User Documentation

By: John Aaron Dayao Lumagbas

PowerUsageLauncher.py

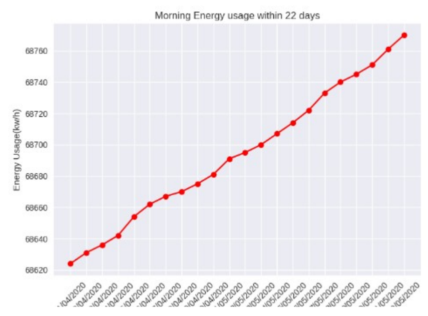
Overview

PowerUsageLauncher.py is the program created for Investigation 1, which simulates a plot for the 3 different energy readings throughout 3-time intervals. These intervals are:

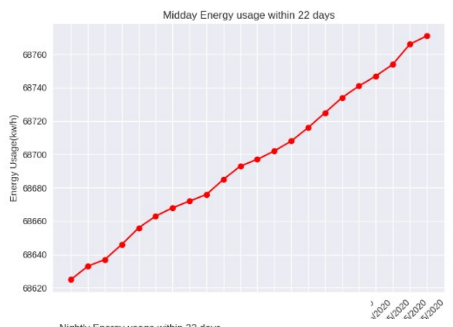
- Morning
- Afternoon (Midday)
- Night
- Cumulative (Total used, found using the difference of Nightly and Morning energy readings)

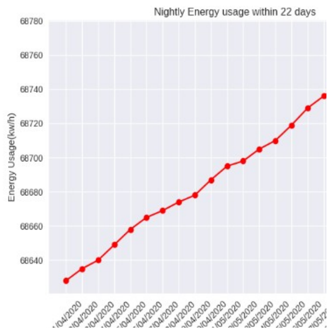
The program plots each energy readings into 3 different line plots, while the cumulative is plotted into a combined bar and line graph. The cumulative energy usage has been plotted this way to clearly see the energy fluctuation of energy throughout each of the 22 days. (21/04 -12/05)

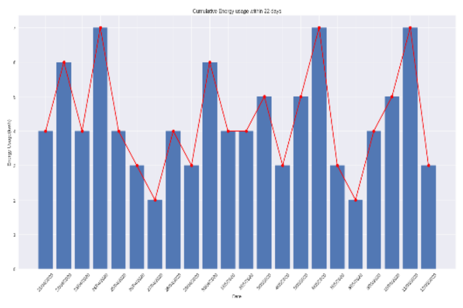
Plots – Morning/Nighty Energy Usage



Plot 2 – Midday Energy Usage



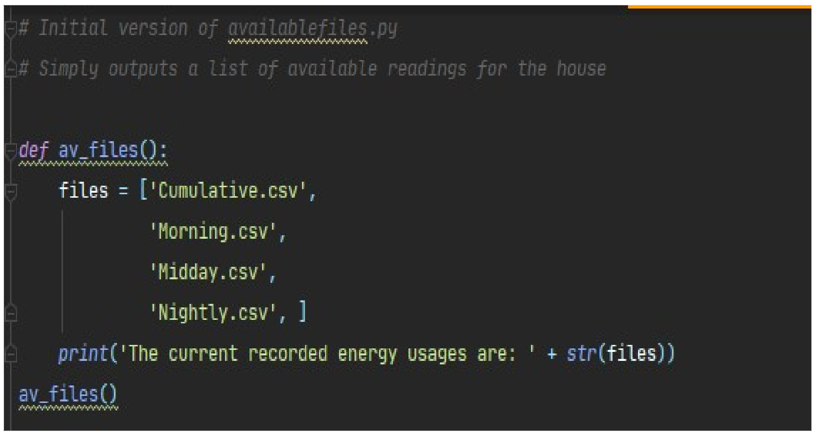




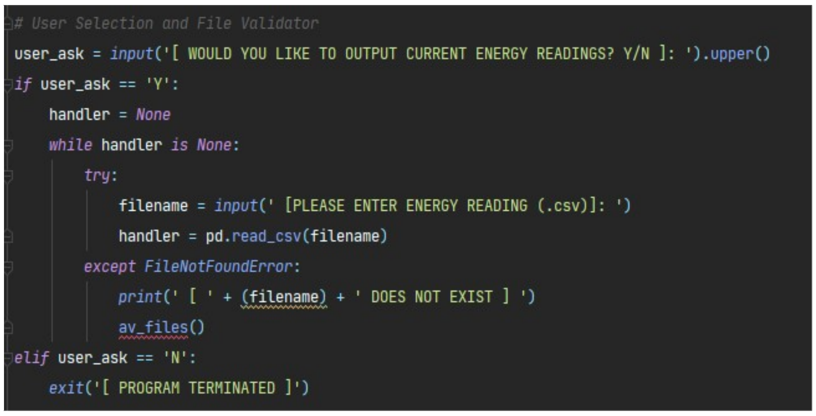
The program has many features evident within its programming and structure, these are:

- <u>User interface and exception handling</u>: The first feature of the program creates a mini user interface which gives the users the option to launch the program and a small selection interface that outputs the available file. If a user inputs ‘Y’ or ‘y’, it prompts the users to select filename from the module/function called ‘availablefiles.py’ , the user inputs a selection of these file and will be read using a pandas library function called pd.read_csv(). However, if a user enters a csv file incorrectly, or the csv file does not exist the program outputs an error message and prompts the user to try again.
- If the user inputs ‘N’ or ‘n’ in the first user prompt, the program will shut itself off and will need to be launched again.
- ‘availablefiles.py’ is a user defined function that outputs the available energy reading files.

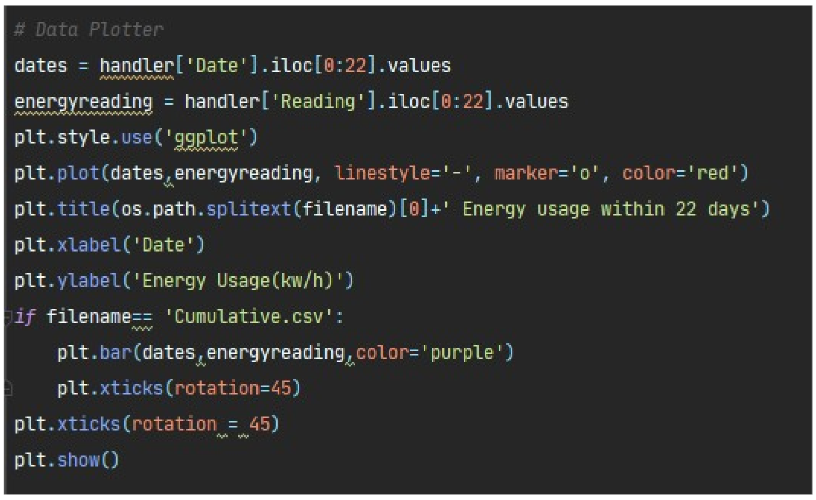
- 'handler' and 'filename' is passed around within the program for plotting purposes.
- Data plotter: The data plotter first loads the date and readings contained in the reading by loading in the first 23 [0:22] line. Once it loads date and reading, the plotter plots the dates as the x axis, and energyreading as its y axis. The title is created on whatever 'filename' is called, for example if 'Morning.csv' is loaded, the plt.title() will output in the graph: 'Morning Energy usage within 22 days'. The small if condition is that if the 'filename' loaded is the Cumulative.csv, it adds an additional bar plot on to the already existing bar graph. The axis labels are rotated to 45 degrees to correct the overlapping of dates and time, so that each date can be read properly.



```
# Initial version of availablefiles.py
# Simply outputs a list of available readings for the house

def av_files():
    files = ['Cumulative.csv',
             'Morning.csv',
             'Midday.csv',
             'Nightly.csv', ]
    print('The current recorded energy usages are: ' + str(files))
av_files()
```



```
# User Selection and File Validator
user_ask = input('[ WOULD YOU LIKE TO OUTPUT CURRENT ENERGY READINGS? Y/N ]: ').upper()
if user_ask == 'Y':
    handler = None
    while handler is None:
        try:
            filename = input(' [PLEASE ENTER ENERGY READING (.csv)]: ')
            handler = pd.read_csv(filename)
        except FileNotFoundError:
            print(' [ ' + (filename) + ' DOES NOT EXIST ] ')
            av_files()
elif user_ask == 'N':
    exit('[ PROGRAM TERMINATED ]')
```



```
# Data Plotter
dates = handler['Date'].iloc[0:22].values
energyreading = handler['Reading'].iloc[0:22].values
plt.style.use('ggplot')
plt.plot(dates,energyreading, linestyle='-', marker='o', color='red')
plt.title(os.path.splitext(filename)[0]+' Energy usage within 22 days')
plt.xlabel('Date')
plt.ylabel('Energy Usage(kw/h)')
if filename== 'Cumulative.csv':
    plt.bar(dates,energyreading,color='purple')
    plt.xticks(rotation=45)
plt.xticks(rotation = 45)
plt.show()
```

- Once all conditions are met and the correct data is loaded, the data plotter plots the above plots shown on the previous page.
- These graphs are also saved in the directory '\Investigation1\Graphs' for a full view of the graph and later use.

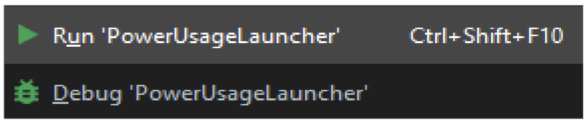
Libraries used:

- matplotlib.pyplot, a module essential to creating and plotting the data, pyplot is intended for creating interactive plots.
- pandas, a library that is used to handle data and data manipulation and allows the reading of different supported file types, particularly .csv files.

