Configuration Manual

MSc Research Project

Data Analytics

Gabriel Dada Ibukun

Student ID: x18176585

School of Computing

National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Gabriel Dada
Student ID:	x18176585
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	12th December 2019
Project Title:	Configuration Manual
Word Count:	
Page Count:	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Gabriel Dada (x18176585)

1 Introduction

This configuration manual provides detailed documentation of the implementation of I.T solution deployed as part of the research thesis in Electric Load Forecasts using Machine Learning and Distributed Systems. The scope covers all steps taken to for solution deployment. The systems configuration requirements are as follows:

- Processor: intel core i5 1.8Ghz DDR3
- RAM: 8GB
- System: x64 processor

2 Integrated Development Environment

The project implementation was deployed in the Anaconda 2019.10 for mac operating system (with 64 bit graphic installer) environment. Python 3.7 accompanies it as both can be downloaded from here. Having installed Anaconda, Jupyter notebook was used for the data for data pre-processing, transformation, feature engineering and modeling.

3 Datasets

Datasets used for this project were downloaded as csv files in two categories namely electric load data and weather data. The load data was originally sourced from PJM open source repository online here. The historical hourly weather datasets were sourced directly from Kaggle containing weather measures of temperature, pressure, humidity, wind direction, and wind speed for 30 US cities here.

4 Assessing the datasets

The datasets were first loaded into R studio for preliminary checks after which all 6 datasets were loaded to the Jupyter python environment. First, all necessary libraries required for our analysis were loaded into python (even though Jupyter notebook has some of these libraries pre-installed). This is shown here:

```
In [1]: import numpy as nmp
        import pandas as pnd
        import seaborn as sns
        import matplotlib.pyplot as mplot
        import xgboost
        from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor, ExtraTreesRegressor
        from xgboost import plot_importance, plot_tree
        from sklearn.feature_selection import SelectFromModel
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score, KFold, GridSearchCV, TimeSeriesSplit
        from sklearn.metrics import mean_squared_error, mean_absolute_error
        import warnings
        warnings.filterwarnings('ignore')
        mplot.style.use('ggplot')
```

After which the datasets are loaded accordingly to Jupyter notebook.

```
In [3]: %%time
        #Load Power Datasets
        power_dom = pnd.read_csv('DOM_hourly.csv', parse_dates=[0], squeeze=True, index_col=[0])
        power_dom = power_dom.loc[~power_dom.index.duplicated(keep='first')].sort_index().dropna()

        CPU times: user 38.3 s, sys: 1.78 s, total: 40.1 s
        Wall time: 2min 23s
```

```
In [5]: %%time
        #Load Weather Datasets
        #Humidity
        humidity = pnd.read_csv('humidity.csv', parse_dates=[0],squeeze=True, index_col=[0])
        humidity = humidity.loc[~humidity.index.duplicated(keep='first')].sort_index().dropna()


        # Pressure
        pressure = pnd.read_csv('pressure.csv', parse_dates=[0],squeeze=True, index_col=[0])
        pressure = pressure.loc[~pressure.index.duplicated(keep='first')].sort_index().dropna()


        # Temperature
        temperature = pnd.read_csv('temperature.csv', parse_dates=[0], squeeze=True, index_col=[0])
        temperature = temperature.loc[~temperature.index.duplicated(keep='first')].sort_index().dropna()


        #Wind Direction
        wind_direction = pnd.read_csv('wind_direction.csv', parse_dates=[0], squeeze=True, index_col=[0])
        wind_direction = wind_direction.loc[~wind_direction.index.duplicated(keep='first')].sort_index().dropna()


        #Wind Speed
        wind_speed = pnd.read_csv('wind_speed.csv', parse_dates=[0], squeeze=True, index_col=[0])
        wind_speed = wind_speed.loc[~wind_speed.index.duplicated(keep='first')].sort_index().dropna()

        CPU times: user 38 s, sys: 898 ms, total: 38.9 s
        Wall time: 50.9 s
```

5 Concatenation to create final project dataset

Since our analysis hinges on a single dataset that will be use electric load consumption as the dependent, and weather features such as temperature, pressure, etc as independent variables. The task will be to concatenate the various times series joining them by the date-time column common to all 6 csv files. Using the pandas library, first we deal with those of weather:

```
In [12]: %%time
         #Concatenate weather Data
         weather_data = pnd.concat([temperature, humidity, pressure, wind_direction, wind_speed], axis=1).sort_index()

         CPU times: user 27.3 ms, sys: 18 ms, total: 45.2 ms
         Wall time: 106 ms
```

