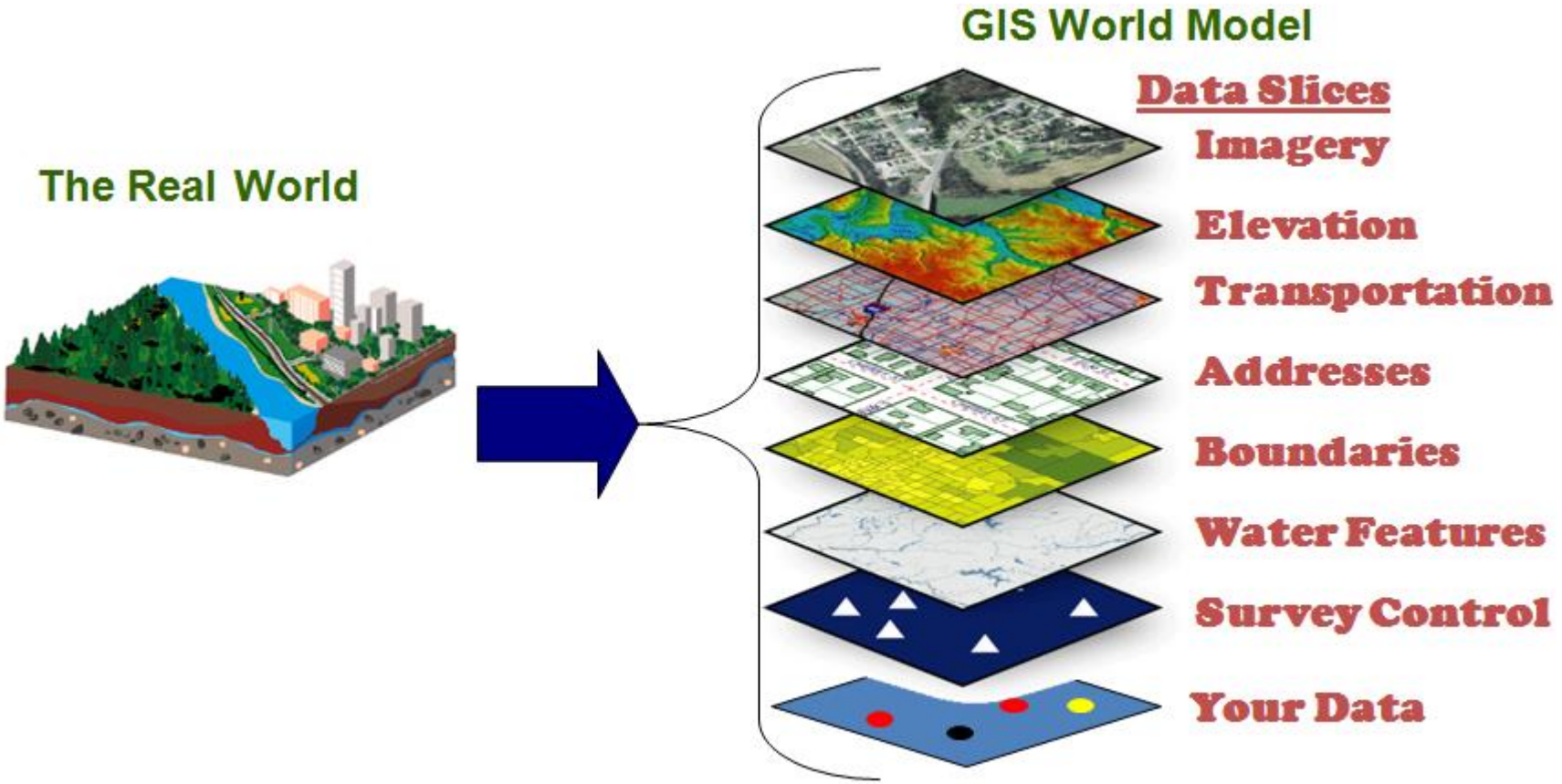
Open-source GIS Tools

-실습-

PATHFINDING AND GRAPH SEARCH ALGORITHMS
WITH OSMNX

GIS(Geographic Information System)



