Performance Loss Analysis Methods on UNSW TEBT PV System

Christopher H. Fowler
z3470052
Supervisor: Dr. Jose Bilbao

1. Reporting on Progress

To report on the progress I have made over the term, given the data focused nature of this project, I have decided it's easier to present iterations of code and results that formed parts of the methodology that I've developed so far. As a refresher, the goals of the projects are as follows:

Primary Goals

- Implement standard performance loss methodologies (LR, STL, YOY)
- Apply performance loss methodologies to UNSW TEBT data

Secondary Goals

- Determine effectiveness of data cleaning methods used in PL analysis
- Determine effectiveness of data fitting methods used in PL analysis
- Observe non-linear PL in analysis
- Explore alternative methodologies as time permits

The primary goals have had major hiccups in 'Applying performance loss methodologies to UNSW TEBT data' where processing the TEBT data into a useable form has taken a significant portion of time. The trials of completing this task make up the bulk of this report, along with reporting on the regular methodology test examples that have been used.

The Industry Methodology

To fulfil any of the standard performance loss methodologies we follow the 5 general steps outlined by the industry for this type of problem:

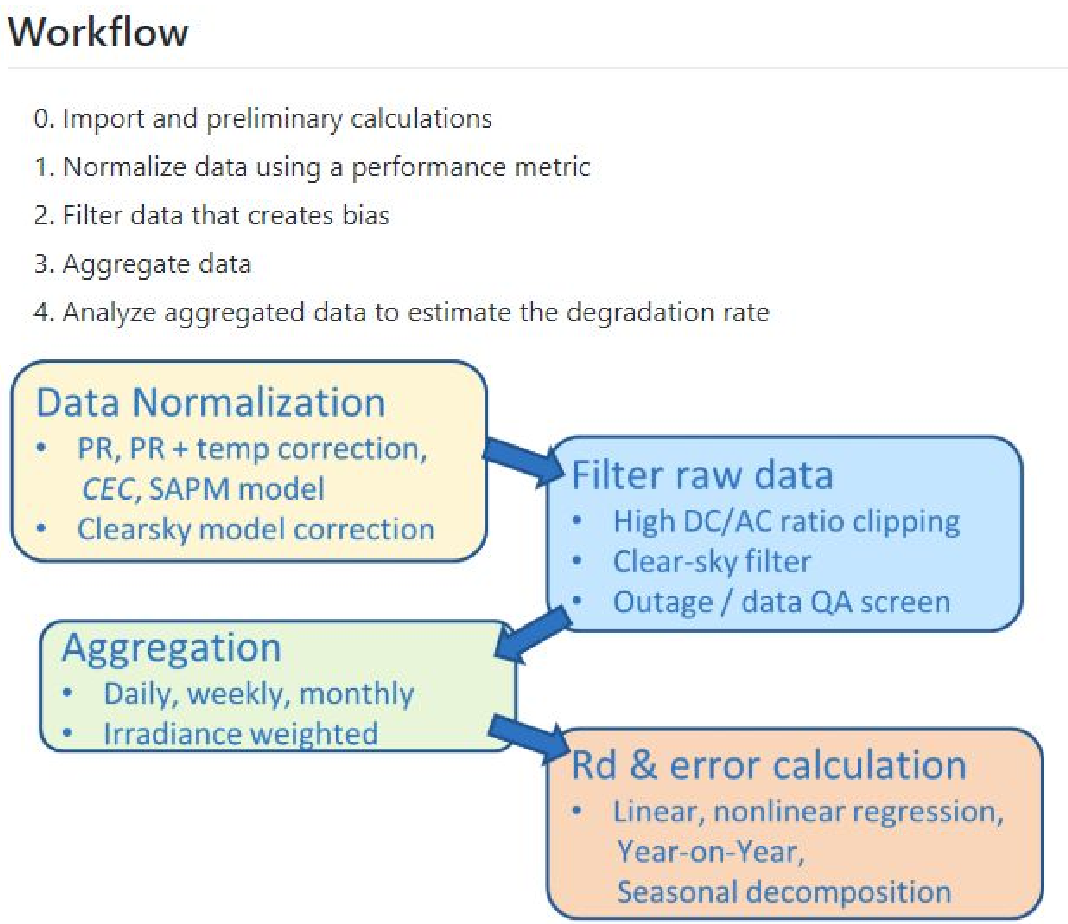


Fig 1: RDTools methodology
[RdTools, version 1.2.2, https://github.com/NREL/rdtools, DOI:10.5281/zenodo.1210316]

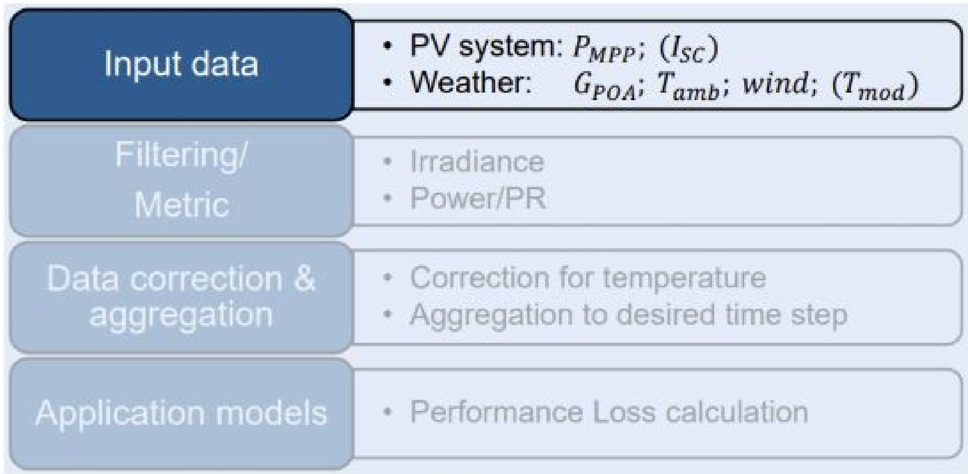


Fig 2: EURAC Methodology
[D Moser, 2019, “Performance, Operation and Reliability of Photovoltaic Systems”]

As part of correctly using these methodologies, the RDTools (giving YOY w/ stages 1-4) and Statsmodels (giving STL w/ stage 2-4) packages give working examples on test datasets that can then be modified to fit the dataset we provide. With respect to how have been and am going about implementing these methodologies, please refer to the ‘Revised Project Plan’ section.

Overview of Progress through Git commits

Setting up and using git to maintain a healthy codebase is essential as part of any coding project, and a tool that I am quite the novice at using. Below in fig 3 is the result of my amertuer attempts to use git properly - the list of commits is relatively short with very large old commits at the bottom and very small more recent commit at the top as I began to use git more appropriately. There were issues with some merge conflicts between my two workstations which were finally fixed in a large merge under the master branch. The ‘ThesisNight’ branch is the culmination of more recent attempts at importing data from TEBT. Some assistance has been given from a helpful computer science 4th year undergrad who was more than eager to groan at my many mistakes in using this tool, however progress has been made to use it properly.

