MEASURE ENERGY CONSUMPTION

DEVELOPMENT PART -1

Measure energy consumption

To measure energy consumption, you'll need to follow these steps:

Identify the Device: Determine which specific device or appliances you want to measure energy consumption for.

Get a Power Meter: You can use a power meter or energy monitor device that plugs into the wall and then plug your device into it. These devices can provide real-time data on energy usage.

Monitor Usage: Leave the device connected for a specific period while using the appliance normally. This will record the energy consumption.

Record Data: Note down the energy consumption data from the power meter. Some meters might provide data in kilowatt-hours (kWh).

Calculate Costs: If you want to calculate the cost, multiply the energy consumption (in kWh) by your electricity rate (per kWh).

Analyze and Reduce: Review the data to see where you can reduce energy consumption and be more energy-efficient.

Measuring energy consumption for development involves monitoring the energy usage of your development process or project. Here's a general outline for Part 1 of such an endeavor:

Define Objectives: Clearly outline your goals for measuring energy consumption during development. Are you looking to reduce energy usage, assess environmental impact, or optimize costs?

Identify Metrics: Decide which energy metrics you want to measure. Common metrics include electricity, gas, water, and fuel consumption.

Baseline Data: Gather historical data if available. This provides a baseline for your measurements.

Instrumentation: Install energy meters, sensors, or data loggers to monitor energy usage. These devices can be specific to your energy sources, e.g., electricity meters, smart thermostats, or water flow sensors.

Data Collection: Collect real-time data on energy consumption. Ensure accurate and consistent data collection.

Data Storage: Establish a secure and accessible database or system to store the collected data.

Analysis Tools: Set up tools for data analysis, such as spreadsheets, energy management software, or custom solutions.

Data Visualization: Create visual representations of energy consumption data, like charts and graphs, to better understand the trends.

Interpretation: Interpret the data to identify patterns, anomalies, and areas where energy usage can be optimized.

Documentation: Document your data collection and analysis methods for future reference and reporting.

Stakeholder Engagement: Involve relevant stakeholders, such as engineers, project managers, and environmental experts, in the process to gain different perspectives.

Compliance: Ensure your project aligns with any applicable energy efficiency regulations and standards.

This is Part 1 of the process, which focuses on setting up the infrastructure for measuring energy consumption. In Part 2, you'll analyze the data and implement energy-saving strategies based on your findings.

PROJECT PROGRAM:

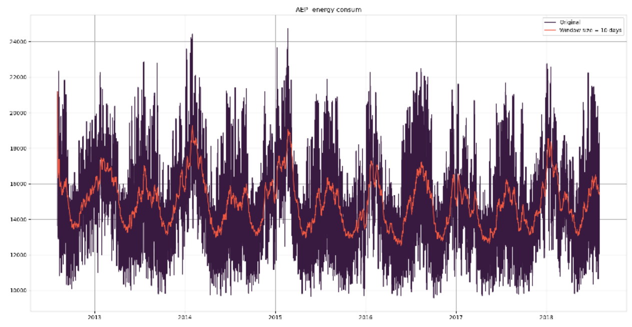
```
import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams

from sklearn.metrics import mean_absolute_error

import seaborn as sns


cmap = plt.colormaps.get_cmap('rocket')

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

df = pd.read_csv('/kaggle/input/hourly-energy-consumption/AEP_hourly.csv', sep =',', parse_dates=['Datetime'], index_col='Datetime')
```

```
df.sort_index(inplace=True)

print(f"Количество измерений: {len(df)}")
```

```
Количество измерений: 121273
```

