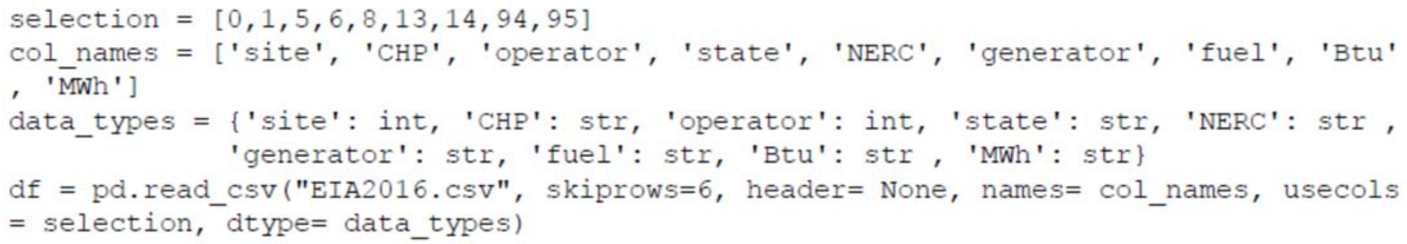
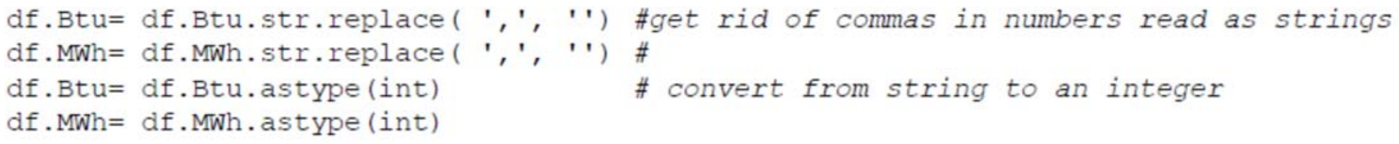
Project Capstone Project Data Wrangling

Data Wrangling Steps Undertaken: The first data wrangling challenge undertaken was simply to read into a Pandas data-frame the voluminous Excel files which are used to store the subject EIA (Energy Information Agency) Form 923 data. The EIA had warned me beforehand that the files were difficult to read and that others had not had luck in this respect before. I even came across a blog post by a California-based Data Science Institute that complained publicly about the problems of reading the EIA files and sought ideas. I subsequently spent than thirty hours on this challenging step alone. The RAM memory in my computer was initially not large enough to read the file, but by selecting a limited number of columns (see "*selection*" below), and by creating a dictionary that specified the data types of the selected columns (see "*data_types*" below), reading the file became possible (specifically by using the code below):



```
selection = [0,1,5,6,8,13,14,94,95]
col_names = ['site', 'CHP', 'operator', 'state', 'NERC', 'generator', 'fuel', 'Btu'
, 'MWh']
data_types = {'site': int, 'CHP': str, 'operator': int, 'state': str, 'NERC': str ,
              'generator': str, 'fuel': str, 'Btu': str , 'MWh': str}
df = pd.read_csv("EIA2016.csv", skiprows=6, header= None, names= col_names, usecols
= selection, dtype= data_types)
```

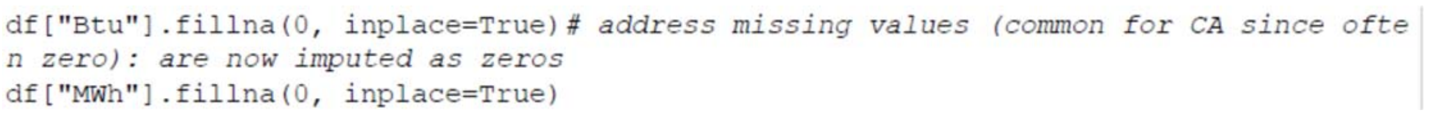
Another challenge is that the Excel data uses commas as separators to denote thousands, but the standard optional argument within the read_excel( ) approach did not work for various reasons (a whole memo was written on just this topic). Accordingly the desired numerical data was read in first as strings, and then using the str.replace() command in Pandas, the troublesome commas were removed: the data type was then changed via the *.astype(int)* method (see the relevant code excerpt below).



```
df.Btu= df.Btu.str.replace( ',', '') #get rid of commas in numbers read as strings
df.MWh= df.MWh.str.replace( ',', '') #
df.Btu= df.Btu.astype(int)           # convert from string to an integer
df.MWh= df.MWh.astype(int)
```