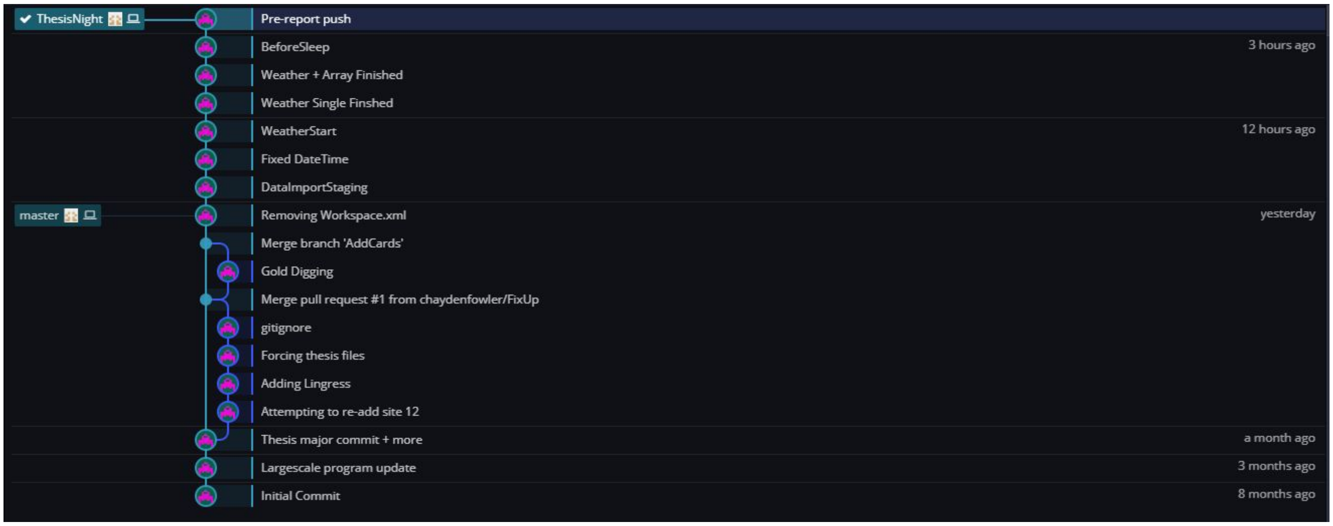


Fig 3: Flow of git commits

Test Examples on Methodologies

In order to test the methodologies, package devs tend to create sample datasets to test on. In order to implement the methodologies, the first step is to execute these test sets to ensure correct installation and use. Below in figures 4 and 5 are the (expected) results of the test examples. Installing the correct package version of statsmodels was more difficult than expected, as the dev branch requires compilation and several OS specific dependencies not linked to the pip package manager, as a result fig 6 is my proof of success in properly getting this package to work.

