EXPANDED STATISTIC CLUB

ESC 23-1

# 파이널 프로젝트 2조

박정현 김종민 김시은

# 01

# EDA

# Step 1. 이상치 제거

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

수치형 자료에 대한 이상치를 IQR을 이용하여 제거하였음
Q1과 Q3 각각과 IQR * 1.5보다 많이 (양 끝 방향으로) 떨어져 있는 데이터를 이상치로 간주

```
def detect_outliers(df, n, features):
    outlier_indices = []
    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col], 75)
        IQR = Q3 - Q1

        outlier_step = 1.5 * IQR

        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step)].index
        outlier_indices.extend(outlier_list_col)
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(k for k, v in outlier_indices.items() if v > n)

    return multiple_outliers
```

Multiple Features에 대해 3개 이상의 outlier를 포함하고 있는 경우 해당 행을 제외하고 분석

# Step 2. 자료 시각화 및 변수 선택

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

```
fig, axs = plt.subplots(nr_rows, nr_cols, figsize=(nr_cols*4,nr_rows*3))

for r in range(0,nr_rows):
    for c in range(0,nr_cols):
        i = r*nr_cols+c
        if i < len(li_cat_feats):
            sns.boxplot(x=li_cat_feats[i], y=train["SalePrice"], data=train[cat_selected], ax = axs[r][c])

plt.tight_layout()
plt.show()
```

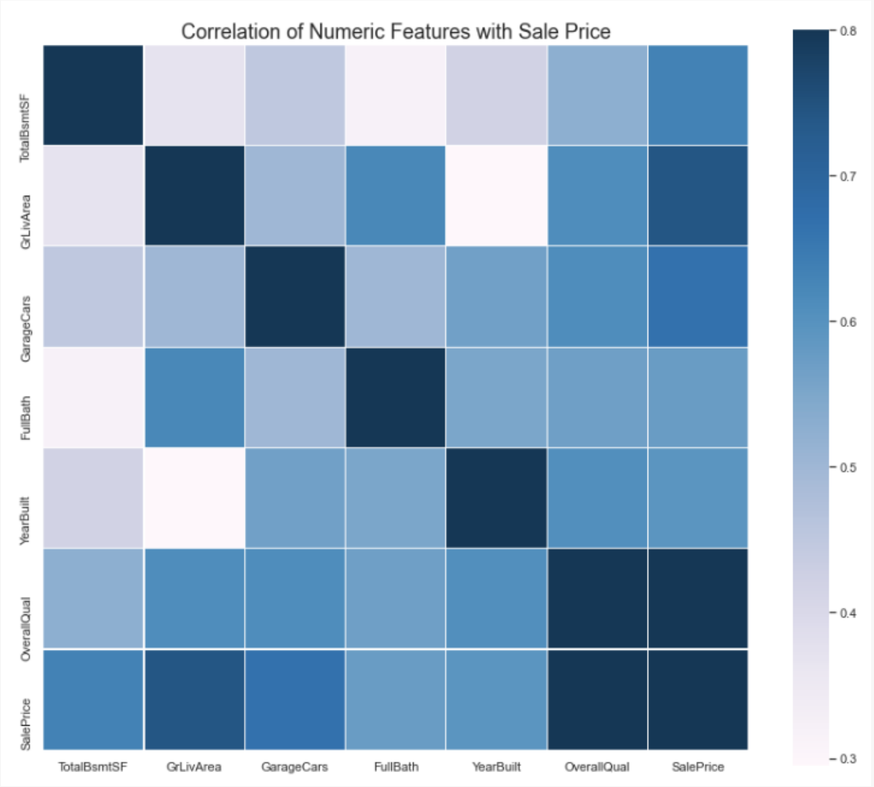
범주형 변수 각각에 대한 그래프를 그린 후 범주에 따라 SalePrice의 차이가 두드러지는 일부 변수를 선택

