딥러닝 분석 프로젝트

집값 예측 머신러닝, 딥러닝 모델 개발

|  |  |  |
| --- | --- | --- |
| 학번 | 학과 | 이름 |
| 20171433 | 정보통계 보험수리학과 | 박우빈 |
| 20171450 | 정보통계 보험수리학과 | 이정진 |
| 20171453 | 정보통계 보험수리학과 | 장재혁 |

제출일: 2022/06/09

목차

[개요 및 데이터 설명 3](#_Toc105716108)

[데이터 탐색 및 전처리 4](#_Toc105716109)

[가격(price) 4](#_Toc105716110)

[평수(sqft\_basement, sqft\_above, sqft\_living) 5](#_Toc105716111)

[방 개수(bedroom, bathroom) 6](#_Toc105716112)

[연도(date, yr\_built, yr\_renovated) 6](#_Toc105716113)

[경치 (waterfront, view) 7](#_Toc105716114)

[위치 (street, city, country, statezip) 8](#_Toc105716115)

[기타(floors, condition) 8](#_Toc105716116)

[데이터 전처리 결과 9](#_Toc105716117)

[모델링 10](#_Toc105716118)

[회귀 모델 10](#_Toc105716119)

[분류 모델 15](#_Toc105716120)

[결과 비교 18](#_Toc105716121)

[결론 19](#_Toc105716122)

[코드 19](#_Toc105716123)

[환경설정 19](#_Toc105716124)

[개요 20](#_Toc105716125)

[데이터 탐색 20](#_Toc105716126)

[데이터 전처리 결과 21](#_Toc105716127)

[모델링 23](#_Toc105716128)

[결과 비교 30](#_Toc105716129)

[역할 31](#_Toc105716130)

# **개요 및 데이터 설명**

집 값은 현 시장의 상황을 잘 설명해주는 아주 중요한 변수이다. 따라서 주어진 데이터로 집 값을 잘 예측할 수 있는 모델을 만드는 것이 목표이다. 딥러닝과 머신러닝을 사용해 모델을 적합할 예정이다. 주어진 데이터의 변수 설명과 예시는 다음과 같다. 데이터는 총 4600개의 행과 18개의 변수로 이루어져 있다.

1. date: 팔린 날짜
2. price: 팔린 가격 (달러)
3. bedrooms: 방 개수
4. bathrooms: 화장실 개수
5. sqft\_living: 실거주 평수 (sqft.)
6. sqrt\_lot: 땅 평수 (sqft.)
7. floors: 층 수
8. waterfront: 오션 뷰 (1: 예, 0: 아니오)
9. view: 특별 뷰
10. condition: 상태
11. sqft\_above: 지상 평수 (sqft.)
12. sqft\_basement: 지하실 평수 (sqft.)
13. yr\_built: 건설 날짜
14. yr\_renovated: 재건축 날짜
15. street: 도로명
16. city: 도시명
17. statezip: 우편번호
18. country: 나라

**Output 1: Data Information**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4600 entries, 0 to 4599

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 date 4600 non-null object

1 price 4600 non-null float64

2 bedrooms 4600 non-null float64

3 bathrooms 4600 non-null float64

4 sqft\_living 4600 non-null int64

5 sqft\_lot 4600 non-null int64

6 floors 4600 non-null float64

7 waterfront 4600 non-null int64

8 view 4600 non-null int64

9 condition 4600 non-null int64

10 sqft\_above 4600 non-null int64

11 sqft\_basement 4600 non-null int64

12 yr\_built 4600 non-null int64

13 yr\_renovated 4600 non-null int64

14 street 4600 non-null object

15 city 4600 non-null object

16 statezip 4600 non-null object

17 country 4600 non-null object

dtypes: float64(4), int64(9), object(5)

memory usage: 647.0+ KB

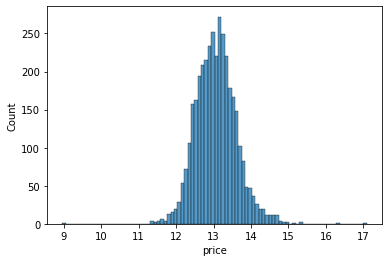
Output 1을 참고하면 데이터에는 결측치는 없으며 연속형 변수와 문자형 변수가 섞여있는 것으로 보인다.

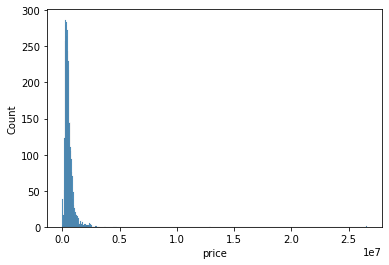
# **데이터 탐색 및 전처리**

가격(price)

Figure 1: Price

Figure 2: Price



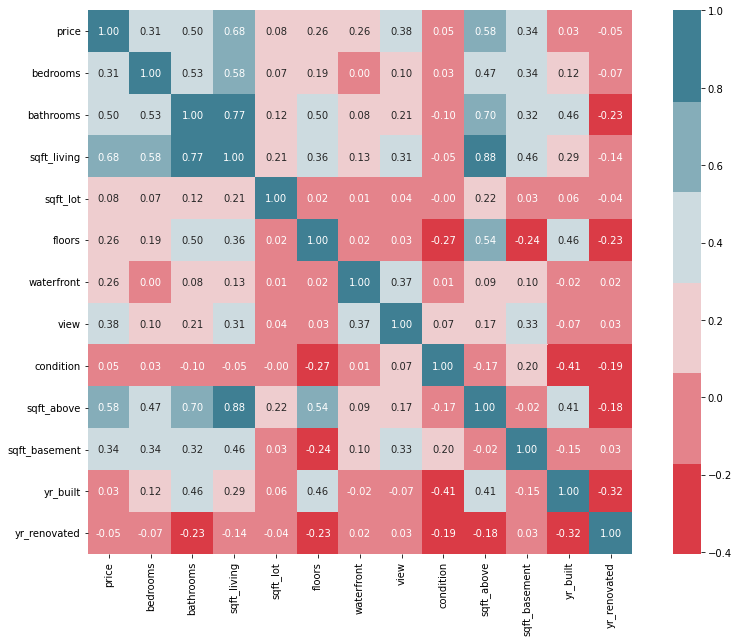


price의 히스토그램을 살펴보면 한쪽으로 치우쳐져 있으며 x축의 범위가 넓은 것으로 보아 극단치가 존재함을 알 수 있다. 또한 0인 값들이 있는 것으로 보인다. Price가 0인 관측치는 이상치라고 판단해 제거하고 분석했다.

Price를 log변환을 했더니 정규분포와 같은 형태를 띄는 것으로 보인다.

따라서 필요에 따라 로그 변환을 해 예측을 진행한다.

Figure 3: Heat Map

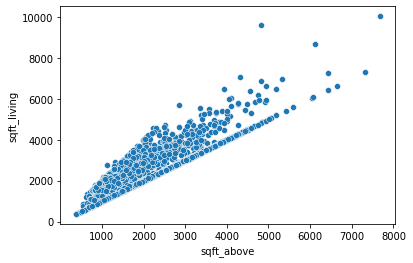
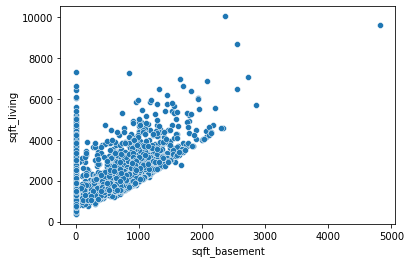


변수들 간 상관관계를 시각화 했을 때 price는 sqft\_living, sqft\_above, sqft\_basement, bathrooms., bedrooms와 상관계수가 큰 것으로 보인다.

평수(sqft\_basement, sqft\_above, sqft\_living)

Figure 4 : Sqft\_above vs Sqft\_living

Figure 5 : Sqft\_basement vs Sqft\_living



Sqft\_living과 sqft\_above간 상관관계가 0.88로 높아 확인했더니 sqft\_above와 sqft\_basement를 더한 값이 sqft\_living과 같았다. 따라서 sqft\_above를 제거하고 sqft\_living만 사용했다.

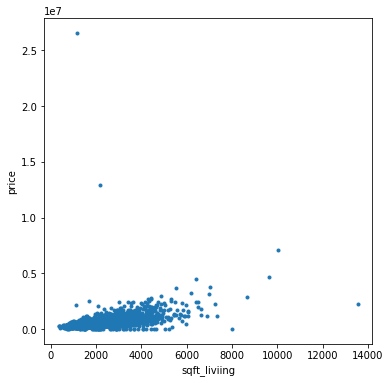


Figure 6 : Sqft\_living

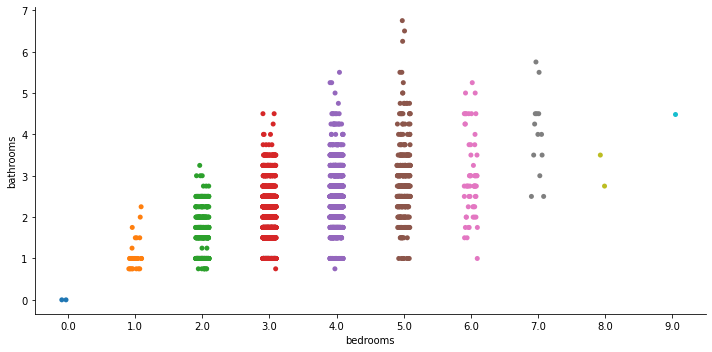
Figure 7 : Sqft\_basement

Price 변수와 sqft\_living간 상관관계가 가장 높아 산점도를 그렸다. 선형관계가 있는 것으로 보이는데 sqft\_living이 작을 때 price가 큰 경우 때문에 선형관계가 나타나지 않는다. 따라서 price > 10000000 & sqft\_living > 12000 인 극단치들을 제거했다.

Sqft\_basement는 지하실 유무라는 변수로 변경해 0과 1의 값을 갖는 변수로 만들었다. Figure7이 변경한 지하실 유무 변수이다. 지하실이 없는 집이 더 많은 것으로 보인다.

방 개수(bedroom, bathroom)

Figure 8 : Bedrooms



bedroom 과 bathroom 은 특성이 비슷하므로 두 변수를 더한 bedrooms\_bathrooms 라는 파생변수를 만들었다.

연도(date, yr\_built, yr\_renovated)

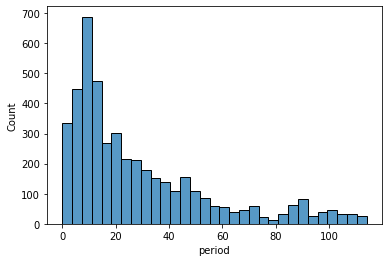
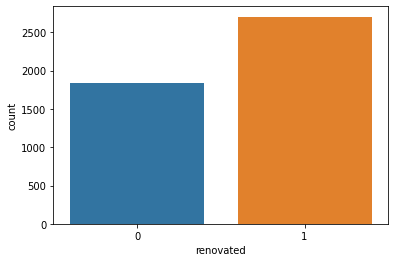


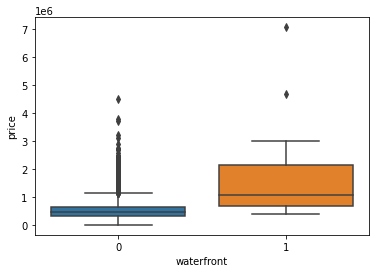
Figure 9 : Renovated

Figure 10 : Period

Date 변수는 전부 2014년 데이터이므로 제거했으며 yr\_renovated 변수는 리모델링 유무라는 변수로 재정의 하였다. Figure9는 리모델링 유무 변수이다. 리모델링 된 집이 더 많은 것으로 보인다.

또한 집이 리모델링 되었다면 리모델링 연도와 리모델링 되지 않았다면 집이 지어진 연도를 빼 집이 판매된 연도를 빼서 판매까지 걸린 시간을 나타내는 period라는 파생변수를 만들었다.

경치 (waterfront, view)



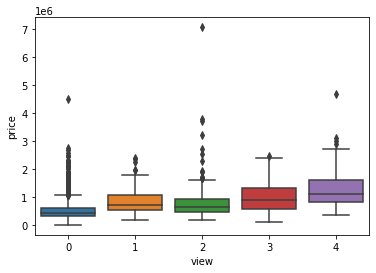


Figure 11 : WaterFront

Figure 12 : View

강 뷰인 집은 price가 높은 것으로 보인다. 따라서 price를 예측할 때 중요한 변수로 보인다.

경치가 좋은 집인 3과 4가 price가 더 높은 것으로 보이며 0일 때는 가장 price가 낮다.

위치 (street, city, country, statezip)

**Output 2: street variable**  **Output 3:city variable**

0 18810 Densmore Ave N

1 709 W Blaine St

2 26206-26214 143rd Ave SE

3 857 170th Pl NE

4 9105 170th Ave NE

...

4595 501 N 143rd St

4596 14855 SE 10th Pl

4597 759 Ilwaco Pl NE

4598 5148 S Creston St

4599 18717 SE 258th St

Name: street, Length: 4548, dtype: object

Seattle 1560

Renton 291

Bellevue 281

Redmond 234

Kirkland 187

Issaquah 186

Kent 183

Auburn 175

Sammamish 171

Federal Way 145

Name: city, dtype: int64

Output 2를 참고하 street 변수는 종류가 많아 유용한 정보를 얻기 어려워 제거하며 country는 USA 한 국가만 있어 제거했다. City는 Output3 처럼 여러 값을 갖고 있지만 city마다 price의 범위가 달라 사용했다. Statezip는 city처럼 여러 값을 가지고 있지만 우편번호와 같이 특정 지역을 담고 있는 정보라고 생각 해 사용했다.

기타(floors, condition)

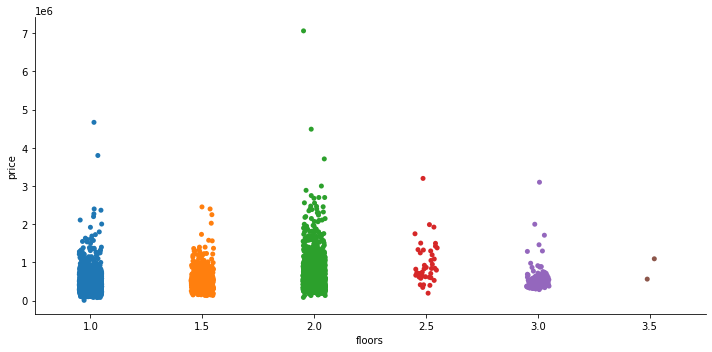
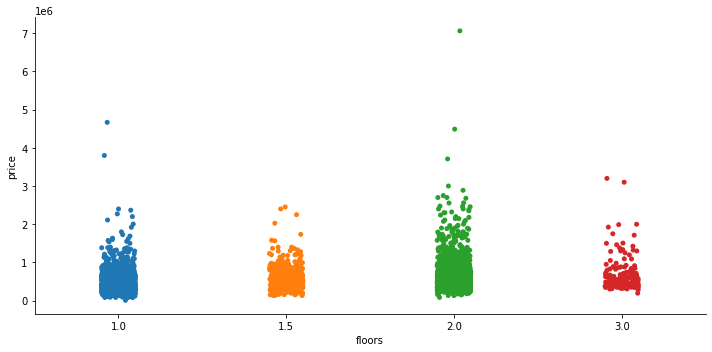


Figure 14 : Floors

Figure 13 : Floors

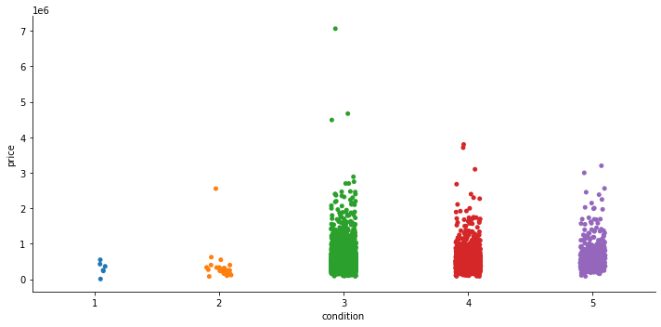
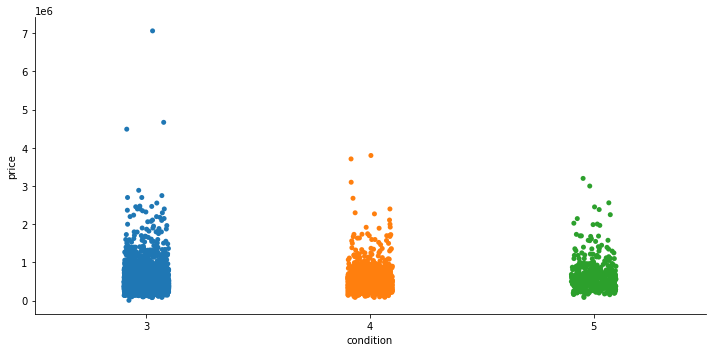


Figure 16 : Condition

Figure 15 : Condition

Figure 14 로 floor 변수는 2.5 층 이상부터는 데이터의 수가 매우 적은 것이 확인되기 때문에 Figure 15 처럼 3 으로 통합했다. Figure 16 으로 condition 변수도 2 이하는 매우 적은 것이 확인되므로 Figure 17 처럼 3 으로 통합했다.

데이터 전처리 결과

앞서 데이터 탐색을 기반으로 전처리를 한 결과는 다음과 같다. 데이터의 크기는 총 4548개의 행과 14개의 변수이다.

**Output 4: Information about Preprocessed Data**

<class 'pandas.core.frame.DataFrame'>

Int64Index: 4548 entries, 0 to 919

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 bedrooms 4548 non-null float64

1 bathrooms 4548 non-null float64

2 sqft\_living 4548 non-null int64

3 sqft\_lot 4548 non-null int64

4 floors 4548 non-null float64

5 waterfront 4548 non-null int64

6 view 4548 non-null int64

7 condition 4548 non-null int64

8 city 4548 non-null object

9 statezip 4548 non-null object

10 basement 4548 non-null int64

11 bedrooms\_bathrooms 4548 non-null float64

12 renovated 4548 non-null int64

13 period 4548 non-null float64

dtypes: float64(5), int64(7), object(2)

memory usage: 533.0+ KB

# 모델링

회귀 모델은 변수의 범위의 영향을 크게 받으므로, scaling 된 데이터로 학습을 한다. 분류 모델은 종속 변수인 price 데이터를 오름차순으로 정렬하여 빈도수에 따라 삼등분을 하여 label을 부여했다.

회귀 모델

딥러닝 모델 1

**Output 5: Layers**

Model: "sequential\_13"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_68 (Dense) (None, 512) 7680

dense\_69 (Dense) (None, 256) 131328

dense\_70 (Dense) (None, 32) 8224

dense\_71 (Dense) (None, 16) 528

dense\_72 (Dense) (None, 4) 68

dense\_73 (Dense) (None, 1) 5

=================================================================

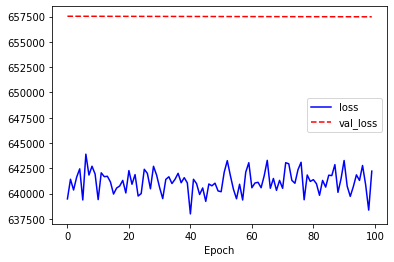
Total params: 147,833

Trainable params: 147,833

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure 17 : Loss & Validation Loss Plot



Dense Layer 을 총 6 개 층을 쌓았고, activation function 은 Relu 함수를 사용했다. 총 parameter 개수는 147,833 개이다. Adam Optimizer 을 사용했으며 학습률은 0.007 로 설정하였다. Batch size 는 32 로 설정하였다. Figure 17 을 참고하면 Valid data의 loss가 떨어지지 않는 것을 보아 전혀 학습이 되지 않은 것을 확인할 수 있다. 모델의 복잡한 구조가 이유인 것으로 예측된다.

딥러닝 모델 2

**Output 6: Layers**

Model: "sequential\_15"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_79 (Dense) (None, 128) 1920

dropout\_22 (Dropout) (None, 128) 0

dense\_80 (Dense) (None, 64) 8256

dropout\_23 (Dropout) (None, 64) 0

dense\_81 (Dense) (None, 32) 2080

dropout\_24 (Dropout) (None, 32) 0

dense\_82 (Dense) (None, 16) 528

dropout\_25 (Dropout) (None, 16) 0

dense\_83 (Dense) (None, 1) 17

=================================================================

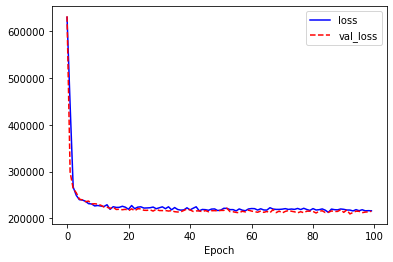
Total params: 12,801

Trainable params: 12,801

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure 18 : Loss & Validation Loss Plot



모델 1이 복잡해 전혀 학습되지 않았다고 판단하였기 때문에 parameter 개수를 줄이는 방향을 택하였다. 따라서, 현 모델은 레이어 수를 줄이고, random uniform intialization 과 dropout (0.1) 을 사용하여 학습의 안정을 도왔다. 학습의 안정을 위해 학습률은 모델1과 달리 0.001을 사용하였으며 Batch size 는 16을 사용하였다. Figure 18 을 참고하면, Figure 17 에 비해 학습이 잘 되는 것으로 보인다. 하지만 validation data가 train data보다 낮은 loss를 보인다.

딥러닝 모델 3

**Output 7: Layers**

Model: "sequential\_16"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_84 (Dense) (None, 128) 1920

batch\_normalization\_24 (Bat (None, 128) 512

chNormalization)

leaky\_re\_lu\_24 (LeakyReLU) (None, 128) 0

dense\_85 (Dense) (None, 64) 8256

batch\_normalization\_25 (Bat (None, 64) 256

chNormalization)

leaky\_re\_lu\_25 (LeakyReLU) (None, 64) 0

dropout\_26 (Dropout) (None, 64) 0

dense\_86 (Dense) (None, 32) 2080

batch\_normalization\_26 (Bat (None, 32) 128

chNormalization)

leaky\_re\_lu\_26 (LeakyReLU) (None, 32) 0

dropout\_27 (Dropout) (None, 32) 0

dense\_87 (Dense) (None, 16) 528

batch\_normalization\_27 (Bat (None, 16) 64

chNormalization)

leaky\_re\_lu\_27 (LeakyReLU) (None, 16) 0

dropout\_28 (Dropout) (None, 16) 0

dense\_88 (Dense) (None, 1) 17

=================================================================

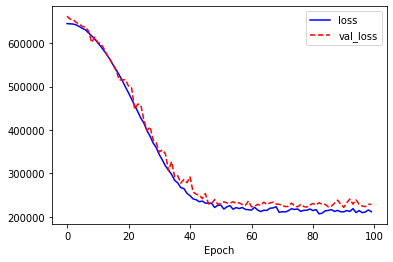
Total params: 13,761

Trainable params: 13,281

Non-trainable params: 480

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figure 18 : Loss & Validation Loss Plot



마지막으로, 모델 3은 좋은 성능을 보였던 모델 2의 구조와 동일하게 하였고, 추가로 배치 정규화와 activation function으로 Leaky Relu를 사용하였다. 배치 정규화로 인해 batch size 는 모델2 와 달리 64를 사용하였다. 배치 정규화를 사용했기 때문에 학습률을 0.07 로 증가시켰다. Figure 21 을 참고하면 모델 1에 비해 과대적합 되지 않았고, 모델 2 에 나타난 valid loss가 train loss보다 작게 나오는 것을 해결하였다.

머신러닝 모델

선형회귀

**Output 8: RMSE**

선형회귀 5-교차검증 rmse : 236224.7476226245

선형모델의 5-kfold 교차 검증을 통해 구한 RMSE 값은 약 236225 이다.

선형회귀를 사용할 때는 데이터를 정규화를 진행했다.

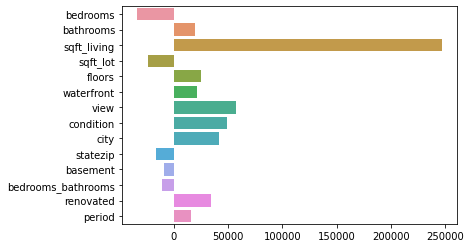
릿지회귀

**Output 9: RMSE**

릿지 5-교차검증 rmse : 236286.24375029508

alpha : 10

Figure 19 : Loss & Validation Loss Plot



5-kfold 로 최적의 RMSE 값을 반환해주는 릿지 상수를 반복문을 통해 구했다. 최적의 상수 값은 10이고 RMSE 값은 약 236286 이다. Figure 19 를 참고하면 sqft\_living 이 가장 중요한 변수로 보인다.

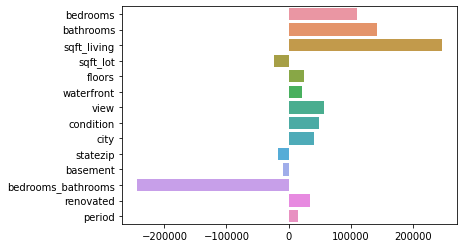
라쏘회귀

**Output 12: RMSE**

라쏘 5-교차검증 rmse : 236305.3606508985

alpha : 100

Figure 20 : Loss & Validation Loss Plot



동일하게 5-kfold 로 최적의 RMSE 값을 반환해주는 라쏘 상수를 반복문을 통해 구했다. 최적의 상수 값은 100 이고 이 때 RMSE 값은 약 236305 이다. 이와 더불어 Figure 20 를 참고하면 sqft\_living 변수와 파생 변수인 bedrooms\_bathrooms 가 중요한 변수로 보인다.

분류 모델

딥러닝 모델 1

**Output 8: Layers**

Model: "sequential\_24"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_125 (Dense) (None, 64) 960

dropout\_45 (Dropout) (None, 64) 0

dense\_126 (Dense) (None, 48) 3120

dropout\_46 (Dropout) (None, 48) 0

dense\_127 (Dense) (None, 32) 1568

dropout\_47 (Dropout) (None, 32) 0

dense\_128 (Dense) (None, 16) 528

dropout\_48 (Dropout) (None, 16) 0

dense\_129 (Dense) (None, 3) 51

=================================================================

Total params: 6,227

Trainable params: 6,227

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

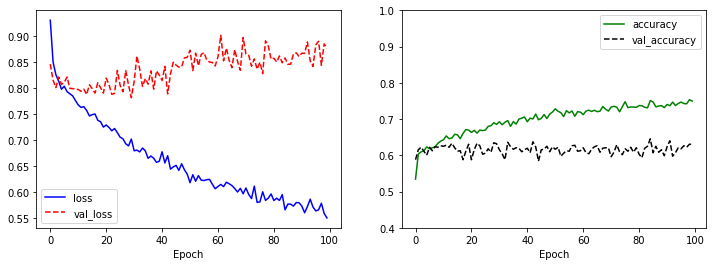


Figure 21 : Loss & Validation Loss Plot

모델 1 은 총 5개의 dense layer을 쌓았고 사이마다 dropout (0.1)을 설정하였다. 그리고 activation function으로 Relu 와 분류를 위해 Softmax를 사용하였다. Figure 21를 참고하면 train loss와 valid loss의 간격이 학습하면서 멀어진다. 과대적합의 경향이 관측된다.

딥러닝 모델 2

**Output 9: Layers**

Model: "sequential\_22"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_114 (Dense) (None, 1024) 15360

batch\_normalization\_40 (Bat (None, 1024) 4096

chNormalization)

leaky\_re\_lu\_40 (LeakyReLU) (None, 1024) 0

dense\_115 (Dense) (None, 512) 524800

batch\_normalization\_41 (Bat (None, 512) 2048

chNormalization)

leaky\_re\_lu\_41 (LeakyReLU) (None, 512) 0

dropout\_38 (Dropout) (None, 512) 0

dense\_116 (Dense) (None, 512) 262656

batch\_normalization\_42 (Bat (None, 512) 2048

chNormalization)

leaky\_re\_lu\_42 (LeakyReLU) (None, 512) 0

dropout\_39 (Dropout) (None, 512) 0

dense\_117 (Dense) (None, 128) 65664

batch\_normalization\_43 (Bat (None, 128) 512

chNormalization)

leaky\_re\_lu\_43 (LeakyReLU) (None, 128) 0

dropout\_40 (Dropout) (None, 128) 0

dense\_118 (Dense) (None, 64) 8256

batch\_normalization\_44 (Bat (None, 64) 256

chNormalization)

leaky\_re\_lu\_44 (LeakyReLU) (None, 64) 0

dropout\_41 (Dropout) (None, 64) 0

dense\_119 (Dense) (None, 3) 195

=================================================================

Total params: 885,891

Trainable params: 881,411

Non-trainable params: 4,480

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

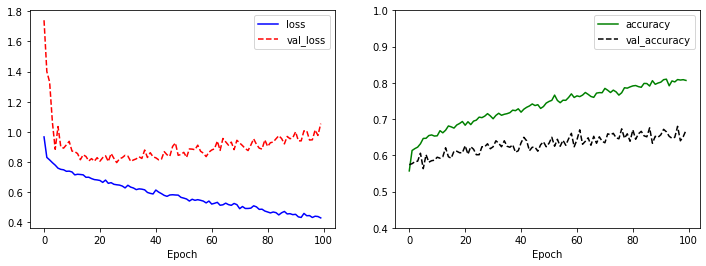


Figure 22 : Loss & Validation Loss Plot

모델 2는 6개의 dense layer 을 쌓았고, activation function 은 Leaky Relu 와 Softmax 를 사용하였다. 층 사이마다 모델 1과 동일하게 dropout (0.1) 을 설정하였고, 추가로 과대적합을 해소하기 위해 배치 정규화를 사용하였다. 또한He 가중치 초기화 방법을 사용했다. Figure 22를 참고하면 Figure 21 보다 과대적합의 경향이 줄어든 것으로 확인된다.

랜덤 포레스트

**Output 13: Accuracy**

1번째 교차검증 accuracy : 73.63%

2번째 교차검증 accuracy : 72.66%

3번째 교차검증 accuracy : 74.59%

4번째 교차검증 accuracy : 71.70%

5번째 교차검증 accuracy : 73.76%

랜덤포레스트 5-교차검증 accuracy : 73.27%

5-kfold로 교차검증을 했을 때 랜덤포레스트 모델을 사용 해 분류 시 73.27% 의 정확도를 확인할 수 있다.

Lightgbm

**Output 14: Accuracy**

1번째 교차검증 accuracy : 80.63%

2번째 교차검증 accuracy : 78.57%

3번째 교차검증 accuracy : 79.12%

4번째 교차검증 accuracy : 80.77%

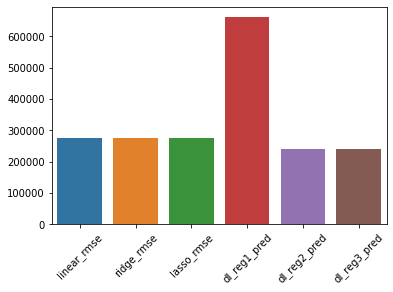
5번째 교차검증 accuracy : 79.26%

lgboost 5-교차검증 accuracy : 79.67%

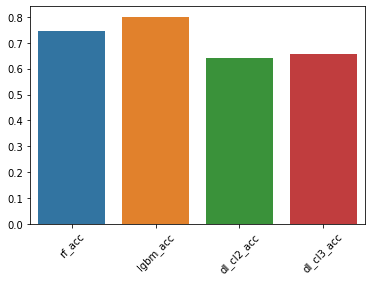
5-kfold 교차 검증을 사용 해 Lightgbm 분류 시 79.67% 의 정확도를 확인할 수 있다.

## 결과 비교

Figure 23 : 회귀 모델 RMSE 비교



**Figure 24 :** **분류 모델 Accuracy 비교**



각 모델의 성능을 Test 데이터를 사용해 비교했다. Figure 23 을 참고하면, 딥러닝 모델 2 가 가장 낮은 RMSE 값을 갖는다. 하지만 학습이 진행되지 않은 모델을 제외하고는 모두 비슷한 RMSE 값을 보이고 있다.

Figure 24 을 참고하면 Light gbm을 사용했을 때 가장 높은 accuracy 값을 갖는다.

# **결론**

House Price 데이터를 활용해 딥러닝 모델과 머신러닝 모델을 사용해 가격을 예측을 진행했다.

딥러닝 모델은 먼저 학습이 되지 않거나 과대 적합이 일어나는 경우로 학습을 시켰으며 이를 해결하는 과정으로 모델링을 진행했다. 드롭아웃, 배치정규화, 가중치 초기화, 활성화 함수를 변경해 가면서 학습을 진행했으며 이러한 방법들을 사용하니 처음 학습보다 train loss와 valid loss간 간격을 줄이는 학습이 가능했고 RMSE나 정확성 측면 모두 머신러닝 모델과 비슷한 성능을 보였다. 데이터가 price와 sqft living 변수간 상관관계가 강했고 이상치를 제거했을 때 선형관계를 띄어 회귀 모델링은 선형모델, Ridge 모델, Lasso 모델 또한 좋은 성능을 보였으며 분류 모델링은 머신러닝 모델의 정확도가 높았지만 딥러닝 모델은 구조의 영향을 받는 것으로 보인다. 따라서 조금 더 복잡하지만 규제를 적용하는 방법으로 학습한다면 좋은 성능으로 예측할 것으로 보인다.

# **코드**

환경설정

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import StratifiedKFold, KFold, cross\_val\_score

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from xgboost import XGBRegressor, XGBClassifier

from lightgbm import LGBMRegressor, LGBMClassifier

from catboost import CatBoostRegressor, CatBoostClassifier

from sklearn.metrics import mean\_squared\_error, accuracy\_score

from sklearn.preprocessing import StandardScaler, RobustScaler

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import BatchNormalization

from keras.layers import advanced\_activations

from keras.utils import layer\_utils

from keras.layers.advanced\_activations import LeakyReLU

import scipy.stats

개요

# table 1

data = pd.read\_csv('data.csv')

data.head(5)

# table 2

data.describe().T

# output 1

data.info()

# figure 1

plt.figure(figsize=(15,10))

cmap = sns.diverging\_palette(10,220,as\_cmap=False)

sns.heatmap(data.corr(),annot=True,square=True,fmt=".2f",cmap=cmap)

데이터 탐색

# figure 2

plt.figure(figsize=(20, 6))

sns.histplot(train["price"])

# figure 3

plt.figure(figsize=(20, 6))

plt.subplot(131)

plt.plot(data['sqft\_living'], data['price'], '.')

plt.ylabel('price')

plt.xlabel('sqft\_living')

# figure 4

plt.figure(figsize=(20, 6))

sns.histplot(train["price"])

# figure 5

plt.figure(figsize=(20, 6))

plt.subplot(131)

plt.plot(data['sqft\_living'], data['price'], '.')

plt.ylabel('price')

plt.xlabel('sqft\_liviing')

# figure 6

data["basement"] = data["sqft\_basement"].apply(lambda x : 1 if x!=0 else 0)

sns.barplot(x=data['basement'], y=data['basement'].value\_counts()).set(ylabel='count')

# figure 7

sns.catplot(x='bedrooms', y='bathrooms', data=data, height=5, aspect=2)

# figure 8

data["renovated"] = data["yr\_renovated"].apply(lambda x : 1 if x !=0 else 0)

sns.barplot(x=data['renovated'], y=data['renovated'].value\_counts()).set(ylabel='count')

# figure 9

data["period"] = data.apply(lambda x : x[19] - x[12] if x[18] == 0 else x[19] - x[13] ,axis=1)

sns.histplot(data.period)

# figure 10

sns.boxplot(x=data['waterfront'], y=data['price'])

# figure 11

sns.boxplot(x=data['view'], y=data['price'])

# figure 12

data["waterfront\_view"] = data["waterfront"].astype(str) + "\_" + data["view"].astype(str)

sns.boxplot(x=data['waterfront\_view'], y=data['price'])

# output 2

data["street"]

# output 3

data['city'].value\_counts().head(10)

# figure 13

colors = sns.color\_palette('pastel')[0:5]

city = data['city'].value\_counts().head(5)

plt.pie(city, labels=city.index, colors = colors, autopct='%.0f%%')

# figure 14

sns.catplot(x="floors", y="price", data=data, height=5, aspect=2)

# figure 15

floor\_scores = {1: 1, 1.5:1.5, 2:2, 2.5:3, 3:3, 3.5:3}

data["floors"] = data["floors"].map(floor\_scores)

print(data["floors"].unique())

sns.catplot(x="floors", y="price", data=data, height=5, aspect=2)

# figure 16

sns.catplot(x="condition", y="price", data=data, height=5, aspect=2)

# figure 17

condition\_scores = {1:3, 2:3, 3:3, 4:4, 5:5}

data["condition"] = data["condition"].map(condition\_scores)

sns.catplot(x="condition", y="price", data=data, height=5, aspect=2)

데이터 전처리 결과

# 새로 불러오기 및 분할

data = pd.read\_csv('data.csv')

train,test = train\_test\_split(data,shuffle=True,test\_size=0.2,random\_state=156)

train = train.reset\_index(drop=True)

test = test.reset\_index(drop=True)

# price 관련 전처리

train = train.loc[(train['price'] < 10000000) & (train['sqft\_living'] < 12000) & (train['price'] != 0)]

test = test.loc[(test['price'] < 10000000) & (test['sqft\_living'] < 12000) & (test['price'] != 0)]

# 평수 관련 전처리

train["basement"] = train["sqft\_basement"].apply(lambda x : 1 if x!=0 else 0)

test["basement"] = test["sqft\_basement"].apply(lambda x : 1 if x!=0 else 0)

# 방 개수 관련 전처리

train["bedrooms\_bathrooms"] = train["bedrooms"] + train["bathrooms"]

test["bedrooms\_bathrooms"] = test["bedrooms"] + test["bathrooms"]

# 년도 관련 전처리

train["renovated"] = train["yr\_renovated"].apply(lambda x : 1 if x !=0 else 0)

train["date"] = pd.to\_datetime(train["date"])

train["year"] = train["date"].dt.year

train["period"] = train.apply(lambda x : x[19] - x[12] if x[18] == 0 else x[19] - x[13] ,axis=1)

test["renovated"] = test["yr\_renovated"].apply(lambda x : 1 if x !=0 else 0)

test["date"] = pd.to\_datetime(test["date"])

test["year"] = test["date"].dt.year

test["period"] = test.apply(lambda x : x[19] - x[12] if x[18] == 0 else x[19] - x[13] ,axis=1)

# 경치 관련 전처리 : 딱히 없음

# 위치 관련 전처리: 딱히 없음

# 기타 관련 전처리

floor\_scores = {1: 1, 1.5:1.5, 2:2, 2.5:3, 3:3, 3.5:3}

train["floors"] = train["floors"].map(floor\_scores)

test["floors"] = test["floors"].map(floor\_scores)

condition\_scores = {1:3, 2:3, 3:3, 4:4, 5:5}

train["condition"] = train["condition"].map(condition\_scores)

test["condition"] = test["condition"].map(condition\_scores)

# Data for regression & classification

train\_regression = train.drop(["price", "sqft\_above", "sqft\_basement", "yr\_renovated", "yr\_built", "date", "year", "street", "country"],axis=1)

test\_regression = test.drop(["price", "sqft\_above", "sqft\_basement", "yr\_renovated", "yr\_built", "date", "year", "street", "country"],axis=1)

train\_regression\_y = train["price"]

test\_regression\_y = test["price"]

train\_classification = train.drop(["price", "sqft\_above", "sqft\_basement", "yr\_renovated", "yr\_built", "date", "year", "street", "country"],axis=1)

test\_classification = test.drop(["price", "sqft\_above", "sqft\_basement", "yr\_renovated", "yr\_built", "date", "year", "street", "country"],axis=1)

train\_classification\_y = pd.qcut(train["price"],3,labels=[0,1,2])

test\_classification\_y = pd.qcut(test["price"],3,labels=[0,1,2])

# output 4

train\_regression.append(test\_regression).info()

# figure 18

plt.figure(figsize=(15,10))

cmap = sns.diverging\_palette(10,220,as\_cmap=False)

sns.heatmap(train\_regression.corr(),annot=True,square=True,fmt=".2f",cmap=cmap)

모델링

# Scaler

lb = LabelEncoder()

c = ["city", "statezip"]

for col in c:

train\_regression[col] = lb.fit\_transform(train\_regression[col])

test\_regression[col] = lb.transform(test\_regression[col])

scaler = StandardScaler()

scaler.fit(train\_regression)

scaler\_train\_regression = scaler.transform(train\_regression)

scaler\_test\_regression = scaler.transform(test\_regression)

for col in c:

train\_classification[col] = lb.fit\_transform(train\_classification[col])

test\_classification[col] = lb.transform(test\_classification[col])

scaler = StandardScaler()

scaler.fit(train\_classification)

scaler\_train\_classification = scaler.transform(train\_classification)

scaler\_test\_classification = scaler.transform(test\_classification)

# 딥러닝

from keras import backend as K

def root\_mean\_squared\_error(y\_true, y\_pred):

return K.sqrt(K.mean(K.square(y\_pred - y\_true)))

# 회귀

train\_regression\_X, train\_regression\_y = scaler\_train\_regression, train\_regression\_y

test\_regression\_X, test\_regression\_y = scaler\_test\_regression, test\_regression\_y

# 분류

train\_classification\_X, train\_dl\_classification\_y = scaler\_train\_classification, tf.keras.utils.to\_categorical(train\_classification\_y, num\_classes=3)

test\_classification\_X, test\_dl\_classification\_y = scaler\_test\_classification, tf.keras.utils.to\_categorical(test\_classification\_y, num\_classes=3)

# output 5

# 레이어 설정

model1 = tf.keras.Sequential([

tf.keras.layers.Dense(units=512, input\_shape=(14,),activation='relu'),

tf.keras.layers.Dense(units=256,activation='relu'),

tf.keras.layers.Dense(units=32,activation='relu'),

tf.keras.layers.Dense(units=16,activation='relu'),

tf.keras.layers.Dense(units=4, activation='relu'),

tf.keras.layers.Dense(units=1)

])

model1.compile(optimizer=tf.keras.optimizers.Adam(lr=0.007), loss=root\_mean\_squared\_error)

model1.summary()

# 모델 학습

tf.random.set\_seed(156)

history1 = model1.fit(train\_regression\_X, train\_regression\_y, epochs=100, batch\_size=32, validation\_split=0.2)

# figure 19

plt.plot(history1.history['loss'], 'b-', label='loss')

plt.plot(history1.history['val\_loss'], 'r--', label='val\_loss')

plt.xlabel('Epoch')

plt.legend()

plt.savefig("model1\_2")

plt.show()

# output 6

# 레이어 설정

model2 = tf.keras.Sequential([

tf.keras.layers.Dense(units=128, activation='relu', input\_shape=(14,)),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=64, activation='relu', kernel\_initializer='random\_uniform'),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=32, activation='relu', kernel\_initializer='random\_uniform'),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=16, activation='relu', kernel\_initializer='random\_uniform'),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=1)

])

model2.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001), loss=root\_mean\_squared\_error)

model2.summary()

# figure 20

plt.plot(history2.history['loss'], 'b-', label='loss')

plt.plot(history2.history['val\_loss'], 'r--', label='val\_loss')

plt.xlabel('Epoch')

plt.legend()

plt.savefig('model2\_2.png')

plt.show()

# 모델 학습

tf.random.set\_seed(156)

history2 = model2.fit(train\_regression\_X, train\_regression\_y, epochs=100, batch\_size=16, validation\_split=0.2)

# output 7

# 레이어 설정

model3 = tf.keras.Sequential([

tf.keras.layers.Dense(units=128, input\_shape=(14,), kernel\_initializer="he\_normal"),

BatchNormalization(),

LeakyReLU(0.3),

tf.keras.layers.Dense(units=64, kernel\_initializer="he\_normal"),

BatchNormalization(),

LeakyReLU(0.3),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=32, kernel\_initializer="he\_normal"),

BatchNormalization(),

LeakyReLU(0.3),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=16,kernel\_initializer="he\_normal"),

BatchNormalization(),

LeakyReLU(0.3),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=1)

])

model3.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss=root\_mean\_squared\_error)

model3.summary()

# 모델 학습

tf.random.set\_seed(156)

history3 = model3.fit(train\_regression\_X, train\_regression\_y, epochs=100, batch\_size=64, validation\_split=0.2)

# figure 21

plt.plot(history3.history['loss'], 'b-', label='loss')

plt.plot(history3.history['val\_loss'], 'r--', label='val\_loss')

plt.xlabel('Epoch')

plt.legend()

plt.savefig('model3\_1.png')

plt.show()

# output 8

model\_cl2 = tf.keras.Sequential([

tf.keras.layers.Dense(units=64, activation='relu', input\_shape=(14,)),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=48, activation='relu'),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=32, activation='relu'),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=16, activation='relu'),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=3, activation = 'softmax')

])

model\_cl2.compile(optimizer=tf.keras.optimizers.Adam(lr=0.004),

loss='categorical\_crossentropy',

metrics=['accuracy'])

model\_cl2.summary()

tf.random.set\_seed(156)

history\_cl2 = model\_cl2.fit(train\_classification\_X, train\_dl\_classification\_y, epochs=100, batch\_size=32, validation\_split=0.2)

# figure 22

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history\_cl2.history['loss'], 'b-', label='loss')

plt.plot(history\_cl2.history['val\_loss'], 'r--', label='val\_loss')

plt.xlabel('Epoch')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history\_cl2.history['accuracy'], 'g-', label='accuracy')

plt.plot(history\_cl2.history['val\_accuracy'], 'k--', label='val\_accuracy')

plt.xlabel('Epoch')

plt.ylim(0.4, 1)

plt.legend()

plt.show()

# output 9

model\_cl3 = tf.keras.Sequential([

tf.keras.layers.Dense(units=1024, input\_shape=(14,), kernel\_initializer = 'he\_normal'),

BatchNormalization(),

LeakyReLU(0.2),

tf.keras.layers.Dense(units=512, kernel\_initializer = 'he\_normal'),

BatchNormalization(),

LeakyReLU(0.2),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=512, kernel\_initializer = 'he\_normal'),

BatchNormalization(),

LeakyReLU(0.2),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=128, kernel\_initializer = 'he\_normal'),

BatchNormalization(),

LeakyReLU(0.2),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=64, kernel\_initializer = 'he\_normal'),

BatchNormalization(),

LeakyReLU(0.2),

tf.keras.layers.Dropout(0.1),

tf.keras.layers.Dense(units=3, activation = 'softmax')

])

model\_cl3.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),

loss='categorical\_crossentropy',

metrics=['accuracy'])

model\_cl3.summary()

tf.random.set\_seed(156)

history\_cl3 = model\_cl3.fit(train\_classification\_X, train\_dl\_classification\_y, epochs=100, batch\_size=128, validation\_split=0.2)

# figure 23

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history\_cl3.history['loss'], 'b-', label='loss')

plt.plot(history\_cl3.history['val\_loss'], 'r--', label='val\_loss')

plt.xlabel('Epoch')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history\_cl3.history['accuracy'], 'g-', label='accuracy')

plt.plot(history\_cl3.history['val\_accuracy'], 'k--', label='val\_accuracy')

plt.xlabel('Epoch')

plt.ylim(0.4, 1)

plt.legend()

plt.show()

# 머신러닝

# output 10

linear = LinearRegression()

linear\_rmse = np.mean(np.sqrt(-cross\_val\_score(linear,scaler\_train\_regression,train["price"],cv=5,scoring="neg\_mean\_squared\_error")))

print("선형회귀 5-교차검증 rmse : {}".format(linear\_rmse))

# output 11

min = 1000000000000000

index = 0

for i in [100,10,1,0.1,0.01,0.001]:

ridge = Ridge(alpha=i)

ridge\_rmse = np.mean(np.sqrt(-cross\_val\_score(ridge,scaler\_train\_regression,train["price"],cv=5,scoring="neg\_mean\_squared\_error")))

if min > ridge\_rmse:

min = ridge\_rmse

index = i

print("릿지 5-교차검증 rmse : {}".format(min))

print("alpha : {}".format(index))

# output 12

min = 1000000000000000

index = 0

for i in [100,10,1,0.1,0.01,0.001]:

lasso = Lasso(alpha=i, max\_iter=1000)

lasso\_rmse = np.mean(np.sqrt(-cross\_val\_score(lasso,scaler\_train\_regression,train["price"],cv=5,scoring = "neg\_mean\_squared\_error")))

if min > ridge\_rmse:

min = ridge\_rmse

index = i

print("라쏘 5-교차검증 rmse : {}".format(min))

print("alpha : {}".format(index))

# figure 24

sns.barplot(y=train\_regression.columns,x=ridge.coef\_)

# figure 25

sns.barplot(y=train\_regression.columns,x=lasso.coef\_)

# output 13

kfold = StratifiedKFold(n\_splits=5,shuffle=True,random\_state=156)

rf = RandomForestClassifier(random\_state=156)

rf\_accuracy = 0

i = 1

for train\_idx,valid\_idx in kfold.split(train\_classification,train["price"]):

X\_train,X\_valid = train\_classification.iloc[train\_idx], train\_classification.iloc[valid\_idx]

y\_train,y\_valid = train["price"].iloc[train\_idx], train["price"].iloc[valid\_idx]

rf.fit(X\_train,y\_train)

rf\_pred = rf.predict(X\_valid)

rf\_accuracy += accuracy\_score(rf\_pred,y\_valid) / 5

print("{}번째 교차검증 accuracy : {:.2f}%".format(i, accuracy\_score(rf\_pred,y\_valid) \* 100))

i += 1

print("랜덤포레스트 5-교차검증 accuracy : {:.2f}%".format(rf\_accuracy \* 100))

# output 14

kfold = StratifiedKFold(n\_splits=5,shuffle=True,random\_state=156)

lgbm = LGBMClassifier()

lgbm\_accuracy = 0

i = 1

for train\_idx,valid\_idx in kfold.split(train\_classification,train["price"]):

X\_train,X\_valid = train\_classification.iloc[train\_idx], train\_classification.iloc[valid\_idx]

y\_train,y\_valid = train["price"].iloc[train\_idx], train["price"].iloc[valid\_idx]

lgbm.fit(X\_train,y\_train,verbose=0)

lgbm\_pred = lgbm.predict(X\_valid)

lgbm\_accuracy += accuracy\_score(lgbm\_pred,y\_valid) / 5

print("{}번째 교차검증 accuracy : {:.2f}%".format(i, accuracy\_score(lgbm\_pred,y\_valid) \* 100))

i += 1

print("lgboost 5-교차검증 accuracy : {:.2f}%".format(lgbm\_accuracy \* 100))

결과 비교

# 딥러닝 - 예측

# 회귀

dl\_reg1\_rmse = model1.evaluate(test\_regression\_X, test\_regression\_y)

dl\_reg2\_rmse = model2.evaluate(test\_regression\_X, test\_regression\_y)

dl\_reg3\_rmse = model3.evaluate(test\_regression\_X, test\_regression\_y)

# 분류

dl\_cl2\_acc = model\_cl2.evaluate(test\_classification\_X, test\_dl\_classification\_y)[1]

dl\_cl3\_acc = model\_cl3.evaluate(test\_classification\_X, test\_dl\_classification\_y)[1]

# 머신러닝 - 예측

linear\_pred = linear.predict(scaler\_test\_regression)

ridge\_pred = ridge.predict(scaler\_test\_regression)

lasso\_pred = lasso.predict(scaler\_test\_regression)

rf\_pred = rf.predict(test\_classification)

lgbm\_pred = lgbm.predict(test\_classification)

# 회귀

linear\_rmse = np.sqrt(mean\_squared\_error(linear\_pred, test\_regression\_y))

ridge\_rmse = np.sqrt(mean\_squared\_error(ridge\_pred, test\_regression\_y))

lasso\_rmse = np.sqrt(mean\_squared\_error(lasso\_pred, test\_regression\_y))

# 분류

rf\_acc = accuracy\_score(rf\_pred, test\_classification\_y)

lgbm\_acc = accuracy\_score(lgbm\_pred, test\_classification\_y)

# figure 26

sns.barplot(x=['linear\_rmse', 'ridge\_rmse', 'lasso\_rmse', 'dl\_reg1\_pred', 'dl\_reg2\_pred', 'dl\_reg3\_pred'], y=[linear\_rmse, ridge\_rmse, lasso\_rmse, dl\_reg1\_pred, dl\_reg2\_pred, dl\_reg3\_pred])

plt.xticks(rotation=45)

# figure 27

sns.barplot(x=['rf\_acc', 'lgbm\_acc', 'dl\_cl2\_acc', 'dl\_cl3\_acc'], y=[rf\_acc, lgbm\_acc, dl\_cl2\_acc, dl\_cl3\_acc])

plt.xticks(rotation=45)

# **역할**

| 학번 | 이름 | 역할 |
| --- | --- | --- |
| 20171450 | 이정진 | 데이터 탐색, 시각화, 전처리, 모델링, 보고서 작성 |
| 20171433 | 박우빈 | 데이터 탐색, 시각화, 전처리, 모델링, 보고서 작성 |
| 20171453 | 장재혁 | 데이터 탐색, 시각화, 전처리, 모델링, 보고서 작성 |