And then add power:

```
In [15]: %%time
project_data = pnd.concat([power_dom.loc[weather_data.index[0]:weather_data.index[-1]], weather_data], axis=1).sort_inc
project_data.head()
```

```
CPU times: user 57 ms, sys: 29.2 ms, total: 86.2 ms
Wall time: 174 ms
```

The final concatenated output looks like this:

Out[15]:

	POWER_MW	temperature	humidity	pressure	wind_direction	wind_speed	pressure_log
2012-10-01 13:00:00	9819.0	288.650000	87.0	1012.0	70.0	4.0	6.919684
2012-10-01 14:00:00	9845.0	288.650172	87.0	1012.0	70.0	4.0	6.919684
2012-10-01 15:00:00	9867.0	288.650582	87.0	1012.0	71.0	4.0	6.919684
2012-10-01 16:00:00	9857.0	288.650991	87.0	1012.0	71.0	4.0	6.919684
2012-10-01 17:00:00	9861.0	288.651401	87.0	1012.0	72.0	4.0	6.919684

6 Feature Engineering

To further prepare the time series data for modeling, date-time features were expanded, also lag features created with this block of code. First date time features:

```
%%time
#Time Series Feature
project_final = (project_data.assign( day_of_week = project_data.index.dayofweek
                            ,year = project_data.index.year
                            ,month = project_data.index.month
                            ,day = project_data.index.day
                            ,day_of_year = project_data.index.dayofyear

                            ,week = project_data.index.week
                            ,week_day = project_data.index.weekday_name
                            ,quarter = project_data.index.quarter
                            ,hour = project_data.index.hour
                            ,hour_x = nmp.sin(2.*nmp.pi*project_data.index.hour/24.)
                            ,hour_y = nmp.cos(2*nmp.pi*project_data.index.hour/24.)
                            ,day_of_year_x = nmp.sin(2.*nmp.pi*project_data.index.dayofyear/365.)
                            ,day_of_year_y = nmp.cos(2.*nmp.pi*project_data.index.dayofyear/365.)

                          )
           )
```

```
CPU times: user 170 ms, sys: 60.4 ms, total: 230 ms
Wall time: 352 ms
```

And then lag features with the below configuration:

```
%%time
#Adding Lagging Feature

lagged_df = project_final.copy()
lagged_df['load_tomorrow'] = lagged_df['POWER_MW'].shift(-24)
for day in range(8):
    lagged_df['temperature_d' + str(day)] = lagged_df.temperature.shift(24*day)
    lagged_df['wind_speed_d' + str(day)] = lagged_df.wind_speed.shift(24*day)
    lagged_df['humidity_d' + str(day)] = lagged_df.humidity.shift(24*day)
    lagged_df['pressure_log_d' + str(day)] = lagged_df.pressure_log.shift(24*day)


    lagged_df['load_d' + str(day)] = lagged_df.POWER_MW.shift(24*day)

lagged_df = lagged_df.dropna()
lagged_df = lagged_df.drop(columns=['temperature', 'wind_speed', 'humidity', 'pressure', 'wind_direction', 'week_day','
```

This process increased the number of features to 54 in total The output file is shown:

```
lagged_df.head()
```

	pressure_log	day_of_week	year	month	day	day_of_year	week	quarter	hour	hour_x	...	temperature_d6	wind_speed_d6	humidity_d6	pressure
2012-10-08 14:00:00	6.925595	0	2012	10	8	282	41	4	14	-0.500000	...	288.65959	4.0	87.0	6
2012-10-08 15:00:00	6.925595	0	2012	10	8	282	41	4	15	-0.707107	...	288.66000	4.0	87.0	6
2012-10-08 16:00:00	6.925595	0	2012	10	8	282	41	4	16	-0.866025	...	288.60000	4.0	82.0	6
2012-10-08 17:00:00	6.925595	0	2012	10	8	282	41	4	17	-0.965926	...	288.65000	0.0	93.0	6
2012-10-08 18:00:00	6.924612	0	2012	10	8	282	41	4	18	-1.000000	...	288.65000	2.0	100.0	6

