



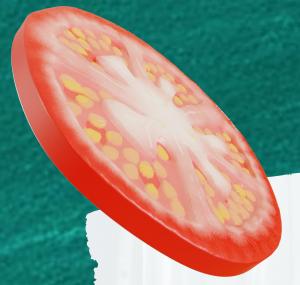
햄버거 肉肉套餐



팀명 : 컨베이어러닝머신

멤버
명노아
이승민
임소영
박희진

역할분담



임소영



명노아



박희진



이승민

주제선정	✓				✓
모델분석	✓	✓	✓	✓	✓
전처리		✓		✓	
git	✓				✓
Readme		✓		✓	
PPT	✓	✓		✓	✓
최종 산출물	✓	✓	✓	✓	✓

0. 주제선정대회

어째서 선정했는가?

햄버거가 먹고싶어서?





You

햄버거를 선택한 이유



ChatGPT

햄버거를 선택하는 이유는 다양할 수 있습니다. 각 개인이나 문화에 따라 다르지만 일반적으로 다음과 같은 이유들이 있을 수 있습니다:

1. **맛:** 햄버거는 다양한 재료를 사용해 맛을 조화롭게 구성할 수 있어, 많은 사람들에게 맛있게 느껴집니다. 고기, 치즈, 신선한 야채, 특제 소스 등이 맛을 높여줍니다.
2. **간편함:** 햄버거는 손에 들어오고 먹기 편리한 음식입니다. 포장이나 트레이로 제공되어 어디에서든 쉽게 먹을 수 있습니다.
3. **다양성:** 다양한 햄버거 종류가 있어 각자의 취향에 맞게 선택할 수 있습니다. 버거 패스트푸드점에서는 고기 선택지뿐만 아니라 채소 버거, 치킨 버거 등 다양한 옵션이 있습니다.
4. **경제적:** 햄버거는 종종 저렴하게 제공되기도 하며, 대부분의 빠른 음식점에서 신속하게 제공되어 시간을 절약할 수 있습니다.
5. **사회적인 모임:** 햄버거는 가족이나 친구와 함께 즐기기 좋은 음식 중 하나입니다. 다양한 사람들이 모여 먹기 쉬운 음식이기 때문입니다.
6. **세계적으로 인기 있는 음식:** 햄버거는 전 세계적으로 많은 사람들에게 사랑받는 음식 중 하나로, 국경을 넘어 인기를 끌고 있습니다.

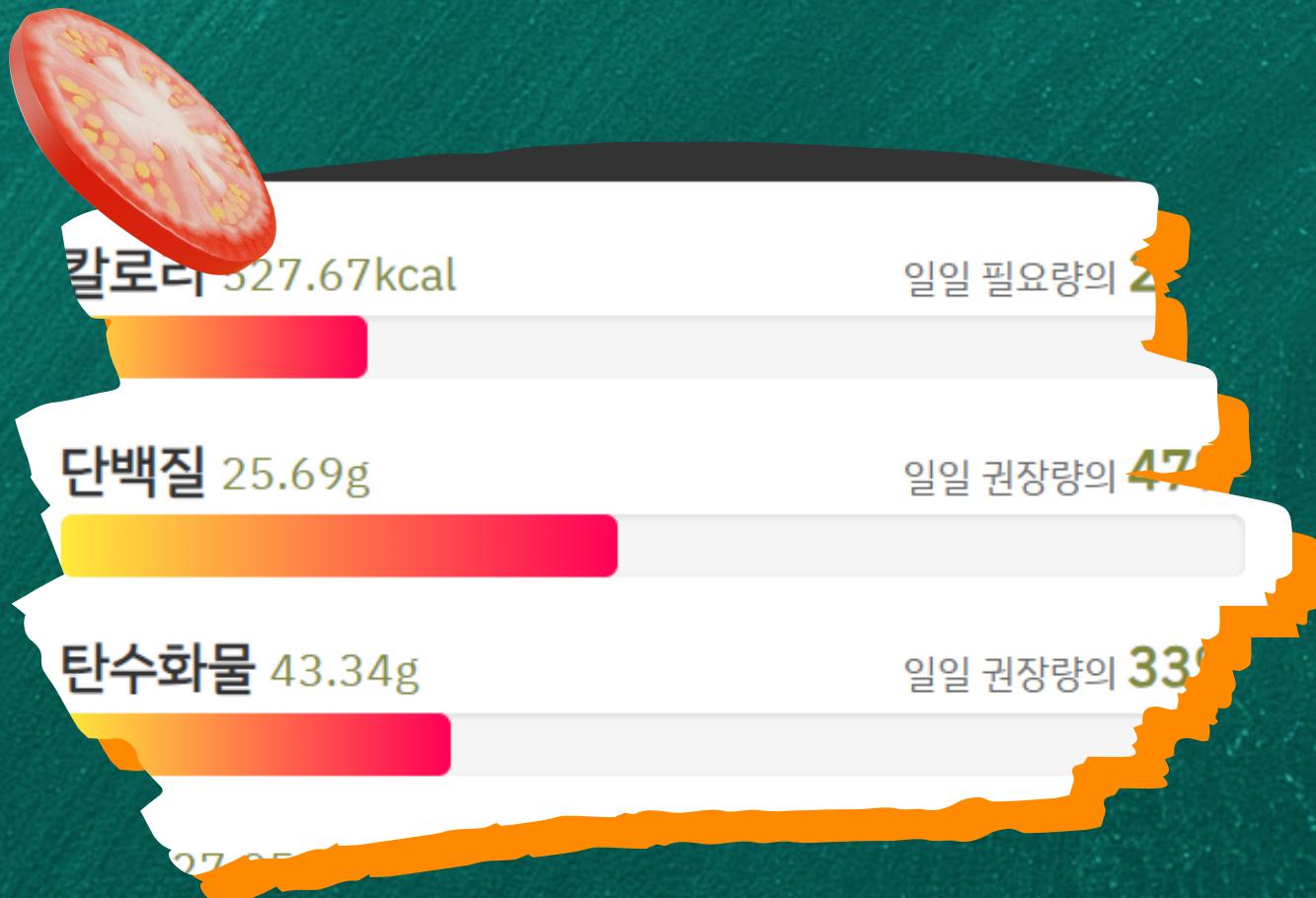
이러한 이유들로 많은 사람들이 햄버거를 선택하게 되며, 다양성과 편의성이 주된 이유 중 하나입니다.



맛, 간편함, 다양성,
경제적, 사회적인 모임 등등...

꽤나 대중적인 주제로 흥미를 유발!

진짜 선점 이유

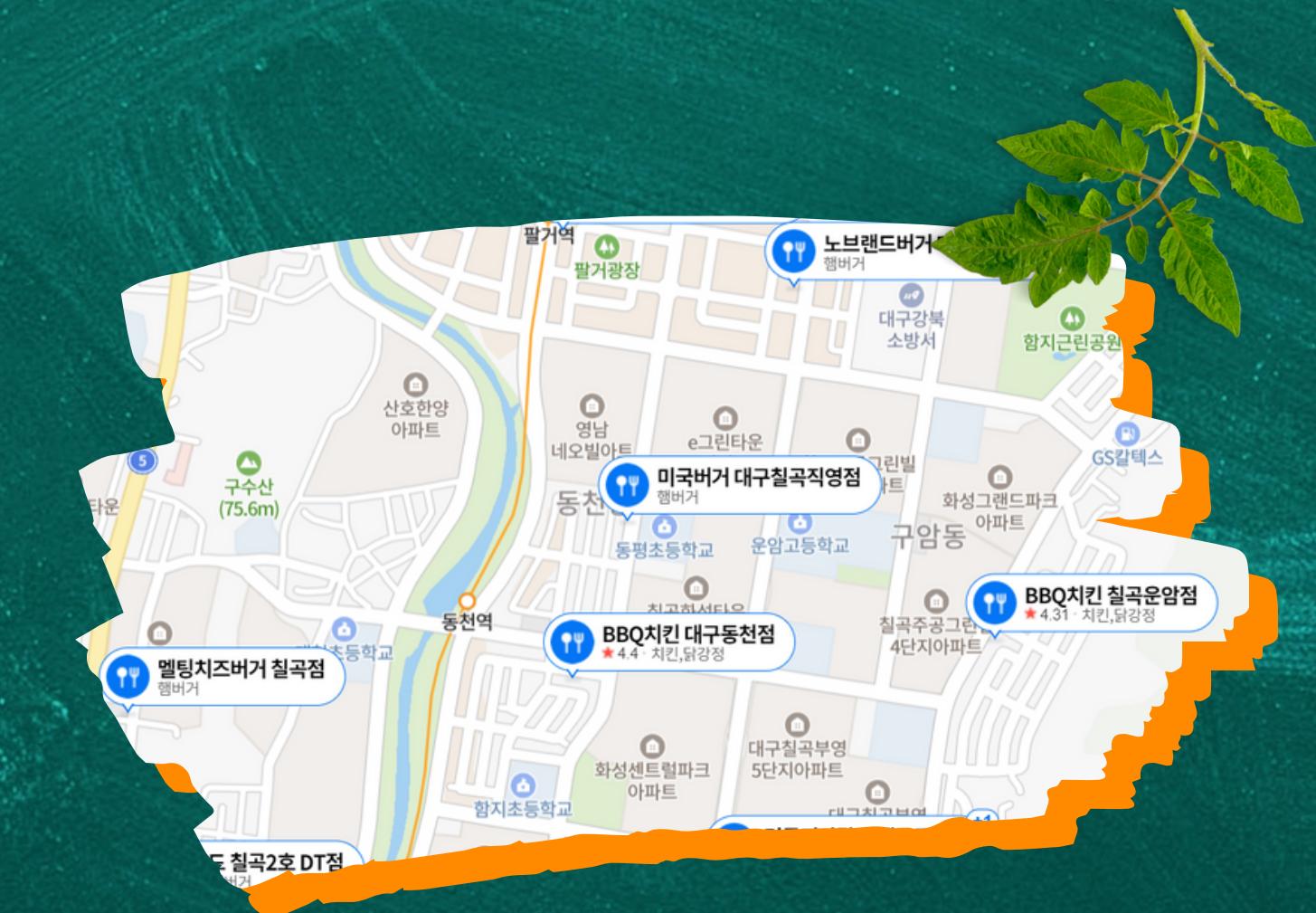


다양한 수치 지표

기본적으로

설탕, 소금, 지방, 단백질

+ 비타민, 수분, 미네랄까지!



많은 지점 데이터
(전국적, 국가적)

전국적으로 퍼져있는

다양한 브랜드의 햄버거 데이터를 수집하여
전체적인 영양 수치를 나타내고, 모델링 할 수 있음

😍최종목표😍

기본 요소(소금,설탕,지방,단백질)

열량 기반 햄버거 추천(회귀모델)



1. 전처리

어떻게 다 갖고오나!!!!



그야... 크롤링으로!

맥도날드
서브웨이
KFC
롯데리아
버거킹
맘스터치
노브랜드
프랭크버거

01

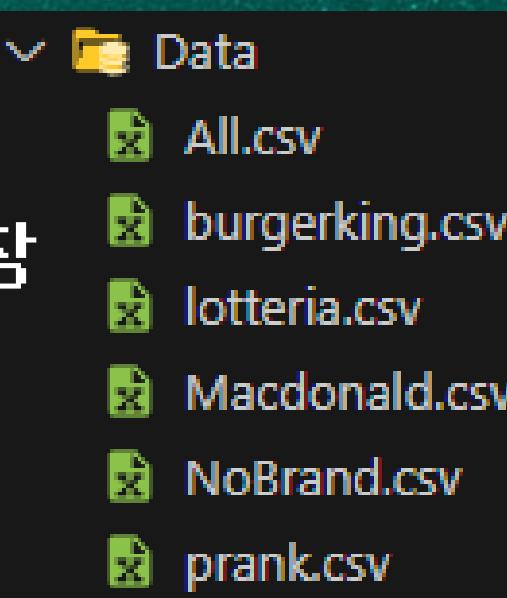
모든 메뉴를 크롤링 (일부는 Kaggle)

02

양쪽 공백, 괄호 내부 값, 물결표 좀복 값 제거

03

각 브랜드 별 “*.csv”로 저장



```
find_text='''#article-view > div.tt_article_useless_p_margin.contents_style > table: nth-child(27)'''  
soup = BeautifulSoup(response.text, "html.parser")  
  
title = soup.select(find_text)  
# table:nth-child에 들어가는 숫자가 27, 39, 52, 63인 요소만 뽑아내기  
content=[]  
text=[]  
for s in title:  
    for i in s:  
        for j in i.select("td"):  
            # trim을 주어서 양쪽 공백을 제거  
            msg=j.text.strip()  
  
            if msg not in ["카페인(mg)", ""]:  
                content.append(msg)  
if len(content)==7:  
    text.append(content)  
    content=[]
```

컬럼 통일

04

공통적으로 있는 컬럼 설정
(메뉴, 소금, 설탕, 지방, 단백질)

05

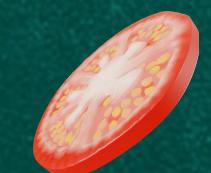
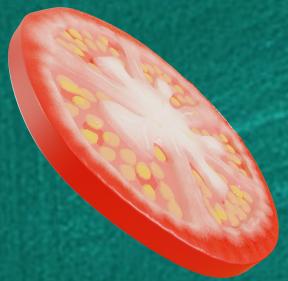
브랜드 명 추가, 및 통일
(macdonald, macdonald, Macdonald)

