```
import numpy as np
import pandas as pd
import seaborn as sns
```

```
dataset = pd.read_csv("energy.csv")
dataset.head()
```

	TimeStamp1	X1	X2	X3	X4	X5	X6	X7	
0	06-11-2020 13:06	14976.340	5.215638	242.111923	9.171348	50.101936	0.792295	4.016511	6.
1	06-11-2020 13:12	14976.897	5.207636	241.313522	9.160061	50.043190	0.798129	3.931103	6.
2	06-11-2020 13:18	14977.453	5.224983	241.313767	9.156878	50.062305	0.795530	3.979594	6.
3	06-11-2020 13:25	14978.028	5.406835	242.118484	9.348110	50.021931	0.804664	3.989499	6.
4	06-11-2020 13:31	14978.606	5.425514	240.884155	9.333550	50.038364	0.810980	3.914182	6.

5 rows × 58 columns

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
dataset.dtypes
```

```
TimeStamp1     object
X1            float64
X2            float64
X3            float64
X4            float64
X5            float64
X6            float64
X7            float64
X8            float64
X9            float64
X10           float64
X11           float64
X12           float64
X13           float64
X14           float64
X15           float64
X16           float64
X17           float64
```

```
X18        float64
X19        float64
X20        float64
X21        float64
X22        float64
X23        float64
X24        float64
X25        float64
X26        float64
X27        float64
X28        float64
X29        float64
X30          int64
X31          int64
X32        float64
X33        float64
X34        float64
X35        float64
X36        float64
X37        float64
X38          int64
X39          int64
X40          int64
X41          int64
X42          int64
X43          int64
X44        float64
X45        float64
X46        float64
X47        float64
X48        float64
X49        float64
X50          int64
X51          int64
X52        float64
X53          int64
X54        float64
X55        float64
X56        float64
X57        float64
```

```
dataset.describe()
```

	X1	X2	X3	X4	X5	X6	
count	4206.000000	4206.000000	4206.000000	4206.000000	4206.000000	4206.000000	420
mean	16307.234829	4.737531	239.668974	8.644678	50.001566	0.771180	3
std	765.784039	0.358424	2.406327	0.363676	0.055718	0.029395	(

```
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
from  scipy.stats import skew, kurtosis, shapiro
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4206 entries, 0 to 4205
Data columns (total 58 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   TimeStamp1  4206 non-null   object 
 1   X1          4206 non-null   float64
 2   X2          4206 non-null   float64
 3   X3          4206 non-null   float64
 4   X4          4206 non-null   float64
 5   X5          4206 non-null   float64
 6   X6          4206 non-null   float64
 7   X7          4206 non-null   float64
 8   X8          4206 non-null   float64
 9   X9          4206 non-null   float64
 10  X10         4206 non-null   float64
 11  X11         4206 non-null   float64
 12  X12         4206 non-null   float64
 13  X13         4206 non-null   float64
 14  X14         4206 non-null   float64
 15  X15         4206 non-null   float64
 16  X16         4206 non-null   float64
 17  X17         4206 non-null   float64
 18  X18         4206 non-null   float64
 19  X19         4206 non-null   float64
 20  X20         4206 non-null   float64
 21  X21         4206 non-null   float64
 22  X22         4206 non-null   float64
 23  X23         4206 non-null   float64
 24  X24         4206 non-null   float64
 25  X25         4206 non-null   float64
 26  X26         4206 non-null   float64
 27  X27         4206 non-null   float64
 28  X28         4206 non-null   float64
 29  X29         4206 non-null   float64
 30  X30         4206 non-null   int64  
 31  X31         4206 non-null   int64  
 32  X32         4206 non-null   float64
 33  X33         4206 non-null   float64
 34  X34         4206 non-null   float64
```

```
 35  X35         4206 non-null   float64
 36  X36         4206 non-null   float64
 37  X37         4206 non-null   float64
 38  X38         4206 non-null   int64
 39  X39         4206 non-null   int64
 40  X40         4206 non-null   int64
 41  X41         4206 non-null   int64
 42  X42         4206 non-null   int64
 43  X43         4206 non-null   int64
 44  X44         4206 non-null   float64
 45  X45         4206 non-null   float64
 46  X46         4206 non-null   float64
 47  X47         4206 non-null   float64
 48  X48         4206 non-null   float64
 49  X49         4206 non-null   float64
 50  X50         4206 non-null   int64
 51  X51         4206 non-null   int64
```

