ESC FINAL PROJECT 기말발표

주제: **Linear Regression**을 통한 아파트 경매 가격 상승률 예측

1조 강태환 김동휘 이재현 조유림 최익준

1. EDA

1) 중간발표 중요 내용

- 날짜 관련 파생변수 생성 : 최종 경매일과 최초 경매일 차이
- 시군구/감정사 클러스터링 진행 후 더미변수 생성
- Y와의 correlation이 높으면서 서로 correlation이 높은 6개의 변수 → PCA를 통해 차원 축소

2) 변경 사항

1. floor(층수) → skyscraper 더미변수 생성

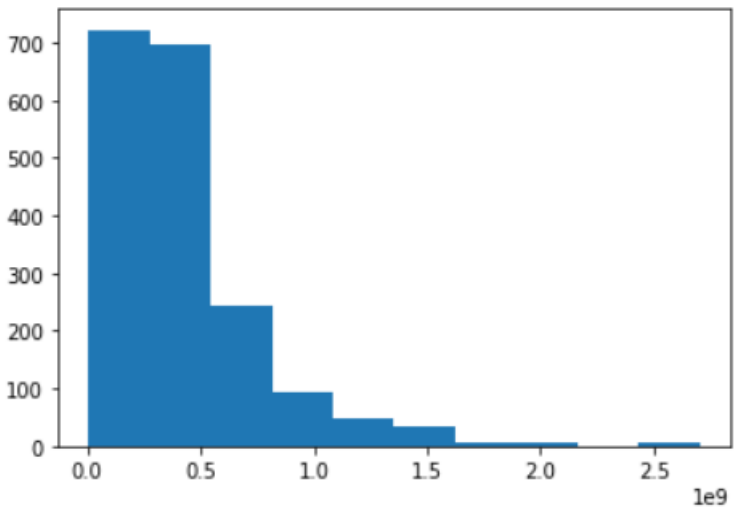
30층 이상의 고층건물은 1, 30층 미만은 0의 값을 부여해 더미 변수 생성

2. Hammer price/ minimum sales price(최저 매각 가격) Y값 새롭게 생성하는 방식 → **기존의 Y값**인 Hammer price를 예측하기로 결정

변환한 y값에 대해서 OLS 모형의 설명력이 너무 낮았음

3. Y값에 대해 **BOX-COX Transformation** 진행

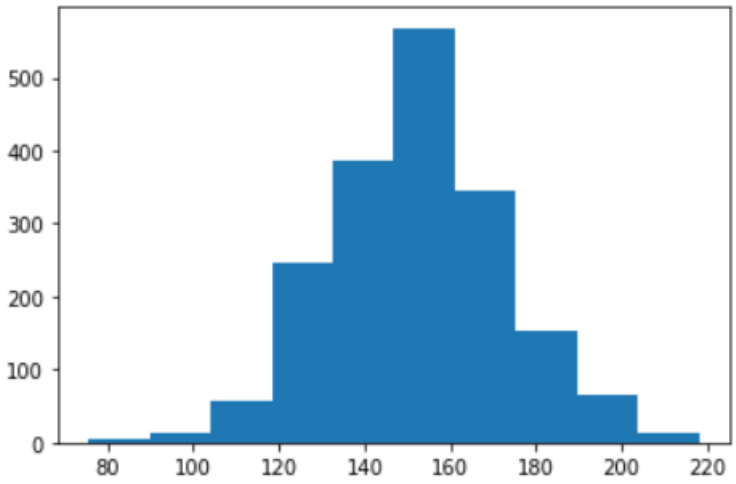
```
plt.hist(data['Hammer_price'])
plt.show()
```



기존의 hammer price 분포는 정규분포 형태가 아니다. OLS 가정사항 위반!

```
lambda_boxcox = boxcox(data['Hammer_price'])[1]
data['Hammer_price'] = boxcox(data['Hammer_price'])[0]
```

```
plt.hist(data['Hammer_price'])
plt.show()
```



BOX COX Transformation을 통해 정규분포 형태로 만들어줌

2. 모델링

1) Frequentist 관점 OLS

backward selection을 통해 변수 선택

> 단계1 : 변수 전체 사용

```
from statsmodels.formula.api import ols
```

```
res = ols('Hammer_price ~ PC1 + PC2 + Claim_price + Auction_count +
Final_First_auction_data + ad_si_0 + ad_si_1 + ad_si_2 + Appr_0 + Appr_1 + 서울 +
skyscraper',data = data).fit()

res.summary()
```

