어디서 만나서 뭐먹지?

약속장소, 맛집 추천 서비스

INDEX

01 주제 선정 배경

02 데이터 수집

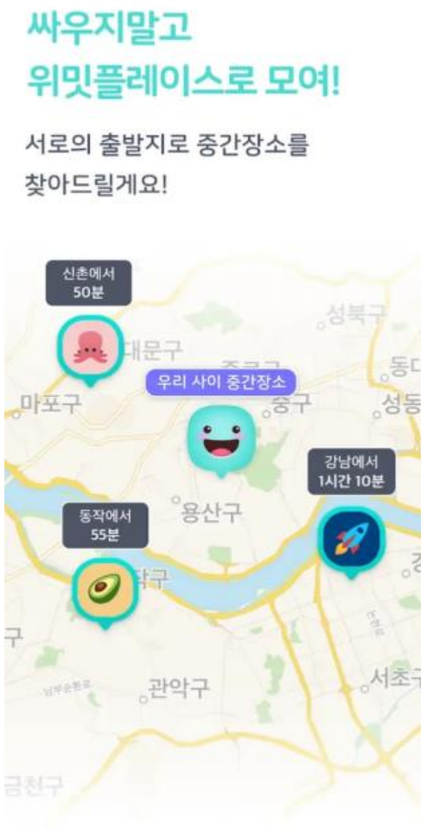
03 개발 과정

04 구동 예시

05 한계점 및 향후 개발방향

01 주제 선정 배경

중간 지점을 찾아주는 기능과 맛집을 추천하는 기능을 접목한
중간 지점 근처의 맛집 리스트와 리뷰, 평점까지 추천해주는 서비스



맛집 추천 서비스

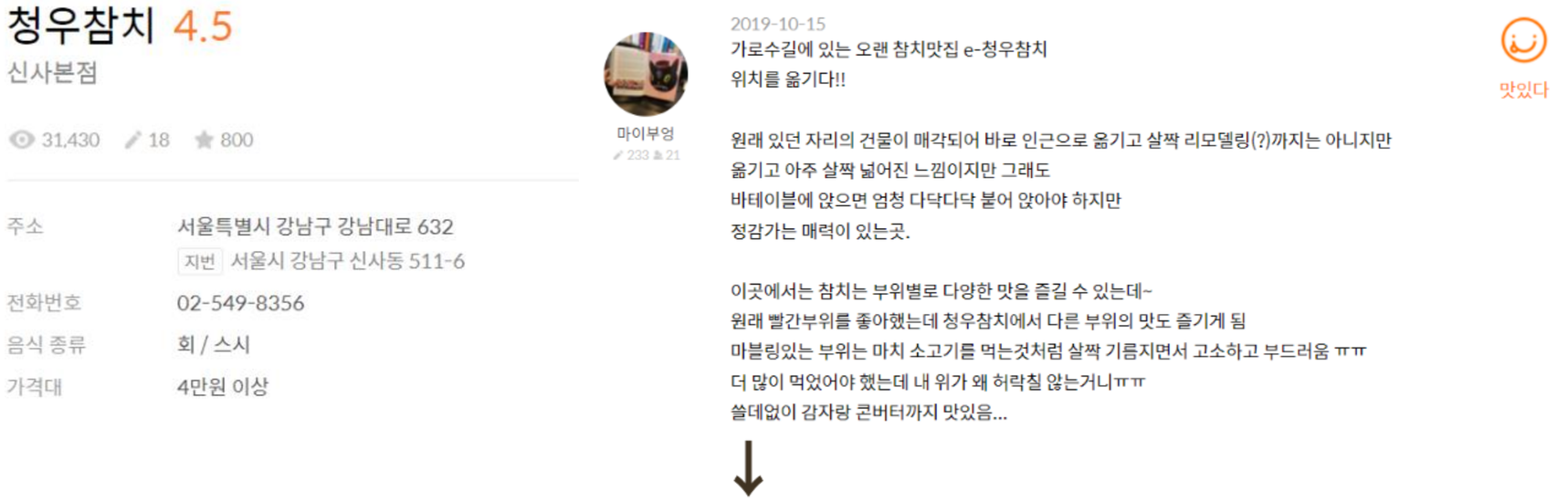
+

중간지점 산출 서비스

02 데이터 수집

1. 망고플레이트 가게 데이터

2. 카카오맵 가게 리뷰 데이터
- 평점 예측 모델 학습용



이름	평점	주소	주력 메	가격대	플레이트	리뷰내
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	가로수길에 있는 오랜 참치맛집 e-청우참치위치를 옮기다!!원래 있던
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	가성비 너무 좋고, 정말 양질의 참치를 양껏 즐길 수 있어서 더 좋다. 참
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	가성비 좋은 한국식 참치횟집. 선도가 꽤 좋다. 카운터석도 있고 바닥에
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	그저 박수만. 참치땡길때 여기만한 곳이 없다.
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	맛있다.. 느끼해서 많이 못먹는게 함정.
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	신사동에 위치한 "e-청우참치". 평이 항상 좋아서 꼭 가보고 싶었던 참
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	어제 저녁 두번째 방문..! 요근래 양도 줄고 술도 조금 줄어서 완전 박살
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	언제가 가봐야겠다하고명절 전날 일찍 끝나 가게된 청우참치메뉴는 단
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	오래전 저장만 해두고 언제쯤 갈수있을까 하다 드디어 갔는데어우 진짜
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	완벽한 해동균일한 혼마구로 처음가도 단골과 차이없이 나오는 참치무
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	정말 너무 맛있게 먹었던 참치집. 혈육이 참치사달라고 딱 이 가게를 골
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	참치만 드신다면 이 이상의 해동+퀄리티를 본적이 없습니다 최상위혼
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	참치의 최상급부위를 계속 먹을수있다!!
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	최고입니다
청우참치	4.5	서울특별시	회 / 스시	4만원 이상	https://ww	퀄리티 좋은 참치회를 무한으로 즐겨요10월 1일이 니혼슈의 날인 것
정돈	4.3	서울특별시	까스 요리	만원-2만원	https://ww	동그란 게 안심. 길쭉한 건 등심. 보이는 것보다 크기가 커서 다 먹으면
정돈	4.3	서울특별시	까스 요리	만원-2만원	https://ww	등심,안심 돈가스 둘다 먹었는데 겉바속촉인데속이 정말 부드러워서 맛
정돈	4.3	서울특별시	까스 요리	만원-2만원	https://ww	등심+안심 돈까스 17,000비싸지만 돈이 아깝지 않을 정도로 맛있음소
정돈	4.3	서울특별시	까스 요리	만원-2만원	https://ww	맛☆☆☆☆☆분위기☆☆☆☆☆혼밥하기 딱좋은 곳이다맛도
정돈	4.3	서울특별시	까스 요리	만원-2만원	https://ww	맛은 있다.양은 보통 적당히 배부름.안심인데 식감은 마치 치돈.다만 힝
정돈	4.3	서울특별시	까스 요리	만원-2만원	https://ww	사람이 늘 너무 많아서 먹기가 힘들지만예전부터 제가 즐겨찾던 맛집인