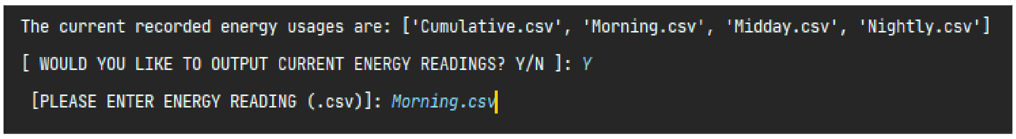
- os, is a module that enables the program to interact with the operating system (used mainly for using 'filename' variable as a plot title)

User Guide

1. To run the program, simply pres 'Ctrl-Shft-F10' or right click on the program title and press 'Run'.
   i. If running on linux, the program can be run by inputting 'python3 PowerUsageLauncher.py' into the terminal
2. The program will now proceed to call the function av_files() and output the available energy reading csv files.

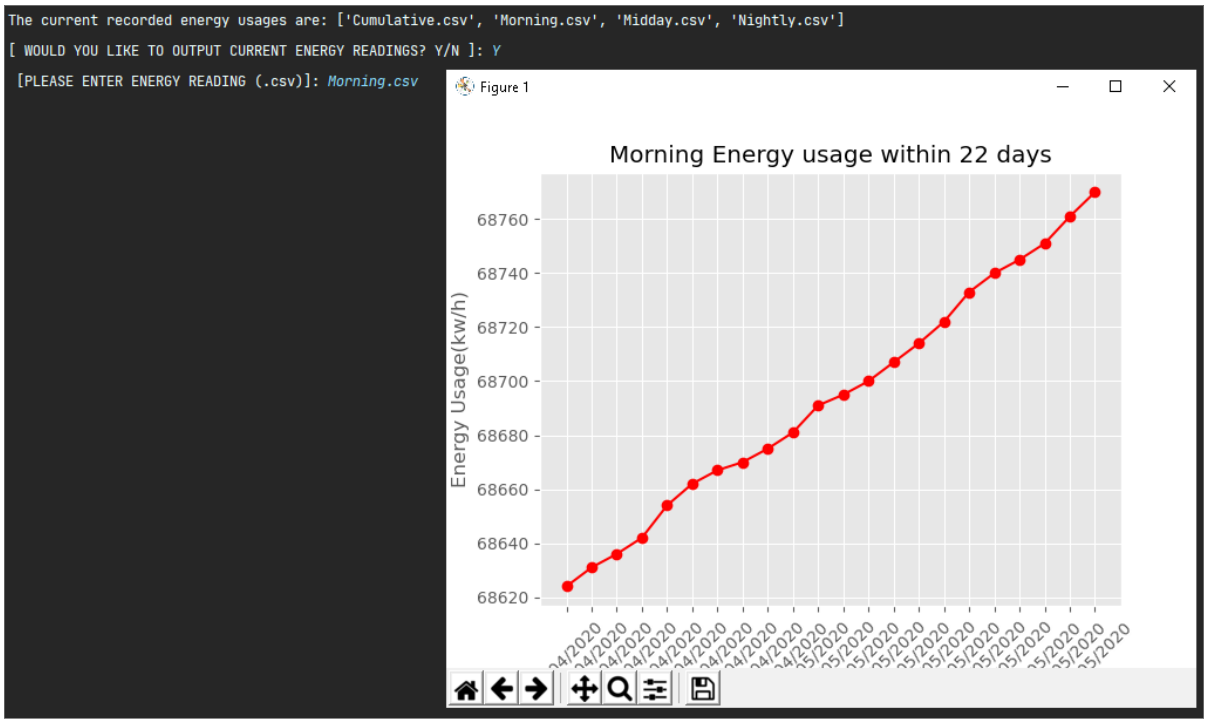

Run 'PowerUsageLauncher' Ctrl+Shift+F10
Debug 'PowerUsageLauncher'

3. The user will then be prompted to launch the program, there are two inputs that can be used: Y to launch the program and N to exit the program. If the user chooses to proceed and input Y, the program will then proceed and launch it's exception handler and prompt the user to select a .csv file outputted from the previous function.



```
The current recorded energy usages are: ['Cumulative.csv', 'Morning.csv', 'Midday.csv', 'Nightly.csv']
[ WOULD YOU LIKE TO OUTPUT CURRENT ENERGY READINGS? Y/N ]: Y
 [PLEASE ENTER ENERGY READING (.csv)]: Morning.csv
```

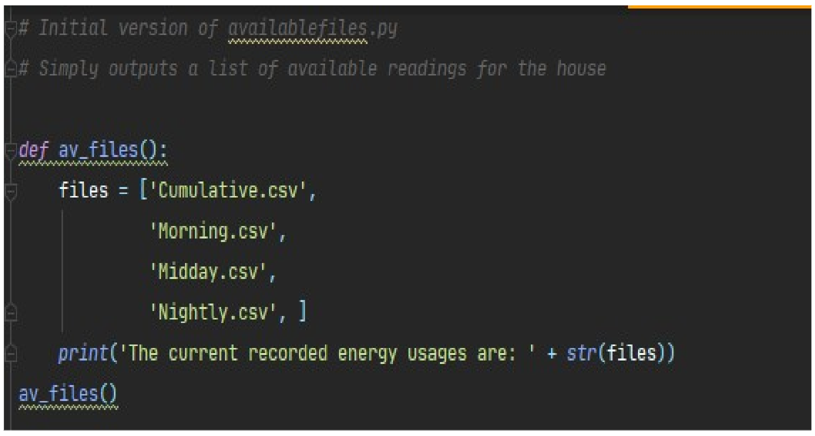
   However, if the user inputs 'N' , the program will be terminated, leaving a message saying: [PROGRAM TERMINATED]
4. If the correct .csv file is inputted, program will then proceed to load the necessary data and plot this according to the data plotter shown previously.
5. The program will then exit as soon as the user closes the graph.



```
The current recorded energy usages are: ['Cumulative.csv', 'Morning.csv', 'Midday.csv', 'Nightly.csv']
[ WOULD YOU LIKE TO OUTPUT CURRENT ENERGY READINGS? Y/N ]: Y
 [PLEASE ENTER ENERGY READING (.csv)]: Morning.csv
```

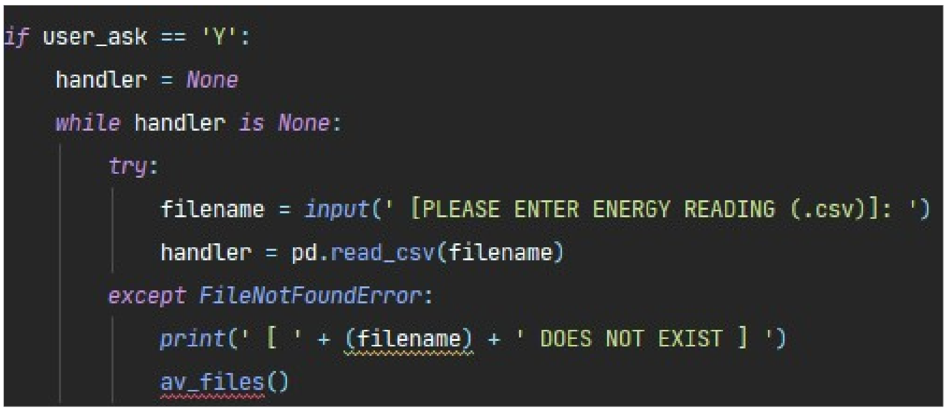
Discussion

PowerUsageLauncher is a simply yet smart program that enables a user to create a simulated graph based on stored .csv files containing energy readings from Morning, Afternoon and Nighttime. As mentioned earlier, the program creates a selection of .csv files and is outputted to the user. This has been implemented by created a user defined function known as 'availablefiles.py' as shown previously:



```python
# Initial version of availablefiles.py
# Simply outputs a list of available readings for the house

def av_files():
    files = ['Cumulative.csv',
             'Morning.csv',
             'Midday.csv',
             'Nightly.csv', ]
    print('The current recorded energy usages are: ' + str(files))
av_files()
```

As mentioned in the overview, the program features an exception handler which validates if the user input is correct. This is implemented by containing setting handler first as 'None' and creating a while loop which contains the try/except handler. The exception handling implements a function called 'FileNotFoundError' which detects an error if the input in 'filename' cannot be found.

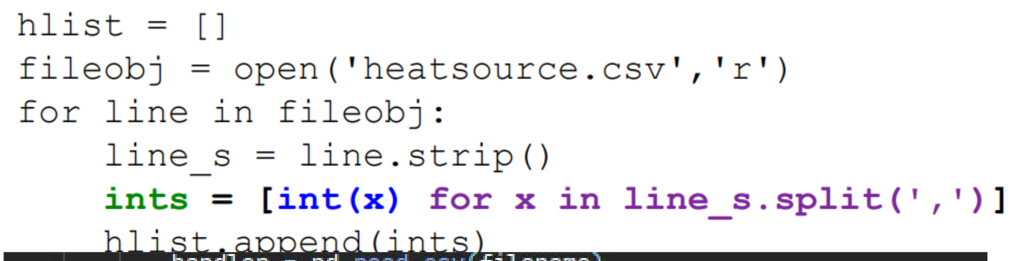


```python
if user_ask == 'Y':
    handler = None
    while handler is None:
        try:
            filename = input(' [PLEASE ENTER ENERGY READING (.csv)]: ')
            handler = pd.read_csv(filename)
        except FileNotFoundError:
            print(' [ ' + (filename) + ' DOES NOT EXIST ] ')
            av_files()
```

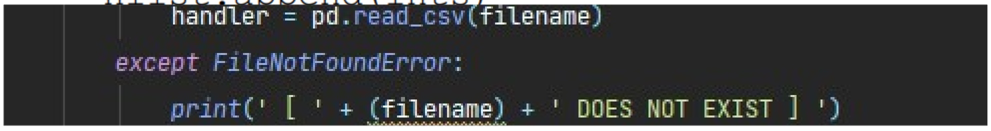
The reason an exception handler has been implemented within the program is so that the program is able to identify the stored csv files that the program will need to use and specify if the inputted filename does not exist.

