.

This software application maintains the data of the patients such as their age, symptoms, gender, temperature, medications given and mortality.

The application also stores all these data for future needs (such as predicting the patient's life expectancy at the current level of care, or visualising the data graphically, both of which are also features of the application) and also lets the user enter new data while allowing them to tune the theme and font size of the software's layout as they wish.

The product is intended for hospital or clinic administrators. Administrators, and those authorised by administrators are guaranteed access.

Need For Computerisation:

The basic functioning of the majority of the hospitals in India is still manual. Not only is manual data management prone to error, it is also unnecessarily tedious. Software is preferred over manual management as it vastly simplifies the workflow of an administrator.

Data visualization is also much easier to accomplish with computers than by hand. At a glance and the press of a button, an administrator can easily have an intuitive view of every patient's data.

Tools Used:

1. Python
    a. PySimpleGUI
    b. Matplotlib
    c. Mysql.connector
    d. Numpy
2. SQL

Python

Python is an interpreted, object oriented and high level programming language, with integrated dynamic semantics primarily for scripting and app development. It was developed by Guido Van Rossum in the early 1990s. Python is meant to be an easily readable language with simple and unique syntax. It supports the use of modules and packages and provides a standard library.

Libraries Used:

1. PySimpleGUI
2. Matplotlib
3. Mysql.connector
4. Numpy

PySimpleGUI

PySimpleGUI is a Python package that enables Python programmers of all levels to create Graphical User Interfaces. PySimpleGUI code is simpler and shorter than writing directly using the underlying framework. Additionally, interfaces are simplified to require as little code as possible to get the desired result. It makes the user's interface look attractive. An alternative

for PySimpleGUI is PyQt. Due to the complexity of PyQt, which requires the installation of external Graphics Libraries that must be installed in addition to Python, PySimpleGUI has been used instead.

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It has a module named pyplot which makes things easy for plotting by providing features to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots which include histograms, bar charts, power spectra, error charts and more. It is used in this project for the graphical comparisons of various parameters. Python(x,y) is an alternative for Matplotlib library which Python offers, and Matplotlib has been used as Python(x,y) is difficult to embed in User Interfaces unlike Matplotlib.

MySQL Connector/Python

MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API Specification. It is written in pure Python and does not have any dependencies except for the Python Standard Library. Another library which could have been used to accomplish

this is “MySQL-python”. MySQL Connector/Python has been used instead of “MySQL-python” as the former is officially endorsed and supported by Oracle, the maintainers of MySQL.

Numpy

NumPy is a Python library used for working with arrays that can be used to perform a number of mathematical operations such as trigonometric, statistical, and algebraic routines on those arrays or some given data. An alternative to Numpy is TensorFlow. TensorFlow has not been used for this project as it is used mainly for building Neural Networks and is very Processor and Memory Intensive, while Numpy maintains a small memory footprint.

SQL

SQL stands for Structured Query Language and lets you manipulate and access data from relational databases like MySQL, Oracle, SQL Server, PostGre, etc. The recent ISO standard version of SQL is SQL:2019. The uses of SQL include modifying database table and index structures; adding, updating and deleting rows of data; and retrieving subsets of information from within a database for transaction processing and analytics applications. Queries and other SQL operations take the form of commands written as

statements -- commonly used SQL statements include `SELECT`, `ADD`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `ALTER` and `TRUNCATE`.

Application Benefits:

1. Data Entry is made easier.
2. Data modification is made easier.
3. Transfer of Data from file to Database is simpler.
4. Intuitive Graphical User Interface (GUI)
5. Data Visualization tools (Graphs)
6. GUI Modification is possible and easy with settings.
7. GUI Theme and font size can be easily modified to suit the user.
8. Persistent storage of user settings.

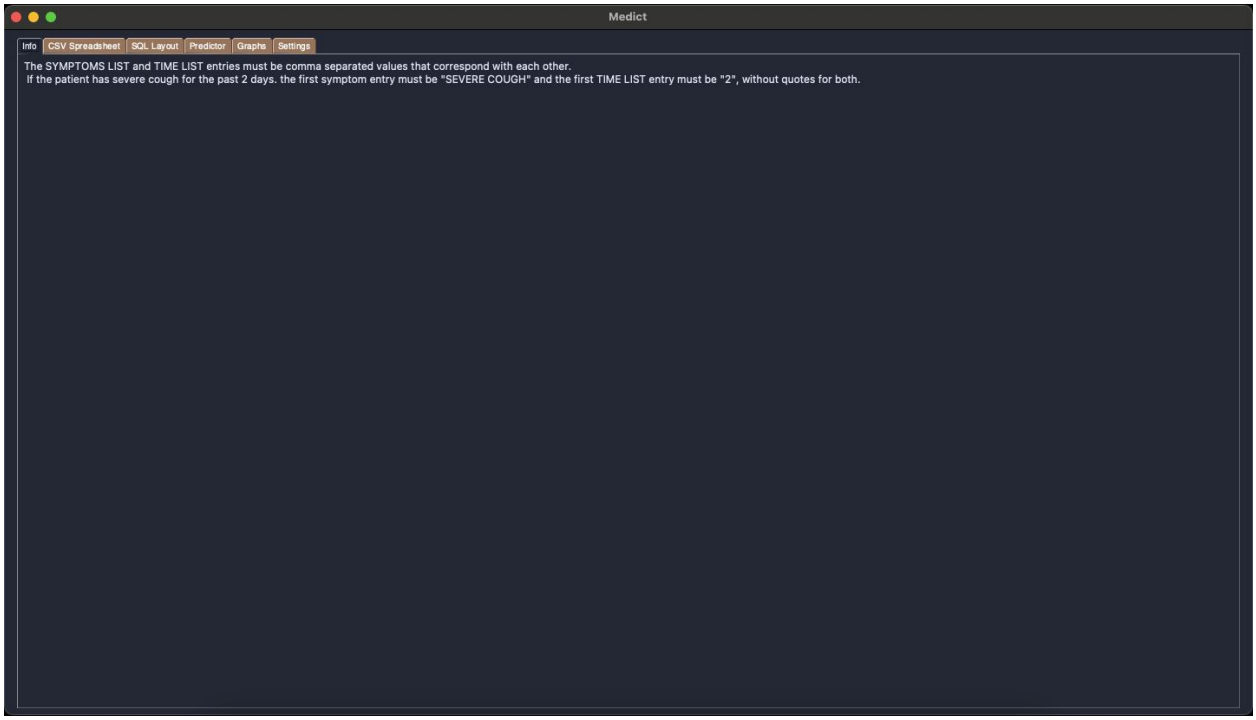
Application Limits:

1. Requires an already existing SQL Database.
2. Only the pre-existing packaged CSV file can be used.
3. Backup and restore feature has not been implemented.
4. Online Storage of Data has not been implemented.
5. Keyboard shortcuts have not been implemented.

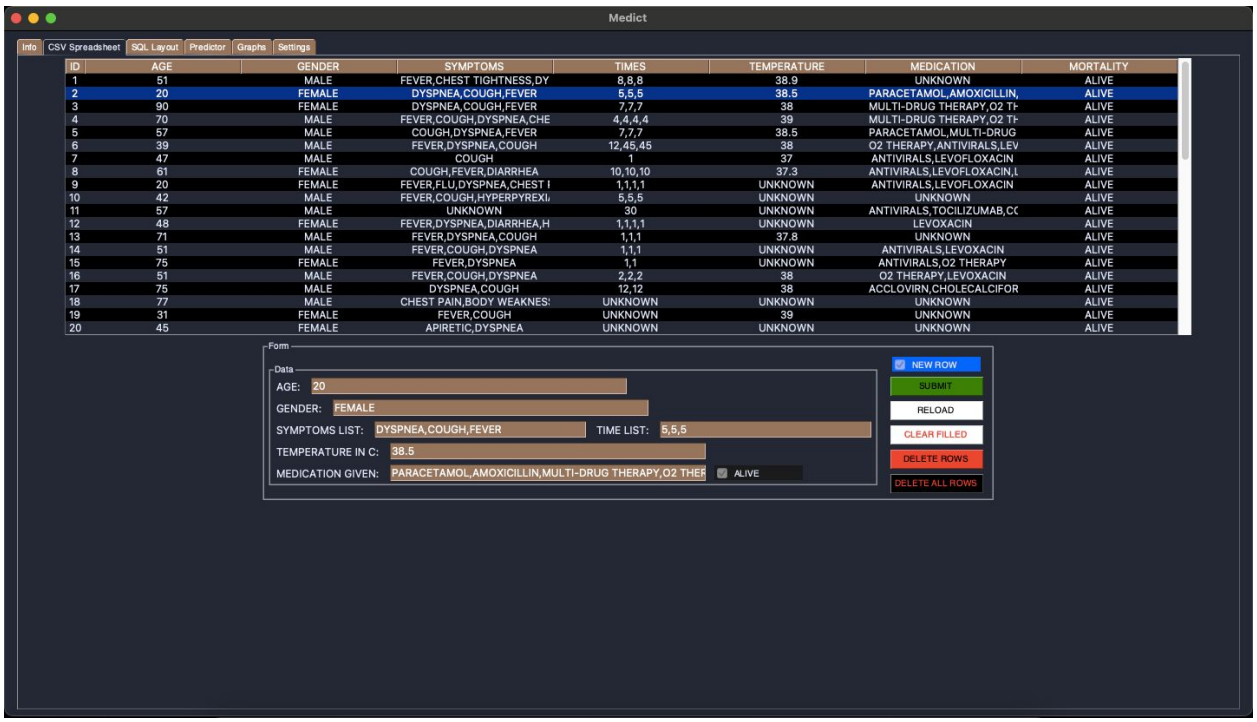
Application Output Images:

1. DATABASE INFORMATION:
    a. DBMS: MySQL
    b. SQL Database Host = `localhost`
    c. SQL Database Username = `root`
    d. SQL Database Password = `password21`
    e. SQL Database Name = `hospital`
    f. SQL Table Name = `patients`
2. Settings Information:
    a. Theme = `DarkTanBlue`, (later `DarkAmber`)
    b. Font Size = `15`

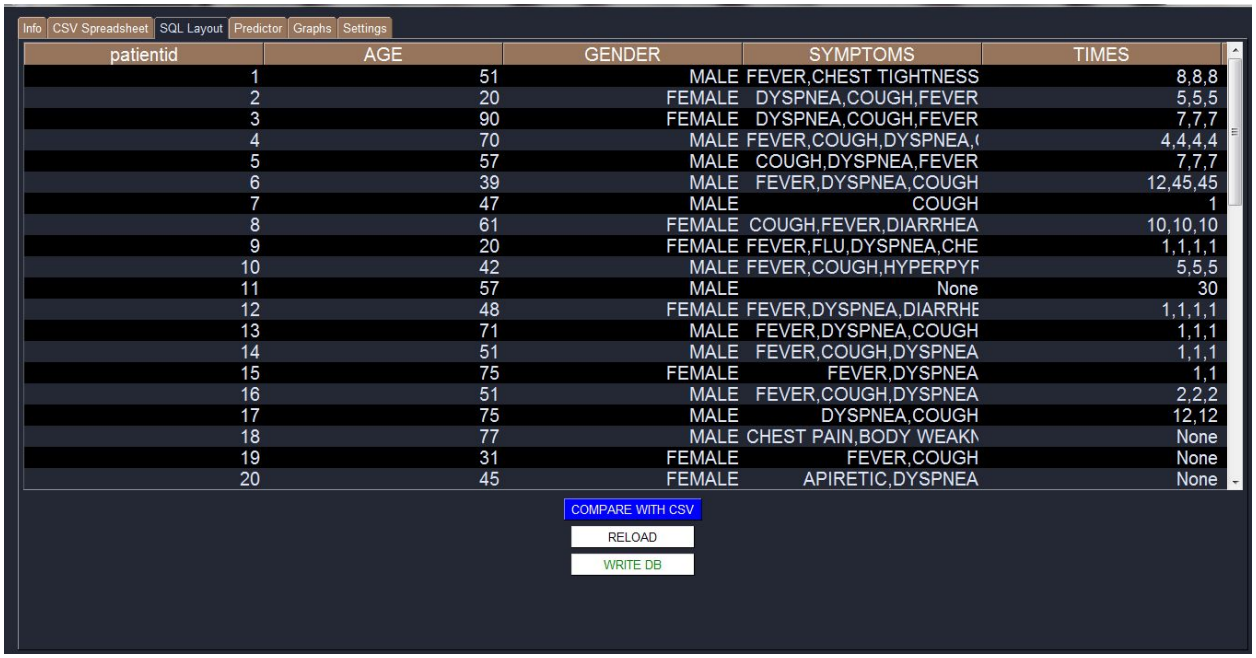
Application Instructions Tab:



CSV Spreadsheet Viewer Tab:

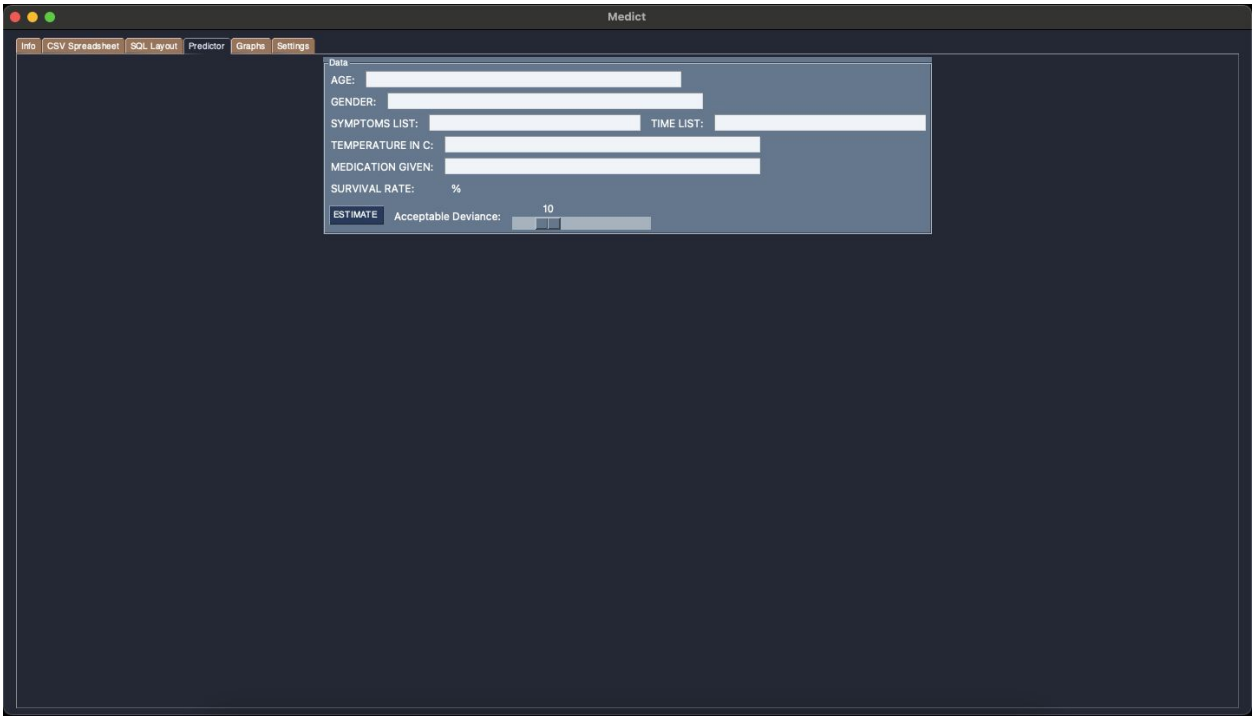


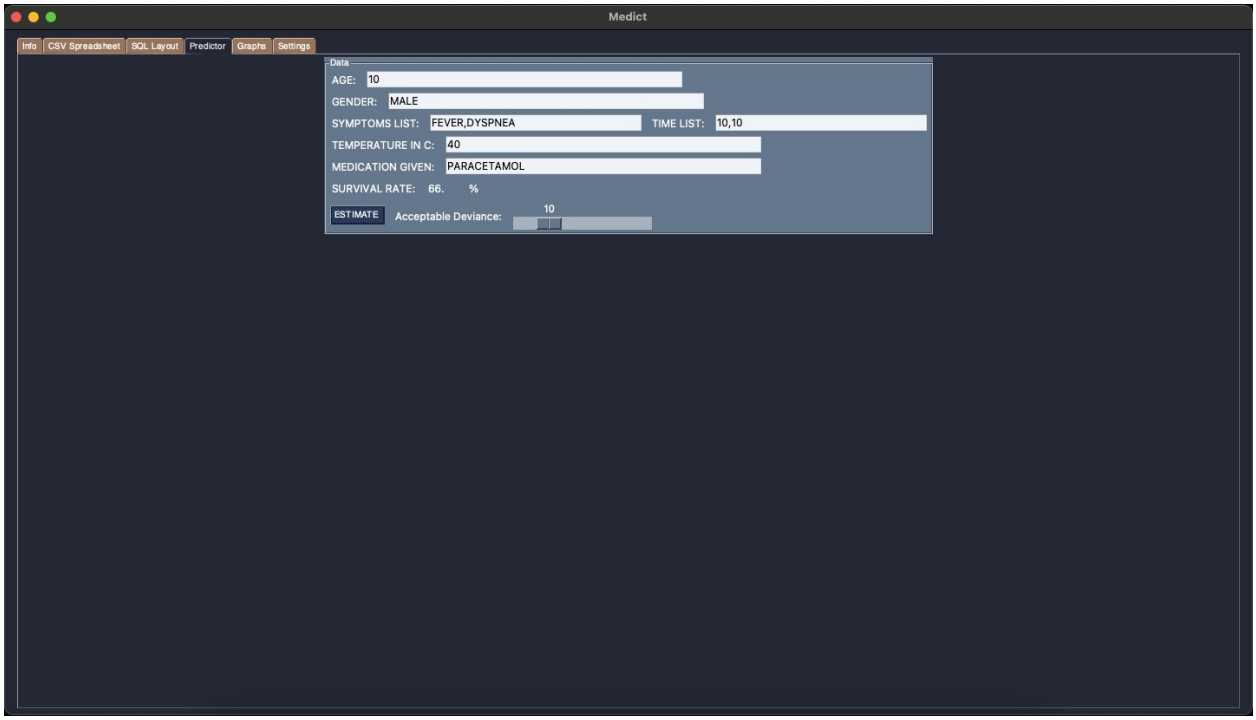
SQL Table Viewer Tab:

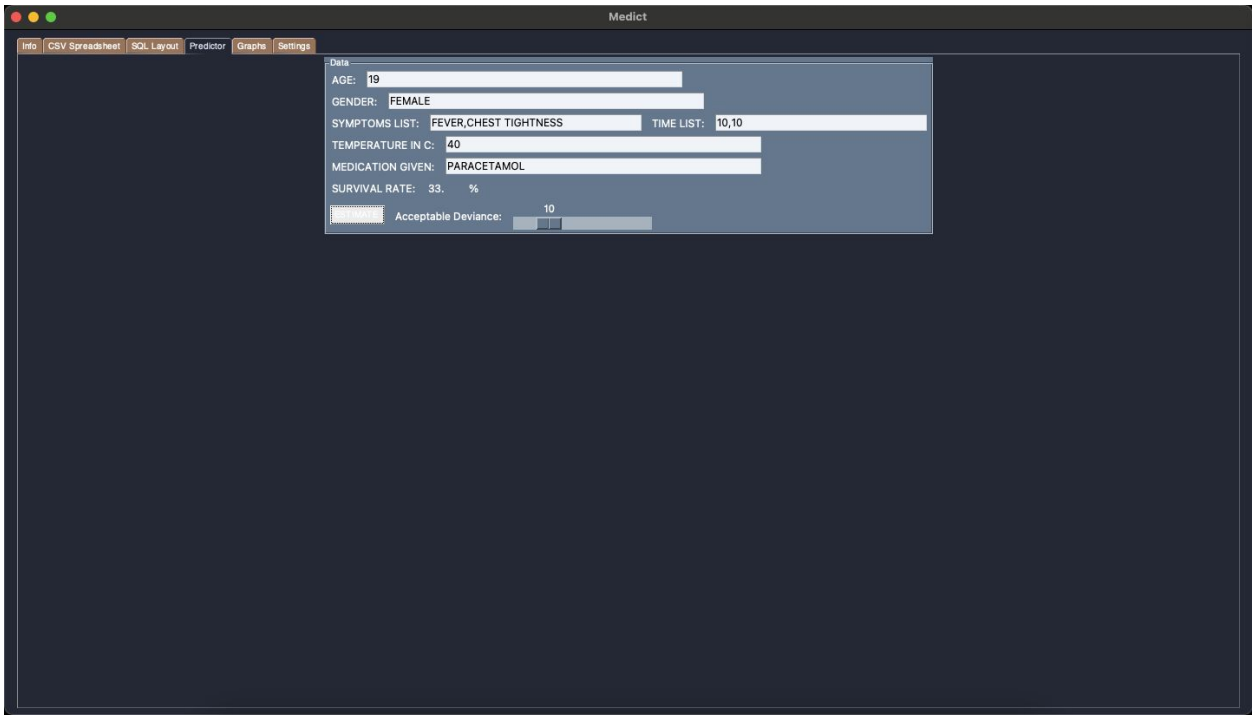


patientid	AGE	GENDER	SYMPTOMS	TIMES
1	51	MALE	FEVER,CHEST TIGHTNESS	8,8,8
2	20	FEMALE	DYSPNEA,COUGH,FEVER	5,5,5
3	90	FEMALE	DYSPNEA,COUGH,FEVER	7,7,7
4	70	MALE	FEVER,COUGH,DYSPNEA,(	4,4,4,4
5	57	MALE	COUGH,DYSPNEA,FEVER	7,7,7
6	39	MALE	FEVER,DYSPNEA,COUGH	12,45,45
7	47	MALE	COUGH	1
8	61	FEMALE	COUGH,FEVER,DIARRHEA	10,10,10
9	20	FEMALE	FEVER,FLU,DYSPNEA,CHE	1,1,1,1
10	42	MALE	FEVER,COUGH,HYPERPYF	5,5,5
11	57	MALE	None	30
12	48	FEMALE	FEVER,DYSPNEA,DIARRHE	1,1,1,1
13	71	MALE	FEVER,DYSPNEA,COUGH	1,1,1
14	51	MALE	FEVER,COUGH,DYSPNEA	1,1,1
15	75	FEMALE	FEVER,DYSPNEA	1,1
16	51	MALE	FEVER,COUGH,DYSPNEA	2,2,2
17	75	MALE	DYSPNEA,COUGH	12,12
18	77	MALE	CHEST PAIN,BODY WEAKI	None
19	31	FEMALE	FEVER,COUGH	None
20	45	FEMALE	APIRETIC,DYSPNEA	None

