

Soccer Player Performance Prediction

2021 혁신_인공지능
채민석

목차

1. 데이터 출처 및 소개
2. 프로젝트 개요
3. Data Preprocessing
 - 1) 데이터 추출 및 정보확인
 - 2) dataset 통합
 - 3) column 정리
 - 4) 기타 작업
4. Modeling
5. Compiling, Training
6. Evaluating, Prediction
7. 프로젝트 성과 및 과제

1. 데이터 출처 및 소개

<https://www.kaggle.com/hugomathien/soccer>

The ultimate Soccer database for data analysis and machine learning

What you get:

- +25,000 matches
- +10,000 players
- 11 European Countries with their lead championship
- Seasons 2008 to 2016
- Players and Teams' attributes* sourced from EA Sports' FIFA video game series, including the weekly updates
- Team line up with squad formation (X, Y coordinates)
- Betting odds from up to 10 providers
- Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches

Data Explorer

298.59 MB

- ▼ database.sqlite
 - Country
 - League
 - Match
 - Player
 - Player_Attributes
 - Team
 - Team_Attributes

Summary

- ▶ 1 file
- ▶ 199 columns

< database.sqlite (298.59 MB)



Table	Total Rows	Total Columns
Country	11	2
League	11	3
Match	25979	115
Player	11060	7
Player_Attributes	183978	42
Team	299	5
Team_Attributes	1458	25

2. 프로젝트 개요

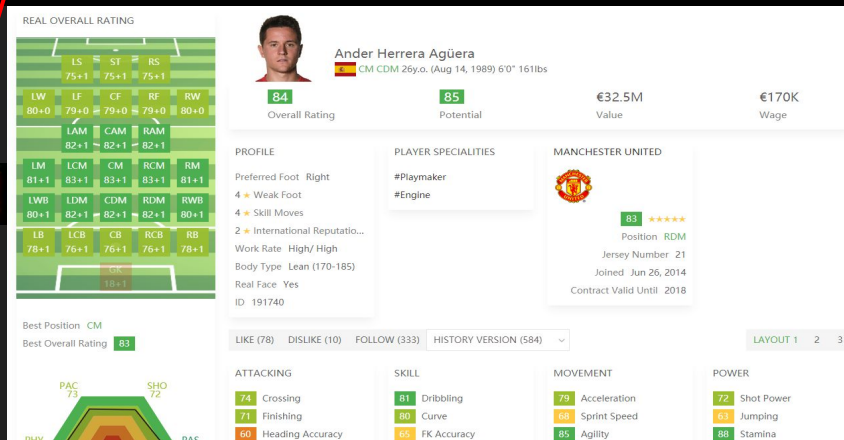
Player 관련 데이터 수집

```
"for columns in player_attributes.columns:  
    print(columns)"
```

```
# [id player_fifa_api_id player_api_id date overall_rating potential preferred_foot attacking_work_rate  
# defensive_work_rate crossing finishing heading_accuracy short_passing volleys dribbling curve  
# free_kick_accuracy long_passing ball_control acceleration sprint_speed agility reactions balance  
# shot_power jumping stamina strength long_shots aggression interceptions positioning vision penalties  
# marking standing_tackle sliding_tackle gk_diving gk_handling gk_kicking gk_positioning gk_reflexes]
```

Player Performance 훈련 및 예측

```
y = player_merged['overall_rating']
```



3. Data Preprocessing

1) 데이터 추출 및 정보확인

이름	수정한 날짜	유형	크기
Ander_Herrera.csv	2021-08-02 오후 3:19	CSV 파일	1KB
database.sqlite	2019-09-19 오전 9:23	SQLITE 파일	305,752KB

- database.sqlite
 - Country
 - League
 - Match
 - Player
 - Player_Attributes
 - Team
 - Team_Attributes

#1. Data Preprocessing

#1-1. 데이터 추출 및 정보확인

1) read_sql

[참고] https://www.w3schools.com/sql/sql_select.asp

```
path = sqlite3.connect('../_data/soccer_prediction/database.sqlite') # 데이터베이스 경로설정
player = pd.read_sql_query('SELECT * FROM Player', path) # SELECT 구문 이용 필요 데이터셋 추출 (Player, Player_Attributes)
player_attributes = pd.read_sql_query('SELECT * FROM Player_Attributes', path)
```

