기상청

[Python을 활용한 분석교육실습]

ASOS(종관기상관측) 기상 자료를 활용한 증발량 산출 모형 분석 사례

기상청

기상기후
빅데이터 분석 플랫폼

분석 개요

분석 교육 실습 주제인 ASOS(종관기상관측장비)와 증발량에 대해 알아봅니다.

● 종관기상관측장비(ASOS)

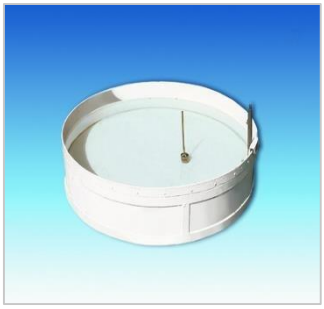
- 기상청은 서울기상관측소를 비롯하여 전국 102개소의 종관기상관측장비(ASOS)와 무인으로 운영되는 510개소의 자동기상관측장비(AWS)를 이용하여 지상기상관측업무를 수행하고있다.(2018년 기준) 종관기상 관측 장비(ASOS)는 지방청, 기상대, 관측소에 설치되어 기상상태를 관측하고 국제전문 작성 및 통계표 작성과 같은 관측업무를 자동으로 처리한다.
- 관측방법은 기압, 기온, 풍향, 풍속, 습도, 강수량, 강수유무, 일사량, 일조시간, 지면온도, 초상온도, 지중온도, 토양수분, 지하수위 14개 요소에 대해서는 자동으로 관측하고, 시정, 구름, 증발량, 일기 현상 등은 일부 자동과 목측(目測)으로 관측한다.
- 증발량은 일정기간 동안 단위 면적에서 증발된 물의 양을 mm 단위로 관측한 것이다. 일반적인 대기상태의 증발량을 대표적으로 관측하기 위해서는 증발계를 설치하여 관측한다.
- 증발량는 크게 소형증발계로 관측한 소형증발량과 대형증발계로 관측한 대형증발량으로 구분한다.

● 소형 증발량

- 증발량을 측정하기 위해 지름 20cm, 깊이10cm 원통형 용기에 일정한 물(20mm)을 채우고 관측기간동안 물의 양이 변화한 정도를 우량되 또는 강수저울을 이용해 증발량을 구한다.
- 현재 기상청에서는 09시(00UTC)를 기준으로 증발량을 관측하고 있다.
- 비가 내렸을 경우에는 비의 양을 감하여 증발량을 구한다.

● 대형 증발량

- 증발량을 측정하기 위해 지름 120cm, 깊이 25cm의 흰색 페인트를 칠한 원통형 용기에 일정한 물을 채우고 수위측정기에 의해 관측기간동안 물의 양이 변화한 정도를 측정한다.
- 겨울철에 수면이 빙결되면 증발량 측정이 불가능할 뿐만 아니라 증발계 자체가 파손될 우려가 있으므로 겨울철에는 철거하여 옥내에 보존한다.



분석 시나리오

실습할 예제는 ASOS(종관기상관측장비) 기상 데이터와 비기상 데이터를 활용하여 소형증발량을 산출하는 모형을 구축하는 것입니다. 실습 예제를 통해 현재 목측관측으로 관측하고 있는 소형증발량을 자동으로 산출하는 모형을 구축해봅니다.

● 증발량 산출 모형 구축 시나리오

[분석 활용 데이터] **[Analytics]**

기상 데이터(ASOS)

- 소형 증발량
- 기온, 강수, 바람, 습도, 일사/일조, 눈, 구름,지면/지중온도
- (파생변수) 강수유무, 2m풍속

비기상 데이터

- (지리적변수) 관측 지점 높이, 풍속계 높이

증발량 영향 변수 선택

정의	변수 중요도가 높은 변수를 기준으로 순서대로 상관계수가 0.45 이상인 변수를 제거하여 후보변수를 도출하고, 모든 후보변수 조합에서 가장 최적의 예측력을 갖는 변수 조합을 선택
분석방법	랜덤포레스트(Random Forest) 피어슨 상관분석
분석결과	• 증발량 영향 최종 변수

증발량 산출 모형 구축

정의	랜덤포레스트(Random Forest) 기법을 활용하여 일별 증발량을 산출하는 산출식 도출
분석방법	랜덤포레스트(Random Forest)
분석결과	•증발량 산출 모형

분석 절차

실습은 데이터 로딩, 데이터 탐색, 데이터 처리, 모형 구축, 모형 검증의 단계에 따라 진행될 예정입니다.

● 증발량 산출 모형 구축절차

	[실습 설명]	[실습 단계]
1 데이터 로딩	분석환경을 설정하고 분석에 필요한 기상데이터 및 비기상데이터를 로딩하여 분석에 필요한 데이터를 준비하는 단계	1. 분석 환경 설정 및 패키지 로딩 2. 데이터 불러오기 3. 데이터 추출 및 결합하기
2 데이터 탐색	분석 데이터의 요약통계를 살펴보고 박스플롯 및 히스토그램을 통해 데이터의 분포를 확인하는 단계	1. 요약통계 보기 2. 박스플롯 그리기 3. 히스토그램 그리기
3 데이터 처리	이상치를 처리하고 결측치를대체하며 파생변수를 생성하여 최종 데이터셋을 구성하는 단계	1. 이상치 처리 2. 결측치 처리 3. 파생변수 생성
4 모형 구축	증발량 영향 변수를 선택하고 모형의 파라미터를 설정하여 증발량 산출 모형을 구축하는 단계	1. 변수 선택 2. 모형 구축
5 모형 검증	최종 구축한 산출 모형의 성능을 검증하는 단계	1. 모형 성능 검증 2. 교차 검증

분석 데이터

증발량 산출 모형 구축에 사용된 파일 및 변수 정보를확인합니다.

- **증발량 산출 모형 구축에 사용된 파일 및 변수정보**

파일명	파일설명	변수명	변수 설명	형식	예제
ASOS_DATA	종관기상 관측장비 (ASOS) 관측 자료	STN_ID	관측지점번호	int	95
		TM	관측날짜	Int	20060101
		TA_AVG	평균기온	num	-12.03
		TA_MAX	최고기온	num	-2.5
		TA_MIN	최저기온	num	-19.1
		SUM_RN	합계강수량	num	0
		WS_AVG	평균풍속	num	0.99
		WS_MAX	최대풍속	num	3.4
		HM_AVG	평균상대습도	num	55.7
		HM_MIN	최소상대습도	num	24
		SUM_SS	합계일조시간	num	8.5
		SUM_SI	합계일사량	num	11.72
		TD_AVG	평균이슬점온도	num	-21.7
		PV_AVG	평균증기압	num	1.11
		PA_AVG	평균현지기압	num	1008
		PS_AVG	평균해면기압	num	1028
		PS_MAX	최대해면기압	num	1030
		PS_MIN	최소해면기압	num	1026
		SD_HR3_MAX	일최심신적설	num	0
		SD_TOT_MAX	일최심적설	num	0
		CA_TOT_AVG	평균전운량	num	0.11
		CA_MID_AVG	평균중하층운량	num	0.11
		TS_AVG	평균지면온도	num	-8.6
		TS_MAX	최대지면온도	num	9.1
		TS_MIN	최소지면온도	num	-14.6
		SUM_SML_EV	소형증발량	num	1.4
GEO_DATA	종관기상 관측장비 (ASOS) 관측 지점별 이력 정보	STN_ID	관측지점번호	int	95
		TM_ST	관측 시작 날짜	char	2006.01.01 00:00
		TM_ED	관측 종료 날짜	char	2100.12.31 00:00
		HT	관측지점 높이	num	154.0
		HT_WD	풍속계 높이	num	18.4

기상청

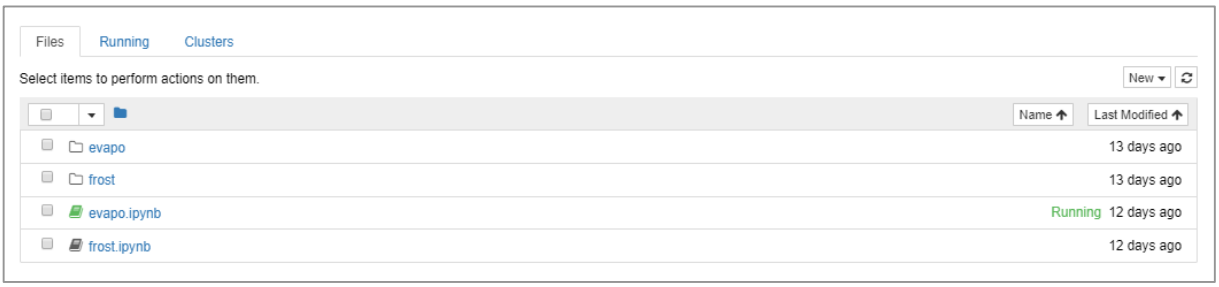
Ⅰ. 데이터 로딩

1. 분석 환경 설정 및 패키지 로딩
2. 데이터 불러오기
3. 데이터 추출 및 결합하기

1. 분석 환경 설정 및 패키지 로딩 (1/2)

- **실습 실행 예제 파일 경로**

  - 분석을 실행하기 위한 예제 파일 경로



  - **evapo.ipynb 파일을 클릭**

- **패키지 로딩**

  - 분석을 위해 사용할 패키지를 로딩

```
import os
import sys
import glob
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import h2o
import array
import numpy as np
import re

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
from h2o.estimators.gbm import H2OGradientBoostingEstimator
from h2o.estimators.random_forest import H2ORandomForestEstimator
from h2o.estimators import H2ORandomForestEstimator
from h2o.grid.grid_search import H2OGridSearch
from missingpy import KNNImputer
```

  - **Import로 사용할 패키지 및 모듈 로딩**

1. 분석 환경 설정 및 패키지 로딩 (2/2)

- **분석 환경 설정**

- 분석을 실행하기 전 메모리를 초기화 하고 현재 설정된 디렉토리를 확인

```
#============================================================
# 분석 환경 셋팅
#============================================================
#Python 메모리에 생성된 모든 객체 삭제 (초기화)
sys.stdout.flush()

#============================================================
# 작업 디렉토리 경로 확인
#============================================================
currentPath=os.getcwd()

#현재 위치한 디렉토리 경로확인
print('Current working dir : %s' % currentPath)
```

- **sys.stdout.flush()로 Python 메모리 초기화**
- **os.getcwd() 로 현재 설정된 디렉토리 위치를 확인**
- **print 문을 사용하여 현재 설정 위치를 확인**

2. 데이터불러오기

- **데이터 불러와 구조 확인하기**

- 분석에 활용할 종관기상관측장비(ASOS) 관측 자료 데이터와 각 ASOS 관측 지점별 이력 정보 데이터를 불러옴

```
#============================================================
# 기상 데이터 읽어오기
#============================================================
#loading  weather data
ASOS_DATA = pd.read_csv(currentPath + '/evapo/ASOS_DATA.csv', encoding='euc-kr')
#loading geographical data
GEO_DATA = pd.read_csv(currentPath + '/evapo/GEO_DATA.csv', encoding='euc-kr')
#============================================================
# 불러온 데이터 구조 확인하기
#============================================================
ASOS_DATA.info()
GEO_DATA.info()
```

- **pd.read_csv()로 기상 및 지리정보 데이터 불러옴**
- **info()로 데이터 구조 확인**

> 실행 결과

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58005 entries, 0 to 58004
Data columns (total 26 columns):
STN_ID        58005 non-null int64
TM            58005 non-null int64
TA_AVG        58005 non-null float64
TA_MAX        58005 non-null float64
TA_MIN        58005 non-null float64
SUM_RN        18255 non-null float64
WS_AVG        58005 non-null float64
WS_MAX        58005 non-null float64
HM_AVG        58005 non-null float64
HM_MIN        58005 non-null float64
SUM_SS        53563 non-null float64
SUM_SI        57962 non-null float64
TD_AVG        58005 non-null float64
PV_AVG        58005 non-null float64
PA_AVG        58005 non-null float64
PS_AVG        58005 non-null float64
PS_MAX        58005 non-null float64
PS_MIN        58005 non-null float64
SD_HR3_MAX    2358 non-null float64
SD_TOT_MAX    4182 non-null float64
CA_TOT_AVG    58005 non-null float64
CA_MID_AVG    58005 non-null float64
TS_AVG        58005 non-null float64
TS_MAX        58005 non-null float64
TS_MIN        58005 non-null float64
SUM_SML_EV    58005 non-null float64
dtypes: float64(24), int64(2)
memory usage: 11.5 MB
```

3. 데이터 추출 및결합하기(1/2)

- **데이터 추출하기**

- 지점정보이력의 STN_ID(지점번호), TM_ST(지점시작날짜), TM_ED(지점종료날짜), HT(관측지점 높이), HT_WD(풍속계높이)컬럼만 추출

```
#=============================================================
# 필요한 컬럼만 추출하기
#=============================================================
GEO_DATA = GEO_DATA.loc[:, ['STN_ID', 'TM_ED', 'TM_ST', 'HT', 'HT_WD']]
```

- **loc로 필요한 컬럼만 추출**

- **데이터 타입 변환**

- 지점시작날짜, 지점종료날짜를 숫자 타입으로 변환

```
#=============================================================
# TM_ED(지점종료날짜), TM_ST(지점시작날짜) 숫자 데이터 타입으로 변환
#=============================================================
# 데이터 변환 및 추출
GEO_DATA['TM_ST'] = GEO_DATA['TM_ST'].str.replace('.', '')
GEO_DATA['TM_ED'] = GEO_DATA['TM_ED'].str.replace('.', '')

GEO_DATA['TM_ST'] = GEO_DATA['TM_ST'].str.slice(0, 8)
GEO_DATA['TM_ED'] = GEO_DATA['TM_ED'].str.slice(0, 8)

# str을 int 로 변환
GEO_DATA['TM_ST'] = GEO_DATA['TM_ST'].astype(int)
GEO_DATA['TM_ED'] = GEO_DATA['TM_ED'].astype(int)

# 데이터 타입 확인
GEO_DATA.dtypes
```

- **replace()로 “.”을 공백(“”)으로 치환 후, slice로 필요한 부분만 추출**
- **astype(int)으로 문자열을 숫자로 변환**
- **dtypes 를 이용하여 데이터 타입 확인**

3. 데이터 추출 및결합하기(2/2)

- **데이터 결합하기**

- 기상데이터에 지점정보이력의 HT(관측지점높이) 컬럼을 결합
- 146(전주) 지점의 경우, 지점 위치의 이동으로 관측 지점 높이가 달라지기 때문에, 2015-07-01 시점을 기준으로 각각의 관측지점높이를 결합
(~2015-06-30까지 HT= 53.40 / 2015-07-01~현재 HT= 61.40)

```
#============================================================
# 기상데이터와 HT(관측지점높이) 결합
#============================================================
#146(전주)지점 제외하고 결합
mergeGeo = GEO_DATA.loc[GEO_DATA['STN_ID'] != 146, ['STN_ID','HT']]

# STNID 컬럼으로 asos 데이터와 geo 데이터 병합
DATA = pd.merge(ASOS_DATA, mergeGeo, how='left', on='STN_ID')

#146(전주)지점 관측기간에 맞는 HT 데이터 결합하기
DATA.at[(DATA['STN_ID'] == 146) & (DATA['TM'] < 20150701), 'HT'] = 53.40
DATA.at[(DATA['STN_ID'] == 146) & (DATA['TM'] >= 20150701), 'HT'] = 61.40

#결과 확인하기
DATA.loc[(DATA['STN_ID'] == 146) & (DATA['TM'] < 20150701),
['STN_ID','TM','HT']].head()
DATA.loc[(DATA['STN_ID'] == 146) & (DATA['TM'] >= 20150701),
['STN_ID','TM','HT']].head()
```

- **merge()로 두개의 데이터 프레임을 조인키를 이용하여 결합**
- **at으로 시점 기준에 맞는 각각의 관측지점 높이를 결합**
- **head()로 결합한 데이터를 출력하여 결과확인**

> 실행 결과

	STN_ID	TM	HT
39768	146	20060102	53.4
39769	146	20060103	53.4
39770	146	20060104	53.4
39771	146	20060105	53.4
39772	146	20060106	53.4

	STN_ID	TM	HT
43232	146	20150701	61.4
43233	146	20150702	61.4
43234	146	20150703	61.4
43235	146	20150704	61.4
43236	146	20150705	61.4

기상청

Ⅱ. 데이터 탐색

1. 요약통계 보기
2. 박스플롯 그리기
3. 히스토그램 그리기

1. 요약통계보기(1/2)

● **요약통계 한번에 보기**

- describe 함수를 사용하여 요약통계량 한번에보기

```
#============================================================
# 요약통계 한번에 보기
#============================================================
DATA.describe()
```

▪ **describe()로 기술통계량을 살펴봄**

- count, mean, std, min, 25%, 50%, 75%, max 제공

> 실행 결과

	STN_ID	TM	TA_AVG	TA_MAX	TA_MIN	SUM_RN	WS_AVG	WS_MAX	HM_AVG	HM_MIN	...	
count	58005.000000	5.800500e+04	58005.000000	58005.000000	58005.000000	18255.000000	58005.000000	58005.000000	58005.000000	58005.000000	...	580
mean	131.809654	2.010538e+07	13.195650	17.434673	9.656818	11.762350	2.463257	4.800527	67.734992	47.173784	...	10
std	25.620342	2.858221e+04	9.724377	9.764689	10.115668	21.630471	1.449396	2.236551	15.946926	18.935333	...	
min	95.000000	2.006010e+07	-18.770000	-11.400000	-26.700000	0.100000	0.040000	0.300000	9.920000	4.000000	...	9
25%	112.000000	2.008063e+07	5.070000	9.200000	1.400000	0.800000	1.450000	3.300000	56.750000	32.000000	...	10
50%	131.000000	2.010122e+07	14.390000	19.000000	10.500000	3.600000	2.100000	4.400000	69.040000	46.000000	...	10
75%	159.000000	2.013061e+07	21.590000	25.700000	18.400000	13.100000	3.080000	5.800000	79.750000	61.000000	...	10
max	184.000000	2.015123e+07	32.720000	37.800000	30.300000	372.500000	16.650000	27.200000	100.000000	100.000000	...	10

8 rows × 27 columns

1. 요약통계보기(2/2)

- **요약통계 그룹별로 보기**

- groupby 함수를 사용하여 지점별로 묶어 요약통계 보기

```
#============================================================
# 지점별 요약통계 보기
#============================================================
grouped = DATA.groupby('STN_ID')
grouped.describe()
```

- **groupby() 로 지점별 데이터를 묶음**
- **묶은 데이터를 describe()로 지점별로 요약통계 보기**

> 실행 결과

	CA_MID_AVG								CA_TOT_AVG		...	WS_AVG		WS_MAX					
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%
STN_ID																				
95	3284.0	3.486934	2.610688	0.0	1.00	3.33	5.78	9.56	3284.0	5.147329	...	2.2800	6.93	3284.0	4.232887	1.562133	0.8	3.1	4.1	5.2000
101	3649.0	3.658523	2.578347	0.0	1.22	3.56	5.89	10.00	3649.0	5.274050	...	1.5000	4.45	3649.0	2.863031	1.168940	0.5	2.0	2.7	3.6000
104	2700.0	3.326896	2.545435	0.0	1.00	3.00	5.56	9.67	2700.0	5.231570	...	2.8625	7.68	2700.0	4.410296	1.720339	1.2	3.1	4.1	5.4000
105	942.0	3.154958	2.554125	0.0	0.67	2.78	5.22	8.78	942.0	4.845234	...	2.6100	6.12	942.0	4.321773	1.604092	1.5	3.2	3.9	5.1000
108	3651.0	3.104960	2.447981	0.0	0.78	2.78	5.22	10.00	3651.0	4.887127	...	3.0000	7.97	3651.0	4.711422	1.333599	1.9	3.8	4.5	5.4000
112	3649.0	3.094204	2.450807	0.0	0.78	2.78	5.11	10.00	3649.0	4.873952	...	3.5300	9.33	3649.0	5.200885	1.963111	0.3	3.8	4.7	6.1000
115	3645.0	4.739890	2.473004	0.0	2.89	5.00	6.78	9.89	3645.0	6.411435	...	4.7800	16.65	3645.0	6.920126	2.821178	0.6	4.8	6.6	8.6000
119	3646.0	3.158719	2.440917	0.0	0.89	2.89	5.22	10.00	3646.0	4.916827	...	2.1800	7.58	3646.0	3.890063	1.231474	1.0	3.0	3.7	4.5675
129	3651.0	3.297072	2.418522	0.0	1.11	3.11	5.33	9.56	3651.0	5.182629	...	3.1300	8.79	3651.0	5.155163	1.806325	1.4	3.9	4.8	6.2000
131	3649.0	3.222664	2.339529	0.0	1.11	3.11	5.33	8.67	3649.0	5.119901	...	1.7500	4.93	3649.0	2.988819	0.947637	0.8	2.3	2.8	3.5000
133	3651.0	3.107406	2.287218	0.0	1.00	2.89	5.11	8.89	3651.0	5.003391	...	2.0900	6.63	3651.0	3.628348	1.308695	1.0	2.6	3.5	4.5000
136	3651.0	3.111115	2.387888	0.0	0.89	2.89	5.22	8.89	3651.0	5.020310	...	2.0400	5.65	3651.0	3.787866	1.424143	1.0	2.7	3.7	4.7000
146	3648.0	3.324405	2.427794	0.0	1.00	3.11	5.56	9.11	3648.0	5.150186	...	2.1400	6.23	3648.0	3.935447	1.270900	1.1	3.0	3.8	4.6000
159	3640.0	2.887052	2.526757	0.0	0.44	2.33	5.11	9.56	3640.0	4.813069	...	3.9000	9.32	3640.0	5.813516	1.907266	1.9	4.4	5.5	6.8000
165	3650.0	3.377734	2.338273	0.0	1.22	3.44	5.33	9.56	3650.0	5.432244	...	3.9500	11.68	3650.0	6.145096	2.520441	1.5	4.3	5.6	7.6000
168	3651.0	2.853881	2.290083	0.0	0.67	2.56	4.89	9.44	3651.0	4.911131	...	5.2050	15.37	3651.0	7.437168	2.815227	1.8	5.4	7.1	9.1000
184	3648.0	3.979287	2.283147	0.0	2.11	4.11	5.78	9.00	3648.0	6.034224	...	3.9700	10.17	3648.0	5.656935	1.950190	2.1	4.2	5.3	6.6000

17 rows × 208 columns

2. 박스플롯그리기

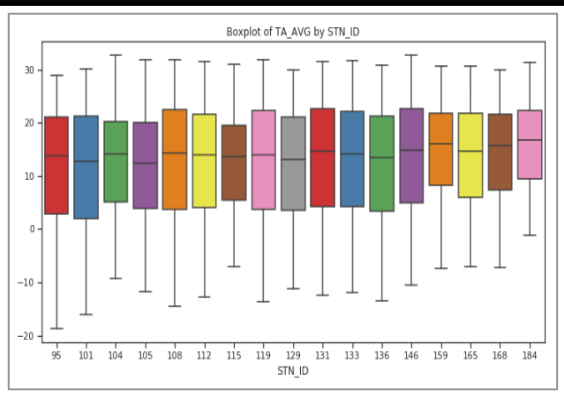
- **지점별 박스플롯 그리기**

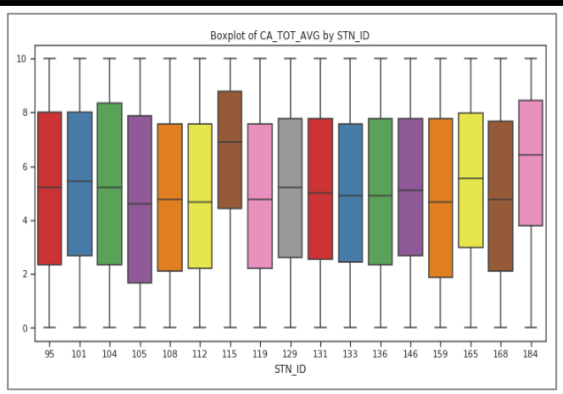
- 분석 데이터를 지점별로 박스플롯을 그려 분포를 확인

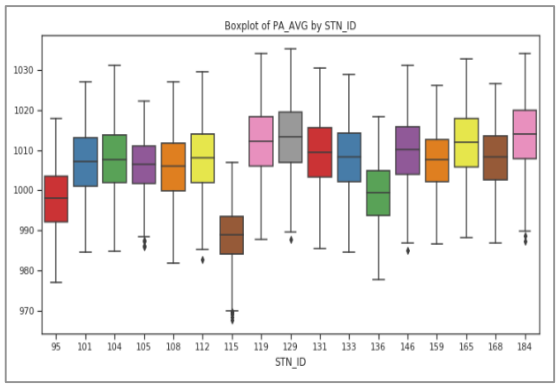
```
#============================================================
# STN별 박스플롯 그리기
#============================================================
sns.set()
sns.set_style("ticks")
%matplotlib inline

for col in DATA.columns.difference(['STN_ID', 'TM']):
    plt.figure(figsize=(12,6))
    plt.title("Boxplot of " + col + " by STN_ID")
    sns.boxplot(x="STN_ID", y=col, data=DATA, palette="Set1")
    plt.show()
```

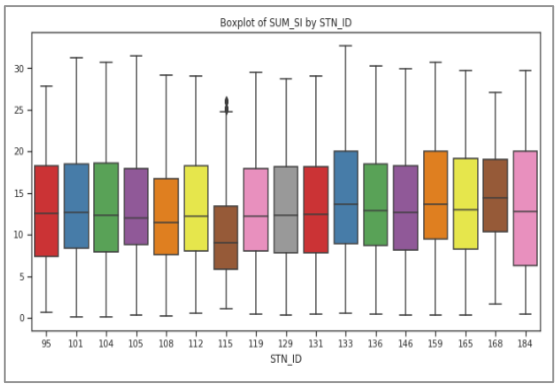
- **for문을 활용하여 데이터의 컬럼 수만큼 박스플롯을 그리는 작업을 반복**
- **박스플롯을 그릴 컬럼과 출력할 박스플롯 그래프의 제목을 지정**
- **sns.boxplot()으로 STN_ID별 각 컬럼의 박스플롯을 그리고, x축 라벨을 붙임**

> boxplot 결과









3. 히스토그램그리기

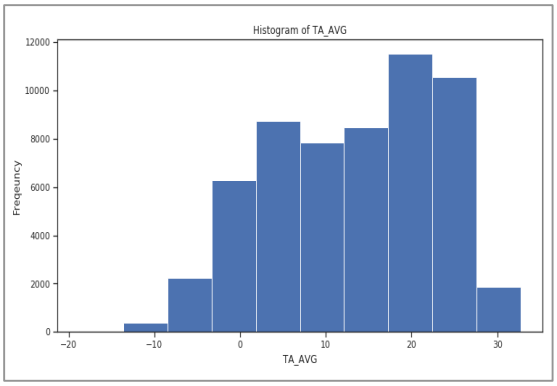
● 히스토그램 그리기

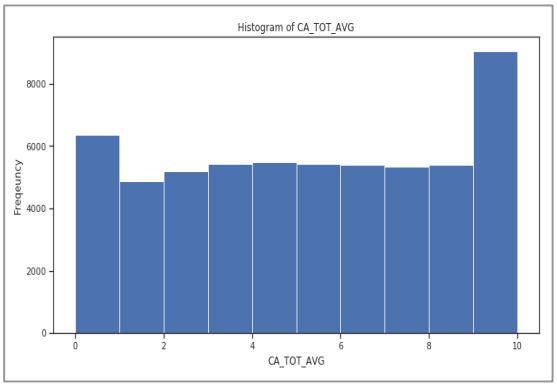
- 분석 데이터를 히스토그램을 그려 분포를 확인

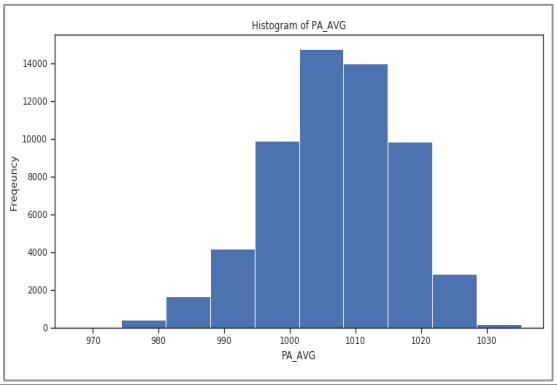
```
#============================================================
# Histogram 그리기
#============================================================
for col in DATA.columns.difference(['STN_ID', 'TM']):
    plt.figure(figsize=(12,6))
    plt.title("Histogram of " + col)
    plt.xlabel(col)
    plt.ylabel('Freqeuncy')
    plt.hist(DATA[col])
    plt.show()
```

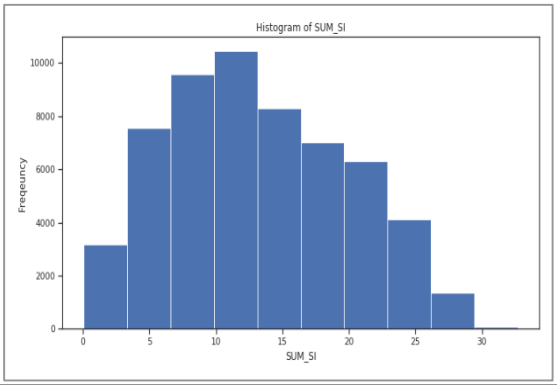
- for문을 활용하여 데이터의 컬럼 수만큼 히스토그램을 그리는 작업을 반복
- 히스토그램을 그릴 컬럼과 출력할 그림의 제목을 지정
- hist()로 각 컬럼의 히스토그램을 그리고, xlabel()로 x축 라벨을 붙임

> histogram 결과









기상청

Ⅲ. 데이터 처리

1. 이상치 처리
2. 결측치 처리
3. 피생변수 생성

1. 이상치처리

- **이상치 처리**
  - **기후자료 품질검사 알고리즘 기준값을 기준으로 이상치를 정의하고 NA로 처리**
  - **소형증발량 기후자료 품질검사 알고리즘 기준값 : 0~15**

```
#============================================================
# 이상치 처리
#============================================================
#소형증발량 기후자료 품질검사 알고리즘 기준값 : 0~15 -> NA로 치환
DATA.at[(DATA['SUM_SML_EV'] > 15) | (DATA['SUM_SML_EV'] < 0), 'SUM_SML_EV'
] = np.nan
DATA['SUM_SML_EV'].describe()
```

  - **소형증발량(SUM_SML_EV)이 15 초과이거나 0 미만인 값은 NA로 치환**
  - **describe()로 이상치 여부 치환되었는지 확인**

> 실행 결과

```
count    58000.000000
mean         3.127098
std          1.981033
min          0.000000
25%          1.500000
50%          2.800000
75%          4.500000
max         15.000000
Name: SUM_SML_EV, dtype: float64
```

2. 결측치처리(1/2)

- **결측치 0으로 대체**
  - 기후자료 품질검사 알고리즘 기준값을 기준으로 이상치를 정의하고 NA로 처리
  - 합계 강수량, 일최심신적설, 일최심적설 : 비 또는 눈이 안온 상태로 판단하여 결측치를 0으로 대체
  - 합계 일조시간 : 일조시간이 없다고 판단하여 결측치를 0으로 대체

```
#=============================================================
# 결측치 처리
#=============================================================
# SUM_RN, SUM_SS, SD_HR3_MAX, SD_TOT_MAX는 0으로 치환
DATA['SUM_RN'] = DATA['SUM_RN'].fillna(value = 0)
DATA['SUM_SS'] = DATA['SUM_SS'].fillna(value = 0)
DATA['SD_HR3_MAX'] = DATA['SD_HR3_MAX'].fillna(value = 0)
DATA['SD_TOT_MAX'] = DATA['SD_TOT_MAX'].fillna(value = 0)
DATA.isnull().sum()
```

- fillna()로 해당 기상변수가 NA인 값 선택하여 0으로 치환
- Isnull().sum() 으로 NA가 0으로 치환되었는지 확인

> 실행 결과

```
STN_ID          0
TM              0
TA_AVG          0
TA_MAX          0
TA_MIN          0
SUM_RN          0
WS_AVG          0
WS_MAX          0
HM_AVG          0
HM_MIN          0
SUM_SS          0
SUM_SI         43
TD_AVG          0
PV_AVG          0
PA_AVG          0
PS_AVG          0
PS_MAX          0
PS_MIN          0
SD_HR3_MAX      0
SD_TOT_MAX      0
CA_TOT_AVG      0
CA_MID_AVG      0
TS_AVG          0
TS_MAX          0
TS_MIN          0
SUM_SML_EV      5
HT              0
dtype: int64
```

2. 결측치처리(2/2)

- **KNN Imputation 을 활용하여대체**

- 합계 일사량, 소형 증발량 기상변수는 KNN Imputation 대체 기법을 활용하여 결측치를 대체

```
# SUM_RN, SUM_SS, SD_HR3_MAX, SD_TOT_MAX는 0으로 치환
DATA['SUM_RN'] = DATA['SUM_RN'].fillna(value = 0)
DATA['SUM_SS'] = DATA['SUM_SS'].fillna(value = 0)
DATA['SD_HR3_MAX'] = DATA['SD_HR3_MAX'].fillna(value = 0)
DATA['SD_TOT_MAX'] = DATA['SD_TOT_MAX'].fillna(value = 0)
DATA.isnull().sum

# SUM_SI, SUM_SML_EV는 KnnImputation을 활용하여 결측치 처리
Imputer = KNNImputer(n_neighbors = 10)
DATA = pd.DataFrame(imputer.fit_transform(DATA), columns=DATA.columns)
DATA.isnull().sum()
```

- **KNNImputer()로 결측치 대체수행**
- **isnull().sum() 으로 결측치가 대체됨을 확인**

> 실행 결과

```
STN_ID        0
TM            0
TA_AVG        0
TA_MAX        0
TA_MIN        0
SUM_RN        0
WS_AVG        0
WS_MAX        0
HM_AVG        0
HM_MIN        0
SUM_SS        0
SUM_SI        0
TD_AVG        0
PV_AVG        0
PA_AVG        0
PS_AVG        0
PS_MAX        0
PS_MIN        0
SD_HR3_MAX    0
SD_TOT_MAX    0
CA_TOT_AVG    0
CA_MID_AVG    0
TS_AVG        0
TS_MAX        0
TS_MIN        0
SUM_SML_EV    0
HT            0
dtype: int64
```

2. 파생변수생성(1/2)

- **강수유무 파생변수 생성**

- 합계 강수량에서 파생하여 강수유무(RAIN) 더미변수를 생성함

: 합계강수량 = 0 일 때 : 강수없음(0)

: 합계강수량 >0 일 때 : 강수있음(1)

```
#강수유무 생성
DATA['RAIN'] = np.nan
DATA.at[(DATA['SUM_RN'] == 0), 'RAIN'] = 0
DATA.at[(DATA['SUM_RN'] > 0), 'RAIN'] = 1
DATA.info()
```

- **데이터셋에 RAIN컬럼을 생성**
- **합계강수량이 0일 때는 0값을, 0보다 클 때는 1값을 입력**
- **info()로 데이터 구조를 확인하여 RAIN 변수가 잘 생성 되었는지 확인**

> 실행 결과

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58005 entries, 0 to 58004
Data columns (total 28 columns):
STN_ID        58005 non-null float64
TM            58005 non-null float64
TA_AVG        58005 non-null float64
TA_MAX        58005 non-null float64
TA_MIN        58005 non-null float64
SUM_RN        58005 non-null float64
WS_AVG        58005 non-null float64
WS_MAX        58005 non-null float64
HM_AVG        58005 non-null float64
HM_MIN        58005 non-null float64
SUM_SS        58005 non-null float64
SUM_SI        58005 non-null float64
TD_AVG        58005 non-null float64
PV_AVG        58005 non-null float64
PA_AVG        58005 non-null float64
PS_AVG        58005 non-null float64
PS_MAX        58005 non-null float64
PS_MIN        58005 non-null float64
SD_HR3_MAX    58005 non-null float64
SD_TOT_MAX    58005 non-null float64
CA_TOT_AVG    58005 non-null float64
CA_MID_AVG    58005 non-null float64
TS_AVG        58005 non-null float64
TS_MAX        58005 non-null float64
TS_MIN        58005 non-null float64
SUM_SML_EV    58005 non-null float64
HT            58005 non-null float64
RAIN          58005 non-null float64
dtypes: float64(28)
memory usage: 12.4 MB
```

2. 파생변수생성(2/2)

- **2m높이의 일평균풍속 파생변수생성**

- 일평균 풍속에서 파생하여 Penman-Monteith 증발산량 계산식의 2m 풍속을 파생변수로활용

: 2m풍속 계산식 $= u \times \frac{4.87}{\ln(67.8Ht - 5.42)}$

- u는 풍속계 높이에서의 일평균풍속(m/s)
- Ht는 풍속계의 지상높이(m)

```
#2m 풍속 생성
#수식 : 일평균풍속 * (4.87/ln(67.8*풍속계높이 - 5.42))
#데이터에 HT_WD(풍속계높이) 컬럼 결합하기
mergeAll = GEO_DATA.loc[GEO_DATA['STN_ID'] != 146, ['STN_ID','HT_WD']]

# STNID 컬럼으로 asos 데이터와 geo 데이터 병합
DATA = pd.merge(DATA, mergeAll, how='left', on='STN_ID')

DATA.at[(DATA['STN_ID'] == 146) & (DATA['TM'] < 20150701), 'HT_WD'] = 18.4
DATA.at[(DATA['STN_ID'] == 146) & (DATA['TM'] >= 20150701), 'HT_WD'] = 10.0

#2m일평균풍속 계산하기
DATA['WS_2m'] = DATA['WS_AVG'] * (4.87 / (np.log((67.8 * DATA['HT_WD']) - 5.42)))

#풍속계높이 변수 삭제
DATA.drop(['HT_WD'], axis=1, inplace=True)

DATA.describe()
```

- **merge()로 데이터셋과 GEO_DATA에 있는 HT_WD(풍속계 높이) 변수와 결합**
- **146(전주) 지점은 관측시작 및 종료일 기준에 맞게 따로 처리**(12p,데이터 결합하기 참고)
- **WS_2m(2m 풍속) 변수계산**
- **HT_WD(풍속계 높이) 컬럼 제거**
- **describe()로 WS_2m 변수가 잘 생성 되었는지확인**

> 실행 결과

	TM	TA_AVG	TA_MAX	TA_MIN	SUM_RN	WS_AVG	WS_MAX	HM_AVG	HM_MIN	SUM_SS	...
count	5.800500e+04	58005.000000	58005.000000	58005.000000	58005.000000	58005.000000	58005.000000	58005.000000	58005.000000	58005.000000	...
mean	2.010538e+07	13.195650	17.434673	9.656818	3.701779	2.463257	4.800527	67.734992	47.173784	5.866404	...
std	2.858221e+04	9.724377	9.764689	10.115668	13.307182	1.449396	2.236551	15.946926	18.935333	3.738440	...
min	2.006010e+07	-18.770000	-11.400000	-26.700000	0.000000	0.040000	0.300000	9.920000	4.000000	0.000000	...
25%	2.008063e+07	5.070000	9.200000	1.400000	0.000000	1.450000	3.300000	56.750000	32.000000	2.400000	...
50%	2.010122e+07	14.390000	19.000000	10.500000	0.000000	2.100000	4.400000	69.040000	46.000000	6.500000	...
75%	2.013061e+07	21.590000	25.700000	18.400000	0.500000	3.080000	5.800000	79.750000	61.000000	8.900000	...
max	2.015123e+07	32.720000	37.800000	30.300000	372.500000	16.650000	27.200000	100.000000	100.000000	14.200000	...

8 rows × 28 columns

기상청

Ⅳ. 모형 구축

1. 변수 선택
2. 모형 구축

1. 변수선택(1/4)

- **H2o 서버 셋팅**

-H2O는 빅데이터 처리를 위한 분산처리 프로세스를 통해 빠른 처리속도를 지원하고 다양한 머신러닝 분석 기술을 제공하는 패키지

- H2O 랜덤 포레스트(Random Forest) 함수를 활용하여 분석

```
#============================================================
# 로컬에 H2O 가상서버 설정하기
#============================================================
h2o.init(max_mem_size = "4G", nthreads = 4)
```

- **h2o.init()으로 H2O자바가상머신을 셋팅**

> 실행 결과

```
Checking whether there is an H2O instance running at http://localhost:54321 . connected.
```

H2O cluster uptime:	16 mins 41 secs
H2O cluster timezone:	Asia/Seoul
H2O data parsing timezone:	UTC
H2O cluster version:	3.28.0.1
H2O cluster version age:	2 days
H2O cluster name:	H2O_from_python_root_65lnbm
H2O cluster total nodes:	1
H2O cluster free memory:	3.023 Gb
H2O cluster total cores:	4
H2O cluster allowed cores:	4
H2O cluster status:	locked, healthy
H2O connection url:	http://localhost:54321
H2O connection proxy:	{'http': None, 'https': None}
H2O internal security:	False
H2O API Extensions:	Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
Python version:	3.6.9 final

1. 변수선택(2/4)

- **변수들간 다중공선성 제거(1/2)**

- 랜덤 포레스트 모형의 변수 중요도를 기준으로 상관계수가 높은 변수들을 제거해 나가는 방법을 활용하여 변수들간 다중공선성 문제를 해결

```
#============================================================
# 변수들간 다중공선성 제거
#============================================================
# 상관계수 구하기
corr_data = DATA.drop(['STN_ID', 'TM', 'SUM_SML_EV'], axis=1)

corr = corr_data.corr()
Abs_corr = abs(pd.DataFrame(corr))
Abs_corr['variable'] = Abs_corr.index
Abs_corr.reset_index(drop=True, inplace=True)
Abs_corr.head()

data_hex = h2o.H2OFrame(DATA)
new_data_hex = data_hex.drop(['STN_ID', 'TM', 'SUM_SML_EV'], axis=1)

xList = new_data_hex.columns
y = "SUM_SML_EV"

fit = H2ORandomForestEstimator(ntrees=50, max_depth=20, seed=1)
fit.train(x=xList, y=y, training_frame=data_hex)

varimp = fit.varimp(True)
varimp.head()
```

- **상관계수를 산출하기 전 STN_ID, TM, SUM_SML_EV(소형증발량)을 제외**
- **corr() 로 변수들의 상관계수를 산출**
- **abs()로 상관계수에 절대값을 적용**
- **Index를 사용하여 행 이름으로 설정되어있는 변수명을 variable이라는 이름의 컬럼으로 추출**
- **h2o.H2OFrame()으로 분석 데이터셋을 H2OFrame구조의 데이터로 변환**
- **X값 변수 목록 및 Y값 변수명을 셋팅(xList, y)**
- **H2ORandomForestEstimator() 및 train()으로 H2O 랜덤 포레스트 기법을 활용하여 모형을구축**
  - 함수의 파라미터는 H2ORandomForestEstimator 함수의 default 값으로 설정
- **varimp()로 구축한 모형에 활용된 변수들의 변수중요도를 산출**

1. 변수선택(3/4)

- **변수들간 다중공선성 제거(2/2)**

- 변수중요도가 높은 변수부터 순서대로 하나씩 선택하여, 해당 변수와 상관계수의 절대값이 0.45가 넘는 변수를 제거

```
# 다중공선성 제거
finalvar = varimp
j=0

while j < len(finalvar.variable):
    tmp = Abs_corr[(Abs_corr.variable != 'y') & (Abs_corr.variable != finalvar.variable[j])].loc[:,
['variable',finalvar.variable[j]]]
    tmp.columns = ['variable', 'Pearson']
    tmp.sort_values(['Pearson'], axis=0, ascending=False, inplace=True)
    tmp = tmp[tmp.Pearson > 0.45]

    finalvar = pd.merge(finalvar, tmp, how='left', on='variable')
    finalvar = finalvar.loc[finalvar.isnull()['Pearson'], :]
    finalvar = finalvar.reset_index(drop=True)
    finalvar = finalvar.drop('Pearson', 1)
    finalvar.sort_values(['scaled_importance'], axis=0, ascending=False, inplace=True)
    j = j + 1

#최종선택변수
finalvar
```

- **최종선택변수 데이터 프레임을 선언**
- **while()문으로 최종선택변수 데이터 프레임의 길이보다 작을 동안, 변수 중요도가 높은 변수를 하나씩 선택하여 작업을 수행**
- **선택한 변수와 다른 변수들간의 상관계수의 절대값을 추출**
- **columns로 컬럼명 지정**
- **sort_values()로 상관계수의 절대값이 높은 순서로 정렬**
- **상관계수의 절대값이 0.45가 넘는 변수들을 추출**
- **merge()로 변수 중요도 순서로 정렬되어 있는 데이터(finalvar)에 결합**
- **결합한 Pearson 컬럼이 NA인 것만 남기고 나머지 변수 삭제한 후 변수 중요도(scaled_importance)가 높은 순서 대로 정렬**