A further challenge is that the EIA data had been entered manually, and was therefore subject to human error. In just a couple of cases (among many thousands, so veritable needles in the proverbial haystack) an omitted or a zero value was erroneously represented by a colon or a semi-colon, rather than a zero or just a blank/whitespace. For example, most of the columns enumerated above are clearly numeric variables, but they could not be read in this manner as the few stray colons, semi-colons etc. created effectively a mixed data type. This forced the data to be read in initially as strings (as mentioned above). A series of exploratory analyses identified the problem of the stray colons etc., and these were addressed by creating a dictionary which changed them to null values.

Next the "suitably-corrected" missing values were set to zero via the .fillna() method, as shown below:



```
df["Btu"].fillna(0, inplace=True)# address missing values (common for CA since ofte
n zero): are now imputed as zeros
df["MWh"].fillna(0, inplace=True)
```

Then the selected variables could be safely converted to the correct Pandas type (mostly categorical) in the manner shown by the code excerpt below:

```
# Convert the remaining variables to Pandas type category
# for col in ['site', 'CHP', 'operator', 'state', 'NERC', 'fuel','generator']:# mor
e concise w/o repetition
#    df[col] = df[col].astype('category')
df["site"] = df["site"].astype('category')
df["CHP"] = df["CHP"].astype('category')
df["operator"] = df["operator"].astype('category')
df["state"] = df["state"].astype('category')
df["NERC"] = df["NERC"].astype('category')
df["fuel"] = df["fuel"].astype('category')
df["generator"] = df["generator"].astype('category')
df.dtypes
```

The focus of this Study was for (a) non-CHP power plants, (b) in Texas (“TX”), (c) for which the fuel was natural gas (coded by the EIA as “NG”). The data wrangling code for this is excerpted below:

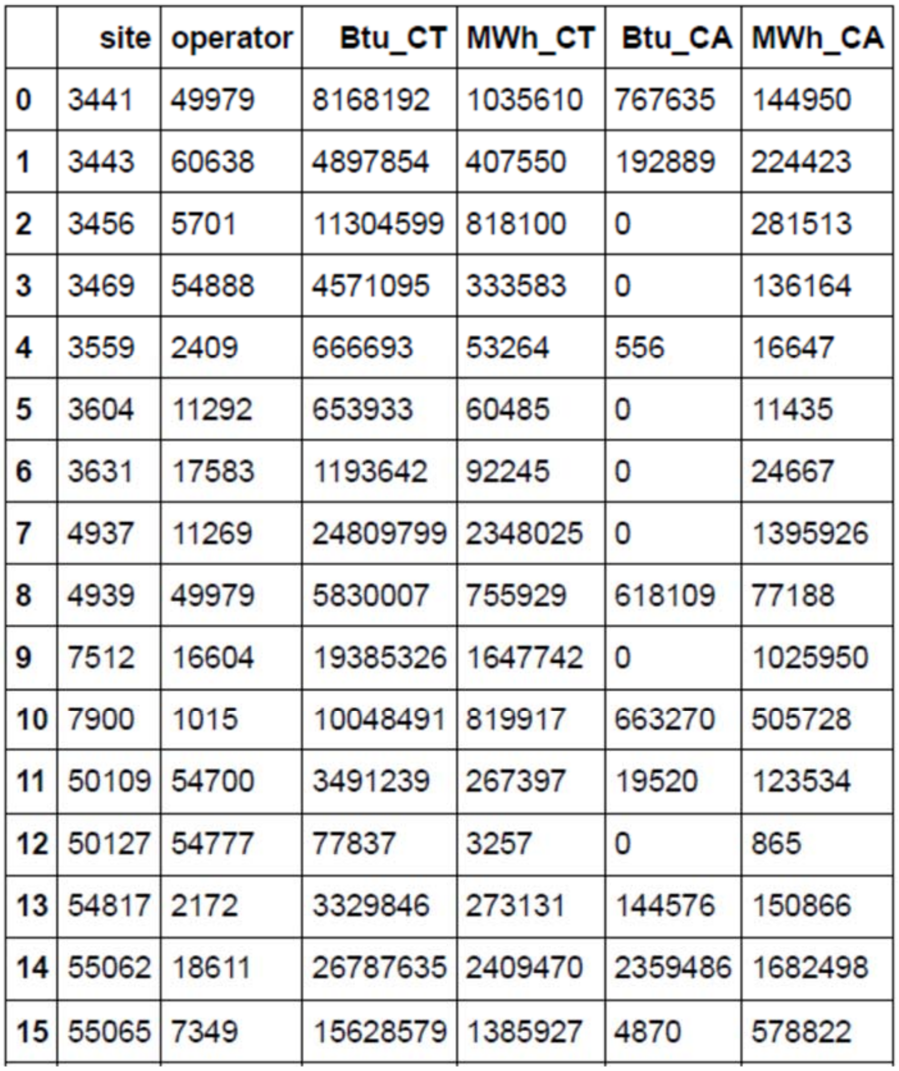
```
m_CHP = df.CHP == "N" # m_ abbreviation  for "mask"
m_state = df.state == "TX"
m_fuel= df.fuel == "NG"
```

