서울시 공공자전거 고장률 예측

4조 엄복동
김민정 김상명 박종원 이지현 정희택

목차

a table of contents

팀원 역할 및 구성

김상명

- 전처리(고장률)
- 선행 연구 조사
- 데이터 분석
- PPT 제작
- 발표

김민정

- 전처리(고장률)
- 데이터 분석
- 데이터 시각화
- PPT 제작

박종원

- 전처리(고장주기)
- 데이터 분석
- 데이터 시각화
- 발표

이지현

- 전처리(고장주기)
- 모델 구축
- 알고리즘 조사
- PPT 제작

정희택

- 전처리(고장주기)
- 모델 구축
- 데이터 시각화
- 프로젝트 계획서 작성

#1

분석 배경 및 방향

주제 선정

주제 후보

league of legend 승률 예측	잡알리오 퇴사예측
열돔,미세먼지 예측	도서 트렌드 예측
분리수거 효율화 방안	자전거 고장 예측

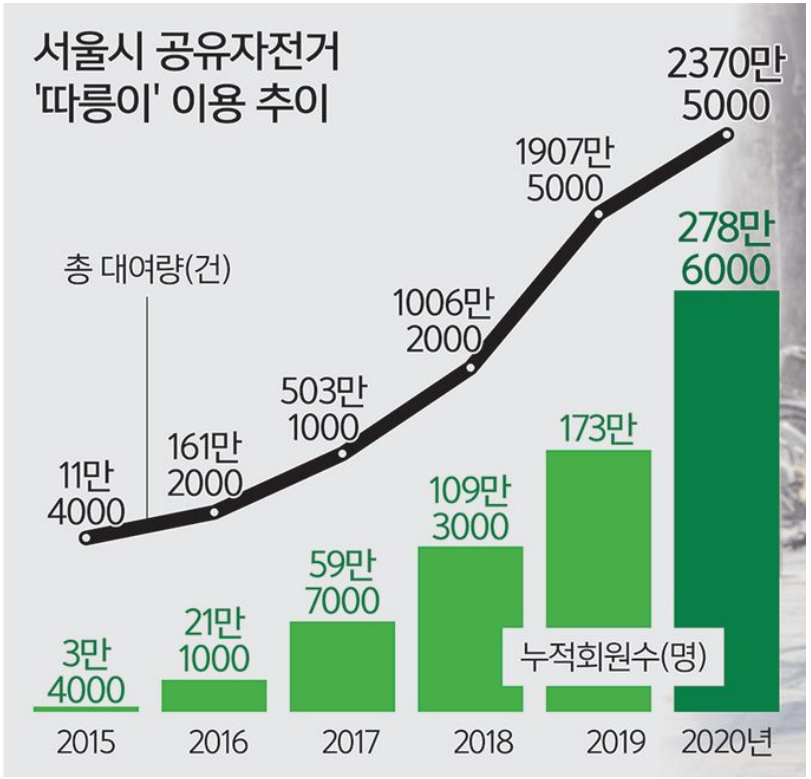
데이터 존재 유무 **vs** 주제의 참신함

Part3,

분석 과정

문제파악	데이터 탐색	데이터 전처리	모델학습	예측
•배경 이해 •분석 목표 확인 •데 이 터 다 운 로 드	•결측치 파악 •이상치 파악 •데이터 시각화 •선행연구	•결측치 처리 •이상치 처리 •자료형 변환 •데이터시각화	• 훈련/검증 • 알고리즘 선택 • 모델링 • 성능 평가	•예측 값 도출 •성능 검사

분석 배경 및 방향



- 서울시에서 성공한 프로젝트의 하나인, 서울 공공자전거 “따릉이”
- 시간에 따른 이용량 증가 추세
- 교통수단으로 부상

Part 1,

분석 배경 및 방향

"안장 뽑고 후미등 가져가"…'따릉이 6년' 올해 정비 16만건

등록 2021.10.29 21:26 / 수정 2021.11.01 15:25

노후화로 녹슬고 고장 늘어
하루평균 1만1300명 이용
작년 보험처리 7배 증가

따릉이 이용하려는 시민들 고장난 자전거 보고 발길 돌려
따릉이 고장 건수 2016년~2019년 8월까지 15만6803건
서울시설공단 "고장난 자전거 이틀 안에 회수해 수리"

따릉이, 고장으로 사고 급증 '안전대책' 시급

4년 간 고장 15만6803건…"고장 및 사고방지위해 노력해야"

> 올해, 전체 따릉이 2만대 운영 중 단말기 고장 건수 1만381건
> 액정 및 전자보드 교체 시, 따릉이 한 대 가격과 맞먹어

분석 배경 및 방향

2015년	2016년	2017년	2018년	2019년	2021년
5건	**1000건**	약 **6천**건	약**3만**건	약**5만**건	**18만**건

분석 배경 및 방향

➡ 고장 자전거의 이용 패턴 파악

➡ 고장을 예측하고 주기적인 정비를 통해 불편 해소

선행연구 조사

빅데이터를 기반으로 한
따릉이 재배치 효율화 연구

재배치 최적화 기법 활용

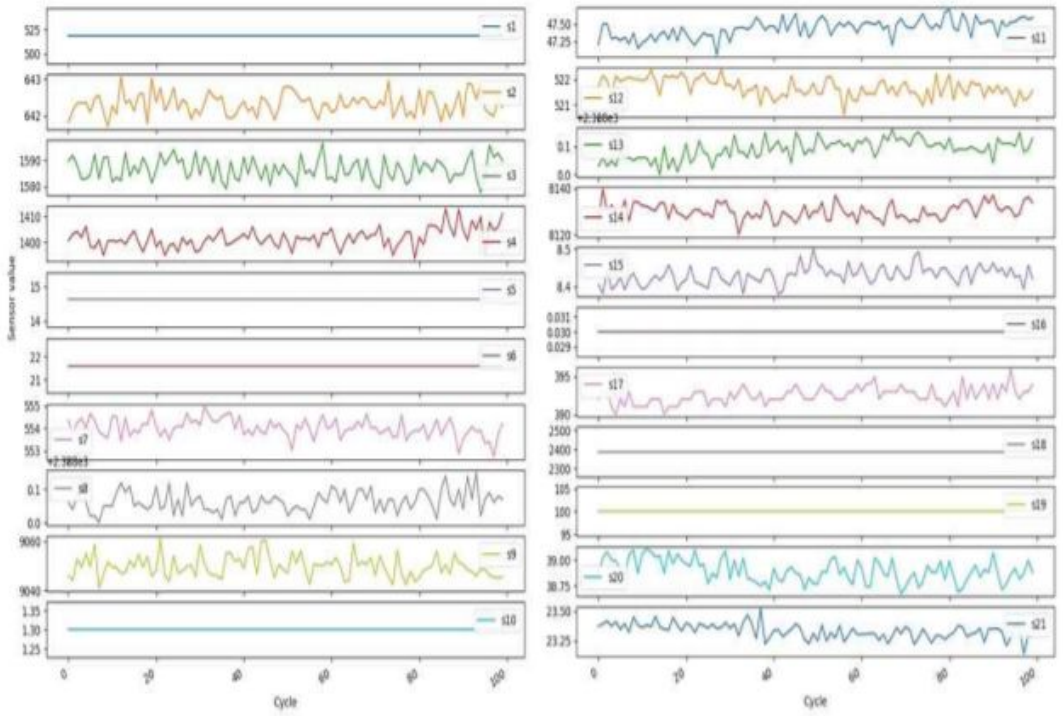
선행연구 조사

기계학습을 이용한 실시간 결함평가 및 고장예측

https://scienceon.kisti.re.kr/commons/util/originalView.do?cn=TRKO201900020693&dbt=TRKO&rn=