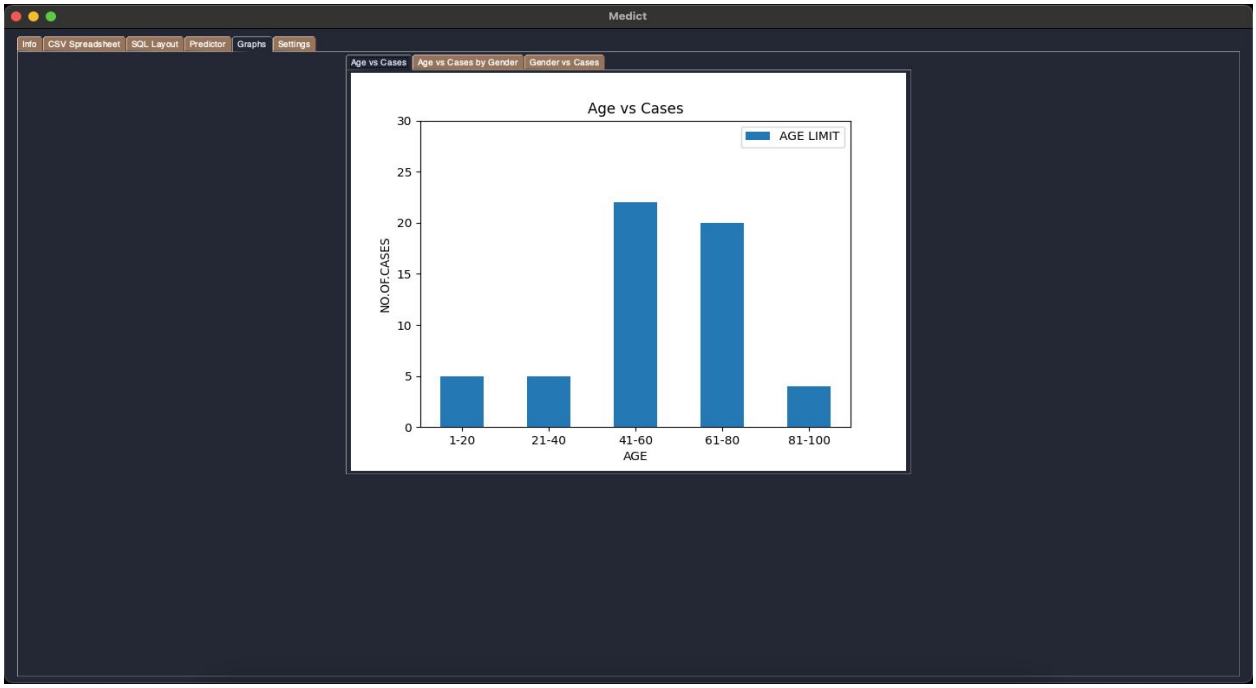
Life Expectancy Predictor Tab:

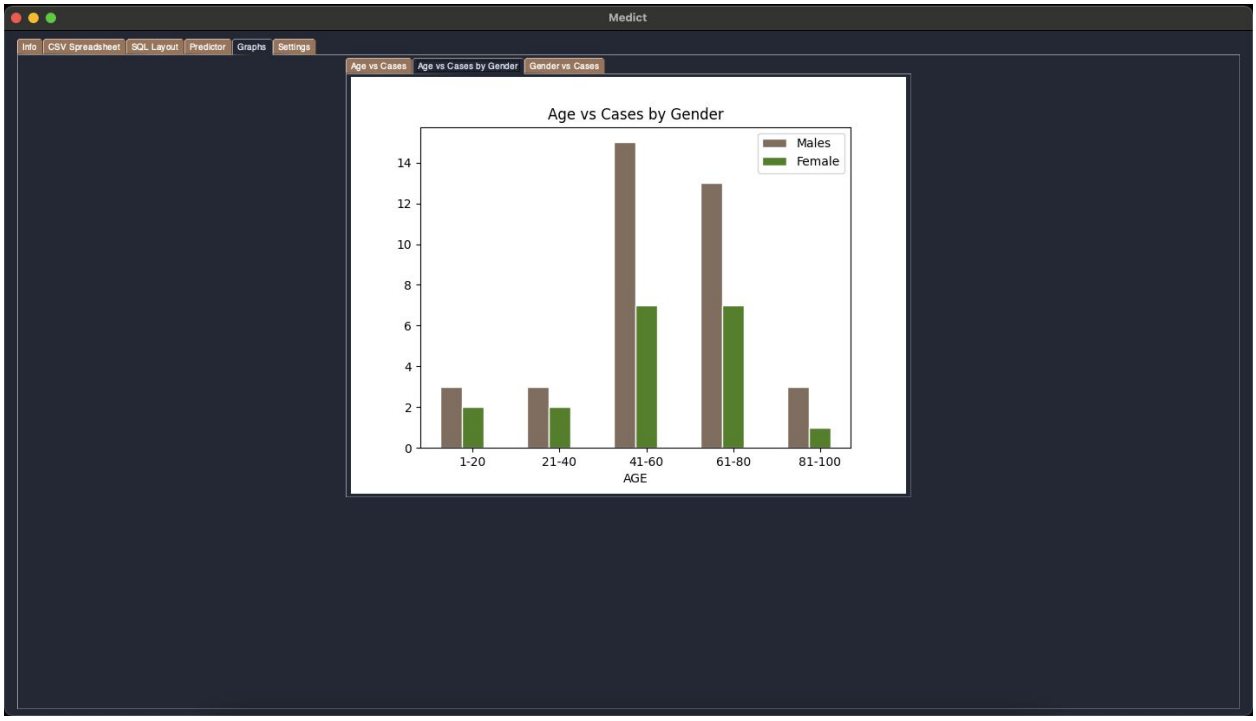


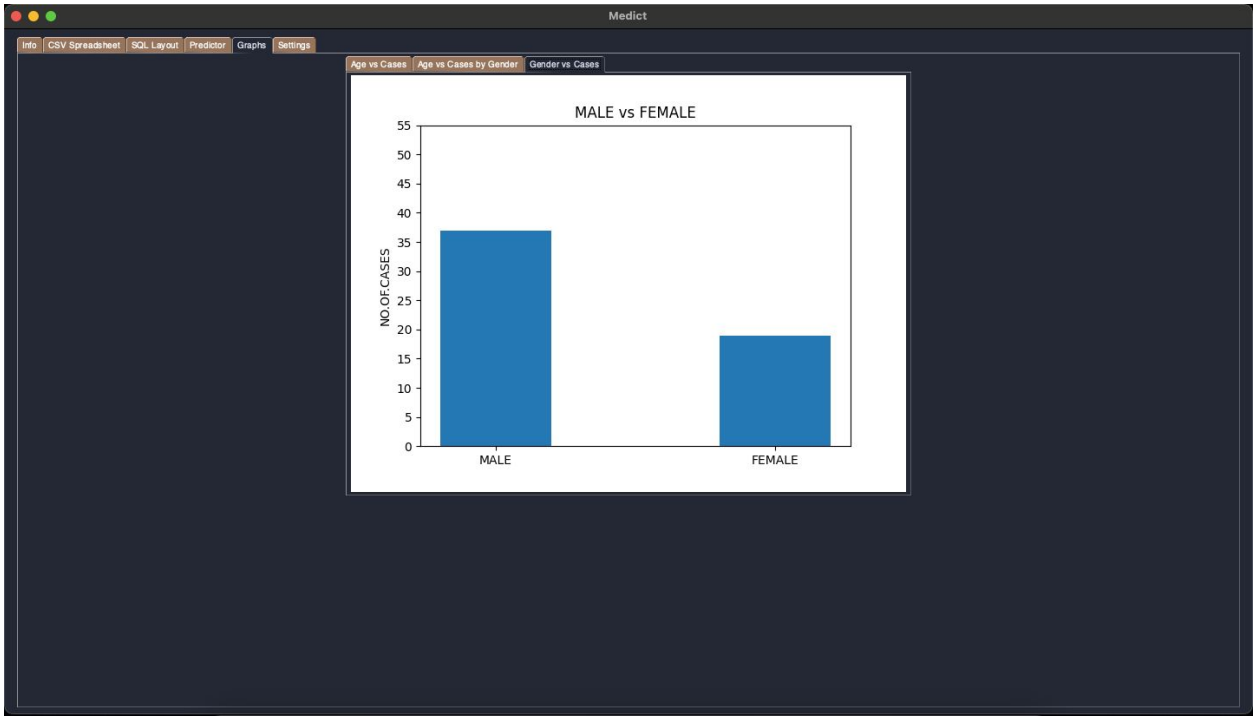

Medict
Info
CSV Spreadsheet
SQL Layout
Predictor
Graphs
Settings
Data
AGE: 10
GENDER: MALE
SYMPTOMS LIST: FEVER,DYSPNEA
TIME LIST: 10,10
TEMPERATURE IN C: 40
MEDICATION GIVEN: PARACETAMOL
SURVIVAL RATE: 66. %
ESTIMATE
Acceptable Deviance:
10


Medict
Info
CSV Spreadsheet
SQL Layout
Predictor
Graphs
Settings
Data
AGE: 19
GENDER: FEMALE
SYMPTOMS LIST: FEVER,CHEST TIGHTNESS
TIME LIST: 10,10
TEMPERATURE IN C: 40
MEDICATION GIVEN: PARACETAMOL
SURVIVAL RATE: 33. %
Acceptable Deviance:
10

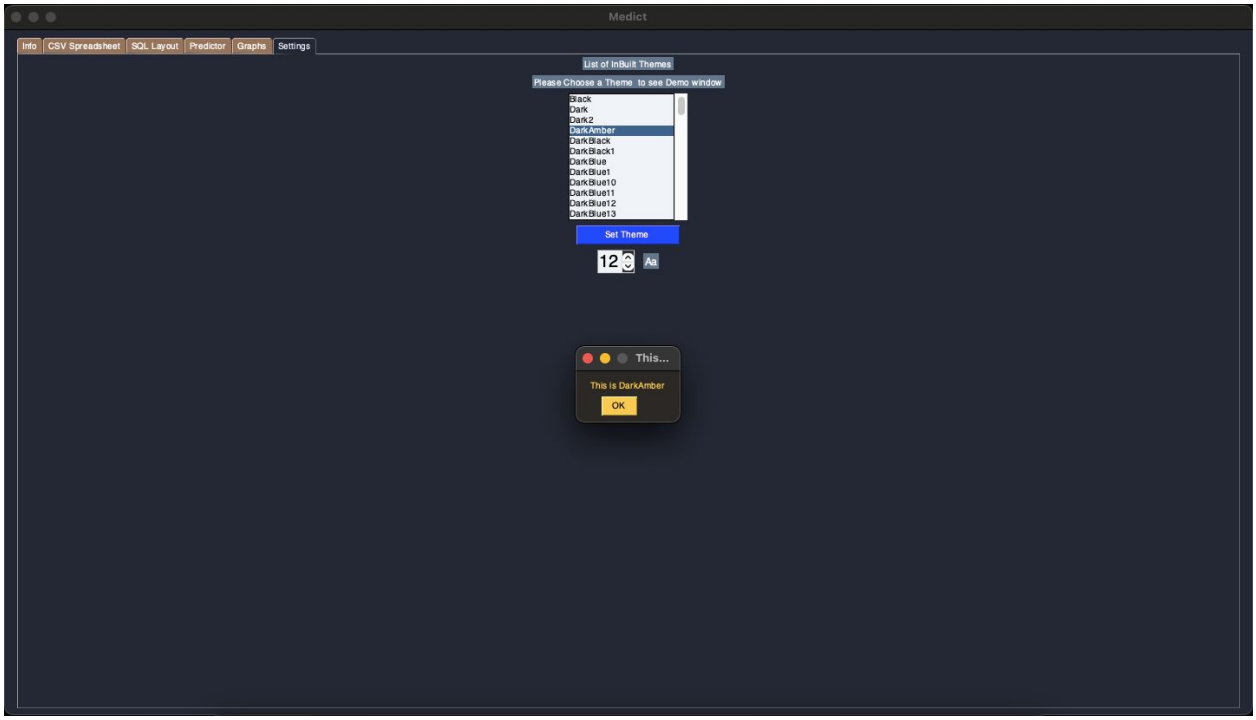
Data Visualiser Tab:

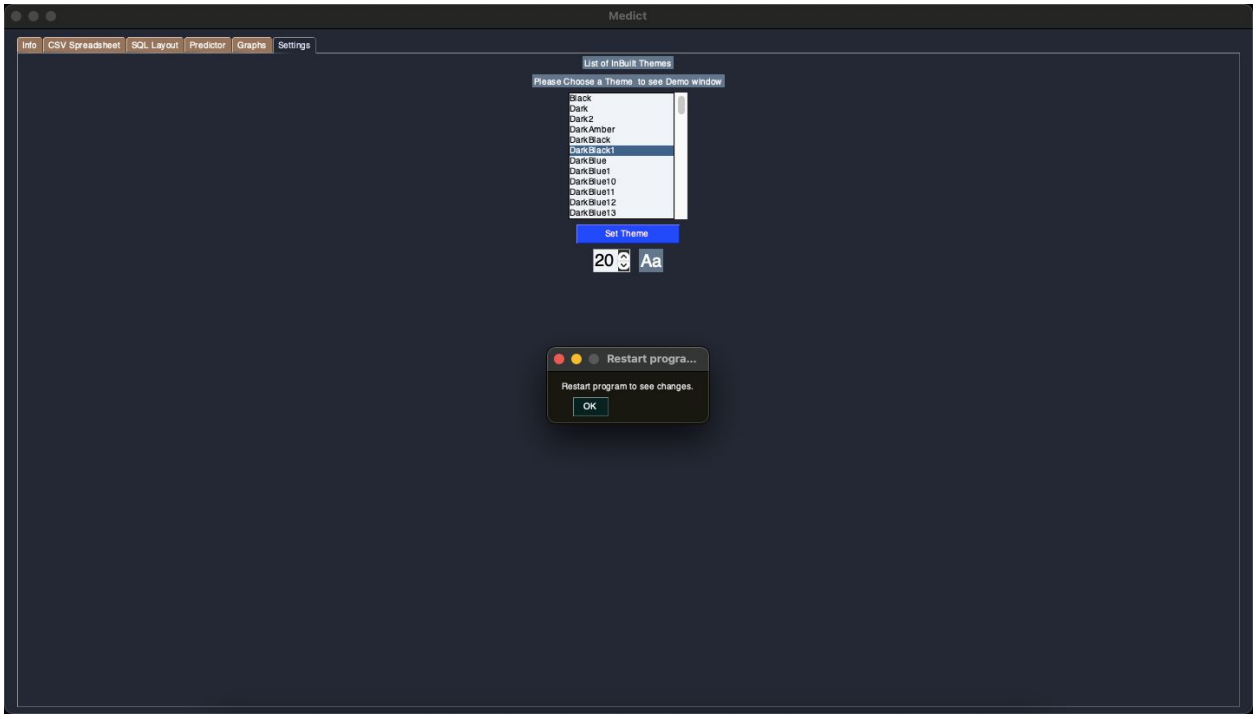


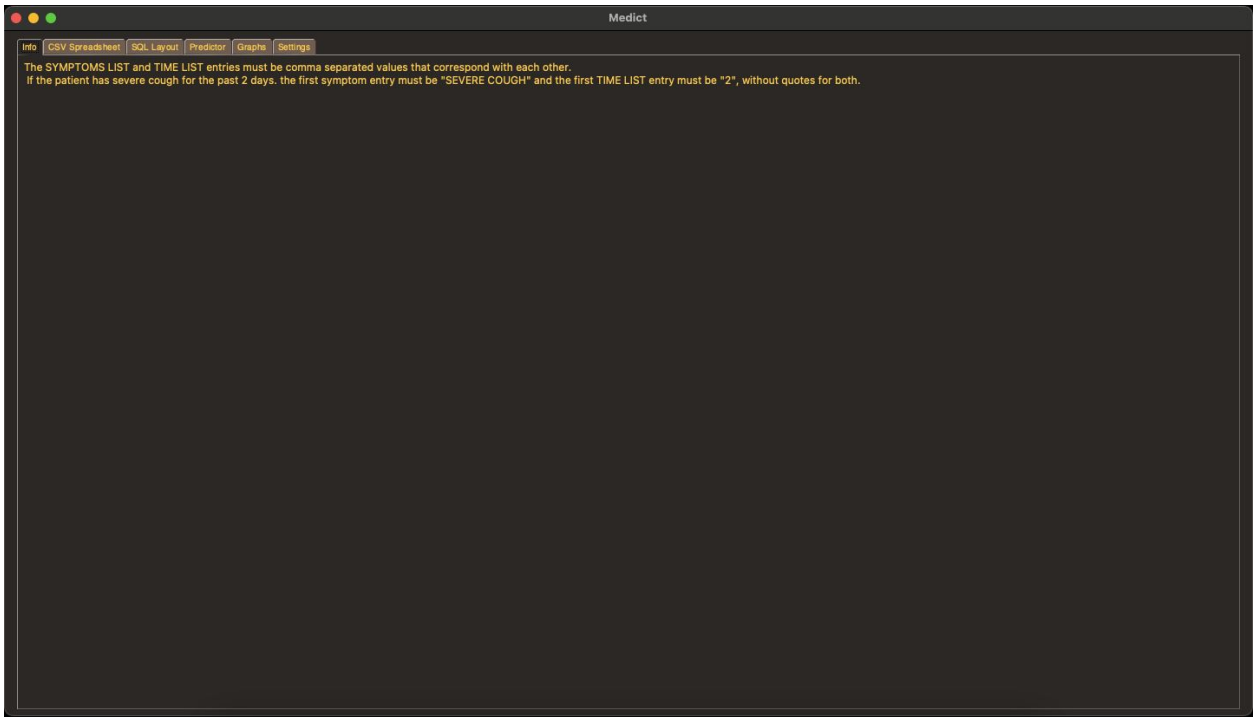




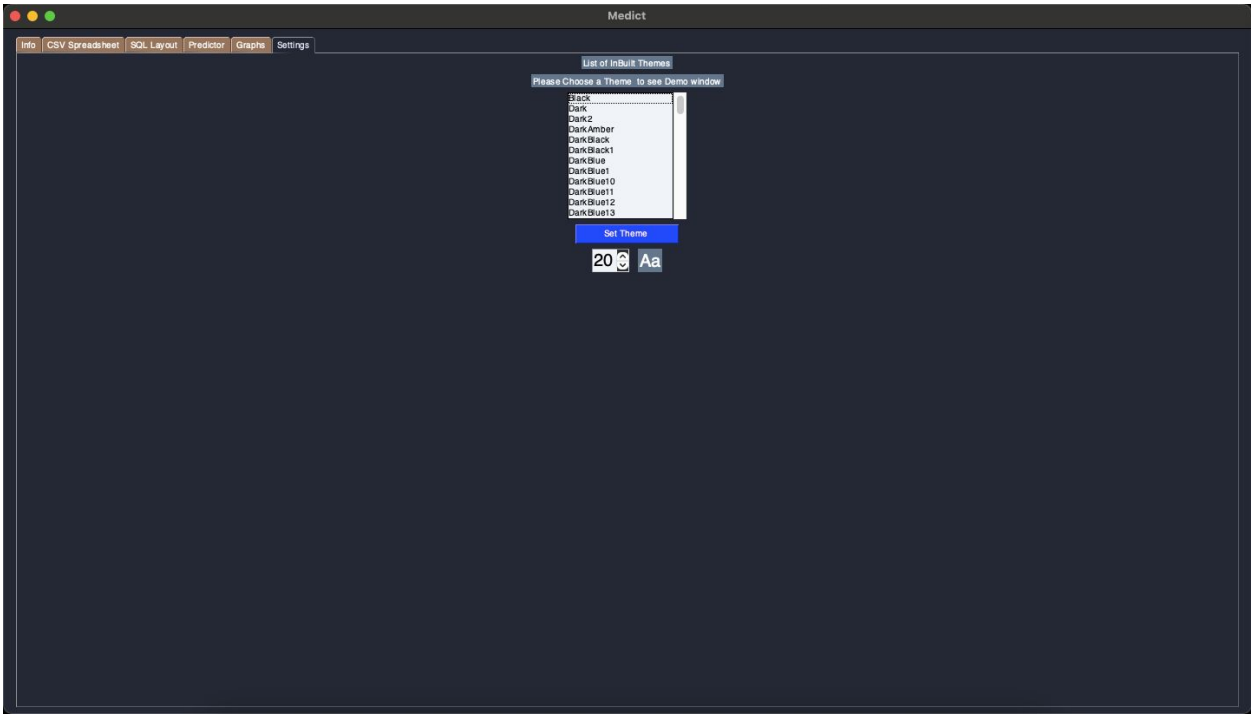
Settings Tab - Theme Changing:




Medict
Info
CSV Spreadsheet
SQL Layout
Predictor
Graphs
Settings
List of InBuilt Themes
Please Choose a Theme to see Demo window
Black
Dark
Dark2
DarkAmber
DarkBlack
DarkBlack1
DarkBlue
DarkBlue1
DarkBlue10
DarkBlue11
DarkBlue12
DarkBlue13
Set Theme
20
Aa
Restart progra...
Restart program to see changes.
OK


Medict
Info
CSV Spreadsheet
SQL Layout
Predictor
Graphs
Settings
The SYMPTOMS LIST and TIME LIST entries must be comma separated values that correspond with each other.
If the patient has severe cough for the past 2 days. the first symptom entry must be "SEVERE COUGH" and the first TIME LIST entry must be "2", without quotes for both.

Setting Tab - Font Size Changing:



Application Source Code:

Main Program:

./medict.py

```
"""
   ./medict.py

   This is the main Python file which must be run to activate the application.
   It makes the necessary imports and in a modular manner builds the user interface
   for easy use.
"""

import os
import csv

try:
   import PySimpleGUI as sg
except ModuleNotFoundError:
   raise ModuleNotFoundError("The PySimpleGUI module needs to be installed.")

from managers import (
   CSVManager,
   ThemeManager,
   Predictor,
   BarGraphManager,
   SQLManager,
   get_settings_config,
   FontManager,
)

if __name__ == "__main__":
   sg.theme(get_settings_config()["theme"])
   os.chdir(os.path.abspath(os.path.dirname(__file__)))
```

```
    csvmanager = CSVManager()
    predictor = Predictor()
    sqlmanager = SQLManager()
    bargraphman = BarGraphManager()
    thememanager = ThemeManager()
    fontmanager = FontManager()
    info_layout = [[sg.Text(csvmanager.INSTRUCTIONS, font=(csvmanager.TEXTFONT,
12))]]
    layout = [  # Main Window layout
        [
            sg.TabGroup(
                [
                    [
                        sg.Tab("Info", info_layout),
                        sg.Tab(
                            "CSV Spreadsheet",
                            csvmanager.spread_layout,
                            element_justification="center",
                        ),
                        sg.Tab(
                            "SQL Layout",
                            sqlmanager.spread_layout,
                            element_justification="center",
                        ),
                        sg.Tab(
                            "Predictor",
                            predictor.layout,
                            element_justification="center",
                        ),
                        sg.Tab(
                            "Graphs",
                            bargraphman.layout,
                            element_justification="center",
                        ),
                        sg.Tab(
                            "Settings",
                            [*thememanager.layout, *fontmanager.layout],
                            element_justification="center",
```

```
                    ),
                ]
            ],
            enable_events=True,
            key="tab",
        )
    ]
]

window = sg.Window("Medict", layout, resizable=True, finalize=True)
window.maximize()
window["tab"].expand(True, True, True)

while True:  # Main event loop.
    event, values = window.read()

    if event in (None, "Exit"):
        break

    elif event == "tab":
        csvmanager.clear_data(values, window)
        csvmanager.reload_table()
        sqlmanager.reload_table()

    elif event == "bargraph_tab":
        window["-CANVAS-"].TKCanvas.delete("all")
        window["-GENDER_CANVAS-"].TKCanvas.delete("all")
        window["-MVF_CANVAS-"].TKCanvas.delete("all")

        if values["bargraph_tab"] == "age-vs-case":
            fig_photo = bargraphman.draw_figure(
                window["-CANVAS-"].TKCanvas, bargraphman.fig
            )
        if values["bargraph_tab"] == "case-vs-gender":

            GENDER_CANVAS = bargraphman.draw_figure(
                window["-GENDER_CANVAS-"].TKCanvas, bargraphman.fig1
            )  # assign to variable or else the graph is killed.
```

```
        if values["bargraph_tab"] == "male-vs-female":
            mvf_graph = bargraphman.draw_figure(
                window["-MVF_CANVAS-"].TKCanvas, bargraphman.mvf_fig
            )
    elif event == "csvtable":  # Table is clicked etc.
        if len(csvmanager.table.SelectedRows) > 0:
            row = csvmanager.table.SelectedRows[-1]
            for i in range(len(values.keys())):
                key = list(values.keys())[i]
                if key in list(csvmanager.FIELDS):
                    window[key](csvmanager.table.get()[row][i - 1])

    elif event == "SUBMIT":
        csvmanager.submit_filled(values)

    elif event == "RELOAD":
        csvmanager.table.update(values=csvmanager.records_from_csv())

    elif event == "THEMEBTN":
        if len(values['THEMELIST']) > 0:
            sg.theme(values['THEMELIST'][0])
            thememanager.set_theme(values["THEMELIST"][0])
        if values["FONTSPIN"] != fontmanager.fontSize:
            fontmanager.set_fontsize(values["FONTSPIN"])
        sg.popup_ok("Restart program to see changes.",keep_on_top=True)
    elif event == "THEMELIST":
        sg.theme(values["THEMELIST"][0])
        sg.popup_ok("This is {}".format(values["THEMELIST"][0]),keep_on_top=True)
    elif event == "FONTSPIN":
        window["FONTSPIN"].update(values["FONTSPIN"])
        window["FontPreview"].update(font = "Helvetica "+str(values['FONTSPIN']))
    elif event == "RELOADSQL":
        sqlmanager.reload_table()

    elif event == "CLEAR FILLED":
        csvmanager.clear_data(values, window)
    elif event == "DELETE ROWS":
        csvmanager.delete_selected_rows()
```

```
        elif event == "DELETE ALL ROWS":
            csvmanager.delete_all_rows()
        elif event == "WRITEDB":
            if sqlmanager.sql_to_list() == []:
                sqlmanager.write_database(
                    tuple(csvmanager.FIELDS), tuple(csvmanager.records_from_csv())
                )

        elif event == "ESTIMATE":
            data = {
                field: values[field] for field in values if field in predictor.FIELDS
            }
            if not all(data.values()):
                sg.popup(csvmanager.UNFILLED_DATA_ERROR)
            else:
                data["AGE"] = int(data.pop("pAGE"))
                data["GENDER"] = data.pop("pGENDER")
                data["SYMPTOMS"] = (
                    ["UNKNOWN"]
                    if data["pSYMPTOMS"] == "UNKNOWN"
                    else data["pSYMPTOMS"].split(",")
                )
                data["TIMES"] = (
                    ["UNKNOWN"]
                    if data["pTIMES"] == "UNKNOWN"
                    else [int(time) for time in data["pTIMES"].split(",")]
                )
                data["TEMPERATURE"] = (
                    ["UNKNOWN"]
                    if data["pTEMPERATURE"] == "UNKNOWN"
                    else float(data["pTEMPERATURE"])
                )
                data["MEDICATION"] = (
                    ["UNKNOWN"]
                    if data["pMEDICATION"] == "UNKNOWN"
                    else data["pMEDICATION"].split(",")
                )
```

```
                del data["pSYMPTOMS"]
                del data["pTIMES"]
                del data["pTEMPERATURE"]
                del data["pMEDICATION"]

                data = csvmanager.expanded_dataset([data])[
                    0
                ]  # expanded_dataset returns a list. In this case, only one element
is in the list.
                prediction = predictor.predict(
                    data, allowed_deviation=values["DEVIANCE"]
                )
                window["pMORTALITY"].update(str(prediction))
        else:
            print(event)

    window.close()
```

Auxiliary Programs:

./managers/init.py

```
"""
   ./managers/__init__.py

   This File is used to make the managers directory into a module, so that its
contents,
   CSVManager, SQLManager etc can be imported with ease.
"""

from .csvmanager import CSVManager
```

```
from .sqlview import SQLManager

from .predictmanager import Predictor

from .bargraphmanager import BarGraphManager

from ._config import *

from .thememanager import ThemeManager

from .fontmanager import FontManager
```

./managers/csvmanager.py

```
"""
   ./managers/csvmanager.py

   This File contains the object CSVManager, which handles all CSV file related
   operations that may arise during the use of the main program application.

   This file may be run on its own to test the features of CSVManager separately.
"""

import os
import csv
import PySimpleGUI as sg

if __name__ == "__main__":
   from _config import get_settings_config
else:
   from ._config import get_settings_config

class CSVManager(object):

   INSTRUCTIONS=" ".join(
       [
           "The SYMPTOMS LIST and TIME LIST entries must be",
```

```
            "comma separated values that correspond with each other.\n",
            "If the patient has severe cough for the past 2 days.",
            "the first symptom entry must be \"SEVERE COUGH\" and the first",
            "TIME LIST entry must be \"2\", without quotes for both."
            ]
        )

    ROW_WARN="Are you ABSOLUTELY SURE you want to DELETE the selected row(s)?"

    UNFILLED_DATA_ERROR="\n".join(
        [
            "Some (or all) fields were left empty.",
            "Please use UNKNOWN as the entry if you don't know the data!"
            ]
        )

    FIELDS=["AGE","GENDER","SYMPTOMS","TIMES","TEMPERATURE","MEDICATION","MORTALITY"]

    def
__init__(self,TEXTFONT="serif",FONTSIZE=get_settings_config()["fontsize"],NUM_ROWS=2
0,CSVFILE=os.path.abspath(__file__+os.sep+os.pardir+os.sep+os.pardir+os.sep+"data.cs
v")):
        """Initialises the CSV Manager.
            Good fontsizes are [12,21]
        Args:
            TEXTFONT (str, optional): The font to use for all text. Defaults to 12.
            FONTSIZE (int, optional): The fontsize to use for all text. Defaults to
"serif".
            NUM_ROWS (int, optional): The number of rows to display in the Table.
Defaults to 20
            CSVFILE (str, optional): The path (relative or full) to the csv file to
read from.
            Defaults to "data.csv" in the same directory as the importing script.
        """
        self.TEXTFONT=TEXTFONT
        self.FONTSIZE=FONTSIZE
        self.NUM_ROWS=NUM_ROWS
        self.CSVFILE=CSVFILE
```

```
        self.spread_layout=[
            [
                sg.Table(

values=self.records_from_csv(),headings=self.FIELDS,key="csvtable",

display_row_numbers=True,header_font=(self.TEXTFONT,self.FONTSIZE),
                    alternating_row_color="black", auto_size_columns=False,
                    def_col_width=20,size=(None,self.NUM_ROWS),
select_mode="extended",

enable_events=True,font=(self.TEXTFONT,self.FONTSIZE),justification="center"
                    )
                ],
            [
                sg.Frame("Form",
                    [[
                        sg.Frame(
                            "Data",[
                            [sg.Text("AGE:",font=(self.TEXTFONT,self.FONTSIZE)),
sg.Input(key="AGE",font=(self.TEXTFONT,self.FONTSIZE))],

[sg.Text("GENDER:",font=(self.TEXTFONT,self.FONTSIZE)),sg.Input(key="GENDER",font=(s
elf.TEXTFONT,self.FONTSIZE))],
                            [
                                sg.Text("SYMPTOMS
LIST:",font=(self.TEXTFONT,self.FONTSIZE)),

sg.Input(key="SYMPTOMS",font=(self.TEXTFONT,self.FONTSIZE),size=(30,1)),
                                sg.Text("TIME
LIST:",font=(self.TEXTFONT,self.FONTSIZE)),

sg.Input(key="TIMES",font=(self.TEXTFONT,self.FONTSIZE),size=(30,1))
                                ],
                            [sg.Text("TEMPERATURE IN
C:",font=(self.TEXTFONT,self.FONTSIZE)),sg.Input(key="TEMPERATURE",font=(self.TEXTFO
NT,self.FONTSIZE))],
                            [
```

```
                                sg.Text("MEDICATION
GIVEN:",font=(self.TEXTFONT,self.FONTSIZE)),sg.Input(key="MEDICATION",font=(self.TEX
TFONT,self.FONTSIZE)),

sg.Checkbox('ALIVE',key="ALIVE",default=True,size=(16,2),background_color=("#1b1b1b"
)),
                                ],
                            ]),

                        sg.Column(
                            [
                                [sg.Checkbox('NEW ROW',key="NEW
ROW",default=True,size=(16,2),background_color=("#0366fc"))],

[sg.Button(button_text="SUBMIT",button_color=("black","green"),size=(16,1))],

[sg.Button(button_text="RELOAD",button_color=("black","WHITE"),size=(16,1))],
                                [sg.Button(button_text="CLEAR
FILLED",button_color=("RED","WHITE"),size=(16,1))],
                                [sg.Button(button_text="DELETE
ROWS",button_color=("BLACK","RED"),size=(16,1))],
                                [sg.Button(button_text="DELETE ALL
ROWS",button_color=("red","black"),size=(16,1))],
                                ],justification="right",
                            element_justification="center")
                        ]]
                    )
                ]
            ]

        self.table=self.spread_layout[0][0]
        self.table.StartingRowNumber=1
        self.table.RowHeaderText="ID"

    def clear_data(self, values = None, window = None):
        """Clears all the unpushed data filled in the form.

        Args:
```

```
            values (dict, optional): The PySimpleGUI values dictionary from
            where the submitted values must be extracted. Pass this if, and
            only if you are importing this from another file.

            window (PySimpleGUI.window, optional): Pass the PySimpleGUI window
            object here if and only if you are importing this file from another
            main program. This is so this function can modify that screen.
        """
        values=globals()["values"] if values is None else values
        window=globals()["window"] if window is None else window
        for key in values.keys():
            if key in self.FIELDS:
                window[key]('')

    def records_from_csv(self,datafile=None):
        """Returns a list of (list of entries for each field)
        for each row of the CSV file.

        Args:
            datafile (str,optional): The path to the csv file. Defaults to None but
            is self.CSVFILE if None.
        Returns:
            list of lists: The outer list holds each row, the inner list holds
                           each value in that row for each field.
        """
        if datafile is None:
            datafile=self.CSVFILE
        with open(datafile,'r') as csvfile:
            csv_reader = csv.DictReader(csvfile)
            return [
                [
                    row[fieldname] for fieldname in csv_reader.fieldnames
                    ] for row in csv_reader
                ]

    def list_od_from_csv(self,datafile=None):
        """Returns a list of ordered dictionaries that map each field to its value
        for each row in the CSV file.
```

```
        Args:
            datafile (str,optional): The path to the csv file. Defaults to None but
            is self.CSVFILE if None.
        Returns:
            list[OrderedDictionary]: List containing the ordered dictionaries that
                                     map the field to their values, for each row.
        """
        if datafile is None:
            datafile=self.CSVFILE
        with open(datafile, "r") as csvfile:
            data=csv.DictReader(csvfile)
            datalist=[d for d in data]
        return datalist

    def typed_list_od_from_csv(self):
        """Converts the String data of the CSV to list, int, float, whatevs.

        Returns:
            list: List of ordered dictionaries.
        """
        data=self.list_od_from_csv()
        for i in range(len(data)):
            data[i]["AGE"]=int(data[i]["AGE"])
            data[i]["SYMPTOMS"]= ["UNKNOWN"] if data[i]["SYMPTOMS"] == "UNKNOWN" else
data[i]["SYMPTOMS"].split(",")
            data[i]["TIMES"]=["UNKNOWN"] if data[i]["TIMES"] == "UNKNOWN" else
[int(time) for time in data[i]["TIMES"].split(",")]
            data[i]["TEMPERATURE"]=["UNKNOWN"] if data[i]["TEMPERATURE"] == "UNKNOWN"
else float(data[i]["TEMPERATURE"])
            data[i]["MEDICATION"]=["UNKNOWN"] if data[i]["MEDICATION"] == "UNKNOWN"
else data[i]["MEDICATION"].split(",")
        return data

    def unique_symptoms(self):
        """Returns all the unique symptoms experienced by all the patients.

        Returns:
```

```
            list: List of strings of the names of the symptoms
        """
        unique_symptoms=[]
        for entry in self.typed_list_od_from_csv():
            symptoms=entry["SYMPTOMS"]
            for symptom in symptoms:
                if symptom not in unique_symptoms and symptom!="UNKNOWN":
                    unique_symptoms.append(symptom)
        return unique_symptoms

    def unique_medications(self):
        """Returns all the unique medicines used by all the patients.

        Returns:
            list: List of strings of the names of the medicines
        """
        unique_medications=[]
        for entry in self.typed_list_od_from_csv():
            medications=entry["MEDICATION"]
            for medication in medications:
                if medication not in unique_medications and medication!="UNKNOWN":
                    unique_medications.append(medication)
        return unique_medications

    def expanded_dataset(self,ds=None):
        """Returns a list of ordered dictionaries of the details of the patient
        creating entries for all medications, symptoms etc. The value for each
symptom
        is the number of days the patient had the symptom, and for the medication, it
is
        1 if they used it, 0 if not.

        Returns:
            list: list of ordered dictionaries.
        """
        if ds==None:
            data=self.typed_list_od_from_csv()
        else:
```

```
            data=ds
        unique_symptoms=self.unique_symptoms()
        unique_medications=self.unique_medications()
        for record in data:
            if record["TEMPERATURE"]==["UNKNOWN"]:
                record["TEMPERATURE"]=37
            for i in range(len(unique_symptoms)):
                if unique_symptoms[i] in record["SYMPTOMS"]:
                    if len(record["TIMES"])==1:
                        if record["TIMES"][0]!="UNKNOWN":
                            record[unique_symptoms[i]]=record["TIMES"][0]
                        else:
                            record[unique_symptoms[i]]=0
                    else:
                        record[unique_symptoms[i]] =
record["TIMES"][record["SYMPTOMS"].index(unique_symptoms[i])]
                else:
                    record[unique_symptoms[i]]=0
            for i in range(len(unique_medications)):
                if unique_medications[i] in record["MEDICATION"]:
                    record[unique_medications[i]]=1
                else:
                    record[unique_medications[i]]=0
            if ds==None:
                record["MORTALITY"] = 1 if record["MORTALITY"]=="ALIVE" else 0
            del record["SYMPTOMS"]
            del record["TIMES"]
            del record["MEDICATION"]
        return data

    def write_list_od_to_csv(self,list_of_ordered_dicts,datafile=None):
        """Writes a list of ordered dictionaries that maps each field to its value
        for a single row, to the CSV file.

        Args:
            list_of_ordered_dicts (list[OrderedDict]): The list containing the
                            ordered dictionaries that map each field to its value.
```

```
        """
        if datafile is None:
            datafile=self.CSVFILE
        with open(datafile,"w",newline="") as csvfile:
            fields=list(list_of_ordered_dicts[0].keys())
            writer=csv.DictWriter(csvfile,fieldnames=fields,extrasaction="ignore")
            writer.writeheader()
            writer.writerows(list_of_ordered_dicts)

    def submit_filled(self, values=None):
        """Submits the data filled in the form by writing it to the CSV file.
        If the NEW ROW checkbox has been unchecked, it edits the currently
        selected row instead of adding a new row.

        Args:
            values (dict, optional): The PySimpleGUI values dictionary from
            where the submitted values must be extracted. Pass this if, and
            only if you are importing this from another file.
        """
        values=globals()["values"] if values is None else values

        data={
            field:values[field] for field in values if field in self.FIELDS[:-1]
            }
        if not all(data.values()):
            sg.popup(self.UNFILLED_DATA_ERROR)
        else:
            if values["NEW ROW"]==True:
                with open(self.CSVFILE, 'a') as csvfile:
                    data["MORTALITY"]="ALIVE" if values["ALIVE"]==True else "DEAD"
                    w = csv.DictWriter(csvfile, data.keys())
                    if csvfile.tell() == 0:
                        w.writeheader()
                    w.writerow(data)
            else:
                if self.table.SelectedRows!=[]:
                    new_rows=[]
                    datalist=self.list_od_from_csv()
```

```
                    for i in range(len(datalist)):
                        if i in self.table.SelectedRows:
                            for field in self.FIELDS:
                                if field != "MORTALITY":
                                    datalist[i][field]=values[field]
                                else:
                                    datalist[i][field]="ALIVE" if
values["ALIVE"]==True else "DEAD"
                    self.write_list_od_to_csv(datalist)
                else:
                    sg.popup("No row(s) selected!")
        sg.popup("Submitted!")
        self.reload_table()

    def reload_table(self):
        """Reloads the table by reading the CSV file and updating as necessary.
        """
        self.table.update(values=self.records_from_csv())

    def delete_selected_rows(self):
        """Deletes only the rows selected in the User Interface.
        """
        if self.table.SelectedRows != [] and sg.popup_yes_no(self.ROW_WARN)=="Yes":
            rows_left=[]
            datalist=self.list_od_from_csv()
            for i in range(len(datalist)):
                if i not in self.table.SelectedRows:
                    rows_left.append(datalist[i])
            self.write_list_od_to_csv(rows_left)
        elif self.table.SelectedRows == []:
            sg.popup("No Rows have been selected!")
        self.reload_table()

    def delete_all_rows(self):
        """Deletes all rows in the CSV.
        """
        confirm=sg.popup_yes_no("Are you sure you want to DELETE ALL ROWS?")
        if confirm=="Yes":
```

```
            with open(self.CSVFILE, 'r') as csvfile:
                firstline=csvfile.readline()
            with open(self.CSVFILE,"w") as csvfile:
                csvfile.write(firstline)
        self.reload_table()

if __name__=="__main__":
    sg.theme('DarkTanBlue')
    os.chdir(os.path.abspath(os.path.dirname(__file__)))

    csvmanager=CSVManager(CSVFILE="../data.csv")


info_layout=[[sg.Text(csvmanager.INSTRUCTIONS,font=(csvmanager.TEXTFONT,csvmanager.F
ONTSIZE))]]
    layout=[ # Main Window layout
        [
            sg.TabGroup(
                [
                    [
                        sg.Tab("Info",info_layout),

                        sg.Tab(
                            "SpreadSheet",csvmanager.spread_layout,
                            element_justification='center'
                            )
                        ]
                    ],
                enable_events=True,key="tab"
                )
            ]
        ]
    window = sg.Window("Data Enterer", layout,resizable=True).finalize()
    window["tab"].expand(True,True,True)
    window.Maximize()

    while True: #Main event loop.
        event, values = window.read()
```

```
        if event in (None, 'Exit'):
            break

        elif event=="tab":
            csvmanager.reload_table()

        elif event=="csvtable": #Table is clicked etc.
            row=csvmanager.table.SelectedRows[-1]
            for i in range(len(values.keys())):
                key = list(values.keys())[i]
                if key in list(csvmanager.FIELDS):
                    window[key](csvmanager.table.get()[row][i-1])

        elif event=="SUBMIT":
            csvmanager.submit_filled()

        elif event=="RELOAD":
            csvmanager.table.update(values=csvmanager.records_from_csv())

        elif event=="CLEAR FILLED":
            csvmanager.clear_data()

        elif event=="DELETE ROWS":
            csvmanager.delete_selected_rows()

        elif event == "DELETE ALL ROWS":
            csvmanager.delete_all_rows()

        else:
            print(event)

    window.close()
```

