Final-Copy1

Julius C. F. Schulz
(Dated: November 6, 2019)

I. ABSTRACT

II. BACKGROUND

A. Introduction

1. *The World Wide Web and Economic Prosperity*

Since the inception of the first home computers in 1977 (REF)[?] , modern society has become utterly dependent on and indeed, inexorably bound to digital technology. The rapid and widespread adoption of computational technology has led to the fastest rate of societal and economic development our species has ever experienced. One need only consider that between the years of X and Y, #### number of people gained access to the WWW [mention benefits of WWW for end users?], to appreciate the magnitude of this digital technology. Such rapid and widespread adoption has created a seemingly insatiable demand for __.

Indeed, our quest for exponentially greater computational power and digital storage capacity has led to a new and utterly ubiquitous technological paradigm; Cloud Computing (REF), defined as; The practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer (https://www.dictionary.com/browse/cloud-computing).

As with the original WWW, Cloud Computing itself acts as a facilitator for new technologies. One such example being the Internet of Things (IoT) paradigm. The Internet of Things can be surmised as the extension of the Internet and the Web into the physical realm, by means of the widespread deployment of spatially distributed devices with embedded identification, sensing and/or actuation capabilities [@article{miorandi2012}]. The Internet of Things (IoT) paradigm enables physical devices to connect and exchange information, and also allows objects to be sensed or controlled remotely through the internet (@article{huang2018} @ 1 of @article{huang2018}) The IoT paradigm enables physical devices to connect and exchange information. IoT devices allow objects to be sensed or controlled remotely through the Internet [@article{atzori2010}]. IoT thus represents a convergence of real-world objects and digital objects into seamless (seamless?) and unified cyber-physical system.

Considering again the bigger picture of societal benefit, Figure X, shows that the benefits of technological deployment have been widely adopted and exploited. However, the success of modern human society is not without consequence. Paradoxically, all of the benefits our society has enjoyed from the development, production and deployment of technology, has required vast amounts of energy. This energy has, since the industrial revolution, primarily been derived from the burning of fossil fuels (REF). The International Panel on Climate Change (IPCC) estimates that ABC by XYZ if we do not do blah (REF). As exemplified by figure Y (co-inciding to the time period of figure X) global greenhouse gas emissions are steadily increasing (REF). At the current rate, blah et al estimate that in 20##, humans will be living in an atmospheric conditions not experienced since #### the mezozoic era, XX years before the evolution of modern humans (REF, REF).

It is therefore imperative moving forward as a species that all steps are be taken to mitigate the emission of greenhouse gases and abate the advance of anthropomorphic climate change. The scale of the challenge is such that technology itself will prove critical in effectively combatting this extential threat to civilisation. When considering energy consumption, the industrial sector (including the non-combusted use of fuels) currently consumes around half of all global energy and feedstock fuels, with residential and commercial buildings (29%) and transport (21%) accounting for the remainder (https://www.bp.com/en/global/corporate/energy-economics/energy-outlook/demand-by-sector.html).

With residential and commercial buildings accounting for 29% of energy demand globally (REF). A qualitative assessment of figures X Y and Z show that our increased energy consumption has coincided with our staggerings global acheivements, including lifting millions out of poverty blah blah blah. We have more development we use more energy, but we have more development we are better off in all ways. We ALSO have more internet connectivity.

It therefore stands that the key to continues human prosperity is to de-couple energy demand growth from economic growth. Clearly economic

prosperity is a direct contributing factor to human prosperity in general. One such avenue for reduced energy consumption is the optimization of existing services, utilities, and so on.

residential and commercial energy consumption

The convergence of

explosive growth of our population

https://ourworldindata.org/energy-production-and-changing-energy-sources https://www.iea.org/statistics/kwes/consumption/

B. Datasets

III. Data Preprocessing and Visualisation

A. Importing & Preprocessing the Activities Meta Data

The dataset `S1Activities.csv` was imported into the Jupyter interactive environment. These data contains a tabulated summary of Heading, Category, Subcategory and a corresponding code. After importation, the dataset has dimensionality of [3, 33], with `Heading`, `Category` & `Subcategory` present as non-null objects. The attribute `Code` (which codefies the unique set of Heading, Category) was imported as an index value. At this time, the activities data will not be subject to preprocessing.

[22]:
```
                        Heading                         ␣
↪Category           Subcategory
Code
1      Employment related  Employment␣
↪work at home         Work at home
5      Employment related          Travel␣
↪employment  Going out to work
10          Personal needs                          ␣
↪  Eating                     Eating
15          Personal needs             ␣
↪Personal hygiene            Toileting
20          Personal needs             ␣
↪Personal hygiene              Bathing
```

B. Importing & Preprocessing the Sensor Meta Data

The dataset `S1sensors.csv` was imported into the Jupyter interactive environment. These data contains a tabulated values for Sensor ID, Room and Sensor Activity Type, with no header row present in the original dataset. After importation, the dataset has dimensionality of [3, 76], with header `0`, `1` & `2` corresponding to `SensorID`, `Room` & `Sensor Activity Type`, respectively. All attributes are nominal, and were imported as dtype str, accordingly. Note that it can be immediately seen (`dsS1Sensors.head()`) that attribute `1` & `2` contain degenerate values. This will be addressed in the subsequent data preprocessing.

