Version 1.0 / 19.09.2019

Hydro power plant constraints forecast

1 Subproblem 1 - Maximum and minimum energy production

This module implements the predictive models for the daily energy values, i.e. minimum and maximum production. First we implement models from teh scikit learn library, using a pipeline to find best parameters. Then we build deep learning models using Keras.

2 Import libraries

Entrée [3]:

```
import math
import pandas as pd
import numpy as np
import array as arr
from pandas import ExcelWriter
from pandas import ExcelFile
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error
import statistics
from functools import reduce
import random
from sklearn import metrics

%matplotlib inline
```

executed in 3.09s, finished 10:16:58 2019-09-19

3 Read source file into data frame and display columns

Entrée [4]:

```
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 53ms, finished 10:16:58 2019-09-19

3.1 Check content and basic statistics

Entrée [5]:

```
df.head()
```

executed in 27ms, finished 10:16:58 2019-09-19

Out[5]:

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

5 rows × 24 columns

Entrée [6]:

```
df.tail()
```

executed in 24ms, finished 10:16:59 2019-09-19

Out[6]:

	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 [%]	Availabi plant 2
Date										
2019-06-26	2019-06-26	479.633867	1179.8	-108.0	785.1	625.8	0.63993	30000.0	1.0	
2019-06-27	2019-06-27	569.565217	1735.9	-111.0	868.0	547.2	0.66066	30000.0	1.0	
2019-06-28	2019-06-28	509.610984	1786.9	79.0	644.6	595.3	0.68188	30000.0	1.0	
2019-06-29	2019-06-29	479.633867	1614.6	83.0	609.2	479.1	0.70126	30000.0	1.0	
2019-06-30	2019-06-30	479.633867	1516.5	-105.0	834.7	357.7	0.71942	30000.0	1.0	

5 rows × 24 columns

Entrée [7]:

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

executed in 13ms, finished 10:16:59 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
```

3.2 Import baseline

Entrée [8]:

```
# Read baseline for benchmark data as well
df_benchmark = pd.read_csv("baseline_dataframe.csv", parse_dates=['Date'], date_parser=date
# Force index to be date (as provided in the first column)
df_benchmark.index = df_benchmark['Date']

df_benchmark.info()
df_benchmark.head()
```

executed in 51ms, finished 10:17:00 2019-09-19

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1918 entries, 2014-04-01 to 2019-07-01
Data columns (total 10 columns):
Date              1918 non-null datetime64[ns]
P1 [MW]           1918 non-null float64
P2 [MW]           1918 non-null float64
P3 [MW]           1918 non-null float64
P4 [MW]           1918 non-null int64
H1 [#]            1918 non-null int64
H2 [#]            1918 non-null int64
H3 [#]            1918 non-null int64
Min Prod [MWh]    1918 non-null float64
MaxEnergy         1918 non-null float64
dtypes: datetime64[ns](1), float64(5), int64(4)
memory usage: 164.8 KB
```

Out[8]:

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

Entrée [9]:

```
# copy baseline values into main dataframe, using consistent labels
df["Baseline_Min prod"] = df_benchmark["Min Prod [MWh]"]
df["Baseline_Max prod"] = df_benchmark["MaxEnergy"]
```

executed in 8ms, finished 10:17:00 2019-09-19

4 Data preparation for ML

4.1 Test / train split and input / output features separation

Entrée [10]:

```
# define train / test split ratio (applied to availabe data points in dataset)
splitRatio = 0.9
```

executed in 3ms, finished 10:17:01 2019-09-19

Entrée [11]:

```
# 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:]
    return [xTrain, xTest, yTrain, yTest, yBaseLine ]
```

executed in 4ms, finished 10:17:01 2019-09-19

4.2 Compute baseline error

Entrée [14]:

```
# Compute target error metrics to beat, i.e. error metrics from the baseline dataset

# define regressors, i.e. list of features to be used as input for our regression problem
regressors = ['Inflow lake 1 [m3]', \
              'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
              'Vol lake 1 [%]', 'Availability plant 1 [%]', \
              'Availability plant 2 [%]', 'Availability plant 3 [%]', \
              'Availability plant 4 [%]']

for target_feature in ["Min prod","Max prod"]:
    xTrain, xTest, yTrain, yTest, yBaseLine =  GetDataSplit(df, regressors,
                                                            target_feature, "Baseline_"
    print('Baseline values for target "'+target_feature+ '" (daily value)')
    print('RMSE :'+ str(round(math.sqrt(metrics.mean_squared_error(yTest,yBaseLine)),2)))
    print('MAE :'+ str(round(metrics.mean_absolute_error(yTest,yBaseLine),2)))
    print('R^2 :'+ str(round(metrics.r2_score(yTest,yBaseLine)*100, 2)))
```

executed in 15ms, finished 10:17:57 2019-09-19

```
Baseline values for target "Min prod" (daily value)
RMSE :237.33
MAE :181.59
R^2 :-192.26
Baseline values for target "Max prod" (daily value)
RMSE :447.8
MAE :430.38
R^2 :6.06
```

Entrée [15]:

```
# 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(df.index.min(), df.index.max(), freq='1H', closed='left')}
    )
```

executed in 4ms, finished 10:18:00 2019-09-19

Entrée [16]:

```
df_hourly.head()
```

executed in 7ms, finished 10:18:01 2019-09-19

Out[16]:

	Hours
0	2014-04-01 00:00:00
1	2014-04-01 01:00:00
2	2014-04-01 02:00:00
3	2014-04-01 03:00:00
4	2014-04-01 04:00:00

Entrée [17]:

```
len(df_hourly.index)
```

executed in 3ms, finished 10:18:01 2019-09-19

Out[17]:

45984

Entrée [18]:

```
firstDate = df.index.min()
powerHours = np.zeros(45984)
powerHoursBaseline =  np.zeros(45984)

# loop over all date
for myDate in df.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 hours)
    currHour = index
    for iPair in range(1,4+1):
        pwrValue = df.loc[myDate, "PrioP"+str(iPair)]
        nbHours = int(df.loc[myDate, "PrioH"+str(iPair)])
        if pwrValue > 0:
            #print("Date %s, Pair %i, Pwr %d, Hours %f" %(myDate, iPair, pwrValue, nbHours))
            powerHours[currHour:currHour+nbHours] = pwrValue
            currHour = currHour + nbHours
    # compute hourly vector of baseline values, using the 3 pairs (power/nb of hours)
    currHour = index
    for iPair in range(1,3+1):
        pwrValue = df_benchmark.loc[myDate, "P"+str(iPair)+" [MW]"]
        nbHours = int(df_benchmark.loc[myDate, "H"+str(iPair)+" [#]"])
        if pwrValue > 0:
            #print("Date %s, Pair %i, Pwr %d, Hours %f" %(myDate, iPair, pwrValue, nbHours))
            powerHoursBaseline[currHour:currHour+nbHours] = pwrValue
            currHour = currHour + nbHours
```

executed in 311ms, finished 10:18:02 2019-09-19

Entrée [19]:

```
df_hourly["Power"] = powerHours
df_hourly["PowerBaseLine"] = powerHoursBaseline

df_hourly.head()
```

executed in 13ms, finished 10:18:03 2019-09-19

Out[19]:

	Hours	Power	PowerBaseLine
0	2014-04-01 00:00:00	73.8	73.7
1	2014-04-01 01:00:00	73.8	73.7
2	2014-04-01 02:00:00	73.8	73.7
3	2014-04-01 03:00:00	73.8	47.8
4	2014-04-01 04:00:00	66.0	47.8

Entrée [20]:

```
# compute error metrics over hourly values calculated above
print('Hourly comparison: prediction baseline vs. actual values')
print('RMSE :'+ str(round(math.sqrt(metrics.mean_squared_error(df_hourly["Power"],df_hourly
print('MAE :'+ str(round( metrics.mean_absolute_error(df_hourly["Power"],df_hourly["PowerBa
print('R^2 :'+ str(round(metrics.r2_score( df_hourly["Power"],df_hourly["PowerBaseLine"])*1
```

executed in 30ms, finished 10:18:04 2019-09-19