Within the exception handler, if the filename is true, a pandas function known as pd.read_csv loads in the csv file as a data frame and allow the program to handle and manipulate the data loaded form the .csv file. Pandas module has been used to read in csv files as it is a much cleaner way of handling data, compared to having to manually append each integer into an empty array. Pandas also does not require the use of the datetime () function as it is able to properly read and pass on strings contained within a file.

Notice in the images below, the code length difference when pandas (pd) is used vs needing to append everything into an empty array:

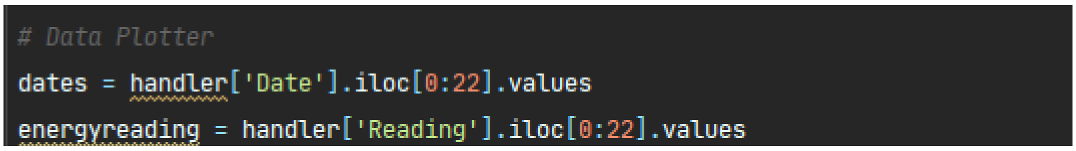


```
hlist = []
fileobj = open('heatsource.csv','r')
for line in fileobj:
    line_s = line.strip()
    ints = [int(x) for x in line_s.split(',')]
    hlist.append(ints)
```



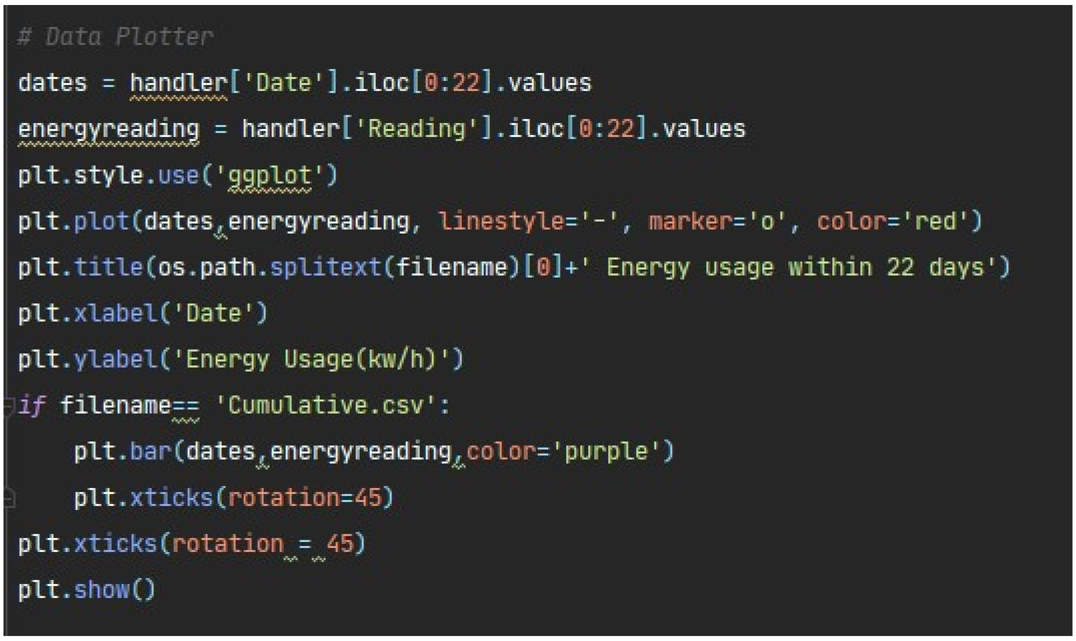
```
        handler = pd.read_csv(filename)
    except FileNotFoundError:
        print(' [ ' + (filename) + ' DOES NOT EXIST ] ')
```

The data plotter further uses the pandas pd to read in the specific columns of the selected csv, by creating our x and y variables named “dates” and “energyreading” which utilize pandas by locating the column names and the range of values within each columns. the. iloc[].values function locates the specified range of values within the csv file. It is then loaded into the dates and energyreading variables and are stored as lists.



```
# Data Plotter
dates = handler['Date'].iloc[0:22].values
energyreading = handler['Reading'].iloc[0:22].values
```

The variables are then passed on and are used as the x and y values of the data plotter



```
# Data Plotter
dates = handler['Date'].iloc[0:22].values
energyreading = handler['Reading'].iloc[0:22].values
plt.style.use('ggplot')
plt.plot(dates,energyreading, linestyle='-', marker='o', color='red')
plt.title(os.path.splitext(filename)[0]+' Energy usage within 22 days')
plt.xlabel('Date')
plt.ylabel('Energy Usage(kw/h)')
if filename== 'Cumulative.csv':
    plt.bar(dates,energyreading,color='purple')
    plt.xticks(rotation=45)
plt.xticks(rotation = 45)
plt.show()
```

As mentioned previously a condition has been created for choosing ‘Cumulative.csv’ where it plots a bar graph in addition to the line graph. This has been implemented so that if ‘filename’ meets the condition ‘Cumulative.csv’, it enables plt.bar() to be used and shown.

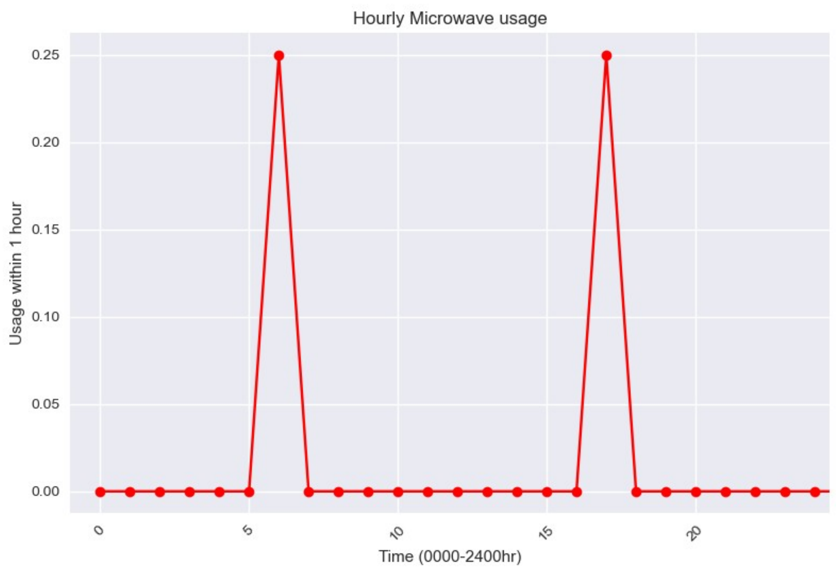
It is also worth noting that the plot takes in the ‘filename’ input as a title, os.path.splitext splits the filename from its file extension name.

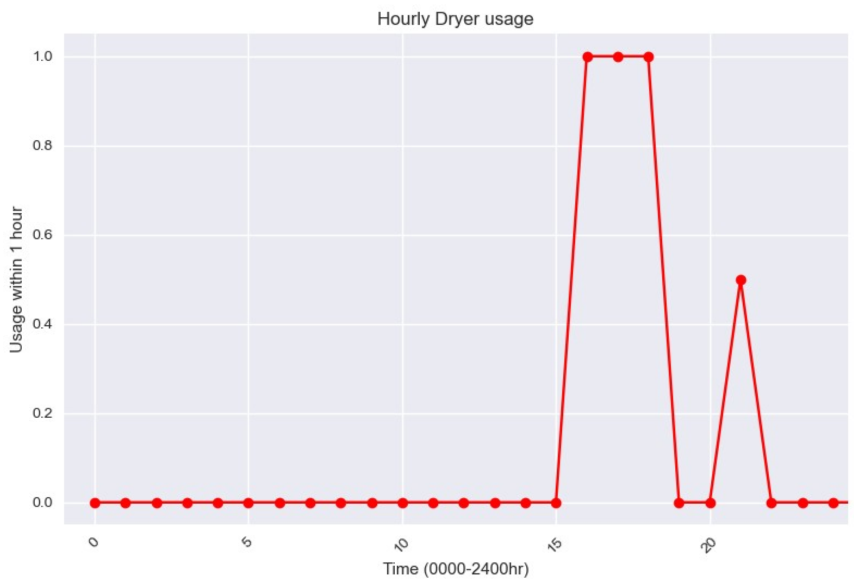
PowerModelLauncher.py

Overview

PowerModelLauncher.py is the program produced for Investigation 2, which models a live moving plot and simulates how each appliance are used within each hour of the day. The program features a wide range of features and functionalities including the csv loader, allows the users to choose from a selection of appliances they want to choose from, a live plot as mentioned previously, and allows to calculate the total consumption of each appliances.

Example: Moving line graph of the Microwave and Dryer appliances

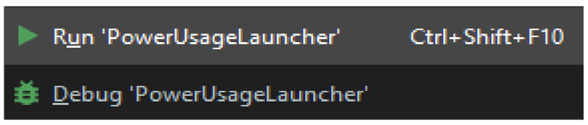




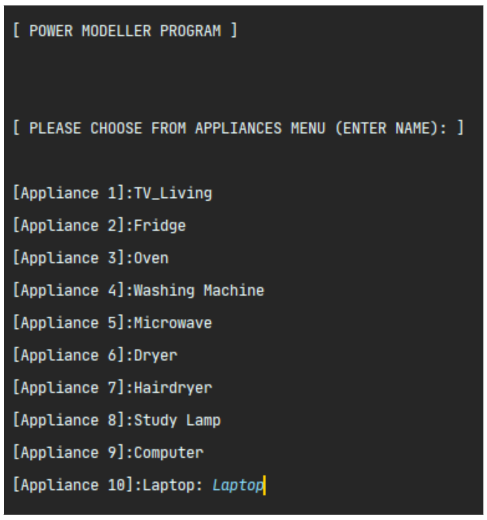
.

