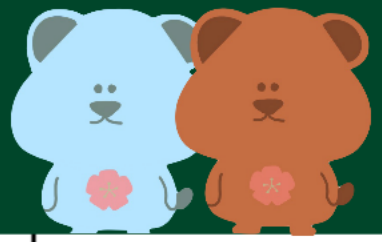
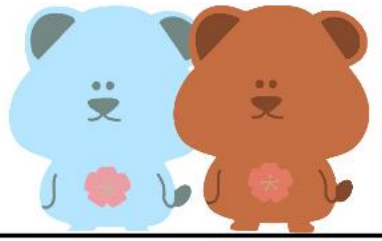


6주차 발표

DA팀 김예진 박보영 이의진

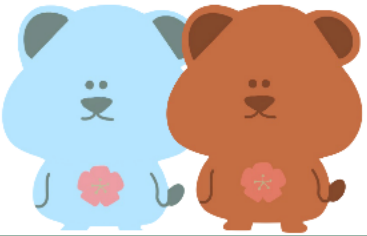
EWHA EURON

목차



EWHA EURON

1. 캐글 주택 가격



EWHA EURON

1.1 RMSLE

RMSLE(Root Mean Square Log Error)

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(\hat{y}_i + 1) - log(y_i + 1))^2}$$

log1p

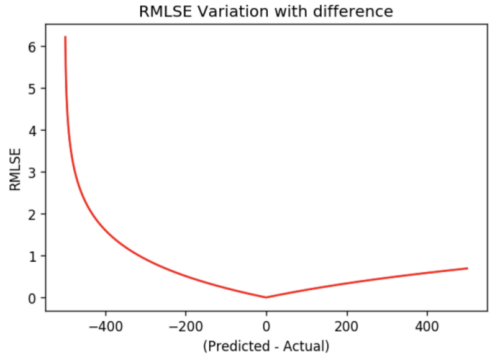
$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

RMSLE특징

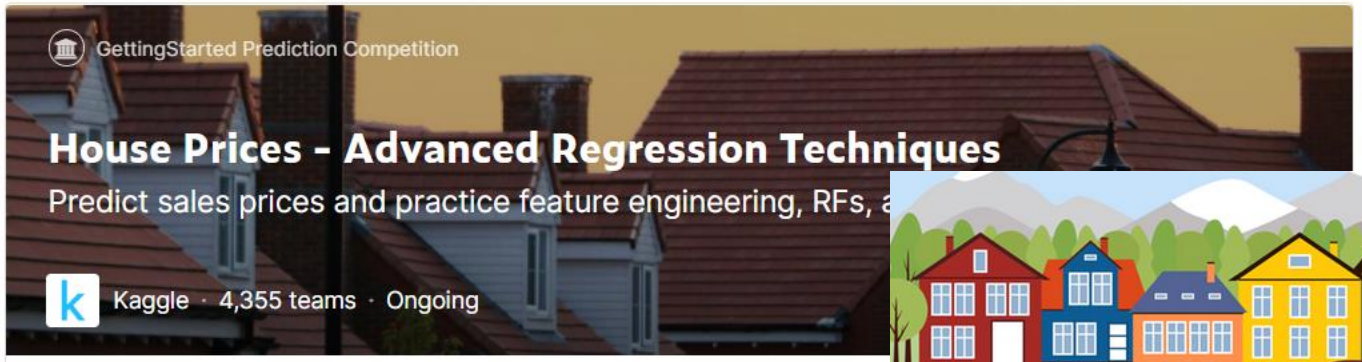
1. 아웃라이어에 강건하다 (robust) : 아웃라이어가 있더라도 값의 변동폭이 크지 않다
2. 상대적 Error를 측정한다 : RMSLE는 상대적 크기가 동일하다면 RMSLE의 값도 동일하다

$$\log(\hat{y} + 1) - \log(y + 1) = \frac{\log((\hat{y}+1)}{\log(y+1)}$$

3. Under Estimation에 큰 패널티를 부여한다 : 예측값이 실제보다 클때(과소평가)



1.2 캐글 주택 가격:고급 회귀 기법



<목적>
아이오와주 에임스(Ames, Iowa)에 있는 주거용 주택의 (거의) 모든 측면을 설명하는 79개의 변수가 있는 데이터세트는 각 주택의 최종 가격을 예측하는 데 도전한다

<칼럼>
Target은 SalePrice
Feature는 Id(단순식별자), YearBulit(건축년도), GrLivArea(주거공간크기),…

1.3 데이터 사전 처리

1. Null 있는 칼럼은?

```
print('데이터 세트의 shape: ', house_df.shape)
print('₩n 전체 피처의 type₩n', house_df.dtypes.value_counts())
isnull_series=house_df.isnull().sum()
print('₩n null 칼럼과 그 건수:₩n', isnull_series[isnull_series>0].sort_values(ascending=False))
```

```
데이터 세트의 shape:  (1460, 81)

 전체 피처의 type
 object     43
int64       35
float64      3
dtype: int64

 null 칼럼과 그 건수:
 PoolQC          1453
MiscFeature      1406
Alley            1369
Fence            1179
FireplaceQu       690
LotFrontage       259
GarageType         81
GarageYrBlt        81
GarageFinish       81
GarageQual         81
GarageCond         81
BsmtExposure       38
BsmtFinType2       38
BsmtFinType1       37
BsmtCond           37
BsmtQual           37
MasVnrArea          8
MasVnrType          8
Electrical          1
dtype: int64
```

2. 타겟 정규 분포인가?

```
plt.title('Original Sale Price Histogram')
sns.distplot(house_df['SalePrice'])
```

```
<AxesSubplot:title={'center':'Original Sale Price Histo
```

3. 무의미한 칼럼은?

4. 문자형 피처?

1.3 데이터 사전 처리

1. Null 있는 칼럼은? -> 평균값 대체 또는 삭제

2. 타겟 정규 분포인가? -> 로그변환

3. 무의미한 칼럼은? -> 삭제

4. 문자형 피처? -> get_dummies() 원핫인코딩

```
#SalePrice 로그 변환
original_SalePrice=house_df['SalePrice']
house_df['SalePrice']=np.log1p(house_df['SalePrice'])

#Null이 너무 많은 칼럼과 불필요한 칼럼 삭제
house_df.drop(['Id','PoolQC', 'MiscFeature','Alley','Fence','FireplaceQu'],axis=1)

#삭제하지 않은 숫자형 Null 칼럼은 평균값으로 대체
house_df.fillna(house_df.mean(),inplace=True)

#Null 값이 있는 피처명과 타입을 추출
null_column_count=house_df.isnull().sum()[house_df.isnull().sum()>0]
print('##null피처의 Type:₩n', house_df.dtypes[null_column_count.index])
```

```
##null피처의 Type:
 Alley           object
MasVnrType       object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinType2     object
Electrical       object
FireplaceQu      object
GarageType       object
GarageFinish     object
GarageQual       object
GarageCond       object
PoolQC           object
Fence            object
MiscFeature      object
dtype: object
```

```
print('get_dummies() 수행 전 데이터shape: ',house_df.shape)
house_df_ohe=pd.get_dummies(house_df)
print('get_dummies() 수행 후 데이터shape: ',house_df_ohe.shape)

null_column_count=house_df_ohe.isnull().sum()[house_df_ohe.isnull().sum()>0]
print('##null 피처의 Type:₩n',house_df.dtypes[null_column_count.index])
```

```
get_dummies() 수행 전 데이터shape:  (1460, 81)
get_dummies() 수행 후 데이터shape:  (1460, 290)
##null 피처의 Type:
 Series([], dtype: object)
```

EWHA EURON

1.4 모델 학습/예측/평가

```
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

y_target=house_df_ohe['SalePrice']
X_features=house_df_ohe.drop('SalePrice',axis=1, inplace=False)
X_train, X_test, y_train, y_test=train_test_split(X_features,y_target,test_size=0.2, random_state=156)

#baseline
from sklearn.dummy import DummyRegressor

dummy_reg = DummyRegressor(strategy='mean')
dummy_reg.fit(X_train,y_train)
pred=dummy_reg.predict(X_test)
mse=mean_squared_error(y_test, pred)
rmse=np.sqrt(mse)
```

Baseline model은 DummyRegressor

```
#선형회귀, 릿찌, 라쏘
lr_reg=LinearRegression()
lr_reg.fit(X_train,y_train)
ridge_reg=Ridge()
ridge_reg.fit(X_train,y_train)
lasso_reg=Lasso()
lasso_reg.fit(X_train, y_train)

models=[dummy_reg, lr_reg, ridge_reg, lasso_reg]
get_rmses(models)
```

```
DummyRegressor 로그 변환된 RMSE: 0.408
LinearRegression 로그 변환된 RMSE: 0.133
Ridge 로그 변환된 RMSE: 0.13
Lasso 로그 변환된 RMSE: 0.177
```

```
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

y_target=house_df_ohe['SalePrice']
X_features=house_df_ohe.drop('SalePrice',axis=1, inplace=False)
X_train, X_test, y_train, y_test=train_test_split(X_features,y_target,test_size=0.2, random_state=156)

#baseline
from sklearn.dummy import DummyRegressor

dummy_reg = DummyRegressor(strategy='quantile', quantile=0.0)
dummy_reg.fit(X_train,y_train)
pred=dummy_reg.predict(X_test)
mse=mean_squared_error(y_test, pred)
rmse=np.sqrt(mse)
print("baseline: ", rmse)

#선형회귀, 릿찌, 라쏘
lr_reg=LinearRegression()
lr_reg.fit(X_train,y_train)
ridge_reg=Ridge()
ridge_reg.fit(X_train,y_train)
lasso_reg=Lasso()
lasso_reg.fit(X_train, y_train)

models=[dummy_reg, lr_reg, ridge_reg, lasso_reg]
get_rmses(models)
```

```
baseline:  1.60766947047605
DummyRegressor 로그 변환된 RMSE: 1.608
LinearRegression 로그 변환된 RMSE: 0.133
Ridge 로그 변환된 RMSE: 0.13
Lasso 로그 변환된 RMSE: 0.177
```

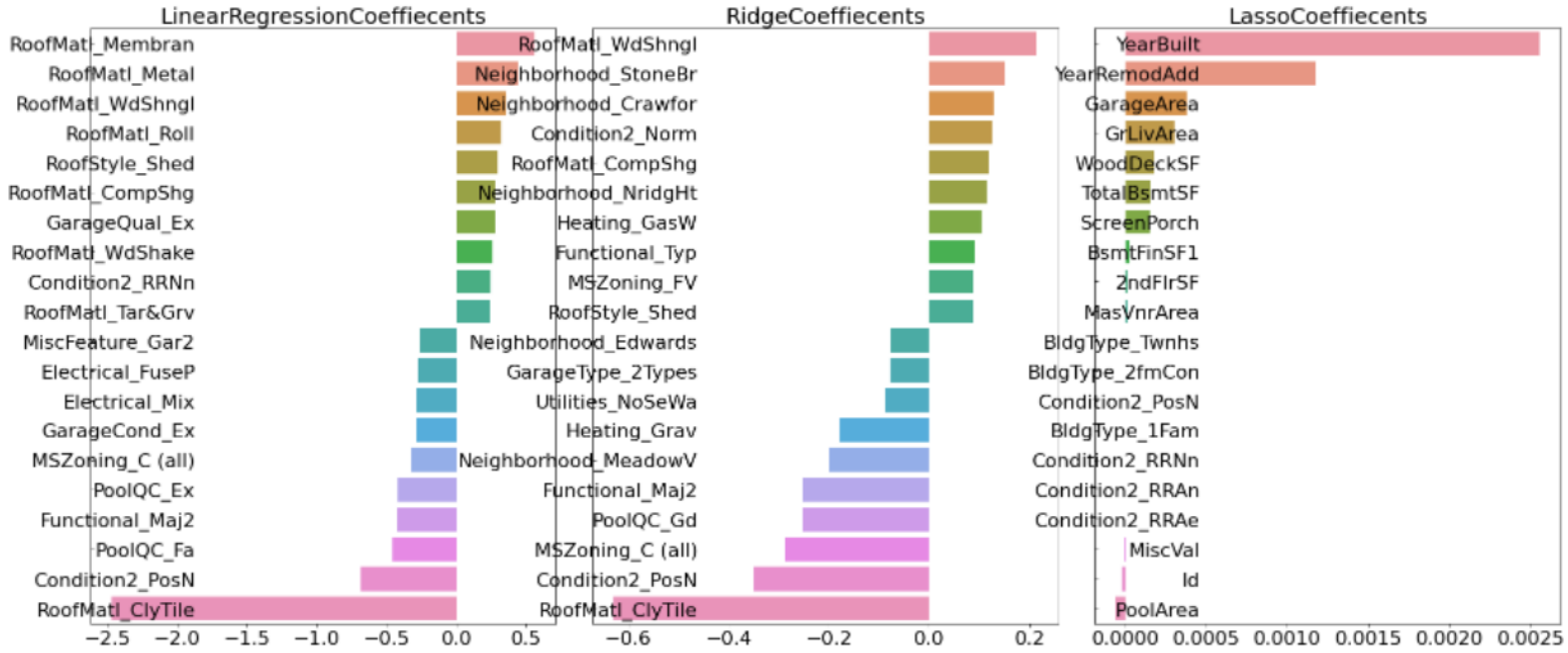
<결과>
라쏘의 비용함수가 높다= 회귀 성능이 좋지 않다

EWHA EURON

1.5 라쏘와 튜닝

<라쏘의 문제>
1. 낮은 회귀 성능
2. 작은 회귀 계수

<TODO>
1. 데이터 분할에 문제?
2. Alpha 하이퍼파라미터 튜닝?
3. 데이터 분포도(왜곡)?
4. 이상치 데이터?



회귀 계수가 클수록, 피처 중요도가 크다

1.5 라쏘와 튜닝

<TODO>

1. 데이터 분할에 문제? -> (cross_val_score) 분할하지 않고, 전체 데이터세트인 X_features, y_target을 5교차검증폴드세트로 분할해 RMSE측정
2. Alpha 하이퍼파라미터 튜닝?
3. 데이터 분포도(왜곡)?
4. 이상치 데이터?

```
from sklearn.model_selection import cross_val_score

def get_avg_rmse_cv(models):
    for model in models:
        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 값 리스트: {1}'.format(model.__class__.__name__, np.round(rmse_avg,3)))

models=[dummy_reg, lr_reg, ridge_reg, lasso_reg]
get_avg_rmse_cv(models)
```

```
DummyRegressor cv rmse 값 리스트: [0.387 0.425 0.41  0.383 0.391]
DummyRegressor cv 평균 값 리스트: 0.399

LinearRegression cv rmse 값 리스트: [0.14  0.167 0.169 0.111 0.2  ]
LinearRegression cv 평균 값 리스트: 0.157

Ridge cv rmse 값 리스트: [0.121 0.154 0.143 0.117 0.189]
Ridge cv 평균 값 리스트: 0.145

Lasso cv rmse 값 리스트: [0.161 0.204 0.177 0.181 0.265]
Lasso cv 평균 값 리스트: 0.198
```

1.5 라쏘와 튜닝

<TODO>

1. 데이터 분할에 문제? (X)
2. Alpha 하이퍼파라미터 튜닝?-> (GridSearchCV) alpha값 최적화
3. 데이터 분포도(왜곡)?
4. 이상치 데이터?

```
from sklearn.model_selection import GridSearchCV

def print_best_params(model, params):
    grid_model=GridSearchCV(model, param_grid=params, scoring='neg_mean_squared_error', cv=5)
    grid_model.fit(X_features,y_target)
    rmse=np.sqrt(-1*grid_model.best_score_)
    print('{0} 5 cv 최적평균 rmse 값{1}, 최적 alpha{2}'.format(model.__class__.__name__, np.round(rmse,4), grid_model.best_params_))

ridge_params={'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20]}
lasso_params={'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1, 5, 10]}
print_best_params(ridge_reg, ridge_params)
print_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 cv 최적평균 rmse 값0.1419, 최적 alpha{'alpha': 12}
Lasso 5 cv 최적평균 rmse 값0.1421, 최적 alpha{'alpha': 0.001}
```

1.5 라쏘와 튜닝

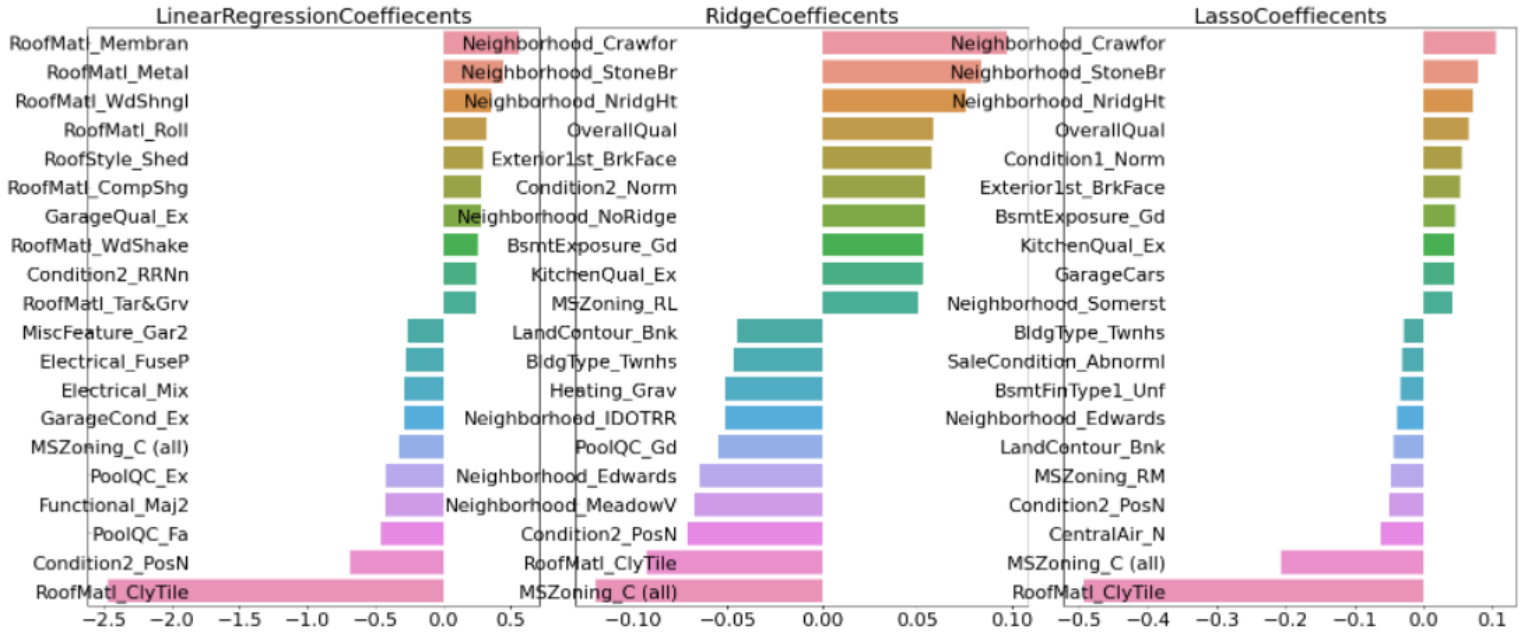
<라쏘의 문제>
1. 낮은 회귀 성능
2. 작은 회귀 계수

<TODO>
1. 데이터 분할에 문제?
2. Alpha 하이퍼파라미터 튜닝?
3. 데이터 분포도(왜곡)?
4. 이상치 데이터?

```
lr_reg=LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg=Ridge(alpha=12) #교재의 최적
ridge_reg.fit(X_train, y_train)
lasso_reg=Lasso(alpha=0.001) #교재의 최적
lasso_reg.fit(X_train, y_train)

models=[lr_reg, ridge_reg, lasso_reg]
get_rmses(models)

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

선형 모델에 최적 alpha값을 설정한 뒤,
train_test_split()으로 데이터 분할

```
LinearRegression 로그 변환된 RMSE: 0.133
Ridge 로그 변환된 RMSE: 0.125
Lasso 로그 변환된 RMSE: 0.121
```



1.5 라쏘와 튜닝

<라쏘의 문제>
작은 회귀 계수

<TODO>
1. 데이터 분포도(왜곡)?->skew() 숫자형 피처의 적용하여 왜곡 정도 확인. 단, 원-핫인 코딩했던 피처는 제외 -> 왜곡 높은 피처 로그변환
2. 이상치 데이터?