```
Hourly comparison: prediction baseline vs. actual values
RMSE :20.31
MAE :14.41
R^2 :42.34
```

5 Model selection

5.1 Model selection using scikit-learn library

Entrée [21]:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble.forest import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
import math
```

executed in 274ms, finished 10:18:05 2019-09-19

5.1.1 plotModelPrediction : function to produce plots out of models predictions

Entrée [22]:

```
# function to produce plots out of models predictions
# plot actual value, prediction and baseline in one plot
# plot error along time below in another plot
# Add error metrics values at the bottom and model's parameters
# Produce file in SVG format, i.e. vector format for best quality import in final report
import matplotlib.pylab as pl
import matplotlib.gridspec as gridspec

def plotModelPrediction(model, target_label, y_train, pred, y_test, target_baseline, test_i
                        addRegressor, export_prediction = False):

   # build dataframe to be plotted, plot graph and save it as a file
    if  isinstance(model, str):
        model_name = model
    else:
        model_name = type(model).__name__

    df_pred = pd.DataFrame(pred, columns=['Forecast '+model_name], index = test_index)
    df_pred["Ground truth"] = y_test
    df_pred["Mean train values"] = y_train.mean()
    df_pred["Baseline"]  = target_baseline

    fig_size = plt.rcParams["figure.figsize"]
    fig_size[0] = 15
    fig_size[1] = 15
    plt.rcParams["figure.figsize"] = fig_size

    # Create 1x3 sub plots
    gs = gridspec.GridSpec(3, 2)

    fig = pl.figure()
    ax = pl.subplot(gs[0, :]) # row 0, col 0
    df_pred.plot(title = target_label+' forecast '+model_name+'(additional regressor: '+add

    # plot error along time
    ax = pl.subplot(gs[1, :]) # row 1, col 0
    df_error = pd.DataFrame(df_pred["Ground truth"]-df_pred['Forecast '+model_name], columr
    #df_error["numIndex"] = [i for i in range(len(test_index))]
    #maxError = df_error["Error"].max()
    #df_error["NormError"] = 10*np.absolute(df_error["Error"]/maxError)
    #df_error["NormError"] = 10
    df_error.plot(title ="Residuals", style=".",grid=True, ax=ax)
    plt.axhline(y=0.5, color='r', linestyle='-')

    # plot predicted vs actual values
    ax = pl.subplot(gs[2, 0]) # row 2, col 0
    plt.scatter( df_pred['Forecast '+model_name], df_pred["Ground truth"], c='b', alpha=0.3
    plt.plot([0,1000], [1,1000], ls="--", c="r")
    plt.title("Predicted values vs ground truth")
    plt.xlabel("Predicted value")
    plt.ylabel("Ground truth")

    # plot predicted value vs residuals
    ax = pl.subplot(gs[2, 1]) # row 2, col 1
    plt.scatter( df_pred['Forecast '+model_name], df_pred["Ground truth"]-df_pred['Forecast
    plt.axhline(y=0.5, color='r', linestyle='-')
    plt.title("Predicted values vs residuals")
    plt.xlabel("Predicted value")
    plt.ylabel("Residuals")
```

```
    plt.tight_layout()

    # add textual elements: error metrics values, model name and parameters
    txt = 'Target :'+ target_label+" / "
    txt = txt + 'RMSE :'+ str(round(math.sqrt(metrics.mean_squared_error(y_test,pred)),2))+
    txt = txt + 'MAE :'+ str(round(metrics.mean_absolute_error(y_test,pred),2))+" / "
    txt = txt + 'R^2 :'+ str(round(metrics.r2_score(y_test,pred)*100, 2))+" "

    fig.text(0.5, 0.01, txt, ha='center')

    fig.savefig('Forecast'+target_label+model_name+'+'+addRegressor+'.svg', format='svg')

    if export_prediction:
        df_pred.to_csv('Forecast'+target_label+model_name+'+'+addRegressor+'export_datafram
```

executed in 17ms, finished 10:18:06 2019-09-19

5.1.2 TrainEvalModel : function to train model and find best hyperparameters

Entrée [23]:

```
# function to train the model and set its best parameters, then evaluate it
# return best parameters

def TrainEvalModel(X_train, X_test, y_train, y_test, target_label, target_baseline, test_ir
    pipeline = Pipeline([('preprocessor', StandardScaler()),
                         ('model', model)])
    print('Model: ', model)
    print(params)
    grid = GridSearchCV(pipeline, params, scoring='r2', n_jobs=-1, cv=3, verbose=1)
    grid.fit(X_train, y_train)
    pred = grid.best_estimator_.predict(X_test)

    print('Target :', target_label)
    print('Regressors :', X_train.columns)
    print('Parameters: ', grid.best_params_ )
    print('Root Mean Square Error: %1.4f' % (math.sqrt(metrics.mean_squared_error(y_test,pr
    print('Mean Absolute Error: %1.4f' % (metrics.mean_absolute_error(y_test,pred)))
    print('R2 score : %1.4f' %(metrics.r2_score(y_test,pred)*100))
    print('\n\n')

    plotModelPrediction(model,target_label, y_train, pred, y_test, target_baseline, test_ir

    return [math.sqrt(metrics.mean_squared_error(y_test,pred)),
                      metrics.mean_absolute_error(y_test,pred),
                      metrics.r2_score(y_test,pred)*100,
                      grid.best_params_]
```

executed in 8ms, finished 10:18:07 2019-09-19

Entrée [24]:

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

executed in 2ms, finished 10:18:08 2019-09-19

5.1.3 Models evaluation (scikit learn)

Entrée [25]:

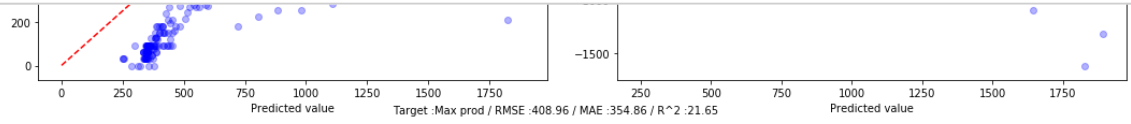
```
# define regressors, i.e. list of features to be used as input for our regression problem
regressors = ['Inflow lake 1 [m3]', \
              'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
              'Vol lake 1 [%]', 'Availability plant 1 [%]', \
              'Availability plant 2 [%]', 'Availability plant 3 [%]', \
              'Availability plant 4 [%]']
```

executed in 4ms, finished 10:18:10 2019-09-19

Entrée [26]:

```
# Build list of algorithms and associated "usual" parameters to be tested
# call function to build train and test datasets, for the two target variables
# call function to tain and evaluate models (plotting results)
#
# Build a table of results:
#  - first column contains algorithm name
#  - second column contains target feature
#  - third column contains list of input features
#  - next columns contain error metrics (RMSE, MAE, R2)
#  - last column contains best parameters
#  - lines contain algorithm name (linear regression, RF, SVR, ...)

models = [ LinearRegression(), DecisionTreeRegressor(), RandomForestRegressor(), SVR(), MLP
params = [{} , # linear regression
          {} , # Decision tree
          {'model__n_estimators' : [500,1000,1500,2000],
           'model__max_depth': [5,7,9,12, 15, 18,None],
           'model__min_samples_split':[0.1,0.5,1.0]
           }, # Random forest
          {'model__C': [1, 10, 100, 1000],
           'model__kernel': ['linear','rbf'],
           'model__epsilon': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10]
           'model__gamma': [0.0001, 0.001, 0.005, 0.1, 1, 3, 5]
          }, # Support Vector Regression
          {}] # Multi layer perceptron regressor

AddRegressors = ['', 'Max lake 1 [1000m3]','Weekend']

columns = ['Algorithm','Target','Features','RMSE','MAE','R^2', 'Parameters']
df_results = pd.DataFrame(columns = columns)
line_nb = 0

for target_feature in ["Min prod","Max prod"]:
    for addRegressor in AddRegressors:
        print("Additional regressor: ", addRegressor)
        fullRegressors = regressors + [addRegressor]
        xTrain, xTest, yTrain, yTest, yBaseLine =  GetDataSplit(df, fullRegressors,
                                                                target_feature, "Baseline_"
        for m,p in zip(models, params):
            res = TrainEvalModel(xTrain, xTest, yTrain, yTest, target_feature, yBaseLine, x
            df_results.loc[line_nb,:] = [m,target_feature, xTest.columns, res[0], res[1], r
            line_nb = line_nb+1
```

executed in 18m 11s, finished 10:36:21 2019-09-19



5.1.4 Detailed analysis of linear regression model

We build this model and train it to extract the weights and provide clarity on results

Entrée [450]:

```
regressors = ['Inflow lake 1 [m3]', 'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', \
       'Inflow lake 4 [m3]', 'Vol lake 1 [%]', 'Availability plant 1 [%]', \
       'Availability plant 2 [%]', 'Availability plant 3 [%]', \
       'Availability plant 4 [%]', 'Weekend']

regr = linear_model.LinearRegression()

# define target
target_feature = "Max prod"
fullRegressors = regressors

# scale input values between 0 and 1, so coefficients are comparable
scaler = MinMaxScaler()
scaled_df = scaler.fit_transform(df[fullRegressors])

scaled_df= pd.DataFrame(scaled_df, columns = fullRegressors)

xTrain, xTest, yTrain, yTest, yBaseLine =  GetDataSplit(scaled_df, fullRegressors,
                                                        target_feature, "Baseline_"+target_

# Train the model using the training sets
regr.fit(xTrain, yTrain)

# Make predictions using the testing set
pred = regr.predict(xTest)

# The coefficients
print('Coefficients:  \n', np.round(regr.coef_,2))
```

