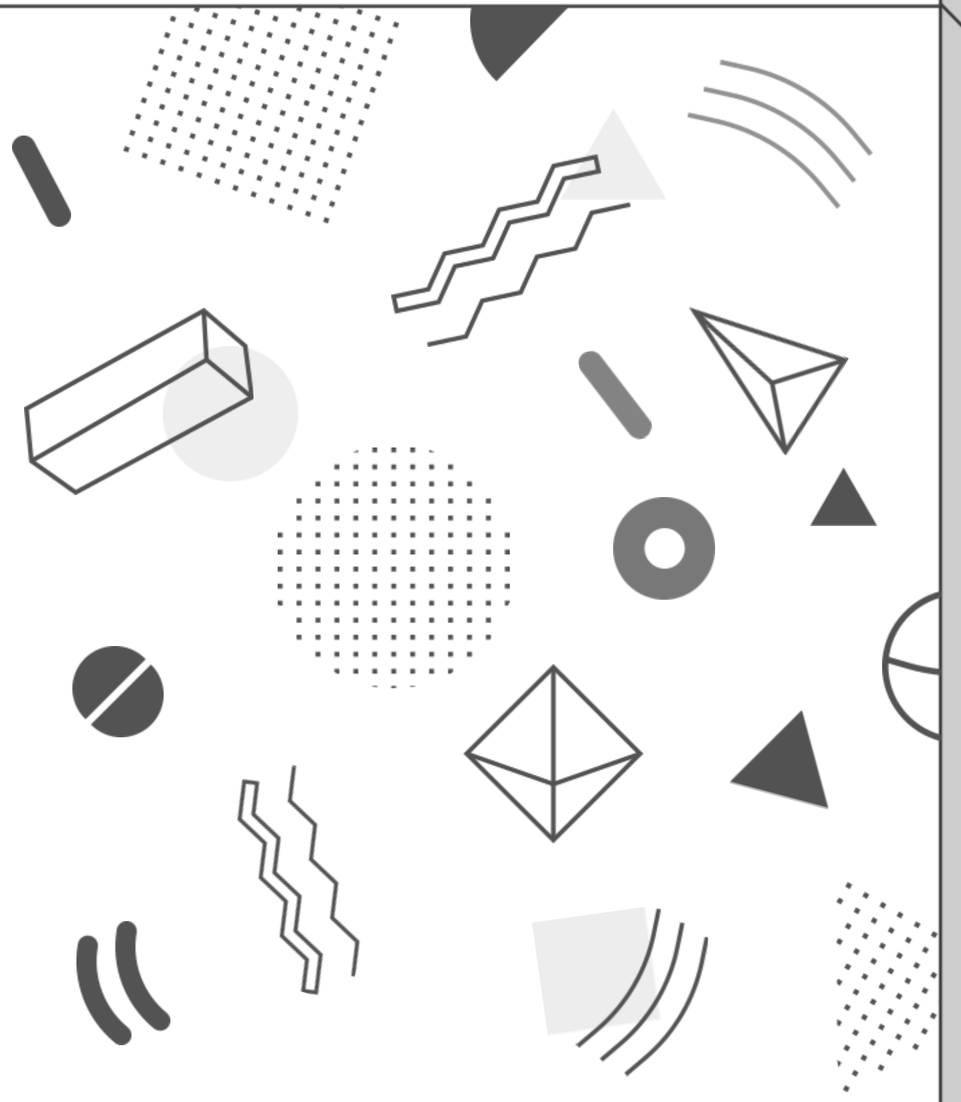
Kaggle 필사
Song Popularity Predict

발표자 : 이다현
@hopebii (깃헙)



노트 소개

도메인

: 음악 , 엔터테인먼트

데이터셋

: Spotify dataset 으로 특정 노래에 대한 음악적 특성들이 feature 로 존재한다. 여기서 우리는 회귀분석을 통해 song_popularity , 즉 노래의 인기도를 예측한다.

키워드

1. df copy

2. **변수 선택** : sklearn.feature selection method

3. **다중 공선성** multicollinearity
   - 해결방법 : VIF, RFE , PCA

4. **Regression** models
   - MLR, Ridge, Lasso, Elastic-Net, Polynomial
   - 결과 해석 : R-sqaured, P > |t|, Durbin-Watson 등

5. model evaluation : R2, RMSE

라이브러리

* Statmodels API : 회귀모델(ols)을 불러오기 위해 사용. Binomial, poisson, ANOVA 등을 불러올 수 있다.

```
# 전처리 라이브러리
from statsmodels.formula import api # 수식 문자열 및 데이터프레임을 사용해 모델을 지정하는 모듈
from sklearn.feature_selection import RFE # 변수 선택법 : 후진 제거법
from sklearn.preprocessing import StandardScaler # 스케일 조정
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor # 다중 공선성
```

```
# 모델링
from sklearn.decomposition import PCA # 차원축소
from sklearn.linear_model import Ridge # 정규화 회귀
from sklearn.linear_model import Lasso # 정규화 회귀
from sklearn.linear_model import ElasticNet # 정규화 회귀
from sklearn.linear_model import LinearRegression # 선형회귀
from sklearn.ensemble import RandomForestClassifier # 랜덤포레스트 분류기
from sklearn.preprocessing import PolynomialFeatures # 다항 회귀 (비선형 회귀)
```

```
# 성능 평가
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

변수선택법

모델을 돌릴 때 쓸모 없는 변수들을 제거함으로써 모델의 속도 개선, 오버피팅 방지 등의 효과를 얻음

1. Wrapper method : 모델을 돌리면서 변수 채택
   - 전진선택법, 후진제거법(RFE), stepwise

2. Filter method : 전처리 단계에서 통계 기법을 사용해 변수 채택
   - 분산기법(VarianceThreshold)
   - 단일변수선택기법 : Chi2, f_classif, mutual_info_classif, SelectKBest
   - Information value : 실제 프로젝트를 할 때 많이 쓰이는 방법

3. Embedded Method : 라쏘, 릿지, 엘라스틱넷 등 모델 안에서 L1,L2 정규화를 통해 변수를 축소해 사용

다중공선성

정의 : 하나의 독립변수가 다른 여러 개의 독립변수들로 잘 예측되는 경우를 말한다. 즉 독립변수의 일부가 다른 독립변수의 조합으로 표현될 수 있는 경우를 의미 (독립변수끼리 상관성이 높은 경우)

발견 방법 : sns_pairplot, sns_heatmap 그려보기, corr() 상관행렬 살펴보기

문제점 : 다중 공선성이 있으면 계수 추정이 잘 되지 않거나 불안정해져 데이터가 약간만 바뀌어도 추정치가 크게 달라진다. 통계적으로 유의미하지 않은 것처럼 나올 수 있다.

해결 방법

1. VIF, RFE(변수선택), PCA 방법을 기반으로 의존적인 변수들을 빼거나, 변수를 새로 생성한다.
2. 정규화 방법을 이용한다 (Scale)

VIF : 보통 값이 5나 10을 넘으면 다중 공선성이 있는 변수라고 판단한다. $\mathrm{VIF}_i = \frac{\sigma^2}{(n-1)\mathrm{Var}[X_i]} \cdot \frac{1}{1-R_i^2}$

PCA : 데이터의 차원을 줄임으로서 분산을 보존하면서 데이터의 다중공선성을 제거할 수 있다.

basic search

변수 파악하기

- song_duration_ms : The duration of the track in milliseconds. (음악 길이)
- acousticness : 0과 1사이의 값을 가지며 1의 값에 가까울수록 음향이 높은 음악이다.
- danceability : 춤에 적합한 노래인지 아닌지에 대한 수치로, 0~1사이의 값을 가지며 1에 가까울수록 댄스음악에 가깝다.
- energy : 0과 1사이의 값을 가지며 1에 가까울수록 energetic 한 사운드를 가진 음악이라 할 수 있다.

(energetic tracks feel fast, loud, and noisy. For example, death metal has high energy.)

- key : 예를 들어 0 = C, 1 = C♯/D♭, 2 = D 등 음의 계이름 높낮이를 정수로 나타냈다.
- instrumentalness : vocal 이 포함된 정도
- loudness : 전체 음량(dB) 값으로 일반적인 범위는 -60~0db 이다.
- speechiness : 음성 단어의 존재를 감지 (spoken words in a track)
- tempo : 박자
- mode : Major (장조) is represented by 1 and minor (단조) is 0.
- liveness : Detects the presence of an audience in the recording
- time_signature : An estimated overall time signature of a track

▫info() : 널값 없음

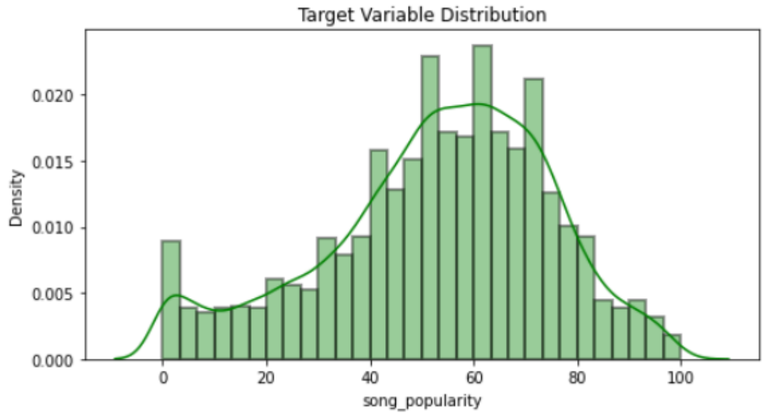
▫nunique() : 데이터가 모두 수치형이었기 때문에, 고유값 개수를 출력해 실제 범주형과 수치형을 구분

