Final

November 6, 2019

[1]:
```
from IPython import get_ipython;
get_ipython().magic('reset -sf')
#https://github.com/kirbs-/hide_code
```

[2]:
```
%run -i Packages.py
%matplotlib inline
%load_ext rpy2.ipython
```

[3]:
```
%%R
library(ggplot2)
library(readr)
library(lubridate)
library(dplyr)
library(tidyr)
library(viridis)
```

1 Abstract

2 Background

2.1 Electricity Usage

- Two ways of looking at this

1. Internet = lots of electricity
2. Internet can save vast amounts of electricity

[@article{itu2018}NOT IN LIB] In 2015, the International Telecommunication Union estimated about 3.2 billion people, or almost half of the world's population, would be online by the end of the year.

[@article{haseeb2019does}NOT IN LIB] examined the impact of ICTs (i.e., internet usage and mobile cellular subscriptions), globalization, electricity consumption, financial development, and economic growth on environmental quality by using 1994–2014 panel data of BRICS economies. ... The empirical results demonstrate that rise in both internet usage and mobile cellular subscription (ICTs) likely mitigates CO2 emissions in BRICS economies.

[@article{feuerriegel2016}] discuss electricity demand response - allowing the management of demand side resources in real-time.

Demand Response allows for the management of demand side resources in real-time; i.e. shifting electricity demand according to fluctuating supply. When integrated into electricity markets, Demand Response can be used for load shifting and as a replacement for both control reserve and balancing energy. These three usage scenarios are compared based on historic German data from 2011 to determine that load shifting provides the highest benefit: its annual financial savings accumulate to €3.110M for both households and the service sector. This equals to relative savings of 2.83% compared to a scenario without load shifting.

& [@article{palensky2011demand}NOT IN LIB]

Talk broadly about - Internet - End Users (household demand for electricity) - IoT

[@article{bober2009distributed}NOT IN LIB] - As recently as 2009 it was posited that "The proposed model allows for introduction of power priorities for consumer's electrical-equipments by the importance of their functions. The relevance of the functions carried out by the data device is evaluated by each consumer individually. The relevance of functions and priorities assigned to power mode / groups of electrical equipments can be changed over time."

'Haseeb et al [@article{haseeb2019does}NOT IN LIB] have (through review) shown that at a macro scale, global adoption of internet reduces energy consumption' + At the micro scale (that is, the behavior of individuals…)

Consider COST OF IOT SERVICES??? Link this back to the aformentioned increased in global internet users

Pivot into micro - end users… As previously mentioned, demand shifting

HMI = Human-Machine Interface

'How smart do smart meters need to be?' - this is fundamentally at odds with the vision described above. In this scenario, end users would still be having to interact with HMIs. - From a behavioural point of view, moving forward this type of behaviour will become increasingly anachronistic (think about tuning a television set) - Additionally, there is associated resouce (time of end user & physical resource to make the HMI) which contributes to environmental burden (this is a NULL point, as authors specifically mention interaction via a tablet computer) ALTHOUGH, the point can still be made that interaction with

(J.-S. Chou and N.-S. Truong) posit that 'A user's ignorance of methods for saving energy is generally attributable to a lack of relevant feedback.'

Review of available literature (bad?) has shown that currently, consumer convenience and consumer energy consumption are considered in relative isolation from one-another (detached, disjoined). For example, analysis of texts X Y Z yield only T mention of convenience, whilst analysis of texts A B C yield only S mentions of energy consumption.

This work aims to 'bridge the gap' between these two areas of consideration.

Bring it back to the idea that previous forecasting / intervention models have focussed on

@article{huang2018} dis

@article{kobus2015} et al contend that *Today's major developments in the production and demand of electricity in domestic areas make it increasingly important that domestic electricity demand can respond to the availability of electricity*. They also note that *the benefits resulting from domestic demand response depend on household acceptance and behavioural change* - Aim is to investigate if households can shift their electricity demand to times when electricity is abundantly available - Two major developments in the coming decades 1. The amount of distributed renewable electricity generation will increase (e.g., growing number of installed photovoltaic (PV) panels) 2. Significant increase in electricity demand - due to widespread introduction of energy-reliant (albeit more efficient) technologies

Smart grids: Consideration of power greid in real time to drive efficiency in the power grid at a macro-level (NOT: We will define macro as …) ADD PROPER DEFINITION for SMARTGRID HOWEVER *the effects of smart grids strongly depend on the successful implementation of demand response*

programs Demand Response = *Demand response as a household action (automated, manual, orboth) due to which electricity use is shifted in time in response to a price signal or other stimuli.* Factors - Availability of PV electricity - Availability of other renewable energy - 'Peak' electricity grid time (this may vary) - Weather / climate considerations (e.g., heatwave) - Interaction with Energy Management System (EMS)

An example of a price signal that several countries have in place is aday- and night pricing scheme. In this scheme, electricity is cheaperduring the night, when demand is low. Households were also equipped with PV panels, an Energy Management System (EMS), and a dynamic tariff. DEMAND RESPONSE - 'Demand response of a household'

The usage of wet appliances, such as dishwashers, washing machines and dryers, is in general not very time critical and therefore can be shifted.

The user defines an ultimate finish time and within this time frame, the smart applianceautomatically defines the most appropriate starting time

*Overall - concept that 'end user' will mediate power consumption between various appliances...

Based on Darby and McKenna (REF), we define demand response and a household action (automated, manual, or both) due to which electricity use is shifted in time in response to a price signal of other stimuli. This can result in more efficient usage of the available sustainable electricity, like self-consumption of on-site PV electricity (REF) and peak demand reductions (REF) - all from @article{kobus2015}

Assumptions 1. The cost of the sensors is negligible (EXCEPT for discussion later RE top 5 features) 2. The energy requirement for the sensors is negligible 3. In our proposed model, computation is performed in the cloud 4. An app is available to interact with the smart home 5. All of the electrical appliances in our smart home can be remote controlled to some extent

- What about, instead of responding to the average time, the antagonistic AI responded to price signals in the market?