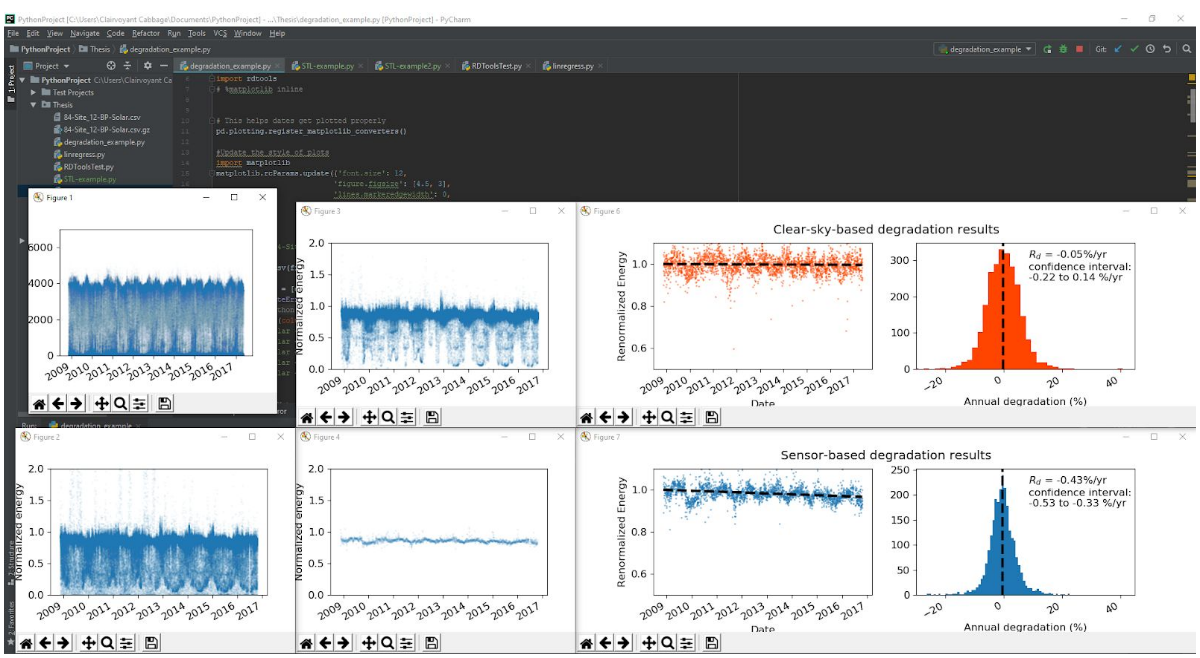


Fig 4: RDTools test example on Desert Knowledge unit 12

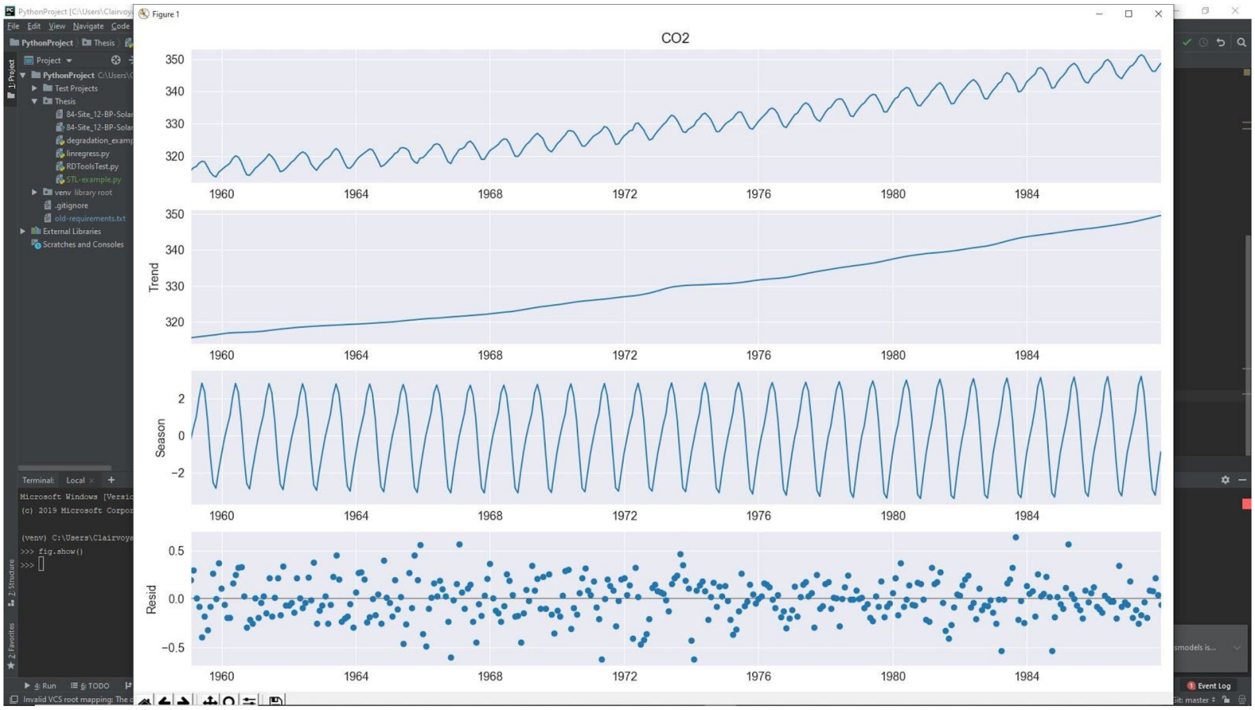


Fig 5: Statsmodels test example on CO2 emissions

Command Prompt

```
(venv) C:\Users\Clairvoyant Cabbage\Documents\PythonProject>pip list
Package         Version
--------------- -------------------------
certifi         2019.9.11
chardet         3.0.4
cycler          0.10.0
Cython          0.29.13
h5py            2.10.0
idna            2.8
kiwisolver      1.1.0
matplotlib      3.1.1
numpy           1.17.3
pandas          0.25.2
patsy           0.5.1
pip             19.3.1
pvlib           0.5.2
pyparsing       2.4.2
python-dateutil 2.8.0
pytz            2019.3
rdtools         1.2.2
requests        2.22.0
scipy           1.3.1
seaborn         0.9.0
setuptools      41.4.0
six             1.12.0
statsmodels     0.11.0.dev0+517.g0315fdd24
urllib3         1.25.6

(venv) C:\Users\Clairvoyant Cabbage\Documents\PythonProject>
```

Fig 6: Successfully installing dev branch of statsmodels

Jupyter Notebooks

In order to become more accustomed to the way the industry is reporting changes, test examples and explaining how their product works, I've taken to using Jupyter notebooks for the data import phase of my methodology. The notebooks are quite space inefficient when converting from ipynb to pdf, and so this section will exceed the '15 page soft limit' under the pretense that there is significantly less information to read through per page than usual.

Formatting space could be saved by instead screenshotting the results I have, but for the purposes of reproducibility and being thorough, the code and results should be fully available by the auto-formatted pdfs. As Jupyter can export .TeX files, in future LaTeX will be considered to improve report formatting.

Some reflections on progress are included in the Jupyter notebooks as being next to the issue or solutions is more relevant than in its own section, although reflections on progress will be summarised later also.

unswDataImportSingle

Data import test on singular TEBT file, using the 01/01/2018 dated file. We are forced to skip the first 6 rows (which is when the actual data starts), and to ignore the headers. The original headers in the file are partially uncomaptable due to an encoding issue (pandas doesn't like the degree symbol in the encoding it chose) and despite a true solution being avaialble, the workaround is to specify all the headings hardcoded. Once we have all the data we can rename the headers to the original headers by enumerating the header list.

Beware, the headers string is very long and is left in for reproducibility purposes. It shall be truncated later.

In [2]:

```
import pandas as pd
file_name = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018
-Array\2018-01-01.csv"
#file_name = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\84-Site_12-B
P-Solar.csv"
df1 = pd.read_csv(file_name, delimiter=";", header=None, skiprows=6)
headers = "TimeStamp;ExlSolIrr;IntSolIrr;SMA-h-On;TmpAmb C;TmpMdul C;WindVel km/h;A.Ms.
Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;G
M.TotWhOut;GridMs.A.phsA;GridMs.A.phsB;GridMs.A.phsC;GridMs.Hz;GridMs.PhV.phsA;GridMs.P
hV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;GridMs.TotVA;GridMs.TotVAr;GridMs.VA.phsA;GridM
s.VA.phsB;GridMs.VA.phsC;GridMs.VAr.phsA;GridMs.VAr.phsB;GridMs.VAr.phsC;GridMs.W.phsA;
GridMs.W.phsB;GridMs.W.phsC;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtC
ntUsr;Op.EvtNo;Op.GriSwStt;Op.Health;Op.Prio;Op.TmsRmg;Pac;PCM-DigInStt;PlntCtl.Stt;Ser
ial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Am
p;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;GridMs.Hz;GridMs.PhV.phsA;GridMs.
PhV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.T
otTmh;Op.EvtCntUsr;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Serial Number;A.Ms.Am
p;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vo
l;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;GridMs.Hz;GridMs.PhV.phsA;GridMs.PhV.phsB;GridMs.Ph
V.phsC;GridMs.TotPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUs
r;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Serial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.W
att;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.M
s.Amp;Error;E-Total;GridMs.Hz;GridMs.PhV.phsA;GridMs.PhV.phsB;GridMs.PhV.phsC;GridMs.To
tPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUsr;Op.EvtNo;Op.Gri
SwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Serial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.
Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Tota
l;GridMs.Hz;GridMs.PhV.phsA;GridMs.PhV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;Inv.TmpLimS
tt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUsr;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pa
c;PlntCtl.Stt;Serial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A
4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;GridMs.Hz;GridM
s.PhV.phsA;GridMs.PhV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mod
e;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUsr;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Ser
ial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Am
p;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;GridMs.Hz;GridMs.PhV.phsA;GridMs.
PhV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.T
otTmh;Op.EvtCntUsr;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Serial Number;A.Ms.Am
p;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vo
l;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;GridMs.Hz;GridMs.PhV.phsA;GridMs.PhV.phsB;GridMs.Ph
V.phsC;GridMs.TotPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUs
r;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Serial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.W
att;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.M
s.Amp;Error;E-Total;GridMs.Hz;GridMs.PhV.phsA;GridMs.PhV.phsB;GridMs.PhV.phsC;GridMs.To
tPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUsr;Op.EvtNo;Op.Gri
SwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Serial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.
Ms.Amp;A3.Ms.Amp;A4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Tota
l;GridMs.Hz;GridMs.PhV.phsA;GridMs.PhV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;Inv.TmpLimS
tt;InvCtl.Stt;Mode;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUsr;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pa
c;PlntCtl.Stt;Serial Number;A.Ms.Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp;A2.Ms.Amp;A3.Ms.Amp;A
4.Ms.Amp;A5.Ms.Amp;B.Ms.Amp;B.Ms.Vol;B.Ms.Watt;B1.Ms.Amp;Error;E-Total;GridMs.Hz;GridM
s.PhV.phsA;GridMs.PhV.phsB;GridMs.PhV.phsC;GridMs.TotPFPrc;Inv.TmpLimStt;InvCtl.Stt;Mod
e;Mt.TotOpTmh;Mt.TotTmh;Op.EvtCntUsr;Op.EvtNo;Op.GriSwStt;Op.TmsRmg;Pac;PlntCtl.Stt;Ser
ial Number"
headers = dict(enumerate(headers.split(';')))
df1 = df1.rename(columns = headers)
print(df1)
```

```
    TimeStamp  ExlSolIrr  IntSolIrr  SMA-h-On  TmpAmb C  TmpMdul C  \
0       00:00          0        0.0  34654.71     22.33      22.23   
1       00:05          0        0.0  34654.79     22.33      22.23   
2       00:10          0        0.0  34654.88     22.31      22.23   
3       00:15          0        0.0  34654.96     22.33      22.23   
4       00:20          0        0.0  34655.05     22.27      22.13   
..        ...        ...        ...       ...       ...        ...   
278     23:10          0        0.0  34677.69     22.43      22.13   
279     23:15          0        0.0  34677.78     22.39      22.15   
280     23:20          0        0.0  34677.86     22.29      21.85   
281     23:25          0        0.0  34677.94     22.09      21.45   
282     23:30          0        0.0  34678.03     22.05      21.15   

     WindVel km/h  A.Ms.Amp  A.Ms.Vol  A.Ms.Watt  ...  Mode  Mt.TotOpTmh  
\
0           11.86       NaN       NaN        NaN  ...   NaN          NaN   
1            4.38       NaN       NaN        NaN  ...   NaN          NaN   
2            5.08       NaN       NaN        NaN  ...   NaN          NaN   
3            6.46       NaN       NaN        NaN  ...   NaN          NaN   
4            4.34       NaN       NaN        NaN  ...   NaN          NaN   
..            ...       ...       ...        ...  ...   ...          ...   
278          6.04       NaN       NaN        NaN  ...   NaN          NaN   
279          5.22       NaN       NaN        NaN  ...   NaN          NaN   
280          4.38       NaN       NaN        NaN  ...   NaN          NaN   
281          6.24       NaN       NaN        NaN  ...   NaN          NaN   
282          3.68       NaN       NaN        NaN  ...   NaN          NaN   

     Mt.TotTmh  Op.EvtCntUsr  Op.EvtNo  Op.GriSwStt  Op.TmsRmg  Pac  \
0          NaN           NaN       NaN          NaN        NaN  NaN   
1          NaN           NaN       NaN          NaN        NaN  NaN   
2          NaN           NaN       NaN          NaN        NaN  NaN   
3          NaN           NaN       NaN          NaN        NaN  NaN   
4          NaN           NaN       NaN          NaN        NaN  NaN   
..         ...           ...       ...          ...        ...  ...   
278        NaN           NaN       NaN          NaN        NaN  NaN   
279        NaN           NaN       NaN          NaN        NaN  NaN   
280        NaN           NaN       NaN          NaN        NaN  NaN   
281        NaN           NaN       NaN          NaN        NaN  NaN   
282        NaN           NaN       NaN          NaN        NaN  NaN   

     PlntCtl.Stt  Serial Number  
0            NaN            NaN  
1            NaN            NaN  
2            NaN            NaN  
3            NaN            NaN  
4            NaN            NaN  
..           ...            ...  
278          NaN            NaN  
279          NaN            NaN  
280          NaN            NaN  
281          NaN            NaN  
282          NaN            NaN  

[283 rows x 362 columns]
```