02 데이터 수집

1. 망고플레이트 가게 데이터

2. 카카오맵 가게 리뷰 데이터
   - 평점 예측 모델 학습용

score	review
4	생크림이 좀더 듬뿍 올라가고, 시트가 더 부드러우면 더 맛있을거같다 우유향이 나고 부드러운 생크림 신선하고 새콤달콤한 딸기 ㄷ
4	굿굿!
4	맛있어요
5	맛있어요
1	진짜 유명해서 한번 먹어봄 생딸기가 아니고 절임딸기 한조각 6300원 시트도 퍼석하고 크림은 그냥그럼 이게 맛있다는 분들은 생딸
4	soso
5	
3	가격창렬
3	너무 기대했나싶은... 그냥 그럼
4	대체적으로 맛있는데 와 엄~ 청 맛있다는 아닌 ..
1	음료 잔 크기 보고 눈을 의심했어요,,^^ 손보다 작은 잔에 얼음 가득 들어있는데 음료 진짜 100미리는 들어가나요,,? 친구들이랑 갔
2	왜 유명하지...느끼해서 많이 못 먹음
5	여기 딸기빙수가 미쳤어요 9000원인데 진짜... 저 딸기 절임? 한스푼 떠가지고 빙수랑 호로록하면 끝장나요 다른거 말고 딸기빙수 꼭
5	잊을만하면 생각나는 딸기 케이크. 희한하게 일년에 한두번 갈 때마다 가격이 계속 오르는걸 보며 나이듦이 실감난다...^^ 아쉬움 없
3	
5	두조각먹었는데 세조각먹을껄
5	딸기케잌은 무조건 여기!
3	작은 한조각에 6300원 값어치 까지는 없다 싶지만 매일매일 생크림 케익만 전문으로 만드는 집이라 신선하리라는 믿음 으로 가봄. ㅋ
1	딸기 케이크 맛있습니다 ! 4점 드립 커피 에티오피아 3점 케냐 1점 콜롬비아 1점 평균 2.25 참고로 총점은 10점

03 개발 과정

1. 리뷰 평점 예측 모델
    1) 리뷰 자연어 처리
    2) LSTM 모델 학습 및 평가

2. 약속장소와 맛집 추천
    1) 가게 위치 클러스터링
    2) 중간 지점 산출
    3) 약속 장소 추천
    4) LSTM 모델을 이용한 평점 예측
    5) 평점 상위5 맛집 추천
    6) 추천 알고리즘 생성

03 개발 과정

1. 리뷰 평점 예측 모델

1) 리뷰 자연어 처리 카카오 맵에서 수집한 리뷰 , 평점 데이터를 다음과 같은 과정을 거쳐 전처리 하였다.

데이터 정제

```
# clean_str 함수를 통해서 X데이터를 정제해주어야함.
train_test_X = [clean_str(sentence) for sentence in review_df["review"]]
```

```
def clean_str(string):
    string = re.sub(r"[^가-힣A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    string = re.sub(r"\'{2,}", "\'", string)
    string = re.sub(r"\'", "", string)

    return string.lower()
```

↓

띄어쓰기 단위로 분리

```
# 문장을 띄어쓰기 단위로 단어 분리
sentences = [sentence.split(' ') for sentence in train_test_X]
for i in range(5):
    print(sentences[i])
```

↓

토큰화, 패딩

```
# 총 3956974행의 데이터 중 30문장이내의 데이터는 3401248으로 85.9%에 육박.
# 따라서, padding의 기준을 30로 잡겠음.
sentence_new = []
for sentence in sentences:
    sentence_new.append([word[:5] for word in sentence][:30])

sentences = sentence_new

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=3000)
tokenizer.fit_on_texts(sentences)
train_X = tokenizer.texts_to_sequences(sentences)
train_X = pad_sequences(train_X, padding='post')
```

↓

```
[[2825 1266 1318    3   15   15  583  585 1560  708  843  271  357  360
  2207   84  563 1476 2222    0    0    0    0    0    0    0    0    0
     0    0]
 [ 373    2    1    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   5    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   5    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   7  888  170 1916  146 2210 2982  323  950  479    4   21  600  150
   321  150  315    1    0    0    0    0    0    0    0    0    0    0
     0    0]]
```