**Huang et al proposed a novel service mining framework to personalize services in an IoT based smart home (REF).

2.2 General IoT + Internet

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 development (societal, economic, e.t.c.,) our species has ever experienced. 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).

Cloud Computing (enabled by the adoption of another ubiquitous computational technology, the World Wide Web) essentially commodifies compute and store, providing on-demand resources for anything from making a social media post, all the way to, searching the Milky Way Galaxy for new planets (https://mast-labs.stsci.io/2018/12/tess-data-available-on-aws).

As with any new technology, Cloud Computing brings with it both new opportunities and new challenges. One such opportunity is the Internet of Things (BRIDGING STATEMENT). 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}). Huang et al explore the idea of Service-oriented Computing (SOC) whereby the inherent complexity associated with networking and programming is abstracted away, shifting the focus from dealing with technical details to a focus on how the services are to be used. For example, under this paradigm, a light connected to the Internet is represented as a light service or a heater connected to the internet is represented as a heating service... 'realm of smart home'... A smart home can be considered as any regular home which has been augmented with various types of IoT services, the purpose of which is to make residents' life more convenient and efficient (@article{huang2018} @ 15/17 of @article{huang2018}).

Discuss IoT *The term "Internet-of-Things" is used as an umbrella keyword for covering various aspects related to the extension of the Internet and the Web into the physical realm, by means ofthe widespread deployment of spatially distributed devices with embedded identification,sensing and/or actuation capabilities. Internet-of-Things envisions a future in which digitaland physical entities can be linked, by means of appropriate information and communica-tion technologies, to enable a whole new class of applications and services. In this article,we present a survey of technologies, applications and research challenges for Internet-of-Things*

[IMAGE - Old Paradigm = user sitting at computer terminal] [IMAGE - New Paradigm = user surrounded by services]

This work aims to bridge the gap between CONVENIENCE and ENERGY USAGE, a previously neglected consideration (these two haven't been directly considered side-by-side, as it were)

Here we observe that the internet is evolving from interconnecting computers to interconnecting things [@article{atzori2010}].

The Internet of Things (IoT) paradigm enables physical devices to connect and exchange information. IoT devices allow objects to be sensed or con-trolled remotely through the Internet [@article{atzori2010}].

2.3 Datasets

The datasets were created during the thesis *Activity Recognition with End-User Sensor Installation in the Home* by Randy Joseph Rockinson, Submitted to the Program of Media Arts and Sciences, School of Architecture and Planning, in partial fulfillment of the requirement for the degree of Master of Science in Media Arts and Sciences at the Massachusetts Institute of Technology (MIT) February 2008 . The work considered the effect of end user versus professional installation of a sensor array in the home - on the basis that, if installation of sensors is to be considered as a high initial cost, and a barrier to entry for end users wanting this technology, is there a difference if a professional versus an end user performs the installation?

- End user installation method is prposed using "stick on" wireless sensors
- Wireless sensors in the home environment
- Data collected from such sensors can be used by software to automatically infer context, such as the activities of daily living.
- This context inference can then be exploited in novel applications for health-care, communication, education, and entertainment.
- Determination of

In this thesis, between 80-100 reed switch sensors where installed in two single-person apartments collecting data about human activity for two weeks. The sensors were installed in everyday objects such as drawers, refrigerators, containers, etc to record opening-closing events (activation deactivation events) as the subject carried out everyday activities.

Based on the two explorator

The PlaceLab The mission of House_n is to conduct research by designing and building real living environments - "living labs" - that are used to study technology and design strategies in context. The PlaceLab is a joint MIT and TIAX, LLC initiative. It is a residential condominium in Cambridge, Massachusetts

GET WEATHER DATA

2.4 Previous Work

This text is italic. Fig 1. DESCRIBE

This text is italic. Fig 1. The work of Huang et al provided initial direction for our data preprocessing. Because we had NO INFORMATION / IT IS A TINY APARTMENT - JUSTIFY the reason for alternate approach RE data processing

$$S =< Seq, T, L >= \begin{Bmatrix} \alpha_1 & \cdots & \alpha_i & \cdots & \alpha_{2n} \\ t_1 & \cdots & t_i & \cdots & t_{2n} \\ l_1 & \cdots & l_i & \cdots & l_{2n} \end{Bmatrix} \tag{1}$$

$$\begin{Bmatrix} s_2^+ & s_1^+ & s_3^+ & s_1^- & s_3^- & s_2^- \\ 48 & 50 & 58 & 65 & 70 & 75 \\ l_2 & l_1 & l_3 & l_1 & l_3 & l_2 \end{Bmatrix} (i.e., l_1 = (1,2), l_2 = (2,4), l_3 = (3,5)) \tag{2}$$

3 Data Preprocessing and Visualisation

3.1 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.

[4]:
```
ds = pd.read_csv(PATH + '/datasets/S1Activities.csv', index_col = 'Code')
ds.head(n=5)
```

[4]:

Code	Heading	Category	Subcategory
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

3.2 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.