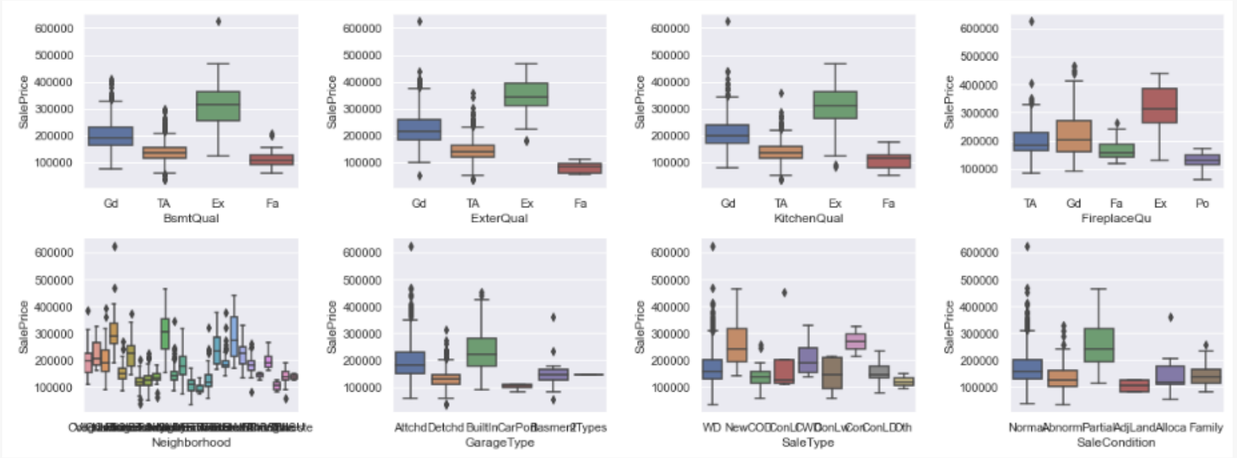
```
colormap = plt.cm.PuBu
sns.set(font_scale=1.0)

f , ax = plt.subplots(figsize = (14,12))
plt.title('Correlation of Numeric Features with Sale Price',y=1,size=18)
sns.heatmap(corr_data.corr(),square = True, linewidths = 0.1,
            cmap = colormap, linecolor = "white", vmax=0.8)
```

수치형 변수 전체에 대한 Correlation Map을 그린 후, SalePrice와 높은 상관관계를 보이는 일부 변수를 선택
이때 해당 map과 data description txt를 참고하여, 다른 변수가 해당 변수에 대한 정보를 포함하고 있다고 판단되는 경우 선택에서 삭제하였음

# Step 2. 자료 시각화 및 변수 선택

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행




(위측) 선택된 범주형 변수들의 Sale Price에 대한 Plot
(좌측) 선택된 수치형 변수들과 Sale Price 간의 Correlation Map

선택된 범주형 변수 중 BsmtQual, ExterQual, KitchenQual, FireplaceQu는
수치형 변수인 OveralQual로 대표될 수 있다고 판단하여 변수 선택에서 제외

# Step 3. 결측치 제거

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

```
selected_train.isnull().sum()
```

Train Data의 결측치를 확인한 결과 'GarageType'에서 65개의 결측치 존재, 이외 결측치는 없는 것으로 확인되었음
이때 'GarageType'과 같은 몇 개의 변수에 대한 NaN은 '측정되지 않음'이 아닌 '없음'을 의미 (예: 집에 Garage가 존재하지 않음)

```
for col in cols_fillna:
    selected_train[col].fillna('None',inplace=True)
    selected_test[col].fillna('None',inplace=True)
```

이와 같은 경우 결측치를 'None'이라는 새로운 범주로 대체

```
final_test = final_test.dropna()
```

동일한 작업을 거친 test data에는 총 2개의 결측치가 존재, 정보 손실이 크지 않다고 판단하여 해당 행을 삭제하였음

# Step 4. 범주형 변수 변환

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

범주형 변수를 Regression에 사용할 수 있도록 수치형 자료로 변환하는 과정을 거침

```
N_meanprice = selected_train.groupby('Neighborhood')['SalePrice'].mean().sort_values()
ST_meanprice = selected_train.groupby('SaleType')['SalePrice'].mean().sort_values()
G_meanprice = selected_train.groupby('GarageType')['SalePrice'].mean().sort_values()
SC_meanprice = selected_train.groupby('SaleCondition')['SalePrice'].mean().sort_values()
```