03 개발 과정

1. 리뷰 평점 예측 모델

2) LSTM 모델 학습 및 평가

LSTM을 이용해 모델을 생성하고 전처리한 데이터로 학습 및 평가하였다.

```
def lstm_model(vocab_size, embedding_dim, input_length):
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=input_length),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(units=64, return_sequences=True)),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(units=64)),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.Dense(6, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    print(model.summary())

    return model

check_path = './data/model_checkpoint/check_220108.ckpt'
checkpoint = ModelCheckpoint(filepath=check_path,save_best_only=True,
                             monitor='val_loss',verbose=1)

model = lstm_model(vocab_size = 30000, embedding_dim = 500, input_length = 30)

history = model.fit(train_X, train_y, epochs=15, batch_size=150, validation_split=0.3, callbacks=[checkpoint])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 30, 500)           15000000
_________________________________________________________________
bidirectional (Bidirectional (None, 30, 128)           289280
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               98816
_________________________________________________________________
dense (Dense)                (None, 32)                4128
_________________________________________________________________
dense_1 (Dense)              (None, 16)                528
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 102
=================================================================
Total params: 15,392,854
Trainable params: 15,392,854
Non-trainable params: 0
```

```
model.evaluate(test_X, test_y, verbose=0)
```

```
[0.12863808870315552, 0.9481304287910461]
```

→ **테스트 정확도 : 0.948**

03 개발 과정

2. 약속장소와 맛집 추천

1) 가게 위치 클러스터링

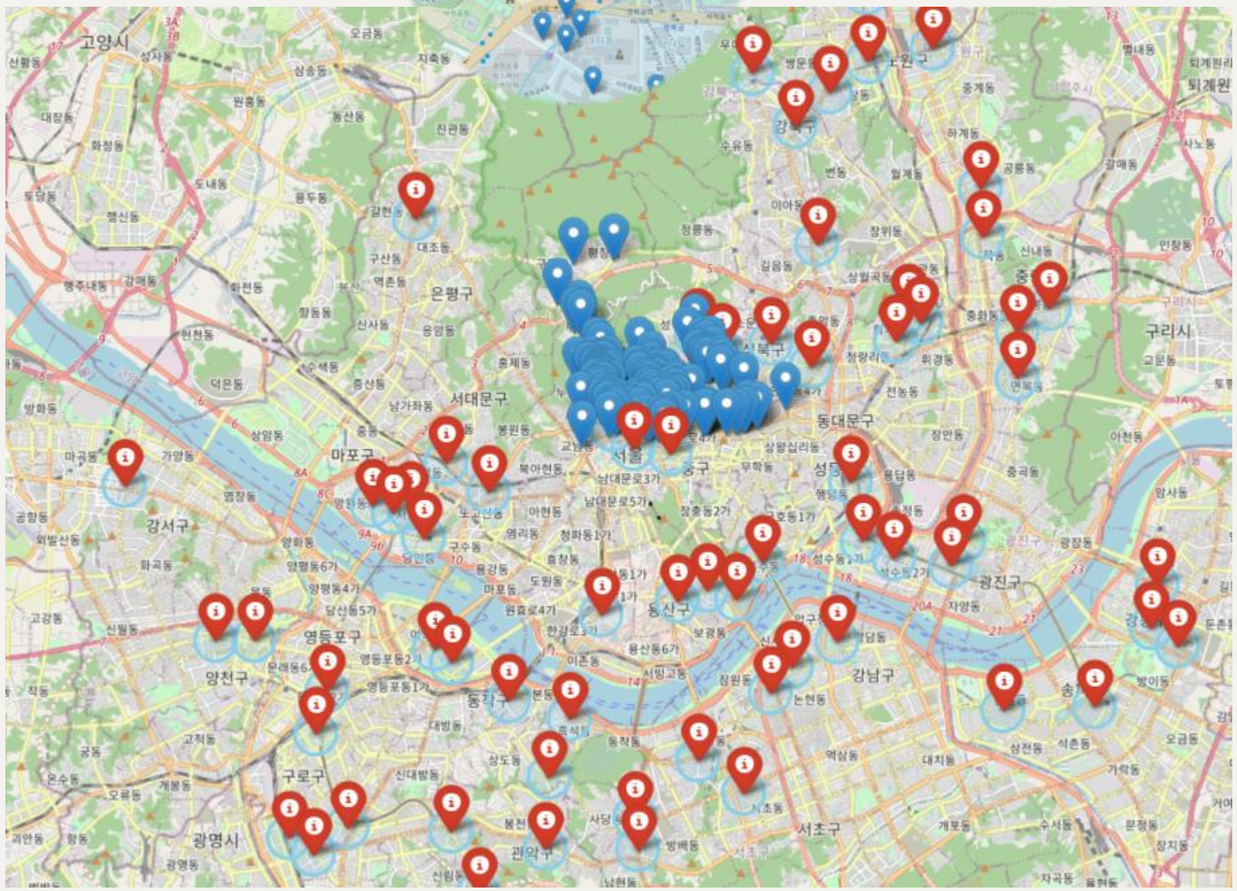
밀도기반 클러스터링(DBSCAN)을 이용해 가게의 위치를 군집화하였다.

가게 위치 (예시: 종로구)

클러스터 중심(centroid)

```
# DBSCAN으로 Cluster 구하기 --> 최대 확장거리 200m, 10개 이상 모여있을 시 Cluster로 간주

kms_per_radian = 6371.0088
epsilon = 0.2 / kms_per_radian #거리를 km 단위로 환산,
db = DBSCAN(eps=epsilon, min_samples=10, algorithm='ball_tree', metric='haversine').fit(np.radians(coords))
print(db)
print("-"*80)
cluster_labels = db.labels_
print(cluster_labels)
print("-"*80)
num_clusters = len(set(cluster_labels))
print(set(cluster_labels))
print("-"*80)
clusters = pd.Series([coords[cluster_labels == n] for n in range(num_clusters)])
print(clusters)
print("-"*80)
print('Number of clusters: {}'.format(num_clusters))
```

```
DBSCAN(algorithm='ball_tree', eps=3.139220275445233e-05, metric='haversine',
       min_samples=10)
--------------------------------------------------------------------------------
[-1  0  0 ... -1 -1 -1]
--------------------------------------------------------------------------------
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 6
2, 63, 64, 65, 66, 67, 68, -1}
--------------------------------------------------------------------------------
0     [[37.5260204, 127.037079], [37.5258965, 127.03...
1     [[37.5181233, 127.025218], [37.5217787, 127.02...
2     [[37.527649, 127.1248143], [37.5285963, 127.12...
3     [[37.538851, 127.124463], [37.5391443, 127.127...
4     [[37.52316800000001, 127.1311612], [37.5244541...
                            ...
65    [[37.5995475, 127.0965411], [37.5997975, 127.0...
66    [[37.5934535, 127.0880654], [37.5954283, 127.0...
67    [[37.61598, 127.0772346], [37.6163946, 127.078...
68    [[37.5837322, 127.0870122], [37.5829185, 127.0...
69                                                   []
Length: 70, dtype: object
--------------------------------------------------------------------------------
Number of clusters: 70
```