```
from scipy.stats import skew

features_index=house_df.dtypes[house_df.dtypes!='object'].index
skew_features=house_df[features_index].apply(lambda x:skew(x))

skew_features_top=skew_features[skew_features>1]
print(skew_features_top.sort_values(ascending=False))
```

```
MiscVal          24.451640
PoolArea         14.813135
LotArea          12.195142
3SsnPorch        10.293752
LowQualFinSF      9.002080
KitchenAbvGr      4.483784
BsmtFinSF2        4.250888
ScreenPorch       4.117977
BsmtHalfBath      4.099186
EnclosedPorch     3.086696
MasVnrArea        2.673661
LotFrontage       2.382499
OpenPorchSF       2.361912
BsmtFinSF1        1.683771
WoodDeckSF        1.539792
TotalBsmtSF       1.522688
MSSubClass        1.406210
1stFlrSF          1.375342
GrLivArea         1.365156
dtype: float64
```

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

```
skew_features_top=skew_features[skew_features>1]
print(skew_features_top.sort_values(ascending=False))
```

```
MiscVal          24.451640
PoolArea         14.813135
LotArea          12.195142
3SsnPorch        10.293752
LowQualFinSF      9.002080
KitchenAbvGr      4.483784
BsmtFinSF2        4.250888
ScreenPorch       4.117977
BsmtHalfBath      4.099186
EnclosedPorch     3.086696
MasVnrArea        2.673661
LotFrontage       2.382499
OpenPorchSF       2.361912
BsmtFinSF1        1.683771
WoodDeckSF        1.539792
TotalBsmtSF       1.522688
MSSubClass        1.406210
1stFlrSF          1.375342
GrLivArea         1.365156
dtype: float64
```

EWHA EURON

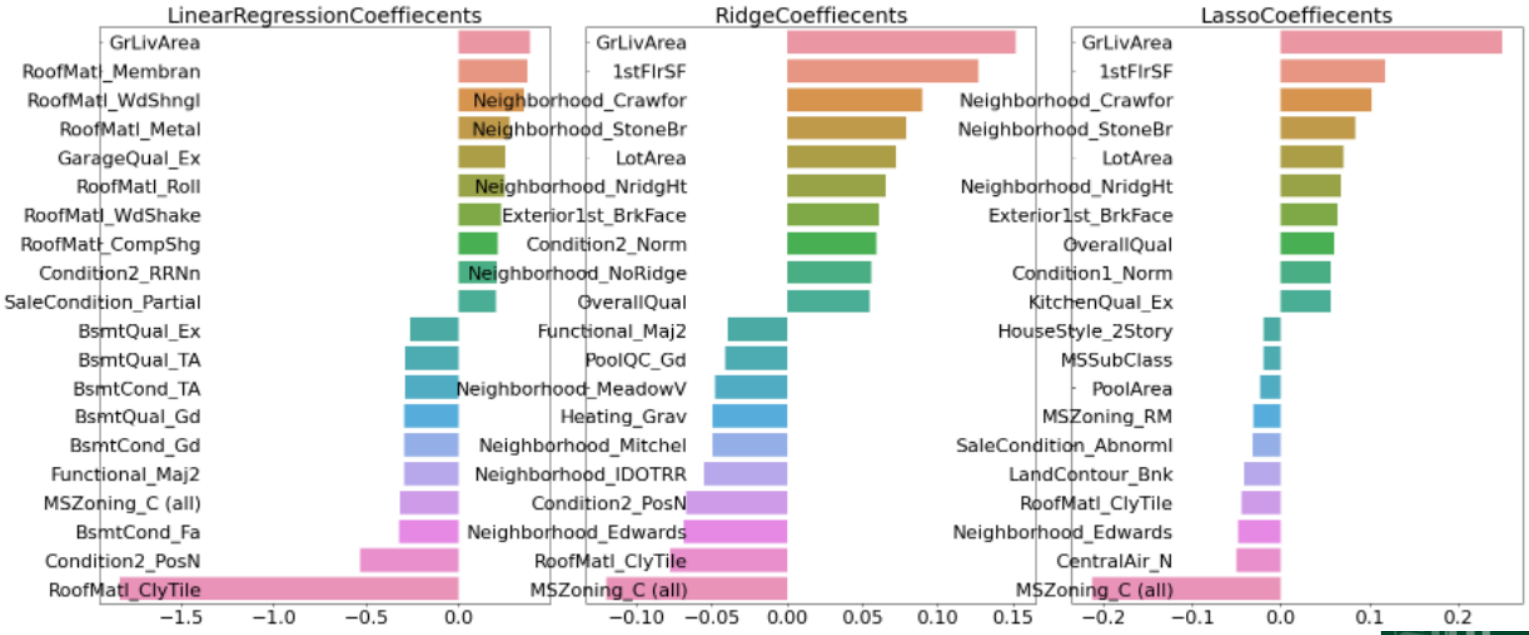
1.5 라쏘와 튜닝

<라쏘의 문제>
작은 회귀 계수

<TODO>
1. 데이터 분포도(왜곡)?->skew() 숫자형 피처의 적용하여 왜곡 정도 확인. 단, 원-핫인코딩했던 피처는 제외 -> 왜곡 높은 피처 로그변환 -> 다시 원-핫인코딩 적용 -> train_test_split()
2. 이상치 데이터?

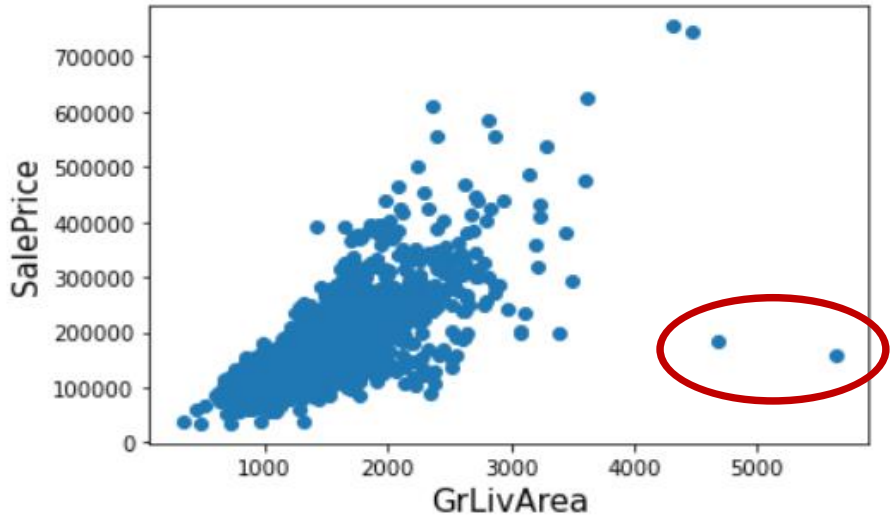
```
house_df_ohe=pd.get_dummies(house_df)
y_target=house_df_ohe['SalePrice']
X_features=house_df_ohe.drop('SalePrice',axis=1, inplace=False)
X_train, X_test, y_train, y_test=train_test_split(X_features,y_tar

ridge_params={'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20]}
lasso_params={'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5,
print_best_params(ridge_reg, ridge_params)
print_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 cv 최적평균 rmse 값0.1281, 최적 alpha{'alpha': 12}
Lasso 5 cv 최적평균 rmse 값0.1254, 최적 alpha{'alpha': 0.001}
```



1.5 라쏘와 튜닝

<TODO>
1. 이상치 데이터? -> 회귀계수가 가장 높은 GrLivArea에 관해 데이터 분포도 확인 ->이상치 삭제




```
cond1=house_df_ohe['GrLivArea']>np.log1p(4000)
cond2=house_df_ohe['SalePrice']<np.log1p(500000)
outlier_index=house_df_ohe[cond1&cond2].index

print('이상치 레코드 index:',outlier_index.values)
print('이상치 삭제 전 house_df_ohe shape: ', house_df_ohe.shape)

house_df_ohe.drop(outlier_index, axis=0, inplace=True)
print('이상치 삭제 후 house_df_ohe shape:', house_df_ohe.shape)
```

```
이상치 레코드 index: [ 523 1298]
이상치 삭제 전 house_df_ohe shape:  (1460, 290)
이상치 삭제 후 house_df_ohe shape: (1458, 290)
```

```
y_target=house_df_ohe['SalePrice']
X_features=house_df_ohe.drop('SalePrice', axis=1, inplace=False)
X_train, X_test, y_train, y_test=train_test_split(X_features,y_target,test_size=0.2, random_state=156)

ridge_params={'alpha':[0.05, 0.1, 1, 5, 8, 10, 12, 15, 20]}
lasso_params={'alpha':[0.001, 0.005, 0.008, 0.05, 0.03, 0.1, 0.5, 1, 5, 10]}
print_best_params(ridge_reg, ridge_params)
print_best_params(lasso_reg, lasso_params)
```

```
Ridge 5 cv 최적평균 rmse 값0.1135, 최적 alpha{'alpha': 8}
Lasso 5 cv 최적평균 rmse 값0.1124, 최적 alpha{'alpha': 0.001}
```

1.6 결과

회귀계수로 피처중요도를 확인하고, 회귀 성능에 의문을 가졌다
데이터 분할/alpha 하이퍼파라미터/데이터 분포도(왜곡)/이상치 데이터를 분석했다

최적의 alpha값과 이상치데이터를 제거했을 경우 회귀계수가 타당하고, 회귀 성능이 향상되었다

```
lr_reg=LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg=Ridge(alpha=8) #교재의 최적
ridge_reg.fit(X_train, y_train)
lasso_reg=Lasso(alpha=0.001) #교재의 최적
lasso_reg.fit(X_train, y_train)

models=[dummy_reg,lr_reg, ridge_reg, lasso_reg]
get_rmses(models)

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

```
DummyRegressor 로그 변환된 RMSE: 0.401
LinearRegression 로그 변환된 RMSE: 0.14
Ridge 로그 변환된 RMSE: 0.104
Lasso 로그 변환된 RMSE: 0.1
```

EWHA EURON

1.7 회귀트리 학습/예측/평가

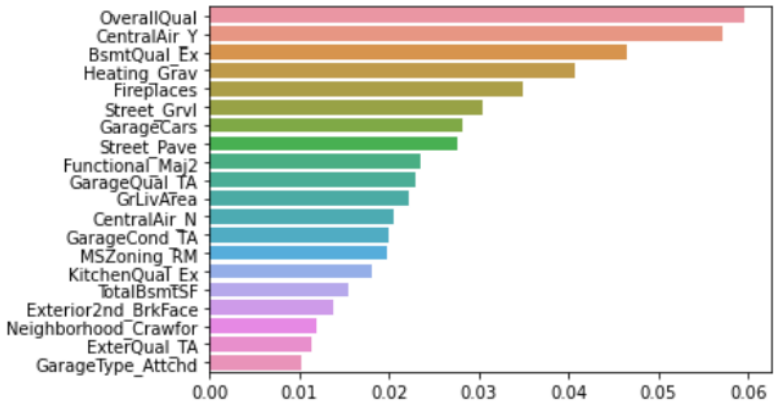
XGB,LightBGM

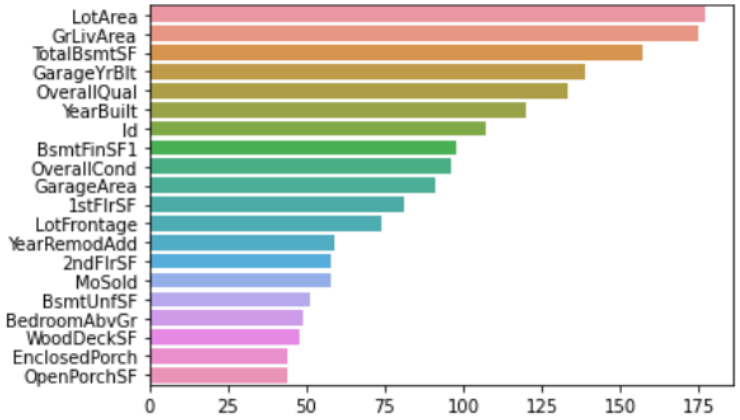
```
from xgboost import XGBRegressor

xgb_params={'n_estimators':[1000]}
xgb_reg=XGBRegressor(n_estimators=1000, learning_rate=0.05, colsample_bytree=0.5, subsample=0.8)
print_best_params(xgb_reg, xgb_params)
```

```
XGBRegressor 5 cv 최적평균 rmse 값0.1178, 최적 alpha{'n_estimators': 1000}
```

```
from lightgbm import LGBMRegressor

lgbm_params={'n_estimators':[1000]}
lgbm_reg=LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4, colsample_bytree=0.4, subsample=0.6, reg_lambda=10, n_jobs=-
print_best_params(lgbm_reg, lgbm_params)
```

```
LGBMRegressor 5 cv 최적평균 rmse 값0.1174, 최적 alpha{'n_estimators': 1000}
```





1.7 예측 결과 혼합을 통한 최종 예측

여러 모델의 예측값이 주어지면 모델의 예측값이 차지할 비율을 나눈 다음 합하여 최종 회귀 값으로 예측

A 100 40% & B 120 60% = 100*0.4+120*0.6=112

작은 예측값 <= 최종 예측값 <= 큰 예측값

라쏘와 릿지

```
ridge_reg=Ridge(alpha=8)
ridge_reg.fit(X_train,y_train)
lasso_reg=Lasso(alpha=0.001)
lasso_reg.fit(X_train,y_train)

ridge_pred=ridge_reg.predict(X_test)
lasso_pred=lasso_reg.predict(X_test)

pred=0.4*ridge_pred+0.6*lasso_pred
preds={'최종 혼합': pred, 'Ridge': ridge_pred, 'Lasso': lasso_pred}

get_rmse_pred(preds)
```

```
최종 혼합 모델의 rmse:0.1155820148467163
Ridge 모델의 rmse:0.11809251122574022
Lasso 모델의 rmse:0.11568802901099672
```

XGB와 LGBM

```
xgb_reg=XGBRegressor(n_estimators=1000, learning_rate=0.05, colsample_bytree=0.5, subsample=0.8)
lgbm_reg=LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4, colsample_bytree=0.4,


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)

pred=0.4*xgb_pred+0.6*lgbm_pred
preds={'최종혼합':pred, 'XGBM': xgb_pred, 'LGBM':lgbm_pred}

get_rmse_pred(preds)
```

```
최종혼합 모델의 rmse:0.10213451431972351
XGBM 모델의 rmse:0.10934526339369302
LGBM 모델의 rmse:0.10245063997396664
```

```
최종혼합 모델의 rmse:0.10271540867123082
XGBM 모델의 rmse:0.10934526339369302
LGBM 모델의 rmse:0.10245063997396664
```

의문? 각 모델마다 피처별 중요도, 회귀 계수가 다른데, 예측값(레이블)의 단순 덧셈이 논리적인가?
(cf. 스태킹은 학습하여 최종 예측을 진행함)

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(\hat{y}_i + 1) - log(y_i + 1))^2}$$

1.8 스태킹 앙상블 모델을 통한 회귀 예측

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error

# 개별 기반 모델에서 최종 메타 모델이 사용할 학습 및 테스트용 데이터를 생성하기 위한 함수.
def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds ):
    # 지정된 n_folds값으로 KFold 생성.
    kf = KFold(n_splits=n_folds, shuffle=False, random_state=0)
    #추후에 메타 모델이 사용할 학습 데이터 반환을 위한 넘파이 배열 초기화
    train_fold_pred = np.zeros((X_train_n.shape[0] ,1 ))
    test_pred = np.zeros((X_test_n.shape[0],n_folds))
    print(model.__class__.__name__ , ' model 시작 ')

    for folder_counter , (train_index, valid_index) in enumerate(kf.split(X_train_n)):
        #입력된 학습 데이터에서 기반 모델이 학습/예측할 폴드 데이터 셋 추출
        print('₩t 폴드 세트: ',folder_counter,' 시작 ')
        X_tr = X_train_n[train_index]
        y_tr = y_train_n[train_index]
        X_te = X_train_n[valid_index]

        #폴드 세트 내부에서 다시 만들어진 학습 데이터로 기반 모델의 학습 수행.
        model.fit(X_tr , y_tr)
        #폴드 세트 내부에서 다시 만들어진 검증 데이터로 기반 모델 예측 후 데이터 저장.
        train_fold_pred[valid_index, :] = model.predict(X_te).reshape(-1,1)
        #입력된 원본 테스트 데이터를 폴드 세트내 학습된 기반 모델에서 예측 후 데이터 저장.
        test_pred[:, folder_counter] = model.predict(X_test_n)

    # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
    test_pred_mean = np.mean(test_pred, axis=1).reshape(-1,1)

    #train_fold_pred는 최종 메타 모델이 사용하는 학습 데이터, test_pred_mean은 테스트 데이터
    return train_fold_pred , test_pred_mean
```

스태킹은 두 종류의 모델 필요

- 개별적인 기반 모델
- 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어 학습하는 최종 메타 모델

get_stacking_base_datasets( ): 개별 모델을 스태킹 모델로 제공하기 위해 데이터 세트를 생성하기 위한 함수

1.8 스태킹 앙상블 모델을 통한 회귀 예측

```
# get_stacking_base_datasets( )은 넘파이 ndarray를 인자로 사용하므로 DataFrame을 넘파이로 변환.
X_train_n = X_train.values
X_test_n = X_test.values
y_train_n = y_train.values

# 각 개별 기반(Base)모델이 생성한 학습용/테스트용 데이터 반환.
ridge_train, ridge_test = get_stacking_base_datasets(ridge_reg, X_train_n, y_train_n, X_test_n, 5)
lasso_train, lasso_test = get_stacking_base_datasets(lasso_reg, X_train_n, y_train_n, X_test_n, 5)
xgb_train, xgb_test = get_stacking_base_datasets(xgb_reg, X_train_n, y_train_n, X_test_n, 5)
lgbm_train, lgbm_test = get_stacking_base_datasets(lgbm_reg, X_train_n, y_train_n, X_test_n, 5)
```

```
Ridge  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
Lasso  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
```

```
XGBRegressor  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
LGBMRegressor  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
```

get_stacking_base_datasets( )를 모델별로 적용해 메타 모델이 사용할 학습 피처 데이터 세트와 테스트 피처 데이터 세트 추출

1.8 스태킹 앙상블 모델을 통한 회귀 예측

```
# 개별 모델이 반환한 학습 및 테스트용 데이터 세트를 Stacking 형태로 결합.
Stack_final_X_train = np.concatenate((ridge_train, lasso_train,
                                      xgb_train, lgbm_train), axis=1)
Stack_final_X_test = np.concatenate((ridge_test, lasso_test,
                                     xgb_test, lgbm_test), axis=1)

# 최종 메타 모델은 라쏘 모델을 적용.
meta_model_lasso = Lasso(alpha=0.0005)

#기반 모델의 예측값을 기반으로 새롭게 만들어진 학습 및 테스트용 데이터로 예측하고 RMSE 측정.
meta_model_lasso.fit(Stack_final_X_train, y_train)
final = meta_model_lasso.predict(Stack_final_X_test)
mse = mean_squared_error(y_test , final)
rmse = np.sqrt(mse)
print('스태킹 회귀 모델의 최종 RMSE 값은:', rmse)


```