The preprocessing of the sensor data is a critical step in our analysis. Careful consideration of the data, including the presence of duplicates. This is because if we dont have a sufficient understanding of where and why duplicates exist, we will not be able to satisfactorarily preprocess them. Failure to do so we mean that there is potentail degeneracy in our source dataset, leading to unknown issues with our downstream analysis.

[23]:
```
     0         1               2
0  100  Bathroom  Toilet Flush
1  101  Bathroom   Light switch
2  104     Foyer   Light switch
3  105   Kitchen   Light switch
4  106   Kitchen         Burner
```

Column [1] & Column [2] of the sensor data will be concatenated, whitespace will be removed, all text will be cast to lowercase and a final whitespace strip will be performed. The python script `S1sensorsPreprocessing.py` is run perform several preprocessing steps in these data. The script concatenates the attributes `dsS1Sensors[1]` and `dsS1Sensors[2]`, with an underscore. Whitespace is then stripped and all string values are coerced to lowercase. This newly created attribute is then added to the dataframe, as seen below (REF). Additionally, the attributes `0`, `1` & `2` are renamed `subActNum`, `room` & `activity`, respectively. - Data types - IF a sub-act requires electricity

[25]:
```
  subActNum       room        activity      ␣
↪               concat
0        100   Bathroom  Toilet Flush  ␣
↪bathroom_toiletflush
1        101   Bathroom   Light switch ␣
↪bathroom_lightswitch
2        104      Foyer   Light switch     ␣
↪foyer_lightswitch
3        105    Kitchen   Light switch  ␣
↪kitchen_lightswitch
4        106    Kitchen         Burner      ␣
↪   kitchen_burner
```

The function `getUniqueValues.py` is invoked to provide a means of capturing a list of unique values in a given column of a dataset. The newly created `dsS1Sensors` is then checked for duplicates in two attributes, `subActNum` & `concat`. The number of unique values in `dsS1Sensors.subActNum` is found to be 76, demonstrating that this attribute

contains a set of completely unique values (recall n(rows) = 76). The number of unique values in `dsS1Sensors.concat` is found to be 41, demonstrating that despite the concatenation methodology, their are still duplicate values in the dataframe. These duplicate values warrant further investigation. The function `seen_dupes_dsS1Sensors.py` is used to list counts with their corresponding values.

[9]:
```
76
```

[10]:
```
41
```

```
3   kitchen_lightswitch
4   kitchen_burner
2   livingroom_lightswitch
7   kitchen_drawer
3   kitchen_refrigerator
15  kitchen_cabinet
2   kitchen_door
5   bedroom_drawer
2   bathroom_medicinecabinet
2   bathroom_cabinet
```

Upon compilation of the above summary list, and with reference to the original work is was determined that these values result from multiple sensors with extremely similar functionality. For example, kitchen_burner has a value of n=4 - this is because on the burner in the apartment under investigation, there were 4 individual burners present. Similarly, kitchen_cabinet has a value of n=15, indicating that for the various cabinets in the apartment, each were given sensors. On the one-hand, this level of granularity may provide fertile grounds for advanced analysis, HOWEVER, for the purposes of this research project, such values will serve to increase the dimensionality of the overall dataset. High dimensionality can lead to difficulties with plotting, ... ML (REF) and thus IN A SUBSEQUENT PREPROCESSING exercise these values will be collapsed down to have n=1.

Creation of JSON Catalogues PRIOR to dup removal - why? Because even if a key-value pair cannot be matched it will simply be ignored **Prior to dupe removal** As this work is largely concerned with energy usage in the home, the sub-activities will be categorized based on their energy requirement. That is, if a sub-activity requires an input of energy beyond what the end user alone can provide, it will be classified as `energyReq` = true. Whereas, if a sub-activity is able to be performed through only interaction with the end user, it will be classified as `energyReq` = false. By way of example, the sub-activity `bathroom_toiletflush` will have an `energyReq` equal to false, while the sub-activity `bathroom_lightswitch` will have an `energyReq` equal to true. Each row (n=76) needs to be inspected manually to determine if the activity requires electricity.

Function `reqEnergy_containSpecialChar.py` is run - After visual inspection, uses `reqEnergy = 'ligh|burn|mach|toas|freez|dvd|lamp|washer|dry|exh|disp|f` to find subActivities which require energy input beyond that of the end user - Checked below for SPECIAL CHAR - Special CHARs removed

[13]:
```
    subActNum          room      activity␣
↪                     concat  reqEnergy  \
58         82  Office/study        Drawer␣
↪       office/study_drawer      False
68         92  Office/study  Light switch␣
↪ office/study_lightswitch       True

    specialChar
58         True
68         True
```

C. Importing & Preprocessing the Activities Data

The activities data set `S1activities.csv` will be imported, evaluated and cleaned. The goal of the pre-processing will be to restructure the dataset into a 'tidy' format, that is, where the attributes are columns, the rows are instances, and each cell contains only one value. Given that the data is time-series, timestamps will be used as indexes. The data will also be cast into continuous 24 hour segments, with timestamps in the form `YYYY-MM-DD hh:mm:ss` (using datetime data type)

MENTION THIS FROM Huang et al.

