

PROJECT

수도권 결빙 예측 웹 서비스 개발 프로젝트

5조 박거량 박소진 손우영 최태호

CONTENT

01 | 팀원 주요 역할 및 소개

02 | 프로젝트 개요

03 | 아키텍처 소개

04 | 데이터 수집 및 관리

05 | 머신러닝 모델 선정 배경

06 | 머신러닝 기반 결빙 예측 및 결과

07 | 트러블 슈팅 요약

08 | 배포 웹사이트 시현 및 백엔드 설계

09 | Q&A

01. 팀원 주요 역할 소개

박거량

데이터 전처리, 모델링

박소진

시스템 아키텍처 설계,
백엔드 개발,
프론트엔드 개발

손우영

Azure 클라우드 인프라 구축,
백엔드 개발,
프론트엔드 개발

최태호

Azure 클라우드 인프라 구축,
백엔드 개발

02. 프로젝트 개요

배 경

겨울철 결빙으로 인한 교통사고와 안전 문제는 심각한 사회적 비용을 유발합니다.
이러한 위험을 사전에 예측하여 예방할 수 있는 시스템은 교통, 물류, 공공안전 분야에서 매우 중요한 역할을 할 것이라고 생각하였습니다.

목 적

Azure Cloud와 머신 러닝 기술을 기반으로 기상 데이터를 분석하여 결빙 가능성을 예측하는 웹 서비스를 개발하고,
이를 통해 사용자에게 정확하고 실시간적인 정보를 제공하는 시스템을 구축하는 데 목적을 두었습니다.

연합뉴스TV PICK · 6일 전 · 네이버뉴스

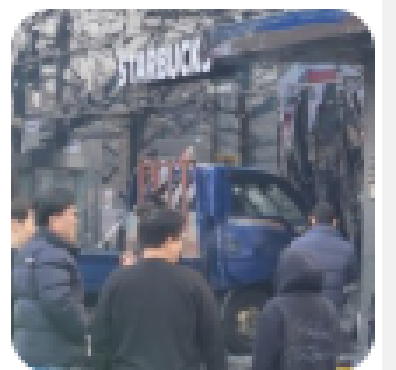
자유로서 '도로 결빙' 44중 추돌사고...극심한 정체
도로 결빙 구간에는 염화칼슘을 살포했습니다. 경찰 관계
음으로 인해 4중 추돌사고에 이어 3중 추돌사고 등이 발생
고를 막기 위해 도로를 통제하고 있다고 밝혔는데요. 이

자유로 44중·서울문산고속도로 43중 추돌 사고..."블랙... 서울신문 PICK · 6일 전 · 네이버뉴스

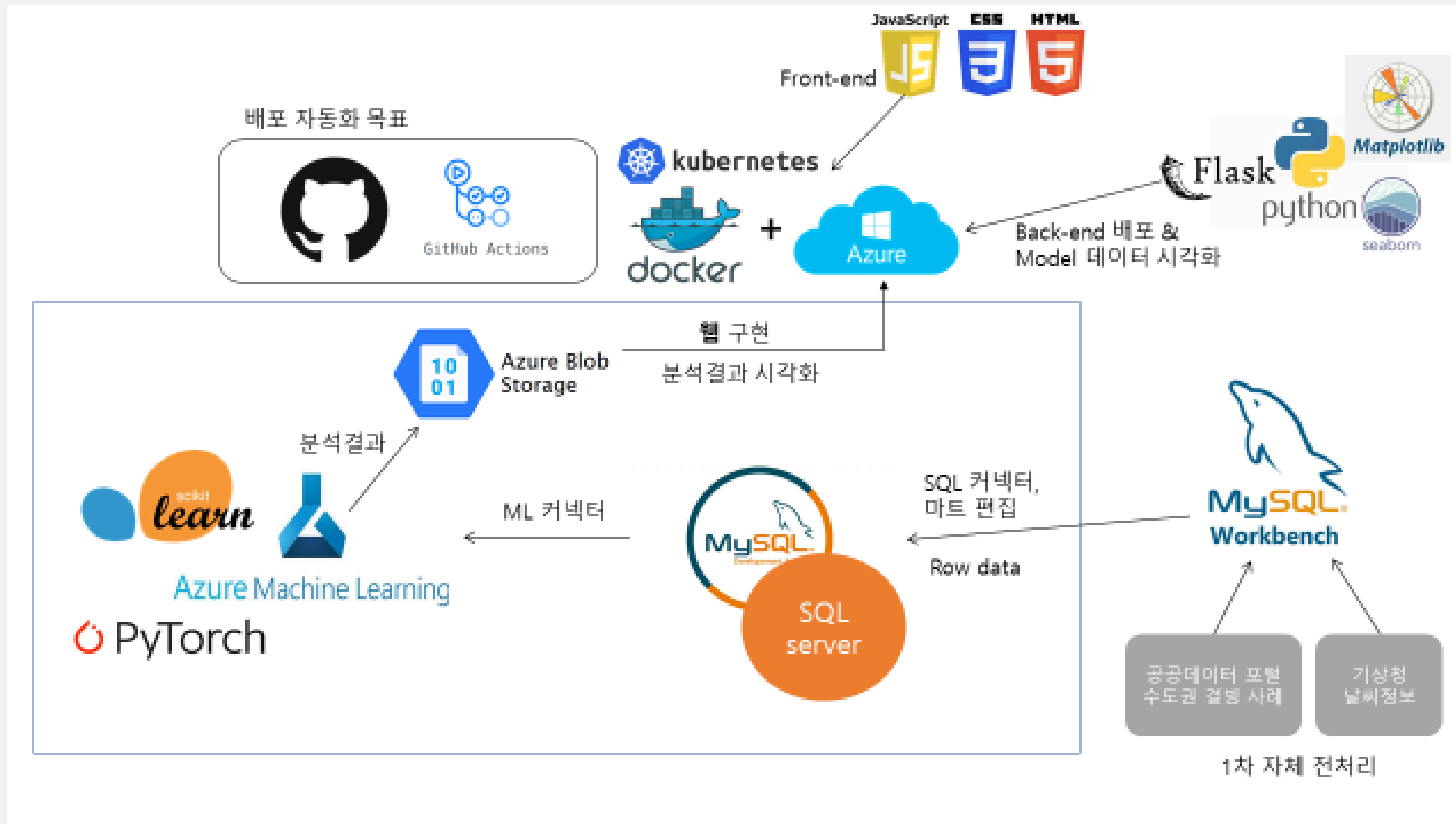
뉴스 PICK · 5일 전 · 네이버뉴스

국회 앞 도로서 '블랙아이스' 사고...1톤 트럭 카페 돌진

서울 영등포구에서 블랙아이스(도로 결빙)로 인해 차량들이 미끄러지며 1
톤 트럭이 도로변에 있던 건물을 들이받는 사고가 발생했다. 14일 경찰 등
에 따르면 이날 오전 8시10분께 서울 영등포구 여의도 국회 앞 국회대...