```
# 모두 수치형 변수이기 때문에, 실제론 범주형으로 간주되는 피처는 범주형으로 할당한다.

nu = df[features].nunique().sort_values()
nf = []; cf=[]; nnf=0; ncf=0 # numerical and categorical features

for i in range(df[features].shape[1]) :
  if nu.values[i] <= 16 : # 기준 : 고유한 값이 16개 이하인 경우는 범주형에 추가
    cf.append(nu.index[i]) # 범주형에 추가
  else :
    nf.append(nu.index[i]) # 수치형에 추가
```

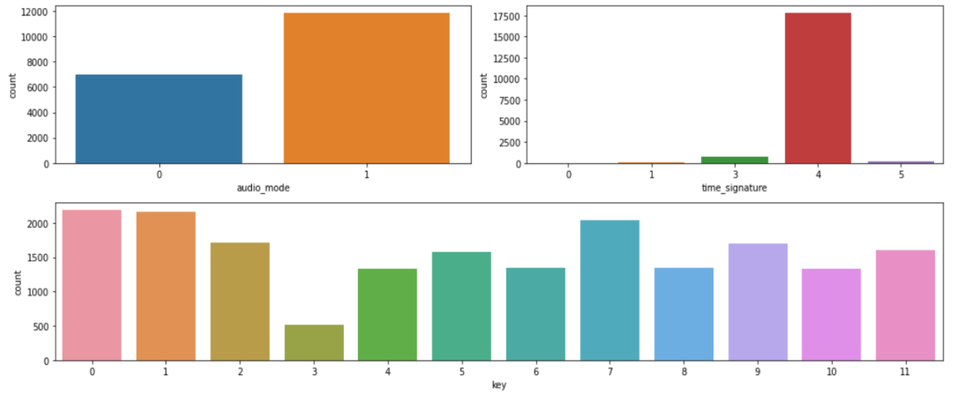
For 문, if 문을 사용해서 범주형/수치형 각각을 한번에 시각화 한 것이 특징

EDA

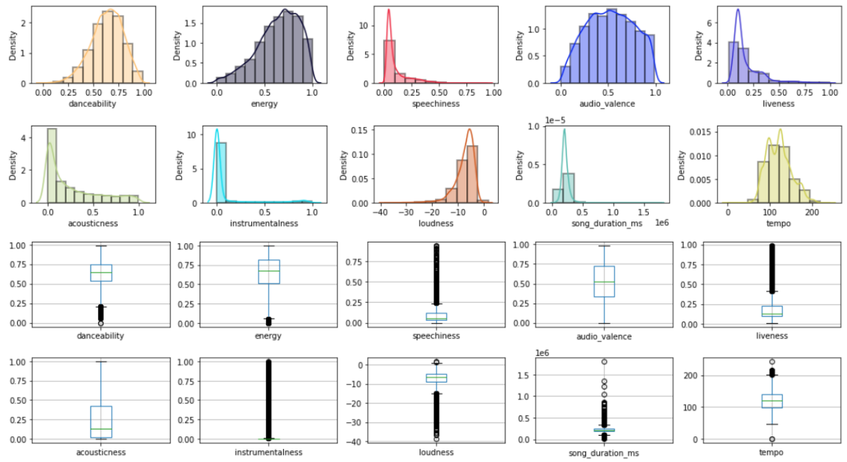
1. 종속변수 분포 시각화 : sns.distplot



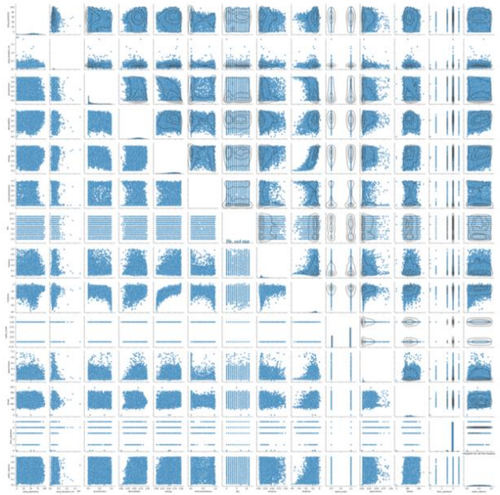
2. 범주형 변수 시각화 : sns.countplot



3. 수치형 변수 시각화 : sns.distplot, boxplot



4. 모든 변수 간 관계



- sns.pairplot(df)
- .map_upper()
- sns.kdeplot()

전처리

1. 중복 데이터 처리 : drop_duplicates 중복 요소를 삭제

```
# 중복되는 행 제거하기 Removal of any Duplicate rows (if any)
counter = 0
rs, cs = original_df.shape # 원 데이터의 행의 개수와 열의 개수를 rs, cs 에 저장

df.drop_duplicates(inplace=True) # drop_duplicates : 중복 데이터의 처리 (중복 요소를 삭제한다)

if df.shape==(rs,cs) : # 원 데이터셋의 행과 열의 개수가 일치하면
  print('중복되는 값이 없습니다')
else : # 중복 값이 삭제되어 차원에 변동이 있었다면
  print('중복 값이 있어 drop/fix 하였습니다 : ', rs - df.shape[0])
```

