DIT: A Primer To Bayesian Networks Model Testing

Yoshiki Shoji

June 13, 2017

Acknowledgements

The idea for this research was mediated through the supervision of Professor Damon Barry and Doctor Matteo Zallio whom both work at Dublin Institute of Technology, DIT.

Abstract

The research proposes to apply Bayesian networking to make web services more usable for users with different types of impairments, such as aging citizens or those with intellectual disabilities. In this particular use case of Bayesian Networks, the research proposes to develop a system that is similar to "IF THIS THEN THAT" (IFTTT), that focuses specially on events that indicate that normal household activities are underway (particularly through monitoring usage of electrical appliances), but instead of acting on a single unusual pattern of activity as in the case of IFTTT it works statistically on a number of such activities.

Project Elements: 3x wemo smart plugs and 1 monitor, ouimeaux open source system, electrical appliance usage, Python programming language, BayesPy library, Python social network libraries, MySQL

Introduction: *A Primer To Bayesian Network*

The research in the long run hopes to develop the necessary Bayesian Network to find peculiar events that may indicate the different types of impairments or intellectual disabilities such as dementia. This paper however, will present a testing model specifically using a common houseware of a kettle using the Smart Wemo Switch along with MySQL database and Python(v.3.5) code used to generate such data.

There will be two datas presented. One which will show how much time elapsed to boil water with different given volumes. The procedures taken in gathering data will be explained on *Page* 2. The other one will show how much time elapsed to boil the water, however, with another procedure which will be explained on *Page* 3.

Experiment 1A: Plastic Kettle

Objective
The first experiment seeks to test the relationship between how much time elapses every time the plastic kettle is cooled down with different volumes of water. In order to accelerate the process, water will be replaced every time with each run. The data will then be plotted in order to find the "best-fit" function.

Experimental Requirements

- Smart Wemo Switch
- mysql_table.py, mysql_data Python Files
- MySQL database named 'DATA'
- Plastic Kettle

Procedure
Step 1: Pour 0.5L of Water in the Kettle

Step 2: Execute the mysql_data file (see Appendix B) which will retrieve the readings from the Wemo Switch and store the values inside the table, 'Insight(number)' within the DATA database (see Appendix A). If an exception error occurs which indicates there are no tables created, run the mysql_table file which will create all the necessary tables.

Step 3: When the kettle stops boiling, execute the command, 'select * from Insight(number)' within MySQL using database, DATA, and retrieve the total time the kettle was boiling using the column, 'onfor'. This column stores how much time has elapsed starting from the moment the appliance was turned on. Execute the same MySQL command again until the 'currentpower' column reaches zero and terminate the code. This indicates there is no more power flowing through the switch.

Step 4: Record the Data indicating the volume of water used, the time and the run number.

Step 5: Replace the boiling water with new cold water. However, rince the kettle a few times in order for the internal mechanisms to reach room temperature.

Step 6: Repeat *Step* 3 four more times, with a total of five runs at the end. Note that whenever a new data is stored, 'totalon' will by default start at zero second once again.

Step 7: Once the data for 0.5L is done, repeat all the steps above for volumes 1.0L and 1.5L. At the very end, a total of 15 data measurements should be produced.

EXP1A: Tables

The tables below show the recorded data for the boiling time using 0.5L,1.0L and 1.5L volumes of water respectively. The last row gives the average boiling time of all the runs.

Table 1: Boiling Time With 0.5L of Water

Run	Time [s]
1	114
2	116
3	116
4	115
5	188
Average Time: 115.8 seconds	

Table 2: Boiling Time With 1.0L of Water

Run	Time [s]
1	212
2	203
3	202
4	205
5	202
Average Time: 204.8 seconds	

Table 3: Boiling Time With 1.5L of Water

Run	Time [s]
1	294
2	286
3	289
4	292
5	290
Average Time: 290.2 seconds	

Data Analysis

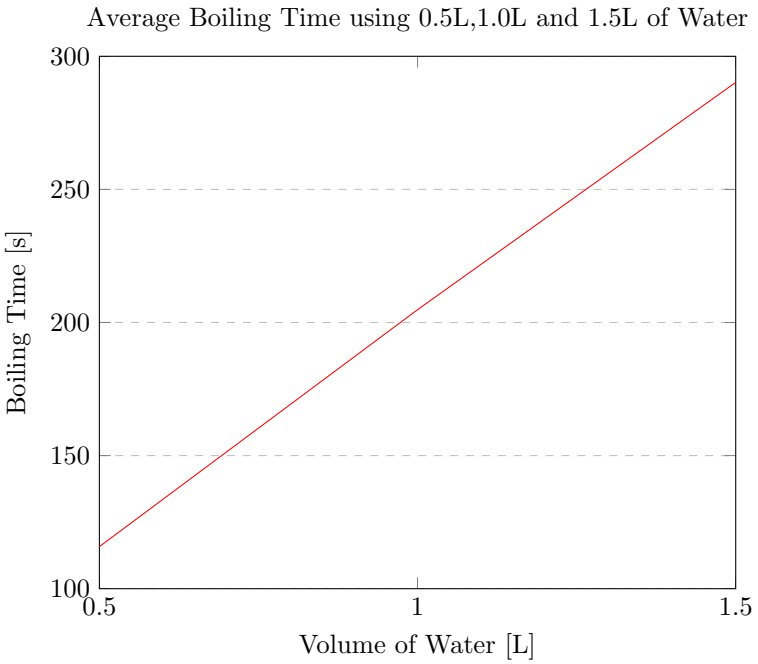

Figure 1

The figure above shows the linear relation between how much time elapses to boil 0.5L,1.0L and 1.5L of water. This is to be expected by considering the heat equation:

$$Q = mc\Delta T$$

where the amount of energy it takes to heat/cool a certain amount of substance is proportional to its mass, specific heat capacity and the amount of temperature change. We also keep in mind the kettle $turns\ off$ once it reaches a certain temperature and further note the initial temperature between each run will be roughly the same due the procedures taken. This allows $\Delta T, and\ c$ to be regarded as constants. As we desire a relation between the average boiling time and the mass, we have:

$$Pt = mc\Delta T$$

$$t(m) = \frac{K}{P}m$$

where $K = c\Delta T$. We will then make easier estimations to the equation, using average power as this will not result in huge errors, as the trends seen between power vs. time can be seen on *Page* 8. Hence, we have an equation which shows the linear relation between the boiling time and the volume of water inside the kettle as:

$$t(m) = \frac{K}{P_{avg}}m$$

Note: using the density of water, $\rho = 1kg/L$, we are able to find the mass.

Actual Plot Of Power vs. Time Using Python

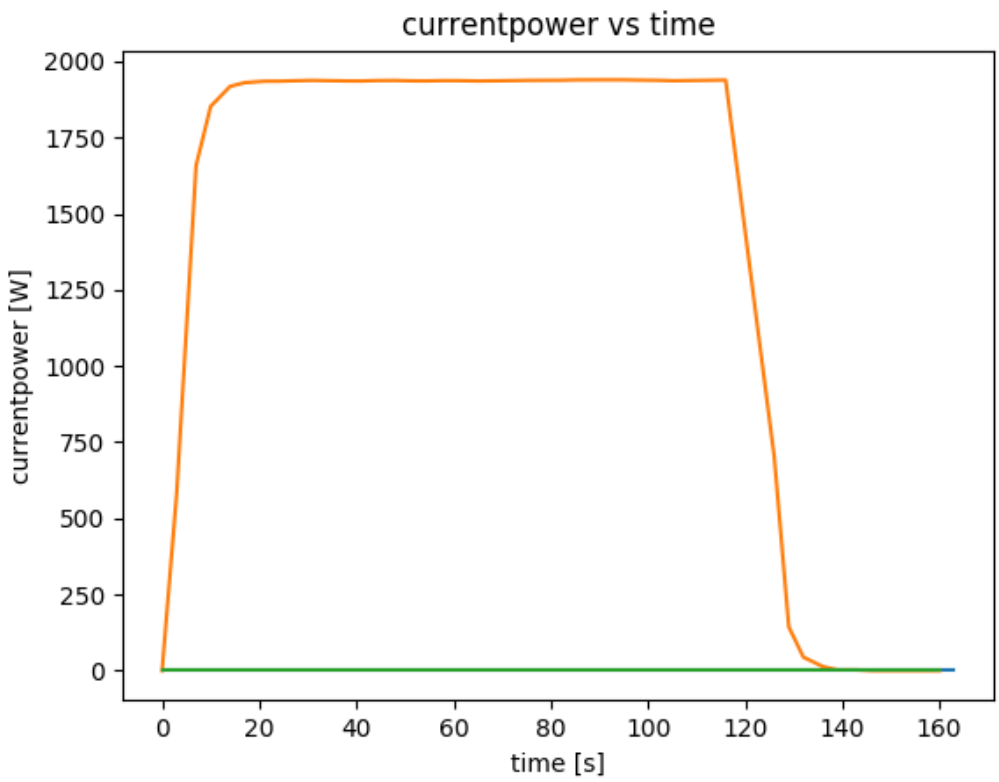

Figure 2: Graph 1: Run1 with 0.5L of Water

The graph above shows one run of generating a plot using Python with 0.5L of Water (Note: Only one run is shown as the others show a similar trend). The orange plot is for the kettle, whereas the other plots are Wemo switches without any appliance attached, hence generating 0W of power.

