빅데이터 분석 프로젝트

신일호

목차

목차

데이터 셋

HeartDisea	BMI	Smoking	AlcoholDri	Stroke	PhysicalHe	MentalHea	DiffWalkin	Sex	AgeCateg	Race	Diabetic	PhysicalAc	GenHealth	SleepTime	Asthma	KidneyDise	SkinCancer
No	16.6	Yes	No	No	3	30	No	Female	55-59	White	Yes	Yes	Very good	5	Yes	No	Yes
No	20.34	No	No	Yes	0	0	No	Female	80 or olde	White	No	Yes	Very good	7	No	No	No
No	26.58	Yes	No	No	20	30	No	Male	65-69	White	Yes	Yes	Fair	8	Yes	No	No
No	24.21	No	No	No	0	0	No	Female	75-79	White	No	No	Good	6	No	No	Yes
No	23.71	No	No	No	28	0	Yes	Female	40-44	White	No	Yes	Very good	8	No	No	No
Yes	28.87	Yes	No	No	6	0	Yes	Female	75-79	Black	No	No	Fair	12	No	No	No
No	21.63	No	No	No	15	0	No	Female	70-74	White	No	Yes	Fair	4	Yes	No	Yes
No	31.64	Yes	No	No	5	0	Yes	Female	80 or olde	White	Yes	No	Good	9	Yes	No	No
No	26.45	No	No	No	0	0	No	Female	80 or olde	White	No, borde	No	Fair	5	No	Yes	No
No	40.69	No	No	No	0	0	Yes	Male	65-69	White	No	Yes	Good	10	No	No	No
Yes	34.3	Yes	No	No	30	0	Yes	Male	60-64	White	Yes	No	Poor	15	Yes	No	No
No	28.71	Yes	No	No	0	0	No	Female	55-59	White	No	Yes	Very good	5	No	No	No
No	28.37	Yes	No	No	0	0	Yes	Male	75-79	White	Yes	Yes	Very good	8	No	No	No
No	28.15	No	No	No	7	0	Yes	Female	80 or olde	White	No	No	Good	7	No	No	No
No	29.29	Yes	No	No	0	30	Yes	Female	60-64	White	No	No	Good	5	No	No	No
No	29.18	No	No	No	1	0	No	Female	50-54	White	No	Yes	Very good	6	No	No	No
No	26.26	No	No	No	5	2	No	Female	70-74	White	No	No	Very good	10	No	No	No
No	22.59	Yes	No	No	0	30	Yes	Male	70-74	White	No, borde	Yes	Good	8	No	No	No
No	29.86	Yes	No	No	0	0	Yes	Female	75-79	Black	Yes	No	Fair	5	No	Yes	No
No	18.13	No	No	No	0	0	No	Male	80 or olde	White	No	Yes	Excellent	8	No	No	Yes
No	21.16	No	No	No	0	0	No	Female	80 or olde	Black	No, borde	No	Good	8	No	No	No
No	28.9	No	No	No	2	5	No	Female	70-74	White	Yes	No	Very good	7	No	No	No
No	26.17	Yes	No	No	0	15	No	Female	45-49	White	No	Yes	Very good	6	No	No	No

데이터 셋

심장병의 주요 지표

1. HeartDisease: 심장질환 또는 심근 경색 유무
2. BMI: 체질량 지수
3. Smoking: 평생동안 100개 이상의 담배를 피운 적이 있는지 유무
4. AlcoholDrinking: 주당 14잔 이상을 섭취하는 성인 남성 또는 7잔 이상을 섭취하는 성인 여성
5. Stroke: 뇌졸중 유무
6. PhysicalHealth: 최근 30일 중 신체 건강이 좋지 않은 날의 수
7. MentalHealth: 최근 30일 중 정신 건강이 좋지 않은 날의 수
8. DiffWalking: 걷기, 계단 오르기의 불편함 유무
9. Sex: 성별
10. AgeCategory: 14단계로 나눈 연령 범위
11. Race: 인종
12. Diabetic: 당뇨병 유무
13. PhysicalActivity: 최근 30일 동안 운동을 했거나 체육활동을 했는지 유무
14. GenHealth: 일반적인 건강 상태
15. SleepTime: 하루 24시간 기준으로 평균적인 수면 시간
16. Asthma: 천식 유무
17. KidneyDisease: 신장병 유무
18. SkinCancer: 피부암 유무

319795개의 데이터, 18개의 변수

ID변수 체크

```
df.info() #ID변수가 존재하지 않음
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   HeartDisease      319795 non-null  object 
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object 
 3   AlcoholDrinking   319795 non-null  object 
 4   Stroke            319795 non-null  object 
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object 
 8   Sex               319795 non-null  object 
 9   AgeCategory       319795 non-null  object 
 10  Race              319795 non-null  object 
 11  Diabetic          319795 non-null  object 
 12  PhysicalActivity  319795 non-null  object 
 13  GenHealth         319795 non-null  object 
 14  SleepTime         319795 non-null  float64
 15  Asthma            319795 non-null  object 
 16  KidneyDisease     319795 non-null  object 
 17  SkinCancer        319795 non-null  object 
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

고유하게 데이터를 식별할 수 있는 **ID변수가 존재하지 않는다.**
데이터의 인덱스를 ID로 만들 수도 있지만, 필요 없기에 만들지 않음

ID변수가 존재하지 않아 아래의 코드로 **중복 데이터를 삭제할 수 있으나** 범주형 변수가 많은 데이터 셋의 특성상 충분히 **중복된 응답을 한 데이터가 많을 수 있다고 판단**하여 **실행하지 않음**

```
#ID변수의 결측값 및 중복을 체크하는 대신 중복된 레코드(행)을 체크
df.duplicated().sum()
#중복된 레코드를 삭제
df.drop_duplicates(inplace=True)
```

타겟변수 생성 및 비율 점검

```
#타겟변수 HeartDisease, 변수의 값이 Yes or No이므로 object타입
df['HeartDisease'].dtype
```

```
dtype('O')
```

```
#결측값 확인 -> 없음
df['HeartDisease'].isnull().sum()
```

```
0
```

```
#타겟변수 개수 확인
df['HeartDisease'].value_counts(dropna=False)
```

```
No      292422
Yes      27373
Name: HeartDisease, dtype: int64
```

```
#타겟변수 비율 확인
df['HeartDisease'].value_counts(dropna=False, normalize=True)
```

```
No      0.914405
Yes     0.085595
Name: HeartDisease, dtype: float64
```

타겟변수 '**HeartDisease**'에 대한 정보

- HeartDisease는 값이 **Yes, No**인 **범주형 변수**이기 때문에 object 데이터 타입
- **결측값 없음**
- 292,422개의 No, 27,373개 Yes
- 약 **91%**의 **No**, 약 **9%**의 **Yes**

타겟변수 생성 및 비율 점검

```
#Yes or No의 값을 가진 타겟변수를 1과 0의 값으로 인코딩
df['HeartDisease_encoded'] = OrdinalEncoder().fit_transform(df['HeartDisease'].values.reshape(-1,1))
df.groupby(['HeartDisease', 'HeartDisease_encoded']).size()
```

```
HeartDisease  HeartDisease_encoded
No            0.0                    292422
Yes           1.0                     27373
dtype: int64
```

'HeartDisease'는 Yes, No로 된 String타입의 범주형 변수이므로 숫자인 0과 1로 인코딩

편의를 위해 인코딩 된 'HeartDisease_encoded'를 다시 'HeartDisease'로 변경 후 열의 첫 번째로 이동

```
#HeartDisease_encoded 변수가 새로 생김
df.columns
```

```
Index(['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime',
       'Asthma', 'KidneyDisease', 'SkinCancer', 'HeartDisease_encoded'],
      dtype='object')
```

```
#편의를 위해 타겟 변수의 이름을 HeartDisease_encoded에서 HeartDisease로 바꾸고 맨 앞의 열로 가져옴
df.drop(['HeartDisease'], axis=1, inplace=True)
df.rename(columns={'HeartDisease_encoded': 'HeartDisease'}, inplace=True)
df = df.reindex(columns=['HeartDisease'] + list(df.columns.drop('HeartDisease')))
```

기타 데이터 처리

```
#구간변수 저장
cols_i = ['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']
df_i = df[cols_i]
df_i.dtypes
```

```
BMI               float64
PhysicalHealth    float64
MentalHealth      float64
SleepTime         float64
dtype: object
```

```
#구간변수 결측값 확인 -> 없음
df_i.isnull().sum()
```

```
BMI               0
PhysicalHealth    0
MentalHealth      0
SleepTime         0
dtype: int64
```

구간변수는 총 4개

1. BMI
2. PhysicalHealth
3. MentalHealth
4. SleepTime

4개 모두 float 데이터 타입이며 **결측값이 존재하지 않음**

기타 데이터 처리

```
#범주형 변수 저장
cols_c = ['Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Sex', 'AgeCategory', 'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'Asthma', 'KidneyDisease', 'SkinCancer']
df_c = df[cols_c]
df_c.dtypes
```

```
Smoking             object
AlcoholDrinking     object
Stroke              object
DiffWalking         object
Sex                 object
AgeCategory         object
Race                object
Diabetic            object
PhysicalActivity    object
GenHealth           object
Asthma              object
KidneyDisease       object
SkinCancer          object
dtype: object
```

