

프로젝트 보고서

배정 : Bbeojung.kr

정류장 혼잡도 예측 ML 모델 개발 및 서비스



2022년 6월 19일

전남대학교

통계학과

온 정 완

백지 간지

프로젝트 보고서

URL Link

<http://Bbeojung.kr>

Github Link

<https://github.com/On-JungWoan/BusStop-Congestion-Project>

2022년 06월 19일

전남대학교

통계학과

온 정 완

목 차

제 1 장 About Bbeojung	1
제 2 장 Model Engineering	2
2.1 Prepare Library & Dataset	2
2.2 EDA	5
2.2.1 Target 데이터 분포 확인	5
2.2.2 Target 데이터 간 상관관계 확인	6
2.2.3 정류장의 고유값 결정	8
2.2.4 결측치 확인	10
2.3 Data Cleansing & Pre-Processing	12
2.3.1 임의의 ARS_ID 값 부여	12
2.3.2 Drop NA	12
2.3.3 target_time 선택	13
2.3.4 target_time 생성	14
2.3.5 Train - Test Split	14
2.4 Feature Engineering	15
2.4.1 Distance Feature	15
2.4.2 Population Feature	20
2.4.3 주요 Variable 요약통계량	23
2.4.4 혼잡도 가중치	27
2.4.5 공휴일 및 주말 여부	28
2.4.6 기상 데이터	29

2.4.7	요일 데이터 One-Hot Encoding	29
2.4.8	주요 Variable Label Encoding	30
2.5	Train & Save Model	31
2.5.1	Train & Test Data	31
2.5.2	Set Up Data	32
2.5.3	Find Best Model (By R2)	33
2.5.4	Hyper Parameter Tuning	34
2.5.5	Model Blending	35
2.5.6	Model Ensemble	36
2.5.7	Predict & Evaluate Model	37
2.5.8	Dump & Save Model	38
제 3 장	Web Building	43
3.1	BackEnd Building	43
3.1.1	Server	43
3.1.2	Domain	45
3.1.3	Model & DB	46
3.1.4	URL & View	49
3.2	FrontEnd Building	54
제 4 장	향후 활용 방안	55
부 록		56
참고문헌		61

스 크 릩 트 목 차

[Script 2.1.1] read DataSet	3
[Script 2.2.1] 3~4월 요일별 승객 plot	5
[Script 2.2.2] 3월13~4월 요일별 승객 plot	5
[Script 2.2.2] 사용자 정의 함수 time_merge()	6
[Script 2.2.3] time_merge()	6
[Script 2.2.4] 인원 간 HeatMap	6
[Script 2.2.5] ARS_ID 고유값 개수 확인	8
[Script 2.2.6] ARS_ID 고유값 관련	8
[Script 2.2.7] 고유값 개수 3이상 출력	9
[Script 2.2.8] 동산마을 ARS_ID 확인	9
[Script 2.2.9] null값 확인	10
[Script 2.2.10] 결측치 시각화	10
[Script 2.2.11] ARS_ID : 정류장명 1:1 매칭 확인	11
[Script 2.2.12] ARS_ID : 정류장명 1:1 매칭 확인	11
[Script 2.2.13] ARS_ID : 정류장명 1:1 매칭 확인	11
[Script 2.3.1] 임의의 ARS_ID 부여	12
[Script 2.3.2] Drop NA	12
[Script 2.3.3] Target Time 선택	13
[Script 2.3.4] Target Time 생성	14
[Script 2.3.5] Train - Test Split	14

[Script 2.4.1] Distance Feature1	15
[Script 2.4.2] Distance Feature2	16
[Script 2.4.3] Distance Feature3	17
[Script 2.4.4] Distance Feature4	18
[Script 2.4.5] Distance Feature5	19
[Script 2.4.6] Population Feature	20
[Script 2.4.7] Population Feature2	21
[Script 2.4.8] Population Feature3	22
[Script 2.4.9] 요약통계량1	24
[Script 2.4.10] 요약통계량2	25
[Script 2.4.11] 요약통계량3	26
[Script 2.4.12] 혼잡도 가중치1	27
[Script 2.4.13] 혼잡도 가중치2	27
[Script 2.4.14] 공휴일 및 주말 여부1	28
[Script 2.4.15] 공휴일 및 주말 여부2	28
[Script 2.4.16] 기상 데이터	29
[Script 2.5.1] Make Train Test Data	31
[Script 2.5.2] Setup Pycaret	32
[Script 2.5.3] Compare Model	33
[Script 2.5.4] Hyper Parameter Tuning	34
[Script 2.5.5] Model Blending	35
[Script 2.5.6] Model Ensemble(Bagging)	36
[Script 2.5.7] Predict & Evaluate Model	37
[Script 2.5.8] Dump & Save Model	38
[Script 2.5.9] Save Model	39

[Script 3.1.1] Mean Sum Model	46
[Script 3.1.2] Congestion Model	47
[Script 3.1.3] Distance Model	47
[Script 3.1.4] BusInfo Model	48
[Script 3.1.5] BackEnd 흐름도	49
[Script 3.1.6] urls.py	50
[Script 3.1.7] views.py	50

그 립 목 차

[그림 2.1.1] 사용 Library	2
[그림 2.1.2] 원시 CSV 파일	3
[그림 2.1.3] 원시 DataSet	4
[그림 2.1.4] 사용 DataSet	4
[그림 2.2.1] 3~4월 요일별 승객 plot	5
[그림 2.2.2] 3월13~4월 요일별 승객 plot	5
[그림 2.2.3] 인원 간 HeatMap	7
[그림 2.2.4] ARS_ID : 정류장명 고유값 수	8
[그림 2.2.5] ARS_ID : 정류장명 고유값 관련	8
[그림 2.2.6] 고유값 수가 3개 이상인 정류장 확인	9
[그림 2.2.7] 고유값 수가 3개 이상인 정류장 확인	9
[그림 2.2.8] 결측치 확인	10
[그림 2.2.9] 결측치 시각화(mnso)	10
[그림 2.2.10] 정류장명은 null이 아니지만 ARS_ID는 null인 경우	11
[그림 2.2.11] ARS_ID는 null이 아니지만 정류장명 null인 경우	11
[그림 2.2.12] ARS_ID가 없는 정류장	11
[그림 2.2.13] ARS_ID가 없는 정류장	11
[그림 2.3.1] Null값의 수	12
[그림 2.3.2] Target Time이 선택된 df	13
[그림 2.3.3] 전처리 끝난 df	14

[그림 2.4.1] 정류장별 탑승객 데이터	15
[그림 2.4.2] location df	16
[그림 2.4.3] 주요 9개 정류장	17
[그림 2.4.4] distance feature가 추가된 df	19
[그림 2.4.5] 위도, 경도, dong_name이 추가된 df	21
[그림 2.4.6] 법정동 및 행정동	21
[그림 2.4.7] population df	22
[그림 2.4.8] population이 추가된 df	22
[그림 2.4.9] 요약통계량	25
[그림 2.4.10] 요약통계량이 추가된 df	26
[그림 2.4.11] 혼잡도 가중치가 추가된 df	27
[그림 2.4.12] 공휴일	28
[그림 2.4.13] weekend가 추가된 df	28
[그림 2.4.14] 기상데이터가 추가된 df	29
 [그림 2.5.1] Train / Test Shape	 31
[그림 2.5.2] Setup	32
[그림 2.5.3] Model Score	33
[그림 2.5.4] Model Blending	35
[그림 2.5.5] Model Ensemble(Bagging)	36
[그림 2.5.6] 06~10 Model Score	39
[그림 2.5.7] 10~14 Model Score	39
[그림 2.5.8] 14~18 Model Score	40
[그림 2.5.9] 18~22 Model Score	40
[그림 2.5.10] 06~08 Model Plot	41

[그림 2.5.11] 08~10 Model Plot	41
[그림 2.5.12] 10~12 Model Plot	41
[그림 2.5.13] 12~14 Model Plot	41
[그림 2.5.14] 14~16 Model Plot	42
[그림 2.5.15] 16~18 Model Plot	42
[그림 2.5.16] 18~20 Model Plot	42
[그림 2.5.16] 20~22 Model Plot	42
[그림 3.1.1] 사용 Server DB	43
[그림 3.1.2] 웹 흐름도	44
[그림 3.1.3] MobaXterm	44
[그림 3.1.4] Github Repository	45
[그림 3.1.5] Domain	45
[그림 3.1.6] SQLite	48
[그림 3.1.7] BackEnd 흐름도	49
[그림 3.1.8] Web Index 페이지	51
[그림 3.1.9] Web Detail 페이지	52
[그림 3.1.10] Detail : Search Box	52
[그림 3.1.11] Detail : Navigation Bar	53

제 1 장 About Bbeojung

광주는 지하철이 수도권에 비해 발달되지 않아 사람들이 가장 많이 사용하는 대중교통은 버스이다. 그렇기 때문에 광주 시민이라면 누구나 버스에 탑승하기 위해 정류장에 갔지만 인구 밀도가 높아 불편함을 겪은 적이 있을 것이다. 이런 상황에서 나는 불편함을 느끼고 "정류장에 사람이 얼마나 밀집되어 있는지 미리 알 수 있을까?"라는 생각이 들었다.

또한 covid-19 사태가 완화되면서 22년 5월 2일 실외 마스크 착용 의무화가 해제되어 마스크를 착용하지 않는 사람이 늘어났다. 버스 정류장처럼 사람이 정적으로 모여있을 수 있는 곳에서는 인구가 얼마나 밀집되어 있는지 알고 미리 착용하면 어떨까라는 생각도 들었다. 이런 생각들은 빼정 프로젝트를 만들었고, 정류장 인구 밀도와 더불어 추가적으로 사용자에게 다양한 정보를 제공하려 한다.



Developer : 온정완

제 2 장 Model Engineering

코드 작업 환경은 Google의 Colab과 Jupyter NoteBook 사용.

2.1 Prepare Library & Dataset

2.1.1 사용 Library

[그림 2.1.1]

분류	Library name	Method name	설명
Data Handling	pandas	DataFrame() merge(), concat() 등	Data Frame 조작
	numpy	array() 관련 메소드	Array 조작
	datetime	datetime() weekday()	Weekdat Feature 생성
	sklearn. preprocessing	minmaxscaler()	Min-Max 스케일링
		labelencoder()	Label 스케일링
Data Visulizing	matplotlib. pyplot	plot()	실제값 / 예측값 plot 생성
	seaborn	heatmap()	Target 데이터 간 Heatmap 생성
	missingno	matrix()	결측치 시각화
	tqdm	tqdm()	for문 시각화
	folium	Map() Marker()	지도 관련 시각화
Model	sklean. model_selction	train_test_split()	train / test 데이터 분리
	sklearn. ensemble	RandomForestRegr essor() fit()	학습 회귀 모델
	sklearn. metrics	mean_squared_error ()	MSE 값
		mean_absolute_erro r()	MAE 값
		r2_score()	R2(결정 계수) 값
	joblib	dump()	Model을 pickle 파일로 Export

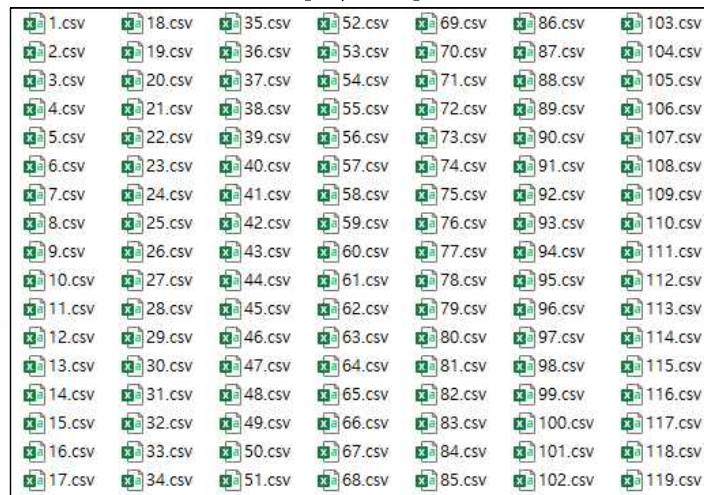
Location	geopy. distance	vincenty()	거리 계산
	googlemaps	reverse_geocode()	경,위도에 해당하는 주소값 반환 API
etc	google. colab	drive()	google drive에 접근
	warnings	filterwarnings()	Warning 무시

2.1.2 Load DataSet

원시 데이터셋은 7,460,871개의 관측치, 15개의 Sheet, 8개의 Excel 파일로 구성되어 있었다. Python에서 외부 데이터를 Import할 때, CSV 파일이 Excel 파일 대비 월등히 빠른 데이터 처리 속도를 보여주기때문에 총 120개의 sheet를 CSV 파일로 변환하여 Import 해주었다.

Google Drive에 미리 저장해 둔 120개의 CSV 파일을 리스트에 계속 추가해주는 방식으로 데이터를 불러왔고, 한글이 깨지는 현상이 발생하여 encoding은 cp949로 진행하였다.

[그림 2.1.2]



[Script 2.1.1]

```
path = "/content/drive/MyDrive/BusProject/NumofPassenger/CSV/"
df_list = []
```

```
for i in tqdm( range(1,120) ):
```

```
    df_list.append( pd.read_csv(path+str(i)+".csv", encoding='cp949') )
```

이후 df_list에 있는 모든 CSV 파일을 행기준으로 합쳐주고, object type으로 되어있는 '일자'열을 yyyy-mm-dd형태의 datetime으로 변환해주었다. 아래는 데이터 셋의 상위 5번째 줄

을 출력해 준 모습.

[그림 2.1.3]

	일자	노선명	정류장명	ARS_ID	시간	승하차	거래건수
0	2022-03-01	1187	4수원지	4263.0	10	승차	1
1	2022-03-01	1187	4수원지	4264.0	12	승차	1
2	2022-03-01	1187	4수원지	4264.0	13	하차	1
3	2022-03-01	1187	4수원지위	4268.0	11	하차	1
4	2022-03-01	1187	4수원지위	4268.0	13	하차	2

[그림 2.1.4]

	출처	내용
광주_버스_이용객.csv	공공데이터 포털	광주광역시 3~4월 버스 이용객
광주광역시_정류장_위치정보.csv	공공데이터 포털	정류장의 경도, 위도 데이터
Google Geocoding API	Google Maps	해당 경/위도의 주소 정보를 반환해 줌.
광주광역시_행정지역_인구.csv	KOSIS	광주광역시의 행정지역(동)별 인구 데이터셋
법정동_행정동_맵핑.csv	공공데이터 포털	전국 법정동과 행정동 맵핑 데이터
2018~2022_공휴일_정보.csv	공공데이터 포털	2018~2022 공휴일 정보
기상청_기상_데이터.csv	기상자료개방포털	기온, 습도, 강수량, 풍속

다음은 원시 데이터셋 이외에 추가적으로 사용해 준 DataSet에 대한 Table이다. 자세한 출처는 부록에서 기술하고 있다.

-
-
-

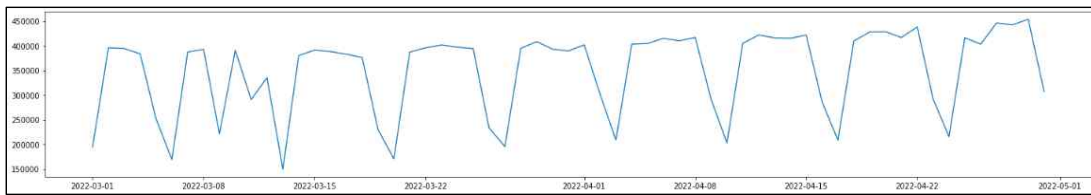
2.2 EDA

2.2.1 Target 데이터 분포 확인

[Script 2.2.1]

```
fig = plt.figure(figsize=(25,4))
fig.add_subplot()
plt.plot(df.groupby('일자')['거래건수'].sum())
```

[그림 2.2.1]

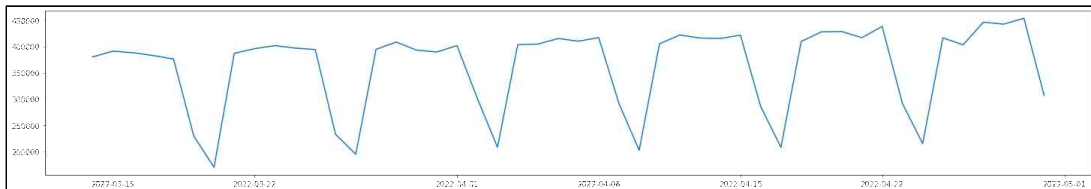


데이터를 일자를 기준으로 그룹화(sum)하여 거래건수에 대한 plot을 그려본 결과, 일주일을 기점으로 일정한 패턴이 나타나는 것을 확인하였다.