Observing the graph above, there are three stages when the water is boiling. When the kettle is turned on, the power rises until it stays relatively constant. Once the water is boiled, then the power is drained until no more current is flowing to the kettle.

Experiment 1B: Metal Kettle

Objective

The second part to the first experiment seeks to test the relationship between how much time elapses every time the metal kettle is cooled down with different volumes of water. In order to accelerate the process, water will be replaced every time with each run. The data will then be plotted in order to find the "best-fit" function.

Experimental Requirements

- Smart Wemo Switch
- mysql_table.py, mysql_data Python Files
- MySQL database named 'DATA'

- Metal Kettle

Procedure

Step 1: Pour 0.5L of Water in the Kettle

Step 2: Execute the mysql_data file (see Appendix B) which will retrieve the readings from the Wemo Switch and store the values inside the table, 'Insight(number)' within the DATA database (see Appendix A). If an exception error occurs which indicates there are no tables created, run the mysql_table file which will create all the necessary tables.

Step 3: When the kettle stops boiling, execute the command, 'select * from Insight(number)' within MySQL using database, DATA, and retrieve the total time the kettle was boiling using the column, 'onfor'. This column stores how much time has elapsed starting from the moment the appliance was turned on. Execute the same MySQL command again until the 'currentpower' column reaches zero and terminate the code. This indicates there is no more power flowing through the switch.

Step 4: Record the Data indicating the volume of water used, the time and the run number.

Step 5: Replace the boiling water with new cold water. However, rince the kettle a few times in order for the internal mechanisms to reach room temperature.

Step 6: Repeat *Step* 3 four more times, with a total of five runs at the end. Note that whenever a new data is stored, 'totalon' will by default start at zero second once again.

Step 7: Once the data for 0.5L is done, repeat all the steps above for volumes 1.0L and 1.5L. At the very end, a total of 15 data measurements should be produced.

EXP1B: Tables

The tables below show the recorded data for the boiling time using 0.5L,1.0L and 1.5L volumes of water respectively. The last row gives the average boiling time of all the runs.

Table 4: Boiling Time With 0.5L of Water

Run	Time [s]
1	202
2	201
3	201
4	200
5	200
Average Time: 201.2 seconds	

Table 5: Boiling Time With 1.0L of Water

Run	Time [s]
1	358
2	345
3	340
4	348
5	350
Average Time: 348.2 seconds	

Table 6: Boiling Time With 1.5L of Water

Run	Time [s]
1	447
2	440
3	448
4	440
5	445
Average Time: 450.0 seconds	

Data Analysis

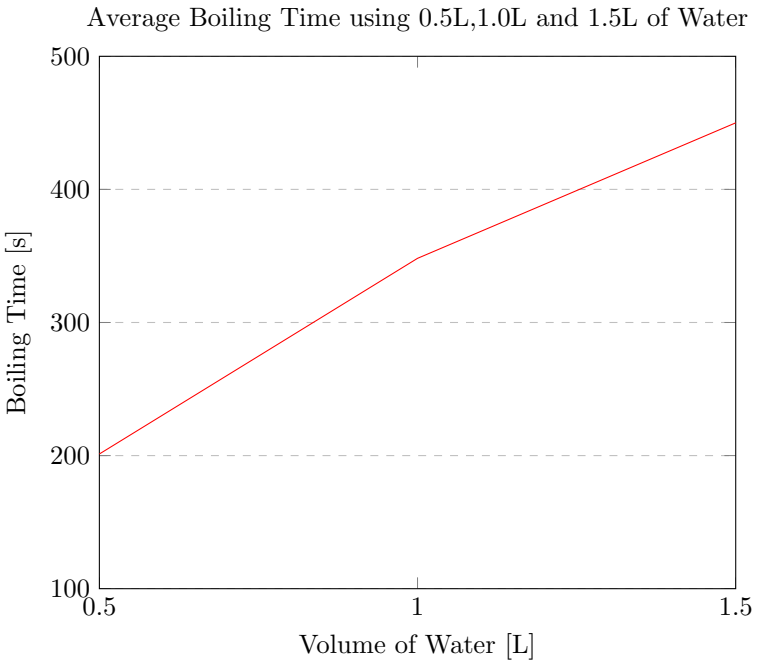

Figure 3

In comparing the graph above with Experiment 1A using the plastic kettle, we do not see a perfect linear fit. However, we note that the boiling time for the metal kettle is higher hence making the interior thermostat ending with a higher temperature. If the same procedures are taken as the plastic one, it would leave this thermostat fairly hot, giving a less average boiling time as seen above than what is theoretically expected. Hence, with the modification of including a step to make the experimenter wait for a couple of minutes and collecting data, a linear plot should be seen.

Experiment 2A

Objective
The second experiment seeks to test the relationship between how much time elapses every time the plastic kettle is cooled down with different volumes of water *without replacing the water* for each run. Furthermore, the experimenter will wait 2 minutes between each run immediately after the kettle stops boiling.

Experimental Requirements

- Stop watch
- Smart Wemo Switch
- mysql_table.py, mysql_data Python Files
- MySQL database named 'DATA'
- Plastic Kettle

Procedure
Step 1: Pour 0.5L of Water in the Kettle

Step 2: Execute the mysql_data file (see Appendix B) which will retrieve the readings from the Wemo Switch and store the values inside the table, 'Insight(number)' within the DATA database (see Appendix A). If an exception error occurs which indicates there are no tables created, run the mysql_table file which will create all the necessary tables.

Step 3: When the kettle stops boiling, execute the command, 'select * from Insight(number)' within MySQL using database, DATA, and retrieve the total time the kettle was boiling using the column, 'onfor'. This column stores how much time has elapsed starting from the moment the appliance was turned on. Execute the same MySQL command again until the 'currentpower' column reaches zero and keep the Python code, 'mysql_data' running.

Step 4: Start the stop watch and wait until 2 minutes elapses. In the meantime, record the Data indicating the volume of water used, the time and the run number.

Step 5: Restart the stop watch and start the kettle once again.

Step 6: Repeat *Step* $3 - $ *Step* 4 four more times, with a total of five runs at the end. Note that whenever a new data is stored, 'onfor' will by default start at zero second once again.

Step 7: Once the data for 0.5L is done, repeat all the steps above for volumes 1.0L and 1.5L. At the very end, a total of 15 data measurements should be produced.

EXP2A: Tables

The tables below show the recorded data for the boiling time using 0.5L,1.0L and 1.5L volumes of water respectively. The last row gives the average boiling time of all the runs.

Table 7: Boiling Time With 0.5L of Water

Run	Time [s]
1	115
2	30
3	28
4	26
5	26
Average Time: 43 seconds	

Table 8: Boiling Time With 1.0L of Water

Run	Time [s]
1	194
2	29
3	28
4	26
5	26
Average Time: 62.2 seconds	

Table 9: Boiling Time With 1.5L of Water

Run	Time [s]
1	275
2	31
3	30
4	29
5	30
Average Time: 79 seconds	

Data Analysis

The graphs above show the boiling time for each volumes of water. We note that after the first run, the time it takes to boil all stay relatively at $30s$ following the 2 minute time waiting time between each run.

