# sklearn Regression - 수치예측

```
import warnings
warnings.filterwarnings('ignore')
```

# 실습용 데이터 설정

- pandas DataFrame
  - Insurance.csv

```
import pandas as pd

DF = pd.read_csv('https://raw.githubusercontent.com/rusita-ai/pyData/master/Insurance.csv')

DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   expenses  1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
DF.head(3)
```

| | age | sex | bmi | children | smoker | region | expenses |
|---|---|---|---|---|---|---|---|
| **0** | 19 | female | 27.90 | 0 | yes | southwest | 16884.9240 |
| **1** | 18 | male | 33.77 | 1 | no | southeast | 1725.5523 |
| **2** | 28 | male | 33.00 | 3 | no | southeast | 4449.4620 |

## 1) 분석 변수 선택

- X : 'age', 'bmi', 'children'
- y : 'expenses'

```
DF1 = DF[['expenses', 'age', 'bmi', 'children']]

DF1.head(3)
```

| | expenses | age | bmi | children |
|---|---|---|---|---|
| **0** | 16884.9240 | 19 | 27.90 | 0 |
| **1** | 1725.5523 | 18 | 33.77 | 1 |
| **2** | 4449.4620 | 28 | 33.00 | 3 |

## 2) Train &Test Split

- 7:3

```
from sklearn.model_selection import train_test_split

X = DF1[['age', 'bmi', 'children']]
y = DF1['expenses']

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3,
                                                    random_state = 2045)

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

```
Train Data :  (936, 3) (936,)
Test Data :  (402, 3) (402,)
```

# I. Multivariate Regression

## 1) 모델 생성

```
%%time

from sklearn.linear_model import LinearRegression

MR = LinearRegression(normalize = True,
                      n_jobs = -1)

MR.fit(X_train, y_train)
```

```
CPU times: user 25 ms, sys: 5.75 ms, total: 30.7 ms
Wall time: 92.6 ms
```

### 2) 모델 평가

```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, MR.predict(X_test))
```

```
114300134.03204553
```

## II. Ridge Regression

### 1) 모델 생성

- alpha : Regularization strength
  - default : 1.0
  - 값이 커지면 weight 값을 0에 가깝게 학습
  - 값이 작아지면 weight 값을 제한하지 않음
- solver : Optimization Method
  - 'cholesy' : Matrix Decomposition(숄레스키 행렬분해)
  - 'sag' : Stochastic Average Gradient Descent
    - solver = 'sag'
    - random_state = 2045
    - max_iter = 1000

```
%%time

from sklearn.linear_model import Ridge

RG = Ridge(normalize = True,
           alpha = 0.3,
           solver = 'cholesky')

RG.fit(X_train, y_train)
```

```
CPU times: user 2.78 ms, sys: 1.7 ms, total: 4.48 ms
Wall time: 7.33 ms
```

### 2) 모델 평가

```
mean_squared_error(y_test, RG.predict(X_test))
```

```
113578068.78448391
```

# III. Lasso Regression

## 1) 모델 생성

- alpha : Regularization strength
  - default : 1.0
  - 값이 커지면 weight 값을 0에 가깝게 학습
  - 값이 작아지면 weight 값을 제한하지 않음

```
%%time

from sklearn.linear_model import Lasso

LS = Lasso(normalize = True,
           alpha = 0.2)

LS.fit(X_train, y_train)
```

```
CPU times: user 3.02 ms, sys: 1 ms, total: 4.03 ms
Wall time: 4.12 ms
```

## 2) 모델 평가

```
mean_squared_error(y_test, LS.predict(X_test))
```

```
114279766.62560356
```

# IV. ElasticNet Regression

## 1) 모델 생성

- l1_ratio : default = 0.5

```
%%time

from sklearn.linear_model import ElasticNet

EN = ElasticNet(normalize = True,
                alpha = 0.001,
                l1_ratio = 0.7)
```

```
EN.fit(X_train, y_train)
```

```
CPU times: user 3.35 ms, sys: 86 µs, total: 3.44 ms
Wall time: 3.77 ms
```

### 2) 모델 평가

```
mean_squared_error(y_test, EN.predict(X_test))
```

```
113571195.19021483
```

## V. Decision Tree Regressor

### 1) 모델 생성

```
%%time

from sklearn.tree import DecisionTreeRegressor

DTR = DecisionTreeRegressor(max_depth = 5,
                            criterion ='mse')