2) null 값 dropping

```
player = player.dropna(axis=0)
player_attributes = player_attributes.dropna(axis=0)
```

3. Data Preprocessing

1) 데이터 추출 및 정보확인

💡 # 3) column 명 확인

```
"""for columns in player.columns:
    print(columns)"""
```

```
# [id player_api_id player_name player_fifa_api_id birthday height weight]
```

```
"""for columns in player_attributes.columns:
    print(columns)"""
```

```
# [id player_fifa_api_id player_api_id date over_away_goals
# defensive_work_rate crossing finishing heading_accuracy
# free_kick_accuracy long_passing ball_control
# shot_power jumping stamina strength long_shot_accuracy
# marking standing_tackle sliding_tackle gk_diving]
```

4) 데이터 shape 확인 및 초기 5행 시범출력

```
"""print(player.shape, player.head()) # (11060, 7) """
```

```
# id player_api_id player_name ... birthday height weight
# 0 1 505942 Aaron Appindangoye ... 1992-02-29 00:00:00 182.88 187
# 1 2 155782 Aaron Cresswell ... 1989-12-15 00:00:00 170.18 146
# 2 3 162549 Aaron Doran ... 1991-05-13 00:00:00 170.18 163
# 3 4 30572 Aaron Galindo ... 1982-05-08 00:00:00 182.88 198
# 4 5 23780 Aaron Hughes ... 1979-11-08 00:00:00 182.88 154
# [11060 rows x 7 columns]
```

```
"""print(player_attributes.shape, player_attributes.head()) # (183978, 42)"""
```

```
# id player_fifa_api_id player_api_id ... gk_kicking gk_positioning gk
# 0 1 218353 505942 ... 10.0 8.0 8.0
# 1 2 218353 505942 ... 10.0 8.0 8.0
# 2 3 218353 505942 ... 10.0 8.0 8.0
# 3 4 218353 505942 ... 9.0 7.0 7.0
# 4 5 218353 505942 ... 9.0 7.0 7.0
# [183978 rows x 42 columns]
```


3. Data Preprocessing

2) dataset 통합

#1-2. dataset 통합 (player + player_attributes)

player dataset: 11060명의 선수 기본 정보, player_attributes dataset: 시즌별로(date) 각 선수 세부 스탯 정보
player_attributes dataset 에서 동일 player_api_id 데이터 = 동일 선수 시즌별 데이터 -> 선수별 스탯 단일화 필요
평균값으로 단일화 시킨후 player dataset 에 추가 (하나의 dataset으로 통합)
평균값 계산을 위해 일부 데이터 수치화 필요 (preferred_foot, attacking_work_rate, defensive_work_rate)

4) 데이터 shape 확인 및 초기 5행 시범출력

```
"print(player.shape, player.head()) # (11060, 7) "
```

```
# id player_api_id player_name ... birthday height weight  
# 0 1 505942 Aaron Appindangoye ... 1992-02-29 00:00:00 182.88 187  
# 1 2 155782 Aaron Cresswell ... 1989-12-15 00:00:00 170.18 146  
# 2 3 162549 Aaron Doran ... 1991-05-13 00:00:00 170.18 163  
# 3 4 30572 Aaron Galindo ... 1982-05-08 00:00:00 182.88 198  
# 4 5 23780 Aaron Hughes ... 1979-11-08 00:00:00 182.88 154
```

```
# [11060 rows x 7 columns]
```

```
"print(player_attributes.shape, player_attributes.head()) # (183978, 42)"
```

```
# id player_fifa_api_id player_api_id ... gk_kicking gk_positioning gk  
# 0 1 218353 505942 ... 10.0 8.0 8.0  
# 1 2 218353 505942 ... 10.0 8.0 8.0  
# 2 3 218353 505942 ... 10.0 8.0 8.0  
# 3 4 218353 505942 ... 9.0 7.0 7.0  
# 4 5 218353 505942 ... 9.0 7.0 7.0
```

```
# [183978 rows x 42 columns]
```