```
X = dataset.iloc[:,1:57]
```

```
X.head()
```

	X1	X2	X3	X4	X5	X6	X7	X8	
0	14976.340	5.215638	242.111923	9.171348	50.101936	0.792295	4.016511	6.582951	2.67
1	14976.897	5.207636	241.313522	9.160061	50.043190	0.798129	3.931103	6.524802	2.65
2	14977.453	5.224983	241.313767	9.156878	50.062305	0.795530	3.979594	6.567923	2.67
3	14978.028	5.406835	242.118484	9.348110	50.021931	0.804664	3.989499	6.719373	2.72
4	14978.606	5.425514	240.884155	9.333550	50.038364	0.810980	3.914182	6.690069	2.70

5 rows × 56 columns

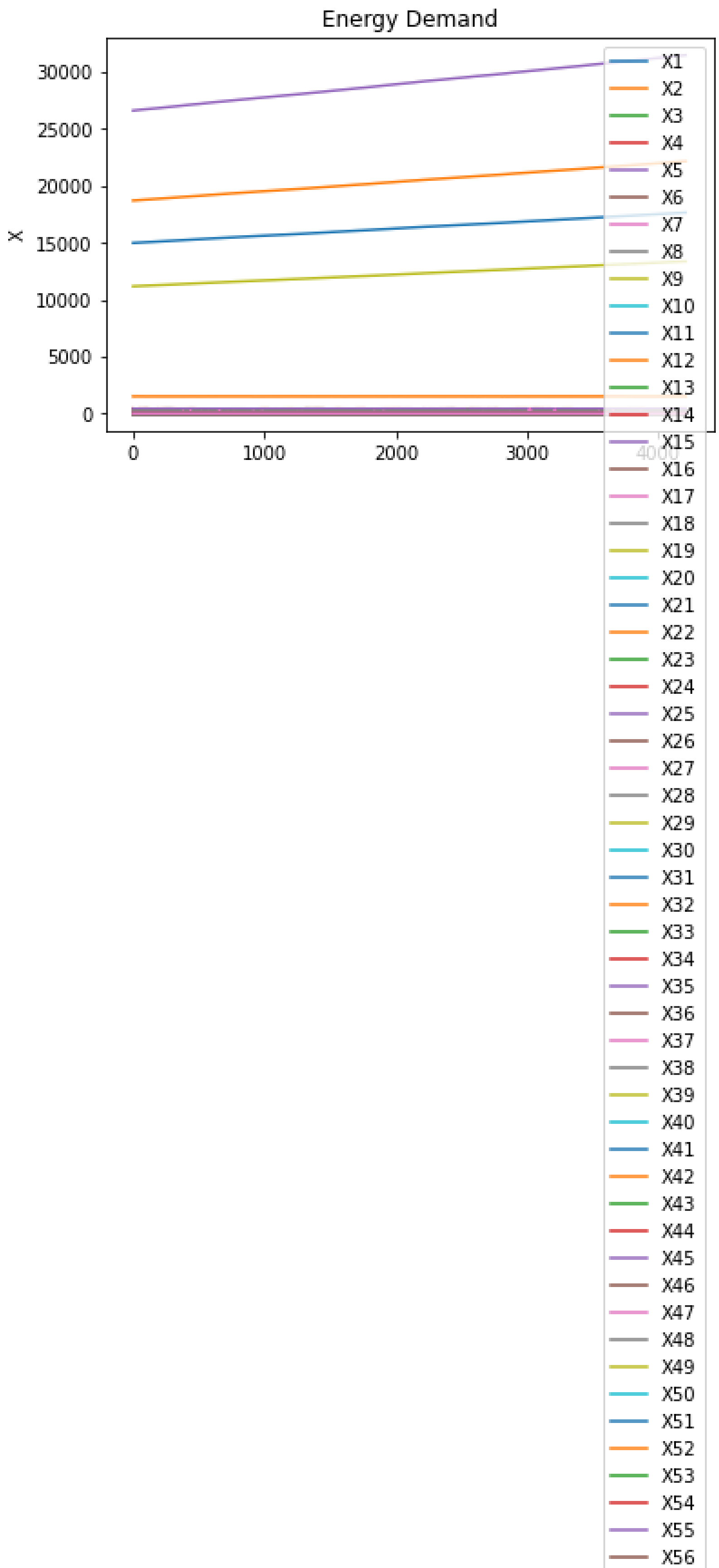
```
Y = dataset.iloc[:0,:]
```

```
Y.head()
```

TimeStamp1	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X48	X49	X50	X51	X52	X53	X54

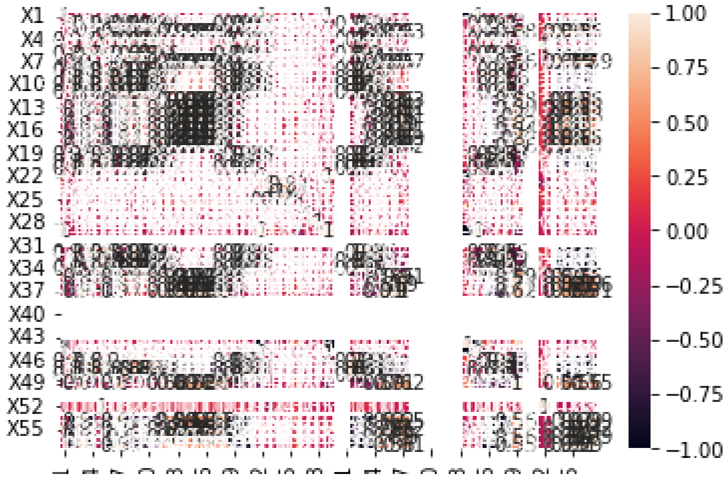
0 rows × 58 columns

```
dataset.plot(title="Energy Demand")
plt.ylabel("X")
plt.show()
```




```
correlation_matrix = dataset.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c14f7cfd0>
```