[5]:
```
# Importing the dataset
dsS1Sensors = pd.read_csv(PATH + '/datasets/S1sensors.csv',
                          dtype={0:str, 1:str, 2:str},
                          index_col = None, header = None)
dsS1Sensors.head()
```

[5]:

	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

[6]:
```
%run -i S1sensorsPreprocessing.py
```

[7]:
```
dsS1Sensors.head()
```

[7]:

	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

[8]:
```
%run -i getUniqueValues.py
```

[9]:
```
uniqueS1SubActNum = getUniqueValues(dsS1Sensors.iloc[:,0])
len(uniqueS1SubActNum)
```

[9]:
```
76
```

[10]:
```
uniqueS1Sensors = getUniqueValues(dsS1Sensors.iloc[:,3])
len(uniqueS1Sensors)
```

[10]:
```
41
```

[11]:
```
%run -i seen_dupes_dsS1Sensors.py
```

```
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.

[12]:
```
%run -i reqEnergy_containSpecialChar.py
```

[13]:
```
dsS1Sensors.loc[dsS1Sensors['specialChar'] == True]                    #␣
 ↪Filter for true
```

[13]:

	subActNum	room	activity	concat	reqEnergy	specialChar
58	82	Office/study	Drawer	office/study_drawer	False	True
68	92	Office/study	Light switch	office/study_lightswitch	True	

```
68         True
```

[14]:
```
%run -i reqEnergy_containSpecialCharClean.py
```

Comment: Later the ## values for subActNum will become subActNum_##

[15]:
```
dsS1Sensors[58:59] # STORE IN VARIABLE AND CALL IN APPENDIX
```

[15]:
```
    subActNum   room activity        concat  reqEnergy subActNumConcat
58         82  Study   Drawer  study_drawer      False    subActNum_82
```

[16]:
```
# ALL JSON NOTES ADDED TO .py file
%run -i sensorDataJSONwDUPES.py
```

```
{"bathroom_toiletflush": false,
 "bathroom_lightswitch": true,
 "foyer_lightswitch"   : true, ...
```

- Special CHAR dropped
- Dupes not dropped (no point) (????)

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.

[18]:
```
%run -i dsS1Activities_processingExample.py
df.head()
```

[18]:
```
                            Type  \
0  An array of comma-sep strings
1  An array of comma-sep strings
2  An array of comma-sep strings
3  An array of comma-sep strings
4  An array of comma-sep strings

                                       Description   \
0  Activity information, date, start time, end time
1                      Sub-activity reference value
2                   Sub-activity descriptive value
3                         Sub-activity start time
4                           Sub-activity end time

                                        Desired Type
0                                                ...
1                                             Levels
2                                             Levels
3  Datetime including the date extracted from the...
4  Datetime including the date extracted from the...
```

As mentioned above, in order to work with these data, they need to be in a 'tidy' format [ref], that is, the attributes are columns, the rows are instances, and each cell contains only one value. * Note: An array of arrays * An array where each element is an array (list?) * Between each increment of 5 (0 - 4), the sub-arrays have different lengths Table [ref], below, contains a summary of the data structure after the operation `np.array(dsS1)` is performed. In order to continue preprocessing, the array had to be flattened from a 2D structure to a 1D structure, using `flatten()`.

[19]:
```
dsS1 = pd.read_csv(PATH + '/datasets/S1activities_data.csv', sep = 'delimiter',␣
↪header = None, engine = 'python')
```

[20]:
```
dsS1.head()
```

[20]:
```
                                                   0
0                  Bathing,4/1/2003,20:41:35,21:32:50
1  100,68,81,101,93,137,93,58,57,67,93,58,68,88,5...
2  Toilet Flush,Sink faucet - hot,Closet,Light sw...
3  20:51:52,20:51:58,20:53:36,20:53:49,20:53:52,2...
4  21:5:20,20:52:5,20:53:43,21:21:43,20:58:42,20:...
```

[21]:
```
# Confirming the length of the dataframe
len(dsS1)
```

[21]:
```
1475
```

[22]:
```
%run -i dsS1Activities_processing.py
```

The function `dsS1Activities_processing.py` was run in-line to perform the required preprocessing. The dataframe was then converted to a 2D array, using `np.array()`, here, each row from the dataframe became an array within the 2D array [Step 1]. The 2D array was flattened to a 1D array using `.flatten()` [Step 2]. each group of observations was then chunked into a list of 5 [Step 3]

Mention sanity checking here

```
i = 0
while i < 5:
    print(stepX[i])
    i += 1
```

We used the Step 3 data structure to then extract the values for `activity`, `date`, `startTime` and `endTime`. These values were populated into a series of temporary arrays, which were then compiled into a dataset with the four attributes previously listed. We then used `Step3[i][j]` to access all the required elements and parse them into the arrays.

[23]:
```
dsIntermediate.head()
```

[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

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]:
```
ds.head()
```

[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
```

[25]:
```
ds.to_csv(PATH + '/intermediate_datasets/S1Activities_preprocessed.csv', index =␣
↪False)
```

3.3 Importing, Visualizing and Preprocessing the SubActivities Data

Importing `S1Activities_data.csv`, convert df to an array (list?), flatten to a 1D array (list?), chunk the array [5], extract subActNum, subActivity, time & date. Merge time and date into datetime elements, determine start and end time. The variable `dsS1` was used (still on stack from previous section). As previously observed (FIGURE X, TABLE Y), data was all in one column. A processing method with additional steps BLAH BLAH.

[26]:
```
%run -i dsS1SubActivities_processing.py
```

[27]:
```
dsIntermediate.head()
```

[27]:

	subActNum	subAct	date	startTime	endTime
0	100	Toilet Flush	4/1/2003	20:51:52	21:5:20
1	68	Sink faucet - hot	4/1/2003	20:51:58	20:52:5
2	81	Closet	4/1/2003	20:53:36	20:53:43
3	101	Light switch	4/1/2003	20:53:49	21:21:43
4	93	Shower faucet	4/1/2003	20:53:52	20:58:42

[28]:
```
ds.head()
```

[28]:

	subActNum	subAct	start	end
0	67	Cabinet	2003-03-27 06:43:40	2003-03-27 06:43:43
1	100	Toilet Flush	2003-03-27 06:44:06	2003-03-27 07:12:41
2	101	Light switch	2003-03-27 06:44:20	2003-03-27 07:46:34
3	57	Medicine cabinet	2003-03-27 06:44:35	2003-03-27 06:44:48
4	58	Medicine cabinet	2003-03-27 06:44:36	2003-03-27 06:44:48

[29]:
```
#ds.info()
%run -i seen_dupes_dsS1Sensors.py
```

```
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
```