03 개발 과정

2. 약속장소와 맛집 추천

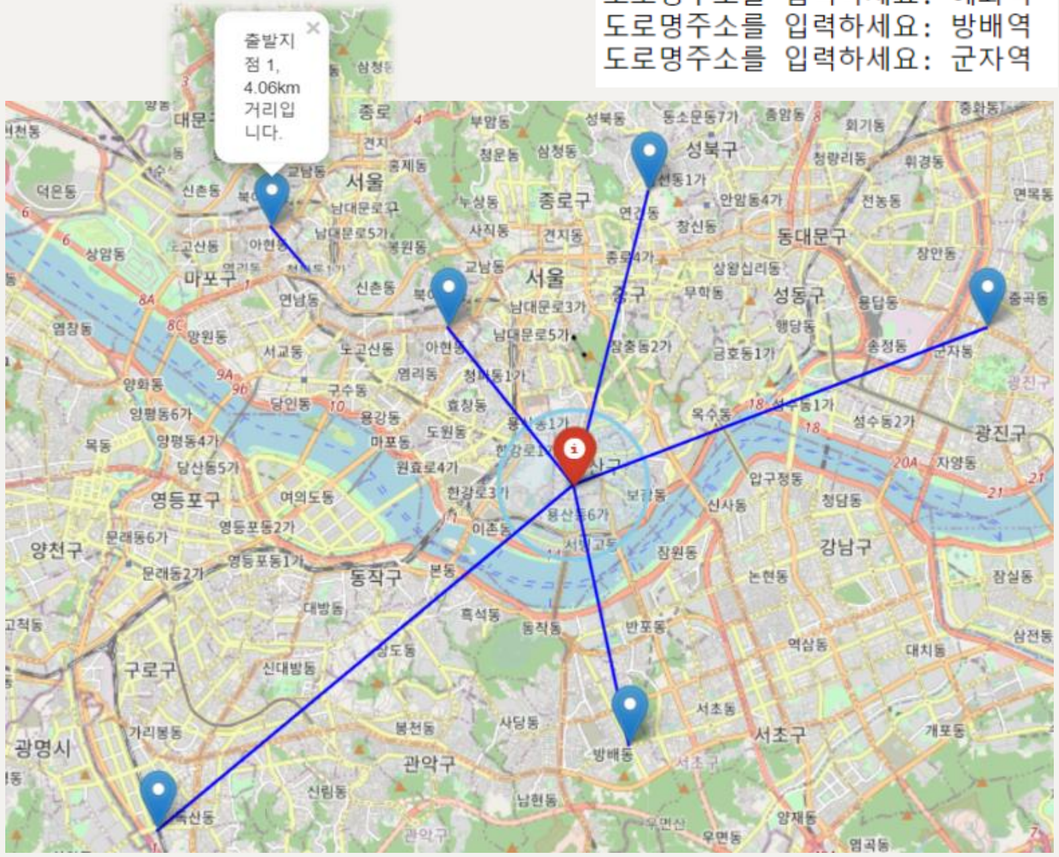
2) 중간 지점 산출

위도, 경도 좌표를 이용해 각 출발지점으로 부터의 중간지점을 산출하였다.

```
center_pointer(5)
```

```
도로명주소를 입력하세요: 독산역
도로명주소를 입력하세요: 아현역
도로명주소를 입력하세요: 혜화역
도로명주소를 입력하세요: 방배역
도로명주소를 입력하세요: 군자역
```

```python
def center_pointer(num_points):
    import googlemaps
    import folium
    from haversine import haversine

    gmaps = googlemaps.Client(key='AIzaSyCNetZW4eZiBmY7Zlwa9IDOj1CtZhgm_a0')
    latitude=[]
    longitude=[]
    coordinate=[]

    for i in range(num_points):
        address = input("도로명주소를 입력하세요: ")
        geocode_result = gmaps.geocode(address)
        geometry=geocode_result[0].get("geometry")
        location=geometry.get("location")
        lng=location.get("lng")
        lat=location.get("lat")
        latitude.append(lat)
        longitude.append(lng)
        coordinate.append((lat, lng))

    center_point=(sum(latitude)/len(latitude), sum(longitude)/len(longitude))
    map=folium.Map(location=[center_point[0], center_point[1]], zoom_start=12)
    folium.Marker([center_point[0], center_point[1]], popup="center", icon=folium.Icon(color="red")).add_to(map)
    folium.CircleMarker([center_point[0], center_point[1]], radius=50, color="skyblue", fill_color="skyblue").add_to(map)
    for i in range(len(coordinate)):
        folium.Marker([latitude[i], longitude[i]], popup="출발지점 {}, {}km 거리입니다.".format(i,
                                                round(haversine(coordinate[i], center_point), 2))).add_to(map)
        folium.PolyLine(locations=[coordinate[i], center_point],weight=2, color = 'blue').add_to(map)
    return map
```



