지하철 혼잡도 데이터 분석 2

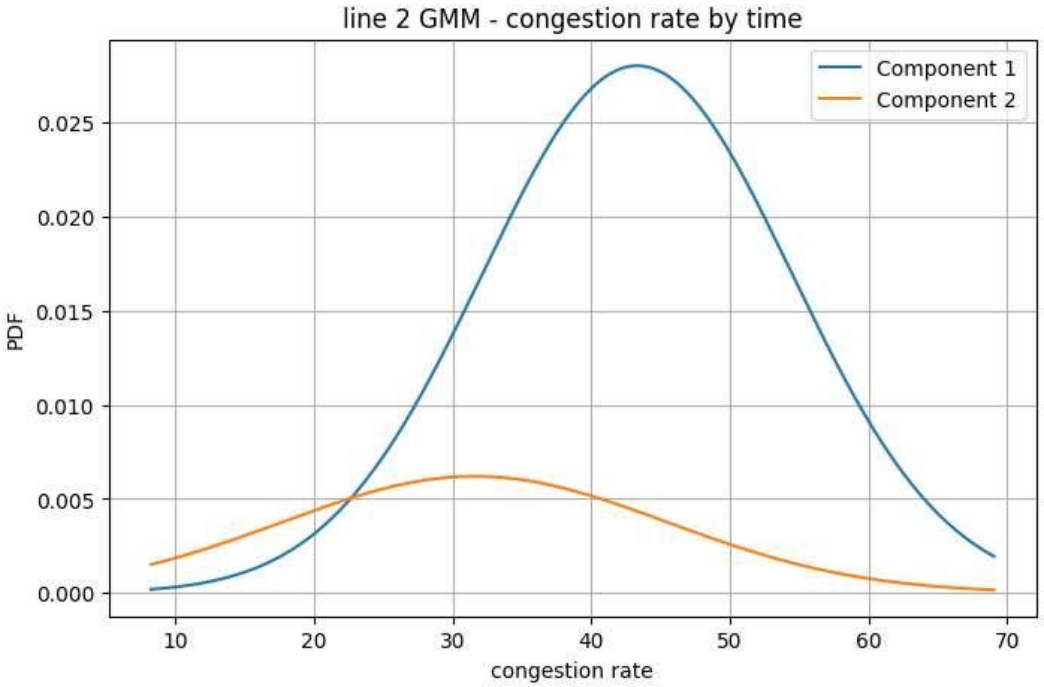
인공지능학과 이지양
인공지능학과 왕성원
인공지능학과 김주현

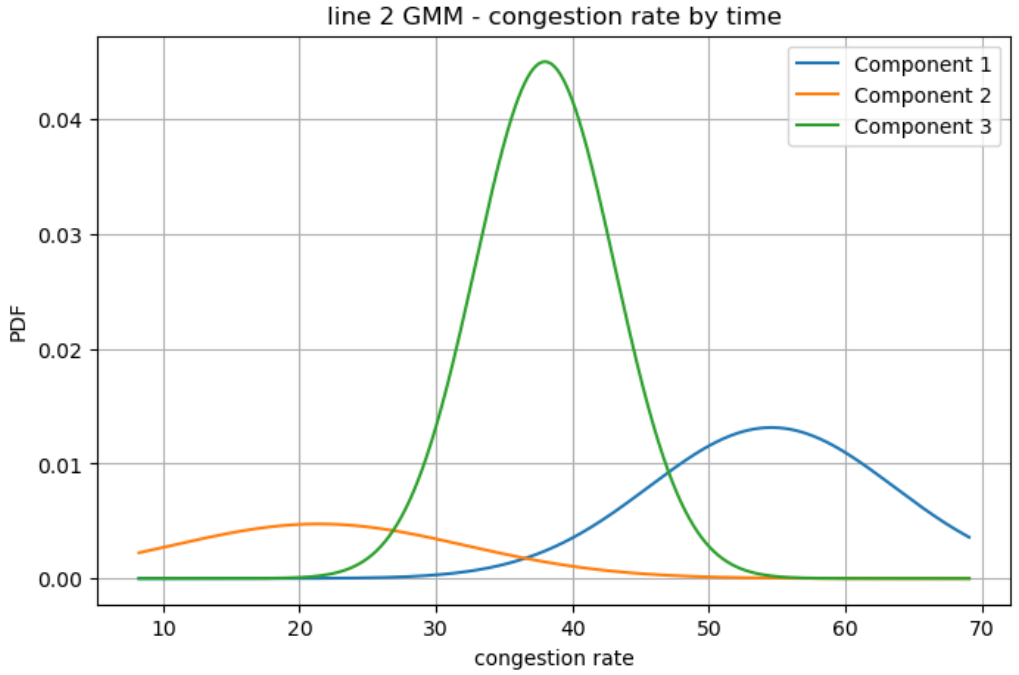
GMM

- 정규 분포 2개 vs 정규 분포 3개

정규 분포 3개일 때)
과적합 발생 가능성 높아짐

3개의 정규분포 합으로 나타낸 그래프들을 보았을 때, 같은 색의 정규 분포 그래프의 위치가 완전히 다른 경우 자주 발생





기계학습 클러스터링을 이용한 승하차 패턴에 따른 서울시 지하철역 분류

사용 데이터 : 2008.01.01~2017.09.30 서울시 지하철역의 시간별 승하차 인원을 기록한 데이터

1. 시각화
 - 각각의 지하철역에 대해 1시간 단위로 10개년의 승하차 인원의 평균을 그래프로 도시
 - 1시간 단위로 데이터가 구성되어 있으므로 시간은 중간값 사용

특징
-각 지하철 역의 시간별 승하차객 그래프에서 몇 개의 피크점이 나타남
-정규분포 곡선과 유사하여 가우시안 모델이 적합할 것이라 예측
-2개의 피크점을 갖는 경우가 가장 일반적이므로 2차 가우시안 혼합 모델 사용

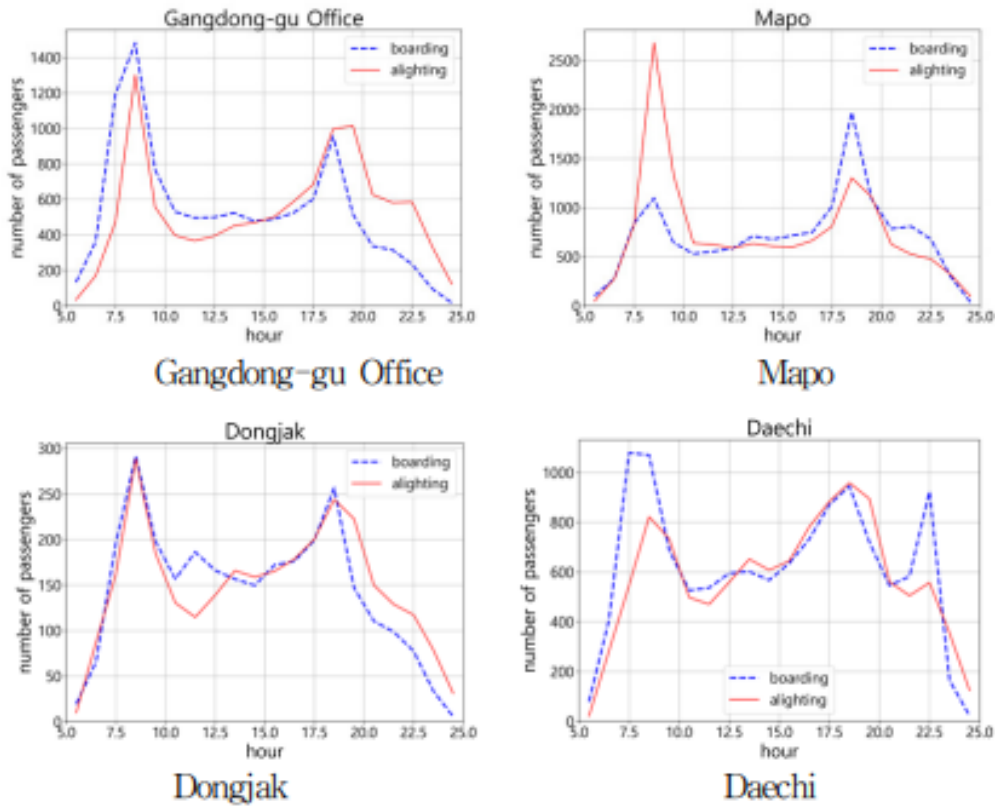


그림 2. 지하철역별 시간별 승하차객 수 시각화

Min, M. (2018) "Classification of Seoul Metro Stations Based on Boarding/ Alighting Patterns Using Machine Learning Clustering," *The Journal of The Institute of Internet, Broadcasting and Communication*. 한국인터넷방송통신학회, 18(4), pp. 13–18. doi: 10.7236/JIIBC.2018.18.4.13.