```
dataset.mean()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Droppi
  """Entry point for launching an IPython kernel.
X1     16307.234829
X2         4.737531
X3       239.668974
X4         8.644678
X5        50.001566
X6         0.771180
X7         3.889704
X8         6.136536
X9         2.502513
X10        1.977516
X11        1.718300
X12      415.100280
X13      414.120187
X14      417.866418
X15      413.314234
X16      238.439104
X17      240.609058
X18      239.958759
X19       10.517634
X20        8.228366
X21        7.188035
X22    20420.306661
X23        2.259012
X24        1.925685
X25        2.960629
X26        5.215343
X27        5.060764
X28        8.443680
X29    12259.366033
X30        0.000000
X31        0.000000
X32        2.166095
X33        1.443711
X34        1.127725
X35        1.250216
X36        1.347409
```

```
X37        1.292079
X38        0.000000
X39        0.000000
X40        0.000000
X41        0.000000
X42        0.000000
X43        0.000000
X44        0.798644
X45    29027.672496
X46        0.865246
X47        0.728935
X48        0.654975
X49        3.807642
X50        0.000000
X51        0.000000
X52     1500.046994
X53        8.000000
X54       -0.771180
X55       -0.865246
```

```
dataset['average'] = dataset.iloc[:, 1:58].astype(float).mean(axis=1)
```

```
dataset['average']
```

```
0       1328.929652
1       1328.897089
2       1328.922117
3       1329.066996
4       1328.827539
           ...
4201    1559.901765
4202    1559.909074
4203    1559.866357
4204    1559.758185
4205    1560.088085
Name: average, Length: 4206, dtype: float64
```

```
dataset
```

	TimeStamp1	X1	X2	X3	X4	X5	X6	X7
0	06-11-2020 13:06	14976.340	5.215638	242.111923	9.171348	50.101936	0.792295	4.016511
1	06-11-2020 13:12	14976.897	5.207636	241.313522	9.160061	50.043190	0.798129	3.931103
2	06-11-2020 13:18	14977.453	5.224983	241.313767	9.156878	50.062305	0.795530	3.979594
3	06-11-2020 13:25	14978.028	5.406835	242.118484	9.348110	50.021931	0.804664	3.989499
4	06-11-2020 13:31	14978.606	5.425514	240.884155	9.333550	50.038364	0.810980	3.914182
...	...	...	...	...	...	...	...	...
4201	29-11-2020 23:29	17641.536	4.869916	241.946655	8.795601	50.072895	0.770478	4.029195

```
df = dataset[['TimeStamp1','average']]
```

```
df
```

	TimeStamp1	average
0	06-11-2020 13:06	1328.929652
1	06-11-2020 13:12	1328.897089
2	06-11-2020 13:18	1328.922117
3	06-11-2020 13:25	1329.066996
4	06-11-2020 13:31	1328.827539
...	...	...
4201	29-11-2020 23:29	1559.901765
4202	29-11-2020 23:35	1559.909074
4203	29-11-2020 23:41	1559.866357
4204	29-11-2020 23:48	1559.758185
4205	29-11-2020 23:54	1560.088085

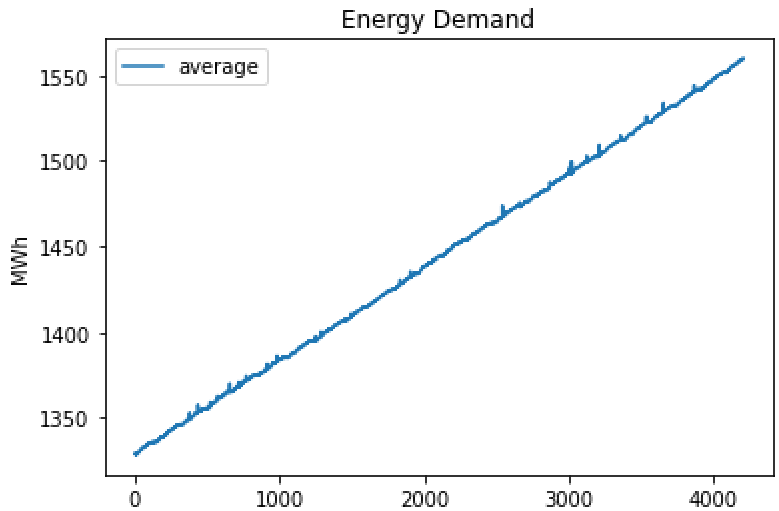
4206 rows × 2 columns

```
df.isnull().sum()
```

```
TimeStamp1    0
average       0
```

```
dtype: int64
```

```
df.plot(title="Energy Demand")
plt.ylabel("MWh")
plt.show()
```




```
mean = np.mean(df.average.values)
std = np.std(df.average.values)
skew = skew(df.average.values)
ex_kurt = kurtosis(df.average)
print("Skewness: {} \nKurtosis: {}".format(skew, ex_kurt+3))
```

