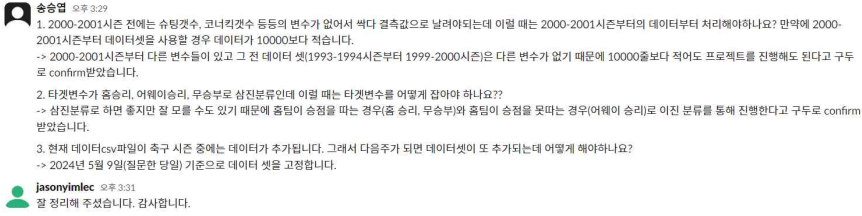
빅데이터 프로그래밍
팀 프로젝트

1991064 김재영
1991003 송승엽

1. 연구 주제 및 데이터 선택 사유

연구주제: 잉글랜드 프리미어 리그(EPL)에서 홈팀이 승리를 할 때 주요 요인은 무엇인가?



송승엽 오후 3:29

1. 2000-2001시즌 전에는 슈팅갯수, 코너킥갯수 등등의 변수가 없어서 싹다 결측값으로 날려야되는데 이럴 때는 2000-2001시즌부터의 데이터부터 처리해야하나요? 만약에 2000-2001시즌부터 데이터셋을 사용할 경우 데이터가 10000보다 적습니다.
-> 2000-2001시즌부터 다른 변수들이 있고 그 전 데이터 셋(1993-1994시즌부터 1999-2000시즌)은 다른 변수가 없기 때문에 10000줄보다 적어도 프로젝트를 진행해도 된다고 구두로 confirm받았습니다.

2. 타겟변수가 홈승리, 어웨이승리, 무승부로 삼진분류인데 이럴 때는 타겟변수를 어떻게 잡아야 하나요??
-> 삼진분류로 하면 좋지만 잘 모를 수도 있기 때문에 홈팀이 승점을 따는 경우(홈 승리, 무승부)와 홈팀이 승점을 못따는 경우(어웨이 승리)로 이진 분류를 통해 진행한다고 구두로 confirm 받았습니다.

3. 현재 데이터csv파일이 축구 시즌 중에는 데이터가 추가됩니다. 그래서 다음주가 되면 데이터셋이 또 추가되는데 어떻게 해야하나요?
-> 2024년 5월 9일(질문한 당일) 기준으로 데이터 셋을 고정합니다.

jasonyimlec 오후 3:31

잘 정리해 주셨습니다. 감사합니다.

데이터는 역대 잉글랜드 프리미어 리그(EPL) 경기 결과 데이터를 사용한다. (24.05.09 기준 데이터를 사용)
데이터 URL : https://www.kaggle.com/datasets/ajaxianazarenka/premier-league

2. ID 변수 설정 및 타겟 변수 설정

	Season	Date	Time	HomeTeam	AwayTeam	FullTimeHomeTeamGoals	FullTimeAwayTeamGoals	FullTimeResult
11752	2023-2024	04/05/2024	05:30:00 p. m.	Man City	Wolves	5	1	H
11753	2023-2024	05/05/2024	02:00:00 p. m.	Brighton	Aston Villa	1	0	H
11754	2023-2024	05/05/2024	02:00:00 p. m.	Chelsea	West Ham	5	0	H
11755	2023-2024	05/05/2024	04:30:00 p. m.	Liverpool	Tottenham	4	2	H
11756	2023-2024	06/05/2024	08:00:00 p. m.	Crystal Palace	Man United	4	0	H

ID 변수는 Season, Date, Home Team, Away Team을 통해서 분류할 수 있다. 왜냐하면 한 시즌에 한 날짜에는 여러 게임을 많이 할 수 있지만 Home Team장소에서 한 경기만 진행하기 때문에 따로 ID 변수를 설정할 필요가 없다.

```
c2 = df['FullTimeResult'] == 'H'
c1 = df['FullTimeResult'] == 'D'
c0 = df['FullTimeResult'] == 'A'

df.loc[c2, "FullTimeResult_B"] = 1
df.loc[c1, "FullTimeResult_B"] = 0
df.loc[c0, "FullTimeResult_B"] = 0
```

타겟 변수는 Full Time Result이지만 이진 분류를 통해 1과 0으로 분류하였습니다. 1은 Home Team이 이기는 경우이고, 0은 비기거나 Away Team이 이기는 경우(Home Team이 진 경우)입니다.

```
df['FullTimeResult_B'].value_counts(dropna=False)
```

```
FullTimeResult_B
0.0    6356
1.0    5401
Name: count, dtype: int64
```

```
df['FullTimeResult_B'].value_counts(dropna=False, normalize=True)
```

```
FullTimeResult_B
0.0    0.540614
1.0    0.459386
Name: proportion, dtype: float64
```

타겟 변수인 FullTimeResult_B은 54:46 비율로 나뉩니다.

3. 결측값 50% 초과 변수 제거

```
df.isna().any()
```

```
Season                  False
Date                    False
Time                     True
HomeTeam                False
AwayTeam                False
FullTimeHomeTeamGoals   False
FullTimeAwayTeamGoals   False
FullTimeResult          False
HalfTimeHomeTeamGoals    True
HalfTimeAwayTeamGoals    True
HalfTimeResult           True
Referee                  True
HomeTeamShots            True
AwayTeamShots            True
HomeTeamShotsOnTarget    True
AwayTeamShotsOnTarget    True
HomeTeamCorners          True
AwayTeamCorners          True
HomeTeamFouls            True
AwayTeamFouls            True
HomeTeamYellowCards      True
AwayTeamYellowCards      True
HomeTeamRedCards         True
AwayTeamRedCards         True
B365HomeTeam             True
B365Draw                 True
B365AwayTeam             True
B365Over2.5Goals         True
B365Under2.5Goals        True
MarketMaxHomeTeam        True
MarketMaxDraw            True
MarketMaxAwayTeam        True
MarketAvgHomeTeam        True
MarketAvgDraw            True
MarketAvgAwayTeam        True
MarketMaxOver2.5Goals    True
MarketMaxUnder2.5Goals   True
MarketAvgOver2.5Goals    True
MarketAvgUnder2.5Goals   True
```

```
df['HalfTimeHomeTeamGoals'].isnull().mean()
```

```
0.06464234073318023
```

```
df['HomeTeamRedCards'].isnull().mean()
```

```
0.22624819256613082
```

```
df['Time'].isnull().mean()
```

```
0.840350429531343
```

```
df['MarketMaxOver2.5Goals'].isnull().mean()
```

```
0.840350429531343
```

```
df['MarketMaxHomeTeam'].isnull().mean()
```

```
0.840350429531343
```

결측값을 가지는 변수들이 True로 있는데 이 중 50%를 초과하는 변수는 B365HomeTeam부터 MarketAvgUnder2.5Goals까지의 모든 변수들이다.
이 변수들은 배팅과 관련된 변수로 2000년도 전에는 배팅 사이트의 디지털화가 활성화되지 않아서 모두 84%(50%초과)로 결측값이 있었습니다.
따라서 저희 조는 모든 배팅과 관련된 변수를 제거하였습니다.
또한 Time도 제거하였는데 이는 같은 시즌(Season), 같은 날짜(Date)에 경기가 이뤄지지만 Home Team과 Away Team으로 구별할 수 있고 결측값도 50% 넘기 때문에 제거하였습니다.

```
#   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
0   Season                  9097 non-null   object
1   Date                    9097 non-null   object
2   HomeTeam                9097 non-null   object
3   AwayTeam                9097 non-null   object
4   FullTimeHomeTeamGoals   9097 non-null   int64
5   FullTimeAwayTeamGoals   9097 non-null   int64
6   HalfTimeHomeTeamGoals   9097 non-null   float64
7   HalfTimeAwayTeamGoals   9097 non-null   float64
8   Referee                 9097 non-null   object
9   HomeTeamShots           9097 non-null   float64
10  AwayTeamShots           9097 non-null   float64
11  HomeTeamShotsOnTarget   9097 non-null   float64
12  AwayTeamShotsOnTarget   9097 non-null   float64
13  HomeTeamCorners         9097 non-null   float64
14  AwayTeamCorners         9097 non-null   float64
15  HomeTeamFouls           9097 non-null   float64
16  AwayTeamFouls           9097 non-null   float64
17  HomeTeamYellowCards     9097 non-null   float64
18  AwayTeamYellowCards     9097 non-null   float64
19  HomeTeamRedCards        9097 non-null   float64
20  AwayTeamRedCards        9097 non-null   float64
21  FullTimeResult_B        9097 non-null   float64
22  HalfTimeResult_B        9097 non-null   float64
```

위에는 결측값 50% 초과한 변수를 제거한 데이터 셋 입니다.

4. 기타 변수 데이터 처리 - 1

```
c2 = df['HalfTimeResult'] == 'H'
c1 = df['HalfTimeResult'] == 'D'
c0 = df['HalfTimeResult'] == 'A'

df.loc[c2, "HalfTimeResult_B"] = 1
df.loc[c1, "HalfTimeResult_B"] = 0
df.loc[c0, "HalfTimeResult_B"] = 0
```

타겟 변수와 유사한 Half Time Result(전반전 매칭 결과)도 홈 팀이 이긴 경우와 비기거나 어웨이 팀이 이긴 경우(홈 팀이 진 경우)로 이진 분류 하였습니다.

```
df = df.rename(columns={'AwayTeamReadCards' : 'AwayTeamRedCards'})
```

AwayTeamReadCards의 오타를 AwayTeamRedCards로 수정하였습니다.

