STARBUCKS Site Selection Analysis

Team The ONI
20121229 JunPyo Park
20161349 JaeHo Kim

Table of Contents

1. Motivation

2. Research Questions

3. Research Hypothesis

3. Research Methodology

4. Data Structure

5. Preliminary Data Analysis

6. Hypothesis Testing

7. Summary and Conclusion

8. Limitation and Further Development

9. Recommendation for Virtual Client

Motivation

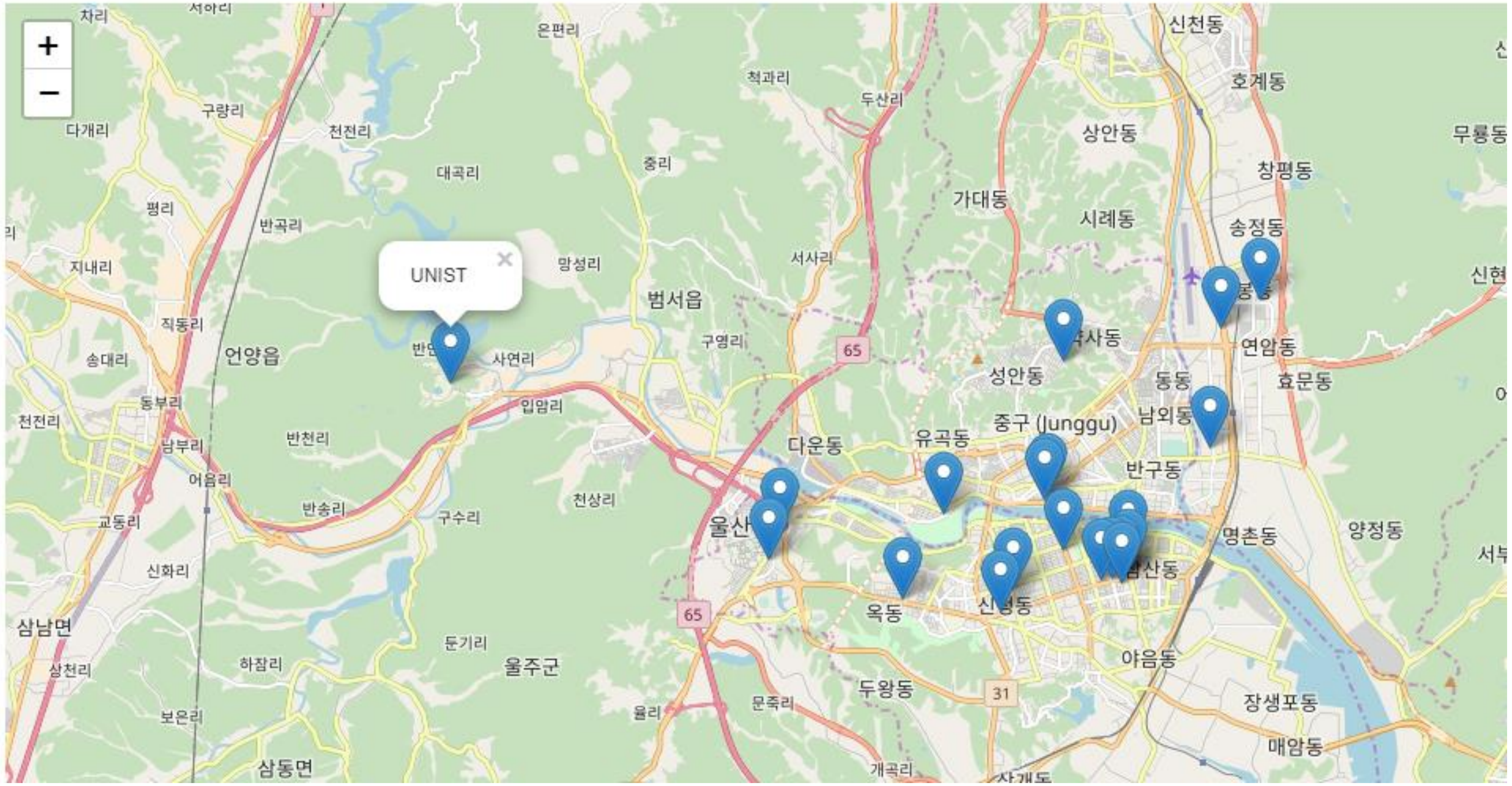
Why there are no STARBUCKS near to UNIST?

STARBUCKS in Ulsan

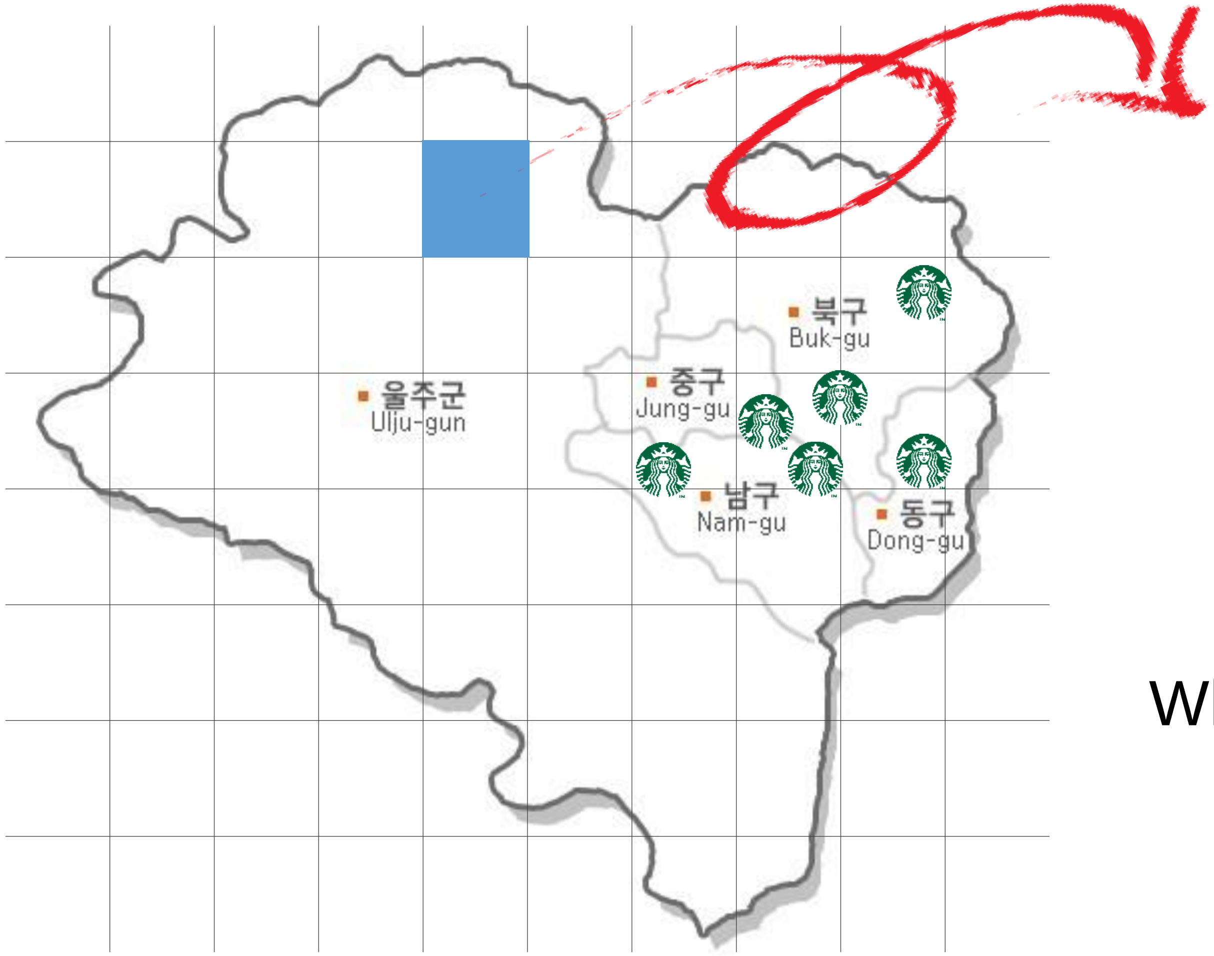
In [41]:

```
map_osm = folium.Map(location=울산태화, zoom_start=13)

for ix, row in df.iterrows():
    location = (row['lat'], row['lot'])
    folium.Marker(location, popup=row['s_name'] + '점').add_to(map_osm)

unist_loc = (35.573830,129.190944)
folium.Marker(unist_loc, popup='UNIST').add_to(map_osm)
map_osm
```

Out[41]:



Research Question



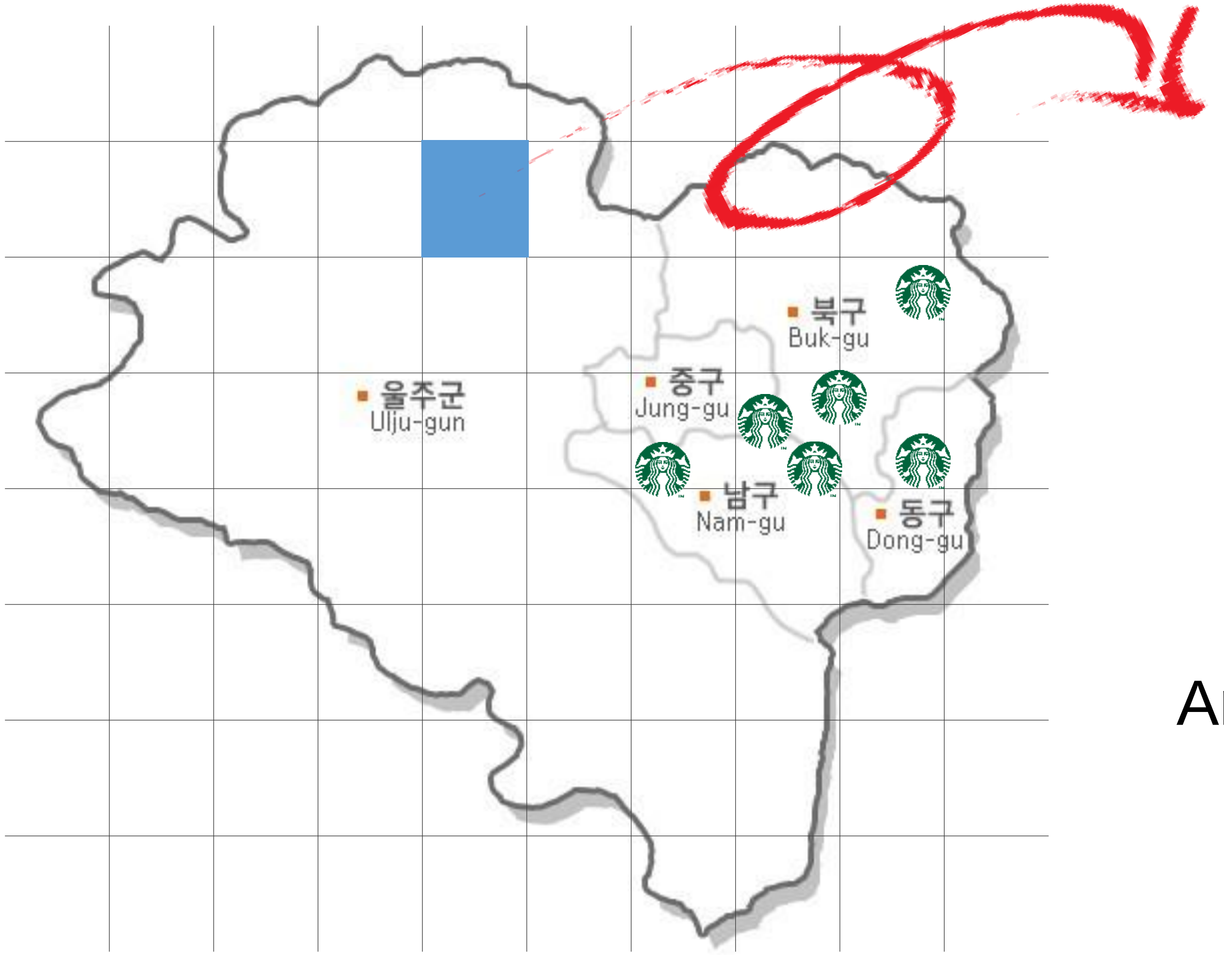
Has Multiple Features

- Longitude & Latitude
- Road Traffic Volume
- Number of Apartments
- Population Distribution
- Number of Office Worker
- Average Income Class

Question

Which one is important or not?