```
Skewness: 0.010132741199053847
Kurtosis: 1.8019830339303589
```

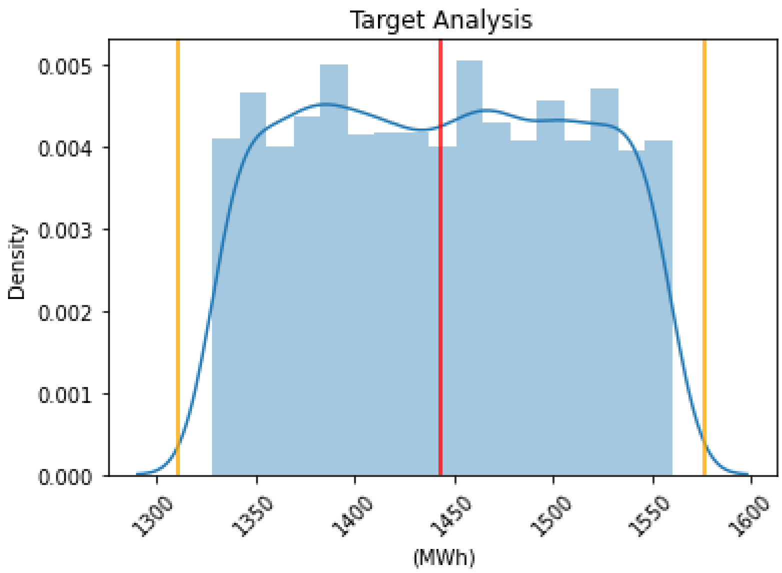
Kurtosis below 3: It means that tails are slightly thinner than in a Normal distribution. It is said that the distribution is platykurtic and the chance of finding extre values is lower than in a normal distribution.

```
def shapiro_test(df, alpha=0.05):
    stat, pval = shapiro(df)
    print("H0: Data was drawn from a Normal Ditribution")
    if (pval<alpha):
        print("pval {} is lower than significance level: {}, therefore null hypothesis is rej
    else:
        print("pval {} is higher than significance level: {}, therefore null hypothesis canno

shapiro_test(df.average, alpha=0.05)
```

```
H0: Data was drawn from a Normal Ditribution
pval 4.824349579452116e-34 is lower than significance level: 0.05, therefore null hypoth
```

```
sns.distplot(df.average)
plt.title("Target Analysis")
plt.xticks(rotation=45)
plt.xlabel("(MWh)")
plt.axvline(x=mean, color='r', linestyle='-', label="\mu: {0:.2f}%".format(mean))
plt.axvline(x=mean+2*std, color='orange', linestyle='-')
plt.axvline(x=mean-2*std, color='orange', linestyle='-')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```

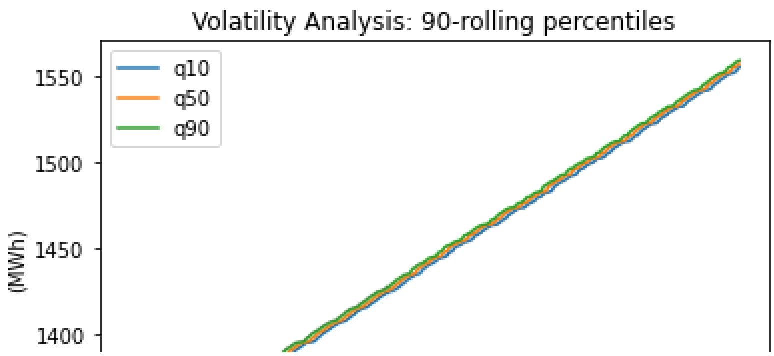


Broadly speaking, data does not look like a normal distribution, because it has a small left tail and the chance of observing extreme values is smaller, comparing to normally distributed data

Volatility Analysis

```
# Insert the rolling quantiles to the monthly returns
data_rolling = df.average.rolling(window=90)
data['q10'] = data_rolling.quantile(0.1).to_frame("q10")
data['q50'] = data_rolling.quantile(0.5).to_frame("q50")
data['q90'] = data_rolling.quantile(0.9).to_frame("q90")

data[["q10", "q50", "q90"]].plot(title="Volatility Analysis: 90-rolling percentiles")
plt.ylabel("(MWh)")
plt.show()
```

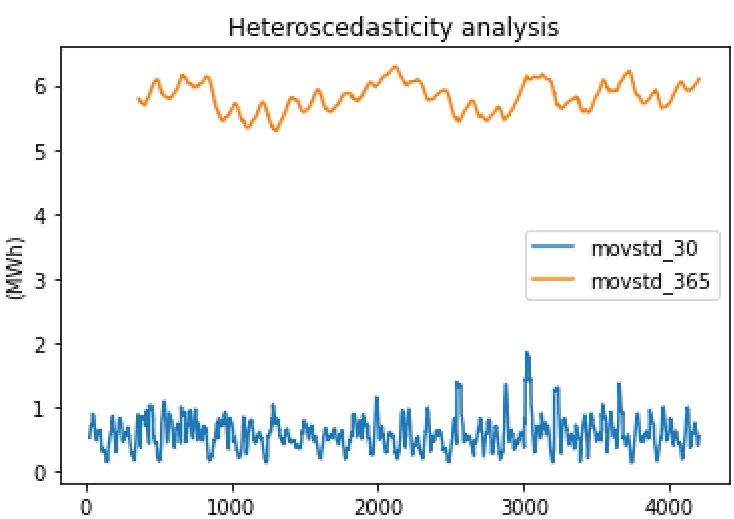
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  """
```