[Script 2.2.2]

```
df = df[df.일자>'2022-03-13'].reset_index()
df = df.drop(['index'],axis=1)
df.head()
fig = plt.figure(figsize=(25,4))
fig.add_subplot()
plt.plot(df.groupby('일자')['거래건수'].sum())
```

[그림 2.2.2]



Seasonal effect를 고려하고, 더 정확한 패턴을 확인하고자 개강, 개학 등 특수한 이벤트가 많이 있는 시기인 3월 첫 째, 둘 째주 데이터를 drop 해주었다. 그리고 다시 일자를 기준으로 데이터를 그룹화하여 plot을 그려주었다.

그 결과, 일주일을 간격으로 평일에는 탑승객이 많아졌다가 주말에는 탑승객이 줄어드는 뚜렷한 패턴을 확인할 수 있었다.

2.2.2 Target 데이터 간 상관관계 확인

[Script 2.2.2]

```
def time_merge(d, start, x):
    while True:
        res = pd.DataFrame()
        start %= 24

        if ( (start>0 and start<5) or (start+1 not in df.시간.unique().tolist()) ): break
        res[str(start)+'_'+str(start+1)+'_r']= d[str(start)+'_r']+ d[str(start+1)+'_r']
        res[str(start)+'_'+str(start+1)+'_t']= d[str(start)+'_t']+ d[str(start+1)+'_t']

        start += 2
    return res
```

사용자 정의 함수 time_merge는 매개 변수로 데이터 프레임과 time의 시작 및 끝 값을 넣어주면 2시간 단위로 column을 묶어주는 함수이다. 최종적으로, 2시간 단위로 column이 묶인 데이터 프레임을 return한다.

[Script 2.2.3]

```
time_merge(df, 6, 2)
```

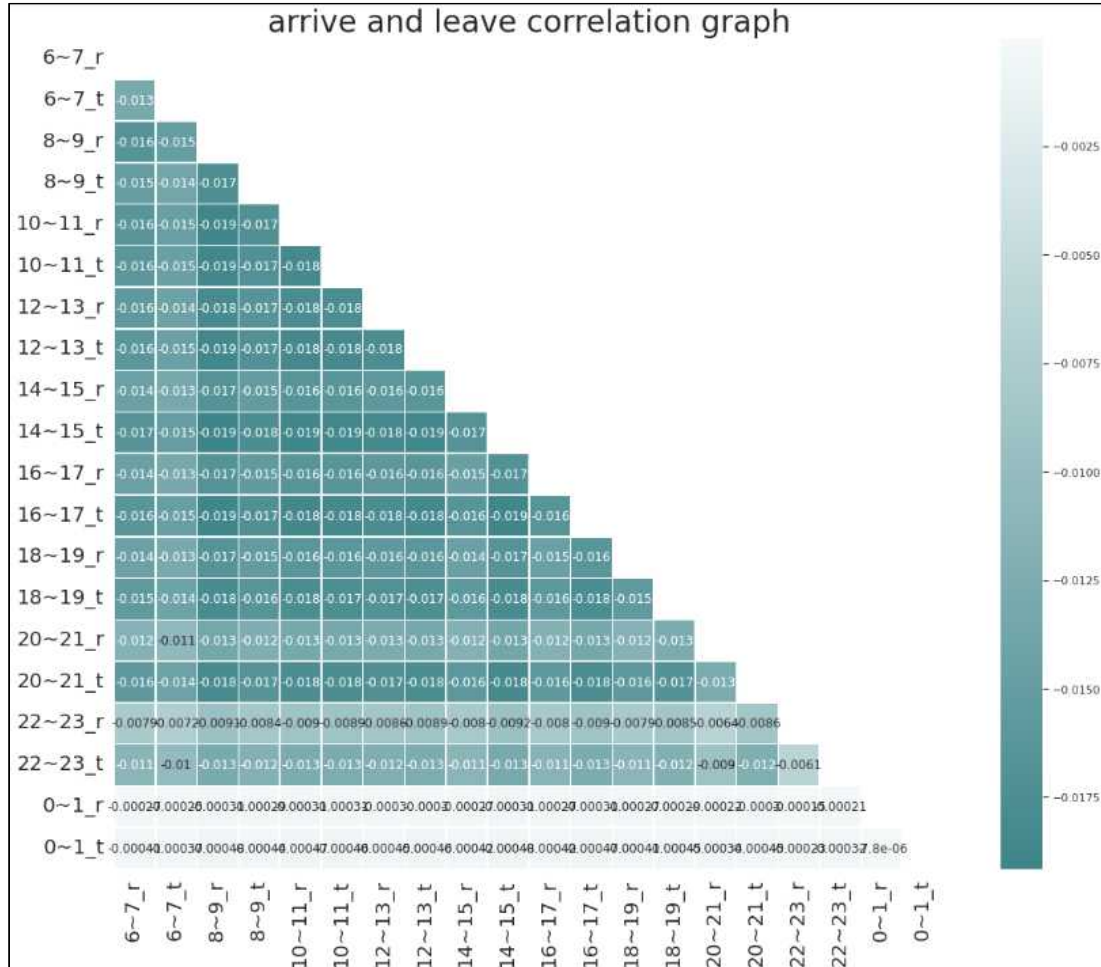
위와 같이 df에 대해서 시작값 6, 끝 값 2를 주면 새벽 6시부터 다음날 새벽2시까지 2시간 단위로 column을 묶어준다. start%=24를 했기 때문에 24를 넘어가는 값도 잘 변환한다.

[Script 2.2.4]

```
#@title기본 제목 텍스트
cor = df[['6~7_r', '6~7_t', '8~9_r', '8~9_t',
          '10~11_r', '10~11_t', '12~13_r', '12~13_t', '14~15_r', '14~15_t',
          '16~17_r', '16~17_t', '18~19_r', '18~19_t', '20~21_r', '20~21_t',
          '22~23_r', '22~23_t', '0~1_r', '0~1_t']].corr()
sns.set(style="white")
mask=np.zeros_like(cor,dtype=np.bool)
mask[np.triu_indices_from(mask)]=True
f,ax=plt.subplots(figsize=(20,15))
cmap=sns.diverging_palette(200,10,as_cmap=True)
sns.heatmap(cor,mask=mask,cmap=cmap,center=0,square=True,linewidths=0.5,cbar_kws={"shrink":1},annot=True); #히트맵 생성
plt.xticks(size=20)
plt.yticks(size=20,rotation=0)
plt.title("arrive and leave correlation graph",size=30);
```

column을 묶어서 만든 dataframe을 바탕으로 시간별 승하차 인원끼리의 상관관계를 파악해주었다. 히트맵은 Seaborn의 heatmap을 사용하였고, Matplotlib의 subplot으로 그래프 사이즈를 조절해 주었다

[그림 2.2.3]



이에 대한 실행결과이다. 최대 상관관계수 절대값이 0.03을 넘지 않는다. 승차 인원간 뚜렷한 상관관계가 보이지 않기에 overfitting을 방지하기 위해 target 데이터만 남겨준다.

•
•
•

2.2.3 정류장의 고유값 결정

정류장을 고유적으로 나타낼 수 있는 Feature로는 ARS_ID(정류장 코드)와 정류장명이 있다. 다음은 둘 중 어느 Feature를 정류장의 고유한 Key값으로 사용할지에 대한 고찰이다.

[Script 2.2.5]

```
print(len(df.정류장명.unique()), len(df.ARS_ID.unique()), sep='\n')
len(df.정류장명.unique())== len(df.ARS_ID.unique())
```

[그림 2.2.4]

```
1496
2803
False
```

정류장명과 ARS_ID의 Unique(고유값) 개수를 파악해 보았다. 정류장명은 1496개, ARS_ID의 고유값은 2803개로 서로 다른 고유값을 가지고 있는 것을 확인하였다.

[Script 2.2.6]

```
nunique_df = df.groupby('정류장명')['ARS_ID'].nunique()
print(nunique_df.unique(), end='\n\n\n')
print(nunique_df, end='\n\n\n')
print(nunique_df.value_counts(), end='\n\n\n')
```

[그림 2.2.5]

```
[2 1 4 3]

정류장명
31사단      2
4.19기념관  2
4수원지     2
4수원지위   2
5.18기념공원 2
..
흑석사거리(동) 2
흑석사거리(서) 2
희망가아파트  2
힐스테이트 각화 2
힐스테이트연제아파트 2
Name: ARS_ID, Length: 1495, dtype: int64

2    1293
1     196
3         4
4         2
Name: ARS_ID, dtype: int64
```

원시 데이터셋을 정류장명을 기준으로 Group화 해주었다. 이 때 Aggregate Func.은 nunique(고유값 개수)를 사용해 주었다. Group화 된 DataFrame은 nunique_df 변수에 저장해 주었다.

첫 번째 출력문은 nunique_df의 고유값을 출력한다. 출력값은 [2, 1, 4, 3]으로, 이는 정류장명 하나 당 ARS_ID가 2개 또는 1개 또는 4개 또는 3개 존재함을 의미한다. 두 번째 출력문은 nunique_df의 출력값이며, 마지막 출력문은 nuique_df의 value_count값을 나타낸다.

마지막 출력문에서, 대부분의 정류장은 정류장명마다 2개의 ARS_ID를 가지고 있는 것을 확인할 수 있는데, 이는 하나의 정류장마다 가는 방향, 오는 방향이 있기 때문이다. 고유값의 개수가 3개 이상인 정류장도 있기에 확인해보았다.

[Script 2.2.7]

```
nunique_df[nunique_df>=3]
```

[그림 2.2.6]

```
정류장명
동산마을      4
명도삼거리    3
문화전당역    3
살레시오고입구  4
평촌          3
효령노인복지타운  3
Name: ARS_ID, dtype: int64
```

[Script 2.2.8]

```
df[df.정류장명=='동산마을'].ARS_ID.unique()
```

[그림 2.2.7]



이름이 같은 정류장이 있었던 것으로 확인되었다. 따라서 정류장의 고유값으로는 정류장명이 아닌 ARS_ID를 사용하는 것이 바람직하다.

2.2.4 결측치 확인

[Script 2.2.9]

```
df.isnull().sum()
```

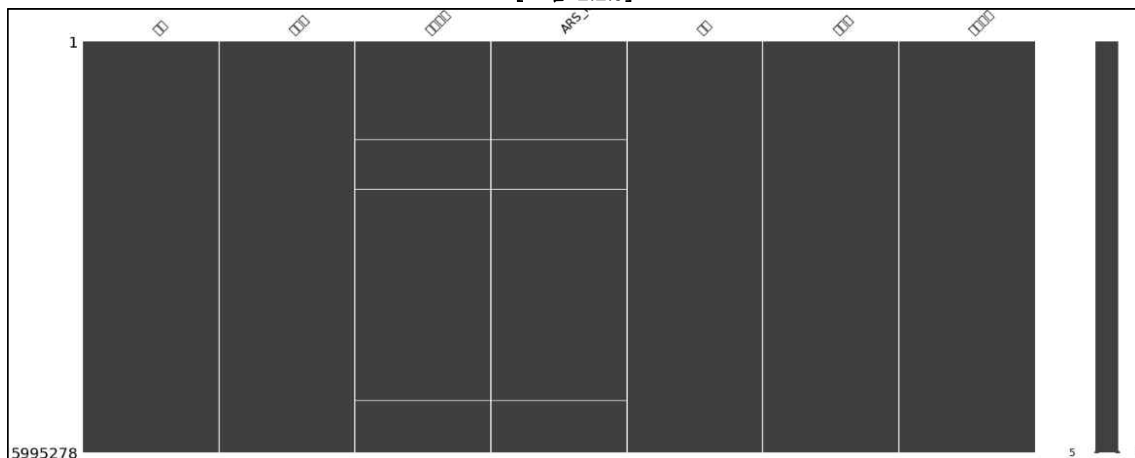
[그림 2.2.8]

```
일자          0
노선명        0
정류장명     39426
ARS_ID       39711
시간          0
승하차        0
거래건수      0
dtype: int64
```

[Script 2.2.10]

```
msno.matrix(df)
```

[그림 2.2.9]



정류장명과 ARS_ID에 4만개 정도의 ARS_ID가 있는 것으로 확인되었다. 결측치 관련 시각화 패키지인 msno를 사용하여 결측치를 시각화 해 본 결과, 전체에 비해 매우 적은 비율(약 0.5%)임을 확인할 수 있었다. 3번째 열이 정류장명이고 4번째 열이 ARS_ID이며, 중간에 세로로 얇게 뻗은 흰색 선이 결측치이다. 따라서 해당 결측치는 Drop 해준다.

또한 정류장명과 ARS_ID간의 결측치의 수가 다르게 나타나고 있었다. 아래는 해당 내용에 대한 EDA 과정이다.

-
-

[Script 2.2.11]

```
df[df.정류장명.notnull() & df.ARS_ID.isnull()]
```

[그림 2.2.10]

	일자	노선명	정류장명	ARS_ID	시간	승하차	거래건수
23430	2022-03-14	마을788	시화문화마을문화관	NaN	7	승차	5
23431	2022-03-14	마을788	시화문화마을문화관	NaN	8	승차	1
23432	2022-03-14	마을788	시화문화마을문화관	NaN	9	승차	1
23433	2022-03-14	마을788	시화문화마을문화관	NaN	10	환승	1
23434	2022-03-14	마을788	시화문화마을문화관	NaN	11	승차	1

[Script 2.2.12]

```
df[df.정류장명.isnull() & df.ARS_ID.notnull()]
```

[그림 2.2.11]

일자	노선명	정류장명	ARS_ID	시간	승하차	거래건수
----	-----	------	--------	----	-----	------

정류장명은 있는데 ARS_ID가 Null값인 경우가 존재했으며, 그 역은 존재하지 않았다. 아래는 어떤 정류장에서 해당 현상이 일어나는지 확인해 본 결과이다.

[Script 2.2.13]

```
df[df.정류장명.notnull() & df.ARS_ID.isnull()].정류장명.unique()
```

[그림 2.2.12]

```
array(['시화문화마을문화관'], dtype=object)
```

“시화문화마을문화관”이라는 정류장의 ARS_ID가 Null 값을 가지고 있었다. 확인 결과, 해당 정류장(특히 각화중 방향)은 ARS_ID를 가지고 있지 않은 것으로 확인되었다. 따라서 해당 정류장에는 임의의 ARS_ID를 부여해준다.

[그림 2.2.13]

정류소검색			동명검색		
정류소명(ARS ID)		시화문화	검색		
보기	ID	정류소명			
	4674	시화문화마을문화관			
		시화문화마을문화관(→각화중방향)			
선택			노선명		

2.3 Data Cleansing & Pre-Processing

2.3.1 임의의 ARS_ID 값 부여

[Script 2.3.1]

```
null_ARS_ID = df[(df.정류장명=='시화문화마을문화관') & df.ARS_ID.isnull()].ARS_ID  
df.loc>null_ARS_ID.index, 'ARS_ID'] = "46740"
```

각화중 방향 시화문화마을문화관 ARS_ID에는 임의의 코드값 “46740”을 부여했다

2.3.2 Drop NA

[Script 2.3.2]

```
df = df.dropna()  
df.isnull().sum()
```

[그림 2.3.1]

```
일자      0  
노선명     0  
정류장명   0  
ARS_ID    0  
시간       0  
승하차     0  
거래건수   0  
dtype: int64
```

나머지 null 값은 전부 drop 해주었다.