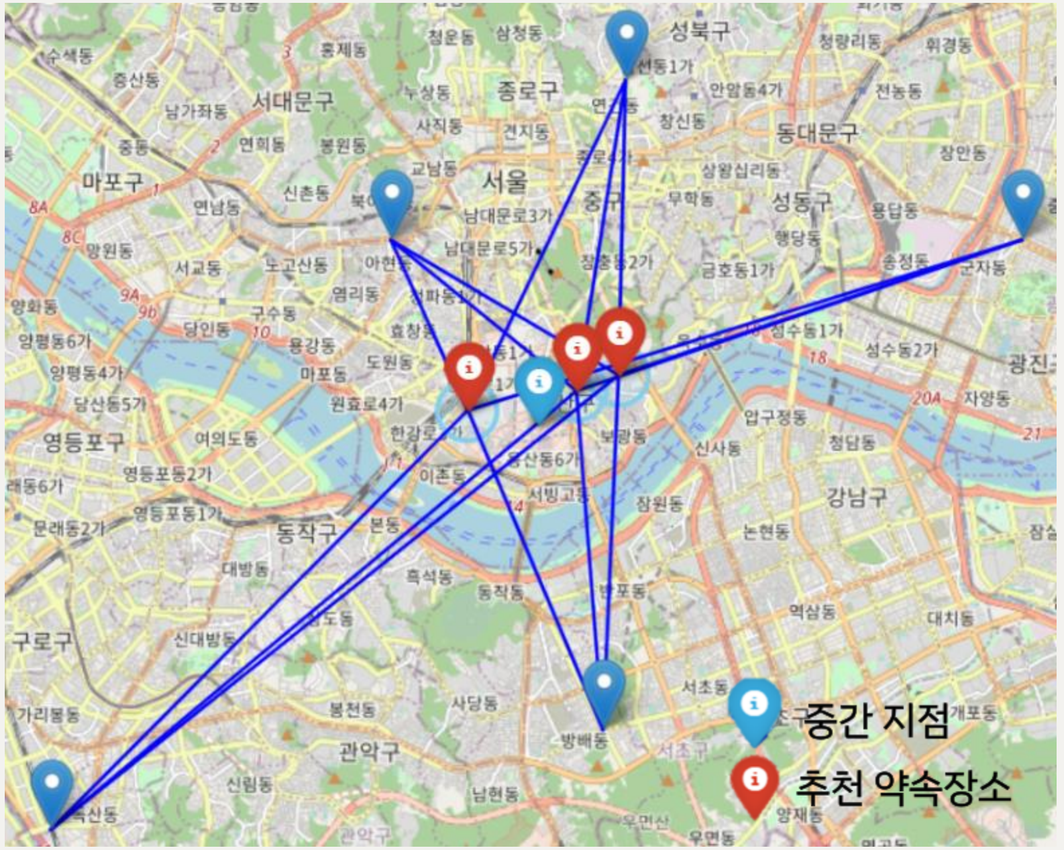
2. 약속장소와 맛집 추천

3) 약속장소 추천

중간지점으로 부터 가장 가까운 세 클러스터를 선택하였다. 세 클러스터는 추천하는 약속장소가 된다.

```
1번째로 가까운 추천지역: (37.53399743333333, 126.99231430555557), 거리 0.8640277565033238km
2번째로 가까운 추천지역: (37.53086211818183, 126.9709413), 거리 1.2541713093905145km
3번째로 가까운 추천지역: (37.53634404848484, 127.00058801515152), 거리 1.6101000305674023km
```

```
cluster = pd.read_csv("data/cluster 중간지점 좌표모음.csv", sep=",")
distance = []
for i in range(len(cluster)):
    d = haversine(center_point, (cluster["0"][i], cluster["1"][i]))
    distance.append(d)
s_distance= pd.Series(distance, name="distance")
cluster_distance=pd.concat([cluster, s_distance], axis=1)
cluster_distance2 = cluster_distance.sort_values(by="distance").reset_index(drop=True)
for i in range(3):
    print("{}번째로 가까운 추천지역: {}, 거리 {}km".format(i+1, (cluster_distance2["0"][i], cluster_distance2["1"][i]),
                                          cluster_distance2["distance"][i]))

map=folium.Map(location=[center_point[0], center_point[1]], zoom_start=12)
folium.Marker([center_point[0], center_point[1]], popup="중간 지점", icon=folium.Icon(color="blue")).add_to(map)
#folium.CircleMarker([center_point[0], center_point[1]], radius=50, color="skyblue", fill_color="skyblue").add_to(map)
for i in range(3):
    folium.Marker([cluster_distance2["0"][i], cluster_distance2["1"][i]], popup="추천지역 {}".format(i+1),
                  icon=folium.Icon(color="red")).add_to(map)
    folium.Circle([cluster_distance2["0"][i], cluster_distance2["1"][i]], radius=500, color="skyblue",
                  fill_color="skyblue").add_to(map)
for i in range(len(coordinate)):
    folium.Marker([latitude[i], longitude[i]], popup="출발지점 {}".format(i)).add_to(map)
    for j in range(3):
        folium.PolyLine(locations=[coordinate[i], (cluster_distance2["0"][j], cluster_distance2["1"][j])],
                        weight=2, color = 'blue').add_to(map)
return map
```



2. 약속장소와 맛집 추천

4) LSTM 모델을 이용해 누락된 평점 예측

망고플레이트 데이터에는 평점이 누락된 가게가 많아 앞서 생성한 LSTM 모델을 통해 평점을 예측하여 추가하고,
위치 좌표 및 클러스터 인덱스를 포함한 새로운 가게 데이터프레임을 만들었다.