```
스태킹 회귀 모델의 최종 RMSE 값은: 0.09723804106266824
```

각 개별 모델이 반환하는 학습용 피처 데이터와 테스트용 피처 데이터 세트를 결합해 최종 메타 모델에 적용
메타모델은 별도의 라쏘 모델을 이용

EWHA EURON

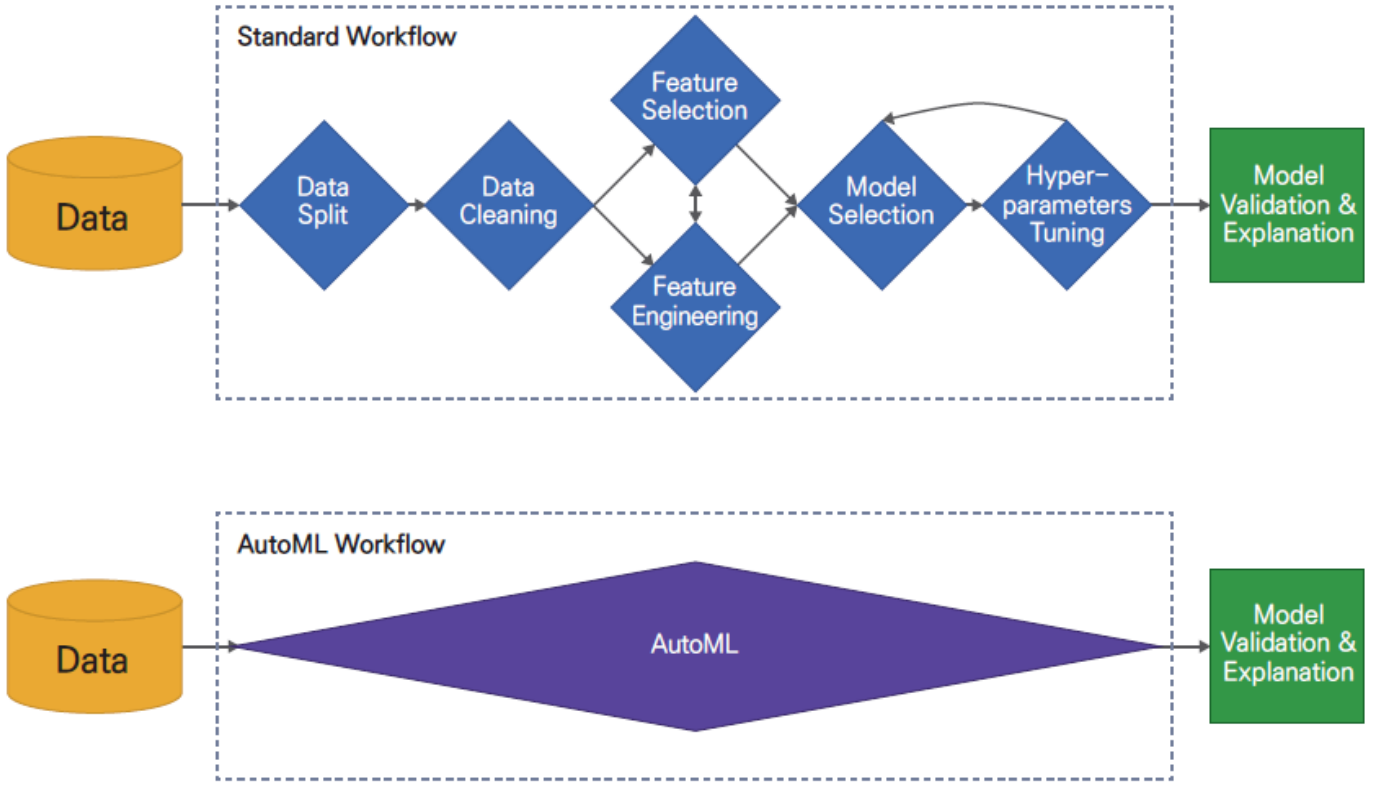
02 pycaret을 이용한 주택 가격 예측

EWHA EURON

AutoML이란?

AutoML: 데이터 성격에 맞게 자동으로 데이터 분석 모델을 추천해주는 auto machine learning 기법
머신러닝 모델 개발, 배포 등을 자동으로 처리할 수 있게 도와주는 프로세스

AutoML 장점
- 인공지능 분석 모델을 학습하기 위해 데이터에서 중요한 특징을 선택하고 인코딩하는 방식에 대한 특징 엔지니어링 자동으로 추출
- 인공지능 모델 학습에 필요한 설정들, 하이퍼 파라미터를 자동으로 탐색해주는 것
- 인공지능 모델의 구조 자체를 더 효율적인 방향으로 찾아주는 아키텍처 탐색



Pycaret이란?

Pycaret은 기존에 있던 scikit-learn, XGBoost, LightGBM, spaCy 등 여러가지 머신러닝 라이브러리를 ML High-Level API로 제작한 라이브러리
적은 코드로 머신러닝 워크 플로우를 자동화하는 오픈 소스 라이브러리
-> 머신러닝 모델 개발시 많은 시간을 소요했던 코딩, 전처리, 모델 선택, 파라미터 튜닝 작업을 자동화해주어 쉽고, 높은 생산성의 작업 가능하게 함

데이터셋만 있으면 간단하게 모델링부터 하이퍼 파라미터 튜닝과 feature importance 등의 작업까지 가능

01 Import Dataset

```
1  import pandas as pd
2  train = pd.read_csv('train.csv')
3  test = pd.read_csv('test.csv')
4  train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

02 Iteration 1 – set up without preprocessing

```
from pycaret.regression import *
reg1 = setup(train, target = 'SalePrice', session_id = 123, silent = True)
```

	Description	Value
0	session_id	123
1	Target	SalePrice
2	Original Data	(1460, 81)
3	Missing Values	True
4	Numeric Features	19
5	Categorical Features	61
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(1021, 405)
10	Transformed Test Set	(439, 405)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	reg-default-name
19	USI	7f01
20	Imputation Type	simple
21	Iterative Imputation Iteration	None
22	Numeric Imputer	mean
23	Iterative Imputation Numeric Model	None
24	Categorical Imputer	constant
25	Iterative Imputation Categorical Model	None
26	Unknown Categoricals Handling	least_frequent
27	Normalize	False

📌 데이터 준비 (set up)

데이터를 머신러닝에 사용할 수 있도록 로드 및 전처리 하는 기능
Pycaret에 구현된 머신러닝 기법들을 사용하려면 무조건 런타임에 먼저 setup이 로드되어야 함

Setup(데이터셋, target 값)
- session_id : 'random_state'와 같음
- Silent : setup이 실행될 때 데이터 타입의 입력 확인을 제어(완전히 자동화된 모드에서 실행시 True여야함)

02 Iteration 1 – set up without preprocessing

compare models

```
compare_models(blacklist = ['tr'])
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	CatBoost Regressor	15896.6	8.03637e+08	27521.8	0.8827	0.1313	0.0926
1	Gradient Boosting Regressor	18225.3	8.62755e+08	28862.4	0.8709	0.1431	0.1063
2	Extreme Gradient Boosting	18242.1	8.66742e+08	28963.9	0.8695	0.1438	0.107
3	Light Gradient Boosting Machine	18258.1	9.62618e+08	30280.5	0.8584	0.1512	0.1079
4	Random Forest	19560.8	1.12225e+09	32638.2	0.8351	0.1596	0.1158
5	AdaBoost Regressor	26262.3	1.35967e+09	36466.2	0.7942	0.2111	0.1731
6	Lasso Least Angle Regression	17471.6	1.34561e+09	34429.2	0.7919	0.165	0.1026
7	Ridge Regression	20511.8	1.50876e+09	36631.5	0.7703	0.1957	0.1249
8	Orthogonal Matching Pursuit	18642.9	1.48601e+09	35661	0.7697	0.1645	0.1101
9	Extra Trees Regressor	22812.6	1.63841e+09	39493.9	0.7534	0.1846	0.1326
10	Lasso Regression	20312.2	1.59921e+09	37780.8	0.7506	0.1927	0.1246
11	Elastic Net	21619.4	1.70987e+09	38558.1	0.7478	0.1723	0.127
12	Bayesian Ridge	25726.8	2.11528e+09	43569	0.6841	0.2078	0.1514
13	K Neighbors Regressor	30063	2.20685e+09	46336.7	0.6636	0.2278	0.1773
14	Huber Regressor	28354.1	2.25475e+09	45584.1	0.6571	0.2319	0.1732
15	Decision Tree	30761.3	2.71101e+09	50610.9	0.5989	0.2394	0.1765
16	Support Vector Machine	56136.2	6.94407e+09	82715.5	-0.0627	0.4067	0.3195
17	Passive Aggressive Regressor	49531.4	1.06904e+10	78031.3	-0.9236	0.314	0.3152
18	Random Sample Consensus	1.47362e+08	1.40042e+18	6.75997e+08	-1.74536e+08	1.3129	1534.54
19	Linear Regression	2.4445e+08	4.57901e+18	1.31013e+09	-6.76386e+08	1.6327	2543.03
20	Least Angle Regression	1.16992e+44	1.40928e+91	1.18713e+45	-1.72136e+81	35.2858	1.91773e+39

compare_models( ) : setup된 데이터를 각각 머신러닝 모델에 적용 후 비교

머신러닝으로 모델링할 때 사용되는 대부분의 알고리즘들은 다 구성되어 있고, 이들 중 어떤 모델이 가장 성능이 좋은 지 확인 가능
20가지의 각 모델별 MAE, MSE, RMSE, R2, RMSLE, MAPE와 프로세스 시간을 비교해줌
코드 실행 후 결과 테이블에 자동으로 추천 모델을 하이라이트 해줌(추천 상위 기준은 분류모델은 정확도, 회귀모델은 R2의 점수가 높은 순에서 낮은 순으로 정렬)

- compare_models(sort= 'RMSLE') 를 사용하면 비교 기준을 바꿀 수 있음
- compare_models(fold=5)를 사용하면 모델 개수 조정 가능
- n_select = n 사용하면 top n개의 모델을 반환

EWHA EURON

02 Iteration 1 – set up without preprocessing

Create and store models in variable

```
catboost = create_model('catboost', verbose = False)
gbr = create_model('gbr', verbose = False)
xgboost = create_model('xgboost', verbose = False)
```

create_model( ): model( ) 에 적힌 머신러닝 모델을 선택해서 생성

모델 파라미터들을 cross validation 과정을 통해 스스로 훈련시키고 검증(하이퍼 파라미터 탐색 자동화)

앞서 추천된 모델 중 3가지 지정해 각 모델 성능을 비교한 것

- estimator: 어떤 모델을 사용할 것인가?
- ensemble: estimator를 앙상블 한 결과 나타냄
- method: bagging, boosting 선택 가능
- fold: K-fold의 수로, 최소 20이상의 숫자 입력
- round: 점수 반올림으로 표시할 자리 적음
- cross_validation: True로 설정 시 cross_validation사용
- verbose: 진행중인 상황 나타냄
- system: internal function으로 인해 바뀌는 것을 제외하고 항상 True를 입력해야함

02 Iteration 1 – set up without preprocessing

Blend models

```
blend_top_3 = blend_models(estimator_list = [catboost, gbr, xgboost])
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	19961.4962	1.555368e+09	39438.1552	0.8100	0.1768	0.1164
1	16795.2814	7.198981e+08	26830.9170	0.8877	0.1233	0.0902
2	13260.3601	3.593057e+08	18955.3605	0.9222	0.1202	0.0868
3	15138.1882	6.243925e+08	24987.8476	0.8796	0.1309	0.0928
4	19238.3354	1.305902e+09	36137.2646	0.8672	0.1792	0.1250
5	16943.3696	8.229131e+08	28686.4624	0.8860	0.1311	0.0984
6	16726.0102	5.370227e+08	23173.7495	0.8861	0.1140	0.0930
7	18653.3367	7.870130e+08	28053.7513	0.8845	0.1425	0.1076
8	18254.4341	7.598277e+08	27564.9722	0.8866	0.1345	0.1021
9	15207.9334	5.978568e+08	24451.1107	0.8917	0.1116	0.0814
Mean	17017.8745	8.069500e+08	27827.9591	0.8802	0.1364	0.0994
SD	1971.4836	3.420273e+08	5705.6681	0.0268	0.0226	0.0129

blend_models( ) : voting 알고리즘을 구현한 모듈
compare_models( ) 에서 성능이 잘 나온 모델들을 선택하는 파라미터를 적용 시켜서 사용할 수 있음

Blending 뒤에 개선이 없음
Best individual model은 RMSLE가 0.1313 인 Catboost
Blender RMSLE 는 0.1364

02 Iteration 1 – set up without preprocessing

Stack models

```
stack1 = stack_models(estimator_list = [gbr, xgboost], meta_model = catboost, restack = True)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	20204.6090	1.988339e+09	44590.7969	0.7571	0.1837	0.1169
1	16381.6495	6.846242e+08	26165.3249	0.8932	0.1215	0.0890
2	14451.1125	4.147400e+08	20365.1652	0.9102	0.1234	0.0916
3	15877.8822	7.110417e+08	26665.3646	0.8629	0.1404	0.0974
4	20342.0227	1.529807e+09	39112.7531	0.8444	0.1772	0.1245
5	17852.5332	1.093093e+09	33061.9582	0.8486	0.1418	0.1024
6	16812.6431	6.639510e+08	25767.2461	0.8592	0.1198	0.0917
7	18875.4063	8.490590e+08	29138.6169	0.8754	0.1480	0.1082
8	18526.6998	8.139377e+08	28529.5926	0.8785	0.1386	0.1038
9	15635.7737	5.822083e+08	24128.9926	0.8945	0.1175	0.0856
Mean	17496.0332	9.330801e+08	29752.5811	0.8624	0.1412	0.1011
SD	1886.4641	4.573580e+08	6918.3859	0.0404	0.0221	0.0120

stack_models( ) : stacking ensemble 방법을 구현한 모듈
compare_models( ) 에서 성능이 잘 나온 모델들을 선택하는 파라미터를 적용시켜서 사용할 수 있음

Stacking으로부터 개선이 없고 RMSLE가 0.1313인 디폴트 하이퍼 파라미터의 catboost regressor이 best model

EWHA EURON

03 Iteration 2 – set up with preprocessing

```
from pycaret.regression import *
reg1 = setup(train, target = 'SalePrice', session_id = 123,
             normalize = True, normalize_method = 'zscore',
             transformation = True, transformation_method = 'yeo-johnson', transform_target = True,
             ignore_low_variance = True, combine_rare_levels = True,
             numeric_features=['OverallQual', 'OverallCond', 'BsmtFullBath', 'BsmtHalfBath',
                               'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                               'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'PoolArea'],
             silent = True #silent is set to True for unattended run during kernel execution
             )
```

- normalize: numeric feature 들을 주어진 범위로 스케일링함
- normalize_method: 스케일링 method를 결정(디폴트는 zscore)
- transformation: 데이터의 power transform 여부
- transformation_method: transformation 의 타입
- transform_target: 타겟 변수가 define 된 method 를 이용해 변환됨(피처 변환과 별개로 타겟 변환)
- ignore_low_variance: low variance 를 가진 카테고리 피처들이 데이터에서 제거됨
- combine_rare_levels: 특정 임계값보다 낮은 카테고리 피처들의 레벨의 빈도 백분위수가 단일 수준으로 결합
- numeric_features: 데이터 타입이 맞지 않거나 silent 파라미터가 true일 때, numeric_features 가 데이터 타입을 define 할 수 있음 (리스트 안의 문자들은 numeric인 column 이름들을 나열한 것)

03 Iteration 2 – set up with preprocessing

```
from pycaret.regression import *
reg1 = setup(train, target = 'SalePrice', session_id = 123,
             normalize = True, normalize_method = 'zscore',
             transformation = True, transformation_method = 'yeo-johnson', transform_target = True,
             ignore_low_variance = True, combine_rare_levels = True,
             numeric_features=['OverallQual', 'OverallCond', 'BsmtFullBath', 'BsmtHalfBath',
                               'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                               'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'PoolArea'],
             silent = True #silent is set to True for unattended run during kernel execution
             )
```

	Description	Value
0	session_id	123
1	Transform Target	True
2	Transform Target Method	box-cox
3	Original Data	(1460, 81)
4	Missing Values	True
5	Numeric Features	33
6	Categorical Features	46
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(1460, 81)
11	Transformed Train Set	(1021, 247)
12	Transformed Test Set	(439, 247)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	True
16	Normalize Method	zscore
17	Transformation	True
18	Transformation Method	yeo-johnson
19	PCA	False
20	PCA Method	None
21	PCA Components	None
22	Ignore Low Variance	True
23	Combine Rare Levels	True
24	Rare Level Threshold	0.1
24	Rare Level Threshold	0.1
25	Numeric Binning	False
26	Remove Outliers	False
27	Outliers Threshold	None
28	Remove Multicollinearity	False
29	Multicollinearity Threshold	None
30	Clustering	False
31	Clustering Iteration	None
32	Polynomial Features	False
33	Polynomial Degree	None
34	Trignometry Features	False
35	Polynomial Threshold	None
36	Group Features	False
37	Feature Selection	False
38	Features Selection Threshold	None
39	Feature Interaction	False
40	Feature Ratio	False
41	Interaction Threshold	None

03 Iteration 2 – set up with preprocessing

compare models

```
compare_models(blacklist = ['tr'])
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Gradient Boosting Regressor	16733.8	7.72428e+08	27231	0.8847	0.1323	0.0953
1	CatBoost Regressor	15639.9	8.44593e+08	27896.1	0.8791	0.1275	0.0891
2	Support Vector Machine	15393.9	8.97393e+08	28488.3	0.8719	0.1282	0.0874
3	Light Gradient Boosting Machine	17467.5	9.08805e+08	29319.2	0.8666	0.1403	0.0992
4	Extreme Gradient Boosting	17300.3	9.36891e+08	29647.1	0.8625	0.1364	0.0976
5	Random Forest	18987.4	1.15884e+09	33105.8	0.83	0.1521	0.1075
6	Extra Trees Regressor	18900.3	1.18108e+09	33291.5	0.8286	0.1488	0.1044
7	Orthogonal Matching Pursuit	16656	1.29507e+09	32431.9	0.8086	0.1388	0.0956
8	AdaBoost Regressor	24368.4	1.40978e+09	36991.4	0.7872	0.1807	0.1364
9	K Neighbors Regressor	21489.5	1.45955e+09	37146.9	0.7858	0.1655	0.1181
10	Bayesian Ridge	16270.6	1.55029e+09	33986	0.7722	0.1375	0.0934
11	Ridge Regression	17430.1	1.54282e+09	34806.1	0.7714	0.1453	0.1001
12	Huber Regressor	15640.6	1.90299e+09	34779.1	0.7219	0.1382	0.0934
13	Decision Tree	27814	2.01147e+09	43582.6	0.6999	0.2202	0.1619
14	Linear Regression	20040.6	1.9342e+09	40424.1	0.6993	1.0892	0.1167
15	Random Sample Consensus	21540.2	2.98427e+09	48146.2	0.5304	1.832	0.1295
16	Passive Aggressive Regressor	26117.9	3.98171e+09	48077	0.4044	0.2091	0.1588
17	Elastic Net	44289.5	4.69437e+09	67842.3	0.2876	0.3187	0.2492
18	Lasso Regression	56217.4	6.80379e+09	81878.9	-0.0413	0.4058	0.3281
19	Lasso Least Angle Regression	56217.4	6.80379e+09	81878.9	-0.0413	0.4058	0.3281
20	Least Angle Regression	179483	3.88587e+10	196954	-5.1764	11.9992	0.9967

Catboost regressor RMSLE 가 0.1313에서 0.1275로 약간 개선

03 Iteration 2 – set up with preprocessing

Create and store models in variable

```
gbr = create_model('gbr', verbose = False)
catboost = create_model('catboost', verbose = False)
svm = create_model('svm', verbose = False)
lightgbm = create_model('lightgbm', verbose = False)
xgboost = create_model('xgboost', verbose = False)
```