DTR.fit(X_train, y_train)
```

```
CPU times: user 22.9 ms, sys: 9.25 ms, total: 32.1 ms
Wall time: 81.4 ms
```

### 2) 모델 평가

```
mean_squared_error(y_test, DTR.predict(X_test))
```

```
140104079.12184903
```

### 3) Feature Importance

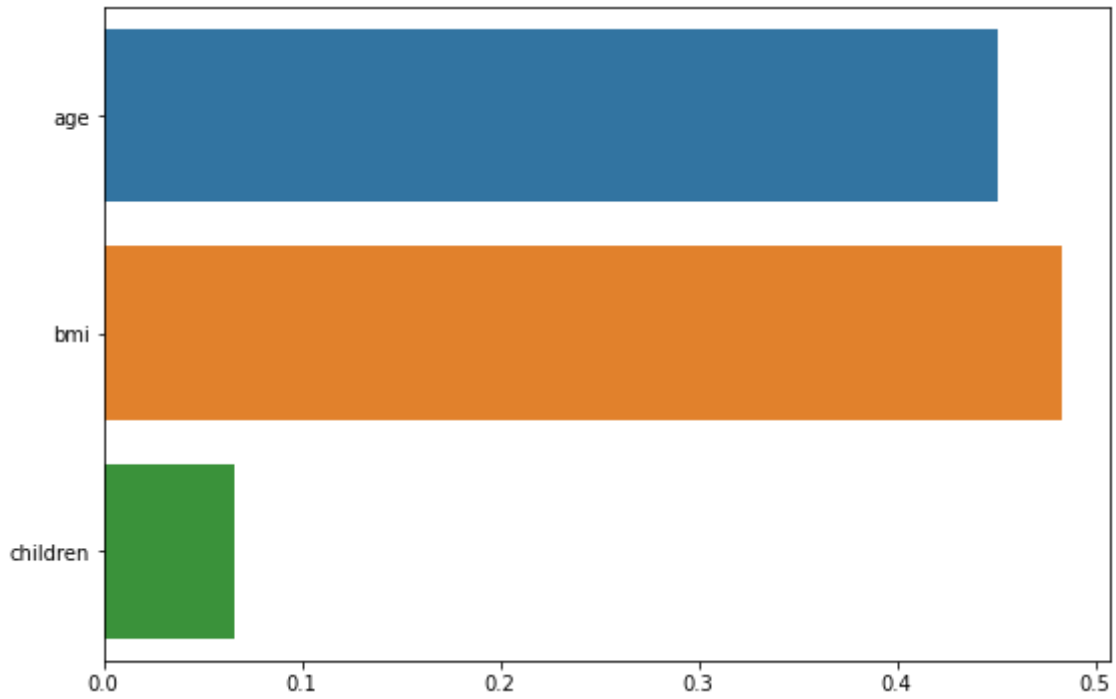
```
DTR.feature_importances_
```

```
array([0.45047558, 0.48298996, 0.06653445])
```

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

```
import seaborn as sns

plt.figure(figsize = (9, 6))
sns.barplot(DTR.feature_importances_,
            ['age', 'bmi', 'children'])
plt.show()
```




# VI. Random Forest Regressor

## 1) 모델 생성

- criterion : default = 'mse'
  - The function to measure the quality of a split.

```
%%time

from sklearn.ensemble import RandomForestRegressor

RFR = RandomForestRegressor(n_estimators = 2000,
                            max_features = 3,
                            max_depth = 1,
                            criterion ='mse',
                            n_jobs = -1,
                            random_state = 2045)

RFR.fit(X_train, y_train)
```

```
CPU times: user 3.46 s, sys: 441 ms, total: 3.9 s
Wall time: 3.26 s
```

## 2) 모델 평가