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attacking_work_rate
0	1	218353	505942	2016-02-18 00:00:00	67.0	71.0	right	medium
1	2	218353	505942	2015-11-19 00:00:00	67.0	71.0	right	medium
2	3	218353	505942	2015-09-21 00:00:00	62.0	66.0	right	medium
3	4	218353	505942	2015-03-20 00:00:00	61.0	65.0	right	medium
4	5	218353	505942	2007-02-22 00:00:00	61.0	65.0	right	medium
...
183973	183974	102359	39902	2009-08-30 00:00:00	83.0	85.0	right	medium
183974	183975	102359	39902	2009-02-22 00:00:00	78.0	80.0	right	medium
183975	183976	102359	39902	2008-08-30 00:00:00	77.0	80.0	right	medium
183976	183977	102359	39902	2007-08-30 00:00:00	78.0	81.0	right	medium

3. Data Preprocessing

2) dataset 통합

```
# 2) binary encoding ( preferred_foot )  
player_attributes['preferred_foot'] = player_attributes['preferred_foot'].replace({'left':0, 'right':1}) # 주발  
print(player_attributes['preferred_foot'].head())
```

```
# 0  1  
# 1  1  
# 2  1  
# 3  1  
# 4  1
```

preferred_foot	attacking_work_rate	defensive_work_rate
right	medium	medium

```
# 3) onehot encoding ( attacking_work_rate, defensive_work_rate )  
# 'Series' object has no attribute 'to_categorical' : pandas_series는 to_categorical 또는 onehotencoder 적용불가
```


3. Data Preprocessing

2) dataset 통합

```
# 3-1) get_dummies
# [참고1] https://stackoverflow.com/questions/58101126/using-scikit-learn-onehotencoder-with-a-pandas-dataframe
# [참고2] https://pandas.pydata.org/pandas-docs/version/0.17.0/generated/pandas.get\_dummies.html

player_attributes['attacking_work_rate'] = pd.get_dummies(
    player_attributes['attacking_work_rate'],
    prefix=['attacking_work_rate'],
    columns=['attacking_work_rate'], drop_first=True)

print(player_attributes['attacking_work_rate'].head(10))

# 0  0
# 1  0
# 2  0
# 3  0
# 4  0
# 5  1
# 6  1
# 7  1
# 8  1
# 9  1
```

3. Data Preprocessing

2) dataset 통합

```
# 3-2) LabelBinarizer
# [참고] https://www.python2.net/questions-1176634.htm
encoder = LabelBinarizer()
encoder.fit(player_attributes['attacking_work_rate'])
transformed = encoder.transform(player_attributes['attacking_work_rate'])
player_attributes['attacking_work_rate'] = pd.DataFrame(transformed)
"""print(player_attributes['attacking_work_rate'].head(10))"""
# 0  0.0
# 1  0.0
# 2  0.0
# 3  0.0
# 4  0.0
# 5  1.0
# 6  1.0
# 7  1.0
# 8  1.0
# 9  1.0
player_attributes['defensive_work_rate'] = pd.get_dummies( # defensive_work_rate 상동
    player_attributes['defensive_work_rate'],
    prefix=['defensive_work_rate'],
    columns=['defensive_work_rate'], drop_first=True)
```

