사용한 데이터 - FIFA20 선수 데이터

	sofifa_id	player_url	short_name	long_name	age	dob	height_cm	weight_kg	nationality	club	...	lwb	ldm	cdm
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	32	1987-06-24	170	72	Argentina	FC Barcelona	...	68+2	66+2	66+2
1	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	34	1985-02-05	187	83	Portugal	Juventus	...	65+3	61+3	61+3
2	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Junior	27	1992-02-05	175	68	Brazil	Paris Saint-Germain	...	66+3	61+3	61+3
3	200389	https://sofifa.com/player/200389/jan-oblak/20/...	J. Oblak	Jan Oblak	26	1993-01-07	188	87	Slovenia	Atlético Madrid	...	NaN	NaN	NaN
4	183277	https://sofifa.com/player/183277/eden-hazard/2...	E. Hazard	Eden Hazard	28	1991-01-07	175	74	Belgium	Real Madrid	...	66+3	63+3	63+3
5	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	28	1991-06-28	181	70	Belgium	Manchester City	...	77+3	77+3	77+3
6	192448	https://sofifa.com/player/192448/marc-andre-te...	M. ter Stegen	Marc-André ter Stegen	27	1992-04-30	187	85	Germany	FC Barcelona	...	NaN	NaN	NaN
7	203376	https://sofifa.com/player/203376/virgil-van-di...	V. van Dijk	Virgil van Dijk	27	1991-07-08	193	92	Netherlands	Liverpool	...	79+3	83+3	83+3
8	177003	https://sofifa.com/player/177003/luka-modric/2...	L. Modrić	Luka Modrić	33	1985-09-09	172	66	Croatia	Real Madrid	...	81+3	81+3	81+3
9	209331	https://sofifa.com/player/209331/mohamed-salah...	M. Salah	Mohamed Salah Ghaly	27	1992-06-15	175	71	Egypt	Liverpool	...	70+3	67+3	67+3

분류 - 포지션 분류시 핵심 스탯은 무엇인가

KNN-라이브러리

In [1]:
```
#판다스
import pandas as pd
#시각화 라이브러리
import matplotlib.pyplot as plt
import seaborn as sns
#skit-learn 정확도,KNN,교차검증,전처리 라이브러리
from sklearn.metrics import accuracy_score
%matplotlib inline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
```

In [3]:
```
player_data=pd.read_csv('players_20.csv')
```

In [4]:
```
player_data.info()
player_data.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18278 entries, 0 to 18277
Columns: 104 entries, sofifa_id to rb
dtypes: float64(16), int64(45), object(43)
memory usage: 14.5+ MB
```

KNN-전처리,정제

In [7]:
```
from sklearn.model_selection import train_test_split
player_df = player_df.dropna(subset=['pace'])
player_df = player_df.dropna(subset=['team_position'])
positionsub = player_df[player_df['team_position']=='SUB'].index
player_df = player_df.drop(positionsub,inplace=False)
```

In [8]:
```
features = ['preferred_foot']
for feature in features:
        le=preprocessing.LabelEncoder()
        le=le.fit(player_df[feature])
        player_df[feature] = le.transform(player_df[feature])
```

In [9]:
```
train, test=train_test_split(player_df,test_size=0.2, random_state=11)
```

In [10]:
```
x_train = train[['pace', 'shooting', 'passing', 'dribbling','defending','physic','preferred_foot','height_cm','weight_kg',"attacking_cr
y_train = train[['team_position']]
x_test = test[['pace', 'shooting', 'passing', 'dribbling','defending','physic','preferred_foot','height_cm','weight_kg',"attacking_cros
y_test = test[['team_position']]
```

KNN-학습,교차검증

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
knn.predict(x_test)
```

```
C:₩Users₩jeon1₩Anaconda3₩envs₩hakat₩lib₩site-packages₩ipykernel_launcher.py:7: DataConversionWarni
n a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel(
  import sys
```

```
array(['CB', 'WB', 'CB', ..., 'FW', 'CB', 'WB'], dtype=object)
```

lb, rb,lwb,rwb → wb으로 치환
lm,rm → wm으로 치환
lw, rw,cf,rf,lf → fw으로 치환

고려할 포지션:
wb,cb,wm,cdm,cam,cm,st,fw

```
max_k_range = 100
k_list = []
for i in range(3, max_k_range, 2):
    k_list.append(i)

cross_validation_scores = []


for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_train, y_train.values.ravel(), cv=10,scoring='accuracy')
    cross_validation_scores.append(scores.mean())
```

KNN-교차검증 결과

In [13]:
```
for i in range(len(k_list)):
    print('k: {0}, 정확도 : {1}'.format(k_list[i], cross_validation_scores[i]))
```

```
k: 61, 정확도 : 0.6561263728365245
k: 63, 정확도 : 0.6561267322178137
k: 65, 정확도 : 0.6548013340233453
k: 67, 정확도 : 0.6555592691622103
k: 69, 정확도 : 0.6547998964981886
k: 71, 정확도 : 0.6534752170662986
k: 73, 정확도 : 0.6548009746420562
k: 75, 정확도 : 0.652526091081594
k: 77, 정확도 : 0.6523374159047783
k: 79, 정확도 : 0.6527158444022769
k: 81, 정확도 : 0.6517688747053073
k: 83, 정확도 : 0.6508219050083377
k: 85, 정확도 : 0.650443476510839
k: 87, 정확도 : 0.6510116583290206
k: 89, 정확도 : 0.6508222643896266
k: 91, 정확도 : 0.6510123770915991
k: 93, 정확도 : 0.6508215456270485
k: 95, 정확도 : 0.6494961474325801
k: 97, 정확도 : 0.6453269650968891
k: 99, 정확도 : 0.6449488959806796
```

KNN-결과 대조

```
comparison = pd.DataFrame(
    {'prediction':pred, 'ground_truth':y_test.values.ravel()})
comparison
```

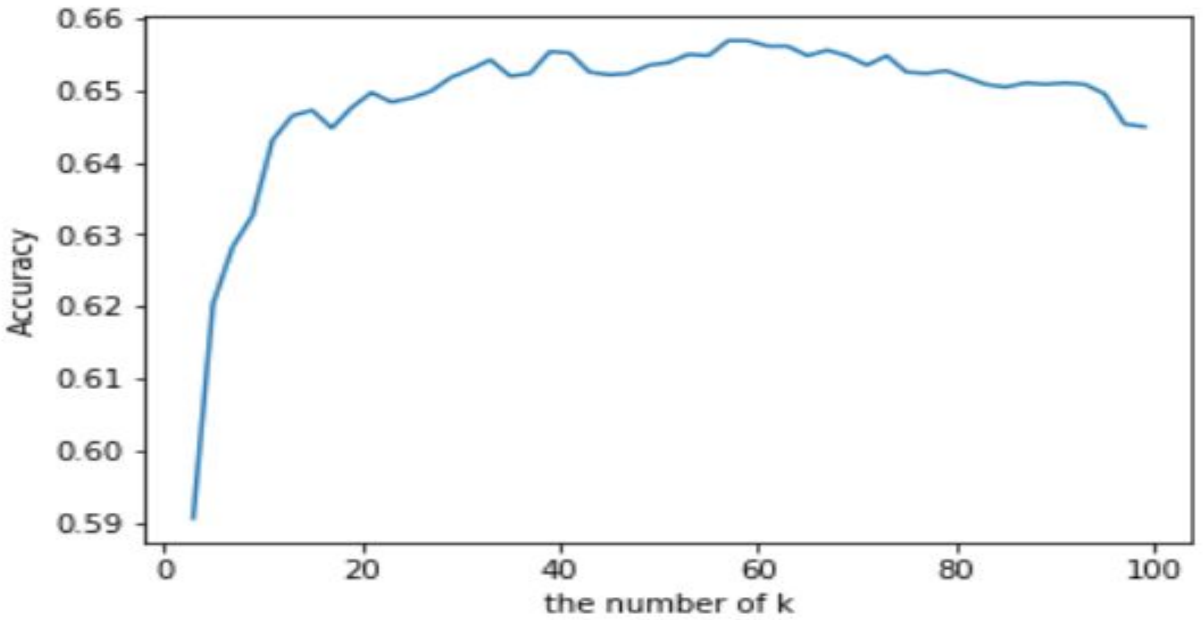
```
The best number of k:59
```

	prediction	ground_truth
0	CB	CB
1	WB	WB
2	CB	CB
3	ST	ST
4	WM	WM
...	...	...
1315	DM	DM
1316	CM	CM
1317	AM	FW
1318	CB	CB
1319	WB	WB

1320 rows × 2 columns

KNN-정확도 그래프

In [15]:
```
plt.plot(k_list, cross_validation_scores)
plt.xlabel('the number of k')
plt.ylabel('Accuracy')
plt.show()
```



KNN-분류 스코어 정리

In [16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score

def get_clf_eval(y_test, pred):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred,average='weighted')
    recall = recall_score(y_test , pred,average='weighted')
    f1 = f1_score(y_test,pred,average='weighted')
    # ROC-AUC 추가
    ##roc_auc = roc_auc_score(y_test, pred)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},₩
    F1: {3:.4f}'.format(accuracy, precision, recall, f1))
```

