신규 버스 노선
승차인원 예측 서비스

2

주제 선정 배경 및 소개

주제 선정 배경 및 소개

평소 서울시 버스를 이용하면서 **불편함**을 느낌

가까운 거리나 사람들이 많이 이용하는 정류장임에도 불구하고
바로 오갈 수 있는 버스가 없는 경우가 많음

⬇

내가 알고 있는 정류장이 정말 사람들이 많이 이용하는 정류장이 맞을까?
과연 현재 운행되고 있는 버스 노선들의 효용 가치는 높을까?
만약 새로운 노선을 만든다면 어떻게 더 편리하고 좋은 노선을 만들 수 있을까?
또한 현재 노선을 개선하면 더 많은 사람들이 탈 수도 있지 않을까?

2020-11-15 뉴스 기사

서울시, 빅데이터 활용해 시내버스 노선 조정 추진

(서울=연합뉴스) 김계연 기자 = 서울시는 그동안 축적된 빅데이터를 활용해 시내버스 노선 조정을 추진 중이라고 15일 밝혔다.

서울시는 승하차 기록과 지역별 이동수요, 혼잡도 등 교통카드 데이터와 버스운송관리시스템(BMS) 정보를 분석해 효율적인 노선을 찾기로 했다.

검토 대상은 356개 전 노선이다. 서울시는 자치구와 시민, 운수회사, 버스조합 등 의견을 수렴한 뒤 빅데이터 분석해 버스정책심의위원회 심의를 거쳐 내년 1월 노선 조정을 시행할 계획이다.

→ 기사 내용에 나와 있듯이 데이터를 활용하여 더 효율적인 노선을 찾을 수 있음

→ 실제로도 유사하게 추진되고 있으므로 충분한 수요가 있을 것임

이러한 의문점들을 해결하기 위해
서울시 버스 노선과 정류장 데이터들을 활용하여

정류장별 유동 인구 분석
노선별, 시간대별 버스 운영 현황 분석
이용 승객 수와 배차 간격 분석

최종적으로

신규 노선 증설 시 **신규 노선에 대한 승차 인원 예측** 서비스

3

EDA 및 시각화

노선 별 데이터 전 처리

간선버스 노선 별 정류장 데이터

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
사용년월	노선번호	노선명	표준버스정	버스정류장	역명	00시승차	00시하차	1시승차총	1시하차총	2시승차총	2시하차총	3시승차총	3시하차총	4시승차총
202103	100	100번(하계	1E+08	1002	창경궁.서	0	0	0	0	0	0	0	0	118
202103	100	100번(하계	1E+08	1003	명륜3가.성	0	0	0	0	0	0	0	0	63
202103	100	100번(하계	1E+08	1005	혜화동로터	0	0	0	0	0	0	0	0	39
202103	100	100번(하계	1E+08	1198	원남동	0	0	0	0	0	0	0	0	22
202103	100	100번(하계	1E+08	1204	종로5가.효	12	2	0	0	0	0	0	0	0
202103	100	100번(하계	1E+08	1205	종로5가.효	47	13	0	0	0	0	0	0	0
202103	100	100번(하계	1E+08	1212	종로5가	106	16	0	0	0	0	0	0	0
202103	100	100번(하계	1E+08	1219	방송통신대	50	19	0	0	0	0	0	0	0
202103	100	100번(하계	1E+08	1220	혜화역.마로	88	85	0	0	0	0	0	0	0
202103	100	100번(하계	1E+08	1229	혜화역.동성	98	78	0	0	0	0	0	0	0
202103	100	100번(하계	1E+08	1247	광장시장	0	0	0	0	0	0	0	0	11
202103	100	100번(하계	1.01E+08	2004	서울역버스	0	0	0	0	0	0	0	0	18
202103	100	100번(하계	1.01E+08	2007	서울역버스	79	25	0	0	0	0	0	0	0
202103	100	100번(하계	1.01E+08	2127	북창동.남대	0	0	0	0	0	0	0	0	11
202103	100	100번(하계	1.01E+08	2140	롯데백화점	0	0	0	0	0	0	0	0	16
202103	100	100번(하계	1.01E+08	2142	롯데영프라	19	13	0	0	0	0	0	0	0
202103	100	100번(하계	1.01E+08	2156	을지로입구	33	10	0	0	0	0	0	0	0
202103	100	100번(하계	1.01E+08	2158	을지로2가	0	0	0	0	0	0	0	0	3
202103	100	100번(하계	1.01E+08	2159	을지로2가	37	8	0	0	0	0	0	0	0

정류장 별 데이터

표준버스정류장ID	역명	총승차승객수	총하차승객수	환승비율승객수
100000001	종로2가사거리	12313	10857	868.56
100000002	창경궁.서울대학교병원	61432	56189	38208.52
100000003	명륜3가.성대입구	90326	69698	47394.64
100000004	종로2가.삼일교	16021	17499	2799.84
100000005	혜화동로터리.여운형활동터	41239	65340	41817.6
100000018	사직단.어린이도서관	20329	10148	2029.6
100000019	사직동주민센터	63724	34994	6998.8
100000022	경복궁	19581	24943	5986.32
100000023	KT광화문지사	19642	6358	762.96
100000025	광화문	17452	12651	1012.08
100000028	서울역사박물관.경교장.강북	23154	23759	5702.16

노선 별 데이터

노선번호	지나는 지	지나는 지	노선별 겹	환승 비율	강남구	강동구	강북구	강서구	관악구	광진구
100	335891	1343564	62	1801764	0	0	0	0	0	0
101	265855	1063418	46	1728763	0	0	1	0	0	0
102	351941	1407765	31	1504571	0	0	1	0	0	0
103	271785	1087140	58	1463297	0	0	0	0	0	0
104	260527	1042107	40	943816	0	0	1	0	0	0
105	312375	1561877	44	808214	0	0	0	0	0	0
106	308038	1232151	38	2076335	0	0	1	0	0	0
107	308038	1232151	29	2007358	0	0	1	0	0	0
108	308038	1232151	29	2017017	0	0	1	0	0	0
109	302800	908399	30	816308	0	0	1	0	0	0
120	368134	1104402	38	886484	0	0	1	0	0	0
121	350706	1402823	27	294940	0	0	1	0	0	0
130	364512	2551581	24	1188158	0	1	1	0	0	1

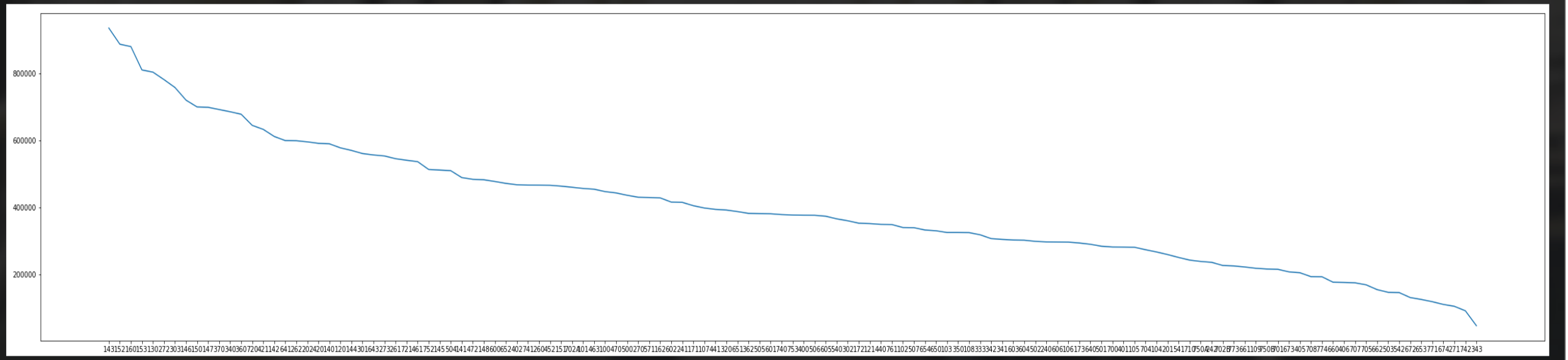
노선 별 데이터 전 처리

최종 데이터 셋

노선번호	강남구	강동구	강북구	강서구	관악구	광진구	구로구	금천구	노원구	도봉구	동대문구	동작구	마포구	서대문구	서초구	성동구	성북구	송파구	양천구	영등포구	용산구	은평구	종로구	중구	중랑구	총승차승객	지나는 지	지나는 지	노선별 겹	환승 비율	총 승
100	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	446729	335891	1343564	62	1801764	
101	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	456114	265855	1063418	46	1728763	
102	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	339532	351941	1407765	31	1504571	
103	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	324650	271785	1087140	58	1463297	
104	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	266534	260527	1042107	40	943816	
105	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	280664	312375	1561877	44	808214	
106	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	295944	308038	1232151	38	2076335	
107	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	397729	308038	1232151	29	2007358	
108	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	324307	308038	1232151	29	2017017	
109	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	217859	302800	908399	30	816308	
120	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	577299	368134	1104402	38	886484	
121	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	351233	350706	1402823	27	294940	
130	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	803258	364512	2551581	24	1188158	
140	1	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	589319	321730	2573836	60	2902959	
141	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	488600	384459	2691216	28	1219145	
142	1	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	611254	345324	2417269	50	1917295	
143	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	934812	323758	1942547	73	1565777	
144	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	1	1	0	569610	325332	2602654	58	1139723	
145	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	511034	388472	1942361	33	517100	
146	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	719649	416149	2080743	21	201911	
147	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	698340	431410	2157049	32	414747	
148	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	482044	394577	2367464	42	1189559	
150	0	0	1	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	1	1	0	1	1	0	699421	297385	2973848	67	3193321	
151	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	1	0	463065	280608	1683648	66	2282129	
152	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	1	0	886647	340542	2383791	72	1703057	
153	0	0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	1	0	0	1	0	0	1	0	0	809802	341307	2730456	55	1181528	
160	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	879506	304868	2438940	55	2759650	
162	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	427875	276195	1380976	60	1036683	
171	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	0	404649	286320	1431600	53	1142946	
172	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	0	352419	325638	1953825	59	1253525	
173	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	293327	312347	1249387	50	417644	
201	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	258843	294362	883085	54	1119953	
202	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	595166	351328	1405310	51	774523	
240	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	296543	429410	1717639	25	162489	
241	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	414593	343247	1716237	39	607213	
242	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	235634	430845	1292536	22	88045	
260	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	465952	305898	1529491	51	1396174	
261	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	1	0	544926	315396	1576979	59	1001361	
262	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	598556	305898	1529491	65	1226090	