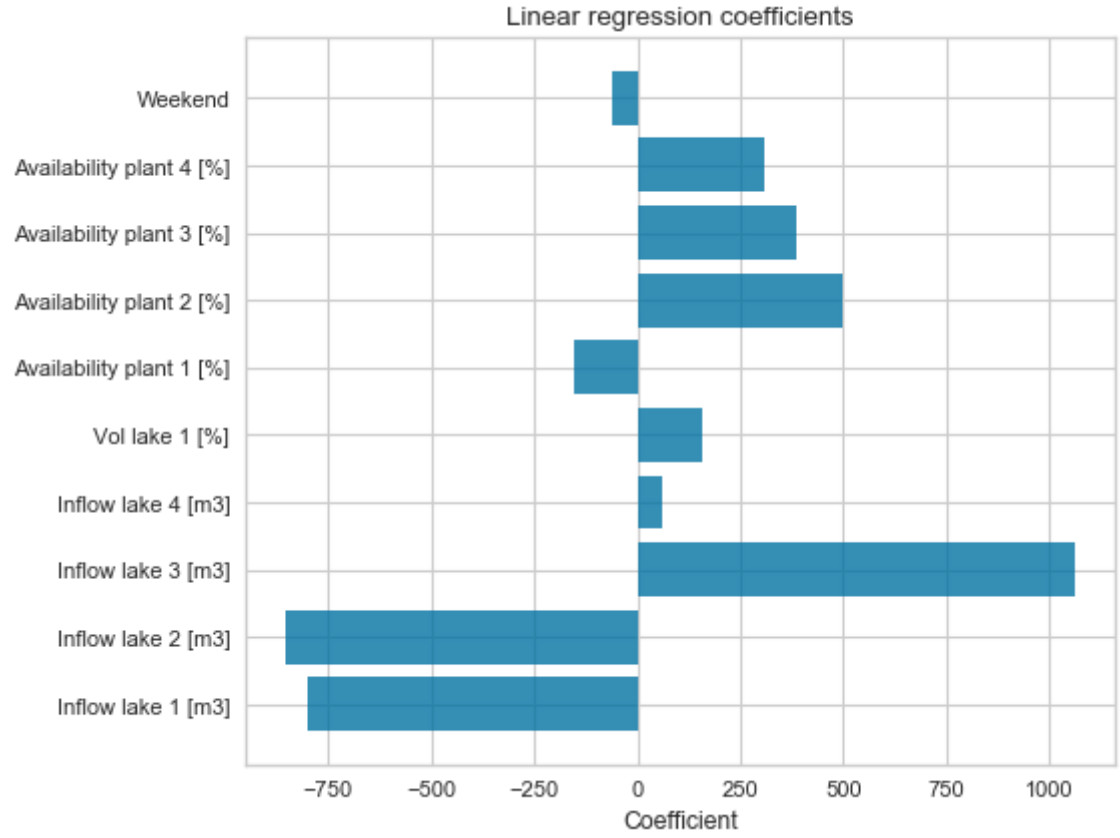
executed in 20ms, finished 12:20:20 2019-09-18

```
Coefficients:
 [-796.95 -852.7  1062.94   62.01  157.13 -153.31  498.45  386.09  305.88
  -59.38]
```

Entrée [451]:

```
fig=plt.figure(figsize=(7, 6), dpi= 80, edgecolor='k')
plt.barh(range(len(regr.coef_)), regr.coef_, align='center', alpha=0.8)
plt.yticks(range(len(regr.coef_)), fullRegressors)
plt.xlabel('Coefficient')
plt.title('Linear regression coefficients')

plt.show()
```

executed in 183ms, finished 12:20:22 2019-09-18



5.1.5 Detailed analysis of SVR model

We build the SVR with best parameters and analyse variable importance

Entrée [470]:

```
# we add max lake and weekend to the list of regressors
regressors = ['Inflow lake 1 [m3]', \
           'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
           'Vol lake 1 [%]', 'Availability plant 1 [%]', \
           'Availability plant 2 [%]', 'Availability plant 3 [%]', \
           'Availability plant 4 [%]' , 'Weekend' ]
```

executed in 4ms, finished 13:31:22 2019-09-18

Entrée [471]:

```
# best parameters: {'model__C': 100, 'model__epsilon': 5, 'model__gamma': 0.005, 'model__ke

model =  SVR()

params =  [{'model__C': [100],
           'model__kernel': ['rbf'],
           'model__epsilon': [5],
           'model__gamma': [0.005]
          }]

# define target
target_feature = "Min prod"
xTrain, xTest, yTrain, yTest, yBaseLine =  GetDataSplit(df, regressors,
                                                        target_feature, "Baseline_"+target_

# scaling
pipeline = Pipeline([('preprocessor', StandardScaler()),
                     ('model', model)])

grid = GridSearchCV(pipeline, params, scoring='r2', n_jobs=-1, cv=3, verbose=1)
grid.fit(xTrain, yTrain)
pred = grid.best_estimator_.predict(xTest)
```

executed in 231ms, finished 13:31:23 2019-09-18

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    0.0s finished
```

Entrée [472]:

```
print('Target :', target_feature)
print('Regressors :', xTrain.columns)
print('Root Mean Square Error: %1.4f' % (math.sqrt(metrics.mean_squared_error(yTest,pred))))
print('Mean Absolute Error: %1.4f' % (metrics.mean_absolute_error(yTest,pred)))
print('R2 score : %1.4f' %(metrics.r2_score(yTest,pred)*100))
print('\n\n')
```

executed in 7ms, finished 13:31:24 2019-09-18

```
Target : Min prod
Regressors : Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m3]', 'Inflow lake
3 [m3]',
       'Inflow lake 4 [m3]', 'Vol lake 1 [%]', 'Availability plant 1 [%]',
       'Availability plant 2 [%]', 'Availability plant 3 [%]',
       'Availability plant 4 [%]', 'Weekend'],
      dtype='object')
Root Mean Square Error: 75.9431
Mean Absolute Error: 61.4425
R2 score : 70.0738
```

Entrée [473]:

```
xTest.iloc[190]
```

executed in 5ms, finished 13:31:25 2019-09-18

Out[473]:

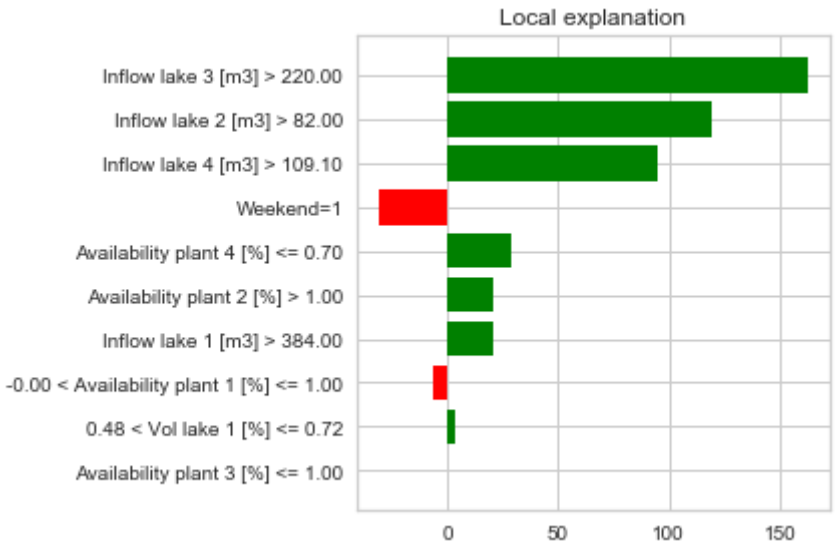
```
Inflow lake 1 [m3]          1614.6
Inflow lake 2 [m3]              83
Inflow lake 3 [m3]           609.2
Inflow lake 4 [m3]           479.1
Vol lake 1 [%]             0.70126
Availability plant 1 [%]         1
Availability plant 2 [%]         1
Availability plant 3 [%]         0
Availability plant 4 [%]         0
Weekend                       True
Name: 2019-06-29 00:00:00, dtype: object
```

Entrée [474]:

```
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(xTrain.values,
    feature_names=xTrain.columns,
    class_names=['Min prod'],
    categorical_features=[9],
    mode='regression')

# Now explain a prediction
exp = explainer.explain_instance(xTest.iloc[190],  grid.best_estimator_.predict,
        num_features=10)

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 6
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size
exp.as_pyplot_figure()
plt.tight_layout()
```

executed in 896ms, finished 13:31:28 2019-09-18

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\ridge.py:14
7: LinAlgWarning: Ill-conditioned matrix (rcond=6.37823e-28): result may not
be accurate.
  overwrite_a=True).T
```