06

Hamburger.csv로 저장



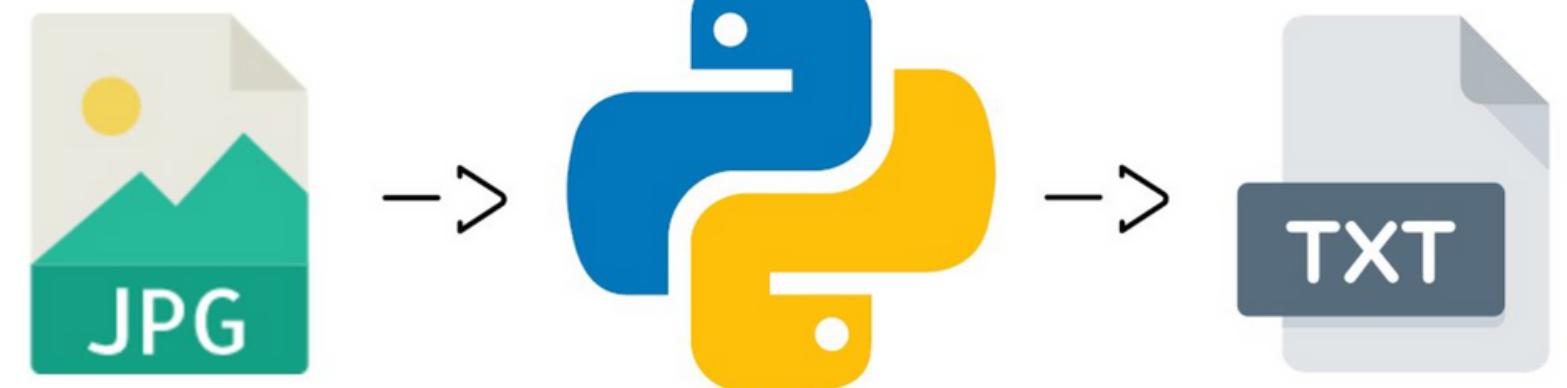
	column 1	column 2	column 3	column 4	column 5	column 6	column 7
1	restaurant	item	calories	sodium	sugar	total_fat	protein
609	lotteria	불고기버거	476.0	778.0	10.0	7.2	20.0
610	lotteria	전주비빔라이스버거	572.0	1460.0	8.0	8.0	17.0
611	lotteria	핫크리스피버거	503.0	929.0	5.0	7.3	22.0
612	lotteria	DoubleX2버거	716.0	1070.0	7.0	25.0	25.0
613	lotteria	불고기 익스트림 오징어	650.0	1070.0	14.0	8.0	29.0
614	lotteria	불고기 베이컨	522.0	840.0	11.0	9.0	22.0
615	lotteria	새우 익스트림 레몬크림	534.0	910.0	12.0	4.1	17.0
616	lotteria	새우_익스트림 레몬크림	534.0	910.0	12.0	4.1	17.0
617	lotteria	새우 베이컨	538.0	870.0	7.0	6.0	17.0
618	lotteria	새우_베이컨	538.0	870.0	7.0	6.0	17.0
619	lotteria	더블 데리버거	446.0	740.0	10.0	7.0	19.0
620	lotteria	더블 치킨버거	478.0	950.0	11.0	5.0	24.0
621	lotteria	더블_치킨버거	478.0	950.0	11.0	5.0	24.0
622	lotteria	더블 클래식치즈버거	731.0	1110.0	6.0	25.0	25.0
623	lotteria	마라로드치킨	573.0	1210.0	14.0	6.0	27.0
624	lotteria	마라로드새우	509.0	1100.0	14.0	4.2	13.0
625	lotteria	마라로드_새우	509.0	1100.0	14.0	4.2	13.0
626	lotteria	마라로드비프	661.0	970.0	13.0	16.0	20.0
627	lotteria	왕돈까스버거	901.0	1570.0	14.0	8.0	35.0
628	lotteria	매운 왕돈까스버거	903.0	1580.0	15.0	8.0	35.0
629	lotteria	한우불고기버거	572.0	797.0	15.0	12.0	24.0
630	lotteria	더블한우불고기버거	802.0	1150.0	18.0	20.0	48.0
631	lotteria	한우트러플머쉬룸버거	584.0	985.0	18.0	14.7	25.0
632	lotteria	빅불버거	761.0	1370.0	12.0	16.0	36.0
633	lotteria	클래식치즈버거_(버터번)	492.0	770.0	5.0	14.0	16.0
634	lotteria	힙&핫치킨버거	597.0	1473.0	20.0	7.6	25.0
635	lotteria	포테이토	267.0	322.0	0.0	3.2	4.0
636	lotteria	포테이토(L)	389.0	471.0	0.0	4.7	6.0
637	lotteria	웨지포테이토	242.0	174.0	0.0	0.5	3.0
722	lotteria	미숫가루라떼 L	339.0	131.0	40.0	8.2	11.0
723	lotteria	복숭아홍차에이드 R	160.0	10.0	38.0	0.0	0.0
724	lotteria	복숭아홍차에이드 L	224.0	10.0	53.0	0.0	0.0
725	lotteria	자두쿨에이드 R	162.0	20.0	36.0	0.1	0.0
726	lotteria	자두쿨에이드 L	226.0	25.0	51.0	0.1	0.0
727	lotteria	아이스티	160.0	0.0	39.0	0.0	0.0
728	lotteria	아이스티(L)	232.0	0.0	57.0	0.0	0.0
729	lotteria	캐모마일 티	2.0	0.0	0.0	0.0	0.0
730	lotteria	애플 캐모마일 티	129.0	10.0	27.0	0.0	0.0
731	lotteria	아메리카노	34.0	2.0	0.0	0.5	1.0
732	lotteria	카페라떼	206.0	105.0	12.0	5.6	11.0
733	lotteria	아이스아메리카노	32.0	2.0	0.0	0.5	1.0
734	lotteria	아이스아메리카노(L)	55.0	3.0	0.0	0.8	2.0
735	lotteria	아이스카페라떼	106.0	40.0	5.0	2.1	5.0
736	McDonalds	Egg McMuffin	300.0	750.0	3.0	13.0	17.0



애로사항



Convert IMG to TXT using
Pytesseract



Pytesseract 사용(OCR)

=> 실패, 공식 사이트, 화질

2. 모델 분석

KDT05 명노아

- Lasso(회귀)
- Ridge(회귀)
- SVM(분류)



모델링 흐름

1. **all_estimator**로 회귀 모델 갖고오기



2. **Ridge, Lasso** 모델 선별



3. 데이터 불러오기,
데이터 셋 분할(훈련용, 시험용)



4. 데이터 정규화



5. 이상값 제거(Z-score)(생략)



6. 데이터 분포, 상관계수 파악



7. 하이퍼 파라미터를 제외하고,
최적의 **random_state** 파악



8. **GridSearchCV** 사용



9. 최적의 파라미터, estimator 찾기



10. 모델 데이터 파일 저장



11. 최종 산출물에 모델 파일 반영

1.all_estimator로 회귀 모델 갖고오기

```
from sklearn.utils.discovery import all_estimators
# 다음 4개에 대해서 진행!
estimators=all_estimators(type_filter='regressor')
for i in estimators:
    if "Ridge" in i[0] :
        print(i)
✓ 1.6s
```

```
('BayesianRidge', <class 'sklearn.linear_model._bayes.BayesianRidge'>)
('KernelRidge', <class 'sklearn.kernel_ridge.KernelRidge'>)
('Ridge', <class 'sklearn.linear_model._ridge.Ridge'>)
('RidgeCV', <class 'sklearn.linear_model._ridge.RidgeCV'>)
```

2. Ridge, Lasso 모델 선별

```
from sklearn.linear_model import Ridge          # Ridge 회귀분석
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import BayesianRidge

from sklearn.model_selection import GridSearchCV # 최적의 파라미터를 찾기 위한 그리드서치 및 기타 기능 import
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
✓ 0.0s

# 위의 Lasso들을 모두 import
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LassoLars
from sklearn.linear_model import LassoLarsCV
from sklearn.linear_model import LassoLarsIC
from sklearn.linear_model import MultiTaskLasso
from sklearn.linear_model import MultiTaskLassoCV

from sklearn.model_selection import GridSearchCV # 최적의 파라미터를 찾기 위한 그리드서치 및 기타 기능 import
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

3. 데이터 불러오기, 데이터 셋 분할(훈련용, 시험용) + 4. 데이터 정규화

```
# Ridge : https://scikit-learn.org/stable/modules/generated/sklearn.
linear_model.Ridge.html#sklearn.linear_model.Ridge
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

test_max=0
train_max=0
for i in range(1000):
    x_train,x_test,y_train,y_test=train_test_split(feature,target,
    test_size=0.3,random_state=i)
    scaler.fit(x_train, y_train)
    x_train=scaler.transform(x_train)
    x_test
    model=Ridge(random_state=i)
    model.fit(x_train,y_train)
    res1 = model.score(x_train,y_train)
    res2 = model.score(x_test,y_test)
    if np.abs(res1-res2) < 0.01 and res1 > train_max:
        train_max=res1
        test_max=res2
    print(i,train_max, test_max)
```

6. 데이터 분포, 상관계수 파악

```
df.describe()
```

✓ 0.0s

	calories	sodium	sugar	total_fat	protein
count	1066.000000	1066.000000	1066.000000	1066.000000	1066.000000
mean	467.100432	904.337467	14.981792	18.103077	35.098255
std	341.998793	788.402823	22.993024	17.269578	116.126794
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	250.000000	231.500000	3.000000	5.000000	10.000000
50%	420.000000	830.000000	7.000000	14.350000	18.000000
75%	612.250000	1300.000000	15.000000		df[["calories","sugar","protein","total_fat","sodium"]].corr()
max	2790.000000	6571.000000	167.000000		✓ 0.0s

	calories	sugar	protein	total_fat	sodium
calories	1.000000	0.305176	0.172225	0.722992	0.811997
sugar	0.305176	1.000000	0.065812	-0.055417	-0.096583
protein	0.172225	0.065812	1.000000	0.093125	0.005111
total_fat	0.722992	-0.055417	0.093125	1.000000	0.706748
sodium	0.811997	-0.096583	0.005111	0.706748	1.000000

7. 하이퍼 파라미터를 제외하고, 최적의 random_state 찾기

```
# Ridge : https://scikit-learn.org/stable/modules/generated/sklearn.
linear_model.Ridge.html#sklearn.linear_model.Ridge
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

test_max=0
train_max=0
for i in range(1000):
    x_train,x_test,y_train,y_test=train_test_split(feature,target,
    test_size=0.3,random_state=i)
    scaler.fit(x_train, y_train)
    x_train=scaler.transform(x_train)
    x_test
    model=Ridge(random_state=i)
    model.fit(x_train,y_train)
    res1 = model.score(x_train,y_train)
    res2 = model.score(x_test,y_test)
    if np.abs(res1-res2) < 0.01 and res1 > train_max:
        train_max=res1
        test_max=res2
    print(i,train_max, test_max)
```

B. GridSearchCV 사용 최적의 파라미터, estimator 찾기

```
x_train, x_test, y_train, y_test = train_test_split(feature, target, test_size=0.3, random_state=28)
param_grid = {
    'tol': [1e-4, 1e-5, 1e-6], # 수렴 기준
    'alpha_1': [1e-7, 1e-6, 1e-5], # alpha_1 값의 후보
    'alpha_2': [1e-7, 1e-6, 1e-5], # alpha_2 값의 후보
    'lambda_1': [1e-7, 1e-6, 1e-5], # lambda_1 값의 후보
    'lambda_2': [1e-7, 1e-6, 1e-5], # lambda_2 값의 후보
    'alpha_init': [None, 1e-8, 1e-7], # alpha_init 값의 후보
    'lambda_init': [None, 1e-8, 1e-7], # lambda_init 값의 후보
    'compute_score': [True, False], # 로그 마진 우도를 계산할지 여부
    'fit_intercept': [True, False], # 절편을 계산할지 여부
}

model = GridSearchCV(BayesianRidge(max_iter=10000, verbose=True), param_grid=param_grid, cv=4)
model.fit(x_train, y_train)
# 최적의 하이퍼 파라미터 출력
print("Best hyperparameters:", model.best_params_)
# 테스트 데이터로 모델 평가
score = model.score(x_test, y_test)
print("Model score on test data:", score)
```

9. 모델 데이터 파일 저장

```
def save_model(model,filename):  
    import joblib  
    import os  
    model_dir='./model/'  
    model_filename=model_dir+f'{filename}.pkl'  
    if not os.path.exists(model_dir):  
        os.makedirs(model_dir)  
    joblib.dump(model.best_estimator_, model_filename)  
    save model(model,"Ridge")
```



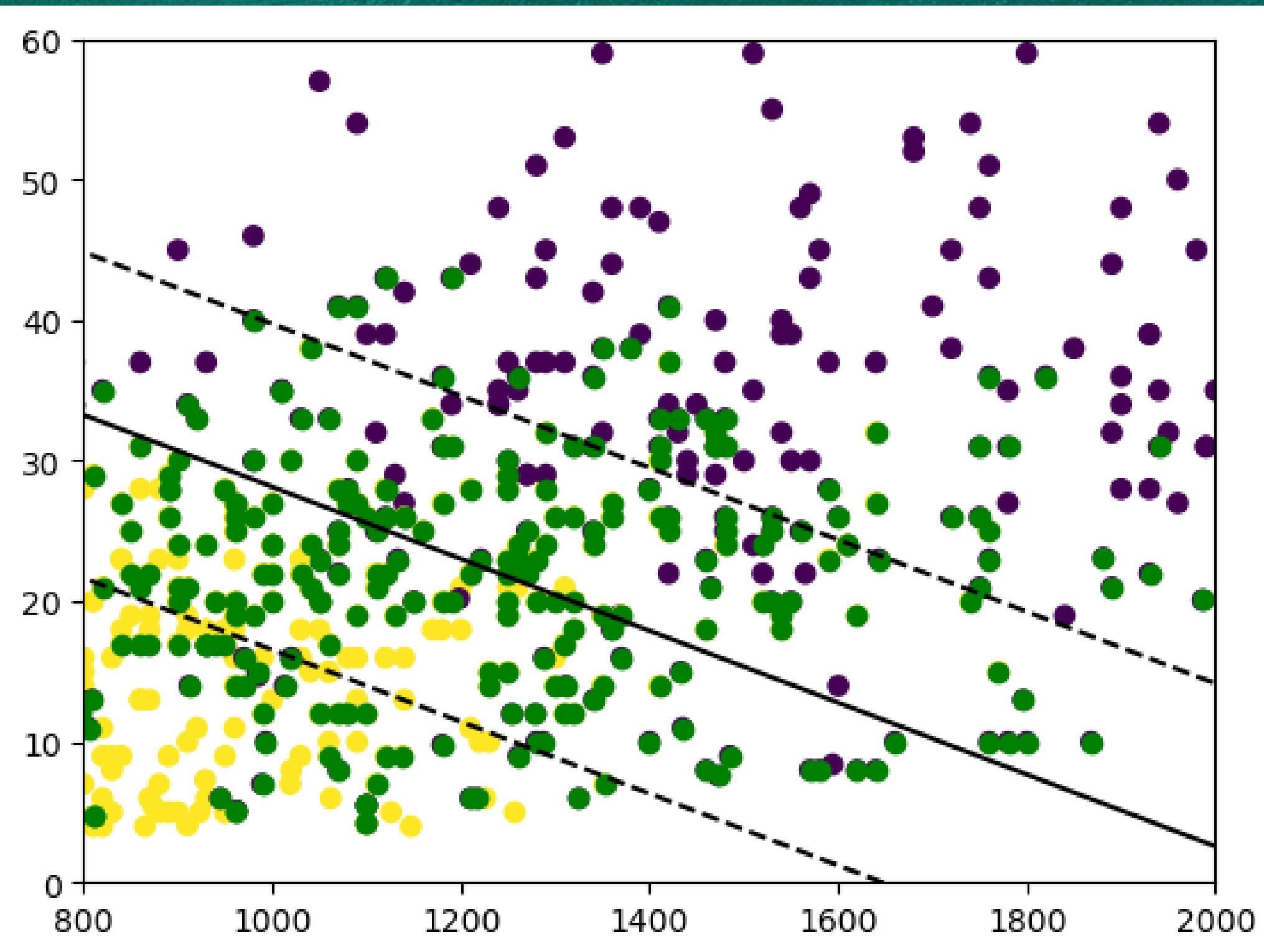
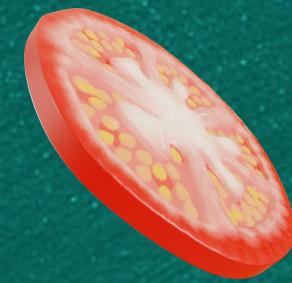
10. 최종 산출물에 모델 파일 반영