EDA 및 시각화

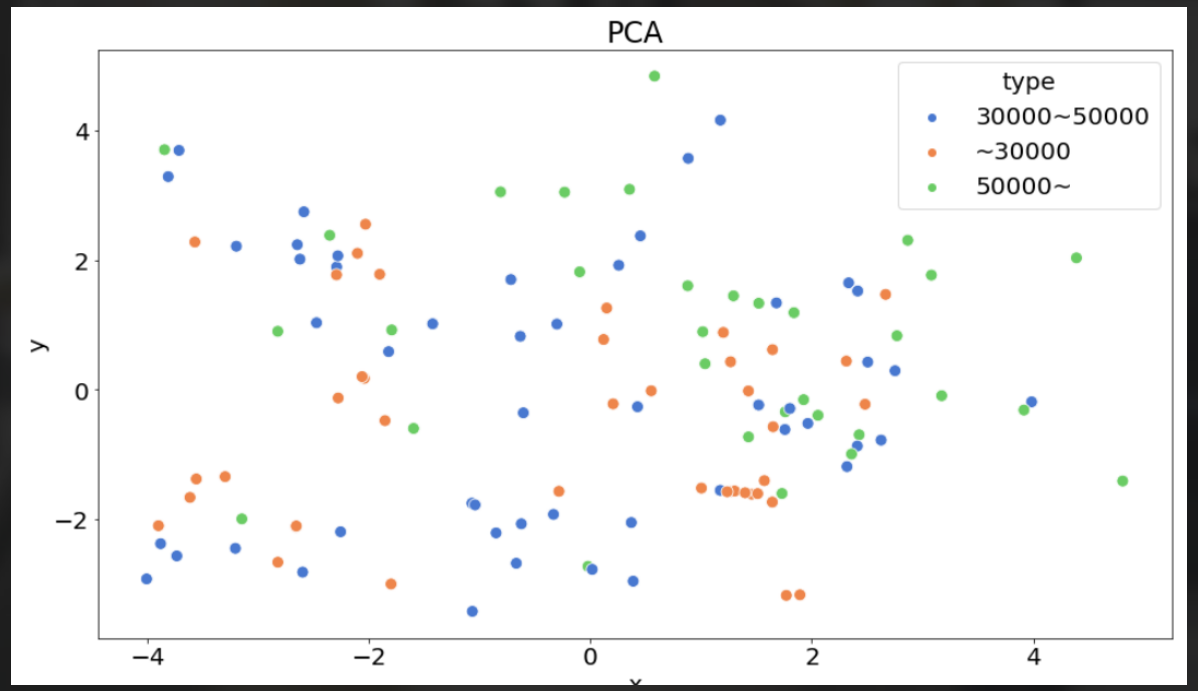
데이터 노선 별 승차인원 수 분포



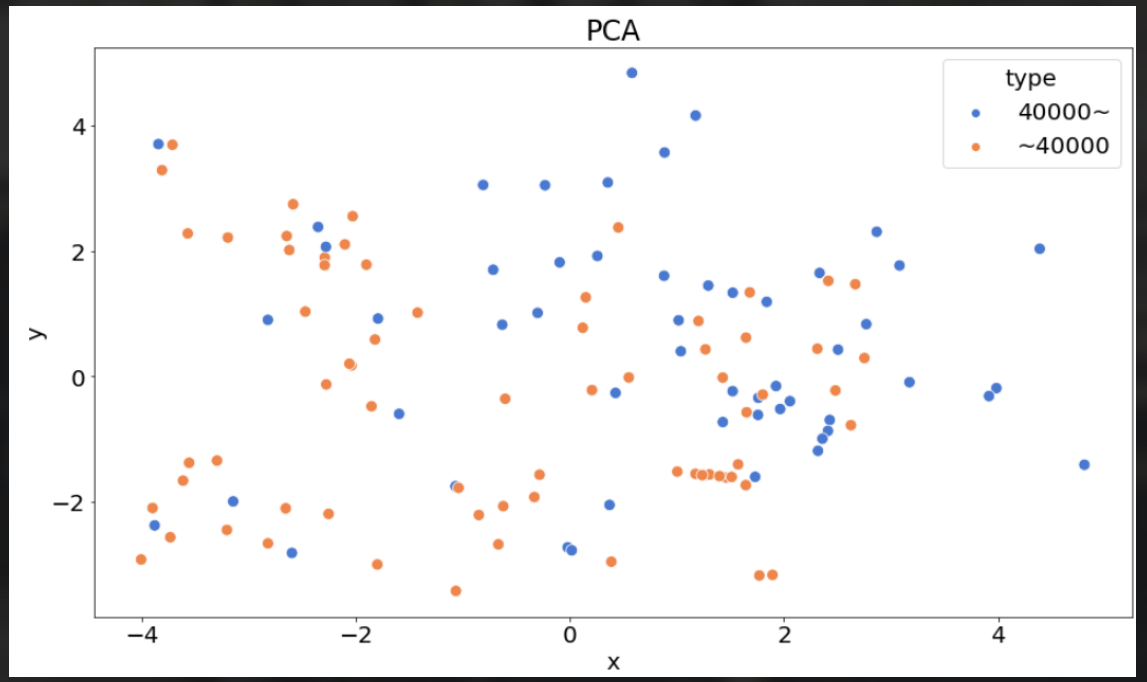
노선별로 전체적으로 고른 분포를 띔

EDA 및 시각화

>> PCA 차원축소 시각화

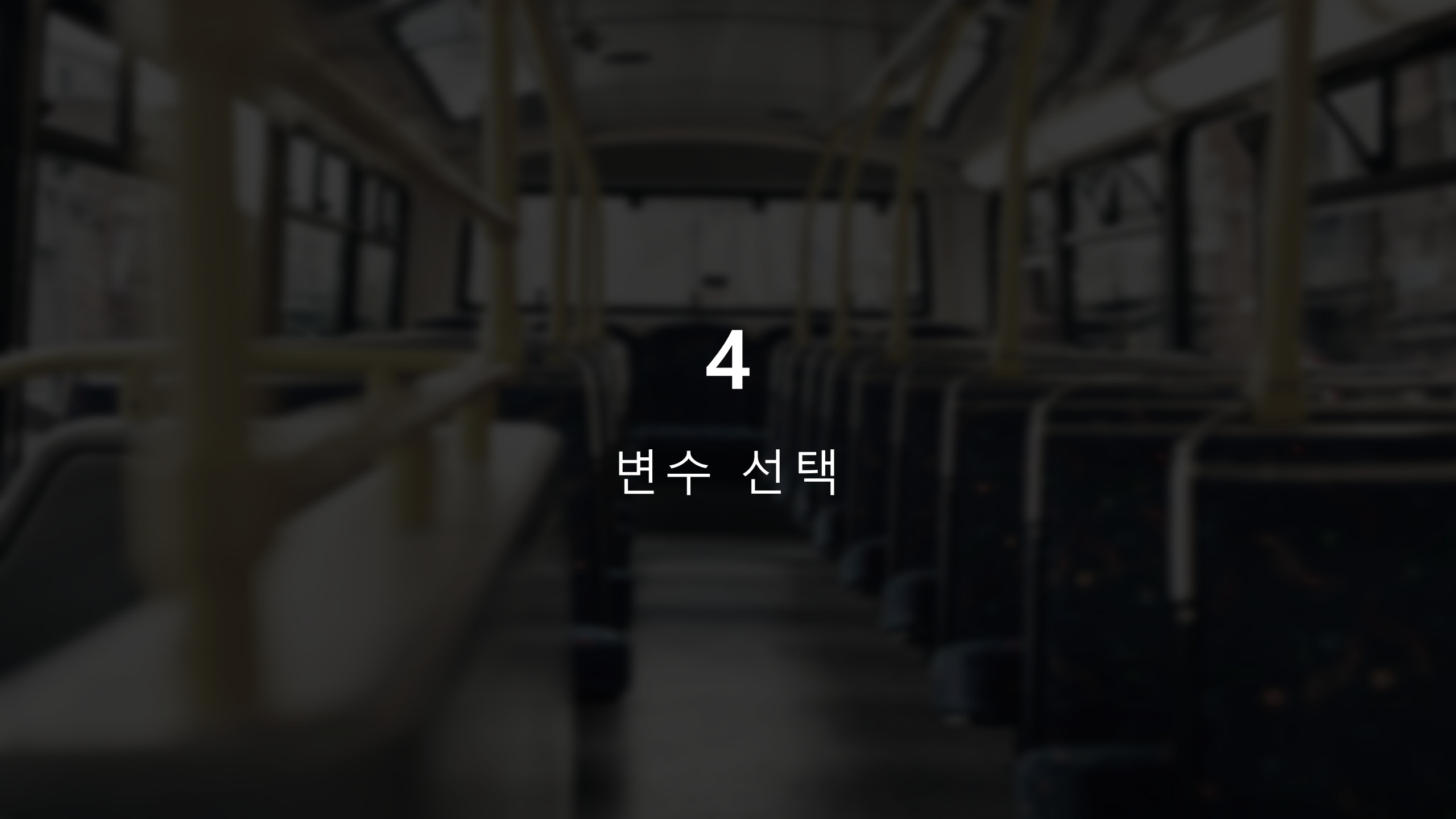


- 3분류



- 2분류

✓ 뚜렷하진 않지만 승객이 많은 쪽은 어느정도 모여 있음



4

변수 선택

Feature Engineering

Feature Construction

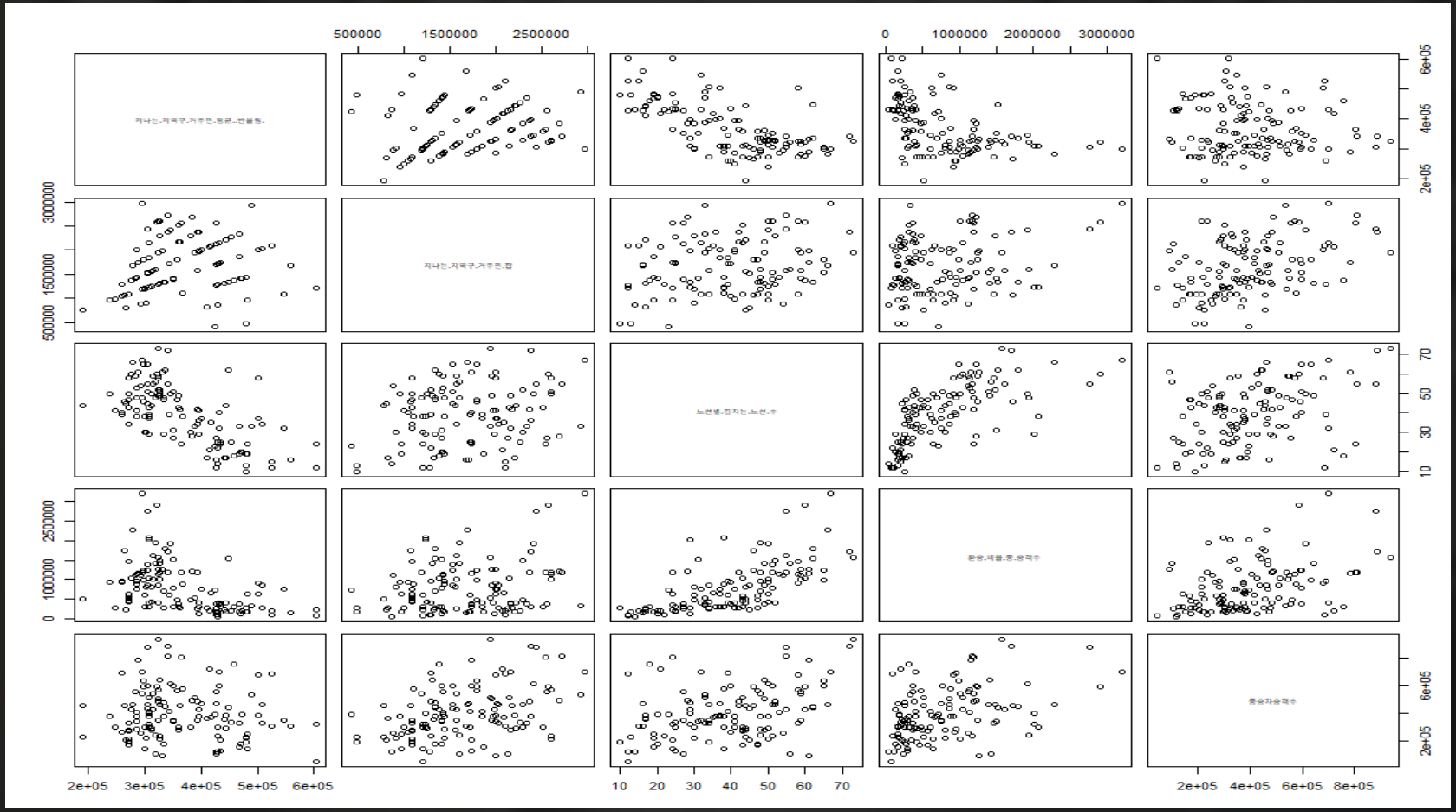
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	노선번호	강남구	강동구	강북구	강서구	관악구	광진구	구로구	금천구	노원구	도봉구	동대문구	동작구	마포
2	100	0	0	0	0	0	0	0	0	1	0	0	0	
3	101	0	0	1	0	0	0	0	0	0	1	0	0	
4	102	0	0	1	0	0	0	0	0	1	0	0	0	
5	103	0	0	0	0	0	0	0	0	0	0	1	0	
6	104	0	0	1	0	0	0	0	0	0	0	0	0	
7	105	0	0	0	0	0	0	0	0	1	0	1	0	
8	106	0	0	1	0	0	0	0	0	0	1	0	0	
9	107	0	0	1	0	0	0	0	0	0	1	0	0	
10	108	0	0	1	0	0	0	0	0	0	1	0	0	
11	109	0	0	1	0	0	0	0	0	0	0	0	0	
12	120	0	0	1	0	0	0	0	0	0	0	1	0	
13	121	0	0	1	0	0	0	0	0	0	0	1	0	
14	130	0	1	1	0	0	1	0	0	0	1	1	0	
15	140	1	0	1	0	0	0	0	0	0	1	0	0	
16	141	1	0	1	0	0	0	0	0	0	1	1	0	
17	142	1	0	1	0	0	0	0	0	0	1	0	0	
18	143	1	0	0	0	0	0	0	0	0	0	0	0	
19	144	1	0	1	0	0	0	0	0	0	0	1	0	
20	145	1	0	1	0	0	0	0	0	0	0	1	0	
21	146	1	0	0	0	0	0	0	0	1	1	0	0	

✓ 기존 노선 데이터에 지나는 지역구 정보를 추가함

*Feature Construction : The manual construction of new features from raw data

변수 상관 관계

✓ 지나는 지역구 정보를 제외한 변수들의 상관 관계 분석 (1. 산점도)



