버스별 수요도 분석

목차

---

1. 프로젝트 개요

1.1 배경 및 목표

버스 노선마다 배차 간격이 다름에 따라 수요도 차이로 인해 이러한 현상이 발생하는지 데이터 분석을 통해 확인하고자 함.

1.2 선정 이유

"버스승객 수요가 버스 운행 간격에 직접적인 영향을 미치는가?"

1.3 데이터 수집 및 분석 개요

수집 및 처리한 데이터:

- 데이터는 공공데이터 포털 API를 활용해 노선별 시간대별 승객 수 데이터를 수집
- 그 중 성남시만 운행하는 39개 노선의 버스만 Pandas를 활용하여 가공 및 정제
- 시각화 분석도구 Matplotlib, Seaborn, Folium.

---

2. 데이터 수집

2.1 주요 자료

1. 성남시 경유 노선 목록 데이터
2. 노선별 이용객 시계열 데이터
3. 노선별 정류소 데이터
4. 정류소 간 통과 노선 데이터

2.2 수집 도구 및 방식

- 공공데이터 API활용
- 교통카드 빅데이터 시스템

2.3 데이터 정리 및 처리

- 간단한 정리는 Excel VBA를 활용하여 정리 및 처리
- Pandas를 활용한 복잡한 정리 및 처리

API 수집

DATA 공공데이터포털 .GO.KR

1	노선정보항목조회	노선ID에 해당하는 노선의 기본 정보 및 배차 정보를 조회한다
2	경유정류소목록조회	노선ID에 해당하는 노선의 경유 정류소 목록을 조회한다
3	노선번호목록조회	노선번호에 해당하는 노선의 목록을 조회한다

버스노선현황 리스트, 노선별 이용객 데이터 Excel

변화의 중심 기회의 경기 경기버스정보

경기도교통정보센터

인허가 노선 | 경유 노선

시군선택 성남시

* 시/군 선택후 노선을 검색하시면 해당지역의 인허가 노선을 확인하실 수 있습니다.

노선번호	기점(출발지)	종점	상행	하행	출퇴근배차	평일배차	주말배차	운수사명
일반 100	사기막골	가락시장.가락시장역	주중 : 05:10 ~ 16:40 주말 : 05:10 ~ 16:40	주중 : 06:00 ~ 17:40 주말 : 06:00 ~ 17:40	150 분	160 분	160 분	성남시내버스
일반 100출	산성역.포레스티아동문	남위례역.창곡교차로	주중 : 03:10 ~ 03:10 주말 : 00:00 ~ 00:00	주중 : 04:00 ~ 04:00	-	-	-	성남시내버스

3. 데이터 전처리

3.1 성남시 경유 노선 목록 데이터

- 원본 데이터 : 총 197개의 버스 노선 목록
- 처리 과정:
  i. 성남시를 벗어나는 버스 노선 제거 -> 72개 노선 남음
  ii. 고속버스, 마을버스 등 기타버스 제외하고 일반 버스만을 필터링 -> 최종 39개의 노선 확보
- 결과 : 성남시 내 일반 버스 39개의 노선 목록 데이터 완성

3.2 노선별 이용객 시계열 데이터

- 원본 데이터 : 23년 기준 버스 이용객 시계열 데이터
- 처리 과정 :
  - 전처리된 39개 버스 노선 목록을 기준으로 관련 없는 데이터 제거
- 결과 : 39개 노선의 이용객 시계열 데이터 생성

3.3 노선별 정류소 데이터

- 원본 데이터 : 197개 노선의 경유 정류소 데이터(API활용)
- 처리 과정 :
  i. 1차 API요청으로 197개 노선의 정류소 데이터 수집
  ii. 노선 목록 수정(39개 노선으로 축소) 후 다시 API요청
- 결과 : 39개 노선에 대한 정류소 데이터 처리 완료

3.4 정류소 간 통과 노선 데이터

- 원본 데이터 : 경기도 기준 정류소 간 통과 노선 데이터
- 처리 과정 :
  i. 성남시 외부 정류소 데이터를 모두 제거
  ii. 성남시 내 정류소만 남긴 데이터로 필터링
- 결과 : 성남시 내 정류소 간 통과 노선 데이터 생성

3.5 추가 데이터 통합

- 배차 간격 데이터:
  - 노선별 노선 ID를 기준으로 이용객 시계열 데이터(2번)와 병합
- 위치 데이터 통합:
  - 경기도 정류소별 데이터에서 위치 정보를 추출
  - 노선별 정류소 데이터(3번)에 위치 데이터 병합
- 결과 : 정류소와 노선 데이터를 통합한 최종 데이터 생성

3.1.1 성남시 경유 노선 목록 데이터

```
# CSV 파일 읽기
df = pd.read_csv('성남시_노선번호목록.csv',encoding ='utf-8-sig')
```

```
   순번 관할관청    운행업체  노선번호          기점          종점  인가거리  인가대수  출퇴근배차  평일배차  #
0   1  성남시  성남시내버스   100        사기막골  가락시장.가락시장역  32.1     1    150   160
1   2  성남시  성남시내버스  100출  산성역.포레스티아동문  남위례역.창곡교차로  32.0     1      0     0
2   3  성남시    대원버스   101          오리역     수서역5번출구  44.1    10     15    30
3   4  성남시    대원버스   103      도촌동9단지앞       사당역(중)  61.0    11     18    40
4   5  성남시  성남시내버스   200    도촌동행정복지센터      거여역5번출구  39.6     7     30    40

  주말배차 주중상행첫차 주중상행막차 주말상행첫차 주말상행막차 주중하행첫차 주중하행막차 주말하행첫차 주말하행막차  비고
0  160  05:10  16:40  05:10  16:40  06:00  17:40  06:00  17:40 NaN
1    0  03:10  03:10  00:00  00:00  04:00  04:00    NaN    NaN NaN
2   45  05:00  22:10  05:00  22:10  06:10  23:25  06:10  23:25 NaN
3   40  05:00  21:50  05:00  21:50  06:10  23:20  06:10  23:20 NaN
4   65  05:00  22:00  05:00  22:00  06:10  23:10  06:10  23:10 NaN
```

3.1.2 API를 활용하여 노선번호와 노선 ID 매치

```
# API URL 및 서비스 키
url = 'http://apis.data.go.kr/6410000/busrouteservice/getBusRouteList'
service_key = ''
```

```
# 결과를 저장할 DataFrame
bus_to_id = pd.DataFrame(columns=['노선번호', '기점', '종점', '노선아이디', '지역명', '노선유형명'])
```

```
<response>
  <comMsgHeader/>
▼<msgHeader>
    <queryTime>2024-11-25 21:27:20.245</queryTime>
    <resultCode>0</resultCode>
    <resultMessage>정상적으로 처리되었습니다.</resultMessage>
  </msgHeader>
▼<msgBody>
  ▼<busRouteList>
      <districtCd>2</districtCd>
      <regionName>구리,남양주,서울</regionName>
      <routeId>234000873</routeId>
      <routeName>100</routeName>
      <routeTypeCd>11</routeTypeCd>
      <routeTypeName>직행좌석형시내버스</routeTypeName>
    </busRouteList>
```

XML로 응답을 받고

regionName이 성남인것과

routeType이 일반형시내일경우만

```
# 최종 결과 출력
print(bus_to_id.head())
```

```
  노선번호     기점          종점      노선아이디        지역명       노선유형명
0  100  사기막골  가락시장.가락시장역  234000873  구리,남양주,서울  직행좌석형시내버스
1  100  사기막골  가락시장.가락시장역  225000004      군포,의왕    일반형시내버스
2  100  사기막골  가락시장.가락시장역  228000396         용인    일반형시내버스
3  100  사기막골  가락시장.가락시장역  215000030         연천    일반형시내버스
4  100  사기막골  가락시장.가락시장역  218000133         고양    일반형시내버스
```

```
seongnam_normal_bus_to_id = bus_to_id[
    bus_to_id['지역명'].str.contains('성남', case=False, na=False) &
    (bus_to_id['노선유형명'] == '일반형시내버스')
]
seongnam_normal_bus_to_id.reset_index(drop=True, inplace=True)
# 결과 출력
print(seongnam_normal_bus_to_id.head())
```

	노선번호	기점	종점	노선아이디	지역명	노선유형명
0	100	사기막골	가락시장.가락시장역	204000018	서울,성남	일반형시내버스
1	100출	산성역.포레스티아동문	남위례역.창곡교차로	204000083	서울,성남	일반형시내버스
2	101	오리역	수서역5번출구	228000179	서울,성남	일반형시내버스
3	103	도촌동9단지앞	사당역(중)	204000060	과천,서울,성남,안양,의왕	일반형시내버스
4	200	도촌동행정복지센터	거여역5번출구	204000029	서울,성남,하남	일반형시내버스

3.2.1 노선별 이용객 시계열 데이터

경기도 교통 정보 데이터 센터에서 Excel 다운 후 csv 변환

```
#23년 성남시 노선별 이용객수 데이터 가져옴
bus_time_client = pd.read_csv('23_성남시_노선별_시간대별_이용객수.csv', encoding='utf-8-sig')
bus_num_list = seongnam_normal_bus_to_id['노선번호'].tolist()
seongnam_normal_bus_time_client = bus_time_client[bus_time_client['노선'].isin(bus_num_list)]
```