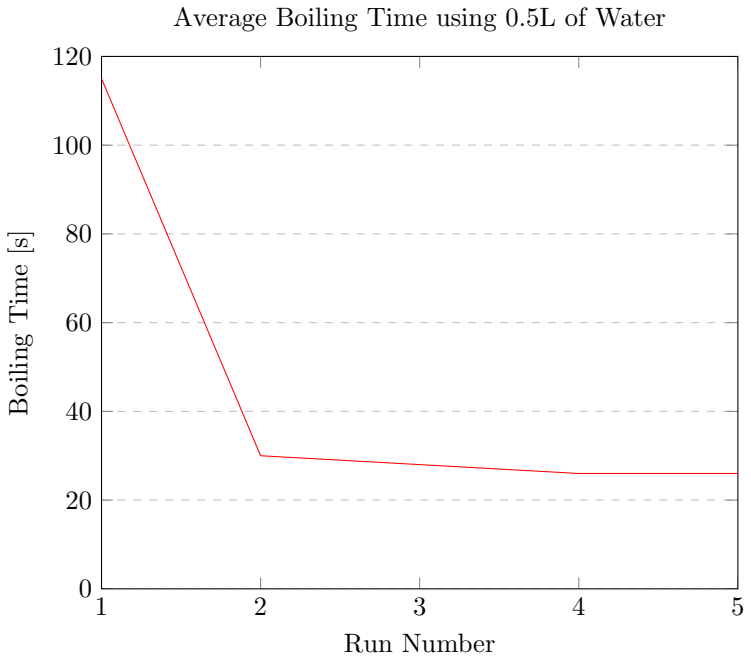

Figure 4

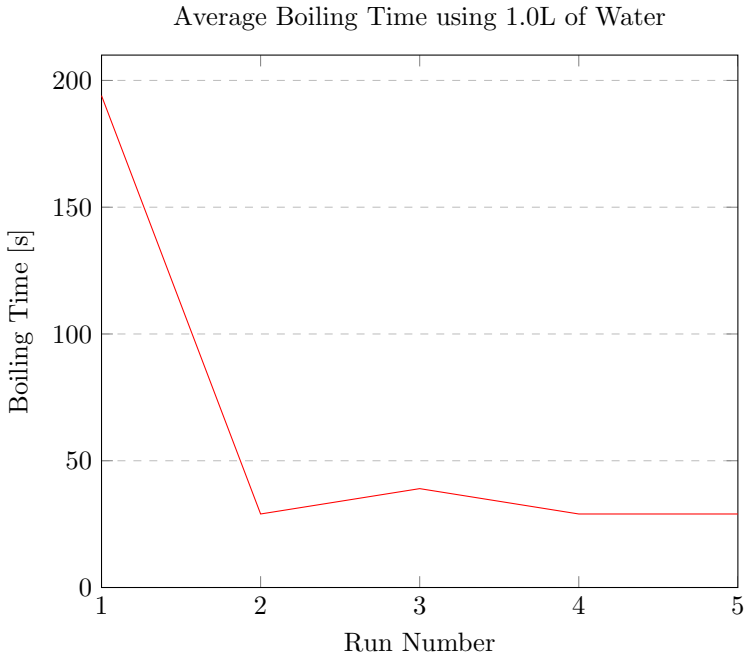

Figure 5

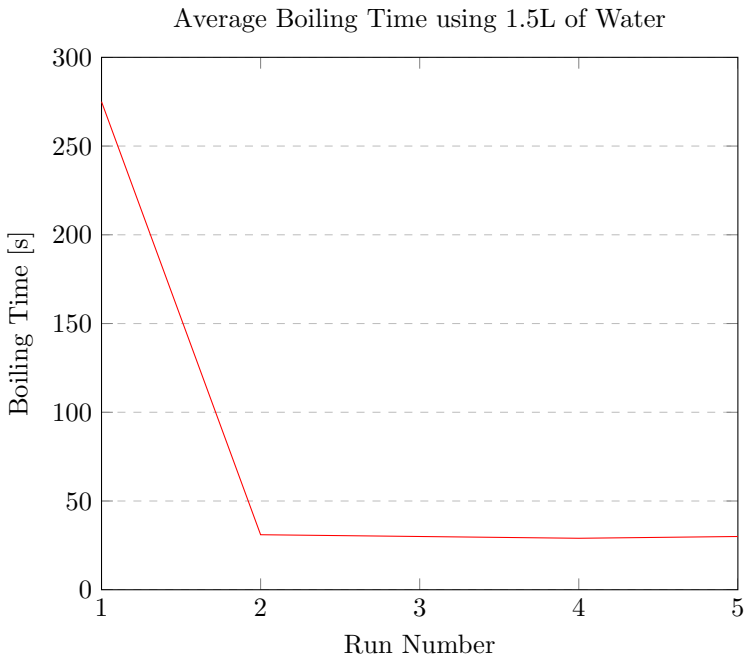
Average Boiling Time using 1.5L of Water
Boiling Time [s]
300
250
200
150
100
50
0
1
2
3
4
5
Run Number

Figure 6

Actual Plot Of Power vs. Time Using Python

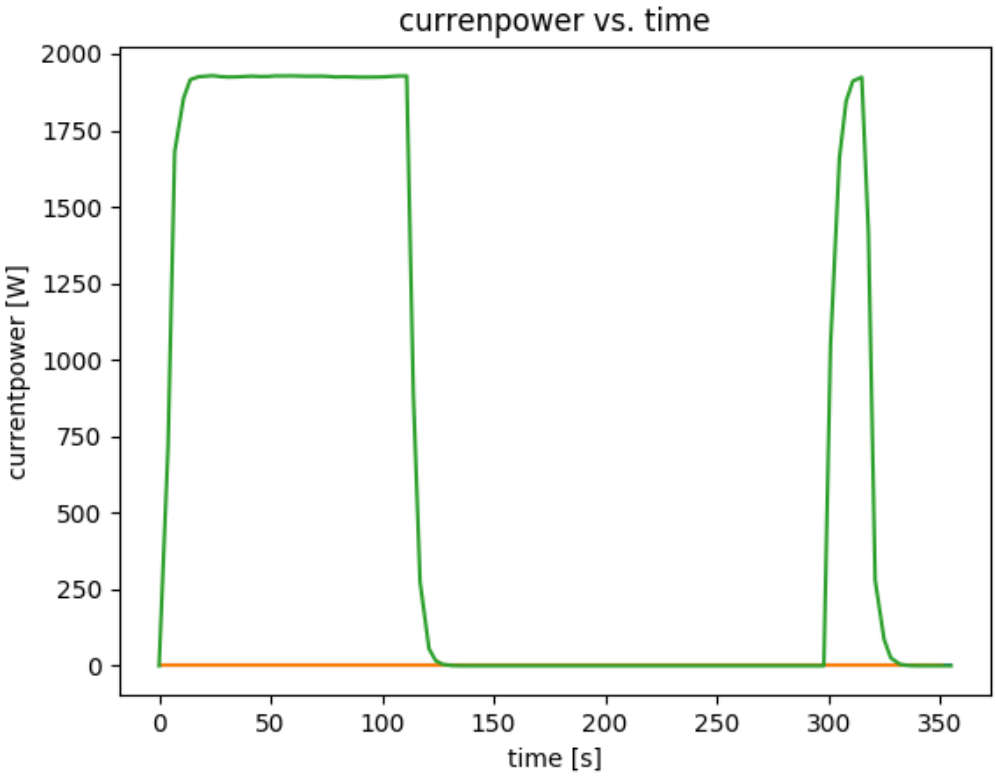

Figure 7: Graph 1: Run1 with 0.5L of Water

The graph above shows the actual plot of power consumed by the kettle with respect to time. The first trend is the same as the one also seen by Experiment 1, which is to be expected. After a waiting time of 2 minutes (ie. the gap between the two spikes) we note the amount of power, hence the energy, consumed by the kettle drops. This is due to the fact the water did not settle to the same initial temperature as before.

Experiment 2B

Objective
The second experiment seeks to test the relationship between how much time elapses every time the metal kettle is cooled down with different volumes of water *without replacing the water* for each run. Furthermore, the experimenter will wait 2 minutes between each run immediately after the kettle stops boiling.

Experimental Requirements

- Stop watch
- Smart Wemo Switch
- mysql_table.py, mysql_data Python Files
- MySQL database named 'DATA'
- Metal Kettle

Procedure
Step 1: Pour 0.5L of Water in the Kettle

Step 2: Execute the mysql_data file (see Appendix B) which will retrieve the readings from the Wemo Switch and store the values inside the table, 'Insight(number)' within the DATA database (see Appendix A). If an exception error occurs which indicates there are no tables created, run the mysql_table file which will create all the necessary tables.

Step 3: When the kettle stops boiling, execute the command, 'select * from Insight(number)' within MySQL using database, DATA, and retrieve the total time the kettle was boiling using the column, 'onfor'. This column stores how much time has elapsed starting from the moment the appliance was turned on. Execute the same MySQL command again until the 'currentpower' column reaches zero and keep the Python code, 'mysql_data' running.

Step 4: Start the stop watch and wait until 2 minutes elapses. In the meantime, record the Data indicating the volume of water used, the time and the run number.

Step 5: Restart the stop watch and start the kettle once again.

Step 6: Repeat *Step* $3 - $ *Step* 4 four more times, with a total of five runs at the end. Note that whenever a new data is stored, 'onfor' will by default start at zero second once again.

Step 7: Once the data for 0.5L is done, repeat all the steps above for volumes 1.0L and 1.5L. At the very end, a total of 15 data measurements should be produced.