03 Iteration 2 – set up with preprocessing

Blend models

```
blend_top_5 = blend_models(estimator_list = [gbr,catboost,svm,lightgbm,xgboost])
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	18326.8314	1.431609e+09	37836.6124	0.8251	0.1615	0.1016
1	15313.3472	8.314091e+08	28834.1653	0.8703	0.1187	0.0803
2	13130.3880	3.468765e+08	18624.6215	0.9249	0.1149	0.0842
3	14084.3179	5.136570e+08	22664.0030	0.9010	0.1148	0.0818
4	18868.4802	1.627255e+09	40339.2520	0.8345	0.1672	0.1112
5	15315.9318	1.107892e+09	33285.0081	0.8465	0.1262	0.0852
6	15165.5791	4.784369e+08	21873.2004	0.8985	0.1065	0.0837
7	15739.0281	5.947321e+08	24387.1305	0.9127	0.1259	0.0901
8	16546.4420	6.607448e+08	25704.9563	0.9014	0.1236	0.0916
9	13482.5395	4.172085e+08	20425.6834	0.9244	0.1059	0.0759
Mean	15597.2885	8.009821e+08	27397.4633	0.8839	0.1265	0.0886
SD	1796.2483	4.216430e+08	7096.5576	0.0353	0.0201	0.0101

Blending top models RMSLE 0.1275에서 0.1265로 약간 개선

03 Iteration 2 – set up with preprocessing

Stack models

```
stack2 = stack_models(estimator_list = [gbr,catboost,lightgbm,xgboost], meta_model = svm, restack = True)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	17409.8890	1.305815e+09	36136.0573	0.8405	0.1593	0.0985
1	14335.1123	7.003411e+08	26463.9587	0.8907	0.1165	0.0785
2	13324.0524	3.545430e+08	18829.3131	0.9232	0.1113	0.0826
3	12909.2774	4.667761e+08	21605.0009	0.9100	0.1122	0.0764
4	19690.0468	2.127673e+09	46126.7065	0.7836	0.1788	0.1133
5	16794.3541	2.024131e+09	44990.3384	0.7196	0.1552	0.0939
6	15066.6020	5.096206e+08	22574.7789	0.8919	0.1051	0.0818
7	16134.0545	7.539584e+08	27458.3035	0.8894	0.1397	0.0936
8	15065.3775	4.834227e+08	21986.8748	0.9279	0.1036	0.0823
9	15258.0220	5.286918e+08	22993.2992	0.9042	0.1128	0.0839
Mean	15598.6788	9.254972e+08	28916.4631	0.8681	0.1295	0.0885
SD	1908.4813	6.276333e+08	9451.7384	0.0642	0.0254	0.0108

Stacking 이후 개선이 없음

04 Iteration 3 – set up with advance preprocessing

```
from pycaret.regression import *
reg1 = setup(train, target = 'SalePrice', session_id = 123,
             normalize = True, normalize_method = 'zscore',
             transformation = True, transformation_method = 'yeo-johnson', transform_target = True,
             numeric_features=['OverallQual', 'OverallCond', 'BsmtFullBath', 'BsmtHalfBath',
                               'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                               'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'PoolArea'],
             ordinal_features= {'ExterQual': ['Fa', 'TA', 'Gd', 'Ex'],
                                'ExterCond' : ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                'BsmtQual' : ['Fa', 'TA', 'Gd', 'Ex'],
                                'BsmtCond' : ['Po', 'Fa', 'TA', 'Gd'],
                                'BsmtExposure' : ['No', 'Mn', 'Av', 'Gd'],
                                'HeatingQC' : ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                'KitchenQual' : ['Fa', 'TA', 'Gd', 'Ex'],
                                'FireplaceQu' : ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                'GarageQual' : ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                'GarageCond' : ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                'PoolQC' : ['Fa', 'Gd', 'Ex']},
             polynomial_features = True, trigonometry_features = True, remove_outliers = True, outliers_threshold = 0.01,
             silent = True #silent is set to True for unattended run during kernel execution
             )
```

- ordinal_features: 카테고리 피처들 인코딩
- polynomial_features, trigonometry_features: 기존 numeric 피처들을 이용해 새 피처들 파생
- remove_outliers: train 데이터의 outlier 제거
- outliers_threshold: 제거되는 outlier 퍼센트 지정

	Description	Value
0	session_id	123
1	Transform Target	True
2	Transform Target Method	box-cox
3	Original Data	(1460, 81)
4	Missing Values	True
5	Numeric Features	33
6	Categorical Features	46
7	Ordinal Features	True
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(1445, 81)
11	Transformed Train Set	(1011, 312)
12	Transformed Test Set	(434, 312)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	True
16	Normalize Method	zscore
17	Transformation	True
18	Transformation Method	yeo-johnson
19	PCA	False
20	PCA Method	None
20	PCA Method	None
21	PCA Components	None
22	Ignore Low Variance	False
23	Combine Rare Levels	False
24	Rare Level Threshold	None
25	Numeric Binning	False
26	Remove Outliers	True
27	Outliers Threshold	0.01
28	Remove Multicollinearity	False
29	Multicollinearity Threshold	None
30	Clustering	False
31	Clustering Iteration	None
32	Polynomial Features	True
33	Polynomial Degree	2
34	Trignometry Features	True
35	Polynomial Threshold	0.1
36	Group Features	False
37	Feature Selection	False
38	Features Selection Threshold	None
39	Feature Interaction	False
40	Feature Ratio	False
41	Interaction Threshold	None

EWHA EURON

04 Iteration 3 – set up with advance preprocessing

compare models

```
compare_models(blacklist = ['tr'])
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	CatBoost Regressor	14903.5	8.78698e+08	27512.8	0.8489	0.1248	0.0855
1	Huber Regressor	13579.6	7.35629e+08	24203.5	0.8486	0.1235	0.0836
2	Orthogonal Matching Pursuit	15423.1	8.59547e+08	27827.3	0.8456	0.1287	0.088
3	Bayesian Ridge	14759.1	8.30011e+08	26499.7	0.8428	0.1232	0.0856
4	Ridge Regression	14790.8	8.11028e+08	26263.6	0.8413	0.1267	0.0876
5	Light Gradient Boosting Machine	16810.8	9.39723e+08	29438.2	0.8389	0.1373	0.0957
6	Extra Trees Regressor	17938	1.03175e+09	31191.7	0.8219	0.1461	0.1024
7	Random Forest	17953.5	1.05049e+09	31534.8	0.8212	0.1453	0.1023
8	Gradient Boosting Regressor	16335.5	9.78845e+08	29102.9	0.8206	0.1335	0.0939
9	Extreme Gradient Boosting	16500.9	1.05235e+09	30059.8	0.8205	0.1345	0.0942
10	Support Vector Machine	17388.6	1.37212e+09	33982.3	0.801	0.1478	0.097
11	Passive Aggressive Regressor	17779.8	1.00522e+09	28881.4	0.7983	0.1417	0.1049
12	AdaBoost Regressor	22695.9	1.40487e+09	36712.1	0.7713	0.1693	0.1242
13	K Neighbors Regressor	22271.1	1.53947e+09	37472.4	0.7673	0.1705	0.1232
14	Decision Tree	25529.9	1.48154e+09	38034.7	0.7476	0.209	0.1512
15	Random Sample Consensus	20786.3	2.10576e+09	42822	0.5964	1.9847	0.1236
16	Elastic Net	43550.5	4.70451e+09	66981.8	0.2724	0.3183	0.2474
17	Lasso Regression	54718	6.63106e+09	79959.5	-0.0456	0.398	0.3202
18	Lasso Least Angle Regression	54718	6.63106e+09	79959.5	-0.0456	0.398	0.3202
19	Linear Regression	88207.7	4.75526e+12	725594	-427.246	1.5592	1.3617
20	Least Angle Regression	3.54369e+29	1.26678e+62	3.56137e+30	-1.87967e+52	12.0989	2.14148e+24

04 Iteration 3 – set up with advance preprocessing

Tune models

tune_models( ) : 모델의 하이퍼파라미터를 최적화하는 모듈
하이퍼 파라미터 반복 횟수나 최적화할 매트릭을 선택할 수 있음

- estimator: 사용할 모델 입력
- fold: K-fold의 수로, 최소 20이상의 숫자 입력
- round: 점수 반올림으로 표시할 자리
- n_iter: random grid search를 한 회차 당 반복할 횟수
- custom_grid: 직접 파라미터 범위 조정
- optimize: 파라미터 튜닝 과정에서 어떤 점수 따라갈 것인지 선택
('Accuracy' ,' AUC' ,' Recall' ,' Precision' ,' F1')
- choose_better: 성능이 높아지지 않을 경우 tuning을 하지 않은 모델 반환
- verbose: 진행중 상황 나타냄

ID	Name
'lr'	Linear Regression
'lasso'	Lasso Regression
'ridge'	Ridge Regression
'en'	Elastic Net
'lar'	Least Angle Regression
'llar'	Lasso Least Angle Regression
'omp'	Orthogonal Matching Pursuit
'br'	Bayesian Ridge
'ard'	Automatic Relevance Determination
'par'	Passive Aggressive Regressor
'ransac'	Random Sample Consensus
'tr'	TheilSen Regressor
'huber'	Huber Regressor
'kr'	Kernel Ridge
'svm'	Support Vector Machine
'knn'	K Neighbors Regressor
'dt'	Decision Tree
'rf'	Random Forest
'et'	Extra Trees Regressor
'ada'	AdaBoost Regressor
'gbr'	Gradient Boosting Regressor
'mlp'	Multi Level Perceptron
'xgboost'	Extreme Gradient Boosting
'lightgbm'	Light Gradient Boosting
'catboost'	CatBoost Regressor

EWHA EURON

04 Iteration 3 – set up with advance preprocessing

Tune models

```
huber = tune_model('huber', n_iter = 100)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	13498.0328	3.786105e+08	19457.9170	0.9226	0.1218	0.0825
1	14372.7724	4.232725e+08	20573.5878	0.9619	0.1356	0.0958
2	15128.1410	6.302112e+08	25104.0082	0.8634	0.1625	0.1052
3	11517.8009	2.345910e+08	15316.3631	0.9255	0.0964	0.0743
4	14609.1411	5.035843e+08	22440.6840	0.9455	0.1122	0.0800
5	15891.0790	3.447466e+09	58715.1248	0.1573	0.1668	0.0931
6	12780.4967	4.629581e+08	21516.4616	0.9305	0.1131	0.0756
7	12279.9130	3.604687e+08	18986.0137	0.9465	0.1077	0.0753
8	11766.8365	2.961185e+08	17208.0926	0.9597	0.0869	0.0615
9	14755.8248	4.625474e+08	21506.9142	0.9106	0.1251	0.0949
Mean	13660.0038	7.199828e+08	24082.5167	0.8524	0.1228	0.0838
SD	1435.9799	9.151762e+08	11832.8018	0.2333	0.0247	0.0125

```
omp = tune_model('omp', n_iter = 100)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	14075.6558	4.339981e+08	20832.6218	0.9113	0.1304	0.0866
1	18908.8139	1.145016e+09	33838.0823	0.8969	0.1422	0.1014
2	15710.1189	6.018364e+08	24532.3546	0.8695	0.1548	0.1006
3	10975.0324	2.015912e+08	14198.2822	0.9360	0.0890	0.0687
4	16830.8527	8.373583e+08	28937.1448	0.9093	0.1053	0.0792
5	17717.1226	3.042673e+09	55160.4295	0.2563	0.1655	0.1008
6	13922.0864	6.024009e+08	24543.8567	0.9096	0.1157	0.0800
7	13759.4682	4.579465e+08	21399.6843	0.9320	0.1120	0.0829
8	15740.0834	5.646112e+08	23761.5485	0.9233	0.1059	0.0781
9	14135.6158	3.848431e+08	19617.4187	0.9256	0.1202	0.0911
Mean	15177.4850	8.272275e+08	26682.1423	0.8470	0.1241	0.0869
SD	2173.7833	7.783141e+08	10737.3526	0.1978	0.0227	0.0107

04 Iteration 3 – set up with advance preprocessing

Tune models

```
ridge = tune_model('ridge', n_iter = 100)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	15050.9981	4.894199e+08	22122.8357	0.9000	0.1305	0.0894
1	16961.0036	6.838990e+08	26151.4631	0.9384	0.1316	0.0977
2	15726.5963	5.450161e+08	23345.5792	0.8819	0.1455	0.1017
3	11163.7203	2.114547e+08	14541.4809	0.9329	0.0921	0.0695
4	17734.4217	1.225972e+09	35013.8863	0.8672	0.1172	0.0857
5	15856.0129	1.872290e+09	43269.9669	0.5423	0.1498	0.0925
6	14890.0665	7.092271e+08	26631.3184	0.8935	0.1263	0.0848
7	15318.9889	1.015730e+09	31870.5136	0.8493	0.1406	0.0855
8	15318.0569	5.009207e+08	22381.2585	0.9319	0.0996	0.0749
9	15424.6054	4.674996e+08	21621.7395	0.9096	0.1224	0.0966
Mean	15344.4471	7.721429e+08	26695.0042	0.8647	0.1255	0.0878
SD	1628.2442	4.582165e+08	7714.8979	0.1110	0.0177	0.0096

```
br = tune_model('br', n_iter = 100)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	13976.3628	4.333965e+08	20818.1773	0.9114	0.1247	0.0836
1	16758.2894	9.058793e+08	30097.8285	0.9185	0.1292	0.0931
2	15465.8235	5.969941e+08	24433.4624	0.8706	0.1581	0.1039
3	11245.8841	2.094682e+08	14473.0164	0.9335	0.0881	0.0697
4	17201.9146	9.492806e+08	30810.3972	0.8972	0.1104	0.0822
5	16847.3120	3.314685e+09	57573.2973	0.1898	0.1666	0.0970
6	13434.4075	5.560883e+08	23581.5247	0.9165	0.1244	0.0808
7	12929.2747	4.456171e+08	21109.6441	0.9339	0.1089	0.0778
8	14609.0746	4.527242e+08	21277.3174	0.9385	0.0990	0.0741
9	14908.0689	4.170724e+08	20422.3497	0.9194	0.1214	0.0937
Mean	14737.6412	8.281205e+08	26459.7015	0.8429	0.1231	0.0856
SD	1814.4618	8.559134e+08	11313.9169	0.2185	0.0231	0.0103

04 Iteration 3 – set up with advance preprocessing

Tune models

```
lightgbm = tune_model('lightgbm', n_iter = 50)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	16895.2552	6.716274e+08	25915.7755	0.8627	0.1427	0.0999
1	19393.7240	1.627986e+09	40348.3054	0.8534	0.1580	0.1029
2	18120.7828	6.720559e+08	25924.0414	0.8543	0.1589	0.1166
3	12922.8567	3.082360e+08	17556.6517	0.9021	0.1068	0.0788
4	19176.3628	1.723235e+09	41511.8604	0.8134	0.1302	0.0867
5	17220.6111	1.780241e+09	42192.9048	0.5648	0.1575	0.0999
6	14668.8595	6.158294e+08	24815.9109	0.9076	0.1372	0.0893
7	15844.6880	6.695658e+08	25875.9693	0.9006	0.1337	0.0964
8	16923.9801	7.386557e+08	27178.2209	0.8996	0.1197	0.0862
9	17076.0399	5.896434e+08	24282.5742	0.8860	0.1330	0.1037
Mean	16824.3160	9.397075e+08	29560.2214	0.8445	0.1378	0.0960
SD	1867.5448	5.174205e+08	8117.9325	0.0974	0.0163	0.0104

```
par = tune_model('par', n_iter = 100)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	19451.1817	6.517765e+08	25529.9143	0.8668	0.1422	0.1136
1	17448.4809	6.782191e+08	26042.6408	0.9389	0.1419	0.1067
2	20300.9924	9.109428e+08	30181.8289	0.8025	0.1851	0.1319
3	11701.4347	2.474870e+08	15731.7198	0.9214	0.0950	0.0744
4	16950.9793	6.952615e+08	26367.8126	0.9247	0.1198	0.0897
5	18668.4178	2.810641e+09	53015.4772	0.3130	0.1690	0.1071
6	15563.2449	5.480239e+08	23409.9108	0.9177	0.1259	0.0911
7	15942.5503	4.617375e+08	21488.0778	0.9315	0.1294	0.0971
8	17690.9310	5.453934e+08	23353.6588	0.9259	0.1136	0.0937
9	17362.7782	6.384593e+08	25267.7523	0.8766	0.1411	0.1104
Mean	17108.0991	8.187942e+08	27038.8793	0.8419	0.1363	0.1016
SD	2272.4744	6.836174e+08	9364.4646	0.1807	0.0249	0.0151

04 Iteration 3 – set up with advance preprocessing

Blend models

```
blend_all = blend_models(estimator_list = [huber, omp, ridge, br])
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	13579.3892	4.032133e+08	20080.1718	0.9176	0.1238	0.0822
1	15705.8602	6.870236e+08	26211.1343	0.9382	0.1279	0.0918
2	14687.2054	5.499336e+08	23450.6631	0.8808	0.1508	0.0986
3	10680.8191	1.862903e+08	13648.8212	0.9409	0.0854	0.0671
4	16333.2187	7.935038e+08	28169.1996	0.9141	0.1065	0.0797
5	15770.5584	2.801441e+09	52928.6447	0.3152	0.1586	0.0910
6	13301.0841	5.547586e+08	23553.3148	0.9167	0.1167	0.0773
7	12772.2291	4.666873e+08	21602.9462	0.9308	0.1079	0.0757
8	13338.5428	4.044251e+08	20110.3234	0.9450	0.0909	0.0662
9	14205.3143	4.036219e+08	20090.3424	0.9220	0.1185	0.0908
Mean	14037.4221	7.250899e+08	24984.5561	0.8621	0.1187	0.0821
SD	1600.5141	7.102047e+08	10042.9998	0.1831	0.0221	0.0103

04 Iteration 3 – set up with advance preprocessing

Evaluate Bayesian ridge model

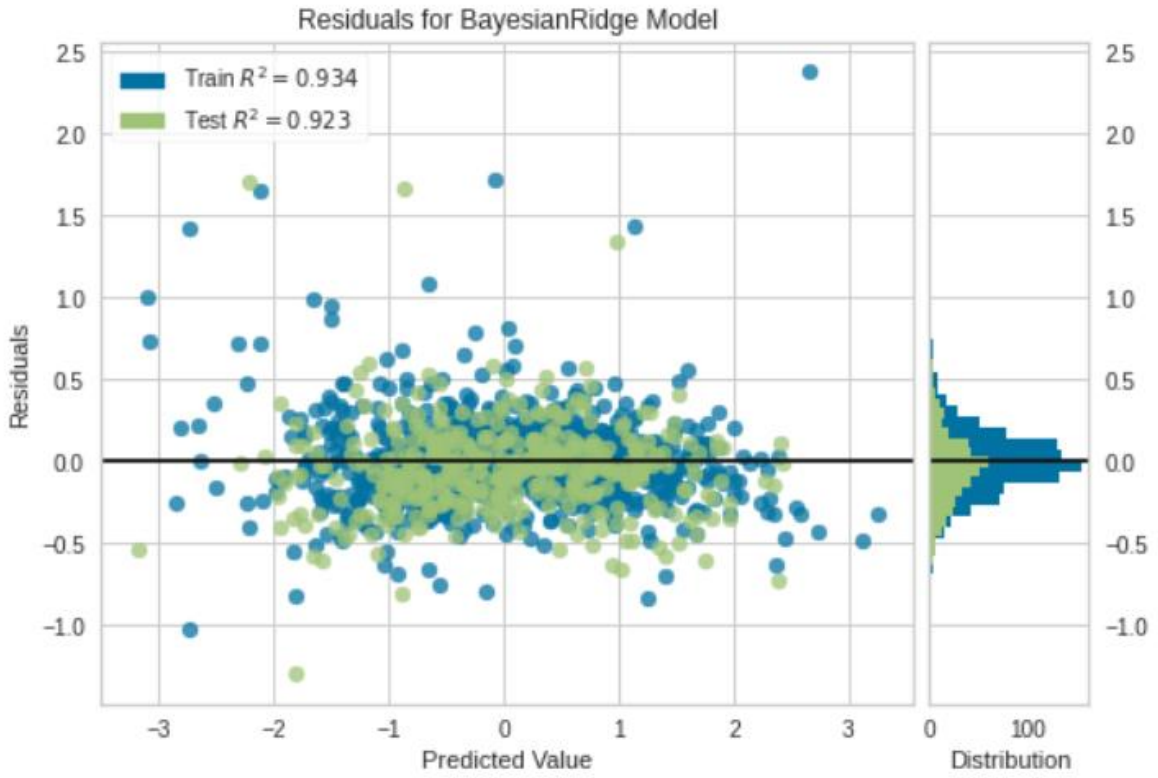
Plot_model( ) : 학습한 모델에 대한 각종 지표들을 시각화한 플롯을 그려주는 모듈
Auc, threshold, confusion matrix 등 약 15가지 이상의 다양한 플롯들 지원