User Guide

1. To run the program, simply press ‘Ctrl-Shft-F10’ or right click on the program title and press ‘Run’.
   i. If running on linux, the program can be run by inputting ‘python3 PowerModelLauncher.py’ into the terminal
2. The program will then output an inventory of appliances an prompt to select an appliance which can be chosen by the following keywords:


Run 'PowerUsageLauncher' Ctrl+Shift+F10
Debug 'PowerUsageLauncher'

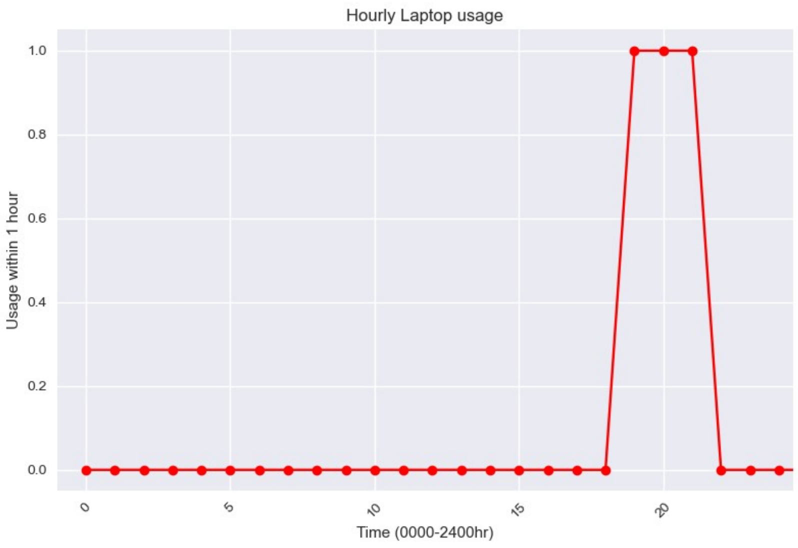
- TV
- Fridge
- Oven
- Washing Machine
- Microwave
- Dryer
- Hairdryer
- Study Lamp
- Computer
- Laptop

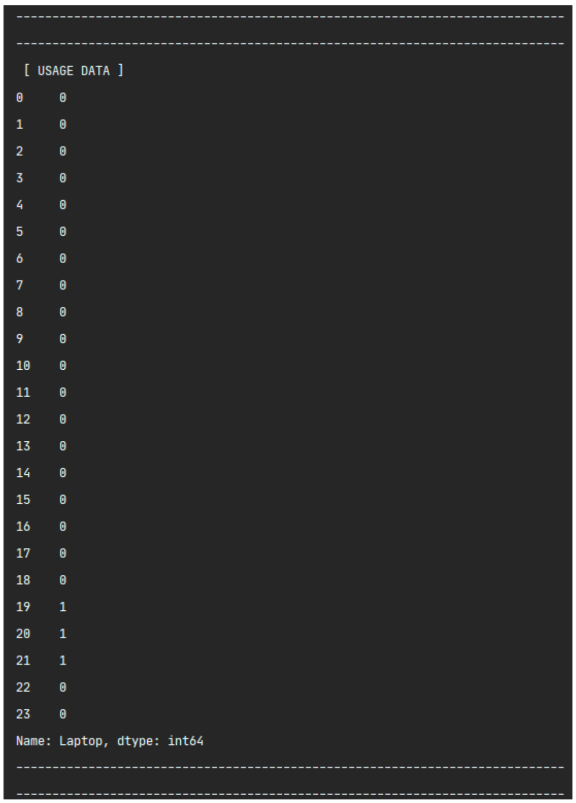


```
[ POWER MODELLER PROGRAM ]


[ PLEASE CHOOSE FROM APPLIANCES MENU (ENTER NAME): ]

[Appliance 1]:TV_Living
[Appliance 2]:Fridge
[Appliance 3]:Oven
[Appliance 4]:Washing Machine
[Appliance 5]:Microwave
[Appliance 6]:Dryer
[Appliance 7]:Hairdryer
[Appliance 8]:Study Lamp
[Appliance 9]:Computer
[Appliance 10]:Laptop: Laptop
```

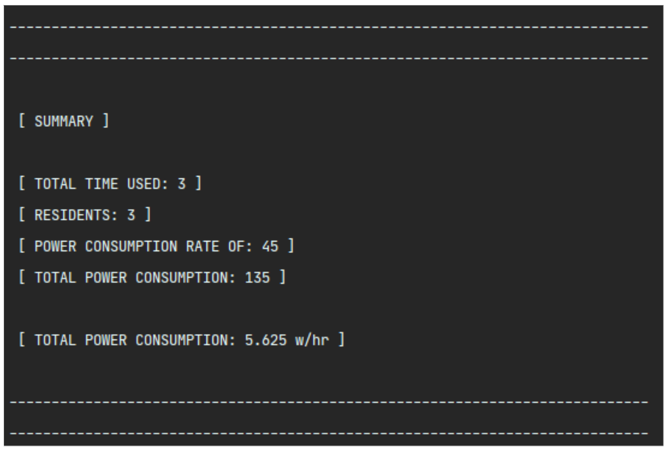
If an appliance does not exist in the inventory or is misspelled, it will prompt to re-enter the proper existing appliances.

3. Once the correct the appliance is inputted, the program will then take it's time to load in the required data for the specific appliance entered. i.e. above laptop.
4. The data gathered will then be used to create a live moving data (unfortunately, we cannot place moving images onto word/pdf docs, so the example show is the resulting finished graphed.
5. After the plot has finished simulating, the graph can be exited to proceed with the calculations
6. The data used will then be outputted for user reviewal.
7. This data is then used to calculate the total watt consumption of the specified





```
---------------------------------------------------------------------------
---------------------------------------------------------------------------
 [ USAGE DATA ]
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
18    0
19    1
20    1
21    1
22    0
23    0
Name: Laptop, dtype: int64
---------------------------------------------------------------------------
---------------------------------------------------------------------------
```



```
---------------------------------------------------------------------------
---------------------------------------------------------------------------


[ SUMMARY ]

[ TOTAL TIME USED: 3 ]
[ RESIDENTS: 3 ]
[ POWER CONSUMPTION RATE OF: 45 ]
[ TOTAL POWER CONSUMPTION: 135 ]

[ TOTAL POWER CONSUMPTION: 5.625 w/hr ]

---------------------------------------------------------------------------
---------------------------------------------------------------------------
```

appliance by multiplying its total usage hour by its power rate. It is then outputted as a summary.

8. Once all the activities are done, the program can be exited, and if needs be, re-do step 1 and forward to simulate other house appliances.

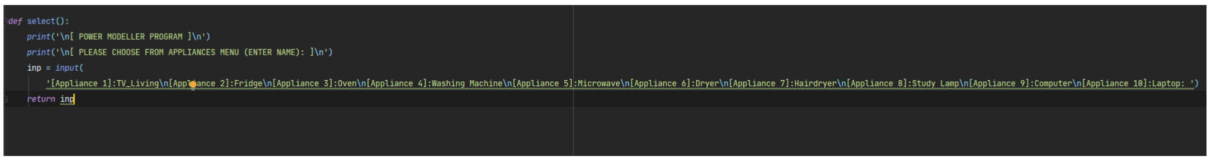
NOTE:

- To simulate the hourly usage, the keywords in step 2 must be clearly entered according to how it is written.
- If the graph wants to be saved, please finish the simulation first where all data has been plotted, other wise the graph will be plotted unfinished if saved during simulation.
- The time in the x-axis is set as 24hr time, i.e 13hr = 1pm
- The numbers plotted in the y axis represent the appliance's on/off time, for example. 1 means the appliance was on for 1 hour, 0.5 for 30 mins, and 0 for off.

Discussion

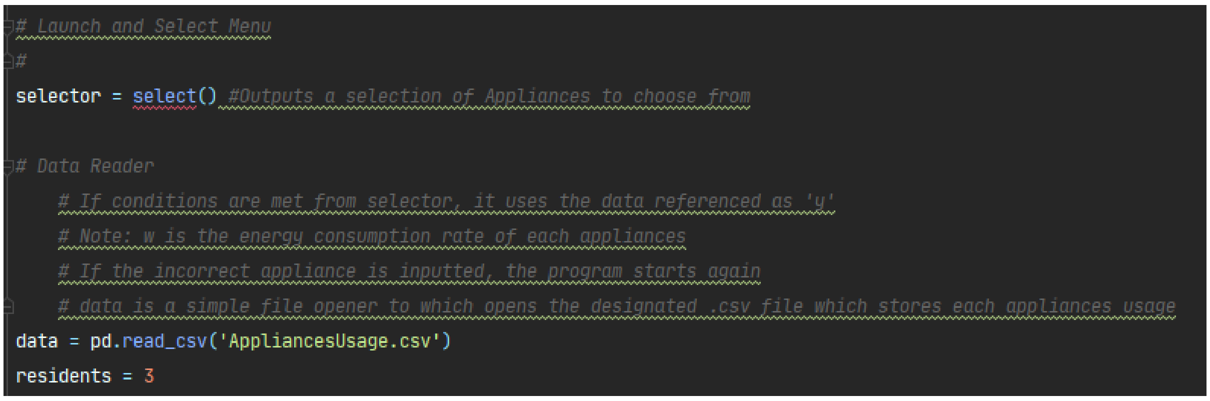
As mentioned in the overview, the program features different functions

