기계학습기초 프로젝트

(주택 가격 요소에 따른 부동산 가격 예측 모델 구현)

서강대학교 전자공학과

20151480 이지호

서강대학교 SOGANG UNIVERSITY

Index

Regression Model 구현을 위한 접근 방법

- Dataset : Boston Housing 1970 Dataset
- 우선적으로 Dataset의 각 변수에 따른 주택 가격이 어떠한 양상을 나타내고 있는지 Dataset에 대한 분석을 진행합니다.
- Google Colab에서 Python을 이용하여 Dataset 분석 / Preprocessing / Model Training / 평가에 필요한 라이브러리를 정의합니다.
- 이후, pandas 라이브러리를 통해 csv data를 불러오는 과정을 거쳤습니다.

```
# Google Drive와 연동하여 코드 및 Model 관리를 편리하게 하고자 합니다.
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
[114] # Python 라이브러리를 이용하기 위해 라이브러리 정의
      import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      import scipy.stats as ss
      from sklearn import metrics
      from sklearn.model_selection import train_test_split

      # Dataset Load
      df = pd.read_csv("https://raw.githubusercontent.com/yoonkt200/FastCampusDataset/master/BostonHousing2.csv")

      # Target 변수인 CMEDV를 기준으로 Data 분석을 하기 위해 CMEDV column의 순서를 바꿉니다.
      df = df[["TOWN", "CMEDV", "LON", "LAT", "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"]]
```

Regression Model 구현을 위한 접근 방법

- Dataframe을 확인했을 때, Dataframe이 506 rows x 17 columns로 이루어져 있는 것을 확인할 수 있었습니다.
- 과제에서 506개의 ID x 17개의 가격 요소 변수인 8602개의 Data라는 조건이 명시되어 있었기 때문에, Dataset을 정상적으로 Load한 것을 확인할 수 있었습니다.

```
# Dataframe 확인
df
```

	TOWN	CMEDV	LON	LAT	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	Nahant	24.0	-70.9550	42.2550	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	Swampscott	21.6	-70.9500	42.2875	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	Swampscott	34.7	-70.9360	42.2830	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	Marblehead	33.4	-70.9280	42.2930	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	Marblehead	36.2	-70.9220	42.2980	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	Winthrop	22.4	-70.9860	42.2312	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
502	Winthrop	20.6	-70.9910	42.2275	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	Winthrop	23.9	-70.9948	42.2260	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	Winthrop	22.0	-70.9875	42.2240	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	Winthrop	19.0	-70.9825	42.2210	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

506 rows × 17 columns

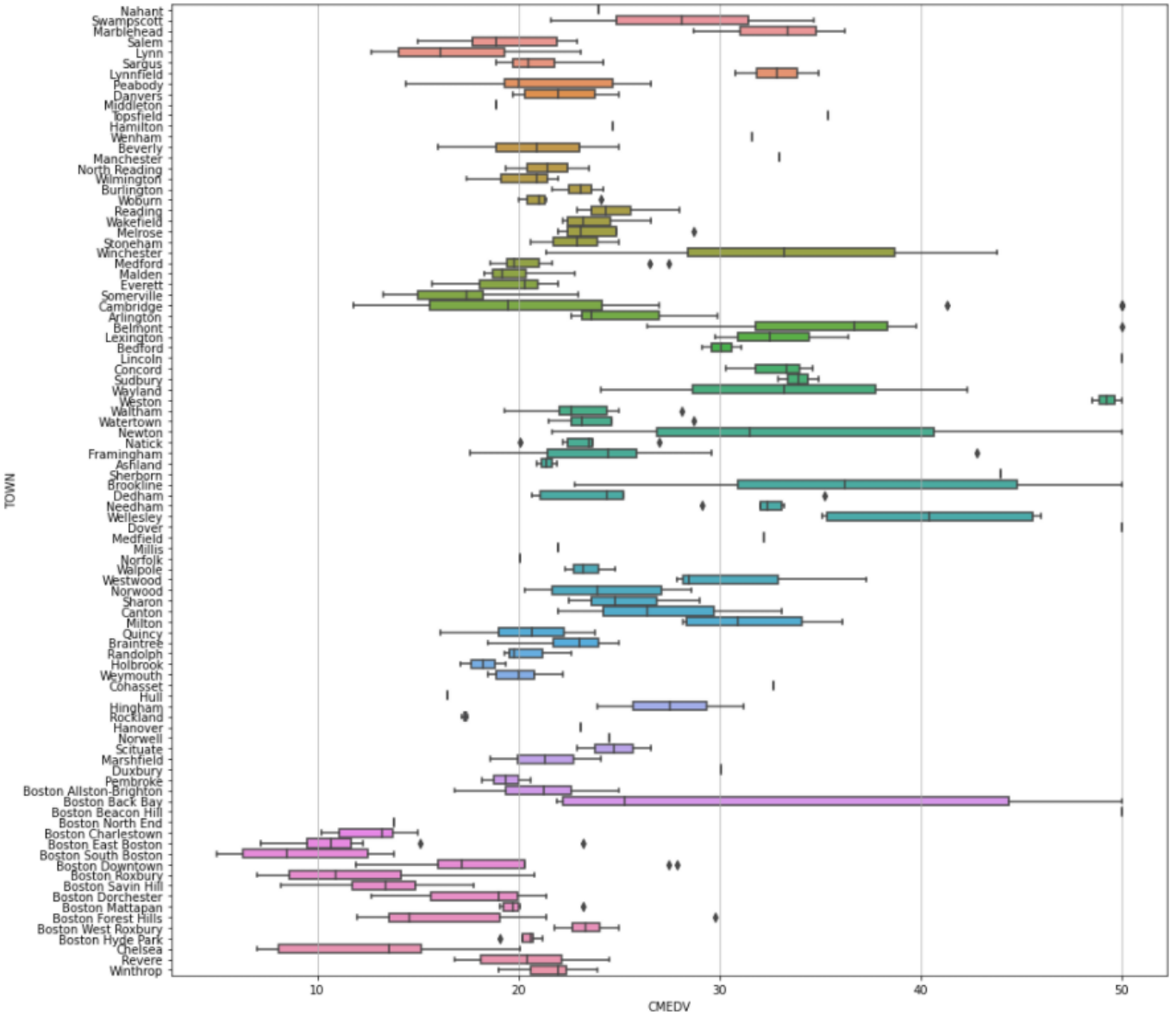
Dataset 분석(TOWN - 1)