2. 학습
2.1. 가우시안 혼합
 - 2차 가우시안 혼합 모델을 통해 각 역에서의 시간별 승하차객수를 2개의 정규분포로 보정할 수 있으며, 이는 혼합되어 가우시안 혼합을 형성

 세로축: 그 시각에 승하차객이 발생할 확률
 - on1: 첫 번째 가우시안 커브의 승차 피크 시간
 - on2: 두 번째 가우시안 커브의 승차 피크 시간
 - off1: 첫 번째 가우시안 커브의 하차 피크 시간
 - off2: 두 번째 가우시안 커브의 하차 피크 시간

2.2. K-평균 클러스터링
입력: 223개(지하철역 개수)의 가우시안 혼합으로 생성된 4차원의 데이터 [on1, on2, off1, off2]

단계1) 초기화: 그룹 역그룹0, 역그룹1, 역그룹2, 역그룹3에 해당하는 초기 중심 값 c0, c1, c2, c3를 무작위로 설정(c0, c1, c2, c3는 각각 4차원 벡터)

단계2) 할당: 223개의 입력 데이터에 대하여 가장 가까운 거리에 있는 c0, c1, c2, c3을 찾아서 이에 해당하는 분류 그룹 역그룹0, 역그룹1, 역그룹2, 역그룹3을 배정

단계3) 보정: 같은 분류 그룹 내의 모든 데이터들의 평균 중심 값을 다시 계산하여 c0, c1, c2, c3을 재설정

단계4) 학습이 끝날 때까지 단계2로 돌아가 반복

3. 분류
3.1. 클러스터링 결과
 - 4차원 그래프 형상화는 불가능하므로 2차원으로 나누어 분포도 시각화

 - 16개 조합 중 의미 있는 2차원 분포도 그래프 4가지 : off1(세로축)-on1(가로축) 그래프, off1(세로축)-on2(가로축) 그래프, off2(세로축)-on1(가로축) 그래프, off2(세로축)-on2(가로축) 그래프

 - 축의 눈금: 시간, 가우시안 승차 피크, 하차 피크

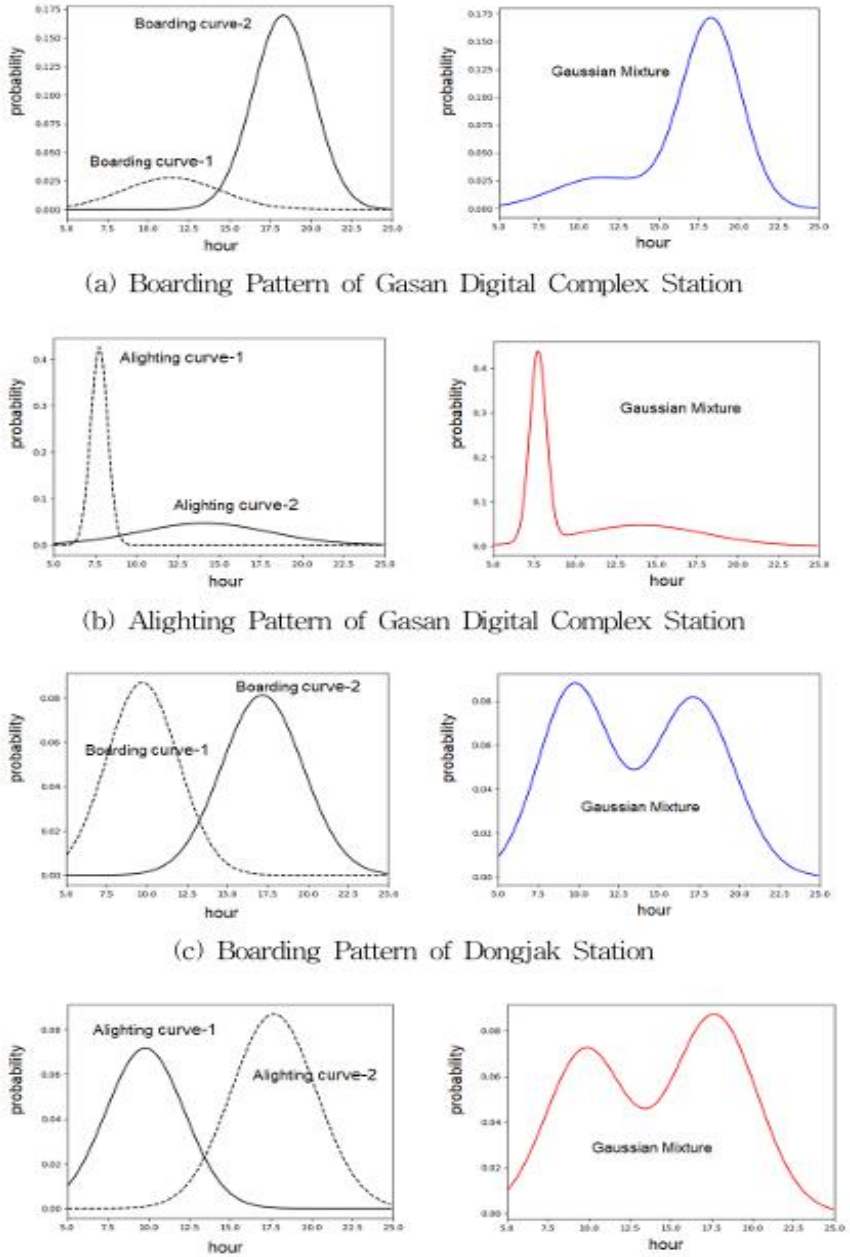


그림 3. 가우시안 혼합에 의한 승차 패턴과 하차 패턴

G0: on1, on2나 off1, off2가 하루 중 넓게 분포하며, 이른 시간에는 나타나지 않음. 저녁 시간에는 승하차가 모두 발생
-> 업무지역은 아니나 사람들의 이동이 많은 역
=> 환승지역 or 상업지역이 많을 것으로 예상

