

# 전국 지역단위 소상공인(SOHO) 폐업예측

금융 데이터를 공간 단위로 재가공된 지역별 SOHO 생존율을 예측하는 모델

발표자 : 이 한

# PROJECT 배경 및 목적

- 신용평가 기관이 제공한 전국 지역별 SOHO현황을 지역별 금융데이터로 분석하여 지역별 12월 신규 창업 대비 폐업률을 예측하는 모델 제작
- 이해관계자들의 의사결정에 도움
- 신용평가 및 리스크 관리에 활용
- 결과물은 결정계수(Coefficient of Determination,  $R^2$ )로 평가

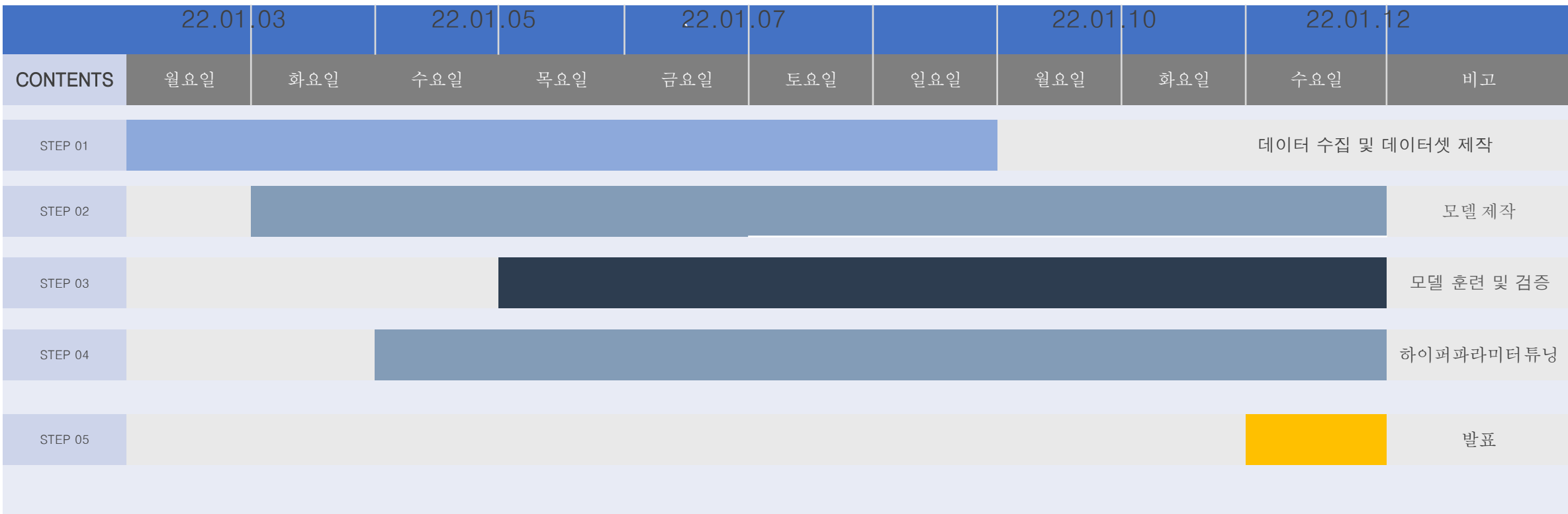
# PROJECT TIMETABLE

## 전국 소상공인(SOHO) 창업대비 폐업비율 예측

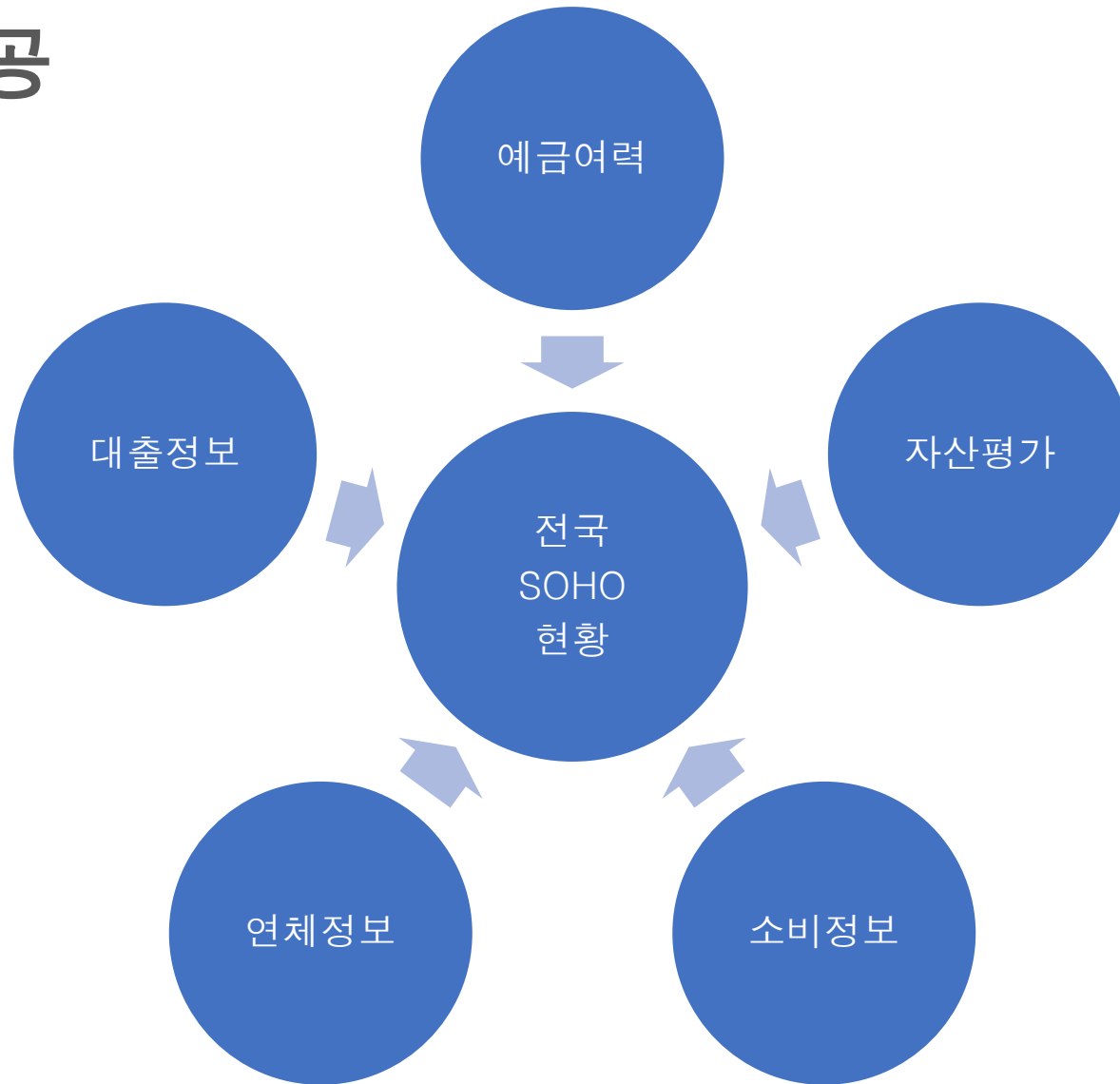
2021년 1월부터 11월까지의 전국 시도단위 SOHO 데이터를 바탕으로

