Version 1.0 / 16.9.2019

Hydro power plant constraints forecast

1 Module #1: File import and cleaning

2 Import libraries

Entrée [62]:

```
import math
import pandas as pd
import numpy as np
import array as arr
from pandas import ExcelWriter
from pandas import ExcelFile
import re
import statistics
from functools import reduce
```

executed in 4ms, finished 15:20:54 2019-09-16

3 Read file

3.1 Read source file into data frame and display columns

Entrée [63]:

```
df = pd.read_excel('../../Documents/_Dissertation/Data/Offer_Data.xlsm', sheet_name='Data')

if df is None:
  from google.colab import files
  uploaded = files.upload()
  df = pd.read_excel('Offer_Data.xlsm', sheet_name='Data')
```

executed in 4.43s, finished 15:21:00 2019-09-16

3.1.1 Check first few lines of imported file

Entrée [64]:

```
df.head()
```

executed in 22ms, finished 15:21:00 2019-09-16

Out[64]:

	Variante 1	1	2	3	4	5	6	7	8	1.1	...	Availability plant 1 [MW]	Availability plant 2 [MW]	Availabili plant [MV
0	2014-04-01	73.8	66.0	42.9	0.0	0.0	0.0	0.0	NaN	4.0	...	7.8	37.8	25.8
1	2014-04-02	73.8	66.0	42.9	0.0	0.0	0.0	0.0	NaN	4.0	...	7.8	37.8	25.8
2	2014-04-03	73.8	66.0	42.9	0.0	0.0	0.0	0.0	NaN	4.0	...	7.8	37.8	7.5
3	2014-04-04	73.8	66.0	42.9	0.0	0.0	0.0	0.0	NaN	4.0	...	7.8	37.8	6.4
4	2014-04-05	72.6	66.0	42.9	0.0	0.0	0.0	0.0	NaN	4.0	...	7.8	37.8	25.8

5 rows × 94 columns

Entrée [65]:

```
df.tail()
```

executed in 24ms, finished 15:21:00 2019-09-16

Out[65]:

	Variante 1	1	2	3	4	5	6	7	8	1.1	...	Availability plant 1 [MW]	Availability plant 2 [MW]	Availability plant 3 [MW
1912	2019-06-26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	7.8	37.8	0.0
1913	2019-06-27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	7.8	37.8	0.0
1914	2019-06-28	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	7.8	37.8	0.0
1915	2019-06-29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	7.8	37.8	0.0
1916	2019-06-30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	7.8	37.8	0.0

5 rows × 94 columns

Entrée [66]:

```
# display info about our dataframe, i.e. features types, labels, number of values including
df.info()
```

executed in 17ms, finished 15:21:00 2019-09-16

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1917 entries, 0 to 1916
Data columns (total 94 columns):
Variante 1                  1917 non-null datetime64[ns]
1                           1912 non-null float64
2                           1912 non-null float64
3                           1912 non-null float64
4                           1912 non-null float64
5                           1912 non-null float64
6                           1912 non-null float64
7                           1912 non-null float64
8                           1364 non-null float64
1.1                         1912 non-null float64
2.1                         1912 non-null float64
3.1                         1912 non-null float64
4.1                         1912 non-null float64
5.1                         1912 non-null float64
6.1                         1912 non-null float64
7.1                         1912 non-null float64
8.1                         1369 non-null float64
Unnamed: 17                 0 non-null float64
Variante 2                  1917 non-null datetime64[ns]
1.2                         1912 non-null float64
2.2                         1912 non-null float64
3.2                         1912 non-null float64
4.2                         1912 non-null float64
5.2                         1912 non-null float64
6.2                         1912 non-null float64
7.2                         1912 non-null float64
8.2                         1416 non-null float64
1.3                         1912 non-null float64
2.3                         1912 non-null float64
3.3                         1912 non-null float64
4.3                         1912 non-null float64
5.3                         1912 non-null float64
6.3                         1912 non-null float64
7.3                         1912 non-null float64
8.3                         1369 non-null float64
Unnamed: 35                 0 non-null float64
Variante 3                  1917 non-null datetime64[ns]
1.4                         1911 non-null float64
2.4                         1911 non-null float64
3.4                         1911 non-null float64
4.4                         1911 non-null float64
5.4                         1911 non-null float64
6.4                         1911 non-null float64
7.4                         1911 non-null float64
8.4                         1363 non-null float64
1.5                         1911 non-null float64
2.5                         1911 non-null float64
3.5                         1911 non-null float64
4.5                         1911 non-null float64
5.5                         1911 non-null float64
6.5                         1911 non-null float64
```

