2022.2.24

해외 축구선수 이적료 예측 프로젝트

22-1 머신러닝 스터디 분반

15기 고태영 김제성 반민정 장수혁

목차

01

분석 목적

FIFA에서 제공하는 데이터(2018년)를 바탕으로 선수들의 이적 시장 가격 예측하기

>> Regression을 사용해 예측 정확도가 가장 높은 모델 만들기

사용데이터: FIFA_train.csv / FIFA_test.csv(value값 제외)

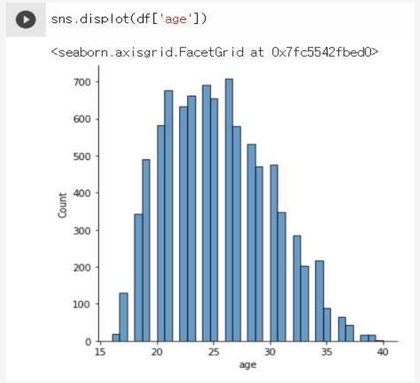
평가 산식: RMSE

02

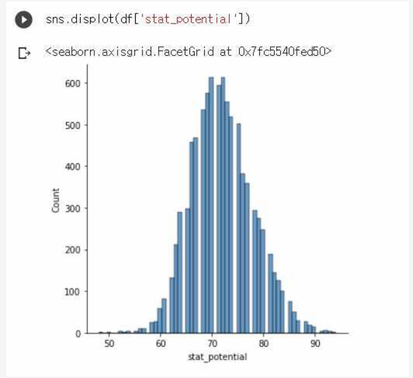
EDA를 통해 변수 인사이트 얻기

1) 데이터 분포 파악

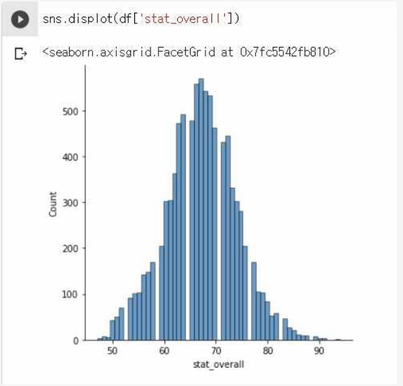
새로운 변수 생성



age



stat_overall

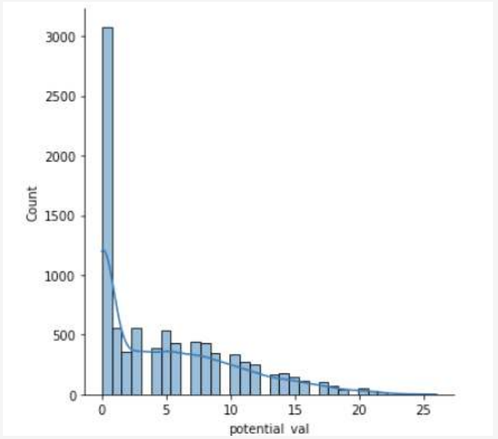


stat_potential

```
#잠재가치 변수 생성
df['potential_val']=df['stat_potential']-df['stat_overall']
sns.displot(df['potential_val'], kde=True)
```

* 잠재가치를 나타내는 새로운 변수 생성

성장할 수 있는 정도(stat_potential)-현재 능력치 (stat_overall)
= potential_val



potential_val

1) 데이터 분포 파악

contract_until 변수 재설정

```
# value가 통일되지 않음
df['contract_until'].unique()
```

```
array(['2021', '2020', '2019', '2023', '2022', '2024', 'Jun 30, 2019',
       '2026', 'Dec 31, 2018', '2018', '2025', 'Jun 30, 2020',
       'May 31, 2020', 'May 31, 2019', 'Jan 31, 2019', 'Jan 1, 2019',
       'Jan 12, 2019'], dtype=object)
```

```
[8] # 계약연도만 이용하고 데이터타입을 숫자로 반환
    df['contract_until'] = df['contract_until'].str[-4:]
    df['contract_until'] = pd.to_numeric(df['contract_until'])
    df['contract_until'].unique()
```

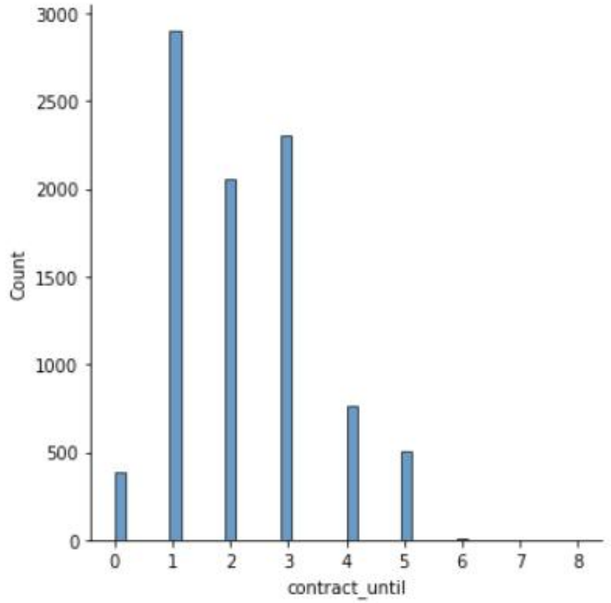
```
array([2021, 2020, 2019, 2023, 2022, 2024, 2026, 2018, 2025])
```

계약 만료 기간이 다가올수록
FA, 재계약, 타구단 이적 등의 이유로 이적료가 낮아짐

➜ 계약만료기간에서 데이터 작성 시점 기준(2018년) 을 뺀 남은 연도로 변수 재설정

```
[9] df['date']=2018
    df['contract_until']-=df['date']
    df.drop('date', axis=1, inplace=True)
    sns.displot(df['contract_until'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7fee84e7c6d0>
```

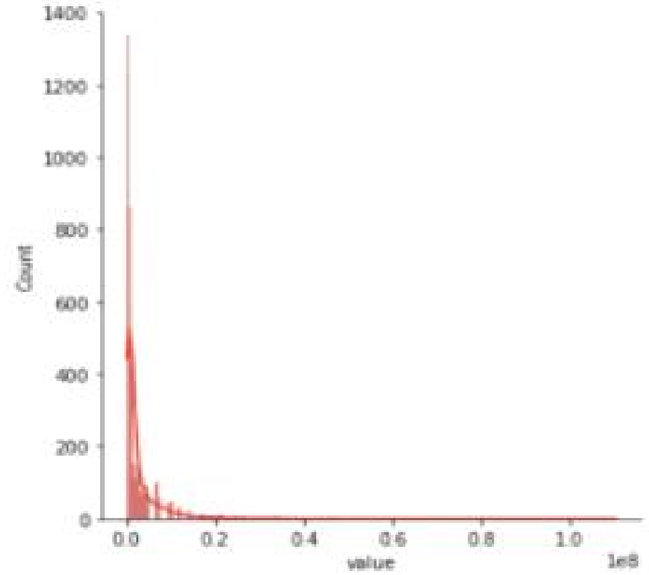


1) 데이터 분포 파악

value 변수 특징

```
sns.displot(df['value'],kde=True,color='red')
print('value 변수의 왜도: ',df['value'].skew())
```