EXP2B: Tables

The tables below show the recorded data for the boiling time using 0.5L,1.0L and 1.5L volumes of water respectively. The last row gives the average boiling time of all the runs.

Table 10: Boiling Time With 0.5L of Water

Run	Time [s]
1	180
2	31
3	30
4	30
5	29
Average Time: 56 seconds	

Table 11: Boiling Time With 1.0L of Water

Run	Time [s]
1	330
2	29
3	30
4	31
5	29
Average Time: 89.8 seconds	

Table 12: Boiling Time With 1.5L of Water

Run	Time [s]
1	447
2	31
3	30
4	29
5	30
Average Time: 113.2 seconds	

Data Analysis

The graphs above show the boiling time for each volumes of water. We note that after the first run, the time it takes to boil all stay relatively at $30s$ following the 2 minute time waiting time between each run.

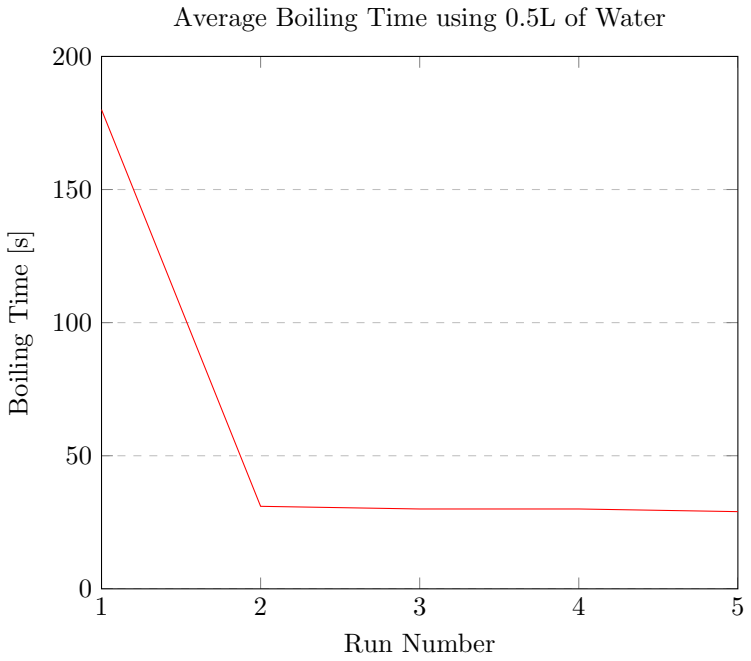

Figure 8

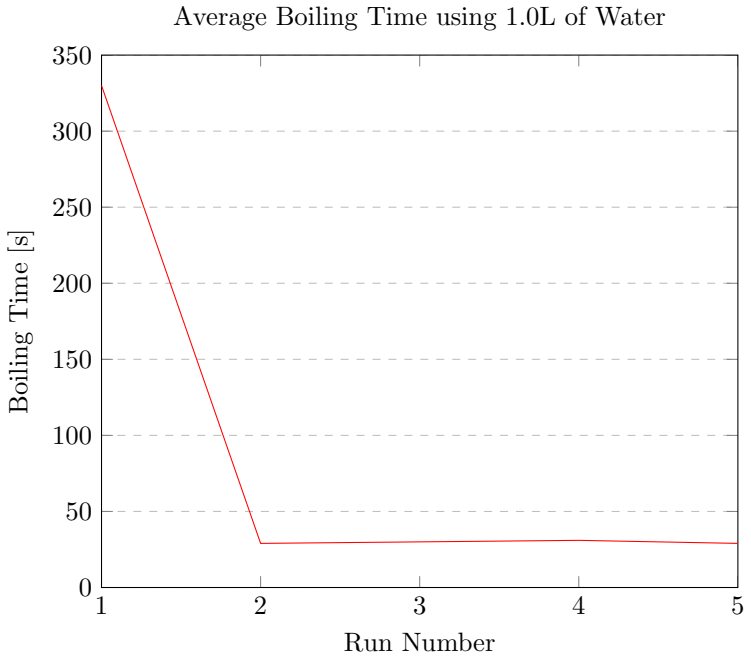

Figure 9

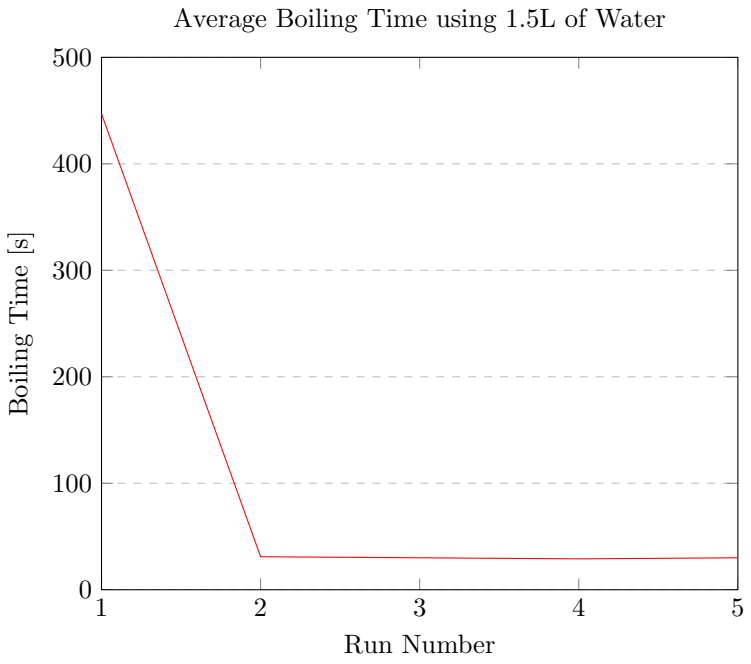
Average Boiling Time using 1.5L of Water
Boiling Time [s]
500
400
300
200
100
0
1
2
3
4
5
Run Number

Figure 10

Actual Plot Of Power vs. Time Using Python

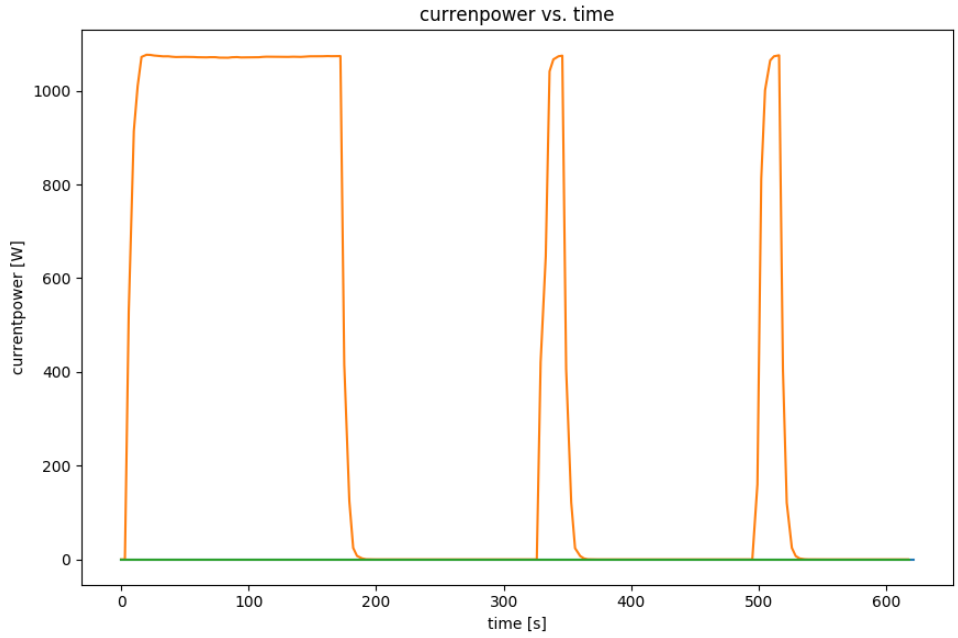

Figure 11: Graph 1: Run1 with 0.5L of Water

The graph above shows the actual plot of power consumed by the kettle with respect to time. The first trend is the same as the one also seen by Experiment 1, which is to be expected. After a waiting time of 2 minutes (ie. the gap between the two spikes) we note the amount of power, hence the energy, consumed by the kettle drops. This is due to the fact the water did not settle to the same initial temperature as before. We further note that this trend is the same as the plastic kettle.

Experiment 3A

Objective
The second part to the second experiment seeks to test the relationship between how much time elapses every time the plastic kettle is cooled down with different volumes of water. It is an extension to Experiment 2A which instead observes boiling time between different waiting intervals for each run. After each run, the water is replaced.