```
중복 값이 있어 drop/fix 하였습니다 :  3911
```

2. 널 값 확인 : df.isnull().sum()

```
# empty elements 확인

nvc = pd.DataFrame(df.isnull().sum().sort_values(), columns = ['total null values'])
nvc['Percentage'] = round(nvc['total null values']/df.shape[0], 3)*100
print(nvc) # 널값이 없다
```

	total null values	Percentage
song_popularity	0	0.0
song_duration_ms	0	0.0
acousticness	0	0.0
danceability	0	0.0
energy	0	0.0
instrumentalness	0	0.0
key	0	0.0
liveness	0	0.0
loudness	0	0.0
audio_mode	0	0.0
speechiness	0	0.0
tempo	0	0.0
time_signature	0	0.0
audio_valence	0	0.0

3. 원핫인코딩으로 범주형 수치화 : pd.get_dummies()

```
oh = True
dm = True

for i in fcc : # 'audio_mode', 'time_signature', 'key' 칼럼에 대해

  if df3[i].nunique() == 2 : # 범주가 2개면
    if oh == True : print('One-Hot Encoding on features:')
    print(i); oh=False

    df3[i] = pd.get_dummies(df3[i], drop_first=True, prefix = str(i))
    # drop_first : 첫번째 카테고리 값은 사용하지 않음 (용량 감소 효과)

  if (df[i].nunique()>2 and df3[i].nunique()<17) : # 범주가 많으면
    if dm == True : print('Dummy Encoding on features:')
    print(i);dm = False

    df3 = pd.concat([df3.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df3[i], drop_first=True, prefix=str(i)))], axis=1)
    # df3.drop([i], axis=1) : dummies 변환 전 칼럼은 삭제하고
    # pd.DataFrame(pd.get_dummies(df3[i], drop_first=True, prefix=str(i))) : dummies 한 칼럼이랑 concat 하기

print()
print(df3.shape)
print(df3.columns) # 각 칼럼에 대해
```

```
One-Hot Encoding on features:
audio_mode
Dummy Encoding on features:
time_signature
key
```

4. Outlier 제거

```
df1 = df3.copy() # 데이터의 원본은 보존하면서 복사

features1 = nf

for i in features1 :  # IQR based remove
  Q1 = df[i].quantile(0.25)
  Q3 = df1[i].quantile(0.75)
  IQR = Q3 - Q1
  df1 = df1[df1[i] <= (Q3 + (1.5*IQR))]
  df1 = df1[df1[i] >= (Q1 - (1.5*IQR))]
  df1 = df1.reset_index(drop = True)

print('before : ',df3.shape[0])
print('after : ',df1.shape[0])
```

```
before :  14924
after :  9119
```

Featrue engineering

1. Train/Test data split

```
X = df.drop([target], axis = 1)
Y = df[target]

Train_x, Test_x, Train_y, Test_y = train_test_split(X, Y, train_size = 0.8, test_size = 0.2, random_state=100)
Train_x.reset_index(drop=True, inplace = True)

print('Original set ---> ', X.shape, Y.shape, '₩n Training set --->', Train_x.shape, Train_y.shape,
      '₩n Testing set --->', Test_x.shape, Test_y.shape)
```

```
Original set --->  (9119, 26) (9119,)
 Training set ---> (7295, 26) (7295,)
 Testing set ---> (1824, 26) (1824,)
```

2. Feature Scaling : standardization *test set도!

```
std = StandardScaler()

Train_X_std = std.fit_transform(Train_x)
Train_X_std = pd.DataFrame(Train_X_std, columns = X.columns)
Train_X_std.describe() # 정규화가 잘 되었는지 확인
```

3. Correlation 확인

3. OLS

[OLS 모델]

- OLS model : 선형회귀 분석에 있어서 각각의 독립변수가 종속변수에 영향이 있는지 단적으로 확인할 수 있다.

```
from statsmodels.formula.api import ols
model = ols(formula = 'Y ~ X1 + X2 + ... + Xn',data = data).fit()
print(model.summary())
```

- 선형 회귀 분석의 가정 전제
  - 상관관계가 있는 독립변수는 제거
  - 변수들이 정규분포를 가진다는 가정 : 정규성을 가질 수 있도록 로그나 지수의 방법으로 치환
  - 독립변수와 종속변수는 선형 상관관계를 가지는 것을 가정

OLS 결과 해석

- [참고]
- **Dep.Variable** : 내가 예측하고자 한 Y 값 데이터
- **No.Observation** : 데이터 개수
- **Df Models** : 예측변수 개수 (전체 변수 개수 - 1)
- **R-squared** : 전체 변동 중에 n% 를 설명할 수 있다는 뜻으로 SSR/SST 값. 보통 **0.4 이상**이면 괜찮은 모델이라 할 수 있다.
- **P>|t| (유의확률)** : 일반적으로 **0.05보다 값이 작으면** 독립변수가 종속변수에 영향을 미치는 것이 유의하다고 보면 된다.
- **Durbin-Watson** : 잔차의 독립성을 확인할 수 있는 수치이다. 보통 1.5에서 2.5 사이이면 독립으로 판단하고 회귀모형이 적합하다고 해석할 수 있다. 0이나 4에 가깝다면 잔차들이 자기상관을 가지고 있다는 의미로, R-squared 를 증가시켜 유의미하지 않은 결과를 유의미한 결과로 왜곡할 수 있다는 의미이다.

Featrue engineering

```
# statsmodels api : ols
API = api.ols(formula = '{} ~ {}'.format( target, '+'.join(i for i in Train_x.columns)), data = Train_xy).fit()
# '구분자'.join(리스트) : 리스트의 값과 값 사이에 구분자를 넣어서 하나의 문자열로 합쳐줌

API.summary()