- **Ensemble Empirical Mode Decomposition (EEMD) 기법**
- **KNN**
- **PCA**

선행연구 조사

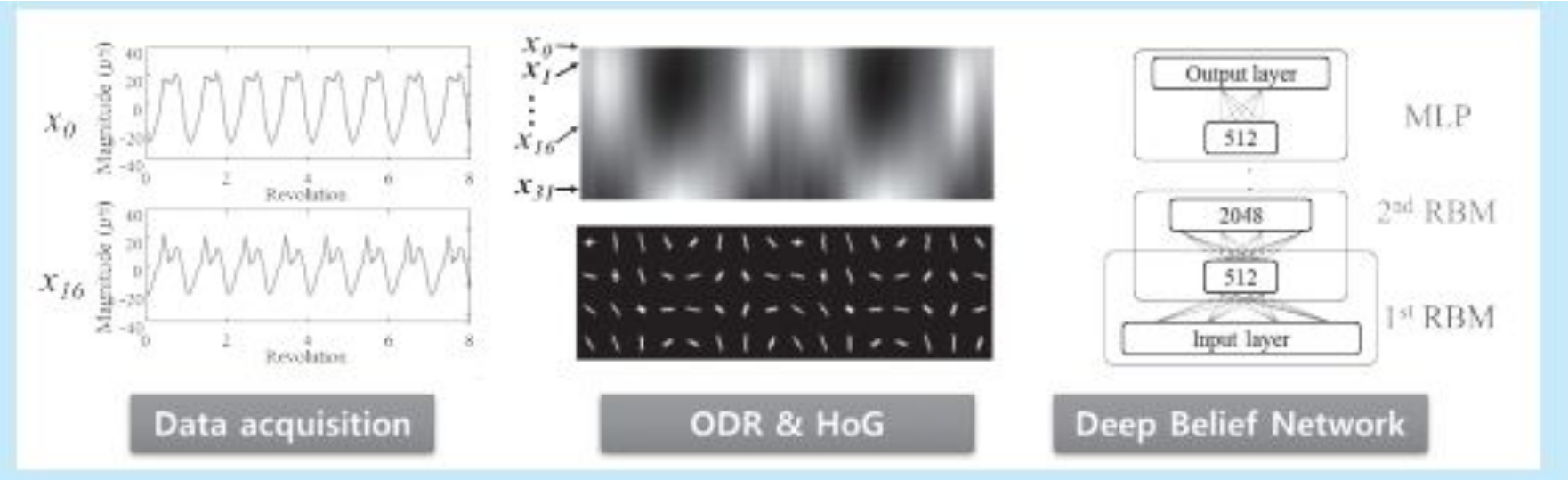


머신러닝을 이용한 고장 예측진단모델 연구

http://mie.pcu.ac.kr/research_file/GgpwYPxlWxRp823opY0rEO1K7luGA0FW.pdf

- **SVM 모델**
- **MCMC 모델**

선행연구 조사



인공지능을 이용한
공학시스템 상태진단 및 예지

- **https://www.koreascience.or.kr/article/JAKO201714940710995.pdf**

#2

분석 과정

데이터셋 설명

서울시 공공자전거 대여이력 정보
index
자전거번호
대여일시
대여 대여소 번호
대여 대여소명
대여거치대
반납일시
반납 대여소 번호
반납 대여소명
반납 거치대
이용시간
이용거리

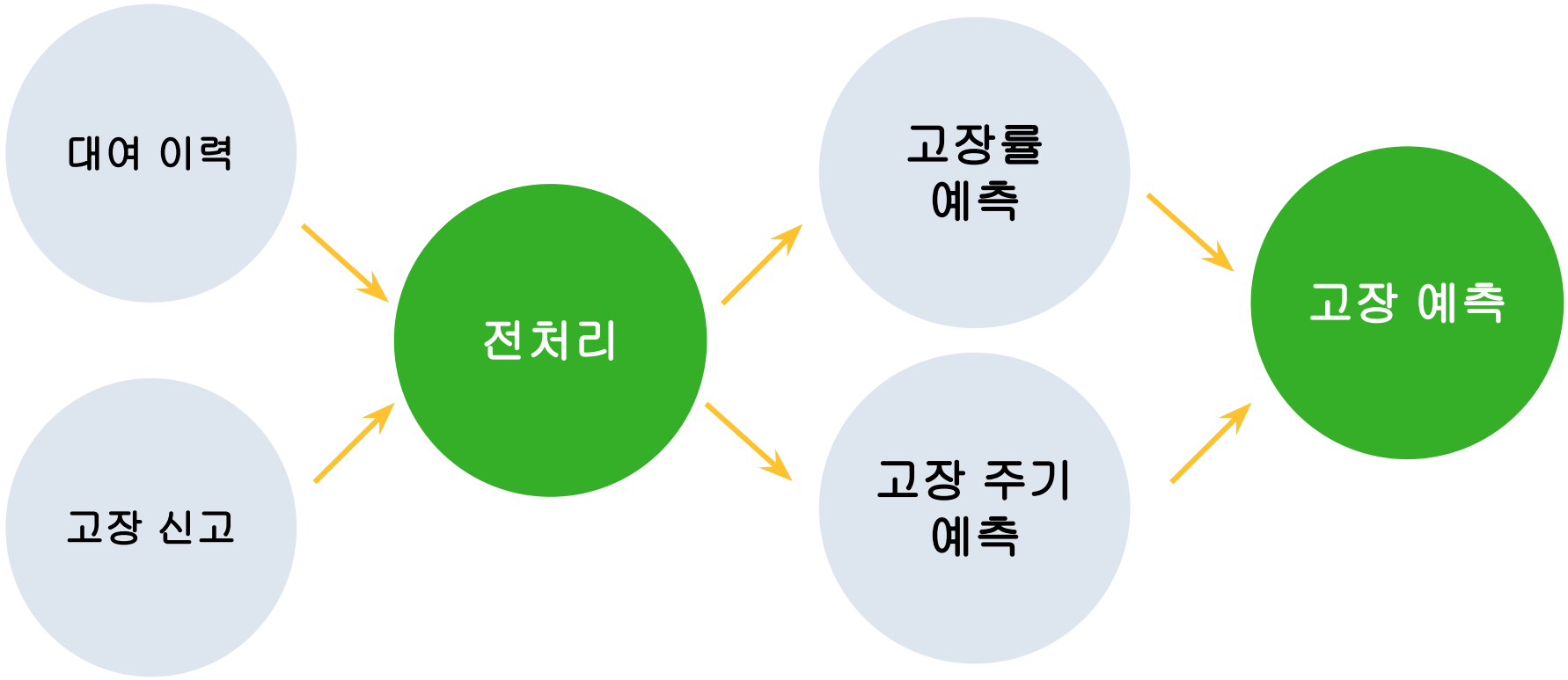
서울시 공공자전거 이용현황
index
등록일시
대여건수

서울시 공공자전거 고장신고 내역
index
자전거번호
등록일시
고장구분

개발 도구

tool	tool	library	library	library	library
python	Tableau	Numpy	Pandas	pycaret	sklearn

분석과정



최종 데이터프레임(고장률)

	breakdown	cumTime	cumDist	cumRide	cumBreak	intensity	meanDist	age	summer
0	1	14910	1398094	458	1	93	3052	54	0
1	1	1281	189910	78	1	148	2434	62	0
2	1	8324	769168	332	3	92	2316	57	1
3	1	31610	2988819	949	1	94	3149	105	1
4	1	5587	518114	176	1	92	2943	41	1

desired target :

1 - breakdown : 고장여부(0:고장X, 1:고장O) (numeric)

input variables :

2 - cumTime : 첫 대여일 ~ 조사당일 까지의 누적이용시간 (numeric)

3 - cumDist : 첫 대여일 ~ 조사당일 까지의 누적이용거리 (numeric)

4 - cumRide : 첫 대여일 ~ 조사당일 까지의 누적이용횟수 (numeric)

5 - cumBreak : 첫 대여일 ~ 조사당일 까지의 누적고장횟수 (numeric)

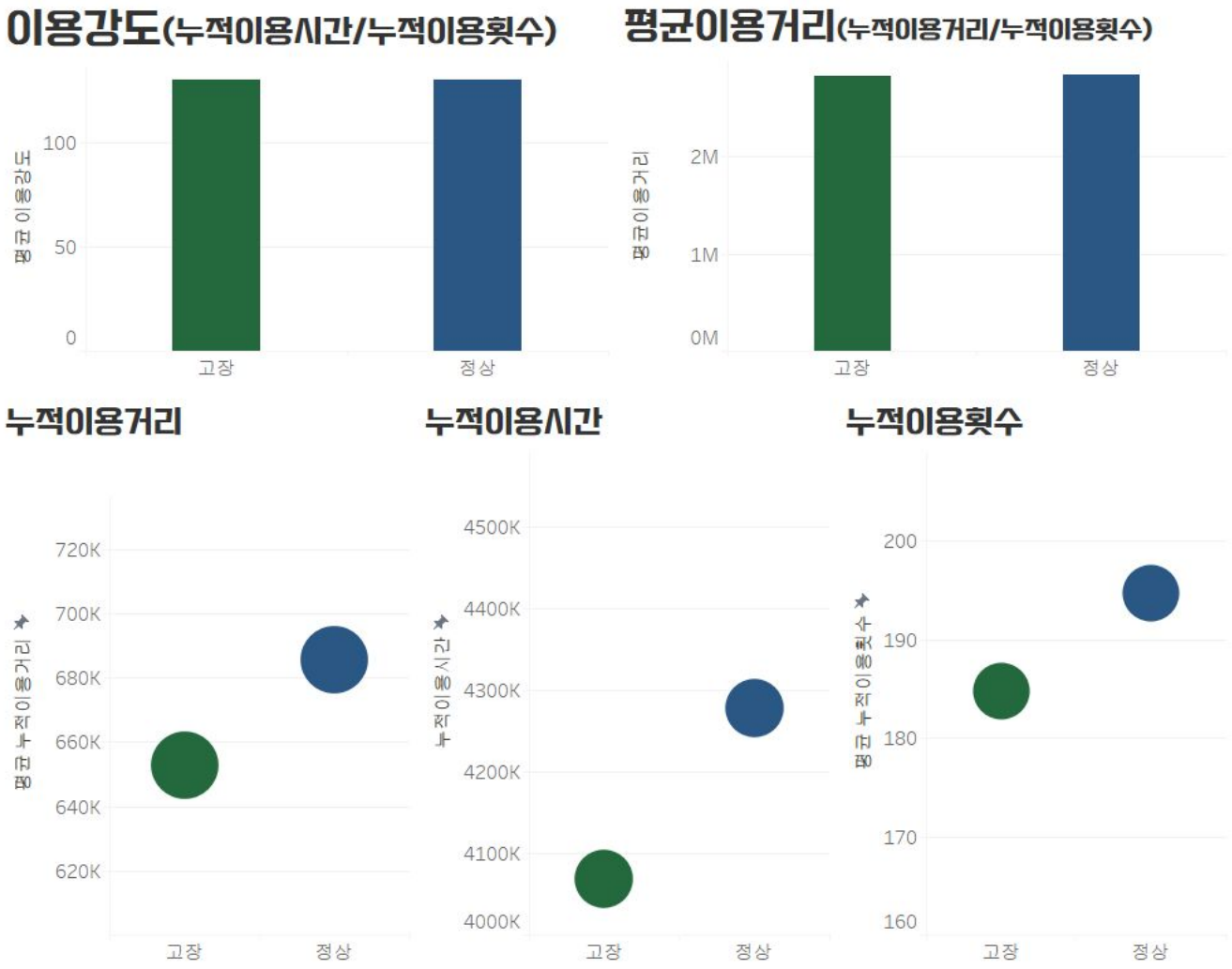
6 - intensity : 이용강도(단위시간 당 평균 이동 거리) (numeric)

7 - meanDist : 평균이용거리(1회 이용 당 평균 이동 거리) (numeric)

8 - age : 따릉이 수명(실제로 이용된 기간, 마지막 대여이력 - 첫 대여이력) (day, numeric)

9 - summer : 따릉이가 7월을 얼마나 겪었는지 (7월 장마기간동안 자전거의 쇠가 녹슬음) (numeric)

전처리 후 EDA



Part 2,

- PREPROCESSING

2 고장(bike_breakdown) 데이터 전처리

2.1 자전거번호, 등록일시, 고장구분 총 3개의 컬럼 이용하여 정렬 'rank' 컬럼 생성

In [12]:
```
df_break['rank'] = factorized
df_break.sort_values(by='rank', ascending=True, inplace = True)

display(df_break)
```

	자전거번호	등록일시	고장구분	rank
10160	SPB-00001	2018-05-17	기타	1
12377	SPB-00001	2018-06-18	기타	2
14514	SPB-00001	2018-07-05	기타	3
14575	SPB-00001	2018-07-06	기타	4
15481	SPB-00001	2018-07-13	기타	5
...	...	...	...	...
336245	SPB-84001	2021-12-23	기타	334677
326098	SPB-84018	2021-11-19	기타	334678
327700	SPB-84039	2021-11-24	단말기	334679
335361	SPB-84149	2021-12-21	기타	334680
316857	SPB-84176	2021-10-29	페달	334681

337873 rows × 4 columns

Part 2,

고장신고 이후 재고장 신고일까지의 날짜 계산

In [33]:
```
df_break_heetaek = df_break.assign(등록일시2=df_break.groupby(['자전거번호','고장구분']).등록일시.shift(-1)).fillna({'등록일시2': df_break.등록일시})
```

In [34]:
```
df_break_heetaek
```

Out[34]:

	자전거번호	등록일시	고장구분	rank	총_고장	등록일시2
10160	00001	2018/05/17	기타	1	1	2018/06/18
12377	00001	2018/06/18	기타	2	1	2018/07/05
14514	00001	2018/07/05	기타	3	1	2018/07/06
14575	00001	2018/07/06	기타	4	1	2018/07/13
15481	00001	2018/07/13	기타	5	1	2018/07/17
...	...	...	...	...	...	...
336245	84001	2021/12/23	기타	334677	1	2021/12/23
326098	84018	2021/11/19	기타	334678	1	2021/11/19
327700	84039	2021/11/24	단말기	334679	1	2021/11/24
335361	84149	2021/12/21	기타	334680	1	2021/12/21
316857	84176	2021/10/29	페달	334681	1	2021/10/29

337873 rows × 6 columns

In [35]:
```
df_break_heetaek.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 337873 entries, 10160 to 316857
Data columns (total 6 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------  ----- 
 0   자전거번호   337873 non-null  object
 1   등록일시    337873 non-null  object
 2   고장구분    337873 non-null  object
 3   rank    337873 non-null  int64 
 4   총_고장    337873 non-null  int64 
 5   등록일시2   337873 non-null  object
dtypes: int64(2), object(4)
memory usage: 18.0+ MB
```

Part 2,

2.4 기간과_기간사이 총 고장횟수 누적값 계산

In [85]:
```
df_break['기간과_기간사이_총_고장횟수'] = df_break.groupby(cols_break)['총_고장'].apply(lambda x: x.cumsum())
```

df_group은 각 고장별 누적합을 위한 dataframe

In [86]:
```
df_group = df_break.groupby(cols_break).max()
df_group.reset_index()
```

Out[86]:

	자전거번호	고장구분	등록일시	총_고장	heetaek_diff	기간과_기관사이_총_고장횟수	기간과_기간사이_총_고장횟수
0	00001	기타	2021-10-28	1	605	45	45
1	00001	단말기	2021-10-28	1	754	65	65
2	00001	안장	2019-10-05	1	165	15	15
3	00001	체인	2019-10-09	1	124	28	28
4	00001	타이어	2019-10-22	1	89	28	28
...	...	...	...	...	...	...	...
164832	84001	기타	2021-12-23	1	0	1	1
164833	84018	기타	2021-11-19	1	0	1	1
164834	84039	단말기	2021-11-24	1	0	1	1
164835	84149	기타	2021-12-21	1	0	1	1
164836	84176	페달	2021-10-29	1	0	1	1

164837 rows × 7 columns

Part 2,

2.5 데이터프레임 Join 후 Preprocessing

In [57]:
```
df_merge = pd.merge(df_break,df_group, how = 'inner', on = (cols_break))
```

In [59]:
```
df_merge = df_merge[['등록일시_x','자전거번호','고장구분','heetaek_diff_x','기간과_기관사이_총_고장횟수_y']]
```

In [60]:
```
df_merge.sample(50)
```

Out[60]:

	등록일시_x	자전거번호	고장구분	heetaek_diff_x	기간과_기관사이_총_고장횟수_y
305194	2021-04-09	50954	페달	5	8
141968	2019-07-28	22735	단말기	71	7
246154	2021-05-31	40419	안장	153	3
158846	2021-07-07	30234	기타	90	6
226431	2021-06-18	37555	체인	24	3
267884	2020-09-14	43491	페달	0	1
80158	2019-09-27	13797	기타	0	1
286816	2021-06-05	46737	기타	124	5
310725	2021-05-06	51900	기타	146	4
86606	2018-08-18	14659	단말기	17	2
84534	2020-03-22	14369	타이어	16	8
94092	2018-11-07	15611	체인	471	3
112147	2019-09-24	17890	체인	22	8
318080	2021-08-25	53035	기타	0	1
154360	2020-03-19	24774	단말기	16	5
251040	2021-10-13	41079	체인	0	3
128267	2020-04-20	20414	단말기	53	7
39504	2019-05-14	07174	페달	0	1

Part 2,

3 최종 데이터 프레임 생성

3.1 대여이력 및 고장이력 데이터프레임 병합

In [74]:
```
cols2 = ['자전거번호','등록일시','고장구분']
```

In [75]:
```
df_final = pd.merge(df_refined_reset,df_merge, how = 'inner', on = cols2)
```

In [88]:
```
df_final.head(10)
```

Out[88]:

	자전거번호	등록일시	등록일시2	고장구분	총이용시간	총이용거리	날짜차이	고장구분별_고장횟수
0	23825	2021-04-29	2021-05-06	단말기	61.0	8750.00	7	3
1	30001	2021-08-24	2021-10-09	타이어	3206.0	353917.70	46	4
2	30001	2021-09-14	2021-11-17	체인	4382.0	462030.56	64	2
3	30008	2021-04-29	2021-06-13	체인	9402.0	1160233.26	45	9
4	30008	2021-05-18	2021-06-16	기타	712.0	83736.18	29	8
5	30008	2021-06-13	2021-07-14	체인	2922.0	325563.42	31	9
6	30008	2021-06-16	2021-07-21	기타	2248.0	254247.74	35	8
7	30008	2021-07-14	2021-07-28	체인	1528.0	154152.72	14	9
8	30008	2021-07-21	2021-08-04	기타	1076.0	122762.64	14	8
9	30008	2021-07-28	2021-08-04	체인	1076.0	122762.64	7	9

3.2 결측치 최종 확인

In [79]:
```
df_final.isnull().sum()
```

Out[79]:
```
자전거번호        0
등록일시          0
등록일시2         0
고장구분          0
총이용시간         0
총이용거리         0
날짜차이          0
고장구분별_고장횟수    0
dtype: int64
```

최종 데이터프레임(고장주기)

columns name	mean	datatype	count	non-null
자전거번호	각 자전거 고유 번호	object	20571	non-null
등록일시	고장신고 등록날짜	datetime64	20571	non-null
등록일시2	대여이력 등록날짜	object	20571	non-null
고장구분	고장상태 구분	object	20571	non-null
총이용시간	대여일 이후 고장신고 날까지의 이용시간	float64	20571	non-null
총이용거리	대여일 이후 고장신고 날까지의 이용거리	float64	20571	non-null
날짜차이	고장신고 이후 재고장까지의 일수	int64	20571	non-null
고장구분별_고장횟수	자전거, 고장부분 별 고장 횟수	int64	20571	non-null

최종 데이터프레임(고장주기)

	총이용시간	총이용거리	고장구분별_고장횟수	날짜차이
2	4382	462030.56	2	64
3	9402	1160233.26	9	45
5	2922	325563.42	9	31
7	1528	154152.72	9	14
9	1076	122762.64	9	7
...	...	...	...	...
20529	199	23980.00	5	10
20530	2544	328240.00	5	20
20547	574	63360.00	2	33

#3

프로젝트 결론

고장률 예측

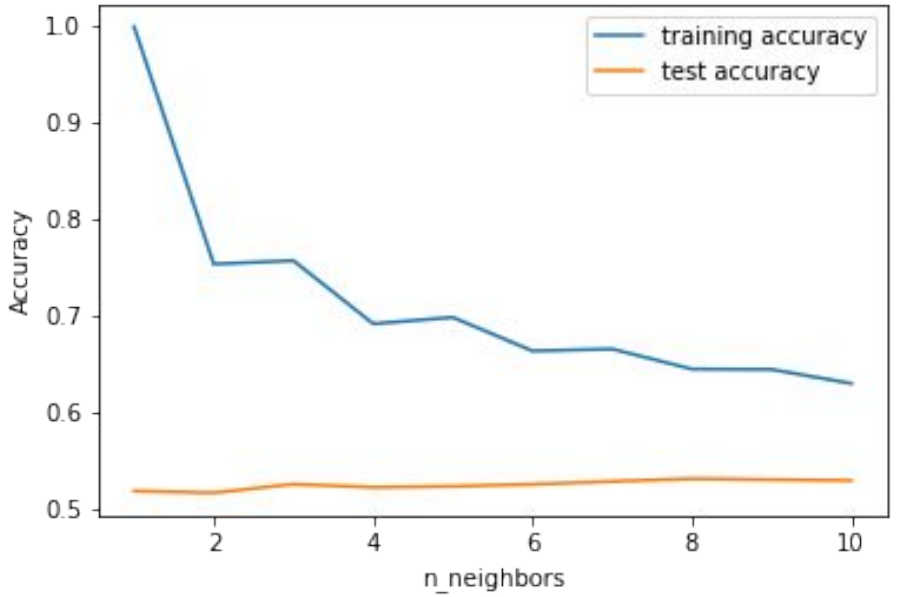
고장률 예측

자전거의 여러 요소들을 고려해 자전거가 고장난다 / 안 난다 예측하면 되므로 지도학습 중 불연속한 값을 예측하는 '분류' 분석을 하고자 함.

>> 사용 알고리즘

1. KNN
2. 로지스틱 회귀분석
3. Decision Tree
4. Gradient Boost

KNN



이웃의 수가 늘수록 training data의 정확도는 떨어지고 test data의 정확도는 올라감

neighbors의 수를 늘려도 test accuracy가 너무 더디게 올라가서 과소적합 상태지만 적당한 fitting을 위해 neighbors의 수는 7로 결정

KNN

```
#데이터 학습
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)

#성능평가
print('training set 정확도: {:.2f}'.format(knn.score(X_train, y_train)))
print('test set 정확도: {:.2f}'.format(knn.score(X_test, y_test)))
```

✓ 1.4s

```
training set 정확도: 0.67
test set 정확도: 0.53
```

◂ KNN 모델링 결과

▼ 10-fold 교차검증 결과

```
kfold = model_selection.KFold(n_splits=10, random_state=7, shuffle=True)
modelCV = KNeighborsClassifier(n_neighbors=7)
scoring = 'accuracy'
results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=scoring)
print("10-fold cross validation average accuracy: %.3f" % (results.mean()))
```

✓ 1.7s

```
10-fold cross validation average accuracy: 0.514
```

KNN

```
print(classification_report(y_test, y_pred))
```

✓ 0.9s

	precision	recall	f1-score	support
0	0.53	0.53	0.53	6002
1	0.53	0.52	0.52	5998
accuracy			0.53	12000
macro avg	0.53	0.53	0.53	12000
weighted avg	0.53	0.53	0.53	12000

◂ confusion matrix

precision(정밀도) : 양성이라 예측한 값 중 실제 양성인 개수

recall(재현율) : 실제 양성인 값 중 양성이라 예측한 개수

accuracy(정확도) : 전체 중 제대로 예측한 샘플의 개수

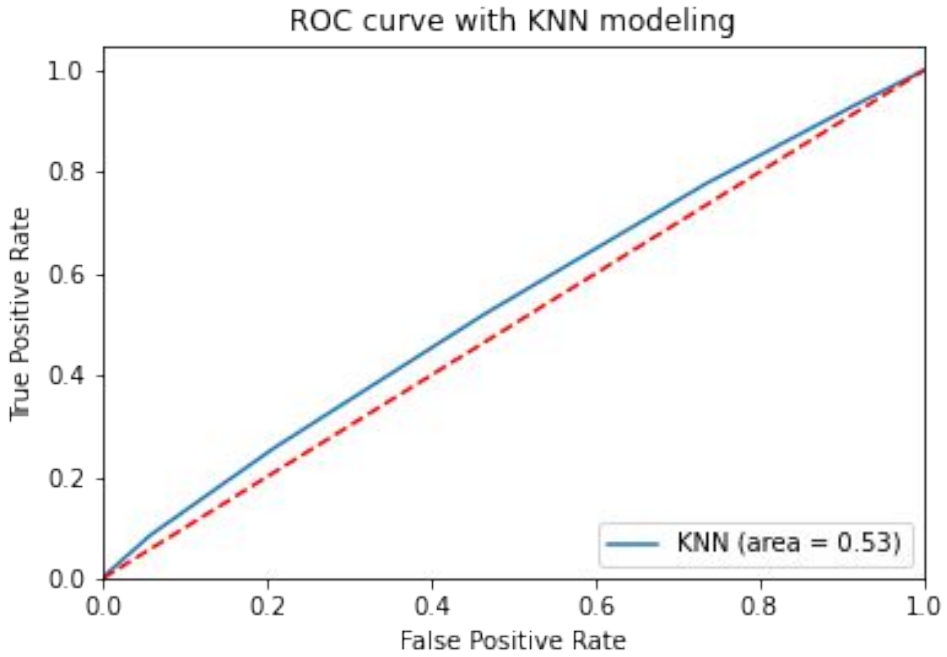
f1-score : precision과 recall의 가중 조화평균

support : 각 라벨의 실제 샘플 개수

macro avg : 단순 평균 (각 클래스의 샘플 개수의 불균형 고려 X)

weighted avg : 가중 평균 (각 클래스의 샘플 개수의 불균형 고려 O)

KNN



◂ ROC curve

AUC score = **0.53**

분석실패 원인 :

EDA를 했을 때 고장 데이터와 정상 데이터의 분포가 비슷했음에도 가까운 위치에 있는 데이터끼리 묶는 KNN 알고리즘을 사용함.

로지스틱 회귀분석

Logit Regression Results

Dep. Variable:	breakdown	No. Observations:	40000
Model:	Logit	Df Residuals:	39992
Method:	MLE	Df Model:	7
Date:	Sat, 12 Mar 2022	Pseudo R-squ.:	0.09472
Time:	01:31:06	Log-Likelihood:	-25100.
converged:	True	LL-Null:	-27726.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
cumTime	1.148e-05	5.94e-06	1.935	0.053	-1.5e-07	2.31e-05
cumDist	2.343e-07	4.01e-08	5.841	0.000	1.56e-07	3.13e-07
cumRide	-0.0018	0.000	-9.008	0.000	-0.002	-0.001
cumBreak	0.3645	0.006	60.202	0.000	0.353	0.376
intensity	0.0033	0.001	4.740	0.000	0.002	0.005
meanDist	-0.0002	2.48e-05	-7.730	0.000	-0.000	-0.000
age	-3.554e-06	0.000	-0.021	0.983	-0.000	0.000
summer	-0.1261	0.027	-4.728	0.000	-0.178	-0.074

◂ logit model summary

age 변수 외 모든 변수의 p-value가 충분히 작음

age 변수를 제외하면 모두 유의함

로지스틱 회귀분석

```
#데이터 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

#데이터 학습
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

print('Train set 정확도: %2f' % logreg.score(X_train, y_train))
print('Test set 정확도: %2f' % logreg.score(X_test, y_test))
```

✓ 0.2s

```
Train set 정확도: 0.551929
Test set 정확도: 0.559333
```

◂ 로지스틱 회귀분석 모델링 결과

▼ 10-fold 교차검증 결과

```
kfold = model_selection.KFold(n_splits=10, random_state=7, shuffle=True)
modelCV = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=scoring)
print("10-fold cross validation average accuracy: %.3f" % (results.mean()))
```

✓ 0.9s

```
10-fold cross validation average accuracy: 0.536
```

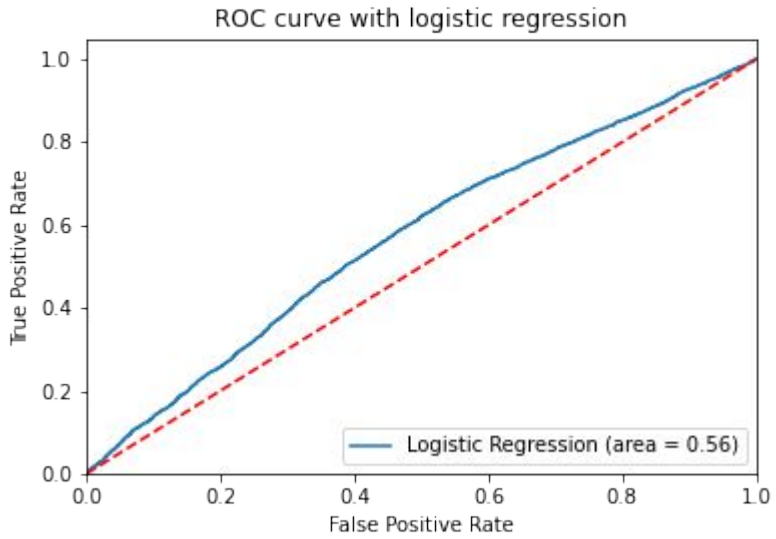
로지스틱 회귀분석

```
print(classification_report(y_test, y_pred))
```

✓ 0.8s

	precision	recall	f1-score	support
0	0.57	0.50	0.53	6002
1	0.55	0.62	0.58	5998
accuracy			0.56	12000
macro avg	0.56	0.56	0.56	12000
weighted avg	0.56	0.56	0.56	12000

▲ confusion matrix



▲ ROC curve

AUC score = **0.56**

의사결정나무

```
#데이터 학습
tree = DecisionTreeClassifier(max_depth=3,random_state=0) #max_depth : 과적합방지
tree_model = tree.fit(X_train, y_train)

#성능평가
print("training set 정확도: {:.2f}".format(tree.score(X_train, y_train)))
print("test set 정확도: {:.2f}".format(tree.score(X_test, y_test)))
```

✓ 0.1s

```
training set 정확도: 0.71
test set 정확도: 0.71
```

▲ 의사결정나무 모델링 결과

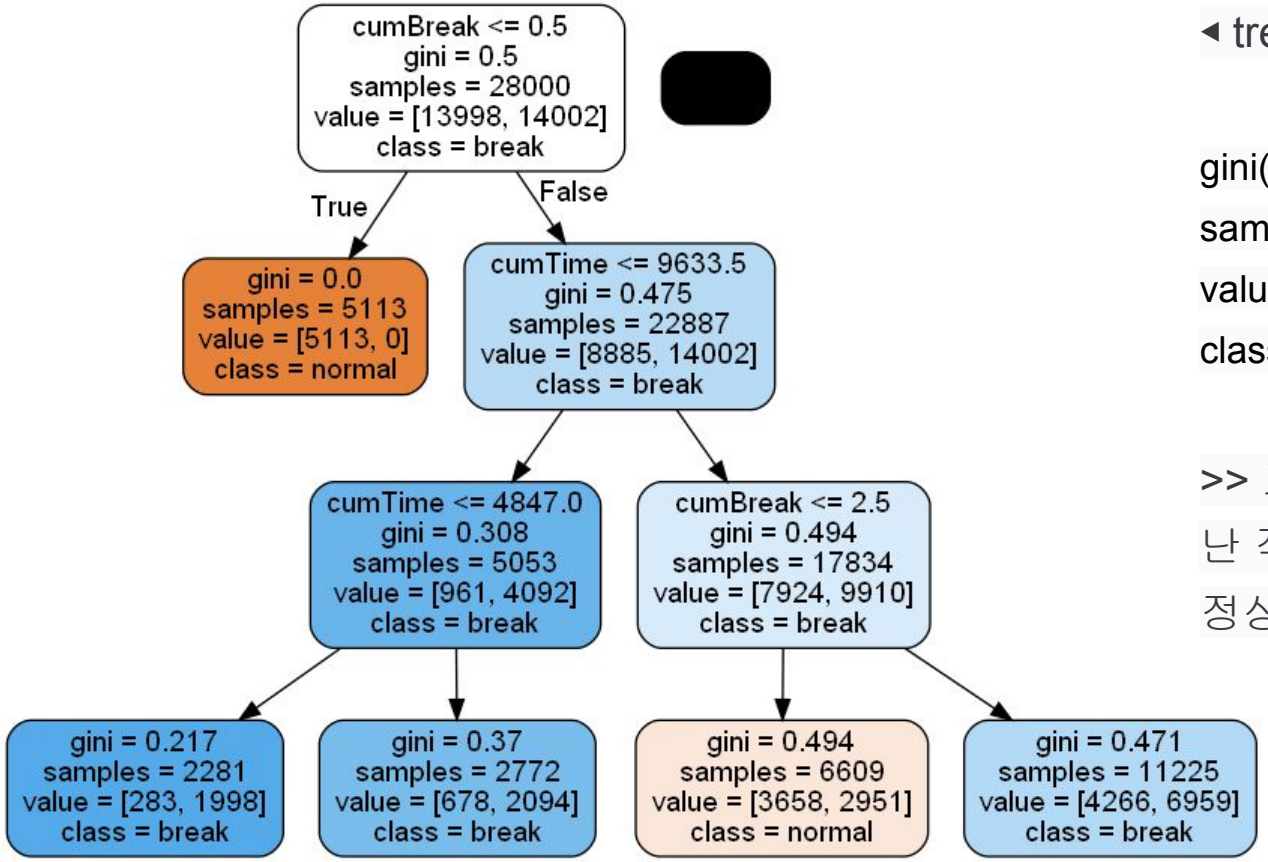
```
print("AUC score: %.2f" % roc_auc_score(y_test, y_test_pred))
```

✓ 0.7s

```
AUC score: 0.71
```

◂ AUC score = **0.71**

의사결정나무



◂ tree graph

gini(불순도) : 0에 가까울수록 불순도가 적은 순수노드

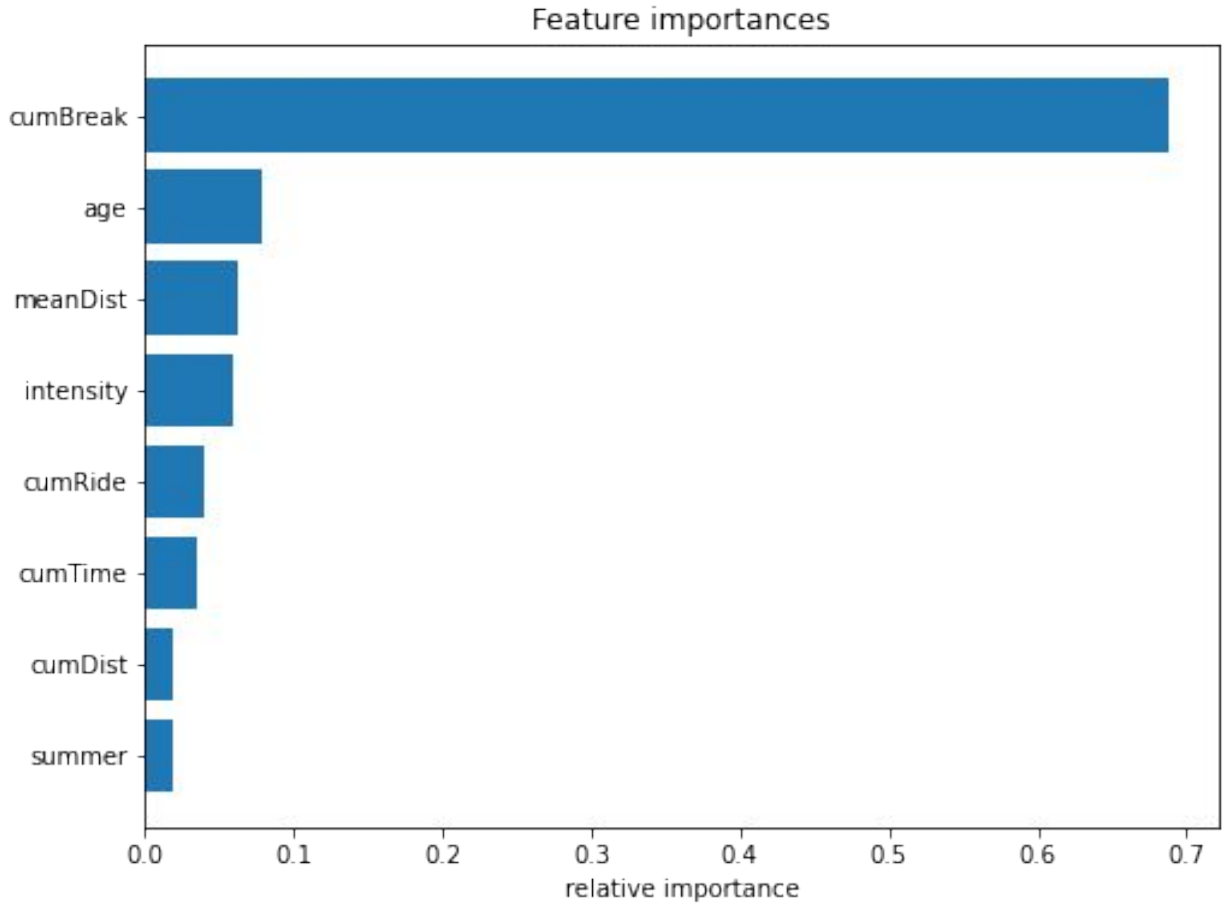
samples : 데이터 수

value : 라벨의 클래스 수

class : 최종분류

>> 고장내력이 <= 0.5 인 데이터는 한번도 고장난 적이 없는 데이터로 이를 만족하면 무조건 정상 자전거임. 정확도가 높을 수 밖에 없음.

Gradient Boost



◂ 변수 별 중요도 순위

고장누적횟수가 가장 중요한 변수로 작용했고 자전거나이, 평균이용거리, 이용강도가 다음을 차지함.

>> 혼란을 줄 수 있는 중요도가 낮은 변수를 제거하고 앙상블 부스트 모델 중 하나인 Gradient Boost 시행

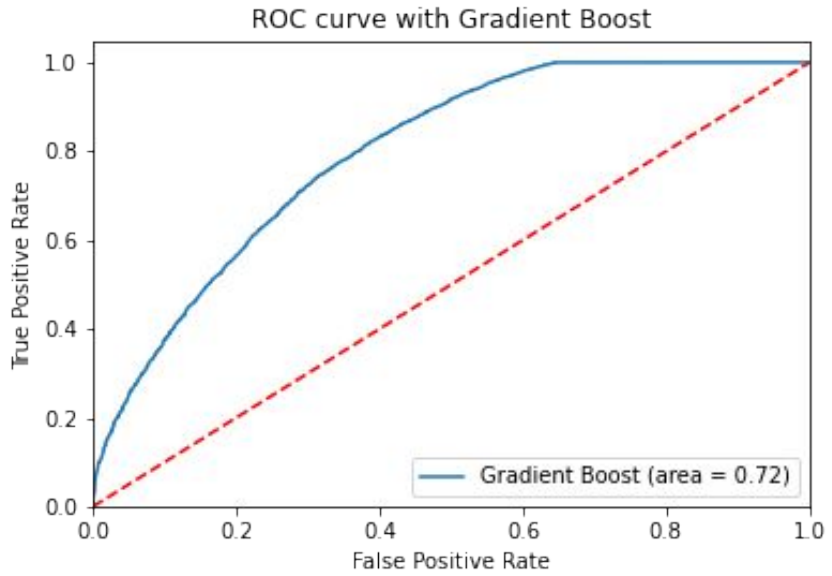
Gradient Boost

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(random_state=0, max_depth=3)
gb.fit(X_train2, y_train)

print("training set 정확도: {:.2f}".format(gb.score(X_train2, y_train)))
print("test set 정확도: {:.2f}".format(gb.score(X_test2, y_test)))
```

✓ 1.9s

```
training set 정확도: 0.72
test set 정확도: 0.72
```

◂ GB 모델링 결과



◂ ROC curve

AUC score = **0.72**

고장주기 예측

고장주기 예측

자전거의 첫 고장 신고가 접수한 이후 다음 고장 신고까지 어느 정도 소요되었는지 확인하고자 다음의 알고리즘을 적용

>> 사용 알고리즘

1. 로지스틱 회귀분석
2. Huber Regression
3. Gradient Boosting Regression

Gradient Boosting Regression

```
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(features, label)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)

model = GradientBoostingRegressor()
model.fit(train_features, train_labels)

print(model.score(train_features, train_labels))
print(model.score(test_features, test_labels))
```

```
0.512214582265118
0.29857811877250096
```

Part 3,

Compare Model

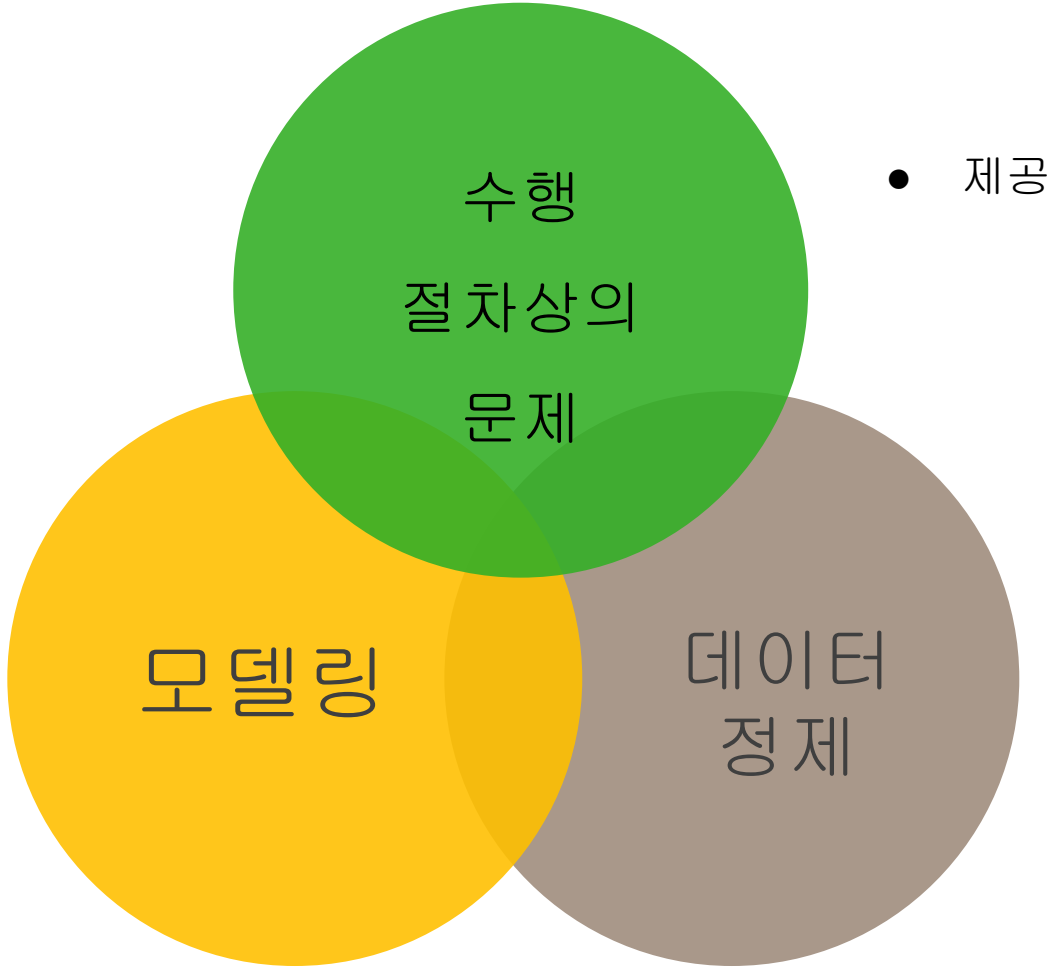
In [100]:
```
best = compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	22.7582	1256.4981	35.2091	0.2565	1.1646	3.0092	0.0330
catboost	CatBoost Regressor	22.7728	1268.7509	35.3201	0.2529	1.1649	2.9321	0.4710
en	Elastic Net	23.7445	1289.9344	35.5825	0.2416	1.2792	3.9852	0.0070
lasso	Lasso Regression	23.7801	1289.6338	35.5809	0.2416	1.2815	3.9963	0.0070
lr	Linear Regression	23.8249	1289.4011	35.5814	0.2415	1.2862	4.0093	0.6320
ridge	Ridge Regression	23.8248	1289.4006	35.5813	0.2415	1.2862	4.0093	0.0060
lar	Least Angle Regression	23.8249	1289.4008	35.5814	0.2415	1.2862	4.0093	0.0070
br	Bayesian Ridge	23.7689	1290.0338	35.5849	0.2414	1.2803	3.9926	0.0070
omp	Orthogonal Matching Pursuit	24.1398	1322.9331	36.0433	0.2222	1.2997	4.0405	0.0070
lightgbm	Light Gradient Boosting Machine	24.0596	1331.1497	36.1902	0.2164	1.2294	3.1315	0.2390
rf	Random Forest Regressor	24.1014	1383.5936	36.9071	0.1815	1.2158	3.1583	0.0900
knn	K Neighbors Regressor	24.8164	1433.1654	37.5786	0.1514	1.2560	3.1127	0.0080
et	Extra Trees Regressor	24.6187	1514.3341	38.5803	0.1096	1.2212	3.0782	0.0750
xgboost	Extreme Gradient Boosting	24.7711	1505.2411	38.5754	0.1047	1.2568	3.1839	0.0920
huber	Huber Regressor	20.8156	1658.3848	40.2965	0.0210	1.2254	0.6597	0.0090
llar	Lasso Least Angle Regression	29.9210	1719.9072	41.2333	-0.0114	1.5089	5.8332	0.0070
ada	AdaBoost Regressor	34.0663	1721.9568	41.2971	-0.0464	1.5632	6.8325	0.0140
dt	Decision Tree Regressor	29.6067	2474.6655	49.4395	-0.4744	1.3766	3.1830	0.0060
par	Passive Aggressive Regressor	29.0332	4729.7270	60.7647	-1.7494	1.3326	1.0073	0.0060

#4

제언

고장주기 예측

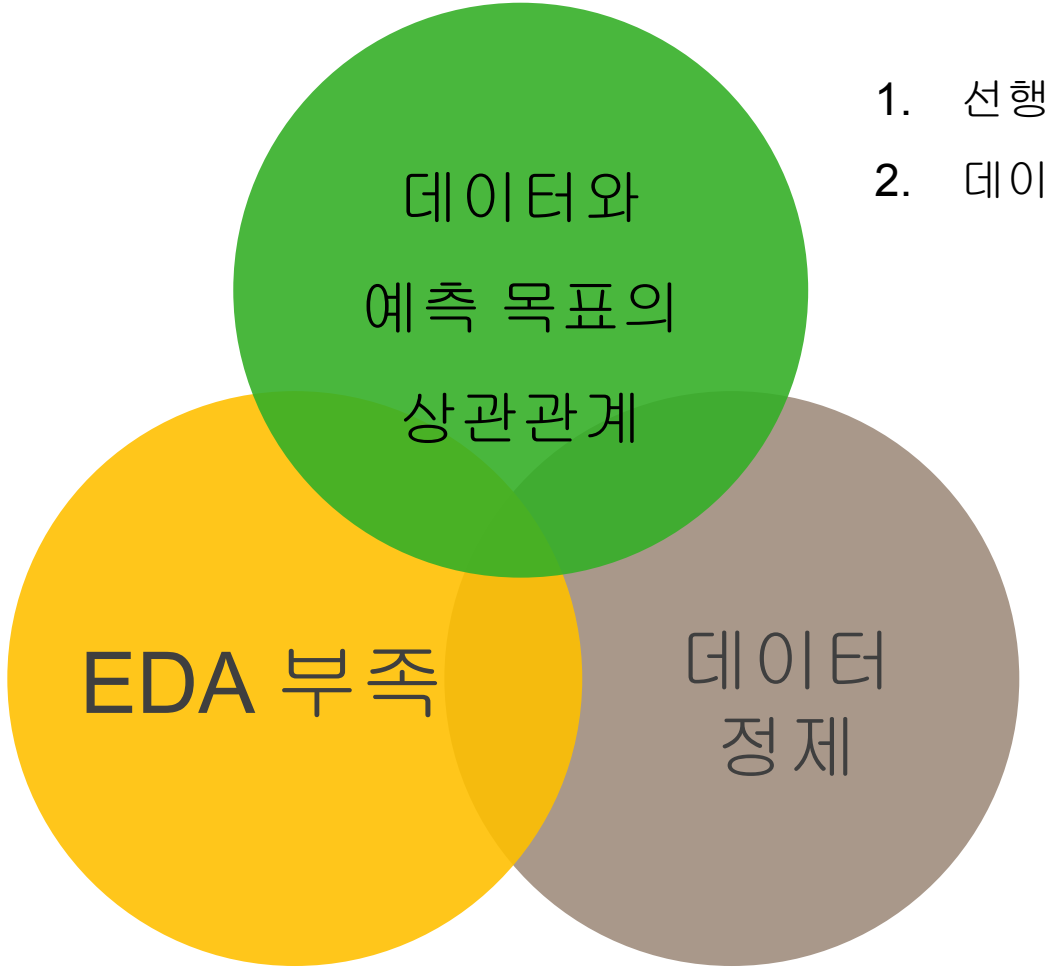


- 제공된 장비들의 성능 문제

- 모델 구축 결과가
서로 많이 다름

- 원본 데이터의 복잡함
- 새 데이터 생성

고장률 예측



1. 선행연구 참고
2. 데이터 필드 부족

1. 필드 선정

1. 저장 방식
2. 큰 데이터

고장 예측

- 누적이용시간과 상관관계가 높다.
  이용시간에 따른 정기적인 정비 필요

- 자전거를 이용하면서 고장이 발생할 수 있지만 그 주기가 짧아지면 이용에 어려움이 발생할 수 있어 사용자들도 대여 시 주의 필요


감사합니다