MACHINE LEARNING

INSURANCE PREDICTION

의료 보험료 예측

PRESENTATION

인공지능 - 컴퓨터공학전공 - 2019108269 - 윤지영

TABLE OF CONTENTS

데이터 분석 및 예측 과정

1. Introduction (개요)

About subject

About Insurance prediction

의료 보험 예측

한 사람이 보험료를 얼마를 내야 할까?

- 의료 보험 데이터를 활용해 한 사람이 보험료를 얼마를 내야 할지 예측하는 회귀 문제를 다룸
- 전체적인 빅데이터 분석과정을 살펴보기 위해 코드 분석
- 체계적으로 정리가 되어있고, 다양한 방법을 사용하여 이번 주제 선정

인공지능 빅데이터 python Skit-learn

의료 보험이 무엇일까?

" 의료 보험은 의료 행위에 대한 비용을 지불하는 보험 "

많은 비용이 드는 의료 서비스에 대한 부담을 줄이고 비용 대비 효과적인 의료 행위를 지향하기 위한 제도

우리나라 의료 보험

사회 보험 방식 - 국가가 전 국민을 대상으로 의료 보험 강제 가입

보험료 = 보험사 + 본인 + 정부

Why important?

보험 업계 & 인공지능

보험 업계도 역시 인터넷 기반으로 연결
고객 데이터 축척
빅데이터를 적절이 활용

“ 소비자 만족도 개선을 위한 신상품 개발 ”

“ 새로운 가치 창출 시도 ”

“ 적정한 보험료 예측 “

보험료 예측 이외에도 고객의 빅데이터를 기반으로 보험 수요의 예측,
위험률, 이탈 고객 관리, 보험 사기 방지 등 다양한 분야에 활용

Why important?

보험 업계 & 인공지능

먼저 교보생명의 디플래닉스(Dplanex)는 지난 24일 카이스트(KAIST)와 'KYOBO-DPLANEX-KAIST (KDK) 미래보험 AI연구센터'를 개소했다.

보험 업계도 역시 인터넷 기반으로 연결

◆ AI 연구센터 설립하고 디지털 플랫폼 구축 나선 보험사들

29일 보험업계에 따르면 교보생명과 신한라이프, NH농협생명 등 주요 보험사들이 앞다퉈 디지털 고도화에 나서고 있다.

" 적정한 보험료 예측 "

KB손해보험은 지난해 모바일 앱을 통해 디지털 건강관리 서비스를 제공하는 자회사 KB헬스케어를 출범시켰다. 최근에는 향후 마이데이터 서비스를 연계, 건강검진 정보를 기반으로 하는 '개인 맞춤형 건강관리 프로그램' 운영을 준비하고 있다. 이어 이달부터는 메타버스 기반의 디지털 연수원인 '인재니움 메타'를 개소하기도 했다.

2. Data explanation

변수 살펴보기

Data(columns)

- Age: 피보험자의 나이
- Sex: 피보험자의 성별
- BMI: 피보험자의 체질량 지수 - $\frac{Weight\ (kg)}{(Height\ (m))^2}$
- Children: 피보험자의 자녀의 수
- Smoker: 흡연 여부 (yes / no)
- Region: 피보험자가 거주하는 지역 (Southeast / Southwest / Northeast / Northwest)
- Charges: 보험료

연속형 데이터

- Age
- BMI
- Children
- Charges

범주형 데이터

- Smoker
- Sex
- Region

3. Data library

데이터 라이브러리

Data library import

```
# library import
import pandas as pd
import numpy as np
from datetime import datetime
from sklearn.preprocessing import (OneHotEncoder, LabelEncoder, StandardScaler,
                                   MinMaxScaler, PowerTransformer, QuantileTransformer)
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.linear_model import LinearRegression, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
%matplotlib inline

warnings.filterwarnings(action='ignore')
```

Data library import

“ 왜 모든 라이브러리를 한번에 불러왔을까? ”

Ans) 전체 코드 가독성이 좋아진다. 하지만
python은 순차적으로 처리하는 언어이므로 그때 그때
선언하여도 문제는 없다.

Data library

코드 분석 - 데이터 라이브러리

01. Pandas

데이터 분석에 많이 사용되는 라이브러리이다.
일차원 데이터는 dictionary 처럼 사용 되고,
데이터 프레임을 통한 이차원 데이터 관리에
유용하다.

02. Numpy

대규모 다차원 배열부터 수치 계산을
효율적으로 처리할 수 있도록 지원하는
라이브러리

03. Datatime

날짜와 시간 데이터를 처리해주는 함수

04. Sklearn

많은 machine learning 알고리즘이 구현되어
있는 machine learning 라이브러리.

05. Matplot

다양한 데이터를 많은 방법으로 시각화 할 수
있게 만들어

06. Seaborn

Seaborn은 matplotlib를 기반으로 다양한
색상 테마와 통계용 차트 등의 기능을 추가한
시각화 패키지

Data library import

```
# data import & view head
df = pd.read_csv('../input/insurance/insurance.csv', engine='python')
df.head()
```

데이터를 담고 있는 .csv를 pandas 내부 모듈 read_csv를 이용하여 dataframe 형태로 읽어 오기

.head()를 이용하여 윗부분만 데이터를 가져오기

	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

4. EDA

탐색적 데이터 분석

What is EDA?

" 탐색적 데이터 분석이란 "

Goals of EDA

EDA의 궁극적인 목표

주어진 raw data에 대해서 이해하고 알아보는 과정

시각화 및 통계 도구를 이용해여 패턴을 발견하거나 특이성 파악

통계와 그래픽을 통해서 가설 검증

"

Exploratory Data Analysis

"

변수를 살펴보았으니 변수 간의 어떤 상관관계가 있는지, 분포는 어떻게 이루어지는지 살펴보아야 합니다. 이를 통해 특징을 파악하고 학습을 어떻게 잘 시킬 수 있을까 고민하며 필요하다면 변수를 변환하기도 합니다.

EDA - info() , describe()

```
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
```

Info를 통해서 결측 값은 없는지 자료형은 어떻게 되어있는지 살펴봅니다. 데이터 분석에서 자료 형태를 분류하는 것도 중요합니다.

✓ 모든 값이 채워져 있기 때문에 결측 값 없음

✓ 성별 흡연 여부 거주지역을 제외한 모든 변수들은 숫자형

✓ Object는 보통 문자열이 들어간 데이터로 연속형 데이터

EDA - info() , describe()

```
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

통계에서 살펴보는 평균, 표준편차, 최댓값, 최솟값 사분위수를 describe 함수를 통해서 한번에 살펴볼 수 있다. Count는 전체 사용한 데이터 개수, mean이 각 변수마다 평균, std가 표준 편차, min, max가 최댓값, 최소값을 나타낸다. 보통 이산형 데이터 즉, 숫자형 데이터를 기본으로 한다.

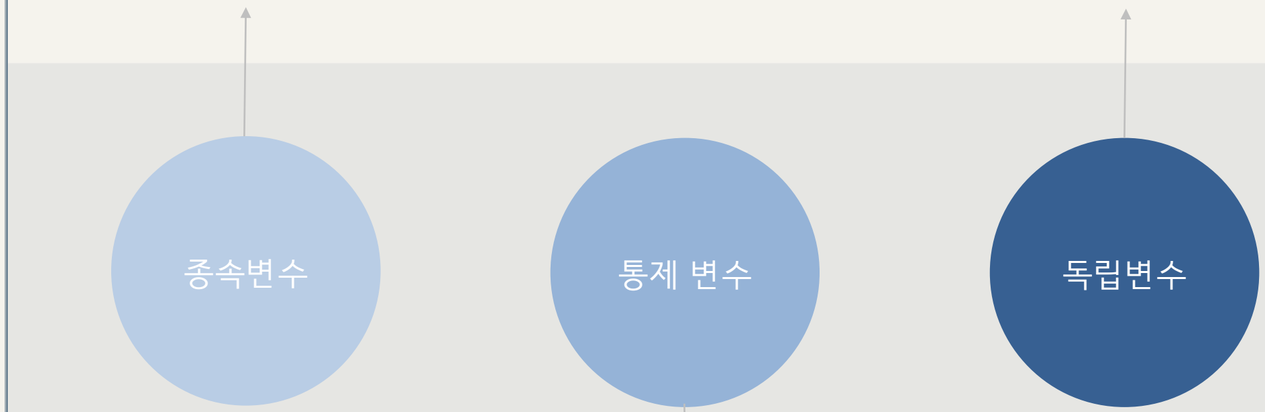
✓ 자녀의 수 표준편차가 평균과 비슷함

✓ feature의 분포가 넓게 퍼져 있음을 의미함.

Charges - 종속 변수

독립 변수에 영향을 받아서
변화하는 변수

종속변수에 영향을 미쳐서 종속
변수가 특저한 값을 갖게 하는
원인이 되는 변수



보려고 하는 변수는 아니지만
종속변수에도 영향을 미쳐서
통제해야 하는 변수

만약 charges도
독립 변수로 가정한다면, 보험료와 다른 특징들이
숫자 단위 차이가 너무 크다. 이렇게 단위 차이가
크면 특징의 영향을 많이 받을 가능성이 높다
→ **scaling 작업**

독립 변수와 종속변수는 인과 관계이다.
종속 변수인 charges가 다른
독립 변수들과 어떤 관계가 있는지
알아보아야 한다.

5. Visualization

시각화를 통한 관계 분석

EDA - visualization

2 X 3개의 그래프

크기는 가로 세로 30 20

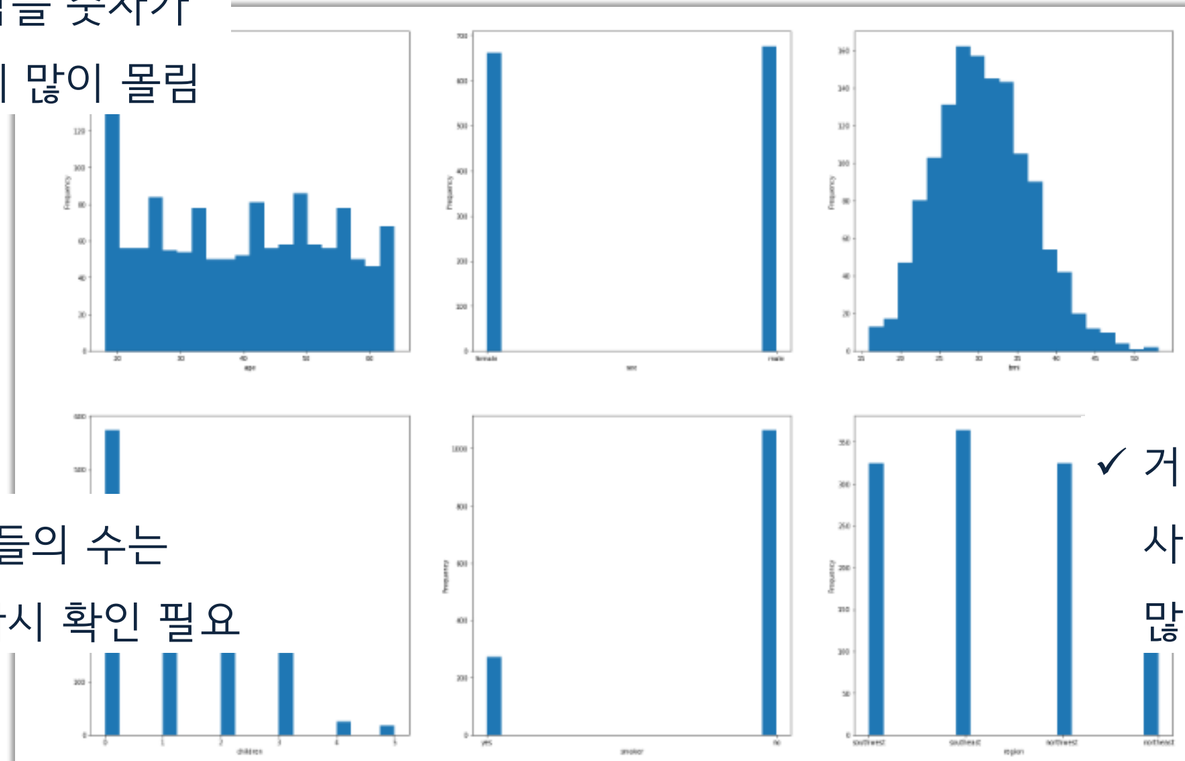
```
fig, ax = plt.subplots(2, 3, figsize=(30, 20))