1. 변수선택(4/4)

> 실행 결과

	TA_AVG	TA_MAX	TA_MIN	SUM_RN	WS_AVG	WS_MAX	HM_AVG	HM_MIN	SUM_SS	SUM_SI	...	SD_TOT_MAX	CA_TOT_AVG	CA_MID_AVG
0	1.000000	0.981688	0.985247	0.159266	0.060841	0.090234	0.341901	0.344547	0.013691	0.397508	...	0.188669	0.248887	0.183079
1	0.981688	1.000000	0.940713	0.122972	0.119422	0.134929	0.279494	0.235833	0.105621	0.470715	...	0.205621	0.161436	0.094607
2	0.985247	0.940713	1.000000	0.186498	0.014390	0.055623	0.394527	0.427476	0.064787	0.320917	...	0.169941	0.317364	0.253219
3	0.159266	0.122972	0.186498	1.000000	0.087562	0.116370	0.313761	0.323218	0.325889	0.261261	...	0.018216	0.328643	0.317658
4	0.060841	0.119422	0.014390	0.087562	1.000000	0.899057	0.131347	0.043338	0.050909	0.065792	...	0.032647	0.056505	0.100148

5 rows × 27 columns

Parse progress: |██| 100%
drf Model Build progress: |██| 100%

	variable	relative_importance	scaled_importance	percentage
0	TS_AVG	1.021892e+06	1.000000	0.136659
1	TS_MAX	1.010656e+06	0.989005	0.135157
2	SUM_SI	5.702114e+05	0.557996	0.076255
3	PS_MAX	5.411224e+05	0.529530	0.072365
4	CA_TOT_AVG	5.357984e+05	0.524320	0.071653

	variable	relative_importance	scaled_importance	percentage
0	TS_AVG	1.021892e+06	1.000000	0.136659
1	CA_TOT_AVG	5.357984e+05	0.524320	0.071653
2	HT	2.496793e+05	0.244330	0.033390
3	WS_2m	1.774918e+05	0.173689	0.023736
4	SUM_RN	8.844138e+04	0.086547	0.011827
5	SD_TOT_MAX	1.472423e+04	0.014409	0.001969

2. 모형구축(1/2)

- **하이퍼 파라미터 최적화(Hyper-parameter Optimization)**
  - 랜덤 포레스트의 하이퍼 파라미터를 조정하여 가장 제곱평균오차(MSE)가 낮은 모형을 선택
  - sample_rate, max_depth, ntrees, mtries 파라미터를 조정

```
#============================================================#
#하이퍼파라미터 최적화
#============================================================#
#최종선택변수
varList = list(finalvar['variable'])
y = "SUM_SML_EV"

#하이퍼파라미터 조합만들기
hyper_params = {'sample_rate': [0.3, 0.4],
                'max_depth': [18, 20, 25],
                'ntrees': [25, 50],
                'mtries': [-1, 1]}

#조합 모형 돌리기
m = H2OGridSearch(H2ORandomForestEstimator, grid_id = 'rf_grid', hyper_params=hyper_para
ms)
m.train(x = varList, y = y, training_frame = data_hex)

#mse가 낮은 순으로 정렬하기
sorted_grid = m.get_grid(sort_by = 'mse')
print(sorted_grid)

#베스트 모형 선택
best_model = h2o.get_model(sorted_grid.models[0])
print(best_model)
```

  - 최종선택변수를 varList로, 소형 증발량을 y값으로 변수 지정
  - 하이퍼 파라미터 조합 리스트를 생성
  - H2OGridSearch()로 하이퍼 파라미터 조합들을 활용하여 모형을 구축한 "rf_grid"를 생성
  - get_grid()로 "m"의 모형 구축 결과를 제곱평균오차(MSE)가 낮은순으로 정렬
  - get_model()로 정렬된 모형 결과 중, 가장 상위에 있는 모형을 베스트 모형으로 선택 (Model Selection)

2. 모형구축(2/2)

> 실행 결과 (1/2)

drf Grid Build progress: |██| 100%

	max_depth	mtries	ntrees	sample_rate	model_ids	mse
0	18	-1	50	0.3	rf_grid_model_6	2.0904202175868254
1	18	-1	50	0.4	rf_grid_model_18	2.097117363819815
2	20	-1	50	0.3	rf_grid_model_7	2.1041456524706867
3	25	-1	50	0.3	rf_grid_model_8	2.115193392705584
4	20	-1	50	0.4	rf_grid_model_19	2.1152642640587187
5	25	-1	50	0.4	rf_grid_model_20	2.139236719159359
6	18	-1	25	0.3	rf_grid_model_0	2.1400823663981945
7	18	-1	25	0.4	rf_grid_model_12	2.154852499971012
8	20	-1	25	0.3	rf_grid_model_1	2.159343413594997
9	20	-1	25	0.4	rf_grid_model_13	2.1706181395655175
10	25	-1	25	0.3	rf_grid_model_2	2.187568630416047
11	25	-1	25	0.4	rf_grid_model_14	2.2036177928476017
12	20	1	50	0.4	rf_grid_model_22	2.339556498850772
13	18	1	25	0.3	rf_grid_model_3	2.353473249479278
14	18	1	50	0.4	rf_grid_model_21	2.3598458902893964
15	18	1	50	0.3	rf_grid_model_9	2.3612944580171966
16	20	1	25	0.3	rf_grid_model_4	2.3722045514930805
17	20	1	25	0.4	rf_grid_model_16	2.3836932309909034
18	20	1	50	0.3	rf_grid_model_10	2.391451886767261
19	25	1	50	0.3	rf_grid_model_11	2.3960213823053933
20	18	1	25	0.4	rf_grid_model_15	2.396566913968954
21	25	1	50	0.4	rf_grid_model_23	2.401025433511345
22	25	1	25	0.3	rf_grid_model_5	2.4393133227167563
23	25	1	25	0.4	rf_grid_model_17	2.543980971088313 2

2. 모형구축(2/2)

> 실행 결과 (2/2)

```
Model Details
=============
H2ORandomForestEstimator :  Distributed Random Forest
Model Key:  rf_grid_model_6
Model Summary:
```

number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	max_leaves	mean_leaves
50.0	50.0	5910670.0	18.0	18.0	18.0	7082.0	10828.0	9405.62

```
ModelMetricsRegression: drf
** Reported on train data. **

MSE: 2.0935364811985098
RMSE: 1.4469058301073052
MAE: 1.0883436031200457
RMSLE: 0.3789536078841909
Mean Residual Deviance: 2.0935364811985098
Scoring History:
```

timestamp	duration	number_of_trees	training_rmse	training_mae	training_deviance
2020-02-21 16:14:57	15.416 sec	0.0	nan	nan	nan
2020-02-21 16:14:57	15.548 sec	1.0	1.9443945	1.4331860	3.7806699
2020-02-21 16:14:57	15.667 sec	2.0	1.8092723	1.3410783	3.2734662
2020-02-21 16:14:57	15.785 sec	3.0	1.7232631	1.2791268	2.9696356
2020-02-21 16:14:57	15.907 sec	4.0	1.6589343	1.2377104	2.7520631
---	---	---	---	---	---
2020-02-21 16:15:00	18.971 sec	30.0	1.4574118	1.0959180	2.1240492
2020-02-21 16:15:01	19.096 sec	31.0	1.4568713	1.0955870	2.1224741
2020-02-21 16:15:01	19.233 sec	32.0	1.4565944	1.0954151	2.1216674
2020-02-21 16:15:01	19.337 sec	33.0	1.4557372	1.0947838	2.1191708
2020-02-21 16:15:03	21.294 sec	50.0	1.4469058	1.0883436	2.0935365

```
See the whole table with table.as_data_frame()
Variable Importances:
```

variable	relative_importance	scaled_importance	percentage
TS_AVG	1841832.6250000	1.0	0.4986703
CA_TOT_AVG	601508.1250000	0.3265813	0.1628564
WS_2m	478360.1250000	0.2597197	0.1295145
HT	349262.4375000	0.1896277	0.0945617
SUM_RN	272706.6562500	0.1480627	0.0738345
SD_TOT_MAX	149817.7187500	0.0813417	0.0405627