Result

We can see from the initial result that we have roughly 24 hours of data available, with 362 columns of data points to choose from. Deciding which columns to keep may prove challenging.

unswDataImportArray

Following from unswDataImportSingle, we now are going to try import the entirety of the 2018-Array folder. We can select our chosen headers using a list comprehension, choosing to keep "Timestamp, TmpAmb, WindVel, and A.Ms.Watt" as (0, 4, 6, 9). Using glob.iglob, we can create an iterable for the path of all the relevant files in the 2018-Array folder. By specifying in pd.read_csv the usecols=headers, we make sure to only import the relevant headers (and ignoring the other 357 that we don't want) which reduces import time and memory usage. After this we can set the indices to the relevant datetime that the data was captured at, making sure we get a datetime64 result for later manipulation.

I've truncated the headers string to only the first 10 or so elements to reduce its space, however if you require the full headers to explore the 350 other columns please refer to either the excel file header, or unswDataImportSingle.

In [2]:

```
import pandas as pd
import glob
import os

#file_name = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\201
8\2018-01-01.csv"
#file_name = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\84-Site_12-B
P-Solar.csv"

headers = "TimeStamp;ExlSolIrr;IntSolIrr;SMA-h-On;TmpAmb C;TmpMdul C;WindVel km/h;A.Ms.
Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp"

headers = dict(enumerate(headers.split(';')))
headers = {k: headers[k] for k in (0, 4, 6, 9)} # choosing which headers we want
for item in headers:
    print(item, headers[item])

path = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Arra
y"
all_files = glob.iglob(os.path.join(path, "*csv"))
df1 = pd.concat((pd.read_csv(f, delimiter=";", header=None, skiprows=6, usecols=headers
).assign(filename = os.path.basename(f)) for f in all_files))
#df1 = pd.read_csv(file_name, delimiter=";", header=None, skiprows=6, usecols=headers)

df1 = df1.rename(columns = headers)
df1.index = df1['filename'].str.split('.', expand = True)[0] + " " + df1['TimeStamp']
df1 = df1.drop(columns = ['filename'])
df1.index = pd.to_datetime(df1.index)
print(df1)
df1.info()
```

```
0 TimeStamp
4 TmpAmb C
6 WindVel km/h
9 A.Ms.Watt
                    TimeStamp  TmpAmb C  WindVel km/h  A.Ms.Watt
2018-01-01 00:00:00     00:00     22.33         11.86        NaN
2018-01-01 00:05:00     00:05     22.33          4.38        NaN
2018-01-01 00:10:00     00:10     22.31          5.08        NaN
2018-01-01 00:15:00     00:15     22.33          6.46        NaN
2018-01-01 00:20:00     00:20     22.27          4.34        NaN
...                       ...       ...           ...        ...
2018-12-17 09:10:00     09:10     29.47          0.00    3069.00
2018-12-17 09:15:00     09:15     29.29          0.00    3173.80
2018-12-17 09:20:00     09:20     28.93          0.00    3282.80
2018-12-17 09:25:00     09:25     29.81          0.00    3382.80
2018-12-17 09:30:00     09:30     29.25          0.00    3443.75

[87540 rows x 4 columns]
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 87540 entries, 2018-01-01 00:00:00 to 2018-12-17 09:30:00
Data columns (total 4 columns):
TimeStamp       87540 non-null object
TmpAmb C        76721 non-null float64
WindVel km/h    76721 non-null float64
A.Ms.Watt       49920 non-null float64
dtypes: float64(3), object(1)
memory usage: 3.3+ MB
```

Result

We can see we now only have the 4 resulting columns, rather than the 361 we had originally. We know that there is definitively missing data as shown by the non-null counts in each column. This shouldn't pose too much of an issue as missing A.Ms.Watt data is primary during the night time, and missing data point can otherwise be imputed.

unswDataImportWeather

Following attempting to import the array data, we now need the weather data for 2018. The weather data is in a completely different format to the Array data (conveniently easier to handle for pandas). Using the same techniques in our Array import we can attempt to bring across the weather data. However, some initial complications revealed some stark issues with the dataset: Dated files could sometimes contain 0KB of data and would break Pandas, and some dated files (dated only for a single day) sometimes contained up to 5 days of weather data.

