빅데이터분석프로그래밍 과제

- 20232503 홍연화
- 20232506 윤채영
- 20232511 김다현

---

seaborn titanic 데이터 분석하기

그래프 설정 및 데이터 불러오기

```
# 그림 해상도와 크기: 해상도를 높이면 크기에 비례에 커짐
import matplotlib.pyplot as plt

plt.rcParams.update({'figure.dpi' : '100'})             # 해상도, 기본값 72
plt.rcParams.update({'figure.figsize' : [14, 8]})      # 그림 크기, 기본값 [6, 4]


import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Seaborn에서 Titanic 데이터셋 로드
titanic = sns.load_dataset('titanic')

# 데이터 확인
titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adu
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

```
# 데이터 정보 확인
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

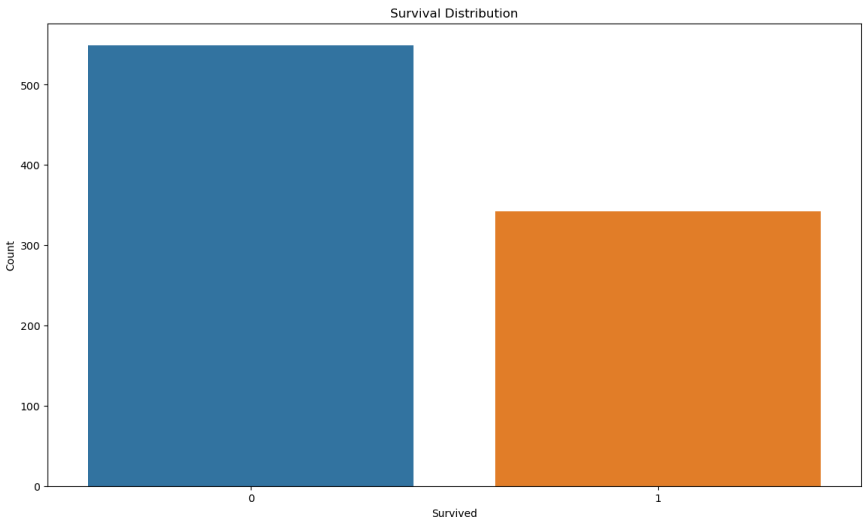
결측치 처리하기

결측치는 각 데이터 별로 상황에 맞게 제거 혹은 대체하였습니다.

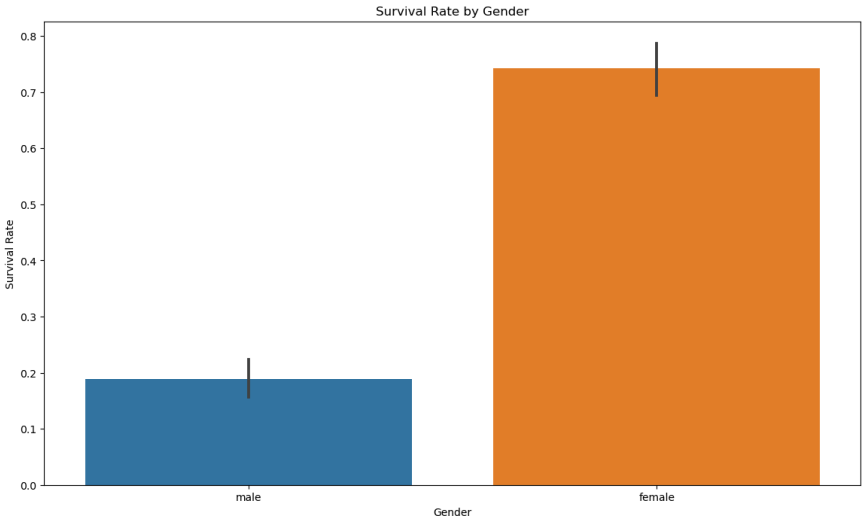
```
# 'age' 컬럼의 결측치를 중간값으로 대체
titanic['age'].fillna(titanic['age'].median(), inplace=True)

# 'embarked' 컬럼의 결측치를 최빈값으로 대체
titanic['embarked'].fillna(titanic['embarked'].mode()[0], inplace=True)



# 'deck' 컬럼의 결측치는 너무 많아서 해당 컬럼 제거
if 'deck' in titanic.columns:
    titanic.drop(columns=['deck'], inplace=True)

# 'embark_town' 컬럼의 결측치는 'embarked' 컬럼으로 대체 가능
if 'embark_town' in titanic.columns:
    titanic.drop(columns=['embark_town'], inplace=True)

# 'alive' 컬럼은 'survived' 컬럼과 중복 정보이므로 제거
if 'alive' in titanic.columns:
    titanic.drop(columns=['alive'], inplace=True)

# 'who' 컬럼은 'sex'와 유사한 정보 제공, 'adult_male'은 나이 정보로 유추 가능하므로 제거
if 'who' in titanic.columns:
    titanic.drop(columns=['who'], inplace=True)
if 'adult_male' in titanic.columns:
    titanic.drop(columns=['adult_male'], inplace=True)

# 'alone' 컬럼은 'sibsp'와 'parch'를 통해 유추 가능하므로 제거
if 'alone' in titanic.columns:
    titanic.drop(columns=['alone'], inplace=True)
```

