

프로젝트형 데이터 분석 서비스 개발

# 전공 프로젝트 일정 및 프로젝트 가이드

# 프로젝트 일정

- ▶ 기간 : 2023년 11월 14일 ~ 12월 4일
- ▶ 세부일정
  - ▶ 11월 15일 1차 팀 미팅
    - ▶ ~ 11월 17일까지 팀별 기획안 제출메일제목 xxx팀 기획안 제출
  - ▶ 11월 17일 팀별 주제 한 줄 발표
  - ▶ 11월 24일 2차 팀 미팅
  - ▶ 12월 01일 3차 팀 미팅
  - ▶ 12월 04일 발표 및 포트폴리오 제출

# 프로젝트 개요

- ▶ 문제해결 빅데이터 활용 프로젝트
  - ▶ 주변에서 일어날 수 있는 문제(현황파악)를 제시하고
    - ▶ 해당 문제를 러닝기법의 예측을 통해 문제의 미래를 예측해보고
    - ▶ 현재를 준비하기 위한 연습을 위한 프로젝트

# 프로젝트 주제 및 범위

## ▶ 주제

- ▶ 공공데이터를 활용한 시각화 분석 및 모델링
  - ▶ 공공데이터는 상업용이 아닌 누구나 이용가능한 데이터를 의미함

## ▶ 범위

- ▶ 주제 기획
- ▶ 데이터 수집
- ▶ 현황 분석을 위한 데이터 전 처리 및 가공
- ▶ 시각화 분석 및 결론
- ▶ 모형 개발을 위한 데이터 전 처리(피처 엔지니어링)
- ▶ 모델링 및 모델 선정
- ▶ 향후 발전 방향

# 프로젝트 수행 기준

- ▶ 주제 기획 설명 시 주제에 대한 당위성 설명에 시각화기법을 활용한다.
- ▶ 데이터 수집
  - ▶ 크롤링과 API 수집 데이터를 포함한다.
- ▶ 파이썬을 활용한 모델링을 진행한다

# 데이터 site

- ▶ <https://data.seoul.go.kr>
- ▶ <https://www.data.go.kr>
- ▶ <http://www.index.go.kr>
- ▶ <http://kosis.kr>
- ▶ <https://bigdata.seoul.go.kr>
- ▶ <http://kostat.go.kr>
- ▶ <http://www.price.go.kr/tprice/portal/main/main.do>
- ▶ <https://data.kma.go.kr/cmmn/main.do>
- ▶ [https://www.airkorea.or.kr/web/pmRelay?itemCode=11008&pMENU\\_NO=109](https://www.airkorea.or.kr/web/pmRelay?itemCode=11008&pMENU_NO=109)
- ▶ <https://www.localdata.kr/>
- ▶ 모든 데이터에는 저작권이 있을 수 있습니다. 확인 후 사용해야 합니다.

# 데이터 site

- ▶ <https://datasetsearch.research.google.com/>
- ▶ <https://www.kaggle.com/datasets>
- ▶ <https://github.com/awesomedata/awesome-public-datasets>
- ▶ <https://dasl.datadescription.com/>
- ▶ <https://datahub.io/>
- ▶ <https://data.worldbank.org/>
- ▶ 모든 데이터에는 저작권이 있을 수 있습니다. 확인 후 사용해야 합니다.

# 프로젝트 참고 사항

- ▶ 전공 수업 내용을 리뷰하는 차원의 프로젝트로 진행하시기 바랍니다
- ▶ 데이터 분석 전공이므로 현황파악에 더 중점을 두고 분석 시각화를 진행해 보면 좋을 것 같습니다
- ▶ 러닝 예측은 관련 데이터와 서브 주제를 다시 선정해야 할 수도 있습니다
  - ▶ 일단 분석시각화를 진행 후 나온 결과에 따라 러닝 예측을 진행하는 것도 한 방법입니다
- ▶ 하루 일정은 별다른 공지를 하지 않습니다
  - ▶ 쉬는 시간 등은 자유롭게 설정하시고 다만 점심시간은 11시30분에서 1시30분내에서 1시간을 사용하시기 바랍니다
- ▶ 특별한 사유없이 메인 세션에 오래 남아있거나 소회의실 접속이 유지 되지 않으면 결석으로 간주될 수 있습니다. 유의 바랍니다





# 분석 시각화 프로젝트 포트폴리오 예시

프로젝트 주제



- OECD 평균 대비 가장 취약한 3대 지표 중 하나, **교통사고**
- 초고령사회가 진행되고 있는 시점에서 **노인 교통사고**가 큰 문제
- 2018년에서 2020년 3년동안 보행 중 교통사고 **전체 사망자의 약 58% 노인**

SAMPLE

[출처 : 한국교통안전공단(2018-2020)]

## “지역별 노인 교통 사망사고 현황 분석”

프로젝트 컨셉



주요 분석 내용

- 연령대별 사상자수 대비 사망자 비율
- 사고 유형별 노인교통사고 사망 발생건수
- 지역별 노인 인구비율과 노인 사망자수
  - 노인교통사망자수가 높은지역
- 지역별, 월별 노인교통사고 사망자수
- 지역별 응급실 현황
  - 노인교통사고 다발지역
- 노인교통사고 다발지역과 응급실거리와의 관계

## 03-2 데이터 명세

출처	데이터 이름	제공 형태	요약
통계청	연령별 지역별 인구수	csv	2019년 전국 연령별, 지역별 인구수 데이터(1838X5)
	시도별 요양기관 현황	csv	2019년 전국 시도별, 종별 요양기관 현황 데이터 (20X17)
교통사고분석시스템 (TAAS)	전국 연령층별 교통사고 사상자	xls	연령층별 사상자수 대비 사망자 비율(21X3)
	차대사람 전국 노인 교통사고 사망자 현황	xls	차대사람 전국 노인 교통사망 사고유형 사망자수 (39X9)
	요일별 노인교통사고	xls	2019년 전국 요일별 노인 교통사고 데이터 (56X10)
	월별 노인교통사고	xls	전국 월별 노인 교통사고 데이터 (56X15)
	시간대별 노인 교통사고	xls	전국 시간대별 노인 교통사고 데이터 (56X15)
충청남도 통합 복지	충남응급의료기관	크롤링	충남 의료기관 현황(16x5)
공공데이터 포털	전라남도 응급의료기관	csv	전라남도 응급의료기관 현황 (39x13)
	제주특별자치도 응급의료기관	csv	제주특별자치도 응급의료기관 현황 (9x5)
	도로교통공단 보행 노인 사고다발 지역 정보 서비스	API	충남교통사고다발지역 (20x1) 전남교통사고다발지역 (23x1) 제주교통사고다발지역 (10x1)
직접 데이터 수집	병원과 사고다발 지역 거리	csv	제주병원사망다발지역거리 (13x4) 전남병원사망다발지역거리 (33x4) 충남병원사망다발지역거리 (24x4)

# 데이터 전처리

## 03-3 데이터 전처리 및 탐색

&lt;전국 연령층별 교통사고 사상자 수&gt;

연령층별	기준년도	2019
합계	사망자수	3,349
	부상자수	341,712
12세이하	사망자수	28
	부상자수	14,115
13-20세	사망자수	108
	부상자수	19,884
21-30세	사망자수	247
	부상자수	58,010
31-40세	사망자수	231
	부상자수	59,022
41-50세	사망자수	324
	부상자수	60,127
51-60세	사망자수	614
	부상자수	64,455
61-64세	사망자수	271
	부상자수	21,608
65세이상	사망자수	1,390
	부상자수	143,900
불명	사망자수	
	부상자수	101



연령층별	사망자수	부상자수	사상자수대비 사망자 비율
12세이하	28.0	14115.0	0.20
13-20세	108.0	19884.0	0.54
21-30세	247.0	58010.0	0.42
31-40세	231.0	59022.0	0.39
41-50세	324.0	60127.0	0.54
51-60세	614.0	64455.0	0.94
61-64세	271.0	21608.0	1.24
65세이상	1390.0	143900.0	3.32

