KUBIG CONTEST
22-1 KUBIG 머신러닝 분반

병원 개/폐업 분류 예측 경진대회

고태영 반민정 염윤석 최경석

CONTENTS

목차

KUBIG CONTEST

대회 목표

병원 폐업 여부 예측 모델 개발

병원 재무 데이터와 AI를 활용하여

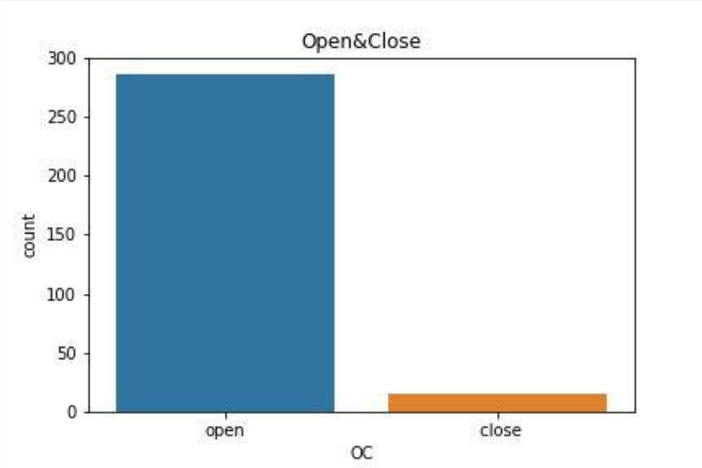
병원의 개업 또는 폐업 여부를 분류 예측

EDA

변수설명

About Categorical Variable

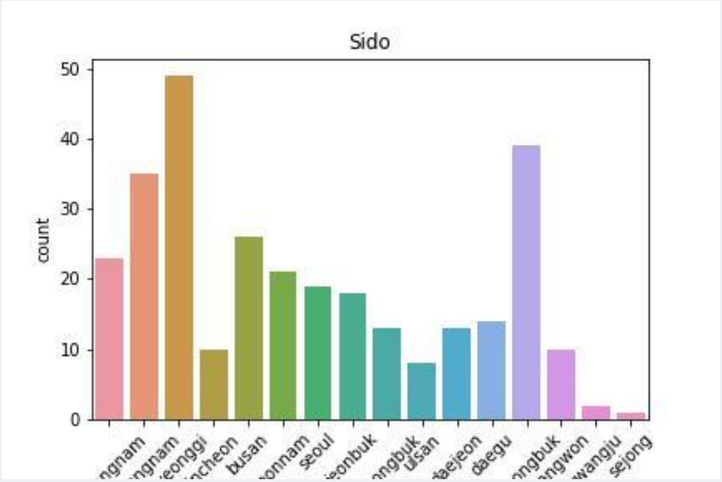
Open & Close



Open: 개원
Close : 폐업

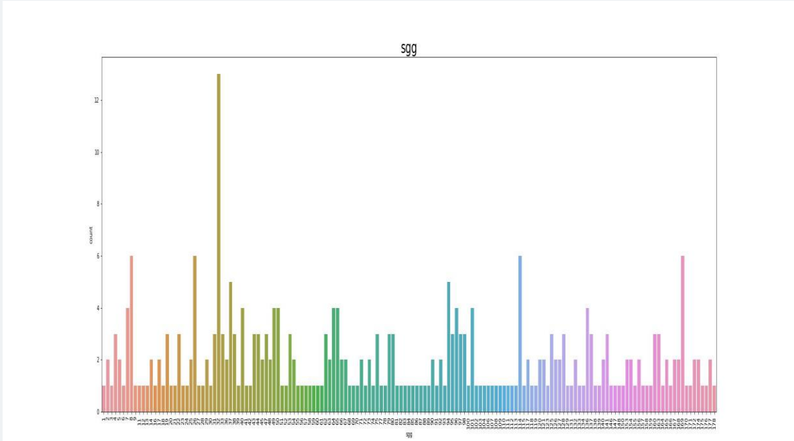
Open과 Close 데이터 개수의
불균형 정도가 심함

Sido



대한민국의 행정구역 (시,도)
총 16곳의 광역 지역에 대한 병원 정보

sgg



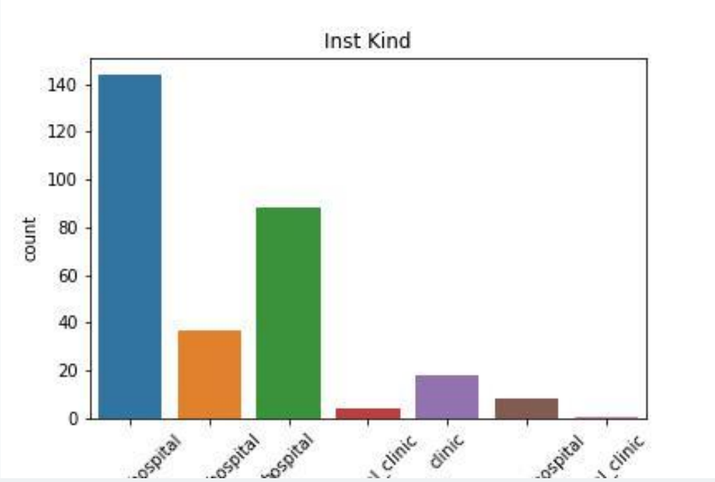
대한민국의 시,군,구에 대한
병원 정보

EDA

변수설명

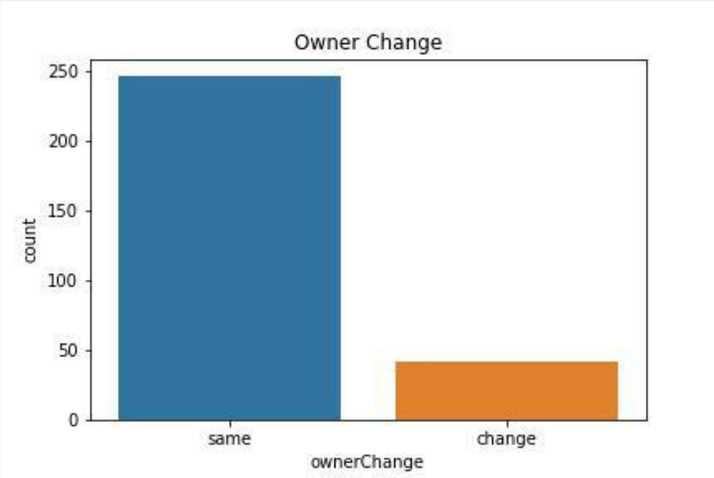
About Categorical Variable

Inst Kind



〈병원의 종류〉
- nursing_hospital : 요양원
- general_hospital : 종합병원
- hospital : 병원
- traditional_clinic : 한의원
- clinic : 의원
- traditional_hospital : 한방병원
- dental_clinic : 치의원

Owner Change



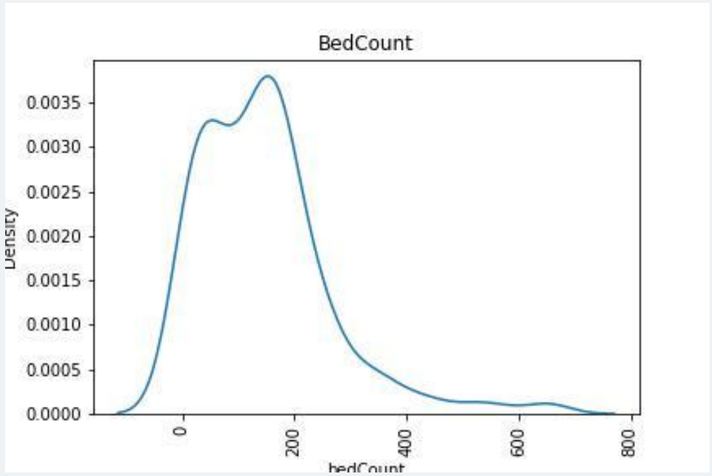
대표자 변동여부

EDA

변수설명

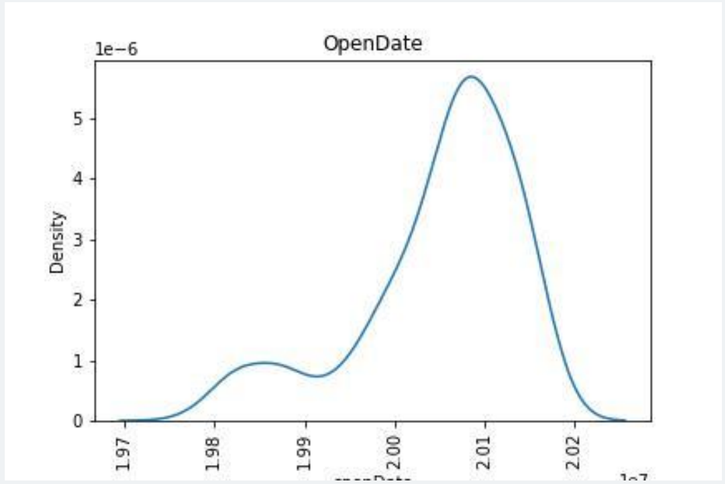
About Numerical Variable

Bed Count



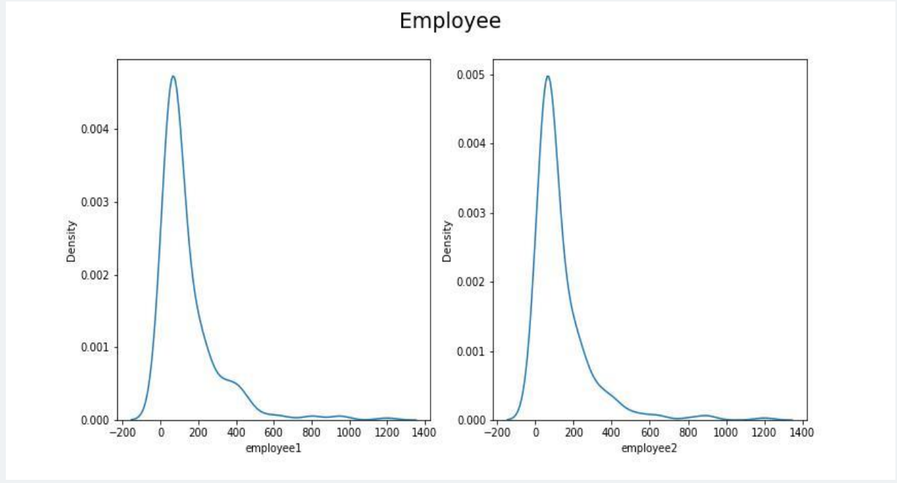
병원이 가지고 있는 병상의 수

Open Date



병원의 개원 날짜(연/월/일)
=>연도 데이터를 활용하여 얼마동안
운영했는지에 대한 새로운 변수 생성 가능

Employee



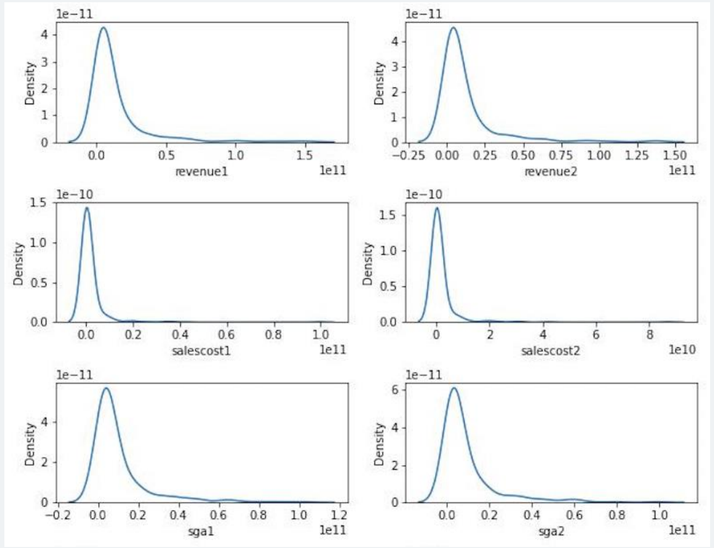
병원이 고용한 총 직원 수
- employee1 : 2017년
- empolyee2 : 2016년

EDA

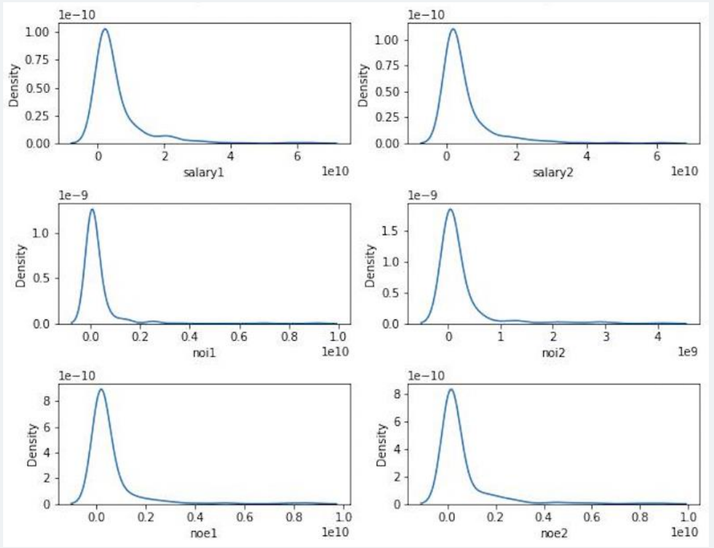
변수설명

About Numerical Variable

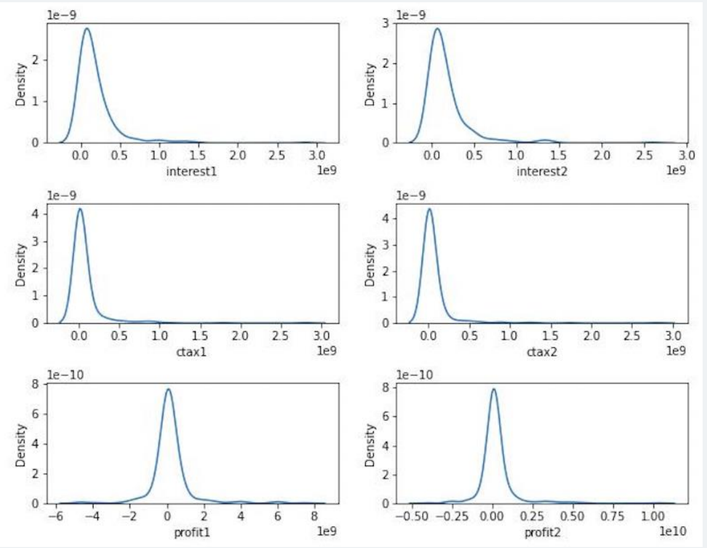
Income Statement(손익계산서)



- revenue : 매출액
- salecost : 매출원가
- sga : 판매비와 관리비



- salary : 급여
- noi : 영업외 수익
- noe : 영업외 비용



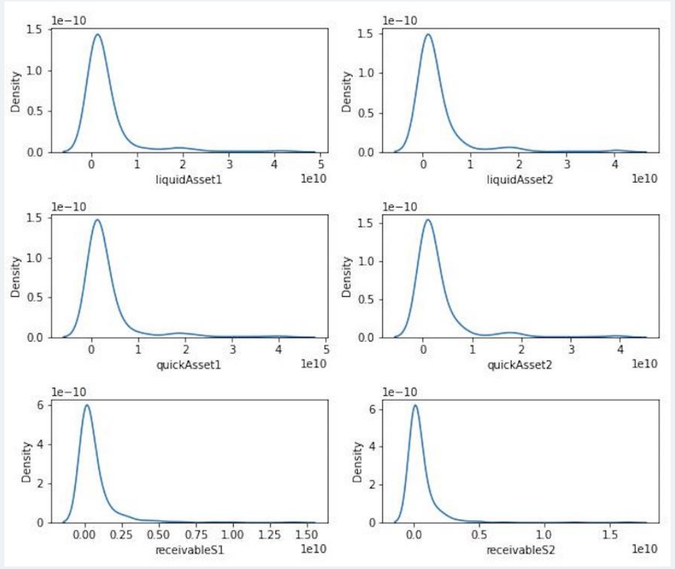
- Interest : 이자비용
- Ctax : 법인세비용
- Profit : 당기순이익

EDA

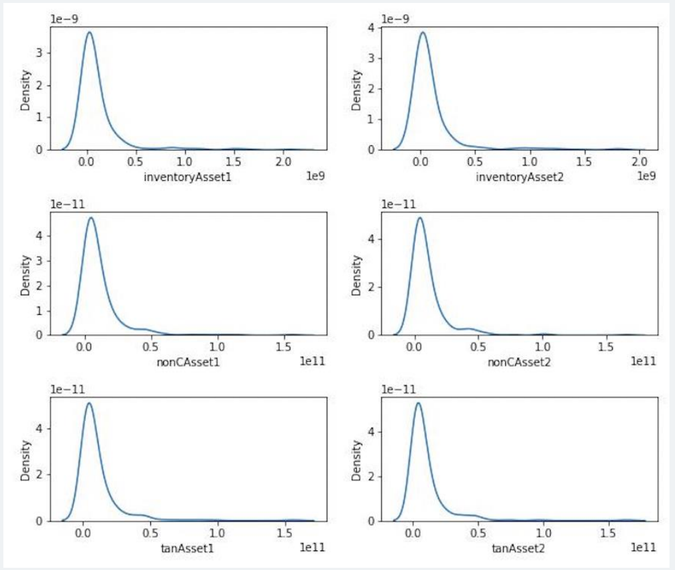
변수설명

About Numerical Variable

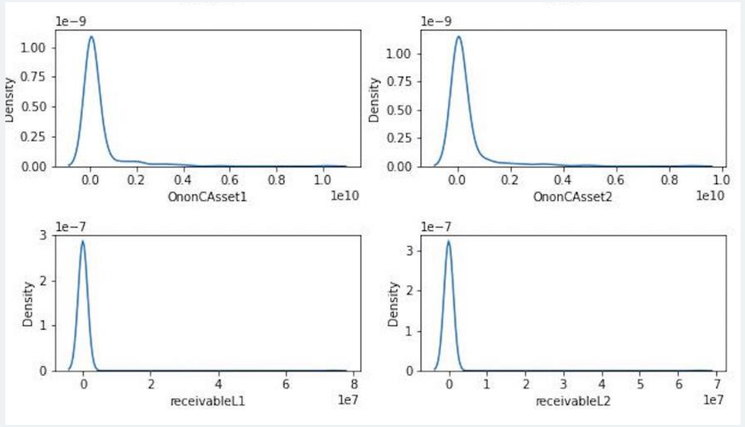
Asset(자산)



- liquidAsset : 유동자산
- quickAsset: 당좌자산
- receivableS : 미수금(단기)



- inventoryAsset : 재고자산
- nonCAsset : 비유동자산
- tanAsset : 유형자산



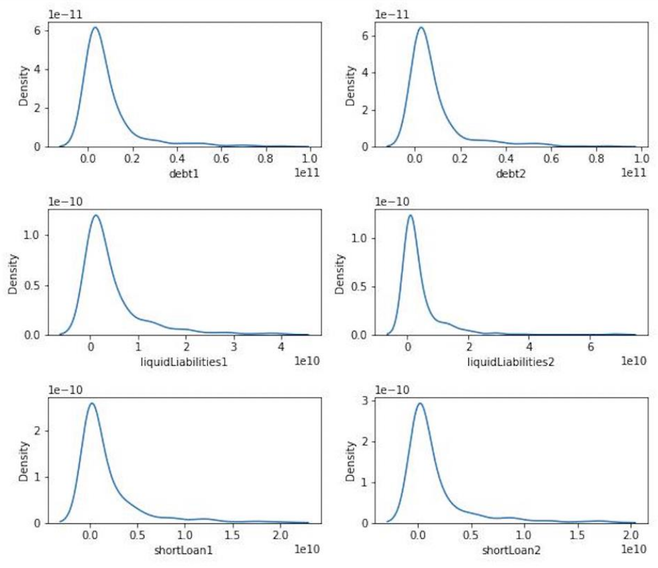
- OnonCAsset : 기타 비유동자산
- receivableL : 장기미수금

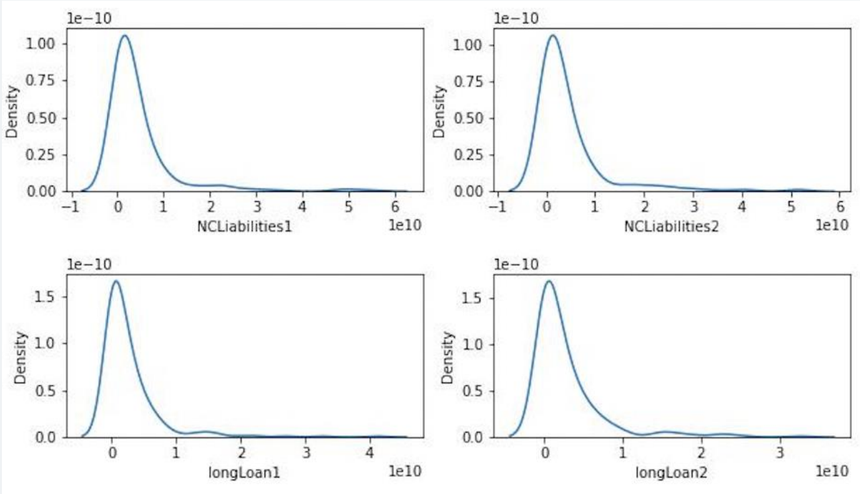
EDA

변수설명

About Numerical Variable

Debt(부채)





- debt : 부채총계
- liquidLiabilities : 유동부채
- shortLoan : 단기차입금

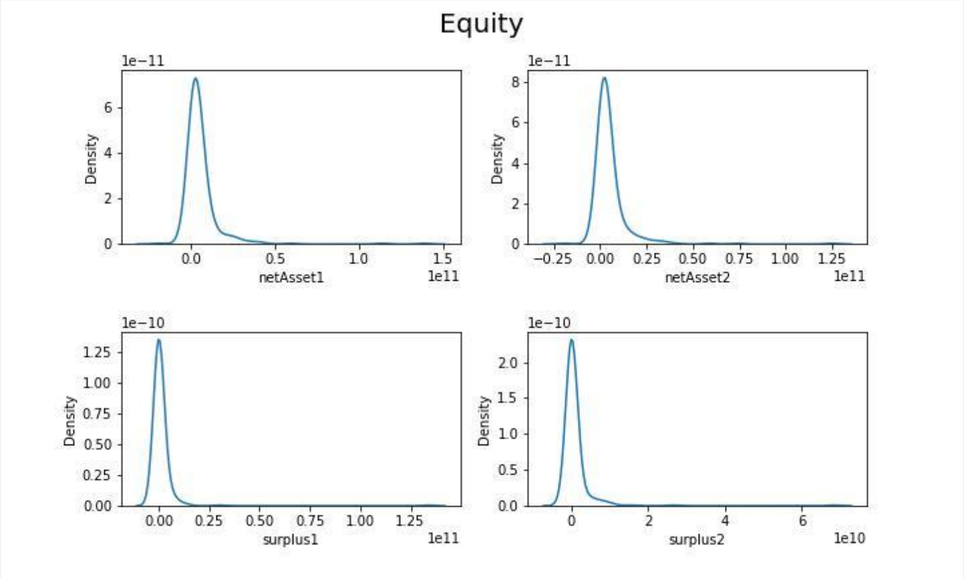
- NCLiabilities : 비유동부채
- longLoan : 장기차입금

EDA

변수설명

About Numerical Variable

Equity(자본)



- netAsset : 순자산총계
- surplus : 이익잉여금

pre-processing

Feature selection

Inst_ID & sgg 변수 삭제

Inst_id : 병원의 id번호로 target과 무관하여 생략

Sgg : 시군구 정보로 이후 one-hot encoding시 모델의 변수가 너무 많아지는 문제가 생겨 삭제

```
# 불필요한 칼럼 제거
sum.drop(['inst_id','sgg'],axis=1,inplace=True)
```

Employee 'Object'=> 'float' 자료형 변환

Employee : test 데이터에서 직원수인 'Employee'의 데이터 타입을 분석에 용이하기 위해 문자열에서 수치형 데이터로 변환

```
# test의 employee 변수를 float로 전환
test["employee1"]=test["employee1"].astype('str').str.replace(",","").astype('float')
test["employee2"]=test["employee2"].astype('str').str.replace(",","").astype('float')
```

pre-processing

Feature selection

Open_Date 개원 날짜 => 2017년 기준 운영기간

Opendate : 기존 데이터에서 주어진 개원연도보다
2017년 기준으로 얼마동안 개원을 유지했는지에 대한 정보가
개폐업 여부 예측에 유의미할 것이라고 판단하여 변경

```
train_test['openDate']=2017-(train_test['openDate']*0.0001).astype('int64')
train_test['openDate'].unique()
```

```
array([10, 20,  1, 17, 12, 15, 35, 30, 11,  9,  4, 36,  5, 23, 21, 13, 14,
       16, 24,  3,  6, 29, 19,  8,  0, 34,  7, 18, 32,  2, 33, 39, 28, 37,
       31, 25, 22, 41])
```

pre-processing

Missing value

Missing Data Type

- Zero : 0으로 채워진 데이터들
- NaN : entry 자체 내용이 없는 데이터들

["0으로 채워진 데이터들"]

Thoughts

회계 데이터에서 "0"은 실제 데이터가 0이 아니라 "결측치 " 를 의미할 수도 있겠다.

Checking

- 대부분은 feature간 회계 수식을 통해 확인이 가능했다. : 0이 결측치가 아님을 확인

But

- Debt(부채총계) + netAsset(순자산총계) : 일부 행에서 불일치
  - debt = liquidLiabilities(유동부채) + NCLiabilities(비유동부채)
  - netAsset = TotalAsset - debt(부채총계)

→수식을 통해 "0"이 결측치임을 확인과 동시에 결측치를 대체

1. 결측치 대체

1-1. 0인 값 대체

```
[ ]  #debt1 결측치(0인값) 대체
     train['debt1']=train['liquidLiabilities1']+train['NCLiabilities1']
     train['debt2']=train['liquidLiabilities2']+train['NCLiabilities2']
```

```
[ ]  #debt1 결측치(0인값) 대체
     test['debt1']=test['liquidLiabilities1']+test['NCLiabilities1']
     test['debt2']=test['liquidLiabilities2']+test['NCLiabilities2']
```

```
[ ]  #netAsset 결측치(0인값) 대체
     train['netAsset1']=train['liquidAsset1']+train['nonCAsset1']-train['debt1']
     train['netAsset2']=train['liquidAsset2']+train['nonCAsset2']-train['debt2']
```

```
[ ]  #netAsset 결측치(0인값) 대체
     test['netAsset1']=test['liquidAsset1']+test['nonCAsset1']-test['debt1']
     test['netAsset2']=test['liquidAsset2']+test['nonCAsset2']-test['debt2']
```

pre-processing

Missing value

["NaN으로 데이터가 없는 entry들"]

train.isna().sum()
test.isna().sum()

코드로 결측치를 확인해본 결과,
결측치의 패턴은 train.csv와 test.csv이 같았다.

↓

NaN 결측치 idea : train.csv와 test.csv를 동일 방식으로 처리

- 범주형 feature

Instkind(병원 종류) : employee(직원 수) 내용을 통해, 각 행의 병원 종류를 유추

  - Train set은 "Traditional_hospital"
  - test set은 "traditional_hospital"과 "general_hospital"로 대체

OwnerChange(대표자 변동) : instkind group 별 최빈값으로 대체

  - 그룹별 최빈값이 모두 "same"이었으므로, "same"으로 동일하게 대체

1-2. instkind

```
[ ] train.groupby('instkind')['employee1'].mean()
    #traditional_hospital이 직원수가 젤 비슷함
```

```
instkind
clinic                 84.764706
dental_clinic         107.000000
general_hospital      406.216216
hospital              125.488095
nursing_hospital       97.822695
traditional_clinic     79.666667
traditional_hospital   45.000000
Name: employee1, dtype: float64
```

```
[ ] #train instkind 결측치 대체
    train['instkind'].fillna('traditional_hospital', inplace=True)
```

```
#test
test.loc[test['instkind'].isna()]
#120행-> traditional_hospital
#125행-> general_hospital
```

	inst_id	OC	sido	sgg	openDate	bedCount	inst
120	413	NaN	gyeonggi	168	NaN	49.0	
125	430	NaN	jeju	76	20010201.0	NaN	

```
[ ] #test 결측치 대체
    test.loc[120,'instkind']='traditional_hospital'
    test.loc[125,'instkind']='general_hospital'
```

1-3. ownerChange

```
train.groupby('instkind')['ownerChange'].value_counts()
```

```
instkind              ownerChange
clinic                same           15
                      change          2
dental_clinic         same            1
general_hospital      same           31
                      change          6
hospital              same           71
                      change         13
nursing_hospital      same          119
                      change         20
traditional_clinic    same            3
traditional_hospital  same            7
                      change          1
Name: ownerChange, dtype: int64
```

```
[ ] #그냥 최빈값으로 대체
    train['ownerChange'].fillna('same', inplace=True)
```

```
[ ] test['ownerChange'].fillna('same', inplace=True)
```

pre-processing

Missing value

["NaN으로 데이터가 없는 entry들"]

train.isna().sum()
test.isna().sum()

코드로 결측치를 확인해본 결과,
결측치의 패턴은 train.csv와 test.csv이 같았다.

↓

NaN 결측치 idea : train.csv와 test.csv를 동일 방식으로 처리

- 수치형 feature

Instkind(병원 종류) groupby 이후,
병원 종류별 데이터들의 "median"으로 대체

1-4. 이 외 수치형 변수들

```
[ ]  train = train.groupby("instkind").apply(lambda x: x.fillna(x.median()))
     train.reset_index(drop = True, inplace = True)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nu
  """Entry point for launching an IPython kernel.
```

```
[ ]  #test의 employees float로 바꿔주기
     test["employee1"]=test["employee1"].astype('str').str.replace(",","").astype('float')
     test["employee2"]=test["employee2"].astype('str').str.replace(",","").astype('float')
```

```
[ ]  test =test.groupby("instkind").apply(lambda x: x.fillna(x.median()))
     test.reset_index(drop = True, inplace = True)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nu
  """Entry point for launching an IPython kernel.
```

pre-processing

Data Scaling

Col-min(col)

- 수치형 데이터에 대해 데이터의 스케일을 통일하기 위해 scaling을 거침
- Log 변환시 음수가 포함되어있으므로 칼럼별 최소값을 빼줌

```
# Train/Test 데이터의 수치형 변수들에 대해 scaling 진행
train_col=train_c.drop(['OC','sido','ownerChange','instkind'],axis=1)
test_col=test_c.drop(['OC','sido','ownerChange','instkind'],axis=1)
```

```
# 로그 변환 이전, Train 데이터 칼럼의 음수 값을 제거하기 위한 과정
for i in train_col.columns:
  train_col[i] -= min(train_col[i])
```

```
# 로그 변환 이전, Test 데이터 칼럼의 음수 값을 제거하기 위한 과정
for i in test_col.columns:
  test_col[i] -= min(test_col[i])
```

Log1p Scale

- 수치형 데이터들의 왜도 정도(left-skewed) 가 심하기 때문에 log변환을 진행
- 데이터에 0이 포함되어있으므로 '-inf'의 출력을 막기 위해 log1p를 사용

```
train_num_scaled_log=np.log1p(train_col)
```

```
test_num_scaled_log=np.log1p(test_col)
```

pre-processing

Outlier & Dummy Variable

Outlier

- 데이터의 개수가 적고 (301개), 0으로 표기되었던 값들을 계산식을 통해 대체하였기때문에 추가적인 이상치 제거는 진행하지 않음

Dummy Variable

- 분류 모델에 적용하기 위해 범주형 데이터를 수치형 데이터로 변환
- 이전에 Train/Test를 병합함으로써 두 데이터에 동일한 기준으로 one-hot encoding이 진행되었음
- 따라서 dummy variable 생성 이후 Train/Test 데이터를 다시 나눔

```
#범주형 변수 Dummy Variables
sum_str=sum[['OC','sido','ownerChange','instkind']]
sum_str=pd.get_dummies(sum_str)
```

```
# train, test 분리
train_str = sum_str.iloc[0:301,:]
test_str = sum_str.iloc[301:,:]
```

pre-processing

(Try)파생변수 & 변화변수

파생변수 : 기존 변수들을 이용하여 새롭게 만든 변수들 → **회계 수식 활용**

변화변수 : 변수 들 중, 1(2017년도 변수) 과 2(2016년도 변수)의 차이

[파생변수 list]

- Ordinary_income(경상이익)
- Roa(총자산순이익률)
- Rol(유동비율)
- 부채비율(debt_percent)
- 당좌비율(quickAsset_percent)
- 영업이익률(ratio_operating_income)
- 고정자산회전율(FIT)
- 자기자본 이익률(ROE)
- 재고자산회전율(inventoryAsset)

[시도해 본 경우들] – 경우를 나눠서 cross-validation을 통해서 결과 확인

- 기존 변수들 포함 여부
- 파생변수들 포함 여부
- 변화변수들 포함 여부
- Scaling 방법 변경

기존변수 + 파생변수

	전처리 과정	정확도(cv)
0	기존변수+파생변수(기본9개)&(min-max scale &log1p)	0.971169
1	기존변수+파생변수(기본9개)&(min-max scale &제곱근)	0.967662
2	기존변수+파생변수(기본9개)&(min-max scale &세제곱근)	0.974805
3	기존변수+파생변수(기본9개)&(min-max scale &네제곱근)	0.967597
4	기존변수+파생변수(기본9개)&(col-min(col)) &log1p)	0.967532
5	기존변수+파생변수(기본9개)&(col-min(col)) &제곱근)	0.963961
6	기존변수+파생변수(기본9개)&(col-min(col)) &세제곱근)	0.963896
7	기존변수+파생변수(기본9개)&(col-min(col)) &네제곱근)	0.967532
8	기존변수+파생변수(기본9개+totalAsset)&(col-min(col)) &세제곱근)	0.967597
10	기존변수+파생변수(기본9개)&(스케일링 안함)	0.974805

변화변수

	전처리 과정	정확도(cv)
0	변화변수&(스케일링 안함)	0.963961
1	변화변수&(min-max scale&log1p)	0.963961
2	변화변수&(min-max scale&세제곱근)	0.967532
3	변화변수&(min-max scale&제곱근)	0.963961
4	변화변수&(min-max scale&네제곱근)	0.964026
5	변화변수&(col-min(col)&log1p)	0.971169
6	변화변수&(col-min(col)&세제곱근)	0.967532
7	변화변수&(col-min(col)&제곱근)	0.967597
8	변화변수&(col-min(col)&네제곱근)	0.971169
9	변화변수+날짜+침대개수&(col-min(col)&log1p)	0.971234
10	변화변수+날짜+침대개수&(col-min(col)&세제곱근)	0.967532
11	변화변수+날짜+침대개수&(min-mix scale&세제곱근)	0.971234
12	변화변수+날짜+침대개수&(min-mix scale&log1p)	0.971169

기존변수 +파생변수+변화변수

13	변화변수+기존변수+파생변수&(co1-min(col)&log1p)	0.967597
14	변화변수+기존변수+파생변수&(co1-min(col)&세제곱근)	0.967597
15	변화변수+기존변수+파생변수&(min-max scale&세제곱근)	0.964026
16	변화변수+기존변수+파생변수&(min-max scale&log1p)	0.964026
17	변화변수+기존변수+파생변수&(min-max scale&네제곱근)	0.971234

[Conclusion]

표시한 부분이 cross-validation상으로는 가장 좋았다.

하지만

실제 " 데이콘" 제출을 통해 확인해 본 결과는 성능이 낮았으며,
오히려 **기존변수만 활용한 예측**의 데이콘 점수가 제일 높았다.

Modeling

모델 선택

GridSearchCV()를 이용해 8가지 classification model의 Hyperparameter tuning 후 성능 비교

```
rfr = RandomForestClassifier(random_state = 2022)
param_distribs = {"n_estimators"  : [200,400,800,1000],
                  "min_samples_split": [2,4,8,16],
                  "max_features" : [10, 12, 16, 22, 30],
                  "max_depth" : [10, 12, 16, 20, 28, 30] }
cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 2022)
grid_cv_rfr = GridSearchCV(rfr, param_grid = param_distribs, cv = cv, n_jobs = -1
                           , scoring = "accuracy")
grid_cv_rfr.fit(X, y)
```

[RandomForestClassifier]
Accuracy:0.95016

```
et = ExtraTreesClassifier(random_state = 2022)
param_distribs = {"n_estimators"  : [200,400,800,1000],
                  "min_samples_split": [2,4,8,16],
                  "max_features" : [10, 12, 16, 22, 30],
                  "max_depth" : [10, 12, 16, 20, 28, 30] }

cv=StratifiedKFold(n_splits=5, shuffle=True, random_state = 2022)
grid_cv_et = GridSearchCV(et, param_grid = param_distribs, cv = cv, n_jobs = -1
                          , scoring = "accuracy")
grid_cv_et.fit(X, y)
```

[ExtraTreeClassifier]
Accuracy:0.95349

```
model= KNeighborsClassifier()

parameters ={'algorithm':['auto'], 'leaf_size':[20,30,40], 'n_jobs': [-1,1], 'n_neighbors':[5,20,33,30],'weights':['distance','uniform']}

grid_search = GridSearchCV(model ,parameters, cv=5, scoring='accuracy')
grid_search.fit(X_train,y)
```

[KNeighborsClassifier]
Accuracy:0.95016

```
model=RidgeClassifier(random_state=2022)

parameters ={'alpha':[1,5,7], 'normalize':[True,False], 'tol': [0.001]}
grid_search = GridSearchCV(model ,parameters, cv=5, scoring='accuracy')
grid_search.fit(X,y)
```

[RidgeClassifier]
Accuracy:0.94688

Modeling

모델 선택

GridSearchCV()를 이용해 8가지 classification model의 Hyperparameter tuning 후 성능 비교

```
lgbm=LGBMClassifier(random_state=2022)
parameters ={
    'num_leaves': [20, 40, 60, 80],
    'max_depth': [-3,-1,5,10,20],
    'min_data_in_leaf': [50, 100, 200,300, 400],
}
grid_search = GridSearchCV(lgbm ,parameters, cv=5, scoring='accuracy')
grid_search.fit(X,y)
```

[LGBMClassifier]
Accuracy:0.95016

```
xgb = XGBClassifier(random_state=2022, eval_metric='logloss')
param_xgb = {'min_child_weight':[1,5,10],
             "booster":['gbtree'],
             'gamma':[0,1,2,3],
              "n_estimators": [10, 30,50,100],
              "max_depth": [3,4,5],
              "learning_rate": [0.01, 0.05, 0.1]
              }
cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=2022)
grid_search_xgb = GridSearchCV(xgb, param_xgb,
                               cv=cv)
grid_search_xgb.fit(X, y)
```

[XGBClassifier]
Accuracy:0.95016

```
param_gb = {"n_estimators": [30,50,100,300],
            "max_depth": [3,5,7],
            "learning_rate": [0.01, 0.05, 0.1],
            "subsample": [0.7, 0.9],
            }


gb = GradientBoostingClassifier(random_state=2022)
cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=2022)

grid_search_gb = GridSearchCV(gb, param_gb,
                              cv=cv, scoring='accuracy')

grid_search_gb.fit(X, y)
```

[GradientBoostingClassifier]
Accuracy:0.95016

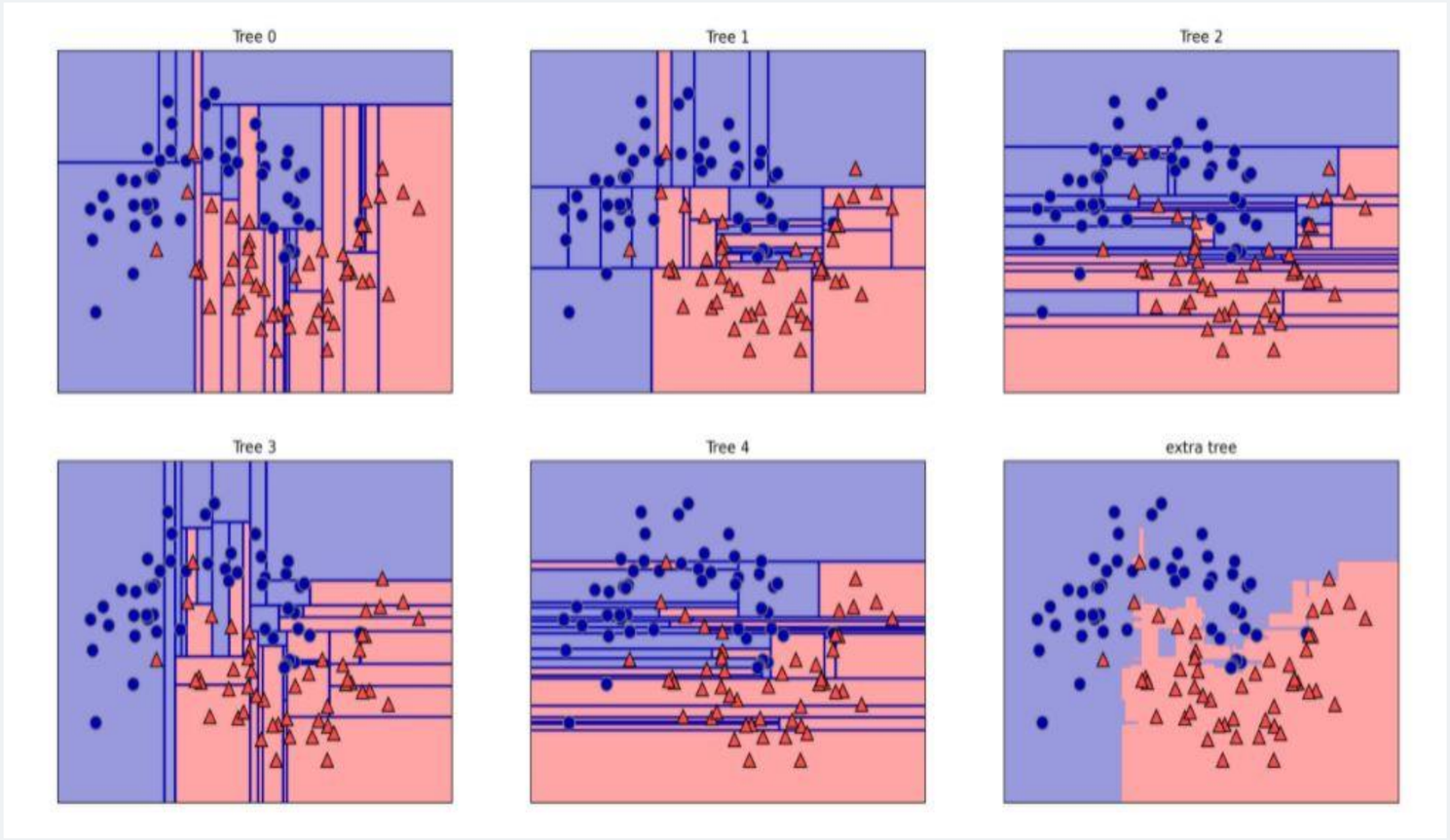
```
clf=CatBoostClassifier(random_state=2022)
parameters ={'depth'         : [4,7,8,9,10],
             'learning_rate' : [0.001,0.05,0.01],
             'iterations'    : [10,40,60,80,100]
                }
grid_search = GridSearchCV(clf ,parameters, cv=5, scoring='accuracy')
grid_search.fit(X,y)
print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )
```

[Catboostclassifier]
Accuracy:0.95016

Modeling

Extra trees Classifier

Gridseachcv 중 성능이 가장 높았던 Extra trees Classifier를 모델로 선정



Extra trees Classifier?

-앙상블 모델의 일종으로
Random forest와 비슷함

-그러나 forest tree의 각 후보 feature를
무작위(splitter ='random')로 분할

-또한 부트스트랩 샘플을 사용하지 않고
whole origin data를 사용

Modeling

Hyperparameter tuning

주요 parameter

max_depth: decision tree의 깊이
min_samples_split: 노드를 분할하기 위한 최소한의 샘플 데이터수
n_estimator: decision tree 개수
max_features: 최적의 분할을 위해 고려할 최대 feature 개수

```python
from sklearn.ensemble import ExtraTreesClassifier

et = ExtraTreesClassifier(random_state = 2022)
param_distribs = {"n_estimators"  : [100,200,600,800,900],
                  "min_samples_split": [2,5, 8,16,20],
                  "max_features" : [5, 10, 15, 22, 30],
                  "max_depth" : [10, 12, 16, 24, 28, 33],
                  "verbose" : [2]}

cv=StratifiedKFold(n_splits=5, shuffle=True, random_state = 2022)
grid_cv_et = GridSearchCV(et, param_grid = param_distribs, cv = cv, n_jobs = -1
                          , scoring = "accuracy")
grid_cv_et.fit(X, y)

print("best_model:", grid_cv_et.best_estimator_)
print("best_score:", grid_cv_et.best_score_)
```

Tuning 최종 결과

max_depth=10

min_samples_split=8

max_features=5

n_estimators=800

최종 결과

Test data 적용

Gridsearchcv 후 최적 모델에 Test data 적용 후 데이콘 제출

178	알록달록		0.85714

Accuracy : 0.85714 로 생각보다 낮은 성능..

Modeling

모델 선택

AutoML-Pycaret

Top5

1. K Neighbors Classifier
2. Dummy Classifier
3. Random forest Classifier
4. Extra trees Classifier
5. Catboost Classifier

```
top5_model = compare_models(
                round=4,
                sort="Accuracy",
                n_select = 5)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
knn	K Neighbors Classifier	0.9524	0.2744	1.0000	0.9524	0.9753	NaN	0.0000	0.1460
dummy	Dummy Classifier	0.9524	0.3333	1.0000	0.9524	0.9753	NaN	0.0000	0.0107
rf	Random Forest Classifier	0.9476	0.3615	0.9952	0.9524	0.9728	NaN	0.0000	0.4780
et	Extra Trees Classifier	0.9476	0.3410	0.9952	0.9524	0.9728	NaN	0.0000	0.4753
catboost	CatBoost Classifier	0.9429	0.3436	0.9901	0.9520	0.9702	NaN	-0.0051	9.7707
svm	SVM - Linear Kernel	0.9381	0.0000	0.9846	0.9512	0.9671	NaN	-0.0127	0.0167
gbc	Gradient Boosting Classifier	0.9381	0.3590	0.9853	0.9520	0.9677	NaN	-0.0051	0.2067
xgboost	Extreme Gradient Boosting	0.9381	0.3795	0.9853	0.9520	0.9677	NaN	-0.0051	1.0633
lightgbm	Light Gradient Boosting Machine	0.9381	0.3692	0.9853	0.9520	0.9677	NaN	-0.0051	0.0773
ridge	Ridge Classifier	0.9286	0.0000	0.9751	0.9513	0.9624	NaN	-0.0154	0.0133
ada	Ada Boost Classifier	0.9190	0.3487	0.9648	0.9505	0.9569	NaN	-0.0229	0.1240
lr	Logistic Regression	0.9095	0.2256	0.9495	0.9545	0.9513	NaN	0.0121	0.6893
lda	Linear Discriminant Analysis	0.8810	0.3436	0.9253	0.9490	0.9358	NaN	-0.0407	0.0187
dt	Decision Tree Classifier	0.8429	0.3000	0.8857	0.9469	0.9120	NaN	-0.0494	0.0180
nb	Naive Bayes	0.6762	0.3949	0.6799	0.9703	0.7939	0.0767	0.1055	0.0253

최종 결과

Test data 적용

AutoML을 이용해 얻은 Top4의 모델
KNeighborClassifier
RandomForestClassifier
ExtraTreesClassifier
CatBoostClassifier

stacking 적용하여 결과 확인

```
1 tuned_top5 = [tune_model(i,choose_better = True) for i in top5_model]
2 tuned_top5
3 blend5_stack = stack_models(estimator_list=tuned_top5)
4 final_model = finalize_model(blend5_stack)
5 prediction = predict_model(final_model, data = X_test)
6 prediction['Label'].value_counts()
7 submission['OC']=pd.DataFrame(prediction['Label'])
```

58	짹짹이얌		0.88888

Accuracy 0.88888로 58등!

보완점 및 Q&A

보완점

1. Tran data을 이용한 cross-validation의 높은 성능이 데이콘의 고득점과 반드시 연결되는 것은 아니다(오버피팅의 영향 등)

2. Target 분포의 불균형으로 smote 를 활용했으나 성능이 더 낮아지거나 효과가 없었다.

3. Data의 개수가 적어 처리하기 까다로웠다.

4. 회계,재무 domain 지식을 활용하여 여러 파생변수를 활용해보았지만 좋은 성능으로 이어지지 않았다.

감사합니다 :)

Q & A