- Dataframe을 통해 Dataset의 변수에 따른 Data를 확인했을 때, ‘TOWN’을 제외한 나머지 변수는 float나 정수형의 숫자로 구성되어 있는 것을 확인할 수 있었습니다.
- 이에 따라 ‘TOWN’과 나머지 변수의 분석 방법을 나누어 ‘TOWN’ 변수에서는 ‘TOWN’ 종류 별 개수 Counting, ‘TOWN’ 종류에 따른 ‘CMEDV’ 값의 분포를 확인하는 방법으로 Data를 분석하기로 하였습니다.
- Value_count 함수를 이용하여 Town의 종류와 종류 별 개수를 확인했을 때, 총 92종의 Town이 각각 1~30개까지 다양하게 분포되어 있는 것을 확인할 수 있었습니다.
- Seaborn 라이브러리에서 Boxplot 함수를 이용하여 Town 종류에 따른 CMEDV 값의 분포를 확인했을 때, 상대적으로 Boston 지역에 속하는 Town의 CMEDV 값이 낮은 경향을 보이고 있는 것을 확인할 수 있었습니다.

```
[4]  # Dataset의 Town의 종류가 몇 종류인지 value_count 함수를 이용하여 확인합니다.
     df['TOWN'].value_counts(ascending = True)
```

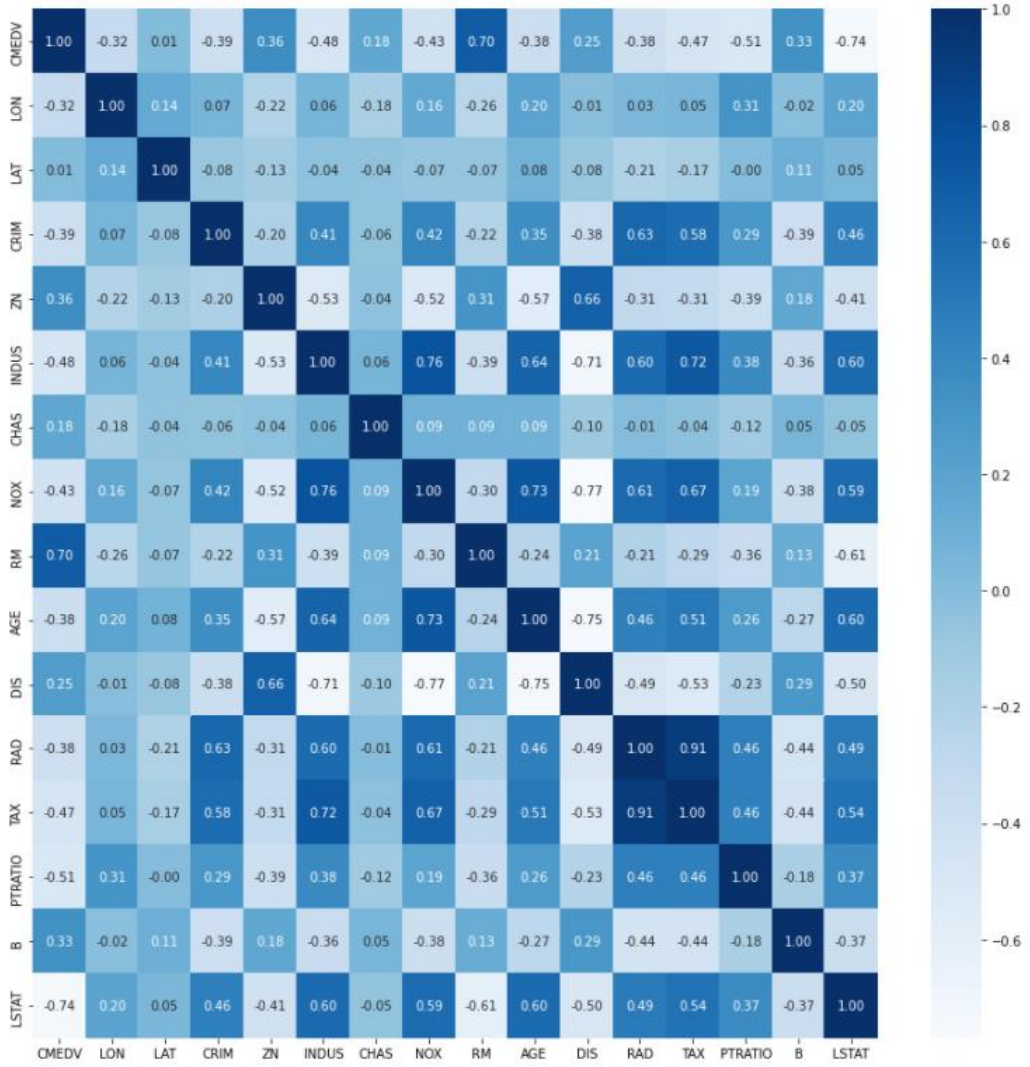
```
Norfolk              1
Hull                 1
Hanover              1
Nahant               1
Sherborn             1
                    ..
Newton              18
Boston Roxbury      19
Lynn                22
Boston Savin Hill   23
Cambridge           30
Name: TOWN, Length: 92, dtype: int64
```

Dataset 분석(TOWN - 2)



Dataset 분석(Heatmap - 1)

- 데이터 요소에 대한 가정을 세우기 위해, 우선 Dataframe 내 변수의 Correlation 값을 구하여 Heatmap을 구현하는 과정을 거쳤습니다.

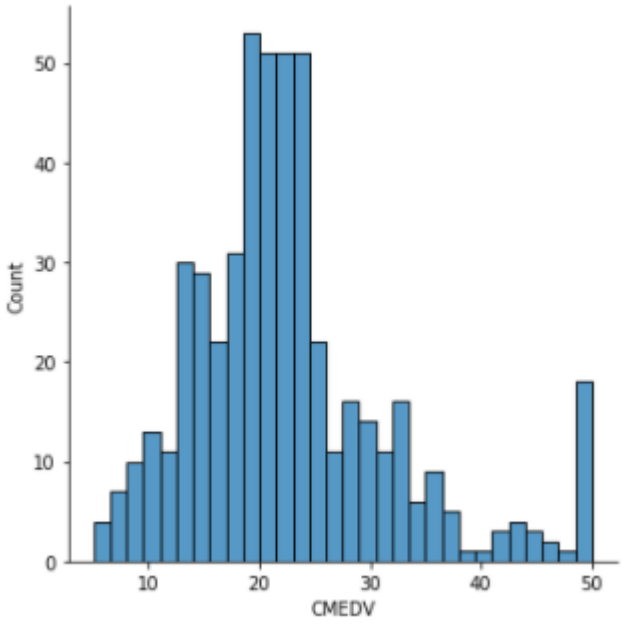


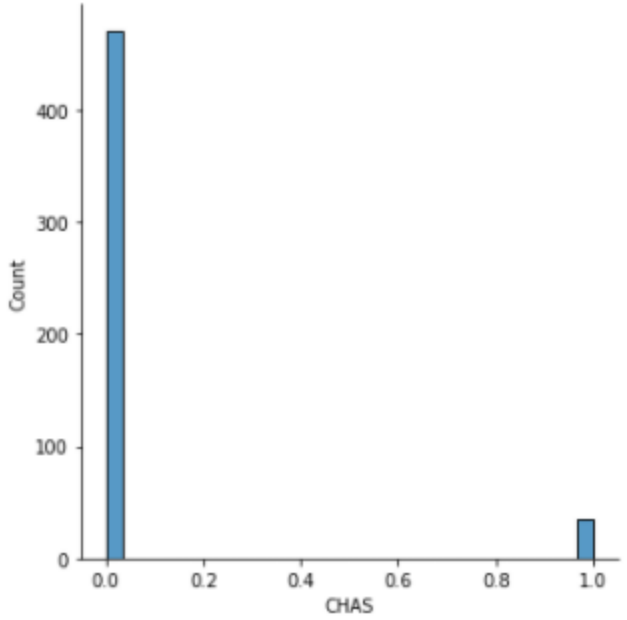
Dataset 분석(Heatmap - 2)

- Heatmap에서 CMEDV와 다른 변수 간의 상관 관계를 관찰했을 때, 상대적으로 높은 상관 관계를 보인 특성은 다음과 같습니다.
  - **RM(해당 지역의 자택 당 평균 방 개수) : 0.7**
  - **LSTAT(해당 지역의 빈곤층 비율) : -0.74**
- 2개의 특성을 제외한 나머지 특성은 Correlation 값이 +- 0.51 이하로 낮은 상관 관계를 보이고 있습니다.
- 따라서 Heatmap Data 분석을 통해 2개의 가정을 세울 수 있었습니다.
  - **해당 지역의 자택 당 평균 방 개수(RM)이 높을 수록 주택 가격은 높을 것이다.**
  - **해당 지역의 빈곤층 비율(LSTAT)이 높을 수록 주택 가격은 낮아질 것이다.**
- 다음 2개의 가정은 이후 Model Training 과정을 거쳐 Test Dataset의 주택 가격을 Prediction한 결과의 Dataset과 분석하여 증명하기로 하였습니다.

Dataset 분석(Data 분포 막대 그래프)

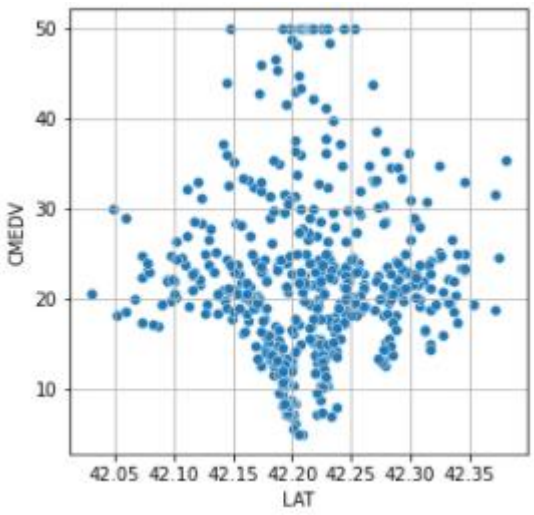
- Seaborn 라이브러리에서 Displot 함수를 이용하여 각 특성 별로 분포 그래프를 출력하였습니다. 코드 내에서는 모든 특성에 대해 분포 그래프를 출력하여 확인하였지만, 프로젝트 보고서에는 분량 관계로 인해 일부 특성의 그래프만 첨부하였습니다.
- 그래프는 각 특성 별 Data scale을 30개의 일정한 구간으로 나누어 출력하였습니다.
- Target 변수인 CMEDV 값에 따른 분포 그래프와, 해당 지역이 Charles 강과 접하고 있는지에 대한 여부를 나타내는 특성인 CHAS 값에 따른 분포 그래프입니다. 그래프에서 확인할 수 있듯이, CHAS는 dummy variable이기 때문에 0과 1로만 Data가 표시되기 때문에, 양 구간에서만 그래프 분포가 나타나는 것을 확인할 수 있습니다.

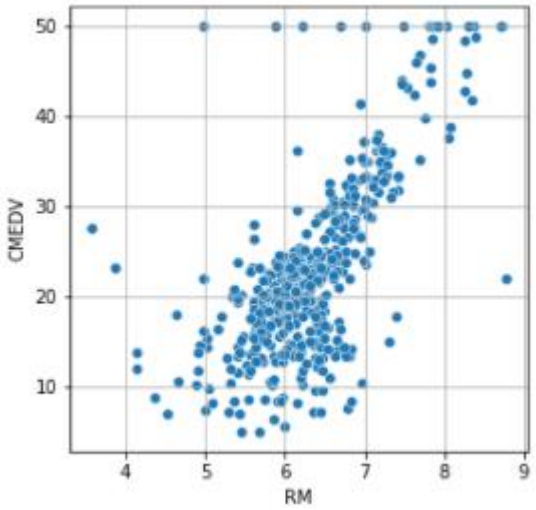


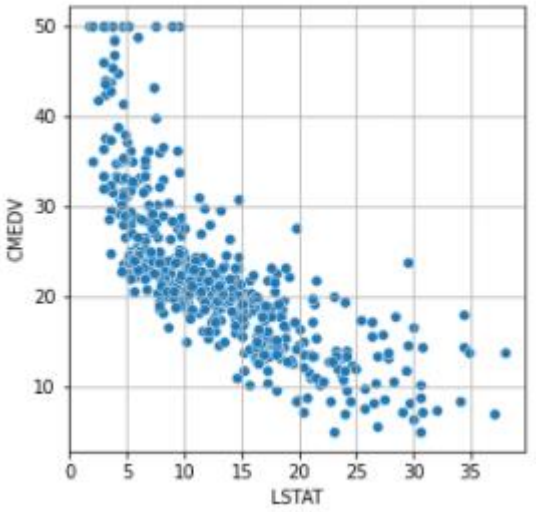


Dataset 분석(CMEDV에 대한 Scatter Plot)

- Seaborn 라이브러리에서 scatterplot 함수를 이용하여 각 특성 별로 CMEDV에 대한 상관 관계를 직관적으로 알아보기 위하여 Scatter Plot을 구현하였습니다. 코드 내에서는 모든 특성에 대해 분포 그래프를 출력하여 확인하였지만, 프로젝트 보고서에는 분량 관계로 인해 일부 특성의 그래프만 첨부하였습니다.
- Correlation 값이 0.01로 CMEDV와 상관 관계가 거의 존재하지 않는 특성인 'LAT'은 그래프에서 확인할 수 있듯이 데이터 분포가 일정한 규칙 없이 산개해 있는 것을 확인할 수 있습니다.
- 반면, Correlation 값이 0.7, -0.74로 CMEDV와 밀접한 상관 관계를 가지고 있는 특성인 'RM'과 'LSTAT'은 그래프에서 각각 CMEDV 값과 비례 / 반비례한 특성을 가지는 Data 분포가 나타나는 것을 확인할 수 있습니다.







Dataset Preprocessing

- String 형태의 Data인 'TOWN'과 Dummy Data인 'CHAS'를 제외한 각 특성의 데이터의 Scale이 다양하기에 상대적 크기로 인해 학습 결과가 제대로 도출되지 않을 가능성이 존재하기에 각 특성 별로 Rescaling을 진행해 데이터의 상대적인 크기 차이를 제거해야 합니다.
- Rescaling 중 하나의 방법인 Normalization을 이용하여 Dataset에서 Target 특성인 'CMEDV'와 'TOWN', 'CHAS'를 제외한 나머지 특성을 Normalization하는 preprocessing을 진행했습니다.
- 또한 'TOWN' 특성의 경우 String Data이므로, 학습을 진행하기에는 적절하지 않은 특성이라 판단하여 Training / Test Dataset에서 'TOWN' 특성을 제거하는 preprocessing 또한 진행하였습니다.

```
# Dataframe 내 변수의 scale이 다양하기에 학습 최적화를 위하여 Normalization을 통해 변수의 scale을 맞춰줍니다.
standardzation = ['LON', 'LAT', 'CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
df[standardzation] = ss.zscore(df[standardzation])
dataset = df[standardzation]
# CHAS의 경우, 0 / 1로 표현되는 Dummy variable이기에 Normalization을 진행하지 않습니다.
dataset['CHAS'] = df['CHAS']
dataset
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

	LON	LAT	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CHAS
0	1.345913	0.624791	-0.419782	0.284830	-1.287909	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666608	-1.459000	0.441052	-1.075562	0
1	1.412287	1.151396	-0.417339	-0.487722	-0.593381	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987329	-0.303094	0.441052	-0.492439	0
2	1.598134	1.078482	-0.417342	-0.487722	-0.593381	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987329	-0.303094	0.396427	-1.208727	0
3	1.704333	1.240514	-0.416750	-0.487722	-1.306878	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106115	0.113032	0.416163	-1.361517	0
4	1.783981	1.321530	-0.412482	-0.487722	-1.306878	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106115	0.113032	0.441052	-1.026501	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.934395	0.239154	-0.413229	-0.487722	0.115738	0.158124	0.439316	0.018673	-0.625796	-0.982843	-0.803212	1.176466	0.387217	-0.418147	0
502	0.868021	0.179203	-0.415249	-0.487722	0.115738	0.158124	-0.234548	0.288933	-0.716639	-0.982843	-0.803212	1.176466	0.441052	-0.500850	0
503	0.817577	0.154898	-0.413447	-0.487722	0.115738	0.158124	0.984960	0.797449	-0.773684	-0.982843	-0.803212	1.176466	0.441052	-0.983048	0
504	0.914483	0.122491	-0.407764	-0.487722	0.115738	0.158124	0.725672	0.736996	-0.668437	-0.982843	-0.803212	1.176466	0.403225	-0.865302	0
505	0.980857	0.073882	-0.415000	-0.487722	0.115738	0.158124	-0.362767	0.434732	-0.613246	-0.982843	-0.803212	1.176466	0.441052	-0.669058	0

506 rows × 15 columns

Regression Model 구현 과정(Gradient Descent - 1)

- Linear Regression 알고리즘으로 구현할 수 있는 대표적인 방법은 Gradient Descent Algorithm, Normal Equation 2가지가 있습니다.
- 따라서 Gradient Descent, Normal Equation Method를 이용한 2개의 Model을 구현하여 Dataset을 학습하기로 결정하였습니다.
- Gradient Descent
  - Linear Regression으로 Dataset을 구하기 위해서는 “Data를 표현할 수 있는 직선이 존재한다”는 Hypothesis를 세울 수 있으며, 이를 표현하는 함수는 다음과 같이 나타낼 수 있습니다.

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

  - 여기서 Y는 Target 변수(‘CMEDV’), θ_0은 Bias, θ_n은 Regression Coefficient 혹은 Weight, x_n은 독립 변수(‘RM’, ‘LSTAT’ 등)을 의미합니다.
  - Linear Regression에서는 Cost(Loss) function을 이용하여 Θ 값이 Training Data를 잘 표현하는 지 확인할 수 있습니다. Cost function은 다음과 같이 나타낼 수 있습니다.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^i) - y^i)^2$$

  - Gradient Descent 알고리즘은 Cost function을 각 Θ 값에 대해 편미분을 진행한 후, 이 값에 Learning rate를 곱하여, 기존 Θ 값에서 빼는 방식으로 알고리즘이 진행되며 이는 다음과 같은 식으로 나타낼 수 있습니다.

$$\theta_0 = \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h(x^i) - y^i) \qquad \theta_1 = \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}(h(x^i) - y^i)x_1^i$$

  - Gradient Descent 알고리즘을 통해 구한 Θ 값을 Test Dataset의 각 변수에 맞게 곱하면 Y 값을 예측할 수 있습니다.

Regression Model 구현 과정(Gradient Descent - 2)

- Gradient Descent Algorithm을 구현하기 위해 우선, Cost function 식을 이용하여 Cost function을 정의하는 코드를 기본 라이브러리인 numpy만을 이용하여 구현했습니다.

```
# Gradient Descent Regression Model 구현을 위해 Cost Function을 계산합니다.
def cost_function(X_train, Y_train, b):
  # b = regression coefficient
  # m = Number of Training Datas
  m = 404
  predictions = X_train.dot(b)
  errors = np.subtract(predictions, Y_train)
  J = 1 / (2 * m) * errors.T.dot(errors)
  return J
```

- Cost function을 이용하여 Gradient Descent 과정을 통해 각 Θ 값을 구할 수 있도록, Gradient Descent 알고리즘을 구현하는 코드를 이어서 기본 라이브러리인 numpy만을 이용하여 구현했습니다.

```
# Regression Model 구현을 위해 Gradient Descent Algorithm function을 구현합니다.
def gradient_descent(X_train, Y_train, b, lr, iterations):
  # lr = learning rate
  m = 404
  cost = np.zeros(iterations)
  for i in range(iterations):
    predictions = X_train.dot(b)
    errors = np.subtract(predictions, Y_train)
    sum_delta = (lr / m) * X_train.transpose().dot(errors)
    b = b - sum_delta
    cost[i] = cost_function(X_train, Y_train, b)

  return b, cost
```

- 이후, Gradient Descent를 통해 구한 Θ 값을 Training Dataset에 이용하여 주택 가격 변수에 맞는 Model Parameter가 되도록 learning rate = 0.1, iterations = 200으로 설정하여 Training하는 과정을 거쳤습니다.

Regression Model 구현 과정(Normal Equation)

- Gradient Descent Algorithm은 최적의 Θ 값을 구하기 위해 어떤 값에 수렴할 때까지 반복해야 하지만, Normal Equation을 이용하면 iteration 없이 한번에 최적의 Θ 값을 구할 수 있다는 특징을 가지고 있습니다. Normal Equation은 다음과 같은 식으로 나타낼 수 있습니다. 여기서 X는 독립 변수, y는 종속 변수에 속합니다.

$$\Theta = (X^T X)^{-1} X^T y$$

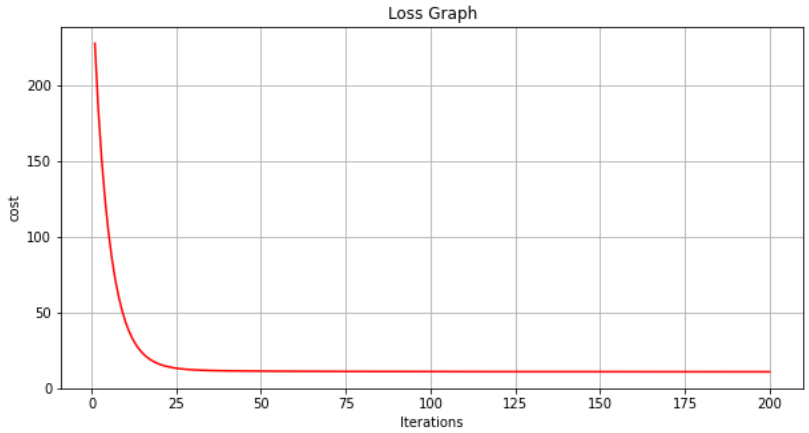
- Normal Equation Algorithm의 경우, Gradient Descent와 달리 Cost Function을 구하지 않고 하나의 식으로 바로 구현할 수 있기에, 기본 라이브러리인 numpy만을 이용하여 Normal Equation을 구현할 수 있었습니다.

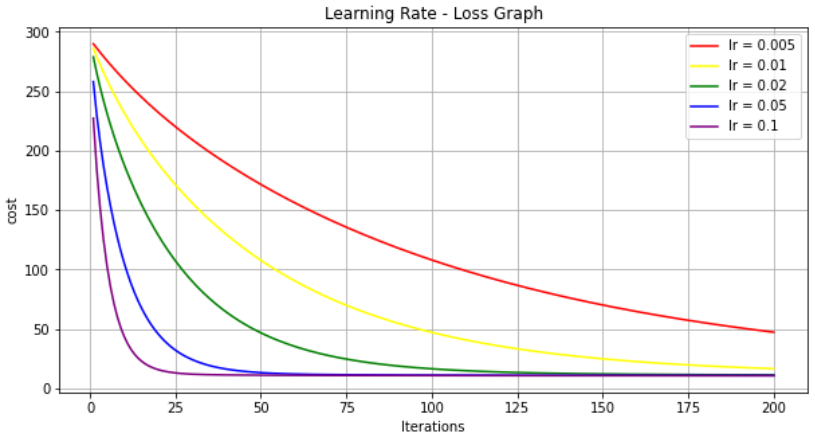
```
# Normal Equation Algorithm 구현
def normal_equation(X_train, Y_train):
  X_train_transpose = np.transpose(X_train)
  temp_1 = np.linalg.inv(X_train_transpose.dot(X_train))
  temp_2 = X_train_transpose.dot(Y_train)
  theta = temp_1.dot(temp_2)

  return theta
```

성능 평가(Gradient Descent - 1)

- Gradient Descent Algorithm에서 최적의 Θ 값을 구하기 위해 iteration과 learning rate를 따로 설정하여, training하는 과정을 거쳤기 때문에, 우선 iteration 및 learning rate의 변화에 따른 loss를 알아보기 위해 iteration / learning rate에 대한 loss graph를 구현했습니다.

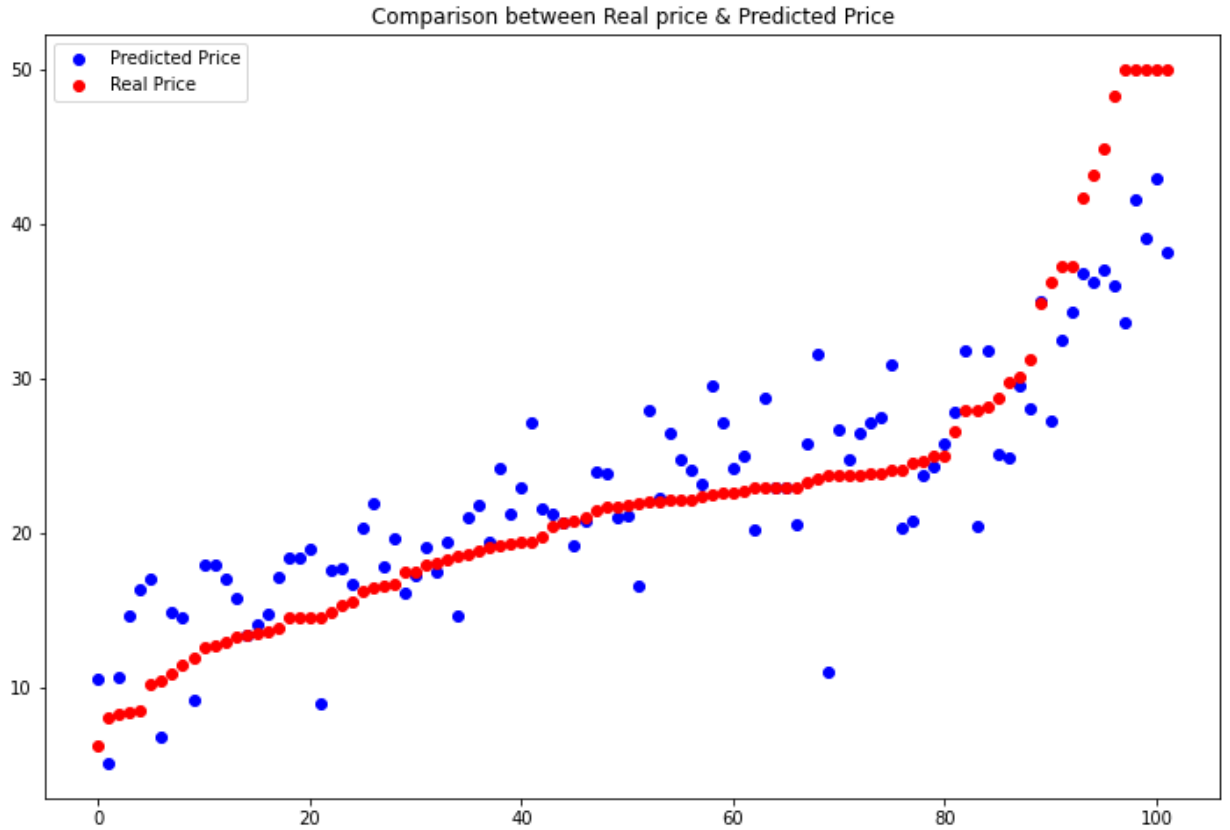




- Graph를 통해 Iteration이 25 이상일 때부터 Loss가 안정화되며, learning rate의 경우, 기존에 설정한 learning rate = 0.1보다 낮아질 경우 Loss 값이 상대적으로 커지는 경향을 확인할 수 있었습니다.
- 이를 통해 최종적으로 Iteration은 200, learning rate는 0.1로 정하였습니다.

성능 평가(Gradient Descent - 2)

- Gradient Descent Model에서 주택 가격을 잘 예측 했는지 확인하기 위해, 실제 주택 가격과 Gradient Descent Algorithm을 통해 예측된 주택 가격을 비교하는 scatter graph를 plot하여 확인했습니다.
- 완벽하게 예측하는 수준은 아니지만, 어느 정도는 Real Price의 Linear Regression에 가깝게 Predicted Price의 Dataset이 형성되어 있는 것을 확인할 수 있었습니다.



성능 평가(Gradient Descent - 3)

- Gradient Descent Model 평가를 위해 Mean Absolute Error(MAE), Root Mean Squared Error(RMSE), R-squared metric 3개의 지표를 이용하여 성능 평가를 진행했습니다.
- MAE : 실제 값과 예측 값의 차이인 오차들의 절댓값 평균을 나타냅니다. 0에 가까울 수록 좋은 성능이라고 할 수 있습니다.
- RMSE : Regression Model의 Cost Function인 MSE에 root를 씌운 값입니다. 0에 가까울 수록 좋은 성능이라고 할 수 있습니다.
- R-Squared : 분산 기반 예측 성능 평가 지표입니다. MAE, RMSE와 같은 지표들과 다르게 상대적인 성능이 어느 정도인지 직관적으로 판단할 수 있다는 특징을 가지고 있습니다. 1에 가까울 수록 좋은 성능이라고 할 수 있습니다. 예를 들어, 0.3이라고 하면 독립변수가 종속변수의 30% 정도를 설명한다고 할 수 있습니다.

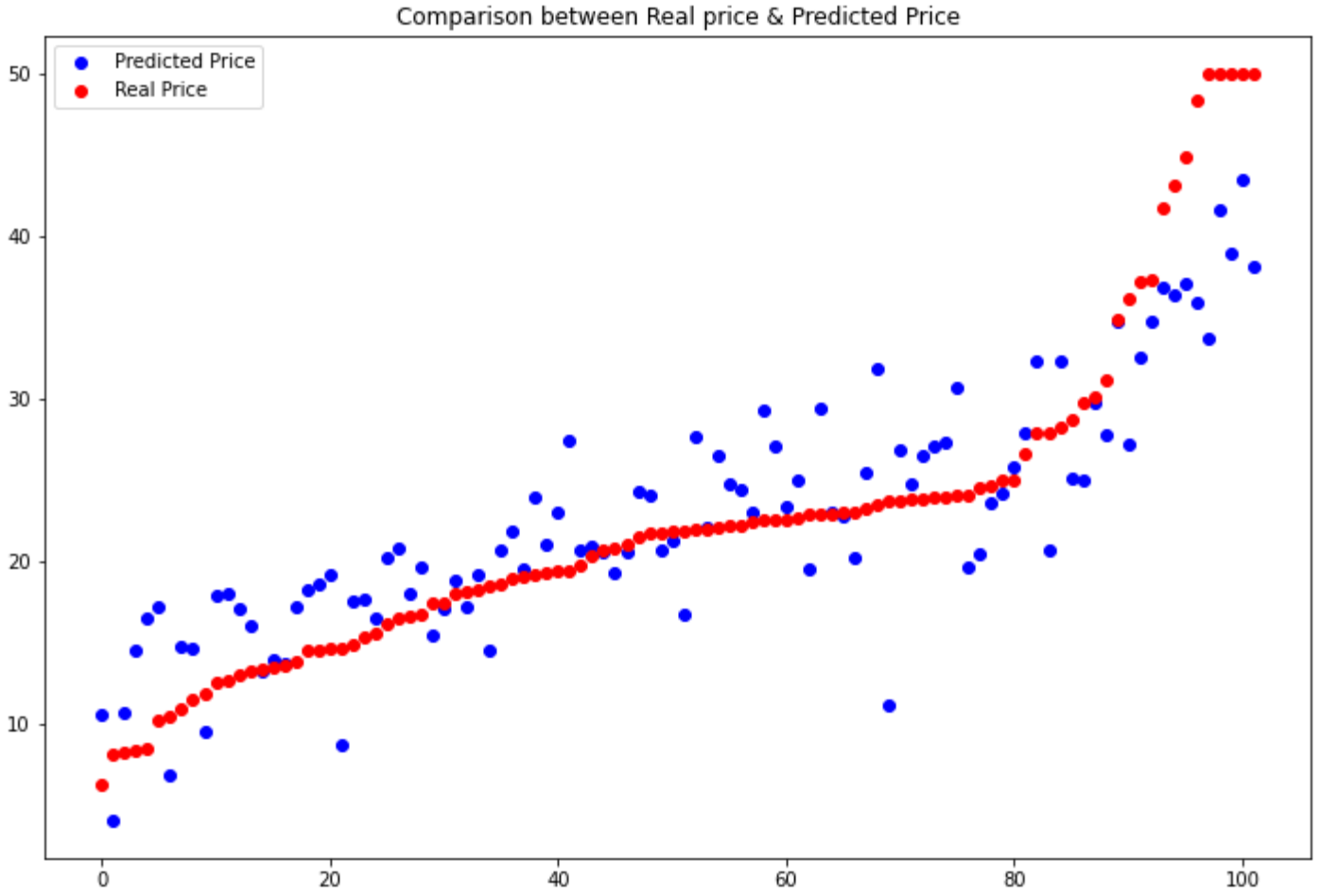
```
# Mean Absolute Error / Root Mean Squared Error / R-Squared metric을 이용하여 성능 평가
# Gradient Descent
MEA = metrics.mean_absolute_error(Y_test_data, grad_price)
print('MEA for the model : {}'.format(MEA))
RMSE = np.sqrt(metrics.mean_squared_error(Y_test_data, grad_price))
print('RMSE for the model : {}'.format(RMSE))
R = metrics.explained_variance_score(Y_test_data, grad_price)
print('R squared of the model : {}'.format(R))
```

```
MEA for the model : 3.7179153415755177
RMSE for the model : 4.817028498748373
R squared of the model : 0.7656236752102229
```

- Gradient Descent Model의 R squared 값은 약 0.765입니다. 독립변수가 종속변수의 76% 정도를 설명한다는 의미이므로, 정확도 면에서 나쁘지 않은 성능을 보일 수 있다고 할 수 있습니다.

성능 평가(Normal Equation - 1)

- Normal Equation Model에서 주택 가격을 잘 예측 했는지 확인하기 위해, 실제 주택 가격과 예측된 주택 가격을 비교하는 scatter graph를 plot하여 확인했습니다.
- Gradient Descent와 마찬가지로 완벽하게 예측하는 수준은 아니지만, 어느 정도는 Real Price의 Linear Regression에 가깝게 Predicted Price의 Dataset이 형성되어 있는 것을 확인할 수 있었습니다.



성능 평가(Normal Equation - 2)

- Normal Equation Model 평가를 위해 Mean Absolute Error(MAE), Root Mean Squared Error(RMSE), R-squared metric 3개의 지표를 이용하여 성능 평가를 진행했습니다.

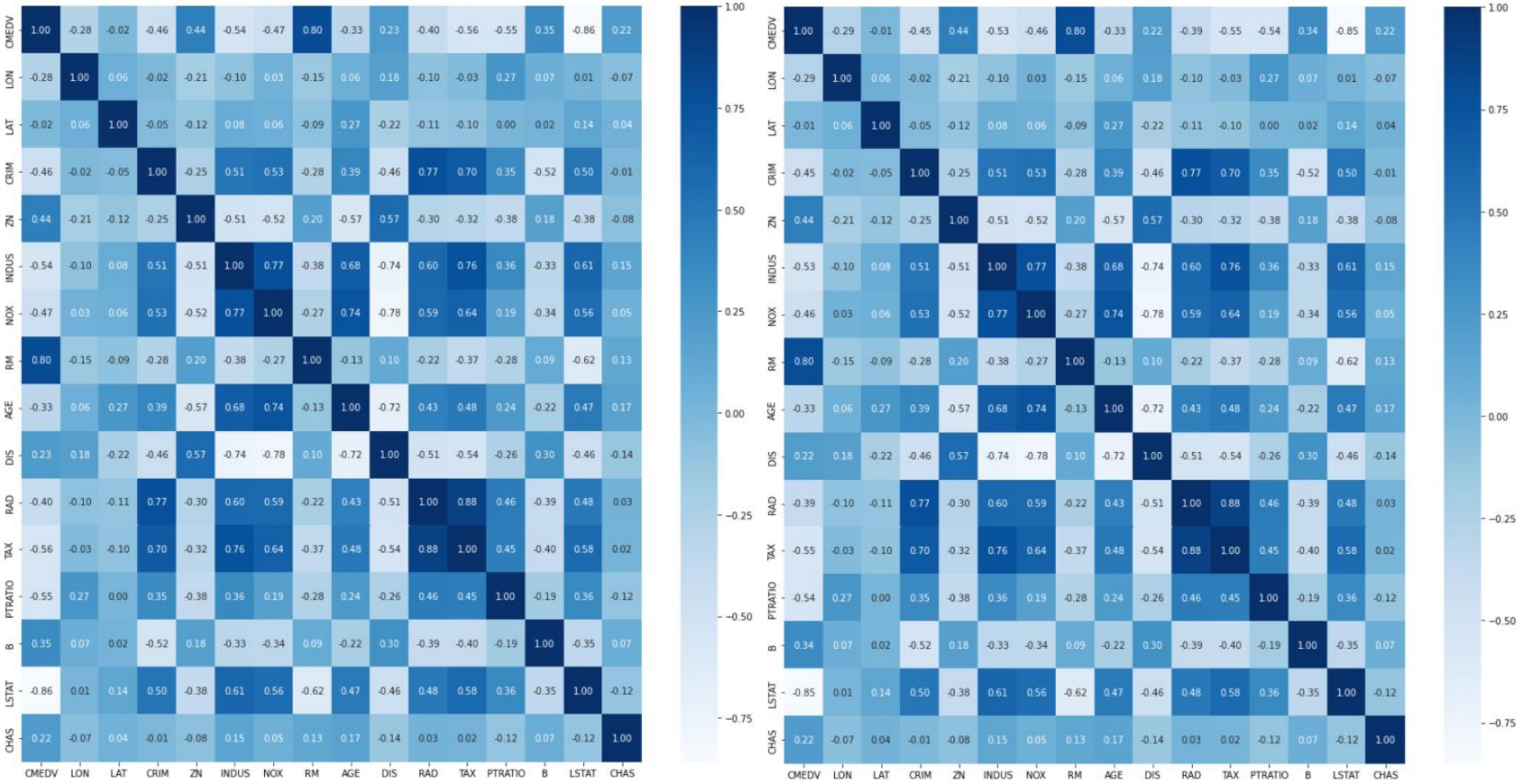
```
# Normal Equation
MEA2 = metrics.mean_absolute_error(Y_test_data, norm_price)
print('MEA for the model : {}'.format(MEA2))
RMSE2 = np.sqrt(metrics.mean_squared_error(Y_test_data, norm_price))
print('RMSE for the model : {}'.format(RMSE2))
R2 = metrics.explained_variance_score(Y_test_data, norm_price)
print('R squared of the model : {}'.format(R2))
```

```
MEA for the model : 3.716796315644998
RMSE for the model : 4.824447376842844
R squared of the model : 0.7646795428800842
```

- Normal Equation의 경우, Gradient Descent Model의 성능 지표와 비교했을 때 MEA, RMSE 값은 더 높고, R squared 값은 더 낮은 것을 확인할 수 있습니다.
- MEA, RMSE는 1에 가까울 수록, R squared는 0에 가까울 수록 좋은 성능의 Model이기에 Normal Equation Model의 성능은 Gradient Descent Model에 비해서 상대적으로 성능이 떨어진다고 볼 수 있습니다.
- Normal Equation Model의 R squared 값은 약 0.764입니다. 독립변수가 종속변수의 76% 정도를 설명한다는 의미이므로, 정확도 면에서 나쁘지 않은 성능을 보일 수 있다고 할 수 있습니다.

가설 증명

- 데이터 분석 단계에서 Correlation과 Heatmap을 통해 정의한 2개의 가설을 증명하기 위해 Test Dataset의 변수와 Gradient Descent Model / Normal Equation을 통해 예측한 CMEDV와의 Correlation을 구하여 Heatmap을 구현했습니다.
- Heatmap에서 확인할 수 있듯이 RM은 0.80, LSTAT은 각각 -0.86, -0.85으로 기존 Dataset에서보다 더 fit하게 높은 상관관계를 보였고, 이를 통해 "**해당 지역의 자택 당 평균 방 개수(RM)이 높을 수록 주택 가격은 높을 것이다.", "해당 지역의 빈곤층 비율(LSTAT)이 높을 수록 주택 가격은 낮아질 것이다.**"의 두 가설이 모두 증명됨을 확인할 수 있었습니다.



Model의 한계

- **Gradient Descent**
  - 최적의 Θ 값을 구하기 위해 learning rate를 설정하여, iteration만큼 반복해야 합니다.
  - 따라서 Model에 적합한 learning rate와 iteration을 직접 찾는 과정이 필요합니다.
  - 만약, 제가 Simulation을 통해 적합한 learning rate와 iteration을 찾는 과정을 거치지 않고 임의로 설정하였다면, 오히려 Normal Equation Model보다 성능 지표가 떨어졌을 것이라고 예상합니다.
  - 많은 반복 계산을 수행하기 때문에 Normal Equation Model에 비해서 속도가 느리다는 한계 또한 가지고 있습니다.
- **Normal Equation**
  - Normal Equation 역시, Dataset의 Feature가 많을 수록 속도가 느리다는 한계를 가지고 있습니다.
  - 제가 진행한 보스턴 주택 Dataset의 경우 Feature가 상대적으로 적은 편에 속하기 때문에 속도 면에서는 Normal Equation이 우월한 편에 속했지만, Dataset의 Feature가 10000 이상일 경우에는 오히려 Gradient Descent가 더 빠른 학습 속도를 가지고 있습니다.
  - Normal Equation의 경우, 역행렬을 이용한 계산을 진행하기에 불필요한 Feature를 가지고 있거나('TOWN'), 데이터 개수에 비해 너무 많은 Feature를 가지고 있을 경우 역행렬이 존재하지 않아 계산이 불가능해지는 경우가 발생하기 때문에 데이터 preprocessing 과정을 통해 불필요한 Feature를 지워야 한다는 한계 역시 존재합니다.

결론

- 프로젝트를 통해 주어진 1970년대 Boston 주택 가격을 Linear Regression Model인 Gradient Descent, Normal Equation 2가지 Model을 직접 numpy 라이브러리만을 이용하여 구현하고, 학습 후 성능을 평가했습니다.

- 두 Model 모두 R squared 성능 지표에서 약 76%의 성능을 보이며, 정확도 면에서 봤을 때 괜찮은 성능을 가지고 있는 것을 확인할 수 있었습니다.