기계학습

Term Project

회귀(Regression)

한준호
2018741035

1) 데이터 선정

'insurance.csv'

Medical Cost Personal Datasets

Insurance Forecast by using Linear Regression

데이터셋 Link: https://www.kaggle.com/datasets/mirichoi0218/insurance

```
df = pd.read_csv("insurance.csv")
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
df.shape
```

```
(1338, 7)
```

Columns

- age: age of primary beneficiary
- sex: insurance contractor gender, female, male
- bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
- children: Number of children covered by health insurance / Number of dependents
- smoker: Smoking
- region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
- charges: Individual medical costs billed by health insurance

데이터 선정 이유:

Dataset 은 kaggle 의 'Insurance Data'로 선정하였다. dataset 은 age, sex, bmi, children, smoker, region 값에 따른 보험료(charges)에 대한 정보를 담고있다. 이 데이터를 사용하여 분석해보면 하나 또는 둘 이상의 정보에 대하여 보험료(charges)에 상관관계가 존재하고 분석한 상관관계를 이해하고 이용하여 모델링하여 회귀 예측 모델을 구현할 수 있을 것이라고 생각하였다.

2) 데이터 전처리

- **Exploratory Data Analysis**

```
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   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

'Insurance data' dataset 은 1338 개의 data 가 담겨있고 age, sex, bmi, children, smoker, region, charges 로 7 개의 column 으로 구성되어 있다. age, bmi, children, charges 정보의 Dtype 은 'int' 나 'float' 이지만, sex, smoker, region 정보의 Dtype 은 'object'인 것을 알 수 있다.

```
df.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

Dtype 이 'int' 나 'float'인 data 들의 각종 통계량을 요약해서 보았다.

```
df.isnull().sum()
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

Missing data 가 없는 것을 확인하였다.

- Encoding

```
df.sex.value_counts()
```

```
female    547
male      517
Name: sex, dtype: int64
```

```
# Encoding for Sex
value1 = {'male': 1, 'female': 0}
df['sex'].replace(value1,inplace = True)
df.head()
```

	age	sex	bmi	children	region	charges
1	18	1	33.770	1	southeast	1725.55230
2	28	1	33.000	3	southeast	4449.46200
3	33	1	22.705	0	northwest	21984.47061
4	32	1	28.880	0	northwest	3866.85520
5	31	0	25.740	0	southeast	3756.62160

```
df.smoker.value_counts()
```

```
no      1064
yes      274
Name: smoker, dtype: int64
```

```
value2 = {"yes": 1,"no": 0}
df['smoker'].replace(value2,inplace= True)
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	southwest	16884.92400
1	18	1	33.770	1	0	southeast	1725.55230
2	28	1	33.000	3	0	southeast	4449.46200
3	33	1	22.705	0	0	northwest	21984.47061
4	32	1	28.880	0	0	northwest	3866.85520

```
df.region.value_counts()
```

```
southeast     273
northwest     267
southwest     267
northeast     257
Name: region, dtype: int64
```

```
value3 = {"northeast": 0,"northwest": 1,
          "southwest": 3, "southeast": 4}
df['region'].replace(value3,inplace= True)
df.head()
```

	age	sex	bmi	children	region	charges
1	18	1	33.770	1	4	1725.55230
2	28	1	33.000	3	4	4449.46200
3	33	1	22.705	0	1	21984.47061
4	32	1	28.880	0	1	3866.85520
5	31	0	25.740	0	4	3756.62160

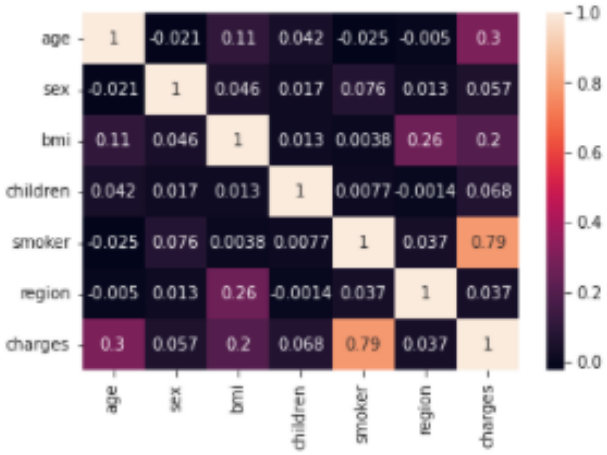
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1064 entries, 1 to 1336
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1064 non-null   int64
 1   sex       1064 non-null   int64
 2   bmi       1064 non-null   float64
 3   children  1064 non-null   int64
 4   region    1064 non-null   int64
 5   charges   1064 non-null   float64
dtypes: float64(2), int64(4)
memory usage: 58.2 KB
```

사이킷런 알고리즘은 문자열 값을 입력 값으로 허용하지 않기 때문에, Dtype 이 'object' 인 데이터들을 Label Encoding 해주었다.

```
sns.heatmap(df.corr(),annot = True)
```

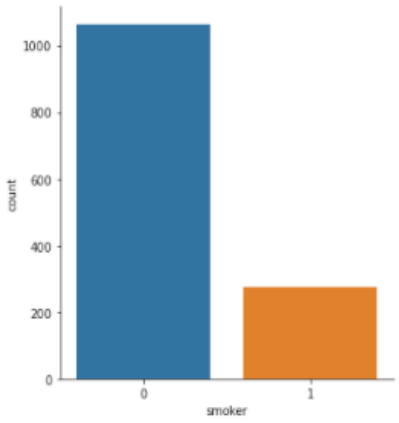
```
<AxesSubplot:>
```



본격적으로 데이터를 전처리하기 전에 데이터의 상관관계에 따라 데이터를 전처리 해주기 위해 heatmap 을 통해 각 변수 간에 상관관계를 표로 나타냈다. smoker, age, bmi 데이터 순으로 charges 데이터와 유의미한 상관관계를 보이고 있었고, 특히 smoker 가 가장 강한 상관관계를 보이고 있었고 region 이 거장 약한 상관관계를 보이고 있었다. 추가적으로 region 과 bmi 간에 상관관계를 확인할 수 있었다.

```
sns.catplot(x='smoker',kind = 'count',data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f77c9264b00>
```



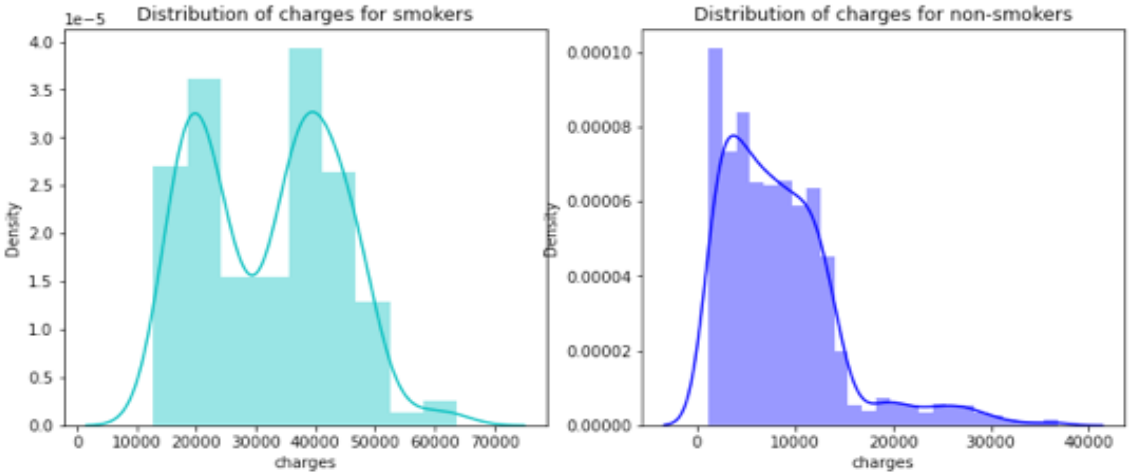

```
df.smoker.value_counts()
```

```
no     1064
yes     274
Name: smoker, dtype: int64
```

smoker 데이터가 charges 데이터와 강한 상관관계를 보이고 있기 때문에 smoker 의 분포를 확인해보았다. non-smoker(0)는 1064 명 smoker(1) 274 명으로 약 4 배 정도의 차이를 확인할 수 있었다. 이는 smoker 데이터의 불균형을 확인할 수 있었다.

```
f= plt.figure(figsize=(12,5))

ax=f.add_subplot(121)
sns.distplot(df[(df.smoker == 1)]["charges"],color='c',ax=ax)
ax.set_title('Distribution of charges for smokers')

ax=f.add_subplot(122)
sns.distplot(df[(df.smoker == 0)]['charges'],color='b',ax=ax)
ax.set_title('Distribution of charges for non-smokers')
```

```
Text(0.5, 1.0, 'Distribution of charges for non-smokers')
```



smoker 와 non-smoker 각각에 데이터에 대한 charges 의 분포를 확인하였다. smoker 데이터에 대한 charges 의 분포에서 정규 분포 형태가 아닌 두개의 봉우리로 나오는 것을 확인하였다.