변수 상관 관계

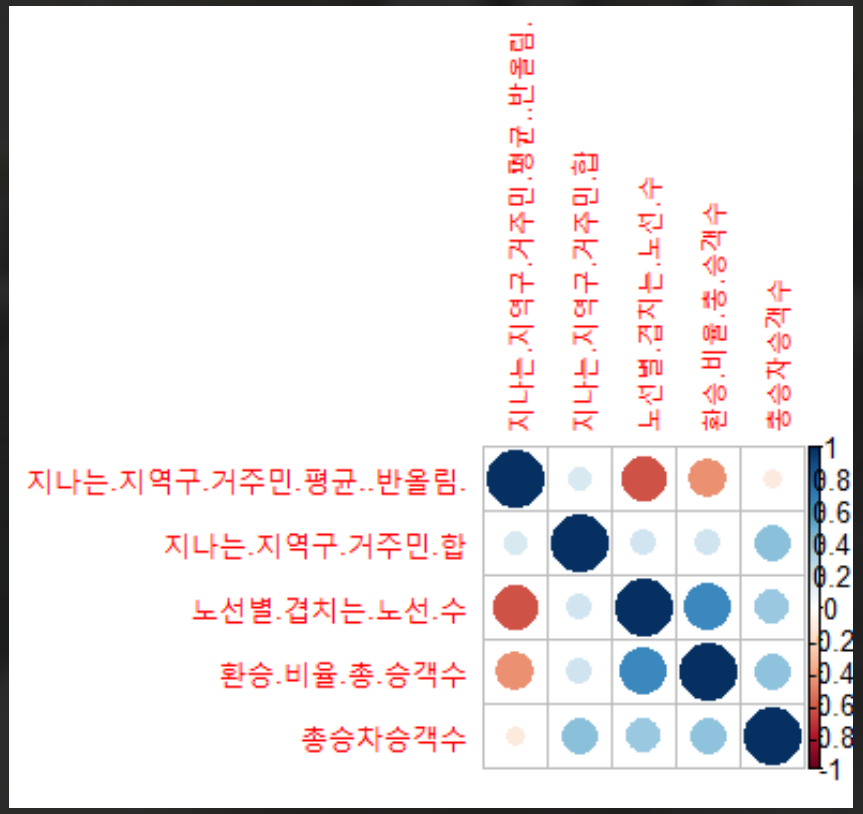
✓ 지나는 지역구 정보를 제외한 변수들의 상관 관계 분석(2. 상관계수)

	지나는.지역구.거주민.평균..반올림.	지나는.지역구.거주민.합	노선별.겹치는.노선.수	환승.비율.총.승객수	총승차승객수
지나는.지역구.거주민.평균..반올림.	1.0000000	0.1673918	-0.6360743	-0.4538564	-0.1122593
지나는.지역구.거주민.합	0.1673918	1.0000000	0.1928420	0.2075092	0.4131082
노선별.겹치는.노선.수	-0.6360743	0.1928420	1.0000000	0.6497570	0.3706107
환승.비율.총.승객수	-0.4538564	0.2075092	0.6497570	1.0000000	0.4042624
총승차승객수	-0.1122593	0.4131082	0.3706107	0.4042624	1.0000000

모두 절댓값 0.1 이상의 유의미한 선형 상관 관계성을 가짐

변수 상관 관계

✓ 지나는 지역구 정보를 제외한 변수들의 상관 관계 분석(3. corrplot)



모델의 종속변수로 쓰일 '총승차승객수'와 다른
모든 변수 간에 유의미한 선형 상관 관계가 있음

해당 모든 변수들을 모델의 독립변수로 사용

5

모델학습

<Linear-regression Model>

>> 선형 회귀 모델 사용

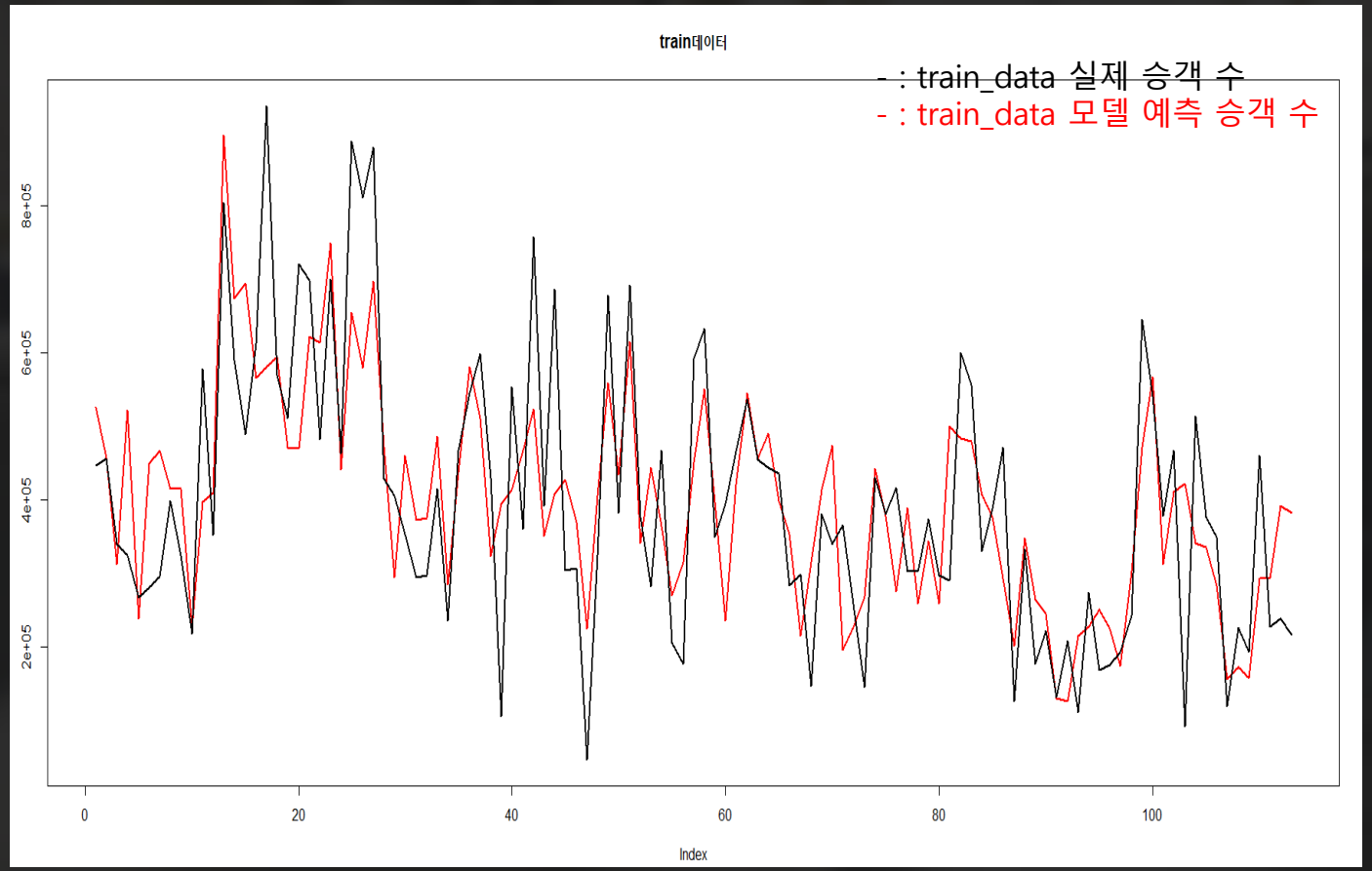
```
linear_model <- lm(총승차승객수 ~ ., data = train_data)
```

>> 예측 결과

```
> mse(test_data$'총승차승객수', predict_linear)
[1] 28053505453
> mae(test_data$'총승차승객수', predict_linear)
[1] 130146.3
> mape(test_data$'총승차승객수', predict_linear)
[1] 0.3314872
```

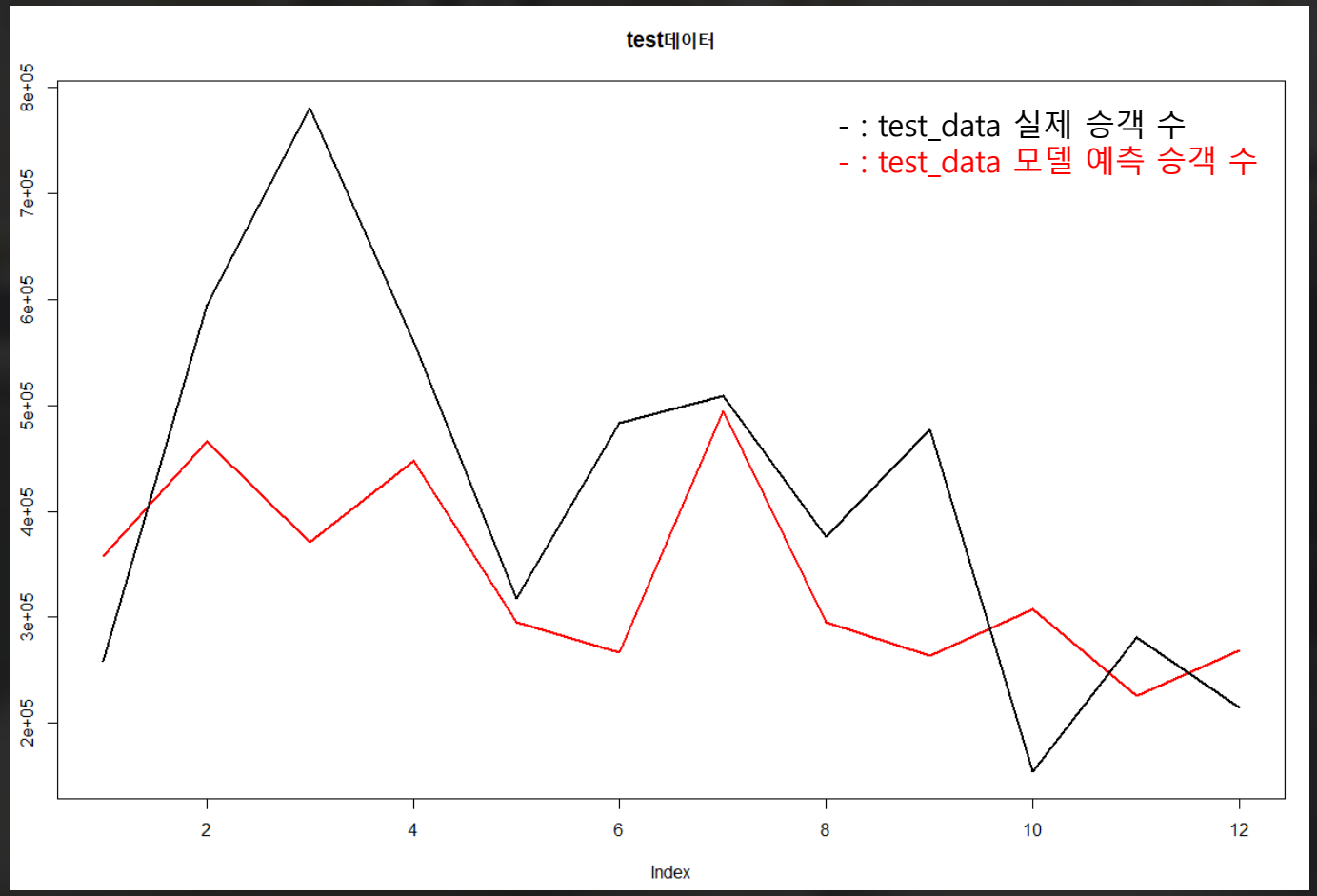
<Linear-regression Model>

>>train data 적합도 시각화



<Linear-regression Model>

>>test data 적합도 시각화



<Robust-regression Model>

>> 로버스트 회귀 모델이란?

- ✓ 기존 선형 회귀 모델은 회귀 계수를 추정할 때 최소 제곱법을 이용
- ✓ 데이터에 아웃라이어가 있을 경우 오차가 제곱이 되기 때문에 전체 추정치가 왜곡되기 쉬움
- ✓ 하지만 로버스트 모델은 잔차의 제곱 대신 절댓값의 합이 최소가 되도록 회귀계수를 추정하기 때문에, 기존 선형 회귀보다 아웃라이어의 영향을 줄일 수 있음

<Robust-regression Model>

>> 로버스트 회귀 모델 사용

```
robust_model <- rlm(총승차승객수 ~ ., data = train_data)
```

>> 예측 결과

```
> mse(test_data$'총승차승객수', predict_robust)
[1] 30129438798
> mae(test_data$'총승차승객수', predict_robust)
[1] 142535.7
> mape(test_data$'총승차승객수', predict_robust)
[1] 0.3661678
```

<Regulation Model>

>> Ridge 모델 사용

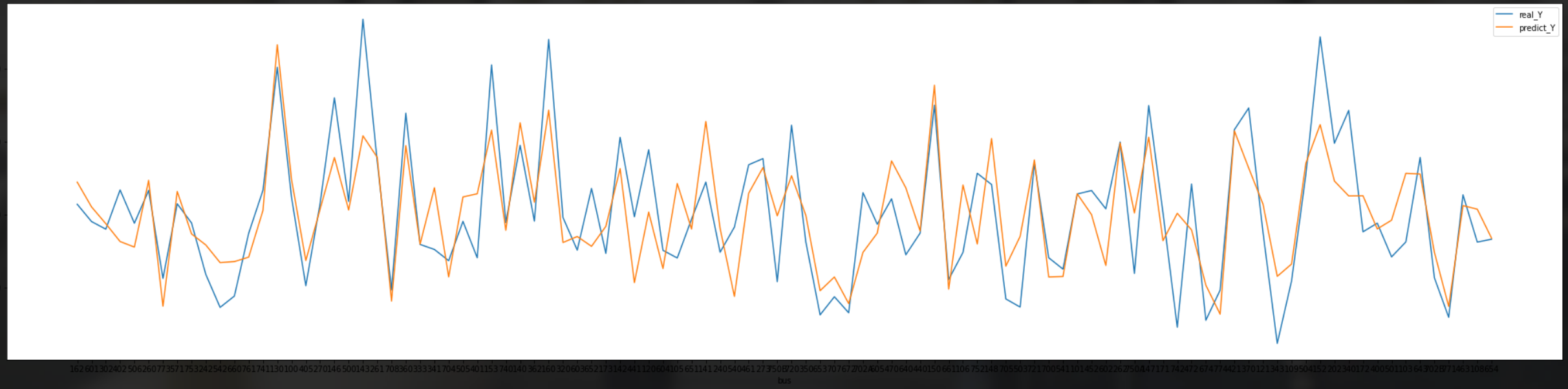
```
model_ridge1 = Ridge(alpha=0.01).fit(X_train, Y_train)
```

>> 예측 결과

```
mse: 22804963066.68262
mae: 108674.11614187964
mape: 0.28725872452495566
```

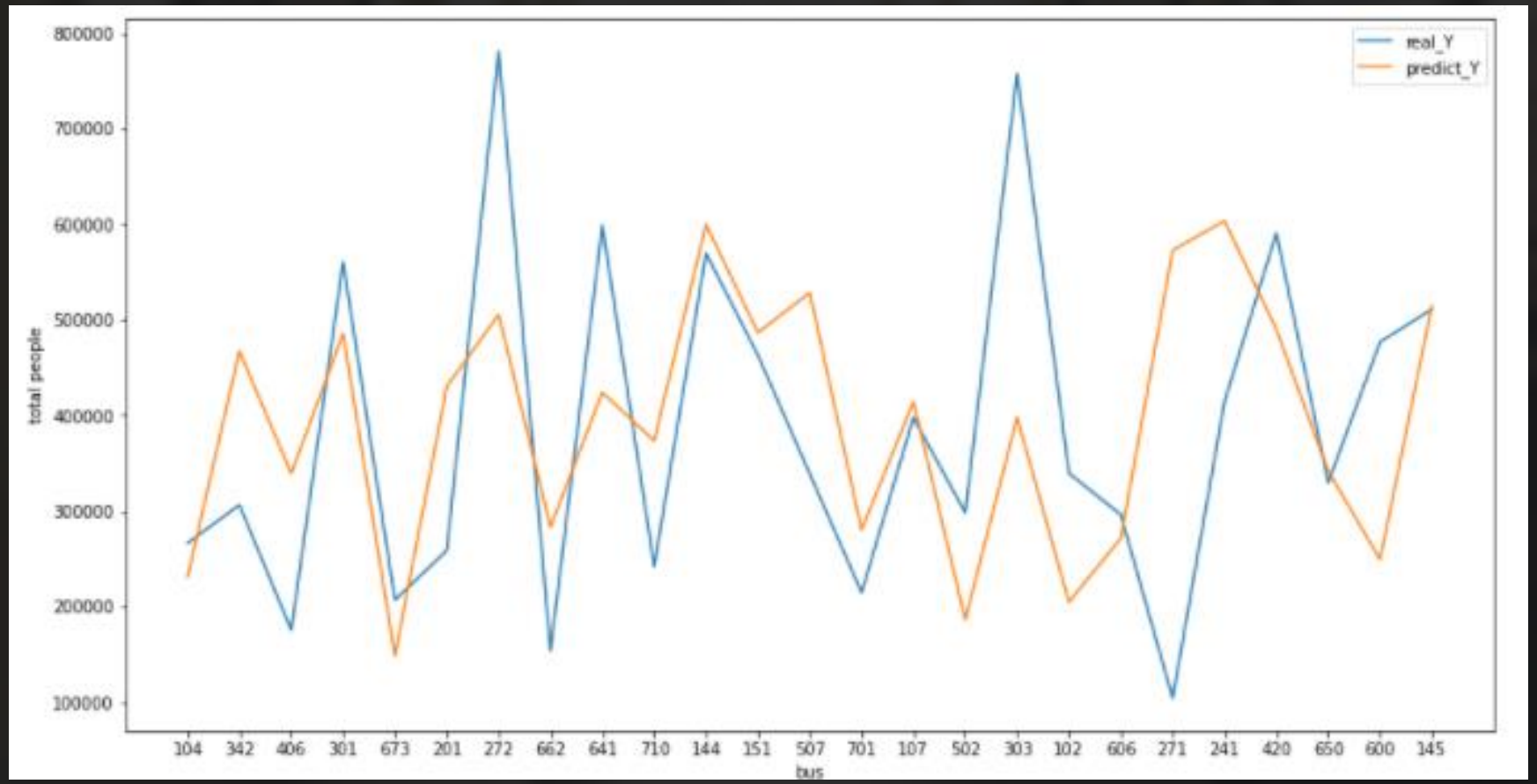
<Regulation Model>

>>train data 적합도 시각화



<Regulation Model>

>>test data 적합도 시각화



<Regulation Model>

>> Rasso 모델 사용

```
model_lasso = Lasso(alpha=0.001, max_iter=100000).fit(X_train, Y_train)
aa= model_lasso.predict(X_test)
```

>> 예측 결과

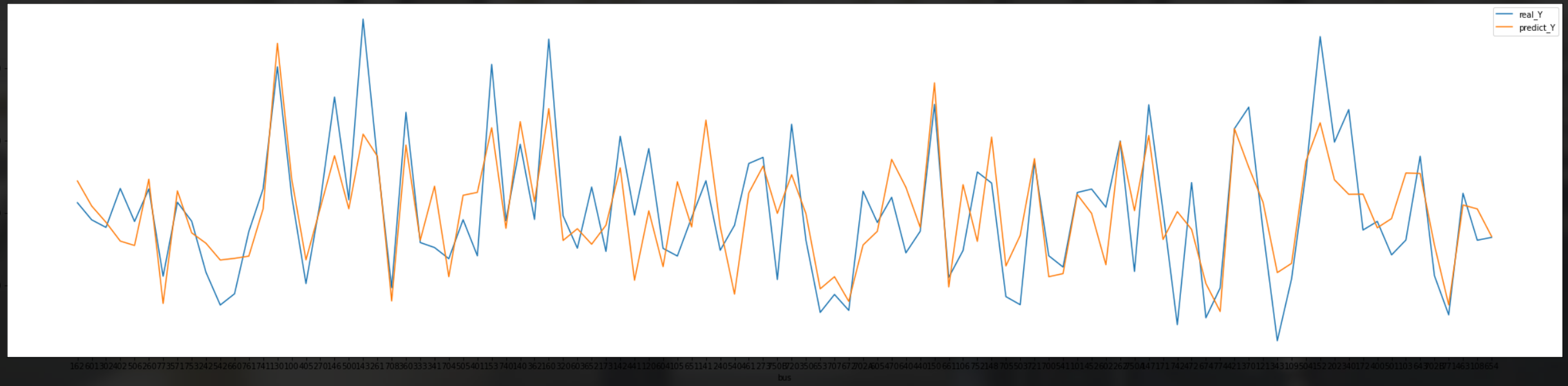
```
사용한 특성의 수 : 29
사용한 max_iter : 6520
```

```
print("mse:",mean_squared_error(aa,Y_test))
print("mae:",mean_absolute_error(aa,Y_test))
print("mape:",mean_absolute_percentage_error(aa,Y_test))
```

```
mse: 26940014085.057983
mae: 130502.69939793549
mape: 0.4073288674324443
```

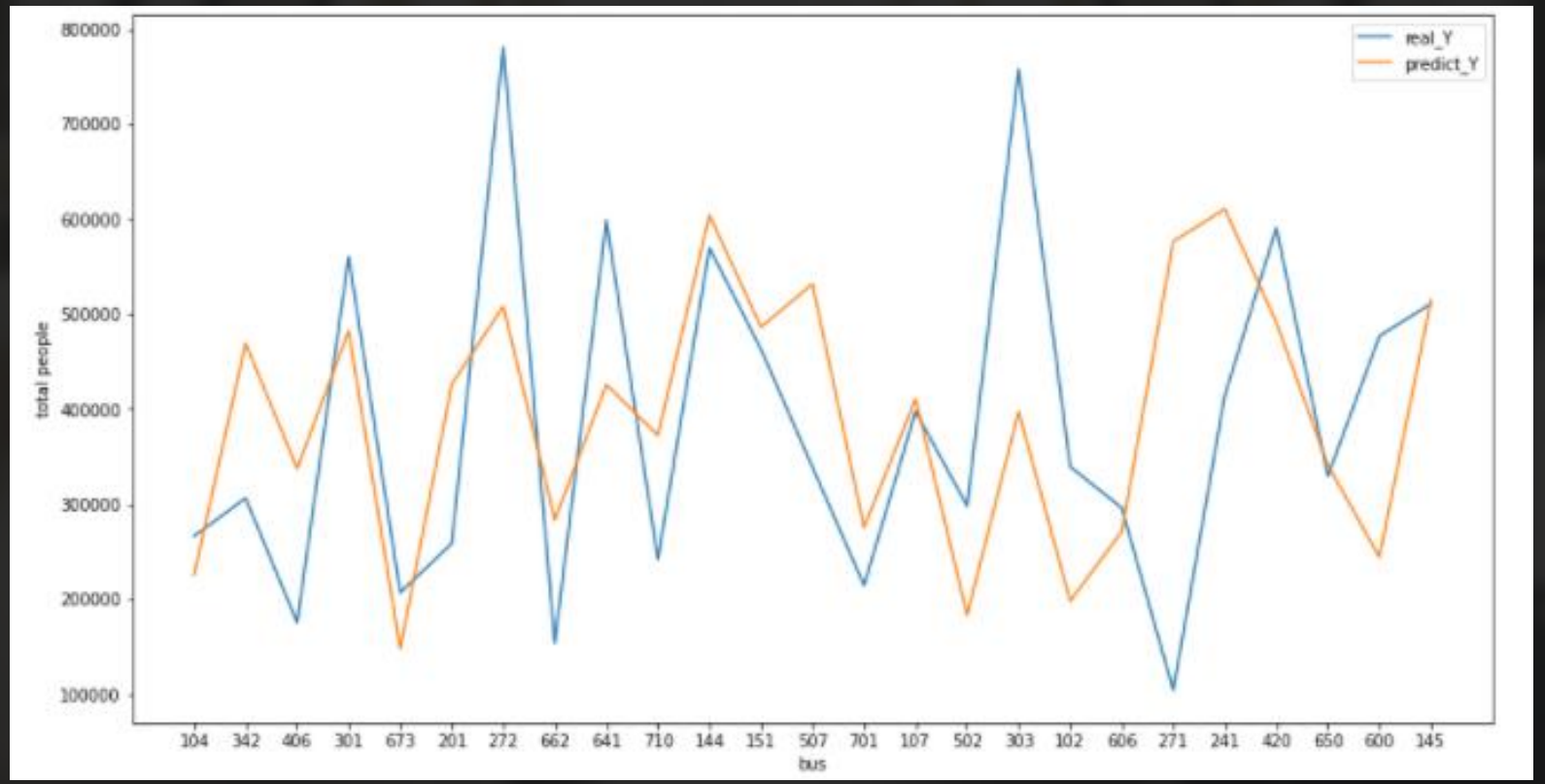
<Regulation Model>

>>train data 적합도 시각화



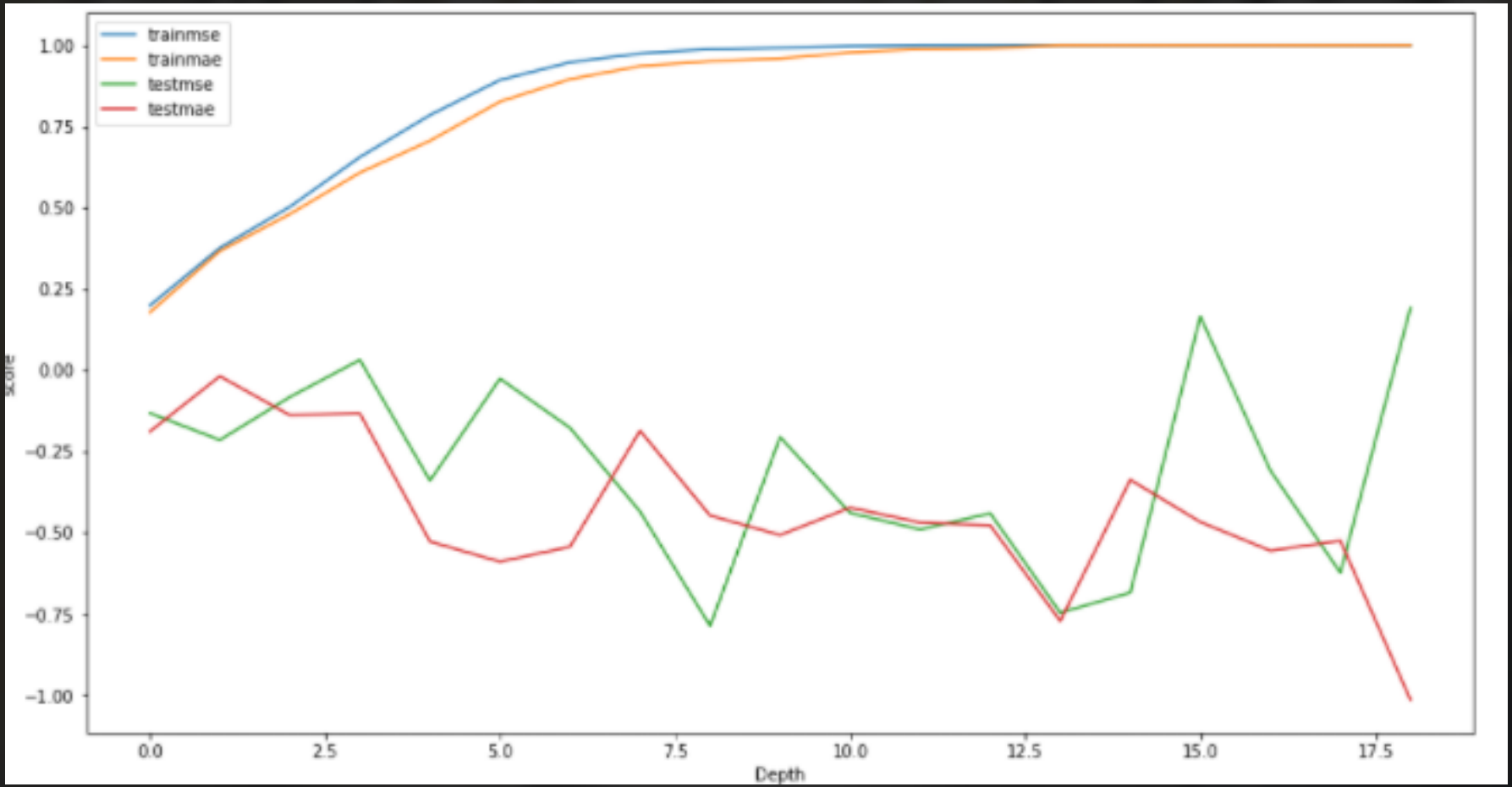
<Regulation Model>

>>test data 적합도 시각화



<Decision Tree>

>> 사전 가지치기(pre-pruning)



<Decision Tree>

>> Decision Tree (Depth=3)

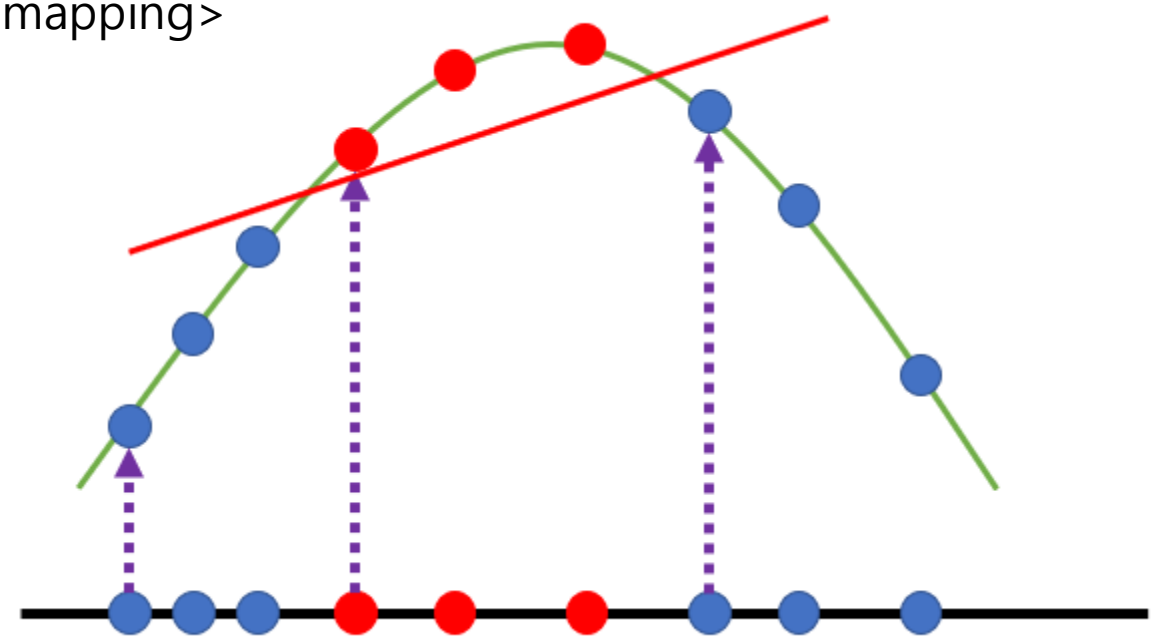
```
tree=DecisionTreeRegressor(max_depth=3)
tree.fit(X_train,Y_train)
result1 = tree1.predict(X_test)
```

>> 예측 결과

```
mse: 22804963066.68262
mae: 108674.11614187964
mape: 0.28725872452495566
```

<Kernel Ridge Regression>

<mapping>



<Kernel function>

(1) Linear Function

$$K(x_i, x_j) = x_i^T x_j$$

(2) Polynomial Function

$$K(x_i, x_j) = (\gamma(x_i^T x_j) + r)^d$$

(3) Radial basis Function(Gaussian)

$$K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$$

✓ Input 변수 X를 커널 함수를 활용하여 Mapping한 변수를 활용하여 파라미터를 추정하는 것

<Kernel Ridge Regression>

>> 예측 결과

```
clf_linear = KernelRidge(kernel ='linear', alpha =0.0)
clf_poly = KernelRidge(kernel = 'polynomial', alpha = 0.0, gamma = 0.7)
clf_rbf = KernelRidge(kernel = 'rbf', alpha =0.0, gamma = 0.00000000000000001)
```

```
<linear Kernel Ridge Regression>
RMSE :  146900.42
MAE: 116942.84 degrees.
MAPE :  27.57349014039713
Accuracy: 72.42651 %.
```

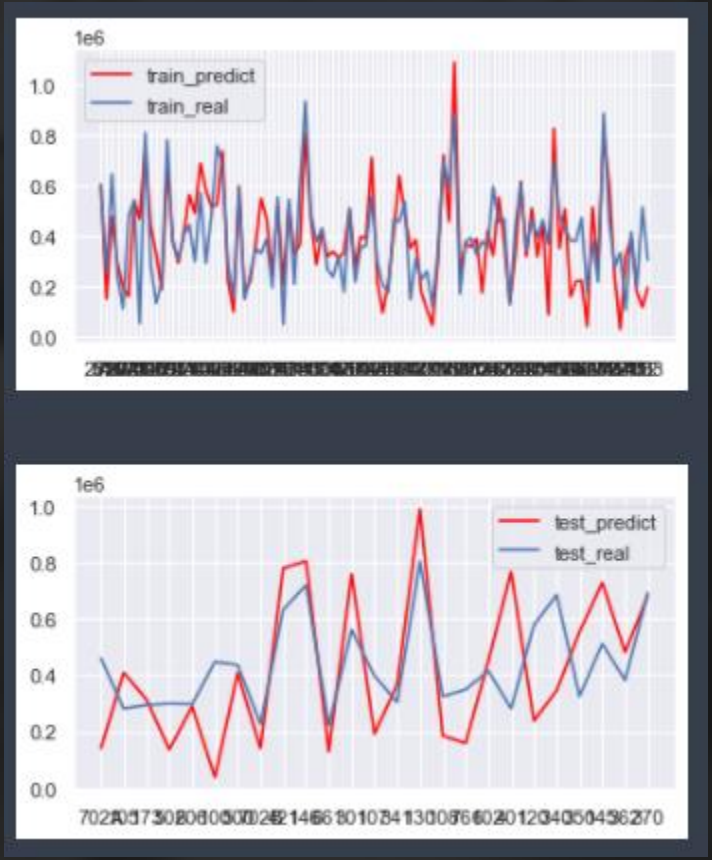
```
<polynomial Kernel Ridge Regression>
RMSE :  211027.87
MAE: 169052.12 degrees.
MAPE :  42.051470248683145
Accuracy: 57.94853 %.
```

```
<rbf Kernel Ridge Regression>
RMSE :  177782.35
MAE: 127169.16 degrees.
MAPE :  24.615007852497982
Accuracy: 75.38499 %.
```

<Kernel Ridge Regression>

>> data 적합도 시각화

<Linear>



<Polynomial>

<Rbf>

<XG Boost Regression>

>> HyperParameter 튜닝

```
xgb_params = {
    "lambda": 0.0030282073258141168,
    "alpha": 0.01563845128469084,
    "colsample_bytree": 0.5,
    "subsample": 0.7,
    "n_estimators": 4000,
    "learning_rate": 0.05,
    "max_depth": 6,
    "random_state": 2020,
}
reg2 = XGBRegressor(**xgb_params)
```

>>학습 진행

```
reg2.fit(
    train_features,
    train_labels,
    eval_set=[(train_features, train_labels), (test_features, test_labels)],
    eval_metric="mae",
    early_stopping_rounds=100,
    verbose=100,
)
```

<XG Boost Regression>

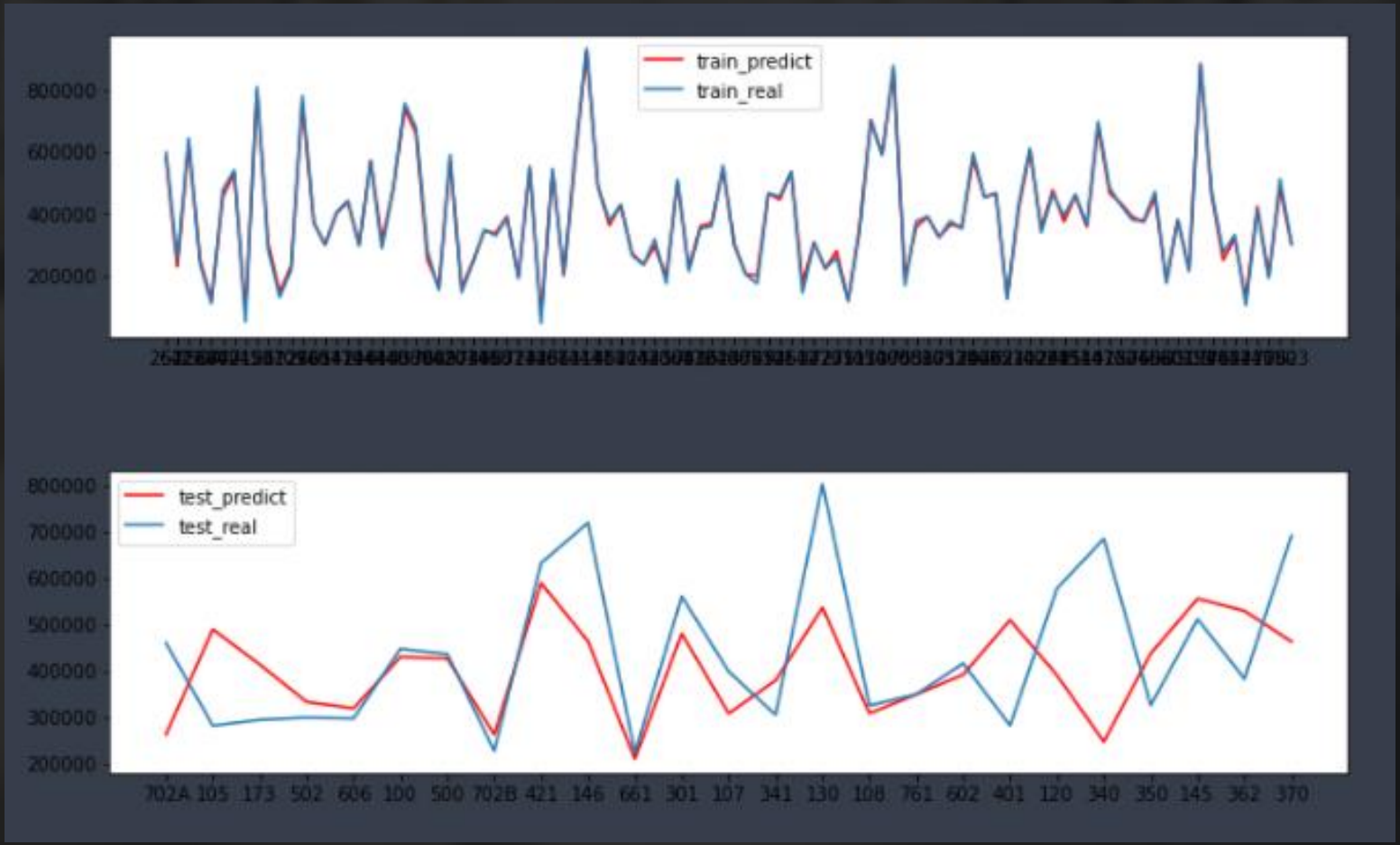
>> 예측 결과

```
predictions = reg2.predict(test_features)
errors = abs(predictions - test_labels)
print("RMSE : ", round(mean_squared_error(test_labels, predictions)**0.5, 2))
print("MAE : ", round(np.mean(errors), 2))
mape = 100 * (errors / test_labels)
print("MAPE : ", np.mean(mape))
accuracy = 100 - np.mean(mape)
print("Accuracy:", round(accuracy, 5), "%.")
```

```
RMSE :  158648.02
MAE :  115954.9
MAPE :  25.826369795237756
Accuracy: 74.17363 %.
```

<XG Boost Regression>

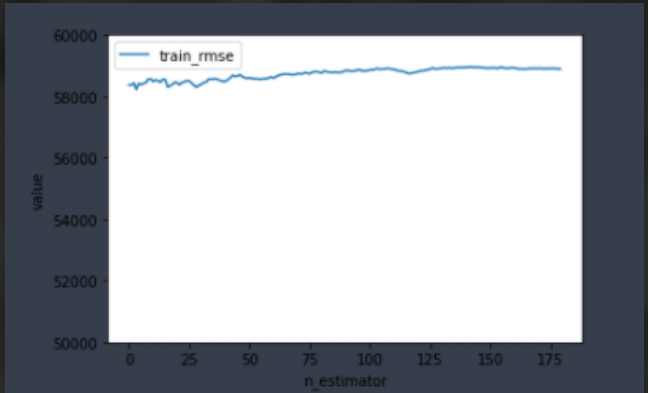
>> data 적합도 시각화

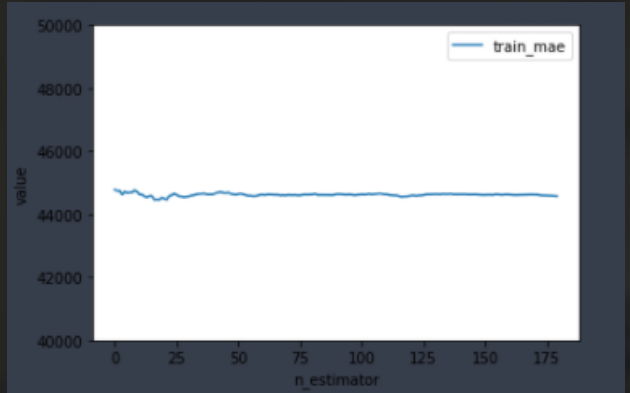


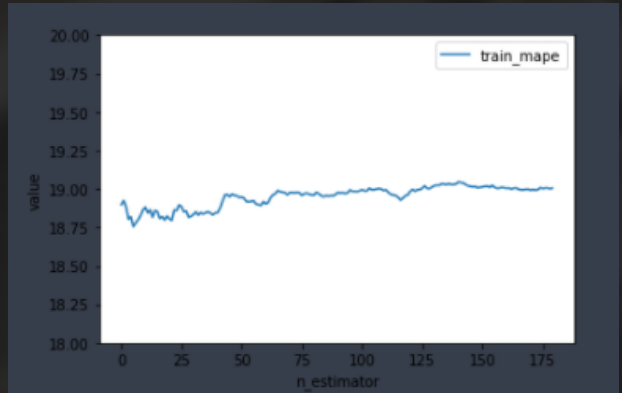
<Random Forest>

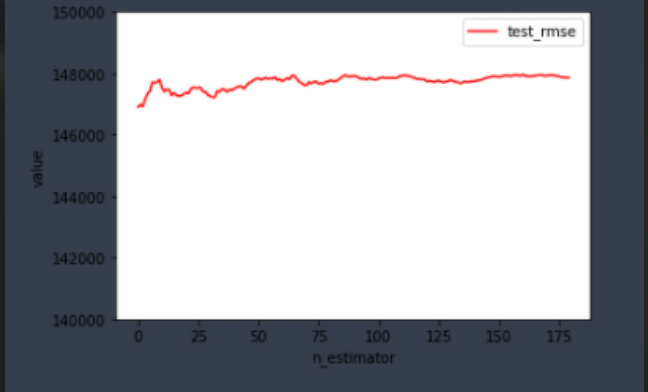
```
for k in tqdm(range(1000,10000,50)):
    dt = RandomForestRegressor(n_estimators=k, random_state=42)
    dt.fit(train_features, train_labels)

    train_predictions = dt.predict(train_features)
    test_predictions = dt.predict(test_features)

    train_rmse.append(round(mean_squared_error(train_labels, train_predictions)**0.5, 2))
    test_rmse.append(round(mean_squared_error(test_labels, test_predictions)**0.5, 2))


    train_errors = abs(train_predictions - train_labels)
    train_mae.append(round(np.mean(train_errors)))

    test_errors = abs(test_predictions - test_labels)
    test_mae.append(round(np.mean(test_errors), 2))

    train__mape = 100 * (train_errors / train_labels)
    train_mape.append(round(np.mean(train__mape), 5))
    test__mape = 100 * (test_errors / test_labels)
    test_mape.append(round(np.mean(test__mape), 5))
    n_estima.append(k)
    print('n_estimators = %d done.' %k)
```

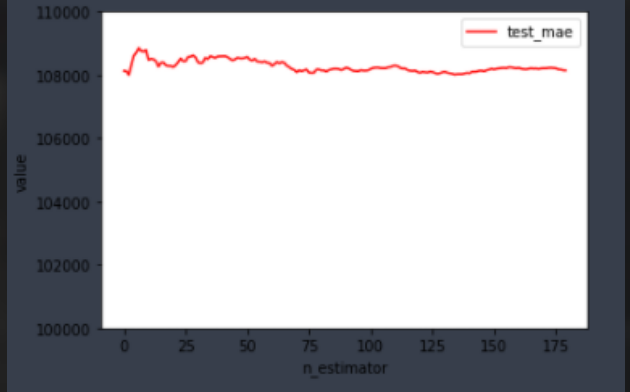
✓ 트리의 개수(n_estimators)를 1000부터 10000까지 50간격으로 학습시키며, rmse, mae, mape 비교

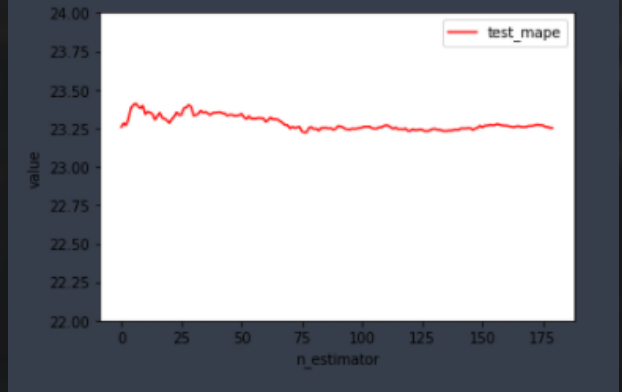












✓ RMSE, MAE, MAPE 모두 큰 변화 없음 >> 트리의 개수, 1200 모델 사용

<Random Forest>

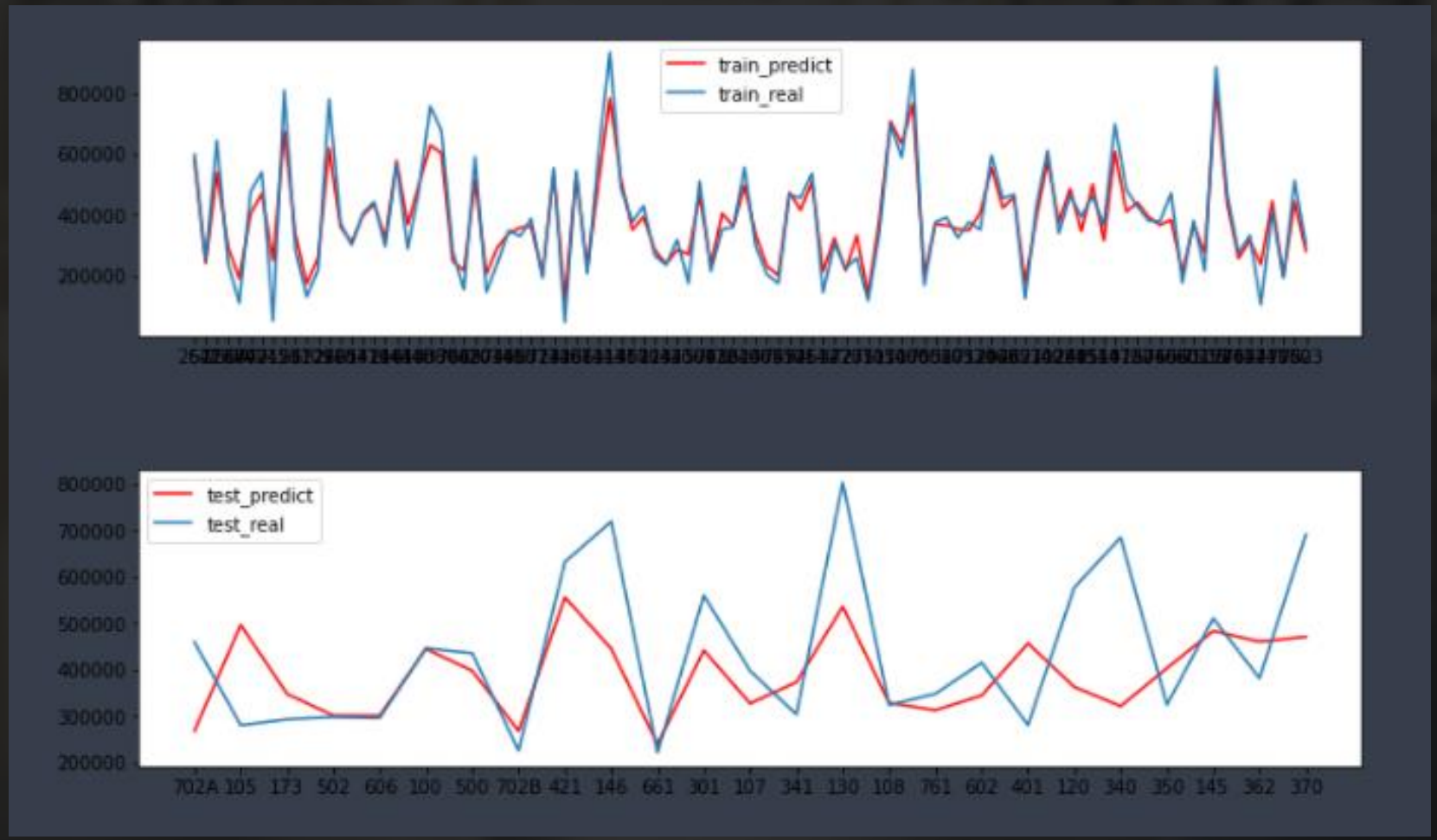
>> 예측 결과

```
predictions = rf1.predict(test_features)
errors = abs(predictions - test_labels)
print("RMSE : ", round(mean_squared_error(test_labels, predictions)**0.5, 2))
print("MAE : ", round(np.mean(errors), 2))
mape = 100 * (errors / test_labels)
print("MAPE : ", np.mean(mape))
accuracy = 100 - np.mean(mape)
print("Accuracy:", round(accuracy, 5), "%.")
```

```
RMSE :  147340.08
MAE :  108607.6
MAPE :  23.379537070466398
Accuracy: 76.62046 %.
```

<Random Forest>

>> data 적합도 시각화



6

시각화

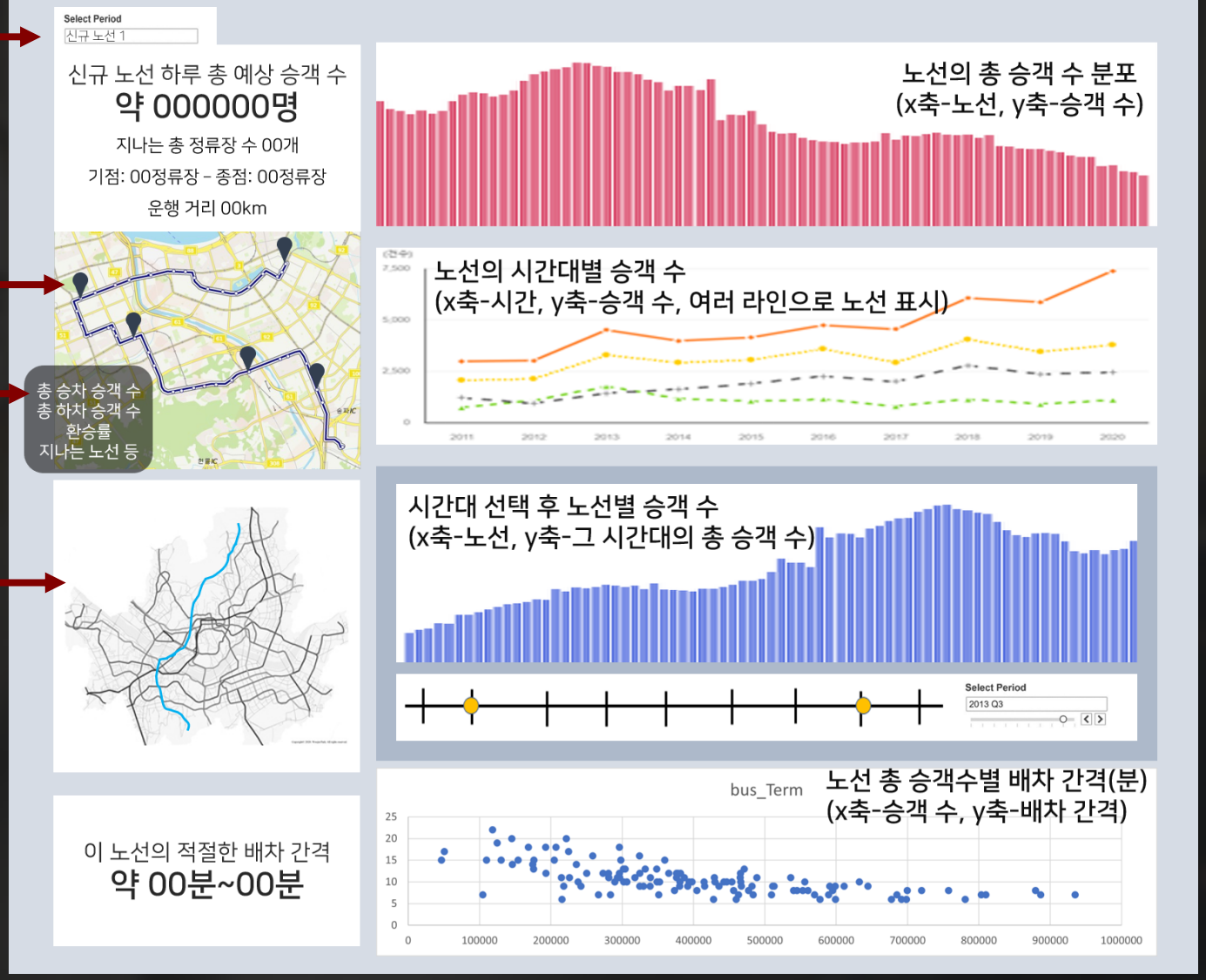
초기 기획 단계

신규 노선들 중 선택

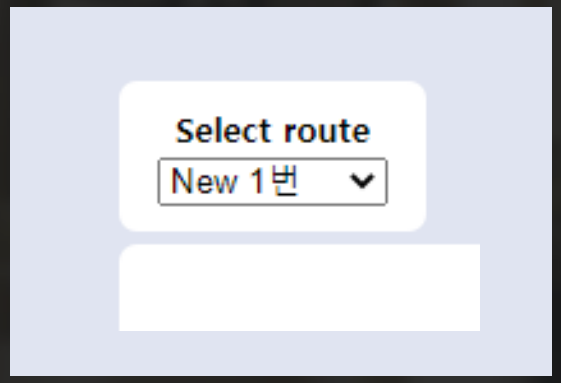
아이콘으로 주요 정류장 표시
(이용승객 많거나 환승률 높은 곳)

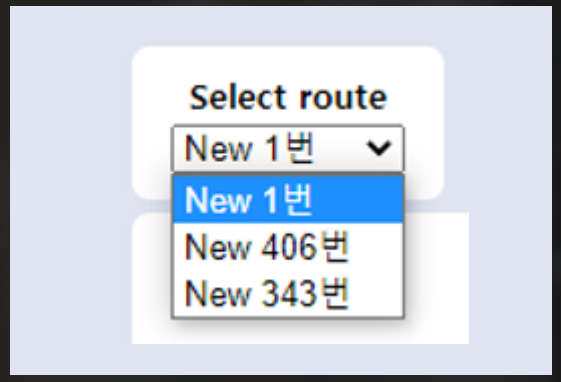
정류장별 정보 툴팁

전체 버스 노선을 지도에 표시
신규 노선을 눈에 띄게 표시



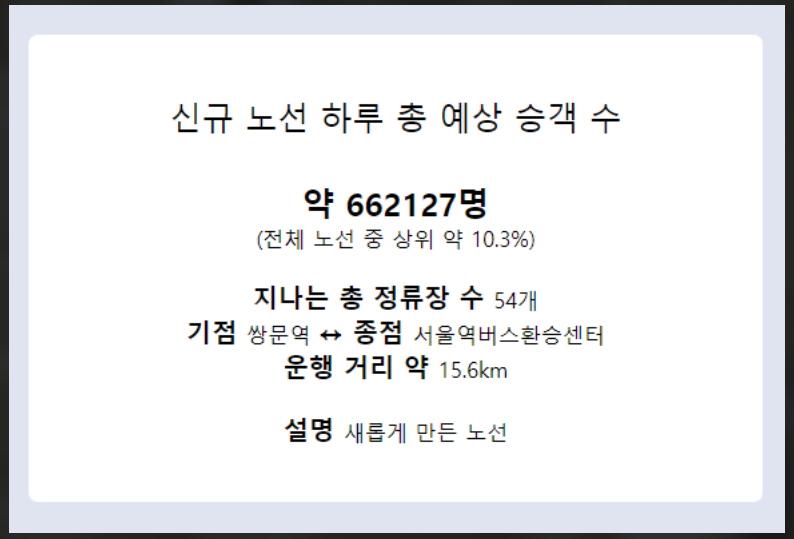
신규 노선 선택

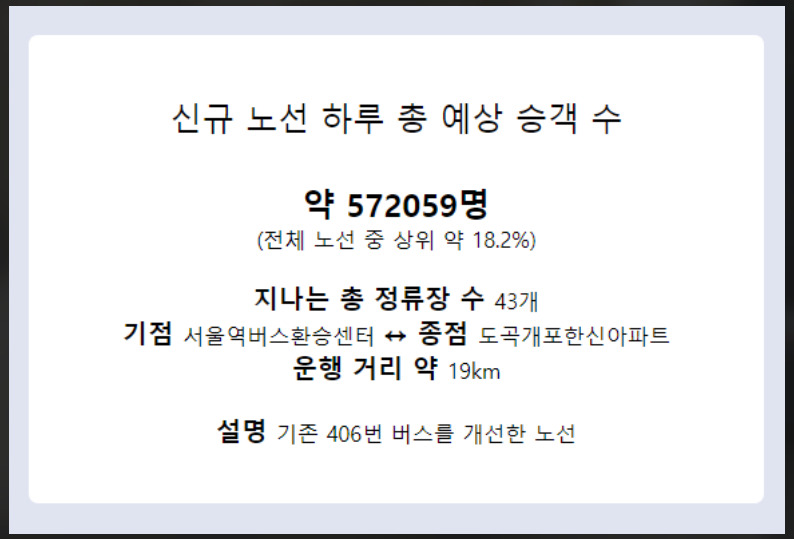




- **콤보 박스(Select Box)**
- 정보를 열람할 노선 선택
- New 1번 / New 406번 / New 343번
- 경로 선택에 따라 해당 경로의 기본 정보와 예측 정보, 경로 시각화, 기존 노선들과의 비교 그래프, 추천 배차 간격 정보 제공

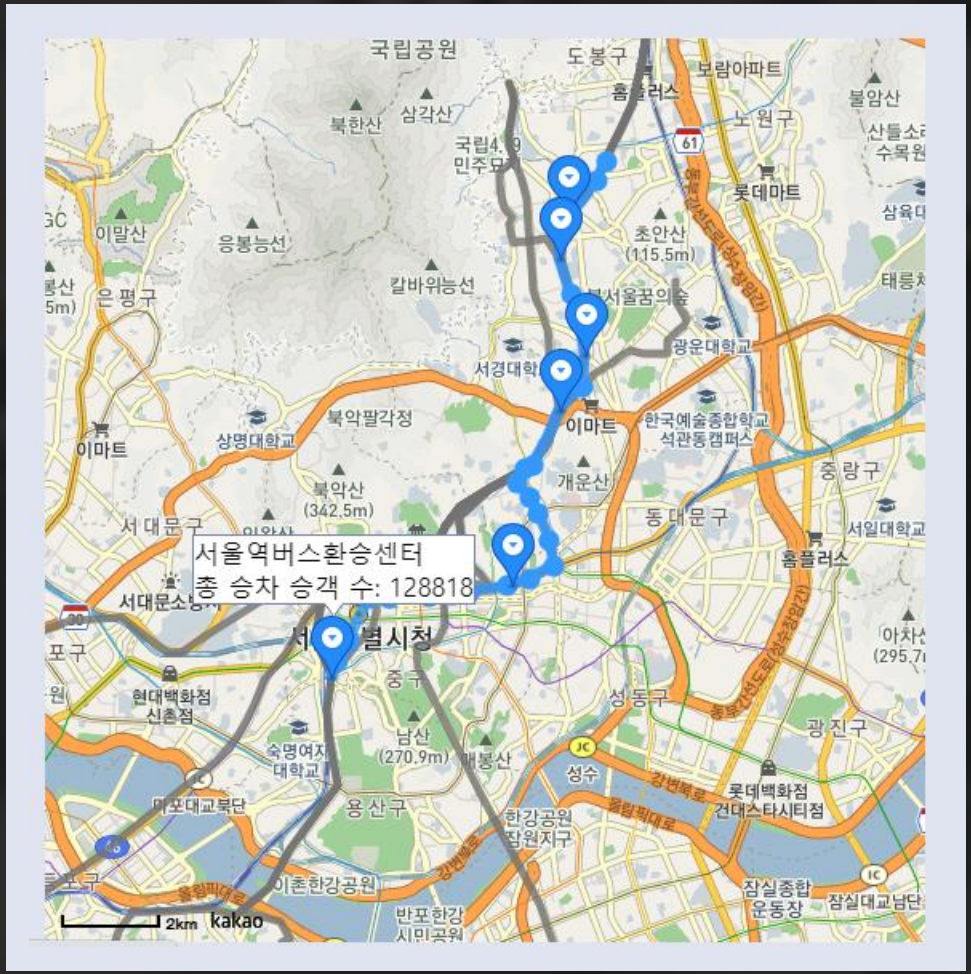
신규 노선 정보





- 콤보 박스에서 선택한 해당 노선의 정보를 **텍스트**로 제공
  - 하루 총 예상 승객 수
  - 전체 버스 노선과 비교(상위 %)
  - 지나는 정류장 수
  - 기점 / 종점
  - 운행 거리(km)
  - 노선 설명
- 주요 정보이므로 가장 눈에 띄는 상단에 배치

노선 경로 시각화

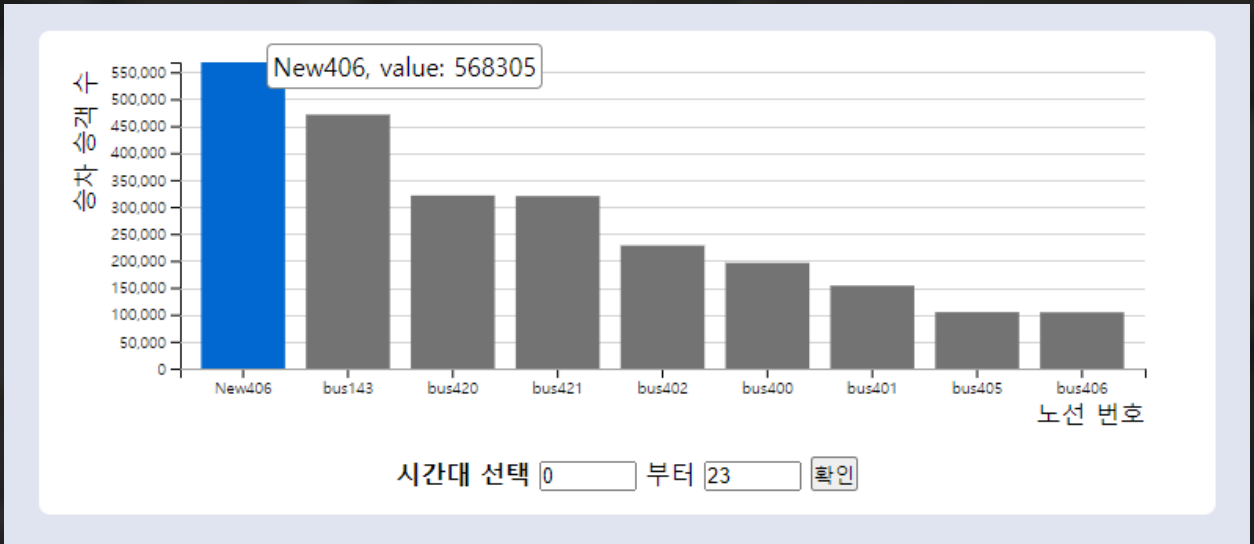


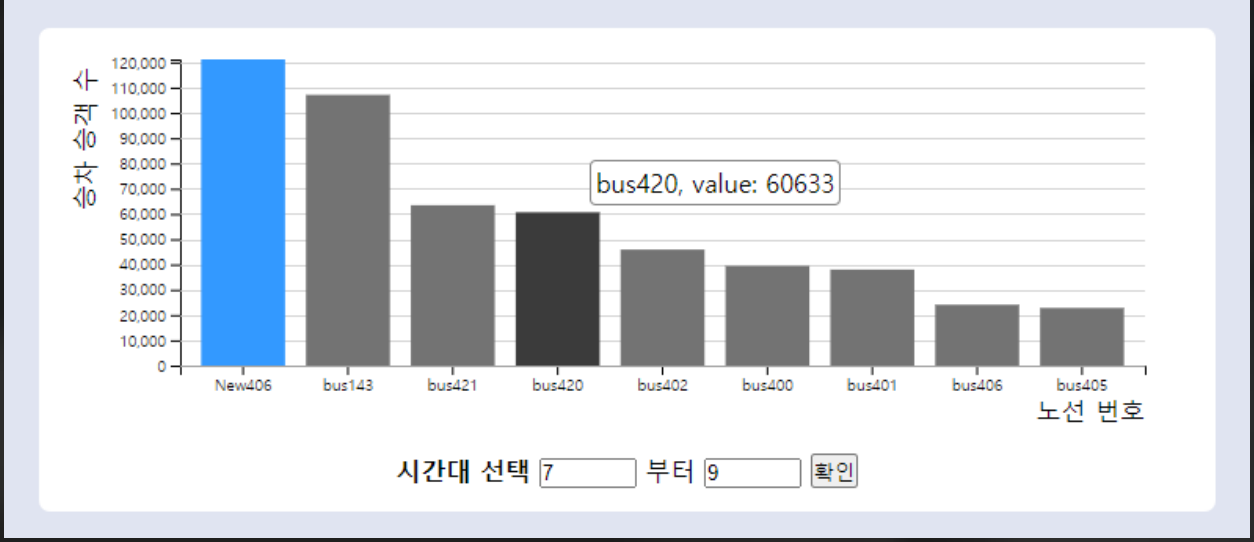
- 확대, 축소 가능한 **지도(Map)**를 사용하여 시각화
- 콤보 박스에서 선택한 해당 신규 노선 및 비교용 기존 노선들의 지도상 경로, 정류장 표시
- 주요 정류장(상위 이용률) 마커로 표시
  - 10만명이 넘는 경우 주요 정류장으로
- 정류장에 마우스 오버 시 해당 정류장의 이름과 하루 총 승차 승객 수를 툴팁으로 제공
- 경로 위로 마우스 오버 시 잘 볼 수 있게 그 경로가 가장 강조되며 노선 이름을 툴팁으로 제공

비교용 기존 노선 선정 기준

가장 비슷한 경로(지역구)를 다니는 노선 약 10개

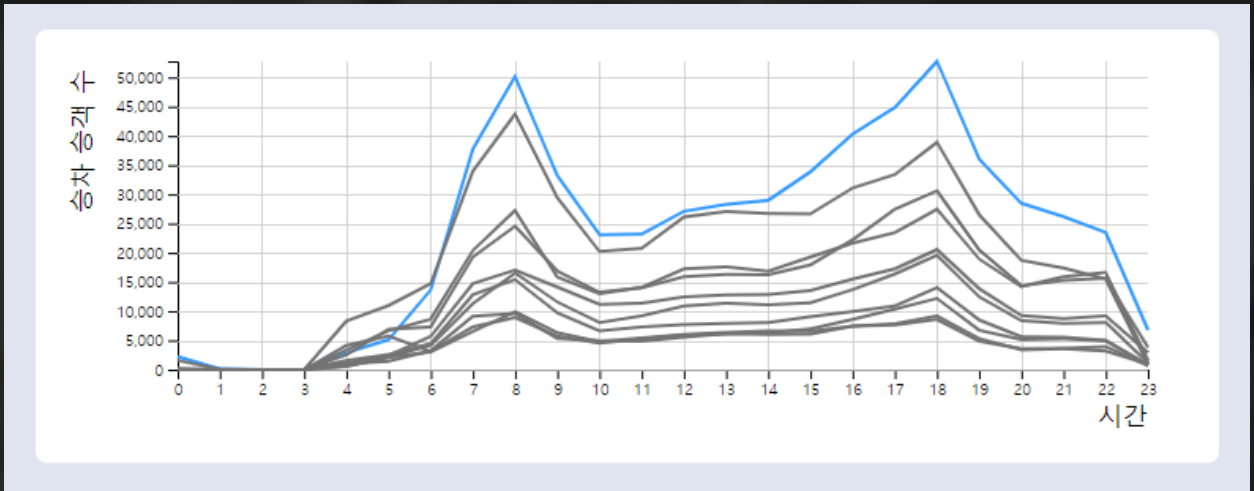
시간대 설정에 따른 노선의 이용 승객 수

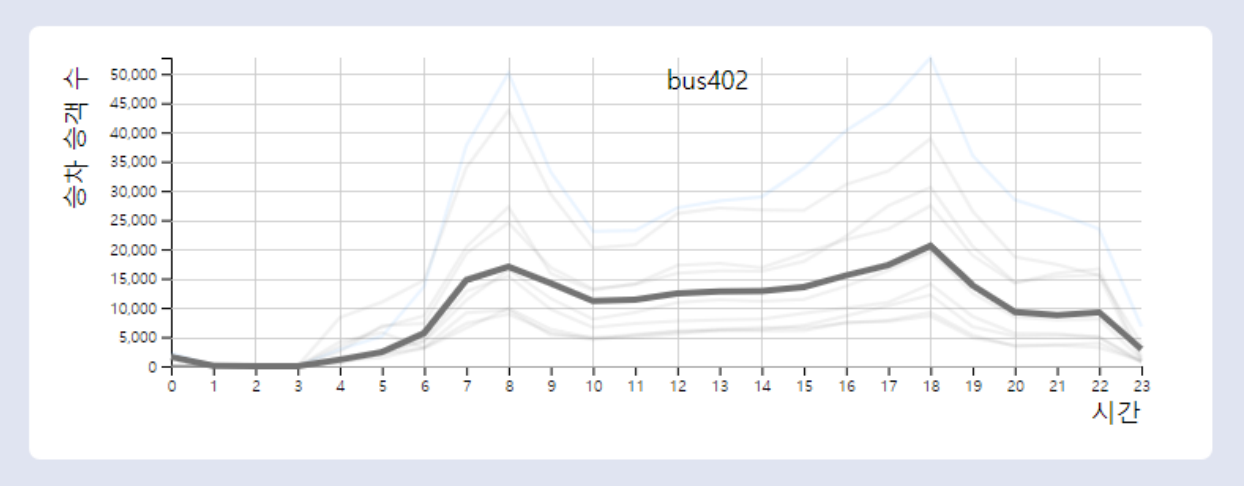




- 선택한 신규 노선과 비교 노선들의 시간대별 이용 승객 수를 나타낸 **막대 그래프**
- 신규 노선은 색상을 달리하여 강조
- 마우스 오버 시 색을 강조하고 노선 정보를 툴팁으로 제공
- 넘버 박스를 조절하여 시간대를 선택 가능
- 애니메이션 효과

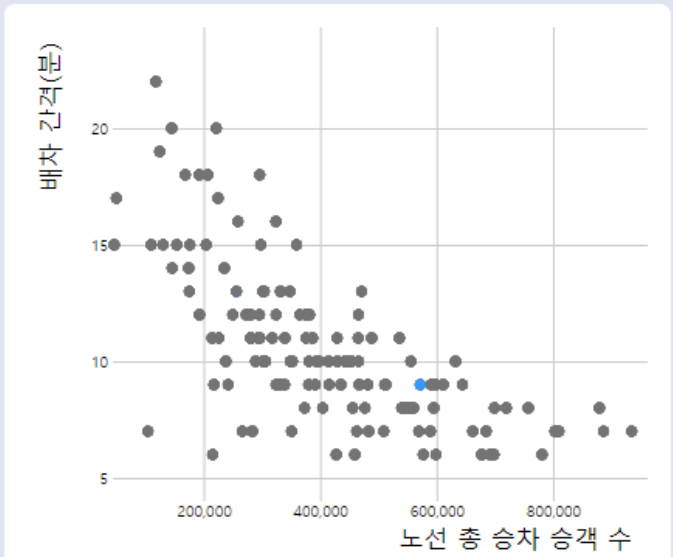
시간대별 노선의 이용 승객 수





- 선택한 신규 노선 및 비교 노선들의 시간대별 이용 승객 수를 나타낸 **라인 그래프**
- 시간에 따른 흐름을 볼 수 있음
- 신규 노선은 색상을 달리하여 강조
- 마우스 오버 시 다른 노선들은 페이드 아웃 시키며 선택 노선을 강조하고 노선 정보를 툴팁으로 제공

신규 노선의 적절한 배차 간격 추천



적절한 배차 간격 추천

약 9분

- 랜덤 포레스트를 사용하여 예측한 적절한 배차 간격 결과를 **텍스트**로 표시
- 전체 노선의 배차 간격을 한 눈에 볼 수 있도록 노선 총 승차 승객 수별 배차 간격을 **산점도**로 나타냄
- 신규 노선은 강조색으로 표시

최종 대시보드

https://dreamy-clarke-06b627.netlify.app

7

기대효과

기능과 기대 효과

→ 기존 버스 노선들의 운영 현황을 여러 측면으로 파악할 수 있음

→ 신규 노선을 추가했을 때의 예상 이용률을 알 수 있음

→ 신규 노선의 예상 이용률을 시간대별로 파악할 수 있음

→ 신규 노선 정보들을 기존 노선 데이터와 쉽게 비교해볼 수 있음

⬇

신규 노선의 효용 가치 판단에 도움을 줄 수 있음

더 나아가, **기존 노선들을 운영**하는 데에도 도움을 줄 수 있음

Epilogue

아쉬운 점

- 양질의 정류장 별로 나타난 구체적인 데이터가 제공되었다면 좋았을 것 같다.
- 노선의 양의 문제로 간선 버스로만 데이터셋을 줄였는데 좀 더 많은 양의 노선으로 특정할 만한 방법이 있었다면 모델링 시각화 부분이 풍부했을 것 같다.
- 대시보드에서 단순히 정보만 제공하는 것이 아니라 사용자와의 인터랙티브 기능이 필요했기 때문에 이 부분이 난이도가 높아서 아쉬운 점이 있다.
- 대시보드에서 하나의 노선을 선택하면 모든 시각화에서 강조할 수 있는 마우스 hover 기능을 구현하지 못해서 아쉽다.