```
try :
    df=pd.read_csv('./Hamburger.csv')
except Exception as e:
    print(e)
print(f"{model_list[model_num-1]}모델 불러오기... (성공!)"

model_file=f"./model/{model_list[model_num-1]}.pkl"
try :
    model=load(model_file)
except Exception as e:
    print(e)
```



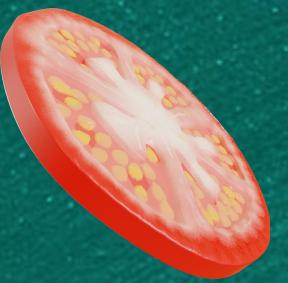
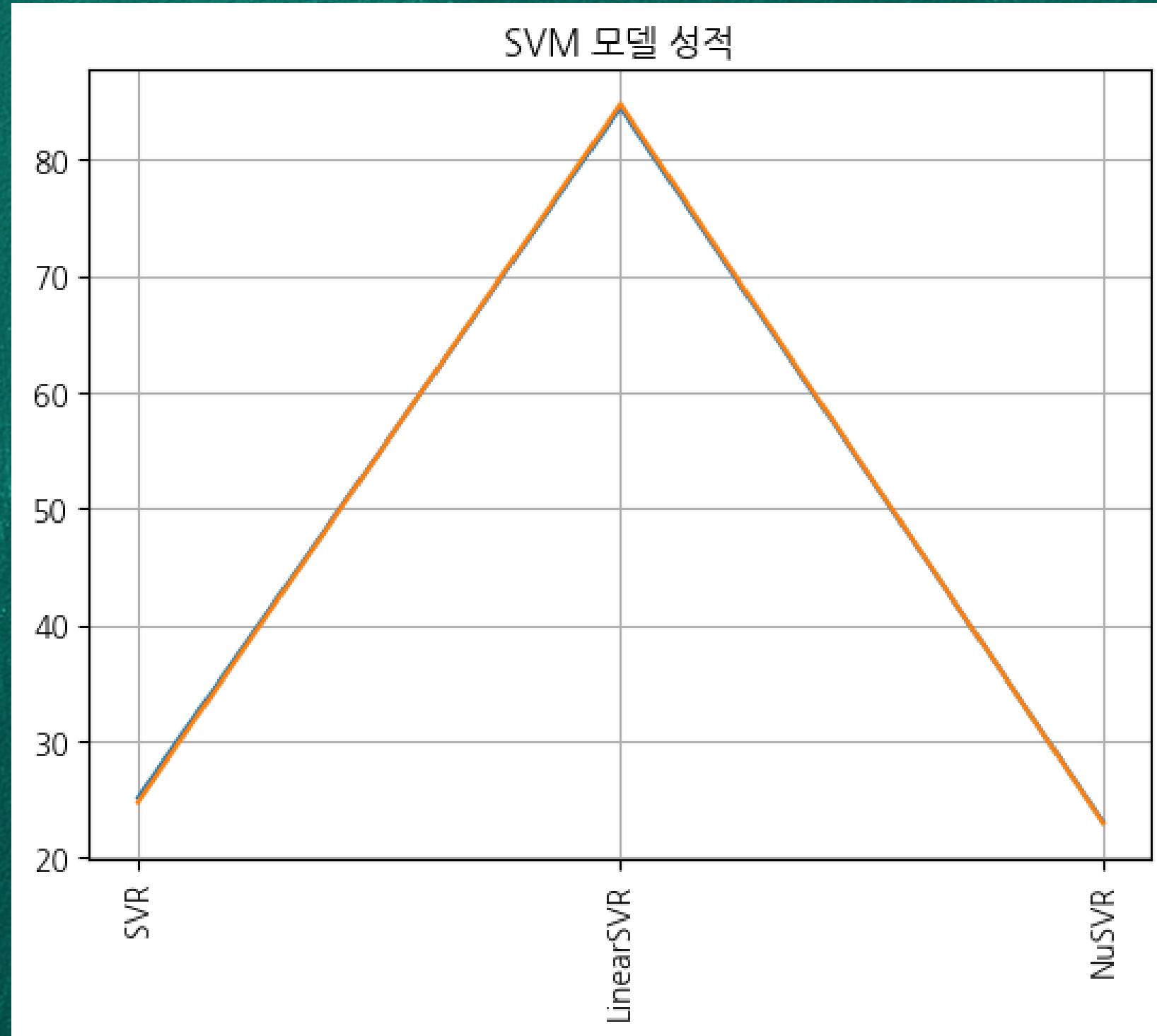
개인 결론



우선, SVM 분류 기법으로 보았을 때,
일반적으로 햄버거 데이터가 제일 많겠지만,
음료수나 감자튀김
추가 메뉴나 세트메뉴 등등...

이상치 값이 너무 많기에,
제대로 된 분류가 되지 않는 모습

개인 결론



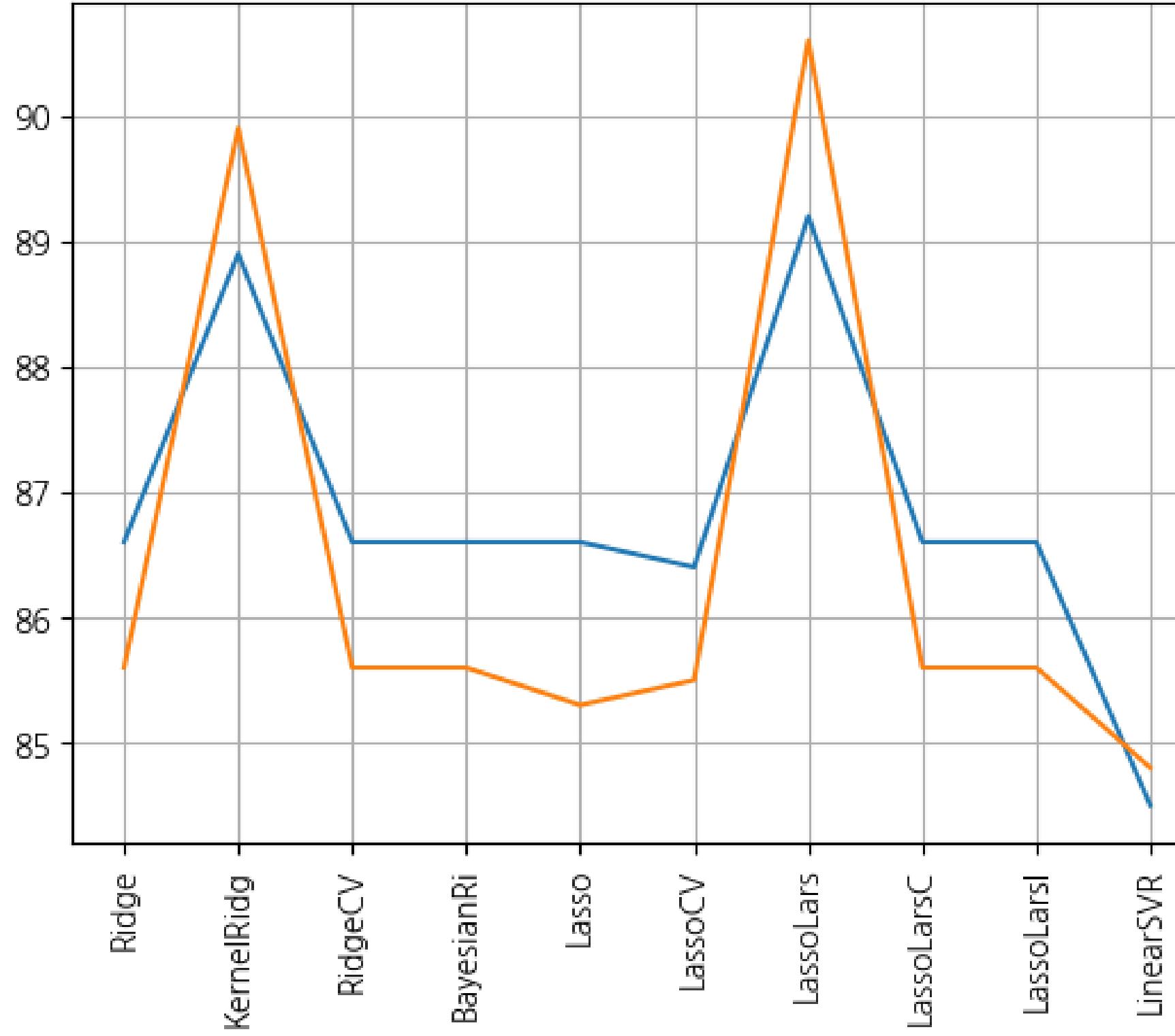
근데 왜 LinearSVR은 잘 나왔는가?

다차원 매핑을 시도하는 SVR, NuSVR과는 달리,
직선을 긋는 LinearSVR은
상관관계가 높은 **지방, 단백질**에 초점을 잡아
정확률이 높게 나옴

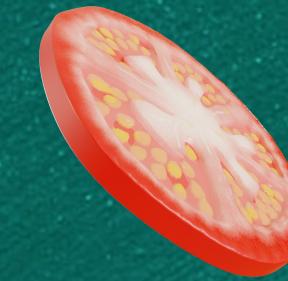


개인 졸론

(Ridge,Lasso,SVM)모델 성적



LassoLars : 90.6%
KernelRidge : 89.9%



01. KernelRidge : 다차원 공간에 매핑하면서도,
특정 계수의 가중치(계수)를 0으로 낮춤으로써 높은 정확률을 보임
비선형 관계를 갖는 데이터에 적합하다

02. LassoLars : 데이터의 일부분만 사용하고,
계수의 축소 경로를 제공하므로, 정규화 강도를 조절하면서 모델을 세밀하게 튜닝할 수 있다

```
from joblib import load

model=load("./model/KernelRidge.pkl")
model
KernelRidge
KernelRidge(alpha=0.9, gamma=0.001, kernel='poly')

from joblib import load

model=load("./model/LassoLars.pkl")
model
LassoLars
LassoLars(eps=0.01, fit_intercept=False, positive=True, precompute=False)
```

KernelRidge

Mean Squared Error: 11158.50
R2 Score: 0.90
Mean Absolute Error: 68.12
Mean Squared Log Error: 0.29
Mean Absolute Percentage Error: 2134320978958745.50
Max Error: 690.25
Explained Variance Score: 0.90
Median Absolute Error: 41.22
Mean Poisson Deviance: 20.33

VS

LassoLars

Mean Squared Error: 12751.61
R2 Score: 0.89
Mean Absolute Error: 81.79
Mean Squared Log Error: 0.25
Mean Absolute Percentage Error: 1287151622406620.00
Max Error: 521.46
Explained Variance Score: 0.90
Median Absolute Error: 56.03
Mean Poisson Deviance: 27.33

예측 오차의 크기 ▲

설명력 ▲

예측 오차의 절댓값 ▲

중앙값 기반의 예측 오차 ▲

평균 포아송 손실 ▲

KernelRidge에 비해 살짝씩 떨어짐...

3. 모델 분석

KDT05 이승민

- KNR(K-최근접 이웃 회귀)
- LR(선형 회귀)





```
[439]: # 데이터 파일  
data_file = './Hamburger.csv'
```

```
[440]: # 데이터 준비  
bergerDF = pd.read_csv(data_file)
```

```
[441]: bergerDF
```

	restaurant	item	calories	sodium	sugar	total_fat	protein
0	Mcdonalds	Artisan Grilled Chicken Sandwich	380.00	1110.00	11.00	7.00	37.00
1	Mcdonalds	Single Bacon Smokehouse Burger	840.00	1580.00	18.00	45.00	46.00
2	Mcdonalds	Double Bacon Smokehouse Burger	1130.00	1920.00	18.00	67.00	70.00
3	Mcdonalds	Grilled Bacon Smokehouse Chicken Sandwich	750.00	1940.00	18.00	31.00	55.00
4	Mcdonalds	Crispy Bacon Smokehouse Chicken Sandwich	920.00	1980.00	18.00	45.00	46.00
...
1063	prank_burger	더블치즈버거(L)	687.04	4.55	33.45	19.51	1149.60
1064	prank_burger	머쉬룸버거(L)	665.30	13.58	26.75	18.10	1203.43
1065	prank_burger	JG버거(L)	828.28	6.42	48.23	22.28	1574.77

머신러닝

목적

- sodium
- sugar
- total_fat
- protein

타겟

- calories

```
[490]: featureDF = bergerDF.iloc[:,3:]
targetSR = bergerDF.iloc[:,2:3]
```

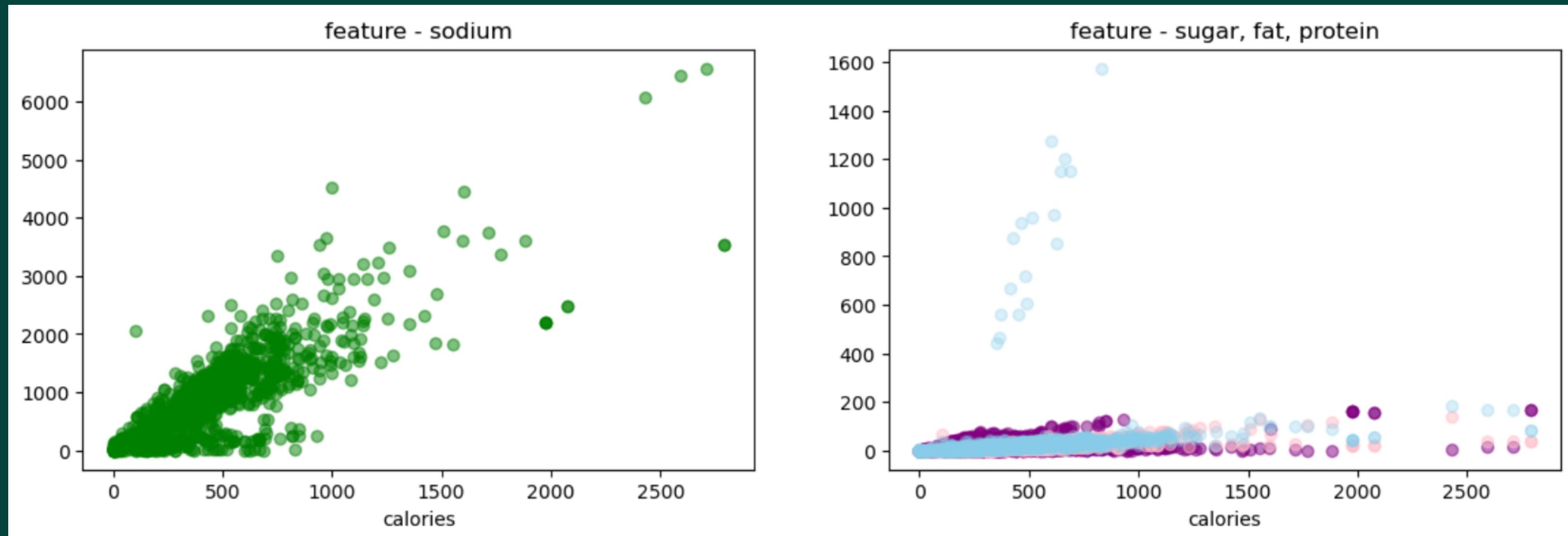
```
[491]: featureDF
```

	sodium	sugar	total_fat	protein
0	1110.00	11.00	7.00	37.00
1	1580.00	18.00	45.00	46.00
2	1920.00	18.00	67.00	70.00
3	1940.00	18.00	31.00	55.00
4	1980.00	18.00	45.00	46.00
...
1063	4.55	33.45	19.51	1149.60
1064	13.58	26.75	18.10	1203.43
1065	6.42	48.23	22.28	1574.77