Experimental Requirements

- Stop watch
- Smart Wemo Switch
- mysql_table.py, mysql_data Python Files
- MySQL database named 'DATA'
- Plastic Kettle

Procedure
Step 1: Pour 0.5L of Water in the Kettle

Step 2: Execute the mysql_data file (see Appendix B) which will retrieve the readings from the Wemo Switch and store the values inside the table, 'Insight(number)' within the DATA database (see Appendix A). If an exception error occurs which indicates there are no tables created, run the mysql_table file which will create all the necessary tables.

Step 3: When the kettle stops boiling, execute the command, 'select * from Insight(number)' within MySQL using database, DATA, and retrieve the total time the kettle was boiling using the column, 'onfor'. This column stores how much time has elapsed starting from the moment the appliance was turned on. Execute the same MySQL command again until the 'currentpower' column reaches zero and keep the Python code, 'mysql_data' running.

Step 4: Start the stop watch and wait until 2 minutes elapses. In the meantime, record the Data indicating the volume of water used, the time and the run number.

Step 5: Restart the stop watch and replace the water.

Step 6: Repeat *Step* $3 - $ *Step* 5 with waiting intervals of 4min, 6min, 8min and 10min. This will give a total of five runs at the end. Note that whenever a new data is stored, 'onfor' will by default start at zero second once again.

Step 7: Once the data for 0.5L is done, repeat all the steps above for volumes 1.0L and 1.5L. At the very end, a total of 15 data measurements should be produced.

EXP3A: Tables

The tables below show the recorded data for the boiling time using 0.5L,1.0L and 1.5L volumes of water respectively.

Table 13: Boiling Time With 0.5L

Run	Waiting Time [min]	Time [s]
1	2	29
2	4	37
3	6	39
4	8	44
5	10	49

Table 14: Boiling Time With 1.0L

Run	Waiting Time [min]	Time [s]
1	2	32
2	4	46
3	6	46
4	8	51
5	10	57

Table 15: Boiling Time With 1.5L

Run	Waiting Time [min]	Time [s]
1	2	29
2	4	37
3	6	53
4	8	58
5	10	61

Data Analysis

The graphs above show the boiling time for each volumes of water using the different waiting time intervals. Note that the water is replaced every time for each run.

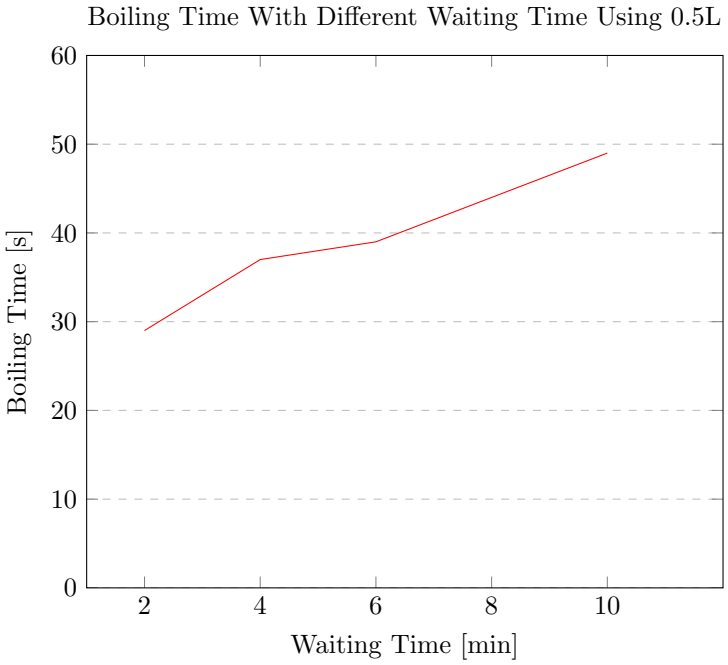

Figure 12

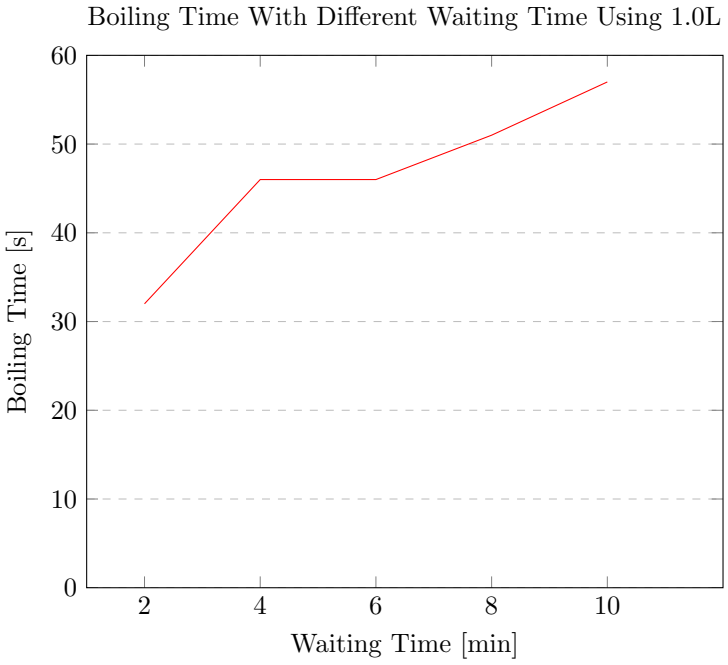

Figure 13

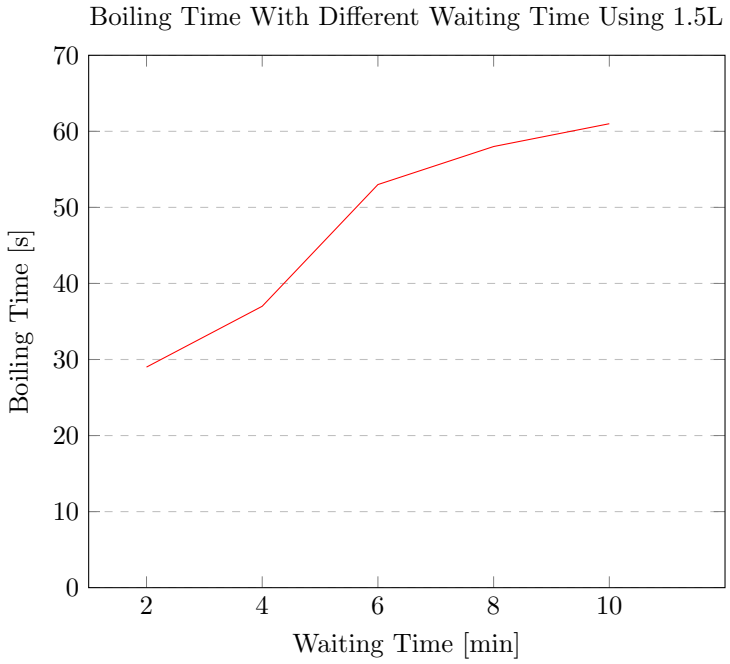
Boiling Time With Different Waiting Time Using 1.5L
Boiling Time [s]
70
60
50
40
30
20
10
0
2
4
6
8
10
Waiting Time [min]

Figure 14

Experiment 3B

Objective
The second part to the second experiment seeks to test the relationship between how much time elapses every time the plastic kettle is cooled down with different volumes of water. The experiment is the same as 3A, however, using a Metal Kettle instead

Experimental Requirements

- Stop watch
- Smart Wemo Switch
- mysql_table.py, mysql_data Python Files
- MySQL database named 'DATA'
- Metal Kettle

Procedure
Follow the same steps as ***EXP3A***

EXP3A: Tables

The tables below show the recorded data for the boiling time using 0.5L,1.0L and 1.5L volumes of water respectively.

Table 16: Boiling Time With 0.5L

Run	Waiting Time [min]	Time [s]
1	2	29
2	4	35
3	6	40
4	8	47
5	10	52

Table 17: Boiling Time With 1.0L

Run	Waiting Time [min]	Time [s]
1	2	30
2	4	37
3	6	48
4	8	52
5	10	58

Data Analysis

The graphs above show the boiling time for each volumes of water using the different waiting time intervals. Note that the water is replaced every time for each run.

