In [1]:

```
import pandas as pd
import datetime
import numpy as np
import ast
from ast import literal_eval
from pandas import DataFrame
from datetime import datetime
%matplotlib inline
```

In [2]:

```
# Read the input file from the previous step, code2:

df1=pd.read_excel('Pilot20190124_Part2C.xlsx')  #This is the output file from coc
```

In [ ]:

```

```

In [3]:

```
#Introduce a new dataframe which includes the electricity and diesel cost for eac
# Low:1, Medium:2 and High:3

df2=pd.read_excel('DEprices2.xlsx')
df2.round(decimals=3)
```

Out[3]:

	country	Dprice1	Dprice2	Dprice3	Eprice1	Eprice2	Eprice3
0	Algeria	0.20	0.250	0.300	0.028	0.035	0.042
1	Tunisia	0.63	0.788	0.945	0.100	0.125	0.150
2	Libya	0.11	0.138	0.165	0.168	0.210	0.252

- Youssef's addition
- Could not run it in any case as the "country" attribute was missing in the data provided

In [4]:

```
#Merging both dataframes on the country name since each country has different cos
data = pd.merge(df1, df2, on='country')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent cal
l last)
/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/index
es/base.py in get_loc(self, key, method, tolerance)
   2392             try:
-> 2393                 return self._engine.get_loc(key)
   2394             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc (pa
```

```
ndas/_libs/index.c:5239)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc (pa
ndas/_libs/index.c:5085)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Py
ObjectHashTable.get_item (pandas/_libs/hashtable.c:20405)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Py
ObjectHashTable.get_item (pandas/_libs/hashtable.c:20359)()

KeyError: 'country'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent cal
l last)
<ipython-input-4-2a74734c7f3e> in <module>()
      1 #Merging both dataframes on the country name since each coun
try has different cost for electricity and diesel:
----> 2 data = pd.merge(df1, df2, on='country')

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/resha
pe/merge.py in merge(left, right, how, on, left_on, right_on, left_i
ndex, right_index, sort, suffixes, copy, indicator)
     51                          right_on=right_on, left_index=left_
index,
     52                          right_index=right_index, sort=sort,
suffixes=suffixes,
---> 53                          copy=copy, indicator=indicator)
     54     return op.get_result()
     55

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/resha
pe/merge.py in __init__(self, left, right, how, on, left_on, right_o
n, axis, left_index, right_index, sort, suffixes, copy, indicator)
    556         (self.left_join_keys,
    557          self.right_join_keys,
--> 558          self.join_names) = self._get_merge_keys()
    559
    560         # validate the merge keys dtypes. We may need to coe
rce

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/resha
pe/merge.py in _get_merge_keys(self)
    821                             right_keys.append(rk)
    822                     if lk is not None:
--> 823                         left_keys.append(left[lk]._values)
    824                         join_names.append(lk)
    825                     else:

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/frame
.py in __getitem__(self, key)
   2060             return self._getitem_multilevel(key)
   2061         else:
-> 2062             return self._getitem_column(key)
   2063
```

```
   2064     def _getitem_column(self, key):

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/frame
.py in _getitem_column(self, key)
   2067         # get column
   2068         if self.columns.is_unique:
-> 2069             return self._get_item_cache(key)
   2070
   2071         # duplicate columns & possible reduce dimensionality

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/gener
ic.py in _get_item_cache(self, item)
   1532         res = cache.get(item)
   1533         if res is None:
-> 1534             values = self._data.get(item)
   1535             res = self._box_item_values(item, values)
   1536             cache[item] = res

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/inter
nals.py in get(self, item, fastpath)
   3588
   3589             if not isnull(item):
-> 3590                 loc = self.items.get_loc(item)
   3591             else:
   3592                 indexer = np.arange(len(self.items))[isnull(
self.items)]

/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/index
es/base.py in get_loc(self, key, method, tolerance)
   2393                 return self._engine.get_loc(key)
   2394             except KeyError:
-> 2395                 return self._engine.get_loc(self._maybe_cast
_indexer(key))
   2396
   2397         indexer = self.get_indexer([key], method=method,
tolerance=tolerance)

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc (pa
ndas/_libs/index.c:5239)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc (pa
ndas/_libs/index.c:5085)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Py
ObjectHashTable.get_item (pandas/_libs/hashtable.c:20405)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Py
ObjectHashTable.get_item (pandas/_libs/hashtable.c:20359)()

KeyError: 'country'
```