이름	평점	주소	좌표	구	위도	경도	Cluster	점수
청우참치	4.5	서울특별시 강남구 강	(37.5182046, 127.01	강남구	37.5182046	127.019356	[0]	3.47
정돈	4.3	서울특별시 강남구 강	(37.5035911, 127.02	강남구	37.5035911	127.0269451	[0]	3.58
꼼다비뛰드	4.5	서울특별시 강남구 강	(37.5033264, 127.02	강남구	37.5033264	127.0294865	[0]	3.8
노들강	4.4	서울특별시 강남구 강	(37.5061534, 127.02	강남구	37.5061534	127.0258181	[0]	3
영동소금구이	4.4	서울특별시 강남구 강	(37.5088173, 127.02	강남구	37.5088173	127.0234738	[0]	3.53
아키토리묵	4.6	서울특별시 강남구 강	(37.5176433, 127.02	강남구	37.5176433	127.021214	[0]	3.32
목포집	4.3	서울특별시 강남구 강	(37.5174493, 127.02	강남구	37.5174493	127.0213987	[0]	3.33
야사이마끼 쿠이신보	4.4	서울특별시 강남구 강	(37.5186497, 127.02	강남구	37.5186497	127.0242496	[0]	3.53
칙피스	4.5	서울특별시 강남구 강	(37.5190769, 127.02	강남구	37.5190769	127.0247078	[0]	3.17
콴안다오	4.5	서울특별시 강남구 강	(37.5184448, 127.02	강남구	37.5184448	127.0214852	[0]	3.2
삼창교자	4.4	서울특별시 강남구 강	(37.5188793, 127.02	강남구	37.5188793	127.021733	[0]	3.4
신비갈비살	4.7	서울특별시 강남구 강	(37.5190821, 127.02	강남구	37.5190821	127.0201081	[0]	3.9
정돈 프리미엄	4.4	서울특별시 강남구 강	(37.5197151, 127.02	강남구	37.5197151	127.0210286	[0]	3.4
테일러커피	4.3	서울특별시 강남구 강	(37.5201201, 127.02	강남구	37.5201201	127.0211953	[0]	3.4
오사카카레콘유	4.4	서울특별시 강남구 강	(37.520621, 127.022	강남구	37.520621	127.0225949	[0]	3.6
레호이	4.5	서울특별시 강남구 강	(37.5201873, 127.02	강남구	37.5201873	127.0201706	[0]	3.42
신사치킨클럽	4.5	서울특별시 강남구 강	(37.5207118999999	강남구	37.5207119	127.0219296	[0]	3.8
양인환대	4.3	서울특별시 강남구 강	(37.4855218999999	강남구	37.4855219	127.0353536	[1]	3.39