```
[492]: targetSR
```

	calories
0	380.00
1	840.00
2	1130.00
3	750.00
4	920.00
...	...
1063	687.04
1064	665.30
1065	828.28

파워와 태자의 성과판계 시각화



K-Nearest Neighbors Regression

모델 인스턴스 생성 후 학습

```
from sklearn.model_selection import train_test_split

# 모델 인스턴스 생성 후 학습
X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                    targetSR,
                                                    random_state=10,
                                                    test_size=0.15)

KNRmodel=KNeighborsRegressor()
# 모델 학습
KNRmodel.fit(X_train, y_train)

train_score = KNRmodel.score(X_train, y_train)
test_score = KNRmodel.score(X_test, y_test)
print(f'[TRAIN_SCORE] : {train_score}\n[TEST SCORE] : {test_score}')

[TRAIN_SCORE] : 0.8762657407199128
[TEST SCORE] : 0.7107895473125767
```

최적 random_state값 추적

```
top_train_score = 0
top_test_score = 0
top_random_state = 0
for num in range(1,1001):
    X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                        targetSR,
                                                        random_state=num,
                                                        test_size=0.15)
    # 모델 인스턴스 생성
    KNRmodel=KNeighborsRegressor(n_neighbors=5)
    # 튜닝 -> 파라미터 값 조절
    # 모델 학습
    KNRmodel.fit(X_train, y_train)

    train_score = KNRmodel.score(X_train, y_train)
    test_score = KNRmodel.score(X_test, y_test)
    if (test_score >= top_test_score):# and (train_score >= top_train_score):
        top_train_score = train_score
        top_test_score = test_score
        top_random_state = num

print(f'[TOP_RANDOM_STATE_NUM] : {top_random_state} \n[TOP_TRAIN_SCORE] : {top_train_score} \n[TOP_TEST_SCORE] : {top_test_score}')
[TOP_RANDOM_STATE_NUM] : 308
[TOP_TRAIN_SCORE] : 0.8593757791815952
[TOP_TEST_SCORE] : 0.8927067712942001
```

score값을 끌어올리기 위해 하이퍼파라미터 조정

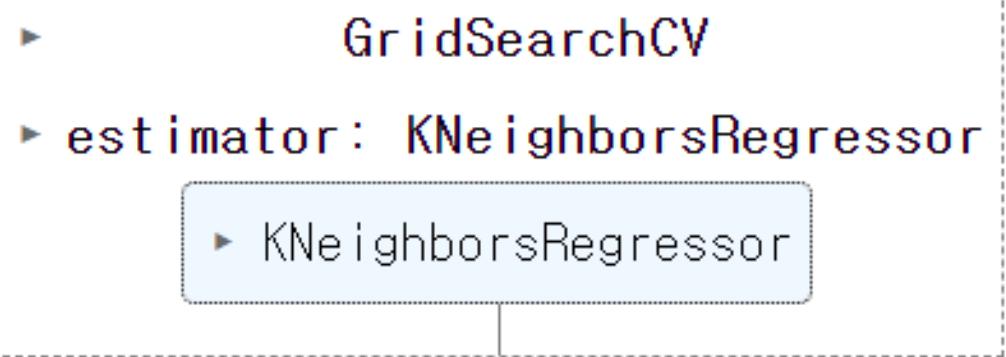
- n-neighbors
- 부동소수점 p
- 가중치 weights

n-neighbors 조정

```
from sklearn.model_selection import GridSearchCV

# 탐색할 하이퍼파라미터 그리드 정의
param_grid = {'n_neighbors': range(1,101)} # 이웃 수 후보 목록

# GridSearchCV를 사용하여 최적의 하이퍼파라미터 탐색
grid_search = GridSearchCV(KNeighborsRegressor, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```



```
# 최적의 이웃 수 출력
print("Best neighbors:", grid_search.best_params_['n_neighbors'])

Best neighbors: 2
```

n-neighbors 조정

```
# 다시 모델 인스턴스 생성 후 학습
KNRmodel=KNeighborsRegressor(n_neighbors=2)
# 모델 학습
KNRmodel.fit(X_train, y_train)

train_score = KNRmodel.score(X_train, y_train)
test_score = KNRmodel.score(X_test, y_test)
print(f'[TRAIN SCORE] : {train_score}\n[TEST SCORE] : {test_score}')

[TRAIN SCORE] : 0.9459763920967256
[TEST SCORE] : 0.9018557049441666
```

부동소수점 p 조정

```
# 투닝(2)
# KNR의 파라미터 중 p 조정 => 부동소수점
top_train_score = 0
top_test_score = 0
top_p = 0
for num in range(1,101):
    KNRmodel=KNeighborsRegressor(n_neighbors=2, p=num)
    # 모델 학습
    KNRmodel.fit(X_train, y_train)

    train_score = KNRmodel.score(X_train, y_train)
    test_score = KNRmodel.score(X_test, y_test)
    if (test_score >= top_test_score):# and (train_score >= top_train_score):
        top_train_score = train_score
        top_test_score = test_score
        top_p = num

print(f'[TRAIN_SCORE] : {top_train_score}\n[TEST SCORE] : {top_test_score}\n[TOP_P] : {top_p}')
[TRAIN_SCORE] : 0.9514125551044949
[TEST SCORE] : 0.9079734985082756
[TOP_P] : 1
```

[초중] 가중치 weights 조정

```
# 투닝(3) 가중치 default='uniform'에서 'distance'로 변경
KNRmodel=KNeighborsRegressor(n_neighbors=2, weights='distance', p=1)
# 모델 학습
KNRmodel.fit(X_train, y_train)

train_score = KNRmodel.score(X_train, y_train)
test_score = KNRmodel.score(X_test, y_test)
print(f'[TRAIN_SCORE] : {train_score}\n[TEST SCORE] : {test_score}')

[TRAIN_SCORE] : 0.9999973480798672
[TEST SCORE] : 0.9158641614791405
```

X_test에 대한 예측값

```
[515]: # 예측  
[516]: y_pre = KNRmodel.predict(X_test)  
[517]: y_pre  
  
[517]: array([[2.01409639e+02],  
[2.86551724e+02],  
[4.52564103e+02],  
[6.20076142e+02],  
[4.26521739e+02],  
[1.41802920e+03],  
[6.78666667e+02],  
[1.61176471e+02],  
[2.46923077e+02],  
[3.89655172e+02],  
[1.02215865e+03],  
[1.64615385e+02],  
[6.18536585e+02],  
[5.38000000e+02],  
[3.40000000e+02],  
[4.02631579e+02],  
[2.13176471e+02],  
[7.52424242e+02],
```

모델 성능평가

```
# 성능 평가
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
# 성능평가 => 결정계수(r2) -> 100% 가까울수록 좋음  
r2_score(y_pre, y_test)
```

```
0.9069528194740757
```

```
# 평균절대오차 => 오차가 적을수록 좋음  
mean_absolute_error(y_pre, y_test)
```

```
66.30999035990929
```

```
# 평균제곱근오차 => 오차가 적을수록 좋음  
mean_squared_error(y_pre, y_test, squared=False)
```

```
99.63602618740421
```

Linear Regression

모델 인스턴스 생성 후 학습

```
# 모델 인스턴스 생성 후 학습
X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                    targetSR,
                                                    random_state=10,
                                                    test_size=0.15)

LRmodel=LinearRegression()
# 모델 학습
LRmodel.fit(X_train, y_train)

train_score = LRmodel.score(X_train, y_train)
test_score = LRmodel.score(X_test, y_test)
print(f'[TRAIN_SCORE] : {train_score}\n[TEST SCORE] : {test_score}')

[TRAIN_SCORE] : 0.860759957789226
[TEST SCORE] : 0.8768822759157684
```

최적 random_state값 추적

```
top_train_score = 0
top_test_score = 0
top_random_state = 0

for num in range(1,1001):
    X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                        targetSR,
                                                        random_state=num,
                                                        test_size=0.15)

    # 모델 인스턴스 생성
    LRmodel=LinearRegression()
    # 투닝 -> 파라미터 값 조절
    # 모델 학습
    LRmodel.fit(X_train, y_train)

    train_score = LRmodel.score(X_train, y_train)
    test_score = LRmodel.score(X_test, y_test)

    if (test_score >= top_test_score):# and (train_score >= top_train_score):
        top_train_score = train_score
        top_test_score = test_score
        top_random_state = num

print(f'[TOP_RANDOM_STATE_NUM] : {top_random_state} \n[TOP_TRAIN_SCORE] : {top_train_score} \n[TOP_TEST_SCORE] : {top_test_score}')

[TOP_RANDOM_STATE_NUM] : 287
[TOP_TRAIN_SCORE] : 0.8475865785527302
[TOP_TEST_SCORE] : 0.928787126167401
```

score값을 끌어올리기 위해 하이퍼파라미터 조정

- 회귀식 상수항 포함 여부 fit-intercept
- 입력 데이터 복사 여부 copy_X
- 병렬처리 n_jobs

fit-intercept 조정

```
top_train_score = 0
top_test_score = 0
top_random_state = 0
for num in range(1,1001):
    X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                        targetSR,
                                                        random_state=num,
                                                        test_size=0.15)

    # 모델 인스턴스 생성
    LRmodel=LinearRegression(fit_intercept=False)
    # 튜닝 -> 파라미터 값 조절
    # 모델 학습
    LRmodel.fit(X_train, y_train)

    train_score = LRmodel.score(X_train, y_train)
    test_score = LRmodel.score(X_test, y_test)
    if (test_score >= top_test_score):# and (train_score >= top_train_score):
        top_train_score = train_score
        top_test_score = test_score
        top_random_state = num

print(f'[TOP_RANDOM_STATE_NUM] : {top_random_state} \n[TOP_TRAIN_SCORE] : {top_train_score} \n[TOP_TEST_SCORE] : {top_test_score}')
[TOP_RANDOM_STATE_NUM] : 287
[TOP_TRAIN_SCORE] : 0.8465368865209739
[TOP_TEST_SCORE] : 0.9293046643036056
# fit_intercept=False일 때 test 점수 up, train 점수 down
```

COPY_X 조절

```
top_train_score = 0
top_test_score = 0
top_random_state = 0
for num in range(1,1001):
    X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                        targetSR,
                                                        random_state=num,
                                                        test_size=0.15)
    # 모델 인스턴스 생성
    LRmodel=LinearRegression(fit_intercept=False, copy_X=False)
    # 투닝 -> 파라미터 값 조절
    # 모델 학습
    LRmodel.fit(X_train, y_train)

    train_score = LRmodel.score(X_train, y_train)
    test_score = LRmodel.score(X_test, y_test)
    if (test_score >= top_test_score):# and (train_score >= top_train_score):
        top_train_score = train_score
        top_test_score = test_score
        top_random_state = num

print(f'[TOP_RANDOM_STATE_NUM] : {top_random_state} \n[TOP_TRAIN_SCORE] : {top_train_score} \n[TOP_TEST_SCORE] : {top_test_score}')
[TOP_RANDOM_STATE_NUM] : 287
[TOP_TRAIN_SCORE] : 0.8465368865209739
[TOP_TEST_SCORE] : 0.9293046643036056
# copy_X는 False로 바꿔도 똑같음
```

[초|중] n_jobs 조정

```
X_train, X_test, y_train, y_test = train_test_split(featureDF,
                                                    targetSR,
                                                    random_state=287,
                                                    test_size=0.15)

# 모델 인스턴스 생성
LRmodel=LinearRegression(fit_intercept=False, copy_X=False, n_jobs=100)
# 튜닝 -> 파라미터 값 조절
# 모델 학습
LRmodel.fit(X_train, y_train)

train_score = LRmodel.score(X_train, y_train)
test_score = LRmodel.score(X_test, y_test)
print(f'[TRAIN_SCORE] : {train_score} \n[TEST_SCORE] : {test_score}')

[TRAIN_SCORE] : 0.8465368865209739
[TEST_SCORE] : 0.9293046643036056

# n_jobs도 변화 없음
```

X_test에 대한 예측값

```
[48]: # 예측  
[49]: y_pre = LRmodel.predict(X_test)  
[50]: y_pre  
  
[50]: array([[ 406.38485369],  
[ 241.25397717],  
[ 127.95826192],  
[ 645.1724444 ],  
[ 857.13405374],  
[ 481.80905667],  
[ 387.27483578],  
[ 310.87387291],  
[ 241.53515392],  
[ 957.02019165],  
[ 583.82846681],  
[ 427.10036443],  
[ 445.89598491],  
[ 250.35166375],  
[ 661.25669684],  
[ 523.70493984],  
[ 880.22958027],  
[ 540.38803012],
```

모델 성능평가

```
# 성능 평가
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
# 성능평가 => 결정계수값(r2) -> 1에 가까울수록 좋음
```

```
r2_score(y_pre, y_test)
```

```
0.9228634738263799
```

```
# 평균절대오차 => 오차가 적을수록 좋음
```

```
mean_absolute_error(y_pre, y_test)
```

```
73.081755354463
```

```
# 평균제곱근오차 => 오차가 적을수록 좋음
```

```
mean_squared_error(y_pre, y_test, squared=False)
```

```
102.47857196463563
```

4. 모델 분석

KDT05 임소영

- DT(의사결정트리)
- Boosting(부스팅)



DecisionTree & Boosting

모델을 선택한 이유

- 그래프가 아닌 또 다른 시각화를 이용해볼 수 있다는 점
- 수업시간에 활용해보지 않았던 모델이라 호기심이 생겼다는 점

Decision Tree

1단계 : 전처리

DecisionTree

```
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split  
  
Executed at 2024.03.11 06:00:20 in 1s 199ms  
  
file = './Hamburger.csv'  
data = pd.read_csv(file)  
data.info()
```