SAMPLE

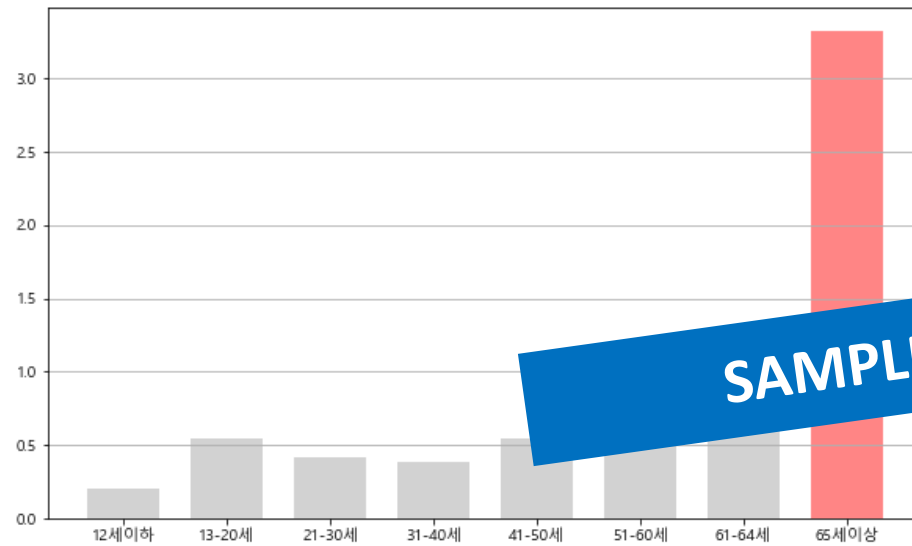
NaN값 제거	<code>age_data.dropna(inplace=True)</code>
연령층 별로 인덱스 설정	<code>age_data.set_index(age_data['연령층별'], inplace=True)</code>
사망자수만 추출	<code>die_data = age_data[0::2]</code>
부상자수만 추출	<code>nondie_data = age_data[1::2]</code>
사상자 수 대비 사망자 비율 구하기	<code>nondie_per_die_result["사상자수대비 사망자 비율"] = (nondie_per_die_result['사망자수'] / nondie_per_die_result["사망자수"] + nondie_per_die_result["부상자수"] * 100).round(2)</code>

데이터 시각화

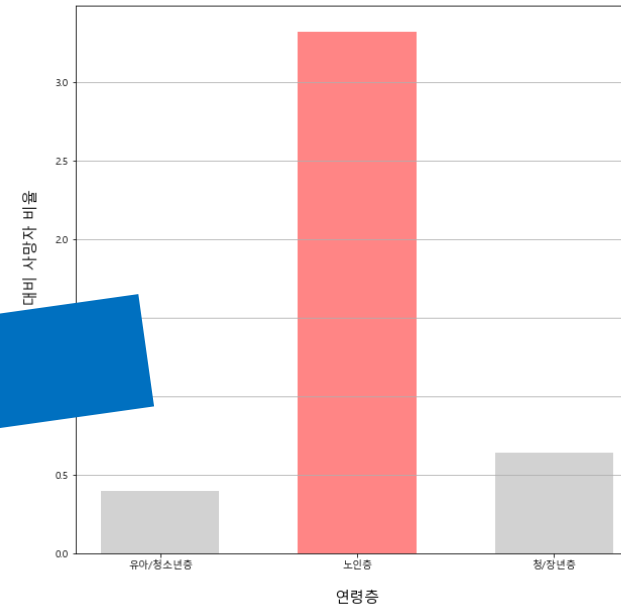
SAMPLE

## 03-4 데이터 시각화

&lt;연령대별 사상자수 대비 사망자 비율&gt;



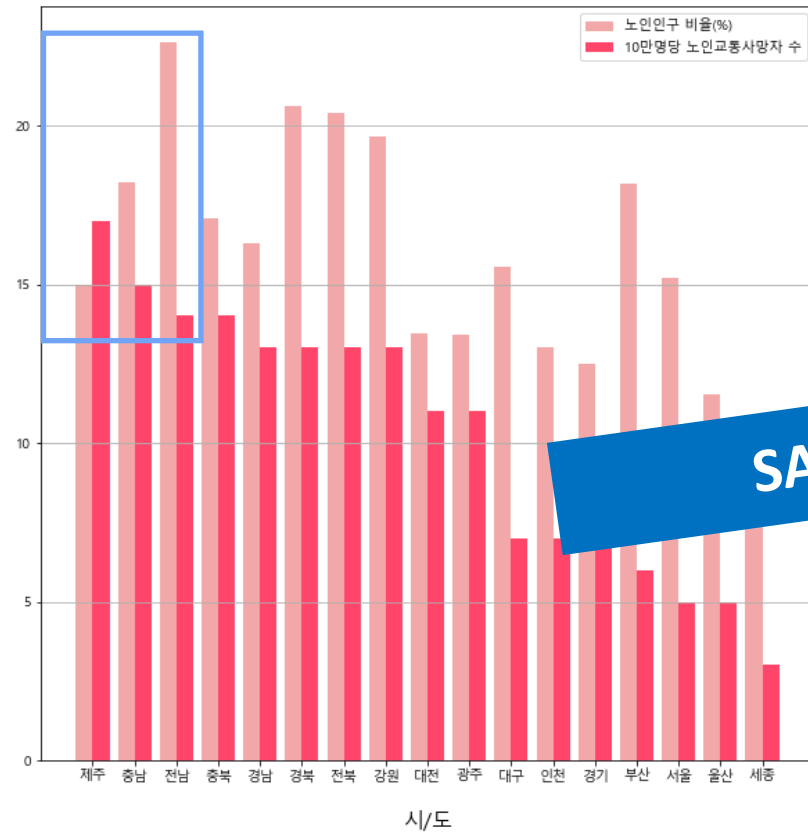
&lt;연령층별 사상자수 대비 사망자 비율&gt;



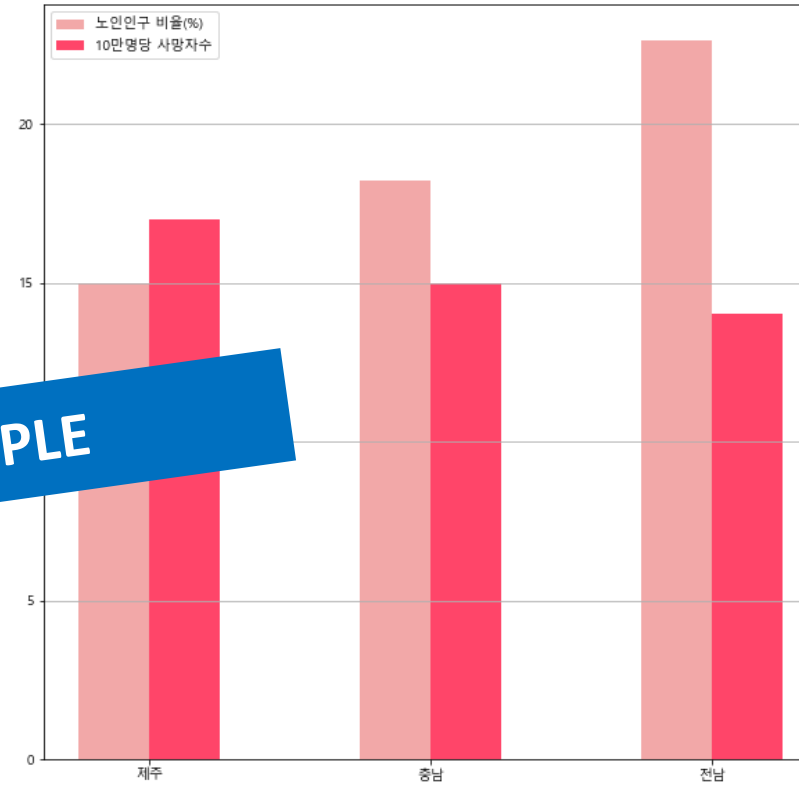


## 03-4 데이터 시각화

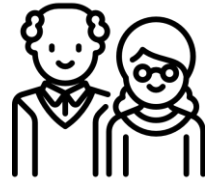
&lt;노인인구비율과 10만명 당 노인교통사고 사망자수&gt;



&lt;10만명 당 노인교통사망자수가 가장 높은 지역 3곳&gt;