5 rows × 54 columns

7 Feature Selection

Feature selection was achieved using ranking the contribution of all features in our model using the F-score. A plot of feature importance from an initial Xgboost regression model was used as a basis. The input blocks of codes and out are outline below.

```
In [63]: #Feature Selection
         X = lagged_df.drop(columns=['load_tomorrow'])
         y = lagged_df['load_tomorrow']

In [64]: X.shape
Out[64]: (44463, 53)

In [65]: y.shape
Out[65]: (44463,)

In [66]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)
```

```
In [67]: def plot_prediction(actual, prediction, start_date, end_date, title, prediction_label):
    mplot.figure(figsize=(20,5))
    mplot.title(title)
    mplot.plot(y_test.index, y_test, label='Actual')
    mplot.plot(y_test.index, prediction, label=prediction_label)
    mplot.ylabel('Power(MW)')
    mplot.xlabel('Datetime')
    mplot.legend()
    mplot.xlim(left= start_date, right=end_date)
    mplot.show()

def subplot_prediction(actual, prediction,prediction_label):
    fig, axes = mplot.subplots(nrows=3, ncols=1, figsize=(10, 12))

    con_df = pnd.concat([actual.rename('Actual'),pnd.DataFrame(prediction, index=actual.index, columns=[prediction_lab
    axes[0].set_title('Actual vs Prediction - Day ahead')
    axes[0].set_ylabel('Power(MW)')
    axes[0].set_xlabel('Datetime')
    con_df.plot(ax=axes[0])
    axes[0].set_xlim(left=con_df.index[-24*1] , right=con_df.index[-1])

    axes[1].set_title('Actual vs Prediction - Week ahead')
    axes[1].set_ylabel('Power(MW)')
    axes[1].set_xlabel('Datetime')
    con_df.plot(ax=axes[1])
    axes[1].set_xlim(left=actual.index[-24*7] , right=actual.index[-1])

    axes[2].set_title('Actual vs Prediction - month ahead')
    axes[2].set_ylabel('Power(MW)')
    axes[2].set_xlabel('Datetime')
    con_df.plot(ax=axes[2])
    axes[2].set_xlim(left=actual.index[-24*7*4] , right=actual.index[-1])

    mplot.tight_layout()
    mplot.show()
```

```
    mplot.tight_layout()
    mplot.show()

def plot_feature_importances( clf, X_train, y_train=None
                         ,top_n=10, figsize=(10,18), print_table=False, title="Feature Importances"):
    feat_imp = pnd.DataFrame({'importance':clf.feature_importances_})
    feat_imp['feature'] = X_train.columns
    feat_imp.sort_values(by='importance', ascending=False, inplace=True)
    feat_imp = feat_imp.iloc[:top_n]

    feat_imp.sort_values(by='importance', inplace=True)
    feat_imp = feat_imp.set_index('feature', drop=True)
    feat_imp.plot.barh(title=title, figsize=figsize)
    mplot.xlabel('Feature Importance Score')
    mplot.show()

    if print_table:
        from IPython.display import display
        print("Top {} features in descending order of importance".format(top_n))
        display(feat_imp.sort_values(by='importance', ascending=False))

    return feat_imp
```

```
In [68]: regression = xgboost.XGBRegressor()
```

