**Predict Power Consumption of Tetouan City**

|  |
| --- |
|  |

**응용통계학과**

**주건재**

|  |
| --- |
|  |

<  목    차  >

1 서론 1

1.1분석개요 1

1.2 분석목표 1

1.3 데이터소개 1

2 본론 2

2.1 분석 플로우 2

2.2 EDA 및 전처리 3

2.3 주성분분석 5

2.4 OLS 6

2.5 분석모델 7

3 결론 9

3.1 분석결과 및 토의 9

3.2 결론 및 추후 연구 9

4 REFERENCES 11

5 Appendix 12

A source code 12

1. **서론**
   1. **분석개요**

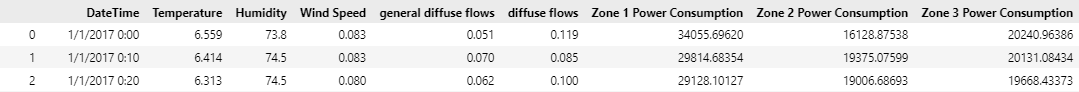
테투안(Tetouan)은 인구 약 38만명이 사는 모로코의 도시로 지중해 인근에 위치한다. 본 연구는 UCI Repository에서 제공하는 테투안 전력소모 데이터를 사용하여 하루 총 전력 소모량을 예측할 것이다.

* 1. **분석 목표**

테투안의 일별 전력소모를 기온, 습도, 풍속, 대기류의 데이터를 사용하여 예측한다.

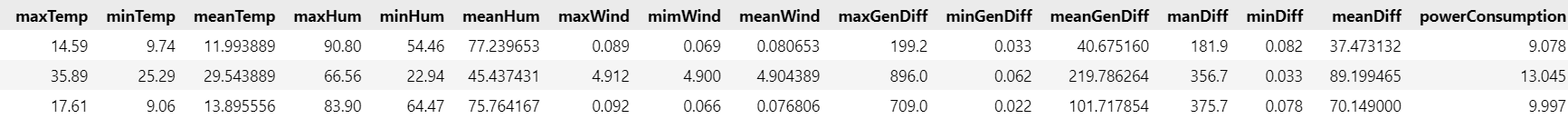
첫째 목표: Streaming data로 가정하여 hold out이 아닌 online learning 방법을 사용한다.  
둘째 목표: Online algorithm을 앙상블하여 예측한다.

* 1. **데이터소개**

본 데이터는 uci 저장소에서 오픈소스로 제공한다[[1]](#footnote-1). 

<표1> 전력소모 원본 데이터

<표1>의 데이터는 10분에 한번씩 수집된 데이터로 1, 2, 3 구역으로 나뉘어 전력소모량이 관측되었다. 독립변수는 기온, 습도, 풍속, 대기류 데이터이며 종속변수는 전력소모량이 된다.

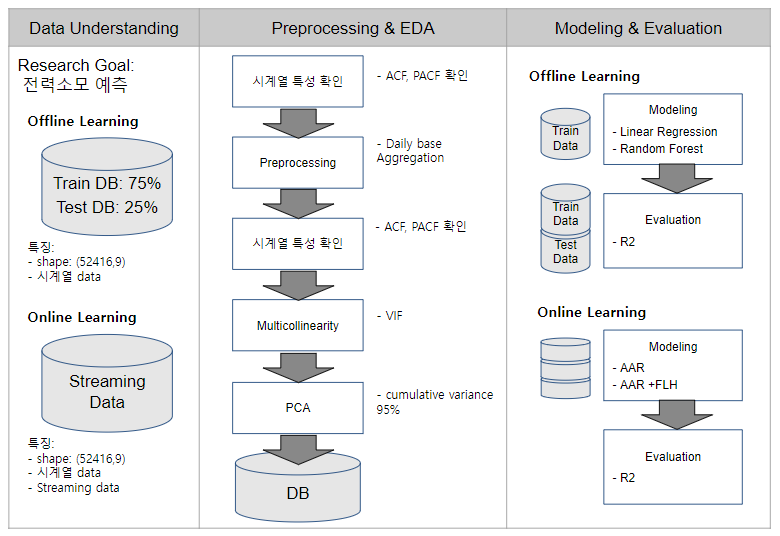


<표2> 전력소모 변환 데이터

데이터를 연구 목표에 맞게 <표2>와 같이 세 구역의 전력소모를 합하고 10분 단위의 데이터를 1일 단위로 합하였다. 독립변수 기온, 습도, 풍속, 대기류의 데이터는 각각 그 날의 최소, 최대, 평균값을 계산하여 재생산하였다. 하루 앞서 다음날 사용될 총전력소모를 예측하여야 하기 때문에 종속변수 총전력소모량은 1 lag를 주어 데이터를 시프트 해주었다. 데이터는 총 17개의 15개의 설명변수와 364개의 샘플로 이루어져 있다.

