서울시 # 아파트 # 매매가격
예측모델

1팀 일기예보

팀장: 박영범

팀원 : 김예진, 권승현, 오병훈, 최상준

회귀모델을 활용한 서울시 아파트 실거래가 예측 모델 구현

프로젝트기간:2022.01.03-2022.01.07

Work Team & Member

1팀 (일기예보)
팀장 : 박영범
팀원 : 권승현, 김예진, 오병훈, 최상준, 김호준

Work Schedule

01.03 프로젝트 계획안 발표
01.04 데이터 수집 및 가공
01.05 모델 설계
01.06 발표 ppt 작성 및 최종 오류 수정
01.07 프로젝트 발표

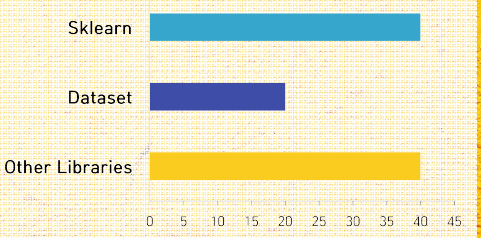
Work Dataset

공공 데이터 포털:
https://www.data.go.kr/
국토교통부 부동산 실거래가 공개 시스템:
http://rtdown.molit.go.kr/

회귀모델 Sklearn 을활용한 서울시 아파트실거래가예측 모델

공공 데이터 포털과 실거래가 공개 시스템에서 수집한 데이터를 바탕으로 Sklearn을 활용한 서울시 실거래가 예측 모델을 설계하여 서울시 아파트 실거래가를 예측함.