결론



#연령대별  
#사상자수대비  
#사망자비율  
#65세이상



#10만명당  
#노인사망자수  
#제주도  
#충남,전남



#사고유형별  
#노인사망자수  
#횡단중

SAMPLE



#제주3월  
#충남,전남10월  
#6시~8시  
#18시~20시




#응급실현황과  
#사고다발지역



#시장  
#병원  
#교차로,회전차로

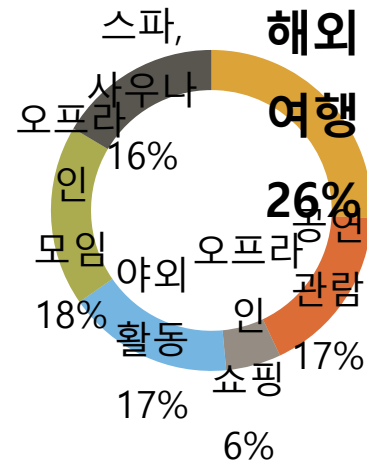


# 모델링 포트폴리오 예시



# 에어비앤비 호스트를 위한 뉴욕시 숙소 가격 예측 모델링

SAMPLE



코로나가 종식되면 가장 하고 싶은 활동  
**해외여행 26%**

[자료 : 더폴, 응답자 2만8669명(조사기간 6월 21~23일)]

**SAMPLE**

“ 에어비앤비 호스트를 위한 숙소 가격 예측 모델링 구현 ”

출처	데이터 이름	제공 형태	요약
kaggle	AB_NYC_2019	CSV	2019 뉴욕 에어비앤비 데이터 (48895*16)

kaggle

+

Create

Home

Competitions

Datasets

Code

Discussions

Courses

More

Search

Sign In

Register

Dataset

2390

New York City Airbnb Open Data

Airbnb listings and metrics in NYC, NY, USA

Dgomonov

SAMPLE

Data

Tasks (2)

Discussion (34)

Activity

Metadata

Download (7 MB)

New Notebook

Usability 10.0

License CC0: Public Domain

Tags business, internet, hotels and accommodations

Description

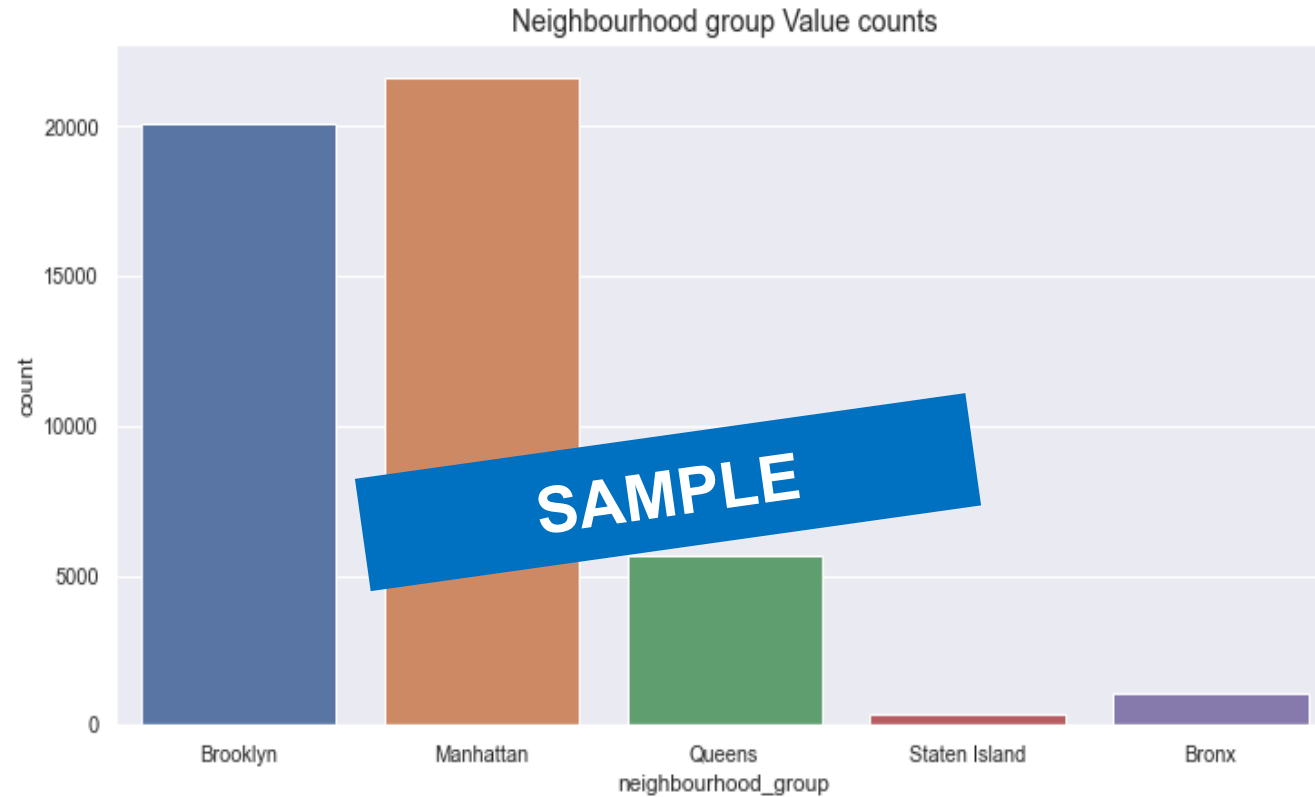
Context

Since 2008, guests and hosts have used Airbnb to expand on traveling possibilities and present more unique, personalized way of experiencing the world. This dataset describes the listing activity and metrics in NYC, NY for 2019.

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability_365
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	1	9	2018-10-19	0.21	6	365
1	2595	Skyliit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	1	45	2019-05-21	0.38	2	355
2	3647	THE VILLAGE OF HARLEM.....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	3	0	NaN	NaN	1	365

변수 이름		변수 의미	전처리 방법
1	id	고객 식별 번호	도메인 지식 하에 필요없는 변수 삭제
2	name	에어비앤비 숙소이름	
3	host_id	호스트 식별 번호	
4	host_name	호스트 이름	
5	neighborhood	자치구	
6	latitude	위도	
7	longitude	경도	
8	last_review	마지막 리뷰 날짜	
9	neighborhood_group	뉴욕 자치구	범주형 변수 인코딩
10	room_type	숙소 타입	
11	price	1박 당 가격	로그 변환
12	minimum_nights	최소 예약 일수	이상치 제거
13	reviews_per_month	월 평균 리뷰 수	NaN 값 '0'으로 대체
14	availability_365	365일 중 이용 가능한 날짜 수	3가지 유형으로 전처리
15	calculated_host_listings_count	호스트가 운영중인 숙소 개수	-
16	number_of_reviews	사용자 리뷰 개수	-

## &lt;뉴욕 자치구별 에어비앤비 개수 시각화&gt;



- 가장 많은 에어비앤비를 가지고 있는 지역은 맨해튼, 브루클린
- 유명한 관광지가 맨해튼과 브루클린에 모여있기 때문



```
df.shape
```

```
(48895, 16)
```

```
df.describe()
```

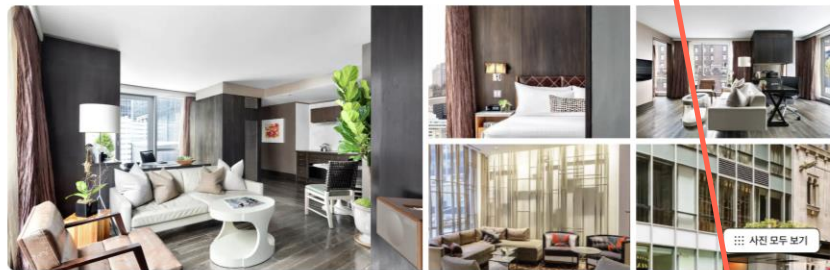
	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000	48895.000000	48895.000000
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	7.143982	112.781327
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	32.952519	131.622289
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	1.000000	0.000000
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	1.000000	0.000000
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.888889	0.720000	1.000000	45.000000
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.888889	12.500000	17.777778	0.920000	2.000000	227.000000
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	1145.888889	1250.000000	58.500000	58.500000	327.000000	365.000000