## R-squared 값이 0.035로, 모델의 성능이 그다지 좋지 않음을 알 수 있다
## Durbin-Watson 값이 1.96으로 잔차의 독립성이 보장됨을 확인할 수 있다.
```

OLS Regression Results

Dep. Variable:	song_popularity	**R-squared:**	0.035
Model:	OLS	**Adj. R-squared:**	0.031
Method:	Least Squares	**F-statistic:**	10.47
Date:	Thu, 10 Feb 2022	**Prob (F-statistic):**	1.74e-40
Time:	23:30:05	**Log-Likelihood:**	-32315.
No. Observations:	7295	**AIC:**	6.468e+04
Df Residuals:	7269	**BIC:**	6.486e+04
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	50.2624	0.238	211.063	0.000	49.796	50.729
song_duration_ms	0.7494	0.244	3.076	0.002	0.272	1.227
acousticness	-1.3805	0.336	-4.111	0.000	-2.039	-0.722
danceability	1.4772	0.286	5.166	0.000	0.917	2.038
energy	-2.4762	0.459	-5.395	0.000	-3.376	-1.576
instrumentalness	-0.0713	0.241	-0.296	0.767	-0.544	0.401
liveness	-0.9090	0.243	-3.741	0.000	-1.385	-0.433
loudness	3.0909	0.365	8.479	0.000	2.376	3.805
audio_mode	0.5618	0.251	2.240	0.025	0.070	1.053
speechiness	-0.2144	0.256	-0.839	0.401	-0.715	0.286
tempo	-0.1454	0.247	-0.588	0.557	-0.630	0.339
audio_valence	-1.3655	0.290	-4.704	0.000	-1.935	-0.796
time_signature_1	0.0075	0.233	0.032	0.974	-0.450	0.465
time_signature_3	-0.0291	0.157	-0.185	0.853	-0.337	0.278
time_signature_4	0.0423	0.125	0.337	0.736	-0.204	0.288
time_signature_5	-0.0442	0.223	-0.198	0.843	-0.481	0.393
key_1	0.6347	0.314	2.024	0.043	0.020	1.249
key_2	-0.2101	0.306	-0.688	0.492	-0.809	0.389
key_3	0.0946	0.262	0.361	0.718	-0.419	0.609
key_4	0.1609	0.295	0.545	0.586	-0.418	0.739
key_5	-0.1113	0.303	-0.367	0.713	-0.705	0.483
key_6	0.6006	0.293	2.049	0.040	0.026	1.175
key_7	-0.2443	0.316	-0.773	0.440	-0.864	0.375
key_8	0.1708	0.296	0.578	0.564	-0.409	0.750
key_9	-0.2283	0.308	-0.741	0.459	-0.832	0.376
key_10	0.3033	0.289	1.049	0.294	-0.264	0.870
key_11	0.4106	0.304	1.351	0.177	-0.185	1.006

Omnibus:	472.769	**Durbin-Watson:**	1.964
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	570.368
Skew:	-0.685	**Prob(JB):**	1.40e-124
Kurtosis:	2.972	**Cond. No.**	6.02e+15

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 5.64e-28. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

다중공선성을 없애서 모델 설명력을 높여볼까

Featrue engineering

4. Feature selection/extraction

4-1. 다중공선성 해결

1) VIF

- 상관계수가 커질수록 R^2 은 커진다.
- 가장 상호의존적인 독립변수를 제거한다. 의존성이 낮은 독립변수를 선택하거나, 의존성이 높은 독립변수를 제거하며 사용
- `variance_inflation_factor(X, i)` : Xi를 x나머지로 회귀분석한 후 VIF값을 구한것. 즉 xi의 vif값. 즉 이값이 높을수록 종속성이 높다는 뜻

```
Drop = []; b=[]


# VIF 값을 저장할 표를 데이터프레임 형태로 만들기

for i in range(len(Train_X_std.columns)) : # 각 열에 대해

  vif = pd.DataFrame()
  X = Train_X_std.drop(Drop, axis=1) # VIF 에 근거하여 칼럼을 제거해 가며 성능 확인
  vif['Features'] = X.columns
  vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

  # 각 칼럼별로 VIF 를 계산한다.
  # variance_inflation_factor(data,all_col_index)

  vif['VIF'] = round(vif['VIF'],2)
  vif = vif.sort_values(by='VIF', ascending = False)
  vif.reset_index(drop=True, inplace=True)

  if vif.loc[0][1] > 1 : # VIF 값이 1보다 크면

    Drop.append(vif.loc[0][0]) # 해당 피처를 drop 시키기 위해 Drop 변수에 저장한다.

    # VIF 를 바탕으로 drop할 칼럼을 제거한 후, 회귀분석 진행
    LR = LinearRegression()
    LR.fit(Train_X_std.drop(Drop,axis=1), Train_y)

    pred1 = LR.predict(Train_X_std.drop(Drop,axis=1))
    pred2 = LR.predict(Test_X_std.drop(Drop,axis=1))

    Trr.append(np.sqrt(mean_squared_error(Train_y, pred1))) # 훈련 set 에 대한 error
    Tss.append(np.sqrt(mean_squared_error(Test_y, pred2))) # test set 에 대한 error


print('drop 한 피처 --> ', Drop)
```

	Features	VIF
0	song_duration_ms	1.0
1	instrumentalness	1.0
2	liveness	1.0
3	tempo	1.0
4	time_signature_1	1.0
5	time_signature_3	1.0
6	time_signature_5	1.0
7	key_10	1.0

살아남은 피처

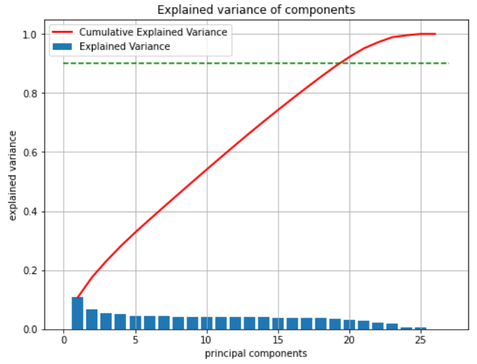
2) RFE

- 모든 feature 들로부터 피처를 하나하나 제거해가면서 원하는 개수의 피처가 남을 때까지 이를 반복한다.
- feature importance 를 도출하는데, 변수 중요도가 낮은 피처부터 하나씩 제거해가면서 원하는 피처 개수가 될 때까지 반복한다. 상위 피처 중요도를 가지는 피처들이 최종 결과가 된다.
- 단점 : 몇 개의 feature 를 남길지 사용자가 직접 정의 ---> RFECV 방법 등장

```
m = df.shape[1]-2  # m = 25

for i in range(m) :
  lm = LinearRegression()
  rfe = RFE(lm, n_features_to_select = Train_X_std.shape[1]-i) # i 개수만큼 피처 제외시켜 살펴보기
  # n_features_to_select : The number of features to select (선택할 피처의 개수)
  rfe = rfe.fit(Train_X_std, Train_y)

  LR = LinearRegression()
  LR.fit(Train_X_std.loc[:, rfe.support_], Train_y)
  # rfe.support_ : 선택된 피처들 리스트

  pred1 = LR.predict(Train_X_std.loc[:, rfe.support_])
  pred2 = LR.predict(Test_X_std.loc[:, rfe.support_])