데이터 시각화 및 분석

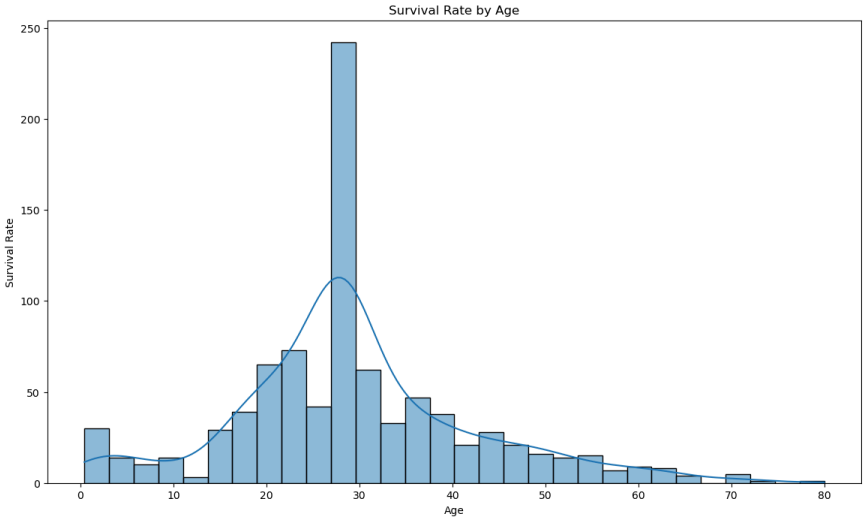
```
# 생존자와 사망자 분포
sns.countplot(data=titanic, x='survived')
plt.title('Survival Distribution')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()
```



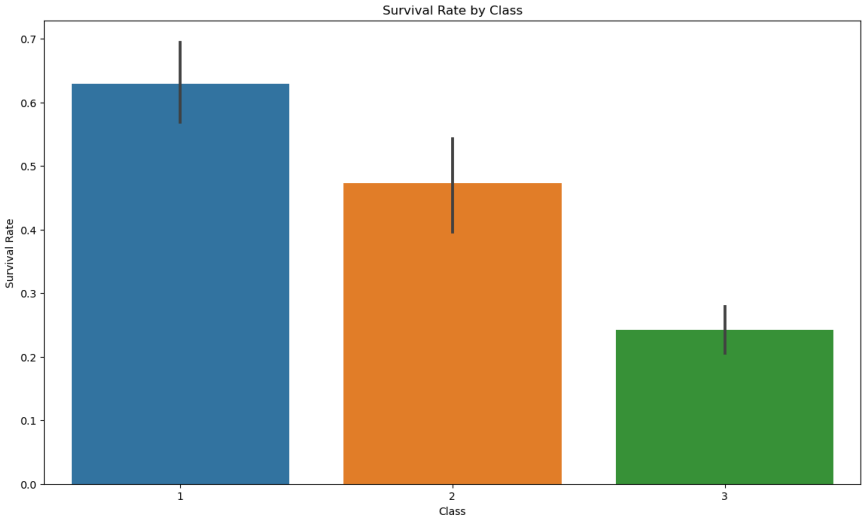
```
# 성별에 따른 생존율
sns.barplot(data=titanic, x='sex', y='survived')
plt.title('Survival Rate by Gender')
plt.xlabel('Gender')
plt.ylabel('Survival Rate')
plt.show()
```



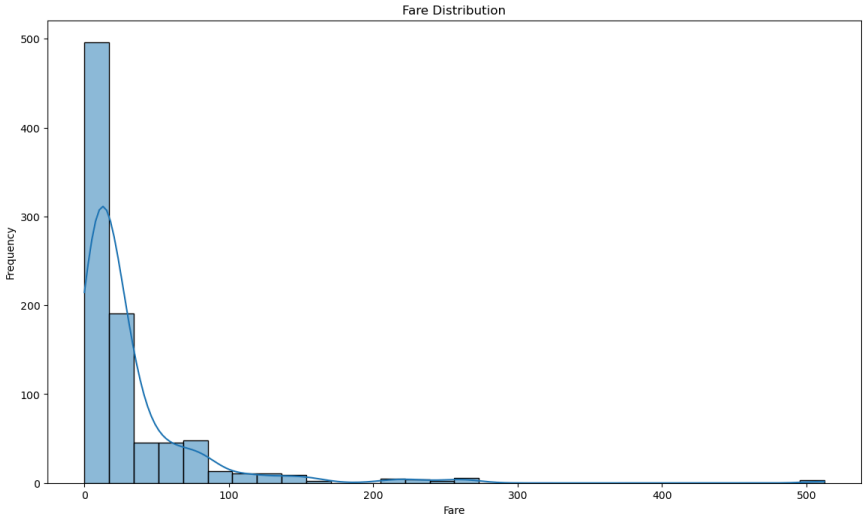

```
# 나이에 따른 생존율
sns.histplot(data=titanic, x='age', bins=30, kde=True)
plt.title('Survival Rate by Age')
plt.xlabel('Age')
plt.ylabel('Survival Rate')
plt.show()
```



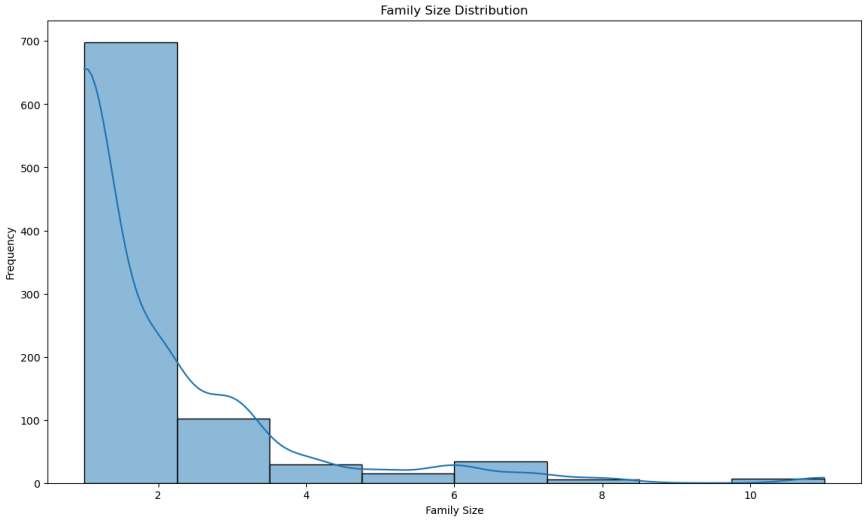

```
# 좌석 등급에 따른 생존율
sns.barplot(data=titanic, x='pclass', y='survived')
plt.title('Survival Rate by Class')
plt.xlabel('Class')
plt.ylabel('Survival Rate')
plt.show()
```



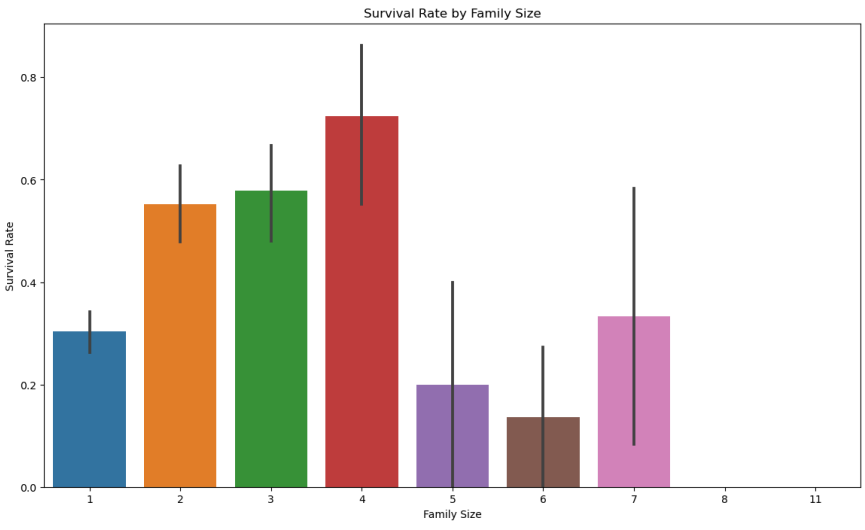

```
# 요금(fare) 분포
sns.histplot(data=titanic, x='fare', bins=30, kde=True)
plt.title('Fare Distribution')
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.show()
```



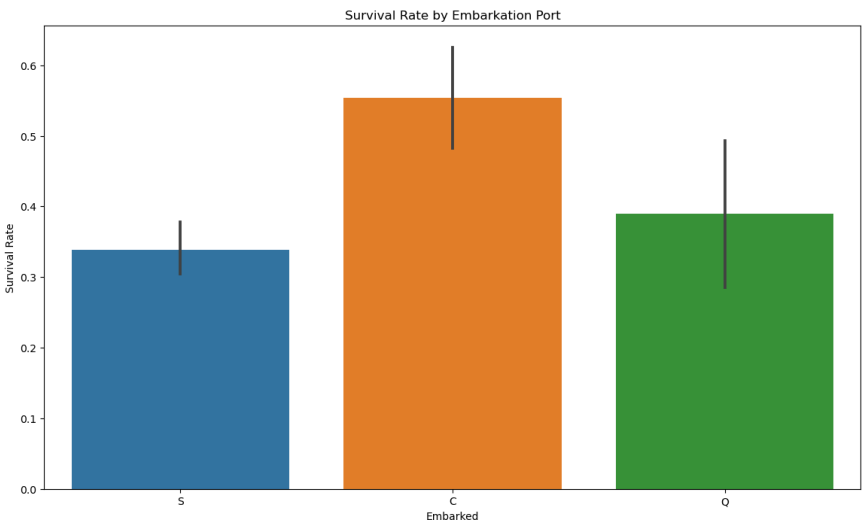

```
# 가족 크기(family_size) 분포 및 생존율
titanic['family_size'] = titanic['sibsp'] + titanic['parch'] + 1

sns.histplot(data=titanic, x='family_size', bins=8, kde=True)
plt.title('Family Size Distribution')
plt.xlabel('Family Size')
plt.ylabel('Frequency')
plt.show()
```



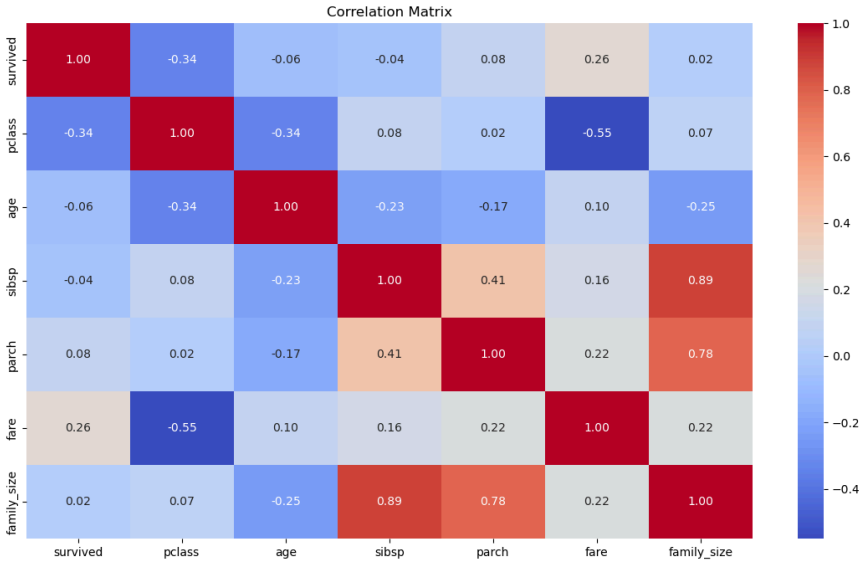

```
sns.barplot(data=titanic, x='family_size', y='survived')
plt.title('Survival Rate by Family Size')
plt.xlabel('Family Size')
plt.ylabel('Survival Rate')
plt.show()
```



```
# 승선 장소에 따른 생존율
sns.barplot(data=titanic, x='embarked', y='survived')
plt.title('Survival Rate by Embarkation Port')
plt.xlabel('Embarked')
plt.ylabel('Survival Rate')
plt.show()
```




```
# 상관 행렬 히트맵 (숫자형 컬럼만 사용)
numeric_titanic = titanic.select_dtypes(include=['float64', 'int64'])
corr_matrix = numeric_titanic.corr()
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



시간대별 지하철 및 버스 혼잡도 분석

---

1. 데이터 개요

이 분석에서는 버스와 지하철의 시간대별 혼잡도를 비교하고, 평균 등하교 시간대에 초점을 맞추어 최적의 경로를 파악하고자 했으며 이를 위해 다음과 같은 데이터를 사용

데이터 설명

1. **버스 승하차 인원 정보**
   - **설명**: 버스 정류소에서 시간대별 승차 및 하차 인원 수
   - **사용 목적**: 시간대별로 버스의 혼잡도를 파악하기 위해 사용
2. **버스 정류소 위치 정보**
   - **설명**: 각 버스 정류소의 위치 정보
   - **사용 목적**: 버스 정류소 간의 거리를 계산하고 경로를 최적화하기 위해 사용
3. **지하철 역별 시간대 혼잡도**
   - **설명**: 지하철 역에서 시간대별 혼잡도
   - **사용 목적**: 시간대별로 지하철의 혼잡도를 파악하기 위해 사용
4. **지하철 승하차 인원 정보**
   - **설명**: 지하철 역에서 시간대별 승차 및 하차 인원 수
   - **사용 목적**: 시간대별로 지하철의 혼잡도를 파악하기 위해 사용

데이터 분석 목표

- **혼잡도 비교**: 버스와 지하철의 시간대별 혼잡도를 비교하여 특정 시간대에 어느 교통수단이 더 혼잡한지 분석
- **최적 경로 파악**: 평균 등하교 시간대(등교 시간: 오전 10시, 오후 2시 / 하교 시간: 오후 1시, 오후 5시)에 최적의 경로를 파악하여 학생들의 이동을 최적화

환경설정 및 데이터 불러오기

```
import site
site.getsitepackages()
```

```
['c:\\Users\\YCY\\anaconda3', 'c:\\Users\\YCY\\anaconda3\\Lib\\site-packages']
```

```
import sys
import pandas as pd
import numpy as np
import seaborn as sns

print(sys.version)
print(pd.__version__)
print(np.__version__)
print(sns.__version__)
```

```
3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]
2.0.3
1.24.3
0.12.2
```

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
import re

# 한글 글꼴 설정
rc('font', family='Malgun Gothic')
plt.rcParams['axes.unicode_minus'] = False

# 파일 불러오기
bus_boarding_info = pd.read_csv('버스승하차인원정보.csv', low_memory=False)
bus_stop_location = pd.read_csv('버스정류소위치정보.csv',encoding='cp949')
subway_congestion = pd.read_csv("역별시간대혼잡도.csv",encoding='cp949')
subway_boarding_info = pd.read_csv("지하철승하차인원정보.csv",encoding='cp949')

# 동양미래대학교 위치 (위도와 경도)
school_location = {'latitude': 37.498, 'longitude': 127.027}
```

데이터 확인해보기

```
bus_boarding_info.info()
```

```
1   노선번호          41810 non-null  object
2   노선명           41810 non-null  object
3   표준버스정류장ID    41810 non-null  int64
4   버스정류장ARS번호   41810 non-null  object
5   역명            41810 non-null  object
6   00시승차총승객수    41810 non-null  int64
7   00시하차총승객수    41810 non-null  int64
8   1시승차총승객수     41810 non-null  int64
9   1시하차총승객수     41810 non-null  int64
10  2시승차총승객수     41810 non-null  int64
11  2시하차총승객수     41810 non-null  int64
12  3시승차총승객수     41810 non-null  int64
13  3시하차총승객수     41810 non-null  int64
14  4시승차총승객수     41810 non-null  int64
15  4시하차총승객수     41810 non-null  int64
16  5시승차총승객수     41810 non-null  int64
17  5시하차총승객수     41810 non-null  int64
18  6시승차총승객수     41810 non-null  int64
19  6시하차총승객수     41810 non-null  int64
20  7시승차총승객수     41810 non-null  int64
21  7시하차총승객수     41810 non-null  int64
22  8시승차총승객수     41810 non-null  int64
23  8시하차총승객수     41810 non-null  int64
24  9시승차총승객수     41810 non-null  int64
25  9시하차총승객수     41810 non-null  int64
26  10시승차총승객수    41810 non-null  int64
27  10시하차총승객수    41810 non-null  int64
28  11시승차총승객수    41810 non-null  int64
29  11시하차총승객수    41810 non-null  int64
30  12시승차총승객수    41810 non-null  int64
```

```
 37  15시하차총승객수    41810 non-null  int64
 38  16시승차총승객수    41810 non-null  int64
 39  16시하차총승객수    41810 non-null  int64
 40  17시승차총승객수    41810 non-null  int64
 41  17시하차총승객수    41810 non-null  int64
 42  18시승차총승객수    41810 non-null  int64
 43  18시하차총승객수    41810 non-null  int64
 44  19시승차총승객수    41810 non-null  int64
 45  19시하차총승객수    41810 non-null  int64
 46  20시승차총승객수    41810 non-null  int64
 47  20시하차총승객수    41810 non-null  int64
 48  21시승차총승객수    41810 non-null  int64
 49  21시하차총승객수    41810 non-null  int64
 50  22시승차총승객수    41810 non-null  int64
 51  22시하차총승객수    41810 non-null  int64
 52  23시승차총승객수    41810 non-null  int64
 53  23시하차총승객수    41810 non-null  int64
 54  교통수단타입코드     41810 non-null  int64
 55  교통수단타입명      41810 non-null  object
 56  등록일자         41810 non-null  int64
dtypes: int64(52), object(5)
memory usage: 18.2+ MB
```

```
bus_stop_location.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50465 entries, 0 to 50464
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   STDR_DE     50465 non-null  int64
 1   NODE_ID     50465 non-null  int64
 2   STTN_NO     50465 non-null  int64
 3   STTN_NM     50465 non-null  object
 4   CRDNT_X     50465 non-null  object
 5   CRDNT_Y     50465 non-null  float64
 6   STTN_TY     50465 non-null  float64
 7   Unnamed: 7  8 non-null      float64
dtypes: float64(3), int64(3), object(2)
memory usage: 3.1+ MB
```

```
subway_congestion.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1641 entries, 0 to 1640
Data columns (total 45 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   연번      1641 non-null   int64
 1   요일구분    1641 non-null   object
 2   호선      1641 non-null   int64
 3   역번호     1641 non-null   int64
 4   출발역     1641 non-null   object
 5   상하구분    1641 non-null   object
 6   5시30분   1641 non-null   float64
 7   6시00분   1641 non-null   float64
 8   6시30분   1641 non-null   float64
 9   7시00분   1641 non-null   float64
 10  7시30분   1641 non-null   float64
 11  8시00분   1641 non-null   float64
 12  8시30분   1641 non-null   float64
 13  9시00분   1641 non-null   float64
 14  9시30분   1641 non-null   float64
 15  10시00분  1641 non-null   float64
 16  10시30분  1641 non-null   float64
 17  11시00분  1641 non-null   float64
 18  11시30분  1641 non-null   float64
 19  12시00분  1641 non-null   float64
 20  12시30분  1641 non-null   float64
 21  13시00분  1641 non-null   float64
 22  13시30분  1641 non-null   float64
 23  14시00분  1641 non-null   float64
 24  14시30분  1641 non-null   float64
 25  15시00분  1641 non-null   float64
 26  15시30분  1641 non-null   float64
 27  16시00분  1641 non-null   float64
 28  16시30분  1641 non-null   float64
 29  17시00분  1641 non-null   float64
 30  17시30분  1641 non-null   float64
 31  18시00분  1641 non-null   float64
 32  18시30분  1641 non-null   float64
 33  19시00분  1641 non-null   float64
 34  19시30분  1641 non-null   float64
 35  20시00분  1641 non-null   float64
 36  20시30분  1641 non-null   float64
 37  21시00분  1641 non-null   float64
 38  21시30분  1641 non-null   float64
 39  22시00분  1641 non-null   float64
 40  22시30분  1641 non-null   float64
```

```
 41  23시00분   1641 non-null   float64
 42  23시30분   1641 non-null   float64
 43  00시00분   1641 non-null   float64
 44  00시30분   1641 non-null   float64
dtypes: float64(39), int64(3), object(3)
memory usage: 577.0+ KB
```

```
subway_boarding_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66207 entries, 0 to 66206
Data columns (total 52 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   사용월           66207 non-null  int64
 1   호선명           66207 non-null  object
 2   지하철역          66207 non-null  object
 3   04시-05시 승차인원  66207 non-null  int64
 4   04시-05시 하차인원  66207 non-null  int64
 5   05시-06시 승차인원  66207 non-null  int64
 6   05시-06시 하차인원  66207 non-null  int64
 7   06시-07시 승차인원  66207 non-null  int64
 8   06시-07시 하차인원  66207 non-null  int64
 9   07시-08시 승차인원  66207 non-null  int64
 10  07시-08시 하차인원  66207 non-null  int64
 11  08시-09시 승차인원  66207 non-null  int64
 12  08시-09시 하차인원  66207 non-null  int64
 13  09시-10시 승차인원  66207 non-null  int64
 14  09시-10시 하차인원  66207 non-null  int64
 15  10시-11시 승차인원  66207 non-null  int64
 16  10시-11시 하차인원  66207 non-null  int64
 17  11시-12시 승차인원  66207 non-null  int64
 18  11시-12시 하차인원  66207 non-null  int64
 19  12시-13시 승차인원  66207 non-null  int64
 20  12시-13시 하차인원  66207 non-null  int64
 21  13시-14시 승차인원  66207 non-null  int64
 22  13시-14시 하차인원  66207 non-null  int64
 23  14시-15시 승차인원  66207 non-null  int64
 24  14시-15시 하차인원  66207 non-null  int64
 25  15시-16시 승차인원  66207 non-null  int64
 26  15시-16시 하차인원  66207 non-null  int64
 27  16시-17시 승차인원  66207 non-null  int64
 28  16시-17시 하차인원  66207 non-null  int64
 29  17시-18시 승차인원  66207 non-null  int64
 30  17시-18시 하차인원  66207 non-null  int64
 31  18시-19시 승차인원  66207 non-null  int64
 32  18시-19시 하차인원  66207 non-null  int64
 33  19시-20시 승차인원  66207 non-null  int64
 34  19시-20시 하차인원  66207 non-null  int64
 35  20시-21시 승차인원  66207 non-null  int64
 36  20시-21시 하차인원  66207 non-null  int64
 37  21시-22시 승차인원  66207 non-null  int64
 38  21시-22시 하차인원  66207 non-null  int64
 39  22시-23시 승차인원  66207 non-null  int64
 40  22시-23시 하차인원  66207 non-null  int64
 41  23시-24시 승차인원  66207 non-null  int64
 42  23시-24시 하차인원  66207 non-null  int64
 43  00시-01시 승차인원  66207 non-null  int64
 44  00시-01시 하차인원  66207 non-null  int64
 45  01시-02시 승차인원  66207 non-null  int64
 46  01시-02시 하차인원  66207 non-null  int64
 47  02시-03시 승차인원  66207 non-null  int64
 48  02시-03시 하차인원  66207 non-null  int64
 49  03시-04시 승차인원  66207 non-null  int64
 50  03시-04시 하차인원  66207 non-null  int64
 51  작업일자          66207 non-null  int64
dtypes: int64(50), object(2)
```

```
subway_boarding_info.head()
```

	사용월	호선명	지하철역	04시-05시 승차인원	04시-05시 하차인원	05시-06시 승차인원	05시-06시 하차인원	06시-07시 승차인원	06시-07시 하차인원	07시-08시 승차인원	...	23시-24시 하차인원	00시-01시 승차인원
0	202404	1호선	동대문	708	7	11309	2206	9917	7632	16477	...	9476	693
1	202404	1호선	동묘앞	231	1	3091	802	4051	5422	9024	...	4900	151
2	202404	1호선	서울역	658	24	8560	9158	14541	62765	43682	...	14443	3203
3	202404	1호선	시청	94	2	2321	5550	4123	26788	7965	...	4110	661
4	202404	1호선	신설동	371	25	8785	2166	10319	9477	22808	...	9376	423

2.혼잡도 계산

혼잡도는 각 시간대별 승차 인원과 하차 인원의 차이를 통해 계산되었습니다. 승차 인원이 많고 하차 인원이 적을수록 혼잡도가 높아지며, 반대의 경우 혼잡도가 낮아집니다.

```
# 시간대 추출 (열 이름에서 숫자 부분만 추출)
subway_boarding_columns = subway_boarding_info.filter(like='승차').columns.tolist()
subway_alighting_columns = subway_boarding_info.filter(like='하차').columns.tolist()
time_columns = [int(re.findall(r'\d+', col)[0]) for col in subway_boarding_columns]

# 지하철 혼잡도 계산
subway_congestion_by_time = pd.DataFrame({
    '시간대': time_columns,
    '혼잡도': [
        subway_boarding_info[col].mean() - subway_boarding_info[subway_alighting_columns[i]].mean()
        if col in subway_boarding_info.columns and subway_alighting_columns[i] in subway_boarding_info.columns
        else 0
        for i, col in enumerate(subway_boarding_columns)
    ]
})

# 버스 혼잡도 계산
bus_boarding_columns = bus_boarding_info.filter(like='승차').columns.tolist()
bus_alighting_columns = bus_boarding_info.filter(like='하차').columns.tolist()
time_columns_bus = [int(re.findall(r'\d+', col)[0]) for col in bus_boarding_columns]

bus_congestion_by_time = pd.DataFrame({
    '시간대': time_columns_bus,
    '혼잡도': [
        bus_boarding_info[col].mean() - bus_boarding_info[bus_alighting_columns[i]].mean()
        if col in bus_boarding_info.columns and bus_alighting_columns[i] in bus_boarding_info.columns
        else 0
        for i, col in enumerate(bus_boarding_columns)
    ]
})
```

3. 최적 경로 분석 함수 정의

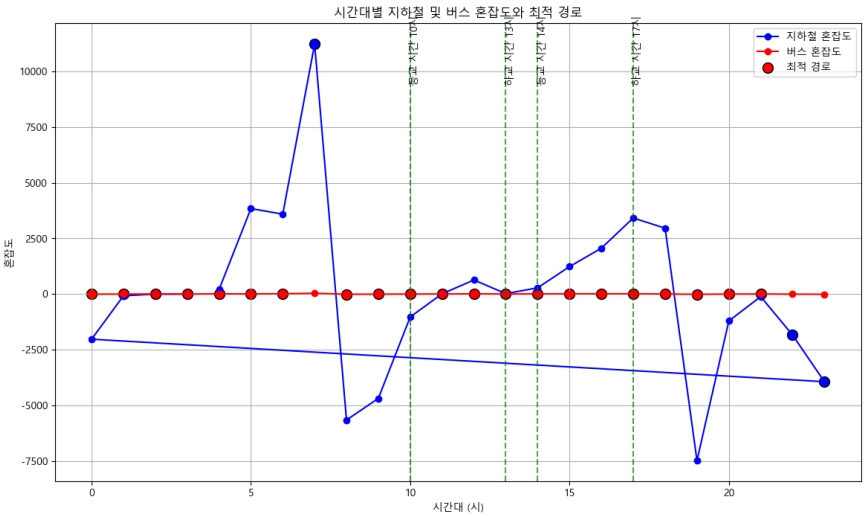
```
# 최적 경로 분석 함수
def find_optimal_route(congestion_data, travel_times):
    optimal_routes = []
    for index, row in congestion_data.iterrows():
        time = int(row['시간대'])  # 시간을 정수로 변환
        optimal_route = {
            '시간대': time,
            '최적 경로': None,
            '최소 혼잡도': float('inf'),
        }
        for route, time_info in travel_times.items():
            if time < len(time_info['혼잡도']) and abs(time_info['혼잡도'][time]) < abs(optimal_route['최소 혼잡도']):
                optimal_route['최적 경로'] = route
                optimal_route['최소 혼잡도'] = time_info['혼잡도'][time]
        optimal_routes.append(optimal_route)
    return pd.DataFrame(optimal_routes)


# 두 데이터를 결합하여 travel_times 데이터 생성
travel_times = {
    '지하철 경로': {'혼잡도': subway_congestion_by_time['혼잡도'].tolist()},
    '버스 경로': {'혼잡도': bus_congestion_by_time['혼잡도'].tolist()},
}

# 최적 경로 찾기
optimal_routes_df = find_optimal_route(subway_congestion_by_time, travel_times)

# 최적 경로 출력
print(optimal_routes_df)
```

```
    시간대   최적 경로     최소 혼잡도
0     4   버스 경로   8.475030
1     5   버스 경로   3.710476
2     6   버스 경로  14.824276
3     7  지하철 경로   9.974761
4     8   버스 경로  -9.571203
5     9   버스 경로  -5.870916
6    10   버스 경로  -0.351830
7    11   버스 경로   3.762593
8    12   버스 경로   6.641210
9    13   버스 경로   3.669409
10   14   버스 경로   6.767472
11   15   버스 경로   9.310213
12   16   버스 경로  11.167281
13   17   버스 경로  11.147118
14   18   버스 경로   3.001818
15   19   버스 경로 -13.556541
16   20   버스 경로  -1.127051
17   21   버스 경로   1.987730
18   22  지하철 경로  -0.849200
19   23  지하철 경로  -0.064283
20    0   버스 경로  -5.701626
21    1   버스 경로  -1.525425
22    2   버스 경로  -0.263191
23    3   버스 경로   0.071036
```

---

4. 평균 등하교 시간대의 최적 경로 시각화

```
# 시각화 함수 정의
def plot_congestion_and_routes(subway_data, bus_data, optimal_routes):
    plt.figure(figsize=(14, 8))

    # 지하철 혼잡도 시각화
    plt.plot(subway_data['시간대'], subway_data['혼잡도'], label='지하철 혼잡도', color='blue', marker='o')

    # 버스 혼잡도 시각화
    plt.plot(bus_data['시간대'], bus_data['혼잡도'], label='버스 혼잡도', color='red', marker='o')

    # 최적 경로 표시
    for index, row in optimal_routes.iterrows():
        time = row['시간대']
        if row['최적 경로'] == '지하철 경로':
            plt.scatter(time, subway_data[subway_data['시간대'] == time]['혼잡도'].values[0], color='blue', s=100, edgecolors='black', label='최적 경
        else:
            plt.scatter(time, bus_data[bus_data['시간대'] == time]['혼잡도'].values[0], color='red', s=100, edgecolors='black', label='최적 경로' if

    # 평균 등하교 시간대 강조
    avg_times = {'등교 시간': [10, 14], '하교 시간': [13, 17]}
    for key, times in avg_times.items():
        for time in times:
            plt.axvline(x=time, linestyle='--', color='green', alpha=0.7)
            plt.text(time, plt.ylim()[1]*0.9, f'{key} {time}시', rotation=90, verticalalignment='center')

    plt.title('시간대별 지하철 및 버스 혼잡도와 최적 경로')
    plt.xlabel('시간대 (시)')
    plt.ylabel('혼잡도')
    plt.legend()
    plt.grid(True)
    plt.show()

# 혼잡도 및 최적 경로 시각화 함수 호출
plot_congestion_and_routes(subway_congestion_by_time, bus_congestion_by_time, optimal_routes_df)
```



4-1. 평균 등하교 시간대의 최적 경로를 산점도로 시각화

```
# 시각화 함수 정의
def plot_congestion_scatter(subway_data, bus_data):
    plt.figure(figsize=(14, 8))

    # 지하철 혼잡도 산점도
    plt.scatter(subway_data['시간대'], subway_data['혼잡도'], label='지하철 혼잡도', color='blue', alpha=0.6)

    # 버스 혼잡도 산점도
    plt.scatter(bus_data['시간대'], bus_data['혼잡도'], label='버스 혼잡도', color='red', alpha=0.6)

    # 평균 등하교 시간대 강조
    avg_times = {'등교 시간': [10, 14], '하교 시간': [13, 17]}
    for key, times in avg_times.items():
        for time in times:
            plt.axvline(x=time, linestyle='--', color='green', alpha=0.7)
            plt.text(time, plt.ylim()[1]*0.9, f'{key} {time}시', rotation=90, verticalalignment='center', color='green')

    plt.title('시간대별 지하철 및 버스 혼잡도 산점도')
    plt.xlabel('시간대 (시)')
    plt.ylabel('혼잡도')
    plt.legend()
    plt.grid(True)
    plt.show()

# 혼잡도 산점도 시각화 함수 호출
plot_congestion_scatter(subway_congestion_by_time, bus_congestion_by_time)
```