```
data[["movave_7", "movstd_7"]] = df.average.rolling(7).agg([np.mean, np.std])
data[["movave_30", "movstd_30"]] = df.average.rolling(30).agg([np.mean, np.std])
data[["movave_90", "movstd_90"]] = df.average.rolling(90).agg([np.mean, np.std])
data[["movave_365", "movstd_365"]] = df.average.rolling(365).agg([np.mean, np.std])

plt.figure(figsize=(20,16))
data[["average", "movave_7"]].plot(title="Daily Energy Demand in Spain (MWh)")
plt.ylabel("(MWh)")
plt.show()
```

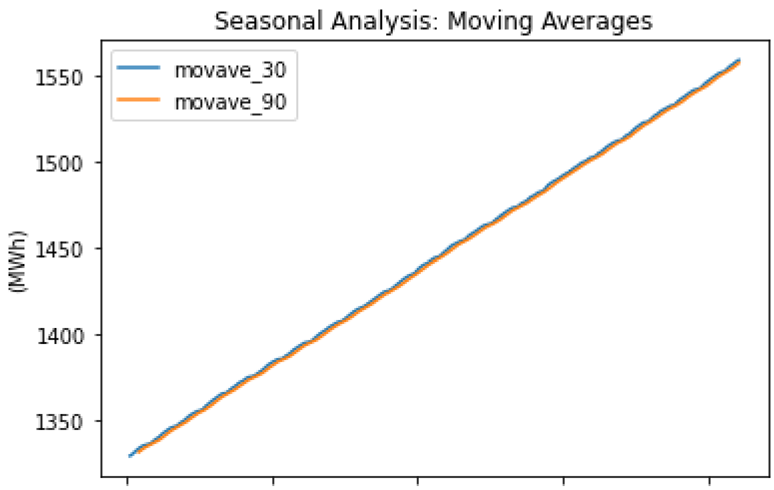
```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
<Figure size 1440x1152 with 0 Axes>
```

```
data[["movstd_30", "movstd_365"]].plot(title="Heteroscedasticity analysis")
plt.ylabel("(MWh)")
plt.show()
```

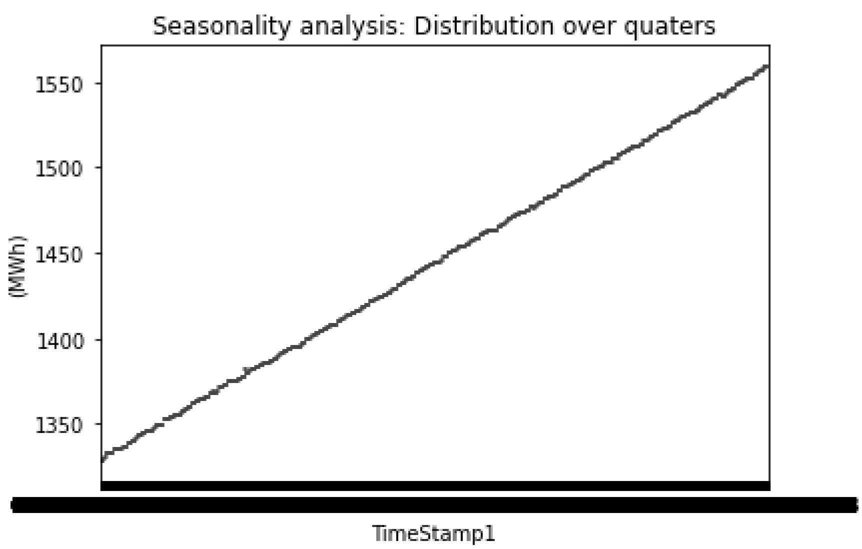


Time Series Analysis: Seasonality and Trend

```
data[["movave_30", "movave_90"]].plot(title="Seasonal Analysis: Moving Averages")
plt.ylabel("(MWh)")
plt.show()
```




```
sns.boxplot(data=df, x="TimeStamp1", y="average")
plt.title("Seasonality analysis: Distribution over quaters")
plt.ylabel("(MWh)")
plt.show()
```



Feature Engineering

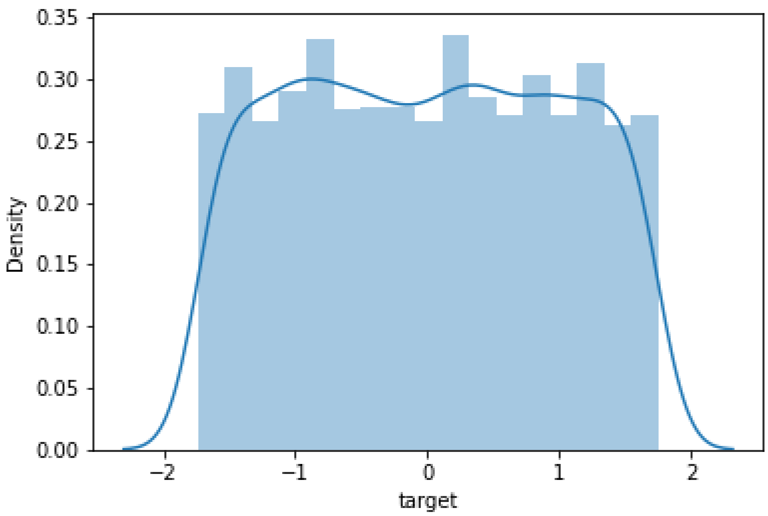
The challenge now is to create some features in a very automated way that can deal with seasonality, trend and changes in volatility. The most basic strategy is to use lagged features and rolling window stats, but consider other advanced techniques for further research:

- Momentum and Mean reversion, like RSI in financial markets
- Sequence minning

Data is standardized in order to allow application of models that are sensitive to scale, like neural networks or svm. Remember that distribution shape is maintained, it only changes first and second momentum (mean and standard deviation)