HomeTeamFouls	0.44
AwayTeamFouls	0.41
HomeTeamYellowCards	0.76
AwayTeamYellowCards	0.63
HomeTeamRedCards	4.16
AwayTeamRedCards	3.34

HomeTeamFouls	0.45
AwayTeamFouls	0.26
HomeTeamYellowCards	0.38
AwayTeamYellowCards	0.29
HomeTeamRedCards	18.11
AwayTeamRedCards	10.71

HomeTeam과 AwayTeam의 파울, 카드 수를 처음에는 구간 변수로 선정하였다. 왜냐하면 다른 구간 변수들처럼 숫자화 되어있기 때문이다. 하지만 왼쪽의 skew(왜도)와 오른쪽의 kurtosis(첨도)를 보니깐 RedCards는 각각 3과 10을 초과해서 정규분포를 따르지 않는다고 생각하였다. 그래서 RedCards와 관련된 변수들인 Fouls, YellowCards와 함께 모두 범주형 변수로 분류하였다.
=> 왜도, 첨도가 기준치보다 초과했지만 로그 변환을 하지 않고 범주형 변수로 분류해서 데이터 분할을 진행하였습니다

4. 기타 변수 데이터 처리 - 2

```
cols = ['FullTimeHomeTeamGoals','FullTimeAwayTeamGoals','HalfTimeHomeTeamGoals',
        'HalfTimeAwayTeamGoals','HomeTeamShots','AwayTeamShots','HomeTeamShotsOnTarget',
        'AwayTeamShotsOnTarget','HomeTeamCorners','AwayTeamCorners']
df[cols].isnull().sum()
```

```
FullTimeHomeTeamGoals    0
FullTimeAwayTeamGoals    0
HalfTimeHomeTeamGoals    0
HalfTimeAwayTeamGoals    0
HomeTeamShots            0
AwayTeamShots            0
HomeTeamShotsOnTarget    0
AwayTeamShotsOnTarget    0
HomeTeamCorners          0
AwayTeamCorners          0
dtype: int64
```

FullTimeHomeTeamGoals, FullTimeAwayTeamGoals, HalfTimeHomeTeamGoals, HalfTimeAwayTeamGoals, HomeTeamShots, AwayTeamShots, HomeTeamShotsOnTarget, AwayTeamShotsOnTarget, HomeTeamCorners, AwayTeamCorners를 구간 변수로 설정하여 구간 변수 모두가 결측 값이 없다는 것을 알 수 있습니다.

```
# 범주형 변수를 cols1에 저장
cols1 = ['Season','Date','HomeTeam','AwayTeam','HalfTimeResult_B','Referee','HomeTeamFouls','AwayTeamFouls','HomeTeamYellowCards','AwayTeamYellowCards','HomeTeamRedCards','AwayTeamRedCards']
df[cols1].isnull().sum()
```

```
Season                0
Date                  0
HomeTeam              0
AwayTeam              0
HalfTimeResult_B      0
Referee               0
HomeTeamFouls         0
AwayTeamFouls         0
HomeTeamYellowCards   0
AwayTeamYellowCards   0
HomeTeamRedCards      0
AwayTeamRedCards      0
```

범주형 변수에 Fouls, YellowCards, RedCards를 추가해서 결측값 개수를 확인하였습니다.

5. 요약 통계 및 도수 분포표 검토 - 1

	FullTimeHomeTeamGoals	FullTimeAwayTeamGoals	HalfTimeHomeTeamGoals	HalfTimeAwayTeamGoals	HomeTeamShots	AwayTeamShots	HomeTeamShotsOnTarget	AwayTeamShotsOnTarget	HomeTeamCorners	AwayTeamCorners
count	9097.00	9097.00	9097.00	9097.00	9097.00	9097.00	9097.00	9097.00	9097.00	9097.00
mean	1.54	1.17	0.69	0.51	13.60	10.74	6.03	4.71	6.06	4.77
std	1.31	1.15	0.84	0.73	5.33	4.66	3.29	2.77	3.10	2.74
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	1.00	0.00	0.00	0.00	10.00	7.00	4.00	3.00	4.00	3.00
50%	1.00	1.00	0.00	0.00	13.00	10.00	6.00	4.00	6.00	4.00
75%	2.00	2.00	1.00	1.00	17.00	14.00	8.00	6.00	8.00	6.00
max	9.00	9.00	5.00	5.00	43.00	31.00	24.00	20.00	20.00	19.00

```
df[cols].skew()  # +- 3 이하면 OK
```

```
FullTimeHomeTeamGoals    0.98
FullTimeAwayTeamGoals    1.08
HalfTimeHomeTeamGoals    1.22
HalfTimeAwayTeamGoals    1.46
HomeTeamShots            0.64
AwayTeamShots            0.66
HomeTeamShotsOnTarget    0.82
AwayTeamShotsOnTarget    0.87
HomeTeamCorners          0.62
AwayTeamCorners          0.73
dtype: float64
```

```
df[cols].kurtosis()  # +- 10 이하면 OK
```

```
FullTimeHomeTeamGoals    1.32
FullTimeAwayTeamGoals    1.33
HalfTimeHomeTeamGoals    1.44
HalfTimeAwayTeamGoals    2.26
HomeTeamShots            0.65
AwayTeamShots            0.48
HomeTeamShotsOnTarget    0.95
AwayTeamShotsOnTarget    1.05
HomeTeamCorners          0.41
AwayTeamCorners          0.64
dtype: float64
```

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['Season'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
Season		
2000-2001	196	184
2001-2002	215	165
2002-2003	193	187
2003-2004	213	167
2004-2005	207	173

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['Date'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
Date		
01/01/2001	3	6
01/01/2002	3	4
01/01/2003	1	7
01/01/2005	5	5
01/01/2007	4	4
01/01/2008	3	3

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['HomeTeam'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
HomeTeam		
Arsenal	146	309
Aston Villa	242	156
Birmingham	83	50
Blackburn	123	86
Blackpool	14	5
Bolton	128	81

5. 요약 통계 및 도수 분포표 검토 - 2

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['AwayTeam'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
AwayTeam		
Arsenal	326	129
Aston Villa	218	180
Birmingham	59	74
Blackburn	105	104
Blackpool	9	10
Bolton	98	111

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['HalfTimeResult_B'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
HalfTimeResult_B		
0.00	4291	1619
1.00	620	2567

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['Referee'], df['FullTimeResult_B'])
```

P Tierney	108	74
P Walton	94	75
P. A. Durkin	3	5
P. A. Durkin	1	0
P. Dowd	0	1

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['HomeTeamRedCards'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
HomeTeamRedCards		
0.00	4498	4055
1.00	390	131
2.00	22	0
3.00	1	0

```
# Frequency Table은 모든 범주형 변수 다해보
pd.crosstab(df['HomeTeamFouls'], df['FullTi
```

FullTimeResult_B	0.00	1.00
HomeTeamFouls		
0.00	2	1
1.00	0	2
2.00	8	13
3.00	20	23
4.00	48	46
5.00	113	129

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['AwayTeamFouls'], df['FullTime
```

FullTimeResult_B	0.00	1.00
AwayTeamFouls		
1.00	3	5
2.00	4	6
3.00	31	20
4.00	56	36
5.00	108	90
6.00	176	165
7.00	280	226

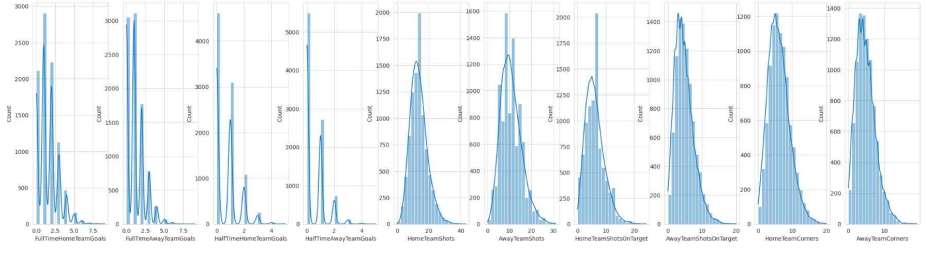
```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['HomeTeamYellowCards'], df['Fu
```

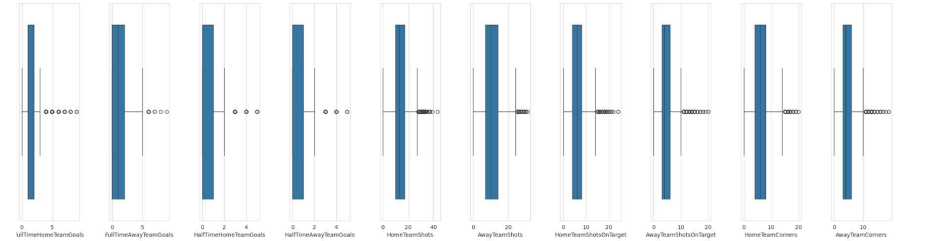
FullTimeResult_B	0.00	1.00
HomeTeamYellowCards		
0.00	969	1250
1.00	1592	1361
2.00	1315	961
3.00	687	434
4.00	258	128
5.00	73	41
6.00	16	9
7.00	1	2

```
# Frequency Table은 모든 범주형 변수 다해보기
pd.crosstab(df['AwayTeamRedCards'], df['FullTimeResult_B'])
```

FullTimeResult_B	0.00	1.00
AwayTeamRedCards		
0.00	4603	3740
1.00	302	422
2.00	6	24