  Trr.append(np.sqrt(mean_squared_error(Train_y, pred1)))
  Tss.append(np.sqrt(mean_squared_error(Test_y, pred2)))
```

3) PCA

- PCA : 예를 들어 데이터에 5개의 수치형 변수가 있을 때, 전체 데이터의 정보(변동)의 손실을 최소화하면서 2개 혹은 3개의 새로운 수치형 변수로 만드는 기법
- 필요성 : PCA는 이상치를 찾을 때, 데이터 축약이 필요한 경우, 다중공선성이 발생할 때, 많이 사용된다.
- 특징 : PCA 전에 변수를 표준화나 정규화 시켜주어야 한다!! (스케일에 의해 분산량이 왜곡됨)
- **주성분** : 독립변수들의 분산을 가장 잘 설명하는 성분으로, 전체 데이터의 분산을 가장 잘 설명하는 축의 개수를 선정해서 그 축에 따라 변형된 데이터를 배열하면 그 데이터가 주성분이 된다. ---> 주성분은 원래의 데이터와 다르다.
- 중요함의 기준 : 전체 데이터의 변산을 얼마나 잘 설명하냐

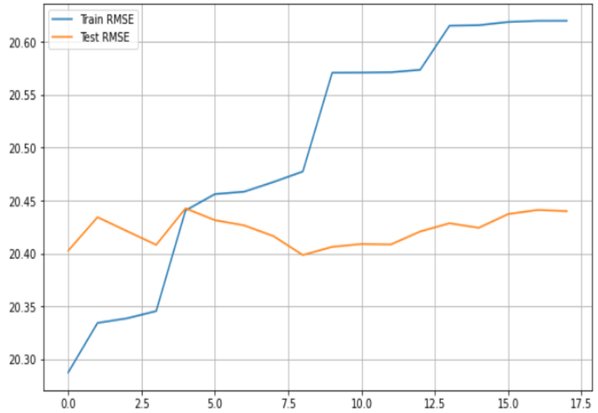


Featrue engineering

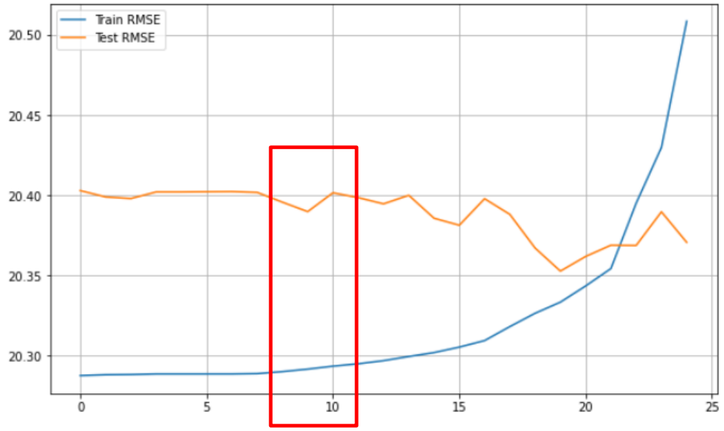
4. Feature selection/extraction

4-1. 다중공선성 해결

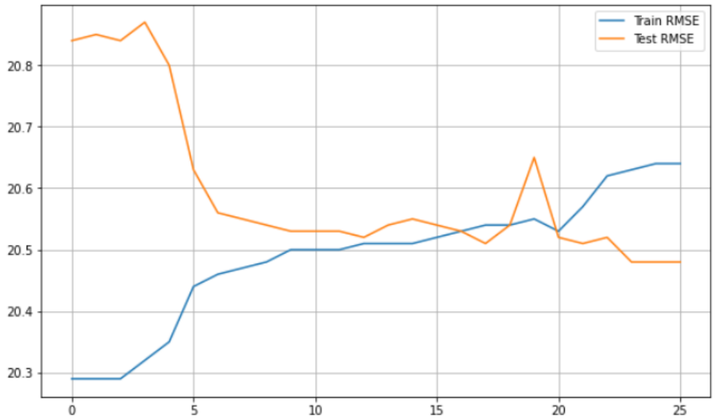
1) VIF



2) RFE



3) PCA



** Train, test set RMSE 보는 방법 더 공부할 것

RFE 가 최적의 방법이라 판단하여 9개 변수를 선택

modeling

모델 평가 비교를 위한 데이터프레임 만들기 : Model_Evaluation_Comparison_Matrix

```
# 모델을 비교하기 위해 평가지표 행렬 만들기
Model_Evaluation_Comparison_Matrix = pd.DataFrame(np.zeros([5,8]), columns = ['Train-R2','Test-R2','Train-RSS','Test-RSS',
                                                                              'Train-MSE','Test-MSE','Train-RMSE','Test-RMSE'])
```

```
#Evaluating the Multiple Linear Regression Model

# Train set
print('₩nR2-Score on Training set --->',round(r2_score(Train_y, pred1),20)) # R-squared 모델 설명력
print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Train_y-pred1)),20))
print('Mean Squared Error (MSE) on Training set      --->',round(mean_squared_error(Train_y, pred1),20))
print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Train_y, pred1)),20))

# Test set
print('₩nR2-Score on Testing set --->',round(r2_score(Test_y, pred2),20))
print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Test_y-pred2)),20))
print('Mean Squared Error (MSE) on Training set      --->',round(mean_squared_error(Test_y, pred2),20))
print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Test_y, pred2)),20))
print('₩n{}Residual Plots{}'.format('-'*20, '-'*20))
```

```
plt.subplot(1,2,1)
sns.distplot((Train_y - pred1)) # error 에 대한 분포 시각화
plt.title('Error Terms')
plt.xlabel('Errors')
```

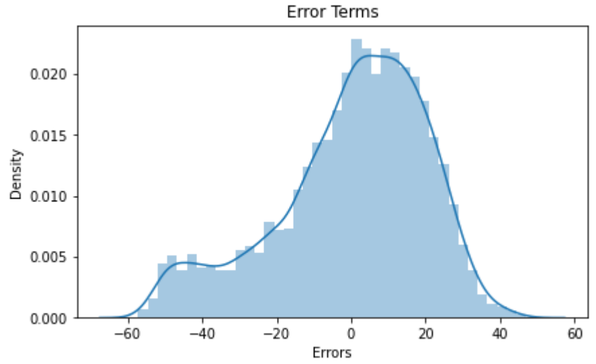
```
plt.subplot(1,2,2)
plt.scatter(Train_y,pred1) # 실제값과 예측값이 비슷하다면 y=x 직선 분포와 비슷할 것임
plt.plot([Train_y.min(),Train_y.max()],[Train_y.min(),Train_y.max()], 'r--') # y=x 직선 그래프
plt.title('Test vs Prediction')
plt.xlabel('y_test')
plt.ylabel('y_pred')
plt.show()
```

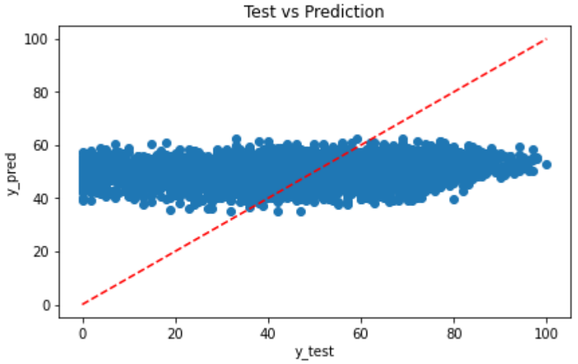
modeling

MLR

```
MLR = LinearRegression().fit(Train_X_std, Train_y)
pred1 = MLR.predict(Train_X_std)
pred2 = MLR.predict(Test_X_std)

print('회귀계수', MLR.coef_)
print('₩n 상수항', MLR.intercept_)

Evaluate(0, pred1, pred2)
```

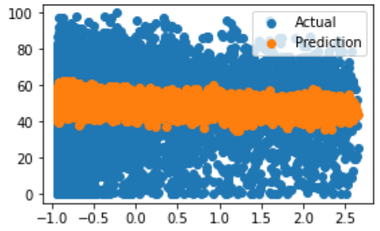
```
R2-Score on Training set ---> 0.03475579056066336
Residual Sum of Squares (RSS) on Training set  ---> 3007206.5787732247
Mean Squared Error (MSE) on Training set       ---> 412.2284549380706
Root Mean Squared Error (RMSE) on Training set ---> 20.303409933754246