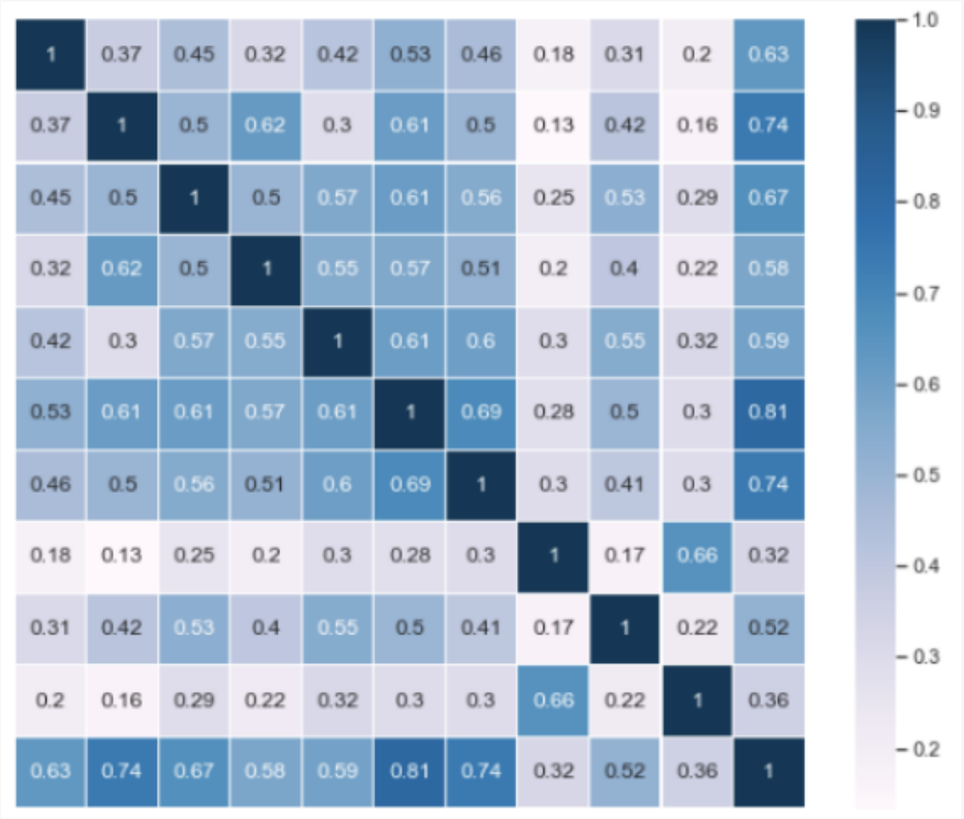
다양한 방법이 있겠으나 프로젝트의 목적을 고려하여 단순하게 진행하였음
특정 변수 내 범주들을 Sale Price의 평균을 기준으로 정렬 후 일정한 구간 단위로 잘라 구간의 순위를 indexing (Neighborhood의 예시)

```
N_catg1 = ['MeadowV', 'IDOTRR', 'BrDale', 'BrkSide', 'Edwards', 'OldTown', 'Sawyer', 'Blueste', 'SWISU', 'NPkVill', 'NAmes']
N_catg2 = ['Mitchel', 'SawyerW', 'NWAmes', 'Gilbert', 'Blmngtn', 'CollgCr']
N_catg3 = ['Crawfor', 'ClearCr', 'Somerst', 'Veenker', 'Timber']
N_catg4 = ['StoneBr', 'NridgHt', 'NoRidge']

for df in [selected_train, selected_test]:

    df['Nb_num'] = 1
    df.loc[(df['Neighborhood'].isin(N_catg2)), 'Nb_num'] = 2
    df.loc[(df['Neighborhood'].isin(N_catg3)), 'Nb_num'] = 3
    df.loc[(df['Neighborhood'].isin(N_catg4)), 'Nb_num'] = 4
```