One fundamental problem with the EIA data is that the subject *Combined Cycle Gas Turbine* power plants are not designated as an integrated unit, but rather the *Combustion Turbine* (coded as “CT”) and the associated *Steam Turbine* (coded as “CA”) are separately identified (ane are on separate rows). Accordingly two data frames are created “side-by-side” (for respectively the CT and the CA units and are then horizontally merged (according to the unit’s unique “site” ID code). The data wrangling code for this is excerpted below:

```
m_CA = df.generator== 'CA'# a CCGT is an integrated unit with usually one "CT" (Com
bustion Turbine) and one "CA".
m_CT = df.generator== 'CT'# we need to seperate out the CT and its matching CA and
then re-integrate them as a unit.
CCTX= df[m_CHP & m_state & m_fuel & (m_CA | m_CT)]
CCTX.head(8)
CCTX.drop(CCTX.columns[[1,3,4,6]], axis=1, inplace=True) # drop now-redundant colum
ns before the CT/CA Merge
CCTX.head(4)  #.head(4)

# perhaps first set Site# as the Index?
CAdf= CCTX[CCTX["generator"] == "CA"]
len(CAdf) # 43 plants w CA, but the last is a fuel adjustment rather than a plant
CAdf.head(4)
CTdf= CCTX[CCTX["generator"] == "CT"]
len(CTdf)  # also 43 plants w CT, and perfect matches on "Site" seem to be the case
(so Site can be safe as the index!)
CTdf.head(4)
CTdf.info()
CACTdf= CTdf.merge(CAdf, how = "inner", on= "site")# , suffixes= ("_CT", "_CA) didn
t Run!
CACTdf.head(40)# Since the match is indeed perfect we can drop columns and relabel
remaining columns
CACTdf.drop(CACTdf.columns[[2,5,6]], axis=1, inplace=True)
CACTdf.head(4)
column_indices = [0,1,2,3,4,5]# This dict approach does work
new_names = ["site","operator", "Btu_CT","MWh_CT", "Btu_CA","MWh_CA"]
old_names = CACTdf.columns[column_indices]
CACTdf.rename(columns=dict(zip(old_names, new_names)), inplace=True)
```

A certain amount of Exploratory Data Analysis (EDA) was then undertaken to verify that the approach adopted did not yield any unpleasant surprises and was in fact doing what it was supposed to be doing. An “*Inner Join*” was used (see the above code excerpt) using the “site” index, to make sure there was a strict one-to-one correspondence. This the yielded the desired consolidation of the separate units into synthetically created Combined Cycle Gas Turbines (see below for the initial fifteen rows).