We aim to perform the preprocessing in such a way that is * minimally computationally intensive * reproducible / traceable code * reasonable checks (validation)

Example of ds when opened in Microsoft excel.

The activities data was imported into an indexed dataframe, containing only one column, with 1475 rows, with all values comma-separated (per row). This style of import had to be used, due to the varying number of comma-separated elements in each row (as seen in figure X, above).

- A array of strings, instead of an array of arrays Analysis of the original dataset, and exploration during pre-processing to this point, shows us that the original dataset follows a structure such that each 5 rows of data contains is one discrete set of data. In this structure,
- Row 1 = Activity, Date, Start Time, End Time
- Row 2 = Sub-activity (an action that can be executed as part of performing the activity) code-values

- Row 3 = Sub-activity descriptive values
- Row 4 = Sub-activity start time
- Row 5 = Sub-activity end time In order to access the values programmatically, we will now turn the 1D array list back into a 2D array list, where each element array[i] contains the 5 rows of information, as described above.

The activities data set `S1activities.csv` will be imported, evaluated and cleaned. The goal of the pre-processing will be to restructure the dataset into a 'tidy' format, that is, where the attributes are columns, the rows are instances, and each cell contains only one value. Given that the data is time-series, timestamps will be used as indexes. The data will also be cast into continuous 24 hour segments, with timestamps in the form `YYYY-MM-DD hh:mm:ss` (using datetime data type)

We aim to perform the preprocessing in such a way that is * minimally computationally intensive * reproducible / traceable code * reasonable checks (validation) The activities data was imported into an indexed dataframe, containing only one column, with 1475 rows, with all values comma-separated (per row). This style of import had to be used, due to the varying number of comma-separated elements in each row (as seen in figure X, above). * A array of strings, instead of an array of arrays Analysis of the original dataset, and exploration during pre-processing to this point, shows us that the original dataset follows a structure such that each 5 rows of data contains is one discrete set of data. In this structure, * Row 1 = Activity, Date, Start Time, End Time * Row 2 = Sub-activity (an action that can be executed as part of performing the activity) code-values * Row 3 = Sub-activity descriptive values * Row 4 = Sub-activity start time * Row 5 = Sub-activity end time In order to access the values programmatically, we will now turn the 1D array list back into a 2D array list, where each element array[i] contains the 5 rows of information, as described above.

[23]:
```
             activity      date startTime␣
↪  endTime
0             Bathing  4/1/2003  20:41:35␣
↪ 21:32:50
1          Toileting  4/1/2003  17:30:36␣
↪ 17:46:41
2          Toileting  4/1/2003   18:4:43␣
↪  18:18:2
3          Toileting  4/1/2003   11:52:1␣
↪ 11:58:50
4  Going out to work  4/1/2003  12:11:26␣
↪ 12:15:12
```

Extracting the Activity, Time and Data If we run `a[0][0]`, we get `'Bathing,4/1/2003,20:41:35,21:32:50'` * A string with 4 comma-separated 'items' * We won't check the entire DF, rather, will rely on errors thrown back to confirm validity ('validation of the method')

Constructing the while loop * Create empty lists * Extracts the relevant elements * Adds (appends) the elements to the lists

Sanity Checks * We won't check the entire DF, rather, will rely on errors thrown back to confirm validity ('validation of the method') * Divisible by 5

[24]:
```
          activity                start␣
↪                end
0          Bathing 2003-04-01 20:41:35␣
↪2003-04-01 21:32:50
1       Toileting 2003-04-01 17:30:36␣
↪2003-04-01 17:46:41
2       Toileting 2003-04-01 18:04:43␣
↪2003-04-01 18:18:02
3       Toileting 2003-04-01 11:52:01␣
↪2003-04-01 11:58:50
4  Going out to work 2003-04-01 12:11:26␣
↪2003-04-01 12:15:12
```

a. Aggregated Box Plot MENTION THIS FROM Huang et al.

b. Strip Plots Text
MENTION THIS FROM Huang et al.

Bathroom Exhaust Fan Text
MENTION THIS FROM Huang et al.

Kitchen Garbage Disposal Text
MENTION THIS FROM Huang et al.

Bathroom Toiletflush Text - ADD LINE SEGMENT CHART

D. Sankey Diagrams - Qualitative Assessment

[68]:
```
<IPython.lib.display.IFrame at␣
↪0x12a26e210>
```

[69]:
```
<IPython.lib.display.IFrame at␣
↪0x12a11b3d0>
```