기상청

V. 모형 검증

1. 모형 성능 검증
2. 모형 예측력 검증

1. 모형 성능검증(1/2)

- **모형 성능 검증**
  - 최종 선택한 모형의 훈련데이터를 활용하여 제곱평균오차(mse), R^2, 평균절대백분율오차(sMAPE)를 확인하여 모형 적합도와 평균예측오차를 검증
  - 증발량은 0 또는 0에 가까운 값이기 때문에, 예측오차를 산출하기 위해 sMAPE 지표를 사용

$$sMAPE = \frac{100\%}{n}\sum_{t=1}^{n}\frac{|F_t - A_t|}{|A_t| + |F_t|}$$

```
#=========================================================
#모형 성능 검증
#=========================================================
#sMAPE 함수 정의
def cal_smape_100(y, yhat):
    return np.mean(abs((yhat - y))/(abs(yhat) + abs(y)))

#MODEL RUNNING
fit = H2ORandomForestEstimator(ntrees=50, max_depth=18, sample_rate=0.3, mtries=-1, seed
=1234)
fit.train(x = varList, y = y, training_frame = data_hex)

#result
perform = pd.DataFrame({'mse':[fit.mse()], 'r2':[fit.r2()]})
varimp = pd.DataFrame(fit.varimp())

#predict
pred = fit.predict(data_hex)
Yhat = pred['predict'].as_data_frame()
Y = data_hex['SUM_SML_EV'].as_data_frame()
tmp = pd.concat([Y, Yhat], axis=1)
mape = pd.DataFrame({'smape_100':[cal_smape_100(tmp['SUM_SML_EV'], tmp['predict'])]})

print(perform)
print(mape)
print(varimp2)
```

- def로 sMAPE 함수를정의
- H2ORandomForestEstimator()로 최종 선택 모형을 구축
- model에서 mse()로 제곱평균오차를, r2()로 R^2를 산출
- varimp()로 변수중요도를 산출
- predict()로 구축한 모형의 예측값을 산출
- 예측값(Yhat)과 실제값(Y)을 정의한 sMAPE()함수를 활용하여 평균예측오차 산출

1. 모형 성능검증(2/2)

> 실행 결과

```
drf Model Build progress: |████████████████████████████████████████████████████████████| 100%
drf prediction progress: |████████████████████████████████████████████████████████████| 100%
        mse        r2
0  2.109224  0.462581
   smape_100
0   0.168689
            0             1         2         3
0      TS_AVG  1.839463e+06  1.000000  0.497763
1  CA_TOT_AVG  5.833800e+05  0.317147  0.157864
2       WS_2m  4.791704e+05  0.260495  0.129665
3          HT  3.455678e+05  0.187863  0.093511
4      SUM_RN  2.332077e+05  0.126780  0.063107
5  SD_TOT_MAX  2.146718e+05  0.116704  0.058091
```

2. 모형 예측력검증(1/2)

- **지점별 교차 검증(K-fold cross-validation) 수행**
  - 교차 검증 : 훈련에 사용되지 않은 데이터에 대해 검증을 수행하는 검증 방법
  - 한 지점을 제외한 나머지 지점의 데이터를 모형의 훈련 데이터로 사용하고, 훈련하지 않은 한 지점을 테스트 데이터로 활용하여 모형을 평가

```
# STN별 리스트 생성
STNList = DATA['STN_ID'].unique()

# 결과리스트 초기화
perform = list()
varimp = list()
mape = list()

# MODEL RUNNING
for i in range(len(STNList)):
    stn = STNList[i]
    print("STN_ID : ", stn, "...computing")
    train = data_hex[data_hex['STN_ID'] != stn, :]
    valid = data_hex[data_hex['STN_ID'] == stn, :]

    m = H2ORandomForestEstimator(ntrees=50, max_depth=18, sample_rate=0.3, mtries=-1, seed=1234)
    m.train(x = varList, y = y, training_frame = train)

    perform_df = pd.DataFrame({'mse':[m.mse()], 'r2':[m.r2()]})
    perform.append(perform_df)

    varimp_df = m.varimp(True)
    varimp.append(varimp_df)

    print("STN_ID : ", stn, "...predicting")

    pred = m.predict(valid)
    Yhat = pred['predict'].as_data_frame()
    Y = valid['SUM_SML_EV'].as_data_frame()
    tmp = pd.concat([Y, Yhat], axis=1)

    mape_df = pd.DataFrame({'STN_ID':stn, 'smape_100':[cal_smape_100(tmp['SUM_SML_EV'], tmp['pred
ict'])]}, index=[i])
    mape.append(mape_df)

# STN별 교차검증 결과
mape_result = pd.concat(mape)
mape_result
```

2. 모형 예측력 검증(2/2)

- 지점 리스트 생성(STNList)
- 지점별 결과값을 저장할 리스트 변수 선언
- for()문으로 STN_ID별로 교차 검증 수행
- print()로 반복문 프로세스 진행상황 출력
- 해당 지점을 제외한 나머지 지점 데이터를 Train 데이터로 생성
- 해당 지점의 데이터를 valid 데이터 생성
- 모형 학습 수행
- mse(), r2(), varimp()로 모형의 성능과 변수 중요도를 산출하여 각각의 리스트에 저장
- predict()로 valid 데이터를 모형 검증수행
- 지점별 sMAPE를 산출하여 sMAPE 리스트에저장
- for문이 수행 완료된 후, concat()으로 리스트를 하나의 데이터 프레임으로 합쳐서 출력

> 실행 결과(1/2)

```
STN_ID :  95.0 ...computing
drf Model Build progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  95.0 ...predicting
drf prediction progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  101.0 ...computing
drf Model Build progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  101.0 ...predicting
drf prediction progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  104.0 ...computing
drf Model Build progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  104.0 ...predicting
drf prediction progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  105.0 ...computing
drf Model Build progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  105.0 ...predicting
drf prediction progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  108.0 ...computing
drf Model Build progress: |████████████████████████████████████████████████████████████| 100%
STN_ID :  108.0 ...predicting
drf prediction progress: |████████████████████████████████████████████████████████████| 100%
```

<실행 결과 일부 생략>

	STN_ID	smape_100
0	95.0	0.224574
1	101.0	0.328832
2	104.0	0.351992
3	105.0	0.319147
4	108.0	0.304644
5	112.0	0.219496
6	115.0	0.225369
7	119.0	0.207498
8	129.0	0.254076
9	131.0	0.197774
10	133.0	0.211680
11	136.0	0.205868
12	146.0	0.211706
13	159.0	0.166313
14	165.0	0.202397
15	168.0	0.193262
16	184.0	0.262038

본 문서의 내용은 기상청 날씨마루(https://bd.kma.go.kr)의
분석 플랫폼 활용을 위한 Python 프로그래밍 교육 자료 입니다.