5.1.6 Algorithms ranking

Entrée [459]:

```
# display results table
df_results[df_results["Target"]=='Min prod'].sort_values(by=['Target','RMSE']).head()
```

executed in 54ms, finished 13:29:04 2019-09-18

Out[459]:

	Algorithm	Target	Features	RMSE	MAE	R^2	F
13	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	75.9431	61.4425	70.0738	{' 'mode 5
3	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	80.2263	64.7046	66.6029	{' 'mode 5
12	RandomForestRegressor(bootstrap=True, criterio...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	99.2355	71.853	48.9014	
11	DecisionTreeRegressor(criterion='mse', max_dep...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	111.684	85.7012	35.2769	
10	LinearRegression(copy_X=True, fit_intercept=Tr...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	114.696	87.2005	31.7395	

Entrée [341]:

```
df_results[df_results["Target"]=='Max prod'].sort_values(by=['Target','RMSE']).head()
```

executed in 18ms, finished 10:56:13 2019-09-18

Out[341]:

	Algorithm	Target	Features	RMSE	MAE	R^2	Paramete
23	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	217.853	167.744	77.7657	{'model__C 10 'model__epsilo 5, 'model_
28	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	242.063	200.899	72.5492	{'model__C 10 'model__epsilo 5, 'model_
18	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	245.503	207.061	71.7635	{'model__C 10 'model__epsilo 5, 'model_
20	LinearRegression(copy_X=True, fit_intercept=Tr...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	246.364	211.871	71.565	
25	LinearRegression(copy_X=True, fit_intercept=Tr...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	249.99	187.935	70.722	

Entrée [342]:

```
# Create column with only algo name (no parameters)
df_results["Algo name"] = df_results['Algorithm'].apply(lambda s: str(s)[:str(s).find('(')]
```

executed in 22ms, finished 10:56:20 2019-09-18

Entrée [343]:

```
# display best results per algorithm
best = pd.merge(df_results.groupby(['Algo name','Target'])['RMSE'].min(),df_results,on = ['
best[["Algo name","Target","RMSE","MAE","R^2"]].sort_values('RMSE')
```

executed in 35ms, finished 10:56:20 2019-09-18

Out[343]:

	Algo name	Target	RMSE	MAE	R^2
9	SVR	Min prod	75.9431	61.4425	70.0738
1	DecisionTreeRegressor	Min prod	102.956	79.0651	44.9979
7	RandomForestRegressor	Min prod	107.604	88.86	39.9196
3	LinearRegression	Min prod	114.696	87.2005	31.7395
8	SVR	Max prod	217.853	167.744	77.7657
5	MLPRegressor	Min prod	224.653	177.613	-161.878
2	LinearRegression	Max prod	246.364	211.871	71.565
4	MLPRegressor	Max prod	339.803	243.241	45.9058
0	DecisionTreeRegressor	Max prod	340.33	267.413	45.7379
6	RandomForestRegressor	Max prod	359.217	323.743	39.5479

5.2 Features engineering to improve predicting power

Entrée [480]:

```
# Concerning minimum production, as ACF suggests, we try adding calculated features to impr
# compute lagged feature: minimum power with one year lag. For first year, use average valu

from datetime import datetime, timedelta

LaggedMinPow = pd.Series(np.zeros(len(df)), index = df.index) # initialize to
lag = timedelta(days=365)
for index in df.index:
    LaggedMinPow[index] = df.loc[index-lag,"Min prod"] if index-lag in df.index \
                          else statistics.mean([df.loc[index+i*lag,"Min prod"] for i in

df["LaggedMinPow"] = LaggedMinPow
```

executed in 115ms, finished 13:39:31 2019-09-18

Entrée [481]:

```
# Same but average value over 5 days 365 days in the past (2 days before, days after)
LaggedAvgMinPow = pd.Series(np.zeros(len(df)), index = df.index) # initialize to
lag = timedelta(days=365)

for index in df.index:
    LaggedAvgMinPow[index] = np.average(df.loc[index-lag-timedelta(days=2):index-lag+timede
                                if index-lag-timedelta(days=2) in df.index else df.loc[inde
df["LaggedAvgMinPow"] = LaggedAvgMinPow
```

executed in 518ms, finished 13:39:32 2019-09-18

Entrée [482]:

```
models = [ LinearRegression(), DecisionTreeRegressor(), RandomForestRegressor(), SVR(), MLP
params = [{}, # Linear regression
          {}, # Decision tree
          {'model__n_estimators' : [500,1000,1500,2000],
           'model__max_depth': [5,7,9,12, 15, 18,None],
           'model__min_samples_split':[0.1,0.5,1.0]
          }, # Random forest
          {'model__C': [1, 10, 100, 1000],
           'model__kernel': ['linear','rbf'],
           'model__epsilon': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10]
           'model__gamma': [0.0001, 0.001, 0.005, 0.1, 1, 3, 5]
          }, # Support Vector Regression
          {}] # Multi layer perceptron regressor

regressors = ['Inflow lake 1 [m3]', \
           'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
           'Vol lake 1 [%]','Availability plant 1 [%]', \
           'Availability plant 2 [%]', 'Availability plant 3 [%]', \
           'Availability plant 4 [%]', 'Weekend']
AddRegressors = ['LaggedMinPow','LaggedAvgMinPow', 'TotalAvailablePower']
target_feature = "Min prod"

columns = ['Algorithm','Target','Features','RMSE','MAE','R^2', 'Parameters']
df_resLagFT = pd.DataFrame(columns = columns)
line_nb = 0

for addRegressor in AddRegressors:
    fullRegressors = regressors + [addRegressor]
    xTrain, xTest, yTrain, yTest, yBaseline = GetDataSplit(df, fullRegressors,target_featur
    for m,p in zip(models, params):
        res = TrainEvalModel(xTrain, xTest, yTrain, yTest, target_feature, yBaseLine, xTest
        df_resLagFT.loc[line_nb,:] = [m,target_feature, xTest.columns, res[0], res[1], res[
        line_nb = line_nb+1
```

executed in 10m 27s, finished 13:50:00 2019-09-18

```
Model:  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, nor
malize=False)
{}
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Target : Min prod
Regressors : Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m3]', 'Inflow la
ke 3 [m3]',
       'Inflow lake 4 [m3]', 'Vol lake 1 [%]', 'Availability plant 1 [%]',
       'Availability plant 2 [%]', 'Availability plant 3 [%]',
       'Availability plant 4 [%]', 'Weekend', 'LaggedMinPow'],
      dtype='object')
Parameters:  {}
Root Mean Square Error: 137.8593
Mean Absolute Error: 94.7158
R2 score : 1.3842



[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
```

Entrée [483]:

```
# display results table
df_resLagFT.sort_values(by=['Target','RMSE']).head()
```

executed in 19ms, finished 13:53:49 2019-09-18

Out[483]:

	Algorithm	Target	Features	RMSE	MAE	R^2	Par
13	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	76.4734	62.308	69.6544	{'mc 'model_ 10,
3	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	89.2483	67.5227	58.6692	{'mc 'model_ 5, 'r
11	DecisionTreeRegressor(criterion='mse', max_dep...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	105.421	83.065	42.3331	
8	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	110.182	78.804	37.0068	{'mc 'model_ 10,
10	LinearRegression(copy_X=True, fit_intercept=Tr...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	114.696	87.2005	31.7395	

Entrée [484]:

```
# Concerning both minimum and maximum production, assumption is made that the lake constrai
# i.e. if there will be a empty lake constraint in the next days, it makes sense that minim
# to force emptying the lake. For the same reason, maximum production is certainly higher t
# We compute an additional feature that is 1 if there is such a lake constrain in the next
# level goes below a given threshold (500)
LakeThreshold = 5000
LaggedLakeConstraint = pd.Series(np.zeros(len(df)), index = df.index) # initialize to 0
for i in range(10,60,5):
    lag = timedelta(days=i)
    for index in df.index:
        LaggedLakeConstraint[index] = 1 if index+lag in df.index and np.min(df.loc[index:ir
    df["LaggedLakeConstraint"+str(i)] = LaggedLakeConstraint
```

executed in 6.12s, finished 13:54:03 2019-09-18

Entrée [485]:

```
df.head()
```

executed in 28ms, finished 13:54:03 2019-09-18

Out[485]:

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 × 38 columns

Entrée [486]:

```
LaggedLakeConstraint[LaggedLakeConstraint>0].head()
```

executed in 9ms, finished 13:54:03 2019-09-18

Out[486]:

```
Date
2017-02-25    1.0
2017-02-26    1.0
2017-02-27    1.0
2017-02-28    1.0
2017-03-01    1.0
dtype: float64
```

