Investigation 3 Report - Suburb Simulation

Abstract

Simulations and parameter sweeps can quickly give insights and compare many possible outcomes. By combining the power model and power usage from Investigations 1 and 2, a simulated look at energy usage for a suburb can be simulated. Using basic object orientation and inheritance, two class models can be created to simulate a suburb; house class and suburb class. Once these classes are made, a driver program given system arguments can test the layout of many different houses. Sub classes for different types can be constructed by using sample files from investigation 2. By running simulations we can compare which combination of house types produce different levels of enery consumption and find different times when energy usage peaks.

Background

The house objects includes 4 sub-classes. Mansion, Family, Flat and Studio. The house.py class model file uses the principle of object orientation; inheritence to re-use fields and methods from the base class. Each subclass has its own appliance csv file eg: a list of appliances. The reason to take these parameters as each house type will have different powerusage trends, and by running a simulation, an optimal ratio or quantity of houses can be derived.

Methodology

The method for performing this simulation is to create multiple python class programs, use some sample files for each house, and then run a simulation using a parameter sweep

```
* house.py            : house model class
* suburb.py           : suburb model class
* housesimulation.py  : run a simulation of a suburb from command line arguments
* sweepHouses.sh      : parameter sweep bash script
* family.csv          : appliance file for family house object
* flat.csv            : appliance file for flat house object
* mansion.csv         : appliance file for mansion house object
* studio.csv          : appliance file for studio house object
```

- House base class attributes

```
class House():
    def __init__(self, address, housetype, postcode, numResidents, numBeds,
                                                    numBaths, appliancefile):
        self.housetype = housetype
        self.address = address
```

```
        self.postcode = postcode
        self.numResidents = numResidents
        self.numBeds = numBeds
        self.numBaths = numBaths

    # list of appliances include name, wattage and a 24hr usage record
        self.appliances = self.readAppliance(appliancefile)
```

Includes the functions: + readAppliance(filename) + getApplianceNames() + getUsage() + getNumResidents() + calcTotalUsage() + calcDailyUsage() + calcUsageAtTime(hour) + printit()

By inheritance, sub classes can be created: eg Mansion. Here you can see that default attributes can be assigned and passed to the base class, allowing the above functions to be reused.

```
class Mansion(House):
    housetype = "Mansion"
    numResidents = 3
    numBeds = 8
    numBaths = 4
    appliancefile = "mansion.csv"

    def __init__(self, address, postcode):
        super().__init__(address, self.housetype, postcode, self.numResidents,
                                self.numBeds, self.numBaths, self.appliancefile)
```

- Suburb class uses an if - elif statement to check the type of house, and construct that subclass accordingly. This allows our driver code housesimulation.py to automate and control suburb creation. The suburb can then tally the total usage of each house.

```
class Suburb():
    def newHouse(self, houseType, address):
        temp = None
        if houseType == 'Mansion':
            temp = Mansion(address, self.postcode)
        elif houseType == "Flat":
            temp = Flat(address, self.postcode)
        elif houseType == "Family":
            temp = Family(address, self.postcode)
        elif houseType == "Studio":
            temp = Studio(address, self.postcode)
        else:
            print("Error processing house -> '" + houseType + "' (please
                                                double check values)")
        if temp:
```

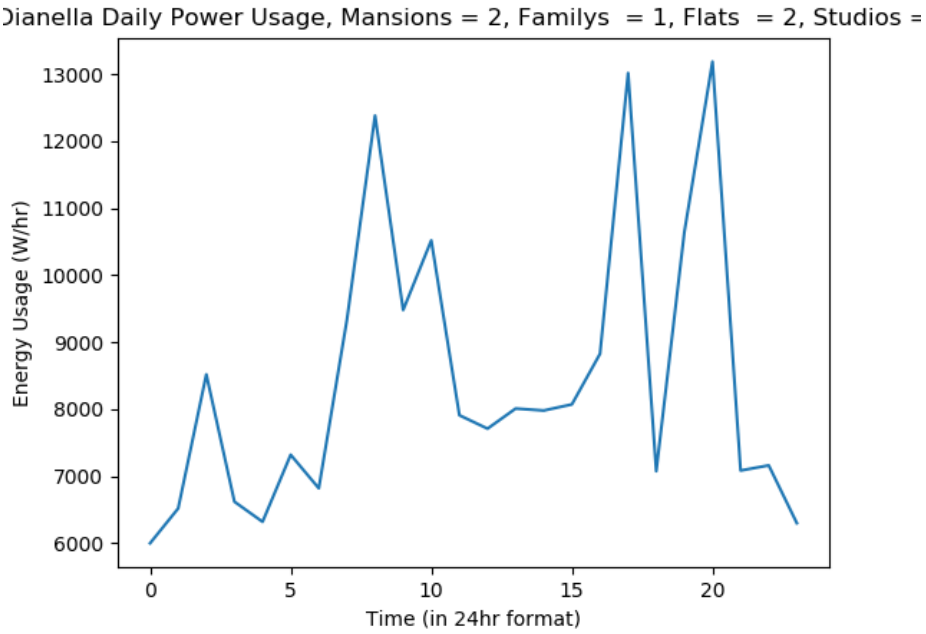
```
            self.houses.append(temp)
            print("Added " + houseType + ": " + address + " to Suburb -
                   " + self.name + " (" + str(self.postcode) + ")")
```