평일 주말로 나눔

```
#평일만
seongnam_normal_bus_time_client_weekday = seongnam_normal_bus_time_client[seongnam_normal_bus_time_client['일시'] == '평일']
#주말만
seongnam_normal_bus_time_client_weekend = seongnam_normal_bus_time_client[seongnam_normal_bus_time_client['일시'] == '주말']

seongnam_normal_bus_time_client_weekday.to_csv('seongnam_normal_bus_time_client_weekday.csv', index=False, encoding='utf-8-sig
seongnam_normal_bus_time_client_weekend.to_csv('seongnam_normal_bus_time_client_weekend.csv', index=False, encoding='utf-8-sig

print(seongnam_normal_bus_time_client_weekday.head())
print(seongnam_normal_bus_time_client_weekend.head())
```

	시/군/구	연도	월	노선	시종점	일시	시간	이용객수
92	성남시	2023	1	100	사기막골-가락시장.가락시장역	주말	05시~06시	16
93	성남시	2023	1	100	사기막골-가락시장.가락시장역	주말	06시~07시	20
94	성남시	2023	1	100	사기막골-가락시장.가락시장역	주말	07시~08시	2
95	성남시	2023	1	100	사기막골-가락시장.가락시장역	주말	08시~09시	34
96	성남시	2023	1	100	사기막골-가락시장.가락시장역	주말	09시~10시	6

3.3.1 노선별 정류소 데이터 추출

```
# API 호출
response = requests.get('http://apis.data.go.kr/6410000/busrouteservice/getBusRouteStationList', params=params)
# 빈 DataFrame 생성 (결과를 저장할 DataFrame)
seongnam_bus_station = pd.DataFrame(columns=['노선번호', '노선아이디', '정류소아이디', '정류소순번', '정류소명', '정류소번호', 'x', 'y'])
```

	노선번호	노선아이디	정류소아이디	정류소순번	정류소명	정류소번호	x	y
0	100	204000018	205000085	1	사기막골	06148	127.1794167	37.4445167
1	100	204000018	205000033	2	영원무역	06135	127.1778833	37.4439333
2	100	204000018	205000032	3	자동차검사소.산성아파트	06134	127.1780833	37.4416333
3	100	204000018	205000038	4	궁전아파트.성지아파트	06146	127.17875	37.4397167
4	100	204000018	205000037	5	근로자종합복지관	06139	127.1785167	37.4383167

Xml을 통하여서 응답값 받아옴

```
<response>
  <comMsgHeader/>
▼<msgHeader>
    <queryTime>2024-11-25 21:44:08.965</queryTime>
    <resultCode>0</resultCode>
    <resultMessage>정상적으로 처리되었습니다.</resultMessage>
  </msgHeader>
▼<msgBody>
  ▼<busRouteStationList>
      <centerYn>N</centerYn>
      <districtCd>2</districtCd>
      <mobileNo> 06148</mobileNo>
      <regionName>성남</regionName>
      <stationId>205000085</stationId>
      <stationName>사기막골</stationName>
      <x>127.1794167</x>
      <y>37.4445167</y>
      <stationSeq>1</stationSeq>
      <turnYn>N</turnYn>
    </busRouteStationList>
```

3.4.1 정류소간 통과 노선 데이터

경기도 교통 정보 데이터 센터에서 추출

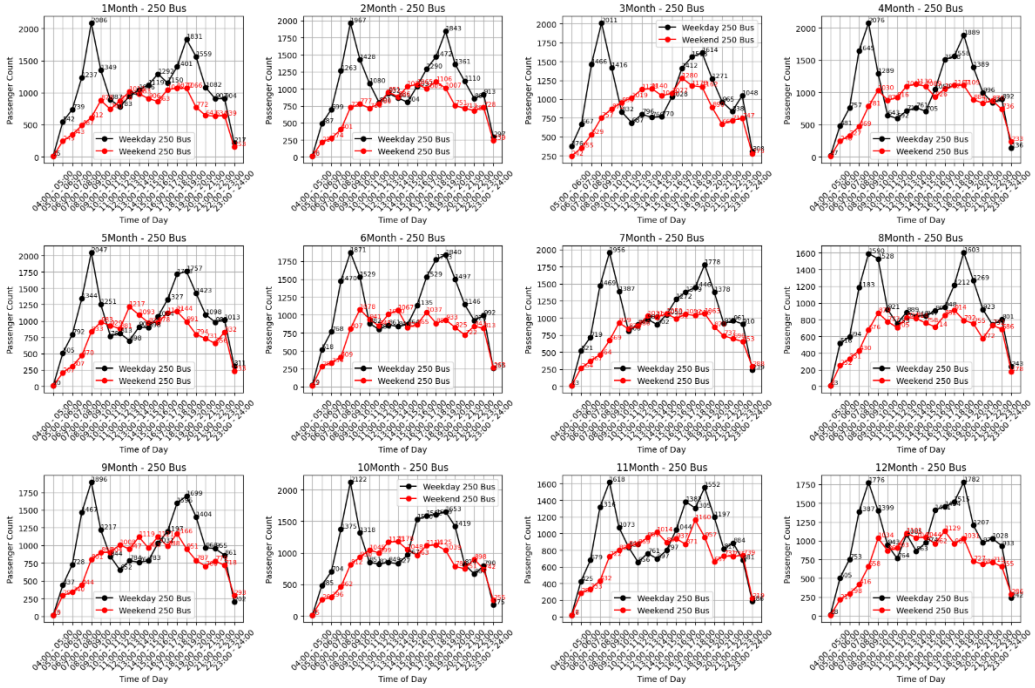
```
pass_bus_station = pd.read_csv('23_성남시_정류소별_노선통과수.csv', encoding='utf-8-sig')
print(pass_bus_station.head())
```

```
   연도 관할지역 버스유형      정류소ID  정류소번호             정류소명  통과노선수     정류소아이디  ₩
0  2023  성남시   일반  206000535   7492   판교역.낙생육교.현대백화점      21  206000535
1  2023  성남시   일반  206000236   7077       수내역.정자아이파크       4  206000236
2  2023  성남시   일반  206000618   7560  이매촌한신.서현역.AK프라자       4  206000618
3  2023  성남시   일반  205000368   6345          모란역5번출구      11  205000368
4  2023  성남시   일반  206000518   7347         국군수도병원정문       1  206000518

           정류소명2  정류소번호2         x좌표         y좌표    행정동             정류장명  ₩
0   판교역.낙생육교.현대백화점     7492  127.111700  37.391483   백현동   판교역.낙생육교.현대백화점
1       수내역.정자아이파크     7077  127.110900  37.376433  수내1동       수내역.정자아이파크
2  이매촌한신.서현역.AK프라자     7560  127.125833  37.386867  서현1동  이매촌한신.서현역.AK프라자
3          모란역5번출구     6345  127.128967  37.430900   성남동          모란역5번출구
4         국군수도병원정문     7347  127.149450  37.390717  서현1동         국군수도병원정문

        상세위치                                    시내버스_경유노선번호 시외버스_경유노선번호
0  판교휴먼시아5단지A  8201(시내),7007-1(시내),103(시내),G8110(시내),8106(시내)...        해당없음
1        아이파크          117(마을),220(시내),370(시내),310(시내),1303(시내)        해당없음
2       한신A건너   33(시내),3(마을),602-1A(마을),3-1(마을),602(마을),32(마을)...        해당없음
3     모란역5번출구  330(시내),357(시내),315(시내),220(시내),88(마을),87(마을),...        해당없음
4      국군수도병원                                          누리2(시내)        해당없음
```

4. 데이터 시각화 및 분석

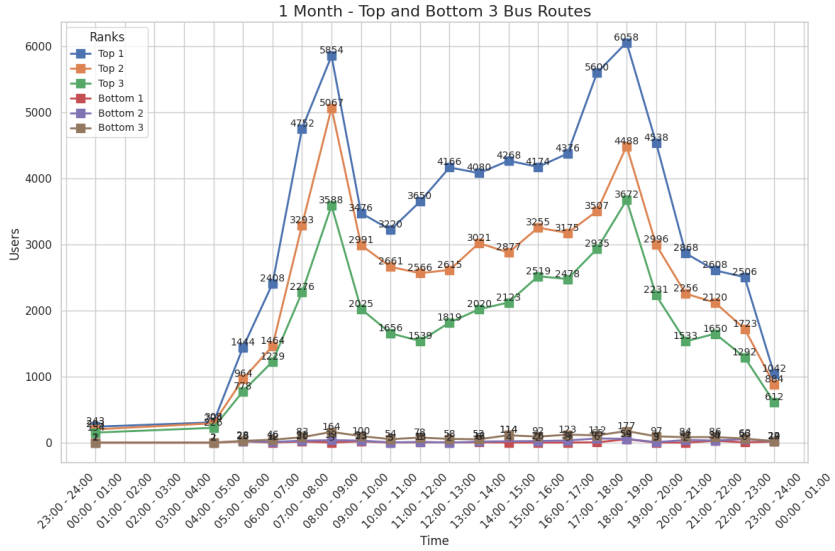
4.1 월별 노선별 수요도(평일, 주말) 비교 시각화

- 목표:월별로 각 노선의 평일과 주말 이용객 수를 비교
- 시각화 방법:
    - Line Plot을 이용하여 X축은 시간대, Y축은 이용개 수로 설정
    - 노선별로 데이터를 시각화하여 평일과 주말의 수요 차이를 분석