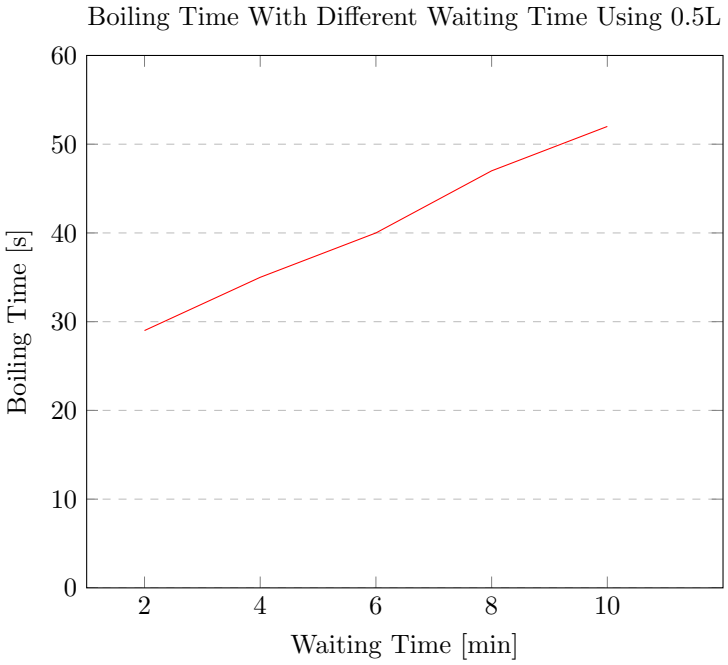

Figure 15

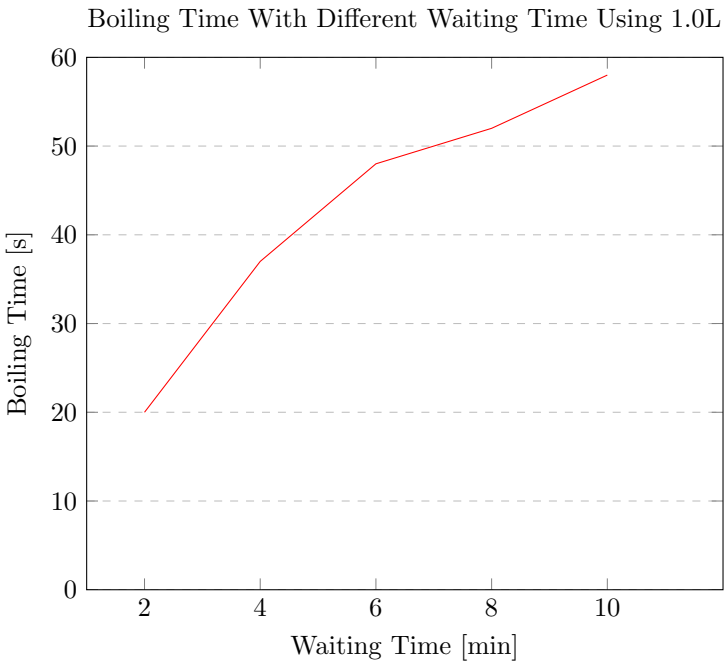

Figure 16

Appendix A

mysql_table.py (Python 3.5)

```
'''
This Python Code will generate the respective tables which are stored
into a MySQL database called, 'DATA'. It will automatically retrieve
all the Wemo Switches which have been named Insight(number), where (number)
indicates the switch number. The most useful table will be from the 'Insight(number)'
table which has the information regarding how much current is drained by the appliance
    ,
the total on time, how much time elapses from the ON and OFF state, and the date
'''

# PYMYSQL MODULES
import pymysql
import pymysql.cursors

# OUIMEAUX MODULES
import ouimeaux
from ouimeaux.environment import Environment
from ouimeaux.signals import receiver, statechange, devicefound

# Initializing the ouimeaux environment
env = Environment()

# Connect to the database
connection = pymysql.connect(host="localhost",
                             user='root',
                             passwd='ditpwd',
                             db="DATA")


def SWITCH(list_switches):
    '''This function intakes the list of switches
    and returns the attributes given by the environment
    NOTE: Returns a Tuple
    '''
    # Obtaining the switch parameters
    switch_name = list_switches
    switch = env.get_switch(switch_name)
    # Note: Parameters are stored in dictionary
    switch_param = switch.insight_params
    # Delete lastchange: Unnecessary Parameter
    switch_param.pop('lastchange')

    return switch_name,switch_param

try:
    # Commit every data as by default it is False
    connection.autocommit(True)

    # Create a cursor object
    cursorObject = connection.cursor()
```

```
    '''ouimeaux notes:

    ### useful attributes to SWITCH ###
        [switch].insight_params: This returns

        'state': state,
        'lastchange': datetime.fromtimestamp(int(lastchange)), <- popped
        'onfor': int(onfor),
        'ontoday': int(ontoday),
        'ontotal': int(ontotal),
        'todaymw': int(float(todaymw)),
        'totalmw': int(float(totalmw)),
        'currentpower': int(float(currentmw))

    ### Turning switch on and off manually ###
        env.get_switch(SWITCH_NAME).on/off()
    '''

    env.start()
    env.discover(3)

    for i in range(len(env.list_switches())):
        '''Iterate through all the switches available
        and create the necessary Tables'''

        # Call the helper function SWITCH:
        switch = SWITCH(list(env.list_switches())[i])
        sqlCreateTableCommand_temp = str()
        for param in list(switch[1].keys()):
            if param == list(switch[1].keys())[-1]:
                sqlCreateTableCommand_temp += param + ' int'
            else:
                sqlCreateTableCommand_temp += param + ' int' + ', '

        # Table commands
        sqlCreateTableCommand = 'CREATE TABLE ' + switch[0] + '('+
            sqlCreateTableCommand_temp + ')'
        cursorObject.execute(sqlCreateTableCommand)

        # Insert TIME into SWITCH_PARAM table
        sqlInsertColCommand = 'ALTER TABLE ' + switch[0] + ' ADD TIME TIME FIRST'
        cursorObject.execute(sqlInsertColCommand)

        # Insert DATE into SWITCH_PARAM table
        sqlInsertColCommand = 'ALTER TABLE ' + switch[0] + ' ADD DATE DATE FIRST'
        cursorObject.execute(sqlInsertColCommand)

        # Create table DATA
        sqlCreateTableCommand = ("CREATE TABLE "
                                 "DATA_" + str(switch[0]) + " (DATE DATE, TIME TIME,"
                                 "STATE varchar(20))")
        cursorObject.execute(sqlCreateTableCommand)
```

```
        # Create table which stores index values (explained in mysql_plot)
        '''Everytime we re-read the information, we do not want to plot the same
        data we have plotted previously. In order to prevent this from happening, we
        create a table and store the previously stored index to MySQL'''

        sqlCreateTableCommand = 'CREATE TABLE IND_' + switch[0] + ' (last_index int)'
        cursorObject.execute(sqlCreateTableCommand)

except (KeyboardInterrupt, SystemExit):
    print("Goodbye!")
    sys.exit(0)

except Exception as e:
    print(type(e))
    print("Exeception occured:{}".format(e))
    pass

finally:
    connection.close()
```

Appendix B

mysql_data.py (Python 3.5)

```
'''
This Python Code will be the one which dynamically stores the data of the Wemo
Switch when it is executed. The data is stored on average every 2-3 seconds.
'''

# PYMYSQL MODULES
import pymysql
import pymysql.cursors

# OUIMEAUX MODULES
import ouimeaux
from ouimeaux.environment import Environment
from ouimeaux.signals import receiver, statechange, devicefound

# Connect to the database
connection = pymysql.connect(host="localhost",
                             user='root',
                             passwd='ditpwd',
                             db="DATA")


# Inserting paramater data in mysql
def kv_pairs(dict):
    '''OBTAINING THE KEY-VALUE PAIR PARAMATERS'''
    '''this function will be used to seperate
    key-value pairs in the dictionary for inserting
    data into MYSQL'''

    keys = str(list(dict.keys()))[:].replace('[', '(').replace(']', ')')
    vals = str(list(dict.values()))[:].replace('[', '(').replace(']', ')')

    keys = keys.replace("'","")
    vals = vals.replace("'","")
    return keys,vals

def SWITCH(list_switches):
    '''This function intakes the list of switches
    and returns the attributes given by the environment
    NOTE: Returns a Tuple
    '''
    # Obtaining the switch parameters
    switch_name = list_switches
    switch_wemo = env.get_switch(switch_name)
    # Note: Parameters are stored in dictionary
    switch_param = switch_wemo.insight_params
    # Delete lastchange: Unnecessary Parameter
    switch_param.pop('lastchange')

    # Adding DATE and TIME dictionary to SWITCH_PARAM
    switch_param['TIME'] = 'CURTIME()'
```

```
    switch_param['DATE'] = 'CURDATE()'

    return switch_name,switch_wemo,switch_param

# Initializing the ouimeaux environment
env = Environment()

try:
    # Commit every data as by default it is False
    connection.autocommit(True)

    # Create a cursor object
    cursorObject = connection.cursor()

    env.start()
    env.discover(3)

    '''using MYSQL functions to insert date and time
       USEFUL FUNCTIONS IN MYSQL:
       CURRENT_TIMESTAMP
       CURRENT_DATE
       CURRENT_TIME
    '''
    try:
        env.start()
        env.discover(5)

        while True:

            for i in range(len(env.list_switches())):

                #Calling the helper function
                switch = SWITCH(list(env.list_switches())[i])

                # Obtaining the switch parameters
                switch_name = switch[0]
                switch_wemo = switch[1]

                # Note: Parameters are stored in dictionary
                switch_param = switch[2]

                keys = kv_pairs(switch_param)[0]
                vals = kv_pairs(switch_param)[1]

                print('------------')

                # 1 indicates switch is on and load is on
                if switch_wemo.get_state() == 1:
                    #print("name:",SWITCH,"STATE:ON")
                    insertStatement = ("INSERT INTO DATA_"+switch[0]+"(DATE,TIME,STATE
                        )"
                                       "VALUES(CURDATE(),CURTIME(),\"S:ON  | L:ON\")")
                    cursorObject.execute(insertStatement)
```

```
                # 8 indicates switch is on but load is off
                elif switch_wemo.get_state() == 8:
                    #print("name:",SWITCH,"STATE:ON")
                    insertStatement = ("INSERT INTO DATA_"+switch[0]+"(DATE,TIME,STATE
                        )"
                                       "VALUES(CURDATE(),CURTIME(),\"S:ON  | L:OFF\")")
                    cursorObject.execute(insertStatement)

                # 0 indicates switch if off
                else:
                    #print("name:",SWITCH,"STATE:OFF")
                    insertStatement = ("INSERT INTO DATA_"+switch[0]+"(DATE,TIME,STATE
                        )"
                                       "VALUES(CURDATE(),CURTIME(),\"S:OFF | L:OFF\")")
                    cursorObject.execute(insertStatement)

                #inserting new params
                cursorObject.execute('INSERT INTO ' +switch[0]+ ' %s VALUES %s' % (
                    keys, vals))
                env.wait(1)

    except (KeyboardInterrupt, SystemExit):
        print("Goodbye!")
        sys.exit(0)

    # # List the tables using SQL command
    # sqlShowTablesCommand = "show tables"
    #
    # # Execute the SQL command
    # cursorObject.execute(sqlShowTablesCommand)


except Exception as e:
    print("Exeception occured:{}".format(e))
    pass

finally:

    connection.close()
```