R2-Score on Testing set ---> 0.025791628173641893
Residual Sum of Squares (RSS) on Training set  ---> 732127.6902936784
Mean Squared Error (MSE) on Training set       ---> 401.3857951171483
Root Mean Squared Error (RMSE) on Training set ---> 20.034614923106165

--------------------Residual Plots--------------------
```

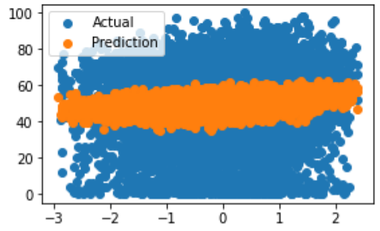


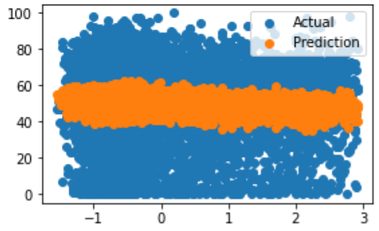



```
회귀계수 [ 7.48783278e-01 -1.38499935e+00  1.47652422e+00 -2.47663094e+00
 -6.83501856e-02 -9.04515285e-01  3.09086120e+00  5.56146648e-01
 -2.16573025e-01 -1.47756802e-01 -1.36479800e+00 -6.95698375e+12
 -2.58678259e+13 -2.90687222e+13 -1.20150111e+13  6.33961077e-01
 -2.04435218e-01  1.00103805e-01  1.50705545e-01 -1.11384070e-01
  5.90511282e-01 -2.40412015e-01  1.63228875e-01 -2.23269281e-01
  3.10085243e-01  4.13478098e-01]
```

상수항 50.25595326633099





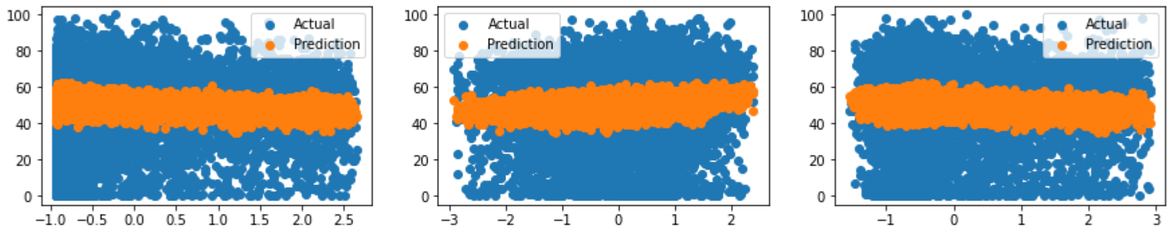


modeling

Ridge

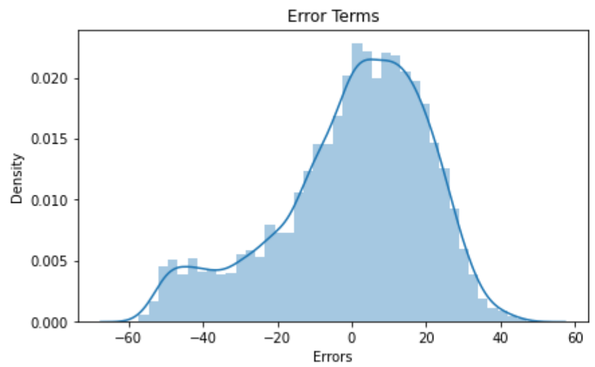
```
RLR = Ridge().fit(Train_X_std, Train_y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('회귀계수', RLR.coef_)
print('₩n 상수항', RLR.intercept_)

Evaluate(1, pred1, pred2)
```

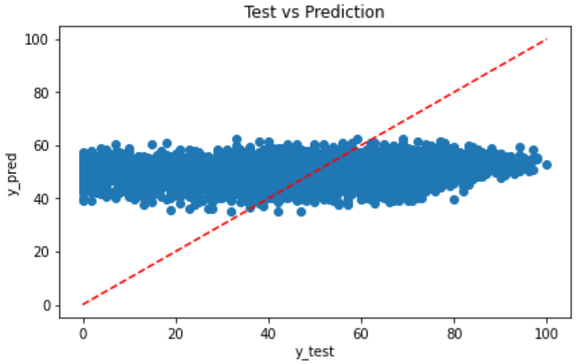
```
회귀계수 [ 0.74930313 -1.37982769  1.47717432 -2.47429018 -0.07141842 -0.9089507
  3.0894157   0.56162935 -0.21438621 -0.14550408 -1.36573571  0.00752128
 -0.02905384  0.04229523 -0.04413097  0.63456799 -0.21027667  0.09454166
  0.16068693 -0.11141131  0.60035197 -0.24441754  0.17062089 -0.22848921
  0.3031653   0.41043601]

상수항 50.26237148732008
```




```
R2-Score on Training set ---> 0.03475730802630761
Residual Sum of Squares (RSS) on Training set  ---> 3007201.8511274913
Mean Squared Error (MSE) on Training set       ---> 412.2278068714861
Root Mean Squared Error (RMSE) on Training set ---> 20.303393974197668

R2-Score on Testing set ---> 0.02577389498550553
Residual Sum of Squares (RSS) on Training set  ---> 732141.0169683892
Mean Squared Error (MSE) on Training set       ---> 401.3931014081081
Root Mean Squared Error (RMSE) on Training set ---> 20.034797263963217
```

----------Residual Plots----------



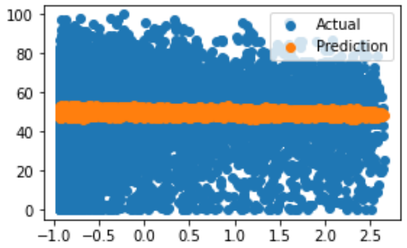


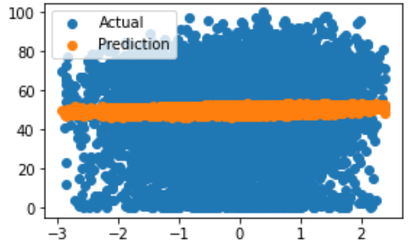
modeling

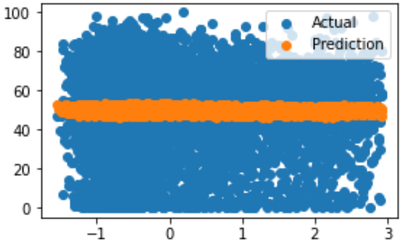
Lasso

```
LLR = Lasso().fit(Train_X_std,Train_y)
pred1 = LLR.predict(Train_X_std)
pred2 = LLR.predict(Test_X_std)

print('회귀계수', LLR.coef_)
print('₩n 상수항', LLR.intercept_)

Evaluate(2, pred1, pred2)
```

```
회귀계수 [ 0.         -0.          0.5772552  -0.         -0.         -0.15434989
  0.96152164  0.          0.         -0.         -0.6383851   0.
 -0.          0.          0.          0.         -0.         -0.
 -0.         -0.          0.         -0.          0.         -0.
  0.          0.        ]

상수항 50.26237148732008
```

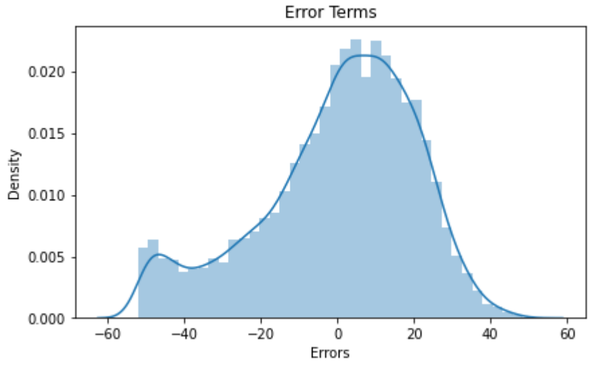


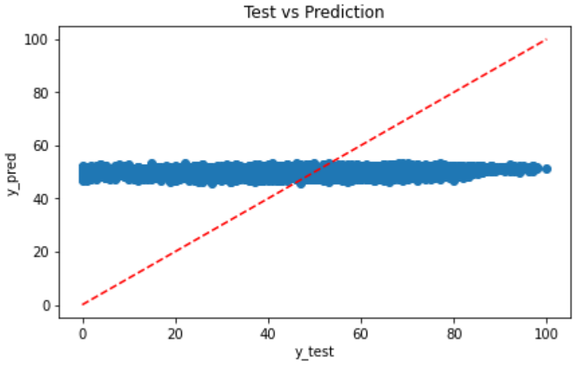





```
R2-Score on Training set ---> 0.01422303719096951 4
Residual Sum of Squares (RSS) on Training set  ---> 3071176.121827554
Mean Squared Error (MSE) on Training set       ---> 420.9974121764982
Root Mean Squared Error (RMSE) on Training set ---> 20.51822146718614

R2-Score on Testing set ---> 0.011420810698448 58
Residual Sum of Squares (RSS) on Training set  ---> 742927.5085974577
Mean Squared Error (MSE) on Training set       ---> 407.306748134571
Root Mean Squared Error (RMSE) on Training set ---> 20.181842040174903
```

```
----------Residual Plots----------
```





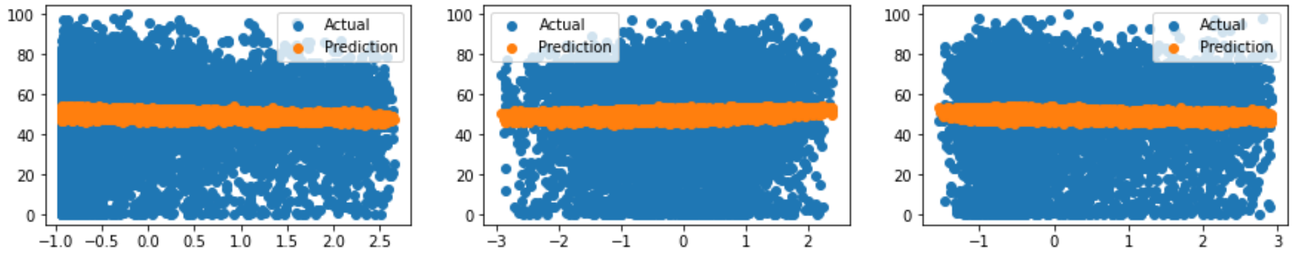
modeling

Elasticnet

```
ENR = ElasticNet().fit(Train_X_std,Train_y)
pred1 = ENR.predict(Train_X_std)
pred2 = ENR.predict(Test_X_std)

print('회귀계수', ENR.coef_)
print('₩n 상수항', ENR.intercept_)

Evaluate(3, pred1, pred2)
```

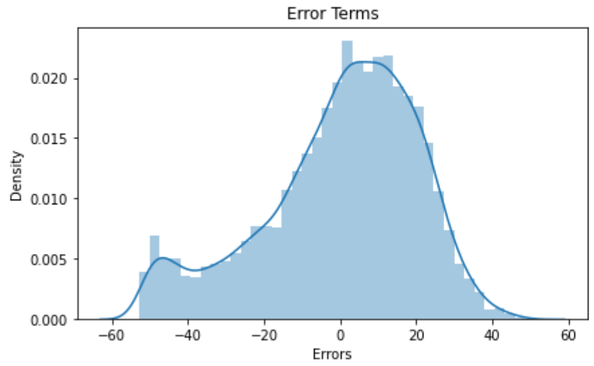
```
회귀계수 [ 0.19090574 -0.2887613   0.70993239 -0.          -0.          -0.4287789
  0.88011308  0.          0.          -0.         -0.77270402  0.
 -0.          0.          0.          0.21288498 -0.          -0.
 -0.          -0.         0.02309262 -0.          0.          -0.
  0.          0.        ]
```

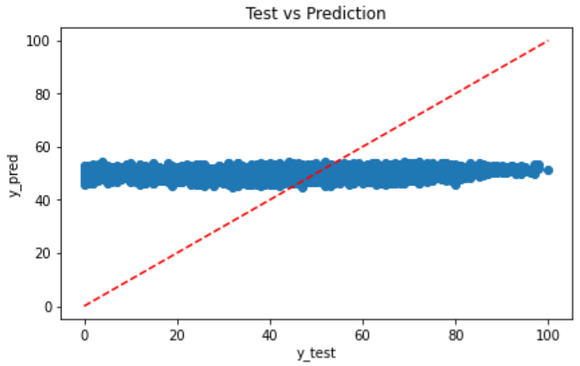
상수항 50.26237148732008




```
R2-Score on Training set ---> 0.018595304808019852
Residual Sum of Squares (RSS) on Training set  ---> 3057554.3753165975
Mean Squared Error (MSE) on Training set       ---> 419.130140550596
Root Mean Squared Error (RMSE) on Training set ---> 20.47266813462759