3.3.1 Sub Activity Preprocessing - Duplicate sub activities

The following duplicate sensors were identified in section XX

- From `S1Sensors_preprocessed.csv` I need to find the numbers associated with these - SUBACTNUM
- 'Arbitrarily' choose one number to represent all of them
- Fill them all with that one number

For dicussion later - The kitchen cabinets have 15 sensors, say one of these cabinets just contains a blender, so there will always be a one-to-one between the cabinet opening and the blendor being used. This isn't particularly helpful to us. Also, more sensors could always be added... Better to do analysis with less (explain?) Note for discussion, S1Sensors_preprocessed itself should no longer be modified, EXAMPLE, going back upstream to remove the 'dupes', would perhaps cause an lack of traceability downstream later, Importing S1Sensors_preprecessed now has been done to inform, how to remove 'dupes' from the current ds

[30]:
```
%run -i cleanDupesSubAct.py
%run -i add_DAY_WDWE_phaseII.py
```

Replace the `ds.subAct` values with `dsS1Sensors.concat` values using the `subActNum` key * Join OR * Dict? USE THIS - cleaner

[31]:
```
%run -i refrigeratorDupes.py
```

```
[91, 126, 144]
```

[32]:
```
%run -i removeDupesJSON.py
```

[33]:
```
%run -i refrigeratorDupes.py
```

```
[126]
```

[34]:
```
subActNumKeyWithStringDictNoDupes
```

[34]:
```
{100: 'bathroom_toiletflush',
 101: 'bathroom_lightswitch',
 104: 'foyer_lightswitch',
 105: 'kitchen_lightswitch',
 106: 'kitchen_burner',
 107: 'livingroom_lightswitch',
 108: 'bedroom_lightswitch',
 109: 'porch_lightswitch',
 119: 'kitchen_coffeemachine',
 125: 'kitchen_drawer',
 126: 'kitchen_refrigerator',
 129: 'kitchen_oven',
 130: 'bathroom_door',
 131: 'kitchen_toaster',
 132: 'kitchen_cabinet',
 136: 'kitchen_window',
 137: 'kitchen_freezer',
 139: 'bedroom_jewelrybox',
 140: 'foyer_door',
```

```
 141: 'kitchen_door',
 142: 'kitchen_washingmachine',
 143: 'kitchen_microwave',
 145: 'kitchen_cereal',
 146: 'bedroom_drawer',
 56: 'livingroom_dvd',
 57: 'bathroom_medicinecabinet',
 60: 'kitchen_containers',
 64: 'bedroom_lamp',
 67: 'bathroom_cabinet',
 68: 'bathroom_sinkfaucet-hot',
 70: 'kitchen_dishwasher',
 76: 'livingroom_lamp',
 81: 'foyer_closet',
 82: 'study_drawer',
 85: 'bedroom_window',
 88: 'bathroom_sinkfaucet-cold',
 90: 'kitchen_laundrydryer',
 92: 'study_lightwitch',
 93: 'bathroom_showerfaucet',
 96: 'bathroom_exhaustfan',
 98: 'kitchen_garbagedisposal'}
```

[35]:
```
ds.to_csv('S1SubActivities_preprocessed.csv', index = False)
%run add_DAY_WDWE_phaseI.py
ds = add_DAY_WDWE_phaseI(ds)
ds.to_csv('S1SubActivities_preprocessedR.csv', index = False)
```

3.3.2 SubActivity Preprocessing Comparison and Outliers

INSERT R CODE - Magic or Text?

```
myVar <- 6
```

Loading the Data

[36]:
```
%%R
ds <- read_csv('S1SubActivities_preprocessedR.csv', col_types = cols(subActNum =␣
 ↪col_character(),
                                                                      dayNumeric␣
 ↪= col_character(),
                                                                      HOUR =␣
 ↪col_character()))
ds$start <- ymd_hms(ds$start)
ds$start <- force_tz(ds$start, "America/New_York")
ds$end <- ymd_hms(ds$end)
ds$end <- force_tz(ds$end, "America/New_York")
ds$subAct <- as.factor(ds$subAct)
```

```
total_counts <- ds %>% group_by(DAY, subAct) %>% summarise(count=n())
```

Aggregated Line Chart

[37]:
```
%%R
p <- ggplot(total_counts, aes(x = subAct, y = count, group = DAY, color = DAY))
p <- p + geom_line()
p <- p + geom_point()
p <- p + coord_flip() + labs(title = "Plot of SubActivity Count versus␣
 ↪SubActivity",
                             subtitle = "ADD SUBTITLE - Preprocessing Stage?",
                             caption = "Source: TBU",
                             x = "SubActivity",
                             y = "Aggregated Count")
ggsave("images/lineChart.png", plot = last_plot(), device = png(),
       scale = 1.5, width = 18, height = 16, units = c("cm"), dpi = 300)
```

MENTION THIS FROM Huang et al.

Aggregated Box Plot

[38]:
```
%%R
p <- ggplot(ds, aes(subAct, durationSec))
p <- p + geom_boxplot(fill= "plum", outlier.alpha = 0.8) # aes(colour = WDWE) #,␣
 ↪outlier.size = 1
p <- p + theme(legend.position = "right") + coord_flip() +
  labs(title = "Box Plots",
       subtitle = "Subactivity Duraiton",
       caption = "Source: TBU",
       x = "SubActivity",
       y = "Duration (sec)")
ggsave("images/boxPlots.png", plot = last_plot(), device = png(),
       scale = 1.5, width = 18, height = 16, units = c("cm"), dpi = 300)
```

MENTION THIS FROM Huang et al.

Strip Plots

[39]:
```
%run -i altairDurationCharts.py
```

[40]:
```
# Example charts
charts[1]
```

[40]:
```
<VegaLite 3 object>

If you see this message, it means the renderer has not been properly enabled
for the frontend that you are using. For more information, see
https://altair-viz.github.io/user_guide/troubleshooting.html
```

1. Set specified column as index
2. Extract dayofweek from index (ds.index.dayofweek.astype(str)), as this as an attribute
3. Replace '0' with 'mon' and so on
4. Duplicate...