6. 이상값 제거 - 1


count
FullTimeHomeTeamGoals
FullTimeAwayTeamGoals
HalfTimeHomeTeamGoals
HalfTimeAwayTeamGoals
HomeTeamShots
AwayTeamShots
HomeTeamShotsOnTarget
AwayTeamShotsOnTarget
HomeTeamCorners
AwayTeamCorners


ullTimeHomeTeamGoals
FullTimeAwayTeamGoals
HalfTimeHomeTeamGoals
HalfTimeAwayTeamGoals
HomeTeamShots
AwayTeamShots
HomeTeamShotsOnTarget
AwayTeamShotsOnTarget
HomeTeamCorners
AwayTeamCorners

6. 이상값 제거 - 2

```
# 하한 구하기
Lower = Q1-3.0*IQR

# 상한 구하기
Upper = Q3+3.0*IQR

print(Lower)
```

```
FullTimeHomeTeamGoals    -2.0
FullTimeAwayTeamGoals    -6.0
HalfTimeHomeTeamGoals    -3.0
HalfTimeAwayTeamGoals    -3.0
HomeTeamShots           -11.0
AwayTeamShots           -14.0
HomeTeamShotsOnTarget    -8.0
AwayTeamShotsOnTarget    -6.0
HomeTeamCorners          -8.0
AwayTeamCorners          -6.0
dtype: float64
```

```
print(Upper)
```

```
FullTimeHomeTeamGoals     5.0
FullTimeAwayTeamGoals     8.0
HalfTimeHomeTeamGoals     4.0
HalfTimeAwayTeamGoals     4.0
HomeTeamShots            38.0
AwayTeamShots            35.0
HomeTeamShotsOnTarget    20.0
AwayTeamShotsOnTarget    15.0
HomeTeamCorners          20.0
AwayTeamCorners          15.0
dtype: float64
```

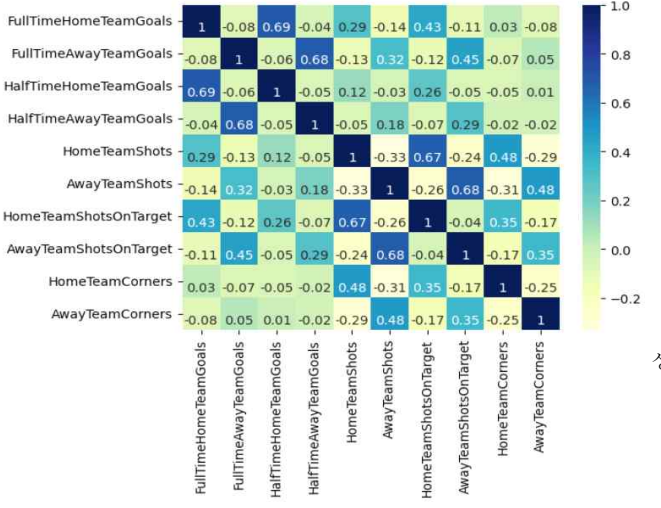
```
df[cols].max()
```

```
FullTimeHomeTeamGoals     9.0
FullTimeAwayTeamGoals     9.0
HalfTimeHomeTeamGoals     5.0
HalfTimeAwayTeamGoals     5.0
HomeTeamShots            43.0
AwayTeamShots            31.0
HomeTeamShotsOnTarget    24.0
AwayTeamShotsOnTarget    20.0
HomeTeamCorners          20.0
AwayTeamCorners          19.0
dtype: float64
```

```
c1 = df['FullTimeHomeTeamGoals'] <= 5.0
c2 = df['FullTimeAwayTeamGoals'] <= 8.0
c3 = df['HalfTimeHomeTeamGoals'] <= 4.0
c4 = df['HalfTimeAwayTeamGoals'] <= 4.0
c5 = df['HomeTeamShots'] <= 38.0
c6 = df['HomeTeamShotsOnTarget'] <= 20.0
c7 = df['AwayTeamShotsOnTarget'] <= 15.0
c8 = df['AwayTeamCorners'] <= 15.0

df1 = df[c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8]
df1.shape
```

Lower은 음수이기 때문에 모든 변수는 0부터 시작해서 상관없고 upper보다 max가 를 경우 이상값이 있기 때문에 조건식을 통해 이상값을 제거하였습니다.

7. 상관 계수 검토, 시각화 및 T-검증 - 1



상관 계수가 0.7을 넘는 변수가 없다.

7. 상관 계수 검토, 시각화 및 T-검증 - 2

	FullTimeHomeTeamGoals	FullTimeAwayTeamGoals	HalfTimeHomeTeamGoals	HalfTimeAwayTeamGoals	HomeTeamShots	AwayTeamShots	HomeTeamShotsOnTarget	AwayTeamShotsOnTarget
FullTimeHomeTeamGoals	1.00	-0.08	0.69	-0.04	0.29	-0.14	0.43	-0.11
FullTimeAwayTeamGoals	-0.08	1.00	-0.06	0.68	-0.13	0.32	-0.12	0.45
HalfTimeHomeTeamGoals	0.69	-0.06	1.00	-0.05	0.12	-0.03	0.26	-0.05
HalfTimeAwayTeamGoals	-0.04	0.68	-0.05	1.00	-0.05	0.18	-0.07	0.29
HomeTeamShots	0.29	-0.13	0.12	-0.05	1.00	-0.33	0.67	-0.24
AwayTeamShots	-0.14	0.32	-0.03	0.18	-0.33	1.00	-0.26	0.68
HomeTeamShotsOnTarget	0.43	-0.12	0.26	-0.07	0.67	-0.26	1.00	-0.04
AwayTeamShotsOnTarget	-0.11	0.45	-0.05	0.29	-0.24	0.68	-0.04	1.00
HomeTeamCorners	0.03	-0.07	-0.05	-0.02	0.48	-0.31	0.35	-0.17
AwayTeamCorners	-0.08	0.05	0.01	-0.02	-0.29	0.48	-0.17	0.35

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['FullTimeHomeTeamGoals']
data_0 = df[df['FullTimeResult_B'] == 0]['FullTimeHomeTeamGoals']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=79.07744066833, pvalue=0.0, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['FullTimeAwayTeamGoals']
data_0 = df[df['FullTimeResult_B'] == 0]['FullTimeAwayTeamGoals']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=-55.67782774598788, pvalue=0.0, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['HalfTimeHomeTeamGoals']
data_0 = df[df['FullTimeResult_B'] == 0]['HalfTimeHomeTeamGoals']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=46.84462351906491, pvalue=0.0, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['HalfTimeAwayTeamGoals']
data_0 = df[df['FullTimeResult_B'] == 0]['HalfTimeAwayTeamGoals']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=-33.92129560846864, pvalue=1.3272863344140575e-237, df=9095.0)
```

7. 상관 계수 검토, 시각화 및 T-검증 - 3

```
data_1 = df[df['FullTimeResult_B'] == 1]['HomeTeamShots']
data_0 = df[df['FullTimeResult_B'] == 0]['HomeTeamShots']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=19.898932467505094, pvalue=2.7997372183490705e-86, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['AwayTeamShots']
data_0 = df[df['FullTimeResult_B'] == 0]['AwayTeamShots']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=-21.151251611971675, pvalue=5.667796188468958e-97, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['HomeTeamShotsOnTarget']
data_0 = df[df['FullTimeResult_B'] == 0]['HomeTeamShotsOnTarget']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=30.572127843876192, pvalue=1.756364675958364e-195, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['AwayTeamShotsOnTarget']
data_0 = df[df['FullTimeResult_B'] == 0]['AwayTeamShotsOnTarget']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=-25.742305538976975, pvalue=4.0778714172504496e-141, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['HomeTeamCorners']
data_0 = df[df['FullTimeResult_B'] == 0]['HomeTeamCorners']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=2.8752469840779558, pvalue=0.0040464864914736?2, df=9095.0)
```

```
from scipy import stats

data_1 = df[df['FullTimeResult_B'] == 1]['AwayTeamCorners']
data_0 = df[df['FullTimeResult_B'] == 0]['AwayTeamCorners']