```
data["target"] = df.average.add(-mean).div(std)
sns.distplot(data["target"])
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```

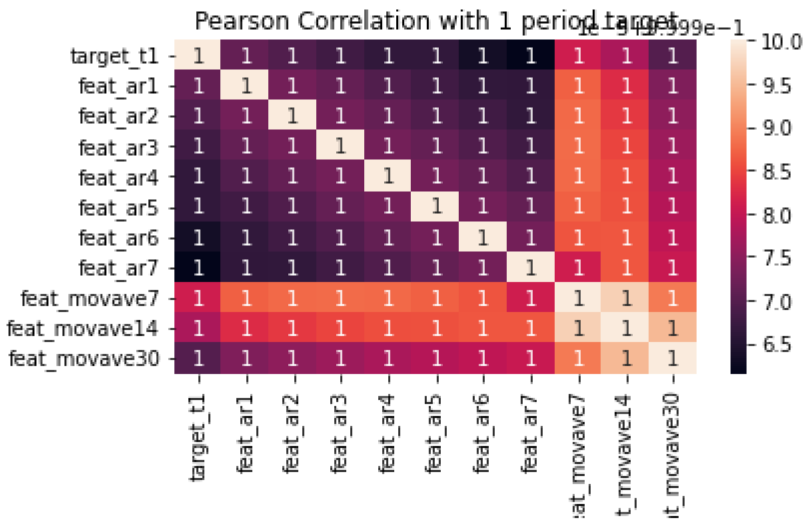



```
features = []
corr_features=[]
targets = []
tau = 30 #forecasting periods

for t in range(1, tau+1):
    data["target_t" + str(t)] = data.target.shift(-t)
    targets.append("target_t" + str(t))

for t in range(1,31):
    data["feat_ar" + str(t)] = data.target.shift(t)
    #data["feat_ar" + str(t) + "_lag1y"] = data.target.shift(350)
    features.append("feat_ar" + str(t))
    #corr_features.append("feat_ar" + str(t))
    #features.append("feat_ar" + str(t) + "_lag1y")


for t in [7, 14, 30]:
    data[["feat_movave" + str(t), "feat_movstd" + str(t), "feat_movmin" + str(t) ,"feat_movma
    features.append("feat_movave" + str(t))
    #corr_features.append("feat_movave" + str(t))
    features.append("feat_movstd" + str(t))
    features.append("feat_movmin" + str(t))
```