User Input and appliance selection: The first function and feature is the user selection called from the user defined function named 'selectionmenu' and calls it through 'select()'



```
def select():
    print('\n[ POWER MODELLER PROGRAM ]\n')
    print('\n[ PLEASE CHOOSE FROM APPLIANCES MENU (ENTER NAME): ]\n')
    inp = input(
        '[Appliance 1]:TV_Living\n[Appliance 2]:Fridge\n[Appliance 3]:Oven\n[Appliance 4]:Washing Machine\n[Appliance 5]:Microwave\n[Appliance 6]:Dryer\n[Appliance 7]:Hairdryer\n[Appliance 8]:Study Lamp\n[Appliance 9]:Computer\n[Appliance 10]:Laptop: ')
    return inp
```

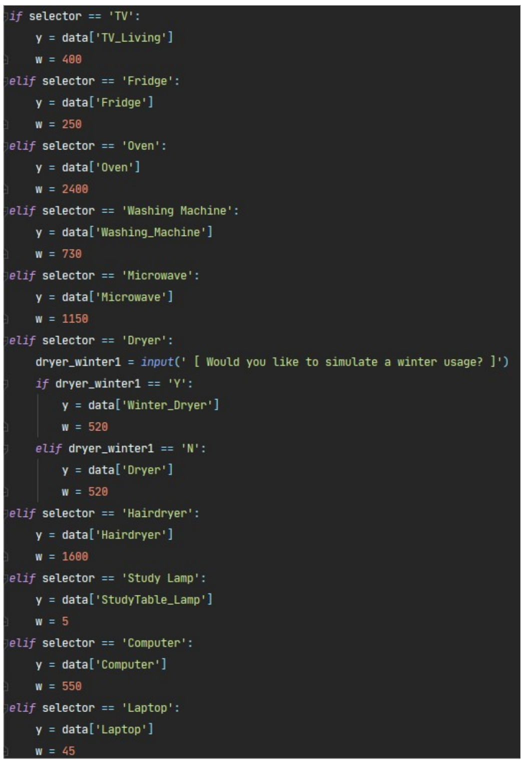
The function returns the input for the variable 'inp' which outputs the user appliances that can be used to simulate. Once an input is given, select is set to equal the variable 'selector' and uses the value of ‘inp’ to specify which appliance the user wants to run, and the data csv named ‘AppliancesUsage.csv’ is loaded along with the house’s resident amount.


```python
# Launch and Select Menu
#
selector = select() #Outputs a selection of Appliances to choose from

# Data Reader
    # If conditions are met from selector, it uses the data referenced as 'y'
    # Note: w is the energy consumption rate of each appliances
    # If the incorrect appliance is inputted, the program starts again
    # data is a simple file opener to which opens the designated .csv file which stores each appliances usage
data = pd.read_csv('AppliancesUsage.csv')
residents = 3
```

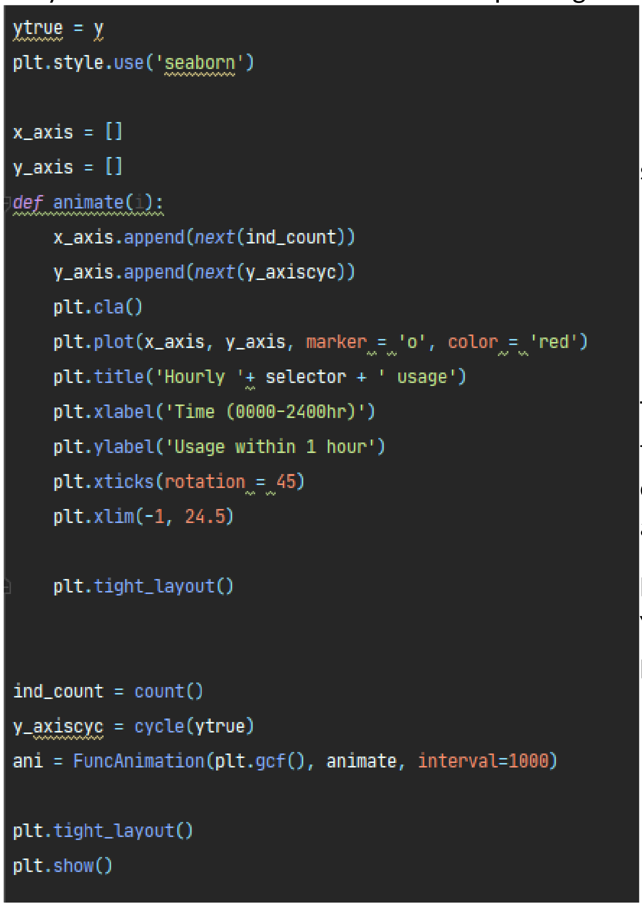
To read in the data, pandas (pd.read_csv) used again for loading in the required data for each appliances and is loaded individually as per user input. (Note also that each appliance is set a power rate as the user chooses an appliance). The program also simulates a winter seasonal usage for the dryer appliance and loads in the data if the user choses to simulate the winter season dryer

As the user inputs the appliance, the data will ‘y’ will then be passed on along ‘w’ or the watt usage to be used later for graphing.

However, if a non-existing appliance is entered, the program will be terminated and will need to be restarted and try again.


```python
if selector == 'TV':
    y = data['TV_Living']
    w = 400
elif selector == 'Fridge':
    y = data['Fridge']
    w = 250
elif selector == 'Oven':
    y = data['Oven']
    w = 2400
elif selector == 'Washing Machine':
    y = data['Washing_Machine']
    w = 730
elif selector == 'Microwave':
    y = data['Microwave']
    w = 1150
elif selector == 'Dryer':
    dryer_winter1 = input(' [ Would you like to simulate a winter usage? ]')
    if dryer_winter1 == 'Y':
        y = data['Winter_Dryer']
        w = 520
    elif dryer_winter1 == 'N':
        y = data['Dryer']
        w = 520
elif selector == 'Hairdryer':
    y = data['Hairdryer']
    w = 1600
elif selector == 'Study Lamp':
    y = data['StudyTable_Lamp']
    w = 5
elif selector == 'Computer':
    y = data['Computer']
    w = 550
elif selector == 'Laptop':
    y = data['Laptop']
    w = 45
```

Both y and w variables will be used for the plotting and summary of the chosen data.


```python
ytrue = y
plt.style.use('seaborn')

x_axis = []
y_axis = []
def animate(i):
    x_axis.append(next(ind_count))
    y_axis.append(next(y_axiscyc))
    plt.cla()
    plt.plot(x_axis, y_axis, marker = 'o', color = 'red')
    plt.title('Hourly '+ selector + ' usage')
    plt.xlabel('Time (0000-2400hr)')
    plt.ylabel('Usage within 1 hour')
    plt.xticks(rotation = 45)
    plt.xlim(-1, 24.5)

    plt.tight_layout()


ind_count = count()
y_axiscyc = cycle(ytrue)
ani = FuncAnimation(plt.gcf(), animate, interval=1000)

plt.tight_layout()
plt.show()
```

The second and most fundamental feature of the program which enables for the simulation of each appliance’s hourly usage is the animate plot.

Based on Cory Schafer and Sentdex’s YouTube tutorial on plotting live data, a live plot has been formulated through the use of matplotlib.pyplt and animation. The funcanimation is the function used to animate the data loaded on to the line graph.

We first create a function called animate(i). The ‘I’ refers to ‘sef’ and is a parameter being passed in by the animation library. Without it the code will not plot properly.

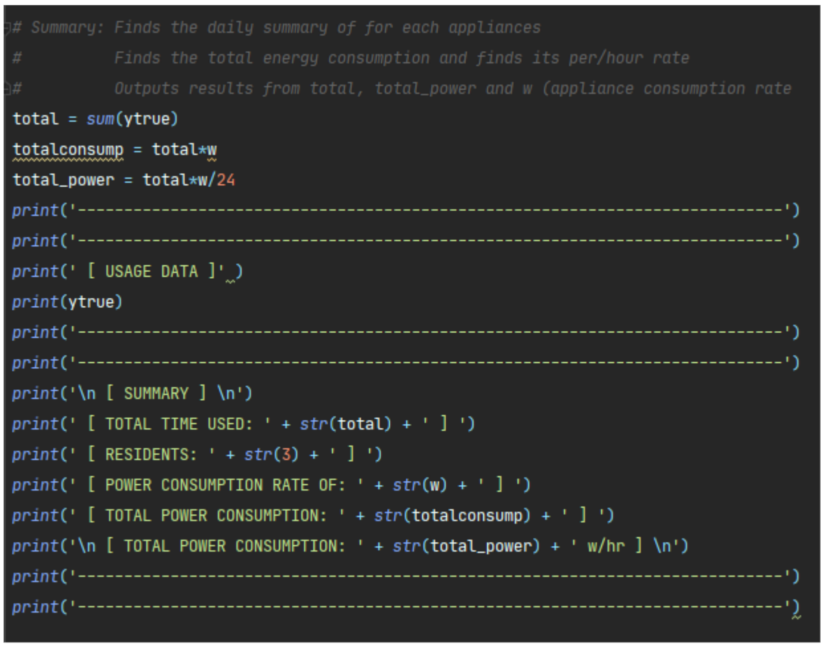
The count() function is used to increase the cycle by 1 every time a y-value is cycled, while the cycle() function is used to cycle through the ‘y’ data as its appended to y_axis [] list.

the ani variable then applies the animate animate function to the def animate(i) function created previously.

A cycle interval of 1000ms cycles through the y data every 1 second.

The summary then calculates the total consumption usage of the chosen appliance by finding the total hours used (total variable) and multiplying it by the appliance’s power rate (in totalconsump variable.