# Step 4. 범주형 변수 변환

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

```
final_label = ['TotalBsmtSF', 'GrLivArea', 'GarageCars', 'FullBath', 'YearBuilt', 'OverallQual'
               'Nb_num', 'ST_num', 'GT_num', 'SC_num', 'SalePrice']
```




결과적으로 수치형 변수 6개, (변환된) 범주형 변수 4개가 분석에 사용되었음

(공간 관련) TotalBsmtSF, GrLivArea, GarageCars, FullBath, GarageType

(시기 관련) YearBuilt

(상태 관련) OverallQual

(위치 관련) Neighborhood

(거래 관련) SaleType, SaleCondition

10개의 독립 변수와 종속 변수인 SaleType의 관계를 가장 잘 나타내는 Coefficient를 4가지 방법으로 탐색하는 것이 과제의 목적

# Step 5. 사용 데이터 완성

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

| | TotalBsmtSF | GrLivArea | GarageCars | FullBath | YearBuilt | OverallQual | Nb_num | ST_num | GT_num | SC_num | SalePrice |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 856 | 1710 | 2 | 2 | 2003 | 7 | 2 | 3 | 3 | 3 | 208500 |
| 1 | 1262 | 1262 | 2 | 2 | 1976 | 6 | 3 | 3 | 3 | 3 | 181500 |
| 2 | 920 | 1786 | 2 | 2 | 2001 | 7 | 2 | 3 | 3 | 3 | 223500 |
| 3 | 756 | 1717 | 3 | 1 | 1915 | 7 | 3 | 3 | 2 | 2 | 140000 |
| 4 | 1145 | 2198 | 3 | 2 | 2000 | 8 | 4 | 3 | 3 | 3 | 250000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1333 | 953 | 1647 | 2 | 2 | 1999 | 6 | 2 | 3 | 3 | 3 | 175000 |
| 1334 | 1542 | 2073 | 2 | 2 | 1978 | 6 | 2 | 3 | 3 | 3 | 210000 |
| 1335 | 1152 | 2340 | 1 | 2 | 1941 | 7 | 3 | 3 | 3 | 3 | 266500 |
| 1336 | 1078 | 1078 | 1 | 1 | 1950 | 5 | 1 | 3 | 3 | 3 | 142125 |
| 1337 | 1256 | 1256 | 1 | 1 | 1965 | 5 | 1 | 3 | 3 | 3 | 147500 |

1338 rows × 11 columns

# Step 5. 사용 데이터 완성

*알고리즘 구현이 목적인 프로젝트로, 매우 간단한 수준에서 전처리를 진행

| | TotalBsmtSF | GrLivArea | GarageCars | FullBath | YearBuilt | OverallQual | Nb_num | ST_num | GT_num | SC_num | SalePrice |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 882.0 | 896 | 1.0 | 1 | 1961 | 5 | 1 | 3 | 3 | 3 | 169277.052498 |
| 1 | 1329.0 | 1329 | 1.0 | 1 | 1958 | 6 | 1 | 3 | 3 | 3 | 187758.393989 |
| 2 | 928.0 | 1629 | 2.0 | 2 | 1997 | 5 | 2 | 3 | 3 | 3 | 183583.683570 |
| 3 | 926.0 | 1604 | 2.0 | 2 | 1998 | 6 | 2 | 3 | 3 | 3 | 179317.477511 |
| 4 | 1280.0 | 1280 | 2.0 | 2 | 1992 | 8 | 4 | 3 | 3 | 3 | 150730.079977 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1454 | 546.0 | 1092 | 0.0 | 1 | 1970 | 4 | 1 | 3 | 1 | 3 | 167081.220949 |
| 1455 | 546.0 | 1092 | 1.0 | 1 | 1970 | 4 | 1 | 3 | 1 | 2 | 164788.778231 |
| 1456 | 1224.0 | 1224 | 2.0 | 1 | 1960 | 5 | 2 | 3 | 2 | 2 | 219222.423400 |
| 1457 | 912.0 | 970 | 0.0 | 1 | 1992 | 5 | 2 | 3 | 1 | 3 | 184924.279659 |
| 1458 | 996.0 | 2000 | 3.0 | 2 | 1993 | 7 | 2 | 3 | 3 | 3 | 187741.866657 |

1457 rows × 11 columns

# 02

# Regression

# 데이터 준비

*코드 및 개념에 대한 설명은 생략 (2, 3조의 중간 발표 참고)

```
train = pd.read_csv("final_train.csv", index_col=0)
test = pd.read_csv("final_test.csv", index_col=0)
```

자료를 불러온 후 MinMaxScaler를 통해 Scaling 진행

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
train_scale = scaler.fit_transform(train)
test_scale = scaler.transform(test)
```

아래와 같은 Input을 사용

```
N = train_scale.shape[0]
one_mat = np.ones((N,1))

X = np.concatenate((one_mat,train_scale[:,0:10]),axis=1)
y = train_scale[:,10]
init_b = np.ones(X.shape[1])
```

```
test_N = test_scale.shape[0]
one_test = np.ones((test_N,1))

test_X = np.concatenate((one_test,test_scale[:,0:10]),axis=1)
test_y = test_scale[:,10]
test_nw = test_y.reshape(-1, 1)
```

# Basic Code

*코드 및 개념에 대한 설명은 생략 (2, 3조의 중간 발표 참고)

```
def solveL(L,b): #forward
    n= L.shape[0] #행갯수
    y= np.zeros((n,1))
    for i in range(n):
        if L[i,i]==0: #역행렬존재x
            break
        y[i]=b[i]/L[i,i]
        for j in range(i+1,n):
            b[j]=b[j]-y[i]*L[j,i]
    return y

def solveB(L_T, Y): #backward
    n=L_T.shape[0]
    y= np.zeros((n,1))
    for i in range(n-1,-1,-1):
        if L_T[i,i]==0 :
            break
        y[i]=Y[i]/L_T[i,i]
        for j in range(0,i):
            Y[j]=Y[j]-y[i]*L_T[j,i]
    return y

def permutation_(P): #permutation matrix (qr분해에서 사용)
    n=P.shape[0]
    p=np.zeros((n,n))
    for i in range(n):
        p[P[i], i]=1
    return p
```

## Cholesky Factorization

```
b= np.dot(X.T,y)
A= X.transpose() @ X
L= scipy.linalg.cholesky(A,lower=True) #L 구하기
Y= solveL(L,b)

Beta_L=solveB(L.T,Y)
Beta_L
```

## QR Factorization

```
Q,R ,P= scipy.linalg.qr(X,pivoting=True,mode="economic")
p= permutation_(P)
b=np.dot(Q.T,y)
B_qr= solveB(R,b)
Beta_qr=np.dot(p,B_qr)
print(Beta_qr)
```

두 방식에서 동일한 coefficient가 얻어지며, test data에 대한 MSE는 아래와 같음

```
test_mse_qr= (1/test_N) * sse_cost(test_X, test_nw, Beta_qr)
print(test_mse_qr)
```

```
0.012831977531239 43
```

```
[[-0.14847409],
 [ 0.19811753],
 [ 0.32196596],
 [ 0.05387092],
 [-0.05568991],
 [ 0.04643968],
 [ 0.19655611],
 [ 0.0715691 ],
 [ 0.03318174],
 [ 0.00914754],
 [ 0.05058508]]
```

```
[[-0.14847409]
 [ 0.19811753]
 [ 0.32196596]
 [ 0.05387092]
 [-0.05568991]
 [ 0.04643968]
 [ 0.19655611]
 [ 0.0715691 ]
 [ 0.03318174]
 [ 0.00914754]
 [ 0.05058508]]
```

# 03

# Optimization

# Basic Code

두 방법 모두 Minimize할 Objective function으로 SSE를 사용

```python
def sse_cost(X, y, b):
    N = X.shape[0]
    pred = X @ b
    obj = np.sum((y - pred)**2)
    return obj


def gradient(X, y, b):
    N = X.shape[0]
    pred = X @ b
    grad = X.T @ (pred - y)
    return grad


def btl_search(X, y, b, alpha = 0.5, beta = 0.9):
    grad = gradient(X, y, b)
    step = 1
    while sse_cost(X, y, b - step * grad) > sse_cost(X, y, b) - alpha * step * np.linalg.norm(grad):
        step = beta * step
    return step


def grad_descent(X, y, init_b, tol = 1e-5, max_iter = 1000):
    b = init_b
    n_iter = 0
    for i in range(max_iter):
        grad = gradient(X, y, b)
        step = btl_search(X, y, b)
        if np.linalg.norm(b - (b-step*grad)) < tol:
            break
        b -= step * grad
        n_iter += 1
    return b
```

```python
from tqdm import tqdm
import scipy
import scipy.linalg
# newton method
def newton(x, y, init, epsilon, max_iterations = int(1e+5)):
    w = init
    for i in tqdm(range(max_iterations)):
        # prediction 계산
        y_pred = np.dot(x, w)

        # gradient 계산
        gradient = np.dot(x.T, (y_pred - y))

        # delta (hessian_inv * grad) calculate
        hessian = np.dot(x.T, x)
        L = scipy.linalg.cholesky(hessian, lower=True)
        k = solveL(L, gradient)

        delta = solveU(L.T, k).reshape(x.shape[1],-1)

        # backtracking
        step_size = backtracking(x, y, w)

        # update w (betas)
        w = w - step_size*delta

        lambda_ = np.dot(gradient.T, delta)

        if (lambda_ / 2) < epsilon:
            break

    return w
```

# Result

*코드 및 개념에 대한 설명은 생략 (2, 3조의 중간 발표 참고)

### Gradient Descent

```
[-0.14862958  0.21453262  0.31657352  0.05407722 -0.04776787  0.04300006
  0.18126404  0.07310345  0.03824237  0.01163932  0.04794742]
```

```
test_mse = (1/test_N) * sse_cost(test_X, test_y, b)
print(test_mse)
```

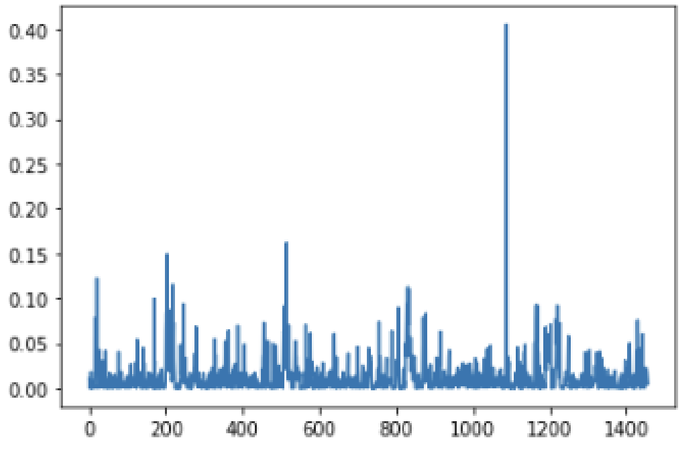
```
0.012830084945876888
```

### Newton's Method

두 방식을 사용하여 구한 coefficient가 서로 상당히 유사함을 확인할 수 있음

Test Data에 이 Linear Model을 적용하여 구한 MSE 역시 비슷한 수치로 얻어짐

```
[[-0.14796585],
 [ 0.19743936],
 [ 0.32086384],
 [ 0.05368651],
 [-0.05549928],
 [ 0.04628071],
 [ 0.19588328],
 [ 0.07132411],
 [ 0.03306815],
 [ 0.00911623],
 [ 0.05041193]]
```