```
#범주형 변수 결측값 확인 -> 없음
df_c.isnull().sum()
```

```
Smoking             0
AlcoholDrinking     0
Stroke              0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

범주형 변수는 총 13개

1. Smoking
2. AlcoholDrinking
3. Stroke
4. DiffWalking
5. Sex
6. AgeCategory
7. Race
8. Diabetic
9. PhysicalActivity
10. GenHealth
11. Asthma
12. KidneyDisease
13. SkinCancer

13개 모두 object 데이터 타입이며
결측값이 존재하지 않음

```
#결측값 확인 -> 없음
df.isnull().sum()
```

```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

```
#모든 열 결측값 확인 -> 없음
df.isna().any()[lambda x: x]
```

```
Series([], dtype: bool)
```

데이터에 결측값이 존재하지 않음

구간변수 요약 통계 및 검토

```
#구간변수 요약 통계
df_i.describe().round(2)
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.00	319795.00	319795.00	319795.00
mean	28.33	3.37	3.90	7.10
std	6.36	7.95	7.96	1.44
min	12.02	0.00	0.00	1.00
25%	24.03	0.00	0.00	6.00
50%	27.34	0.00	0.00	7.00
75%	31.42	2.00	3.00	8.00
max	94.85	30.00	30.00	24.00

```
#왜도 -> +-3 범위 안에 있으므로 정규분포에서 많이 벗어나지 않음
df_i.skew()
```

```
BMI               1.332431
PhysicalHealth    2.603973
MentalHealth      2.331112
SleepTime         0.679035
dtype: float64
```

```
#첨도 -> +-10 범위 안에 있으므로 정규분포에서 많이 벗어나지 않음
df_i.kurtosis()
```

```
BMI               3.890043
PhysicalHealth    5.528450
MentalHealth      4.403937
SleepTime         7.854869
dtype: float64
```

4개의 구간변수에 대한 요약 통계
- count: 데이터의 수
- mean: 평균값
- std: 표준편차
- min:. 최솟값
- 25%: 1사분위수
- 50%: 2사분위수(중앙값)
- 75%: 3사분위수
- max: 최댓값

- **skew()**를 이용해 **왜도**를 계산
  -> 모든 구간 변수의 왜도가 **±3의 범위 안**에 있으므로 **정규분포에서 많이 벗어나지 않음**

- **kurtosis()**를 이용해 **첨도**를 계산
  -> 모든 구간 변수의 첨도가 **±10의 범위 안**에 있으므로 **정규분포에서 많이 벗어나지 않음**

이상값 제거

```
#3.0 IQR을 적용
Q1 = df[['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']].quantile(0.25)
Q3 = df[['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']].quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
BMI               7.39
PhysicalHealth    2.00
MentalHealth      3.00
SleepTime         2.00
dtype: float64
```

```
Lower = Q1-3.0*IQR
Lower
```

```
BMI               1.86
PhysicalHealth   -6.00
MentalHealth     -9.00
SleepTime         0.00
dtype: float64
```

```
Upper = Q3+3.0*IQR
Upper
```

```
BMI              53.59
PhysicalHealth    8.00
MentalHealth     12.00
SleepTime        14.00
dtype: float64
```

```
#Lower보다 낮은 값은 존재하지 않지만 Upper보다 높은 값은 존재
df[cols_i].describe()
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.000000	319795.00000	319795.000000	319795.000000
mean	28.325399	3.37171	3.898366	7.097075
std	6.356100	7.95085	7.955235	1.436007
min	12.020000	0.00000	0.000000	1.000000
25%	24.030000	0.00000	0.000000	6.000000
50%	27.340000	0.00000	0.000000	7.000000
75%	31.420000	2.00000	3.000000	8.000000
max	94.850000	30.00000	30.000000	24.000000

```
#Upper보다 높은 이상값들을 제거하고 df1으로 저장, 이상값 제거 후 데이터 수 319795->253116
c1 = df['BMI'] <= 53.59
c2 = df['PhysicalHealth'] <= 8.00
c3 = df['MentalHealth'] <= 12.00
c4 = df['SleepTime'] <= 14.00
df1 = df[c1 & c2 & c3 & c4]
df1.shape
```

```
(253116, 18)
```

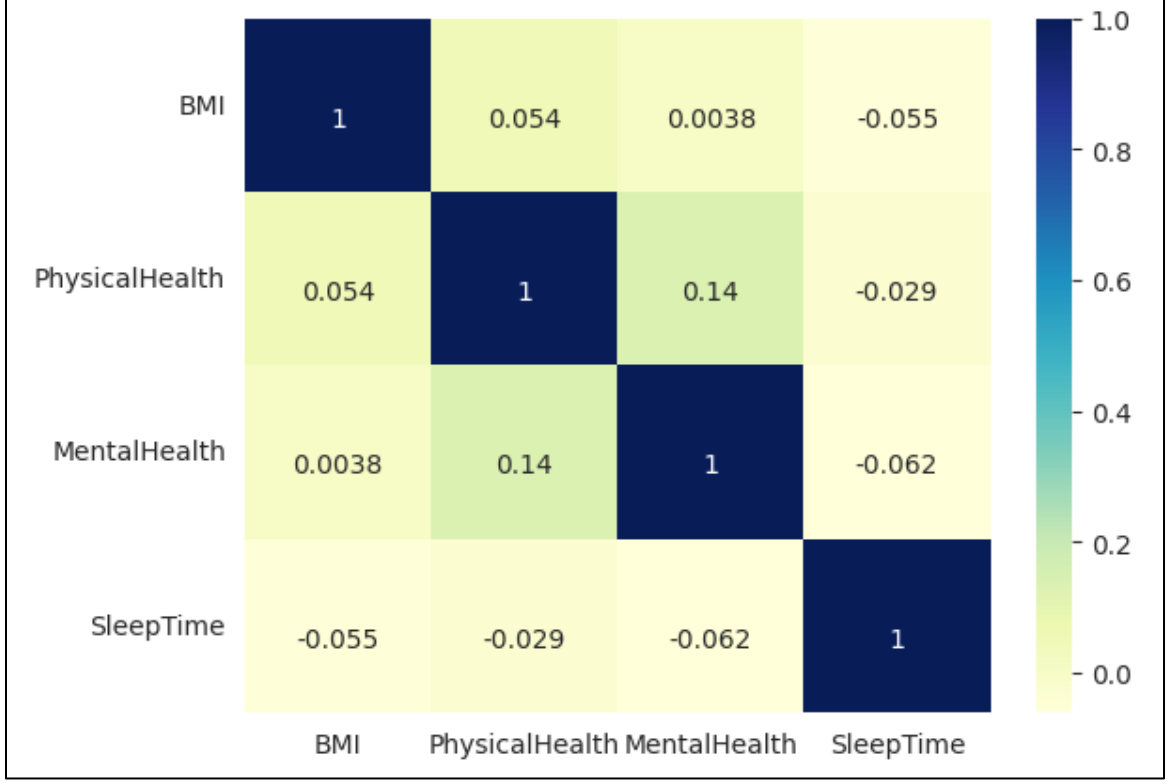
구간변수의 **IQR**(3사분위수-1사분위수)을 구하고 **3.0IQR**을 적용하여 Lower값과 Upper의 값을 구함

구간변수 요약 통계를 보면 각 변수들에게 **Lower보다 낮은 값은 존재하지 않지만 Upper보다 높은 값들은 존재**하므로 **Upper보다 높은 값(이상값)들을 제거**하고 새로운 데이터 프레임에 저장

이상값 제거 후 데이터의 수
319,795개 -> 253,116개

상관관계 검토

```
corr = df1[cols_i].corr()
annot_kws = {"ha": 'center',"va": 'top'}
sns.heatmap(data=corr, annot=True, annot_kws=annot_kws, cmap="YlGnBu");
```




```
#구간변수 간 상관계수 -> 모두 0.7보다 낮으므로 문제가 없음
round(df1[cols_i].corr(), 2)
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
BMI	1.00	0.05	0.00	-0.05
PhysicalHealth	0.05	1.00	0.14	-0.03
MentalHealth	0.00	0.14	1.00	-0.06
SleepTime	-0.05	-0.03	-0.06	1.00

구간변수 간 상관계수 **모두 0.7보다 낮고 0에 가까움**
-> **선형적 관계가 존재하지 않음**
-> 변수 제거 필요성 없음

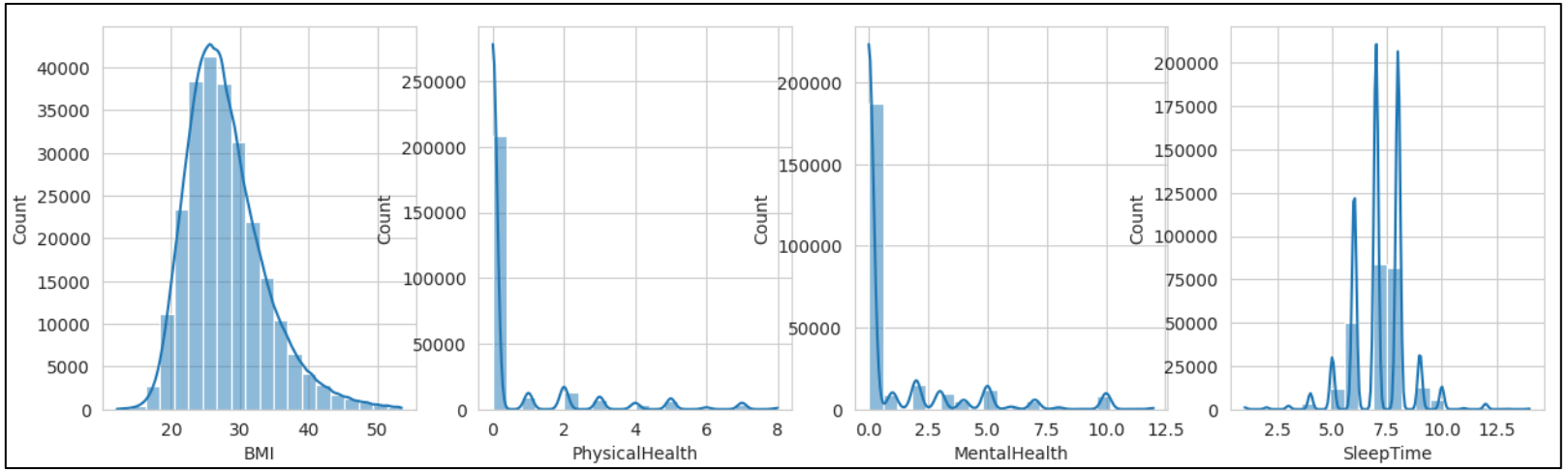
T-검정

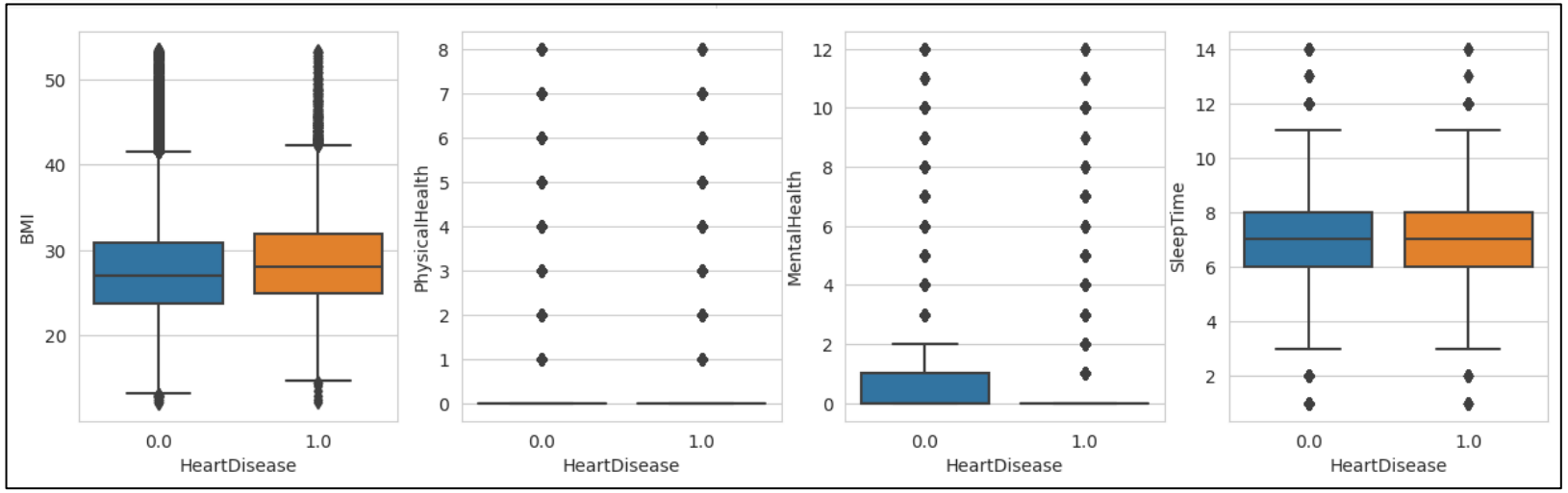
```python
#구간변수 cols_i = ['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']
#모든 구간변수의 pvalue < 0.05이므로 두 그룹의 평균이 같다는 귀무가설을 기각
for variable in cols_i:
    data_1 = df1[df1['HeartDisease'] == 1][variable]
    data_0 = df1[df1['HeartDisease'] == 0][variable]
    print(f"{variable}: {stats.ttest_ind(data_1, data_0)}")
    print("------")
```

```
BMI: Ttest_indResult(statistic=21.52080104216 86, pvalue=1.2297378314638942e-102)
------
PhysicalHealth: Ttest_indResult(statistic=27.3628570433716, pvalue=1.3214816107819437e-164)
------
MentalHealth: Ttest_indResult(statistic=-20.428638987106268, pvalue=1.1059745755458137e-92)
------
SleepTime: Ttest_indResult(statistic=11.1589269114179, pvalue=6.57839597284694e-29)
------
```

모든 구간변수의 pvalue < 0.05
-> 두 그룹의 평균이 같다는 **귀무가설 기각**
-> 두 그룹의 평균은 **다르다**

구간변수 시각화





Histplot
: PhysicalHealth와 MentalHealth가 0에 가까운 값이 많지만 대체로 구간 변수들이 정규분포를 많이 벗어나지 않음

Boxplot
: 대체로 BMI가 높은 사람이 HeartDisease의 경향을 조금 더 보이는 것 같고 PhysicalHealth와 MentalHealth는 0에 가까운 값이 많아 추론하기 어려워 보인다. 그리고 SleepTime과 HeartDisease는 큰 관련이 없어 보인다.

범주형 변수 도수분포표 검토

```
df1[cols_c].describe()
```

	Smoking	AlcoholDrinking	Stroke	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealth	Asthma	KidneyDisease	SkinCancer
count	253116	253116	253116	253116	253116	253116	253116	253116	253116	253116	253116	253116	253116
unique	2	2	2	2	2	13	6	4	2	5	2	2	2
top	No	No	No	No	Female	65-69	White	No	Yes	Very good	No	No	No
freq	155985	236479	246001	232157	127724	27475	195176	218434	206179	100412	224837	246231	229741

범주형 변수들의 요약 통계

- count
  : 총 데이터의 수
- unique
  : 몇 개의 범주인가, 값의 종류의 수, 고유한 값의 개수
- top
  : 가장 빈도가 높은 범주
- freq
  : 가장 빈도가 높은 범주의 개수, top의 개수, 가장 자주 등장하는 범주의 수

대체적으로 Smoking, Sex, AgeCategory, GenHealth를 제외한 나머지 범주형 변수들이 한 쪽으로 치우친 값을 갖는 것을 볼 수 있다.

범주형 변수 도수분포표 검토

```
#Sex
df1['Sex'].value_counts(dropna=False)
```

```
Female    127724
Male      125392
Name: Sex, dtype: int64
```

```
pd.crosstab(df1['Sex'], columns='count')
```

col_0	count
Sex	
Female	127724
Male	125392

```
pd.crosstab(df1['Sex'], columns='ratio', normalize=True)
```

col_0	ratio
Sex	
Female	0.504607
Male	0.495393

변수 Sex의 값인 Female, Male의 수와 비율, 그리고 타겟변수인 HeartDisease와 매칭되는 수와 비율을 나타냄

```
pd.crosstab(df1['Sex'], df1['HeartDisease'])
```

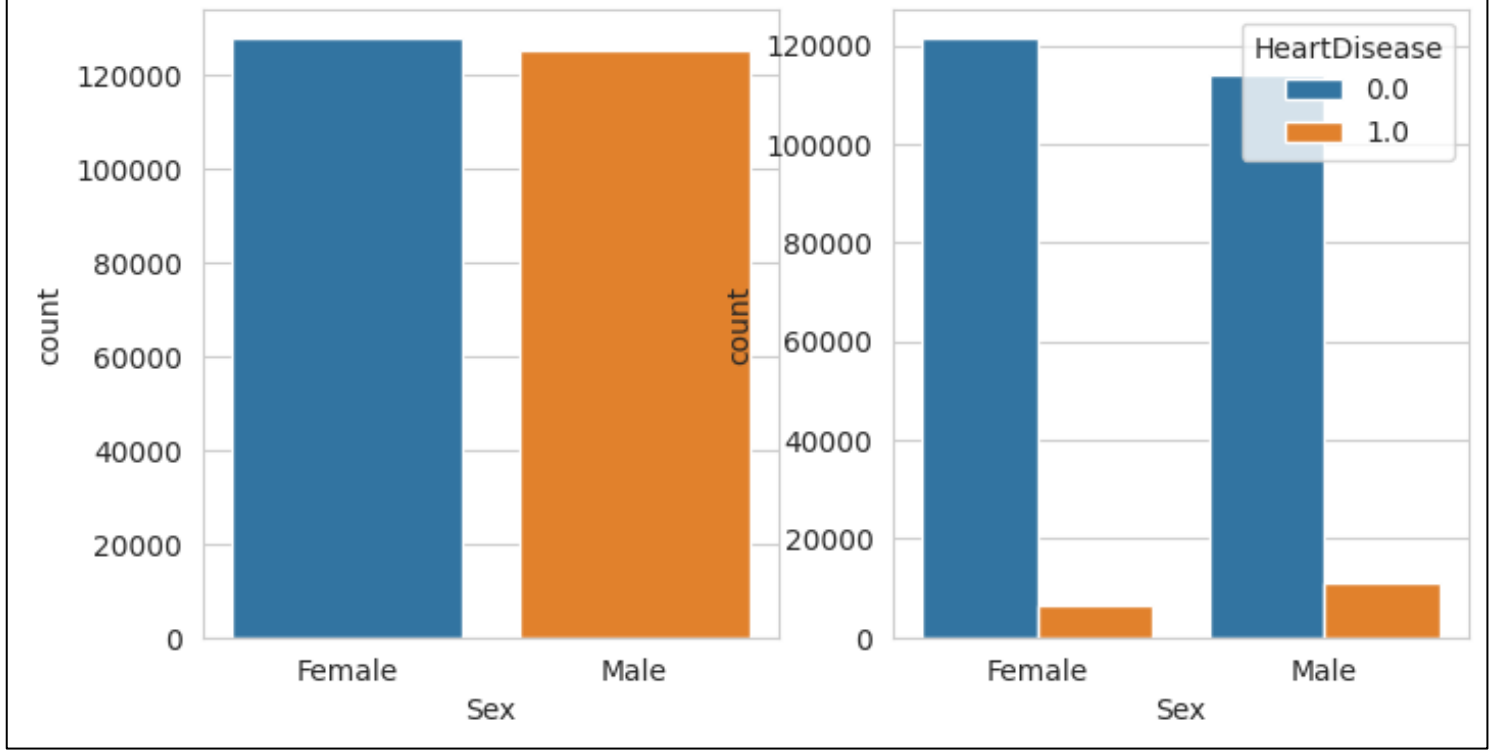
HeartDisease	0.0	1.0
Sex		
Female	121339	6385
Male	114192	11200

```
pd.crosstab(df1['Sex'], df1['HeartDisease'], normalize=True)
```

HeartDisease	0.0	1.0
Sex		
Female	0.479381	0.025226
Male	0.451145	0.044248

범주형 변수 시각화

```
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
sns.countplot(ax=axes[0], x='Sex', data=df1)
sns.countplot(ax=axes[1], x='Sex', hue='HeartDisease', data=df1);
plt.show()
```



앞에서 검토한 Sex변수에 대한 countplot

Female과 Male의 수는 다른 변수들에 비해 상대적으로 한 쪽으로 치우치지 않고 비슷함

우측의 countplot을 보면 Male의 경우 Female보다 HeartDisease가 조금 더 많이 나타나는 경향을 보인다.

인코딩

```python
#범주형 변수들을 인코딩
df2 = df1.copy()

for variable in cols_c:
    print("-----")
    encoded_variable = variable + '_encoded'
    df2[encoded_variable] = OrdinalEncoder().fit_transform(df2[variable].values.reshape(-1, 1))
    print(df2.groupby([variable, encoded_variable]).size())
```

```
-----
Smoking  Smoking_encoded
No       0.0               155985
Yes      1.0                97131
dtype: int64
-----
AlcoholDrinking  AlcoholDrinking_encoded
No               0.0                       236479
Yes              1.0                        16637
dtype: int64
-----
Stroke  Stroke_encoded
No      0.0              246001
Yes     1.0                7115
dtype: int64
-----
```

새로운 데이터프레임(df2)에 이상값을 제거한 데이터프레임(df1)을 복사

범주형 변수들이 모두 문자열로 되어있어 인코딩 실행
해당 코드를 실행하면 범주형 변수들이 들어있는 cols_c를 이용해 모든 범주형 변수들을 인코딩하고 결과를 출력

좌측 코드에는 출력 결과가 길어서 중도 생략

인코딩

```
df2.columns
```

```
Index(['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime',
       'Asthma', 'KidneyDisease', 'SkinCancer', 'Smoking_encoded',
       'AlcoholDrinking_encoded', 'Stroke_encoded', 'DiffWalking_encoded',
       'Sex_encoded', 'AgeCategory_encoded', 'Race_encoded',
       'Diabetic_encoded', 'PhysicalActivity_encoded', 'GenHealth_encoded',
       'Asthma_encoded', 'KidneyDisease_encoded', 'SkinCancer_encoded'],
      dtype='object')
```

```
df2.shape
```

```
(253116, 31)
```

```
#인코딩 되기 전의 범주형 변수들을 제거
for variable in cols_c:
    df2.drop(variable, axis=1, inplace=True)
df2.columns
```

```
Index(['HeartDisease', 'BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime',
       'Smoking_encoded', 'AlcoholDrinking_encoded', 'Stroke_encoded',
       'DiffWalking_encoded', 'Sex_encoded', 'AgeCategory_encoded',
       'Race_encoded', 'Diabetic_encoded', 'PhysicalActivity_encoded',
       'GenHealth_encoded', 'Asthma_encoded', 'KidneyDisease_encoded',
       'SkinCancer_encoded'],
      dtype='object')
```

```
#df2.to_csv('/content/drive/MyDrive/Colab Notebooks/project/HA_2071506_project.csv', index=False)
```

인코딩 후 df2의 열

인코딩 전의 범주형 변수들을 제거

인코딩 완료한 데이터프레임 df2를 새로운 csv파일로 저장

타겟변수 비율 점검 및 언더샘플링

```
df2.shape
```

```
(253116, 18)
```

```
data = df2.drop(['HeartDisease'], axis=1)   # 타겟변수를 제외한 변수만 data에 저장
target = df2['HeartDisease']                # 타겟변수만 target에 저장
data.shape
```

```
(253116, 17)
```

```
target.shape
```

```
(253116,)
```

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

```
0.0    292422
1.0     27373
Name: HeartDisease, dtype: int64
```

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

```
0.0    0.914405
1.0    0.085595
Name: HeartDisease, dtype: float64
```

타겟 변수 HeartDisease를 target, 나머지 예측 변수들을 data에 저장 후 타겟 변수의 클래스 별 개수 및 비율 확인 결과
0의 비율이 지나치게 높음

```
from imblearn.under_sampling import RandomUnderSampler
undersample = RandomUnderSampler(sampling_strategy=0.333, random_state=1)
data_under, target_under = undersample.fit_resample(data, target)
```

```
target_under.value_counts(dropna=True)
```

```
0.0    52807
1.0    17585
Name: HeartDisease, dtype: int64
```

```
target_under.value_counts(dropna=True, normalize=True).round(2)
```

```
0.0    0.75
1.0    0.25
Name: HeartDisease, dtype: float64
```

0과 1의 비율을 약 3:1로 맞춰 언더샘플링 진행

타겟변수 비율 점검 및 언더샘플링

```
# 50:50 비율로 데이터 분할
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_under, target_under, test_size=0.5, random_state=1, stratify = target_under)

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

```
X_train shape: (35196, 17)
X_test shape: (35196, 17)
```

```
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
y_train shape: (35196,)
y_test shape: (35196,)
```

```
y_train.value_counts(dropna=True, normalize=True)
```

```
0.0    0.750199
1.0    0.249801
Name: HeartDisease, dtype: float64
```

```
y_train.value_counts(dropna=True)
```

```
0.0    26404
1.0     8792
Name: HeartDisease, dtype: int64
```

언더샘플링된 데이터들로 train, test dataset을 5:5로 분할

트리모델

```
# 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=1, max_depth=5)

params = {'criterion':['gini','entropy'],'max_depth': range(1,21)} # 2 X 20 = 40

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:0.79268
GridSearchCV best parameter: {'criterion': 'gini', 'max_depth': 8}
```

```
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:0.78915
```

```
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:0.81899
```

Gini와 Entropy, 그리고 1~20까지의 depth범위로 train date set을 Grid Search진행한 결과
-> best parameter는 **gini**와 max depth = **8**이다.
-> 가장 높은 accuracy는 약 0.79이다.

이 best parameter로 test data를 사용해 검증한 결과
-> **accuracy**가 약 **0.79**
-> **ROC AUC**가 약 **0.82**

트리모델

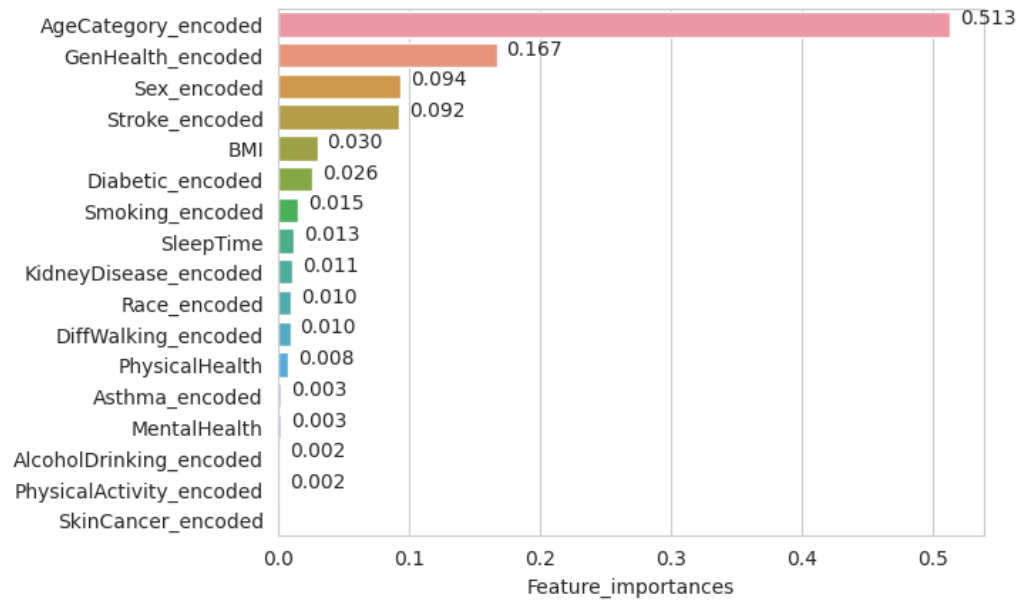
```
feature_names = list(data.columns)   # 변수명(컬럼명)을 리스트 형태로 만들기
dft = pd.DataFrame(np.round(best_clf.feature_importances_, 4), index=feature_names, columns=['Feature_importances'])
dft1 = dft.sort_values(by='Feature_importances', ascending=False)
dft1                                 # 컬럼 Feature_importances의 값들을 내림차순으로 정
```

	Feature_importances
AgeCategory_encoded	0.5132
GenHealth_encoded	0.1671
Sex_encoded	0.0936
Stroke_encoded	0.0925
BMI	0.0303
Diabetic_encoded	0.0264
Smoking_encoded	0.0155
SleepTime	0.0126
KidneyDisease_encoded	0.0114
Race_encoded	0.0100
DiffWalking_encoded	0.0097
PhysicalHealth	0.0081
Asthma_encoded	0.0030
MentalHealth	0.0027
AlcoholDrinking_encoded	0.0019
PhysicalActivity_encoded	0.0016
SkinCancer_encoded	0.0003

Feature_importances를 이용해 예측 변수들을 중요 순서대로 정리
-> **AgeCategory가 가장 중요**하고 다음은 GenHealth, Sex, Stroke 등
->반대로 아래에서 SkinCancer, PyhsicalActivity, AlcholDrinking 등은 중요하지 않다고 나온다.

```
ax = sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(), p.get_y()+1), xytext=(5, 10), textcoords='offset points')
```



더미변수 생성 및 데이터 표준화

```
cols_d = ['AgeCategory_encoded', 'Race_encoded', 'Diabetic_encoded', 'GenHealth_encoded']

df_d = pd.get_dummies(df2, columns=cols_d)
```

```
df_d.shape

(253116, 42)
```

범주형 변수 중에서 값으로 0과 1만 가지는 변수를 제외하고 더미변수 생성

```
cols_b = ['AgeCategory_encoded_0.0', 'Race_encoded_1.0', 'Diabetic_encoded_0.0', 'GenHealth_encoded_0.0']

df_d.drop(cols_b, axis=1, inplace=True)
df_d.shape

(253116, 38)
```

각 범주의 기준을 잡을 변수들을 cols_b에 저장 후 데이터프레임에서 제거함
기준은 다음과 같다

-AgeCategory: **제일 젊은 18-24세**
-Race: 우리가 소속된 **Asian**
-Diabetic: 당뇨에 걸리지 않은 **No**
-GenHealth: 가장 건강해 심장병과 **관련이 없을 것** 같은 **Excellent**

```
list(df_d.columns)

['HeartDisease',
 'BMI',
 'PhysicalHealth',
 'MentalHealth',
 'SleepTime',
 'Smoking_encoded',
 'AlcoholDrinking_encoded',
 'Stroke_encoded',
 'DiffWalking_encoded',
 'Sex_encoded',
 'PhysicalActivity_encoded',
 'Asthma_encoded',
 'KidneyDisease_encoded',
 'SkinCancer_encoded',
 'AgeCategory_encoded_0.0',
 'AgeCategory_encoded_1.0',
 'AgeCategory_encoded_2.0',
 'AgeCategory_encoded_3.0',
 'AgeCategory_encoded_4.0',
 'AgeCategory_encoded_5.0',
 'AgeCategory_encoded_6.0',
 'AgeCategory_encoded_7.0',
 'AgeCategory_encoded_8.0',
 'AgeCategory_encoded_9.0',
 'AgeCategory_encoded_10.0',
 'AgeCategory_encoded_11.0',
 'AgeCategory_encoded_12.0',
 'Race_encoded_0.0',
 'Race_encoded_1.0',
 'Race_encoded_2.0',
 'Race_encoded_3.0',
 'Race_encoded_4.0',
 'Race_encoded_5.0',
 'Diabetic_encoded_0.0',
 'Diabetic_encoded_1.0',
 'Diabetic_encoded_2.0',
 'Diabetic_encoded_3.0',
 'GenHealth_encoded_0.0',
 'GenHealth_encoded_1.0',
 'GenHealth_encoded_2.0',
 'GenHealth_encoded_3.0',
 'GenHealth_encoded_4.0']
```

더미변수 생성 및 데이터 표준화

```
# 구간 변수들만 별도로 모아 데이터프레임 df_num을 만든다.
numeric_cols = ['BMI','PhysicalHealth','MentalHealth', 'SleepTime']
df_num = df_d[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 변수
df_num_standard.columns = df_num.columns
df_num_standard.head()
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
0	-1.326940	-0.383279	-0.468416	-0.124916
1	-0.649325	-0.383279	-0.468416	-0.934872
2	0.166615	3.784651	-0.468416	3.924865
3	0.651626	3.089996	-0.468416	1.494996
4	-0.257113	-0.383279	-0.468416	-1.744828

4개의 구간 변수들을 모아 변수들 간의 스케일 차이를 없애기 위해 StandardScaler를 이용해 표준화

```
df_cat = df_d.drop(numeric_cols, axis=1) # 원래 데이터프레임 df에서 구간변수들을 제거하여 df_cat에 저장

# 구간변수 스케일을 표준화한 df_num_standard와 범주형 변수만 담은 df_cat을 병합
# index문제로 바로 concat하면 데이터프레임에 결측값이 추가되는 에러가 발생하여 인덱스 초기화 후 concat수행
df_num_standard.reset_index(drop=True, inplace=True)
df_cat.reset_index(drop=True, inplace=True)
dfu_standard = pd.concat([df_num_standard, df_cat], axis=1)
dfu_standard.columns   # dfu의 변수명을 나열
```

```
Index(['BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime', 'HeartDisease',
       'Smoking_encoded', 'AlcoholDrinking_encoded', 'Stroke_encoded',
       'DiffWalking_encoded', 'Sex_encoded', 'PhysicalActivity_encoded',
       'Asthma_encoded', 'KidneyDisease_encoded', 'SkinCancer_encoded',
       'AgeCategory_encoded_1.0', 'AgeCategory_encoded_2.0',
       'AgeCategory_encoded_3.0', 'AgeCategory_encoded_4.0',
       'AgeCategory_encoded_5.0', 'AgeCategory_encoded_6.0',
       'AgeCategory_encoded_7.0', 'AgeCategory_encoded_8.0',
       'AgeCategory_encoded_9.0', 'AgeCategory_encoded_10.0',
       'AgeCategory_encoded_11.0', 'AgeCategory_encoded_12.0',
       'Race_encoded_0.0', 'Race_encoded_2.0', 'Race_encoded_3.0',
       'Race_encoded_4.0', 'Race_encoded_5.0', 'Diabetic_encoded_1.0',
       'Diabetic_encoded_2.0', 'Diabetic_encoded_3.0', 'GenHealth_encoded_1.0',
       'GenHealth_encoded_2.0', 'GenHealth_encoded_3.0',
       'GenHealth_encoded_4.0'],
      dtype='object')
```

로지스틱 회귀 모델

```
lr = LogisticRegression(solver='lbfgs',penalty=None,random_state=1,n_jobs=-1)

# 그리드 서치 실행
from sklearn.model_selection import GridSearchCV
params = {'solver':['lbfgs', 'saga'], 'penalty':[None]}

grid_lr = GridSearchCV(lr, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1,)
grid_lr.fit(X_train, y_train)

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

```
GridSearchCV max accuracy:0.79981
GridSearchCV best parameter: {'penalty': None, 'solver': 'saga'}
```

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

```
Accuracy on test set:0.79955
```

```
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:0.83926
```

2개의 solve인 lbfgs, saga를 정규화를 하지 않으며 train dataset으로 Grid Search를 진행함
-> best solver는 **saga**
-> 최고 accuracy는 약 0.8

Best parameter로 test dataset을 검증한 결과
-> **accuracy**가 **약 0.8**로 train과 거의 같음
-> **ROC AUC**는 **약 0.84**

아래는 Odds ratio를 시각화하는 코드

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

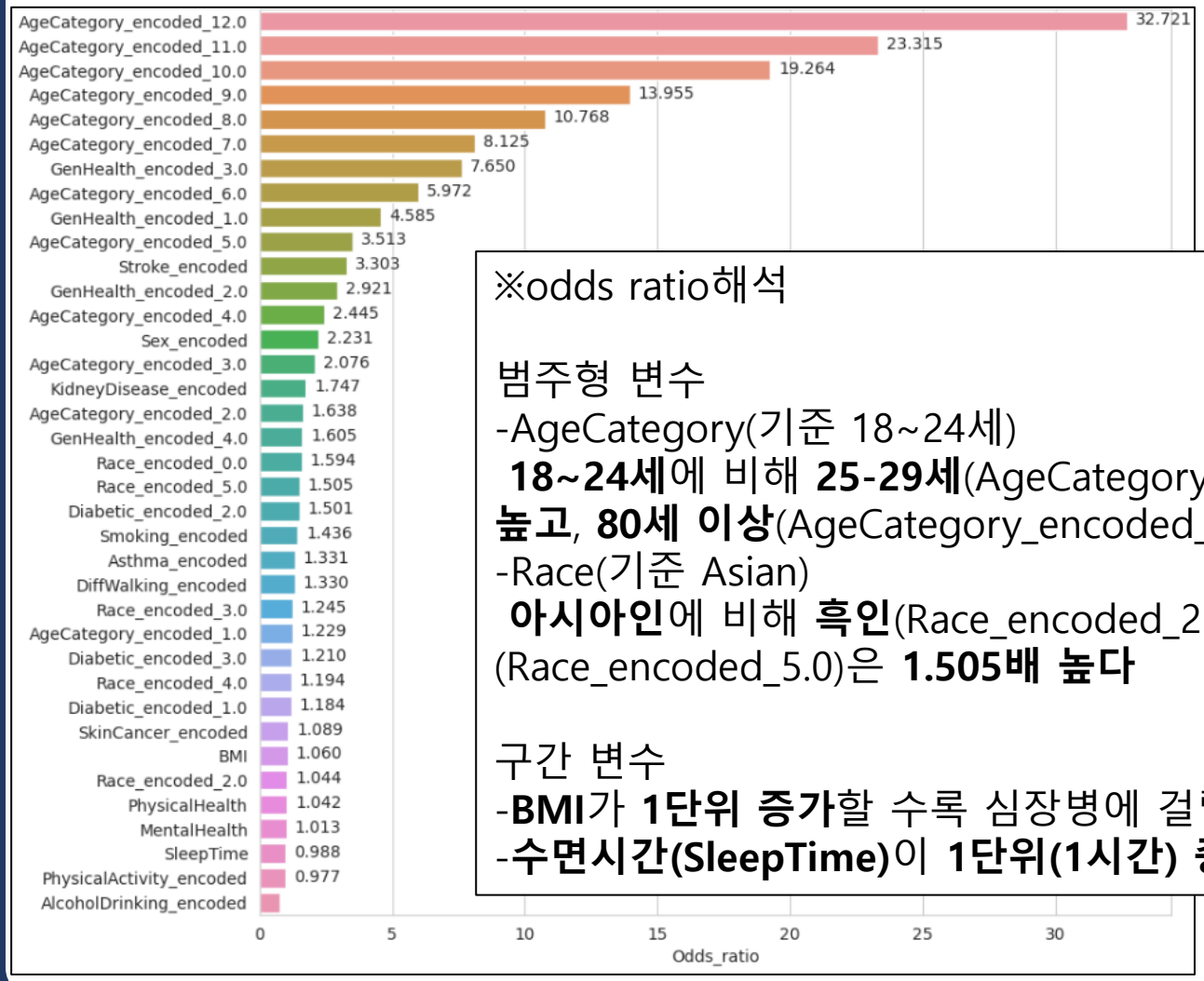
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

plt.figure(figsize=(10, 10))

ax = sns.barplot(y=dft1.index, x="Odds_ratio", data=dft1)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(), p.get_y()+1.2),
                xytext=(5, 10), textcoords='offset points')
```

로지스틱 회귀 모델



※odds ratio해석

범주형 변수
-AgeCategory(기준 18~24세)
18~24세에 비해 **25-29세**(AgeCategory_encoded_1.0)는 심장병이 걸릴 가능성이 **약 1.229배 높고**, **80세 이상**(AgeCategory_encoded_12.0)은 **32.721배 높다**
-Race(기준 Asian)
아시아인에 비해 **흑인**(Race_encoded_2.0)은 심장병이 걸릴 가능성이 **약 1.044배 높고**, **백인**(Race_encoded_5.0)은 **1.505배 높다**

구간 변수
-**BMI**가 **1단위 증가**할 수록 심장병에 걸릴 가능성이 **약 6% 높아진다**
-**수면시간(SleepTime)**이 **1단위(1시간) 증가**할 수록 심장병에 걸릴 가능성이 **1.2%씩 낮아진다**

신경망 모델

```
dfu_standard.shape
```

```
(253116, 38)
```

```
#데이터가 많아 Grid Search 시간이 너무 오래걸림
#10000개의 데이터만 사용
sample_df = dfu_standard.sample(n=10000, random_state=1)
sample_df.shape
```

```
(10000, 38)
```

```
sample_df['HeartDisease'].value_counts()
```

```
0.0    9321
1.0     679
Name: HeartDisease, dtype: int64
```

```
data = sample_df.drop(['HeartDisease'], axis=1)   # 타겟변수를 제외한 변수만 data에 저장
target = sample_df['HeartDisease']                # 타겟변수만 target에 저장

from imblearn.under_sampling import RandomUnderSampler    # RandomUnderSampler를 import
undersample = RandomUnderSampler(sampling_strategy=0.333, random_state=1)
                                                  # 타겟변수의 소수 클래스 및 다수 클래스를
                                                  # 1:3의 비율(=1/3)로 언더샘플링
data_under, target_under = undersample.fit_resample(data, target)
                                                  # data 및 target에 언더샘플링 적용
```

```
target_under.value_counts()
```

```
0.0    2039
1.0     679
Name: HeartDisease, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data_under, target_under, test_size=0.5, random_state=1, stratify=target_under)

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

```
X_train shape: (1359, 37)
X_test shape: (1359, 37)
```

신경망 모델에서 Grid Search하기에는 데이터의 양이 많아 **10,000개의 데이터**만 가지고 시작함

타겟 변수를 약 1:3의 비율로 언더샘플링

신경망 모델

```
clf_mlp = MLPClassifier(max_iter = 2000, random_state = 0)

from sklearn.model_selection import GridSearchCV
params = {'solver':['sgd', 'lbfgs', 'adam'],
          'alpha':[0.0001, 0.01, 1],
          'activation':['tanh','relu', 'logistic'],
          'hidden_layer_sizes': [(100,), (100,100)]
         }

grid_mlp = GridSearchCV(clf_mlp, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1)
grid_mlp.fit(X_train, y_train)

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

```
GridSearchCV max accuracy:0.81090
GridSearchCV best parameter: {'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (100, 100), 'solver': 'sgd'}
```

```
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:0.80132
```

```
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:0.83152
```

3개의 solver인 sgd, lbfgs, adam과 alpha의 값을 0.0001, 0.01, 1, 활성화함수를 tanh, relu, logistic, 은닉층을 100개의 단일 층, 2층을 가지고 Grid Search진행
-> 각각 **sgd, 1, tanh, 100x100 2층**이 best parameter로 나옴

Best parameter를 test dataset으로 검증해본 결과
->**accuracy**는 **약 0.8**, **ROC AUC**는 **약 0.83**

랜덤 포레스트 모델

```
#train 및 test data를 10000개의 sample에서 원래의 데이터로 바꿈
data = dfu_standard.drop(['HeartDisease'], axis=1)   # 타겟변수를 제외한 변수만 data에 저장
target = dfu_standard['HeartDisease']                # 타겟변수만 target에 저장

from imblearn.under_sampling import RandomUnderSampler    # RandomUnderSampler를 import
undersample = RandomUnderSampler(sampling_strategy=0.333, random_state=1)
                                                          # 타겟변수의 소수 클래스 및 다수 클래스를
                                                          # 1:3의 비율(=1/3)로 언더샘플링
data_under, target_under = undersample.fit_resample(data, target)
                                                          # data 및 target에 언더샘플링 적용
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data_under, target_under, test_size=0.5, random_state=42, stratify=target_under)

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

```
X_train shape: (35196, 37)
X_test shape: (35196, 37)
```

신경망에서 10,000개의 데이터만을 샘플링해서 사용했기 때문에
다시 원래의 데이터를 불러온 후 언더샘플링 및 train, test dataset 분할

랜덤 포레스트 모델

```
rf = RandomForestClassifier(n_estimators = 100, random_state=0)

# 그리드 서치 실행
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

cross_validation = StratifiedKFold(n_splits=3, shuffle=True, random_state=1)
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 3 folds for each of 62 candidates, totalling 186 fits
GridSearchCV max accuracy:0.79876
GridSearchCV best parameter: {'max_depth': 14, 'n_estimators': 200}
```