03. 아키텍처 소개



04. 데이터 수집 및 관리

API를 활용한 데이터 수집

- 기상청 API허브를 사용하여 일별 기상 정보를 수집.
- 주요 데이터: 기온, 습도, 적설량, 풍속, 기압, 날씨가, 기상청 지점번호
- 파이프라인:
 - i. Python 스크립트를 사용하여 API 데이터를 호출.
 - ii. JSON 형식의 응답 데이터를 Pandas를 통해 정리.
 - iii. 정리된 데이터를 MySQL 서버에 저장. (1.5MB)

결빙발생가능성 데이터

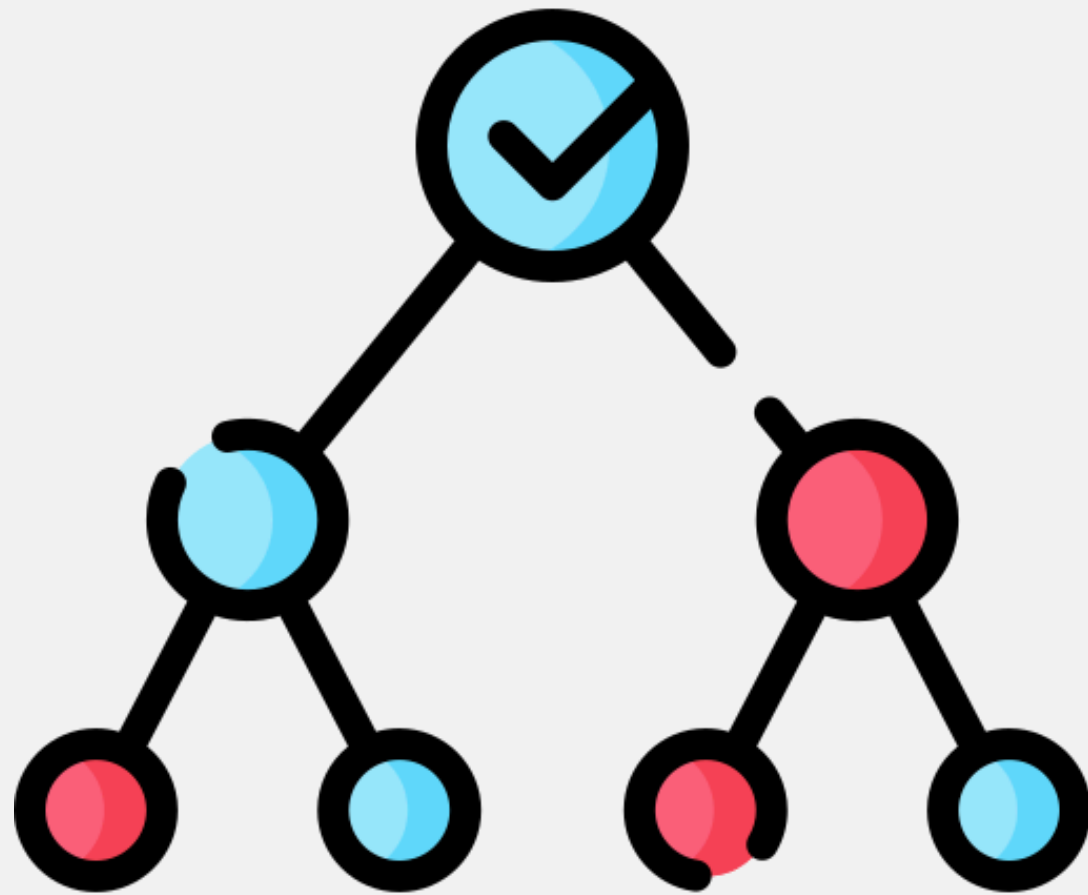
- 공공기관 데이터에서 최근 2년간 겨울철 빙결여부 정보를 수집
- 주요 데이터: 경도, 위도, 예보시간, 결빙 상태
- 대용량 데이터 업로드 및 관리 (565MB)
 - i. CSV 데이터를 MySQL Workbench를 사용하여 Azure MySQL 서버에 연결,
LOAD DATA LOCAL INFILE 쿼리로 csv 데이터 업로드
 - i. 데이터베이스 인덱스를 활용하여 검색 속도 향상
 - ii. 테이블 정규화를 통해 데이터 중복 최소화



최종 학습 데이터

- API 데이터와 결빙발생 데이터 결합
- 주요 데이터: 경도, 위도, 날씨, 풍속, 기온, 습도, 결빙상태 (305MB)
- DATA Split : Train 80%, Test 20%