stats.ttest_ind(data_1, data_0)
```

```
TtestResult(statistic=-3.4906417160673024, pvalue=0.0004841468057575195, df=9095.0)
```

모든 구간 변수의 pvalue가 0.05 미만이라서 귀무 가설을 기각할 수 있다. 즉 두 그룹의 평균이 같다는 가설을 기각할 수 있다. 두 그룹의 평균이 유의미하게 다르다. 두 그룹은 각 구간 변수들과 타겟변수이다.

8. 데이터 추가 처리 for 머신러닝/딥러닝 - 1

HomeTeamYellowCards	HomeTeamYellowCards_encoded	
0.00	0.00	2219
1.00	1.00	2953
2.00	2.00	2276
3.00	3.00	1121
4.00	4.00	386
5.00	5.00	114
6.00	6.00	25
7.00	7.00	3

AwayTeamYellowCards	AwayTeamYellowCards_encoded	
0.00	0.00	1474
1.00	1.00	2712
2.00	2.00	2511
3.00	3.00	1520
4.00	4.00	633
5.00	5.00	201
6.00	6.00	31
7.00	7.00	13
8.00	8.00	1
9.00	9.00	1

HomeTeamRedCards	HomeTeamRedCards_encoded	
0.00	0.00	8553
1.00	1.00	521
2.00	2.00	22
3.00	3.00	1

AwayTeamRedCards	AwayTeamRedCards_encoded	
0.00	0.00	8343
1.00	1.00	724
2.00	2.00	30

HomeTeamFouls	HomeTeamFouls_encoded	
0.00	0	3
1.00	0	2
2.00	0	21
3.00	0	43
4.00	0	94
5.00	1	242
6.00	1	443
7.00	1	539
8.00	1	732
9.00	1	893
10.00	2	948
11.00	2	1003
12.00	2	952
13.00	2	815
14.00	2	670
15.00	3	509
16.00	3	388
17.00	3	280
18.00	3	187
19.00	3	133
20.00	4	79
21.00	4	38
22.00	4	36
23.00	4	22
24.00	4	12
25.00	5	9
26.00	5	5
27.00	5	1
28.00	5	3
33.00	6	1

AwayTeamFouls	AwayTeamFouls_encoded	
1.00	0	8
2.00	0	10
3.00	0	51
4.00	0	92
5.00	1	198
6.00	1	341
7.00	1	506
8.00	1	637
9.00	1	807
10.00	2	934
11.00	2	931
12.00	2	907
13.00	2	857
14.00	2	715
15.00	3	592
16.00	3	440
17.00	3	342
18.00	3	229
19.00	3	163
20.00	4	114
21.00	4	100
22.00	4	49
23.00	4	31
24.00	4	18
25.00	5	8
26.00	5	10
27.00	5	3
28.00	5	3
29.00	5	1

HomeTeam	HomeTeam_encoded	
Arsenal	0.00	445
Aston Villa	1.00	398
Birmingham	2.00	336
Blackburn	3.00	292
Blackpool	4.00	119
Bolton	5.00	287
Bournemouth	6.00	130
Bradford	7.00	79
Brentford	8.00	155
Brighton	9.00	130
Burnley	10.00	169
Cardiff	11.00	98
Charlton	12.00	135
Chelsea	13.00	440
Coventry	14.00	19
Crystal Palace	15.00	227
Derby	16.00	95
Everton	17.00	451
Fulham	18.00	318
Huddersfield	19.00	38
Hull	20.00	94
Ipswich	21.00	38
Leeds	22.00	132
Leicester	23.00	233
Liverpool	24.00	449
Luton	25.00	18
Man City	26.00	415
Man United	27.00	444
Middlesbrough	28.00	186
Newcastle	29.00	413
Norwich	30.00	139
Nott's Forest	31.00	37
Portsmouth	32.00	139
QPR	33.00	56
Reading	34.00	54
Sheffield United	35.00	74
Southampton	36.00	306
Stoke	37.00	189
Sunderland	38.00	266
Swansea	39.00	136
Tottenham	40.00	451
Watford	41.00	133
West Brom	42.00	246
West Ham	43.00	395
Wigan	44.00	152
Wolves	45.00	187

AwayTeam	AwayTeam_encoded	
Arsenal	0.00	446
Aston Villa	1.00	393
Birmingham	2.00	133
Blackburn	3.00	294
Blackpool	4.00	18
Bolton	5.00	277
Bournemouth	6.00	129
Bradford	7.00	17
Brentford	8.00	59
Brighton	9.00	131
Burnley	10.00	170
Cardiff	11.00	39
Charlton	12.00	133
Chelsea	13.00	440
Coventry	14.00	19
Crystal Palace	15.00	227
Derby	16.00	95
Everton	17.00	451
Fulham	18.00	321
Huddersfield	19.00	38
Hull	20.00	93
Ipswich	21.00	38
Leeds	22.00	129
Leicester	23.00	224
Liverpool	24.00	452
Luton	25.00	18
Man City	26.00	429
Man United	27.00	450
Middlesbrough	28.00	189
Newcastle	29.00	413
Norwich	30.00	139
Nott's Forest	31.00	36
Portsmouth	32.00	132
QPR	33.00	53
Reading	34.00	55
Sheffield United	35.00	75
Southampton	36.00	299
Stoke	37.00	187
Sunderland	38.00	261
Swansea	39.00	133
Tottenham	40.00	446
Watford	41.00	131
West Brom	42.00	241
West Ham	43.00	395
Wigan	44.00	150
Wolves	45.00	188

dtype: int64

YellowCards와 RedCards는 1단위(장)로 분할 하였고, Fouls는 5단위(번)로 끊어서 분할 하였습니다. Home Team과 Away Team도 팀 당으로 분류하였습니다.

8. 데이터 추가 처리 for 머신러닝/딥러닝 - 2

Season	Season_encoded	
2000-2001	0.00	380
2001-2002	1.00	380
2002-2003	2.00	380
2003-2004	3.00	380
2004-2005	4.00	380
2005-2006	5.00	380
2006-2007	6.00	380
2007-2008	7.00	380
2008-2009	8.00	380
2009-2010	9.00	380
2010-2011	10.00	380
2011-2012	11.00	380
2012-2013	12.00	380
2013-2014	13.00	380
2014-2015	14.00	380
2015-2016	15.00	380
2016-2017	16.00	380
2017-2018	17.00	380
2018-2019	18.00	380
2019-2020	19.00	380
2020-2021	20.00	380
2021-2022	21.00	380
2022-2023	22.00	380
2023-2024	23.00	357

```
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df.drop('Date', axis=1, inplace=True)
```

FullTimeHomeTeamGoals	8978	non-null	int64
FullTimeAwayTeamGoals	8978	non-null	int64
HalfTimeHomeTeamGoals	8978	non-null	float64
HalfTimeAwayTeamGoals	8978	non-null	float64
HomeTeamShots	8978	non-null	float64
AwayTeamShots	8978	non-null	float64
HomeTeamShotsOnTarget	8978	non-null	float64
AwayTeamShotsOnTarget	8978	non-null	float64
HomeTeamCorners	8978	non-null	float64
AwayTeamCorners	8978	non-null	float64
FullTimeResult_B	8978	non-null	float64
HalfTimeResult_B	8978	non-null	float64
HomeTeamYellowCards_encoded	8978	non-null	float64
AwayTeamYellowCards_encoded	8978	non-null	float64
HomeTeamRedCards_encoded	8978	non-null	float64
AwayTeamRedCards_encoded	8978	non-null	float64
HomeTeam_encoded	8978	non-null	float64
AwayTeam_encoded	8978	non-null	float64
HomeTeamFouls_encoded	8978	non-null	int64
AwayTeamFouls_encoded	8978	non-null	int64
Season_encoded	8978	non-null	float64
Year	8978	non-null	int32
Month	8978	non-null	int32
Day	8978	non-null	int32

Date 부분이 '/'로 처리 되어있어서 년, 월, 일로 분할해서 처리하였고, Season도 '-'로 처리 되어있어서 구간을 나눠서 분할하여 처리하였습니다.

8. 데이터 추가 처리 for 머신러닝/딥러닝 - 3

HomeTeamYellowCards_encoded

1.0	2908
2.0	2260
0.0	2176
3.0	1111
4.0	381
5.0	114
6.0	25
7.0	3

AwayTeamYellowCards_encoded

1.0	2666
2.0	2487
3.0	1499
0.0	1453
4.0	626
5.0	201
6.0	31
7.0	13
8.0	1
9.0	1

HomeTeamRedCards_encoded

0.0	8440
1.0	515
2.0	22
3.0	1

AwayTeamRedCards_encoded

0.0	8242
1.0	707
2.0	29

HomeTeamFouls_encoded

2	4333
1	2800
3	1485
4	185
0	156
5	18
6	1

AwayTeamFouls_encoded

2	4297
1	2433
3	1759
4	310
0	154
5	25

HomeTeam_encoded

40.0	451
17.0	451
24.0	449
0.0	445
27.0	444
13.0	440
26.0	416
29.0	413
43.0	395
1.0	391
18.0	318
36.0	301
38.0	266

AwayTeam_encoded

24.0	452
17.0	450
27.0	450
13.0	449
40.0	446
0.0	446
26.0	429
29.0	413
43.0	395
1.0	393
18.0	321
36.0	299

기준 더미 변수 제거를 각 인코딩한 더미 변수들 값중 가장 value_counts가 높은 것을 기준으로 삼아 제거하겠습니다. 가장 value_counts가 높은 것을 기준으로 삼는 이유는 빈도수가 가장 많아서 비교에 용이하다고 생각했기 때문입니다.

Home Team Yellow Cards : 1장
Away Team Yellow Cards : 1장
Home Team Red Cards : 0장
Away Team Red Cards : 0장
Home Team Fouls : 10~14번
Away Team Fouls : 10~14번
Home Team : 토트넘
Away Team : 리버풀

9. 로지스틱 회귀 모델 - 1

```
X_train shape: (4489, 137)    GridSearchCV max accuracy:0.92359
X_test shape: (4489, 137)     GridSearchCV best parameter: {'penalty': 'none', 'solver': 'lbfgs'}
```

