머신러닝을 활용한 버스도착시간 예측 모델

버스오긴오조 오승아 김기찬 김하연

A table of contents

Part 1 구성원 & 프로젝트 소개

0 구성원 역할 및 소개

조장 오승아

데이터전처리, 분석/머신러닝 모델링

팀원 김기찬

데이터전처리, 분석/실시간데이터수집(ALL)/코드전임코치/회의록작성

팀원 김하연

데이터전처리, 분석/머신러닝 모델링/PPT작성 및 발표

1 프로젝트 소개

- 프로젝트진행요약

프로젝트 목표설정	데이터 수집	데이터 분석 및 전처리	모델링	보완 및 수정
▪ 프로젝트 설정 ▪ 분석 방향	▪ API ▪ 크롤링 ▪ 스트리밍	▪ 데이터 파악 ▪ 데이터 결측치 제거 ▪ 이상치 제거 ▪ 값 추출 및 칼럼 생성 ▪ 칼럼 기준 병합	▪ Scaling ▪ 회귀분석 (Regressor) ▪ Feature Importance ▪ 하이퍼 파라미터 조정	▪ 결론도출 ▪ 보완점

1 프로젝트 소개

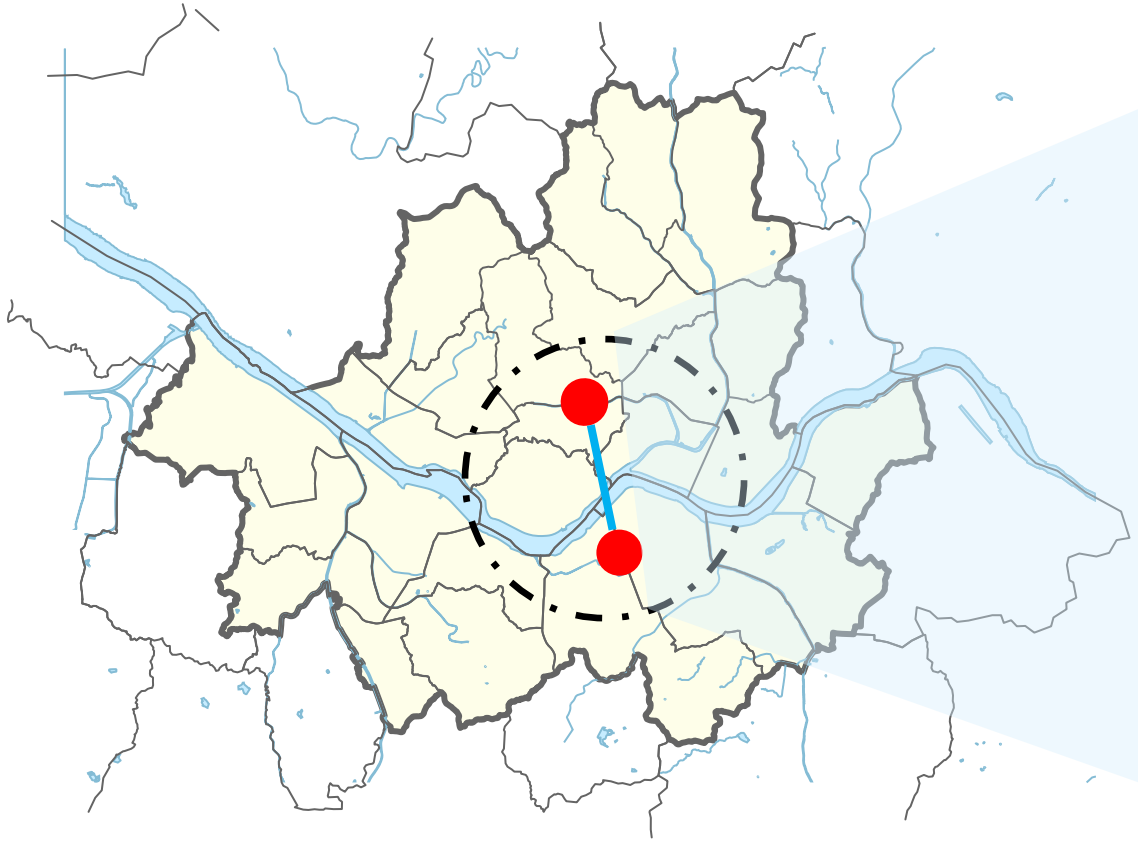
버스

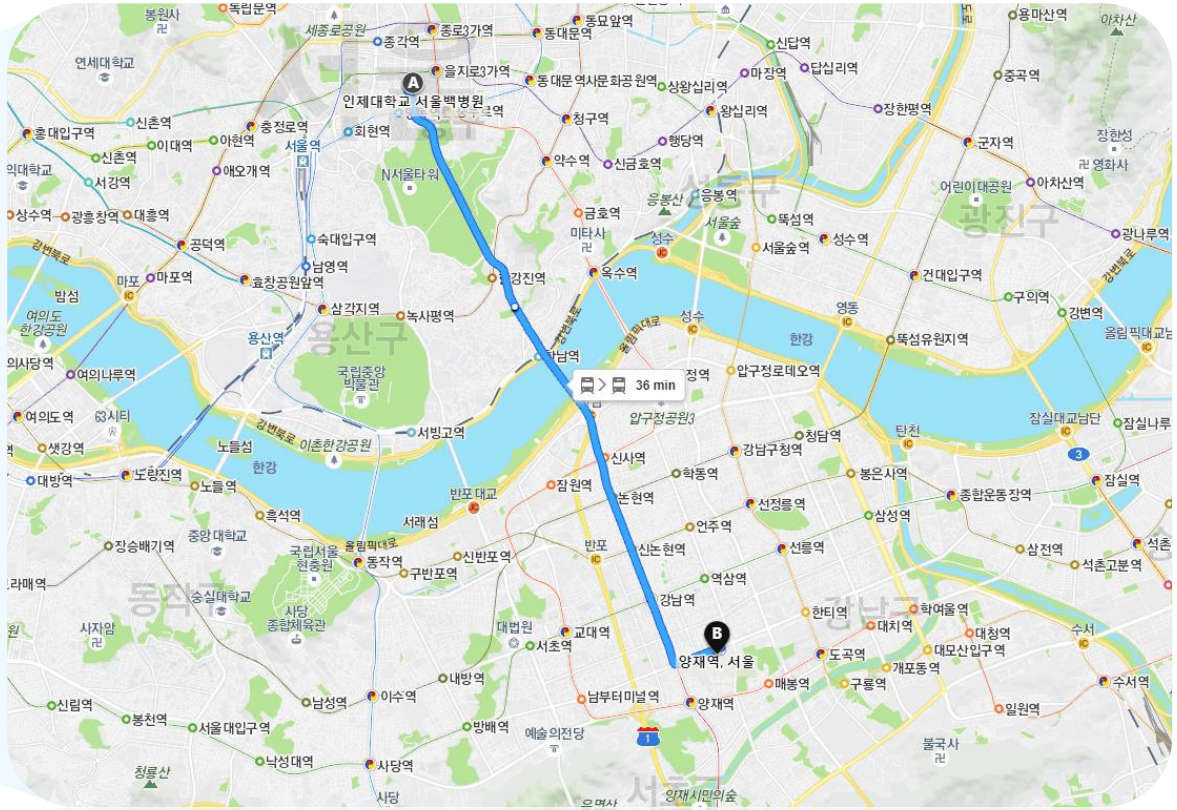
날씨

서울백병원-양재역 구간

실시간버스위치API와 날씨데이터를활용한 버스도착시간예측

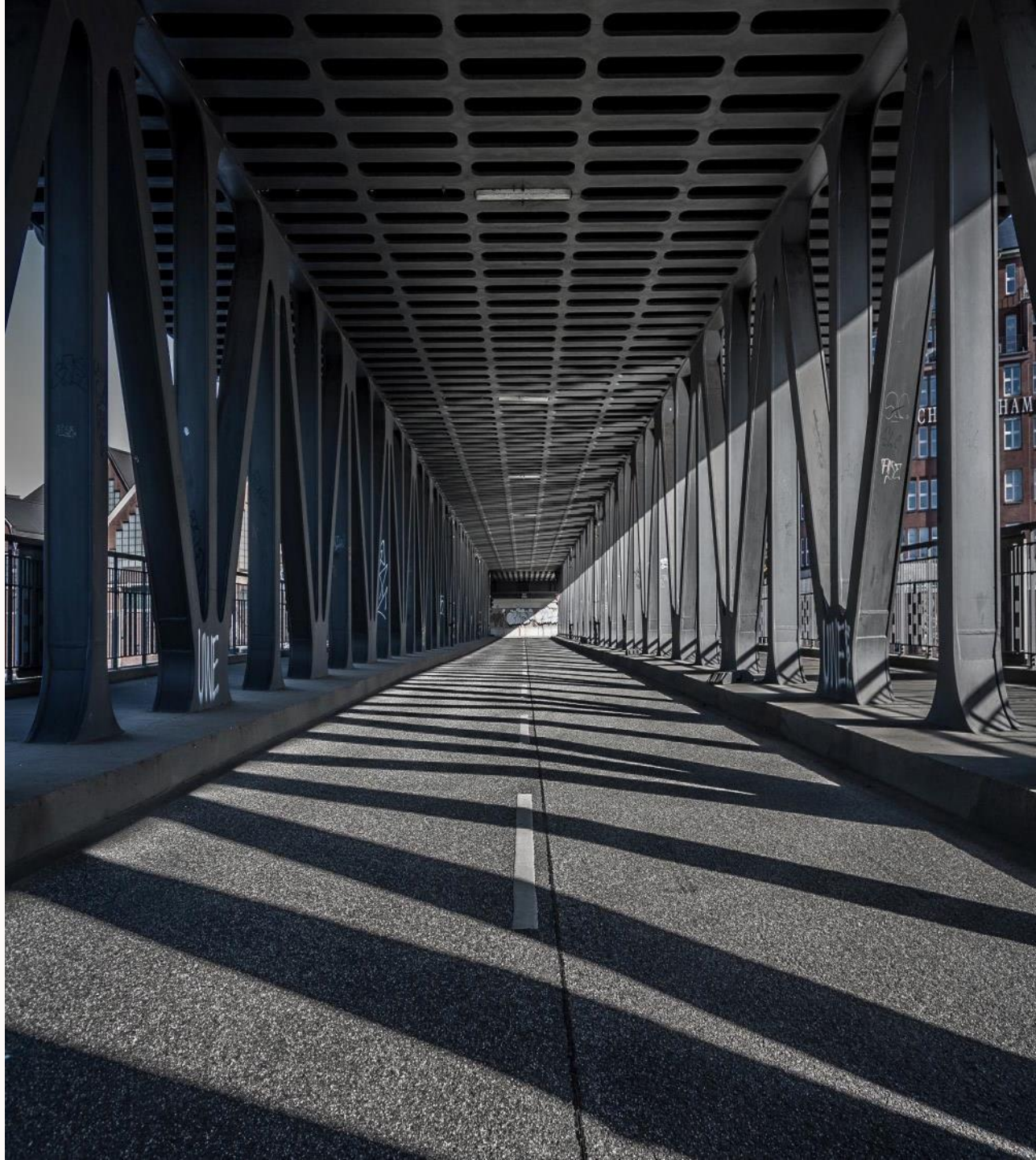
1 프로젝트 소개





420번 버스 (서울백병원-양재역 구간)

Part 2 데이터 수집



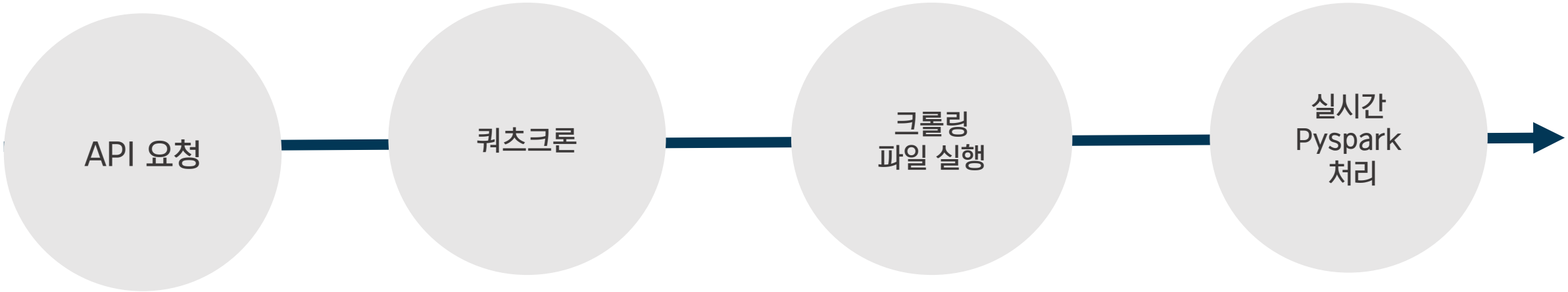
2 데이터 수집

활용데이터 (공공데이터포털)

- https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15000193
- https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15000332
- https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15084084

2 데이터 수집

데이터 수집 과정



2 데이터 수집

Ⅰ. 서울특별시 버스위치정보조회 서비스, 기상청 단기 예보 조회서비스에 실시간으로 요청

Ⅱ. 버스위치의 경우 인당 트래픽 제한으로 인해 한 노선에 대해서 1분 간격으로 데이터 요청

Ⅲ. 기상실황의 경우 1시간마다 새로운 정보가 갱신되어, 1시간 간격으로 데이터 요청

Ⅳ. 쿼츠 크론을 이용해 리눅스에서 1분 or 1시간 간격 API request 요청

Ⅵ. 데이터를 새로 수집할 때마다 동시에 pyspark를 이용하여 전처리

2 데이터 수집

세부코드

```
# bus, weather crawling
* 5-19 * * * bash -i ~/crawl_bus.sh
20 5-19 * * * bash -i ~/crawl_weather.sh
```

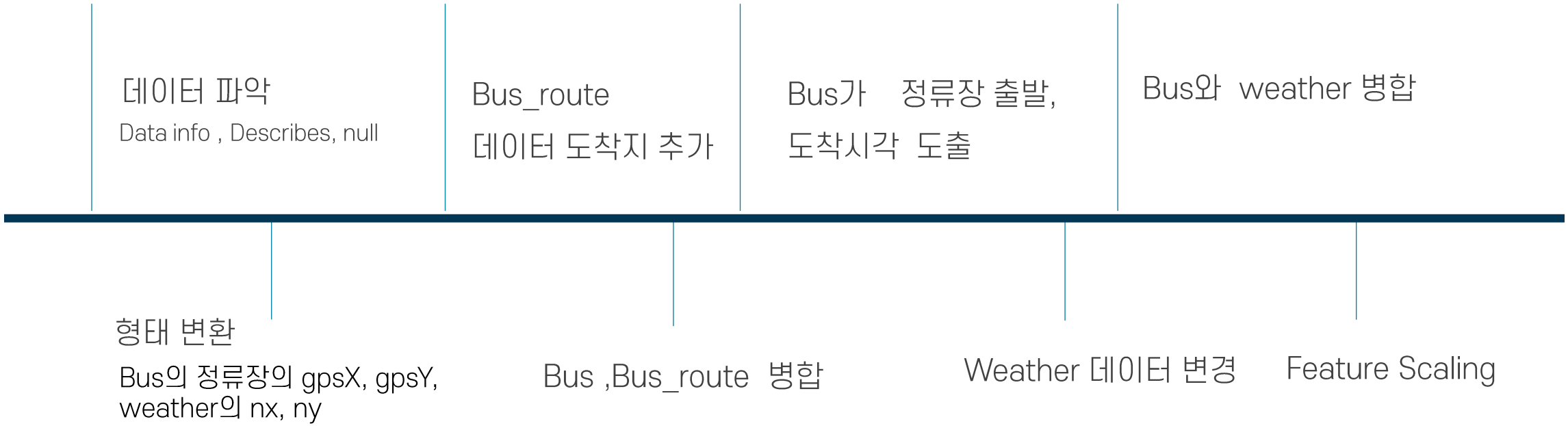
```
# crawl_bus.sh
conda activate gichan
python crawl_bus.py
conda deactivate
```

```
# process_bus100100073.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode

spark = SparkSession \
    .builder \
    .appName("bus100100073") \
    .getOrCreate()

busSchema = spark.read.format('json').load('/home/lab01/bus/20210813142502Bus100100073.json').schema
busDf = spark.readStream.schema(busSchema).json('/home/lab01/bus/*100100073.json')
df_bus = busDf.select(explode(busDf.ServiceResult.msgBody.itemList).alias("buses")).select('buses.*')
df_bus.coalesce(1).writeStream.format('json') \
    .option("checkpointLocation", "/home/lab01/bus100100073_check") \
    .option("path", "/home/lab01/bus100100073") \
    .trigger(processingTime='3600 seconds') \
    .start().awaitTermination()
```

Part 3 분석 및 전처리

3 분석 및 전처리

데이터 전처리 과정



3 분석 및 전처리

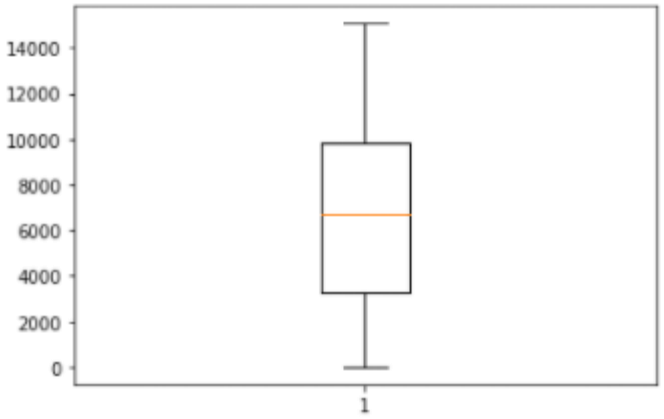
핵심 프로세스

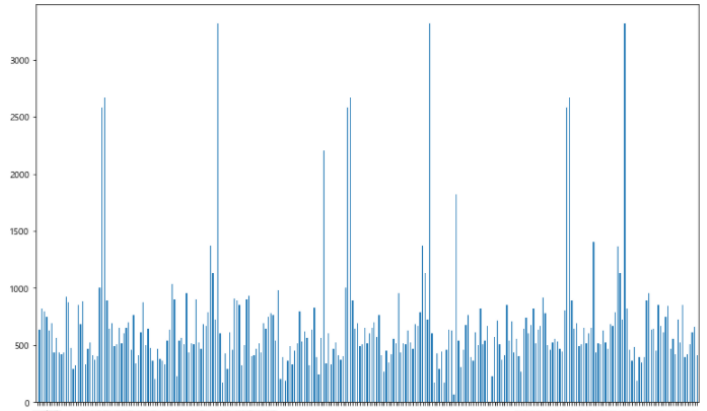
df_route_100100073

	busRouteId	section	seq	station_x	stationNm_x	nx_x	ny_x	station_y	stationNm_y	nx_y	ny_y
0	100100073	113600767	1	113000020	서부면허시험장	58	127	113000026	월드컵파크3단지정문	58	127
1	100100073	113600709	2	113000026	월드컵파크3단지정문	58	127	113000027	월드컵파크2단지.에스플렉스센터	58	127
2	100100073	113600064	3	113000027	월드컵파크2단지.에스플렉스센터	58	127	113000192	상암DMC홍보관.YTN	58	127
3	100100073	113600711	4	113000192	상암DMC홍보관.YTN	58	127	113000204	누리꿈스퀘어.MBC	58	127
4	100100073	113604856	5	113000204	누리꿈스퀘어.MBC	58	127	113000194	월드컵파크5단지.상암중고등학교입구	58	127
5	100100073	113604943	6	113000194	월드컵파크5단지.상암중고등학교입구	58	127	113000196	상암DMC입구	58	127
6	100100073	113600770	7	113000196	상암DMC입구	58	127	113000021	DMC첨단산업센터	58	127
7	100100073	111602463	8	113000021	DMC첨단산업센터	58	127	111000003	수색교	58	127
8	100100073	111900086	9	111000003	수색교	58	127	111000005	수색역앞	58	127
9	100100073	111600004	10	111000005	수색역앞	58	127	111000007	디지털미디어시티역	58	127

Ⅰ. bus_route 도착지 추가

Ⅱ. Seq값에서 -1 값의 df_route와 , 기존 df_route 의 Seq 기준으로 병합





Ⅲ. 각 출발지, 도착지의gpsX, gpsY값을 nx_x, ny_x(출발지), nx_y, ny_y(도착지)로 칼럼명 수정

Ⅳ. 각 칼럼 분석

3 분석 및 전처리

핵심 프로세스

8월 14일 8시에 발생 기록

```
df_bus_100100073_shift[(df_bus_100100073_shift['baseDateTime'].str.contains('2021081408'))&(df_bus_100100073_shift['plainNo']==df_bus_1001(
```

	baseDateTime	plainNo	sectionId	stopFlag	seq	stationNm_x	stationNm_y	preStation_x	preSeq	prePlainNo	preBaseDateTime	seqDiff
2192	202108140843	서울70사6588	100601959	1	69	광화문	광화문.금호아시아나본관앞	113000022	88.0	서울70사6588	202108131742	0.0
2193	202108140845	서울70사6588	101600128	1	71	서울역사박물관.경교장.강북삼성병원	서대문역사거리.적십자병원	100000385	69.0	서울70사6588	202108140843	2.0
2194	202108140846	서울70사6588	112603437	1	72	서대문역사거리.적십자병원	서대문.서울시교육청	100000028	71.0	서울70사6588	202108140845	1.0
2195	202108140848	서울70사6588	112603900	0	73	서대문.서울시교육청	독립문(가상)	100000368	72.0	서울70사6588	202108140846	1.0
2196	202108140849	서울70사6588	112603446	1	75	독립문공원.극동아파트	이대후문	100000073	73.0	서울70사6588	202108140848	2.0
2197	202108140852	서울70사6588	112602882	1	76	이대후문	세브란스병원앞	112000187	75.0	서울70사6588	202108140849	1.0

bus가 정류장 출발, 도착한 시각

Ⅰ. 시간순으로 나열했을 때 현재 행의 seq값과 그 전 행에 있던 seq값(preSeq)을 뺀 값인 seqDiff값을 저장

Ⅱ. 전 행에 있던 버스차량번호, 시간도 prePlainNo, preBaseDateTime으로 저장

Ⅲ. 문제점 파악

seq를 보면 중간중간에 빠진 정류장 번호가 있는 것을 알 수 있음
(버스위치를 알려주는 센서가 잠시동안 기록을 안보내주거나, 1분이내로 2정거장을 이동한 경우가 있다)

정류장 번호를 빠진 그대로 두면 다음정류장까지 가는데 걸린 시간을 구하기 힘들고, 0분 걸렸다는 정보도 의미를 가지기 때문에 빠진 seq값을 채워야겠다고 판단.

3 분석 및 전처리

핵심 프로세스

8월 14일 8시에 발생 기록

df_bus_100100073_shift[(df_bus_100100073_shift['baseDateTime'].str.contains('2021081408'))&(df_bus_100100073_shift['plainNo']==df_bus_1001(

	baseDateTime	plainNo	sectionId	stopFlag	seq	stationNm_x	stationNm_y	preStation_x	preSeq	prePlainNo	preBaseDateTime	seqDiff
2192	202108140843	서울70사6588	100601959	1	69	광화문	광화문.금호아시아나본관앞	113000022	88.0	서울70사6588	202108131742	0.0
2193	202108140845	서울70사6588	101600128	1	71	서울역사박물관.경교장.강북삼성병원	서대문역사거리.적십자병원	100000385	69.0	서울70사6588	202108140843	2.0
2194	202108140846	서울70사6588	112603437	1	72	서대문역사거리.적십자병원	서대문.서울시교육청	100000028	71.0	서울70사6588	202108140845	1.0
2195	202108140848	서울70사6588	112603900	0	73	서대문.서울시교육청	독립문(가상)	100000368	72.0	서울70사6588	202108140846	1.0
2196	202108140849	서울70사6588	112603446	1	75	독립문공원.극동아파트	이대후문	100000073	73.0	서울70사6588	202108140848	2.0
2197	202108140852	서울70사6588	112602882	1	76	이대후문	세브란스병원앞	112000187	75.0	서울70사6588	202108140849	1.0

73번째 정류장에서 75번째 정류장으로 가는 데이터가 없을 때, 74번째 정류장 도착시각을 75번째 정류장 도착시각으로 채움

만약 n개의 정류장을 스킵했다면, n개의 정류장데이터를 스킵하고 난 다음의 정류장(위의 예시에선 75번째)의 시간값을 넣어준다

데이터중에서 nextPlainNo와 plainNo, nextBaseDateTime과 baseDateMinute 중에서 일자가 같고, 0보다 큰 값만 남겼다(이유 : 시간, 버스차량번호 순으로 정렬하면 무조건 seqDiff값은 0보다 커야함)

3 분석 및 전처리

핵심 프로세스

```
df_bus_100100073_weather.sort_values(['plainNo', 'baseDateMinute']).head(10).drop(columns=['fullSectDist', 'stopFlag', 'busRouteId', 'stati
```

	baseDateMinute	plainNo	gpsX	gpsY	sectionId	congetion	seq	stationNm_x	stationNm_y	interval	nx	ny	PTY	REH	RN1	T1H	WSD
0	202108101042	서울70사6587	126.904464	37.575563	112602628	3	12	북가좌동삼거리	DMC래미안e.편한세상요진아파트	2	59	127	0	62	0	28.9	1.1
1	202108101044	서울70사6587	126.909702	37.571645	112602629	3	13	DMC래미안e.편한세상요진아파트	모래내시장.가좌역	2	59	127	0	62	0	28.9	1.1
2	202108101046	서울70사6587	126.914576	37.569268	112602300	3	14	모래내시장.가좌역	사천교	2	59	127	0	62	0	28.9	1.1
3	202108101048	서울70사6587	126.917966	37.567651	112603411	3	15	사천교	연희104고지앞.구성산회관	3	59	127	0	62	0	28.9	1.1
4	202108101051	서울70사6587	126.925254	37.56629	112602251	3	16	연희104고지앞.구성산회관	서대문우체국	2	59	127	0	62	0	28.9	1.1
5	202108101053	서울70사6587	126.931595	37.56224	112603415	3	17	서대문우체국	연세대앞	2	59	127	0	62	0	28.9	1.1
6	202108101055	서울70사6587	126.935377	37.559991	112603417	3	18	연세대앞	세브란스병원앞	1	59	127	0	62	0	28.9	1.1
7	202108101056	서울70사6587	126.940853	37.560835	112603422	3	19	세브란스병원앞	이대후문	1	59	127	0	62	0	28.9	1.1
8	202108101057	서울70사6587	126.942896	37.562349	112603553	3	20	이대후문	영천시장	6	59	127	0	62	0	28.9	1.1
118	202108101103	서울70사6587	126.961941	37.570299	112603555	3	21	영천시장	금화초등학교.서울시교육청	2	60	127	0	60	0	29.7	3.6

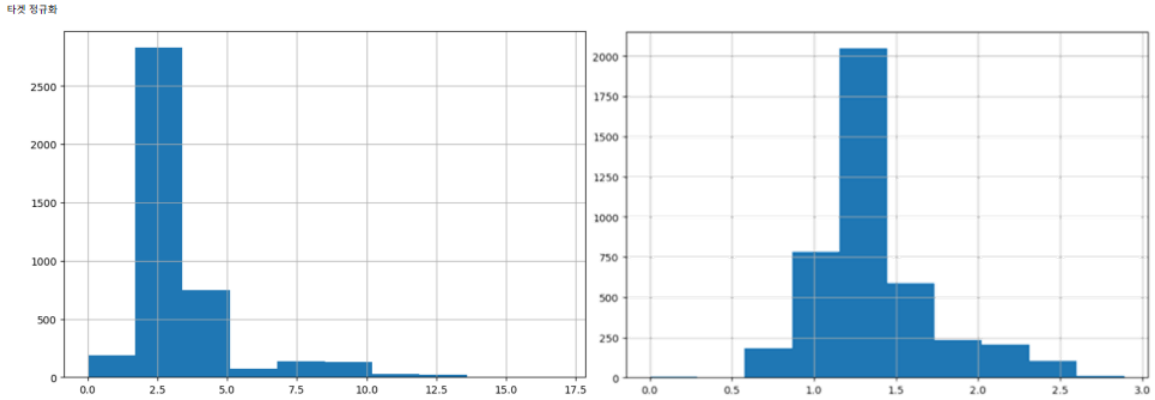
Bus_Bus_route 와 weather 데이터 병합

- bus에서 baseDateMinute값에서 '시' 까지만 baseDateHour 생성
- bus의 경우는 'baseDateHour', 'nx_x', 'ny_x', weather의 경우는 'baseDateHour', 'nx', 'ny' 이용하여 merge
- bus 출발 정류장 nx, ny값이랑 weather의 nx, ny값을 이용하여 merge

'baseDateMinute', 'plainNo', 'fullSectDist', 'gpsX', 'gpsY', 'sectionId', 'stopFlag', 'congetion', 'busRouteId', 'seq', 'station_x', 'stationNm_x', 'nx_x', 'ny_x', 'station_y', 'stationNm_y', 'nx_y', 'ny_y', 'nextBaseDateTime', 'nextPlainNo', 'baseDateHour', 'interval', 'nx', 'ny', 'PTY', 'REH', 'RN1', 'T1H', 'UUU', 'VEC', 'VVV', 'WSD'

3 분석 및 전처리

핵심 프로세스

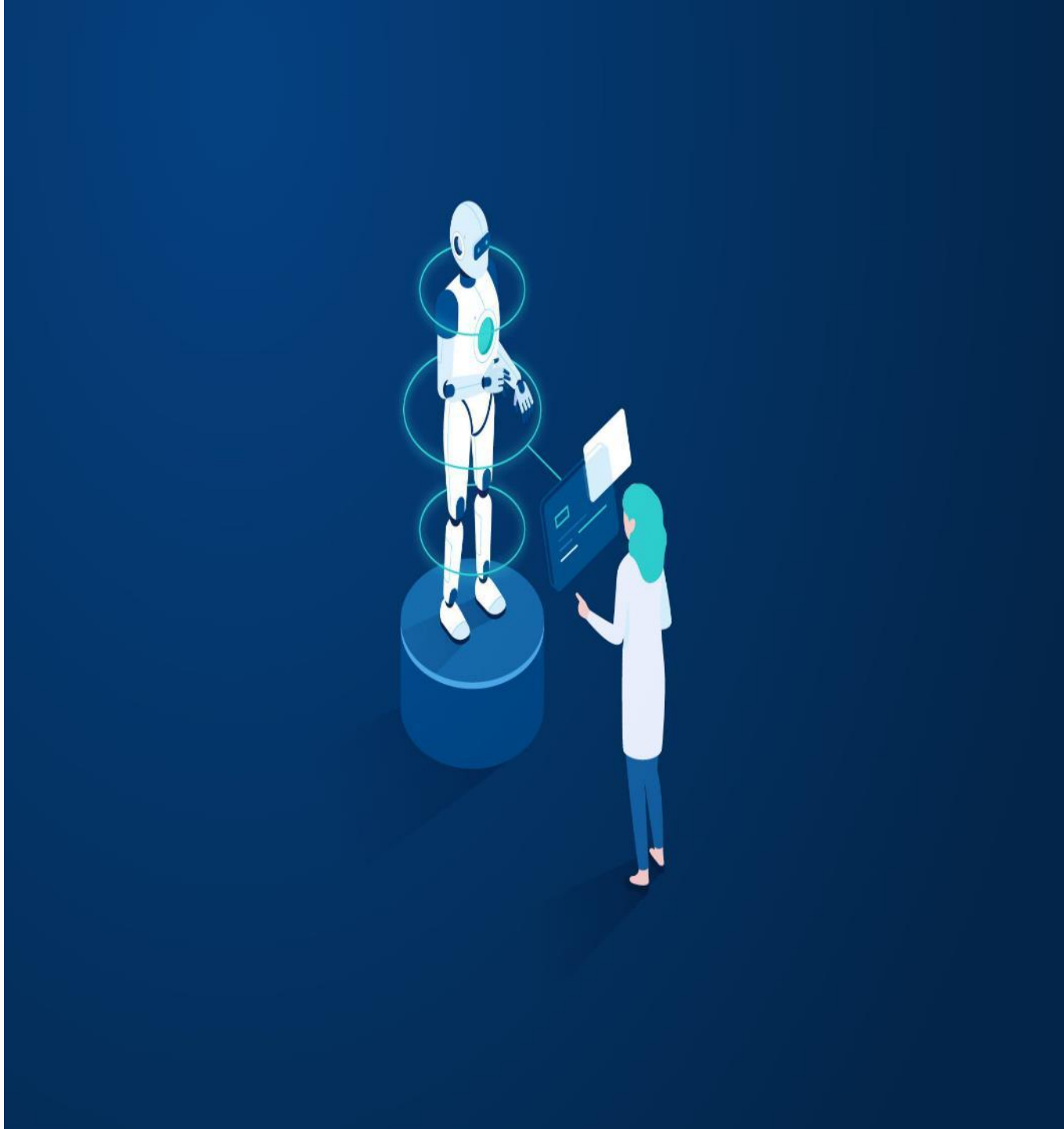



```
df_weather_time_loc.sort_values('baseDateHour')
```

	baseDateHour	nx	ny	PTY	REH	RN1	T1H	UUU	VEC	VVV	WSD
26	2021081008	60	124	0	65	0	27.4	-0.5	159	1.6	1.7
525	2021081008	61	126	0	58	0	28	-0.8	144	1.3	1.6
526	2021081008	61	124	0	67	0	27.5	-1.5	125	1.1	2
527	2021081008	59	127	0	61	0	27.5	-1	151	1.9	2.2
32	2021081008	61	127	0	51	0	28.9	-0.2	170	1.8	1.8
...	...	...	...	...	...	...	...	...	...	...	...
923	2021081813	59	124	0	51	0	29.4	-1.1	31	-1.9	2.3
922	2021081813	59	126	0	59	0	28.2	-2.8	63	-1.4	3.3

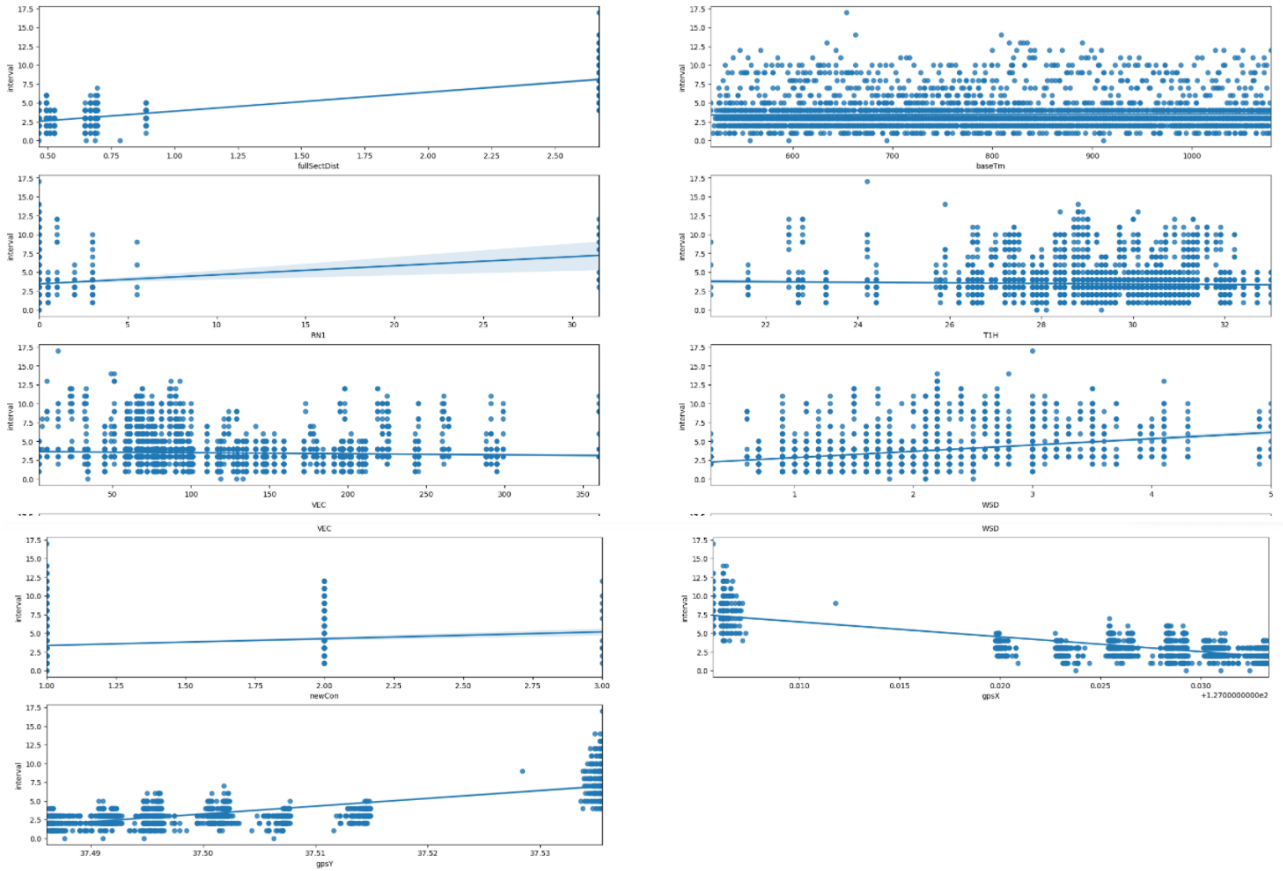
- 왜곡된 값의 Target을 log를 씌어 정규분포화
- weather의 스키마를 기계학습을 위해 적절하게 변경
- 'category', 'obsrValue' => 'PTY', 'REH', 'RN1', 'T1H', 'UUU', 'VEC', 'VVV', 'WSD' 로 스키마 변경

Part 3 머신러닝 모델링



4 머신러닝 모델링

선형회귀 (Linear Regression)



- 각 feature들과의 선형 관계를 시각화.
- fullSectDist / baseTm / RN1 / T1H / VEC / WSD / newCon / gpsX / gpsY 순서
- fullsectdist, RN1, WSD, gpsX, gpsY 등의 feature가 소요시간(interval)과 가장 상관관계가 큰 걸로 보임

4 머신러닝 모델링

선형회귀 (Linear Regression)

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(df_bus_100100073_weather[['fullSectDist','baseTm','RN1','T1H','VEC','WSD','newCon','g

# 선형 회귀를 이용하여 학습 및 예측 수행.
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_preds = lr.predict(X_test)

lr_mse = mean_squared_error(y_test, lr_preds)
lr_rmse = np.sqrt(lr_mse)

print('MSE : {: .3f}, RMSE : {: .3f}' .format(lr_mse, lr_rmse))
print('Variance score : {0: .3f}' .format(r2_score(y_test, lr_preds)))
```

선택 Feature	MSE	RMSE	R2 Score
모두 / interval	1.122	**1.059**	**0.702**
모두 / log interval	**0.051**	**0.225**	0.578
fullSectDist, WSD, RN1 / interval	1.149	1.072	0.695
fullSectDist, WSD, RN1 / log interval	0.052	0.229	0.565
gpsX, gpsY, WSD, RN1 / interval	1.134	1.065	0.699
gpsX, gpsY, WSD, RN1 / log interval	**0.051**	0.226	0.577

- 모든 feature를 모두 사용하였을 때, R2 Score가 가장 높게 나왔다. RMSE 또한 가장 낮게 나와서 예측 정확도가 가장 높다.
- Feature를 선별한다면, fullSecDist 보다 gpsX, gpsY를 feature로 사용하였을 때, MSE가 더 낮고, R2 score가 더 높게 나와서 예측 정확도가 높다.
- 다만 데이터 정규화 작업 후에는 MSE가 낮아졌지만(target scale이 작아졌으므로), R2 score도 더 낮게 나와서 model의 예측 정확도가 더 떨어진다는 결론이 나온다

4 머신러닝 모델링

회귀분석 (RandomForestRegressor)

```
X_train, X_test, y_train, y_test = train_test_split(df_bus_100100073_weather[['fullSectDist','sectionId','baseTm','RN1','T1H','VEC','WSD'

from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

rf_clf = RandomForestRegressor(random_state = 0)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)

rf_mse = mean_squared_error(y_test, rf_pred)
rf_rmse = np.sqrt(rf_mse)

print('MSE : {: .3f}, RMSE : {: .3f}' .format(rf_mse, rf_rmse))
```

```
MSE :  0.873, RMSE :  0.934
```

- test size는 0.3, target은 정규화하지 않은 interval을 그대로 사용했고, 선형 회귀에 비해서 RMSE가 조금 적어졌다

```
from sklearn.model_selection import GridSearchCV

params = {'max_depth': [8, 10, 12],
 'min_samples_leaf': [1, 2, 4, 8],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [50,100]}

rf_clf = RandomForestRegressor(random_state = 0, n_jobs = -1)
grid_cv = GridSearchCV(rf_clf, param_grid = params, scoring = mean_squared_error,cv = 2, n_jobs=-1)
grid_cv.fit(X_train, y_train)

print('최적 하이퍼 파라미터 : {0}' .format(grid_cv.best_params_))
```

```
최적 하이퍼 파라미터 : {'max_depth': 8, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
```

```
# 최적의 하이퍼 파라미터를 사용하여 ramdom forest regression
rf_clf1 = RandomForestRegressor(max_depth=8, min_samples_leaf=1, min_samples_split=2, n_estimators=50)

rf_clf1.fit(X_train,y_train)
rf1_pred = rf_clf1.predict(X_test)

rf1_mse = mean_squared_error(y_test, rf1_pred)
rf1_rmse = np.sqrt(rf1_mse)

print('MSE : {: .3f}, RMSE : {: .3f}' .format(rf1_mse, rf1_rmse))
```

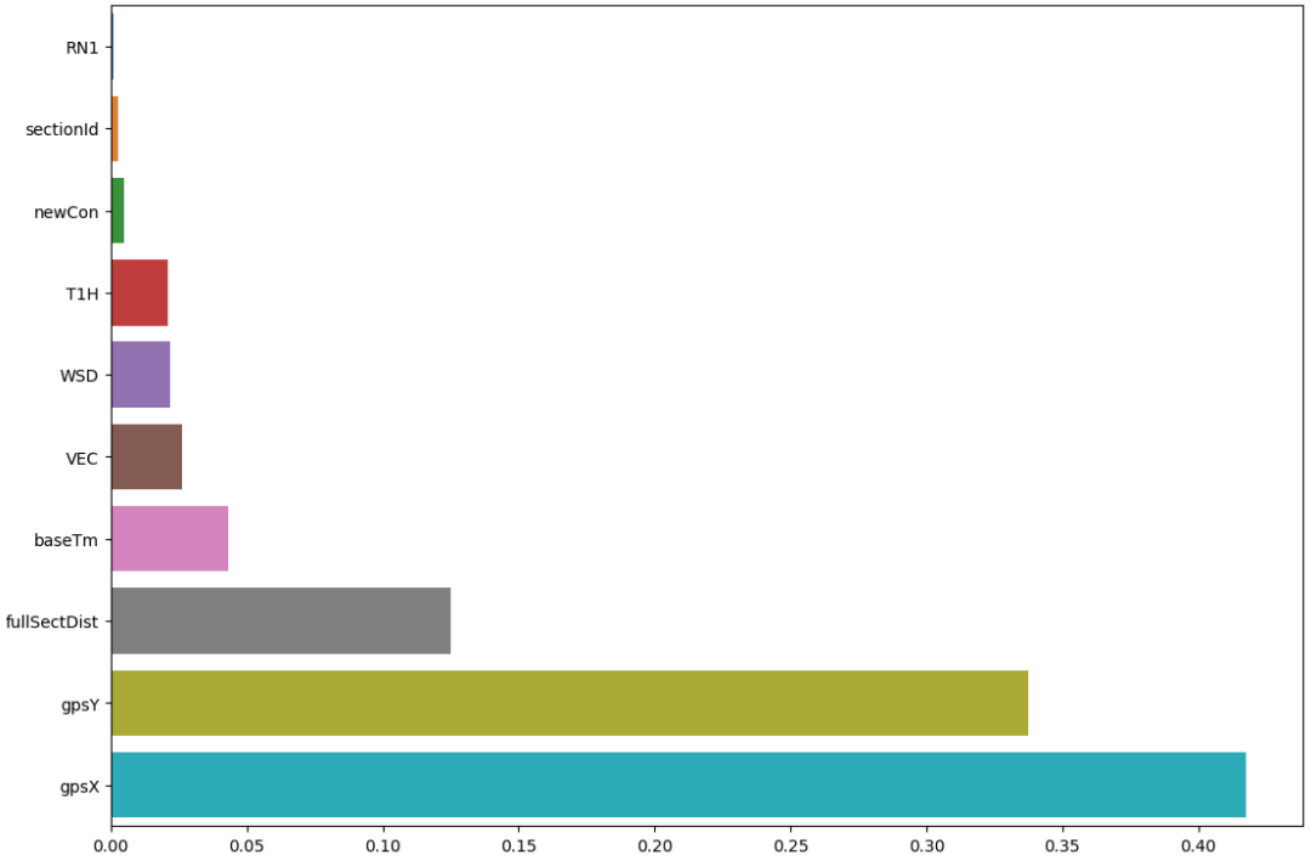
```
MSE :  0.857, RMSE :  0.926
```

- 최적의 하이퍼 파라미터를 적용하여 RMSE를 더 감소

4 머신러닝 모델링

회귀분석 (RandomForestRegressor)

- 마지막 모델의 feature importance를 시각화



분석 summary

- 가장 중요도가 낮은 두가지 feature를 제외하고 모델링을 돌린 결과: MSE : 0.860, RMSE : 0.928

- 결론: Random Forest(모든 feature/best param)
  가장 오차가 낮았다. MSE : 0.857, RMSE : 0.926

- 날씨는 버스 소요시간에 큰 영향을 주지 않는다. 특히 가장 영향을 줄 것이라고 예상한 한시간 이내의 강수량, RN10이 가장 중요도가 낮았다

- 1)버스 운행 데이터는 주중 오전 9~ 오후 6:30,
  2) 버스의 위치는 api에 10초마다/정류장에 출도착할 때마다 갱신되는데, 수집의 한계로 1분 간격으로만 수집,
  3) 하나의 버스 루트에 중에서도 특정 정류장만 확인한 결과

5 결론 요약

< Total summary >

- 최적 옵션 Random Forest(모든 feature/best param)
  MSE : 0.857, RMSE : 0.926

Ⅰ. 프로젝트를 시작할 때의 가정과 다르게, 날씨와 버스도착시간은 상관관계가 적었다. (조사 범위 내)

Ⅰ-2. 강남 거주민 또는 강남에서 버스를 탄다면 버스 도착 예정시간은 날씨와 상관관계가 없음을 알 수 있었다.

Ⅱ. 버스도착시간 예측에 가장 크게 영향을 미치는 변수는 버스의 현재 위치(gpsX,gpsY), FullsectDist 순 이다.

프로젝트 후기

오승아

실시간 데이터를 정제하여 분석을 하는 건 처음이어서 걱정이 되었지만, 팀원분들과 협업을 통해 데이터 전처리를 잘 마무리 할 수 있어서 뿌듯하고 즐거운 경험이었습니다.

모두가 적극적으로 회의에 참석하고 아이디어를 내며, 코드에 대해 빠른 피드백을 주고 받아서 코딩에 더 익숙해졌고 새로운 코드를 많이 배웠습니다. 시간의 한계로 더 많은 데이터를 분석하지 못한 것과 모델링에 시간을 많이 할애하지 못한 점이 아쉽습니다.

김기찬

시간이 짧아 여러 가지 기능을 구현하지 못했지만 아주 기본적인 spark streaming을 적용할 수 있어서 좋았습니다

현실 데이터는 예상치 못한 결측치가 많이 발생하고 그 때마다 데이터를 들여다보면서 전처리하다는 것이 중요하다는 걸 배웠습니다

김하연

평소 버스를 기다릴 때 주로 카카오맵을 썼는데, 이번 프로젝트를 하면서 카카오맵 개발자, 분석가분들이 새삼 대단하다고 느껴졌습니다.
이전에는 데이터를 조금만 전처리하면 바로 분석에 활용할 수 있을거라 생각 했었습니다. 하지만 이번 프로젝트를 통해 전처리에 시간이 많이 걸리며, 또한 중요하다는 점을 깨닫는 계기가 되었습니다. 모두 감사합니다 !


감사합니다