```
mean_squared_error(y_test, RFR.predict(X_test))
```

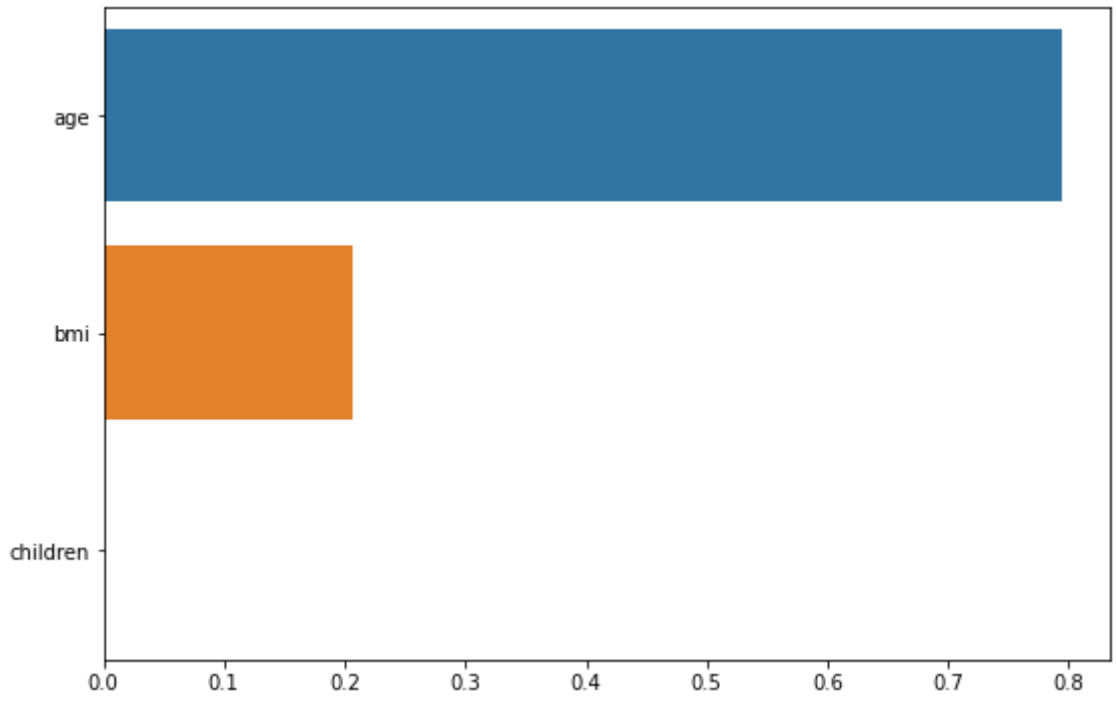
```
114282870.90114409
```

## 3) Feature Importance

```
RFR.feature_importances_
```

```
array([0.794, 0.206, 0.   ])
```

```
plt.figure(figsize = (9, 6))
sns.barplot(RFR.feature_importances_,
            ['age', 'bmi', 'children'])
plt.show()
```




# VII. Gradient Boosting Machine(GBM) Classifier

- 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리를 생성

## 1) 모델 생성

- loss : Optimization Method

    - 'ls' : Least Squares Regression
- n_estimators : 생성되는 트리의 수
    - 값이 크면 모델의 복잡도가 증가
    - 오차를 보정할 기회가 증가
- learning_rate : 이전 트리의 오차를 얼마나 강한게 보정할 것인지 제어
    - 값이 크면 강한 보정에 의해 복잡한 트리 생성

```
%%time

from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor(loss = 'ls',
                                n_estimators = 9000,
                                learning_rate = 0.0001,
                                criterion ='mse',
                                max_features = 3,
                                max_depth = 1)

GBR.fit(X_train, y_train)
```

```
CPU times: user 3.98 s, sys: 0 ns, total: 3.98 s
Wall time: 3.99 s
```

## 2) 모델 평가

```
mean_squared_error(y_test, GBR.predict(X_test))
```

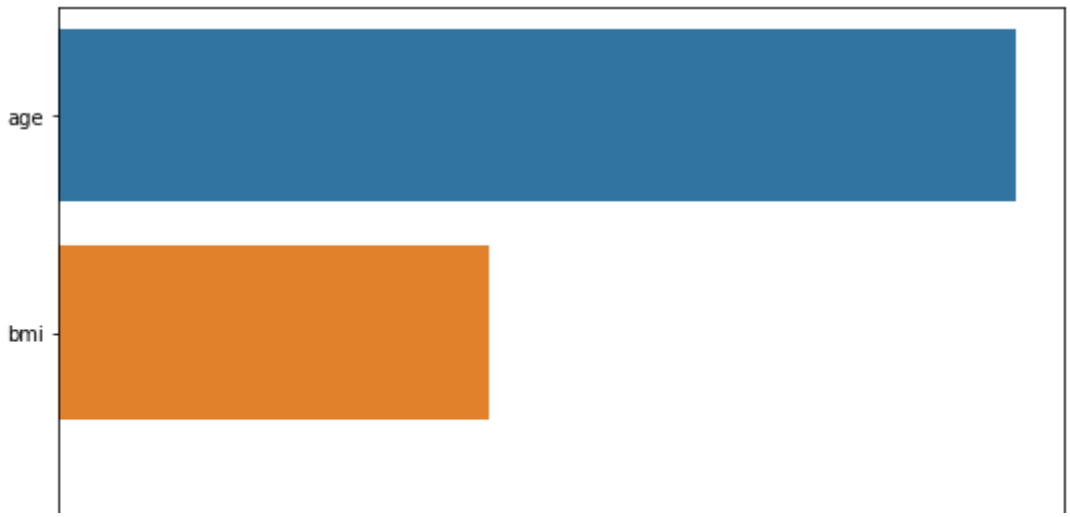
```
118523943.40353534
```

## 3) Feature Importance

```
GBR.feature_importances_
```

```
array([0.68958147, 0.31041853, 0.        ])
```

```
plt.figure(figsize = (9, 6))
sns.barplot(GBR.feature_importances_,
            ['age', 'bmi', 'children'])
plt.show()
```




# VIII. Adaptive Boosting Regressor

- 이전 트리가 잘못 예측한 샘플에 가중치를 높여서 다음 트리를 훈련
- 훈련된 모델은 성능에 따라 가중치가 부여

## 1) 모델 생성

- loss : The loss function to use when updating the weights after each boosting iteration
- base_estimator = None
  - DecisionTreeRegressor
  - max_depth = 3
  - random_state = 2045

```
%%time

from sklearn.ensemble import AdaBoostRegressor

ABR = AdaBoostRegressor(loss = 'square',
                        n_estimators = 500,
                        learning_rate = 0.0001,
                        random_state = 2045)

ABR.fit(X_train, y_train)
```

```
CPU times: user 937 ms, sys: 0 ns, total: 937 ms
Wall time: 940 ms
```

## 2) 모델 평가

```
mean_squared_error(y_test, ABR.predict(X_test))
```

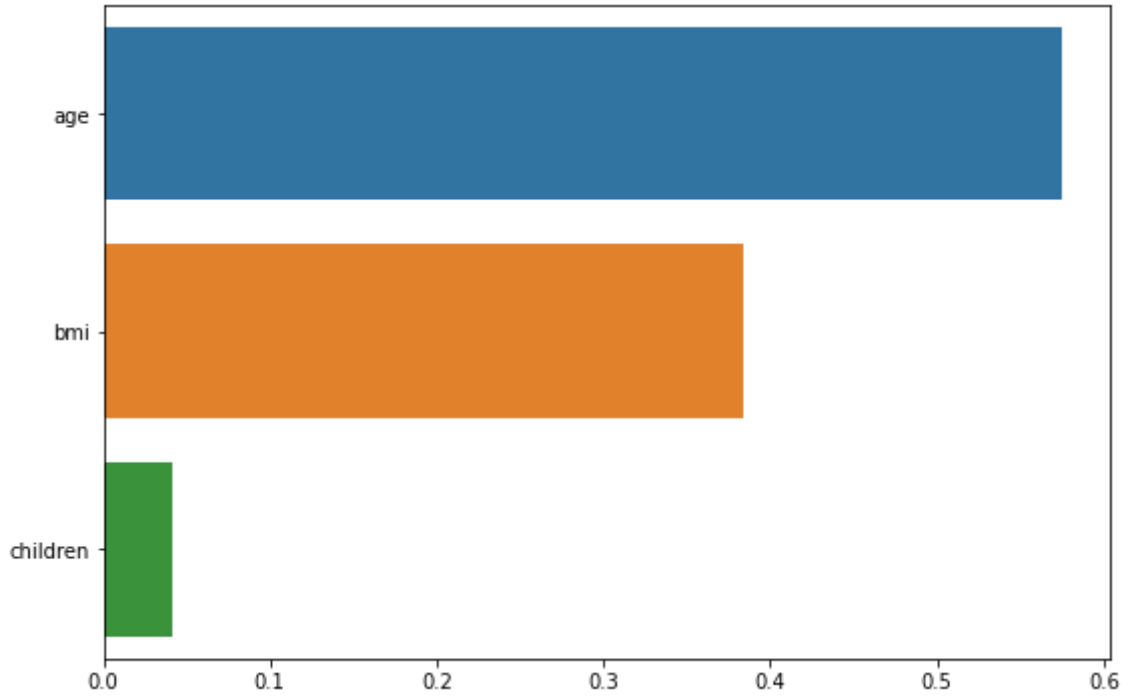
```
118481893.52922821
```

### 3) Feature Importance