1. **본론**

**2-1. 분석 플로우**

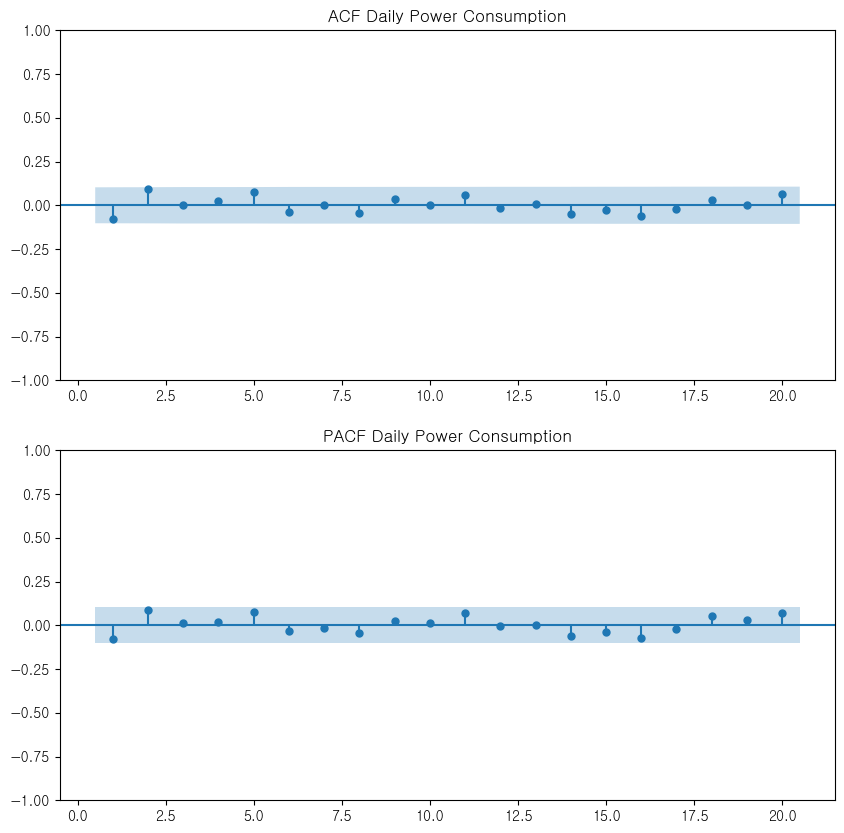


<표3> 전체 분석 플로우

<표3>은 분석의 전체 플로우를 나타내고 있다. 먼저 시계열 특성을 확인하여 발견되면 그에 맞는 전처리를 한다. 이후 다중공선성을 확인하고 만약 존재한다면 주성분분석(PCA)로 분산 누적 95%에 해당하는 성분을 가지고 모델링을 진행한다. 전처리가 끝난 데이터를 Offline learning 방법의 Linear Regression과 Random Forest 방법으로 모델링하고 Online learning 방법의 Aggregating Algorithm for Regression(AAR) 모델과 앙상블 기법인 Follow the leading History(FLH)를 사용하여 모델링하여 모델들의 성능을 비교한다.

* 1. **EDA 및 전처리**

1. **시계열 특성 확인**



<그림1> ACF, PACF 그래프

<그림1>은 종속 변수인 총전력소모의 Autocorrelation Function(ACF)와 Partial Autocorrelation Function(PACF)의 그래프이다.

https://lh4.googleusercontent.com/0271LDPMKQbQhk70znWbhwPu6LdTqT7QqPLv8FyCAuRGQfAcD4idr_0OXZHfDr9_U0DG_AnBoCMJKox6GXWaEGKpIgfQx1DelljchdeFg2i9M6Hkk11YIG7K5GdBjWhCyfoNFsP6Jmo

<식1> ACF 식

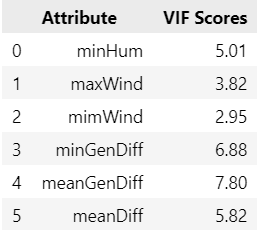
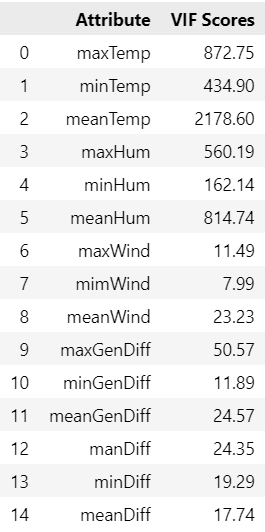
<식1>은 ACF의 식으로 와 의 상관관계를 나타내는 식이다. 즉, t번째 값이 lag k 이후 값과 얼마나 관계가 있는지 보는 것이다. 자기상관관계가 높으면 ACF값이 1에서 0으로 k값이 커짐에 따라 서서히 수렴한다. 아니라면 <그림1>과 같이 바로 0으로 수렴한다.

<식2> PACF 식

<식2>는 PACF의 식으로 와 의 상관관계를 나타낸다. ACF와 다른 점은 t-k까지의 영향을 빼고 상관관계를 본다는 것이다.

<그림1>에서 나타나듯이 데이터에 시계열적 특성은 존재하지 않는다.

1. **VIF**

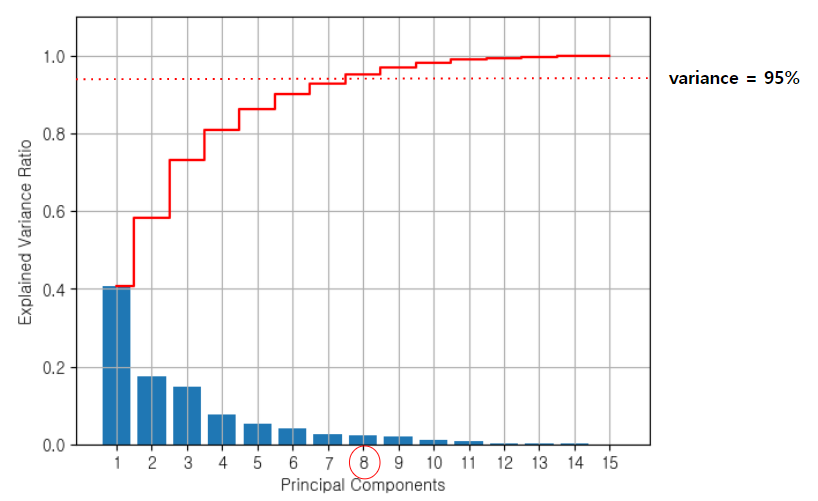


<표4> 제거 전 설명 변수들의 VIF <표5> 반복 제거 후 설명 변수들의 VIF

<표4>는 설명 변수들의 VIF값이다. 기준을 10으로 세팅한 후 가장 값이 가장 높은 변수를 제거한다. 다시 VIF값을 구한 후 제거를 반복하면 <표5>와 같이 설명변수들이 남는다. 너무 많은 변수들이 제거되어 설명력이 떨어질 것이 우려되어 변수 제거가 아닌 주성분분석(PCA)를 진행하기로 결정하였다.