./managers/sqlview.py

```
"""
   ./managers/sqlview.py

   This File contains the object SQLManager which handles all functions
   related to interfacing with the SQL Database.

   This file may be run on its own to test the features of SQLManager separately.
"""

import json
from os import chdir, path

import mysql.connector as ms
import PySimpleGUI as sg

if __name__ == "__main__":
   from csvmanager import CSVManager
   from _config import get_sql_config,get_settings_config

else:
   from .csvmanager import CSVManager
   from ._config import get_sql_config,get_settings_config


class SQLManager(object):
   def __init__(
       self,
       TEXTFONT="serif",
       FONTSIZE=get_settings_config()["fontsize"],
       NUM_ROWS=20,
       SQLTableCreationPath=path.join("database.sql"),
   ):
       """Initialises the SQL Manager.
```

```
        Args:
            TEXTFONT (str, optional): The font to use for all text. Defaults to 15.
            FONTSIZE (int, optional): The fontsize to use for all text. Defaults to
"serif".
            NUM_ROWS (int, optional): The number of rows to display in the Table.
Defaults to 20
        """
        # MySQL Configuration Starts
        sql_config = get_sql_config()
        self.mySqlHost = sql_config["host"]
        self.mySqlUsername = sql_config["username"]
        self.mySqlDatabase = sql_config["database"]
        self.mySqlPassword = sql_config["password"]
        self.SQLTableName = sql_config["table_name"]
        self.SQLTableCreationPath = SQLTableCreationPath
        # MySQl Configuration ends
        self.TEXTFONT = TEXTFONT
        self.FONTSIZE = FONTSIZE
        self.NUM_ROWS = NUM_ROWS
        if self.checkTableExists():  # check table exists or not.
            self.FIELDS = self.show_table_rows()
        else:
            self.create_table()
            self.FIELDS = self.show_table_rows()
        self.spread_layout = [
            [
                sg.Table(
                    values=self.sql_to_list(),
                    headings=self.FIELDS,
                    key="sqltable",
                    display_row_numbers=False,
                    header_font=(self.TEXTFONT, self.FONTSIZE),
                    alternating_row_color="black",
                    auto_size_columns=False,
                    def_col_width=20,
                    size=(None, self.NUM_ROWS),
                    select_mode="extended",
                    enable_events=True,
```

```
                font=(self.TEXTFONT, self.FONTSIZE),
            )
        ],
        [
            sg.Column(
                [
                    [
                        sg.Button(
                            button_text="COMPARE WITH CSV",
                            button_color=("white", "blue"),
                            size=(18, 1),
                        )
                    ],
                    [
                        sg.Button(
                            button_text="RELOAD",
                            key="RELOADSQL",
                            button_color=("black", "WHITE"),
                            size=(16, 1),
                        )
                    ],
                    [
                        sg.Button(
                            button_text="WRITE DB",
                            key="WRITEDB",
                            button_color=("green", "WHITE"),
                            size=(16, 1),
                        )
                    ],
                ],
                justification="center",
                element_justification="center",
            ),
        ],
    ]
    self.table = self.spread_layout[0][0]
    self.table.StartingRowNumber = 1
    self.table.RowHeaderText = "ID"
```

```
    def reload_table(self):
        """Reloads the table by reading the Database and updating as necessary."""
        self.table.update(values=self.sql_to_list())

    def is_num(self, var):
        """Some checking of value of data written in SQL database and converting
it"""
        if var == "null":  # Check whether it is null
            return True
        try:
            var = int(var)
            return True
        except ValueError:
            return False

    def intialise_dataBase(self):
        """Initialises all the variable required for qurying MySql"""
        # global con,cursor
        self.con = ms.connect(
            host=self.mySqlHost,
            user=self.mySqlUsername,
            passwd=self.mySqlPassword,
            database=self.mySqlDatabase,
        )
        self.cursor = self.con.cursor()

    def checkTableExists(self, tablename="patients"):
        """Check whether the table `tablenames` already exits"""
        self.intialise_dataBase()
        self.cursor.execute(
            """
            SELECT COUNT(*)
            FROM information_schema.tables
            WHERE table_name = '{0}'
            """.format(
                tablename.replace("'", "''")
            )
```

```
        )
        if self.cursor.fetchone()[0] == 1:
            return True
        return False

    def sql_to_list(self):
        """Converts things in sql database to lists which contain rows as tuples
        Eg.
        [
            (1, 51, 'MALE', '["FEVER, CHEST TIGHTNESS, DYSPNEA"]', '["8, 8, 8"]',
Decimal('38.90'), 'ANTI-INFLAMMATORY', None)
            (2, 20, 'FEMALE', '["DYSPNEA, DRY COUGH, FEVER"]', '["5, 5, 5"]',
Decimal('38.50'), 'SELF:PARACETAMOL, AMOXILLIN', None)
            (3, 90, 'FEMALE', '["DYSPNEA, DRY COUGH, DEVER"]', '["7+"]',
Decimal('38.00'), 'MULTI-DRUG THERAPY, O2-THERAPY', None)
            (4, 70, 'MALE', '["FEVER, DRY COUGH, DYSPNEA, CHEST PAIN"]', '["4, 4, 4,
4"]', Decimal('39.00'), 'MULTI-DRUG THERAPY, O2 THERAPY, INTUBATION, CPAP THERAPY',
None)
            (5, 57, 'MALE', '["DYSPNEA, DRY COUGH, FEVER"]', '["7, 7, 7"]',
Decimal('38.50'), 'MULTI-DRUG THERAPY, O2-THERAPY', None)
        ]
        """
        self.intialise_dataBase()
        self.cursor.execute("SELECT * from %s" % self.SQLTableName)
        items = self.cursor.fetchall()
        tempItems = []
        finalItems = []
        for item in items:
            for things in item:
                if type(things) == str:
                    if "[" in things:
                        things = json.loads(str(things))
                if type(things) == list:
                    tempItems.append(str(things)[2:-2])
                else:
                    tempItems.append(str(things))
            else:
                finalItems.append(tempItems)
```

```
                tempItems = []
        return finalItems

    def create_table(self):
        """This create a table defined in `database.sql`"""
        self.intialise_dataBase()
        with open(self.SQLTableCreationPath, "r") as f:
            SQLStatement = f.read()
        self.cursor.execute(SQLStatement)

    def show_table_rows(self):
        """This function would return the column names in SQL Tables."""
        self.intialise_dataBase()
        self.cursor.execute("DESC %s" % self.SQLTableName)
        desc = self.cursor.fetchall()
        return [i[0] for i in desc]

    def write_database(self, columnNames, rows):
        """Write into MySql table name as specified.
        columnNames -> List
        rows ->List
        tablesName ->str
        """
        self.intialise_dataBase()
        sympIndex = columnNames.index("SYMPTOMS")
        timesIndex = columnNames.index("TIMES")
        for row in rows:
            row[sympIndex] = (
                "null" if row[sympIndex] == "UNKNOWN" else
json.dumps([row[sympIndex]])
            )
            row[timesIndex] = (
                "null"
                if row[timesIndex] == "UNKNOWN"
                else json.dumps([row[timesIndex]])
            )
            SQLStatement = "INSERT INTO %s(" % self.SQLTableName
            for i in range(len(columnNames)):
```

```
                if i != len(columnNames) - 1:
                    SQLStatement += columnNames[i] + ","
                else:
                    SQLStatement += columnNames[i] + ")"
            SQLStatement += " VALUES("
            for i in range(len(row)):
                if (
                    row[i] == "UNKNOWN"
                ):  # Make it to null in SQL instead of saving it as is as it cause
errors.
                    row[i] = "null"
                if i != (len(row) - 1) and self.is_num(row[i]):
                    SQLStatement += row[i] + ","
                elif i != (len(row) - 1) and (not self.is_num(row[i])):
                    SQLStatement += "'%s'" % row[i] + ","
                elif i == (len(row) - 1) and self.is_num(row[i]):
                    SQLStatement += row[i] + ");"
                elif i == (len(row) - 1) and (not self.is_num(row[i])):
                    SQLStatement += "'%s'" % row[i] + ");"
            print(SQLStatement)
            if self.checkTableExists() == True:
                self.cursor.execute(SQLStatement)
            else:
                self.create_table()
                self.cursor.execute(SQLStatement)
            self.con.commit()

    def update_row(self, patientid, fields_to_change, updated_data):
        """This updates a specific row in a SQL table with specific patientid.
        Parameters
        ----------
        patientid -> int
        fields_to_change -> List or Tuple
        updated_data -> List or Tuple

        Raises
        ------
        ValueError -> When length of `fields_to_change` and `updated_data` doesn't
```

```
match
        Norows affected


        Returns
        -------
        None
        """
        self.intialise_dataBase()
        SQLStatement = "UPDATE " + self.SQLTableName + " SET "
        if len(fields_to_change) == len(updated_data):
            for field, data in zip(fields_to_change, updated_data):
                SQLStatement += str(field) + "=" + "'%s'" % data + ", "
        else:
            raise ValueError(
                "Looks like fields_to_change and updated_data doesn't have equal
values."
            )
        SQLStatement = SQLStatement[:-2]  # remove last comma and space
        SQLStatement += " WHERE patientid=%s" % patientid + ";"
        self.cursor.execute(SQLStatement)
        if self.cursor.rowcount == 0:
            print("There is no Rows affected Please Check Your Parameters")
        else:
            self.con.commit()


if __name__ == "__main__":  # For if you want to run this standalone to edit
quickly.
   # Some check of whether function works is below
   """This is to test some functions"""
   csvmanager = CSVManager(CSVFILE=path.abspath(__file__ + "/../../data.csv"))
   a = csvmanager.records_from_csv()
   sqlmanager = SQLManager(TEXTFONT="Roboto Regular")
   print(sqlmanager.write_database(tuple(csvmanager.FIELDS), tuple(a)))
   print(sqlmanager.sql_to_list())
   print(sqlmanager.show_table_rows())
   sqlmanager.update_row(20, ["GENDER"], ["FEMALE"])
   sg.theme("DarkTanBlue")
```

```
chdir(path.abspath(path.dirname(__file__)))
info_layout = [[sg.Text("Hi")]]
layout = [  # Main Window layout
    [
        sg.TabGroup(
            [
                [
                    sg.Tab("Info", info_layout),
                    sg.Tab(
                        "SpreadSheet",
                        sqlmanager.spread_layout,
                        element_justification="center",
                    ),
                ]
            ],
            enable_events=True,
            key="tab",
        )
    ]
]

window = sg.Window("Data Enterer", layout).Finalize()
window.Maximize()

while True:  # Main event loop.
    event, values = window.read()

    if event in (None, "Exit"):
        print("Exiting")
        break

    elif event == "tab":
        sqlmanager.reload_table()
    elif event == "RELOADSQL":
        sqlmanager.reload_table()
        print("Reloaded")
    else:
        print(event)
```

```
    window.close()
```