We deal with both these file issues below, firstly with a try, except statement and simply ignoring any data that can't easily be extracted. The second issue of the multiple days in a single dated file is solved almost for us by pandas and the standardised datetime format in the file (when viewing in excel, it only has minutes:seconds, however when imported throuhg pandas correctly contains yyyy/mm/dd hh/mm/ss/milliseconds) which allows us to assign the data properly to it's date after extraction

In [1]:

```
import pandas as pd
import glob
import os

#file_name = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\201
8\2018-01-01.csv"
#file_name = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\84-Site_12-B
P-Solar.csv"

headers = "Timestamp,TZ,01Tpvtg_in (oC),02Tpvtg_out (oC),03Ttankg_in (oC),04Ttankg_out
 (oC),05Ttankg (oC),07Tpvtug_in (oC),08Tpvtug_out (oC),09Ttankug_in (oC),10Ttankug_out
 (oC),11Ttankug (oC),06Flowg,12Flowug,(IR02)T (oC),(SPN1)G_ht (W/m2),(SPN1)G_hd (W/m2),
(SR12)G_tilt (W/m2),(IR02)U/S (W/m2)"
headers = dict(enumerate(headers.split(',')))
headers = {k: headers[k] for k in (0, 15, 16)}
for item in headers:
    print(item, headers[item])
path = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weat
her"
all_files = glob.iglob(os.path.join(path, "*csv"))


li = []

for f in all_files:
    try:
        df = pd.read_csv(f, header=0, usecols=headers).assign(filename = os.path.basena
me(f))
        li.append(df)
        break #remove to attempt all files
    except:
        print("failed: " + f)

df1 = pd.concat(li, axis=0)
df1 = df1.rename(columns = {'Timestamp':'timestamp', '(SPN1)G_ht (W/m2)':'GHI', '(SPN1)
G_hd (W/m2)':'DHI'})

#df1 = pd.concat((pd.read_csv(f, delimiter=";", header=None, skiprows=6, usecols=header
s).assign(filename = os.path.basename(f)) for f in all_files))
#df1 = pd.read_csv(file_name, delimiter=";", header=None, skiprows=6, usecols=headers)

'''df1 = df1.rename(columns = headers)
df1.index = df1['filename'].str.split('.', expand = True)[0] + " " + df1['TimeStamp']
df1 = df1.drop(columns = ['filename'])'''
print("at datetime")
df1.info()
print(df1)
#df1.index = pd.to_datetime(df1.timestamp, errors='coerce')

df1['timestamp'] = pd.to_datetime(df1['timestamp'].map(lambda x: '.'.join(str(x).split(
'.')[:-1])))
df1.index = df1['timestamp']
print(df1)
df1.info()
```

```
0 Timestamp
15 (SPN1)G_ht (W/m2)
16 (SPN1)G_hd (W/m2)
at datetime
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5754 entries, 0 to 5753
Data columns (total 4 columns):
timestamp    5754 non-null object
GHI          5754 non-null float64
DHI          5754 non-null float64
filename     5754 non-null object
dtypes: float64(2), object(2)
memory usage: 179.9+ KB
                    timestamp        GHI        DHI                filena
me
0     2018/01/17 00:00:15.000  36.318780  32.853374  000_20180118T000000.C
SV
1     2018/01/17 00:00:30.000  35.687920  32.845600  000_20180118T000000.C
SV
2     2018/01/17 00:00:45.000  36.332024  32.874088  000_20180118T000000.C
SV
3     2018/01/17 00:01:00.000  38.123100  34.693352  000_20180118T000000.C
SV
4     2018/01/17 00:01:15.001  38.755364  35.947684  000_20180118T000000.C
SV
...                       ...        ...        ...
...
5749  2018/01/17 23:59:00.002   2.138962   1.270248  000_20180118T000000.C
SV
5750  2018/01/17 23:59:15.000   1.756393   0.709045  000_20180118T000000.C
SV
5751  2018/01/17 23:59:30.000   2.074356   1.308228  000_20180118T000000.C
SV
5752  2018/01/17 23:59:45.000   1.919243   1.022835  000_20180118T000000.C
SV
5753  2018/01/18 00:00:00.007   1.687950   0.910660  000_20180118T000000.C
SV

[5754 rows x 4 columns]
                              timestamp        GHI        DHI  \
timestamp
2018-01-17 00:00:15 2018-01-17 00:00:15  36.318780  32.853374
2018-01-17 00:00:30 2018-01-17 00:00:30  35.687920  32.845600
2018-01-17 00:00:45 2018-01-17 00:00:45  36.332024  32.874088
2018-01-17 00:01:00 2018-01-17 00:01:00  38.123100  34.693352
2018-01-17 00:01:15 2018-01-17 00:01:15  38.755364  35.947684
...                                 ...        ...        ...
2018-01-17 23:59:00 2018-01-17 23:59:00   2.138962   1.270248
2018-01-17 23:59:15 2018-01-17 23:59:15   1.756393   0.709045
2018-01-17 23:59:30 2018-01-17 23:59:30   2.074356   1.308228
2018-01-17 23:59:45 2018-01-17 23:59:45   1.919243   1.022835
2018-01-18 00:00:00 2018-01-18 00:00:00   1.687950   0.910660

                                    filename
timestamp
2018-01-17 00:00:15  000_20180118T000000.CSV
2018-01-17 00:00:30  000_20180118T000000.CSV
2018-01-17 00:00:45  000_20180118T000000.CSV
2018-01-17 00:01:00  000_20180118T000000.CSV
2018-01-17 00:01:15  000_20180118T000000.CSV
...                                      ...
```

```
2018-01-17 23:59:00  000_20180118T000000.CSV
2018-01-17 23:59:15  000_20180118T000000.CSV
2018-01-17 23:59:30  000_20180118T000000.CSV
2018-01-17 23:59:45  000_20180118T000000.CSV
2018-01-18 00:00:00  000_20180118T000000.CSV

[5754 rows x 4 columns]
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5754 entries, 2018-01-17 00:00:15 to 2018-01-18 00:00:00
Data columns (total 4 columns):
timestamp    5754 non-null datetime64[ns]
GHI          5754 non-null float64
DHI          5754 non-null float64
filename     5754 non-null object
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 224.8+ KB
```

Result

As a result of the weather extraction, we end up with (maybe) the Timestamp, GHI, and DHI which are required further down the pipeline. Pandas has some issue converting the datetime object from the weather file to a proper datetime64 type due to containing millisecond data, so I used a string manipulation workaround to 'round' off the millisecond component. We can see that we properly have a datetime64[ns] aligning in dtype to the Array data we processed earlier. This datetime data however is at 15 second intervals, unlike the 5 minute intervals of the Array data which is an issue we will overcome later.

unswDataImportArray&Weather

The culmination of our previous work comes here where we try to combine the Array data import and the Weather import for 2018. We have some issues to overcome especially now that we're beginning to handle a bit more data than my computer can easily throw around, but that's not to say it can't be done!

