중고차 가격 예측 프로그램

2019112495 유건우

목차

데이터 전처리 - Web Crawling

```python
# 분석할 차량 상세 페이지 url 크롤링
# ------------------------------------------------
# 순서대로 [G80, GV80, G90, 그랜저, 쏘나타, 싼타페, 카니발, 쏘렌토, K7]
car_urls = ["https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=1010&group_no=893&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=1010&group_no=1032&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=1010&group_no=958&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=3&group_no=76&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=3&group_no=69&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=3&group_no=45&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=3&group_no=76&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=3&group_no=69&page=%d",
            "https://www.bobaedream.co.kr/mycar/mycar_list.php?gubun=K&maker_no=3&group_no=45&page=%d"]
cars = []

for car_type in range(len(car_urls)):
    detail_urls = []
    for i in range(1, 20):
        car_url = (car_urls[car_type] % i)
        response = requests.get(car_url)
        soup = BeautifulSoup(response.text, features: "html.parser")
        data = soup.find_all( name: "p", class_="tit")
        # 예외 처리 (차량 정보가 더 이상 없는 경우)
        try:
            temp = data[0]
        except IndexError:
            break

        # 차량 상세 페이지 url 수집
        for j in range(len(data)):
            href = data[j].a['href']
            detail_urls += ["https://www.bobaedream.co.kr" + href]
    # ------------------------------------------------
```

차량 상세페이지 URI 크롤링

9개의 차량 모델을 검색할 수 있는 사이트의 주소를 car_urls 리스트에 저장한다.

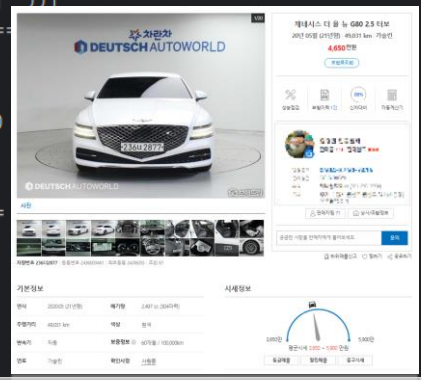
각각의 차량 모델의 모든 중고차를 확인하기 위하여 반복문으로 페이지를 변경하며 'href' 값을 detail_urls 리스트에 저장한다.

데이터 전처리 - Web Crawling

```python
# 차량 상세 페이지로 들어가 상세 정보 크롤링
# ------------------------------------------------
for detail_url in detail_urls:
    response = requests.get(detail_url)
    soup = BeautifulSoup(response.text, features: "html.parser")
    options = soup.find_all( name: "span", class_="radioBox")
    # 예외 처리 (옵션 정보가 없는 차량은 Feature를 만들 수 없어 스킵)
    try:
        check = soup.find_all( name: "div", class_="option-list-container")[0]
    except IndexError:
        print(".", end="")
        continue

    # 차종
    car = [car_type]

    # 가격(단위: 만) / 예외(렌트/리스, 계약 차량 제외)
    data = soup.find_all( name: "b", class_="cr")
    try:
        car += [float(data[0].get_text().replace( _old: ",", _new: ""))]
        if soup.find_all( name: "h4", class_="tit")[1].get_text() ==
            print(".", end="")
            continue
        elif soup.find_all( name: "h4", class_="tit")[1].get_text()
            print(".", end="")
            continue
        elif soup.find( name: "span", class_="price").get_text() ==
            print(".", end="")
            continue
    except IndexError:
        print(".", end="")
        continue
```



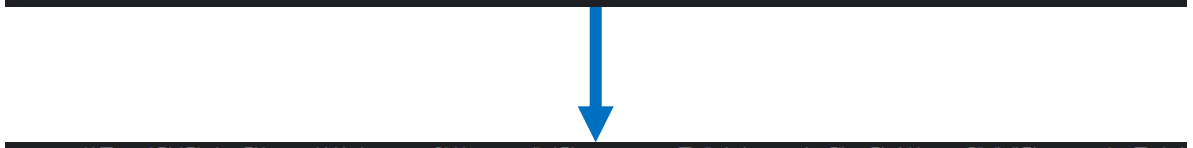
차량 상세 정보 크롤링

detail_urls 리스트에 있는 URI를 하나씩 실행하면서 차량의 상세 정보 크롤링한다.

예외문을 사용하여 가격, 보험 처리 횟수, 연식, 배기량, 주행거리, 연비, 차량 옵션의 정보를 모두 가지고 있는 차량만 크롤링하여 저장할 수 있도록 한다.

데이터 전처리 - Processed Data

	차종	가격(단위: 만)	보험처리	연식	배기량	주행거리	연비	옵션
0	0.0	2370.0	0.0	2018.0	3342.0	86330.0	9.1	52.0
1	0.0	5580.0	0.0	2022.0	2497.0	46600.0	9.8	65.0
2	0.0	4650.0	1.0	2020.0	2497.0	49031.0	10.8	62.0
3	0.0	4880.0	2.0	2021.0	2497.0	39840.0	10.1	61.0
4	0.0	6390.0	0.0	2023.0	3470.0	10304.0	8.4	60.0
..	...	...	...	...	...	...	...	...
517	8.0	930.0	3.0	2016.0	2359.0	159927.0	11.1	29.0
518	8.0	1030.0	8.0	2015.0	2999.0	134109.0	10.4	37.0
519	8.0	620.0	0.0	2012.0	2359.0	141000.0	12.8	38.0
520	8.0	550.0	0.0	2013.0	2359.0	217330.0	11.3	39.0
521	8.0	1100.0	0.0	2017.0	2359.0	140000.0	11.0	33.0



	차종	가격(단위: 만)	보험처리	연식	배기량	...	CD플레이어	AV시스템	뒷좌석TV	텔레매틱스	스마트폰미러
0	0.0	2370.0	0.0	2018.0	3342.0	...	1.0	1.0	0.0	1.0	0.0
1	0.0	5580.0	0.0	2022.0	2497.0	...	0.0	1.0	0.0	1.0	1.0
2	0.0	4650.0	1.0	2020.0	2497.0	...	0.0	0.0	0.0	1.0	1.0
3	0.0	4880.0	2.0	2021.0	2497.0	...	0.0	1.0	0.0	1.0	1.0
4	0.0	6390.0	0.0	2023.0	3470.0	...	0.0	0.0	0.0	1.0	1.0
..	...	...	...	...	...	...	...	...	...	...	...
517	8.0	930.0	3.0	2016.0	2359.0	...	0.0	1.0	0.0	0.0	0.0
518	8.0	1030.0	8.0	2015.0	2999.0	...	1.0	0.0	0.0	0.0	0.0
519	8.0	620.0	0.0	2012.0	2359.0	...	1.0	0.0	0.0	0.0	0.0
520	8.0	550.0	0.0	2013.0	2359.0	...	1.0	1.0	0.0	0.0	0.0
521	8.0	1100.0	0.0	2017.0	2359.0	...	0.0	0.0	0.0	0.0	0.0

처리된 데이터 확인

처음에는 특정 차량 옵션이 있는 경우 개수를 +1 하여 옵션의 개수를 저장했다.

하지만 선형회귀 모델을 만들다 보니 옵션을 하나씩 따로 저장하는 것이 정확도가 더 높아 옵션을 하나씩 따로 저장했다.
(정확도는 선형회귀 모델에서 추가 설명)

저장된 데이터: 521개 (2024. 6. 6.)

선형회귀 모델 - K-fold Cross Validation

```python
# Linear Regression 모델 만들기
# ------------------------------------------------------
X = np.array(df_car.drop( labels: ['가격(단위: 만)'], axis=1, inplace=False))
Y = np.array(df_car['가격(단위: 만)'])

# 데이터 양이 적으므로 K겹 교차검증
kf = KFold(n_splits=5, shuffle=True, random_state=0)
list_rmse = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # 모델 학습
    model = LinearRegression()
    model.fit(X_train, Y_train)

    Y_pred = model.predict(X_test)
    # 가끔 음수 값이 나와서 절대값 처리
    Y_pred = np.abs(Y_pred)

    mse = mean_squared_error(Y_test, Y_pred)
    list_rmse += [math.sqrt(mse)]
```

```
K겹 교차검증
5개 평균: 991.692888

[0]RMSE: 1053.783609
[1]RMSE: 876.292025
[2]RMSE: 1025.789539
[3]RMSE: 1161.891453
[4]RMSE: 840.707815
```

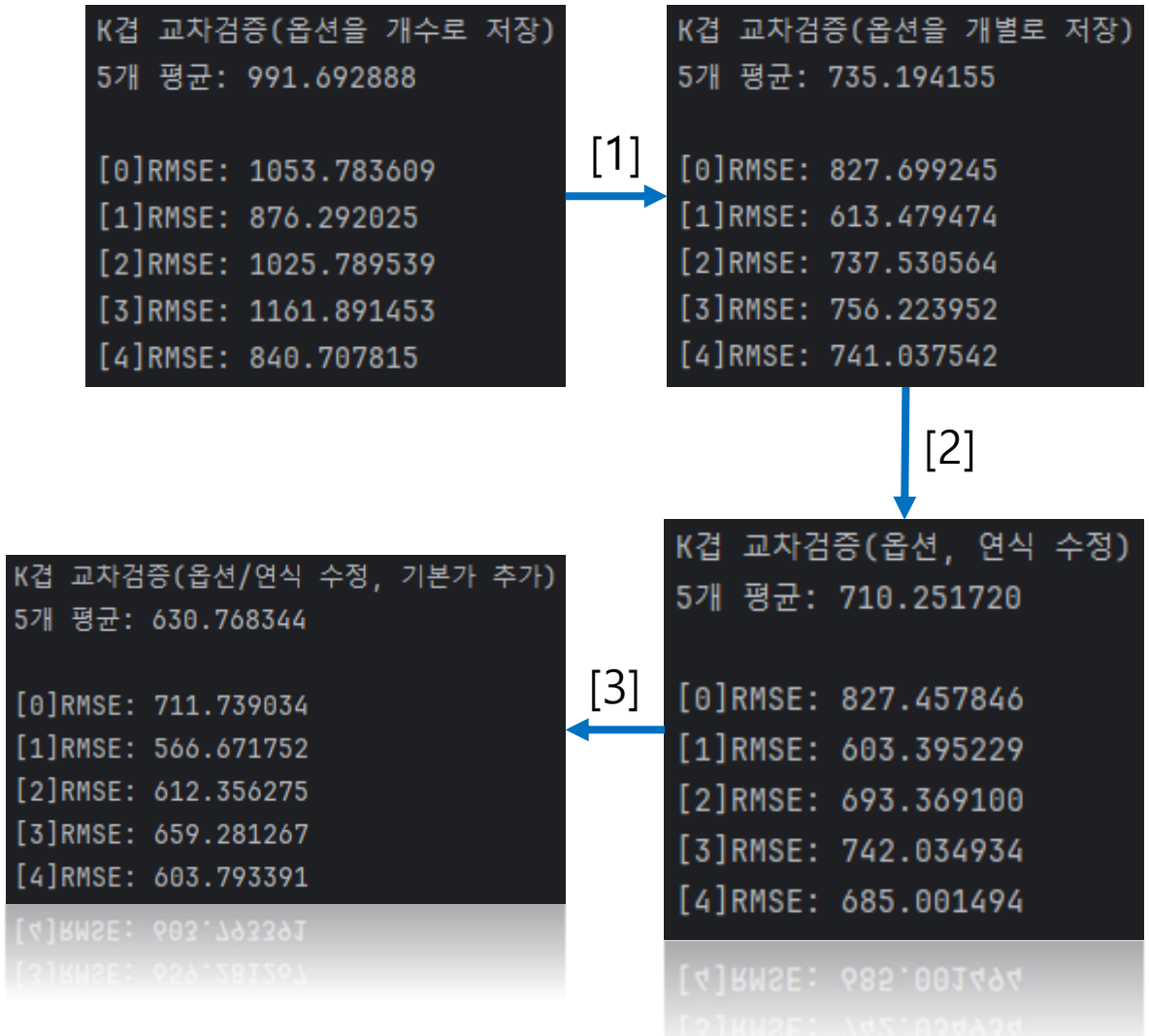
K겹 교차 검증

데이터가 521개 밖에 없으므로 K겹 교차 검증을 사용한다.

데이터를 5개로 쪼개어 선형회귀를 실시했다. 예측된 가격으로 음수가 나오는 경우가 있어 절대값으로 만들어 RMSE 값을 계산했다.

평균 RMSE = 991.69

선형회귀 모델 - Find Optimal Model



최적의 모델 찾기

[1] 전처리에서 설명했던 것과 같이 옵션을 개수로 저장하는 것이 아니라 개별로 저장했다.
평균 RMSE = 991.69 -> 735.19

[2] 연식이 중고가에 큰 영향을 끼친다고 생각, 연식을 (df_car['연식']-1985)**5 로 수정하였다. (제곱 5가 RMSE가 가장 낮았다.)
평균 RMSE = 735.19 -> 710.25

[3] 차량 별 기본가격이 중요하므로 추가했다.
평균 RMSE = 710.25 -> 630.77

결과 - Use Real Data


```
Y_pred = model.predict([[4, 5, (2012-1985)**5, 1999, 180870, 14.0,
          0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
          1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
          1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
          1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2489]])
```

초기 모델

```
예상 차량 가격 (단위: 만)
[849.84278484]
```

→

최종 모델

```
예상 차량 가격 (단위: 만)
[509.93281363]
```

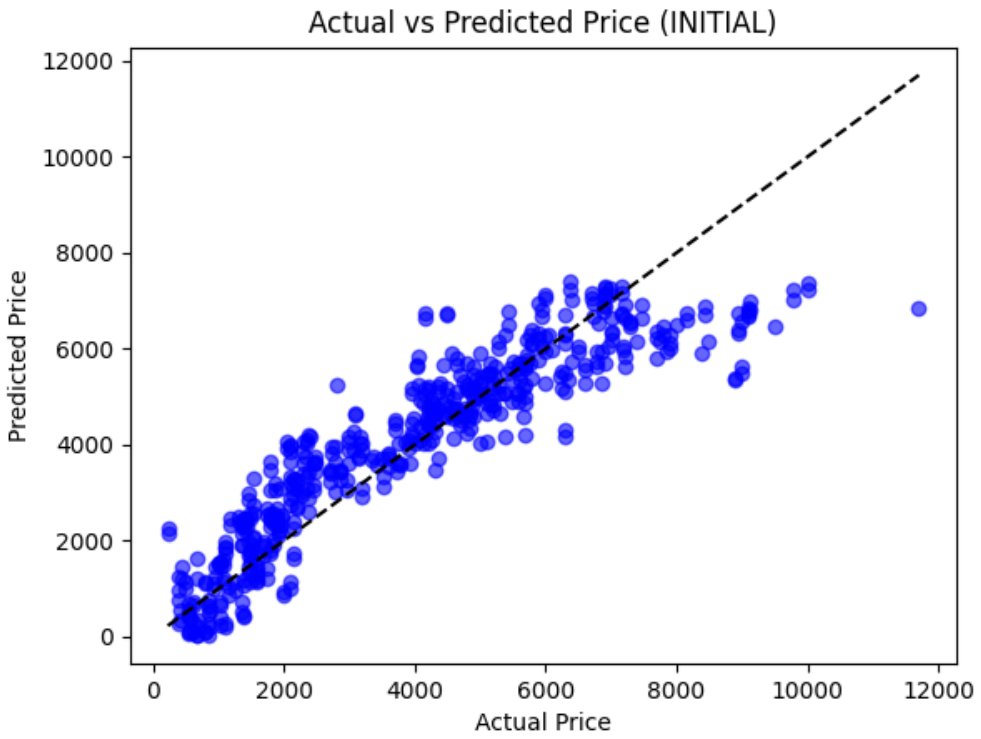
실제로 사용해보기

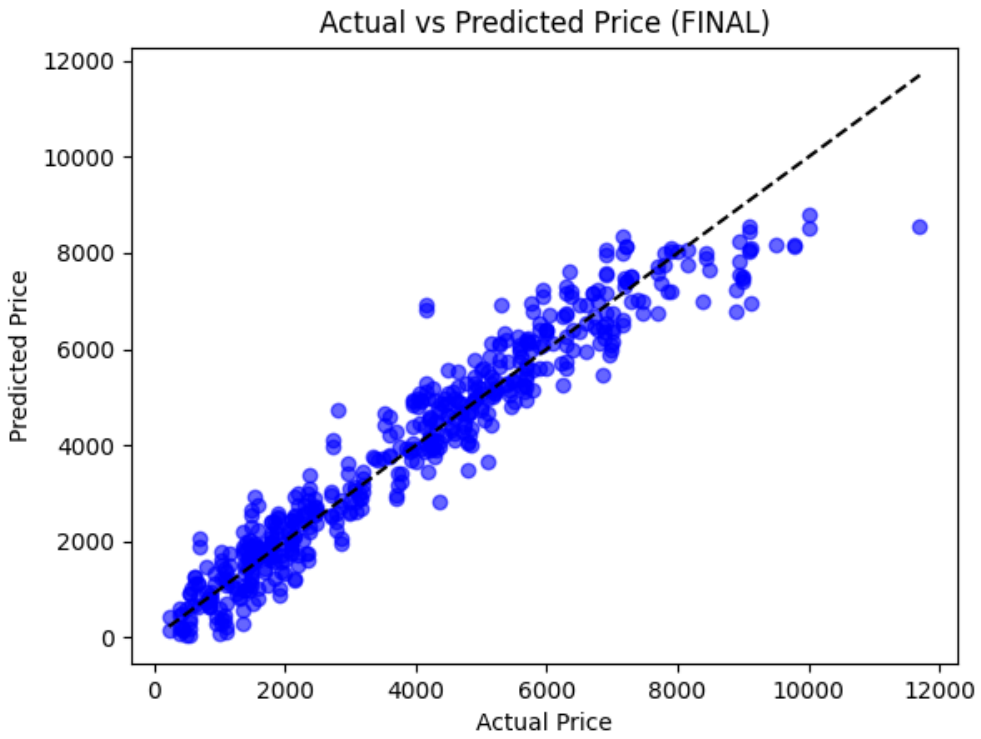
데이터를 수집한 날짜는 2024. 6. 6. 이다.
만든 모델을 실제로 사용해보기 위해 보배드림 사이트에 2024. 6. 7. 에 올라온 중고 차량을 수기로 입력해서 집어넣어 보았다.

실제 가격은 390만원이고 예측한 가격은 509만원으로 실제 가격보다 약 30% 비싸게 예측되었다.

초기 회귀 모델은 849만원으로 예측되었다.

결과 - Scatter Plot (Initial vs Final)





확실히 초기의 선형회귀 모델보다는 최종 모델이 개선된 모습을 보여준다.