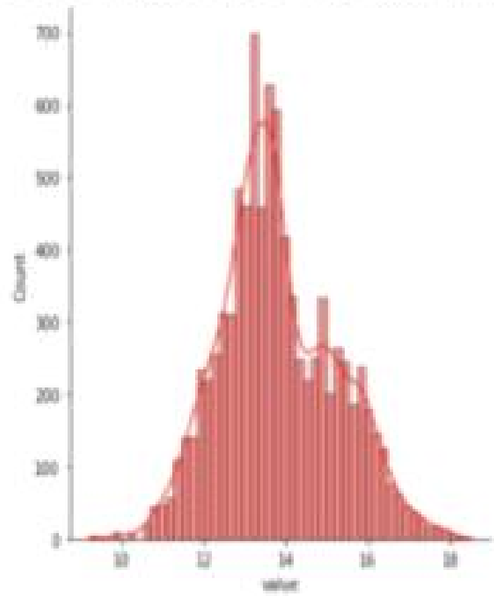
```
value 변수의 왜도:  5.885798215027232
```



Right-skewed로 왜도가 심하다

```
sns.displot(np.log(df['value']),kde=True,color='red')
print('로그변환한 value 변수의 왜도: ',np.log(df['value']).skew())
```

```
로그변환한 value 변수의 왜도:  0.3087906004737585
```



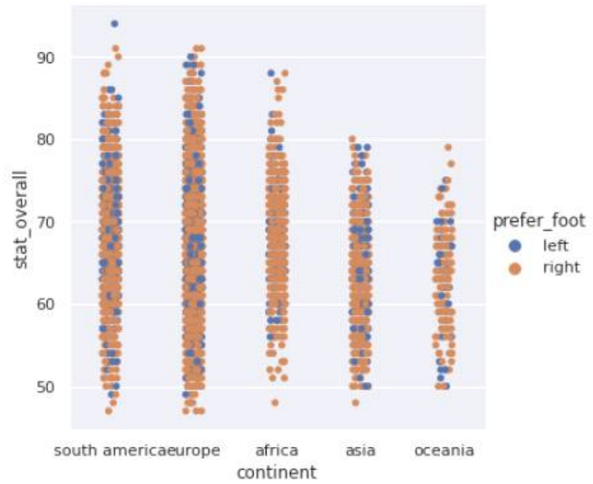
➜ log변환을 이용해 분포의 비대칭성 완화

1) 데이터 분포 파악

데이터 분포 특성

```
sns.catplot(data = df, x= "continent", y = "stat_overall", hue = "prefer_foot")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fee7add2690>
```

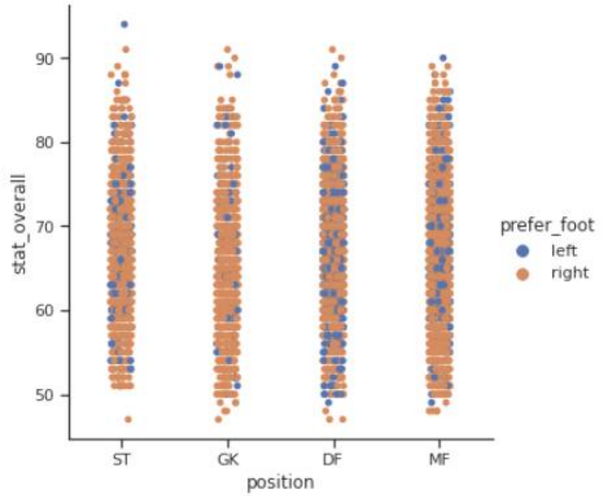


- 주로 유럽 대륙에 능력치가 우수한 선수들이 포진됨

- 남미 출신 공격수의 능력치가 상대적으로 우수함

```
sns.set_theme(style="ticks")
sns.catplot(data = df, x= "position", y = "stat_overall", hue = "prefer_foot")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fee7ab2cf90>
```

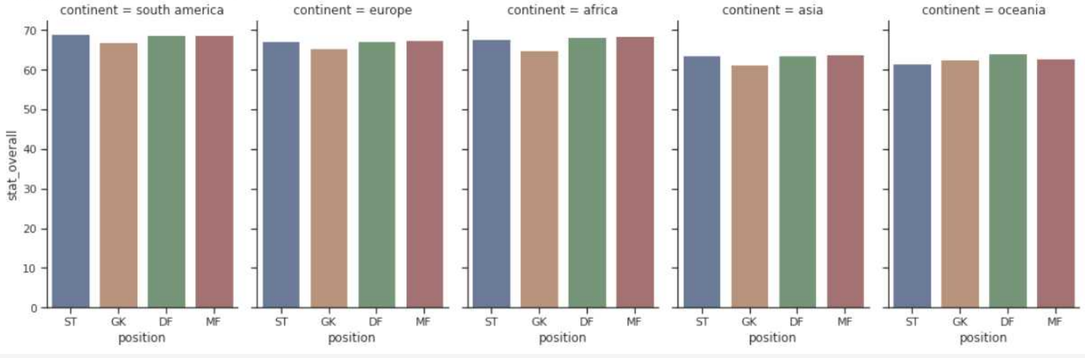


- 오른발 선수의 비중이 왼발 선수보다 크다.