In [17]:
```
get_clf_eval(y_test,pred)
```

```
오차 행렬
[[  9   0  26   2   2   6   1  34]
 [  0 245   7  15   0   0  15   0]
 [  5   9 110  36   1   1  13  20]
 [  0  12  66  38   0   0  12   2]
 [  5   0   6   0   1  22   3  49]
 [  2   0   5   0   2 136   0  17]
 [  1   7  14   5   0   0 196   5]
 [  3   0  24   1   2  12  14 101]]
정확도: 0.6333, 정밀도: 0.6035, 재현율: 0.6333,    F1: 0.6065
```

분류 - Ensemble

Ensemble - 라이브러리

In [1]:

```
#판다스
import pandas as pd
#넘파이
import numpy as np
#Light GBM
import lightgbm
from lightgbm import LGBMClassifier
#skit-learn 라이브러리
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
#경고 무시
import warnings
warnings.filterwarnings('ignore')

data_df = pd.read_csv("players_20.csv",header=0)
data_df.head(3)
```

Ensemble - 전처리, 정제

In [2]:
```
def format_features(df):
    features = ['team_position']
    for feature in features:
        le=preprocessing.LabelEncoder()
        le=le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df
```

In [3]:
```
#세부스탯이 존재하지 않는 선수들과 팀 포지션이 존재하지 않는 선수들 제거
data_df = data_df.dropna(subset=['pace'])
data_df = data_df.dropna(subset=['team_position'])
#포지션이 지정되지 않고 후보선수로 지정되어있는 선수들 제거
positionsub = data_df[data_df['team_position']=='SUB'].index
data_df = data_df.drop(positionsub,inplace=False)
```

In [4]:
```
x_data = data_df[['preferred_foot','weak_foot','skill_moves','height_cm','weight_kg',"attacking_crossing",'attacking_finishing','attack
y_data = data_df[['team_position']]
#포지션 인코딩
y_data = format_features(y_data)
#주 사용 발(LEFT,RIGHT) 인코딩
feature = ['preferred_foot']
le=preprocessing.LabelEncoder()
le=le.fit(x_data[feature])
x_data[feature] = le.transform(x_data[feature])
```

Ensemble - LR, KNN 앙상블 학습/예측

In [28]:
```
# 개별 모델은 로지스틱 회귀와 KNN 임.
lr_clf = LogisticRegression()
knn_clf = KNeighborsClassifier(n_neighbors=8)

# 개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기
vo_clf = VotingClassifier( estimators=[('LR',lr_clf),('KNN',knn_clf)] , voting='soft' )

X_train, X_test, y_train, y_test = train_test_split(x_data, y_data,
                                                    test_size=0.2 , random_state= 156)

# VotingClassifier 학습/예측/평가.
vo_clf.fit(X_train , y_train)
pred = vo_clf.predict(X_test)
print('Voting 분류기 정확도: {0:.4f}'.format(accuracy_score(y_test , pred)))


# 개별 모델의 학습/예측/평가.
classifiers = [lr_clf, knn_clf]
for classifier in classifiers:
    classifier.fit(X_train , y_train)
    pred = classifier.predict(X_test)
    class_name= classifier.__class__.__name__
    print('{0} 정확도: {1:.4f}'.format(class_name, accuracy_score(y_test , pred)))
```

```
Voting 분류기 정확도: 0.6818
LogisticRegression 정확도: 0.6894
KNeighborsClassifier 정확도: 0.6402
```

분류 - Random Forest

Ensemble에서 정제한 자료 이용합니다

RF - 학습 및 예측 결과

In [6]:
```
# 랜덤 포레스트 학습 및 별도의 테스트 셋으로 예측 성능 평가
rf_clf = RandomForestClassifier(random_state=0)
rf_clf.fit(X_train , y_train)
pred = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test , pred)
print('랜덤 포레스트 정확도: {0:.4f}'.format(accuracy))
```

```
랜덤 포레스트 정확도: 0.6682
```

RF - 하이퍼 파라미터 튜닝

In [7]:
```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators':[300],
    'max_depth' : [6, 8, 10, 12],
    'min_samples_leaf' : [8, 12, 18 ],
    'min_samples_split' : [8, 16, 20]
}
# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf_clf = RandomForestClassifier(random_state=0, n_jobs=-1)
grid_cv = GridSearchCV(rf_clf , param_grid=params , cv=5, n_jobs=-1 )
grid_cv.fit(X_train , y_train)

print('최적 하이퍼 파라미터:₩n', grid_cv.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

```
최적 하이퍼 파라미터:
 {'max_depth': 12, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 300}
최고 예측 정확도: 0.6652
```

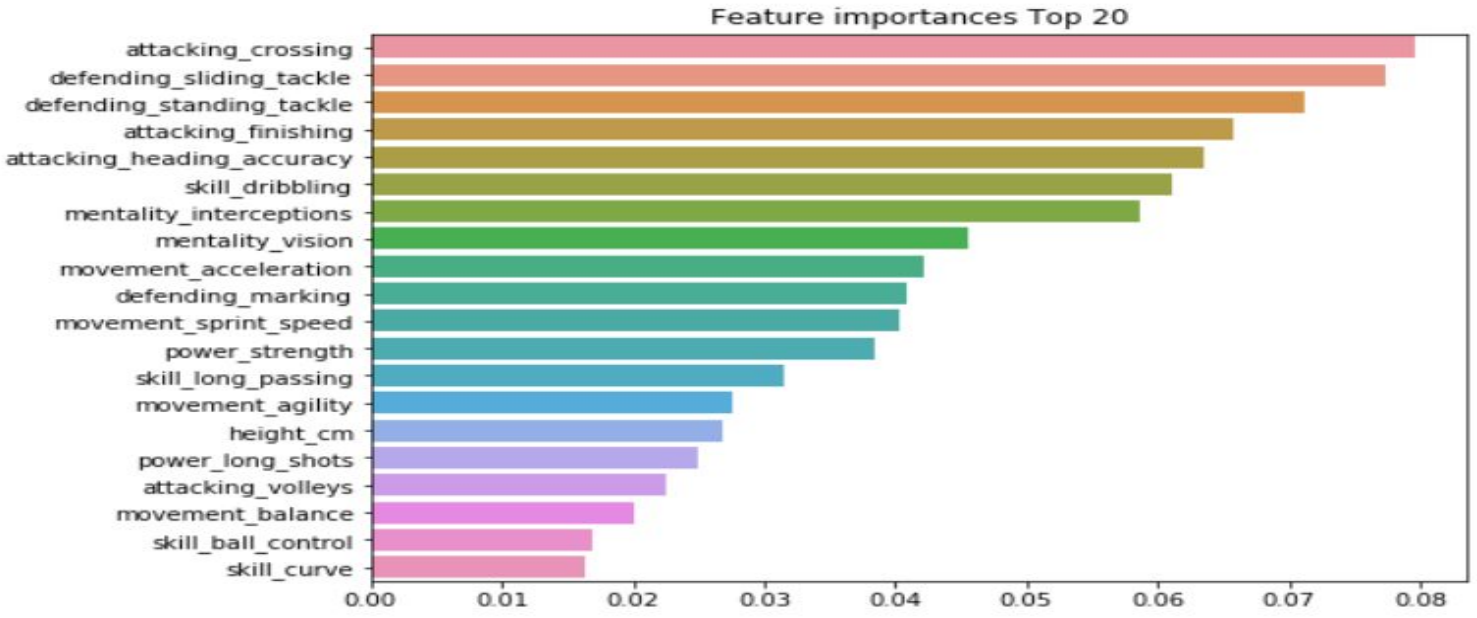
튜닝된 하이퍼 파라미터로 재 학습 및 예측/평가

In [8]:
```
rf_clf1 = RandomForestClassifier(n_estimators=300, max_depth=10, min_samples_leaf=8, ₩
                                 min_samples_split=8, random_state=0)
rf_clf1.fit(X_train , y_train)
pred = rf_clf1.predict(X_test)
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test , pred)))
```

```
예측 정확도: 0.6614
```

RF - feature 중요도 시각화

```
ftr_importances_values = rf_clf1.feature_importances_
ftr_importances = pd.Series(ftr_importances_values,index=X_train.columns  )
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]

plt.figure(figsize=(8,6))
plt.title('Feature importances Top 20')
sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
```



분류 - Light GBM

Ensemble에서 정제한 자료 이용합니다

LGBM - 학습/예측

In [10]:
```
lgbm_wrapper = LGBMClassifier(n_estimators=400)

evals = [(X_test, y_test)]
lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="logloss",
                 eval_set=evals, verbose=True)
lgbm_preds = lgbm_wrapper.predict(X_test)
lgbm_accuracy = accuracy_score(y_test, lgbm_preds)
print('LightGBM 정확도: {0:.4f}'.format(lgbm_accuracy))
```

```
[149]   valid_0's multi_logloss: 0.970735
[150]   valid_0's multi_logloss: 0.972396
[151]   valid_0's multi_logloss: 0.973943
[152]   valid_0's multi_logloss: 0.975368
[153]   valid_0's multi_logloss: 0.976405
[154]   valid_0's multi_logloss: 0.977509
[155]   valid_0's multi_logloss: 0.979122
[156]   valid_0's multi_logloss: 0.980333
[157]   valid_0's multi_logloss: 0.981313
[158]   valid_0's multi_logloss: 0.982526
[159]   valid_0's multi_logloss: 0.983375
[160]   valid_0's multi_logloss: 0.98452
[161]   valid_0's multi_logloss: 0.985635
[162]   valid_0's multi_logloss: 0.986331
[163]   valid_0's multi_logloss: 0.987271
[164]   valid_0's multi_logloss: 0.988068
[165]   valid_0's multi_logloss: 0.988412
Early stopping, best iteration is:
[65]    valid_0's multi_logloss: 0.903336
LightGBM 정확도: 0.6720
```

LGBM - 분류 스코어 확인

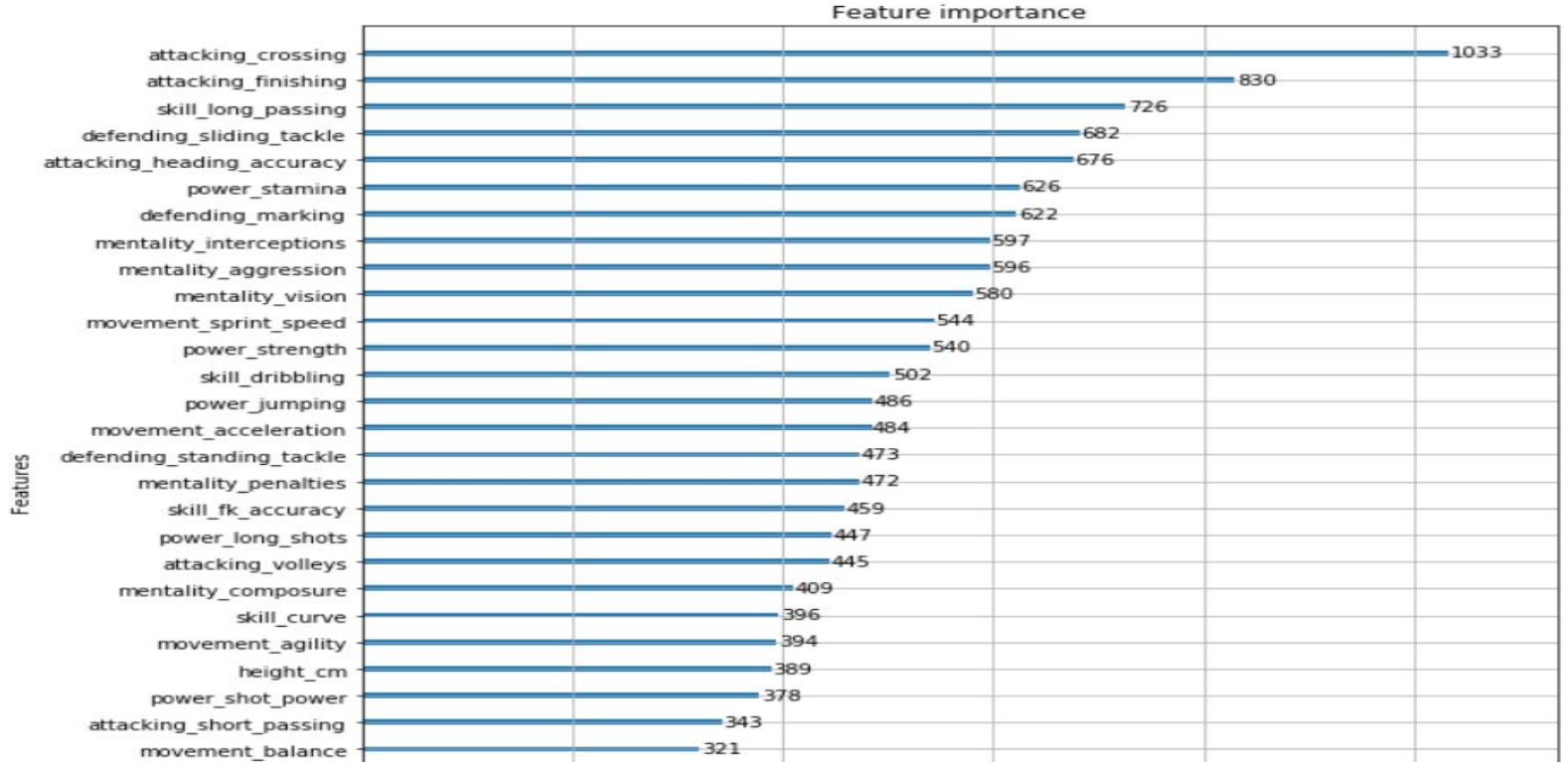
In [12]:
```
get_clf_eval(y_test, lgbm_preds)
```

```
오차 행렬
[[ 14   1  22   1   5   5   3  33]
 [  0 254   1   8   0   0  14   0]
 [  6  11 106  49   0   5   7  14]
 [  2  10  52  63   0   0  14   0]
 [  2   0   8   0  10  14   0  32]
 [  3   0   2   0   4 127   0  15]
 [  0  10   5   0   1   0 222   9]
 [ 13   0  15   0   6  11  20  91]]
정확도: 0.6720, 정밀도: 0.6523, 재현율: 0.6720,F1: 0.6556
```

LGBM - feature 중요도 시각화

```
fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(lgbm_wrapper, ax=ax)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1aea2897588>



분류 - Stacking Ensemble

Stacking Ensemble - KNN,RF,GBM 학습

In [15]:
```
# 개별 ML 모델을 위한 Classifier 생성.
knn_clf  = KNeighborsClassifier(n_neighbors=8)
rf_clf = RandomForestClassifier(n_estimators=500, random_state=0)
gb_clf = GradientBoostingClassifier(n_estimators=500,random_state=0)
```

In [16]:
```
# 개별 모델들을 학습.
knn_clf.fit(X_train, y_train)
rf_clf.fit(X_train , y_train)
gb_clf.fit(X_train, y_train)
```

Out[16]:
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=500,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=0, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Stacking Ensemble - KNN,RF,GBM 예측

In [17]:
```
# 학습된 개별 모델들이 각자 반환하는 예측 데이터 셋을 생성하고 개별 모델의 정확도 측정.
knn_pred = knn_clf.predict(X_test)
rf_pred = rf_clf.predict(X_test)
gb_pred = gb_clf.predict(X_test)

print('KNN 정확도: {0:.4f}'.format(accuracy_score(y_test, knn_pred)))
print('랜덤 포레스트 정확도: {0:.4f}'.format(accuracy_score(y_test, rf_pred)))
print('gbm 정확도 : {0:.4f}'.format(accuracy_score(y_test,gb_pred)))
```

```
KNN 정확도: 0.6402
랜덤 포레스트 정확도: 0.6705
gbm 정확도 : 0.6515
```

In [18]:
```
pred = np.array([knn_pred, rf_pred,lgbm_preds])
print(pred.shape)

# transpose를 이용해 행과 열의 위치 교환. 컬럼 레벨로 각 알고리즘의 예측 결과를 피쳐로 만듦.
pred = np.transpose(pred)
print(pred.shape)
```

```
(3, 1320)
(1320, 3)
```

Stacking Ensemble - LGBM으로 최종 학습 및 예측

In [19]:
```
lgbm_wrapper = LGBMClassifier(n_estimators=800)

# LightGBM도 XGBoost와 동일하게 조기 중단 수행 가능.
lgbm_wrapper.fit(pred, y_test,verbose=True)
lgbm_preds = lgbm_wrapper.predict(pred)
lgbm_accuracy = accuracy_score(y_test, lgbm_preds)
print('LightGBM 정확도: {0:.4f}'.format(lgbm_accuracy))
```

```
LightGBM 정확도: 0.7205
```

In [20]:
```
get_clf_eval(y_test, lgbm_preds)
```

```
오차 행렬
[[ 18   1  17   1   5   5   3  34]
 [  0 260   1   2   0   0  14   0]
 [  6  12 126  35   0   4   3  12]
 [  1  11  52  63   0   0  13   1]
 [  1   0   5   0  17  13   0  30]
 [  0   0   1   0   2 132   0  16]
 [  3  10   5   0   0   0 222   7]
 [  4   0  10   0   5   9  15 113]]
정확도: 0.7205, 정밀도: 0.7125, 재현율: 0.7205,F1: 0.7045
```

