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]:

```python
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.