Appendix C

mysql_plot.py (Python 3.5)

```
'''
This Python Code generates the graphs of Currentpower vs. Time.
There are three graphs:

MAKE_PLOT(X,Y):
This plots the currentpower vs. time from the original set of data. This may
not be optimal if one wants to minimize the amount of data sets for optimizing
memory space

PLOT_FINALDATA_SMOOTHING(X,Y):
This plots the currentpower vs. time first smoothing out the original data set.
The maximum value is then stored using argmax which will be a reference point to
be connected. This will reduce the amount of dataset although if one wants to
keep the currentpower drained by the appliance as is, then it may not be optimal.

PLOT_FINALDATA(X,Y):
This plots the currentpower vs. time first storing the point where
currentpower = 0[W]. It will neglect all the other points with 0[W] until there
is a change in current - this interval will then be stored. It is the same plot#
as MAKE_PLOT(X,Y) reducing the amount of unnecessary data.

'''

# NUMPY MODULES
import numpy as np

# PYMYSQL MODULES
import pymysql
import pymysql.cursors

# OUIMEAUX MODULES
import ouimeaux
from ouimeaux.environment import Environment
from ouimeaux.signals import receiver, statechange, devicefound

# MATPLOTLIB MODULES
import matplotlib.pyplot as plt

# PLOTLY and PANDAS MODULES
import plotly
import pandas
from plotly.graph_objs import *

# NUMPY MODULES
import numpy as np

# SCIPY MODULES
from scipy.interpolate import spline
from scipy.interpolate import interp1d
from scipy.signal import argrelextrema
```

```
# Initializing the ouimeaux environment
env = Environment()

'''SIDENOTES
In order to access a certain column of interest the
Mysql syntax is: select column from table

TABLE STRUCTURE:
In order to store the current data, we have the time axis. There are
different time interval readings between each successive data. Whenever
we store the current data, the time will start from a reference of 0 seconds.
The time interval between each reading will be used from the stored query called,
'TIME_DIFF'

USEFUL MYSQL FUNCTIONS:
SELECT TIME_TO_SEC(TIMEDIFF('12:01:00', '12:00:00')) diff;
^ Gives the time difference in seconds
'''

# Connect to the database
connection = pymysql.connect(host="localhost",
                             user='root',
                             passwd='ditpwd',
                             db="DATA")

def empty(list):
    '''A simple function which checks if a list
       is empty or not'''

    if len(list) == 0:
        return True
    else:
        return False

def np_empty(nparray):
    '''A simple function which checks if a numpy
       array is empty or not'''

    if np.array.size == 0:
        return True
    else:
        return False

def SWITCH(list_switches):
    '''This function intakes the list of switches
    and returns the attributes given by the environment
    NOTE: Returns a Tuple
    '''
    # Obtaining the switch parameters
    switch_name = list_switches
    switch_wemo = env.get_switch(switch_name)
    # Note: Parameters are stored in dictionary
```

```
    switch_param = switch_wemo.insight_params
    # Delete lastchange: Unnecessary Parameter
    switch_param.pop('lastchange')

    # Adding DATE and TIME dictionary to SWITCH_PARAM
    switch_param['TIME'] = 'CURTIME()'
    switch_param['DATE'] = 'CURDATE()'

    return switch_name,switch_wemo,switch_param

def TIME_IND(switch,last_index):
    '''This function will compute the time difference
    between each currentpower and store it in the table IND_Insight
    '''
    cursorObject = connection.cursor()
    last_ind = str(last_index)

    # We must be careful not to keep on storing the same values
    ROW = "Select last_index from IND_" + switch[0]
    cursorObject.execute(ROW)
    prev_index = cursorObject.fetchall()

    if empty(prev_index) == True:
        insertStatement = ("INSERT INTO IND_"+switch[0]+"(last_index) "
                           "VALUES("+last_ind+")")

        cursorObject.execute(insertStatement)

    elif prev_index[-1][0] == last_index:
        return

    else:
        insertStatement = ("INSERT INTO IND_"+switch[0]+"(last_index) "
                           "VALUES("+last_ind+")")

        cursorObject.execute(insertStatement)

def store_time_diff(time_rows):
    '''This function uses a list called time_rows which is a list
       that contains the rows of the times switch was on or off
    '''
    # Create a list to store the time differences
    time_del = []

    for j in range(len(time_rows)):
        if j == len(time_rows)-1:
            break
        else:
            # Fetch the time in the rows
            t1 = time_rows[j][0].seconds
            t2 = time_rows[j+1][0].seconds
            time_del.append(t2-t1)
```

```
    time_del.insert(0,0)
    return time_del

def store_currentpower(cp_rows):
    '''This function uses a list called time_rows which is a list
       contains the rows of the switches currentpower'''

    # Create a list to store the cp_rows
    currentpower = []

    for i in range(len(cp_rows)):
        currentpower.append(cp_rows[i][0]/1000)

    return currentpower

def MAKE_PLOT(time_del,current_power):
    '''A function which generates currentpower vs. time plot'''

    # Using matplotlib to plot currentpower[W] vs time[s]
    # Initialize these variables as X-Y
    X = np.cumsum(time_del)
    Y = list(current_power)

    # Show plot
    plt.plot(X,Y)
    plt.ylabel('currentpower [W]')
    plt.xlabel('time [s]')
    plt.title('currentpower vs. time')
    plt.show()

def fetch_data(switch,start_index,end_index):
    '''Function used to fetch certain rows of data
        By default it fetches from the very recent to last'''

    # TIME DATA
    ROW = "Select TIME from " + switch[0]
    cursorObject.execute(ROW)
    time_rows = cursorObject.fetchall()
    time_rows = time_rows[start_index:end_index+1]
    time_del = store_time_diff(time_rows)

    # CURRENTPOWER DATA
    ROW = "Select currentpower from " + switch[0]
    cursorObject.execute(ROW)
    cp_rows = cursorObject.fetchall()
    cp_rows = cp_rows[start_index:end_index+1]
    current_power = store_currentpower(cp_rows)

    return time_del,current_power

def CONV_NPARRAY(x_list,y_list):
    '''simple function which converts lists to
       numpy arrays. Returns a tuple of X & Y'''
```

```
210
    X = np.array(x_list)
    Y = np.array(y_list)

    return X,Y
215


def CPT_SWITCH(switch):
    '''A function called twice within the try-exception statement. It's
       The one which compactly generates the plot'''
220
    # Rows are stored in a tuple
    ROW = "Select TIME from " + switch[0]
    cursorObject.execute(ROW)
    time_rows = cursorObject.fetchall()
225
    # A list which contains the difference between each successive time
    time_del = store_time_diff(time_rows)

    # CALL THE TIME_DIFF FUNCTION TO STORE THE LAST
230 # USED INDEX
    TIME_IND(switch,len(time_del))

    # Fetching the currentpower Data
    ROW = "Select currentpower from " + switch[0]
235 cursorObject.execute(ROW)

    # Rows are stored in a tuple
    cp_rows = cursorObject.fetchall()

240 # list containing currentpower
    current_power = store_currentpower(cp_rows)

    # Generate actual plot
    MAKE_PLOT(time_del,current_power)
245 # Generate plot
    # PLOT_FINALDATA(time_del,current_power)

### First approach in obtaining max and min values from smoothing out plot
def smooth(X,Y,spacing):