Array

To start we're going to repeat the same set of code from unswDataImportArray with some slight changes. Despite the length of repeating this code, it won't be omitted for reproducability purposes.

In [1]:

```
import pandas as pd
import glob
import os

headers = "TimeStamp;ExlSolIrr;IntSolIrr;SMA-h-On;TmpAmb C;TmpMdul C;WindVel km/h;A.Ms.
Amp;A.Ms.Vol;A.Ms.Watt;A1.Ms.Amp"

headers = dict(enumerate(headers.split(';')))
headers = {k: headers[k] for k in (0, 4, 6, 9)} # choosing which headers we want
for item in headers:
    print(item, headers[item])

path = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Arra
y"
all_files = glob.iglob(os.path.join(path, "*csv"))
df1 = pd.concat((pd.read_csv(f, delimiter=";", header=None, skiprows=6, usecols=headers
).assign(filename = os.path.basename(f)) for f in all_files))

df1 = df1.rename(columns = headers)
df1.index = df1['filename'].str.split('.', expand = True)[0] + " " + df1['TimeStamp']
df1 = df1.drop(columns = ['filename'])
df1.index = pd.to_datetime(df1.index)
print(df1)
df1.info()
```

```
0 TimeStamp
4 TmpAmb C
6 WindVel km/h
9 A.Ms.Watt
                    TimeStamp  TmpAmb C  WindVel km/h  A.Ms.Watt
2018-01-01 00:00:00     00:00     22.33         11.86        NaN
2018-01-01 00:05:00     00:05     22.33          4.38        NaN
2018-01-01 00:10:00     00:10     22.31          5.08        NaN
2018-01-01 00:15:00     00:15     22.33          6.46        NaN
2018-01-01 00:20:00     00:20     22.27          4.34        NaN
...                       ...       ...           ...        ...
2018-12-17 09:10:00     09:10     29.47          0.00    3069.00
2018-12-17 09:15:00     09:15     29.29          0.00    3173.80
2018-12-17 09:20:00     09:20     28.93          0.00    3282.80
2018-12-17 09:25:00     09:25     29.81          0.00    3382.80
2018-12-17 09:30:00     09:30     29.25          0.00    3443.75

[87540 rows x 4 columns]
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 87540 entries, 2018-01-01 00:00:00 to 2018-12-17 09:30:00
Data columns (total 4 columns):
TimeStamp       87540 non-null object
TmpAmb C        76721 non-null float64
WindVel km/h    76721 non-null float64
A.Ms.Watt       49920 non-null float64
dtypes: float64(3), object(1)
memory usage: 3.3+ MB
```

Weather

So now that we have our Array data imported, we have to merge the weather data to the Array table without breaking anything. In the process of doing this, I did break everything, many times. There are some critical changes to the code, being first that instead of creating a new weather table for everything, we are simply attempting to put it straight into the TEBT-Array dataframe in order to save memory.

The key lines to doing this is the "firstMerge" on df1.merge, and all subsequent df1.update(df2). These functions were quite difficult to find as merge, combine_first, and update all do small variations on the same idea. When attempting to use only df1.merge I resulted with a 1.4GB+ dataframe and about 10 minutes of computation time, only to realise instead of having 7 comlumns as a result that I had over 2000. Merge forces new columns to be made from the right_merge into the left_merge, so every merge was adding 3-4 columns. df1.update overcomes this by replacing the NaN's in already existing columns from columns of the same name in df2, overcoming the issue.

We also see that this time, many files didn't make the cut for the dataframe, and failed in the import, but half of these are 0KB files with no header, and others are corrupted in another way that I didn't investiage, so there is little issue.

In [2]:

```
headers = "Timestamp,TZ,01Tpvtg_in (oC),02Tpvtg_out (oC),03Ttankg_in (oC),04Ttankg_out
 (oC),05Ttankg (oC),07Tpvtug_in (oC),08Tpvtug_out (oC),09Ttankug_in (oC),10Ttankug_out
 (oC),11Ttankug (oC),06Flowg,12Flowug,(IR02)T (oC),(SPN1)G_ht (W/m2),(SPN1)G_hd (W/m2),
(SR12)G_tilt (W/m2),(IR02)U/S (W/m2)"
headers = dict(enumerate(headers.split(',')))
headers = {k: headers[k] for k in (0, 15, 16)}
for item in headers:
    print(item, headers[item])
path = r"C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weat
her"
all_files = glob.iglob(os.path.join(path, "*csv"))

firstMerge = True

for f in all_files:
    try:
        df2 = pd.read_csv(f, header=0, usecols=headers)
        df2 = df2.rename(columns = {'(SPN1)G_ht (W/m2)':'GHI', '(SPN1)G_hd (W/m2)':'DH
I'})
        df2['Timestamp'] = pd.to_datetime(df2['Timestamp'].map(lambda x: '.'.join(str(x
).split('.')[:-1])))
        df2.index = df2['Timestamp']
        if firstMerge:
            df1 = df1.merge(df2, how='left', left_index=True, right_index=True, validat
e="one_to_many")
            firstMerge = False
        #df1.combine_first(df2)
        df1.update(df2)
        #break #remove to attempt all files
    except:
        print("failed: " + f)

df1.info()
print(df1)
```

```
0 Timestamp
15 (SPN1)G_ht (W/m2)
16 (SPN1)G_hd (W/m2)
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180124T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180307T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180308T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180404T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180502T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180503T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180515T000000.CSV

C:\Users\Clairvoyant Cabbage\Documents\PythonProject\venv\lib\site-package
s\IPython\core\interactiveshell.py:3058: DtypeWarning: Columns (15,16) hav
e mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\000_20180605T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\001_20180309T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\001_20180405T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\001_20180504T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\002_20180310T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\002_20180406T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\003_20180407T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\004_20180408T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\004_20180512T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\005_20180123T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\005_20180409T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\005_20180513T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\006_20180410T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\007_20180411T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\008_20180412T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\009_20180413T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\010_20180414T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWDa
ta\2018-Weather\235_20190126T000000.CSV
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 87540 entries, 2018-01-01 00:00:00 to 2018-12-17 09:30:00
Data columns (total 7 columns):
TimeStamp       87540 non-null object
TmpAmb C        76721 non-null float64
WindVel km/h    76721 non-null float64
A.Ms.Watt       49920 non-null float64
Timestamp       75368 non-null datetime64[ns]
GHI             75367 non-null float64
DHI             75367 non-null object
dtypes: datetime64[ns](1), float64(4), object(2)
memory usage: 7.8+ MB
                    TimeStamp  TmpAmb C  WindVel km/h  A.Ms.Watt  \
2018-01-01 00:00:00     00:00     22.33         11.86        NaN
2018-01-01 00:05:00     00:05     22.33          4.38        NaN
2018-01-01 00:10:00     00:10     22.31          5.08        NaN
2018-01-01 00:15:00     00:15     22.33          6.46        NaN
2018-01-01 00:20:00     00:20     22.27          4.34        NaN
...                       ...       ...           ...        ...
2018-12-17 09:10:00     09:10     29.47          0.00    3069.00
2018-12-17 09:15:00     09:15     29.29          0.00    3173.80
2018-12-17 09:20:00     09:20     28.93          0.00    3282.80
2018-12-17 09:25:00     09:25     29.81          0.00    3382.80
2018-12-17 09:30:00     09:30     29.25          0.00    3443.75
```

```
                              Timestamp        GHI      DHI
2018-01-01 00:00:00 2018-01-01 00:00:00 -22.640380  -28.733
2018-01-01 00:05:00 2018-01-01 00:05:00 -22.004570 -28.3443
2018-01-01 00:10:00 2018-01-01 00:10:00 -21.189972 -26.4709
2018-01-01 00:15:00 2018-01-01 00:15:00 -20.946122 -26.4408
2018-01-01 00:20:00 2018-01-01 00:20:00 -19.850174 -24.6612
...                                 ...        ...      ...
2018-12-17 09:10:00                 NaT        NaN      NaN
2018-12-17 09:15:00                 NaT        NaN      NaN
2018-12-17 09:20:00                 NaT        NaN      NaN
2018-12-17 09:25:00                 NaT        NaN      NaN
2018-12-17 09:30:00                 NaT        NaN      NaN

[87540 rows x 7 columns]
```