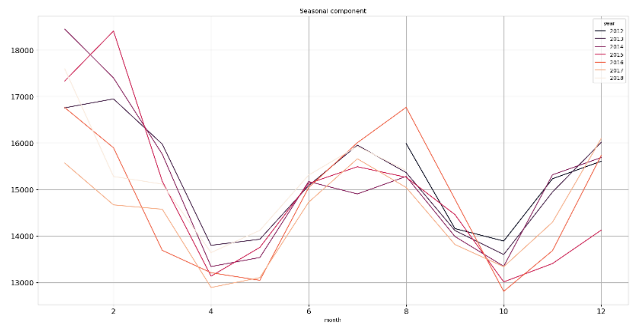
1. Analisys

```
#let's analyse last 6 years

c = []

for i in range(2):

    c.append(cmap(0.15+i/2))


data_len = 365*24*6

df = df.iloc[-data_len:]

plt.figure(figsize = (20,10))

plt.title('AEP  energy consum')

plt.plot(df.AEP_MW, label='Original', color = c[0])

plt.plot(df.AEP_MW.ewm(240).mean(), label='Window size = 10 days', color = c[1])

plt.legend()

plt.grid()

plt.show()
```

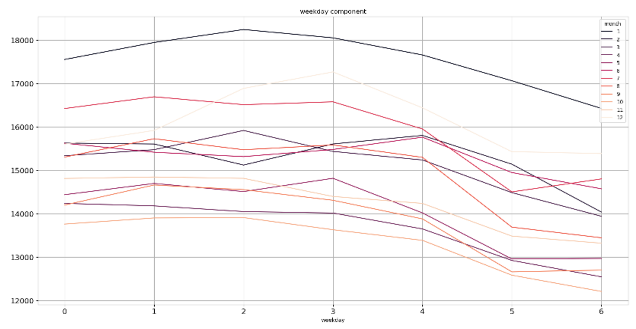


```
df['year'] = df.index.year

df['month'] = df.index.month
```

```
df['hour'] = df.index.hour

df['weekday'] = df.index.dayofweek

monthly_stat = pd.pivot_table(df, values = "AEP_MW", columns = "year", index = "month")
```



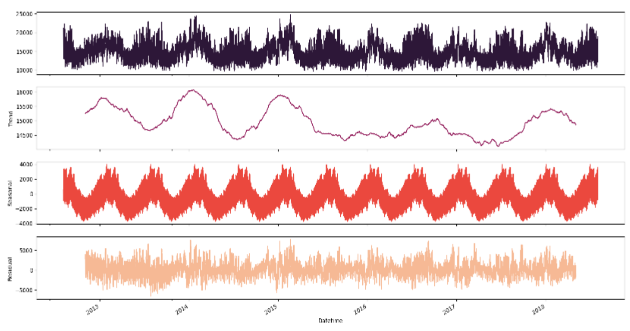
```
daily_stat = pd.pivot_table(df[-365*24:], values = "AEP_MW", columns = "month", index =
"weekday")

# monthly_stat.head(5)

monthly_stat.plot( figsize=(20, 10),title= 'Seasonal component', fontsize=14, cmap = 'rocket')

plt.grid()

daily_stat.plot( figsize=(20, 10),title= 'weekday component', fontsize=14, cmap = 'rocket')

plt.grid()
```



It's well seen that there is correlation bettwen month and expences and also between weekday and expenses.

Then make simple decomposition to seasonal + trend + res

```
import matplotlib.pyplot as plt

from statsmodels.api import tsa as sm


c = []

for i in range(4):

    c.append(cmap(0.12+i/4))


decomp = sm.seasonal_decompose(df.AEP_MW, model='additive', period= 24 * 30 * 6)

fig, axes = plt.subplots(4, 1, sharex=True, figsize=(20, 10))


decomp.observed.plot(ax=axes[0], legend=False, color=c[0])

axes[i].set_ylabel('Observed')

decomp.trend.plot(ax=axes[1], legend=False, color=c[1])

axes[1].set_ylabel('Trend')

decomp.seasonal.plot(ax=axes[2], legend=False, color = c[2])

axes[2].set_ylabel('Seasonal')

decomp.resid.plot(ax=axes[3], legend=False, color=c[3])

axes[3].set_ylabel('Residual')

plt.show()
```



```
import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams

from sklearn.metrics import mean_absolute_error

import seaborn as sns


cmap = plt.colormaps.get_cmap('rocket')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
df = pd.read_csv('/kaggle/input/hourly-energy-consumption/AEP_hourly.csv', sep =',', parse_dates=['Datetime'], index_col='Datetime')

df.sort_index(inplace=True)

print(f"Количество измерений: {len(df)}")
```

```
Количество измерений: 121273
```