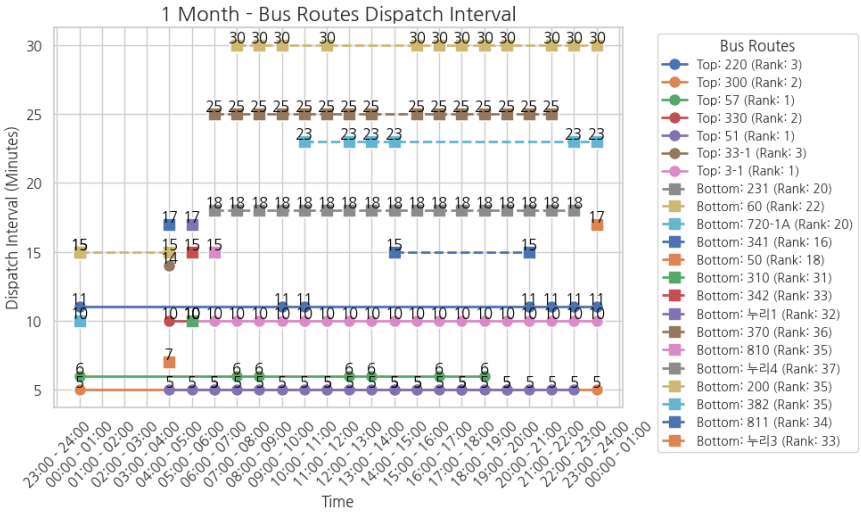
4.2 월별 노선 수요도 순위 비교 시각화

- 목표: 각 노선의 월별 이용객 수에 따른 순위를 비교
- 시각화 방법:
    - Plot을 사용하여 X축은 시간대, Y축은 이용객 수를 기준
    - 상위 3개, 하위 3개 노선에 대한 비교 시각화
    - 순위 변동을 시각적으로 파악할 수 있도록 상위/하위 노선의 변화 추이를 나타냄



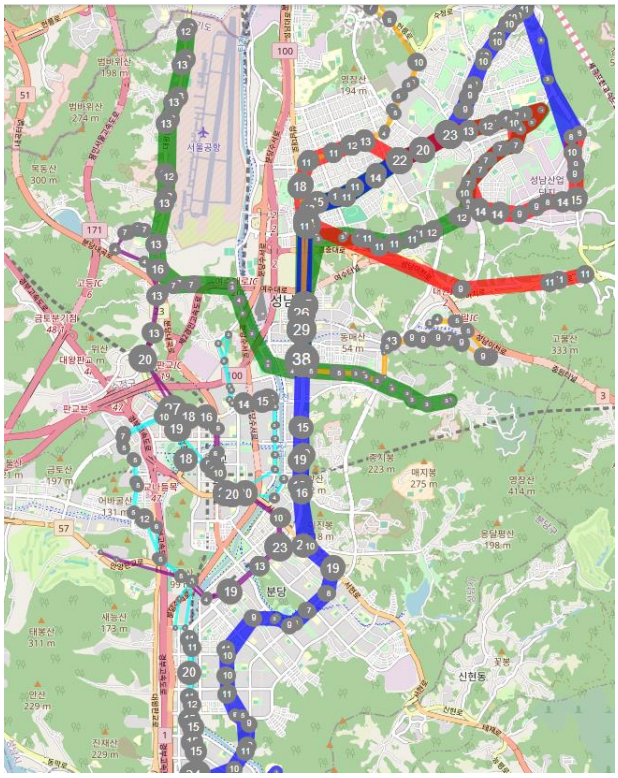
4.3 월별 노선별 순위에 따른 배차 간격 비교 시각화

- 목표: 상위 3개 및 하위 3개 노선의 배차 간격 차이를 분석
- 시각화 방법:
    - Plot을 사용하여 X축은 시간대, Y축은 배차 간격(분)을 기준으로 상위/하위 노선의 배차 간격 비교
    - 상위 노선일수록 배차 간격이 비교적 짧고, 하위 노선은 길다는 패턴을 시각적으로 비교



4.4 월별 노선 수요도에 따른 정류소 지도 시각화

- 목표: 상위 3개 및 하위 3개 노선이 지나가는 정류소의 트래픽을 비교
- 시각화 방법:
    - Folium 지도 시각화를 사용하여 각 노선이 지나가는 정류소를 Line으로 표시
    - 이용객 수가 많고 배차 간격이 짧은 노선의 라인을 두껍게 처리하여 트래픽을 시각적으로 표현
    - 지도 상에서 각 노선의 위치와 통행량을 한눈에 파악할 수 있도록 시각화



4.4.1.1 월별 노선별 수요도(평일, 주말)비교 시각화

시각화 전 처리 과정 250번 노선을 기준으로 진행

```
# CSV 파일 읽기
weekday_df = pd.read_csv('23_성남시_일반노선별_시간대_이용객_평일.csv', encoding='utf-8')
weekend_df = pd.read_csv('23_성남시_일반노선별_시간대_이용객_주말.csv', encoding='utf-8')

# 필요한 열만 추출 (예시: 노선, 일시, 시간, 이용객수)
weekday_df = weekday_df[['연도', '월', '노선', '일시', '시간', '이용객수']]
weekend_df = weekend_df[['연도', '월', '노선', '일시', '시간', '이용객수']]
```

```
route_id = '250'
weekday_route_df = weekday_df[weekday_df['노선'] == route_id]
weekend_route_df = weekend_df[weekend_df['노선'] == route_id]
```

```
# 데이터 확인
print(weekday_route_df.head())
print(weekend_route_df.head())
```

```
      연도  월   노선  일시             시간  이용객수
137  2023  1  250  평일  04:00 - 05:00    15
138  2023  1  250  평일  05:00 - 06:00   542
139  2023  1  250  평일  06:00 - 07:00   739
140  2023  1  250  평일  07:00 - 08:00  1237
141  2023  1  250  평일  08:00 - 09:00  2086
      연도  월   노선  일시             시간  이용객수
135  2023  1  250  주말  04:00 - 05:00     8
136  2023  1  250  주말  05:00 - 06:00   245
137  2023  1  250  주말  06:00 - 07:00   343
138  2023  1  250  주말  07:00 - 08:00   487
139  2023  1  250  주말  08:00 - 09:00   612
```

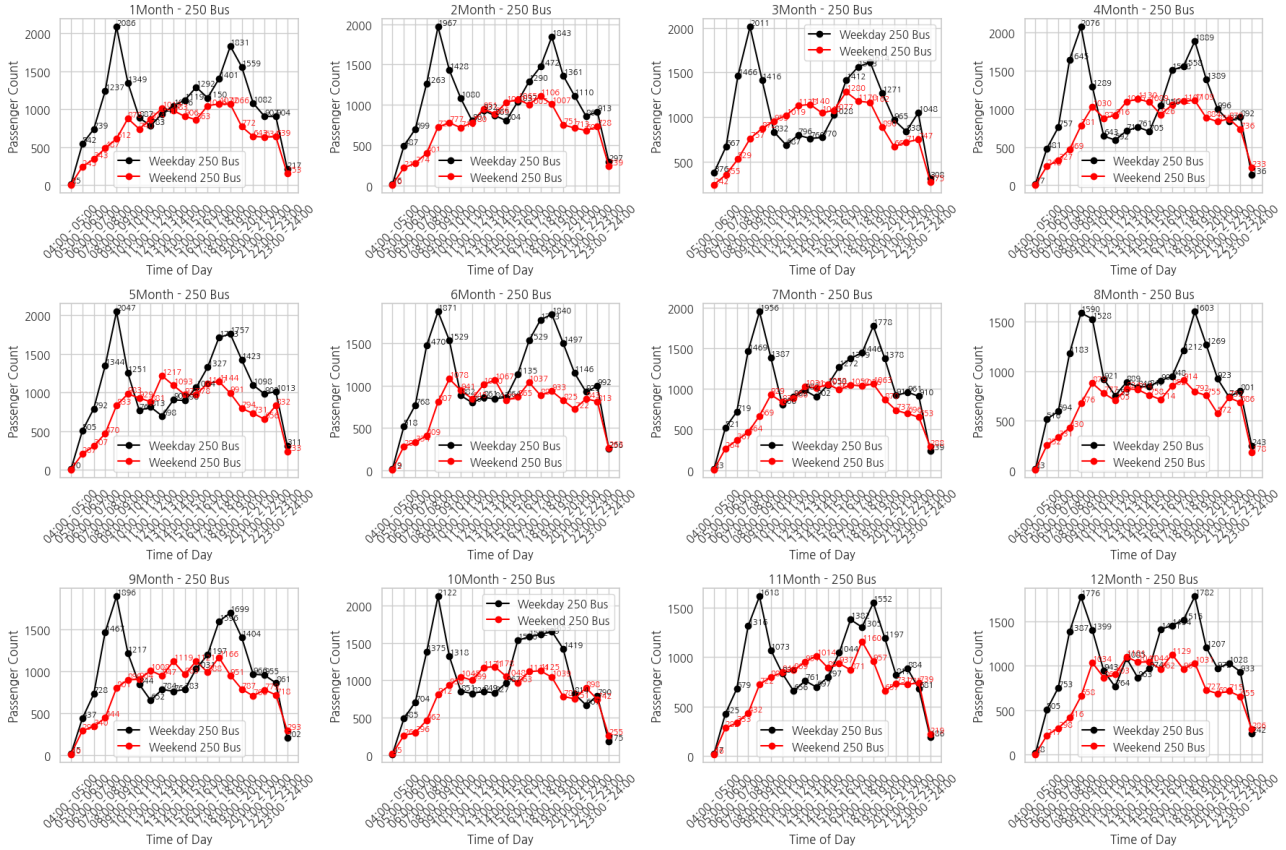
시각화 진행

```
for i, month in enumerate(range(1, 13)):  # 1월부터 12월까지
    # 해당 월에 해당하는 평일 데이터와 주말 데이터 필터링
    month_data_weekday = weekday_route_df[weekday_route_df['월'] == month]
    month_data_weekend = weekend_route_df[weekend_route_df['월'] == month]
```

```
# 평일 데이터 그리기 (검정색)
axes[i].plot(month_data_weekday['시간'], month_data_weekday['이용객수'], marker='o', label=f"Weekday {route_id} Bus", color='black')

# 주말 데이터 그리기 (빨간색)
axes[i].plot(month_data_weekend['시간'], month_data_weekend['이용객수'], marker='o', label=f"Weekend {route_id} Bus", color='red')
```

```
for j, txt in enumerate(month_data_weekday['이용객수']):
    axes[i].text(month_data_weekday['시간'].iloc[j], txt, str(txt), color='black', fontsize=8, verticalalignment='bottom')

for j, txt in enumerate(month_data_weekend['이용객수']):
    axes[i].text(month_data_weekend['시간'].iloc[j], txt, str(txt), color='red', fontsize=8, verticalalignment='bottom')
```

```
    # 그래프 제목 및 레이블 설정
    axes[i].set_title(f"{month}Month - {route_id} Bus")
    axes[i].set_xlabel('Time of Day')
    axes[i].set_ylabel('Passenger Count')
    axes[i].tick_params(axis='x', rotation=45)
    axes[i].grid(True)
    axes[i].legend()

# 레이아웃 조정
plt.tight_layout()
plt.show()
```