Using the data

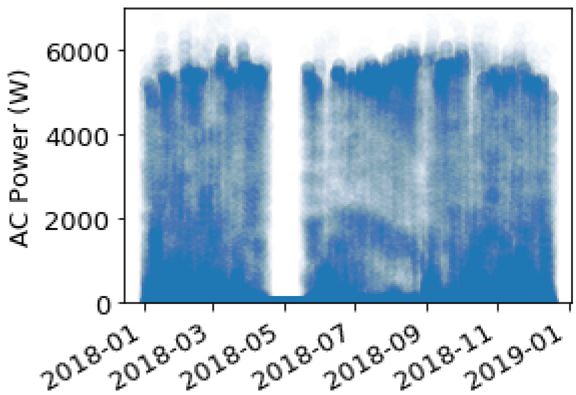
After getting a combined Array-Weather dataframe, I then attempted to use it for any of the processing methodologies available. I opted to try STL as RDTools-YOY requires more than one year of data (we only have 2018 imported) and weather for all years (only weather data for 2017-18 available as current on unsw servers). Doubtful that STL would simply work straight up we try to execute it, but naturally it throws us an error.

Viewing the Power from the dataframe, we can see that somewhere in the import we have zeroes. The files for this period definitely exist, however I am still identifying the cause and solution. Otherwise, the Power looks completely normal, very much like our test set of Desert Knowledge unit 12 which we used to RDTools test.

In [3]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
#sns.set_style('darkgrid')
%matplotlib inline

plt.rc('figure',figsize=(16,12))
plt.rc('font',size=13)

freq = pd.infer_freq(df1.index[:10])
df1 = df1.resample(freq).median()

# plot the AC power time series
fig, ax = plt.subplots(figsize=(4,3))
ax.plot(df1.index, df1['A.Ms.Watt'], 'o', alpha = 0.01)
ax.set_ylim(0,7000)
fig.autofmt_xdate()
ax.set_ylabel('AC Power (W)');

try:
    from statsmodels.tsa.seasonal import STL
    stl = STL(df1['A.Ms.Watt'], seasonal=13)
    res = stl.fit()
    fig = res.plot()
    print("done")
except AssertionError as error:
    print(error)
    print("STL calculation failed")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-0b0c0c47621e> in <module>
     21 try:
     22     from statsmodels.tsa.seasonal import STL
---> 23     stl = STL(df1['A.Ms.Watt'], seasonal=13)
     24     res = stl.fit()
     25     fig = res.plot()

statsmodels\tsa\_stl.pyx in statsmodels.tsa._stl.STL.__init__()

~\Documents\PythonProject\venv\lib\site-packages\statsmodels\tsa\tsatools.py in freq_to_period(freq)
    813     else:  # pragma : no cover
    814         raise ValueError("freq {} not understood. Please report if you "
--> 815                          "think this is in error.".format(freq))
    816 
    817 

ValueError: freq T not understood. Please report if you think this is in error.
```



2. Reflection on Progress

Challenges

The reflections on issues faced over the process of building this project are primarily explained with the challenge or solution relevant in the Jupyter notebooks, however I will summarise some of the issues faced.

When it came to importing the data from UNSW TEBT, overcoming the awkward file composition, missing data and corrupted files was quite the learning experience. After choosing to include some tolerance of failure for the import as shown in fig 7, I see that trying to get absolutely everything to work would have been unfeasible, as somewhat expected. Of the 350 or so weather files (which we could only hope contained 365 days of data) around 20 or so failed on any attempt, sometimes changing depending on what data was being asked from it. Any 0KB files always failed, but depending on what columns were requested, or what dtypes pandas thought the column was, sometimes more or less files were accepted, but this was always consistent when using the same process.