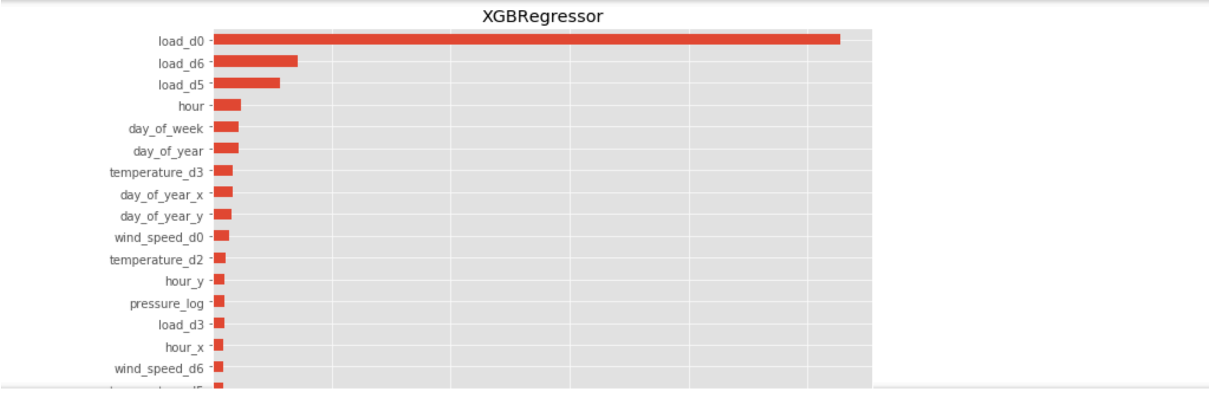
```
In [69]: tscv = TimeSeriesSplit(n_splits=5)
scores = cross_val_score(regression, X.values, y.values, cv=tscv
                         ,scoring='explained_variance'
                        )
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() ))
print(scores)
```

```
[00:43:54] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:43:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:44:06] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:44:15] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:44:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Accuracy: 0.78 (+/- 0.04)
[0.71688917 0.74830455 0.83257863 0.76274014 0.81890592]
```

```
In [70]: regression.fit(X_train,y_train)
prediction = regression.predict(X_test)
```

```
[00:44:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
ı [71]: = plot_feature_importances(regression, X_train, y_train, top_n=X_train.shape[1], title=regression.__class__.__name
```



8 Modeling

Here, the machine learning models were implemented and necessary evaluation metric obtained. Xgboost, Extra Trees regressor, SARIMA and ARIMA were applied to the different lengths of

8.1 Modeling: XGBoost

```
In [78]: #DATA LENGHT: 6 YEARS

In [79]:
X = project_final.drop(columns = ['POWER_MW','week_day'])
y = project_final['POWER_MW']

In [80]: X.shape

Out[80]: (44655, 18)

In [81]: y.shape

Out[81]: (44655,)

In [82]: %%time
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)

CPU times: user 11.2 ms, sys: 9.32 ms, total: 20.5 ms
Wall time: 28.9 ms

In [83]: %%time
regression = xgboost.XGBRegressor()

CPU times: user 1.3 ms, sys: 6.14 ms, total: 7.44 ms
Wall time: 5.05 ms
```

In [84]:
```
%%time
#MODEL VALIDATION USING K-FOLD CROSS VALIDATION SCORE
tscv = TimeSeriesSplit(n_splits=10)
scores = cross_val_score(regression, X.values, y.values, cv=tscv
                         ,scoring='explained_variance'
                        )
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() ))
print(scores)
```

```
[00:45:00] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:05] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:07] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:11] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:16] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[00:45:29] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Accuracy: 0.73 (+/- 0.21)
[0.14180571 0.62351797 0.81927497 0.81779954 0.82524587 0.84653834
 0.81897553 0.80287451 0.72909061 0.84004309]
CPU times: user 27.7 s, sys: 542 ms, total: 28.2 s
Wall time: 33.2 s
```

In [85]:
```
%%time
regression.fit(X_train,y_train)
prediction = regression.predict(X_test)
```

```
[00:45:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
CPU times: user 3.78 s, sys: 87.8 ms, total: 3.87 s
Wall time: 4.28 s
```

In [86]:
```
%%time
#RMSE
rmse = nmp.sqrt(mean_squared_error(y_test, prediction))
norm_rmse = rmse/nmp.std(y_test)
print("RMSE: %f" % (norm_rmse))
```

```
RMSE: 0.492036
CPU times: user 1.42 ms, sys: 744 µs, total: 2.16 ms
Wall time: 1.8 ms
```