R2-Score on Testing set ---> 0.013547867338772114
Residual Sum of Squares (RSS) on Training set  ---> 741329.0034827003
Mean Squared Error (MSE) on Training set       ---> 406.4303747163928
Root Mean Squared Error (RMSE) on Training set ---> 20.160118420197655
```

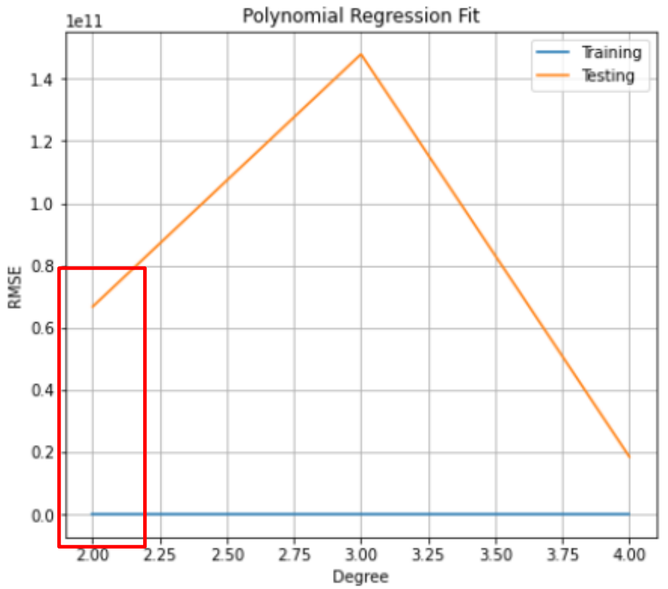
----------Residual Plots----------





modeling

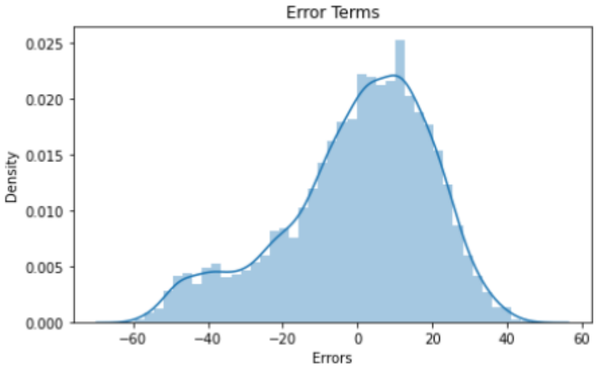
Polynomial

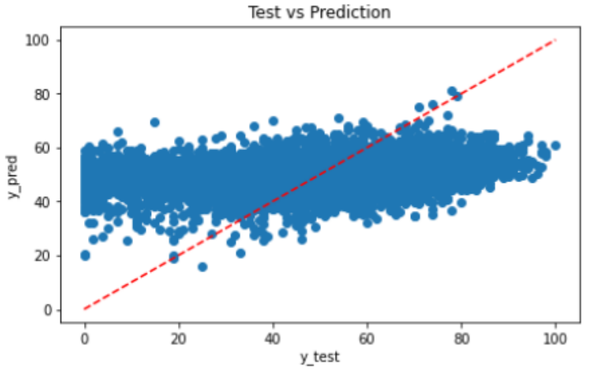


2차 다항회귀로 결정!

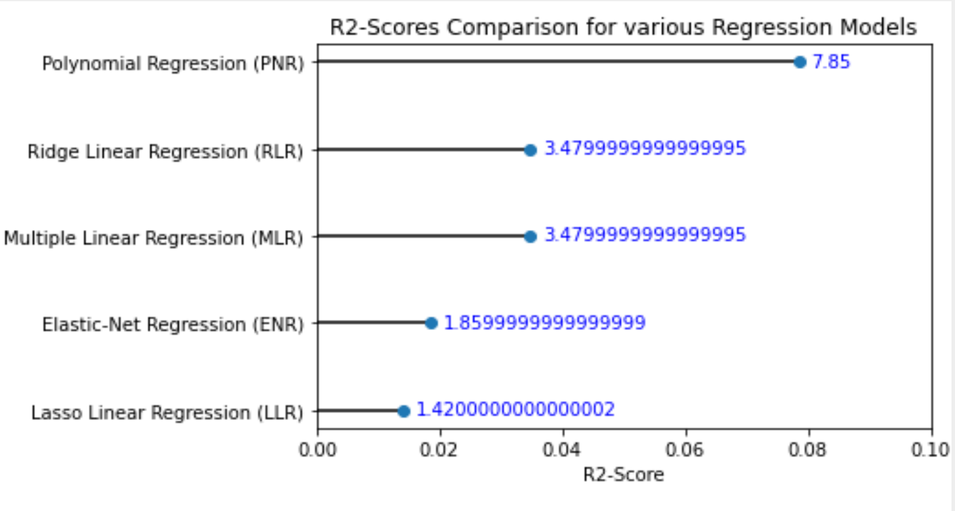
```
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(Train_X_std)
X_poly1 = poly_reg.fit_transform(Test_X_std)
PR = LinearRegression()
PR.fit(X_poly, Train_Y)

pred1 = PR.predict(X_poly)
pred2 = PR.predict(X_poly1)
```

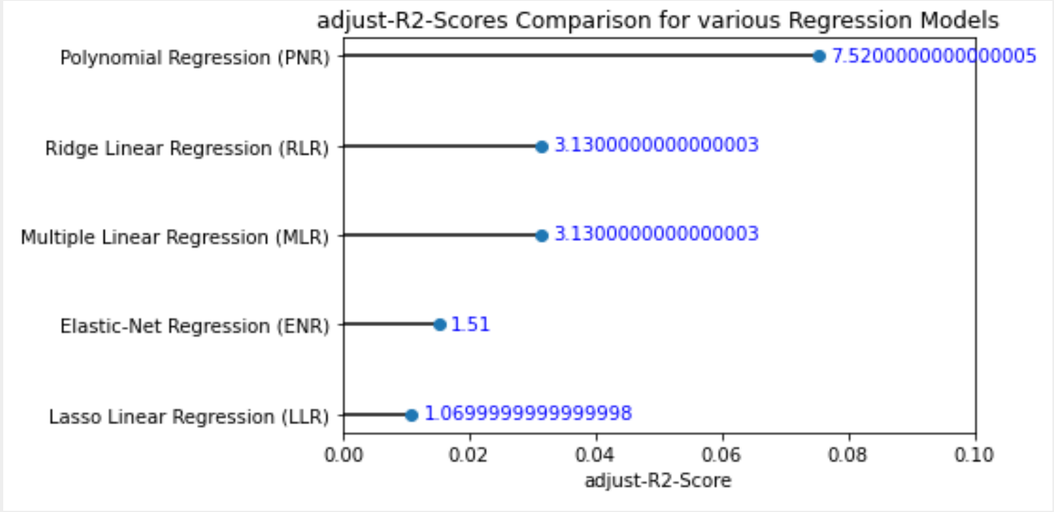
```
R2-Score on Training set ---> 0.07763927257673142
Residual Sum of Squares (RSS) on Training set  ---> 2823112.234224039
Mean Squared Error (MSE) on Training set       ---> 394.289418187715
Root Mean Squared Error (RMSE) on Training set ---> 19.85672224179295
```



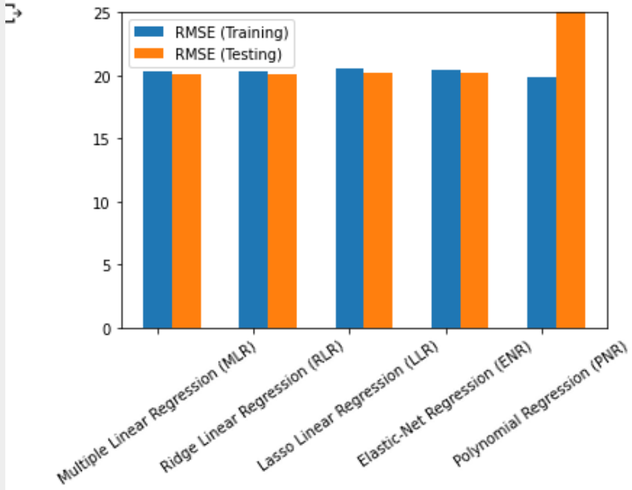


comparing






```
# RMSE 값이 작을수록, training & Testing score 이 근접한 값이여야 좋은 모델 !
# Polynomial regression -> 오버피팅

## 여기선 MLR 모델이 가장 좋은 모델인 것처럼 보인다.
```



EDA, 상관관계 분석에서 피처에 대한 인사이트 도출

다중공선성 발견, feature extraction 방법으로 해결

다양한 회귀 알고리즘 적용, 최적의 모형 선택하기

분석을 위한 함수/프레임 미리 만들어 놓는 것도 좋음!

01

Drop_duplicates
중복행 선정
기준

02

변수선택 RMSE
시각화 결과
해석

03

모델링 결과
해석

04

앙상블 회귀
모형으로도
돌려보기