./managers/predictmanager.py

```
"""
    ./managers/predictmanager.py

    This File contains the object Predictor which handles all functions
    related to predicting survival rate of a patient that may arise during
    the usage of the main program.

    This file may be run on its own to test the features of Predictor separately.
"""

from os import path
from functools import reduce

import PySimpleGUI as sg

if __name__=="__main__":
    from csvmanager import CSVManager
    from _config import get_settings_config
else:
    from .csvmanager import CSVManager
    from ._config import get_settings_config

class Predictor(object):
    TEXTFONT="serif"
    FONTSIZE=get_settings_config()["fontsize"]
    FIELDS=[]
    FIELDS=["pAGE","pGENDER","pSYMPTOMS","pTIMES","pTEMPERATURE","pMEDICATION"]
    layout=[
        [
            sg.Frame(
                "Data",[
                [sg.Text("AGE:",font=(TEXTFONT,FONTSIZE)),
sg.Input(key="pAGE",font=(TEXTFONT,FONTSIZE))],
```

```
[sg.Text("GENDER:",font=(TEXTFONT,FONTSIZE)),sg.Input(key="pGENDER",font=(TEXTFONT,F
ONTSIZE))],
                [
                    sg.Text("SYMPTOMS LIST:",font=(TEXTFONT,FONTSIZE)),
                    sg.Input(key="pSYMPTOMS",font=(TEXTFONT,FONTSIZE),size=(30,1)),
                    sg.Text("TIME LIST:",font=(TEXTFONT,FONTSIZE)),
                    sg.Input(key="pTIMES",font=(TEXTFONT,FONTSIZE),size=(30,1))
                    ],
                [sg.Text("TEMPERATURE IN
C:",font=(TEXTFONT,FONTSIZE)),sg.Input(key="pTEMPERATURE",font=(TEXTFONT,FONTSIZE))]
,
                [
                    sg.Text("MEDICATION
GIVEN:",font=(TEXTFONT,FONTSIZE)),sg.Input(key="pMEDICATION",font=(TEXTFONT,FONTSIZE
)),
                    ],
                [sg.Text("SURVIVAL RATE:",font=(TEXTFONT,FONTSIZE)),sg.Text("
",key="pMORTALITY",font=(TEXTFONT,FONTSIZE)),
                sg.Text("   %",font=(TEXTFONT,FONTSIZE))],
                [
                    sg.Button("ESTIMATE"),
                    sg.Text(
                        "Acceptable Deviance:",
                        font=(TEXTFONT,FONTSIZE)),
                    sg.Slider(range=(0,50),
                        default_value=10,
                        key="DEVIANCE",
                        size=(20,15),
                        orientation='horizontal',
                        font=('Helvetica', 12)
                        )
                    ]
                ]),
            ]
        ]

   def __init__(self,csvfile=path.abspath(__file__ + "/../../data.csv")):
```

```
        self.csvmanager=CSVManager(CSVFILE=csvfile)
        expanded_dataset=self.csvmanager.expanded_dataset()

    def predict(self, datadict, allowed_deviation = 5):
        """
        Return a percentage survival rate pertaining to the given data.
        Works by calculating probability of each individiual property of the patient
        and multiplying them together.

        """
        ex_ds = self.csvmanager.expanded_dataset()
        prob_dict = {key:0 for key in datadict.keys()}
        count_dict = {key:0 for key in datadict.keys()}
        for key in datadict:
            for entry in ex_ds:
                if key not in ["GENDER","MORTALITY"] :
                    minim = datadict[key] - allowed_deviation
                    maxim = datadict[key] + allowed_deviation
                    if entry[key] != 0 and minim <= entry[key] <= maxim:
                        count_dict[key]+=1
                        prob_dict[key] +=1 if entry["MORTALITY"] == 1 else 0
                elif key == "MORTALITY":
                    pass
                else:
                    count_dict[key]+=1
                    if entry[key] == datadict[key]:
                        prob_dict[key] += 1
        for key in prob_dict:
            prob_dict[key] = prob_dict[key]/count_dict[key] if prob_dict[key] !=0
else 1
        return (reduce(lambda x, y: x * y, prob_dict.values() )*100)

if __name__=="__main__":
    predictor=Predictor()
    csvmanager=CSVManager(CSVFILE=path.abspath(__file__ + "/../../data.csv"))
    print(
        "Survival Rate: ",
        predictor.predict(
```

```
            (
                {
                    key:value for key,value in
csvmanager.expanded_dataset()[1].items() if key != "MORTALITY"}
                )
            )
    )
```