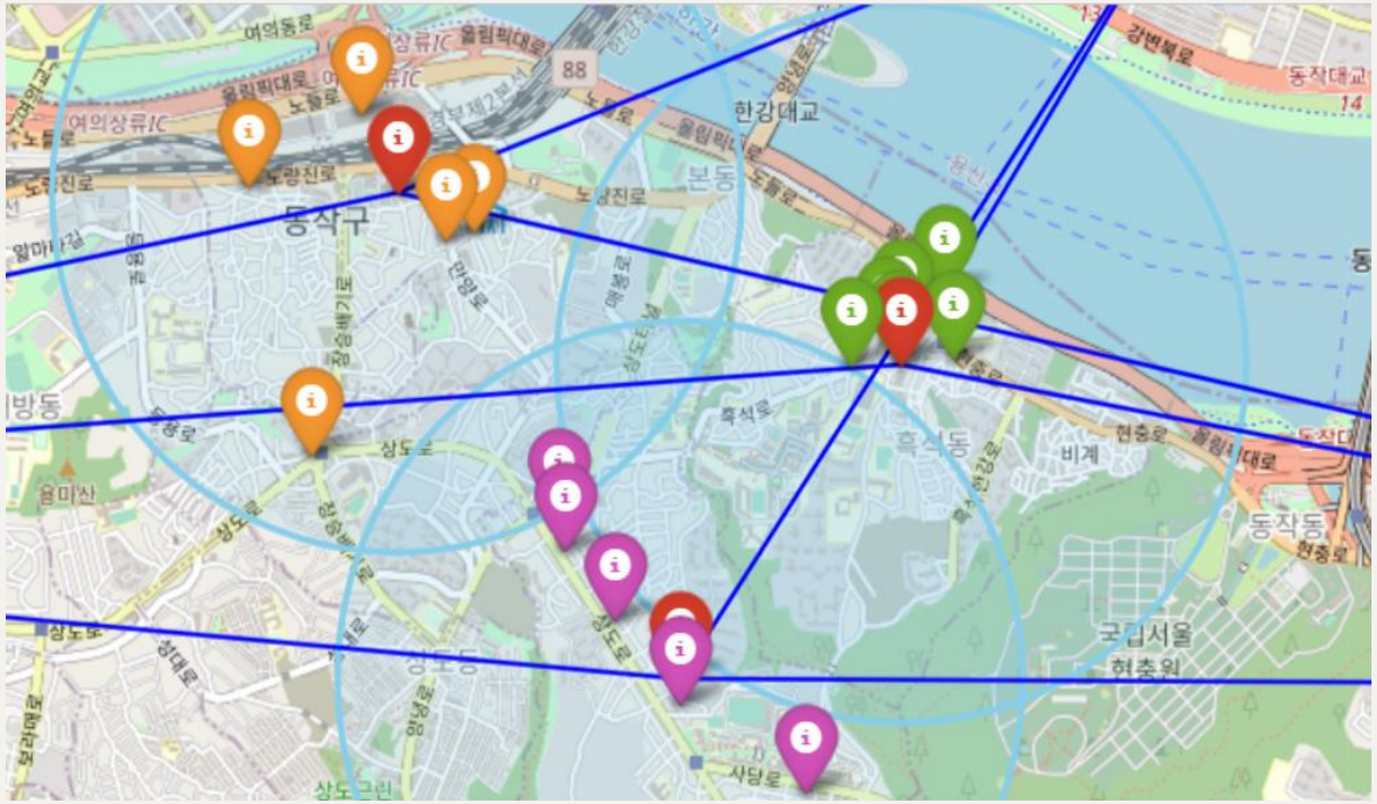
03 개발 과정

2. 약속장소와 맛집 추천

5) 평점 상위 5 맛집 추천

세 클러스터 각각의 가게에서 평점 상위 다섯 곳을 뽑아 총 15곳의 맛집을 추천한다.

```
#각 cluster 내 음식점 표시
places = pd.read_csv("data/점수포함 맛집리스트2.csv", sep=",")
color_list = ['green', 'purple', 'orange']
for i in range(3):
    restaurants = pd.DataFrame(columns=places.columns)

    for j in range(len(places)):
        if idx[i] in ast.literal_eval(places['Cluster'][j]):
            restaurants = pd.concat([restaurants, places.iloc[j].to_frame().T])

    restaurants["점수"] = restaurants["점수"].astype(float)
    hiscore=restaurants.nlargest(5, "점수", keep="all").index

    for k in range(len(hiscore)):
        folium.Marker([places["위도"][hiscore[k]], places["경도"][hiscore[k]]],
                      popup=places["이름"][hiscore[k]],
                      icon=folium.Icon(color=color_list[i])).add_to(map)
```



2. 약속장소와 맛집 추천

6) 추천 알고리즘 생성

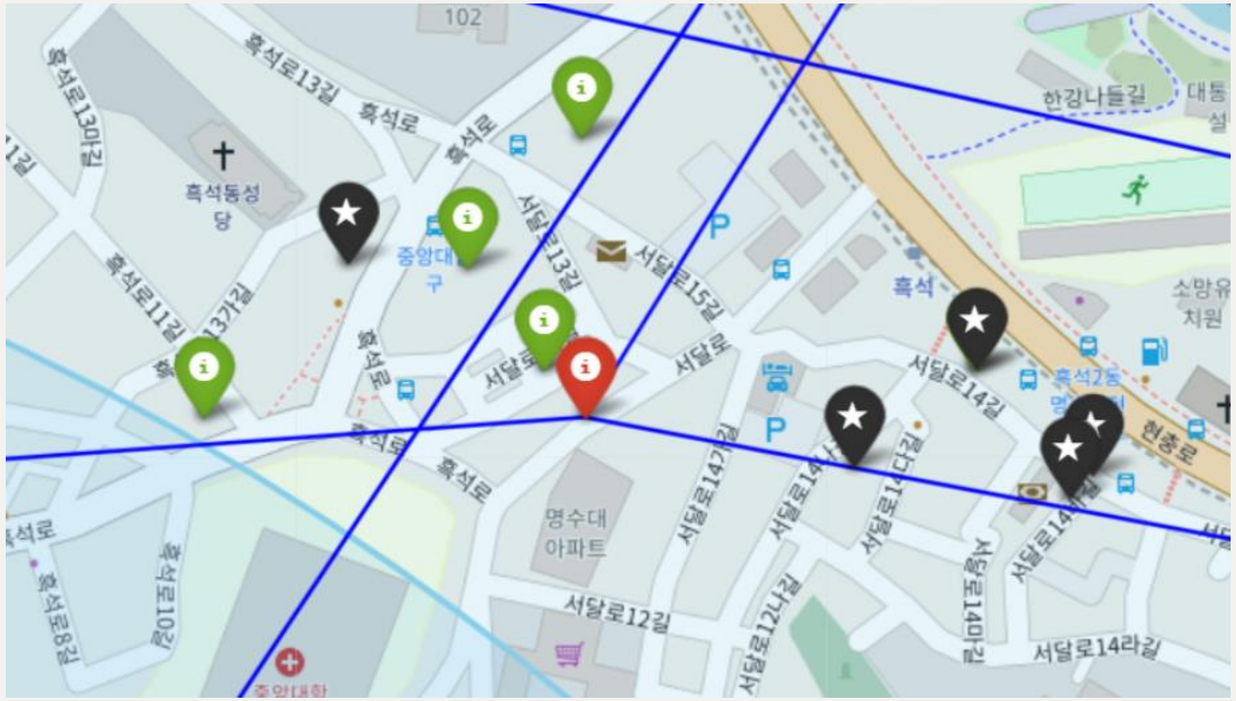
위에서 추천한 15곳의 맛집 중 사용자가 한 곳을 선택하면 추천 알고리즘을 통해 같은 클러스터 내에 있는 맛집 중 코사인 유사도가 높은 곳을 추출한다. 그 중 평점 상위 5곳을 추천한다.

```
###### 추천 알고리즘
# CountVectorizer
count_vect = CountVectorizer(min_df=0, ngram_range=(1,2))
menu_mat = count_vect.fit_transform(cluster_in_store["주력 메뉴"])

print(menu_mat.shape)

menu_sim = cosine_similarity(menu_mat, menu_mat)

menu_c_sim = cosine_similarity(menu_mat,menu_mat).argsort()[:,::-1]
```

```
# 2번째 선택때 추천해줄 명단.
result = get_recommend_menu_list(cluster_in_store, store_title = store_dict[select])

recommend_list = result.sort_values("평점", ascending=False)[:5]
```

	index	이름	평점	주소	주력 메뉴	가격대
20	11635	대박	4.3	서울특별시 동작구 흑석로 111-5	해산물 요리	2만원-3만원
4	11334	재팔이네닭발	4.3	서울특별시 동작구 서달로14길 34	닭 / 오리 요리	만원-2만원
15	11579	프랑세즈	4.1	서울특별시 동작구 현충로 96	베이커리	만원 미만
7	11370	성민양꼬치	4.1	서울특별시 동작구 서달로14나길 9	기타 중식	만원-2만원
12	11551	필수	4.0	서울특별시 동작구 현충로 102	일반 주점	만원-2만원