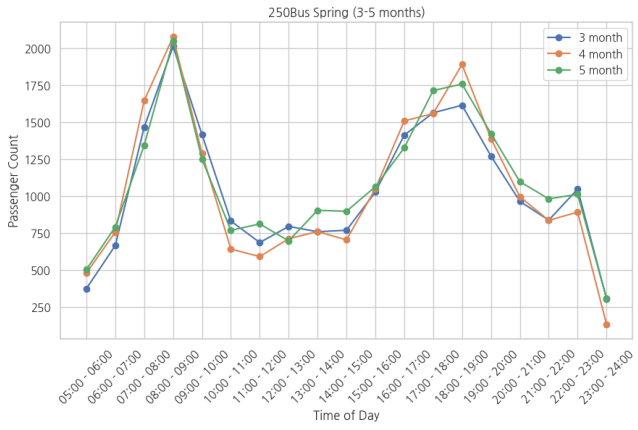


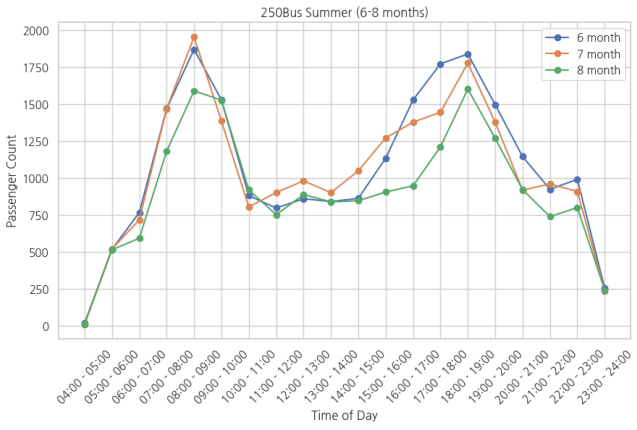
4.4.1.2 평일 및 주말 250번 계절별로 시각화

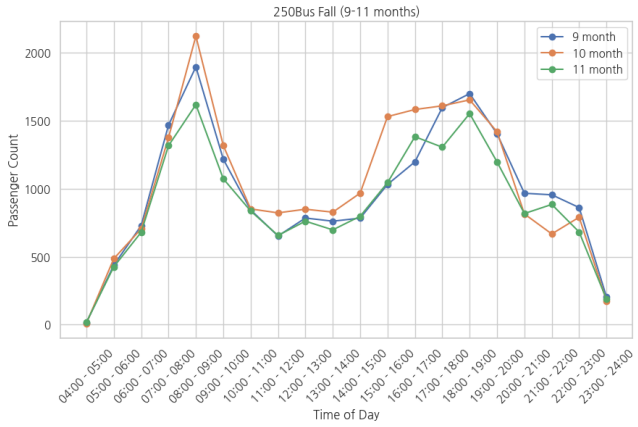
```
def get_season(month):
    if month in [3, 4, 5]:
        return 'Spring'  # 봄
    elif month in [6, 7, 8]:
        return 'Summer'  # 여름
    elif month in [9, 10, 11]:
        return 'Fall'    # 가을
    else:
        return 'Winter'  # 겨울
```

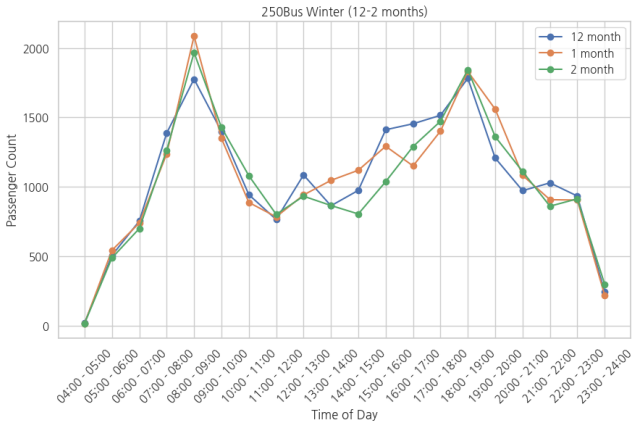
```
def plot_season_graph(df, axes, season, i, labels):
    # 계절별로 3개월 데이터만 필터링
    season_data = df[df['Season'] == season]

    # 각 계절에 맞는 월 범위 지정
    months = {
        'Spring': [3, 4, 5],
        'Summer': [6, 7, 8],
        'Fall': [9, 10, 11],
        'Winter': [12, 1, 2]
    }[season]
```

