Hydro powerplant constraints forecast - power blocks by clusering

In this experiment, we attempy at forecasting power blocks by using clustering. Hence regression will be achieved by doing solving a classification probem.

1 Initial preparation

1.1 Import libraries

Entrée [117]:

```
import math
import pandas as pd
import numpy as np
import array as arr
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statistics
from functools import reduce
from pyclustering.cluster.silhouette import silhouette
from pyclustering.cluster.kmedoids import kmedoids
from pyclustering.utils import metric, distance_metric, type_metric
```

executed in 6ms, finished 17:40:28 2019-09-19

1.1.1 General helper functions

Entrée [118]:

```
def computeEnergyPairs(dfInput):
    # Compute prediction energy by pair, i.e. nb of hours times power (from 1 to 8 pairs)
    for i in range(1, 4+1):
        HourIndex = "PrioH"+str(i)
        PwrIndex  = "PrioP"+str(i)
        PowerVar  = dfInput[HourIndex]*dfInput[PwrIndex]
        # Add column to dataframe
        New_Col_Name = "EnergyPair"+str(i)
        dfInput[New_Col_Name] = PowerVar
```

executed in 4ms, finished 17:40:29 2019-09-19

Entrée [119]:

```
# helper function to make 24 hour vector out of blocks list
# returns a dataframe with 24 hours vectors as lines in a dataframe , from the blocks recei
def get24hoursEnergyVector(prefixHour, prefixPower, dfInput, prefixOutputCols = ""):
    # create result dataframe
    resCols = []
    dfResult = pd.DataFrame(index=dfInput.index, columns = [prefixOutputCols+str(i) for i i

    for myIndex in dfInput.index:
        iVectorIndex = 0
        resVect = np.zeros(24)
        for iPair in range(1,4+1):
            NbHours = int(dfInput.loc[myIndex,prefixHour+str(iPair)])
            Power = dfInput.loc[myIndex,prefixPower+str(iPair)]
            resVect[iVectorIndex:iVectorIndex+NbHours] = Power
            iVectorIndex = iVectorIndex + NbHours
        dfResult.loc[myIndex] = resVect
    return dfResult
```

executed in 8ms, finished 17:40:29 2019-09-19

Entrée [120]:

```
# plot stacked bars corresponding to energy blocks with different colors
# saves result in SVG format
def plotEnergyBlocks(dfInput, blockLabel, filename):
    computeEnergyPairs(dfInput)
    fig=plt.figure(figsize=(14, 6), dpi= 80, edgecolor='k')
    plt.bar(dfInput.index, dfInput.EnergyPair1, color = 'b', label=blockLabel+" 1")
    plt.bar(dfInput.index, dfInput.EnergyPair2, color = 'r', bottom = dfInput.EnergyPair1,
    plt.bar(dfInput.index, dfInput.EnergyPair3, color = 'y', bottom = dfInput.EnergyPair1+d
    plt.bar(dfInput.index, dfInput.EnergyPair4, color = 'c', bottom = dfInput.EnergyPair1+d
    plt.legend(loc='upper right')
    plt.show()
    fig.savefig(filename+'.svg', format='svg')
```

executed in 8ms, finished 17:40:30 2019-09-19

1.1.2 Read source file into data frame and display columns

Entrée [121]:

```
dateparse = lambda x: pd.datetime.strptime(x, '%Y-%m-%d')

df = pd.read_csv("clean_dataframe.csv", parse_dates=['Date'], date_parser=dateparse, index_
# rename date column
df.rename(columns={ df.columns[0]: "Date"}, inplace=True)
df.index = df["Date"]

df.rename(columns={ "Variante Prio": "Max prod"}, inplace=True)
```

executed in 52ms, finished 17:40:31 2019-09-19

1.1.3 Check first few lines of imported file

Entrée [122]:

```
df.head()
```

executed in 25ms, finished 17:40:32 2019-09-19

Out[122]:

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 [%]	.
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

Entrée [123]:

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

executed in 12ms, finished 17:40:33 2019-09-19

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1917 entries, 2014-04-01 to 2019-06-30
Data columns (total 24 columns):
Date                         1917 non-null datetime64[ns]
Min prod                     1917 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
Vol lake 1 [%]               1917 non-null float64
Max lake 1 [1000m3]          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
SDL [MWh]                    1917 non-null float64
Weekend                      1917 non-null bool
Max prod                     1917 non-null float64
PrioH1                       1917 non-null float64
PrioP1                       1917 non-null float64
PrioH2                       1917 non-null float64
PrioP2                       1917 non-null float64
PrioH3                       1917 non-null float64
PrioP3                       1917 non-null float64
PrioH4                       1917 non-null float64
PrioP4                       1917 non-null float64
TotalAvailablePower          1917 non-null float64
dtypes: bool(1), datetime64[ns](1), float64(22)
memory usage: 361.3 KB
```

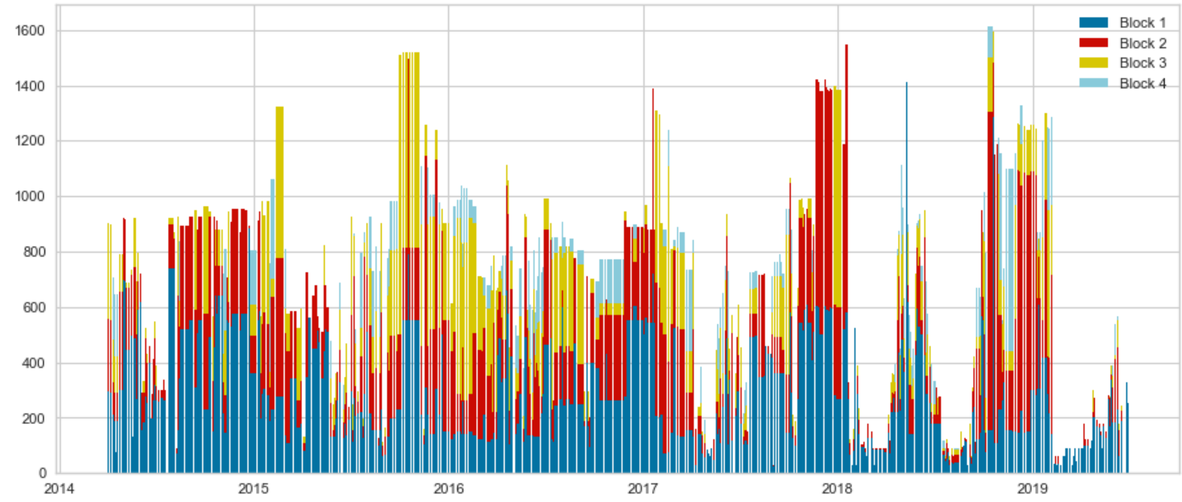
2 Targets exploratory analysis

2.1 Plotting

Entrée [124]:

```
# plot the energy blocks by color (full dataset)
plotEnergyBlocks(df, "Block", "complete_dataframe")
```

executed in 15.1s, finished 17:40:51 2019-09-19



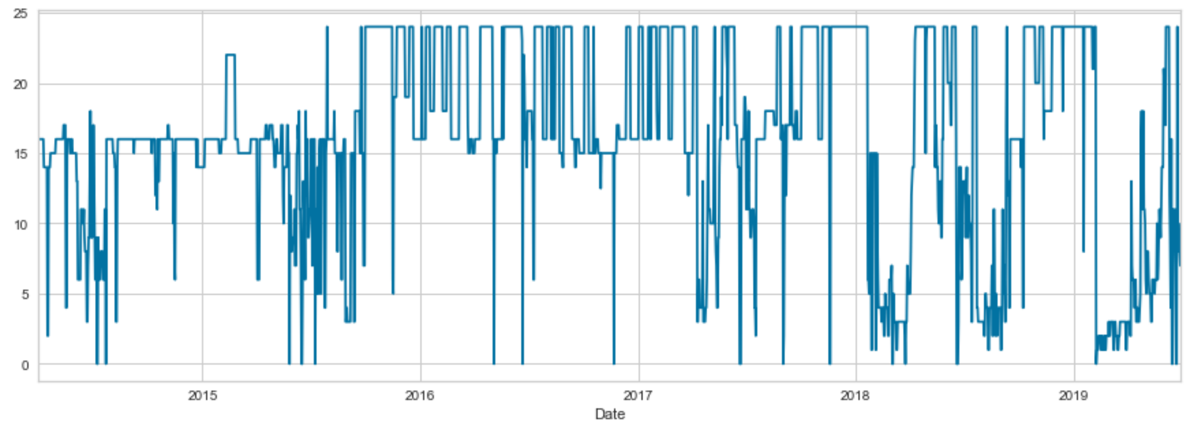
Entrée [125]:

```
# compute total number of allowed hours (sum of hour component in pairs: Nb Hours 1 + Nb Ho
SumHours = df.apply(
      lambda row: reduce((lambda x,y: x+y),([row['PrioH'+str(i)] for i in range(1,4+1)] ))
        ,
        axis=1
      )
# Add this as new feature
df["TotalNbHours"] = SumHours

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size
SumHours.plot()
```

executed in 529ms, finished 17:40:51 2019-09-19

Out[125]:

<matplotlib.axes._subplots.AxesSubplot at 0x1fb5de20048>



Entrée [126]:

```
#quality check: identify dates when total nb hours = 0 (is this possible?)
display(SumHours[SumHours==0])
```

executed in 9ms, finished 17:40:51 2019-09-19

```
Date
2014-07-08    0.0
2014-07-09    0.0
2014-07-23    0.0
2015-05-27    0.0
2015-06-16    0.0
2015-06-17    0.0
2015-07-09    0.0
2016-05-04    0.0
2016-06-21    0.0
2016-11-21    0.0
2017-06-20    0.0
2017-06-21    0.0
2017-06-22    0.0
2017-09-01    0.0
2017-11-18    0.0
2017-11-19    0.0
2018-03-03    0.0
2018-03-04    0.0
2018-03-25    0.0
2018-06-19    0.0
2018-06-20    0.0
2018-06-21    0.0
2019-02-08    0.0
2019-06-15    0.0
2019-06-16    0.0
2019-06-22    0.0
2019-06-23    0.0
dtype: float64
```

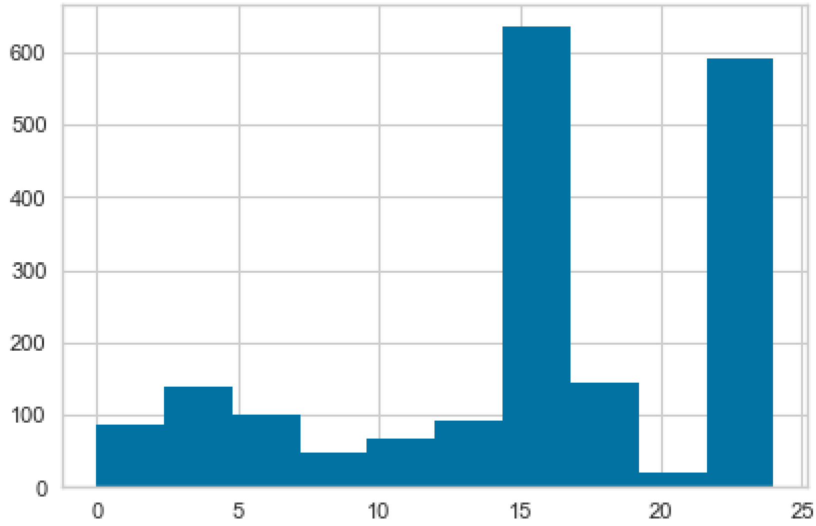
Entrée [127]:

```
# check distribution of total number of hours
fig=plt.figure(figsize=(6, 4), dpi= 80, facecolor='w', edgecolor='k')
SumHours.hist()
```

executed in 168ms, finished 17:40:51 2019-09-19

Out[127]:

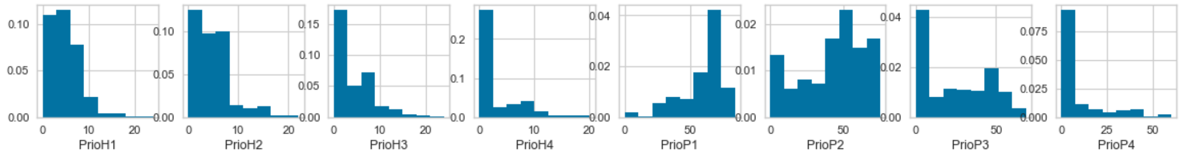
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fb119a8be0>
```



Entrée [128]:

```
# plot distribution of all pairs values: nb of hours and power

indexPlot = 1
fig=plt.figure(figsize=(18, 4), dpi= 80, facecolor='w', edgecolor='k')
for valType in ["PrioH","PrioP"]:
    for nbHours in range(1,4+1):
        plt.subplot(2,8,indexPlot) # equivalent to: plt.subplot(2, 2, 1)
        plt.hist(df[valType+str(nbHours)], density=True,bins=8)
        plt.xlabel(valType+str(nbHours));
        indexPlot = indexPlot+1
```

executed in 908ms, finished 17:40:52 2019-09-19



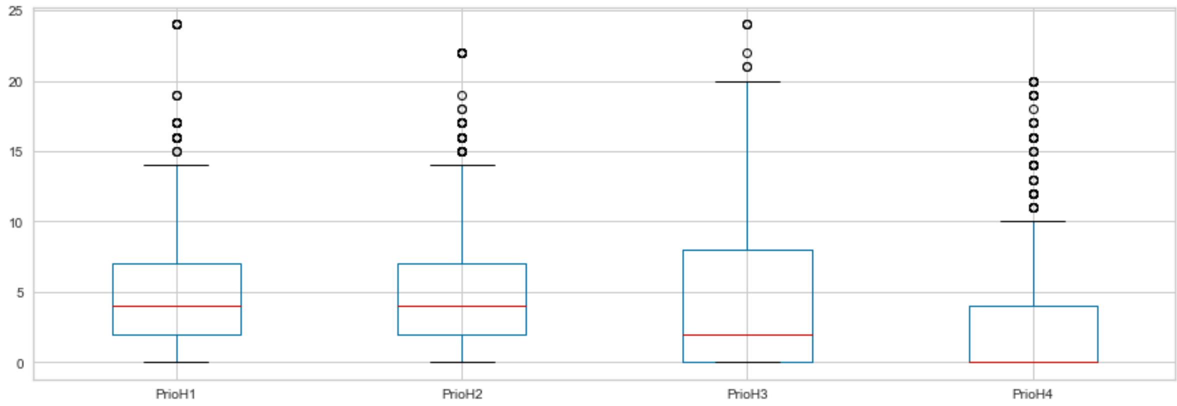
Entrée [129]:

```
# draw boxplots for nb of hours values
df.boxplot(column= ['PrioH1','PrioH2','PrioH3','PrioH4'])
```

executed in 197ms, finished 17:40:52 2019-09-19

Out[129]:

<matplotlib.axes._subplots.AxesSubplot at 0x1fb1374ab38>



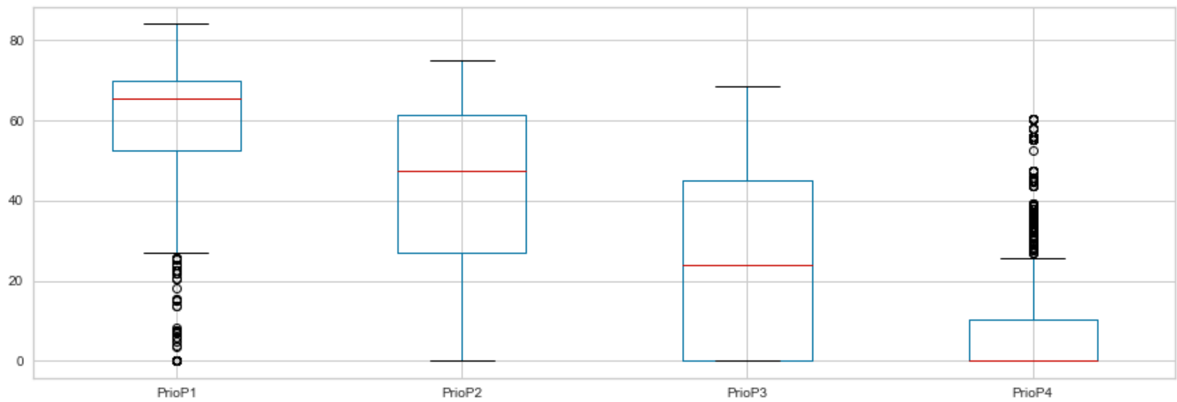
Entrée [130]:

```
# draw boxplots for power values
df.boxplot(column= ['PrioP1','PrioP2','PrioP3','PrioP4'])
```

executed in 189ms, finished 17:40:53 2019-09-19

Out[130]:

<matplotlib.axes._subplots.AxesSubplot at 0x1fb138bda20>



Entrée [131]:

```
# count number of unique pair combinations
cols =  ['PrioH1','PrioH2','PrioH3','PrioH4', 'PrioP1','PrioP2','PrioP3','PrioP4']
print(df.filter(cols).drop_duplicates().count()) # = 688 ! vs 1917 total number of entries
```

executed in 12ms, finished 17:40:53 2019-09-19

```
PrioH1    688
PrioH2    688
PrioH3    688
PrioH4    688
PrioP1    688
PrioP2    688
PrioP3    688
PrioP4    688
dtype: int64
```

Entrée [132]:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import pylab as pl
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

executed in 7ms, finished 17:40:53 2019-09-19

Entrée [133]:

```
#define dataset
df_cluster = df.filter(cols, axis=1) # not including ["TotalNbHours"] in the list

df_cluster.head()
```

executed in 16ms, finished 17:40:53 2019-09-19

Out[133]:

	PrioH1	PrioH2	PrioH3	PrioH4	PrioP1	PrioP2	PrioP3	PrioP4
Date								
2014-04-01	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-02	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-03	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-04	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-05	4.0	4.0	8.0	0.0	72.6	66.0	42.9	0.0

Entrée [134]:

```
# check basic statistics
display(round(df_cluster.describe(),2))
```

executed in 33ms, finished 17:40:53 2019-09-19

	PrioH1	PrioH2	PrioH3	PrioH4	PrioP1	PrioP2	PrioP3	PrioP4
count	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00	1917.00
mean	4.68	4.87	3.86	2.52	59.03	42.46	24.06	8.06
std	3.32	3.95	4.31	4.16	16.40	21.27	21.59	14.10
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	2.00	2.00	0.00	0.00	52.50	27.00	0.00	0.00
50%	4.00	4.00	2.00	0.00	65.40	47.40	24.00	0.00
75%	7.00	7.00	8.00	4.00	69.90	61.50	45.00	10.50
max	24.00	22.00	24.00	20.00	84.00	75.00	68.65	60.30

Entrée [135]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2).fit_transform(df_cluster)
# Save components to a DataFrame
PCA_components = pd.DataFrame(pca)
```

executed in 12ms, finished 17:40:53 2019-09-19

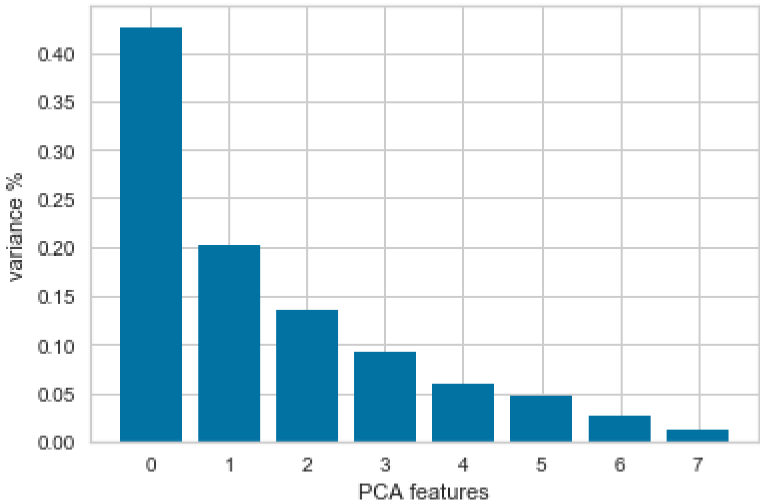
Entrée [136]:

```
#PCA_components
from sklearn.preprocessing import StandardScaler
# Standardize the data to have a mean of ~0 and a variance of 1
X_std = StandardScaler().fit_transform(df_cluster)

# Create a PCA instance: pca
pca = PCA(n_components=8)
principalComponents = pca.fit_transform(X_std)
# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_)
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
```

executed in 722ms, finished 17:40:53 2019-09-19

Out[136]:

```
([<matplotlib.axis.XTick at 0x1fb5d9c2d30>,
  <matplotlib.axis.XTick at 0x1fb1350e908>,
  <matplotlib.axis.XTick at 0x1fb136f8b00>,
  <matplotlib.axis.XTick at 0x1fb5fceec50>,
  <matplotlib.axis.XTick at 0x1fb5fcee668>,
  <matplotlib.axis.XTick at 0x1fb5fcee6a0>,
  <matplotlib.axis.XTick at 0x1fb5fbf3f98>,
  <matplotlib.axis.XTick at 0x1fb5fbf3a20>],
 <a list of 8 Text xticklabel objects>)
```



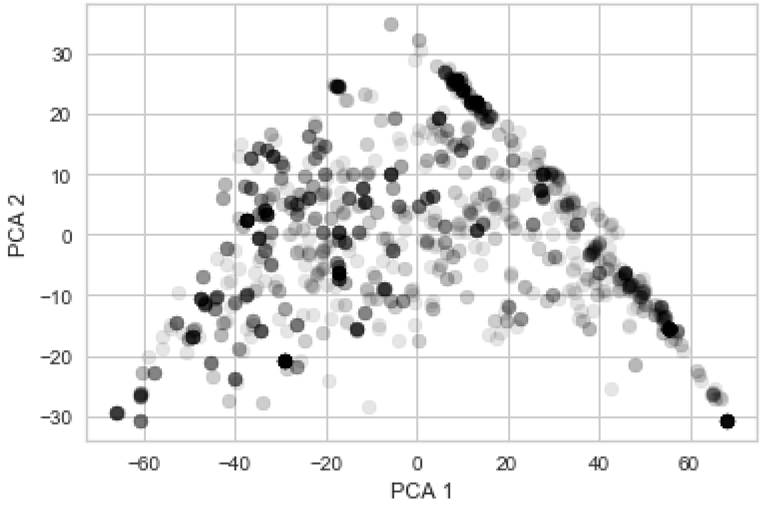
Entrée [137]:

```
plt.scatter(PCA_components[0], PCA_components[1], alpha=.1, color='black')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
```

executed in 194ms, finished 17:40:54 2019-09-19

Out[137]:

```
Text(0, 0.5, 'PCA 2')
```



Entrée [138]:

```
#standardize the data to normal distribution
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
mms.fit(df_cluster)
data_transformed = mms.transform(df_cluster)
```

executed in 8ms, finished 17:40:54 2019-09-19

Entrée [139]:

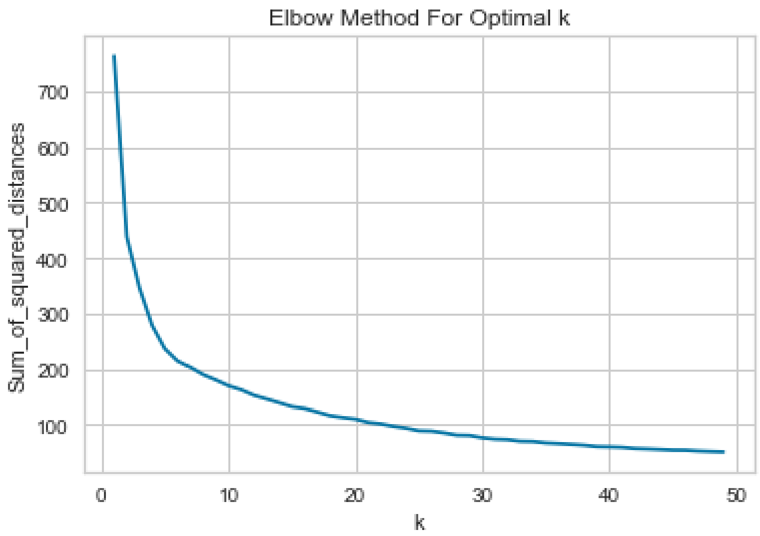
```
Sum_of_squared_distances = []
SilouhetteAvg = []
K = range(1,50)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(data_transformed)
    Sum_of_squared_distances.append(km.inertia_)
    #silhouette_avg = silhouette_score(df_cluster, km)
    #SilouhetteAvg.append(silhouette_avg)
```

executed in 10.3s, finished 17:41:04 2019-09-19

Entrée [140]:

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```

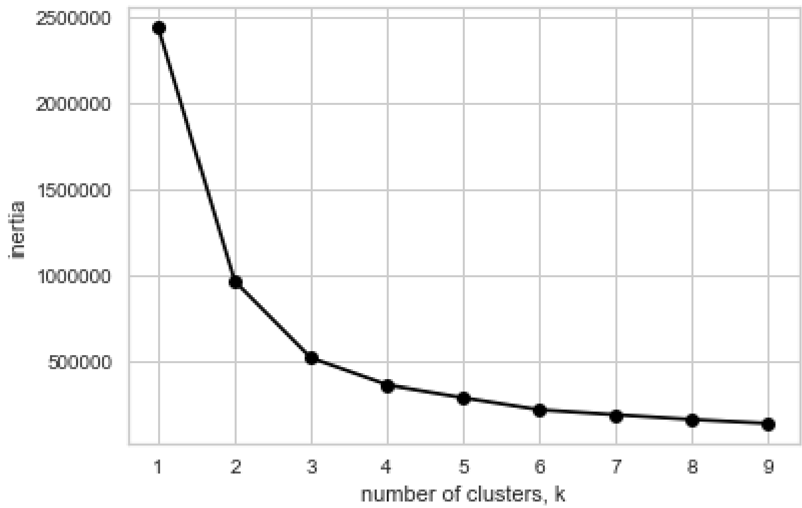
executed in 144ms, finished 17:41:04 2019-09-19



Entrée [141]:

```
# ref: https://medium.com/@dmitriy.kavyazin/principal-component-analysis-and-k-means-cluste
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

executed in 658ms, finished 17:41:05 2019-09-19



2.2 k-medoids clustering method implementation

Entrée [142]:

```
df_cluster.head()
```

executed in 14ms, finished 17:41:05 2019-09-19

Out[142]:

	PrioH1	PrioH2	PrioH3	PrioH4	PrioP1	PrioP2	PrioP3	PrioP4
Date								
2014-04-01	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-02	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-03	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-04	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0
2014-04-05	4.0	4.0	8.0	0.0	72.6	66.0	42.9	0.0

Entrée [143]:

```
# define number of clusters
ClusterNb = 8
IndexStep = int(round(1917/ClusterNb))

# Set random initial medoids.
initial_medoids = [i*IndexStep for i in range(1,ClusterNb)]

# use the scaled input values
cluster_input = data_transformed

# Create instance of K-Medoids algorithm.
kmedoids_instance = kmedoids(cluster_input, initial_medoids)
# Run cluster analysis and obtain results.
kmedoids_instance.process()
clusters = kmedoids_instance.get_clusters()
centers = kmedoids_instance.get_medoids()
print(centers)
#print(cluster_input[centers])
# build list of cluster ID to assign back to dataframe
cluster_allocation = np.zeros(len(df_cluster.index))
for clust_nb in range(len(clusters)):
    for i in range(len(clusters[clust_nb])):
        #print("Cluster nr %i, element nr %i, valeur %i" %(clust_nb, i, clusters[clust_nb][
        cluster_allocation[clusters[clust_nb][i]] = int(clust_nb)

cluster_allocation = cluster_allocation.astype(int)
df_cluster["k_medoids"] = cluster_allocation
df_cluster.index = df_cluster["k_medoids"]
```

executed in 44ms, finished 17:41:05 2019-09-19

```
[249, 692, 622, 9, 1197, 1137, 742]
```

Entrée [144]:

```
df_cluster.index.get_level_values('k_medoids')
```

executed in 8ms, finished 17:41:05 2019-09-19

Out[144]:

```
Int64Index([1, 1, 1, 1, 1, 1, 1, 1, 1, 3,
            ...
            5, 5, 5, 5, 5, 5, 5, 5, 5, 5],
           dtype='int64', name='k_medoids', length=1917)
```

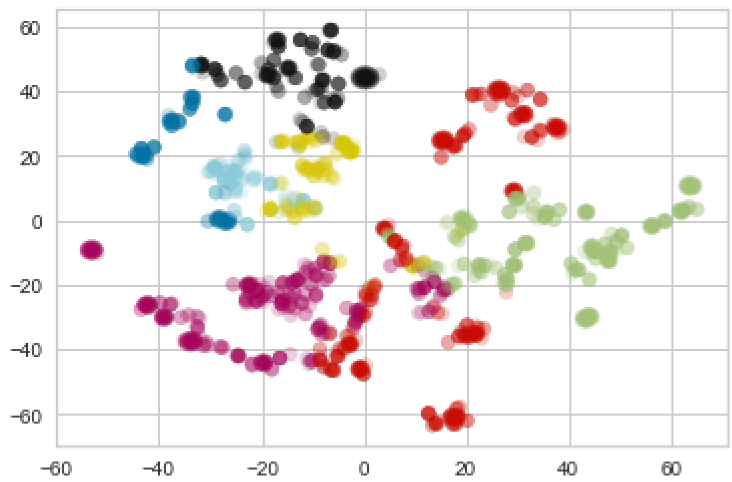
Entrée [145]:

```
# plot the resulting cluster allocation with TSNE algorithm
from sklearn.manifold import TSNE
import matplotlib.colors

np.random.seed(42) # set seed for reproducibility

color_list = 'rgbkymc'
cluster_values = sorted(df_cluster.index.get_level_values('k_medoids').unique())

tsne = TSNE()
results_tsne = tsne.fit_transform(data_transformed) #df_cluster)

cmap = matplotlib.colors.LinearSegmentedColormap.from_list(cluster_values, color_list)

plt.scatter(results_tsne[:,0], results_tsne[:,1],
    c=df_cluster.index.get_level_values('k_medoids'),
    cmap=cmap,
    alpha=0.2,
    )
```

executed in 11.7s, finished 17:41:16 2019-09-19

Out[145]:

```
<matplotlib.collections.PathCollection at 0x1fb5fee51d0>
```



Entrée [146]:

```
df_cluster.index.get_level_values('k_medoids').unique()
```

executed in 7ms, finished 17:41:17 2019-09-19

Out[146]:

```
Int64Index([1, 3, 5, 0, 4, 2, 6], dtype='int64', name='k_medoids')
```

Entrée [147]:

```
df_cluster.index.get_level_values('k_medoids')
```

executed in 6ms, finished 17:41:17 2019-09-19

Out[147]:

```
Int64Index([1, 1, 1, 1, 1, 1, 1, 1, 1, 3,
            ...
            5, 5, 5, 5, 5, 5, 5, 5, 5, 5],
           dtype='int64', name='k_medoids', length=1917)
```

Entrée [148]:

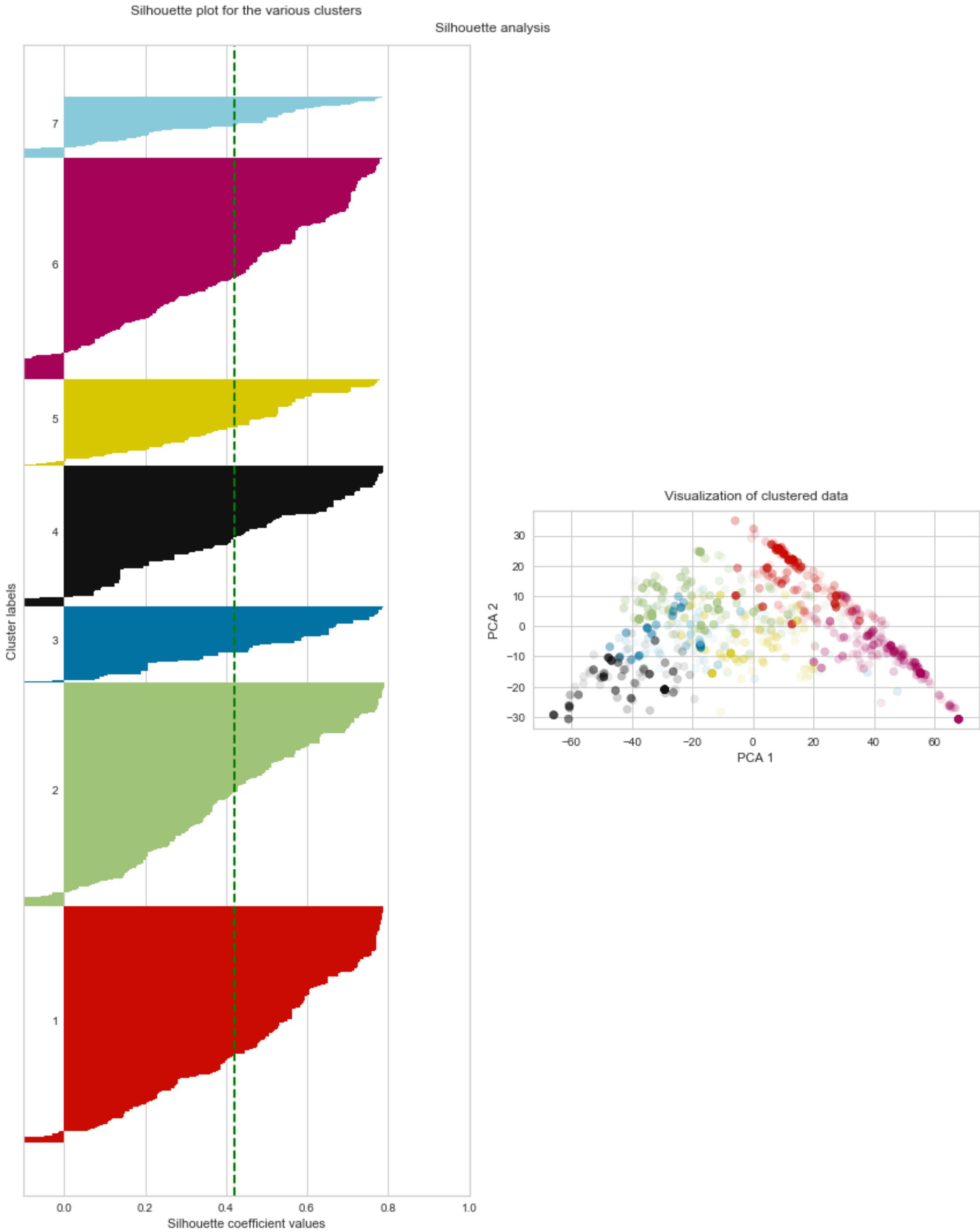
```
# PLot silhouette scores
def plotSilhouette(data_transformed, clusters, cluster_allocation, centers, usedMetric=""):
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(12, 15)

    labels = cluster_allocation
    centroids = centers

    # Get silhouette scores
    silhouette_vals =silhouette(data_transformed, clusters).process().get_score()

    my_colors = 'rgbkymcrgbkymc'

    # Silhouette plot
    y_ticks = []
    y_lower, y_upper = 0, 0
    for i, cluster in enumerate(np.unique(labels)):
        cluster_silhouette_vals = [i for (i, v) in zip(silhouette_vals, labels) if v==clust
        cluster_silhouette_vals.sort()
        y_upper += len(cluster_silhouette_vals)
        ax1.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor='none', height
                 color = my_colors[cluster] )
        ax1.text(-0.03, (y_lower + y_upper) / 2, str(i + 1))
        y_lower += len(cluster_silhouette_vals)

    # Get the average silhouette score and plot it
    avg_score = np.mean(silhouette_vals)
    ax1.axvline(avg_score, linestyle='--', linewidth=2, color='green')
    ax1.set_yticks([])
    ax1.set_xlim([-0.1, 1])
    ax1.set_xlabel('Silhouette coefficient values')
    ax1.set_ylabel('Cluster labels')
    ax1.set_title('Silhouette plot for the various clusters', y=1.02);

    # Scatter plot of data colored with labels, along PCA axis

    ax2.scatter(PCA_components[0], PCA_components[1], alpha=.1, cmap=cmap,
                c = [my_colors[i] for i in cluster_allocation]  )
    ax2.set_xlabel('PCA 1')
    ax2.set_ylabel('PCA 2')

    #ax2.scatter(X_std[:, 0], X_std[:, 1], c=labels)
    #ax2.scatter(centroids[:, 0], centroids[:, 1], marker='*', c='r', s=250)

    ax2.set_title('Visualization of clustered data', y=1.02)
    ax2.set_aspect('equal')
    plt.tight_layout()
    plt.suptitle('Silhouette analysis');
```

executed in 11ms, finished 17:41:17 2019-09-19

Entrée [149]:

```
# plot
plotSilhouette(data_transformed, clusters, cluster_allocation, centers)
```

executed in 4.75s, finished 17:41:21 2019-09-19



3 Classification problem to predict typical power blocks

Entrée [150]:

```
df_cluster = df.filter(cols, axis=1) # power and no of hours columns

# define number of culsters
ClusterNb = 10
IndexStep = int(round(1917/ClusterNb))
# Set random initial medoids.
initial_medoids = [i*IndexStep for i in range(1,ClusterNb)] # initial centroids distributed
# transform data frame into list (of lists)
cluster_input = df_cluster.values
# Create instance of K-Medoids algorithm.
kmedoids_instance = kmedoids(cluster_input, initial_medoids)
# Run cluster analysis and obtain results.
kmedoids_instance.process()
clusters = kmedoids_instance.get_clusters()
centers = kmedoids_instance.get_medoids()
print(centers)
#print(cluster_input[centers])
# build list of cluster ID to assign back to dataframe
cluster_allocation = np.zeros(len(df_cluster.index))
for clust_nb in range(len(clusters)):
    for i in range(len(clusters[clust_nb])):
        #print("Cluster nr %i, element nr %i, valeur %i" %(clust_nb, i, clusters[clust_nb][
        cluster_allocation[clusters[clust_nb][i]] = int(clust_nb)

df_cluster["cluster_id"] = cluster_allocation
df_cluster["cluster_id"] = df_cluster["cluster_id"].astype(int)
```

executed in 47ms, finished 17:41:21 2019-09-19

```
[363, 1420, 483, 667, 9, 1627, 231, 650, 724]
```

Entrée [151]:

```
df_cluster.iloc[centers[5]]
```

executed in 8ms, finished 17:41:21 2019-09-19

Out[151]:

```
PrioH1        4.0
PrioH2        5.0
PrioH3        4.0
PrioH4        0.0
PrioP1       55.5
PrioP2       28.8
PrioP3       12.6
PrioP4        0.0
cluster_id    5.0
Name: 2018-09-14 00:00:00, dtype: float64
```

Entrée [152]:

```
# check cluster "quality", i.e. delta of values towards medoids.
# Check energy value for first pair, find occurence where difference lies beyond threshold
nbGtThreshold = 0
colCluster = 8
for index, row in df_cluster.iterrows():
    #print(row.values)
    deltaP1 = row["PrioH1"]*row["PrioP1"]- (df_cluster.iloc[centers[int(row[colCluster])]][
                                              df_cluster.iloc[centers[int(row[colCluster])]]["PrioP

    if (deltaP1> 50):
        #print(round(deltaP1))
        nbGtThreshold = nbGtThreshold+1

print("Nb of values beyond threshold:",nbGtThreshold)
```

executed in 782ms, finished 17:41:22 2019-09-19

```
Nb of values beyond threshold: 668
```