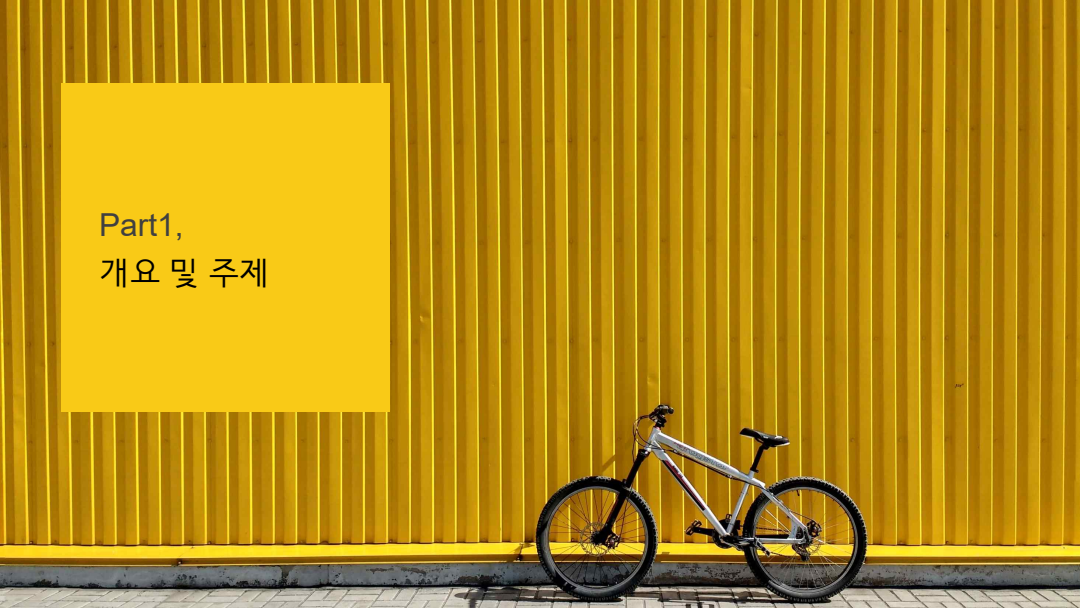
Skills



목차

Part1,

개요 및 주제



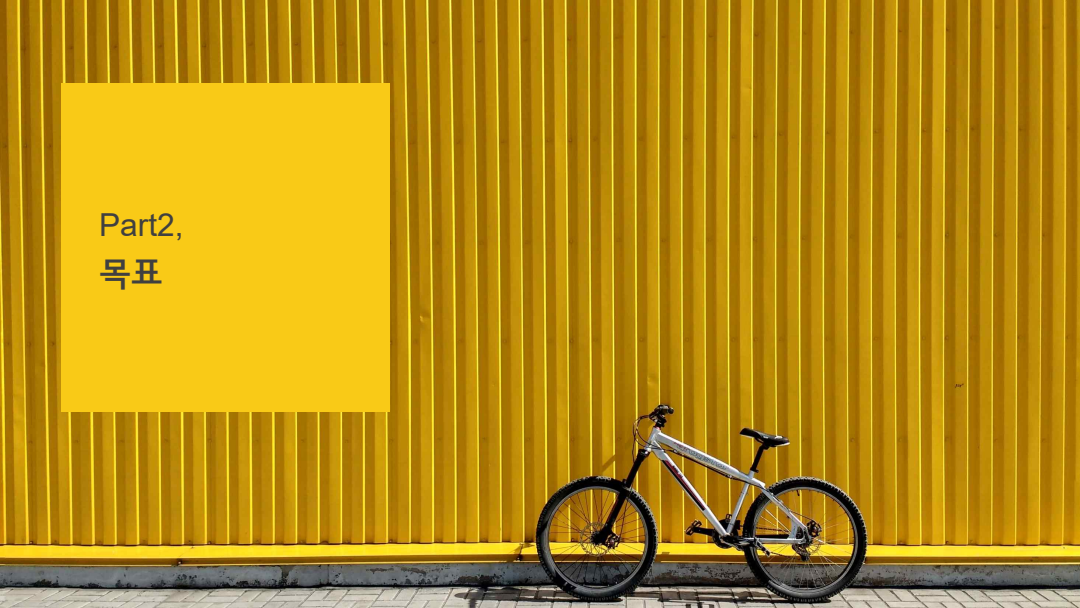
개요 및 주제

개요 및 주제

- 주택가격은 국가 경제와 국민의 삶에 큰 영향을 주며, 급격한 변화시 큰 혼란을 야기할 수 있습니다. 변화를 사전에예측하고 대응하는 것을 관점으로 프로젝트를 기획하게 되었습니다.

Part2,

목표



목표

1. 주도적인 학습을 통한 딥러닝 숙지
2. 미니 프로젝트를 통한 팀원과의 협업 활동
3. 프로젝트 활동 부족한 지식 학습

Part3,

데이터 및 변수

데이터 및 변수

변수

동행지수
지가변동률
생산자물가지수
소비자물가지수
아파트거래현황
종합주가지수
주택건설실적
다우존스산업지수
나스닥종합지수

자료수집 : 구글링
코딩 프로그램 : 코랩

Part4,

문제점 발견 및 해결

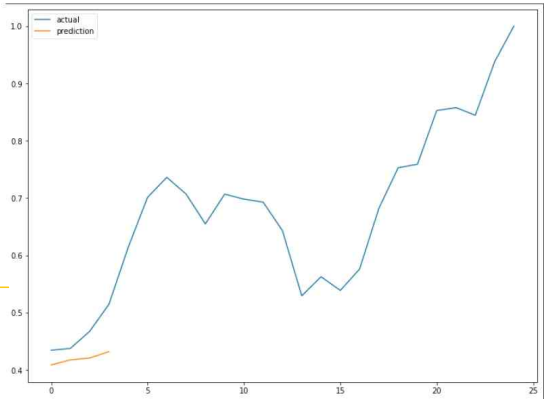
문제의 시작

시세예측 데이터

시세 예측 데이터가 딥러닝으로 돌리기에 너무 나도 적은 양이었기에 제대로 학습을 하지 못하는 문제점 발견

해결방안

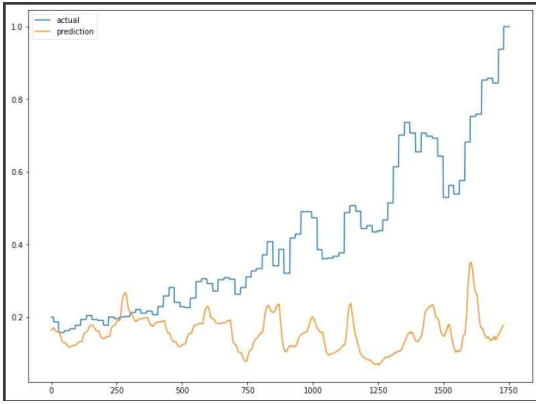
월별 데이터 였던 양을 => 인위적으로 일별 데이터로 늘림



또 다른 문제

시세예측 데이터

데이터 양을 인위적으로 늘리니 예측이 맞지 않는 상황 여러번 반복적인 학습을 진행하였으나 나아가는 기미가 보이지 않음

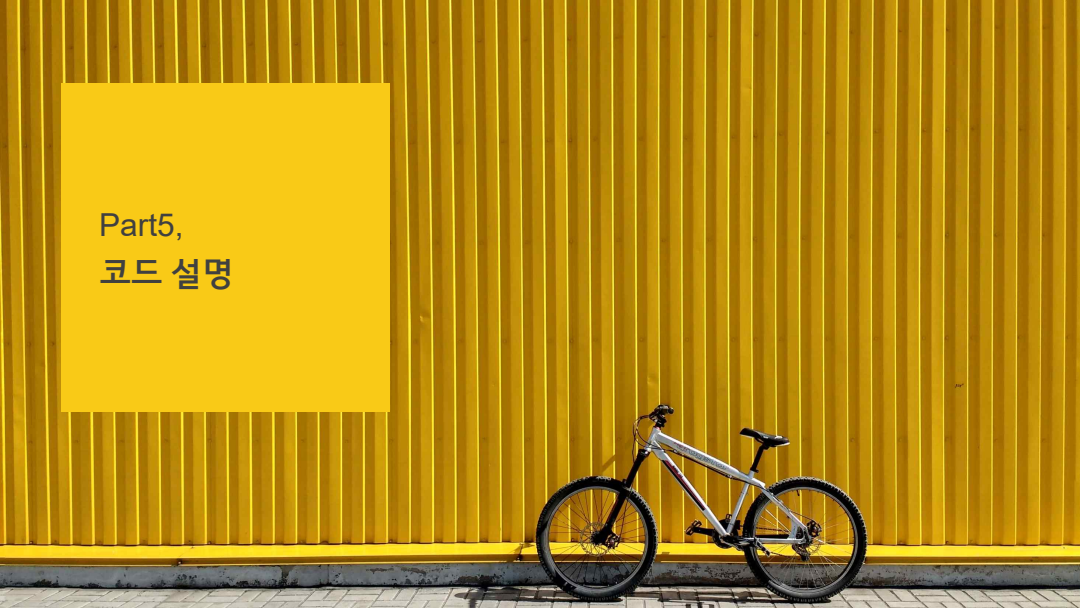


해결방안

데이터 양을 인위적으로 늘리는 건 예측이 어렵단 걸 판단
lstm 방식 => 회귀모델 사이킷런 으로 재 코딩

Part5,

코드 설명



```
[104]  1 import pandas as pd
       2 import numpy as np
       3 import matplotlib.pyplot as plt
       4 import seaborn as sns
       5 import warnings
       6 import os
       7 from tensorflow import keras
       8
       9 %matplotlib inline
      10 warnings.filterwarnings('ignore')
      11
      12 plt.rcParams['font.family'] = 'NanumGothic'
```

```
  1 cd /content/drive/MyDrive/코랩으로 한 번 해보자/22-01-05 프로젝트/입력 변수
```

```
[106]  1 df = pd.read_csv('데이터 입출력 포함.csv', encoding = 'CP949')
```

```
  1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168 entries, 0 to 167
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   날짜                  168 non-null    int64
 1   실거래가                168 non-null    float64
 2   아파트거래현황             168 non-null    int64
 3   다우존스                168 non-null    object
 4   동행종합지수 (2015=100)  168 non-null    float64
 5   생산자물가               168 non-null    float64
 6   나스닥                 168 non-null    object
 7   소비자물가                168 non-null    float64
 8   지난변동률                168 non-null    float64
 9   코스피                 168 non-null    object
dtypes: float64(5), int64(2), object(3)
memory usage: 13.2+ KB
```

필요한 라이브러리 임포트 해주고 df변수명 지정하여 파일입력

df.info() 함수사용하여 Dtype 확인하여 float 변경할 목록을 확인

```
[108]  1 import locale
       2
       3 locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')
       4
       5 df['다우존스'] = df['다우존스'].apply(lambda x : float(x.split()[0].replace(',','')))
       6 df['나스닥'] = df['나스닥'].apply(lambda x : float(x.split()[0].replace(',','')))
       7 df['코스피'] = df['코스피'].apply(lambda x : float(x.split()[0].replace(',','')))
       8 # 천 단위 넘어가서 1,000 이런식으로 나오는 건 문자열 처리로 되서 이 함수로 float 형식으로 바꿔준다
```

```
[109]  1 df = df.astype(float)
```

```
[110]  1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168 entries, 0 to 167
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   날짜                 168 non-null    float64
 1   실거래가               168 non-null    float64
 2   아파트거래현황            168 non-null    float64
 3   다우존스               168 non-null    float64
 4   동행종합지수 (2015=100)  168 non-null    float64
 5   생산자물가              168 non-null    float64
 6   나스닥                168 non-null    float64
 7   소비자물가              168 non-null    float64
 8   지가변동률              168 non-null    float64
 9   코스피                168 non-null    float64
dtypes: float64(10)
```

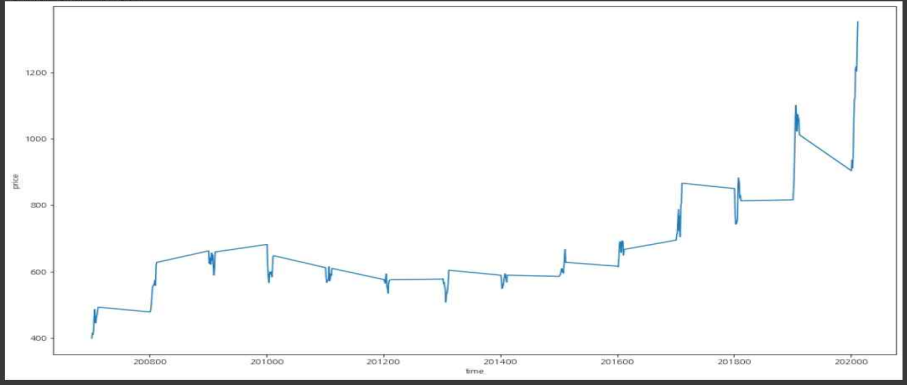
사진에 보이는 **df** 컬럼에 변수들이 1000단위가 넘어가서 자동으로 , 찍혀 **object Dtype** 인식이 되어 **float** 변경이 불가능 오류

locale 라이브러리를 사용하여 ',' 부분을 공백으로 바꾸어서 **float** 변경하고 **info()** 제대로 변경되었는지 확인

현재 데이터를 기반으로 실거래가를 시각화 그래프로 표현

```
[111]  1 plt.figure(figsize=(16, 9))
       2 sns.lineplot(y=df['실거래가'], x=df['날짜'])
       3 plt.xlabel('time')
       4 plt.ylabel('price')
       5
       6 # 시각화 해서 이제까지 실거래가 시각화
```

```
Text(0, 0.5, 'price')
```



사이킷런을 이용하여 전처리 동시에 x(변수값) 값, y 값을 지정 값의 수치를 알기쉽게 0과 1 사이로 값을 맞쳐주는 MinmaxScaler 정규화 활용

[112]
```
1 from sklearn.model_selection import train_test_split
```

[113]
```
1 x_train, x_test, y_train, y_test = train_test_split(df.drop('실거래가', 1), df['실거래가'])
```

[114]
```
1 x_train.shape, x_test.shape
```

```
((126, 9), (42, 9))
```

[14]
```
1 x_train.head()
```

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 scaler = MinMaxScaler()
4 # 스케일을 적용할 column을 정의합니다.
5 scale_cols = ['날짜', '실거래가', '아파트거래현황', '다우존스', '동행종합지수 (2015=100)', '생산자물가', '나스닥', '소비자물가', '지가변동률', '코스피']
6 # 스케일 후 columns
7 scaled = scaler.fit_transform(df[scale_cols])
8 scaled
9
10 # 스케일이 완료
```

[76]
```
1 df2 = pd.DataFrame(scaled, columns = scale_cols)
```

[77]
```
1 from sklearn.model_selection import train_test_split
```

[78]
```
1 x_train, x_test, y_train, y_test = train_test_split(df2.drop('실거래가', 1), df2['실거래가'])
```

[79]
```
1 x_train.shape, x_test.shape
```

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns

my_predictions = {}

colors = ['r', 'c', 'm', 'y', 'k', 'khaki', 'teal', 'orchid', 'sandybrown',
          'greenyellow', 'dodgerblue', 'deepskyblue', 'rosybrown', 'firebrick',
          'deeppink', 'crimson', 'salmon', 'darkred', 'olivedrab', 'olive',
          'forestgreen', 'royalblue', 'indigo', 'navy', 'mediumpurple', 'chocolate',
          'gold', 'darkorange', 'seagreen', 'turquoise', 'steelblue', 'slategray',
          'peru', 'midnightblue', 'slateblue', 'dimgray', 'cadetblue', 'tomato'
         ]

def plot_predictions(name_, pred, actual):
    df = pd.DataFrame({'prediction': pred, 'actual': y_test})
    df = df.sort_values(by='actual').reset_index(drop=True)

    plt.figure(figsize=(12, 9))
    plt.scatter(df.index, df['prediction'], marker='x', color='r')
    plt.scatter(df.index, df['actual'], alpha=0.7, marker='o', color='black')
    plt.title(name_, fontsize=15)
    plt.legend(['prediction', 'actual'], fontsize=12)
    plt.show()

def mse_eval(name_, pred, actual):
    global predictions
    global colors

    plot_predictions(name_, pred, actual)

    mse = mean_squared_error(pred, actual)
    my_predictions[name_] = mse

    y_value = sorted(my_predictions.items(), key=lambda x: x[1], reverse=True)

```

테디노트 깃허브 시각화 함수, 모델링
코드를 참고하여
그래프를 여러 색상으로 눈에 띄게 코딩

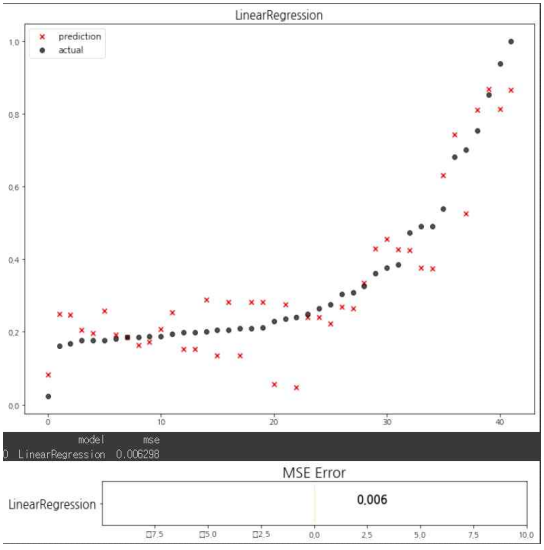
```
36
37      df = pd.DataFrame(y_value, columns=['model', 'mse'])
38      print(df)
39      min_ = df['mse'].min() - 10
40      max_ = df['mse'].max() + 10
41
42      length = len(df)
43
44      plt.figure(figsize=(10, length))
45      ax = plt.subplot()
46      ax.set_yticks(np.arange(len(df)))
47      ax.set_yticklabels(df['model'], fontsize=15)
48      bars = ax.barh(np.arange(len(df)), df['mse'])
49
50      for i, v in enumerate(df['mse']):
51          idx = np.random.choice(len(colors))
52          bars[i].set_color(colors[idx])
53          ax.text(v + 2, i, str(round(v, 3)), color='k', fontsize=15, fontweight='bold')
54
55      plt.title('MSE Error', fontsize=18)
56      plt.xlim(min_, max_)
57
58      plt.show()
59
60  def remove_model(name_):
61      global my_predictions
62      try:
63          del my_predictions[name_]
64      except KeyError:
65          return False
66      return True
```

위에 시각화 함수코드와 이어지는 코드이며
mse에 시각화 그래프와,
위에 코드와 마찬가지로 기초적인 모델링 코드

```
1 from sklearn.linear_model import LinearRegression

[123]  1 model = LinearRegression(n_jobs=-1)

[124]  1 model.fit(x_train, y_train)
       LinearRegression(n_jobs=-1)

[125]  1 pred = model.predict(x_test)

       1 mse_eval('LinearRegression', pred, y_test)
```

시각화 코드를 블러와서 실행시킨 모습

mse error 0.006 준수한 성적을 내고 있다.



규제 (Regularization)

학습이 과대적합 되는 것을 방지하고자 일종의 패널티 부여

alphas = 규제 값 큰 값일수록 규제가 큼

L2 규제 (L2 Regularization) = λ

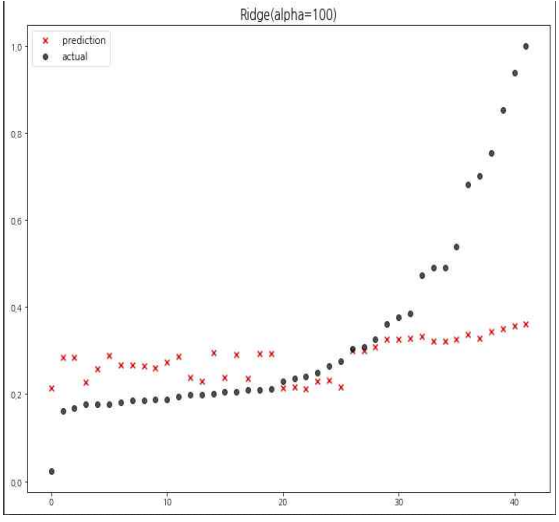
각 가중치 제곱의 합에 규제 강도 λ를 곱한다
λ를 크게 하면 가중치가 더 많이 감소되고 λ를 작게 하면 가중치가 증가

릿지(Ridge) Error=MSE+αw2

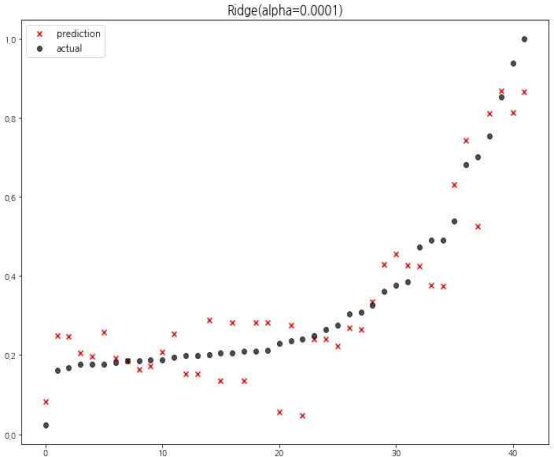
```
1 from sklearn.linear_model import Ridge
```

```
[136] 1 # 값이 커질 수록 큰 규제입니다.
      2 alphas = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]
```

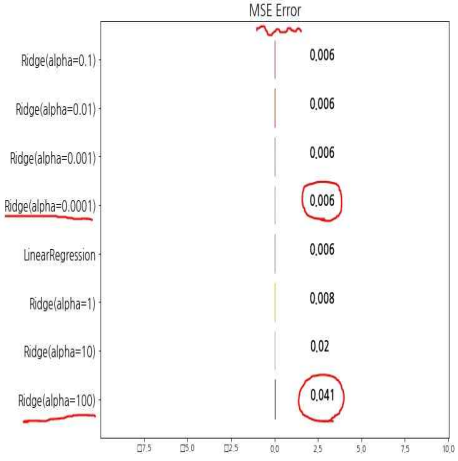
```
1 for alpha in alphas:
2     ridge = Ridge(alpha=alpha)
3     ridge.fit(x_train, y_train)
4     pred = ridge.predict(x_test)
5     mse_eval('Ridge(alpha={})'.format(alpha), pred, y_test)
```



가장 큰 alpha(규제) 값을 주었을 때 본래의 mse 에러보다 더 많은 에러가 났으며, 그래프 또한 많이 떨어진 예측을 보여준다.



최솟값의 규제를 주었더니 향상된 모습의 그래프와 mse 값 또한 원래 수치대로 돌아왔다.
여기서의 최소 영향을 주는 값은 0.1 이상



가장 큰 규제를 주었던 그래프의 mse 에러 값이
가장 큰 걸 확인 할 수 있다.

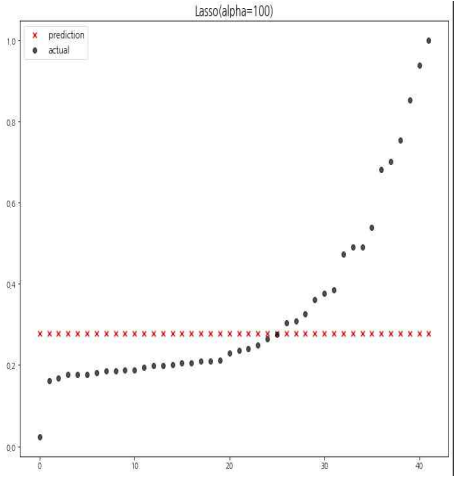
L1 규제 (L1 Regularization)

가중치의 제곱의 합이 아닌 가중치의 합을 더한 값에 규제 강도 λ를 곱하여 오차에 더한다.

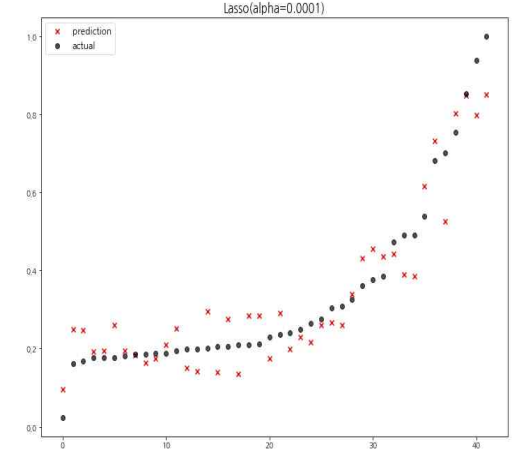
어떤 가중치(w)는 실제로 0이 된다. 즉, 모델에서 완전히 제외되는 특성이 생기
라쏘(Lasso) - L1 규제 Error=MSE+α|w|

```
[145]  1 from sklearn.linear_model import Lasso

[146]  1 # 값이 커질 수록 큰 규제입니다.
       2 alphas = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]

       1 for alpha in alphas:
       2     lasso = Lasso(alpha=alpha)
       3     lasso.fit(x_train, y_train)
       4     pred = lasso.predict(x_test)
       5     mse_eval('Lasso(alpha={})'.format(alpha), pred, y_test)
```

규제를 최대 값을 준 라쏘 그래프



규제를 최소 값을 준 라쏘 그래프



이 또한 위에 릿지(Ridge) 규제 처럼 최솟 값의 규제일 때에 mse 값은 0.005 좋은 결과과 나오며,
최대로 값을 주었을 때 커진 오차 범위를 보인다.

```
[138]  1 def plot_coef(columns, coef):
       2     coef_df = pd.DataFrame(list(zip(columns, coef)))
       3     coef_df.columns=['feature', 'coef']
       4     coef_df = coef_df.sort_values('coef', ascending=False).reset_index(drop=True)
       5
       6     fig, ax = plt.subplots(figsize=(9, 7))
       7     ax.barh(np.arange(len(coef_df)), coef_df['coef'])
       8     idx = np.arange(len(coef_df))
       9     ax.set_yticks(idx)
      10     ax.set_yticklabels(coef_df['feature'])
      11     fig.tight_layout()
      12     plt.show()
```

```
[139]  1 plot_coef(x_train.columns, ridge.coef_)
```

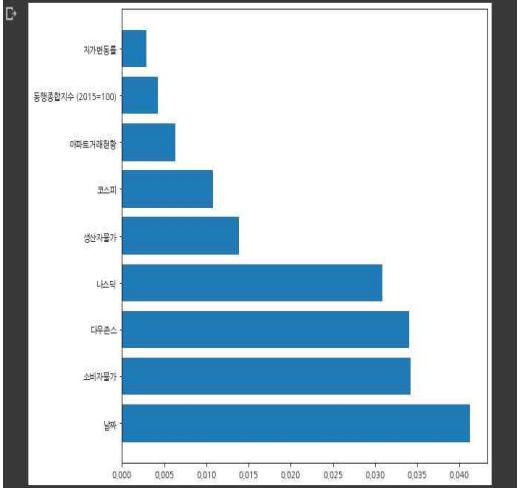
```
[140]  1 ridge_100 = Ridge(alpha=100)
       2 ridge_100.fit(x_train, y_train)
       3 ridge_pred_100 = ridge_100.predict(x_test)
       4
       5 ridge_001 = Ridge(alpha=0.001)
       6 ridge_001.fit(x_train, y_train)
       7 ridge_pred_001 = ridge_001.predict(x_test)
```

```
[141]  1 plot_coef(x_train.columns, ridge_100.coef_)
```

마지막으로 **coef** 함수를 사용하여 계수를 확인하는
시각화 그래프를 호출하는 코드

어떠한 x값(변수)이 가장 y값(거래가) 영향을 끼치는 지 확인 가능

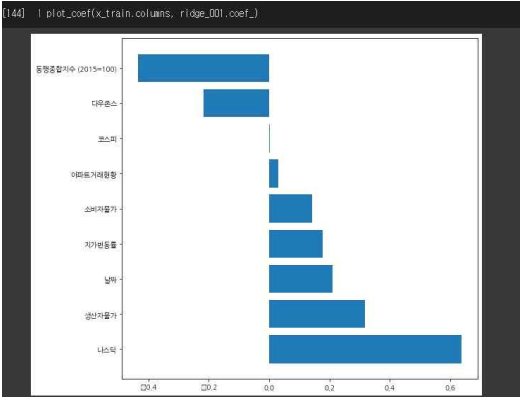
```
plot_coef(x_train.columns, ridge_100.coef_)
```



그래프의 변화를 알아보기 쉽게 하기 위하여 규제 넣어 그래프화

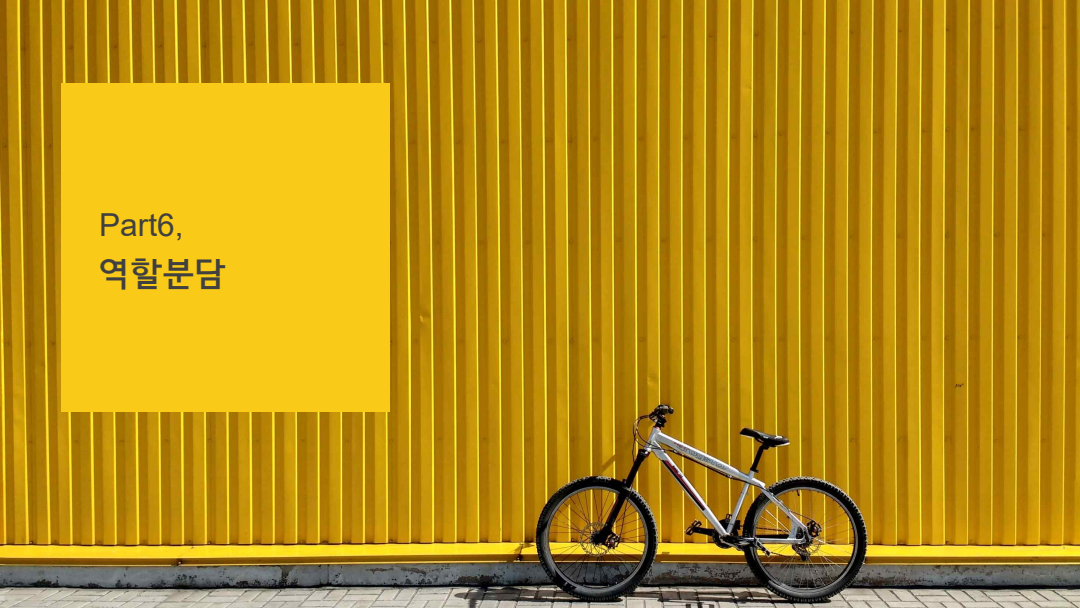
그래프로 알 수 있는점 '날짜' 즉, 시간영향이 값에 가장 큰 영향을 끼친다는 걸 알 수 있다.

```
[144]  plot_coef(x_train.columns, ridge_001.coef_)
```



Part6,

역할분담



역할분담

데이터 수집 및 가공

데이터 모델링

ppt 제작 및 발표

감사합니다