Entrée [487]:

```
models = [ LinearRegression(), DecisionTreeRegressor(), RandomForestRegressor(), SVR(), MLP
params = [{}, # linear regression
          {}, # Decision tree
          {          }, # Random forest
          {'model__C': [100],
           'model__kernel': ['rbf'],
           'model__epsilon': [5],
           'model__gamma': [0.005]
          }, # Support Vector Regression
          {}] # Multi layer perceptron regressor

regressors = ['Inflow lake 1 [m3]', \
              'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
              'Vol lake 1 [%]','Availability plant 1 [%]', \
              'Availability plant 2 [%]', 'Availability plant 3 [%]', \
              'Availability plant 4 [%]', 'Weekend']

AddRegressors = []
for i in range(10,60,5):
    AddRegressors = AddRegressors+ ['LaggedLakeConstraint'+str(i)]

columns = ['Algorithm','Target','Features','RMSE','MAE','R^2', 'Parameters']
df_resLagLakeMax = pd.DataFrame(columns = columns)
line_nb = 0

for target_feature in ["Min prod","Max prod"]:
    for addRegressor in AddRegressors:
        fullRegressors = regressors + [addRegressor]
        print("Full list of regressors:", fullRegressors)
        xTrain, xTest, yTrain, yTest, yBaseLine = GetDataSplit(df, fullRegressors, target_f
        for m,p in zip(models, params):
            res = TrainEvalModel(xTrain, xTest, yTrain, yTest, target_feature, yBaseLine, x
            df_resLagLakeMax.loc[line_nb,:] = [m,target_feature, xTest.columns, res[0], res
            line_nb = line_nb+1
```

executed in 3m 35s, finished 13:57:37 2019-09-18

```
Full list of regressors: ['Inflow lake 1 [m3]', 'Inflow lake 2 [m3]', 'Inf
low lake 3 [m3]', 'Inflow lake 4 [m3]', 'Vol lake 1 [%]', 'Availability pl
ant 1 [%]', 'Availability plant 2 [%]', 'Availability plant 3 [%]', 'Avail
ability plant 4 [%]', 'Weekend', 'LaggedLakeConstraint10']
Model:  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, nor
malize=False)
{}
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Target : Min prod
Regressors : Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m3]', 'Inflow la
ke 3 [m3]',
       'Inflow lake 4 [m3]', 'Vol lake 1 [%]', 'Availability plant 1 [%]',
       'Availability plant 2 [%]', 'Availability plant 3 [%]',
       'Availability plant 4 [%]', 'Weekend', 'LaggedLakeConstraint10'],
      dtype='object')
Parameters:  {}
Root Mean Square Error: 116.4515
Mean Absolute Error: 90.6218
R2 score : 29.6336
```

Entrée [488]:

```
df_resLagLakeMax[(df_resLagLakeMax["Target"]=="Max prod") & (df_resLagLakeMax["RMSE"]<280)]
```

executed in 26ms, finished 14:25:40 2019-09-18

Out[488]:

	Algorithm	Target	Features	RMSE	MAE	R^2	Paramete
98	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	249.083	183.839	70.9341	{'model__C 10 'model__epsilo 5, 'model_
93	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	253.615	179.797	69.8667	{'model__C 10 'model__epsilo 5, 'model_
88	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	254.307	173.811	69.7019	{'model__C 10 'model__epsilo 5, 'model_
53	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	264.344	187.463	67.2633	{'model__C 10 'model__epsilo 5, 'model_
95	LinearRegression(copy_X=True, fit_intercept=Tr...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	266.594	188.413	66.7035	
83	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	266.677	174.682	66.6829	{'model__C 10 'model__epsilo 5, 'model_
73	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	267.857	182.613	66.3873	{'model__C 10 'model__epsilo 5, 'model_
78	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	271.019	180.094	65.589	{'model__C 10 'model__epsilo 5, 'model_
90	LinearRegression(copy_X=True, fit_intercept=Tr...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	273.85	194.572	64.8664	
85	LinearRegression(copy_X=True, fit_intercept=Tr...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	277.664	200.45	63.8809	
68	SVR(C=1.0, cache_size=200, coef0=0.0, degree=3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	277.704	190.503	63.8705	{'model__C 10 'model__epsilo 5, 'model_
70	LinearRegression(copy_X=True, fit_intercept=Tr...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	279.112	219.146	63.5032	

Entrée [489]:

```
df_resLagLakeMax.iloc[98,2]
```

executed in 5ms, finished 14:25:49 2019-09-18

Out[489]:

```
Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]',
       'Inflow lake 4 [m3]', 'Vol lake 1 [%]', 'Availability plant 1 [%]',
       'Availability plant 2 [%]', 'Availability plant 3 [%]',
       'Availability plant 4 [%]', 'Weekend', 'LaggedLakeConstraint55'],
      dtype='object')
```

5.3 Further model implementation with Multi Layer Perceptron

Entrée [31]:

```
# Keras MLP implementation
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.python.keras.initializers import RandomUniform
from tensorflow import set_random_seed
from keras.callbacks import EarlyStopping

def TrainEvalMLP(X_train, X_test, y_train, y_test, target_label, target_baseline, test_inde
                 addRegressor, mlpTopology, modelName, export_prediction = False):

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

    init = RandomUniform(minval=-0.05, maxval=0.05)

    # define model
    model = Sequential()
    FirstLayer = True
    for layerParams in mlpTopology:
        if FirstLayer:
            model.add(Dense(layerParams["size"], input_dim=len(X_train[0]), kernel_initiali
                            activation=layerParams["activation"]))
        else:
            model.add(Dense(layerParams["size"], kernel_initializer=init, activation=layerP
        if layerParams["dropOutAfter"] >0:
            model.add(Dropout(layerParams["dropOutAfter"]))
        FirstLayer = False

    model.summary()
    model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])

    es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=100)

    history = model.fit(X_train, y_train, validation_data=(X_test,y_test),
        epochs=200, batch_size=8, callbacks=[es])

    fig = pl.figure()
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()

    # plot graph and save it as a file
    pred = model.predict(X_test)

    print("Model name", modelName)
    print('Target :', target_label)
    print('Root Mean Square Error: %1.4f' % (math.sqrt(metrics.mean_squared_error(y_test,pr
    print('Mean Absolute Error: %1.4f' % (metrics.mean_absolute_error(y_test,pred)))
    print('R2 score : %1.4f' %(metrics.r2_score(y_test,pred)*100))
    print('\n\n')
```

```
    plotModelPrediction(modelName, target_label, y_train, pred, y_test, target_baseline, te

    # plot residuals qq plot
    return [math.sqrt(metrics.mean_squared_error(y_test,pred)),
                    metrics.mean_absolute_error(y_test,pred),
                    metrics.r2_score(y_test,pred)*100]
```

executed in 14ms, finished 11:19:49 2019-09-19

Entrée [32]:

```
regressors = ['Inflow lake 1 [m3]', \
           'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
           'Vol lake 1 [%]','Availability plant 1 [%]', \
           'Availability plant 2 [%]', 'Availability plant 3 [%]', \
           'Availability plant 4 [%]', 'Weekend','LaggedLakeConstraint55', 'TotalAvailableP

AddRegressors = ['none']

columns = ['Algorithm','Target','Features','RMSE','MAE','R^2', 'Parameters']
df_resultsMLP = pd.DataFrame(columns = columns)
line_nb = 0

models = ["MLP (7,5,1) Act(lin, lin, relu) Dropout (0.5,0.5)",
          "MLP (7,5,1) Act(lin, lin, relu) Dropout (0.4,0.4)",
          "MLP (7,5,1) Act(lin, lin, relu) Dropout (0.3,0.3)",
          "MLP (9,7,1) Act(lin, lin, relu) Dropout (0.5,0.5)",
          "MLP (11,7,1) Act(lin, lin, relu) Dropout (0.5,0.5)",
          "MLP (11,7,1) Act(lin, lin, relu) Dropout (0.1,0.1)",
          "MLP (11,7,3,1) Act(lin, lin, lin, relu) Dropout (0.1,0.1,0.1)"]

mlpTopologies = [[{"size":7, "activation":"linear", "dropOutAfter" : 0.5},
                  {"size":5, "activation":"linear", "dropOutAfter" : 0.5},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":7, "activation":"linear", "dropOutAfter" : 0.4},
                  {"size":5, "activation":"linear", "dropOutAfter" : 0.4},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":7, "activation":"linear", "dropOutAfter" : 0.3},
                  {"size":5, "activation":"linear", "dropOutAfter" : 0.3},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":9, "activation":"linear", "dropOutAfter" : 0.5},
                  {"size":7, "activation":"linear", "dropOutAfter" : 0.5},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":11, "activation":"linear", "dropOutAfter" : 0.5},
                  {"size":7, "activation":"linear", "dropOutAfter" : 0.5},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":11, "activation":"linear", "dropOutAfter" : 0.1},
                  {"size":7, "activation":"linear", "dropOutAfter" : 0.1},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                [{"size":11, "activation":"linear", "dropOutAfter" : 0.1},
                  {"size":7, "activation":"linear", "dropOutAfter" : 0.1},
                  {"size":3, "activation":"linear", "dropOutAfter" : 0.1},
                  {"size":1, "activation":"relu", "dropOutAfter" : 0}]]

for target_feature in ["Min prod","Max prod"]:
    for addRegressor in AddRegressors:
        fullRegressors = regressors + [addRegressor]
        xTrain, xTest, yTrain, yTest, yBaseLine = GetDataSplit(df, fullRegressors,target_fe
        #Scale the data
        stScaler = StandardScaler()
        stScaler.fit(xTrain)
        xTrainScaled = stScaler.transform(xTrain)
        xTestScaled = stScaler.transform(xTest)

        for m,mlpTopology in zip(models, mlpTopologies):
            res = TrainEvalMLP(xTrainScaled, xTestScaled, yTrain, yTest, target_feature, yB
                               addRegressor, mlpTopology, m)
            df_resultsMLP.loc[line_nb,:] = [m,target_feature, xTest.columns, res[0], res[1]
            line_nb = line_nb+1
```