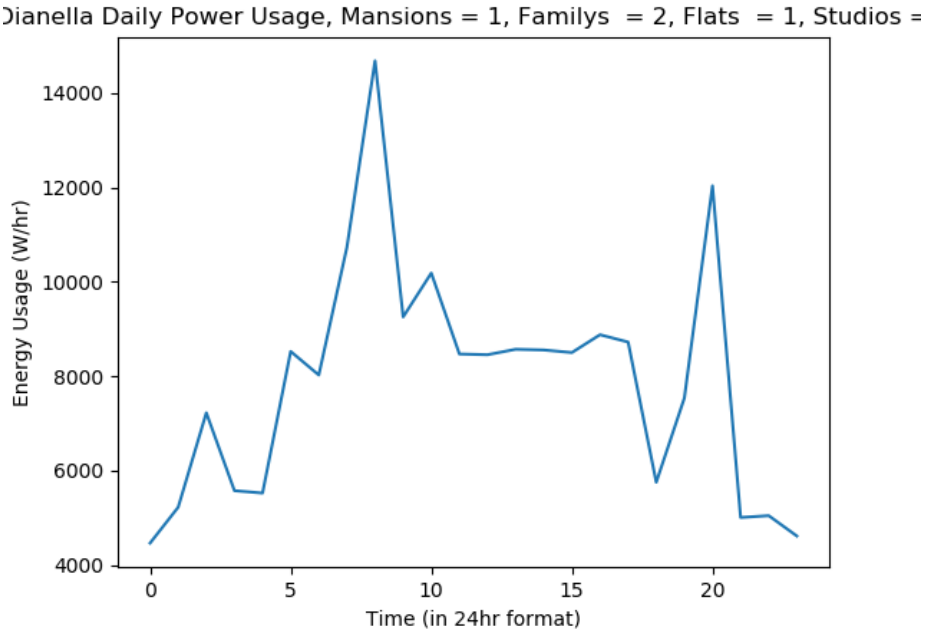
Results

Output from running simulation

```
$ bash sweepHouses.sh Dianella 6059 1 2 1 1 2 1 1 2 1 1 2 1

####### HOUSE SIMULATION SWEEP #######
simhouses2020-06-05_12-48-16

Parameters are:
mansions:  1 2 1
family homes:  1 2 1
flats:  1 2 1
studios:  1 2 1

Running experiments
Experiment:  1 1 1 1
Experiment:  1 1 1 2
Experiment:  1 1 2 1
Experiment:  1 1 2 2
Experiment:  1 2 1 1
Experiment:  1 2 1 2
Experiment:  1 2 2 1
Experiment:  1 2 2 2
Experiment:  2 1 1 1
Experiment:  2 1 1 2
Experiment:  2 1 2 1
Experiment:  2 1 2 2
Experiment:  2 2 1 1
Experiment:  2 2 1 2
Experiment:  2 2 2 1
Experiment:  2 2 2 2

.txt OUTPUT FILE

### CREATING SUBURB
Added Mansion: 1 Copper Street to Suburb - Dianella (6059)
Added Mansion: 2 Copper Street to Suburb - Dianella (6059)
Added Family: 3 Copper Street to Suburb - Dianella (6059)
Added Flat: 4 Copper Street to Suburb - Dianella (6059)
Added Flat: 5 Copper Street to Suburb - Dianella (6059)
```

```
Added Studio: 6 Copper Street to Suburb - Dianella (6059)
### Calculating Power Usage ###
Daily Power Usage (Total) (w/hour): 202828.19999999987
Power Usage at Midday (w/hour): 1220.0
Number of Residents: 15
Average daily power usage per resident: 13521.879999999992w
```





Conclusion and Further Work

The conclusion to be made is that variations in the types of houses have impacts on total energy consumption of a Suburb. One recommendation for further work is to have a way to sort or filter the results that were simulated. For example a filter could be used by ranking the arrangements by average power usage per resident, number of residents, total power usage.

References

Curtin University (2020) "Lecture 10 Scripts and Automation" Fundamentals Of Programming COMP1005 Curtin University (2020) "Lecture 7 Object Relationships" Fundamentals Of Programming COMP1005 *DEVHINTS (2020) "Bash Scripting Cheatsheet" https://devhints.io/bash, accessed 01/06/2020