G1: 승차와 하차가 이른 아침, 저녁보다 낮 시간대에 주로 발생
-> 주거, 업무, 상업 어느 쪽으로도 구분하기 어려운 그룹

G2: on1은 오전 이른 시간, off2은 저녁 시간에 집중
-> 주거지

G3: off1이 오전 이른 시간 on1이 저녁 시간에 집중
-> 업무지역

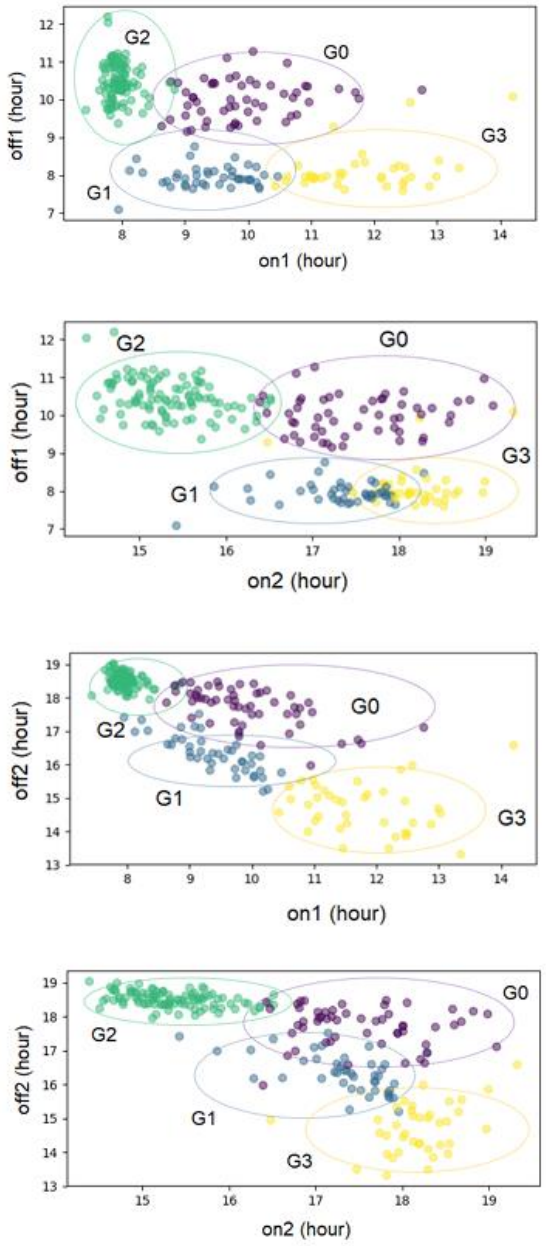


그림 4. 기계학습에 의한 클러스터링 결과

표 1. 서울시 지하철역 분류
Table 1. Classification of Seoul Metro Stations

분류	역 이 름	개수
역 그 룹 0	강변 개롱 개화산 건대입구 고덕 고려대 고속터미널 김포공항 노원 녹사평 당산 대치 독립문 동대문 동대문운동장 동작 마장 마포구청 망원 모란 발산 사당 상수 성신여대입구 송정 신당 신도림 신천 신촌 안암 약수 어린이대공원 영등포시장 오목교 올림픽공원 왕십리 용두 월드컵경기장 이대 이수 이촌 이태원, 일원 잠실 장지 제기동 종합운동장 천호 청량리 총신대입구 학여울 한성대입구 혜화 홍대입구	54
역 그 룹 1	가락시장 강동구청 공덕 광흥창 구로디지털단지 남구로 남태령 내방 대흥 도곡 도림천 디지털미디어시티 뚝섬 마곡 마포 매봉 몽촌토성 문래 문정 반포 방배 방이 버티고개 보라매 복정 상왕십리 서울역 석촌 성수 송파 수서 숙대입구 신길 신설동 애오개 양평 영등포구청 오금 장한평 청구 충정로 합정 효창공원앞	43
역 그 룹 2	강동 거여 공릉 광나루 광명사거리 구산 구의 구파발 군자 굽은다리 금호 길동 길음 까치산, 낙성대 남성 남한산성입구 녹번 단대오거리 답십리 당고개 대림 대청 도봉산 독바위 돌곶이 둔촌동 뚝섬유원지 마들 마천 먹골 면목 명일 목동 무악재 미아 미아삼거리 방화 보문 봉천 봉화산 불광 사가정 산성 상계 상도 상봉 상월곡 상일동 새절 서울대입구 석계 성내 수락산 수유 수진 숭실대입구 신금호 신답 신대방 신대방삼거리 신림 신정 신정네거리 신풍 신흥 쌍문 아차산 아현 암사 양천구청 역촌 연신내 옥수 온수 용답 용마산 우장산 월곡 응암 잠원 장승배기 장암 중계 중곡 중화 증산 지축 창동 창신 철산 태릉입구 하계 행당 홍제 화곡 화랑대	98
역 그 룹 3	가산디지털단지 강남 강남구청 경복궁 광화문 교대 남부터미널 논현 동대문역사문화공원 동대입구 동묘앞 명동 삼각지 삼성 서대문 서초 선릉 시청 신사 신용산 안국 압구정 양재 여의나루 여의도 역삼 을지로3가 을지로4가 을지로입구 종각 종로3가 종로5가 청담 충무로 학동 한강진 한양대 회현	38

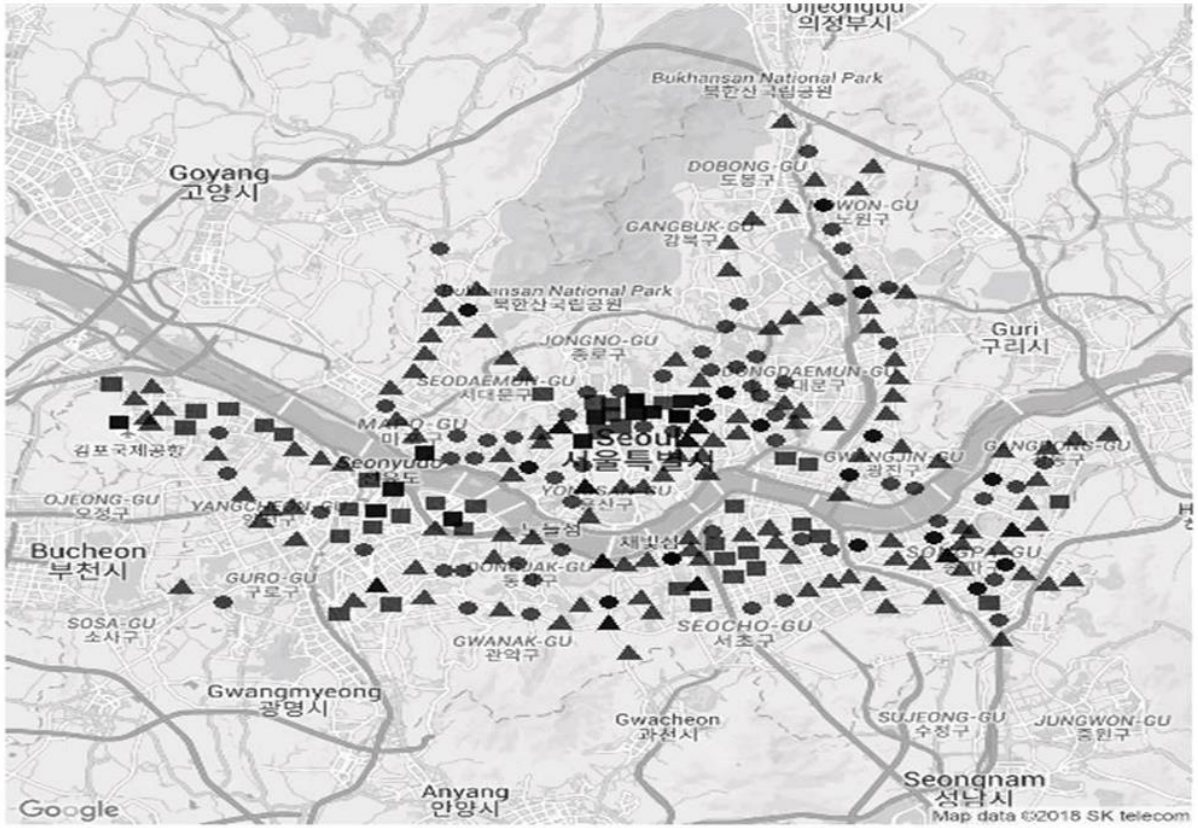
데이터마이닝 기법을 이용한 서울시 지하철역 승차인원 예측

사용 데이터 : 2013.01.01~2018.03.31 서울시 지하철역의 월별 월평균 승차 인원수, 최대 인원수

1. 가우시안 혼합 모형(GMM) 군집화 분석을 실행해 지하철역을 3개 군집으로 유형화

-역관련, 역주변, 행정동 관련 정보들을 설명변수로 활용하여 GMM 군집분석 실행

- 군집 분석 결과
그룹 1: 상공업지구 (상업지구 비율 월등, 공업지구 비율 ↑)
그룹 2: 주상복합지구 (주거지구,상업지구 비율 ↑)
그룹 3: 주거지구



그룹 1 (●) 그룹 2 (■) 그룹 3 (▲)

Figure 3.1. Location of subway stations by group.

Cho, S. J., Kim, B. G., Kim, N. H., Song, J. W., 2019, A Study on the number of passengers using the subway stations in seoul, Kor. J. Appl. Stat., 32, 111-128.

2. 그룹별로 데이터마이닝 기법을 이용해 지하철 승차 인원 예측 모형 후보 선택

-훈련 데이터 : 2013년 1월~2017년 3월 데이터
-선형 모형과 비선형 모형으로 10겹 교차검정 100번 실시

2.1. 월평균 시간당 평균 승차 인원 예측 모형

Table 3.2. Test root mean squared error of each model using 10-fold cross validation (Model 1)

		전체	그룹 1	그룹 2	그룹 3
Linear	Linear (stepwise)	0.5746	0.9920	0.7465	0.5990
	Ridge	**0.5688**	0.7440	0.6722	0.5864
	LASSO	0.5742	0.9698	0.7412	0.5945
	Poisson GLM (stepwise)	0.5888	0.8404	0.6846	0.6030
Non linear	Random forest	0.5977	**0.7434**	**0.5687**	**0.5598**
	Gredient boosting	0.6984	0.8127	0.6057	0.6756
	Support vector machine	0.6210	0.7757	0.6094	0.7406
	Extreme gradient boosting	0.6011	0.7628	0.5992	0.5654

RMSE 기준으로 예측 모형 평가.
값이 작을수록 우수한 모형.