추천 알고리즘을 통해 추출된 맛집 5곳



04 구동 예시

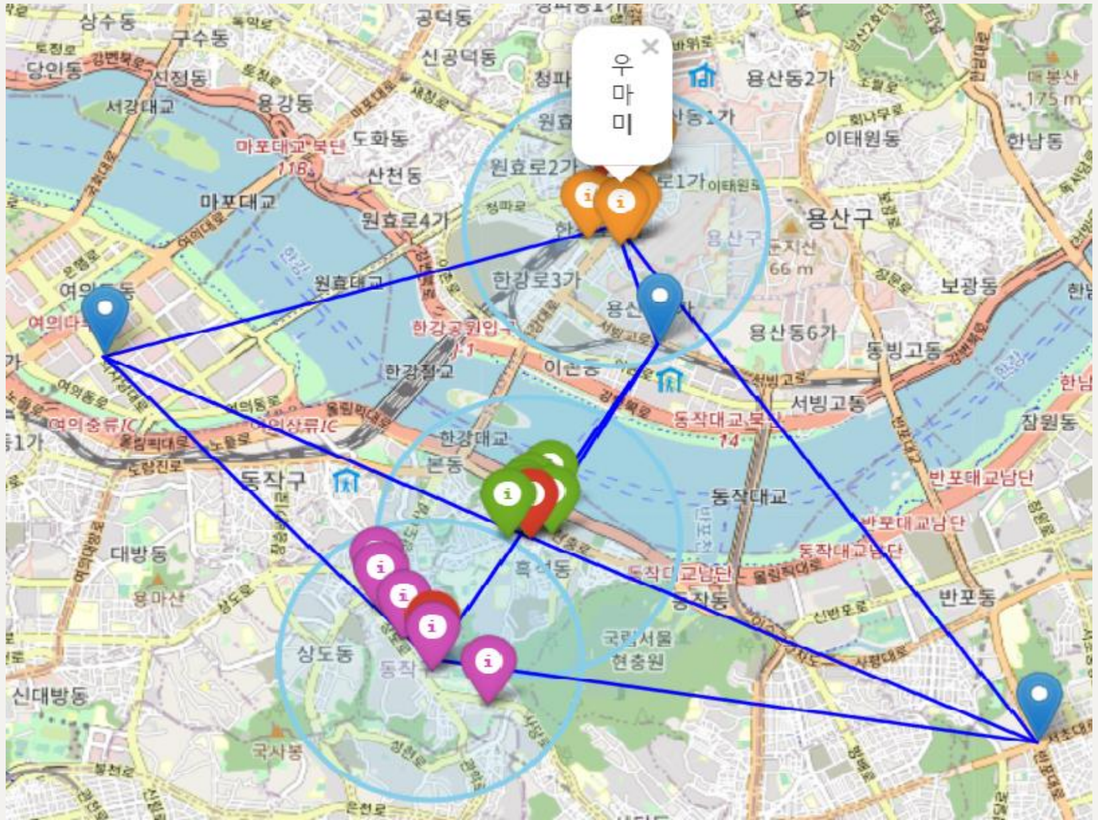
04 구동 예시

1. 출발위치 입력

여의도역, 이촌역, 서초역으로 입력하였다.

```
도로명주소를 입력하세요: 여의도역
도로명주소를 입력하세요: 이촌역
도로명주소를 입력하세요: 서초역
```

2. 지역 3곳 & 맛집 15곳 추천




```
1번째로 가까운 추천지역: (37.50771006538461, 126.96224941153844), 거리 0.7333412254747825km
2번째로 가까운 추천지역: (37.49824105806451, 126.95348508064517), 거리 2.026471901827503km
3번째로 가까운 추천지역: (37.53158447777778, 126.97006005), 거리 2.185707974369639km
```

04 구동 예시

3. 15곳 중 원하는 가게 한 곳 선택

3 : ' 뚜스뚜스 ' 를 선택하였다.

```
다음 가게 중 한 곳을 골라주세요.
0 : 흑석호치민
1 : 장독대
2 : 프랑세즈
3 : 뚜스뚜스
4 : 유정쌈밥
5 : 커리야
6 : 투디카페
7 : 비비엣
8 : 마루스시
9 : 우부래도
10 : 프라이밋 스테이크하우스
11 : 용산양꼬치
12 : 우마미
13 : 손문
14 : 르 굩
15 : BAR CHAM
다음 가게들 중 번호를 입력해주세요 : 3
```

4. 추천 알고리즘을 통해 추천된 맛집 5곳

	index	이름	평점	주소	주력 메뉴	가격대
20	11635	대박	4.3	서울특별시 동작구 흑석로 111-5	해산물 요리	2만원-3만원
4	11334	재팔이네닭발	4.3	서울특별시 동작구 서달로14길 34	닭 / 오리 요리	만원-2만원
15	11579	프랑세즈	4.1	서울특별시 동작구 현충로 96	베이커리	만원 미만
7	11370	성민양꼬치	4.1	서울특별시 동작구 서달로14나길 9	기타 중식	만원-2만원
3	11323	리앤홍	4.0	서울특별시 동작구 서달로14길 18 2F	카페 / 디저트	만원 미만

05 한계점 및 향후 개발방향

한계점

1. 서울로 한정된 데이터

서울로 한정하여 다른 지역에는 적용할 수 없다.

2. 망고플레이트 맛집 데이터

망고플레이트는 맛집 추천 사이트이기 때문에
리뷰와 평점이 좋아
평점 예측시 대부분 5점으로 높게 예측되었다.

향후 개발방향

1. 인터페이스 개선

유저들이 사용하기 용이하도록
Java Script 혹은 안드로이드 스튜디오를 이용해
웹 또는 앱으로 구현

2. DB구축하여 사용자 정보 저장

사용자가 어떤 메뉴를 선호하는지 DB에 저장하여
다음에 주소를 입력하면, 선택했던 메뉴 위주로 추천

감사합니다!

데이터 및 소스코드 : https://github.com/papertiger0016/teamwork