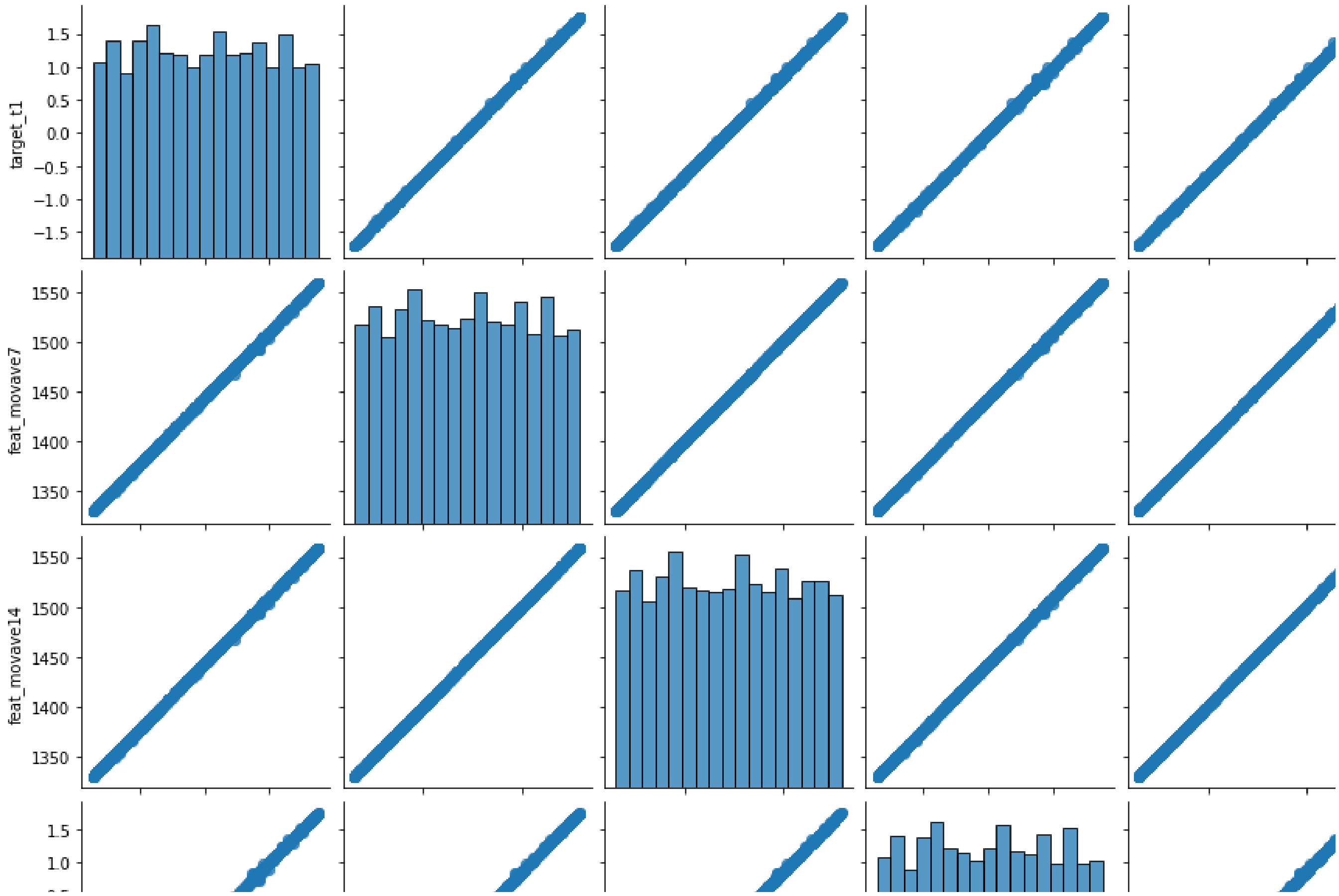
```
    features.append("feat_movmax" + str(t))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[k1] = value[k2]
```

```
corr_features = ["feat_ar1", "feat_ar2", "feat_ar3", "feat_ar4", "feat_ar5", "feat_ar6", "fea


# Calculate correlation matrix
corr = data[["target_t1"] + corr_features].corr()

top5_mostCorrFeats = corr["target_t1"].apply(abs).sort_values(ascending=False).index.values[:


# Plot heatmap of correlation matrix
sns.heatmap(corr, annot=True)
plt.title("Pearson Correlation with 1 period target")
plt.yticks(rotation=0); plt.xticks(rotation=90)  # fix ticklabel directions
plt.tight_layout()  # fits plot area to the plot, "tightly"
plt.show()
```




```
sns.pairplot(data=data[top5_mostCorrFeats].dropna(), kind="reg")
plt.title("Most important features Matrix Scatter Plot")
plt.show()
```



There are some features that are quite strongly linearly correlated with target, like AR_6 and MOVAVE_7, let's build some models and check this assumption

Model Building

In this step,models are build using an nice feature in Scikit-Learn such us MultiOutput Regression, it provides a framework to automatically and easily fit models to predict several target variables.

First a baseline model (linear regression) will be fit and compared to a more advanced model, like Random Forest. A linear model does not need hyperparamenter tunning, and there is some correlation in data, so it is a strongh foundation, but there are several caveats:

- Target variable is not perfectly normally distributed with constant variance
- There are a lot of multicollinearity among predictors
- Observations are not independent

On the other hand an advanced model, like Random Forest, needs to perform hyperparamenter tunning, tipically it is solved by using GridSearch and Cross Validation, but time series data is not suitable to be used in CV, because data is shuffled in order to build k-folds. On the other hand, Scikit-

Learng provide us with a nice solution: TimeSeries Splits, that respect time structure of date and

```
data_feateng = data[features + targets].dropna()
nobs= len(data_feateng)
print("Number of observations: ", nobs)
```

```
Number of observations:  4146
```

Split Data

```
X_train = data_feateng.loc[0:2800,:][features]
y_train = data_feateng.loc[0:2800,:][targets]

X_test = data_feateng.loc[2800:,:][features]
y_test = data_feateng.loc[2800:,:][targets]

n, k = X_train.shape
print("Total number of observations: ", nobs)
print("Train: {}{}, \nTest: {}{}".format(X_train.shape, y_train.shape,
                                          X_test.shape, y_test.shape))

plt.plot(y_train.index, y_train.target_t1.values, label="train")
plt.plot(y_test.index, y_test.target_t1.values, label="test")
plt.title("Train/Test split")
plt.xticks(rotation=45)
plt.show()
```

```
Total number of observations:  4146
Train: (2771, 42)(2771, 30),
Test: (1376, 42)(1376, 30)
```

Baseline Model: Linear Regression

```
from sklearn.linear_model import LinearRegression
```