```
7.5                          1911 non-null float64
8.5                          1369 non-null float64
Unnamed: 53                  0 non-null float64
Variante 4                   1917 non-null datetime64[ns]
1.6                          1912 non-null float64
2.6                          1912 non-null float64
3.6                          1912 non-null float64
4.6                          1912 non-null float64
5.6                          1912 non-null float64
6.6                          1912 non-null float64
7.6                          1912 non-null float64
8.6                          1390 non-null float64
1.7                          1912 non-null float64
2.7                          1912 non-null float64
3.7                          1912 non-null float64
4.7                          1912 non-null float64
5.7                          1912 non-null float64
6.7                          1912 non-null float64
7.7                          1912 non-null float64
8.7                          1369 non-null float64
Unnamed: 71                  0 non-null float64
Min prod                     1917 non-null datetime64[ns]
Unnamed: 73                  1913 non-null float64
Unnamed: 74                  0 non-null float64
Inflow lake 1 [m3]           1917 non-null float64
Inflow lake 2 [m3]           1917 non-null float64
Inflow lake 3 [m3]           1917 non-null float64
Inflow lake 4 [m3]           1917 non-null float64
Unnamed: 79                  0 non-null float64
Vol lake 1 [1000m3]          1917 non-null int64
Vol lake 1 [%]               1917 non-null float64
Max lake 1 [1000m3]          1917 non-null int64
Unnamed: 83                  0 non-null float64
Availability plant 1 [MW]    1917 non-null float64
Availability plant 2 [MW]    1917 non-null float64
Availability plant 3 [MW]    1917 non-null float64
Availability plant 4 [MW]    1917 non-null float64
Availability plant 1 [%]     1917 non-null float64
Availability plant 2 [%]     1917 non-null float64
Availability plant 3 [%]     1917 non-null float64
Availability plant 4 [%]     1917 non-null float64
Unnamed: 92                  0 non-null float64
SDL [MWh]                    1917 non-null float64
dtypes: datetime64[ns](5), float64(87), int64(2)
memory usage: 1.4 MB
```

Entrée [67]:

```
# Rename first column to "date"
df.rename(columns={ df.columns[0]: "Date"}, inplace=True)
# Force index to be date (as provided in the first column)
df.index = df['Date']
```

executed in 9ms, finished 15:21:00 2019-09-16

Entrée [68]:

```
# Heading for "Min prod" change to "Min prod date" and next column named "Min prod" (since
indexMP =  np.where(df.columns =="Min prod")[0][0]
df.rename(columns={ df.columns[indexMP]: "Min prod date"}, inplace=True)
df.rename(columns={ df.columns[indexMP+1]: "Min prod"}, inplace=True)
```

executed in 9ms, finished 15:21:00 2019-09-16

Entrée [69]:

```
# Extract columns names to be renamed, in order to make them more explicit
col_names = []
for col_name in df.columns:
    col_names.append(col_name)

# loop through all column names, to check which ones need to be made more explicit, i.e. in
# where 1 is the variant number and 2 identifies the 2nd value for Power ("P")
for i in range(len(col_names)):
    col_name = col_names[i]
    # if column name if numerical (f. ex 3.4), it is an automatic name assigned by the pana
    # we give it a more explicit name, in the for Var2H3 (variant number 2, Hour value, 3ra
    if re.match( r"^([0-9]?).([0-9]?)$", str(col_name), re.M):
        # check actual format: is there a decimal point?
        if re.match( r"^([0-9]?)$", str(col_name), re.M):
            # no decimal point -> we add one for consistency
            col_name = str(col_name)+".0"
        # name is in format: n.m, n= value index, m = variant nr + indication nb hour or po
        (n,m) = col_name.split(".")
        # if n is even, this is a power value, if odd, a number of hours
        # m is the variant number (option)
        New_Col_Name = "Var" + str((int(m)+2) // 2)
        if (int(m) % 2) == 0:
            New_Col_Name += "P"+str(n)
        else:
            New_Col_Name += "H"+str(n)
        #print(New_Col_Name)
        col_names[i] = New_Col_Name