QGIS

ArcGIS

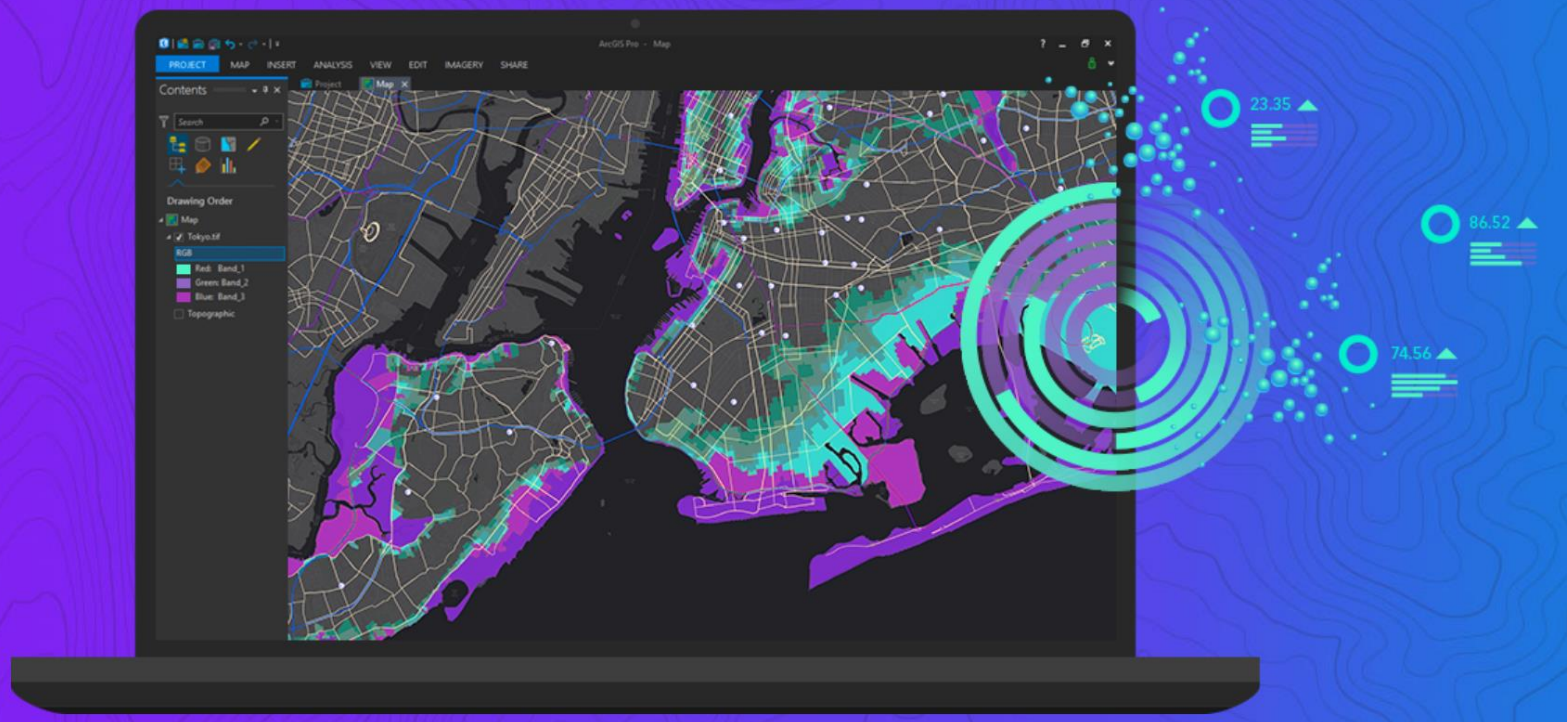
https://www.admitnetwork.org/work-packages/gis-gps/