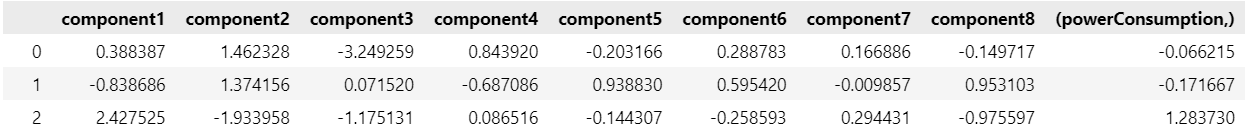
* 1. **주성분분석(PCA)**

다중공선성의 문제를 해결하기 위해 변수 제거가 아닌 주성분분석(PCA) 방법을 채택하였다. PCA는 설명변수들의 선형결합으로 만들어진 새로운 변수이다. PCA의 변수(컴포넌트)들은 서로 직교하며 본래 변수들을 가장 잘 대변할 수 있도록 분산을 최대화하는 컴포넌트들부터 순서대로 나열한다. 원래 데이터의 변수 개수만큼의 컴포넌트를 만들고 분산이 가장 큰 컴포넌트부터 누적 분산이 일정 이상 되는 컴포넌트까지 채택해서 사용한다. 본 연구에서는 누적 분산 95%로 세팅한다.



<그림2> PCA 결과 그래프

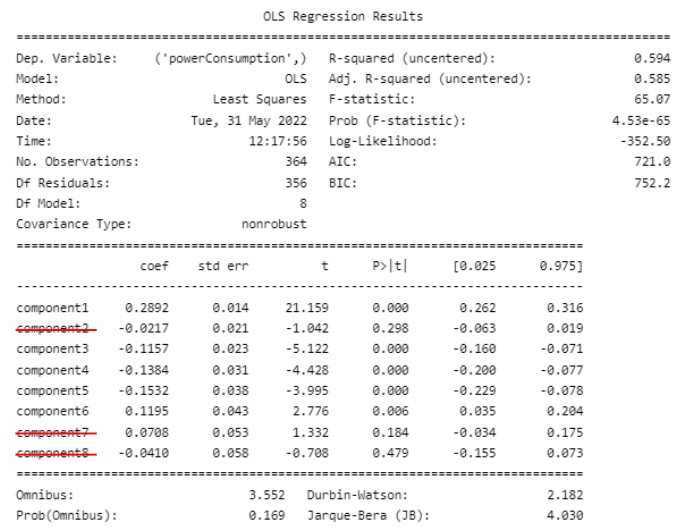
<그림2>는 데이터의 주성분 분석 결과 그래프이다. 막대그래프는 각 컴포넌트의 분산을 나타내며 실선은 누적 분산을 나타낸다. 누적분산 95% 기준 8개의 컴포넌트를 채택하여 사용한다.



<표6> 주성분분석 후 데이터

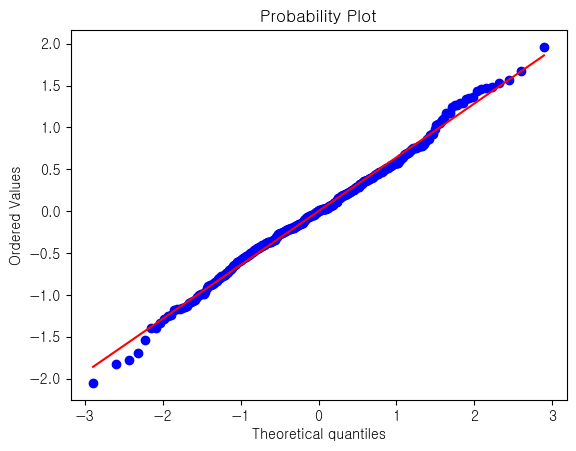
<표6>은 주성분분석 후 8개의 컴포넌트만 선택한 전처리가 완료된 데이터이다.

* 1. **OLS**



<표7> OLS Summary

<표7>은 OLS의 서머리이다. 테이블에서 나타나듯이 컴포넌트2, 7, 8은 신뢰구간을 볼 때 유의하지 않음을 알 수 있다.



<그림3> 컴포넌트 2, 7, 8 제거 후 QQ-Plot

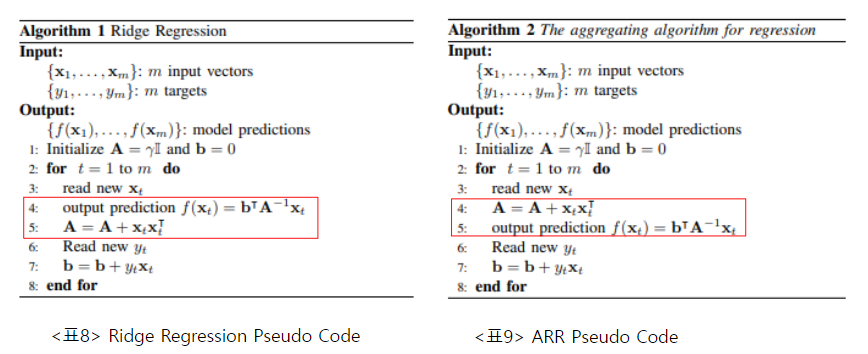
<그림3>은 컴포넌트 2, 7, 8을 제거한 후의 QQ-Plot이다. 정규성을 만족하는 것을 볼 수 있다. 사용할 모델이 회귀모델이기 때문에 해당 컴포넌트들을 제거하고 모델링을 한다.

* 1. **분석모델**
* **Online Learning**

Online Learning은 머신러닝의 한 분야로 Streaming data 샘플이 하나 들어오면 모델을 재 학습하며 예측값을 산출하는 모형이다. Train/test 셋으로 나누어 평가하는 Hold Out 방식과는 다르게 샘플이 들어오면 샘플의 설명변수를 받아 모델을 업데이트하고 예측값을 계산한 뒤 실제 종속변수를 가지고 마저 업데이트한다.