These variables are then passed on and converted as strings and are outputted to the user.



```python
# Summary: Finds the daily summary of for each appliances
#          Finds the total energy consumption and finds its per/hour rate
#          Outputs results from total, total_power and w (appliance consumption rate
total = sum(ytrue)
totalconsump = total*w
total_power = total*w/24
print('---------------------------------------------------------------------------')
print('---------------------------------------------------------------------------')
print(' [ USAGE DATA ]' )
print(ytrue)
print('---------------------------------------------------------------------------')
print('---------------------------------------------------------------------------')
print('\n [ SUMMARY ] \n')
print(' [ TOTAL TIME USED: ' + str(total) + ' ] ')
print(' [ RESIDENTS: ' + str(3) + ' ] ')
print(' [ POWER CONSUMPTION RATE OF: ' + str(w) + ' ] ')
print(' [ TOTAL POWER CONSUMPTION: ' + str(totalconsump) + ' ] ')
print('\n [ TOTAL POWER CONSUMPTION: ' + str(total_power) + ' w/hr ] \n')
print('---------------------------------------------------------------------------')
print('---------------------------------------------------------------------------')
```

PowerSimulatorLauncher.py

Overview

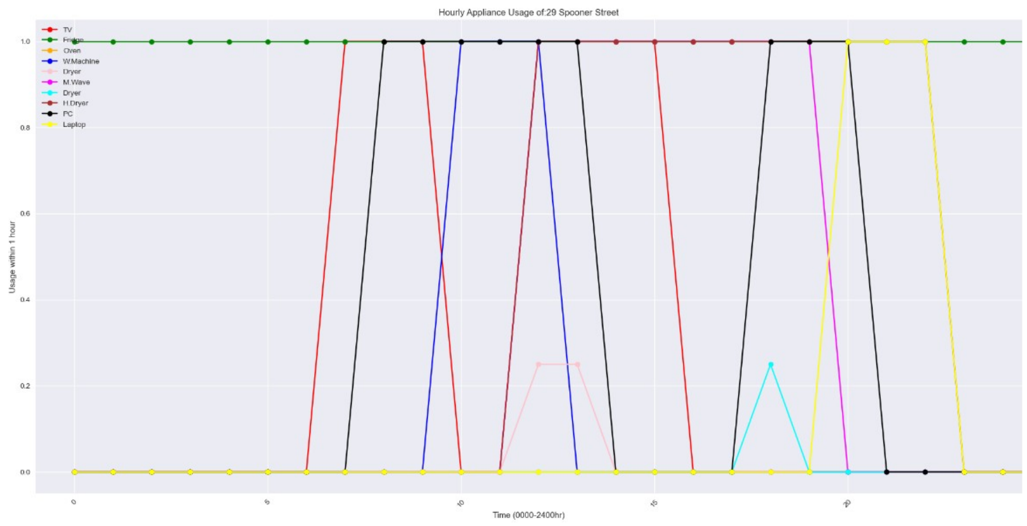
PowerSimulatorLauncher.py is an object-based program used to simulate the hourly appliance usage of a street/collection of houses. It features various functionalities that creates a clear and concise simulation that enables users to see the trends and patterns of the hourly appliance usage of each selected house.

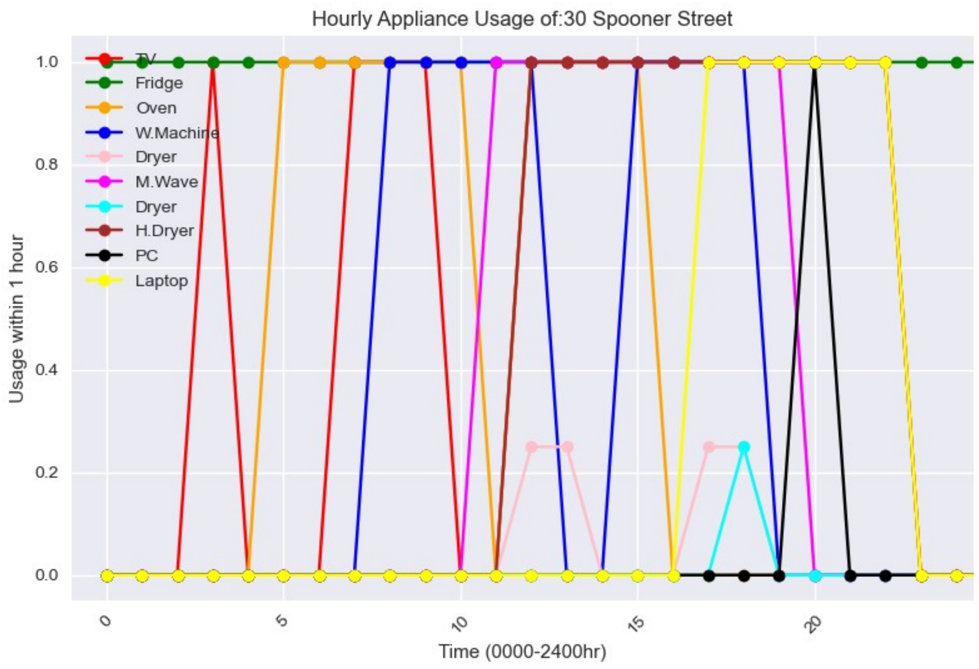
The program features:

- A simple interface that outputs the houses in the street allows for individual selection.
- Detects invalid house inputs

- Simulates the data by creating a live moving plot that simulates the fluctuation of hourly appliance usage.

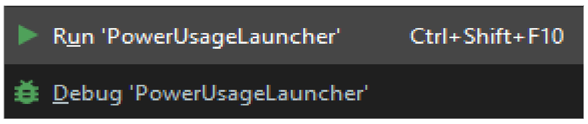
Examples: 29 and 30 Spooner Street Appliance Usage graph (after plot):



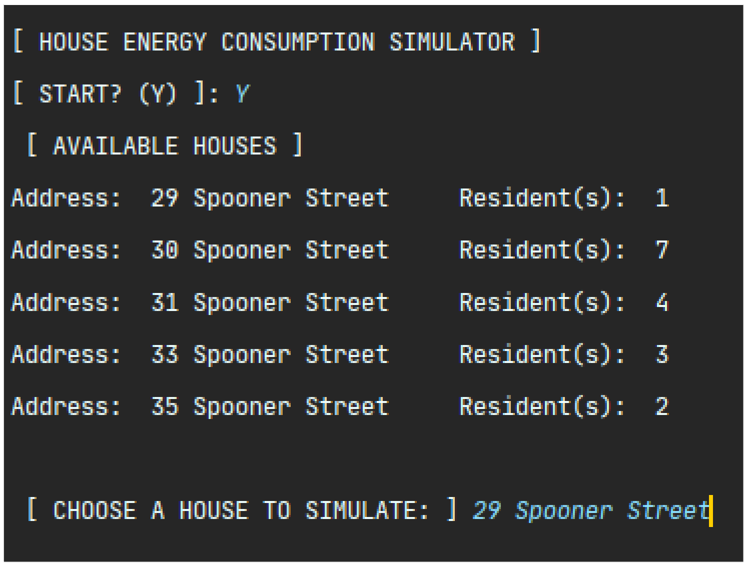


User Guide

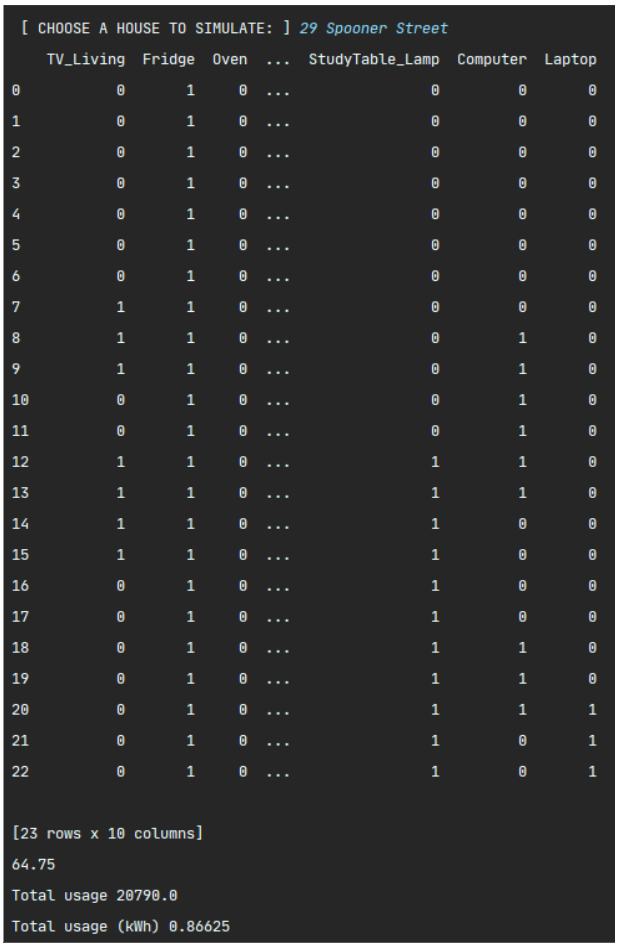
1. To run the program, simply press 'Ctrl-Shft-F10' or right click on the program title and press 'Run'.
   i. If running on linux, the program can be run by inputting 'python3 PowerSimulatorLauncher.py' into the terminal


Run 'PowerUsageLauncher' Ctrl+Shift+F10
Debug 'PowerUsageLauncher'