df.columns = col_names

# debug
display(df.columns)
```

executed in 11ms, finished 15:21:00 2019-09-16

```
Index(['Date', 'Var1P1', 'Var1P2', 'Var1P3', 'Var1P4', 'Var1P5', 'Var1P6',
       'Var1P7', 'Var1P8', 'Var1H1', 'Var1H2', 'Var1H3', 'Var1H4', 'Var1H5',
       'Var1H6', 'Var1H7', 'Var1H8', 'Unnamed: 17', 'Variante 2', 'Var2P1',
       'Var2P2', 'Var2P3', 'Var2P4', 'Var2P5', 'Var2P6', 'Var2P7', 'Var2P8',
       'Var2H1', 'Var2H2', 'Var2H3', 'Var2H4', 'Var2H5', 'Var2H6', 'Var2H7',
       'Var2H8', 'Unnamed: 35', 'Variante 3', 'Var3P1', 'Var3P2', 'Var3P3',
       'Var3P4', 'Var3P5', 'Var3P6', 'Var3P7', 'Var3P8', 'Var3H1', 'Var3H2',
       'Var3H3', 'Var3H4', 'Var3H5', 'Var3H6', 'Var3H7', 'Var3H8',
       'Unnamed: 53', 'Variante 4', 'Var4P1', 'Var4P2', 'Var4P3', 'Var4P4',
       'Var4P5', 'Var4P6', 'Var4P7', 'Var4P8', 'Var4H1', 'Var4H2', 'Var4H3',
       'Var4H4', 'Var4H5', 'Var4H6', 'Var4H7', 'Var4H8', 'Unnamed: 71',
       'Min prod date', 'Min prod', 'Unnamed: 74', 'Inflow lake 1 [m3]',
       'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]',
       'Unnamed: 79', 'Vol lake 1 [1000m3]', 'Vol lake 1 [%]',
       'Max lake 1 [1000m3]', 'Unnamed: 83', 'Availability plant 1 [MW]',
       'Availability plant 2 [MW]', 'Availability plant 3 [MW]',
       'Availability plant 4 [MW]', 'Availability plant 1 [%]',
       'Availability plant 2 [%]', 'Availability plant 3 [%]',
       'Availability plant 4 [%]', 'Unnamed: 92', 'SDL [MWh]'],
      dtype='object')
```

4 Removal of empty columns

Entrée [70]:

```
# Remove columns that are unnamed (as they are empty)
df = df[df.columns.drop(list(df.filter(regex='Unnamed')))]
# display key numbers
print("Number of columns:", len(df.columns))
print("Column headings:")
display(df.columns)
```

executed in 12ms, finished 15:21:00 2019-09-16

```
Number of columns: 86
Column headings:

Index(['Date', 'Var1P1', 'Var1P2', 'Var1P3', 'Var1P4', 'Var1P5', 'Var1P6',
       'Var1P7', 'Var1P8', 'Var1H1', 'Var1H2', 'Var1H3', 'Var1H4', 'Var1H5',
       'Var1H6', 'Var1H7', 'Var1H8', 'Variante 2', 'Var2P1', 'Var2P2',
       'Var2P3', 'Var2P4', 'Var2P5', 'Var2P6', 'Var2P7', 'Var2P8', 'Var2H1',
       'Var2H2', 'Var2H3', 'Var2H4', 'Var2H5', 'Var2H6', 'Var2H7', 'Var2H8',
       'Variante 3', 'Var3P1', 'Var3P2', 'Var3P3', 'Var3P4', 'Var3P5',
       'Var3P6', 'Var3P7', 'Var3P8', 'Var3H1', 'Var3H2', 'Var3H3', 'Var3H4',
       'Var3H5', 'Var3H6', 'Var3H7', 'Var3H8', 'Variante 4', 'Var4P1',
       'Var4P2', 'Var4P3', 'Var4P4', 'Var4P5', 'Var4P6', 'Var4P7', 'Var4P8',
       'Var4H1', 'Var4H2', 'Var4H3', 'Var4H4', 'Var4H5', 'Var4H6', 'Var4H7',
       'Var4H8', 'Min prod date', 'Min prod', 'Inflow lake 1 [m3]',
       'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]',
       'Vol lake 1 [1000m3]', 'Vol lake 1 [%]', 'Max lake 1 [1000m3]',
       'Availability plant 1 [MW]', 'Availability plant 2 [MW]',
       'Availability plant 3 [MW]', 'Availability plant 4 [MW]',
       'Availability plant 1 [%]', 'Availability plant 2 [%]',
       'Availability plant 3 [%]', 'Availability plant 4 [%]', 'SDL [MWh]'],
      dtype='object')
```

5 Missing values management

Entrée [71]:

```
# fill up missing 8th value with 0s, for all variants (power nd number of hours)
for iVar in range(1,4+1):
    HourIndex = "Var"+str(iVar)+"H8"
    PwrIndex = "Var"+str(iVar)+"P8"
    # for column 8, replace missing values with 0
    df[HourIndex].fillna(0, inplace=True)
    df[PwrIndex].fillna(0, inplace=True)
```

executed in 231ms, finished 15:21:01 2019-09-16

Entrée [72]:

```
# Check missing values
null_columns=df.columns[df.isnull().any()]
df[df.isnull().any(axis=1)][null_columns].head(50)
```

executed in 29ms, finished 15:21:01 2019-09-16

Out[72]:

Date	Var1P1	Var1P2	Var1P3	Var1P4	Var1P5	Var1P6	Var1P7	Var1H1	Var1H2	Var1H3	...
2015-10-08	69.0	66.0	0.0	0.0	0.0	0.0	0.0	8.0	4.0	0.0	...
2015-12-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2016-11-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2016-11-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2016-11-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2016-12-26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

6 rows × 57 columns

Entrée [73]:

```
#Fill up missing values by interpolation
df.interpolate(method='linear',limit_direction ='forward',inplace=True)
```

executed in 89ms, finished 15:21:02 2019-09-16

6 Correct anomalies

Entrée [74]:

```
# check when max sevel constraint to zero but lake level remains high
df[(df["Max lake 1 [1000m3]"] < 1500) & (df["Vol lake 1 [%]"] > 0.2)][["Max lake 1 [1000m3]
```

executed in 10ms, finished 15:21:03 2019-09-16

Out[74]:

Date	Max lake 1 [1000m3]	Vol lake 1 [%]
2014-06-25	0	0.41065
2014-06-26	0	0.42008

Entrée [75]:

```
# These are clearly anomalies. We fix them using linear interpolation as above, first repla
df.loc[(df["Max lake 1 [1000m3]"] < 1500) & (df["Vol lake 1 [%]"] > 0.2), ["Max lake 1 [100
#Fill up missing values by interpolation
df.interpolate(method='linear',limit_direction ='forward',inplace=True)
```

executed in 43ms, finished 15:21:03 2019-09-16

7 Feature engineering

7.1 Weekend feature

Entrée [76]:

```
# Add feature indicating if the date is a weekend day or a weekday
Weekend = df['Date'].map(lambda x: x.weekday()==5 or x.weekday()==6 )
# Add column that indicates if day is weekend (True in this case)
df.insert(len(df.columns),"Weekend", Weekend)
```

executed in 10ms, finished 15:21:04 2019-09-16

7.2 Keep only priority variant

Entrée [77]:

```
# Add calculated columns for gloabl values, i.e. total energy per variant, average energy p
# compute total energy for variante 1 to 4, as scalar product: power * nb of hours

#Loop over 4 variants
for iVar in range(1,4+1):
    PowerVar = pd.Series(np.zeros(len(df))) # initialize to 0
    PowerVar.index = df.index
    VarCount = 0 # count the number of variants defined on a given day
    # loop over the 8 pairs : power, nb of hours
    for i in range(1, 8+1):
        HourIndex = "Var"+str(iVar)+"H"+str(i)
        PwrIndex = "Var"+str(iVar)+"P"+str(i)
        PowerVar += df[HourIndex]*df[PwrIndex]
    # Add column to dataframe
    New_Col_Name = "EnergyVar"+str(iVar)
    df.insert(len(df.columns),New_Col_Name, PowerVar)
```

executed in 20ms, finished 15:21:05 2019-09-16

Entrée [78]:

```
# Compute number of defined variants
NbVar = pd.Series(np.zeros(len(df)), index = df.index) # initialize to 0
NbVar = df["EnergyVar1"].map(lambda x: 1 if x > 0 else 0 ) + \
        df["EnergyVar2"].map(lambda x: 1 if x > 0 else 0 ) + \
        df["EnergyVar3"].map(lambda x: 1 if x > 0 else 0 ) + \
        df["EnergyVar4"].map(lambda x: 1 if x > 0 else 0 )