다수의 GIS 도구는 Closed 소스이며 Tabular 기반 데이터모델

ArcGIS

매핑 및 공간 분석 플랫폼

ArcGIS는 Esri 공간정보 클라우드의 핵심입니다



OSMNX: OpenStreetMap + NetworkX

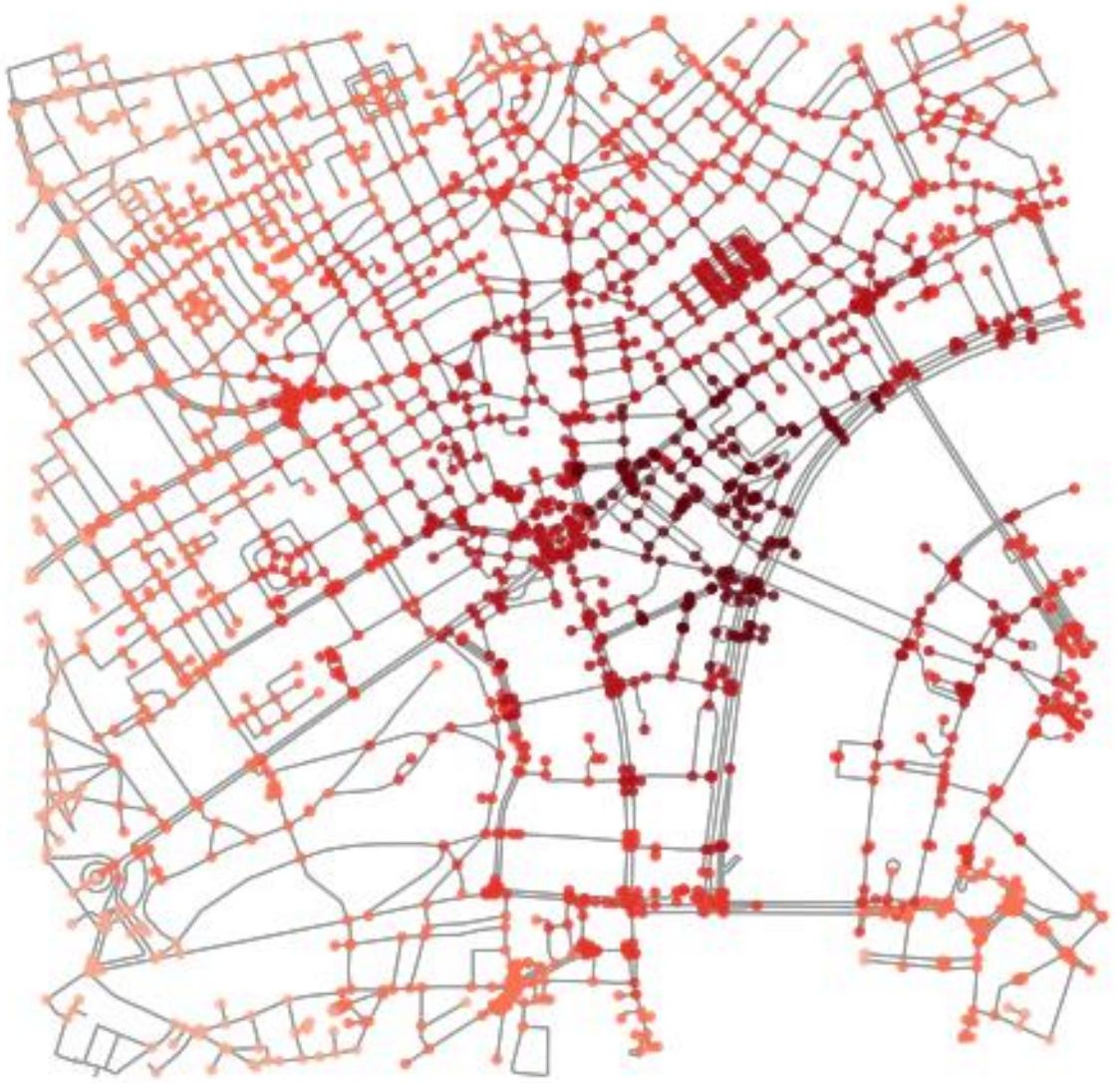
- 주어진 지도를
  그래프 모델로 모델링
  - Nodes: 교차점
  - Edges: 도로

- OSMNX는
  도로네트워크 레이어만
  제공하는 한계점을 가짐

Singapore's Raffles Place MRT

OSMNX: OpenStreetMap + NetworkX

- 왜 좋은가?
  - 도시 계획/문제점 분석 시 매우 유용함
  - 노드/엣지 중심으로 분석
    - 데이터 수집/매핑/계산/분석을 노드(OR 노드 그룹) 위에서 수행
  - 예) 해운대구 물류망 분석
    - 노드: 해운대구 내 소속되는 노드들
    - 엣지: 해당 노드들에 관련된 엣지
    - 가중치에 대한 고려
      - 노드의 중요 값, 엣지에 대한 비용



Central London

Code: Setup

```
!pip install osmnx
!python -m pip uninstall matplotlib
!pip install matplotlib==3.1.3
```

matplotlib==3.1.3 버전에 맞춰야함

```
import networkx as nx
import osmnx as ox
import matplotlib.pyplot as plt
```

Code: Setup

1.ox.graph_from_place: 지역명
2.ox.graph_from_address: 주소정보
3.ox.graph_from_point: 위경도정보
4.ox.graph_from_bbox: 동서남북좌표

network_type: ‘walk’, ‘bike’, ‘drive’, ‘drive_se
rvice’, ‘all’, or ‘all_private’ 중 하나

```
# download/model a street network for some city then visualize it
place = "Saha-gu, Busan, Korea"
G = ox.graph_from_place(place, network_type="drive")
fig, ax = ox.plot_graph(G)
```

매우 중요한 건 G는 그래프 모델이라는 점

Codes: 도로타입 표시하기

```
# edge의 타입 따라서 도로에 대한 색깔 다르게 표시하기
hwy_colors = {'footway': 'skyblue',
              'residential': 'paleturquoise',
              'cycleway’: lightgreen',
              'service': 'sienna',
              'living street': ’orange',
              'secondary': 'black',
              'pedestrian': 'lightskyblue'}
```