-전체 데이터에서는 선형 회귀 모형이,
그룹별 데이터에서는 랜덤포레스트가
최적 모형

2.2. 월평균 시간당 최대 승차 인원 예측 모형

Table 3.3. Test root mean squared error of each model using 10-fold cross validation (Model 2)

		전체	그룹 1	그룹 2	그룹 3
Linear	Linear (stepwise)	0.6185	1.0458	0.7919	0.6468
	Ridge	**0.6147**	**0.7612**	0.7219	0.6346
	LASSO	0.6181	1.0265	0.7857	0.6440
	Poisson GLM (stepwise)	0.6340	0.8690	0.7727	0.6605
Non linear	Random forest	0.6458	0.7842	**0.6027**	**0.6142**
	Gredient boosting	0.7328	0.8489	0.6181	0.7178
	Support vector machine	0.6911	0.8671	0.6999	0.7711
	Extreme gradient boosting	0.6454	0.8447	0.6511	0.6187

-전체 데이터에서는 선형 회귀 모형
-그룹별 군집화 데이터에서는 랜덤포
레스트가 최적 모형

=>테스트 데이터에 활용할 모델 후보 :
선형 회귀, 랜덤포레스트, 익스트림 그
래디언트 부스팅

3. 테스트 데이터 최종 예측 결과
-2017년 4월에서 2018년 3월의 월평균 시간당 평균 및 최대 승차 인원 예측

Table 3.5. Test root mean squared error of each group (Model 2)

	전체	그룹 1		그룹 2	그룹 3
		전체모형	군집화 모형		
Linear (stepwise)	0.5427	0.5314	0.4635	0.4636	0.5107
Random forest	**0.1086**	**0.1252**	**0.1407**	**0.1039**	**0.1046**
Extreme gradient boosting	0.2640	0.2719	0.2079	0.1712	0.2037

그룹별 최종 후보 모형의 승차인원 오차 vs 그룹별 평균 모형을 이용해 예측한 승차인원 오차 시 최종 후보 모형들의 예측력이 7~10배 더 좋음을 확인할 수 있음

-그룹 1 : 전체 테스트 데이터를 활용한 모형에서 더 성능이 좋음(랜덤포레스트 전체 모형)
-그룹 2,3 : 랜덤포레스트 군집화 모형

시사점
-지하철의 효과적인 운영에 중요한 것은 시간당 평균 승차 인원수와 시간당 최대 승차 인원수

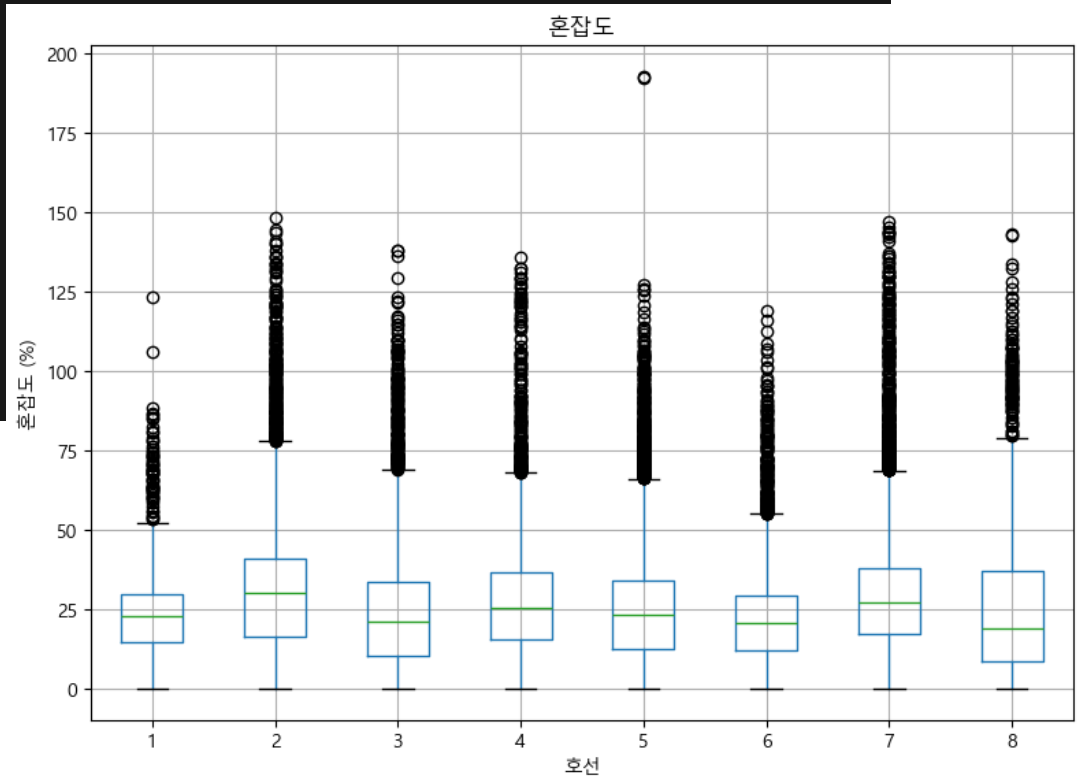
-GMM 군집 분석의 다른 활용 방안 습득
우리의 분석에서는 데이터를 정규분포의 합으로 표현하는 데 그쳤으나 지하철 역을 유형화하는 등의 데이터 분류도 가능하다는 것을 알게 됨

-GMM의 최적화 방안
논문에서는 GMM 군집화 분석에서 BIC(Bayesian Information Criterion) 값을 계산해 최적 군집 계수를 결정 -> 우리도 BIC를 사용했더라면 2개의 정규 분포 합으로 나타내는 것에 설득력을 얻을 수 있었을 것임

Chapter 3. Statistial Experiments and Significance Testing

- Resampling

```
# 시간대 열만 선택하여 melt 함수로 세로 길게 변환
time_columns = sub_cong_data.columns[6:]  # 혼잡도 데이터가 포함된 시간대 열만 선택
melted_data = sub_cong_data.melt(id_vars=['호선'], value_vars=time_columns, var_name='시간대', value_name='혼잡도')

# 호선별 혼잡도 박스플롯 생성
ax = melted_data.boxplot(by='호선', column='혼잡도', figsize=(8, 6))
ax.set_xlabel('호선')
ax.set_ylabel('혼잡도 (%)')
plt.suptitle('')

plt.tight_layout()
plt.show()
```



```python
# 모든 호선 조합의 혼잡도 평균 차이의 절댓값 계산
mean_diffs_all_pairs_abs = {}

# 1호선부터 8호선까지의 모든 조합에 대해 평균 혼잡도 차이의 절댓값 계산
unique_lines = melted_data['호선'].unique()
for line1, line2 in combinations(unique_lines, 2):
    mean_line1 = melted_data[melted_data['호선'] == line1]['혼잡도'].mean()
    mean_line2 = melted_data[melted_data['호선'] == line2]['혼잡도'].mean()
    mean_diffs_all_pairs_abs[f"{line1}호선 vs {line2}호선"] = abs(mean_line2 - mean_line1)
```

```
{'1호선 vs 2호선': 7.060607752169325,
 '1호선 vs 3호선': 0.6265618450498529,
 '1호선 vs 4호선': 4.2219553745162735,
 '1호선 vs 5호선': 1.255331055460914,
 '1호선 vs 6호선': 1.5131966210968315,
 '1호선 vs 7호선': 6.126335274021688,
 '1호선 vs 8호선': 1.0674567079897415,
 '2호선 vs 3호선': 6.434045907119472,
 '2호선 vs 4호선': 2.8386523776530517,
 '2호선 vs 5호선': 5.805276696708411,
 '2호선 vs 6호선': 8.573804373266157,
 '2호선 vs 7호선': 0.934272478147637,
 '2호선 vs 8호선': 5.993151044179584,
 '3호선 vs 4호선': 3.5953935294664205,
 '3호선 vs 5호선': 0.6287692104110612,
 '3호선 vs 6호선': 2.1397584661466844,
 '3호선 vs 7호선': 5.499773428971835,
 '3호선 vs 8호선': 0.44089486293988855,
 '4호선 vs 5호선': 2.9666243190553594,
 '4호선 vs 6호선': 5.735151995613105,
 '4호선 vs 7호선': 1.9043798995054146,
 '4호선 vs 8호선': 3.154498666526532,
 '5호선 vs 6호선': 2.7685276765577456,
 '5호선 vs 7호선': 4.871004218560774,
 '5호선 vs 8호선': 0.1878743474711726,
 '6호선 vs 7호선': 7.63953189511852,
 '6호선 vs 8호선': 2.580653329086573,
 '7호선 vs 8호선': 5.058878566031947}
```

Statistical Significance and P-Values

```python
# 순열 검정 함수 정의
def perm_fun(x, nA, nB):
    n = nA + nB
    idx_B = set(random.sample(range(n), nB))
    idx_A = set(range(n)) - idx_B
    return x.loc[list(idx_B)].mean() - x.loc[list(idx_A)].mean()

# 모든 호선 조합에 대한 순열 검정 및 시각화
unique_lines = melted_data['호선'].unique()
p_values = {}

for line1, line2 in combinations(unique_lines, 2):
    # 각 호선의 혼잡도 데이터 추출
    line1_data = melted_data[melted_data['호선'] == line1]['혼잡도']
    line2_data = melted_data[melted_data['호선'] == line2]['혼잡도']

    # 각 호선의 데이터 크기
    nA = line1_data.shape[0]
    nB = line2_data.shape[0]

    # 결합된 데이터와 관측된 평균 차이 계산 (백분율로 변환했더니 차이 커서 절댓값)
    combined_data = pd.concat([line1_data, line2_data]).reset_index(drop=True)
    obs_pct_diff = line2_data.mean() - line1_data.mean()
    print(f"Observed difference for {line1}호선 vs {line2}호선: {obs_pct_diff:.4f}%")

    # 순열 검정 실행
    random.seed(1)
    perm_diffs = [100 * perm_fun(combined_data, nA, nB) for _ in range(1000)]
    perm_diffs = np.array(perm_diffs)  # numpy 배열로 변환

    # p-value 계산
    p_value = np.mean(perm_diffs > obs_pct_diff)
    p_values[f"{line1}호선 vs {line2}호선"] = p_value

    # 시각화
    fig, ax = plt.subplots(figsize=(5, 5))
    ax.hist(perm_diffs, bins=11, rwidth=0.9)
    ax.axvline(x=obs_pct_diff, color='black', lw=2)
    ax.text(obs_pct_diff + 0.1, max(ax.get_ylim()) * 0.8, 'Observed\ndifference', bbox={'facecolor':'white'})
    ax.set_xlabel('혼잡도 차이 (percent)')
    ax.set_ylabel('Frequency')
    plt.title(f"Permutation Test for {line1}호선 vs {line2}호선")
    plt.tight_layout()
    plt.show()
```

```
1호선 vs 2호선: p-value for single sided test: 0.1647
1호선 vs 3호선: p-value for single sided test: 0.0003
1호선 vs 4호선: p-value for single sided test: 0.0004
1호선 vs 5호선: p-value for single sided test: 0.0776
1호선 vs 6호선: p-value for single sided test: 0.0624
1호선 vs 7호선: p-value for single sided test: 0.0001
1호선 vs 8호선: p-value for single sided test: 0.0000
2호선 vs 3호선: p-value for single sided test: 0.0000
2호선 vs 4호선: p-value for single sided test: 0.0000
2호선 vs 5호선: p-value for single sided test: 0.0000
2호선 vs 6호선: p-value for single sided test: 0.0000
2호선 vs 7호선: p-value for single sided test: 0.0000
2호선 vs 8호선: p-value for single sided test: 0.0000
3호선 vs 4호선: p-value for single sided test: 0.4826
3호선 vs 5호선: p-value for single sided test: 0.0003
3호선 vs 6호선: p-value for single sided test: 0.0020
3호선 vs 7호선: p-value for single sided test: 0.3008
3호선 vs 8호선: p-value for single sided test: 0.0024
4호선 vs 5호선: p-value for single sided test: 0.0007
4호선 vs 6호선: p-value for single sided test: 0.0031
4호선 vs 7호선: p-value for single sided test: 0.3387
4호선 vs 8호선: p-value for single sided test: 0.0044
5호선 vs 6호선: p-value for single sided test: 0.3942
5호선 vs 7호선: p-value for single sided test: 0.0000
5호선 vs 8호선: p-value for single sided test: 0.0000
6호선 vs 7호선: p-value for single sided test: 0.0002
6호선 vs 8호선: p-value for single sided test: 0.0000
7호선 vs 8호선: p-value for single sided test: 0.0067
```

p-value가 0.0000인 조합 :
1호선 vs 8호선, 3호선 vs 4호선
혼잡도 차이가 매우 유의미하다는 것을 의미

p-value가 0.3942인 조합 :
5호선 vs 8호선
혼잡도 차이가 통계적으로 유의미하지 않다는 것을 의미

1호선 vs 8호선, 3호선 vs 4호선
=>두 호선의 혼잡도 차이가 확실히 다르다

5호선 vs 8호선
=>두 호선의 혼잡도 차이가 다르다고 할 수 없다

t-Tests

```python
# 모든 호선 조합에 대해 t-검정 수행
p_values_ttest = {}

for line1, line2 in combinations(melted_data['호선'].unique(), 2):  # 고유한 호선 번호 조합 생성
    # 각 호선의 혼잡도 데이터 추출 후 NaN 제거
    line1_data = melted_data[melted_data['호선'] == line1]['혼잡도'].dropna()
    line2_data = melted_data[melted_data['호선'] == line2]['혼잡도'].dropna()

    # 데이터가 충분하지 않은 경우 건너뛰기
    if len(line1_data) < 2 or len(line2_data) < 2:
        p_values_ttest[f"{line1}호선 vs {line2}호선"] = "Not enough data"
        continue

    # 독립 두 표본 t-검정 (단측 검정)
    res = stats.ttest_ind(line1_data, line2_data, equal_var=False)

    # 단측 검정 p-value 계산
    p_value_single_sided = res.pvalue / 2
    p_values_ttest[f"{line1}호선 vs {line2}호선"] = p_value_single_sided
```

```
1호선 vs 2호선: p-value for single sided test: 1.7384802897210583e-103
1호선 vs 3호선: p-value for single sided test: 0.030828620122245
1호선 vs 4호선: p-value for single sided test: 1.4317059929736563e-32
1호선 vs 5호선: p-value for single sided test: 2.3142426300118107e-05
1호선 vs 6호선: p-value for single sided test: 4.3342518062998777e-07
1호선 vs 7호선: p-value for single sided test: 7.096829586616522e-74
1호선 vs 8호선: p-value for single sided test: 0.006002300795276843
2호선 vs 3호선: p-value for single sided test: 1.918603502879084e-127
2호선 vs 4호선: p-value for single sided test: 8.603463651221887e-23
2호선 vs 5호선: p-value for single sided test: 1.8195869054897162e-138
2호선 vs 6호선: p-value for single sided test: 7.002496140414775e-296
2호선 vs 7호선: p-value for single sided test: 0.0001836087151568379
2호선 vs 8호선: p-value for single sided test: 2.097703552146938e-57
3호선 vs 4호선: p-value for single sided test: 1.3975652962666778e-31
3호선 vs 5호선: p-value for single sided test: 0.006420956028516703
3호선 vs 6호선: p-value for single sided test: 1.0525817819878576e-17
3호선 vs 7호선: p-value for single sided test: 3.5050022484190114e-84
3호선 vs 8호선: p-value for single sided test: 0.1271559904191421
4호선 vs 5호선: p-value for single sided test: 1.0085544947454662e-26
4호선 vs 6호선: p-value for single sided test: 1.9339723973984228e-93
4호선 vs 7호선: p-value for single sided test: 2.1818535650303087e-10
4호선 vs 8호선: p-value for single sided test: 3.2118376370595286e-15
5호선 vs 6호선: p-value for single sided test: 1.9013979783200347e-38
5호선 vs 7호선: p-value for single sided test: 6.228646365382577e-85
5호선 vs 8호선: p-value for single sided test: 0.3025179099262457
6호선 vs 7호선: p-value for single sided test: 1.1927602760761404e-203
6호선 vs 8호선: p-value for single sided test: 6.193971815431453e-13
7호선 vs 8호선: p-value for single sided test: 1.850479507346897e-39
```

모든 조합에서 p-value가 0에 가까운 모습
순열 검정의 p-value와 큰 차이를 보임

순열 검정 vs T-검정

```
1호선 vs 2호선: p-value for single sided test: 0.1647
1호선 vs 3호선: p-value for single sided test: 0.0003
1호선 vs 4호선: p-value for single sided test: 0.0004
1호선 vs 5호선: p-value for single sided test: 0.0776
1호선 vs 6호선: p-value for single sided test: 0.0624
1호선 vs 7호선: p-value for single sided test: 0.0001
1호선 vs 8호선: p-value for single sided test: 0.0000
2호선 vs 3호선: p-value for single sided test: 0.0000
2호선 vs 4호선: p-value for single sided test: 0.0000
2호선 vs 5호선: p-value for single sided test: 0.0000
2호선 vs 6호선: p-value for single sided test: 0.0000
2호선 vs 7호선: p-value for single sided test: 0.0000
2호선 vs 8호선: p-value for single sided test: 0.0000
3호선 vs 4호선: p-value for single sided test: 0.4826
3호선 vs 5호선: p-value for single sided test: 0.0003
3호선 vs 6호선: p-value for single sided test: 0.0020
3호선 vs 7호선: p-value for single sided test: 0.3008
3호선 vs 8호선: p-value for single sided test: 0.0024
4호선 vs 5호선: p-value for single sided test: 0.0007
4호선 vs 6호선: p-value for single sided test: 0.0031
4호선 vs 7호선: p-value for single sided test: 0.3387
4호선 vs 8호선: p-value for single sided test: 0.0044
5호선 vs 6호선: p-value for single sided test: 0.3942
5호선 vs 7호선: p-value for single sided test: 0.0000
5호선 vs 8호선: p-value for single sided test: 0.0000
6호선 vs 7호선: p-value for single sided test: 0.0002
6호선 vs 8호선: p-value for single sided test: 0.0000
7호선 vs 8호선: p-value for single sided test: 0.0067
```

```
1호선 vs 2호선: p-value for single sided test: 1.7384802897210583e-103
1호선 vs 3호선: p-value for single sided test: 0.030828620122245
1호선 vs 4호선: p-value for single sided test: 1.4317059929736563e-32
1호선 vs 5호선: p-value for single sided test: 2.3142426300118107e-05
1호선 vs 6호선: p-value for single sided test: 4.3342518062998777e-07
1호선 vs 7호선: p-value for single sided test: 7.096829586616522e-74
1호선 vs 8호선: p-value for single sided test: 0.006002300795276843
2호선 vs 3호선: p-value for single sided test: 1.918603502879084e-127
2호선 vs 4호선: p-value for single sided test: 8.603463651221887e-23
2호선 vs 5호선: p-value for single sided test: 1.8195869054897162e-138
2호선 vs 6호선: p-value for single sided test: 7.002496140414775e-296
2호선 vs 7호선: p-value for single sided test: 0.0001836087151568379
2호선 vs 8호선: p-value for single sided test: 2.097703552146938e-57
3호선 vs 4호선: p-value for single sided test: 1.3975652962666778e-31
3호선 vs 5호선: p-value for single sided test: 0.006420956028516703
3호선 vs 6호선: p-value for single sided test: 1.0525817819878576e-17
3호선 vs 7호선: p-value for single sided test: 3.5050022484190114e-84
3호선 vs 8호선: p-value for single sided test: 0.1271559904191421
4호선 vs 5호선: p-value for single sided test: 1.0085544947454662e-26
4호선 vs 6호선: p-value for single sided test: 1.9339723973984228e-93
4호선 vs 7호선: p-value for single sided test: 2.1818535650303087e-10
4호선 vs 8호선: p-value for single sided test: 3.2118376370595286e-15
5호선 vs 6호선: p-value for single sided test: 1.9013979783200347e-38
5호선 vs 7호선: p-value for single sided test: 6.228646365382577e-85
5호선 vs 8호선: p-value for single sided test: 0.3025179099262457
6호선 vs 7호선: p-value for single sided test: 1.1927602760761404e-203
6호선 vs 8호선: p-value for single sided test: 6.193971815431453e-13
7호선 vs 8호선: p-value for single sided test: 1.850479507346897e-39
```

순열 검정의 가정 : x (비모수적 방법)

T-검정의 가정 : 데이터가 정규분포를 따름

순열 검정은 데이터를 무작위로 섞어 가상의 평균 차이로 검정

T-검정은 두 그룹의 평균 차이를 표준화해 p-value 계산

=>지하철 혼잡도 데이터는 시간대와 혼잡도의 관련도 ↑, 데이터 분포가 정규분포 양상을 띠므로 순열 검정에서 좋은 결과를 얻을 수 없음

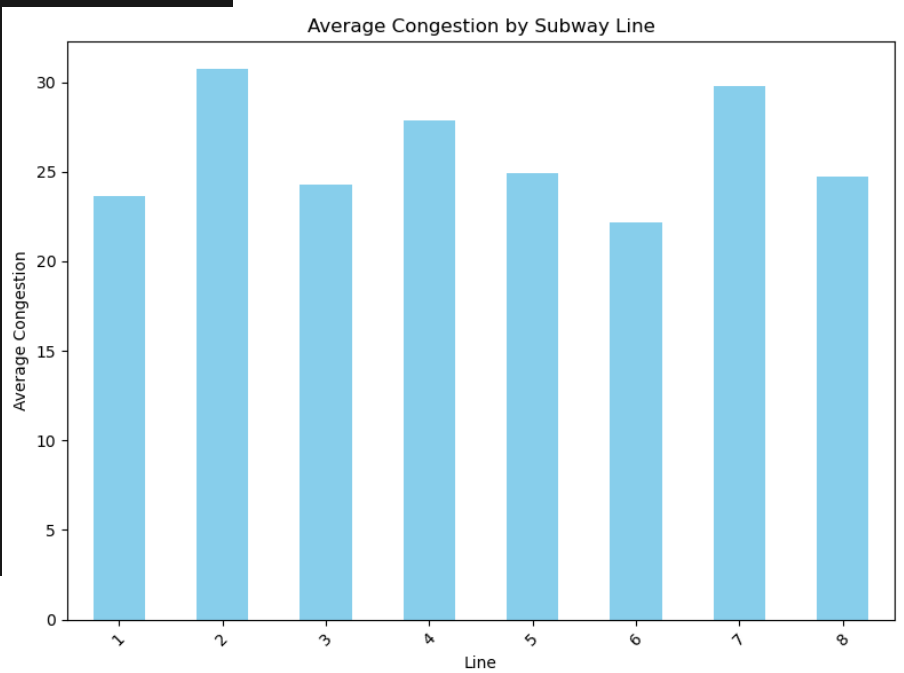
ANOVA

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

# 시간대 열만 선택하여 melt 함수로 세로 길게 변환
time_columns = sub_cong_data.columns[6:]  # 혼잡도 데이터가 포함된 시간대 열만 선택
melted_data = sub_cong_data.melt(id_vars=['호선'], value_vars=time_columns, var_name='시간대', value_name='혼잡도'

# 관측된 호선별 평균 혼잡도 분산 계산
observed_variance = melted_data.groupby('호선')['혼잡도'].mean().var()
print('Observed means:', melted_data.groupby('호선')['혼잡도'].mean().values)
print('Variance:', observed_variance)

# 순열 검정 함수 정의
def perm_test(df):
    df = df.copy()
    df['혼잡도'] = np.random.permutation(df['혼잡도'].values)
    return df.groupby('호선')['혼잡도'].mean().var()

# 순열 검정 실행
print('Permutation test variance:', perm_test(melted_data))
```

```
Observed means: [23.66312582 30.72373357 24.28968766 27.88508119 24.91845687 22.1499292
 29.78946109 24.73058252]
Variance: 9.459248759794702
Permutation test variance: 0.0837333627000491
```

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

# 시간대 열만 선택하여 melt 함수로 세로 길게 변환
time_columns = sub_cong_data.columns[6:]  # 혼잡도 데이터가 포함된 시간대 열만 선택
melted_data = sub_cong_data.melt(id_vars=['호선'], value_vars=time_columns, var_name='시간대', value_name='혼잡도')

# ANOVA 모델 설정
anova_model = ols('혼잡도 ~ C(호선)', data=melted_data).fit()
anova_table = sm.stats.anova_lm(anova_model, typ=2)

# ANOVA 결과 출력
print("ANOVA Table:")
print(anova_table)

# p-value 확인
p_value = anova_table["PR(>F)"][0]
print(f"\nANOVA p-value: {p_value}")

# 시각화 - 각 노선의 평균 혼잡도 시각화
mean_congestion_by_line = melted_data.groupby('호선')['혼잡도'].mean()

fig, ax = plt.subplots(figsize=(8, 6))
mean_congestion_by_line.plot(kind='bar', color='skyblue', ax=ax)
ax.set_xlabel('Line')
ax.set_ylabel('Average Congestion')
ax.set_title('Average Congestion by Subway Line')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
ANOVA Table:
                sum_sq       df           F  PR(>F)
C(호선)     5.993697e+05      7.0  268.240912     0.0
Residual  2.026578e+07  63488.0         NaN     NaN

ANOVA p-value: 0.0
```



F-Statistic

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

# 시간대 열만 선택하여 melt 함수로 세로 길게 변환
time_columns = sub_cong_data.columns[6:]  # 혼잡도 데이터가 포함된 시간대 열만 선택
melted_data = sub_cong_data.melt(id_vars=['호선'], value_vars=time_columns, var_name='시간대', value_name='혼잡도')

# ANOVA를 위한 모델 생성
model = smf.ols('혼잡도 ~ C(호선)', data=melted_data).fit()
aov_table = sm.stats.anova_lm(model)

# 각 호선의 혼잡도 데이터를 추출하여 f_oneway 함수에 입력
lines = [melted_data[melted_data['호선'] == line]['혼잡도'].dropna() for line in melted_data['호선'].unique()]
res = stats.f_oneway(*lines)

# F-Statistic과 p-value 출력 (단측 검정)
print(f'F-Statistic: {res.statistic / 2:.4f}')
print(f'p-value: {res.pvalue / 2:.4f}')
```

```
F-Statistic: 134.1205
p-value: 0.0000
```

각 지하철 노선의 평균 혼잡도가 유의미하게 다르다는 것을 알 수 있다

시간대와 노선별 혼잡도 상호작용 분석

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

# 시간대 열만 선택하여 melt 함수로 세로 길게 변환
time_columns = sub_cong_data.columns[6:]  # 혼잡도 데이터가 포함된 시간대 열만 선택
melted_data = sub_cong_data.melt(id_vars=['호선'], value_vars=time_columns, var_name='시간대', value_name='혼잡도')

# 시간대를 범주형 변수로 변환
melted_data['시간대'] = melted_data['시간대'].astype('category')

# ANOVA 모델 생성 (호선, 시간대, 그리고 상호작용 효과 포함)
model = smf.ols('혼잡도 ~ C(호선) + C(시간대) + C(호선):C(시간대)', data=melted_data).fit()
aov_table = sm.stats.anova_lm(model)
print("ANOVA Table with Line and Time Interaction:")
print(aov_table)

# F-Statistic과 p-value 출력 (단측 검정)
print(f'F-Statistic: {res.statistic / 2:.4f}')
print(f'p-value: {res.pvalue / 2:.4f}')
```

```
ANOVA Table with Line and Time Interaction:
                    df         sum_sq       mean_sq           F        PR(>F)
C(호선)              7.0  5.993697e+05  85624.249719  316.030764  0.000000e+00
C(시간대)           38.0  2.910490e+06  76591.845179  282.693039  0.000000e+00
C(호선):C(시간대)    266.0  2.364459e+05    888.894186    3.280822  4.923163e-65
Residual       63184.0  1.711885e+07    270.936439         NaN           NaN
```

시간대에서 F-값이 282.69로 매우 높다는 것은 시간대가 혼잡도에 큰 영향 미친다는 것

호선:시간대도 p-value가 매우 낮으므로 시간대에 따른 혼잡도 차이가 노선마다 다르게 나타남을 알 수 있음

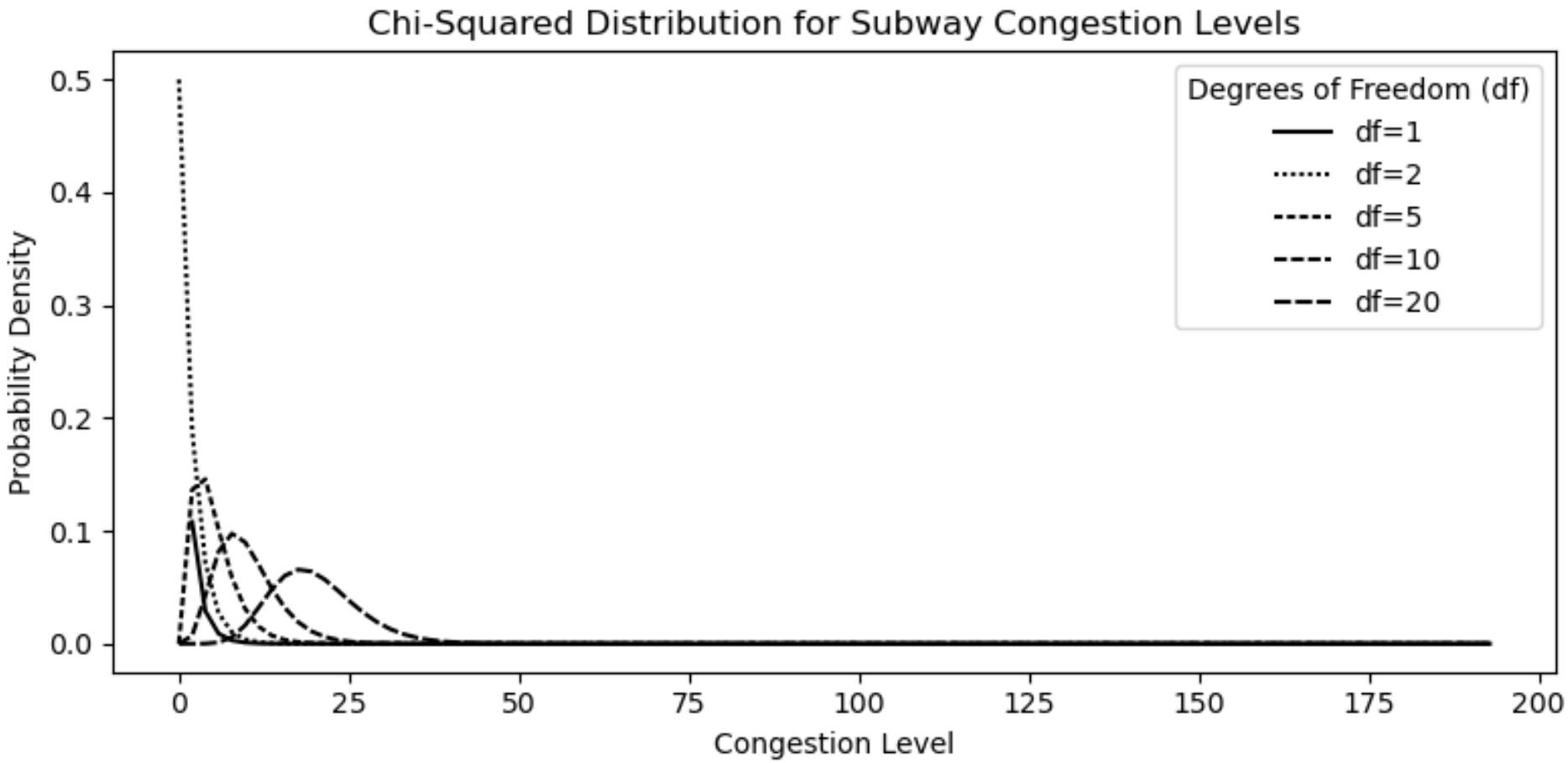
Chi-Square Test

- Chi-Square Test:

A Resampling Approach

기준 호선	비교 호선	Observed chi2	p-value
1	2	1602.847466	8.327670e-35
1	3	1049.674979	3.873939e-08
1	4	890.858049	2.089611e-02
1	5	1001.629930	7.780506e-03
1	6	798.399814	2.206399e-02
1	7	1059.823421	8.228817e-03
1	8	1392.670855	2.029302e-36
2	3	2252.402776	1.033137e-94
2	4	1688.654336	7.771020e-36
2	5	2169.164988	3.497775e-82
2	6	3278.706206	2.168286e-242
2	7	1947.387062	4.325628e-51
2	8	2098.848407	2.011755e-77
3	4	1212.509301	2.372726e-11
3	5	1285.368539	3.056500e-11
3	6	1414.835868	2.897010e-28
3	7	1952.031854	4.553811e-61
3	8	1270.659069	1.824439e-15
4	5	1166.768730	4.210861e-06
4	6	1362.762729	1.915472e-26
4	7	1229.353667	5.197829e-06
4	8	1587.781238	9.660768e-41
5	6	1368.939026	1.010155e-19
5	7	1739.028057	3.195409e-37
5	8	1647.158938	2.080750e-38
6	7	2012.950021	8.899122e-74
6	8	2017.543881	3.973373e-97
7	8	2256.511520	3.037097e-96

```python
# 카이제곱 검정 결과를 저장할 리스트
results = []

# 호선 간 모든 조합에 대해 카이제곱 검정 수행
for base_line, compare_line in combinations(melted_data['호선'].unique(), 2):
    # 기준 호선 데이터 선택
    base_data = melted_data[melted_data['호선'] == base_line]
    compare_data = melted_data[melted_data['호선'] == compare_line]

    # 혼잡도를 기준으로 관측된 수 계산
    observed_counts_base = base_data['혼잡도'].value_counts()
    observed_counts_compare = compare_data['혼잡도'].value_counts()

    # 두 호선의 혼잡도 레벨을 동일하게 맞추고, 없는 항목은 0으로 채움
    all_levels = observed_counts_base.index.union(observed_counts_compare.index)
    observed_counts_base = observed_counts_base.reindex(all_levels, fill_value=0)
    observed_counts_compare = observed_counts_compare.reindex(all_levels, fill_value=0)

    # 기대 빈도가 0인 항목 제외
    combined_counts = np.array([observed_counts_base.values, observed_counts_compare.values])
    if np.any(combined_counts == 0):
        combined_counts = combined_counts[:, combined_counts.sum(axis=0) > 0]

    # 카이제곱 검정 수행
    chisq, pvalue, _, _ = stats.chi2_contingency(combined_counts)

    # 결과 저장
    results.append({
        '기준 호선': base_line,
        '비교 호선': compare_line,
        'Observed chi2': chisq,
        'p-value': pvalue
    })

# 결과를 DataFrame으로 변환
results_df = pd.DataFrame(results)
```