1. Analisys

```
#let's analyse last 6 years

c = []

for i in range(2):

   c.append(cmap(0.15+i/2))


data_len = 365*24*6

df = df.iloc[-data_len:]

plt.figure(figsize = (20,10))
plt.title('AEP  energy consum')

plt.plot(df.AEP_MW, label='Original', color = c[0])

plt.plot(df.AEP_MW.ewm(240).mean(), label='Window size = 10 days', color = c[1])

plt.legend()

plt.grid()

plt.show()


df['year'] = df.index.year

df['month'] = df.index.month

df['hour'] = df.index.hour

df['weekday'] = df.index.dayofweek

monthly_stat = pd.pivot_table(df, values = "AEP_MW", columns = "year", index = "month")

daily_stat = pd.pivot_table(df[-365*24:], values = "AEP_MW", columns = "month", index = 
"weekday")

# monthly_stat.head(5)

monthly_stat.plot( figsize=(20, 10),title= 'Seasonal component', fontsize=14, cmap = 'rocket')

plt.grid()


daily_stat.plot( figsize=(20, 10),title= 'weekday component', fontsize=14, cmap = 'rocket')

plt.grid()
```

It's well seen that there is correlation bettwen month and expences and also between weekday and expenses.

Then make simple decomposition to seasonal + trend + res

```
import matplotlib.pyplot as plt
```

```
from statsmodels.api import tsa as sm


c = []

for i in range(4):

    c.append(cmap(0.12+i/4))


decomp = sm.seasonal_decompose(df.AEP_MW, model='additive', period= 24 * 30 * 6)

fig, axes = plt.subplots(4, 1, sharex=True, figsize=(20, 10))


decomp.observed.plot(ax=axes[0], legend=False, color=c[0])

axes[i].set_ylabel('Observed')

decomp.trend.plot(ax=axes[1], legend=False, color=c[1])

axes[1].set_ylabel('Trend')

decomp.seasonal.plot(ax=axes[2], legend=False, color = c[2])

axes[2].set_ylabel('Seasonal')

decomp.resid.plot(ax=axes[3], legend=False, color=c[3])

axes[3].set_ylabel('Residual')

plt.show()
```

Trend looks pretty strange for me. It looks like affection of some other things like crises appearence of another companies, political situation and etc. It requiers future analysis

2. One Day Preidction

I will compare 5 different simple ways to predict expences on the next day:

Mean

Weighted sum

Exponential Smoothin

Random Forest

Xgboosting

```
w_hours = 1*24

n = 10

train = df.iloc[:-w_hours]

tr_sample = train.iloc[-n*w_hours:]

val  = df.iloc[-w_hours:]

from statsmodels.tsa.stattools import adfuller

#Perform Dickey-Fuller test


print('Results of Dickey-Fuller Test:')


dftest = adfuller(tr_sample.AEP_MW, autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

for key,value in dftest[4].items():

    dfoutput['Critical Value (%s)'%key] = value

pvalue = dftest[1]

if pvalue < 0.05:

    print(f'p-value = {round(pvalue,4)}.4f. The series is likely stationary.')

else:

    print(f'p-value = {round(pvalue,4)} The series is likely non-stationary.')
```

```
Results of Dickey-Fuller Test:

p-value = 0.4565 The series is likely non-stationary.
```

