# AI track 3주차

## 1. 데이터 분석이란? (Data Analysis)

⇒ 패턴 또는 관계를 찾기 위해 중요함

- 데이터의 성격을 파악하고 데이터 간의 패턴 또는 관계를 찾아내서 더 효과적인 모델을 만들기 위해 필수적임.
- 패턴과 관계는 변수들 사이에 존재함.

### 월마트 “기저귀 맥주 사례”

- **기저귀와 맥주**는 연관성 x
- 하지만 둘이 함께 구매하는 고객 많음

⇒ 고객들의 구매내역 분석 결과 아이가 있는 20~30대의 남자가 퇴근 무렵인 저녁 6시부터 8시 사이에 기저귀를 구매하는 경우 맥주를 함께 잘 구매하는 현상 발견

⇒ 기저귀와 맥주를 가까이 진열하여 소비자들이 구매를 빠르고 편리하게 만들어 구매율 높이는데 성공

### [데이터 분석 단계]

### 문제 정의 및 계획 -> 원시 데이터 수집 -> 데이터 전처리 - -> 탐색적 데이터 분석 -> 모델 & 알고리즘 -> 결과보고

1. **문제 정의 및 계획** : 문제가 명확해야 그 문제를 해결하기 위한 데이터가 어떤 것인지 추정할 수 있고, 어떤 분석기법을 적용해야 할지도 계획 가능
2. **원시 데이터 수집**: 엑셀 파일, 종이 문서, 필요하다면 크롤링 등의 방식으로 데이터 수집
3. **데이터 전처리**: 수집된 데이터는 대부분 RAW DATA(정제되지 않은 데이터)이므로 바로 분석에 사용할 수 없음. 단위 차이, 결측치, 오류 데이터 등이 존재하므로 보정이 필요함, 수집된 데이터를 분석이 가능한 정제된 데이터로 정돈하는 과정을 전처리 과정이라고 함.
4. **탐색적 데이터 분석 (EDA)**: 데이터의 특징과 내재하는 구조를 알아내기 위한 통계적 기법들을 말함. 데이터 수집 -> 시각화 탐색 -> 패턴 도출 -> 인사이트 발견 / 문제 정의 단계에서 미처 발견하지 못한 다양한 패턴을 발견하고 이를 통해 기존 가설 수정 및 새로운 가설 도출 가능, 데이터 분포 및 값 검토를 통해 데이터가 표현하는 현상을 더 잘 이해하고 데이터에 대한 잠재적인 문제 파악 가능

5. **모델 & 알고리즘**: 데이터 단계에서 파악한 정보를 바탕으로 심화한 분석을 수행하는 단계 머신러닝 기술 및 전통적인 통계 분석 등의 고급 분석 기법이 사용됨
6. **결과 보고**: 데이터 제품 또는 의사결정을 위한 의사소통 시각화 또는 보고서, 데이터 시각화가 해당 단계에서 중요함

1-3 단계는 여러 배경 지식과 다양한 컴퓨터 기술이 요구되므로 4-5단계에 집중하여 수업 진행

# 2. 회귀(Regression) vs Classification(분류)

## 데이터 분석의 종류

1. **지도 학습(supervised)**: 옳고 그름이 있어서 그것을 학습시키는 것
   - 결과를 알고 있는 데이터를 사용
     - ex) titanic data 사용
     - ⇒ 인공지능으로 하여금 어떤 사람이 살았는지 죽었는지를 학습시키고 그것을 다시 예상하게 만들음
2. **비지도 학습(unsupervised)**: 옳고 그름이 없음
   - 분류화가 되지 않은 데이터에서 숨겨진 패턴을 찾아낼 때가 많음
   - 예측할 변수가 존재하지 않는다.