Name	Plot
Residuals Plot	‘residuals’
Prediction Error Plot	‘error’
Cooks Distance Plot	‘cooks’
Recursive Feature Selection	‘rfe’
Learning Curve	‘learning’
Validation Curve	‘vc’
Manifold Learning	‘manifold’
Feature Importance (top 10)	‘feature’
Feature Importance (all)	'feature_all'
Model Hyperparameter	‘parameter’

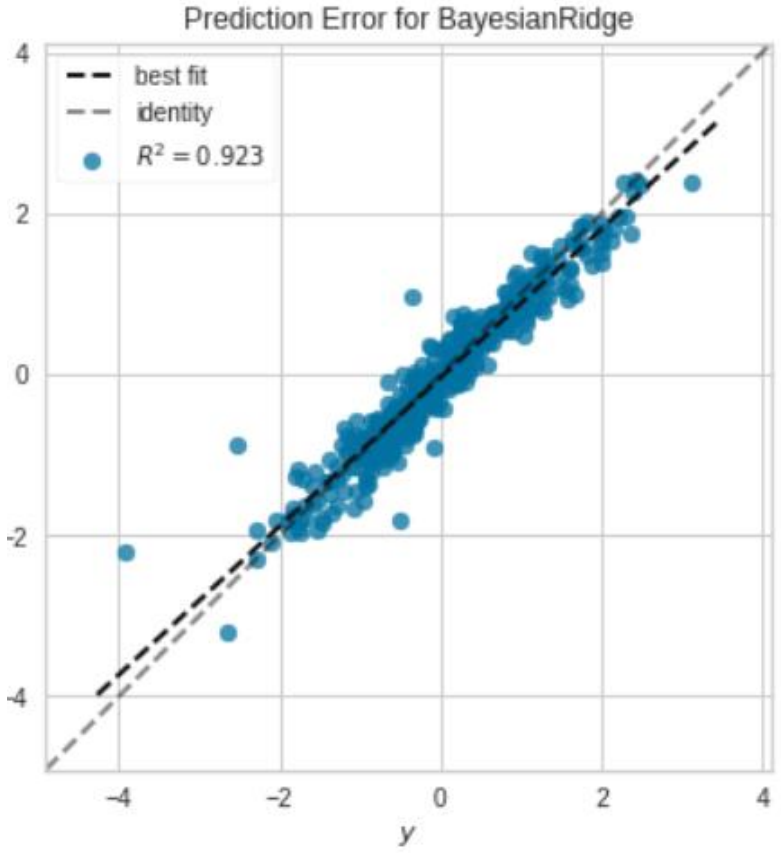
04 Iteration 3 – set up with advance preprocessing

Evaluate Bayesian ridge model

```
1  plot_model(br, plot = 'residuals')
```



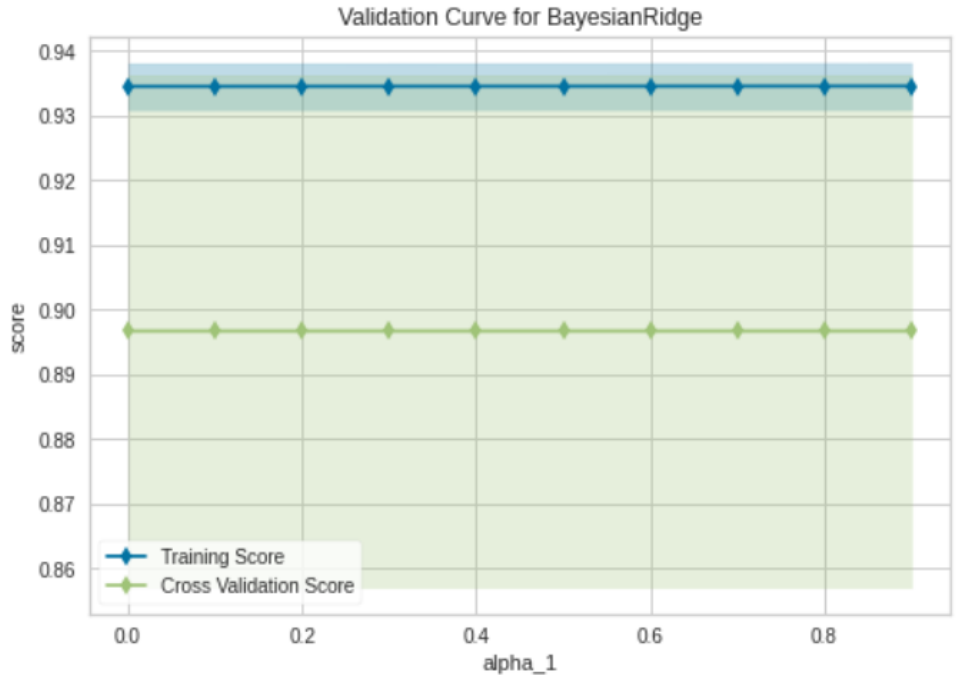

```
plot_model(br, plot = 'error')
```



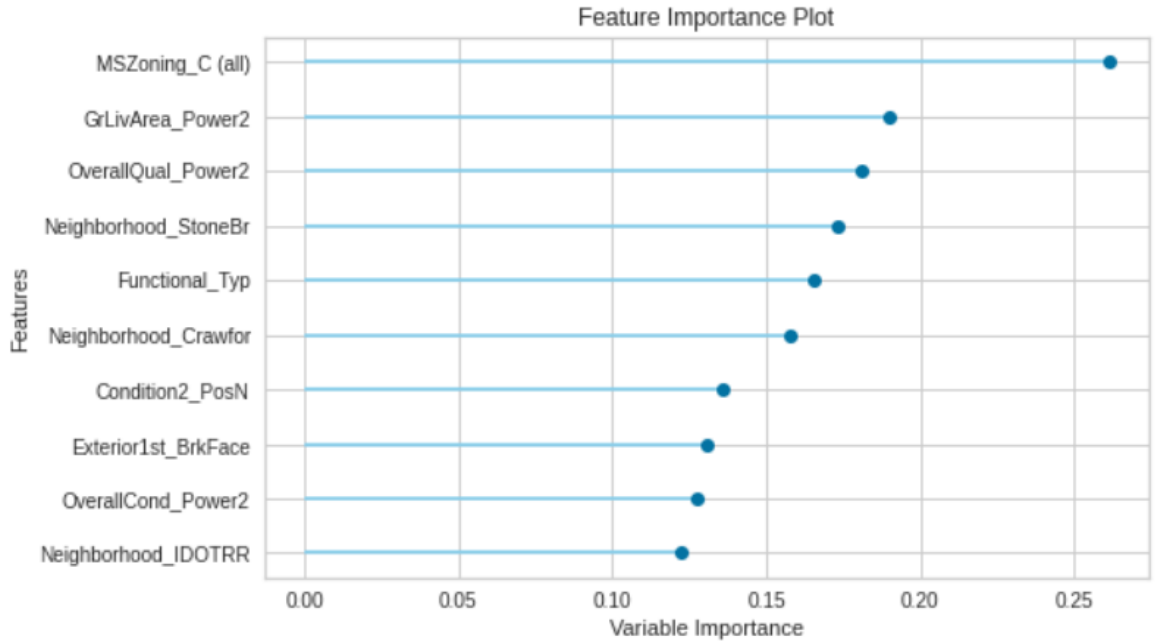
04 Iteration 3 – set up with advance preprocessing

Evaluate Bayesian ridge model

```
plot_model(br, plot = 'vc')
```



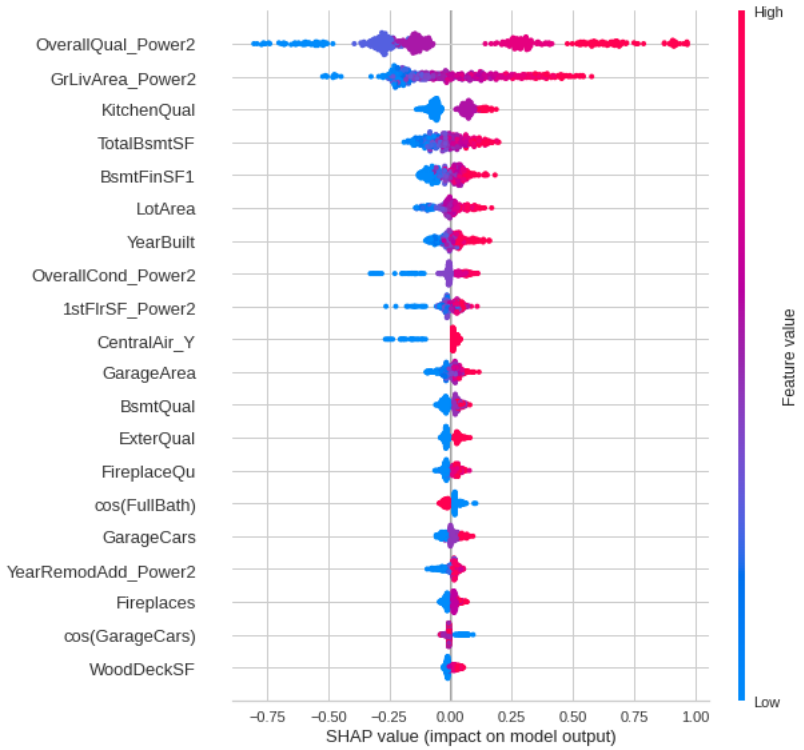

```
plot_model(br, plot = 'feature')
```



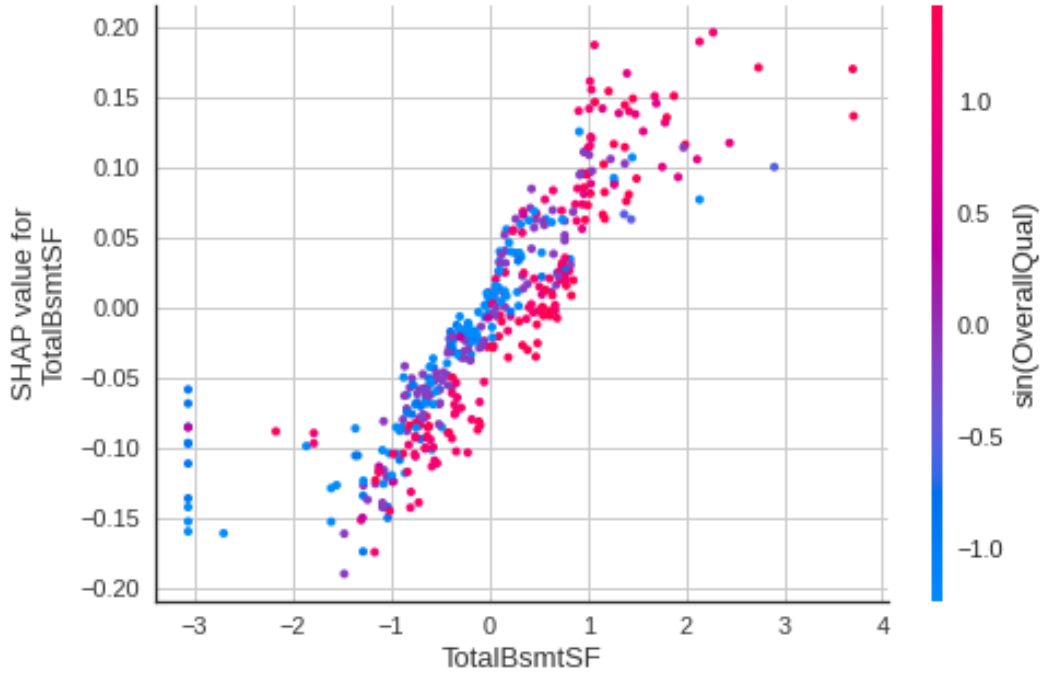
04 Iteration 3 – set up with advance preprocessing

Interpret LightGBM

```
interpret_model(lightgbm)
```




```
interpret_model(lightgbm, plot = 'correlation', feature = 'TotalBsmtSF')
```

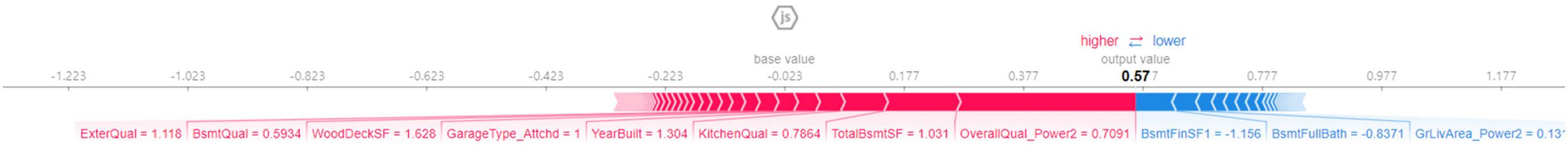


Interpret_model( ): 모델이 예측한 결과에 대해 각 파라미터들이 얼마나 영향을 줬는지 시각화해서 보여줌
SHAP 를 사용하여 plot을 그려줌

04 Iteration 3 – set up with advance preprocessing

Interpret LightGBM

```
1   interpret_model(lightgbm, plot = 'reason', observation = 0)
```



05 Finalize blender and predict test dataset

```
1   # check predictions on hold-out
2   predict_model(blend_all);
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Voting Regressor	13530.1052	4.449803e+08	21094.5565	0.9268	0.1072	0.0747

05 Finalize blender and predict test dataset

마지막 학습

```
final_blender = finalize_model(blend_all)
print(final_blender)
```

```
VotingRegressor(estimators=[('Huber Regressor_0',
                             HuberRegressor(alpha=0.5869, epsilon=1.55,
                                            fit_intercept=True, max_iter=100,
                                            tol=1e-05, warm_start=False)),
                            ('Orthogonal Matching Pursuit_1',
                             OrthogonalMatchingPursuit(fit_intercept=True,
                                                       n_nonzero_coefs=78,
                                                       normalize=False,
                                                       precompute='auto',
                                                       tol=None)),
                            ('Ridge_2',
                             Ridge(alpha=0.336, copy_X=True, fit_intercept=True,
                                   max_iter=None, normalize=True,
                                   random_state=123, solver='auto',
                                   tol=0.001)),
                            ('Bayesian Ridge_3',
                             BayesianRidge(alpha_1=0.1, alpha_2=0.05,
                                           alpha_init=None, compute_score=True,
                                           copy_X=True, fit_intercept=True,
                                           lambda_1=0.1, lambda_2=0.01,
                                           lambda_init=None, n_iter=300,
                                           normalize=False, tol=0.001,
                                           verbose=False))],
                n_jobs=None, weights=None)
```

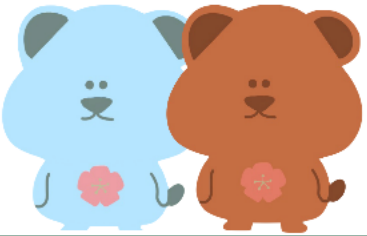
05 Finalize blender and predict test dataset

```
predictions = predict_model(final_blender, data = test)
predictions.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	Label
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal	122664.9116
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal	160301.5086
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal	186283.7094
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2010	WD	Normal	199598.1946
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	0	NaN	NaN	NaN	0	1	2010	WD	Normal	195325.7252

5 rows × 81 columns

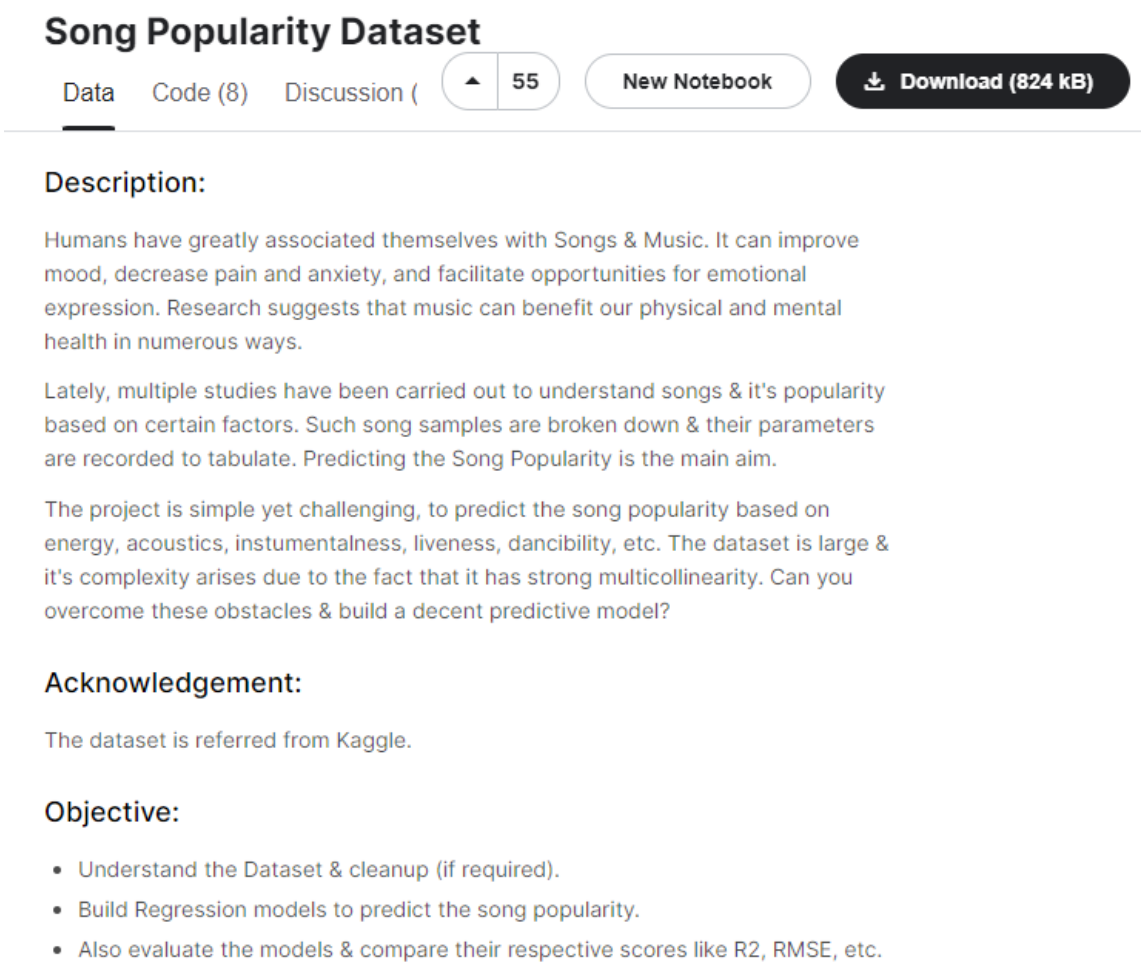
03. Song Popularity



EWHA EURON

3.1 대회 소개

Song Popularity Dataset



Song Popularity Dataset

Data Code (8) Discussion (55 New Notebook Download (824 kB)

Description:

Humans have greatly associated themselves with Songs & Music. It can improve mood, decrease pain and anxiety, and facilitate opportunities for emotional expression. Research suggests that music can benefit our physical and mental health in numerous ways.

Lately, multiple studies have been carried out to understand songs & it's popularity based on certain factors. Such song samples are broken down & their parameters are recorded to tabulate. Predicting the Song Popularity is the main aim.

The project is simple yet challenging, to predict the song popularity based on energy, acoustics, instumentalness, liveness, dancibility, etc. The dataset is large & it's complexity arises due to the fact that it has strong multicollinearity. Can you overcome these obstacles & build a decent predictive model?

Acknowledgement:

The dataset is referred from Kaggle.

Objective:

- Understand the Dataset & cleanup (if required).
- Build Regression models to predict the song popularity.
- Also evaluate the models & compare their respective scores like R2, RMSE, etc.