In [9]:

```
%%time

#Setting the default value for these parameters

for i in range (1,13):
    data['PD_D_{}'.format(i)]=0      #PD_D_: Peak Demand (kw) using diesel powere
    data['PD_E_{}'.format(i)]=0      #PD_E_: Peak Demand (kw) using electric powe
    data['ED_D_{}'.format(i)]=0      #ED_D_: Electricity Demand (kwh) using diese
    data['ED_E_{}'.format(i)]=0      #ED_E_: Electricity Demand (kwh) using elect
    data['Dcons_{}'.format(i)]=0     #Dcons_: Monthly amount of diesel needed to
    data['Dcost_{}'.format(i)]=0     #Dcost_: Diesel cost (USD) to run diesel pow
    data['Ecost_{}'.format(i)]=0     #Ecost_: Electricity cost (USD) using electr
```

```
CPU times: user 240 ms, sys: 79.7 ms, total: 319 ms
Wall time: 213 ms
```

In [ ]:

In [10]:

```
#tdh(m): total dynamic head
#gw_depth: depth to groundwater (m) + wdd:water drawdown(?) + oap: operating appl
#operating application pressure: 0 m (SU), 30 m (SP), 10 m (DR) , check if it is
#pressure loss in distributionL: 1 m (SU), 20% of oap (SP, DR) , check if it is i

wdd=0   #water drawdown
oap=0   #operating application pressure (m)
pld=0   #pressure loss in distribution

#This step of ground water level specification was removed since we have a layer

#data.rename(index=str, columns={'gw_depth': 'gw_m'})

def tdh_gw(row):
    tdh=(row['gw_depth']+wdd+oap+pld)
    return tdh


data['tdh_gw'] = data.apply(tdh_gw , axis=1)
```

- Not sure if it should be removed or not
- Conversion of groundwater class (GIS layer) to groundwater depth (in m) + additional parameter variables such us water drop down (wdd), operating application pressure (oap) and pressure loss in distribution (pld) = total dynamic head
- Could be used for sensitivity analysis - different scenarios/ assumptions

In [ ]:

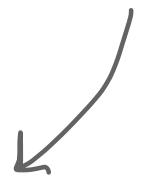
In [11]:

```
%time

#pumping plant efficiencty (%)= fuel efficiency (%) * "power unit eff (%)" * tran
#The Power Unit: can be diesel engine or electric engine. in the first we call it

#Value obtained from FAO1992, pages 26-27:

# Diesel powered pump
#Worst case: 0.9*0.3*0.9*0.4 ~ 10 % (0.1)
#Best case: 1*0.4*1*0.8 = 32% (0.32)

# Electric powered pump
#Worst case: 0.9*0.75*0.9*0.4 ~ 25% (0.25)
#Best case: 1*0.85*1*0.8 ~ 70% (0.7)

Dpump_plant_eff=0.1

for i in range (1,13):
    PD_D = 'PD_D_{}'.format(i)
    PWD = 'PWD_{}'.format(i)
    SSWD = 'SSWD_{}'.format(i)
    ED_D = 'ED_D_{}'.format(i)

    data[PD_D]=(9.81*(data[PWD]/1000)*data['tdh_gw'])/Dpump_plant_eff
    data[ED_D]=(data[SSWD]*data['tdh_gw']*0.00272)/Dpump_plant_eff
```

- Scenario analysis - technologies' efficiencies (assumptions)

- Electric vs diesel pumps, pumping plant efficiencies, power unit efficiencies etc (most of them retrieved from the FAO training manual)

- Electricity demand calculation!



```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.87 µs
```

In [12]:

```
Epump_plant_eff=0.25

for i in range (1,13):
    PD_E = 'PD_E_{}'.format(i)
    PWD = 'PWD_{}'.format(i)
    SSWD = 'SSWD_{}'.format(i)
    ED_E = 'ED_E_{}'.format(i)

    data[PD_E]=(9.81*(data[PWD]/1000)*data['tdh_gw'])/Epump_plant_eff
    data[ED_E]=(data[SSWD]*data['tdh_gw']*0.00272)/Epump_plant_eff
```

In [ ]:

In [13]:

```
%time

#fcons (l/kWh): fuel consumption for diesel motors
#fprice ($/l): diesel fuel price
#wcost ($/m3): Water cost (sourse: OSS2015_for better irrigation)

#Diesel motor:
#Dcons=0.09 l/kWh (page 48, FAO1992) and Dprice=$ 1.9 /l
#wcost=0.036 $/m3 NWSAS average cost  DZ:0.036, TN:0.04, LY:0.028

#Calculating the amount of diesel consumed and the total cost

Dreq= 0.09   #Dreq: Diesel Requirement is the amount of Diesel needed to generate


for i in range (1,13):
    ED_D = 'ED_D_{}'.format(i)
    Dcons = 'Dcons_{}'.format(i)
    data[Dcons]=data[ED_D]*Dreq
```

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 9.06 µs
```

In [14]:

```
%%time

# Low:1, Medium:2 and High:3

for i in range (1,13):
    Dcost = 'Dcost_{}'.format(i)
    Dcons = 'Dcons_{}'.format(i)
    data[Dcost] = data[Dcons]*data['Dprice1']
```

Costs scenarios (low, medium, high prices) developed by Youssef (!)