**<modeling>**

modeling을 한다는 것은 변수들간의 존재하는 존재들간의 대표적인 관계를 찾아내는 것을 의미함

⇒ 우리가 풀어야하는 문제가 무엇인지에 따라서 사용해야하는 매커니즘이 상당히 다름

### 모델을 사용하는 두가지 이유

1. **예측** : classification 분류(죽었냐 살았냐), 회귀(집값을 예측)
2. **설명**
   a. 수입이 돈을 갚는 것과 영향이 있는가 등

⇒ 우리는 **지도학습 중 예측의 회귀와 분류**에 대해 알아볼 것임

## 회귀 vs 분류

**⇒ 회귀와 분류는 결국 둘다 본질적으로 prediction(예측)을 위한 것임**

**회귀 - 수치형 데이터**

**수치형 데이터**: 연속형 , 이산형

- 어떤 수치 값을 예측하려고 하는데, 그 수치값이 이론적으로는 무한히(연속)되는 것

**분류 - 범주형 데이터**

범주형 데이터: 서열형, 명목형

# 3. 선형 회귀(Linear Regression)

## 1. Linear Regression(선형 회귀)

**<선형회귀의 종류>**

- **단순 선형회귀**: 한개의 독립변수를 사용하여 종속변수 예측
  - f(x) = Wx + b (W: 가중치, b: 편향성)
- **다중 선형회귀**: 두 개 이상의 독립변수를 사용하여 종속변수 예측
  - f(x, y, z) = W0 + W1x + W2y+ W3z

⇒ 학습을 통해 최적의 가중치(w)와 편향성(b)를 찾아냄

최적의 가중치와 편향성: 모든 데이터로부터 나타나는 오차의 평균을 최소화함

*오차의 평균

**MSE**(Mean squared Erro, 평균 제곱 오차)

**MAE**(Mean Absolute Error, 평균 절대 오차)

후버 손실(Huber loss, MSE와 MAE를 절충한 오차)

- 손실함수 or 비용함수: 직선과 데이터 사이의 간격을 제곱하여 합한 값
- 모델이 만들어지고 나면 우리가 알아야 하는 값은 weight값 또는 intercept값인데 coef_attribute에 저장됨

**<선형 회귀 학습 과정>**

1. 처음에는 최적의 w(가중치), b(편향성)을 모르기 때문에 임의의 값으로 시작해서, 점점 성능을 개선시켜가며 최적의 값에 도달
   a. 첨에 최적의 값을 모르기에 선을 하나 그어놓고 그 다음 계속하면서 최적의 값을 찾아가는 과정

i. 독립변수가 하나일 때: 단순 선형회귀인 경우→ 손실함수(비용함수)가 일차함수로 그려짐

ii. 독립변수가 여러개 일 때: 다중 선형회귀인 경우 → 손실함수가 일차함수로 그려지지 않으므로 경사하강법을 사용해서(이차방정식) 비용함수를 최소로 만들어감

1. **경사하강법(Gradient Descent Method):** 비용함수를 최소화하기 위해 반복해서 파라미터를 조정해가는 과정

⇒ 기울기 이용해서 계산

2. 학습을 마치면 최종적으로 가중치와 편향성을 도출할 수 있음

**<선형회귀의 장, 단점>**

**장점:**

1. 다른 기계학습 모델에 비해 상대적으로 학습이 빠름
2. 쉽고 간단한 설명 가능
3. 데이터가 많고 적음에 따른 큰 영향이 없음

**단점:**

1. 선형 모델의 특성상 과소적합이 일어나기 쉬움
2. 현실의 데이터는 선형관계보다는 비선형 관계가 많기 때문에 문제를 해결하지 못하는 경우가 많음

선형회귀는 overfitting(과대적합)을 피할 수 있는 방법이 없음

## 과소적합 vs 과대적합

- **과소적합(Underfitting)**
  - 모델 학습 시, 충분하지 못한 특징만으로 학습되어 특정 특징으로 편향되게 나타나는 것
  - 해결 방안: 데이터 수 증가, 학습에 사용되는 특징(열)의 개수 증가
- **과대적합(Overfitting)**
  - 너무 많은 특징들을 알려줘서 학습 데이터에 해당하는 것만 정확도가 높음
  - 학습 데이터에 대한 정확도는 매우 높지만 테스트 데이터 정확도가 낮게 나옴
  - 해결 방안: 데이터 수 줄이기, 전처리 등

---

# 4. 데이터 설명: Boston Housing

## 칼럼 설명: Boston Housing

- CRIM: 자치시(town) 별 1인당 범죄율
- ZN: 25,000 평방피트를 초과하는 거주지역의 비율
- INDUS:비소매상업지역이 점유하고 있는 토지의 비율
- CHAS: 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
- NOX: 10ppm 당 농축 일산화질소
- RM:주택 1가구당 평균 방의 개수
- AGE: 1940년 이전에 건축된 소유주택의 비율
- DIS: 5개의 보스턴 직업센터까지의 접근성 지수
- RAD: 방사형 도로까지의 접근성 지수
- TAX: 10,000 달러 당 재산세율
- PTRATIO: 자치시(town)별 학생/교사 비율
- B: 1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.
- LSTAT: 모집단의 하위계층의 비율(%)
- CMEDV: 본인 소유의 주택가격(중앙값) (단위: $1,000)