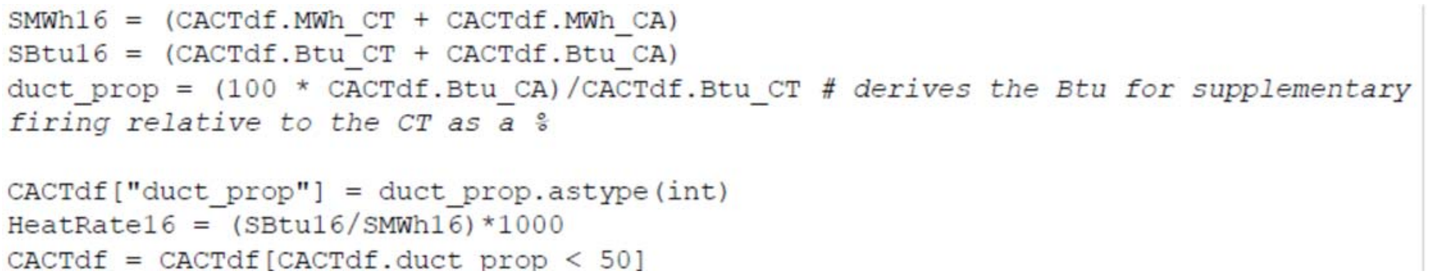
	site	operator	Btu_CT	MWh_CT	Btu_CA	MWh_CA
0	3441	49979	8168192	1035610	767635	144950
1	3443	60638	4897854	407550	192889	224423
2	3456	5701	11304599	818100	0	281513
3	3469	54888	4571095	333583	0	136164
4	3559	2409	666693	53264	556	16647
5	3604	11292	653933	60485	0	11435
6	3631	17583	1193642	92245	0	24667
7	4937	11269	24809799	2348025	0	1395926
8	4939	49979	5830007	755929	618109	77188
9	7512	16604	19385326	1647742	0	1025950
10	7900	1015	10048491	819917	663270	505728
11	50109	54700	3491239	267397	19520	123534
12	50127	54777	77837	3257	0	865
13	54817	2172	3329846	273131	144576	150866
14	55062	18611	26787635	2409470	2359486	1682498
15	55065	7349	15628579	1385927	4870	578822

From this excerpt we can now see the fuel consumed (Btu_CT) and the power generated (MWh_CT) at each CT, and we can also view the fuel consumed (Btu_CA) and the power generated (MWh_CA) at each CA.

```
# Remove single tiny outlier plant. eg Remove row of dataframe with df.drop(df.inde
x[[2,3]])or bool min(MWh)
CACTdf.drop(CACTdf.index[12], inplace= True) # another approach meat.ix[meat.beef.i
dxmax()]
# CACTdf= CACTdf.drop(CACTdf.index[MWh_CT.idxmin()])#
CACTdf.drop(CACTdf.index[-1], inplace= True)#index -1 drops the last element (mostl
y NA - a fuel adjustment)!
CACTdf # we now have clean plants
```

Regarding outliers, we can quickly see (visually, but also from a scatter plot) that the 12th plant is an outlier with an output of less than 1% of many other plants: in particular, the CA output component of this plant is anomalous and it appears to be operating effectively in so-called *simple cycle* mode. Accordingly, this outlier plant was dropped using the code excerpted above.

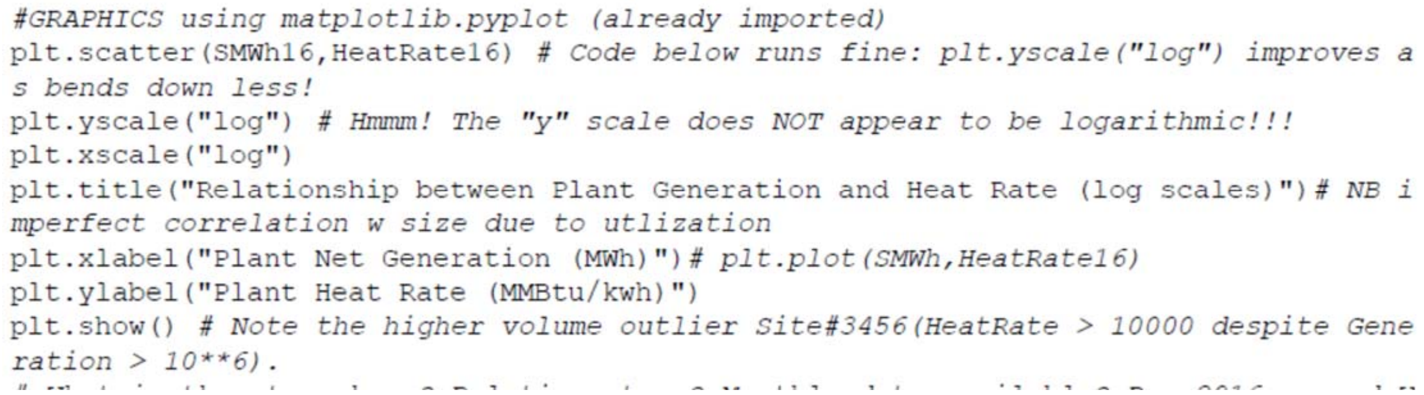
The next step was to derive an overall Heat Rate (see below – “HeatRate16”) that reflected the combined fuel consumption (in Btu) and the output power generation (in MWh) for the respective CA & CT units properly combined at the site level. In addition the proportion of fuel consumed at the CA (steam Turbine) level was computed and two outlier plants were identified and dropped using the code excerpted below.