executed in 8m 25s, finished 11:28:15 2019-09-19

```
Epoch 13/200
1725/1725 [==============================] - 0s 87us/step - loss: 42884.87
89 - mean_squared_error: 42884.8789 - mean_absolute_error: 148.2944 - val_
loss: 9685.8762 - val_mean_squared_error: 9685.8762 - val_mean_absolute_er
ror: 58.3798
Epoch 14/200
1725/1725 [==============================] - 0s 87us/step - loss: 37149.40
66 - mean_squared_error: 37149.4066 - mean_absolute_error: 137.8597 - val_
loss: 9958.6636 - val_mean_squared_error: 9958.6636 - val_mean_absolute_er
ror: 58.2044
Epoch 15/200
1725/1725 [==============================] - 0s 87us/step - loss: 35565.95
32 - mean_squared_error: 35565.9532 - mean_absolute_error: 136.5026 - val_
loss: 11077.0134 - val_mean_squared_error: 11077.0134 - val_mean_absolute_
error: 63.1057
Epoch 16/200
1725/1725 [==============================] - 0s 86us/step - loss: 37414.32
64 - mean_squared_error: 37414.3264 - mean_absolute_error: 137.1716 - val_
loss: 10714.2393 - val_mean_squared_error: 10714.2393 - val_mean_absolute_
error: 62.5021
```

Entrée [33]:

```
df_resultsMLP.sort_values(by=['Target','RMSE'])
```

executed in 39ms, finished 11:46:47 2019-09-19

Out[33]:

	Algorithm	Target	Features	RMSE	MAE	R^2	Parameters
12	MLP (11,7,1) Act(lin, lin, relu) Dropout (0.1,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	226.321	158.789	76.0037	[{'size': 11, 'activation': 'linear', 'dropOut...
13	MLP (11,7,3,1) Act(lin, lin, lin, relu) Dropou...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	239.123	165.847	73.212	[{'size': 11, 'activation': 'linear', 'dropOut...
9	MLP (7,5,1) Act(lin, lin, relu) Dropout (0.3,0.3)	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	254.505	174.657	69.6549	[{'size': 7, 'activation': 'linear', 'dropOutA...
8	MLP (7,5,1) Act(lin, lin, relu) Dropout (0.4,0.4)	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	271.702	182.099	65.4156	[{'size': 7, 'activation': 'linear', 'dropOutA...
10	MLP (9,7,1) Act(lin, lin, relu) Dropout (0.5,0.5)	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	271.992	174.961	65.3416	[{'size': 9, 'activation': 'linear', 'dropOutA...
11	MLP (11,7,1) Act(lin, lin, relu) Dropout (0.5,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	272.117	180.739	65.3096	[{'size': 11, 'activation': 'linear', 'dropOut...
7	MLP (7,5,1) Act(lin, lin, relu) Dropout (0.5,0.5)	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	289.294	186.71	60.7919	[{'size': 7, 'activation': 'linear', 'dropOutA...
0	MLP (7,5,1) Act(lin, lin, relu) Dropout (0.5,0.5)	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	59.1986	45.0423	81.8156	[{'size': 7, 'activation': 'linear', 'dropOutA...
1	MLP (7,5,1) Act(lin, lin, relu) Dropout (0.4,0.4)	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	67.4866	51.5356	76.3675	[{'size': 7, 'activation': 'linear', 'dropOutA...
4	MLP (11,7,1) Act(lin, lin, relu) Dropout (0.5,...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	73.7303	56.893	71.7924	[{'size': 11, 'activation': 'linear', 'dropOut...
3	MLP (9,7,1) Act(lin, lin, relu) Dropout (0.5,0.5)	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	75.3122	58.4959	70.569	[{'size': 9, 'activation': 'linear', 'dropOutA...
2	MLP (7,5,1) Act(lin, lin, relu) Dropout (0.3,0.3)	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	78.5578	60.3855	67.9776	[{'size': 7, 'activation': 'linear', 'dropOutA...
6	MLP (11,7,3,1) Act(lin, lin, lin, relu) Dropou...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	91.3767	70.3331	56.6743	[{'size': 11, 'activation': 'linear', 'dropOut...

	Algorithm	Target	Features	RMSE	MAE	R^2	Parameters
5	MLP (11,7,1) Act(lin, lin, relu) Dropout (0.1,...	Min prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	102.722	77.6024	45.2482	[{'size': 11, 'activation': 'linear', 'dropOut...

Entrée [34]:

```
# Best results for min prod on line 0: RMSE = 56.3822, R^2 = 83.50
df_resultsMLP.iloc[0,6]
```

executed in 5ms, finished 11:46:49 2019-09-19

Out[34]:

```
[{'size': 7, 'activation': 'linear', 'dropOutAfter': 0.5},
 {'size': 5, 'activation': 'linear', 'dropOutAfter': 0.5},
 {'size': 1, 'activation': 'relu', 'dropOutAfter': 0}]
```

Entrée [35]:

```
# look for better results focusing on max energy only
# at this stage, best configuration seems to be the one on line 8 above (RMSE = 246.252):
df_resultsMLP.iloc[8,6]
```

executed in 5ms, finished 11:46:50 2019-09-19

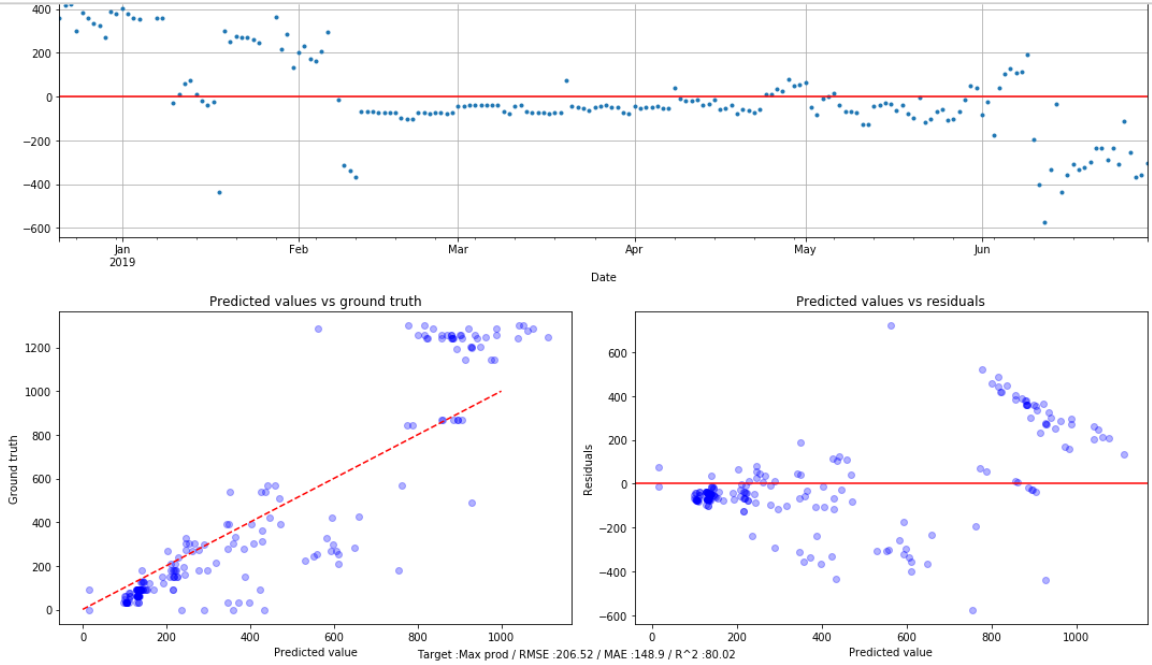
Out[35]:

```
[{'size': 7, 'activation': 'linear', 'dropOutAfter': 0.4},
 {'size': 5, 'activation': 'linear', 'dropOutAfter': 0.4},
 {'size': 1, 'activation': 'relu', 'dropOutAfter': 0}]
```