- 결측값 확인
- 중복값 확인 (소수라 그대로 유지)

```
data.isna().sum()
```

Executed at 2024.03.11 06:00:23 in 15ms

	123 <unnamed>	Length: 7, dtype: int64
restaurant	0	
item	0	
calories	0	
sodium	0	
sugar	0	
total_fat	0	
protein	0	

```
data.duplicated().sum()
```

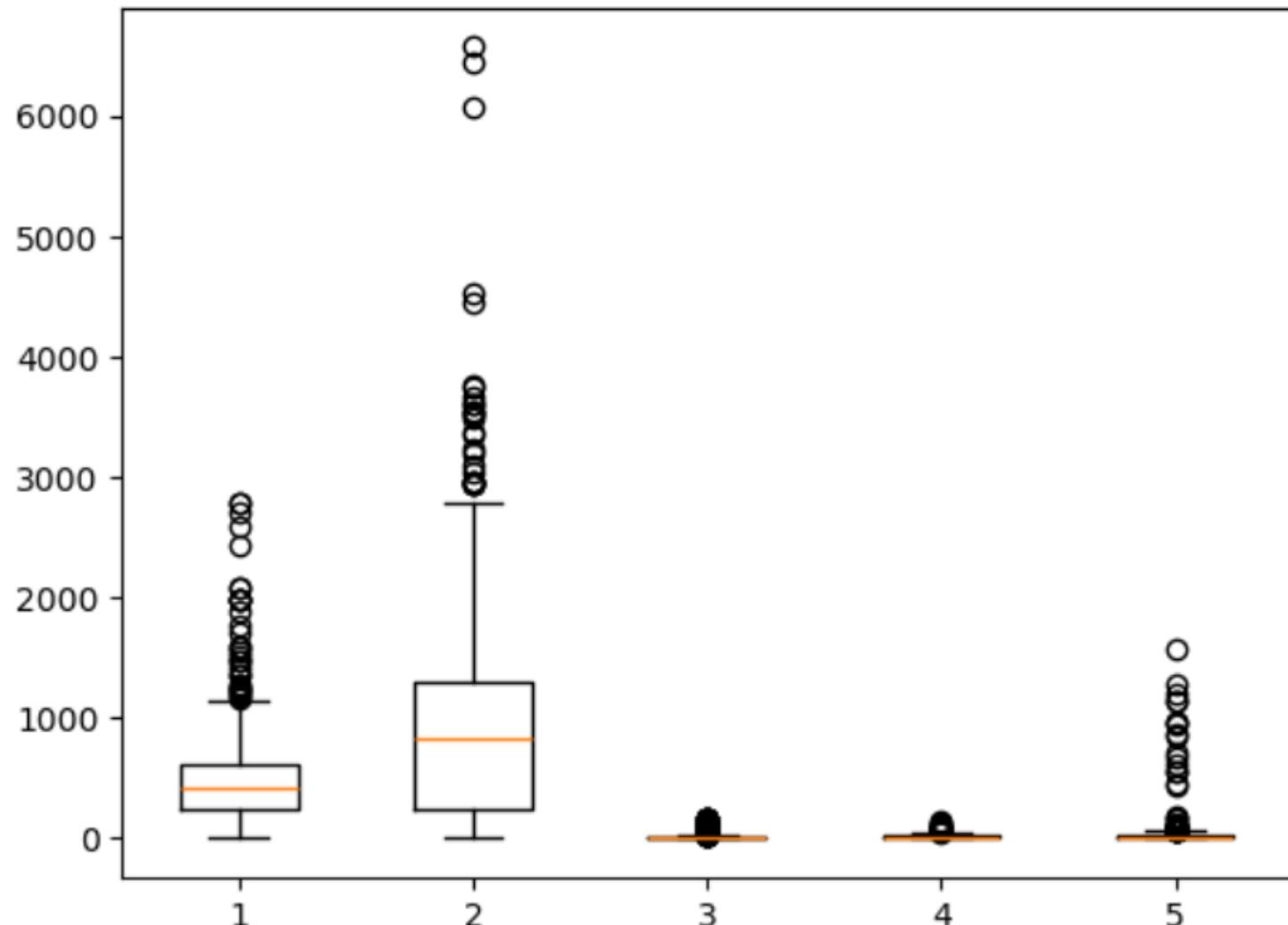
Executed at 2024.03.11 06:00:24 in 20ms

1단계 : 전처리

DecisionTree

```
plt.boxplot(data[['calories', 'sodium', 'sugar', 'total_fat', 'protein']])
plt.show()
```

Executed at 2024.03.11 06:25:13 in 81ms



```
def print_flier(insert_data):
    q1 = insert_data.quantile(0.25)
    q3 = insert_data.quantile(0.75)
    iqr = q3 - q1
    #print(f"[ q1 ] \n{q1}\n\n[ q3 ] \n{q3}\n")
    #print(f"[ iqr의 범위 ]\n{iqr}\n")

    print('----- 이상값이 될 기준 계산 -----\\n')

    lower = q1 - 1.5 * iqr
    #print(f"[ lower의 값 ]\n{lower}\n")
    upper = q3 + 1.5 * iqr
    #print(f"[ upper의 값 ]\n{upper}\n")

    mask = insert_data < lower
    print(f"[ lower의 개수 ]\n{mask.sum()}\n")
    mask2 = insert_data > upper
    print(f"[ upper의 개수 ]\n{mask2.sum()}\n")

    return lower, upper

lower, upper = print_flier(data[['calories', 'sodium', 'sugar', 'total_fat', 'protein']])
```

이상치 시각화로 확인

이상치 제거를 위한
upper, lower 값 계산함

1단계 : 전처리

```
def drop_flier(insert_data, col_name):
    global lower
    global upper
    print(f"이상치 제거 전 shape 확인: {insert_data.shape}")
    mask = insert_data[col_name] >= lower.loc[col_name]
    insert_data = insert_data[mask]
    #print(f" '{col_name}' 컬럼에서 lower 미만의 값 제거 후 shape: {insert_data.shape}")

    mask2 = insert_data[col_name] <= upper.loc[col_name]
    insert_data = insert_data[mask2]
    #print(f" '{col_name}' 컬럼에서 upper 초과의 값 제거 후 shape: {insert_data.shape}\n")

    print(f"이상치 제거 후 shape 확인: {insert_data.shape} \n")
    return insert_data

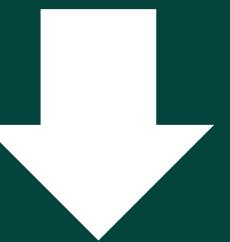
data_for_drop = data[['calories','sodium','sugar','total_fat','protein']]
# data_for_drop = drop_flier(data_for_drop, 'calories')
# data_for_drop = drop_flier(data_for_drop, 'sodium')
# data_for_drop = drop_flier(data_for_drop, 'sugar')
# data_for_drop = drop_flier(data_for_drop, 'total_fat')
# data_for_drop = drop_flier(data_for_drop, 'protein')

# 이상치 제거한 데이터에 맞게 원본 데이터도 제거된 인덱스 번호를 같이 drop 시켜줌
# for i in list(data.index):
#     if i not in (list(data_for_drop.index)):
#         data.drop(i,inplace = True)
# print(data.shape)
#
# # 인덱스 번호가 서로 일치하는지 확인
# print(len(list(data_for_drop.index)) == len(list(data.index)))
# # 서로 일치하는 것을 확인 했으니 이 데이터를 이용해서 알고리즘 돌리겠다.
Executed at 2024.03.11 06:00:27 in 11ms
```

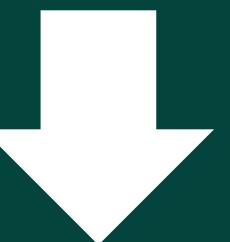
이상치를 제거했더니 과대 적합이 계속 나왔다. 이상치를 다시 복구 했더니 과대적합이 덜나와서 이상치를 다시 복구했다.

DecisionTree

컬럼별로 이상치 제거 함



그러나 학습 및 평가 과정에서
과대 적합이 계속 발생함
(10점 이상 차이남)



그래서 다시 이상치를 복구함

1단계 : 전처리

DecisionTree

```
data.corr(numeric_only=True)  
Executed at 2024.03.10 20:21:23 in 43ms
```

이상치 제거 0

	calories	sodium	sugar	total_fat	protein
calories	1.000000	0.857458	0.280034	0.792475	0.837067
sodium	0.857458	1.000000	0.160648	0.742896	0.827102
sugar	0.280034	0.160648	1.000000	0.099064	0.204924
total_fat	0.792475	0.742896	0.099064	1.000000	0.665665
protein	0.837067	0.827102	0.204924	0.665665	1.000000

```
data.corr(numeric_only=True)  
Executed at 2024.03.11 06:00:30 in 24ms
```

이상치 제거 X

	123 calories	123 sodium	123 sugar	123 total_fat	123 protein
calories	1.000000	0.811985	0.305116	0.723007	0.172215
sodium	0.811985	1.000000	-0.096722	0.706815	0.005060
sugar	0.305116	-0.096722	1.000000	-0.055567	0.065918
total_fat	0.723007	0.706815	-0.055567	1.000000	0.093055
protein	0.172215	0.005060	0.065918	0.093055	1.000000

상관관계 순위

1. 나트륨
2. 단백질
3. 지방
4. 설탕

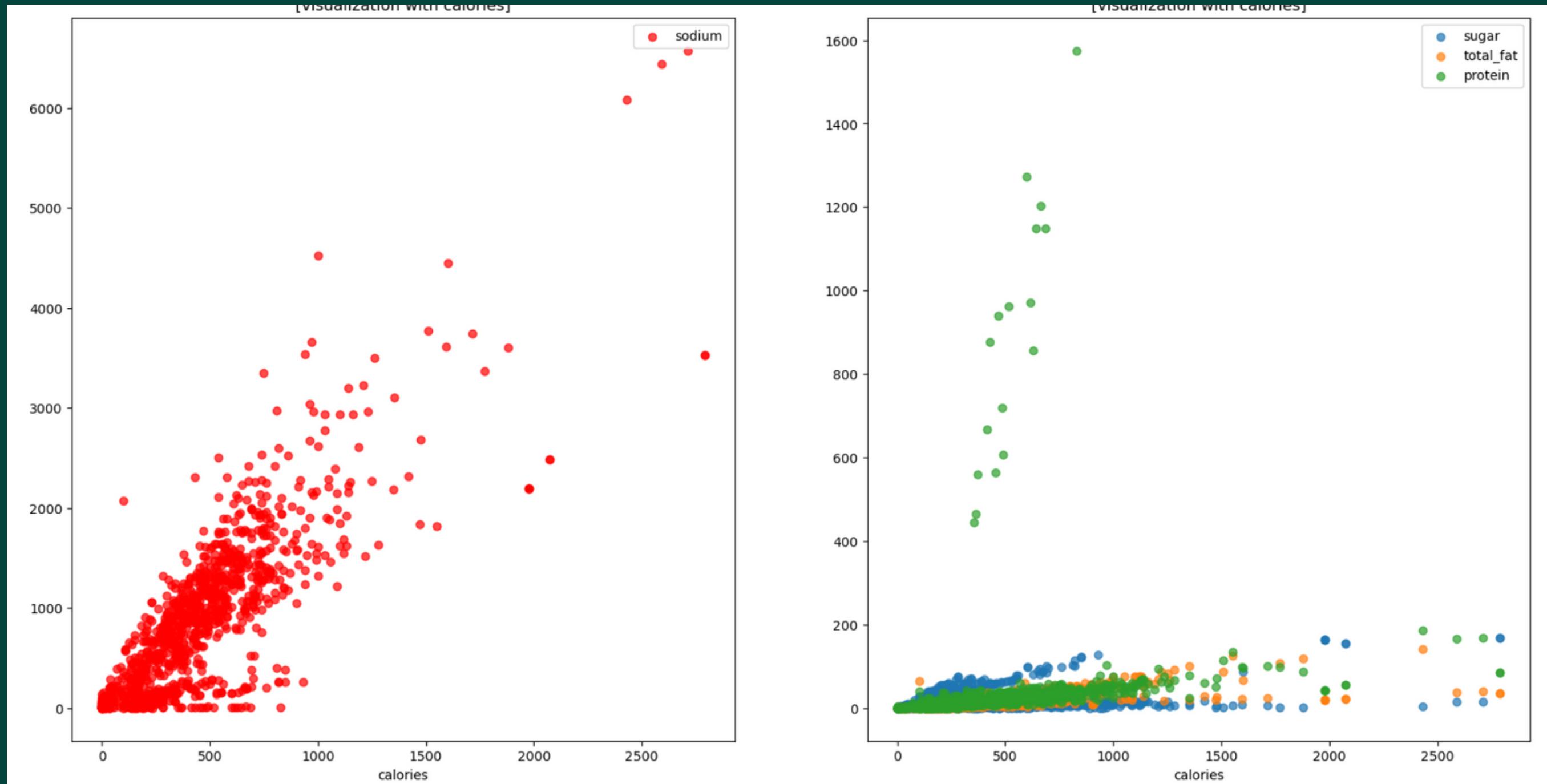
상관관계 순위

1. 나트륨
2. 지방
3. 설탕
4. 단백질

칼로리와 다른 영양소 간의 상관관계를 분석해보았다.

1단계 : 전처리

DecisionTree



상관관계를 시각화로 확인해보았다.

2단계 : 데이터셋 준비

DecisionTree

```
target = data['calories']
feature = data[['sodium','sugar','total_fat','protein']]
Executed at 2024.03.11 06:00:34 in 16ms

from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
train_score_list = []
test_score_list = []

# 최고의 random_state 값 찾기
...
for number in range(1,1001):
    X_train, X_test, y_train, y_test = train_test_split(feature, target, random_state = number)

    scaler = StandardScaler()
    scaler.fit(X_train)
    scaled_X_train = scaler.transform(X_train)
    scaled_X_test = scaler.transform(X_test)
    X_train = pd.DataFrame(scaled_X_train, columns = ['sodium', 'sugar', 'total_fat', 'protein'])
    X_test = pd.DataFrame(scaled_X_test, columns = ['sodium', 'sugar', 'total_fat', 'protein'])
```

최고의 random_state 값을 찾으라!

