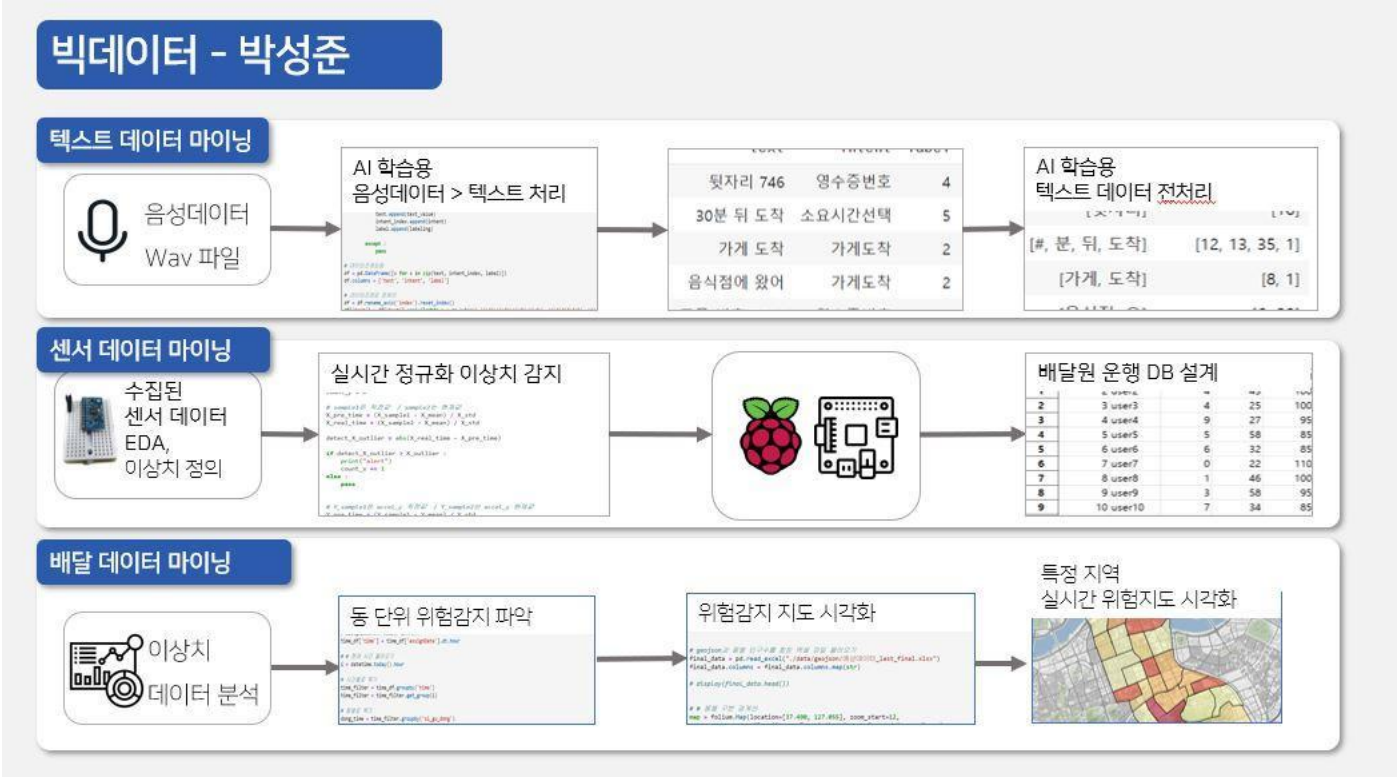
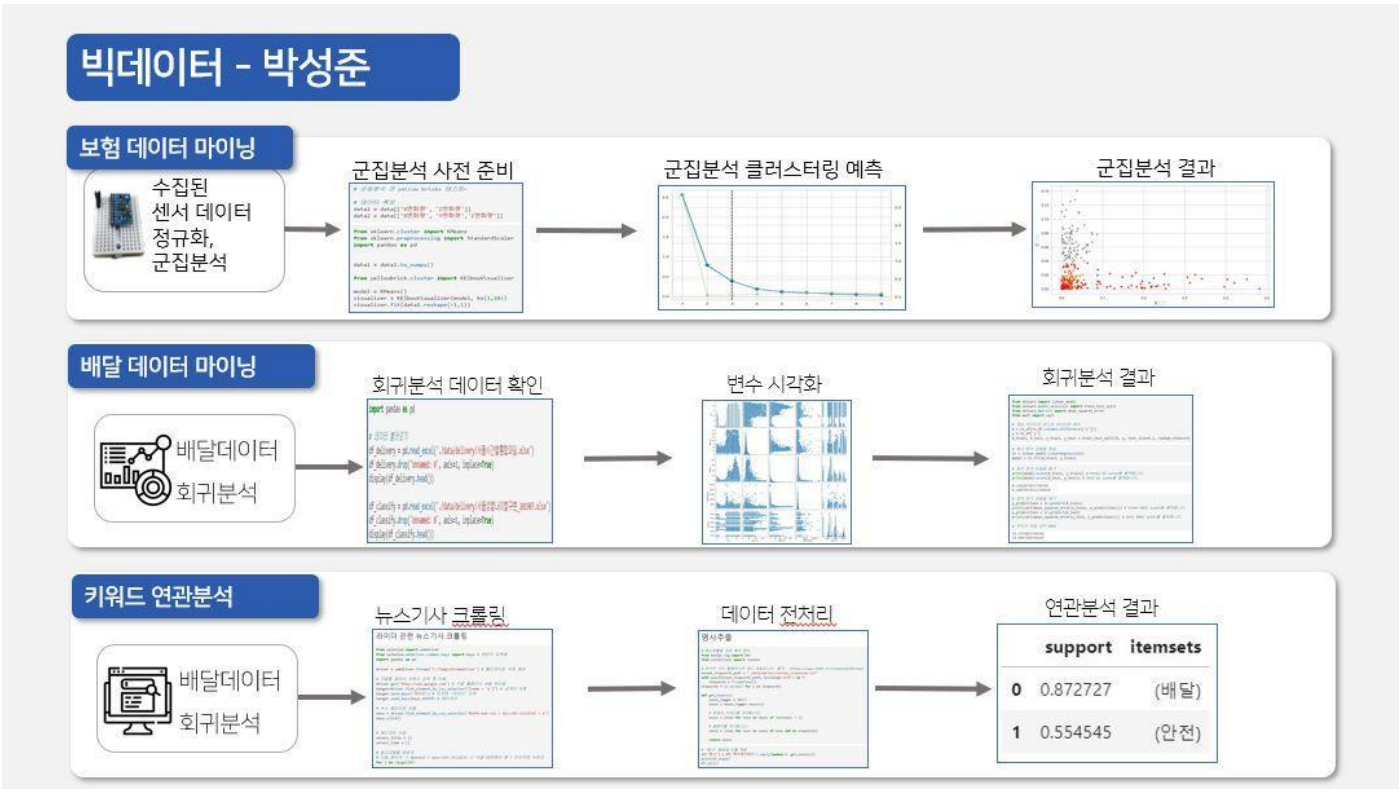
빅데이터 보고서

작성자 : 박성준

빅데이터 주요 업무 및 협력사안





빅데이터 주요 업무 세부사항

1. 배달데이터 시각화 및 인사이트 도출

- KT 빅데이터 플랫폼과 한국데이터거래소 등의 배달 관련 데이터를 수집하고 이를 확인하고자 하였습니다. 배달데이터는 전국 단위로 있었기에 서울시 단위로 그룹핑하였습니다. 이러한 데이터를 시각화한 결과 특정 구에서는 데이터가 존재하지 않으며, 비교적 강남구에 배달데이터가 많다는 것을 알 수 있었습니다. 때문에 예상주문건수, 평균배달시간, 평균주문금액 등을 강남구에 초점을 맞추어 찾아보는 작업을 수행하였습니다.

배달관련 데이터 확인

```
# 배달 관련 데이터 확인   //    # ./fusion/data/delivery

delivery_shop = pd.read_csv('./data/delivery/배달 상점 데이터.csv', header=None)
delivery_shop.columns = ['상점ID', '업종명', '상점코드', '시도명', '구군명', '읍면동명', '법정동리명', '행정동코드', '행정동 읍면동명',

# delivery_loc = pd.read_csv('./data/delivery/배달자 위치정보 데이터.csv')

delivery_time_mean = pd.read_csv('./data/delivery/시간-지역별 배달 소요시간평균.csv', header=None)
delivery_time_mean.columns = ['날짜', '시간', '도', '시구', '평균시간']

delivery_time_order_count = pd.read_csv('./data/delivery/시간-지역별 배달 주문건수.csv', header=None)
delivery_time_order_count.columns = ['날짜', '시간', '도', '시구', '주문건수']

delivery_time_pay_mean = pd.read_csv('./data/delivery/시간-지역별 배달 평균주문금액.csv', header=None)
delivery_time_pay_mean.columns = ['날짜', '시간', '도', '시구', '평균주문금액']

delivery_job_order_count = pd.read_csv('./data/delivery/업종-지역별 배달 주문건수.csv', header=None)
delivery_job_order_count.columns = ['날짜', '업종', '도', '시', '주문건수']

delivery_job_time_mean = pd.read_csv('./data/delivery/업종-지역별 평균배달소요시간.csv', header=None)
delivery_job_time_mean.columns = ['날짜', '업종', '도', '시', '평균시간']

delivery_job_pay_mean = pd.read_csv('./data/delivery/업종-지역별 평균주문금액.csv', header=None)
delivery_job_pay_mean.columns = ['날짜', '업종', '도', '시', '평균주문금액']

delivery_hum = pd.read_csv('./data/delivery/주문지역 인구 특성.csv', header=None)
delivery_hum.columns = ['기준연월', '시군구코드', '시도명', '시군구명', '5세단위구분', '총인구수', '남성인구수', '여성인구수']
```

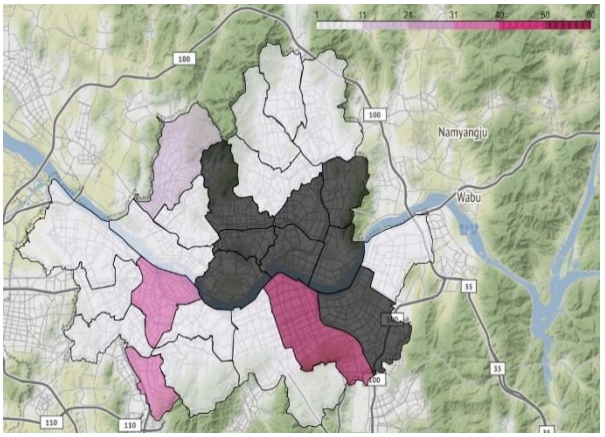
<배달데이터 확인>

배달 상점 데이터

	상점ID	업종명	상점코드	시도명	구군명	읍면동명	법정동리명	행정동코드	행정동 읍면동명	도로명주소코드	도로명
0	S0000001	치킨	4817012500	경상남도	진주시	신안동	\N	4817071500	신안동	481704797560	평거로116번길
1	S0000002	치킨	1130510100	서울특별시	강북구	미아동	\N	1130553400	삼양동	113053005042	솔샘로
2	S0000003	카페/디저트	2644010400	부산광역시	강서구	명지동	\N	2644054500	명지2동	264403136044	명지오션시티 3로
3	S0000004	한식	1130510100	서울특별시	강북구	미아동	\N	1130555500	송천동	113053005042	솔샘로
4	S0000005	치킨	1130510100	서울특별시	강북구	미아동	\N	1130555500	송천동	113053005041	삼양로

<배달데이터 확인>

```
# 데이터 불러오기
delivery_time_order_count = pd.read_csv('./data/delivery/시간-지역별 배달 주문건수.csv', header=None)
delivery_time_order_count.columns = ['날짜', '시간', '도', '구', '주문건수']

# 서울특별시만 골라내기
delivery_time_order_count = delivery_time_order_count.groupby('도')
df3 = delivery_time_order_count.get_group('서울특별시')

# 시간별 구분레지 -> 날짜 컬럼 삭제
df3.drop(df3.columns[0], axis=1, inplace=True)

# 0시 시간대 주문건수를 그룹핑 하지
df3 = df3.groupby('시간')
df = df3.get_group(0)


# 지도 데이터 패스 설정 및 데이터 지정
geo_path = './data/geojson/seoul_geo.json'
geo_str = json.load(open(geo_path, encoding='utf-8')) # 덕셔너리 객체로 리턴
pop = df

# folium 시각화
map = folium.Map(location=[37.5502, 126.982], zoom_start=12,
                 tiles='Stamen Terrain') # 'Stamen Terrain' / 'Stamen Toner'
fmap=folium.Choropleth(geo_data = geo_str,
              data = pop, # 인구수 데이터
              columns = ['구', '주문건수'], # 구별, 인구수로 컬럼을 가지고
              fill_color = 'PuRd', #PuRd, YlGnBu   // # 인구수 밀도로 색 표현
              key_on='feature.properties.name').add_to(map) # 키로에 지역 명를 넣어주는 것

# 지역 마우스오버 시 이름이 나오도록 지정
fmap.geojson.add_child(
    folium.features.GeoJsonTooltip(['name'],labels=False) # 해당 지역마다 툴팁으로 마우스오버 적용하게 만듦
```



<시간-지역별 배달주문건수 지도시각화>

```python
# 시간-지역별 배달 주문건수
import pandas as pd
from datetime import datetime

def dekiveryorder():

    # 데이터 불러오기
    delivery_time_order_count = pd.read_csv('./data/delivery/시간-지역별 배달 주문건수.csv', header=None)
    delivery_time_order_count.columns = ['날짜', '시간', '도', '구', '주문건수']

    # 서울특별시만 걸러내기
    delivery_time_order_count = delivery_time_order_count.groupby('도')
    df3 = delivery_time_order_count.get_group('서울특별시')

    # 날짜 컬럼 삭제 -> 날짜가 불분명해져도 괜찮. 나중에 mean()으로 통합
    df3.drop(df3.columns[0], axis=1, inplace=True)
    # display(df3)

    # 현재 시간 불러오기
    i = datetime.today().hour

    # 시간별로 묶기
    time_zero = df3.groupby('시간')
    time_zero = time_zero.get_group(i)

    # 구 별로 묶기
    gangnam = time_zero.groupby('구')
    gangnam = gangnam.get_group('강남구')
    # display(gangnam)

    # 예상 주문건수 --> 지금까지 있었던 모든 날짜의 주문건수를 더한 뒤 평균 값을 전달
#     print("강남구",i, "시 예상주문건수 :",gangnam['주문건수'].mean().round(1), "건")
    return "강남구 " + str(i) + "시 예상주문건수 :" + str(gangnam['주문건수'].mean().round(1)) + "건"

dekiveryorder()
```

'강남구 17시 예상주문건수 :40.3건'

<강남구 실시간 예상주문건수 파악>

```python
# 시간-지역별 배달 소요시간평균
import pandas as pd
from datetime import datetime

# 데이터 불러오기
delivery_time_mean = pd.read_csv('./data/delivery/시간-지역별 배달 소요시간평균.csv', header=None)
delivery_time_mean.columns = ['날짜', '시간', '도', '구', '평균시간']

# 서울특별시만 걸러내기
delivery_time_mean = delivery_time_mean.groupby('도')
df2 = delivery_time_mean.get_group('서울특별시')

# 시간별 구분하자 -> 날짜 컬럼 삭제
df2.drop(df2.columns[0], axis=1, inplace=True)
# display(df2)

# 현재 시간 불러오기
i = datetime.today().hour

# 시간별로 묶기
time_zero = df2.groupby('시간')
time_zero = time_zero.get_group(i)

# 구 별로 묶기
gangnam = time_zero.groupby('구')
gangnam = gangnam.get_group('강남구')
# display(gangnam)

# 예상 주문건수 --> 지금까지 있었던 모든 날짜의 평균시간을 더한 뒤 평균 값을 전달
print("강남구",i, "시 평균배달시간 :",gangnam['평균시간'].mean().round(1), "분")
```

강남구 16 시 평균배달시간 : 24.2 분

<강남구 실시간 평균배달시간 파악>

```python
# 시간-지역별 배달 평균주문금액
import pandas as pd
from datetime import datetime

# 데이터 불러오기
delivery_time_pay_mean = pd.read_csv('./data/delivery/시간-지역별 배달 평균주문금액.csv', header=None)
delivery_time_pay_mean.columns = ['날짜', '시간', '도', '구', '평균주문금액']

# 서울특별시만 걸러내기
delivery_time_pay_mean = delivery_time_pay_mean.groupby('도')
df4 = delivery_time_pay_mean.get_group('서울특별시')

# 시간별 구분하자 -> 날짜 컬럼 삭제
df4.drop(df4.columns[0], axis=1, inplace=True)

# 현재 시간 불러오기
i = datetime.today().hour

# 시간별로 묶기
time_zero = df4.groupby('시간')
time_zero = time_zero.get_group(i)

# 구 별로 묶기
gangnam = time_zero.groupby('구')
gangnam = gangnam.get_group('강남구')
# display(gangnam)

# 예상 주문건수 --> 지금까지 있었던 모든 날짜의 평균시간을 더한 뒤 평균 값을 전달
print("강남구",i, "시 평균주문금액 :",gangnam['평균주문금액'].mean().round(-2), "원")
```

강남구 16 시 평균주문금액 : 24500.0 원

<강남구 실시간 평균주문금액 파악>

2. 안전사고 데이터 전처리 및 시각화를 통한 인사이트 도출

- 공공데이터포털, 그리고 서울 열린데이터광장 등에서 배달 및 오토바이 사고에 대한 분석과 시각화를 진행해보았습니다. 이를 통해 서울 내에서 강남구의 교통안전지수가 가장 낮고, 오토바이 사고 건수가 가장

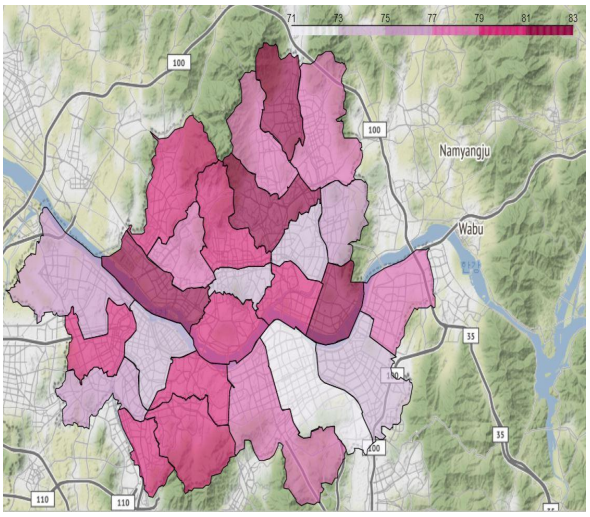
많다는 것을 알 수 있었습니다. 때문에 앞으로 서비스 구현의 예상지역을 강남구로 설정하였습니다.

사고 관련 데이터 확인

```
# ./fusion/data/accident
# 전년도도 같이 가져와야 함

accident_kind1 = pd.read_csv('./data/accident/서울시 교통사고 현황 (자동차종류별) 통계.txt', sep="\t")
accident_safe_count = pd.read_csv('./data/accident/서울시 교통안전지수 통계.txt', sep="\t")
accident_safetyproduct = pd.read_csv('./data/accident/서울시 보호장구 착용여부별 교통사고 사상자수 통계.txt', sep="\t")
accident_bike = pd.read_csv('./data/accident/서울시 자전거 교통사고 통계.txt', sep="\t")
accident_kind2 = pd.read_csv('./data/accident/서울시 차량용도별 교통사고 현황 통계.txt', sep="\t")
```

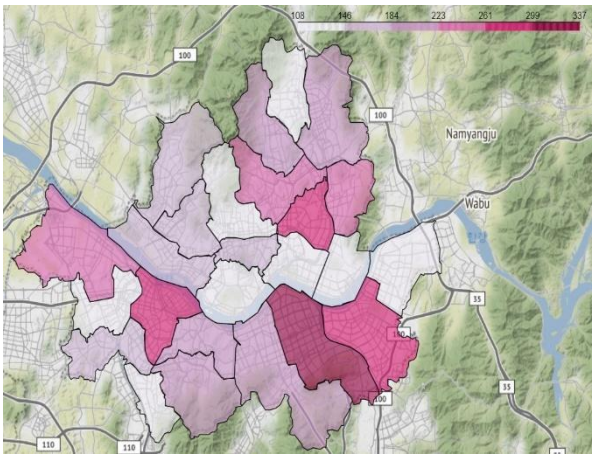
<사고 관련 데이터 확인>

```
# 서울시 교통안전지수 통계 시각화

import pandas as pd
import json
import folium

geo_path = './data/geojson/seoul_geo.json'
geo_str = json.load(open(geo_path, encoding='utf-8')) # 딕셔너리 객체로 리턴
pop = pd.read_excel('./data/accident/서울시 교통안전지수 통계_전처리.xlsx')

map = folium.Map(location=[37.5502, 126.982], zoom_start=11,
                 tiles='Stamen Terrain') # 'Stamen Terrain' / 'Stamen Toner'
fmap=folium.Choropleth(geo_data = geo_str,
              data = pop, # 인구수 데이터
              columns = ['자치구', '교통안전지수'], # 구별, 인구수로 컬럼을 가지고
              fill_color = 'PuRd', #puRd, YlGnBu  // # 인구수 밀도로 색 표현
              key_on='feature.properties.name').add_to(map) # 키은에 지역 명을 넣어주는 것
fmap.geojson.add_child(
    folium.features.GeoJsonTooltip(['name'],labels=False) # 해당 지역마다 툴팁으로 마우스오버 작동하게 만듦
)
display(map)

# 강남구가 제일 위험!!
map.save('output/서울시 교통안전지수 통계 시각화.html')
```



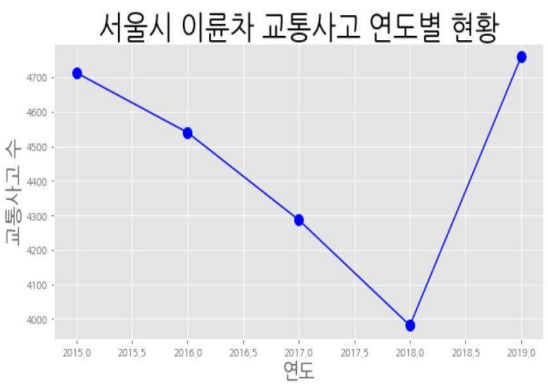
<서울시 교통안전지수 통계 시각화>

```
# 서울시 차량용도별 교통사고 현황 통계_이륜차 교통사고 발생 시각화

import pandas as pd
import json
import folium

geo_path = './data/geojson/seoul_geo.json'
geo_str = json.load(open(geo_path, encoding='utf-8')) # 딕셔너리 객체로 리턴
pop = pd.read_excel('./data/accident/서울시 차량용도별 교통사고 현황 통계_전처리.xlsx')

map = folium.Map(location=[37.5502, 126.982], zoom_start=11,
                 tiles='Stamen Terrain') # 'Stamen Terrain' / 'Stamen Toner'
fmap=folium.Choropleth(geo_data = geo_str,
              data = pop, # 인구수 데이터
              columns = ['자치구', '이륜차'], # 구별, 인구수로 컬럼을 가지고
              fill_color = 'PuRd', #puRd, YlGnBu  // # 인구수 밀도로 색 표현
              key_on='feature.properties.name').add_to(map) # 키은에 지역 명을 넣어주는 것
fmap.geojson.add_child(
    folium.features.GeoJsonTooltip(['name'],labels=False) # 해당 지역마다 툴팁으로 마우스오버 작동하게 만듦
)
display(map)

# 강남구가 제일 위험!!
map.save('output/서울시 차량용도별 교통사고 현황 통계_전처리 시각화.html')
```



<서울시 차량용도별(이륜차) 교통사고 시각화>

```
# 이륜차 연도별 사고 현황 선 그래프
import pandas as pd
import matplotlib.pyplot as plt

# 데이터 불러오기
df = pd.read_excel('./data/accident/서울시 교통사고 현황 (자동차종류별) 통계_전처리.xlsx', thousands=",")

# 스타일 서식 지정
plt.style.use('ggplot')

# 그림 사이즈 지정
plt.figure(figsize=(10,5))

# x축, y축 데이터를 plot 함수에 입력 => 시리즈의 인덱스를 x축 데이터로, 데이터값을 y축 데이터로 전달
plt.plot(df['기간'], df['이륜차'], marker='o', markersize=10, color='blue')

# 그래프 객체에 차트 제목을 추가할 때 title()함수를 이용한다.
plt.title("서울시 이륜차 교통사고 연도별 현황", size=30)

# 축 이름 추가
plt.xlabel("연도", size=20)
plt.ylabel("교통사고 수", size=20)

plt.savefig('./output/서울시 이륜차 교통사고 연도별 현황.png')
# 그래프 출력
plt.show()
```



<서울시 이륜차 교통사고 연도별 현황>

3. 회귀분석을 통한 상관관계 파악 **->** 없음

- 앞서 확인했던 배달데이터 중 서울을 중심으로 배달데이터를 회귀분석하여 주문건수에 영향을 주는 요인을 알아보고자 회귀분석을 시도하였습니다. 회귀분석을 위해 다음과 같은 순서를 거쳤습니다.

1) 데이터 확인

```
import pandas as pd

# 데이터 불러오기
df_delivery = pd.read_excel('./data/delivery/서울시간별통합파일.xlsx')
df_delivery.drop('Unnamed: 0', axis=1, inplace=True)
display(df_delivery.head())

df_classify = pd.read_excel('./data/delivery/서울성별나이별구분_201907.xlsx')
df_classify.drop('Unnamed: 0', axis=1, inplace=True)
display(df_classify.head())
```

	날짜	시간	시	구	평균시간	주문건수	평균주문금액
0	2019-07-18	0	서울특별시	구로구	22.65	16	23679
1	2019-07-18	0	서울특별시	동작구	22.17	1	15400
2	2019-07-18	0	서울특별시	영등포구	20.41	14	20864
3	2019-07-18	1	서울특별시	구로구	28.09	6	19667
4	2019-07-18	1	서울특별시	영등포구	22.19	8	18625

	기준연월	시	구	5세단위구분	총인구수	남성인구수	여성인구수
0	201907	서울특별시	종로구	20-24세	10947	5251	5696
1	201908	서울특별시	종로구	20-24세	10975	5272	5703
2	201909	서울특별시	종로구	20-24세	11032	5261	5771
3	201910	서울특별시	종로구	20-24세	10990	5238	5752
4	201911	서울특별시	종로구	20-24세	10950	5201	5749

<데이터 확인>

2) 데이터 병합을 위해 키 데이터를 생성한 후, on 옵션을 이용해 데이터 병합

- 키 데이터를 생성할 때, '연월' + '구' 를 합쳐 유니크한 데이터컬럼 생성

```
# 배달 데이터 전처리
import pandas as pd
import numpy as np
import re

# 전처리 참고 블로그 : https://blog.naver.com/hankrah/222085743804

# 타임테이블로 바꿔서 출력하자
df_delivery['타임테이블'] = pd.to_datetime(df_delivery['날짜'])
df_delivery['기준연월'] = df_delivery['드윈레이블'].dt.to_period(freq='M')

# 기준연월을 str로 변환
df_delivery['기준연월'] = df_delivery['기준연월'].astype(str)

# 기준연월에서 '-' 제거
# df_delivery['기준연월'] = re.sub('[-=.#/?:$]]', '', df_delivery['기준연월'])
df_delivery['기준연월'] = df_delivery['기준연월'].replace('[^\d.]', '',regex=True).astype(str)

# 유니코드 만들자 ->
df_delivery['유니크드'] = df_delivery['기준연월'] + df_delivery['구']


df_delivery.info()
display(df_delivery)
```

```
# 데이터 합쳐보기
df_EDA = pd.merge(df_delivery, df_classify, how='inner', on='유니코드')

# 필요없는 컬럼 제거
df_EDA.drop(df_EDA.columns[[7,8,9,10,11,12]], axis=1, inplace=True)

# 컬럼 이름 변경
df_EDA.rename(columns = {'시_x':'시', '구_x':'구'}, inplace=True)

# 데이터 확인
display(df_EDA)

# 데이터 저장
df_EDA.to_excel('./data/delivery/회귀분석데이터.xlsx')
```

	날짜	시간	시	구	평균시간	주문건수	평균주문금액	타임테이블	기준연월	유니코드
0	2019-07-18	0	서울특별시	구로구	22.65	16	23679	2019-07-18	201907	201907구로구
1	2019-07-18	0	서울특별시	동작구	22.17	1	15400	2019-07-18	201907	201907동작구
2	2019-07-18	0	서울특별시	영등포구	20.41	14	20864	2019-07-18	201907	201907영등포구
3	2019-07-18	1	서울특별시	구로구	28.09	6	19667	2019-07-18	201907	201907구로구
4	2019-07-18	1	서울특별시	영등포구	22.19	8	18625	2019-07-18	201907	201907영등포구
...	...	...	...	...	...	...	...	...	...	...
64642	2020-08-31	23	서울특별시	도봉구	22.76	4	30600	2020-08-31	202008	202008도봉구
64643	2020-08-31	23	서울특별시	양천구	24.33	8	26557	2020-08-31	202008	202008양천구
64644	2020-08-31	23	서울특별시	영등포구	24.36	44	18679	2020-08-31	202008	202008영등포구
64645	2020-08-31	23	서울특별시	용산구	33.07	1	20900	2020-08-31	202008	202008용산구
64646	2020-08-31	23	서울특별시	은평구	25.85	9	24667	2020-08-31	202008	202008은평구

<데이터 유니크코드 생성 및 통합>

3) 회귀분석을 위한 데이터 선별

```
# 필요없는 컬럼 unnamed:0, 날짜, 시, 구, 5세단위구분, 남성인구수, 여성인구수 제거
df.drop(df.columns[[0,1,3,4,8,10,11]], axis=1, inplace=True)

df.columns
```

```
Index(['시간', '평균시간', '주문건수', '평균주문금액', '총인구수'], dtype='object')
```

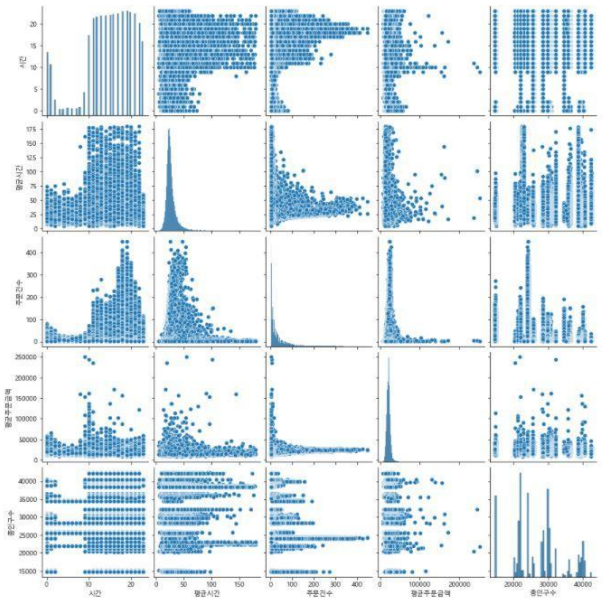
```
display(df.head())
print(df.shape)
```

	시간	평균시간	주문건수	평균주문금액	총인구수
0	0	22.65	16	23679	24384
1	1	28.09	6	19667	24384
2	9	20.25	3	15167	24384
3	10	21.37	21	14721	24384
4	11	22.12	73	24563	24384

```
(55615, 5)
```

<필요없는 컬럼 제거>

4) 시각화



<모든 변수 간의 시각화>

5) 스케일링 및 구간화

표준화를 진행해봅시다

```
# 일단 표준화부터 한번 해봅시다.
```

```
# pandas 형태로 정의된 데이터를 출력할 때, scientific-notation이 아닌 float 모양으로 출력
pd.options.mode.chained_assignment = None
```

```
# 피처 각각에 대한 scaling을 수행하는 함수를 정의
# 표준편차와 평균을 이용해 스케일링 (Z score)

def standard_scaling(df, scale_columns):
    for col in scale_columns:
        series_mean = df[col].mean()
        series_std = df[col].std()
        df[col] = df[col].apply(lambda x: (x-series_mean)/series_std)
    return df
```

```
# 피처 각각에 대한 scaling을 수행
scale_columns = ['평균시간', '평균주문금액', '총인구수']
re_df = standard_scaling(df, scale_columns)
```

```
re_df = re_df.rename(columns={'주문건수': 'y'})
re_df.head(5)
```

	시간	평균시간	y	평균주문금액	총인구수
0	0	-0.399011	16	0.399378	-0.37888
1	1	-0.027347	6	-0.334507	-0.37888
2	9	-0.562980	3	-1.157660	-0.37888
3	10	-0.486461	21	-1.239243	-0.37888
4	11	-0.435221	73	0.561082	-0.37888

<표준화>

```
# 시간 피처를 one-hot encoding으로 변환 --> 범주형이기 때문
time_encoding = pd.get_dummies(re_df['시간'])
re_df = re_df.drop(re_df.columns[0], axis=1)
re_df = re_df.join(time_encoding) # 합치기!
```

```
time_encoding.head(5)
```

	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 24 columns

	평균시간	y	평균주문금액	총인구수	0	1	2	3	4	5	...	14	15	16	17	18	19	20	21	22	23
0	-0.399011	16	0.399378	-0.37888	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	-0.027347	6	-0.334507	-0.37888	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	-0.562980	3	-1.157660	-0.37888	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	-0.486461	21	-1.239243	-0.37888	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	-0.435221	73	0.561082	-0.37888	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 28 columns

<시간 구간화>

6) 결과 확인

```
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

# 학습 데이터와 테스트 데이터로 분리
X = re_df[re_df.columns.difference(['y'])]
y = re_df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=19)
```

```
# 회귀 분석 모델을 학습
lr = linear_model.LinearRegression()
model = lr.fit(X_train, y_train)
```

```
# 회귀 분석 모델을 평가
print(model.score(X_train, y_train)) # train R2 score를 출력합니다.
print(model.score(X_test, y_test)) # test R2 score를 출력합니다.
```

```
0.2583872015758766
0.24870539112728418
```

```
# 회귀 분석 모델을 평가
y_predictions = lr.predict(X_train)
print(sqrt(mean_squared_error(y_train, y_predictions))) # train RMSE score를 출력합니다.
y_predictions = lr.predict(X_test)
print(sqrt(mean_squared_error(y_test, y_predictions))) # test RMSE score를 출력합니다.

# 차이의 제곱 값이 RMSE
```

```
39.3374843399196
39.9097209526107
```

<회귀분석 결과 확인>

R스퀘어값을 확인한 결과, **0.25**로 종속변수(예상주문건수)와 독립변수(다른 변수) 사이에는 큰 상관관계가 없다는 것을 알 수 있었습니다. 때문에 전진선택법이나 후진제거법, 단계적 선택법을 이용하더라도 크게 의미있는 상관관계를 얻을 수 없을 것이라 판단하여, 이후 과정은 진행하지 않았습니다.

4. 연관분석을 통한 배달 및 안전의 연관성 분석

- 배달라이더와 안전이 상관이 있는지에 대해 알아보고자 구글에서 키워드 '라이더'를 입력하여 뉴스 기사를 크롤링하였습니다. 이를 기반으로 명사를 추출하였고, 연관분석을 통해 '라이더'와 어떠한 단어가 연관이 있는지 알아보고자 하였습니다. 실제 연관분석을 수행한 결과 **'라이더', '배달', '안전'** 등이 많은 연관성을 가지는 것이 확인되었습니다.

1) 뉴스기사 크롤링

라이더 관련 뉴스기사 크롤링

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys # 엔터키 입력용
import pandas as pd

driver = webdriver.Chrome('C:/Temp/chromedriver') # 웹드라이버 객체 생성

# 구글을 열어서 키워드 입력 후 이동
driver.get('http://www.google.com') # 구글 홈페이지 내용 랜더링
target=driver.find_element_by_css_selector("[name = 'q']") # 검색어 지정
target.send_keys('라이더') # 타겟에 '라이더' 입력
target.send_keys(Keys.ENTER) # 엔터처리

# 뉴스 페이지로 이동
news = driver.find_element_by_css_selector('#hdtb-msb-vis > div:nth-child(4) > a')
news.click()


# 헤드라인 수집
untact_title = []
untact_time = []

# 알고리즘을 짜보자
# 다음 페이지 -> #pnnext > span:nth-child(2) // 이걸 19번해야 함 / 마지막에 누르자
for i in range(18):
```

```
df = pd.DataFrame(untact_time, untact_title)
display(df)
df.to_excel('./data/apriori/키워드_라이더.xlsx')
```

메쉬코리아, DB손해보험과 라이더 위한 보험 개발 나선다
전국배달라이더협회, 2020년 정책연구용역 착수보고회 개최
레다텍,밀라그룹 자율주행 셔틀에 라이더 공급
이마트에 '카트라이더 러쉬플러스'가 떴다
라이더 수수료 올리고 배달주문은 반값세일 - 서울경제
...
코로나발 언택트 열풍에 올라탄 넥슨, 피파·카트라이더 고속질주
Vet accused of inventing expert to greenlight heavy horse rider study
'비싼 몸' 배달 라이더 공급 잡는다...유현철 스파이더크래프트 ...
배달라이더 '억'대 연봉, 실현 가능할까[영상] - 서울경제
Queensland rider Baylee Nothdurft to take an extended break ...

<뉴스기사 크롤링>

특수문자 전처리

```
# 특수문자 전처리
import re

# 텍스트 정제 함수 : 한글 이외의 문자는 전부 제거합니다.
def text_cleaning(text):
    hangul = re.compile('[^ ㄱ-ㅣ가-힣]+') # 한글 처음부터 끝을 의미는 정규표현식
    result = hangul.sub('', text)
    return result
```

```
# '특수문자제거' 컬럼을 생성
df['특수문자제거'] = df['뉴스기사제목'].apply(lambda x: text_cleaning(x))
df.tail()
```

	뉴스기사제목	특수문자제거
172	코로나발 언택트 열풍에 올라탄 넥슨, 피파·카트라이더 고속질주	코로나발 언택트 열풍에 올라탄 넥슨 피파카트라이더 고속질주
173	Vet accused of inventing expert to greenlight ...	
174	'비싼 몸' 배달 라이더 공급 잡는다...유현철 스파이더크래프트 ...	비싼 몸 배달 라이더 공급 잡는다유현철 스파이더크래프트
175	배달라이더 '억'대 연봉, 실현 가능할까[영상] - 서울경제	배달라이더 억대 연봉 실현 가능할까영상 서울경제
176	Queensland rider Baylee Nothdurft to take an e...	

명사추출

```
# 명사추출을 위한 함수 정의
from konlpy.tag import Okt
from collections import Counter

# 한국어 약식 불용어사전 예시 파일입니다. 출처 - (https://www.ranks.nl/stopwords/korean)
korean_stopwords_path = "./data/apriori/korean_stopwords.txt"
with open(korean_stopwords_path, encoding='utf8') as f:
    stopwords = f.readlines()
stopwords = [x.strip() for x in stopwords]

def get_nouns(x):
    nouns_tagger = Okt()
    nouns = nouns_tagger.nouns(x)

    # 한글자 키워드를 제거합니다.
    nouns = [noun for noun in nouns if len(noun) > 1]

    # 불용어를 제거합니다.
    nouns = [noun for noun in nouns if noun not in stopwords]

    return nouns
```

```
# '명사' 컬럼에 이를 적용
df['명사'] = df['특수문자제거'].apply(lambda x: get_nouns(x))
print(df.shape)
df.tail()
```

	뉴스기사제목	특수문자제거	명사
172	코로나발 언택트 열풍에 올라탄 넥슨, 피파·카트라이더 고속질주	코로나발 언택트 열풍에 올라탄 넥슨 피파카트라이더 고속질주	[코로, 나발, 언택트, 열풍, 넥슨, 피파, 카트라이더, 고속, 질주]
173	Vet accused of inventing expert to greenlight ...		[]
174	'비싼 몸' 배달 라이더 공급 잡는다...유현철 스파이더크래프트 ...	비싼 몸 배달 라이더 공급 잡는다유현철 스파이더크래프트	[배달, 라이더, 공급, 유현, 스파이더, 크래프트]
175	배달라이더 '억'대 연봉, 실현 가능할까[영상] - 서울경제	배달라이더 억대 연봉 실현 가능할까영상 서울경제	[배달, 라이더, 억대, 연봉, 실현, 영상, 서울, 경제]
176	Queensland rider Baylee Nothdurft to take an e...		[]

<특수문자 전처리 및 명사 추출>

2) 라이더 키워드 연관분석

라이더 라는 키워드 연관분석

```
# apriori에 필요한 모듈 설치 -> pandas와 mlxtend 미리 설치해야 함
import pandas as pd
import re
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# 연관분석할 데이터셋
df = pd.read_excel('./data/apriori/키워드_라이더.xlsx')

# 리스트형식으로 바꾸기
dataset = df['명사'].apply(lambda x : eval(x).tolist())

# 데이터셋을 원핫인코딩처럼 트랜스포밍해주기!
te = TransactionEncoder()
te_result = te.fit(dataset).transform(dataset)


# 트랜스포밍된 결과를 데이터프레임으로 형성
# 컬럼명이 키워드 / 각 row마다 포함된 키워드에 True를 반환
df = pd.DataFrame(te_result, columns=te.columns_)


# apriori 알고리즘 사용 / 옵션을 주지 않으면 자동으로 지지도 0.5
keywordset = apriori(df, use_colnames=True)
display(keywordset)
```

	support	itemsets
0	0.127273	(뉴스)
1	0.172727	(라이더)
2	0.109091	(민족)
3	0.872727	(배달)
4	0.136364	(사고)
5	0.554545	(안전)
6	0.154545	(이륜차)
7	0.172727	(배달, 라이더)
8	0.109091	(라이더, 안전)
9	0.109091	(배달, 민족)
10	0.136364	(사고, 배달)
11	0.472727	(배달, 안전)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(배달)	(라이더)	0.872727	0.172727	0.172727	0.197917	1.145833	0.021983	1.031405
1	(라이더)	(배달)	0.172727	0.872727	0.172727	1.000000	1.145833	0.021983	inf
2	(라이더)	(안전)	0.172727	0.554545	0.109091	0.631579	1.138913	0.013306	1.209091
3	(안전)	(라이더)	0.554545	0.172727	0.109091	0.196721	1.138913	0.013306	1.029870
4	(배달)	(민족)	0.872727	0.109091	0.109091	0.125000	1.145833	0.013884	1.018182
5	(민족)	(배달)	0.109091	0.872727	0.109091	1.000000	1.145833	0.013884	inf
6	(사고)	(배달)	0.136364	0.872727	0.136364	1.000000	1.145833	0.017355	inf
7	(배달)	(사고)	0.872727	0.136364	0.136364	0.156250	1.145833	0.017355	1.023569
8	(배달)	(안전)	0.872727	0.554545	0.472727	0.541667	0.976776	-0.011240	0.971901
9	(안전)	(배달)	0.554545	0.872727	0.472727	0.852459	0.976776	-0.011240	0.862626

<라이더 키워드 연관성 분석>

해당 표에서 antecedents에 해당하는 키워드가 포함된 글에서 cosequents에 해당하는 키워드를 포함하는 글의 확률이 confidence입니다. confidence만을 보고 판단하였을 때, 라이더라는 키워드를 검색하였을 때 안전과 관련된 단어가 많으며 높은 상관관계를 가진다는 것을 파악할 수 있었습니다.

5. STT 인텐트 분석 기본 구조 설계

- STT나 TTS를 이루는 기술 자체를 개발하는 것은 힘들었기에, 중간과정에서 처리되는 데이터를 어떻게 분석할지에 대해 고민해보았습니다. 기본적으로 배달운행에 사용되는 발화내용은 한정적이라 예상했습니다. 때문에 특정 목적을 갖는 발화내용을 예상해 녹음하여 이를 STT로 변환한 후, 이를 형태소 분석하여 해당 intent을 분석하고자 하였습니다. 이러한 기본 마인드에서 더 나아가 AI가 각 발화내용별 고도화를 진행하였습니다.

운행시작을 판단해보자!!

```
import pandas as pd
from konlpy.tag import Okt


def drive_start(sample):
    correct_answer = ['운행시작', '운행', '시작']
    okt = Okt()
    input_answer = okt.nouns(sample)

    for input_answer_series in input_answer:
        if input_answer_series in correct_answer:
            print("여기에 운행 서비스를 연결해주는 코드를 집어넣자!")
            return True
        else :
            return False
```

```
test = '운행 시작 합니당!'
print(drive_start(test))

test1 = '집에 가고 싶어요'
print(drive_start(test1))
```

```
여기에 운행 서비스를 연결해주는 코드를 집어넣자!
True
False
```

6. AI 학습을 위한 STT API활용 학습데이터 구축

- AI에서 학습할 수 있는 데이터를 만들기 위해 어플리케이션 사용 시 등장할 발화내용을 녹음하였습니다. 이를 이용하여 STT처리를 통해 텍스트로 변환하고, 변환된 텍스트를 학습에 맞춰 전처리하였습니다. 그 과정 중 intent에 맞춰 라벨링을 주었습니다.

```
# 인식할 인텐트 리스트와 라벨링 리스트
intent_list = ['운행시작', '가게전화', '가게도착', '픽업완료', '영수증번호', '소요시간선택', '배달완료']

# 나중에 csv 파일 저장하기 전 시리즈를 저장할 리스트 생성
text = []
intent_index = []
label = []
labeling = -1 # 요러 라벨 인덱스에 들어갈 값 먼저 정해주기

# 인텐트 폴더이름을 for 문으로 돌리자
for intent in intent_list :
    path= "./data/음성녹음/연하_voice/"
    intent_path = intent
    labeling += 1
    print("여기서부터는 " + intent + " 폴더입니다.")

    # 폴더 내에서 파일을 하나씩 돌리자
    for k in range(30):

        try :
            # 파일명 지정
            file_path = str(k) + ".wav"

            # 구글 클라우드에서 speech 모듈 끌기
            from google.cloud import speech

            # 객체 생성
            client = speech.SpeechClient()

            # 파일 패쓰 지정
            file_name = os.path.join(os.path.dirname(path), intent_path, file_path)
```

	index	text	intent	label
0	0	운행 시작해	운행시작	0
1	1	운행 하자	운행시작	0
2	2	은행 해	운행시작	0
3	3	운행을 시작해	운행시작	0
4	4	운행을 시작하자	운행시작	0
...	...	...	...	...
135	135	음식 배달 끝났어	배달완료	6
136	136	배달 끝남	배달완료	6
137	137	배달 완료	배달완료	6
138	138	음식 전달했어	배달완료	6
139	139	음식 전달 완료	배달완료	6

<STT변환 및 데이터 저장>

```
import pandas as pd
import re

df = pd.read_excel("./data/text/최종.xlsx")
df = df.rename_axis('index').reset_index()
display(df)
# 한번에 어플라이로 적용은 불가능
# df.groupby("label").get_group(4).apply(lambda x : re.sub(r'(.)(\d)\s(\d)\s(\d)\s(\d)', r'\1\2\3\4\5', x))


# 새로운 컬럼을 만들어서 하는 수 밖에
data = df.groupby("label").get_group(4)
data['tt'] = data['text'].apply(lambda x : re.sub(r'(.)(\d)\s(\d)\s(\d)\s(\d)', r'\1\2\3\4\5', x))
display(data)
```

```
# 전처리 함수
# 정 안되면 리스트를 만들어 지우는 수 밖에

def re_sub(df):
    check_list = []
    for idx in (df['tt']):

        #전처리
        idx = re.sub(r',', '', idx) # 8,000 -> 8000

        # 삭제할 키워드 찾기
        check_point = idx[-4:]
        m = re.match('\d\d\d\d', check_point)
        # 삭제할 키워드이면 리스트 추가   영수증번호 8944 -> 영수증번호 894 / 일종2종
        if m == None:
            print(idx)
            print(df['index'])
```

<AI학습을 위한 데이터 전처리>

7. 자이로센서 예상데이터 실시간 이상치 설계 및 군집분석 시도 / 실제데이터 분석

- 프로젝트 중 자이로센서는 프로젝트 마감 2주전에 도착하였습니다. 그렇기에 보험데이터 설계를 위한 자이로센서 데이터가 없었습니다. 때문에 사전에 예상데이터를 난수로 만들어 실시간 이상치를 어떻게 감지할지, 그리고 이러한 데이터를 어떻게 분석할 지에 대한 설계를 진행하였습니다. 이때 난수는 정규화한 상태를 가정하여 -1에서 1사이의 값을 준 뒤, 직전값과 현재값의 차를 변화량으로 설정하였습니다. 이를 바탕으로 군집분석을 한 결과, 깔끔하게 군집이 되는 것을 알 수 있습니다.

IoT 데이터를 만들어 이상치를 잡아내보자

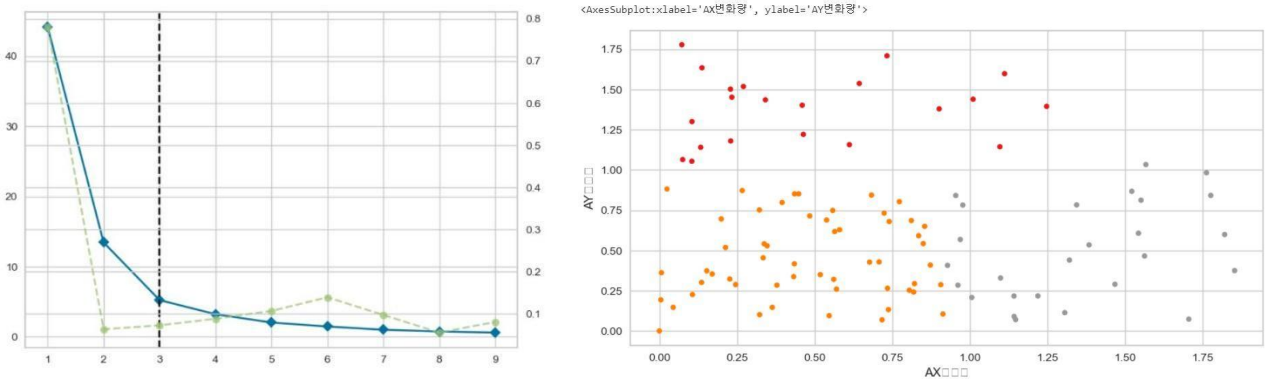
```
import pandas as pd
import random

ax = []
for x in range(101):
    a = random.uniform(-1,1)
    ax.append(a)

ay = []
for x in range(101):
    b = random.uniform(-1,1)
    ay.append(b)

az = []
for x in range(101):
    c = random.uniform(-1,1)
    az.append(c)

print(ax)
print(ay)
print(az)

df = pd.DataFrame([ x for x in zip(ax, ay, az)])
df.columns = ['AX', 'AY', 'AZ']
display(df)
df.to_csv('./data/motion/예상데이터.csv')
```

	AX	AY	AZ
0	0.093206	0.943231	-0.547067
1	0.326777	-0.508324	0.517453
2	0.113541	0.009926	0.486535
3	0.919444	0.262653	0.358042
4	0.400635	0.612231	0.846264
...	...	...	...
96	-0.322429	0.432167	-0.708846
97	-0.329087	0.070448	0.235886
98	-0.193040	0.371733	-0.840139
99	-0.931428	0.504205	0.747499
100	0.375414	0.617667	0.410820

<예상데이터 난수 컬럼 생성>

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

sns.set_style('whitegrid')

fig = plt.figure(figsize=(15,10))

sns.boxplot(y='AX변화량', data=df)
plt.show()
```

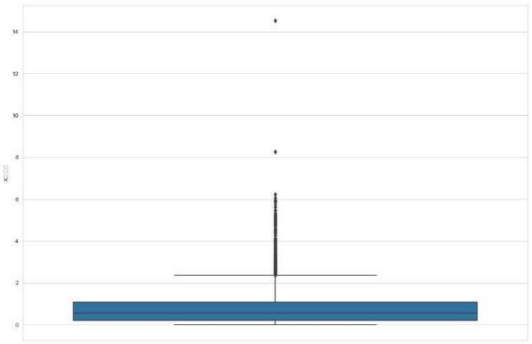
<박스 플롯을 통한 이상치 확인>



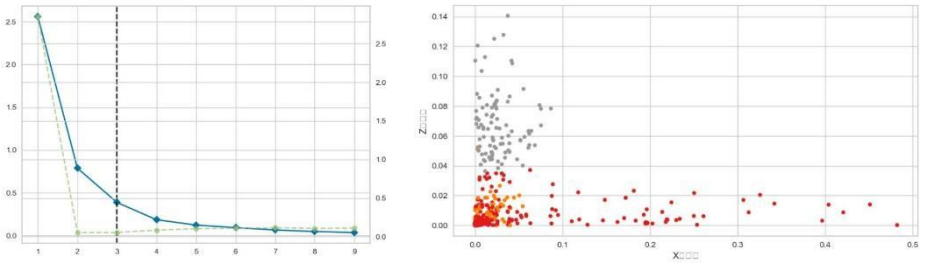
<군집 내 변동성 파악 및 군집분석>

- 하지만 실제데이터를 기반으로 이상치를 측정하고 군집분석을 한 결과, 깔끔하게 구별하는 것은 어려웠습니다. 때문에 비지도분석을 통한 분류보다 지도분석을 통한 분류가 필요할 것으로 예상됩니다.

```python
# 박스플롯으로 파악

import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

# 한글 및 마이너스 부호 깨짐현상 방지
from matplotlib import font_manager, rc
font_path = "./data/malgun.ttf"   #폰트파일의 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)
plt.rcParams['font.family'] = font_name
plt.rcParams['axes.unicode_minus'] = False # 마이너스부호 출력설정

sns.set_style('whitegrid')

fig = plt.figure(figsize=(15,10))

sns.boxplot(y='X변화량', data=gyro_data)
plt.show()
```



<박스플롯을 통한 이상치 확인>



<군집 내 변동성 파악 및 군집분석>

8. 보험(멘토링) / 이상치 감지 구조 설계 및 라즈베리 파이 내 구축

- 멘토님께 보험에 대한 조언을 듣고, 어떻게 보험이라는 영역을 배달 라이더 서비스에 구현할 수 있을지 고민해보았습니다. 고민한 결과, 자이로센서의 **X**축과 **Y**축이 주행에 많은 영향을 끼친다는 사실을 파악하여 해당 값을 주요변수로 설정하여 이상치를 찾아내도록 구상하였습니다. 예컨대 실시간으로 데이터가 쌓이는 자이로센서 데이터의 직전값과 현재값의 차이가 이상치 범위에 해당할 경우 **alert** 처리할 수 있도록 만들었습니다.

```python
# 우선 4분위로 나눌 값을 먼저 구합시다. ==> X 변화량

import pandas as pd

# 데이터 불러오기
gyro_df = pd.read_csv("./data/gyro/자전거.csv", sep=";")

display(gyro_df)

# 정규화용 평균 및 표준편차 구하기
print("X 평균 :",  gyro_df['accel_xout'].mean())
print("X 표준편차 :",  gyro_df['accel_xout'].std())
print("Y 평균 :",  gyro_df['accel_yout'].mean())
print("Y 표준편차 :",  gyro_df['accel_yout'].std())
print("Z 평균 :",  gyro_df['accel_zout'].mean())
print("Z 표준편차 :",  gyro_df['accel_zout'].std())

# 정규화하기
def standard_scaling(df, scale_columns):
    for col in scale_columns:
        series_mean = df[col].mean()
        series_std = df[col].std()
        df[col] = df[col].apply(lambda x: (x-series_mean)/series_std)
    return df

scale_columns = ['gyro_xout', 'gyro_yout', 'gyro_zout', 'accel_xout', 'accel_yout', 'accel_zout']
gyro_data = standard_scaling(gyro_df, scale_columns)
```

	1	gyro_xout	gyro_yout	gyro_zout	accel_xout	accel_yout	accel_zout	X변화량	Y변화량	Z변화량
0	1287	-0.012083	-0.077004	0.022206	-0.348025	1.635053	0.492454	0.074373	0.015157	0.064318
1	1288	-0.098304	-0.044756	-0.014539	-0.273652	1.619897	0.428136	0.042145	0.026179	0.106125
2	1289	0.411692	-0.050619	0.063828	-0.315797	1.646076	0.534261	0.168579	0.096450	0.043415
3	1290	0.741487	-0.060001	0.062527	-0.147218	1.742527	0.577675	0.200807	0.203924	0.093261
4	1291	0.969541	0.207369	-0.718860	-0.348025	1.538603	0.484414	0.037186	0.717867	0.035375
...	...	...	...	...	...	...	...	...	...	...
6718	8005	1.038517	2.533956	0.157802	1.166705	2.100771	0.412056	0.567714	0.049603	0.075574
6719	8006	-0.483711	-2.084624	-0.059087	0.598991	2.051168	0.487630	1.041222	0.128141	0.056278
6720	8007	0.234940	-6.227685	-0.441814	1.640213	1.923027	0.543908	2.347708	0.265928	0.630318
6721	8008	0.896253	-3.577439	-0.062664	3.987921	2.188955	-0.086410	1.452753	0.086805	0.192955
6722	8009	0.457820	-2.540208	-0.105587	2.535168	2.275760	0.106545	0.000000	0.000000	0.000000

<자이로센서 데이터 평균 및 표준편차 추출 / 정규화 / 변화량 컬럼 생성>

quantile 을 이용해 4분위를 파악하자

```
# X 값을 구해봅시다.

X_Q1 = gyro_data['X변화량'].quantile(.25)
X_Q2 = gyro_data['X변화량'].quantile(.5)
X_Q3 = gyro_data['X변화량'].quantile(.75)
X_Q4 = gyro_data['X변화량'].quantile(1)
X_IQR = X_Q3 - X_Q1 # IQR은 4분위 범위를 말한다.

# 제3사분위수 + 1.5*사분위범위
X_outlier = X_Q3 + 1.5*X_IQR

print("X변화량 1분위 값 :",X_Q1)
print("X변화량 2분위 값 :",X_Q2)
print("X변화량 3분위 값 :",X_Q3)
print("X변화량 4분위 값 :",X_Q4)
print("X변화량 이상치 범위 시작 :",X_outlier)

# 이상치는 어떤 게 제일 좋을까
```

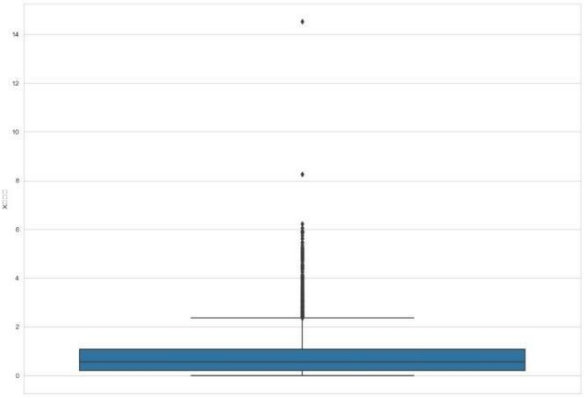
```
X변화량 1분위 값 : 0.21568169367526852
X변화량 2분위 값 : 0.5553183837156338
X변화량 3분위 값 : 1.0734502685217389
X변화량 4분위 값 : 14.534962873771255
X변화량 이상치 범위 시작 : 2.3601031307914444
```

```
# Y 값을 구해봅시다.

Y_Q1 = gyro_data['Y변화량'].quantile(.25)
Y_Q2 = gyro_data['Y변화량'].quantile(.5)
Y_Q3 = gyro_data['Y변화량'].quantile(.75)
Y_Q4 = gyro_data['Y변화량'].quantile(1)
Y_IQR = Y_Q3 - Y_Q1 # IQR은 4분위 범위를 말한다.

# 제3사분위수 + 1.5*사분위범위
Y_outlier = Y_Q3 + 1.5*Y_IQR

print("Y변화량 1분위 값 :",Y_Q1)
print("Y변화량 2분위 값 :",Y_Q2)
print("Y변화량 3분위 값 :",Y_Q3)
print("Y변화량 4분위 값 :",Y_Q4)
print("Y변화량 이상치 범위 시작 :",Y_outlier)

# 이상치는 어떤 게 제일 좋을까
```

```
Y변화량 1분위 값 : 0.12538560433533408
Y변화량 2분위 값 : 0.3141529427302876
Y변화량 3분위 값 : 0.6393287957318133
Y변화량 4분위 값 : 11.622281017236736
Y변화량 이상치 범위 시작 : 1.4102435828265323
```

<자이로센서 X,Y 축 변화량의 4분위 값>

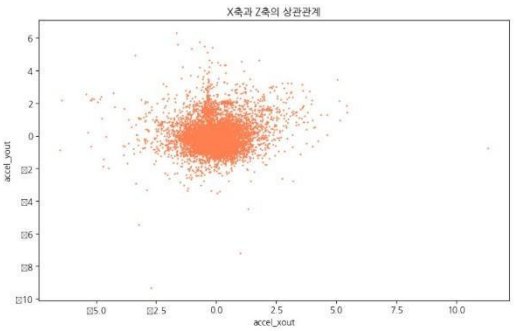
```
# 박스플롯으로 파악

import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

# 한글 및 마이너스 부호 깨짐현상 방지
from matplotlib import font_manager, rc
font_path = "./data/malgun.ttf"   #폰트파일의 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)
plt.rcParams['font.family'] = font_name
plt.rcParams['axes.unicode_minus'] = False # 마이너스부호 출력설정

sns.set_style('whitegrid')

fig = plt.figure(figsize=(15,10))

sns.boxplot(y='X변화량', data=gyro_data)
plt.show()
```



<박스플롯을 통한 X변화량 이상치 파악>

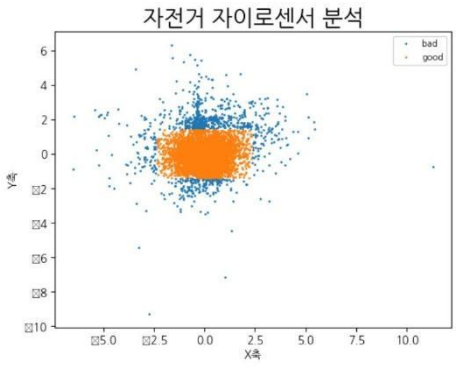
군집분석을 시도해봅시다.

```
# 군집분석을 어떻게 할 것인가? -> 산점도부터 그려보자!!

# 먼저 accel_xout과 accel_zout을 산점도를 그려보자
plt.style.use('default')

gyro_data.plot(kind='scatter', x='accel_xout', y='accel_yout', c='coral', s=1, figsize=(10,6))
# plt.xlim(0, 1)
# plt.ylim(0, 0.5)
plt.title('X축과 Z축의 상관관계')
plt.show()
```



<산점도를 통한 데이터 분포 파악>

```
# 참고사이트 : https://rfriend.tistory.com/414

# Scatter plot with a different color by groups

groups = gyro_data.groupby('label')
plt.figure(figsize=(10,6))
plt.style.use('default')


fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.accel_xout,
            group.accel_yout,
            marker='o',
            markersize=1,
            linestyle='',
            label=name)

ax.legend(fontsize=8, loc='upper right') # legend position
plt.title('자전거 자이로센서 분석', fontsize=20)
plt.xlabel('X축', fontsize=10)
plt.ylabel('Y축', fontsize=10)
plt.show()
```



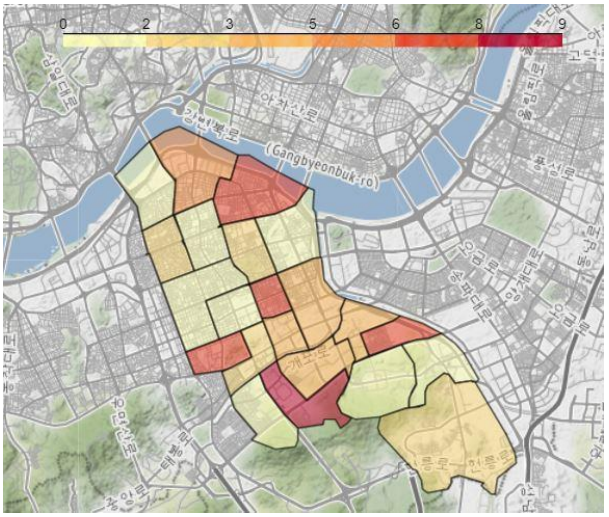
<이상치를 기준으로 데이터 분류분석 시각화>

9. 보험데이터를 통한 모니터링 페이지 시각화

- 누적된 보험데이터를 실시간으로 파악하고 관리하기 위해 가상 데이터를 만들어 시각화를 진행하였습니다. 예컨대 유저의 일일 위험감지 횟수, 특정지역 위험주행횟수 등을 시각화하였습니다. 이러한 시각화는 클라우드가 진행한 모니터링 페이지에서 확인할 수 있도록 제공하였습니다.

강남구 위험 시각화 (지도시각화)

```
import pandas as pd
import json
import folium


# https://blog.naver.com/kcchang61/221350672356

# 구단위 지도 그리기
geo_path = './data/geojson/new_gangnam_geo.json'
# geo_path = './data/geojson/hankuk2_geo.json'
geo_str = json.load(open(geo_path, encoding='utf-8')) # 덕셔너리 객체로 리턴


# geojson과 동별 인구수를 합친 엑셀 파일 불러오기
final_data = pd.read_excel("./data/geojson/예상데이터_last_final.xlsx")
final_data.columns = final_data.columns.map(str)

# display(final_data.head())


# # 동별 구분 경계선
map = folium.Map(location=[37.490, 127.055], zoom_start=12,
                 tiles='Stamen Terrain') # Stamen Toner / Stamen Terrain
```



<강남구 동별 위험주행횟수 시각화>

재환이는 데이터값만 받고 싶대!! 그럼 어떻게?!

```
# 위의 코드를 실행해야 해당 코드도 실행이 됩니다!!
# 이거는 user1을 바탕으로 짠 코드입니다. ==> 유저마다 바꾸고 싶으면 groupby에서 바꿔야 함!

# 요거는 날짜데이터! -> 리스트에 넣자!
day_list = day_df['day'].unique()

# 날짜하고 알러트만 넣어봅시다
day_df_new = day_df[['alertCount', 'day']]

# 날짜끼리 묶어서 계산하기 편하게
day_df_value = day_df_new.groupby("day")

# for 문으로 한번 출력해보자!
for day_series in day_list:
    a = day_df_value.get_group(day_series)

    b = a['alertCount'].mean()
    b.astype(int)
#     print(a)
    print(b)
    print("++++++++++")
```

```
6.0
++++++++++
1.0
++++++++++
8.0
++++++++++
9.0
++++++++++
5.0
++++++++++
7.0
++++++++++
1.0
++++++++++
4.0
++++++++++
0.0
++++++++++
5.0
++++++++++
```

<특정 유저의 날짜별 위험주행횟수>

느낀 점 및 보완할 점

느낀 점

- 처음 융복합 프로젝트를 진행할 때는 많은 걱정이 있었습니다. AI는 빅데이터와 관련이 있는 부분이 있어 어느 정도 파악할 수 있었지만, IoT나 Cloud와 같은 경우에는 어떠한 사전지식도 없었기 때문입니다. 하지만 프로젝트를 진행하면서 각 분야에 대한 설명을 팀원에게 들을 수 있었고, 해당 분야에 대한 인지범위를 넓힐 수 있었습니다. 이번 프로젝트를 진행하며 단순히 IT역량 뿐만 아니라 커뮤니케이션 역량 등을 제고하며 많은 것을 배웠습니다.

보완할 점

1. 회귀분석을 위한 데이터수집 프로세스를 구축해야 한다.

- 회귀분석을 진행하며 부족한 예상주문건수를 예측하기 위해서는 더 많은 변수가 필요하다고 생각하였습니다. 때문에 보험데이터를 저장하는 과정에서 예측에 필요한 데이터를 추가적으로 지정한다면, 이후 더욱 효과적인 데이터 분석을 이룰 수 있으리라 판단됩니다.

2. 이상치 분석을 세분화하여 위험등급 정도를 나누어보아야 한다**.**

- 이상치 분석은 단순히 위험감지만을 잡아낼 수 있도록 설계하였습니다. 때문에 이를 **U**턴위험감지, 급회전 위험감지, 급제동 위험감지 등과 같이 특정 위험감지 분류를 이룰 수 있다면, 보험설계 측면에서 더욱 효과적일 것이라 판단됩니다. 또한 특정 주기를 반복하거나 지나치게 안정적인 치트데이터를 찾아내는 과정을 수행한다면 보험료 책정 부분에 도움이 될 것이라 생각합니다.

3. 비지도학습**(**군집분석**)**으로 분류가 되지 않는다면**,** 각 유형별**(**유턴**,** 회전 등**)** 규칙을 찾아내고 **AI**가 학습할 수 있도록 라벨링을 해야한다**.**

- 군집분석으로 데이터를 분류하려 시도하였지만, 적절한 분류를 이루지는 못했습니다. 때문에 각 유형별 라벨링 등을 통해 지도학습으로 **U**턴 혹은 급제동 등을 찾아낼 수 있다면, 자이로센서 분석을 좀 더 고도화시킬 수 있으리라 생각합니다.

데이터 목록

- 배달데이터**(KT** 빅데이터 플랫폼**) :**
  1. 배달상점 데이터 (42769 rows × 11 columns)
  2. 시간지역별 배달 소요시간 평균 (375220 rows × 5 columns)
  3. 시간 지역별 배달 주문건수 (381068 rows × 5 columns)
  4. 시간 지역별 배달 평균주문금액 (374561 rows × 5 columns)
  5. 업종 지역별 배달 주문건수 (197192 rows × 5 columns)
  6. 업종 지역별 평균배달소요시간 (195986 rows × 5 columns)
  7. 업종 지역별 평균주문금액(191306 rows × 5 columns)
  8. 주문지역 인구 특성 (157374 rows × 8 columns)
- 연관분석 **:** 구글 기사 크롤링 (185개)
- **AI**학습데이터 **:** 녹음파일 (660개)
- 보험데이터 **:** 자이로센서 (자전거 – 6723 rows × 7 columns)

참고사이트

1. 데이터 전처리

- https://blog.naver.com/hankrah/222085743804

2. 텍스트 연관분석

- https://needjarvis.tistory.com/59
- https://lemontia.tistory.com/903

3. 군집분석

- https://tariat.tistory.com/819
- https://m.blog.naver.com/samsjang/221017639342

4. 이상치 감지

- http://www.databaser.net/moniwiki/wiki.php/%EC%9D%B4%EC%83%81%EC%B9%98%EC%A0%9C%EA%B1%B0%EB%B0%A9%EB%B2%95
- https://sosomemo.tistory.com/34

5. 지도시각화

- https://blog.naver.com/kcchang61/221350672356