* **Aggregating Algorithm for Regression (AAR)**
  + 모델

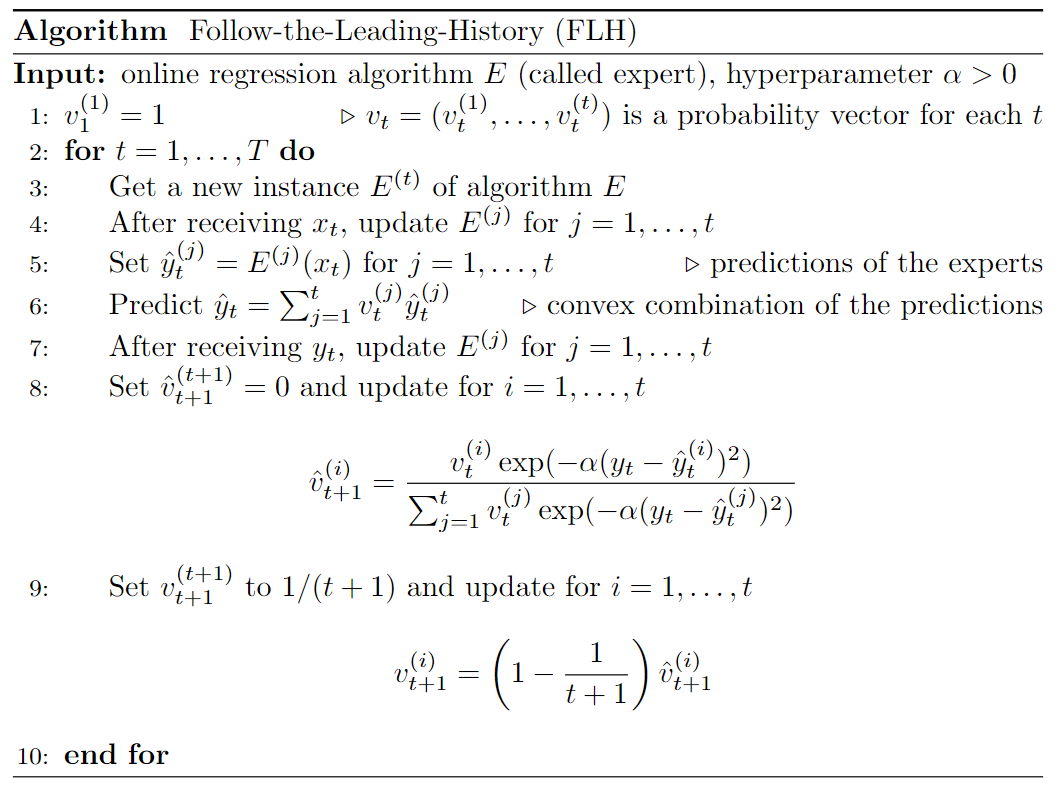
AAR 모형은 Ridge Regression의 Online Learning 버전이다.



<표8>과 <표9>은 각각 Ridge Regression과 AAR의 Pseudo 코드이다. 이 둘의 차이는 데이터 샘플들로 반복 학습을 하는 과정에서 언제 예측을 하는지에 있다. 빨간 박스를 보면 Ridge Regression의 경우 예측이 A와 b의 예측과 분리되어 있어 모델을 업데이트 하지 않아도 예측을 할 수 있다. 다시 말해 예측해야 하는 샘플 데이터가 들어왔을 때 예측만 하고 모델을 업데이트 시키지 않으면 된다. 반면 AAR의 경우 예측 과정이 A와 b의 업데이트 중간에 위치한다. 즉, A를 업데이트 해야 예측이 가능하다.

* **Follow the Leading History (FLH)**
  + 모델

Follow the Leading History {FLH}: data drift/shift 문제 해결을 목적으로 최근 1개, 2개, …, n개의 샘플로 fitting한 n개의 모델로 구성되고 스트리밍 샘플을 추가하면 online learning 모델도 추가 생성하고 기존 모델도 업데이트하는 방식이다. 최종 예측 값은 각 모델의 예측 값은 weighted mean으로 계산하며 weight는 목적함수에 의해 결정된다. 즉 잘못된 예측 값을 산출한 모델은 weight를 줄여서 해당 모델의 예측 값을 영향력을 낮춘다.



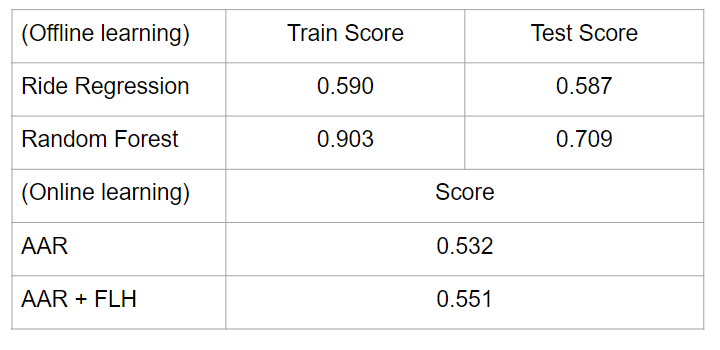
<표10> FLH Pseudo Code

<표10> 는 FLH의 Pseudo 코드로 는 t시점에 생성된 expert(모델)로 생성된 시점부터 샘플들이 들어올 때마다 추가학습이 된다. 는 각 expert의 weight이다. 목적함수에 의해 업데이트된다. AAR과 FLH를 결합해서 사용하면 각 Expert가 AAR이다.

1. **결론**

**3-1. 분석결과 및 토의**

분석의 결과는 R2 값으로 평가하여 모델들을 비교했다. R2의 경우 0에서 1까지의 값을 가지며 1에 가까울수록 성능이 좋은 모델이다.



<표11> 모델 분석결과 비교표

<표11>은 모델들의 분석결과를 비교하는 테이블이다. Online Learning 알고리즘들 보다 Offline Learning 알고리즘들이 좋은 성능을 나타냈다. 또 회귀모형 보다는 Random Forest모형이 좋은 결과를 도출하였다.