Entrée [36]:

```
regressors = ['Inflow lake 1 [m3]', \
           'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
           'Vol lake 1 [%]','Availability plant 1 [%]', \
           'Availability plant 2 [%]', 'Availability plant 3 [%]', \
           'Availability plant 4 [%]', 'Weekend','LaggedLakeConstraint55',
            'TotalAvailablePower']

AddRegressors = ['none']

columns = ['Algorithm','Target','Features','RMSE','MAE','R^2', 'Parameters']
df_resultsMLPMaxNRJ = pd.DataFrame(columns = columns)
line_nb = 0

models = ["MLP (11,5,1) Act(lin, lin, relu) Dropout (0.1,0.1)",
          "MLP (11,6,1) Act(lin, lin, relu) Dropout (0.1,0.1)",
          "MLP (11,8,1) Act(lin, lin, relu) Dropout (0.5,0.5)",
          "MLP (10,7,1) Act(lin, lin, relu) Dropout (0.1,0.1)",
          "MLP (9,7,1)  Act(lin, lin, relu) Dropout (0.1,0.1)",
          "MLP (10,7,1)  Act(lin, relu, relu) Dropout (0.1,0.1)" ]

mlpTopologies = [
                [{"size":11, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":5, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                [{"size":11, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":6, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                [{"size":11, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":8, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                [{"size":10, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":7, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                [{"size":9, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":7, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                [{"size":10, "activation":"linear", "dropOutAfter" : 0.1},
                       {"size":7, "activation":"relu", "dropOutAfter" : 0.1},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                ]

for target_feature in ["Max prod"]:
    for addRegressor in AddRegressors:
        fullRegressors = regressors + [addRegressor]
        xTrain, xTest, yTrain, yTest, yBaseLine = GetDataSplit(df, fullRegressors,target_fe
        #Scale the data
        stScaler = StandardScaler()
        stScaler.fit(xTrain)
        xTrainScaled = stScaler.transform(xTrain)
        xTestScaled = stScaler.transform(xTest)

        for m,mlpTopology in zip(models, mlpTopologies):
            res = TrainEvalMLP(xTrainScaled, xTestScaled, yTrain, yTest, target_feature, yE
                               addRegressor, mlpTopology, m)

            df_resultsMLPMaxNRJ.loc[line_nb,:] = [m,target_feature, xTest.columns, res[0],
            line_nb = line_nb+1
```

executed in 4m 9s, finished 11:51:00 2019-09-19



Entrée [37]:

```
df_resultsMLPMaxNRJ.sort_values(by='RMSE')
```

executed in 26ms, finished 12:02:40 2019-09-19

Out[37]:

	Algorithm	Target	Features	RMSE	MAE	R^2	Parameters
5	MLP (10,7,1) Act(lin, relu, relu) Dropout (0....	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	206.522	148.901	80.0184	[{'size': 10, 'activation': 'linear', 'dropOut...
1	MLP (11,6,1) Act(lin, lin, relu) Dropout (0.1,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	226.186	158.866	76.0322	[{'size': 11, 'activation': 'linear', 'dropOut...
3	MLP (10,7,1) Act(lin, lin, relu) Dropout (0.1,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	227.464	160.064	75.7605	[{'size': 10, 'activation': 'linear', 'dropOut...
4	MLP (9,7,1) Act(lin, lin, relu) Dropout (0.1,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	227.5	159.733	75.7529	[{'size': 9, 'activation': 'linear', 'dropOutA...
2	MLP (11,8,1) Act(lin, lin, relu) Dropout (0.5,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	227.548	159.119	75.7427	[{'size': 11, 'activation': 'linear', 'dropOut...
0	MLP (11,5,1) Act(lin, lin, relu) Dropout (0.1,...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	227.861	160.421	75.676	[{'size': 11, 'activation': 'linear', 'dropOut...

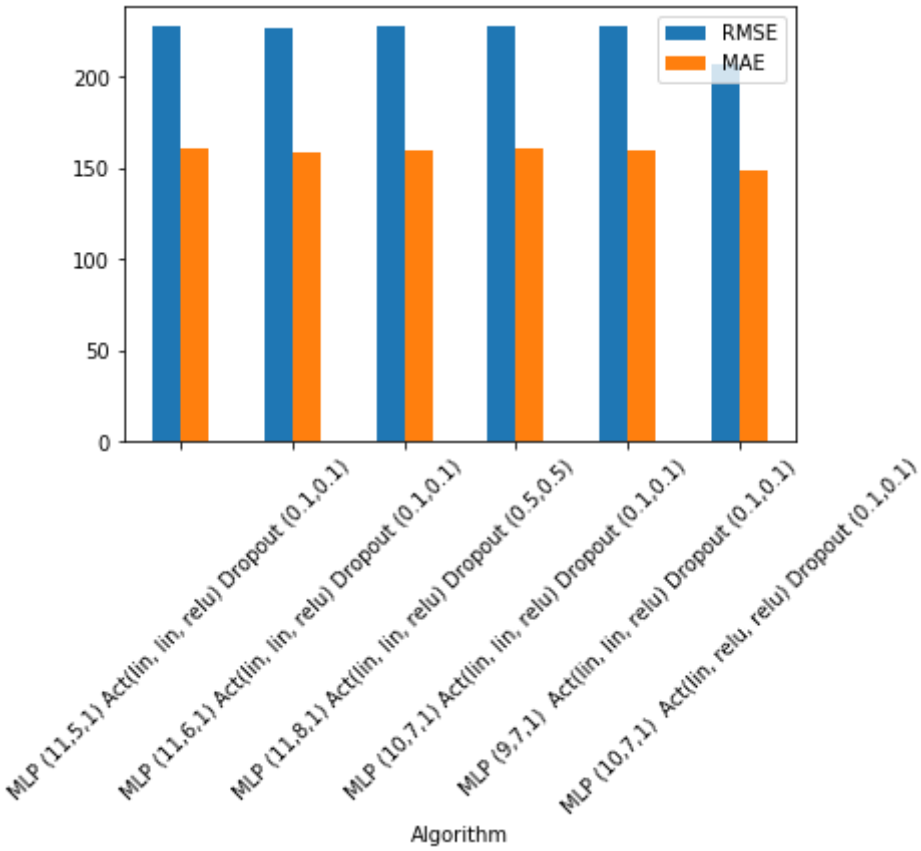
Entrée [38]:

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 6
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size

df_resultsMLPMaxNRJ.plot.bar(x='Algorithm', y=['RMSE','MAE'], rot=45)
```

executed in 207ms, finished 12:02:41 2019-09-19

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25dd2b31a20>
```



Entrée [39]:

```
# Best topology up to here:
df_resultsMLPMaxNRJ["Parameters"].iloc[5,]
```

executed in 5ms, finished 12:02:42 2019-09-19

Out[39]:

```
[{'size': 10, 'activation': 'linear', 'dropOutAfter': 0.1},
 {'size': 7, 'activation': 'relu', 'dropOutAfter': 0.1},
 {'size': 1, 'activation': 'relu', 'dropOutAfter': 0}]
```