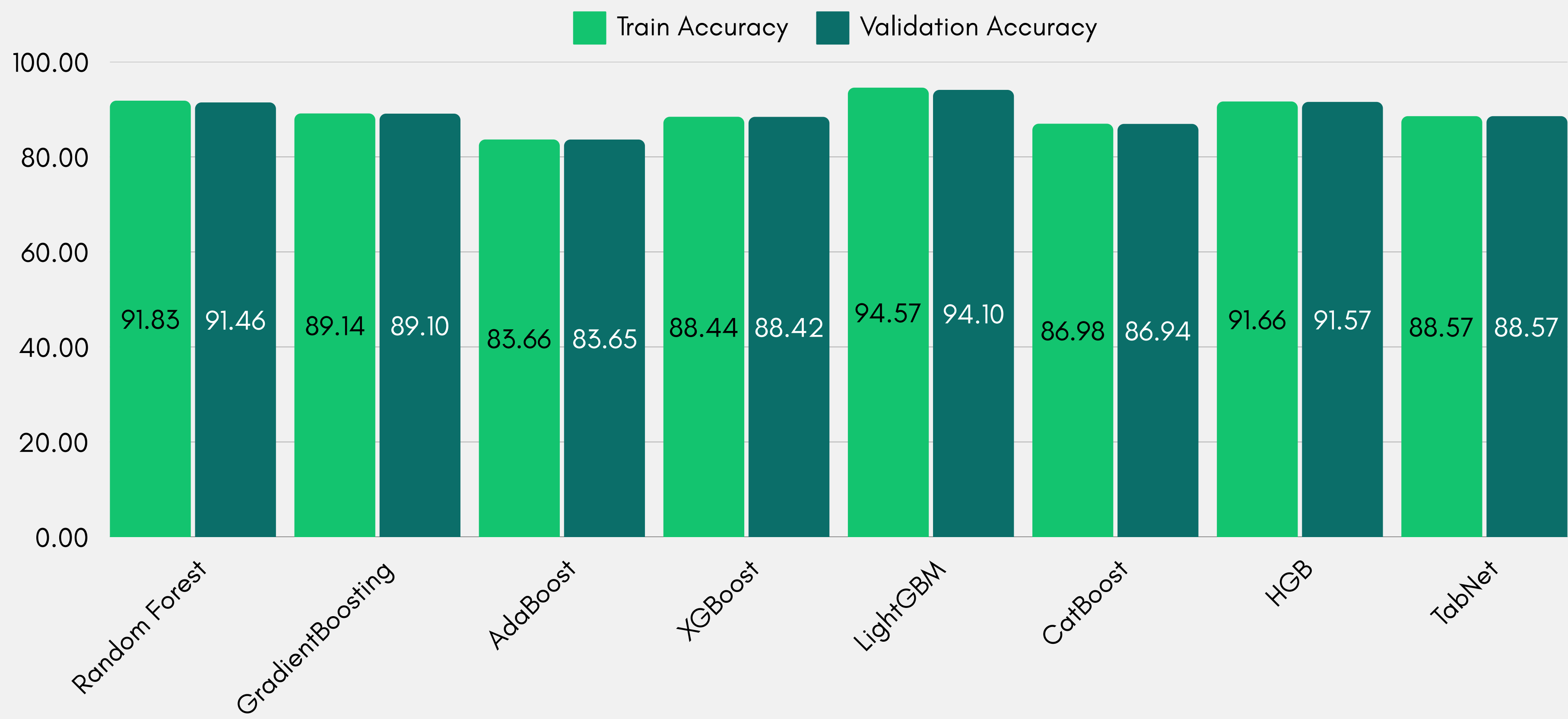
05. Tree 기반 Ensemble 모델 선정 배경



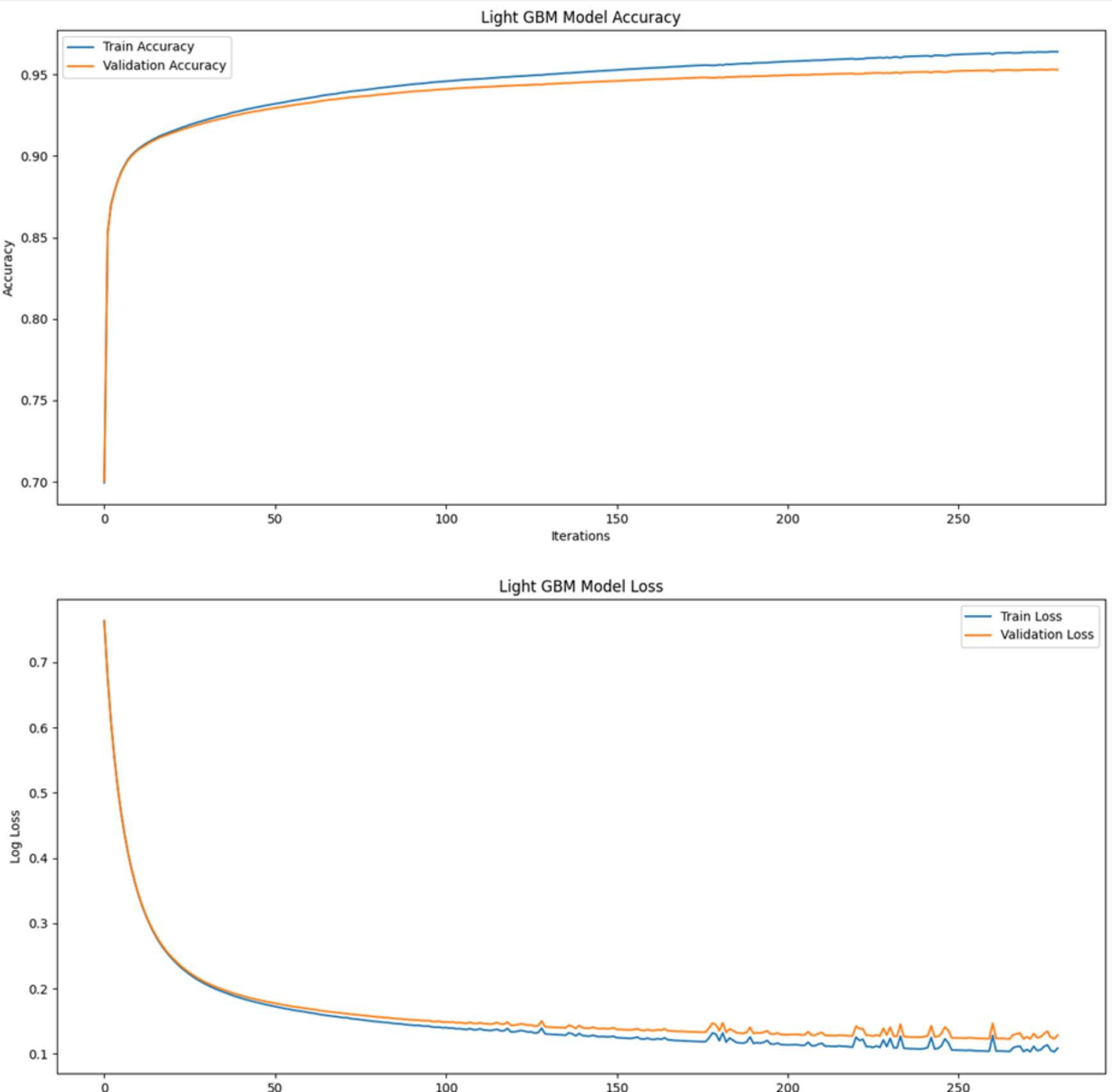
Tree-based Ensemble Model

- Excel CSV 파일과 같은 Tabular data는 대략적인 Hyperplane(초평면) 경계를 가지는 Manifold
 - 부스팅 모델들은 이러한 Manifold에서 결정(decision)을 할 때, 더 효율적으로 작동
- Tree기반의 모델 → 학습이 빠르고 쉽게 개발 가능
- Tree기반의 모델 → 높은 해석력을 가지고 있다
 - 변수 중요도를 구할 수 있으므로 딥러닝 모델에 비해 상대적으로 해석이 용이