SAMPLE

## Hotel 48 Lex, The House

뉴욕, 미국

공유하기 | 저장



48Lex님이 호스팅하는 부티크 호텔의 객실

최대 인원 2명 · 침실 2개 · 침대 2개 · 욕실 2개



₩13,698,499 / 박 ₩10,273,868 / 박

(1) price : '0' (달러)

- 삭제

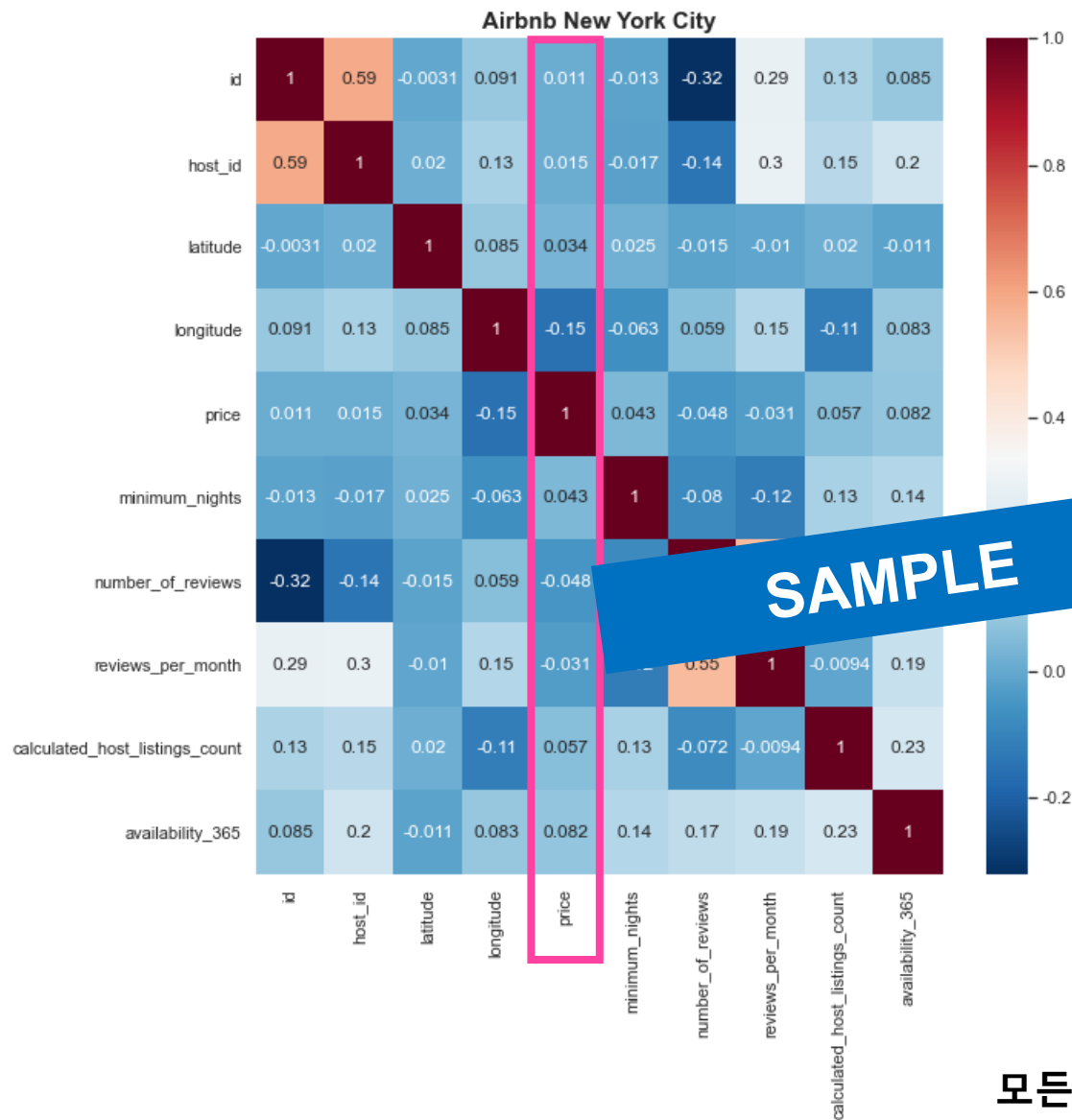
(2) minimum\_nights : 1250

- 이상치로 판단되어 365 이상 삭제

(3) availability\_365 : '0'값

- 평균값, 증위수, 365 세 가지 케이스로 수행

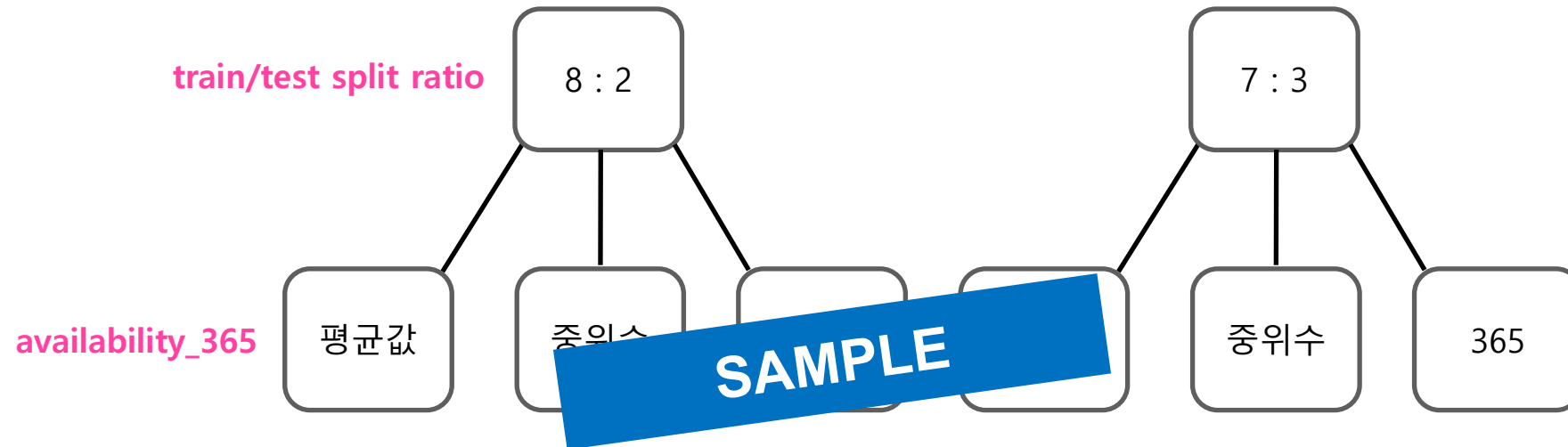
## &lt;Pearson 상관계수&gt;



피어슨 상관계수 R	상관관계 정도
$\pm 0.9$ 이상	매우 높은 상관관계
$\pm 0.9 \sim \pm 0.7$	높은 상관관계
$\pm 0.7 \sim \pm 0.4$	다소 높은 상관관계
$\pm 0.4 \sim \pm 0.2$	낮은 상관관계
$\pm 0.2$ 미만	상관관계 없음