3. Data Preprocessing

2) dataset 통합

```
# 4) merging
# [참고1] https://data-make.tistory.com/139
# [참고2] https://data-newbie.tistory.com/133
# [참고3] https://nittaku.tistory.com/121

player_attributes = player_attributes.groupby(player_attributes['player_api_id']).mean() # 평균값으로 선수별 스탯 단일화
print(player_attributes.head())

#           overall_rating potential ... gk_positioning gk_reflexes
# player_api_id ...
# 2625      60.142857 61.142857 ...    10.357143  10.428571
# 2752      69.380952 70.380952 ...     9.095238  15.095238
# 2768      69.285714 70.571429 ...    15.142857  12.095238
# 2770      71.133333 73.533333 ...    16.333333  17.000000
# 2790      70.200000 75.800000 ...    16.600000  17.400000
# [5 rows x 38 columns]

player_merged = player.merge(player_attributes, on='player_api_id') # player_api_id 기준으로 두 dataset 통합 -> 하나의 df 생성(player_merged)
print(player_merged.head())

# id player_api_id  player_name ... gk_kicking gk_positioning gk_reflexes
# 0  1      505942  Aaron Appindangoye ...   9.600000   7.600000   7.600000
# 1  2      155782   Aaron Cresswell ...  14.242424  10.363636  12.909091
# 2  3      162549   Aaron Doran ...   17.730769  10.115385  13.500000
# 3  4       30572   Aaron Galindo ...  22.869565  11.173913  10.173913
# 4  5       23780   Aaron Hughes ...   24.920000  12.840000  11.920000
# [5 rows x 45 columns]
```

3. Data Preprocessing

2) dataset 통합

```
"""print(player_merged.info())"""
# Int64Index: 10410 entries, 0 to 10409
# Data columns (total 45 columns):
#  # Column          Non-Null Count  Dtype
# ---  -
# 0 id                10410 non-null int64
# 1 player_api_id     10410 non-null int64
# 2 player_name       10410 non-null object
# 3 player_fifa_api_id 10410 non-null int64
# 4 birthday          10410 non-null object
# 5 height             10410 non-null float64
# 6 weight             10410 non-null int64
# 7 overall_rating     10410 non-null float64
# 8 potential          10410 non-null float64
# 9 preferred_foot     10410 non-null float64
# 10 attacking_work_rate 10410 non-null float64
# 11 defensive_work_rate 10410 non-null float64
# 12 crossing           10410 non-null float64
# 13 finishing          10410 non-null float64
# 14 heading_accuracy  10410 non-null float64
# 15 short_passing      10410 non-null float64
# 16 volleys            10410 non-null float64
# 17 dribbling          10410 non-null float64
# 18 curve              10410 non-null float64
# 19 free_kick_accuracy 10410 non-null float64
# 20 long_passing       10410 non-null float64
# 21 ball_control       10410 non-null float64
# 22 acceleration       10410 non-null float64
# 23 sprint_speed       10410 non-null float64
# 24 agility            10410 non-null float64
# 25 reactions          10410 non-null float64
# 26 balance            10410 non-null float64
# 27 shot_power         10410 non-null float64
# 28 jumping            10410 non-null float64
# 29 stamina            10410 non-null float64
# 30 strength           10410 non-null float64
# 31 long_shots         10410 non-null float64
# 32 aggression         10410 non-null float64
# 33 interceptions      10410 non-null float64
# 34 positioning        10410 non-null float64
# 35 vision             10410 non-null float64
# 36 penalties          10410 non-null float64
# 37 marking            10410 non-null float64
# 38 standing_tackle    10410 non-null float64
# 39 sliding_tackle     10410 non-null float64
# 40 gk_diving          10410 non-null float64
# 41 gk_handling        10410 non-null float64
# 42 gk_kicking         10410 non-null float64
# 43 gk_positioning     10410 non-null float64
# 44 gk_reflexes        10410 non-null float64
# dtypes: float64(39), int64(4), object(2)
```

player

player_attributes

3. Data Preprocessing

3) column 정리

#1-3. column 정리

1) birthday column 으로부터 birth_year column 생성 (age)

[참고] <https://hiiio.tistory.com/30>

```
"""print(player_merged.info())"""
```

```
## Column Non-Null Count Dtype
```

```
# 4 birthday non-null 10410 object
```

```
player_merged['birthday'] = pd.to_datetime(player_merged['birthday']) # apply() 적용을 위해 dtype 변환 object -> datetime64
```

```
"""print(player_merged.info())"""
```

```
## Column Non-Null Count Dtype
```

```
# 4 birthday non-null 10410 datetime64[ns]
```

```
player_merged['birthday'] = player_merged['birthday'].apply(lambda x:x.year) # 출생년도만 추출
```

```
player_merged.rename(columns={'birthday': 'birthyear'}, inplace=True) # column명 변경
```

```
"""print(player_merged['birthyear'].head())"""
```

```
# 0 1992
```

```
# 1 1989
```

```
# 2 1991
```

```
# 3 1982
```

```
# 4 1979
```

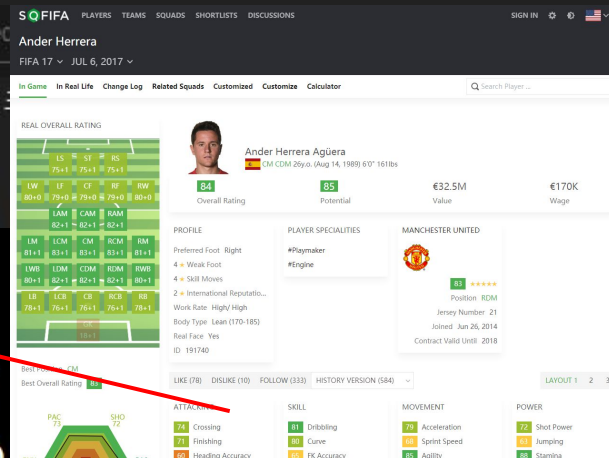
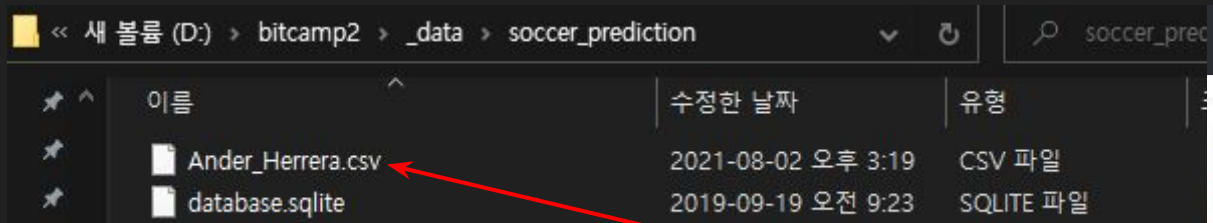
2) 불필요한 column drop (id, player_api_id, player_name, player_fifa_api_id)

```
player_merged = player_merged.drop(['id', 'player_api_id', 'player_name', 'player_fifa_api_id'], axis=1)
```

```
birthday height weight
... 1992-02-29 00:00:00 182.88 187
1989-12-15 00:00:00 170.18 146
1991-05-13 00:00:00 170.18 163
1982-05-08 00:00:00 182.88 198
1979-11-08 00:00:00 182.88 154
```

3. Data Preprocessing

4) 기타 작업



#1-4. 기타 작업

1) 예측할 데이터 불러오기

[출처] <https://sofifa.com/player/191740/ander-herrera-aguera/?attr=classic>

엑셀로 수치입력후 csv파일로 변환

```
x_pred = pd.read_csv('../_data/soccer_prediction/Ander_Herrera.csv', header=0)
```

```
"""print(x_pred.shape)"""
```

```
# (1, 41)
```

2) x, y 분리

```
y = player_merged['overall_rating'] # target value (=player performance)
```

```
x = player_merged.drop(['overall_rating'], axis=1)
```

```
x_pred = x_pred.drop(['overall_rating'], axis=1)
```

```
"""print(x.shape, x_pred.shape)"""
```

```
# (10410, 40) (1, 40)
```


3. Data Preprocessing

4) 기타 작업

```
# # Column
# ---
# 0 id 10
# 1 player_api_id
# 2 player_name
# 3 player_fifa_api
# 4 birthday
# 5 height
# 6 weight

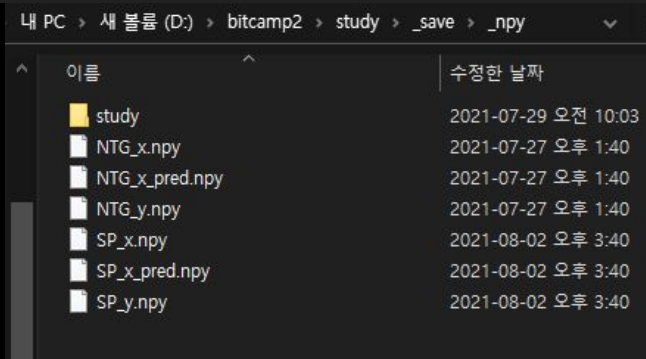
# 3) scaling
# 수치차이로 인한 개별적용
# [참고] https://www.python2.net/questions-607283.htm
x1 = x.iloc[:, 0].values # birthyear, 1900~2100, reshaping을 위한 np array converting
x1_pred = x_pred.iloc[:, 0].values
x2 = x.iloc[:, 1:3] # height, wieght, 150~300
x2_pred = x_pred.iloc[:, 1:3]
x3 = x.iloc[:, 3:] # x1, x2 제외 나머지, 0~100
x3_pred = x_pred.iloc[:, 3:]
scaler = StandardScaler()
scaler.fit(x1.reshape(-1,1)) # 스케일러 적용을 위한 2D reshaping
x1 = scaler.transform(x1.reshape(-1,1))
x1_pred = scaler.transform(x1_pred.reshape(-1,1)) # x_pred에 동일 스케일 적용
scaler.fit(x2)
x2 = scaler.transform(x2)
x2_pred = scaler.transform(x2_pred)
scaler.fit(x3)
x3 = scaler.transform(x3)
x3_pred = scaler.transform(x3_pred)
x = np.concatenate((x1, x2, x3), axis=1) # x1, x2, x3 병합
x_pred = np.concatenate((x1_pred, x2_pred, x3_pred), axis=1)
"""print(x.shape, x_pred.shape)"""
# (10410, 40) (1, 40)
```

3. Data Preprocessing

4) 기타 작업

```
# 4) 전처리 데이터 npy저장 및 로드
# np.save('./_save/_np/SP_x.npy', arr=x)
# np.save('./_save/_np/SP_y.npy', arr=y)
# np.save('./_save/_np/SP_x_pred.npy', arr=x_pred)
x = np.load('./_save/_np/SP_x.npy')
y = np.load('./_save/_np/SP_y.npy')
x_pred = np.load('./_save/_np/SP_x_pred.npy')
```

```
# 5) train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, shuffle=True, random_state=21)
```



이름	수정된 날짜
study	2021-07-29 오전 10:03
NTG_x.npy	2021-07-27 오후 1:40
NTG_x_pred.npy	2021-07-27 오후 1:40
NTG_y.npy	2021-07-27 오후 1:40
SP_x.npy	2021-08-02 오후 3:40
SP_x_pred.npy	2021-08-02 오후 3:40
SP_y.npy	2021-08-02 오후 3:40

4. Modeling

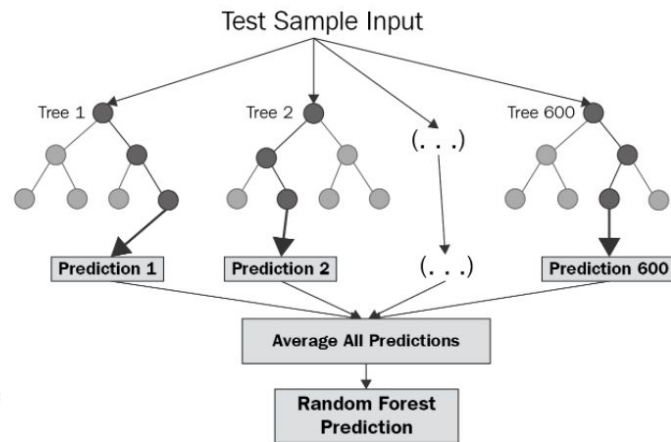
```
#2. Modeling    "print(x.shape, x_pred.shape)"
input = Input((40,))  # (10410, 40) (1, 40)
d = Dense(512, activation='relu')(input)
d = Dense(256, activation='relu')(d)
output = Dense(1, activation='relu')(d)
model = Model(inputs=input, outputs=output)
# model = RandomForestRegressor()
```

A random forest regressor.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

[출처]

<https://post.naver.com/viewer/postView.nhn?volumeNo=28037302&memberNo=18071586>
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>



장점 및 특징

가장 정확한 학습 알고리즘 중 하나다. 많은 데이터셋에 대해서 매우 정확한 분류자(classifier)를 만든다.