Research Hypothesis



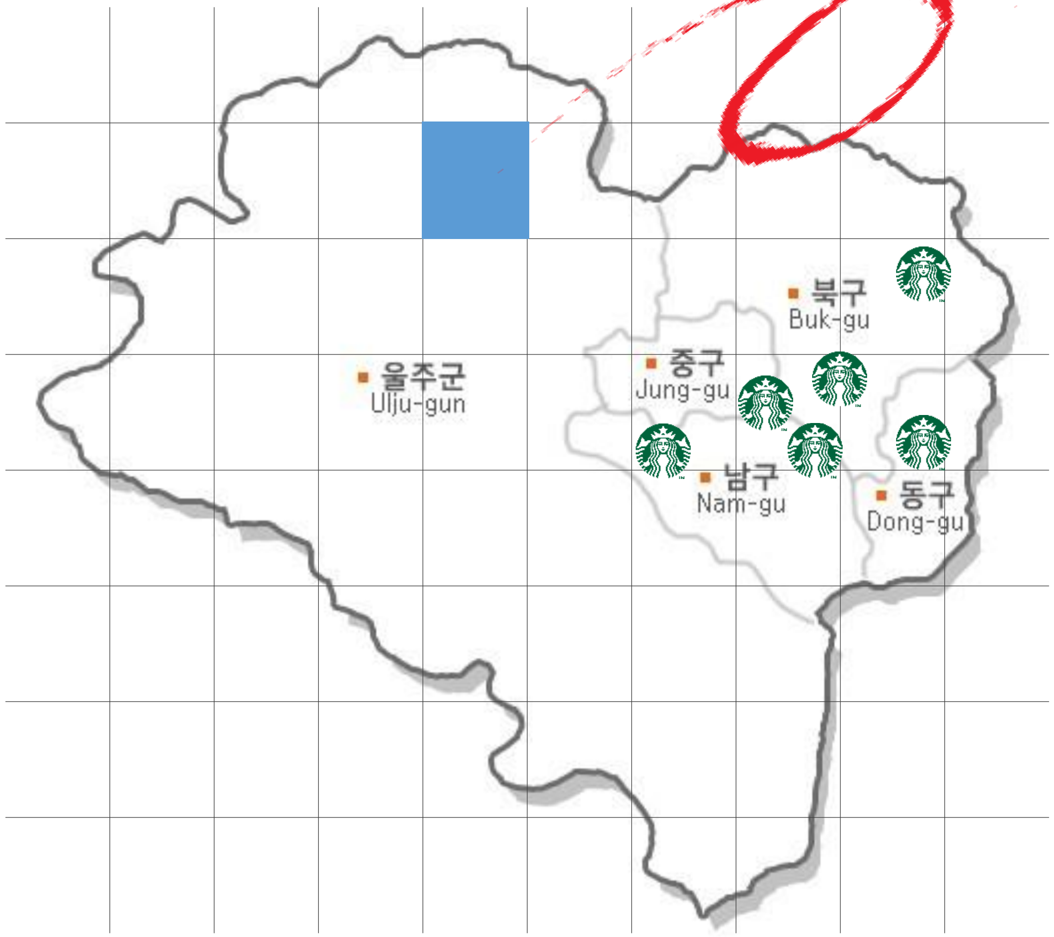
Has Multiple Features

- Longitude & Latitude
- Road Traffic Volume
- Number of Apartments
- Population Distribution
- Number of Office Worker
- Average Income Class

Hypothesis

Among them there are some significance features.

Research Methodology : Logistic Regression



Has Multiple Features

- Longitude & Latitude
- Road Traffic Volume
- Number of Apartments
- Population Distribution
- Number of Office Worker
- Average Income Class

Logistic Regression
to get Odds or Probability

Hypothesis Testing

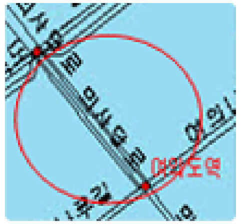
Data Structure

Data Source and Overview

Ulsan Node-Link Data

Ulsan Node Data from ITS (국가교통정보센터)

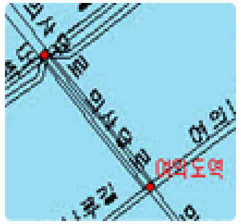
노드/링크 구성



필드명	NODE_ID	NODE_TYPE	NODE_NAME	TURN_P	REMARK
속성명	노드 ID 노드유형 교차로명칭 회전제한유무 비고	노드유형	교차로명칭	회전제한유무	비고

노드
링크를 구분하는 점(링크 시작점, 링크 종료점)으로 표준 로드/링크 구축 운영 지침에 따라 도로교차점, 도로 시종점, 교통통제점, 도로 구조 변환점,행정구역 변환점, 도로 운영 변환점, 교통 진출입점. 그외에 ITS사업 주체가 필요에 따라 정하는 지점

부가정보(회전제한정보)
회전제한이 존재하는 노드에 한해 회전제한 상세 정보를 입력한 테이블



필드명	LINK_ID	F_NODE	T_NODE	ROAD_USE	LANES	ROAD_RANK	ROAD_TYPE	ROAD_NO
속성명	링크 ID	시작노드 ID	종료노드 ID	도로사용여부	차로수	도로등급	도로유형	도로번호

필드명	ROAD_NAME	MULTI_LINK	CONNECT	MAX_SPD	REST_VEH	REST_W	REST_H	REMARK
속성명	도로명	중용구간여부	연결로코드	최고제한속도	통과제한차량	통과제한하중	통과제한높이	비고

링크
노드와 노드를 이은 도로중심선을 방향별로 일정간격 이격시켜 생성한 선으로서 실제 도로구간에 대한 정보를 담게됨

Ulsan Node-Link Data

Ulsan Node Data Overview

In [2]:
```
nodes = pd.read_csv('ulsan_nodes.csv')
links = pd.read_csv('ulsan_links.csv')
```

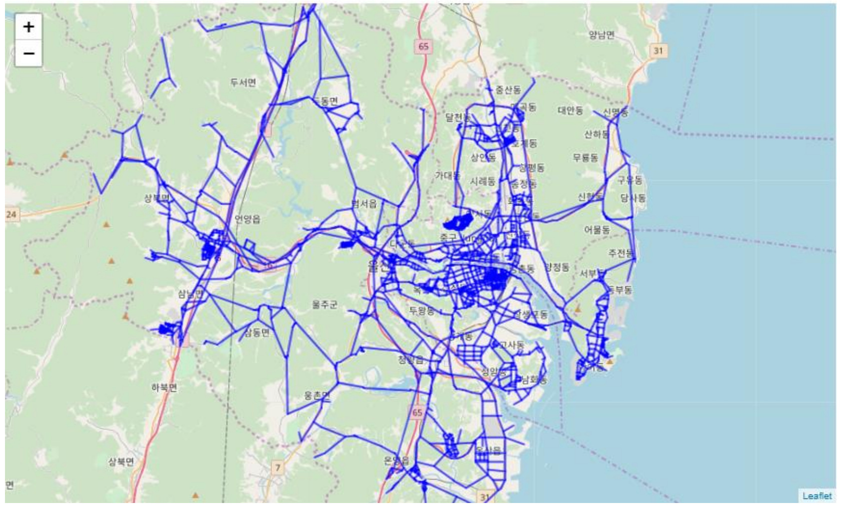
In [67]:
```
nodes.sort_values(by='Lanes', ascending=False).head(10)
```

Out[67]:

	Id	NODE_NAME	STNL_REG	latitude	longitude	Lanes
34	1920011300	-	192	35.558685	129.327780	32.0
1128	1930003300	신부로터리	193	35.550077	129.263743	32.0
638	1930002304	공업탑로터리	193	35.532176	129.307525	30.0
603	1930041600	-	193	35.539861	129.347710	28.0
35	1920012900	mbc사거리	192	35.561221	129.331311	28.0
179	1920015000	병영사거리	192	35.568539	129.343471	28.0
600	1930041900	삼산사거리	193	35.539911	129.344027	28.0
596	1930049000	예술회관사거리	193	35.541637	129.326243	28.0
36	1920013600	복산사거리	192	35.564049	129.336121	28.0
619	1930059300	kbs사거리	193	35.545322	129.324646	28.0

Ulsan Node-Link Data

Ulsan Link(Edge) Data

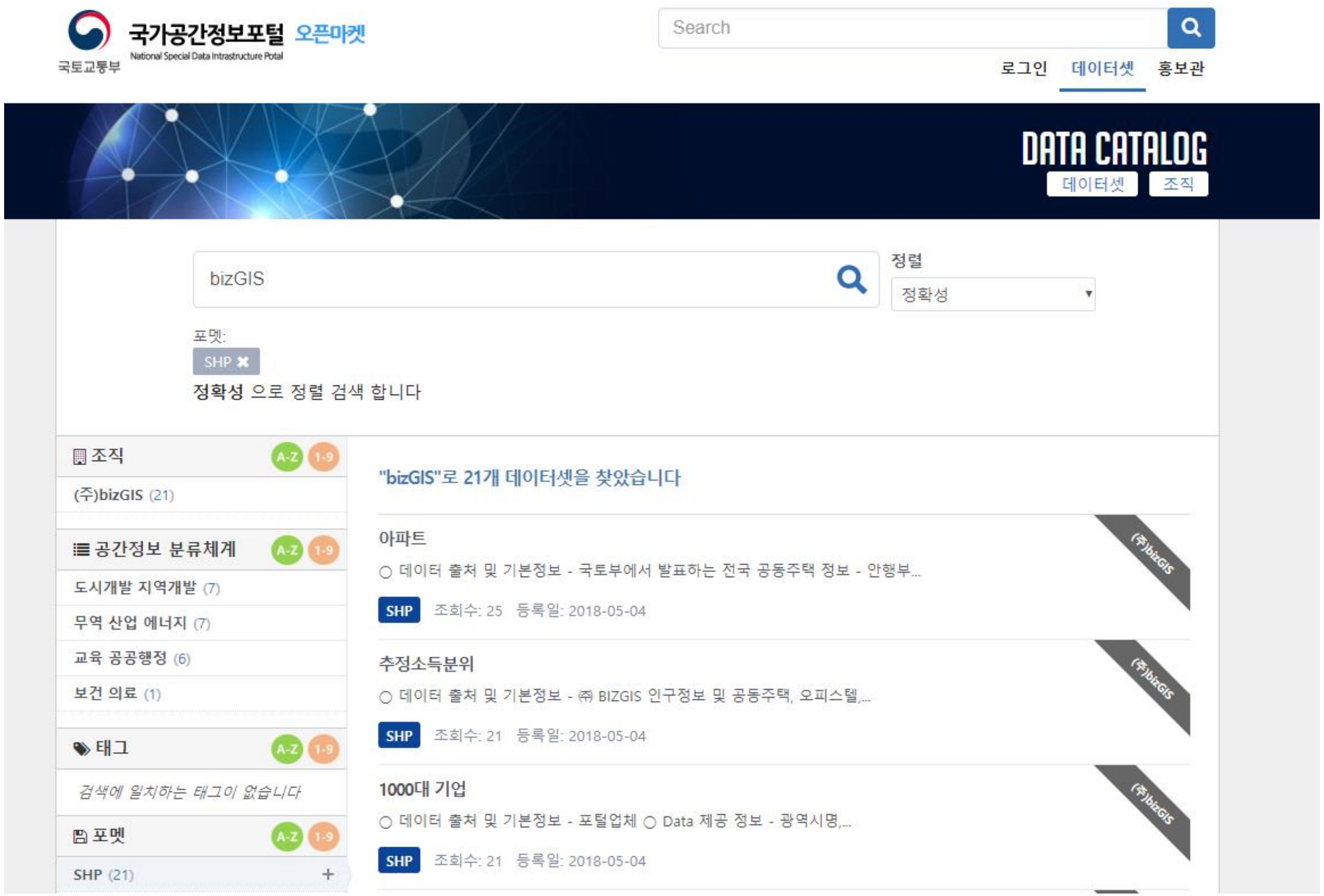


Data Table

Nodes Edges | Configuration | Add node | Add edge | Search/Replace | Import Spreadsheet | Export table

Source	Target	...	...	Weight
1610003100	1680004000	...	0	5367,867188
1610003100	1610003000	...	1	5223,630371
1610003100	1680003800	...	2	4171,646973
1610003000	1610002900	...	3	768,006042
1610003000	1610025700	...	4	812,851807
1610002900	1610002800	...	5	973,03772
1610002400	1610002200	...	6	570,108093
1610002400	1610002500	...	7	217,577591
1610002300	1610002200	...	8	597,513245
1610002300	1610002100	...	9	838,134888
1610002300	1610018200	...	10	810,184448
1610002300	1610025700	...	11	1978,141602
1610002100	1610002200	...	12	566,683655
1610002100	1610000800	...	13	2318,129639
1610002100	1610018200	...	14	619,151855
1610000800	1610000700	...	15	399,196625
1610000800	1610000600	...	16	1147,136963
1610000700	1610007700	...	17	524,237122
1610000700	1610022800	...	18	585,383179
1610000700	1610000600	...	19	875,785339
1610000600	1610021100	...	20	912,308167
1610000600	1610007700	...	21	586,073486
1610000500	1610000300	...	22	279,882141
1610000500	1610005200	...	23	95,262077
1610000500	1610021300	...	24	80,854507

Data Source



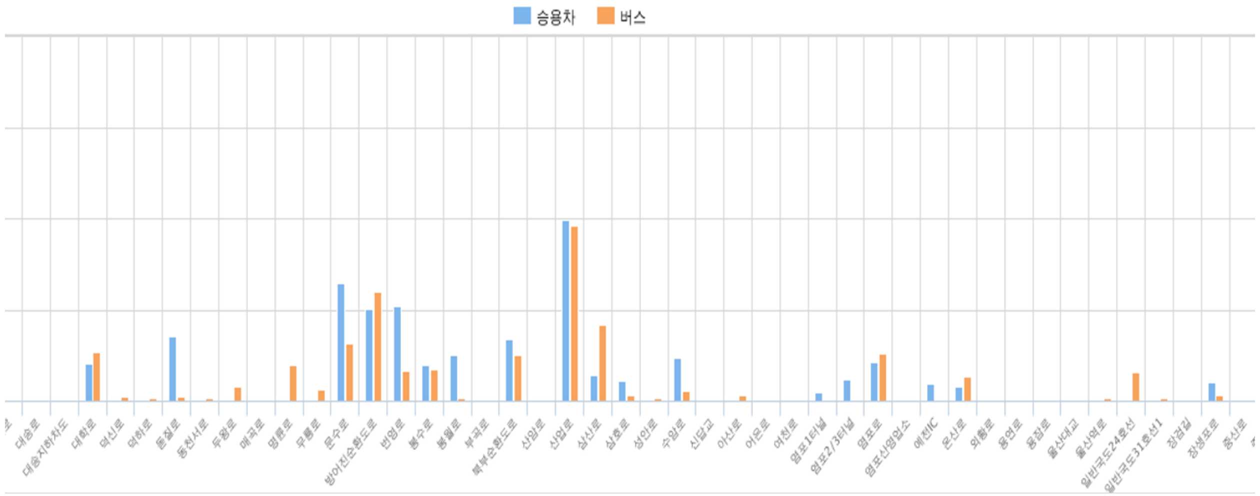
XsDB_주거인구_100M_TM.dbf
XsDB_주거인구_100M_TM.sbn
XsDB_주거인구_100M_TM.sbx
XsDB_주거인구_100M_TM.shp
XsDB_주거인구_100M_TM.shx

XsDB_아파트_100M_TM.dbf
XsDB_아파트_100M_TM.sbn
XsDB_아파트_100M_TM.sbx
XsDB_아파트_100M_TM.shp
XsDB_아파트_100M_TM.shx

http://data.nsdi.go.kr/dataset?q=bizGIS&sort=score+desc%2C+metadata_modified+desc&res_format=SHP&page=1

Data Source

	A	B	C	D	E	F	G	H	I	J	K	L
1	32012_가로별 교통량(일)											
2												
3	단위 : 대											
4												
5						일자	20180404	20180404	20180405	20180405	계	계
6	가로명	방면	요일	순번(LV2)	시작지점	종료지점	승용차	버스	승용차	버스	승용차	버스
7	강남로	상행	수요일	1	태화R	번영교남교차로	0	0	-	-	0	0
8	강남로	상행	수요일	2	번영교남교차로	학성교남교차로	26,546	0	-	-	26,546	0
9	강남로	상행	수요일	3	학성교남교차로	명촌교남단	0	20	-	-	0	20
10	강남로	상행	목요일	1	태화R	번영교남교차로	-	-	0	0	0	0
11	강남로	상행	목요일	2	번영교남교차로	학성교남교차로	-	-	26,162	0	26,162	0
12	강남로	상행	목요일	3	학성교남교차로	명촌교남단	-	-	0	24	0	24
13	강남로	하행	수요일	1	명촌교남단	학성교남교차로	0	20	-	-	0	20
14	강남로	하행	수요일	2	학성교남교차로	번영교남교차로	28,043	0	-	-	28,043	0
15	강남로	하행	수요일	3	번영교남교차로	태화R	0	0	-	-	0	0
16	강남로	하행	목요일	1	명촌교남단	학성교남교차로	-	-	0	26	0	26
17	강남로	하행	목요일	2	학성교남교차로	번영교남교차로	-	-	27,171	0	27,171	0
18	강남로	하행	목요일	3	번영교남교차로	태화R	-	-	0	0	0	0
19	강남로	계					54,589	40	53,333	50	107,922	90
20	강북로	상행	수요일	1	태화교사거리	번영교북교차로	0	1,097	-	-	0	1,097



http://utrhub.its.ulsan.kr/

Data Processing

*.shp files(Database -> SHP_to_CSV.ipynb)

Using Python to covert to csv files

Construct dataframe by using shp file

In [3]:
```
# read data (Copy all files from nodelink into nodelink folder: I made it.)
# using old_data
shp_path_node = './HOUSPOP/XsDB_주거인구_100M_TM.shp'
sf_node = shapefile.Reader(shp_path_node)
```

.

.

.

In [243]:
```
ulsan_data.to_csv('Ulsan_population.csv', encoding='cp949')
```

Ulsan_apart	2018-04-29 오전 4:43
Ulsan_income_class	2018-04-29 오전 4:57
Ulsan_population	2018-04-29 오전 4:15
Ulsan_STARBUCKS_list	2018-05-04 오후 3:19
Ulsan_worker_num	2018-04-29 오전 5:05

SHP File Processing Result

Example : Apart Price Distribution(for each unit cell)

PY_SUM	PR_SUM	PY_10U	PY_10	PY_20	PY_30	PY_40	PY_50O	PR_05U	PR_05	PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7_O	latitude	longitude
2428	7.29E+09	0	0	68	22	0	0	0	90	0	0	0	0	0	0	0	35.56446	129.1153
2391	1.38E+10	0	132	0	0	0	0	0	20	112	0	0	0	0	0	0	35.54047	129.2616
1044	3.14E+09	0	0	48	0	0	0	0	48	0	0	0	0	0	0	0	35.5201	129.312
2594	8.03E+09	0	139	30	0	0	0	130	39	0	0	0	0	0	0	0	35.52191	129.3121
10669	6.33E+10	0	195	300	0	0	0	0	80	415	0	0	0	0	0	0	35.52718	129.3243
5058	2.94E+10	0	0	200	0	0	0	0	36	164	0	0	0	0	0	0	35.52891	129.3309
4022	1.28E+10	0	84	80	20	0	0	84	80	20	0	0	0	0	0	0	35.47213	129.4093
3478	1.37E+10	0	0	144	0	0	0	0	108	36	0	0	0	0	0	0	35.48464	129.4184
6056	2.65E+10	0	349	0	0	0	0	0	349	0	0	0	0	0	0	0	35.52062	129.4256
5722	1.96E+10	0	97	180	0	0	0	0	277	0	0	0	0	0	0	0	35.49274	129.4196
1882	5.87E+09	0	105	0	0	0	0	8	97	0	0	0	0	0	0	0	35.49273	129.4207
2050	1.03E+10	0	114	0	0	0	0	0	114	0	0	0	0	0	0	0	35.58084	129.3627
530	1.08E+09	56	0	0	0	0	0	56	0	0	0	0	0	0	0	0	35.56714	129.1175
5302	1.54E+10	0	222	50	0	0	0	19	253	0	0	0	0	0	0	0	35.48423	129.2949
7100	3.09E+10	0	0	180	0	60	0	0	8	232	0	0	0	0	0	0	35.56568	129.2653
1816	6.16E+09	0	47	30	10	0	0	0	77	10	0	0	0	0	0	0	35.556	129.327
991	2.65E+09	0	27	23	0	0	0	27	23	0	0	0	0	0	0	0	35.57834	129.3439
2156	1.09E+10	0	0	84	0	0	0	0	0	84	0	0	0	0	0	0	35.5835	129.366
627	1.07E+09	0	36	0	0	0	0	36	0	0	0	0	0	0	0	0	35.43272	129.3073
1168	4.11E+09	0	65	0	0	0	0	0	65	0	0	0	0	0	0	0	35.52459	129.3143
4217	1.92E+10	0	0	176	0	0	0	0	36	140	0	0	0	0	0	0	35.54849	129.3533
2385	1.73E+10	0	0	0	0	50	0	0	0	0	2	48	0	0	0	0	35.55627	129.3016
1025	8.42E+09	0	0	40	0	0	0	0	0	4	36	0	0	0	0	0	35.56494	129.3337
991	2.55E+09	32	44	0	0	0	0	71	5	0	0	0	0	0	0	0	35.54868	129.3368
2180	1.04E+10	0	121	0	0	0	0	0	121	0	0	0	0	0	0	0	35.48204	129.4095
6075	3.86E+10	0	0	244	0	0	0	0	0	244	0	0	0	0	0	0	35.56859	129.2455
3140	1.86E+10	0	0	57	14	28	0	0	0	58	41	0	0	0	0	0	35.56536	129.1153
3126	2.6E+10	0	0	0	25	56	0	0	0	0	22	57	2	0	0	0	35.5389	129.3245
1521	6.91E+09	0	84	0	0	0	0	0	84	0	0	0	0	0	0	0	35.52896	129.3265
1641	4.16E+09	0	35	47	0	0	0	35	47	0	0	0	0	0	0	0	35.48732	129.4206
2678	8.38E+09	0	148	0	0	0	0	0	148	0	0	0	0	0	0	0	35.56161	129.2223
5836	4.12E+10	0	0	0	149	0	0	0	0	0	149	0	0	0	0	0	35.52004	129.3175
4568	1.66E+10	0	91	120	0	0	0	0	211	0	0	0	0	0	0	0	35.61681	129.4472

Special attribute (STARBUCKS Score)

Calculating 'Starbucks Score' by using the statistics by age

In [473]:
```
weight = np.array([0.0093,0.3,0.42,0.22,0.045])
```

In [474]:
```
# construct the 'POP_50_Over' attribute
work_num_data['POP_50_O'] = work_num_data['POP_50'] + work_num_data['POP_60'] + work_num_data['POP_70'] + work_num_data['POP_80_O
```

In [475]:
```
# Weighted average to get 'STARBUCKS SCORE'
work_num_data['SB_score'] = work_num_data[['POP_10', 'POP_20', 'POP_30', 'POP_40', 'POP_50_O']].apply(lambda x : np.matmul(x,weig
```

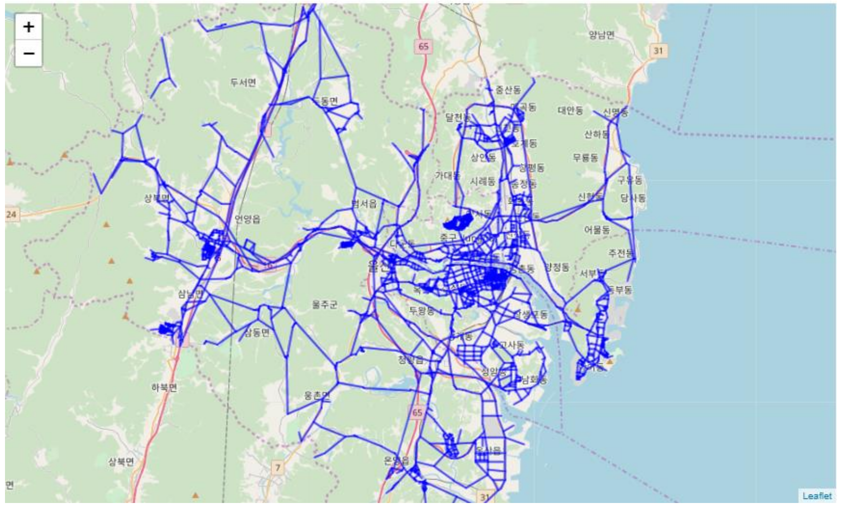
In [476]:
```
work_num_data['SB_score'].sort_values(ascending=False).head(10)
```

Out[476]:
```
1130    392.306864
6452    200.761776
1497    153.827386
7664    136.096706
7807     95.447389
8304     94.060246
1664     92.151632
5909     88.265151
3749     83.859728
2605     80.758033
Name: SB_score, dtype: float64
```

Apply same method to office worker(직장인) distribution data!

Traffic Data Construction Failure

Plan : Combine traffic data to Ulsan road network



	A	B	C	D	E	F	G	H	I	J	K	L
1	32012_가로별 교통량(일)											
2												
3	단위 : 대											
4												
5						일자	20180404	20180404	20180405	20180405	계	계
6	가로명	방면	요일	순번(LV2)	시작지점	종료지점	승용차	버스	승용차	버스	승용차	버스
7	강남로	상행	수요일	1	태화R	번영교남교차로	0	0	-	-	0	0
8	강남로	상행	수요일	2	번영교남교차로	학성교남교차로	26,546	0	-	-	26,546	0
9	강남로	상행	수요일	3	학성교남교차로	명촌교남단	0	20	-	-	0	20
10	강남로	상행	목요일	1	태화R	번영교남교차로	-	-	0	0	0	0
11	강남로	상행	목요일	2	번영교남교차로	학성교남교차로	-	-	26,162	0	26,162	0
12	강남로	상행	목요일	3	학성교남교차로	명촌교남단	-	-	0	24	0	24
13	강남로	하행	수요일	1	명촌교남단	학성교남교차로	0	20	-	-	0	20
14	강남로	하행	수요일	2	학성교남교차로	번영교남교차로	28,043	0	-	-	28,043	0
15	강남로	하행	수요일	3	번영교남교차로	태화R	0	0	-	-	0	0
16	강남로	하행	목요일	1	명촌교남단	학성교남교차로	-	-	0	26	0	26
17	강남로	하행	목요일	2	학성교남교차로	번영교남교차로	-	-	27,171	0	27,171	0
18	강남로	하행	목요일	3	번영교남교차로	태화R	-	-	0	0	0	0
19	강남로	계					54,589	40	53,333	50	107,922	90
20	강북로	상행	수요일	1	태화교사거리	번영교북교차로	0	1,097	-	-	0	1,097

Problem : Road names are different from each dataset…….

Traffic Data Construction Failure

Problem : Too many unclassified traffic data…

Ulsan_Weekly_Traffic -> Add_Traffic_.ipynb

In [218]:
```
total_car = 0
total_bus = 0
missing_car = 0
missing_bus = 0
for idx,row in data.iterrows():
    total_car += row['Car'] * 2
    total_bus += row['Bus'] * 2

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Car']) > 0 ): # 노드에 있는 도로의 경우
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Car'] += row['Car'] # 자동차 대수를 증가 시킵니다.
    else:
        missing_car += row['Car'] # 노드에 없는 이름인 경우 missing value 를 증가 시킵니다.

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Car']) > 0 ):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Car'] += row['Car']
    else:
        missing_car += row['Car']

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Bus']) > 0):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Bus'] += row['Bus']
    else:
        missing_bus += row['Bus']

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Bus']) > 0):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Bus'] += row['Bus']
    else:
        missing_bus += row['Bus']
```

In [220]:
```
missing_car / total_car
```
Out[220]:
```
0.7124665727606904
```

In [221]:
```
missing_bus / total_bus
```
Out[221]:
```
0.7306401211899067
```

Traffic Data Construction Failure

Solution

In [218]:
```python
total_car = 0
total_bus = 0
missing_car = 0
missing_bus = 0
for idx,row in data.iterrows():
    total_car += row['Car'] * 2
    total_bus += row['Bus'] * 2

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Car']) > 0 ): # 노드에 있는 도로의 경우
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Car'] += row['Car'] # 자동차 대수를 증가 시킵니다.
    else:
        missing_car += row['Car'] # 노드에 없는 이름인 경우 missing value 를 증가 시킵니다.

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Car']) > 0 ):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Car'] += row['Car']
    else:
        missing_car += row['Car']

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Bus']) > 0):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Bus'] += row['Bus']
    else:
        missing_bus += row['Bus']

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Bus']) > 0):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Bus'] += row['Bus']
    else:
        missing_bus += row['Bus']
```

Throw it away...

In [220]:
```python
missing_car / total_car
```
Out[220]: 0.7124665727606904

In [221]:
```python
missing_bus / total_bus
```
Out[221]: 0.7306401211899067

Final Data Overview

Final Merged Data

In [4]:
```
data = pd.read_csv('unlabled_final_data.csv')
data = data[['latitude','longitude','SB_score','CLSS','SB_worker_score','PR_per_PY']]
data.head(10)
```

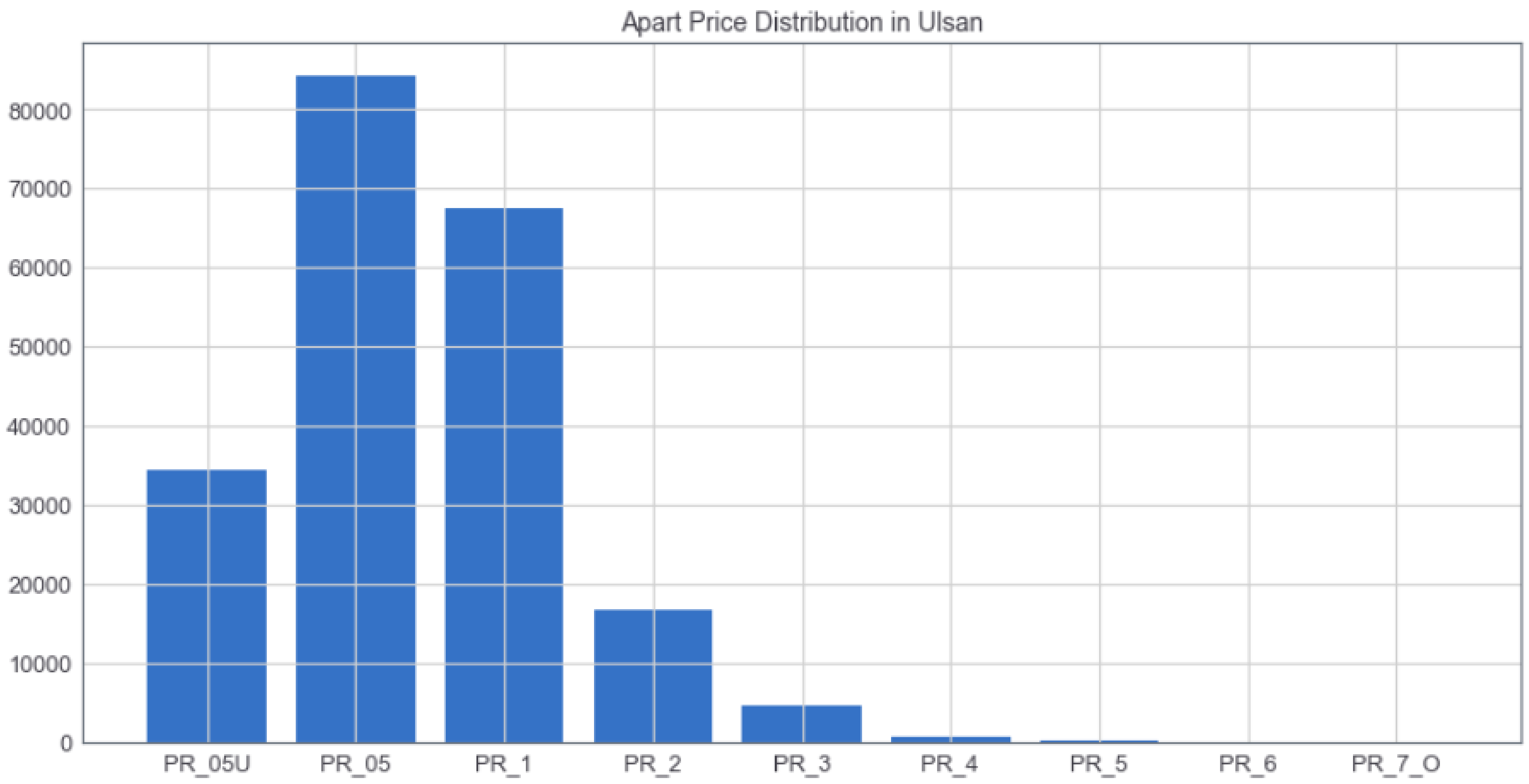
Out[4]:

	latitude	longitude	SB_score	CLSS	SB_worker_score	PR_per_PY
0	35.380959	129.341694	29.796767	3	11.287944	1.879607e+06
1	35.383626	129.345041	78.290981	4	3.440323	2.439039e+06
2	35.401375	129.288085	106.704431	4	0.932416	2.319935e+06
3	35.404102	129.285927	61.260468	2	5.867073	3.380989e+06
4	35.404126	129.283725	64.346015	4	3.628084	2.211868e+06
5	35.404138	129.282624	46.157984	4	8.293141	3.271311e+06
6	35.404149	129.281523	40.445864	5	2.635290	3.263792e+06
7	35.406853	129.281566	105.559989	4	15.690818	2.302905e+06
8	35.406865	129.280465	150.039112	3	3.578586	2.448231e+06
9	35.409534	129.283811	38.161393	5	1.162007	2.831749e+06

Use these four features for regression analysis.

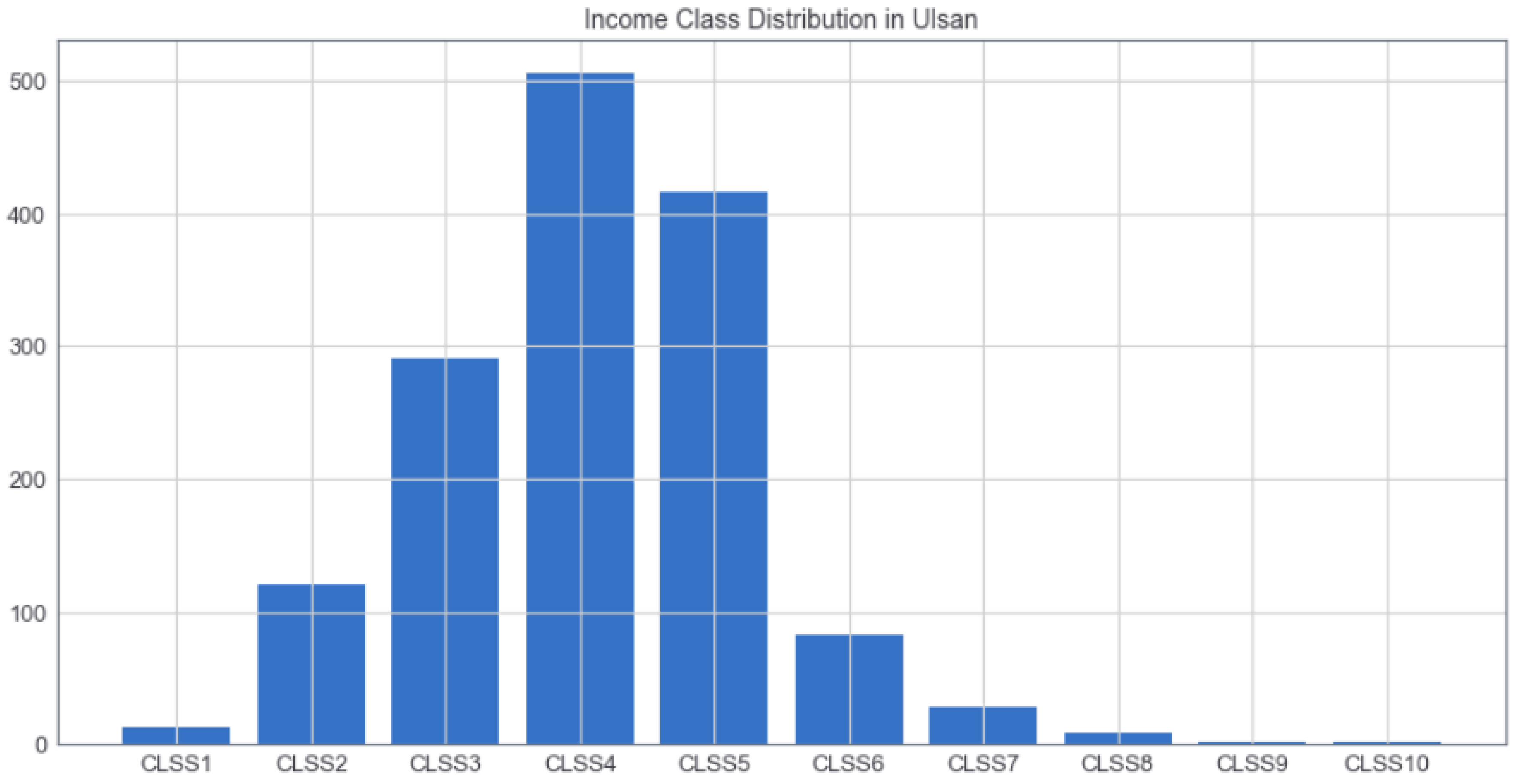
Brief Visualization

Apart Price Distribution(whole Ulsan)



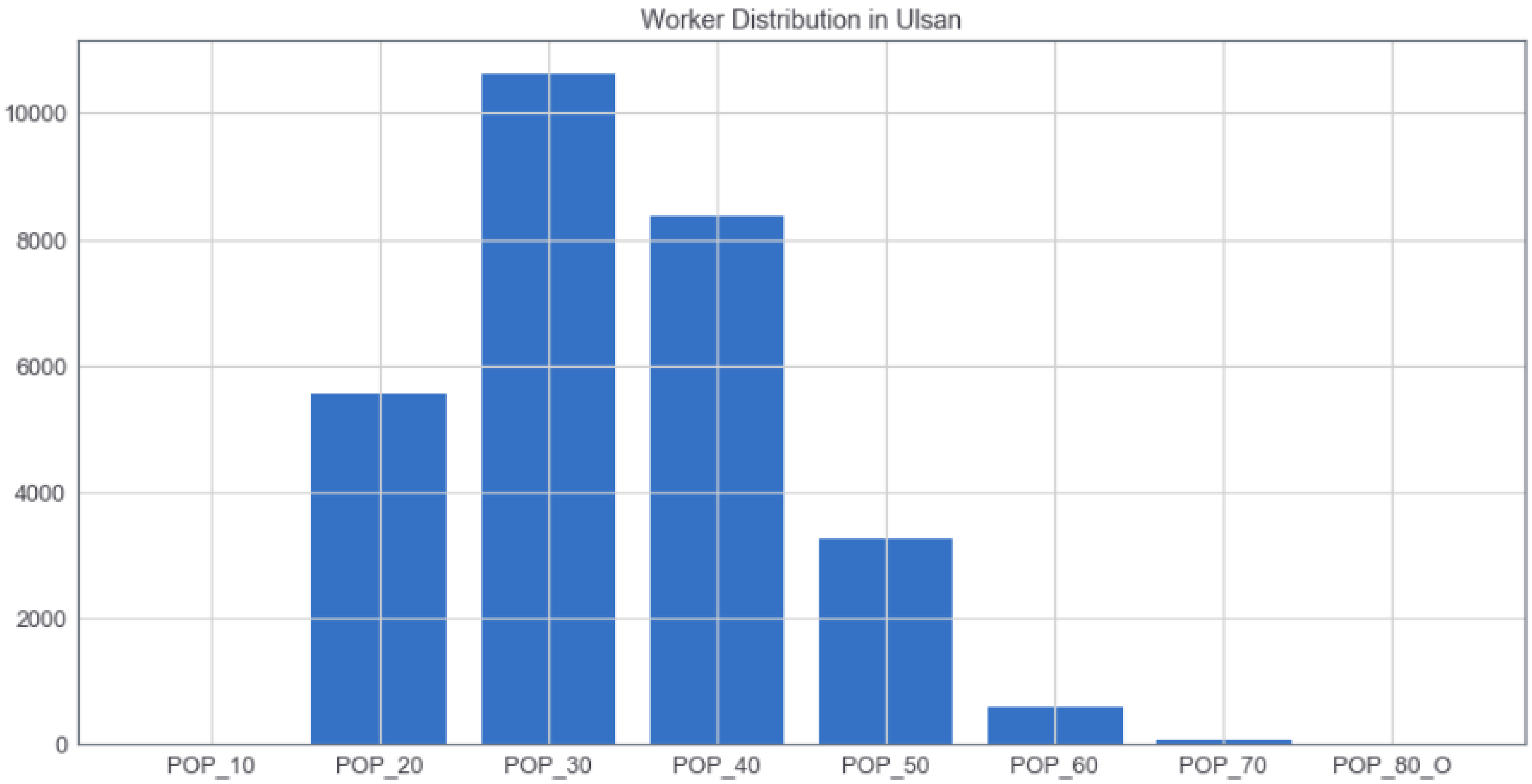
Brief Visualization

Income Class Distribution



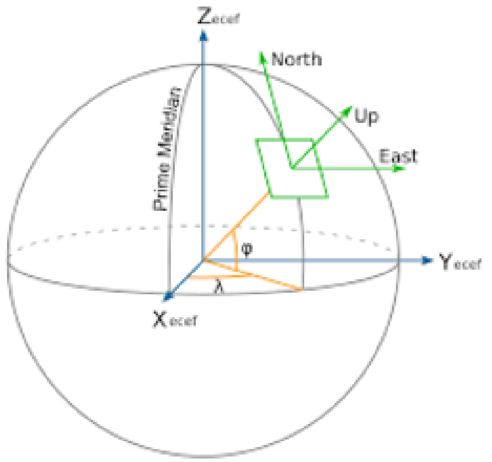
Brief Visualization

Worker Distribution



Calculating Distance between two points

-Assumption : The Earth is “sphere”



Distance

This uses the ‘**haversine**’ formula to calculate the great-circle distance between two points – that is, the shortest distance over the earth’s surface – giving an ‘as-the-crow-flies’ distance between the points (ignoring any hills they fly over, of course!).

Haversine formula:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where φ is latitude, λ is longitude, R is earth’s radius (mean radius = 6,371km);
note that angles need to be in radians to pass to trig functions!

Nearest Distance to STARBUCKS

Calculating nearest distance to STARBUCKS

Length Calculating Function between two point

In [13]:

```
def get_length(lat1,lon1,lat2,lon2):
    R = 6371e3
    lat1 *= np.pi / 180
    lon1 *= np.pi / 180
    lat2 *= np.pi / 180
    lon2 *= np.pi / 180
    d_lat = lat2 - lat1
    d_lon = lon2 - lon1
    a = np.sin(d_lat/2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(d_lon/2) ** 2
    c = 2 * np.arctan2(a**0.5, (1-a) ** 0.5)
    d = R * c
    return d
```

In [21]:

```
final_data = final_data.sort_values(by = ['distance_to_SB'])
final_data.head()
```

Out[21]:

	PR_per_PY	latitude	longitude	SB_score	CLSS	SB_worker_score	distance_to_SB
319	4.753507e+06	35.537216	129.313418	37.268609	4	7.587481	18.845226
341	5.387504e+06	35.539665	129.336618	91.735068	6	15.203293	23.895260
295	9.863784e+06	35.535664	129.290234	9.856892	4	0.626328	55.361825
445	3.807190e+06	35.550001	129.298185	200.612922	5	0.951267	62.919465
258	6.450355e+06	35.532734	129.311139	36.182702	7	2.805681	78.872308

Y_labeling with criteria = 300

Binary Labeling : 0 or 1 (pass or fail)

In [21]:
```
# 300m is affordable when considering go to walk
criteria = 300
```

In [22]:
```
final_data['y_label'] = final_data['distance_to_SB'].apply(lambda x : int(x < criteria))
final_data.head()
```

Out[22]:

	PR_per_PY	latitude	longitude	SB_score	CLSS	SB_worker_score	distance_to_SB	y_label
319	4.753507e+06	35.537216	129.313418	37.268609	4	7.587481	18.845226	1
341	5.387504e+06	35.539665	129.336618	91.735068	6	15.203293	23.895260	1
295	9.863784e+06	35.535664	129.290234	9.856892	4	0.626328	55.361825	1
445	3.807190e+06	35.550001	129.298185	200.612922	5	0.951267	62.919465	1
258	6.450355e+06	35.532734	129.311139	36.182702	7	2.805681	78.872308	1

In [23]:
```
len(final_data[final_data['y_label'] == 1])
```

Out[23]: 50

In [24]:
```
len(final_data[final_data['y_label'] == 0])
```

Out[24]: 660

There are fifty data for y=1 and 660 for y=0

Y_labeling with criteria = 500

Binary Labeling : 0 or 1 (pass or fail)

In [23]:
```
# 300m is affordable when considering go to walk
criteria = 500
```

In [24]:
```
final_data['y_label'] = final_data['distance_to_SB'].apply(lambda x : int(x < criteria))
final_data.head()
```

Out[24]:

	PR_per_PY	latitude	longitude	SB_score	CLSS	SB_worker_score	distance_to_SB	y_label
319	4.753507e+06	35.537216	129.313418	37.268609	4	7.587481	18.845226	1
341	5.387504e+06	35.539665	129.336618	91.735068	6	15.203293	23.895260	1
295	9.863784e+06	35.535664	129.290234	9.856892	4	0.626328	55.361825	1
445	3.807190e+06	35.550001	129.298185	200.612922	5	0.951267	62.919465	1
258	6.450355e+06	35.532734	129.311139	36.182702	7	2.805681	78.872308	1

In [25]:
```
final_data = final_data.drop(['distance_to_SB'], axis=1)
```

In [26]:
```
len(final_data[final_data['y_label'] == 1])
```

Out[26]: 111

In [27]:
```
len(final_data[final_data['y_label'] == 0])
```

Out[27]: 599

I choose this criteria.
Here labeling is imbalanced.

Final Y-Labeled Data

Binary Labeling without PCA

In [57]:
```
final_data = final_data[col]
final_data = final_data.reindex(list(range(len(final_data))))
final_data.head()
```

Out[57]:

	latitude	longitude	SB_score	CLSS	SB_worker_score	PR_per_PY	y_label
0	35.380959	129.341694	29.796767	3	11.287944	1.879607e+06	0
1	35.383626	129.345041	78.290981	4	3.440323	2.439039e+06	0
2	35.401375	129.288085	106.704431	4	0.932416	2.319935e+06	0
3	35.404102	129.285927	61.260468	2	5.867073	3.380989e+06	0
4	35.404126	129.283725	64.346015	4	3.628084	2.211868e+06	0

Binary Labeling with PCA

In [81]:
```
pc_data = pd.concat([final_data[col[0:2]], score_df, final_data[col[-1]]], axis=1)
pc_data.head()
```

Out[81]:

	latitude	longitude	PC 1	PC 2	PC 3	PC 4	y_label
0	35.380959	129.341694	-0.179099	-1.538754	-0.910998	0.066271	0
1	35.383626	129.345041	-0.346197	-0.674096	-0.080136	-0.705045	0
2	35.401375	129.288085	-0.649273	-0.341075	-0.126003	-1.106231	0
3	35.404102	129.285927	-1.000125	-0.567735	-1.262804	0.661681	0
4	35.404126	129.283725	-0.243787	-0.964072	-0.085540	-0.617812	0

Using R for Logistic Regression model fitting

Case I. Normalized Data with *imbalanced* training data set

```
x <- strata(c("y_label"), size=c(500, 100), method="srswor", data=data)
train = getdata(data,x)
```

Case II. Normalized Data with *balanced* training data set

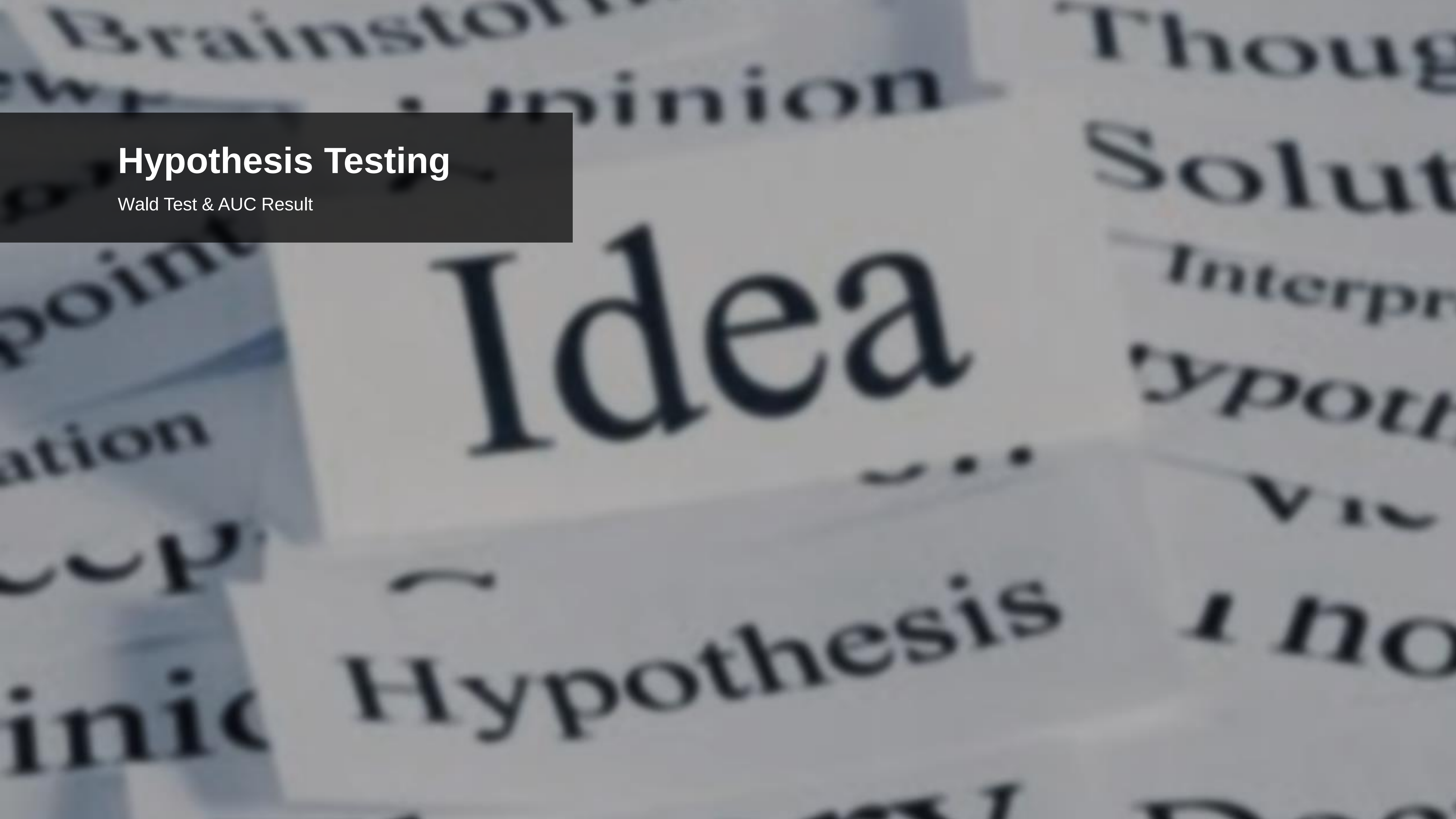
```
x <- strata(c("y_label"), size=c(100, 100), method="srswor", data=data)
train = getdata(data,x)
```

Case III. PCA Data with *imbalanced* training data set

Case IV. PCA Data with *balanced* training data set

Hypothesis Testing

Wald Test & AUC Result



Hypothesis Testing : Wald Test for Evaluation

Logistic Regression with 1 Predictor

- β_0, β are unknown parameters and must be estimated using maximum likelihood estimation
- Primary interest in estimating and testing hypotheses regarding β
- $\boxed{\text{Large-Sample test (Wald Test):}}$
  - $H_0: \beta = 0$ vs. $H_A: \beta \neq 0$
  - Test statistic (TS): $X^2_{obs} = \left(\frac{\hat{\beta}}{\hat{\sigma}_{\hat{\beta}}}\right)^2$
  - Rejected region (RR): $X^2_{obs} \geq \chi^2_{\alpha,1}$
  - p value: $P(\chi^2 \geq X^2_{obs})$

Hypothesis Testing : Case Introduction

Select Best Performance(AUC) Model

Case I. Normalized Data with *imbalanced* training data set

```
x <- strata(c("y_label"), size=c(500, 100), method="srswor", data=data)
train = getdata(data,x)
```

Case II. Normalized Data with *balanced* training data set

```
x <- strata(c("y_label"), size=c(100, 100), method="srswor", data=data)
train = getdata(data,x)
```

Case III. PCA Data with *imbalanced* training data set

Case IV. PCA Data with *balanced* training data set

Hypothesis Testing : Evaluation Result

Case II. Normalized Data with *balanced* training data set

Hide

```
x <- strata(c("y_label"), size=c(100, 100), method="srswor", data=data)
train = getdata(data,x)
```

Hide

```
model <- glm(y_label ~ SB_score + CLSS + SB_worker_score + PR_per_PY, data = train, family = "binomial")
summary(model)
```

```
Call:
glm(formula = y_label ~ SB_score + CLSS + SB_worker_score + PR_per_PY,
    family = "binomial", data = train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.41853  -0.98154  -0.01023   1.02208   1.86395

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     -0.08427    0.15706  -0.537 0.591565
SB_score        -0.46410    0.16838  -2.756 0.005848 **
CLSS             0.57305    0.17129   3.345 0.000821 ***
SB_worker_score  0.59607    0.26650   2.237 0.025308 *
PR_per_PY        0.35062    0.16486   2.127 0.033439 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 277.26  on 199  degrees of freedom
Residual deviance: 240.85  on 195  degrees of freedom
AIC: 250.85

Number of Fisher Scoring iterations: 4
```

Hypothesis Testing : Wald Test Result (ANOVA Table)

Case II. Normalized Data with *balanced* training data set

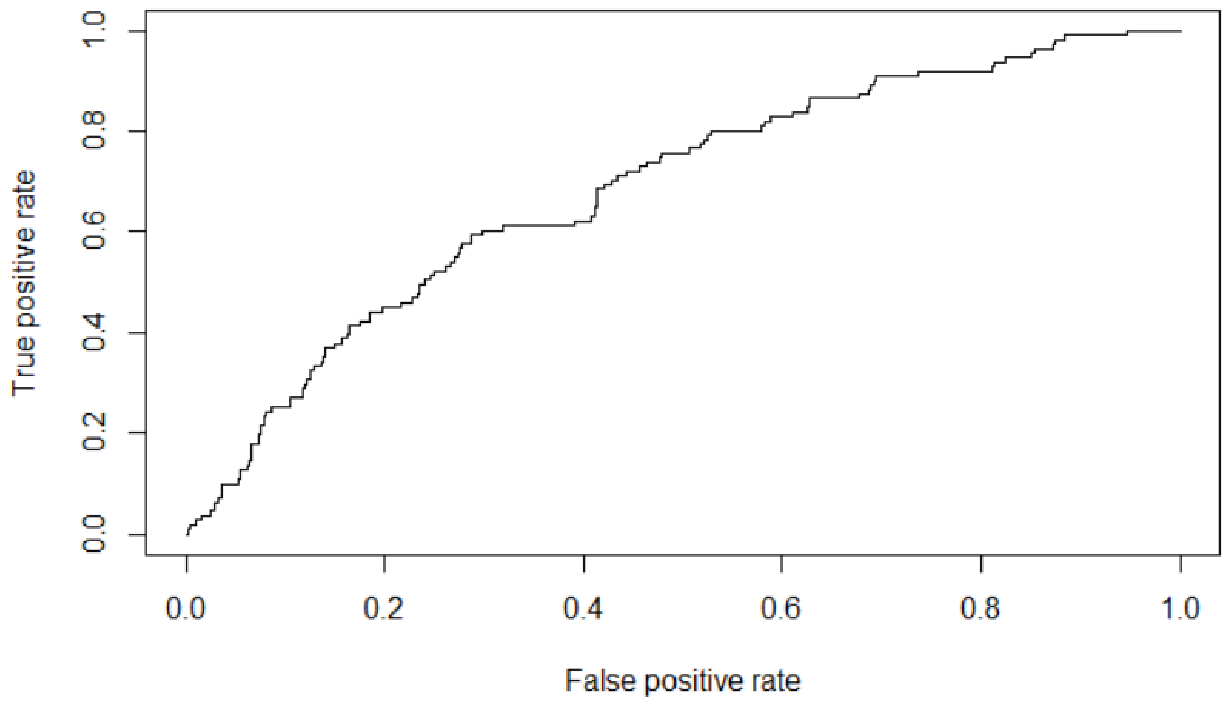
Hide

```
anova(model, test="Chisq")
```

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: y_label

Terms added sequentially (first to last)

                Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                              199     277.26
SB_score         1   8.7526       198     268.51  0.003092 **
CLSS             1  15.6162       197     252.89 7.759e-05 ***
SB_worker_score  1   7.3851       196     245.50  0.006577 **
PR_per_PY        1   4.6551       195     240.85  0.030962 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Hypothesis Testing : AUC-ROC Curve Result

Case II. Normalized Data with *balanced* training data set



Hide

```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
[1] 0.6857826
```

Summary and Conclusion

Result & Visualization

Conclusion : Feature Significance Result

```
                  Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                                199     277.26
SB_score           1   8.7526       198     268.51  0.003092 **
CLSS               1  15.6162       197     252.89 7.759e-05 ***
SB_worker_score    1   7.3851       196     245.50  0.006577 **
PR_per_PY          1   4.6551       195     240.85  0.030962 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     -0.08427    0.15706  -0.537 0.591565
SB_score        -0.46410    0.16838  -2.756 0.005848 **
CLSS             0.57305    0.17129   3.345 0.000821 ***
SB_worker_score  0.59607    0.26650   2.237 0.025308 *
PR_per_PY        0.35062    0.16486   2.127 0.033439 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Coefficients : Office Worker Score > Income Class > Land Price > Population Score

(Negatively Related)

Conclusion : Probability Formula for Model

	Df	Deviance	Resid. Df	Resid. Dev	Pr(>Chi)	
NULL			199	277.26		
SB_score	1	8.7526	198	268.51	0.003092	**
CLSS	1	15.6162	197	252.89	7.759e-05	***
SB_worker_score	1	7.3851	196	245.50	0.006577	**
PR_per_PY	1	4.6551	195	240.85	0.030962	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Coefficients:

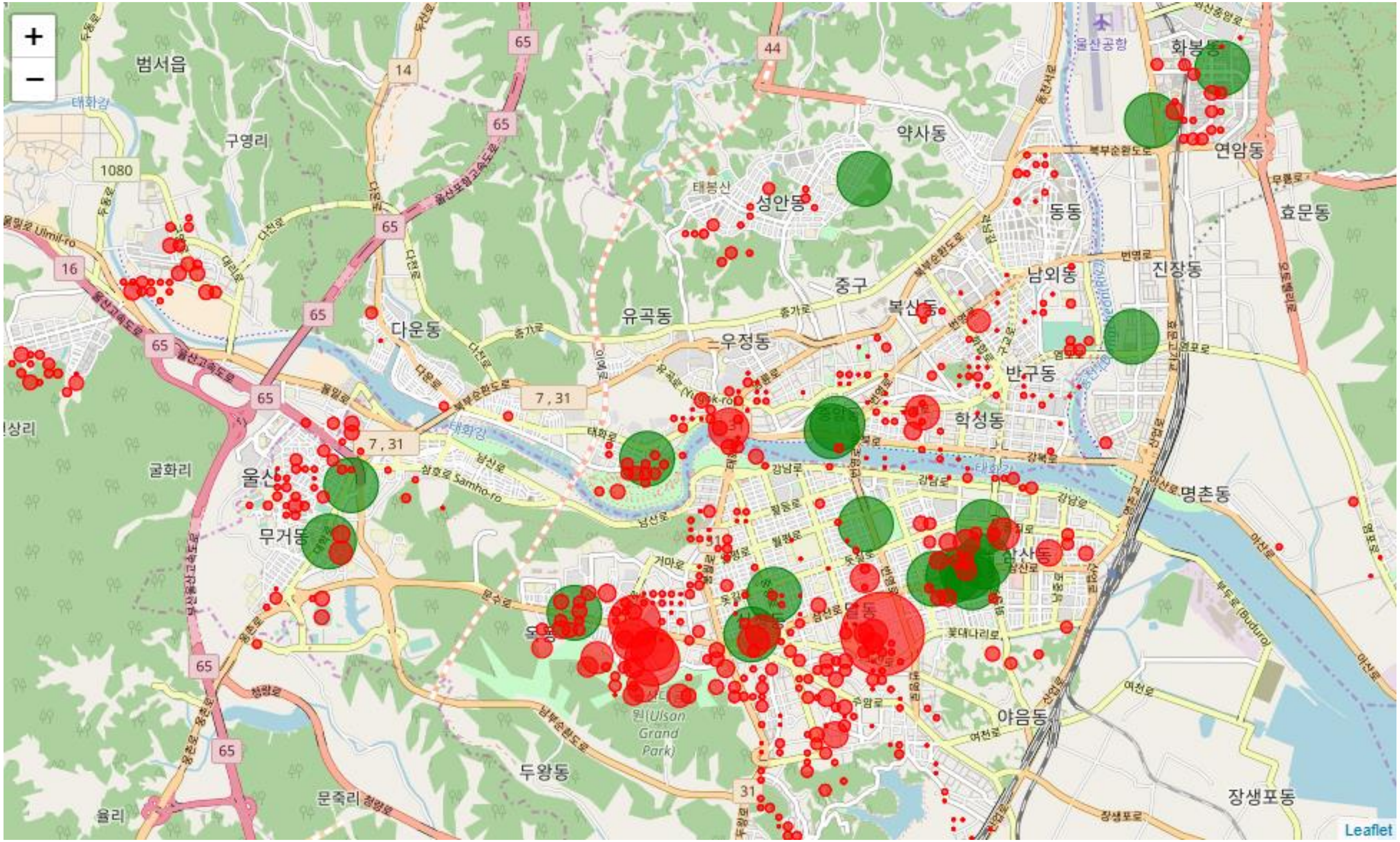
	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-0.08427	0.15706	-0.537	0.591565	
SB_score	-0.46410	0.16838	-2.756	0.005848	**
CLSS	0.57305	0.17129	3.345	0.000821	***
SB_worker_score	0.59607	0.26650	2.237	0.025308	*
PR_per_PY	0.35062	0.16486	2.127	0.033439	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

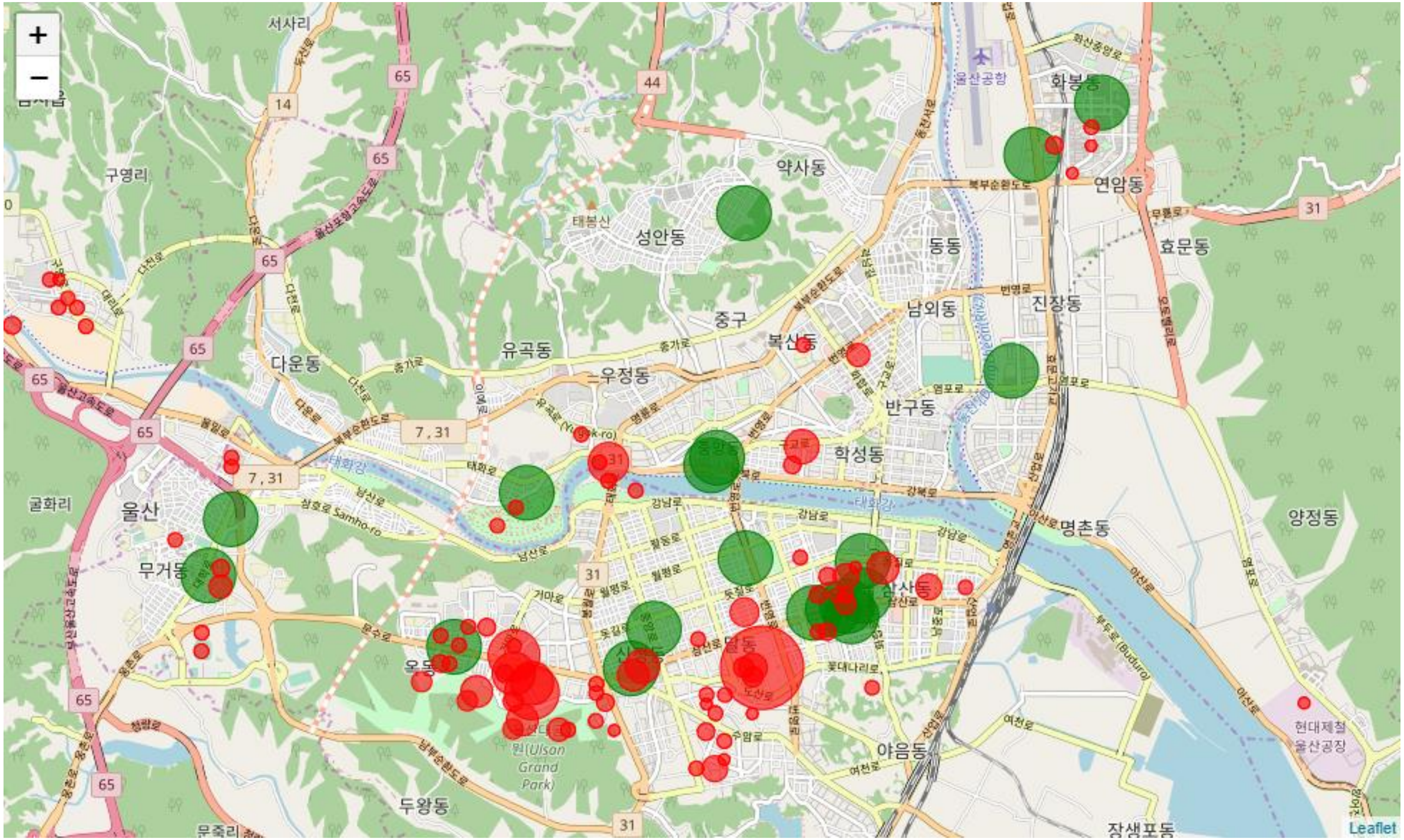
$$P(y=1) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

$$P(y=1) = \frac{e^{-0.08427-0.4610X_1+0.57305X_2+0.59607X_3+0.35062X_4}}{1+e^{-0.08427-0.4610X_1+0.57305X_2+0.59607X_3+0.35062X_4}}$$

Visualization : Plot Probability Result



Visualization : Top 100 Probability Spot



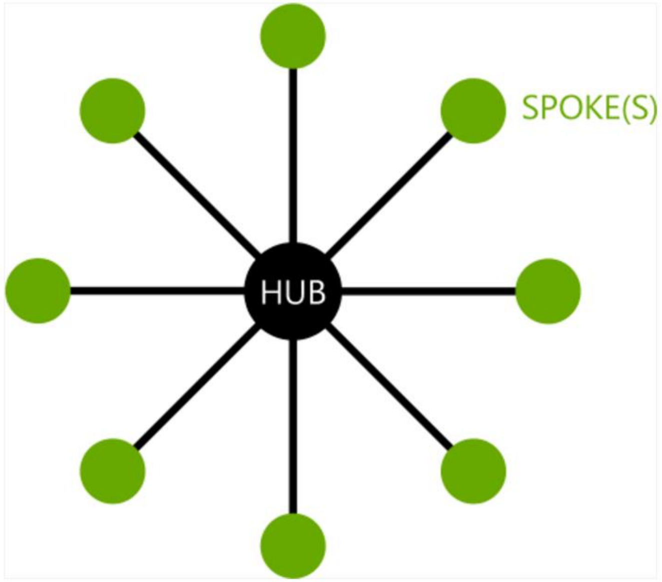
Other Trial : Hub & Spoke Strategy

Add ‘nearest distance to Starbucks’ as a feature to mimic the Hub & Spoke Strategy

> 허브 & 스포크 전략

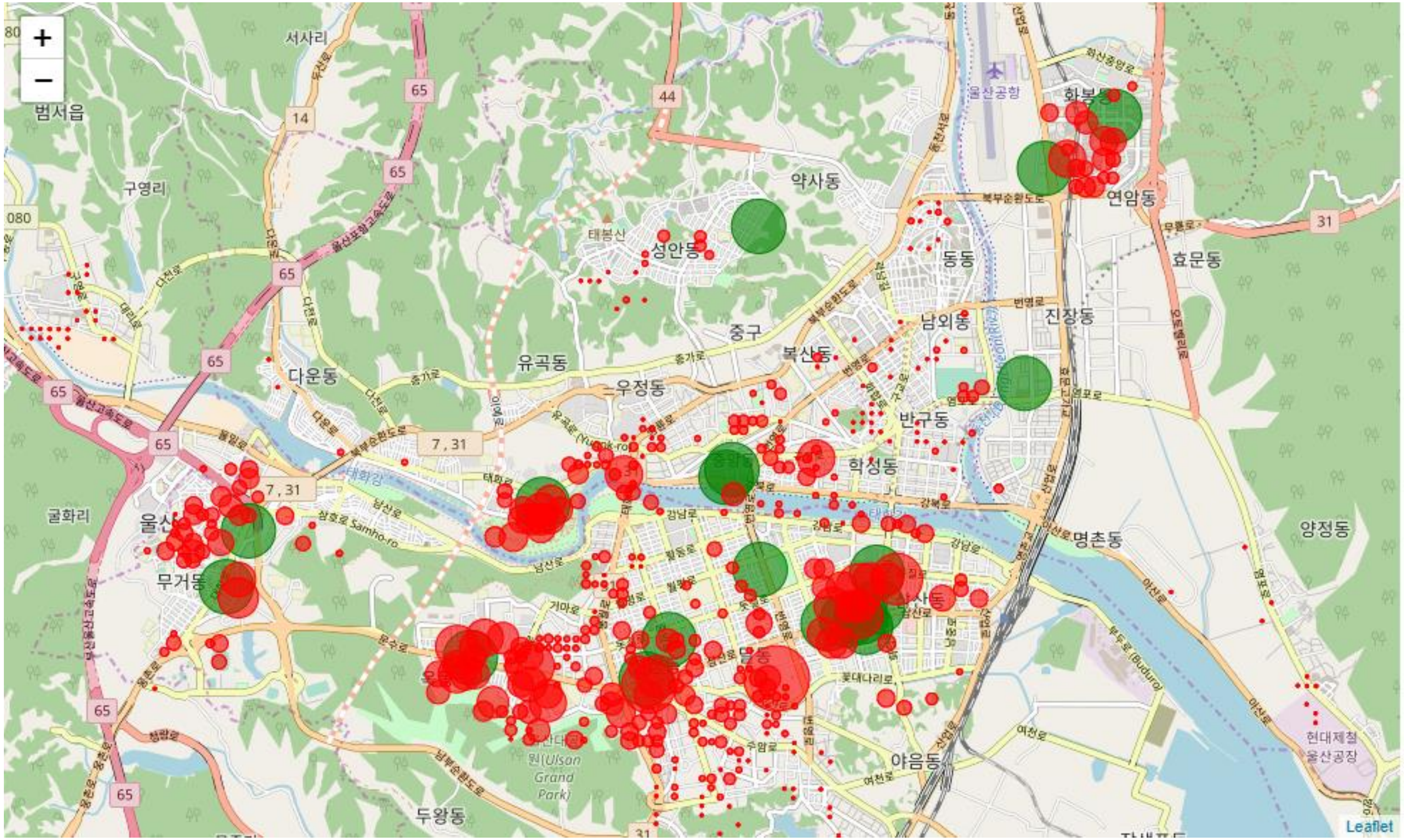
스타벅스는 출점전략에서도 창조적 길을 걸었다. 허브앤스포크(Hub & Spoke) 전략은 자전거 바퀴에서 비롯됐다. 자전거 바퀴를 보면 한가운데 허브(Hub)가 있고 바퀴까지 가느다란 바퀴살(Spoke)이 펼쳐져 있다.

허브에서 뻗어 나온 에너지가 부채살처럼 바퀴에 전달된다. 원형의 영향력이 형성되며 전진하는 것이다. 다른 말로 표현하자면 클러스터(Cluster) 전략이다. 한 지역을 석권하듯 집중적으로 매장을 출점한 후에 인근 지역과 인접 위성도시로 진출하는 것이다.



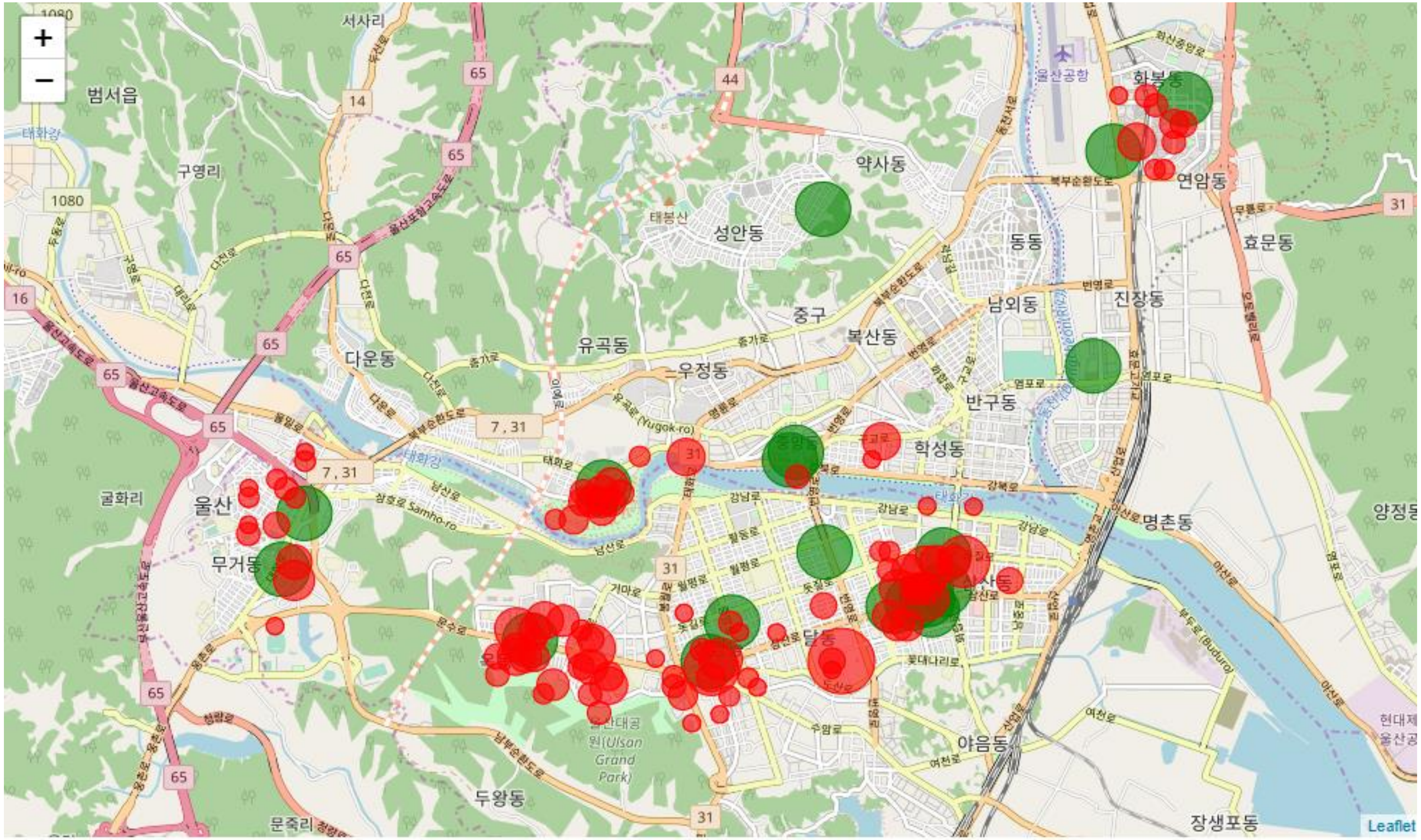
Other Trial : Hub & Spoke Strategy

Hub & Spoke Strategy Result



Other Trial : Hub & Spoke Strategy

Hub & Spoke Strategy Top 100 Spots Results



Limitation & Further Development

Problems and Suggestions

Floating Population Measure Failure

Failure of Traffic Data Construction with Node_Link Data

In [218]:
```
total_car = 0
total_bus = 0
missing_car = 0
missing_bus = 0
for idx,row in data.iterrows():
    total_car += row['Car'] * 2
    total_bus += row['Bus'] * 2

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Car']) > 0 ): # 노드에 있는 도로의 경우
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Car'] += row['Car'] # 자동차 대수를 증가 시킵니다.
    else:
        missing_car += row['Car'] # 노드에 없는 이름인 경우 missing value 를 증가 시킵니다.

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Car']) > 0 ):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Car'] += row['Car']
    else:
        missing_car += row['Car']

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Bus']) > 0):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['시작지점'],'Bus'] += row['Bus']
    else:
        missing_bus += row['Bus']

    if(len(ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Bus']) > 0):
        ulsan_node.loc[ulsan_node['NODE_NAME'] == row['종료지점'],'Bus'] += row['Bus']
    else:
        missing_bus += row['Bus']
```

In [220]:
```
missing_car / total_car
```

Out[220]: 0.7124665727606904

In [221]:
```
missing_bus / total_bus
```

Out[221]: 0.7306401211899067

Floating Population Measure Failure

Can not access to dataset like below

SKT시연용

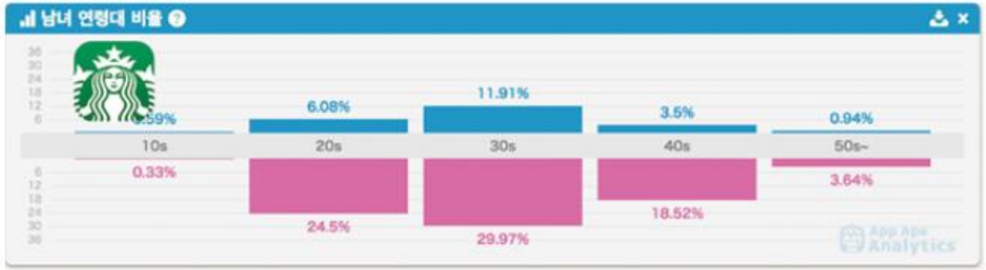
Geovision 상권분석

■ 본 보고서에 사용 된 데이터 및 업데이트 주기

DB 제공사	DB 내역	DB 특징	업데이트 주기
SK 텔레콤	유동인구	기지국 통화량 통계분석 (시간, 성, 연령)	월별
	주간상주인구	건물데이터와 통화량 통계 분석	1년
	주거인구	통계청 센서스와 행정안전부 주민등록 통계 활용	1년
SK 플래닛	지도	지도 및 시설정보(POI)	3개월
현대카드	업종 매출	업종별 카드 가맹점 매출 통계	월별
부동산 114	부동산 시세/매물	부동산 시세 및 매물 정보(아파트, 상가)	월별
	개발 예정 정보	개발 예정 구역 및 정보	수시

STARBUCKS Score for Population : Why negatively related?

```
                Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                              199     277.26
SB_score         1   8.7526       198     268.51  0.003092 **
CLSS             1  15.6162       197     252.89 7.759e-05 ***
SB_worker_score  1   7.3851       196     245.50  0.006577 **
PR_per_PY        1   4.6551       195     240.85  0.030962 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     -0.08427    0.15706  -0.537 0.591565
SB_score        -0.46410    0.16838  -2.756 0.005848 **
CLSS             0.57305    0.17129   3.345 0.000821 ***
SB_worker_score  0.59607    0.26650   2.237 0.025308 *
PR_per_PY        0.35062    0.16486   2.127 0.033439 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



남녀 연령대 비율: 조사 대상 기간 중 앱 설치 사용자의 남녀 연령대 분포입니다.
[데이터 출처 : App Ape Analytics (해당 기사를 위해 한국 약 2만대의 안드로이드 단말기를 분석) / 액티브 수는 App Ape 추정치]

We expect that this score will be important.
However it is negatively related…
We don't know why…

Recommendation for Virtual Client

Business Location Analysis Consulting Product

이디야커피의 초기 입점전략..스타벅스 옆에 매장 오픈
커피 소비 많은 장소 손쉽게 찾아..저렴한 가격으로 소비자 빼앗아오기도

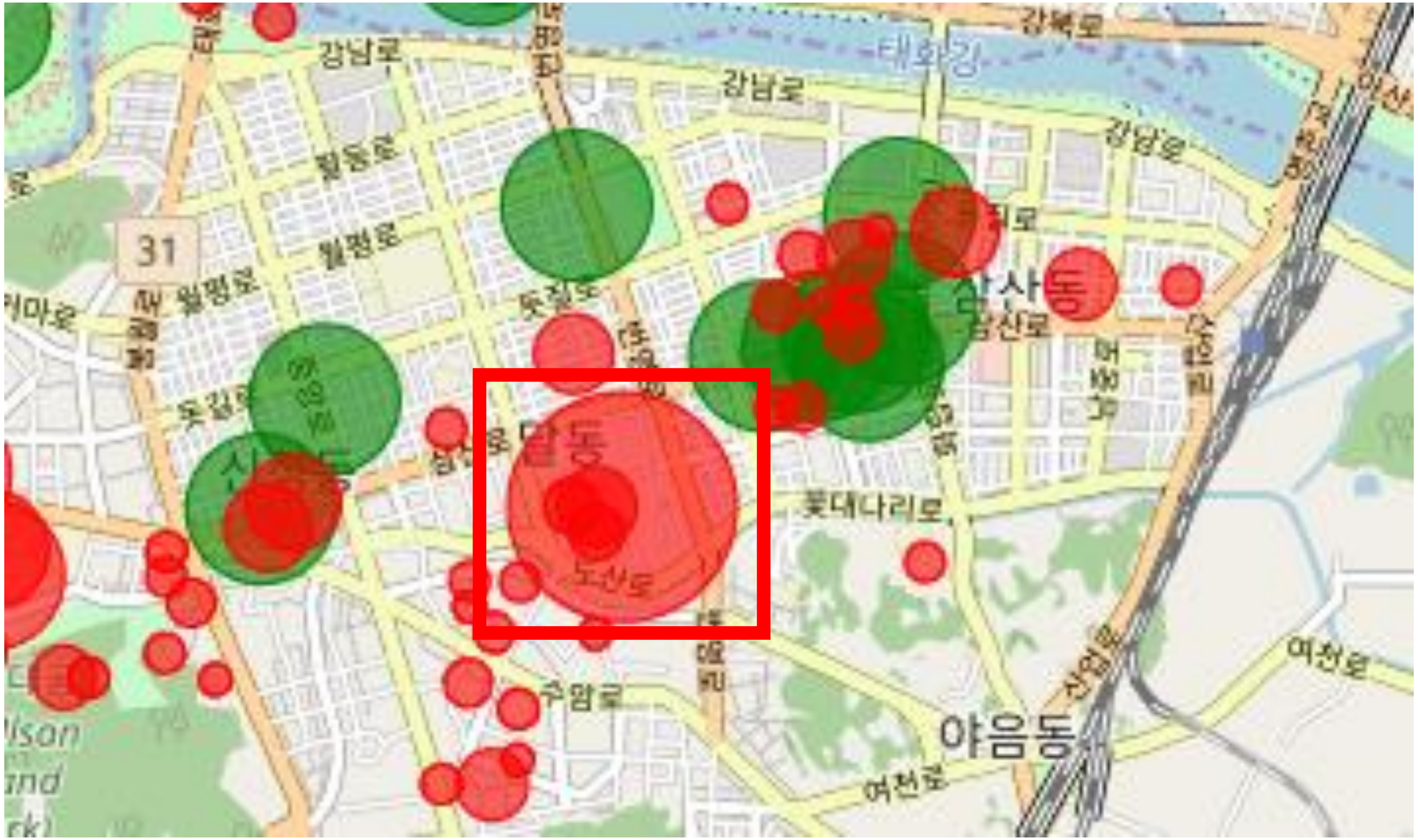
[이데일리 함정선 기자] '스타벅스 옆 이디야' 저가 커피의 '대명사'로 불리는 이디야커피의 초기 입점 전략이 눈길을 끌고 있다.

이디야는 초기 스타벅스 옆 자리를 사수하는 전략을 통해 지난해 대기업과 해외 유명 커피 전문점들을 제치고 커피전문점 만족도 1위를 거머쥐는 쾌거를 이뤘다. 사실상 이디아가 살아남기 위해 기초 체력을 만들기까지 스타벅스의 도움이 없었다면 더 많은 시간이 걸렸을 것이라고 전문가들은 평가하고 있다.

말 그대로 스타벅스 바로 옆 매장을 노린 것이다. 만약 스타벅스가 입점한 곳 바로 옆자리 월세가 비싸다면 뒷골목이나 옆 골목 등을 공략하는 '서브 스트리트' 전략을 썼다.

실제로 서울 명동이나 강남 테헤란로 등에서는 스타벅스 바로 옆에 자리를 잡은 이디야 매장을 발견하는 것이 어렵지 않다. 바로 옆 매장이 아니더라도 스타벅스 근처에서 손쉽게 이디야 매장을 찾을 수 있다.

Providing Intuitive Information to Client



We could recommend this spot to our virtual customers

Thank you