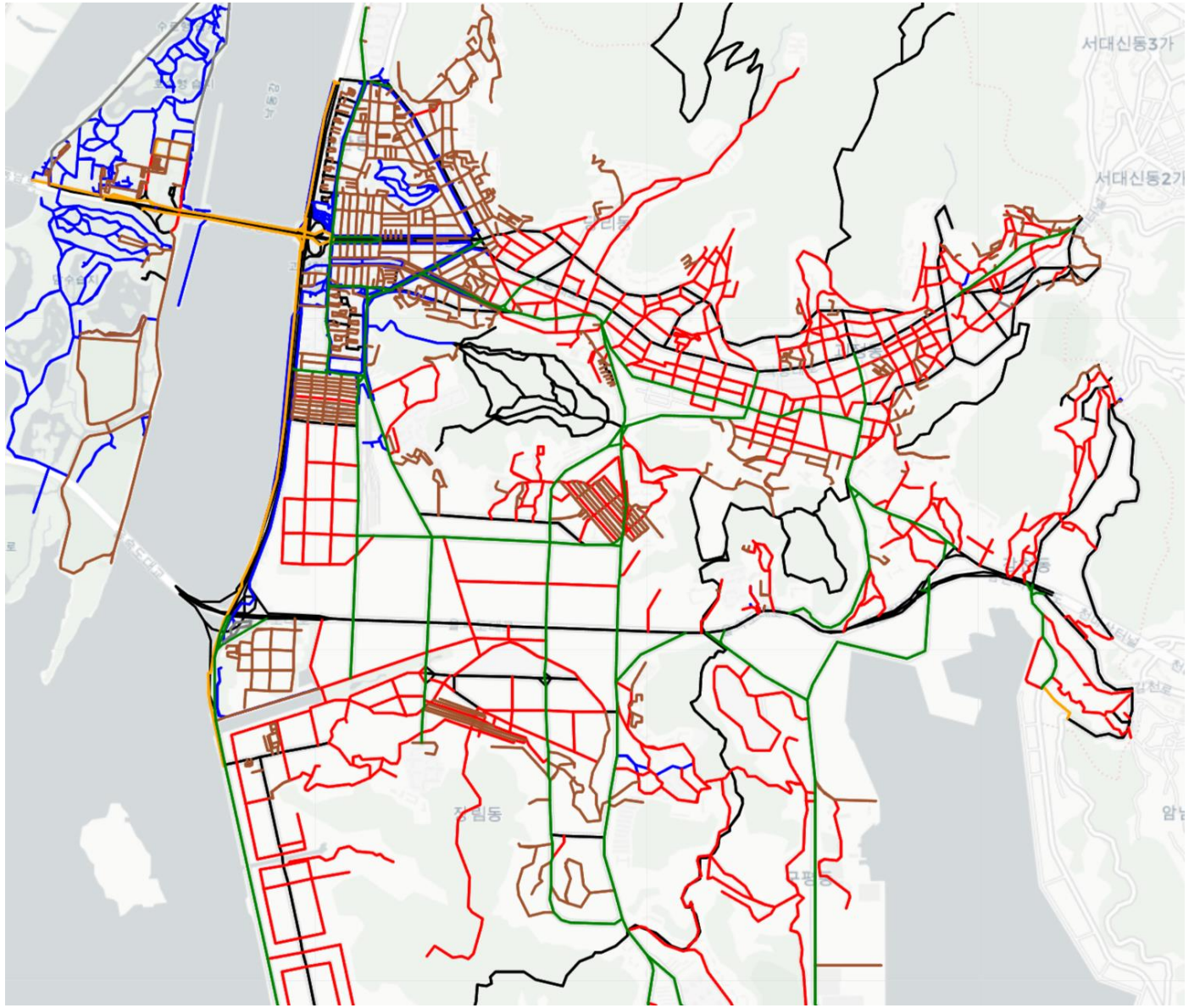
Codes: 도로타입 표시하기

```
# 엣지를 탐색
def find_edges(G, hwys):
    edges = []
    for u, v, k, data in G.edges(keys=True, data='highway'):
        check1 = isinstance(data, str) and data not in hwys
        check2 = isinstance(data, list) and all([d not in hwys for d in data])
        if check1 or check2:
            edges.append((u, v, k))
    return set(edges)
```

Codes: 도로타입 표시하기

```
# hwy 이 정해지지 않은 경우는 black
G_tmp = G.copy()
G_tmp.remove_edges_from(G.edges - find_edges(G, hwy_colors.keys()))
m = ox.plot_graph_folium(G_tmp, popup_attribute='highway', weight=5, color='black')

# 지도 위에서 Hwy_colors에서 정해진 타입에 따라 추가적인 edge를 그림
for hwy, color in hwy_colors.items():
    G_tmp = G.copy()
    G_tmp.remove_edges_from(find_edges(G_tmp, [hwy]))
    if G_tmp.edges:
        m = ox.plot_graph_folium(G_tmp,
                                 graph_map=m,
                                 popup_attribute='highway',
                                 weight=5,
                                 color=color)
```


서대신동3가
서대신동2가

Codes: 최단거리

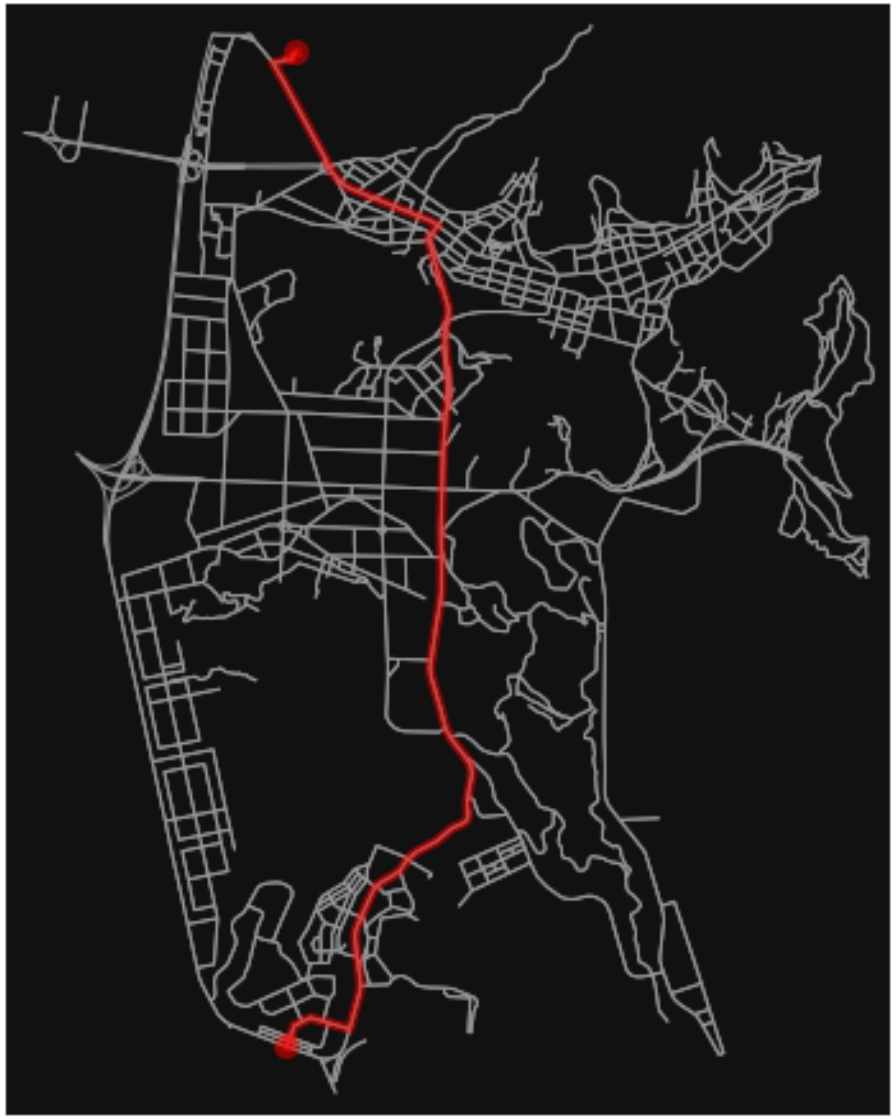
```
# impute missing edge speeds and calculate
edge travel times with the speed module
G = ox.speed.add_edge_speeds(G)
G = ox.speed.add_edge_travel_times(G)
```

Codes: 최단거리

```
# get the nearest network nodes to two lat/lng points with the distance module
orig = ox.distance.nearest_nodes(G, X=128.96755631796773, Y=35.116015941374444)
dest = ox.distance.nearest_nodes(G, X=128.96517223758627, Y=35.046698756214056)
```

Codes: 최단거리

```
# find the shortest path between nodes, minimizing travel time, then plot i
t
route = ox.shortest_path(G, orig, dest, weight="travel_time")
fig, ax = ox.plot_graph_route(G, route, node_size=0)
```

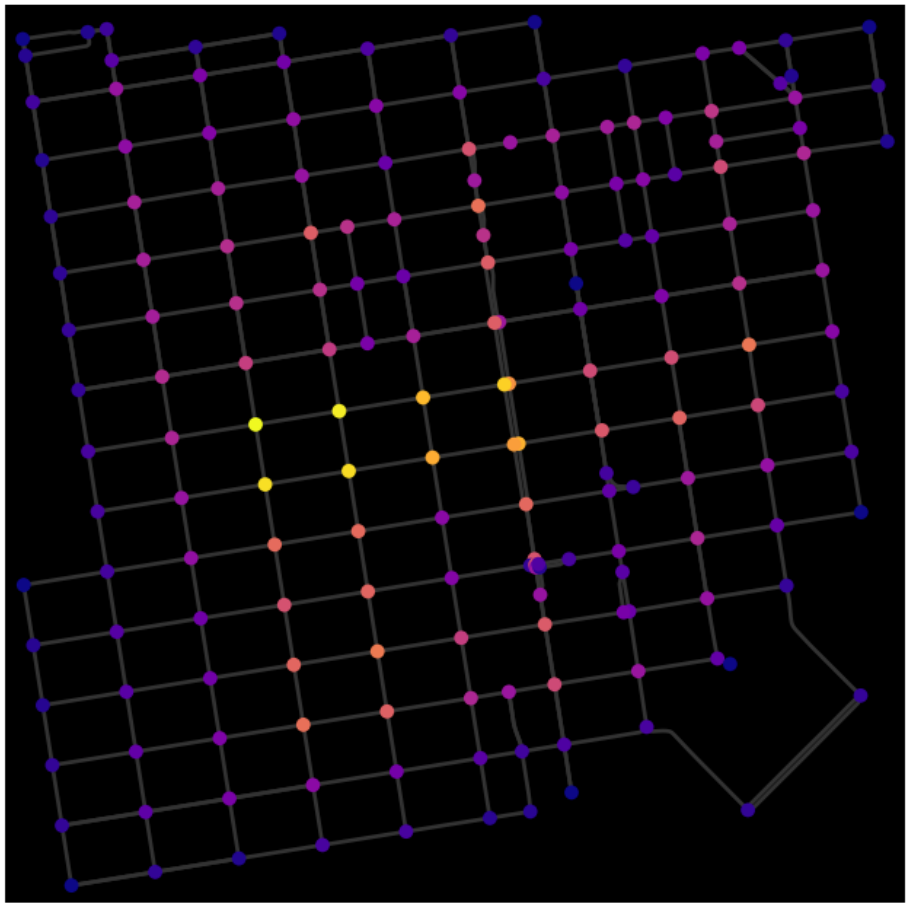


Codes: K-Shortest

```
routes  = ox.k_shortest_paths(G, orig, dest, 3, weight="travel_time")
paths = [r for r in routes]
```

```
# find the k-
shortest paths between nodes, minimizing travel time, then plot it
fig, ax = ox.plot_graph_routes(G, paths, route_colors=['r','b','c']
, route_linewidth=3)
```

그래프를 통한 도시계획 분석: 잠재 상권 분석



entertainments(V):
주위 편의시설

travel_time(E): 이동시간

traffic(V): 주위 교통혼잡

cost(V): 주위 부동산 가격

PotentialCommercialArea(G)
= w1*entertainments(V) + w2*travel_time(E) + w3*traffic(V) + w4*cost(V)

그래프를 통한 도시계획 분석: 동아대 인근

'인근'은 '이웃한 가까운 곳'을
뜻하고 비슷한말로
'가까이, 근방, 근처' 등이 있습니다

추천 원룸·오픈형 원룸
월세 100/15
26㎡·1층
사하구 하단동
ㅁ 대학로최저가 ㅁ 풀옵션방 ㅁ …

추천 원룸·오픈형 원룸
월세 300/38
19㎡·7층
사하구 하단동
🍒하단역 근처에 이런 집이🍒…

추천 원룸·오픈형 원룸
월세 200/20
29㎡·6층
사하구 하단동
✅큰대로변✅가성비갑✅큰방…

추천 원룸·분리형 원룸
월세 300/32
26㎡·1층
사하구 하단동
💟투룸급💟올리모델링💟

추천 원룸·오픈형 원룸
월세 500/40
26㎡·4층
사하구 하단동
💟큰길위치💟놀라운방💟

추천 원룸·오픈형 원룸
월세 100/18
23㎡·2층
사하구 하단동
❎가성비갑❎보증금낮게❎학…

추천 원룸·오픈형 원룸
전세 6,500
14㎡·7층
사하구 하단동
♥첫입주♥동아대인근♥전세…

그래프를 통한 도시계획 분석: 동아대 인근

1분,5분,10분, 20분, 30분내로
걷거나 뛰어갈 수 있는 지역을 찾아보자


추천 원룸·오픈형
월세 300/3
19㎡·7층
사하구 하단동
하단역 근처


추천 원룸·오픈형
월세 200/2
29㎡·6층
사하구 하단동
큰대로변


추천 원룸·분리형
월세 300/3
26㎡·1층
사하구 하단동
투룸급


추천 원룸·오픈형
월세 500/4
26㎡·4층
사하구 하단동
큰길위치


추천 원룸·오픈형
월세 100/1
23㎡·2층
사하구 하단동
가성비갑


추천 원룸·오픈형
전세 6,500
14㎡·7층
사하구 하단동
첫입주 동아대인근 전세…

시간범위와 이동속도

- `trip_times = [1, 5, 10, 20, 30]`

- `travel_speed = 4.5 # (km/hour)`
- `travel_Speed = 13 # (km/hour)` 만약 뛸 때
- `travel_Speed = 40 # (km/hour)` 계주선수라면
- `travel_Speed = 60 # (km/hour)` 자동차 보유 중
- `travel_Speed = 100 # (km/hour)` 레이서 라면

시간당 km -> 분당 meter 로 변환

```
meters_per_minute = travel_speed * 1000 / 60
# km per hour to m per minute
# 4.5 km/h -> 75 m/m
```

Code: Setup

```
!pip install osmnx
!python -m pip uninstall matplotlib
!pip install matplotlib==3.1.3
```

```
import networkx as nx
import osmnx as ox
import matplotlib.pyplot as plt
```

Code: Setup

```
import geopandas as gpd
from descartes import PolygonPatch
from shapely.geometry import LineString
from shapely.geometry import Point
from shapely.geometry import Polygon
```

```
# configure the place, network type, trip times, and travel speed
address = "Hadan-dong, Saha-gu, Busan, Korea"
network_type = "walk"
trip_times = [1, 5, 10, 15, 20, 25, 30]  # 분 단위의 여행시간, 5분거리내
travel_speed = 4.5  # 걷는 속도 (km/hour
```

```
# 위치에 따른 지도 가져오기
G = ox.graph_from_address( address, network_type=network_type)
```

```
# 원하는 지점과 해당 지도에 대해서 그래프
gdf_nodes = ox.graph_to_gdfs(G, edges=False)
# x, y = gdf_nodes["geometry"].unary_union.centroid.
xy # 지도 중심점 좌표
# 동아대학교의 위경도 좌표: lat -> 위도 x , 경도->lon y
x = 128.96817249950897
y = 35.11755694483541
center_node = ox.distance.nearest_nodes(G, x, y)
G = ox.project_graph(G)
```

그래프에 가중치 넣기

```
# 계산을 위한 edges에 속성값을 넣습니다.
meters_per_minute = travel_speed * 1000 / 60  # km per hour to m per minute
for _, _, _, data in G.edges(data=True, keys=True):
    data["time"] = data["length"] / meters_per_minute
```

```
{'osmid': 59528585, 'bridge': 'yes', 'oneway': False, 'ref': '2', 'name': '낙동남로', 'highway': 'primary', 'length': 98.955, 'geometry': <shapely.geometry.linestring.LineString o
{'osmid': [59528596, 157995253], 'bridge': 'yes', 'oneway': False, 'highway': 'primary_link', 'length': 182.09500000000003, 'lanes': '1', 'geometry': <shapely.geometry.linestring.
{'osmid': 59528585, 'bridge': 'yes', 'oneway': False, 'ref': '2', 'name': '낙동남로', 'highway': 'primary', 'length': 98.955, 'geometry': <shapely.geometry.linestring.LineString o
{'osmid': [59528585, 564201227, 942931998], 'bridge': 'yes', 'oneway': False, 'ref': '2', 'name': '낙동남로', 'highway': 'primary', 'length': 419.789, 'lanes': '4', 'geometry': <s
{'osmid': [157995241, 157995221], 'bridge': 'yes', 'oneway': False, 'highway': 'primary_link', 'length': 268.84099999999995, 'lanes': '1', 'geometry': <shapely.geometry.linestring
{'osmid': [59528596, 157995253], 'bridge': 'yes', 'oneway': False, 'highway': 'primary_link', 'length': 182.09500000000003, 'lanes': '1', 'geometry': <shapely.geometry.linestring.
{'osmid': 157995276, 'oneway': False, 'name': '강변대로', 'highway': 'primary', 'length': 360.35699999999997, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1
{'osmid': 164881861, 'oneway': False, 'name': '낙동대로450번길', 'highway': 'residential', 'length': 78.86, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1fa
{'osmid': 620726366, 'oneway': False, 'ref': '2', 'name': '낙동대로', 'highway': 'primary', 'length': 57.065000000000005, 'lanes': '6', 'geometry': <shapely.geometry.linestring.Li
{'osmid': 164881854, 'oneway': False, 'name': '승학로2번길', 'highway': 'residential', 'length': 40.408, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1e7c49
{'osmid': 164881861, 'oneway': False, 'name': '낙동대로450번길', 'highway': 'residential', 'length': 78.86, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1e7
{'osmid': 164881861, 'oneway': False, 'name': '낙동대로450번길', 'highway': 'residential', 'length': 72.544, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1e
{'osmid': 620726366, 'oneway': False, 'ref': '2', 'name': '낙동대로', 'highway': 'primary', 'length': 7.058, 'lanes': '6', 'geometry': <shapely.geometry.linestring.LineString obje
{'osmid': 665487678, 'oneway': False, 'highway': 'service', 'length': 71.195, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1f4c0d10>, 'time': 0.949266666666
{'osmid': 620726366, 'oneway': False, 'ref': '2', 'name': '낙동대로', 'highway': 'primary', 'length': 57.065, 'lanes': '6', 'geometry': <shapely.geometry.linestring.LineString obj
{'osmid': 37398620, 'oneway': False, 'highway': 'tertiary', 'length': 93.363, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1fa1a0d0>, 'time': 1.24484}
{'osmid': 551375433, 'oneway': False, 'name': '낙동대로', 'highway': 'primary', 'length': 53.247, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1fa1ae10>, 't
{'osmid': 37398620, 'oneway': False, 'highway': 'tertiary', 'length': 35.271, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1fa1a7d0>, 'time': 0.470280000000
{'osmid': 37398620, 'oneway': False, 'highway': 'tertiary', 'length': 93.363, 'geometry': <shapely.geometry.linestring.LineString object at 0x7f7c1fa1af90>, 'time': 1.24484}
```