idx = 0 # 보험료를 제외한 feature들을 지정할 인덱스입니다.
for i in range(2): # subplot들의 행(row)
    for j in range(3): # subplot들의 열(column)
        colname = list(df.columns)[idx]
        ax[i][j].hist(df[colname], bins=20)
        ax[i][j].set_xlabel(colname)
        ax[i][j].set_ylabel('Frequency')

        idx += 1
```

- 코드에 보이는 for문은 종속 변수인 charges(보험료)를 제외하고, 각 feature라고하는 변수들의 빈도(frequency)를 살펴보는 것이다.
- 'idx' 로 각 변수들을 index로 사용할 수 있게 지정해준다. 히스토그램에서 넓이를 bins로 지정하는데, 20으로 설정했다.
- Set_xlabel, Set_ylabel 로 라벨링을 해준다.

EDA - Each columns frequency

✓ BMI는 약간 왼쪽에 치우치고
무리에서 떨어진 값이 있어 보임.

✓ 나이가 20대보다 어린 사람들 숫자가
매우 많고, 자녀 숫자도 0에 많이 몰림



✓ 자녀가 4명 이상인 사람들의 수는
이상치에 가깝고 추후 다시 확인 필요

✓ 거주지는 southeast에 거주하는
사람들이 다른 지역에 비해
많지만 거의 일정.

✓ 남녀 비율은 비슷하고, 흡연자에 비해 비흡연자가
많음. 흡연 여부도 관련된 변수 일 것임을 예상

6. Correlation Analysis

상관관계 파악 및 구간 만들기

EDA - Correlation & Quantile

상관관계 파악 및 구간 만들기

```
df.corr()
```

	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000

Corr 함수를 이용하면 변수와 변수 간의 얼마나 관계가 있는지 0 과 1 사이의 수치로 나온다.

1에 가까울수록 양의 상관관계

-1에 가까울수록 음의 상관관계

→ 양의 상관관계는 두 변수 간의 비례관계

음의 상관 관계는 반비례관계

“ Scatter matrix ”

산점도 행렬은 데이터 특성 중 하나를 x축에 넣고 다른 특성 중 하나를 y 축에 놓아 두 특성상의 관계를 하나의 점으로 나타내는 그래프.

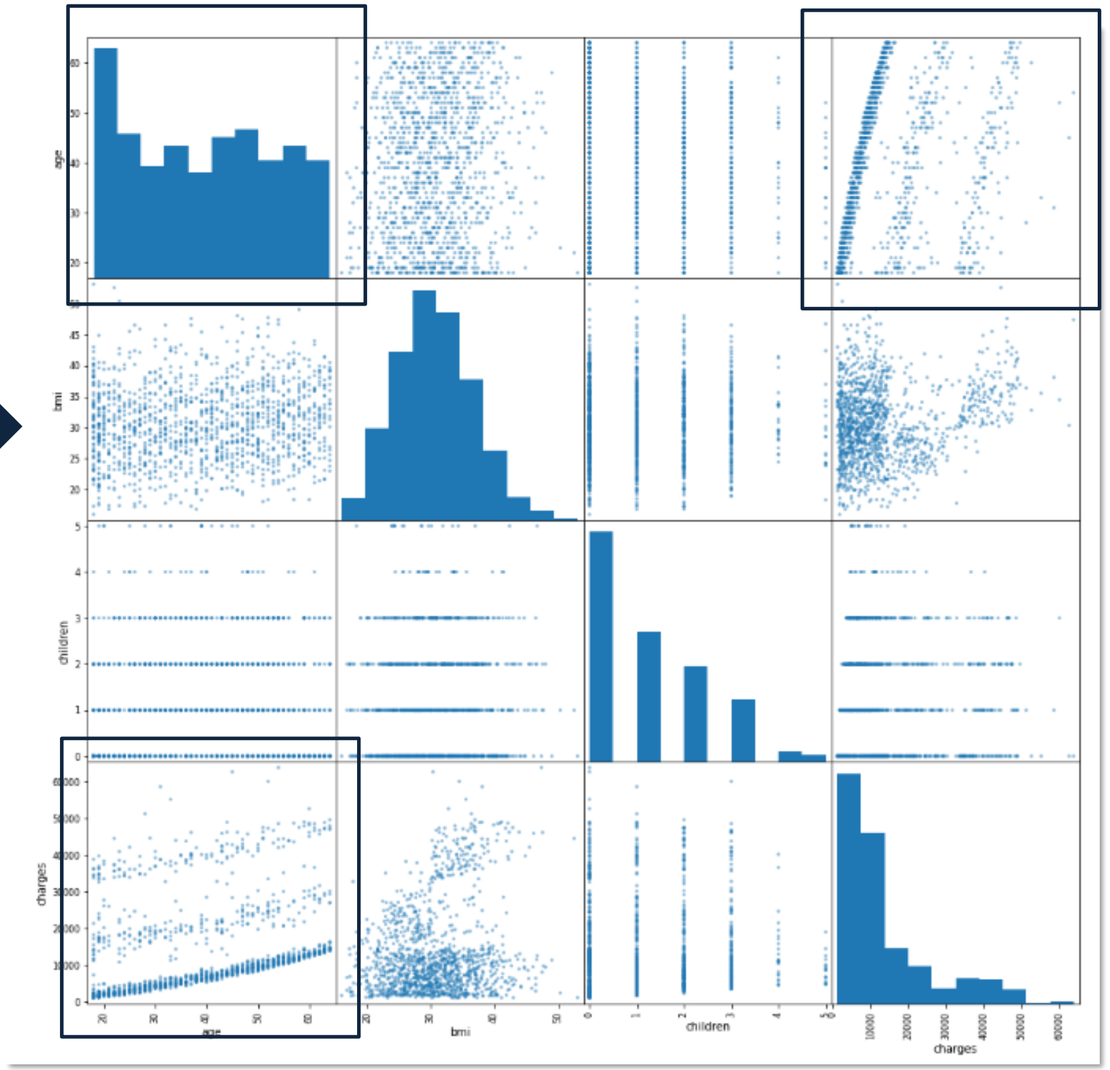
즉 두 feature간의 관계만 나타낼 수 있다.

EDA - Correlation & Quantile

상관관계 파악 및 구간 만들기

```
df.corr()
```

	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000



Corr 함수를 이용하면 변수와 변수 간의 얼마나 관계가 있는지 0 과 1 사이의 수치로 나온다.

1에 가까울수록 양의 상관관계

-1에 가까울수록 음의 상관관계

→ 양의 상관관계는 두 변수 간의 비례관계

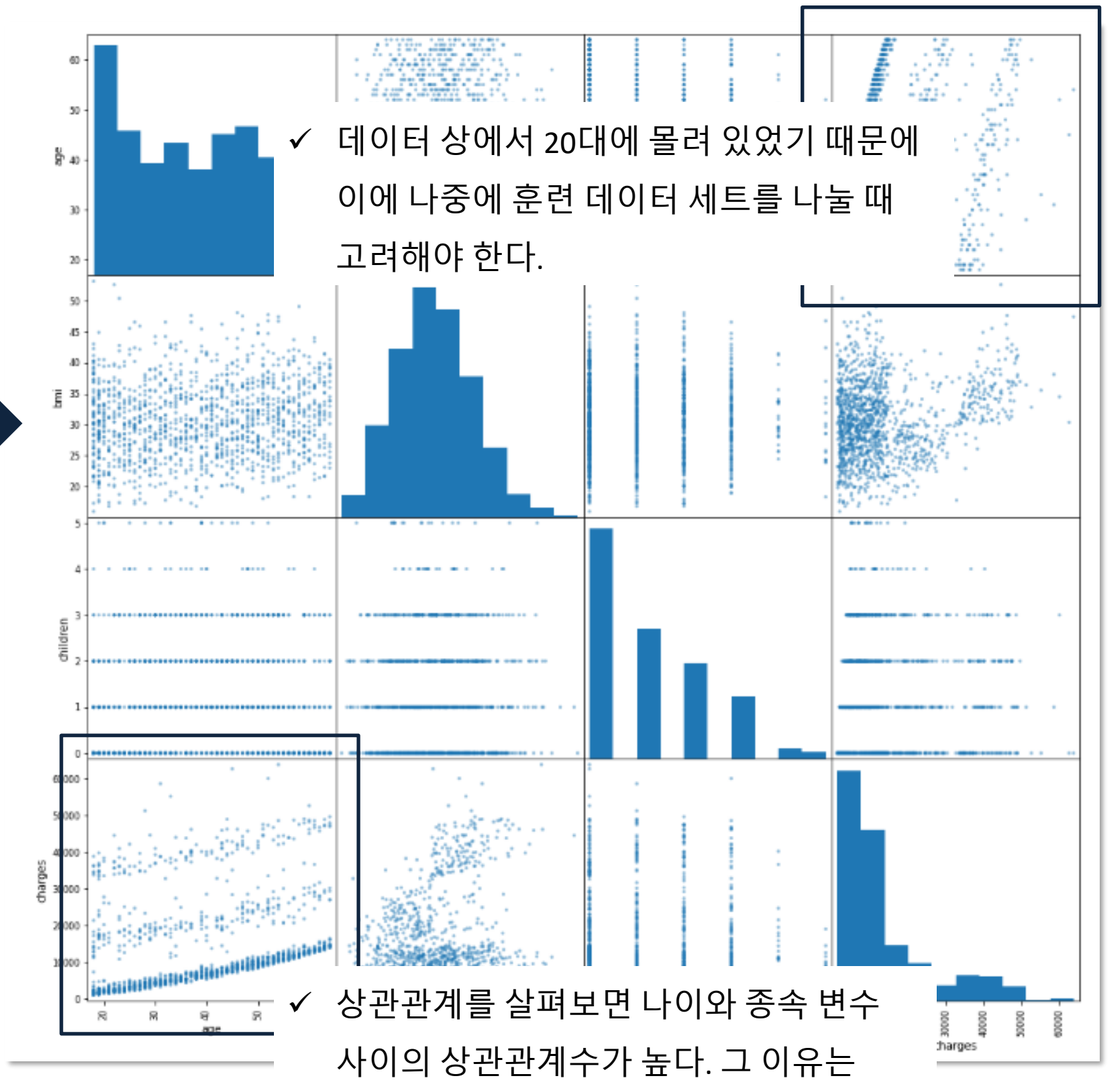
음의 상관 관계는 반비례관계

EDA - Correlation & Quantile

상관관계 파악 및 구간 만들기