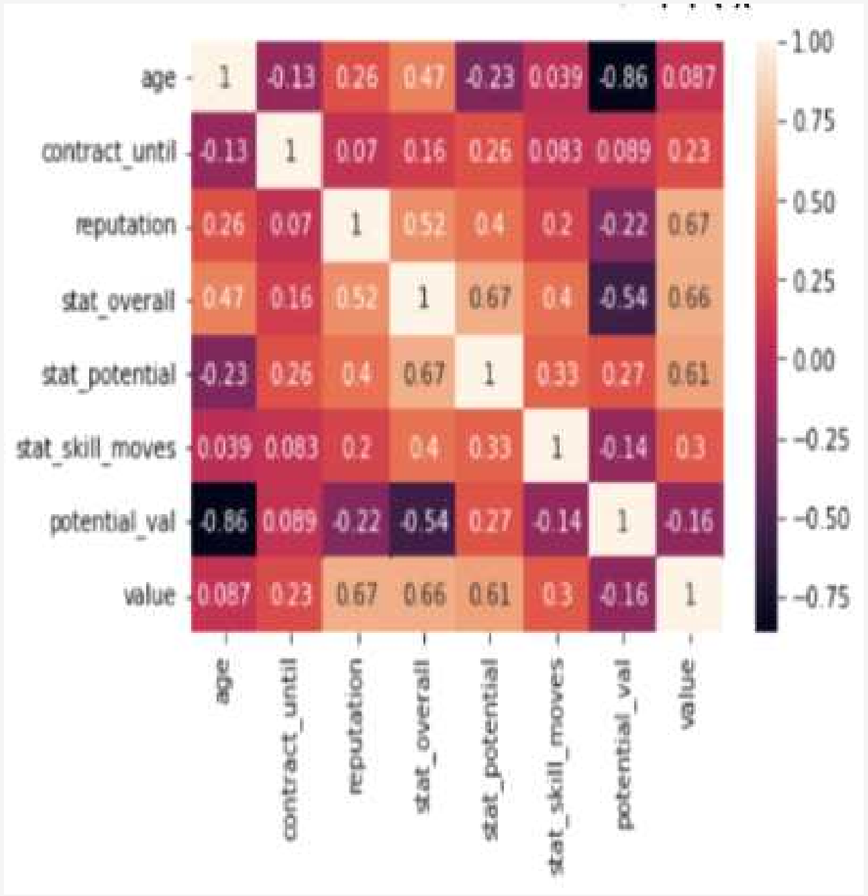
  그러나 특이하게 수비수에서는 왼발 선수의 비중이 크다.

```
sns.catplot(x="position", y="stat_overall", col="continent",
            data=df, saturation=.5,
            kind="bar", ci=None, aspect=.6)
```

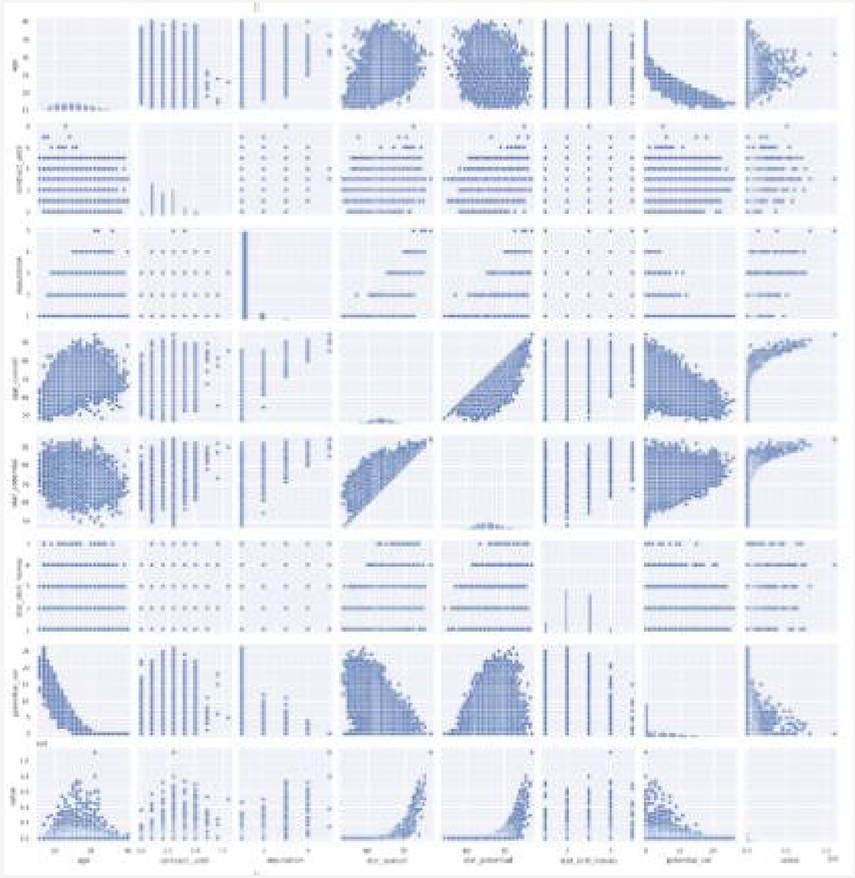
```
<seaborn.axisgrid.FacetGrid at 0x7fee784451d0>
```



2) 변수 간 상관관계 파악



correlation heatmap



pairplot

Reputation, stat_overall, stat_potential 세가지 변수가 타켓변수(value)와 높은 상관성을 지니고 있다.

03

전처리

1) 변수선택

2) 이상치 처리

3) 로그 스케일링

4) 스케일링

5) one-hot encoding

1) 변수 선택

불필요한 변수 id, name 제거

```
[9] fifa_train.drop(['id','name'], axis=1, inplace=True)
    fifa_train
```

	age	continent	contract_until	position	prefer_foot	reputation	stat_overall	stat_potential	stat_skill_moves	value
0	31	south america	2021	ST	left	5.0	94	94	4.0	110500000.0
1	27	europe	2020	GK	right	4.0	91	93	1.0	72000000.0
2	31	south america	2021	ST	right	5.0	91	91	3.0	80000000.0
3	32	europe	2020	DF	right	4.0	91	91	3.0	51000000.0
4	25	europe	2021	GK	right	3.0	90	93	1.0	68000000.0
...	...	...	...	...	...	...	...	...	...	...
8927	18	africa	2019	MF	right	1.0	48	63	3.0	60000.0
8928	19	europe	2020	DF	right	1.0	47	59	2.0	40000.0
8929	18	south america	2021	DF	right	1.0	47	64	2.0	50000.0
8930	18	europe	2021	GK	right	1.0	47	65	1.0	50000.0
8931	19	europe	2020	ST	right	1.0	47	63	2.0	60000.0

8932 rows × 10 columns

1) 변수 선택

변수 포함 여부에 따른 RMSE 비교

models

	Model	RMSE	RMSE (Cross-Validation)
0	Basic	0.079810	0.082018
1	Age 삭제	0.249475	0.234089
2	stat_skill_moves 삭제	0.086640	0.084587
3	potential_val 삭제	0.084362	0.085659
4	Age, stat_skill_moves 삭제	0.252774	0.236081
5	Age, potential_val 삭제	0.262337	0.242927
6	potential_val, stat_skill_moves 삭제	0.090492	0.088370
7	셋 다 삭제	0.266864	0.245454

age, stat_skill_moves, potential_val
세가지 변수에 대한 8가지 경우의 수에 대해 feature selection을 진행

➜ 세가지 변수 모두를 포함하는 것이 성능이 가장 좋음

2) 이상치 처리

변수 별 이상치 처리 유무에 따른 RMSE 비교

	Case	Model	cv_rmse	rmse
0	모든 변수(x,y) 이상치 제거	RandomForest	0.094509	0.080157
1	x변수만 이상치 제거	RandomForest	0.087252	0.074359
2	y변수만 이상치 제거	RandomForest	0.105189	0.091318
3	정규분포인 변수만 이상치 제거	RandomForest	0.098208	0.084788
4	정규분포 아닌 변수만 이상치 제거	RandomForest	0.091986	0.079872
5	정규분포 아닌 x변수만 이상치 제거	RandomForest	0.089199	0.079781
6	reputation제외한 x변수들 이상치 제거	RandomForest	0.089666	0.083208
7	reputation,stat_skill_moves 제외한 x변수들 이상치 제거	RandomForest	0.092364	0.096148
8	stat_skill_moves 제외한 x변수들 이상치 제거	RandomForest	0.088896	0.081226
9	이상치 제거 안함	RandomForest	0.096822	0.101430

이상치를 제거할 경우 1과 5사이 자연수로 이루어진
reputation과 stat_skill_moves의 데이터 값이 특정 수로만 남아서
제외하고 이상치 비교를 해봄

➜X변수만 이상치 제거하기로 결정!

3) 로그 스케일링

- <Log-Scaling> : 수치형 변수의 비선형변환
- 큰수의 범위를 압축하고, 작은 수의 범위를 확장하여 데이터가 정규분포형태에 가까워지도록 함

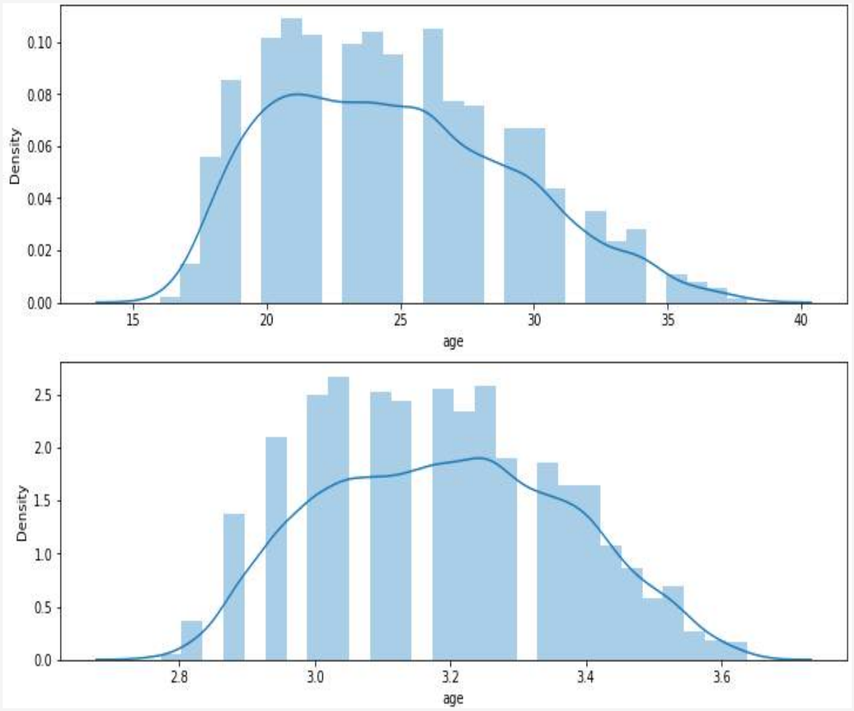
  => 첨도와 왜도를 줄여줌

- X numeric feature와 y target에 여러차례 적용하고, 성능 비교해보면서 최적의 상태 찾기

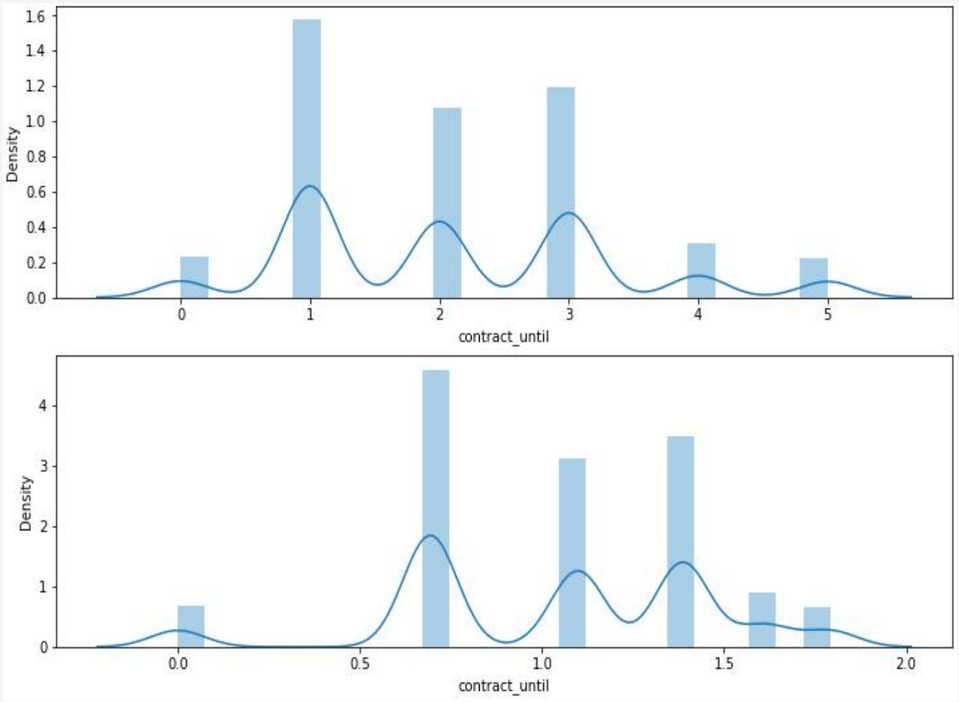
  => Randomforest model 이용

3) 로그 스케일링

X numeric feature(6개), Y target 시각화 비교
log 변환 전: 위, log 변환 후: 아래



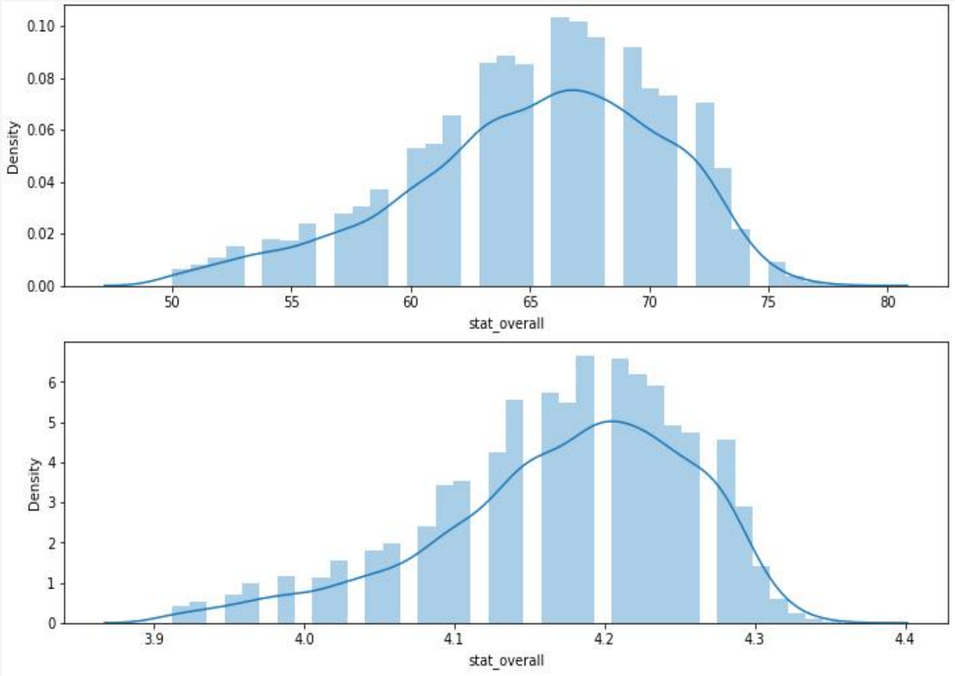
〈age〉



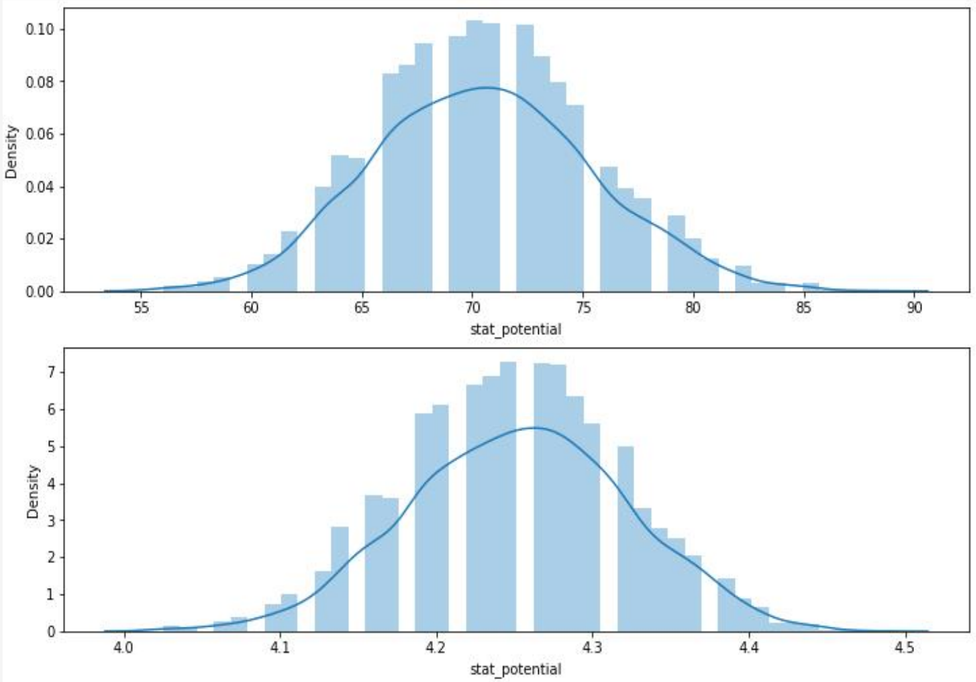
〈contract_until〉

3) 로그 스케일링

X numeric feature(6개), Y target 시각화 비교
log 변환 전: 위, log 변환 후: 아래



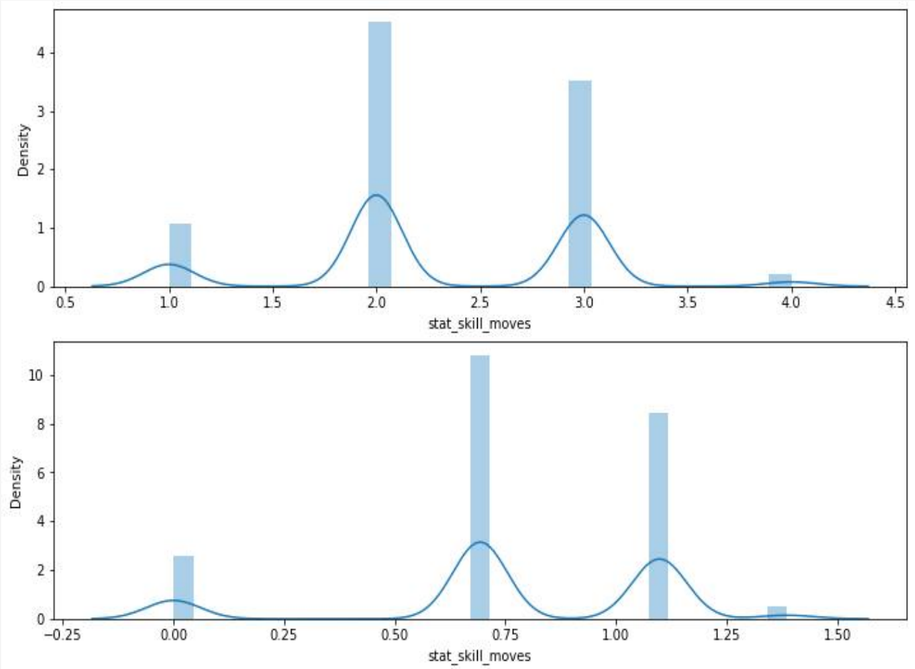
〈stat_overall〉



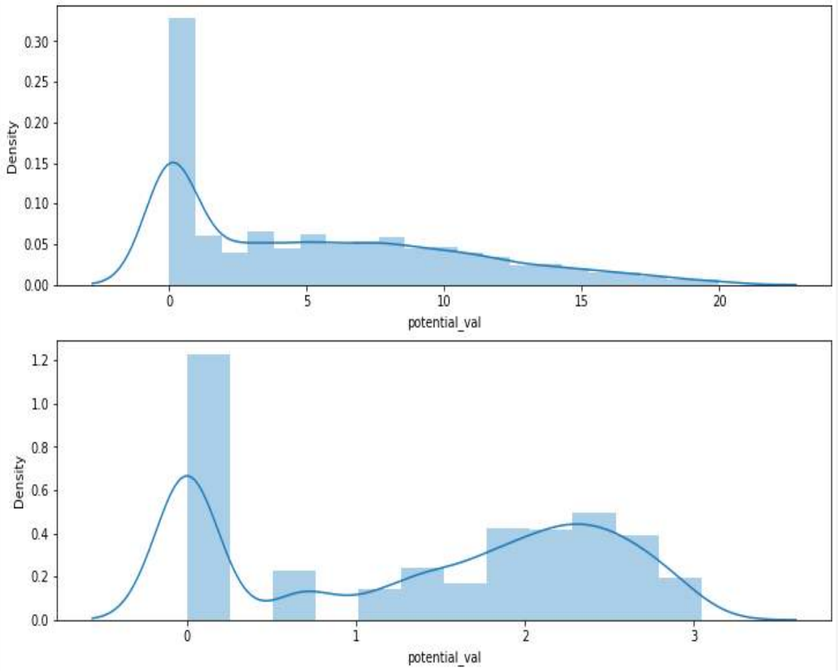
〈stat_potential〉

3) 로그 스케일링

X numeric feature(6개), Y target 시각화 비교
log 변환 전: 위, log 변환 후: 아래



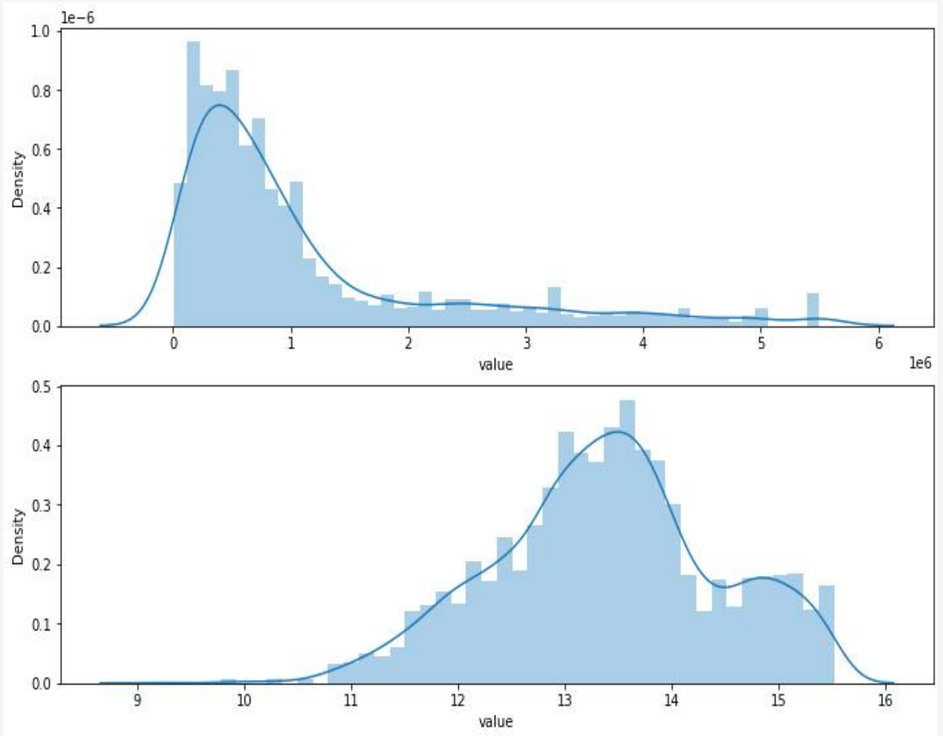
〈stat_skill_moves〉



〈potential_val〉

3) 로그 스케일링

X numeric feature(6개), Y target 시각화 비교
log 변환 전: 위, log 변환 후: 아래



〈value〉-y target

3) 로그 스케일링

다양한 케이스에 대해 RandomForest모델 적용 후 RMSE 비교

	Case	Model	RMSE	RMSE(Cross-Validation)
1	y log-scaling	randomforest	0.075434	0.087727
5	y+ 'potential_val' log-scaling	randomforest	0.075463	0.087735
4	y+ 'contract_until' log-scaling	randomforest	0.075472	0.087867
6	y+ 'contract_until','potential_val' log-scaling	randomforest	0.075501	0.087873
3	both x+y log-scaling	randomforest	0.074971	0.088863
2	x log-scaling	randomforest	322658.825155	284819.607074
0	No log-scaling	randomforest	332470.469786	286080.772075

Cross-validation(cv=5)을 이용한 RMSE기법으로 최저값인 케이스 선정

=> X_train,Y_train을 5개의 subset으로 나눠 5번 평가진행한 평균을 점수화

➜ RMSE(CV)값이 0.087727로 가장 작으므로 y target만 log-scaling하기로 결정!

4) 스케일링

여러 스케일링 방법에 따른 RMSE 비교

```
from sklearn.ensemble import RandomForestRegressor
random_forest = RandomForestRegressor(random_state=32)
#None
cv_rmse = -cross_val_score(random_forest, X_train, Y_train, scoring="neg_root_mean_squared_error", cv=5).mean()
models = pd.DataFrame(columns=["Case","Model","CV_RMSE"])
new_row = {"Case":"no scaling","Model": "randomforest", "CV_RMSE": cv_rmse}
models = models.append(new_row,ignore_index=True)
#Minmax
cv_rmse = -cross_val_score(random_forest, X_train_1, Y_train, scoring="neg_root_mean_squared_error", cv=5).mean()
new_row = {"Case":"minmax scaling","Model": "randomforest", "CV_RMSE":cv_rmse }
models = models.append(new_row,ignore_index=True)
#Standard
cv_rmse = -cross_val_score(random_forest, X_train_2, Y_train, scoring="neg_root_mean_squared_error", cv=5).mean()
new_row = {"Case":"standard scaling","Model": "randomforest", "CV_RMSE": cv_rmse}
models = models.append(new_row,ignore_index=True)