./managers/bargraphmanager.py

```
"""
    ./managers/bargraphmanager.py

    This File contains the object BarGraphManager, which handles all Bar Graph
    related operations that may arise during the use of the main program application.

    This file may be run on its own to test the features of BarGraphManager
separately.
"""

import tkinter as Tk

import matplotlib as mpl
import numpy as np
import PySimpleGUI as sg
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

if __name__ == "__main__":
    from _config import get_settings_config
    from csvmanager import CSVManager
else:
    from ._config import get_settings_config
    from .csvmanager import CSVManager


class BarGraphManager:
    def __init__(self):
```

```
        csvmanager = CSVManager()
        dataset = csvmanager.list_od_from_csv()
        self.ages = []
        self.dataset = dataset
        for entry in dataset:
            self.ages.append(int(entry["AGE"]))
        self.fig = self.bar_graph_age_vs_Case()
        self.fig1 = self.bar_graph_case_gender_wise()
        self.mvf_fig = self.bar_graph_m_vs_f()
        self.layout = [
            [
                sg.TabGroup(
                    [
                        [
                            sg.Tab(
                                "Age vs Cases",
                                self.bar_graph_age_vs_Case,
                                element_justification="center",
                                key="age-vs-case",
                            ),
                            sg.Tab(
                                "Age vs Cases by Gender",
                                self.layout_bar_graph_case_gender_wise,
                                element_justification="center",
                                key="case-vs-gender",
                            ),
                            sg.Tab(
                                "Gender vs Cases",
                                self.layout_bar_graph_mvf,
                                element_justification="center",
                                key="male-vs-female",
                            ),
                        ]
                    ],
                    enable_events=True,
                    key="bargraph_tab",
                )
            ]
```

```
        ]

    def draw_figure(self, canvas, figure, loc=(0, 0)):
        """Draw a matplotlib figure onto a Tk canvas

        loc: location of top-left corner of figure on canvas in pixels.

        """
        canvas.pack()
        figure_canvas_agg = FigureCanvasTkAgg(figure, master=canvas)
        figure_canvas_agg.draw()
        _, _, figure_w, figure_h = figure.bbox.bounds
        # _ can be used in places of unused variable while unpacking
        figure_w, figure_h = int(figure_w), int(figure_h)
        photo = Tk.PhotoImage(master=canvas, width=figure_w, height=figure_h)
        figure_canvas_agg._tkphoto = photo
        canvas.create_image(loc[0] + figure_w / 2,
                            loc[1] + figure_h / 2, image=photo)
        figure_canvas_agg.blit()
        # Contains a reference to the photo object
        # which must be kept live or else the picture disappears
        return photo

    def bar_graph_age_vs_Case(self):
        v1, v2, v3, v4, v5 = 0, 0, 0, 0, 0
        for i in range(len(self.ages)):
            a = self.ages[i]
            if a >= 1 and a <= 20:
                v1 += 1
            elif a >= 21 and a <= 40:
                v2 += 1
            elif a >= 41 and a <= 60:
                v3 += 1
            elif a >= 61 and a <= 80:
                v4 += 1
            elif a >= 81:
                v5 += 1
        values_to_plot = (v1, v2, v3, v4, v5)
```

```
        ind = np.arange(len(values_to_plot))
        width = 0.5
        fig = mpl.figure.Figure()
        subplt = fig.add_subplot(1, 1, 1)
        p1 = subplt.bar(ind, values_to_plot, width)
        subplt.set_title(
            "Age vs Cases", fontdict={"fontsize": get_settings_config()["fontsize"]}
        )
        subplt.set_ylabel("NO.OF.CASES")
        subplt.set_xlabel("AGE")
        subplt.set_xticks(ind)
        subplt.set_xticklabels(("1-20", "21-40", "41-60", "61-80", "81-100"))
        subplt.set_yticks(np.arange(0, 31, 5))
        subplt.legend((p1[0],), ("AGE LIMIT",))
        fig.align_labels(subplt)
        figure_x, figure_y, figure_w, figure_h = fig.bbox.bounds
        self.bar_graph_age_vs_Case = [
            [sg.Canvas(size=(figure_w, figure_h), key="-CANVAS-")],
        ]
        return fig

    def bar_graph_case_gender_wise(self):
        dataset = self.dataset
        m1, m2, m3, m4, m5 = 0, 0, 0, 0, 0
        f1, f2, f3, f4, f5 = 0, 0, 0, 0, 0
        for i in range(len(dataset)):
            if dataset[i]["GENDER"] == "MALE":
                a = int(dataset[i]["AGE"])
                if a >= 1 and a <= 20:
                    m1 += 1
                elif a >= 21 and a <= 40:
                    m2 += 1
                elif a >= 41 and a <= 60:
                    m3 += 1
                elif a >= 61 and a <= 80:
                    m4 += 1
                elif a >= 81:
                    m5 += 1
```

```
            else:
                a = int(dataset[i]["AGE"])
                if a >= 1 and a <= 20:
                    f1 += 1
                elif a >= 21 and a <= 40:
                    f2 += 1
                elif a >= 41 and a <= 60:
                    f3 += 1
                elif a >= 61 and a <= 80:
                    f4 += 1
                elif a >= 81:
                    f5 += 1
        barWidth = 0.25
        bars1 = [m1, m2, m3, m4, m5]
        bars2 = [f1, f2, f3, f4, f5]
        r1 = np.arange(len(bars1))
        r2 = [x + barWidth for x in r1]
        fig = mpl.figure.Figure()
        plt = fig.add_subplot(1, 1, 1)
        p1 = plt.bar(
            r1, bars1, color="#7f6d5f", width=barWidth, edgecolor="white",
label="MALE"
        )
        p2 = plt.bar(
            r2,
            bars2,
            color="#557f2d",
            width=barWidth,
            edgecolor="white",
            label="FEMALE",
        )
        plt.set_title(
            "Age vs Cases by Gender", fontdict={"fontsize":
get_settings_config()["fontsize"]}
        )
        plt.set_xlabel("AGE")
        plt.set_xticks([r + barWidth for r in range(len(bars1))])
        plt.set_xticklabels(["1-20", "21-40", "41-60", "61-80", "81-100"])
```

```
        plt.legend([p1[0], p2[0]], ("Males", "Female"))

        figure_x, figure_y, figure_w, figure_h = fig.bbox.bounds

        self.layout_bar_graph_case_gender_wise = [
            [sg.Canvas(size=(figure_w, figure_h), key="-GENDER_CANVAS-")],
        ]
        return fig

    def bar_graph_m_vs_f(self):
        list1 = self.dataset
        m, f = 0, 0
        for i in range(len(list1)):
            a = list1[i]["GENDER"]
            if a == "MALE":
                m += 1
            else:
                f += 1
        values_to_plot = (m, f)
        ind = np.arange(len(values_to_plot))
        width = 0.4
        fig = mpl.figure.Figure()
        plt = fig.add_subplot(1, 1, 1)

        p1 = plt.bar(ind, values_to_plot, width)

        plt.set_ylabel("NO.OF.CASES")
        plt.set_title("MALE vs FEMALE")
        plt.set_xticks(ind)
        plt.set_xticklabels(("MALE", "FEMALE"))
        plt.set_yticks(np.arange(0, len(list1) + 1, 5))

        figure_x, figure_y, figure_w, figure_h = fig.bbox.bounds

        self.layout_bar_graph_mvf = [
            [sg.Canvas(size=(figure_w, figure_h), key="-MVF_CANVAS-")],
        ]
        return fig
```

```
if __name__ == "__main__":
    a = BarGraphManager()
    window = sg.Window(
        "Demo Application - Embedding Matplotlib In PySimpleGUI",
        a.layout,
        force_toplevel=True,
        finalize=True,
    )
    fig_photo = a.draw_figure(window["-CANVAS-"].TKCanvas, a.fig)
    event, values = window.read()
    window.close()
```

./managers/thememanager.py

```
"""
    ./managers/thememanager.py

    This File contains the object ThemeManager which handles all functions
    related to setting the theme of the User Interface of the Main program
    application.

    This file may be run on its own to test the features of Predictor separately.
"""

import PySimpleGUI as sg

if __name__ == "__main__":
    from _config import set_settings_config
else:
    from ._config import set_settings_config


class ThemeManager:

    layout = [
```

```
        [sg.Text("List of InBuilt Themes")],
        [sg.Text("Please Choose a Theme  to see Demo window")],
        [sg.Listbox(
                values=[
                    t
                    for t in sg.theme_list()
                    if ( ("dark" in t.lower() or t.lower() in ["reds",
"lightbrown12", "black"])
                    and not t.lower().startswith("green") )
                ],
                size=(20, 12),
                key="THEMELIST",
                enable_events=True,
            )
        ],
        [sg.Button(
            button_text="Set Theme",
            button_color=("white", "blue"),
            size=(18, 1),
            key="THEMEBTN",
            )
        ],
    ]

    def set_theme(self, theme):
        settings = {}
        settings["theme"] = theme
        set_settings_config(settings)


if __name__ == "__main__":
    thememanager = ThemeManager()

    window = sg.Window("Theme List", thememanager.layout)

    # This is an Event Loop
    while True:
        event, values = window.read()
```

```
        if event in (None, "Exit"):
            break

        sg.theme(values["THEMELIST"][0])
        sg.popup_get_text("This is {}".format(values["THEMELIST"][0]))

    # Close
    window.close()
```

./managers/fontmanager.py

```
"""
    ./managers/fontmanager.py

    This File contains the object FontManager, which handles all Font Size related
    operations that may arise during the use of the main program application.

    This file may be run on its own to test the features of FontManager separately.
"""

import PySimpleGUI as sg
if __name__=="__main__":
    from _config  import set_settings_config
    from _config import get_settings_config
else:
    from ._config import set_settings_config
    from ._config import get_settings_config


class FontManager:
    fontSize= int(get_settings_config()["fontsize"])
    layout = [[sg.Spin([sz for sz in range(12, 21)], font=('Helvetica 20'),
initial_value=fontSize,key="FONTSPIN",change_submits=True),
               sg.Text("Aa", size=(2, 1), font="Helvetica "  + str(fontSize),
key="FontPreview")]]
    def set_fontsize(self,fontsize):
```

```
        settings={}
        settings['fontsize']=str(fontsize)
        set_settings_config(settings)

if __name__=="__main__":
    fontmanager=FontManager()
    window = sg.Window("Font size selector", fontmanager.layout, grab_anywhere=False)
    # Event Loop
    while True:
        event, values= window.read()
        if event == sg.WIN_CLOSED:
             break
        else:
               print(event)
        sz_spin = int(values['FONTSPIN'])
        fontSize = sz_spin
        font = "Helvetica "  + str(fontSize)
        window["FontPreview"].update(font=font)
        window['FONTSPIN'].update(sz_spin)
```

./managers/_config.py

```
"""
   ./managers/_config.py

   Functions for accessing the configuration stored in medict.cfg.
   The user can edit the details about SQL and settings in the
   config file, as in, medict.cfg and the stored value are
   parsed using the configparser module.

   medict.cfg should follow ini like setting.
   See
https://docs.python.org/3/library/configparser.html?highlight=configparser#supported
-ini-file-structure
   for how to actually set it up.
"""
```

```
import os
import configparser
from pathlib import Path


cfg_file = Path(__file__).parent.parent / "medict.cfg"
config = configparser.ConfigParser()
config.read(cfg_file)


def get_sql_config():
    """Returns the Configuration of ``SQL`` section
    These configuration it returns can be accessed as
    dictionaries.
    """
    return config["sql"]


def get_settings_config():
    """Returns the Configuration of ''settings'' section
    These configuration it returns can be accessed as
    dictionaries.
    """
    return config["settings"]


def set_settings_config(d):
    if "theme" in d:
        config["settings"]["theme"] = d["theme"]
    if "fontsize" in d:
        print(d["fontsize"])
        config["settings"]["fontsize"] = d["fontsize"]
    with open(cfg_file, "w") as configfile:
        config.write(configfile)
```

./medict.cfg

```
[sql]
host = localhost
username = root
password = password21
database = hospital
table_name = patients

[settings]
theme = DarkTanBlue
fontsize = 15
```

Future Enhancements:

1. Allow for usage of custom CSV File.
2. Implementation of a “Backup and Restore” feature.
3. Implementation of Database to CSV data transmission.
4. Implementation of online/cloud storage of data.
5. Implementation of keyboard shortcuts.
6. Automatic theme changer based on Operating System preferences.
7. Automated Application Update System.
8. Input sanitisation and input validation.

Bibliography and References:

Sites Referred for Basic Documentation:

1. https://docs.python.org/3/library/csv.html
   (Official Python3 Documentation for `stdlib csv` module.)
2. https://pysimplegui.readthedocs.io/en/latest/
   (Official PySimpleGUI documentation.)
3. https://matplotlib.org/contents.html
   (Official Matplotlib documentation)
4. https://numpy.org/doc/stable/
   (Official Numpy Documentation)

Sites Referred For Patient Data:

https://www.sirm.org/en/category/articles/covid-19-database/

(Italian Society of Radiology COVID-19 Database)

Books Referred:

1. Computer Science with Python - Textbook for Class 11
   (Sumita Arora, 2019)

2. Computer Science with Python - Textbook for Class 12 (Sumita Arora, 2020)

The Guidance of our Teacher Dr.H Vidhya was also incredibly useful.