```
mask = df['smoker'].isin(['yes'])
df = df[~mask]
df =df.drop('smoker',axis=1)
df.head()
```

	age	sex	bmi	children	region	charges
1	18	male	33.770	1	southeast	1725.55230
2	28	male	33.000	3	southeast	4449.46200
3	33	male	22.705	0	northwest	21984.47061
4	32	male	28.880	0	northwest	3866.85520
5	31	female	25.740	0	southeast	3756.62160

```
df.reset_index()
```

	index	age	sex	bmi	children	region	charges
0	1	18	1	33.770	1	4	1725.55230
1	2	28	1	33.000	3	4	4449.46200
2	3	33	1	22.705	0	1	21984.47061
3	4	32	1	28.880	0	1	3866.85520
4	5	31	0	25.740	0	4	3756.62160
...	...	...	...	...	...	...	...
1059	1332	52	0	44.700	3	3	11411.68500
1060	1333	50	1	30.970	3	1	10600.54830
1061	1334	18	0	31.920	0	0	2205.98080
1062	1335	18	0	36.850	0	4	1629.83350
1063	1336	21	0	25.800	0	3	2007.94500

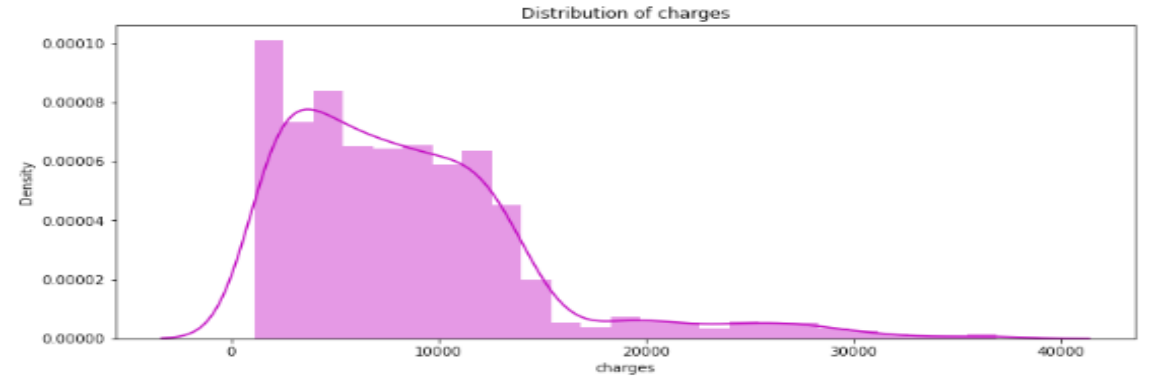
1064 rows × 7 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1064 entries, 1 to 1336
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1064 non-null   int64
 1   sex       1064 non-null   object
 2   bmi       1064 non-null   float64
 3   children  1064 non-null   int64
 4   region    1064 non-null   object
 5   charges   1064 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 58.2+ KB
```

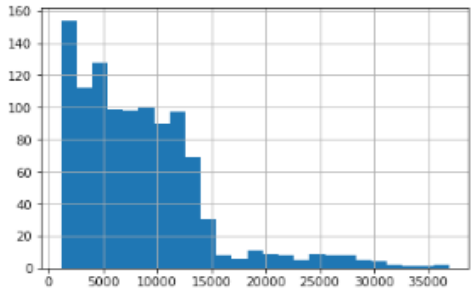
smoker 데이터의 불균형은 회귀 성능에 영향을 주고, smoker 에 대한 charges 에 분포도 정규분포가 아니므로, 'yes'인 smoker 데이터 값을 이상치로 판단하였고 이상치를 제거하여 non-smoker 의 데이터만 남겨주었고 smoker column 을 drop 했다. 이로 인해, 1064 개에 데이터가 남은 것을 확인하였다.

```
plt.figure(figsize=(12,5))
plt.title("Distribution of charges")
ax = sns.distplot(df["charges"], color = 'm')
```

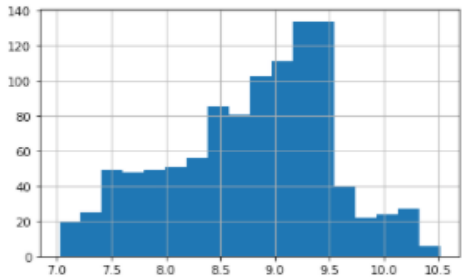


타겟 값인 charges 의 분포를 보았다. charges 데이터는 왜곡된(살짝 왼쪽으로 쏠린) 정규 분포 형태를 보였다. 선형 회귀 모델을 훈련할 때 특징과 타겟이 정규 분포일 필요는 없지만, 정규 분포 가정은 특정 통계와 가설 검증에 필요하다.

```
df['charges'].hist(bins='auto')
```

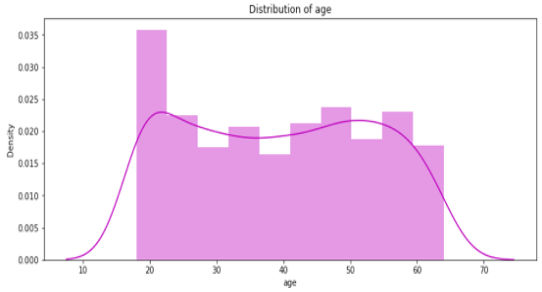



```
y_log = np.log1p(df['charges'])
y_log.hist(bins='auto')
```

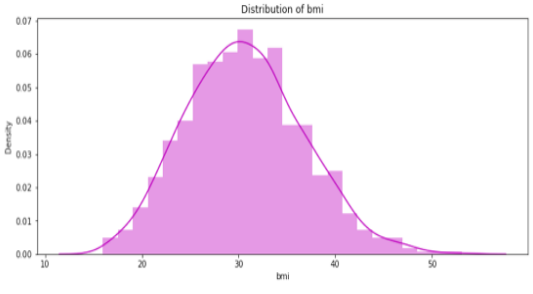



charges 데이터가 왜곡된 정규분포 형태를 보이는 것은 피처 간의 단위가 크게 차이 나기 때문이라는 것을 유추할 수 있었다. 위와 같이 로그 변환을 통해 정규화해 정규 분포 형태로 바꿔주었다.

```
plt.figure(figsize=(12,5))
plt.title("Distribution of age")
ax = sns.distplot(df["age"], color = 'm')
```




```
plt.figure(figsize=(12,5))
plt.title("Distribution of bmi")
ax = sns.distplot(df["bmi"], color = 'm')
```

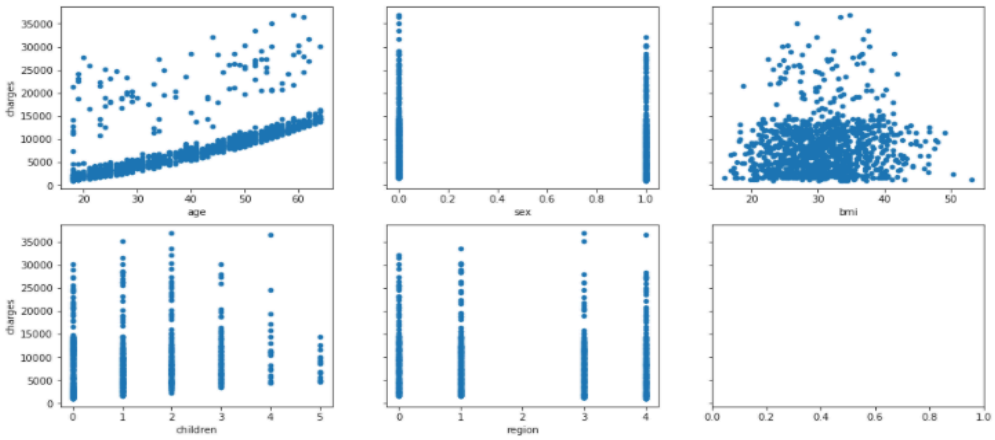


순서대로 age, bmi 의 분포도이다. 완전한 정규 분포는 아니지만 비슷한 모양을 보이고 있으므로 데이터 변환은 생략하였다.

3) 회귀 및 파라미터 최적화 + 결과 및 분석

```
fig, axs = plt.subplots(2,3,sharey=True)
df.plot(kind='scatter', x='age', y='charges', ax=axs[0][0], figsize=[16,8])
df.plot(kind='scatter', x='sex', y='charges', ax=axs[0][1])
df.plot(kind='scatter', x='bmi', y='charges', ax=axs[0][2])
df.plot(kind='scatter', x='children', y='charges', ax=axs[1][0])
df.plot(kind='scatter', x='region', y='charges', ax=axs[1][1])
```

```
<AxesSubplot:xlabel='region', ylabel='charges'>
```



본격적으로 회귀 모델을 모델링하기 전에 age, sex, bmi, children, region 과 charges 의 관계를 나타냈다. 결과를 보면 age 와는 강한 선형관계를 보이고 있고, bmi 와는 약한 선형관계, sex, children, region 과는 선형관계가 보이지 않았다.