평일 250번 계절별 시각화

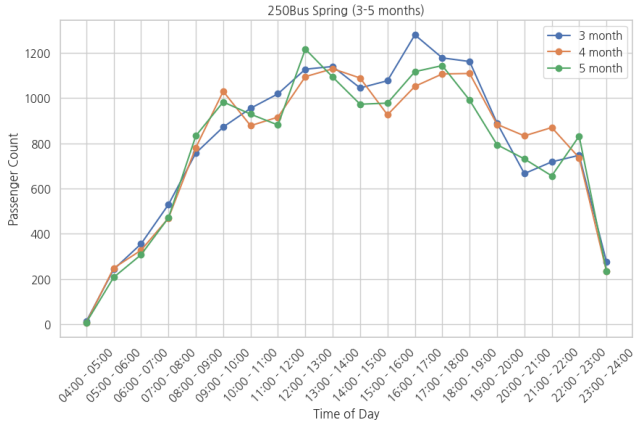


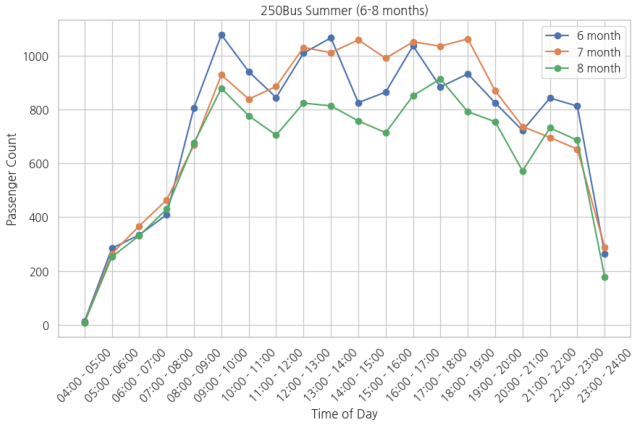


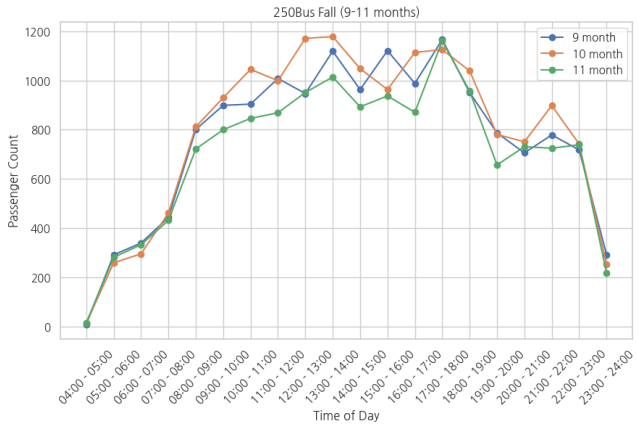


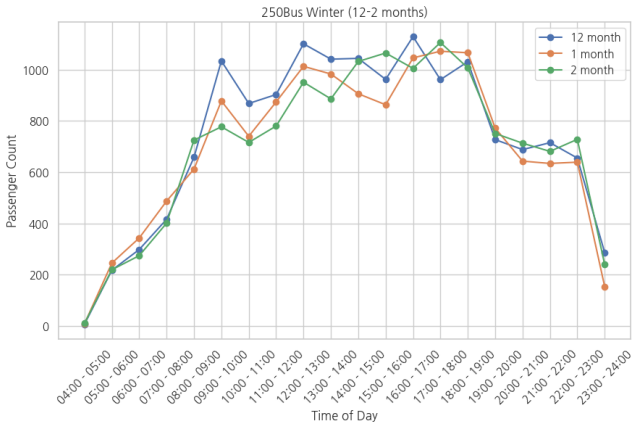


주말 250번 계절별 시각화









4.4.2 월별 노선 수요도 순위 비교 시각화

시각화 전 처리 과정

```
def load_data(file_path):
    # CSV 파일을 로드하는 함수
    df = pd.read_csv(file_path, parse_dates=False, encoding='utf-8-sig')
    df['시간'] = df['시간'].apply(lambda x: x.replace('시~', ':00 - ').replace('시', ':00'))
    return df
```

```
def get_top_bottom_routes(df_month):
    # 상위 3개, 하위 3개 노선을 추출하는 함수
    top_3_routes = df_month[df_month['순위'] <= 3].sort_values(by='시작시간')

    bottom_3_routes = pd.DataFrame()
    for time in df_month['시간'].unique():
        time_data = df_month[df_month['시간'] == time]
        num_routes = len(time_data)
        bottom_3_routes_for_time = time_data[time_data['순위'].isin([num_routes - 2, num_routes - 1, num_routes])]
        bottom_3_routes = pd.concat([bottom_3_routes, bottom_3_routes_for_time])

    bottom_3_routes = bottom_3_routes.sort_values(by='시작시간')
    return top_3_routes, bottom_3_routes
```

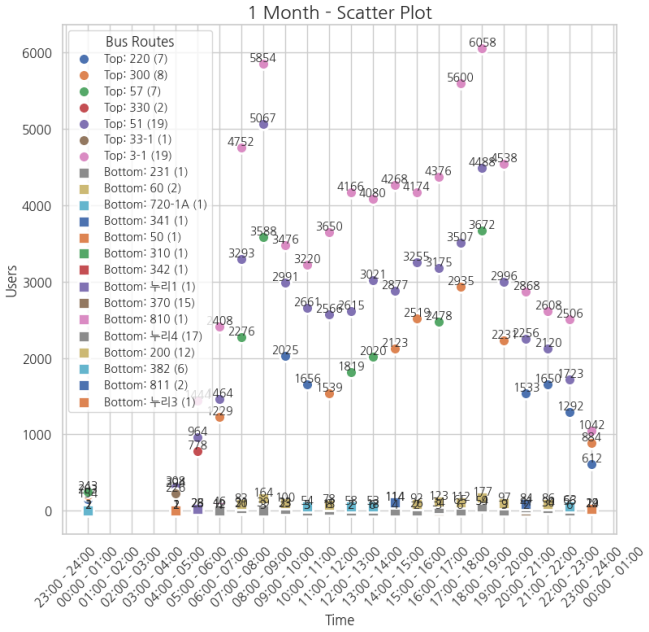
```
df = load_data('23_성남시_일반노선별_시간대_이용객순위_평일_순위수정.csv')
```

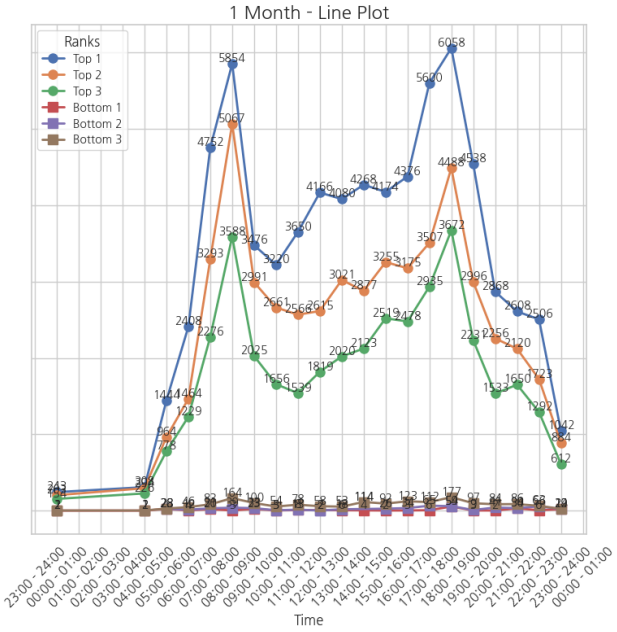
```
for month in range(1, 13):
    df_month = process_month_data(df, month)
    top_3_routes, bottom_3_routes = get_top_bottom_routes(df_month)
    rank_lines_bottom, rank_lines_top = get_rank_lines(df_month)
    # 그래프 그리기
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), sharey=True)
```

```
# 왼쪽 그래프 (산점도)
plot_scatter(ax1, top_3_routes, bottom_3_routes)
ax1.set_title(f"{month} Month - Scatter Plot", fontsize=16)
ax1.set_xlabel("Time", fontsize=12)
ax1.set_ylabel("Users", fontsize=12)
ax1.xaxis.set_major_formatter(FuncFormatter(custom_time_formatter))
ax1.xaxis.set_major_locator(HourLocator(interval=1))
ax1.legend(title="Bus Routes", loc="upper left", fontsize=10)
ax1.tick_params(axis='x', rotation=45)
```

```
# 오른쪽 그래프 (라인 그래프)
plot_line(ax2, rank_lines_top, rank_lines_bottom)
ax2.set_title(f"{month} Month - Line Plot", fontsize=16)
ax2.set_xlabel("Time", fontsize=12)
ax2.xaxis.set_major_formatter(FuncFormatter(custom_time_formatter))
ax2.xaxis.set_major_locator(HourLocator(interval=1))
ax2.legend(title="Ranks", loc="upper left", fontsize=10)
ax2.tick_params(axis='x', rotation=45)
```

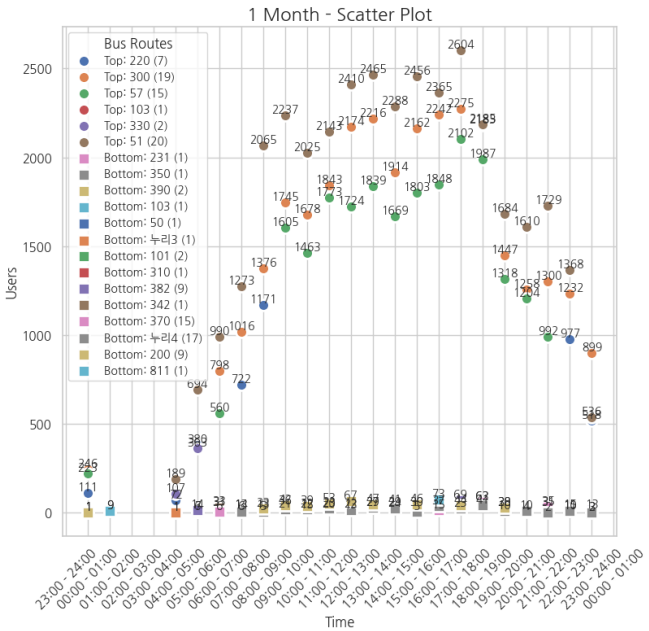
평일 기준 1월달 상위3 하위3 노선 시간대별 이용객 시각화

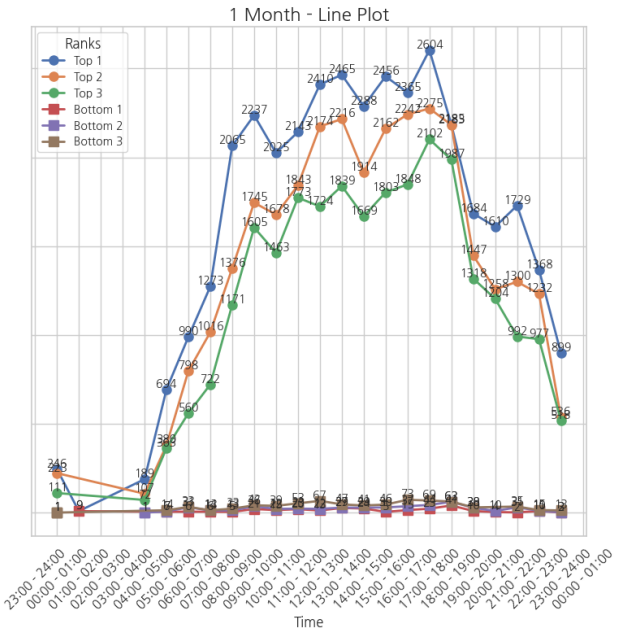




주말 기준 1월달 상위3 하위3 노선 시간대별 이용객 시각화

```
df = load_data('23_성남시_일반노선별_시간대_이용객순위_주말_순위수정.csv')
```