**3-2. 연구의 결론 및 한계**

연구의 결과를 보면 Online Learning의 결과가 Offline Learning의 결과보다 성능이 다소 안 좋은 것을 볼 수 있다. 이는 많은 데이터를 학습한 후 예측을 하는 Hold out 방법에 비해 한개의 샘플을 학습해도 예측을 해야 하는 Online Learning의 특성 때문이라고 할 수 있다. 이를 극복하기 위해서는 학습 초기에 스트림 데이터를 모아서 미니배치 전체를 먼저 학습시킨 후 Online Learning으로 전환하는 방법이 있다. 이를 활용하면 이러한 loss를 줄일 수 있을 것이다.

AAR 모형 보다 AAR과 FLH를 결합한 모형이 좋은 성능을 보였다. 이는 FLH 앙상블 기법이 의미 있음을 시사한다. 리소스 코스트는 다소 있지만 무겁지 않은 AAR 모형과 같은 모델이라면 FLH를 활용하여 모델의 성능을 높이는 것을 고려할 수 있겠다.

또한 회귀 모형보다는 성능이 좋은 Random Forest를 사용하는 것이 좋다. Random Forest를 online Learning 알고리즘으로 변환하여 학습하면 본 연구에서 쓰인 Online Algorithm보다 월등히 좋은 성능을 보일 것이라 기대된다. 추후 연구에서는 회귀모형이 아닌 다른 Online Algorithm들을 적용하면 더 좋은 결과를 이끌어 낼 수 있을 것이다.

**REFRENCE**

[1] Arce, P., & Salinas, L. (2012, November). Online ridge regression method using sliding windows. In *2012 31st International Conference of the Chilean Computer Science Society* (pp. 87-90). IEEE.

[2]Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

[3]Hazan, E., & Seshadhri, C. (2009, June). Efficient learning algorithms for changing environments. In *Proceedings of the 26th annual international conference on machine learning* (pp. 393-400).

[4] McDonald, G. C. (2009). Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, *1*(1), 93-100.

**APPENDIX A**

Source Code

# base tool

import pandas as pd

pd.set\_option('display.max\_columns', 50)

pd.set\_option('display.max\_rows',100)

import numpy as np

from sklearn.model\_selection import train\_test\_split

import copy

import warnings

warnings.filterwarnings('ignore')

#visualization

from IPython.display import display

import matplotlib.pyplot as plt

plt.style.use('default')

from matplotlib import font\_manager, rc

font\_path = "C:/Windows/Fonts/gulim.ttc"

font = font\_manager.FontProperties(fname=font\_path).get\_name()

rc('font', family=font)

import seaborn as sns

import plotly.io as pio

import plotly.express as px

import plotly.graph\_objects as go

import plotly.figure\_factory as ff

from plotly.subplots import make\_subplots

pio.templates.default = "plotly\_white"

#

import datetime

import scipy as sp

# test

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# scaling

from sklearn.preprocessing import StandardScaler

# modeling

from sklearn.linear\_model import LinearRegression,Ridge

from sklearn.ensemble import RandomForestRegressor

import statsmodels.api as sm

# evaluation

from sklearn.metrics import accuracy\_score, r2\_score, mean\_squared\_error, mean\_squared\_error, mean\_absolute\_error

from collections import Counter

# %%

def summary(df, pred=None):

    obs = df.shape[0]

    Types = df.dtypes

    Counts = df.apply(lambda x: x.count())

    Min = df.min()

    Max = df.max()

    Uniques = df.apply(lambda x: x.unique().shape[0])

    Nulls = df.apply(lambda x: x.isnull().sum())

    print('Data shape:', df.shape)

    if pred is None:

        cols = ['Types', 'Counts', 'Uniques', 'Nulls', 'Min', 'Max']

        st = pd.concat([Types, Counts, Uniques, Nulls, Min, Max], axis = 1, sort=True)

    st.columns = cols

    print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nData Types:')

    print(st.Types.value\_counts())

    print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

    return st

# %%

raw\_data = pd.read\_csv('./Tetuan City power consumption.csv')

raw\_data= pd.DataFrame(raw\_data)

# %%

summary(raw\_data)

# %%

heat\_temp = raw\_data.iloc[:,-3:]

r\_column = list(heat\_temp.columns)

heat\_temp.rename(columns={r\_column[0]:'Zone1',r\_column[1]:'Zone2',r\_column[2]:'Zone3'},inplace=True)

# %%

plt.figure(figsize=(5,5))

sns.heatmap(data = heat\_temp.corr(), annot=True,

fmt = '.2f', linewidths=.5, cmap='Blues')

# %%

def getWeekDay(ts):

    days = ['MON','TUE','WED','THU','FRI','SAT','SUN']

    return days[ts.weekday()]

def getMonth(ts):

    months = ['JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP','OCT','NOV','DEC']

    return months[ts.month-1]

# %%

def vif(df:pd.DataFrame)->None:

    print('vif score')

    df = df.iloc[:,:-1]

    vif\_scores = pd.DataFrame()

    vif\_scores["Attribute"] = df.columns

    # calculating VIF for each feature

    vif\_scores["VIF Scores"] = [round(variance\_inflation\_factor(df.values, i) ,2)for i in range(len(df.columns))]

    display(vif\_scores)

    print('-'\*50)

    return vif\_scores

# %%

def acf(data, k):

    data  = np.array(data).reshape(-1)

    mean = data.mean()

    numerator = np.sum((data[:len(data)-k] - mean) \* (data[k:] - mean))

    denominator = np.sum(np.square(data - mean))

    acf\_val = numerator / denominator

    return acf\_val

# %%

def pacf(data, k):

    if k == 0:

        pacf\_val = 1

    else:

        gamma\_array = np.array([acf(data, k) for k in range(1, k + 1)])

        gamma\_matrix = []

        for i in range(k):

            temp = [0] \* k

            temp[i:] = [acf(data, j) for j in range(k - i)]     # making diagonal

            gamma\_matrix.append(temp)

        gamma\_matrix = np.array(gamma\_matrix)

        gamma\_matrix = gamma\_matrix + gamma\_matrix.T - np.diag(gamma\_matrix.diagonal())     # making symmetric matrix

        pacf\_val = np.linalg.inv(gamma\_matrix).dot(gamma\_array)[-1]

    return pacf\_val

# %%

def explained\_variance\_ratio\_plot(explained\_variance\_ratio):

    x\_axis = range(1, len(explained\_variance\_ratio)+1)

    plt.bar(x\_axis, explained\_variance\_ratio,

            align = 'center', label = 'Individual Explained Variance Ratio')

    plt.step(x\_axis, np.cumsum(explained\_variance\_ratio),

             where = 'mid', color='red', label='Cumulative Explained Variance Ratio')

    plt.ylim(0, 1.1)

    plt.xticks(x\_axis)

    plt.xlabel('Principal Components')

    plt.ylabel('Explained Variance Ratio')

    plt.grid()

    plt.show()