모든 피처가 가격(price)와 상관 관계가 없음

## &lt;모델링 수행 방법&gt;



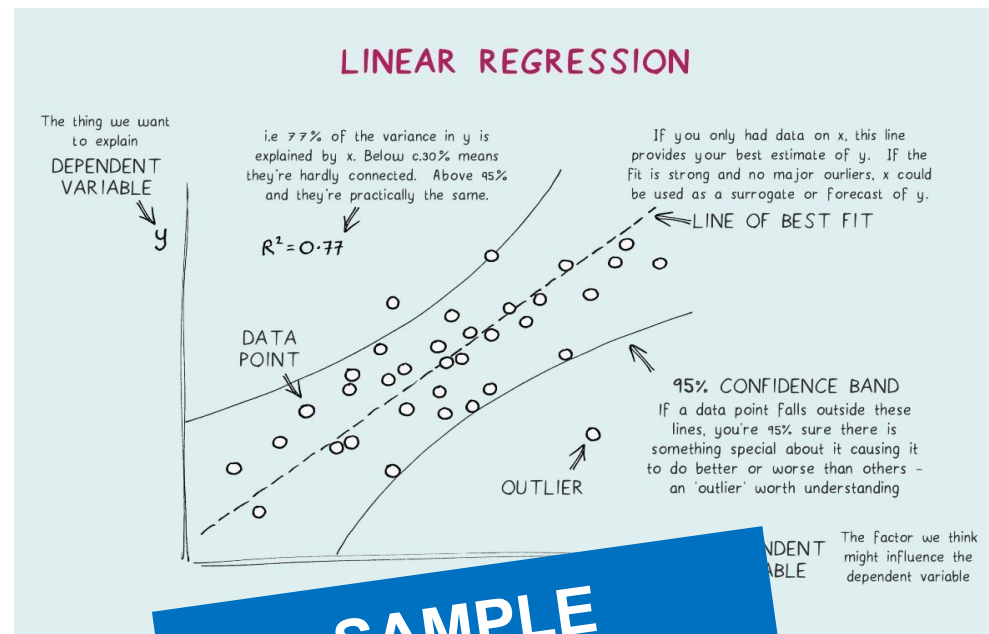
총 여섯가지 케이스로 모델링 수행

## 회귀분석 모델링 종류

1. 선형회귀
2. 다항회귀
3. 릿지, 라쏘
4. 5-fold 교차검증 (cross-validation)
5. 로컬 혼합
6. 모델 혼합
7. 스택킹

**SAMPLE**

## &lt;선형 회귀&gt;



```
from sklearn.linear_model import LinearRegression

lr = LinearRegression() # 객체 생성
lr.fit(X_train, y_train) # 학습

# 예측
y_preds = lr.predict(X_test)
y_preds

# 평가
mse = mean_squared_error(y_test, y_preds) # 예측 오류 값 계산 (실제값과 예측값의 차이)
rmse = np.sqrt(mse) # 사이킷런이 루트를 지원하지 않기 때문에 np.sqrt()로 루트씌워서 계산

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_preds)))
```

## &lt;선형 회귀&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	LinearRegression RMSE : 0.507	LinearRegression RMSE : 0.509
availability_365 중위수	LinearRegression RMSE : 0.505	LinearRegression RMSE : 0.507
availability_365 365	LinearRegression R	LinearRegression RMSE : 0.514

SAMPLE

- availability\_365 '0'값을 중위수로 대체하고, 8:2 비율로 모델링했을 때 성능이 가장 좋음

## &lt;다항 회귀&gt;

## (1) degree = 2

```
# (1) degree=2 2차다항으로 만들기
p_model = Pipeline([ ('poly', PolynomialFeatures(degree=2)),
                     ('linear', LinearRegression()) ])
# pipeline에 data를 전달해서 학습 진행
p_model.fit(X_train, y_train)
# 학습된 회귀식을 이용해 예측
y_preds = p_model.predict(X_test)
# 평가
mse = mean_squared_error(y_test, y_preds)
rmse = np.sqrt(mse)
print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_preds)))
```

## (2) degree = 2, 편향성 제외

```
# (2) degree=2, include_bias 튜닝
p_model = Pipeline([ ('poly', PolynomialFeatures(degree=2, include_bias=False)),
                     ('linear', LinearRegression()) ])
# pipeline에 data를 전달해서 학습 진행
p_model.fit(X_train, y_train)
# 예측
y_preds = p_model.predict(X_test)
mse = mean_squared_error(y_test, y_preds)
rmse = np.sqrt(mse)
print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_preds)))
```

## (3) degree = 3, 편향성 제외

```
# (3) degree=3, include_bias=False
p_model = Pipeline([ ('poly', PolynomialFeatures(degree=3, include_bias=False)),
                     ('linear', LinearRegression()) ])
# 학습
p_model.fit(X_train, y_train)
# 예측
y_preds = p_model.predict(X_test)
mse = mean_squared_error(y_test, y_preds)
rmse = np.sqrt(mse)
print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_preds)))
```

## &lt;다항 회귀 - 2차&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	MSE : 0.249 RMSE : 0.499 Variance Score : 0.481	MSE : 0.249 RMSE : 0.499 Variance Score : 0.479
availability_365 중위수	MSE : 0.247 RMSE : 0.497 Variance Score : 0.485	MSE : 0.247 RMSE : 0.497 Variance Score : 0.484
availability_365 365	MSE : 0.255 RMSE : 0.505 Variance Score : 0.468	MSE : 0.254 RMSE : 0.504 Variance Score : 0.468

- availability\_365 '0'값을 중위수로 대체하고, 8:2 비율로 모델링했을 때 성능이 가장 좋음



## &lt;다항 회귀 - 2차, 편향성 제외&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	MSE : 0.249 RMSE : 0.499 Variance Score : 0.481	MSE : 0.249 RMSE : 0.499 Variance Score : 0.479
availability_365 중위수	MSE : 0.247 RMSE : 0.497 Variance Score : 0.485	MSE : 0.247 RMSE : 0.497 Variance Score : 0.484
availability_365 365	MSE : 0.255 RMSE : 0.505 Variance Score : 0.466	MSE : 0.254 RMSE : 0.504 Variance Score : 0.468

- availability\_365 '0'값을 중위수로 대체하고, 8:2 비율로 모델링했을 때 성능이 가장 좋음

## &lt;다항 회귀 - 3차, 편향성 제외&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	MSE : 0.248 RMSE : 0.498 Variance Score : 0.482	MSE : 0.247 RMSE : 0.497 Variance Score : 0.484
availability_365 중위수	MSE : 0.248 RMSE : 0.498 Variance Score : 0.482	MSE : 0.247 RMSE : 0.497 Variance Score : 0.484
availability_365 365	MSE : 0.254 RMSE : 0.504 Variance Score : 0.471	MSE : 0.254 RMSE : 0.502 Variance Score : 0.473

- availability\_365 '0'값을 중위수로 대체하고, 7:3 비율로 모델링했을 때 성능이 가장 좋음

## &lt;선형, 릿지, 라쏘 회귀&gt;

```
# 단일 모델의 RMSE 값 반환
def get_rmse(model): # 학습된 모델을 받아서 예측
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test , pred)
    rmse = np.sqrt(mse)
    print('{0} 로그 변환된 RMSE: {1}'.format(model.__class__.__name__, np.round(rmse, 3)))
    return rmse

# 여러 모델의 RMSE 값 반환
def get_rmses(models) :
    rmses = []
    for model in models :
        rmse = get_rmse(model) # 단일 모델의 RMSE 값 반환 함수 = get_rmse
        rmses.append(rmse)
    return rmses

# 일반 선형 회귀 = linear regression
lr_reg = LinearRegression() # 1차 선형 회귀
lr_reg.fit(X_train, y_train)

# 릿지 회귀 = ridge regression
ridge_reg = Ridge()
ridge_reg.fit(X_train, y_train)

# 라쏘 회귀 = lasso regression
lasso_reg = Lasso()
lasso_reg.fit(X_train, y_train)

models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models) # 학습된 모델 전달하고 # rmse값 반환
```

SAMPLE

## &lt;선형, 릿지, 라쏘 회귀&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	LinearRegression RMSE : 0.507 Ridge RMSE : 0.507 Lasso RMSE : 0.686	LinearRegression RMSE : 0.509 Ridge RMSE : 0.509 Lasso RMSE : 0.685
availability_365 중위수	LinearRegression RMSE : 0.505 Ridge RMSE : 0.505 Lasso RMSE : 0.685	LinearRegression RMSE : 0.507 Ridge RMSE : 0.507 Lasso RMSE : 0.684
availability_365 365	LinearRegression RMSE : 0.512 Ridge RMSE : 0.512 Lasso RMSE : 0.687	LinearRegression RMSE : 0.514 Ridge RMSE : 0.514 Lasso RMSE : 0.686