---

## 코드 설명 : 0. 데이터 불러오기

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

**pandas, numpy, matplotlib.pyplot, seaborn 파이썬 패키지 사용**

- **pandas**: 데이터 구조를 제공하는 파이썬 패키지
- **numpy**: 수학, 과학 연산을 위한 파이썬 패키지
- **matplotlib**: 데이터를 시각화하는 파이썬 패키지
- **seaborn:** matplotlib와 같이 데이터를 시각화하는 파이썬 패키지, 조금 더 예쁘게 가능

```
from sklearn import datasets
boston = datasets.load_boston()
```

- **Sklearn:** 파이썬을 대표하는 머신러닝 라이브러리
- 사이킷런에서 제공하는 보스턴 주택 가격 데이터를 boston 이라는 이름으로 불러옴

```
boston.keys() #데이터 key 값 확인하기
bt = pd.DataFrame(boston['data'],columns=boston['feature_names'])
target = pd.DataFrame(boston['target'],columns=['Target'])
df = pd.concat([bt,target],axis=1) #데이터 통합하기
```

- boston 데이터의 키값을 확인하고 data와 칼럼 이름으로 나눠져있는 데이터를 사용할 수 있게 데이터 통합 진행
- pd.concat() : 데이터 통합

```
df.info()
```

- 데이터 칼럼 및 특징 확인하기

---

## 코드 설명: 1. 문제 정의 및 계획

## 1. 문제 정의 및 계획

- 보스턴 지역 주택 가격에 영향을 미치는 요소들을 살펴보고, 보스턴 지역 주택 가격을 회귀 방식으로 예측하여 실제값과의 오차 비교해보기

```
df #위에서 데이터를 df라는 변수로 저장함
```

```
df # 데이터 확인하기
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.5380 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.4690 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.4690 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.4580 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.4580 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 |
| 5 | 0.02985 | 0.0 | 2.18 | 0.0 | 0.4580 | 6.430 | 58.7 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.12 | 5.21 | 28.7 |
| 6 | 0.08829 | 12.5 | 7.87 | 0.0 | 0.5240 | 6.012 | 66.6 | 5.5605 | 5.0 | 311.0 | 15.2 | 395.60 | 12.43 | 22.9 |
| 7 | 0.14455 | 12.5 | 7.87 | 0.0 | 0.5240 | 6.172 | 96.1 | 5.9505 | 5.0 | 311.0 | 15.2 | 396.90 | 19.15 | 27.1 |
| 8 | 0.21124 | 12.5 | 7.87 | 0.0 | 0.5240 | 5.631 | 100.0 | 6.0821 | 5.0 | 311.0 | 15.2 | 386.63 | 29.93 | 16.5 |
| 9 | 0.17004 | 12.5 | 7.87 | 0.0 | 0.5240 | 6.004 | 85.9 | 6.5921 | 5.0 | 311.0 | 15.2 | 386.71 | 17.10 | 18.9 |
| 10 | 0.22489 | 12.5 | 7.87 | 0.0 | 0.5240 | 6.377 | 94.3 | 6.3467 | 5.0 | 311.0 | 15.2 | 392.52 | 20.45 | 15.0 |
| 11 | 0.11747 | 12.5 | 7.87 | 0.0 | 0.5240 | 6.009 | 82.9 | 6.2267 | 5.0 | 311.0 | 15.2 | 396.90 | 13.27 | 18.9 |

```
df.describe() #평균, 최솟값, 최댓값, 사분위 수 등 확인하기
```

```
df.describe()
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | |
| mean | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.284634 | 68.574901 | 3.795043 | 9.549407 | 408.237154 | 18.455534 | 356.674032 | |
| std | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.702617 | 28.148861 | 2.105710 | 8.707259 | 168.537116 | 2.164946 | 91.294864 | |
| min | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 187.000000 | 12.600000 | 0.320000 | |
| 25% | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.025000 | 2.100175 | 4.000000 | 279.000000 | 17.400000 | 375.377500 | |
| 50% | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 77.500000 | 3.207450 | 5.000000 | 330.000000 | 19.050000 | 391.440000 | |
| 75% | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 94.075000 | 5.188425 | 24.000000 | 666.000000 | 20.200000 | 396.225000 | |
| max | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 | |

```
df.info() #행의 개수 차이가 존재하지 않음 => 동일하게 506개의 행이 존재함
```

```
df.info() #개수 확인해도 결측치 자체 존재하지 않음. 모두 동일하게 506개의 row가 존재
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  Target   506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

---

## 2. 원시 데이터 수집

- 생략
- 크롤링, 공공 데이터 수집 등 다양한 방법이 존재

---

## 3. 데이터 전처리

**전처리**란 ? 깔끔하지 않은 데이터를 머신러닝 즉 기계가 학습할 수 있도록 깔끔하게 만드는 것

**특히, 회귀 문제를 해결할 경우 회귀 문제에서 다룰 수 있는 데이터 = 수치형 데이터이므로**

**서열형, 명목형 데이터로 이루어진 범주형 데이터의 형식을 바꿔야함**

⇒ One Hot Encoding, Label encoding 등의 방식을 이용함

### 1) 데이터 결측값 확인하기

- (Null 값 확인)

```
df.isnull().sum() #결측치 자체가 존재하지 않음
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Target     0
dtype: int64
```

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

- 중복값 확인

```
df.duplicated().sum() #중복값 존재하지 않음
```

```
0
```

⇒ 현재 상황에서 할 수 있는 데이터 전처리가 없으므로, EDA 과정을 진행하면서 추가적인 데이터 결측값이 존재하는지 혹은 이상치가 존재하는지 파악해야함

---

## 4. 탐색적 데이터 분석 [EDA]

- 탐색적 데이터 분석 과정에서 데이터 특징과 내재하는 구조 알아내기 위해 데이터를 다룸
  데이터 수집 -> 시각화 탐색 -> 패턴 도출 -> 인사이트 발견

### 1) 가설 설정

⇒ 해당 데이터를 보고 궁금했던 점 혹은 주관적인 견해를 넣어 새로운 패턴 및 인사이트를 발견할 수 있다고 생각하는 부분을 가설로 설정하여 탐구

ex) 사회적 통념 상 흑인 비율이 많은 지역에 범죄율이 높을 것이라 보기에 해당 부분을 확인하고, 범죄율이 높을 경우 주택가격이 하락하기에 흑인 비율이 높은 지역의 주택 가격이 낮을 것이다

⇒ **가설 설정: 흑인 비율이 높은 지역에 주택 가격이 낮을 것이다.**

```
low = df[['B','CRIM','Target']] #흑인비율과 자치시 1인당 범죄율과 Target
low
```

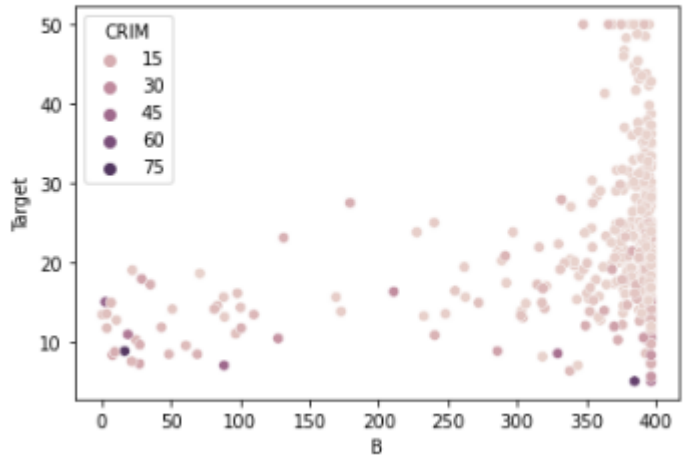
| | B | CRIM | Target |
|---|---|---|---|
| 0 | 396.90 | 0.00632 | 24.0 |
| 1 | 396.90 | 0.02731 | 21.6 |
| 2 | 392.83 | 0.02729 | 34.7 |
| 3 | 394.63 | 0.03237 | 33.4 |
| 4 | 396.90 | 0.06905 | 36.2 |
| 5 | 394.12 | 0.02985 | 28.7 |
| 6 | 395.60 | 0.08829 | 22.9 |
| 7 | 396.90 | 0.14455 | 27.1 |
| 8 | 386.63 | 0.21124 | 16.5 |
| 9 | 386.71 | 0.17004 | 18.9 |
| 10 | 392.52 | 0.22489 | 15.0 |
| 11 | 396.90 | 0.11747 | 18.9 |

```
low = df[['B','CRIM','Target']]
low
```

- low라는 변수에 df라는 데이터 셋에 저장되어있던 흑인비율, 범죄율, 그리고 주택가격(우리가 예측하고자 하는 것)을 저장하여 새로운 데이터 프레임으로 만들음

**시각화 (산점도로 확인하기)**

```
ax = sns.scatterplot(x='B',y='Target',hue='CRIM',data=low)
```




```
ax = sns.scatterplot(x='B',y='Target',hue='Crim',data=df)
```

- x축: 흑인 비율
- y축: 타겟(즉, 집 값)
- hue: 카테고리별로 다르게 나타낸다 . 여기서 hue를 범죄율이라 했으니, 내부적으로 범죄율을 단위별로 나눠서 색깔로 표시해줌. 색깔이 진할 수록 범죄율이 높은 것임

- **가설 검증**
  - ⇒ **가설 설정: 흑인 비율이 높은 지역에 주택 가격이 낮을 것이다.**
  - 해당 가설이 참이기 위해서는 우상향 그래프가 나타나야하는데 , 현재 해당 그래프를 보면 정확한 상관관계가 있다고 말하기는 어려움. 오히려 흑인 비율이 높을 수록 가격이 높은 것으로 나타남.
  - 범죄율과 흑인 비율간의 큰 연관은 없지만, 흑인 비율이 높을수록 집의 가격이 낮다는 것은 성립하지 않는 듯 보임. 하지만, 데이터 자체의 불균형이 존재할 수도 있기 때문에 흑인 인구 데이터를 자세히 탐색해봄

⇒ 흑인 비율 분포 자체가 몰려 있을 수도 있기 때문에 그래프를 통해 확인

```
# row 생략 없이 출력
pd.set_option('display.max_rows', None)
# col 생략 없이 출력
pd.set_option('display.max_columns', None)
```

```
Black=df['B'].value_counts()
Black
```

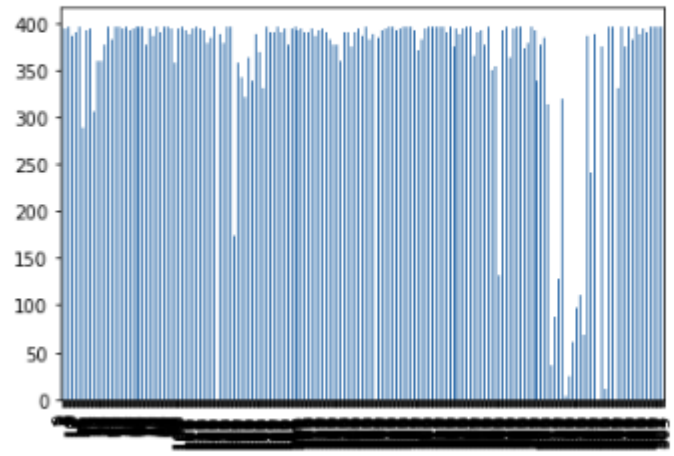
```
396.90    121
393.74      3
395.24      3
376.14      2
394.72      2
395.63      2
392.80      2
395.56      2
390.94      2
393.68      2
395.11      2
395.58      2
396.21      2
392.78      2
393.23      2
395.69      2
396.06      2
393.37      2
389.71      2
```

```
pd.set_option('display.max_rows',None)
```

⇒ 해당 코드를 통해 데이터를 눈으로 확인 해볼 수 있음

```
df['B'].plot.bar()
```

```
<AxesSubplot:>
```




```
df['B'].plot.bar()
```

- 흑인 비율 분포를 그래프를 통해 확인하기 ⇒ 현저히 낮은 값이 존재한다는 것을 발견

**⇒ 이상치 존재 확인**

이상치란? 데이터 분포 특성상 현저하게 낮거나 높은 값을 의미함

⇒ 이상치로 인해 기계 학습시 편향되어 학습될 가능성도 있으므로 다양한 방법(사분위수, scaling 등)을 통해 이상치 제거를 해줌

---

## 5. 모델 & 알고리즘

### 1) Train, Test 데이터 셋 나누기

⇒ 결국, 기계에 학습시키기 위해서는 어떤 데이터를 학습에 사용할 것인지, 어떤 데이터를 테스트 데이터로 사용할 것이지 나눠주는 작업이 필요함

```
from sklearn.model_selection import train_test_split
```

- 사이킷런에서 제공해줌
  - train_test_split: 데이터를 train 데이터 셋과 test 데이터 셋으로 분리해줌

```
feature_columns = list(df.columns.difference(['Target'])) #Target를 제외한 나머지 칼럼을 featrue_columns로 묶음
X = df[feature_columns]
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(379, 14) (127, 14) (379,) (127,)
```

- feature_columns에 target을 제외한 모든 특징 데이터를 저장
- X: 특징
- y: target(예측하고 싶은 주택 가격)
- **X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42)**
  - X_train, X_test, y_train, y_test 라는 이름으로 X, y의 데이터를 나눠줌
- print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
  - 데이터 프레임의 행과 열의 개수를 반환

**2) 모델(알고리즘) 선정: 선형 회귀**

- 어떤 알고리즘의 모델을 선정할지 고민
  - 우리의 목표 = 주택 가격 예측( 회귀)

⇒ **회귀 알고리즘** 사용: 선형회귀(linear regression)

**3) 모델 활용하여 학습 데이터 학습**

```
from sklearn import linear_model

linear_regression = linear_model.LinearRegression()
linear_regression.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
from sklearn import linear_model
linear_regression = linear_mode.LinearRegression()
linear_regression.fit(X_train,y_train)
```

- sklearn의 linearmodel 에서 제공하는 linearregression 모델
- linear_regression.fit(X_train, y_train)
  - X_train,y_train 데이터를 linear_regression로 학습

**4) 예측 값 확인**

```
prediction = linear_regression.predict(X_test)
p= pd.DataFrame(prediction)
p
```

| | 0 |
|---|---|
| 0 | 28.846163 |
| 1 | 36.002095 |
| 2 | 15.158330 |
| 3 | 25.227891 |
| 4 | 18.909692 |
| 5 | 23.254205 |
| 6 | 17.614401 |
| 7 | 14.338896 |
| 8 | 23.070873 |
| 9 | 20.635981 |
| 10 | 24.812991 |
| 11 | 18.704515 |

```
prediction = linear_regression.predict(X_test)
p = pd.DataFrame(prediction)
p
```

- linear_regression은 위에서 학습 데이터로 학습이 되었기 때문에 X_test 데이터를 활용해 y값을 예측

⇒ linear_regression.predict(X_test)

⇒ 학습한 데이터를 바탕으로 X_test를 통해 예측한 y 값을 출력함

```
pre_real = pd.concat([p,y],axis=1)
pre_real
```

| | predict | Target |
|---|---|---|
| 0 | 28.838854 | 24.0 |
| 1 | 36.007833 | 21.6 |
| 2 | 15.083248 | 34.7 |
| 3 | 25.230909 | 33.4 |
| 4 | 18.878641 | 36.2 |
| ... | ... | ... |
| 501 | NaN | 22.4 |
| 502 | NaN | 20.6 |
| 503 | NaN | 23.9 |
| 504 | NaN | 22.0 |
| 505 | NaN | 11.9 |

506 rows × 2 columns

- 예측한 값과 원래 값을 비교

**⇒ NaN 값이 존재하는 이유는?**

⇒ p값은 X_test값을 통해 예측한 y_test값이므로 126개만 예측 (나머지는 예측하지 않은것임) 반면에 y값 자체는 원래 506개의 데이터를 가지고 있으므로 밑부터는 값이 없다는 의미의 NaN 나타남

### 5) 모델 성능 평가

```
linear_regression.score(X_train,y_train)
```

```
0.7481438828141436
```

```
linear_regression.score(X_test,y_test)
```

```
0.6849551738716659
```

```
linear_regression.score(X_train, y_train)
```

```
linear_regression.score(X_test, y_test)
```

- linear_regression.score() : 모델의 성능을 평가