〈 단순 선형 회귀 (Simple Linear Regression) 〉

- 두 변수 간의 선형 관계를 분석

1. 경사하강법 (Gradient Descent) - 행렬 계산 풀이

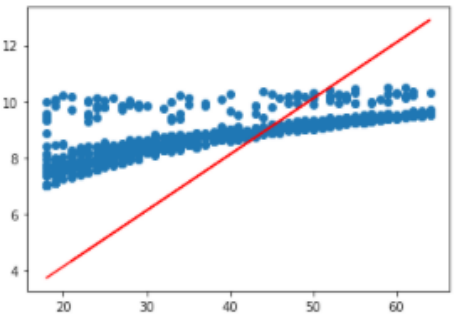
• 입력: age

```
# 경사하강법 - 행렬 풀이
X = df.age
y = y_log

def get_cost(y, y_pred):
    N = len(y)
    cost = np.sum(np.square(y - y_pred)) / N #손실 함수 or 비용 함수
    return cost

def get_weight_updates(w1, w0, X, y, learning_rate=0.0001):
    N = len(y)
    y_pred = np.dot(X, w1) + w0
    print(get_cost(y, y_pred))
    diff = y_pred - y
    ones = np.ones((N, 1))
    w1_update = learning_rate * 2 * np.dot(X.T, diff) / N
    w0_update = learning_rate * 2 * np.dot(ones.T, diff) / N
    return w1_update, w0_update

def gradient_descent_steps(X, y, iters=10000):
    w0 = 0
    w1 = 0
    for _ in range(iters):
        w1_update, w0_update = get_weight_updates(w1, w0, X ,y)
        w1 = w1 - w1_update
        w0 = w0 - w0_update
    return w1, w0

w1, w0 = gradient_descent_steps(X, y, 1000)
print(w1, w0)
```

```
5.711467558987967
5.711218131134546
5.710968714576686
5.710719309313877
5.710469915345604
5.710220532671357
5.709971161290626
5.709721801202899
5.709472452407662
5.709223114904406
5.708973788692617
5.708724473771789
5.708475170141407
5.708225877800960 5
5.707976596749936
5.707727326987827
5.707478068514115
5.707228821328297
5.706979585429858
0.19893491256751464 [0.1641881]
```

```
y_pred1 = w1 * X + w0
plt.scatter(X, y)
plt.plot(X, y_pred1, c='red')
```

```
[<matplotlib.lines.Line2D at 0x7f77cb165470>]
```



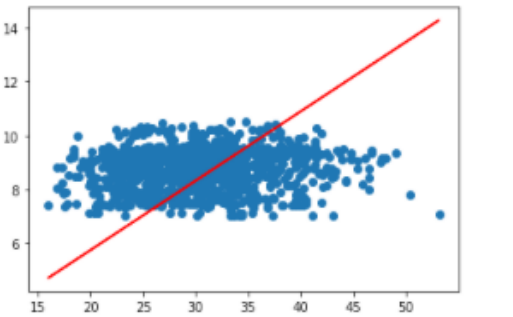
• 입력: bmi

```
# 경사하강법 - 행렬 풀이
X = df.bmi
y = y_log

def get_cost(y, y_pred):
    N = len(y)
    cost = np.sum(np.square(y - y_pred)) / N #손실 함수 or 비용 함수
    return cost

def get_weight_updates(w1, w0, X, y, learning_rate=0.001):
    N = len(y)
    y_pred = np.dot(X, w1) + w0
    print(get_cost(y, y_pred))
    diff = y_pred - y
    ones = np.ones((N, 1))
    w1_update = learning_rate * 2 * np.dot(X.T, diff) / N
    w0_update = learning_rate * 2 * np.dot(ones.T, diff) / N
    return w1_update, w0_update

def gradient_descent_steps(X, y, iters=10000):
    w0 = 0
    w1 = 0
    for _ in range(iters):
        w1_update, w0_update = get_weight_updates(w1, w0, X ,y)
        w1 = w1 - w1_update
        w0 = w0 - w0_update
    return w1, w0

w1, w0 = gradient_descent_steps(X, y, 1000)
print(w1, w0)
```

```
2.8454699129294876
2.8451268112017885
2.844783760724084
2.8444407614887153
2.8440978134880313
2.8437549167143765
2.8434120711601003
2.843069276817551
2.842726533679079
2.8423838417370386
2.8420412009837785
2.8416986114411653
2.8413560730130 19
2.8410135857802326
2.84067114970565
2.8403287647816287
2.83998643100053
2.8396441483547137
2.839301916836543
0.25709395028071336 [0.61572449]
```

```
y_pred3 = w1 * X + w0
plt.scatter(X, y)
plt.plot(X, y_pred3, c='red')
```

```
[<matplotlib.lines.Line2D at 0x7f77cb35c7f0>]
```



age 와 charges, bmi 와 charges 가 선형적인 관계를 이루고 있기 때문에 행렬 계산을 이용한 경사하강법을 이용하여 회귀모델을 모델링하였다. wo 와 w1 의 초기값을 0 으로 설정해주었고 learning rate 를 바꿔주면서 행렬 계산 풀이를 이용한 회귀를 10000 번 반복하면서 RSS(Residual Sum of Square)를 최소로 가지는 회귀계수 w0, w1 을 찾아주었지만 learning rate 의 값이 크면 overshooting 이 일어나 최소점을 찾지 못하는 것을 확인했다. 그렇기 때문에 적절한 learning rate 를 설정해주는 것이 어려웠고 결과도 만족스럽지 못했다.

2. 경사하강법 (Gradient Descent) - 편미분 방정식 풀이

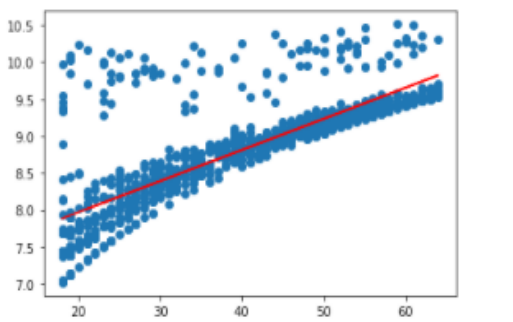
• 입력: age

```
# 경사하강법 - 편미분 방정식 풀이
X = df.age
y = y_log

def get_cost(y, y_pred):
    N = len(y)
    cost = np.sum(np.square(y-y_pred))/N
    return cost

def get_weight_updates(w1, w0, X, y):
    N = len(y)
    y_pred =np.sum(X*y)+w0
    print(get_cost(y, y_pred))
    xy_bar = np.sum(X*y)/N
    y_bar = np.sum(y)/N
    x_bar = np.sum(X)/N
    xx_bar =np.sum(X*X)/N
    return xy_bar, y_bar ,x_bar, xx_bar

def gradient_descent_steps(X, y, iters=10000):
    w0 = 0
    w1 = 0
    for _ in range(iters):
        xy_bar, y_bar, x_bar, xx_bar = get_weight_updates(w1, w0, X, y)
        w1 =(xy_bar - x_bar*y_bar) / (xx_bar - x_bar*x_bar)
        w0 = y_bar - w1*x_bar
    return w1, w0

w1, w0 = gradient_descent_steps(X, y, 1000)
print(w1, w0)
```

```
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
142235665146.77597
0.041998565636998615 7.134308287966998
```

```
y_pred2 = w1 * X + w0
plt.scatter(X, y)
plt.plot(X, y_pred2, c='red')
```

```
[<matplotlib.lines.Line2D at 0x7f77c9ed50f0>]
```



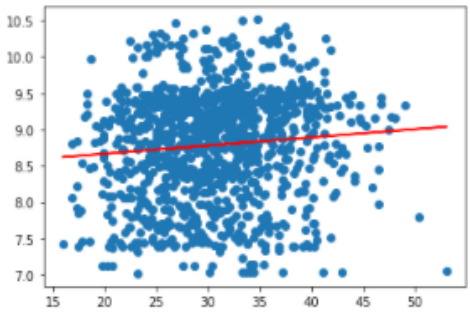
• 입력: bmi

```
# 경사하강법 - 편미분 방정식 풀이
X = df.bmi
y = y_log

def get_cost(y, y_pred):
    N = len(y)
    cost = np.sum(np.square(y-y_pred))/N
    return cost

def get_weight_updates(w1, w0, X, y):
    N = len(y)
    y_pred =np.sum(X*y)+w0
    print(get_cost(y, y_pred))
    xy_bar = np.sum(X*y)/N
    y_bar = np.sum(y)/N
    x_bar = np.sum(X)/N
    xx_bar =np.sum(X*X)/N
    return xy_bar, y_bar ,x_bar, xx_bar

def gradient_descent_steps(X, y, iters=10000):
    w0 = 0
    w1 = 0
    for _ in range(iters):
        xy_bar, y_bar, x_bar, xx_bar = get_weight_updates(w1, w0, X, y)
        w1 =(xy_bar - x_bar*y_bar) / (xx_bar - x_bar*x_bar)
        w0 = y_bar - w1*x_bar
    return w1, w0

w1, w0 = gradient_descent_steps(X, y, 1000)
print(w1, w0)
```

```
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
82402524881.28322
0.011257092852335612 8.44338590194584
```

```
y_pred4 = w1 * X + w0
plt.scatter(X, y)
plt.plot(X, y_pred4, c='red')
```

```
[<matplotlib.lines.Line2D at 0x7f77cc812dd8>]
```



learning rate 를 직접 결정하는데 어려움을 느꼈기 때문에 편미분 방정식 풀이를 이용한 경사하강법을 이용하여 회귀모델을 모델링하였다. wo 와 w1 의 초기값을 0 으로 설정해주었고 편미분 방정식을 이용한 회귀를 10000 번 반복하면서 RSS(Residual Sum of Square)를 최소로 가지는 회귀계수 w0, w1 을 찾아주었다. 구한 선형식을 시각화하였다.