OLS Regression Results

Dep. Variable:	Hammer_price	**R-squared:**	0.867
Model:	OLS	**Adj. R-squared:**	0.866
Method:	Least Squares	**F-statistic:**	1004.
Date:	Wed, 02 Jun 2021	**Prob (F-statistic):**	0.00
Time:	11:18:30	**Log-Likelihood:**	-6356.4
No. Observations:	1855	**AIC:**	1.274e+04
Df Residuals:	1842	**BIC:**	1.281e+04
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	138.4719	4.073	34.001	0.000	130.484	146.459
PC1	7.0462	0.105	66.989	0.000	6.840	7.253
PC2	-5.4190	0.195	-27.841	0.000	-5.801	-5.037
Claim_price	-1.59e-10	1.6e-10	-0.997	0.319	-4.72e-10	1.54e-10
Auction_count	-3.5842	0.229	-15.655	0.000	-4.033	-3.135
Final_First_auction_data	0.0038	0.001	2.898	0.004	0.001	0.006
ad_si_0	3.0595	0.763	4.010	0.000	1.563	4.556
ad_si_1	3.3228	0.441	7.535	0.000	2.458	4.188
ad_si_2	5.2771	0.956	5.518	0.000	3.401	7.153
Appr_0	-1.3242	1.029	-1.287	0.198	-3.343	0.694
Appr_1	-21.1708	7.570	-2.797	0.005	-36.018	-6.324
서울	10.2006	0.409	24.928	0.000	9.398	11.003
skyscraper	0.4129	0.132	3.117	0.002	0.153	0.673

Omnibus:	554.932	**Durbin-Watson:**	2.122
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2222.534
Skew:	-1.404	**Prob(JB):**	0.00
Kurtosis:	7.568	**Cond. No.**	5.03e+10

- Claim price의 p-value가 0.319로 제거

> 단계2 : Claim price 제거 후 fitting

```
res2 = ols('Hammer_price ~ PC1 + PC2 + Auction_count + Final_First_auction_data + ad_si_0 + ad_si_1 + ad_si_2 + Appr_0 + Appr_1 + 서울 + skyscraper',data = data).fit()

res2.summary()
```

Dep. Variable:	Hammer_price	**R-squared:**	0.867
Model:	OLS	**Adj. R-squared:**	0.866
Method:	Least Squares	**F-statistic:**	1095.
Date:	Wed, 02 Jun 2021	**Prob (F-statistic):**	0.00
Time:	11:18:30	**Log-Likelihood:**	-6356.9
No. Observations:	1855	**AIC:**	1.274e+04
Df Residuals:	1843	**BIC:**	1.280e+04
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	138.3632	4.071	33.986	0.000	130.379	146.348
PC1	7.0346	0.105	67.292	0.000	6.830	7.240
PC2	-5.4089	0.194	-27.827	0.000	-5.790	-5.028
Auction_count	-3.5865	0.229	-15.666	0.000	-4.035	-3.137
Final_First_auction_data	0.0038	0.001	2.871	0.004	0.001	0.006
ad_si_0	3.0427	0.763	3.989	0.000	1.547	4.539
ad_si_1	3.3059	0.441	7.503	0.000	2.442	4.170
ad_si_2	5.2602	0.956	5.501	0.000	3.385	7.136
Appr_0	-1.3189	1.029	-1.281	0.200	-3.337	0.700
Appr_1	-20.9751	7.568	-2.772	0.006	-35.817	-6.133
서울	10.1912	0.409	24.912	0.000	9.389	10.994
skyscraper	0.4154	0.132	3.137	0.002	0.156	0.675

Omnibus:	555.406	**Durbin-Watson:**	2.123
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2222.982
Skew:	-1.406	**Prob(JB):**	0.00
Kurtosis:	7.567	**Cond. No.**	7.49e+03

- Appr_0 변수의 p-value값이 0.2로 유의하지 않음
- Appr_0는 감정사 변수에 대한 dummy variable로 Appr_1 변수도 함께 제거해주기로 함

> 단계3 : 최종모형

```
res3 = ols('Hammer_price ~ PC1 + PC2 + Auction_count + Final_First_auction_data + ad_si_0 + ad_si_1 + ad_si_2 + 서울 + skyscraper', data = data).fit()

res3.summary()
```

Dep. Variable:	Hammer_price	R-squared:	0.867
Model:	OLS	Adj. R-squared:	0.866
Method:	Least Squares	F-statistic:	1332.
Date:	Tue, 01 Jun 2021	Prob (F-statistic):	0.00
Time:	19:05:37	Log-Likelihood:	-6361.5
No. Observations:	1855	AIC:	1.274e+04
Df Residuals:	1845	BIC:	1.280e+04
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	137.7325	4.070	33.842	0.000	129.750	145.715
PC1	6.9914	0.103	67.573	0.000	6.788	7.194
PC2	-5.3327	0.192	-27.715	0.000	-5.710	-4.955
Auction_count	-3.5937	0.229	-15.668	0.000	-4.044	-3.144
Final_First_auction_data	0.0039	0.001	2.942	0.003	0.001	0.006
ad_si_0	3.1037	0.762	4.071	0.000	1.608	4.599
ad_si_1	3.3343	0.441	7.555	0.000	2.469	4.200
ad_si_2	5.0524	0.953	5.303	0.000	3.184	6.921
서울	10.2332	0.410	24.985	0.000	9.430	11.036
skyscraper	0.4339	0.132	3.276	0.001	0.174	0.694

Omnibus:	551.890	Durbin-Watson:	2.123
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2194.362
Skew:	-1.398	Prob(JB):	0.00
Kurtosis:	7.535	Cond. No.	4.02e+03

- 모든 변수가 유의하다고 나옴
- 최종 변수 : PC1, PC2, Auction_count(총 경매횟수), Final_First_auction_data(최종 경매일과 최초 경매일 차이), 시군구 더미변수, 시도 더미변수, 고층건물 더미변수
- adjusted r square값은 0.866

2) Bayesian Linear Regression

> 단계 1 . Model Selection

```
class Model:

    def z_function(self,data,category_index):
        p = data.shape[1]
        category = [i for i in range(p)]
        Ncategory = []
        for i in category_index:
            Ncategory.append(i)
            for j in i:
                category.remove(j)
        x = [[i] for i in category]
        for i in Ncategory:
            x.append(i)
        result = []
        cnt=0
        for i in range(len(x)):
            count = len(list(combinations(x,i+1)))
            for j in range(count):
```

```
                z = [0]*p
                for k in range(i+1):
                    if len(list(combinations(x,i+1))[j][k]) == 1: #(1,2)
                        z[list(combinations(x,i+1))[j][k][0]] = 1
                    else:
                        for m in range(len(list(combinations(x,i+1))[j][k])):#(1,
(2,3,4))
                            z[list(combinations(x,i+1))[j][k][m]] = 1
                result.append(z)
        return result

    def sig(self,X,y):
        X = np.array(X)
        n = X.shape[0]
        yhat = X@inv(t(X)@X)@t(X)@y
        res = y-yhat
        return sum(res**2)/n


    def posterior(self,X,y,category_index,g):
        nu0 = 1
        z = Model.z_function(self,X,category_index)
        y = np.array(y)
        l = []
        for i in z:
            i = np.array(i)
            Xz = X.iloc[:,np.where(i==1)[0]]
            n,p = Xz.shape
            Xz = np.array(Xz)
            sig0 = Model.sig(self,Xz,y)
            ssr = t(y)@(np.eye(n)-g/(g+1)*Xz@inv(t(Xz)@Xz)@t(Xz))@y
            loglikelihood = (-n/2)*log(np.pi)+loggamma((nu0+n)/2)-
loggamma(nu0/2)-p/2*log(1+g)+nu0/2*log(nu0*sig0)-(nu0+n)/2*log(nu0*sig0+ssr)
            l.append(loglikelihood)
        l = l/sum(l)
        return l
```

z_function(self,data,category_index)

- PC1 + PC2 + Claim_price + Auction_count + Final_First_auction_data + ad_si_0 + ad_si_1 + ad_si_2 + Appr_0 + Appr_1 + 서울 + skyscraper 변수 포함여부를 z 로 나타냄
- 더미 변수에 대해서는 제거할 때 동시에 제거하고 포함할 때는 동시에 포함하도록 설계

```
z = m.z_function(X,[[5,6,7],[8,9]]) # 카테고리 변수의 인덱스를 입력할 수 있도록 함
```

```
[[1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
 [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
 [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 ...
```

