DACON

FIFA 선수 이적료 예측

이장춘 , 이도훈 , 이정원 ,이지원

Tap to Start

CONTENTS

데이터 소개

In [4]:
```
data.head()
```

Out[4]:

	id	name	age	continent	contract_until	position	prefer_foot	reputation	stat_overall	stat_potential	stat_skill_moves	value
0	0	L. Messi	31	south america	2021	ST	left	5.0	94	94	4.0	110500000.0
1	3	De Gea	27	europe	2020	GK	right	4.0	91	93	1.0	72000000.0
2	7	L. Suárez	31	south america	2021	ST	right	5.0	91	91	3.0	80000000.0
3	8	Sergio Ramos	32	europe	2020	DF	right	4.0	91	91	3.0	51000000.0
4	9	J. Oblak	25	europe	2021	GK	right	3.0	90	93	1.0	68000000.0

id : 선수 고유의 아이디
name : 이름
age : 나이
continent : 선수들의 국적이 포함되어 있는 대륙입니다
contract_until : 선수의 계약기간이 언제까지인지 나타내어 줍니다
position : 선수가 선호하는 포지션입니다. ex) 공격수, 수비수 등
prefer_foot : 선수가 선호하는 발입니다. ex) 오른발
reputation : 선수가 유명한 정도입니다. ex) 높은 수치일 수록 유명한 선수
stat_overall : 선수의 현재 능력치 입니다.
stat_potential : 선수가 경험 및 노력을 통해 발전할 수 있는 정도입니다.
stat_skill_moves : 선수의 개인기 능력치 입니다.
value : FIFA가 선정한 선수의 이적 시장 가격 (단위 : 유로) 입니다

데이터 소개

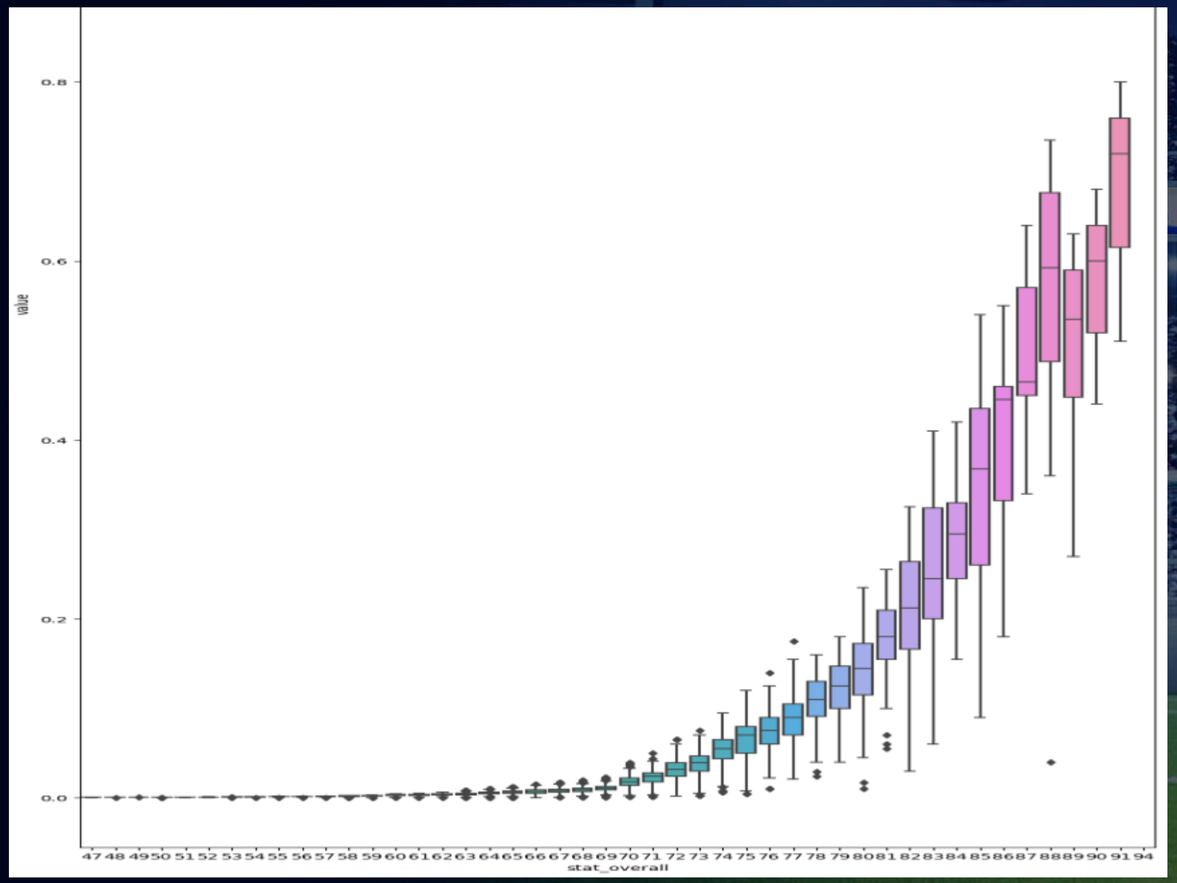
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8932 entries, 0 to 8931
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                8932 non-null   int64
 1   name              8932 non-null   object
 2   age               8932 non-null   int64
 3   continent         8932 non-null   object
 4   contract_until    8932 non-null   object
 5   position          8932 non-null   object
 6   prefer_foot       8932 non-null   object
 7   reputation        8932 non-null   float64
 8   stat_overall      8932 non-null   int64
 9   stat_potential    8932 non-null   int64
 10  stat_skill_moves  8932 non-null   float64
 11  value             8932 non-null   float64
dtypes: float64(3), int64(4), object(5)
memory usage: 837.5+ KB
```

```
#결측치 없음
data.isnull().sum()
```

```
id                  0
name                0
age                 0
continent           0
contract_until      0
position            0
prefer_foot         0
reputation          0
stat_overall        0
stat_potential      0
stat_skill_moves    0
value               0
dtype: int64
```

파생변수 1

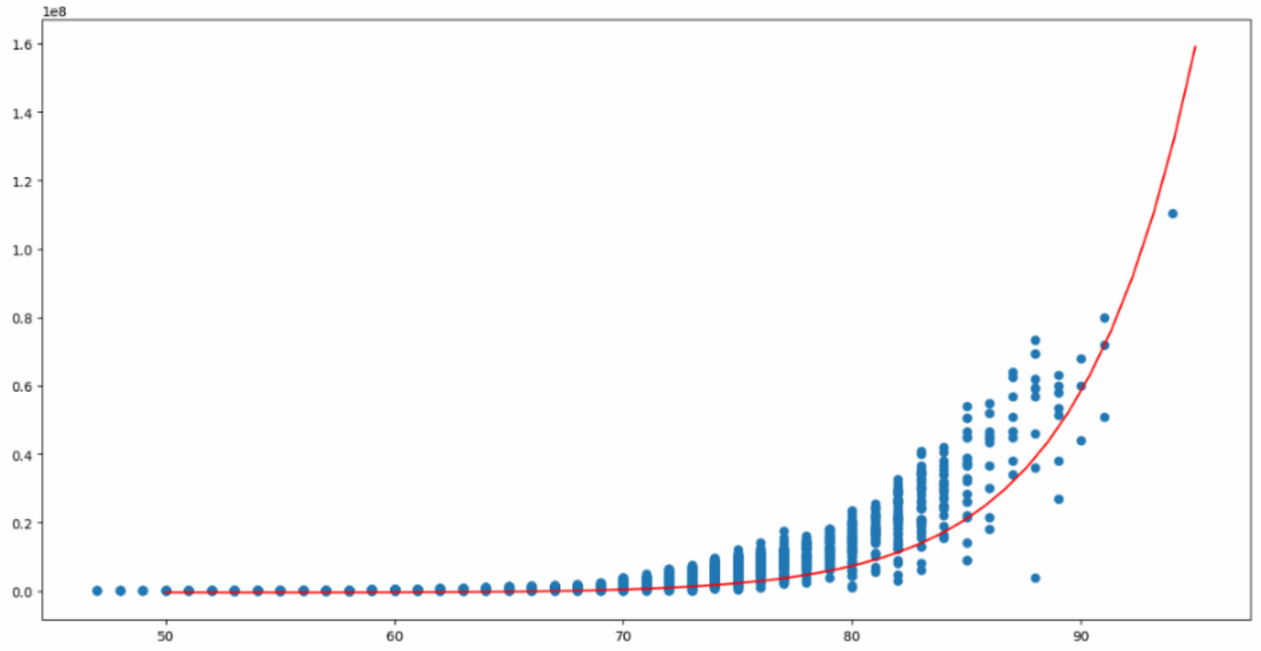


능력치 별 평균 이적료를 나타낸 그림. 예상대로, 높은 능력치일수록 이적료가 지수적으로 증가

파생변수 1

```
In [73]: plt.figure(figsize = (16, 8))
         plt.scatter(data = number_value, x = 'stat_overall', y ='value')
         plt.plot(x_line, y_line, color='r', label=r"$(1/2)^x$")
```

```
Out[73]: [<matplotlib.lines.Line2D at 0x20ae08a03d0>]
```



개인 능력치와 이적료에는
지수함수 처럼 보이는
관계를 볼 수 있다.

상관관계

파생변수 가설 1: 개인능력치와 이적료 사이는 지수관계이다.

```
number_value['relation_stat_value'] = 1.20**number_value['stat_overall'].values - number_value['stat_overall'].values*10000
```

```
number_value['relation_stat_value_po'] = 1.20**number_value['stat_potential'].values - number_value['stat_potential'].values*10000
```

지수함수로 FITTING

```
number_value.corr()['relation_stat_value']['value']
```

```
0.9208875795920758
```

```
number_value.corr()['relation_stat_value_po']['value']
```

```
0.8317417176762215
```

FITTING 한 변수와 이적료가 가지는 상관관계

파생변수 2

나이에 비해 명성이 높은 선수들은 이적료가 높을 것이다.

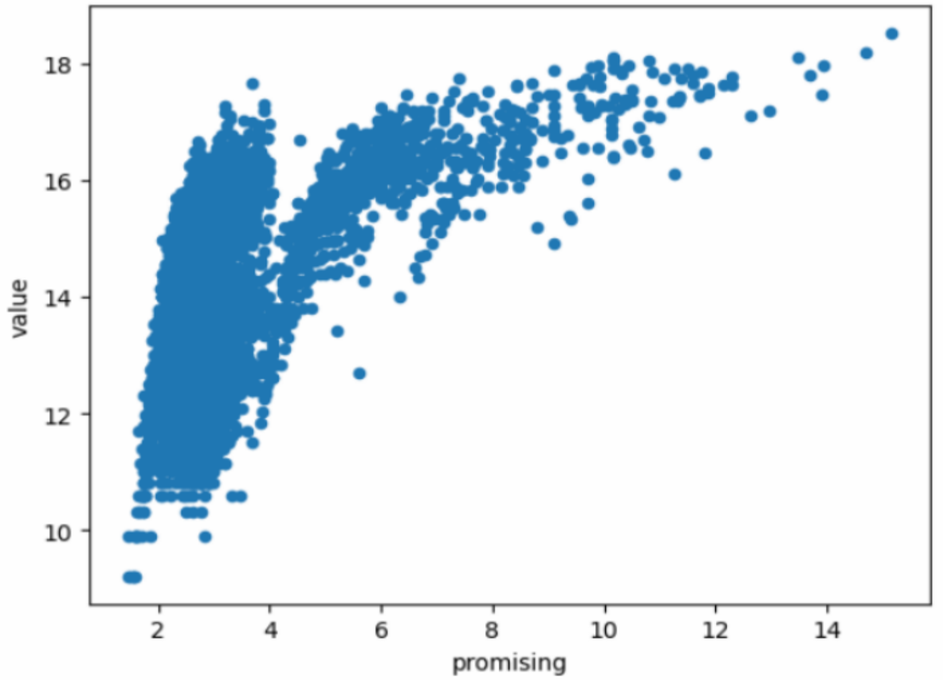
단, 값 간의 차이를 주기 위해 가중치로 개인 능력치를 곱하여 출력
-> 개인 능력치를 고려한 나이 대비 명성을 의미
명성이 축구를 잘해서 높은 사람과 다른 콘텐츠로(방송 등) 인해
높은 사람에게 차별성을 주기 위함

상관관계

```
In [778]: number_value['promising'] = number_value['stat_overall']*(number_value['reputation']/number_value['age'])

In [779]: number_value.plot(kind='scatter', x = 'promising', y = 'value')

Out[779]: <AxesSubplot:xlabel='promising', ylabel='value'>
```



promising = overall * (reputation / age)

발전 가능한 선수

현재의 능력치도 중요하지만
미래에 발전할 가능한 선수가 가치가 높을 것이라는 추측

stat=(overall + 가중치 * potential)/ 1+가중치 ¶
가중치= potential / overall - 발전가능한 크기
가중치>=1, 1이상이면 발전가능성이 있다

상관관계

```
In [780]: number_value['stat'] = ((number_value['stat_overall'].values * number_value['stat_overall'].values)+(number_value['stat_potential'].
```

```
In [781]: number_value.corr()['stat']['value']

Out[781]: 0.9442825495342195
```

['stat'] =
['stat_overall']**2+['stat_potential']**2
/(['stat_potential'] + ['stat_overall'])

전처리 과정

```
eda_data = pd.concat([cat, number_value], axis = 1)
```

```
#label encoding
eda_data['position'] = eda_data['position'].replace(['ST','MF','DF','GK'],[4,3,2,1])
eda_data['prefer_foot'] = eda_data['prefer_foot'].replace(['right','left'],[1,2])
eda_data['continent'] = eda_data['continent'].replace(['south america','europe','africa','asia','oceania'],[1,2,3,4,5])
```

In [802]:
```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
X_cat_1hot = cat_encoder.fit_transform(X_cat)
X_cat_1hot
```

Out[802]:
```
<8932x2 sparse matrix of type '<class 'numpy.float64'>'
        with 8932 stored elements in Compressed Sparse Row format>
```

In [803]:
```
X_cat_1hot.toarray()  #직접 보면 이렇게
```

Out[803]:
```
array([[0., 1.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

In [804]:
```
cat_encoder = OneHotEncoder(sparse=False)
eda_data_cat_1hot = cat_encoder.fit_transform(X_cat)
eda_data_cat_1hot
```

Out[804]:
```
array([[0., 1.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

전처리 과정

In [807]:
```
#1. 기존 변수를 제거
eda_data.drop('prefer_foot', axis = 1, inplace = True)

#2. 인코딩한 변수를 추가
encoded = pd.DataFrame(eda_data_cat_1hot, columns=['right', 'left'])
sex_jangchun = pd.concat([eda_data, encoded], 1)

#3. 확인

sex_jangchun.head()
```

Out[807]:

sition	age	reputation	stat_overall	stat_potential	stat_skill_moves	value	relation_stat_value	relation_stat_value_po	promising	stat	right	left
4	31	5.0	94	94	4.0	18.520526	2.679557e+07	2.679557e+07	15.161290	94.00000	0.0	1.0
1	27	4.0	91	93	1.0	18.092177	1.514068e+07	2.218298e+07	13.481481	92.01087	1.0	0.0
4	31	5.0	91	91	3.0	18.197537	1.514068e+07	1.514068e+07	14.677419	91.00000	1.0	0.0
2	32	4.0	91	91	3.0	17.747336	1.514068e+07	1.514068e+07	11.375000	91.00000	1.0	0.0
1	25	3.0	90	93	1.0	18.035018	1.247557e+07	2.218298e+07	10.800000	91.52459	1.0	0.0

전처리 과정

In [818]:
```
data_test = pd.read_csv('FIFA_test.csv')
```

In [819]:
```
data_test
```

0	1	Cristiano Ronaldo	33	europe	2022	ST	right	5.0	94	94	5.0
1	2	Neymar Jr	26	south america	2022	ST	right	5.0	92	93	5.0
2	4	K. De Bruyne	27	europe	2023	MF	right	4.0	91	92	4.0
3	5	E. Hazard	27	europe	2020	ST	right	4.0	91	91	4.0
4	6	L. Modrić	32	europe	2020	MF	right	4.0	91	91	4.0
...	...	...	...	...	...	...	...	...	...	...	...
3823	16924	R. Takae	20	asia	2021	MF	right	1.0	48	63	2.0
3824	16929	L. Wahlstedt	18	europe	2018	GK	right	1.0	48	65	1.0
3825	16932	Y. Góez	18	south america	2021	MF	right	1.0	48	65	2.0
3826	16937	A. Kaltner	18	europe	2020	ST	right	1.0	47	61	2.0
3827	16943	K. Fujikawa	19	asia	2021	MF	right	1.0	47	61	2.0

3828 rows × 11 columns

In [820]:
```
data_test['position'] = data_test['position'].replace(['ST','MF','DF','GK'],[4,3,2,1])
data_test['prefer_foot'] = data_test['prefer_foot'].replace(['right','left'],[1,2])
data_test['continent'] =data_test['continent'].replace(['south america','europe','africa','asia','oceania'],[1,2,3,4,5])
```

In [821]:
```
data_test.drop(['id','name','contract_until'],axis=1 , inplace=True)
data_test
```

Out [821]:

LGBM 을 선택한 이유

데이터 수가 적기 때문에 과적합이 생길 확률이 크다고 생각.
이 때문에 과적합이 적게 발생하는 모델인 LGBM 을 선택

LGBM 은 리프 분할 방식을 이용하여 Loss가
가장 큰 node를 선택해 loss를 줄여가는 방식이다.

과적합을 줄이는 LGBM 요소 : max_depth,
'n_estimators','min_child_samples', 'subsample'.

모델 선택과 이유

max_depth

max_depth tree의 최대 깊이다. 값을 줄이면 overfitting를 해결할 수 있다.
기본값은 -1로 깊이의 제한이 없다. 최대 2 N 개의 leaf node가 생길 수 있다.

n_estimators

크게 하면 정확도가 높아진다.
하지만 너무 크게하면 시간이 오래걸리는 단점이 생긴다.

min_child_samples

leaf node에 포함되어야 할 최소한의 데이터 개수를 의미한다.
따라서 분할을 조정하여 overfitting을 제어한다.

subsample.

boosting 단계마다 데이터를 랜덤으로 선택하는 비율이다. 즉 매 번 subsample 비율만큼 데이터를 랜덤으로 선택해서 학습을 진행한다. 값이 작으면 속도를 향상할 수 있고, overfitting을 방지할 수 있다.

모델 선택과 이유

LGBM 단점 : 데이터의 갯수가 10000개 이하면 과적합이 발생할 확률이 높다.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from sklearn.linear_model import LinearRegression, ElasticNet, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import  LabelEncoder
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
# import xgboost as xgb
import lightgbm as lgb
import re
import optuna
# from optuna.integration import XGBoostPruningCallback
# sns.set_theme(style="darkgrid")
```

```
import lightgbm as lgb
import optuna
```

```
#타겟분리
train_X, train_y = train.drop('value', axis=1), train['value']
print(train_X.shape, train_y.shape)
```

LGBM 과 optuma 를 불러온 후
MINMAX 스케일링 진행

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(train)
scaler.transform(train)
```

모델 선택과 이유

```
cut = int(len(train)*0.8)
h_train = train[:cut]
h_valid = train[cut:]

h_train_X = h_train.drop('value', axis=1)
h_train_y = h_train['value']
h_valid_X = h_valid.drop('value', axis=1)
h_valid_y = h_valid['value']
print(h_train_X.shape, h_train_y.shape, h_valid_X.shape, h_valid_y.shape)
```

train valid 으로 split 실시

```
from optuna.samplers import TPESampler
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning) # FutureWarning 제거
warnings.filterwarnings(action='ignore')

sampler = TPESampler(seed=10)
# LGBMRegressor(max_depth=9, min_child_weight=5, n_estimators=500)
def objective(trial):
    dtrain = lgb.Dataset(h_train_X, label=h_train_y)
    dtest = lgb.Dataset(h_valid_X, label=h_valid_y)

    param = {
        'objective': 'regression', # 회귀
        'verbose': -1,
        'metric': 'rmse',
        'max_depth': trial.suggest_int('max_depth',3, 15),
        'learning_rate': trial.suggest_loguniform("learning_rate", 1e-3, 1e-1),
        'n_estimators': trial.suggest_int('n_estimators', 350, 600),
        'min_child_samples': trial.suggest_int('min_child_samples', 10, 100),
        'subsample': trial.suggest_loguniform('subsample', 0.4, 1),
    }

    model = lgb.LGBMRegressor(**param)
    lgb_model = model.fit(h_train_X, h_train_y, eval_set=[(h_valid_X, h_valid_y)], verbose=0, early_stopping_rounds=25)
    rmse = RMSE(h_valid_y, lgb_model.predict(h_valid_X))
    return rmse

study_lgb = optuna.create_study(direction='minimize', sampler=sampler)
study_lgb.optimize(objective, n_trials=300)
```

최상의 파라미터를
찾아가며
훈련과 학습을 진행


예기치 못한 문제 발생
데이터 수가 부족하여 Overfitting 발생
데이터 수를 늘려보자

외부 데이터 가져오기

```
kl = pd.read_csv("kl.csv", encoding = 'latin')
kl
```

4	4	192985	K. De Bruyne	27.0	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91.0	92
...	...	...	...	...	...	...	...	...	...
18202	18202	238813	J. Lundstram	19.0	https://cdn.sofifa.org/players/4/19/238813.png	England	https://cdn.sofifa.org/flags/14.png	47.0	65
18203	18203	243165	N. Christoffersson	19.0	https://cdn.sofifa.org/players/4/19/243165.png	Sweden	https://cdn.sofifa.org/flags/46.png	47.0	63
18204	18204	241638	B. Worman	16.0	https://cdn.sofifa.org/players/4/19/241638.png	England	https://cdn.sofifa.org/flags/14.png	47.0	67
18205	18205	246268	D. Walker-Rice	17.0	https://cdn.sofifa.org/players/4/19/246268.png	England	https://cdn.sofifa.org/flags/14.png	47.0	66
18206	18206	246269	G. Nugent	16.0	https://cdn.sofifa.org/players/4/19/246269.png	England	https://cdn.sofifa.org/flags/14.png	46.0	66

18207 rows × 89 columns

외부 데이터 전처리

```
In [337]:
khhh.rename(columns = {'Name' : 'name'}, inplace = True)
khhh.rename(columns = {'Age' : 'age'}, inplace = True)
khhh.rename(columns = {'Nationality' : 'continent'}, inplace = True)
khhh.rename(columns = {'Contract Valid Until' : 'contract_until'}, inplace = True)
khhh.rename(columns = {'Position' : 'position'}, inplace = True)
khhh.rename(columns = {'Preferred Foot' : 'prefer_foot'}, inplace = True)
khhh.rename(columns = {'International Reputation' : 'reputation'}, inplace = True)
khhh.rename(columns = {'Overall' : 'stat_overall'}, inplace = True)
khhh.rename(columns = {'Potential' : 'stat_potential'}, inplace = True)
khhh.rename(columns = {'Skill Moves' : 'stat_skill_moves'}, inplace = True)
khhh.rename(columns = {'Value' : 'value'}, inplace = True)
```

항목에 맞춰 column 을 정렬하고 column 명이 맞도록 네이밍 해주는 작업

```
In [333]:
khhh['Value'].iloc[0][1:-1]

Out[333]:
'110.5'

In [334]:
khhh['Value'].iloc[452]

Out[334]:
'₩x80 0'

In [335]:
drop_list = []
for i in range(len(khhh)):

    if khhh['Value'].iloc[i][-1] == 'M':
        khhh['Value'].iloc[i] = float(khhh['Value'].iloc[i][1:-1]) * 10**6

    elif khhh['Value'].iloc[i][-1] == 'K':
        khhh['Value'].iloc[i] = float(khhh['Value'].iloc[i][1:-1]) * 10**3

    elif khhh['Value'].iloc[i][-1] == '0':
        drop_list.append(i)
```

value 가
숫자 M , K 로 표기 되어있었기에
IF문을 사용하여 숫자로 바꿔줌

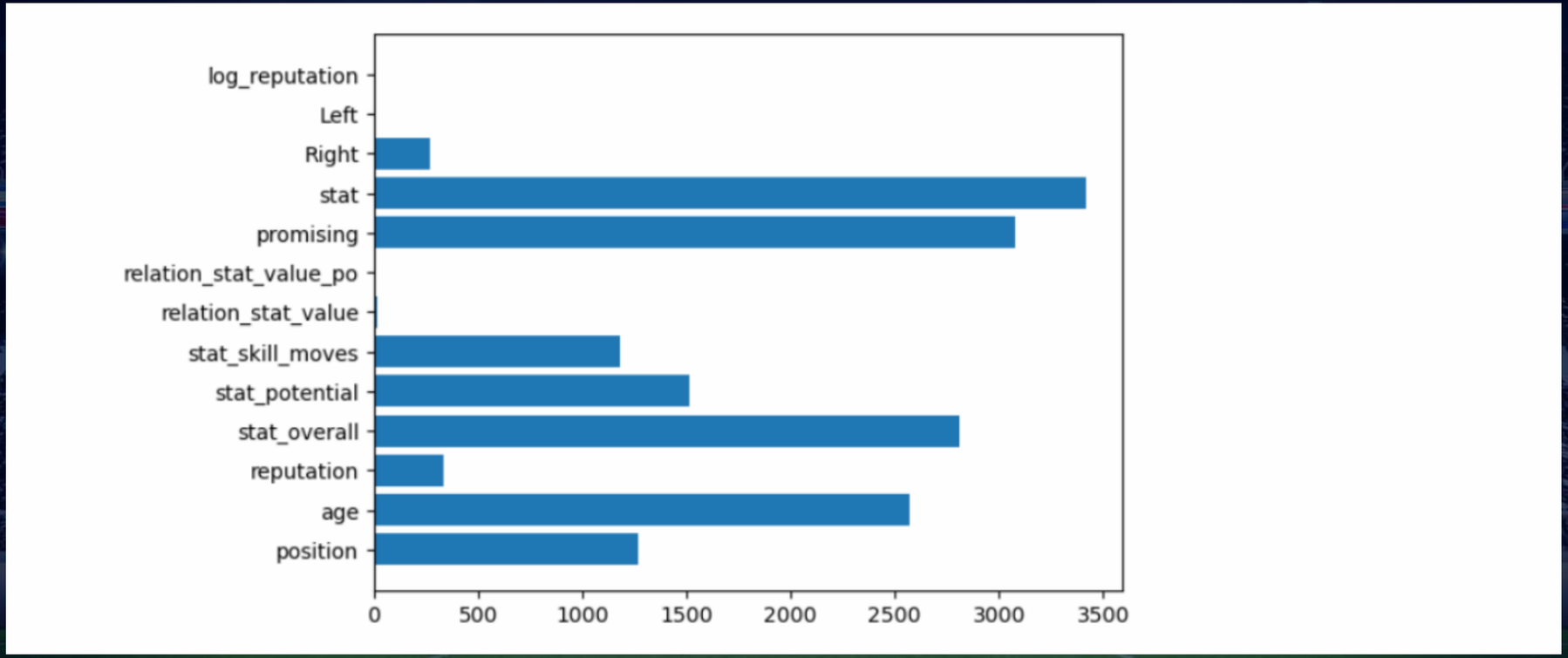
테스트 결과
최상의 파라미터

In [553]:
```
trial = study_lgb.best_trial
trial_params = trial.params
print('Best Trial: score {},₩nparams {}'.format(trial.value, trial_params))
```

```
Best Trial: score 539874.6836704725,
params {'max_depth': 14, 'learning_rate': 0.0706965864735882, 'n_estimators': 596, 'min_child_samples': 21, 'subsample': 0.6818377266085626}
```

In [554]:
```
final_lgb_model = lgb.LGBMRegressor(**trial_params)
final_lgb_model.fit(train_X, train_y)
train_lgb_pred = final_lgb_model.predict(train_X)
```

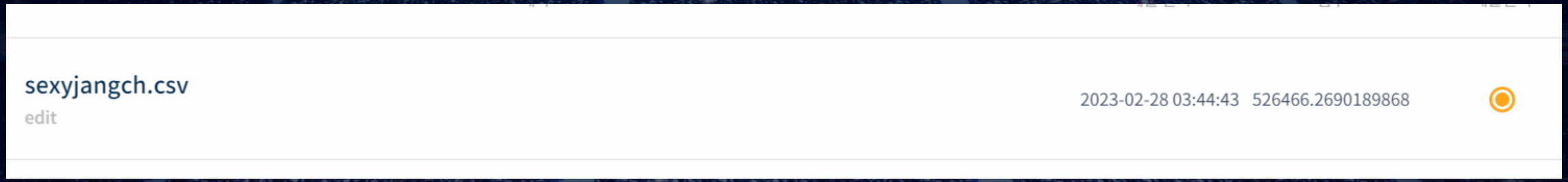
TEST 결과



모델 학습때 가장 유용하게 이용한 변수

우리가 만든 파생변수인 유망주 변수와 발전가능성 변수가 가장 유용했다.

TEST 결과



외부데이터를 가져와 데이터의 갯수를 늘려
LGBM의 성능을 늘려줌과 동시에
OVERFITTING을 막아내어
좋은 성능을 얻어냈습니당!!

QnA

감사합니다