```
test_mse_nw = (1/test_N) * sse_cost(test_X, test_nw, beta_est1)
print(test_mse_nw)
```

```
0.012747236228185323
```

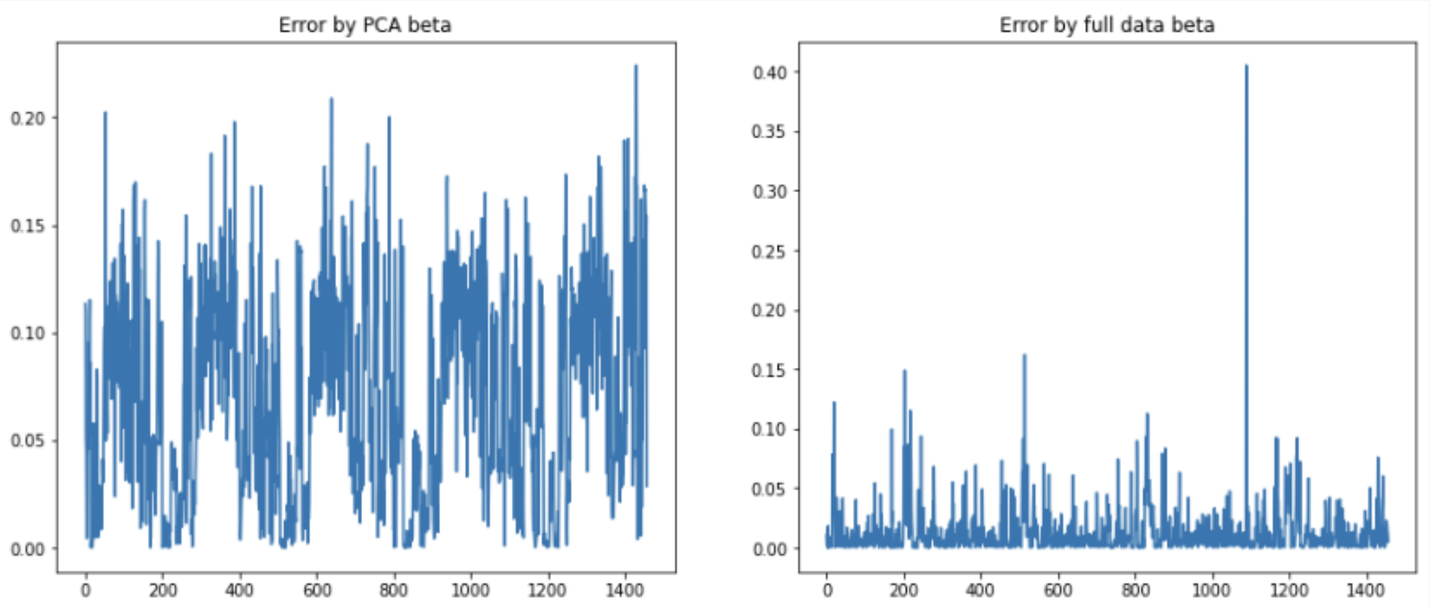
세 방식으로 구한 beta 추정값 간의 distance

```
Factorization & Gradient: 0.025459136829176113
Factorization & Newton: 0.0016078302982952917
Gradient & Newton: 0.025255045613234 89
```

# (Bonus) PCA

*코드 및 개념에 대한 설명은 생략 (2, 3조의 중간 발표 참고)

처음에는 별도의 변수 선택 없이 수치형 자료만을 사용한 PCA를 통해 분석을 진행하려고 계획하였음
이에 정리된 데이터 PCA 후 Newton's Method 돌려보는 방식을 추가적으로 시도




PCA 거친 Data를 사용한 경우에 비해 (전처리 된) 데이터 전체를 사용하였을 때의 예측력이 더 좋게 나타나 특별한 소득은 X
추후 원본 데이터에 PCA 사용한 경우와 직접 변수 선택을 한 경우를 비교해볼 수는 있을듯함

ESC 23-1 파이널 프로젝트 2조

# Thank you