



NUGU OIL

현재 위치에서 가까운 가장 싼 주유소를 찾으려면 어떻게 해야할까? 아빠의 경험에서 우리나라오는 짬밥? 주변 사람의 카더라? 우리 팀은 SKT NUGU Speaker를 활용해 실시간으로 업데이트되는 가장 싼 주유소 정보를 제공하는 서비스를 개발했다. 자세한 내용은 아래에서 확인할 수 있다.

 Meeting Notes

 Tasks

SKT NUGU Project

Aa Name

1. Introduction

2. Datasets : 지역별 주유소 위치 정보 및 가격 정보 데이터셋과 사용자의 현재 위치 받기

3-1. Methodology : 유가가격 예측 알고리즘

3-2. Methodology : NUGU Backend proxy server with Flask

3-3. Methodology : NUGU Play Builder 구조

4. Evaluation & Analysis

5. Related Work

6. Conclusion

Untitled

1. Introduction

1. 프로젝트 배경

가족들과 집콕하던 아무개씨. 내일 출근하려면 미리 주유소는 갔다와야 한다. 자주 가는 주유소는 있지만, 과연 그 곳이 제일 싼게 맞을까? NUGU Speaker에서 현재 위치에서 가깝고, 가장 저렴한 순으로 정렬 해서 바로 알려주면 어떨까?



NUGU 오일 서비스는 기름값을 조금이라도 줄이고자 하는 사회초년생, 자신 단골 집으로 갔으면 더 싸게 살 수 있었다고 말다툼하는 부부 혹은 친구 사이를 위해 만들어졌다.

우리 팀이 서비스하고자하는 항목은 다음과 같다

- 사용자의 현재 위치를 기반으로 **최저가 주유소** 추천
 - 해당 주유소 경유/휘발유 가격 정보 제공
 - 2007년 경유/휘발유 가격부터 현재까지의 가격을 수집해 다음주 예측 가격 정보 제공
-

2. 원하는 결과

사용자가 NUGU Speaker에게 최저가 주유소를 알려달라는 요청을 말하면, 해당 데이터를 서버에서 가져와 사용자에게 전달한다. 다음은 NUGU 오일 서비스를 사용할 때의 사용자와 NUGU의 대화 흐름이다.

사용자 : 아리야, 최저가 주유소 찾아줘

NUGU : 경유를 원하시면 1번, 휘발유를 원하시면 2번을 말씀해주세요

사용자 : 2번

NUGU : 반경 5km이내에 주유소 3곳을 알려드릴게요. 현대오일뱅크(주)직영 토피 1호 셀프주유소, 1387원, 푸른 주유소, 1399원, 현대오일뱅크(주)직영 토피 2호주유소, 1427 원이에요

서비스 제공을 위해 필요한 데이터셋은 다음과 같다

- **유가 정보 API**

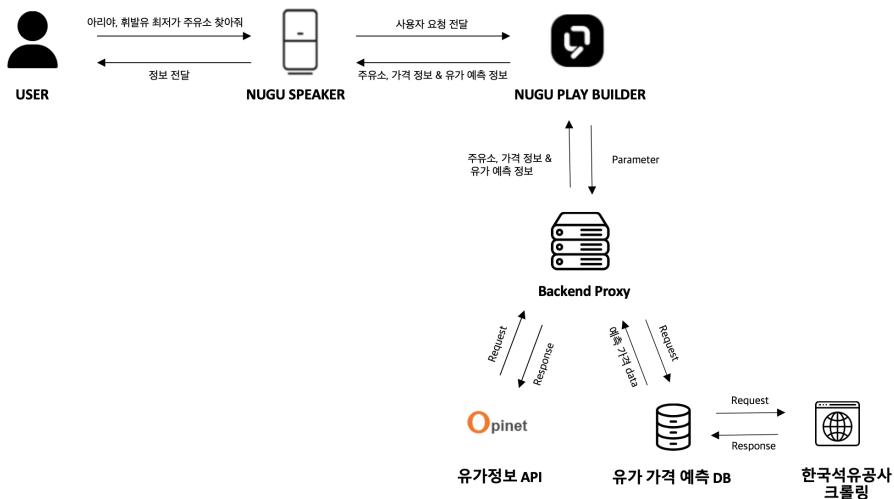
한국석유공사에서 제공하는 API로 주유소 판매 가격, 주유소 위치, 부가 서비스 등 전국 주유소 정보 및 평균 유가를 제공한다



사진 출처 : <http://www.opinet.co.kr/user/main/mainView.do>

3. 서비스 flow

NUGU 오일의 서비스 구조는 다음과 같다.



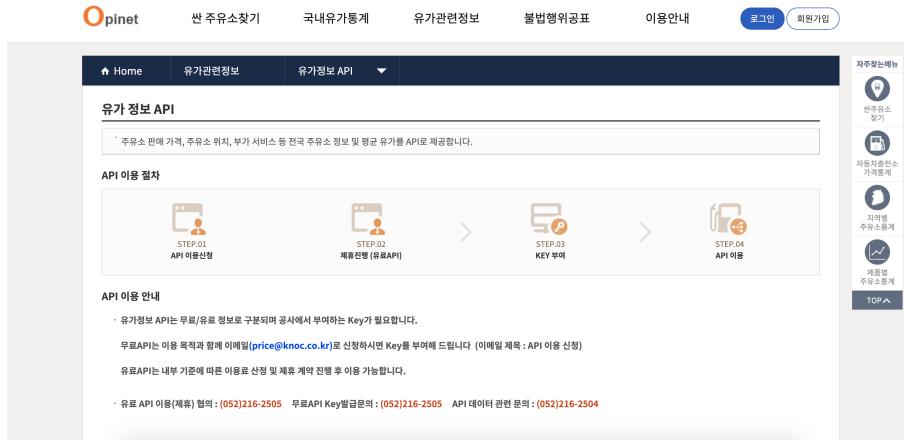
*유가정보 API의 경우 최근 일주일치 가격만 제공했기 때문에
한국석유공사 오피넷의 유가관련정보>국가별가격통계 자료를 크롤링해 사용했다

자세한 내용은 다음 포스팅에서 확인할 수 있다.

2. Datasets : 지역별 주유소 위치 정보 및 가격 정보 데이터셋과 사용자의 현재 위치 받기

1. 유가정보 API 신청

우리팀은 지역별 주유소 위치 정보 및 가격 정보를 제공하기 위해 오피넷(Opinet)의 유가정보 API를 활용하였다. 오피넷이란 한국석유공사에서 운영중인 유가 정보 사이트로, 석유사업자가 보고한 자료를 토대로 전국 주유소와 자동차충전소의 현재 판매 가격과 정유사, 대리점의 평균 공급가격 등 다양한 유가 정보를 제공하고 있다.



오피넷에서 제공하는 유가정보 API 리스트를 보면 주유소 전국 평균가격(현재)부터 전국/지역별 최저가 주유소 (Top 20)까지 있는 것을 확인할 수 있는데, 우리는 그 중에서 특정한 위치에서 지정된 거리 내에 있는 주유소의 유가정보가 필요하므로 아래의 **반경 내 주유소 API**만을 사용했다. 해당 API를 사용하면 검색할 반경(radius)을 원하는대로 설정이 가능하고, 제품(prodcd)과 정렬방식(sort)도 선택이 가능하다.

■ 반경 내 주유소

기본정보	http://www.opinet.co.kr/api/aroundAll.do - 특정 위치 중심으로 반경 내 주유소
요청 파라미터	
Element	Description
code	필수 공사에서 부여한 키(key) 정보
out	필수 정보 노출 형식을 정의 한다.(xml/json)
x	필수 기준 위치 X좌표 (KATEC)
y	필수 기준 위치 Y좌표 (KATEC)
radius	필수 반경 선택 (최대 5000, 단위 : m)
prodcd	필수 제품 (휘발유:B027, 경유:D047, 고급휘발유: B034, 실내등유: C004, 자동차부탄: K015)
sort	필수 1: 가격순, 2: 거리순
반환값	
Element	Description
UNI_ID	주유소코드
POLL_DIV_CD	상표(SKE:SK에너지, GSC:GS칼텍스, HDO:현대오일뱅크, SOL:S-OIL, RTO:자영일통, RTX:고속도로알뜰, NHO:농협일통, ETC:자가상표, E1G: E1, SKG:SK가스)
OS_NM	상호
PRICE	판매가격
DISTANCE	기준 위치로부터의 거리 (단위 : m)
GIS_X_COOR	GIS X좌표(KATEC)
GIS_Y_COOR	GIS Y좌표(KATEC)

2. 사용자의 현재 위치 받기

유가 정보 API를 사용하기 전, 서비스의 편리한 사용을 위해 우리는 사용자의 위치 정보를 Google Geolocation API로 받아오고자 했다. 사용자가 "최저가 주유소 찾아줘" 했을 때, 스피커로 하여금 "어디 사시는데요?"라고 되묻지 않게 말이다. [Google Geolocation API](#)는 사용자의 기지국 및 WiFi 노드에 대한 정보를 기반으로 위치 및 정확도 반경을 반환한다. 우리는 Google Geolocation API를 이용해서 사용자의 Ip주소를 통해 현위치의 위도, 경도값을 받아 사용하기로 했다.

```
def location(): #find users location
    url = 'https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyDkx6muQn1Jz-y6hL0cTPVdYAhk1m6WJQo'
    data = {
        'considerIp': True,
        #'homeMobileCountryCode': 450,
        #'homeMobileNetworkCode': 5,
        #'radioType':'gsm',
        #'carrier': "SKTelecom",
        #'wifiAccessPoints':[{'macAddress':'40:DC:9D:06:EC:CA'}]
    }
    result = requests.post(url, data)
    a=result.json()
    lat=a['location']['lat'] # Y point
    lng=a['location']['lng'] # X point
    return lat,lng
```

Google Geolocation API를 불러오는 데에는 성공했지만, 오피넷과 Geolocation에서 요구하는 위도랑 경도값 측정 단위가 서로 달랐다. 오피넷은 kotec이라는 기준으로 측정한 값이고, Geolocation은 wgs84라는 기준으로 측정한 값이었다. 따라서 우리는 wgs84에서 kotec(=tm128)으로 변환하는 과정을 함수로 만들어주었다. 함수의 내용은 [링크](#)를 참고했다.

WGS84 형식에서 오피넷의 TM128 형식으로 변환하기

우리는 좌표 변환에 관련된 함수를 포함하고 있는 Pyproj 패키지를 이용했다. parameter로 받은 WGS84형식의 lat,lng 값을 TM128형식으로 변환하여 `x_point`, `y_point` 변수에 담아 반환한 것이다. 자세한 코드 설명은 다음 포스팅에서 계속된다.

```
def trans(lat,lng): #wgs84 -> tm128
    WGS84 = { 'proj':'latlong', 'datum':'WGS84', 'ellps':'WGS84', }

    TM128 = { 'proj':'tmerc', 'lat_0':'38N', 'lon_0':'128E', 'ellps':'bessel',
    'x_0':'400000', 'y_0':'600000', 'k':'0.9999',
    'towgs84': '-146.43,507.89,681.46' }

    def wgs84_to_tm128(longitude, latitude):
        return transform( Proj(**WGS84), Proj(**TM128), longitude, latitude )

    x_point,y_point=wgs84_to_tm128(lng,lat)
    return x_point,y_point
```

3. 유가정보 API 요청

사용자의 위치정보를 알고 난 후에는 그에 맞는 주유소 상호와 판매 가격을 유가 정보 API에서 불러왔다. 우리팀은 `browse` 함수를 통해 불러 왔으며 우리가 사용하고자하는 Element는 다음과 같았다

- **OS_NM** : 주유소 상호
 - **PRICE** : 판매 가격

```
def ask_oil_type(ans):
    if ans == "경유" :
        return "D047"
    elif ans == "휘발유" :
        return "B027"
    else :
        return None
def browse(x_point,y_point,oil_type):
    url = 'http://www.opinet.co.kr/api/aroundAll.do'
    payload = {
        "code" : 'key값', // 인증키 같은 오피넷에 메일을 보내 받을 수 있음
        "out" : "json",
        "x" : x_point,
        "y" : y_point,
        "radius" : "5000",
        "prodcd" : oil_type ,
        "sort" : "1"
    }
    result = requests.get(url,params=payload).json()
    return result
```

좌표 변환 후 뽑아낸 유가정보 API 속 주유소 상호와 가격은 아래와 같다

(RESULT : [{ 'OIL' : [{ 'UNI_ID' : 'A0033432', 'POLL_DIV_CD' : 'RTX', 'OS_NM' : '대보건설(주) 구리(일산방법)주유소', 'PRICE' : 1243, 'DISTANCE' : 2616.8, 'GIS_X_COOR' : 324215.09865, 'GIS_Y_COOR' : 558739.24076 }, { 'UNI_ID' : 'A0083319', 'POLL_DIV_CD' : 'RTX', 'OS_NM' : '한국가스(주) 구리(교현교수지점)', 'PRICE' : 1245, 'DISTANCE' : 2536.4, 'GIS_X_COOR' : 32565.09865, 'GIS_Y_COOR' : 54973.19803 }, { 'UNI_ID' : 'A0068627', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한진(주)화성시점', 'PRICE' : 1246, 'DISTANCE' : 3583.0, 'GIS_X_COOR' : 328820.35982, 'GIS_Y_COOR' : 558748.74889 }, { 'UNI_ID' : 'A0083320', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한진(주)수원점', 'PRICE' : 1247, 'DISTANCE' : 4464.8, 'GIS_X_COOR' : 330898.52257, 'GIS_Y_COOR' : 559242.48511 }, { 'UNI_ID' : 'A0011133', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1258, 'DISTANCE' : 1831.8, 'GIS_X_COOR' : 327895.8, 'GIS_Y_COOR' : 557466.8 }, { 'UNI_ID' : 'A0088018', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1258, 'DISTANCE' : 3000.0, 'GIS_X_COOR' : 328813.12089, 'GIS_Y_COOR' : 558742.52527 }, { 'UNI_ID' : 'A0086801', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '(주)한국에너지 디자인주유소', 'PRICE' : 1263, 'DISTANCE' : 877.0, 'GIS_X_COOR' : 326927.68887, 'GIS_Y_COOR' : 55735.07114 }, { 'UNI_ID' : 'A0068659', 'POLL_DIV_CD' : 'GSC', 'OS_NM' : '(주)한국에너지 디자인주유소', 'PRICE' : 1263, 'DISTANCE' : 896.7, 'GIS_X_COOR' : 326862.48564, 'GIS_Y_COOR' : 55734.00001 }, { 'UNI_ID' : 'A0086811', 'POLL_DIV_CD' : 'GSC', 'OS_NM' : '(주)한국에너지 디자인주유소', 'PRICE' : 1268, 'DISTANCE' : 5463.0, 'GIS_X_COOR' : 328662.48564, 'GIS_Y_COOR' : 55614.91661 }, { 'UNI_ID' : 'A0086812', 'POLL_DIV_CD' : 'GSC', 'OS_NM' : '(주)한국에너지 디자인주유소', 'PRICE' : 1275, 'DISTANCE' : 32998.8, 'GIS_X_COOR' : 329344.00001, 'GIS_Y_COOR' : 556974.02591 }, { 'UNI_ID' : 'A0013154', 'POLL_DIV_CD' : 'HDO', 'OS_NM' : '한화(주)수소', 'PRICE' : 1275, 'DISTANCE' : 4212.4, 'GIS_X_COOR' : 32216.55664, 'GIS_Y_COOR' : 558866.79965 }, { 'UNI_ID' : 'A0086860', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '(주)뉴미디어디자인주유소', 'PRICE' : 1278, 'DISTANCE' : 2895.0, 'GIS_X_COOR' : 328652.006, 'GIS_Y_COOR' : 558842.99874 }, { 'UNI_ID' : 'A0089281', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1285, 'DISTANCE' : 4357.1, 'GIS_X_COOR' : 327384.01786, 'GIS_Y_COOR' : 561175.44663, 'UNI_ID' : 'A0086813', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1285, 'DISTANCE' : 27996.864, 'GIS_X_COOR' : 325582.006, 'GIS_Y_COOR' : 558663.58887 }, { 'UNI_ID' : 'A0086388', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1285, 'DISTANCE' : 3546.6, 'GIS_X_COOR' : 32136.0, 'GIS_Y_COOR' : 556149.01616 }, { 'UNI_ID' : 'A0086729', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1285, 'DISTANCE' : 32349.0, 'GIS_X_COOR' : 32498.0, 'GIS_Y_COOR' : 556499.01616 }, { 'UNI_ID' : 'A0086730', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1285, 'DISTANCE' : 2429.9, 'GIS_X_COOR' : 323776.7563, 'GIS_Y_COOR' : 556643.26333 }, { 'UNI_ID' : 'A0086729', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1285, 'DISTANCE' : 2509.8, 'GIS_X_COOR' : 323995.4794, 'GIS_Y_COOR' : 556586.06717 }, { 'UNI_ID' : 'A00891983', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1288, 'DISTANCE' : 1822.5, 'GIS_X_COOR' : 326496.0, 'GIS_Y_COOR' : 558184.0 }, { 'UNI_ID' : 'A0087017', 'POLL_DIV_CD' : 'ETC', 'OS_NM' : '동창미스트(주) 동탄주유소', 'PRICE' : 1293, 'DISTANCE' : 2243.8, 'GIS_X_COOR' : 324334.8, 'GIS_Y_COOR' : 558296.0 }, { 'UNI_ID' : 'A0086780', 'POLL_DIV_CD' : 'GSC', 'OS_NM' : '한국주유소', 'PRICE' : 1293, 'DISTANCE' : 4924.8, 'GIS_X_COOR' : 321338.59688, 'GIS_Y_COOR' : 558667.0 }, { 'UNI_ID' : 'A0086468', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1293, 'DISTANCE' : 3709.2, 'GIS_X_COOR' : 324787.78543, 'GIS_Y_COOR' : 556988.00001 }, { 'UNI_ID' : 'A0086469', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1295, 'DISTANCE' : 4878.5, 'GIS_X_COOR' : 321550.8, 'GIS_Y_COOR' : 557988.0 }, { 'UNI_ID' : 'A0066317', 'POLL_DIV_CD' : 'HDO', 'OS_NM' : '성기리주유소', 'PRICE' : 1298, 'DISTANCE' : 2298.4, 'GIS_X_COOR' : 328215.0, 'GIS_Y_COOR' : 557979.0 }, { 'UNI_ID' : 'A0086648', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1298, 'DISTANCE' : 872.7, 'GIS_X_COOR' : 326828.1783, 'GIS_Y_COOR' : 557521.34511 }, { 'UNI_ID' : 'A0093342', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한진(주)화성시점', 'PRICE' : 1298, 'DISTANCE' : 1883.1, 'GIS_X_COOR' : 329567.93657, 'GIS_Y_COOR' : 557521.34511 }, { 'UNI_ID' : 'A00832675', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '한국주유소', 'PRICE' : 1305, 'DISTANCE' : 4197.8, 'GIS_X_COOR' : 326818.02924, 'GIS_Y_COOR' : 561118.07039 }, { 'UNI_ID' : 'A0089717', 'POLL_DIV_CD' : 'SKE', 'OS_NM' : '(주)제이에스미네랄화학 배관부문', 'PRICE' : 1308, 'DISTANCE' : 32439.8, 'GIS_X_COOR' : 325364.0, 'GIS_Y_COOR' : 556567.0 }, { 'UNI_ID' : 'A0086572', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1308, 'DISTANCE' : 5695.0, 'GIS_X_COOR' : 323604.31977, 'GIS_Y_COOR' : 56181.797937 }, { 'UNI_ID' : 'A0086572', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1325, 'DISTANCE' : 4917.8, 'GIS_X_COOR' : 323202.0, 'GIS_Y_COOR' : 559587.0 }, { 'UNI_ID' : 'A0083393', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한진(주)화성', 'PRICE' : 1325, 'DISTANCE' : 3482.0, 'GIS_X_COOR' : 322782.0, 'GIS_Y_COOR' : 556419.515591 }, { 'UNI_ID' : 'A0086316', 'POLL_DIV_CD' : 'GSC', 'OS_NM' : '한국주유소', 'PRICE' : 1325, 'DISTANCE' : 4865.6, 'GIS_X_COOR' : 322782.0, 'GIS_Y_COOR' : 556419.515591 }, { 'UNI_ID' : 'A0086733', 'POLL_DIV_CD' : 'HDO', 'OS_NM' : '성신주유소', 'PRICE' : 1335, 'DISTANCE' : 3139.9, 'GIS_X_COOR' : 325369.33868, 'GIS_Y_COOR' : 560816.07039 }, { 'UNI_ID' : 'A0086572', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1335, 'DISTANCE' : 32345.8, 'GIS_X_COOR' : 324358.0, 'GIS_Y_COOR' : 559587.0 }, { 'UNI_ID' : 'A0086866', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한진(주)화성', 'PRICE' : 1346, 'DISTANCE' : 3492.6, 'GIS_X_COOR' : 323394.0, 'GIS_Y_COOR' : 556512.0 }, { 'UNI_ID' : 'A0086332', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1346, 'DISTANCE' : 3492.6, 'GIS_X_COOR' : 323394.0, 'GIS_Y_COOR' : 556512.0 }, { 'UNI_ID' : 'A0086332', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1346, 'DISTANCE' : 3492.6, 'GIS_X_COOR' : 323394.0, 'GIS_Y_COOR' : 556512.0 }, { 'UNI_ID' : 'A0086332', 'POLL_DIV_CD' : 'SOL', 'OS_NM' : '한국주유소', 'PRICE' : 1346, 'DISTANCE' : 3492.6, 'GIS_X_COOR' : 323394.0, 'GIS_Y_COOR' : 556512.0 }

3-1. Methodology : 유가가격 예측 알고리즘

1. 시계열 분석을 통한 유가의 예측

유가정보 API의 경우 일주일 간의 경유, 휘발유 가격 밖에 제공하고 있지 않아 한국석유공사 홈페이지에서 기간을 설정해 가격 정보를 다운 받았다. 우리의 목표는 앞으로 4주간의 유가를 예측하는 것이다. 이러한 목표를 이루기 위해 Facebook에서 공개한 FBprophet 라이브러리를 활용한 시계열 데이터 예측을 진행할 것이다.

Prophet 알고리즘은 Facebook이 만든 시계열 예측 라이브러리로 R과 Python으로 사용할 수 있다. Arima 등의 다양한 시계열 모형이 존재하지만, Prophet의 사용법이 간단할 뿐만 아니라 높은 정확도를 가지며, 불규칙적인 주기를 다룰 수 있다는 점에서 우리는 Prophet을 선택하였다. 하지만 Prophet은 내부 알고리즘을 공개하지 않기 때문에 실제로 코드 내에서 어떻게 작동하는지는 알 수가 없다.

아래는 Prophet을 활용하여 작성한 시계열 예측 코드이다.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from fbprophet import Prophet

df = pd.read_csv("/Users/choiyun/Downloads/sample3.csv", engine='python', encoding='CP949')

model=Prophet()
model.fit(df)

future=model.make_future_dataframe(periods=4, freq='w')
forecast=model.predict(future)

result=forecast[['ds', 'yhat']]
result.tail()

new_file.to_csv("/Users/choiyun/Downloads/result.csv", index=False)
```

오피넷을 통해 받아온 유가 정보를 `read_csv` 함수를 통해 불러온다. Prophet 호출을 통한 인스턴스의 생성 후, `fit` 함수를 통해 오피넷에서 불러온 csv 파일의 데이터 프레임을 넘겨준다. 예측한 데이터를 넣을 데이터프레임을 만들어 주는 함수, `make_future_dataframe` 을 호출한다. 이를 통해 앞으로 4주 간의 데이터를 예측할 dataframe을 만들어 주었고, 변수 명은 `future`로 저장하였다.

`predict` 메소드는 `future`의 각 행에 `yhat`(예측된 값)라는 이름의 예측 값을 할당한다. `forecast`에 저장된 값 중 실제 예측 값인 `yhat` 값만을 'result'에 저장하겠다.

	ds	yhat
243	2020-07-01	1660.581871
244	2020-07-05	1646.854458
245	2020-07-12	1627.461742
246	2020-07-19	1619.522181
247	2020-07-26	1629.225889

result의 마지막 5개행을 출력한 결과

마지막으로 pandas library의 `to_csv` 함수를 이용하여 예측 결과를 csv 파일 형태로 저장하였다.

2. 교차 검증을 통한 성능의 측정

#예측 알고리즘의 overfitting을 막고 모델의 변동성을 줄이기 위해서는 교차 검증이 필요하다. 일반적으로 교차 검증을 하기 위해서는 아래와 같은 순서를 따른다.

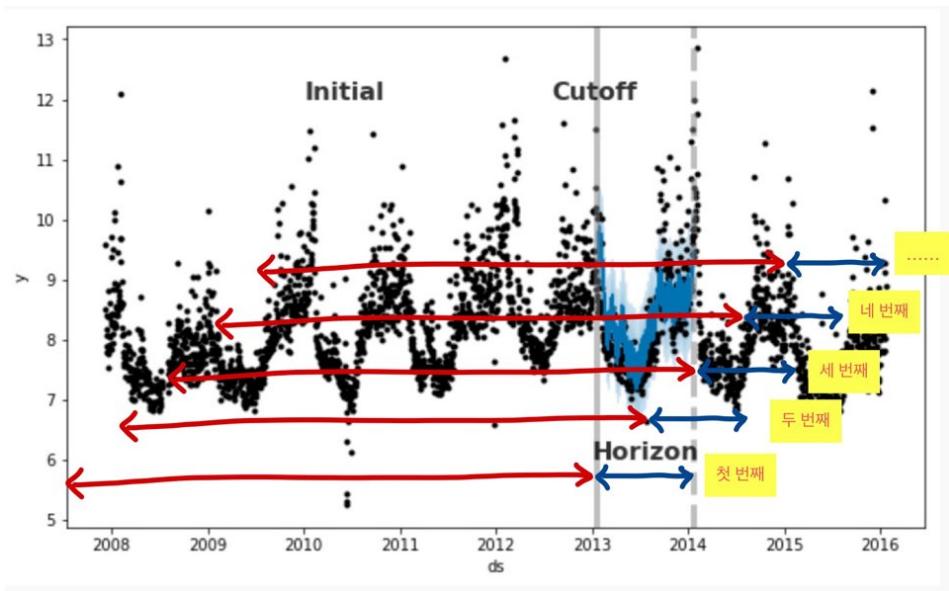
- 1) 데이터를 training set과 test set으로 나눈다.
- 2) 모델을 테스트 한 후 테스트 성능을 기록한다.
- 3) 교차 검증의 매 단계마다 다른 파티션으로 위의 작업을 수행
- 4) 매 단계의 테스트 성능을 평균 계산을 통해 최종 성능 도출

fbprophet 라이브러리에서는 사용자의 편의를 위해 이러한 교차 검증의 과정을 자동으로 실행해 주는 함수(`cross_validation`)를 제공한다.

이를 위해서 우리는 과거의 데이터(우리가 컴퓨터에게 준 데이터)에서 컷오프 포인트를 선택하고, 각각의 컷오프 포인트에 대해서만 데이터를 사용하여 모델을 학습시키는 방법으로 수행된다. 그런 다음 예측 값을 실제 값과 비교한다.

이 교차 검증 절차는 `cross_validation` 함수를 사용하여 다양한 과거 cutoff 지점에 대해 자동으로 수행될 수 있다. `cross_validation` 함수의 결과값은 각각 시뮬레이션된 예측 날짜와 컷오프 날짜에 대해 실제 값 y와 샘플 외 예측 값 yhat를 갖는 데이터 프레임이다.

아래는 설명을 돋기 위한 그래프 예시이다.



←→ : training data(initial) ←→ : test data(horizon)
cutoff point는 initial 과 horizon을 구분해 주는 날짜

특히 예측은 컷오프와 컷오프+지평선 사이(파란색으로 칠해진 지점)에 관측된 모든 지점에서 이루어진다. 이제 이 dataframe은 yhat과 y사이의 error를 계산하는 데 사용된다.

앞서 설명한 바를 토대로 한 `cross_validation` 함수를 이용한 교차검증의 과정이다. (단, 'initial'을 설정할 때는 계절성, 및 주기를 모두 포함 할 수 있을 정도로 충분히 길어야 한다.) 우리는 `initial`을 60주로 주었고, 이를 통해 예측할 미래는 20주로 주었다.

```
from fbprophet.diagnostics import cross_validation
df_cv = cross_validation(model, initial='60 w', period='10 w', horizon = '20 w')
df_cv[['ds','yhat','y','cutoff']].head()
```

	ds	yhat	y	cutoff
0	2017-02-01	1868.762677	1842.4	2017-01-18
1	2017-02-02	1859.142824	1843.8	2017-01-18
2	2017-02-03	1855.886044	1841.0	2017-01-18
3	2017-02-04	1851.709906	1842.8	2017-01-18
4	2017-03-01	1909.371344	1842.9	2017-01-18

`performance_metrics` 패키지를 사용하여 성능 예측에 대한 유용한 통계(mse, rmse, mae, mape, mdape, coverage)를 계산할 수 있다.

```
from fbprophet.diagnostics import performance_metrics
df_p = performance_metrics(df_cv)
df_p.head()
```

	horizon	mse	rmse	mae	mape	mdape	coverage
0	15 days	1509.164134	38.847962	23.103932	0.013143	0.006677	0.558824
1	16 days	1983.336647	44.534668	26.551584	0.015149	0.007291	0.519608
2	17 days	2395.451473	48.943350	28.409288	0.016252	0.006677	0.529412
3	18 days	2269.349384	47.637689	26.390397	0.015070	0.005173	0.588235
4	19 days	2240.963012	47.338811	25.747870	0.014701	0.004888	0.617647



MSE(mean squared error) 평균제곱오차



RMSE(root mean squared error) 평균 제곱근 오차



MAE(mean absolute error) 평균 절대 오차



MAPE(mean absolute percent error) 평균 절대 비율 오차

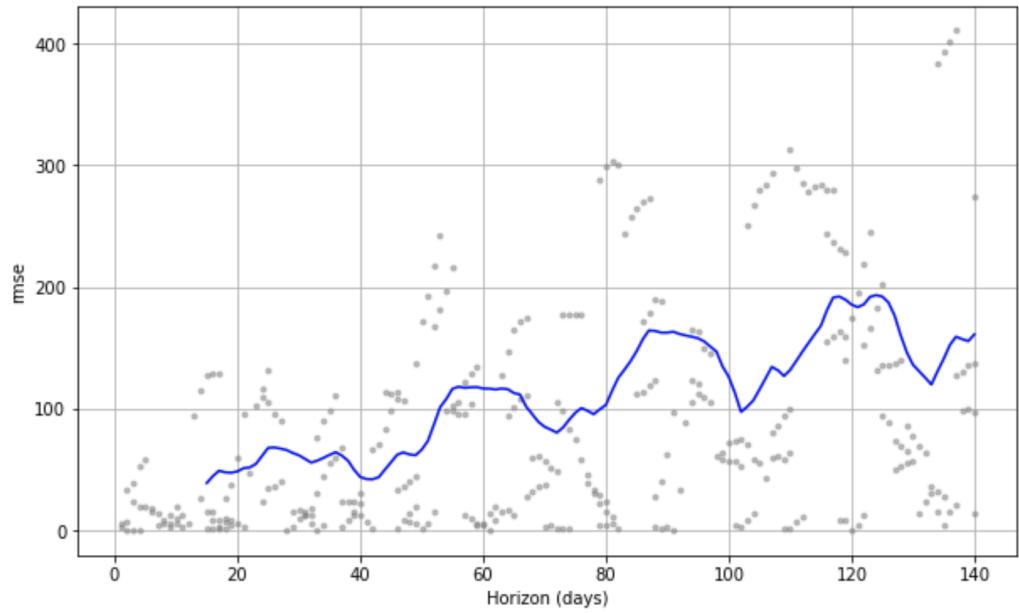


MDAPE(median absolute percent error) 중앙 절대 비율 오차

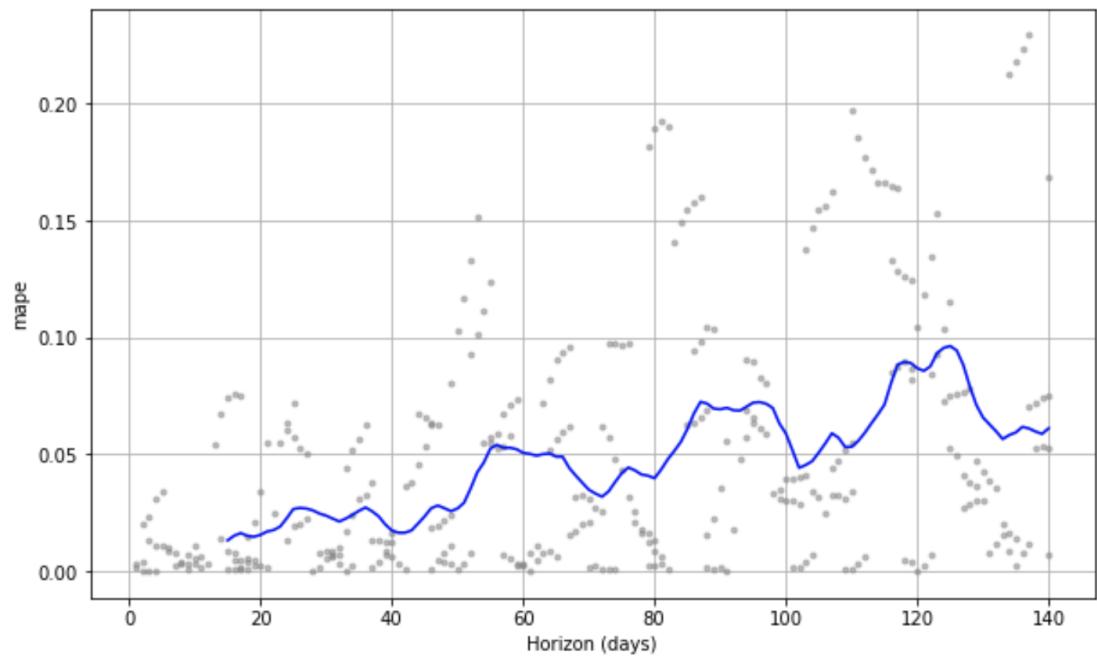
'`plot_cross_validation_metric`'은 앞서 '`performance_metrics`'를 통해 도출해 낸 여러 유용한 통계들(MSE, RMSE, MAE, MAPE, MDAPE)에 대해 시각화할 수 있다.

다음은 RMSE(평균 제곱근 오차)와 MAPE(평균 절대 비율 오차)를 그래프로 나타낸 것이다.

```
from fbprophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='rmse')
```



RMSE - 평균 제곱근 오차



MAPE - 평균 절대 비율 오차

3-2. Methodology : NUGU Backend proxy server with Flask

1. nuguoil 서버 전체 코드

python flask를 이용해서 REST API를 구현하여 nugu speaker로부터 받은 post 요청을 처리하는 코드이다.
크게 **Function part**와 **Flask part**로 나눌 수 있다.

```
from flask import Flask, request, jsonify
from flask_restful import Resource, Api
import json
import os
import requests
from pyproj import Proj
from pyproj import transform
import datetime
import time
import pandas as pd
from pandas import DataFrame
import telepot
from apscheduler.schedulers.background import BackgroundScheduler

#Function part
def location(): #find users location
    url = 'https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyDkx6muQn1Jz-y6hL0cTPVdYAhk1m6WJQo'
    data = {
        'considerIp': True,
        #'homeMobileCountryCode': 450,
        #'homeMobileNetworkCode': 5,
        #'radioType':'gsm',
        #'carrier': "SKTelecom",
        #'wifiAccessPoints": [{"macAddress': '40:DC:9D:06:EC:CA'}]
    }
    result = requests.post(url, data)
    a=result.json()
    lat=a['location']['lat'] # Y point
    lng=a['location']['lng'] # X point
    return lat,lng

def trans(lat,lng): #wgs84 -> tm128
    WGS84 = { 'proj':'latlong', 'datum':'WGS84', 'ellps':'WGS84', }

    TM128 = { 'proj':'tmerc', 'lat_0':'38N', 'lon_0':'128E', 'ellps':'bessel',
    'x_0': '400000', 'y_0': '600000', 'k': '0.9999',
    'towgs84': '-146.43,507.89,681.46' }

    def wgs84_to_tm128(longitude, latitude):
        return transform( Proj(**WGS84), Proj(**TM128), longitude, latitude )

    x_point,y_point=wgs84_to_tm128(lng,lat)
    return x_point,y_point

def ask_oil_type(ans):
    if ans == "2번" : #경유
        return "D047"
    elif ans == "1번" : #휘발유
        return "B027"
    else :
        return None

def browse(x_point,y_point,oil_type):
    url = 'http://www.opinet.co.kr/api/aroundAll.do'
    payload = {
        "code" : "F886201116",
        "out" : "json",
        "x" : x_point,
        "y" : y_point,
        "radius" : "1000",
        "prodcd" : oil_type ,
        "sort" : "1"
    }
    result = requests.get(url,params=payload).json()
    return result

def content():
    global data_num
```

```

global oil_list
global title
global cost
data_num = len(oil_list["RESULT"]["OIL"]) #주유소 개수 확인

if (data_num == 1):
    for i in range(0, 1):
        title.append(oil_list["RESULT"]["OIL"][i]['OS_NM'])
elif (data_num >= 3):
    for i in range(0, 3):
        title.append(oil_list["RESULT"]["OIL"][i]['OS_NM'])
elif (data_num == 2):
    for i in range(0,2):
        title.append(oil_list["RESULT"]["OIL"][i]['OS_NM'])

content_num = data_num

if content_num == 0: #검색된 주유소가 0개인 경우
    title.append("null")
    cost.append("null")
elif content_num > 3: #검색된 주유소 3개 초과일 경우 차례대로 3개만 처리
    for i in range(0, 3):
        cost.append(oil_list['RESULT']['OIL'][i]['PRICE'])

else: #검색된 주유소가 2개 혹은 3개인 경우
    for i in range(content_num):
        cost.append(oil_list['RESULT']['OIL'][i]['PRICE'])
return title, cost

def action(c):
    global data_num
    keys=['number','title','cost']
    arr=[]
    if data_num == 0:
        return arr

    elif data_num == 1:
        values = ["1", c[0], c[1]]
        A = dict(zip(keys, values))
        arr.append(A)
        return arr

    elif data_num == 2:
        values_1 = ["1", c[0][0], c[1][0]]
        values_2 = ["2", c[0][1], c[1][1]]
        A = dict(zip(keys, values_1))
        B = dict(zip(keys, values_2))
        arr.append(A)
        arr.append(B)
        return arr

    else:
        values_1 = ["1", c[0][0], c[1][0]]
        values_2 = ["2", c[0][1], c[1][1]]
        values_3 = ["3", c[0][2], c[1][2]]
        A = dict(zip(keys, values_1))
        B = dict(zip(keys, values_2))
        C = dict(zip(keys, values_3))
        arr.append(A)
        arr.append(B)
        arr.append(C)
        return arr

def make_response(result_list):
    global select
    global oil_type

    response = {
        "version": "2.0",
        "resultCode": "OK",
        "output": {
            "COUNT": "0",
            "STATION_INFORMATION": "",
        }
    }

    result_len = len(result_list)
    temp = ""
    if result_len == 0:
        return response

    elif result_len>0:
        response["output"]["COUNT"] = str(result_len)

        if result_len == 1:
            temp = str(result_list[0]["title"]) + "," + str(result_list[0]["cost"]) + "원"
            temp = temp.replace('[','')
            temp = temp.replace(']','')


```

```

temp = temp.replace("'", '')
response["output"]["STATION_INFORMATION"] = temp
else:
    for i in range(result_len):
        temp = temp + str(result_list[i]["title"]) + "," + str(result_list[i]["cost"]) + "원" + ","
        temp = temp.replace("]", "")
        temp = temp.replace("[", "")
        temp = temp.replace("'", "")
    response["output"]["STATION_INFORMATION"] = temp

return response

#flask part
app = Flask(__name__)
api = Api(app)

class Getparams(Resource):
    def post(self):
        data = request.get_json()
        print(data)
        global select
        global oil_type

        ans = ""
        if 'SELECT' in data['action']['parameters'].keys():
            select = data['action']['parameters']['SELECT']['value']
            if select == "1번" or select == "2번":
                ans = select
        if 'OIL_TYPE' in data['action']['parameters'].keys():
            oil_type = data['action']['parameters']['OIL_TYPE']['value']
            if oil_type == "경유":
                ans = "2번"
            elif oil_type == "휘발유":
                ans = "1번"

        a,b = location()
        a,b = trans(a,b)

        global oil_list
        oil_list = browse(a,b,ask_oil_type(ans))
        print(oil_list)

        global data_num
        global title
        global cost

        title = []
        cost = []
        result = action(content())
        response = make_response(result)

        if 'SELECT' in data['action']['parameters'].keys():
            response["output"]["SELECT"] = select
        if 'OIL_TYPE' in data['action']['parameters'].keys():
            response["output"]["OIL_TYPE"] = oil_type
        print(response)

        return jsonify(response)

api.add_resource(Getparams,'/answer.lowprice','/answer.lowprice.diesel','/answer.lowprice.gasoline','/answer.lowprice.diesel.0'

if __name__ == "__main__":
    app.run()

```

2. nuguoil 서버 Function part

a. Geolocation API로 현재 위치 찾기

`url`에는 필요한 Google Geolocation API url과 api key를 입력해주고, `data`에는 어떤 정보를 기반으로 위치추적을 요청할지에 대한 코드가 들어간다. nuguoil은 ip를 기반으로 사용자 위치를 추적하기 때문에 `'considerIp': True`를 넣어주었다.

`result 변수`에 geolocation api에게서 얻은 정보들이 들어오고, 그것을 json형식으로 변환하여 `a 변수`에 넣어준다. 여기서 우리가 필요한 값은 `a['location']['lat']` 와 `a['location']['lng']` 이고 각각 y좌표, x좌표로 대응되는 값이다. 이 값들을 각각 `lat변수`와 `lng변수`에 넣어주고 `return lat,lng` 해준다.

```

def location(): #find users location
    url = 'https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyDkx6muQn1Jz-y6hL0cTPVdYAhk1m6WJQo'
    data = {
        'considerIp': True,
        #'homeMobileCountryCode': 450,
        #'homeMobileNetworkCode': 5,
        #'radioType':'gsm',
        #'carrier': "SKTelecom",
        #'wifiAccessPoints':[{'macAddress':'40:DC:9D:06:EC:CA'}]
    }
    result = requests.post(url, data)
    a=result.json()
    lat=a['location']['lat'] # Y point
    lng=a['location']['lng'] # X point
    return lat,lng

```



하지만 현재 playbuilder 채팅 테스트시에 사용자 위치 좌표값이 제대로 나오지 않는 문제가 발생했다. 아마 사용자의 ip가 아닌 nugu playbuilder 서버와 관련된 ip를 기반으로 추적을 하는 등의 문제로 추측중이다. 일단 스피커 테스트를 통해 다시 확인해보기로 했다. data 딕셔너리 안의 주석들은 ip를 기반으로 위치를 추적하는데 실패했을 때를 대비해서 wifi나 기지국 정보를 이용할 수 있게 하는 코드들이다. 일단 스피커테스트에서 ip 기반 추적이 원활하게 실행 되는지 확인 후에 테스트 할 예정이다.

→ 관련 답변을 받았고 큰 문제가 생겼다. Conclusion에서 다루겠다..

b. WGS84 형식에서 오피넷의 TM128 형식으로 변환하기

우리는 좌표 변환에 관련된 함수를 포함하고 있는 Pyproj 패키지를 이용했다. parameter로 받은 WGS84형식의 lat,lng 값을 TM128 형식으로 변환하여 `x_point`, `y_point` 변수에 담아 반환한 것이다.

```

def trans(lat,lng): #wgs84 -> tm128
    WGS84 = { 'proj':'latlong', 'datum':'WGS84', 'ellps':'WGS84', }

    TM128 = { 'proj':'tmerc', 'lat_0':'38N', 'lon_0':'128E', 'ellps':'bessel',
    'x_0':'400000', 'y_0':'600000', 'k':'0.9999',
    'towgs84':'-146.43,507.89,681.46' }

    def wgs84_to_tm128(longitude, latitude):
        return transform( Proj(**WGS84), Proj(**TM128), longitude, latitude )

    x_point,y_point=wgs84_to_tm128(lng,lat)
    return x_point,y_point

```

c. 유기정보 API용 "prodcd" 코드로 변환

해당 함수는 nugu speaker에게서 post요청으로 받아온 정보를 인자로 받아서 오피넷 api를 사용 할 때 입력해야하는 "prodcd" 코드로 변환해주는 역할을 수행한다.

```

def ask_oil_type(ans):
    if ans == "2번" : #경유
        return "D047"
    elif ans == "1번" : #휘발유
        return "B027"
    else :
        return None

```

d. 반경 5km내에 있는 주유소, 최저가 순으로 정렬

이 함수는 오피넷 api를 이용하여 `x_point` 와 `y_point`, `oil_type` 을 인자로 받고 다시 request 인자로 넣어 해당하는 위치에서 반경 5km내에 있는 가장 저렴한 주유소 순으로 정렬한 주유소 정보를 가져온다.

```

def browse(x_point,y_point,oil_type):
    url = 'http://www.opinet.co.kr/api/aroundAll.do'
    payload = {
        "code" : "F886201116",
        "out" : "json",
        "x" : x_point,
        "y" : y_point,
        "radius" : "5000",
        "prodcd" : oil_type ,
        "sort" : "1"
    }
    result = requests.get(url,params=payload).json()
    return result

```

e. 검색된 주유소 개수를 확인하고, 최대 3개의 결과값을 제공

해당 함수는 앞에서 **brows**함수의 리턴값으로 나온 주유소 리스트가 저장 되어 있는 global 타입의 `oil_list`에 몇개의 주유소가 있는지 카운트해서 global타입의 `data_num`변수에 넣어준 후, 최대 3개의 주유소 이름과 가격을 각각 global 타입의 `title`, `cost`에 넣어 준다.

```

def content():
    global data_num
    global oil_list
    global title
    global cost
    data_num = len(oil_list["RESULT"]["OIL"]) #주유소 개수 확인

    if (data_num == 1):
        for i in range(0, 1):
            title.append(oil_list["RESULT"]["OIL"][i]['OS_NM'])
    elif (data_num >= 3):
        for i in range(0, 3):
            title.append(oil_list["RESULT"]["OIL"][i]['OS_NM'])
    elif (data_num == 2):
        for i in range(0,2):
            title.append(oil_list["RESULT"]["OIL"][i]['OS_NM'])

    content_num = data_num

    if content_num == 0: #검색된 주유소가 0개인 경우
        title.append("null")
        cost.append("null")
    elif content_num > 3: #검색된 주유소 3개 초과일 경우 차례대로 3개만 처리
        for i in range(0, 3):
            cost.append(oil_list['RESULT']['OIL'][i]['PRICE'])
    else: #검색된 주유소가 2개 혹은 3개인 경우
        for i in range(content_num):
            cost.append(oil_list['RESULT']['OIL'][i]['PRICE'])
    return title, cost

```

f. 결과 값을 딕셔너리 리스트에 넣기

action(c) 함수는 앞에서 **content**함수의 리턴인 `title`, `cost` 리스트를 c라는 인자로 받아서 `keys=['number','title','cost']`를 키 값으로 하는 딕셔너리에 c값을 넣어서 리턴하는 함수이다.

```

def action(c):
    global data_num
    keys=['number','title','cost']
    arr=[]
    if data_num == 0:
        return arr

    elif data_num == 1:
        values = ["1", c[0], c[1]]
        A = dict(zip(keys, values))
        arr.append(A)
        return arr

    elif data_num == 2:
        values_1 = ["1", c[0][0], c[1][0]]
        values_2 = ["2", c[0][1], c[1][1]]
        A = dict(zip(keys, values_1))
        arr.append(A)
        return arr

```

```

B = dict(zip(keys, values_2))
arr.append(A)
arr.append(B)
return arr

else:
    values_1 = ["1", c[0][0], c[1][0]]
    values_2 = ["2", c[0][1], c[1][1]]
    values_3 = ["3", c[0][2], c[1][2]]
    A = dict(zip(keys, values_1))
    B = dict(zip(keys, values_2))
    C = dict(zip(keys, values_3))
    arr.append(A)
    arr.append(B)
    arr.append(C)
return arr

```

G. NUGU PlayBuilder의 답변 형식에 맞게 반환

make_response(result_list) 함수는 **action** 함수의 리턴값으로 나온 딕셔너리 리스트를 받아서, nugu speaker에서 post 요청을 받은 후 nugu speaker에서 필요로 하는 response block 데이터 형식에 맞춘 딕셔너리를 반환하는 함수이다. `response["output"]`

`["COUNT"]`에는 파싱한 주유소의 총 개수가 들어가며, `response["output"]["STATION_INFORMATION"]`에는 파싱한 주유소의 정보가 들어간다. "~주유소, ~원, ~주유소, ~원"과 같이 최대 3개의 주유소의 정보가 나열된 형태의 문자열의 형식을 취한다.



"~주유소, ~원, ~주유소, ~원"처럼 나열된 형식을 이용하는 이유는 추후에 nugu speaker에서 주유소 정보에 대한 발화를 할 때, 간단한 방식으로 끊김 없이 정보를 말할 수 있도록 하기 위함이다.

```

def make_response(result_list):
    global select
    global oil_type

    response = {
        "version": "2.0",
        "resultCode": "OK",
        "output": {
            "COUNT": "0",
            "STATION_INFORMATION": ""
        }
    }
    result_len = len(result_list)
    temp = ""
    if result_len == 0:
        return response

    elif result_len > 0:
        response["output"]["COUNT"] = str(result_len)

        if result_len == 1:
            temp = str(result_list[0]["title"]) + "," + str(result_list[0]["cost"]) + "원"
            temp = temp.replace('[', '')
            temp = temp.replace(']', '')
            temp = temp.replace("'", '')
            response["output"]["STATION_INFORMATION"] = temp
        else:
            for i in range(result_len):
                temp = temp + str(result_list[i]["title"]) + "," + str(result_list[i]["cost"]) + "원" + ","
                temp = temp.replace("[", "")
                temp = temp.replace("]", "")
                temp = temp.replace("'", "")
            response["output"]["STATION_INFORMATION"] = temp

    return response

```

2. nuguoil 서버 Flask part

먼저 `data = request.get_json()`를 통해 받아온 request body에는 서비스 사용자가 어떤 종류의 기름을 기준으로 최저가 주유소를 검색하고 싶어하는지와 관련된 정보가 담겨있다. 해당 정보는 `SELECT` 또는 `OIL_TYPE`이라는 이름의 parameter에 담겨있으며 nugu playbuilder에서 설정한 action에 따라서 둘 중 하나의 parameter만 request body로 전달된다. 해당 key값이 있는지 if문의 조건을 통해 확인한 후 `ans변수`에 value값을 넣어준다. browse함수에 oiltype를 인자로 넣을 때, 관련된 코드로 변환해주는 함수인

ask_oil_type(ans) 함수를 이용하기 위해서는 ans변수에 들어가는 값을 "1번", 혹은 "2번"으로 변환을 해주어야 한다. 따라서 `data['action']['parameters']['OIL_TYPE']['value']`의 값이 "휘발유"라면 "1번", "경유"라면 "2번"으로 변환해준다.

다음으로 **location()** 함수로 구한 위치 정보를 `a, b`변수에 초기화한 후, **trans(a,b)** 함수로 WGS84형식의 a,b 값을 TM128형식으로 변환한다.

browse(a,b,ask_oil_type(ans)) 함수로 구한 주유소 리스트를 `oil_list`에 초기화한 후 **action(content())** 함수로 `oil_list`의 값을 필요한 형식으로 변환해준다.

make_response(result)함수로 response body형식에 맞추어 정보를 가공한 후 select로 들어온 값 혹은 oiltype으로 들어온 값을 response body에 추가해준다.



request body로 받은 parameter값을 response body에도 입력해주어야 한다.

```
app = Flask(__name__)
api = Api(app)

class Getparams(Resource):
    def post(self):
        data = request.get_json()
        print(data)
        global select
        global oil_type

        ans = ""
        if 'SELECT' in data['action']['parameters'].keys():
            select = data['action']['parameters']['SELECT']['value']
            if select == "1번" or select == "2번":
                ans = select
        if 'OIL_TYPE' in data['action']['parameters'].keys():
            oil_type = data['action']['parameters']['OIL_TYPE']['value']
            if oil_type == "경유":
                ans = "2번"
            elif oil_type == "휘발유":
                ans = "1번"

        a,b = location()
        a,b = trans(a,b)

        global oil_list
        oil_list = browse(a,b,ask_oil_type(ans))
        print(oil_list)

        global data_num
        global title
        global cost

        title = []
        cost = []
        result = action(content())
        response = make_response(result)

        if 'SELECT' in data['action']['parameters'].keys():
            response["output"]["SELECT"] = select
        if 'OIL_TYPE' in data['action']['parameters'].keys():
            response["output"]["OIL_TYPE"] = oil_type
        print(response)

        return jsonify(response)

api.add_resource(Getparams, '/answer.lowprice', '/answer.lowprice.diesel', '/answer.lowprice.gasoline', '/answer.lowprice.diesel.0'

if __name__ == "__main__":
    app.run()
```

3-3. Methodology : NUGU Play Builder 구조

1. User Utterance Model 만들기

Intent란 사용자의 의도를 말하며, 이 의도를 처리하는 기능은 Action이 한다. 예를 들어 날씨를 묻고자 할때 '아리야, 날씨 알려줘'는 Intent고, '오늘은 맑습니다'같은 답변이 Action이다. NUGU 오일에서 정의한 Intent는 다음과 같다

NUGU 오일에서 정의한 Intent



`ask.lowprice` : 이 intent는 사용자가 발화한 Utterance Parameter인 오일 종류(경유/휘발유) entity를 받아 Backend Proxy 서버에 전송하고, 그 결과로 얻은 주유소 정보와 가격 Backend Proxy를 받아온다.



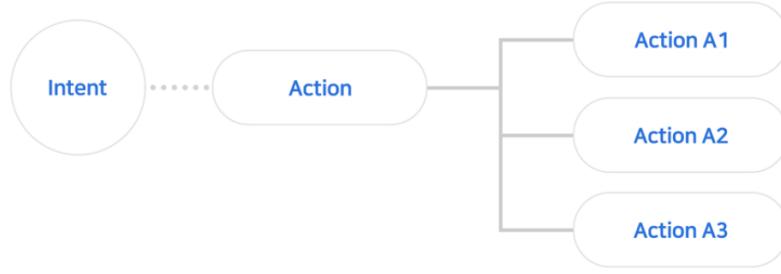
`ask.price` : 이 intent는 경유, 휘발유 예측 가격을 Backend Proxy 서버에서 받아온다



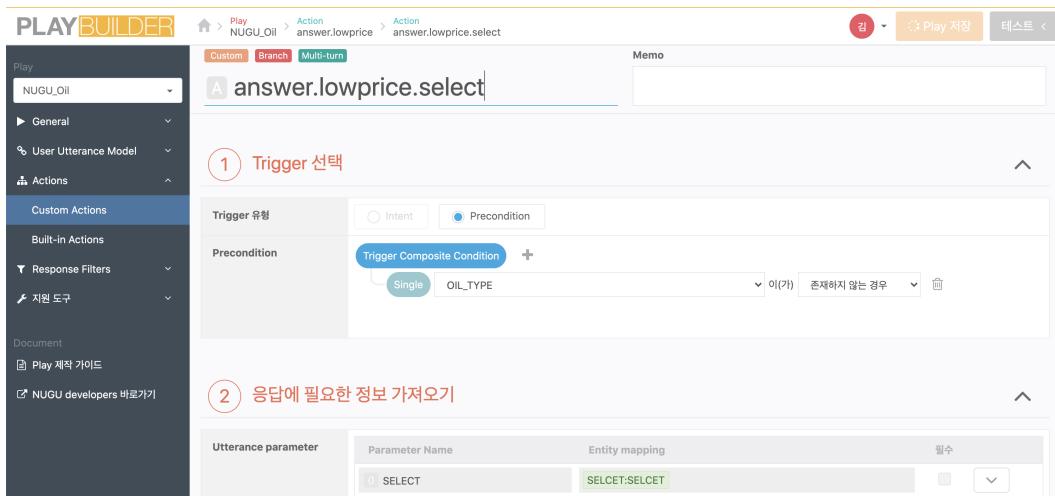
`ask.select` : 사용자가 물어보는 게 경유인지 휘발유인지 알아보기 위해 다시 물어볼 때 쓰는 Intent

2. Action 만들기

Action은 Intent들이 실제로 NLU 엔진을 통해 분석되었을 때 이를 처리하는 것을 의미한다. '날씨 알려줘'에서 살펴보았듯이 Intent를 처리하기 위해 하나의 Action을 만들고, 우리는 이 Action에 응답을 작성할 수 있다.

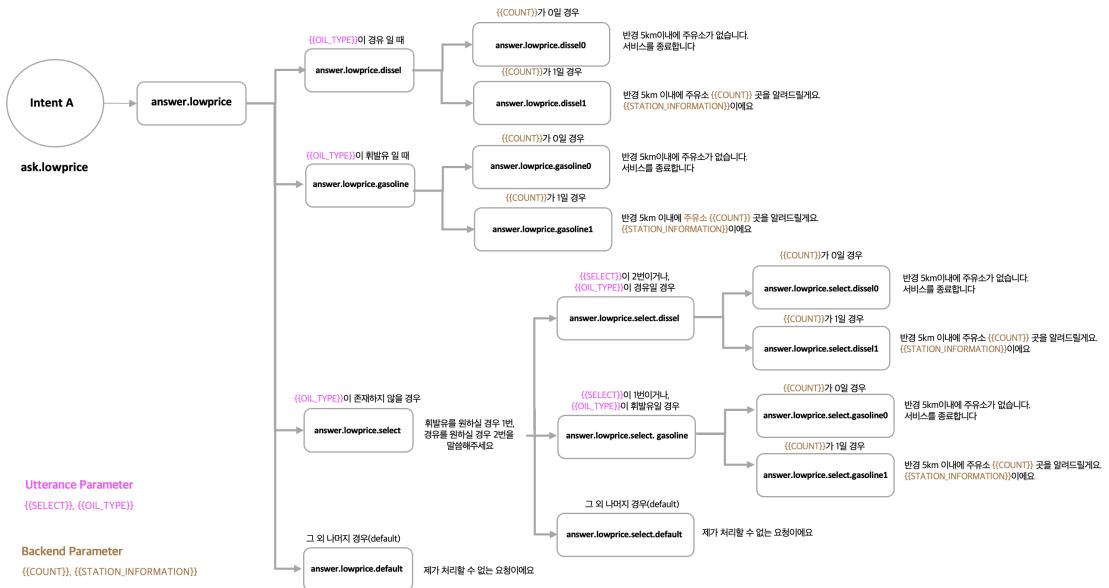


Action 정의에서는 Output 유형 4가지를 우리의 의도에 맞게 배치 해야했다. NUGU Play Builder에서는 구분을 `Response`, `Branch Action`, `Response + Branch Action`, `Common Action`으로 하고 있는데, 우리가 자주 사용했던 것은 `Branch Action`으로 "경유 최저가 주유소 알려줘"라고 물으면 Backend Proxy에서 값을 받아왔을 때는 해당 정보를, Backend Proxy에서 값이 없을 때는 없다는 메세지를 내보내는 것이었다. 또한 경유인지, 휘발유인지 사용자의 의도를 한 번 더 물어볼 때는 `Response + Branch Action`을 사용했는데, 아래 사진처럼 Trigger을 명확하게 지정해줘야 했다.



3. Action Tree

우리가 작성한 NUGU Play Builder의 발화문 처리는 다음과 같다. 사용자가 경유인지 휘발유인지를 언급하면, 바로 Backend Proxy에서 가져온 주유소 명칭과 가격을 불러올 수 있게 연결했고, 사용자가 경유인지 휘발유인지를 미리 언급하지 않았다면, 한 번 더 물어보는 구조로 작성했다.



4. 외부 연동 서버 (Backend proxy) 설정

Intent와 Action을 모두 설정했다면, 이제 Back단의 데이터와 연동시킬 차례이다. 연결은 NUGU Play Builder의 General 메뉴에서 Global 설정으로 진행할 수 있다.

Backend Parameter를 위에서 기획한대로 기입하고, 서버와의 연결을 확인한다.

The screenshot shows the 'Global' settings page in NUGU Play Builder. It includes sections for Backend Parameter, Configurable Parameter, and 예외 상황 관리 (예외 상황). The Backend Parameter section lists COUNT and STATION_INFORMATION. The Configurable Parameter section has a placeholder 'Parameter 이름을 입력하세요.' The 예외 상황 관리 section also has a placeholder 'Exception code'.

연결을 마치면 상단에 테스트 버튼을 통해 테스트란을 진행할 수 있다. 아래 NLU 분석 결과 열기 버튼을 통해 NUGU Play Builder가 제대로 intent를 인식 했는지 확인했고, 결과

값이 정확한지 확인했다.

User

경유 최저가 주유소 알려줘

NLU 분석 결과 열기

NUGU

반경 5km 이내에 주유소 3 곳을 알려드릴게
요. 현대오일뱅크(주)직영 토픽1호셀프주유
소, 1387원, 푸른주유소, 1399원, 현대오일뱅
크(주)직영 토픽2호주유소, 1427원, 이에요



4. Evaluation & Analysis : 예측 정확도 분석

우리 팀은 유가 정보의 예측을 위해 Prophet 라이브러리를 사용한 바 있다. 아래 사진은 다음은 우리가 오피넷에서 받아온 **2015년 11월 첫째주부터 2020년 6월 첫째 주까지의** 유가 데이터이다.

	ds	y
0	2015-11-01	1835.4
1	2015-11-02	1835.4
2	2015-11-03	1824.1
3	2015-11-04	1825.5
4	2015-12-01	1818.5
...
235	2020-05-01	1585.8
236	2020-05-02	1577.0
237	2020-05-03	1574.9
238	2020-05-04	1580.6
239	2020-06-01	1588.5

우리가 비교할 구간은 2020년 6월 둘째주부터 2020년 7월 첫째 주로, 왼쪽에는 우리가 도출해낸 예측 결과이며 오른쪽은 실제 가격이다.

	ds	yhat	Actual value
240	2020-06-07	1737.888957	
241	2020-06-14	1754.893891	
242	2020-06-21	1754.476336	
243	2020-06-28	1733.447534	
예측된 유가			
	2020-06-02		1600.5
	2020-06-03		1614
	2020-06-04		1621.6
	2020-07-01		1628.1
실제 유가			

엑셀을 이용하여 실제 유가와 예측한 유가의 오차율을 구하였다.



$$\text{오차율} = (|\text{실제 값} - \text{측정값}|) / (\text{실제 값}) * 100$$

	A	B	C	D	E	F	G
1	actual value	predicted value	measuring efficiency				
2	1600.5	1737.888957	8.584127255				
3	1614	1754.893891	8.729485163				
4	1621.6	1754.476336	8.194149948				
5	1628.1	1733.447534	6.470581265				
6							
7							
8							
9							
10							
11							

오차율 4개의 평균을 구한 결과 실제 유가와 예측한 유가와의 오차율은 다음과 같다.

7.994585908

5. Related Work

- 오피넷 - 유가 정보 api 및 국내 유가 동향.csv 제공

<http://www.opinet.co.kr/user/main/mainView.do>

- 오피넷 제공 무료 api 가이드

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1aba8e20-8dd0-455a-b448-6a34d402ab45/Opinet_API_Free.pdf

- 구글 클라우드 플랫폼 - geolocation api 제공

<https://console.cloud.google.com/home/dashboard?project=organic-diode-295813&hl=ko>

- geolocation 관련 참고자료

<https://velog.io/@yvvyoon/python-current-location-coordinate>

<https://developers.google.com/maps/documentation/geolocation/overview>

- 위도,경도 측정단위 변환 관련 참고자료

<https://gist.github.com/allieus/1180051/ab33229e820a5eb60f8c7971b8d1f1fc8f2cfabb>

- Python Flask - REST API , HEROKU 배포

<https://rekt77.tistory.com/103?category=825845>

<https://ebbnflow.tistory.com/220>

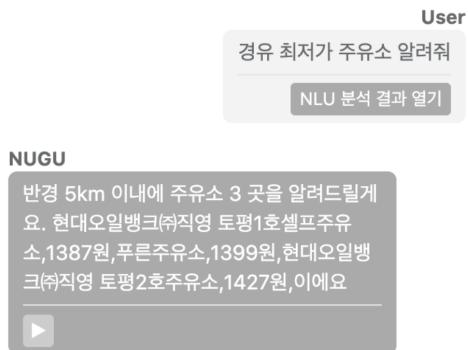
6. Conclusion

이렇게 반경 5km 내의 가장 저렴한 주유소를 찾아주는 nuguoil 서비스가 완성되었다. 사용자가 nuguoil을 실행하면 아리아가 사용자의 선택에 따라서 "반경 5km 내의 경유 최저가 주유소 정보" 또는 "반경 5km 내의 휘발유 최저가 주유소 정보"를 알려준다.

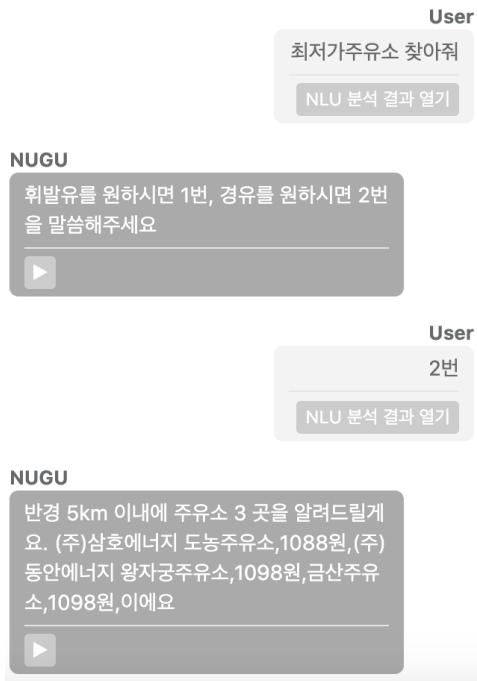
현재 주간 유가 동향을 예측해주는 기능은 완성도 문제로 nuguoil 서비스에 추가하지 않았다. 유가를 결정짓는 모든 요소를 파악해서 기계학습 모델을 만들기는 어려울 것으로 예상되므로, 가격 시계열 데이터 이외에 다른 요소를 혼합하여 가격을 예측할 수 있는 방안을 고민해 볼 예정이다.

1. 결과

nuguoil 시작/오픈 등의 발화로 nuguoil서비스를 시작한다



"경유 최저가 주유소 알려줘" 혹은 "휘발유 최저가 주유소 알려줘"라고 기름 종류와 함께 최저가 주유소를 찾아달라는 요청을 하면 바로 원하는 정보를 사용자가 얻을 수 있다.



"최저가 주유소 알려줘" 와 같이 기름 종류를 말하지 않고 주유소를 찾아달라고 요청하면 아리아가 어떤 기름 종류를 기준으로 주유소를 검색하고 싶은지 물어본다. 사용자가 1번을 말하면 휘발유 최저가 주유소, 2번을 말하면 경유 최저가 주유소를 알려준다.



위 테스트에서는 임의로 사용자 위치정보에 37.585876, 127.143135 좌표를 넣어서 실행했다.

2. 문제점

- 개인정보 처리관련 이슈 : 사용자의 위치정보 Backend proxy 연결

앞서 채팅 테스트를 진행할 때 geolocation api가 제대로 사용자 위치를 파악하지 못하는 문제가 있었다. 관련해서 sk에 문의 사항을 남겼고 아래와 같은 답변을 받았다.

2. 위치 정보 사용

사용자 위치 정보를 Backend proxy로 전송하는 것은 원칙적으로 불가합니다.

이는 악관상 개인정보 이슈로 인해 위치 정보를 외부 서버로 전송하는데 제한이 있기 때문입니다.

위치정보는 NUGU Platform과 제휴가 체결된 파트너사에 한하여

OAuth 인증 및 개인정보처리 방침 추가, 개인정보 수집 등의 절차 추가, SKT와의 개인정보 위탁처리 계약 등을 거쳐 권한을 열어드리고 있습니다.
양해부탁드립니다.

감사합니다.

사용자의 위치를 추적할 수 있는 권한을 얻을 수 없기 때문에, 사용자가 본인의 위치를 지정하지 않아도 간편하게 최저가 주유소를 찾을 수 있는 서비스를 제공하는 원래의 목적을 이룰 수 없게 되었다.

geolocation api를 사용해서 위치 추적을 하지 않고 사용자 발화로 사용자의 현재 위치를 파악하려면 도로명 주소를 받아와서 좌표로 변환하는 과정을 거쳐야 할 것으로 예상된다.

현재 프록시 서버에 들어가있는 코드에서 location()함수를 주석처리하고 a와 b에 임의의 좌표값을 넣어 놓았다.

- 유가 정보 예측 정확도 관련 이슈

주간 유가 정보로 시계열데이터 분석을 해서 최대 2주까지의 유가 가격 정보 동향을 예측하고 알려주는 서비스를 제공하고 싶었지만 오직 가격정보로만 유가를 예측하다보니 정확도가 떨어진다는 문제점을 발견 할 수 있었다.