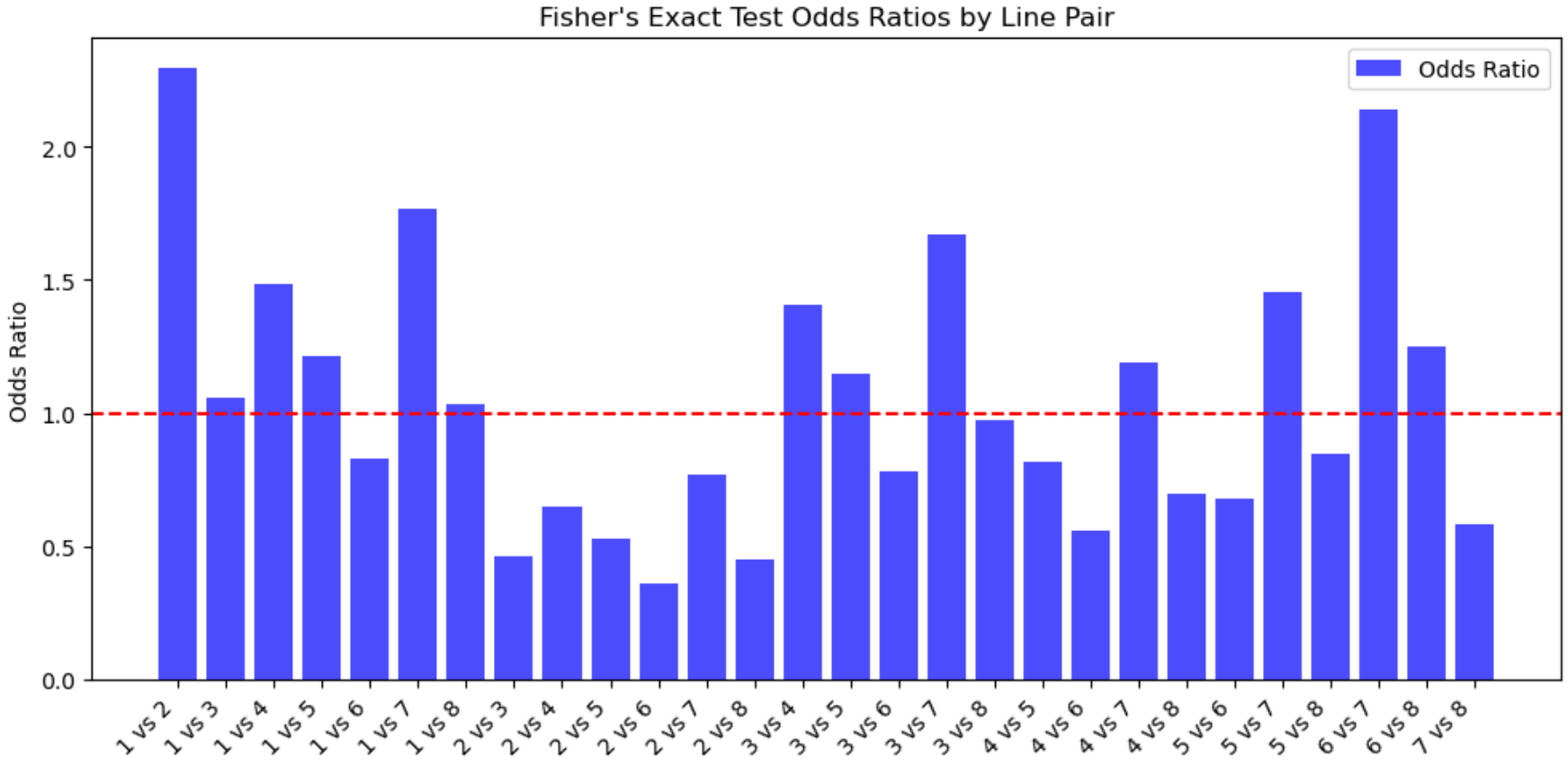


지하철 혼잡도 평균 : 26.57
자유도가 높아질수록 평균 부근에서 확률 밀도가
높아지며 정규분포와 유사한 종 모양이 됨

Fisher's Exact Test

```python
# 혼잡도를 기준으로 이진화 (예: 기준 초과/미달로 나누기)
threshold = 26  # 예시로 30을 기준으로 설정
melted_data['혼잡도_이진'] = (melted_data['혼잡도'] > threshold).astype(int)  # 1: 초과, 0: 미달

# 호선 간의 조합 생성
unique_lines = melted_data['호선'].unique()
line_combinations = list(combinations(unique_lines, 2))

# 결과를 저장할 리스트
results = []

# Fisher의 정확 검정을 각 호선 간에 수행
for line1, line2 in line_combinations:
    # 각 호선의 이진화된 혼잡도 데이터 추출
    line1_data = melted_data[melted_data['호선'] == line1]['혼잡도_이진'].value_counts()
    line2_data = melted_data[melted_data['호선'] == line2]['혼잡도_이진'].value_counts()

    # 빈도수 테이블 생성 (2x2 형태)
    observed_data = pd.DataFrame({
        'Line 1': line1_data.reindex([0, 1], fill_value=0),
        'Line 2': line2_data.reindex([0, 1], fill_value=0)
    })

    # Fisher의 정확 검정 수행 (2x2 확인)
    if observed_data.shape[0] == 2:  # 2x2인지 확인
        odds_ratio, p_value = stats.fisher_exact(observed_data)
        results.append({
            'Line Pair': f'{line1} vs {line2}',
            'Odds Ratio': odds_ratio,
            'p-value': p_value
        })
    else:
        print(f"Selected lines {line1} and {line2} do not form a 2x2 contingency table.")

# 결과를 DataFrame으로 변환
results_df = pd.DataFrame(results)
```

Line Pair	Odds Ratio	p-value
1 vs 2	2.294629	3.386589e-74
1 vs 3	1.056717	2.659773e-01
1 vs 4	1.485025	1.615501e-15
1 vs 5	1.214463	2.490530e-05
1 vs 6	0.826095	9.091743e-05
1 vs 7	1.768506	3.409446e-34
1 vs 8	1.030581	5.767226e-01
2 vs 3	0.460518	5.262070e-159
2 vs 4	0.647175	5.141116e-44
2 vs 5	0.529263	4.281984e-143
2 vs 6	0.360012	4.875426e-282
2 vs 7	0.770715	4.674960e-22
2 vs 8	0.449127	7.991986e-110
3 vs 4	1.405319	5.799683e-23
3 vs 5	1.149279	1.612197e-06
3 vs 6	0.781756	3.263585e-14
3 vs 7	1.673584	4.305762e-64
3 vs 8	0.975266	5.308491e-01
4 vs 5	0.817806	1.186706e-10
4 vs 6	0.556283	1.876631e-65
4 vs 7	1.190893	9.531219e-08
4 vs 8	0.693982	3.048507e-19
5 vs 6	0.680214	4.547979e-41
5 vs 7	1.456204	9.362373e-45
5 vs 8	0.848590	6.279114e-06
6 vs 7	2.140802	2.042705e-140
6 vs 8	1.247533	1.826364e-08
7 vs 8	0.582741	1.815487e-47



1호선vs2호선은 Odds Ratio가 2 이상인데,
2호선이 1호선보다 출퇴근 시간 혼잡도가
훨씬 높으므로 제대로 된 검정이 필요함