```
Accuracy on test set:0.90688
```

```
Logreg Training set score:0.89797
Logreg Test set score:0.90688
```

```
절편 [0.]
회귀계수 [[ 2.466e+00 -1.831e+00  9.680e-01 -7.220e-01 -1.760e-01  1.760e-01
   5.150e-01 -5.010e-01  3.500e-02  1.720e-01  6.950e-01 -2.000e-02
  -1.000e-03  6.100e-02 -7.900e-02  1.090e-01 -2.600e-02 -3.400e-02
  -3.500e-02 -1.800e-02  1.000e-03  1.000e-03 -2.000e-03 -6.000e-03
  -1.600e-02 -7.000e-03 -6.000e-03 -1.000e-03 -2.000e-03  0.000e+00
   0.000e+00 -6.800e-02 -8.000e-03  0.000e+00  5.200e-02  1.100e-02
   1.500e-02  1.600e-02 -6.900e-02  3.000e-03 -3.000e-03 -1.000e-03
  -2.000e-03  2.100e-02 -2.300e-02 -2.000e-03  2.000e-03  4.200e-02
  -1.400e-02 -1.700e-02 -8.000e-03  2.000e-03 -2.900e-02  2.000e-03
  -0.000e+00 -6.000e-03 -1.200e-02 -1.400e-02  3.000e-03  8.000e-03
   2.100e-02 -2.000e-03 -1.200e-02 -1.000e-02  7.000e-03  9.000e-03
  -4.000e-03 -4.000e-03  1.000e-03 -1.800e-02 -3.000e-03  3.400e-02
  -4.000e-03  2.300e-02  6.900e-02  1.000e-03  4.000e-03 -2.000e-03
   2.000e-03 -1.000e-02 -5.000e-03  4.000e-03 -1.300e-02 -1.500e-02
   6.000e-03 -2.300e-02 -2.000e-03 -3.000e-03 -2.000e-02  9.000e-03
  -2.300e-02  0.000e+00 -3.000e-02  2.000e-03 -5.000e-03  8.000e-03
  -4.000e-03  1.300e-02  1.300e-02  4.000e-03  0.000e+00 -2.000e-03
   5.000e-03  9.000e-03 -9.000e-03 -3.600e-02  4.000e-03 -1.000e-03
   5.000e-03 -2.700e-02  6.000e-03  6.000e-03  9.000e-03  3.000e-03
   1.000e-03  0.000e+00  1.000e-03 -3.100e-02 -7.900e-02  3.000e-03
   7.000e-03  5.000e-03 -2.000e-03  1.500e-02  1.300e-02  2.000e-03
   1.600e-02  2.500e-02  1.000e-03  2.700e-02  6.000e-03 -9.000e-03
   1.900e-02 -7.000e-03  8.000e-03  1.700e-02  1.000e-03]]
```

<성능 평가>

1. 로지스틱 회귀 모델: 0.90688

	coef
FullTimeAwayTeamGoals	-1.831
HalfTimeAwayTeamGoals	-0.722
AwayTeamShotsOnTarget	-0.501
HomeTeamShots	-0.176
AwayTeam_encoded_27.0	-0.079
Day	-0.079
HomeTeamFouls_encoded_3	-0.069
HomeTeamRedCards_encoded_1.0	-0.068
AwayTeam_encoded_13.0	-0.036
HomeTeamYellowCards_encoded_4.0	-0.035
HomeTeamYellowCards_encoded_3.0	-0.034
AwayTeam_encoded_26.0	-0.031
AwayTeam_encoded_0.0	-0.030
HomeTeam_encoded_5.0	-0.029

9. 로지스틱 회귀 모델 - 2

	Odds_ratio
FullTimeHomeTeamGoals	11.778
HalfTimeHomeTeamGoals	2.632
HalfTimeResult_B	2.004
HomeTeamShotsOnTarget	1.673
AwayTeamShots	1.192
AwayTeamCorners	1.188
HomeTeamYellowCards_encoded_0.0	1.115
HomeTeam_encoded_27.0	1.072
Month	1.063
AwayTeamRedCards_encoded_1.0	1.054
HomeTeam_encoded_0.0	1.043
HomeTeamCorners	1.035
HomeTeam_encoded_24.0	1.034
AwayTeam_encoded_38.0	1.027
AwayTeam_encoded_36.0	1.025
HomeTeam_encoded_26.0	1.024
HomeTeam_encoded_13.0	1.021

HomeTeam_encoded_5.0	0.972
AwayTeam_encoded_0.0	0.970
AwayTeam_encoded_26.0	0.970
HomeTeamYellowCards_encoded_4.0	0.966
HomeTeamYellowCards_encoded_3.0	0.966
AwayTeam_encoded_13.0	0.965
HomeTeamRedCards_encoded_1.0	0.934
HomeTeamFouls_encoded_3	0.933
Day	0.924
AwayTeam_encoded_27.0	0.924
HomeTeamShots	0.839
AwayTeamShotsOnTarget	0.606
HalfTimeAwayTeamGoals	0.486
FullTimeAwayTeamGoals	0.160

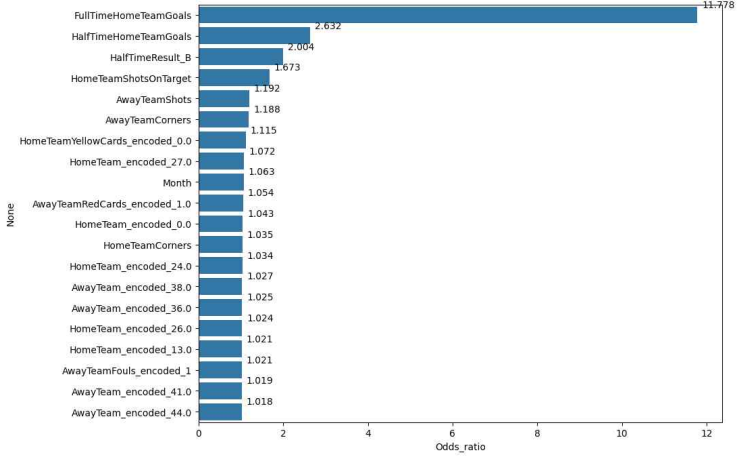
Half Time Home Team Goals가 1단위(득점)가 증가할 경우, 홈팀이 이길 가능성은 2.632만큼 변합니다. 즉 263% 증가합니다.

Home Team Yellow Cards가 1단위(장)일 때와 비교하여, Home Team Yellow Cards가 0단위(장)일 경우에 홈팀이 이길 가능성은 1.115배 높습니다.

Away Team Shots On Target이 1단위(개) 증가할 경우, 홈팀이 이길 가능성은 0.606만큼 변합니다. 즉 39.4% 감소합니다.

Home Team Fouls 개수가 10~14번일 때와 비교하여, Home Team Fouls 개수가 15~19번으로 증가할 경우에 홈팀이 이길 가능성은 0.933배 낮습니다.

9. 로지스틱 회귀 모델 - 3


FullTimeHomeTeamGoals
HalfTimeHomeTeamGoals
HalfTimeResult_B
HomeTeamShotsOnTarget
AwayTeamShots
AwayTeamCorners
HomeTeamYellowCards_encoded_0.0
HomeTeam_encoded_27.0
Month
AwayTeamRedCards_encoded_1.0
HomeTeam_encoded_0.0
HomeTeamCorners
HomeTeam_encoded_24.0
AwayTeam_encoded_38.0
AwayTeam_encoded_36.0
HomeTeam_encoded_26.0
HomeTeam_encoded_13.0
AwayTeamFouls_encoded_1
AwayTeam_encoded_41.0
AwayTeam_encoded_44.0
11.778
2.632
2.004
1.673
1.192
1.188
1.115
1.072
1.063
1.054
1.043
1.035
1.034
1.027
1.025
1.024
1.021
1.021
1.019
1.018
None
0
2
4
6
8
10
12
Odds_ratio

10. 표준화한 로지스틱 회귀 모델

```
numeric_cols = ['FullTimeHomeTeamGoals','FullTimeAwayTeamGoals','HalfTimeHomeTeamGoals',
        'HalfTimeAwayTeamGoals','HomeTeamShots','AwayTeamShots','HomeTeamShotsOnTarget',
        'AwayTeamShotsOnTarget','HomeTeamCorners','AwayTeamCorners']
df_num = df[numeric_cols]

# StandardScaler( )로 데이터 스케일 표준화를 실행하고 결과를 데이터프레임으로 만든다.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_num_standard = pd.DataFrame(scaler.fit_transform(df_num))

# StandardScaler( )는 변수명을 지우므로, 새로 만든 데이터프레임에 다시 변수명을 넣는다.
df_num_standard.columns = df_num.columns
df_num_standard.head()
```

절편 [-0.]
회귀계수 [[3.620e+00 -2.915e+00 1.595e+00 -1.426e+00 -1.240e-01 -1.220e-01
  5.462e-01 -5.542e-01 -5.249e-01 4.806e-01 8.733e-03 2.060e-03
 -0.000e+00 -2.400e-02 -5.000e-03 1.610e-01 -3.200e-02 -5.000e-02
 -5.500e-02 -3.300e-02 0.000e+00 1.000e-03 -9.000e-03 1.000e-03
 -2.500e-02 -2.400e-02 -1.200e-02 -2.000e-03 -3.000e-03 0.000e+00
  0.000e+00 -1.280e-01 -1.300e-02 0.000e+00 8.800e-02 2.200e-02
  2.400e-02 3.000e-02 -1.160e-01 5.000e-03 -6.000e-03 -3.000e-03
 -2.000e+00 4.550e-03 -3.500e-02 2.000e-03 4.000e-03 6.000e-02
 -2.400e-02 -2.800e-02 -1.400e-02 2.000e-03 -4.800e-02 3.000e-03
 -4.000e-03 -1.000e-02 -2.200e-02 -2.000e-02 6.000e-03 1.100e-02
  2.200e-02 -4.000e-03 -1.800e-02 -2.000e-02 1.900e-02 2.100e-02
 -6.000e-03 -7.000e-03 3.000e-03 -3.600e-02 -6.000e-03 4.400e-02
 -5.000e-03 3.500e-02 1.168e-01 -1.000e-03 -1.000e-03 4.000e-03
  8.000e-03 -9.000e-03 -8.000e-03 8.000e-03 -2.000e-02 -2.600e-02
  7.000e-03 -2.500e-02 -2.000e-03 -0.000e+00 -2.600e-02 7.000e-03
 -3.400e-02 5.000e-03 -4.400e-02 1.000e-02 -6.000e-03 1.400e-02
 -6.000e-03 1.900e-02 2.500e-02 7.000e-03 -4.000e-03 -7.000e-03
 -6.000e-03 1.100e-02 -1.600e-02 -5.100e-02 1.100e-02 -1.000e-03
  5.000e-03 -5.000e-02 8.000e-03 1.200e-02 9.000e-03 6.000e-03
  1.000e-03 -1.500e-02 2.000e-03 -4.300e-02 -1.410e-01 -0.000e+00
  2.400e-02 8.000e-03 -4.000e-03 2.000e-02 2.200e-02 7.000e-03
  3.000e-02 4.200e-02 4.000e-03 4.300e-02 1.100e-02 -1.200e-02
  3.900e-02 -2.000e-02 1.700e-02 3.400e-02 8.000e-03]