Entrée [153]:

```
# import maximum production energy best forecast, to use as input variable
# but only in the prediction part, of course
df_MLPprediction = pd.read_csv("ForecastMax prodMLP (10,7,1) Act(lin, relu, relu) Dropout (
# rename first column
df_MLPprediction.rename(columns={ df_MLPprediction.columns[0]: "Prediction"}, inplace=True)

df_MLPprediction.head()
```

executed in 21ms, finished 17:41:22 2019-09-19

Out[153]:

	Prediction	Ground truth	Mean train values	Baseline
Date				
2018-12-21	935.94086	1241.052632	727.020546	855.1
2018-12-22	864.19230	1241.052632	727.020546	853.5
2018-12-23	861.32270	1241.052632	727.020546	824.0
2018-12-24	1002.55360	1241.052632	727.020546	924.7
2018-12-25	910.25616	1241.052632	727.020546	868.5

Entrée [154]:

```
# splits the input dataframe into train, test for target and input features, using the prov
def GetDataSplit(df_input, regressors, target_feature, target_baseline, ratio):
    # We split using a 90/10 ratio (parameter), but keeping the data in chronological order
    CutPoint = round(len(df.index)*ratio)
    df_model = df_input.filter(regressors, axis=1)

    xTrain = df_model.iloc[:CutPoint, :]
    xTest  = df_model.iloc[CutPoint:, :]
    yTrain = df[target_feature][:CutPoint]
    yTest  = df[target_feature][CutPoint:]
    yBaseLine = df[target_baseline][CutPoint:] if target_baseline != [] else []
    return [xTrain, xTest, yTrain, yTest, yBaseLine ]
```

executed in 7ms, finished 17:41:22 2019-09-19

Entrée [155]:

```
df_cluster.head()
```

executed in 16ms, finished 17:41:22 2019-09-19

Out[155]:

	PrioH1	PrioH2	PrioH3	PrioH4	PrioP1	PrioP2	PrioP3	PrioP4	cluster_id
Date									
2014-04-01	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0	2
2014-04-02	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0	2
2014-04-03	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0	2
2014-04-04	4.0	4.0	8.0	0.0	73.8	66.0	42.9	0.0	2
2014-04-05	4.0	4.0	8.0	0.0	72.6	66.0	42.9	0.0	2

Entrée [156]:

```
# define regression target and explanatory variables
target_feature = 'cluster_id'
regressors = ['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 [%]', 'Availability plant 3 [%]', \
            'Availability plant 4 [%]','Weekend','Max prod']

# implement mutli-output random forest
fullRegressors = regressors

# add cluster allocation column to our dataset
df["cluster_id"] = df_cluster["cluster_id"]

xTrain, xTest, yTrain, yTest, yBaseline =  GetDataSplit(df, fullRegressors, target_feature,

# for the prediction part, replace the max energy with the forecasted value
xTest["Max prod"] = df_MLPprediction["Prediction"]
xTest.head()
```

executed in 254ms, finished 17:41:22 2019-09-19

Out[156]:

Date	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 [%]	Availability plant 3 [%]
2018-12-21	51.0	22.9	68.7	41.4	0.34519	30000.0	1.0	1.0	1.0
2018-12-22	55.0	28.3	63.5	44.6	0.34262	30000.0	1.0	1.0	1.0
2018-12-23	43.0	30.3	58.4	42.2	0.34089	30000.0	1.0	1.0	1.0
2018-12-24	96.0	23.3	172.7	194.6	0.34020	30000.0	1.0	1.0	1.0
2018-12-25	69.0	32.0	78.1	95.2	0.34089	30000.0	1.0	1.0	1.0

3.1 Hourly Error measure

Entrée [157]:

```
# compute hourly error. Build first an hourly dataframe, then fill it up with hourly power
def hourlyErrorMeasure (dfPred, dfGroundTruth):
    # rounding function is ceiling function (not needed for clustering part)
    rounding_func = round

    # Compute hourly metrics for baseline

    # create an hourly dataframe over same period
    # compute hourly values filling up a 24 hours vector
    df_hourly = pd.DataFrame(
            {'Hours': pd.date_range(dfGroundTruth.index.min(), dfGroundTruth.index.max(), f
        )
    df_hourly.index = df_hourly['Hours']

    # hourly dataframe to store restults, out of complete one
    df_hourly_pred = df_hourly

    for mydf, mylabel in zip([dfPred, dfGroundTruth],["Pred","GT"]):

        firstDate = mydf.index.min()
        powerHours = np.zeros((len(dfPred.index)-1)*24)
        # loop over all date
        for myDate in dfPred.index:
            # index is date difference times 24
            index = (myDate-firstDate).days * 24
            # compute hourly vector of original values, using the 4 pairs (power/nb of hour
            currHour = index

            for iPair in range(1,4+1):
                pwrValue = mydf.loc[myDate, "PrioP"+str(iPair)]
                nbHours = int(rounding_func(mydf.loc[myDate, "PrioH"+str(iPair)]))
                if pwrValue > 0:
                    #print("Date %s, Pair %i, Pwr %d, Hours %f" %(myDate, iPair, pwrValue,
                    powerHours[currHour:currHour+nbHours] = pwrValue
                    currHour = currHour + nbHours
        df_hourly_pred[mylabel+"Power"]= powerHours

    RMSE = round(math.sqrt(metrics.mean_squared_error(df_hourly_pred["PredPower"],df_hourly
    MAE = round(metrics.mean_absolute_error(df_hourly_pred["PredPower"],df_hourly_pred["GTP
    R2 = round(metrics.r2_score( df_hourly_pred["PredPower"],df_hourly_pred["GTPower"])*100

    # compute error metrics over hourly values calculated above
    print('Hourly comparison: prediction vs. actual values')
    print('RMSE :'+ str(RMSE))
    print('MAE :'+ str(MAE))
    print('R^2 :'+ str(R2))

    # Compute error towards daily energy as well
    # add column with total daily energy to input dataframe, computed back from blocks
    dfPred["DailyMaxEnergy"] = 0
    for i in range (1,4+1):
        dfPred["DailyMaxEnergy"] +=  dfPred["PrioP"+str(i)]*dfPred["PrioH"+str(i)]

    DailyRMSE = round(math.sqrt(metrics.mean_squared_error(dfGroundTruth["Max prod"],dfPred
    DailyMAE = round(metrics.mean_absolute_error(dfGroundTruth["Max prod"],dfPred["DailyMax
    DailyR2 = round(metrics.r2_score(dfGroundTruth["Max prod"],dfPred["DailyMaxEnergy"])*10

    print('Daily comparison (max prod): prediction vs. actual values')
    print('RMSE :'+ str(DailyRMSE))
```

```
    print('MAE :'+ str(DailyMAE))
    print('R^2 :'+ str(DailyR2))

    return [RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2]
```

executed in 16ms, finished 17:41:22 2019-09-19

Entrée [158]:

```
# Import labraries for classification problem

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.ensemble.forest import RandomForestClassifier
```

executed in 6ms, finished 17:41:22 2019-09-19

Entrée [159]:

```
# classification algorithm

np.random.seed(42) # set seed for reproducibility
cols = ['PrioH1','PrioH2','PrioH3','PrioH4', 'PrioP1','PrioP2','PrioP3','PrioP4']

# Forecast using Random Forest (1000 trees)
RFmodel = RandomForestClassifier(n_estimators=1000)

# Forecast using k-NN (4 neighbours)
kNNmodel = KNeighborsClassifier(n_neighbors=4)

for myModel in [kNNmodel, RFmodel]:
    # Fit the RF model with features and labels.
    classifyModel = myModel.fit(xTrain, yTrain)

    # predictions
    xPredict = classifyModel.predict(xTest)

    report = classification_report(yTest, xPredict)
    print(myModel)
    print(report)

    # assign the predicted power blocks from the corresponding clusters
    dfxPredict = pd.DataFrame(index=xTest.index, columns=cols)
    dfxPredict["clustedId"] = xPredict

    for prefix in ["PrioP","PrioH"]:
        for iPair in range(1,4+1):
            res = np.zeros(len(xPredict))
            for myIndex in range(len(xPredict)):
                res[myIndex] =  df.iloc[centers[xPredict[myIndex]]][prefix+str(iPair)]
            dfxPredict[prefix+str(iPair)] = res

    # compute error metrics on hourly values, using the prediction
    hourlyErrorMeasure(dfxPredict, df.iloc[1917-192:,])
```

executed in 3.30s, finished 17:41:26 2019-09-19

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.p
y:1439: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.10      0.19       125
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         6
           4       0.40      0.25      0.31        16
           5       0.17      0.15      0.16        20
           6       0.00      0.00      0.00         2
           7       0.00      0.00      0.00         0
           8       1.00      0.09      0.16        23

    accuracy                           0.11       192
   macro avg       0.29      0.07      0.09       192
```

```
weighted avg       0.82      0.11      0.18       192

Hourly comparison: prediction vs. actual values
RMSE :29.32
MAE :22.11
R^2 :-37.24
Daily comparison (max prod): prediction vs. actual values
RMSE :578.55
MAE :511.93
R^2 :-56.81
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.p
y:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.p
y:1439: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=N
one,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
              precision    recall  f1-score   support

           1       0.98      0.68      0.80       125
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         6
           4       0.45      0.62      0.53        16
           5       0.28      0.80      0.42        20
           6       0.00      0.00      0.00         2
           8       1.00      0.09      0.16        23

    accuracy                           0.59       192
   macro avg       0.39      0.31      0.27       192
weighted avg       0.82      0.59      0.63       192

Hourly comparison: prediction vs. actual values
RMSE :14.28
MAE :8.19
R^2 :60.06
Daily comparison (max prod): prediction vs. actual values
RMSE :239.3
MAE :167.63
R^2 :73.17
```

Entrée [160]:

```
# performs clustering on input data filtered by provided columns
def clusterCreation (iNbClusters, dfOrig, cols, targetType="", plotSil=False):

    df_cluster = dfOrig.filter(cols, axis=1) # filter on columns used for clustering

    mms = MinMaxScaler()
    mms.fit(df_cluster)
    cluster_input = mms.transform(df_cluster)

    # define number of culsters
    ClusterNb = iNbClusters
    IndexStep = int(round(len(dfOrig.index)/ClusterNb))
    # Set random initial medoids.
    initial_medoids = [i*IndexStep for i in range(1,ClusterNb)] # initial centroids distrib

    # transform data frame into list (of lists)
    #cluster_input = df_cluster.values

    # depending on the target type, we use a different metric
    if ("targetType" == "Vector"):
        metric = distance_metric(type_metric.MANHATTAN)
    else:
        metric = distance_metric(type_metric.EUCLIDEAN)

    # Create instance of K-Medoids algorithm.
    kmedoids_instance = kmedoids(cluster_input, initial_medoids,metric = metric)
    # Run cluster analysis and obtain results.
    kmedoids_instance.process()
    clusters = kmedoids_instance.get_clusters()
    centers = kmedoids_instance.get_medoids()
    print("Nb of clusters %i" %(len(clusters)))
    print("Nb of centers %i, list:%s" %(len(centers),centers))

    # build list of cluster ID to assign back to dataframe
    #NB: cluster ids are indexes of data points in the dataframe
    cluster_allocation = np.zeros(len(df_cluster.index))
    for clust_nb in range(len(clusters)):
        for i in range(len(clusters[clust_nb])):
            #print("Cluster nr %i, element nr %i, valeur %i" %(clust_nb, i, clusters[clust_
            cluster_allocation[clusters[clust_nb][i]] = int(clust_nb)

    cluster_allocation = cluster_allocation.astype(int)
    df_cluster["cluster_id"] = cluster_allocation
    df_cluster["cluster_id"] = df_cluster["cluster_id"].astype(int) # force type

    if plotSil:
        plotSilhouette(cluster_input, clusters, cluster_allocation, centers, metric)

    return [df_cluster, centers]
```

executed in 11ms, finished 17:41:26 2019-09-19

3.2 Cluster classification

Entrée [161]:

```
def clusterClassification (modelList, modelNameList, xTrain, yTrain, xTest, yTest,
                           dfFull, dfCentroids, targetType):
    # create dataframe to store results
    resDf = pd.DataFrame(columns=["Algorithm", "Target Type", "NbClusters", "RMSE", "MAE",
                                  "Daily RMSE", "Daily MAE", "Daily R2"])

    for myModel, MyModelName in zip(modelList,modelNameList): #[kNNmodel, RFmodel]
        print("**** "+MyModelName+" classification *****")

        # Fit model with features and labels.
        classifyModel = myModel.fit(xTrain, yTrain)

        # predictions
        xPredict = classifyModel.predict(xTest)

        report = classification_report(yTest, xPredict, output_dict=True)
        print("Result of classification with "+MyModelName)
        print("**** accuracy *****",report['accuracy'])

        # assign the predicted power blocks from the corresponding clusters
        # we resize back the power values using the predicted energy!
        dfxPredict = pd.DataFrame(index=xTest.index, columns=cols)
        dfxPredict["clustedId"] = xPredict

        for prefix in ["PrioP","PrioH"]:
            for iPair in range(1,4+1):
                res = np.zeros(len(xPredict))
                for myIndex in range(len(xPredict)):
                    res[myIndex] = dfCentroids.iloc[xPredict[myIndex]][prefix+str(iPair)]
                if prefix == "PrioP": #  resize power columns
                    res = res*xTest["Max prod"]
                dfxPredict[prefix+str(iPair)] = res

        # plot enrgy block values
        nbCluster = len(dfCentroids.index)
        plotEnergyBlocks(dfxPredict, "Block", "PredBlocksClust"+targetType+str(nbCluster)+M
        plotEnergyBlocks(dfFull.iloc[1917-192:,], "Block", "TruthClust"+str(nbCluster))

        # compute error metrics on hourly values, using the prediction
        RSME, MAE, R2, DailyRMSE, DailyMAE, DailyR2 = \
                                hourlyErrorMeasure(dfxPredict, dfFull.iloc[1917-192:,])
        resDf = resDf.append({'Algorithm':MyModelName  , "Target Type": targetType, "NbClus
                              "Accuracy": report['accuracy'] ,
                              "RMSE":RSME, "MAE":MAE, "R^2": R2, "Daily RMSE": DailyRMSE,
                              "Daily MAE":DailyMAE, "Daily R2":DailyR2 } , ignore_index=Tru

    return resDf
```

executed in 13ms, finished 17:41:26 2019-09-19

Entrée [162]:

```
df.head()
```

executed in 24ms, finished 17:41:26 2019-09-19

Out[162]:

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 [%]	.
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 × 30 columns

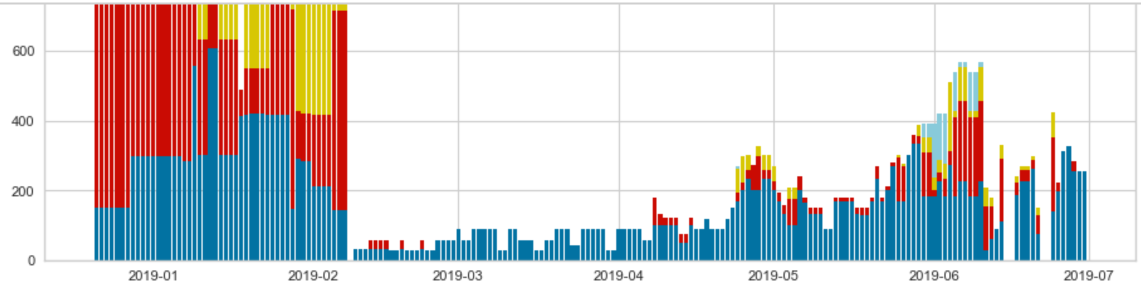
3.2.1 Grid search fo best number of clusters

Entrée [170]:

```
# grid search to identify best number of clusters, from to 60

initialClusterNb = 5 #5
finalCusterNb =  60 #60
stepNbCluster = 1
dfResult = pd.DataFrame(columns=["Algorithm", "NbClusters", "RMSE", "MAE", "R^2"])

# classification using Random Forest (500 trees)
RFmodel = RandomForestClassifier(n_estimators=1000)
# classification using k-NN (4 neighbours)
kNNmodel = KNeighborsClassifier(n_neighbors=4)

modelList = [RFmodel, kNNmodel]
modelNameList = ["RF", "kNN"]
targetTypes = ["Power-Nohours", "Vector"]

# resize power values before clustering
df_resizePower= df.copy()
for i in range (1,4+1):
    df_resizePower["PrioP"+str(i)] =  df_resizePower["PrioP"+str(i)]/df_resizePower["Max pr
# as there as some days with energy = 0, fill up NaN with 0
df_resizePower.replace(np.inf, np.nan, inplace=True)
df_resizePower.fillna(0,inplace=True)

# create the 24 hours vector columns in the dataframe, using the blocks definition
dfVector = get24hoursEnergyVector("PrioH", "PrioP", df_resizePower, "P")
for i in range(24):
    df_resizePower["P"+str(i)] =  dfVector["P"+str(i)]

for targetType in targetTypes:

    if targetType == "Vector":
        clusteringCols =  ['PrioH1','PrioH2','PrioH3','PrioH4', 'PrioP1','PrioP2','PrioP3',
    else:
        clusteringCols =  ['P'+str(i) for i in range(24)]

    for iNbCluster in range(initialClusterNb,finalCusterNb,stepNbCluster ):
        print("***********************************")
        print("** clustering %s with %i clusters **" %(targetType,iNbCluster))

        # perform clustering
        df_cluster, centers = clusterCreation(iNbCluster, df_resizePower, clusteringCols, t
        # copy the cluster allocation in the source dataframe
        df["cluster_id"] = df_cluster["cluster_id"]
        print("Number of centers:", len(centers))

        # create the centroid table, i.e. for each cluster ID the associated values
        dfCentroids = pd.DataFrame(index = [i for i in range(len(centers))], columns=["Clus
        for i in range(len(centers)):
            dfCentroids.iloc[i]["ClusterID"] = centers[i]
            dfCentroids.iloc[i][cols] = df_resizePower.iloc[centers[i]][cols]
        #force index to ClusterID values
        dfCentroids.index = dfCentroids["ClusterID"]

        xTrain, xTest, yTrain, yTest, yBaseline =  GetDataSplit(df, fullRegressors, target_
        # for the prediction part, replace the max energy with the forecasted value
        xTest["Max prod"] = df_MLPprediction["Prediction"]

        # performClassification and store results for reporting
```

```
        for index, row in clusterClassification(modelList, modelNameList,
                                                xTrain, yTrain, xTest, yTest, df, dfCentroi
            dfResult = dfResult.append(row)
```

executed in 20m 48s, finished 18:44:45 2019-09-19




```
Hourly comparison: prediction vs. actual values
```

Entrée [167]:

```
dfResult
```

executed in 34ms, finished 18:23:30 2019-09-19

Out[167]:

NbClusters	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Tar Ty
4	RF	4	15.66	9.87	60.02	72.916667	120.93	85.00	178.91	Pow Noho
4	kNN	4	15.00	9.53	60.35	72.916667	120.93	85.00	178.91	Pow Noho
5	RF	5	14.48	8.03	66.53	71.875000	120.93	85.00	178.91	Pow Noho
5	kNN	5	14.25	8.19	66.38	67.708333	120.93	85.00	178.91	Pow Noho
6	RF	6	13.13	7.61	69.47	50.000000	120.93	85.00	178.91	Pow Noho
6	kNN	6	13.57	7.83	68.74	54.687500	120.93	85.00	178.91	Pow Noho
7	RF	7	12.86	7.53	70.89	45.312500	120.93	85.00	178.91	Pow Noho
7	kNN	7	13.79	7.77	67.74	46.875000	120.93	85.00	178.91	Pow Noho
8	RF	8	13.66	7.79	67.73	55.208333	120.93	85.00	178.91	Pow Noho
8	kNN	8	14.26	8.08	66.15	55.208333	120.93	85.00	178.91	Pow Noho
9	RF	9	13.80	7.72	66.05	55.729167	120.93	85.00	178.91	Pow Noho
9	kNN	9	13.86	7.87	67.86	59.895833	120.93	85.00	178.91	Pow Noho
10	RF	10	13.11	7.13	69.09	41.145833	120.93	85.00	178.91	Pow Noho
10	kNN	10	13.21	7.66	68.62	34.895833	120.93	85.00	178.91	Pow Noho
11	RF	11	12.52	7.30	71.06	34.375000	120.08	85.23	177.58	Pow Noho
11	kNN	11	13.44	7.90	68.36	25.520833	121.35	85.01	178.91	Pow Noho
12	RF	12	13.29	7.39	68.12	42.187500	122.32	84.97	179.13	Pow Noho
12	kNN	12	13.84	7.78	67.22	35.937500	121.35	85.01	178.91	Pow Noho
13	RF	13	12.69	7.07	67.54	50.000000	120.93	85.00	178.91	Pow Noho
13	kNN	13	13.98	7.76	66.29	45.833333	120.93	85.00	178.91	Pow Noho
14	RF	14	12.34	7.04	72.20	42.708333	120.93	85.00	178.91	Pow Noho

	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Tarç Ty
NbClusters										
14	kNN	14	13.79	7.70	67.36	38.541667	120.93	85.00	178.91	Pow Noho
15	RF	15	14.17	7.84	65.99	20.312500	119.82	85.23	177.55	Pow Noho
15	kNN	15	13.91	7.76	66.88	26.562500	121.35	85.01	178.91	Pow Noho
16	RF	16	12.66	7.06	71.37	26.041667	120.19	85.22	177.59	Pow Noho
16	kNN	16	12.93	7.30	70.77	30.208333	118.40	85.40	176.54	Pow Noho
17	RF	17	12.74	7.08	69.89	28.645833	120.93	85.00	178.91	Pow Noho
17	kNN	17	13.26	7.47	68.91	25.520833	120.93	85.00	178.91	Pow Noho
18	RF	18	12.70	7.39	70.74	25.000000	120.19	85.22	177.59	Pow Noho
18	kNN	18	12.84	7.41	70.28	18.229167	121.35	85.01	178.91	Pow Noho
...	...	...	...	...	...	...	...	...	...	
44	RF	44	13.54	7.84	69.55	18.750000	120.93	85.00	178.91	Vec
44	kNN	44	15.18	8.37	64.20	9.375000	120.93	85.00	178.91	Vec
45	RF	45	14.18	8.32	67.61	14.062500	118.72	85.26	177.37	Vec
45	kNN	45	12.64	7.35	71.90	5.208333	120.30	85.04	178.69	Vec
46	RF	46	12.92	7.66	70.45	21.875000	118.72	85.26	177.37	Vec
46	kNN	46	12.36	7.14	72.57	8.333333	115.59	85.64	175.09	Vec
47	RF	47	14.03	8.25	67.68	16.145833	118.72	85.26	177.37	Vec
47	kNN	47	13.80	7.71	67.43	5.729167	117.17	85.41	176.45	Vec
48	RF	48	12.85	7.65	69.48	7.812500	120.93	85.00	178.91	Vec
48	kNN	48	12.76	7.60	70.33	5.208333	120.93	85.00	178.91	Vec
49	RF	49	12.60	7.42	70.88	11.979167	118.72	85.26	177.37	Vec
49	kNN	49	12.17	7.27	71.87	5.729167	120.30	85.04	178.69	Vec
50	RF	50	12.81	7.59	70.31	8.854167	118.72	85.26	177.37	Vec
50	kNN	50	12.17	7.27	71.87	5.729167	120.30	85.04	178.69	Vec
51	RF	51	13.89	8.11	67.21	15.625000	120.93	85.00	178.91	Vec
51	kNN	51	14.31	8.08	66.88	7.291667	120.93	85.00	178.91	Vec
52	RF	52	13.12	7.79	69.37	17.708333	118.72	85.26	177.37	Vec
52	kNN	52	14.39	7.95	66.79	8.333333	115.59	85.64	175.09	Vec
53	RF	53	13.01	7.53	70.17	18.229167	116.92	85.38	176.64	Vec
53	kNN	53	14.45	8.14	66.32	4.687500	120.30	85.04	178.69	Vec
54	RF	54	13.55	8.08	66.94	13.020833	119.36	85.11	178.29	Vec
54	kNN	54	13.47	7.77	69.10	3.645833	120.30	85.04	178.69	Vec

NbClusters	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Tar Ty
55	RF	55	12.79	7.45	71.33	14.062500	117.98	85.28	177.26	Vec
55	kNN	55	12.71	7.41	71.07	2.604167	118.92	85.21	177.67	Vec
56	RF	56	13.54	7.78	68.40	12.500000	118.67	85.18	177.86	Vec
56	kNN	56	12.94	7.52	70.60	4.687500	118.92	85.21	177.67	Vec
57	RF	57	12.79	7.68	69.01	14.583333	118.72	85.26	177.37	Vec
57	kNN	57	12.41	7.17	72.09	5.729167	115.59	85.64	175.09	Vec
58	RF	58	14.19	8.08	67.12	21.354167	120.93	85.00	178.91	Vec
58	kNN	58	13.57	7.95	69.22	10.416667	120.93	85.00	178.91	Vec

220 rows × 10 columns

Entrée [172]:

```
dfResult.index = dfResult["NbClusters"]
dfResult["Accuracy"] = dfResult["Accuracy"]*100
```

executed in 4ms, finished 18:45:09 2019-09-19

Entrée [173]:

```
dfResult[dfResult["Target Type"]=="Vector"].iloc[5:,].head(10)
```

executed in 17ms, finished 18:45:10 2019-09-19

Out[173]:

NbClusters	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Targe Typ
6	kNN	6	13.51	8.46	66.09	41.145833	120.93	85.0	178.91	Vecto
7	RF	7	13.90	9.34	64.76	42.708333	120.93	85.0	178.91	Vecto
7	kNN	7	13.03	7.95	70.16	16.145833	120.93	85.0	178.91	Vecto
8	RF	8	13.87	8.18	68.06	60.416667	120.93	85.0	178.91	Vecto
8	kNN	8	14.58	8.51	65.28	53.645833	120.93	85.0	178.91	Vecto
9	RF	9	13.64	8.72	65.84	48.958333	120.93	85.0	178.91	Vecto
9	kNN	9	15.73	9.72	57.57	19.791667	120.93	85.0	178.91	Vecto
10	RF	10	13.95	8.41	67.90	45.312500	120.93	85.0	178.91	Vecto
10	kNN	10	12.88	7.92	66.91	16.145833	120.93	85.0	178.91	Vecto
11	RF	11	14.43	8.71	66.46	28.125000	120.93	85.0	178.91	Vecto

Entrée [174]:

```
dfResult[dfResult["Target Type"]=="Power-Nohours"].iloc[3:,].head(10)
```

executed in 16ms, finished 18:45:15 2019-09-19

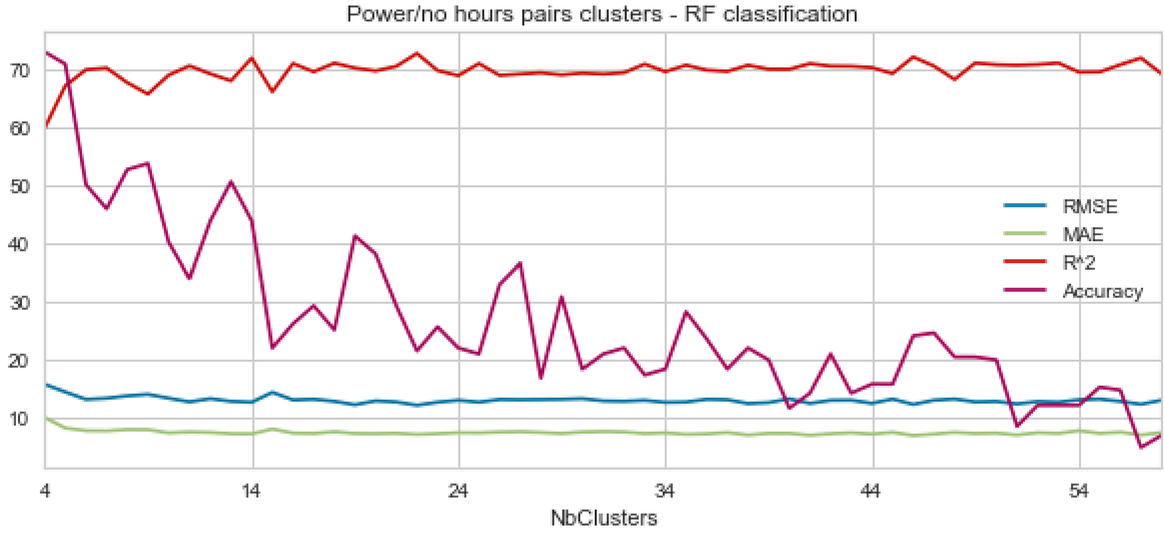
Out[174]:

NbClusters	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Targ Ty
5	kNN	5	14.25	8.19	66.38	67.708333	120.93	85.0	178.91	Pow Nohou
6	RF	6	12.99	7.58	69.85	50.000000	120.93	85.0	178.91	Pow Nohou
6	kNN	6	13.57	7.83	68.74	54.687500	120.93	85.0	178.91	Pow Nohou
7	RF	7	13.18	7.54	70.15	45.833333	120.93	85.0	178.91	Pow Nohou
7	kNN	7	13.79	7.77	67.74	46.875000	120.93	85.0	178.91	Pow Nohou
8	RF	8	13.59	7.77	67.61	52.604167	120.93	85.0	178.91	Pow Nohou
8	kNN	8	14.26	8.08	66.15	55.208333	120.93	85.0	178.91	Pow Nohou
9	RF	9	13.85	7.75	65.65	53.645833	120.93	85.0	178.91	Pow Nohou
9	kNN	9	13.86	7.87	67.86	59.895833	120.93	85.0	178.91	Pow Nohou
10	RF	10	13.19	7.20	68.91	40.104167	120.93	85.0	178.91	Pow Nohou

Entrée [175]:

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 4
dfResult[(dfResult["Algorithm"]=="RF")&(dfResult["Target Type"]=="Power-Nohours")][["RMSE",
plt.legend()
plt.show()
```

executed in 196ms, finished 18:45:19 2019-09-19



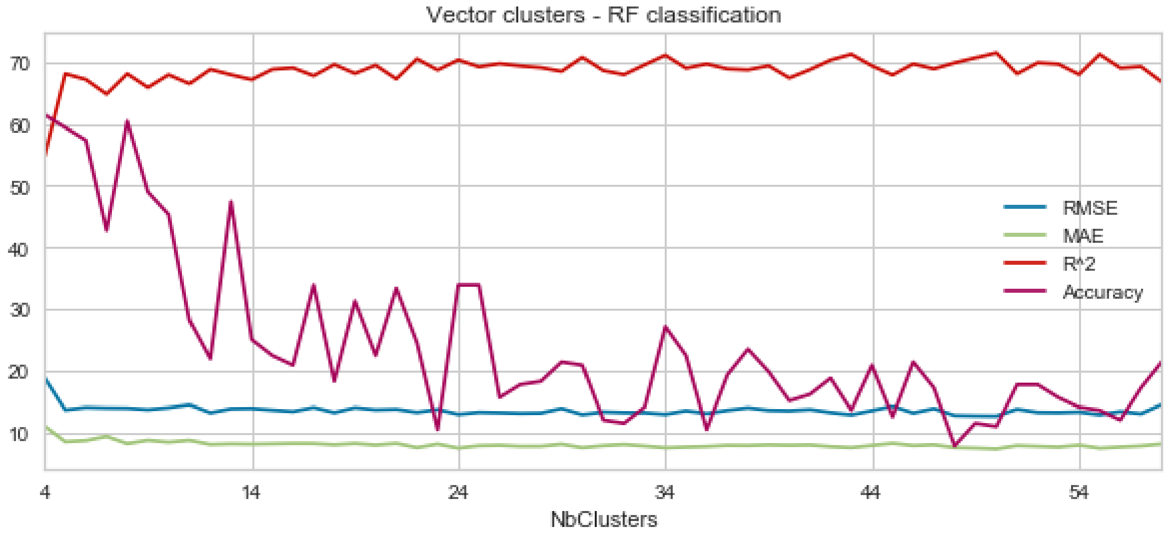
Entrée [176]:

```
dfResult[(dfResult["Algorithm"]=="RF")&(dfResult["Target Type"]=="Vector")][["RMSE","MAE","
```

executed in 229ms, finished 18:45:21 2019-09-19

Out[176]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fb11d04d30>
```



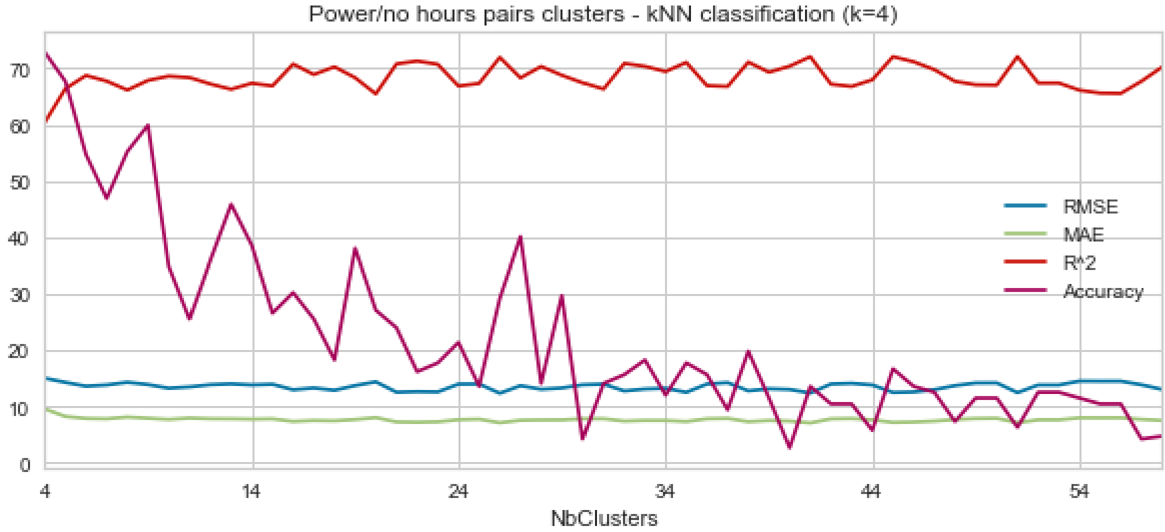
Entrée [177]:

```
dfResult[(dfResult["Algorithm"]=="kNN")&(dfResult["Target Type"]=="Power-Nohours")][["RMSE"
```

executed in 220ms, finished 18:45:22 2019-09-19

Out[177]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fb5dab34e0>
```



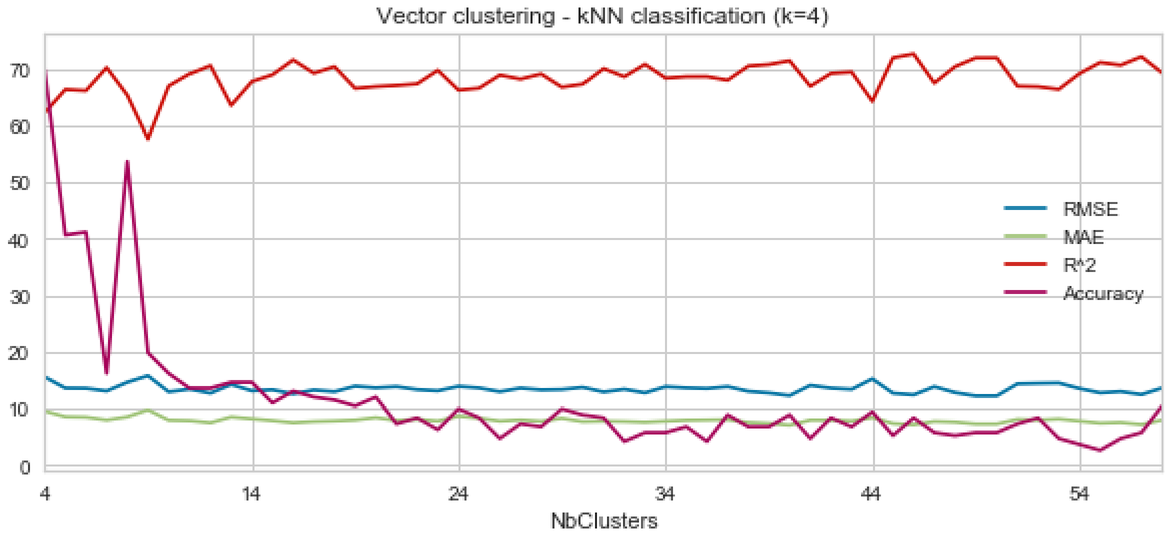
Entrée [178]:

```
dfResult[(dfResult["Algorithm"]=="kNN")&(dfResult["Target Type"]=="Vector")][["RMSE","MAE",
```

executed in 363ms, finished 18:45:24 2019-09-19

Out[178]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fb5a3ea8d0>
```



Entrée [179]:

```
dfResult["R^2"].idxmax()
```

executed in 6ms, finished 18:45:25 2019-09-19

Out[179]:

```
'22'
```

Entrée [180]:

```
dfResult["R^2"].max()
```

executed in 7ms, finished 18:45:26 2019-09-19

Out[180]:

```
72.66
```

Entrée [181]:

```
dfResult[dfResult["Target Type"]=="Vector"]["R^2"].max()
```

executed in 6ms, finished 18:45:27 2019-09-19

Out[181]:

```
72.57
```

Entrée [182]:

```
dfResult[(dfResult["R^2"]>=dfResult["R^2"].max())]
```

executed in 13ms, finished 18:45:27 2019-09-19

Out[182]:

	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Targ Ty
NbClusters										
22	RF	22	11.97	6.93	72.66	21.354167	120.19	85.22	177.59	Pow Noho

Entrée [183]:

```
dfResult["Accuracy"].max()
```

executed in 6ms, finished 18:45:32 2019-09-19

Out[183]:

```
72.91666666666666
```

Entrée [184]:

```
dfResult[(dfResult["Accuracy"]>dfResult["Accuracy"].max()-20)]
```

executed in 17ms, finished 18:45:39 2019-09-19

Out[184]:

NbClusters	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Tar Ty
4	RF	4	15.60	9.85	59.75	72.916667	120.93	85.0	178.91	Pow Noho
4	kNN	4	15.00	9.53	60.35	72.916667	120.93	85.0	178.91	Pow Noho
5	RF	5	14.26	8.04	67.06	70.833333	120.93	85.0	178.91	Pow Noho
5	kNN	5	14.25	8.19	66.38	67.708333	120.93	85.0	178.91	Pow Noho
6	kNN	6	13.57	7.83	68.74	54.687500	120.93	85.0	178.91	Pow Noho
8	kNN	8	14.26	8.08	66.15	55.208333	120.93	85.0	178.91	Pow Noho
9	RF	9	13.85	7.75	65.65	53.645833	120.93	85.0	178.91	Pow Noho
9	kNN	9	13.86	7.87	67.86	59.895833	120.93	85.0	178.91	Pow Noho
4	RF	4	18.81	11.00	54.72	61.458333	120.93	85.0	178.91	Ve
4	kNN	4	15.52	9.51	62.18	69.791667	120.93	85.0	178.91	Ve
5	RF	5	13.60	8.48	68.06	59.375000	120.93	85.0	178.91	Ve
6	RF	6	13.99	8.67	67.13	57.291667	120.93	85.0	178.91	Ve
8	RF	8	13.87	8.18	68.06	60.416667	120.93	85.0	178.91	Ve
8	kNN	8	14.58	8.51	65.28	53.645833	120.93	85.0	178.91	Ve

Entrée [185]:

```
dfResult[ (dfResult["NbClusters"]=="8")]
```

executed in 15ms, finished 18:45:41 2019-09-19

Out[185]:

	Algorithm	NbClusters	RMSE	MAE	R^2	Accuracy	Daily MAE	Daily R2	Daily RMSE	Targ Ty
NbClusters										
8	RF	8	13.59	7.77	67.61	52.604167	120.93	85.0	178.91	Pow Nohou
8	kNN	8	14.26	8.08	66.15	55.208333	120.93	85.0	178.91	Pow Nohou
8	RF	8	13.87	8.18	68.06	60.416667	120.93	85.0	178.91	Vec
8	kNN	8	14.58	8.51	65.28	53.645833	120.93	85.0	178.91	Vec

Entrée [186]:

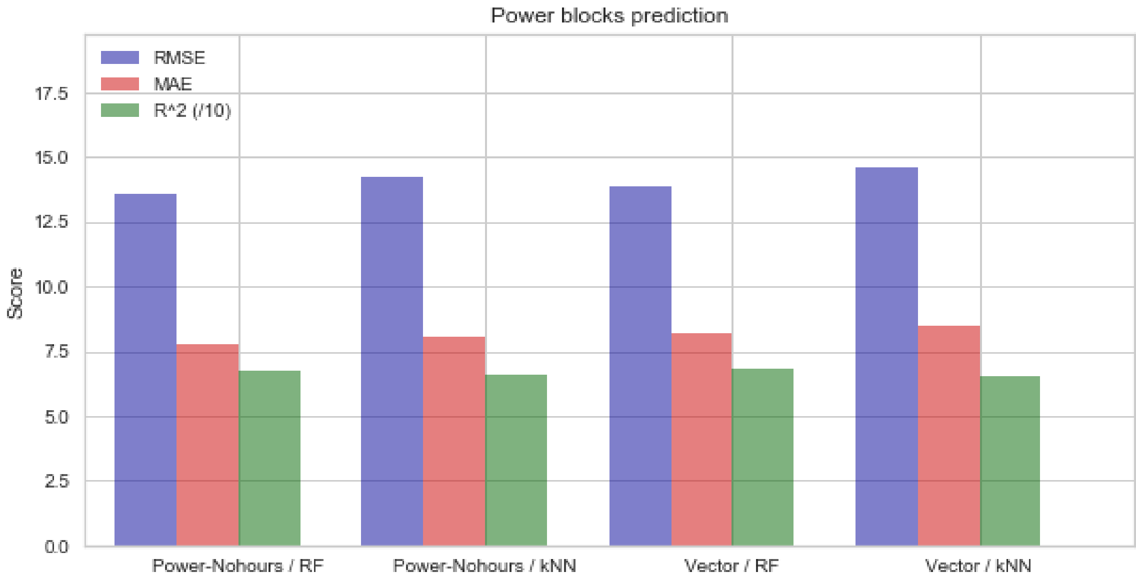
```
#Add column with algo and target concatenated
dfResult["AlgoTarget"] = dfResult["Target Type"]+" / "+dfResult["Algorithm"]
```

executed in 6ms, finished 18:45:47 2019-09-19

Entrée [187]:

```
# Setting the positions and width for the bars
df_plot = dfResult[(dfResult["NbClusters"]=="8")].copy()
df_plot["R^2"] = df_plot["R^2"] / 10
pos = list(range(len(df_plot['AlgoTarget'])))
width = 0.25
colors = ['#000099', '#CC0000', '#006600']
colNames = ['RMSE','MAE','R^2']
# Plotting the bars
fig, ax = plt.subplots(figsize=(10,5))

iNbGroup = 0
# Create a bar with pre_score data,
# in position pos,
for colName, colColor in zip(colNames,colors):
    plt.bar([p + iNbGroup*width for p in pos],
            # data in column
            df_plot[colName],
            # of width
            width,
            # with alpha 0.5
            alpha=0.5,
            # with color
            color=colColor,
            # with label the first value in first_name
            label="")
    iNbGroup += 1

#plot grid
ax.grid()

# Set the y axis label
ax.set_ylabel('Score')

# Set the chart's title
ax.set_title('Power blocks prediction')

# Set the position of the x ticks
ax.set_xticks([p + 1.5 * width for p in pos])

# Set the labels for the x ticks
ax.set_xticklabels(df_plot['AlgoTarget'])



# Setting the x-axis and y-axis limits
plt.xlim(min(pos)-width, max(pos)+width*4)
plt.ylim([0, max(df_plot['RMSE'] + df_plot['MAE'] + df_plot['R^2'])/1.5] )

# Adding the legend and showing the plot
plt.legend(['RMSE', 'MAE', 'R^2 (/10)'], loc='upper left')
plt.grid()
plt.show()
```

executed in 174ms, finished 18:45:50 2019-09-19



3.3 Clustering results detailed analysis

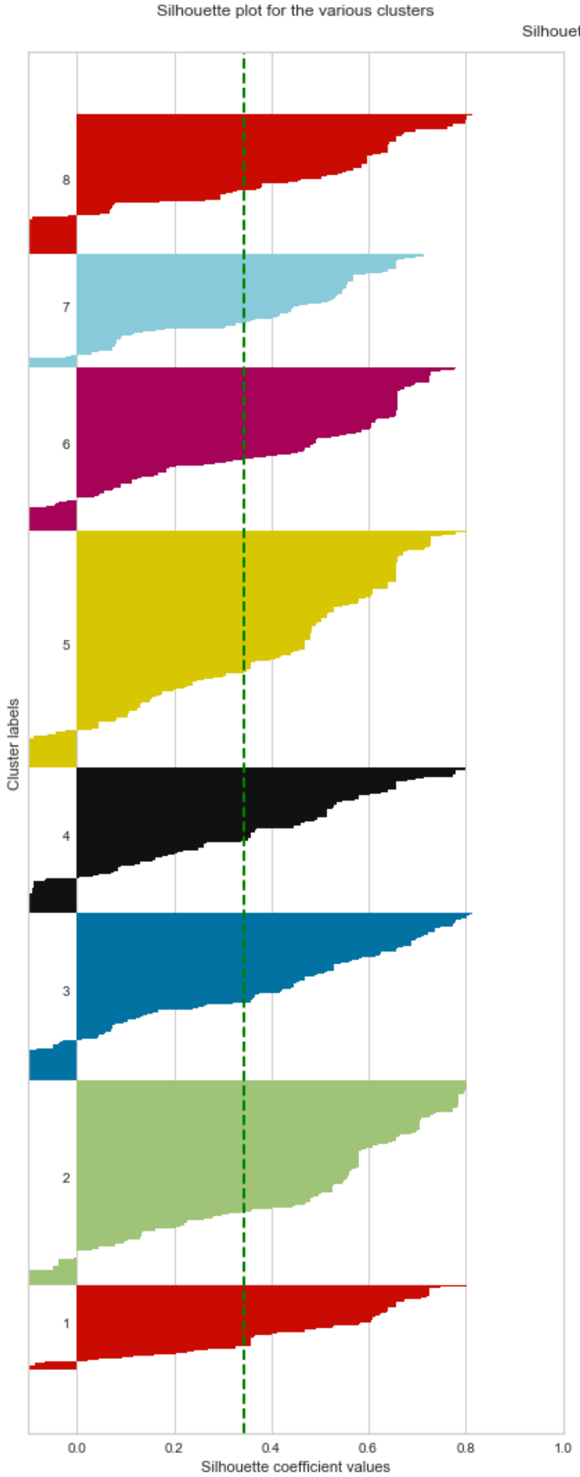
Entrée [188]:

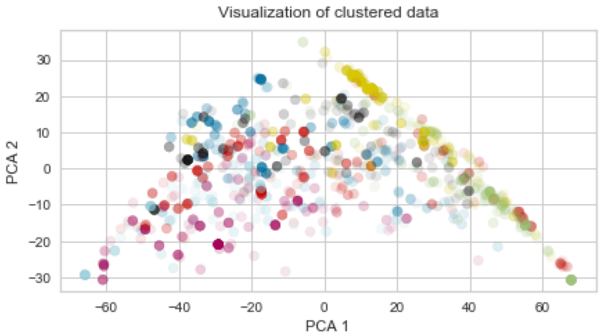
```
# Look into more details at the solution using 8 clusters

iNbClusters = 9

cols =  ['PrioH1','PrioH2','PrioH3','PrioH4', 'PrioP1','PrioP2','PrioP3','PrioP4']

# resize power values before clustering
df_resizePower= df.copy()
for i in range (1,4+1):
    df_resizePower["PrioP"+str(i)] =  df_resizePower["PrioP"+str(i)]/df_resizePower["Max pr
df_resizePower.replace(np.inf, np.nan, inplace=True)
df_resizePower.fillna(0, inplace=True)

df_cluster, centers = clusterCreation(iNbClusters, df_resizePower, cols, plotSil = True)
iNbClustRes = len(centers)
```

executed in 4.74s, finished 18:45:57 2019-09-19

```
Nb of clusters 8
Nb of centers 8, list:[114, 1609, 692, 1184, 230, 489, 1889, 622]
```

Silhouette analysis





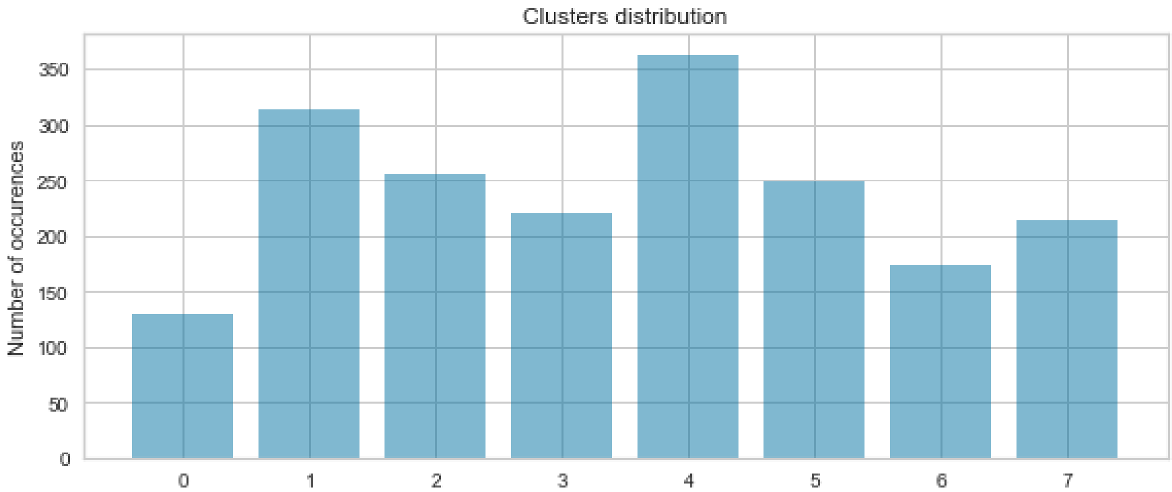
Entrée [189]:

```
# plot distribution
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 4

NbOfOccurences = [len(np.where(df_cluster["cluster_id"]==i)[0]) for i in range(iNbClustRes)
plt.bar(np.arange(iNbClustRes), NbOfOccurences, align='center', alpha=0.5 )

plt.xticks(np.arange(iNbClustRes), np.arange(iNbClustRes))
plt.ylabel('Number of occurences')
plt.title('Clusters distribution')
```

executed in 202ms, finished 18:45:57 2019-09-19

Out[189]:

```
Text(0.5, 1.0, 'Clusters distribution')
```



Entrée [190]:

```
# same about test period
df_clustest= df_cluster.iloc[1917-192:,].copy()

NbOfOccurences = [len(np.where(df_clustest["cluster_id"]==i)[0]) for i in range(iNbClustRes
plt.bar(np.arange(iNbClustRes), NbOfOccurences, align='center', alpha=0.5 )

plt.xticks(np.arange(iNbClustRes), np.arange(iNbClustRes))
plt.ylabel('Number of occurences')
plt.title('Clusters distribution')
```

executed in 185ms, finished 18:45:58 2019-09-19

Out[190]:

```
Text(0.5, 1.0, 'Clusters distribution')
```

Entrée [191]:

```
# print out details of the profiles, i.e. clusters centroids
df_TypicalProfiles = pd.DataFrame(index=np.arange(iNbClustRes), columns = cols)

for i in np.arange(iNbClustRes):
    df_TypicalProfiles.iloc[i,][cols] = df_cluster.iloc[centers[i],][cols]

# round last column for pretty printing
df_TypicalProfiles[['PrioP1', 'PrioP2', 'PrioP3', 'PrioP4']] = \
            df_TypicalProfiles[['PrioP1', 'PrioP2', 'PrioP3', 'PrioP4']]. \
            apply(pd.to_numeric, errors='coerce').round(2)

df_TypicalProfiles
```

executed in 39ms, finished 18:48:05 2019-09-19

Out[191]:

	PrioH1	PrioH2	PrioH3	PrioH4	PrioP1	PrioP2	PrioP3	PrioP4
0	12	3	1	0	0.07	0.06	0.03	0.00
1	3	1	0	0	0.31	0.07	0.00	0.00
2	2	6	8	0	0.09	0.08	0.05	0.00
3	5	4	10	0	0.10	0.05	0.03	0.00
4	8	8	0	0	0.08	0.05	0.00	0.00
5	3	5	4	4	0.08	0.07	0.06	0.05
6	4	2	2	10	0.11	0.06	0.05	0.03
7	3	5	8	8	0.07	0.06	0.05	0.01

Entrée [192]:

```
len(df_TypicalProfiles.index)
```

executed in 6ms, finished 18:48:06 2019-09-19

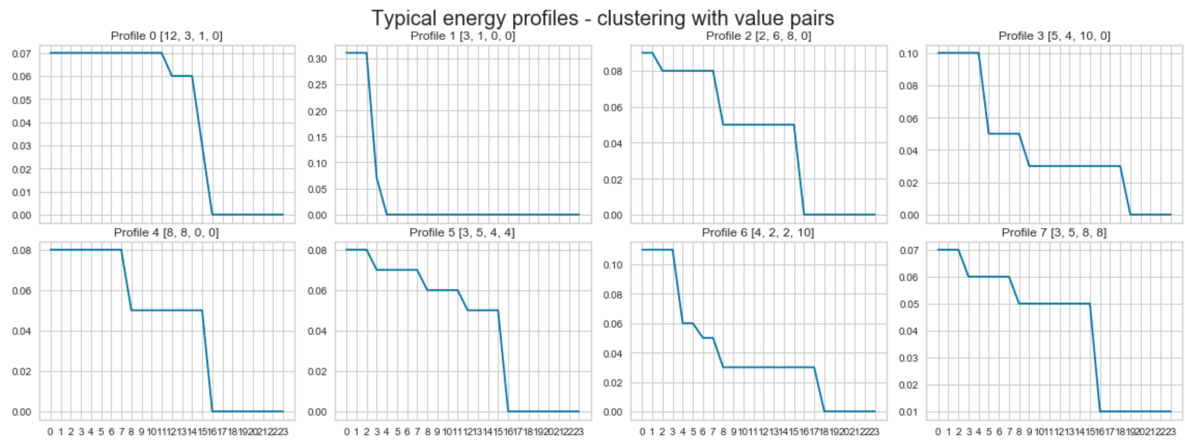
Out[192]:

8

Entrée [193]:

```
#turn blocks into 24 hours vectors
dfVect = get24hoursEnergyVector("PrioH", "PrioP", df_TypicalProfiles)

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 16
fig_size[1] = 6

nbcols = 4
nbrows = 2

fig, axes = plt.subplots(ncols=nbcols, nrows = nbrows, sharex=True)
fig.suptitle('Typical energy profiles - clustering with value pairs', size=20)
fig.tight_layout()
fig.subplots_adjust(top=0.9)

# plot all profiles in the grid
plt.figure(0)
for i in range(nbrows): # rows
    for j in range(nbcols): # columns
        if (i*nbcols+j) < len(df_TypicalProfiles.index):
            axes.flat[i*nbcols+j].plot(dfVect.loc[i*nbcols+j,])
            axes.flat[i*nbcols+j].xaxis.grid(True)
            axes.flat[i*nbcols+j].yaxis.grid(True)
            hoursLst = [int(df_TypicalProfiles.iloc[i*nbcols+j,]["PrioH"+str(k)]) for k in
            axes.flat[i*nbcols+j].title.set_text('Profile '+str(i*nbcols+j)+" "+str(hoursLs
plt.tight_layout()
plt.show()
```

executed in 1.39s, finished 18:48:08 2019-09-19




```
<Figure size 1152x432 with 0 Axes>
```

Entrée [194]:

```
df_cluster.head()
```

executed in 13ms, finished 18:48:08 2019-09-19

Out[194]:

	PrioH1	PrioH2	PrioH3	PrioH4	PrioP1	PrioP2	PrioP3	PrioP4	cluster_id
Date									
2014-04-01	4.0	4.0	8.0	0.0	0.081782	0.073138	0.047540	0.0	2
2014-04-02	4.0	4.0	8.0	0.0	0.081782	0.073138	0.047540	0.0	2
2014-04-03	4.0	4.0	8.0	0.0	0.081782	0.073138	0.047540	0.0	2
2014-04-04	4.0	4.0	8.0	0.0	0.081782	0.073138	0.047540	0.0	2
2014-04-05	4.0	4.0	8.0	0.0	0.080882	0.073529	0.047794	0.0	2

Entrée [195]:

```
# compute clustering error, i.e error if classification could reach a 100% accuracy in this
# copy the centroid values as power values on the dataframe
for prefix in ["PrioP","PrioH"]:
            for iPair in range(1,4+1):
                res = np.zeros(len(df_cluster.index))
                for myIndex in range(len(df_cluster.index)):
                    indexInOrigDF = int(df_cluster.iloc[myIndex]["cluster_id"])
                    res[myIndex] = df_cluster.iloc[indexInOrigDF][prefix+str(iPair)]
                df_cluster[prefix+str(iPair)] = res

# resize power values
for i in range (1,4+1):
    df_cluster["PrioP"+str(i)] =  df_cluster["PrioP"+str(i)]*df_resizePower["Max prod"]
```

executed in 4.46s, finished 18:48:13 2019-09-19

Entrée [196]:

```
df.shape
```

executed in 6ms, finished 18:48:13 2019-09-19

Out[196]:

```
(1917, 30)
```

Entrée [197]:

```
1917*24
```

executed in 7ms, finished 18:48:13 2019-09-19

Out[197]:

```
46008
```

Entrée [198]:

```
df_cluster.fillna(0, inplace=True)
df_cluster.isnull().values.any
```

executed in 8ms, finished 18:48:13 2019-09-19

Out[198]:

```
<function ndarray.any>
```

Entrée [199]:

```
RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2 = hourlyErrorMeasure (df_cluster, df)

df_error = pd.DataFrame(data = [[RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2]],
                                  columns = ["RMSE", "MAE", "R2", "Daily RMSE", "Daily MAE"
df_error
```

executed in 683ms, finished 18:48:14 2019-09-19

```
Hourly comparison: prediction vs. actual values
RMSE :15.92
MAE :9.49
R^2 :72.26
Daily comparison (max prod): prediction vs. actual values
RMSE :0.0
MAE :0.0
R^2 :100.0
```

Out[199]:

	RMSE	MAE	R2	Daily RMSE	Daily MAE	DailyR2
0	15.92	9.49	72.26	0.0	0.0	100.0

Entrée [200]:

```
# same analysis with vector representation ?
```

executed in 3ms, finished 18:48:17 2019-09-19

3.3.1 Random forest classification variables importance scores

Entrée [201]:

```
# add cluster allocation to our dataset
df["cluster_id"] = df_cluster["cluster_id"]
```

executed in 4ms, finished 18:48:18 2019-09-19

Entrée [202]:

```
df.head()
```

executed in 22ms, finished 18:48:19 2019-09-19

Out[202]:

	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 × 30 columns

Entrée [203]:

```
def selectKImportance(model, X, k=5):
     return X[:,model.feature_importances_.argsort()[::-1][:k]]
```

executed in 5ms, finished 18:48:20 2019-09-19

Entrée [204]:

```
# train RF model to extract its variables importance

target_feature = 'cluster_id'
regressors = ['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 [%]', 'Availability plant 3 [%]', \
            'Availability plant 4 [%]','Weekend','Max prod']

np.random.seed(42) # set seed for reproducibility

# split train and test
xTrain, xTest, yTrain, yTest, yBaseline =  GetDataSplit(df, regressors, target_feature,[],0

# Forecast using Random Forest (1000 trees)
RFmodel = RandomForestClassifier(n_estimators=1000)

# Fit model with features and labels.
classifyModel = RFmodel.fit(xTrain, yTrain)

importances = classifyModel.feature_importances_
std = np.std([tree.feature_importances_ for tree in classifyModel.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(xTrain.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

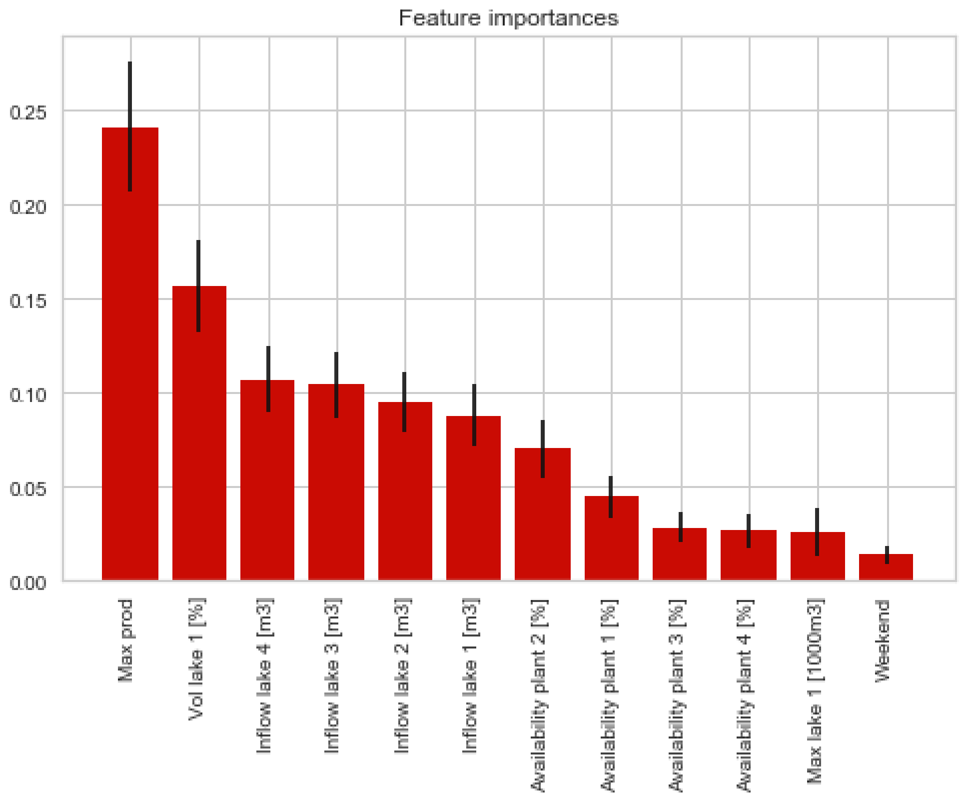
executed in 2.56s, finished 18:48:24 2019-09-19

```
Feature ranking:
1. feature 11 (0.240781)
2. feature 4 (0.156389)
3. feature 3 (0.107092)
4. feature 2 (0.103944)
5. feature 1 (0.095017)
6. feature 0 (0.087912)
7. feature 7 (0.070100)
8. feature 6 (0.044350)
9. feature 8 (0.028235)
10. feature 9 (0.026494)
11. feature 5 (0.025959)
12. feature 10 (0.013727)
```

Entrée [205]:

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 8
fig_size[1] = 5

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(xTrain.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(xTrain.shape[1]), xTrain.columns[indices], rotation='vertical')
plt.xlim([-1, xTrain.shape[1]])
plt.show()
```

executed in 188ms, finished 18:48:24 2019-09-19



Entrée [206]:

```
# perform prediction using the model
pred = RFmodel.predict(xTest)

report = classification_report(yTest, pred, output_dict=False)
print(report)
```

executed in 89ms, finished 18:48:24 2019-09-19

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.97      0.92      0.94       107
           2       0.31      0.93      0.46        28
           3       0.00      0.00      0.00         7
           4       0.00      0.00      0.00         7
           5       0.00      0.00      0.00         2
           6       0.00      0.00      0.00        11
           7       0.00      0.00      0.00        17

    accuracy                           0.65       192
   macro avg       0.16      0.23      0.18       192
weighted avg       0.59      0.65      0.59       192


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.p
y:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Entrée [207]:

```
iNbClustRes
```

executed in 7ms, finished 18:48:24 2019-09-19

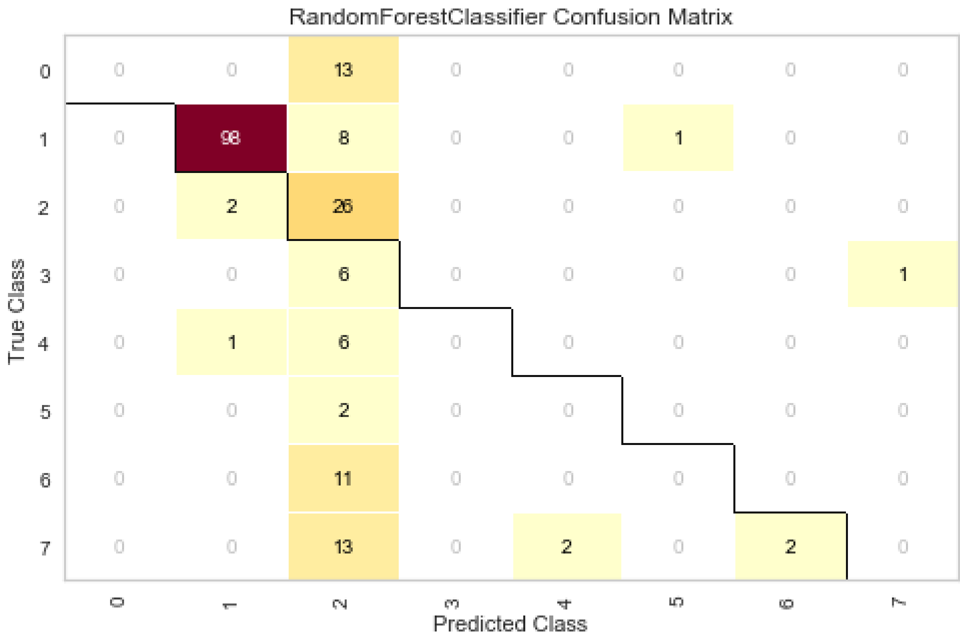
Out[207]:

```
8
```

Entrée [208]:

```
from yellowbrick.classifier import ConfusionMatrix

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 8
fig_size[1] = 5

# The ConfusionMatrix visualizer taxes a model
cm = ConfusionMatrix(RFmodel)

# Fit fits the passed model. This is unnecessary if you pass the visualizer a pre-fitted mo
cm.fit(xTrain, yTrain)

# To create the ConfusionMatrix, we need some test data. Score runs predict() on the data
# and then creates the confusion_matrix from scikit-learn.
cm.score(xTest, yTest)

# How did we do?
cm.poof()
```

executed in 591ms, finished 18:48:24 2019-09-19



Out[208]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fb11b65518>
```

3.3.2 Classification results detailed analysis - Vector representation

Entrée [209]:

```
# Look into more details at the solution using 8 clusters using vector representation
df.head()
```

executed in 24ms, finished 18:48:25 2019-09-19

Out[209]:

	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	.

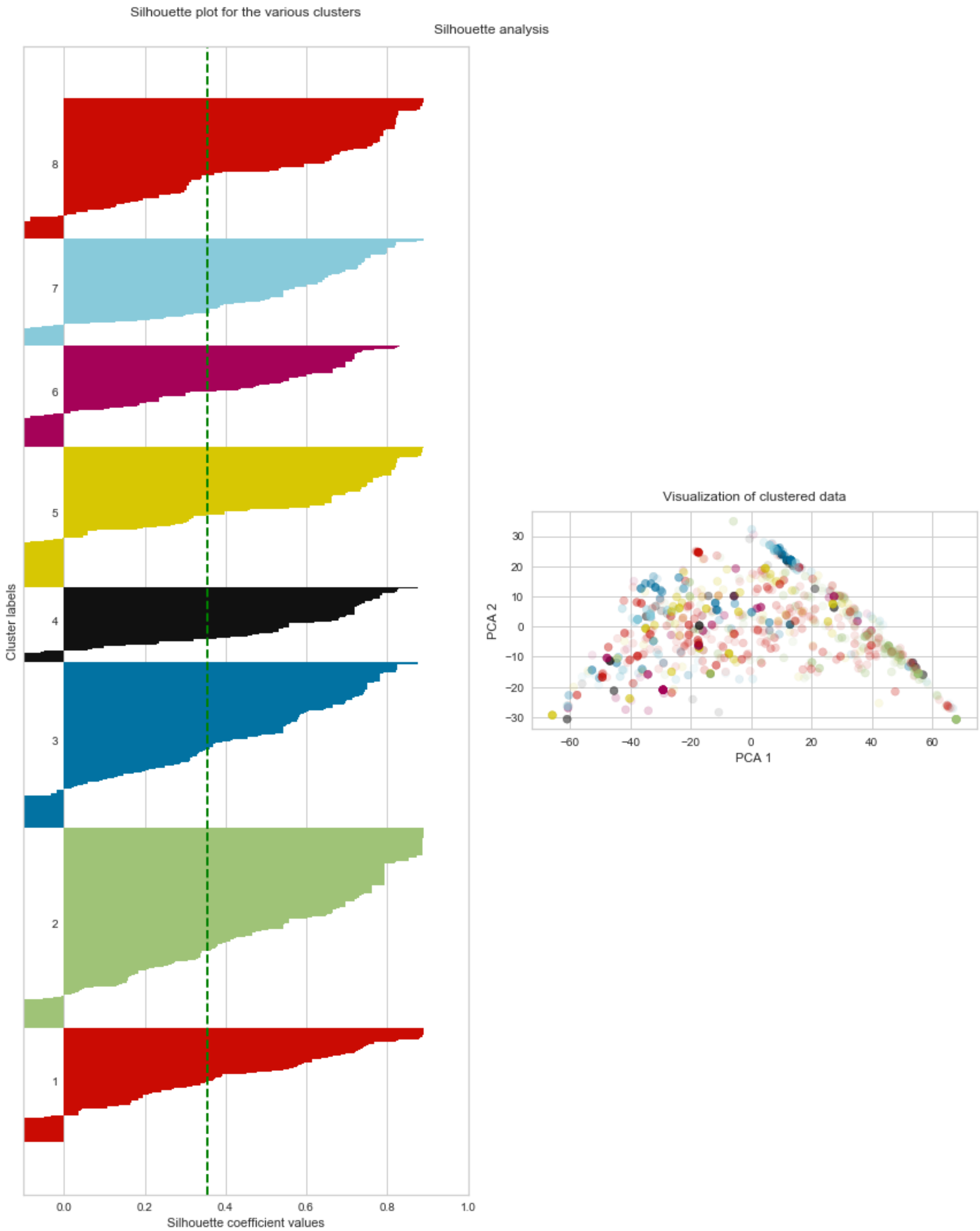
5 rows × 30 columns

Entrée [210]:

```
iNbClusters = 9

df["cluster_id"] = 0

cols =  ['P'+str(i) for i in range(24)]

# resize power values before clustering
df_resizePower= df.copy()
for i in range (1,4+1):
    df_resizePower["PrioP"+str(i)] =  df_resizePower["PrioP"+str(i)]/df_resizePower["Max pr
df_resizePower.replace(np.inf, np.nan, inplace=True)
df_resizePower.fillna(0, inplace=True)

# create the 24 hours vector columns in the dataframe, using the blocks definition
dfVector = get24hoursEnergyVector("PrioH", "PrioP", df_resizePower, "P")
for i in range(24):
    df_resizePower["P"+str(i)] =  dfVector["P"+str(i)]

df_cluster, centers = clusterCreation(iNbClusters, df_resizePower, cols, targetType = "Vect
iNbClustRes = len(centers)
```

executed in 7.71s, finished 18:48:34 2019-09-19

```
Nb of clusters 8
Nb of centers 8, list:[1885, 1590, 639, 290, 1065, 923, 1355, 710]
```


Silhouette analysis
Silhouette plot for the various clusters
Cluster labels
8
7
6
5
4
3
2
1
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette coefficient values
Visualization of clustered data
PCA 2
30
20
10
0
−10
−20
−30
−60
−40
−20
0
20
40
60
PCA 1

Entrée [211]:

```
# plot distribution
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 4

NbOfOccurences = [len(np.where(df_cluster["cluster_id"]==i)[0]) for i in range(iNbClustRes)
plt.bar(np.arange(iNbClustRes), NbOfOccurences, align='center', color='r', alpha=0.5 )

plt.xticks(np.arange(iNbClustRes), np.arange(iNbClustRes))
plt.ylabel('Number of occurences')
plt.title('Clusters distribution')
```

executed in 182ms, finished 18:48:34 2019-09-19

Out[211]:

```
Text(0.5, 1.0, 'Clusters distribution')
```

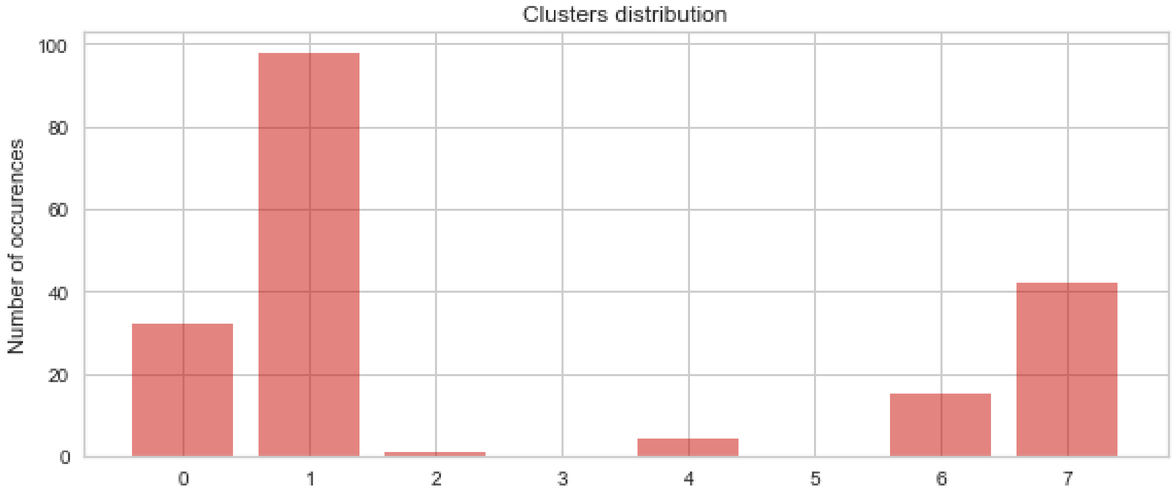
Entrée [212]:

```
# same about test period
df_clustest= df_cluster.iloc[1917-192:,].copy()

NbOfOccurences = [len(np.where(df_clustest["cluster_id"]==i)[0]) for i in range(iNbClustRes
plt.bar(np.arange(iNbClustRes), NbOfOccurences, align='center', color='r', alpha=0.5 )

plt.xticks(np.arange(iNbClustRes), np.arange(iNbClustRes))
plt.ylabel('Number of occurences')
plt.title('Clusters distribution')
```

executed in 174ms, finished 18:48:34 2019-09-19

Out[212]:

```
Text(0.5, 1.0, 'Clusters distribution')
```



Entrée [213]:

```
# print out details of the profiles, i.e. clusters centroids
df_TypicalProfiles = pd.DataFrame(index=np.arange(iNbClustRes), columns = cols)

for i in np.arange(iNbClustRes):
    df_TypicalProfiles.iloc[i,][cols] = df_cluster.iloc[centers[i],][cols]

# round last column for pretty printing
df_TypicalProfiles = df_TypicalProfiles.apply(pd.to_numeric, errors='coerce').round(2)

df_TypicalProfiles
```

executed in 52ms, finished 18:48:34 2019-09-19

Out[213]:

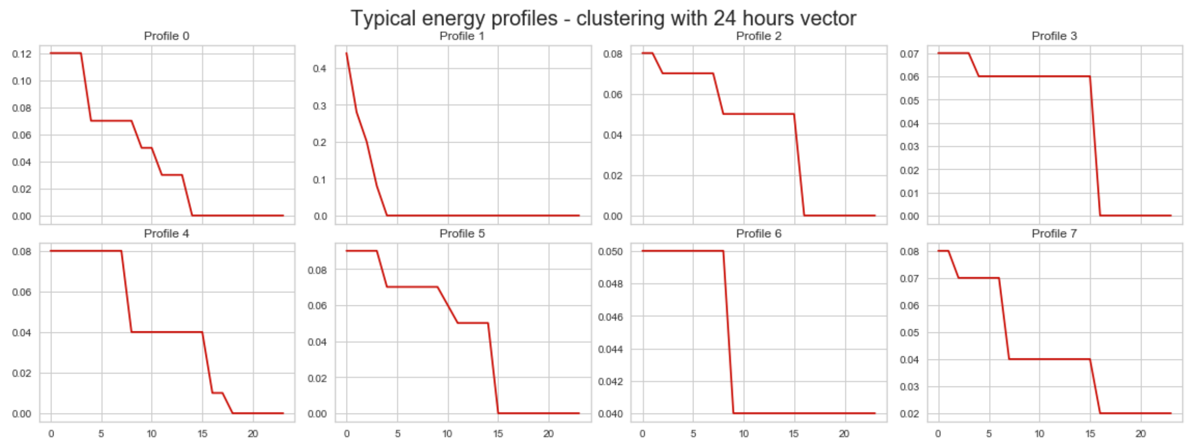
	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	...	P14	P15	P16	P17	P18	P
0	0.12	0.12	0.12	0.12	0.07	0.07	0.07	0.07	0.07	0.05	...	0.00	0.00	0.00	0.00	0.00	0.
1	0.44	0.28	0.20	0.08	0.00	0.00	0.00	0.00	0.00	0.00	...	0.00	0.00	0.00	0.00	0.00	0.
2	0.08	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.05	0.05	...	0.05	0.05	0.00	0.00	0.00	0.
3	0.07	0.07	0.07	0.07	0.06	0.06	0.06	0.06	0.06	0.06	...	0.06	0.06	0.00	0.00	0.00	0.
4	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.04	0.04	...	0.04	0.04	0.01	0.01	0.00	0.
5	0.09	0.09	0.09	0.09	0.07	0.07	0.07	0.07	0.07	0.07	...	0.05	0.00	0.00	0.00	0.00	0.
6	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.04	...	0.04	0.04	0.04	0.04	0.04	0.
7	0.08	0.08	0.07	0.07	0.07	0.07	0.07	0.04	0.04	0.04	...	0.04	0.04	0.02	0.02	0.02	0.