2. To launch the simulator, input ‘Y’ or ‘y’ on the first prompt, this will then launch a collective of houses in the same street.
3. The program will then prompt a user input, and from the selection of the outputted houses, enter the house address as following:
    a. i.e 29 Spooner Street
4. The program will then start a live plot of the hourly appliances contained in the house, and once finished, can be saved an exited.
5. The program will then plot a summary containing the total hours of each appliance, the total watt and kw/h per hour.
6. The summary will also include the data set used to for the live plot.



```
[ HOUSE ENERGY CONSUMPTION SIMULATOR ]
[ START? (Y) ]: Y
 [ AVAILABLE HOUSES ]
Address:  29 Spooner Street     Resident(s):  1
Address:  30 Spooner Street     Resident(s):  7
Address:  31 Spooner Street     Resident(s):  4
Address:  33 Spooner Street     Resident(s):  3
Address:  35 Spooner Street     Resident(s):  2


 [ CHOOSE A HOUSE TO SIMULATE: ] 29 Spooner Street
```



```
 [ CHOOSE A HOUSE TO SIMULATE: ] 29 Spooner Street
    TV_Living  Fridge  Oven  ...  StudyTable_Lamp  Computer  Laptop
0           0       1     0  ...                0         0       0
1           0       1     0  ...                0         0       0
2           0       1     0  ...                0         0       0
3           0       1     0  ...                0         0       0
4           0       1     0  ...                0         0       0
5           0       1     0  ...                0         0       0
6           0       1     0  ...                0         0       0
7           1       1     0  ...                0         0       0
8           1       1     0  ...                0         1       0
9           1       1     0  ...                0         1       0
10          0       1     0  ...                0         1       0
11          0       1     0  ...                0         1       0
12          1       1     0  ...                1         1       0
13          1       1     0  ...                1         1       0
14          1       1     0  ...                1         0       0
15          1       1     0  ...                1         0       0
16          0       1     0  ...                1         0       0
17          0       1     0  ...                1         0       0
18          0       1     0  ...                1         1       0
19          0       1     0  ...                1         1       0
20          0       1     0  ...                1         1       1
21          0       1     0  ...                1         0       1
22          0       1     0  ...                1         0       1

[23 rows x 10 columns]
64.75
Total usage 20790.0
Total usage (kWh) 0.86625
```

Discussion

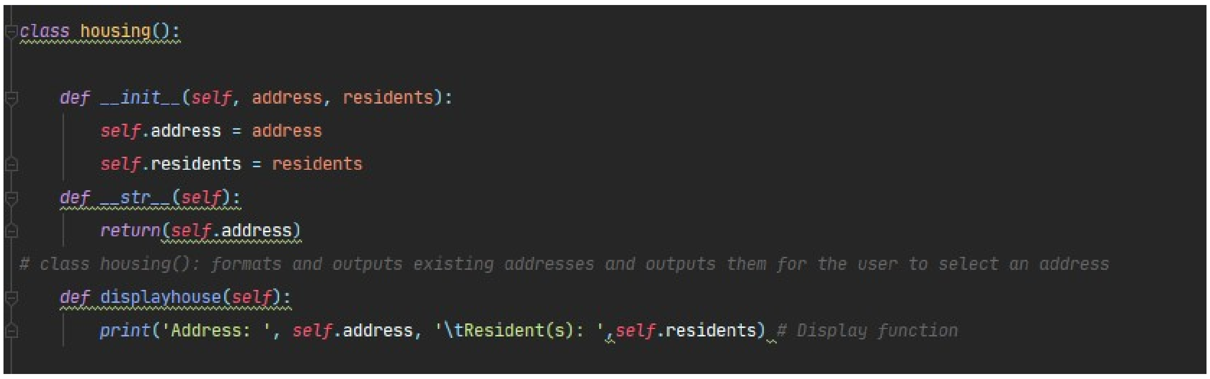
As mentioned previously, PowerSimulatorLauncher is an object-oriented program that features various functionalities.

Usage of object:

Object-oriented programming was used majority for building and outputting the end summary and the mini user interface that will allow the user to choose a house they want to simulate.

The object is contained and imported as 'house_sim.py'

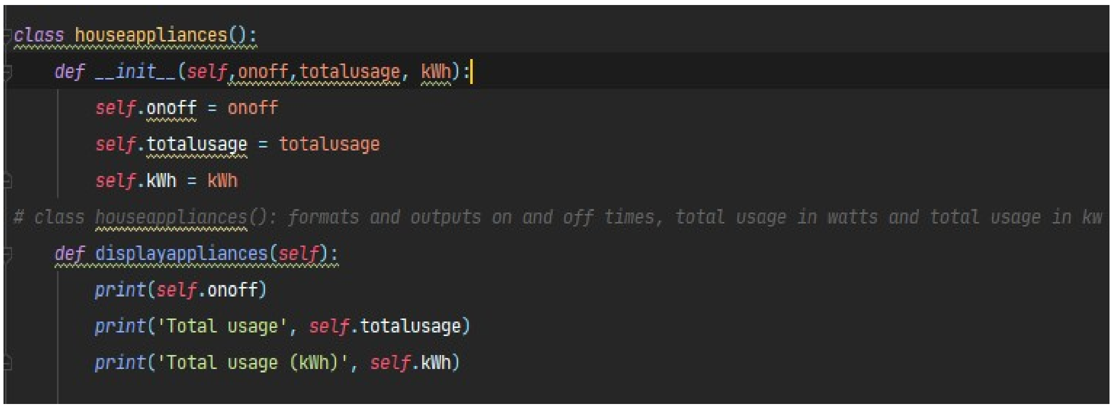
Object one: housing()



```python
class housing():

    def __init__(self, address, residents):
        self.address = address
        self.residents = residents
    def __str__(self):
        return(self.address)
# class housing(): formats and outputs existing addresses and outputs them for the user to select an address
    def displayhouse(self):
        print('Address: ', self.address, '\tResident(s): ',self.residents) # Display function
```

The first objects finds the values of address, residents from the main program file, gathers it and builds an object containing specific house details, def displayhouse(self) is a function within the object that enables the object to be called and printed once the program is started. Below is the house information found in the main program:

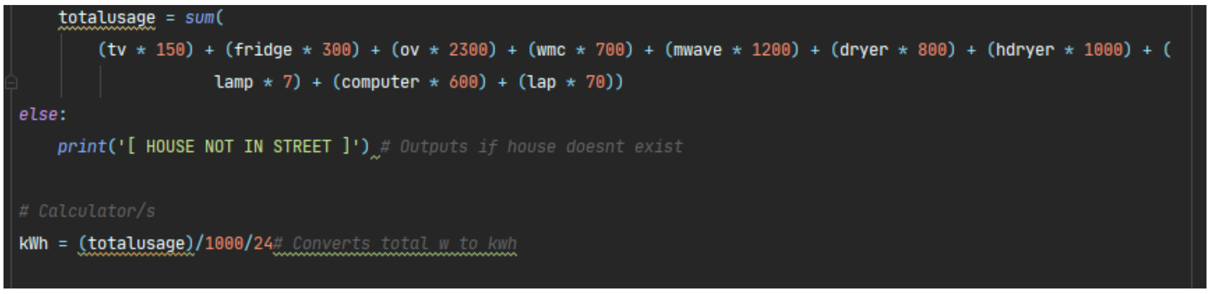
Object two: appliances()



```python
class houseappliances():
    def __init__(self,onoff,totalusage, kWh):
        self.onoff = onoff
        self.totalusage = totalusage
        self.kWh = kWh
# class houseappliances(): formats and outputs on and off times, total usage in watts and total usage in kw
    def displayappliances(self):
        print(self.onoff)
        print('Total usage', self.totalusage)
        print('Total usage (kWh)', self.kWh)
```

The math done in the summary part of the program is stored within the object, for later outputted. The math needed for the object is also found in the main program and is the contents of the object is plotted by def displayappliances(self) function in the object.

The math needed is below is shown:

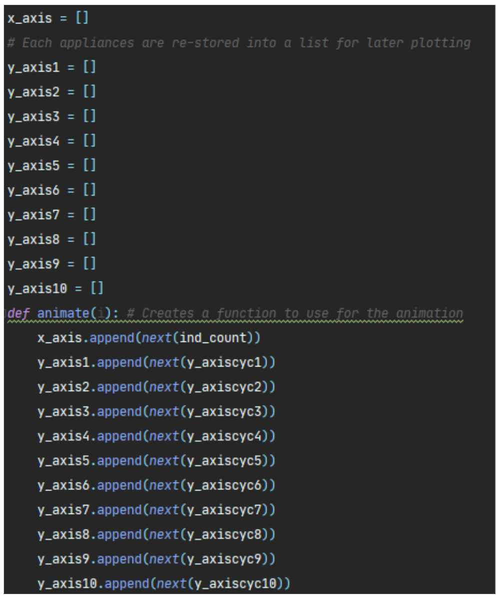


```
    totalusage = sum(
        (tv * 150) + (fridge * 300) + (ov * 2300) + (wmc * 700) + (mwave * 1200) + (dryer * 800) + (hdryer * 1000) + (
                lamp * 7) + (computer * 600) + (lap * 70))
else:
    print('[ HOUSE NOT IN STREET ]') # Outputs if house doesnt exist

# Calculator/s
kWh = (totalusage)/1000/24# Converts total w to kwh
```

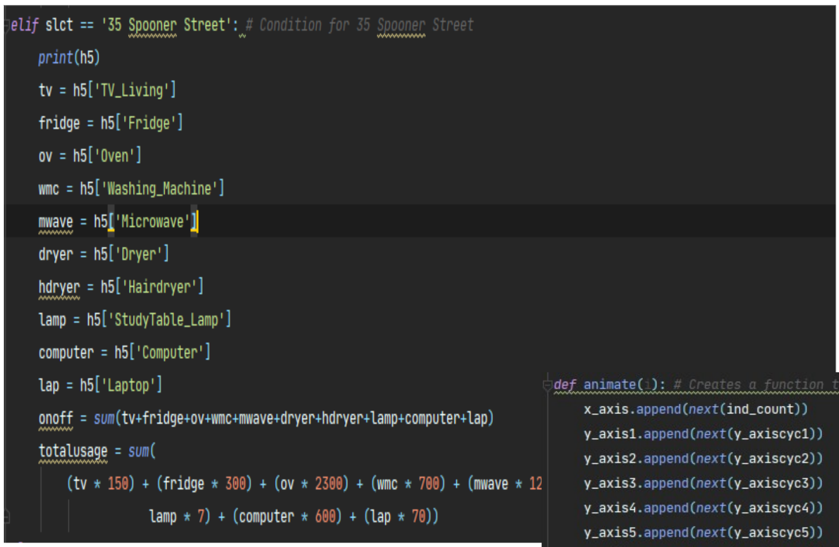
The totalusage variable contains the sum the hourly usage of each appliances, multiplied by their respective power rate and added together to find the grand total power consumption of the day. kwH variable then converts this total in to kw/h per hour. The math done here is stored in the houseappliancesobjects().