# Add column that counts number of defined variants
df.insert(len(df.columns),"NbVar", NbVar)
```

executed in 10ms, finished 15:21:06 2019-09-16

Entrée [79]:

```
# debug - fix
#df.drop("NbVar", axis=1, inplace=True)
#df.drop("EnergyVar1", axis=1, inplace=True)
#df.drop("EnergyVar2", axis=1, inplace=True)
#df.drop("EnergyVar3", axis=1, inplace=True)
#df.drop("EnergyVar4", axis=1, inplace=True)
df.head()
```

executed in 23ms, finished 15:21:06 2019-09-16

Out[79]:

	Date	Var1P1	Var1P2	Var1P3	Var1P4	Var1P5	Var1P6	Var1P7	Var1P8	Var1H1	...	A P
Date												
2014-04-01	2014-04-01	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-02	2014-04-02	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-03	2014-04-03	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-04	2014-04-04	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-05	2014-04-05	72.6	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	

5 rows × 92 columns

Entrée [80]:

```
# Calculating the single variante according to defined variants by priority order (1,2,3,4)
VariantePrioOrder = df.apply(
    lambda row: row['EnergyVar1'] if row['EnergyVar1']>0 else \
      ( row['EnergyVar2'] if row['EnergyVar2']>0 else \
       ( row['EnergyVar3'] if row['EnergyVar3']>0 else \
        ( row['EnergyVar4'])))
      ,
    axis=1
)
# Add column that counts number of defined variants
df.insert(len(df.columns),"Variante Prio", VariantePrioOrder)
```

executed in 71ms, finished 15:21:07 2019-09-16

Entrée [81]:

```
#df.drop(labels=["PrioH1","PrioP1","PrioH2","PrioP2","PrioH3","PrioP3","PrioH4","PrioP4"],
```

executed in 3ms, finished 15:21:07 2019-09-16

Entrée [82]:

```
# the same way add power and number of hours columns corresponding to the priority variant
NewColPow= pd.Series(np.zeros(len(df))) # initialize to 0
NewColPow.index = df.index
NewColNbH= pd.Series(np.zeros(len(df))) # initialize to 0
NewColNbH.index = df.index

for i in range(1, 8+1):
  for valueType in ["H","P"]:
    VariantePrio = df.apply(
      lambda row: row['Var1'+valueType+str(i)] if row['EnergyVar1']>0 else \
        ( row['Var2'+valueType+str(i)] if row['EnergyVar2']>0 else \
         ( row['Var3'+valueType+str(i)] if row['EnergyVar3']>0 else \
          ( row['Var4'+valueType+str(i)])))
           ,
        axis=1
      )
    # Add column to dataframe
    New_Col_Name = "Prio"+valueType+str(i)
    df.insert(len(df.columns),New_Col_Name, VariantePrio)

df.head()
```

executed in 1.12s, finished 15:21:09 2019-09-16

Out[82]:

	Date	Var1P1	Var1P2	Var1P3	Var1P4	Var1P5	Var1P6	Var1P7	Var1P8	Var1H1	...	P
Date												
2014-04-01	2014-04-01	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-02	2014-04-02	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-03	2014-04-03	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-04	2014-04-04	73.8	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	
2014-04-05	2014-04-05	72.6	66.0	42.9	0.0	0.0	0.0	0.0	0.0	4.0	...	

5 rows × 109 columns

Entrée [83]:

```
# remove Variants as they are not relevant and not needed anymore
for iVar in range(1,4+1):
    df.drop('EnergyVar'+str(iVar), axis=1, inplace=True)
    for i in range(1, 8+1):
      for valueType in ["H","P"]:
        df.drop('Var'+str(iVar)+valueType+str(i), axis=1, inplace=True)