8 rows × 24 columns

Entrée [214]:

```
#turn blocks into 24 hours vectors
dfVect = df_TypicalProfiles
dfVect.columns = [i for i in range(24)]

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 16
fig_size[1] = 6

nbcols = 4
nbrows = 2

fig, axes = plt.subplots(ncols=nbcols, nrows = nbrows, sharex=True)
fig.suptitle('Typical energy profiles - clustering with 24 hours vector', size=20)
fig.tight_layout()
fig.subplots_adjust(top=0.9)

# plot all profiles in the grid
plt.figure(0)
for i in range(nbrows): # rows
    for j in range(nbcols): # columns
        if (i*nbcols+j) < len(df_TypicalProfiles.index):
            axes.flat[i*nbcols+j].plot(dfVect.loc[i*nbcols+j,], c='r')
            axes.flat[i*nbcols+j].xaxis.grid(True)
            axes.flat[i*nbcols+j].yaxis.grid(True)
            axes.flat[i*nbcols+j].title.set_text('Profile '+str(i*nbcols+j))
plt.tight_layout()
plt.show()
```

executed in 1.51s, finished 18:48:36 2019-09-19




```
<Figure size 1152x432 with 0 Axes>
```

Entrée [215]:

```
# compute clustering error, i.e error if classification could reach a 100% accuracy in this
# copy the centroid values as power values on the dataframe
for prefix in ["PrioP","PrioH"]:
            for iPair in range(1,4+1):
                res = np.zeros(len(df_cluster.index))
                for myIndex in range(len(df_cluster.index)):
                    indexInOrigDF = int(df_cluster.iloc[myIndex]["cluster_id"])
                    res[myIndex] = df_resizePower.iloc[centers[indexInOrigDF]][prefix+str(i
                df_cluster[prefix+str(iPair)] = res

# resize power values
for i in range (1,4+1):
    df_cluster["PrioP"+str(i)] =   df_cluster["PrioP"+str(i)]*df["Max prod"]

df_cluster.fillna(0, inplace=True)
```

executed in 5.77s, finished 18:48:41 2019-09-19

Entrée [216]:

```
df_cluster.head()
```

executed in 25ms, finished 18:48:41 2019-09-19

Out[216]:

	P0	P1	P2	P3	P4	P5	P6	P7	P
Date									
2014-04-01	0.081782	0.081782	0.081782	0.081782	0.073138	0.073138	0.073138	0.073138	0.04754
2014-04-02	0.081782	0.081782	0.081782	0.081782	0.073138	0.073138	0.073138	0.073138	0.04754
2014-04-03	0.081782	0.081782	0.081782	0.081782	0.073138	0.073138	0.073138	0.073138	0.04754
2014-04-04	0.081782	0.081782	0.081782	0.081782	0.073138	0.073138	0.073138	0.073138	0.04754
2014-04-05	0.080882	0.080882	0.080882	0.080882	0.073529	0.073529	0.073529	0.073529	0.04779

5 rows × 33 columns

Entrée [217]:

```
RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2 = hourlyErrorMeasure (df_cluster, df)

df_error = pd.DataFrame(data = [[RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2]],
                                  columns = ["RMSE", "MAE", "R2", "Daily RMSE", "Daily MAE"
df_error
```

executed in 455ms, finished 18:48:42 2019-09-19

```
Hourly comparison: prediction vs. actual values
RMSE :8.84
MAE :4.88
R^2 :89.55
Daily comparison (max prod): prediction vs. actual values
RMSE :0.0
MAE :0.0
R^2 :100.0
```

Out[217]:

	RMSE	MAE	R2	Daily RMSE	Daily MAE	DailyR2
0	8.84	4.88	89.55	0.0	0.0	100.0

Entrée [218]:

```
# classification problem with RF
# train RF model to extract its variables importance

target_feature = 'cluster_id'
regressors = ['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 [%]', 'Availability plant 3 [%]', \
            'Availability plant 4 [%]','Weekend','Max prod']

df["cluster_id"] = df_cluster["cluster_id"]

np.random.seed(42) # set seed for reproducibility

# split train and test
xTrain, xTest, yTrain, yTest, yBaseline =  GetDataSplit(df, regressors, target_feature,[],0
xTest["Max prod"] = df_MLPprediction["Prediction"]

# Forecast using Random Forest (1000 trees)
RFmodel = RandomForestClassifier(n_estimators=1000)

# Fit model with features and labels.
classifyModel = RFmodel.fit(xTrain, yTrain)

importances = classifyModel.feature_importances_
std = np.std([tree.feature_importances_ for tree in classifyModel.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(xTrain.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

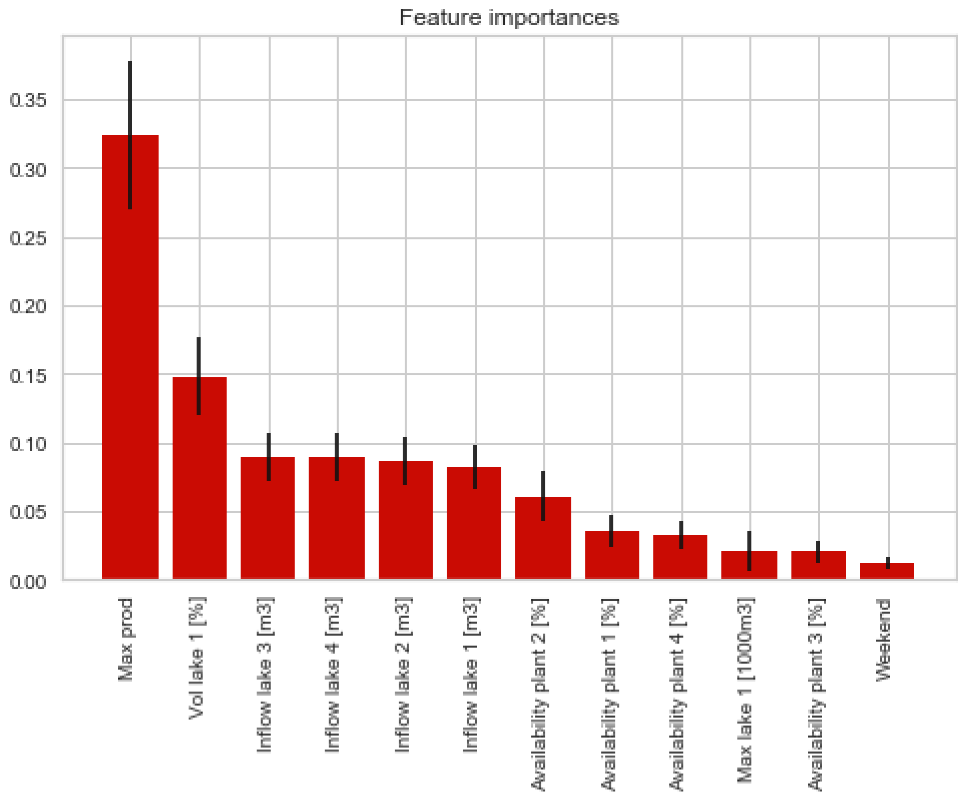
executed in 2.92s, finished 18:48:45 2019-09-19

```
Feature ranking:
1. feature 11 (0.323884)
2. feature 4 (0.147991)
3. feature 2 (0.089369)
4. feature 3 (0.089342)
5. feature 1 (0.086067)
6. feature 0 (0.082316)
7. feature 7 (0.060660)
8. feature 6 (0.035298)
9. feature 9 (0.032499)
10. feature 5 (0.020621)
11. feature 8 (0.020273)
12. feature 10 (0.011680)
```

Entrée [219]:

```
# Plot the scores as bars

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 8
fig_size[1] = 5

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(xTrain.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(xTrain.shape[1]), xTrain.columns[indices], rotation='vertical')
plt.xlim([-1, xTrain.shape[1]])
plt.show()
```

executed in 206ms, finished 18:48:45 2019-09-19



Entrée [220]:

```
# perform prediction using the model
pred = RFmodel.predict(xTest)

report = classification_report(yTest, pred, output_dict=False)
print(report)
```

executed in 96ms, finished 18:48:45 2019-09-19

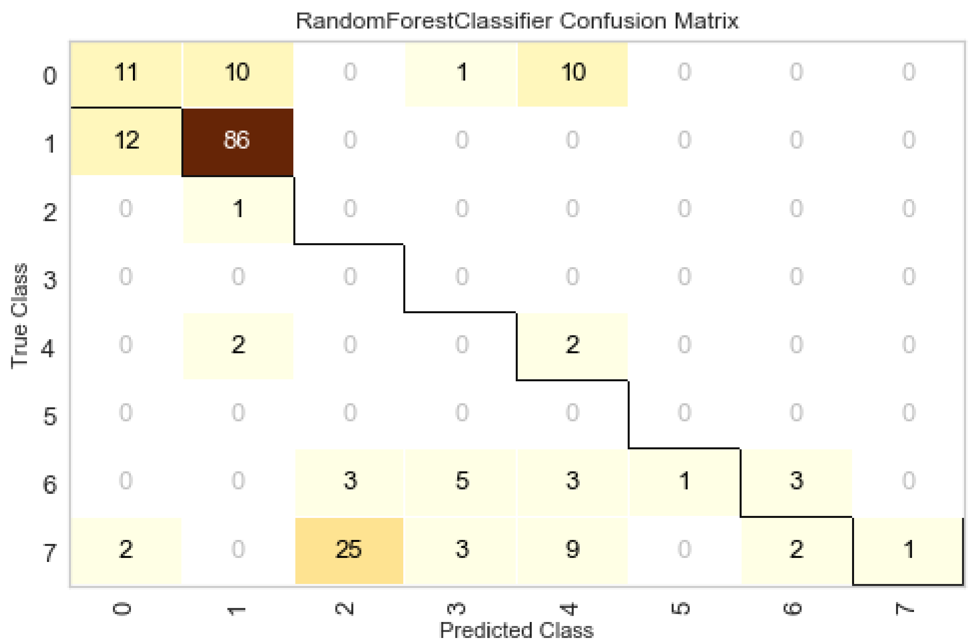
```
              precision    recall  f1-score   support

           0       0.44      0.34      0.39        32
           1       0.87      0.88      0.87        98
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         0
           4       0.08      0.50      0.14         4
           5       0.00      0.00      0.00         0
           6       0.60      0.20      0.30        15
           7       1.00      0.02      0.05        42

    accuracy                           0.54       192
   macro avg       0.37      0.24      0.22       192
weighted avg       0.78      0.54      0.55       192


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.p
y:1439: UndefinedMetricWarning: Recall and F-score are ill-defined and being
set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

Entrée [221]:

```
from yellowbrick.classifier import ConfusionMatrix

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 8
fig_size[1] = 5

# The ConfusionMatrix visualizer taxes a model
cm = ConfusionMatrix(RFmodel,fontsize=13, cmap='YlOrBr')

# Fit fits the passed model. This is unnecessary if you pass the visualizer a pre-fitted mo
#cm.fit(xTrain, yTrain)

# To create the ConfusionMatrix, we need some test data. Score runs predict() on the data
# and then creates the confusion_matrix from scikit-learn.
cm.score(xTest, yTest)

# How did we do?
cm.poof()
```

executed in 538ms, finished 18:48:46 2019-09-19

```
C:\ProgramData\Anaconda3\lib\site-packages\yellowbrick\classifier\base.py:23
2: YellowbrickWarning: could not determine class_counts_ from previously fit
ted classifier
  YellowbrickWarning,
```



Out[221]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fb1374a278>
```

Entrée [222]:

```
# compute hourly error on the forecast
dfPref = pd.DataFrame(index=xTest.index, columns=cols)
dfPref["cluster_id"] = pred

# populate the power values
for prefix in ["PrioP","PrioH"]:
            for iPair in range(1,4+1):
                res = np.zeros(len(dfPref.index))
                for myIndex in range(len(dfPref.index)):
                    indexInOrigDF = int(dfPref.iloc[myIndex]["cluster_id"])
                    res[myIndex] = df_resizePower.iloc[centers[indexInOrigDF]][prefix+str(i
                dfPref[prefix+str(iPair)] = res

# resize power values with the predicted maximum energy (imported data)
for i in range (1,4+1):
    dfPref["PrioP"+str(i)] =  dfPref["PrioP"+str(i)]*df_MLPprediction["Prediction"]

dfPref.fillna(0, inplace=True)
```

executed in 601ms, finished 18:48:46 2019-09-19

Entrée [223]:

```
RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2 = hourlyErrorMeasure (dfPref, df.iloc[1917-192:

df_error = pd.DataFrame(data = [[RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2]],
                                  columns = ["RMSE", "MAE", "R2", "Daily RMSE", "Daily MAE"
df_error
```

executed in 73ms, finished 18:48:46 2019-09-19

```
Hourly comparison: prediction vs. actual values
RMSE :13.7
MAE :7.78
R^2 :67.7
Daily comparison (max prod): prediction vs. actual values
RMSE :178.91
MAE :120.93
R^2 :85.0
```

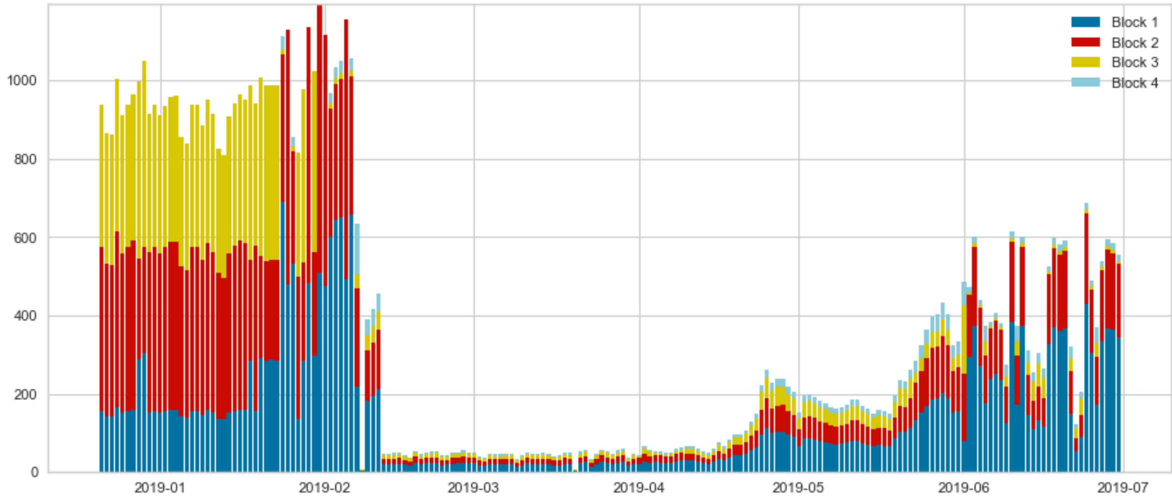
Out[223]:

	RMSE	MAE	R2	Daily RMSE	Daily MAE	DailyR2
0	13.7	7.78	67.7	178.91	120.93	85.0

Entrée [224]:

```
# plot the predicted energy by block
plotEnergyBlocks(dfPref, "Block", "RF_8Clust_Vector")
# df.iloc[1917-192:,]
```

executed in 1.84s, finished 18:48:48 2019-09-19



Entrée [225]:

```
# as a final experiment, we use the forecasted profile in its original magnitude
# i.e. not resized towards the predicted maximum energy
# unfortunately it doesn't bring any improvement

# populate the power values
for prefix in ["PrioP","PrioH"]:
            for iPair in range(1,4+1):
                res = np.zeros(len(dfPref.index))
                for myIndex in range(len(dfPref.index)):
                    indexInOrigDF = int(dfPref.iloc[myIndex]["cluster_id"])
                    res[myIndex] = df.iloc[centers[indexInOrigDF]][prefix+str(iPair)]
                dfPref[prefix+str(iPair)] = res

dfPref.fillna(0, inplace=True)

RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2 = hourlyErrorMeasure (dfPref, df.iloc[1917-192:

df_error = pd.DataFrame(data = [[RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2]],
                                  columns = ["RMSE", "MAE", "R2", "Daily RMSE", "Daily MAE"
df_error
```

executed in 668ms, finished 18:48:49 2019-09-19

```
Hourly comparison: prediction vs. actual values
RMSE :14.3
MAE :8.43
R^2 :65.23
Daily comparison (max prod): prediction vs. actual values
RMSE :237.53
MAE :168.93
R^2 :73.57
```

Out[225]:

	RMSE	MAE	R2	Daily RMSE	Daily MAE	DailyR2
0	14.3	8.43	65.23	237.53	168.93	73.57

3.4 Classification using MLP

Here we try to beat the classification accuracy at C=8 clusters,using vector representation

Entrée [692]:

```
df.head()
```

executed in 24ms, finished 12:31:26 2019-09-14

Out[692]:

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 [%]	.
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	.

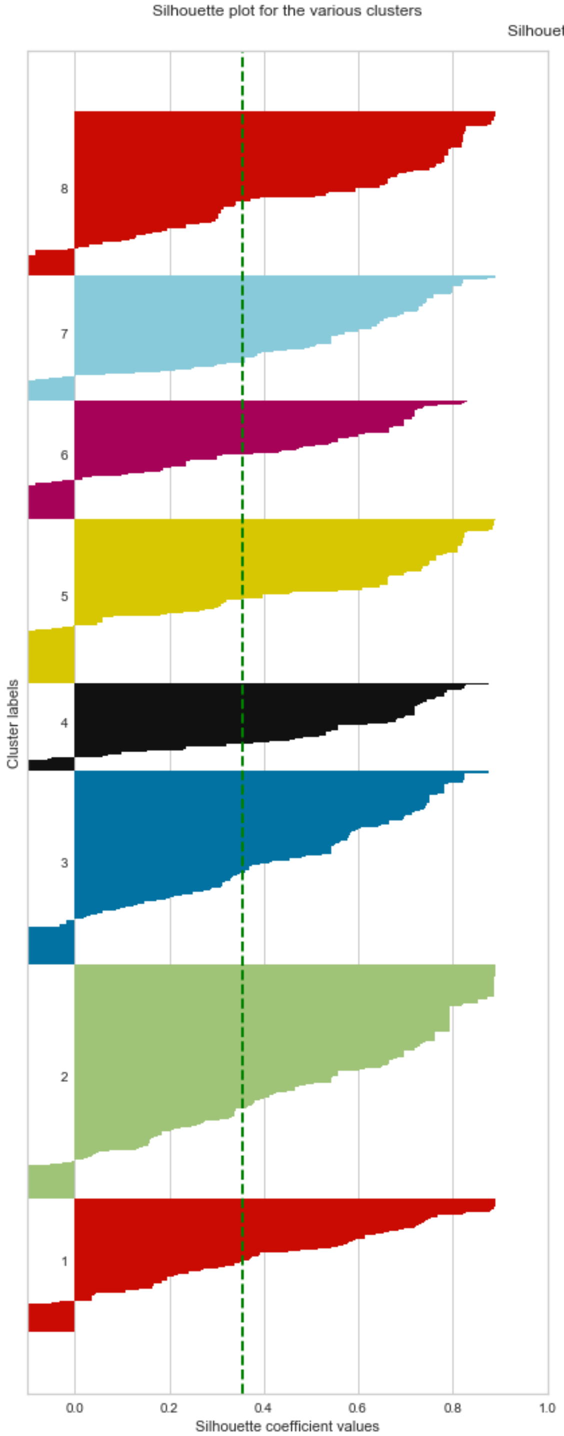
5 rows × 29 columns

Entrée [830]:

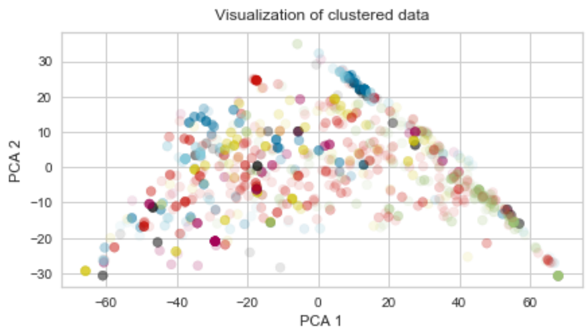
```
# Perform clustering with C=8

iNbClusters = 9

df["cluster_id"] = 0

cols =  ['P'+str(i) for i in range(24)]

# resize power values before clustering
df_resizePower= df.copy()
for i in range (1,4+1):
    df_resizePower["PrioP"+str(i)] =  df_resizePower["PrioP"+str(i)]/df_resizePower["Max pr
df_resizePower.replace(np.inf, np.nan, inplace=True)
df_resizePower.fillna(0, inplace=True)