- 대량의 데이터베이스에서 효과적으로 작동한다.
- 수천 개의 입력 변수를 처리할 수 있다.
- 분류할 때 중요한 변수가 무엇인지 예측할 수 있다.
- 포레스트(숲)을 만들면서 일반화 오류의 편향되지 않은 내부 예측치를 생성한다.
- 예측치를 예측하는 효과적인 방법을 제공해서 높은 비율의 데이터가 결여되도 정확도를 유지한다.

5. Compiling, Training

#3. Compiling, Training

```
model.compile(loss='mse', optimizer='adam')
```

```
date = datetime.datetime.now()
```

```
date_time = date.strftime('%m%d_%H%M')
```

```
path = './_save/_mcp/'
```

```
info = '{epoch:02d}_{val_loss:.4f}'
```

```
filepath = ".join([path, 'SP', '_', date_time, '_', info, '.hdf5'])
```

```
cp = ModelCheckpoint(monitor='val_loss', save_best_only=True, mode='auto', verbose=1, filepath=filepath)
```

```
es = EarlyStopping(monitor='val_loss', restore_best_weights=False, mode='auto', verbose=1, patience=10)
```

```
start_time = time.time()
```

```
model.fit(x_train, y_train, epochs=100, batch_size=8, verbose=1, validation_split=0.1, callbacks=[es, cp])
```

```
# model.fit(x_train, y_train) # RandomForestRegressor
```

```
end_time = time.time() - start_time
```

6. Evaluating, Prediction

#4. Evaluating, Prediction

```
loss = model.evaluate(x_test, y_test)
y_predict = model.predict(x_test)
r2 = r2_score(y_test, y_predict)
prediction = model.predict(x_pred)


print('loss = ', loss)
print('r2 score =', r2)
print('Performance Prediction = ', prediction)
```

'''


```
loss = 0.9929283857345581
r2 score = 0.9737266283123291
Performance Prediction = [[84.824684]]
```

```
RandomForestRegressor
r2 score = 0.9451363556335565
Performance Prediction = [82.08301616]
```

'''



Ander Herrera Agüera

 CM CDM 26y.o. (Aug 14, 1989) 6'0" 161lbs

84	85
Overall Rating	Potential

A red box highlights the 'Overall Rating' of 84, with a red arrow pointing from the 'Performance Prediction' value of 84.824684 in the code block on the left.

7. 프로젝트 성과 및 과제


전처리 스킬 학습

- `pd.read_sql_query(SELECT * FROM)`
- `pd.get_dummies()`
- `LabelBinarizer()`
- `df.groupby().mean()`
- `df.merge(on='column')`
- `pd.to_datetime()`
- `df.apply(lambda x:)`

tensorboard 적용 → 훈련과정 시각화, `loss`, `val_loss` 추적 및 분석

RandomForestRegressor 심화 응용

https://github.com/MinseokCHAE



MinseokCHAE

[Follow](#)

5 followers · 1 following · ☆ 0

[Block or Report](#)

[Overview](#) [Repositories](#) 3 [Projects](#) [Packages](#)

Popular repositories


[Bitcamp](#)

Python

[Bitcamp2_new](#)

Python

62 contributions in the last year



Learn how we count contributions

main [Bitcamp2_new / project / soccer_prediction /](#)

MinseokCHAE update_0803_17:42

...

soccer_prediction_final_01.pdf	update_0803_17:42
soccer_prediction_final_01.pptx	update_0803_17:42
soccer_prediction_intro_01.pdf	update_0803_17:42
soccer_prediction_intro_01.pptx	update_0803_17:42
soccer_prediction_npy_load.py	update_0803_17:42
soccer_prediction_npy_save.py	update_0802_18:15