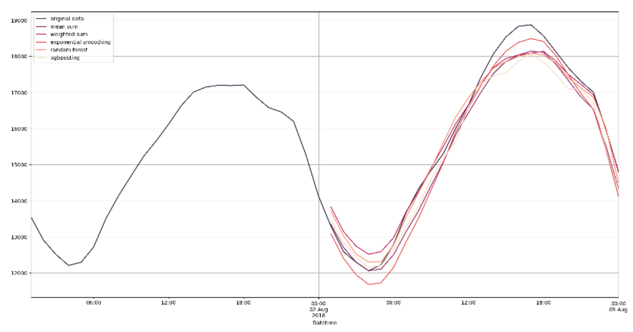
```
def mape(y_true, y_pred):

    #mean_absolute_percentage_error

    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Mean

forecast_arr = np.array(tr_sample.AEP_MW).reshape(n,-1)

forecast_mn = forecast_arr.mean(axis = 0)

# Weighted sum

W = np.array([0.001, 0.003, 0.006, 0.015 , 0.024 , 0.09 , 0.15  , 0.21 , 0.25 , 0.3])

forecast_arr_W = np.array([forecast_arr[x]*W[x] for x in range(len(W))])


forecast_ws = forecast_arr_W.sum(axis = 0)

# Exponential Smoothin

from statsmodels.tsa.api import ExponentialSmoothing


ES = ExponentialSmoothing(tr_sample.AEP_MW, seasonal_periods=24, seasonal='add').fit()

forecast_es = pd.Series(ES.forecast(len(val)))

forecast_es.index = val.index

/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor


RFR = RandomForestRegressor()

X_train = pd.get_dummies(tr_sample[['year','month','hour', 'weekday']])

X_val = pd.get_dummies(val[['year','month','hour', 'weekday']])
```

```
y_train = pd.get_dummies(tr_sample[['AEP_MW']])

y_val = pd.get_dummies(val[['AEP_MW']])


RFR.fit(X_train,y_train)

forecast_rf = pd.Series(RFR.predict(X_val), index = val.index)

/tmp/ipykernel_20/2998306517.py:10: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,), for
example using ravel().

  RFR.fit(X_train,y_train)

# XGboostong

import xgboost as xgb

XGB =  xgb.XGBRegressor()


XGB.fit(X_train, y_train)

forecast_xg = pd.Series(XGB.predict(X_val), index = val.index)

print('MAPE (mean) - ', mape(val.AEP_MW,  forecast_mn))


print('MAPE (weighted sum) - ', mape(val.AEP_MW,  forecast_ws))


print('MAPE (exponential smoothing) - ', mape(val.AEP_MW, forecast_es))


print('MAPE (random forest) - ', mape(val.AEP_MW, forecast_rf))


print('MAPE (xgboosting) - ', mape(val.AEP_MW, forecast_xg))

MAPE (mean) -  2.261731940436
MAPE (weighted sum) -  1.8414168970997982

MAPE (exponential smoothing) -  2.5463113342110097

MAPE (random forest) -  1.833551687353754

MAPE (xgboosting) -  2.403194989184267
```

```
f, ax = plt.subplots(1,1, figsize=(20,10))

pd.concat([tr_sample[-(1)*w_hours:].AEP_MW, val.AEP_MW]).plot(color = cmap(0.1), label = 'original data')

#plotting omly part of data to more inderstanding


forecast_mn = pd.Series(forecast_mn, index = val.index)

forecast_ws = pd.Series(forecast_ws, index = val.index)

forecast_mn.plot(color = cmap(0.33), label = 'mean sum')

forecast_ws.plot(color = cmap(0.46), label = 'weighted sum')

forecast_es.plot(color = cmap(0.60), label = 'exponential smoothing')

forecast_rf.plot(color = cmap(0.76), label = 'random forest')

forecast_xg.plot(color = cmap(0.95), label = 'xgbposting')


plt.grid()

plt.legend()
```