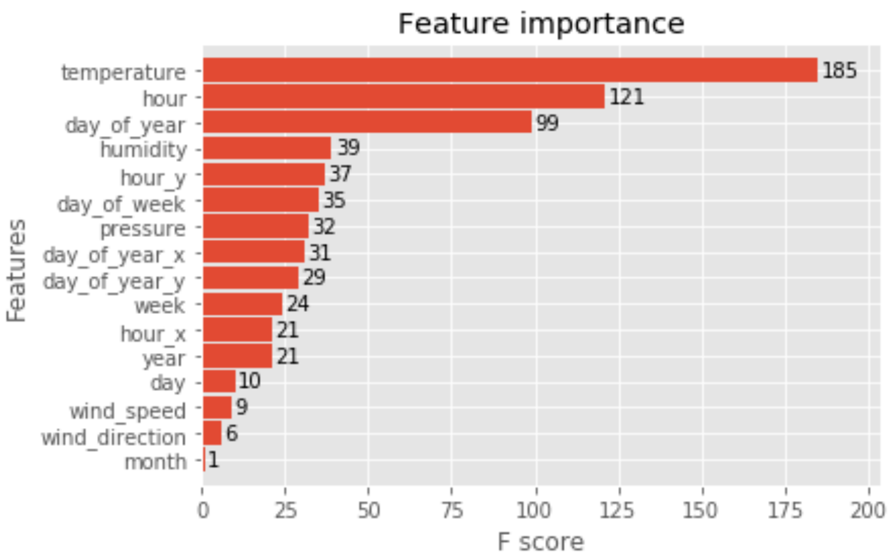
In [87]:
```
#MAPE


def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = nmp.array(y_true), nmp.array(y_pred)
    return nmp.mean(nmp.abs((y_true - y_pred) / y_true)) * 100


mean_absolute_percentage_error(y_test,prediction)
```

Out[87]: 7.660149964965591

In [89]:
```
_ = plot_importance(regression, height=0.9)
```



8.2 Modeling: Exratrees Regressor

In [132]:
```
X = project_final.drop(columns = ['POWER_MW','week_day'])
y = project_final['POWER_MW']
```

In [133]:
```
X.shape
```

Out[133]:
```
(44655, 18)
```

In [134]:
```
y.shape
```

Out[134]:
```
(44655,)
```

In [135]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)
```

In [136]:
```
regression = ExtraTreesRegressor()
```

In [137]:
```
%%time
#MODEL VALIDATION USING K-FOLD CROSS VALIDATION SCORE
tscv = TimeSeriesSplit(n_splits=10)
scores = cross_val_score(regression, X.values, y.values, cv=tscv
                         ,scoring='explained_variance'
                        )
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() ))
print(scores)
```

```
Accuracy: 0.71 (+/- 0.19)
[0.21945934 0.53197164 0.71098586 0.79369168 0.82068105 0.85682173
 0.81769279 0.81277603 0.72358419 0.81744909]
CPU times: user 12.2 s, sys: 572 ms, total: 12.8 s
Wall time: 14.6 s
```


```
CPU times: user 12.2 s, sys: 572 ms, total: 12.8 s
Wall time: 14.6 s
```


In [138]:
```
%%time
regression.fit(X_train,y_train)
prediction = regression.predict(X_test)
```

```
CPU times: user 1.8 s, sys: 71.4 ms, total: 1.87 s
Wall time: 2.29 s
```

In [139]:
```
%%time
#RMSE
rmse = nmp.sqrt(mean_squared_error(y_test, prediction))
norm_rmse = rmse/nmp.std(y_test)
print("RMSE: %f" % (norm_rmse))
```

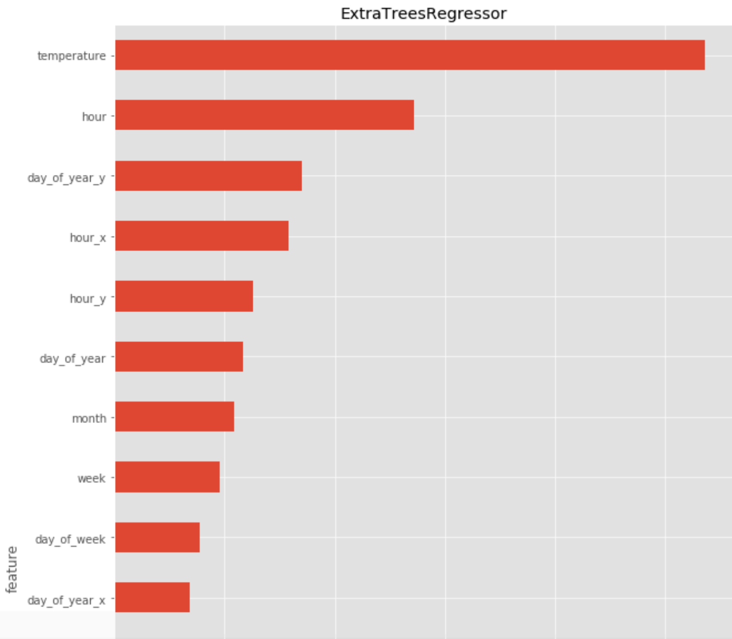
```
RMSE: 0.460499
CPU times: user 2.25 ms, sys: 1.57 ms, total: 3.82 ms
Wall time: 4.43 ms
```

In [140]:
```
%%time
#MAPE
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = nmp.array(y_true), nmp.array(y_pred)
    return nmp.mean(nmp.abs((y_true - y_pred) / y_true)) * 100
mean_absolute_percentage_error(y_test,prediction)
```