- availability\_365 '0'값을 중위수로 대체하고, 8:2 비율로 모델링했을 때 성능이 가장 좋음

- 라쏘 모델만 0.68대로 다른 두 모델에 비해 성능이 떨어짐을 확인

## &lt;선형, 릿지, 라쏘 회귀 시각화&gt;

```
def get_top_bottom_coef(model) :
    coef = pd.Series(model.coef_, index=X_features.columns)
    coef_high = coef.sort_values(ascending=False).head(10)
    coef_low = coef.sort_values(ascending=False).tail(10)
    return coef_high, coef_low

# 모델별 회귀 계수 시각화 함수
def visualize_coefficient(models) :
    # 3개 회귀 모델의 시각화를 위해 3개의 컬럼을 가지는 subplot 생성
    fig, axs = plt.subplots(figsize=(20,10),nrows=1, ncols=3)
    fig.tight_layout()
    # 입력인자로 받은 list객체인 models에서 차례로 model을 추출하여 회귀 계수 시각화
    for i_num, model in enumerate(models) :
        # 상위 10개, 하위 10개 회귀 계수를 구하고, 이를 판다스 데이터프레임으로 저장
        coef_high, coef_low = get_top_bottom_coef(model)
        coef_concat = pd.concat([coef_high, coef_low])

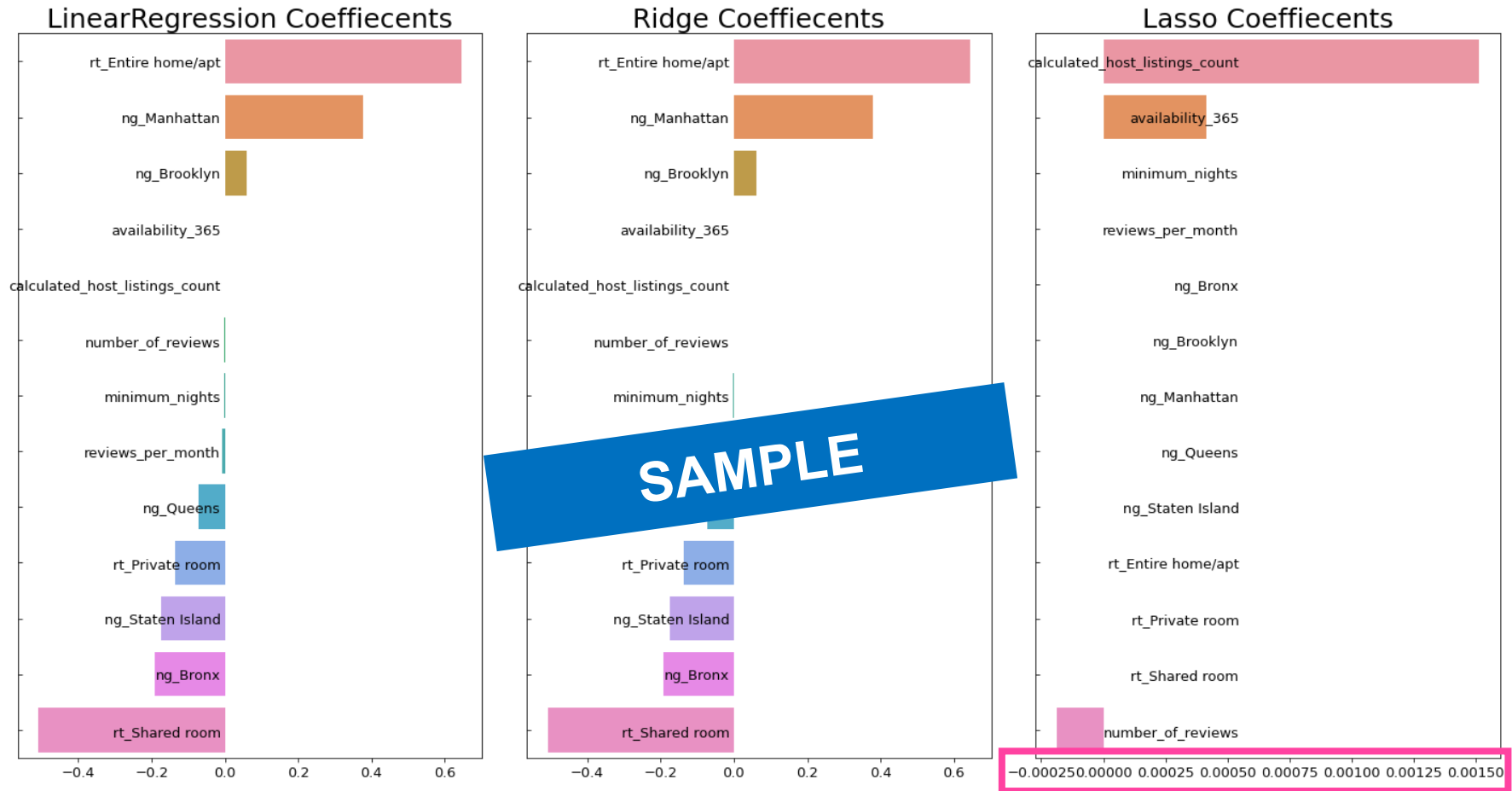
        # 순차적으로 ax subplot에 barchar로 표현. 한 화면에 표현하기 위해 tick label 위치와 font 크기
        axs[i_num].set_title(model.__class__.__name__ + ' Coeffiecents', size=25)
        axs[i_num].tick_params(axis="y",direction="in", pad=-190) # 안쪽, 패딩값

        for label in (axs[i_num].get_xticklabels() + axs[i_num].get_yticklabels()) :
            label.set_fontsize(13)
        sns.barplot(x=coef_concat.values, y=coef_concat.index, ax=axs[i_num])

models = [lr_reg, ridge_reg, lasso_reg]
visualize_coefficient(models)
```

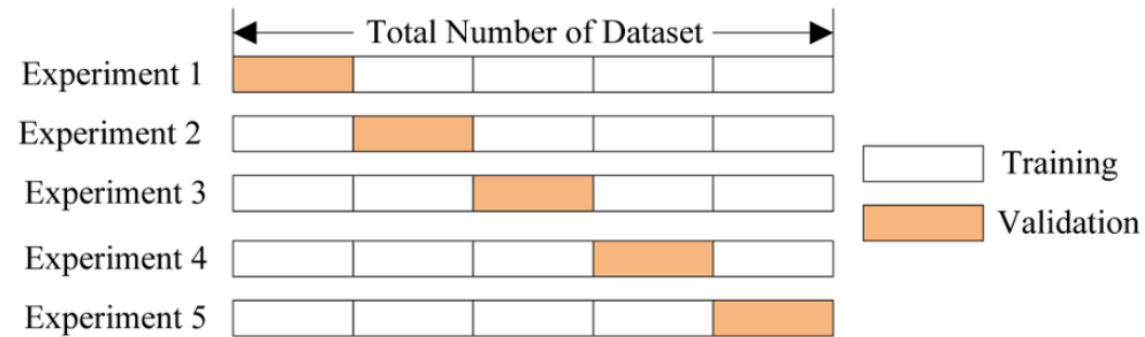
SAMPLE

## &lt;선형, 릿지, 라쏘 회귀 시각화&gt;



선형, 릿지보다 라쏘 회귀 값이 작게 나옴을 확인

## &lt;5-폴드 교차 검증&gt;



```

from sklearn.model_selection import cross_val_score

def get_avg_rmse_cv(models) :
    for model in models :
        # 데이터 분할하지 않고, 전체 데이터 세
        rmse_list = np.sqrt(-cross_val_score(model, X_features, y_target, scoring='neg_mean_squared_error', cv=5))
        rmse_avg = np.mean(rmse_list)
        print('\n{0} CV RMSE 값 리스트: {1}'.format( model.__class__.__name__, np.round(rmse_list, 3)))
        print('{0} CV 평균 RMSE 값: {1}'.format( model.__class__.__name__, np.round(rmse_avg, 3)))

# 앞에서 학습한 lr_reg, ridge_reg, lasso_reg 모델의 CV RMSE값 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_avg_rmse_cv(models)