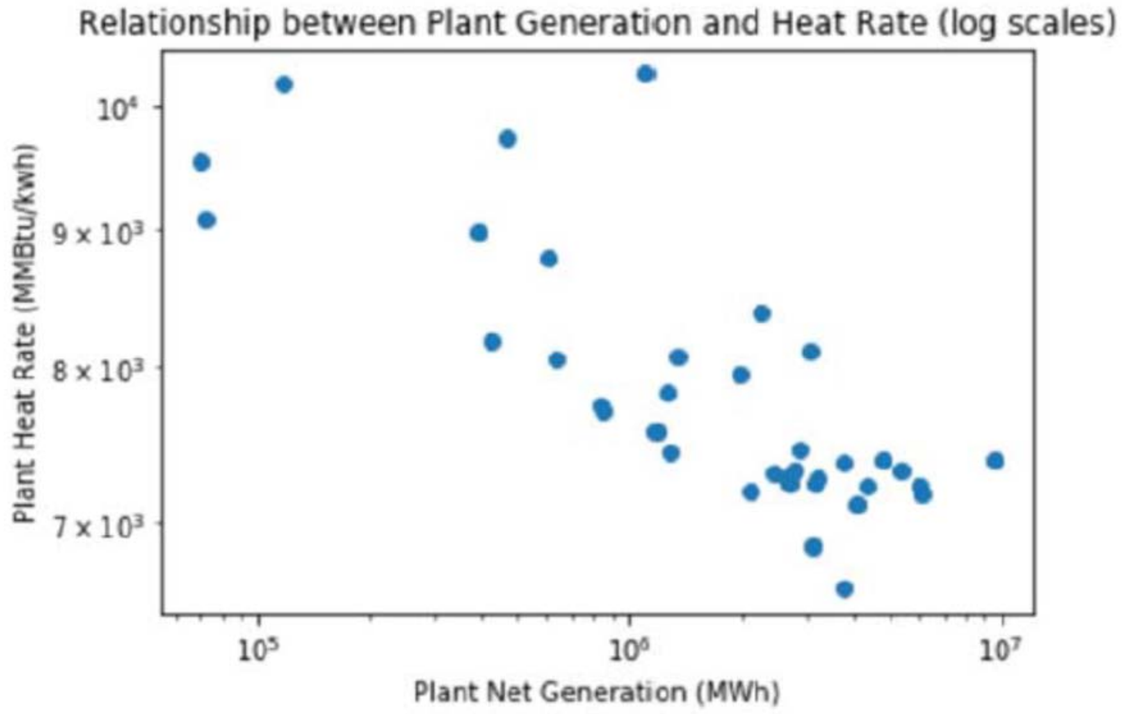
```
SMWh16 = (CACTdf.MWh_CT + CACTdf.MWh_CA)
SBtu16 = (CACTdf.Btu_CT + CACTdf.Btu_CA)
duct_prop = (100 * CACTdf.Btu_CA)/CACTdf.Btu_CT # derives the Btu for supplementary
firing relative to the CT as a %

CACTdf["duct_prop"] = duct_prop.astype(int)
HeatRate16 = (SBtu16/SMWh16)*1000
CACTdf = CACTdf[CACTdf.duct_prop < 50]
```

The so-derived consolidated Heat Rate can then be related to the power output (each using a logarithmic scale) and an inverse relationship may be discerned.



```
#GRAPHICS using matplotlib.pyplot (already imported)
plt.scatter(SMWh16,HeatRate16) # Code below runs fine: plt.yscale("log") improves a
s bends down less!
plt.yscale("log") # Hmmm! The "y" scale does NOT appear to be logarithmic!!!
plt.xscale("log")
plt.title("Relationship between Plant Generation and Heat Rate (log scales)")# NB i
mperfect correlation w size due to utlization
plt.xlabel("Plant Net Generation (MWh)")# plt.plot(SMWh,HeatRate16)
plt.ylabel("Plant Heat Rate (MMBtu/kwh)")
plt.show() # Note the higher volume outlier Site#3456(HeatRate > 10000 despite Gene
ration > 10**6).
```



Post Script: the foregoing commentary relates to just a fraction of the data wrangling undertaken in the context of this project, but nevertheless illustrates the general approach.