# %%

class AAR:

    def \_\_init\_\_(self,gamma:float=1.0):

        self.a\_inv=[]

        self.b=[]

        self.gamma = gamma

    def set\_gamma(self,gamma:float):

        self.gamma = gamma

    def get\_a\_inv(self)->np.array:

        return self.a\_inv

    def get\_b(self)->np.array:

        return self.b

    def get\_expert(self)->np.array:

        a\_inv = self.a\_inv

        b = self.b

        w = b.transpose().dot(a\_inv)

        return w

    def fit(self,X:pd.DataFrame,y:pd.Series)->list:

        '''train chunk of X and y, collect and return prediction from every step as a list'''

        # todo:  check x,y, gamma

        X\_matrix = X.to\_numpy()

        gamma = self.gamma

        length = X\_matrix.shape[1]

        a\_inv = np.linalg.inv(gamma \*np.identity(length))

        b = np.zeros(length).reshape(length,1)

        pred = []

        for i in range(X\_matrix.shape[0]):

            x\_vec = X\_matrix[i].reshape(length,1)

            a\_inv = cal\_inv(a\_inv,x\_vec)

            y\_pred = b.transpose().dot(a\_inv).dot(x\_vec).item()

            pred.append(y\_pred)

            b = b+y[i]\*x\_vec

        #객체 변수 업데이트

        self.a\_inv = a\_inv

        self.b = b

        self.gamma = gamma

        return pred

    def update\_a\_then\_predict(self,x:np.ndarray)->np.ndarray:

        #todo check parameter shape and reshape

        # set A\_inverse and b

        length = len(x)

        a\_inv = self.a\_inv

        b = self.b

        gamma = self.gamma

        # reshape x

        x = x.reshape((length,1))

        if len(a\_inv) == 0:

            a\_inv = np.linalg.inv(gamma \*np.identity(length))

            b = np.zeros(length).reshape(length,1)

            self.b = b

        # update with x\_t

        a\_inv = cal\_inv(a\_inv,x)

        self.a\_inv = a\_inv

        # predict

        y\_tj = b.transpose().dot(a\_inv).dot(x)

        return y\_tj

    def update\_b(self,x:np.ndarray,y:float)->None:

        # check parameter

        # reshape x

        x = x.reshape((len(x),1))

        b = self.b

        b+=y\*x

        self.b  = b

class FLH:

# X: pd.DataFrame,y: pd.Series , model:AAR,gamma:float =1.0,

    def \_\_init\_\_(self,model,alpha:float=1.0):

        self.model = model

        self.alpha =alpha

        self.v\_list = [1]

        self.model\_list = []

        self.pred\_result\_list =[]

    def set\_alpha(self,alpha:float)->None:

        self.alpha =alpha

    def fit(self, X: pd.DataFrame,y: pd.Series )->None:

        model  = self.model

        alpha = self.alpha

        v\_list = self.v\_list

        pred\_result\_list = self.pred\_result\_list

        # initialize v\_list, model\_list

        v\_list = [1]

        model\_list = []

        pred\_result\_list = []

        # prepare data

        X = X.to\_numpy()

        for t in range(len(y)):

            model\_list.append(model)

            pred\_t\_j=[]

            # 4,5 update a and predict

            for m in model\_list:

                # print(X[t])

                y\_t\_j = m.update\_a\_then\_predict(x=X[t]).item()

                pred\_t\_j.append(y\_t\_j)

            #6 calculate prediction

            pred = np.multiply(v\_list,pred\_t\_j).sum()

            pred\_result\_list.append(pred)

            #7 update b

            for m in model\_list:

                m.update\_b(X[t],y[t])

            #8 update v\_t1\_i

            v\_loss = [v\_list[i]\*np.exp(-1\*alpha\*np.power((y[t].item()-pred\_t\_j[i]),2)) for i in range(len(pred\_t\_j))]

            v\_list = np.multiply(v\_loss,1/sum(v\_loss))

            #9 update v\_t1\_t1

            v\_t1\_t1 = 1/(t+2)

            v\_list = list(v\_list\*(t+1)/(t+2))

            v\_list.append(v\_t1\_t1)

        self.pred\_result\_list = pred\_result\_list

        self.v\_list = v\_list

        self.model\_list = model\_list

    def r2\_score(self)->float:

        return round(r2\_score(y,np.array(self.pred\_result\_list)),4)

def cal\_inv(m\_inv:np.ndarray,x:np.ndarray)->np.ndarray:

    new\_inv = m\_inv-(m\_inv.dot(x)).dot(m\_inv.dot(x).transpose())/(1+x.transpose().dot(m\_inv).dot(x).item())

    return new\_inv

# %% [markdown]

# # 시게열적 특성

# %%

t = []

x = range(3000)

for n in x:

    t.append(acf(raw\_data['Zone 1 Power Consumption'],n))

plt.plot(x,t)

# %%

import statsmodels.graphics.tsaplots as sgt

## Close

fig = plt.figure()

fig.set\_size\_inches(10,10)

ax1 = fig.add\_subplot(2, 1, 1)

ax2 = fig.add\_subplot(2, 1, 2)

sgt.plot\_acf(raw\_data['Zone 1 Power Consumption'], lags = 20, zero = False, ax=ax1)

ax1.set\_title("ACF 10 Minute Power Consumption")

sgt.plot\_pacf(raw\_data['Zone 1 Power Consumption'], lags = 20, zero = False, method = ('ols'), ax=ax2)

ax2.set\_title("PACF 10 Minute Power Consumption")

plt.show()