#Robust
cv_rmse = -cross_val_score(random_forest, X_train_3, Y_train, scoring="neg_root_mean_squared_error", cv=5).mean()
new_row = {"Case":"robust scaling","Model": "randomforest", "CV_RMSE": cv_rmse}
models = models.append(new_row,ignore_index=True)
```

	Case	Model	CV_RMSE
0	no scaling	randomforest	0.096568
1	minmax scaling	randomforest	0.103129
2	standard scaling	randomforest	0.103079
3	robust scaling	randomforest	0.103036

1) None
2) Minmax
3) Standard
4) Robust

➜성능이 가장 좋았던None Scaling으로 사용 결정!

5) one-hot encoding

범주형 변수를 one-hot encoding 해줌

```
X = pd.get_dummies(X)
X
```

	continent_africa	continent_asia	continent_europe	continent_oceania	continent_south america
)	0	0	0	0	1
2	0	0	1	0	0
)	0	0	0	0	1
)	0	0	1	0	0
3	0	0	1	0	0

04

모델훈련 및 비교

모델훈련 및 비교

1)Linear regression

2)Ridge regression

3)Lasso regression

4)Elasaticnet regression

5)SVR

6)Randomforest

7)XGBbootregressor

8)Lightgbmregressor

9)Extratreeregressor

10)AdaBoostregressor

sklearn.model_selection.GridSearchCV

class sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, *, *scoring=None*, *n_jobs=None*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score=nan*, *return_train_score=False*) [source]

Exhaustive search over specified parameter values for an estimator.

Important members are fit, predict.

GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

10가지 regression 모델을 GridSearchCV()를 이용해 Hyperparameter tuning 후 성능 비교

모델훈련 및 비교

```
LR=LinearRegression()
parameters ={'fit_intercept': [True, False]
             }
grid_search = GridSearchCV(LR,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Linear regression] RMSE:0.232113

```
LR=LinearRegression()
parameters ={'fit_intercept': [True, False]
             }
grid_search = GridSearchCV(LR,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Ridge regression] RMSE:0.232152

```
LS=Lasso(random_state=42)
parameters ={'alpha': [10,1,0,1,0.01,0.001]
            }
grid_search = GridSearchCV(LS ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Lasso regression] RMSE:0.232152

```
ESN=ElasticNet(random_state=42)
parameters ={'alpha': [10,1,0,1,0.01,0.001],
             'l1_ratio': [0.5,0.7]
            }
grid_search = GridSearchCV(ESN ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Elasaticnet regression] RMSE:0.232152

```
SVR=SVR()
parameters ={'kernel': ['rbf'] , 'C':[10,100], 'gamma': [0.001,0.01,0.1]}
grid_search = GridSearchCV(SVR ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[SVR] RMSE:0.088785

```
rf=RandomForestRegressor(random_state=42)
parameters ={'max_depth': [10, 50, 100],
      'max_features': [2,4,6,10,20],
      'n_estimators': [3,10,30,50]}
grid_search = GridSearchCV(rf,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Randomforest] RMSE:0.086585

```
XgbBoost=XGBRegressor(random_state=42)
parameters ={'nthread':[4],
             'objective':['reg:linear'],
             'learning_rate': [.03, 0.05, .07],
             'max_depth': [5, 6, 7],
             'min_child_weight': [4],
             'silent': [1],
             'subsample': [0.7],
             'colsample_bytree': [0.7],
             'n_estimators': [500]}
grid_search = GridSearchCV(XgbBoost ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[XGBbootregressor] RMSE:0.060479

모델훈련 및 비교

```
Extree=ExtraTreesRegressor(random_state=42)
parameters={'max_depth': [10, 30, 100, 120],
    'max_features': [2,4,7,11,],
    'n_estimators': [3,15,30,55,80,100]}
grid_search = GridSearchCV(Extree ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Lightgbmregressor] RMSE:0.066351

```
Lgbm=LGBMRegressor(random_state=42)
parameters = {
    'num_leaves': [7, 14, 21, 28, 31, 50],
    'learning_rate': [0.1, 0.03, 0.003],
    'max_depth': [-1, 3, 5],
    'n_estimators': [50, 100, 200, 500],
}
grid_search = GridSearchCV(Lgbm ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[Extratreeregressor] RMSE:0.084724

```
AdaBoost=AdaBoostRegressor(random_state=42)
parameters ={'n_estimators': [10,50,100,500] , 'learning_rate':[1,0.1,0.01,0.001,0.0001]}
grid_search = GridSearchCV(AdaBoost ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
```

[AdaBoostregressor] RMSE:0.277793

최종결과

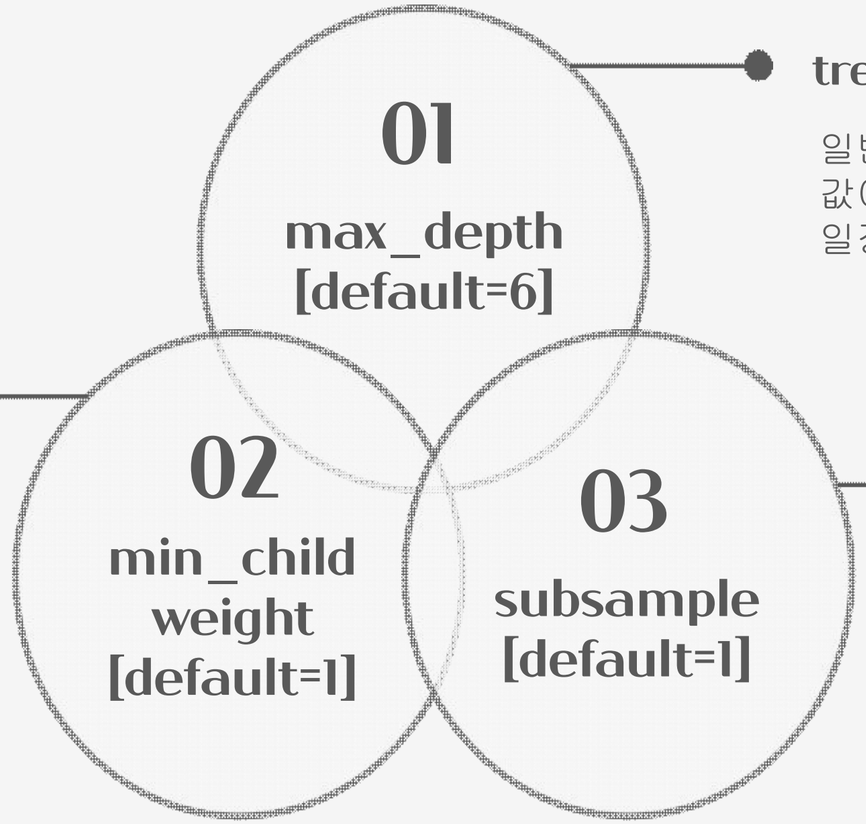
	Model	기본 basemodel의 cv_rmse	최적 모델의 cv_RMSE
7	Xgbboostregressor	0.082371	0.060479
8	Lightgbm regressor	0.079042	0.066351
9	Extratree regressor	0.080705	0.084724
0	randomforest	0.087142	0.086585
5	SVR	0.222216	0.088785
1	linear regression	0.232113	0.232113
2	Lidge regression	0.232152	0.232152
3	Lasso regression	0.389331	0.232152
4	Elasticnet regression	0.351477	0.232152
6	Adaboostregressor	0.297237	0.277793

성능이 가장 좋았던
Xgbboost regressor를 선정!

05

모델 세부 튜닝

1) XGBoost의 주요 parameters



tree의 최대 깊이

일반적으로 feature 수에 따라
값이 증가하면 모델의 성능이 증가하지만
일정 값 이후부터는 overfitting이 발생하여 성능 감소

child를 만드는데 필요한 최소 관측치 수

이 값보다 sample 수가 작으면
leaf node가 된다(자식 X)

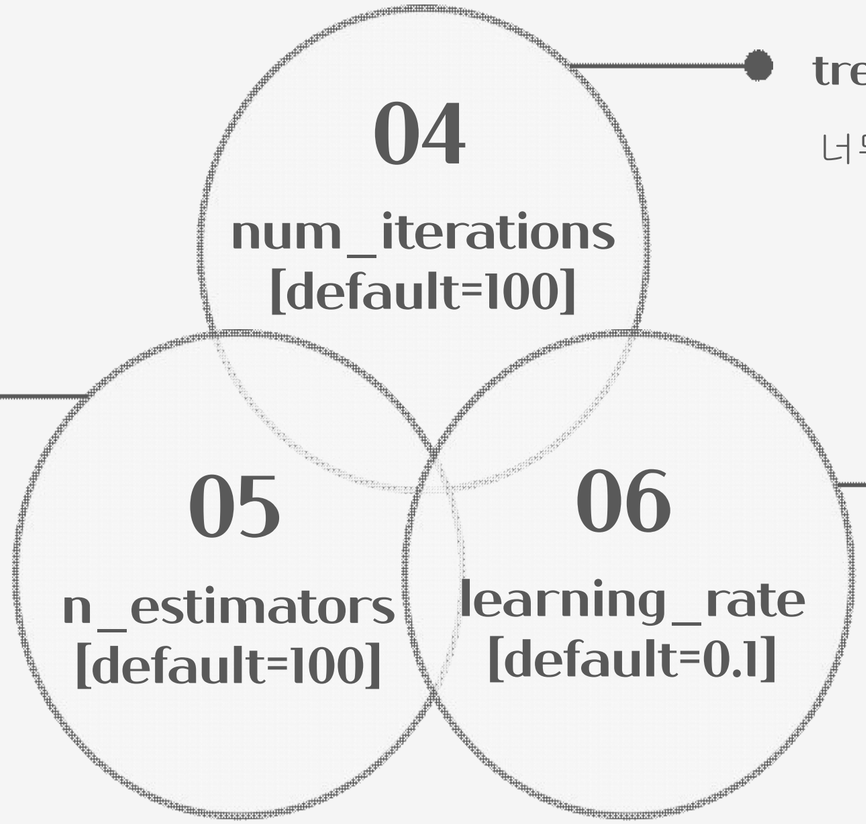
overfitting 조절 용도로 사용

subsample

각각의 step마다 사용할 sample 비율 지정

1이하의 값 사용하면 overfitting 방지 가능

1) XGBoost의 주요 parameters



tree boosting을 반복할 횟수

너무 크면 overfitting 발생

전체 tree의 개수

학습률을 고려해 적절한 값 설정해야함

학습률이자 gradient boost에서는 기울기의 의미

너무 작으면 학습속도가 지나치게 느려지고
너무 크면 기울기가 급하게 내려가 overshooting 발생

적당한 값을 설정해야 함

2) Hyperparameter Tuning

```python
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV


XgbBoost=XGBRegressor(n_estimators= 3000,objective='reg:squarederror',learning_rate = 0.03,random_state=42)

parameters ={'max_depth': [3,4,5,6], #tree의 최대 깊이, 많은 feature가 있는 경우 더더욱 높게 설정
             'num_iterations' : [1000,3000],
             'min_child_weight': [1,2],
             'subsample': [0.5,0.7]}

grid_search = GridSearchCV(XgbBoost ,parameters, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train,Y_train)
print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', -grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )
```

```
최적의 하이퍼파라미터 {'max_depth': 3, 'min_child_weight': 1, 'num_iterations': 1000, 'subsample': 0.5}
최적 모델의 cv score 0.059269175539268706
최적 모델 XGBRegressor(learning_rate=0.03, n_estimators=3000, num_iterations=1000,
             objective='reg:squarederror', random_state=42, subsample=0.5)
```

GridSearchCV를 통한 hyperparameter tuning 진행
=> n_estimators와 learning_rate 수동으로 조정(3000,0.03)
=>나머지 parameters GridSearchCV사용

최종 rmse_cv값 : 0.059269175539268706

3) overfitting 확인

```
XgbBoost=XGBRegressor(n_estimators= 3000,objective='reg:squarederror',learning_rate = 0.03,
                      random_state=42, max_depth = 3,
                      min_child_weight= 1, num_iterations= 1000, subsample =0.5)


XgbBoost_rmse_cv=rmse_cv(XgbBoost)

XgbBoost.fit(X_train,Y_train)
predictions_train = XgbBoost.predict(X_train)
predictions_test = XgbBoost.predict(X_test)
XgbBoost_train_rmse = np.sqrt(mean_squared_error(Y_train,predictions_train))
XgbBoost_test_rmse = np.sqrt(mean_squared_error(Y_test,predictions_test))


print(XgbBoost_rmse_cv,XgbBoost_train_rmse,XgbBoost_test_rmse)
```

	Model	cv_rmse	train_rmse	test_rmse
0	Xgboost(이상치 제거O)	0.059269	0.047494	0.052587

세부 튜닝된 모델의 overfitting여부를 파악하기 위해 train_rmse,test_rmse 추가 비교

test_rmse와 train_rmse값의 차이가 약 0.005밖에 나지 않으므로 **overfitting이 아니라고 판단**

06

최종결과, 의의 및 한계

최종결과

253	춘천구조대장	춘천	3589787.78489	7	몇 초 전

✓ DACON 제출결과 3589787.78489의 점수로 253등!

✓ 모델 성능대비 점수가 너무 나오지 않은 것 같아 원인 분석

✓ 이상치제거를 통해 너무 많은 sample을 삭제했다고 생각(전체의 약1/9)
=> 이상치제거하지 않고 동일 모델 사용 후 다시 제출

최종결과

WINNER 1% 4% 10% 전체 랭킹 >

#	팀	팀 멤버	점수	제출수	등록일
117	춘천구조대장	춘천	741193.05504	11	2일 전
1	gwangheekim	gw	12,348.87749	3	2년 전
2	uberphoebus	ub	114,676.39668	2	4달 전
3	윤석준	윤석	115,543.60245	1	4달 전
4	bbiibber	삐버	115,570.92163	6	4달 전
5	오성호		115,872.71577	3	4달 전

	Model	cv_rmse	train_rmse	test_rmse
0	Xgboost(이상치 제거X)	0.063242	0.049225	0.063347

✓ 이상치제거하지 않으니 741193.05504의 점수로 117등

✓ 모델의 성능은 이전보다 오히려 좋지 않았다!

한계

- ✓ 전처리에서 개별적 최선의 선택이 종합적으로는 최선의 선택이 아닐 수 있음
- ✓ 이상치 제거는 데이터 손실을 항상 감안하고 처리해야함
- ✓ 모델에서 목표로 한 최소 loss값이 항상 최적의 성능, 최고의 결과를 보장하지 않음

의의

✓ 스터디에서 배웠던 다양한 모델을 직접 실습해보고 새로운 모델도 공부해볼 수 있었음

✓ 세심한 데이터 전처리의 중요성을 배울 수 있었음

✓ 데이터 분석의 전과정을 직접 다뤄보면서 한층 더 성장할 수 있었음

발표를 들어주셔서

감사합니다 :)