- Intermittant and persistant??? Definitions...

3.3.3 SubActivity Cleansing - Outlier Removal

TEXT

Bathroom - Toilet Flush, Sub-activity # 100

[ ]:
```
# Toilet Duration Chart
```

Text

[48]:
```
%run -i cleanToilet.py
```

Dropping Values

[42]:
```
%run -i countUnique.py
```

```
38
```

Based on the investigation of the strip plots + XYZ

```
# valueDrop.py
ds = ds[ds.subAct != 'bedroom_jewelrybox']
ds = ds[ds.subAct != 'foyer_closet']
ds = ds[ds.subAct != 'kitchen_cereal']
ds = ds[ds.subAct != 'kitchen_containers']
ds = ds[ds.subAct != 'bedroom_lamp']
ds = ds[ds.subAct != 'livingroom_dvd']
```

[56]:
```
# Jewelry + foyer closet + cereal + containers + lamp
%run -i valueDrop.py
```

[57]:
```
%run -i countUnique.py
```

```
32
```

[58]:
```
# NAME CHANGE?
ds.to_csv('S1SubActivities_temporalFeaturesNoDUPES.csv', index = False)
```

Filling outliers with Median Based on strip plot and so on

[59]:
```
subActNames = ['bathroom_cabinet', 'bathroom_medicinecabinet', 'study_drawer',
               'bedroom_drawer', 'kitchen_cabinet', 'kitchen_microwave',
               'kitchen_door', 'bathroom_showerfaucet', 'kitchen_drawer',
```

```
                'bathroom_sinkfaucet-hot', 'kitchen_freezer', 'bathroom_door',
                'kitchen_toaster', 'kitchen_lightswitch', 'study_lightwitch',
                'kitchen_dishwasher', 'livingroom_lightswitch']
```

[50]:
```
%run -i determineMeanMedian.py
```

[60]:
```
allMedianValues
```

[60]:

	SubAct	Count	Median	Mean	Std
0	bathroom_cabinet	104	3.0	459.221154	3176.163323
1	bathroom_medicinecabinet	194	87.0	3187.974227	7354.142408
2	study_drawer	45	5.0	1633.488889	5432.406346
3	bedroom_drawer	99	9.0	272.474747	1918.726179
4	kitchen_cabinet	406	6.0	111.820197	1864.012246
5	kitchen_microwave	61	5.0	376.409836	2868.667916
6	kitchen_door	134	3.0	45.111940	174.185652
7	bathroom_showerfaucet	88	14.0	1753.352273	6540.729518
8	kitchen_drawer	208	3.0	144.062500	1727.576706
9	bathroom_sinkfaucet-hot	169	10.0	56.905325	501.272871
10	kitchen_freezer	130	38.0	1737.107692	4512.573808
11	bathroom_door	73	16.0	460.616438	2310.811556
12	kitchen_toaster	71	4.0	789.366197	3793.674375
13	kitchen_lightswitch	32	4453.0	9317.718750	13081.340412
14	study_lightwitch	26	1727.5	6429.730769	11002.302517
15	kitchen_dishwasher	86	62.5	1517.406977	4146.637482
16	livingroom_lightswitch	8	6351.5	12544.125000	15556.662849

[52]:
```
%run -i cleanseOutliers.py
```

[53]:
```
ds.head()
```

[53]:

	subActNum	subAct	start	end	dayNumeric	DAY	WDWE	HOUR	durationSec
0	67	bathroom_cabinet	2003-03-27 06:43:40	2003-03-27 06:43:43	3	Thu	WD	6	4
1	100	bathroom_toiletflush	2003-03-27 06:44:06	2003-03-27 06:44:07	3	Thu	WD	6	2
2	101	bathroom_lightswitch	2003-03-27 06:44:20	2003-03-27 07:46:34	3	Thu	WD	6	3735
3	57	bathroom_medicinecabinet	2003-03-27 06:44:35	2003-03-27 06:44:48	3	Thu	WD	6	14
4	57	bathroom_medicinecabinet	2003-03-27 06:44:36	2003-03-27 06:44:48	3	Thu	WD	6	13

[54]:
```
ds.to_csv('S1SubActivities_temporalFeaturesCLEANSED.csv', index = False)
```

3.3.4 Subactivity Visualisation

[ ]:

Livingroom DVD DROPPED - One value only * 2
MENTION THIS FROM Huang et al.

Bathroom Medicine Cabinet Text
MENTION THIS FROM Huang et al.

Kitchen Containers DROPPED
MENTION THIS FROM Huang et al.

Bedroom Lamp DROPPED
MENTION THIS FROM Huang et al.

Bathroom Cabinet Text
MENTION THIS FROM Huang et al.

Bathroom Sickfaucet - Hot Text
MENTION THIS FROM Huang et al.

Kitchen Dishwasher Text
MENTION THIS FROM Huang et al.

Livingroom Lamp Text
MENTION THIS FROM Huang et al.

Foyer Closet DROPPED
MENTION THIS FROM Huang et al.

Study Drawer Text
MENTION THIS FROM Huang et al.

Bathroom Sickfaucet - Cold Text
MENTION THIS FROM Huang et al.

Kitchen Laundry Dryer Text
MENTION THIS FROM Huang et al.

Study Lightswitch Text
MENTION THIS FROM Huang et al.

Bathroom Shower Faucet 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
MENTION THIS FROM Huang et al.

Bathroom Lightswitch Text
MENTION THIS FROM Huang et al.

Foyer Lightswitch Text
MENTION THIS FROM Huang et al.

Kitchen Lightswitch Text
MENTION THIS FROM Huang et al.

Kitchen Burner Text
MENTION THIS FROM Huang et al.

Livingroom Lightswitch Text
MENTION THIS FROM Huang et al.

Bedroom Lightswitch Text
MENTION THIS FROM Huang et al.