# %% [markdown]

# # 하루 총소모량으로 변환

# %%

raw\_data

# %%

raw\_data['date']=raw\_data['DateTime'].map(lambda x: x.split()[0])

col = ['date','weekDay','month','maxTemp','minTemp','meanTemp','maxHum','minHum','meanHum','maxWind','mimWind','meanWind','maxGenDiff','minGenDiff','meanGenDiff','manDiff','minDiff','meanDiff','powerConsumption']

df = pd.DataFrame(columns=col)

uni\_date= set(raw\_data['date'])

for day in uni\_date:

    dateStamp = datetime.datetime.strptime(day,'%m/%d/%Y')

    tmp = raw\_data[raw\_data['date']==day]

    date=day

    weekDay = getWeekDay(dateStamp)

    month = getMonth(dateStamp)

    maxTemp=tmp['Temperature'].max()

    minTemp=tmp['Temperature'].min()

    meanTemp=tmp['Temperature'].mean()

    maxHum=tmp['Humidity'].max()

    minHum=tmp['Humidity'].min()

    meanHum=tmp['Humidity'].mean()

    maxWind=tmp['Wind Speed'].max()

    mimWind=tmp['Wind Speed'].min()

    meanWind=tmp['Wind Speed'].mean()

    maxGenDiff=tmp['general diffuse flows'].max()

    minGenDiff=tmp['general diffuse flows'].min()

    meanGenDiff=tmp['general diffuse flows'].mean()

    manDiff=tmp['diffuse flows'].max()

    minDiff=tmp['diffuse flows'].min()

    meanDiff=tmp['diffuse flows'].mean()

    powerConsumption=round((tmp['Zone 1 Power Consumption'].sum()+tmp['Zone 2  Power Consumption'].sum()+tmp['Zone 3  Power Consumption'].sum())/1000000,3)

    dt = [date,weekDay,month,maxTemp,minTemp,meanTemp,maxHum,minHum,meanHum,maxWind,mimWind,meanWind,maxGenDiff,minGenDiff,meanGenDiff,manDiff,minDiff,meanDiff,powerConsumption]

    dt\_insert = {}

    for idx,c in enumerate(col):

        dt\_insert[c]=dt[idx]

    df = df.append(dt\_insert,ignore\_index=True)

# %%

df = df.drop(columns=['weekDay','month'])

# %%

df

# %%

summary(df)

# %%

t = []

x = range(300)

for n in x:

    t.append(acf(df['powerConsumption'],n))

plt.plot(x,t)

# %%

import statsmodels.graphics.tsaplots as sgt

## Close

fig = plt.figure()

fig.set\_size\_inches(10,10)

ax1 = fig.add\_subplot(2, 1, 1)

ax2 = fig.add\_subplot(2, 1, 2)

sgt.plot\_acf(df['powerConsumption'], lags = 20, zero = False, ax=ax1)

ax1.set\_title("ACF Daily Power Consumption")

sgt.plot\_pacf(df['powerConsumption'], lags = 20, zero = False, method = ('ols'), ax=ax2)

ax2.set\_title("PACF Daily Power Consumption")

plt.show()

# %%

sns.pairplot(df.iloc[:,1:])

# %%

df.shape

# %%

vif(df.iloc[:,1:])['VIF Scores'].argmax()

# %%

tmp = df.drop(columns=['meanTemp','meanHum','maxTemp','maxHum','maxGenDiff','meanWind','manDiff','minTemp','minDiff'])

vif(tmp.iloc[:,1:])['VIF Scores']

#선형성이 가장 높은 temperature 데이터가 모두 제거되었다. -> PCA loading

# %%

df

# %%

# 표준화

ss = StandardScaler()

result\_s = pd.DataFrame(ss.fit\_transform(df.iloc[:,1:]),columns=[df.iloc[:,1:].columns])

X\_s = result\_s.iloc[:,:-1]

y = result\_s.iloc[:,-1]

# %%

X\_s

# %%

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

pca = PCA(n\_components=X\_s.shape[1])

pca.fit(X\_s)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

explained\_variance\_ratio\_plot(explained\_variance\_ratio)

# %%

pca = PCA(n\_components=0.95)

X\_reduced = pca.fit\_transform(X\_s)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

explained\_variance\_ratio\_plot(explained\_variance\_ratio)

X\_reduced = pd.DataFrame(X\_reduced,columns=['component1','component2','component3','component4','component5','component6','component7','component8'])

# %%

pca = PCA(n\_components=0.80)

X\_reduced = pca.fit\_transform(X\_s)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

explained\_variance\_ratio\_plot(explained\_variance\_ratio)

X\_reduced = pd.DataFrame(X\_reduced,columns=['component1','component2','component3','component4'])

# %%

X\_reduced.merge(y,left\_index=True,right\_index=True)

# %%

X\_train,X\_test, y\_train,y\_test = train\_test\_split(X\_reduced,y)

# %%

lr =LinearRegression()

lr.fit(X\_train,y\_train)

print(lr.score(X\_train,y\_train))

print(lr.score(X\_test,y\_test))

print(mean\_squared\_error(y\_test ,lr.predict(X\_test))\*\*0.5)

print(mean\_absolute\_error(y\_test ,lr.predict(X\_test)))

# %%

rd =Ridge(alpha=1.0)

rd.fit(X\_train,y\_train)

print(rd.score(X\_train,y\_train))

print(rd.score(X\_test,y\_test))

print(mean\_squared\_error(y\_test ,rd.predict(X\_test))\*\*0.5)

print(mean\_absolute\_error(y\_test ,rd.predict(X\_test)))