PowerSimulatorLauncher, like wise with the second program PowerModelLauncher also features a similar live plot, but however plots all appliances within one house and not individually. Likewise, the y values loaded in are cycled using cycle () and count() is used to count through every value that is being outputted in the plot.

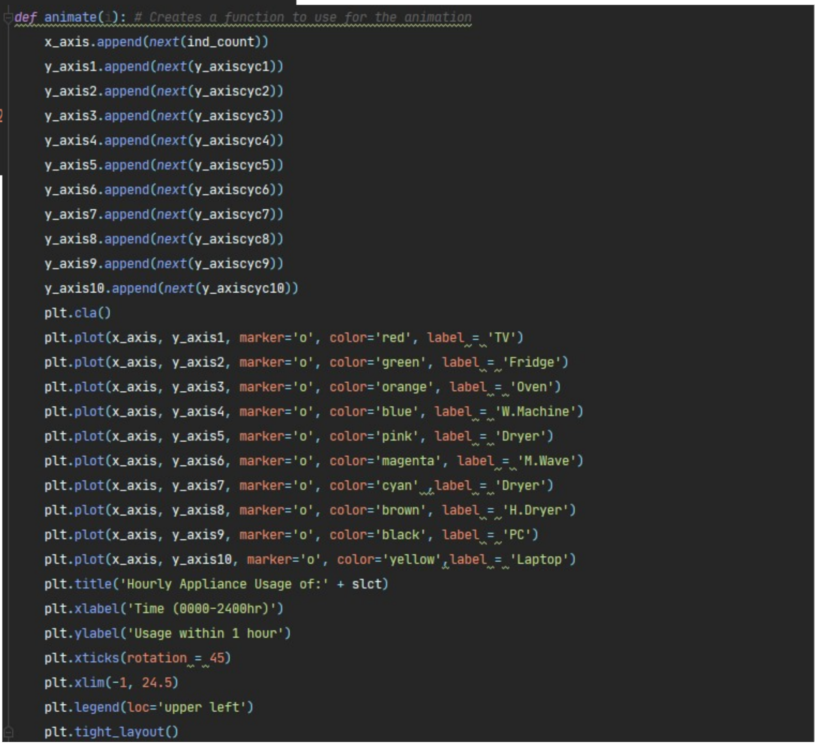
Pandas pd are again used to load in the .csv files as data frame for when loading each appliance and are appended in the empty list for later plotting.



```
x_axis = []
# Each appliances are re-stored into a list for later plotting
y_axis1 = []
y_axis2 = []
y_axis3 = []
y_axis4 = []
y_axis5 = []
y_axis6 = []
y_axis7 = []
y_axis8 = []
y_axis9 = []
y_axis10 = []
def animate(i): # Creates a function to use for the animation
    x_axis.append(next(ind_count))
    y_axis1.append(next(y_axiscyc1))
    y_axis2.append(next(y_axiscyc2))
    y_axis3.append(next(y_axiscyc3))
    y_axis4.append(next(y_axiscyc4))
    y_axis5.append(next(y_axiscyc5))
    y_axis6.append(next(y_axiscyc6))
    y_axis7.append(next(y_axiscyc7))
    y_axis8.append(next(y_axiscyc8))
    y_axis9.append(next(y_axiscyc9))
    y_axis10.append(next(y_axiscyc10))
```



```
elif slct == '35 Spooner Street': # Condition for 35 Spooner Street
    print(h5)
    tv = h5['TV_Living']
    fridge = h5['Fridge']
    ov = h5['Oven']
    wmc = h5['Washing_Machine']
    mwave = h5['Microwave']
    dryer = h5['Dryer']
    hdryer = h5['Hairdryer']
    lamp = h5['StudyTable_Lamp']
    computer = h5['Computer']
    lap = h5['Laptop']
    onoff = sum(tv+fridge+ov+wmc+mwave+dryer+hdryer+lamp+computer+lap)
    totalusage = sum(
        (tv * 150) + (fridge * 300) + (ov * 2300) + (wmc * 700) + (mwave * 12
                lamp * 7) + (computer * 600) + (lap * 70))
```



```
def animate(i): # Creates a function to use for the animation
    x_axis.append(next(ind_count))
    y_axis1.append(next(y_axiscyc1))
    y_axis2.append(next(y_axiscyc2))
    y_axis3.append(next(y_axiscyc3))
    y_axis4.append(next(y_axiscyc4))
    y_axis5.append(next(y_axiscyc5))
    y_axis6.append(next(y_axiscyc6))
    y_axis7.append(next(y_axiscyc7))
    y_axis8.append(next(y_axiscyc8))
    y_axis9.append(next(y_axiscyc9))
    y_axis10.append(next(y_axiscyc10))
    plt.cla()
    plt.plot(x_axis, y_axis1, marker='o', color='red', label = 'TV')
    plt.plot(x_axis, y_axis2, marker='o', color='green', label = 'Fridge')
    plt.plot(x_axis, y_axis3, marker='o', color='orange', label = 'Oven')
    plt.plot(x_axis, y_axis4, marker='o', color='blue', label = 'W.Machine')
    plt.plot(x_axis, y_axis5, marker='o', color='pink', label = 'Dryer')
    plt.plot(x_axis, y_axis6, marker='o', color='magenta', label = 'M.Wave')
    plt.plot(x_axis, y_axis7, marker='o', color='cyan', label = 'Dryer')
    plt.plot(x_axis, y_axis8, marker='o', color='brown', label = 'H.Dryer')
    plt.plot(x_axis, y_axis9, marker='o', color='black', label = 'PC')
    plt.plot(x_axis, y_axis10, marker='o', color='yellow',label = 'Laptop')
    plt.title('Hourly Appliance Usage of:' + slct)
    plt.xlabel('Time (0000-2400hr)')
    plt.ylabel('Usage within 1 hour')
    plt.xticks(rotation = 45)
    plt.xlim(-1, 24.5)
    plt.legend(loc='upper left')
    plt.tight_layout()
```

The live plot is also based on Corey Schafer and Sentdex's live plotting

YouTube tutorial likewise with plot 2 (please find references in references section). However, it plots each loaded appliance and their hourly usages represented as 1 (on/1hour) and 0 as off

Student Notes (Extended Ver.)

- The inspiration for creating a live plot can be referenced to various educational programming videos located into YouTube, which teaches how to create moving line graphs:
  https://www.youtube.com/watch?v=Ercd-Ip5PfQ (Corey Shafer)
  https://www.youtube.com/watch?v=ZmYPzESC5YY (sentdex)
- The usage of pandas library was to ease the manipulation and loading of data from a csv.
  https://www.youtube.com/watch?v=AFnUhiRXXWk (OSPY)
  https://www.youtube.com/watch?v=0P7QnIQDBJY&t=2m22s (Keith Galli
- Object Orienteering, creating loops and graph styles were based off of the COMP1005 unit materials and lectures.
- The usage of different functions was read from the user documentations of each modules such as matplotlib, and pandas

Please note that a shorter, and more specific version these student notes can be found on the Project Investigation file.

References

Galli, Keith. 2019. https://www.youtube.com/watch?v=0P7QnIQDBJY&t=2m22s.

Maxville, Valerie. 2020. *LECTURE 3 ARRAYS AND PLOTS*. Ebook. Curtin University.

Maxville, Valerie. 2020. *LECTURE 4 MULTI-DIMENSIONAL ARRAYS*. Ebook. 1st ed. Curtin University.

Maxville, Valerie. 2020. *LECTURE 6 MODELLING THE WORLD WITH OBJECTS*. Ebook. 1st ed. Curtin University.

Maxville, Valerie. 2020. *LECTURE 7 OBJECT RELATIONSHIPS*. Ebook.

OSPY, Username. 2017. https://www.youtube.com/watch?v=AFnUhiRXXWk.

sentdex, username:. 2015. https://www.youtube.com/watch?v=ZmYPzESC5YY.

Shafer, Corey. 2019. https://www.youtube.com/watch?v=Ercd-Ip5PfQ.

McKinney, Wes. 2020. *Pandas User Dock*. Ebook. https://pandas.pydata.org/docs/pandas.pdf.

"OS Module In Python With Examples - Geeksforgeeks". 2020. *Geeksforgeeks*. https://www.geeksforgeeks.org/os-module-python-examples/.

"Python | Read Csv Using Pandas.Read_Csv() - Geeksforgeeks". 2020. *Geeksforgeeks*. https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/#:~:text=Code%20%231%20%3A%20read_csv%20is%20an,(%20%22filename.csv%22%20).

VanderPlas, Jake. 2020. "Matplotlib Animation Tutorial | Pythonic Perambulations". *Jakevdp.Github.Io*. https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/.