```
X_train shape: (4489, 137)
X_test shape: (4489, 137)

Logreg Training set score:0.91401
Logreg Test set score:0.91802

GridSearchCV max accuracy:0.94809
GridSearchCV best parameter: {'penalty': 'none', 'solver': 'lbfgs'}
```

Accuracy on test set:0.91802

	coef
FullTimeAwayTeamGoals	-2.915
HalfTimeAwayTeamGoals	-1.426
AwayTeamShotsOnTarget	-0.554
HomeTeamCorners	-0.524
AwayTeam_encoded_27.0	-0.141
HomeTeamRedCards_encoded_1.0	-0.128
HomeTeamShots	-0.124
AwayTeamShots	-0.122
HomeTeamFouls_encoded_3	-0.116
HomeTeamYellowCards_encoded_4.0	-0.065
AwayTeam_encoded_13.0	-0.051
AwayTeam_encoded_17.0	-0.050
HomeTeamYellowCards_encoded_3.0	-0.050
HomeTeam_encoded_5.0	-0.048
AwayTeam_encoded_0.0	-0.044
AwayTeam_encoded_26.0	-0.043

<성능 평가>
1. 표준화 로지스틱 회귀 모델: 0.91802
2. 로지스틱 회귀 모델: 0.90688

11. Linear 회귀 및 Ridge 모델

<Linear Regression 모델>

```
data = df.drop(['FullTimeResult_B'], axis=1)   # 타겟
target = df['FullTimeResult_B']                # 타겟

# 50:50 data partition.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data, target, test_size=0.5, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
X_train shape: (4489, 137)
X_test shape: (4489, 137)
```

<Ridge 모델>

```
Linear Regression Training set score:0.65413
Linear Regression Test set score:0.64222

GridSearchCV max score:0.63567
GridSearchCV best parameter: {'alpha': 100, 'solver': 'saga'}

R2 Score on test set:0.64862
```

```
Linear Regression Training set r2 score:0.65431
Linear Regression Test set r2 score:0.64127
```

<표준화 Ridge 모델>

```
Linear Regression Training set score:0.65413
Linear Regression Test set score:0.64223

GridSearchCV max score:0.63540
GridSearchCV best parameter: {'alpha': 100, 'solver': 'saga'}

R2 Score on test set:0.64886
```

<성능 평가>

1. 표준화 로지스틱 회귀 모델 : 0.91802
2. 로지스틱 회귀 모델 : 0.90688
3. 표준화 Ridge 모델 : 0.64886
4. Ridge 모델 : 0.64862
5. Linear Regression 모델 : 0.64127

12. Lasso 모델

```
X_train shape: (4489, 137)
X_test shape: (4489, 137)
```

```
# Lasso 모델 (Default 모델 for liblinear)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
ls = LogisticRegression(penalty='l1',solver='liblinear',C=1,random_state=0)
model = ls.fit(X_train, y_train)
pred = model.predict(X_test)  # 학습된 Classifier로 테스트 데이터셋 자료이용해서 타겟변수 예측값 생성

print ("Lasso Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
print ("Lasso Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Lasso Accuracy on training set:1.00000
Lasso Accuracy on test set:1.00000
```

```
best_clf = grid_ls.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
Accuracy on test set:1.00000
ROC AUC on test set:1.00000
```

Train Data Set과 Test Data Set에 대한 정확도가 모두 1이 나왔기 때문에 과적합 되었다고 볼 수 있다.

<성능 평가>

1. 표준화 로지스틱 회귀 모델 : 0.91802
2. 로지스틱 회귀 모델 : 0.90688
3. 표준화 Ridge 모델 : 0.64886
4. Ridge 모델 : 0.64862
5. Linear Regression 모델 : 0.64127

13. K-NN 모델

```
# KNN 모델 (Default 모델 with n_neighbors=3)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

clf_knn = KNeighborsClassifier(n_neighbors=3) # random_state 파라미터가 없음에 주의!
clf_knn.fit(X_train, y_train)
pred = clf_knn.predict(X_test)   # 학습된 Classifier로 테스트 데이터셋 자료이용해서 타겟변수 예측값 생성
accuracy = accuracy_score(y_test, pred)

print ("KNN Training set score:{:.5f}".format(clf_knn.score(X_train, y_train)))
print ("KNN Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
KNN Training set score:0.90065
KNN Test set score:0.76899
```

```
[5] # KNN 모델 (Default 모델 with n_neighbors=3)
clf_knn = KNeighborsClassifier(n_neighbors=3) # random_state 파라미터가 없음에 주의!

# 그리드 서치 실행
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors': range(3, 31)}

grid_knn = GridSearchCV(clf_knn, param_grid=params, scoring='accuracy', cv=3, n_jobs=-1)
grid_knn.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_knn.best_score_))
print("GridSearchCV best parameter:", (grid_knn.best_params_))
```

```
GridSearchCV max accuracy:0.78391
GridSearchCV best parameter: {'n_neighbors': 25}
```

```
[6] best_clf = grid_knn.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Accuracy on test set:0.79728
```

<성능 평가>

1. 표준화 로지스틱 회귀 모델 : 0.91802
2. 로지스틱 회귀 모델 : 0.90688
3. K-NN 모델 : 0.79728
4. 표준화 Ridge 모델 : 0.64886
5. Ridge 모델 : 0.64862
6. Linear Regression 모델 : 0.64127

14. 사이킷런 신경망 모델

```
# Neural Network 모델 (Default 모델 with adam solver)
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
clf_mlp = MLPClassifier(max_iter = 2000, random_state = 0)
                    # convergence warinng을 회피하기 위해 max_iter = 2000으로 올려서 설정
clf_mlp.fit(X_train, y_train)
pred = clf_mlp.predict(X_test)  # 학습된 Classifier로 테스트 데이터셋 자료이용해서 타겟변수 예측값 생성
accuracy = accuracy_score(y_test, pred)

print ("Neural Network Training set score:{:.5f}".format(clf_mlp.score(X_train, y_train)))
print ("Neural Network Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Neural Network Training set score:1.00000
Neural Network Test set score:1.00000
```

사이킷런 신경망 모델은 Train Data Set과 Test Data Set에 대한 정확도가 모두 1이 나왔기 때문에 과적합 되었다고 볼 수 있다.

```
GridSearchCV max accuracy:1.00000
GridSearchCV best parameter: {'activation': 'tanh', 'alpha': 0.1, 'solver': 'adam'}
```

```
best_clf = grid_mlp.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Accuracy on test set:1.00000
```

<성능 평가>

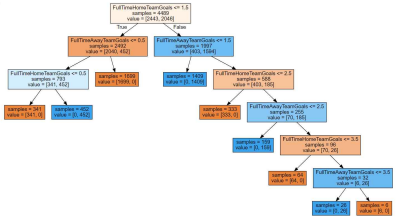
1. 표준화 로지스틱 회귀 모델 : 0.91802
2. 로지스틱 회귀 모델 : 0.90688
3. K-NN 모델 : 0.79728
4. 표준화 Ridge 모델 : 0.64886
5. Ridge 모델 : 0.64862
6. Linear Regression 모델 : 0.64127

15. 결정 트리 모델

```
[10] # Decision Tree 모델 (Default인 GINI기준이자 maximal depth 조건)
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import accuracy_score
     tree = DecisionTreeClassifier(random_state=0)  # Classifier로 DecisionTreeClassifier 지정
     model = tree.fit(X_train, y_train)  # Classifier를 트레이닝셋 (GINI인자)데이터 학습시킴
     pred = model.predict(X_test)  # 학습된 Classifier로 테스트 (GINI인자 지표)데이터셋 타겟변수 예측값 생성

     print("Accuracy(GINI) on training set:{:.5f}".format(model.score(X_train, y_train)))
     print("Accuracy(GINI) on test set:{:.5f}".format(accuracy_score(y_test, pred)))

     Accuracy(GINI) on training set:1.00000
     Accuracy(GINI) on test set:0.99855

[11] # Decision Tree 모델 (Gini 기준)
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import GridSearchCV

     tree = DecisionTreeClassifier(criterion='gini', random_state=0, max_depth=5)

     params = {'criterion':['gini','entropy'],'max_depth':range(1,21)}

     grid_tree = GridSearchCV(tree, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1,
                              verbose=1)
     grid_tree.fit(X_train, y_train)

     print("GridSearchCV max accuracy:{:.5f}".format(grid_tree.best_score_))
     print("GridSearchCV best parameter:", grid_tree.best_params_)

     Fitting 5 folds for each of 40 candidates, totalling 200 fits
     GridSearchCV max accuracy:1.00000
     GridSearchCV best parameter: {'criterion': 'gini', 'max_depth': 8}

[12] # 참조 코딩
     # range() 함수의 결과를 디스플레이

     for i in range(1,21):
         print(i, end = " ")

     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

[13] best_clf = grid_tree.best_estimator_
     pred = best_clf.predict(X_test)
     print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