4.4.3 노선 수요 순위에 따른 배차 간격 비교

시각화 전 처리 과정

평일

```
def plot_routes(df_month, month, title_suffix):
    # 상위 3개 노선 필터링
    top_3_routes = df_month[df_month['순위'] <= 3]
    top_3_routes = top_3_routes.sort_values(by='시작시간')

    # 하위 3개 노선 필터링
    bottom_3_routes = pd.DataFrame()
    for time in df_month['시간'].unique():
        time_data = df_month[df_month['시간'] == time]
        num_routes = len(time_data)
        bottom_3_routes_time = time_data[time_data['순위'].isin([num_routes - 2, num_routes - 1, num_routes])]
        bottom_3_routes = pd.concat([bottom_3_routes, bottom_3_routes_time])

    bottom_3_routes = bottom_3_routes.sort_values(by='시작시간')
```

```
# 상위 3개 노선 시각화
for route in top_3_routes['노선'].unique():
    route_data = top_3_routes[top_3_routes['노선'] == route]
    x = route_data['시작시간']
    y = route_data['배차간격(평일)(분)']
    ax.plot(x, y, label=f"Top: {route} (Rank: {route_data['순위'].iloc[0]})", marker='o', markersize=8, linewidth=2)
    ax.scatter(x, y, s=100, edgecolors='w', linewidth=2, marker='o')
    for i in range(len(x)):
        ax.text(x.iloc[i], y.iloc[i], f'{y.iloc[i]}', ha='center', va='bottom', fontsize=12, color='black')

# 하위 3개 노선 시각화
for route in bottom_3_routes['노선'].unique():
    route_data = bottom_3_routes[bottom_3_routes['노선'] == route]
    x = route_data['시작시간']
    y = route_data['배차간격(평일)(분)']
    ax.plot(x, y, label=f"Bot: {route} (Rank: {route_data['순위'].iloc[0]})", marker='s', markersize=8, linewidth=2, linestyle='--')
    ax.scatter(x, y, s=100, edgecolors='w', linewidth=2, marker='s')
    for i in range(len(x)):
        ax.text(x.iloc[i], y.iloc[i], f'{y.iloc[i]}', ha='center', va='bottom', fontsize=12, color='black')
```

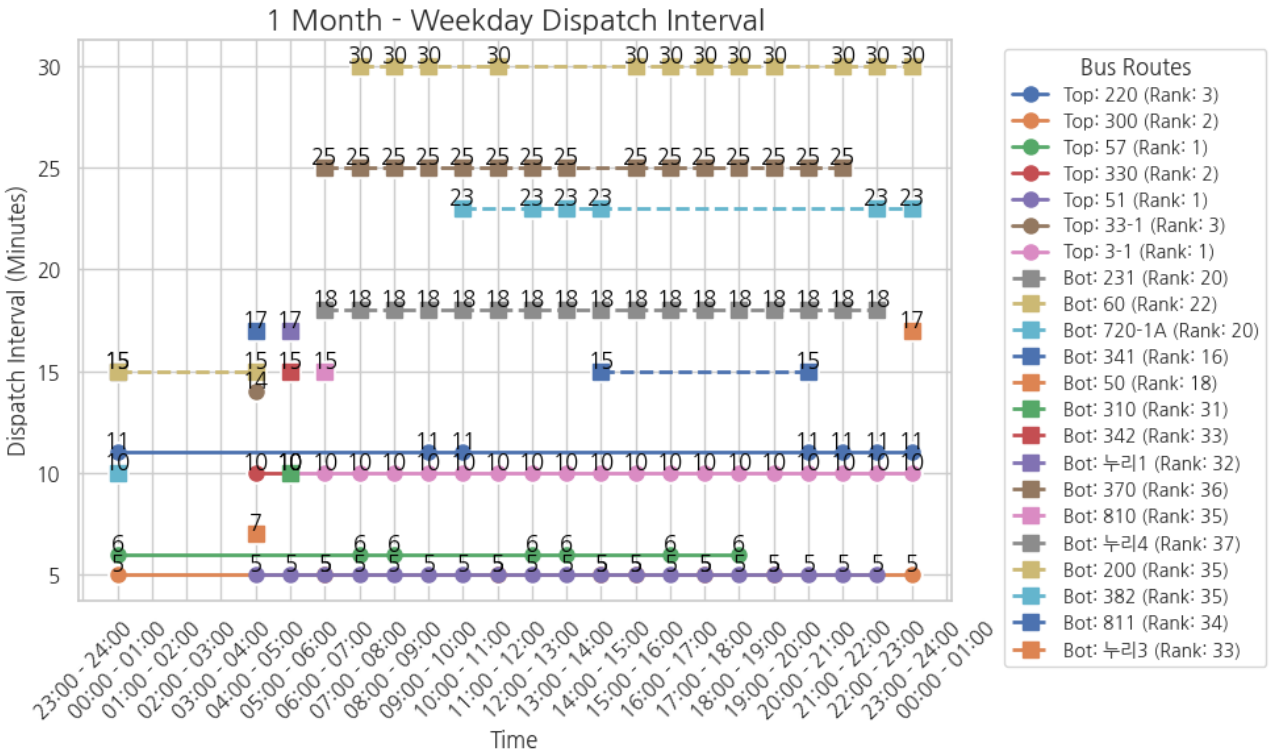
주말(토요일, 일요일)

```
# 토요일 그래프
for route in top_3_saturday['노선'].unique():
    route_data = top_3_saturday[top_3_saturday['노선'] == route]
    x = route_data['시작시간']
    y = route_data['배차간격(토요일)(분)']
    ax1.plot(x, y, label=f"Top: {route} (Rank: {route_data['순위'].iloc[0]})", marker='o', markersize=8, linewidth=2)
    ax1.scatter(x, y, s=100, edgecolors='w', linewidth=2, marker='o')
    for i in range(len(x)):
        ax1.text(x.iloc[i], y.iloc[i], f'{y.iloc[i]}', ha='center', va='bottom', fontsize=12, color='black')

for route in bottom_3_saturday['노선'].unique():
    route_data = bottom_3_saturday[bottom_3_saturday['노선'] == route]
    x = route_data['시작시간']
    y = route_data['배차간격(토요일)(분)']
    ax1.plot(x, y, label=f"Bot: {route} (Rank: {route_data['순위'].iloc[0]})", marker='s', markersize=8, linewidth=2, linestyle='--')
    ax1.scatter(x, y, s=100, edgecolors='w', linewidth=2, marker='s')
    for i in range(len(x)):
        ax1.text(x.iloc[i], y.iloc[i], f'{y.iloc[i]}', ha='center', va='bottom', fontsize=12, color='black')
# 일요일 그래프
for route in top_3_sunday['노선'].unique():
    route_data = top_3_sunday[top_3_sunday['노선'] == route]
    x = route_data['시작시간']
    y = route_data['배차간격(일요일)(분)']
    ax2.plot(x, y, label=f"Top: {route} (Rank: {route_data['순위'].iloc[0]})", marker='o', markersize=8, linewidth=2)
    ax2.scatter(x, y, s=100, edgecolors='w', linewidth=2, marker='o')
    for i in range(len(x)):
        ax2.text(x.iloc[i], y.iloc[i], f'{y.iloc[i]}', ha='center', va='bottom', fontsize=12, color='black')

for route in bottom_3_sunday['노선'].unique():
    route_data = bottom_3_sunday[bottom_3_sunday['노선'] == route]
    x = route_data['시작시간']
    y = route_data['배차간격(일요일)(분)']
    ax2.plot(x, y, label=f"Bot: {route} (Rank: {route_data['순위'].iloc[0]})", marker='s', markersize=8, linewidth=2, linestyle='--')
    ax2.scatter(x, y, s=100, edgecolors='w', linewidth=2, marker='s')
    for i in range(len(x)):
        ax2.text(x.iloc[i], y.iloc[i], f'{y.iloc[i]}', ha='center', va='bottom', fontsize=12, color='black')
```

평일 시각화 1월

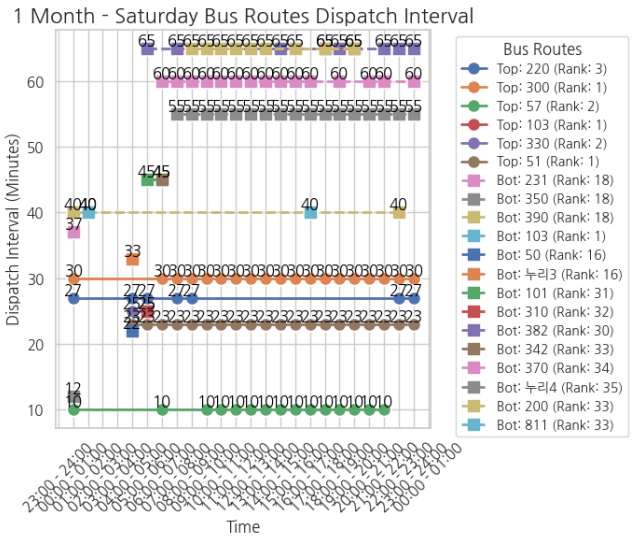
```
df_weekday = pd.read_csv('23_성남시_일반노선별_시간대_이용객순위_평일_순위수정.csv', parse_dates=False, encoding='utf-8-sig')
```

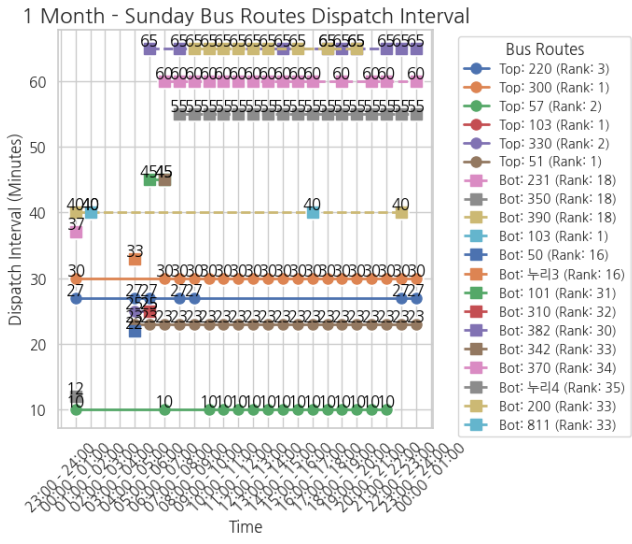
```
for month in range(1, 13):
    # 해당 월 데이터 필터링
    df_month = df_weekday[df_weekday['월'] == month]
    plot_routes(df_month, month, "Weekday")
```



주말(토요일, 일요일) 시각화 1월

```
df_weekday = pd.read_csv('23_성남시_일반노선별_시간대_이용객순위_주말_순위수정.csv', parse_dates=False, encoding='utf-8-sig')
```





4.4.4 월별 노선 수요도에 따른 정류소 지도 시각화

시각화 전 처리 과정

```
def get_top_bottom_routes(df, month):
    top_3_routes_per_time = pd.DataFrame()
    bottom_3_routes_per_time = pd.DataFrame()

    df_month = df[df['월'] == month]
    for time in df_month['시간'].unique():
        time_data = df_month[df_month['시간'] == time]

        # 상위 3개 노선
        top_3_routes = time_data[time_data['순위'] <= 3]
        top_3_routes_per_time = pd.concat([top_3_routes_per_time, top_3_routes])

        # 하위 3개 노선
        num_routes = len(time_data)
        bottom_3_routes = time_data[time_data['순위'].isin([num_routes - 2, num_routes - 1, num_routes])]
        bottom_3_routes_per_time = pd.concat([bottom_3_routes_per_time, bottom_3_routes])

    return top_3_routes_per_time, bottom_3_routes_per_time
```

```
# 정류소마다 마커 생성
for _, station in combined_routes_stops.iterrows():
    lat, lon = station['y좌표'], station['x좌표']
    traffic = station['통과노선수']
    marker_size = math.sqrt(traffic) * 3  # 마커 크기 계산
    marker_color = 'gray'

    folium.Marker(
        location=[lat, lon],
        icon=folium.DivIcon(
            icon_size=(marker_size * 2, marker_size * 2),
            icon_anchor=(marker_size, marker_size),
            html=f'<div style="background-color: {marker_color}; width: {marker_size * 2}px; height: {marker_size * 2}px;
        ),
        popup=f"<b>{station['정류소명']}</b><br>통과 노선 수: {traffic}<br>상세위치: {station['상세위치']}"
    ).add_to(m)
```

```
# 색상 설정
top_colors = ['green', 'blue', 'red']
bottom_colors = ['orange', 'purple', 'cyan']
```

```
# 상위 및 하위 노선별 정류소 연결
for route_df, colors in [(top_routes_stops, top_colors), (bottom_routes_stops, bottom_colors)]:
    unique_routes = route_df['노선번호_x'].unique()
    for i, route in enumerate(unique_routes):
        color = colors[i % len(colors)]
        route_stops = route_df[route_df['노선번호_x'] == route].sort_values('정류소순번')
        coordinates = route_stops[['y좌표', 'x좌표']].values.tolist()

        traffic_data = df_traffic[(df_traffic['노선'] == route) & (df_traffic['시간'] == time) & (df_traffic['월'] == month)]

        if not traffic_data.empty:
            passengers = traffic_data.iloc[0]['이용객수']
            interval = traffic_data.iloc[0][week_dispatch]
            traffic = passengers / interval

            # 100 이상인 경우, 1자리만 남기고 나머지 절삭
            if traffic >= 100:
                first_digit = int(str(int(traffic))[0])  # traffic의 첫 번째 자리를 구함
                traffic = traffic / first_digit  # 첫 번째 자리를 나누어줌

            # 가중치 계산
            weight = (traffic / 10) + 3  # 크기를 조정한 후 가중치 계산
```

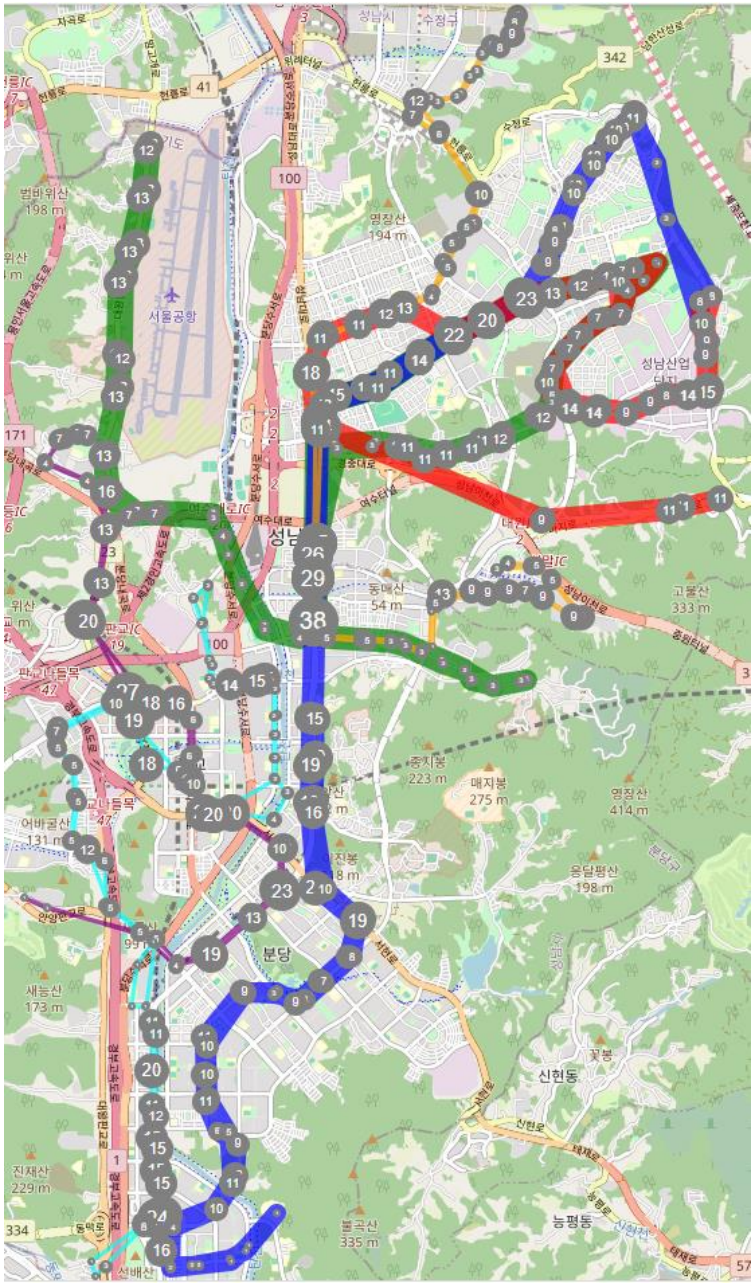
```
folium.PolyLine(
    locations=coordinates,
    color=color,
    weight=weight,
    opacity=0.7,
    tooltip=f"노선: {route}<br>이용객수: {passengers}<br>배차간격: {interval}분"
).add_to(m)
```

평일 1월 07:00-08:00 시간대 노선별 상위3 하위3 트래픽

```
traffic_file = '23_성남시_일반노선별_시간대_이용객순위_평일_순위수정.csv'  # 예: 교통 데이터 파일
stops_file = '성남시_일반노선_정류소별_노선통과.csv'      # 예: 정류소 데이터 파일

# 데이터 로드
df_traffic, df_stops = load_and_preprocess_data(traffic_file, stops_file)

print(df_traffic.head())
print(df_stops.head())
```

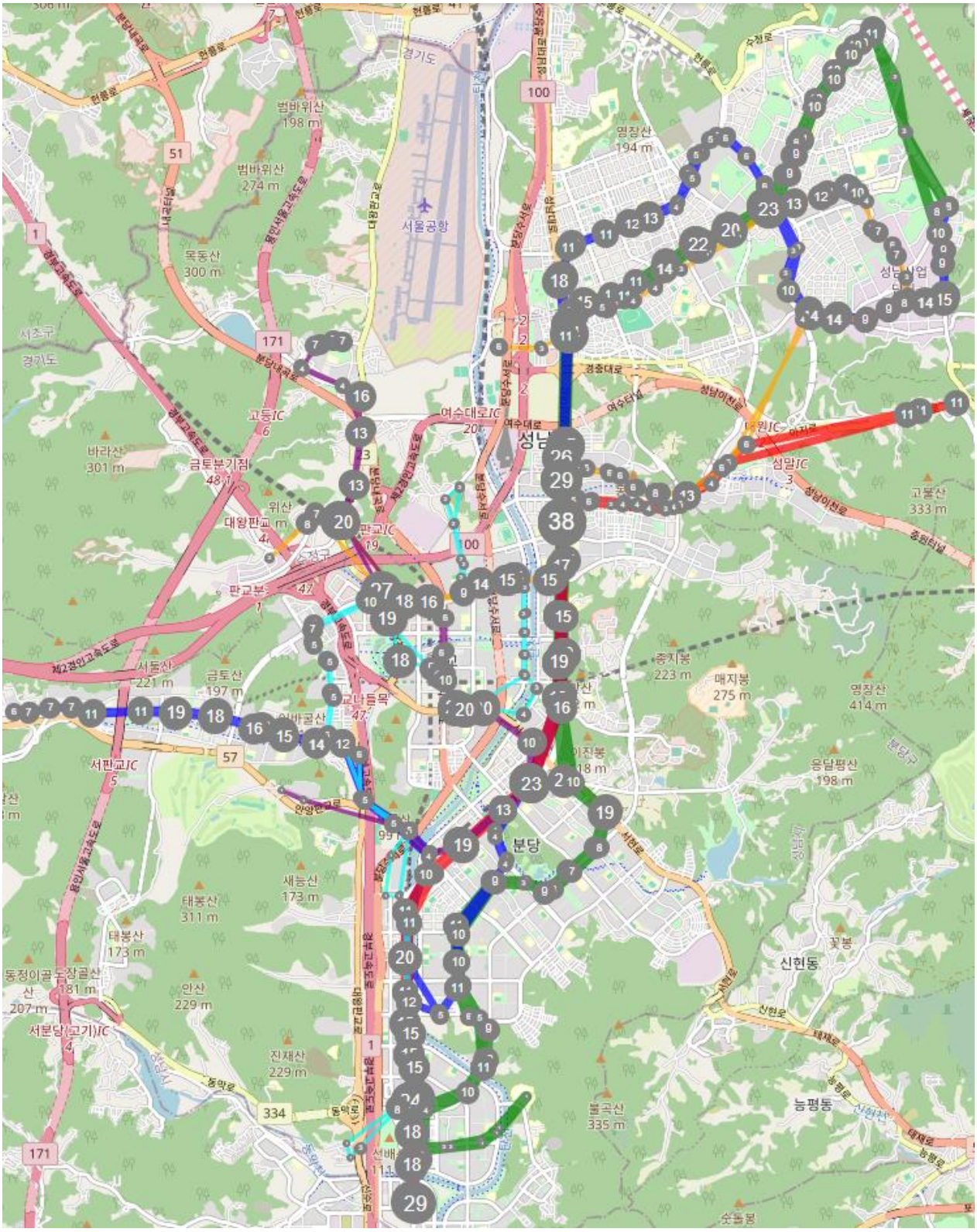
```
month = 1
time = "07:00 - 08:00"
m, route_info_df = generate_map_for_month_and_time(df_traffic, month, time, 0)
# 노선 정보 출력
print(route_info_df)
```

```
   노선번호  순위  이용객수  배차간격(평일)(분) line color
0  3-1    1  4752            10        red
1   51    2  3293             5       blue
2   57    3  2276             6      green
3  200   35    83            30     orange
4  370   37    20            25     purple
5  누리4  36    31             18       cyan
```



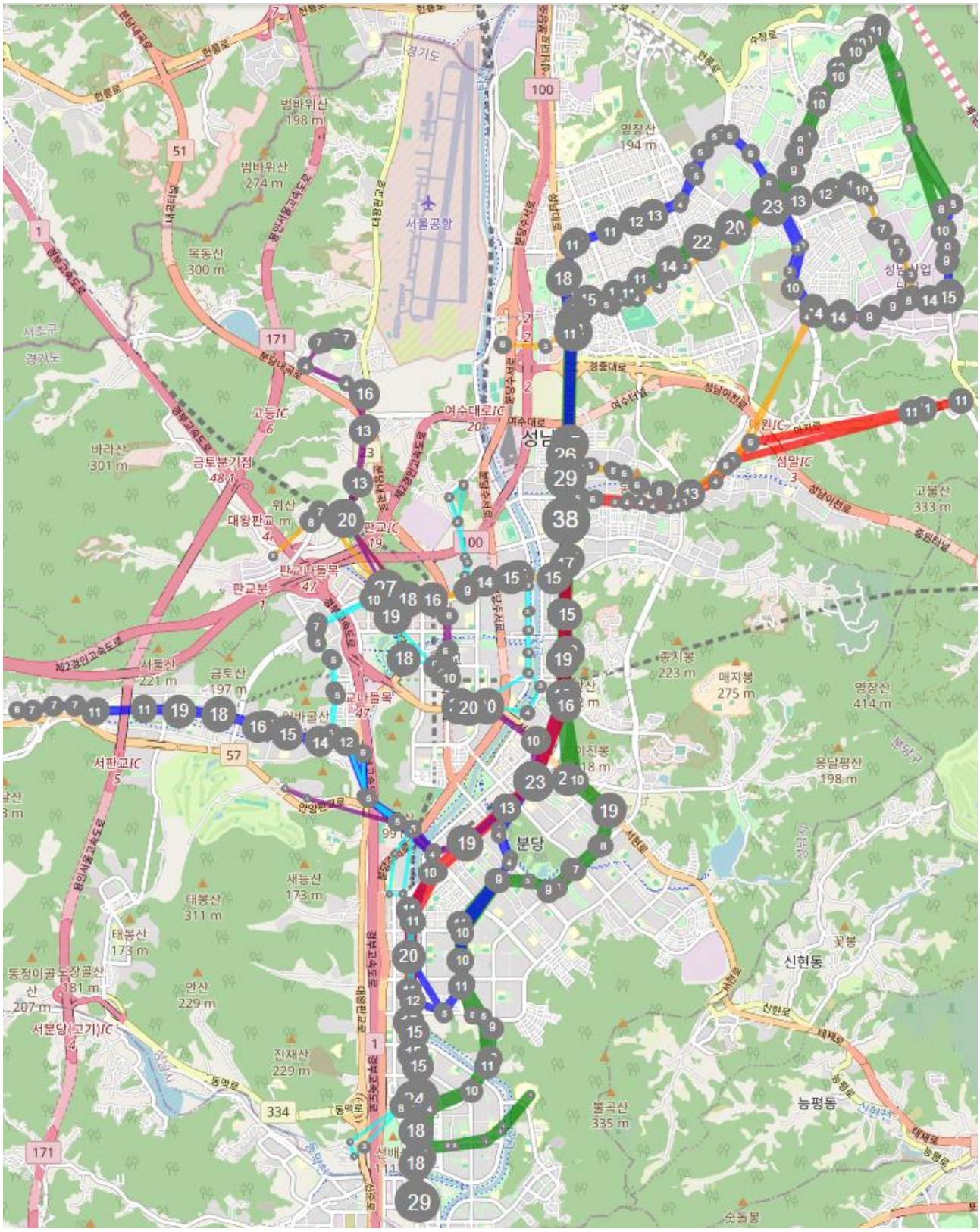
토요일 1월 07:00-08:00 시간대 노선별 상위3 하위3 트래픽

	노선번호	순위	이용객수	배차간격(토요일)(분)	line color
0	220	3	722	27	red
1	300	2	1016	30	blue
2	51	1	1273	23	green
3	370	34	13	60	orange
4	382	33	14	65	purple
5	누리4	35	6	55	cyan



일요일 1월 07:00-08:00 시간대 노선별 상위3 하위3 트래픽

	노선번호	순위	이용객수	배차간격(일요일)(분)	line color
0	220	3	722	27	red
1	300	2	1016	30	blue
2	51	1	1273	23	green
3	370	34	13	60	orange
4	382	33	14	65	purple
5	누리4	35	6	55	cyan



5. 결론

5.1 특정 노선의 시간대별 수요 분석

- 특정 노선의 시간대별 데이터를 분석한 결과, 평일 출퇴근 시간대에 이용객이 가장 많이 몰리는 현상이 두드러졌다.
- 주말에는 상대적으로 완만한 이용 패턴을 보이며, 평일과 주말 간 그래프의 형태가 달랐다.
    - 예 : 250번 평일에는 출퇴근 시간대에 집중된 M자 형태, 주말에는 완만한 포물선 형태로 나타남.

5.2 상위/하위 노선별 수요 비교

- 상위 3개 노선은 평일 기준 1대당 최대 약 6,000명의 이용객을 기록하며, 높은 수요를 보임.
- 하위 3개 노선은 출퇴근 시간대에도 3자리 이용객 수준이며, 대부분의 시간대에서는 평균적으로 2자리 이용객을 넘지 못함.
- 주말에는 상위 노선조차 최대 약 2,500명 수준으로, 주말의 전반적인 수요 감소가 뚜렷하게 나타남.
- 그래프 패턴 역시 평일은 출퇴근 시간대에 집중 M자형태 , 주말은 완만한 포물선 형태의 분포를 보임.

5.3 상위/하위 노선별 배차간격 비교

- 상위 노선은 평일 평균 10~15분 간격으로 운행되며, 일부 시간대 수요도가 1등인 노선은 5분 간격으로도 배차됨.
- 하위 노선은 평일에는 15~30분 간격, 주말에는 40~65분 간격으로 배차가 크게 늘어남.
- 주말 배차간격이 상위/하위 노선 모두 평일보다 길어지는 현상이 관찰됨

5.4 상위/하위 노선의 정류소 및 경로 비교

- 상위 노선은 많은 정류소를 지나며, 도시 주요 지역 및 지하철역을 연결하는 경향이 있음 -예: 모란역은 60개 가량의 노선이 지나며 주요 교통 거점으로 작용, 출퇴근 시간대 교통 체증 유발 가능성이 높음.
- 하위 노선은 주로 동네 내부만 운행하며, 주요 역과 연결되지 않는 경향이 강함.

5.5 종합 결론

- 수요도가 적은 노선일수록 배차간격이 길어지는 현상이 관찰됨.
- 수요도가 높은 노선은 주로 사람들이 많이 사는 지역 및 주요 역을 연결하는 특징을 가짐.