```

**SAMPLE**

## &lt;5-폴드 교차 검증&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	LinearRegression CV 평균 RMSE : 0.507 Ridge CV 평균 RMSE : 0.507 Lasso CV 평균 RMSE : 0.686	LinearRegression CV 평균 RMSE : 0.507 Ridge CV 평균 RMSE : 0.507 Lasso CV 평균 RMSE : 0.686
availability_365 중위수	LinearRegression CV 평균 RMSE : 0.505 Ridge CV 평균 RMSE : 0.505 Lasso CV 평균 RMSE : 0.686	LinearRegression CV 평균 RMSE : 0.505 Ridge CV 평균 RMSE : 0.505 Lasso CV 평균 RMSE : 0.686
availability_365 365	LinearRegression CV 평균 RMSE : 0.514 Ridge CV 평균 RMSE : 0.514 Lasso CV 평균 RMSE : 0.687	LinearRegression CV 평균 RMSE : 0.514 Ridge CV 평균 RMSE : 0.514 Lasso CV 평균 RMSE : 0.687

- availability\_365 '0'값을 중위수로 대체하고, 8:2 비율로 모델링했을 때 성능이 가장 좋음

- 5-폴드 교차검증을 했음에도 여전히 라쏘 모델만 성능이 떨어짐을 확인



## &lt;릿지, 라쏘 모델 alpha 하이퍼 파라미터 튜닝&gt;

```
for i in range(len(alpha)) :  
    print(alpha[i], get_params(ridge_reg, alpha)[i]) # 0.05 0.5076497800675304
```

```
1e-05 0.5076497450442649  
0.0001 0.5076497451072713  
0.001 0.5076497457373452  
0.005 0.507649748537884  
0.01 0.507649752039039  
0.05 0.5076497800675304  
0.1 0.5076498151512664  
1 0.507650455781342  
5 0.507653509858632  
7 0.5076551616019265  
10 0.5076577920327185
```

```
for i in range(len(alpha )) :  
    print(alpha[i], get_params(lasso_re
```

```
1e-05 0.5076504258686468  
0.0001 0.5076557831609503  
0.001 0.5078215088561538  
0.005 0.5101572297319558  
0.01 0.5133823201276145  
0.05 0.5310503621742259  
0.1 0.578562122360455  
1 0.6871396246833519  
5 0.6928072369262379  
7 0.6932286767833257  
10 0.6932357326606897
```

SAMPLE

## &lt;최적 alpha값을 릿지, 라쏘 모델에 대입&gt;

```
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)

ridge_reg = Ridge(alpha=0.05) # 위에서 구한 최적의 알파 값(0.05)을 릿지에 대입
ridge_reg.fit(X_train, y_train)

lasso_reg = Lasso(alpha=0.0001) # 위에서 구한 최적의 알파 값(0.0001)을 라쏘에 대입
lasso_reg.fit(X_train, y_train)

# 모든 모델의 RMSE 출력
models = [lr_reg, ridge_reg, lasso_reg]
get_rmses(models)
```

SAMPLE

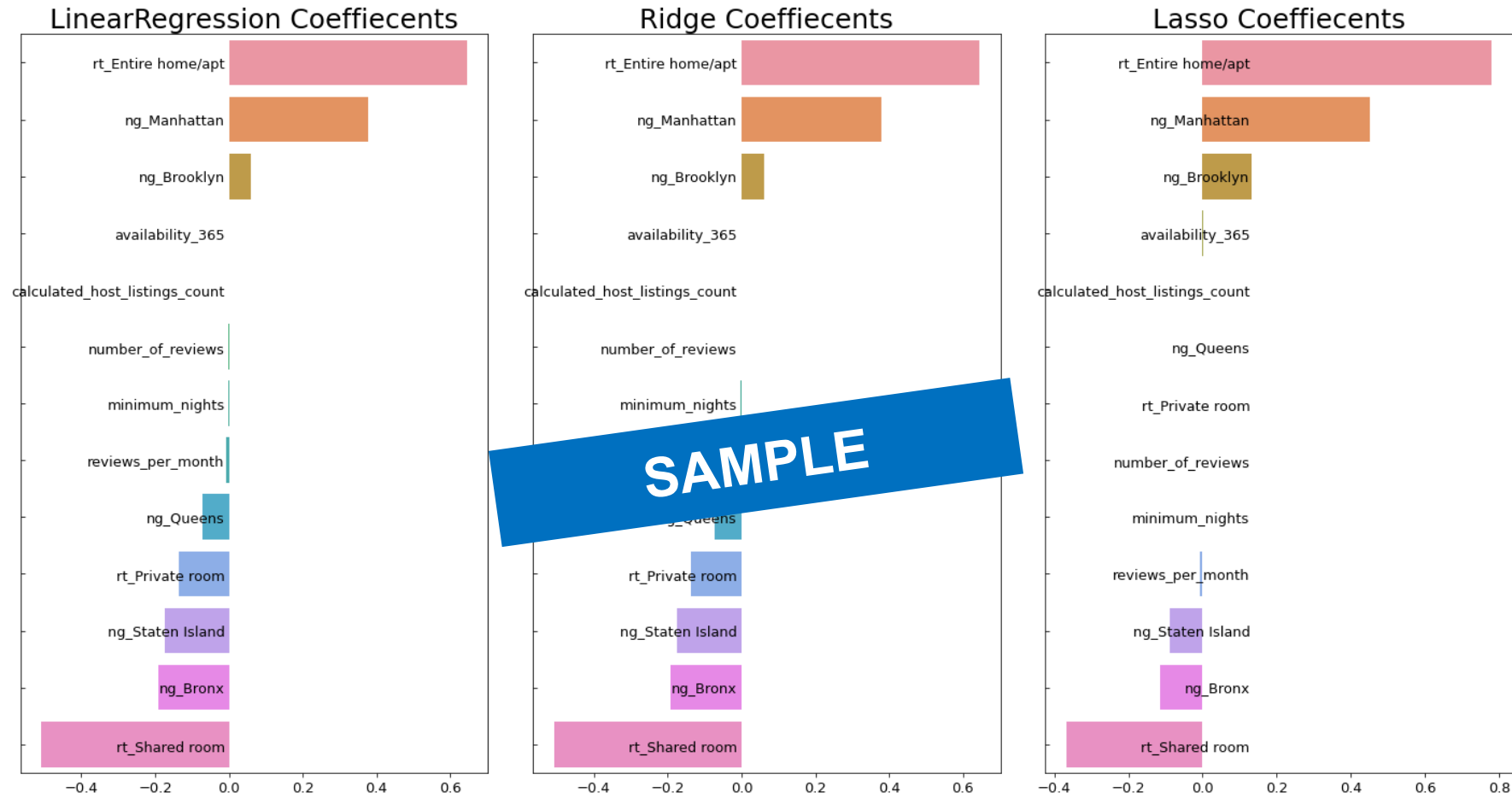
## &lt;최적 alpha값을 릿지, 라쏘 모델에 대입&gt;

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	Ridge RMSE : 0.507 Lasso RMSE : 0.507	Ridge RMSE : 0.509 Lasso RMSE : 0.509
availability_365 중위수	Ridge RMSE : 0.505 Lasso RMSE : 0.505	Ridge RMSE : 0.507 Lasso RMSE : 0.507
availability_365 365	Ridge RMSE : 0.512 Lasso RMSE : 0.512	Ridge RMSE : 0.514 Lasso RMSE : 0.514

SAMPLE

- availability\_365 '0'값을 중위수로 대체하고, 8:2 비율로 모델링했을 때 성능이 가장 좋음
- 라쏘 모델 성능 좋아짐 확인(0.685 -> 0.505)