Output:

```
<matplotlib.legend.Legend at 0x7f2cd8338fd0>
```



```
import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams

from sklearn.metrics import mean_absolute_error

import seaborn as sns


cmap = plt.colormaps.get_cmap('rocket')

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

df = pd.read_csv('/kaggle/input/hourly-energy-consumption/AEP_hourly.csv', sep =',', parse_dates=['Datetime'], index_col='Datetime')

df.sort_index(inplace=True)

print(f"Количество измерений: {len(df)}")

Количество измерений: 121273
```

1. Analisys

```
#let's analyse last 6 years

c = []

for i in range(2):

   c.append(cmap(0.15+i/2))


data_len = 365*24*6

df = df.iloc[-data_len:]

plt.figure(figsize = (20,10))
plt.title('AEP  energy consum')

plt.plot(df.AEP_MW, label='Original', color = c[0])

plt.plot(df.AEP_MW.ewm(240).mean(), label='Window size = 10 days', color = c[1])

plt.legend()

plt.grid()

plt.show()


df['year'] = df.index.year

df['month'] = df.index.month

df['hour'] = df.index.hour

df['weekday'] = df.index.dayofweek

monthly_stat = pd.pivot_table(df, values = "AEP_MW", columns = "year", index = "month")

daily_stat = pd.pivot_table(df[-365*24:], values = "AEP_MW", columns = "month", index = "weekday")

# monthly_stat.head(5)

monthly_stat.plot( figsize=(20, 10),title= 'Seasonal component', fontsize=14, cmap = 'rocket')

plt.grid()


daily_stat.plot( figsize=(20, 10),title= 'weekday component', fontsize=14, cmap = 'rocket')

plt.grid()
```

It's well seen that there is correlation bettwen month and expences and also between weekday and expenses.

Then make simple decomposition to seasonal + trend + res

```
import matplotlib.pyplot as plt
```

```
from statsmodels.api import tsa as sm


c = []

for i in range(4):

    c.append(cmap(0.12+i/4))


decomp = sm.seasonal_decompose(df.AEP_MW, model='additive', period= 24 * 30 * 6)

fig, axes = plt.subplots(4, 1, sharex=True, figsize=(20, 10))


decomp.observed.plot(ax=axes[0], legend=False, color=c[0])

axes[i].set_ylabel('Observed')

decomp.trend.plot(ax=axes[1], legend=False, color=c[1])

axes[1].set_ylabel('Trend')

decomp.seasonal.plot(ax=axes[2], legend=False, color = c[2])

axes[2].set_ylabel('Seasonal')

decomp.resid.plot(ax=axes[3], legend=False, color=c[3])

axes[3].set_ylabel('Residual')

plt.show()
```

Trend looks pretty strange for me. It looks like affection of some other things like crises appearence of another companies, political situation and etc. It requiers future analysis