- 음악은 신체적, 정신적 건강에 도움이 됨
- 최근 특정 요소를 바탕으로 노래와 노래의 인기를 이해하기 위한 연구 진행 중
- 노래 샘플을 분해하여 여러 parameters를 기록
- 노래의 energy, acoustics, instumentalness, liveness, dancibility 등을 바탕으로 노래의 인기를 예측하는 것이 목표
- 크고 복잡한 데이터, 다중공선성 강함

3.2 Data Description

Columns	설명	특징
Song_name	Name of the song	사용 X
song_popularity	Song popularity	Numerical – Target 변수!
Song_duration_ms	time duration for the song	Numerical (단위: ms)
Acousticness	describes how acoustic a song is	Numerical (0~1)
Danceability	How danceable (1 is more danceable)	Numerical (0~1)
Energy	represents a perceptual measure of intensity and activity	Numerical (0~1)
Instrumentalness	amount of vocals in the song	Numerical (0~1)
Key	integers map to pitches using standard Pitch Class notation	Categorical (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
Liveness	probability that the song was recorded with a live audience	Numerical (0~1)
Loudness	loudness of a track in dB	Numerical
Audio_mode	indicates the modality of a track	Categorical (0: 1)
Speechiness	presence of spoken words in the song	Numerical (0~1)
Tempo	estimated tempo of a track	Numerical
Time_signature	how many beats are in each measure	Categorical (0, 1, 2, 3, 4, 5)
Audio_valence	musical positiveness conveyed by a track	Numerical (0~1)

3.3 Notebook 소개

문제 정의

- Understand the Dataset & cleanup (if required)
- Build Regression models to predict the song popularity.
- Also evaluate the models & compare their respective scores like R2, RMSE, etc.

노트의 핵심 아이디어

- 다중공선성 해결 – VIF, RFE, PCA
- 각 모델을 평가하는 과정에 Evaluate()라는 함수를 미리 만들어서 사용

3.3 Notebook 소개

1. Data Exploration

2. Exploratory Data Analysis (EDA)

3. Data Pre-processing

4. Data Manipulation

5. Feature Selection/Extraction
- VIF
- RFE
- PCA

6. Predictive Modelling
- MLR
- 릿지
- 라쏘
- 엘라스틱넷
- Polynomial

7. Project Outcomes & Conclusion

3.4 EDA

'song_name' drop
Target = 'song_popularity'

```
#Importing the dataset

df = pd.read_csv('song_data.csv')

df.drop(['song_name'], axis=1, inplace=True)
display(df.head())

target = 'song_popularity'
features = [i for i in df.columns if i not in [target]]

original_df = df.copy(deep=True)

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} samples.'.format(df.shape[1], df.shape
```

	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	audio_m
0	73	262333	0.005520	0.496	0.682	0.000029	8	0.0589	-4.095	1
1	66	216933	0.010300	0.542	0.853	0.000000	3	0.1080	-6.407	0
2	76	231733	0.008170	0.737	0.463	0.447000	0	0.2550	-7.828	1
3	74	216933	0.026400	0.451	0.970	0.003550	0	0.1020	-4.938	1
4	56	223826	0.000954	0.447	0.766	0.000000	10	0.1130	-5.065	1

Inference: The Datset consists of 14 features & 18835 samples.

```
#Checking number of unique rows in each feature

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

for i in range(df[features].shape[1]):
    if nu.values[i]<=16:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical features.'.format(len(nf),len(cf)
```

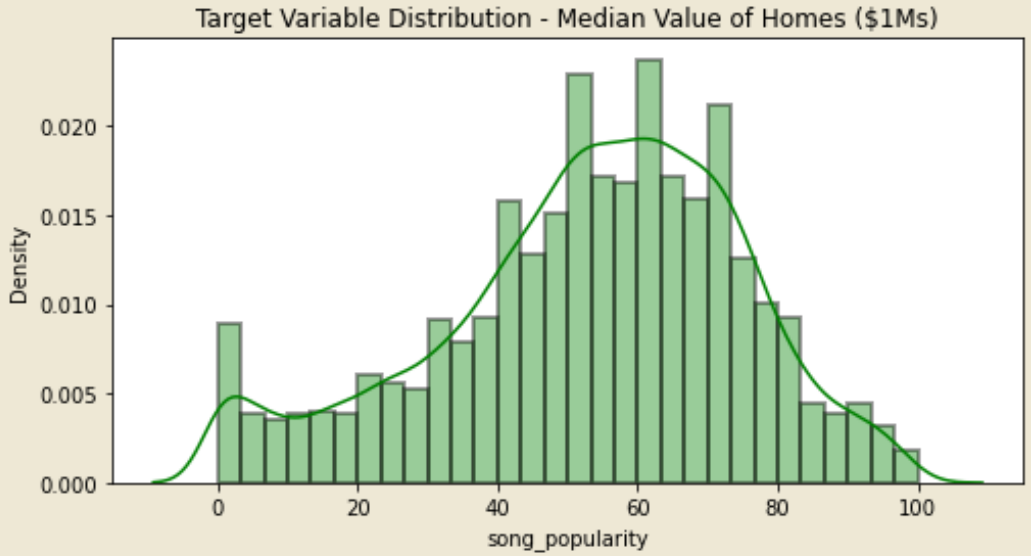
Inference: The Datset has 10 numerical & 3 categorical features.

16을 기준으로
Numerical & categorical features 구분

3.4 EDA

Target 변수
- 평균 60 정도의 정규 분포

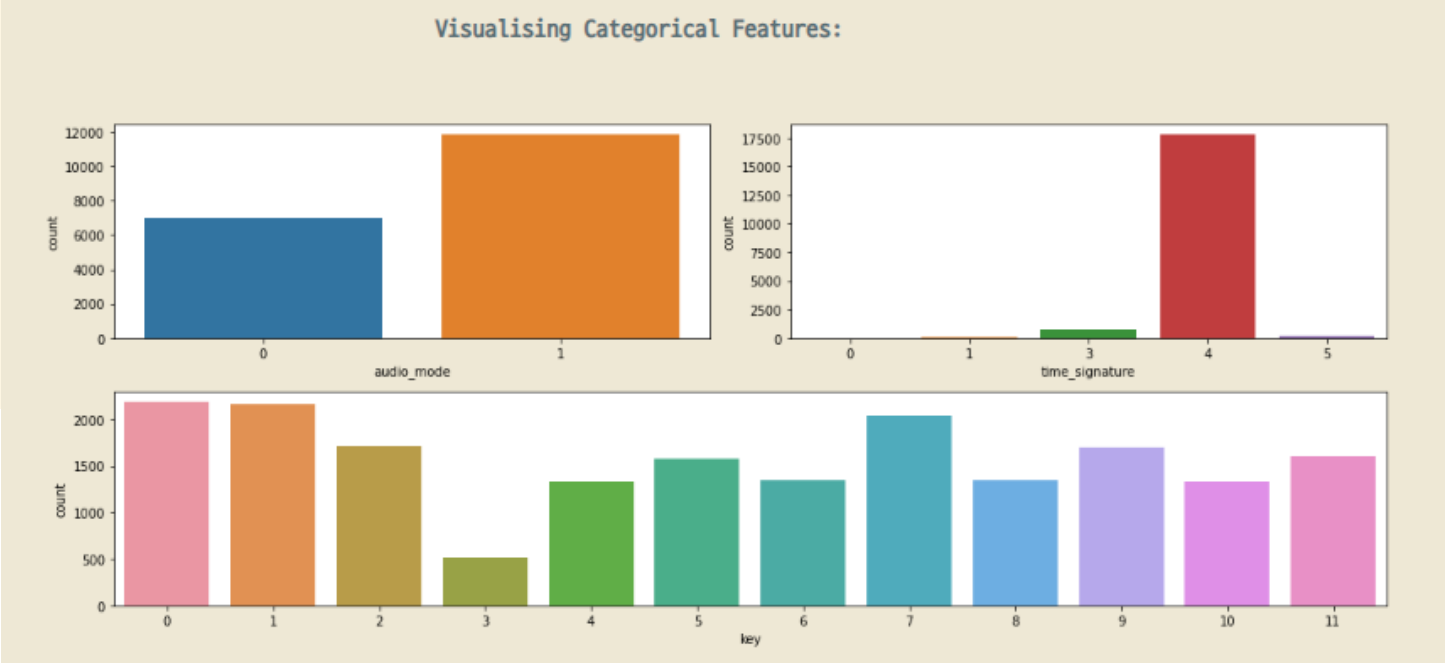
```
plt.figure(figsize=[8,4])
sns.distplot(df[target], color='g',hist_kws=dict(edgecolor="black", linewidth=2), bins=30)
plt.title('Target Variable Distribution - Median Value of Homes ($1Ms)')
plt.show()
```



3.4 EDA

Categorical 변수
- audio_mode, time_signature, key

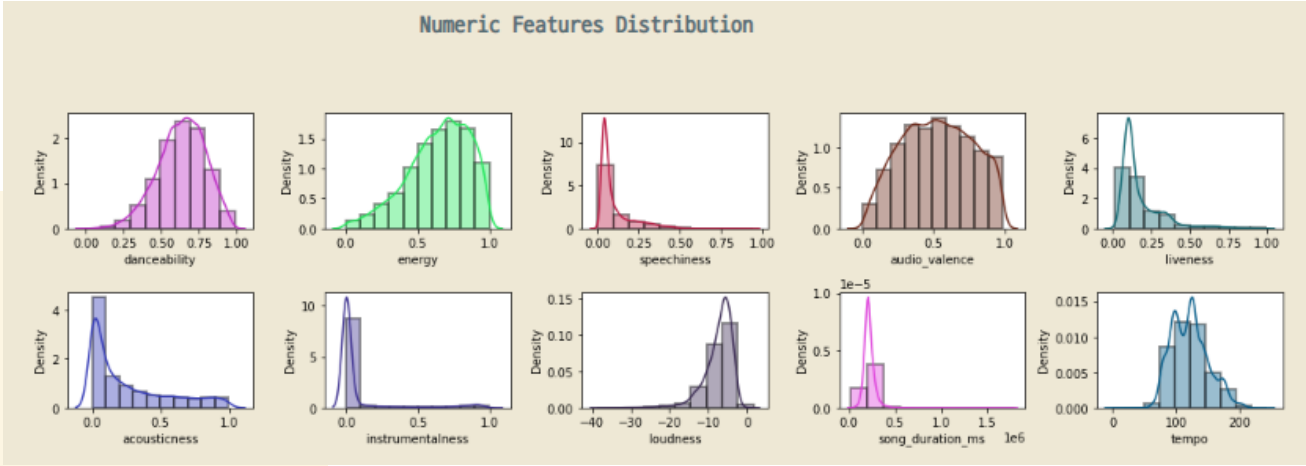
```
print('\033[1mVisualising Categorical Features:'.center(100))

n=2
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=8:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    else:
        plt.subplot(2,1,2)
        sns.countplot(df[cf[i]])

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

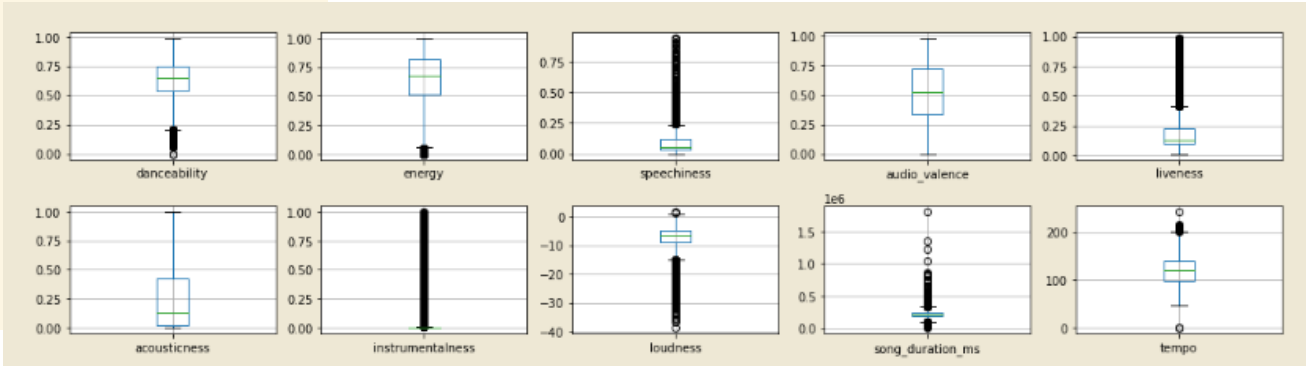


3.4 EDA

Numerical 변수
- Outlier 있는 것으로 보임

```
print('\033[1mNumeric Features Distribution'.center(100))

n=5

clr=['r','g','b','g','b','r']

plt.figure(figsize=[15,4*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    sns.distplot(df[nf[i]],hist_kws=dict(edgecolor="black", linewidth=2), bins=10,
                 color=list(np.random.randint([255,255,255])/255))
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,4*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    df.boxplot(nf[i])
plt.tight_layout()
plt.show()
```

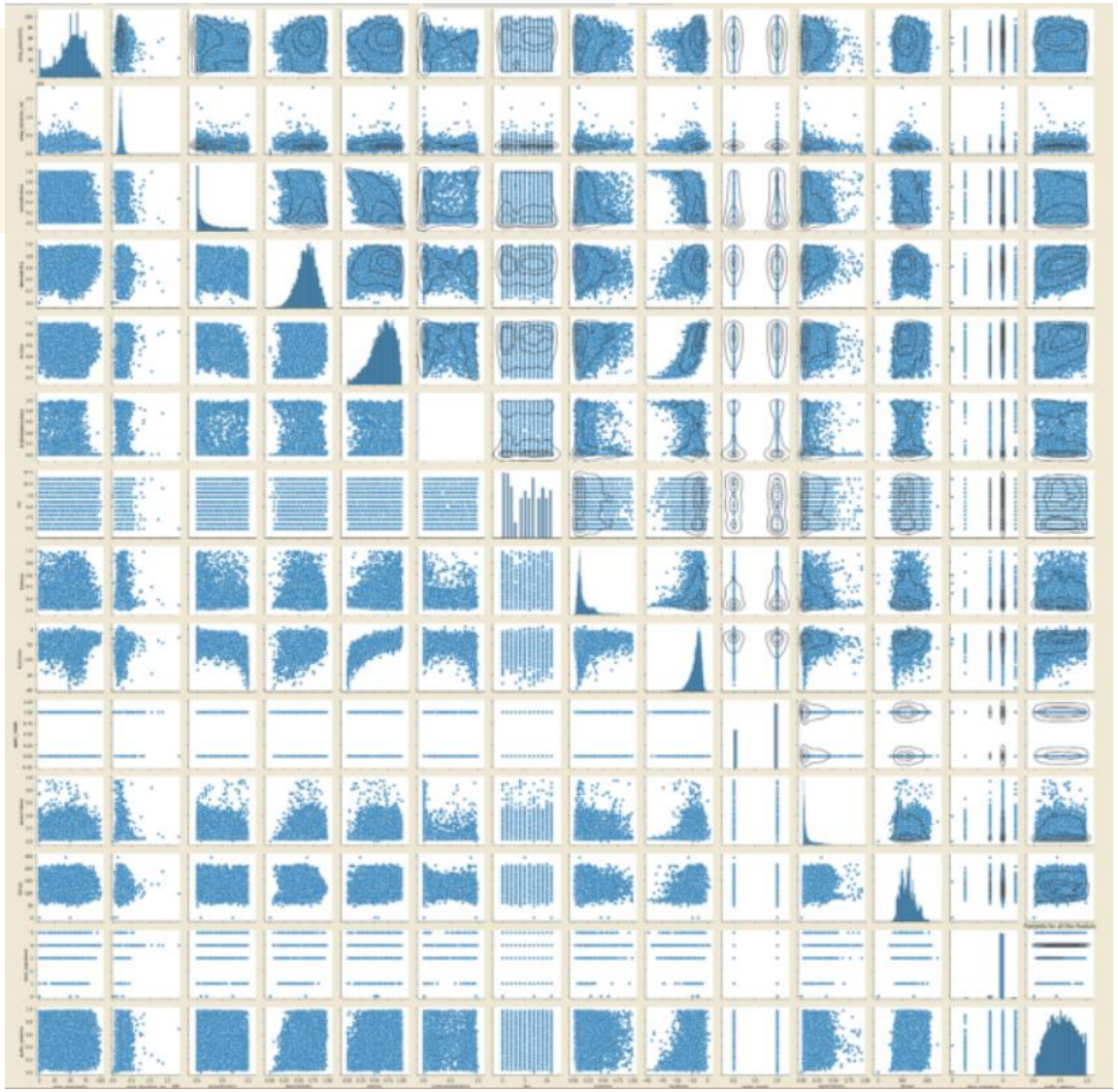




3.4 EDA

```
g = sns.pairplot(df)
plt.title('Pairplots for all the Feature')
g.map_upper(sns.kdeplot, levels=4, color=".2")
plt.show()
```

선형 관계 → 다중공선성



3.5 Data Preprocessing & Manipulation

```
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)
```

	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

결측치 X

```
counter = 0
rs,cs = original_df.shape

df.drop_duplicates(inplace=True)

if df.shape==(rs,cs):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped/fixed ---> {rs-df.shape[0]}')
```

Duplicated data 삭제

Inference: Number of duplicates dropped/fixed ---> 3911

EWHA EURON

3.5 Data Preprocessing & Manipulation

```
df3 = df.copy()

ecc = nvc[nvc['Percentage']!=0].index.values
fcc = [i for i in cf if i not in ecc]
#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    #print(i)
    if df3[i].nunique()==2:
        if oh==True: print("\033[1mOne-Hot Encoding on features:\033[0m")
        print(i);oh=False
        df3[i]=pd.get_dummies(df3[i], drop_first=True, prefix=str(i))
    if (df3[i].nunique()>2 and df3[i].nunique()<17):
        if dm==True: print("\n\033[1mDummy Encoding on features:\033[0m")
        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.shape
```

```
One-Hot Encoding on features:
audio_mode

Dummy Encoding on features:
time_signature
key


(14924, 27)
```

범주형 변수 encoding

```
df1 = df3.copy()

#features1 = [i for i in features if i not in ['CHAS','RAD']]
features1 = nf

for i in features1:
    Q1 = df1[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)
display(df1.head())
print('\n\033[1mInference:\033[0m\nBefore removal of outliers, The dataset had {} samples.'.format(df3.shape[0]))
print('After removal of outliers, The dataset now has {} samples.'.format(df1.shape[0]))
```

	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	liveness	loudness	audio_mode
0	73	262333	0.005520	0.496	0.682	0.000029	0.0589	-4.095	1
1	66	216933	0.010300	0.542	0.853	0.000000	0.1080	-6.407	0
2	74	216933	0.026400	0.451	0.970	0.003550	0.1020	-4.938	1
3	56	223826	0.000954	0.447	0.766	0.000000	0.1130	-5.065	1
4	80	235893	0.008950	0.316	0.945	0.000002	0.3960	-3.169	0

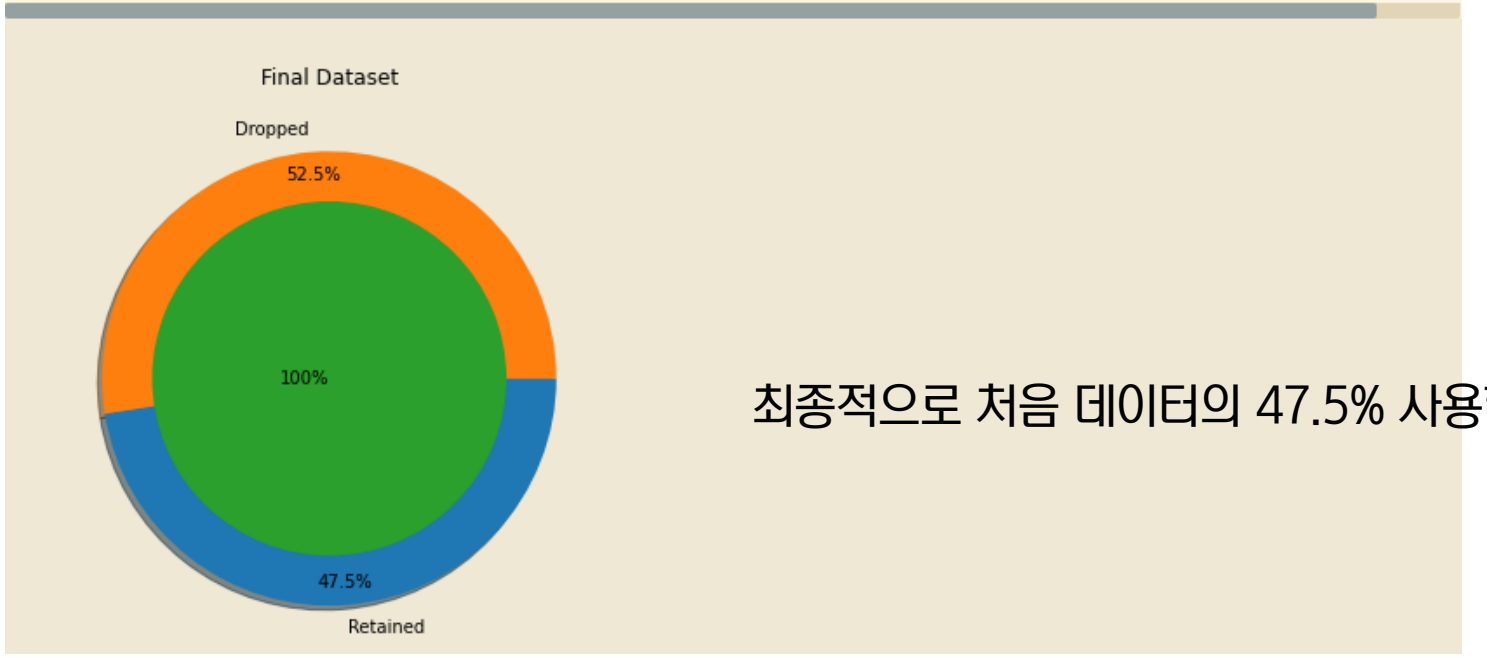
5 rows × 27 columns

```
Inference:
Before removal of outliers, The dataset had 14924 samples.
After removal of outliers, The dataset now has 8950 samples.
```

Outlier 삭제

3.5 Data Preprocessing & Manipulation

```
df = df1.copy()
df.columns=[i.replace('-','_') for i in df.columns]

plt.title('Final Dataset')
plt.pie([df.shape[0], original_df.shape[0]-df.shape[0]], radius = 1, labels=['Retained','Dropped'], counterclock=
        autopct='%1.1f%%', pctdistance=0.9, explode=[0,0], shadow=True)
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78)
plt.show()

print(f'\n\033[1mInference:\033[0m After the cleanup process,{original_df.shape[0]-df.shape[0]} samples were drop
while retaining {round((df.shape[0]*100/(original_df.shape[0])),2)}% of the data.')
```



최종적으로 처음 데이터의 47.5% 사용할 것

EWHA EURON

3.5 Data Preprocessing & Manipulation

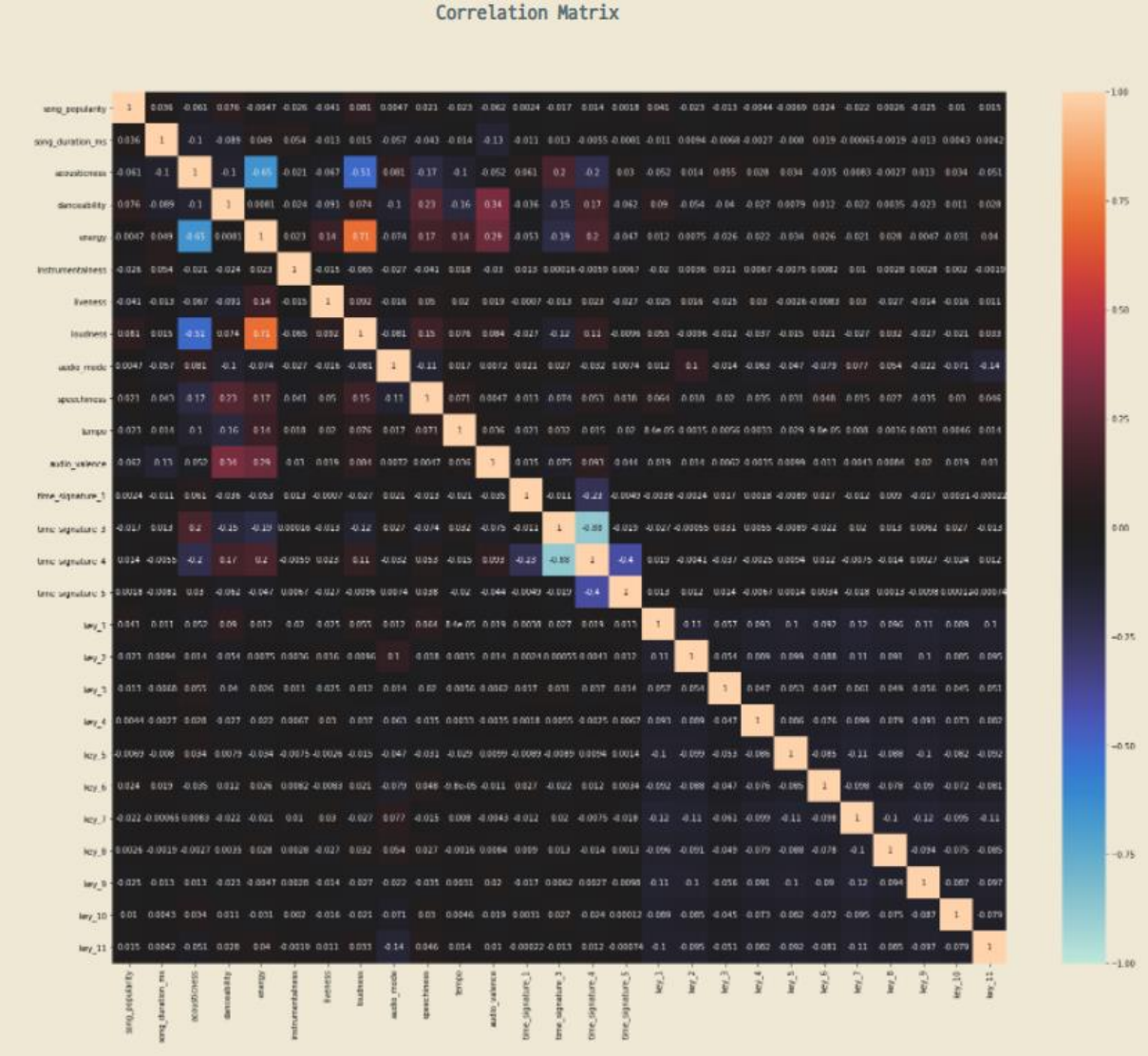
Train, Test split

```
m=[]
for i in df.columns.values:
    m.append(i.replace(' ','_'))

df.columns = m
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)
```

StandardScaler() 사용

```
std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(120))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(120))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

3.6 Feature Selection/Extraction



Omnibus:	437.955	Durbin-Watson:	2.019
Prob(Omnibus):	0.000	Jarque-Bera (JB):	522.571
Skew:	-0.661	Prob(JB):	3.35e-114
Kurtosis:	2.946	Cond. No.	4.45e+15

Condition number 30 이상
→ 다중공선성 강함

3.6 Feature Selection/Extraction

VIF (Variance Inflation Factors)

- 분산팽창요인
- 다중회귀모델에서 독립변수 사이의 상관관계 측정하는 척도

$$VIF_i = \frac{1}{1 - {R_i}^2}$$

- 10 넘으면 다중공선성이 있다고 판단 (5를 기준으로 잡기도 함)
- 두 독립변수 사이에 상관관계 0 → 두 변수의 VIF 모두 높음

3.6 Feature Selection/Extraction

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]

DROP=[];b=[]

for i in range(len(Train_X_std.columns)):
    vif = pd.DataFrame()
    X = Train_X_std.drop(DROP,axis=1)
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    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:
        DROP.append(vif.loc[0][0])
        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)))
        Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

print('Dropped Features --> ',DROP)

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
plt.legend()
plt.grid()
plt.show()
```

1) VIF를 기준으로 내림차순으로 정렬
2) VIF가 1보다 크면 DROP에 넣고 제외시킴
3) Linear regression 돌려서 RMSE 결과 기록
→ 이 과정을 반복

RMSE 변화를 그래프로 확인

```
Dropped Features -->  ['time_signature_4', 'energy', 'key_7', 'acousticness', 'danceability', 'key_1', 'key_1
1', 'key_9', 'key_2', 'loudness', 'audio_mode', 'key_5', 'speechiness', 'audio_valence', 'key_4', 'key_6', 'key
_3', 'key_8']
```

2.4

3.6 Feature Selection/Extraction

RFE (Recursive Feature Elimination)

- Feature selection algorithm

1) 주어진 ML 알고리즘에 fitting 하여 importance에 따라 피처의 랭킹 매김
2) 가장 중요하지 않은 피처를 제외시켜 다시 fitting
→ 정해진 수의 피처만 남을 때까지 이 과정 반복

- n_features_to_select: 몇 개의 피처를 남길건지
- fit() function 사용하여 fit
- support_: True or False로 어떤 피처가 선택되었는지 제공

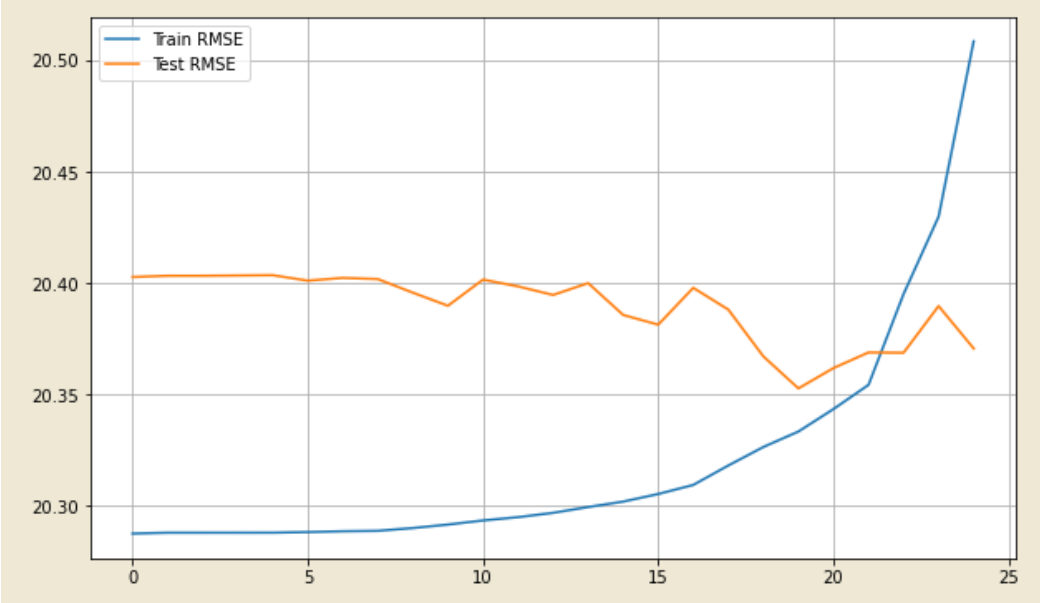
참고: https://machinelearningmastery.com/rfe-feature-selection-in-python/

3.6 Feature Selection/Extraction

```
from sklearn.feature_selection import RFE

Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

m=df.shape[1]-2
for i in range(m):
    lm = LinearRegression()
    rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-i)
    rfe = rfe.fit(Train_X_std, Train_Y)

    LR = LinearRegression()
    LR.fit(Train_X_std.loc[:,rfe.support_], Train_Y)

    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)))

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
plt.legend()
plt.grid()
plt.show()
```

1) 현재의 피처 중 1개를 제외시키도록 RFE run & fit
2) Linear regression 돌려서 RMSE 결과 기록
→ 이 과정을 반복

RMSE 변화를 그래프로 확인



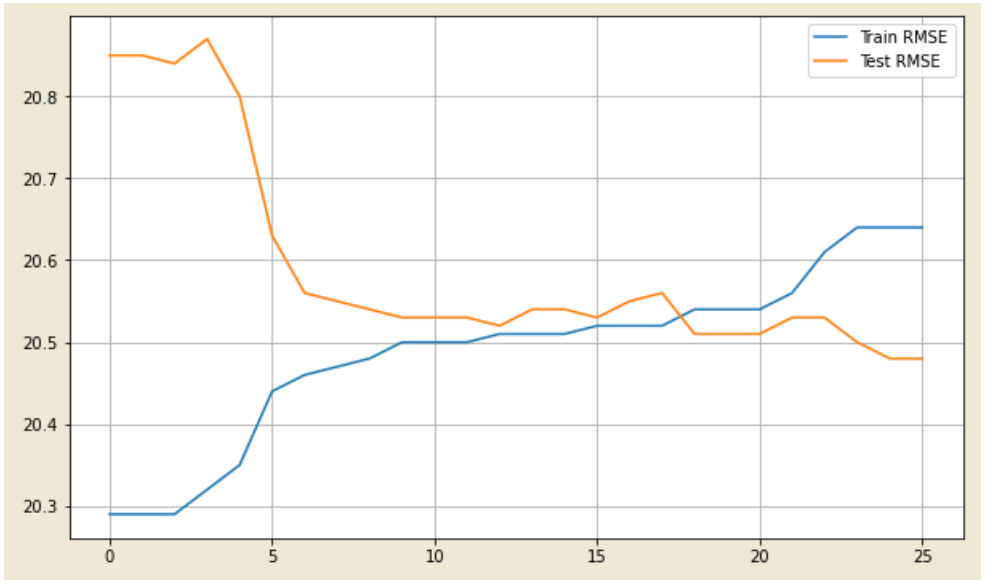
3.6 Feature Selection/Extraction

PCA (Principal Component Analysis)

- 대표적인 차원 축소 기법
- 변수 사이에 존재하는 상관관계를 이용하여 이를 대표하는 주성분 추출

```
from sklearn.decomposition import PCA

Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
m=df.shape[1]-1

for i in range(m):
    pca = PCA(n_components=Train_X_std.shape[1]-i)
    Train_X_std_pca = pca.fit_transform(Train_X_std)
    Test_X_std_pca = pca.fit_transform(Test_X_std)

    LR = LinearRegression()
    LR.fit(Train_X_std_pca, Train_Y)

    pred1 = LR.predict(Train_X_std_pca)
    pred2 = LR.predict(Test_X_std_pca)

    Trr.append(round(np.sqrt(mean_squared_error(Train_Y, pred1)),2))
    Tss.append(round(np.sqrt(mean_squared_error(Test_Y, pred2)),2))
```

```
plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
plt.legend()
plt.grid()
plt.show()
```



2.4

3.6 Feature Selection/Extraction

가장 결과가 좋았던 RFE 사용하여 변수 선택

```
lm = LinearRegression()
rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-df.shape[1]+10)
rfe = rfe.fit(Train_X_std, Train_Y)

LR = LinearRegression()
LR.fit(Train_X_std.loc[:,rfe.support_], Train_Y)

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

print(np.sqrt(mean_squared_error(Train_Y, pred1)))
print(np.sqrt(mean_squared_error(Test_Y, pred2)))
```

```
20.318048532828218
20.388036077109874
```

이 노트북에서는 해결방법을 소개만 하고 뒤의 모델링 과정에는 실제로 적용 X
(Advanced ML Algorithms는 다중공선성을 처리하므로 skip한다고..)

3.7 Predictive Modeling

각 모델을 평가하는 과정에 Evaluate()라는 함수를 미리 만들어서 사용

```
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'])
rc=np.random.choice(Train_X_std.loc[:,Train_X_std.nunique()>=50].columns.values,3,replace=False)
def Evaluate(n, pred1,pred2):
    #Plotting predicted predicteds alongside the actual datapoints
    plt.figure(figsize=[15,6])
    for e,i in enumerate(rc):
        plt.subplot(2,3,e+1)
        plt.scatter(y=Train_Y, x=Train_X_std[i], label='Actual')
        plt.scatter(y=pred1, x=Train_X_std[i], label='Prediction')
        plt.legend()
    plt.show()

    #Evaluating the Multiple Linear Regression Model

    print('\n\n{}Training Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Training set --->',round(r2_score(Train_Y, pred1),20))
    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))

    print('\n{}Testing Set Metrics{}'.format('-'*20, '-'*20))
    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)),
    print('\n{}Residual Plots{}'.format('-'*20, '-'*20))
```

- 빈 Model_Evaluation_Comparison_Matrix (나중에 score 비교하기 위함)
- Numerical 변수 3개 랜덤으로 뽑기

- n = 몇 번째 모델인지
- pred1 = RLR.predict(Train_X_std)
- pred2 = RLR.predict(Test_X_std)

- 랜덤으로 뽑은 numerical 변수 3개 실제 train_Y와 pred1 비교하는 plot
- Train, test에 대해 R2, RSS, MSE, RMSE 출력

EWHA EURON

3.7 Predictive Modeling

```
Model_Evaluation_Comparison_Matrix.loc[n,'Train-R2']  = round(r2_score(Train_Y, pred1),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Test-R2']   = round(r2_score(Test_Y, pred2),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Train-RSS'] = round(np.sum(np.square(Train_Y-pred1)),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Test-RSS']  = round(np.sum(np.square(Test_Y-pred2)),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Train-MSE'] = round(mean_squared_error(Train_Y, pred1),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Test-MSE']  = round(mean_squared_error(Test_Y, pred2),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Train-RMSE']= round(np.sqrt(mean_squared_error(Train_Y, pred1)),20)
Model_Evaluation_Comparison_Matrix.loc[n,'Test-RMSE'] = round(np.sqrt(mean_squared_error(Test_Y, pred2)),20)

