지하철 혼잡도 데이터 분석

인공지능학과 이지양
인공지능학과 왕성원
인공지능학과 김주현

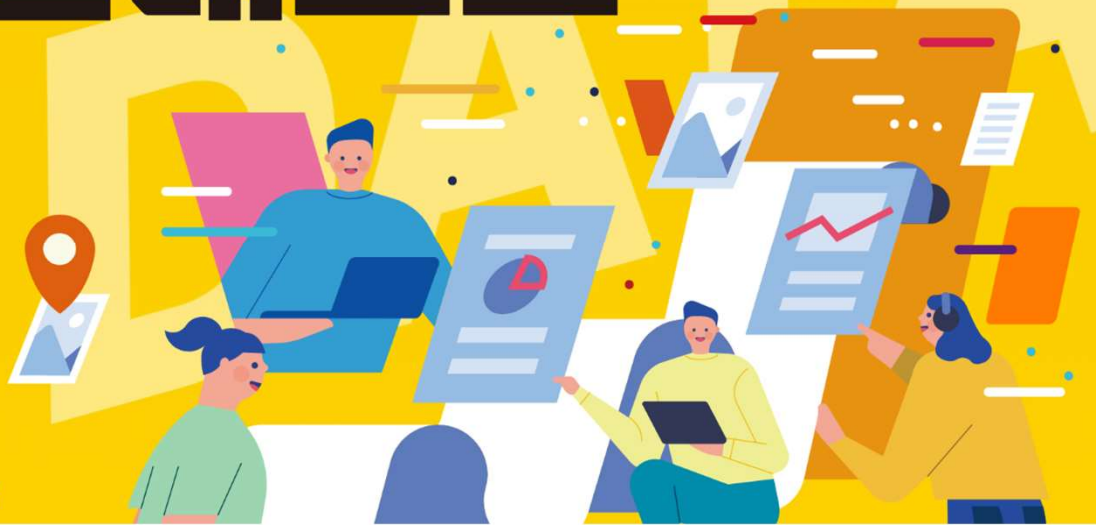
KAIT 한국정보통신진흥협회
Korea Association for ICT Promotion

대회개요 | 공지사항 | 대회 구성 및 주제 | 대회일정 | 심사평가 | 상훈 및 시상금 | 참가신청 및 결과물 제출

2024 빅 콘테스트

데이터 분석 경진대회

BIG DATA



참가 신청
마감
~10.25

결과물 제출
마감
~10.25

대회 문의
bigcontest
@kait.or.kr

유튜브 채널

TOP▲

구분		주 요 내 용	주제설명	데이터 정의서	제공 데이터
생성형AI분야		LLM활용 제주도 맛집 추천 대화형 AI서비스 개발 ※ 제공데이터 : 소비데이터, 관광데이터 등 ※ 분석계획서 제출 팀에 한해 Colab pro 3개월 사용료 지급 (제출 마감 9/30)	다운로드 분석계획서	다운로드	다운로드
데이터분석 분야		OD데이터 분석을 통한 활용방안 제시 ※ 제공데이터 : 인구데이터, OD데이터 등	다운로드	다운로드	다운로드
데이터 활용 분야	지정주제	플랫폼별 지정문제 ※ 플랫폼별 지정 데이터 활용 ※ 감염병, 공간융합, 부동산, 스마트팜, 연안 플랫폼	감염병 공간융합 부동산 스마트팜 연안	감염병 공간융합 부동산 스마트팜 연안	감염병 공간융합 부동산 스마트팜 연안
	자유주제	통합데이터지도를 활용한 혁신 사업화 서비스 제안 ※ 통합 데이터지도에 공개된 데이터 자유롭게 활용	자유주제		통합데이터지도

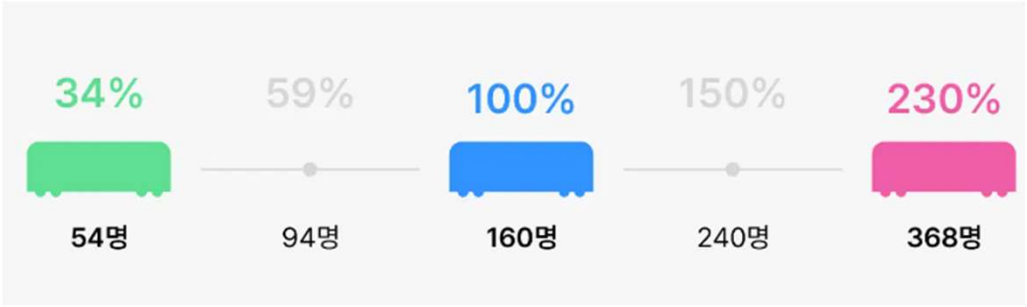
감염 확산 지역과 연계한 지하철 혼잡도 안내 서비스 개발

- 다양한 분야의 데이터 종합하여 팬데믹 위험에 대응하고자 함
- 지하철 혼잡도 데이터를 실시간으로 수집하여 감염병 데이터와 결합
- 특정 장소에서의 감염병 확산 위험을 효과적으로 모니터링
- 예를 들어, 최근 감염병이 발생한 지역에서 혼잡도가 높은 장소는 감염병 전파의 중심이 될 가능성이 높기 때문에, 이를 예방하기 위한 조치를 개인 차원에서 취할 수 있음

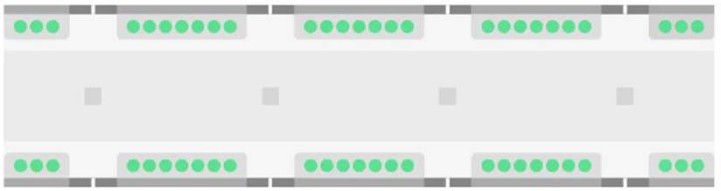
데이터 분석

- 서울교통공사_지하철혼잡도정보_20240630.csv
- 서울시 1호선~8호선의 24년도 상반기 혼잡도 통계
- 30분 간격
- 분석 목적: 혼잡도가 높아지는 시간대와 혼잡한 정도 파악
  혼잡도 패턴 파악

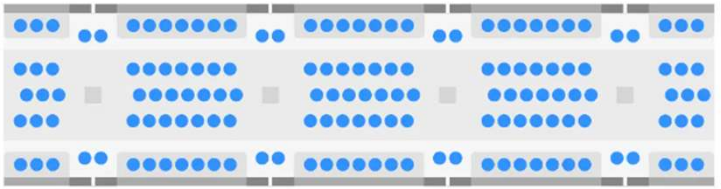
지하철 혼잡도 기준



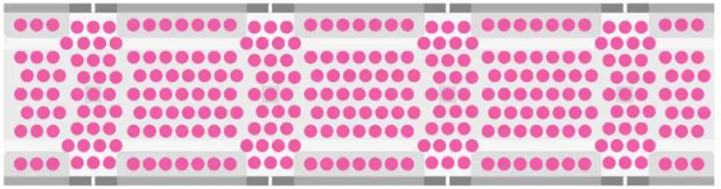
34% 좌석에 모두 앉은 상태



100% 통로에 3줄, 각 출입문에 2명씩 서 있는 상태



230% 통로에 5줄, 각 출입문에 30-40명씩 서 있는 상태



Chapter 1. Exploratory Data Analysis

```
# 데이터 읽어오기
sub_cong_data = pd.read_csv('./서울교통공사_지하철혼잡도정보_20240630.csv', encoding = 'cp949')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1668 entries, 0 to 1667
Data columns (total 45 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   연번      1668 non-null   int64
 1   요일구분    1668 non-null   object
 2   호선      1668 non-null   int64
 3   역번호     1668 non-null   int64
 4   출발역     1668 non-null   object
 5   상하구분    1668 non-null   object
 6   5시30분   1629 non-null   float64
 7   6시00분   1668 non-null   float64
 8   6시30분   1668 non-null   float64
 9   7시00분   1668 non-null   float64
 10  7시30분   1668 non-null   float64
 11  8시00분   1668 non-null   float64
 12  8시30분   1668 non-null   float64
 13  9시00분   1668 non-null   float64
 14  9시30분   1668 non-null   float64
 15  10시00분  1668 non-null   float64
 16  10시30분  1668 non-null   float64
 17  11시00분  1668 non-null   float64
 18  11시30분  1668 non-null   float64
 19  12시00분  1668 non-null   float64
...
 44  00시30분  536 non-null    float64
dtypes: float64(39), int64(3), object(3)
memory usage: 586.5+ KB
None
```

```
print(sub_cong_data.head())
```

✓ 0.0s

	연번	요일구분	호선	역번호	출발역	상하구분	5시30분	6시00분	6시30분	7시00분	...	20시00분	\
0	1	평일	1	158	청량리	상선	9.6	9.3	5.2	8.6	...	25.0	
1	2	평일	1	158	청량리	하선	22.4	20.4	30.0	43.8	...	10.9	
2	3	평일	1	157	제기동	상선	9.7	10.5	7.6	8.9	...	29.0	
3	4	평일	1	157	제기동	하선	22.6	21.3	31.3	43.9	...	12.6	
4	5	평일	1	156	신설동	상선	8.7	13.0	8.3	10.0	...	29.4	

	20시30분	21시00분	21시30분	22시00분	22시30분	23시00분	23시30분	00시00분	00시30분
0	24.8	27.5	24.6	22.4	22.8	23.6	15.8	8.9	NaN
1	9.4	10.8	8.8	7.1	7.7	6.5	6.4	2.0	0.2
2	24.2	34.1	27.5	25.0	25.6	26.3	17.9	12.1	NaN
3	10.8	11.4	9.3	7.9	8.3	6.9	6.5	1.9	0.4
4	25.1	35.3	28.2	26.0	27.1	26.6	16.7	13.3	NaN

[5 rows x 45 columns]

Exploring Numerical Data

- Estimates of Location

```
# 시간대 리스트 생성
time_list = sub_cong_data.columns.tolist()
del time_list[0:6]                          # 인덱스 0부터 5까지 삭제
```

```
# 각 이용시간대마다 전체 열차의 평균 혼잡도
for i in range(len(time_list)) :
    print(time_list[i], '평균 혼잡도', sub_cong_data[time_list[i]].mean())
```

```
# 각 이용시간대마다 전체 열차의 (절사)평균 혼잡도
for i in range(len(time_list)) :
    print(time_list[i], '절사 평균 혼잡도', trim_mean(sub_cong_data[time_list[i]], 0.1))
```

```
# 각 이용시간대마다 전체 열차의 혼잡도 중위값
for i in range(len(time_list)) :
    print(time_list[i], '혼잡도 중위값', sub_cong_data[time_list[i]].median())
```

알고 싶은 것은 사람들이 많이 몰리는 시간대와 그 정도인데 너무 온건한 값들만 산출됨.

```
# 각 이용시간대마다 전체 열차의 혼잡도 최댓값
for i in range(len(time_list)) :
    print(time_list[i], '혼잡도 최댓값', sub_cong_data[time_list[i]].max())
```

```
5시30분 평균 혼잡도 16.9963167587477
6시00분 평균 혼잡도 16.63896882494005
6시30분 평균 혼잡도 15.80617505995204
7시00분 평균 혼잡도 20.24484412470024
7시30분 평균 혼잡도 25.776678657074342
8시00분 평균 혼잡도 32.198800959232614
8시30분 평균 혼잡도 29.879676258992806
9시00분 평균 혼잡도 29.19412470023981
9시30분 평균 혼잡도 27.465167865707436
10시00분 평균 혼잡도 25.92248201438849
10시30분 평균 혼잡도 26.294004796163076
11시00분 평균 혼잡도 26.210911270983214
11시30분 평균 혼잡도 26.307673860911272
12시00분 평균 혼잡도 27.593285371702642
12시30분 평균 혼잡도 28.870743405275782
13시00분 평균 혼잡도 29.10305755395683
13시30분 평균 혼잡도 28.61193045563549
14시00분 평균 혼잡도 27.787649880095923
14시30분 평균 혼잡도 28.225119904076738
15시00분 평균 혼잡도 29.27182254196643
15시30분 평균 혼잡도 31.155335731414866
16시00분 평균 혼잡도 33.18429256594724
16시30분 평균 혼잡도 35.72961630695444
17시00분 평균 혼잡도 36.64130695443645
17시30분 평균 혼잡도 37.26241007194245
...
23시00분 평균 혼잡도 20.007194244604317
23시30분 평균 혼잡도 14.525753920386007
00시00분 평균 혼잡도 4.837973704563032
00시30분 평균 혼잡도 5.902985074626866
```

```
5시30분 혼잡도 최댓값 121.2
6시00분 혼잡도 최댓값 78.3
6시30분 혼잡도 최댓값 83.9
7시00분 혼잡도 최댓값 126.6
7시30분 혼잡도 최댓값 147.0
8시00분 혼잡도 최댓값 145.1
8시30분 혼잡도 최댓값 148.3
9시00분 혼잡도 최댓값 140.2
9시30분 혼잡도 최댓값 109.0
10시00분 혼잡도 최댓값 73.5
10시30분 혼잡도 최댓값 81.7
11시00분 혼잡도 최댓값 75.4
11시30분 혼잡도 최댓값 74.3
12시00분 혼잡도 최댓값 87.1
12시30분 혼잡도 최댓값 75.5
13시00분 혼잡도 최댓값 71.9
13시30분 혼잡도 최댓값 72.2
14시00분 혼잡도 최댓값 69.7
14시30분 혼잡도 최댓값 71.9
15시00분 혼잡도 최댓값 76.0
15시30분 혼잡도 최댓값 72.8
16시00분 혼잡도 최댓값 91.8
16시30분 혼잡도 최댓값 87.6
17시00분 혼잡도 최댓값 120.4
17시30분 혼잡도 최댓값 135.7
...
23시00분 혼잡도 최댓값 99.7
23시30분 혼잡도 최댓값 87.7
00시00분 혼잡도 최댓값 41.2
00시30분 혼잡도 최댓값 23.9
```

값이 꽤 차이가 난다는 것을 알 수 있음.

Exploring Numerical Data

- Estimates of Variability

```
# 각 이용시간대마다 전체 열차의 혼잡도 표준편차
for i in range(len(time_list)) :
    print(time_list[i], '혼잡도 표준편차', sub_cong_data[time_list[i]].std())
```

```
# 각 이용시간대마다 전체 열차의 혼잡도 IQR
for i in range(len(time_list)) :
    print(time_list[i], 'IQR', sub_cong_data[time_list[i]].quantile(0.75) - sub_cong_data[t
```

```
# 각 이용시간대마다 전체 열차의 혼잡도 MAD (평균절대편차)
for i in range(len(time_list)) :
    print(time_list[i], '혼잡도 MAD', robust.scale.mad(sub_cong_data[time_list[i]]))
```

```
5시30분 혼잡도 표준편차 15.18899320827187
6시00분 혼잡도 표준편차 11.965557124696137
6시30분 혼잡도 표준편차 13.197347682051925
7시00분 혼잡도 표준편차 20.048059019805162
7시30분 혼잡도 표준편차 25.94013160325786
8시00분 혼잡도 표준편차 29.38042288176616
8시30분 혼잡도 표준편차 23.58285958830768
9시00분 혼잡도 표준편차 21.00438812698717
9시30분 혼잡도 표준편차 16.191398070503457
10시00분 혼잡도 표준편차 13.782659243248856
10시30분 혼잡도 표준편차 13.662144970235802
11시00분 혼잡도 표준편차 13.024516161634256
11시30분 혼잡도 표준편차 12.806477107056784
12시00분 혼잡도 표준편차 13.213453288528939
12시30분 혼잡도 표준편차 13.364209925974576
13시00분 혼잡도 표준편차 13.028315211278702
13시30분 혼잡도 표준편차 12.82506666974262
14시00분 혼잡도 표준편차 12.075672858169124
14시30분 혼잡도 표준편차 12.445365605742541
15시00분 혼잡도 표준편차 12.826188801504877
15시30분 혼잡도 표준편차 13.727792941597887
16시00분 혼잡도 표준편차 14.9839475887387
16시30분 혼잡도 표준편차 16.70502542942519
17시00분 혼잡도 표준편차 19.434429833620335
17시30분 혼잡도 표준편차 22.20033740443447
...
23시00분 혼잡도 표준편차 14.072020887103225
23시30분 혼잡도 표준편차 10.91089743075599
00시00분 혼잡도 표준편차 6.388730008628242
00시30분 혼잡도 표준편차 4.603565303390466
```

Exploring Numerical Data

```
for i in range(len(time_list)):
    df = pd.DataFrame(sub_cong_data[time_list[i]].quantile(percentages))
    df.index = [f'{p * 100}%' for p in percentages]
    print(df.transpose())
```

✓ 0.0s

```
       5.0%  25.0%  50.0%  75.0%  95.0%
5시30분   2.8    7.7   12.4   21.9  47.26
        5.0%  25.0%  50.0%  75.0%  95.0%
6시00분  2.635    8.6   13.9   21.5   40.5
        5.0%  25.0%  50.0%   75.0%  95.0%
6시30분  2.735    7.6   11.9  19.225  45.13
       5.0%  25.0%  50.0%  75.0%  95.0%
7시00분   2.8    8.0   12.8   24.5  65.66
        5.0%  25.0%  50.0%  75.0%   95.0%
7시30분  3.235    9.2  15.85   32.4  83.495
        5.0%   25.0%  50.0%   75.0%  95.0%
8시00분  4.335  12.775   21.7  40.625  99.53
        5.0%  25.0%  50.0%   75.0%  95.0%
8시30분  5.105   14.8  22.55  37.225  79.83
       5.0%  25.0%  50.0%  75.0%  95.0%
9시00분   5.1   15.9   23.7   36.7  71.53
        5.0%   25.0%  50.0%  75.0%   95.0%
9시30분  5.735  17.075   24.4   35.6  58.965
        5.0%   25.0%  50.0%  75.0%  95.0%
10시00분  5.34  16.675   24.2   33.8   51.6
        5.0%   25.0%  50.0%   75.0%   95.0%
10시30분   5.8  17.375   24.9  34.225  50.965
         5.0%   25.0%  50.0%  75.0%   95.0%
11시00분  5.235  17.675  25.65   34.4  48.265
        5.0%   25.0%  50.0%  75.0%   95.0%
...
        5.0%  25.0%  50.0%  75.0%  95.0%
00시00분   0.0    0.0    2.0    8.3  17.64
         5.0%  25.0%  50.0%  75.0%   95.0%
00시30분  0.275    2.5    4.6  8.725  14.875
```

- Percentiles

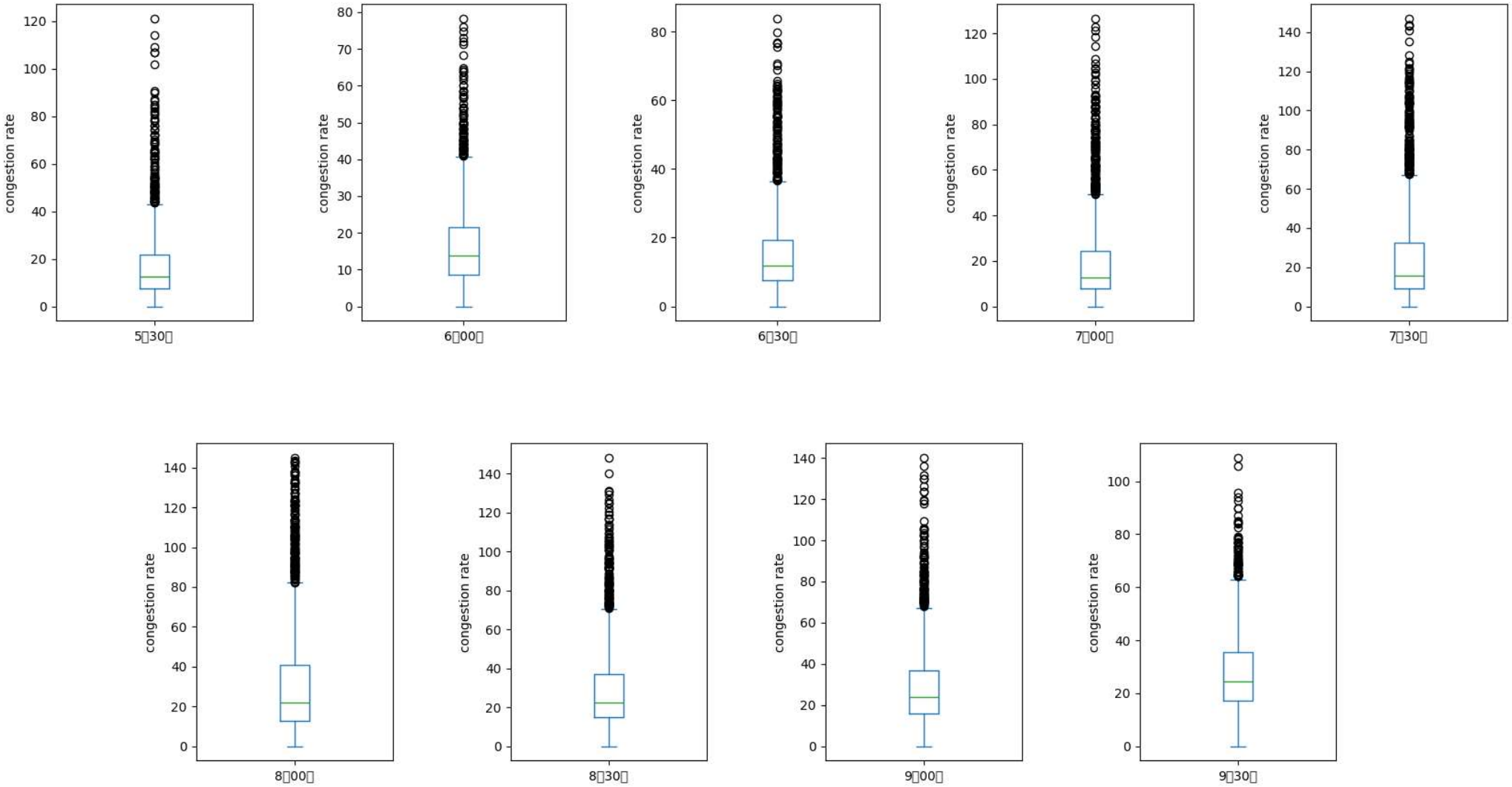
Exploring Numerical Data

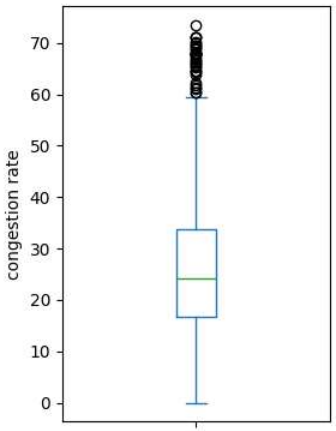
- Box plots

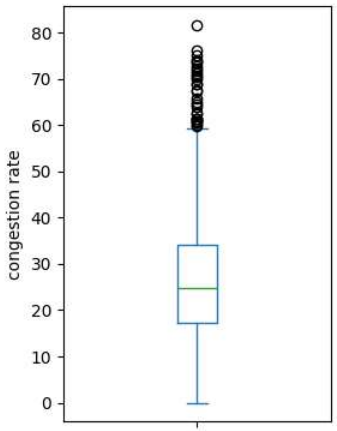
```
# Box Plot 그리기
for i in range(len(time_list)) :
    ax = (sub_cong_data[time_list[i]]).plot.box(figsize=(3, 4))
    ax.set_ylabel('congestion rate')
    plt.tight_layout()
    plt.show()
```

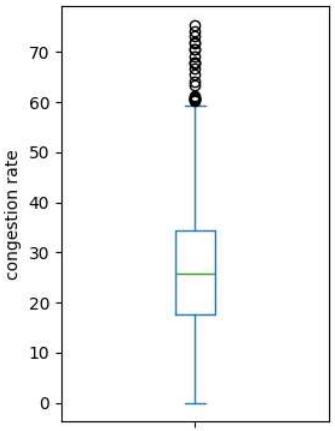
- 이상치가 매우 많음

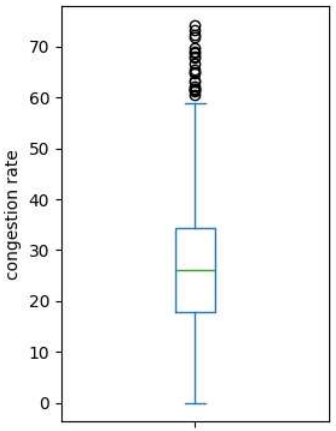
- 출퇴근 시간대에 더욱 몰려 있는 것을 볼 수 있음

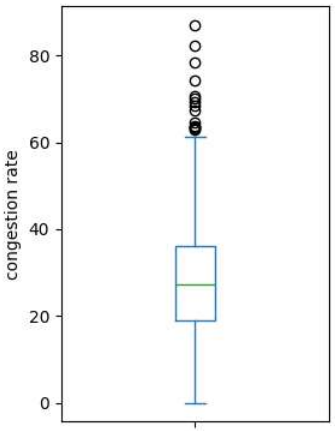

congestion rate
5時30分
6時00分
6時30分
7時00分
7時30分
8時00分
8時30分
9時00分
9時30分

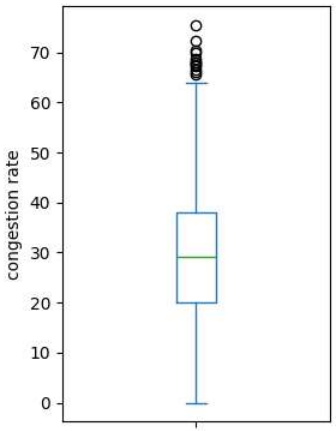

congestion rate
0
10
20
30
40
50
60
70
10月00日

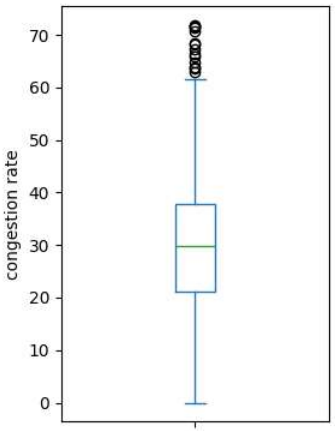

congestion rate
0
10
20
30
40
50
60
70
80
10月30日

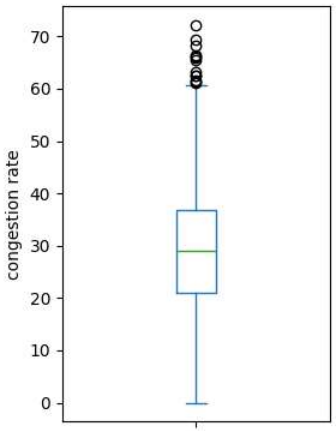

congestion rate
0
10
20
30
40
50
60
70
11月00日

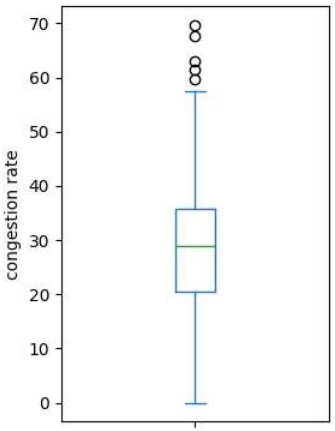

congestion rate
0
10
20
30
40
50
60
70
11月30日

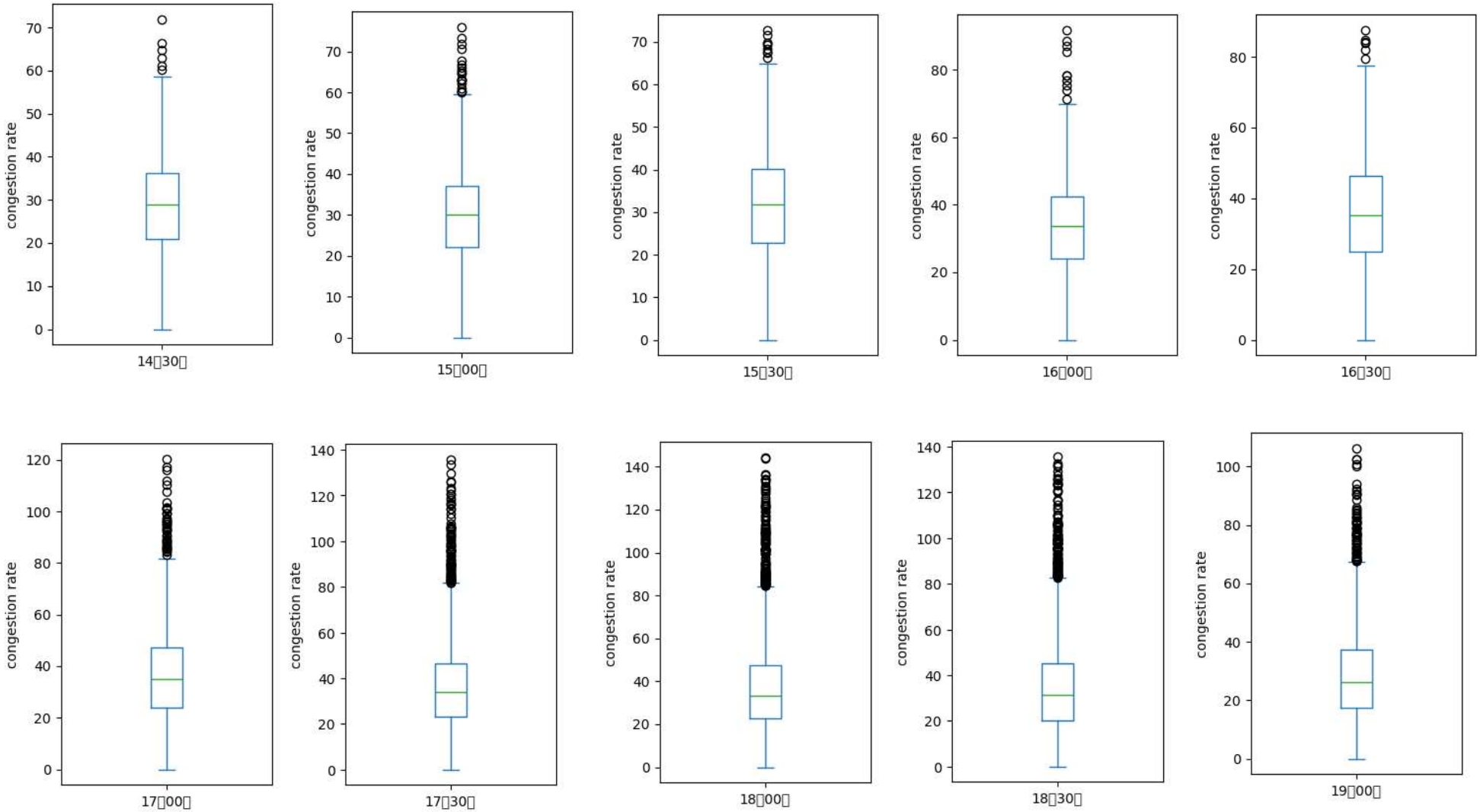

congestion rate
0
20
40
60
80
12月00日

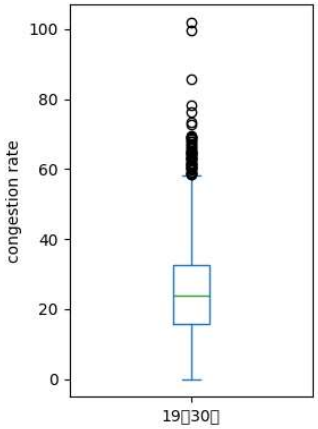

congestion rate
0
10
20
30
40
50
60
70
12月30日

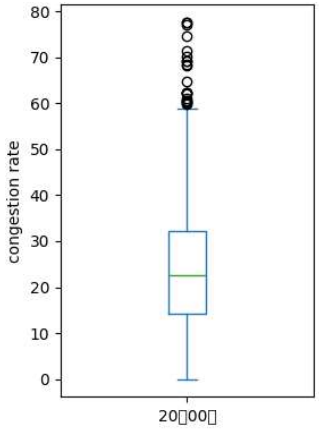

congestion rate
0
10
20
30
40
50
60
70
13月00日

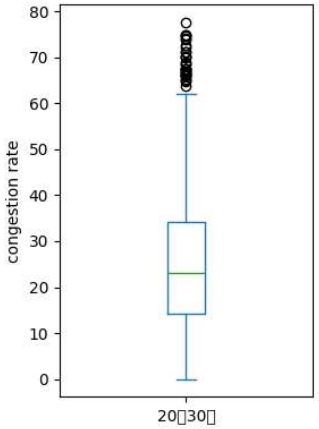

congestion rate
0
10
20
30
40
50
60
70
13月30日

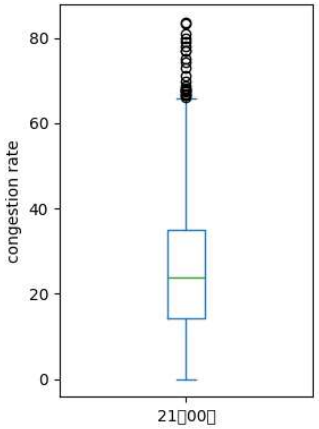

congestion rate
0
10
20
30
40
50
60
70
14月00日

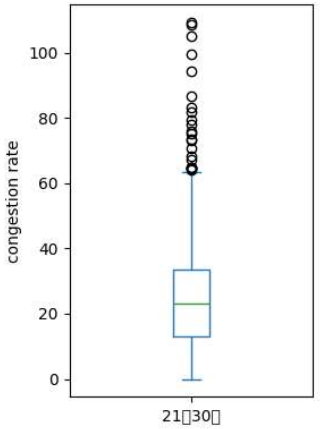

congestion rate
14時30分
15時00分
15時30分
16時00分
16時30分
17時00分
17時30分
18時00分
18時30分
19時00分

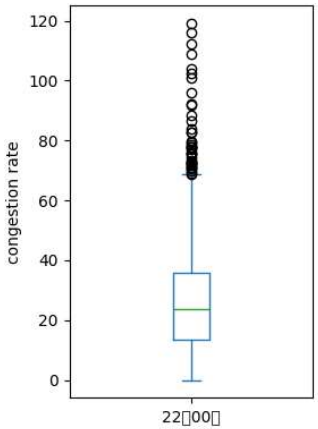

congestion rate
19：30～

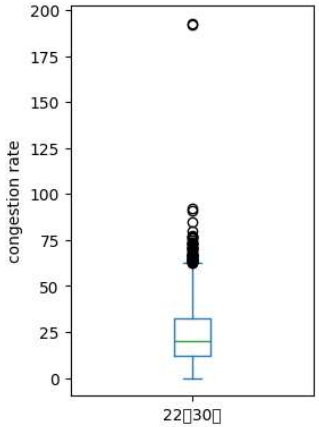

congestion rate
20：00～

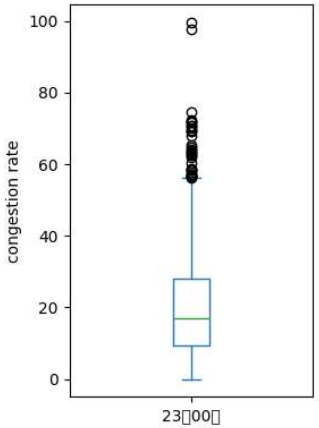

congestion rate
20：30～

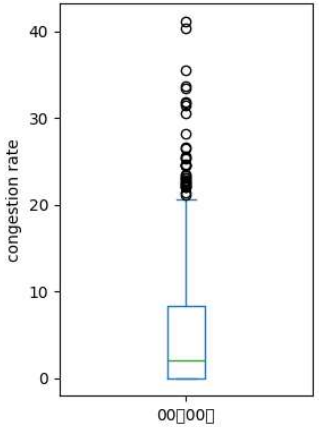

congestion rate
21：00～

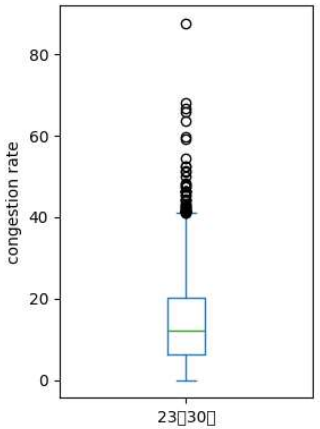

congestion rate
21：30～

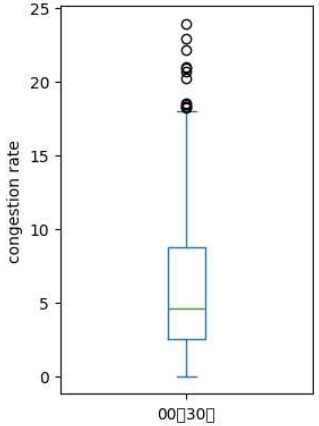

congestion rate
22：00～


congestion rate
22：30～


congestion rate
23：00～


congestion rate
00：00～

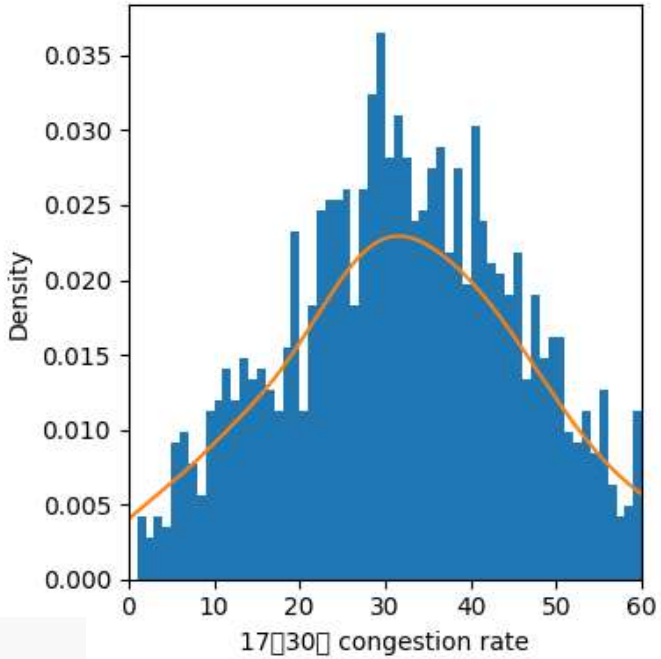

congestion rate
23：30～


congestion rate
00：30～

Exploring Numerical Data

- Histogram and KDE

```
# 히스토그램
for i in range(len(time_list)) :
    ax = (sub_cong_data[time_list[i]]).plot.hist(figsize=(4, 4))
    ax.set_xlabel(time_list[i]+' congestion rate')
    plt.tight_layout()
    plt.show()
```




```
# 히스토그램 & KDE
for i in range(len(time_list)) :
    ax = sub_cong_data[time_list[i]].plot.hist(density=True, xlim=[0, 60],
                                               bins=range(1,61), figsize=(4, 4))
    sub_cong_data[time_list[i]].plot.density(ax=ax)
    ax.set_xlabel(time_list[i]+' congestion rate')
    plt.tight_layout()
    plt.show()
```

KDE 결과: 그래프의 가장 높은 부분이 왼쪽에 쏠려 있다가 가운데로 이동하고, 다시 왼쪽으로 쏠리는 것을 관측 가능.

Exploring Binary and Categorical Data

```
# 범주형 열에서 각 범주의 빈도 계산
day_counts = sub_cong_data['요일구분'].value_counts()
line_counts = sub_cong_data['호선'].value_counts()
updown_counts = sub_cong_data['상하구분'].value_counts()
# 각 범주의 비율을 계산 (전체 대비 %)
day_percentage = 100 * day_counts / day_counts.sum()
line_percentage = 100 * line_counts / line_counts.sum()
updown_percentage = 100 * updown_counts / updown_counts.sum()
# 출력
print(day_percentage)
print(line_percentage)
print(updown_percentage)
```

```
요일구분
평일      33.333333
토요일     33.333333
일요일     33.333333
Name: count, dtype: float64
호선
5    20.263789
2    19.664269
7    15.107914
6    13.309353
3    12.230216
4     9.352518
8     6.474820
1     3.597122
Name: count, dtype: float64
상하구분
하선    40.827338
상선    39.748201
내선     9.712230
외선     9.712230
Name: count, dtype: float64
```

호선별 정보가 불균형하다.

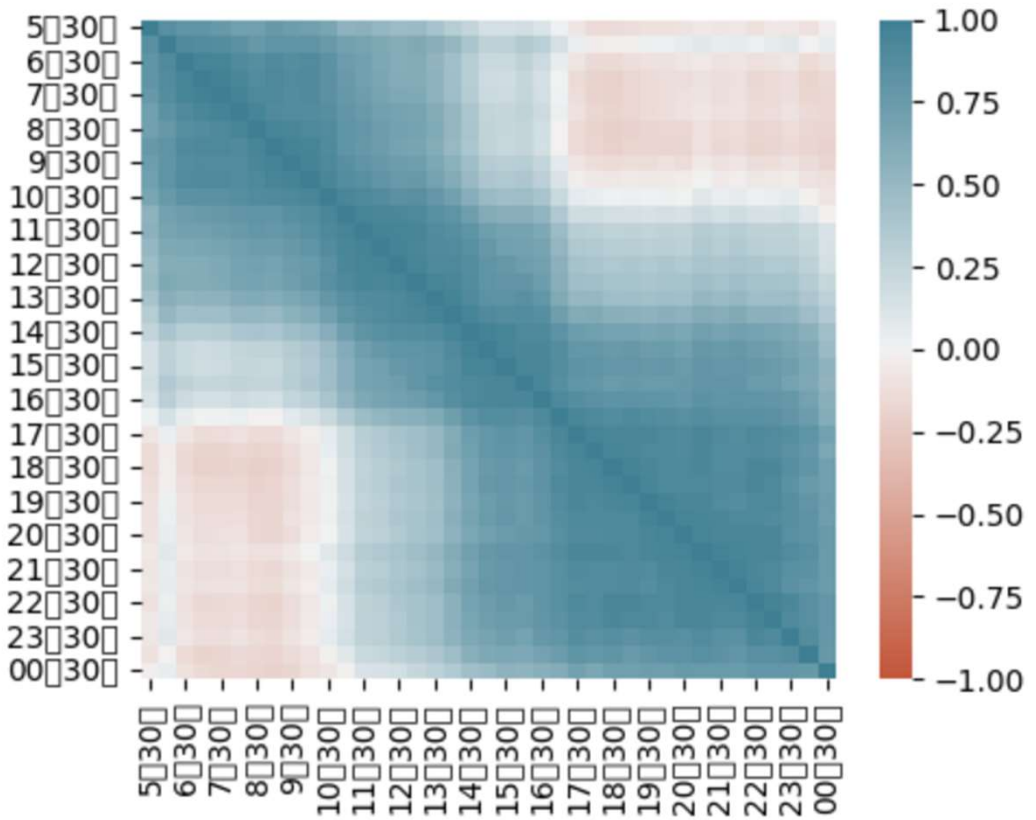
Correlation

```
# 시간대별 승객 수의 상관관계 분석
# 시간대 데이터 선택
time_columns = sub_cong_data.columns[6:]
# 결측치가 있는 행을 삭제
scd_cleaned = sub_cong_data[time_columns].dropna()
# 상관관계 계산
correlation_matrix = scd_cleaned.corr()
print(correlation_matrix)
```

Correlation

	5시30분	6시00분	6시30분	7시00분	7시30분	8시00분	8시30분
5시30분	1.000000	0.856417	0.812332	0.808030	0.772398	0.715876	0.708309
6시00분	0.856417	1.000000	0.901785	0.885021	0.868889	0.830977	0.779983
6시30분	0.812332	0.901785	1.000000	0.958418	0.948781	0.899253	0.865784
7시00분	0.808030	0.885021	0.958418	1.000000	0.976151	0.938867	0.886657
7시30분	0.772398	0.868889	0.948781	0.976151	1.000000	0.966811	0.918697
8시00분	0.715876	0.830977	0.899253	0.938867	0.966811	1.000000	0.949704
8시30분	0.708309	0.779983	0.865784	0.886657	0.918697	0.949704	1.000000
9시00분	0.773178	0.814666	0.898556	0.910404	0.909430	0.901458	0.950736
9시30분	0.728402	0.811608	0.872328	0.891101	0.890424	0.886782	0.932701
10시00분	0.718264	0.819374	0.893938	0.902504	0.899309	0.883860	0.914494
10시30분	0.661200	0.764905	0.830862	0.839615	0.842441	0.861003	0.890746
11시00분	0.552189	0.696506	0.741491	0.758271	0.774459	0.799732	0.817403
11시30분	0.557839	0.677281	0.707657	0.709881	0.733973	0.764327	0.792325
12시00분	0.515126	0.626846	0.655003	0.662084	0.676038	0.717552	0.740900
12시30분	0.474826	0.599357	0.597603	0.614998	0.638410	0.683572	0.701732
13시00분	0.487251	0.632647	0.615620	0.616289	0.631937	0.675027	0.680998
13시30분	0.410788	0.585359	0.536637	0.541199	0.546476	0.605824	0.618266
14시00분	0.325360	0.511968	0.443207	0.448093	0.456265	0.514514	0.512506
14시30분	0.256237	0.393829	0.324179	0.316977	0.335363	0.383929	0.401777
15시00분	0.158065	0.275208	0.212167	0.193993	0.205380	0.250283	0.267013
15시30분	0.159356	0.288578	0.213723	0.186683	0.199614	0.234964	0.239197
16시00분	0.178603	0.346755	0.274682	0.237331	0.244593	0.272736	0.251132
16시30분	0.122281	0.298174	0.200885	0.158327	0.164440	0.197091	0.175773
17시00분	0.012612	0.143743	0.043606	0.003027	0.009689	0.028515	-0.002876
...							
00시00분	0.835016	0.803206	0.857527	0.848691	0.879667	1.000000	0.771441
00시30분	0.757573	0.744625	0.795540	0.796603	0.785334	0.771441	1.000000

[39 rows x 39 columns]



중간점검

- 전체 혼잡도 데이터에 대한 분석을 해봄

- 그러나 실제로 활용하기에는 덜 구체화된 분석

- 너무 뻔한 결과만 얻고 있다.

- 어떻게 하면 좋을까?

→ 각 지하철 호선별 정보를 분석해보자.

호선별 데이터 분석

```
# 각 호선별로 데이터를 분리하여 저장
line_groups = {}

for line in sub_cong_data['호선'].unique():
    line_groups[line] = sub_cong_data[sub_cong_data['호선'] == line]

# line_groups는 각 호선을 key로 가지는 딕셔너리이다.
# 딕셔너리의 값은 해당 호선의 데이터를 담고 있는 데이터프레임이다.
# 예: 1호선의 데이터 접근
print(line_groups[1])
```

호선별 데이터 분석

```python
# 딕셔너리의 각 호선별 데이터프레임에서 혼잡도 70을 넘는 데이터를 필터링
filtered_line_groups = {}

for line, df in line_groups.items():
    congestion_columns = df.columns[6:]  # 혼잡도 관련 열 (6번째 열부터)

    # 혼잡도 70을 넘는 데이터를 포함하는 행만 필터링
    filtered_data = df[(df[congestion_columns] > 70).any(axis=1)]

    # 필터링된 데이터 저장
    filtered_line_groups[line] = filtered_data

    # 각 호선별로 필터링된 데이터의 개수 확인
    print(f"{line}호선 원래 데이터 개수: {len(df)}")
    print(f"{line}호선 필터링된 데이터 개수: {len(filtered_data)}\n")
```

```
1호선 원래 데이터 개수: 60
1호선 필터링된 데이터 개수: 16

2호선 원래 데이터 개수: 328
2호선 필터링된 데이터 개수: 110

3호선 원래 데이터 개수: 204
3호선 필터링된 데이터 개수: 44

4호선 원래 데이터 개수: 156
4호선 필터링된 데이터 개수: 38

5호선 원래 데이터 개수: 338
5호선 필터링된 데이터 개수: 97

6호선 원래 데이터 개수: 222
6호선 필터링된 데이터 개수: 38

7호선 원래 데이터 개수: 252
7호선 필터링된 데이터 개수: 81

8호선 원래 데이터 개수: 108
8호선 필터링된 데이터 개수: 23
```

혼잡도 70을 기준으로 한 이유: 지하철 한 칸이 '혼잡하다'고 말할 수 있는 칸당 승차객 수를 생각해 봄.
모든 좌석이 차 있고 각 좌석의 앞에 사람이 한 명씩 탄 경우 승차객 수는 108명.
승차객 108명일 경우 도출되는 혼잡도 지수 = 67.6 반올림하여 70%를 기준으로 정함.

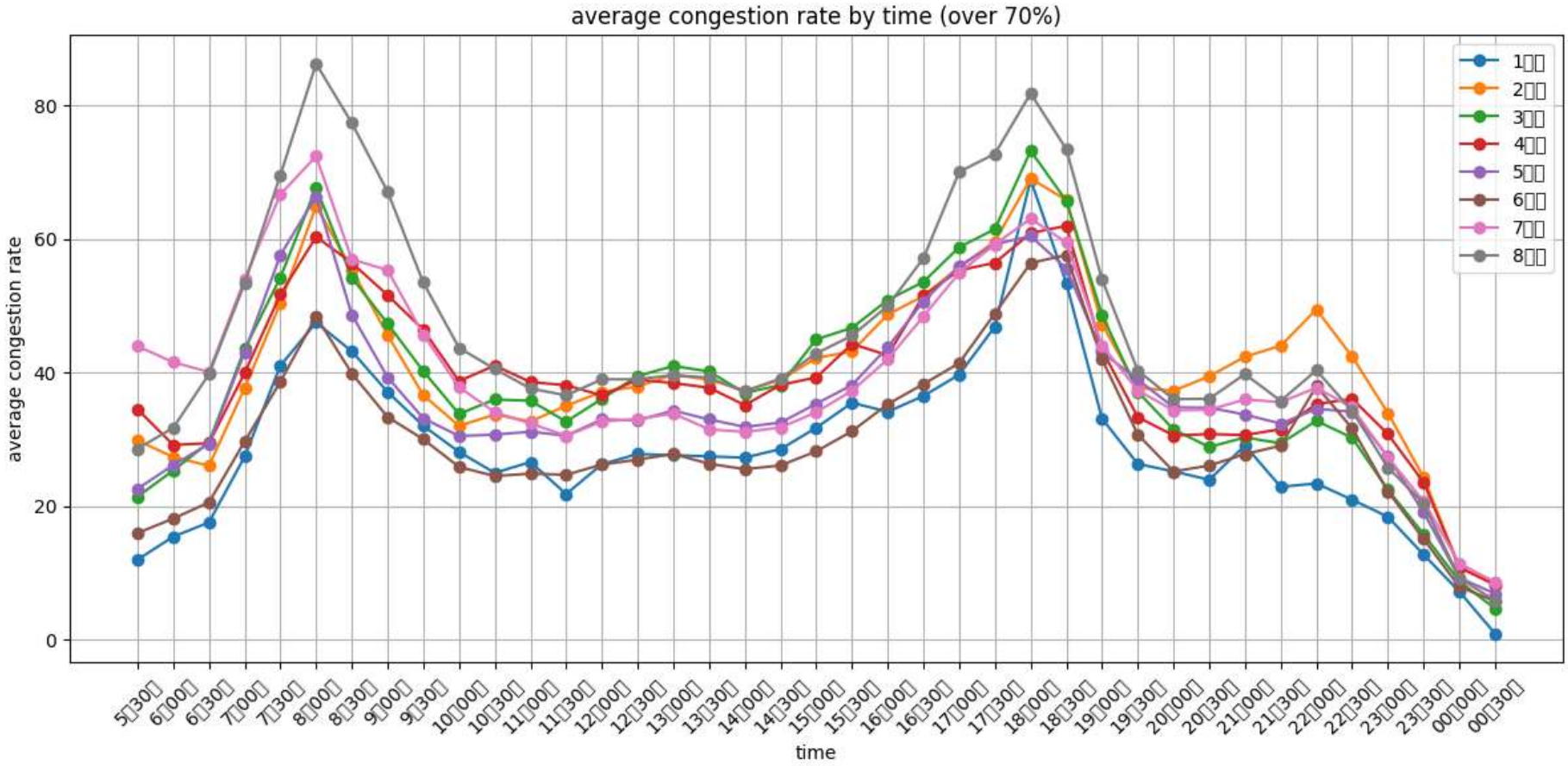
호선별 데이터 분석

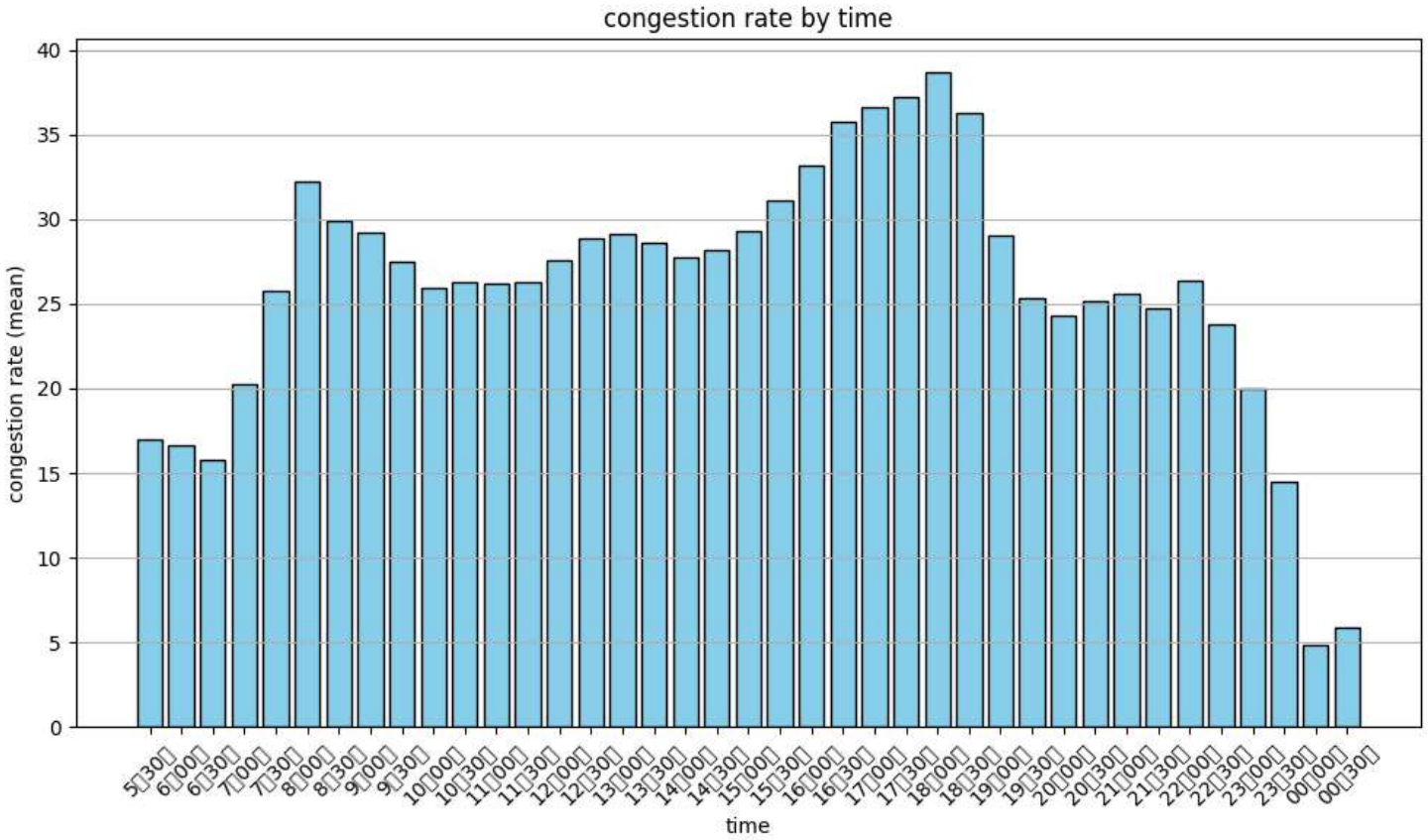
- 이렇게 추출한 데이터의 혼잡도를 평균 내서 그래프로 표현

```
# 시간대별 열을 지정 (혼잡도 관련 열)
time_columns = sub_cong_data.columns[6:]

# 시각화를 위한 figure 생성
plt.figure(figsize=(12, 6))

# 각 호선별로 시간대별 평균 혼잡도 계산 및 시각화
for line, df in filtered_line_groups.items():
    if len(df) > 0:  # 필터링된 데이터가 있는 경우만 처리
        # 시간대별 평균 혼잡도 계산
        mean_congestion_by_time = df[time_columns].mean()

        # 시간대별 평균 혼잡도 선 그래프 그리기
        plt.plot(time_columns, mean_congestion_by_time, marker='o', label=f'{line}호선')

# 그래프 설정
plt.title('average congestion rate by time (over 70%)')
plt.xlabel('time')
plt.ylabel('average congestion rate')
plt.xticks(rotation=45)  # x축 레이블 45도 회전
plt.grid(True)  # 그리드 추가
plt.legend()  # 범례 추가
plt.tight_layout()  # 레이아웃 자동 조정
plt.show()
```

호선별 데이터 분석

시간대별로 변동하는 혼잡도를 직관적으로 파악할 수 있다.





전체 데이터셋을 평균 내어 그린 그래프와 비교해 보면 증감이 훨씬 뚜렷하게 나타난다는 것을 알 수 있다.

호선별 데이터 분석

- 이 그래프를 표현할 수 있는 분포 탐색
- 두 개의 뿔을 가진 듯한 모양새의 그래프
- GMM

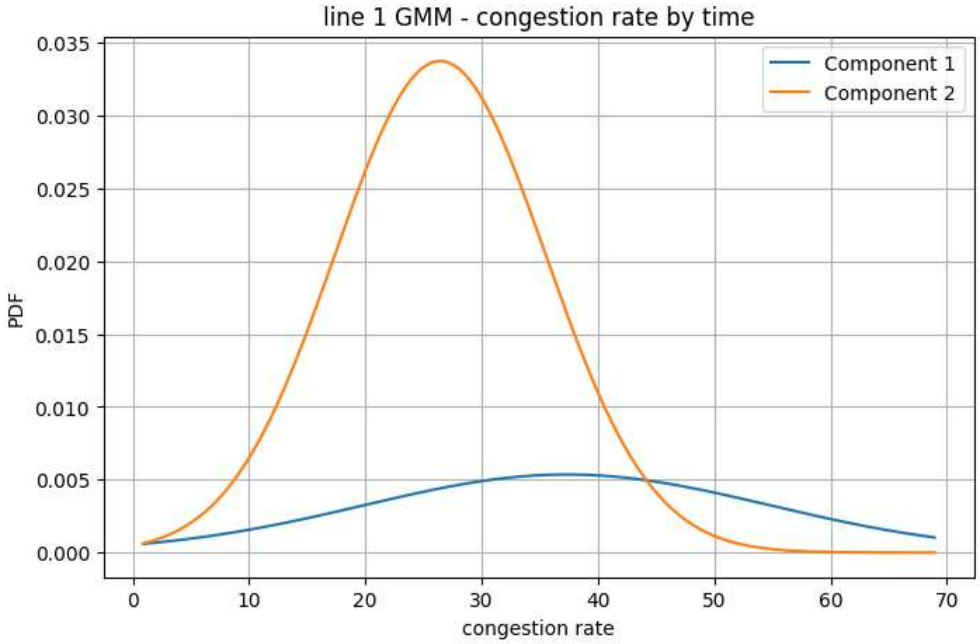
- 서로 다른 여러 개의 정규 분포를 결합한 모델
- 각각의 정규 분포는 데이터의 일부를 설명하며, 전체 데이터는 이 여러 개의 정규 분포가 섞여서 구성됨
- 각 데이터 포인트가 어떤 정규 분포에 속할 가능성이 높은지 계산

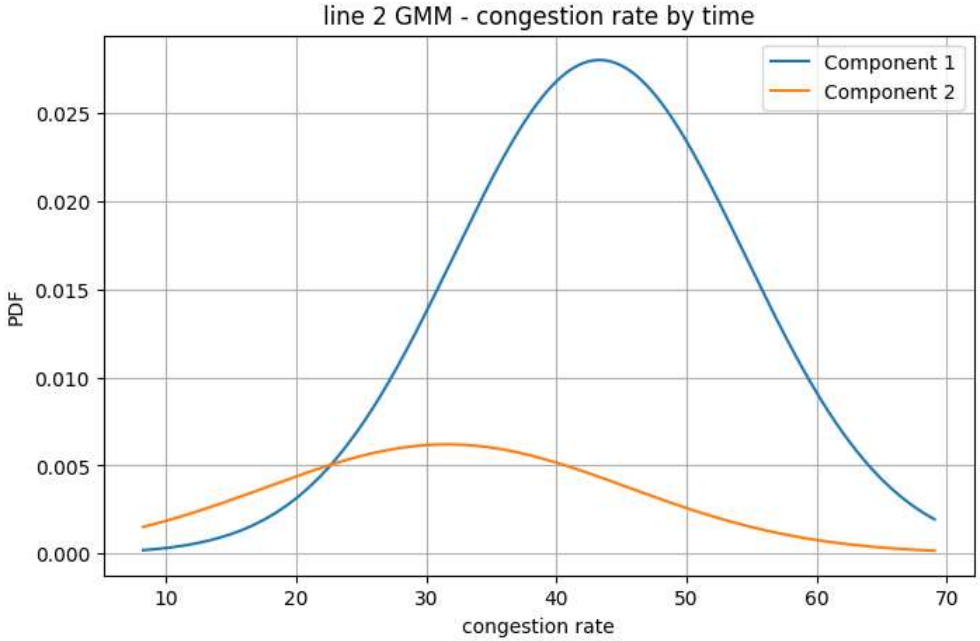
- '출근 시간' 정규 분포와 '퇴근 시간' 정규 분포가 합해진 것으로 가정

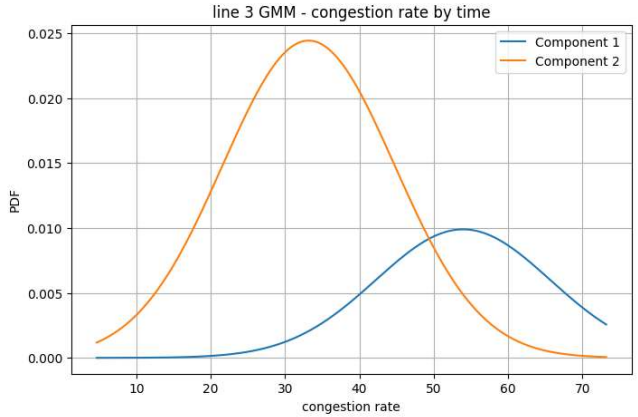
```python
# GMM 모델 적용을 위한 데이터 준비 (필터링된 데이터를 사용)
gmm_results = {}

for line, df in filtered_line_groups.items():
    if len(df) > 0:  # 필터링된 데이터가 있는 경우만 처리
        # 시간대별 평균 혼잡도 계산
        mean_congestion_by_time = df[time_columns].mean().values.reshape(-1, 1)

        # GMM 모델 적용 (정규 분포 2개로 설정: 출퇴근 시간대 피크를 모델링)
        gmm = GaussianMixture(n_components=2, random_state=0)
        gmm.fit(mean_congestion_by_time)

        # GMM으로 생성된 각 정규 분포의 평균 및 표준편차
        means = gmm.means_.flatten()
        stds = np.sqrt(gmm.covariances_).flatten()
        weights = gmm.weights_

        # 신뢰구간 계산
        confidence_intervals = [confidence_interval(means[i], stds[i]) for i in range(2)]

        # 각 호선에 대한 GMM 결과 저장
        gmm_results[line] = {
            'means': means,
            'stds': stds,
            'weights': weights,
            'confidence_intervals': confidence_intervals
        }

        # 결과 출력
        print(f"{line}호선 GMM 결과:")
        for i in range(2):
            print(f"  Component {i+1}:")
            print(f"    Mean: {means[i]:.2f}, Std: {stds[i]:.2f}, Weight: {weights[i]:.2f}")
            print(f"    95% Confidence Interval: {confidence_intervals[i]}")

        # 각 정규 분포를 그래프로 시각화
        plt.figure(figsize=(8, 5))
        x = np.linspace(min(mean_congestion_by_time), max(mean_congestion_by_time), 1000)

        for i in range(2):  # 2개의 정규 분포
            plt.plot(x, gmm.weights_[i] * np.exp(-0.5 * ((x - means[i]) / stds[i])**2) / (stds[i] * np.sqrt(2 * np.pi)),
                     label=f'Component {i+1}')

        plt.title(f'line {line} GMM - congestion rate by time')
        plt.xlabel('congestion rate')
        plt.ylabel('PDF')
        plt.legend()
        plt.grid(True)
        plt.show()
```

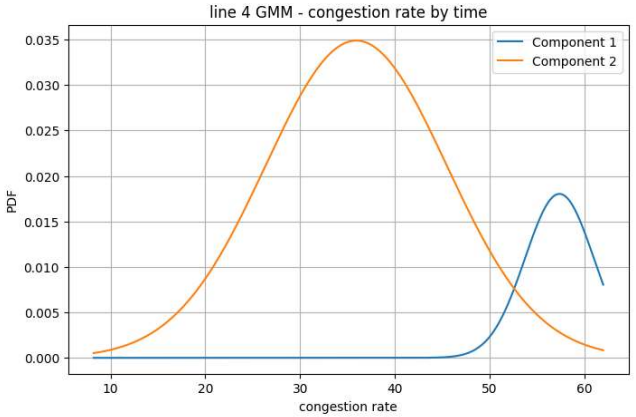
GMM

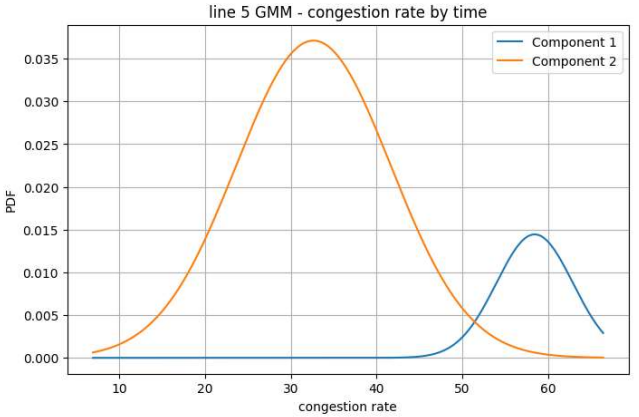
- 두 그래프의 양상이 다르다

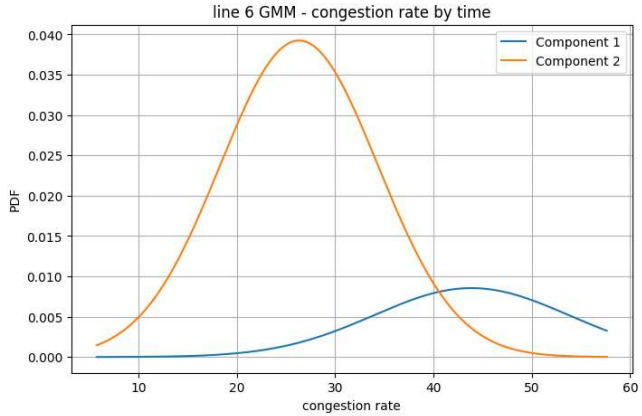


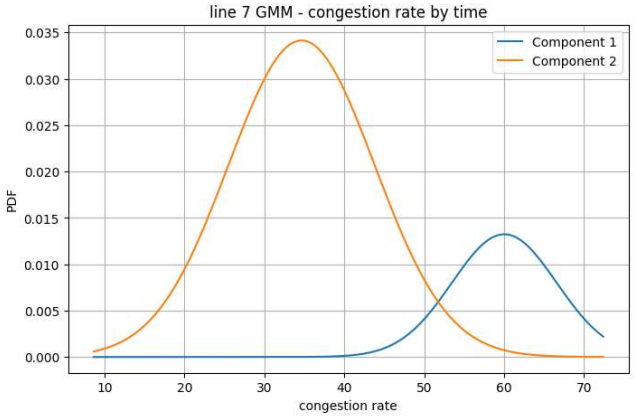


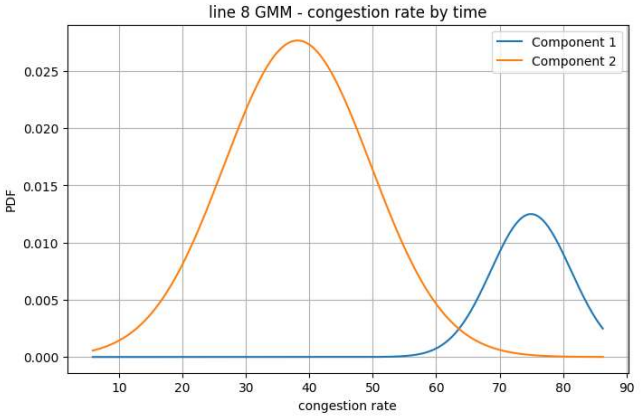












보통은 주황색 그래프가 훨씬 높이 솟아 있는 반면, 2호선은 파란색이 훨씬 높이 솟아 있음.
이는 2호선의 퇴근 시간대 혼잡도가 다른 호선에 비해 매우 높고 집중되어 있다는 것을 시사한다.