Kitchen Coffee Machine Text
MENTION THIS FROM Huang et al.

Kitchen Drawer Text
MENTION THIS FROM Huang et al.

Kitchen Refrigerator Text
MENTION THIS FROM Huang et al.

Kitchen Oven Text
MENTION THIS FROM Huang et al.

Bathroom Door Text
MENTION THIS FROM Huang et al.

Kitchen Toaster Text
MENTION THIS FROM Huang et al.

Kitchen Cabinet Text
MENTION THIS FROM Huang et al.

Kitchen Freezer Text
MENTION THIS FROM Huang et al.

Bedroom Jewelrybox DROPPED
MENTION THIS FROM Huang et al.

Foyer Door Text
MENTION THIS FROM Huang et al.

Kitchen Door - CORRECT TYPO Text
MENTION THIS FROM Huang et al.

Kitchen Washingmachine Text
MENTION THIS FROM Huang et al.

Kitchen Microwave Text
MENTION THIS FROM Huang et al.

Kitchen Cereal DROPPED
MENTION THIS FROM Huang et al.

Bedroom Drawer Text
MENTION THIS FROM Huang et al.

---

4 Data Analysis

4.1 Determination of the most 'effective' data structure - NOT HERE

Key finding = data structure most amenable...

- Driven by two main concepts (main?)
- Driven by two competing concepts (competing?)

MENTION THIS FROM Huang et al.

[1]:
```
%run -i Packages.py
%matplotlib inline
%load_ext rpy2.ipython
```

[3]:
```
%run -i PlotlyPackages.py
ds = pd.read_csv('S1SubActivities_temporalFeaturesCLEANSED.csv', index_col =␣
 ↪None)
ds.start = pd.to_datetime(ds.start, format='%Y-%m-%d %H:%M:%S')
ds.end = pd.to_datetime(ds.end, format='%Y-%m-%d %H:%M:%S')
ds = ds.sort_values('start')
ds.reset_index(drop = True, inplace = True)
```

[4]:
```
ds.head() # REALLY RE-IMPORT???
```

[4]:

	subActNum	subAct	start	\
0	67	bathroom_cabinet	2003-03-27 06:43:40	
1	100	bathroom_toiletflush	2003-03-27 06:44:06	
2	101	bathroom_lightswitch	2003-03-27 06:44:20	
3	57	bathroom_medicinecabinet	2003-03-27 06:44:35	
4	57	bathroom_medicinecabinet	2003-03-27 06:44:36	

	end	dayNumeric	DAY	WDWE	HOUR	durationSec
0	2003-03-27 06:43:43	3	Thu	WD	6	4
1	2003-03-27 06:44:07	3	Thu	WD	6	2
2	2003-03-27 07:46:34	3	Thu	WD	6	3735
3	2003-03-27 06:44:48	3	Thu	WD	6	14
4	2003-03-27 06:44:48	3	Thu	WD	6	13

MENTION SEGMETNATION HERE OF 3 PARTS OF DAY - Morning - Afternoon - Evening

4.2 ADD DAY SEGMENTATATION SECTION

Problem Statement **If start EVENTA between X and Y on a WD/WE the PR that I will start EVENTB within Z minutes is Q.**

> *If I switch on the bathroom light switch between 6am and 7am on a weekday, the probability that I will use my razor is X.*

- Relationships
- Intersection

- Separate
- Enclosed
- Equal

1. Equal Start
2. Delta positive (the highest)
3. Delta Zero (the first)
4. Delta Negative (closest to zero)

And nearest 5

The dataset @SANKEY The sankey diagrams show the sequences such that certain activities (NOT requiring electricity) mostly [always?] come before those that require electricity. In other words, 'energy poor' activities often act of preparative in the lead up to 'energy intensive' activities. The Sankey can show this by looking at the 'terminal events'.

4.3 Sequential Analysis

Reasoning

[5]:
```
import datetime as dt

def id_delta(events, n=1, delta_threshold=dt.timedelta(-99)):
    nns = []
    for row in events.itertuples():
        #print(row)
        start_time = getattr(row, 'start')
        end_time = getattr(row, 'end')
        subActNum = getattr(row, 'subActNum')
        row_index = getattr(row, 'Index')

        nn = events[(events.start >= start_time) &
                    (events.index != row_index) &
                    ((start_time - events.start) > delta_threshold)][:n]
        #print(len(nn))
        ordered = pd.DataFrame()
        ordered['Dummy'] = nn['subActNum']
        ordered['EventA'] = subActNum
        ordered['EventB'] = nn['subActNum']
        ordered['EvA_Start'] = start_time
        ordered['EvB_Start'] = nn['start']
        ordered['EvA_End'] = end_time
        ordered['EvB_End'] = nn['end']
        del ordered['Dummy']
        nns.append(ordered)

    #print(nns)
    result = pd.concat(nns)
    result['Delta'] = np.where(result['EvA_Start']==result['EvB_Start'],
```

```
                                0,
                                (result['EvB_Start'] - result['EvA_Start']))
    result['Delta'] = result['Delta'].dt.total_seconds()
    return result

#ds_1n_25s['Delta'].dt.total_seconds()
```

[6]:
```
%run -i add_DAY_WDWE_phaseII.py
```

[23]:
```
ds_1n_60s = id_delta(ds, 1, dt.timedelta(0,-60))     # Creating DS with specified␣
 ↪time and n
ds_1n_60s = add_DAY_WDWE_phaseII(ds_1n_60s)          # Adding temporal features
# SHOULD I LIMIT TO 60 SECONDS?
```

[24]:
```
ds_1n_60s.head(n=10) # MENTION THAT OWING TO TIME CONSTAINTS, COULD NOT DO␣
 ↪TEMPORAL EVALUATION
```

[24]:

	EventA	EventB	EvA_Start	EvB_Start	EvA_End	EvB_End	Delta	DAY	WDWE	Hour
0	67	100	2003-03-27 06:43:40	2003-03-27 06:44:06	2003-03-27 06:43:43	2003-03-27 06:44:07	26.0	Thu	WD	6
1	100	101	2003-03-27 06:44:06	2003-03-27 06:44:20	2003-03-27 06:44:07	2003-03-27 07:46:34	14.0	Thu	WD	6
2	101	57	2003-03-27 06:44:20	2003-03-27 06:44:35	2003-03-27 07:46:34	2003-03-27 06:44:48	15.0	Thu	WD	6
3	57	57	2003-03-27 06:44:35	2003-03-27 06:44:36	2003-03-27 06:44:48	2003-03-27 06:44:48	1.0	Thu	WD	6
4	57	67	2003-03-27 06:44:36	2003-03-27 06:44:49	2003-03-27 06:44:48	2003-03-27 06:44:56	13.0	Thu	WD	6
5	67	82	2003-03-27 06:44:49	2003-03-27 06:45:45	2003-03-27 06:44:56	2003-03-27 06:45:48	56.0	Thu	WD	6
6	82	146	2003-03-27 06:45:45	2003-03-27 06:46:12	2003-03-27 06:45:48	2003-03-27 06:46:20	27.0	Thu	WD	6
7	143	132	2003-03-27 06:54:09	2003-03-27 06:54:16	2003-03-27 06:54:14	2003-03-27 06:54:19	7.0	Thu	WD	6
8	141	93	2003-03-27 07:04:55	2003-03-27 07:05:22	2003-03-27 07:04:57	2003-03-27 07:05:24	27.0	Thu	WD	7
9	93	132	2003-03-27 07:05:22	2003-03-27 07:05:39	2003-03-27 07:05:24	2003-03-27 07:05:57	17.0	Thu	WD	7

[15]:
```
ds_10n_60s = id_delta(ds, 10, dt.timedelta(0,-60))     # Creating DS with␣
 ↪specified time and n
ds_10n_60s = add_DAY_WDWE_phaseII(ds_10n_60s)          # Adding temporal features
# SHOULD I LIMIT TO 60 SECONDS?
```

[19]:
```
ds_10n_60s.head(n=10) # MENTION THAT OWING TO TIME CONSTAINTS, COULD NOT DO␣
 ↪TEMPORAL EVALUATION
```

[19]:
```
   EventA  EventB           EvA_Start           EvB_Start             EvA_End  \
0      67     100 2003-03-27 06:43:40 2003-03-27 06:44:06 2003-03-27 06:43:43
1      67     101 2003-03-27 06:43:40 2003-03-27 06:44:20 2003-03-27 06:43:43
2      67      57 2003-03-27 06:43:40 2003-03-27 06:44:35 2003-03-27 06:43:43
3      67      57 2003-03-27 06:43:40 2003-03-27 06:44:36 2003-03-27 06:43:43
4     100     101 2003-03-27 06:44:06 2003-03-27 06:44:20 2003-03-27 06:44:07
5     100      57 2003-03-27 06:44:06 2003-03-27 06:44:35 2003-03-27 06:44:07
6     100      57 2003-03-27 06:44:06 2003-03-27 06:44:36 2003-03-27 06:44:07
7     100      67 2003-03-27 06:44:06 2003-03-27 06:44:49 2003-03-27 06:44:07
8     101      57 2003-03-27 06:44:20 2003-03-27 06:44:35 2003-03-27 07:46:34
9     101      57 2003-03-27 06:44:20 2003-03-27 06:44:36 2003-03-27 07:46:34

              EvB_End  Delta  DAY WDWE  Hour    Phase
0 2003-03-27 06:44:07   26.0  Thu   WD     6  Morning
1 2003-03-27 07:46:34   40.0  Thu   WD     6  Morning
2 2003-03-27 06:44:48   55.0  Thu   WD     6  Morning
3 2003-03-27 06:44:48   56.0  Thu   WD     6  Morning
4 2003-03-27 07:46:34   14.0  Thu   WD     6  Morning
5 2003-03-27 06:44:48   29.0  Thu   WD     6  Morning
6 2003-03-27 06:44:48   30.0  Thu   WD     6  Morning
7 2003-03-27 06:44:56   43.0  Thu   WD     6  Morning
8 2003-03-27 06:44:48   15.0  Thu   WD     6  Morning
9 2003-03-27 06:44:48   16.0  Thu   WD     6  Morning
```