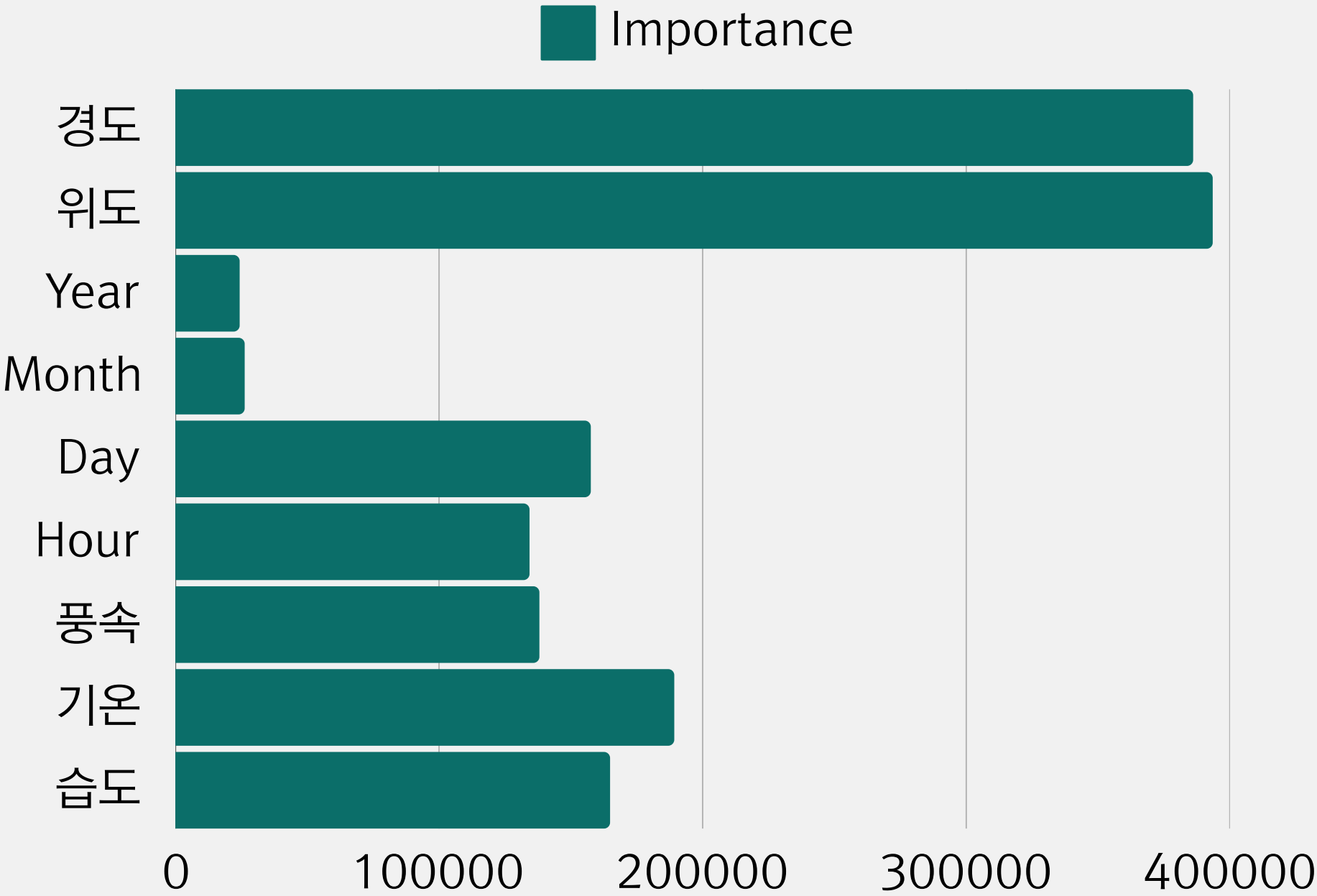
06. 머신러닝 기반 결빙 예측 및 결과



06. 머신러닝 기반 결빙 예측 및 결과

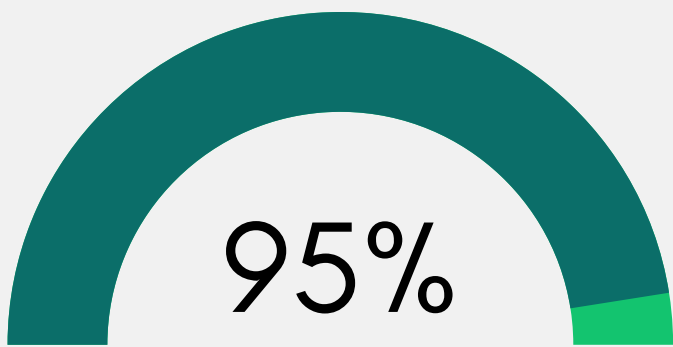


Light Gradient Boosting Machine

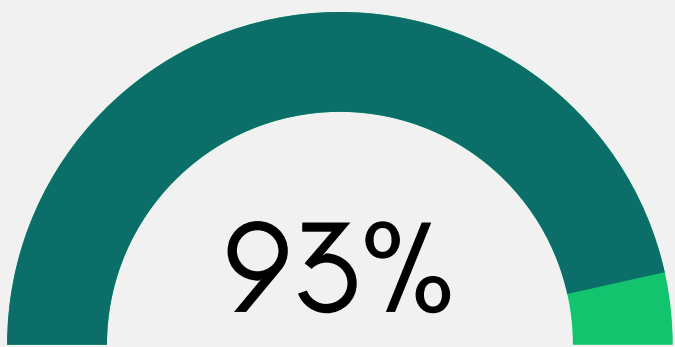


06. 머신러닝 기반 결빙 예측 및 결과

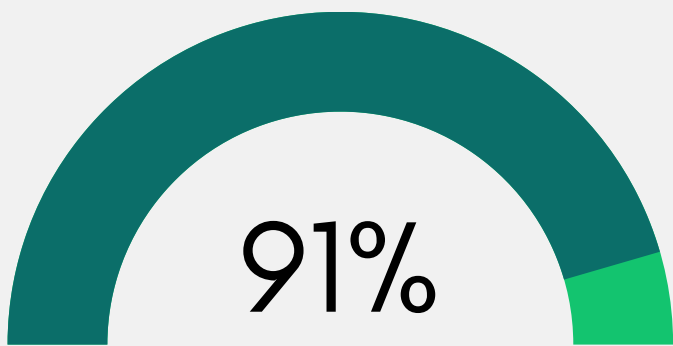
Light Gradient Boosting Machine



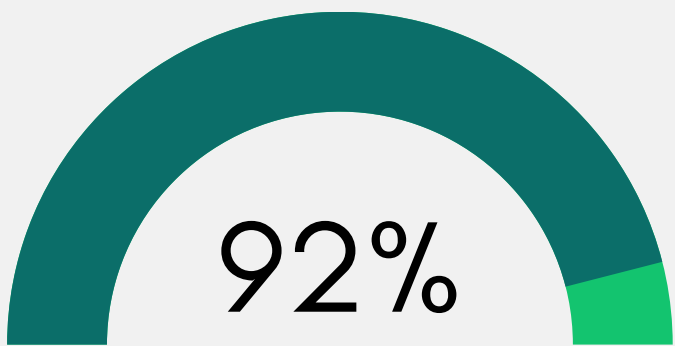
Accuracy



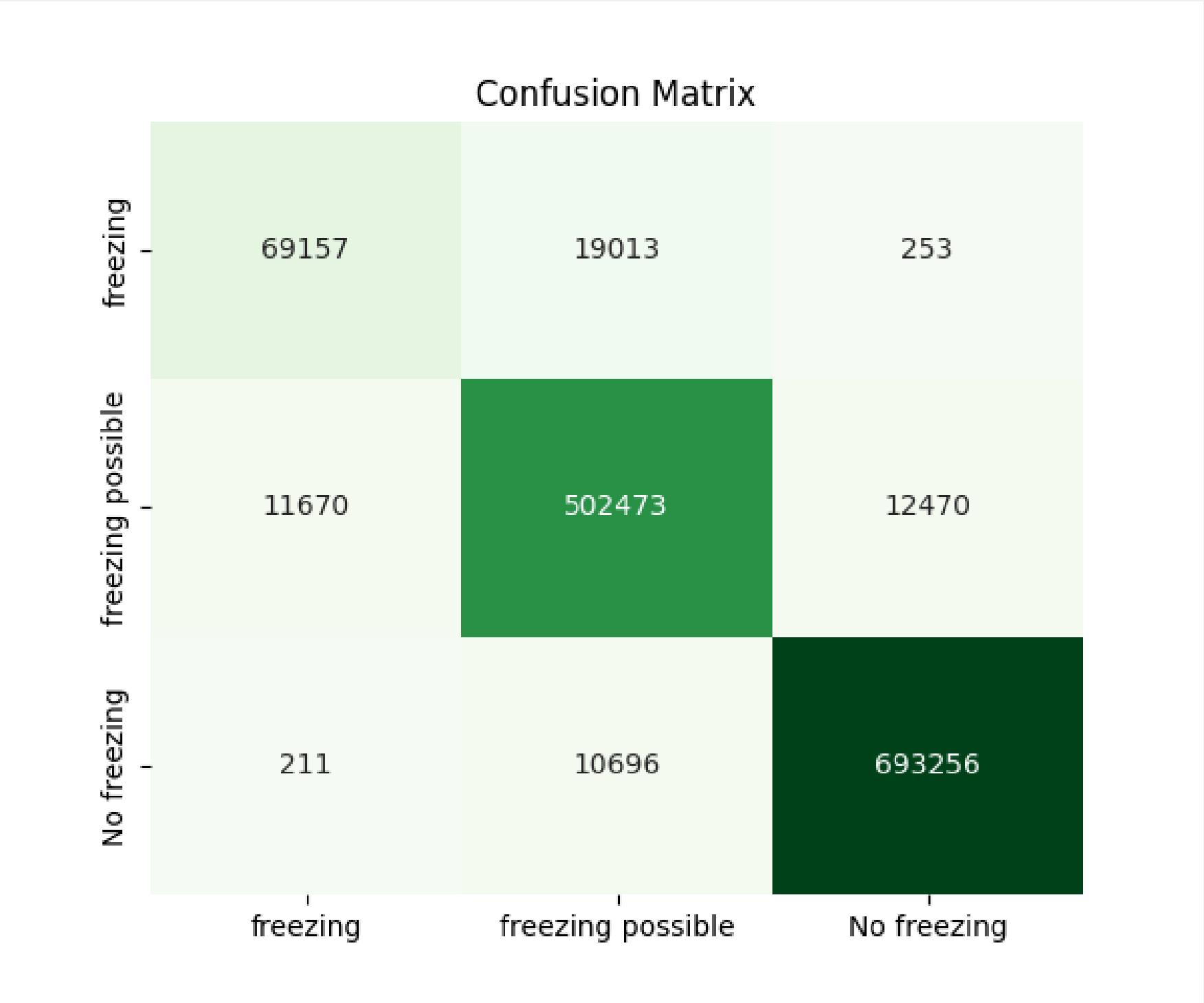
Precision



Recall



F1 score



07. 트러블 슈팅 요약(1)

기상청 API 데이터에 기상청 지점별 특정 시간 및 날짜의 데이터 누락 건 처리

98-2840 → 16건 누락

203-2846 → 10건 누락

99-2855 → 1건 누락

202,119,112,108-2856 (정상)

기상청 데이터에 위와 같이 누락된 데이터가 발견

→ 없는 지역은 어쩔 수 없이 제외하되 대신 제일

가까운 지역으로 끌고 와서 대체

```
1 def haversine_distance(lat1, lon1, lat2, lon2):
2     # 위도와 경도의 차이 계산
3     lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
4
5     # Haversine formula
6     dlat = lat2 - lat1
7     dlon = lon2 - lon1
8     a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
9     c = 2 * np.arcsin(np.sqrt(a))
10    radius = 3959 # Earth radius in miles
11
12    return radius * c
```

```
1 address = [[37.57142,126.9658,108],[37.47772,126.6249,112],[37.90188,127.0607,98],[37.88589,126.76648,99],[37.
2
3 # address 배열을 DataFrame으로 변환 (벡터화 연산을 위해)
4 address_df = pd.DataFrame(address, columns = ['LAT','LON','STNID'])
```

```
1 # 각 행에 대해 최단 거리 계산 및 필요한 값 추출
2 def find_closest_station(row):
3     lat1, lon1 = row['GRID_Y'], row['GRID_X']
4     distances = haversine_distance(lat1, lon1, address_df['LAT'].values, address_df['LON'].values)
5
6     # 거리값을 오름차순으로 정렬하여 첫 번째와 두 번째, 세 번째 최단 거리 인덱스를 찾음
7     sorted_idx = np.argsort(distances)
8
9     first_closest_idx = sorted_idx[0]
10    second_closest_idx = sorted_idx[1]
11    third_closest_idx = sorted_idx[2]
12
13    first_place = address_df.iloc[first_closest_idx]['STNID']
14    second_place = address_df.iloc[second_closest_idx]['STNID']
15    third_place = address_df.iloc[third_closest_idx]['STNID']
16
17    # 필요한 값 추출
18    station_data_first = dfw[(dfw['YYMMDDHHMI'] == row['FCST_TM']) & (dfw['STNID'] == first_place)]
19    station_data_second = dfw[(dfw['YYMMDDHHMI'] == row['FCST_TM']) & (dfw['STNID'] == second_place)]
20    station_data_third = dfw[(dfw['YYMMDDHHMI'] == row['FCST_TM']) & (dfw['STNID'] == third_place)]
21
22    # empty 여부에 따라 station_data 지정
23    if not station_data_first.empty:
24        station_data = station_data_first.iloc[0]
25    elif not station_data_second.empty:
26        station_data = station_data_second.iloc[0]
27    elif not station_data_third.empty:
28        station_data = station_data_third.iloc[0]
29    else:
30        station_data = dfw[(dfw['YYMMDDHHMI'] == row['FCST_TM']) & (dfw['STNID'] == address_df.iloc[sorted_idx[0]])]
```

```
32    return pd.Series({
33        'WS':station_data['WS'],
34        'TA_C':station_data['TA_C'],
35        'HM':station_data['HM']
36    })
```

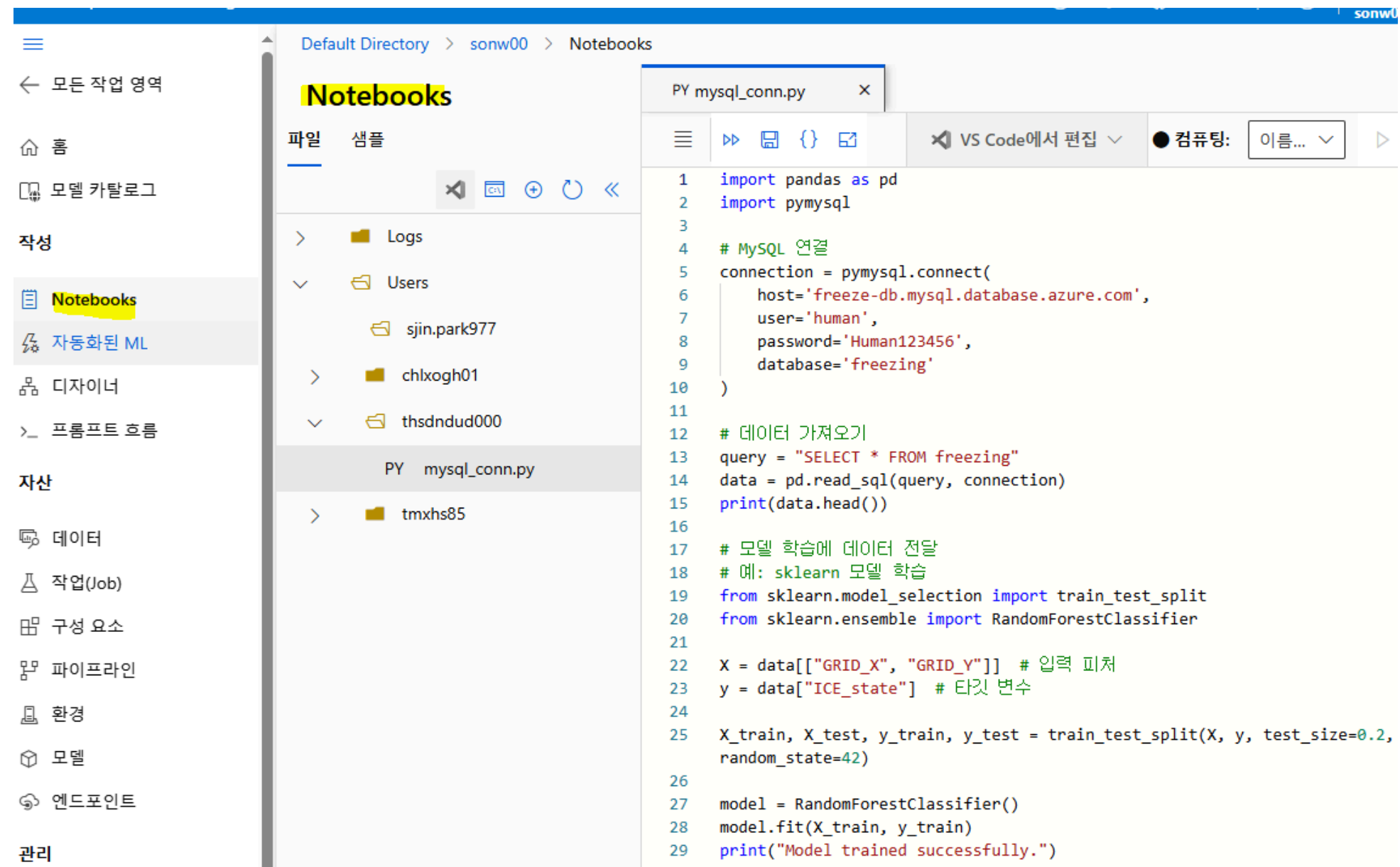
07. 트러블 슈팅 요약 (2)

Azure Mysql 데이터 베이스 연결

azure blob storage 는 바로 ML에 연결 가능하지만.
MySQL은 서버만 제공하며 ML연결은 사이트에서 제공 X

CSV의 경우 blob storage보다는 rdbms db가 더 적합하다
판단되어 azure 사이트에 연결할 방법을 고민

-> pymysql을 import 하여 직접 connection 코드 구현



```
1 import pandas as pd
2 import pymysql
3
4 # MySQL 연결
5 connection = pymysql.connect(
6     host='freeze-db.mysql.database.azure.com',
7     user='human',
8     password='Human123456',
9     database='freezing'
10 )
11
12 # 데이터 가져오기
13 query = "SELECT * FROM freezing"
14 data = pd.read_sql(query, connection)
15 print(data.head())
16
17 # 모델 학습에 데이터 전달
18 # 예: sklearn 모델 학습
19 from sklearn.model_selection import train_test_split
20 from sklearn.ensemble import RandomForestClassifier
21
22 X = data[["GRID_X", "GRID_Y"]] # 입력 피쳐
23 y = data["ICE_state"] # 타겟 변수
24
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
26     random_state=42)
27
28 model = RandomForestClassifier()
29 model.fit(X_train, y_train)
30 print("Model trained successfully.")
```

07. 트러블 슈팅 요약 (3)

Azure 배포 웹과 Blob storage 연결

처음엔 Base_dir 경로를 절대경로로 설정하고 모델을 불러오려고 하였으나, 연결되지 않음.

→ 디렉토리 경로와 모델 경로를 모두 상대경로로 지정했더니 해결

```
# Azure Blob Storage 설정
AZURE_CONNECTION_STRING = "DefaultEndpointsProtocol=https;AccountName=sonw006428547489;AccountKey=rUMn7ZlgjmoXH5x9t7Fy"
BLOB_CONTAINER_NAME = "freezing-ml" # 컨테이너 이름
MODEL_BLOB_NAME = "freezing_model" # 모델 파일 이름
BASE_DIR = os.path.dirname(os.path.abspath(__file__)) # FlaskApp 디렉토리 경로
LOCAL_MODEL_PATH = os.path.join(BASE_DIR, "freezing_model") # 모델 경로
#LOCAL_MODEL_PATH = "C:\\py\\freezing_model.pkl" # 로컬에 저장될 모델 경로

# Blob Storage에서 모델 다운로드
def download_model_from_blob(): # blob Storage 클라이언트 생성하여 컨테이너와 블롭 지정
    blob_service_client = BlobServiceClient.from_connection_string(AZURE_CONNECTION_STRING)
    blob_client = blob_service_client.get_blob_client(container=BLOB_CONTAINER_NAME, blob=MODEL_BLOB_NAME)

    # Blob 다운로드 및 로컬에 저장
    with open(LOCAL_MODEL_PATH, "wb") as model_file:
        model_file.write(blob_client.download_blob().readall())

# Flask 앱 초기화 시 모델 로드
with app.app_context():
    if not os.path.exists(LOCAL_MODEL_PATH): # 모델 파일이 존재하지 않을 경우
        download_model_from_blob() # blob Storage에서 모델 다운로드 함수 호출
    global model
    with open(LOCAL_MODEL_PATH, "rb") as model_file: # 저장된 모델 파일 열기
        model = pickle.load(model_file) # 모델 파일을 로드하여 메모리에 저장
    print("모델이 성공적으로 로드되었습니다!")
    print(LOCAL_MODEL_PATH)
```


07. 트러블 슈팅 요약 (4)

```
// AJAX 요청
$.ajax({
  url: "/predict_freezing",
  method: "POST",
  contentType: "application/json",
  data: JSON.stringify({ region, city, day }),
  success: function (data) {
    const tableBody = $("#weather-table tbody");
    tableBody.empty();

    if (data.error) {
      alert(data.error);
      return;
    }

    // 시간을 기준으로 정렬
    const sortedTimes = Object.keys(data).sort();

    // 각 시간대별 데이터를 테이블에 추가
    sortedTimes.forEach(time => {
      const weatherData = data[time];
      const rowClass = weatherData.freezing_status === 'Freezing confirmed' ?
        'freezing-confirmed' :
        weatherData.freezing_status === 'Freezing possible' ?
        'freezing-possible' : '';

      tableBody.append(`
        <tr class="${rowClass}">
          <td>${formatDateTime(weatherData.fcst_time)}</td>
          <td>${weatherData.temperature || "-"}</td>
          <td>${weatherData.wind_speed || "-"}</td>
          <td>${weatherData.humidity || "-"}</td>
          <td>${weatherData.freezing_status || "-"}</td>
        </tr>
      `);
    });
  }
});
```

```
@app.route("/predict_freezing", methods=['POST'])
def predict_freezing():
    req_data = request.json

    region = req_data.get("region")
    city = req_data.get("city")
    date = req_data.get("day")

    if not region or not city or not date:
        return jsonify({"error": "Missing region, city, or day"}), 400
    if region not in LOCATION_COORDS or city not in LOCATION_COORDS[region]:
        return jsonify({"error": "Invalid region or city"}), 400

    try:
        year = int(date[:4])
        month = int(date[4:6])
        day = int(date[6:8])
        hour = 5

        coords = LOCATION_COORDS[region][city]
        nx, ny, latitude, longitude = coords["nx"], coords["ny"], coords["위도"], coords["경도"]

        today = datetime.now()
        today_str = today.strftime("%Y%m%d")

        params = {"serviceKey": SERVICE_KEY, "numOfRows": 1000,
                  "pageNo": 1, "dataType": DATA_TYPE,
                  "base_date": today_str, #f"{year:04}{month:02}{day:02}",
                  "base_time": f"{hour:02}00", # "0500",
                  "nx": nx, "ny": ny,
                  }

        response = requests.get(BASE_URL, params=params)
```

```
    # 요청 데이터 검증
    # 요청 데이터 JSON 파싱
    # 지역 정보
    # 도시 정보
    # 예측 날짜 (YYYYMMDD 형식)
    # 요청 데이터 검증
    # 시간대별 데이터 초기화
    # 예보 시간
    # 필요한 정보가 있을 경우 저장
    # 예보 결과를 결과 딕셔너리에 저장
    # 예측 결과를 JSON 형식으로 반환

    if response.status_code == 200:
        weather_data = response.json()

    try:
        items = weather_data['response']['body']['items']['item']
        filtered_items = [item for item in items if item['fcstDate'] == date]
        time_grouped_data = {}
        for item in filtered_items:
            fcst_time = item['fcstTime']

            if fcst_time not in time_grouped_data:
                time_grouped_data[fcst_time] = {
                    'TMP': None, 'REH': None, 'WSD': None
                }

            if item['category'] in ['TMP', 'REH', 'WSD']:
                time_grouped_data[fcst_time][item['category']] = float(item['fcstValue'])

    except Exception as e:
        return jsonify({"error": "Failed to parse weather data", "details": str(e)}), 500

    results = {}

    for fcst_time, data in time_grouped_data.items():
        if all(v is not None for v in data.values()):
            input_features = [latitude, longitude, year, month, day, int(fcst_time[:2]),
                              data['WSD'], data['TMP'], data['REH']]
            prediction = model.predict([input_features])
            freezing_status = int(prediction[0])

            results[fcst_time] = {
                "fcst_time": f"{date} {fcst_time}", "temperature": data['TMP'],
                "humidity": data['REH'], "wind_speed": data['WSD'],
                "freezing_status": { 0: "freezing", 1: "Freezing possible", 2: "No Freezing"
            }[freezing_status]
        }

    return jsonify(results)
```