```
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\000_20180308T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\000_20180421T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\001_20180309T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\001_20180405T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\001_20180422T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\001_20180504T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\002_20180310T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\002_20180406T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\003_20180407T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\004_20180408T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\004_20180512T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\005_20180409T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\005_20180513T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\006_20180410T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\006_20180427T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\007_20180411T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\008_20180412T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\009_20180413T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\010_20180414T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\015_20180419T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\027_20180702T000000.CSV
failed: C:\Users\Clairvoyant Cabbage\Documents\PythonProject\Thesis\UNSWData\2018-Weather\235_20190126T000000.CSV
```

Fig 7: Importing weather data files, sometimes things just don’t work

Failing at getting STL to work on my 1 year TEBT dataset was quite disheartening, but this is still in early works and hopefully can be remedied. Some of the other coding challenges were quite enjoyable, if time consuming, the more minor of which can be found in the Jupyter notebooks as there are too many more to list.

Differences in Research and Learning

This thesis project has definitely changed direction and momentum over the course of its life. Originally on the case of my literature review, a large section of my personal research was into degradation failure modes and more into the physics based models for identifying degradation sources. These topics were somewhat common when discussing with other students, postgraduates, and professors. Upon finding the analytics and statistics work being done by the international community, the project completely shifted and the skills required more suited to what I had in mind when I originally was deciding my thesis project.

Now that the project is less focused on physics and failure mechanisms I have more freedom to learn about the statistics and methodologies behind the analysis, rather than the science behind the degradation modes (although this was initially done at the lit review stage). In this way, I've completely changes the angle of approach for the thesis to something I find more satisfying.

The literature presented when doing both my lit review and project replanning are quite undiscussed during internal courses. Some talks and seminars have discussed these areas of knowledge however. When I was learning the information required, almost all information was completely new to me (although perhaps some bias towards ignoring the context I already have), and I am glad that this research has broadened my skill set.

3. Revised Project Plan

The project plan is still changing a little as goals are being reached, or not reached. After my literature review the 'Initial plan' and goals were mostly scrapped for the 'Recalibrated Plan' and goals already. As we are now reaching the end of thesis B some further planning is required, and updates to my recalibrated plan must be made to accomodate thesis C. The thesis C plan will include a significant portion of time attributed to writing in order to develop a cleaner final report.

Initial Goals and timeline

Primary Goals

- Determine whether Clear-sky YOY or RPCA is a more effective methodology
- Determine degradation rate of UNSW TETB array using Clear-sky YOY and RPCA

Secondary Goals

- Determine effectiveness of data cleaning methods used in degradation analysis
- Determine effectiveness of data fitting methods used in degradation analysis
- Observe non-linear degradation in analysis
- Determine soiling rates of UNSW PV arrays based on tilt

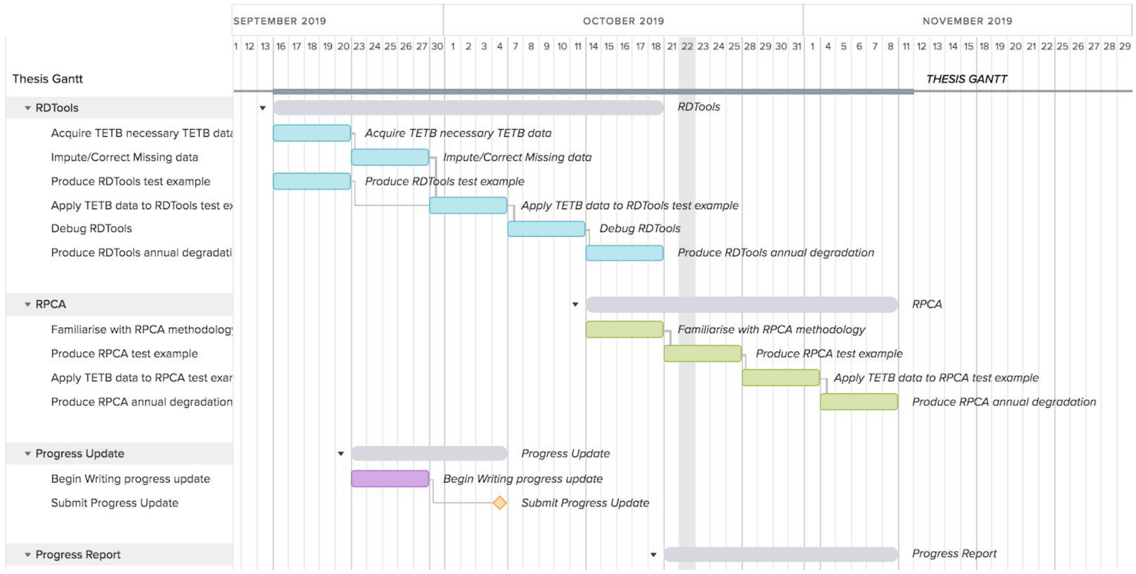


Fig 8: Initial Plan

Recalibrated Goals and timeline

Primary Goals

- Implement standard performance loss methodologies (LR, STL, YOY)
- Apply performance loss methodologies to UNSW TEBT data

Secondary Goals

- Determine effectiveness of data cleaning methods used in PL analysis
- Determine effectiveness of data fitting methods used in PL analysis
- Observe non-linear PL in analysis
- Explore alternative methodologies as time permits

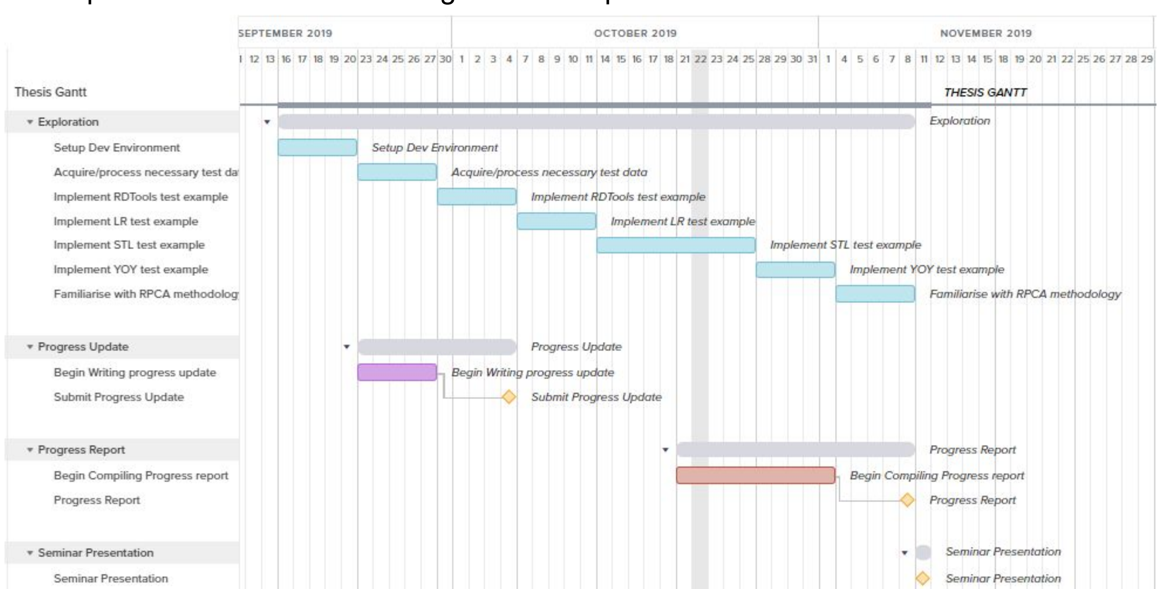


Fig 9: Recalibrated Plan

Thesis C plan

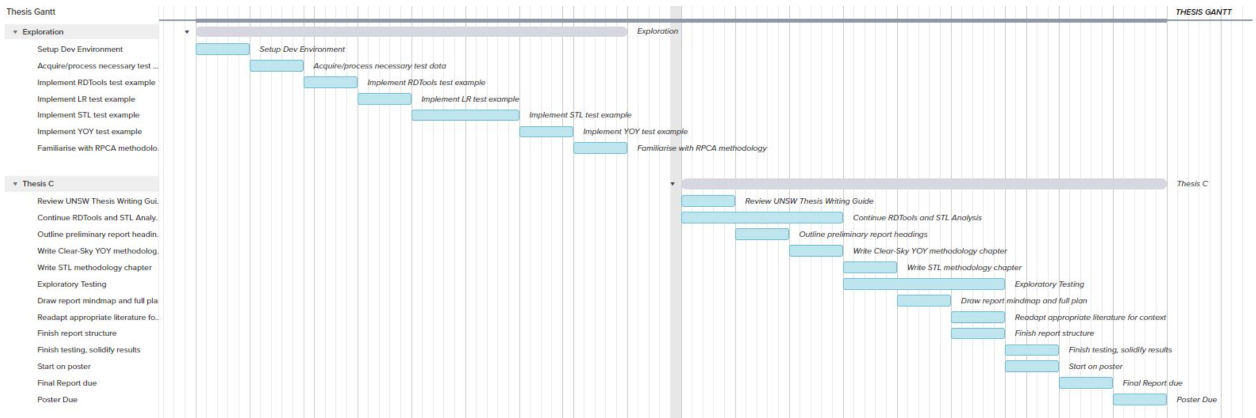


Fig 10: Recalibrated + Thesis C plan