기상청

[Python을 활용한 분석교육실습]

ASOS(종관기상관측) 자료를 활용한 적조 발생 모형 분석 사례

기상기후
빅데이터 분석 플랫폼

분석 개요

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

● 종관기상관측장비(ASOS)[1)]

- 기상청은 서울기상관측소를 비롯하여 전국 95개소의 종관기상관측장비(ASOS)와 무인으로 운영되는 493개소의 자동기상관측장비(AWS)를 이용하여 지상기상관측업무를 수행하고 있다.
  종관기상관측 장비(ASOS)는 지방청, 지청, 기상대, 관측소 등에 설치되어 기상 현상 관측 및 국제 전문을 통한 자료 공유 등의 관측 업무를 수행한다.
- 기압, 기온, 풍향, 풍속, 습도, 강수량, 강수 유무, 일사량, 일조시간, 지면 온도, 초상 온도, 지중 온도, 토양 수분, 지하수위 14개 요소에 대해서는 종관기상관측장비(ASOS)로 자동 관측하고, 시정, 구름, 증발량, 일기 현상 등은 일부 자동과 목측(目測)으로 관측한다.
- 현상은 일정 시간(오전, 오후) 동안 발생한 일기 현상을 관측한 것이다. 일반적으로 일기 현상을 관측하기 위해 목측 또는 CCTV를 통해 원격 관측한다. 특히 이슬과 서리의 경우 다양한 기상요소(기온, 습도, 풍향, 풍속 등)에 의해 영향을 받으며 시간에 따라 변화한다.

● 적조[2)]

- 플랑크톤의 대량 번식으로 바다나 강, 운하, 호수의 색이 적색, 황색, 적갈색등으로 변하는 자연현상이다.
- 대량의 담수 유입 등의 원인으로 적조 생물의 에너지 원인 영양염류가 풍부해지고, 수온, 염분, 일조량이 적절하여 적조 생물이 대량 번식 할 수 있는 환경일 때 적조가 발생하는 것으로 알려져 있다.

1) 출처:기상청 홈페이지

2) 출처:기상기후 빅데이터 융합서비스 기술노트 2017

분석 시나리오

실습할 예제는 ASOS(종관기상관측장비) 기상데이터와 비기상데이터를 활용하여 적조 관측을 산출하는 모형을 구축하는 것입니다. 실습 예제를 통해 현재 수집되고 있는 적조 발생을 자동으로 산출하는 모형을 구축해봅니다.

● 적조 관측 산출 모형 구축 시나리오

[분석 활용 데이터]

기상 데이터

- 기온, 일조, 강수량, 습도, 중발량, 해면기압, 풍향, 풍속, 해면기압, 유의파고, 최대파고
- 해수면 높이, 수온, 염도, 온도, 심층별 염도 · 수온 · 유속

비기상 데이터

- 적조 발생 이력 : 발생일자, 해역, 수온범위, 적조 원인 생물, 관측 지점의 위도와 경도

[Analytics]

적조 영향 변수 선택	
정의	남해안의 해구 중 발생 빈도와 지리적 환경을 고려하여 97, 98, 99, 213, 214번 해구를 대상으로 하나의 모형을 산출 및 7일간 평균값, 14일간 평균값 및 누적값을 기준으로 선정
분석방법	랜덤포레스트 (Random Forest)
분석결과	적조 이력 발생

적조 산출 모형 구축	
정의	머신러닝 기법(RF)을 활용하여 적조를 예측하는 모형 구축
분석방법	Random ForestModel
분석결과	적조 예측 모형

※본 실습 예제는 적조 발생 모형 분석을 중심으로 합니다.

분석 절차

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

● 적조 관측 산출 모형 구축절차

	[실습 설명]	[실습 단계]
1 데이터 로딩	분석환경을 설정하고 분석에 필요한 기상 데이터 및비기상데이터를 로딩하여 분석에 필요한 데이터를 준비하는단계	1. 분석 환경 설정 및 패키지로딩 2. 데이터불러오기 3. 데이터결합하기
2 데이터 탐색	분석 데이터의 요약 통계를 확인하는 단계	(1. 요약 통계 보기)
3 데이터 처리	이상치를 처리하고 결측치를 대체하며 파생 변수를 생성하여 최종 데이터셋을 구성하는 단계	(1. 이상치 처리) 2. 결측치 처리 3. 파생변수 생성
4 모형 구축	분석할 데이터를 선정하고, 적조 관측 산출 모형을 구축하는 단계	1. 분석 데이터셋 2. 모형구축
5 모형 검증	최종 구축한 산출 모형의성능을 검증하는 단계	1. 최종 모형 선택 2. 모형 성능 및 예측력 검증

※적조 발생 모형 분석에 사용되는 데이터들은 이상치가 기처리된 데이터이므로 본 실습 예제에서는 데이터 탐색 과정과 데이터 처리에서 이상치 처리 과정이 없습니다.

분석 데이터 (1/3)

적조 발생 관측산출 모형 구축에 사용된 파일 및 변수 정보를 확인합니다.

● 적조 발생 관측 산출 모형 구축에 사용된 파일 및 변수정보

파일명	파일설명	변수명	변수설명	형식	예제
ASOS	종관기상 관측장비 (ASOS) 관측 자료	HAEGU_NUM	해구번호	Int	97
		YYMMDD	년월일	DATE	2008-09-20
		YEAR	년도	Num	2008
		MONTH	월	Num	9
		WS_MAX_ASOS	최대 풍속	Num	4.665211304
		WS_INS	일 최대순간풍속	Num	6.248399844
		TA_MAX	최대 기온	Num	27.50820413
		HM_AVG	평균 상대습도	Num	81.39204617
		PA_AVG	평균 현지기압	Num	998.3676807
		SS_DAY	일조	Num	2.18794463
		EV_L	대형 증발량	Num	2.319477555
		SI_DAY	일사	Num	0.08144039
		RN_DAY	강수	Num	0.654936286
		RN_DUR	강수 지속	Num	1.103152996
BUOY_DP	부이_등표	HAEGU_NUM	해구번호	Int	97
		YYMMDD	년월일	DATE	2008-09-19
		YEAR	년도	Num	2008
		MONTH	월	Num	9
		WS_MAX_BD	최대 풍속	Num	9.904917206
		WS_MIN	최소 풍속	Num	1.452665542
		PS_MIN	최소 해면기압	Num	1008.818021
		WH_MAX	최대 파고	Num	2.46486516
HYCOM	HYCOM	YYMMDD	년월일	Date	2008-09-20
		HAEGU_NUM	해구번호	Int	97
		YEAR	년도	Num	2008
		MONTH	월	Num	9
		AVG_EMP	평균 표면 유력	Num	-0.0000679318
		MAX_SSH	최대 해수 고도	Num	0.259959
		AVG_SURFACE_SALINITY_TREND	평균 표면 염분 동향	Num	0.0193792
		STDEV_SURFACE_TEMPERATURE_TREND	표준 편차 평균 수온 동향	Num	0.131165
		AVG_SALINITY_01	평균 염분 10m	Num	31.6075
		AVG_SALINITY_02	평균 염분 20m	Num	31.6081
		AVG_SALINITY_03	평균 염분 30m	Num	31.7308
		AVG_SALINITY_04	평균 염분 40m	Num	-999
		MAX_SALINITY_01	최대 염분 10m	Num	32.3698
		MAX_SALINITY_02	최대 염분 20m	Num	32.3677
		MAX_SALINITY_03	최대 염분 30m	Num	31.7308

※본 실습 예제는 적조 발생 모형 분석을 중심으로 합니다.

분석 데이터 (2/3)

적조 발생 관측산출 모형 구축에 사용된 파일 및 변수 정보를 확인합니다.

● 적조 발생 관측 산출 모형 구축에 사용된 파일 및 변수정보

파일명	파일설명	변수명	변수설명	형식	예제
HYCOM	HYCOM	MAX_SALINITY_04	최대 염분 40m	Num	-999
		MIN_SALINITY_01	최소 염분 10m	Num	30.8946
		MIN_SALINITY_02	최소 염분 20m	Num	30.8968
		MIN_SALINITY_03	최소 염분 30m	Num	31.7308
		MIN_SALINITY_04	최소 염분 40m	Num	-999
		STDEV_SALINITY_01	표준편차 염분 10m	Num	0.496018
		STDEV_SALINITY_02	표준편차 염분 20m	Num	0.496018
		STDEV_SALINITY_03	표준편차 염분 30m	Num	0.496018
		STDEV_SALINITY_04	표준편차 염분 40m	Num	0.496018
		AVG_TEMP_01	평균 수온 10m	Num	26.3307
		AVG_TEMP_02	평균 수온 20m	Num	26.2999
		AVG_TEMP_03	평균 수온 30m	Num	26.1335
		AVG_TEMP_04	평균 수온 40m	Num	-999
		MAX_TEMP_01	최고 수온 10m	Num	26.5592
		MAX_TEMP_02	최고 수온 20m	Num	26.5603
		MAX_TEMP_03	최고 수온 30m	Num	26.1335
		MAX_TEMP_04	최고 수온 40m	Num	-999
		MIN_TEMP_01	최저 수온 10m	Num	26.1057
		MIN_TEMP_02	최저 수온 20m	Num	26.0367
		MIN_TEMP_03	최저 수온 30m	Num	26.1335
		MIN_TEMP_04	최저 수온 40m	Num	-999
		STDEV_TEMP_01	표준편차 수온 10m	Num	0.161473
		STDEV_TEMP_02	표준편차 수온 20m	Num	0.161473
		STDEV_TEMP_03	표준편차 수온 30m	Num	0.161473
		STDEV_TEMP_04	표준편차 수온 40m	Num	0.161473
		AVG_U_VELOCITY_01	평균 유속(동쪽) 10m	Num	-0.0215417
		AVG_U_VELOCITY_02	평균 유속(동쪽) 20m	Num	0.0072745
		AVG_U_VELOCITY_03	평균 유속(동쪽) 30m	Num	0.0462051
		AVG_U_VELOCITY_04	평균 유속(동쪽) 40m	Num	-999
		MAX_U_VELOCITY_01	최대 유속(동쪽) 10m	Num	-0.00495796
		MAX_U_VELOCITY_02	최대 유속(동쪽) 20m	Num	0.0288187
		MAX_U_VELOCITY_03	최대 유속(동쪽) 30m	Num	0.0462051
		MAX_U_VELOCITY_04	최대 유속(동쪽) 40m	Num	-999
		MIN_U_VELOCITY_01	최저 유속(동쪽) 10m	Num	-0.0441653
		MIN_U_VELOCITY_02	최저 유속(동쪽) 20m	Num	-0.00253834
		MIN_U_VELOCITY_03	최저 유속(동쪽) 30m	Num	0.0462051
		MIN_U_VELOCITY_04	최저 유속(동쪽) 40m	Num	-999

※본 실습 예제는 적조 발생 모형 분석을 중심으로 합니다.

분석 데이터 (3/3)

적조 발생 관측산출 모형 구축에 사용된 파일 및 변수 정보를 확인합니다.

● 적조 발생 관측 산출 모형 구축에 사용된 파일 및 변수정보

파일명	파일설명	변수명	변수설명	형식	예제
HYCOM	HYCOM	STDEV_U_VELOCITY_01	표준편차 유속(동쪽) 10m	Num	0.0131619
		STDEV_U_VELOCITY_02	표준편차 유속(동쪽) 20m	Num	0.0131619
		STDEV_U_VELOCITY_03	표준편차 유속(동쪽) 30m	Num	0.0131619
		STDEV_U_VELOCITY_04	표준편차 유속(동쪽) 40m	Num	0.0131619
		AVG_V_VELOCITY_01	평균 유속(북쪽) 10m	Num	-0.059696
		AVG_V_VELOCITY_02	평균 유속(북쪽) 20m	Num	-0.0138137
		AVG_V_VELOCITY_03	평균 유속(북쪽) 30m	Num	-0.017062
		AVG_V_VELOCITY_04	평균 유속(북쪽) 40m	Num	-999
		MAX_V_VELOCITY_01	최대 유속(북쪽) 10m	Num	-0.030235
		MAX_V_VELOCITY_02	최대 유속(북쪽) 20m	Num	0.00260627
		MAX_V_VELOCITY_03	최대 유속(북쪽) 30m	Num	-0.017062
		MAX_V_VELOCITY_04	최대 유속(북쪽) 40m	Num	-999
		MIN_V_VELOCITY_01	최저 유속(북쪽) 10m	Num	-0.121754
		MIN_V_VELOCITY_02	최저 유속(북쪽) 20m	Num	-0.0356542
		MIN_V_VELOCITY_03	최저 유속(북쪽) 30m	Num	-0.017062
		MIN_V_VELOCITY_04	최저 유속(북쪽) 40m	Num	-999
		STDEV_V_VELOCITY_01	표준편차 유속(북쪽) 10m	Num	0.0323702
		STDEV_V_VELOCITY_02	표준편차 유속(북쪽) 20m	Num	0.0149721
		STDEV_V_VELOCITY_03	표준편차 유속(북쪽) 30m	Num	-999
		STDEV_V_VELOCITY_04	표준편차 유속(북쪽) 40m	Num	-999
REDTIDE	적조이력 자료	YYYYMMDD	적조발생일자	Char	20090825
		COCHLO_YN	적조발생여부	Int	0
		TYPE	적조생물유형	Char	Gonyaulax sp.
		CELL_MIN	적조 최소농도	Num	500
		CELL_MAX	적조 최대농도	Num	15000
		LAT	적조 발생 위도	Num	35.9271433
		LON	적조 발생 경도	Num	129.6089087
HAEGU	해구 자료	HAEGU_NUM	해구번호	Int	1
		LAT	위도	Num	42.66666667
		LON	경도	Num	131.6666667

※본 실습 예제는 적조 발생 모형 분석을 중심으로 합니다.

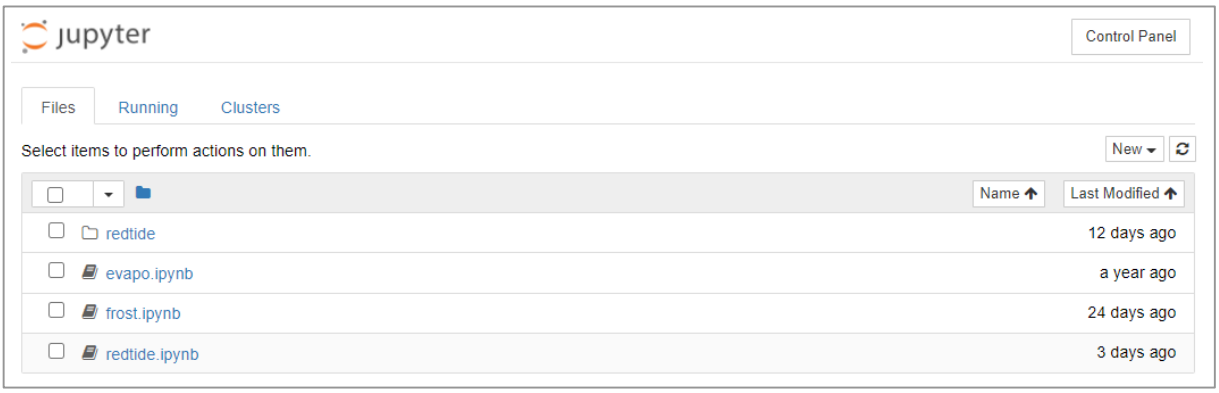
기상청

Ⅰ. 데이터 로딩

1. 분석 환경 설정 및 패키지로딩
2. 데이터 불러오기 : 기상데이터
3. 데이터 결합하기 : 기상데이터
4. 데이터 불러오기 : 적조 및 해구 데이터
5. 데이터 결합하기 : 적조 및 해구 데이터
6. 데이터 결합하기 : 기상 및 적조 데이터

1. 분석 환경 설정 및 패키지로딩 (1/2)

- **실습 실행 예제 파일 경로**
  - 분석을 실행하기 위한 예제 파일 경로



  - redtide.ipynb 파일을 클릭

- **패키지 로딩**
  - 분석을 실행하기 전 사용할 패키지를 로딩

```
#=========================================================
# 패키지 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 numpy as np
import itertools
import math
import datetime

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.grid.grid_search import H2OGridSearch
from pandasql import sqldf
```

1. 분석 환경 설정 및 패키지로딩 (2/2)

● 분석 환경 설정

- 분석을 실행하기 전 메모리를 초기화 하고 현재 설정된 디렉토리를 확인

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


#=============================================================
# 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 확인한다

2. 데이터불러오기 : 기상데이터 (1/2)

- **데이터를 불러온 뒤, 구조확인하기**
  - **분석에 활용 할 종관기상관측장비(ASOS) 관측 자료 데이터와 부이(BUOY), 등표(DP), 해양 데이터 (HYCOM) 를 data.table 형태로 불러옴**

```
#============================================================
# 기상 데이터 읽어오기
#============================================================
ASOS = pd.read_csv(currentPath + "/redtide/input/ASOS_imput.csv", encoding='UTF-8')
BUOY_DP = pd.read_csv(currentPath + "/redtide/input/BUOY_DP_imput.csv", encoding='UTF-8')
HYCOM = pd.read_csv(currentPath + "/redtide/input/SEA_IVs_hycom_all.csv", encoding='UTF-8')

ASOS = ASOS.rename(columns = {"WS_MAX":"WS_MAX_ASOS"})
BUOY_DP = BUOY_DP.rename(columns = {"WS_MAX":"WS_MAX_BD"})

#============================================================
# 불러온 데이터 구조 확인하기
#============================================================
ASOS
BUOY_DP
HYCOM
```

  - **pd.read_csv()로 기상 데이터 및 부이(BUOY), 등표(DP), 해양 데이터를 불러옴**
  - **rename() 으로 컬럼명을 변경**
  - **데이터 구조 확인**

2. 데이터불러오기 : 기상데이터 (2/2)

> 실행 결과

	HAEGU_NUM	YYMMDD	year	month	WS_MAX_ASOS	WS_INS	TA_MAX	HM_AVG	PA_AVG	SS_DAY
0	97	2008-09-19	2008	9	4.701703	6.997652	30.666605	67.651732	998.790286	10.581403
1	97	2008-09-20	2008	9	4.665211	6.248400	27.508204	81.392046	998.367681	2.187945
2	97	2008-09-21	2008	9	5.520223	8.754942	25.648252	82.103245	997.070368	2.028152
3	97	2008-09-22	2008	9	6.420983	8.780729	25.827896	75.537215	998.256090	6.096607
4	97	2008-09-23	2008	9	4.208098	6.173675	27.989062	81.541756	999.196706	2.193075
5	97	2008-09-24	2008	9	4.508694	6.629025	24.288109	81.355187	1002.273661	0.079262

	HAEGU_NUM	YYMMDD	year	month	WS_MAX_BD	WS_MIN	PS_MIN	WH_MAX
0	97	2008-09-19	2008	9	9.904917	1.452666	1008.818021	2.464865
1	97	2008-09-20	2008	9	9.288222	2.390402	1007.305026	1.235755
2	97	2008-09-21	2008	9	9.690123	1.218966	1007.076554	8.596796
3	97	2008-09-22	2008	9	8.917646	3.311054	1007.822278	1.394302
4	97	2008-09-23	2008	9	6.518775	0.932692	1008.615224	1.049736
5	97	2008-09-24	2008	9	10.736216	3.193103	1011.243100	1.772686

	YYMMDD	HAEGU_NUM	year	month	AVG_EMP	MAX_SSH	AVG_SURFACE_SALINITY_TREND
0	2008-09-20	97	2008	9	-0.000068	0.259959	0.019379
1	2008-09-20	98	2008	9	0.000088	0.240687	0.061153
2	2008-09-20	99	2008	9	0.000333	0.258393	-0.242847
3	2008-09-20	213	2008	9	0.000091	0.249247	-0.293269
4	2008-09-20	214	2008	9	0.000153	0.239578	-0.443171
5	2008-09-21	97	2008	9	-0.000011	0.266343	0.014799
6	2008-09-21	98	2008	9	0.000075	0.249976	-0.272004

3. 데이터결합하기 : 기상데이터

- **데이터 결합하기**

- 해구 번호(HAEGU_NUM) 및 날짜 정보를 중심으로 기상데이터(ASOS), 부이등표(BUOY_DP), 해양 데이터(HYCOM) 테이블을 결합

```
#============================================================
# 테이블 결합 및 확인
#============================================================
HYCOM['YYMMDD'] = pd.to_datetime(HYCOM['YYMMDD'], format='%Y-%m-%d')
HYCOM[['year', 'month']] = HYCOM[['year', 'month']].apply(pd.to_numeric)
HYCOM.dtypes

ASOS['YYMMDD'] = pd.to_datetime(ASOS['YYMMDD'], format='%Y-%m-%d')
ASOS[['year', 'month']] = ASOS[['year', 'month']].apply(pd.to_numeric)
ASOS.dtypes

BUOY_DP['YYMMDD'] = pd.to_datetime(BUOY_DP['YYMMDD'], format='%Y-%m-%d')
BUOY_DP[['year', 'month']] = BUOY_DP[['year', 'month']].apply(pd.to_numeric)
BUOY_DP.dtypes

DT = pd.merge(HYCOM, ASOS, how='left', on=['HAEGU_NUM', 'YYMMDD', 'year', 'month'])
DT = pd.merge(DT, BUOY_DP, how='left', on=['HAEGU_NUM', 'YYMMDD', 'year', 'month'])

print(DT)
```

- **merge() 로 기상 데이터(ASOS) 와 부이(등표), 해양(HYCOM) 데이터를 결합**
- **columns로 결합한 테이블의 Column 이름 확인**

> 실행 결과

```
          YYMMDD  HAEGU_NUM  year  month   AVG_EMP   MAX_SSH  #
0     2008-09-20         97  2008      9 -0.000068  0.259959
1     2008-09-20         98  2008      9  0.000088  0.240687
2     2008-09-20         99  2008      9  0.000333  0.258393
3     2008-09-20        213  2008      9  0.000091  0.249247
4     2008-09-20        214  2008      9  0.000153  0.239578
5     2008-09-21         97  2008      9 -0.000011  0.266343
                              ...
12743  1026.637640   1.753968
12744  1026.173269   2.472960
12745  1029.682347   1.915282
12746  1029.563176   1.787385
12747  1029.237699   1.775651
12748  1030.444385   1.422014
12749  1029.843490   2.373297
12750  1028.553362   1.214834
12751  1028.489670   1.298987
12752  1028.002832   1.344273
12753  1029.031511   0.868813
12754  1028.739520   1.271545

[12755 rows x 86 columns]
```

4. 데이터불러오기 : 적조 및 해구데이터 (1/2)

- **데이터를 불러온 뒤, 구조확인하기**
  - **분석에 활용할 적조 이력(redtide) 데이터와 해구(HAEGU) 데이터를 data.table 형태로 불러옴**

```
#============================================================
# 적조 데이터 읽어오기
#============================================================
redtide = pd.read_csv(currentPath + "/redtide/input/redtide.csv", sep='₩t', encoding='UTF-8')

# column 수정
redtide.rename(columns={redtide.columns[0]:"YYMMDD"}, inplace = True)
redtide.rename(columns={redtide.columns[1]:"Cochlo_YN"}, inplace = True)
redtide.rename(columns={redtide.columns[5]:"LAT_r"}, inplace = True)
redtide.rename(columns={redtide.columns[6]:"LON_r"}, inplace = True)

redtide['YYMMDD'] = pd.to_datetime(redtide['YYMMDD'], format='%Y%m%d')
redtide['month'] = pd.to_numeric(redtide['YYMMDD'].dt.month)
redtide['year'] = pd.to_numeric(redtide['YYMMDD'].dt.year)

#============================================================
# 불러온 데이터 구조 확인하기
#============================================================
redtide.info()
```

- **pd.read_csv()로 적조 데이터 불러옴**
- **rename()로 컬럼명을 변경**
- **to_datetime() 로 문자열인 데이터를 날짜 데이터로 변환**
- **to_numeric() 으로 문자열인 데이터를 숫자형 데이터로 변환**
- **Info() 로 데이터 구조 확인**

> 실행 결과

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2324 entries, 0 to 2323
Data columns (total 9 columns):
YYMMDD       2324 non-null datetime64[ns]
Cochlo_YN    2324 non-null int64
type         1697 non-null object
Cell_min     2303 non-null float64
Cell_max     2262 non-null float64
LAT_r        2324 non-null float64
LON_r        2324 non-null float64
month        2324 non-null int64
year         2324 non-null int64
dtypes: datetime64[ns](1), float64(4), int64(3), object(1)
memory usage: 163.5+ KB
```

4. 데이터불러오기 : 적조 및 해구데이터 (2/2)

```
#============================================================
# 해구 데이터 읽어오기
#============================================================
HAEGU = pd.read_csv(currentPath + "/redtide/input/SEA_latlon.csv", sep='₩t', encoding='UTF-8') #loadi
ng redtide data
HAEGU = HAEGU.sort_values(by=['HAEGU_NUM'])
HAEGU['row_h'] = HAEGU.index + 1

HAEGU.rename(columns={HAEGU.columns[1]:"LAT_h"}, inplace = True)
HAEGU.rename(columns={HAEGU.columns[2]:"LON_h"}, inplace = True)

#============================================================
# 불러온 데이터 구조 확인하기
#============================================================
HAEGU.info()
```

- **pd.read_csv()로 해구 데이터 불러옴**
- **rename()로 컬럼명을 변경**
- **Info() 로 데이터 구조 확인**

> 실행 결과

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1318 entries, 0 to 1317
Data columns (total 4 columns):
HAEGU_NUM    1318 non-null int64
LAT_h        1318 non-null float64
LON_h        1318 non-null float64
row_h        1318 non-null int64
dtypes: float64(2), int64(2)
memory usage: 51.5 KB
```

5. 데이터결합하기 : 적조 및 해구데이터 (1/2)

- **데이터 결합하기**

- 위도, 경도를 중심으로 적조 데이터(redtide) 테이블과 해구 정보 (HAEGU) 테이블을 결합

```
#============================================================
# 테이블 결합 및 확인
#============================================================
match=redtide[["LAT_r","LON_r"]].drop_duplicates()
match["row_r"]=match.index+1

def expand_grid(data_dict):
    rows = itertools.product(*data_dict.values())
    return pd.DataFrame.from_records(rows, columns=data_dict.keys())

tmp = expand_grid({'row_h' : HAEGU['row_h'], 'row_r' : match['row_r']})
np.shape(tmp)[0]

tmp = pd.merge(tmp, HAEGU, how='left', on=['row_h'])
tmp = pd.merge(tmp, match, how='left', on=['row_r'])

tmp['dist'] = np.hypot(tmp['LAT_h'].sub(tmp['LAT_r']), tmp['LON_h'].sub(tmp['LON_r']))

pysqldf = lambda q: sqldf(q, globals())
tmp = pysqldf("select HAEGU_NUM, LAT_h, LON_h, LAT_r, LON_r, min(dist) as dist from tmp group by ro
w_r;")
redtide = pd.merge(redtide, tmp, how='left', on=['LAT_r','LON_r'])

redtide.info()
redtide.head(5)

#============================================================
# HAEGU, period 선택
#============================================================
start = datetime.datetime.strptime("2008-09-19", "%Y-%m-%d").date()
end = datetime.datetime.strptime("2016-12-31", "%Y-%m-%d").date()

redtide = redtide.loc[pd.to_datetime(redtide['YYMMDD']) >= start]
redtide = redtide.loc[pd.to_datetime(redtide['YYMMDD']) <= end]

redtide = redtide[redtide['month'].isin([5, 6, 7, 8, 9, 10, 11, 12])]
redtide = redtide[redtide['HAEGU_NUM'].isin([97, 98, 99, 213, 214])]

# 해구Num, 같은 날짜 여러 상태의 경우 Cochlo_YN값 MAX로 표출
redtide = redtide.groupby(['HAEGU_NUM', 'YYMMDD']).max()['Cochlo_YN'].reset_index()
redtide.head(5)
```

5. 데이터결합하기 : 적조 및 해구데이터 (2/2)

- 적조 데이터의 위도, 경도로 가장 가까운 해구 번호를 찾아 그 값을 중심으로 적조 데이터 결합
- drop_duplicates() 로 unique한 1개의 행만 남기고 나머지 중복은 제거
- expand_grid 함수를 정의하여 제공된 벡터 또는 요인의 모든 조합에서 데이터 프레임 생성
- np.hypot() 로 제곱근을 구합
  : np.hypot() 함수는 유클리드 표준을 반환하는 Python의 내장 수학함수로서, sqrt(x*x + y*y) 를 갖는 float 값을 반환
- pysqldf() sql문을 사용하여 필요한 데이터 테이블 추출
  : 적조 관측 위치에서 가장 가까운 거리 (min dist)를 가진 해구정보(HAEGU_NUM, LAT_h, LON_h) 추출
- 데이터 날짜 및 해구 구간 설정
  - 기간 : 2008-09-19 ~ 2016-12-31
  - month : 5, 6, 7, 8, 9, 10, 11, 12
  - HAEGU : 97, 98, 99, 213, 214
- head() 로 결합한 데이터 확인

> 실행 결과

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2324 entries, 0 to 2323
Data columns (total 13 columns):
YYMMDD       2324 non-null datetime64[ns]
Cochlo_YN    2324 non-null int64
type         1697 non-null object
Cell_min     2303 non-null float64
Cell_max     2262 non-null float64
LAT_r        2324 non-null float64
LON_r        2324 non-null float64
month        2324 non-null int64
year         2324 non-null int64
HAEGU_NUM    2324 non-null int64
LAT_h        2324 non-null float64
LON_h        2324 non-null float64
dist         2324 non-null float64
dtypes: datetime64[ns](1), float64(7), int64(4), object(1)
memory usage: 254.2+ KB
```

	YYMMDD	Cochlo_YN	type	Cell_min	Cell_max	LAT_r	LON_r	month	year	HAEGU_NUM	LAT_h	LON_h	dist
0	2009-08-25	0	Gonyaulax sp.	500.0	15000.0	35.927143	129.608909	8	2009	82	36.166667	129.666667	0.246389
1	2009-08-25	0	Scrippsiella trochoidea	500.0	7000.0	35.927143	129.608909	8	2009	82	36.166667	129.666667	0.246389
2	2009-09-07	0	Noctiluca scintillans	4500.0	5000.0	36.067519	129.511865	9	2009	82	36.166667	129.666667	0.183831
3	2009-09-07	0	Noctiluca scintillans	170.0	NaN	36.067519	129.511865	9	2009	82	36.166667	129.666667	0.183831
4	2009-08-25	0	Peudo-nitzschia spp.	20.0	40.0	35.927143	129.608909	8	2009	82	36.166667	129.666667	0.246389

	HAEGU_NUM	YYMMDD	Cochlo_YN
0	97	2009-08-17	0
1	97	2009-10-28	1
2	97	2009-10-30	1
3	97	2010-07-19	0
4	97	2010-07-21	0

6. 데이터결합하기 : 기상 및 적조데이터

- **데이터 결합하기**

- 년월일(YYMMDD), 해구 번호(HAEGU_NUM)를 중심으로 기상 데이터(DT) 테이블과 적조 (redtide) 테이블을 결합

```
#=============================================================
# 기상 데이터, 적조 데이터 결합
#=============================================================
AB = pd.merge(DT, redtide, how='left', on=['YYMMDD', 'HAEGU_NUM'])
AB['Cochlo_YN'] = AB['Cochlo_YN'].fillna(0)

# 데이터 결측치 확인
AB.isnull().sum()
```

- **merge() 로 기상 데이터(DT) 테이블을 중심으로 적조 데이터(redtide) 테이블 결합**
- **isnull().sum() 으로 데이터 결측치 확인**

> 실행 결과

```
YYMMDD                          0
HAEGU_NUM                       0
year                            0
month                           0
AVG_EMP                         0
MAX_SSH                         0
AVG_SURFACE_SALINITY_TREND      0
STDEV_SURFACE_TEMPERATURE_TREND 0
AVG_SALINITY_01                 0
AVG_SALINITY_02                 0
AVG_SALINITY_03                 0
AVG_SALINITY_04                 0
MAX_SALINITY_01                 0
MAX_SALINITY_02                 0
MAX_SALINITY_03                 0
MAX_SALINITY_04                 0
MIN_SALINITY_01                 0
MIN_SALINITY_02                 0
                ...
STDEV_V_VELOCITY_01             0
STDEV_V_VELOCITY_02             0
STDEV_V_VELOCITY_03             0
STDEV_V_VELOCITY_04             0
WS_MAX_ASOS                     0
WS_INS                          0
TA_MAX                          0
HM_AVG                          0
PA_AVG                          0
SS_DAY                          0
EV_L                            0
SI_DAY                          0
RN_DAY                          0
RN_DUR                          0
WS_MAX_BD                       0
WS_MIN                          0
PS_MIN                          0
WH_MAX                          0
Cochlo_YN                       0
Length: 87, dtype: int64
```

기상청

Ⅱ. 데이터 처리

1. 결측처리

2. 파생변수 생성

1. 결측 처리

- **결측 처리**

- 해당 데이터의 -999 값을 NA로 판단하여 결측 처리 (결측은 변수별 mean 값으로 대체)

```
#============================================================
# 데이터 전처리
#============================================================
AB.describe()

# 기상변수에서 NA값이 -999로 처리된 경우 확인
AB[AB==-999]=np.nan
AB = AB.fillna(AB.mean())
AB.isnull().sum()

AB = AB.sort_values(by=['HAEGU_NUM', 'year', 'YYMMDD'], axis=0)
```

- **fillna() 로 해당 데이터가 NA인 값을 선택하여 mean값으로 치환**
- **isnull().sum()으로 최종적으로 모든 NA가 mean으로 치환되었는지 확인**

2. 파생변수 생성 (1/4)

- **파생변수 생성**

- 7일 평균값, 14일 평균값, 14일 누적치의 파생변수 생성

```
#============================================================
# 파생변수 생성
#============================================================
# 7일 평균값을 구함
AB['mean_AVG_EMP'] = AB.groupby(['HAEGU_NUM', 'year'])['AVG_EMP'].apply(lambda x : x.rolling(7).sum().shift(1)) / 7
AB['mean_MAX_SSH'] = AB.groupby(['HAEGU_NUM', 'year'])['MAX_SSH'].apply(lambda x : x.rolling(7).sum().shift(1)) / 7
...
AB['mean_STDEV_V_VELOCITY_03'] = AB.groupby(['HAEGU_NUM', 'year'])['STDEV_V_VELOCITY_03'].apply(lambda x : x.rolling(7).sum().shift(1)) / 7
AB['mean_STDEV_V_VELOCITY_04'] = AB.groupby(['HAEGU_NUM', 'year'])['STDEV_V_VELOCITY_04'].apply(lambda x : x.rolling(7).sum().shift(1)) / 7

# 전처리 이전 변수 삭제
AB.drop(['AVG_EMP', 'MAX_SSH', 'AVG_SURFACE_SALINITY_TREND',
   'STDEV_SURFACE_TEMPERATURE_TREND', 'AVG_SALINITY_01', 'AVG_SALINITY_02',
   'AVG_SALINITY_03', 'AVG_SALINITY_04', 'MAX_SALINITY_01', 'MAX_SALINITY_02',  ···
   'MAX_V_VELOCITY_04', 'MIN_V_VELOCITY_01', 'MIN_V_VELOCITY_02', 'MIN_V_VELOCITY_03',
   'MIN_V_VELOCITY_04', 'STDEV_V_VELOCITY_01', 'STDEV_V_VELOCITY_02', 'STDEV_V_VELOCITY_03',
   'STDEV_V_VELOCITY_04'], axis='columns', inplace=True)
```

- **groupby() 와 shift()를 이용하여 7일 평균값을 구함**

2. 파생변수 생성 (2/4)

```
#14days mean
AB['mean_TA_MAX'] = AB.groupby(['HAEGU_NUM', 'year'])['TA_MAX'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14
AB['mean_WS_MAX_ASOS'] = AB.groupby(['HAEGU_NUM', 'year'])['WS_MAX_ASOS'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14
AB['mean_WS_MAX_BD'] = AB.groupby(['HAEGU_NUM', 'year'])['WS_MAX_BD'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14
AB['mean_WS_INS'] = AB.groupby(['HAEGU_NUM', 'year'])['WS_INS'].apply(lambda x : x.rolling(14).sum().shift(1))
/ 14
AB['mean_WS_MIN'] = AB.groupby(['HAEGU_NUM', 'year'])['WS_MIN'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14
AB['mean_HM_AVG'] = AB.groupby(['HAEGU_NUM', 'year'])['HM_AVG'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14
AB['mean_EV_L'] = AB.groupby(['HAEGU_NUM', 'year'])['EV_L'].apply(lambda x : x.rolling(14).sum().shift(1)) / 14
AB['mean_PA_AVG'] = AB.groupby(['HAEGU_NUM', 'year'])['PA_AVG'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14
AB['mean_PS_MIN'] = AB.groupby(['HAEGU_NUM', 'year'])['PS_MIN'].apply(lambda x : x.rolling(14).sum().shift(1))
/ 14
AB['mean_WH_MAX'] = AB.groupby(['HAEGU_NUM', 'year'])['WH_MAX'].apply(lambda x :
x.rolling(14).sum().shift(1)) / 14

# 전처리 이전 변수 삭제
AB.drop(['TA_MAX', 'WS_MAX_ASOS', 'WS_MAX_BD', 'WS_INS', 'WS_MIN', 'HM_AVG', 'EV_L', 'PA_AVG',
'PS_MIN', 'WH_MAX'], axis='columns', inplace=True)

#14days cumulative
AB['mean_SS_DAY'] = AB.groupby(['HAEGU_NUM', 'year'])['SS_DAY'].apply(lambda x : x.rolling(14).sum().shift(1))
AB['mean_RN_DAY'] = AB.groupby(['HAEGU_NUM', 'year'])['RN_DAY'].apply(lambda x :
x.rolling(14).sum().shift(1))
AB['mean_SI_DAY'] = AB.groupby(['HAEGU_NUM', 'year'])['SI_DAY'].apply(lambda x : x.rolling(14).sum().shift(1))
AB['mean_RN_DUR'] = AB.groupby(['HAEGU_NUM', 'year'])['RN_DUR'].apply(lambda x :
x.rolling(14).sum().shift(1))
AB.drop(['SS_DAY', 'RN_DAY', 'SI_DAY', 'RN_DUR'], axis='columns', inplace=True)
```

- **groupby() 와 shift()를 이용하여 14일 평균값, 14일 누적치를 구함**

2. 파생변수 생성 (3/4)

```
# time interval create
AB['Cochlo_YN'] = AB.groupby(['HAEGU_NUM', 'year'])['Cochlo_YN'].shift(-7)

start = datetime.datetime.strptime("2008-10-04", "%Y-%m-%d").date()
end = datetime.datetime.strptime("2016-12-31", "%Y-%m-%d").date()

AB = AB.loc[pd.to_datetime(AB['YYMMDD']) >= start]
AB = AB.loc[pd.to_datetime(AB['YYMMDD']) <= end]

AB = AB[AB['month'].isin([5, 6, 7, 8, 9, 10, 11])]

#=============================================================
# 메모리 용량 줄이기
#=============================================================
del(start, end)
del(ASOS,BUOY_DP,DT,HAEGU,HYCOM,redtide,tmp)

# 데이터 확인
AB.info()
AB.head(5)
```

- **del() 를 이용하여 불필요한 데이터를 삭제 함으로써 메모리 용량을 줄임**

2. 파생변수 생성 (4/4)

> 실행 결과

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8850 entries, 70 to 12599
Data columns (total 87 columns):
YYMMDD                                  8850 non-null datetime64[ns]
HAEGU_NUM                               8850 non-null int64
year                                    8850 non-null int64
month                                   8850 non-null int64
Cochlo_YN                               8850 non-null float64
mean_AVG_EMP                            8850 non-null float64
mean_MAX_SSH                            8850 non-null float64
mean_AVG_SURFACE_SALINITY_TREND         8850 non-null float64
mean_STDEV_SURFACE_TEMPERATURE_TREND    8850 non-null float64
mean_AVG_SALINITY_01                    8850 non-null float64
mean_AVG_SALINITY_02                    8850 non-null float64
mean_AVG_SALINITY_03                    8850 non-null float64
mean_AVG_SALINITY_04                    8850 non-null float64
mean_MAX_SALINITY_01                    8850 non-null float64
mean_MAX_SALINITY_02                    8850 non-null float64
mean_MAX_SALINITY_03                    8850 non-null float64
mean_MAX_SALINITY_04                    8850 non-null float64
mean_MIN_SALINITY_01                    8850 non-null float64
mean_MIN_SALINITY_02                    8850 non-null float64
mean_MIN_SALINITY_03                    8850 non-null float64
mean_MIN_SALINITY_04                    8850 non-null float64
mean_STDEV_SALINITY_01                  8850 non-null float64
mean_STDEV_SALINITY_02                  8850 non-null float64
mean_STDEV_SALINITY_03                  8850 non-null float64
mean_STDEV_SALINITY_04                  8850 non-null float64
```

...

	YYMMDD	HAEGU_NUM	year	month	Cochlo_YN	mean_AVG_EMP	mean_MAX_SSH	mean_AVG_SURFACE_SALINITY_TREND
70	2008-10-04	97	2008	10	0.0	-0.000148	0.208562	0.079170
75	2008-10-05	97	2008	10	0.0	-0.000140	0.201662	0.107633
80	2008-10-06	97	2008	10	0.0	-0.000136	0.210660	0.106639
85	2008-10-07	97	2008	10	0.0	-0.000132	0.200848	0.130929
90	2008-10-08	97	2008	10	0.0	-0.000137	0.208403	0.159245

5 rows × 87 columns

기상청

Ⅲ. 모형 구축

1. 분석 데이터 셋
2. 모형 구축

1. 분석 데이터셋 (1/2)

- **H2O 서버 세팅**

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

- H2O 랜덤 포레스트 모델(Random Forest)을 활용하여분석

```
#=================================================================
# 로컬에 H2O 가상서버 설정하기
#=================================================================
h2o.init(max_mem_size = "4G", nthreads = 1)
h2o.remove_all()
```

- **h2o.init()으로 H2O 자바가상머신을 세팅**

> 실행 결과

```
Checking whether there is an H2O instance running at http://localhost:54321. connected.
Warning: Your H2O cluster version is too old (4 years, 2 months and 13 days)! Please download and install the latest version from http://h2o.ai/download/
```

H2O cluster uptime:	21 days 7 hours 11 mins
H2O cluster version:	3.10.0.8
H2O cluster version age:	4 years, 2 months and 13 days !!!
H2O cluster name:	H2O_from_python_p00000678_wkpuxj
H2O cluster total nodes:	1
H2O cluster free memory:	3.488 Gb
H2O cluster total cores:	10
H2O cluster allowed cores:	1
H2O cluster status:	locked, healthy
H2O connection url:	http://localhost:54321
H2O connection proxy:	None
Python version:	3.6.1 final

1. 분석 데이터셋 (2/2)

- **데이터 셋 분할**

- 모델 학습 및 최적화, 테스트를 위해 데이터 셋을 train_data, valid_data, test_data로 나눔

```
# setting AB data
dataXY = AB
dataXY = dataXY.drop(['HAEGU_NUM', 'YYMMDD', 'month', 'year'], axis=1)
dataXY = dataXY.rename(columns = {"Cochlo_YN":"Y"})
dataXY.reset_index(drop=True, inplace=True)
dataXY.info()

# train, valid, test 데이터 분리 (split to train, valid, test)
dataXY = h2o.H2OFrame(dataXY)
train_data, valid_data, test_data = dataXY.split_frame(ratios=[0.7,0.15], seed=1111)

# 독립변수, 종속변수 설정(x: 독립변수, y: 종속변수)
x = dataXY.columns
x.remove('Y')
y='Y'

train_data['Y'] = train_data['Y'].asfactor()
valid_data['Y'] = valid_data['Y'].asfactor()
test_data['Y'] = test_data['Y'].asfactor()
```

- **h2o.H2OFrame() 으로 분석 데이터셋을 H2OFrame구조의 데이터로 변환**
- **split_frame()으로 데이터 셋을 분할**
  1) train → 0.7, valid → 0.15, test → 0.15
- **asfactor()를 이용하여 문자형을 요인형(factor) 로 변환**

2. 모형구축

● 하이퍼 파라미터 최적화(Hyper-parameter Optimization)

- 랜덤 포레스트 모델의 하이퍼 파라미터를 조정하여 AUC(Area Under the Curve)가 가장 높은 모형을 선택

```
#============================================================
# 모형 튜닝 자동화
#============================================================
hyper_parameters = {'max_depth': [4, 6, 8, 12, 16, 20]}

# 조합 모형 돌리기
m = H2OGridSearch(H2ORandomForestEstimator,
          hyper_params=hyper_parameters,
          search_criteria={'strategy': "Cartesian"},
          grid_id='RF_depth_grid')

m.train(x = x,
     y = y,
     training_frame = train_data,
     validation_frame = valid_data,
     ntrees = 10000,
     stopping_rounds = 5,
     stopping_tolerance = 1e-4,
     stopping_metric = 'AUC',
     score_tree_interval = 5,
     seed=1111)

# AUC가 높은 순으로 정렬하기
sortedGrid = m.get_grid(sort_by='auc', decreasing=True)

print('===== sortedGrid =====')
print(sortedGrid)
```