3. 사이킷런 - LinearRegression 알고리즘

회귀 평가 지표

– MAE(Mean Absolute Error) $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$

– MSE(Mean Squared Error) $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

– RMSE(Root Mean Squared Error) $RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$

– R^2 : 분산 기반 예측 성능, 1에 가까울수록 예측 정확도 높음

$$R^2 = \frac{\text{예측 값 분산}}{\text{실제 값 분산}}$$

• 입력: age

```
y = y_log
X = df[['age']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_hat = lr.predict(X_test)
mae = mean_absolute_error(y_test, y_hat)
mse = mean_squared_error(y_test, y_hat)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_hat)
print(mae, mse, rmse, r2)
```

```
0.28357719593254105 0.20859747965131709 0.4567247307200663 0.6423404230862009
```

Train 데이터와 test 데이터를 각각 80%, 20%로 나누고 사이킷런의 LinearRegression 클래스를 사용하여 단순 선형 회귀를 진행하였다. age 와 charges 가 선형적인 관계를 이루고 있기 때문에 선형 관계를 모델링 하는 알고리즘인 Linear Regression 을 사용하였다. . y 절편을 구할 것이기 때문에 fit_intercept 는 기본 값인 True 를 사용하였고, normalize 또한 기본 값인 False 를 사용하여 입력데이터 정규화는 진행하지 않았다. 회귀 성능 확인하기 위해 MAE, MSE, RMSE, RMSE, R^2 값을 확인하였다. 분산 기반으로 예측 성능을 평가하는 R^2 값이 약 0.6423 으로 1 에 가까운 값이었지만 매우 좋은 예측 정확도를 보이진 않은 것 같다.

```
print(lr.intercept_)
print(lr.coef_)
```

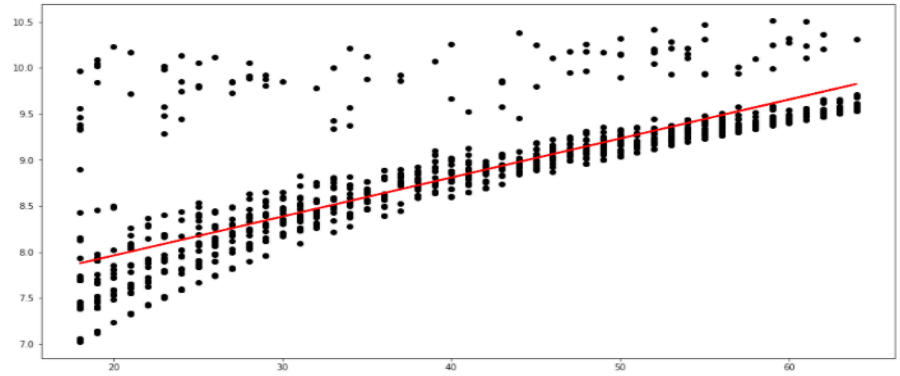
```
7.114295917123479
[0.04235106]
```

절편 값과 기울기를 확인해보았다. 절편 값은 약 7.1143, 기울기는 약 0.0424 이 나왔다.

$$y = w_0 + w_1 x$$

위 선형식에서 lr.coef_ 값은 여기서 w1 을 의미하고 lr.intercept 는 절편이므로 w0 를 의미한다.

```
plt.figure(figsize=(16,8))
plt.scatter(df['age'], y_log, color='black')
plt.plot(X_test, y_hat, c='red', linewidth=2)
plt.show()
```



구한 선형식을 시각화 해보았는데, 선형적으로 증가하는 직선을 확인했다.

• 입력: bmi

```
y = y_log
X = df[['bmi']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_hat = lr.predict(X_test)
mae = mean_absolute_error(y_test, y_hat)
mse = mean_squared_error(y_test, y_hat)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_hat)
print(mae, mse, rmse, r2)
```

```
0.6089673543144983 0.5523532432468008 0.7432047115343126 -0.014902008830834479
```

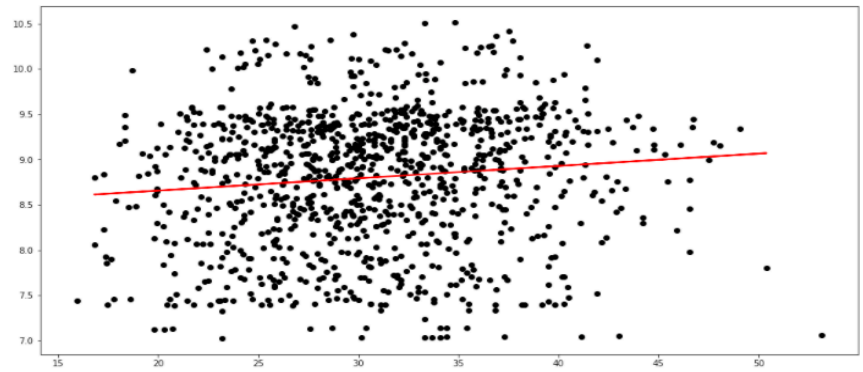
age 데이터와 마찬가지로, Train 데이터와 test 데이터를 각각 80%, 20%로 나누고 LinearRegression 클래스를 사용하여 단순 선형 회귀를 진행하였다. bmi 와 charges 가 선형적인 관계를 이루고 있기 때문에 선형 관계를 모델링 하는 알고리즘인 Linear Regression 을 사용하였다. . y 절편을 구할 것이기 때문에 fit_intercept 는 기본 값인 True 를 사용하였고, normalize 또한 기본 값인 False 를 사용하여 입력데이터 정규화는 진행하지 않았다. 회귀 성능을 확인하기 위해 MAE, MSE, RMSE, RMSE, R^2 값을 확인하였다. MAE, MSE, RMSE 값을 보면 age 데이터 오차가 큰 것을 확인할 수 있었다. 예외적으로, R^2 값은 음수가 나왔는데 이는 회귀모델의 성능이 평균값으로 예측하는 값보다 결과가 좋지 않았음을 의미한다.

```
print(lr.intercept_)
print(lr.coef_)
```

```
8.38307263204801
[0.01361799]
```

절편 값과 기울기를 확인해보았다. 절편 값은 약 8.3831, 기울기는 약 0.0136 이 나왔다.

```
plt.figure(figsize=(16,8))
plt.scatter(df['bmi'], y_log, color='black')
plt.plot(X_test, y_hat, c='red', linewidth=2)
plt.show()
```



구한 선형식을 시각화 해보았는데, 데이터 자체가 약한 선형관계를 보이고 있었기 때문에 오차가 매우 큰 데이터들도 있었지만 선형적으로 증가하는 직선을 확인했다.

〈 다중 선형 회귀 (Multiple Linear Regression) 〉

- 두 개 이상의 독립 변수들과 하나의 종속 변수의 관계를 분석.
- 다중 회귀 분석은 다중의 독립 변수가 있는 형태.

• 입력: age, sex, bmi, children, region (독립 변수 5 개)

```
y = y_log
X = df.drop(['charges'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_hat = lr.predict(X_test)
mae = mean_absolute_error(y_test, y_hat)
mse = mean_squared_error(y_test, y_hat)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_hat)
print(mae, mse, rmse, r2)
```

```
0.20380514443455314 0.1631885868288038 0.4039660713832336 0.6930177351671973
```

Train 데이터와 test 데이터를 각각 80%, 20%로 나누고 LinearRegression 클래스를 사용하여 다중 선형 회귀를 진행하였다. 특징과 타겟 값인 charges 가 선형적인 관계를 이루고 있기 때문에 선형 관계를 모델링 하는 알고리즘인 Linear Regression 을 사용하였다. . y 절편을 구할 것이기 때문에 fit_intercept 는 기본 값인 True 를 사용하였고, normalize 또한 기본 값인 False 를 사용하여 입력데이터 정규화는 진행하지 않았다. 회귀 성능을 확인하기 위해 MAE, MSE, RMSE, RMSE, R^2 값을 확인하였다. 분산 기반으로 예측 성능을 평가하는 R^2 값이 약 0.69 으로 1 에 가까운 값이었지만 매우 좋은 예측 정확도를 보이진 않은 것 같다.

```
print(lr.intercept_)
print(lr.coef_)
```

```
7.125008830929001
[ 0.04154804 -0.09344081  0.00092231  0.12443965 -0.04552749]
```

절편과 계수를 확인해보았다. 절편 값은 약 7.1251, 계수는 age, sex, bmi, children, region 5 개 였기 때문에 5 개의 데이터를 가진 행렬로 출력되었다.

$$y = W_1x_1 + W_2x_2 + \ldots W_nx_n + b$$

위 식에서 lr.coef_ 행렬의 세 값은 여기서 각각 W1, W2, W3, W4, W5 를 의미하고 lr.intercept 는 절편이므로 b 를 의미한다.

2D 그래프에서 여러 변수의 예측을 시각화 할 수 없으므로 시각화는 수행하지 않았다.

• 입력: age, sex, bmi, children (독립 변수 4 개)

```
y = y_log
X = df.drop(['charges','region'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_hat = lr.predict(X_test)
mae = mean_absolute_error(y_test, y_hat)
mse = mean_squared_error(y_test, y_hat)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_hat)
print(mae, mse, rmse, r2)
```

```
0.1996650782950398 0.1179043749624158 0.34337206491270633 0.773427602178555
```

입력을 region 만 빼고 위와 같은 방식으로 다중 선형 회귀를 진행하였다. R^2 값이 약 0.77 로 1 에 가까운 값이 나왔다. 입력에 region 을 포함했을 때보다 포함하지 않았을 때 회귀 성능이 높은 것을 확인할 수 있었다

```
print(lr.intercept_)
print(lr.coef_)
```

```
7.066951629611735
[ 0.04111811 -0.10057288  0.00091551  0.12559842]
```

절편 값과 계수를 확인해보았다. 절편 값은 약 7.1251, 계수는 age, sex, bmi, children 4 개 였기 때문에 4 개의 데이터를 가진 행렬로 출력되었다.