```
df.corr()
```

	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000



✓ 데이터 상에서 20대에 몰려 있었기 때문에 이에 나중에 훈련 데이터 세트를 나눌 때 고려해야 한다.

✓ 상관관계를 살펴보면 나이와 종속 변수 사이의 상관관계수가 높다. 그 이유는 선형의 형태를 띄고 있기 때문이다

Corr 함수를 이용하면 변수와 변수 간의 얼마나 관계가 있는지 0 과 1 사이의 수치로 나온다.

1에 가까울수록 양의 상관관계

-1에 가까울수록 음의 상관관계

→ 양의 상관관계는 두 변수 간의 비례관계

음의 상관 관계는 반비례관계

EDA - Correlation & Quantile

상관관계 파악 및 구간 만들기

✓ 구간 별로 잘 표현이 되는데 구간에 라벨링을 하기 위해서 label 파라미터를 리스트 형식으로 넣어주면 됩니다.

```
bins = [0, 20, 25, 30, 35, 40, 45, 50, 55, 60, np.inf]
age_bin = pd.cut(df['age'], bins=bins, labels=[i+1 for i in ran
ge(len(bins)-1)])
df['age_bin'] = age_bin
df.head()
```

✓ pd.cut를 이용하여 같은 길이로 구간을 나눌 수 있다. 즉 dataframe age 부분을 bins 만큼 나누겠다는 뜻

나이를 구간으로 나타냄

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

7. Outlier Detect

이상치 탐지 및 제거

Outlier

“이상치 제거”

이상치 값은 수가 너무 크거나 너무 작아서 적은 수로도
전체적인 관계에 영향을 주는 값이다. 따라서 데이터 전 처리
과정에서 적절하게 처리해 주어야 한다.

EDA - Outlier Detection

이상치 추적 및 제거

```
# 이상치 탐지를 위해 boxplot을 그려보겠습니다.
plt.subplot(1,3,1)
sns.boxplot(data=df, y='age')
plt.subplot(1,3,2)
sns.boxplot(data=df, y='children')
plt.subplot(1,3,3)
sns.boxplot(data=df, y='bmi')
```


위에 히스토그램을 보면 5명의 자녀가 적다.

이 수를 이상치로 보고 제외할 지에 대한 문제

→ 이미 변수의 범위가 0-5사이에 정수로 주어졌기도 하고, 5명의 자녀를 가진 사람들이 다른 데이터와 완전히 떨어져 있다고 보기는 어렵기도 하다. 무엇보다 이 변수가 종속변수인 보험료에 영향이 적다는 점은 위에 상관관계를 통해서 확인 했다..

EDA - Outlier Detection

이상치 추적 및 제거

```
# 이상치 탐지를 위해 boxplot을 그려보겠습니다.
plt.subplot(1,3,1)
sns.boxplot(data=df, y='age')
plt.subplot(1,3,2)
sns.boxplot(data=df, y='children')
plt.subplot(1,3,3)
sns.boxplot(data=df, y='bmi')
```

위에 히스토그램을 보면 5명의 자녀가 적다.

이 수를 이상치로 보고 제외할 지에 대한 문제

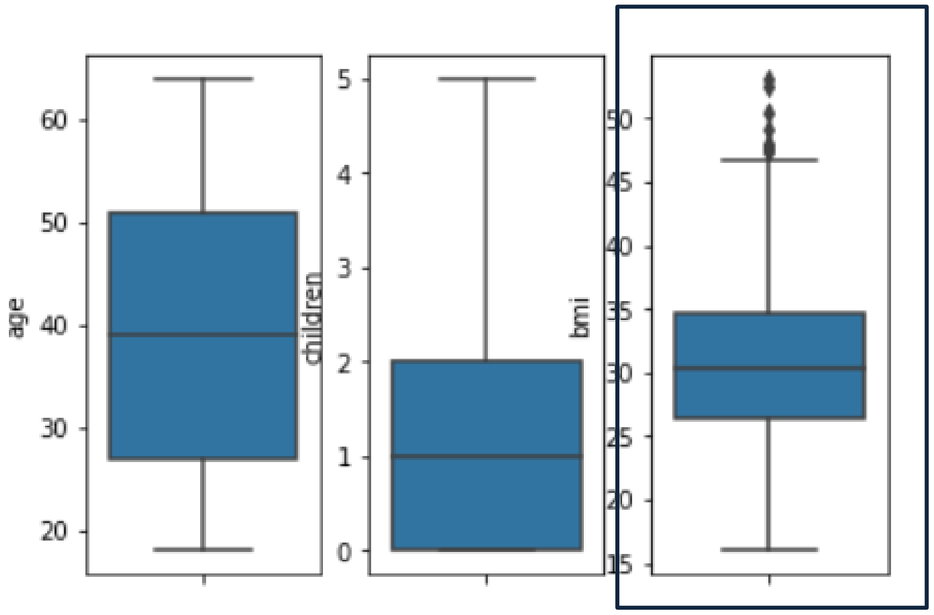
→ 이미 변수의 범위가 0-5사이에 정수로 주어졌기도 하고, 5명의 자녀를 가진 사람들이 다른 데이터와 완전히 떨어져 있다고 보기는 어렵기도 하다. 무엇보다 이 변수가 종속변수인 보험료에 영향이 적다는 점은 위에 상관관계를 통해서 확인 했다..

BMI 값이 위에 직선보다 벗어나 있는 것을 확인할 수 있다.

이 값들이 보통 이상치 값일 확률이 높다.

→ **IQR을 통해** 이상치의 존재를 파악한다.

```
<AxesSubplot:ylabel='bmi'>
```



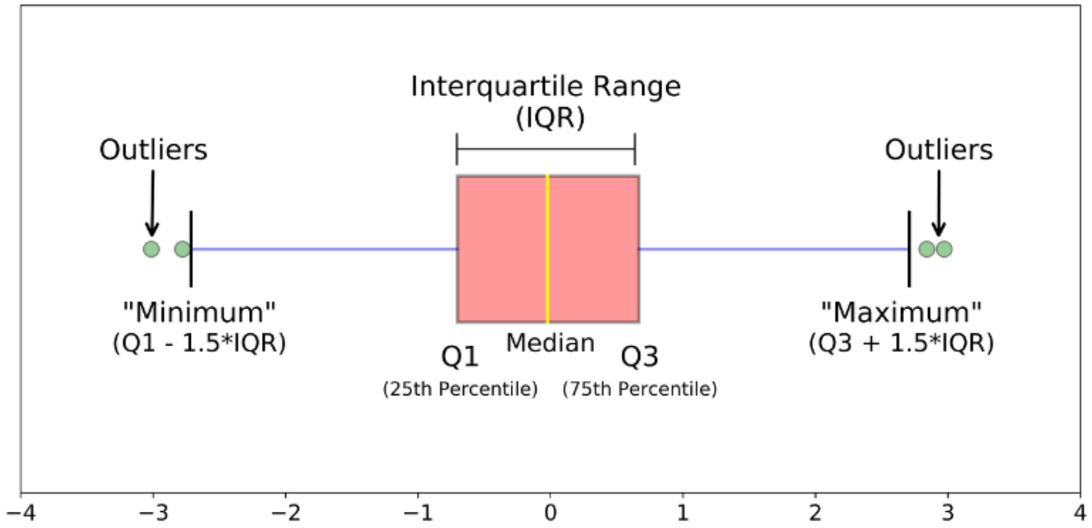
IQR

“inter quantile range 사분위개념”

전체 데이터 오름차순 정렬 하고 4등분(25%, 50%, 75%, 100%)

→ 75% 구간 값 - 25% 구간 값

- ✓ 노란 줄이 median 중앙값
- ✓ IQR = Q3 -Q1 이 값에 1.5를 곱해서 최소 제한서 최대 제한선을 두고 그 이상의 것을 이상치라고 하는 것입니다



https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

```
bmi_q1 = df['bmi'].quantile(q=0.25)
bmi_q3 = df['bmi'].quantile(q=0.75)
iqr = bmi_q3 - bmi_q1
```

✓ dataframe에 BMI를 가지고 와서 25% 구간에 값과 75% 구간에 값을 가져온다

```
condi1 = (df['bmi'] < (bmi_q1 - (1.5 * iqr)))
condi2 = (df['bmi'] > (bmi_q3 + (1.5 * iqr)))
outliers = df[condi1 | condi2]
outliers['bmi']
```

✓ Dataframe에 값이 그 제한선보다 이상 이하인 값을 조건문을 이용해 outliers 변수에 넣어주었다