     Accuracy on test set:0.99855

[14] # 참조 코딩
     from sklearn.metrics import roc_auc_score
     ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(X_test)[:, 1])
     print('ROC AUC on test set:{:.5f}'.format(ROC_AUC))

     ROC AUC on test set:0.99951
```

```
[115] # 변수명을 Index로 만들고 feature_importances를 매칭해서 나열한 데이터프레

feature_names = list(data.columns)  # 변수명(컬럼명)을 리스트 형태로 만
dft = pd.DataFrame(np.round(best_clf.feature_importances_, 4), index=feat
                   columns=['Feature_importances'])

dft1 = dft.sort_values(by='Feature_importances', ascending=False)
dft1                                  # 결정 Feature_importances의 값들을
```

	Feature_importances
FullTimeHomeTeamGoals	0.6343
FullTimeAwayTeamGoals	0.3657
AwayTeam_encoded_44.0	0.0000
HomeTeam_encoded_38.0	0.0000
AwayTeam_encoded_7.0	0.0000
AwayTeam_encoded_6.0	0.0000



홈 팀이 승리하는데 중요한 변수로는 Full Time Home Team Goals가 중요하고 다음으로는 Full Time Away Team Goals가 중요하다.

<성능 평가>

1. 결정 트리 모델 : 0.99951
2. 표준화 로지스틱 회귀 모델 : 0.91802
3. 로지스틱 회귀 모델 : 0.90688
4. K-NN 모델 : 0.79728
5. 표준화 Ridge 모델 : 0.64886
6. Ridge 모델 : 0.64862
7. Linear Regression 모델 : 0.64127

16. SVM 모델

[121] # SVM model (default 모델)

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svm = SVC(kernel='rbf', C=1, gamma = 'auto', random_state=0, probability=True)
                                                    # probability=True 꼭 주의

model = svm.fit(X_train, y_train)
pred = model.predict(X_test)  # 학습된 Classifier로 테스트 데이터셋 자료이름해서 타겟변수 예측
accuracy = accuracy_score(y_test, pred)

print ("SVM Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
print ("SVM Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

SVM Accuracy on training set:0.98552
SVM Accuracy on test set:0.98582

[122] # SVM model (default 모델)

```
svm = SVC(kernel='rbf', C=1, gamma = 'auto', random_state=0, probability=True)
                                                    # probability=True 꼭 주의
```

[123] # 그리드 서치 실행. 시간이 많이 걸림에 주의

```
import time
start = time.time()

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# StratifiedFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
params = {'kernel':['sigmoid'], 'C':[0.0001, 0.01, 1, 10],
          'gamma':['auto','scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
                        cv=cross_validation, n_jobs=-1)
grid_svm.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))
print("GridSearchCV best parameter:", (grid_svm.best_params_))

end = time.time()
print(f"Runtime of the program is {end - start}")
```

GridSearchCV max accuracy:0.54422
GridSearchCV best parameter: {'C': 0.0001, 'gamma': 'auto', 'kernel': 'sigmoid'}
Runtime of the program is 140.7663618934336

[ ] # SVM model (default 모델)

```
svm = SVC(kernel='rbf', C=1, gamma = 'auto', random_state=0, probability=True)
                                                    # probability=True 꼭 주의
```

[125] # 그리드 서치 재실행. 시간이 많이 걸림에 주의

```
import time
start = time.time()

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# StratifiedFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
params = {'kernel':['rbf'], 'C':[0.0001, 0.01, 1, 10],
          'gamma':['auto','scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
                        cv=cross_validation, n_jobs=-1)
grid_svm.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))
print("GridSearchCV best parameter:", (grid_svm.best_params_))

end = time.time()
print(f"Runtime of the program is {end - start}")
```

GridSearchCV max accuracy:0.99911
GridSearchCV best parameter: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Runtime of the program is 144.45130995592298

[130] best_clf = grid_svm.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

```
from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

Accuracy on test set:0.99978
ROC AUC on test set:1.00000

<성능 평가>

1. SVM 모델 : 0.99978
2. 결정 트리 모델 : 0.99951
3. 표준화 로지스틱 회귀 모델 : 0.91802
4. 로지스틱 회귀 모델 : 0.90688
5. K-NN 모델 : 0.79728
6. 표준화 Ridge 모델 : 0.64886
7. Ridge 모델 : 0.64862
8. Linear Regression 모델 : 0.64127

17. 그레디언트 부스팅 모델

```
[41] # Gradient Boosting 모델 (Default 모델)
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.metrics import accuracy_score
     gr = GradientBoostingClassifier(random_state = 0)
     model = gr.fit(X_train, y_train)
     pred = model.predict(X_test) # 학습된 Classifier로 테스트 데이터셋 타겟변수 예측값 생성
     accuracy = accuracy_score(y_test, pred)

     print ("grbt Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
     print ("grbt Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

     grbt Accuracy on training set:1.00000
     grbt Accuracy on test set:0.99955
```

```
[42] # Gradient Boosting 모델 (Default 모델)
     gr = GradientBoostingClassifier(random_state = 0)
```

```
     # 그리드 서치 실행
     # 아래 코딩 실행에 한참 소요
     import time
     start = time.time()

     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import StratifiedKFold

     # Gradient Boosting 모델 (Default 모델)
     gr = GradientBoostingClassifier(random_state = 0)

     # StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
     cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
     params = {'max_depth': range(5, 51, 5)
      }
     # GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedFold와 random_state 옵션값을 적용시켜서
     # GridSearchCV를 실행할 데이터 결과가 항상 동일하게 나오도록 설정
     grid_gr = GridSearchCV(model, param_grid=params, scoring='accuracy', cv=cross_validation,
                            n_jobs=-1)
     grid_gr.fit(X_train, y_train)

     print("GridSearchCV max accuracy:{:.5f}".format(grid_gr.best_score_))
     print("GridSearchCV best parameter:", grid_gr.best_params_)

     end = time.time()
     print(f"Runtime of the program is {end - start}")

     GridSearchCV max accuracy:1.00000
     GridSearchCV best parameter: {'max_depth': 5}
     Runtime of the program is 36.40998447113037
```

```
[44] # 그리드 서치 추가 실행
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import StratifiedKFold

     # Gradient Boosting 모델 (Default 모델)
     gr = GradientBoostingClassifier(random_state = 0)

     # StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
     cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
     params = {'max_depth': range(3, 10), 'n_estimators':[200]}

     # GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedFold와 random_state 옵션값을 적용시켜서
     # GridSearchCV를 실행할 데이터 결과가 항상 동일하게 나오도록 설정
     grid_gr = GridSearchCV(model, param_grid=params, scoring='accuracy', cv=cross_validation,
                            n_jobs=-1)
     grid_gr.fit(X_train, y_train)

     print("GridSearchCV max accuracy:{:.5f}".format(grid_gr.best_score_))
     print("GridSearchCV best parameter:", grid_gr.best_params_)

     GridSearchCV max accuracy:1.00000
     GridSearchCV best parameter: {'max_depth': 5, 'n_estimators': 200}
```

```
[47] # 그리드 서치 추가 실행
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import StratifiedKFold

     # Gradient Boosting 모델 (Default 모델)
     gr = GradientBoostingClassifier(random_state = 0)

     # StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
     cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
     params = {'max_depth': range(1, 10), 'n_estimators':[100,200],
               'learning_rate':[0.01, 0.1, 1]}

     # GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedFold와 random_state 옵션값을 적용시켜서
     # GridSearchCV를 실행할 데이터 결과가 항상 동일하게 나오도록 설정
     grid_gr = GridSearchCV(model, param_grid=params, scoring='accuracy', cv=cross_validation,
                            n_jobs=-1)
     grid_gr.fit(X_train, y_train)

     print("GridSearchCV max accuracy:{:.5f}".format(grid_gr.best_score_))
     print("GridSearchCV best parameter:", grid_gr.best_params_)

     GridSearchCV max accuracy:1.00000
     GridSearchCV best parameter: {'learning_rate': 0.01, 'max_depth': 11, 'n_estimators': 100}
```

```
[10] best_clf = grid_gr.best_estimator_
     pred = best_clf.predict(X_test)
     print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

     from sklearn.metrics import roc_auc_score
     ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
     print("ROC AUC on test set:{:.5f}".format(ROC_AUC))

     Accuracy on test set:0.99955
     ROC AUC on test set:0.99954
```

<성능 평가>

1. SVM 모델 : 0.99978
2. Gradient Boosting 모델 : 0.99955
3. 결정 트리 모델 : 0.99951
4. 표준화 로지스틱 회귀 모델 : 0.91802
5. 로지스틱 회귀 모델 : 0.90688
6. K-NN 모델 : 0.79728
7. 표준화 Ridge 모델 : 0.64886
8. Ridge 모델 : 0.64862
9. Linear Regression 모델 : 0.64127