Entrée [40]:

```
# Finally, play with drop out values
regressors = ['Inflow lake 1 [m3]', \
           'Inflow lake 2 [m3]', 'Inflow lake 3 [m3]', 'Inflow lake 4 [m3]', \
           'Vol lake 1 [%]','Availability plant 1 [%]', \
           'Availability plant 2 [%]', 'Availability plant 3 [%]', \
           'Availability plant 4 [%]', 'Weekend','LaggedLakeConstraint55',
            'TotalAvailablePower']

AddRegressors = ['none']

columns = ['Algorithm','Target','Features','RMSE','MAE','R^2', 'Parameters']
df_resultsMLPMaxNRJDropOut = pd.DataFrame(columns = columns)
line_nb = 0

models = ["MLP (10,7,1) Act(lin, relu, relu) Dropout (0.0,0.0)",
          "MLP (10,7,1) Act(lin, relu, relu) Dropout (0.2,0.2)",
          "MLP (10,7,1) Act(lin, relu, relu) Dropout (0.3,0.3)",
          "MLP (10,7,1) Act(lin, relu, relu) Dropout (0.2,0.1)",
          "MLP (10,7,1) Act(lin, relu, relu) Dropout (0.1,0.2)" ]

mlpTopologies = [
                [{"size":10, "activation":"linear", "dropOutAfter" : 0},
                       {"size":7, "activation":"relu", "dropOutAfter" : 0},
                       {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":10, "activation":"linear", "dropOutAfter" : 0.2},
                                  {"size":7, "activation":"relu", "dropOutAfter" : 0.2},
                                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":10, "activation":"linear", "dropOutAfter" : 0.3},
                                  {"size":7, "activation":"relu", "dropOutAfter" : 0.3},
                                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":10, "activation":"linear", "dropOutAfter" : 0.2},
                                  {"size":7, "activation":"relu", "dropOutAfter" : 0.1},
                                  {"size":1, "activation":"relu", "dropOutAfter" : 0}],
                 [{"size":10, "activation":"linear", "dropOutAfter" : 0.1},
                                  {"size":7, "activation":"relu", "dropOutAfter" : 0.2},
                                  {"size":1, "activation":"relu", "dropOutAfter" : 0}]

                ]

for target_feature in ["Max prod"]:
    for addRegressor in AddRegressors:
        fullRegressors = regressors + [addRegressor]
        xTrain, xTest, yTrain, yTest, yBaseLine = GetDataSplit(df, fullRegressors,target_fe
        #Scale the data
        stScaler = StandardScaler()
        stScaler.fit(xTrain)
        xTrainScaled = stScaler.transform(xTrain)
        xTestScaled = stScaler.transform(xTest)

        for m,mlpTopology in zip(models, mlpTopologies):
            res = TrainEvalMLP(xTrainScaled, xTestScaled, yTrain, yTest, target_feature, yE
                               addRegressor, mlpTopology, m, True)

            df_resultsMLPMaxNRJDropOut.loc[line_nb,:] = [m,target_feature, xTest.columns, r
            line_nb = line_nb+1
```

executed in 3m 59s, finished 12:06:42 2019-09-19

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_66 (Dense)             (None, 10)                120
_________________________________________________________________
dense_67 (Dense)             (None, 7)                 77
_________________________________________________________________
dense_68 (Dense)             (None, 1)                 8
=================================================================
Total params: 205
Trainable params: 205
Non-trainable params: 0
_________________________________________________________________
Train on 1725 samples, validate on 192 samples
Epoch 1/200
1725/1725 [==============================] - 1s 799us/step - loss: 662559.
6256 - mean_squared_error: 662559.6256 - mean_absolute_error: 725.9841 - v
al_loss: 384408.9842 - val_mean_squared_error: 384408.9842 - val_mean_abso
lute error: 414.7146
```

Entrée [41]:

```
df_resultsMLPMaxNRJDropOut.sort_values(by='RMSE')
```

executed in 16ms, finished 12:07:27 2019-09-19

Out[41]:

	Algorithm	Target	Features	RMSE	MAE	R^2	Parameters
0	MLP (10,7,1) Act(lin, relu, relu) Dropout (0.0...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	178.909	120.933	85.0045	[{'size': 10, 'activation': 'linear', 'dropOut...
4	MLP (10,7,1) Act(lin, relu, relu) Dropout (0.1...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	205.232	139.215	80.2672	[{'size': 10, 'activation': 'linear', 'dropOut...
3	MLP (10,7,1) Act(lin, relu, relu) Dropout (0.2...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	230.695	157.73	75.0671	[{'size': 10, 'activation': 'linear', 'dropOut...
2	MLP (10,7,1) Act(lin, relu, relu) Dropout (0.3...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	238.796	175.551	73.2852	[{'size': 10, 'activation': 'linear', 'dropOut...
1	MLP (10,7,1) Act(lin, relu, relu) Dropout (0.2...	Max prod	Index(['Inflow lake 1 [m3]', 'Inflow lake 2 [m...	240.073	185.357	72.9989	[{'size': 10, 'activation': 'linear', 'dropOut...

Entrée [42]:

```
# at this stage, best configuration seems to be the one on line 0 above (RMSE = 192.842, R2
df_resultsMLPMaxNRJDropOut.iloc[0,0]
```

executed in 4ms, finished 12:07:28 2019-09-19

Out[42]:

```
'MLP (10,7,1) Act(lin, relu, relu) Dropout (0.0,0.0)'
```

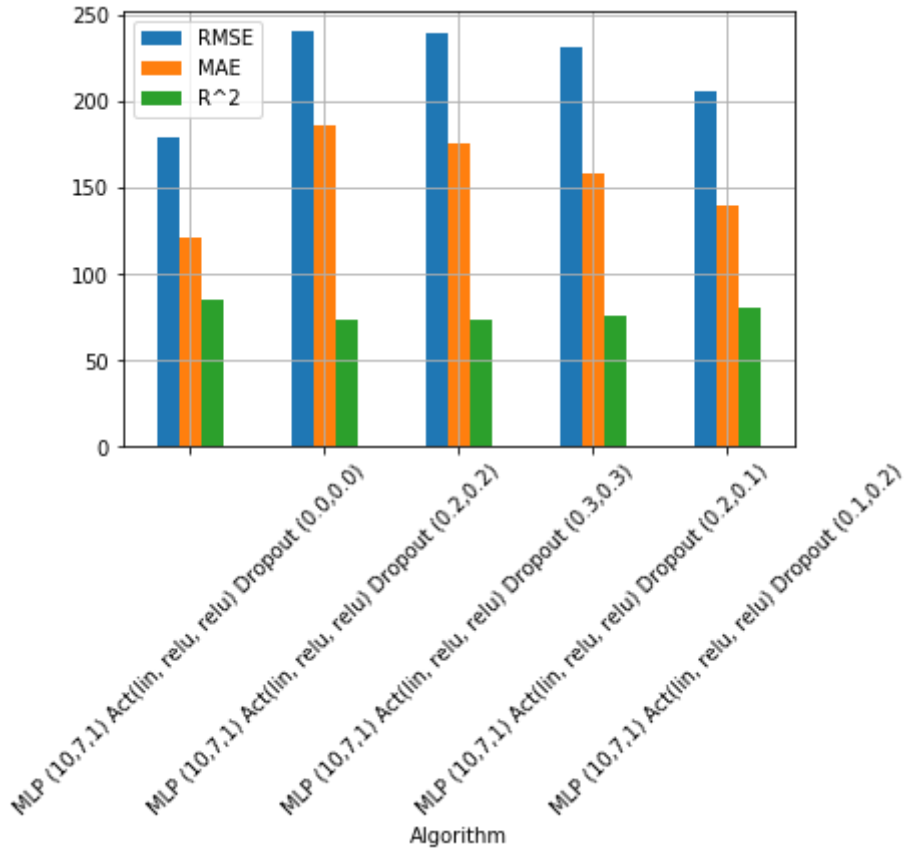
Entrée [43]:

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 6
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size

df_resultsMLPMaxNRJDropOut.plot.bar(x='Algorithm', y=['RMSE','MAE','R^2'], rot=45, grid=Tru
```

executed in 218ms, finished 12:07:28 2019-09-19

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25dd31a12e8>
```



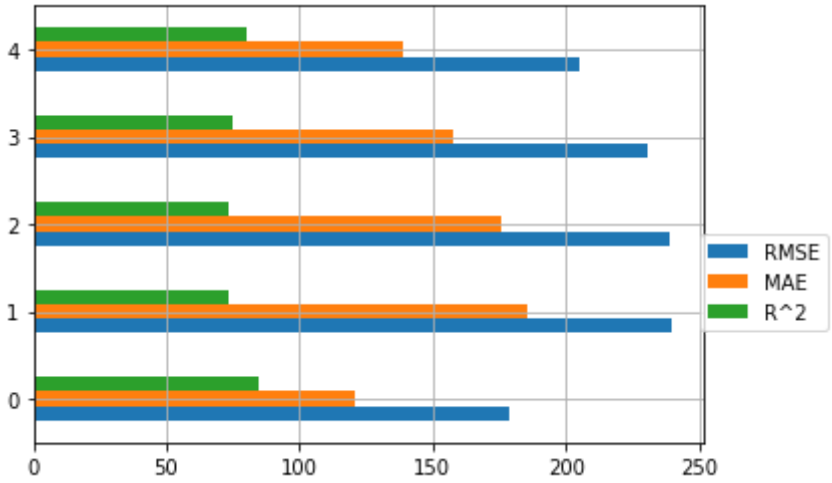
Entrée [44]:

```
df_resultsMLPMaxNRJDropOut[['RMSE','MAE','R^2']].plot(kind="barh", grid=True).legend(bbox_t
```

executed in 178ms, finished 12:07:29 2019-09-19

Out[44]:

```
<matplotlib.legend.Legend at 0x25dd29ad940>
```



Entrée [45]:

```
# At this stage we have identified our best predicting model
```

executed in 3ms, finished 12:07:30 2019-09-19

Entrée [ ]:

```

```