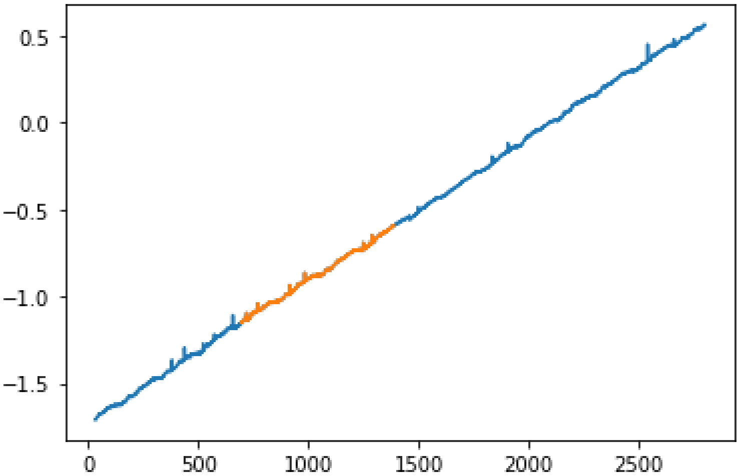
```
from sklearn.metrics import mean_squared_error

reg = LinearRegression().fit(X_train, y_train["target_t1"])
p_train = reg.predict(X_train)
p_test = reg.predict(X_test)

RMSE_train = np.sqrt(mean_squared_error(y_train["target_t1"], p_train))
RMSE_test = np.sqrt(mean_squared_error(y_test["target_t1"], p_test))

print("Train RMSE: {}\nTest RMSE: {}".format(RMSE_train, RMSE_test) )
```

```
Train RMSE: 0.0050225406152153585
Test RMSE: 0.0070773105851623655
```

Train a Random Forest with Time Series Split to tune Hyperparameters

```
from sklearn.model_selection import TimeSeriesSplit, ParameterGrid

splits = TimeSeriesSplit(n_splits=3, max_train_size=365*2)
for train_index, val_index in splits.split(X_train):
    print("TRAIN:", len(train_index), "TEST:", len(val_index))
    y_train["target_t1"].plot()
    y_train["target_t1"][val_index].plot()
    plt.show()
```

```
TRAIN: 695 TEST: 692
```




```
TRAIN: 730 TEST: 692
```

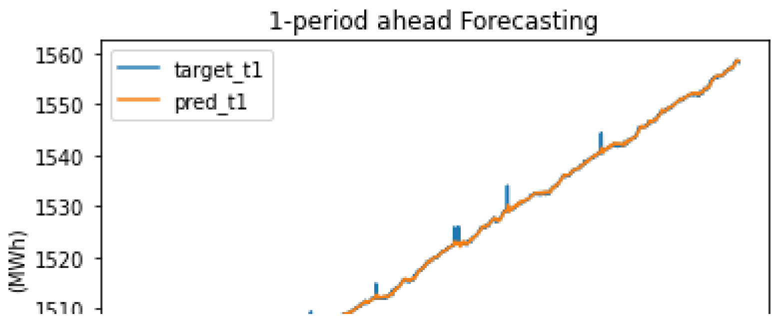
Model Assessment:

Performance Metrics: MAPE (Mean Absolute Percent Error) Even though RMSE is a very common performance metric, MAPE is very suitable to use, and much easer to understand and communicate. Let's use one period ahead model to compute MAPE in test period

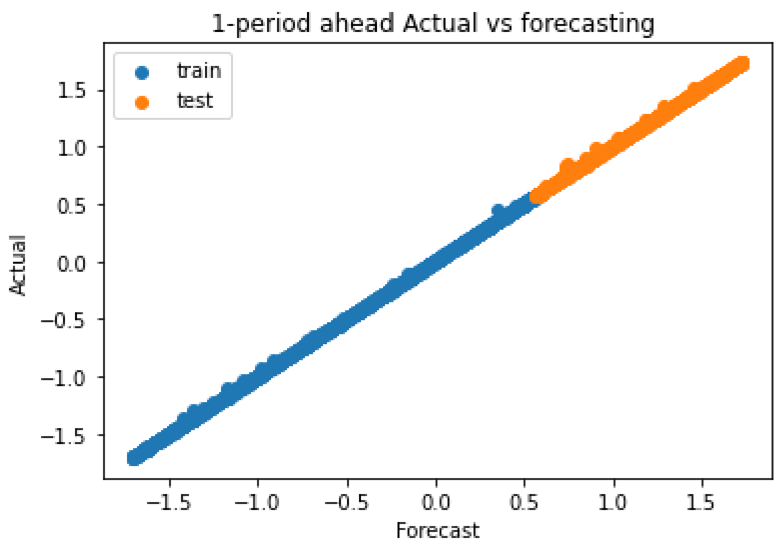
```
test_df = y_test[["target_t1"]]*std+mean
test_df["pred_t1"] = p_test*std+mean
test_df["resid_t1"] = test_df["target_t1"].add(-test_df["pred_t1"])
test_df["abs_resid_t1"] = abs(test_df["resid_t1"])
test_df["ape_t1"] = test_df["resid_t1"].div(test_df["target_t1"])

test_MAPE = test_df["ape_t1"].mean()*100
print("1-period ahead forecasting MAPE: ", test_MAPE)
```

```
1-period ahead forecasting MAPE:  0.002329528246454599
```

```
test_df[["target_t1", "pred_t1"]].plot()

plt.title("1-period ahead Forecasting")
plt.ylabel("(MWh)")
plt.legend()
plt.show()
```




```
plt.scatter(y=y_train["target_t1"],x=p_train, label="train")
plt.scatter(y=y_test["target_t1"],x=p_test, label="test")
plt.title("1-period ahead Actual vs forecasting ")
plt.ylabel("Actual")
plt.xlabel("Forecast")
plt.legend()
plt.show()
```



Plotting actual vs forecasted provides a glance on how good model can fit train data and generalize to test data