- {} 로 하이퍼 파라미터 조합 리스트를 생성
- H2OGridSearch()로 하이퍼 파라미터 조합들을 활용하여 모형을 구축한 "RF_depth_grid" 를 생성
- get_grid()로 "RF_depth_grid"의 모형 구축 결과를 AUC(Area Under theCurve)가 높은 순으로 정렬

> 실행 결과

```
drf Grid Build progress: |████████████████████████████████████████████████████████████| 100%
===== sortedGrid =====
    max_depth              model_ids                 auc
0          12  RF_depth_grid_model_3  0.9833234610652937
1           8  RF_depth_grid_model_2  0.9763293388618035
2          16  RF_depth_grid_model_4  0.9762837745477743
3          20  RF_depth_grid_model_5   0.973914430218253
4           6  RF_depth_grid_model_1  0.9708388390212784
5           4  RF_depth_grid_model_0  0.9608374720918577
```

Ⅳ. 모형 검증

1. 최종 모형 선택
2. 모형 성능 및 예측력 검증

1. 최종 모형 선택 (1/3)

● **최종 모형 선택**

- **모형 성능을 AUC(Area Under the Curve) 기준으로 예측력 검증하여 최종 모형 선택**

```
# 모형 튜닝 자동화
minDepth = 12
maxDepth = 20

# options for grid search
max_runtime_secs = 60*10
max_models = 100

# random grid search
hyper_params = {
   'max_depth': list(range(minDepth, maxDepth + 1)),
   'sample_rate': [i * 0.01 for i in range(20, 100 + 1)],
   'col_sample_rate_per_tree': [i * 0.01 for i in range(20, 100 + 1)],
   'col_sample_rate_change_per_level': [i * 0.01 for i in range(90, 110 + 1)],
   'min_rows': [1,5,10,20,50,100],
   'min_split_improvement': [0,1e-8,1e-6,1e-4],
   'histogram_type': ['UniformAdaptive', 'QuantilesGlobal', 'RoundRobin']
}

search_criteria = {
   'strategy': "RandomDiscrete",
   'max_runtime_secs': max_runtime_secs,
   'max_models': max_models
}

grid = H2OGridSearch(H2ORandomForestEstimator
                , hyper_params=hyper_params
                , search_criteria=search_criteria
                , grid_id='RF_grid')

grid.train(
    x = x
   , y = y
   , training_frame = train_data
   , validation_frame = valid_data
   , ntrees = 10000
   , stopping_rounds = 5
   , stopping_tolerance = 1e-4
   , stopping_metric = 'AUC'
   , score_tree_interval = 5
   , seed = 1111
)
```

1. 최종 모형 선택 (2/3)

```
# Sort the grid models by AUC
sortedGrid = grid.get_grid(sort_by='auc', decreasing=True)
RF_AB_Tune = h2o.get_model(sortedGrid.model_ids[1])
print(RF_AB_Tune)
```

- **getGrid 의 summary table 에서 상위 3개의 값을 확인 후, max_depth의 min, max 값을 확인하여 minDepth, maxDepth에 입력**
  : 패키지에서 R과 Python 제공 함수가 다르므로, 해당 부분 python에서는 값을 확인하여 직접 지정
- **{} 로 하이퍼 파라미터와 search_criteria 조합 리스트 생성**
- **H2OGridSearch()로 하이퍼 파라미터 및 search_criteria 조합 들을 활용하여 모형을 구축한 "RF_grid " 를 생성**
- **get_grid()로 "RF_grid " 의 모형 구축 결과를 AUC(Area Under theCurve)가 높은 순으로 정렬**
- **get_model()로 최종 모형 선택**

*** Python 실행결과는 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

1. 최종 모형 선택 (3/3)

> 실행 결과

drf Grid Build progress: | 100%

Model Details
==========
H2ORandomForestEstimator : Distributed Random Forest
Model Key: RF_grid_model_46
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
115.0	115.0	61218.0	8.0	13.0	10.313044	32.0	46.0	37.46087

ModelMetricsBinomial: drf
** Reported on train data. **

MSE: 0.010901588650698062
RMSE: 0.10441067306888727
LogLoss: 0.07831897059746006
Mean Per-Class Error: 0.15750276348541115

...

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

variable	relative_importance	scaled_importance	percentage
mean_PS_MIN	164.9516754	1.0	0.0816302
mean_AVG_SURFACE_SALINITY_TREND	71.7687759	0.4350897	0.0355165
mean_TA_MAX	63.9231339	0.3875264	0.0316339
mean_MAX_TEMP_02	59.8882446	0.3630654	0.0296371
mean_MAX_TEMP_01	57.0225105	0.3456922	0.0282189
---	---	---	---
mean_STDEV_U_VELOCITY_03	8.8039131	0.0533727	0.0043568
mean_AVG_V_VELOCITY_04	7.5203571	0.0455913	0.0037216
mean_STDEV_SALINITY_04	7.1529312	0.0433638	0.0035398

2. 모형 성능 및 예측력검증

- **모형 성능 및 예측력 검증**

- 최종 선택한 모형으로 테스트 데이터를 예측하고, AUC(Area Under the Curve)와 혼동행렬(Confusion Matrix) 로 결과 확인

```
#============================================================
# forecast
#============================================================
pred= RF_AB_Tune.predict(test_data)
pred=pred.as_data_frame()
test_data=test_data.as_data_frame()

pred = pd.concat([pred['predict'], pred['p1'], test_data['Y']], axis=1)
pred.columns = ['Yhat','p1','Y']

# confusion matrix
print(confusion_matrix(pred['Yhat'],pred['Y']),
      classification_report(pred['Yhat'],pred['Y']))
```

- **predict()로 최종 선택 모형을 예측**
- **confusion_matrix()로 예측값과 실제값으로 혼동행렬을 출력**
- **classification_report()로 주요 분류 지표를 보여주는 텍스트 보고서를 작성**

> 실행 결과

```
drf prediction progress: |████████████████████████████████████████████████████████████████| 100%
[[1294    6]
 [  15   10]]              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1300
           1       0.62      0.40      0.49        25

    accuracy                           0.98      1325
   macro avg       0.81      0.70      0.74      1325
weighted avg       0.98      0.98      0.98      1325
```

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