Azure 배포 웹에 모델링 결과값 구현

프론트) ajax를 이용한 비동기식 웹프론트 구현으로 속도 향상, 날짜나 지역설정등 기본적인 유효성 검사기능 구현

백엔드) Front에서 요청받은 데이터를 검증→파라미터 형태 변환→기상청 api 호출→기상청 예보 데이터 처리→결빙예측 모델링으로 결빙예측

→ 예측결과 딕셔너리 저장후 JSON으로 반환하여 프론트로 응답

08. 배포 웹사이트 시현 및 프론트&백엔드 설계

<http://20.249.66.50/>

Front-end

UI/UX 설계

직관적인 사용자 경험 제공을 목표로
기상 데이터와 결빙 가능성 시각화

기술 스택

- HTML, CSS, JavaScript를 사용한 인터페이스 개발
- 날씨 정보와 예측 결과를 시각적으로 제공하는
그래프 및 차트 구현

Back-end

RESTful API

- Flask를 활용하여 머신러닝 모델 결과를 반환하는 API 구현.
- API 엔드포인트:
 - /predict_freezing: 사용자 입력(위치, 날짜)을 받아 결빙 가능성 예측 결과 반환.
 - /index: 기상현황 및 결빙 가능성 예측 페이지 연결

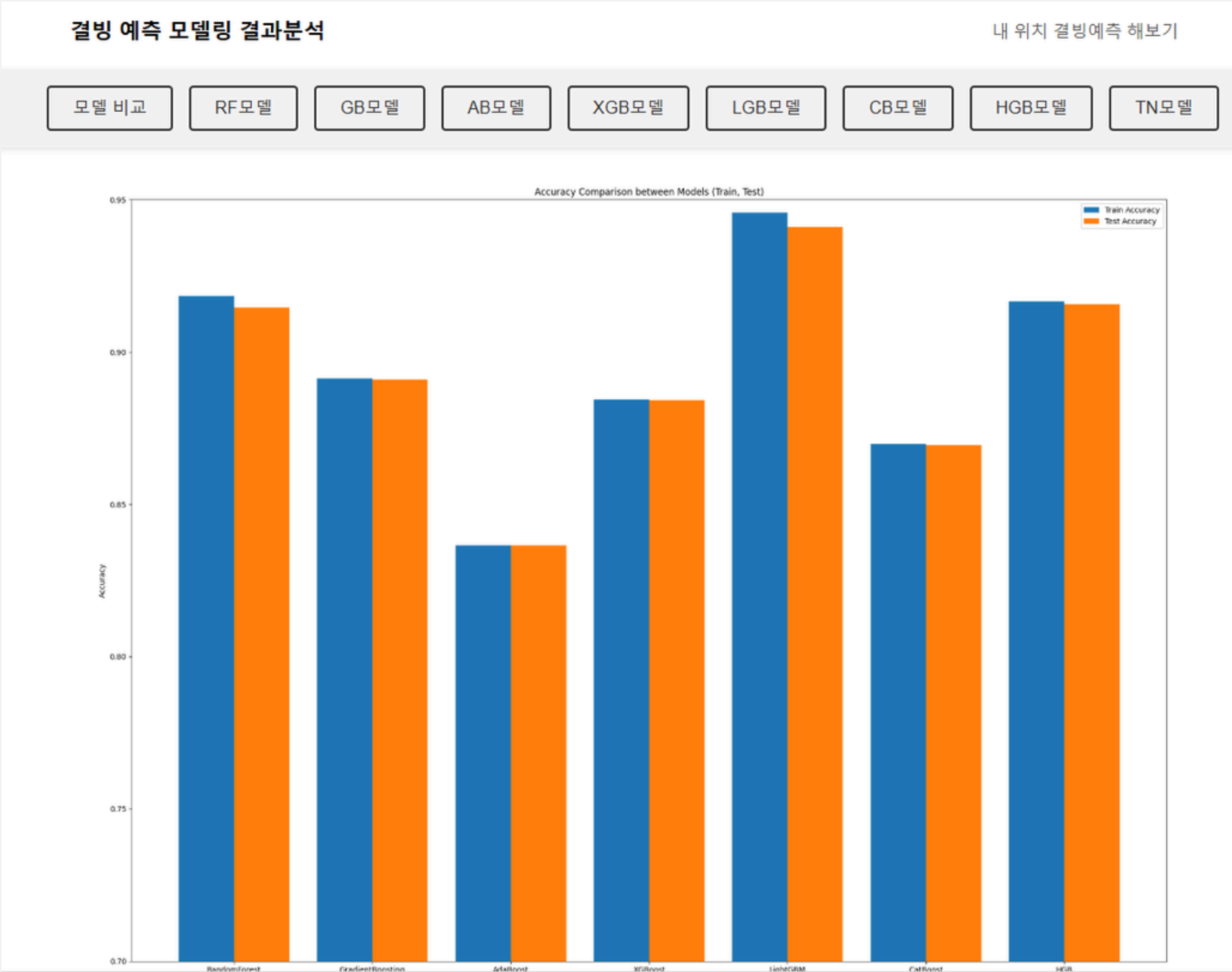
API 최적화

- 데이터 캐싱을 활용하여 API 응답 속도 향상.
- Ajax를 활용한 비동기 처리로 사용자 경험 개선.
- azure-storage-blob 라이브러리를 통해 AzureBlob Storage에 저장된 머신러닝 모델 호출



Azure MySQL 서버와 Blob Storage를 통해 대규모 데이터를 안정적으로 저장 및 관리
머신러닝의 높은 예측 정확도

08. 배포 웹사이트 시현 및 프론트&백앤드 설계



각 모델링 정확도 시각화 구현

우리 지역 결빙예측하기

모델링 분석결과 다시보기

지역: 서울특별시 도시: -- 예상날짜: 20250124 조회

결빙예측은 현재날짜 기준 +3일까지 기본적으로 제공하며, 이후는 3시간 간격으로 제공합니다.

날짜 / 시간	기온(℃)	풍속(m/s)	습도(%)	결빙 상태
2025-01-24 00:00	-	1.1	60	No Freezing
2025-01-24 01:00	-1	1.2	60	Freezing possible
2025-01-24 02:00	-1	1.2	65	Freezing possible
2025-01-24 03:00	-1	1.1	65	Freezing possible
2025-01-24 04:00	-2	1.1	65	Freezing possible
2025-01-24 05:00	-2	1.2	65	Freezing possible
2025-01-24 06:00	-2	1.2	65	Freezing possible
2025-01-24 07:00	-2	1.3	65	Freezing possible
2025-01-24 08:00	-2	1.2	60	Freezing possible
2025-01-24 09:00	-1	1.3	55	Freezing possible
2025-01-24 10:00	2	1.5	45	No Freezing
2025-01-24 11:00	4	1.3	45	No Freezing
2025-01-24 12:00	6	1.5	40	No Freezing
2025-01-24 13:00	8	1.6	35	No Freezing
2025-01-24 14:00	8	1.7	30	No Freezing
2025-01-24 15:00	9	1.6	30	No Freezing
2025-01-24 16:00	9	1.2	35	No Freezing

수도권 지역의 결빙 예측 결과 구현

Q&A



감사합니다