2단계 : 데이터셋 준비

DecisionTree

```
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

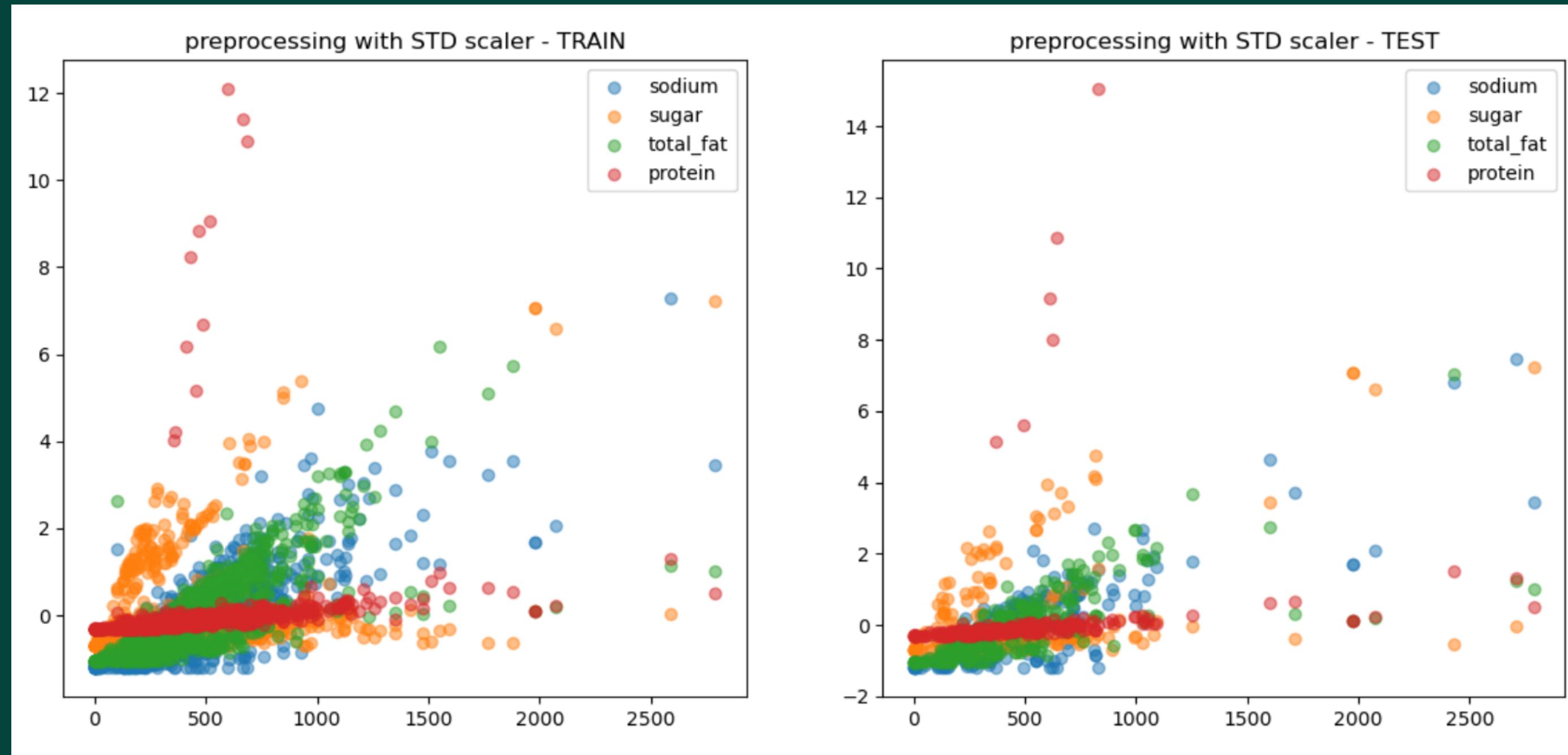
train_score = model.score(X_train, y_train)
test_score = model.score(X_test, y_test)
train_score_list.append(train_score)
test_score_list.append(test_score)
#print(f"{number} => [train score] {train_score} [test score] {test_score}")

max_score = max(test_score_list)
for i in range(len(test_score_list)):
    if test_score_list[i] == max_score:
        print(f"최고의 random_state 값은 {i+1}")
        best_random = i+1
        ...
# 425 ←
```

최고의 random_state 값을 찾으라!

2단계 : 데이터셋 준비

DecisionTree



피처 중 하나인 나트륨의 값의 범위가 상당히 차이가 나서
Standard scaler를 이용하여 스케일링을 완료했다.

3단계: 학습, 평가

DecisionTree

```
model = DecisionTreeRegressor()  
model.fit(X_train, y_train)  
  
train_score = model.score(X_train, y_train)  
test_score = model.score(X_test, y_test)  
train_score_list.append(train_score)  
test_score_list.append(test_score)  
print(f"[train score] {train_score} [test_score] {test_score}")  
# 과대 적합이 나왔다.
```

Executed at 2024.03.11 06:00:44 in 23ms

[train score] 0.9999993659170866 [test_score] 0.9046606477456087

과대 적합이 발생함

3단계: 학습, 평가 + 튜닝

DecisionTree
+
파라미터 조정

```
criterion_list = ['squared_error','friedman_mse','absolute_error','poisson']
max_depth_list = [3, 4, 5, 6, 7]
max_features_list = [2, 3, 4, 'sqrt','log2']
min_impurity_decrease_list = [0, 1, 2, 3]
min_samples_leaf_list = [2, 3, 4]
min_samples_split_list = [2, 3, 4]
splitter_list = ['best', 'random']
list_for_df = []

for A in criterion_list:
    for B in splitter_list:
        for C in max_depth_list:
            for D in min_samples_split_list:
                for E in min_samples_leaf_list:
                    for F in max_features_list:
                        for G in min_impurity_decrease_list:
                            sample_list = []
                            model = DecisionTreeRegressor(criterion = A, splitter=B, max_depth = C, min_samples_split=D, min_samples_leaf=E, max_features = F, min_impurity_decrease=G)
                            model.fit(X_train, y_train)

                            train_score = model.score(X_train, y_train)
                            test_score = model.score(X_test, y_test)
                            sample_list.append(A)
                            sample_list.append(B)
                            sample_list.append(C)
                            sample_list.append(D)
                            sample_list.append(E)
                            sample_list.append(F)
                            sample_list.append(G)
                            sample_list.append(train_score)
                            sample_list.append(test_score)
                            list_for_df.append(sample_list)
```

DecisionTreeRegressor의
각 파라미터에 대한
for문을 돌렸다.

3단계: 학습, 평가 + 튜닝1

DecisionTree
+
파라미터 조정

```
result.sort_values(by = 'test_score', ascending = False)[:5]
```

Executed at 2024.03.11 08:33:49 in 23ms

	criterion	splitter	max_depth	min_samples_split	min_samples_leaf	max_features	min_impurity_decrease
2647	friedman_mse	best		7	4	2 3	
2645	friedman_mse	best		7	4	2 3	
2546	friedman_mse	best		7	2	3 3	
780	squared_error	best		7	3	2 2	
2485	friedman_mse	best		6	4	3 3	

```
total_model_01 = DecisionTreeRegressor(criterion = 'friedman_mse', splitter='best', max_depth = 7, min_samples_split=4, min_samples_leaf=3, max_features = 3, min_impurity_decrease=3)
model = total_model_01
model.fit(X_train, y_train)

train_score = model.score(X_train, y_train)
test_score = model.score(X_test, y_test)
print(train_score, test_score)
```

Executed at 2024.03.11 08:34:20 in 18ms

0.9117074826138167 0.9016390563963234

조합된 파라미터 중 test score가
가장 높은 파라미터 조합을 뽑았다.

[score 결과]

train score : 0.91 test score : 0.90

4단계 : 예측, 성능 평가

DecisionTree
+

total_model_01

Executed at 2024.03.11 08:34:35 in 17ms

DecisionTreeRegressor

```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=7, max_features=3,  
min_impurity_decrease=3, min_samples_leaf=3,  
min_samples_split=4)
```

y_pred = model.predict(X_test)

Y_pred

Executed at 2024.03.11 08:34:38 in 8ms

267 rows × 1 columns

	0
0	578.812500
1	288.258065
2	2204.250000
3	176.421053
4	356.774194
5	562.500000
6	376.040000
7	165.000000
8	188.000000
9	941.055556

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score  
  
print(f"R2 SCORE : {r2_score(y_test, y_pred)}")  
print(f"MSE : {mean_squared_error(y_test, y_pred)}")  
print(f"MAE : {mean_absolute_error(y_test, y_pred)}")
```

Executed at 2024.03.11 08:34:39 in 11ms

R2 SCORE : 0.9016390563963234

MSE : 15298.067772077899

MAE : 75.37967119202818

파라미터 조정

3단계: 학습, 평가 + 튜닝

DecisionTree
+

```
from sklearn.model_selection import GridSearchCV
Executed at 2024.03.11 08:34:41 in 19ms

DTR_model = DecisionTreeRegressor()
param = {
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'],
    'max_depth': [3, 4, 5, 6, 7],
    'max_features': [2, 3, 4, 'sqrt', 'log2'],
    'min_impurity_decrease': [0, 1, 2, 3],
    'min_samples_leaf': [2, 3, 4],
    'min_samples_split': [2, 3, 4],
    'splitter' : ['best', 'random']
}
cv_num = 5

# 모델 생성
gs = GridSearchCV(DTR_model, param_grid = param, cv = cv_num, return_train_score = True)

# 학습 및 평가
result = gs.fit(X_train, y_train)
resultDF = pd.DataFrame(gs.cv_results_)
print(f"max train score : {max(resultDF['mean_train_score'])}")
print(f"max test score : {max(resultDF['mean_test_score'])}\n")

# 최고 성능의 파라미터, 최고의 점수
print(f"best parameters : {gs.best_params_}")
print(f"best score : {gs.best_score_}")
Executed at 2024.03.11 08:37:24 in 2m 42s 180ms
```

```
max train score : 0.9400479899454067
max test score : 0.8286991895827454
```

```
best parameters : {'criterion': 'poisson', 'max_depth': 5, 'max_features': 2, 'min_impurity_decrease': 0, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
best score : 0.8286991895827454
```

이번엔 GridSearchCV를 활용한
튜닝을 해보았다.

GridSearchCV

4단계 : 예측, 성능 평가

```
total_model_02 = gs.best_estimator_
total_model_02
```

Executed at 2024.03.11 08:37:57 in 19ms

```
DecisionTreeRegressor
```

```
DecisionTreeRegressor(criterion='poisson', max_depth=5, max_features=2,
                      min_impurity_decrease=0, min_samples_leaf=2)
```

```
y_pred = total_model_02.predict(X_test)
```

Executed at 2024.03.11 08:59:56 in 5ms

```
print(f"R2 SCORE : {r2_score(y_test, y_pred)}")
print(f"MSE : {mean_squared_error(y_test, y_pred)}")
print(f"MAE : {mean_absolute_error(y_test, y_pred)}")
```

Executed at 2024.03.11 08:38:00 in 12ms

R2 SCORE : 0.9007550546385743

MSE : 15435.556477505235

MAE : 91.12809880210177

DecisionTree
+
GridSearchCV

직접 파라미터를 조절한 것에 비해
미세하게 MSE와 MAE 값이 더 높다

5단계 : 모델 저장

DecisionTree

```
import joblib
import os
Executed at 2024.03.11 08:38:09 in 8ms

model_dir = './YSY_model/'
model_filename = model_dir + 'decision_tree_total_02.pkl'

# 저장 폴더 존재 여부 확인 후 저장
if not os.path.exists(model_dir):
    os.mkdir(model_dir)

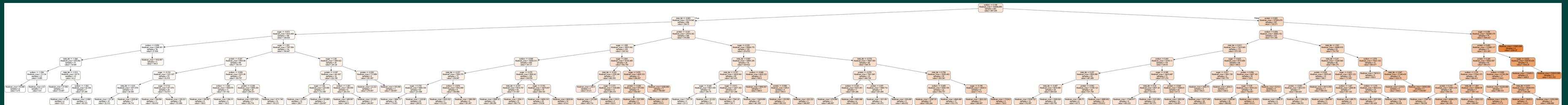
joblib.dump(total_model_01, model_filename)
Executed at 2024.03.11 08:38:18 in 9ms

['./YSY_model/decision_tree_total_02.pkl']
```

직접 파라미터 조정한 모델로
최종 모델 파일로 저장하였다!

6단계 : Decision Tree 시각화 DecisionTree

최종 시각화 트리



Boosting

3단계 : 학습, 평가

Boosting
=> GradientBoostingRegressor

```
1 from sklearn.ensemble import AdaBoostRegressor
2 from sklearn.ensemble import GradientBoostingRegressor
3 from sklearn.ensemble import HistGradientBoostingRegressor
Executed at 2024.03.11 08:16:16 in 113ms
```

```
1 model_01 = AdaBoostRegressor()
2 model_01.fit(X_train, y_train)
3 train_score = model_01.score(X_train, y_train)
4 test_score = model_01.score(X_test, y_test)
5 print(f"[train score] {train_score} [test_score] {test_score}")
Executed at 2024.03.11 08:18:59 in 107ms
```

[train score] 0.7827652801778328 [test_score] 0.8302801791369199

```
1 model_02 = HistGradientBoostingRegressor()
2 model_02.fit(X_train, y_train)
3 train_score = model_02.score(X_train, y_train)
4 test_score = model_02.score(X_test, y_test)
5 print(f"[train score] {train_score} [test_score] {test_score}")
Executed at 2024.03.11 08:18:59 in 218ms
```

[train score] 0.941164256388292 [test_score] 0.8957109263742226

```
1 model_03 = GradientBoostingRegressor()
2 model_03.fit(X_train, y_train)
3 train_score = model_03.score(X_train, y_train)
4 test_score = model_03.score(X_test, y_test)
5 print(f"[train score] {train_score} [test_score] {test_score}")
Executed at 2024.03.11 08:19:00 in 148ms
```

[train score] 0.9510166614281074 [test_score] 0.9549542493779698

- AdaBoostRegressor
- GradientBoostingRegressor
- HistGradientRegressor

따로 패키지를 불러오지 않아도 되는
모듈끼리 비교



GradientBoostingRegressor가 가장
점수차이가 적고 점수대가 높아서 채택함!

- GradientBoostingRegressor가 가장 점수 차이가 적기 때문에 이 모델로 채택했다.

4단계 : 예측&성능평가

Boosting

=> GradientBoostingRegressor

```
y_pred = model_03.predict(X_test)
```

```
y_pred
```

Executed at 2024.03.11 08:19:33 in 11ms

	0
0	530.100510
1	263.268514
2	1962.587307
3	196.232874
4	412.044347
5	648.807265
6	416.668422
7	123.588004
8	50.129619
9	770.827715

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
print(f"R2 SCORE : {r2_score(y_test, y_pred)}")
print(f"MSE : {mean_squared_error(y_test, y_pred)}")
print(f"MAE : {mean_absolute_error(y_test, y_pred)}")
```