```
116     49.06
286     48.07
401     47.52
543     47.41
847     50.38
860     47.60
1047    52.58
1088    47.74
1317    53.13
Name: bmi, dtype: float64
```

	age	sex	bmi	children	smoker	region	charges	age_bin
	58	male	49.06	0	no	southeast	11381.32540	9
	46	female	48.07	2	no	northeast	9432.92530	7
	47	male	47.52	1	no	southeast	8083.91980	7
543	54	female	47.41	0	yes	southeast	63770.42801	8
847	23	male	50.38	1	no	southeast	2438.05520	2
860	37	female	47.60	2	yes	southwest	46113.51100	5
1047	22	male	52.58	1	yes	southeast	44501.39820	2
1088	52	male	47.74	1	no	southeast	9748.91060	8
1317	18	male	53.13	0	no	southeast	1163.46270	1

```
df.drop(outliers.index, axis=0, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1329 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1329 non-null   int64
 1   sex       1329 non-null   object
 2   bmi       1329 non-null   float64
 3   children  1329 non-null   int64
 4   smoker    1329 non-null   object
 5   region    1329 non-null   object
 6   charges   1329 non-null   float64
 7   age_bin   1329 non-null   category
dtypes: category(1), float64(2), int64(2), object(3)
memory usage: 84.7+ KB
```

```
bmi_q1 = df['bmi'].quantile(q=0.25)
bmi_q3 = df['bmi'].quantile(q=0.75)
iqr = bmi_q3 - bmi_q1
```

✓ dataframe에 BMI를 가지고 와서 25% 구간에 값과 75% 구간에 값을 가져온다

```
condi1 = (df['bmi'] < (bmi_q1 - (1.5 * iqr)))
condi2 = (df['bmi'] > (bmi_q3 + (1.5 * iqr)))
outliers = df[condi1 | condi2]
outliers['bmi']
```

✓ Dataframe에 값이 그 제한선보다 이상 이하인 값을 조건문을 이용해 outliers 변수에 넣어주었다

```
116     49.06
286     48.07
401     47.52
543     47.41
847     50.38
860     47.60
1047    52.58
1088    47.74
1317    53.13
Name: bmi, dtype: float64
```

	age	sex	bmi	children	smoker	region	charges	age_bin
	58	male	49.06	0	no	southeast	11381.32540	9
	46	female	48.07	2	no	northeast	9432.92530	7
	47	male	47.52	1	no	southeast	8083.91980	7
543	54	female	47.41	0	yes	southeast	63770.42801	8
847	23	male	50.38	1	no	southeast	2438.05520	2
860	37	female	47.60	2	yes	southwest	46113.51100	5
1047	22	male	52.58	1	yes	southeast	44501.39820	2
1088	52	male	47.74	1	no	southeast	9748.91060	8
1317	18	male	53.13	0	no	southeast	1163.46270	1

```
df.drop(outliers.index, axis=0, inplace=True)
df.info()
```

✓ Drop을 써서 이상치를 삭제한다 (axis = 0 행을 말함. inplace =true 는 dataframe에서 삭제한걸 모두 dataframe에 반영

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1329 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
                                  64
                                  ect
                                  at64
                                  64
 4   smoker    1329 non-null   object
 5   region    1329 non-null   object
 6   charges   1329 non-null   float64
 7   age_bin   1329 non-null   category
dtypes: category(1), float64(2), int64(2), object(3)
memory usage: 84.7+ KB
```

EDA - Outlier Detection

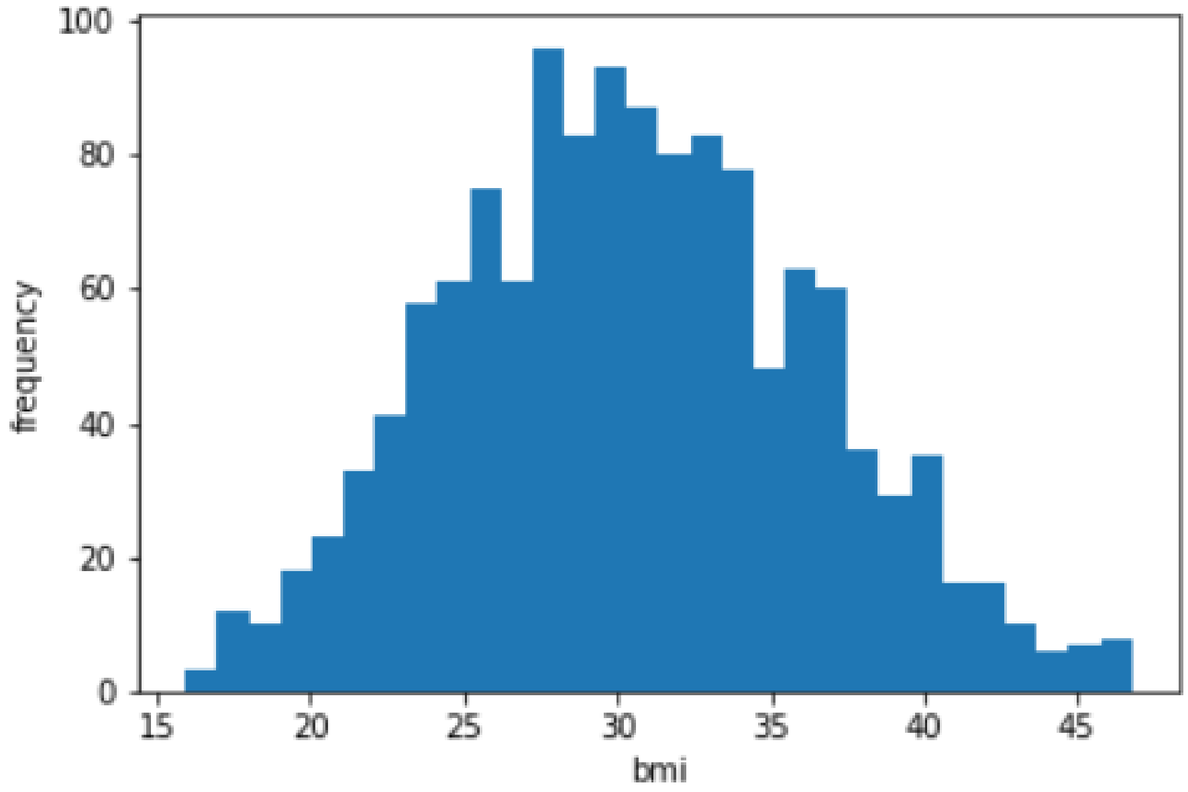
이상치 제거 후 BMI 분포

```
plt.figsize = (10, 10)
plt.hist(df['bmi'], bins=30)
plt.xlabel('bmi')
plt.ylabel('frequency')
plt.show()
```

→ 이상치를 제거하고 나서 BMI의 분포를 다시 살펴보면

이전에 왼쪽에 조금 치우쳐져 있던 것에 비해

미세하게 개선 됨



8. Scaling

스케일링을 통한 단위 통합

Scaling

“변수의 단위를 조정하기 위해 변수 바꾸는 작업”

대표적으로 표준화와 정규화 있다

표준화

평균을 0, 표준편차를 1로 맞춰주는 작업

$$\frac{X - \bar{X}}{\sigma_X}$$

정규화

변수의 범위를 [0,1] 사이로 옮기는 작업

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

→ 반드시 훈련 데이터만 fitting을 해주고, 이후에 정답 데이터와 동시에 transform

테스트 데이터까지 훈련을 시키면 학습데이터에는 존재하지 않는 평균과 표준편차를 이용하게 되어 학습이 되지 않는다

(데이터 누수)

Scaling

“변수의 단위를 조정하기 위해 변수 바꾸는 작업”

표준화 모듈

StandardScaler()

- ✓ Fit() - 변수의 평균과 표준 편차를 학습시킴
- ✓ Transform() - 학습된 평균과 표준편차를 바탕으로 실제 변수를 변환함
- ✓ Fit_transform() - 둘을 한번에 진행

→ 반드시 훈련 데이터만 fitting을 해주고, 이후에 정답 데이터와 동시에 transform

테스트 데이터까지 훈련을 시키면 학습데이터에는 존재하지 않는 평균과 표준편차를 이용하게 되어 학습이 되지 않는다

(데이터 누수)

9. Normal Transformation

분포 및 단위 맞추기

Normal transformation

“ 왜 필요할까? ”

Ans) 스케일링은 단위만 변환해주는 것이지 분포 형태를 바꾸는 것은 아니다. 머신 러닝의 학습이 잘 되기 위해서는 독립변수가 최대한 정규분포를 따를 수록 좋다. 회귀 분석은 특히 변수 간의 분포가 대칭일 수록 좋다는 말이다.

→ 정규 분포에 가깝도록 변환

Normal transformation

로그 변환

변수에 로그를 취해주면 정규 분포의 형태와 가깝게 변환되는 경우가 있다.
모듈은 np.log를 사용하고 변수에 0이 포함된 경우 np.log1p를 사용한다

Power transformation

특정 수식에 따라 변수를 변환
Box-cox는 0보다 큰 변수에 대해서만 사용할 수 있다
Yeo jonson는 변수의 부호와 관계 없이 사용할 수 있다 sklearn에 preprocessing의 power_transform 라이브러리

Quantile transformation

수학적으로 확률 값이 주어지면 미리 정한 분포에 따라 그에 상응하는 x값을 반환하는 함수.
변수에 있는 데이터들의 순위가 매겨지고, 분석자가 지정한 분포 형태로 바뀐다

Normal transformation

3가지 방법 모두 진행

```
to_scale = ['age', 'bmi', 'children', 'charges']
df_to_scale = df[to_scale].copy()

quantile = QuantileTransformer(n_quantiles=100, random_state=42,
output_distribution='normal')
power = PowerTransformer(method='yeo-johnson')
q_scaled = quantile.fit_transform(df_to_scale)
yj = power.fit_transform(df_to_scale)

q_scaled_df = pd.DataFrame(q_scaled, columns=to_scale)
scaled_df = pd.DataFrame(yj, columns=to_scale)
logged_df = pd.DataFrame(np.log1p(df_to_scale), columns=to_scal
e)
```

이 모듈은 지정된 분위수에 맞게 데이터를 변환

→ n_quantiles 매개변수를 통해서 변경 가능

Output distribution을 normal로 설정하면 내부 함수를 이용해서 정규분포로 변환,

random_state는 무작위 변수이지만 특정 규칙이 있는 랜덤 난수를 이야기한다

- ✓ 첫번째 행은 숫자형 데이터
- ✓ 두번째행은 power을 사용한 변환 데이터
- ✓ 세번째 행은 quantiles로 변환한 데이터
- ✓ 네번째는 로그 변환을 이용한 변환 데이터

```
for i in range(4):
    idx = 0
    for j in range(4): # subplot들의 열(column)
        colname = to_scale[idx]
        if i == 0:
            ax[i][j].hist(df_to_scale[colname], bins=30)
            ax[i][j].set_xlabel(colname)
            ax[i][j].set_ylabel('Frequency')
        elif i == 1:
            ax[i][j].hist(scaled_df[colname], bins=30)
            ax[i][j].set_xlabel(colname)
            ax[i][j].set_ylabel('Transformed Frequency')
        elif i == 2:
            ax[i][j].hist(q_scaled_df[colname], bins=30)
            ax[i][j].set_xlabel(colname)
            ax[i][j].set_ylabel('Transformed Frequency')
        elif i == 3:
            ax[i][j].hist(logged_df[colname], bins=30)
            ax[i][j].set_xlabel(colname)
            ax[i][j].set_ylabel('Logged Frequency')


        idx += 1
```

Normal transformation

3가지 방법 모두 진행

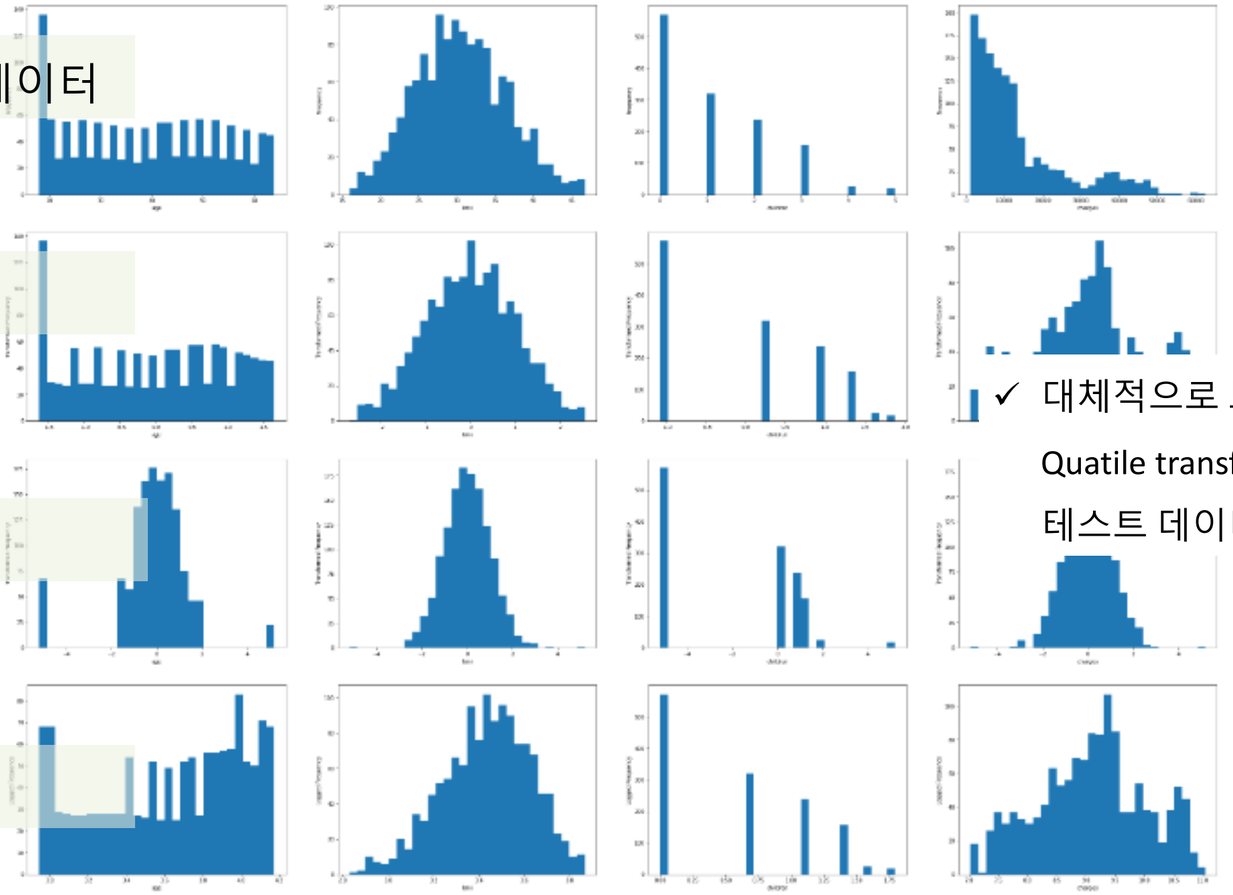
✓ BMI가 특히 정규분포와 가까워 짐

✓ 첫번째 행은 숫자형 데이터

✓ Power transform

✓ Quantiles

✓ 로그 변환



✓ 대체적으로 모두 정규분포에 가까운 Quatile transformations를 훈련 데이터셋과 테스트 데이터 셋으로 사용

Normal transformation

Quantile로 단위 조정

✓ train_Test_split을 사용하여 무작위로 75%
25% 비율로 train data, test data를 나누어 줌

```
X = df.drop(['charges'], axis=1)
y = df['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_s
ize=0.25, shuffle=True, stratify=X['age_bin'])
```

✓ Stratify는 나이 범위를 기준으로 하여
한쪽에 쏠려서 분배되지 않도록 도와 줌

```
to_scale = ['age', 'bmi']

quantile = QuantileTransformer(n_quantiles=10, random_state=0, o
utput_distribution='normal')

for col in to_scale:
    quantile.fit(X_train[[col]])
    X_train[col] = quantile.transform(X_train[[col]]).flatten()
    X_test[col] = quantile.transform(X_test[[col]]).flatten()
```

```
print(X_train.isnull().sum())
X_train.head()
```

✓ Quantile transformer 모듈을 이용해서 age와
BMI변수를 가지고 학습한 후 transform →
x_train과 x_test 데이터에 넣음

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

	age	sex	bmi	children	smoker	region	age_bin
13	1.044409	female	1.353174	0	no	southeast	9
623	-5.199338	male	0.502665	0	yes	northeast	1
1249	-0.372904	male	0.515705	1	yes	northeast	4
240	-0.923867	female	0.969904	2	yes	northeast	2
585	-0.316302	female	-0.363800	1	no	southeast	4

✓ X_train data 와 X_test data에 빈 값 확인
✓ 데이터 프레임에 잘 적용되었는지 확인

```
print(X_test.isnull().sum())
X_test.head()
```

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

	age	sex	bmi	children	smoker	region	age_bin
12	-0.923867	male	0.624976	0	no	southwest	2
140	-0.260697	male	-1.284891	2	no	northeast	4
255	0.967422	female	-0.828215	3	no	northeast	8
1155	-0.151689	female	-1.310406	3	no	northeast	5
253	-0.645631	male	-0.049434	3	no	southwest	3

Normal transformation

StandardScaler로 표준화 진행

```
cols = ['age', 'bmi', 'children']
for col in cols:
    std = StandardScaler()
    std.fit(X_train[[col]])
    X_train[col] = std.transform(X_train[[col]]).flatten()
    X_test[col] = std.transform(X_test[[col]]).flatten()

X_train.describe()
```

	age	bmi	children
count	9.960000e+02	9.960000e+02	9.960000e+02
mean	-2.457873e-17	-3.344045e-18	-2.675236e-18
std	1.000502e+00	1.000502e+00	1.000502e+00
min	-3.138085e+00	-5.574101e+00	-9.174707e-01
25%	-3.270751e-01	-7.131080e-01	-9.174707e-01
50%	6.816732e-02	-4.548806e-04	-7.912133e-02
75%	4.986938e-01	6.955858e-01	7.592281e-01
max	3.281032e+00	5.567415e+00	3.274276e+00

정규 분포에 가깝도록 변환을 시도한 후, 숫자형 데이터인 Age ,BMI, children 표준화를 진행한다.

StandardScaler을 이용하여 마찬가지로 fit 과 transform을 진행해주면 아래처럼 모든 평균은 0 표준편차는 1로 맞출 수 있다

One hot encoding

```
onehot_sex = OneHotEncoder()
sex_train = X_train[['sex']]
sex_test = X_test[['sex']]

onehot_sex.fit(sex_train) # sklearn의 OneHotEncoder는 2차원 배열
(또는 데이터프레임)이 input으로 들어가야 합니다.
sex_train_onehot = onehot_sex.transform(sex_train).toarray()
sex_test_onehot = onehot_sex.transform(sex_test).toarray()

X_train['sex'] = sex_train_onehot[:, 1].astype(np.uint8) # 여성
을 0, 남성을 1로 두고, 정수형으로 바꾸겠습니다.
X_test['sex'] = sex_test_onehot[:, 1].astype(np.u

X_train.head()
```

- ✓ 범주형 변수 처리 - 범주형 변수에는 성별 거주지역 흡연자 여부가 있었다.
- ✓ **One hot encoding을** 이용해 성별을 바꿔본다.

	age	sex	bmi	children	smoker	region	age_bin
13	0.716189	0	1.446497	-0.917471	no	southeast	9
623	-3.138085	1	0.535231	-0.917471	yes	northeast	1
1249	-0.158721	1	0.549202	-0.079121	yes	northeast	4
240	-0.498831	0	1.035848	0.759228	yes	northeast	2
585	-0.123780	0	-0.393131	-0.079121	no	southeast	4

One hot encoding

```
onehot_smoker = OneHotEncoder()
smoker_train = X_train[['smoker']]
smoker_test = X_test[['smoker']]

onehot_smoker.fit(smoker_train) # sklearn의 OneHotEncoder는 2차원 배열(또는 데이터프레임)이 input으로 들어가야 합니다.
smoker_train_onehot = onehot_smoker.transform(smoker_train).toarray()
smoker_test_onehot = onehot_smoker.transform(smoker_test).toarray()

X_train['smoker'] = smoker_train_onehot[:, 1].astype(np.uint8) # 비흡연자를 0, 흡연자를 1로 두겠습니다.
X_test['smoker'] = smoker_test_onehot[:, 1].astype(np.uint8)

X_train.head()
```

✓ 흡연 여부도 성별과 똑같은 방식으로 one hot encoding 진행.

	age	sex	bmi	children	smoker	region	age_bin
13	0.716189	0	1.446497	-0.917471	0	southeast	9
623	-3.138085	1	0.535231	-0.917471	1	northeast	1
1249	-0.158721	1	0.549202	-0.079121	1	northeast	4
240	-0.498831	0	1.035848	0.759228	1	northeast	2
585	-0.123780	0	-0.393131	-0.079121	0	southeast	4

One hot encoding

“ 정답 이외의 값들은 False ”

정답인 값 이외의 값들은 False 즉 0으로 만들어주는 것이다.

대부분의 머신 러닝에서는 문자열을 입력 값으로 허락하지 않기 때문에 모든 문자열 값들은 숫자형으로 인코딩하는 전처리 작업 후에 학습을 시킨다..

→ Dummy 변수

One hot encoding

✓ 일단 거주 지역 종류를 숫자형으로 바꿔주기 위해서 labelencoder을 사용한다.

```
label_region = LabelEncoder()
label_region.fit(X_train['region']) # L
input으로 들어가야 합니다.

X_train['region'] = label_region.transform(X_train['region'])
X_test['region'] = label_region.transform(X_test['region'])

X_train.head()
```

	age	sex	bmi	children	smoker	region	age_bin
13	0.716189	0	1.446497	-0.917471	0	2	9
623	-3.138085	1	0.535231	-0.917471	1	0	1
1249	-0.158721	1	0.549202	-0.079121	1	0	4
240	-0.498831	0	1.035848	0.759228	1	0	2
585	-0.123780	0	-0.393131	-0.079121	0	2	4

주거여부는 4개의 카테고리

각 주거지에 해당할 때 그 고유 값에 1이 들어 감.

→ 즉 이 거주지역에 사는지 아닌지

여기서 주의해야 할 점은 전체 카테고리 개수보다

더미 변수는 꼭 하나 적어야 함

→ 거주지역의 더미 변수는 3개이여야 한다.

Why?

모든 더미 변수가 포함이 되면 수학적으로

완전 다중 공선성이라는 것이 발생

→ 분석이 불가능 해진다.

One hot encoding

```
# one-hot encoding으로 거주지역 바꾸기
onehot_region = OneHotEncoder()
region_train = X_train[['region']]
region_test = X_test[['region']]

onehot_region.fit(region_train) # sklearn의 OneHotEncoder는 2차원 배열(또는 데이터프레임)이 input으로 들어가야 합니다.
region_train_onehot = onehot_region.transform(region_train).toarray()
region_test_onehot = onehot_region.transform(region_test).toarray()

X_train['region_1'] = region_train_onehot[:, 1].astype(np.uint8) # northwest이면 1, 아니면 0
X_train['region_2'] = region_train_onehot[:, 2].astype(np.uint8) # southeast이면 1, 아니면 0
X_train['region_3'] = region_train_onehot[:, 3].astype(np.uint8) # southwest이면 1, 아니면 0

X_test['region_1'] = region_test_onehot[:, 1].astype(np.uint8)
X_test['region_2'] = region_test_onehot[:, 2].astype(np.uint8)
X_test['region_3'] = region_test_onehot[:, 3].astype(np.uint8)

# 이제 문자열로 된 region 변수와 연령구간 변수는 중복을 제거하기 위해 배제하겠습니다.
X_train.drop(['region', 'age_bin'], axis=1, inplace=True)
X_test.drop(['region', 'age_bin'], axis=1, inplace=True)
```

✓ 지역이 dummy data 로 잘 전처리 되었다

	age	sex	bmi	children	smoker	region_1	region_2	region_3
13	0.716189	0	1.446497	-0.917471	0	0	1	0
623	-3.138085	1	0.535231	-0.917471	1	0	0	0
1249	-0.158721	1	0.549202	-0.079121	1	0	0	0
240	-0.498831	0	1.035848	0.759228	1	0	0	0
585	-0.123780	0	-0.393131	-0.079121	0	0	1	0

one hot encoding은 2차원 배열이 들어가야 하기 때문에 2차원 배열 형태로 region_train과 resion_test에 값 넣기.

↓

Fit, Transform을 한 후
astype으로 단위 정수형으로 바꾸기

↓

Drop()을 통해
사용할 데이터 이외의 데이터는 모두 제거

완전 다중 공선성?

“ 독립 변수들 간의 선형 상관 관계 ”

즉 독립 변수 간의 완벽한 선형 조합으로 표현될 수 있는 경우

주로 가공된 자료에서 발생 → **Dummy 변수**

✓ 쉬운 예시

X_2	X_3	X_3^*
10	50	52
15	75	75
18	90	97
24	120	129
30	150	152

X_3은 X_2에 5를 곱해서 만들어진 변수

$5X_2 - 1X_3 = 0$ 의 관계가 성립 → 상관계수가 1인 완전 다중 공선성이 발생

임의의 오차를 더한 X_3^* → 상관 계수가 0.9959

→ 완전 다중 공선성은 발생하지 않음

따라서 완전 다중 공선성에 경우, 회귀 계수가 추정될 수 없고 표준 오차가 무한대가 된다. 회귀 계수는 추정되어도 표준 오차가 크게 발생하여 추정의 정확도가 떨어지게 된다

10. Model Selection

회귀 모델 설정하기

Model Selection
모델 선택하기

✓ 사용할 모델 종류

- Linear model: Linear Regression, Elastic Net
- Decision Tree
- Ensemble(앙상블): Random Forest, AdaBoost, Gradient Boosting
- Boosting: XGBoost, LightGBM, CatBoost

✓ 사용할 모델 데이터를 모두 선언
✓ 먼저 default 모델을 설정한 뒤 교차 검증을 통해 성능 평가를 진행한다
✓ 평가 지표 RMSE

```
lr = LinearRegression()
enet = ElasticNet(random_state=42)
dt = DecisionTreeRegressor(random_state=42)
rf = RandomForestRegressor(random_state=42)
ada = AdaBoostRegressor(random_state=42)
gbr = GradientBoostingRegressor(random_state=42)
xgb = XGBRegressor(random_state=42)
lgbm = LGBMRegressor(random_state=42)
cat = CatBoostRegressor(silent=True, random_state=42)

models = [lr, enet, dt, rf, ada, gbr, xgb, lgbm, cat]
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(996, 8) (996,)
(333, 8) (333,)
```

✓ shape함수는 행,열 개수 출력
✓ Train과 test data의 수가 잘 맞아야 함

의사 결정 트리

의사 결정 트리는 분류 규칙을 통해 데이터를 분류하는 것이다.

매 분기마다 변수 영역을 두개로 구분하여 노드가 설정된다.

리프 노드가 가장 섞이지 않은 상태로 완전히 분류되는 것

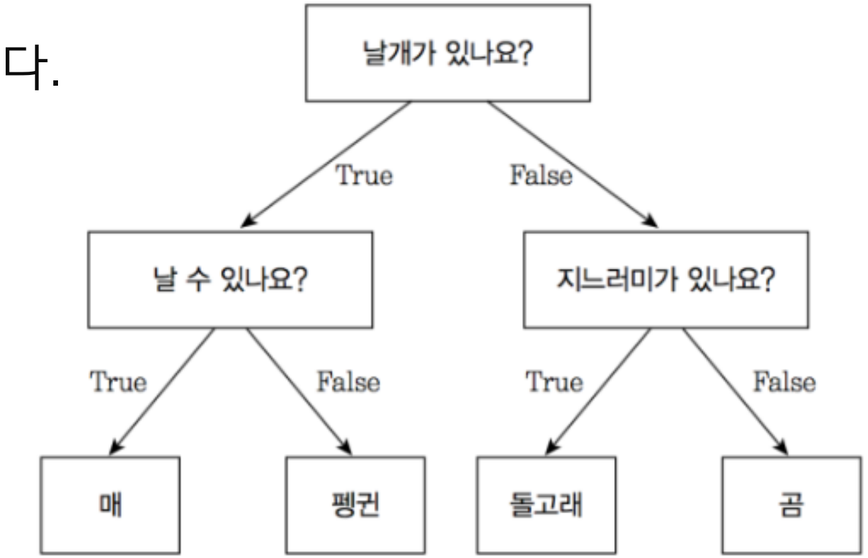
자식 노드가 만들어지면 계속 불순도가 감소되도록 설정

불순도를 수치적으로 나타낼 수 있는 방법

→ gini

→ 엔트로피

→ 두 값 모두 줄어드는 방향으로 트리를 형성



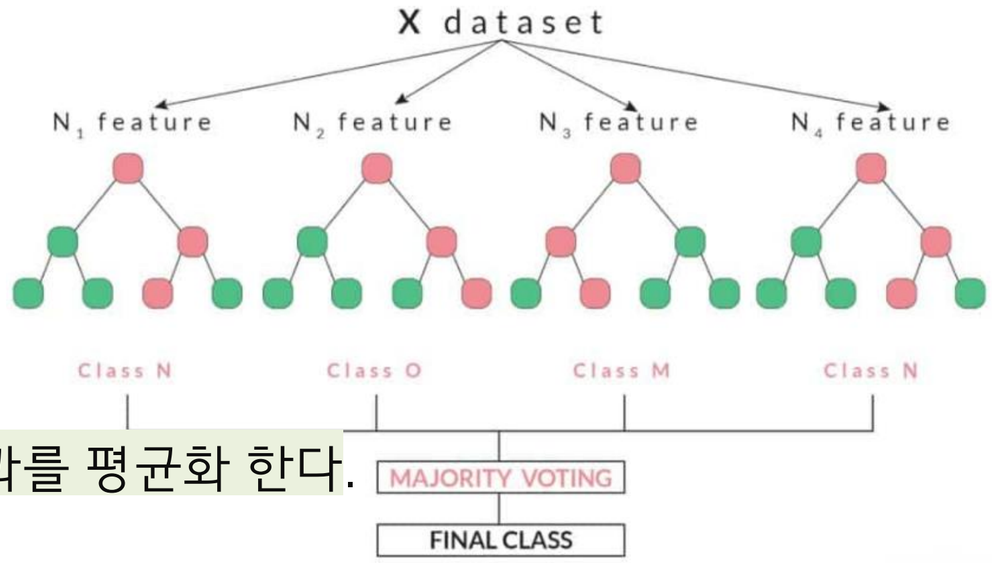
출처: 텐서 플로우 블로그

장점 - 수치형 범주형을 한번에 다룰 수 있음, 전처리를 하지 않아도 됨.

단점 - 한번의 하나의 변수만 고려해서 상호작용을 파악하기 어렵고

약간의 차이로도 트리 모양이 많이 달라질 수 있다.

랜덤 포레스트



의사 결정 트리를 개선하여 나온 것이 랜덤 포레스트.

의사결정 트리의 원리가 비슷하지만 차이점이 있다.

의사결정트리 → 의사결정에 사용하는 규칙이 있음

랜덤 포레스트 → 기능을 무작위로 선택하고 관찰

의사 결정트리를 여러 개 만들어 그 결과를 평균화 한다.

제대로 작동하기 위한 조건

- 트리로 만든 예측이 다른 트리들과 상관관계가 작아야 한다.