250 '''This function returns the respective X and Y coordinates
       To smooth out the plot. Note that the coordinates are
       converted to numpy arrays'''

    X = np.cumsum(X)
255 Y = list(Y)
    XP,YP = CONV_NPARRAY(X,Y)
    XP_SMOOTH = np.linspace(XP.min(),XP.max(),spacing)
    YP_SMOOTH = spline(XP,YP,XP_SMOOTH)

260 return XP_SMOOTH,YP_SMOOTH

def refined_data(XP,YP):
```

```
    '''This function takes the X-Y array coordinates which have
       been smoothed out and stores the local max/min vals. It
       then after appends these such values in order to truncate
       values'''

    Y_IND_MIN = argrelextrema(YP, np.less)[0]
    Y_REFINED_MIN = YP[argrelextrema(YP, np.less)[0]]
    X_REFINED_MIN = np.array([XP[i] for i in Y_IND_MIN])

    Y_IND_MAX = argrelextrema(YP, np.greater)[0]
    Y_REFINED_MAX = YP[argrelextrema(YP, np.greater)[0]]
    X_REFINED_MAX = np.array([XP[i] for i in Y_IND_MAX])

    X_REFINED = np.append(X_REFINED_MIN,X_REFINED_MAX)
    Y_REFINED = np.append(Y_REFINED_MIN,Y_REFINED_MAX)

    return X_REFINED,Y_REFINED

def xintervals(XP,YP):
    '''This function takes the X and Y coordinates and
       stores the beginning and ending intervals intersecting
       the x-axis. However, we store all the values in
       numpy array'''

    XY = []
    XY_TEMP = []
    XP = np.cumsum(XP)
    # Placeholders
    XP = np.append(XP,0)
    YP = np.append(YP,0)
    # First column: X # Seconds column: Y
    xy_pairs = np.vstack([XP,YP]).T

    i = 0
    while i < xy_pairs.shape[0]-1:
        if xy_pairs[i][1] == 0:
            # Store the first value where it hits
            # x-axis
            XY_TEMP.append((xy_pairs[i][0],xy_pairs[i][1]))
            #print(XY_TEMP)
            if xy_pairs[i+1][1] > 0:
                if 0 <= i < xy_pairs.shape[0]:
                    if i == 0:
                        XY.append(XY_TEMP[0])
                        XY_TEMP = []
                    else:
                        XY.append(XY_TEMP[0])
                        XY.append(XY_TEMP[-1])
                        XY_TEMP = []
                else:
                    pass

            elif i == xy_pairs.shape[0]-2 and XY_TEMP[-2][1] == 0:
```

```
                XY.append(XY_TEMP[0])
                XY.append(XY_TEMP[-1])

        i += 1

    return XY

def remove_xintervals(XP,YP):
    '''This function removes all the points with y=0. This will greatly
       reduce the data as many devices will not be on for most of the time,
       with the exception with fridges, etc.'''

    num_nonzeros = np.count_nonzero(YP)
    XP = np.cumsum(XP)
    XY_TEMP = np.vstack((XP,YP)).T
    XY_TEMP = XY_TEMP[np.argsort(XY_TEMP[:,1])]
    num_zeros = XY_TEMP.shape[0] - num_nonzeros

    # New X-Y plots without having values y=0
    XY_NEW = XY_TEMP[(num_zeros):XY_TEMP.shape[0]-(num_zeros-2)]

    return XY_NEW

def final_data(XP,YP):
    '''This function obtains the final data points using the smoothing
       function, spline, provided by Python scipy module'''

    XP_SMOOTH,YP_SMOOTH = smooth(X,Y,200)
    X_REFINED,Y_REFINED = refined_data(XP_SMOOTH,YP_SMOOTH)
    XY_TEMP = np.vstack((X_REFINED,Y_REFINED)).T
    XY = np.array(xintervals(XP,YP))

    len_history = []
    i = XY_TEMP.shape[0]-1
    len_history.append(i)

    while i >= 0:
        # base case
        if i == XY_TEMP.shape[0]-1:

            if XY_TEMP[i][1] < 0:
                XY_TEMP = np.delete(XY_TEMP,i,axis = 0)
            else:
                pass

        else:
            if XY_TEMP[i][1] < 0:
                # Check if the update is the same as the previous length.
                # If it is, then it must mean there are no more values to
                # remove.
                if len_history[-1] == len_history[-2]:
                    break
                else:
```

```
                    XY_TEMP = np.delete(XY_TEMP,i,axis = 0)
                    len_history.append(XY_TEMP.shape[0])
            else:
                pass

        len_history.append(i)
        i -= 1

    # Gathering all the refined data
    # There are three cases we have to keep in mind:
    if XY.shape[0] == 0:
        return XY_TEMP
    elif XY_TEMP.shape[0] == 0:
        return XY
    else:
        return np.vstack((XY,XY_TEMP))

def PLOT_FINALDATA_SMOOTHING(XP,YP):
    '''function which plots the points using python spline'''

    XY = final_data(XP,YP)
    x,y = XY[np.argsort(XY[:,0])].T
    # print((XY[np.argsort(XY[:,0])].T).shape[1])
    plt.plot(x,y)
    plt.ylabel('currentpower [W]')
    plt.xlabel('time [s]')
    plt.title('currentpower vs time')
    plt.show()

def PLOT_FINALDATA(XP,YP):

    XY1 = xintervals(XP,YP)
    XY2 = remove_xintervals(XP,YP)
    XY = np.vstack((XY1,XY2))
    x,y = XY[np.argsort(XY[:,0])].T
    # print((x,y))
    plt.plot(x,y)
    plt.ylabel('currentpower [W]')
    plt.xlabel('time [s]')
    plt.title('currentpower vs time')
    plt.show()

try:
    # Start the ouimeaux environment
    env.start()
    env.discover(3)

    # Commit every data as by default it is False
    connection.autocommit(True)

    # Create a cursor object
    cursorObject = connection.cursor()
```

```
    # We will have to iterate through each available switch
    for i in range(len(env.list_switches())):

        switch = SWITCH(list(env.list_switches())[i])

        # Rows are stored in a tuple
        ROW = "Select last_index from IND_" + switch[0]
        cursorObject.execute(ROW)
        index_rows = cursorObject.fetchall()

        if empty(index_rows) == True or len(index_rows) == 1:
            ''' This part of the code only executes once - when we generate a plot
            the first time'''
            #print('hi')
            CPT_SWITCH(switch)

        else:
            ''' This part of the code is executed every time after the first if
            statement'''

            ROW = "Select TIME from " + switch[0]
            cursorObject.execute(ROW)
            end_index = len(cursorObject.fetchall())
            TIME_IND(switch,end_index)

            ROW = "Select last_index from IND_" + switch[0]
            cursorObject.execute(ROW)

            # Initiate the starting index to whatever is needed
            start_index = cursorObject.fetchall()[-2][0]

            # Generate the plot from the last stored value
            X,Y = fetch_data(switch,start_index,end_index)

            ###
            # Generate the plot which gives the smooth curves

            # XP_SMOOTH,YP_SMOOTH = smooth(X,Y,200)
            # plt.plot(XP_SMOOTH,YP_SMOOTH)
            # plt.show()

            ###



            ###
            # Generate the points where the max/min values lie from the
            # Smoothed out curve.

            # X_REFINED,Y_REFINED = refined_data(XP_SMOOTH,YP_SMOOTH)
            # plt.plot(X_REFINED,Y_REFINED,'ro')
            # plt.plot(XP_SMOOTH,YP_SMOOTH)
```

```
            ###



            ###
            # Generate the actual plot from the Wemo Devices

            # MAKE_PLOT(X,Y)

            ###
            # Generate the final plot using the spline function

            # PLOT_FINALDATA_SMOOTHING(X,Y)

            ###

            ###
            # Generate the final plot containing all the ON state points
            # and only the beginning and ending points when currentpower = 0

            PLOT_FINALDATA(X,Y)

            ###

except Exception as e:

    print("Exeception occured:{}".format(e))
```