Executed at 2024.03.11 08:19:34 in 12ms

R2 SCORE : 0.9549542493779698

MSE : 7005.9611123350405

MAE : 61.159774451321844

R2 Score = 0.95

MSE = 7005

MAE = 61

3단계 : 학습, 평가

Boosting

=> AdaBoostRegressor

학습 및 평가 - decision_tree로 만든 모델을 활용해보기 (AdaBoostRegressor)

```
In 30 1 from joblib import load  
2 model_file = './YSY_model/decision_tree_total_02.pkl'  
3 model = load(model_file)  
Executed at 2024.03.11 08:40:48 in 22ms
```

```
In 31 1 model  
Executed at 2024.03.11 08:40:48 in 11ms
```

```
Out 31 ▾  
      DECISIONTREEREGRESSOR  
DecisionTreeRegressor(criterion='friedman_mse', max_depth=7, max_features=3,  
                      min_impurity_decrease=3, min_samples_leaf=3,  
                      min_samples_split=4)
```

```
In 38 1 boost_model = AdaBoostRegressor(model)  
2 boost_model.fit(X_train, y_train)  
3 train_score = boost_model.score(X_train, y_train)  
4 test_score = boost_model.score(X_test, y_test)  
5 print(f"[train score] {train_score} [test score] {test_score}")  
6  
7 # ADA를 하니 확실히 올라갔다! 최적적합으로 튜닝시켜보면 점수를 더 끌어올릴 수 있지 않을까?  
Executed at 2024.03.11 08:52:51 in 120ms
```

```
[train score] 0.9640746387152226 [test score] 0.9524433885964456
```

앞서 만든 decision tree를
업그레이드 해보자

모델 파일을 불러온다
-> 해당 모델에 ada boost
regressor 적용

[결과]

train : 0.96 test: 0.95

3단계 : 학습, 평가 + 튜닝

Boosting

=> AdaBoostRegressor

+

GridSearchCV

```
1 from sklearn.model_selection import GridSearchCV
2
3 boost_model = AdaBoostRegressor(model)
4
5 param = {
6     'n_estimators':[50,100,150],
7     'learning_rate':[0.1, 0.5, 1.0, 1.5],
8     'loss':['linear', 'square','exponential']
9 }
10
11 cv_num = 5
12
13 gs = GridSearchCV(boost_model, param_grid = param, cv = cv_num, return_train_score = True)
14
15 result = gs.fit(X_train, y_train)
16
17 resultDF = pd.DataFrame(gs.cv_results_)
18
19 print(f"max train score : {max(resultDF['mean_train_score'])}")
20 print(f"max test score : {max(resultDF['mean_test_score'])}\n")
21
22 # 최고 성능의 파라미터, 최고의 점수
23 print(f"best parameters : {gs.best_params_}")
24 print(f"best score : {gs.best_score_}")
25
26 # 튜닝을 하지 않는 것이 더 높은 점수가 나와서 튜닝을 하지 않는 것으로 최종 모델로 결정하겠다.
27 Executed at 2024.03.11 08:50:44 in 32s 271ms
28
29 max train score : 0.9703733436990165
30 max test score : 0.8712063767688042
31
32 best parameters : {'learning_rate': 0.1, 'loss': 'linear', 'n_estimators': 50}
33 best score : 0.8712063767688042
```

train : 0.96 test : 0.95

여기서 튜닝을 통해 더 끌어올려보고자 했다

하지만,,

[결과]

train : 0.97 test : 0.87

과대적합은 되었고 성능은 더 떨어졌다.

그래서 튜닝은 하지 않는것으로!

Boosting

=> AdaBoostRegressor

4단계 : 예측 & 성능 평가

```
1 total_model = boost_model
Executed at 2024.03.11 08:53:01 in 8ms
튜닝을 하지 않은 adaBoost 모델로 설정!
예측

1 y_pred = total_model.predict(X_test)
2 y_pred
Executed at 2024.03.11 08:53:03 in 18ms
▼


|   | 0           |
|---|-------------|
| 0 | 609.369427  |
| 1 | 317.777778  |
| 2 | 1977.000000 |
| 3 | 186.750000  |
| 4 | 393.000000  |
| 5 | 624.230769  |
| 6 | 411.441176  |
| 7 | 167.416667  |
| 8 | 122.666667  |
| 9 | 652.000000  |


```

성능 평가

```
1 1 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
Executed at 2024.03.11 08:53:05 in 10ms
2 1 print(f"R2 SCORE : {r2_score(y_test, y_pred)}")
2 2 print(f"MSE : {mean_squared_error(y_test, y_pred)}")
3 3 print(f"MAE : {mean_absolute_error(y_test, y_pred)}")
Executed at 2024.03.11 08:53:06 in 19ms
▼
R2 SCORE : 0.9524433885964456
MSE : 7396.475039862805
MAE : 64.12161442660035
```

R2 Score = 0.95

MSE = 7396

MAE = 64

grandboosting보다
오차값이 더 크게 나왔다.

5단계 : 모델 저장

Boosting

```
import joblib
import os
Executed at 2024.03.11 08:53:14 in 8ms

model_dir = './YSY_model/'

model_filename = model_dir + 'boosting_total.pkl'

# 저장 폴더 존재 여부 확인 후 저장
if not os.path.exists(model_dir):
    os.mkdir(model_dir)

joblib.dump(model_03, model_filename)
Executed at 2024.03.11 10:27:04 in 58ms

['./YSY_model/boosting_total.pkl']
```

최종 저장한 파일은
Grandboosting한 모델로 하였다.

BEST : decision tree(튜닝0) + adaboost(튜닝X)

- train : 0.96 test : 0.95
- R2 Score = 0.95 MSE = 7396 MAE = 64

5. 모델 분석

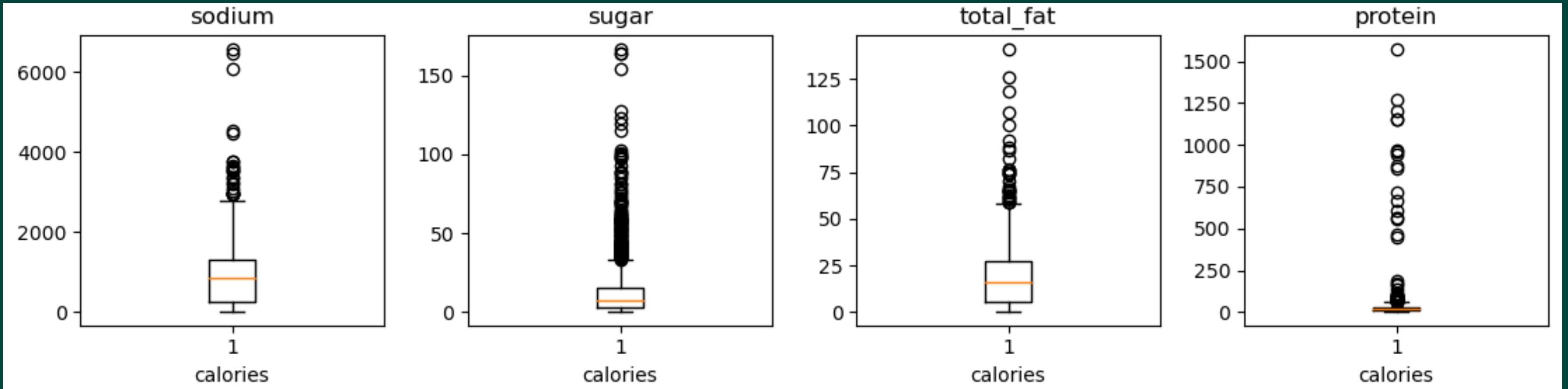
KDT05 박희진

Bagging(Random Forest)
Voting



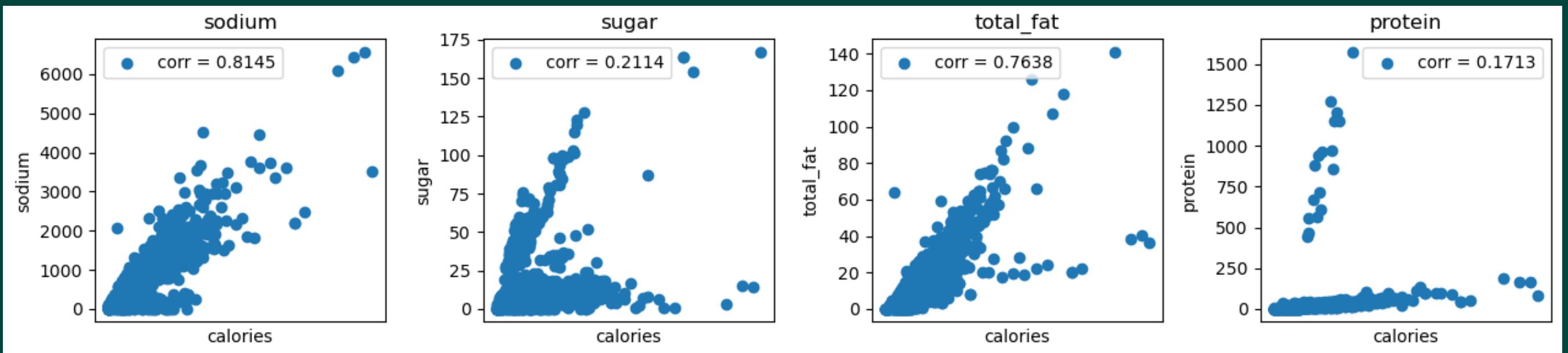
RandomForest model

데이터셋 준비



이상치 많이 존재

제거 X



약간 선형?
2차 곡선?

데이터셋 준비

```
# 피쳐들끼리 상호관계 있는지 확인  
featureDF.corr() # total_fat과 sodium 상관관계 높음  
15ms 2024.03.10 21:20:16에 실행되었습니다
```

	sodium	sugar	total_fat	protein
sodium	1.000000	-0.175234	0.707923	-0.010113
sugar	-0.175234	1.000000	-0.088508	0.064603
total_fat	0.707923	-0.088508	1.000000	0.080772
protein	-0.010113	0.064603	0.080772	1.000000

poly 실행 해보자

데이터셋 준비

```
# total_fat과 sodium_nolu 지제  
selected_features = ['total_fat', 'sodium']  
  
poly = PolynomialFeatures(degree=3, include_bias=False)  
poly_features = poly.fit_transform(featureDF[selected_features])  
  
poly_featureDF = pd.DataFrame(poly_features, columns=[f'total_fat & sodium {i}' for i in range(1,10)])  
  
# 다양한 특성을 포함한 데이터프레임과 원래의 데이터프레임을 합치기  
poly_featureDF = pd.concat([dataDF, poly_featureDF], axis=1)  
poly_featureDF.dropna(inplace=True)  
poly_targetSR = poly_featureDF['calories']  
poly_featureDF = poly_featureDF[poly_featureDF.columns[3:]]
```

더 심해진 과적합

```
model = RandomForestRegressor(random_state=18)  
model.fit(poly_xtrain,ytrain)  
model.score(poly_xtrain,ytrain),model.score(poly_xtest,ytest) # 과적합 더 심함  
1s 389ms 2024.03.10 23:42:48에 실행되었습니다
```

(0.9825548074996646, 0.9374795473398546)

poly 진행 X

데이터셋 준비

```
random_state = 29  
score : 0.9160878170953889
```

train_test_split에서
최적의 **random_state** 값 도출

데이터셋 준비

```
model : RandomForestRegressor()  
scaler : StandardScaler()  
Train score : 0.976202648651411  
Test score : 0.942028578802201
```

[모델 설명도]

R2 : 0.942028578802201

[에러]

MAE : 56.015049401709405

MSE : 7609.1878526749115

RMSE : 87.230658903134

```
model : RandomForestRegressor()  
scaler : MinMaxScaler()  
Train score : 0.981596094217946  
Test score : 0.946574492667642
```

[모델 설명도]

R2 : 0.946574492667642

[에러]

MAE : 54.006795995116

MSE : 7012.502247086662

RMSE : 83.74068453915731

```
model : RandomForestRegressor()  
scaler : RobustScaler()  
Train score : 0.9796202846445624  
Test score : 0.9428358075425068
```

[모델 설명도]

R2 : 0.9428358075425068

[에러]

MAE : 56.26639880341881

MSE : 7503.232970111186

RMSE : 86.62120392901028

MAE와 RMSE가 제일 낮은 MinMaxScaler() 선택

학습 진행

RandomForest 모델에서 최적의 random_state 값 도출

```
random_state_list = []
for i in range(1,50):
    model = RandomForestRegressor(random_state=i)
    model.fit(xtrain, ytrain)
    random_state_list.append(model.score(xtest,ytest))
    max_score = max(random_state_list)
print(f'random_state = {random_state_list.index(max_score)+1}\nscore : {max_score}')
27s 672ms 2024.03.10 23:33:36에 실행되었습니다

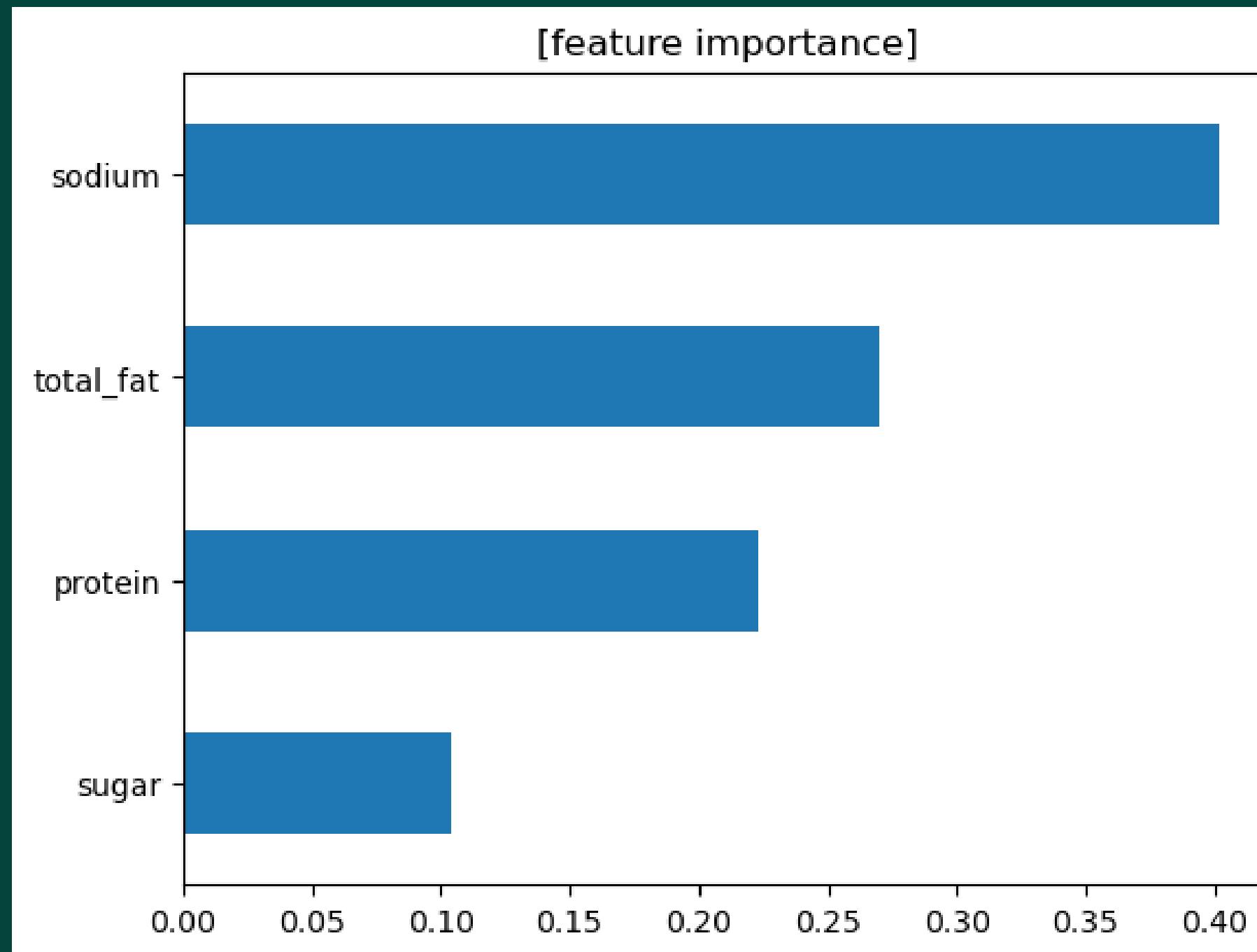
random_state = 18
score : 0.950145702659303
```



```
model = RandomForestRegressor(random_state=18)
model.fit(xtrain, ytrain)
1s 80ms 2024.03.11 10:23:54에 실행되었습니다

▼ RandomForestRegressor
RandomForestRegressor(random_state=18)
```

feature importance



sodium > total_fat > protein > sugar

모델 평가

```
model.score(xtrain,ytrain),model.score(xtest, ytest) # 조금 과적합  
41ms 2024.03.11 10:19:56에 실행되었습니다
```

```
(0.9799029022999607, 0.950145702659303)
```

gridsearchCV() 실행

```
# 랜덤 포레스트 모델 파라미터 조절  
params = { 'n_estimators' : [10,50,100,200,400], # 결정 트리의 개수  
          'max_depth' : [5,10,15,20], # 트리의 최대 깊이 (과적합 제어)  
          'min_samples_split' : [5,10,15,20], # 노드를 분할하기 위한 최소한의 샘플 데이터 수  
          'max_features' : ['sqrt', 'log2', None] # 최적의 분할을 위해 고려할 피쳐의 최대 개수  
          } min_sample_leaf  
cv = GridSearchCV(model, param_grid=params, cv = 5, return_train_score=True, n_jobs=-1)  
cv.fit(xtrain,ytrain)
```

```
(0.9604551139369996, 0.9476294323874069)
```

과대적합 완화

RandomForest model

```
RandomForestRegressor  
RandomForestRegressor(max_depth=10, max_features='sqrt', min_samples_split=5,  
n_estimators=50, random_state=18)
```

```
(0.9604551139369996, 0.9476294323874069)
```

[모델 설명도]

R2 : 0.9476294323874069

[에러]

MAE : 57.07360880221679

MSE : 6874.033423395934

RMSE : 82.9097908777723

Voting model

voting model estimators_

```
▼ KNeighborsRegressor
KNeighborsRegressor(n_neighbors=2, p=1, weights='distance')

▼ LinearRegression
LinearRegression(copy_X=False, fit_intercept=False, n_jobs=100)

▼ Ridge
Ridge(alpha=0.9, max_iter=1000, random_state=28, solver='lsqr')

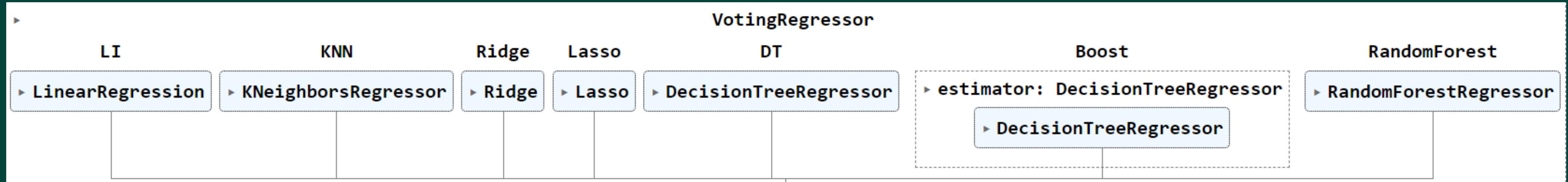
▼ Lasso
Lasso(alpha=1.5, max_iter=10000, positive=True, selection='random', tol=0.01)

▼ DecisionTreeRegressor
DecisionTreeRegressor(criterion='friedman_mse', max_depth=7, max_features=3,
                      min_impurity_decrease=3, min_samples_leaf=3,
                      min_samples_split=4)

▼ RandomForestRegressor
RandomForestRegressor(max_depth=10, max_features='sqrt', min_samples_split=5,
                      n_estimators=50, random_state=18)
```

```
▶ AdaBoostRegressor
▶ estimator: DecisionTreeRegressor
    ▶ DecisionTreeRegressor
```

Voting model



(0.9124816417318052, 0.9120334802958219)

[모델 설명도]

R2 : 0.9120334802958219

[예측]

MAE : 71.659755748153

MSE : 8129.04214048125

RMSE : 90.16120085980027

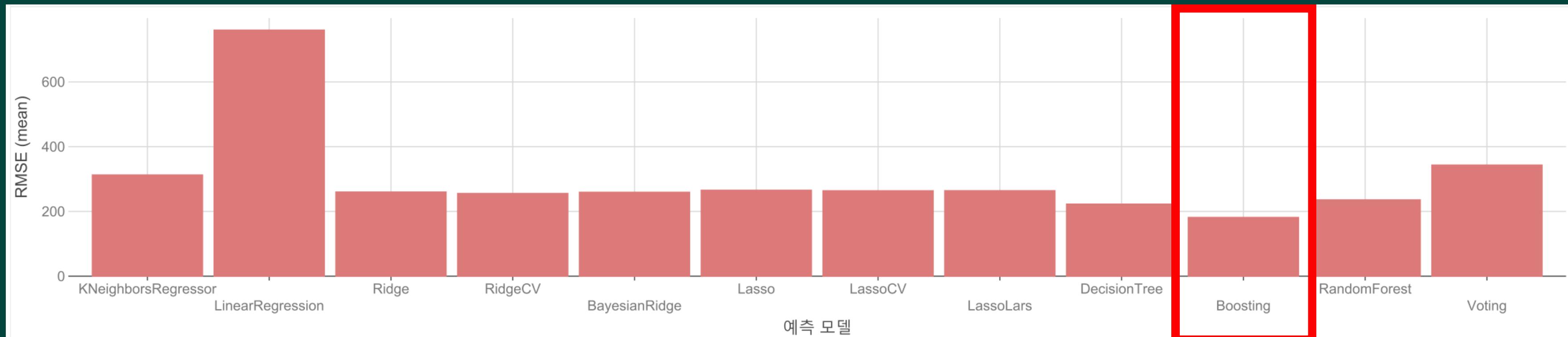
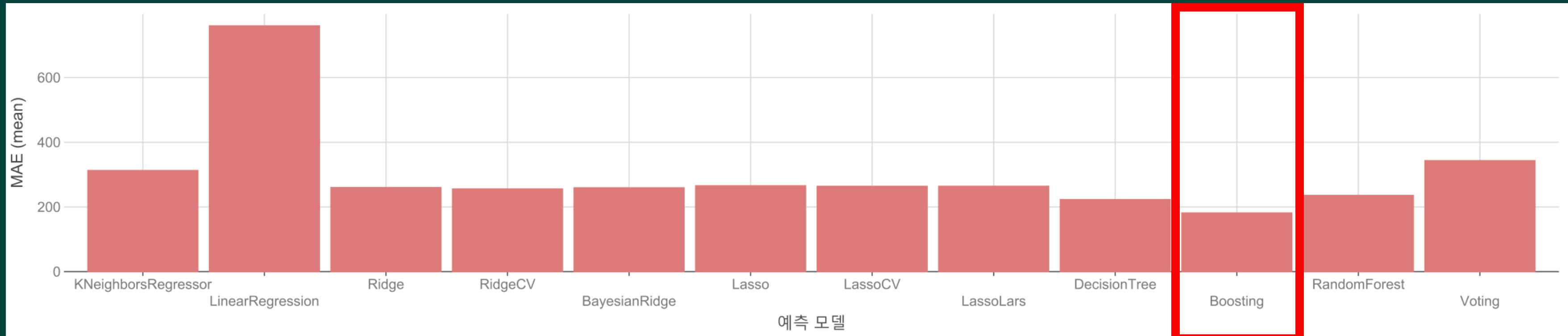
맘스터치 버거로의 데이터로 칼로리 예측

버거류 제품명	단품 중량(g)	칼로리(kcal)	나트륨(mg)	당류(g)	포화지방(g)	단백질(g)
화이트갈릭싸이버거	263	759	1,345(67)	10(10)	13(87)	37(67)

↳ 예측 모델	↳ 예측 값	↳ 실제 값	↳ MAE	↳ RMSE
KNeighborsRegressor	447.292	759	311.707854	311.707854
LinearRegression	0.000	759	759.000000	759.000000
Ridge	499.914	759	259.085587	259.085587
RidgeCV	504.449	759	254.550842	254.550842
BayesianRidge	500.740	759	258.260229	258.260229
Lasso	494.420	759	264.579784	264.579784
LassoCV	496.170	759	262.829565	262.829565
LassoLars	495.930	759	263.070039	263.070039
DecisionTree	537.333	759	221.666667	221.666667
Boosting	578.500	759	180.500000	180.500000
RandomForest	524.245	759	234.754662	234.754662
Voting	416.784	759	342.216032	342.216032

각 모델별 MAE & RMSE

Boosting model이 가장 우수



결론

1. 어떤 모델의 정확도가 제일 높은가?

decision tree (튜닝 O) + AdaBoost(튜닝 X)를 결합한 모델
이 가장 좋았다.

이유: Boosting 알고리즘이 앞서 적용된 모델의 학습 결과의 오답에 대한 가중치를 부여하는 원리이기 때문이다.



2. 어떤 모델의 정확도가 제일 낮은가?

SVM 모델인 NuSVR의 정확도가 제일 낮았
다(22.8%)

이유 : 이상치가 많은 데이터였고, 다차원에
매핑하려다 보니 왜곡이 일어남

3. 그래서, 해당 분석이 의미 있는가?

1. 햄버거의 영양 성분을 이해하고,
2. 건강 관리에 이바지 함
3. 다양한 식품 비교
4. 영양 정보 제공

활용



칼로리 예측을 기반한
햄버거 주천 프로그램

입력순서 : 소금, 설탕, 지방, 단백질

섭취량 기준 : <https://www.khidi.or.kr/kps/dhraStat/result5?menuId=MENU01657&siteId=SITE00002>

설탕 섭취량 : <https://feelgoodpal.com/ko/blog/how-much-sugar-per-day/>

소금	설탕	지방	단백질	=
3299.0	여자(25), 남자(37.5)	47.85	72.4(38.56)	=

남자 기준 섭취량 : 2500

여자 기준 섭취량 : 1800

하루 영양소 섭취 기준과, 칼로리를 반영하여 햄버거 추천 프로그램을 작성

한 끼 식사량 : 1/3

한 끼 칼로리 섭취량(남자) : 2500/3

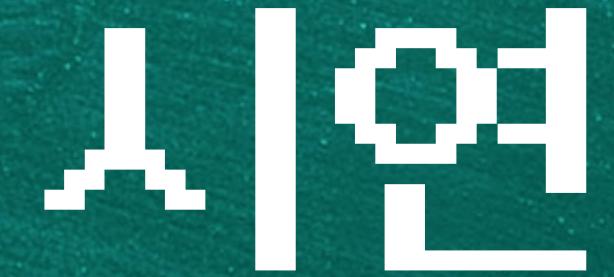
한 끼 칼로리 섭취량(여자) : 1800/3

감자튀김 평균 칼로리 : 250~300

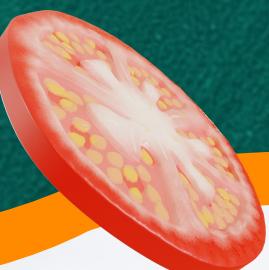
음료수 평균 칼로리 : 0~50

적정 칼로리 섭취량(남자) : $2500/3 - (250\sim300) - (0\sim50) = 480\sim580\text{Kcal}$ 정도, 감자튀김을 포기하면 780~880Kcal 정도

적정 칼로리 섭취량(여자) : $1800/3 - (250\sim300) - (0\sim50) = 250\sim350\text{Kcal}$ 정도, 감자튀김을 포기하면 550~650Kcal 정도



이제 햄버거 좀 먹어보자...



출처

맥도날드 : <https://www.kaggle.com/datasets/mcdonalds/nutrition-facts>

롯데리아 : <https://www.lotteeatz.com/upload/stg/etc/ria/items.html>

미국 전 지점 : <https://www.kaggle.com/datasets/ulrikthygepedersen/fastfood-nutrition>

버거킹 : <https://emilysinglelife.tistory.com/62>

맘스터치(이미지) : <https://www.momstouch.co.kr/m/brand/notice-view.php?idx=49>

노브랜드 : <https://realjace.tistory.com/entry/%EB%85%B8%EB%B8%8C%EB%9E%9C%EB%93%9C%EB%B2%84%EA%B1%BO-%EB%A9%94%EB%89%B4%EB%B3%84-%EC%B9%BC%EB%A1%9C%EB%A6%AC-%EC%98%81%EC%96%91%EC%84%B1%EB%B6%84-%EC%B4%9D%EC%AO%95%EB%A6%AC>

프랭크 버거 : <https://rooftoper.tistory.com/entry/%ED%94%84%EB%9E%AD%ED%81%AC-%EB%B2%84%EA%B1%BO-%EC%98%81%EC%96%91%EC%84%B1%EB%B6%84>

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge

https://scikit-learn.org/stable/modules/generated/sklearn.kernel_ridge.KernelRidge.html#sklearn.kernel_ridge.KernelRidge

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html#sklearn.linear_model.RidgeCV

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html#sklearn.linear_model.BayesianRidge

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsIC.html#sklearn.linear_model.LassoLarsIC

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLars.html#sklearn.linear_model.LassoLars

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html#sklearn.svm.LinearSVR>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html#sklearn.svm.LinearSVR>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html#sklearn.svm.NuSVR>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC>

섭취량 기준 : <https://www.khidi.or.kr/kps/dhraStat/result5?menuId=MENU01657&siteId=SITE00002>

설탕 섭취량 : <https://feelgoodpal.com/ko/blog/how-much-sugar-per-day/>