```
CPU times: user 46.6 ms, sys: 22.1 ms, total: 68.7 ms
Wall time: 64.1 ms
```

In [15]:

```
%%time

#Calculating monthly electricity cost:

Econs= 1
#Eprice= 0.11

for i in range (1,13):
    Ecost = 'Ecost_{}'.format(i)
    ED_E = 'ED_E_{}'.format(i)
    data[Ecost] = data[ED_E]*Econs*data['Eprice1']
```

```
CPU times: user 37.3 ms, sys: 7.4 ms, total: 44.7 ms
Wall time: 37.4 ms
```

In [ ]:

```
```

In [17]:

```
final=data.groupby('FID', as_index=False).sum() #changed from .max() instead of .
```

In [18]:

```
sswd = final.filter(like='SSWD_').sum()
powerdemandD = final.filter(like='PD_D_').sum()
powerdemandE = final.filter(like='PD_E_').sum()
energydemandD = final.filter(like='ED_D_').sum()
energydemandE = final.filter(like='ED_E_').sum()
dcost = final.filter(like='Dcost_').sum()
dcons = final.filter(like='Dcons_').sum()
ecost = final.filter(like='Ecost_').sum()
```

In [21]:

```
#Create a Pandas Excel writer using XlsxWriter as the engine.
writer = pd.ExcelWriter('Pilot20190124_Part3.xlsx', engine='xlsxwriter')
writer.book.use_zip64()

# Convert the dataframe to an XlsxWriter Excel object.
data.to_excel(writer, sheet_name='sur_ref')
final.to_excel(writer, sheet_name='results aggregated')
sswd.to_excel(writer, sheet_name='SSWD in m3')
powerdemandD.to_excel(writer, sheet_name='DP peak power demand in kw')
powerdemandE.to_excel(writer, sheet_name='EP peak power demand in kw')
energydemandD.to_excel(writer, sheet_name='DP electricity demand in kwh')
energydemandE.to_excel(writer, sheet_name='EP electricity demand in kwh')
ecost.to_excel(writer, sheet_name='electricity cost in USD')
dcons.to_excel(writer, sheet_name='diesel consumption in Liter')
dcost.to_excel(writer, sheet_name='diesel cost in USD')


# Close the Pandas Excel writer and output the Excel file.
writer.save()
```

In [ ]:

```
import multiprocessing
pool = multiprocessing.Pool(processes=2)
```

In [ ]:

```
```

In [1]:

In [2]:

In [ ]:

In [3]:

Out[3]:

	country	Dprice1	Dprice2	Dprice3	Eprice1	Eprice2	Eprice3
0	Algeria	0.20	0.250	0.300	0.028	0.035	0.042
1	Tunisia	0.63	0.788	0.945	0.100	0.125	0.150
2	Libya	0.11	0.138	0.165	0.168	0.210	0.252

In [4]:

```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent cal
l last)
/Users/kostas/anaconda/lib/python3.6/site-packages/pandas/core/index
es/base.py in get_loc(self, key, method, tolerance)
   2392             try:
-> 2393                 return self._engine.get_loc(key)
   2394             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc (pa
ndas/_libs/index.c:5239)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc (pa
ndas/_libs/index.c:5085)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Py
ObjectHashTable.get_item (pandas/_libs/hashtable.c:20405)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Py
```

In [9]:

```
CPU times: user 240 ms, sys: 79.7 ms, total: 319 ms
Wall time: 213 ms
```

In [ ]:

In [10]:

In [ ]:

In [11]:

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.87 µs
```

In [12]:

In [ ]:

In [13]:

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 9.06 µs
```

In [14]:

```
CPU times: user 46.6 ms, sys: 22.1 ms, total: 68.7 ms
Wall time: 64.1 ms
```

In [15]:

```
CPU times: user 37.3 ms, sys: 7.4 ms, total: 44.7 ms
Wall time: 37.4 ms
```

In [ ]:

In [17]:

In [18]:

In [21]:

In [ ]:

In [ ]: