서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

CH1. 주제 선정 이유

- 저렴한 가격, 건강 개선의 장점으로 공공 자전거 사용자 증가
- 자전거 전용도로 구체화 및 홍보 필요성 어필
- 특정 시간에만 사용자 증가하여 불편 호소
- 사용자가 없는 곳의 자전거, 사용자가 필요한 곳에 없는 자전거로 낮은 회전율

서울시 공공자전거의 대여 이력을 분석함으로써 사용자가 몰리는 시간대를 파악해 적절한 해결방안을 제안한다. 월별 대여 횟수 비교를 통해선 일괄적이고 포괄적인 자전거 수리 기간을 알 수 있고, 반납과 대여가 잦은 대여소에 대해서는 각 대여소에서 담당하는 자전거 수를 늘리거나 대여소를 늘린다.

CH2. 프로젝트 코드 설명

서울시 공공 자전거의 빠른 회전율을 위한 대여이력 분석

- 사용할 데이터는 http://data.seoul.go.kr/dataList/OA-15182/F/1/datasetView.do 에서 다운받을 수 있음.
- 2020년 9월부터 2021년 01월 데이터를 다운받음.

시작하기에 앞서 http://data.seoul.go.kr/dataList/OA-15182/F/1/datasetView.do에서 공공자전거 대여이력 정보 csv 파일을 2020.09 데이터부터 2021.01 데이터까지 다운받은 후 mid_data라는 폴더를 만들어 업로드 시킨다.

[1]:
```
# 필요하는 라이브러리 import

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import glob

# 경고 메시지 출력하지 않도록 다음과 같이 설정
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
# 주피터 노트북에서 그래프가 보이게 설정
%matplotlib notebook
```

[3]:
```
# 한글 font 설정
import platform
import matplotlib.font_manager as fm

if platform.system() == 'Windows': # Windows 환경 폰트 설정
    plt.rc('font', family='Malgun Gothic')
plt.rcParams['axes.unicode_minus'] = False #한글 폰트 사용시 마이너스 폰트 깨짐 해결
```

[4]:
```
# 글씨 선명하게 출력하는 설정

from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

사용할 라이브러리를 import하고,
한글깨짐 방지나 경고 메시지 출력 방지, 글씨가 선명히 출력되도록 기본적인 환경설정을 끝내둔다.

(import한 라이브러리 : pandas, matplotlib.pyplot, seaborn, datetime, glob, warnings, platform, matplotlib.font_manager)

1913397 오수민

서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

시작하기 전, 대용량 데이터(공공자전거 대여이력)를 read

1. glob 라이브러리와 glob 메소드를 이용하여 관련된 파일을 불러온다.
2. 반복문과 list를 이용하여 read 한다.

```
bic_data= glob.glob('./mid_data/공공자전거 대여이력 정보_*')
bic_data
```

```
['./mid_data₩₩공공자전거 대여이력 정보_2020.09.csv',
 './mid_data₩₩공공자전거 대여이력 정보_2020.10.csv',
 './mid_data₩₩공공자전거 대여이력 정보_2020.11.csv',
 './mid_data₩₩공공자전거 대여이력 정보_2020.12.csv',
 './mid_data₩₩공공자전거 대여이력 정보_2021.01.csv']
```

glob() 함수는 인자로 받은 패턴과 이름이 일치하는 모든 파일과 디렉터리의 리스트를 반환하기 때문에 csv 파일을 불러오기 위해 사용했다. 또한 필요한 파일이 많을 때 맨끝을 * 로 주면 앞부분이 같은 모든 파일을 불러오기 때문에 해당 방법을 사용했다.

```
bic_all=[]

for file in bic_data:
    try:
        raw = pd.read_csv(file,index_col=False,encoding = 'cp949',parse_dates=[0])
        bic_all.append(raw)
    except:
        raw=pd.read_csv(file,index_col=False,encoding='UTF-8',parse_dates=[0])
        bic_all.append(raw)
```

For문을 사용하여 여러 데이터를 불러오는데, 이때 리스트에 그 파일을 추가하는 식으로 for문을 작성했다. For문 안에는 try문을 사용하여 bic_data에 속하는 파일은 전부 추가한다.

```
bic = pd.concat(bic_all)
bic.reset_index(drop=True, inplace=True)
bic.head()
```

모아진 리스트 속의 파일은 pandas의 concat 함수로 리스트 안의 원소들을 데이터프레임으로 합친다. 합쳐진 데이터는 각자의 인덱스를 가지고 있으므로 이를 초기화한 뒤 새롭게 0부터 인덱스를 부여했다.

	자전거번호	대여일시	대여 대여소번호	대여 대여소명	대여거치대	반납일시	반납대여소번호	반납대여소명	반납거치대	이용시간	이용거리
0	SPB-32450	2020-09-01 00:01:11	1041	묘곡초등학교	0	2020-09-01 00:04:00	1083	광문고교사거리(아이파크)	0	3	0.0
1	SPB-32422	2020-09-01 00:01:05	240	문래역 4번출구 앞	0	2020-09-01 00:04:33	262	영문초등학교 사거리	0	3	0.0
2	SPB-33319	2020-09-01 00:00:53	122	신성기사식당 앞	0	2020-09-01 00:06:03	143	공덕역 2번출구	0	5	0.0
3	SPB-31368	2020-09-01 00:01:06	1024	강동구청 앞	0	2020-09-01 00:06:08	1029	롯데 시네마	0	5	0.0
4	SPB-33604	2020-09-01 00:01:51	1157	강서구청	0	2020-09-01 00:06:32	1131	꿈돌이공원 앞	0	5	0.0

```
bic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8109777 entries, 0 to 8109776
Data columns (total 11 columns):
 #   Column     Dtype
---  ------     -----
 0   자전거번호      object
 1   대여일시       object
 2   대여 대여소번호   int64
 3   대여 대여소명    object
 4   대여거치대      int64
 5   반납일시       object
 6   반납대여소번호    int64
 7   반납대여소명     object
 8   반납거치대      int64
 9   이용시간       int64
 10  이용거리       float64
dtypes: float64(1), int64(5), object(5)
memory usage: 680.6+ MB
```

bic.info()의 결과에서 '대여 일시', '반납 일시'가 object로 인식되고 있음을 파악했다.

```
bic['대여일시']=pd.to_datetime(bic['대여일시'])
bic['반납일시']=pd.to_datetime(bic['반납일시'])
bic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8109777 entries, 0 to 8109776
Data columns (total 11 columns):
 #   Column     Dtype
---  ------     -----
 0   자전거번호      object
 1   대여일시       datetime64[ns]
 2   대여 대여소번호   int64
 3   대여 대여소명    object
 4   대여거치대      int64
 5   반납일시       datetime64[ns]
 6   반납대여소번호    int64
 7   반납대여소명     object
 8   반납거치대      int64
 9   이용시간       int64
 10  이용거리       float64
dtypes: datetime64[ns](2), float64(1), int64(5), object(3)
memory usage: 680.6+ MB
```

'대여 일시', '반납 일시' 를 시간으로 인식시키기 위해 datetime으로 변환했다.
변환 결과를 확인했더니 두 변수 모두 datetime으로 변했음을 확인할 수 있었다.

1913397 오수민

서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

```
bic=bic.drop('대여거치대',axis=1)
bic=bic.drop('반납거치대',axis=1)
bic=bic.drop('대여 대여소명',axis=1)
bic=bic.drop('반납대여소명',axis=1)

bic.head()
```

	자전거번호	대여일시	대여 대여소번호	반납일시	반납대여소번호	이용시간	이용거리
0	SPB-32450	2020-09-01 00:01:11	1041	2020-09-01 00:04:00	1083	3	0.0
1	SPB-32422	2020-09-01 00:01:05	240	2020-09-01 00:04:33	262	3	0.0
2	SPB-33319	2020-09-01 00:00:53	122	2020-09-01 00:06:03	143	5	0.0
3	SPB-31368	2020-09-01 00:01:06	1024	2020-09-01 00:06:08	1029	5	0.0
4	SPB-33604	2020-09-01 00:01:51	1157	2020-09-01 00:06:32	1131	5	0.0

```
bic.insert(1,'대여(YM)',bic['대여일시'].dt.strftime('%Y%m'))
bic.insert(2,'대여(month)',bic['대여일시'].dt.month)
bic.insert(3,'대여(hour)',bic['대여일시'].dt.strftime('%H'))
bic.insert(4,'고객',1)
bic.head()
```

	자전거번호	대여(YM)	대여(month)	대여(hour)	고객	대여일시	대여 대여소번호	반납일시	반납대여소번호	이용시간	이용거리
0	SPB-32450	202009	9	00	1	2020-09-01 00:01:11	1041	2020-09-01 00:04:00	1083	3	0.0
1	SPB-32422	202009	9	00	1	2020-09-01 00:01:05	240	2020-09-01 00:04:33	262	3	0.0
2	SPB-33319	202009	9	00	1	2020-09-01 00:00:53	122	2020-09-01 00:06:03	143	5	0.0
3	SPB-31368	202009	9	00	1	2020-09-01 00:01:06	1024	2020-09-01 00:06:08	1029	5	0.0
4	SPB-33604	202009	9	00	1	2020-09-01 00:01:51	1157	2020-09-01 00:06:32	1131	5	0.0

Info에서 memory usage로 확인할 수 있지만 현재 bic의 크기가 680MB로 매우 크기 때문에 필요로 하지 않는 요소는 drop()으로 제거한다. '대여거치대', '반납거치대', '대여 대여소명', '반납대여소명'은 분석에 필요로 하지 않으므로 제거했다. Axis의 의미는 '기준'으로 0은 인덱스, 1은 열을 의미한다. bic는 열이 기준이므로 axis=1로 설정한다.

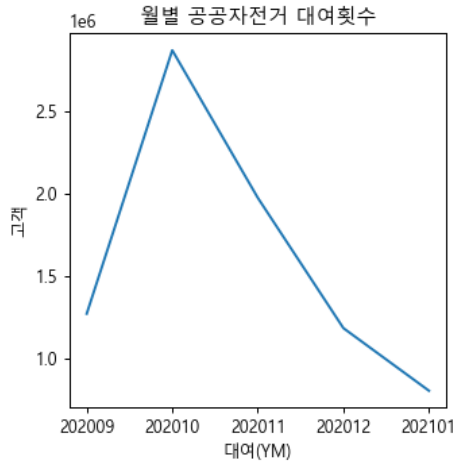
데이터분석을 위해 '대여일시'에서 필요한 년, 월, 시간을 추출한다. Datetime 라이브러리를 이용했으며 마지막에는 아래에서 고객의 명수 계산을 쉽게 하기 위해 각 행마다 1을 넣도록 했다.

1. 월별 공공자전거 대여 횟수

```
# 월별 공공 자전거 대여 횟수 비교한 데이터 프레임 (month_bic)

month_bic=pd.DataFrame(bic.groupby('대여(YM)').count()['고객'])
month_bic
```

대여(YM)	고객
202009	1273048
202010	2869182
202011	1975137
202012	1185907
202101	806503

```
# 시각화
month_bic.reset_index(level=['대여(YM)'],inplace=True)

plt.figure(figsize=(4,4))
plt.title("월별 공공자전거 대여횟수")
sns.lineplot(data=month_bic, x="대여(YM)", y="고객")
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x20605d3d760>
```

월별 공공 자전거 대여 횟수 비교한 데이터 프레임은 month_bic이다. 월별 비교를 위해 대여(month)를 사용할까 고민했지만 년도가 달라졌기 때문에 대여(YM)를 groupby()의 변수로 사용했고 대여(YM)의 값에 따라 그 값에 해당하는 '고객'의 칸의 개수를 모두 세도록 month_bic을 만들었다.

이렇게 분석한 내용에 대해선 대여 횟수의 변화를 한눈에 알아보기 위해 위 데이터를 lineplot으로 나타냈다. 먼저 month_bic의 그룹이었던 대여(YM)을 하나의 열로 month_bic에 다시 넣어준다. 그리고 각각 대여(YM)의 값들은 x로, x에 대응하는 고객(=월간 대여횟수)은 y로 오도록 선 그래프를 그렸다.

이 그래프를 통해 우린 2020년 10월에 공공자전거 사용이 가장 늘었고 2021년 01월에 사용이 가장 줄었다는 점을 알 수 있다.

1913397 오수민

서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

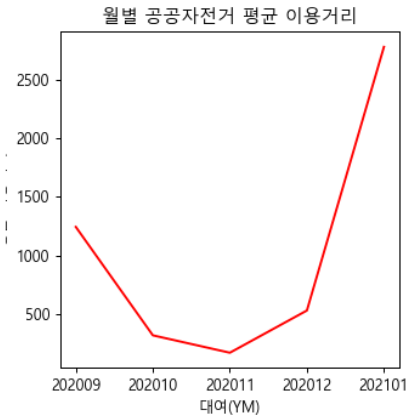
2. 월별 자전거를 이용해 이동한 평균거리

```
# 월별 자전거 이용해 이동한 평균거리 데이터 프레임 (month_dis)

month_dis=pd.DataFrame(bic.groupby('대여(YM)').mean()['이용거리'])
month_dis=month_dis.rename(columns={'이용거리':'평균 이용거리'})
month_dis
```

평균 이용거리

대여(YM)	
202009	1244.242216
202010	317.182635
202011	168.376186
202012	529.407123
202101	2783.126334

```
# 시각화
month_dis.reset_index(level=['대여(YM)'],inplace=True)

plt.figure(figsize=(4,4))
plt.title("월별 공공자전거 평균 이용거리")
sns.lineplot(data=month_dis, x="대여(YM)", y="평균 이용거리",color='red')
```



월별 자전거를 이용해 이동한 평균거리를 비교한 데이터 프레임은 month_dis다. 1번 문제와 같은 이유로 대여(YM)를 groupby()의 변수로 사용했고 대여(YM)의 값에 따라 그 값에 해당하는 '이용거리'의 평균을 구하도록 했다. 그리고 이용거리라는 colum의 이름은 '평균 이용거리 ' 로 바꾸어 최종적인 month_dis를 만들었다.

이렇게 분석한 내용에 대해선 이동거리 차이를 한눈에 알아보기 위해 위 데이터를 lineplot으로 나타냈다. 먼저 month_dis의 그룹이었던 대여(YM)을 하나의 열로 month_dis에 다시 넣어주었고 그 대여(YM)의 값들은 x로, 평균 이동거리는 y로 오도록 그래프를 그렸다.

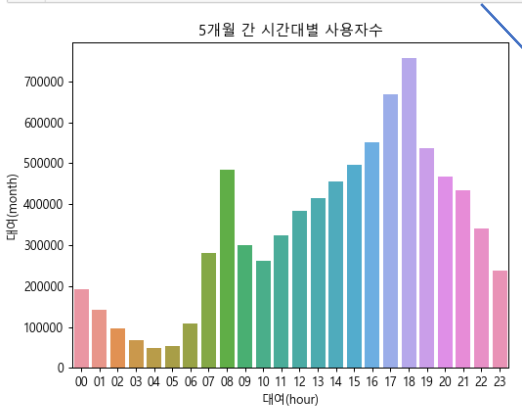
이 그래프를 통해 우린 2020년 11월에 공공자전거 사용 이동거리가 가장 줄었고 2021년 01월에 가장 늘었음을 확인했다.

3. 시간대별 공공자전거 사용자수 내림차순으로 비교

```
# 시간대별 사용자수 내림차순으로 비교 (time_user)

time_user=bic.groupby('대여(hour)').count()[['대여(month)']].sort_values(by='대여(month)',ascending=False)
time_user.head()
```

대여(hour)	대여(month)
18	757435
17	668880
16	551820
19	537509
15	496109

```
# 시각화
time_user.reset_index(level=['대여(hour)'],inplace=True)

plt.title("5개월 간 시간대별 사용자수")
sns.barplot(data=time_user,y='대여(month)',x='대여(hour)')
```



<matplotlib.axes._subplots.AxesSubplot at 0x206069a0070>

시간대별 공공자전거 사용자수를 내림차순으로 비교한 데이터 프레임은 time_user이다. 시간이 분석의 기준이 되기 때문에 '대여(hour)'를 groupby()의 변수로 사용했고 대여(hour)의 값에 따라 그 값에 해당하는 '대여(month)'의 개수를 구하도록 했다. 내림차순으로 자전거 사용자수를 정렬하기 위해선 by에 대여(month)를 설정, 오름차순(ascending)은 false로 지정했다. 이렇게 나온 결과값이 바로 time_user다.

분석한 내용에 대해선 시간대별 사용자 차이를 한눈에 알아보기 위해 위 데이터를 barplot으로 나타냈다. 먼저 time_user의 그룹이었던 대여(hour)를 하나의 열로 time_user에 다시 넣어주었다. 그리고 대여(hour)의 값들은 x로, 대여(month)는 y로 오도록 그래프를 그렸다.

이 그래프를 통해 우린 사람들은 18시, 퇴근시간에 공공자전거를 가장 많이 이용하고 새벽 4시에 가장 적게 이용한다는 점을 알 수 있다.

1913397 오수민

서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

- 4. '대여', '반납'이 잦은 10개의 대여소 번호 찾기

```
# 대여가 잦은 대여소 10개 추출 (go_bic)

go_bic=bic.groupby('대여 대여소번호').sum()[['고객']].sort_values(by='고객',
                                                    ascending=False).head(10)
go_bic=go_bic.rename(columns={'고객':'대여고객'})
go_bic
```

대여 대여소번호	대여고객
502	33027
207	30459
2102	27634
1153	24884
152	23180
1210	21320
2177	20716
2715	20344
583	19408
1160	18830

대여가 잦은 대여소 10개를 찾는다. 이름은 go_bic으로 설정하기로 하고 '대여 대여소번호'를 기준으로 그룹별로 고객의 수를 구했다. 그리고 고객의 수에 따라 내림차순으로 정렬한 다음, 그 위에 있는 10개의 항목을 뽑아낸다. '고객'이라고 설정된 go_bic의 column은 다음 설정할 반납 고객과 구별하기 위해 '대여고객'으로, 열 이름을 바꾸었다.

```
# 반납이 잦은 대여소 10개 추출 (back_bic)
back_bic=bic.groupby('반납대여소번호').sum()[['고객']].sort_values(by='고객',
                                                    ascending=False).head(10)
back_bic=back_bic.rename(columns={'고객':'반납고객'})
back_bic
```

반납대여소번호	반납고객
502	37206
207	31464
2102	28206
152	25910
1153	25074
2177	21638
1210	21418
583	20257
2715	20198
565	19417

반납이 잦은 대여소 10개를 찾는다. 방법은 go_bic과 동일하다. 이름은 back_bic으로 설정하기로 하고 '반납대여소번호'를 기준으로 그룹별로 고객의 수를 구했다. 그리고 고객의 수에 따라 내림차순으로 정렬한 다음, 그 위에 있는 10개의 항목을 뽑아낸다. '고객'이라고 설정된 back_bic의 column은 '반납고객'으로, 열 이름을 바꾸었다.

```
# 대여와 반납을 합쳐서 비교하는 데이터 프레임 (all_bic)

go_bic.reset_index(level=['대여 대여소번호'],inplace=True)
go_bic=go_bic.rename(columns={'대여 대여소번호':'대여소'})

back_bic.reset_index(level=['반납대여소번호'],inplace=True)
back_bic=back_bic.rename(columns={'반납대여소번호':'반납소'})

all_bic=pd.concat([go_bic, back_bic], axis=1, join='inner')
all_bic
```

	대여소	대여고객	반납소	반납고객
0	502	33027	502	37206
1	207	30459	207	31464
2	2102	27634	2102	28206
3	1153	24884	152	25910
4	152	23180	1153	25074
5	1210	21320	2177	21638
6	2177	20716	1210	21418
7	2715	20344	583	20257
8	583	19408	2715	20198
9	1160	18830	565	19417

대여와 반납고객을 합쳐서 비교하는 데이터프레임을 생성한다. 이름은 all_bic으로 두고, 먼저 go_bic의 대여 대여소번호 인덱스를 열로 넣는다. 그리고 열 이름은 '대여소'로 변경한다.
back_bic도 같은 방법으로 처리한다. Back_bic의 반납대여소번호 인덱스를 열로 넣는다. 그리고 열 이름은 '반납소'로 변경한다.

이 all_bic은 두 데이터 프레임을 합쳐서 만드는데 axis=1로 두어 df가 옆으로 이어지도록, join='inner'로 두어 대여소와 반납소 모두 있는 곳(intersection)만 추출하도록 한다.

이렇게 구한 표로 대여소, 반납소에 방문하는 고객 명수가 많은 순위를 한눈에 파악할 수 있으므로 시각화는 생략했다.

1913397 오수민

서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

- 5. 사용자가 많은(대여기준) 10개의 대여소 추출, 그 대여소의 고객들의 자전거 이용 평균시간

```
# 대여소별 자전거 이용 평균시간 비교한 데이터 프레임 (top10_avg)

top10=bic.groupby('대여 대여소번호').sum()[['고객','이용시간']].sort_values(by='고객',
                                                    ascending=False).head(10)
top10
```

대여 대여소번호	고객	이용시간
502	33027	1453605
207	30459	1501531
2102	27634	941520
1153	24884	466203
152	23180	1096324
1210	21320	601533
2177	20716	621333
2715	20344	344042
583	19408	867829
1160	18830	406275

```
top10['고객']= top10['이용시간']/top10['고객']
top10
```

대여 대여소번호	고객	이용시간
502	44.012626	1453605
207	49.296792	1501531
2102	34.071072	941520
1153	18.735051	466203
152	47.296117	1096324
1210	28.214493	601533
2177	29.992904	621333
2715	16.911227	344042
583	44.715014	867829
1160	21.575943	406275

사용자가 많은 10개의 대여소를 추출하여 그 고객들의 자전거 이용 평균시간을 구한다.

대여 대여소번호가 분석의 기준이 되기 때문에 '대여 대여소번호'를 groupby()의 변수로 사용했고, 고객들의 이용시간의 평균을 구하기 위해 고객과 이용시간의 총합을 먼저 구해주었다. 고객이 많이 찾는 대여소를 찾기 위해, 정렬은 '고객'을 기준으로 내림차순으로 정렬했으며 상위 10개를 추출해 이름을 top10이라고 먼저 정해 두었다.

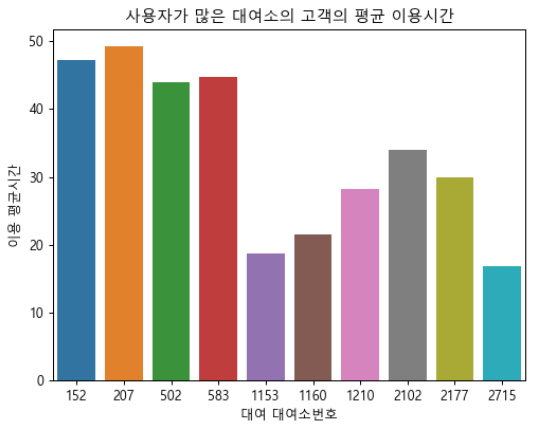
대여소별로 고객들이 이용한 평균 시간을 구할 차례다. top10의 '고객'에 해당하는 각각의 값을 그 대여소 따라 총 이용시간에서 총 고객수로 나누어서 대체했다.

```
top10_avg=top10.drop('이용시간',axis=1)
top10_avg=top10_avg.rename(columns={'고객':'이용 평균시간'})
top10_avg
```

대여 대여소번호	이용 평균시간
502	44.012626
207	49.296792
2102	34.071072
1153	18.735051
152	47.296117
1210	28.214493
2177	29.992904
2715	16.911227
583	44.715014
1160	21.575943

하지만 top10은 아직 총 이용시간의 열을 포함하고 있기 때문에 drop() 함수로 그 열을 제거하고 이제부턴 top10_avg로 이름을 설정해준다.
'고객'으로 설정되어있는 열은 이름을 '이용 평균시간'으로 바꾸어 더 내용을 명확히 한다.

```
# 시각화
top10_avg.reset_index(level=['대여 대여소번호'],inplace=True)

plt.title("사용자가 많은 대여소의 고객의 평균 이용시간")
sns.barplot(data=top10_avg, x="대여 대여소번호", y="이용 평균시간")
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x26ec4500160>
```

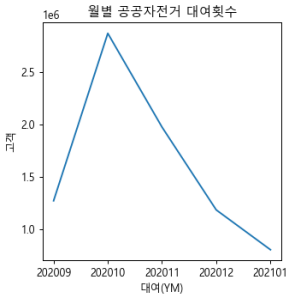
top10_avg의 시각화는 barplot을 사용하여 구현했다. 먼저 top10_avg의 그룹인 대여 대여소번호를 열로 추가해서 새로운 데이터 프레임으로 변경해준다. 그리고 각각 대여 대여소번호들의 값은 x로, x에 대응하는 이용 평균시간은 y로 오도록 막대 그래프를 그렸다.

이 그래프를 통해 152번의 대여소에 사용자가 가장 많고 이용시간도 50분 즈음으로 가장 긴 것을 확인할 수 있다. 그리고 눈에 띄는 1153번 대여소를 보면 5번째로 사용자가 많지만 사용자들의 이용시간은 짧다는 것을 알 수 있다.

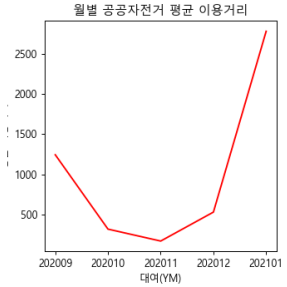
1913397 오수민

서울시 공공자전거의 빠른 회전율을 위한 대여 이력 분석

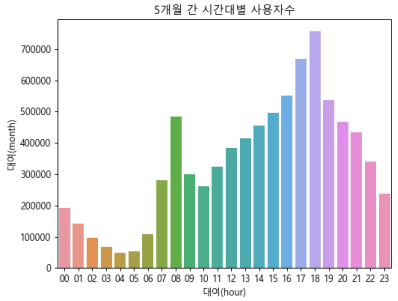
CH3. 개선 방법



1번



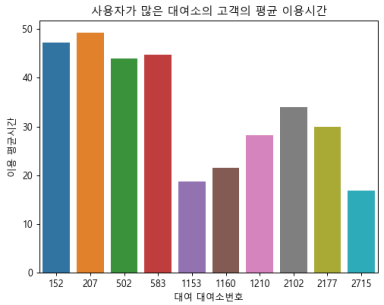
2번



3번

	대여소	대여고객	반납소	반납고객
0	502	33027	502	37206
1	207	30459	207	31464
2	2102	27634	2102	28206
3	1153	24884	152	25910
4	152	23180	1153	25074
5	1210	21320	2177	21638
6	2177	20716	1210	21418
7	2715	20344	583	20257
8	583	19408	2715	20198
9	1160	18830	565	19417

4. 대여, 반납이 잦은 10개의 대여소



5. 사용자가 많은 10개의 대여소 고객의 평균 자전거 이용시간

- **고장난 공공자전거가 수리되지 않은 문제.**

해결방안) 1번 데이터를 활용하여 년도별로 공공자전거 대여횟수를 분석하여 사람들이 많이 이용하지 않는 달에 총괄적으로 수리 및 점검을 실시한다. 또 수리 및 점검에 2번 데이터를 참고하여 고객들의 이용거리, 즉 자전거의 사용도나 사용감을 확인한다.

- **특정 시간이 되면 사용자가 몰려 자전거 이용이 불가한 문제.**

해결방안) 3번 데이터를 활용하여 자전거 이용이 몰리는 시간대를 고객에게 공지하고, 4번 데이터를 찾은 방법을 응용하거나 4번 데이터를 이용하여 대여가 많지 않은 대여소의 자전거를 사람이 몰리는 대여소에 배치한다. 또한 자전거 이용률이 높은 시간일수록 교통이 복잡할 확률도 증가하기 때문에 4번 데이터에서 알려진 대여소와 반납소의 거리를 파악해 그 사이에 자전거 전용도로를 넓히거나 개수를 늘려 교통 안전도 개선한다.

- **자전거 이용시간이 긴 대여소의 낮은 회전율 문제.**

해결방안) 4번과 5번 데이터를 활용하여 이용시간이 길고 방문률이 높은 대여소와 반납소에는 자전거 거치대와 배치하는 자전거의 개수를 늘린다. 또한 자전거의 이용시간이 길수록 그에 대한 자전거 고장 확률도 높기 때문에 자주 사용되는 자전거 번호를 분석하여 주기적인 관리를 유지한다.

CH4. 어려웠던 점

1. 데이터의 크기

데이터의 크기가 너무 커서 파일을 분석하는 데에 시간이 너무 오래 걸렸다. 분석하는 데이터의 크기를 줄이는 것을 고민해보지 않은 부분은 아니지만, 분석 데이터를 줄이면 정확성이 떨어질 것이라고 생각하여 데이터를 줄이지 않았다.

2. 데이터 read 과정의 오류

다운받은 공공자전거 대여이력 정보 데이터가 ANSI로 인코딩된 상태로 업로드하고, 파일을 클릭하여 데이터를 확인해보니 오른쪽 사진처럼 csv 파일이 파이썬에서 로딩되지 않는 오류가 발생했다. 그래서 모든 csv 파일을 연결프로그램에서 메모장>다른이름으로 저장>인코딩:UTF-8(BOM) 으로 저장하고 아나콘다에 업로드 시켜서 실행하니 오류가 발생하지 않았다.

```
1 Error! C:₩Users₩osoomin₩Bigdata_analysis_2021₩mid_data₩공공자전거 대여이력 정보
  _2021.01(1).csv is not UTF-8 encoded
2 Saving disabled.
3 See Console for more details.
```

1913397 오수민