18. 랜덤 포레스트 모델

```
[84] # Random Forest 모델 (Default 모델, tree depth 제한 없음)
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
rf = RandomForestClassifier(n_estimators = 100, random_state=0)
model = rf.fit(X_train, y_train)
pred = model.predict(X_test)  # 학습된 Classifier로 테스트 데이터셋 자료이용해서 타겟변수 예측값 생성
#accuracy = accuracy_score(y_test, pred)

print ("Random Forest Accuracy on training set:{:.5f}".format(model.score(X_train, y_train)))
print ("Random Forest Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
Random Forest Accuracy on training set:1.00000
Random Forest Accuracy on test set:0.98819
```

```
[85] # Random Forest 모델 (Default 모델, tree depth 제한 없음)
rf = RandomForestClassifier(n_estimators = 100, random_state=0)
```

```
[86] # 그리드 서치 실행

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

# StratifiedKFold의 random_state 옵션값을 특정 숫자(예: 0)로 고정
cross_validation = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
params = {'max_depth': range(10, 41), 'n_estimators': [100, 200]}

# GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedKFold의 random_state 옵션값을 적용시켜서
# GridSearchCV를 실행할 때마다 결과가 항상 동일하게 나오도록 보장
grid_rf = GridSearchCV(rf, param_grid=params, scoring='accuracy', cv=cross_validation,
                       verbose=1, n_jobs=-1)
grid_rf.fit(X_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_rf.best_score_))
print("GridSearchCV best parameter:", (grid_rf.best_params_))
```

```
Fitting 5 folds for each of 62 candidates, totalling 310 fits
GridSearchCV max accuracy:0.98619
GridSearchCV best parameter: {'max_depth': 26, 'n_estimators': 200}
```

```
[87] best_clf = grid_rf.best_estimator_
pred = best_clf.predict(X_test)
print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test, pred)))

from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(X_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
Accuracy on test set:0.98842
ROC AUC on test set:0.99941
```

```
# 참조 코딩
print ("Feature importances:")
print (best_clf.feature_importances_)
```

```
Feature importances:
[2.93198341e-01 1.76688772e-01 5.26263220e-02 3.94192266e-02
 2.06923491e-02 2.36718210e-02 3.41561704e-02 2.91512988e-02
 1.66514505e-02 1.51702775e-02 7.70846055e-02 1.67615530e-02
 1.68249446e-02 1.44123935e-02 1.86056416e-02 4.31571801e-03
 3.79567594e-03 3.25695304e-03 2.51515059e-03 1.00267485e-03
 3.19495711e-04 5.59262436e-05 3.03305112e-03 4.14132289e-03
 3.50462562e-03 2.32425579e-03 1.45416746e-03 3.47782369e-04
 1.84069099e-04 0.00000000e+00 0.00000000e+00 3.46066004e-03
 2.07993297e-04 0.00000000e+00 3.01445220e-03 6.20341897e-04
 6.88301212e-04 3.90354871e-03 3.55363297e-03 1.03147330e-03
 2.04454184e-04 5.20646659e-05 5.47862516e-04 3.86901840e-03
 3.45895492e-03 1.46705644e-03 3.43132300e-04 3.04272422e-03
 1.83011964e-03 9.19555112e-04 1.18404048e-03 1.03327001e-04
 1.25062037e-03 7.25986511e-04 1.39058440e-04 2.80072132e-04
 1.18241671e-03 8.28314652e-04 3.90272349e-04 7.25299003e-04
 2.47001385e-03 8.31328596e-05 9.76013156e-04 5.65629657e-04
 2.41568471e-03 1.44996837e-03 1.62064534e-04 8.12946185e-04
 3.97845226e-04 9.59153411e-04 1.08854517e-03 2.39843971e-03
 8.55786520e-05 2.08591545e-03 3.27325704e-03 1.10073401e-03
 1.77687897e-03 8.28291976e-04 3.08642263e-04 8.70445761e-04
 4.94515854e-04 4.63651742e-04 6.43967025e-04 1.61374971e-03
 8.63252780e-04 1.82024529e-03 8.08312787e-04 5.31668627e-04
 1.11819348e-03 1.65844763e-03 1.28730625e-03 9.10389355e-04
 1.99100018e-03 1.44891540e-03 8.13668680e-04 1.32305004e-03
 1.03976226e-04 9.69569791e-04 6.67547919e-04 2.68821027e-04
 2.96584203e-04 8.79254380e-04 9.47837615e-04 4.26603845e-04
 7.91301790e-04 2.20310294e-03 2.23612617e-04 1.49856250e-03
 4.34590629e-04 2.00592803e-03 1.62457760e-03 2.42062352e-04
 7.67812332e-04 1.36805887e-04 6.69300951e-04 1.41410194e-03
 1.84950511e-04 2.05639852e-03 4.00327880e-03 1.29209408e-03
 1.88802653e-03 9.51939963e-04 1.59529822e-04 7.77978191e-04
 6.53481607e-04 6.99108298e-04 7.07091004e-04 1.64654654e-03
 1.08231735e-03 1.60082742e-03 9.46278287e-04 1.82216156e-03
 9.37444071e-04 1.33364685e-03 2.13019957e-03 8.53431725e-04
 1.41100952e-03]
```

```
# 변수명을 Index로 만들고 feature_importances를 매칭해서 나열한 데이터프레임 만들기

feature_names = list(data.columns)   # 변수명(컬럼명)을 리스트 형태로 만들기
dft = pd.DataFrame(np.round(best_clf.feature_importances_, 3), index=feature_names,
                   columns=['Feature_importances'])
dft1 = dft.sort_values(by='Feature_importances', ascending=False)
dft1                                  # 컬럼 Feature_importances의 값들을 내림차순으로 정리
```

	Feature_importances
FullTimeHomeTeamGoals	0.293
FullTimeAwayTeamGoals	0.177
HalfTimeResult_B	0.077
HalfTimeHomeTeamGoals	0.053
HalfTimeAwayTeamGoals	0.039
HomeTeamShotsOnTarget	0.034
AwayTeamShotsOnTarget	0.029
AwayTeamShots	0.024
HomeTeamShots	0.021

홈 팀이 승리하는데 중요한 변수로는 Full Time Home Team Goals가 중요하고 다음으로는 Full Time Away Team Goals가 중요하고 결정 트리와 달리 Half Time Result_B도 중요하다.

<성능 평가>

1. SVM 모델 : 0.99978
2. Gradient Boosting 모델 : 0.99955
3. 결정 트리 모델 : 0.99951
4. Randomforest 모델 : .098842
5. 표준화 로지스틱 회귀 모델 : 0.91802
6. 로지스틱 회귀 모델 : 0.90688
7. K-NN 모델 : 0.79728
8. 표준화 Ridge 모델 : 0.64886
9. Ridge 모델 : 0.64862
10. Linear Regression 모델 : 0.64127

19. XGB, LGB 모델

<XGB 모델>

```
[154] # 기본 XGBRegressor 모델
      from sklearn.metrics import r2_score

      xgb = XGBRegressor(random_state=0)
      xgb.fit(X_train, y_train)
      pred = xgb.predict(X_test)

      print('r2: {0:.5f}'.format(r2_score(y_test, pred)))

  r2: 0.99821
```

```
[155] # 그리드 서치 실행
      from sklearn.model_selection import GridSearchCV

      xgb = XGBRegressor()

      parameters = {'colsample_bytree': [0.7],
                    'learning_rate': [0.05],
                    'max_depth': [15],
                    'min_child_weight': [4],
                    'n_estimators': [1000],
                    'subsample': [0.8, 0.9]
                    }

      xgb_grid = GridSearchCV(xgb,
                              parameters,
                              scoring = 'r2',
                              cv = 3,
                              n_jobs = -1,
                              verbose=True)
      xgb_grid.fit(X_train, y_train)

  Fitting 3 folds for each of 2 candidates, totalling 6 fits
```

GridSearchCV
- estimator: XGBRegressor
  - XGBRegressor

```
[156] print('GridSearchCV 최적 파라미터:', xgb_grid.best_params_)

  GridSearchCV 최적 파라미터: {'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 15, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.8}
```

```
      model = xgb_grid.best_estimator_
      pred = model.predict(X_test)

      print('r2: {0:.5f}'.format(r2_score(y_test, pred)))

  r2: 0.99494
```

<LGB 모델>

```
# 기본 LGBMRegressor 모델
from lightgbm import LGBMRegressor
from sklearn.metrics import r2_score

lgb = LGBMRegressor(random_state=0)
lgb.fit(X_train, y_train)
pred = lgb.predict(X_test)

print('r2: {0:.5f}'.format(r2_score(y_test, pred)))
```

```
r2: 0.99807
```

우리의 본래 타겟 변수(FullTimeResult)는 H, D, A로 숫자로 구성 되어있지 않기 때문에 XGB 모델과 LGB 모델은 적합하지 않다.

<성능 평가>

1. SVM 모델 : 0.99978
2. Gradient Boosting 모델 : 0.99955
3. 결정 트리 모델 : 0.99951
4. Randomforest 모델 : .098842
5. 표준화 로지스틱 회귀 모델 : 0.91802
6. 로지스틱 회귀 모델 : 0.90688
7. K-NN 모델 : 0.79728
8. 표준화 Ridge 모델 : 0.64886
9. Ridge 모델 : 0.64862
10. Linear Regression 모델 : 0.64127

20. 최종 보고 - 1

SVM 모델이 축구 경기 데이터셋에서 가장 적합한 모델로 나온 근거

- **비선형 관계를 잘 학습: 커널 트릭을 사용하여 비선형 패턴을 잘 포착.**
- **고차원 데이터 처리: 고차원 데이터에서도 효과적으로 작동.**
- **마진 극대화: 일반화 성능을 높이는 최적의 결정 경계 설정.**
- **규제 능력: 적절한 정규화 파라미터로 과적합 방지.**

20. 최종 보고 - 2

순위	모델명	표준화	정확도
1	SVM 모델	O	0.99978
2	Gradient Boosting 모델	X	0.99955
3	결정 트리 모델	X	0.99951
4	Random Forest 모델	X	0.98842
5	로지스틱 회귀 모델	O	0.91802
6	로지스틱 회귀 모델	X	0.90688
7	K-NN 모델	O	0.79728
8	Ridge 모델	O	0.64886
9	Ridge 모델	X	0.64862
10	Linear Regression 모델	X	0.64127