회귀 - 어떤 세부스탯이 공격수 몸값에 큰 영향을 줄것인가

LR-라이브러리

In [78]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

fifa20 = pd.read_csv('fifa20_data.csv')
fifa20
fifa20.head(3)
```

Out[78]:

	sofifa_id	player_url	short_name	long_name	age	dob	height_cm	weight_kg	nationality	club	...	lwb	ldm	cdm
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andr챕s Messi Cuccittini	32	1987.6.24	170	72	Argentina	FC Barcelona	...	68+2	66+2	66+2
1	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	34	1985.2.5	187	83	Portugal	Juventus	...	65+3	61+3	61+3
2	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Junior	27	1992.2.5	175	68	Brazil	Paris Saint-Germain	...	66+3	61+3	61+3

3 rows × 104 columns

LR- 포지션은 공격수만으로 분류. 나머지 포지션은

In [80]:
```
##공격수 데이터만 보기전, 팀포지션이 빈값으로 되어 있는 데이터를 sub로 바꾸고 지우겟다.
fifaDF['team_position'].fillna('sub',inplace=True)
```

In [81]:
```
##공격수 데이터만 보겟다!!
data_rcm= fifaDF[fifaDF['team_position']=='RCM'].index
fifaDF.drop(data_rcm, axis=0, inplace=True)
data_lcm= fifaDF[fifaDF['team_position']=='LCM'].index
fifaDF.drop(data_lcm, axis=0, inplace=True)
data_cm= fifaDF[fifaDF['team_position']=='CM'].index
fifaDF.drop(data_cm, axis=0, inplace=True)
data_cdm= fifaDF[fifaDF['team_position']=='CDM'].index
fifaDF.drop(data_cdm, axis=0, inplace=True)
data_rdm= fifaDF[fifaDF['team_position']=='RDM'].index
fifaDF.drop(data_rdm, axis=0, inplace=True)
data_ldm= fifaDF[fifaDF['team_position']=='LDM'].index
fifaDF.drop(data_ldm, axis=0, inplace=True)
data_cam= fifaDF[fifaDF['team_position']=='CAM'].index
fifaDF.drop(data_cam, axis=0, inplace=True)
data_lam= fifaDF[fifaDF['team_position']=='LAM'].index
fifaDF.drop(data_lam, axis=0, inplace=True)
data_ram= fifaDF[fifaDF['team_position']=='RAM'].index
fifaDF.drop(data_ram, axis=0, inplace=True)
data_rm= fifaDF[fifaDF['team_position']=='RM'].index
fifaDF.drop(data_rm, axis=0, inplace=True)
data_lm= fifaDF[fifaDF['team_position']=='LM'].index
fifaDF.drop(data_lm, axis=0, inplace=True)
data_cb= fifaDF[fifaDF['team_position']=='CB'].index
fifaDF.drop(data_cb, axis=0, inplace=True)
data_rcb= fifaDF[fifaDF['team_position']=='RCB'].index
fifaDF.drop(data_rcb, axis=0, inplace=True)
data_lcb= fifaDF[fifaDF['team_position']=='LCB'].index
fifaDF.drop(data_lcb, axis=0, inplace=True)
data_rb= fifaDF[fifaDF['team_position']=='RB'].index
fifaDF.drop(data_rb, axis=0, inplace=True)
data_lb= fifaDF[fifaDF['team_position']=='LB'].index
fifaDF.drop(data_lb, axis=0, inplace=True)
data_rwb= fifaDF[fifaDF['team_position']=='RWB'].index
fifaDF.drop(data_rwb, axis=0, inplace=True)
data_lwb= fifaDF[fifaDF['team_position']=='LWB'].index
fifaDF.drop(data_lwb, axis=0, inplace=True)
data_gk= fifaDF[fifaDF['team_position']=='GK'].index
fifaDF.drop(data_gk, axis=0, inplace=True)
data_res= fifaDF[fifaDF['team_position']=='RES'].index
```

공격수 포메이션인
rw,lw,st,cf,rf,lf만 고려.

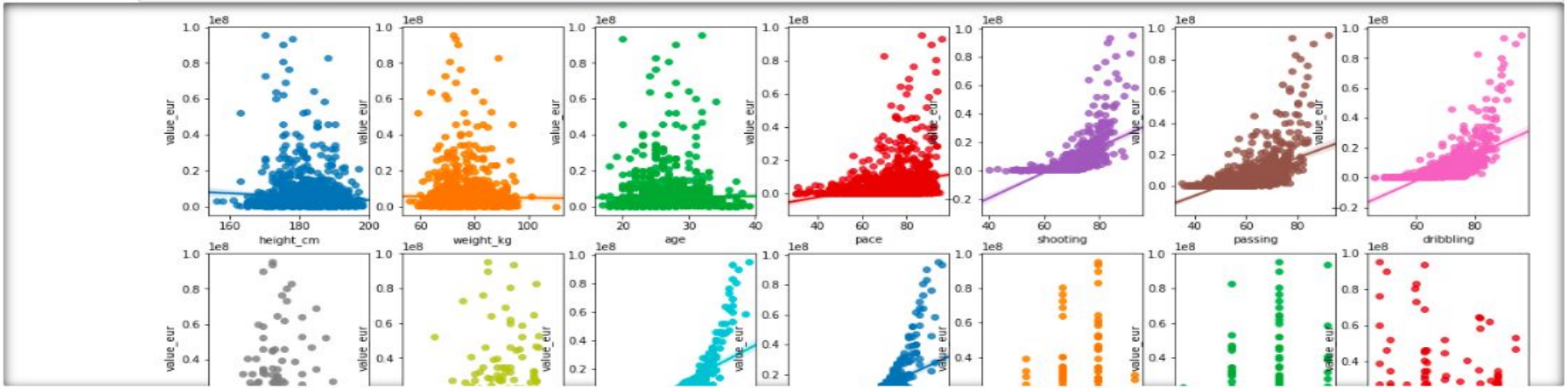
LR- 결측치 제거, 데이터 전처리.

In [82]:
```
#불필요한 칼럼 제거
def drop_features(df):
    df.drop(['sofifa_id','player_url','short_name','long_name','dob','work_rate','body_type','player_positions','real_
    df_value_zero1= df[df['wage_eur']==0].index
    df.drop(df_value_zero1)
    return df

#레이블 인코딩 수행
def format_features(df):
    features = ['nationality','club','team_position','preferred_foot']
    for feature in features:
        le=preprocessing.LabelEncoder()
        le=le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df

#앞에서 설정한 데이터 전처리 함수 호출
def transform_features(df):
    df=drop_features(df)
    df=format_features(df)

    return df
```

In [83]:
```
from sklearn import preprocessing
#몸값이 0원인 선수들을 미리 전처리 시킴. 왜 함수에 넣어서 전처리 안했나? --> X_fifa_df=fifa_df.drop('value_eur', axis=1) 이 코드에서 value
fifaDF_value_zero= fifaDF[fifaDF['value_eur']==0].index
fifaDF.drop(fifaDF_value_zero, axis=0, inplace=True)
#팀포지션이 정확히 없는 선수의 세부 스탯 지우기.
fifaDF = fifaDF.dropna(subset=['pace'])

#원본 데이터를 재로딩하고, 피처 데이터 세트와 레이블 데이터 세트 추출.
fifaDF
Y_fifaDF=fifaDF['value_eur']
X_fifaDF=fifaDF.drop('value_eur', axis=1)
fifaDF=transform_features(fifaDF)
```

LR- 측정할 feature 값과 몸값과의 상관계수

In [85]:
```
# 6개의 행과 7개의 열을 가진 subplots를 이용. axs는 6x7개의 ax를 가짐.
fig, axs = plt.subplots(figsize=(20,30) , ncols=7 , nrows=7)
lm_features = ['height_cm','weight_kg','age','pace','shooting','passing','dribbling','defending','physic','overall','po
for i , feature in enumerate(lm_features):
    row = int(i/7)
    col = i%7
    # 시본의 regplot을 이용해 산점도와 선형 회귀 직선을 함께 표현
    sns.regplot(x=feature , y='value_eur',data=fifaDF , ax=axs[row][col])
```



In [86]:
```
need_corr_features=['height_cm','weight_kg','age','pace','shooting','passing','dribbling','defending','physic','overall

fifaDF_temp=pd.DataFrame(data=fifaDF[need_corr_features])
corr=fifaDF_temp.corr(method='pearson')
print(corr)
```

LR- 측정할 feature 값과 몸값과의 상관계수

	value_eur
height_cm	-0.062411
weight_kg	-0.013474
age	0.014200
pace	0.249083
shooting	0.634707
passing	0.556082
dribbling	0.635530
defending	0.253337
physic	0.189323
overall	0.747326
potential	0.690870
weak_foot	0.190228
skill_moves	0.441155
nationality	-0.005326
club	0.012572
preferred_foot	-0.082438
attacking_crossing	0.410156
attacking_finishing	0.601566
attacking_heading_accuracy	0.215288
attacking_short_passing	0.596918
attacking_volleys	0.521186
skill_dribbling	0.614249
skill_curve	0.471028
skill_fk_accuracy	0.368969
skill_long_passing	0.427514
skill_ball_control	0.671803
movement_acceleration	0.245764
movement_sprint_speed	0.240138
movement_agility	0.274310
movement_reactions	0.654047
movement_balance	0.210720
power_shot_power	0.510311
power_jumping	0.112388
power_stamina	0.324999
power_strength	0.042462
power_long_shots	0.564509
mentality_aggression	0.188589
mentality_interceptions	0.199754
mentality_positioning	0.623113
mentality_vision	0.536849
mentality_penalties	0.397956
mentality_composure	0.576456
value_eur	1.000000

[43 rows x 43 columns]

LR- 측정할 feature 값들의 다중 공선성

In [87]:
```
#다중공선성 판별
from statsmodels.stats.outliers_influence import variance_inflation_factor
from patsy import dmatrices
import statsmodels.api as sm;
```

x와 y의 상관계수가 0.2이상인 것과 다중공선성이 10미만인 변수들을 선별했다.

In [88]:
```
y, X = dmatrices('value_eur ~ international_reputation + physic + potential + skill_moves + attacking_crossing + attack

vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["features"] = X.columns
vif
```

LR- 측정할 feature 값들의 다중 공선성

1	1.677095	international_reputation
2	3.566775	physic
3	3.273376	potential
4	2.540150	skill_moves
5	3.562935	attacking_crossing
6	5.929671	attacking_finishing
7	3.490425	attacking_heading_accuracy
8	6.273905	attacking_short_passing
9	3.563727	attacking_volleys
10	7.861263	skill_dribbling
11	3.376377	skill_curve
12	2.557029	skill_fk_accuracy
13	3.570619	skill_long_passing
14	8.629954	skill_ball_control
15	5.296485	movement_reactions
16	7.401639	movement_acceleration
17	5.434839	movement_sprint_speed
18	4.273596	movement_agility
19	2.803173	movement_balance
20	3.587928	power_shot_power
21	2.096377	power_stamina
22	3.431549	power_long_shots
23	7.090278	mentality_positioning
24	4.095903	mentality_vision
25	2.113159	mentality_penalties
26	3.858805	mentality_composure
27	1.690435	defending

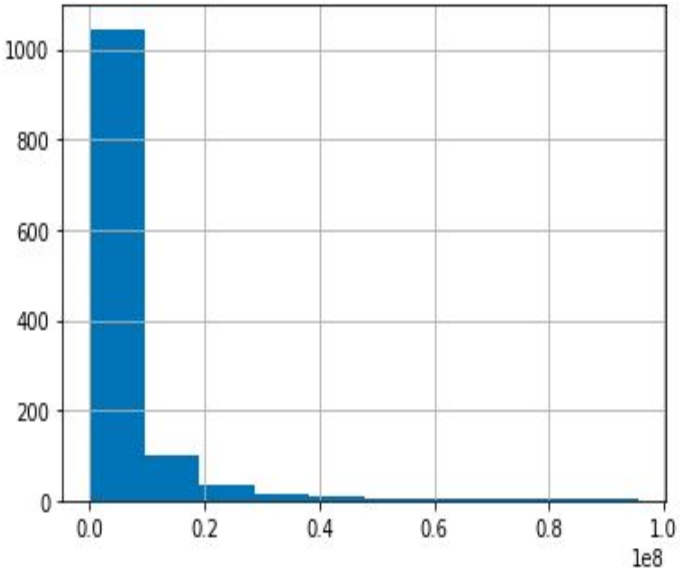
In [90]:
```
#드롭한 변수가 잘 드롭되었는지 확인.
X_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1223 entries, 0 to 18148
Data columns (total 28 columns):
potential                    1223 non-null int64
international_reputation     1223 non-null int64
skill_moves                  1223 non-null int64
pace                         1223 non-null float64
defending                    1223 non-null float64
attacking_crossing           1223 non-null int64
attacking_finishing          1223 non-null int64
attacking_heading_accuracy   1223 non-null int64
attacking_short_passing      1223 non-null int64
attacking_volleys            1223 non-null int64
skill_dribbling              1223 non-null int64
skill_curve                  1223 non-null int64
skill_fk_accuracy            1223 non-null int64
skill_long_passing           1223 non-null int64
skill_ball_control           1223 non-null int64
movement_acceleration        1223 non-null int64
movement_sprint_speed        1223 non-null int64
movement_agility             1223 non-null int64
movement_reactions           1223 non-null int64
movement_balance             1223 non-null int64
power_shot_power             1223 non-null int64
power_stamina                1223 non-null int64
power_long_shots             1223 non-null int64
mentality_interceptions      1223 non-null int64
mentality_positioning        1223 non-null int64
mentality_vision             1223 non-null int64
mentality_penalties          1223 non-null int64
mentality_composure          1223 non-null int64
dtypes: float64(2), int64(26)
```

LR- y_target(몸값) 로그변환

```
In [91]: y_target.hist()
```

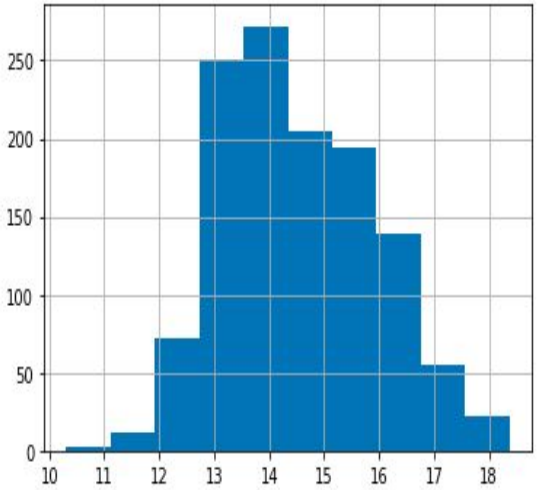
```
Out[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1a23fe6750>
```




```
In [92]: y_log_transform=np.log1p(y_target)
```

```
In [93]: y_log_transform.hist()
```

```
Out[93]: <matplotlib.axes._subplots.AxesSubplot at 0x1a227ee750>
```



LR- 몸값과 선정한 피처값의 rmsle(root mean square log error) 구하기

RMSLE 구하는 함수.

In [94]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# log 값 변환 시 NaN등의 이슈로 log() 가 아닌 log1p() 를 이용하여 RMSLE 계산
def rmsle(y, pred):
    log_y = np.log1p(y)
    log_pred = np.log1p(pred)
    squared_error = (log_y - log_pred) ** 2
    rmsle = np.sqrt(np.mean(squared_error))
    return rmsle

# 사이킷런의 mean_square_error() 를 이용하여 RMSE 계산
def rmse(y,pred):
    return np.sqrt(mean_squared_error(y,pred))

# MSE, RMSE, RMSLE 를 모두 계산
def evaluate_regr(y,pred):
    rmsle_val = rmsle(y,pred)
    rmse_val = rmse(y,pred)
    # MSE 는 scikit learn의 mean_absolute_error() 로 계산
    mse_val = mean_absolute_error(y,pred)
    print('RMSLE: {0:.3f}, RMSE: {1:.3F}, MSE: {2:.3F}'.format(rmsle_val, rmse_val, mse_val))
```

LR- 몸값과 선정한 피처값의 rmsle(root mean square log error) 구하기

y_target값을 로그변환하고 지수화 하고 다시 로그변환 함수를 활용하여 테스트를 돌림.

linear regression으로 rmsle와 회귀 계수 구해보기.

In [95]:
```
from sklearn.model_selection import train_test_split

# 타겟 컬럼인 value_eur 값을 log1p 로 Log 변환
y_target_log = np.log1p(y_target)

# 로그 변환된 y_target_log를 반영하여 학습/테스트 데이터 셋 분할
X_train, X_test, y_train, y_test = train_test_split(X_data, y_target_log, test_size=0.3, random_state=0)
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
pred = lr_reg.predict(X_test)

# 테스트 데이터 셋의 Target 값은 Log 변환되었으므로 다시 expm1를 이용하여 원래 scale로 변환
y_test_exp = np.expm1(y_test)

# 예측 값 역시 Log 변환된 타겟 기반으로 학습되어 예측되었으므로 다시 exmp1으로 scale변환
pred_exp = np.expm1(pred)

evaluate_regr(y_test_exp ,pred_exp)
```

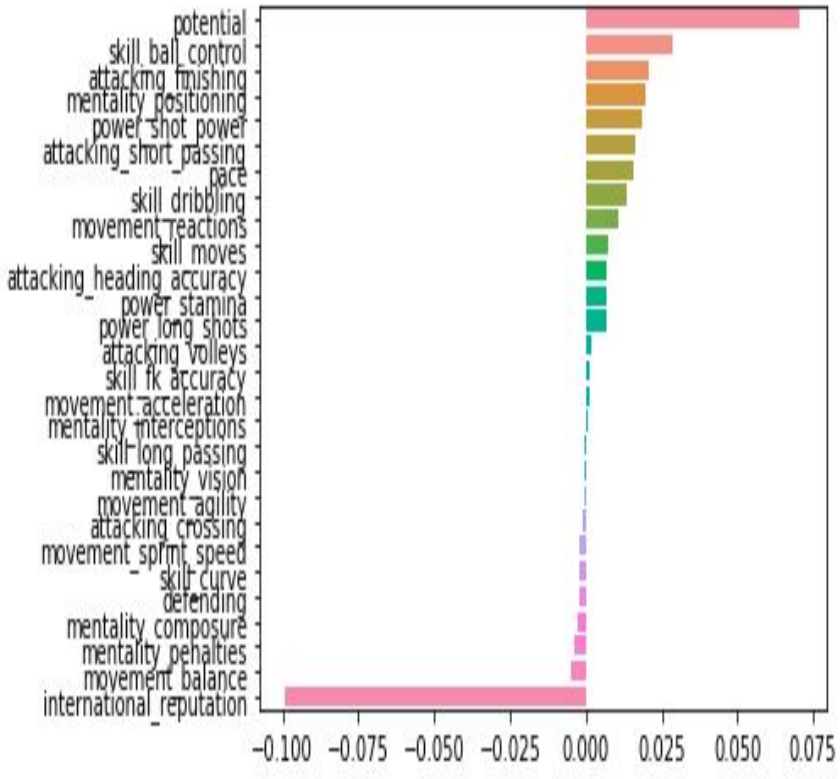
```
RMSLE: 0.345, RMSE: 8009669.735, MSE: 1416086.709
```

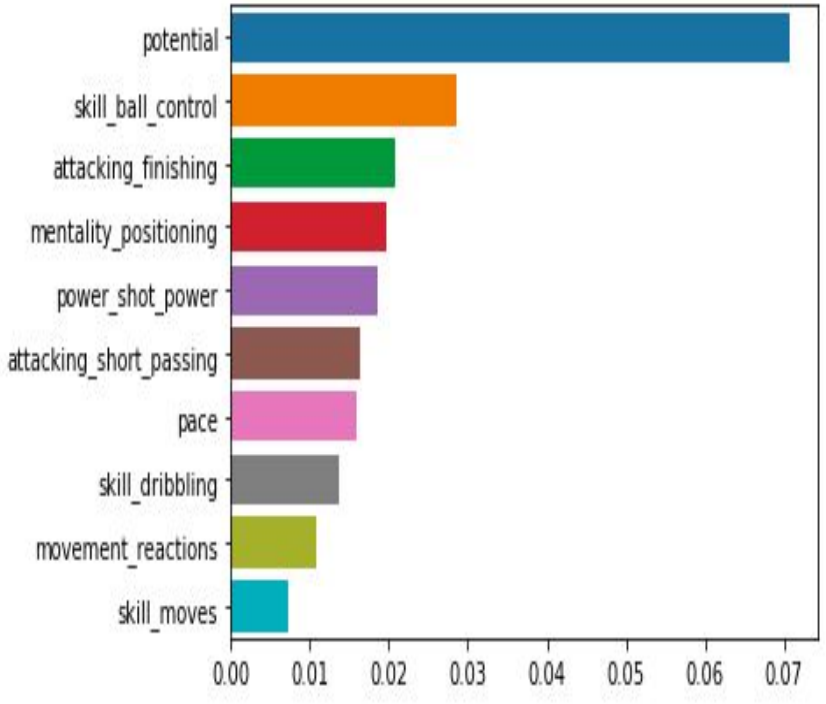
LR- 몸값과 선정한 피처값의 회귀계수 구하기

In [96]:
```
coef = pd.Series(lr_reg.coef_, index=X_data.columns)
coef_sort = coef.sort_values(ascending=False)
print(coef_sort)
sns.barplot(x=coef_sort.values, y=coef_sort.index)
```

```
potential                    0.070721
skill_ball_control           0.028639
attacking_finishing          0.020732
mentality_positioning        0.019592
power_shot_power             0.018624
attacking_short_passing      0.016258
pace                         0.015819
skill_dribbling              0.013746
movement_reactions           0.010789
skill_moves                  0.007257
attacking_heading_accuracy   0.007075
power_stamina                0.006822
power_long_shots             0.006764
attacking_volleys            0.001617
skill_fk_accuracy            0.001361
movement_acceleration        0.001092
mentality_interceptions      0.000453
skill_long_passing          -0.000149
mentality_vision            -0.000168
movement_agility            -0.000282
attacking_crossing          -0.000925
movement_sprint_speed       -0.001807
skill_curve                 -0.001886
defending                   -0.002017
mentality_composure         -0.002460
mentality_penalties         -0.003791
movement_balance            -0.004624
international_reputation    -0.099194
dtype: float64
```

LR- 몸값과 선정한 피처값의 회귀계수 구하기





회귀 - LR,RIDGE,LASSO

위에서 정제한 자료 이용합니다

선형 회귀 모델 학습/예측

선형 회귀 모델 학습/예측/평가

In [99]:
```
def get_rmse(model):
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test , pred)
    rmse = np.sqrt(mse)
    print('{0} 로그 변환된 RMSLE: {1}'.format(model.__class__.__name__,np.round(rmse, 3)))
    return rmse

def get_rmses(models):
    rmses = [ ]
    for model in models:
        rmse = get_rmse(model)
        rmses.append(rmse)
    return rmses
```

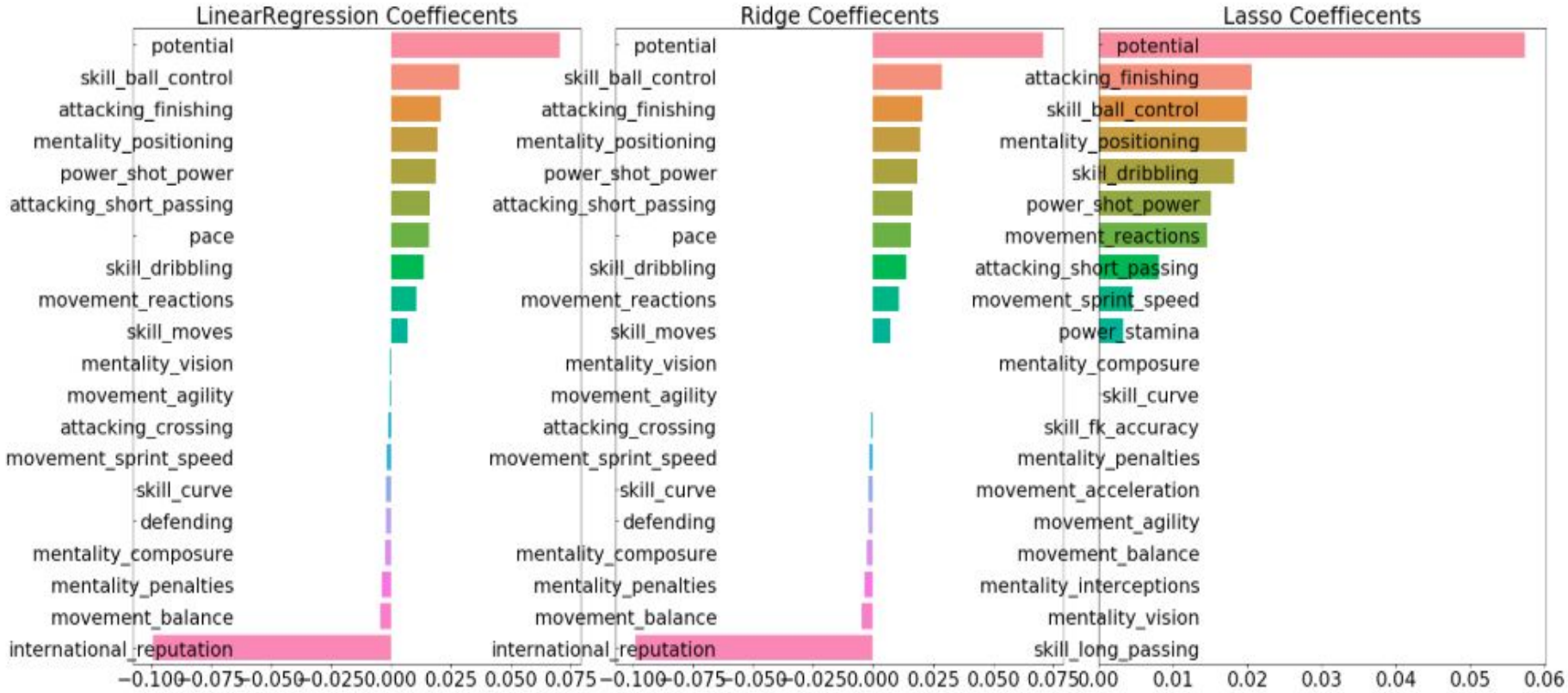
선형 회귀 모델 학습/예측

In [100]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 타겟 컬럼인 value_eur 값을 log1p 로 Log 변환
y_target_log = np.log1p(y_target)

# 로그 변환된 y_target_log를 반영하여 학습/테스트 데이터 셋 분할
X_train, X_test, y_train, y_test = train_test_split(X_data, y_target_log, test_size=0.3, random_state=0)

# LinearRegression, Ridge, Lasso 학습, 예측, 평가
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)

ridge_reg = Ridge()
ridge_reg.fit(X_train, y_train)

lasso_reg = Lasso()
lasso_reg.fit(X_train, y_train)

models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)
```

```
LinearRegression 로그 변환된 RMSLE: 0.345
Ridge 로그 변환된 RMSLE: 0.345
Lasso 로그 변환된 RMSLE: 0.397
```

Out[100]:
```
[0.344587866603889, 0.34457010403657773, 0.3966040904240122]
```

선형 회귀 모델 학습/예측 - 시각화



선형 회귀 모델 학습/예측 - 하이퍼 파라미터 튜닝

In [104]:
```
from sklearn.model_selection import GridSearchCV

def get_best_params(model, params):
    grid_model = GridSearchCV(model, param_grid=params,
                              scoring='neg_mean_squared_error', cv=5)
    grid_model.fit(X_data, y_target_log)
    rmse = np.sqrt(-1* grid_model.best_score_)
    print('{0} 5 CV 시 최적 평균 RMSE 값: {1}, 최적 alpha:{2}'.format(model.__class__.__name__,
                                        np.round(rmse, 4), grid_model.best_params_))
    return grid_model.best_estimator_

ridge_params = { 'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20] }
lasso_params = { 'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1,5, 10] }
best_ridge = get_best_params(ridge_reg, ridge_params)
best_lasso = get_best_params(lasso_reg, lasso_params)
```

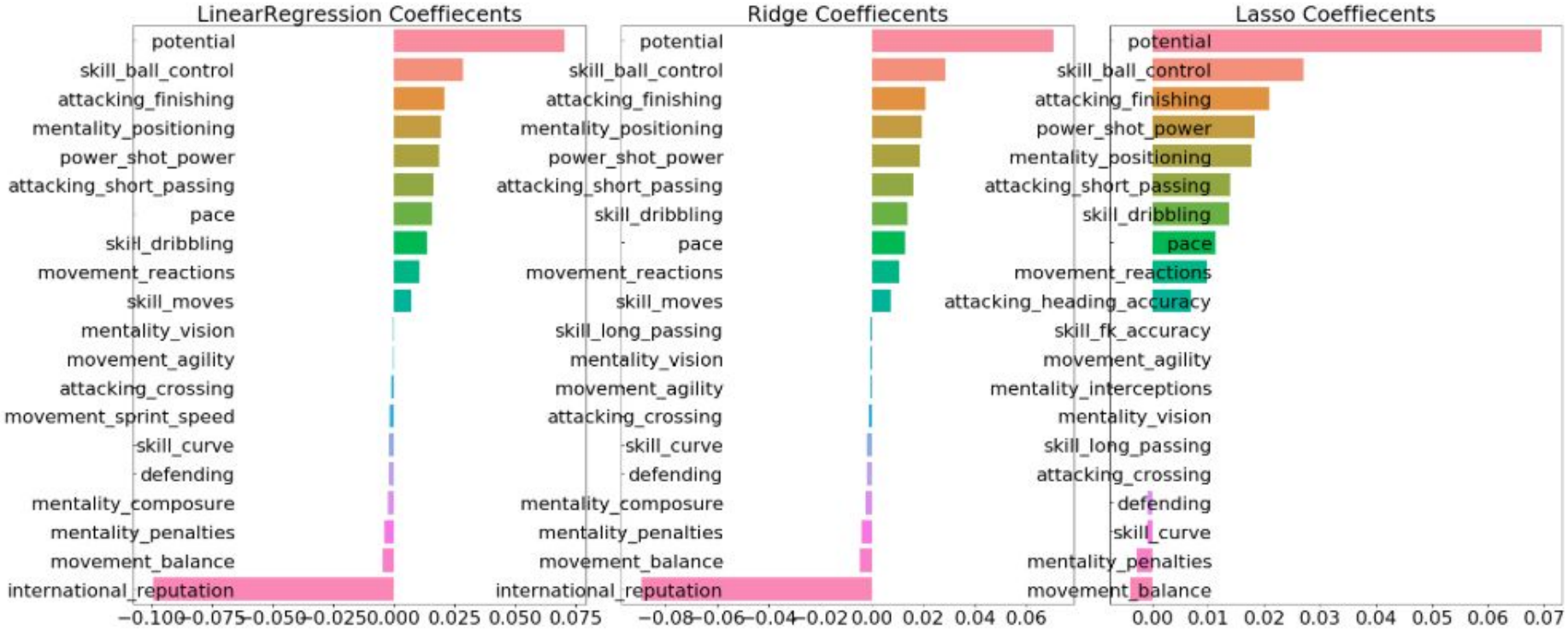
```
Ridge 5 CV 시 최적 평균 RMSE 값: 0.3515, 최적 alpha:{'alpha': 20}
Lasso 5 CV 시 최적 평균 RMSE 값: 0.3468, 최적 alpha:{'alpha': 0.05}
```

선형 회귀 모델 학습/예측 - 하이퍼 파라미터 튜닝

In [105]:
```
# 앞의 최적화 alpha값으로 학습데이터로 학습, 테스트 데이터로 예측 및 평가 수행.
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg = Ridge(alpha=20)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.05)
lasso_reg.fit(X_train, y_train)

# 모든 모델의 RMSE 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)

# 모든 모델의 회귀 계수 시각화
models = [lr_reg, ridge_reg, lasso_reg]
visualize_coefficient(models)
```

```
LinearRegression 로그 변환된 RMSLE: 0.345
Ridge 로그 변환된 RMSLE: 0.344
Lasso 로그 변환된 RMSLE: 0.344
```

선형 회귀 모델 -하이퍼 파라미터 튜닝 시각화



선형 회귀 모델 -skew, outlier 확인

In [106]:
```
from scipy.stats import skew

# object가 아닌 숫자형 피처의 컬럼 index 객체 추출.
features_index = X_data.dtypes[X_data.dtypes != 'object'].index
# house_df에 컬럼 index를 [ ]로 입력하면 해당하는 컬럼 데이터 셋 반환. apply lambda로 skew( )호출
skew_features = X_data[features_index].apply(lambda x : skew(x))
# skew 정도가 5 이상인 컬럼들만 추출.
skew_features_top = skew_features[skew_features > 1]
print(skew_features_top.sort_values(ascending=False))
```

```
international_reputation    2.958299
dtype: float64
```

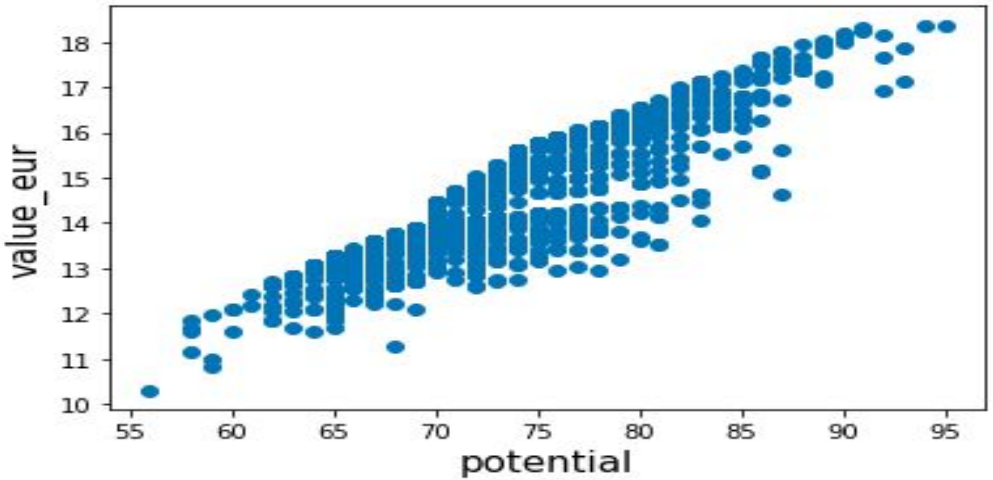
In [107]:
```
X_data[skew_features_top.index] = np.log1p(X_data[skew_features_top.index])
```

In [108]:
```
# Skew가 높은 피처들을 로그 변환 후 피처/타겟 데이터 셋 재생성

X_train, X_test, y_train, y_test = train_test_split(X_data, y_target_log, test_size=0.2, random_state=156)

# 피처들을 로그 변환 후 다시 최적 하이퍼 파라미터와 RMSE 출력
ridge_params = { 'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20] }
lasso_params = { 'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1,5, 10] }
best_ridge = get_best_params(ridge_reg, ridge_params)
best_lasso = get_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 CV 시 최적 평균 RMSE 값: 0.3504, 최적 alpha:{'alpha': 20}
Lasso 5 CV 시 최적 평균 RMSE 값: 0.3468, 최적 alpha:{'alpha': 0.05}
```

선형 회귀 모델 -skew, outlier 확인

In [109]:
```
plt.scatter(x = X_data['potential'], y = y_target_log)
plt.ylabel('value_eur', fontsize=15)
plt.xlabel('potential', fontsize=15)
plt.show()
```



----------이상치 없다고 판단. 재학습 안함.----------

회귀트리 - XGBM, LGBM

위에서 정제한 자료 이용합니다

회귀 트리 모델

회귀 트리 모델 학습/예측/평가

In [110]:
```
from xgboost import XGBRegressor

xgb_params = {'n_estimators':[1000]}
xgb_reg = XGBRegressor(n_estimators=1000, learning_rate=0.05,
                       colsample_bytree=0.5, subsample=0.8)
best_xgb = get_best_params(xgb_reg, xgb_params)
```

```
[20:53:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[20:53:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[20:53:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[20:53:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[20:53:30] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[20:53:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor 5 CV 시 최적 평균 RMSE 값: 0.6194, 최적 alpha:{'n_estimators': 1000}
```

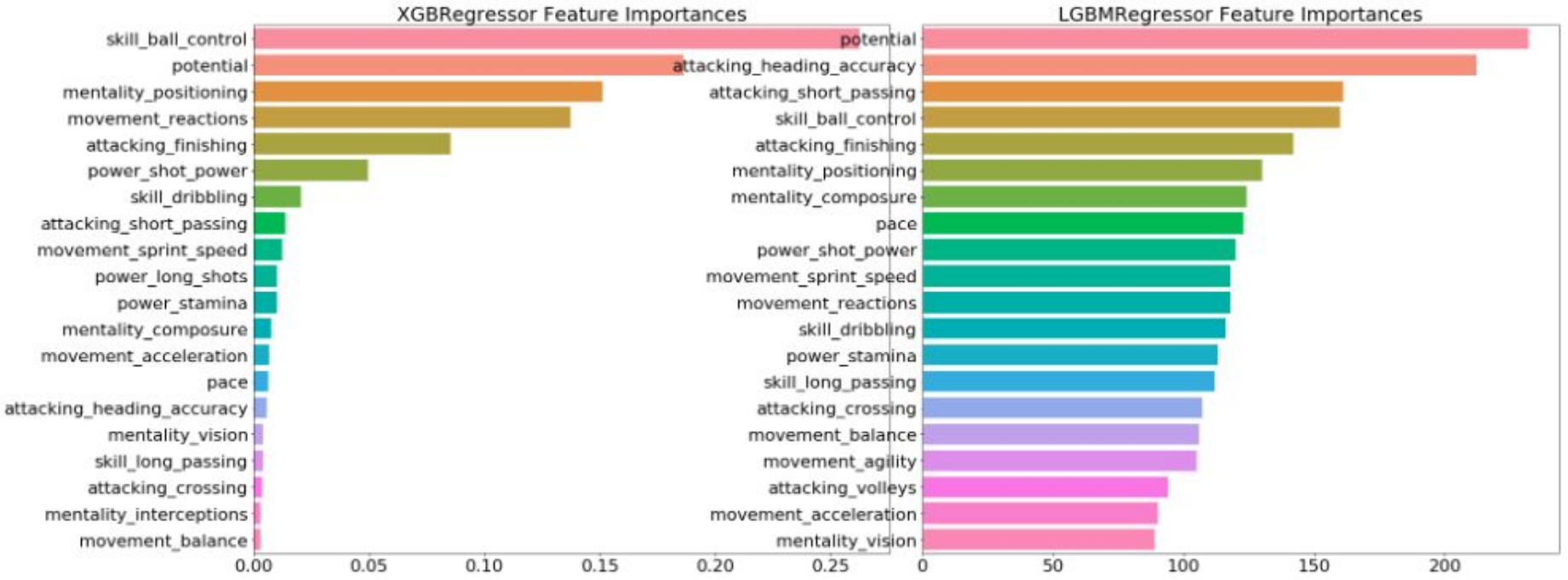
In [111]:
```
from lightgbm import LGBMRegressor

lgbm_params = {'n_estimators':[1000]}
lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                         subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)
best_lgbm = get_best_params(lgbm_reg, lgbm_params)
```

```
LGBMRegressor 5 CV 시 최적 평균 RMSE 값: 0.4747, 최적 alpha:{'n_estimators': 1000}
```

회귀 트리 모델 - 중요도 상위 20개 피처값으로 돌림

In [112]:
```
# 모델의 중요도 상위 20개의 피처명과 그때의 중요도값을 Series로 반환.
def get_top_features(model):
    ftr_importances_values = model.feature_importances_
    ftr_importances = pd.Series(ftr_importances_values, index=X_data.columns  )
    ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]
    return ftr_top20

def visualize_ftr_importances(models):
    # 2개 회귀 모델의 시각화를 위해 2개의 컬럼을 가지는 subplot 생성
    fig, axs = plt.subplots(figsize=(24,10),nrows=1, ncols=2)
    fig.tight_layout()
    # 입력인자로 받은 list객체인 models에서 차례로 model을 추출하여 피처 중요도 시각화.
    for i_num, model in enumerate(models):
        # 중요도 상위 20개의 피처명과 그때의 중요도값 추출
        ftr_top20 = get_top_features(model)
        axs[i_num].set_title(model.__class__.__name__+' Feature Importances', size=25)
        #font 크기 조정.
        for label in (axs[i_num].get_xticklabels() + axs[i_num].get_yticklabels()):
            label.set_fontsize(22)
        sns.barplot(x=ftr_top20.values, y=ftr_top20.index , ax=axs[i_num])

# 앞 예제에서 get_best_params( )가 반환한 GridSearchCV로 최적화된 모델의 피처 중요도 시각화
models = [best_xgb, best_lgbm]
visualize_ftr_importances(models)
```

회귀 트리 모델 - 중요도 상위 20개 피처값 시각화



회귀 트리 모델 - 중요도 상위 20개 피처값 릿지, 라쏘로 학습/예측

회귀 모델의 예측 결과 혼합을 통한 최종 예측

```
def get_rmse_pred(preds):
    for key in preds.keys():
        pred_value = preds[key]
        mse = mean_squared_error(y_test , pred_value)
        rmse = np.sqrt(mse)
        print('{0} 모델의 RMSLE: {1}'.format(key, rmse))

# 개별 모델의 학습
ridge_reg = Ridge(alpha=8)
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, y_train)
# 개별 모델 예측
ridge_pred = ridge_reg.predict(X_test)
lasso_pred = lasso_reg.predict(X_test)

# 개별 모델 예측값 혼합으로 최종 예측값 도출
pred = 0.4 * ridge_pred + 0.6 * lasso_pred
preds = {'최종 혼합': pred,
         'Ridge': ridge_pred,
         'Lasso': lasso_pred}
#최종 혼합 모델, 개별모델의 RMSE 값 출력
get_rmse_pred(preds)
```

```
최종 혼합 모델의 RMSLE: 0.31382140150481524
Ridge 모델의 RMSLE: 0.31386426821270885
Lasso 모델의 RMSLE: 0.31379798323730385
```

회귀 트리 모델 - 중요도 상위 20개 피처값 XGBM,LGBM 으로 학습/예측

```
xgb_reg = XGBRegressor(n_estimators=1000, learning_rate=0.05,
                       colsample_bytree=0.5, subsample=0.8)
lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                         subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)
xgb_reg.fit(X_train, y_train)
lgbm_reg.fit(X_train, y_train)
xgb_pred = xgb_reg.predict(X_test)
lgbm_pred = lgbm_reg.predict(X_test)

pred = 0.5 * xgb_pred + 0.5 * lgbm_pred
preds = {'최종 혼합': pred,
         'XGBM': xgb_pred,
         'LGBM': lgbm_pred}

get_rmse_pred(preds)
```

```
[20:53:50] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
최종 혼합 모델의 RMSLE: 0.2675261798595299
XGBM 모델의 RMSLE: 0.2768390815446319
LGBM 모델의 RMSLE: 0.2689035482617811
```

스태킹 앙상블 - 릿지, 라쏘, XGBM,LGBM

위에서 정제한 자료 이용합니다

스태킹 앙상블 모델

스태킹 앙상블 모델을 통한 회귀 예측

In [115]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error

# 개별 기반 모델에서 최종 메타 모델이 사용할 학습 및 테스트용 데이터를 생성하기 위한 함수.
def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds ):
    # 지정된 n_folds값으로 KFold 생성.
    kf = KFold(n_splits=n_folds, shuffle=False, random_state=0)
    #추후에 메타 모델이 사용할 학습 데이터 반환을 위한 넘파이 배열 초기화
    train_fold_pred = np.zeros((X_train_n.shape[0] ,1 ))
    test_pred = np.zeros((X_test_n.shape[0],n_folds))
    print(model.__class__.__name__ , ' model 시작 ')

    for folder_counter , (train_index, valid_index) in enumerate(kf.split(X_train_n)):
        #입력된 학습 데이터에서 기반 모델이 학습/예측할 폴드 데이터 셋 추출
        print('\t 폴드 세트: ',folder_counter,' 시작 ')
        X_tr = X_train_n[train_index]
        y_tr = y_train_n[train_index]
        X_te = X_train_n[valid_index]

        #폴드 세트 내부에서 다시 만들어진 학습 데이터로 기반 모델의 학습 수행.
        model.fit(X_tr , y_tr)
        #폴드 세트 내부에서 다시 만들어진 검증 데이터로 기반 모델 예측 후 데이터 저장.
        train_fold_pred[valid_index, :] = model.predict(X_te).reshape(-1,1)
        #입력된 원본 테스트 데이터를 폴드 세트내 학습된 기반 모델에서 예측 후 데이터 저장.
        test_pred[:, folder_counter] = model.predict(X_test_n)

    # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
    test_pred_mean = np.mean(test_pred, axis=1).reshape(-1,1)

    #train_fold_pred는 최종 메타 모델이 사용하는 학습 데이터, test_pred_mean은 테스트 데이터
    return train_fold_pred , test_pred_mean
```

스태킹 앙상블 모델

In [116]:
```
# get_stacking_base_datasets( )은 넘파이 ndarray를 인자로 사용하므로 DataFrame을 넘파이로 변환.
X_train_n = X_train.values
X_test_n = X_test.values
y_train_n = y_train.values

# 각 개별 기반(Base)모델이 생성한 학습용/테스트용 데이터 반환.
ridge_train, ridge_test = get_stacking_base_datasets(ridge_reg, X_train_n, y_train_n, X_test_n, 5)
lasso_train, lasso_test = get_stacking_base_datasets(lasso_reg, X_train_n, y_train_n, X_test_n, 5)
xgb_train, xgb_test = get_stacking_base_datasets(xgb_reg, X_train_n, y_train_n, X_test_n, 5)
lgbm_train, lgbm_test = get_stacking_base_datasets(lgbm_reg, X_train_n, y_train_n, X_test_n, 5)
```

In [117]:
```
# 개별 모델이 반환한 학습 및 테스트용 데이터 세트를 Stacking 형태로 결합.
Stack_final_X_train = np.concatenate((ridge_train, lasso_train,
                                      xgb_train, lgbm_train), axis=1)
Stack_final_X_test = np.concatenate((ridge_test, lasso_test,
                                     xgb_test, lgbm_test), axis=1)

# 최종 메타 모델은 라쏘 모델을 적용.
meta_model_lasso = Lasso(alpha=0.0005)

#기반 모델의 예측값을 기반으로 새롭게 만들어진 학습 및 테스트용 데이터로 예측하고 RMSE 측정.
meta_model_lasso.fit(Stack_final_X_train, y_train)
final = meta_model_lasso.predict(Stack_final_X_test)
mse = mean_squared_error(y_test , final)
rmse = np.sqrt(mse)
print('스태킹 회귀 모델의 최종 RMSLE 값은:', rmse)
```

```
스태킹 회귀 모델의 최종 RMSLE 값은: 0.2542875935026318
```