```
CPU times: user 942 µs, sys: 432 µs, total: 1.37 ms
Wall time: 1.11 ms
```

Out[140]:
```
7.026220323155585
```

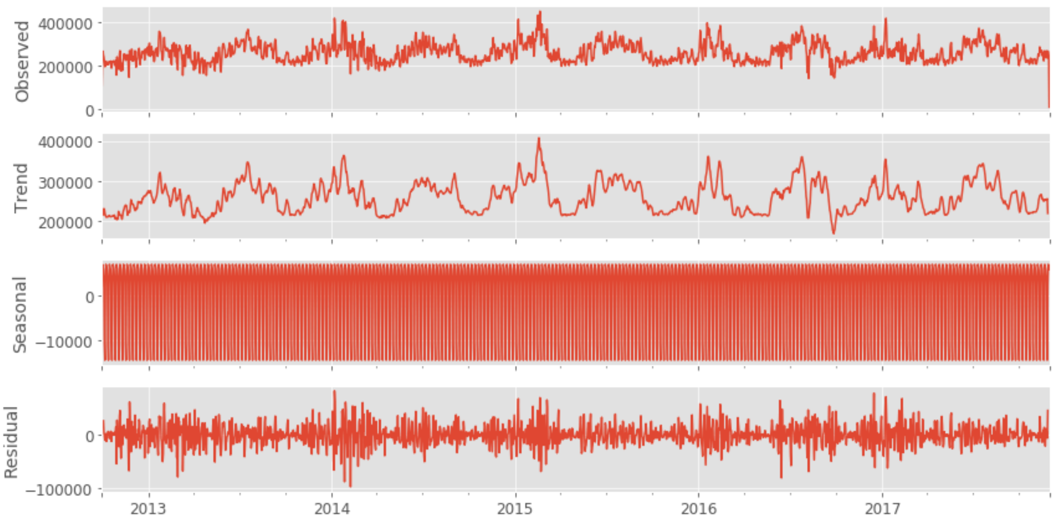
```
In [142]: _ = plot_feature_importances(regression, X_train, y_train, top_n=X_train.shape[1], title=regression.__class__.__name__)
```



8.3 Modeling: SARIMA

First the time series decompose plot is used to split the time series into its trend, seasonal, and residual elements using this block of codes:

```
In [187]: %%time
#Using sm.tsa.seasonal_decompose to show trend, seasonality and noise
from pylab import rcParams
rcParams['figure.figsize'] = 12, 6
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
mplot.show()
```




```
CPU times: user 2.03 s, sys: 49.1 ms, total: 2.08 s
Wall time: 2.32 s
```

Next is to find the Optimum (p,d,q)(P, D, Q)m parameters using grid search iteration

In [188]:
```
%%time
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter for SARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
```

```
Examples of parameter for SARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
CPU times: user 1.17 ms, sys: 951 µs, total: 2.12 ms
Wall time: 1.4 ms
```

In [189]:
```
%%time
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,order=param,seasonal_order=param_seasonal,enforce_stationarity=False,enfo:
            results = mod.fit()
            print('SARIMA{}x{}12 - AIC:{}'.format(param,param_seasonal,results.aic))
        except:
            continue
```

```
SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:52443.69758062365
SARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:50890.07752954849
SARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:45783.94872155711
SARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:45106.66787527268
SARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:45798.145538940866
SARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:45424.86828650036
SARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:45183.1204103619
```

The combination of parameters with the lowest AIC score is selected and used for the forecast:

In [190]:
```
%%time
#use it for the forecast

from matplotlib import pyplot
from pylab import rcParams
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
from pandas import read_csv
from pandas import datetime
from math import sqrt
import warnings
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import Normalizer