- 511 경우의 수!!

posterior(self,X,y,category_index,g)

- weak prior 부여

$\rightarrow g = n,\ \nu_0 = 1, \sigma_0^2 = \text{estimated residual variance under the least squares estimate}$

$\rightarrow$ unit information prior for $p(\sigma^2)$

- log likelihood값을 이용해 posterior값을 구함

	model	posterior
0	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.001954
1	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.001955
2	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.001955
3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]	0.001955
4	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	0.001955

```
np.argmax(posterior)
509

result.iloc[509,:]
model        [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

- PC1 변수를 제외하고 다른 변수 모두 사용하기로 결정

> **단계2 β 샘플링**

```
from scipy.stats import gamma
from scipy.stats import multivariate_normal

BETA = []

for i in range(1000):
    precision_MC = gamma.rvs(a = nu0+n, scale = (nu0+n)/((nu0*s0)+ssr),size=1)
    cov_MC = 1/precision_MC * inv(t(Xz).dot(Xz))
    beta_MC = multivariate_normal.rvs(mean = beta * g/(g+1),
cov=cov_MC*g/(g+1),size=1)
    BETA.append(beta_MC)

pd.DataFrame(BETA).mean()
```

- Monte Carlo approximation 을 이용해 beta 샘플링

$$\text{sample } 1/\sigma^2 \sim \text{gamma}\left(\left[\nu_0+n\right]/2, \left[\nu_0\sigma_0^2+\text{SSR}_g\right]/2\right);$$
$$\text{sample } \boldsymbol{\beta} \sim \text{ multivariate normal } \left(\tfrac{g}{g+1}\hat{\boldsymbol{\beta}}_{\text{ols}}, \tfrac{g}{g+1}\sigma^2\left[\mathbf{X}^T\mathbf{X}\right]^{-1}\right).$$

```
0     87.368541 #intercept
1     -1.632595 #PC2
2      0.000023 #Claim price
3     -2.945322 #Auction count
4      0.005012 #Final_First_auction_data
5     24.100314 #ad_si_0
6      9.487440 #ad_si_1
7     30.645763 #ad_si_2
8      7.222672 #Appr_0
9     29.947249 #Appr_1
10    14.348095 #서울
11     1.803751 #skyscrapper
```

- weak prior를 사용했기 때문에 일반 OLS에서 추정한 계수와 값이 유사한 것을 확인할 수 있음

	coef
Intercept	87.4158
PC2	-1.6335
Claim_price	1.024e-09
Auction_count	-2.9464
Final_First_auction_data	0.0049
ad_si_0	24.1131
ad_si_1	9.4923
ad_si_2	30.6621
Appr_0	7.2263
Appr_1	29.9635
서울	14.3558
skyscraper	1.8042

3. Test data에 적용

Test data에 train data와 동일한 방식으로 전처리 진행

> 주요내용

1. 시군구/ 감정사

- train data에서 클러스터링 한 결과를 그대로 사용
- train data에 없는 감정사는 가장 큰 규모의 군집으로 넣어줌

```
for i in range(829):
    if testdata['addr_si'][i] == '서초구':
        testdata['ad_si_0'][i] = 1
    elif testdata['addr_si'][i] == '용산구':
        testdata['ad_si_0'][i] = 1
    elif testdata['addr_si'][i] == '송파구':
        testdata['ad_si_0'][i] = 1
```

2. PCA

- train data에서 구한 loading 값을 이용해서 계산


```
PC2 = []
for i in range(820):
    PC2.append(PC11[i]+PC22[i]+PC33[i]+PC44[i]+PC55[i]+PC66[i])
```

3. Y값을 구한 후에 inverse-transformation 진행

```
prehammer2 = inv_boxcox(hammer2, lambda_boxcox)
```

> 결과 : RMSE 기준

tt.csv edit	2021-06-02 23:23:11	238613007.3490338 189620750.87831324	○
tt2.csv edit	2021-06-02 18:35:47	180564322.9969324 172375958.09373492	○

- 일반회귀 : RMSE 238613007
- Bayesian Linear Regression : RMSE 180564322
- Bayesian Linear Regression의 결과가 더 좋은 것을 확인할 수 있음