# create the 24 hours vector columns in the dataframe, using the blocks definition
dfVector = get24hoursEnergyVector("PrioH", "PrioP", df_resizePower, "P")
for i in range(24):
    df_resizePower["P"+str(i)] =  dfVector["P"+str(i)]

df_cluster, centers = clusterCreation(iNbClusters, df_resizePower, cols, targetType = "Vect
iNbClustRes = len(centers)
```

executed in 7.91s, finished 17:24:20 2019-09-14

```
Nb of clusters 8
Nb of centers 8, list:[1885, 1590, 639, 290, 1065, 923, 1355, 710]
```



Silhouette analysis



Entrée [831]:

```
np.max(yTrain)
```

executed in 8ms, finished 17:24:28 2019-09-14

Out[831]:

7

Entrée [840]:

```
# define regression target and explanatory variables
target_feature = 'cluster_id'
regressors = ['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 [%]', 'Availability plant 3 [%]', \
           'Availability plant 4 [%]','Weekend','Max prod']

df["cluster_id"] = df_cluster["cluster_id"]

# implement mutli-output random forest
fullRegressors = regressors

xTrain, xTest, yTrain, yTest, yBaseline =  GetDataSplit(df, fullRegressors, target_feature,
# for the prediction part, replace the max energy with the forecasted value
xTest["Max prod"] = df_MLPprediction["Prediction"]

xTest.head()
```

executed in 325ms, finished 17:27:42 2019-09-14

Out[840]:

Date	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 [%]	Availability plant 3 [%]	Av pl
2018-12-21	51.0	22.9	68.7	41.4	0.34519	30000.0	1.0	1.0	1.0	
2018-12-22	55.0	28.3	63.5	44.6	0.34262	30000.0	1.0	1.0	1.0	
2018-12-23	43.0	30.3	58.4	42.2	0.34089	30000.0	1.0	1.0	1.0	
2018-12-24	96.0	23.3	172.7	194.6	0.34020	30000.0	1.0	1.0	1.0	
2018-12-25	69.0	32.0	78.1	95.2	0.34089	30000.0	1.0	1.0	1.0	

Entrée [841]:

```
xTrain.head()
```

executed in 18ms, finished 17:27:44 2019-09-14

Out[841]:

Date	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 [%]	Availability plant 3 [%]	Av pl
2014-04-01	31.0	4.0	129.0	107.0	0.16467	30000.0	1.0	1.0	1.000000	
2014-04-02	0.0	-14.0	148.0	116.0	0.15557	30000.0	1.0	1.0	1.000000	
2014-04-03	10.0	6.0	132.0	118.0	0.14765	30000.0	1.0	1.0	0.291667	
2014-04-04	19.0	6.0	150.0	118.0	0.13716	30000.0	1.0	1.0	0.250000	
2014-04-05	41.0	15.0	148.0	124.0	0.13091	30000.0	1.0	1.0	1.000000	

Entrée [842]:

```
xTest.shape[1]
```

executed in 9ms, finished 17:27:45 2019-09-14

Out[842]:

12

Entrée [843]:

```
# scale the input data
from keras.utils import np_utils
from sklearn.preprocessing import MinMaxScaler

#standardizing the input feature
minMaxScaler  = MinMaxScaler()
xTrain_minmax = minMaxScaler.fit_transform(xTrain)
xTest_minmax  = minMaxScaler.transform(xTest)

xTrain_minmax = pd.DataFrame(xTrain_minmax)
xTest_minmax  = pd.DataFrame(xTest_minmax)

xTest_minmax.tail()
```

executed in 26ms, finished 17:27:50 2019-09-14

Out[843]:

	0	1	2	3	4	5	6	7	8	9	10	11
187	0.400288	0.392445	1.000101	0.776782	0.657978	1.0	1.0	1.0	0.0	0.0	0.0	0.236045
188	0.560455	0.389297	1.083585	0.704936	0.679293	1.0	1.0	1.0	0.0	0.0	0.0	0.312101
189	0.575144	0.588667	0.858610	0.748903	0.701111	1.0	1.0	1.0	0.0	0.0	0.0	0.342295
190	0.525518	0.592865	0.822961	0.642687	0.721038	1.0	1.0	1.0	0.0	0.0	1.0	0.324422
191	0.497264	0.395593	1.050050	0.531718	0.739710	1.0	1.0	1.0	0.0	0.0	1.0	0.307139

Entrée [844]:

```
xTrain_minmax.tail()
```

executed in 16ms, finished 17:27:55 2019-09-14

Out[844]:

	0	1	2	3	4	5	6	7	8	9	10	
1720	0.073157	0.525813	0.310574	0.251371	0.402665	1.0	1.0	0.978836	1.0	1.0	1.0	0.822
1721	0.069700	0.530535	0.316012	0.241408	0.390645	1.0	1.0	1.000000	1.0	1.0	0.0	0.777
1722	0.076901	0.523924	0.318127	0.247258	0.380045	1.0	1.0	1.000000	1.0	1.0	0.0	0.769
1723	0.074021	0.528961	0.320544	0.243236	0.368899	1.0	1.0	1.000000	1.0	1.0	0.0	0.769
1724	0.071141	0.528332	0.308258	0.239488	0.357918	1.0	1.0	1.000000	1.0	1.0	0.0	0.769

Entrée [860]:

```
def plotHystory(history):
    fig_size = plt.rcParams["figure.figsize"]
    fig_size[0] = 8
    fig_size[1] = 4

    # summarize history for loss

    plt.plot(history.history['loss'])
    plt.plot(history.history['acc'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'])
    plt.show()
```

executed in 9ms, finished 17:40:20 2019-09-14

Entrée [870]:

```
from keras import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adadelta

inputDimX = xTest.shape[1]
outputDimY = np.max(yTrain)+1

dfRes = pd.DataFrame(columns=["N1","N2","Accuracy"])

for neurons1 in range (5,10):
    for neurons2 in range(3,7):

        np.random.seed(42) # set seed for reproducibility

        classifier = Sequential()
        #First Hidden Layer
        classifier.add(Dense(neurons1, activation='relu', kernel_initializer='random_normal
        #Second  Hidden Layer
        #classifier.add(Dropout(0.2))
        classifier.add(Dense(neurons2, activation='relu', kernel_initializer='random_normal
        #Output Layer
        classifier.add(Dense(outputDimY, activation='softmax'))

        #Compiling the neural network
        classifier.compile(optimizer ="adam",loss='sparse_categorical_crossentropy',
                           metrics =['accuracy'])
        #classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac

        #Fitting the data to the training dataset
        history = classifier.fit(xTrain_minmax,yTrain, batch_size=10, epochs=100)
        print("MLP %i %i:" %(neurons1, neurons2))
        plotHystory(history)
        accuracy = classifier.evaluate(
          xTest_minmax,
          yTest
        )
        print(accuracy)
        dfRes = dfRes.append({'N1':neurons1  , "N2": neurons2, "Accuracy": accuracy[1]} , i
```

executed in 9m 44s, finished 18:15:11 2019-09-14

```
Epoch 1/100
1725/1725 [==============================] - 2s 936us/step - loss: 2.0746
- acc: 0.1780
Epoch 2/100
1725/1725 [==============================] - 0s 117us/step - loss: 2.0647
- acc: 0.2116
Epoch 3/100
1725/1725 [==============================] - 0s 148us/step - loss: 2.0515
- acc: 0.2209
Epoch 4/100
1725/1725 [==============================] - 0s 154us/step - loss: 2.0266
- acc: 0.2701
Epoch 5/100
1725/1725 [==============================] - 0s 157us/step - loss: 1.9880
- acc: 0.2470
Epoch 6/100
1725/1725 [==============================] - 0s 159us/step - loss: 1.9418
- acc: 0.2626
```

Epoch 7/100

Entrée [871]:

```
dfRes
```

executed in 14ms, finished 18:16:11 2019-09-14

Out[871]:

	N1	N2	Accuracy
0	5.0	3.0	0.520833
1	5.0	4.0	0.531250
2	5.0	5.0	0.541667
3	5.0	6.0	0.557292
4	6.0	3.0	0.531250
5	6.0	4.0	0.520833
6	6.0	5.0	0.526042
7	6.0	6.0	0.557292
8	7.0	3.0	0.468750
9	7.0	4.0	0.572917
10	7.0	5.0	0.541667
11	7.0	6.0	0.572917
12	8.0	3.0	0.520833
13	8.0	4.0	0.614583
14	8.0	5.0	0.473958
15	8.0	6.0	0.536458
16	9.0	3.0	0.625000
17	9.0	4.0	0.531250
18	9.0	5.0	0.489583
19	9.0	6.0	0.510417

Entrée [872]:

```
dfRes["Accuracy"].max()
```

executed in 7ms, finished 18:17:23 2019-09-14

Out[872]:

```
0.625
```

Entrée [876]:

```
# build the winner architecture and make predictions

dfRes = pd.DataFrame(columns=["dropOut1","dropOut2","Accuracy"])
neurons1 = 9
neurons2 = 3

for dropOut1 in [i/10 for i in range(3)]:
    for dropOut2 in [i/10 for i in range(3)]:

        np.random.seed(42) # set seed for reproducibility

        classifier = Sequential()
        #First Hidden Layer
        classifier.add(Dense(neurons1, activation='relu', kernel_initializer='random_normal
        #Second  Hidden Layer
        if dropOut1 >0:
            classifier.add(Dropout(dropOut1))
        classifier.add(Dense(neurons2, activation='relu', kernel_initializer='random_normal
        if dropOut2 >0:
            classifier.add(Dropout(dropOut2))
        #Output Layer
        classifier.add(Dense(outputDimY, activation='softmax'))

        #Compiling the neural network
        classifier.compile(optimizer ="adam",loss='sparse_categorical_crossentropy',
                           metrics =['accuracy'])
        #classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac

        #Fitting the data to the training dataset
        history = classifier.fit(xTrain_minmax,yTrain, batch_size=10, epochs=100)
        print("MLP %i %i:" %(neurons1, neurons2))
        plotHystory(history)
        accuracy = classifier.evaluate(
          xTest_minmax,
          yTest
        )
        print("drop out 1 %f, drop out 2 %f, accuracy %f" %(dropOut1, dropOut2, accuracy[1]
        dfRes = dfRes.append({'dropOut1':dropOut1  , "dropOut2": dropOut2, "Accuracy": accu
```

executed in 14m 47s, finished 20:36:22 2019-09-14

```
Epoch 6/100
1725/1725 [==============================] - 1s 494us/step - loss: 1.9508
- acc: 0.3159
Epoch 7/100
1725/1725 [==============================] - 1s 475us/step - loss: 1.8690
- acc: 0.3739 0s - loss: 1.8705
Epoch 8/100
1725/1725 [==============================] - 1s 469us/step - loss: 1.7854
- acc: 0.4029
Epoch 9/100
1725/1725 [==============================] - 1s 472us/step - loss: 1.7128
- acc: 0.4116
Epoch 10/100
1725/1725 [==============================] - 1s 484us/step - loss: 1.6508
- acc: 0.4157
Epoch 11/100
1725/1725 [==============================] - 1s 554us/step - loss: 1.5978
- acc: 0.4133
Epoch 12/100
```

```
Epoch 12/100
1725/1725 [==============================] - 1s 468us/step - loss: 1.5546
```

Entrée [877]:

```
dfRes["Accuracy"].max()
```

executed in 13ms, finished 20:37:50 2019-09-14

Out[877]:

```
0.6302083333333334
```

Success!!! We have a new best classification model, beating the previous by 4%!

Entrée [878]:

```
dfRes
```

executed in 15ms, finished 20:38:01 2019-09-14

Out[878]:

	dropOut1	dropOut2	Accuracy
0	0.0	0.0	0.630208
1	0.0	0.1	0.609375
2	0.0	0.2	0.593750
3	0.1	0.0	0.562500
4	0.1	0.1	0.583333
5	0.1	0.2	0.515625
6	0.2	0.0	0.567708
7	0.2	0.1	0.510417
8	0.2	0.2	0.005208

Entrée [881]:

```
# train best model again
# build the winner architecture and make predictions

dfRes = pd.DataFrame(columns=["dropOut1","dropOut2","Accuracy"])
neurons1 = 9
neurons2 = 3

np.random.seed(42) # set seed for reproducibility

classifier = Sequential()
#First Hidden Layer
classifier.add(Dense(neurons1, activation='relu', kernel_initializer='random_normal', input
#Second  Hidden Layer
classifier.add(Dense(neurons2, activation='relu', kernel_initializer='random_normal'))
#Output Layer
classifier.add(Dense(outputDimY, activation='softmax'))

#Compiling the neural network
classifier.compile(optimizer ="adam",loss='sparse_categorical_crossentropy',
                   metrics =['accuracy'])

#Fitting the data to the training dataset
history = classifier.fit(xTrain_minmax,yTrain, batch_size=10, epochs=100)

plotHystory(history)
accuracy = classifier.evaluate(
  xTest_minmax,
  yTest
)
print ("Accuracy:", accuracy[1])
```

executed in 1m 55.0s, finished 20:49:08 2019-09-14

```
Epoch 1/100
1725/1725 [==============================] - 11s 6ms/step - loss: 2.0664 -
acc: 0.1774
Epoch 2/100
1725/1725 [==============================] - 1s 516us/step - loss: 2.0507
- acc: 0.1757
Epoch 3/100
1725/1725 [==============================] - 1s 531us/step - loss: 2.0464
- acc: 0.1757
Epoch 4/100
1725/1725 [==============================] - 1s 542us/step - loss: 2.0383
- acc: 0.1849
Epoch 5/100
1725/1725 [==============================] - 1s 505us/step - loss: 2.0095
- acc: 0.2638
Epoch 6/100
1725/1725 [==============================] - 1s 548us/step - loss: 1.9518
- acc: 0.3241
Epoch 7/100
```

Entrée [882]:

```
xPredict = classifier.predict(xTest_minmax)
```

executed in 3.65s, finished 20:49:35 2019-09-14

Entrée [883]:

```
xPredict.shape
```

executed in 10ms, finished 20:49:35 2019-09-14

Out[883]:

```
(192, 8)
```

Entrée [884]:

```
xPredict = pd.DataFrame(
    classifier.predict(xTest_minmax), index = xTest.index)
xPredict.round(2).head()
```

executed in 75ms, finished 20:49:37 2019-09-14

Out[884]:

	0	1	2	3	4	5	6	7
Date								
2018-12-21	0.05	0.0	0.32	0.12	0.21	0.18	0.00	0.12
2018-12-22	0.07	0.0	0.33	0.12	0.19	0.19	0.00	0.10
2018-12-23	0.08	0.0	0.33	0.12	0.19	0.19	0.00	0.10
2018-12-24	0.03	0.0	0.26	0.14	0.24	0.14	0.01	0.18
2018-12-25	0.06	0.0	0.31	0.12	0.21	0.18	0.00	0.11

Entrée [885]:

```
xPrecdictClust = xPredict.apply(np.argmax, axis=1)

xPrecdictClust.head()
```

executed in 35ms, finished 20:49:40 2019-09-14

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:56: Fut
ureWarning:
The current behaviour of 'Series.argmax' is deprecated, use 'idxmax'
instead.
The behavior of 'argmax' will be corrected to return the positional
maximum in the future. For now, use 'series.values.argmax' or
'np.argmax(np.array(values))' to get the position of the maximum
row.
  return getattr(obj, method)(*args, **kwds)
```

Out[885]:

```
Date
2018-12-21    2
2018-12-22    2
2018-12-23    2
2018-12-24    2
2018-12-25    2
dtype: int64
```

Entrée [886]:

```
differences = [i for i in (xPrecdictClust-yTest)]

differences = np.asarray(differences)
print(differences)
```

executed in 11ms, finished 20:49:42 2019-09-14

```
[-5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -4 -5 -5 -5 -5
 -5 -5 -5 -5  2 -4 -4 -4 -4 -4  1  1 -4 -4 -2  1 -3  0  0 -3  0  0  1  1
 -2  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0 -4 -4  0  0  0  0  1  1  0  0  1  1  1  0  0  0  0  0  0  0
  0  0  0  0  0 -1 -1  0 -1 -1  0  0  0  0  0  0  0  0 -7  0  3  3  0  0
  0  0  0  0  7  0 -1 -1  0  0  0  0  0  0 -1  0  0  0  7  7  7  7  7  7]
```

Entrée [887]:

```
dif0 = differences[differences==0]
len(dif0)/len(differences) # check accuracy number
```

executed in 10ms, finished 20:49:46 2019-09-14

Out[887]:

```
0.6302083333333334
```

Entrée [888]:

```
report = classification_report(yTest, xPrecdictClust)
print(report)
```

executed in 20ms, finished 20:49:48 2019-09-14

```
              precision    recall  f1-score   support

           0       0.64      0.56      0.60        32
           1       0.93      0.93      0.93        98
           2       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         4
           6       0.00      0.00      0.00        15
           7       0.46      0.29      0.35        42

    accuracy                           0.63       192
   macro avg       0.34      0.30      0.31       192
weighted avg       0.68      0.63      0.65       192


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.p
y:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Entrée [889]:

```
# Check classification metrics report
from sklearn.metrics import classification_report
report = classification_report(yTest, xPrecdictClust, output_dict=True)
display(report)
print("Precision: %s" %(round(report['accuracy'],2)))
```

executed in 16ms, finished 20:49:52 2019-09-14

```
{'0': {'precision': 0.6428571428571429,
  'recall': 0.5625,
  'f1-score': 0.6000000000000001,
  'support': 32},
 '1': {'precision': 0.9285714285714286,
  'recall': 0.9285714285714286,
  'f1-score': 0.9285714285714286,
  'support': 98},
 '2': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 1},
 '4': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 4},
 '6': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 15},
 '7': {'precision': 0.46153846153846156,
  'recall': 0.2857142857142857,
  'f1-score': 0.35294117647058826,
  'support': 42},
 'accuracy': 0.6302083333333334,
 'macro avg': {'precision': 0.3388278388278389,
  'recall': 0.2961309523809524,
  'f1-score': 0.3135854341736695,
  'support': 192},
 'weighted avg': {'precision': 0.682062728937729,
  'recall': 0.6302083333333334,
  'f1-score': 0.6511642156862746,
  'support': 192}}
```

```
Precision: 0.63
```

Entrée [890]:

```
# compute error values
# compute hourly error on the forecast
dfPref = pd.DataFrame(index=xTest.index, columns=cols)
dfPref["cluster_id"] = xPrecdictClust

# populate the power values
for prefix in ["PrioP","PrioH"]:
            for iPair in range(1,4+1):
                res = np.zeros(len(dfPref.index))
                for myIndex in range(len(dfPref.index)):
                    indexInOrigDF = int(dfPref.iloc[myIndex]["cluster_id"])
                    res[myIndex] = df_resizePower.iloc[centers[indexInOrigDF]][prefix+str(i
                dfPref[prefix+str(iPair)] = res

# resize power values with the predicted maximum energy (imported data)
for i in range (1,4+1):
    dfPref["PrioP"+str(i)] =  dfPref["PrioP"+str(i)]*df_MLPprediction["Prediction"]

dfPref.fillna(0, inplace=True)

RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2 = hourlyErrorMeasure (dfPref, df.iloc[1917-192:

df_error = pd.DataFrame(data = [[RMSE, MAE, R2, DailyRMSE, DailyMAE, DailyR2]],
                                  columns = ["RMSE", "MAE", "R2", "Daily RMSE", "Daily MAE"
df_error
```

executed in 719ms, finished 20:54:57 2019-09-14

```
Hourly comparison: prediction vs. actual values
RMSE :14.19
MAE :7.94
R^2 :62.27
Daily comparison (max prod): prediction vs. actual values
RMSE :205.33
MAE :143.04
R^2 :80.25
```

Out[890]:

	RMSE	MAE	R2	Daily RMSE	Daily MAE	DailyR2
0	14.19	7.94	62.27	205.33	143.04	80.25