2022년 지속 영업 가능성 예측 모델

### TIME LINE



# 데이터셋 가공



# 전국 SOHO 데이터

- – 집계 기간: 2021.01 ~ 2021.11
- – 집계 차원: 전국, 시도, 업종(대분류) (※ 10차 한국표준산업분류 기준)
- 1. 사업체 수(현재, 폐업, 창업)
- 2. 업력(기준: 6개월, 3년, 5년)
- \* 현재 업체수(개)
- \* 폐업 사업체수(개)
- \* 창업 사업체수(개)
- \* **창업 사업체 대비 폐업 비율**
- \* 6개월 이하 업력 사업체수(개)
- \* 6개월 초과 3년 이하 업력 사업체수(개)
- \* 3년 초과 5년 이하 업력 사업체수(개)
- \* 5년 초과 업력 사업체수(개)

# 예금여력

1. 월평균 예금여력 금액(소비가능금액) : 해당지역 전체 고객군의 월평균 예금여력금액 (= 소비가능금액)
2. 연평균 소득 금액 : 해당지역 전체 고객군의 평균 KCB 결정 연소득 금액
3. 월평균 카드소비 금액 : 해당지역 전체 고객군의 월평균 카드소비 금액 (\*신용, 체크 등 모든 카드이용금액)
4. 월평균 채무상환 금액 : 해당지역 전체 고객군의 월평균 채무상환 금액

# 자산평가

1. 월평균 총자산 평가금액 : 해당지역 고객들이 보유한 주택의 총자산 평균평가금액
2. 월평균 순자산 평가금액 : 해당지역 고객들이 보유한 주택의 총자산 평균평가금액
3. 자가거주비율 : 해당지역 고객군의 자가주택 거주 비율
4. 다주택자수 비율 : 해당지역 고객군의 다주택자수(보유주택 2채 이상) 비율
5. 아파트 거주 비율 : 해당지역 고객군의 아파트 거주 비율

## 소비정보

1. 카드보유자수(명)
2. 평균카드개수(개)
3. 평균카드한도금액(원)
4. 카드이용금액(원)
5. 신용판매이용금액(원)
6. 일시불이용금액(원)
7. 할부이용금액(원)
8. 현금서비스이용금액(원)
9. 해외소비금액(원)

## 연체정보

1. 전체연체보유자수(명)
2. 평균연체건수(건)
3. 평균연체일수(일)
4. 평균연체금액(천원)
5. 중위연체금액(천원)

## 대출정보

1. 대출보유자수(명)
2. 평균대출건수(건)
3. 전체대출잔액(천원)
4. 평균대출잔액(천원)

# 데이터 개요

DATASET : 소득 정보  
소비 정보  
대출 정보  
연체 정보  
예금 여력  
자산 평가  
soho 정보


총 42개 특성 187행 (187, 42)

TARGET : 창업 사업체 대비 폐업 비율

```
# Data columns (total 42 columns):
# #      Column      Non-Null Count  Dtype
# ---  -
# 0     STD_YM          187 non-null    int64
# 1     BLCK_SP_CD      187 non-null    int64
# 2     CTPV_CD         187 non-null    int64
# 3     CTPV_NM         187 non-null    object
# 4     AVG_ICYR_AMT    187 non-null    int64
# 5     MED_ICYR_AMT    187 non-null    int64
# 6     POP_CT          187 non-null    int64
# 7     CARD_HLDR_CT    187 non-null    int64
# 8     AVG_CARD_CNT    187 non-null    int64
# 9     AVG_CARD_LMT_AMT 187 non-null    int64
# 10    CARD_USE_AMT    187 non-null    int64
# 11    CRD_SLE_USE_AMT 187 non-null    int64
# 12    LSP_USE_AMT     187 non-null    int64
# 13    INSTL_USE_AMT   187 non-null    int64
# 14    CASH_SVC_SUE_AMT 187 non-null    int64
# 15    OVSEA_CSMP_AMT  187 non-null    int64
# 16    LN_HLDR_CT      187 non-null    int64
# 17    AVG_LN_CONT     187 non-null    int64
# 18    ALL_LN_BLC      187 non-null    int64
# 19    AVG_LN_BLC      187 non-null    int64
# 20    ALL_ARR_HLDR_CT 187 non-null    int64
# 21    AVG_ARR_CONT    187 non-null    int64
# 22    AVG_ARR_DACT    187 non-null    int64
# 23    AVG_ARR_AMT     187 non-null    int64
# 24    MED_ARR_AMT     187 non-null    int64
# 25    DEPOSIT_AMT     187 non-null    int64
# 26    AVG_ICYR_ICM    187 non-null    int64
# 27    CARD_USE_AMT.1  187 non-null    int64
# 28    RE_DEBT_AMT     187 non-null    int64
# 29    TOT_ASST_AMT    187 non-null    int64
# 30    NET_ASST_AMT    187 non-null    int64
# 31    ONW_HOUS_RATIO  187 non-null    float64
# 32    PLU_HOUS_RATIO  187 non-null    float64
# 33    APT_RES_RATIO   187 non-null    float64
# 34    OPN_CNT         187 non-null    int64
# 35    CLSD_CNT        187 non-null    int64
# 36    CLSD_RATIO      187 non-null    float64
# 37    NEW_CNT         187 non-null    int64
# 38    MANAGE01_CNT    187 non-null    int64
# 39    MANAGE02_CNT    187 non-null    int64
# 40    MANAGE03_CNT    187 non-null    int64
# 41    MANAGE04_CNT    187 non-null    int64
# dtypes: float64(4), int64(37), object(1)
```



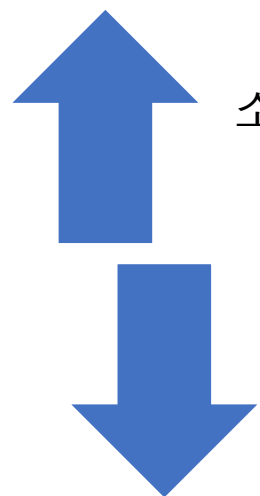
# 데이터 개요

	column 8	column 9	column 10	column 11	column 12	column 13	column 14	column 15	column 16	column 17	column 18	column 19	
1	CRD_HLDR_CT	AVG_CARD_CT	AVG_CARD_LMT	CARD_USE_AMT	CRD_SLE_USE_AMT	LSP_USE_AMT	INSTL_USE_AMT	CASH_SVC_SUE_AMT	OVSEA_CSMP_AMT	LN_HLDR_CT	AVG_LN_CON	ALL_LN_I	
2	1584	4	11914	12483074618	11648762470	9660057219	1988705251	834312148	173264981	3087332	2	2800240	
3	6481	4	11097	3756682618	3451578664	2738934254	712644410	305103954	27124725	1006150	2	8244479	
4	7871	4	10600	2699121077	2514701176	2032202314	482498863	184419901	18306312	729004	2	5789373	
5	3481	4	10532	3202585467	2948791409	2367523639	581267770	253794058	26937705	980611	2	7772645	
6	6403	4	10717	1750410616	1639884305	1340030945	299853360	110526311	8790534	459717	2	3307032	
7	1859	4	10698	1796183865	1680965322	1390893603	290071719	115218543	11977095	470004	2	3266362	
8	571	4	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>		column 32	column 33	column 34	column 35	column 36	column 37	column 38	column 39	column 40
9	739	5		1	ONW_HOUS_RATIO	PLU_HOUS_RATIO	APT_RES_RATIO	OPN_CNT	CLSD_CNT	CLSD_RATIO	NEW_CNT	MANAGE01_CNT	MANAGE02_CNT
10	74316	4		2	0.15	0.04	0.46	1061747	5797	0.5460	8358	56625	274012
11	3026	3		3	0.18	0.05	0.58	323508	1743	0.5388	2660	16697	82635
12	1410	3		4	0.21	0.05	0.58	245658	1239	0.5044	2035	12649	65335
13	0710	3		5	0.19	0.04	0.56	254347	1593	0.6263	2444	16007	77454
14	9746	4		6	0.22	0.05	0.65	135402	812	0.5997	1315	8166	40231
15	8739	3		7 	0.2	0.05	0.57	134922	746	0.5529	1178	7531	37314
16	8071	3		8	0.23	0.05	0.57	92788	648	0.6984	923	5578	25797
17	0539	4		9	0.15	0.06	0.72	25585	139	0.5433	306	2041	9439
18	199	4		10	0.18	0.04	0.57	1213299	7007	0.5775	12621	77672	365500
19	1839	4		11	0.2	0.05	0.44	152424	808	0.5301	1200	7968	41387
20	6745	4		12	0.21	0.05	0.48	146757	780	0.5315	1312	8408	40918
21	8709	4		13	0.19	0.05	0.46	196833	1072	0.5446	1725	11485	55856
22	4304	4		14	0.21	0.04	0.49	166647	959	0.5755	1433	9165	45912
23	7356	4		15	0.2	0.04	0.39	164084	830	0.5058	1509	9355	45641
24	2364	4		16	0.2	0.05	0.43	247513	1221	0.4933	2024	12956	66589
25	842	4		17	0.22	0.05	0.5	304357	1648	0.5415	2728	16824	81818



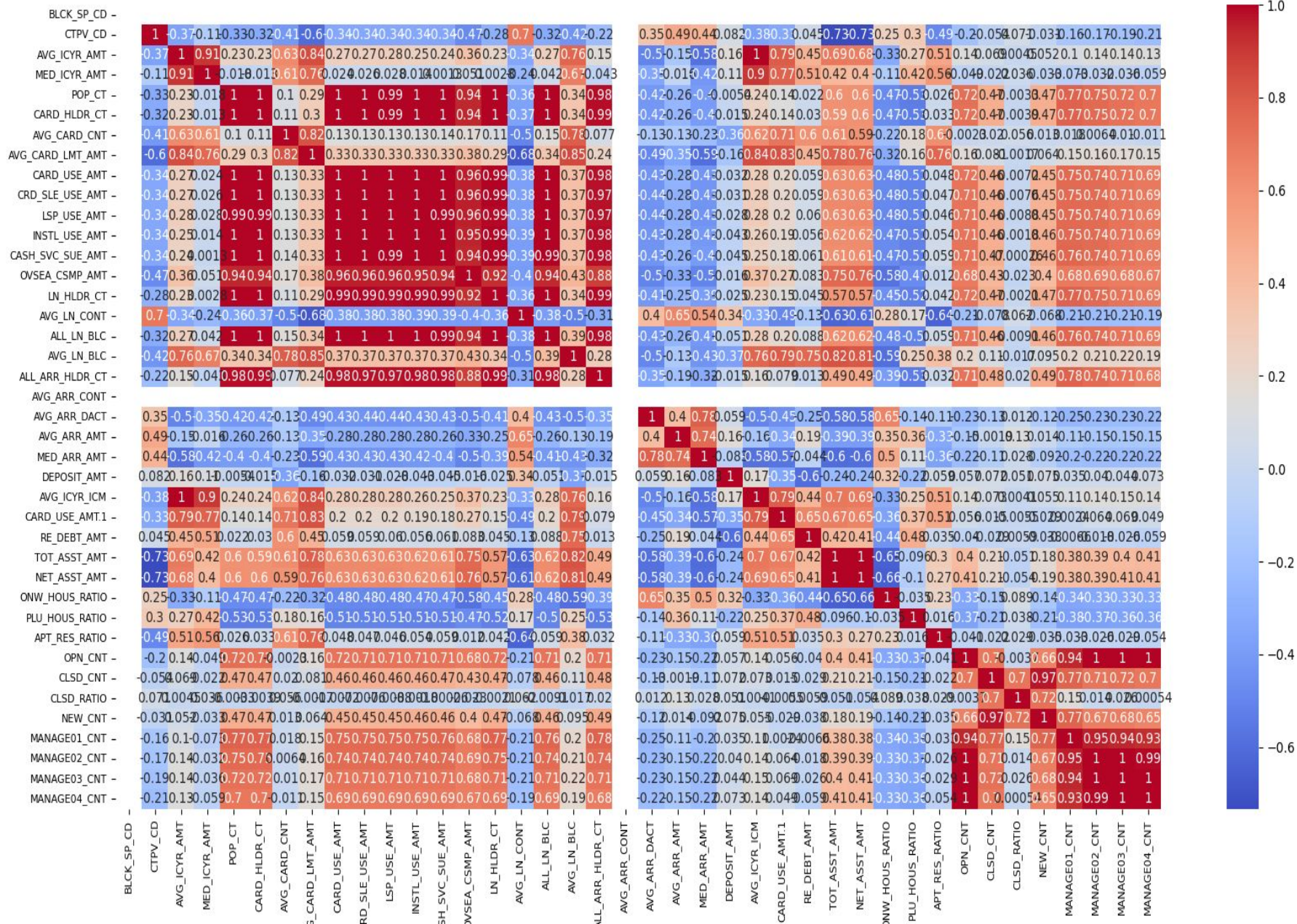
# 피처 상관관계

## Correlation between Features

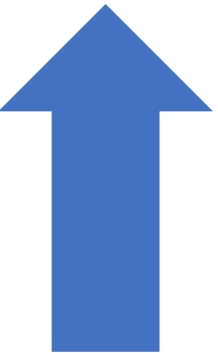



소비데이터

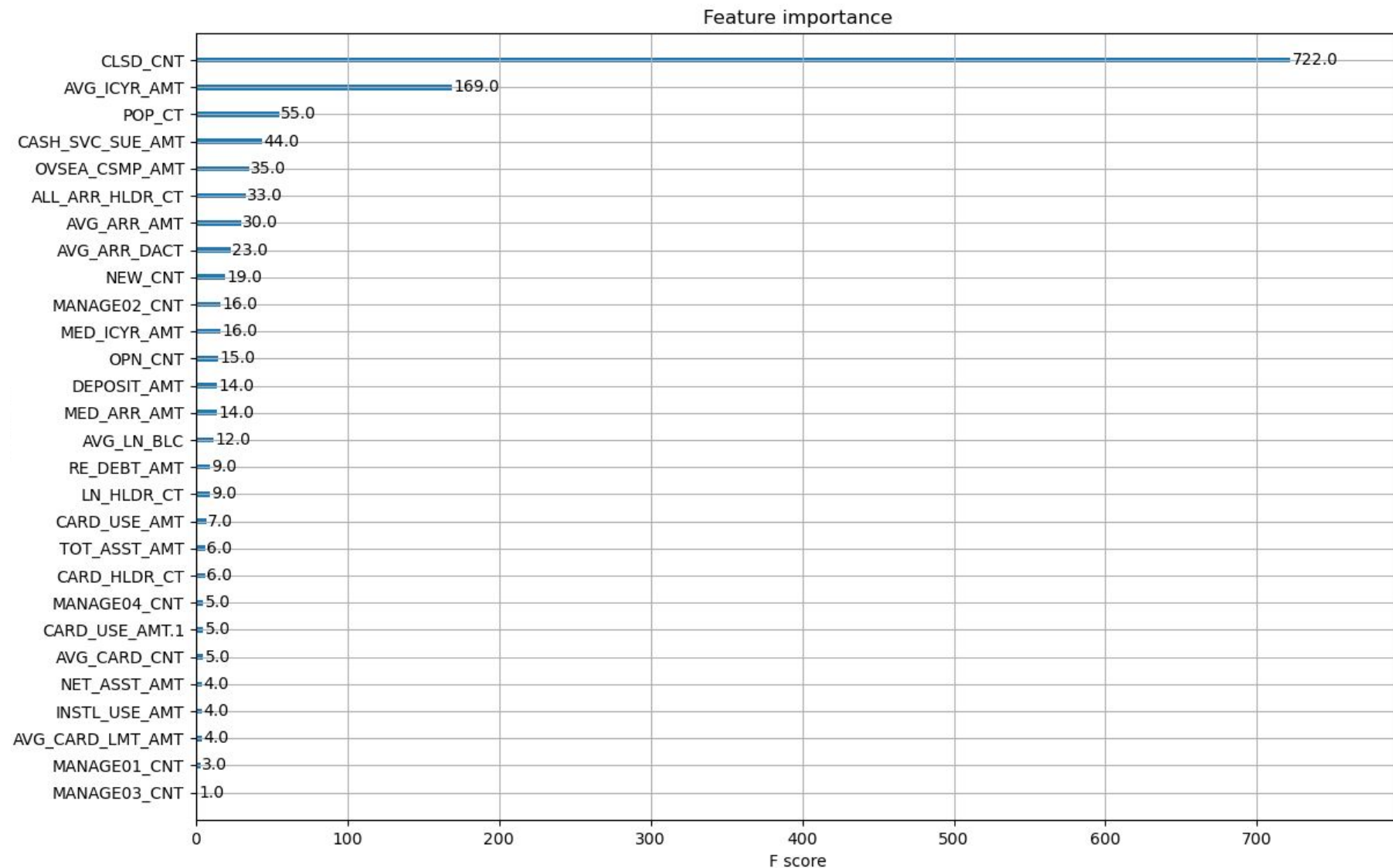
자산데이터



# 피쳐 중요도

- 
1. 폐업장수
  2. 평균소득
  3. 인구

- 
1. 평균카드한도금액
  2. 6개월 이하업장
  3. 3년 초과업장



```
x = dataset.drop(['STD_YM', 'CTPV_NM', 'BLCK_SP_CD', 'CTPV_CD', 'ONW_HOUS_RATIO', 'PLU_HOUS_RATIO', 'APT_RES_RATIO', 'MED_ARR_AMT', 'CLSD_CNT', 'CLSD_RATIO'], axis=1)
y = dataset['CLSD_RATIO']
```



# 데이터 전처리

1. 데이터 표준화
2. 결측치 제거
3. 중복값 제거
4. 정규분포(로그변환)

```
#1-1 데이터표준화
mean = np.mean(x_train, axis=0)
std = np.std(x_train, axis=0)

x_train = (x_train - mean) / std
x_test = (x_test - mean) / std

#1-2 결측치확인
def check_missing_col(dataframe):
    counted_missing_col = 0
    for i, col in enumerate(dataframe.columns):
        missing_values = sum(dataframe[col].isna())
        is_missing = True if missing_values >= 1 else False
        if is_missing:
            counted_missing_col += 1
            print(f'결측치가 있는 컬럼은: {col}입니다')
            print(f'총 {missing_values}개의 결측치가 존재합니다.')

    if i == len(dataframe.columns) - 1 and counted_missing_col == 0:
        print('결측치가 존재하지 않습니다')

check_missing_col(dataset)

#결측치가 존재하지 않습니다

#1-3 중복값 제거
print(dataset.duplicated())
dataset.drop_duplicates(inplace=True)

#1-4 로그변환
x = np.log1p(x)
```

# 모델 #1. Simple RNN

```
x_train, x_val, y_train, y_val = train_test_split(x_train,
| | | | | y_train, test_size=0.3, random_state=42)

scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

model = Sequential()
model.add(Dense(200, input_dim=34))
model.add(Dense(130, activation='relu'))
model.add(Dense(130, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(80, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(1))
model.summary()

#3. 컴파일, 훈련
model.compile(loss='mae', optimizer = 'adam') #, metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
es = EarlyStopping(monitor='val_loss', patience=50,
| | | | | mode='auto', verbose=1, restore_best_weights=True)
mcp = ModelCheckpoint (monitor = 'val_loss', mode = 'min', verbose = 1,
| | | | | save_best_only=True,
| | | | | filepath = './_ModelCheckPoint/keras27_1_MCP.hdf5')
model.fit(x_train, y_train, epochs=500,
| | | | | batch_size=32, validation_split=0.3, callbacks=[es, mcp])
```

## 심플RNN

LOSS	0.003939
예측값	0.685633
R2스코어	0.998
훈련시간	116.696 초

```
...
loss: 0.0039396812207996845
예측값 : [0.6856333]
r2 스코어 : 0.998
걸린시간: 116.696 초
...
```

## 모델 #2. LSTM

#2. 모델구성

```
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(32,1)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(180, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(160, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(140, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(120, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(80, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(50, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1))
```

### LSTM

LOSS	0.04169
예측값	0.72781
R2스코어	0.879
훈련시간	422.75 초

...

loss: 0.04169400781393051

예측값 : [0.72781277]

r2 스코어 : 0.879

걸린시간: 422.75 초

...



## 모델 #3. 머신러닝

```
#2. 모델
model1 = RandomForestRegressor(n_estimators = 100,
                               max_depth=10, min_samples_split=30,
                               min_samples_leaf =40)
model2 = GradientBoostingRegressor(n_estimators = 1000,
                                   learning_rate = 0.3, max_depth=100,
                                   min_samples_split=40,
                                   min_samples_leaf =30)
model3 = ExtraTreesRegressor(n_estimators=1000,
                             max_depth=16, random_state=7)
model4 = AdaBoostRegressor(n_estimators=1000, random_state=7)
model5 = XGBRegressor(n_estimators = 1000,
                     learning_rate = 0.3, max_depth=100, min_samples_split=40,
                     min_samples_leaf =30)
model6 = LGBMRegressor(n_estimators = 100,
                      learning_rate = 0.1, max_depth=10, min_samples_split=40,
                      min_samples_leaf =30)
model7 = CatBoostRegressor(n_estimators=1000, max_depth=16, random_state=7)

from sklearn.ensemble import VotingClassifier
voting_model = VotingClassifier(estimators=[('RandomForestRegressor', model1),
                                           ('GradientBoostingRegressor', model2),
                                           ('ExtraTreesRegressor', model3),
                                           ('AdaBoostRegressor', model4),
                                           ('XGBRegressor', model5),
                                           ('LGBMRegressor', model6),
                                           ('CatBoostRegressor', model7)])

classifiers = [model1,model2,model3,model4,model5,model6,model7]
from sklearn.metrics import r2_score, mean_squared_error
```

```
=====RandomForestRegressor =====
modle_socore: 0.04976779230494499
r2 스코어: 0.05
예측값 : 0.8570676629063514
loss=mse : 0.014576354266724335
=====GradientBoostingRegressor =====
modle_socore: 0.6068622574732012
r2 스코어: 0.607
예측값 : 0.7926910076088625
loss=mse : 0.006030646997949253
=====ExtraTreesRegressor =====
modle_socore: 0.9999083819271255
r2 스코어: 1.0
예측값 : 0.6863492000000057
loss=mse : 1.4054012026092653e-06
=====AdaBoostRegressor =====
modle_socore: 0.9879261636250195
r2 스코어: 0.988
예측값 : 0.6807406250000001
loss=mse : 0.0001852100096534052
=====XGBRegressor =====
modle_socore: 0.9988428001784639
r2 스코어: 0.999
예측값 : 0.6835744
loss=mse : 1.7751192202814536e-05
=====LGBMRegressor =====
modle_socore: -1.0656474168353456
r2 스코어: -1.066
예측값 : 0.9815852242167775
loss=mse : 0.03168657965295852
Learning rate set to 0.029661
0: learn: 1.0071496 total: 171ms remaining: 2m 50s
1: learn: 0.9995311 total: 191ms remaining: 1m 35s
...
=====CatBoostRegressor =====
modle_socore: 0.7908870138667055
r2 스코어: 0.791
예측값 : 0.6548057439273343
loss=mse : 0.0032077474778982673
걸린시간: 115.305 초
```

# 검증모델 #.K-FOLD

```
x_train, x_test, y_train, y_test = train_test_split(x,y,
                                                    train_size =0.7, shuffle=True, random_state = 42)

kfold = KFold(n_splits=2, shuffle=True)

for train, test in kfold.split(x, y):

    model = Sequential()
    model.add(LSTM(200, activation='relu', input_shape=(32,1)))
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(180, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(160, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(140, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(120, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(80, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(30, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(1))
```

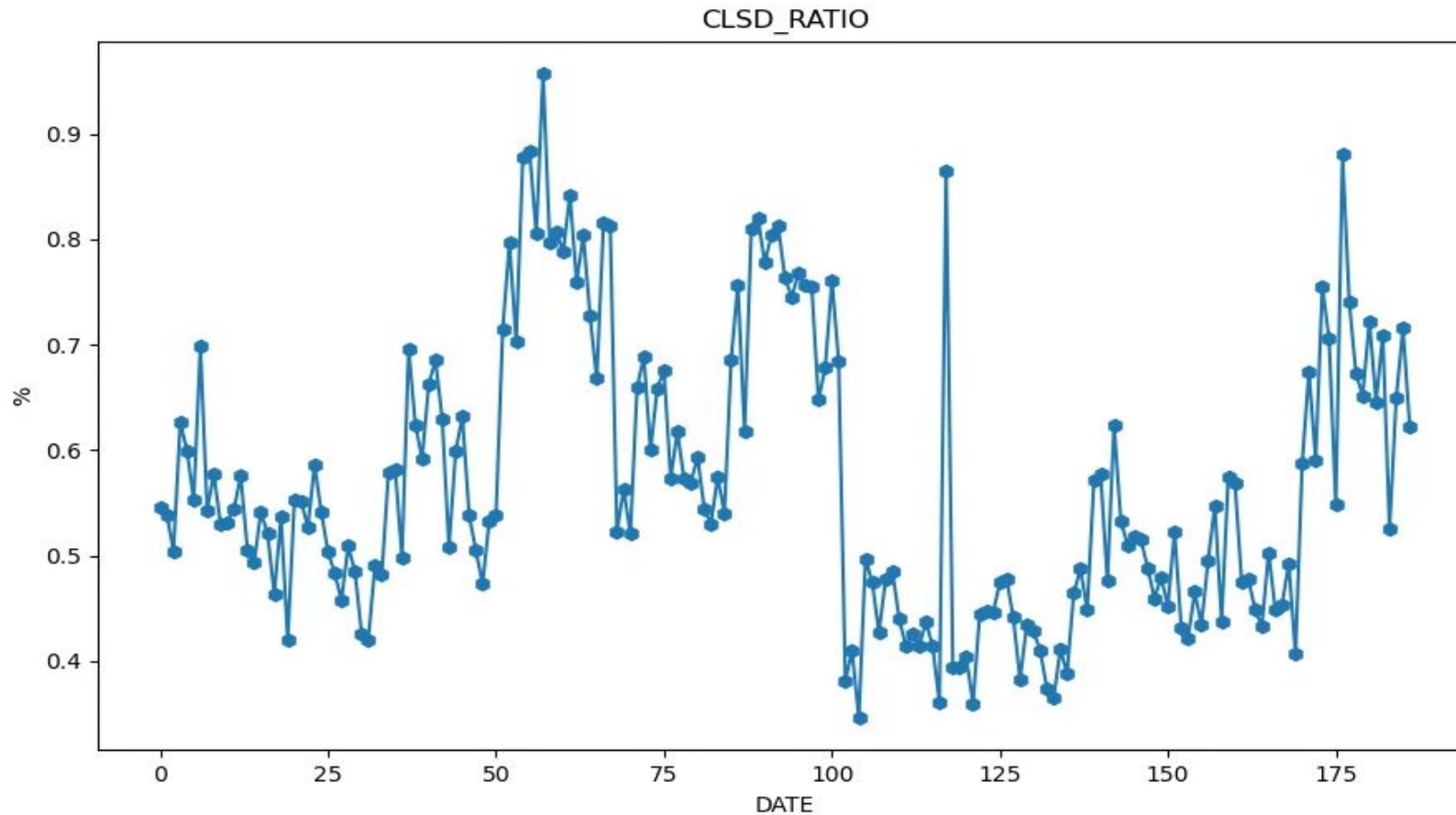
## LSTM K-FOLD 검증

LOSS	0.0252248
예측값	0.7060925
R2스코어	0.936
훈련시간	63.444 초

```
예측값 : [0.7060925]
cv5: 0.025224849581718445
r2 스코어 : 0.936
걸린시간: 63.444 초
```



# 신규 창업 대비 폐업 비율

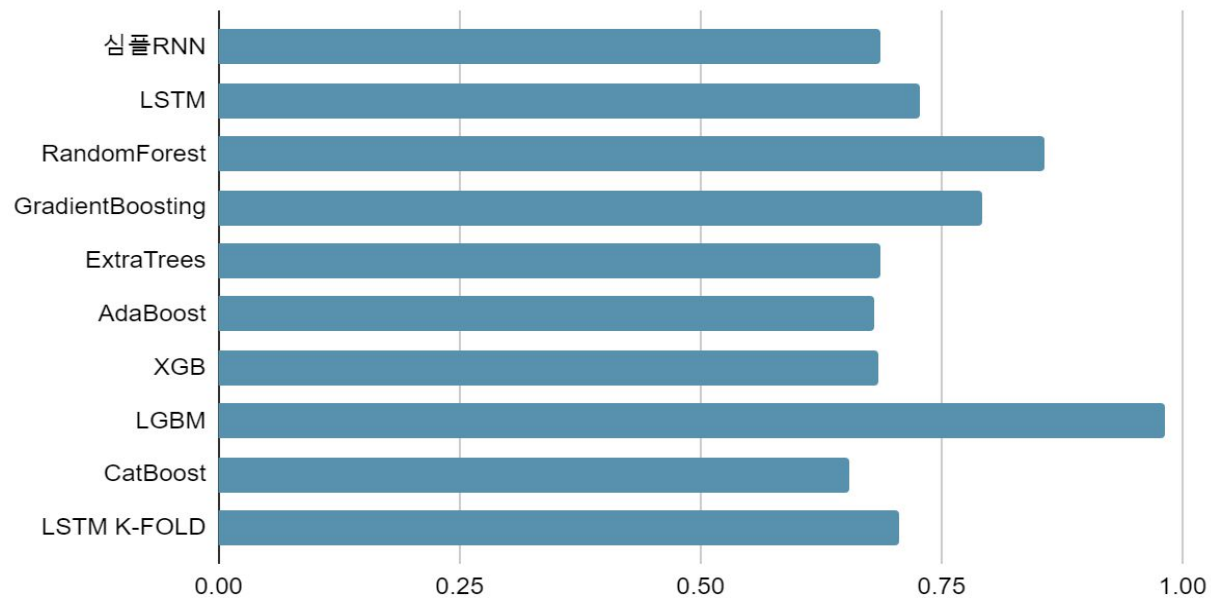


1 <= 예측값  
폐업보다 창업이 더 많다

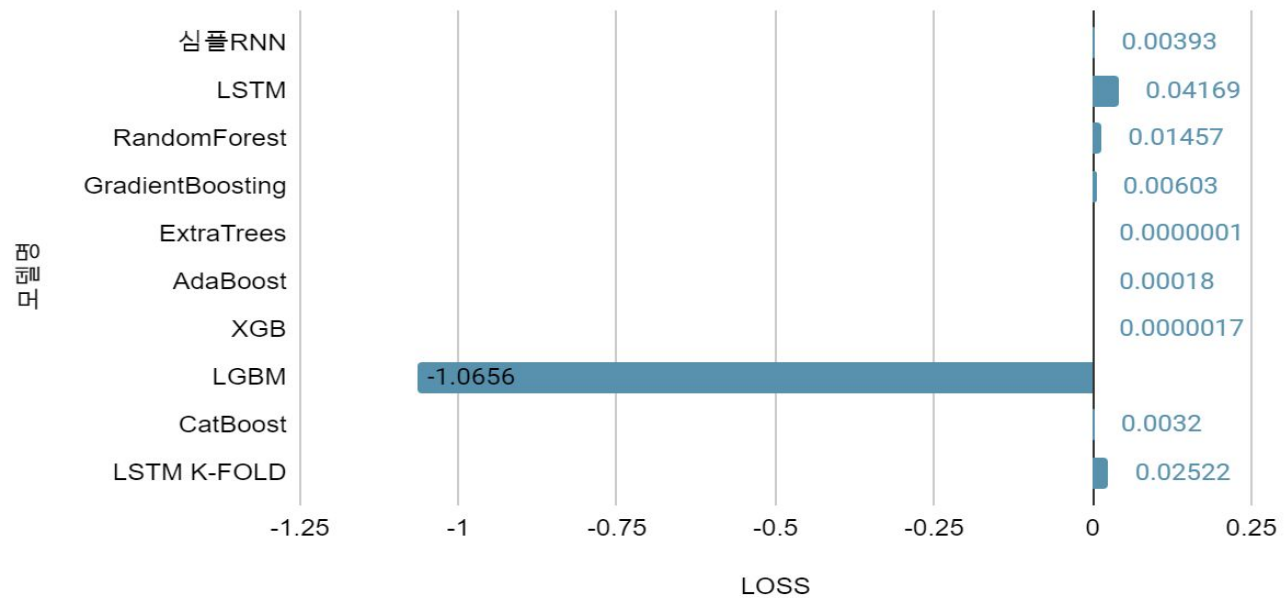
# 모델별 결과 비교

모델명	예측값	LOSS	R2 스코어	시간	비고
심플RNN	0.6856	0.00393	0.998	116.69 초	
LSTM	0.7278	0.04169	0.879	422.75 초	
RandomForest	0.8570	0.01457	0.05	115.30 초	
GradientBoosting	0.7926	0.00603	0.607		
ExtraTrees	0.6863	0.0000001	1.0		
AdaBoost	0.6807	0.00018	0.988		
XGB	0.6835	0.0000017	0.999		
LGBM	0.9815	-1.0656	-1.066		
CatBoost	0.6548	0.00320	0.791		
LSTM K-FOLD	0.7060	0.02522	0.936	63.44 초	

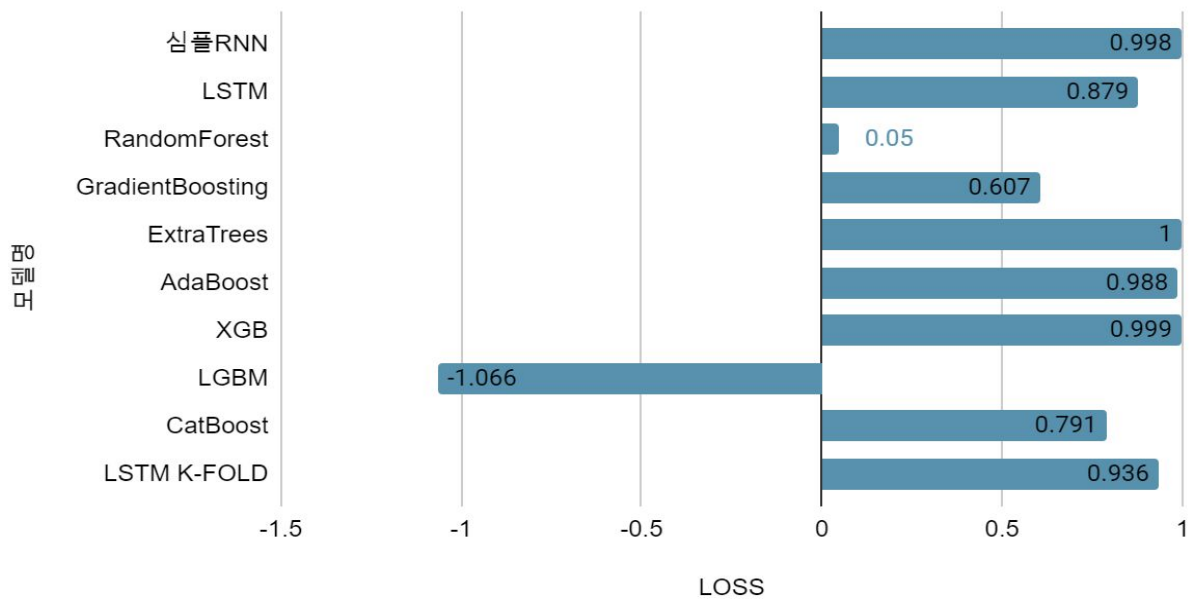
### 모델별 예측값



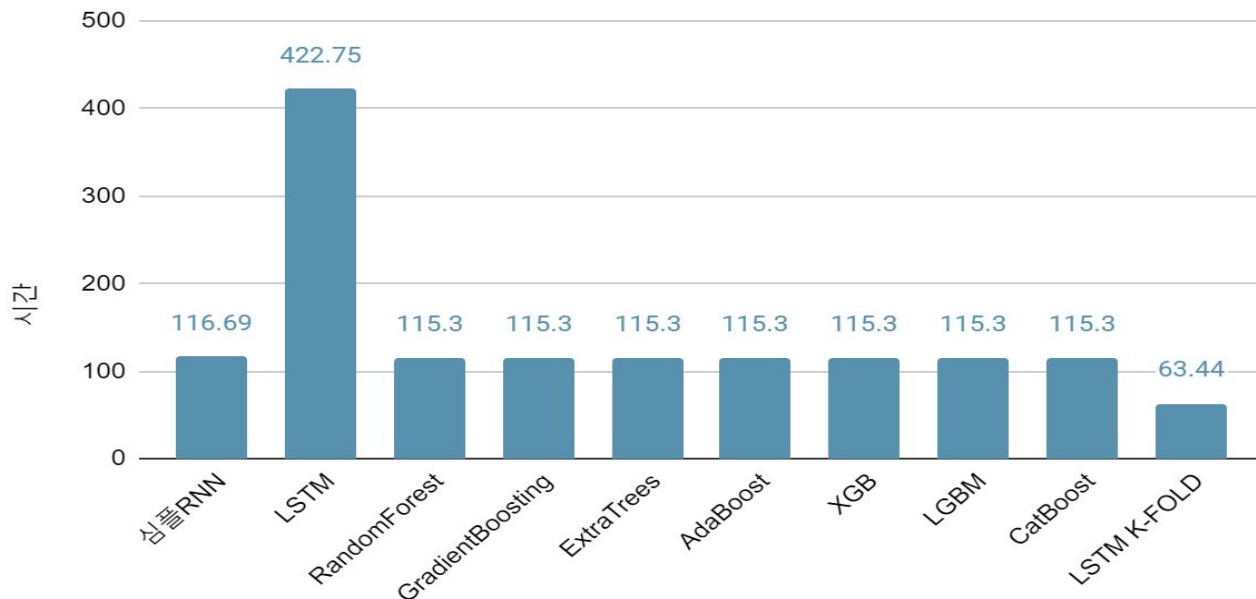
### 모델별 LOSS 값



### 모델별 R2스코어 값



### 모델별 훈련 시간



결론

Q&A 질문과 답변