```
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:0.79498
ROC AUC on test set:0.82838
```

max depth를 10~40, n_estimators를 100개, 200개를 parameter로 하여 train dateset으로 Grid Search를 진행
-> **max depth**는 **14**, **n_estimators**는 **200개**가 best parameter고 최고 accuracy는 약 0.8

test dataset을 사용하여 best parameter를 검증
-> **accuracy**는 **약 0.8**, **ROC AUC**는 **약 0.83**

랜덤 포레스트 모델

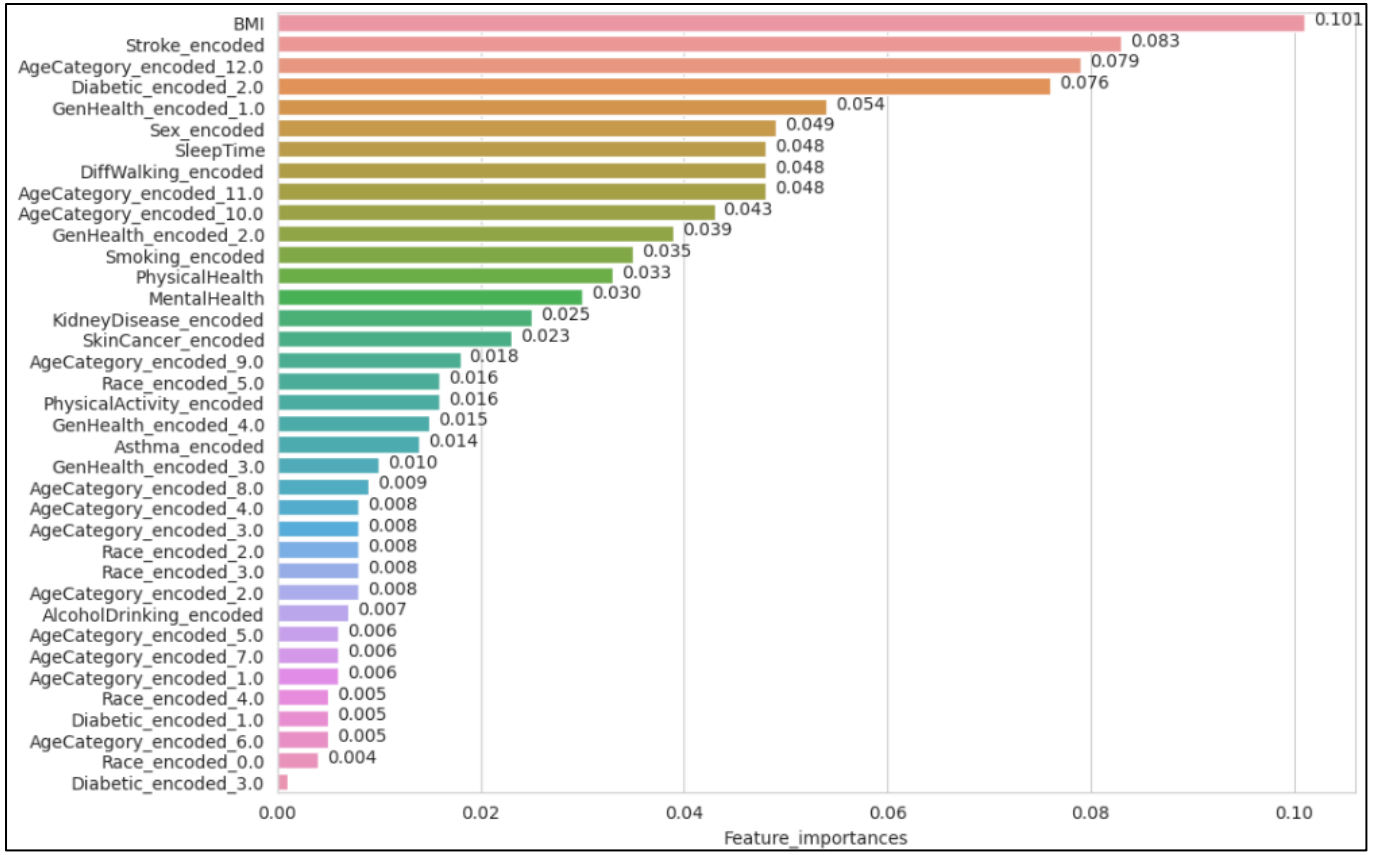
```
feature_names = list(data.columns)   # 변수명(컬럼명)을 리스트 형태로 만들기
dft = pd.DataFrame(np.round(best_clf.feature_importances_, 3), index=feature_names, columns=['Feature_importances'])
dft1 = dft.sort_values(by='Feature_importances', ascending=False)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

plt.figure(figsize=(10, 10))

fig, ax = plt.subplots(figsize=(11, 8))
ax = sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(), p.get_y()+1.3),
                xytext=(5, 10), textcoords='offset points')
```



feature_importances를 이용하여 변수들의 중요도를 시각화

-> **BMI**가 **가장 중요**, Diabetic_encoded_3.0(임신성 당뇨)는 제일 중요하지 않다

연속변수 타겟 데이터 전처리 및 표준화

데이터 셋

beds	baths	size	size_units	lot_size	lot_size_ur	zip_code	price
3	3	2850	sqft	4200	sqft	98119	1175000
4	5	3040	sqft	5002	sqft	98106	1057500
3	1	1290	sqft	6048	sqft	98125	799000
3	2	2360	sqft	0.28	acre	98188	565000
3	3.5	1942	sqft	1603	sqft	98107	1187000
2	2	963	sqft	4753	sqft	98122	701000
1	1	756	sqft			98105	480000
4	6	3300	sqft	5810	sqft	98199	1795000
4	2	2060	sqft	4206	sqft	98144	1025000
1	1	672	sqft			98122	450000
3	2.5	1760	sqft	3630	sqft	98122	1135000

-beds: 침실의 수

-baths: 욕실의 수

-size: 총 층면적

-size_units: size의 단위

-lot_size: 토지 총 면적

-lot_size_units: lot_size의 단위

-zip_code: 우편번호

-price: 부동산 가격

연속변수 타겟 데이터 전처리 및 표준화

```
dft = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/project/train.csv')
dfv = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/project/test.csv')
```

```
dft.head()
```

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
0	3	2.5	2590.0	sqft	6000.00	sqft	98144	795000.0
1	4	2.0	2240.0	sqft	0.31	acre	98106	915000.0
2	4	3.0	2040.0	sqft	3783.00	sqft	98107	950000.0
3	4	3.0	3800.0	sqft	5175.00	sqft	98199	1950000.0
4	2	2.0	1042.0	sqft	NaN	NaN	98102	950000.0

```
dfv.head()
```

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
0	3	3.0	2850.0	sqft	4200.00	sqft	98119	1175000.0
1	4	5.0	3040.0	sqft	5002.00	sqft	98106	1057500.0
2	3	1.0	1290.0	sqft	6048.00	sqft	98125	799000.0
3	3	2.0	2360.0	sqft	0.28	acre	98188	565000.0
4	3	3.5	1942.0	sqft	1603.00	sqft	98107	1187000.0

데이터 셋이 train dataset과 test dataset이 **따로 나눠진 두 개의 csv파일**이므로 **각각 train, test로 사용**하기 위해 데이터 프레임을 만듦

```
dft.isnull().sum()
```

```
beds             0
baths            0
size             0
size_units       0
lot_size       347
lot_size_units 347
zip_code         0
price            0
dtype: int64
```

```
dfv.isnull().sum()
```

```
beds             0
baths            0
size             0
size_units       0
lot_size        77
lot_size_units  77
zip_code         0
price            0
dtype: int64
```

isnull().sum()이나 info()를 통해 **결측값들이 있는 것**을 알 수 있음

```
dft.info(), dfv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2016 entries, 0 to 2015
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   beds            2016 non-null   int64
 1   baths           2016 non-null   float64
 2   size            2016 non-null   float64
 3   size_units      2016 non-null   object
 4   lot_size        1669 non-null   float64
 5   lot_size_units  1669 non-null   object
 6   zip_code        2016 non-null   int64
 7   price           2016 non-null   float64
dtypes: float64(4), int64(2), object(2)
memory usage: 126.1+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   beds            505 non-null    int64
 1   baths           505 non-null    float64
 2   size            505 non-null    float64
 3   size_units      505 non-null    object
 4   lot_size        428 non-null    float64
 5   lot_size_units  428 non-null    object
 6   zip_code        505 non-null    int64
 7   price           505 non-null    float64
dtypes: float64(4), int64(2), object(2)
memory usage: 31.7+ KB
(None, None)
```

연속변수 타겟 데이터 전처리 및 표준화

```
print(dft['size_units'].unique())
print(dfv['size_units'].unique())
print(dft['lot_size_units'].unique())
print(dfv['lot_size_units'].unique())
```

```
['sqft']
['sqft']
['sqft' 'acre' nan]
['sqft' 'acre' nan]
```

```
#size_units의 값이 하나의 범주만 있으므로 제거
dft = dft.drop('size_units', axis=1)
dfv = dfv.drop('size_units', axis=1)
```

```
#zip_code 변수 제거
dft = dft.drop('zip_code', axis=1)
dfv = dfv.drop('zip_code', axis=1)
```

```
print(dft.columns)
print(dfv.columns)
```

```
Index(['beds', 'baths', 'size', 'lot_size', 'lot_size_units', 'price'], dtype='object')
Index(['beds', 'baths', 'size', 'lot_size', 'lot_size_units', 'price'], dtype='object')
```

단위를 나타내는 변수인 size_units에는 **하나의 단위만 존재하므로 drop()**을 이용해 제거함

우편번호를 나타내는 **zip_code변수도 동일하게 제거**함

lot_size의 단위를 나타내는 변수인 lot_size_units는 결측치를 제외하면 2개의 범주를 가지고 있으므로 lot_size를 아래의 코드를 사용하여 **size와 같은 단위인 sqft로 변환**함
cf) 1arce = 43560sqft