3. One Week Prediction

```
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import TimeSeriesSplit


errors = []


tscv = TimeSeriesSplit(n_splits=5, max_train_size = 3*7*24, test_size = 7*24)


n = 7*7

k = 7

train = df.iloc[:-k*w_hours]

tr_sample = train.iloc[-n*w_hours:]

val   = df.iloc[-k*w_hours:]


# data = pd.concat([tr_sample, val]).AEP_MW


for train_idx, test_idx in tscv.split(tr_sample.AEP_MW):

    ES = ExponentialSmoothing(tr_sample.AEP_MW.iloc[train_idx] ,seasonal_periods=24*7,
seasonal='add').fit()

    forecast_es = ES.forecast(len(test_idx))


#     # Считаем ошибку

    actual = tr_sample.AEP_MW.iloc[test_idx]

    error = mape(actual.values, forecast_es.values)

    errors.append(error)


forecast_es = ES.forecast(len(test_idx))
```

```
forecast_es.index = val.index

print('MAPE(cross val) - ', np.mean(errors))

/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

MAPE(cross val) -  6.904413349137849

/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

import warnings

warnings.filterwarnings('ignore')

RFR = RandomForestRegressor()


depth = np.arange(6, 9, 1)

feat = np.arange(5, 9, 1)
```

```
samp = np.arange(2, 4, 1)

leaf = np.arange(1, 4, 1)

est = [200]


## Search grid for optimal parameters

rf_param_grid = {

    "max_depth": depth,

    "max_features": feat,

    "min_samples_split": samp,

    "min_samples_leaf": leaf,

    "bootstrap": [False],

    "n_estimators": est

    # "criterion": ["gini"]

}



# errors = []

X_train = pd.get_dummies(tr_sample[['year','month','hour', 'weekday']])

X_val = pd.get_dummies(val[['year','month','hour', 'weekday']])


gsRFR = GridSearchCV(RFR,

                    param_grid=rf_param_grid,

                    cv=tscv.split(X_train),

                    scoring='r2',

                    n_jobs = 8,
                    verbose = 1)


y_train = pd.get_dummies(tr_sample[['AEP_MW']])

# y_val = pd.get_dummies(val[['AEP_MW']])


gsRFR.fit(X_train, y_train.squeeze())

print("score = ", gsRFR.best_score_)

print(gsRFR.best_params_)

Fitting 5 folds for each of 72 candidates, totalling 360 fits

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy
```

```
version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy
version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

score =  0.628859530693294

{'bootstrap': False, 'max_depth': 6, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}

forecast_rf = pd.Series(gsRFR.predict(X_val), index = val.index)

print('MAPE(cross val) - ', mape(val.AEP_MW, forecast_rf))

MAPE(cross val) -  7.081567637390411

# XGboostong

import xgboost as xgb

XGB =  xgb.XGBRegressor()


lr = np.arange(0.01, 0.04, 0.003)

depth = [3, 4, 6]

# depth = np.arange(11, 13, 1)

# leaf = [9, 13, 14]

sub = [0.75, 0.8, 0.85, 0.9]

tree = [0.7, 0.75, 0.8]

split = [2, 3]


## Search grid for optimal parameters

xg_param_grid = {

   "max_depth": depth,
    "learning_rate": lr,

    "subsample": sub,

    "colsample_bytree": tree

    # "colsample_bylevel": 0.8,

    # "reg_lambda": 0.1,

    # "eval_metric": "rmse",

    # "random_state": 42,

}


gsXGB = GridSearchCV(XGB,

                     param_grid=xg_param_grid,

                     cv=tscv.split(X_train),

                     scoring='r2',

                     n_jobs = 8,

                     verbose = 1)


gsXGB.fit(X_train, y_train)


print(gsXGB.best_params_)

Fitting 5 folds for each of 360 candidates, totalling 1800 fits

{'colsample_bytree': 0.7, 'learning_rate': 0.037000000000000001, 'max_depth': 6, 'subsample': 0.75}

forecast_xg = pd.Series(gsXGB.predict(X_val), index = val.index)

print('MAPE(cross val) - ', mape(val.AEP_MW, forecast_xg))

MAPE(cross val) -  5.0714998847131705

plt.figure(figsize=(20,10))
```

```
data = pd.concat([tr_sample.AEP_MW.iloc[-4*24:], val.AEP_MW])

data.plot(color = cmap(0.1), label = 'original')

# data.index

# forecast_es.plot(color = cmap(0.5), label = 'exponential smoothing')

forecast_rf.plot(color = cmap(0.3), label = 'random forest')

forecast_es.plot(color = cmap(0.6), label = 'exponential smoothing')

forecast_xg.plot(color = cmap(0.9), label = 'xg boosting')


plt.grid()

plt.legend()
```

Output:

```
<matplotlib.legend.Legend at 0x7f2cd3347760>
```

4. Results:

One day prediction works very nice for all methods. Even without presize parametres fitting of RFregressor and XGBregressor. However, when we try to predict one week expenses it didn't works as good as with one day.

Probably it will be nice to try SARIMA and RNN.