2D 그래프에서 여러 변수의 예측을 시각화 할 수 없으므로 시각화는 수행하지 않았다.

• 단순 선형 회귀, 다중 선형 회귀 결과 및 분석
- 사이킷런의 LinearRegression 클래스를 사용하여 선형회귀를 진행,
- 회귀 성능 평가 지표: MAE, MSE, RMSE, RMSE, R^2

	독립 변수 개수	회귀 성능 평가 지표	회귀 성능
단순 선형 회귀	1 개 (age)	R^2	약 0.6423
단순 선형 회귀	1 개 (bmi)	R^2	약 -0.149
다중 선형 회귀	5 개(age, sex, bmi, children, region)	R^2	약 0.6930
다중 선형 회귀	4 개(age, sex, bmi, children)	R^2	약 0.7734

회귀 성능: Linear Regression 다중 선형 회귀(입력 4 개) > Linear Regression 다중 선형 회귀(입력 5 개) > Linear Regression 단순 선형 회귀(age) > Linear Regression 단순 선형 회귀(bmi)

입력이 4 개인 다중 선형 회귀의 성능이 약 0.7734 로 가장 좋은 것을 확인할 수 있었다. 독립 변수의 개수가 5 개일 때가 4 개일 때보다 성능이 낮은 것은 독립 변수 개수가 많을 경우 적절한 회귀 모형을 선택하지 못하는 문제가 발생했기 때문이라고 판단할 수 있었다.

< 다항 회귀 (Polynomial Regression) >
- 다항 회귀 분석은 독립변수의 차수를 높이는 형태.

```
y = y_log
X = df1.drop(['charges', 'region'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(x_pol, y, test_size=0.2)

pol = PolynomialFeatures (degree = 2)
x_pol = pol.fit_transform(X)
Pol_reg = LinearRegression()
Pol_reg.fit(X_train, y_train)
y_test_pred = Pol_reg.predict(X_test)
mae = mean_absolute_error(y_test, y_test_pred)
mse = mean_squared_error(y_test, y_test_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_test_pred)
print(mae, mse, rmse, r2)
```

```
0.21470396449595094 0.16357635022045586 0.4044457321080986 0.6992318744419972
```

Train 데이터와 test 데이터를 각각 80%, 20%로 나누고 PolynomialFeatures 클래스를 사용하여 다항 회귀를 진행하였다. 위에서 했던 회귀 모델 중 성능이 가장 좋았던 입력이 age, sex, bmi, children 인 다중 선형 회귀와 비교를 위해 입력을 동일하게 설정해주었고 차수는 2 차로 설정하였다. 회귀 성능을 확인하기 위해 MAE, MSE, RMSE,

RMSE, R^2 값을 확인하였다. 분산 기반으로 예측 성능을 평가하는 R^2 값이 약 0.6992 로 1 에 가까운 값이었지만 매우 좋은 예측 정확도를 보이진 않은 것 같다.

```
print(Pol_reg.intercept_)
print(Pol_reg.coef_)
```

```
6.248872524743888
[ 0.00000000e+00  6.49558209e-02 -1.48534306e-01  2.72560290e-02
  3.33640180e-01 -3.10190976e-04  5.76231361e-03  8.64535221e-05
 -5.19044516e-03 -1.48534306e-01 -1.87360790e-03  1.95925617e-02
 -4.96802436e-04  1.22088030e-03 -2.08487922e-02]
```

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$

절편 값과 계수를 확인해보았다.

다항식의 차수가 높을수록 복잡한 피처 간의 관계까지 모델링이 가능하지만 차수가 높을 경우 과적합 문제가 발생할 수 있다.

• LinearRegression 다중 선형 회귀, 다항 회귀 결과 및 분석
- 회귀 성능 평가 지표: MAE, MSE, RMSE, RMSE, R^2

	독립 변수 개수	회귀 성능 평가 지표	회귀 성능
다중 선형 회귀	5 개(age, sex, bmi, children, region)	R^2	약 0.6930
다중 선형 회귀	4 개(age, sex, bmi, children)	R^2	약 0.7734
다항 회귀	4 개(age, sex, bmi, children) (다항식 차수 2 차)	R^2	약 0.6992

회귀 성능: Linear Regression 다중 선형 회귀(입력 4 개) > 다항 회귀 > Linear Regression 단순 선형 회귀(bmi) > Linear Regression 다중 선형 회귀(입력 5 개)

〈 릿지 (Ridge) 회귀 〉

```
y = y_log
X = df.drop(['charges','region'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

규제 선형 모델로 모델로 alpha 값으로 패널티를 부여하여 회귀 계수 값의 크기를 감소하여 과적합을 개선하는 Ridge, Lasso 회귀를 진행하기 전에, 위에서 했던 회귀 모델 중 성능이 가장 좋았던 입력이 age, sex, bmi, children 인 다중 선형 회귀와 비교를 위해 입력(X)은 age, sex, bmi, children 로 설정해주었고, 출력(y)은 위에서 charges 데이터를 로그 변환을 통해 정규화한 값으로 설정해주었다.

```
parameters = {'alpha': [0.1, 1, 10, 20, 100, 1000]}
for alpha in parameters:
    ridge = Ridge(alpha=alpha)
    ridgeCV = GridSearchCV(ridge, parameters,
                           scoring='neg_mean_squared_error', cv=5)
    ridgeCV.fit(X_train, y_train)
    print(ridgeCV.best_params_)
    ridgeCV_pred = ridgeCV.predict(X_test)
    mae = mean_absolute_error(y_test, ridgeCV_pred)
    mse = mean_squared_error(y_test, ridgeCV_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, ridgeCV_pred)
    print(mae, mse, rmse, r2)
```

```
{'alpha': 10}
0.2475795472019324 0.23039724428236039 0.4799971294522087 0.6150629104071571
```

최적의 파라미터를 찾기 위해 GridSearchCV 를 사용하였다. 위에 경우에는 10 에서 가장 성능이 좋았다. 이보다 더 큰 값을 넣어보았다.

```
parameters = {'alpha': [10, 15, 20, 25, 30, 50, 100, 1000]}
for alpha in parameters:
    ridge = Ridge(alpha=alpha)
    ridgeCV = GridSearchCV(ridge, parameters,
                           scoring='neg_mean_squared_error', cv=5)
    ridgeCV.fit(X_train, y_train)
    print(ridgeCV.best_params_)
    ridgeCV_pred = ridgeCV.predict(X_test)
    mae = mean_absolute_error(y_test, ridgeCV_pred)
    mse = mean_squared_error(y_test, ridgeCV_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, ridgeCV_pred)
    print(mae, mse, rmse, r2)
```

```
{'alpha': 15}
0.2476461714510891 0.23046659647707665 0.48006936631811514 0.6149470399588186
```

alpha 를 위와 같이 수정한 결과 15 에서 가장 좋은 결과가 나왔다. 이보다 더 큰 값을 넣어보았다.

```
parameters = {'alpha': [15, 16, 17, 20, 25, 50]}
for alpha in parameters:
    ridge = Ridge(alpha=alpha)
    ridgeCV = GridSearchCV(ridge, parameters,
                           scoring='neg_mean_squared_error', cv=5)
    ridgeCV.fit(X_train, y_train)
    print(ridgeCV.best_params_)
    ridgeCV_pred = ridgeCV.predict(X_test)
    mae = mean_absolute_error(y_test, ridgeCV_pred)
    mse = mean_squared_error(y_test, ridgeCV_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, ridgeCV_pred)
    print(mae, mse, rmse, r2)
```

```
{'alpha': 15}
0.2476461714510891 0.23046659647707665 0.48006936631811514 0.6149470399588186
```

이 과정을 통해 15 일 때가 최적의 파라미터로 구해졌다. alpha 를 15 로 설정하고 릿지 회귀를 진행한 결과, R^2 값이 0.6149 로 1 에 가까운 값이 나왔지만 이는 다중 선형 회귀보다 낮은 성능이었다.

〈 라쏘 (Lasso) 회귀 〉

```
parameters = {'alpha': [1e-3, 1e-2, 1, 5, 10, 20]}
lasso = Lasso()
lassoCV = GridSearchCV(lasso, parameters,
                       scoring='neg_mean_squared_error', cv=5)
lassoCV.fit(X_train, y_train)
print(lassoCV.best_params_)
lassoCV_pred = lassoCV.predict(X_test)
mae = mean_absolute_error(y_test, lassoCV_pred)
mse = mean_squared_error(y_test, lassoCV_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, lassoCV_pred)
print(mae, mse, rmse, r2)
```

```
{'alpha': 0.001}
0.24761226047574683 0.23040481457062395 0.48000501515153354 0.6150502623186758
```

Ridge 와 같이 최적의 파라미터를 찾기 위해 GridSearchCV 를 사용하였다. 위의 경우에서 0.001 에서 가장 성능이 좋았다.

```
parameters = {'alpha': [0.001, 0.0012, 0.0013, 0.0014, 0.0015]}
lasso = Lasso()
lassoCV = GridSearchCV(lasso, parameters,
                       scoring='neg_mean_squared_error', cv=5)
lassoCV.fit(X_train, y_train)
print(lassoCV.best_params_)
lassoCV_pred = lassoCV.predict(X_test)
mae = mean_absolute_error(y_test, lassoCV_pred)
mse = mean_squared_error(y_test, lassoCV_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, lassoCV_pred)
print(mae, mse, rmse, r2)
```

```
{'alpha': 0.001}
0.24761226047574683 0.23040481457062395 0.48000501515153354 0.6150502623186758
```

이 과정을 통해 0.001 일 때가 최적의 파라미터로 구해졌다. alpha 를 0.001 로 설정하고 라쏘 회귀를 진행한 결과, R^2 값이 0.6150 으로 1 에 가까운 값이 나왔지만 이는 릿지

회귀보다는 아주 조금 좋은 성능이었지만 거의 차이가 없었고, 다중 선형 회귀보다 낮은 성능이었다.

〈 엘라스틱넷 (Elastic Net) 회귀 〉

```
df1 = df
df.head()
```

	age	sex	bmi	children	region	charges
1	18	1	33.770	1	4	1725.55230
2	28	1	33.000	3	4	4449.46200
3	33	1	22.705	0	1	21984.47061
4	32	1	28.880	0	1	3866.85520
5	31	0	25.740	0	4	3756.62160

```
y = y_log
X = df1.drop(['charges','region'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

라쏘(L1 정규화) 회귀과 릿지(L2 정규화) 회귀는 여전히 모델의 복잡성이 높기 때문에, 이 L1, L2 정규화를 동시에 사용하여 두 회귀의 장점을 가지고 L1 규제에 따라 회귀 계수가 급격하게 변하는 것을 완화하는 엘라스틱넷 회귀를 진행하기 전에, 위에서 했던 회귀 모델 중 성능이 가장 좋았던 입력이 age, sex, bmi, children 인 다중 선형 회귀와 비교를 위해 입력(X)은 age, sex, bmi, children 로 설정해주었고, 출력(y)은 위에서 charges 데이터를 로그 변환을 통해 정규화한 값으로 설정해주었다.

```
alphas = [0.07, 0.1, 0.5, 1, 3]
for alpha in alphas:
    enet_model = ElasticNet(alpha=alpha, l1_ratio=0.7).fit(X_train,y_train)
    enetYpred_ = enet_model.predict(X_test)
    mae = mean_absolute_error(y_test, enetYpred_)
    mse = mean_squared_error(y_test, enetYpred_)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, enetYpred_)
    print(alpha, max(mae, mse, rmse, r2))
```

```
0.07 0.6313948118884687
0.1 0.6279983407149958
0.5 0.5914698078291203
1 0.5896494869628139
3 0.5600350205333626
```

```
alphas = [0.07, 0.08, 0.09, 0.1, 0.12]
for alpha in alphas:
    enet_model = ElasticNet(alpha=alpha, l1_ratio=0.7).fit(X_train,y_train)
    enetYpred_ = enet_model.predict(X_test)
    mae = mean_absolute_error(y_test, enetYpred_)
    mse = mean_squared_error(y_test, enetYpred_)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, enetYpred_)
    print(alpha, max(mae, mse, rmse, r2))
```

```
0.07 0.6313948118884687
0.08 0.6303536106368153
0.09 0.6292211319408854
0.1 0.6279983407149958
0.12 0.6252856259533022
```

적절한 alpha 를 찾는 과정을 수행하였다. 각 alpha 값마다 오차가 매우 크다면 MAE, MSE, RMSE 값이 1 을 넘을 것이고 이는 R^2 값보다 클 것이기 때문에 오차가 크다면 MAE, MSE, RMSE 출력되고 크지 않다면 R^2 값이 출력될 수 있도록 회귀 성능 평가 지표인 MAE, MSE, RMSE, RMSE, R^2 중 가장 큰 값을 출력하도록 했다. 위 경우에서 alpha 가 0.07 일 때 가장 성능이 좋았다.

```
enet_model = ElasticNet(alpha=0.07, l1_ratio=0.7).fit(X_train,y_train)
enetYpred_ = enet_model.predict(X_test)
mae = mean_absolute_error(y_test, enetYpred_)
mse = mean_squared_error(y_test, enetYpred_)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, enetYpred_)
print(alpha, mae, mse, rmse, r2)
```

```
3 0.23772886631807374 0.20232683137374277 0.4498075492627294 0.6313948118884687
```

alpha 를 0.07 로 설정하고 엘라스틱넷 회귀를 진행한 결과, R^2 값이 0.6314 으로 1 에 가까운 값이 나왔다. 이는 릿지, 라쏘 회귀보다는 아주 조금 좋은 성능이었지만 다중 선형 회귀보다 낮은 성능이었다.

• LinearRegression 다중 선형 회귀, 릿지 회귀, 라쏘 회귀, 엘라스틱넷 회귀 결과 및 분석
- 회귀 성능 평가 지표: MAE, MSE, RMSE, RMSE, R^2

	입력(feature)	회귀 성능 평가 지표	회귀 성능
Linear Regression 다중 선형 회귀	age, sex, bmi, children	R^2	약 0.7734
릿지 회귀	age, sex, bmi, children	R^2	약 0.6149
라쏘 회귀	age, sex, bmi, children	R^2	약 0.6150
엘라스틱넷 회귀	age, sex, bmi, children	R^2	약 0.6314

회귀 성능: Linear Regression 다중 선형 회귀 > 엘라스틱넷 회귀 > 라쏘 회귀 > 릿지 회귀

Ridge 회귀, Lasso 회귀, Elastic Net 회귀는 alpha 값에 따라 성능 차이가 많이 나기 때문에 적절한 alpha 값 설정이 중요하다는 것을 알 수 있었다. 이를 위해 릿지 회귀와 라쏘 회귀에서는 GridSearchCV 를 사용해 적절한 alpha 값을 설정해주었고, 엘라스틱넷 회귀에서는 실험적으로 적절한 alpha 값을 찾고 설정해주었다. 엘라스틱넷 회귀는 릿지 회귀와 라쏘 회귀의 장점을 가진 회귀인 만큼 두 회귀보다 좋은 회귀 성능을 보여줬다. 하지만 Linear Regression 알고리즘을 이용한 다중 선형 회귀보다 낮은 성능을 보였다.

〈 로지스틱 (Logistic) 회귀 〉

```
df2 = df
df2.describe()
```

	age	sex	bmi	children	region	charges
count	1064.000000	1064.000000	1064.000000	1064.000000	1064.000000	1064.000000
mean	39.385338	0.485902	30.651795	1.090226	2.030075	8434.268298
std	14.083410	0.500036	6.043111	1.218136	1.579811	5993.781819
min	18.000000	0.000000	15.960000	0.000000	0.000000	1121.873900
25%	26.750000	0.000000	26.315000	0.000000	1.000000	3986.438700
50%	40.000000	0.000000	30.352500	1.000000	3.000000	7345.405300
75%	52.000000	1.000000	34.430000	2.000000	4.000000	11362.887050
max	64.000000	1.000000	53.130000	5.000000	4.000000	36910.608030

로지스틱 회귀의 목적은 일반적인 회귀 분석의 목표와 동일하게 종속 변수와 독립 변수간의 관계를 구체적인 함수로 나타내어 향후 예측 모델에 사용하는 것이다. 이는 독립 변수의 선형 결합으로 종속 변수를 설명한다는 관점에서는 선형 회귀 분석과 유사하다. 하지만 로지스틱 회귀는 선형 회귀 분석과는 다르게 종속 변수가 범주형 데이터를 대상으로 하며 입력 데이터가 주어졌을 때 해당 데이터의 결과가 특정 분류로 나뉘기 때문에 일종의 분류 기법으로도 볼 수 있다.

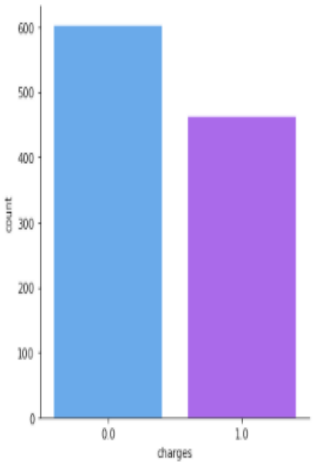
```
df2.loc[df1['charges'] <= 8434, 'charges'] = 0 #on or under avarage
df2.loc[df1['charges'] > 8434, 'charges'] = 1 # above avarage
df2.describe()
```

	age	sex	bmi	children	region	charges
count	1064.000000	1064.000000	1064.000000	1064.000000	1064.000000	1064.000000
mean	39.385338	0.485902	30.651795	1.090226	2.030075	0.434211
std	14.083410	0.500036	6.043111	1.218136	1.579811	0.495886
min	18.000000	0.000000	15.960000	0.000000	0.000000	0.000000
25%	26.750000	0.000000	26.315000	0.000000	1.000000	0.000000
50%	40.000000	0.000000	30.352500	1.000000	3.000000	0.000000
75%	52.000000	1.000000	34.430000	2.000000	4.000000	1.000000
max	64.000000	1.000000	53.130000	5.000000	4.000000	1.000000

로지스틱 회귀는 일종의 분류기법이기 때문에 타겟 값인 charges 데이터를 charges 데이터의 평균값인 8434 를 기준으로, 평균보다 낮으면 0, 높으면 1 로 데이터를 변환해주었다. 최소 0, 최대 1 로 정규화하는 MinMaxScaler 를 사용할 수도 있었지만 간단하기 때문에 직접적으로 데이터를 변환해주었다.

```
sns.catplot(x="charges", kind="count", palette="cool", data=df2)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdca1ed9ac8>
```




```
df2.groupby(['charges']).count()
```

charges	age	sex	bmi	children	region
0.0	602	602	602	602	602
1.0	462	462	462	462	462

변환된 데이터를 확인하였다. charges 데이터의 평균값을 기준으로 데이터를 변환했기 때문에 거의 데이터의 절반씩으로 나눠진 것을 확인하였다.

```
feature_cols = ['age', 'sex', 'bmi', 'children']
X = df1[feature_cols]
y = df1['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

lg = LogisticRegression()
lg.fit(X_train, y_train)
lg.score(X_test, y_test)
```

```
0.9107981220657277
```

Logistic Regression 을 사용하여 데이터를 학습시키고, test 데이터를 인자로 받아 학습이 완료된 모델의 평가 점수를 출력했다. 약 0.9108 로 높은 점수였지만 로지스틱 회귀는 종속 변수가 범주형 데이터를 대상으로 하며 입력 데이터가 주어졌을 때 해당 데이터의 결과가 특정 분류(charges 평균값 기준)로 나뉘는 일종의 분류 기법이기 때문에 앞서 모델링한 회귀 모델들과 비교는 어려웠다,

〈 랜덤 포레스트 (Random Forest) Regressor 〉

```
df1.head()
```

	age	sex	bmi	children	region	charges
1	18	1	33.770	1	4	1725.55230
2	28	1	33.000	3	4	4449.46200
3	33	1	22.705	0	1	21984.47061
4	32	1	28.880	0	1	3866.85520
5	31	0	25.740	0	4	3756.62160

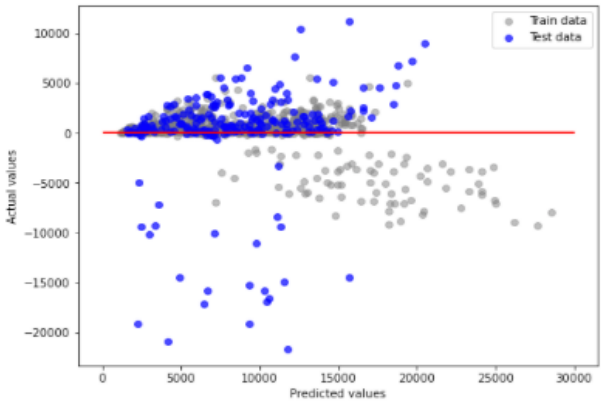
```
X = df1.drop(['charges', 'region'], axis=1)
y = df1.charges
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

Rfr = RandomForestRegressor(n_estimators = 100, criterion = 'mse',
                            random_state = 1,
                            n_jobs = -1)
Rfr.fit(X_train,y_train)
X_train_pred = Rfr.predict(X_train)
X_test_pred = Rfr.predict(X_test)

mae = mean_absolute_error(y_test, X_test_pred)
mse = mean_squared_error(y_test, X_test_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, X_test_pred)
print(mae, mse, rmse, r2)
```

```
2794.7970130810922 25477057.59966076 5047.480321869592 0.2464581330818677
```

Train 데이터와 test 데이터를 각각 80%, 20%로 나누고 Random Forest Regressor 클래스를 사용하여 학습을 진행하였다. 위에서 했던 회귀 모델 중 성능이 가장 좋았던 입력이 age, sex, bmi, children 인 다중 선형 회귀와 비교를 위해 입력을 동일하게 설정해주었다. 회귀 성능을 확인하기 위해 MAE, MSE, RMSE, RMSE, R^2 값을 확인하였다. MAE, MSE, RMSE, RMSE 값을 통해 오차가 매우 크다는 것을 알 수 있었고, R^2 값이 약 0.2465 로 0 에 가까운 값으로 성능이 좋지 않았다.

```
plt.figure(figsize=(8,6))

plt.scatter(X_train_pred, X_train_pred - y_train,
            c = 'gray', marker = 'o', s = 35, alpha = 0.5,
            label = 'Train data')
plt.scatter(X_test_pred, X_test_pred - y_test,
            c = 'blue', marker = 'o', s = 35, alpha = 0.7,
            label = 'Test data')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.legend(loc = 'upper right')
plt.hlines(y = 0, xmin = 0, xmax = 30000, lw = 2, color = 'red')
```

```
<matplotlib.collections.LineCollection at 0x7fefb24a9da0>
```



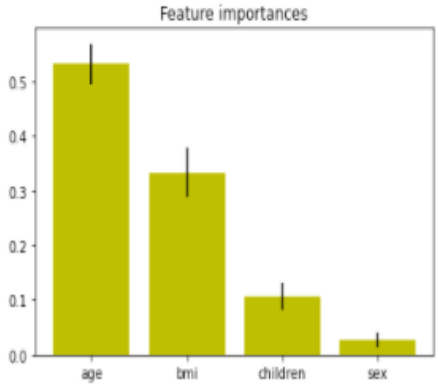
Train 데이터와 Test 데이터의 예상 값과 실제 값 사이의 오차를 시각화 하였다. Train 데이터보다 Test 데이터가 주로 예상 값과 실제 값 사이의 오차가 적은 편이었지만 오차가 큰 데이터들이 존재했다.

```
print('Feature importance ranking')
importances = Rfr.feature_importances_
std = np.std([tree.feature_importances_ for tree in Rfr.estimators_],axis=0)
indices = np.argsort(importances)[::-1]
variables = ['age', 'sex', 'bmi', 'children']
importance_list = []
for f in range(X.shape[1]):
    variable = variables[indices[f]]
    importance_list.append(variable)
    print("%d.%s(%f)" % (f + 1, variable, importances[indices[f]]))
```

```
Feature importance ranking
1.age(0.532397)
2.bmi(0.333440)
3.children(0.107364)
4.sex(0.026798)
```

```
# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(importance_list, importances[indices],
       color="y", yerr=std[indices], align="center")
```

```
<BarContainer object of 4 artists>
```



추가적으로, 랜덤 포레스트의 피처 중요도를 확인해보았다.
1. age(0.532397) 2. bmi(0.333440) 3. children(0.107364) 4. sex(0.026798) 순으로 피처에 큰 영향을 끼친 것으로 보인다. 따라서 낮은 중요도의 피처(ex: children, sex)를 제거하고 다시 학습한다면 더 좋은 성능 값을 얻을 수 있을 것으로 예상된다.

〈 회귀 트리 〉

• StandardScaler

```
df1.head()
```

	age	sex	bmi	children	region	charges
1	18	1	33.770	1	4	1725.55230
2	28	1	33.000	3	4	4449.46200
3	33	1	22.705	0	1	21984.47061
4	32	1	28.880	0	1	3866.85520
5	31	0	25.740	0	4	3756.62160

```
y = y_log
X = df1.drop(['charges','region'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

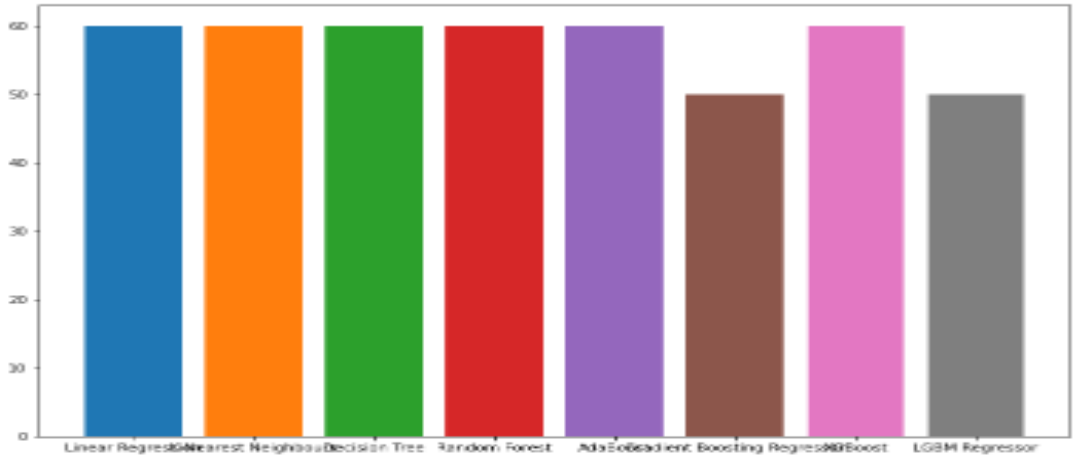
학습을 진행하기 전에 위에서 했던 회귀 모델 중 성능이 가장 좋았던 입력이 age, sex, bmi, children 인 다중 선형 회귀와 비교를 위해 입력을 동일하게 설정해주었다.. Train 데이터와 test 데이터를 각각 80%, 20%로 나누었다. 선형 회귀 모델을 위해 피처 값과 타겟 값의 분포가 정규 분포 형태를 선호하기 때문에 출력은 charges 데이터를 로그 변환하여 정규화한 값을 사용하였고, 입력은 StandardScaler 를 이용해 평균 0, 분산 1 의 표준 정규 분포로 데이터를 변환하였다.

```
lr = LinearRegression()

knn = KNeighborsRegressor(n_neighbors=10)

dt = DecisionTreeRegressor(max_depth = 3)

rf = RandomForestRegressor(max_depth = 3, n_estimators=500)

ada = AdaBoostRegressor( n_estimators=50, learning_rate =.01)

gbr = GradientBoostingRegressor(max_depth=2, n_estimators=100, learning_rate =.2)

xgb = XGBRegressor(max_depth = 3, n_estimators=50, learning_rate =.2)

lgb = LGBMRegressor(max_depth = 3, n_estimators=500)

regressors = [('Linear Regression', lr), ('K Nearest Neighbours', knn),
              ('Decision Tree', dt), ('Random Forest', rf), ('AdaBoost', ada),
             ('Gradient Boosting Regressor', gbr), ('XGBoost', xgb),('LGBM Regressor', lgb)]
```

LinearRegression, KNeighborsRegressor, DecisionTreeRegressor, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, XGBRegressor, LGBMRegressor 를 이용하여 각각 학습을 진행하였다.

```
for regressor_name, regressor in regressors:

    # Fit regressor to the training set
    regressor.fit(X_train, y_train)

    # Predict
    y_pred = regressor.predict(X_test)
    accuracy = round(r2_score(y_test,y_pred),1)*100

    # Evaluate  accuracy on the test set
    print('{:s} : {:.0f} %'.format(regressor_name, accuracy))
    plt.rcParams["figure.figsize"] = (12,8)
    plt.bar(regressor_name,accuracy)
```

```
Linear Regression : 60 %
K Nearest Neighbours : 60 %
Decision Tree : 60 %
Random Forest : 60 %
AdaBoost : 60 %
Gradient Boosting Regressor : 50 %
[16:51:24] WARNING: /workspace/src/objective/regression_obj.cu:152:
XGBoost : 60 %
LGBM Regressor : 50 %
```



위에서 각각 학습한 회귀 모델의 성능을 확인하기 위해 R^2 값을 확인하였다. R^2 값의 100 을 곱해줘서 퍼센트로 출력하였고 이를 바 그래프로 시각화 하였다. 대체로 비슷한 성능이었지만 GradientBoostingRegressor, LGBMRegressor 에서 낮은 성능을 보였다.

• MinMaxScaler

```
y = y_log
X = df1.drop(['charges','region'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

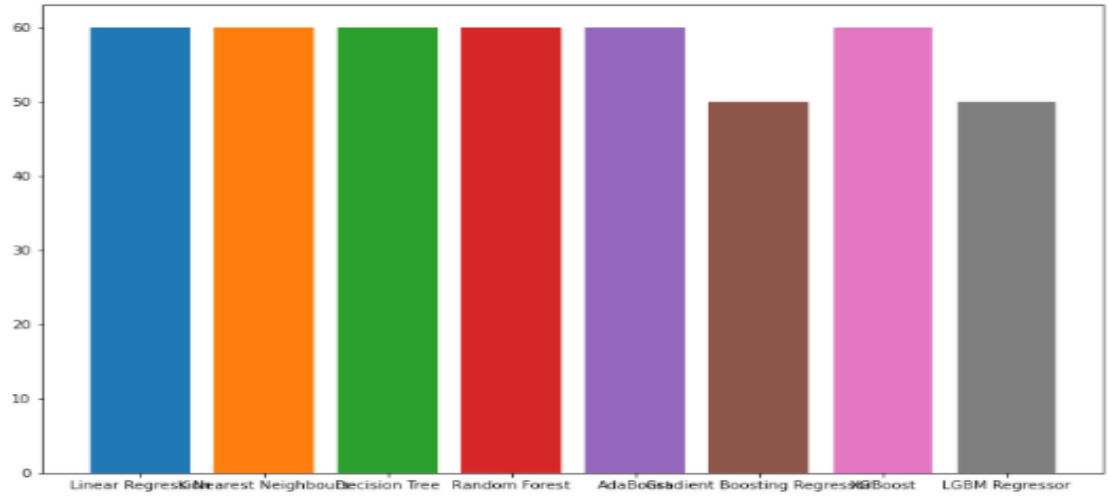
입력을 MinMaxScaler 를 이용해 최소 0, 최대 1 로 정규화하여 데이터를 변환해주었다.

```
lr = LinearRegression()

knn = KNeighborsRegressor(n_neighbors=10)

dt = DecisionTreeRegressor(max_depth = 3)

rf = RandomForestRegressor(max_depth = 3, n_estimators=500)

ada = AdaBoostRegressor( n_estimators=50, learning_rate =.01)

gbr = GradientBoostingRegressor(max_depth=2, n_estimators=100, learning_rate =.2)

xgb = XGBRegressor(max_depth = 3, n_estimators=50, learning_rate =.2)

lgb = LGBMRegressor(max_depth = 3, n_estimators=500)

regressors = [('Linear Regression', lr), ('K Nearest Neighbours', knn),
              ('Decision Tree', dt), ('Random Forest', rf), ('AdaBoost', ada),
             ('Gradient Boosting Regressor', gbr), ('XGBoost', xgb),('LGBM Regressor', lgb)]
```

위와 마찬가지로, LinearRegression, KNeighborsRegressor, DecisionTreeRegressor, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, XGBRegressor, LGBMRegressor 를 이용하여 각각 학습을 진행하였다.

```
for regressor_name, regressor in regressors:

    # Fit regressor to the training set
    regressor.fit(X_train, y_train)

    # Predict
    y_pred = regressor.predict(X_test)
    accuracy = round(r2_score(y_test,y_pred),1)*100

    # Evaluate  accuracy on the test set
    print('{:s} : {:.0f} %'.format(regressor_name, accuracy))
    plt.rcParams["figure.figsize"] = (12,8)
    plt.bar(regressor_name,accuracy)
```

```
Linear Regression : 60 %
K Nearest Neighbours : 60 %
Decision Tree : 60 %
Random Forest : 60 %
AdaBoost : 60 %
Gradient Boosting Regressor : 50 %
[16:55:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is
XGBoost : 60 %
LGBM Regressor : 50 %
```



위에서 각각 학습한 회귀 모델의 성능을 확인하기 위해 R^2 값을 확인하였다. 대체로 비슷한 성능이었지만 GradientBoostingRegressor, LGBMRegressor 에서 낮은 성능을 보였다.

• 결과 분석

	입력	출력	회귀 성능 평가 지표	회귀 성능
Linear Regression 단순 선형 회귀	age	Charges 로그변환	R^2	약 0.6423
Linear Regression 단순 선형 회귀	bmi	Charges 로그변환	R^2	약 -0.149
Linear Regression 다중 선형 회귀	age, sex, bmi, children, region	Charges 로그변환	R^2	약 0.6930
Linear Regression 다중 선형 회귀	age, sex, bmi, children	Charges 로그변환	R^2	약 0.7734
릿지 회귀	age, sex, bmi, children	Charges 로그변환	R^2	약 0.6149
라쏘 회귀	age, sex, bmi, children	Charges 로그변환	R^2	약 0.6150
엘라스틱넷 회귀	age, sex, bmi, children	Charges 로그변환	R^2	약 0.6314
랜덤 포레스트	age, sex, bmi, children	Charges	R^2	약 0.2465
Linear Regression	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 60 %
K Neighbors Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 60 %
Decision Tree Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 60 %

Random Forest Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 60 %
AdaBoost Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 60 %
Gradient Boosting Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 50 %
XGB Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 60 %
LGBM Regressor	age, sex, bmi, children StandardScaler MinMaxScaler	Charges 로그변환	R^2	약 50 %

회귀 평가 지표 R^2를 기준으로 비교해봤을 때, 입력이 age, sex, bmi, children 이고, Charges 데이터를 로그 변환하여 정규화하여 출력으로 설정하고 Linear Regression 알고리즘을 사용한 다중 선형 회귀의 R^2 값이 약 0. 7734 로 가장 성능이 좋았다.

4) 고찰

이번 텀프로젝트를 통해 여러가지 회귀 모델을 모델링 해보면서 그 방법과 그 특성에 대해 공부할 수 있었다.

또한 원하는 목적에 맞게 dataset 을 선별하는 방법과 찾은 dataset 을 전처리하는 방법에 대해서도 처음부터 끝까지 직접 해보며 습득하였다. 이 과정 속에서 같은 회귀 알고리즘을 사용한다고 하더라도 데이터의 상관 관계를 분석하고 이를 이용해 데이터의 중요도를 파악하여 데이터 전처리를 어떻게 해주는지에 따라 회귀 성능의 큰 차이가 발생한다는 것을 알 수 있었고 그렇기 때문에 목적에 맞게 데이터를 전처리 해주는 것이 무엇보다 중요한 과정이라는 것을 알 수 있었다.

자신이 사용하려는 알고리즘과 dataset 에 따라 여러가지 시도와 실패를 통해 각 파라미터의 특징과 튜닝 방법을 익혔다.

이번 기계학습 수업을 통해 pandas, numpy 와 같은 라이브러리를 능숙하게 다루는 능력을 길렀으며 머신러닝에 관한 이론을 이해하고 실습을 통해 실력을 향상시킬 수 있었던 시간이었다.