•

•

•

•

2.3.3 target_time 선택

[Script 2.3.3]

```
target_time = [18, 19]
target_name = str(target_time[0]) + '~' + str(target_time[1]) + '_ride'

df = df[ df.시간.apply( lambda x: x in target_time ) ]

df = df[ df.승하차 == '승차' ].drop(['승하차'], axis=1)
df.head()
```

[그림 2.3.2]

	일자	노선명	정류장명	ARS_ID	시간	거래건수
27	2022-03-01	1187	광주종합버스터미널	2001.0	18	1
29	2022-03-01	1187	광주종합버스터미널	2001.0	19	2
49	2022-03-01	1187	광주종합버스터미널	2002.0	18	1
51	2022-03-01	1187	광주종합버스터미널	2002.0	19	1
75	2022-03-01	1187	국립아시아문화전당(구.도청)	1122.0	19	5

target time을 지정하여 그 시간대에 해당하는 승차인원 자료만 추출한다. 원시 데이터는 새벽 5시부터 다음날 새벽1시까지의 모든 승차 데이터를 가지고 있지만, 모델의 정확도를 높이기 위해 target 시간대를 지정하여, 해당 시간대에 해당하는 승차인원을 predict 하도록 모델을 설계한다. 현재 2.3절에 나오는 모든 내용은 차후에 사용자 정의함수 안에 모두 들어갈 내용이며, target_time을 매개변수로 지정하여, 모든 시간대에 대한 예측 함수를 만든다. Script 2.2.16은 이해를 돕기위한 예시이며, target_time에 들어가는 값은 사용자 정의 함수의 매개변수이다.

모델 세분화의 또 다른 이유는 Web에서의 효율 문제 때문이다. model을 dump한 pickle파일을 서버에서 불러와야 하는데, 그 용량이 매우 크다. pickle 파일은 말 그대로 절인 오이처럼 객체를 통째로 담아 놓은 파일이기 때문에 개당 용량이 1GB를 초과한다. 1시간 단위로 예측 모델을 만들면 약 20GB 이상의 메모리가 필요하다는 문제점이 생기게 되며, AWS Lightsail로 해당 기능을 구현하려면 꽤 많은 돈이 필요하다. 따라서 2시간 단위로 예측 모델을 만들어 필요한 메모리의 양을 줄이고, 모델의 정확도를 높여주었다.

•
•
•

2.3.4 target_time 생성

[Script 2.3.4]

```
df = df.groupby(['일자', 'ARS_ID', '노선명']).sum().reset_index().drop(['시간'], axis=1)
df.columns = ['date', 'id', 'route', target_name]
df.id = df.id.astype('str').apply(lambda x: x[:-2])

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

[그림 2.3.3]

(293086, 4)

	date	id	route	18~19_ride
0	2022-03-01	1002	문흥39	2
1	2022-03-01	1002	운림54	1
2	2022-03-01	1003	228(화순전대병원)	1
3	2022-03-01	1003	518	3
4	2022-03-01	1003	매월06	4

필요한 Column만 추출하여주고, 앞으로 사용하기 쉽게 Column name을 바꾸어준다.

2.3.5 Train - Test Split

[Script 2.3.5]

```
x_train, x_test, y_train, y_test = train_test_split(df.iloc[:, :-1],
df[target_name], test_size=0.2, shuffle=True, random_state=34)

train_df = pd.concat([x_train, y_train], axis=1)
test_df = pd.concat([x_test, y_test], axis=1)
```

여기서 Train Test Split을 진행해 주었다. Test 데이터는 Feature Engineering 과정에 전혀 참여하지 않은 완전한 새 데이터여야 후에 Model 평가 시 그 검정력이 의미가 있는 것이기 때문에 Feature Engineering 전에 분할을 해주었다. 이 때, Test 데이터는 전체 데이터 셋의 20%를 사용하였다. 검정력을 높이기 위해 index를 무작위로 섞어주고 동일 결과를 추출하기 위해 random_state 옵션을 주었다. 앞으로는 train_df와 test_df에 대해서 서로 다른 작업을 진행해 줄 것이다.

2.4 Feature Engineering

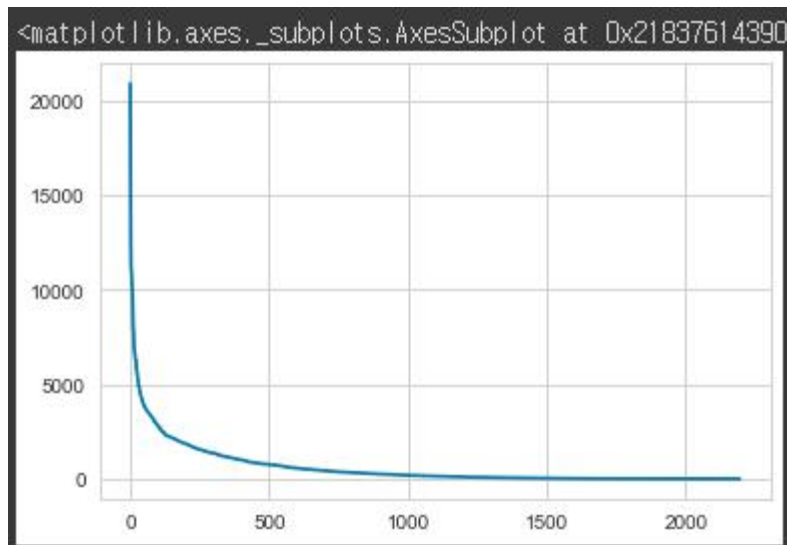
2.4.1 Distance Feature

광주광역시의 주요 정류장과의 거리가 가까울수록 탑승 인원이 많아지는 것을 확인하였음. (충장로 일대나 광천터미널 일대 등등) 따라서 주요 정류장과의 거리를 Train Feature에 추가해 줌.

[Script 2.4.1]

```
passenger_df =  
    train_df.groupby('id').sum().sort_values(by=target_name,  
        ascending=False).reset_index()  
passenger_df[target_name].astype('int').plot()
```

[그림 2.4.1]



train 데이터셋의 정류장을 기준으로 데이터를 그룹화하여 탑승객 수에 대한 plot을 그려보았다. 그 결과, 광주광역시 정류장의 탑승인원은 상위 20개 정도의 정류장에 치우쳐있는 것을 확인할 수 있었다. 따라서 상위 20개의 정류장을 추출하여 지도 위에 나타내보았다.

[Script 2.4.2]

```

path = 'C:/python_venvs/Bus/광주버스정류장 위치정보.csv'

location_df = pd.read_csv(path, encoding='cp949')
location_df = location_df[['단축아이디', '위도', '경도']]

# 광주 최북단 위도 : 35.308578746848085 (6172 - 수복리)
# 광주 최남단 위도 : 34.95417768769266 (6302 - 천덕리)
# 광주 최서단 경도 : 126.64175559345797 / (6234 - 삼서우체국)
# 광주 최동단 경도 : 127.11094258884617 (6284 - 화순전남대병원)
location_df = location_df[ ((location_df.위도>=34.95417768769266) &
 (location_df.위도<=35.308578746848085)) ]
location_df = location_df[ ((location_df.경도>=126.64175559345797) &
 (location_df.경도<=127.11094258884617)) ]

location_df = location_df.apply(lambda x: x.astype('str'))
location_df['단축아이디'] = location_df.단축아이디.apply(lambda x :
x[:-2])

location_df = location_df.drop_duplicates(['단축아이디'],
keep='last')
location_df = location_df.reset_index().drop(['index'], axis=1)

```

[그림 2.4.2]

	단축아이디	위도	경도
0	3111	35.12785833	126.9103444
1	3110	35.12790833	126.9100778
2	4185	35.17166111	126.9353889
3	4172	35.17156944	126.9356333
4	4234	35.18895278	126.9257833

그 전에 정류장의 위도, 경도 데이터를 이용하기 위해서 정류장 위도, 경도 데이터를 불러와 주었다. 광주가 아닌 전국의 데이터이기 때문에, 좌표 검색 사이트를 통해 광주의 동서남북 한계 좌표를 미리 알아둔 뒤, 그 한계를 넘지 않는 좌표만 추출한다. 또한 데이터 타입을 맞추어주고, 중복행을 제거하여 location_df에 저장해준다.

[Script 2.4.3]

```
location_df_20 = pd.merge(passenger_df, location_df, how='left',
left_on='id', right_on='단축아이디')[:20]
location_df_20 = location_df_20.drop('단축아이디', axis=1)

map_osm= folium.Map( (35.16094598877183, 126.88140589783481),
zoom_start=12 )
mc = MarkerCluster()

for loc in location_df_20.itertuples():
    mc.add_child( folium.Marker(location=(float(loc.위도),
float(loc.경도)), popup=loc.id ) )
    map_osm.add_child(mc)

map_osm
```

[그림 2.4.3]



지도 라이브러리는 folium을 사용하였다. 탑승객 기준 상위 20개 정류장의 위치를 지도상에 나타냈더니 크게 9개의 구역으로 나뉘어지는 것을 확인할 수 있었다. 지도의 기본값은 광주광역시청의 위도, 경도값으로 하였고, location_df의 위도, 경도값을 사용하여 지도 위에 표시해 주었다. 9개의 정류장은 다음과 같다.

5159(송정역)
 5745(수완모아엘가)
 5445(침단2동행정복지센터)
 4475(일곡사거리)
 4406(전남대스포츠센터)
 2002(광천터미널)
 2232(시청)
 1130(문화전당역)
 3236(대성여고)

따라서 해당 주요 정류장과 각각 정류장 간의 거리를 파생변수로 추가해준다.

[Script 2.4.4]

```
def makeLoc(target):
    ldf = location_df_20[location_df_20.id == target]
    res = location_df[['위도', '경도']].apply(
        lambda x:
            geopy.distance.geodesic(
                ( float(x.위도), float(x.경도) ),
                (
                    float(ldf.위도),
                    float(ldf.경도)
                )
            ).km,
        axis=1
    )
    res = pd.concat([location_df['단축아이디'], res], axis=1)
    res.columns = ['id', 'dis_'+target]

    return res
```

사용자 정의 함수 makeLoc()은 위에서 추출한 상위 20개의 정류장 중, 선택된 9개의 주요 정류장에 대해서 거리값을 계산해준다. 이 때, 데이터의 양이 많으므로 연산시, 데이터를 Vector화 시켜줘야한다. 100만개를 넘어서는 데이터에 대해서는 수행 시간에서 수십분 이상의 차이를 보였다. Pandas의 DataFrame에서는 apply()가 해당 기능을 대신한다. geopy.distance의 geodesic을 사용하여 주요 정류장과 나머지 정류장간의 거리를 구해 dis_<ARS_ID> Column을 생성하여 concat한 값을 반환한다.

[Script 2.4.5]

```

top_list =
['5159', '5745', '5445', '4475', '4406', '2002', '2232', '1130', '3236']
for top in tqdm( top_list ):
    train_df = pd.merge( train_df, makeLoc(top), how='left', on='id')
    test_df = pd.merge( test_df, makeLoc(top), how='left', on='id')

train_df = train_df.dropna()
test_df = test_df.dropna()

print(train_df.shape, test_df.shape, sep='\n')
train_df.head()

```

[그림 2.4.4]

	date	id	route	18-19_ride	dis_5159	dis_5745	dis_5445	dis_4475	dis_4406	dis_2002	dis_2232	dis_1130	dis_3236
0	2022-04-18	5271	진곡196	9	2.691604	3.747122	7.003298	9.612062	9.700836	6.567251	4.254649	10.209237	9.434778
1	2022-03-09	2230	순환01(운천저수지)	6	5.228835	4.371078	6.653847	7.191297	6.476364	3.188848	0.907714	6.743823	6.340535
2	2022-04-27	3213	진월79	8	9.872854	9.522300	10.621856	7.958063	4.929508	3.741650	5.121813	2.348028	1.472844
3	2022-04-24	5173	송정29	1	8.022307	2.389540	2.502536	4.597789	6.094624	4.517247	3.844495	8.197890	9.442071
4	2022-04-16	5431	첨단92	1	9.265645	2.888153	0.482613	5.124836	7.491567	6.463589	5.860374	9.964572	11.446696

train과 test 데이터에 대해서 모든 연산을 수행하면 다음과 같은 데이터프레임이 생성된다. 단위는 km이다.

-
-
-
-
-
-

2.4.2 Population Feature

[Script 2.4.6]

```
address_df = location_df.apply(lambda x:
gmaps.reverse_geocode((x.위도, x.경도), language='ko'), axis=1 )

def make(data):
    for d in data:
        if (d['types'][-1] == 'sublocality_level_2'):
            break

    data_detail = d['address_components']
    for dt in data_detail:
        if (dt['types'][-1] == 'sublocality_level_2'):
            break

    return dt['long_name']

address_df['동명'] = address_df.apply(lambda x: make(x))
address_df['동명'].to_csv('/content/drive/MyDrive/BusProject/dong_3_
1~4_30_18ride.csv')
```

사용자 정의 함수 make()는 google의 geocodingAPI를 사용하여 경도, 위도에 해당하는 주소값을 반환한다. 얻고자 하는 값은 '동명'이기 때문에 해당 데이터를 추출해주는 작업을 한다. 반환값이 xml이나 json이 아니라 list 값이어서 제어하기 쉽다. '동명'은 다음과 같은 형식을 가지고 있다.

result의 types의 마지막 값이 sublocality_level_2의 long_name
또는
result의 address_components의 types의 마지막 값이 sublocality_level_2의 long_name

make()는 이러한 조건을 만족하는 String을 반환한다. 광주광역시 모든 정류장의 경, 위도 데이터에 대하여 make()를 적용한다. 해당 작업은 시간이 오래 걸리는 관계로 따로 csv 파일로 저장해둔다.

•

•

[Script 2.4.7]

```

dong_name =
pd.read_csv('C:/python_venvs/Bus/dong_3_1_4_30_18ride.csv')['0']
location_df = pd.concat( [location_df, dong_name], axis=1 )
location_df.columns = ['id', '위도', '경도', 'dong_name']

train_df = pd.merge(train_df, location_df, how='left', on='id')
test_df = pd.merge(test_df, location_df, how='left', on='id')

train_df.head()

```

[그림 2.4.5]

_4406	dis_2002	dis_2232	dis_1130	dis_3236	위도	경도	dong_name
00836	6.567251	4.254649	10.209237	9.434778	35.15881944	126.8078361	우산동
76364	3.188848	0.907714	6.743823	6.340535	35.15518056	126.8456222	치평동
29508	3.741650	5.121813	2.348028	1.472844	35.131783	126.900075	주월1동
94624	4.517247	3.844495	8.197890	9.442071	35.19322778	126.8493361	신창동
91567	6.463589	5.860374	9.964572	11.446696	35.21075556	126.8426667	월계동

결과값을 location_df와 concat 해주고, location_df는 다시 train, test 데이터와 concat한다. concat 결과값은 그림 2.2.19와 같다.

이제 dong_name을 기준으로 해당 소재지의 인구 데이터를 추가해 준다. 하지만, Google의 Geocoding API는 해당 경위도의 법정동을 추출한다. 그러나 KOSIS나 광주광역시에서 제공하는 인구통계량은 모두 행정동을 기준으로 데이터를 집계한다. 그 어디에서도 법정동을 기준으로 한 인구 통계량 데이터는 찾을 수 없었다. 따라서 행정동과 법정동의 맵핑데이터를 이용하여 법정동 데이터를 행정동 데이터로 변환하여준다. 해당 과정은 Excel에서 진행해주었다.

[그림 2.4.6]

상무동(尙武洞)은 광주광역시 서구의 동명이다. 법정동으로는 쌍촌동(雙村洞)이다. 행정동으로 상무1동·상무2동으로 나뉜다. 행정동명은 상무대에서 따왔다.

[Script 2.4.8]

```
pop_df = pd.read_csv('C:/python_venvs/Bus/광주_인구.csv',
encoding='cp949')
pop_df.columns = ['dong_name', 'dist', 'population']
pop_df['population'] = pop_df['population'].apply(lambda x:
x.replace(',','')).astype('int')
pop_df.dong_name = pop_df.dong_name.apply(lambda x: x.replace('
',''))
pop_df.head()
```

[그림 2.4.7]

	dong_name	dist	populat ion
0	충장동	동구	4704
1	동명동	동구	4059
2	계림1동	동구	11094
3	계림2동	동구	9842
4	산수1동	동구	8661

법정동/행정동 변환이 끝난 데이터를 바탕으로 인구 데이터를 불러와준다. 이 때, population에 있는 쉼표(,)는 Python이 문자열로 인식하므로 제거해준다. 불러온 인구 데이터를 바탕으로 원래 데이터 프레임과 merge 해준다. dong_name을 join 기준으로 하여 left join을 해준다. 결과는 다음과 같다.

[그림 2.4.8]

dis_2002	dis_2232	dis_1130	dis_3236	위도	경도	dong_name	population
6.567251	4.254649	10.209237	9.434778	35.15881944	126.8078361	우산동	10363
6.567251	4.254649	10.209237	9.434778	35.15881944	126.8078361	우산동	29523
3.188848	0.907714	6.743823	6.340535	35.15518056	126.8456222	치평동	30457
3.741650	5.121813	2.348028	1.472844	35.131783	126.900075	주월1동	22039
4.517247	3.844495	8.197890	9.442071	35.19322778	126.8493361	신창동	35158

2.4.3 주요 Variable 요약통계량

기본적인 아이디어는 다음과 같다.

현재 데이터에서 데이터를 유의미한 그룹으로 분할하는 변수로는 id(ARS_ID), route(노선명) weekday(요일)이다. 따라서 이 세 가지의 변수를 주요 변수로 설정하고 아래와 같이 주요 변수에 대한 모든 조합을 생성한다.

1. id
2. route
3. weekday
4. id_route
5. id_weekday
6. route_weekday
7. id_route_weekday

이렇게 생성된 7가지의 Feature에 대한 각각의 요약통계량(Mean, Sum)을 구하여 새로운 Feature로 넣어준다. 따라서 총 14개의 Column이 생성된다. 아래는 이해를 돕기 위한 간단한 예시이다.

ex) id=4407 route="진월07" weekday=1(월요일) 일 때,

route_weekday_mean = 월요일에 진월07번 버스를 타는 평균 탑승객 수

id_route_weekday_sum = 월요일에 4407 정류장에서 진월07번 버스를 탄 3~4월 총 탑승객 수

해당 아이디어를 코드로 구현해보면 다음과 같다.

.

.

.

.

[Script 2.4.9]

```
# weekday
train_df['weekday'] = train_df.date.dt.weekday
test_df['weekday'] = test_df.date.dt.weekday

# route_id
train_df['route_id'] = train_df['route'] + '_' + train_df['id']
test_df['route_id'] = test_df['route'] + '_' + test_df['id']

# route_weekday
train_df['route_weekday'] = train_df['route'] + '_' +
train_df['weekday'].astype(str)
test_df['route_weekday'] = test_df['route'] + '_' +
test_df['weekday'].astype(str)

# id_weekday
train_df['id_weekday'] = train_df['id'] + '_' +
train_df['weekday'].astype(str)
test_df['id_weekday'] = test_df['id'] + '_' +
test_df['weekday'].astype(str)

# route_id_weekday
train_df['route_id_weekday'] = train_df['route_id'] + '_' +
train_df['weekday'].astype(str)
test_df['route_id_weekday'] = test_df['route_id'] + '_' +
test_df['weekday'].astype(str)
```

7개의 조합을 만들어주는 과정이다. Python에서 int형을 자동으로 String형으로 변환해주지 않기 때문에, 문자열 결합을 위해서는 Datetime 타입의 weekday를 String 타입으로 변환해줘야 한다. 결과는 다음과 같다.

•

•

•

•

[그림 2.4.9]

id	route	weekday	route_id	route_weekday	id_weekday	route_id_weekday
5271	진곡 196	0	진곡 196_5271	진곡196_0	5271_0	진곡196_5271_0
5271	진곡 196	0	진곡 196_5271	진곡196_0	5271_0	진곡196_5271_0
2230	순환 01(운 천저 수지)	2	순환01(운 천저수 지)_2230	순환01(운천저수 지)_2	2230_2	순환01(운천저수 지)_2230_2
3213	진월 79	2	진월 79_3213	진월79_2	3213_2	진월79_3213_2
5173	송정 29	6	송정 29_5173	송정29_6	5173_6	송정29_5173_6

이제 해당 Feature를 groupby의 기준값으로 잡고 mean값과 sum값을 구하여 각각 train과 test 데이터에 merge 해준다. 이 때, test 데이터셋의 순결성을 위해서 통계량은 오직 train 데이터에 대한 통계량을 구해준다. 이것은 단순히 정확한 Model의 성능을 파악하기 위한 작업이고, 추후에 최종적으로 Model을 학습할 때는 test데이터의 통계량까지 전부 학습해준다.

통계량을 생성하기 위해 사용자 정의함수 id_statistic()을 생성해 준다. 매개변수로 ID(7개의 과생변수) col1(이름지정 : Mean에 대한 이름을 지정할 것이다.) col2(Sum에 대한 이름 지정)을 받는다. 코드는 다음과 같다.

[Script 2.4.10]

```
def id_statistic(ID, col1, col2) :

    # train mean, sum
    train_df_mean = train_df.groupby([ID])[target_name].agg([(col1,
'mean')]).reset_index()
    train_df_sum = train_df.groupby([ID])[target_name].agg([(col2,
'sum')]).reset_index()
    train_mean_sum = pd.merge(train_df_mean, train_df_sum, on=ID)

    # merge
    train_res = pd.merge(train_df, train_mean_sum, how='left', on=ID)
    test_res = pd.merge(test_df, train_mean_sum, how='left', on=ID)

    return train_res, test_res
```

test 데이터는 통계량에 전혀 개입을 하고 있지 않은 것을 확인할 수 있다. 이제 id_statistic을 사용하여 7개 주요 변수에 대한 통계량을 생성해준다.

[Script 2.4.11]

```

train_df, test_df = id_statistic('route_id', target_name+'_ri_mean',
target_name+'_ri_sum')
train_df, test_df = id_statistic('route', target_name+'_r_mean',
target_name+'_r_sum')
train_df, test_df = id_statistic('id', target_name+'_i_mean',
target_name+'_i_sum')
train_df, test_df = id_statistic('weekday', target_name+'_w_mean',
target_name+'_w_sum')
train_df, test_df = id_statistic('route_weekday',
target_name+'_rw_mean', target_name+'_rw_sum')
train_df, test_df = id_statistic('id_weekday',
target_name+'_iw_mean', target_name+'_iw_sum')
train_df, test_df = id_statistic('route_id_weekday',
target_name+'_riw_mean', target_name+'_riw_sum')

print(train_df.shape, test_df.shape, sep='\n\n')
train_df.head()

```

[그림 2.4.10]

18-19_ride_w_sum	18-19_ride_rw_mean	18-19_ride_rw_sum	18-19_ride_iw_mean	18-19_ride_iw_sum	18-19_ride_riw_mean	18-19_ride_riw_sum
203764	3.913924	1546	12.142857	680	6.500000	78
203764	3.913924	1546	12.142857	680	6.500000	78
221583	9.221429	2582	11.382353	387	32.285714	226
221583	3.480000	696	4.288136	253	4.200000	21
99598	5.045889	2639	1.583333	19	1.250000	5

Column의 수가 많아 일부 행은 생략하였다. 위와 같이 통계량에 대한 컬럼 14개가 생성되는 것을 확인할 수 있다.

•

•

•

2.4.4 혼잡도 가중치

다음으로는 주요 변수(id, route, weekday)에 대한 혼잡도 가중치 Column을 만들어주었다. 각각 변수마다 스케일이 다르기 때문에 스케일을 제거하고, 순수 혼잡도만 파악하고자 Min-Max Scaling을 진행해주었다.

[Script 2.4.12]

```
scaler = MinMaxScaler()

def congestion(ID) :
    res = train_df.groupby([ID])[target_name].agg(['passenger',
    'sum'])
    res = res.sort_values(by='passenger',
ascending=False).reset_index()

    scaler.fit( res[['passenger']] )
    minmax = scaler.transform( res[['passenger']])
    res = pd.concat([res[ID],pd.DataFrame(minmax,
columns=[ID+'_congestion'])], axis=1)

    train_r = pd.merge(train_df, res, how='left', on=ID)
    test_r = pd.merge(test_df, res, how='left', on=ID)

    return train_r, test_r
```

스케일러의 적합값은 가공이 가능한 형태가 아니기 때문에 transform으로 Series 타입으로 변환해준다. 이를 다시 train과 test 데이터에 ID(주요 변수)를 기준으로 left join 시켜준다.

[Script 2.4.13]

```
train_df, test_df = congestion('route')
train_df, test_df = congestion('id')
train_df, test_df = congestion('weekday')
train_df.head()
```

[그림 2.4.11]

route_congestion	id_congestion	weekday_congestion
0.177531	0.221440	0.725597
0.177531	0.221440	0.725597

0에서 1사이의 값으로 잘 적합된 모습을 확인할 수 있다.

2.4.5 공휴일 및 주말 여부

EDA 과정에서 주말에 이용객의 수가 감소하는 것을 확인하였기에 공휴일 및 주말 여부에 대한 Column을 생성해주었다.

[Script 2.4.14]

```
path = 'C:/python_venvs/Bus/holiday.csv'
month = (3 or 4)

holiday_df = pd.read_csv(path, encoding='cp949')
holiday_df = holiday_df[(holiday_df.년>=2022) &
(holiday_df.월==month)]
holiday_df
```

[그림 2.4.12]

순서	년	월	일	설명	
75	76	2022	3	1	삼일절
76	77	2022	3	9	대통령선거일

[Script 2.4.15]

```
train_df['weekend'] = train_df.apply(lambda x: 1 if x.weekday>4 else 0, axis=1)
test_df['weekend'] = test_df.apply(lambda x: 1 if x.weekday>4 else 0, axis=1)

print(train_df.shape, test_df.shape, sep='\n\n')
train_df.head()
```

[그림 2.4.13]

day_congestion	weekend
0.725597	0
0.725597	0

해당 데이터는 미리 다운로드 받아 놓았기 때문에 불러와서 기존 데이터와 합쳐주었다. 사용 데이터에 대한 자세한 출처는 부록 참조.

2.4.6 기상 데이터

기상 데이터도 마찬가지로 미리 받아놓은 자료를 기존 데이터와 병합해주었다. 필요 없는 column이 있어서 필요한 column만 추출해주었고, datetime을 기준으로 기존 데이터와 join 해 주기 위해 ['일시'] column의 format을 수정해줬다.

[그림 2.4.14]

기온	강수량	풍속	습도
19.00	0.0	1.75	27.0
19.00	0.0	1.75	27.0
14.50	0.0	1.40	33.0
19.65	0.0	1.40	49.0
21.90	0.0	1.35	66.5

2.4.7 요일 데이터 One-Hot Encoding

[Script 2.4.16]

```
train_df = pd.get_dummies(train_df, columns=['weekday'])
test_df = pd.get_dummies(test_df, columns=['weekday'])

print(train_df.shape, test_df.shape, sep='\n\n')
train_df.head()
```

weekday Feature에 대해서는 One-Hot Encoding(더미변수화)을 진행해 주었다. 3개 이상의 범주를 가지는 범주형 자료에 대해서 회귀 모델에 맞게 변환시키는 기법 중 하나이다. 해당 피처가 가지는 고유값을 모두 column으로 올리고 값의 유무를 0과 1이라는 2개의 범주로 표현하는 기법이다. 해당 기능은 pandas의 get_dummies()가 제공하고 있다.

.

2.4.8 주요 Variable Label Encoding

해당 Feature는 원래 Train Feature에 추가하고자 하였으나 최종적으로 추가하지 않기로 결정하여 간략히 소개만 하고 넘어가도록 하겠다. route나 id같은 column은 target(탑승객 수)에 대해 영향력이 높은 Feature이지만, 문자열이기 때문에 회귀 모델에 적용시킬 수 없다. 따라서 이러한 문자열 Feature를 ML 모델에 적용시키기 위한 기법이 Label Encoding 값이다. 쉽게 표현하자면, 문자열 자료를 범주형 자료로 표현해주는 기법이다. 모든 문자의 고유값을 줄 세운 뒤, 하나씩 고유 번호를 부여해주는 것이다.

하지만 회귀 모델에서는 모델이 해당 수치를 가중치로 인식할 우려가 있어 사용하지 않고, 주로 분류 모델에서 많이 사용한다. 실제로 Label Encoding Feature가 추가되고 성능이 소폭 하락하는 것을 확인할 수 있었다. 따라서 회귀 모델에서는 One-Hot Encoding을 해주어야 하는데, 해당 Feature의 범주가 1만개 이상을 초과하고, 성능에 있어서 그리 중요하지 않은 피처이기 때문에 해당 피처는 추가해주지 않았다.

•

•

•

•

•

•

•

•

•

2.5 Train & Save Model

2.5.1 Train & Test Data

[Script 2.5.1]

```
y_train = train_df[target_name]; y_test = test_df[target_name]

input_var = ['dis_5159', 'dis_5745', 'dis_5445',
             'dis_4475', 'dis_4406', 'dis_2002', 'dis_2232', 'dis_1130',
             'dis_3236',
             'population', '18~19_ride_ri_mean',
             '18~19_ride_ri_sum', '18~19_ride_r_mean', '18~19_ride_r_sum',
             '18~19_ride_i_mean', '18~19_ride_i_sum', '18~19_ride_w_mean',
             '18~19_ride_w_sum', '18~19_ride_rw_mean', '18~19_ride_rw_sum',
             '18~19_ride_iw_mean', '18~19_ride_iw_sum',
             '18~19_ride_riw_mean',
             '18~19_ride_riw_sum', 'route_congestion', 'id_congestion',
             'weekday_congestion', 'weekend', '기온', '강수량', '풍속',
             '습도',
             'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3',
             'weekday_4',
             'weekday_5', 'weekday_6', 'route_encode', 'id_encode',
             'route_id_weekday_encode', 'route_id_encode']

x_train = train_df[input_var]; x_test = test_df[input_var];
```

[그림 2.5.1]

(241115, 43) (60276, 43)

총 43개의 Train Feature로 이루어져있고, Train Data 24만개, Test Data 6만개로 학습을 진행한다. 합쳐주었던 Target 데이터를 다시 분리하여 x와 y 데이터로 나누어준다. Train과 Test 데이터에 대해서 동일한 작업을 수행한다.

•

•

•

2.5.2 Set Up Data

여기서부터는
Colab과 Pycaret간의 버전 문제로 Jupyter Notebook을 사용하여 개발 진행

[Script 2.5.2]

```
exp_clf = setup(data = pd.concat([x_train,y_train], axis=1), target  
= target_name, session_id=123)
```

[그림 2.5.2]

	Description	Value
0	session_id	123
1	Target	18~19_ride
2	Original Data	(241115, 44)
3	Missing Values	False
4	Numeric Features	41
5	Categorical Features	2
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(168780, 37)
10	Transformed Test Set	(72335, 37)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold
14	Fold Number	10

학습 전, Pycaret의 다양한 Setting값을 설정해주는 단계이다. 이미 전처리과정에서 모든 작업을 끝마친 관계로 따로 설정값을 바꾸진 않았다. Train DataSet과 target을 지정해주고, 동일 결과를 얻기 위해 seed값을 지정해주었다. 더 정확한 검증을 위해 모델 검증은 k-fold 교차검증 방식을 사용하였고 fold값은 기본값인 10으로 지정해주었다.

•

•

•

2.5.3 Find Best Model (By R2)

[Script 2.5.3]

```
best = compare_models(sort='r2')
```

[그림 2.5.3]

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	1.6817	7.8966	2.8091	0.8712	0.3475	0.4591	0.8580
gbr	Gradient Boosting Regressor	1.7059	8.2371	2.8687	0.8657	0.3530	0.4708	18.0530
br	Bayesian Ridge	1.7554	8.7519	2.9573	0.8573	0.3638	0.4939	0.5610
lr	Linear Regression	1.7555	8.7525	2.9574	0.8573	0.3639	0.4940	1.1600
ridge	Ridge Regression	1.7555	8.7525	2.9574	0.8573	0.3639	0.4940	0.1000
omp	Orthogonal Matching Pursuit	1.7537	8.7546	2.9577	0.8573	0.3632	0.4925	0.1230
lar	Least Angle Regression	1.7591	8.7808	2.9621	0.8568	0.3649	0.4955	0.1390
en	Elastic Net	1.7623	8.9206	2.9856	0.8546	0.3626	0.4924	1.7730
lasso	Lasso Regression	1.7619	8.9709	2.9940	0.8537	0.3616	0.4887	0.8830
rf	Random Forest Regressor	1.8008	9.4207	3.0683	0.8464	0.3717	0.4894	60.6140
et	Extra Trees Regressor	1.8408	10.0983	3.1766	0.8354	0.3807	0.4955	52.1830
knn	K Neighbors Regressor	2.1357	12.8309	3.5806	0.7909	0.4365	0.6034	2.6480
dt	Decision Tree Regressor	2.4030	18.0453	4.2468	0.7054	0.5024	0.5924	1.2270
ada	AdaBoost Regressor	4.5447	28.9637	5.3067	0.5272	0.8746	2.3414	12.5010
par	Passive Aggressive Regressor	4.8999	74.3239	6.4673	-0.2261	0.7017	1.9639	0.8940
llar	Lasso Least Angle Regression	4.6730	61.3696	7.8309	-0.0001	0.8453	1.8061	0.1400
dummy	Dummy Regressor	4.6730	61.3696	7.8309	-0.0001	0.8453	1.8061	0.0450

Pycaret의 `compare_models()`를 사용하면, 결정 계수값을 기준으로 Python에서 제공하는 지도학습 회귀 라이브러리들의 성능을 비교해 볼 수 있다. 여기서 상위 3개의 모델에 대해서 Blending 및 Ensemble을 진행해 주었다. 상위 3개의 모델은 LGBM, GBR, BR이며 결정계수 값은 각각 0.8712, 0.8657, 0.8573로 괜찮은 성능을 보여주고 있었다.

-
-
-
-

2.5.4 Hyper Parameter Tuning

[Script 2.5.4]

```
lgbm = create_model('lightgbm')
tuned_lgbm = tune_model(lgbm, n_iter = 1000, optimize = 'R2',
choose_better = True)

gbr = create_model('gbr')
tuned_gbr = tune_model(gbr, n_iter = 100, optimize = 'R2',
choose_better = True)

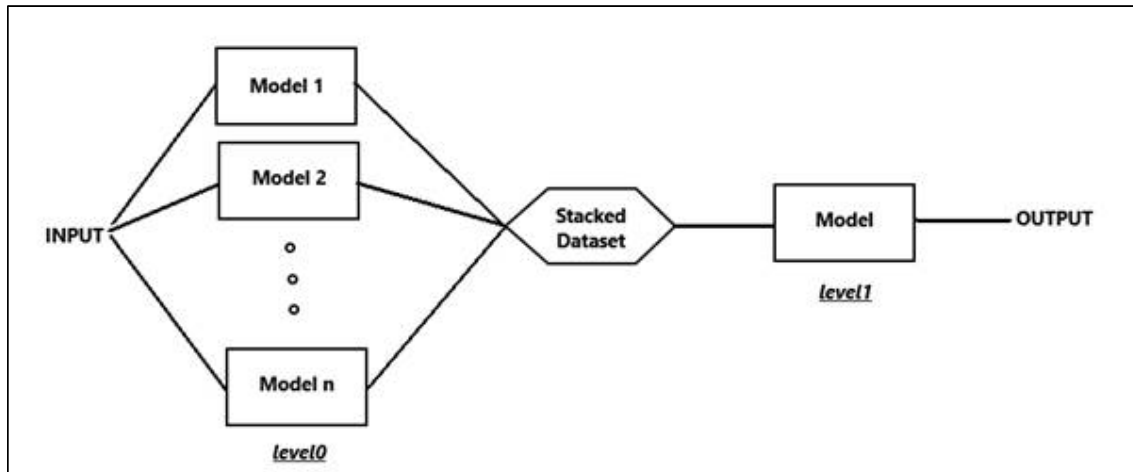
br = create_model('br')
tuned_br = tune_model(br, n_iter = 100, optimize = 'R2',
choose_better = True)
```

위에서 찾아낸 상위 3개의 모델에 대하여 Hyper Parameter Tuning을 진행해주었다. Search는 시간 대비 효율이 좋은 Random Search 방식을 채택하였다. LGBM 같은 경우는 이를 그대로 모델이 가벼워서 학습 시간이 짧아 n_iter(반복횟수) 값을 충분히 늘려줄 수 있었다. 실제로 1000번 Search 하는 데 10분 남짓 걸렸다. 하지만 GBR과 BR의 경우 학습 시간이 길어져서 Search 횟수를 100회로 제한하였다. 또한, Search 후 모델 성능이 오히려 안좋아질 수도 있기 때문에, 성능이 향상되었을 때만, Select 해주도록 하였다.

-
-
-
-
-
-
-

2.5.5 Model Blending

[그림 2.5.4]



Model Blending은 앙상블 테크닉의 일종으로서, 그림 2.5.4는 Blending에 대한 간단한 FlowChart이다. 앞서 HyperParameter 튜닝이 끝난 3개의 모델을 Blending 하여 새로운 모델을 생성한다.

[Script 2.5.5]

```
blender_3 = blend_models([tuned_lgbm, tuned_gbr, tuned_br])
```

•

•

•

•

•

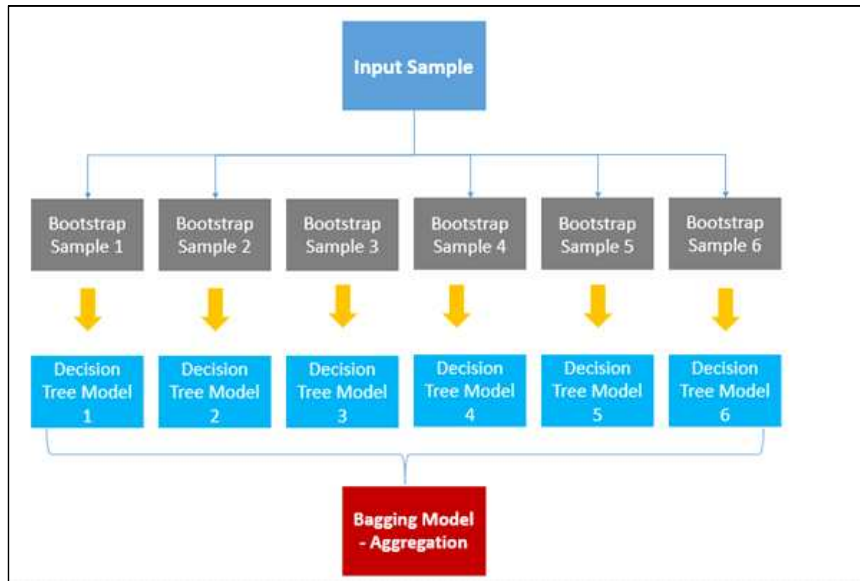
2.5.6 Model Ensemble

마지막으로, 배깅을 통해 모델의 결정력을 높여주었다. 배깅이란, Bootstrap Aggregation의 약자로서, 샘플을 여러 번 뽑아(Bootstrap) 각 모델을 학습시켜 결과물을 집계(Aggregation)하는 방법이다. 그림 [2.5.5]는 배깅에 대한 간단한 flow chart이다.

이 때, 일반적으로 배깅보다 더 좋은 성능을 내는 부스팅을 사용하지 않은 이유에 대한 의문을 제기할 수도 있을 것이다. 그 이유는 여러 가지가 있는데, 우선은 시간문제이다. 시간대별로 Predict값을 얻기 위해서는 총 10개 가까이 모델 생성해야하기 때문에 직렬 학습을 하는 부스팅이 병렬 학습을 하는 배깅보다 학습 속도가 현저히 느리다.

두 번째 문제는 OverFitting의 문제이다. 특히 지금 사용하고 있는 LGBM은 알고리즘의 특성 상 과대적합에 빠지기 쉬워서 과대적합을 더욱 더 조심해야 한다.

[그림 2.5.5]



[Script 2.5.6]

```
ensemble_blender_3 = ensemble_model(blender_3, fold = 10,
n_estimators = 100, choose_better = True)

final_model = finalize_model(ensemble_lgbm_tuned)
```

100개의 분류기를 fold 값을 10번으로 하여 총 1,000번 학습시켰다. 이 때도, 성능 저하의 우려가 있기 때문에 각 학습시에, 성능이 좋은 모델을 고르도록 하였다. 또한 앙상블까지 모두 끝난 모델을 finalize 해주었다. Pycaret 자체적으로 hold-out 교차검증을 하는데, 이 때, 전체 데이터의 30%는 학습하는 데 사용되지 않는다. 따라서 나머지 30%의 데이터도 모두 학습하는 데 사용함으로써 최종 모델이 완성된다.

2.5.7 Predict & Evaluate Model

[Script 2.5.7]

```
prediction = predict_model(final_model, data = x_test)

y_pred = prediction['Label']

MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)
RMSE = np.sqrt(MSE)
R2 = r2_score(y_test, y_pred)

print(MAE, MSE, RMSE, R2, sep='\n')

fig = plt.figure(figsize=(30,10))
fig.add_subplot()

plt.plot(y_pred[:200].tolist(), label='Prediction', color='red')
plt.plot(y_test[:200].tolist(), label='Test', color='green')
plt.legend(fontsize = 30)
```

최종 모델의 적합값을 구하고 그에 대한 MAE, MSE, RMSE, R2 값을 출력한다. 또한 실제 값과 예측값의 plot을 그려서 실제값의 추세를 적합값이 잘 따라가고 있는지 확인한다. 자세한 결과는 2.5.8에서 보여주고 있다.

•
•
•
•
•

2.5.8 Dump & Save Model

이제 전체적인 모델 학습의 가이드라인이 형성되었으므로, 2.3절의 내용부터 직전의 내용까지를 전부 사용자 정의 함수로 정의한다. 다음은 해당 코드의 일부이다.

[Script 2.5.8]

```
def makeModel(first_time, second_time):
    df = df_.copy()

    #
    target_time = [first_time, second_time]
    target_name = str(target_time[0]) + '~' + str(target_time[1])
    + '_ride'
    df = df[ df.시간.apply( lambda x: x in target_time ) ]
    df = df[ df.승하차 == '승차' ].drop(['승하차'], axis=1)

    #
    df =
df.groupby(['일자', 'ARS_ID', '노선명']).sum().reset_index().drop(['시간
'], axis=1)
    df.columns = ['date', 'id', 'route', target_name]
    df.id = df.id.astype('str').apply(lambda x: x[:-2])

    #
    x_train, x_test, y_train, y_test =
train_test_split(df.iloc[:, :-1], df[target_name], test_size=0.2,
shuffle=True, random_state=34)
    train_df = pd.concat([x_train, y_train], axis=1)
    test_df = pd.concat([x_test, y_test], axis=1)
```

(중간 생략)

```
final_model = finalize_model(ensemble_lgbm_tuned)
prediction = predict_model(final_model, data = x_test)

return final_model
```

target time의 시작 시간과 끝 시간을 매개변수로 입력하면 해당 시간에 대한 승차인원 예측 모델을 반환한다.

[Script 2.5.9]

```
for i in tqdm(range(6, 23, 2)):
    save_model(makeModel(i, i+1), 'C:/Users/user/Desktop/'+ str(i) +
'_ ' + str(i+1) + '_model_pycaret')
```

6시부터 22시까지 2시간씩을 매개변수로 넣어주어 return된 모델을 pickle파일로 저장한다. 이 때, 출력되는 모델의 성능은 다음과 같다.

[그림 2.5.6] : 06~08시, 08~10시 Model]

	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	1.4093	6.9832	2.6426	0.8611	0.3159	0.3983
1	1.3776	5.9856	2.4466	0.8854	0.3121	0.3897
2	1.4393	7.8765	2.8065	0.8515	0.3207	0.4064
3	1.3920	6.6757	2.5837	0.8772	0.3120	0.3903
4	1.3988	6.7266	2.5936	0.8807	0.3122	0.3857
5	1.3621	6.0460	2.4589	0.8785	0.3136	0.3951
6	1.3967	6.7223	2.5927	0.8654	0.3115	0.3883
7	1.4385	7.8792	2.8070	0.8579	0.3169	0.3961
8	1.4030	6.4213	2.5340	0.8788	0.3126	0.3906
9	1.4541	7.7348	2.7811	0.8721	0.3144	0.3928
Mean	1.4071	6.9051	2.6247	0.8709	0.3142	0.3933
Std	0.0275	0.6725	0.1274	0.0107	0.0028	0.0057
1.7828693581630144 8.693683151435367 2.948505240191268 0.18231250156700457 0.4269806805547583 0.8374312179484186						

	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	1.6874	8.3267	2.8856	0.8776	0.3379	0.4296
1	1.6782	8.4293	2.9033	0.8754	0.3362	0.4261
2	1.6585	8.8787	2.9797	0.8650	0.3348	0.4241
3	1.6965	8.8594	2.9765	0.8514	0.3415	0.4354
4	1.7042	8.4296	2.9034	0.8730	0.3445	0.4438
5	1.6515	8.5472	2.9236	0.8537	0.3350	0.4304
6	1.6837	8.5144	2.9179	0.8605	0.3374	0.4315
7	1.6774	8.1116	2.8481	0.8696	0.3406	0.4361
8	1.6837	8.4901	2.9138	0.8664	0.3364	0.4291
9	1.6685	7.9243	2.8150	0.8673	0.3357	0.4259
Mean	1.6789	8.4511	2.9067	0.8660	0.3380	0.4312
Std	0.0153	0.2781	0.0479	0.0083	0.0030	0.0056
2.0821565509381394 11.422957134202507 3.379786551574301 0.19818994314925728 0.44518529080514024 0.8224089751987382						

[그림 2.5.7] : 10~12시, 12~14시 Model]

	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	1.4054	5.5723	2.3606	0.8232	0.3362	0.4433
1	1.4021	5.7419	2.3962	0.8298	0.3311	0.4364
2	1.4368	6.2685	2.5037	0.8315	0.3346	0.4424
3	1.4062	5.6442	2.3757	0.8396	0.3340	0.4402
4	1.4045	5.5694	2.3600	0.8219	0.3347	0.4484
5	1.4241	5.7902	2.4063	0.8122	0.3383	0.4509
6	1.4284	6.4997	2.5495	0.7968	0.3402	0.4547
7	1.4169	5.7076	2.3891	0.8226	0.3336	0.4383
8	1.4480	5.8578	2.4203	0.8131	0.3414	0.4544
9	1.4502	5.8195	2.4124	0.8221	0.3410	0.4534
Mean	1.4223	5.8471	2.4174	0.8213	0.3365	0.4462
Std	0.0173	0.2884	0.0588	0.0113	0.0033	0.0066
1.8166061578807184 8.333107516998618 2.886712233146667 0.19988497160661892 0.4470849713495399 0.7466501147790319						

	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	1.5917	6.7660	2.6011	0.8435	0.3515	0.4699
1	1.6351	7.7519	2.7842	0.8413	0.3503	0.4617
2	1.6427	7.8969	2.8101	0.8328	0.3534	0.4718
3	1.6370	7.9851	2.8258	0.8476	0.3521	0.4669
4	1.6491	8.4147	2.9008	0.8208	0.3522	0.4640
5	1.6385	7.8502	2.8018	0.8425	0.3540	0.4717
6	1.6300	8.0260	2.8330	0.8253	0.3492	0.4611
7	1.5976	7.4262	2.7251	0.8424	0.3496	0.4629
8	1.6278	7.6945	2.7739	0.8321	0.3541	0.4704
9	1.6341	8.9593	2.9932	0.8181	0.3496	0.4603
Mean	1.6284	7.8771	2.8049	0.8347	0.3516	0.4661
Std	0.0178	0.5461	0.0976	0.0099	0.0018	0.0044
2.0488026321547887 10.88597635888564 3.299390300629139 0.21484548543904572 0.463514277492124 0.7706705581067177						

[그림 2.5.8] : 14~16시, 16~18시 Model]

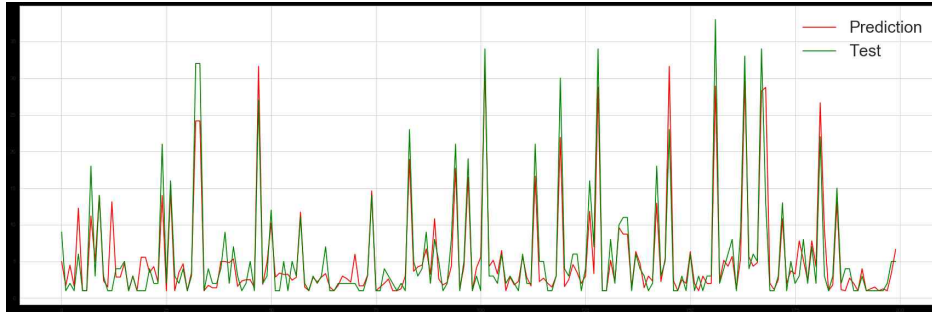
	MAE	MSE	RMSE	R2	RMSLE	MAPE		MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold							Fold						
0	1.7112	9.8531	3.1390	0.8470	0.3579	0.4788	0	2.0141	12.7770	3.5745	0.8787	0.3629	0.4784
1	1.7138	8.7372	2.9559	0.8557	0.3590	0.4795	1	2.0489	14.5997	3.8210	0.8794	0.3602	0.4718
2	1.7285	11.0199	3.3196	0.8429	0.3557	0.4736	2	2.0091	11.4549	3.3845	0.8891	0.3571	0.4630
3	1.7126	9.0099	3.0017	0.8652	0.3566	0.4713	3	2.0339	14.3903	3.7935	0.8625	0.3593	0.4643
4	1.7143	9.0883	3.0147	0.8473	0.3578	0.4770	4	2.0560	14.8641	3.8554	0.8739	0.3579	0.4637
5	1.6793	8.1536	2.8555	0.8514	0.3567	0.4761	5	2.0352	12.9759	3.6022	0.8757	0.3596	0.4711
6	1.7158	9.7578	3.1237	0.8411	0.3577	0.4773	6	2.0058	12.4593	3.5298	0.8890	0.3574	0.4608
7	1.7079	8.9638	2.9940	0.8534	0.3573	0.4745	7	1.9963	11.5996	3.4058	0.8806	0.3575	0.4665
8	1.7076	8.8369	2.9727	0.8433	0.3578	0.4766	8	2.0277	13.2786	3.6440	0.8697	0.3627	0.4758
9	1.7299	9.2153	3.0357	0.8593	0.3597	0.4788	9	2.0245	12.4556	3.5292	0.8863	0.3601	0.4713
Mean	1.7121	9.2636	3.0412	0.8507	0.3576	0.4764	Mean	2.0252	13.0855	3.6140	0.8785	0.3595	0.4687
Std	0.0131	0.7453	0.1204	0.0074	0.0011	0.0024	Std	0.0181	1.1384	0.1569	0.0081	0.0020	0.0056
2.1599096835001278 13.139418640890218 3.6248336018209466 0.22100152355142572 0.47010799137158443 0.7848403817738328							2.5410284362022018 18.65964492299773 4.3196811135774515 0.22793226503588174 0.47742252254777606 0.8265109173926287						

[그림 2.5.9] : 18~20시, 20~22시 Model]

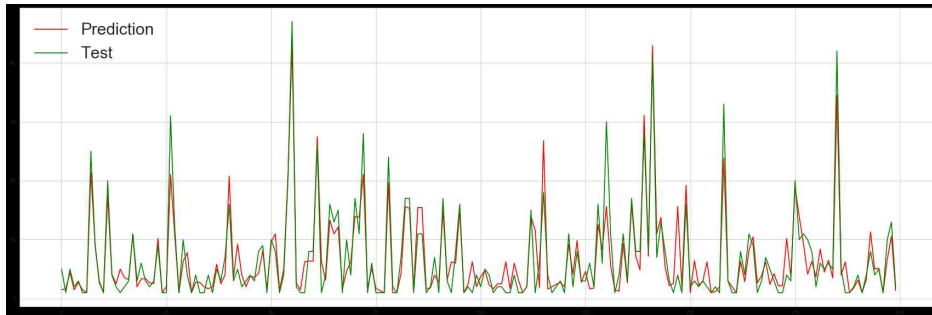
	MAE	MSE	RMSE	R2	RMSLE	MAPE		MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold							Fold						
0	1.6902	7.7132	2.7773	0.8759	0.3465	0.4568	0	1.3323	5.4123	2.3264	0.8731	0.3309	0.4427
1	1.6929	8.8742	2.9790	0.8714	0.3470	0.4552	1	1.4063	6.0723	2.4642	0.8630	0.3450	0.4694
2	1.6935	8.0363	2.8348	0.8678	0.3499	0.4646	2	1.3630	6.5312	2.5556	0.8585	0.3345	0.4471
3	1.6623	7.4136	2.7228	0.8736	0.3472	0.4591	3	1.3706	5.8976	2.4285	0.8610	0.3386	0.4570
4	1.6513	7.6512	2.7661	0.8628	0.3465	0.4640	4	1.3527	5.7083	2.3892	0.8703	0.3382	0.4589
5	1.6695	7.7256	2.7795	0.8691	0.3487	0.4647	5	1.3991	6.5338	2.5561	0.8564	0.3404	0.4571
6	1.6915	8.1124	2.8482	0.8713	0.3485	0.4589	6	1.3841	6.5940	2.5679	0.8641	0.3412	0.4596
7	1.6710	7.4719	2.7335	0.8783	0.3460	0.4526	7	1.3719	6.0820	2.4662	0.8481	0.3400	0.4564
8	1.7090	8.1954	2.8628	0.8722	0.3469	0.4557	8	1.3927	6.8913	2.6251	0.8643	0.3400	0.4592
9	1.6808	7.4643	2.7321	0.8750	0.3476	0.4588	9	1.3786	6.3456	2.5191	0.8774	0.3343	0.4454
Mean	1.6812	7.8658	2.8036	0.8717	0.3475	0.4590	Mean	1.3751	6.2068	2.4898	0.8636	0.3383	0.4553
Std	0.0165	0.4272	0.0751	0.0042	0.0012	0.0040	Std	0.0211	0.4311	0.0870	0.0081	0.0038	0.0076
2.1192377932674065 11.21454196971788 3.348812023646278 0.2079750581574901 0.4560428249161367 0.8146698021438726							1.769285597812044 8.706557842109499 2.9506876896936243 0.20490636515909752 0.4526658427130299 0.8231902533045252						

결정계수 값은 평균적으로 약 0.85, MAE 값은 약 1.7정도로, 모든 시간대에서 준수한 성능을 보여주고 있는 것을 확인할 수 있었다. 모델 검증은 fold 값을 10으로 하여 k-fold 교차 검증을 진행해주었다.

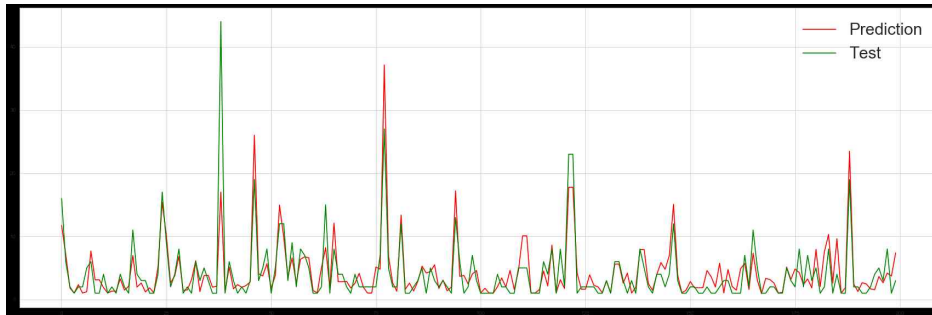
[그림 2.5.10 : 06~08시]



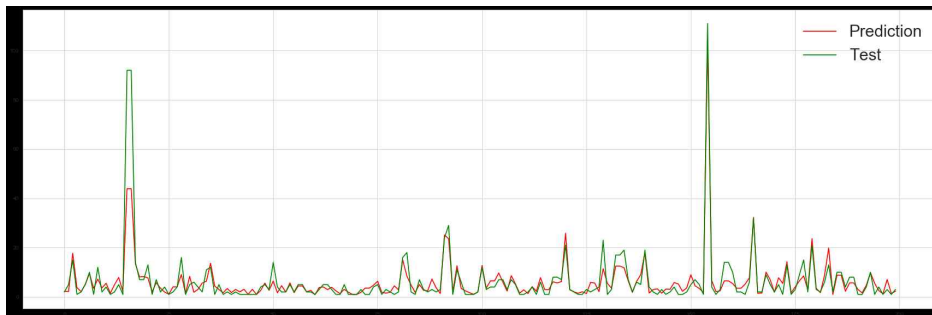
[그림 2.5.11 : 08~10시]



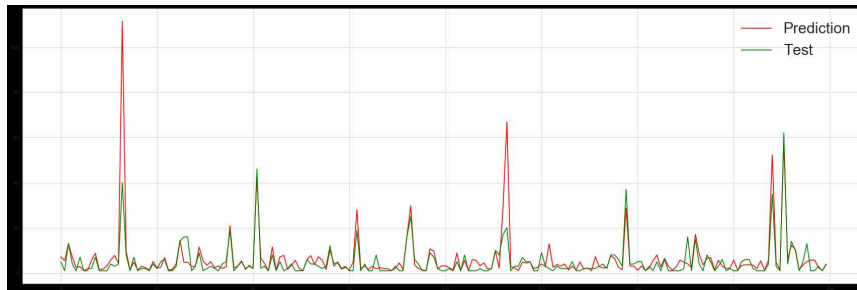
[그림 2.5.12 : 10~12시]



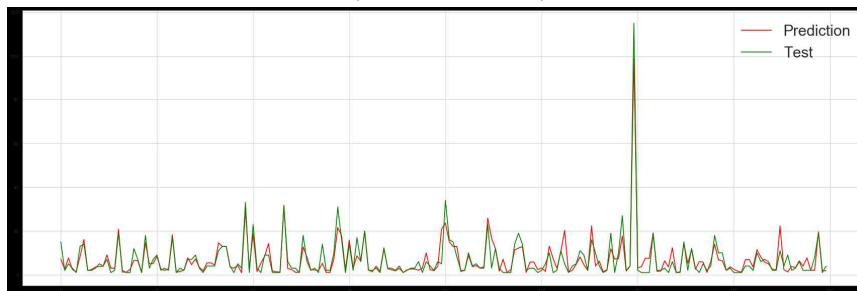
[그림 2.5.13 : 12~14시]



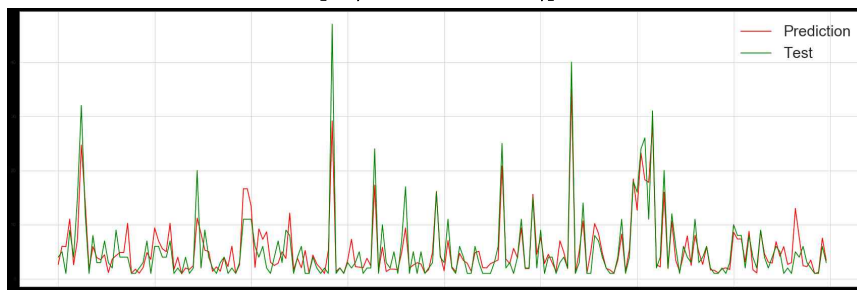
[그림 2.5.14 : 14~16시]



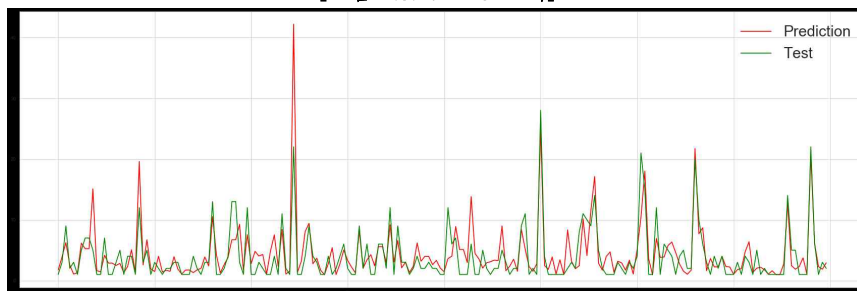
[그림 2.5.15 : 16~18시]



[그림 2.5.16 : 18~20시]



[그림 2.5.17 : 20~22시]



또한 모든 시간대의 Model이 실제값의 추세를 잘 따라가고 있는 모습을 확인할 수 있다. 이로써 모델의 성능 검정을 마친다.

제 3 장 Web Building

코드 작업 환경은 Pycharm을 사용.

3.1 BackEnd Building

웹은 김응용 저 “점프 투 장고” 저서를 참고하여 구축하였음.

3.1.1 Server

[그림 3.1.1]

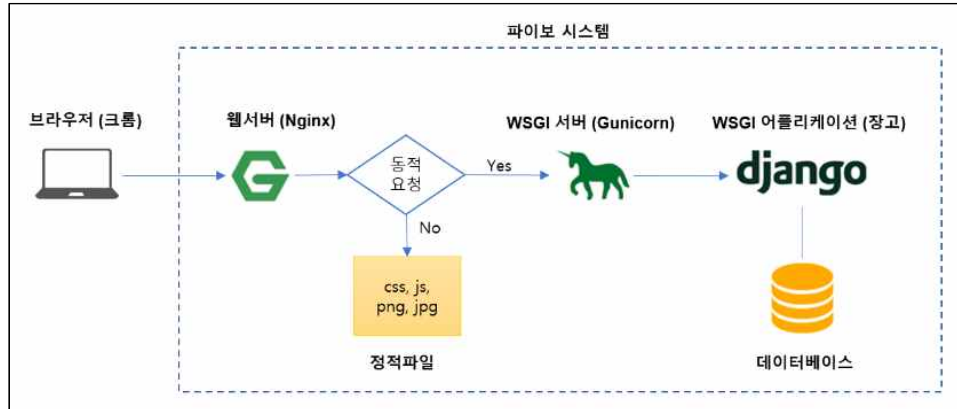
분류	이름
Cloud System	AWS Lightsail
Web Server (Static)	Nginx
WSGI Server	Gunicorn
DB	SQLite
Application	Django

우선, 클라우드 시스템으로는 아마존 웹 서비스의 Lightsail을 사용해 주었다. AWS가 비싸기도 하고 사용 방법이 어렵다는 단점이 있지만, 라이트 세일이 이 단점들을 모두 보완해 줄 수 있기에 클라우드 서버로는 AWS의 라이트 세일을 채택하였다. 한 달 5달러 요금의 서버를 구입하였으며, 현재는 AWS에서 3개월 무료 플랜을 지원해줘서 무료로 사용하고 있다.

웹 서버로는 일반적으로 장고와 많이 사용하는 엔진엑스를 사용하였다. 또한, 빠정은 특성상 시간 및 정류장 등에 따라 응답 데이터가 수시로 변하는데, 이러한 동적 웹페이지를 처리할 위스키 서버가 필요하다. 이 때, 위스키 서버로는 g유니콘과 u위스키 중 g유니콘을 채택하였다. 위스키 서버 어플리케이션으로는 사용하기 편리한 완성형 프레임워크 장고를 사용하였다.

아래는 이에 대한 간략한 흐름도를 나타내고 있다. 엔진 엑스가 정적 요청을 처리하고 동적 요청은 g유니콘이 장고에 넘겨줘서 처리하는 방식으로 웹이 동작하고 있음을 알 수 있다.

[그림 3.1.2]



출처 : 점프 투 장고

SSH 클라이언트는 MobaXterm을 사용하였다. 운영체제는 Ubuntu를 사용하였으며, 로컬 환경에서 작업한 내용을 깃허브에 push한 뒤, MobaXterm에서 pull해서 사용하는 식으로 개발을 진행하였다. 아래는 systemctl status로 가동중인 서버 상태를 확인한 창이다. 서버 이름 bus로 정상 가동중인 모습이다.

[그림 3.1.3]

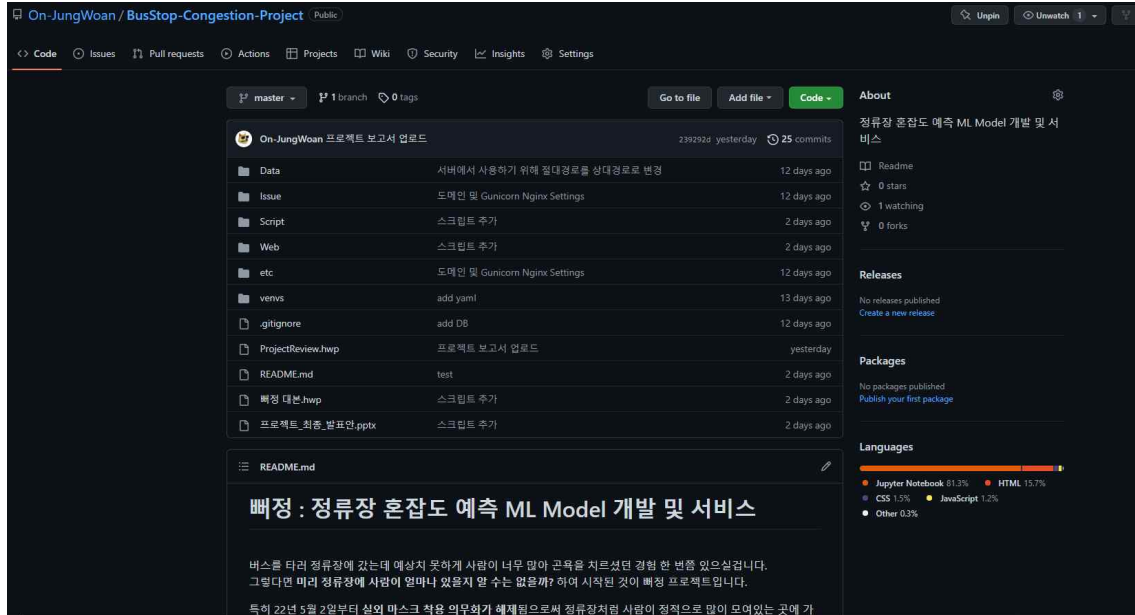
The screenshot shows a terminal window with the command `systemctl status bus` executed. The output indicates that the `bus` service is running. The terminal also shows a file explorer on the left with the `/home/ubuntu/` directory selected.

```

State: running
Jobs: 0 queued
Failed: 0 units
Since: Tue 2022-06-07 02:24:05 KST; 1 weeks 5 days ago
CGroup: /
├─user.slice
│ └─user-1000.slice
│   └─user@1000.service
│     └─init.scope
│       ├── 174442 /lib/systemd/systemd --user
│       └─174446 (sd-pam)
│         └─session-377.scope
│           ├── 174431 sshd: ubuntu [priv]
│           ├── 174551 sshd: ubuntu@pts/0
│           ├── 174557 -bash
│           ├── 174589 systemctl status
│           └─174590 pager
├─init.scope
│ └─1 /lib/systemd/systemd --system --deserialize 32
├─system.slice
│ └─bus.service
│   ├── 108393 /home/ubuntu/anaconda3/envs/bus_venvs/bin/python /home/ubuntu/ana
│   ├── 108406 /home/ubuntu/anaconda3/envs/bus_venvs/bin/python /home/ubuntu/ana
│   └─108407 /home/ubuntu/anaconda3/envs/bus_venvs/bin/python /home/ubuntu/ana
├─systemd-networkd.service
│ └─32302 /lib/systemd/systemd-networkd
├─systemd-udevd.service
│ └─41203 /lib/systemd/systemd-udevd
├─cron.service
│ └─477 /usr/sbin/cron -f
├─system-serial\x2dgetty.slice
│ └─serial-getty@tty50.service
│   └─618 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 tty50 vt220

```


[그림 3.1.4]



깃허브 레퍼지토리 화면이다. 로컬에서 push 한 내용을 확인할 수 있다. 서버에서는 해당 내용을 pull해서 사용한다.

3.1.2 Domain

도메인은 가비아에서 구매하였으며, 연당 2만원의 비용이 발생한다.

[그림 3.1.5]



3.1.3 Model & DB

다음은 파생변수 Mean_Sum에 대한 Model Script이다. 고유 필드값 idx와 target값(주요 변수 별 7가지 조합), mean값 sum값 필드로 구성되어 있다.

[Script 3.1.1]

```
class r_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)

class i_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)

class w_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)

class ri_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)

class rw_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)

class iw_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)

class riw_mean_sum(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    m = models.FloatField(blank=True, null=True)
    s = models.FloatField(blank=True, null=True)
```

다음은 파생변수 Congestion에 대한 Model Script이다. 고유 필드값 idx와 target값(3가지 주요 변수), 혼잡도 Min-Max Scaling값 필드로 구성되어 있다.

[Script 3.1.2]

```
class i_congestion(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    con = models.FloatField(blank=True, null=True)

class r_congestion(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    con = models.FloatField(blank=True, null=True)

class w_congestion(models.Model):
    idx = models.AutoField(primary_key=True)
    target = models.CharField(max_length=200, blank=True, null=True)
    con = models.FloatField(blank=True, null=True)
```

다음은 파생변수 Distance에 대한 Model Script이다. 고유 필드값 idx와 i값(ARS_ID), 주요 정류장과의 거리 필드로 구성되어 있다.

[Script 3.1.3]

```
class distance(models.Model):
    idx = models.AutoField(primary_key=True)
    i = models.CharField(max_length=200, blank=True, null=True)

    dis_0 = models.FloatField(blank=True, null=True)
    dis_1 = models.FloatField(blank=True, null=True)
    dis_2 = models.FloatField(blank=True, null=True)
    dis_3 = models.FloatField(blank=True, null=True)
    dis_4 = models.FloatField(blank=True, null=True)
    dis_5 = models.FloatField(blank=True, null=True)
    dis_6 = models.FloatField(blank=True, null=True)
    dis_7 = models.FloatField(blank=True, null=True)
    dis_8 = models.FloatField(blank=True, null=True)

    time = models.CharField(max_length=200, blank=True, null=True)
```

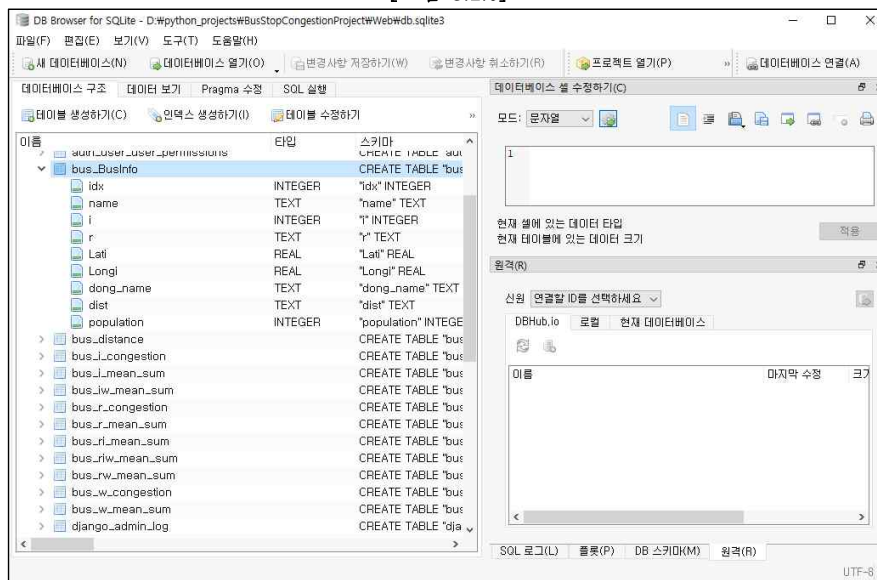
다음은 Web에서 버스 데이터를 처리하기 위해 버스 정보를 담은 Model을 생성해 준 스크립트이다. 고유 인덱스 번호와, 정류장명, ARS_ID, 노선명, 경/위도, 동명, 행정구역(동), 행정구역 별 인구 필드로 구성되어 있다.

[Script 3.1.4]

```
class BusInfo(models.Model):
    idx = models.AutoField(primary_key=True)
    name = models.CharField(max_length=200, blank=True, null=True)
    i = models.CharField(max_length=200, blank=True, null=True)
    r = models.CharField(max_length=200, blank=True, null=True)
    Lati = models.FloatField(blank=True, null=True)
    Longi = models.FloatField(blank=True, null=True)
    dong_name = models.CharField(max_length=200, blank=True, null=True)
    dist = models.CharField(max_length=200, blank=True, null=True)
    population = models.FloatField(blank=True, null=True)
```

DB 생성을 위해서는 SQL문을 작성해야 하지만, 해당 기능은 Django에서 기본 기능으로 제공하고 있다. makemigration과 migrate 명령어로 테이블을 생성해주고 SQLite에서 확인한다.

[그림 3.1.6]

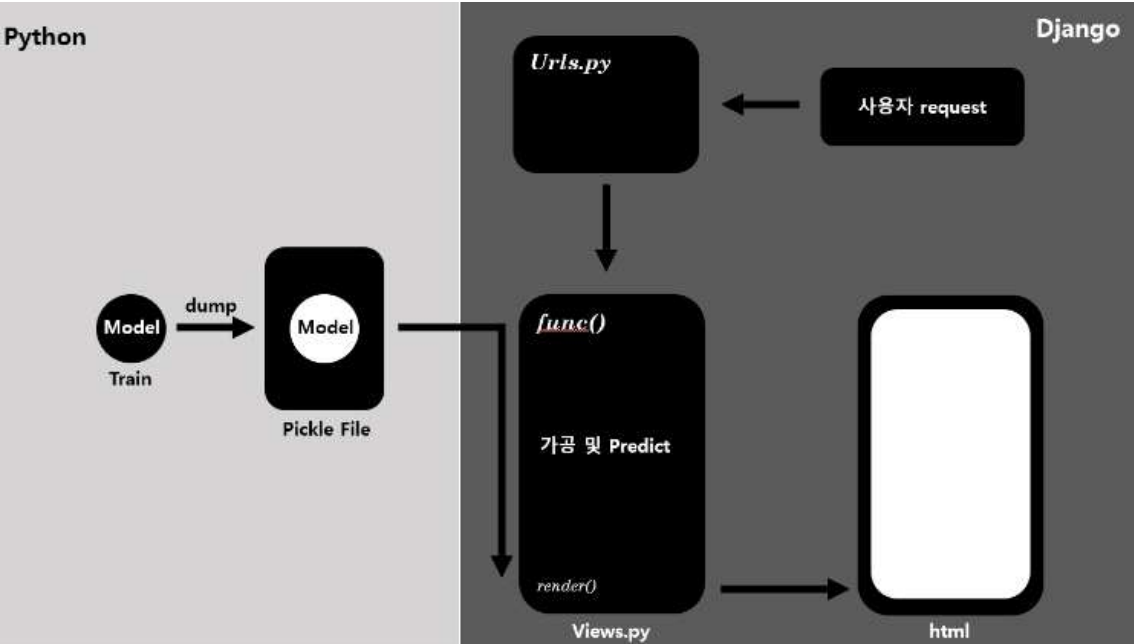


정상적으로 테이블이 생성된 것을 확인할 수 있다. 이제 미리 준비해 둔 csv 파일을 SQLite에 업로드 한다. 해당 csv 파일은 본인 github에서 확인할 수 있다.
[BusStop-Congestion-Project/Data at master · On-JungWoan/BusStop-Congestion-Project \(github.com\)](https://github.com/On-JungWoan/BusStop-Congestion-Project)

3.1.4 URL & View

다음은 간단한 백엔드 흐름도이다. 우선, 파이썬 환경에서 학습을 마친 모델을 pickle파일로 저장하여 서버에 저장해둔다. 이후 Django 환경의 View함수에, pickle 파일을 불러와 predict 하는 알고리즘을 미리 작성해두고, 사용자가 url을 요청하면 View함수를 호출하여 html로 넘겨주는 형태로 웹이 동작한다.

[그림 3.1.7]



-
-
-
-

다음은 해당 내용에 대한 스크립트이다.

한글 버전 문제로 인해
더 이상 코드 복사 붙여넣기가 되지 않아 여기서부터는 코드를 사진으로 첨부하겠음

[Script 3.1.5]

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('bus/', include('bus.urls')),
    path('', views.index)
]
```

[Script 3.1.6]

```
from django.urls import path

from . import views

app_name = 'bus'

urlpatterns = [
    path('main/', views.index, name='main'),
    path('main/about_us', views.about, name='about-us'),
    path('detail/', views.detail, name='detail'),
    path('detail/search', views.search, name='search'),
    path('detail/<str:dist>+<str:id_>+<str:route>/', views.info, name='info'),
]
```

config의 urls.py에서 “도메인 주소/” url에 대해서는 bus app의 urls.py로 넘기고 “도메인 주소/”에 대해서는 views.py의 index 함수로 넘겨주고 있다. bus app의 urls.py에서는 다음과 같은 url을 정의하고 있다.

도메인주소/main : views.py의 index
도메인주소/main/about_us : (현재는 사용x)
도메인주소/detail : views.py의 detail
도메인주소/detail/search : views.py의 search
도메인주소/dist/id/route/ : views.py의 info에 dist, id, route를 넘겨줌

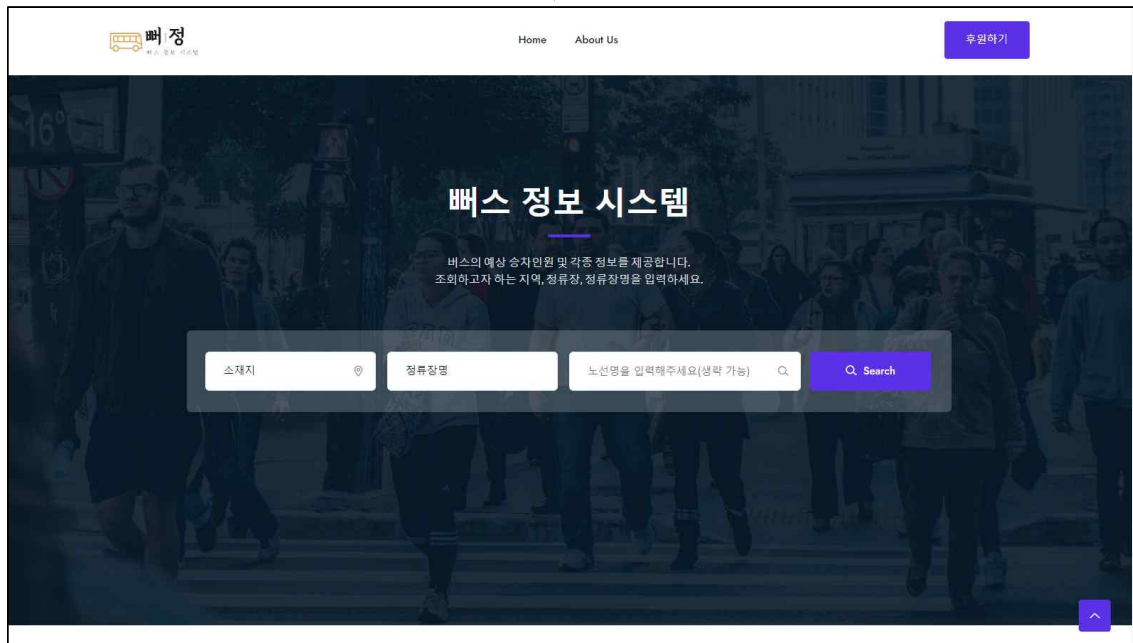
view 함수는 코드가 400줄이 넘어가는 관계로 간단히 글로만 설명하겠다. 자세한 코드는 본인의 깃허브에서 확인할 수 있다.

파일 경로 : On-JungWoan/BusStop-Congestion-Project/Web/bus/views.py

URL 주소 : [BusStop-Congestion-Project/Web/bus at master · On-JungWoan/BusStop-Congestion-Project \(github.com\)](#)

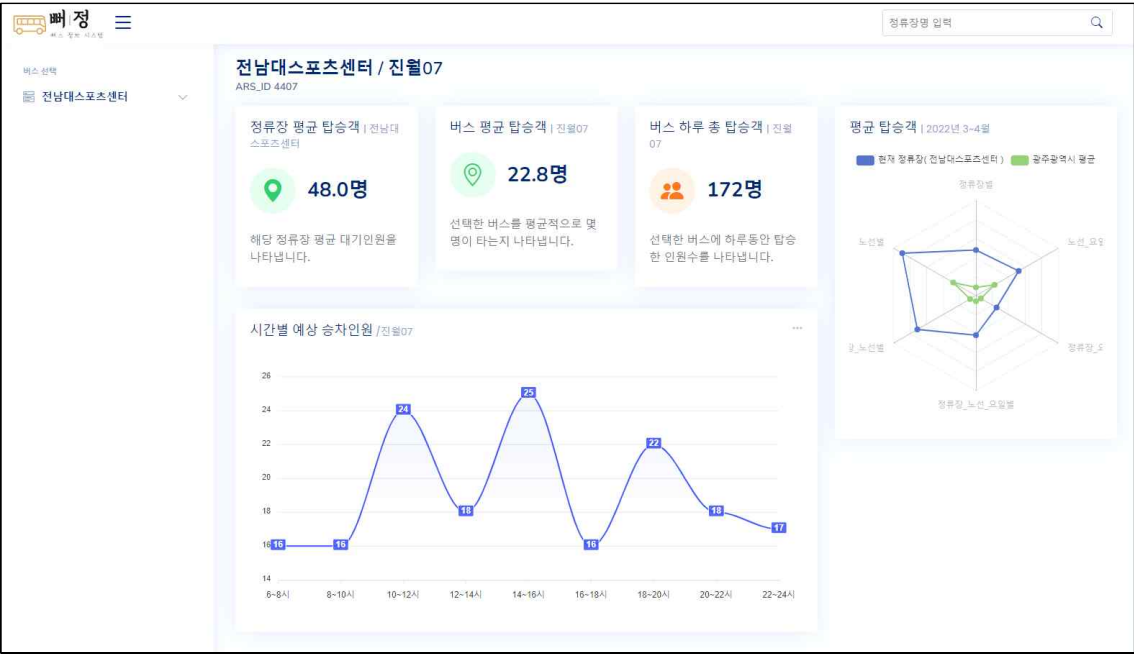
index 함수는 메인 화면을 출력하는 함수로, DB에서 Businfo를 가져와서 템플릿 경로 상의 main/index.html로 넘겨준다. 해당 URL이 동작하면 다음과 같은 화면이 출력된다.

[그림 3.1.8]



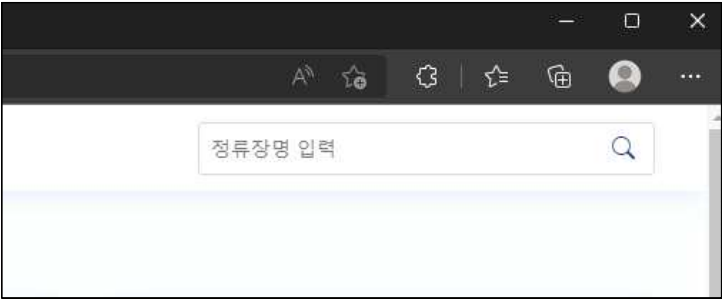
detail 함수는 상세 정보 화면을 출력하는 함수로, 메인 화면 select box에서 선택한 값을 GET-POST 방식으로 가져와 상세 정보 화면을 출력한다. 해당 URL이 동작하면 다음과 같은 화면이 출력된다.

[그림 3.1.9]



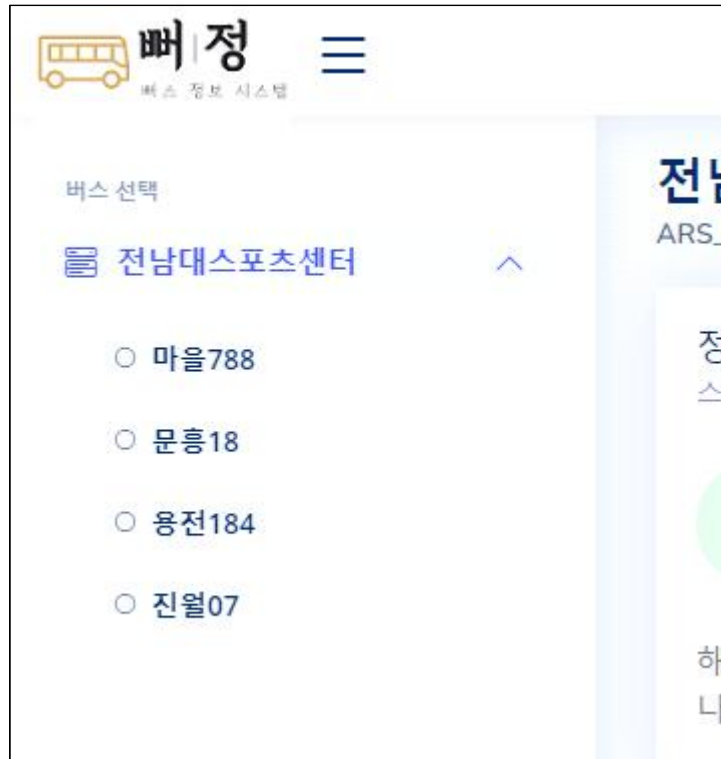
search 함수는 그림 3.1.7의 우측 상단 search box 관련 함수로, search box에 입력 된 정류장명을 GET-POST 방식으로 가져와 해당 내용의 상세 페이지를 출력한다. 해당 URL이 동작하면 그림 3.1.7과 같은 상세 페이지가 출력된다.

[그림 3.1.10]



info 함수는 그림 상세 페이지의 좌측 네비게이션 바 관련 함수로, 네비게이션 바에서 선택한 정보를 url 주소로 dist id route에 받아와 해당 정보의 상세 페이지를 출력한다. 해당 URL이 동작하면 그림 3.1.7과 같은 상세 페이지가 출력된다.

[그림 3.1.11]



•

•

•

•

3.2 FrontEnd Building

프론트엔드 템플릿은 NiceAdmin과 ClassiGrids-free-lite를 사용하였다. 해당 템플릿의 static 경로와 bus app의 경로를 맞추어주는 작업을 해주었으며, 앞서 작성한 view 함수와 연동 해주는 작업을 해주었다. 해당 코드는 페이지 당 500줄이 넘어가므로 분량 문제로 인해 본인 github 주소로 대체하겠다. 백엔드 작업과 프론트엔드 연동 작업 모두 본인이 “직접” 작성한 코드임을 밝힌다.

static 파일 경로 : On-JungWoan/BusStop-Congestion-Project/Web/static

템플릿 파일 경로 : On-JungWoan/BusStop-Congestion-Project/Web/templates

URL 주소 : [BusStop-Congestion-Project/Web at master · On-JungWoan/BusStop-Congestion-Project \(github.com\)](#)

main : 메인 페이지

detail : 상세페이지

•

•

•

•

•

제 4 장 향후 활용 방안

혹자는 “버스 승차 인원을 예측해서 뭐해?” 라고 생각할 수도 있습니다. 하지만 앞서 소개 하였던 Bbeojung 웹은 제가 제시하는 모델 활용 방안의 일부이고, 이외에도 해당 모델을 활용할 수 있는 방안은 많이 있을 것입니다. 모델을 버스 전광판 시스템과 연동하여 전광판에 정류장 혼잡도를 제공할 수도 있고, 사용자가 접근하기 편리하도록 앱이나 챗봇을 출시하는 것도 이에 대한 한가지 활용 방안입니다.

이에 더불어 제시하는 또 다른 활용 방안은 다음과 같습니다.

.

.

1. 실시간으로 사람이 많이 모이는 상위 n개의 정류장 표시 기능 추가

해당 기능은 이동 장사(ex) 푸드 트럭, 잡상인)나 선거 유세 등에 좋은 효과를 보여줄 것으로 예상되며 경험이나 감에 의존하는 현행 방식보다 훨씬 좋은 성과를 보여줄 것임.

2. 버스 내부의 혼잡도 계산

해당 ML 모델을 응용하여 하차 인원과 한 정거장 전 승차인원을 예측한다. 그리고 나서 이 두 값의 차이를 구해주면 정류장의 혼잡도 뿐만 아니라 버스 내부의 혼잡도를 계산할 수 있다. 이를 버스 전광판과 연동하여 버스 예상 도착 시간과 더불어 내부 혼잡도를 표시하는 기능을 구현할 수도 있을 것임.

.

.

이처럼 해당 모델의 활용 방안은 무궁무진합니다. 해당 내용은 일회성으로 끝나는 것이 아니라 지속적으로 발전시키고 다듬어 나가 광주광역시의 대중 교통 발전에 이바지할 것입니다.

부 록

참고자료 출처

1. DataSet

	출처	내용
광주_버스_이용객.csv	공공데이터 포털	광주광역시 3~4월 버스 이용객
광주광역시_정류장_위치정보.csv	공공데이터 포털	정류장의 경도, 위도 데이터
Google Geocoding API	Google Maps	해당 경/위도의 주소 정보를 반환해 줌.
광주광역시_행정지역_인구.csv	KOSIS	광주광역시의 행정지역(동)별 인구 데이터셋
법정동_행정동_맵핑.csv	공공데이터 포털	전국 법정동과 행정동 맵핑 데이터
2018~2022_공휴일_정보.csv	공공데이터 포털	2018~2022 공휴일 정보
기상청_기상_데이터.csv	기상자료개방포털	기온, 습도, 강수량, 풍속

1.1. 광주광역시_ 시내버스 노선별 승하차 인원 정보

XLS

광주광역시_시내버스 노선별 승하차 인원 정보

다운로드

오류신고 및 담당자 문의

광주광역시 내 시내버스 노선별, 정류장별 5월 승하차 인원정보에 대한 데이터로, 일자별, 노선명, 정류장명, 시간별, 승하차별 거래건수를 제공합니다.

0

0

관심

[출처] 공공데이터 포털 : <https://www.data.go.kr/data/15088456/fileData.do>

1.2. 국토교통부_전국 버스정류장 위치정보

CSV

국토교통부_전국 버스정류장 위치정보

다운로드

오류신고 및 담당자 문의

버스정보시스템(BIS)이 구축된 지자체 중 국가대중교통정보센터(TAGO)와 연계된 136개 지자체에 대한 버스정류장 위치정보 데이터입니다.
데이터에 대한 세부 설명은 첨부한 파일을 참고하십시오.
*안동·예천의 경우 안동, 원주·횡성은 원주, 제주·서귀포는 제주, 대전·계룡은 대전, 목포·무안·신안·영암은 목포, 영덕·청송·영양은 영덕으로 통합되어 있습니다.

5

0

관심

[출처] 공공데이터 포털 : <https://www.data.go.kr/data/15067528/fileData.do>

1.3. Geocoding API

The screenshot shows the Google Maps Platform API console. The left sidebar has a menu with '개요' (Overview) selected. The main content area is titled '사용 설정된 API' (Enabled APIs) and shows a table of API usage for the 'Directions API'. The table has columns for 'API', '요청' (Requests), '오류' (Errors), and '평균 지연 시간(밀리초)' (Average latency in milliseconds). The 'Directions API' row shows 0 requests and 0 errors. The 'Distance Matrix API' row shows 0 requests and 0 errors. The 'Geocoding API' row shows 22,407 requests and 92 errors. The '평균 지연 시간(밀리초)' column shows 43 ms for the Geocoding API. A '세부정보' (Details) link is visible next to the Geocoding API row.

API	요청	오류	평균 지연 시간(밀리초)
Directions API	0	0	-
Distance Matrix API	0	0	-
Geocoding API	22,407	92	43

[출처] Google Maps :

<https://console.cloud.google.com/google/maps-apis/api-list?project=sunlit-context-350805>

1.4. 광주광역시 행정구역 인구

KOSIS

통계목록

주제별 통계

스크랩 ☆

내가 본 통계표 ①

통계목록

다이어리

검색

▼ 1 오름작순

인구

사회일반

법치·안전

노동

소득·소비·자산

보건

복지

교육·훈련

문화·여가

주거

국도이용

1. 행정구역(시군구)별, 성별 ...

모두 닫기

행정구역(시군구)별, 성별 인구수

주인등록인구현황, 행정안전부 [자료문의지:044-205-3158]

통계설명자료

수속기간: 월, 연 1992 ~ 2022.05 / 자료갱신일: 2022-06-03 / 주시작정보

시현

승진(출진)

행정현황

열고자찾해

새 열 열기

화면복사

주소정보

스크랩

인쇄


다운로드

행정구역(시군구)별	2022.03			2022.04		
	총인구수 (명)	남자인구수 (명)	여자인구수 (명)	총인구수 (명)	남자인구수 (명)	여자인구수 (명)
전국	51,610,695	25,730,584	25,880,111	51,592,660	25,721,053	25,871,607
서울특별시	9,506,778	4,613,910	4,892,868	9,500,480	4,610,082	4,890,398
2부산광역시	3,343,504	1,634,958	1,708,546	3,340,214	1,633,170	1,707,044
대구광역시	2,380,494	1,172,632	1,207,862	2,378,573	1,171,451	1,207,122
인천광역시	2,952,699	1,478,722	1,474,427	2,953,260	1,478,514	1,474,746
광주광역시	1,438,463	711,289	727,174	1,436,916	710,524	726,392
강원광역시	1,459,433	725,111	734,322	1,457,811	723,711	734,100


[출처] KOSIS :

https://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_1B040A3


1.5. 전국 법정동 행정동 매핑데이터


KDX 한국데이터거래소

[데이터 마켓](#)
[AI 비즈니스](#)
[서비스](#)
[커뮤니티](#)
[기업회원 사이트](#)


[로그인](#)

전국 법정동 행정동 맵핑데이터



기본 속성

상품 카테고리	물류/교통	<div>총 구매금액</div> <div>무료</div>
상품명	전국 법정동 행정동 맵핑데이터	

[출처] 한국데이터거래소 : <https://kdx.kr/data/view/418>

1.6. 한국천문연구원_특일정보 (공휴일 정보)

XML 한국천문연구원 특일 정보

(천문우주정보)국경일정보, 공휴일정보, 기념일정보, 24절기정보, 잡절정보를 조회하는 서비스 입니다.
활용시 날짜, 순번, 특일정보의 분류, 공공기관 휴일 여부, 명칭을 확인할 수 있습니다.

61 4 관심

활용신청
오류신고 및
담당자 문의

[출처] 공공데이터 포털 : <https://www.data.go.kr/data/15012690/openapi.do>

1.7. 기상 데이터

기상청 기상자료개방포털

국가기후데이터센터 소개 | *가~카 | 로그인 | 사이트맵 | 즐겨찾기 | ENG(info)

'관측'을 검색하세요
인기검색어

기상자료개방포털이란? 데이터 기후통계분석 간행물 소통과 참여

데이터

기상관측

지상

* Home > 데이터 > 기상관측 > 지상 > 종관기상관측(ASOS)

종관기상관측(ASOS) - 자료

자료설명

종관기상관측이란 종관규모의 날씨를 파악하기 위하여 정해진 시각에 모든 관측소에서 같은 시각에 실시하는 지상관측을 말합니다.
종관규모는 일기도에 표현되어 있는 보통의 고기압이나 저기압의 공간적 크기 및 수명을 말하며, 주로 매일의 날씨 현상을 뜻합니다.

자료형태	분, 시간(매정시), 일, 월, 연	제공기간	1904년-(지점별, 요소별 다름)
제공지점	103개 * 원하는 지점이 없는 경우, 방재기상관측(AWS) 메뉴 이용	제공요소	기온, 강수, 바람, 기압, 습도, 일사, 일조, 눈, 구름, 시정, 지면상태, 지면·초상온도, 일기현상, 중발량, 현상번호

[출처] 기상청 기상자료개방포털 : <https://data.kma.go.kr/data/grnd/selectAsosRltnList.do?pgmNo=36>

2. Script

[Dacon : 제주 퇴근시간 버스승차인원 예측 경진대회]

2.1.1) 제주 퇴근시간 버스승차인원 예측 베이스 라인 중급 코드

<https://dacon.io/competitions/official/229255/codeshare/710?page=1&dtype=recent>

참고 내용 : 승차인원끼리의 heatmap 비교(EDA에서 사용) script, folium을 사용한 지도 그리기 script 및 좌표 데이터를 활용한 location Feature idea, 기상청 자료 사용 idea

2.1.2) [버스 승차예측 스터디] 1st week_EDA

<https://dacon.io/competitions/official/229255/codeshare/1402?page=1&dtype=recent>

참고 내용 : 결측치 시각화 방법(mnso), 버스 수요는 요일의 영향을 받는다는 idea

2.1.3) 제주 퇴근시간 버스승차인원 예측 1등 돌하르방팀 코드

<https://dacon.io/competitions/official/229255/codeshare/511?page=1&dtype=recent>

참고 내용 : 요약통계량 생성에 대한 idea 및 script, 혼잡도 feature에 대한 idea(해당 코드는 scoring, 본인은 min-max scaling을 사용), Label Encoding script(사용하지는 않음), 날씨 정보에 대한 idea, 거리 구하는 script

3. 기타 자료 출처

1) 메인 화면 템플릿 : [ClassiGrids - Free Classified Ads HTML Template and UI Kit - GrayGrids](#)

2) 상세 화면 템플릿 : [Nice Admin -Free bootstrap admin HTML template | BootstrapMade](#)

참고문헌

1. 앙상블 학습법

[머신러닝 - 11. 앙상블 학습 \(Ensemble Learning\): 배깅\(Bagging\)과 부스팅\(Boosting\) \(tistory.com\)](#)

2. 광주광역시 버스 운행정보 사이트

[광주광역시 버스운행정보 \(gwangju.go.kr\)](#)

3. Pycaret 공식 홈페이지

[Welcome to PyCaret - PyCaret Official \(gitbook.io\)](#)

4. 점프 투 장고 WikiDocs

[점프 투 장고 - WikiDocs](#)

5. 부트스트랩 템플릿 적용법

[SpringBoot - 부트스트랩 템플릿 적용하기 \(velog.io\)](#)