```
ABR.feature_importances_
```

```
array([0.57474029, 0.3843226 , 0.04093711])
```

```
plt.figure(figsize = (9, 6))
sns.barplot(ABR.feature_importances_,
            ['age', 'bmi', 'children'])
plt.show()
```




## IX. eXtra Gradient Boost(XGBoost) Classifier

### 1) 모델 생성

```
%%time

from xgboost import XGBRegressor

XGB = XGBRegressor(booster = 'gblinear',
                   n_estimators = 100,
                   learning_rate = 0.4,
                   reg_lambda = 2.0,
                   n_jobs = -1)

XGB.fit(X_train, y_train)
```

```
[12:39:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
CPU times: user 59.2 ms, sys: 11.5 ms, total: 70.8 ms
Wall time: 187 ms
```

### 2) 모델 평가

```
mean_squared_error(y_test, XGB.predict(X_test))
```

```
112756662.7317411
```

## X. LightGBM Regressor

### 1) 모델 생성

```
%%time

from lightgbm import LGBMRegressor

LGB = LGBMRegressor(linear_tree = True,
                    boosting_type = 'gbdt',
                    objective = 'regression',
                    n_estimators = 500,
                    learning_rate = 0.001,
                    max_depth = 2,
                    n_jobs = -1)

LGB.fit(X_train, y_train)
```

```
CPU times: user 112 ms, sys: 16 ms, total: 128 ms
Wall time: 112 ms
```

### 2) 모델 평가

```
mean_squared_error(y_test, LGB.predict(X_test))
```

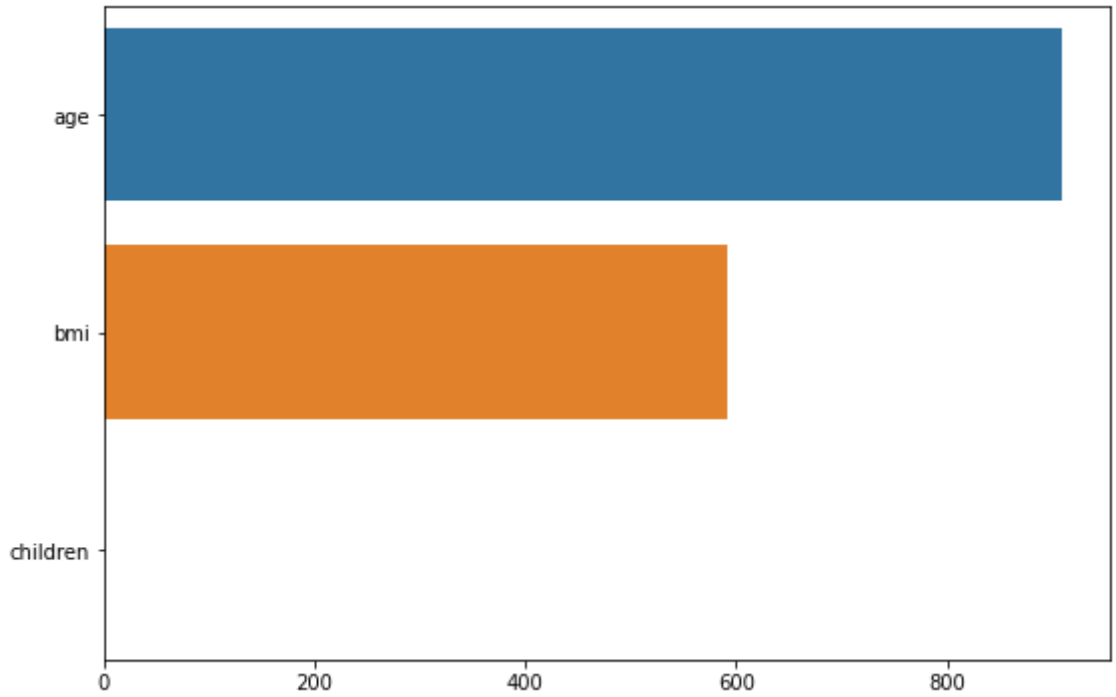
```
119555446.90396042
```

### 3) Feature Importance

```
LGB.feature_importances
```

```
array([909, 591,   0])
```

```
plt.figure(figsize = (9, 6))
sns.barplot(LGB.feature_importances_,
            ['age', 'bmi', 'children'])
plt.show()
```




#

#

#

# The End

#

#

#