## &lt;최적 alpha값을 모델에 대입 후 시각화&gt;



- 피쳐 중요도가 동일해졌으며, 라쏘 모델 성능 좋아짐을 확인

## &lt;데이터 세트 가공 후 튜닝 수행&gt;

```
skew_featsues = df_ohe[features_index].apply(lambda x : skew(x))
```

```
skew_features_top = skew_featsues[skew_featsues > 1] # 왜곡정도가 '1'보다 큰  
print(skew_features_top.sort_values(ascending=False)) # 내림차순 정렬
```

```
# 로그 변환
```

```
df_ohe[skew_features_top.index] = np.log1p(df_ohe[skew_features_top.index])
```

```
# 원-핫 인코딩
```

```
airbnb_df_ohe = pd.get_dummies(df_ohe)
```

```
# 피쳐/타겟 데이터 세트 다시 생성
```

```
y_target = airbnb_df_ohe['price']
```

```
X_features = airbnb_df_ohe.drop('price',axis=1, inplace=False)
```

```
# 학습 데이터와 테스트 데이터 분리
```

```
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.2, random_state=156)
```

```
# 피쳐들을 로그 변환 후 다시 최적 하이퍼 파라미터
```

```
ridge_params = { 'alpha' : [0.000001, 0.00001, 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 1, 5, 7, 10] }
```

```
lasso_params = { 'alpha' : [0.000001, 0.00001, 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 1, 5, 7, 10] }
```

```
best_ridge = get_best_params(ridge_reg, ridge_params)
```

```
best_lasso = get_best_params(lasso_reg, lasso_params)
```

```
# 앞의 최적화 alpha값으로 학습데이터로 학습, 테스트 데이터로 예측 및 평가 수행
```

```
lr_reg = LinearRegression()
```

```
lr_reg.fit(X_train, y_train)
```

```
ridge_reg = Ridge(alpha=0.05) # 위에서 구한 최적의 알파 값(0.05)을 릿지에 대입
```

```
ridge_reg.fit(X_train, y_train)
```

```
lasso_reg = Lasso(alpha=0.0001) # 위에서 구한 최적의 알파 값(0.0001)을 라쏘에 대입
```

```
lasso_reg.fit(X_train, y_train)
```

## &lt;데이터 세트 가공 전/후 최적 RMSE 비교&gt;

## 데이터 세트 가공 전

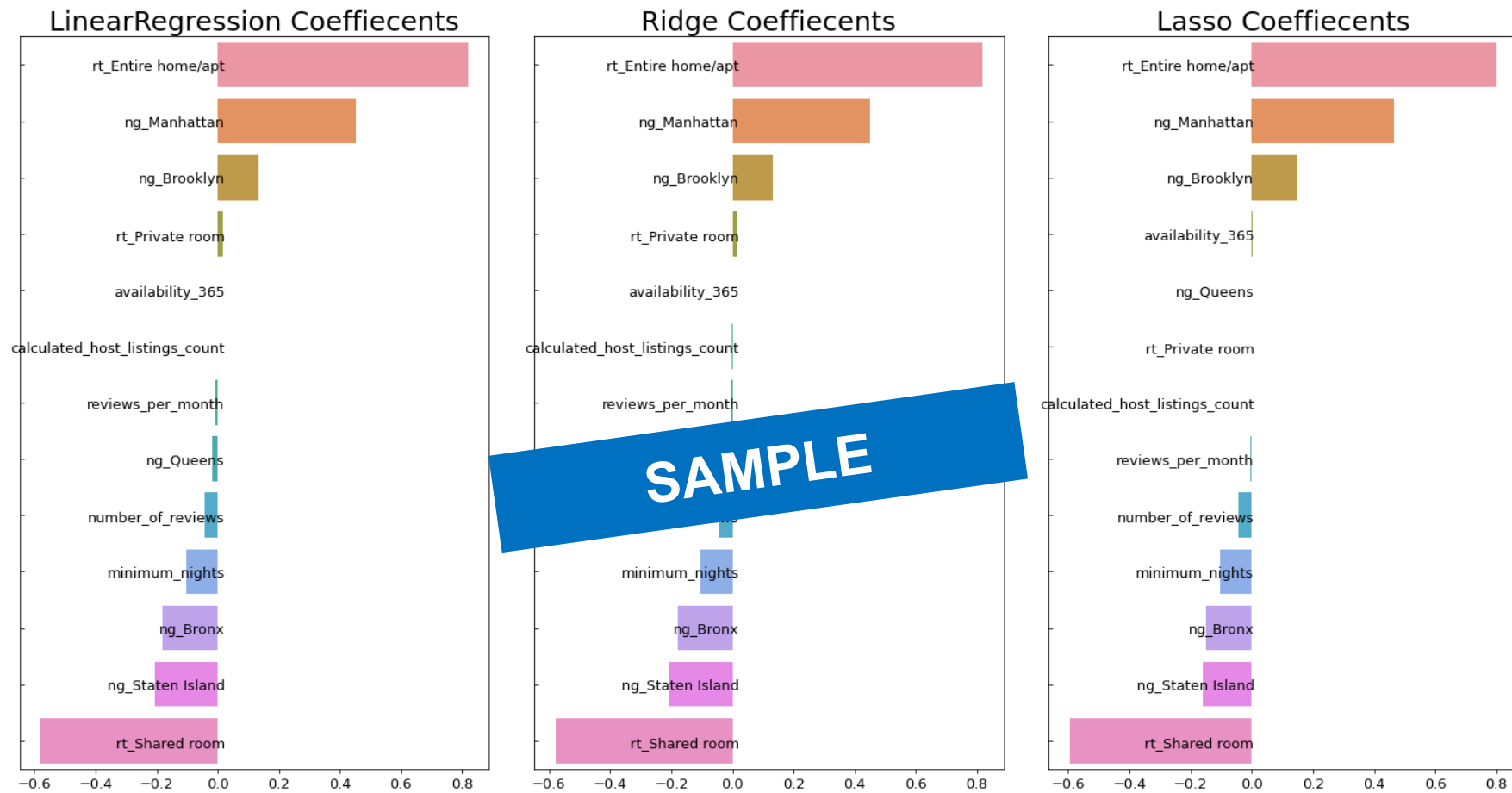
	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	Ridge RMSE : 0.507 Lasso RMSE : 0.507	Ridge RMSE : 0.509 Lasso RMSE : 0.509
availability_365 중위수	Ridge RMSE : 0.505 Lasso RMSE : 0.505	Ridge RMSE : 0.507 Lasso RMSE : 0.507
availability_365 365	Ridge RMSE : 0.512 Lasso RMSE : 0.512	Ridge RMSE : 0.514 Lasso RMSE : 0.514

## 데이터 세트 가공 후

	학습 및 테스트 데이터 분리 비율	
	8:2	7:3
availability_365 평균	Ridge RMSE : 0.503 Lasso RMSE : 0.503	Ridge RMSE : 0.504 Lasso RMSE : 0.504
availability_365 중위수	Ridge RMSE : 0.501 Lasso RMSE : 0.501	Ridge RMSE : 0.502 Lasso RMSE : 0.502
availability_365 365	Ridge RMSE : 0.509 Lasso RMSE : 0.509	Ridge RMSE : 0.51 Lasso RMSE : 0.51

- 데이터 세트 가공 후 근소하게 RMSE 값 줄어듦을 확인

## &lt;최적 alpha값을 모델에 대입 후 시각화&gt;



- 세 모델 모두 Entire\_home/apt를 가장 중요한 피처로 보고 있음
- room type이 숙소 가격(price)에 미치는 영향이 제일 높음을 확인

## &lt;XGB, LGBM 모델 혼합 – 50 : 50&gt;

```
[ ] xgb_reg = XGBRegressor(objective='reg:squarederror', n_estimators=1000, learning_rate=0.05, # 손실함수를 집어넣음
                           colsample_bytree=0.5, subsample=0.8)

lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                          subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)

xgb_reg.fit(X_train, y_train)
lgbm_reg.fit(X_train, y_train)
xgb_pred = xgb_reg.predict(X_test)
lgbm_pred = lgbm_reg.predict(X_test)
```

**SAMPLE**

```
[ ] pred = 0.5 * xgb_pred + 0.5 * lgbm_p
preds = {'최종 혼합': pred,
        'XGBM': xgb_pred,
        'LGBM': lgbm_pred}

get_rmse_pred(preds)
```

최종 혼합 모델의 RMSE: 0.474010058627334

XGBM 모델의 RMSE: 0.47506216807301455

LGBM 모델의 RMSE: 0.48014583476722966



## &lt;XGB, LGBM 모델 혼합 – 50 : 50&gt;

	XGB(0.5) + LGBM(0.5)	
	8:2	7:3
availability_365 평균	0.470936	0.474010
availability_365 중위수	SAMPLE	0.473722
availability_365 365		0.479297

- 데이터 분리 비율 8:2일 때, XGB(0.5) + LGBM(0.5)의 성능이 가장 좋음