장점 - 회귀 분류에 모두 사용되고 과적합도 비교적 적다

단점 - 속도는 늦고 어떻게 이 결과에 도달했는지 방법이나 이유를 이해하기 어려움

에이다 부스팅

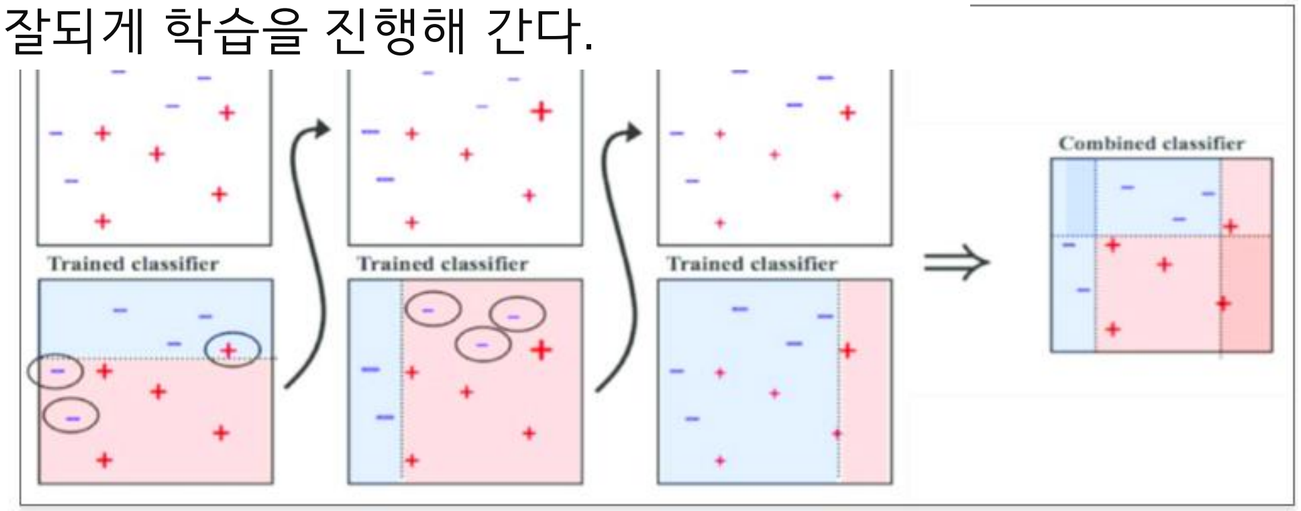
일단 부스팅은 예측 성능이 낮은 약한 분류기를 사용하여 더 좋은 성능을 보이는 하나의 강한 분류기를 만드는 것이다.

에이다 부스트 = adaptive + 부스팅

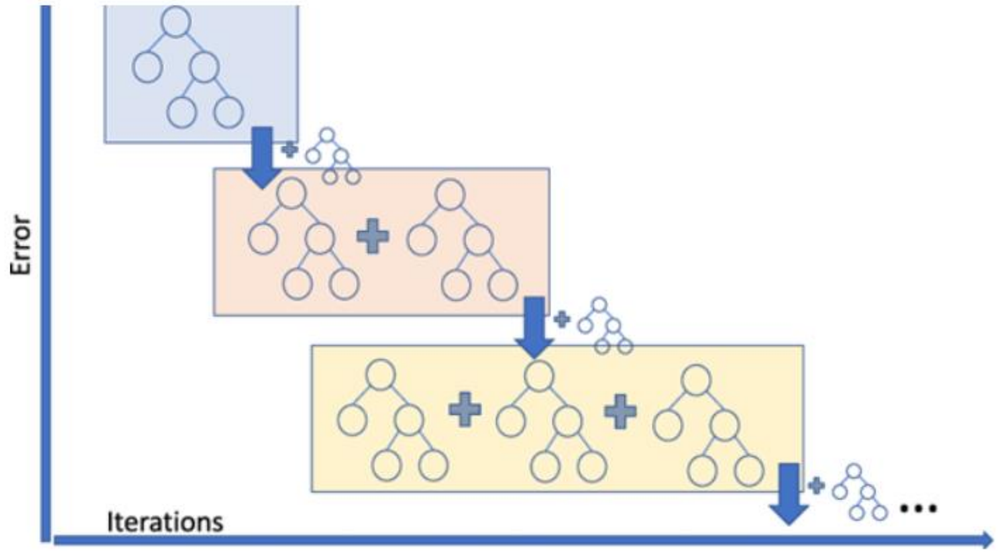
제대로 분류한 분류기와 잘 못한 정보를 다음 분류기에 전달

다음 분류기에서 잘 분류하지 못한 분류기의 가중치를 높인다

그 가중치를 적용하여 바꿔가며 학습이 더 잘되게 학습을 진행해 간다.



그레디언트 부스팅



이전에 살펴본 에이다와 유사하나,

그레디언트는 가중치 업데이트를 경사 하강법으로 한다.

경사 하강법은 반복 수행을 통해서 오류를 최소화할 수 있도록 가중치 업데이트 값을 도출하는 방법이다.

단점 - 학습이 조금 느림.

- Residual을 줄이는 방향으로 약한 학습기를 결합해 만들지만,

train data에 residual을 계속 줄여 과적합 될 수 있다

X그레디언트 부스트 &Lgbm

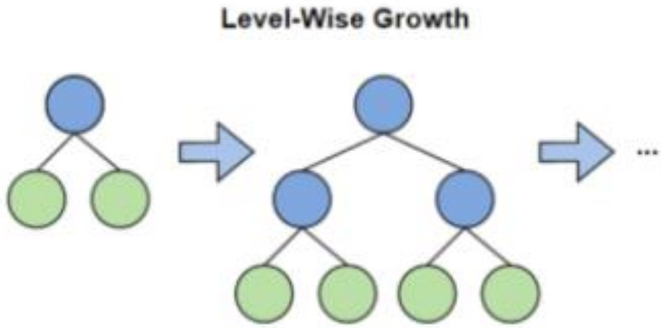
X그레디언트 부스트는 그레디언트 부스팅의 과적합을 막기위해서 regularization term을 추가한 알고리즘이다.

대부분 계산 비용은 각단계의 약한 학습기의 best tree를 찾는데 쓰인다
따라서 그 데이터 수만큼 전부 스캔해야 한다 비용이 너무 많이 들고 시간이 오래 걸려 이를 개선한 것이 light gbm이다

ElasticNet

선형 회귀 종류로 제약 조건을 주고 모델이 과도하게 학습되는 현상을 막는 것이다. 과적합은 너무 잘 적합하여 새로운 데이터가 조금만 바뀌어도 성능이 크게 변할 수 있다. 그래서 계수의 크기를 제한하는 방법으로 제약 조건을 추가하는 방법으로 진행한다

캣부스트



Yandex에서 개발된 오픈소스 머신 러닝

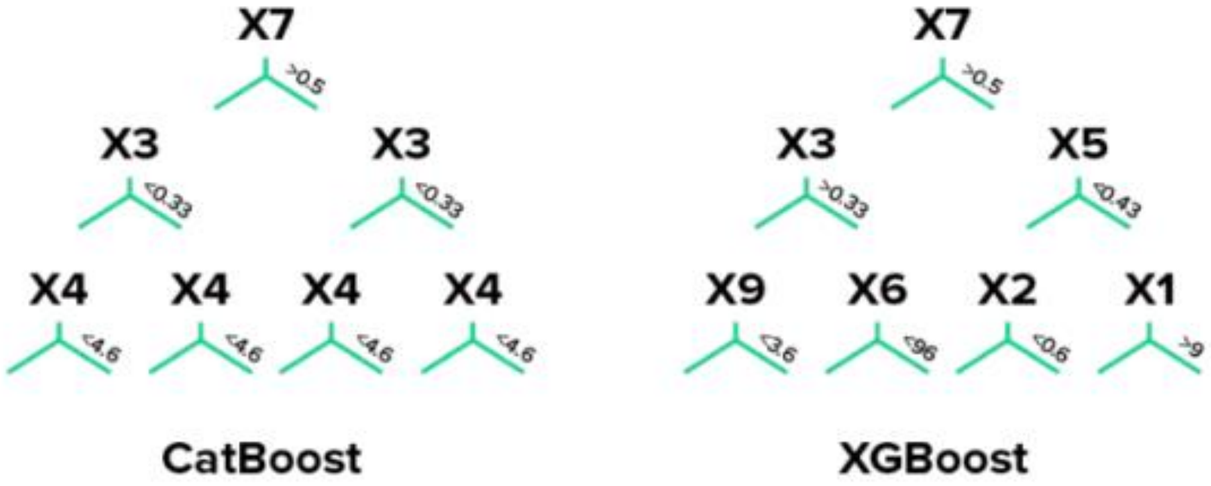
분류에서 높은 정확성과 높은 점수를 제공한다.

카테고리 + 부스팅 = **캣부스트**

→ 원래는 카테고리 특징을 원 핫 인코딩 해야 하지만

여기서는 다른 작업을 하지 않고 자동 변환하여 해준다.

→ 다른 부스팅에 비해 예측 시간이 빠름



Cross - Validation

“ 교차 검증 ”

교차 검증은 전체 데이터를 subset으로 나누고, k번의 평가를
진행하는데, test data set을 중복없이 바꿔가면서 진행

→ 교차 검증은 cross_val_sore 모듈을 사용

→ 이 모듈은 score이 가장 큰 값을 return

RMSE

“ 모델 최적화를 위한 오차 함수 ”

일단 선형 회귀에서 사용

선형 회귀는 예측선을 긋는 것이다.

일차함수 방정식 형태로,

y = ax + b 에서 a와 b의 가장 최적인 값을

찾아내는 것이다

→ 단일 변수인 경우 최소 제곱 법 사용

But 변수가 여러 개

→ 오차를 가장 최소화하는 최적선을
그리기 위해 평균 제곱근 오차를 사용

MSE

실제값 - 예측값은 음수를 가지고 있을 수 있다.

따라서 제곱을 해서 합을 구하고, 평균을 구한다.

→ 이 값을 최소한으로 만들어줘야 함.

RMSE

평균 제곱 오차 값이 너무 커지면 연산 속도가

느려짐 → 루트를 씌워 줌

Model Selection

평가 지표 RMSE

✓ 모든 모델을 for문으로 교차 검증을 진행하고 음의 점수에 다시 -1을 곱해서 양의 점수로 나타내 준다

✓ cv = 교차 검증 (몇 번 검증할 것인가), n_jobs는 병렬로 실행할 작업 수인데 -1은 전부라는 뜻

```
# 평가지표를 RMSE로 삼겠습니다.
for model in models:
    name = model.__class__.__name__
    scores = cross_val_score(model, X=X_train, y=y_train, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)
    mse = (-1) * np.mean(scores) # negative mean squared error로
설정했으므로 -1을 곱해 부호를 맞춰줍니다.
    print('Model %s - RMSE: %.4f' % (name, np.sqrt(mse)))
```

```
Model LinearRegression - RMSE: 6338.1372
Model ElasticNet - RMSE: 9688.1055
Model DecisionTreeRegressor - RMSE: 6803.5529
Model RandomForestRegressor - RMSE: 4845.0105
Model AdaBoostRegressor - RMSE: 5040.8203
Model GradientBoostingRegressor - RMSE: 4577.6611
Model XGBRegressor - RMSE: 5249.4960
Model LGBMRegressor - RMSE: 4820.5552
Model CatBoostRegressor - RMSE: 4743.4723
```

MSE는 오차의 평균이기 때문에 작은 숫자일수록 좋다.