```
dft.loc[dft['lot_size_units'] == 'acre', 'lot_size'] = dft.loc[dft['lot_size_units'] == 'acre', 'lot_size'] * 43560
dfv.loc[dfv['lot_size_units'] == 'acre', 'lot_size'] = dfv.loc[dfv['lot_size_units'] == 'acre', 'lot_size'] * 43560
```

연속변수 타겟 데이터 전처리 및 표준화

```
dft = dft.drop('lot_size_units', axis=1)
dfv = dfv.drop('lot_size_units', axis=1)


print(dft.columns)
print(dfv.columns)

Index(['beds', 'baths', 'size', 'lot_size', 'price'], dtype='object')
Index(['beds', 'baths', 'size', 'lot_size', 'price'], dtype='object')


#lot_size 결측치를 평균값으로 대체
#또는 아래 코드를 이용해 제거 가능
#dft1 = dft.dropna()
#dfv1 = dfv.dropna()

lot_size_mean_t = dft['lot_size'].mean()
lot_size_mean_v = dft['lot_size'].mean()
dft['lot_size'].fillna(lot_size_mean_t, inplace=True)
dfv['lot_size'].fillna(lot_size_mean_v, inplace=True)
```

모든 단위를 sqft로 변환 후 의미 없는 **lot_size_units변수를 제거**

그리고 lot_size의 결측 값들을 모델에 사용하기 위해 fillna()를 사용하여 **평균 값으로 대체**

```
dft.isnull().sum()

beds        0
baths       0
size        0
lot_size    0
price       0
dtype: int64


dfv.isnull().sum()

beds        0
baths       0
size        0
lot_size    0
price       0
dtype: int64
```

결측 값을 모두 처리한 것을 볼 수 있음

연속변수 타겟 데이터 전처리 및 표준화

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
dfts = pd.DataFrame(scaler.fit_transform(dft))
dfvs = pd.DataFrame(scaler.fit_transform(dfv))

# StandardScaler( )는 변수명을 지우므로, 새로 만
dfts.columns = dft.columns
dfvs.columns = dfv.columns
```

모든 변수가 연속 변수이기 때문에 바로 StandardScaler()를 사용해 표준화

우측은 표준화 후 데이터프레임의 앞 5행

```
dfts.head()
```

	beds	baths	size	lot_size	price
0	0.113455	0.339428	0.928639	-0.048811	-0.178655
1	0.910407	-0.159687	0.548165	-0.020175	-0.051517
2	0.910407	0.838542	0.330751	-0.057272	-0.014436
3	0.910407	0.838542	2.243994	-0.051960	1.045042
4	-0.683497	-0.159687	-0.754144	0.000000	-0.014436

```
dfvs.head()
```

	beds	baths	size	lot_size	price
0	0.037524	0.770642	1.083019	-0.352134	0.321478
1	0.861421	2.746145	1.289173	-0.307018	0.128181
2	0.037524	-1.204861	-0.609611	-0.248176	-0.297072
3	0.037524	-0.217110	0.551360	0.097721	-0.682021
4	0.037524	1.264518	0.097822	-0.498227	0.341219

연속변수 회귀 모델

```
X_train = dfts.drop('price', axis=1)
y_train = dfts['price']

X_test = dfvs.drop('price', axis=1)
y_test = dfvs['price']

print("Train shape:", X_train.shape, y_train.shape)
print("Test shape:", X_test.shape, y_test.shape)
```

```
Train shape: (2016, 4) (2016,)
Test shape: (505, 4) (505,)
```

```
# 연속변수 타겟변수일 때 Linear Regression 모델 (Default 모델)
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

linr = LinearRegression(n_jobs=-1)
model = linr.fit(X_train, y_train)
pred = model.predict(X_test)

print ("Linear Regression Training set r2 score:{:.5f}".format(model.score(X_train, y_train)))
print ("Linear Regression Test set r2 score:{:.5f}".format(r2_score(y_test, pred)))
```

```
Linear Regression Training set r2 score:0.20610
Linear Regression Test set r2 score:0.45984
```

train.csv는 train data로, test.csv는 test data로 사용

LinearRegression()을 이용해 회귀 모델을 사용함

-training dataset의 r2 score는 약 0.2
-test dataset의 **r2 score**는 **약 0.46**

Ridge 모델

```
from sklearn.linear_model import Ridge
clf_Ridge = Ridge()
```

```
# 그리드 서치 실행
from sklearn.model_selection import GridSearchCV
# 'lbfgs'는 'positive' 매개변수가 True로 설정된 경우에만 사용가능 하다는 warning, 'lbfgs'를 추가해도 fitting에 실패하여 제거함
params = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000],'solver':['auto', 'svd', 'lsqr', 'cholesky','sparse_cg','sag','saga']}

grid_Ridge = GridSearchCV(clf_Ridge, param_grid=params, scoring='r2', cv=5,n_jobs=-1, verbose=1)
grid_Ridge.fit(X_train, y_train)

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

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
GridSearchCV max score:0.39174
GridSearchCV best parameter: {'alpha': 0.001, 'solver': 'sparse_cg'}
```

```
best_clf = grid_Ridge.best_estimator_
pred = best_clf.predict(X_test)

print("R2 Score on test set:{:.5f}".format(best_clf.score(X_test, y_test)))
```

```
R2 Score on test set:0.45984
```

training dataset에 대하여 여러 alpha값들과 solver를 이용해 Grid Search 실행
-> best parameter는 **alpha**값 **0.001**, **solver**는 **sparse_cg**
-> 최대 score는 약 0.4

best parameter를 이용해 test dataset을 검증한 결과
-> **r2 score**는 **약 0.46**

XGBoost 모델

```
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

#standardization되지 않은 dft, dfv 사용
dft.head(3)
```

	beds	baths	size	lot_size	price
0	3	2.5	2590.0	6000.0	795000.0
1	4	2.0	2240.0	13503.6	915000.0
2	4	3.0	2040.0	3783.0	950000.0

```
dfv.head(3)
```

	beds	baths	size	lot_size	price
0	3	3.0	2850.0	4200.0	1175000.0
1	4	5.0	3040.0	5002.0	1057500.0
2	3	1.0	1290.0	6048.0	799000.0

XGBoost와 LightGBM모델은 **표준화되지 않은 데이터프레임**인 dft, dfv 사용

```
dftl = dft.transform(lambda x: np.log(x+1))
dfvl = dfv.transform(lambda x: np.log(x+1))

X_train = dftl.drop('price', axis=1)
y_train = dftl['price']

X_test = dfvl.drop('price', axis=1)
y_test = dfvl['price']

print("Train shape:", X_train.shape, y_train.shape)
print("Test shape:", X_test.shape, y_test.shape)
```

```
Train shape: (2016, 4) (2016,)
Test shape: (505, 4) (505,)
```

그대로 사용하면 XGBoost, LightGBM모델의 **r2 score 값이 음수**가 나와 **로그 변환**을 수행 후 train, test dataset으로 만듦

XGBoost 모델

```
from sklearn.model_selection import GridSearchCV

xgb = XGBRegressor()

parameters = {'colsample_bytree': [0.7],
              'learning_rate': [0.05],
              'max_depth': [16],
              '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
```

train dataset을 colsample_bytree, learnig_rate, max_depth, min_child_weight, n_estimators, subsampe의 값들로 Grid Search를 수행
->best parameter
colsample_bytree: 0.7
learning_rate: 0.05
max_depth: 16
min_child_weight: 4
n_estimators: 1000
subsample: 0.9

test dataset을 best parameter로 XGBoost모델을 수행한 결과
->**r2 score는 약 0.5**

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

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

```
r2: 0.49864
```

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

```
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 16, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.9}
```

LightGBM 모델

train dataset에 대해 XGBoost와 같은 parameter, 다른 값들로 Grid Search 수행
-> best parameter

colsample_bytree: 0.7
learning_rate: 0.1
max_depth: 11
min_child_weight: 4
n_estimators: 1000
subsample: 0.3

test dataset을 best parameter를 이용해 검증한 결과
-> **r2 score**는 **약 0.52**

```
from sklearn.model_selection import GridSearchCV
from lightgbm import LGBMRegressor

lgb = LGBMRegressor()

parameters = {'colsample_bytree': [0.7, 0.8],
              'learning_rate': [0.1, 0.15, 0.2],
              'max_depth': [11],
              'min_child_weight' : [4],
              'n_estimators': [1000],
              'subsample': [0.3, 0.4]
              }

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

```
from sklearn.metrics import r2_score

model = lgb_grid.best_estimator_
pred = model.predict(X_test)

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

```
r2: 0.52124
```

```
print('GridSearchCV 최적 파라미터:', lgb_grid.best_params_)
```

```
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 11, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.3}
```

챔피언 모델 선정

이진 값 타겟 모델의 **ROC AUC**

- 트리 모델
  : 약 0.82
- **로지스틱 회귀 모델**
  : **약 0.84**
- 신경망 모델
  : 약 0.83
- 랜덤 포레스트 모델
  : 약 0.83

-> ROC AUC가 가장 높은 **로지스틱 회귀 모델**이 챔피언!

연속 변수 타겟 모델의 **R2 Score**

- 연속 변수 회귀
  : 약 0.46
- Ridge 모델
  : 약 0.46
- XGBoost 모델
  : 약 0.5
- **LightGBM 모델**
  : **약 0.52**

-> R2 Score가 가장 높은 **LightGBM** 모델이 챔피언!