[ ]:
```
# SAVE THE CSV AND CB (below)!!!
```

4.4 Sankey Diagrams - Qualitative Assessment

[ ]:
```
```

[9]:
```
def genSankey(df,cat_cols=[],value_cols='',title='Sankey Diagram'):
    # maximum of 6 value cols -> 6 colors
    labelList = []
    colorNumList = []
    for catCol in cat_cols:
        labelListTemp =  list(set(df[catCol].values))
        colorNumList.append(len(labelListTemp))
        labelList = labelList + labelListTemp

    # remove duplicates from labelList
    labelList = list(dict.fromkeys(labelList))

    ds = pd.read_csv('S1Sensors_preprocessed.csv', index_col = 'subActNum')

    colorList = []
    for subActNum in labelList:
        if ds.loc[subActNum, 'reqEnergy']:
```

```
            colorList.append("red")
        else:
            colorList.append("blue")

    newLabelList = []
    for subActNum in labelList:
        newLabelList.append(ds.loc[subActNum, 'room'] + " - " + ds.
↪loc[subActNum, 'activity'])

    # transform df into a source-target pair
    for i in range(len(cat_cols)-1):
        if i==0:
            sourceTargetDf = df[[cat_cols[i],cat_cols[i+1],value_cols]]
            sourceTargetDf.columns = ['source','target','count']
        else:
            tempDf = df[[cat_cols[i],cat_cols[i+1],value_cols]]
            tempDf.columns = ['source','target','count']
            sourceTargetDf = pd.concat([sourceTargetDf,tempDf])
        sourceTargetDf = sourceTargetDf.groupby(['source','target']).
↪agg({'count':'sum'}).reset_index()

    # add index for source-target pair
    sourceTargetDf['sourceID'] = sourceTargetDf['source'].apply(lambda x:␣
↪labelList.index(x))
    sourceTargetDf['targetID'] = sourceTargetDf['target'].apply(lambda x:␣
↪labelList.index(x))

    labelList = newLabelList

    # creating the sankey diagram
    data = dict(type='sankey',
                node = dict(pad = 15, thickness = 20, line = dict(color =␣
↪"black", width = 0.5),
                            label = labelList,
                            color = colorList),
                link = dict(source = sourceTargetDf['sourceID'],
                            target = sourceTargetDf['targetID'],
                            value = sourceTargetDf['count']))

    layout =  dict(title = title, font = dict(size = 10))
    fig = dict(data=[data], layout=layout)

    return fig
# COLOUR - https://stackoverflow.com/questions/55862005/
↪plotly-sankey-diagram-group-label-and-color
```

```
# https://medium.com/plotly/
 ↪4-interactive-sankey-diagram-made-in-python-3057b9ee8616
# https://community.periscopedata.com/t/63nx0x/
 ↪sankey-diagrams-with-plot-ly-in-periscope
# https://medium.com/kenlok/
 ↪how-to-create-sankey-diagrams-from-dataframes-in-python-e221c1b4d6b0
# https://community.periscopedata.com/t/k9s9mg/sankey-plot-ly
# https://plot.ly/python/sankey-diagram/
```

[13]:
```
#fig = genSankey(ds_1n_25s, cat_cols=['EventA', 'EventB'], value_cols='Delta',␣
 ↪title='ds')
#py.iplot(go.Figure(fig))
# CLEAN UP - SCALING + COLOURS + NUMBERS CONVERTED TO WORDS (or KEY?)
```

[25]:
```
wdwe_title = {"WD":"Weekday", "WE":"Weekend"}

def addPhase(df):
    df['Phase'] = "Afternoon"
    df.loc[df['Hour'] < 12, 'Phase'] = "Morning"
    df.loc[df['Hour'] >= 18, 'Phase'] = "Evening"

def plotASankey(df, wdwe, phase):
    df = df[(df.WDWE==wdwe) & (df.Phase==phase)]
    fig = genSankey(df, cat_cols=['EventA', 'EventB'], value_cols='Delta',␣
 ↪title=wdwe_title[wdwe] + ' ' + phase)
    py.plot(go.Figure(fig), filename=wdwe_title[wdwe] + ' ' + phase + '.html')

addPhase(ds_1n_60s)

for wdwe in ['WD','WE']:
    for phase in ['Morning', 'Afternoon', 'Evening']:
        plotASankey(ds_1n_60s, wdwe, phase)
```

[61]:
```
# SEGMENT TIMES 3 for WD and WE
```

[22]:
```
wdwe_title = {"WD":"Weekday", "WE":"Weekend"}

def addPhase(df):
    df['Phase'] = "Afternoon"
    df.loc[df['Hour'] < 12, 'Phase'] = "Morning"
    df.loc[df['Hour'] >= 18, 'Phase'] = "Evening"

def plotASankey(df, wdwe, phase):
    df = df[(df.WDWE==wdwe) & (df.Phase==phase)]
    fig = genSankey(df, cat_cols=['EventA', 'EventB'], value_cols='Delta',␣
 ↪title=wdwe_title[wdwe] + ' ' + phase)
    py.plot(go.Figure(fig), filename=wdwe_title[wdwe] + ' ' + phase + '.html')
```

```
addPhase(ds_10n_60s)

for wdwe in ['WD','WE']:
    for phase in ['Morning', 'Afternoon', 'Evening']:
        plotASankey(ds_10n_60s, wdwe, phase)
```

[ ]:
```
import gc
gc.collect()
```

[ ]:
```
from IPython import get_ipython;
get_ipython().magic('reset -sf')
```

[62]:
```
%run -i Packages.py
%matplotlib inline
```

4.5 Timestamp Structure with Boolean

4.5.1 Constructing Boolean Array

4.5.2 Preprocessing

Input `ds = pd.read_csv('S1SubActivities_preprocessed.csv', index_col = None)` **Output** `ds.to_csv('S1SubActivities_timeStampRanges.csv',index=False)`

- WHY

1. This enables us to calculate the FREQUENCY per hour (which in the end was just used to inform later analysis)

[ ]:
```
ds = pd.read_csv(PATH + '/intermediate_datasets/
 ↪S1SubActivities_temporalFeaturesPREPROCESSED.csv',
                 index_col = None)
```

[ ]:
```
ds.head()
```

[ ]:
```
%run -i addTemporalArrays.py
```

[ ]:
```
ds.head()
```

[ ]:
```
ds.to_csv(PATH + '/intermediate_datasets/S1SubActivities_timeStampRanges.
 ↪csv',index = False)
```

[ ]:
```
ds = ds.set_index(start, drop = True)
ds = ds.drop(columns = ['durationSec', 'subAct','dayNumeric', 'DAY',
                        'WDWE', 'HOUR', 'timeStampList', 'start'])
```

[ ]:
```
ds.head()
```

[ ]:
```
s = ds.apply(lambda x: pd.Series(x['timeStampArrayList']),axis=1).stack().
 ↪reset_index(level=1, drop=True)
s.name = 'duration'
ds = ds.drop('end', axis=1).join(s)
```

[ ]:
```
ds.head(n=4)
```

[ ]:
```
ds = ds.drop(columns = ['timeStampArrayList'])
```

[ ]:
```
ds.to_csv(PATH + '/intermediate_datasets/S1SubActivities_timeRangeMelt.csv',␣
↪index=False)
```

[ ]:
```
ds = ds.drop(columns = ['subAct', 'actDuration'])
```

[ ]:
```

```

[ ]:
```

```