→ neg mean squared error는 값에 음수를 취해, 가장 작은 값이 가장 높은 순위(score)가 되도록 해준다

Model Selection

평가 지표 RMSE

✓ 모든 모델을 for문으로 교차 검증을

```
# 평가지표를 RM
for model in m
    name = moc
    scores = c
scoring='neg_m
    mse = (-1)
설정했으므로 -1
    print('Moc
```

MSE는

→ neg

음수를

순

→확인 결과 gradient boost, Regressor가 가장 좋은 성능 그

밑으로는 LightGBM CatBoost 그리고 Random forest 가 좋음.

네 개의 모델을 가지고 hyper parameter 튜닝을 진행

가),

인데

529

105

77.6611

11. Hyper parameter Tuning

GridSearchCV를 이용한 모델 튜닝

Hyper parameter Tuning

“ 좋은 성능을 위한 튜닝 과정 ”

튜닝은 좀 더 좋은 성능을 내도록 하기 위해 조금씩 값을 변경한다. 여기서는 grid search CV를 사용한다. Param 값이 리스트 형태로 되어 있는데 순차적으로 진행된다

→ Gridsearchcv() 사용

일일이 직접 하나씩 튜닝하지 않아도 알아서 튜닝을 하고 최적의 파라미터를 알려준다

Hyper parameter Tuning

GridSearchCV

✓ Gradient boosting

```
# Gradient Boosting
gbr_params = {
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [50, 80, 100, 200, 300],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 3, 4, 5],
    'min_samples_leaf': [1, 2, 3, 5],
    'random_state': [42]
}

gbr_search = GridSearchCV(gbr, param_grid=gbr_params, cv=5, sco
ring='neg_mean_squared_error', n_jobs=-1)
gbr_search.fit(X_train, y_train)
best_mse = (-1) * gbr_search.best_score_
best_rmse = np.sqrt(best_mse)

print('Best score: {}, Best params: {}'.format(round(best_rmse,
4), gbr_search.best_params_))
```

- ✓ Learning rate 학습률
- ✓ n_estimators 학습 횟수
- ✓ max_depth 트리 깊이
- ✓ Min samples split 분할 할 수 있는 샘플 수

```
Best score: 4473.8525, Best params: {'learning_rate': 0.1,
'max_depth': 3, 'min_samples_leaf': 5, 'min_samples_split':
2, 'n_estimators': 50, 'random_state': 42}
```

✓ Random Forest

```
# Random Forest
rf_params = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 3, 4, 5],
    'min_samples_leaf': [1, 2, 3, 5],
    'random_state': [42]
}

rf_search = GridSearchCV(gbr, param_grid=rf_params, cv=5, scori
ng='neg_mean_squared_error', n_jobs=-1)
rf_search.fit(X_train, y_train)
best_mse = (-1) * rf_search.best_score_
best_rmse = np.sqrt(best_mse)

print('Best score: {}, Best params: {}'.format(round(best_rmse,
4), rf_search.best_params_))
```

```
Best score: 4548.2026, Best params: {'max_depth': 3, 'min_sa
mples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 100,
'random_state': 42}
```

- ✓ Best score와 함께 가장 성능이 좋았던 params를 추출할 수 있다

Hyper parameter Tuning

GridSearchCV

✓ LGBM

```
# LGBM
lgbm_params = {
    'num_leaves': [25, 31, 35],
    'learning_rate': [0.01, 0.05, 0.1, 0.5],
    'n_estimators': [100, 300, 500, 1000],
    'max_depth': [3, 7, 11],
    'random_state': [42]
}

lgbm_search = GridSearchCV(lgbm, param_grid=lgbm_params, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)
lgbm_search.fit(X_train, y_train)
best_mse = (-1) * lgbm_search.best_score_
best_rmse = np.sqrt(best_mse)

print('Best score: {}, Best params: {}'.format(round(best_rmse,
4), lgbm_search.best_params_))
```

✓ num leaves는 트리모델의 복합성을 컨트롤 , 오버피팅을 줄이기 위해 활용

```
Best score: 4457.7149, Best params: {'learning_rate': 0.05,
'max_depth': 3, 'n_estimators': 100, 'num_leaves': 25, 'rand
om_state': 42}
```

✓ Cat boosting

```
# CatBoost는 categorical feature를 parameter로 넣을 수 있습니다.
cat_features = ['sex', 'smoker', 'region_1', 'region_2', 'regio
n_3']

cat_params = {
    'learning_rate': [0.01, 0.05, 0.1, 0.5],
    'n_estimators': [100, 300, 500, 1000],
    'max_depth': [3, 7, 11],
    'cat_features': [cat_features],
    'silent': [True],
    'random_state': [42]
}

cat_search = GridSearchCV(cat, param_grid=cat_params, cv=5, sco
ring='neg_mean_squared_error', n_jobs=-1)
cat_search.fit(X_train, y_train)
best_mse = (-1) * cat_search.best_score_
best_rmse = np.sqrt(best_mse)

print('Best score: {}, Best params: {}'.format(round(best_rmse,
4), cat_search.best_params_))
```

✓ Cat_features에 카테고리 특징을 파라미터로 넣을 수 있다

```
Best score: 4488.9718, Best params: {'cat_features': ['sex',
'smoker', 'region_1', 'region_2', 'region_3'], 'learning_rat
e': 0.1, 'max_depth': 3, 'n_estimators': 100, 'random_stat
e': 42, 'silent': True}
```

12. Model Test

튜닝한 후 개선된 성능 테스트

Model Selection
평가 지표 RMSE

✓ Gradient boosting

```
gbr_final = GradientBoostingRegressor(**gbr_search.b
_)

gbr_final.fit(X_train, y_train)
y_pred_gbr = gbr_final.predict(X_test)

name = gbr_final.__class__.__name__

# Test RMSE
mse_gbr = mean_squared_error(y_test, y_pred_gbr
print('RMSE of %s: %.4f' % (name, np.sqrt(mse_(
```

Model GradientBoostingRegressor - RMSE: 4577.6611

```
RMSE of GradientBoostingRegressor: 4516.5654
```

✓ 이전 보다 성능이 개선되었음을 알 수 있다

✓ Random Forest

```
rf_final = RandomForestRegressor(**rf_search.best_params_)

rf_final.fit(X_train, y_train)
y_pred_rf = rf_final.predict(X_test)

name = rf_final.__class__.__name__

# Test RMSE
mse_rf = mean_squared_error(y_test, y_pred_
print('RMSE of %s: %.4f' % (name, np.sqrt(m
```

Model RandomForestRegressor - RMSE: 4845.0105

```
RMSE of RandomForestRegressor: 4621.7059
```

✓ Cat boosting

```
cat_final = CatBoostRegressor(**cat_search.best_params_)

cat_final.fit(X_train, y_train)
y_pred_cat = cat_final.predict(X_test)

name = cat_final.__class__.__name__

# Test RMSE
mse_cat = mean_squared_error(y_test, y_pred_cat)
print('RMSE of %s: %.4f' % (name, np.sqrt(mse_cat)))
```

Model CatBoostRegressor - RMSE: 4743.4723

```
RMSE of CatBoostRegressor: 4637.9920
```

✓ LGBM

```
lgbm_final = LGBMRegressor(**lgbm_search.best_para

_train, y_train)
y_pred_lgbm = lgbm_final.predict(X_test)

name = lgbm_final.__class__.__name__

# Test RMSE
mse_lgbm = mean_squared_error(y_test, y_pred_lgbm)
print('RMSE of %s: %.4f' % (name, np.sqrt(mse_lgbm)))
```

Model LGBMRegressor - RMSE: 4820.5552

```
RMSE of LGBMRegressor: 4527.8546
```

12. RESULT

결론 / 맺음말

결론

데이터 분석을 통해 나이가 종속변수와의 상관관계를 파악할 수 있었다
숫자형 데이터는 서로 단위와 분포를 맞추고, 범주형 데이터는 catboost 를 제외하고 one hot encoding을 진행해 주어야 머신 러닝 모델에 활용하였다,
교차 검증을 통해서 모델의 성능을 비교해 보았으며 xgboost , Gradient boost, random forest, cat boosting 이 가장 높게 나타났다.
네가지 모델의 hyper param 튜닝 과정을 통해서, predict 한 값을 가지고 RMSE를 줄여보았다. 네가지 모델 모두 성능이 개선되었음을 알 수 있다. 그중에서도 gradient boost가 가장 높은 성능을 보였다.

맺음말 / 소감

4차 산업 혁명으로 인해 인공지능에 대한 관심도가 높아지면서 다양한 분야에 융합되는 모습을 볼 수 있다. 보험사들도 4차 산업 혁명을 직감하고, 다양한 인공지능 모델을 통해 고객의 만족도를 개선하는 등 새로운 가치를 창출해 나간다. 또 다양한 플랫폼을 통해 고객에게 직접 다가가는 D2C(direct to customer)의 방향으로 나아가고 있다. 보험사 뿐만 아니라 앞으로 다양한 분야에서 이러한 방향으로 흘러갈 것이다. 따라서 4차 산업 혁명 시대에 맞춰서, 받아들이고 준비해 나아가야 한다….

데이터 분석을 목적으로 주제를 선정하다 보니 다양한 방법이 있는 내용을 선택하게 되었다. 한번에 너무 많은 정보량을 보는 것이 쉽지 않았다. 자세하게 알아보면 수학적인 부분과 통계적인 부분에서 기본기가 필요하다는 것을 느꼈다. 넓게 다룬 만큼 아주 깊게는 보지 못한 점이 아쉬운 것 같다. 그래도 혼자 하는 것 보다 체계적인 데이터 분석 경험을 통해 다른 주제도 분석해 보고 싶다. 그리고 머신 러닝 모델들에 특징에 대해 좀 더 자세하게 이해하고 싶다.

THANK YOU

EMAIL
yunjiyeong0106@gmail.com

CONTENT
kaggle.com

ABOUT
machine learning