등시성(isochrone) 맵 구현

```
# 등시성의 polygon 구현
isochrone_polys = []
for trip_time in sorted(trip_times, reverse=True):
    subgraph = nx.ego_graph(G, center_node, radius=trip_time, distance="time")
    node_points = [Point((data["x"], data["y"])) for node, data in subgraph.node
s(data=True)]
    bounding_poly = gpd.GeoSeries(node_points).unary_union.convex_hull
    isochrone_polys.append(bounding_poly)


# 등시성에 따른 네트워크를 표기
fig, ax = ox.plot_graph(
    G, show=False, close=False, edge_color="#999999", edge_alpha=0.2, node_size
=0
)
for polygon, fc in zip(isochrone_polys, iso_colors):
    patch = PolygonPatch(polygon, fc=fc, ec="none", alpha=0.6, zorder=-1)
    ax.add_patch(patch)
plt.show()
```

등시성(isochrone) 맵 구현

Polyline 기반 등시성(isochrone) 맵 구현

```
  # isochrone map을 위한 컬러값을 가져옴
  iso_colors = ox.plot.get_colors(n=len(trip_times), cmap="plasma", start=0, return_h
  ex=True)


# 등시성에 따라 노드에 색칠
node_colors = {}
for trip_time, color in zip(sorted(trip_times, reverse=True), iso_colors):
    subgraph = nx.ego_graph(G, center_node, radius=trip_time, distance="time")
    for node in subgraph.nodes():
        node_colors[node] = color
nc = [node_colors[node] if node in node_colors else "none" for node in G.nodes()]
ns = [15 if node in node_colors else 0 for node in G.nodes()]
fig, ax = ox.plot_graph(
    G,
    node_color=nc,
    node_size=ns,
    node_alpha=0.8,
    edge_linewidth=0.2,
    edge_color="#999999",
)
```

Polyline 기반 등시성(isochrone) 맵 구현

Isolated Polyline 기반 등시성(isochrone) 맵 구현

```
def make_iso_polys(G, edge_buff=25, node_buff=50, infill=False):
    isochrone_polys = []
    for trip_time in sorted(trip_times, reverse=True):
        subgraph = nx.ego_graph(G, center_node, radius=trip_time, distance="time")

        node_points = [Point((data["x"], data["y"])) for node, data in subgraph.nodes(data=True)]
        nodes_gdf = gpd.GeoDataFrame({"id": list(subgraph.nodes)}, geometry=node_points)
        nodes_gdf = nodes_gdf.set_index("id")

        edge_lines = []
        for n_fr, n_to in subgraph.edges():
            f = nodes_gdf.loc[n_fr].geometry
            t = nodes_gdf.loc[n_to].geometry
            edge_lookup = G.get_edge_data(n_fr, n_to)[0].get("geometry", LineString([f, t]))
            edge_lines.append(edge_lookup)

        n = nodes_gdf.buffer(node_buff).geometry
        e = gpd.GeoSeries(edge_lines).buffer(edge_buff).geometry
        all_gs = list(n) + list(e)
        new_iso = gpd.GeoSeries(all_gs).unary_union

        if infill:
            new_iso = Polygon(new_iso.exterior)
        isochrone_polys.append(new_iso)
    return isochrone_polys
```

Isolated Polyline 기반 등시성(isochrone) 맵 구현

```
isochrone_polys = make_iso_polys(G, edge_buff=25, node_buff=0, infill=True)
fig, ax = ox.plot_graph(
    G, show=False, close=False, edge_color="#999999", edge_alpha=0.2, node_size=0
)
for polygon, fc in zip(isochrone_polys, iso_colors):
    patch = PolygonPatch(polygon, fc=fc, ec="none", alpha=0.7, zorder=-1)
    ax.add_patch(patch)
plt.show()
```

GIS에 관심있는 학생은

- https://geo-python-site.readthedocs.io