X = project_final.POWER_MW.resample('D').sum().values
size = int(len(X) * 0.70)
train, test = X[0:size], X[size:]
# normalizer =Normalizer().fit(train)
# train = normalizer.transform(train)
history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = SARIMAX(history, order=(1,1,1),seasonal_order=(0,0,1,12),enforce_stationarity=False,
                    enforce_invertibility=False)
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
rmse = sqrt(mean_squared_error(test, predictions))

norm_rmse = rmse/nmp.std(test)
print("RMSE: %f" % (norm_rmse))
```

```
    obs = test[t]
    history.append(obs)
rmse = sqrt(mean_squared_error(test, predictions))

norm_rmse = rmse/nmp.std(test)
print("RMSE: %f" % (norm_rmse))

#MAPE
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = nmp.array(y_true), nmp.array(y_pred)
    return nmp.mean(nmp.abs((y_true - y_pred) / y_true)) * 100


print("MAPE: %f" % (mean_absolute_percentage_error(test,predictions)))


#MAE
norm_mae = (mean_absolute_error(test,predictions)) / nmp.std(test)
print("MAE: %f" % norm_mae)


# plot forecasts against actual outcomes
pyplot.rcParams['figure.figsize'] = 10, 6
pyplot.title(label='ARIMA prdeiction - 6 Years')
pyplot.plot(test)
pyplot.plot(predictions, color='blue')
pyplot.show()
```

```
RMSE: 0.573660
MAPE: 11.278496
MAE: 0.400242
```

8.5 Modeling: ARIMA

Similar procedure is carried out to grid search for best (p,d,q) parameters for ARIMA as shown below:

```
import warnings

def evaluate_arima_model(X, arima_order):
    train_size = int(len(X) * 0.70 )
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    predictions = list()
    for t in range(len(test)):
        model= ARIMA(history, order=arima_order)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    rmse = sqrt(mean_squared_error(test, predictions))
    return rmse

def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    rmse = evaluate_arima_model(dataset, order)
                    if rmse < best_score:
                        best_score, best_cfg = rmse, order
                    print('ARIMA%s RMSE=%.3f' % (order, rmse))
                except:
                    continue
    print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))


p_values = [0, 1, 2, 4, 6, 8, 10]
d_values = range(0,3)
q_values = range(0,3)

warnings.filterwarnings("ignore")
evaluate_models(y_values, p_values, d_values, q_values)
```

Then used for forecasts:

```
In [ ]: #use it for the forecast

        from matplotlib import pyplot
        from pylab import rcParams
        from statsmodels.tsa.arima_model import ARIMA
        from statsmodels.tsa.statespace.sarimax import SARIMAX
        from sklearn.metrics import mean_squared_error
        from pandas import read_csv
        from pandas import datetime
        from math import sqrt
        import warnings

        X = y.values
        size = int(len(X) * 0.70)
        train, test = X[0:size], X[size:]
        history = [x for x in train]
        predictions = list()

        for t in range(len(test)):
            model = ARIMA(history, order=(0,0,1))
            model_fit = model.fit(disp=0)
            output = model_fit.forecast()
            yhat = output[0]
            predictions.append(yhat)
            obs = test[t]
            history.append(obs)
        rmse = sqrt(mean_squared_error(test, predictions))
        norm_rmse = rmse/nmp.std(test)
        print("RMSE: %f" % (norm_rmse))

        #MAPE
        def mean_absolute_percentage_error(y_true, y_pred):
            y_true, y_pred = nmp.array(y_true), nmp.array(y_pred)
            return nmp.mean(nmp.abs((y_true - y_pred) / y_true)) * 100


        print("MAPE: %f" % (mean_absolute_percentage_error(test,predictions)))
```

References

[1] Hong T and Shahidehpour, M 2015. *Load Forecasting Case Study for the Eastern Interconnection States' Planning Council (EISPC) in response to the NARUC solicitation NARUC-2014-RFP042–DE0316*. University of North Carolina at Charlotte (UNCC) teamed with Illinois Institute of Technology (IIT), ISO-New England, and North Carolina Electric Membership Corporation (NCEMC). The work was supported by the Department of Energy, National Energy Technology Laboratory, under Award Number DE-OE0000316

[2] Brownlee J. Machine learning mastery: Introduction to Time Series Forecasting with Python. How to prepare data and develop models to predict the future.