# %%

rf =RandomForestRegressor(max\_depth=6,max\_leaf\_nodes=40)

rf.fit(X\_train,y\_train)

print(rf.score(X\_train,y\_train))

print(rf.score(X\_test,y\_test))

print(mean\_squared\_error(y\_test ,rf.predict(X\_test))\*\*0.5)

print(mean\_absolute\_error(y\_test ,rf.predict(X\_test)))

# %%

model2 = sm.OLS(y,X\_reduced)

result2 = model2.fit()

print(result2.summary())

sp.stats.probplot(result2.resid, plot=plt)

plt.show()

test = sm.stats.omni\_normtest(result2.resid)

plt.hist(y1)

plt.show()

# print(r2\_score(y\_test, result2.predict(X\_test)))

print(mean\_squared\_error(y,result2.predict(X\_reduced)))

# %%

X\_tmp =X\_reduced.iloc[:,[0,2,3,4,5]]

model2 = sm.OLS(y,X\_tmp)

result2 = model2.fit()

print(result2.summary())

sp.stats.probplot(result2.resid, plot=plt)

plt.show()

test = sm.stats.omni\_normtest(result2.resid)

plt.hist(y1)

plt.show()

# print(r2\_score(y\_test, result2.predict(X\_test)))

print(mean\_squared\_error(y,result2.predict(X\_tmp)))

# %%

X\_train,X\_test ,y\_train,y\_test = train\_test\_split(X\_tmp,y)

rf =RandomForestRegressor(max\_depth=6,max\_leaf\_nodes=8)

rf.fit(X\_train,y\_train)

print(rf.score(X\_train,y\_train))

print(rf.score(X\_test,y\_test))

print(mean\_squared\_error(y\_test ,rf.predict(X\_test))\*\*0.5)

print(mean\_absolute\_error(y\_test ,rf.predict(X\_test)))

# %%

flh = FLH(AAR(gamma=15),alpha=15)

flh.fit(X\_reduced,y)

flh.r2\_score()

# %%

# set grid search parameters

gamma\_num = 0

gammas= []

alpha\_num = 0

alphas= []

for \_ in range(10):

    gamma\_num+=0.1

    gammas.append(round(gamma\_num,2))

    alpha\_num+=5

    alphas.append(round(alpha\_num,2))

#grid search

grid=[]

result = []

for alpha in alphas:

    # print(alpha)

    g\_result =[]

    for gamma in gammas:

        # print(gamma)

        arr = AAR(gamma=gamma)

        flh = FLH(arr,alpha)

        flh.fit(X\_reduced,y)

        score = round(flh.r2\_score(),4)

        g\_result.append(score)

        grid.append({'alpha':alpha,'gamma':gamma,'score':score, 'prediction':flh.pred\_result\_list})

        del arr

        del flh

    result.append(g\_result)

# find best hyper parameters

s=[]

for d in grid:

    s.append(d['score'])

print(grid[s.index(max(s))])

#plot

import plotly.graph\_objects as go

import pandas as pd

import numpy as np

# Read data from a csv

z = result

sh\_0, sh\_1 = len(result[0]),len(result)

x, yd = np.linspace(0, 1, sh\_0), np.linspace(0,50, sh\_1)

fig = go.Figure(data=[go.Surface(z=z, x=x, y=yd)])

fig.update\_layout(title='FLH\_AAR grid search (x = gamma , y = alpha) ', autosize=False,

                  width=500, height=500,

                  margin=dict(l=65, r=50, b=65, t=90))

fig.show()

# %%

aar = AAR()

pred = aar.fit(X\_reduced,y)

r2\_score(y,pred)

# %%

y

# %%

pred

# %%

pca = PCA(n\_components=1)

pca.fit(X\_s)

newIndex = pca.fit\_transform(X\_s)

print(newIndex)

pc1 = newIndex[:, 0]

# Coefficients

loadings = pd.DataFrame(pca.components\_.T, columns = ['PC1'], index = df.iloc[:,1:-1].columns)

result = pd.DataFrame(X\_s)

result['SRVR'] = y

result['PC1'] = pc1

# print(result)

abs(result.corr()['PC1'])

# %%

sns.pairplot(raw\_data.iloc[:2000,1:-2])

# %%

raw\_data['general diffuse flows'].value\_counts()

# %%

raw\_data['Wind Speed'].value\_counts()

# %%

vif(raw\_data.iloc[:,1:-2])

# %%

# 주기성 확인 auto correlation ACF plot

# 계절 변수

# lag? PACF확인

# 이상치

# 정규성

# %%

raw\_data.iloc[:,1:-2].corr()

# %%

X = raw\_data.iloc[:,1:-3]

y = raw\_data.iloc[:,-3:]

y1 = raw\_data.iloc[:,-3]

y2 = raw\_data.iloc[:,-2]

y3 = raw\_data.iloc[:,-1]

# %%

plt.plot(y[:1000])

# %% [markdown]

# # Baseline

# %%

X\_pre\_train = X.iloc[:,:]

y\_pre\_train = y.iloc[:,0]

# %%

y\_pre\_train

# %%

ss = StandardScaler()

X\_pre\_train\_s = ss.fit\_transform(X\_pre\_train)

X\_pre\_train\_s = pd.DataFrame(X\_pre\_train\_s,columns=X\_pre\_train.columns)

X\_pre\_train\_s

# %%

model2 = sm.OLS(y\_pre\_train,X\_pre\_train)

result2 = model2.fit()

print(result2.summary())

sp.stats.probplot(result2.resid, plot=plt)

plt.show()

test = sm.stats.omni\_normtest(result2.resid)

plt.hist(y1)

plt.show()

# print(r2\_score(y\_test, result2.predict(X\_test)))

print(mean\_squared\_error(y\_pre\_train,result2.predict(X\_pre\_train)))

# %%

result2.predict(X\_pre\_train)

# %%

y\_pre\_train

# %%

resid = result2.resid

yhat = result2.predict(X)

sns.scatterplot(yhat,resid,alpha=0.2)

# %%

summary(X)

1. [1] https://archive.ics.uci.edu/ml/datasets/Power+consumption+of+Tetouan+city [↑](#footnote-ref-1)