df.drop('NbVar', axis=1, inplace=True)
df.head()
```

executed in 133ms, finished 15:21:09 2019-09-16

Out[83]:

Date	Date	Variante 2	Variante 3	Variante 4	Min prod date	Min prod	Inflow lake 1 [m3]	Inflow lake 2 [m3]	Inflow lake 3 [m3]	Inflow lake 4 [m3]	...	PrioH
2014-04-01	2014-04-01	2014-04-01	2014-04-01	2014-04-01	2014-04-01	0.0	31.0	4.0	129.0	107.0	...	0.
2014-04-02	2014-04-02	2014-04-02	2014-04-02	2014-04-02	2014-04-02	150.0	0.0	-14.0	148.0	116.0	...	0.
2014-04-03	2014-04-03	2014-04-03	2014-04-03	2014-04-03	2014-04-03	150.0	10.0	6.0	132.0	118.0	...	0.
2014-04-04	2014-04-04	2014-04-04	2014-04-04	2014-04-04	2014-04-04	150.0	19.0	6.0	150.0	118.0	...	0.
2014-04-05	2014-04-05	2014-04-05	2014-04-05	2014-04-05	2014-04-05	180.0	41.0	15.0	148.0	124.0	...	0.

5 rows × 40 columns

7.3 Sort power value pairs

Entrée [84]:

```
# sort the pairs by decreasing power values
# create list 8 tuples for each line in dataset, 1 by pair (power, nb of hours)
PairsPowerNbH = df.apply(
      lambda row: sorted([(row['PrioP'+str(i)], row['PrioH'+str(i)]) for i in range(1,8+1)]
          ,
        axis=1
      )
PairsPowerNbH.index = df.index
# This creates a pandas series! containing list of pairs (ordered by power values decreasin

# assign back to dataframe (8x2 original columns)
ListSortedPairsPowerNbH = PairsPowerNbH.apply(
    lambda row: [row[i][1] for i in range(0,8)]+[row[i][0] for i in range(0,8)] )

ListSortedPairsPowerNbH.values.tolist()
cols = ["PrioH"+str(i) for i in range(1,8+1)]+["PrioP"+str(i) for i in range(1,8+1)]
dfSortedPairsPowerNbH = pd.DataFrame( data = ListSortedPairsPowerNbH.values.tolist(),
                                      columns = cols, index = ListSortedPairsPowerNbH.index)

# We update the original elements with the sorted ones
df.loc[:,"PrioH1":"PrioP8"] =dfSortedPairsPowerNbH
```

executed in 307ms, finished 15:21:09 2019-09-16

7.4 Compress 8 power pairs to 4

Entrée [85]:

```
# compress information of 5 last pairs to only 1 column
# retain total energy and number of hours, compute resulting power

colEnergy5to8 = pd.Series(np.zeros(len(df))) # initialize to 0
colEnergy5to8.index = df.index
ColSumHours5to8 = pd.Series(np.zeros(len(df))) # initialize to 0
ColSumHours5to8.index = df.index

# loop over the 5 last pairs : power, nb of hours
for i in range(4, 8+1):
    HourIndex = "PrioH"+str(i)
    PwrIndex = "PrioP"+str(i)
    colEnergy5to8 += df[HourIndex]*df[PwrIndex]
    ColSumHours5to8 += df[HourIndex]

ColPower5to8 =  colEnergy5to8 / ColSumHours5to8
ColPower5to8.fillna(0, inplace=True)

ColPower5to8[ColPower5to8>0].head(5)
```

executed in 14ms, finished 15:21:10 2019-09-16

Out[85]:

```
Date
2014-04-10    37.8
2014-04-11    37.8
2014-04-12    37.8
2014-04-13    37.8
2014-04-14    37.8
dtype: float64
```

Entrée [86]:

```
df["PrioH4"] = ColSumHours5to8
df["PrioP4"] = ColPower5to8

# remove columns 5-8 now not needed anymore
for i in range(5, 8+1):
  for valueType in ["H","P"]:
    df.drop('Prio'+valueType+str(i), axis=1, inplace=True)
df.head()
```

executed in 33ms, finished 15:21:10 2019-09-16

Out[86]:

Date	Date	Variante 2	Variante 3	Variante 4	Min prod date	Min prod	Inflow lake 1 [m3]	Inflow lake 2 [m3]	Inflow lake 3 [m3]	Inflow lake 4 [m3]	...	Week
2014-04-01	2014-04-01	2014-04-01	2014-04-01	2014-04-01	2014-04-01	0.0	31.0	4.0	129.0	107.0	...	F
2014-04-02	2014-04-02	2014-04-02	2014-04-02	2014-04-02	2014-04-02	150.0	0.0	-14.0	148.0	116.0	...	F
2014-04-03	2014-04-03	2014-04-03	2014-04-03	2014-04-03	2014-04-03	150.0	10.0	6.0	132.0	118.0	...	F
2014-04-04	2014-04-04	2014-04-04	2014-04-04	2014-04-04	2014-04-04	150.0	19.0	6.0	150.0	118.0	...	F
2014-04-05	2014-04-05	2014-04-05	2014-04-05	2014-04-05	2014-04-05	180.0	41.0	15.0	148.0	124.0	...	T

5 rows × 32 columns

Entrée [87]:

```
# display basic stasticss of data frame
display(round(df.describe(),2))
```

executed in 85ms, finished 15:21:11 2019-09-16

	Min prod	Inflow lake 1 [m3]	Inflow lake 2 [m3]	Inflow lake 3 [m3]	Inflow lake 4 [m3]	Vol lake 1 [1000m3]	Vol lake 1 [%]	Max lake 1 [1000m3]	Availability plant 1 [MW]	A
count	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00	
mean	228.44	279.38	55.39	172.24	86.18	43694.71	0.44	26305.37	5.56	
std	194.11	393.60	89.28	155.39	100.92	30532.48	0.31	9228.22	3.24	
min	0.00	-210.00	-482.00	-208.00	-224.00	0.00	0.00	0.00	-0.00	
25%	90.00	42.00	8.70	69.00	37.30	14554.00	0.15	30000.00	3.90	
50%	180.00	119.00	31.00	119.40	59.30	42999.00	0.43	30000.00	7.80	
75%	300.00	376.00	75.00	228.00	116.00	70430.00	0.70	30000.00	7.80	
max	1470.00	3951.60	471.00	985.60	1349.70	97257.00	0.97	30000.00	7.80	

8 rows × 26 columns

Entrée [88]:

```
# check specific date for expected result
df.loc['2015-10-21',"PrioH1":"PrioP4"]
```

executed in 7ms, finished 15:21:11 2019-09-16

Out[88]:

```
PrioH1       8
PrioP1      69
PrioH2       4
PrioP2      66
PrioH3      12
PrioP3    58.5
PrioH4       0
PrioP4       0
Name: 2015-10-21 00:00:00, dtype: object
```

7.5 Total available energy as absolute value

Entrée [89]:

```
df['TotalAvailablePower'] = 0

for i in range (1,4+1):
    df['TotalAvailablePower'] =  df['TotalAvailablePower'] + df['Availability plant '+str(i
```

executed in 5ms, finished 15:21:12 2019-09-16

8 Removal of redundant columns

Entrée [90]:

```
df.drop(labels=['Availability plant 1 [MW]', 'Availability plant 2 [MW]',  \
                'Availability plant 3 [MW]','Availability plant 4 [MW]',   \
                'Min prod date', 'Variante 2','Variante 3','Variante 4',   \
                'Vol lake 1 [1000m3]'],  \
        axis = 1, inplace = True)
```

executed in 7ms, finished 15:21:13 2019-09-16

Entrée [91]:

```
df.head()
```

executed in 23ms, finished 15:21:14 2019-09-16

Out[91]:

	Date	Min prod	Inflow lake 1 [m3]	Inflow lake 2 [m3]	Inflow lake 3 [m3]	Inflow lake 4 [m3]	Vol lake 1 [%]	Max lake 1 [1000m3]	Availability plant 1 [%]	Availability plant 2 [%]	.
Date											
2014-04-01	2014-04-01	0.0	31.0	4.0	129.0	107.0	0.16467	30000.0	1.0	1.0	.
2014-04-02	2014-04-02	150.0	0.0	-14.0	148.0	116.0	0.15557	30000.0	1.0	1.0	.
2014-04-03	2014-04-03	150.0	10.0	6.0	132.0	118.0	0.14765	30000.0	1.0	1.0	.
2014-04-04	2014-04-04	150.0	19.0	6.0	150.0	118.0	0.13716	30000.0	1.0	1.0	.
2014-04-05	2014-04-05	180.0	41.0	15.0	148.0	124.0	0.13091	30000.0	1.0	1.0	.

5 rows × 24 columns

9 Export to CSV

Entrée [92]:

```
# Export resulting dataset for further treatments
export_csv = df.to_csv (r'clean_dataframe.csv', header=True)
```

executed in 79ms, finished 15:21:14 2019-09-16

Entrée [93]:

```
df.head(100)
```

executed in 64ms, finished 15:21:15 2019-09-16

Out[93]:

Date	Date	Min prod	Inflow lake 1 [m3]	Inflow lake 2 [m3]	Inflow lake 3 [m3]	Inflow lake 4 [m3]	Vol lake 1 [%]	Max lake 1 [1000m3]	Availability plant 1 [%]	Availability plant 2 [%]	...	Variante Prio
2014-04-01	2014-04-01	0.0	31.0	4.0	129.0	107.0	0.164670	30000.0	1.000000	1.000000	...	902.4
2014-04-02	2014-04-02	150.0	0.0	-14.0	148.0	116.0	0.155570	30000.0	1.000000	1.000000	...	902.4
2014-04-03	2014-04-03	150.0	10.0	6.0	132.0	118.0	0.147650	30000.0	1.000000	1.000000	...	902.4
2014-04-04	2014-04-04	150.0	19.0	6.0	150.0	118.0	0.137160	30000.0	1.000000	1.000000	...	902.4
2014-04-05	2014-04-05	180.0	41.0	15.0	148.0	124.0	0.130910	30000.0	1.000000	1.000000	...	897.6

10 Baseline file

The baseline file contains a first approach prediction for both minimum production and maximum production in the form of power / nb of hours pairs. This prediction is based on a simple set of algorithmic rules.

Entrée [94]:

```
# read original excel file
df_benchmark = pd.read_excel('./Data/proxy_unav.xlsx', sheet_name='proxy_unav',decimal=",",
df_benchmark.rename(columns={ df_benchmark.columns[0]: "Date"}, inplace=True)
df_benchmark['Date'] = pd.to_datetime(df_benchmark['Date'], format='%d.%m.%Y')
df_benchmark['Min Prod [MWh]'].astype(float)
# Force index to be date (as provided in the first column)
df_benchmark.index = df_benchmark['Date']
df_benchmark.info()
```

executed in 221ms, finished 15:21:17 2019-09-16

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1918 entries, 2014-04-01 to 2019-07-01
Data columns (total 9 columns):
Date              1918 non-null datetime64[ns]
P1 [MW]           1918 non-null float64
P2 [MW]           1918 non-null float64
P3 [MW]           1918 non-null float64
P4 [MW]           1918 non-null int64
H1 [#]            1918 non-null int64
H2 [#]            1918 non-null int64
H3 [#]            1918 non-null int64
Min Prod [MWh]    1918 non-null float64
dtypes: datetime64[ns](1), float64(4), int64(4)
memory usage: 149.8 KB
```

Entrée [95]:

```
# calculate total energy for benchmark values
#Loop over 3 pairs
PowerVarMax = pd.Series(np.zeros(len(df_benchmark))) # initialize to 0
PowerVarMax.index = df_benchmark.index
for i in range(1, 3+1):
    HourIndex = "H"+str(i)+" [#]"
    PwrIndex  = "P"+str(i)+" [MW]"
    PowerVarMax = PowerVarMax + df_benchmark[HourIndex]*df_benchmark[PwrIndex]
# Add column to dataframe
New_Col_Name = "MaxEnergy"
df_benchmark.insert(len(df_benchmark.columns),New_Col_Name, PowerVarMax)

df_benchmark.head()
```

executed in 15ms, finished 15:21:17 2019-09-16

Out[95]:

Date	Date	P1 [MW]	P2 [MW]	P3 [MW]	P4 [MW]	H1 [#]	H2 [#]	H3 [#]	Min Prod [MWh]	MaxEnergy
2014-04-01	2014-04-01	73.7	47.8	40.0	0	3	4	9	254.2	772.3
2014-04-02	2014-04-02	73.6	47.8	40.0	0	3	4	9	276.5	772.0
2014-04-03	2014-04-03	55.3	47.7	39.9	0	3	4	9	269.5	715.8
2014-04-04	2014-04-04	54.1	47.7	39.9	0	3	4	9	289.9	712.2
2014-04-05	2014-04-05	73.5	47.7	39.9	0	3	5	8	297.6	778.2

Entrée [96]:

```
# Export resulting dataset for further treatments
export_csv = df_benchmark.to_csv (r'baseline_dataframe.csv', header=True)
```

executed in 52ms, finished 15:21:19 2019-09-16

Entrée [ ]:

Entrée [ ]: