

부동산 시장과 공유 경제의 상호작용에 대한 이해

진행기간 : 2023.10.31 ~ 2024.01.19

사용 데이터 : New York Airbnb Data, The Price Of A New York House

사용 언어 : Python

분석 목적 : 공유 경제와 부동산 시장의 상호작용을 알아보고 가격을 결정짓는 요인을 분석해 비교 한다.

목차

1. 소개

1.1 Airbnb 소개

1.2 핵심 전략

1.3 분석 목적

2. 데이터 출처

2.1 Inside Airbnb

2.2 Kaggle

3. Airbnb 가격 분석

3.1 문제 제기

3.2 타깃 변수 설정

3.3 데이터 처리

3.3.1 데이터 불러오기

3.3.2 구간 변수 데이터 처리

3.3.3 범주형 변수 데이터 처리

3.3.4 구간 변수 이상값 제거

3.4 탐색적 자료 분석 및 시각화

3.4.1 구간 변수 요약 통계 검토

3.4.2 구간 변수 상관 관계 검토

3.4.3 구간 변수 기초 통계량 및 시각화

3.4.4. 범주형 변수 시각화

3.4.5 구간 변수 t-검정

3.5 머신러닝 모델 수립

3.6 머신러닝 모델 실행

3.6.1 결정 트리 분류 모델

3.6.2 로지스틱 회귀 분류 모델

3.6.3 표준화한 로지스틱 회귀 모델

3.6.4 사이킷런 신경망 분류 모델

3.6.5 최근접 이웃 분류 모델(KNN)

3.6.6 랜덤 포레스트

3.6.7 그레이디언트 부스팅 모델

3.6.8 라쏘 모델

3.6.9 텐서플로우 케라스 신경망 모델

3.6.10 서포트 백터 머신

3.6.11 회귀, 릿지

3.6.12 XGBoost

3.6.13 LightGBM

3.6.14 스태킹 모델(앙상블)

3.7 모델 정확도 비교 및 최적 파라미터

3.7.1 머신러닝 모델 정확도 및 최적 파라미터

3.7.2 이진 분류 머신러닝 모델 정확도 및 최적 파라미터

3.7.3 연속형 회귀 머신러닝 모델 r2 score

4. 부동산 가격 분석

4.1 문제 제기

4.2 타깃 변수 설정

4.3 데이터 처리

4.3.1 데이터 불러오기

4.3.2 구간 변수 데이터 처리

4.3.3 구간 변수 이상값 제거

4.4 탐색적 자료 분석 및 시각화

4.4.1 구간 변수 요약 통계 검토

4.4.2 구간 변수 상관 관계 검토

4.4.3 구간 변수 기초 통계량 및 시각화

4.4.4. 범주형 변수 시각화

4.4.5 구간 변수 t-검정

4.5 머신러닝 모델 수립

4.6 머신러닝 모델 실행

4.6.1 결정 트리 분류 모델

4.6.2 로지스틱 회귀 분류 모델

4.6.3 표준화한 로지스틱 회귀 모델

4.6.4 사이킷런 신경망 분류 모델

4.6.5 최근접 이웃 분류 모델(KNN)

4.6.6 랜덤 포레스트

4.6.7 그레이디언트 부스팅 모델

4.6.8 라쏘 모델

4.6.9 텐서플로우 케라스 신경망 모델

4.6.10 서포트 벡터 머신

4.6.11 회귀, 릿지

4.6.12 XGBoost

4.6.13 LightGBM

4.6.14 스태킹 모델(앙상블)

4.7 모델 정확도 비교 및 최적 파라미터

4.7.1 머신러닝 모델 정확도 및 최적 파라미터

4.7.2 이진 분류 머신러닝 모델 정확도 및 최적 파라미터

4.7.3 연속형 회귀 머신러닝 모델 r2 score

5. 결론

1. 소개

1.1 Airbnb 소개

Airbnb는 2008년 8월 시작된 세계 최대의 숙박 공유 서비스이다. 자신의 방이나 집, 별장 등 사람이 지낼 수 있는 모든 공간을 임대할 수 있다. 현재 약 220여 개국, 10만여 도시, 400만 호스트 7백만의 숙소를 보유하고 있다.

1.2 핵심 전략

Airbnb는 네트워크 효과를 기반으로 규모를 확장한다. 인터넷을 통해 숙박 공유에 참여하는 사용자 수가 증가할수록 가능한 연결의 수가 기하급수적으로 증가한다. 또한 Airbnb는 보다 광범위하고 다양한 품종의 숙소 및 서비스를 제공한다. 숙소는 전 세계의 여러 국가 및 도시에 분포하고 있으며 다양한 공간이 있으며 호스트 별로 제공하는 서비스가 차별화된다. Airbnb는 기존의 숙박업에서 보이는 소품종 대량생산과 달리 다품종의 숙박시설과 서비스를 제공한다.

1.3 분석 목적

부동산 시장과 공유 경제의 상호작용에 대한 이해를 통해 지역별 부동산 투자 결정에 대한 통찰력 제공한다. Airbnb의 존재가 지역 부동산 시장에 미치는 영향을 연구하여 정보와 수요에 따른 가격 변동을 이해하고, 특정 지역에서의 관광업계와 부동산 시장의 형태 비교 분석한다.

2. 데이터 출처

2.1 Inside Airbnb

Airbnb의 가격을 분석하기 위해 Inside Airbnb에서 데이터를 수집했다. Inside Airbnb는 Murray Cox가 2016년에 설립했다. Inside Airbnb는 관광객에게 주거용 주택을 임대하는 역할을 이해, 결정 및 제어할 수 있는 데이터와 정보를 제공한다. Airbnb의 공식 웹사이트에서 자료가 추출되었기 때문에 이 프로젝트에 적합한 자료이다.

홈페이지 주소 : <http://insideairbnb.com>

2.2 Kaggle

New York의 부동산 가격을 분석하기 위해 Kaggle에서 데이터를 수집했다. Kaggle은 데이터 분석 및 머신러닝에 대한 학습 플랫폼이다. Kaggle에서 12개월 동안 New York에서 판매된 부동산 거래의 기록은 모아둔 데이터를 사용했다.

홈페이지 주소 : <https://www.kaggle.com/datasets/new-york-city/nyc-property-sales#>

3. Airbnb 가격 분석

3.1 문제 제기

Airbnb의 가격을 분석하기 위해 연구 주제를 다음과 같이 설정했다.

- Airbnb 가격에 영향을 미치는 요인은 무엇인가?

3.2 타깃 변수 설정

이 프로젝트에서 연속형 변수와 범주형 변수를 구분하여 분석하기 위해 2개의 타깃 변수를 생성했다. 먼저 연속형 변수의 경우 숙소 가격인 price로 설정했다. 범주형 변수의 경우 이진 값을 가지고 있는 price_B로 설정했다. Price_B는 원본 데이터세트에는 없던 변수로 기존에 있던 연속형 변수인 price에 기반하여 숙소 가격 변수값이 Airbnb 숙소 가격의 중위수 이상이면 price_B는 1, 중위수 미만이면 0으로 값을 부여했다.

파이썬 코드

```
#price 변수 object에서 float로 변환
delete_columns_data['price'] =
delete_columns_data['price'].str.replace('[$,]', '',
regex=True).astype(float)

delete_columns_data.info()

# 타겟 변수 값이 중위수 이상이면 1, 아니면 0
# 이진값 타겟 변수 price_B
c1 = delete_columns_data['price'] >=
delete_columns_data['price'].median()

c0 = delete_columns_data['price'] <
delete_columns_data['price'].median()

delete_columns_data.loc[c1, "price_B"] = 1
delete_columns_data.loc[c0, "price_B"] = 0

delete_columns_data.head(3)
```

3.3 데이터 처리

Airbnb 숙소에 관련된 Raw data는 총 38792개의 행, 75개의 열로 구성되어 있다. 이 중에서 listing_url, scrape_id 등 가격에 영향을 주지 않을 것이라고 판단되는 49개의 변수를 제거했다. 그리고 주어진 데이터 셋에서 다음과 같이 파생 변수를 생성했다.

[bathroom_text] : 통일되지 않은 값을 삭제하고 변수의 종류 통일

[amenities] : 숙소에서 제공하는 편의 시설로 나열되어 있는 문자를 편의 시설의 개수를 수치화하여 사용

파생 변수 생성 후 최종적인 데이터는 38750개의 행, 26개의 열로 구성되어 있다.

- 최종 변수 정리

변수명	타입	변수 설명
id	int64	임대 공간 id 변수
host_response_time	object	host의 답장 속도
host_response_rate	float64	host의 답장 비율
host_is_superhost	object	superhost 여부
neighbourhood_group_cleansed	object	임대 공간 지역
latitude	float64	임대 공간 위도
longitude	float64	임대 공간 경도
room_type	object	방 종류
accommodates	int64	수용 인원
bathrooms_text	object	화장실 종류
bedrooms	float64	침실 수
beds	float64	침대 수
amenities	int64	편의시설 수
price	float64	임대 공간 가격
availability_90	int64	연중 이용 가능 일수(최대 90일)
availability_365	int64	연중 이용 가능 일수(최대 365일)
number_of_reviews	int64	후기 개수
review_scores_rating	float64	전체 평점
review_scores_accuracy	float64	정확도
review_scores_cleanliness	float64	청결도
review_scores_checkin	float64	체크인
review_scores_communication	float64	의사소통
review_scores_location	float64	위치
review_scores_value	float64	가격 대비 만족도

instant_bookable	object	즉시 예약 가능 여부
price_B	object	임대 공간 가격 이진값

3.3.1 데이터 불러오기

```

import pandas as pd

import numpy as np

source_data = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/detailed_listings.csv')

source_data.shape

delete_columns_data = source_data.drop(labels=['listing_url',
'scrape_id', 'last_scraped', 'source', 'name', 'description',
'neighborhood_overview', 'picture_url',
                           'host_id', 'host_url',
'host_name', 'host_since', 'host_location', 'host_about',
'host_acceptance_rate', 'host_thumbnail_url', 'host_picture_url',
                           'host_neighbourhood',
'host_listings_count', 'host_total_listings_count',
'host_verifications', 'host_has_profile_pic', 'host_identity_verified',
                           'neighbourhood',
'neighbourhood_cleansed', 'property_type', 'bathrooms',
'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
                           'maximum_minimum_nights',
', 'minimum_maximum_nights', 'maximum_maximum_nights',
'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'calendar_updated',
                           'has_availability',
'availability_30', 'availability_60', 'calendar_last_scraped',
'number_of_reviews_ltm', 'number_of_reviews_130d', 'first_review',
                           'last_review',
'license', 'calculated_host_listings_count',
'calculated_host_listings_count_entire_homes',
'calculated_host_listings_count_private_rooms',
                           'calculated_host_listings_count_shared_rooms', 'reviews_per_month'], axis = 1)
#사용하지 않는 열 제거

delete_columns_data.shape

delete_columns_data.head(3)

delete_columns_data.info()

#ID 변수인 id 값의 중복 여부 체크

n = len(pd.unique(delete_columns_data['id']))

print(n)

```

3.3.2 구간 변수 데이터 처리

```
#구간 변수 : host_response_rate, latitude , longitude , accommodates,
bedrooms, beds, availability_90, availability_365, number_of_reviews,
amenities, review_scores_rating, review_scores_accuracy,
review_scores_cleanliness, review_scores_checkin,
review_scores_communication, review_scores_location,
review_scores_value (17 개)

#host_response_rate 을 numeric 으로 변환
delete_columns_data['host_response_rate'] =
delete_columns_data['host_response_rate'].str.replace('%',
'').astype(float)

#amenities 변수를 개수로 변환
delete_columns_data['amenities'] =
delete_columns_data['amenities'].apply(lambda x: x.count(','))

delete_columns_data['amenities'].astype(int)

delete_columns_data['amenities'] = delete_columns_data['amenities'] + 1

delete_columns_data.info()

delete_columns_data.head(3)

#구간 변수 결측값 평균으로 대체

#host_response_rate
mean_host_response_rate =
delete_columns_data['host_response_rate'].mean()

delete_columns_data['host_response_rate'].fillna(mean_host_response_rate,
inplace=True)

#bedrooms
mean_bedrooms = delete_columns_data['bedrooms'].mean()

delete_columns_data['bedrooms'].fillna(mean_bedrooms, inplace=True)

#beds
mean_beds = delete_columns_data['beds'].mean()

delete_columns_data['beds'].fillna(mean_beds, inplace=True)

#beds
mean_beds = delete_columns_data['beds'].mean()

delete_columns_data['beds'].fillna(mean_beds, inplace=True)
```

```
#review_scores_rating
mean_review_scores_rating =
delete_columns_data['review_scores_rating'].mean()

delete_columns_data['review_scores_rating'].fillna(mean_review_scores_rating, inplace=True)

#review_scores_accuracy
mean_review_scores_accuracy =
delete_columns_data['review_scores_accuracy'].mean()

delete_columns_data['review_scores_accuracy'].fillna(mean_review_scores_accuracy, inplace=True)

#review_scores_cleanliness
mean_review_scores_cleanliness =
delete_columns_data['review_scores_cleanliness'].mean()

delete_columns_data['review_scores_cleanliness'].fillna(mean_review_scores_cleanliness, inplace=True)

#review_scores_checkin
mean_review_scores_checkin =
delete_columns_data['review_scores_checkin'].mean()

delete_columns_data['review_scores_checkin'].fillna(mean_review_scores_checkin, inplace=True)

#review_scores_communication
mean_review_scores_communication =
delete_columns_data['review_scores_communication'].mean()

delete_columns_data['review_scores_communication'].fillna(mean_review_scores_communication, inplace=True)

#review_scores_location
mean_review_scores_location =
delete_columns_data['review_scores_location'].mean()

delete_columns_data['review_scores_location'].fillna(mean_review_scores_location, inplace=True)

#review_scores_value
mean_review_scores_value =
delete_columns_data['review_scores_value'].mean()

delete_columns_data['review_scores_value'].fillna(mean_review_scores_value, inplace=True)

delete_columns_data.info()
```

3.3.3 범주형 변수 데이터 처리

```
#범주형 변수 : host_response_time, host_is_superhost,
neighbourhood_group_cleansed, room_type, bathrooms_text,
instant_bookable, price_B (7 개)

#host_is_superhost
delete_columns_data['host_is_superhost'].fillna('f', inplace=True)

#bathrooms_text
delete_columns_data =
delete_columns_data.dropna(subset=['bathrooms_text'])

#host_response_time
delete_columns_data['host_response_time'].fillna('a few days or more',
inplace=True)

unique_values = delete_columns_data['bathrooms_text'].unique()

print(unique_values)

delete_columns_data['bathrooms_text'] =
delete_columns_data['bathrooms_text'].str.replace('1 private bath', '1
bath')

delete_columns_data['bathrooms_text'] =
delete_columns_data['bathrooms_text'].str.replace('Shared half-bath',
'0.5 shared bath')

delete_columns_data['bathrooms_text'] =
delete_columns_data['bathrooms_text'].str.replace('Half-bath', '0.5
bath')

delete_columns_data['bathrooms_text'] =
delete_columns_data['bathrooms_text'].str.replace('Private half-bath',
'0.5 bath')

unique_values = delete_columns_data['bathrooms_text'].unique()

print(unique_values)

delete_columns_data.info()
```

3.3.4 구간 변수 이상값 제거

```
# 구간변수 이상값 확인 및 제거

import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'host_response_rate', data =
delete_columns_data)

sns.boxplot(ax = axes[1], x = 'latitude', data = delete_columns_data)

sns.boxplot(ax = axes[2], x = 'longitude', data = delete_columns_data)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'accommodates', data =
delete_columns_data)

sns.boxplot(ax = axes[1], x = 'bedrooms', data = delete_columns_data)

sns.boxplot(ax = axes[2], x = 'beds', data = delete_columns_data)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'availability_90', data =
delete_columns_data)

sns.boxplot(ax = axes[1], x = 'availability_365', data =
delete_columns_data)

sns.boxplot(ax = axes[2], x = 'number_of_reviews', data =
delete_columns_data)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'amenities', data = delete_columns_data)

sns.boxplot(ax = axes[1], x = 'review_scores_rating', data =
delete_columns_data)

sns.boxplot(ax = axes[2], x = 'review_scores_accuracy', data =
delete_columns_data)
```

```

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'review_scores_cleanliness', data =
delete_columns_data)

sns.boxplot(ax = axes[1], x = 'review_scores_checkin', data =
delete_columns_data)

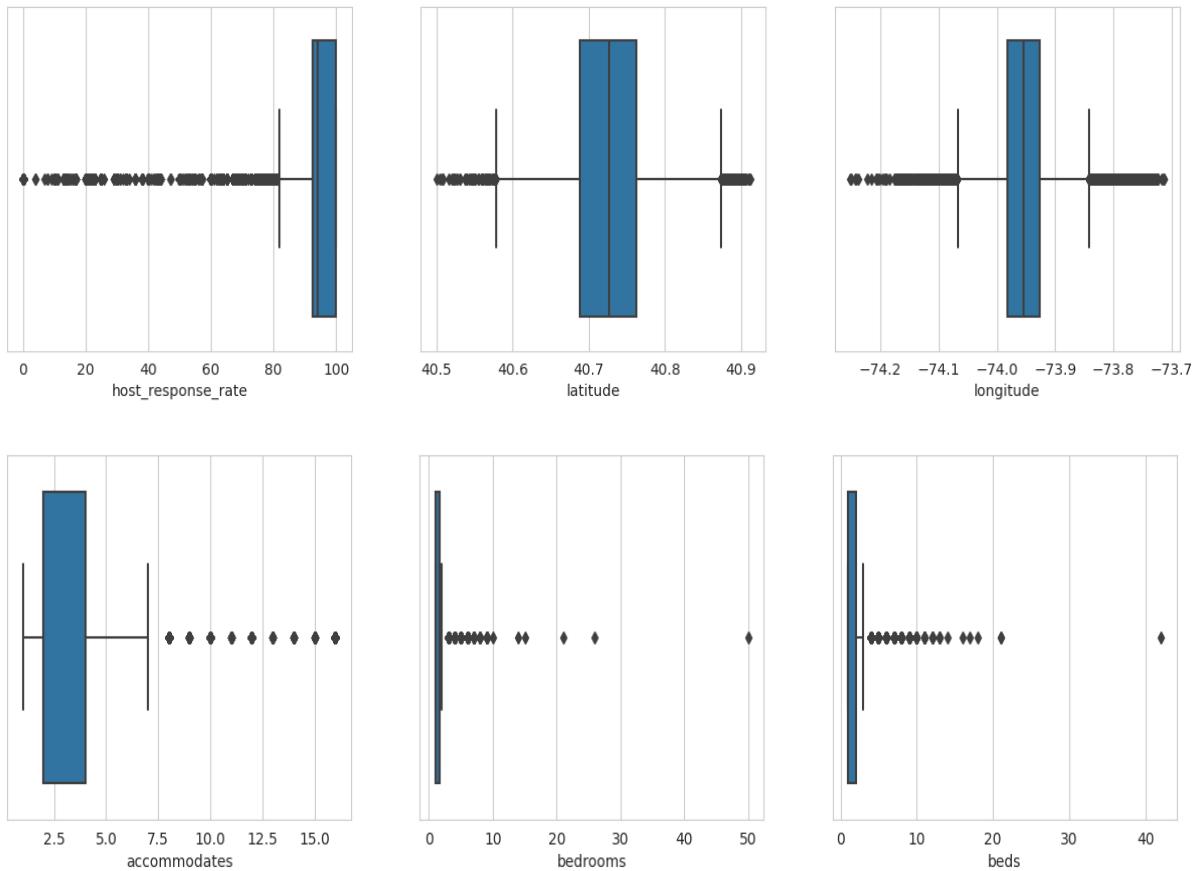
sns.boxplot(ax = axes[2], x = 'review_scores_communication', data =
delete_columns_data)

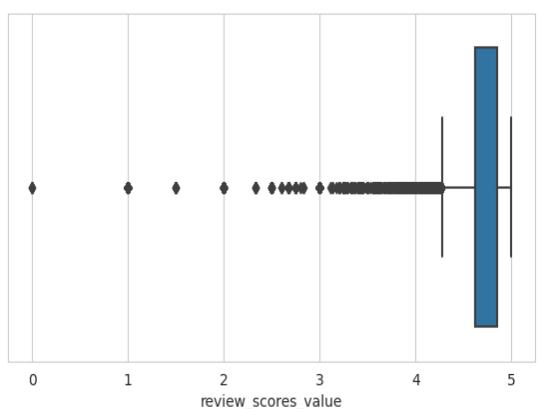
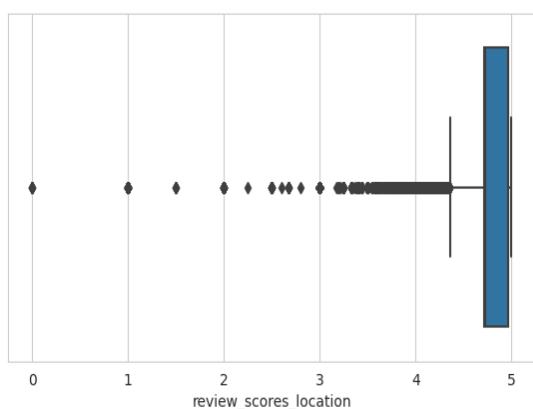
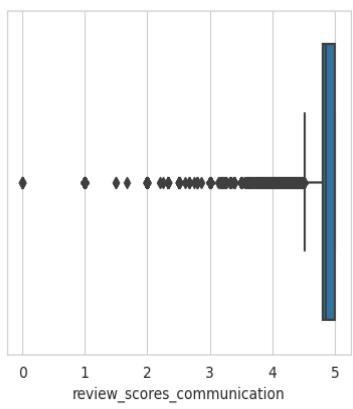
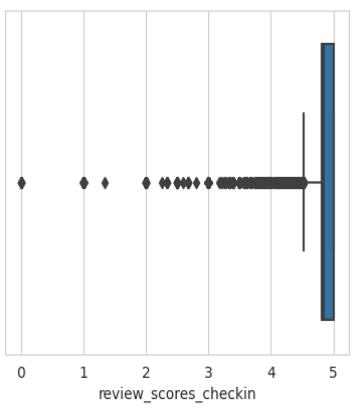
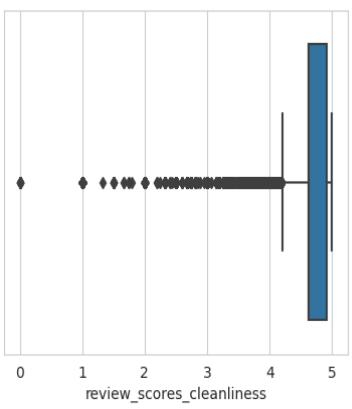
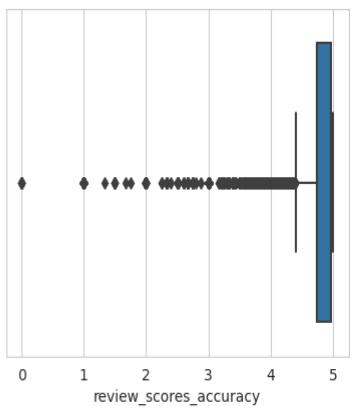
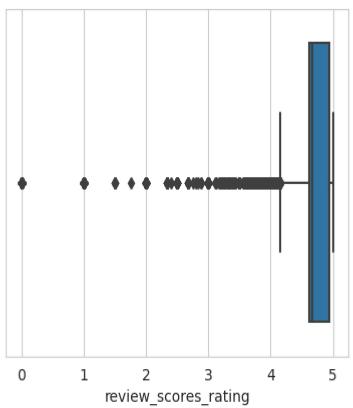
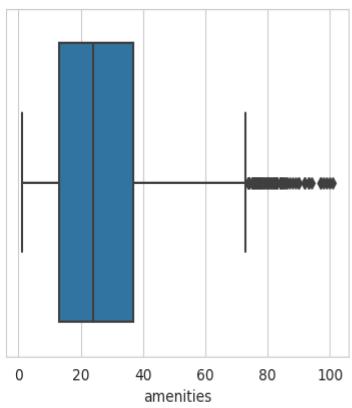
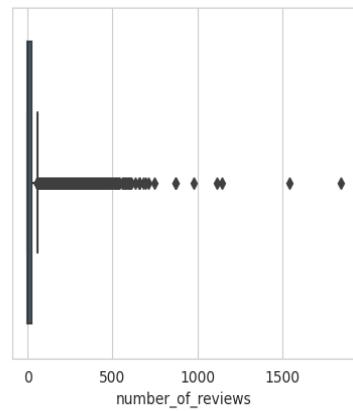
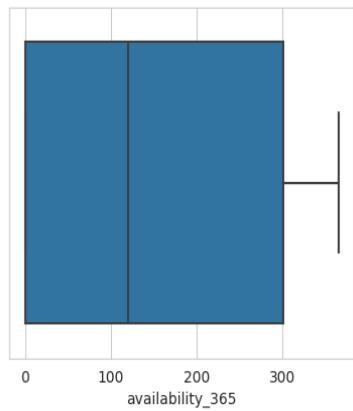
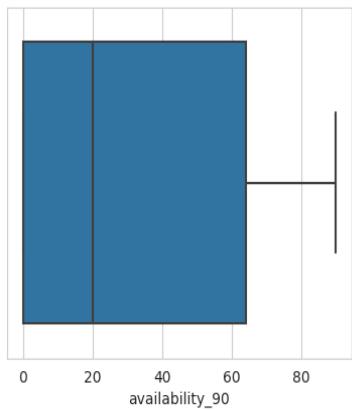
fig, axes = plt.subplots(1, 2, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'review_scores_location', data =
delete_columns_data)

sns.boxplot(ax = axes[1], x = 'review_scores_value', data =
delete_columns_data)

```





```
#host_response_rate, accommodates, bedrooms, beds 의 이상치를 하한값과  
상한값으로 대체  
#나머지 변수들은 이상치 처리 x  
  
Q1 = delete_columns_data[['host_response_rate', 'accommodates',  
'bedrooms', 'beds']].quantile(0.25)  
  
Q3 = delete_columns_data[['host_response_rate', 'accommodates',  
'bedrooms', 'beds']].quantile(0.75)  
  
IQR = Q3 - Q1  
  
print(IQR)  
  
Lower = Q1 - 1.5 * IQR  
  
Upper = Q3 + 1.5 * IQR  
  
print(Lower)  
  
print(Upper)  
  
delete_columns_data['host_response_rate'] =  
delete_columns_data['host_response_rate'].apply(lambda x: 82 if x < 82  
else x)  
  
delete_columns_data['accommodates'] =  
delete_columns_data['accommodates'].apply(lambda x: 7 if x > 7 else x)  
  
delete_columns_data['bedrooms'] =  
delete_columns_data['bedrooms'].apply(lambda x: 2.478508 if x >  
2.478508 else x)  
  
delete_columns_data['beds'] = delete_columns_data['beds'].apply(lambda  
x: 3.5 if x > 3.5 else x)  
  
delete_columns_data.to_csv('/content/drive/MyDrive/Colab  
Notebooks/Data_Analysis/preprocessing_completed.csv', index = False)
```

3.4 탐색적 자료 분석 및 시각화

3.4.1 구간 변수 요약 통계 검토

```
import pandas as pd

# 결측값이 50% 초과인 변수는 전처리 과정에서 수행했으므로 미조치
preprocessing_completed = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/preprocessing_completed.csv')

preprocessing_completed['price_B'] =
preprocessing_completed['price_B'].astype('object')

preprocessing_completed.info()

# 요약 통계가 필요 없는 id, latitude, longitude 변수는 제거하고 검토

interval_variable = preprocessing_completed[['host_response_rate',
'accommodates', 'bedrooms', 'beds', 'amenities', 'availability_90',
'availability_365', 'number_of_reviews', 'review_scores_rating',
'review_scores_accuracy', 'review_scores_cleanliness',
'review_scores_checkin', 'review_scores_communication',
'review_scores_location', 'review_scores_value']]

interval_variable.describe()

# 구간 변수 왜도 확인

interval_variable.skew()

# 로그 변환 및 boxcox 변환

import numpy as np

import pandas as pd

from scipy.stats import boxcox

# 로그 변환 및 boxcox 변환
interval_variable['number_of_reviews'] =
np.log(interval_variable['number_of_reviews'] + 1)

interval_variable['review_scores_rating'] =
pd.Series(boxcox(interval_variable['review_scores_rating'] + 0.1)[0])
```

```
interval_variable['review_scores_accuracy'] =
pd.Series(boxcox(interval_variable['review_scores_accuracy'] + 0.1)[0])

interval_variable['review_scores_cleanliness'] =
pd.Series(boxcox(interval_variable['review_scores_cleanliness'] +
0.1)[0])

interval_variable['review_scores_checkin'] =
pd.Series(boxcox(interval_variable['review_scores_checkin'] + 0.1)[0])

interval_variable['review_scores_communication'] =
pd.Series(boxcox(interval_variable['review_scores_communication'] +
0.1)[0])

interval_variable['review_scores_location'] =
pd.Series(boxcox(interval_variable['review_scores_location'] + 0.1)[0])

interval_variable['review_scores_value'] =
pd.Series(boxcox(interval_variable['review_scores_value'] + 0.1)[0])

# skewness 확인

interval_variable.skew()

interval_variable.kurtosis()
```

3.4.2 구간 변수 상관 관계 검토

```
round(interval_variable.corr(), 2)
```

	host_response_rate	accommodates	bedrooms	beds	amenities	availability_90	availability_365	number_of_reviews
host_response_rate	1.00	0.09	0.06	0.10	0.27	0.11	0.14	0.14
accommodates	0.09	1.00	0.49	0.78	0.23	0.12	0.13	0.08
bedrooms	0.06	0.49	1.00	0.57	0.13	0.09	0.10	0.03
beds	0.10	0.78	0.57	1.00	0.25	0.12	0.14	0.11
amenities	0.27	0.23	0.13	0.25	1.00	0.29	0.31	0.39
availability_90	0.11	0.12	0.09	0.12	0.29	1.00	0.72	0.13
availability_365	0.14	0.13	0.10	0.14	0.31	0.72	1.00	0.11
number_of_reviews	0.14	0.08	0.03	0.11	0.39	0.13	0.11	1.00
review_scores_rating	0.10	0.01	0.00	0.02	0.23	0.01	0.01	0.20
review_scores_accuracy	0.06	-0.02	-0.02	-0.01	0.15	-0.08	-0.08	0.13
review_scores_cleanliness	0.10	0.03	-0.00	0.04	0.23	0.04	0.04	0.17
review_scores_checkin	0.06	-0.02	-0.00	0.00	0.13	-0.07	-0.06	0.12
review_scores_communication	0.09	-0.01	-0.01	0.01	0.15	-0.08	-0.08	0.15
review_scores_location	-0.01	-0.03	-0.07	-0.04	0.02	-0.12	-0.10	0.02
review_scores_value	0.04	-0.03	-0.00	-0.00	0.12	-0.08	-0.10	0.13

review_scores_rating	review_scores_accuracy	review_scores_cleanliness	review_scores_checkin	review_scores_communication	review_scores_location	review_scores_value
0.10	0.06	0.10	0.06	0.09	-0.01	0.04
0.01	-0.02	0.03	-0.02	-0.01	-0.03	-0.03
0.00	-0.02	-0.00	-0.00	-0.01	-0.07	-0.00
0.02	-0.01	0.04	0.00	0.01	-0.04	-0.00
0.23	0.15	0.23	0.13	0.15	0.02	0.12
0.01	-0.08	0.04	-0.07	-0.08	-0.12	-0.08
0.01	-0.08	0.04	-0.06	-0.08	-0.10	-0.10
0.20	0.13	0.17	0.12	0.15	0.02	0.13
1.00	0.73	0.69	0.61	0.66	0.46	0.71
0.73	1.00	0.65	0.63	0.65	0.46	0.70
0.69	0.65	1.00	0.51	0.52	0.37	0.61
0.61	0.63	0.51	1.00	0.68	0.42	0.56
0.66	0.65	0.52	0.68	1.00	0.43	0.60
0.46	0.46	0.37	0.42	0.43	1.00	0.46
0.71	0.70	0.61	0.56	0.60	0.46	1.00

3.4.3 구간 변수 기초 통계량 및 시각화

```
interval_variable = preprocessing_completed[['host_response_rate',
'latitude', 'longitude', 'accommodates', 'bedrooms', 'beds',
'amenities', 'availability_90', 'availability_365',
'number_of_reviews', 'review_scores_rating', 'review_scores_accuracy',
'review_scores_cleanliness', 'review_scores_checkin',
'review_scores_communication', 'review_scores_location',
'review_scores_value']]  
  
interval_variable.describe()
```

	host_response_rate	latitude	longitude	accommodates	bedrooms	beds	amenities	availability_90	availability_365
count	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000
mean	95.077469	40.729383	-73.945808	2.780568	1.516860	1.554757	26.322994	32.874039	148.848284
std	5.415922	0.056735	0.055128	1.624631	0.446453	0.811567	15.690079	34.844344	142.250654
min	82.000000	40.500314	-74.251907	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000
25%	92.557264	40.688790	-73.982800	2.000000	1.000000	1.000000	13.000000	0.000000	0.000000
50%	94.000000	40.726275	-73.953870	2.000000	1.591403	1.000000	24.000000	20.000000	120.000000
75%	100.000000	40.762690	-73.926191	4.000000	1.591403	2.000000	37.000000	64.000000	301.000000
max	100.000000	40.911380	-73.713650	7.000000	2.478508	3.500000	101.000000	90.000000	365.000000

number_of_reviews	review_scores_rating	review_scores_accuracy	review_scores_cleanliness	review_scores_checkin	review_scores_communication	review_scores_location	review_scores_value	price
38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000	38750.000000
25.373135	4.626053	4.740621	4.623132	4.811376	4.805671	4.724157	4.621473	215.959381
55.877476	0.636900	0.412740	0.476852	0.359257	0.385890	0.357005	0.445515	496.167553
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	8.000000
0.000000	4.625896	4.740582	4.622926	4.811373	4.805698	4.720000	4.621431	79.000000
4.000000	4.670000	4.750000	4.622926	4.840000	4.860000	4.724220	4.621431	135.000000
24.000000	4.940000	4.970000	4.910000	5.000000	5.000000	4.960000	4.850000	225.000000
1843.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	30000.000000

시각화 1 – 박스 플롯으로 변수의 분포 확인

```
import matplotlib.pyplot as plt  
  
import seaborn as sns  
  
sns.set_style('whitegrid')  
  
fig, axes = plt.subplots(1, 3, figsize=(15, 4))  
  
sns.boxplot(ax = axes[0], x = 'host_response_rate', data = interval_variable)  
  
sns.boxplot(ax = axes[1], x = 'latitude', data = interval_variable)  
  
sns.boxplot(ax = axes[2], x = 'longitude', data = interval_variable)
```

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'accommodates', data = interval_variable)

sns.boxplot(ax = axes[1], x = 'bedrooms', data = interval_variable)

sns.boxplot(ax = axes[2], x = 'beds', data = interval_variable)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'availability_90', data =
interval_variable)

sns.boxplot(ax = axes[1], x = 'availability_365', data =
interval_variable)

sns.boxplot(ax = axes[2], x = 'number_of_reviews', data =
interval_variable)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'amenities', data = interval_variable)

sns.boxplot(ax = axes[1], x = 'review_scores_rating', data =
interval_variable)

sns.boxplot(ax = axes[2], x = 'review_scores_accuracy', data =
interval_variable)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'review_scores_cleanliness', data =
interval_variable)

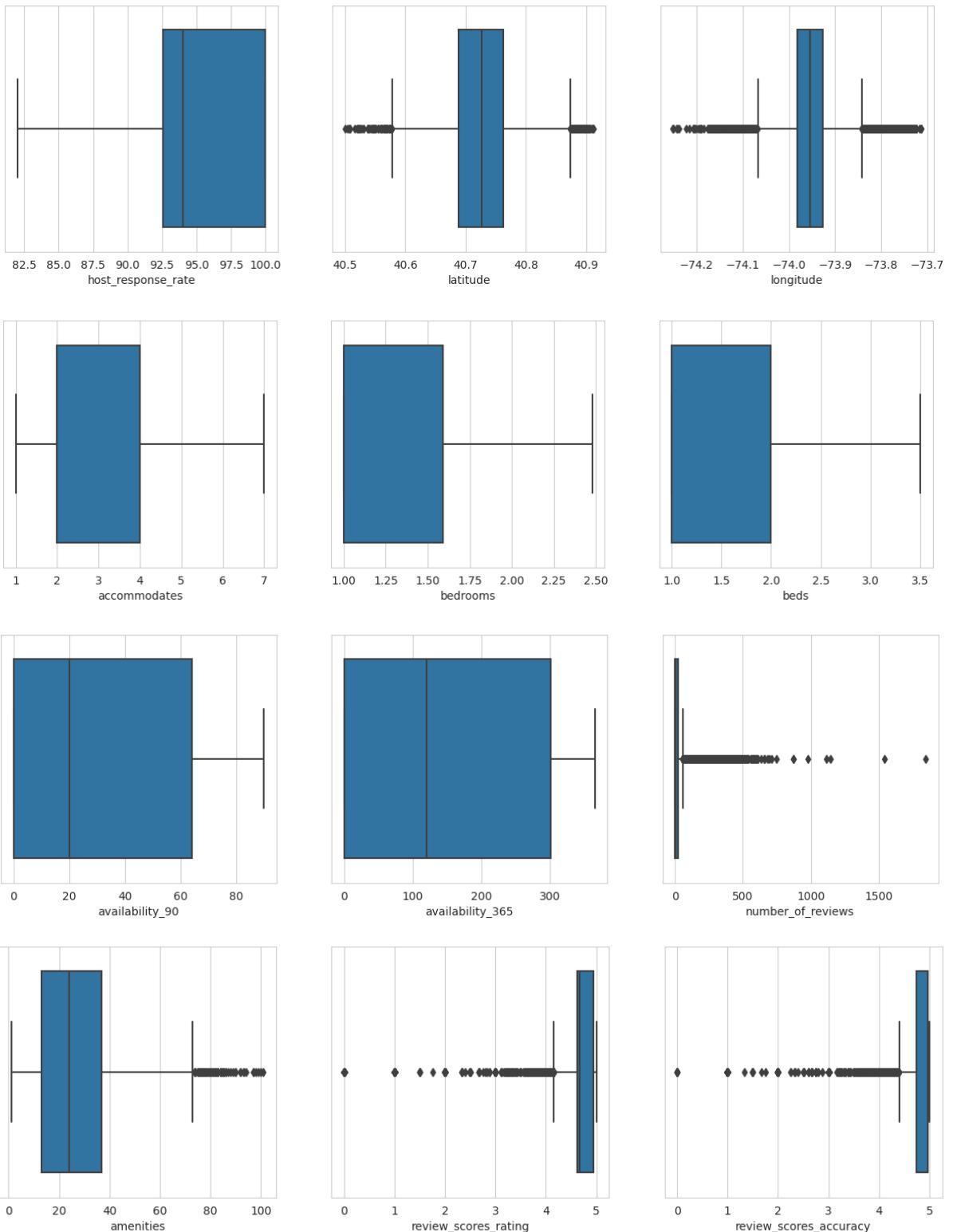
sns.boxplot(ax = axes[1], x = 'review_scores_checkin', data =
interval_variable)

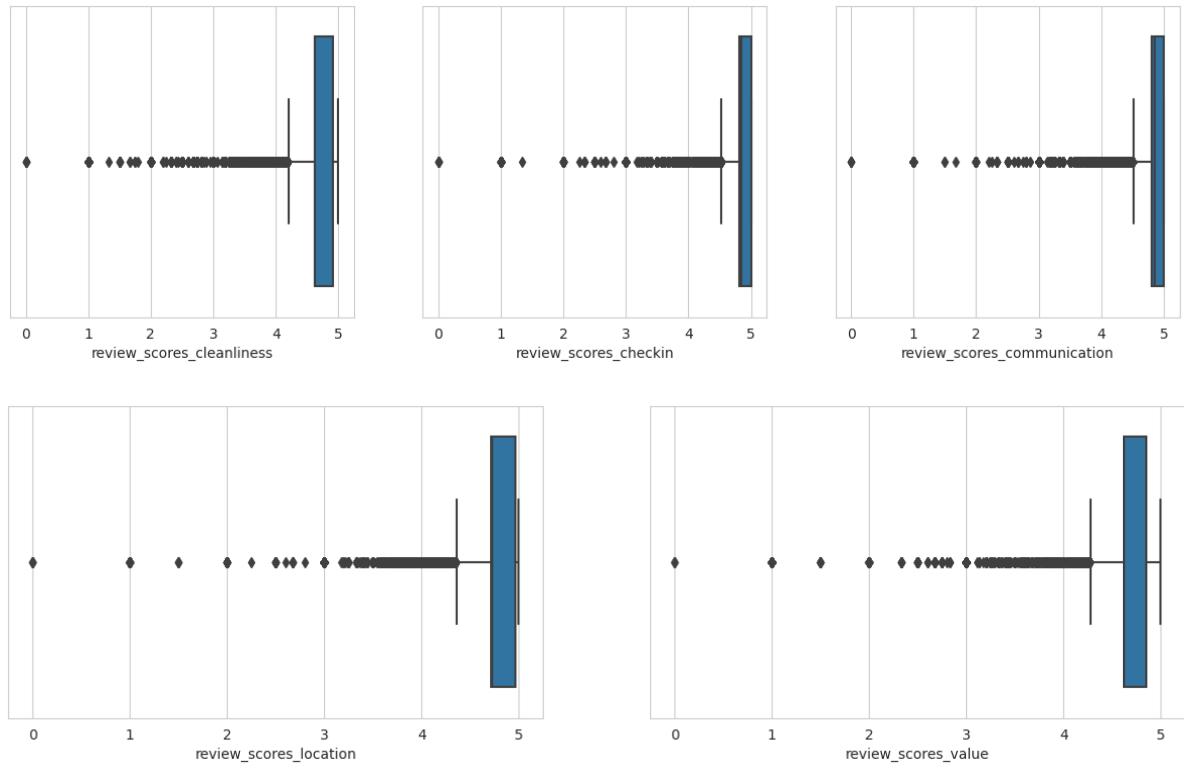
sns.boxplot(ax = axes[2], x = 'review_scores_communication', data =
interval_variable)

fig, axes = plt.subplots(1, 2, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'review_scores_location', data =
interval_variable)

sns.boxplot(ax = axes[1], x = 'review_scores_value', data =
interval_variable)
```





위도, 경도, 리뷰에 관한 속성들은 제거나 변환이 의미가 없다고 판단하여 따로 조치하지 않았다.

시각화 2 – 히트맵으로 상관계수 관계 확인

```
plt.figure(figsize=(20, 7))

sns.heatmap(interval_variable.corr(), annot=True, fmt=".2f",
            cmap="Blues")
```



3.4.4 범주형 변수 기초 통계량 및 시각화

시각화 3 – 막대그래프로 변수 구성 확인

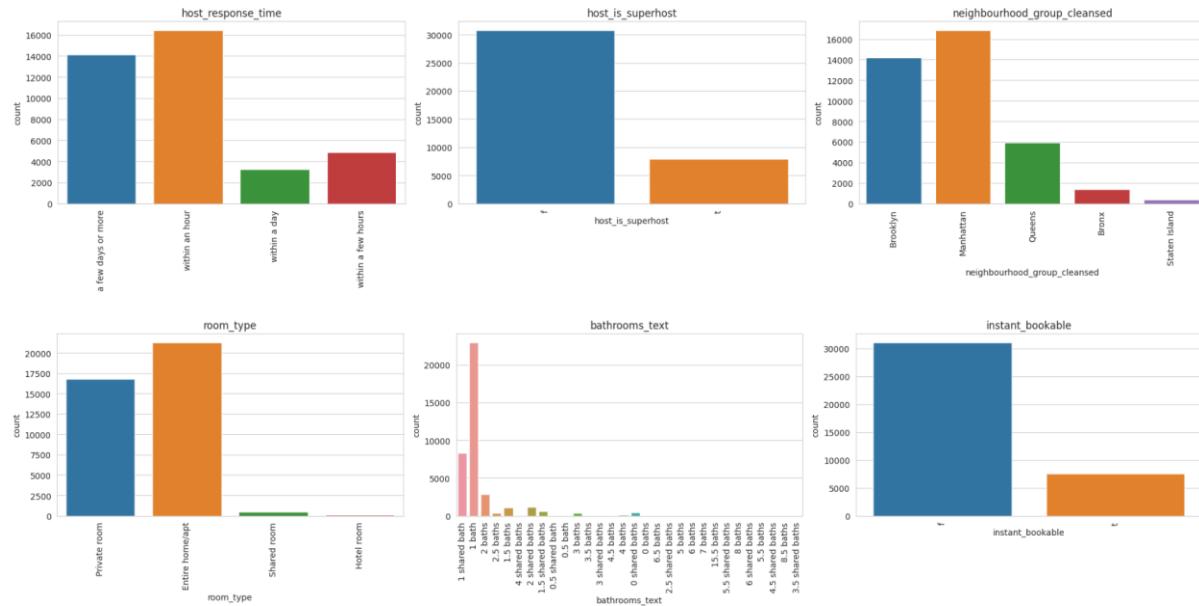
```
categorical_variable = preprocessing_completed[['host_response_time',
'host_is_superhost', 'neighbourhood_group_cleansed', 'room_type',
'bathrooms_text', 'instant_bookable', 'price_B']]

fig, axes = plt.subplots(2, 3, figsize=(20, 10))

for i, col in enumerate(categorical_variable.columns[:-1]):
    sns.countplot(x=col, data=categorical_variable, ax=axes[i//3, i%3])
    axes[i//3, i%3].set_title(col)
    axes[i//3, i%3].tick_params(axis='x', rotation=90)

plt.tight_layout()

plt.show()
```



```

#host_response_time

#frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['host_response_time'],
columns='count'))

print()

#frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['host_response_time'],
categorical_variable['price_B'], normalize=True))

```

col_0	count
host_response_time	
a few days or more	14117
within a day	3267
within a few hours	4900
within an hour	16466

price_B	0.0	1.0
host_response_time		
a few days or more	0.212155	0.152155
within a day	0.042039	0.042271
within a few hours	0.070632	0.055819
within an hour	0.173677	0.251252

```

#host_is_superhost

#frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['host_is_superhost'],
columns='count'))

print()

#frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['host_is_superhost'],
categorical_variable['price_B'], normalize=True))

```

col_0	count
host_is_superhost	
f	30804
t	7946

price_B	0.0	1.0
host_is_superhost		
f	0.404955	0.389987
t	0.093548	0.1111510

```

#neighbourhood_group_cleansed

#frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['neighbourhood_group_cleansed'],
columns='count'))

print()

#frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['neighbourhood_group_cleansed'],
categorical_variable['price_B'], normalize=True))

col_0          count
neighbourhood_group_cleansed
Bronx           1373
Brooklyn        14177
Manhattan       16882
Queens          5946
Staten Island   372

price_B          0.0      1.0
neighbourhood_group_cleansed
Bronx            0.025884  0.009548
Brooklyn         0.212981  0.152877
Manhattan        0.144516  0.291148
Queens           0.108387  0.045058
Staten Island    0.006735  0.002865

#room_type

#frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['room_type'], columns='count'))

print()

#frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['room_type'],
categorical_variable['price_B'], normalize=True))

col_0          count
room_type
Entire home/apt 21311
Hotel room      129
Private room    16818
Shared room     492

price_B          0.0      1.0
room_type
Entire home/apt 0.145084  0.404877
Hotel room      0.000000  0.003329
Private room    0.343794  0.090219
Shared room     0.009626  0.003071

```

```

#bathrooms_text

#frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['bathrooms_text'],
columns='count'))

print()

#frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['bathrooms_text'],
categorical_variable['price_B'], normalize=True))

```

col_0	count				
bathrooms_text		price_B	0.0	1.0	
0 baths	59	bathrooms_text			
0 shared baths	446	0 baths	0.000568	0.000955	
0.5 bath	54	0 shared baths	0.009497	0.002013	
0.5 shared bath	13	0.5 bath	0.000723	0.000671	
1 bath	22974	0.5 shared bath	0.000335	0.000000	
1 shared bath	8306	1 bath	0.228439	0.364439	
1.5 baths	1092	1 shared bath	0.189006	0.025342	
1.5 shared baths	613	1.5 baths	0.009342	0.018839	
15.5 baths	1	1.5 shared baths	0.014090	0.001729	
2 baths	2830	15.5 baths	0.000000	0.000026	
2 shared baths	1144	2 baths	0.011819	0.061213	
2.5 baths	367	2 shared baths	0.027768	0.001755	
2.5 shared baths	72	2.5 baths	0.000774	0.008697	
3 baths	382	2.5 shared baths	0.001755	0.000103	
3 shared baths	83	3 baths	0.001110	0.008748	
3.5 baths	79	3 shared baths	0.002065	0.000077	
3.5 shared baths	1	3.5 baths	0.000026	0.002013	
4 baths	133	3.5 shared baths	0.000026	0.000000	
4 shared baths	30	4 baths	0.000387	0.003045	
4.5 baths	36	4 shared baths	0.000748	0.000026	
4.5 shared baths	2	4.5 baths	0.000000	0.000929	
5 baths	14	4.5 shared baths	0.000026	0.000026	
5.5 baths	3	5 baths	0.000000	0.000361	
5.5 shared baths	1	5.5 baths	0.000000	0.000077	
6 baths	6	5.5 shared baths	0.000000	0.000026	
6 shared baths	1	6 baths	0.000000	0.000155	
6.5 baths	5	6 shared baths	0.000000	0.000026	
7 baths	1	6.5 baths	0.000000	0.000129	
8 baths	1	7 baths	0.000000	0.000026	
8.5 baths	1	8 baths	0.000000	0.000026	
		8.5 baths	0.000000	0.000026	

```
#instant_bookable

#frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['instant_bookable'],
columns='count'))

print()

#frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['instant_bookable'],
categorical_variable['price_B'], normalize=True))
```

col_0	count
instant_bookable	
f	31128
t	7622

price_B	0.0	1.0
instant_bookable		
f	0.422865	0.380439
t	0.075639	0.121058

3.4.5 구간 변수 t-검정

- host_response_rate

```
from scipy import stats

# host_response_rate t-검정

data_host_response_rate_1 =
interval_variable[categorical_variable['price_B'] ==
1]['host_response_rate']

data_host_response_rate_0 =
interval_variable[categorical_variable['price_B'] ==
0]['host_response_rate']

stats.ttest_ind(data_host_response_rate_1, data_host_response_rate_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 host_response_rate 는 서로 다르다

- accommodates

```
# accommodates t-검정

data_accommodates_1 = interval_variable[categorical_variable['price_B'] ==
1]['accommodates']

data_accommodates_0 = interval_variable[categorical_variable['price_B'] ==
0]['accommodates']

stats.ttest_ind(data_accommodates_1, data_accommodates_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 accommodates 는 서로 다르다

- bedrooms

```
# bedrooms t-검정

data_bedrooms_1 = interval_variable[categorical_variable['price_B'] == 1]['bedrooms']

data_bedrooms_0 = interval_variable[categorical_variable['price_B'] == 0]['bedrooms']

stats.ttest_ind(data_bedrooms_1, data_bedrooms_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각
두 그룹의 bedrooms 는 서로 다르다

- beds

```
# beds t-검정

data_beds_1 = interval_variable[categorical_variable['price_B'] == 1]['beds']

data_beds_0 = interval_variable[categorical_variable['price_B'] == 0]['beds']

stats.ttest_ind(data_beds_1, data_beds_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각
두 그룹의 beds 는 서로 다르다

- amenities

```
# amenities t-검정

data_amenities_1 = interval_variable[categorical_variable['price_B'] == 1]['amenities']

data_amenities_0 = interval_variable[categorical_variable['price_B'] == 0]['amenities']

stats.ttest_ind(data_amenities_1, data_amenities_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각
두 그룹의 amenities 는 서로 다르다

- availability_90

```
# availability_90 t-검정

data_availability_90_1 =
interval_variable[categorical_variable['price_B'] ==
1]['availability_90']

data_availability_90_0 =
interval_variable[categorical_variable['price_B'] ==
0]['availability_90']

stats.ttest_ind(data_availability_90_1, data_availability_90_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 availability_90 은 서로 다르다

- availability_365

```
# availability_365 t-검정

data_availability_365_1 =
interval_variable[categorical_variable['price_B'] ==
1]['availability_365']

data_availability_365_0 =
interval_variable[categorical_variable['price_B'] ==
0]['availability_365']

stats.ttest_ind(data_availability_365_1, data_availability_365_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 availability_365 는 서로 다르다

- number_of_reviews

```
# number_of_reviews t-검정

data_number_of_reviews_1 =
interval_variable[categorical_variable['price_B'] ==
1]['number_of_reviews']

data_number_of_reviews_0 =
interval_variable[categorical_variable['price_B'] ==
0]['number_of_reviews']

stats.ttest_ind(data_number_of_reviews_1, data_number_of_reviews_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 number_of_reviews 는 서로 다르다

- reviews_scores_rating

```
# review_scores_rating t-검정

data_review_scores_rating_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_rating']

data_review_scores_rating_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_rating']

stats.ttest_ind(data_review_scores_rating_1,
data_review_scores_rating_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 review_scores_rating 은 서로 다르다

- reviews_scores_accuracy

```
# review_scores_accuracy t-검정

data_review_scores_accuracy_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_accuracy']

data_review_scores_accuracy_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_accuracy']

stats.ttest_ind(data_review_scores_accuracy_1,
data_review_scores_accuracy_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 review_scores_accuracy 는 서로 다르다

- review_scores_cleanliness

```
# review_scores_cleanliness t-검정

data_review_scores_cleanliness_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_cleanliness']

data_review_scores_cleanliness_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_cleanliness']

stats.ttest_ind(data_review_scores_cleanliness_1,
data_review_scores_cleanliness_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 review_scores_cleanliness 는 서로 다르다

- review_scores_checkin

```
# review_scores_checkin t-검정

data_review_scores_checkin_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_checkin']

data_review_scores_checkin_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_checkin']

stats.ttest_ind(data_review_scores_checkin_1,
data_review_scores_checkin_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 review_scores_checkin 은 서로 다르다

- review_scores_communication

```
# review_scores_communication t-검정

data_review_scores_communication_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_communication']

data_review_scores_communication_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_communication']

stats.ttest_ind(data_review_scores_communication_1,
data_review_scores_communication_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 review_scores_communication 은 서로 다르다

- review_scores_location

```
# review_scores_location t-검정

data_review_scores_location_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_location']

data_review_scores_location_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_location']

stats.ttest_ind(data_review_scores_location_1,
data_review_scores_location_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 review_scores_location 은 서로 다르다

- review_scores_value

```
# review_scores_value t-검정

data_review_scores_value_1 =
interval_variable[categorical_variable['price_B'] ==
1]['review_scores_value']

data_review_scores_value_0 =
interval_variable[categorical_variable['price_B'] ==
0]['review_scores_value']

stats.ttest_ind(data_review_scores_value_1, data_review_scores_value_0)
```

review_scores_value 변수만 귀무가설 채택 - 두 그룹별의 평균 점수는 차이가 없으므로 review_scores_value 는 price_B 변수에 영향을 미칠 가능성은 작다

3.5 머신러닝 모델 수립

결정 트리, 로지스틱 회귀, 신경망 모델, K-최근접 이웃, 랜덤 포레스트, 그레이언트 부스팅, Lasso, SVM, XGBoost, LightBGM, 앙상블 등 여러 모델을 사용하여 가장 높은 정확도를 가지고 있는 모델 판별해서 분석에 적용할 계획이다.

3.6 머신러닝 모델 실행

```
#구간 변수 데이터프레임에 price 변수 추가 후 최종 확인

interval_variable['price'] = preprocessing_completed['price']

print(interval_variable.info())

print(categorical_variable.info())

interval_variable.to_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/interval_variable.csv', index = False)

categorical_variable.to_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/categorical_variable.csv', index = False)
```

```
#변환완료한 연속형 변수와 범주형 변수를 하나로 합치고 저장

import pandas as pd

transformation_completed = pd.concat([interval_variable,
categorical_variable], axis=1)

print(transformation_completed.info())

transformation_completed.head()

transformation_completed.to_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/transformation_completed.csv', index = False)
```

3.6.1 결정 트리 분류 모델

```
!pip install -U imbalanced-learn
```

```
!pip install graphviz
```

```
# OrdinalEncoder 를 import 하여 범주형 데이터 변환

df_decisiontree = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/transformation_completed.csv')

from sklearn.preprocessing import OrdinalEncoder

df_decisiontree['host_response_time_encoded'] =
OrdinalEncoder().fit_transform(df_decisiontree['host_response_time'].va
lues.reshape(-1,1))

df_decisiontree['host_is_superhost_encoded'] =
OrdinalEncoder().fit_transform(df_decisiontree['host_is_superhost'].val
ues.reshape(-1,1))

df_decisiontree['neighbourhood_group_cleansed_encoded'] =
OrdinalEncoder().fit_transform(df_decisiontree['neighbourhood_group_cle
ansed'].values.reshape(-1,1))

df_decisiontree['room_type_encoded'] =
OrdinalEncoder().fit_transform(df_decisiontree['room_type'].values.res
hape(-1,1))

df_decisiontree['bathrooms_text_encoded'] =
OrdinalEncoder().fit_transform(df_decisiontree['bathrooms_text'].values
.reshape(-1,1))

df_decisiontree['instant_bookable_encoded'] =
OrdinalEncoder().fit_transform(df_decisiontree['instant_bookable'].valu
es.reshape(-1,1))

df_decisiontree.head(3)
```

범주형 데이터 변환 결과

변수명	데이터 정의
host_response_time_encoded (host의 답장 비율)	0. a few days or more 1. within a day 2. within a few hours 3. within an hour
host_is_superhost_encoded (superhost 여부)	0. f 1. t
neighbourhood_group_cleansed_encoded (임대 공간 지역)	0. Bronx 1. Brooklyn 2. Manhattan 3. Queens 4. Staten Island
room_type_encoded (방 종류)	0. Entire home/apt 1. Hotel room 2. Private room 3. Shared room
bathrooms_text_encoded (화장실 종류)	0. 0 baths 1. 0 shared baths 2. 0.5 bath 3. 0.5 shared bath 4. 1 bath 5. 1 shared bath 6. 1.5 baths 7. 1.5 shared baths 8. 15.5 baths 9. 2 baths 10. 2 shared baths 11. 2.5 baths 12. 2.5 shared baths 13. 3 baths 14. 3 shared baths 15. 3.5 baths 16. 3.5 shared baths 17. 4 baths 18. 4 shared baths 19. 4.5 baths 20. 4.5 shared baths

	21. 5 baths 22. 5.5 baths 23. 5.5 shared baths 24. 6 baths 25. 6 shared baths 26. 6.5 baths 27. 7 baths 28. 8 baths 29. 8.5 baths
instant_bookable_encoded (즉시 예약 가능 여부)	0. f 1. t

```
# 기존 범주형 변수 열 삭제
```

```
df_decisiontree.drop(['host_response_time', 'host_is_superhost',
'neighbourhood_group_cleansed', 'room_type', 'bathrooms_text',
'instant_bookable'], axis=1, inplace=True)

df_decisiontree.info()
```

```
# 타겟 변수 분할 및 비율 확인
```

```
import graphviz

import pandas as pd

import numpy as np

print("df_decisiontree shape : " + str(df_decisiontree.shape))

data_decisiontree = df_decisiontree.drop(['price', 'price_B'], axis=1)

target_decisiontree = df_decisiontree['price_B']

print("data_decisiontree shape : " + str(data_decisiontree.shape))

print("target_decisiontree shape : " + str(target_decisiontree.shape))

df_decisiontree['price_B'].value_counts(dropna=False, normalize=True)
```

```
# 7:3 비율로 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data_decisiontree,
target_decisiontree, test_size=0.3, random_state=4,
stratify=target_decisiontree)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)

print("y_train shape : ", y_train.shape)

print("y_test shape : ", y_test.shape)

y_train.value_counts(normalize=True)
```

```
# 결정 트리 모델(지니 기준)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

# Classifier로 DecisionTreeClassifier를 지정

tree = DecisionTreeClassifier(random_state=0)

# Classifier를 학습 데이터세트에서 학습시킴

model = tree.fit(x_train, y_train)

# 학습된 Classifier로 테스트데이터세트의 자료를 이용해서 타겟 변수 예측값을 생성

pred = model.predict(x_test)

print("Accuracy on training set : {:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set : ", accuracy_score(y_test, pred))
```

```
GridSearch를 실행하기 전 정확도 : 0.78538
```

```

# 결정 트리 모델(지니 기준)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.model_selection import GridSearchCV

tree = DecisionTreeClassifier(criterion="gini", random_state=0,
max_depth=5)

params = {'criterion':['gini', 'entropy'], 'max_depth':range(1, 21)}
grid_tree = GridSearchCV(
    tree, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1,
verbose=1)

grid_tree.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_tree.best_score_))

print("GridSearchCV best parameter:", (grid_tree.best_params_))

```

```

best_clf = grid_tree.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch를 실행한 후 테스트 데이터셋의 정확도 : 0.81454

bets parameter : {'criterion': 'entropy', 'max_depth': 9}

```

# 최적 모델의 변수 중요도 수치 확인

print("Feature importances:")

print(best_clf.feature_importances_)

```

```

# 변수명을 리스트 형태로 만들기

feature_names = list(data_decisiontree.columns)

# 변수명을 index로 만들고, feature_importances를 매칭해서 나열한 데이터프레임
# 만들기

dft = pd.DataFrame(np.round(best_clf.feature_importances_, 4),
index=feature_names, columns=['Feature_importances'])

# Feature_importances의 값을 내림차순으로 정리

dft1 = dft.sort_values(by='Feature_importances', ascending=False)

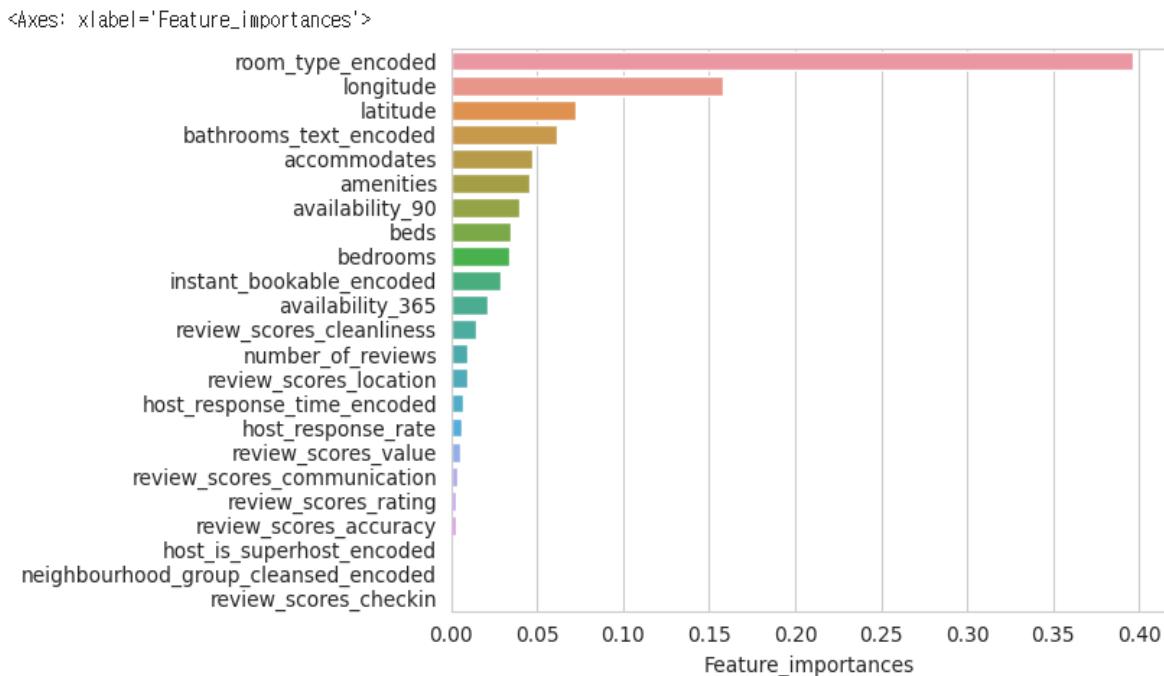
dft1

```

	Feature_importances		
room_type_encoded	0.3965		
longitude	0.1581		
latitude	0.0723		
bathrooms_text_encoded	0.0614		
accommodates	0.0469		
amenities	0.0455		
availability_90	0.0396		
beds	0.0345		
bedrooms	0.0334		
instant_bookable_encoded	0.0288		
availability_365	0.0211		
review_scores_cleanliness	0.0141		
number_of_reviews	0.0093		
review_scores_location	0.0092		
host_response_time_encoded	0.0073		
host_response_rate	0.0058		
review_scores_value	0.0055		
review_scores_communication	0.0038		
review_scores_rating	0.0028		
review_scores_accuracy	0.0024		
host_is_superhost_encoded	0.0010		
neighbourhood_group_cleansed_encoded	0.0004		
review_scores_checkin	0.0003		

```
# 데이터 프레임 df1 의 막대그래프 그리기
```

```
import seaborn as sns  
  
sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)
```



```

# graphviz 불러오기

import graphviz

# model의 결과물을 tree.dot에 저장

from sklearn.tree import export_graphviz

export_graphviz(best_clf, out_file="tree.dot",
feature_names=list(data_decisiontree.columns), impurity=False,
filled=True)

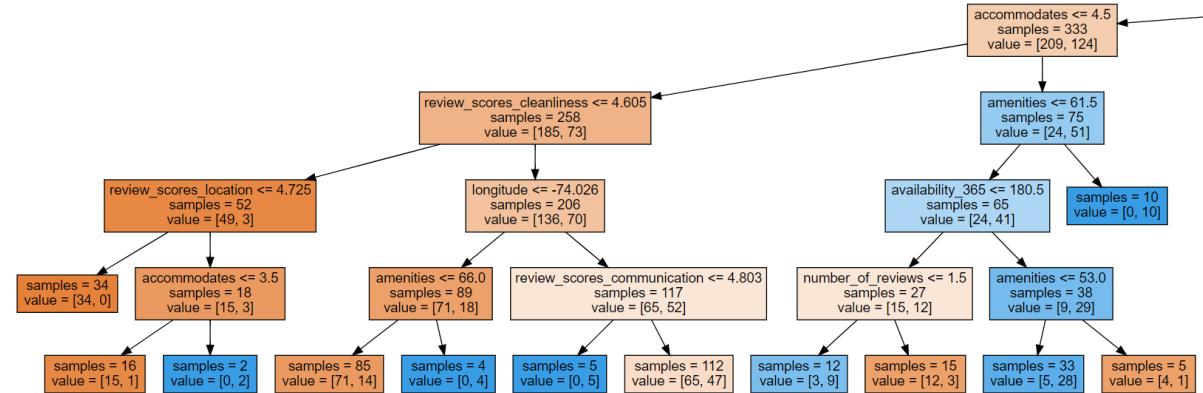
# tree.dot 을 graphviz 기능을 통해 출력

with open("tree.dot") as f:

    dot_graph = f.read()

display(graphviz.Source(dot_graph))

```



3.6.2 로지스틱 회귀 분류 모델

```
import pandas as pd

df_logisticregression = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/transformation_completed.csv')

print(df_logisticregression.shape)

df_logisticregression.head()
```

```
# 더미 변수를 만들 필요 없는 host_is_superhost, instant_bookable의 f 와 t 를
# 이진값으로 변경

df_logisticregression['host_is_superhost'] =
df_logisticregression['host_is_superhost'].replace({'f': 0, 't': 1})

df_logisticregression['instant_bookable'] =
df_logisticregression['instant_bookable'].replace({'f': 0, 't': 1})

df_logisticregression.head()
```

```
# 나머지 범주형 변수 더미 변수 생성

col = ['host_response_time', 'neighbourhood_group_cleansed',
'room_type', 'bathrooms_text']

df_logisticregression = pd.get_dummies(df_logisticregression,
columns=col)

df_logisticregression.head()
```

```
# 기준 더미 변수 제거

col = ['host_response_time_a few days or more',
'neighbourhood_group_cleansed_Manhattan', 'room_type_Entire home/apt',
'bathrooms_text_0 baths']

df_logisticregression.drop(col, axis=1, inplace=True)

df_logisticregression.shape
```

```
# 타겟 변수 설정

data = df_logisticregression.drop(['price', 'price_B'], axis=1)

target = df_logisticregression['price_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```
# 로지스틱 회귀 기본 모델

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

lr = LogisticRegression(solver='lbfgs', penalty='none', random_state=4,
n_jobs=-1)

model = lr.fit(x_train, y_train)

pred = model.predict(x_test)

print("Training set score{:.5f}".format(model.score(x_train, y_train)))

print("Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

GridSearch를 실행하기 전 정확도 : 0.79510

```
lr = LogisticRegression(solver='lbfgs', penalty='none', random_state=4,
n_jobs=-1)

from sklearn.model_selection import GridSearchCV

params = {'solver':['lbfgs', 'saga'], 'penalty':['none']}

grid_lr = GridSearchCV(lr, param_grid=params, scoring='accuracy', cv=5,
n_jobs=-1)

grid_lr.fit(x_train, y_train)

print("GridsearchCV max accuracy:{:.5f}".format(grid_lr.best_score_))

print("GridSearchCV best parameter:", (grid_lr.best_params_))

best_clf = grid_lr.best_estimator_

pred = best_clf.predict(x_test)

print("accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

GridSearch 를 실행한 후 테스트 데이터세트의 정확도 : 0.79510

bets parameter : {'penalty': 'none', 'solver': 'lbfgs'}

```

# 변수명을 index로 만들고 coefficient 값을 매칭한 데이터프레임 만들기

import numpy as np

feature_names = list(data.columns)

dft = pd.DataFrame(np.round(best_clf.coef_, 3).transpose(),
index=feature_names, columns=['coef'])

# coef 값을 오름차순으로 정리

dft1 = dft.sort_values(by='coef', ascending=True)

dft1

```

	coef		
room_type_Private room	-1.025	number_of_reviews	-0.001
neighbourhood_group_cleansed_Queens	-0.855	bathrooms_text_4 baths	-0.001
bathrooms_text_1 shared bath	-0.652	availability_365	-0.001
neighbourhood_group_cleansed_Brooklyn	-0.620	bathrooms_text_4.5 shared baths	0.000
neighbourhood_group_cleansed_Bronx	-0.315	bathrooms_text_15.5 baths	0.000
review_scores_value	-0.211	bathrooms_text_5.5 baths	0.000
bathrooms_text_2 shared baths	-0.193	bathrooms_text_5.5 shared baths	0.000
bedrooms	-0.162	bathrooms_text_8.5 baths	0.000
host_response_time_within a few hours	-0.140	bathrooms_text_0 shared baths	0.000
host_is_superhost	-0.134	bathrooms_text_8 baths	0.000
bathrooms_text_1.5 shared baths	-0.104	bathrooms_text_6 baths	0.000
review_scores_checkin	-0.104	bathrooms_text_6.5 baths	0.000
neighbourhood_group_cleansed_Staten Island	-0.101	bathrooms_text_7 baths	0.000
review_scores_communication	-0.095	bathrooms_text_3.5 shared baths	0.000
latitude	-0.069	bathrooms_text_6 shared baths	0.001
room_type_Shared room	-0.038	bathrooms_text_5 baths	0.002
review_scores_accuracy	-0.034	bathrooms_text_4.5 baths	0.002
bathrooms_text_3 shared baths	-0.018	bathrooms_text_0.5 bath	0.002
review_scores_rating	-0.018	bathrooms_text_3.5 baths	0.008
longitude	-0.017	availability_90	0.010
bathrooms_text_2.5 shared baths	-0.012	bathrooms_text_3 baths	0.016
bathrooms_text_4 shared baths	-0.009	amenities	0.020
host_response_rate	-0.004	bathrooms_text_2.5 baths	0.040
bathrooms_text_0.5 shared bath	-0.003	host_response_time_within a day	0.040

room_type_Hotel room	0.050
bathrooms_text_1.5 baths	0.091
review_scores_cleanliness	0.158
beds	0.164
host_response_time_within an hour	0.169
instant_bookable	0.254
bathrooms_text_2 baths	0.304
review_scores_location	0.392
accommodates	0.460
bathrooms_text_1 bath	0.506

```
# coefficient 값을 제곱한 오즈비값을 index에 매칭한 데이터프레임 만들기
```

```
feature_names = list(data.columns)

dft = pd.DataFrame(np.round(np.exp(best_clf.coef_), 3).transpose(),
index=feature_names, columns=['Odds_ratio'])

# coef를 내림차순으로 정리

dft1 = dft.sort_values(by='Odds_ratio', ascending=False)

dft1
```

Odds_ratio			
bathrooms_text_1 bath	1.659	bathrooms_text_3.5 baths	1.008
accommodates	1.584	bathrooms_text_5 baths	1.002
review_scores_location	1.480	bathrooms_text_0.5 bath	1.002
bathrooms_text_2 baths	1.356	bathrooms_text_4.5 baths	1.002
instant_bookable	1.289	bathrooms_text_6 shared baths	1.001
host_response_time_within an hour	1.184	bathrooms_text_15.5 baths	1.000
beds	1.178	bathrooms_text_8 baths	1.000
review_scores_cleanliness	1.171	bathrooms_text_4.5 shared baths	1.000
bathrooms_text_1.5 baths	1.095	bathrooms_text_3.5 shared baths	1.000
room_type_Hotel room	1.051	bathrooms_text_7 baths	1.000
bathrooms_text_2.5 baths	1.041	bathrooms_text_5.5 baths	1.000
host_response_time_within a day	1.040	bathrooms_text_5.5 shared baths	1.000
amenities	1.020	bathrooms_text_6 baths	1.000
bathrooms_text_3 baths	1.017	bathrooms_text_6.5 baths	1.000
availability_90	1.010	bathrooms_text_0 shared baths	1.000
		bathrooms_text_8.5 baths	1.000

bathrooms_text_4 baths	0.999		
availability_365	0.999	neighbourhood_group_cleansed_Staten Island	0.904
number_of_reviews	0.999	bathrooms_text_1.5 shared baths	0.901
bathrooms_text_0.5 shared bath	0.997	review_scores_checkin	0.901
host_response_rate	0.996	host_is_superhost	0.874
bathrooms_text_4 shared baths	0.991	host_response_time_within a few hours	0.870
bathrooms_text_2.5 shared baths	0.988	bedrooms	0.850
longitude	0.983	bathrooms_text_2 shared baths	0.825
review_scores_rating	0.983	review_scores_value	0.810
bathrooms_text_3 shared baths	0.982	neighbourhood_group_cleansed_Bronx	0.729
review_scores_accuracy	0.966	neighbourhood_group_cleansed_Brooklyn	0.538
room_type_Shared room	0.963	bathrooms_text_1 shared bath	0.521
latitude	0.934	neighbourhood_group_cleansed_Queens	0.425
review_scores_communication	0.909	room_type_Private room	0.359

```
# 데이터프레임 dft1 의 막대그래프 그리기
```

```
import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

plt.figure(figsize=(10, 12))

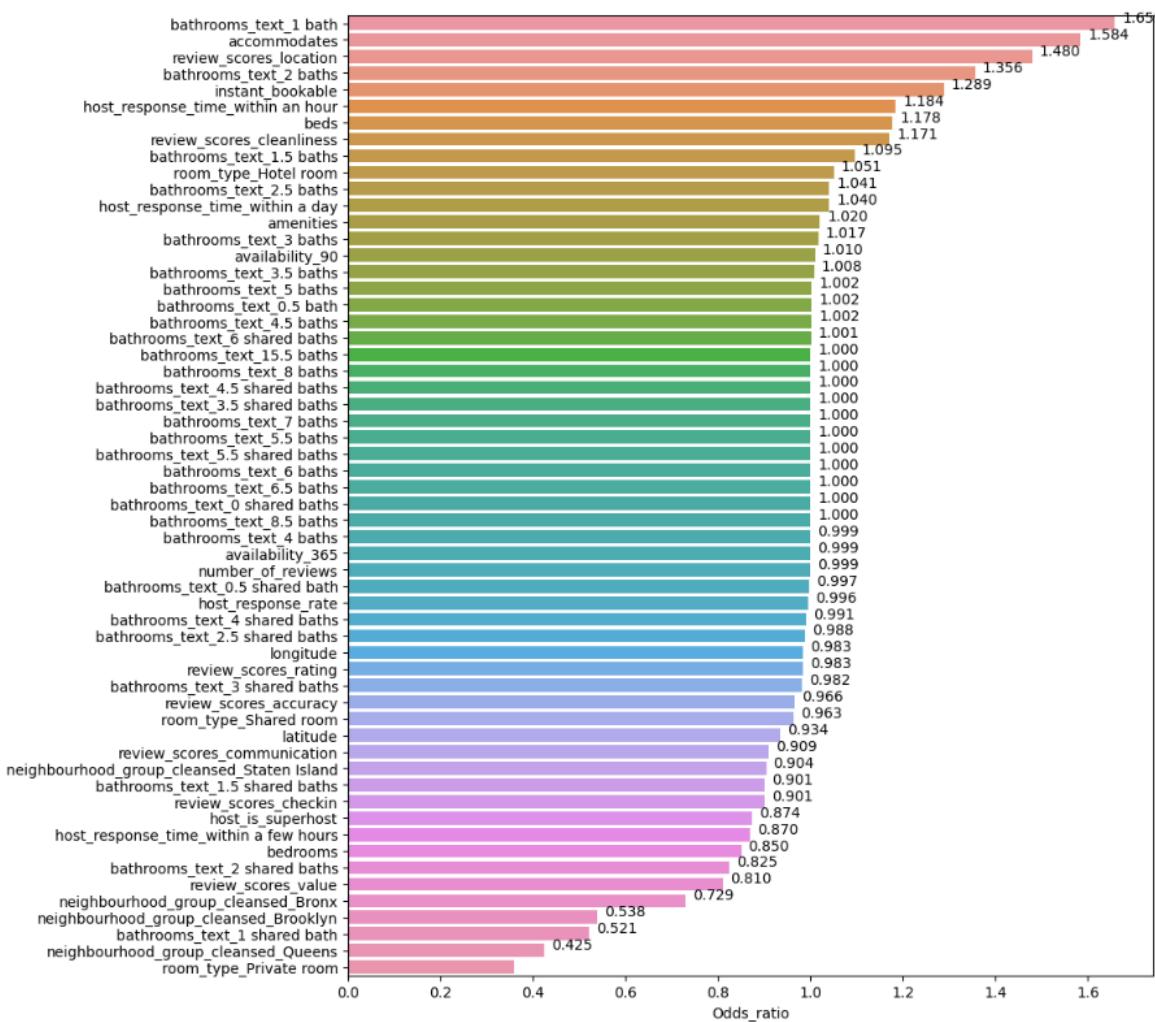
ax = sns.barplot(y=dft1.index, x="Odds_ratio", data=dft1)

for p in ax.patches:

    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y()+1.2), xytext=(5, 10), textcoords='offset points')

ax.set_yticklabels(ax.get_yticklabels(), fontsize=10)

plt.show()
```



- 오즈비 해석

- 구간 변수

수용인원이 1 명 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 58.4% 증가한다.

위치에 대한 리뷰 점수가 1 점 늘어날 경우 가격이 중위값보다 비쌀 확률은 48% 증가한다.

침대가 1 개 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 17.8% 증가한다.

가격 대비 만족도 점수가 1 점 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 19% 감소한다.

침실 수가 1 개 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 15% 감소한다.

체크인 만족도 점수가 1 점 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 9.9% 감소한다.

- 범주형 변수

개인 화장실이 없는 경우와 비교하여 개인 화장실이 1 개인 경우가 숙소 가격이 중위값보다 비쌀 가능성이 1.66 배 높다.

개인 화장실이 없는 경우와 비교하여 개인 화장실이 2 개인 경우가 숙소 가격이 중위값보다 비쌀 가능성이 1.63 배 높다.

즉시 예약이 불가능한 경우와 비교하여 즉시 예약이 가능한 경우가 숙소 가격이 중위값보다 비쌀 가능성이 1.29 배 높다.

숙소 지역이 Manhattan 인 경우와 비교하여 숙소 지역이 Queens 일 경우가 숙소 가격이 중위값보다 비쌀 가능성이 0.43 배 낮다.

개인 화장실이 없는 경우와 비교하여 공용 화장실이 1 개인 경우가 숙소 가격이 중위값보다 비쌀 가능성이 0.52 배 낮다.

숙소 지역이 Manhattan 인 경우와 비교하여 숙소 지역이 Brooklyn 일 경우가 숙소 가격이 중위값보다 비쌀 가능성이 0.54 배 낮다.

3.6.3 표준화한 로지스틱 회귀 모델

```
# 구간 변수들만 별도로 모아 데이터프레임 df_num 을 만든다

numeric_cols = ['host_response_rate', 'latitude', 'longitude',
'accommodates', 'bedrooms', 'beds', 'amenities', 'availability_90',
'availability_365', 'number_of_reviews', 'review_scores_rating',
'review_scores_accuracy', 'review_scores_cleanliness',
'review_scores_checkin', 'review_scores_communication',
'review_scores_location', 'review_scores_value']

df_num = df_logisticregression[numeric_cols]

# StandardScaler()로 데이터 스케일을 표준화하고, 결과를 데이터프레임으로 만든다

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_num_standard = pd.DataFrame(scaler.fit_transform(df_num))

# StandardScaler()는 변수명을 지우므로 데이터프레임에 다시 변수명을 넣는다

df_num_standard.columns = df_num.columns

df_num_standard.head(3)
```

```
# 원래 데이터프레임에서 구간 변수들을 제거하여 df_cat 에 저장

df_cat = df_logisticregression.drop(numeric_cols, axis=1)

# 구간 변수 스케일을 표준화한 df_num_standard 와 범주형 변수만 담은 df_cat 을 병합

dfu_standard = pd.concat([df_num_standard, df_cat], axis=1)

# dfu 의 변수명을 나열

dfu_standard.columns
```

```

# 표준화한 데이터세트로 로지스틱 회귀 재실행

data = dfu_standard.drop(['price', 'price_B'], axis=1)

target = dfu_standard['price_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)

# 로지스틱 회귀 기본 모델

lr = LogisticRegression(solver='lbfgs', penalty='none', random_state=4,
n_jobs=-1)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {'solver':['lbfgs', 'saga'], 'penalty':['none']}

grid_lr = GridSearchCV(lr, param_grid=params, scoring='accuracy', cv=5,
n_jobs=-1)

grid_lr.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_lr.best_score_))

print("GridSearchCV best parameter:", grid_lr.best_params_)

best_clf = grid_lr.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch 를 실행한 후 테스트 데이터세트의 정확도 : 0.81308

```

bests parameter : {'penalty': 'none', 'solver': 'lbfgs'}

dfu_standard.to_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/standard-scaled.csv', index=False)

```

3.6.4 사이킷런 신경망 분류 모델

```
import pandas as pd

import numpy as np

df_neural_network_knn = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/standard-scaled.csv')

data = df_neural_network_knn.drop(['price', 'price_B'], axis=1)

target = df_neural_network_knn['price_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```
# 신경망 기본 모델

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score

clf_mlp = MLPClassifier(max_iter=2000, random_state=0)

clf_mlp.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수의 예측값 생성

pred = clf_mlp.predict(x_test)

accuracy = accuracy_score(y_test, pred)

print("Training set score:{:.5f}".format(clf_mlp.score(x_train,
y_train)))

print("Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```

# 신경망 기본 모델

clf_mlp = MLPClassifier(max_iter=500, random_state=4)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {
    'solver': ['sgd', 'lbfgs', 'adam'],
    'activation': ['tanh', 'relu', 'logistic'],
    'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
}

grid_mlp = GridSearchCV(clf_mlp, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1, error_score='raise')

grid_mlp.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_mlp.best_score_))

print("GridSearchCV best parameter:", (grid_mlp.best_params_))

best_clf = grid_mlp.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.83613

bets parameter : {'activation': 'logistic', 'alpha': 0.01, 'solver': 'adam'}

```

# 신경망 기본 모델

clf_mlp = MLPClassifier(max_iter=500, random_state=4)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {
    'solver': ['adam'],
    'activation': ['logistic'],
    'alpha': [0.01],
}

grid_mlp = GridSearchCV(clf_mlp, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1, error_score='raise')

grid_mlp.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_mlp.best_score_))

print("GridSearchCV best parameter:", (grid_mlp.best_params_))

```

3.6.5 최근접 이웃 분류 모델(KNN)

```
import pandas as pd

import numpy as np

df_knn = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/standard-scaled.csv')

data = df_knn.drop(['price', 'price_B'], axis=1)

target = df_knn['price_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```
# KNN 모델

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

clf_knn = KNeighborsClassifier(n_neighbors=3)

clf_knn.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타식 변수 예측값을 생성

pred = clf_knn.predict(x_test)

accuracy = accuracy_score(y_test, pred)

print("Training set score:{:.5f}".format(clf_knn.score(x_train,
y_train)))

print("Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
# KNN 모델

clf_knn = KNeighborsClassifier(n_neighbors=3)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {'n_neighbors' : range(3, 31)}

grid_knn = GridSearchCV(
    clf_knn, param_grid=params, scoring='accuracy', cv=3, n_jobs=-1)

grid_knn.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_knn.best_score_))

print("GridSearchCV best parameter:", (grid_knn.best_params_))
```

```
best_clf = grid_knn.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.83019

bets parameter : {'n_neighbors': 17}

3.6.6 랜덤 포레스트

```
import pandas as pd

# OrdinalEncoder 를 import 하여 범주형 데이터 변환

df_randomforest = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/transformation_completed.csv')

from sklearn.preprocessing import OrdinalEncoder

df_randomforest['host_response_time_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['host_response_time'].va
lues.reshape(-1,1))

df_randomforest['host_is_superhost_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['host_is_superhost'].val
ues.reshape(-1,1))

df_randomforest['neighbourhood_group_cleansed_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['neighbourhood_group_cle
ansed'].values.reshape(-1,1))

df_randomforest['room_type_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['room_type'].values.resh
ape(-1,1))

df_randomforest['bathrooms_text_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['bathrooms_text'].values
.reshape(-1,1))

df_randomforest['instant_bookable_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['instant_bookable'].valu
es.reshape(-1,1))

df_randomforest.head(3)
```

```
# 기존 범주형 변수 열 삭제
```

```
df_randomforest.drop(['host_response_time', 'host_is_superhost',
'neighbourhood_group_cleansed', 'room_type', 'bathrooms_text',
'instant_bookable'], axis=1,inplace=True)

df_randomforest.info()
```

```
df_randomforest.head()
```

```
df_randomforest.to_csv('/content/drive/MyDrive/Colab  
Notebooks/Data_Analysis/encoded.csv', index = False)
```

```
# 범주형 변수를 cols에 저장.
```

```
cols = ['host_response_time_encoded', 'host_is_superhost_encoded',  
'neighbourhood_group_cleansed_encoded', 'room_type_encoded',  
'bathrooms_text_encoded', 'instant_bookable_encoded', 'price_B']
```

```
# 범주형 변수의 dtype을 category로 변경
```

```
df_randomforest[cols] = df_randomforest[cols].astype('category')
```

```
df_randomforest.dtypes
```

```
data = df_randomforest.drop(['price', 'price_B'], axis=1)
```

```
target = df_randomforest['price_B']
```

```
print("data_shape : ", data.shape)
```

```
print("target_shape : ", target.shape)
```

```
# 데이터 분할
```

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

```
x_train, x_test, y_train, y_test = train_test_split(data, target,  
test_size=0.3, random_state=4, stratify=target)
```

```

# 랜덤 포레스트 모델(기본 모델, tree depth 제한 없음)

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(n_estimators=100, random_state=4)

model = rf.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수 예측값 생성

pred = rf.predict(x_test)

print("Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

과적합이 발생했으므로 과적합 해소, 교차 검증, 모델 최적화를 위해 학습하지 않은 기본 모델을 다시 생성한 후 그리드 서치 실행

```

# 랜덤 포레스트 모델(기본 모델, tree depth 제한 없음)

rf = RandomForestClassifier(n_estimators=100, random_state=4)

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold

# StratifiedKFold의 random_state 옵션 값을 4로 고정

cross_validation = StratifiedKFold(n_splits=5, shuffle=True,
random_state=4)

params = {"max_depth" : range(10, 41), 'n_estimators' : [100,200]}

#GridSearchCV의 cv=cross_validation 옵션값은 StratifiedKFold의
#random_state 옵션값을 적용해서 GridSearchCV를 실행할 때마다 항상 동일한 결과가
#나오도록 보장

grid_rf = GridSearchCV(rf, param_grid=params, scoring='accuracy',
cv=cross_validation, verbose=1, n_jobs=-1)

grid_rf.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_rf.best_score_))

print("GridSearchCV best parameter:", (grid_rf.best_params_))

```

```
best_clf = grid_rf.best_estimator_
pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:, 1])

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.85006

ROC AUC 값 : 0.92615

bets parameter : {'max_depth': 24, 'n_estimators': 200}

```
print ("Feature importances:")
```

```
print (best_clf.feature_importances_)
```

Feature importances:

```
[0.01758667 0.09107983 0.12699807 0.09748449 0.05977246 0.04918135
 0.06440887 0.03559836 0.03700048 0.03075192 0.02347329 0.0218179
 0.02673539 0.01885674 0.01803783 0.02831958 0.02504138 0.01639593
 0.00484147 0.02698775 0.12363632 0.04668772 0.00930623]
```

```

import numpy as np

# 변수명을 index로 만들고 feature_importances를 매칭해서 나열한 데이터프레임
만들기

feature_names = list(data.columns)

dft = pd.DataFrame(np.round(best_clf.feature_importances_,
3),index=feature_names, columns=['Feature_importances'])

# Feature_importances의 값을 내림차순으로 정리

dft1 = dft.sort_values(by="Feature_importances", ascending=False)

dft1

```

Feature_importances	
longitude	0.127
room_type_encoded	0.124
accommodates	0.097
latitude	0.091
amenities	0.064
bedrooms	0.060
beds	0.049
bathrooms_text_encoded	0.047
availability_365	0.037
availability_90	0.036
number_of_reviews	0.031
review_scores_location	0.028
review_scores_cleanliness	0.027
neighbourhood_group_cleansed_encoded	0.027
review_scores_value	0.025
review_scores_rating	0.023
review_scores_accuracy	0.022
review_scores_checkin	0.019
host_response_rate	0.018
review_scores_communication	0.018
host_response_time_encoded	0.016
instant_bookable_encoded	0.009
host_is_superhost_encoded	0.005

```
# 데이터프레임 dft1 의 막대그래프(barplot) 그리기

import matplotlib.pyplot as plt

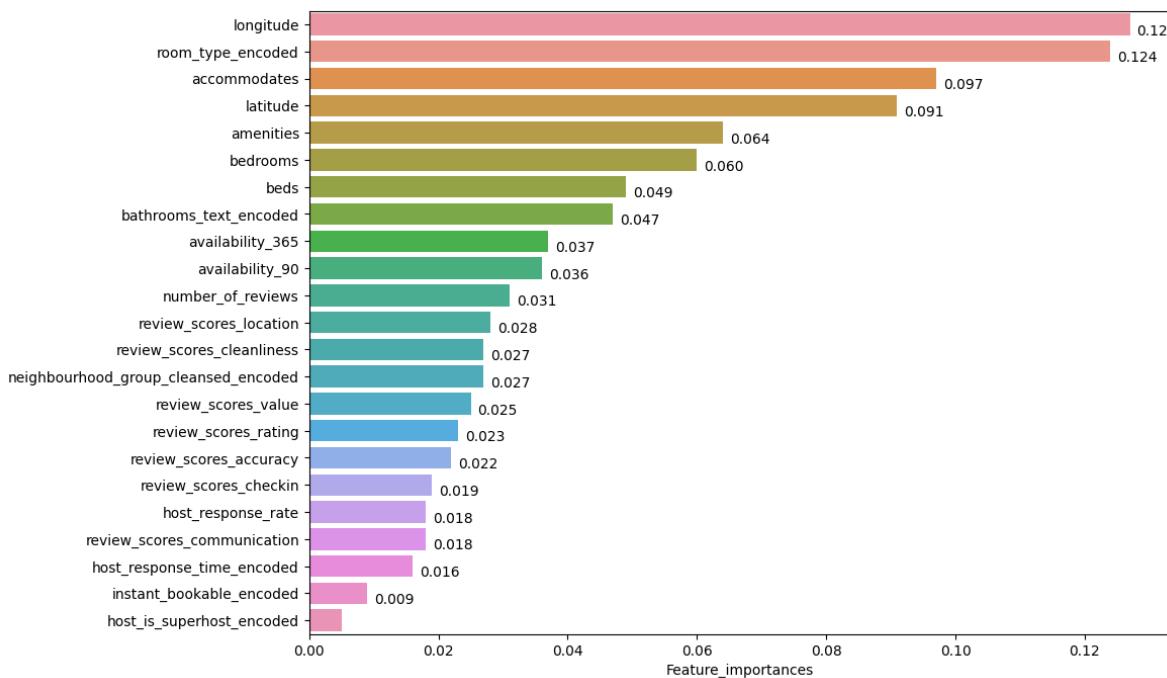
import seaborn as sns

%matplotlib inline

fig, ax = plt.subplots(figsize=(11,8))

ax = sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y() + 1.3), xytext=(5,10), textcoords='offset points')
```



3.6.7 그레디언트 부스팅 모델

```
import pandas as pd

df_gradientboosting = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/encoded.csv')
```

```
# 범주형 변수를 cols에 저장.

cols = ['host_response_time_encoded', 'host_is_superhost_encoded',
'neighbourhood_group_cleansed_encoded', 'room_type_encoded',
'bathrooms_text_encoded', 'instant_bookable_encoded', 'price_B']

# 범주형 변수의 dtype을 category로 변경

df_gradientboosting[cols] =
df_gradientboosting[cols].astype('category')

df_gradientboosting.dtypes
```

```
data = df_gradientboosting.drop(['price', 'price_B'], axis=1)

target = df_gradientboosting['price_B']

print("data_shape : ", data.shape)

print("target_shape : ", target.shape)
```

```
# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)
```

```

# 그레디언트 부스팅 모델 (기본 모델)

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score

gr = GradientBoostingClassifier(random_state=4)

model = gr.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수 예측값 생성

pred = model.predict(x_test)

accuracy = accuracy_score(y_test, pred)

print("Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

```

gr = GradientBoostingClassifier(random_state=4)

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold

# StratifiedKFold의 random_state 옵션값을 4로 고정

cross_validation = StratifiedKFold(n_splits=3, shuffle=True,
random_state=4)

params = {'max_depth':range(11,16), 'n_estimators':[100,200],
'learning_rate':[0.01, 0.1]}

# GridSearchCV의 cv=cross_validation 옵션값은 StratifiedKFold의
# random_state 옵션값을 적용해 GridSearchCV를 실행할 때마다 항상 동일한 결과가
# 나오도록 보장

grid_gr = GridSearchCV(model, param_grid=params, scoring='accuracy',
cv=cross_validation, n_jobs=-1)

grid_gr.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_gr.best_score_))

print("GridSearchCV best parameter:", (grid_gr.best_params_))

```

```
best_clf = grid_gr.best_estimator_
pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:,1])

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.84723

ROC AUC 값 : 0.92579

bets parameter : {'learning_rate': 0.1, 'max_depth': 12,
'n_estimators': 200}

```
# 데이터프레임의 행과 열 전체를 보이게 하는 조치를 추가

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

```

import numpy as np

# 변수명을 index로 만들고 feature_importances를 매칭해서 나열한 데이터프레임
만들기

feature_names = list(data.columns)

dft = pd.DataFrame(np.round(best_clf.feature_importances_, 3),
index=feature_names, columns=['Feature_importances'])

dft1 = dft.sort_values(by='Feature_importances', ascending=False)

# 데이터프레임 dft1의 막대그래프 그리기

import matplotlib.pyplot as plt

import seaborn as sns

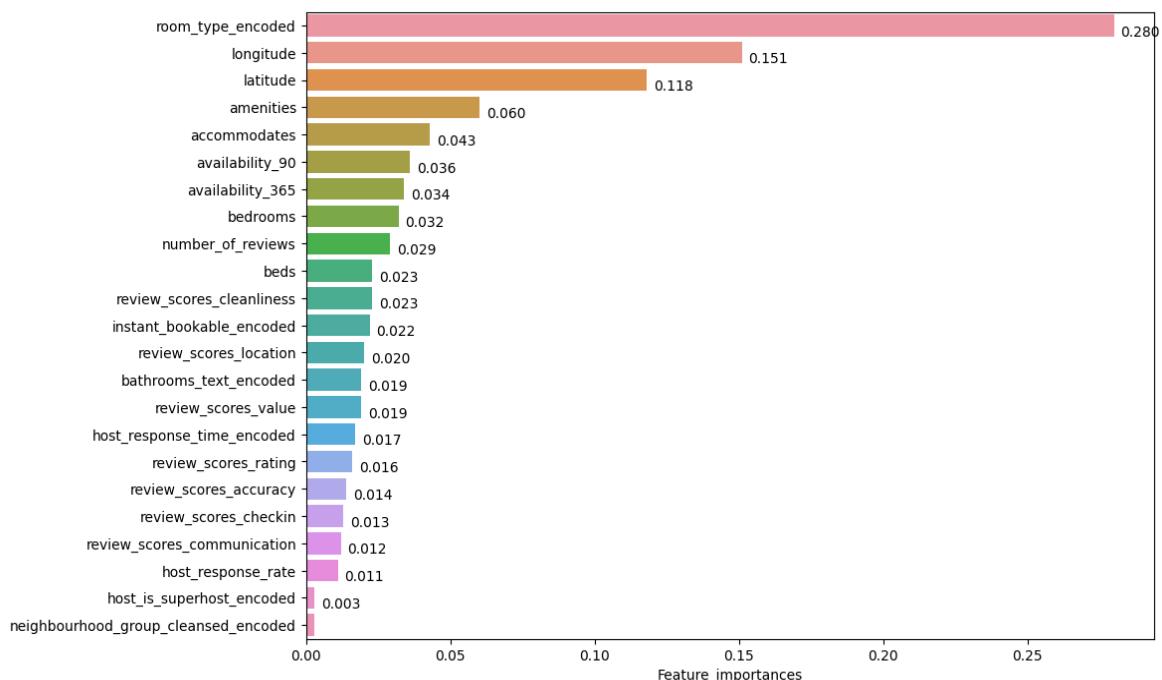
%matplotlib inline

fig, ax = plt.subplots(figsize=(11,8))

ax = sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y() + 1.3), xytext=(5,10), textcoords='offset points')

```



3.6.8 라쏘 모델

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/encoded.csv')

df.info()

cols=['host_response_rate', 'latitude', 'longitude', 'accommodates',
'bedrooms', 'beds', 'amenities', 'availability_90', 'availability_365',
'number_of_reviews', 'review_scores_rating', 'review_scores_accuracy',
'review_scores_cleanliness', 'review_scores_checkin',
'review_scores_communication', 'review_scores_location',
'review_scores_value', 'price', 'price_B']

df_cat = df.drop(cols, axis=1) # 데이터프레임에서 7 개 구간 변수 및 2 개의
# 타겟변수 제외

df_cat.shape

pd.options.display.float_format = '{:.2f}'.format # 소숫점 2 자리로 숫자
# 표기 제한

df_cat.describe()

df_cat.max() - df_cat.min()

df_cat.min()

# 6 개 범주형 변수중에서 값으로 0 과 1 만 가지는 (이미 더미 변수화 되어 있는) 2 개
# 변수명을 제외하고서 cols1에 저장

cols1 = ['host_response_time_encoded',
'neighbourhood_group_cleansed_encoded',
'room_type_encoded', 'bathrooms_text_encoded']

df_lasso = pd.get_dummies(df, columns=cols1) # cols1에 담긴 변수들의
# 더미변수를 생성

# 이 명령은 더미변수를 생성한 원본변수는 제거함
```

```

df_lasso.head()

df_lasso.shape

list(df_lasso.columns)

# 기준 더미변수로 정한 4 개의 더미변수명을 cols2에 저장

cols2 = ['host_response_time_encoded_0.0',
        'neighbourhood_group_cleansed_encoded_2.0', 'room_type_encoded_0.0',
        'bathrooms_text_encoded_0.0']

df_lasso.drop(cols2, axis=1, inplace=True)      # cols2에 저장된 더미
                                                # 변수명을 데이터프레임에서 제거

df_lasso.shape

data = df_lasso.drop(['price', 'price_B'], axis=1)

target = df_lasso['price_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)

print("x_train shape:", x_train.shape)

print("x_test shape:", x_test.shape)

# 라쏘 모델(liblinear를 사용한 기본 모델)

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

ls = LogisticRegression(penalty="l1", solver='liblinear', C=1,
random_state=4)

model = ls.fit(x_train, y_train)

# 학습된 classifier로 테스트 데이터세트를 이용해서 타깃 변수 예측값 생성

pred = model.predict(x_test)

print("Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

```

# 라쏘 모델(liblinear를 사용한 기본 모델)

ls =
LogisticRegression(penalty='l1', solver='liblinear', C=1, random_state=0)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold

# StratifiedKFold의 random_state 옵션값을 특정 숫자로 고정

cross_validation = StratifiedKFold(n_splits=5, shuffle=True,
random_state=0)

params = {'solver':['lbfgs', 'liblinear', 'sag', 'saga'],
'C':[0.01,0.05,0.1,0.2,0.3,0.5,1]}

# GridSearchCV의 cv=cross_validation 옵션값은 위의 StratifiedKFold의
# random_state 옵션값을 적용시켜서
# GridSearchCV를 실행할 때마다 결과가 항상 동일하게 나오도록 보장

grid_ls = GridSearchCV(ls, param_grid=params, scoring='accuracy',
cv=cross_validation, n_jobs=-1, verbose=1)

grid_ls.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_ls.best_score_))

print("GridSearchCV best parameter:", (grid_ls.best_params_))

```

```

best_clf = grid_ls.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test,best_clf.predict_proba(x_test)[:, 1])

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))

```

```
GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.80817
```

```
ROC AUC 값 : 0.88493
```

```
bets parameter : {'C': 1, 'solver': 'liblinear'}
```

```
import numpy as np
```

```
print("Number of features used:", np.sum(best_clf.coef_ != 0))
```

```
print('회귀계수', best_clf.coef_)
```

```
feature_names = list(data.columns)
```

```
# 변수 index에 coefficient 값을 매칭해 데이터프레임으로 저장
```

```
dft = pd.DataFrame(best_clf.coef_.transpose(), index=feature_names,
columns=['coef'])
```

```
dft1 = dft.sort_values(by='coef', ascending=False)
```

```
dft1
```

	coef		
room_type_encoded_1.0	3.55	bathrooms_text_encoded_19.0	0.00
bathrooms_text_encoded_15.0	1.57	bathrooms_text_encoded_25.0	0.00
bathrooms_text_encoded_11.0	1.04	bathrooms_text_encoded_21.0	0.00
review_scores_location	0.93	bathrooms_text_encoded_22.0	0.00
bathrooms_text_encoded_9.0	0.73	bathrooms_text_encoded_4.0	0.00
review_scores_cleanliness	0.63	bathrooms_text_encoded_23.0	0.00
accommodates	0.43	bathrooms_text_encoded_24.0	0.00
bathrooms_text_encoded_2.0	0.31	bathrooms_text_encoded_20.0	0.00
bathrooms_text_encoded_6.0	0.27	bathrooms_text_encoded_29.0	0.00
beds	0.24	availability_365	-0.00
instant_bookable_encoded	0.20	number_of_reviews	-0.00
host_response_time_encoded_3.0	0.10	bedrooms	-0.06
review_scores_rating	0.08	longitude	-0.08
amenities	0.02	host_is_superhost_encoded	-0.08
host_response_rate	0.01	review_scores_communication	-0.14
availability_90	0.01	host_response_time_encoded_1.0	-0.16
review_scores_accuracy	0.01	review_scores_checkin	-0.25
bathrooms_text_encoded_16.0	0.00	latitude	-0.26
bathrooms_text_encoded_8.0	0.00	host_response_time_encoded_2.0	-0.31
bathrooms_text_encoded_28.0	0.00	room_type_encoded_3.0	-0.42
bathrooms_text_encoded_27.0	0.00	bathrooms_text_encoded_3.0	-0.53
bathrooms_text_encoded_13.0	0.00	bathrooms_text_encoded_17.0	-0.54
bathrooms_text_encoded_26.0	0.00	bathrooms_text_encoded_1.0	-0.56

review_scores_value	-0.68
bathrooms_text_encoded_5.0	-1.10
neighbourhood_group_cleansed_encoded_1.0	-1.17
room_type_encoded_2.0	-1.18
bathrooms_text_encoded_7.0	-1.18
bathrooms_text_encoded_12.0	-1.20
bathrooms_text_encoded_10.0	-1.87
bathrooms_text_encoded_14.0	-2.07
neighbourhood_group_cleansed_encoded_3.0	-2.07
neighbourhood_group_cleansed_encoded_0.0	-2.26
bathrooms_text_encoded_18.0	-2.70
neighbourhood_group_cleansed_encoded_4.0	-3.06

오즈비 계산

```
feature_names = list(data.columns) # 변수명(컬럼명)을 리스트 형태로 만들기

dft = pd.DataFrame(np.exp(best_clf.coef_).transpose(),
index=feature_names, columns=['Odds_ratio'])

dft1 = dft.sort_values(by='Odds_ratio', ascending=False) # 컬럼 coef의
값들을 내림차순으로 정리

dft1
```

	Odds_ratio		
room_type_encoded_1.0	34.64	bathrooms_text_encoded_26.0	1.00
bathrooms_text_encoded_15.0	4.80	bathrooms_text_encoded_19.0	1.00
bathrooms_text_encoded_11.0	2.83	bathrooms_text_encoded_25.0	1.00
review_scores_location	2.54	bathrooms_text_encoded_21.0	1.00
bathrooms_text_encoded_9.0	2.08	bathrooms_text_encoded_22.0	1.00
review_scores_cleanliness	1.88	bathrooms_text_encoded_4.0	1.00
accommodates	1.53	bathrooms_text_encoded_23.0	1.00
bathrooms_text_encoded_2.0	1.36	bathrooms_text_encoded_24.0	1.00
bathrooms_text_encoded_6.0	1.31	bathrooms_text_encoded_20.0	1.00
beds	1.27	bathrooms_text_encoded_29.0	1.00
instant_bookable_encoded	1.22	availability_365	1.00
host_response_time_encoded_3.0	1.10	number_of_reviews	1.00
review_scores_rating	1.08	bedrooms	0.94
amenities	1.02	longitude	0.93
host_response_rate	1.01	host_is_superhost_encoded	0.93
availability_90	1.01	review_scores_communication	0.87
review_scores_accuracy	1.01	host_response_time_encoded_1.0	0.86
bathrooms_text_encoded_16.0	1.00	review_scores_checkin	0.78
bathrooms_text_encoded_8.0	1.00	latitude	0.77
bathrooms_text_encoded_28.0	1.00	host_response_time_encoded_2.0	0.73
bathrooms_text_encoded_27.0	1.00	room_type_encoded_3.0	0.66
bathrooms_text_encoded_13.0	1.00	bathrooms_text_encoded_3.0	0.59
		bathrooms_text_encoded_17.0	0.58

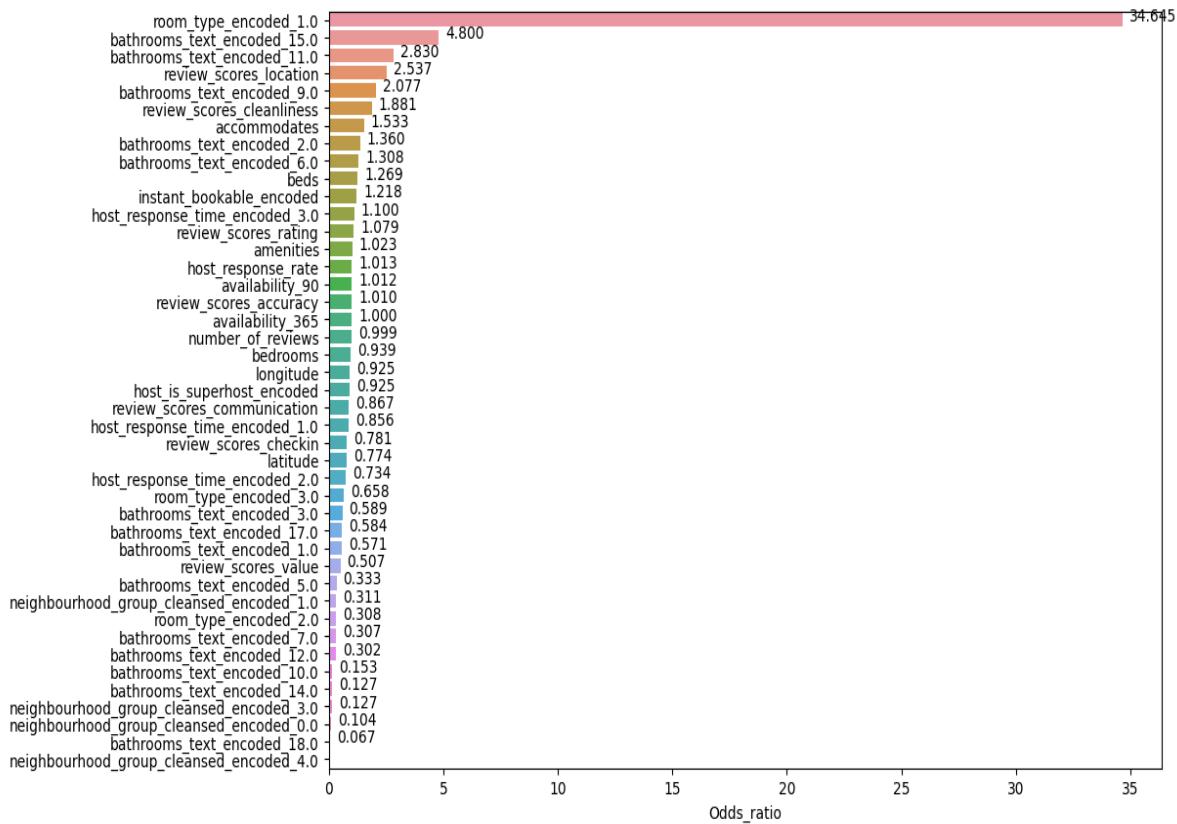
bathrooms_text_encoded_1.0	0.57
review_scores_value	0.51
bathrooms_text_encoded_5.0	0.33
neighbourhood_group_cleansed_encoded_1.0	0.31
room_type_encoded_2.0	0.31
bathrooms_text_encoded_7.0	0.31
bathrooms_text_encoded_12.0	0.30
bathrooms_text_encoded_10.0	0.15
bathrooms_text_encoded_14.0	0.13
neighbourhood_group_cleansed_encoded_3.0	0.13
neighbourhood_group_cleansed_encoded_0.0	0.10
bathrooms_text_encoded_18.0	0.07
neighbourhood_group_cleansed_encoded_4.0	0.05

```
# 오즈비가 1(계수값이 0)인 변수를 제거
```

```
dft2 = dft1[dft1['Odds_ratio'] != 1]
dft2.shape
```

```
# 데이터프레임 dft2 의 막대그래프 그리기
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
fig, ax= plt.subplots(figsize=(11,8))
ax = sns.barplot(y=dft2.index, x="Odds_ratio", data=dft2)
for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y()+1.4), xytext=(5, 10), textcoords='offset points')
```



- 오즈비 해석

- 구간 변수

위치에 대한 리뷰 점수가 1 점 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 2.54 배 증가한다.

청결도에 대한 리뷰 점수가 1 점 늘어날 경우 가격이 중위값보다 비쌀 확률은 88.1% 증가한다.

수용 인원이 1 명 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 53.3% 증가한다.

체크인 만족도 점수가 1 점 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 21.9% 감소한다.

의사소통 만족도 점수가 1 점 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 13.3% 감소한다.

경도가 1 도 늘어날 경우 숙소 가격이 중위값보다 비쌀 확률은 7.5% 감소한다.

- 범주형 변수

방의 종류가 아파트인 경우와 비교하여 방의 종류가 호텔인 경우가 숙소 가격이 중위값보다 비쌀 가능성은 34.64배 높다.

개인 화장실이 없는 경우와 비교하여 개인 화장실이 3.5개인 경우가 숙소 가격이 중위값보다 비쌀 가능성은 4.80배 높다.

개인 화장실이 없는 경우와 비교하여 개인 화장실이 2.5개인 경우가 숙소 가격이 중위값보다 비쌀 가능성은 2.83배 높다.

숙소 지역이 Manhattan인 경우와 비교하여 숙소 지역이 Staten Island인 경우가 숙소 가격이 중위값보다 비쌀 가능성은 0.05배 낮다.

개인 화장실이 없는 경우와 비교하여 공용 화장실이 4개인 경우가 숙소 가격이 중위값보다 비쌀 가능성은 0.07배 낮다.

숙소 지역이 Manhattan인 경우와 비교하여 숙소 지역이 Bronx인 경우가 숙소 가격이 중위값보다 비쌀 가능성은 0.10배 낮다.

3.6.9 텐서플로우 케라스 신경망 모델

```
import pandas as pd

df_tf_keras = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/standard-scaled.csv')

df_tf_keras.head()
```

```
df_tf_keras.shape
```

```
data = df_tf_keras.drop(['price', 'price_B'], axis=1)

target = df_tf_keras['price_B']

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)

print("x_train shape:", x_train.shape)

print("x_test shape:", x_test.shape)
```

```
type(x_train)
```

```
x_train_np = x_train.to_numpy()

x_test_np = x_test.to_numpy()

y_train_np = y_train.to_numpy()

y_test_np = y_test.to_numpy()
```

```
print(x_train_np)
```

```
x_train_np.shape
```

```
print(y_train_np)
```

```
import tensorflow as tf

from tensorflow.keras import layers

# 활성화 함수 ralu, 옵티마이저 adam

import random as python_random

import numpy as np

np.random.seed(123)

python_random.seed(123)

tf.random.set_seed(1234)
```

```
# 활성화 함수 ralu
```

```
model = tf.keras.Sequential([
    layers.Dense(100, activation='relu', input_shape=[58]),
    layers.Dropout(.5),
    layers.Dense(100, activation='relu'),
    layers.Dropout(.5),
    layers.Dense(1, activation='sigmoid')
])
```

```
# 옵티마이저 adam
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'], run_eagerly=True)
```

```
from tensorflow import keras

from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint('Ch8-NN1.tf', save_best_only=True)

early_stopping_cb = keras.callbacks.EarlyStopping(patience=5,
monitor='val_accuracy', restore_best_weights=True)

history = model.fit(x_train_np, y_train_np,
                     validation_split=0.25,
                     shuffle=True,
                     epochs=30,
                     callbacks = [checkpointer, early_stopping_cb]
)
```

```
model.summary()
```

```
Model: "sequential_3"
-----

| Layer (type)        | Output Shape | Param # |
|---------------------|--------------|---------|
| dense_9 (Dense)     | (None, 100)  | 5900    |
| dropout_6 (Dropout) | (None, 100)  | 0       |
| dense_10 (Dense)    | (None, 100)  | 10100   |
| dropout_7 (Dropout) | (None, 100)  | 0       |
| dense_11 (Dense)    | (None, 1)    | 101     |


-----  

Total params: 16101 (62.89 KB)  

Trainable params: 16101 (62.89 KB)  

Non-trainable params: 0 (0.00 Byte)
-----
```

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']          # 모델의 학습 정확도를 변수 acc에
저장

val_acc = history.history['val_accuracy'] # 모델의 검증 정확도를 변수
val_acc에 저장

loss=history.history['loss']              # 모델의 학습 손실을 변수 loss에
저장

val_loss=history.history['val_loss']       # 모델의 검증 손실을 변수
val_loss에 저장

epochs_range = range(1, 24+1)

# 학습 정확도와 검증 정확도 그리기

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs_range, acc, label='Training Accuracy')

plt.plot(epochs_range, val_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

# 학습 손실과 검증 손실 그리기

plt.subplot(1, 2, 2)

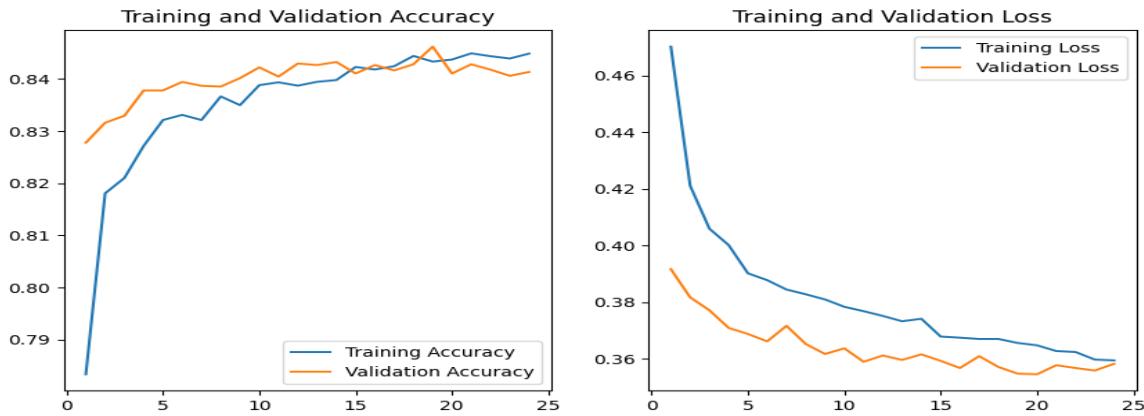
plt.plot(epochs_range, loss, label='Training Loss')

plt.plot(epochs_range, val_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()
```



```
# model.fit() 실행시 검증 정확도가 가장 높은 에포크에 해당하는 모델 가중치 계수 불러오기
```

```
model.load_weights('/content/Ch8-NN1.tf')
```

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)

print("test accuracy:", test_accuracy)
```

```
y_prob = model.predict(x_test)

y_prob.round(2)
```

```
from sklearn.metrics import roc_auc_score

y_prob = model.predict(x_test)

ROC_AUC = roc_auc_score(y_test, y_prob)

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.83622
```

```
ROC AUC 값 : 0.91674
```

3.6.10 서포트 벡터 머신

```
import pandas as pd

df_svm = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/standard-scaled.csv')

df_svm.head()

data = df_svm.drop(['price', 'price_B'], axis=1) # 타겟변수를 제외한
# 입력변수를 data에 저장

target = df_svm['price_B'] # 타겟변수만 target 데이터프레임에
# 저장

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)

print("x_train shape:", x_train.shape)

print("x_test shape:", x_test.shape)

# SVM 모델(default 모델)

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

svm = SVC(kernel='rbf', C=1, gamma = 'auto', random_state=4,
probability=True)

model = svm.fit(x_train, y_train)

pred = model.predict(x_test) # 학습된 classifier로 테스트 데이터셋
# 자료이용해서 타겟변수 예측값 생성

accuracy = accuracy_score(y_test, pred)

print("SVM Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("SVM Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

```
# SVM 모델(default 모델)

svm = SVC(kernel='rbf', C=1, gamma='auto', random_state=4,
probability=True)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold

cross_validation = StratifiedKFold(n_splits=3, shuffle=True,
random_state=4)

params = {'kernel':['sigmoid'], 'C':[0.0001, 0.01, 1, 10],
'gamma':['auto', 'scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
cv=cross_validation, n_jobs=-1)

grid_svm.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))

print("GridSearchCV best parameter:", (grid_svm.best_params_))
```

```
# SVM 모델(default 모델)

svm = SVC(kernel='rbf', C=1, gamma='auto', random_state=4,
probability=True)

# 그리드 서치 재실행

from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold

cross_validation = StratifiedKFold(n_splits=3, shuffle=True,
random_state=4)

params = {'kernel':['rbf'], 'C':[0.0001, 0.01, 1, 10], 'gamma':['auto',
'scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
cv=cross_validation, n_jobs=-1)

grid_svm.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))

print("GridSearchCV best parameter:", (grid_svm.best_params_))
```

```
best_clf = grid_svm.best_estimator_
pred = best_clf.predict(x_test)
print("Accuracy on test:{:.5f}".format(accuracy_score(y_test, pred)))
from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.83690

ROC AUC 값 : 0.91320

bests parameter : {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}

3.6.11 회귀, 릿지 모델

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/encoded.csv')

df.info()

cols=['host_response_rate', 'latitude', 'longitude', 'accommodates',
'bedrooms', 'beds', 'amenities', 'availability_90', 'availability_365',
'number_of_reviews', 'review_scores_rating', 'review_scores_accuracy',
'review_scores_cleanliness', 'review_scores_checkin',
'review_scores_communication', 'review_scores_location',
'review_scores_value', 'price', 'price_B']

df_cat = df.drop(cols, axis=1) # 데이터프레임에서 7 개 구간 변수 및 2 개의
# 타겟변수 제외

df_cat.shape

pd.options.display.float_format = '{:.2f}'.format # 소숫점 2 자리로 숫자
# 표기 제한

df_cat.describe()

df_cat.max() - df_cat.min()

df_cat.min()
```

```
# 6 개 범주형 변수중에서 값으로 0 과 1 만 가지는 (이미 더미 변수화 되어 있는) 2 개  
변수명을 제외하고서 cols1에 저장  
  
cols1 = ['host_response_time_encoded',  
'neighbourhood_group_cleansed_encoded',  
'room_type_encoded', 'bathrooms_text_encoded']  
  
df_ridge = pd.get_dummies(df, columns=cols1) # cols1에 담긴 변수들의  
더미변수를 생성  
# 이 명령은 더미변수를 생성한 원본변수는 제거함
```

```
df_ridge.shape
```

```
list(df_ridge.columns)
```

```
# 기준 더미변수로 정한 4 개의 더미변수명을 cols2에 저장  
  
cols2 = ['host_response_time_encoded_0.0',  
'neighbourhood_group_cleansed_encoded_2.0', 'room_type_encoded_0.0',  
'bathrooms_text_encoded_0.0']  
  
df_ridge.drop(cols2, axis=1, inplace=True) # cols2에 저장된 더미  
변수명을 데이터프레임에서 제거  
  
df_ridge.shape
```

```
data = df_ridge.drop(['price', 'price_B'], axis=1) # 타겟변수를 제외한  
입력변수를 data에 저장  
  
target = df['price'] # 타겟변수만 target에 저장  
  
# 데이터 분할  
  
from sklearn.model_selection import train_test_split  
  
x_train, x_test, y_train, y_test = train_test_split(data, target,  
test_size=0.3, random_state=4)  
  
print("x_train shape:", x_train.shape)  
  
print("x_test shape:", x_test.shape)
```

```
# 선형 회귀 모델(기본 모델)

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

linr = LinearRegression(n_jobs=-1)

model = linr.fit(x_train, y_train)

pred = model.predict(x_test)

print("Linear Regression Training set
score:{:.5f}".format(model.score(x_train, y_train)))

print("Linear Regression Test set r2
score:{:.5f}".format(r2_score(y_test, pred)))
```

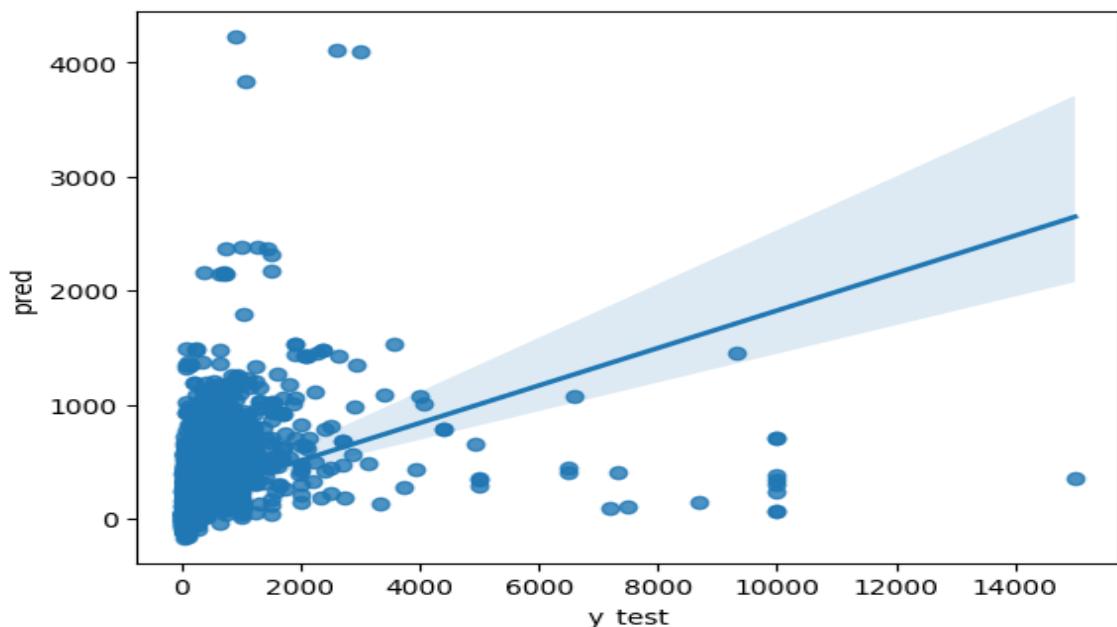
```
import seaborn as sns

final_result = pd.concat([y_test.reset_index(drop=True),
pd.DataFrame(pred)], axis = 1)

final_result.columns = ['y_test', 'pred']

sns.regplot(x='y_test', y='pred', data=final_result)
```

<Axes: xlabel='y_test', ylabel='pred'>



```
# 럿지 모델(기본 모델)

from sklearn.linear_model import Ridge

from sklearn.metrics import r2_score

Ridge = Ridge()

model = Ridge.fit(x_train, y_train)

pred = model.predict(x_test)

print("Linear Regression Training set
score:{:.5f}".format(model.score(x_train, y_train)))

print("Linear Regression Test set score:{:.5f}".format(r2_score(y_test,
pred)))
```

테스트 데이터세트의 r2 score : 0.12533

```
# 럿지 모델(기본 모델)

from sklearn.linear_model import Ridge

from sklearn.model_selection import GridSearchCV

Ridge = Ridge()

params = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000],
'solver':['auto', 'svd', 'lsqr',
'cholesky', 'sparse_cg', 'sag', 'saga', 'lbfgs']}}

grid_Ridge = GridSearchCV(Ridge, param_grid=params, scoring='r2', cv=5,
n_jobs=-1, verbose=1)

grid_Ridge.fit(x_train, y_train)

print("GridSearchCV max score:{:.5f}".format(grid_Ridge.best_score_))

print("GridSearchCV best parameter:", (grid_Ridge.best_params_))
```

```
best_clf = grid_Ridge.best_estimator_
pred = best_clf.predict(x_test)
print("R2 Score on test
set:{:.5f}".format(best_clf.score(x_test,y_test)))
```

GridSearch 를 실행한 후 r2 score: 0.13386

bets parameter : {'alpha': 10, 'solver': 'svd'}

3.6.12 XGBoost

```
pip install xgboost
```

```
pip install lightgbm
```

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

df_xgboost = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/encoded.csv')

df_xgboost.shape
```

```
df_xgboost.head()
```

```
# 이진값 타겟변수 제외

df_xgboost.drop(['price_B'], axis=1, inplace=True)

df_xgboost.shape
```

```
from sklearn.model_selection import train_test_split

data = df_xgboost.drop(['price'], axis=1)

target = df_xgboost['price']

# 데이터 분할

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)
```

```
# 기본 XGBRegressor 모델

from sklearn.metrics import r2_score

xgb = XGBRegressor(random_state=4)

xgb.fit(x_train, y_train)

pred = xgb.predict(x_test)

print("r2: {:.5f}".format(r2_score(y_test, pred)))
```

```
# XGBRegressor 최적화

from sklearn.model_selection import GridSearchCV

xgb = XGBRegressor()

parameters = {'colsample_bytree': [0.7],
              'learning_rate': [0.05],
              'max_depth': [16],
              'min_child_weight' : [4],
              'n_estimators': [1000],
              'subsample': [0.8, 0.9]
             }

xgb_grid = GridSearchCV(xgb,
                        parameters,
                        scoring = 'r2',
                        cv = 3,
                        n_jobs = -1,
                        verbose=True)

xgb_grid.fit(x_train, y_train)
```

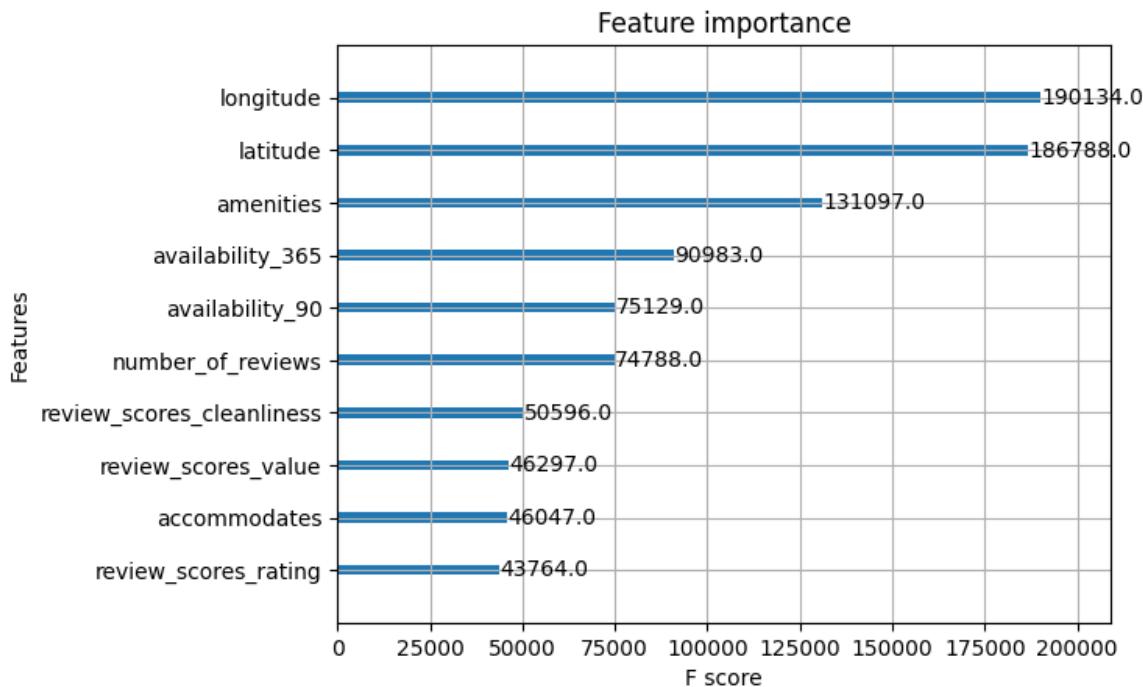
```
print('GridSearchCV 최적 파라미터:', xgb_grid.best_params_)
```

```
model = xgb_grid.best_estimator_
pred = model.predict(x_test)
print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

GridSearch를 실행한 후 r2 score : 0.18120

```
bests parameter : {'colsample_bytree': 0.7, 'learning_rate': 0.05,
'max_depth': 16, 'min_child_weight': 4, 'n_estimators': 1000,
'subsample': 0.9}
```

```
from xgboost import plot_importance
plot_importance(model, max_num_features=10)
```



```

from xgboost import plot_importance

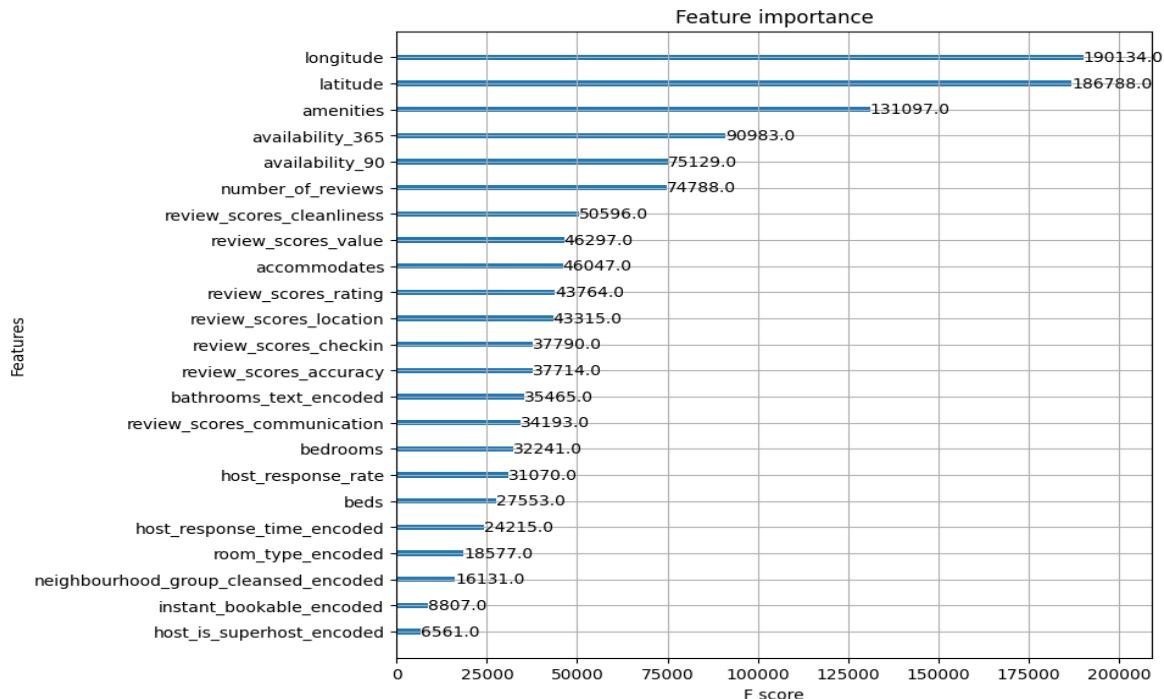
import matplotlib.pyplot as plt

%matplotlib inline

fig, ax = plt.subplots(figsize=(8, 8))

plot_importance(model, ax=ax)

```



3.6.13 LightGBM

```
# 기본 LGBMRegressor 모델

from lightgbm import LGBMRegressor

from sklearn.metrics import r2_score

lgb = LGBMRegressor(random_state=4)

lgb.fit(x_train, y_train)

pred = lgb.predict(x_test)

print("r2: {:.5f}".format(r2_score(y_test, pred)))
```

```
# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

from lightgbm import LGBMRegressor

lgb = LGBMRegressor()

parameters = {'colsample_bytree':[0.7, 0.8],
              'learning_rate':[0.1, 0.15, 0.2],
              'max_depth':[11],
              'min_child_weight':[4],
              'n_estimators':[1000],
              'subsample':[0.3, 0.4]}

lgb_grid = GridSearchCV(lgb,
                        parameters,
                        scoring='r2',
                        cv=3,
                        n_jobs=-1,
                        verbose=True)

lgb_grid.fit(x_train, y_train)

print("GridSearchCV 최적 파라미터:", lgb_grid.best_params_)
```

```

from sklearn.metrics import r2_score

model = lgb_grid.best_estimator_

pred = model.predict(x_test)

print("r2: {:.5f}".format(r2_score(y_test, pred)))

```

GridSearch 를 실행한 후 r2 score : 0.20502

bets parameter : {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 11, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.3}

```

from lightgbm import plot_importance

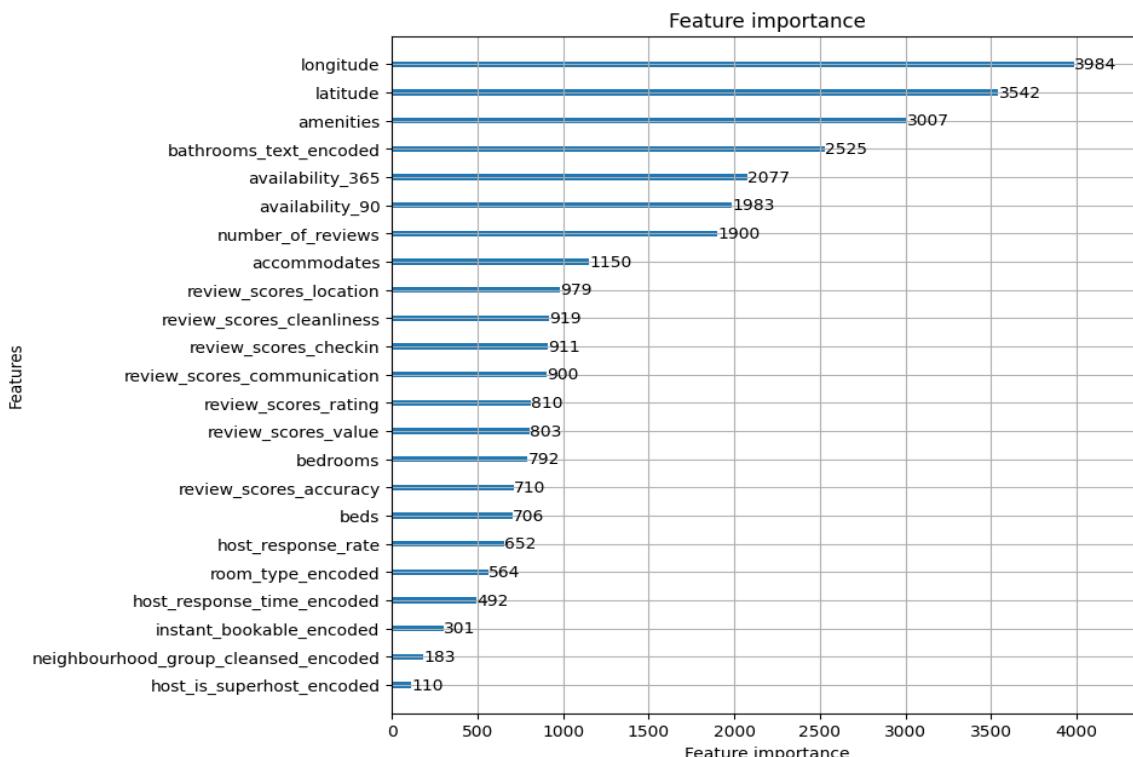
import matplotlib.pyplot as plt

%matplotlib inline

fig, ax = plt.subplots(figsize=(8, 8))

plot_importance(model, ax=ax);

```



3.6.14 스태킹 모델(양상블)

```
model_xgb = xgb_grid.best_estimator_
pred_xgb = model_xgb.predict(x_test)

model_lgb = lgb_grid.best_estimator_
pred_lgb = model_lgb.predict(x_test)
```

```
from sklearn.metrics import r2_score

pred = 0.5*pred_xgb + 0.5*pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.4*pred_xgb + 0.6*pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.3*pred_xgb + 0.7*pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.2*pred_xgb + 0.8*pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.1*pred_xgb + 0.9*pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
0.4 * pred_xgb + 0.6* pred_lgb 을 때 r2 score : 0.22618
```

3.7 모델 정확도 및 최적 파라미터

3.7.1 머신러닝 모델 정확도 및 최적 파라미터

모델명	정확도	순위	최적 파라미터
결정 트리	0.81454	3	{'criterion': 'entropy', 'max_depth': 9}
로지스틱 회귀	0.80275	5	{'penalty': 'none', 'solver': 'lbfgs'}
로지스틱 회귀 (표준화)	0.81308	4	{'penalty': 'none', 'solver': 'lbfgs'}
신경망 (표준화)	0.83613	1	{'activation': 'logistic', 'alpha': 0.01, 'solver': 'adam'}
K-최근접 이웃 (표준화)	0.83019	2	{'n_neighbors': 17}

3.7.2 이진 분류 머신러닝 모델 정확도 및 최적 파라미터

모델명	정확도	ROC AUC	순위	최적 파라미터
랜덤 포레스트	0.85006	0.92615	1	{'max_depth': 24, 'n_estimators': 200}
그래디언트 부스팅	0.84723	0.92579	2	{'learning_rate': 0.1, 'max_depth': 12, 'n_estimators': 200}
라쏘 (로지스틱 회귀)	0.80817	0.88493	5	{'C': 1, 'solver': 'liblinear'}
신경망 (tf.keras)	0.83622	0.91674	4	
서포트 벡터 머신	0.83690	0.91320	3	{'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}

3.7.3 연속형 회귀 머신러닝 모델 r2 score

모델명	r2 score	최적 파라미터
회귀 모델	0.12533	
릿지 모델	0.13386	{'alpha': 10, 'solver': 'svd'}
XGBoost	0.18120	{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 16, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.9}
LightGBM	0.20502	{'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 11, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.3}
스태킹 모델 (앙상블)	0.22618	

4. 부동산 가격 분석

4.1 문제 제기

부동산의 가격을 분석하기 위해 연구 주제를 다음과 같이 설정했다.

- 부동산 가격에 영향을 미치는 요인은 무엇인가?

4.2 타깃 변수 설정

이 프로젝트에서 연속형 변수와 범주형 변수를 구분하여 분석하기 위해 2개의 타깃 변수를 생성했다. 먼저 연속형 변수의 경우 판매 가격인 SALE PRICE로 설정했다. 범주형 변수의 경우 이진 값을 가지고 있는 PRICE_B로 설정했다. PRICE_B는 원본 데이터세트에는 없던 변수로 기존에 있던 연속형 변수인 SALE PRICE에 기반하여 판매 가격 변수값이 New York 주택 가격의 중위수 이상이면 price_B는 1, 중위수 미만이면 0으로 값을 부여했다.

파이썬 코드

```
# 범주형 변수 BOROUGH, BUILDING CLASS CATEGORY, BUILDING CLASS AT
PRESENT, TAX CLASS AT TIME OF SALE, PRICE_B (5)

# 타겟 변수 값이 중위수 이상이면 1, 아니면 0

# 이진값 타겟 변수 PRICE_B

c1 = delete_columns_data['SALE PRICE'] >= delete_columns_data['SALE
PRICE'].median()

c0 = delete_columns_data['SALE PRICE'] < delete_columns_data['SALE
PRICE'].median()

delete_columns_data.loc[c1, "PRICE_B"] = 1

delete_columns_data.loc[c0, "PRICE_B"] = 0

delete_columns_data.head(3)
```

```
# SALE PRICE 변수 object에서 float로 변환

import pandas as pd

delete_columns_data = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/delete_columns_data.csv')

delete_columns_data['SALE PRICE'] = delete_columns_data['SALE
PRICE'].astype(int)

delete_columns_data.info()
```

```
delete_columns_data['PRICE_B'] =
delete_columns_data['PRICE_B'].astype('object')

delete_columns_data.info()
```

4.3 데이터 처리

부동산에 관련된 Raw data는 총 84548개의 행, 21개의 열로 구성되어 있다. 이 중에서 가격에 영향을 주지 않을 것이라고 판단되는 13개의 변수를 제거했다. 최종적인 데이터는 37237개의 행, 10개의 열로 구성되어 있다.

- 최종 변수 정리

변수명	타입	변수 설명
BOROUGH	object	부동산 위치
BUILDING_CLASS_CATEGORY	object	부동산 종류
BUILDING_CLASS_AT_PRESENT	object	건물 분류
TOTAL_UNITS	int64	총 단위 수
LAND_SQUARE_FEET	int64	토지 면적
GROSS_SQUARE_FEET	int64	건물 층의 총 면적
YEAR_BUILT	int64	건물이 지어진 연도
TAX_CLASS_AT_TIME_OF_SALE	object	판매시점의 세무급
SALE PRICE	int64	판매 가격
PRICE_B	object	판매 가격 이진값

4.3.1 데이터 불러오기

```
from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

source_data = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/nyc-rolling-sales.csv')

source_data.shape

delete_columns_data = source_data.drop(['Unnamed: 0', 'BLOCK', 'LOT',
'EASE-MENT', 'ADDRESS', 'APARTMENT NUMBER', 'ZIP CODE', 'RESIDENTIAL
UNITS', 'COMMERCIAL UNITS', 'BUILDING CLASS AT TIME OF SALE', 'SALE
DATE', 'NEIGHBORHOOD', 'TAX CLASS AT PRESENT'], axis=1)

delete_columns_data.shape

delete_columns_data.info()

# '-'가 있는 행 찾기

rows_with_minus_sign =
delete_columns_data[delete_columns_data.apply(lambda row: ' - ' in
row.values, axis=1)]

# 해당 행을 삭제

delete_columns_data =
delete_columns_data.drop(rows_with_minus_sign.index)

delete_columns_data.info()
```

4.3.2 구간 변수 데이터 처리

```
# 구간 변수 TOTAL UNITS, LAND SQUARE FEET, GROSS SQUARE FEET, YEAR BUILT,  
SALE PRICE (5)  
  
# 구간 변수 타입 변경  
  
delete_columns_data['LAND SQUARE FEET'] = delete_columns_data['LAND  
SQUARE FEET'].astype(int)  
  
delete_columns_data['GROSS SQUARE FEET'] = delete_columns_data['GROSS  
SQUARE FEET'].astype(int)  
  
delete_columns_data['SALE PRICE'] = delete_columns_data['SALE  
PRICE'].astype(int)  
  
  
delete_columns_data.to_csv('/content/drive/MyDrive/Colab  
Notebooks/Data_Analysis/NYC Property Sales/delete_columns_data.csv',  
index=False)
```

4.3.3 구간 변수 이상값 제거

```
# 구간 변수 이상값 확인 및 제거

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('whitegrid')

fig, axes = plt.subplots(1, 2, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'TOTAL UNITS', data =
delete_columns_data)

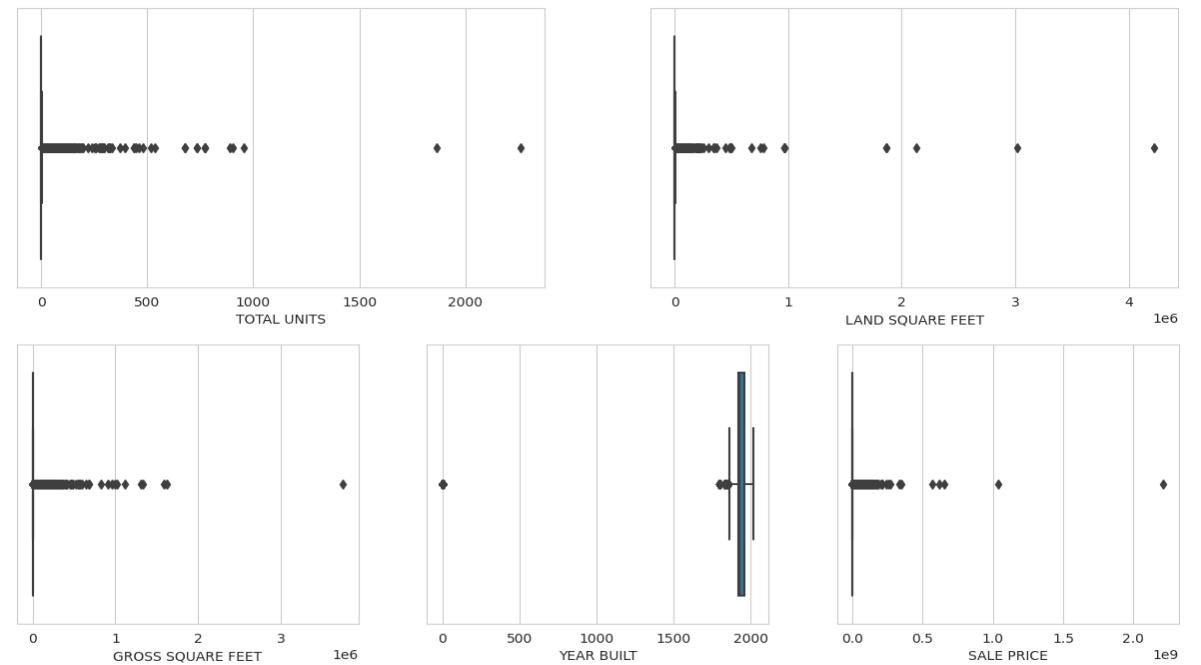
sns.boxplot(ax = axes[1], x = 'LAND SQUARE FEET', data =
delete_columns_data)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'GROSS SQUARE FEET', data =
delete_columns_data)

sns.boxplot(ax = axes[1], x = 'YEAR BUILT', data = delete_columns_data)

sns.boxplot(ax = axes[2], x = 'SALE PRICE', data = delete_columns_data)
```



```
Q1 = delete_columns_data[['TOTAL UNITS', 'LAND SQUARE FEET', 'GROSS  
SQUARE FEET', 'YEAR BUILT', 'SALE PRICE']].quantile(0.25)  
  
Q3 = delete_columns_data[['TOTAL UNITS', 'LAND SQUARE FEET', 'GROSS  
SQUARE FEET', 'YEAR BUILT', 'SALE PRICE']].quantile(0.75)  
  
IQR = Q3 - Q1  
  
print(IQR)
```

```
Lower = Q1 - 1.5 * IQR  
  
Upper = Q3 + 1.5 * IQR  
  
print(Lower)  
  
print(Upper)
```

```
c1 = (delete_columns_data['TOTAL UNITS'] >= -0.5) &  
(delete_columns_data['TOTAL UNITS'] <= 3.5)  
  
c2 = (delete_columns_data['LAND SQUARE FEET'] >= -1074.0) &  
(delete_columns_data['LAND SQUARE FEET'] <= 5558.0)  
  
c3 = (delete_columns_data['GROSS SQUARE FEET'] >= -1710.0) &  
(delete_columns_data['GROSS SQUARE FEET'] <= 5058.0)  
  
c4 = (delete_columns_data['YEAR BUILT'] >= 1858.5) &  
(delete_columns_data['YEAR BUILT'] <= 2022.5)  
  
c5 = (delete_columns_data['SALE PRICE'] >= -1043950.0) &  
(delete_columns_data['SALE PRICE'] <= 1954370.0)  
  
df = delete_columns_data[c1 & c2 & c3 & c4 & c5]  
  
df.shape
```

```
df.to_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/NYC  
Property Sales/preprocessing_completed.csv', index=False)
```

4.4 탐색적 자료 분석 및 시각화

4.4.1 구간 변수 요약 통계 검토

```
import pandas as pd

preprocessing_completed = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/NYC Property Sales/preprocessing_completed.csv')

preprocessing_completed['PRICE_B'] =
preprocessing_completed['PRICE_B'].astype('object')

preprocessing_completed.info()
```

```
interval_variable = preprocessing_completed[['TOTAL UNITS', 'LAND SQUARE FEET', 'GROSS SQUARE FEET', 'YEAR BUILT', 'SALE PRICE']]

interval_variable.describe()
```

```
# 구간 변수 왜도 확인
```

```
interval_variable.skew()
```

```
# 첨도 확인
```

```
interval_variable.kurtosis()
```

4.4.2 구간 변수 상관 관계 검토

```
round(interval_variable.corr(), 2)
```

	TOTAL UNITS	LAND SQUARE FEET	GROSS SQUARE FEET	YEAR BUILT	SALE PRICE
TOTAL UNITS	1.00	0.38	0.75	-0.19	0.09
LAND SQUARE FEET	0.38	1.00	0.59	-0.25	0.10
GROSS SQUARE FEET	0.75	0.59	1.00	-0.29	0.08
YEAR BUILT	-0.19	-0.25	-0.29	1.00	0.01
SALE PRICE	0.09	0.10	0.08	0.01	1.00

4.4.3 구간 변수 기초 통계량 및 시각화

- 기초 통계량

```
interval_variable.describe()
```

	TOTAL UNITS	LAND SQUARE FEET	GROSS SQUARE FEET	YEAR BUILT	SALE PRICE
count	37237.000000	37237.000000	37237.000000	37237.000000	3.723700e+04
mean	1.439831	2079.937562	1525.097672	1946.865027	4.908181e+05
std	0.819393	1393.243339	1051.142158	34.036405	4.103516e+05
min	0.000000	0.000000	0.000000	1865.000000	0.000000e+00
25%	1.000000	1417.000000	960.000000	1920.000000	1.400000e+05
50%	1.000000	2017.000000	1536.000000	1937.000000	4.598000e+05
75%	2.000000	2850.000000	2200.000000	1965.000000	7.250000e+05
max	3.000000	5554.000000	5025.000000	2017.000000	1.950000e+06

- 시각화 1 : 박스 플롯으로 변수의 분포 확인

```
import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('whitegrid')

fig, axes = plt.subplots(1, 2, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'TOTAL UNITS', data = interval_variable)

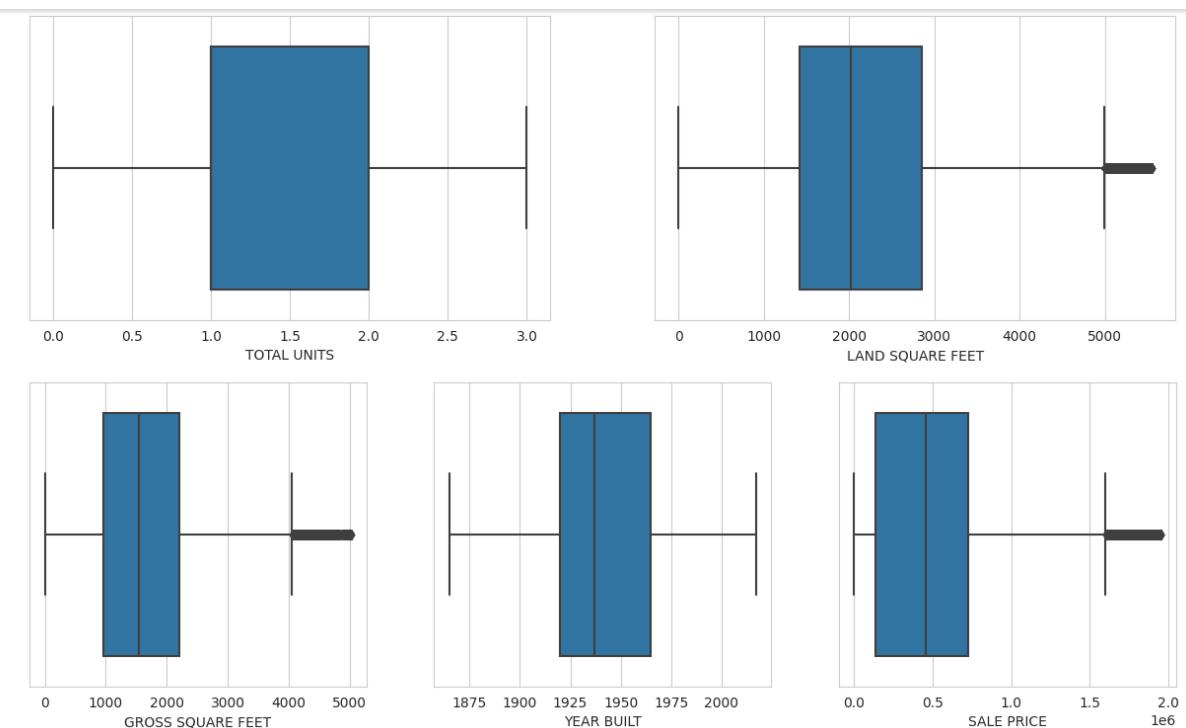
sns.boxplot(ax = axes[1], x = 'LAND SQUARE FEET', data =
interval_variable)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

sns.boxplot(ax = axes[0], x = 'GROSS SQUARE FEET', data =
interval_variable)

sns.boxplot(ax = axes[1], x = 'YEAR BUILT', data = interval_variable)

sns.boxplot(ax = axes[2], x = 'SALE PRICE', data = interval_variable)
```



- 시각화 2 : 히트맵으로 상관계수 관계 확인

```
plt.figure(figsize=(10, 5))

sns.heatmap(interval_variable.corr(), annot=True, fmt=".2f",
cmap="Blues")
```



4.4.4 범주형 변수 시각화

- 시각화 3 : 막대그래프로 변수 구성 확인

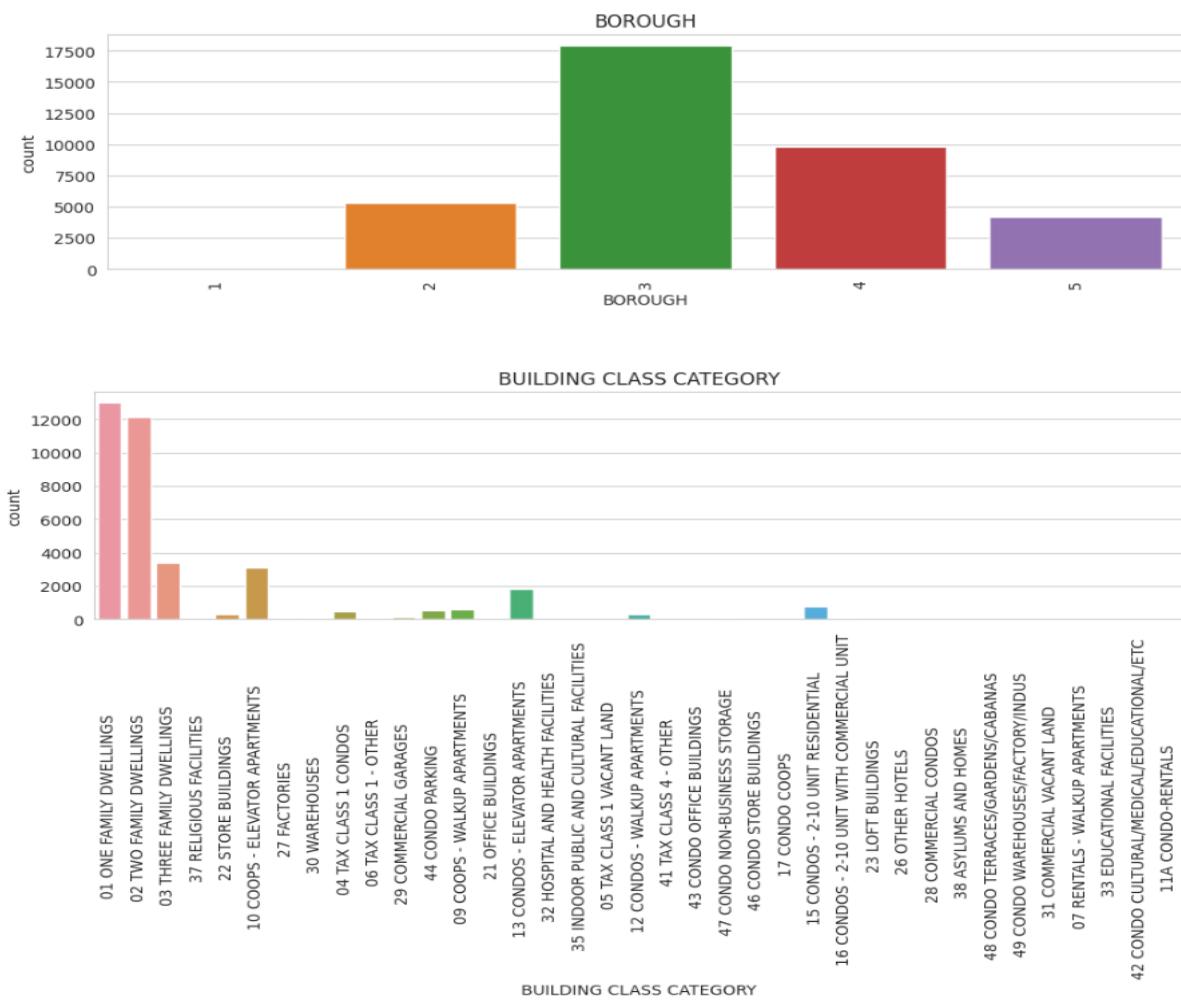
```
categorical_variable = preprocessing_completed[['BOROUGH', 'BUILDING CLASS CATEGORY', 'BUILDING CLASS AT PRESENT', 'TAX CLASS AT TIME OF SALE', 'PRICE_B']]

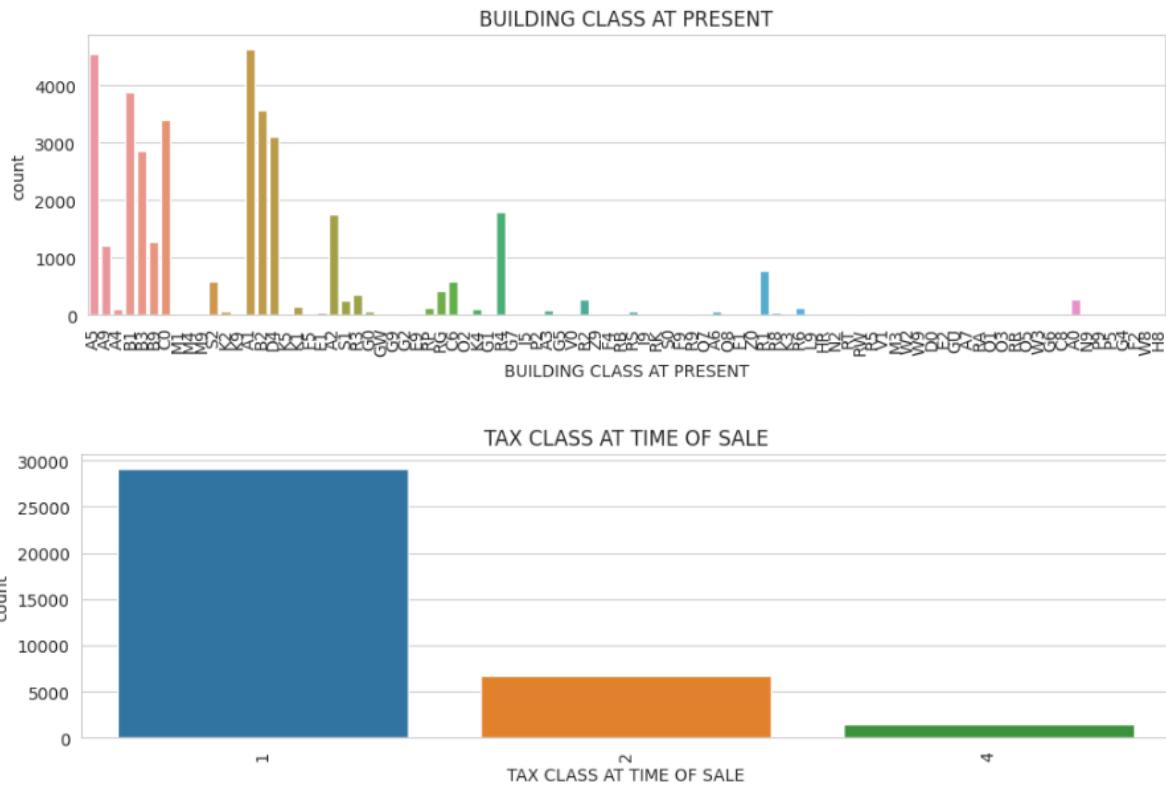
fig, axes = plt.subplots(2,2, figsize=(20,10))

for i, col in enumerate(categorical_variable.columns[:-1]):
    sns.countplot(x=col, data=categorical_variable, ax=axes[i//2, i%2])
    axes[i//2, i%2].set_title(col)
    axes[i//2, i%2].tick_params(axis='x', rotation=90)

plt.tight_layout()

plt.show()
```





```
# BOROUGH

# frequency table 생성 (개수 기준)

print(pd.crosstab(categorical_variable['BOROUGH'], columns='count'))

print()

# frequency table 생성 (비율 기준)

print(pd.crosstab(categorical_variable['BOROUGH'],
categorical_variable['PRICE_B'], normalize=True))
```

col_0	count	PRICE_B	0.0	1.0
BOROUGH		BOROUGH		
1	46	1	0.000242	0.000994
2	5314	2	0.106346	0.036362
3	17888	3	0.266267	0.214115
4	9785	4	0.091414	0.171362
5	4204	5	0.059914	0.052985

```

# BUILDING CLASS CATEGORY

# frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['BUILDING CLASS CATEGORY'],
columns='count'))

print()

# frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['BUILDING CLASS CATEGORY'],
categorical_variable['PRICE_B'], normalize=True))

```

col_0	count	PRICE_B	0.0	1.0
BUILDING CLASS CATEGORY		BUILDING CLASS CATEGORY		
01 ONE FAMILY DWELLINGS	12991	01 ONE FAMILY DWELLINGS	0.187609	0.161264
02 TWO FAMILY DWELLINGS	12164	02 TWO FAMILY DWELLINGS	0.150307	0.176357
03 THREE FAMILY DWELLINGS	3410	03 THREE FAMILY DWELLINGS	0.046191	0.045385
04 TAX CLASS 1 CONDOS	487	04 TAX CLASS 1 CONDOS	0.006929	0.006150
05 TAX CLASS 1 VACANT LAND	34	05 TAX CLASS 1 VACANT LAND	0.000591	0.000322
06 TAX CLASS 1 - OTHER	70	06 TAX CLASS 1 - OTHER	0.001584	0.000295
07 RENTALS - WALKUP APARTMENTS	2	07 RENTALS - WALKUP APARTMENTS	0.000000	0.000054
09 COOPS - WALKUP APARTMENTS	588	09 COOPS - WALKUP APARTMENTS	0.008218	0.007573
10 COOPS - ELEVATOR APARTMENTS	3121	10 COOPS - ELEVATOR APARTMENTS	0.068588	0.015227
11A CONDO-RENTALS	10	11A CONDO-RENTALS	0.000054	0.000215
12 CONDOS - WALKUP APARTMENTS	285	12 CONDOS - WALKUP APARTMENTS	0.003625	0.004028
13 CONDOS - ELEVATOR APARTMENTS	1798	13 CONDOS - ELEVATOR APARTMENTS	0.013078	0.035207
15 CONDOS - 2-10 UNIT RESIDENTIAL	777	15 CONDOS - 2-10 UNIT RESIDENTIAL	0.006177	0.014690
16 CONDOS - 2-10 UNIT WITH COMMERCIAL UNIT	49	16 CONDOS - 2-10 UNIT WITH COMMERCIAL UNIT	0.001101	0.000215
17 CONDO COOPS	30	17 CONDO COOPS	0.000349	0.000457
21 OFFICE BUILDINGS	77	21 OFFICE BUILDINGS	0.000967	0.001101
22 STORE BUILDINGS	322	22 STORE BUILDINGS	0.004941	0.003706
23 LOFT BUILDINGS	1	23 LOFT BUILDINGS	0.000000	0.000027
26 OTHER HOTELS	2	26 OTHER HOTELS	0.000000	0.000054
27 FACTORIES	35	27 FACTORIES	0.000591	0.000349
28 COMMERCIAL CONDOS	9	28 COMMERCIAL CONDOS	0.000215	0.000027
29 COMMERCIAL GARAGES	96	29 COMMERCIAL GARAGES	0.001611	0.000967
30 WAREHOUSES	74	30 WAREHOUSES	0.001047	0.000940
31 COMMERCIAL VACANT LAND	4	31 COMMERCIAL VACANT LAND	0.000081	0.000027
32 HOSPITAL AND HEALTH FACILITIES	3	32 HOSPITAL AND HEALTH FACILITIES	0.000000	0.000081
33 EDUCATIONAL FACILITIES	13	33 EDUCATIONAL FACILITIES	0.000134	0.000215
35 INDOOR PUBLIC AND CULTURAL FACILITIES	9	35 INDOOR PUBLIC AND CULTURAL FACILITIES	0.000134	0.000107
37 RELIGIOUS FACILITIES	24	37 RELIGIOUS FACILITIES	0.000322	0.000322
38 ASYLUMS AND HOMES	3	38 ASYLUMS AND HOMES	0.000081	0.000000
41 TAX CLASS 4 - OTHER	23	41 TAX CLASS 4 - OTHER	0.000591	0.000027
42 CONDO CULTURAL/MEDICAL/EDUCATIONAL/ETC	2	42 CONDO CULTURAL/MEDICAL/EDUCATIONAL/ETC	0.000054	0.000000
43 CONDO OFFICE BUILDINGS	33	43 CONDO OFFICE BUILDINGS	0.000671	0.000215
44 CONDO PARKING	553	44 CONDO PARKING	0.014690	0.000161
46 CONDO STORE BUILDINGS	28	46 CONDO STORE BUILDINGS	0.000725	0.000027
47 CONDO NON-BUSINESS STORAGE	74	47 CONDO NON-BUSINESS STORAGE	0.001987	0.000000
48 CONDO TERRACES/GARDENS/CABANAS	12	48 CONDO TERRACES/GARDENS/CABANAS	0.000295	0.000027
49 CONDO WAREHOUSES/FACTORY/INDUS	24	49 CONDO WAREHOUSES/FACTORY/INDUS	0.000645	0.000000

```

# BUILDING CLASS AT PRESENT

# frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['BUILDING CLASS AT PRESENT'],
columns='count'))

print()

# frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['BUILDING CLASS AT PRESENT'],
categorical_variable['PRICE_B'], normalize=True))

```

col_0	count	PRICE_B	0.0	1.0
BUILDING CLASS AT PRESENT		BUILDING CLASS AT PRESENT		
A0	282	A0	0.001584	0.005989
A1	4637	A1	0.062921	0.061605
A2	1750	A2	0.023606	0.023391
A3	95	A3	0.000591	0.001960
A4	119	A4	0.002148	0.001047
...
W3	3	W3	0.000000	0.000081
W8	1	W8	0.000000	0.000027
W9	7	W9	0.000081	0.000107
Z0	2	Z0	0.000054	0.000000
Z9	23	Z9	0.000591	0.000027

[90 rows x 1 columns] [90 rows x 2 columns]

```

# BTAX CLASS AT TIME OF SALE

# frequency table 생성(개수 기준)

print(pd.crosstab(categorical_variable['TAX CLASS AT TIME OF SALE'],
columns='count'))

print()

# frequency table 생성(비율 기준)

print(pd.crosstab(categorical_variable['TAX CLASS AT TIME OF SALE'],
categorical_variable['PRICE_B'], normalize=True))

```

col_0	count	PRICE_B	0.0	1.0
TAX CLASS AT TIME OF SALE		TAX CLASS AT TIME OF SALE		
1	29156	1	0.393211	0.389774
2	6660	2	0.101190	0.077665
4	1421	4	0.029782	0.008379

4.4.5 구간 변수 t-검정

- TOTAL UNITS

```
from scipy import stats

# TOTAL UNITS t-검정

data_TOTAL_UNITS_1 = interval_variable[categorical_variable['PRICE_B']
== 1]['TOTAL UNITS']

data_TOTAL_UNITS_0 = interval_variable[categorical_variable['PRICE_B']
== 0]['TOTAL UNITS']

stats.ttest_ind(data_TOTAL_UNITS_1, data_TOTAL_UNITS_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 TOTAL UNITS 는 서로 다르다

- LAND SQUARE FEET

```
# LAND SQUARE FEET t-검정

data_LAND_SQUARE_FEET_1 =
interval_variable[categorical_variable['PRICE_B'] == 1]['LAND SQUARE
FEET']

data_LAND_SQUARE_FEET_0 =
interval_variable[categorical_variable['PRICE_B'] == 0]['LAND SQUARE
FEET']

stats.ttest_ind(data_LAND_SQUARE_FEET_1, data_LAND_SQUARE_FEET_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 LAND SQUARE FEET 는 서로 다르다

- GROSS SQUARE FEET

```
# GROSS SQUARE FEET t-검정

data_GROSS_SQUARE_FEET_1 =
interval_variable[categorical_variable['PRICE_B'] == 1]['GROSS SQUARE
FEET']

data_GROSS_SQUARE_FEET_0 =
interval_variable[categorical_variable['PRICE_B'] == 0]['GROSS SQUARE
FEET']

stats.ttest_ind(data_GROSS_SQUARE_FEET_1, data_GROSS_SQUARE_FEET_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 GROSS SQUARE FEET 는 서로 다르다

- YEAR BUILT

```
# YEAR BUILT t-검정

data_YEAR_BUILT_1 = interval_variable[categorical_variable['PRICE_B']
== 1]['YEAR BUILT']

data_YEAR_BUILT_0 = interval_variable[categorical_variable['PRICE_B']
== 0]['YEAR BUILT']

stats.ttest_ind(data_YEAR_BUILT_1, data_YEAR_BUILT_0)
```

pvalue 가 0.05 보다 작으므로 귀무가설 기각

두 그룹의 YEAR BUILT 는 서로 다르다

```

import pandas as pd

new_column_names = {
    'BOROUGH' : 'BOROUGH',
    'BUILDING CLASS CATEGORY' : 'BUILDING_CLASS_CATEGORY',
    'BUILDING CLASS AT PRESENT': 'BUILDING_CLASS_AT_PRESENT',
    'TOTAL UNITS': 'TOTAL_UNITS',
    'LAND SQUARE FEET': 'LAND_SQUARE_FEET',
    'GROSS SQUARE FEET': 'GROSS_SQUARE_FEET',
    'YEAR BUILT': 'YEAR_BUILT',
    'TAX CLASS AT TIME OF SALE': 'TAX_CLASS_AT_TIME_OF_SALE',
    'SALE PRICE': 'SALE_PRICE',
    'PRICE_B' : 'PRICE_B'
}

df = preprocessing_completed.rename(columns=new_column_names)

df.info()

df.to_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/NYC
Property Sales/preprocessing_completed.csv', index=False)

```

4.5 머신러닝 모델 수립

결정 트리, 로지스틱 회귀, 신경망 모델, K-최근접 이웃, 랜덤 포레스트, 그레이디언트 부스팅, Lasso, SVM, XGBoost, LightBGM, 앙상블 등 여러 모델을 사용하여 가장 높은 정확도를 가지고 있는 모델 판별해서 분석에 적용할 계획이다.

4.6 머신러닝 모델 실행

4.6.1 결정 트리 분류 모델

```
!pip install -U imbalanced-learn  
!pip install graphviz  
  
# OrdinalEncoder 를 import 하여 범주형 데이터 변환  
  
df_decisiontree = pd.read_csv('/content/drive/MyDrive/Colab  
Notebooks/Data_Analysis/NYC_Property  
Sales/preprocessing_completed.csv')  
  
from sklearn.preprocessing import OrdinalEncoder  
  
df_decisiontree['BOROUGH_encoded'] =  
OrdinalEncoder().fit_transform(df_decisiontree['BOROUGH'].values.reshape(-1,1)) + 1  
  
df_decisiontree['BUILDING_CLASS_CATEGORY_encoded'] =  
OrdinalEncoder().fit_transform(df_decisiontree['BUILDING_CLASS_CATEGORY  
'].values.reshape(-1,1))  
  
df_decisiontree['BUILDING_CLASS_AT_PRESENT_encoded'] =  
OrdinalEncoder().fit_transform(df_decisiontree['BUILDING_CLASS_AT_PRESE  
NT'].values.reshape(-1,1))  
  
df_decisiontree['TAX_CLASS_AT_TIME_OF_SALE_encoded'] =  
OrdinalEncoder().fit_transform(df_decisiontree['TAX_CLASS_AT_TIME_OF_SA  
LE'].values.reshape(-1,1)) + 1  
  
df_decisiontree.head(3)  
  
df_decisiontree.groupby(['BOROUGH', 'BOROUGH_encoded']).size()  
  
df_decisiontree.groupby(['BUILDING_CLASS_CATEGORY',  
'BUILDING_CLASS_CATEGORY_encoded']).size()  
  
df_decisiontree.groupby(['BUILDING_CLASS_AT_PRESENT',  
'BUILDING_CLASS_AT_PRESENT_encoded']).size()  
  
df_decisiontree.groupby(['TAX_CLASS_AT_TIME_OF_SALE',  
'TAX_CLASS_AT_TIME_OF_SALE_encoded']).size()
```

범주형 데이터 변환 결과

변수명	데이터 정의
BOROUGH_encoded (위치)	<ul style="list-style-type: none"> 1. Manhattan 2. The Bronx 3. Brooklyn 4. Queens 5. Staten Island
BUILDING_CLASS_CATEGORY_encoded (빌딩 종류)	<ul style="list-style-type: none"> 0. 01 ONE FAMILY DWELLINGS 1. 02 TWO FAMILY DWELLINGS 2. 03 THREE FAMILY DWELLINGS 3. 04 TAX CLASS 1 CONDOS 4. 05 TAX CLASS 1 VACANT LAND 5. 06 TAX CLASS 1 - OTHER 6. 07 RENTALS - WALKUP APARTMENTS 7. 09 COOPS - WALKUP APARTMENTS 8. 10 COOPS - ELEVATOR APARTMENTS 9. 11A CONDO-RENTALS 10. 12 CONDOS - WALKUP APARTMENTS 11. 13 CONDOS - ELEVATOR APARTMENTS 12. 15 CONDOS - 2-10 UNIT RESIDENTIAL 13. 16 CONDOS - 2-10 UNIT WITH COMMERCIAL UNIT 14. 17 CONDO COOPS 15. 21 OFFICE BUILDINGS 16. 22 STORE BUILDINGS 17. 23 LOFT BUILDINGS 18. 26 OTHER HOTELS 19. 27 FACTORIES 20. 28 COMMERCIAL CONDOS 21. 29 COMMERCIAL GARAGES 22. 30 WAREHOUSES 23. 31 COMMERCIAL VACANT LAND 24. 32 HOSPITAL AND HEALTH FACILITIES 25. 33 EDUCATIONAL FACILITIES 26. 35 INDOOR PUBLIC AND CULTURAL FACILITIES 27. 37 RELIGIOUS FACILITIES 28. 38 ASYLUMS AND HOMES

	29. 41 TAX CLASS 4 - OTHER 30. 42 CONDO CULTURAL/MEDICAL/EDUCATIONAL/ETC 31. 43 CONDO OFFICE BUILDINGS 32. 44 CONDO PARKING 33. 46 CONDO STORE BUILDINGS 34. 47 CONDO NON-BUSINESS STORAGE 35. 48 CONDO TERRACES/GARDENS/CABANAS 36. 49 CONDO WAREHOUSES/FACTORY/INDUS
BUILDING_CLASS_AT_PRESENT_encoded (건물 분류)	0. A0 1. A1 2. A2 3. A3 4. A4 5. A5 6. A6 7. A7 8. A9 9. B1 10. B2 11. B3 12. B9 13. C0 14. C6 15. C8 16. D0 17. D4 18. E1 19. E2 20. E9 21. F1 22. F2 23. F4 24. F5 25. F9 26. G0 27. G1

- | | |
|--|--------|
| | 28. G2 |
| | 29. G4 |
| | 30. G5 |
| | 31. G6 |
| | 32. G7 |
| | 33. G9 |
| | 34. GU |
| | 35. GW |
| | 36. H8 |
| | 37. HR |
| | 38. I5 |
| | 39. I9 |
| | 40. K1 |
| | 41. K2 |
| | 42. K3 |
| | 43. K4 |
| | 44. K5 |
| | 45. K9 |
| | 46. L9 |
| | 47. M1 |
| | 48. M3 |
| | 49. M4 |
| | 50. M9 |
| | 51. N2 |
| | 52. N9 |
| | 53. O1 |
| | 54. O2 |
| | 55. O3 |
| | 56. O5 |
| | 57. O7 |
| | 58. O8 |
| | 59. P2 |
| | 60. P5 |
| | 61. P9 |
| | 62. R1 |
| | 63. R2 |
| | 64. R3 |
| | 65. R4 |
| | 66. R5 |

	67. R6 68. R8 69. R9 70. RA 71. RB 72. RG 73. RK 74. RP 75. RR 76. RS 77. RT 78. RW 79. S0 80. S1 81. S2 82. V0 83. V1 84. W2 85. W3 86. W8 87. W9 88. Z0 89. Z9
TAX_CLASS_AT_TIME_OF_SALE_encoded (판매 시점의 세무급)	1. 1 2. 2 3. 4

```
# 기존 범주형 변수 열 삭제
```

```
df_decisiontree.drop(['BOROUGH', 'BUILDING_CLASS_CATEGORY',
'BUILDING_CLASS_AT_PRESENT', 'TAX_CLASS_AT_TIME_OF_SALE'], axis=1,
inplace=True)

df_decisiontree.info()
```

```
# 타겟 변수 분할 및 비율 확인

import graphviz

import pandas as pd

import numpy as np

print("df_decisiontree shape : " + str(df_decisiontree.shape))

data = df_decisiontree.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_decisiontree['PRICE_B']

print("data shape : ", data.shape)

print("target shape : ", target.shape)

df_decisiontree['PRICE_B'].value_counts(dropna=False, normalize=True)
```

```
# 7:3 비율로 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)

print("y_train shape : ", y_train.shape)

print("y_test shape : ", y_test.shape)

y_train.value_counts(normalize=True)
```

```

# 결정 트리 모델(지니 기준)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

# Classifier로 DecisionTreeClassifier를 지정

tree = DecisionTreeClassifier(random_state=4)

# Classifier를 학습 데이터세트에서 학습시킴

model = tree.fit(x_train, y_train)

# 학습된 Classifier로 테스트데이터세트의 자료를 이용해서 타겟 변수 예측값을 생성

pred = model.predict(x_test)

print("Accuracy on training set : {:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set : {:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch를 실행하기 전 정확도 : 0.64599

```

# 결정 트리 모델(지니 기준)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.model_selection import GridSearchCV

tree = DecisionTreeClassifier(criterion="gini", random_state=4,
max_depth=5)

params = {'criterion':['gini', 'entropy'], 'max_depth':range(1, 21)}

grid_tree = GridSearchCV(
    tree, param_grid=params, scoring='accuracy', cv=5, n_jobs=-1,
verbose=1)

grid_tree.fit(x_train, y_train)

print("GridSearchCV max accuracy :
{:.5f}".format(grid_tree.best_score_))

print("GridSearchCV best parameter:", (grid_tree.best_params_))

```

```

best_clf = grid_tree.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.67168

bets parameter : {'criterion': 'entropy', 'max_depth': 8}

최적 모델의 변수 중요도 수치 확인

```

print("Feature importances:")

print(best_clf.feature_importances_)

```

변수명을 리스트 형태로 만들기

```

feature_names = list(data.columns)

# 변수명을 index로 만들고 feature_importances를 매칭해서 나열한 데이터프레임
만들기

dft = pd.DataFrame(np.round(best_clf.feature_importances_, 4),
index=feature_names, columns=['Feature_importances'])

# Feature_importances의 값을 내림차순으로 정리

dft1 = dft.sort_values(by='Feature_importances', ascending=False)

dft1

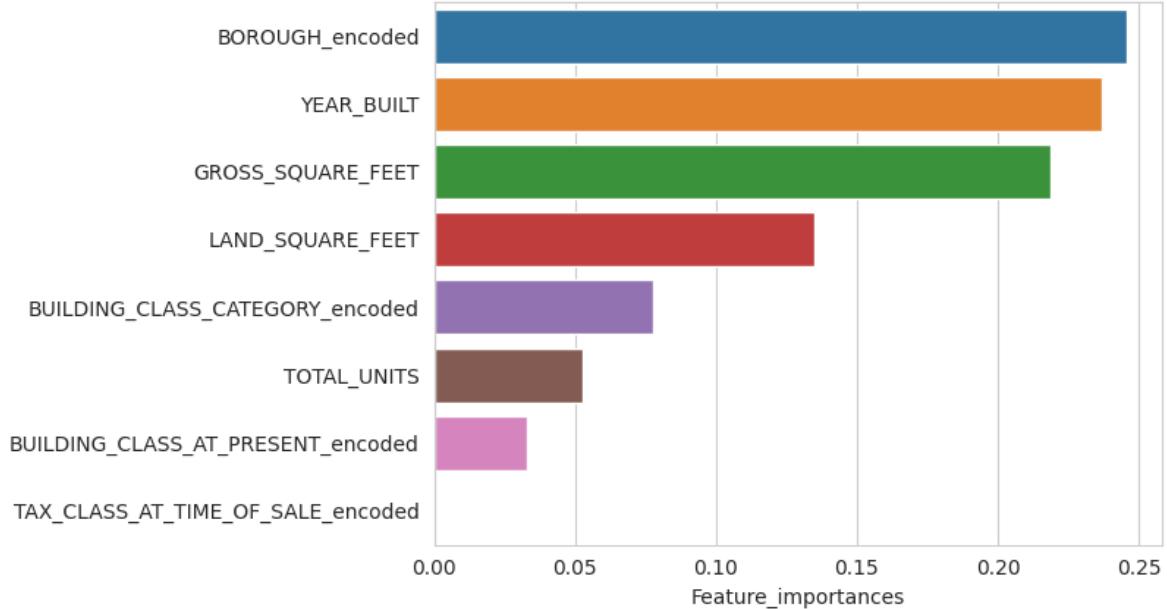
```

Feature_importances	
BOROUGH_encoded	0.2458
YEAR_BUILT	0.2370
GROSS_SQUARE_FEET	0.2189
LAND_SQUARE_FEET	0.1348
BUILDING_CLASS_CATEGORY_encoded	0.0777
TOTAL_UNITS	0.0528
BUILDING_CLASS_AT_PRESENT_encoded	0.0330
TAX_CLASS_AT_TIME_OF_SALE_encoded	0.0000

```
# 데이터프레임 df1 의 막대그래프 그리기

import seaborn as sns

sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)
```



```
# graphviz 불러오기

import graphviz

# model의 결과물을 tree.dot에 저장

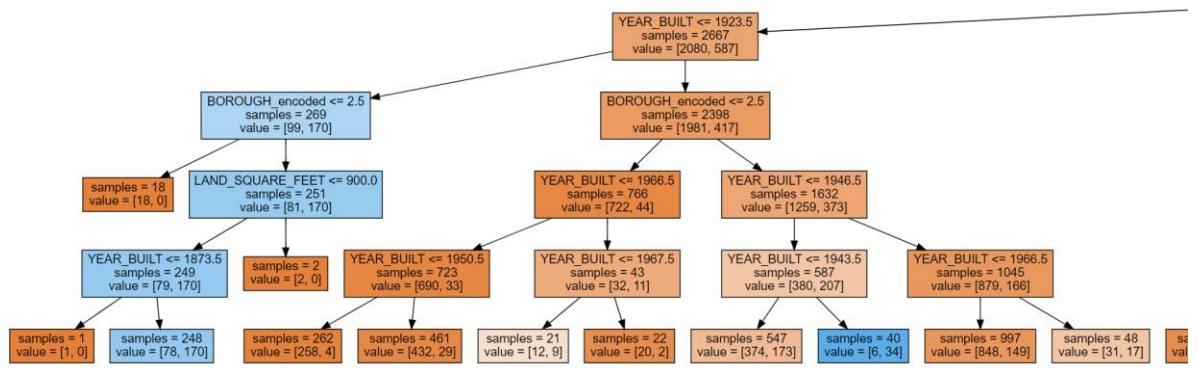
from sklearn.tree import export_graphviz

export_graphviz(best_clf, out_file="tree.dot",
feature_names=list(data.columns), impurity=False, filled=True)

# tree.dot 을 graphviz 기능을 통해 출력

with open("tree.dot") as f:
    dot_graph = f.read()

display(graphviz.Source(dot_graph))
```



4.6.2 로지스틱 회귀 분류 모델

```
import pandas as pd

df_logisticregression = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC_Property
Sales/preprocessing_completed.csv')

print(df_logisticregression.shape)

df_logisticregression.head()
```

```
# 범주형 변수 더미 변수 생성

col = ['BOROUGH', 'BUILDING_CLASS_CATEGORY',
'BUILDING_CLASS_AT_PRESENT', 'TAX_CLASS_AT_TIME_OF_SALE']

df_logisticregression = pd.get_dummies(df_logisticregression,
columns=col)

df_logisticregression.head()
```

```
# 기준 더미 변수 제거

col = ['BOROUGH_3', 'BUILDING_CLASS_CATEGORY_01 ONE FAMILY
DWELLINGS', 'BUILDING_CLASS_AT_PRESENT_A1',
'TAX_CLASS_AT_TIME_OF_SALE_1']

df_logisticregression.drop(col, axis=1, inplace=True)

df_logisticregression.shape
```

```
# 타겟 변수 설정

data = df_logisticregression.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_logisticregression['PRICE_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```
# 로지스틱 회귀 기본 모델
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

lr = LogisticRegression(solver='lbfgs', penalty='none', random_state=4,
n_jobs=-1)

model = lr.fit(x_train, y_train)

pred = model.predict(x_test)

print("Training set score{:.5f}".format(model.score(x_train, y_train)))

print("Test set score{:.5f}".format(accuracy_score(y_test, pred)))
```

```
GridSearch 를 실행하기 전 정확도 : 0.62818
```

```
lr = LogisticRegression(solver='lbfgs', penalty='none', random_state=4,
n_jobs=-1)

from sklearn.model_selection import GridSearchCV

params = {'solver':['lbfgs', 'saga'], 'penalty':['none']}

grid_lr = GridSearchCV(lr, param_grid=params, scoring='accuracy', cv=5,
n_jobs=-1)

grid_lr.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_lr.best_score_))

print("GridSearchcv best parameter:", (grid_lr.best_params_))
```

```
best_clf = grid_lr.best_estimator_
```

```
pred = best_clf.predict(x_test)
```

```
print("accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

```
GridSearch 를 실행한 후 데스트 데이터세트의 정확도 : 0.62818
```

```
bets parameter : {'penalty': 'none', 'solver': 'lbfgs'}
```

```
# 변수명을 index로 만들고 coefficient 값을 매칭한 데이터프레임 만들기

import numpy as np

feature_names = list(data.columns)

dft = pd.DataFrame(np.round(best_clf.coef_, 3).transpose(),
index=feature_names, columns=['coef'])

# coef 값을 오름차순으로 정리

dft1 = dft.sort_values(by='coef', ascending=True)

dft1
```

	coef
BOROUGH_2	-0.951
BUILDING_CLASS_AT_PRESENT_D4	-0.453
TAX_CLASS_AT_TIME_OF_SALE_4	-0.449
BUILDING_CLASS_CATEGORY_10 COOPS - ELEVATOR APARTMENTS	-0.448
BUILDING_CLASS_CATEGORY_44 CONDO PARKING	-0.340
...	...
BUILDING_CLASS_AT_PRESENT_R1	0.469
BUILDING_CLASS_AT_PRESENT_R4	0.780
BOROUGH_4	0.784
BUILDING_CLASS_CATEGORY_13 CONDOS - ELEVATOR APARTMENTS	0.785
TAX_CLASS_AT_TIME_OF_SALE_2	1.042

135 rows × 1 columns

```

# coefficient 값을 제곱한 오즈비값을 index에 매칭한 데이터프레임 만들기

feature_names = list(data.columns)

dft = pd.DataFrame(np.round(np.exp(best_clf.coef_), 3).transpose(),
index=feature_names, columns=['Odds_ratio'])

# coef 를 내림차순으로 정리

dft1 = dft.sort_values(by='Odds_ratio', ascending=False)

dft1

```

	Odds_ratio
TAX_CLASS_AT_TIME_OF_SALE_2	2.835
BUILDING_CLASS_CATEGORY_13 CONDOS - ELEVATOR APARTMENTS	2.193
BOROUGH_4	2.189
BUILDING_CLASS_AT_PRESENT_R4	2.181
BUILDING_CLASS_AT_PRESENT_R1	1.598
...	...
BUILDING_CLASS_CATEGORY_44 CONDO PARKING	0.712
BUILDING_CLASS_CATEGORY_10 COOPS - ELEVATOR APARTMENTS	0.639
TAX_CLASS_AT_TIME_OF_SALE_4	0.638
BUILDING_CLASS_AT_PRESENT_D4	0.635
BOROUGH_2	0.386

135 rows × 1 columns

```

# 데이터프레임 dft1 의 막대그래프 그리기

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

plt.figure(figsize=(15, 20))

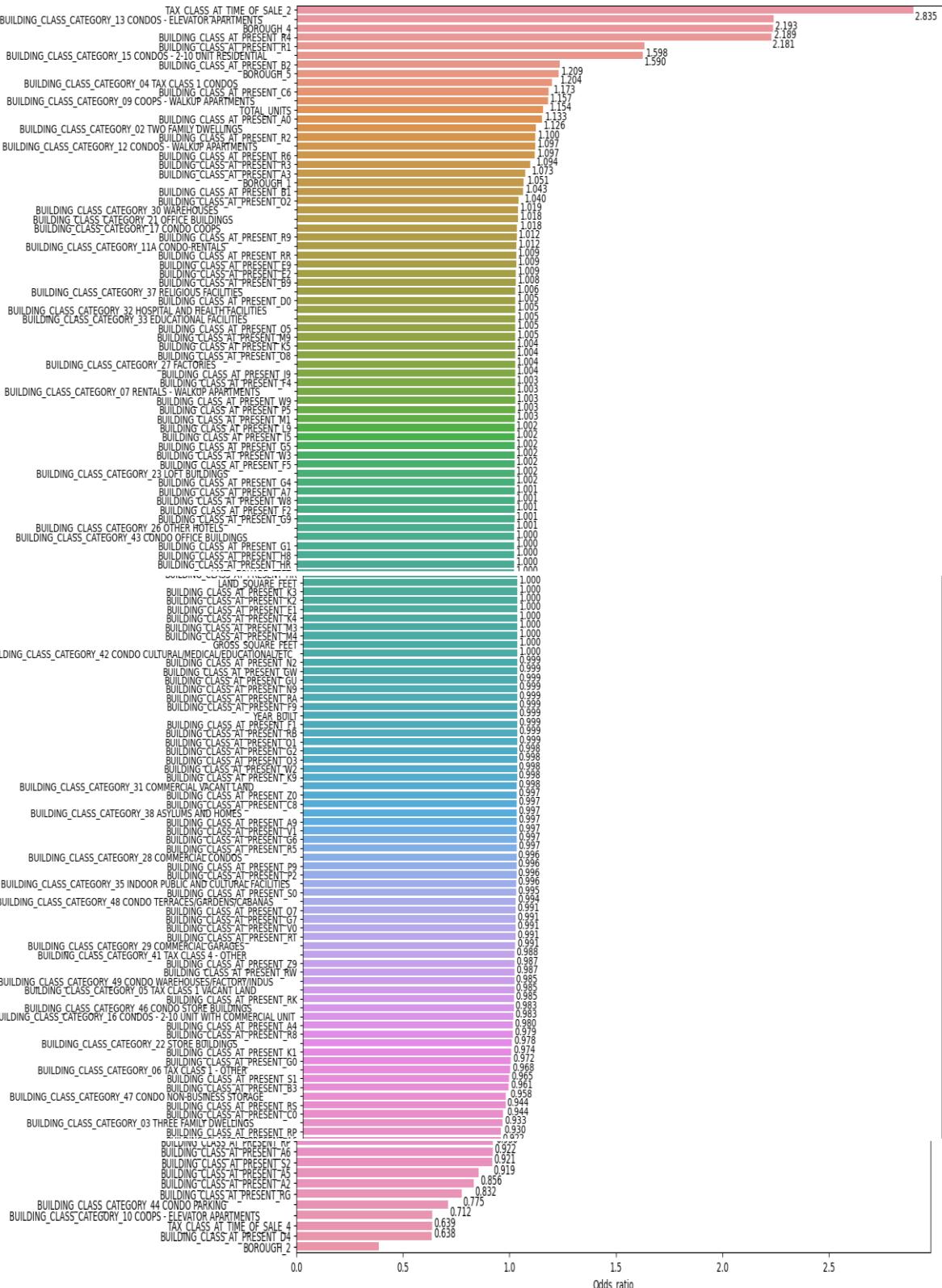
ax = sns.barplot(y=dft1.index, x='Odds_ratio', data=dft1)

for p in ax.patches:
    ax.annotate("%." + str(3) + "f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y() + 2.0), xytext=(2, 4), textcoords='offset points')

ax.set_yticklabels(ax.get_yticklabels(), fontsize=10)

plt.show()

```



오즈비 해석

- 구간 변수

총 단위 수가 1 늘어날 경우 부동산 가격이 중위값보다 비쌀 확률은 13.3% 증가한다.

토지 면적과 건물 총 면적은 부동산 가격과 관련이 없다.

건물이 지어진 연도가 1년 증가할 경우 부동산 가격이 중위값보다 비쌀 확률은 0.1% 감소한다.

- 범주형 변수

판매 시점의 세무급이 3 가구 이내의 주거용재산인 경우와 비교하여 판매 시점의 세무급이 주거용인 기타 모든 부동산인 경우가 부동산 가격이 중위값보다 비쌀 가능성은 2.835 배 높다.

빌딩의 종류가 01 ONE FAMILY DWELLINGS인 경우와 비교하여 빌딩의 종류가 13 CONDOS - ELEVATOR APARTMENTS인 경우가 부동산 가격이 중위값보다 비쌀 가능성은 2.193 배 높다.

부동산 위치가 Brooklyn인 경우와 비교하여 부동산 위치가 Queens인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 2.189 배 높다.

부동산 위치가 Brooklyn인 경우와 비교하여 부동산 위치가 Bronx인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 0.386 배 낮다.

건물 분류가 A1인 경우와 비교하여 건물 분류가 D4인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 0.635 배 낮다.

판매 시점의 세무급이 3 가구 이내 주거용 재산인 경우와 비교하여 판매시점의 세무급이 사무실, 공장, 차고, 건물 등인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 0.638 배 낮다.

4.6.3 표준화한 로지스틱 회귀 모델

```
# 구간 변수들만 별도로 모아 데이터프레임 df_num 을 만든다

numeric_cols = ['TOTAL_UNITS', 'LAND_SQUARE_FEET', 'GROSS_SQUARE_FEET',
'YEAR_BUILT']

df_num = df_logisticregression[numeric_cols]

# StandardScaler()로 데이터 스케일을 표준화하고, 결과를 데이터 프레임으로 만든다.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_num_standard = pd.DataFrame(scaler.fit_transform(df_num))

# StandardScaler()는 변수명을 지우므로 데이터프레임에 다시 변수명을 넣는다

df_num_standard.columns = df_num.columns

df_num_standard.head()
```

```
# 원래 데이터프레임에서 구간 변수들을 제거하여 df_cat 에 저장

df_cat = df_logisticregression.drop(numeric_cols, axis=1)

# 구간 변수 스케일을 표준화한 df_num_standard 와 범주형 변수만 담은 df_cat 을 병합

dfu_standard = pd.concat([df_num_standard, df_cat], axis=1)

# dfu 의 변수명을 나열

dfu_standard.columns
```

```
# 표준화한 데이터세트로 로지스틱 회귀 재실행

data = dfu_standard.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = dfu_standard['PRICE_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```

# 로지스틱 회귀 기본 모델

lr = LogisticRegression(solver='lbfgs', penalty='none', random_state=4,
n_jobs=-1)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

paramas = {'solver':['lbfgs', 'saga'], 'penalty':['none']}

grid_lr = GridSearchCV(lr, param_grid=paramas, scoring='accuracy', cv=5,
n_jobs=-1)

grid_lr.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_lr.best_score_))

print("GridSearchCV best parameter:", grid_lr.best_params_)

best_clf = grid_lr.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch를 실행한 후 테스트 데이터셋의 정확도 : 0.63883

bets parameter : {'penalty': 'none', 'solver': 'saga'}

```

dfu_standard.to_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/standard-scaled.csv',
index=False)

```

4.6.4 사이킷런 신경망 분류 모델

```
import pandas as pd

import numpy as np

df_neural_network_knn = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/NYC Property Sales/standard-scaled.csv')

data = df_neural_network_knn.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_neural_network_knn['PRICE_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```
# 신경망 기본 모델

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score

clf_mlp = MLPClassifier(max_iter=2000, random_state=4)
clf_mlp.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수의 예측값 생성

pred = clf_mlp.predict(x_test)

accuracy = accuracy_score(y_test, pred)

print("Training set score:{:.5f}".format(clf_mlp.score(x_train,
y_train)))

print("Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```

# 신경망 기본 모델

clf_mlp = MLPClassifier(max_iter=500, random_state=4)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {
    'solver': ['sgd', 'lbfgs', 'adam'],
    'activation': ['tanh', 'relu', 'logistic'],
    'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
}

grid_mlp = GridSearchCV(clf_mlp, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1, error_score='raise')

grid_mlp.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_mlp.best_score_))

print("GridSearchCV best parameter:", (grid_mlp.best_params_))

```

```

best_clf = grid_mlp.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

```

GridSearch 를 실행한 후 테스트 데이터세트의 정확도 : 0.67365

bets parameter : {'activation': 'relu', 'alpha': 0.0001, 'solver': 'adam'}

```

# 신경망 기본 모델

clf_mlp = MLPClassifier(max_iter=500, random_state=4)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {
    'solver': ['adam'],
    'activation': ['relu'],
    'alpha': [0.0001],
}

grid_mlp = GridSearchCV(clf_mlp, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1, error_score='raise')

grid_mlp.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_mlp.best_score_))

print("GridSearchCV best parameter:", (grid_mlp.best_params_))

```

4.6.5 최근접 이웃 분류 모델(KNN)

```
import pandas as pd

import numpy as np

df_knn = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/standard-scaled.csv')

data = df_knn.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_knn['PRICE_B']

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)

print("x_train shape : ", x_train.shape)

print("x_test shape : ", x_test.shape)
```

```
# KNN 모델

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

clf_knn = KNeighborsClassifier(n_neighbors=3)

clf_knn.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타겟 변수 예측값을 생성

pred = clf_knn.predict(x_test)

accuracy = accuracy_score(y_test, pred)

print("Training set score:{:.5f}".format(clf_knn.score(x_train,
y_train)))

print("Test set score:{:.5f}".format(accuracy_score(y_test, pred)))
```

```
# KNN 모델

clf_knn = KNeighborsClassifier(n_neighbors=3)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {'n_neighbors' : range(3, 31)}

grid_knn = GridSearchCV(clf_knn, param_grid=params, scoring='accuracy',
cv=3, n_jobs=-1)

grid_knn.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_knn.best_score_))

print("GridSearchCV best parameter:", (grid_knn.best_params_))
```

```
best_clf = grid_knn.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.67633

bets parameter : {'n_neighbors': 28}

4.6.6 랜덤 포레스트

```
import pandas as pd

# OrdinalEncoder 를 import 하여 범주형 데이터 변환

df_randomforest = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC_Property
Sales/preprocessing_completed.csv')

from sklearn.preprocessing import OrdinalEncoder

df_randomforest['BOROUGH_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['BOROUGH'].values.reshape(-1,1)) + 1

df_randomforest['BUILDING_CLASS_CATEGORY_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['BUILDING_CLASS_CATEGORY
'].values.reshape(-1,1))

df_randomforest['BUILDING_CLASS_AT_PRESENT_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['BUILDING_CLASS_AT_PRESENT
'].values.reshape(-1,1))

df_randomforest['TAX_CLASS_AT_TIME_OF_SALE_encoded'] =
OrdinalEncoder().fit_transform(df_randomforest['TAX_CLASS_AT_TIME_OF_SALE
'].values.reshape(-1,1)) + 1

df_randomforest.head(3)
```

```
df_randomforest.groupby(['BOROUGH', 'BOROUGH_encoded']).size()
```

```
df_randomforest.groupby(['BUILDING_CLASS_CATEGORY',
'BUILDING_CLASS_CATEGORY_encoded']).size()
```

```
df_randomforest.groupby(['BUILDING_CLASS_AT_PRESENT',
'BUILDING_CLASS_AT_PRESENT_encoded']).size()
```

```
df_randomforest.groupby(['TAX_CLASS_AT_TIME_OF_SALE',
'TAX_CLASS_AT_TIME_OF_SALE_encoded']).size()
```

```
# 기존 범주형 변수 열 삭제

df_randomforest.drop(['BOROUGH', 'BUILDING_CLASS_CATEGORY',
'BUILDING_CLASS_AT_PRESENT', 'TAX_CLASS_AT_TIME_OF_SALE'], axis=1,
inplace=True)

df_randomforest.info()
```

```
df_randomforest.head()
```

```
df_randomforest.to_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/encoded.csv', index=False)
```

```
# 범주형 변수를 cols에 저장

cols = ['BOROUGH_encoded', 'BUILDING_CLASS_CATEGORY_encoded',
'BUILDING_CLASS_AT_PRESENT_encoded',
'TAX_CLASS_AT_TIME_OF_SALE_encoded', 'PRICE_B']

# 범주형 변수의 dtype을 category로 변경

df_randomforest[cols] = df_randomforest[cols].astype('category')

df_randomforest.dtypes
```

```
data = df_randomforest.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_randomforest['PRICE_B']

print("data shape : ", data.shape)

print("target shape : ", target.shape)
```

```
# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)
```

```
# 랜덤 포레스트 모델(기본 모델, tree depth 제한 없음)

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(n_estimators=100, random_state=4)

model = rf.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수 예측값 생성

pred = rf.predict(x_test)

print("Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

```
# 랜덤 포레스트 모델(기본 모델, tree depth 제한 없음)

rf = RandomForestClassifier(n_estimators=100, random_state=4)

from sklearn.model_selection import GridSearchCV

params = {"max_depth" : range(10, 41), 'n_estimators' : [100,200]}

grid_rf = GridSearchCV(rf, param_grid=params, scoring='accuracy', cv=5,
n_jobs=-1)

grid_rf.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_rf.best_score_))

print("GridSearchCV best parameter:", (grid_rf.best_params_))
```

```
best_clf = grid_rf.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:, 1])

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.68188
```

```
ROC AUC 값 : 0.76077
```

```
bests parameter : {'max_depth': 17, 'n_estimators': 200}
```

```
print("Feature importances:")
```

```
print(best_clf.feature_importances_)
```

```
import numpy as np
```

```
# 변수명을 index로 만들고 feature_importances를 매칭해서 나열한 데이터프레임 만들기
```

```
feature_names = list(data.columns)
```

```
dft = pd.DataFrame(np.round(best_clf.feature_importances_, 3),  
index=feature_names, columns=['Feature_importances'])
```

```
# Feature_importances의 값을 내림차순으로 정리
```

```
dft1 = dft.sort_values(by='Feature_importances', ascending=False)
```

```
dft1
```

Feature_importances

YEAR_BUILT	0.215
GROSS_SQUARE_FEET	0.179
LAND_SQUARE_FEET	0.165
BOROUGH_4	0.105
BOROUGH_2	0.064
...	...
BUILDING_CLASS_AT_PRESENT_F4	0.000
BUILDING_CLASS_AT_PRESENT_F2	0.000
BUILDING_CLASS_AT_PRESENT_F1	0.000
BUILDING_CLASS_AT_PRESENT_E9	0.000
BUILDING_CLASS_AT_PRESENT_F5	0.000

```
135 rows × 1 columns
```

```
# 데이터프레임 dft1 의 막대그래프 (barplot) 그리기
```

```
import matplotlib.pyplot as plt

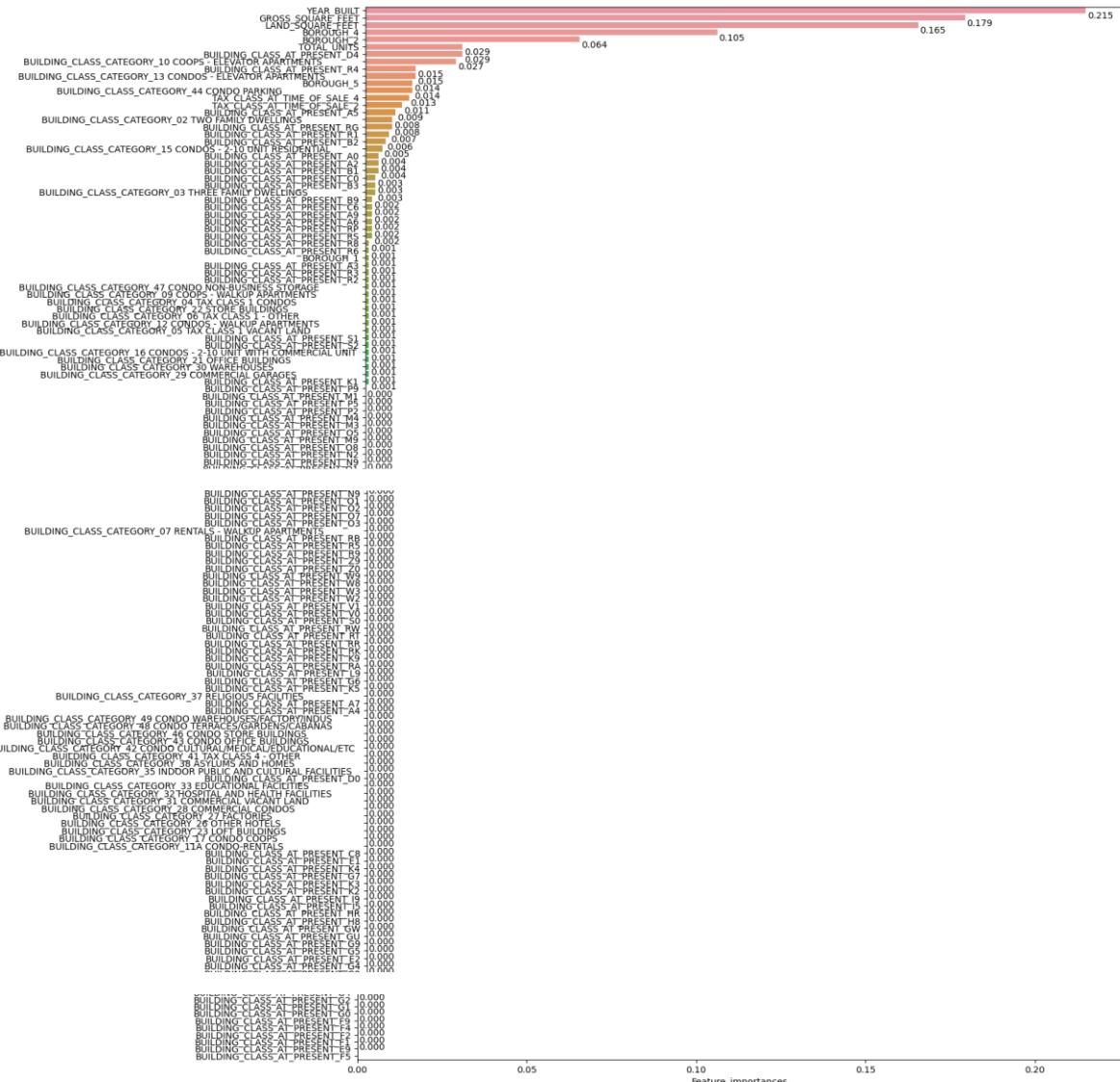
import seaborn as sns

%matplotlib inline

fig, ax = plt.subplots(figsize=(15, 20))

ax = sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y() + 2.0), xytext=(2, 4), textcoords='offset points')
```



4.6.7 그레디언트 부스팅 모델

```
import pandas as pd

df_gradientboosting = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/encoded.csv')
```

```
# 범주형 변수를 cols에 저장

cols = ['BOROUGH_encoded', 'BUILDING_CLASS_CATEGORY_encoded',
'BUILDING_CLASS_AT_PRESENT_encoded',
'TAX_CLASS_AT_TIME_OF_SALE_encoded', 'PRICE_B']

# 범주형 변수의 dtype을 category로 변경

df_gradientboosting[cols] =
df_gradientboosting[cols].astype('category')

df_gradientboosting.dtypes
```

```
data = df_gradientboosting.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_gradientboosting['PRICE_B']

print("data shape : ", data.shape)

print("target shape : ", target.shape)
```

```
# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4, stratify=target)
```

```
# 그레디언트 부스팅 모델 (기본 모델)

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score

gr = GradientBoostingClassifier(random_state=4)

model = gr.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수 예측값 생성

pred = model.predict(x_test)

accuracy = accuracy_score(y_test, pred)

print("Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

```
gr = GradientBoostingClassifier(random_state=4)

from sklearn.model_selection import GridSearchCV

params = {'max_depth':range(11, 16), 'n_estimators':[100, 200],
'learning_rate':[0.01, 0.1]}

grid_gr = GridSearchCV(model, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1)
grid_gr.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_gr.best_score_))
print("GridSearchCV best parameter:", (grid_gr.best_params_))
```

```
best_clf = grid_gr.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:,1])

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.68090
```

```
ROC AUC 값 : 0.75943
```

```
bets parameter : {'learning_rate': 0.01, 'max_depth': 11,  
'n_estimators': 200}
```

```
# 데이터프레임의 행과 열 전체를 보이게 하는 조치
```

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

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

```
import numpy as np
```

```
# 변수명을 index로 만들고 feature_importances를 매칭해서 나열한 데이터프레임  
만들기
```

```
feature_names = list(data.columns)
```

```
dft = pd.DataFrame(np.round(best_clf.feature_importances_, 3),  
index=feature_names, columns=['Feature_importances'])
```

```
dft1 = dft.sort_values(by='Feature_importances', ascending=False)
```

```
# 데이터프레임 dft1의 막대그래프 그리기
```

```
import matplotlib.pyplot as plt
```

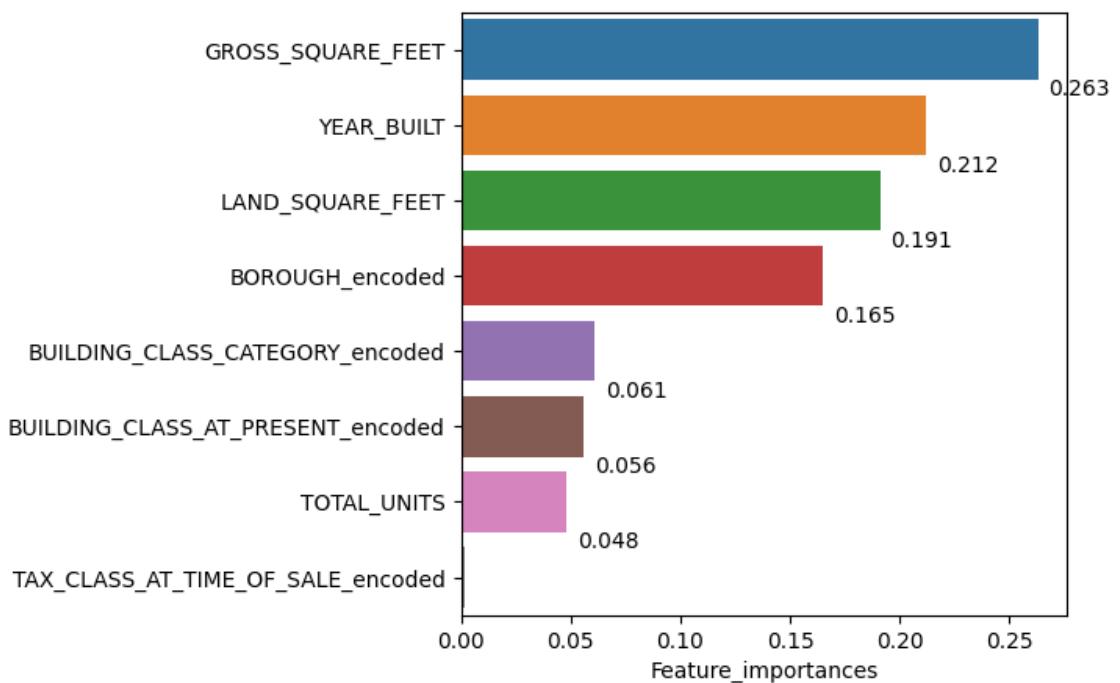
```
import seaborn as sns
```

```
%matplotlib inline
```

```
fig, ax = plt.subplots(figsize=(5, 5))
```

```
ax = sns.barplot(y=dft1.index, x="Feature_importances", data=dft1)
```

```
for p in ax.patches:  
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),  
p.get_y() + 1.3), xytext=(5, 10), textcoords='offset points')
```



4.6.8 라쏘 모델

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/encoded.csv')

df.info()

cols = ['TOTAL_UNITS', 'LAND_SQUARE_FEET', 'GROSS_SQUARE_FEET',
'YEAR_BUILT', 'SALE_PRICE', 'PRICE_B']

df_cat = df.drop(cols, axis=1) # 구간 변수 및 타겟 변수 제외

df_cat.shape

pd.options.display.float_format = '{:.2f}'.format # 소수점 2 자리로 숫자
표기 제한

df_cat.describe()

df_cat.max() - df_cat.min()

df_cat.min()

cols1 = ['BOROUGH_encoded', 'BUILDING_CLASS_CATEGORY_encoded',
'BUILDING_CLASS_AT_PRESENT_encoded',
'TAX_CLASS_AT_TIME_OF_SALE_encoded']

df_lasso = pd.get_dummies(df, columns=cols1)

df_lasso.shape

list(df_lasso.columns)
```

```
# 기준 더미변수로 정한 4 개의 더미변수명을 cols2에 저장

cols2 = ['BOROUGH_encoded_3.0', 'BUILDING_CLASS_CATEGORY_encoded_0.0',
'BUILDING_CLASS_AT_PRESENT_encoded_1.0',
'TAX_CLASS_AT_TIME_OF_SALE_encoded_1.0']

df_lasso.drop(cols2, axis=1, inplace=True) # cols2에 저장된 더미 변수명을
데이터프레임에서 제거

df_lasso.shape
```

```
data = df_lasso.drop(['SALE_PRICE', 'PRICE_B'], axis=1)

target = df_lasso['PRICE_B']

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)

print("x_train shape:", x_train.shape)

print("x_test shape:", x_test.shape)
```

```
# 라쏘 모델(liblinear를 사용한 기본 모델)

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

ls = LogisticRegression(penalty="l1", solver='liblinear', C=1,
random_state=4)

model = ls.fit(x_train, y_train)

# 학습된 Classifier로 테스트 데이터세트를 이용해서 타깃 변수 예측값 생성

pred = model.predict(x_test)

print("Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

```

# 라쏘 모델(liblinear를 사용한 기본 모델)

ls = LogisticRegression(penalty='l1', solver='liblinear', C=1,
random_state=4)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {'solver':['lbfgs', 'liblinear', 'sag', 'saga'],
'C':[0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 1, 10, 100]}

grid_ls = GridSearchCV(ls, param_grid=params, scoring='accuracy',
cv=5,n_jobs=-1)

grid_ls.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_ls.best_score_))

print("GridSearchCV best parameter:", (grid_ls.best_params_))

```

```

best_clf = grid_ls.best_estimator_

pred = best_clf.predict(x_test)

print("Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:,1])

print("ROC AUC on test set:{:.5f}".format(ROC_AUC))

```

GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.57223

ROC AUC 값 : 0.59019

best parameter : {'C': 0.1, 'solver': 'saga'}

```

import numpy as np

print("Number of features used:", np.sum(best_clf.coef_ != 0))

```

```

print("회귀계수", best_clf.coef_)

feature_names = list(data.columns)

# 변수 index에 coefficient 값을 매칭해 데이터프레임으로 저장

dft = pd.DataFrame(best_clf.coef_.transpose(), index=feature_names,
columns=['coef'])

dft1 = dft.sort_values(by='coef', ascending=False)

dft1

```

	coef
BOROUGH_encoded_4.0	0.00
TOTAL_UNITS	0.00
BUILDING_CLASS_CATEGORY_encoded_11.0	0.00
BUILDING_CLASS_AT_PRESENT_encoded_65.0	0.00
TAX_CLASS_AT_TIME_OF_SALE_encoded_2.0	0.00
...	...
BUILDING_CLASS_AT_PRESENT_encoded_5.0	-0.00
TAX_CLASS_AT_TIME_OF_SALE_encoded_3.0	-0.00
BUILDING_CLASS_CATEGORY_encoded_8.0	-0.00
BUILDING_CLASS_AT_PRESENT_encoded_17.0	-0.00
BOROUGH_encoded_2.0	-0.00

135 rows × 1 columns

```
# 오즈비 계산
```

```
feature_names = list(data.columns) # 변수명(컬럼명)을 리스트 형태로 만들기

dft = pd.DataFrame(np.exp(best_clf.coef_).transpose(),
index=feature_names, columns=['Odds_ratio'])

dft1 = dft.sort_values(by='Odds_ratio', ascending=False) # 컬럼 coef의
값들을 내림차순으로 정리

dft1
```

Odds_ratio	
BOROUGH_encoded_4.0	1.00
TOTAL_UNITS	1.00
BUILDING_CLASS_CATEGORY_encoded_11.0	1.00
BUILDING_CLASS_AT_PRESENT_encoded_65.0	1.00
TAX_CLASS_AT_TIME_OF_SALE_encoded_2.0	1.00
...	...
BUILDING_CLASS_AT_PRESENT_encoded_5.0	1.00
TAX_CLASS_AT_TIME_OF_SALE_encoded_3.0	1.00
BUILDING_CLASS_CATEGORY_encoded_8.0	1.00
BUILDING_CLASS_AT_PRESENT_encoded_17.0	1.00
BOROUGH_encoded_2.0	1.00

135 rows × 1 columns

```
# 오즈비가 1(계수값이 0)인 변수를 제거
```

```
dft2 = dft1[dft1['Odds_ratio'] != 1]

dft2.shape
```

```
# 데이터프레임 dft2 의 막대그래프 그리기
```

```
import matplotlib.pyplot as plt

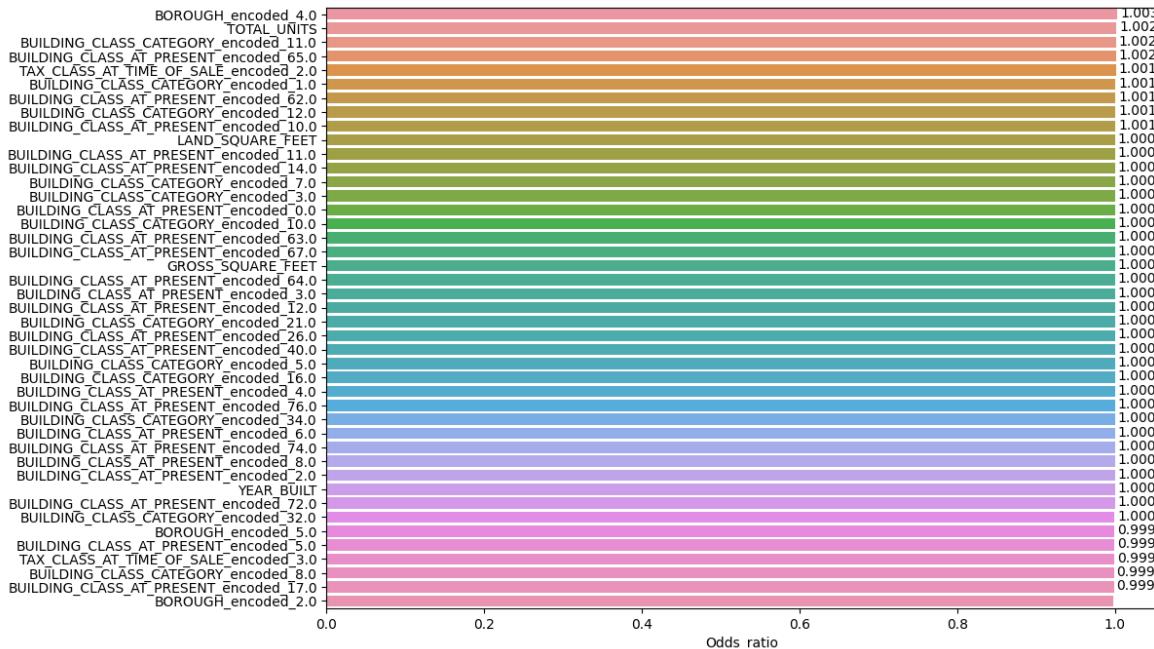
import seaborn as sns

%matplotlib inline

fig, ax= plt.subplots(figsize=(11,8))

ax = sns.barplot(y=dft2.index, x="Odds_ratio", data=dft2)

for p in ax.patches:
    ax.annotate("%.3f" % p.get_width(), (p.get_x() + p.get_width(),
p.get_y()+1.0), xytext=(2, 4), textcoords='offset points')
```



오즈비 해석

- 구간 변수

총 단위 수가 1 늘어날 경우 부동산 가격이 중위값보다 비쌀 확률은 0.3% 증가한다.

토지 면적이 1 늘어날 경우 부동산 가격이 중위값보다 비쌀 확률은 0.01% 증가한다.

건물 총면적 1 늘어날 경우 부동산 가격이 중위값보다 비쌀 확률은 0.004% 증가한다.

건물이 지어진 연도가 1년 증가할 경우 부동산 가격이 중위값보다 비쌀 확률은 0.03% 감소한다.

- 범주형 변수

부동산 위치가 Brooklyn 인 경우와 비교하여 부동산 위치가 Queens 인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 1.003 배 높다.

빌딩의 종류가 01 ONE FAMILY DWELLINGS 인 경우와 비교하여 빌딩의 종류가 13 CONDOS - ELEVATOR APARTMENTS 인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 1.001 배 높다.

건물 분류가 A1 인 경우와 비교하여 건물 분류가 R4 인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 1.001 배 높다.

부동산 위치가 Brooklyn 인 경우와 비교하여 부동산 위치가 Bronx 인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 0.997 배 낮다.

건물 분류가 A1 인 경우와 비교하여 건물 분류가 D4 인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 0.998 배 낮다.

빌딩의 종류가 01 ONE FAMILY DWELLINGS 인 경우와 비교하여 빌딩의 종류가 10 COOPS - ELEVATOR APARTMENTS 인 경우가 부동산 가격이 중위값보다 비쌀 가능성이 0.998 배 낮다.

4.6.9 텐서플로우 케라스 신경망 모델

```
import pandas as pd

df_tf_keras = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/standard-scaled.csv')

df_tf_keras.head()
```

```
df_tf_keras.shape
```

```
data = df_tf_keras.drop(['SALE_PRICE', 'PRICE_B'], axis=1)
target = df_tf_keras['PRICE_B']
```

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)

print("x_train shape: ", x_train.shape)

print("x_test shape: ", x_test.shape)
```

```
type(x_train)
```

```
x_train_np = x_train.to_numpy()

x_test_np = x_test.to_numpy()

y_train_np = y_train.to_numpy()

y_test_np = y_test.to_numpy()
```

```
print(x_train_np)
```

```
x_train_np.shape
```

```
print(y_train_np)
```

```
import tensorflow as tf

from tensorflow.keras import layers

# 활성화 함수 relu, 옵티마이저 adam

import random as python_random

import numpy as np

np.random.seed(123)

python_random.seed(123)

tf.random.set_seed(1234)
```

```
# 활성화 함수 relu
```

```
model = tf.keras.Sequential([
    layers.Dense(100, activation='relu', input_shape=[135]),
    layers.Dropout(.5),
    layers.Dense(100, activation='relu'),
    layers.Dropout(.5),
    layers.Dense(1, activation='sigmoid')
])
```

```
# 옵티마이저 adam
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'], run_eagerly=True)
```

```
from tensorflow import keras

from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint('Ch8-NN1.tf', save_best_only=True)

early_stopping_cb = keras.callbacks.EarlyStopping(patience=5,
monitor='val_accuracy', restore_best_weights=True)

history = model.fit(x_train_np, y_train_np,
                     validation_split=0.25,
                     shuffle=True,
                     epochs=30,
                     callbacks = [checkpointer, early_stopping_cb])
```

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 100)	13600
dropout_2 (Dropout)	(None, 100)	0
dense_4 (Dense)	(None, 100)	10100
dropout_3 (Dropout)	(None, 100)	0
dense_5 (Dense)	(None, 1)	101

```
=====
Total params: 23801 (92.97 KB)
Trainable params: 23801 (92.97 KB)
Non-trainable params: 0 (0.00 Byte)
=====
```

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']          # 모델의 학습 정확도를 변수 acc에 저장

val_acc = history.history['val_accuracy'] # 모델의 검증 정확도를 변수 val_acc에 저장

loss = history.history['loss']            # 모델의 학습 손실을 변수 loss에 저장

val_loss = history.history['val_loss']    # 모델의 검증 손실을 변수 val_loss에 저장

epochs_range = range(1, 10+1)

# 학습 정확도와 검증 정확도 그리기

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs_range, acc, label='Training Accuracy')

plt.plot(epochs_range, val_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title("Training and Validation Accuracy")

# 학습 손실과 검증 손실 그리기

plt.subplot(1, 2, 2)

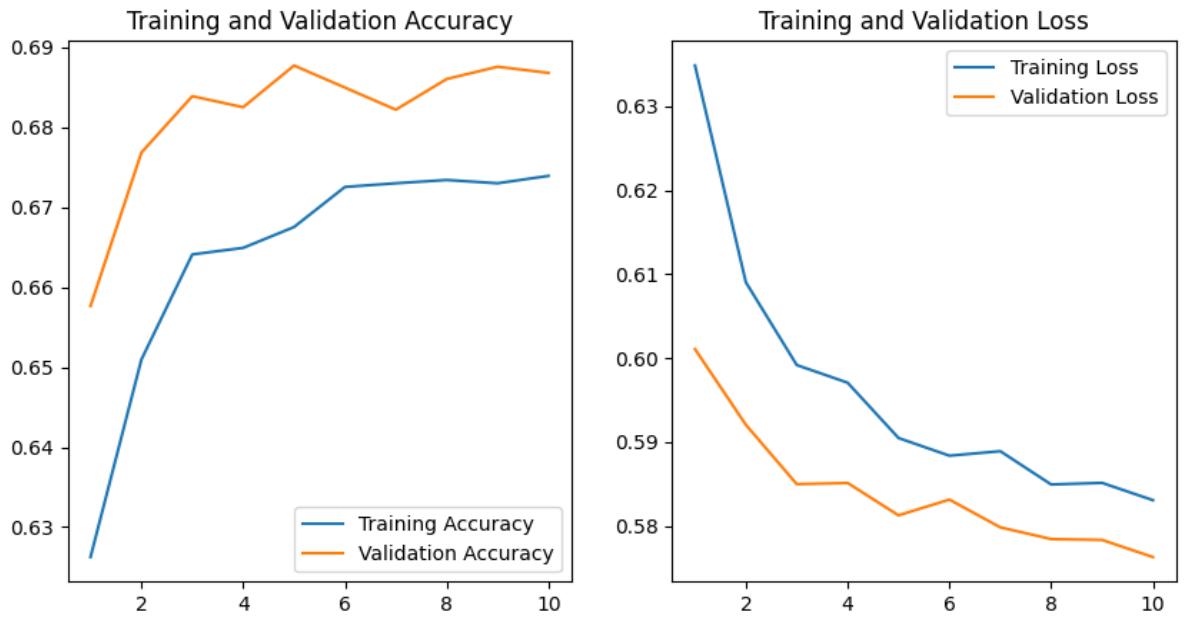
plt.plot(epochs_range, loss, label="Training Loss")

plt.plot(epochs_range, val_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title("Training and Validation Loss")

plt.show()
```



```
# model.fit() 실행시 검증 정확도가 가장 높은 에포크에 해당하는 모델 가중치 계수 불러오기
```

```
model.load_weights('/content/Ch8-NN1.tf')
```

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("test accuracy:", test_accuracy)
```

```
y_prob = model.predict(x_test)
y_prob.round(2)
```

```
from sklearn.metrics import roc_auc_score
y_prob = model.predict(x_test)
ROC_AUC = roc_auc_score(y_test, y_prob)
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

```
GridSearch를 실행한 후 데스트 데이터세트의 정확도 : 0.67982
```

```
ROC AUC 값 : 0.75338
```

4.6.10 서포트 벡터 머신

```
import pandas as pd

df_svm = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/standard-scaled.csv')

df_svm.head()

data = df_svm.drop(['SALE_PRICE', 'PRICE_B'], axis=1)    # 타겟변수를
제외한 입력변수를 data에 저장

target = df_svm['PRICE_B']        # 타겟변수만 target 데이터프레임에 저장

# 데이터 분할

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)

print("x_train shape:", x_train.shape)

print("x_test shape:", x_test.shape)

# SVM 모델 (default 모델)

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

svm = SVC(kernel='rbf', C=1, gamma = 'auto', random_state=4,
probability=True)

model = svm.fit(x_train, y_train)

pred = model.predict(x_test)      # 학습된 Classifier로 테스트 데이터셋 자료
이용해서 타겟변수 예측값 생성

accuracy = accuracy_score(y_test, pred)

print("SVM Accuracy on training set:{:.5f}".format(model.score(x_train,
y_train)))

print("SVM Accuracy on test set:{:.5f}".format(accuracy_score(y_test,
pred)))
```

```
# SVM 모델 (default 모델)

svm = SVC(kernel='rbf', C=1, gamma='auto', random_state=4,
probability=True)

# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

params = {'kernel':['sigmoid'], 'C':[0.0001, 0.01, 1, 10],
'gamma':['auto', 'scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1)

grid_svm.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))

print("GridSearchCV best parameter:", (grid_svm.best_params_))
```

```
# SVM 모델 (default 모델)

svm = SVC(kernel='rbf', C=1, gamma='auto', random_state=4,
probability=True)

# 그리드 서치 재실행

from sklearn.model_selection import GridSearchCV

params = {'kernel':['rbf'], 'C':[0.0001, 0.01, 1, 10], 'gamma':['auto',
'scale']}

grid_svm = GridSearchCV(svm, param_grid=params, scoring='accuracy',
cv=5, n_jobs=-1)

grid_svm.fit(x_train, y_train)

print("GridSearchCV max accuracy:{:.5f}".format(grid_svm.best_score_))

print("GridSearchCV best parameter:", (grid_svm.best_params_))
```

```
best_clf = grid_svm.best_estimator_
pred = best_clf.predict(x_test)
print("Accuracy on test:{:.5f}".format(accuracy_score(y_test, pred)))
from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test, best_clf.predict_proba(x_test)[:, 1])
print("ROC AUC on test set:{:.5f}".format(ROC_AUC))
```

GridSearch를 실행한 후 테스트 데이터세트의 정확도 : 0.63704

ROC AUC 값 : 0.68885

bests parameter : {'C': 10, 'gamma': 'auto', 'kernel': 'sigmoid'}

4.6.11 회귀, 릿지

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Data_Analysis/NYC Property Sales/encoded.csv')

df.info()

cols = ['TOTAL_UNITS', 'LAND_SQUARE_FEET', 'GROSS_SQUARE_FEET',
'YEAR_BUILT', 'SALE_PRICE', 'PRICE_B']

df_cat = df.drop(cols, axis=1) # 데이터프레임에서 구간 변수 및 타겟변수 제외

df_cat.shape

pd.options.display.float_format = '{:.2f}'.format # 소수점 2 자리로 숫자
표기 제한

df_cat.describe()

df_cat.max() - df_cat.min()

df_cat.min()

cols1 = ['BOROUGH_encoded', 'BUILDING_CLASS_CATEGORY_encoded',
'BUILDING_CLASS_AT_PRESENT_encoded',
'TAX_CLASS_AT_TIME_OF_SALE_encoded']

df_ridge = pd.get_dummies(df, columns=cols1) # cols1에 담긴 변수들의
더미변수를 생성

df_ridge.head()
```

```
df_ridge.shape
```

```
list(df_ridge.columns)
```

```
# 기준 더미변수로 정한 4 개의 더미변수명을 cols2에 저장
```

```
cols2 = ['BOROUGH_encoded_3.0', 'BUILDING_CLASS_CATEGORY_encoded_0.0',
'BUILDING_CLASS_AT_PRESENT_encoded_1.0',
'TAX_CLASS_AT_TIME_OF_SALE_encoded_1.0']
```

```
df_ridge.drop(cols2, axis=1, inplace=True)      # cols2에 저장된 더미변수명을 제거
```

```
df_ridge.shape
```

```
data = df_ridge.drop(['SALE_PRICE', 'PRICE_B'], axis=1)    # 타겟변수를 제외한 입력변수를 data에 저장
```

```
target = df['SALE_PRICE']        # 타겟변수만 target에 저장
```

```
# 데이터 분할
```

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

```
x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)
```

```
print("x_train shape : ", x_train.shape)
```

```
print("x_test shape : ", x_test.shape)
```

```
# 선형 회귀 모델(기본 모델)
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import r2_score
```

```
linr = LinearRegression(n_jobs=-1)
```

```
model = linr.fit(x_train, y_train)
```

```
pred = model.predict(x_test)
```

```
print("Linear Regression Training set score:{:.5f}".format(model.score(x_train, y_train)))
```

```
print("Linear Regression Test set r2 score:{:.5f}".format(r2_score(y_test, pred)))
```

```

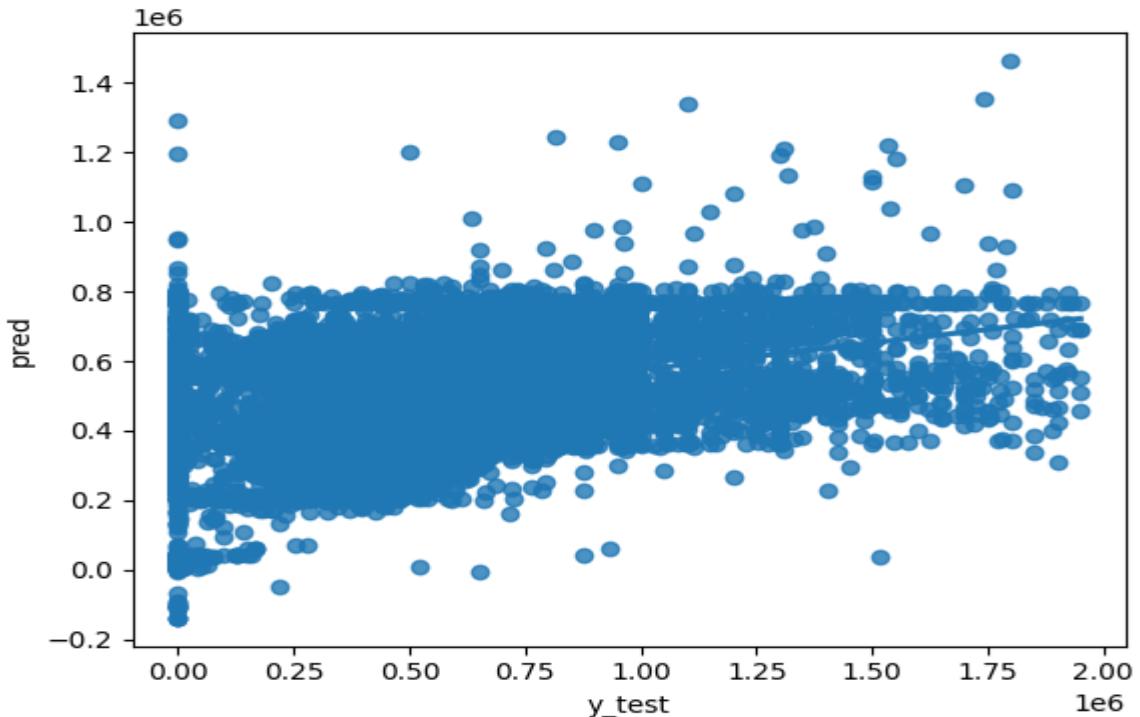
import seaborn as sns

final_result = pd.concat([y_test.reset_index(drop=True),
pd.DataFrame(pred), axis=1)

final_result.columns = ['y_test', 'pred']

sns.regplot(x='y_test', y='pred', data=final_result)

```



```

# 릿지 모델(기본 모델)

from sklearn.linear_model import Ridge

from sklearn.metrics import r2_score

Ridge = Ridge()

model = Ridge.fit(x_train, y_train)

pred = model.predict(x_test)

print("Linear Regression Training set
score:{:.5f}".format(model.score(x_train, y_train)))

print("Linear Regression Test set score:{:.5f}".format(r2_score(y_test,
pred)))

```

```
테스트 데이터세트의 r2 score : 0.14933
```

```
# 럿지 모델(기본 모델)
```

```
from sklearn.linear_model import Ridge

from sklearn.model_selection import GridSearchCV

Ridge = Ridge()

params = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000],
'solver':['auto', 'svd', 'lsqr',
'cholesky','sparse_cg','sag','saga','lbfgs']}}

grid_Ridge = GridSearchCV(Ridge, param_grid=params, scoring='r2', cv=5,
n_jobs=-1, verbose=1)

grid_Ridge.fit(x_train, y_train)

print("GridSearchCV max score:{:.5f}".format(grid_Ridge.best_score_))

print("GridSearchCV best parameter:", (grid_Ridge.best_params_))

best_clf = grid_Ridge.best_estimator_

pred = best_clf.predict(x_test)

print("R2 Score on test set:{:.5f}".format(best_clf.score(x_test,
y_test)))
```

```
GridSearch를 실행한 후 r2 score : 0.14924
```

```
bets parameter : {'alpha': 10, 'solver': 'auto'}
```

4.6.12 XGBoost

```
pip install xgboost
```

```
pip install lightgbm
```

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

df_xgboost = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Data_Analysis/NYC Property Sales/encoded.csv')

df_xgboost.shape
```

```
df_xgboost.head()
```

```
# 이진값 타겟변수 제외

df_xgboost.drop(['PRICE_B'], axis=1, inplace=True)

df_xgboost.shape
```

```
from sklearn.model_selection import train_test_split

data = df_xgboost.drop(['SALE_PRICE'], axis=1)

target = df_xgboost['SALE_PRICE']

# 데이터 분할

x_train, x_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=4)
```

```
# 기본 XGBRegressor 모델

from sklearn.metrics import r2_score

xgb = XGBRegressor(random_state=4)

xgb.fit(x_train, y_train)

pred = xgb.predict(x_test)

print("r2: {:.5f}".format(r2_score(y_test, pred)))
```

```
# XGBRegressor 최적화

from sklearn.model_selection import GridSearchCV

xgb = XGBRegressor()

parameters = {'colsample_bytree': [0.7, 0.8, 0.9],
              'learning_rate': [0.05, 0.1, 0.2],
              'max_depth': [8, 12, 16],
              'min_child_weight': [3, 4, 5],
              'n_estimators': [500, 1000, 1500],
              'subsample': [0.8, 0.9]}

xgb_grid = GridSearchCV(xgb,
                        parameters,
                        scoring='r2',
                        cv=3,
                        n_jobs=-1,
                        verbose=True)

xgb_grid.fit(x_train, y_train)
```

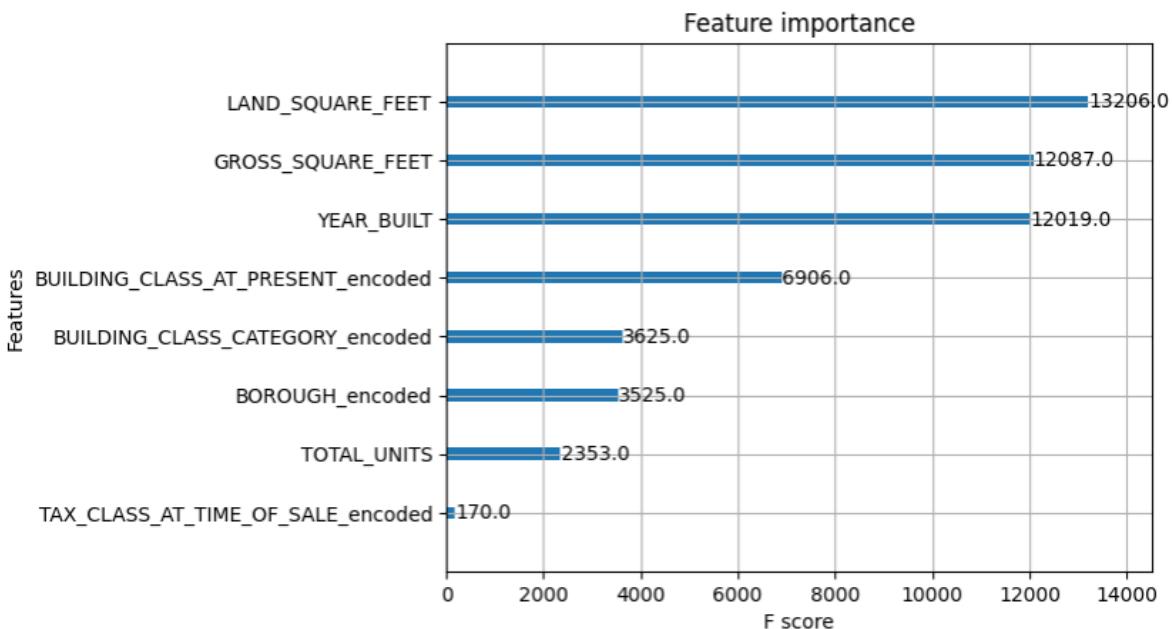
```
print("GridSearchCV 최적 파라미터:", xgb_grid.best_params_)
```

```
model = xgb_grid.best_estimator_
pred = model.predict(x_test)
print('r2 : {:.5f}'.format(r2_score(y_test, pred)))
```

GridSearch를 실행한 후 r2 score : 0.17694

```
bets parameter : {'colsample_bytree': 0.7, 'learning_rate': 0.05,
'max_depth': 8, 'min_child_weight': 5, 'n_estimators': 500,
'subsample': 0.9}
```

```
from xgboost import plot_importance
plot_importance(model, max_num_features=8)
```



4.6.13 LightGBM

```
# 기본 LGBMRegressor 모델

from lightgbm import LGBMRegressor

from sklearn.metrics import r2_score

lgb = LGBMRegressor(random_state=4)

lgb.fit(x_train, y_train)

pred = lgb.predict(x_test)

print("r2 : {:.5f}".format(r2_score(y_test, pred)))
```

```
# 그리드 서치 실행

from sklearn.model_selection import GridSearchCV

from lightgbm import LGBMRegressor

lgb = LGBMRegressor()

parameters = {'colsample_bytree':[0.7, 0.8],
              'learning_rate':[0.1, 0.15, 0.2],
              'max_depth':[11],
              'min_child_weight':[4],
              'n_estimators':[1000],
              'subsample':[0.3, 0.4]}

lgb_grid = GridSearchCV(lgb,
                        parameters,
                        scoring='r2',
                        cv=3,
                        n_jobs=-1,
                        verbose=True)

lgb_grid.fit(x_train, y_train)

print("GridSearchCV 최적 파라미터:", lgb_grid.best_params_)
```

```

from sklearn.metrics import r2_score

model = lgb_grid.best_estimator_

pred = model.predict(x_test)

print('r2: {:.5f}'.format(r2_score(y_test, pred)))

```

GridSearch를 실행한 후 r2 score : 0.16315

```

bests parameter : {'colsample_bytree': 0.7, 'learning_rate': 0.1,
'max_depth': 11, 'min_child_weight': 4, 'n_estimators': 1000,
'subsample': 0.3}

```

```

from lightgbm import plot_importance

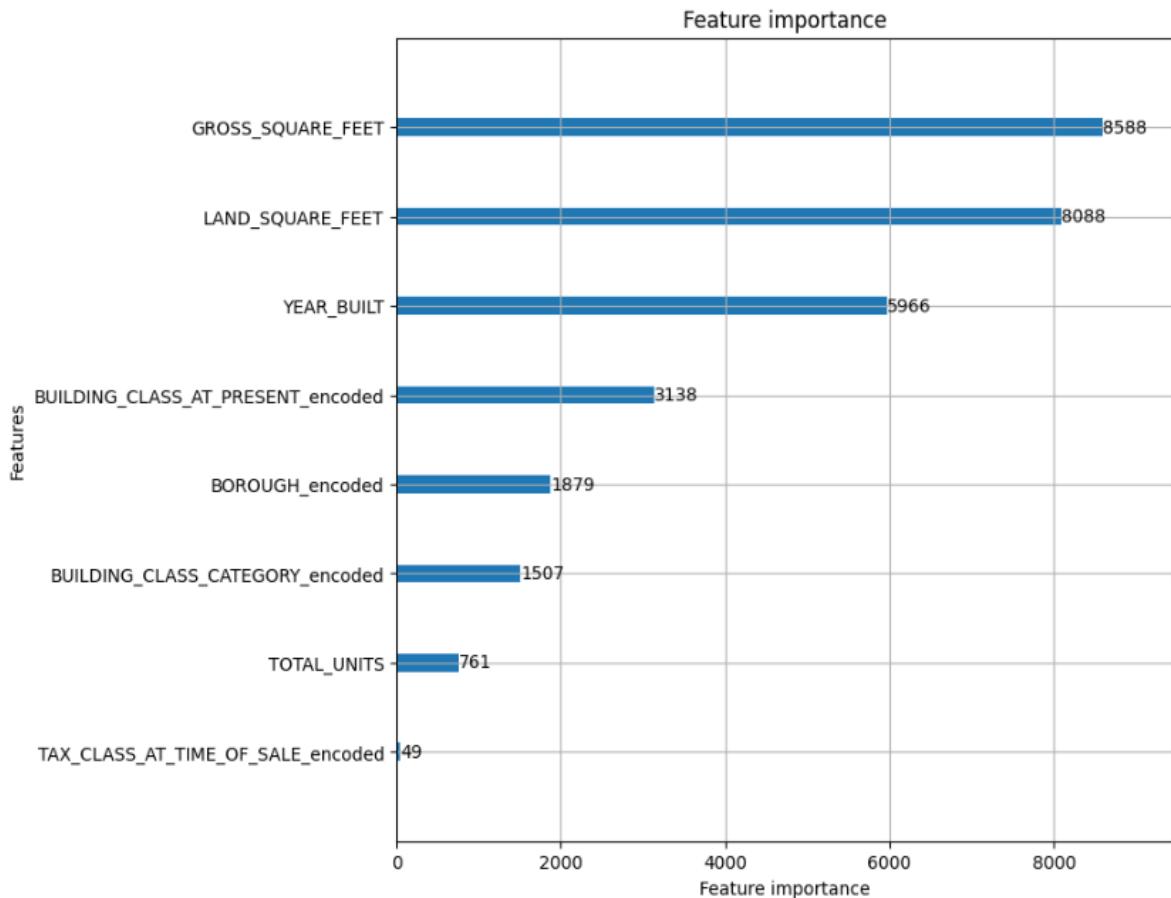
import matplotlib.pyplot as plt

%matplotlib inline

fig, ax = plt.subplots(figsize=(8, 8))

plot_importance(model, ax=ax)

```



4.6.14 스태킹 모델(양상블)

```
model_xgb = xgb_grid.best_estimator_
pred_xgb = model_xgb.predict(x_test)

model_lgb = lgb_grid.best_estimator_
pred_lgb = model_lgb.predict(x_test)
```

```
from sklearn.metrics import r2_score

pred = 0.5 * pred_xgb + 0.5 * pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.6 * pred_xgb + 0.4 * pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.7 * pred_xgb + 0.3 * pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.8 * pred_xgb + 0.2 * pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
pred = 0.9 * pred_xgb + 0.1 * pred_lgb

print('r2: {:.5f}'.format(r2_score(y_test, pred)))
```

```
0.7 * pred_xgb + 0.3 * pred_lgb 일 때 r2 score : 0.17901
```

4.7 모델 정확도 비교 및 최적 파라미터

4.7.1 머신러닝 모델 정확도 및 최적 파라미터

모델명	정확도	순위	최적 파라미터
결정 트리	0.67168	3	{'criterion': 'entropy', 'max_depth': 8}
로지스틱 회귀	0.62818	5	{'penalty': 'none', 'solver': 'lbfgs'}
로지스틱 회귀 (표준화)	0.63883	4	{'penalty': 'none', 'solver': 'saga'}
신경망 (표준화)	0.67365	2	{'activation': 'relu', 'alpha': 0.0001, 'solver': 'adam'}
K-최근접 이웃 (표준화)	0.67633	1	{'n_neighbors': 28}

4.7.2 이진 분류 머신러닝 모델 정확도 및 최적 파라미터

모델명	정확도	ROC AUC	순위	최적 파라미터
랜덤 포레스트	0.68188	0.76077	1	{'max_depth': 17, 'n_estimators': 200}
그레이디언트 부스팅	0.68090	0.75943	2	{'learning_rate': 0.01, 'max_depth': 11, 'n_estimators': 200}
라쏘 (로지스틱 회귀)	0.57223	0.59019	5	{'C': 0.1, 'solver': 'saga'}
신경망 (tf.keras)	0.67982	0.75338	3	
서포트 벡터 머신	0.63704	0.68885	4	{'C': 10, 'gamma': 'auto', 'kernel': 'sigmoid'}

4.7.3 연속형 회귀 머신러닝 모델 r2 score

모델명	r2 score	최적 파라미터
회귀 모델	0.14933	
릿지 모델	0.14924	{'alpha': 10, 'solver': 'auto'}
XGBoost	0.17694	{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 8, 'min_child_weight': 5, 'n_estimators': 500, 'subsample': 0.9}
LightGBM	0.16315	{'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 11, 'min_child_weight': 4, 'n_estimators': 1000, 'subsample': 0.3}
스태킹 모델 (앙상블)	0.17901	

5. 결론

Airbnb의 경우 랜덤 포레스트 모델로 분석한 결과 숙소 가격에 영향을 미치는 요인에는 방의 종류, 수용인원, 편의시설 수, 침실 수, 침대 개수 등이 있다. 부동산의 경우 그레이언트 부스팅 모델로 분석한 결과 부동산 가격에 영향을 미치는 요인에는 건물 층의 총 면적, 지어진 연도, 토지 면적, 부동산 지역, 빌딩 종류 등이 있다. 이러한 분석을 토대로 부동산 시장과 공유 경제의 상호작용에는 다양한 영향을 미친다.

1. 수요와 공급의 변화

Airbnb와 같은 공유 경제 플랫폼은 부동산 시장의 수요와 공급을 변화시킨다. 여행자들이 호텔 대신 Airbnb를 선호하면서 특정 지역에서는 숙박 시설의 수요가 늘어난다. 이로 인해 부동산 수요가 증가하고 주택 가격이 상승할 수 있다.

2. 관광 및 지역 발전

Airbnb와 같은 공유 경제 서비스는 지역 발전을 촉진할 수 있다. 관광 명소의 경우 호스트가 숙소 대여로 인한 추가 수입을 얻을 수 있어 지역 경제에 긍정적인 영향을 미칠 수 있다. 이는 지역의 부동산 가치를 올릴 수도 있다.

3. 규제와 법적 요인

지역 정부의 규제와 법적 요인은 공유 경제와 부동산 시장의 상호작용에 큰 영향을 미칠 수 있다. 공유 경제 플랫폼에 대한 규제를 강화하면 부동산 가격에 영향을 미칠 수 있다.

4. 부동산 투자 및 수익

부동산 투자자들은 공유 경제 플랫폼을 통해 부동산을 투자하고 추가 수익을 얻을 수 있다. 이로 인해 부동산의 가치가 상승할 수 있다.

5. 고용 및 경제 성장

Airbnb 호스트 등 부동산과 직접적으로 연관된 서비스를 제공하는 사람들은 지역 경제에 기여하고 새로운 일자리가 생성되어 부동산 가격에 영향을 미칠 수 있다.