# Plotting y_test and y_pred to understand the spread.
plt.figure(figsize=[15,4])

plt.subplot(1,2,1)
sns.distplot((Train_Y - pred1))
plt.title('Error Terms')
plt.xlabel('Errors')

plt.subplot(1,2,2)
plt.scatter(Train_Y,pred1)
plt.plot([Train_Y.min(),Train_Y.max()],[Train_Y.min(),Train_Y.max()], 'r--')
plt.title('Test vs Prediction')
plt.xlabel('y_test')
plt.ylabel('y_pred')
plt.show()
```

- Model_Evaluation_Comparison_Matrix에 R2, RSS, MSE, RMSE 기록
- Error = Train_Y – pred1 plot
- Train_Y vs. pred1 plot

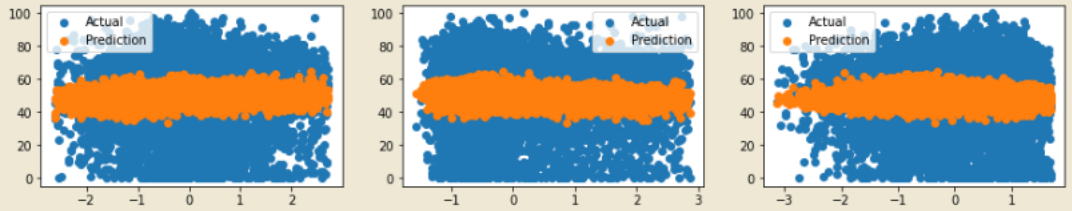
EWHA EURON

3.7 Predictive Modeling

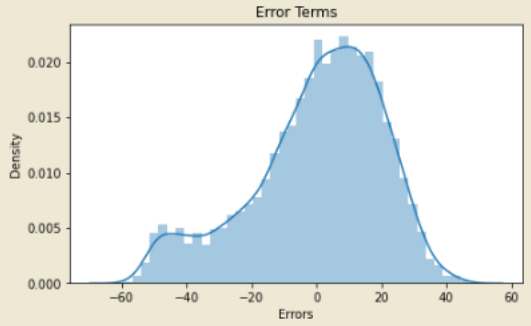
Multiple Linear Regression(MLR)

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

print('{}{}\033[1m Evaluating Multiple Linear Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3)
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(0, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Multiple Linear Regression Model ----------------------------
------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.77439477e-01 -1.59848127e+00  1.75996348e+00 -2.584
37798e+00
 -3.14928011e-01 -6.16275715e-01  2.87342467e+00  4.39914936e-01
 -3.53957449e-01 -1.25286924e-01 -1.48119052e+00  1.18596701e+13
  4.31843933e+13  4.89743105e+13  2.09853482e+13  4.58759689e-01
 -5.83618450e-01 -1.62755702e-01  1.58490417e-01 -1.77115102e-01
  2.34941197e-01 -5.72504408e-01 -1.29839874e-01 -5.00125193e-01
  2.56854264e-01  3.37706544e-01]
The Intercept of the Regresion Model was found to be  50.28469640445756
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.03718662357286395
Residual Sum of Squares (RSS) on Training set  ---> 2946927.532202551
Mean Squared Error (MSE) on Training set       ---> 411.5820575701887
Root Mean Squared Error (RMSE) on Training set ---> 20.28748524510095

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.003411394056330397
Residual Sum of Squares (RSS) on Training set  ---> 745120.9201589487
Mean Squared Error (MSE) on Training set       ---> 416.2686704798596
Root Mean Squared Error (RMSE) on Training set ---> 20.402663318298902

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

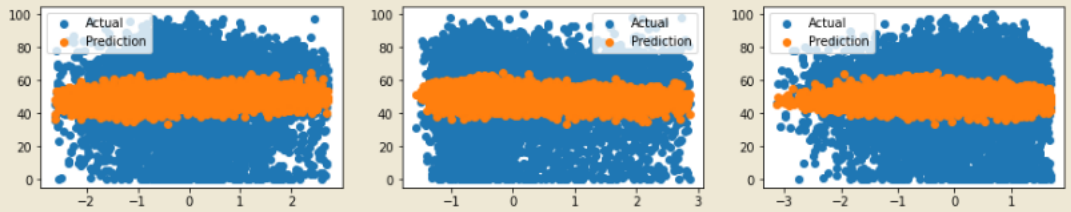


3.7 Predictive Modeling

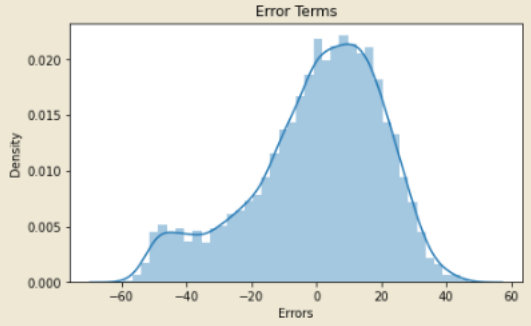
Ridge Regression Model

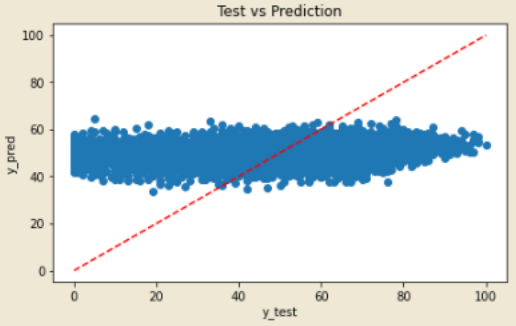
```
RLR = Ridge().fit(Train_X_std,Train_Y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Ridge Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(1, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Ridge Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.77439477e-01 -1.59848127e+00  1.75996348e+00 -2.584
37798e+00
 -3.14928011e-01 -6.16275715e-01  2.87342467e+00  4.39914936e-01
 -3.53957449e-01 -1.25286924e-01 -1.48119052e+00  1.18596701e+13
  4.31843933e+13  4.89743105e+13  2.09853482e+13  4.58759689e-01
 -5.83618450e-01 -1.62755702e-01  1.58490417e-01 -1.77115102e-01
  2.34941197e-01 -5.72504408e-01 -1.29839874e-01 -5.00125193e-01
  2.56854264e-01  3.37706544e-01]
The Intercept of the Regresion Model was found to be  50.28469640445756
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.037189091772393934
Residual Sum of Squares (RSS) on Training set  ---> 2946919.977669837
Mean Squared Error (MSE) on Training set       ---> 411.5810024678543
Root Mean Squared Error (RMSE) on Training set ---> 20.28745924131098

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.0034207109152710746
Residual Sum of Squares (RSS) on Training set  ---> 745113.9542088411
Mean Squared Error (MSE) on Training set       ---> 416.2647788876207
Root Mean Squared Error (RMSE) on Training set ---> 20.40256794836426

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



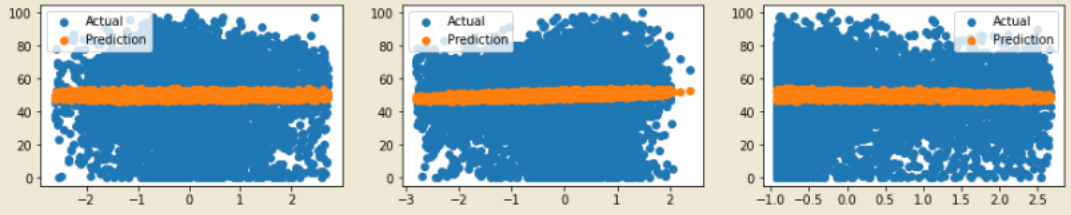


3.7 Predictive Modeling

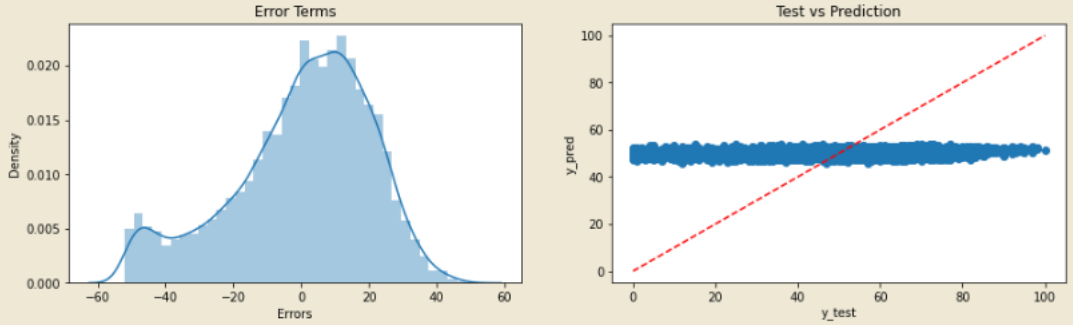
Lasso Regression Model

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

print('{}{}\033[1m Evaluating Lasso Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(2, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Lasso Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.77439477e-01 -1.59848127e+00  1.75996348e+00 -2.584
37798e+00
 -3.14928011e-01 -6.16275715e-01  2.87342467e+00  4.39914936e-01
 -3.53957449e-01 -1.25286924e-01 -1.48119052e+00  1.18596701e+13
  4.31843933e+13  4.89743105e+13  2.09853482e+13  4.58759689e-01
 -5.83618450e-01 -1.62755702e-01  1.58490417e-01 -1.77115102e-01
  2.34941197e-01 -5.72504408e-01 -1.29839874e-01 -5.00125193e-01
  2.56854264e-01  3.37706544e-01]
The Intercept of the Regresion Model was found to be  50.28469640445756
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.015140165780251126
Residual Sum of Squares (RSS) on Training set  ---> 3014406.14752641
Mean Squared Error (MSE) on Training set       ---> 421.00644518525286
Root Mean Squared Error (RMSE) on Training set ---> 20.51844158763654

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.007117147140071945
Residual Sum of Squares (RSS) on Training set  ---> 742350.234109388
Mean Squared Error (MSE) on Training set       ---> 414.72080117842904
Root Mean Squared Error (RMSE) on Training set ---> 20.364694968951266

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

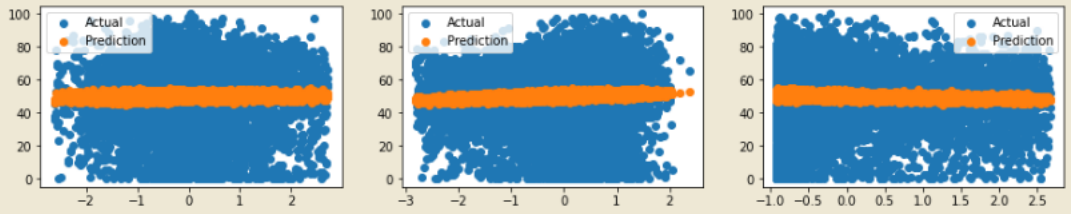


3.7 Predictive Modeling

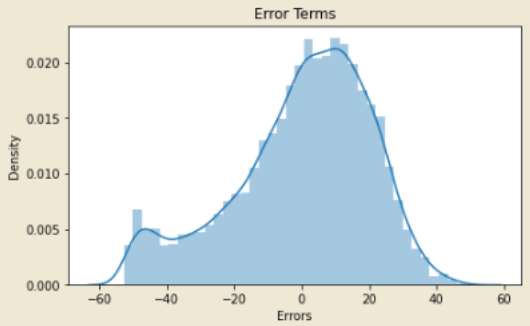
Elastic-Net Regression

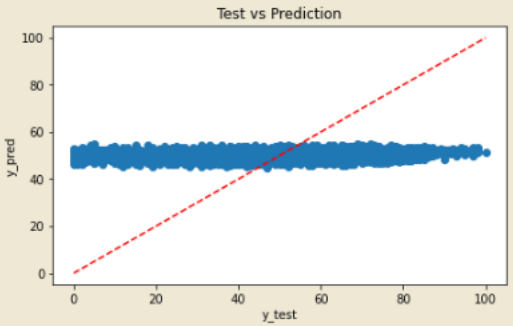
```
ENR = ElasticNet().fit(Train_X_std,Train_Y)
pred1 = ENR.predict(Train_X_std)
pred2 = ENR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Elastic-Net Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(3, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Elastic-Net Regression Model ----------------------------------
--->>>

The Coeffecient of the Regresion Model was found to be  [ 6.77439477e-01 -1.59848127e+00  1.75996348e+00 -2.584
37798e+00
 -3.14928011e-01 -6.16275715e-01  2.87342467e+00  4.39914936e-01
 -3.53957449e-01 -1.25286924e-01 -1.48119052e+00  1.18596701e+13
  4.31843933e+13  4.89743105e+13  2.09853482e+13  4.58759689e-01
 -5.83618450e-01 -1.62755702e-01  1.58490417e-01 -1.77115102e-01
  2.34941197e-01 -5.72504408e-01 -1.29839874e-01 -5.00125193e-01
  2.56854264e-01  3.37706544e-01]
The Intercept of the Regresion Model was found to be  50.28469640445756
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.019205092299083293
Residual Sum of Squares (RSS) on Training set  ---> 3001964.438501573
Mean Squared Error (MSE) on Training set       ---> 419.26877632703537
Root Mean Squared Error (RMSE) on Training set ---> 20.476053729345296

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.007624464892474702
Residual Sum of Squares (RSS) on Training set  ---> 741970.9270731356
Mean Squared Error (MSE) on Training set       ---> 414.50889780622094
Root Mean Squared Error (RMSE) on Training set ---> 20.35949159007221

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

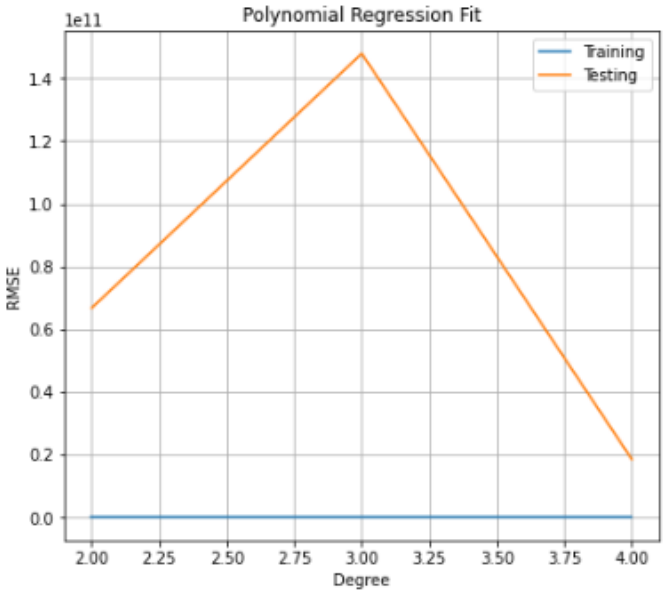


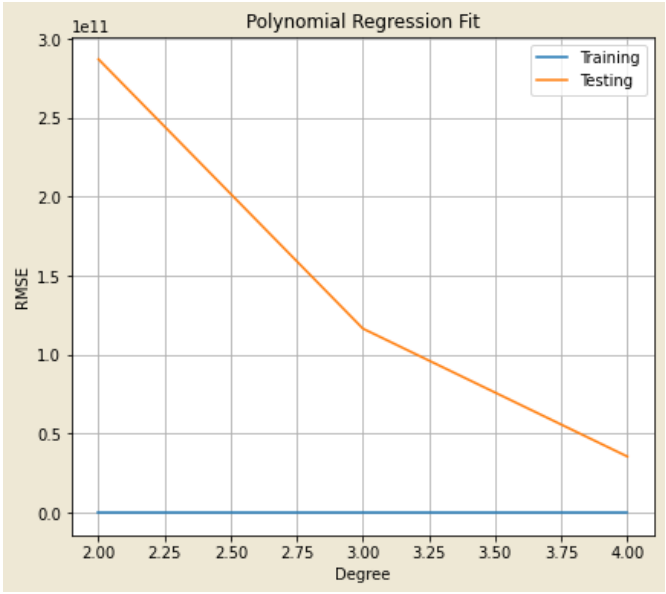


3.7 Predictive Modeling

Polynomial Regression Model

```
Trr=[]; Tss=[]
n_degree=5

for i in range(2,n_degree):
    poly_reg = PolynomialFeatures(degree=i)
    X_poly = poly_reg.fit_transform(Train_X_std)
    X_poly1 = poly_reg.fit_transform(Test_X_std)
    LR = LinearRegression()
    LR.fit(X_poly, Train_Y)

    pred1 = LR.predict(X_poly)
    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))

    pred2 = LR.predict(X_poly1)
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

plt.figure(figsize=[15,6])
plt.subplot(1,2,1)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
plt.title('Polynomial Regression Fit')
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
plt.show()
```





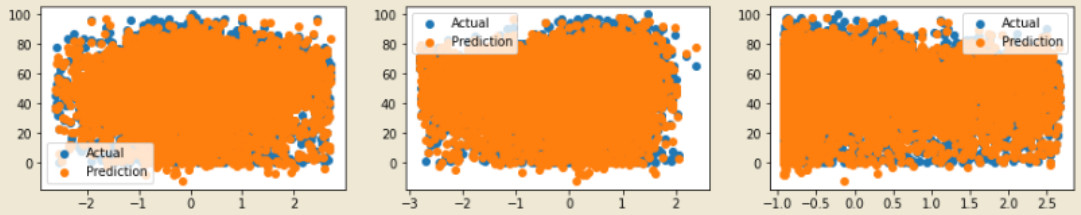
Degree = 4

3.7 Predictive Modeling

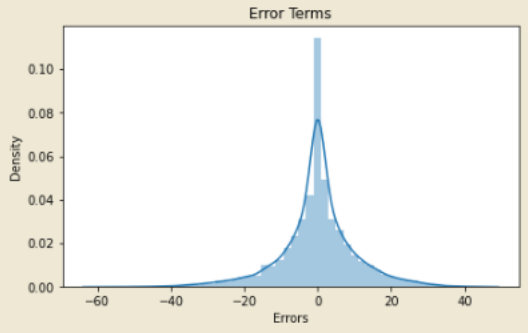
Polynomial Regression Model

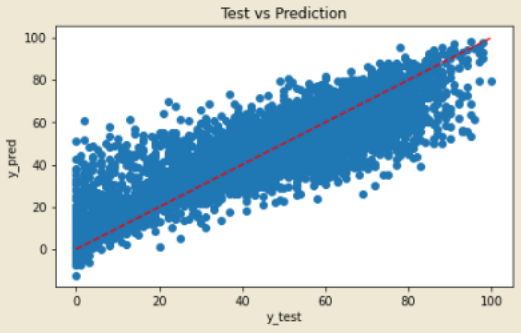
```
poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(Train_X_std)
X_poly1 = poly_reg.fit_transform(Test_X_std)
PR = LinearRegression()
PR.fit(X_poly, Train_Y)

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

print('{}{}\033[1m Evaluating Polynomial Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(4, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Polynomial Regression Model ----------------------------------
-->>>

The Coeffecient of the Regresion Model was found to be  [ 6.77439477e-01 -1.59848127e+00  1.75996348e+00 -2.584
37798e+00
 -3.14928011e-01 -6.16275715e-01  2.87342467e+00  4.39914936e-01
 -3.53957449e-01 -1.25286924e-01 -1.48119052e+00  1.18596701e+13
  4.31843933e+13  4.89743105e+13  2.09853482e+13  4.58759689e-01
 -5.83618450e-01 -1.62755702e-01  1.58490417e-01 -1.77115102e-01
  2.34941197e-01 -5.72504408e-01 -1.29839874e-01 -5.00125193e-01
  2.56854264e-01  3.37706544e-01]
The Intercept of the Regresion Model was found to be  50.28469640445756
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.7445835594698369
Residual Sum of Squares (RSS) on Training set  ---> 781764.9392955602
Mean Squared Error (MSE) on Training set       ---> 109.18504738764808
Root Mean Squared Error (RMSE) on Training set ---> 10.449164913410453

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> -3.0050318847087145e+18
Residual Sum of Squares (RSS) on Training set  ---> 2.2467767639395423e+24
Mean Squared Error (MSE) on Training set       ---> 1.255182549686895e+21
Root Mean Squared Error (RMSE) on Training set ---> 35428555568.734314

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





3.7 Predictive Modeling

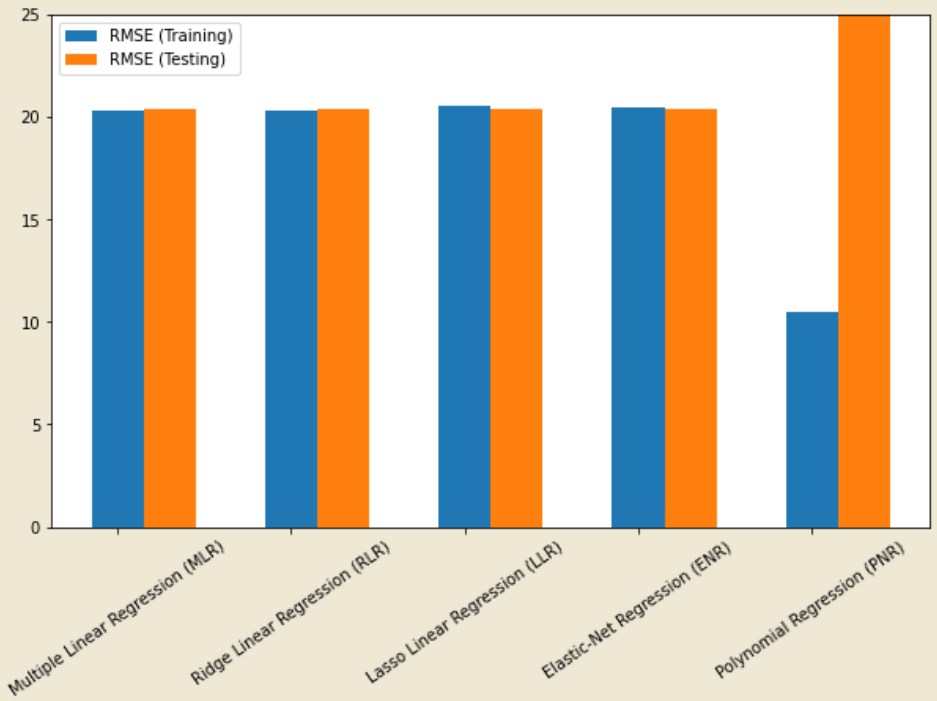
```
EMC = Model_Evaluation_Comparison_Matrix.copy()
EMC.index = ['Multiple Linear Regression (MLR)','Ridge Linear Regression (RLR)','Lasso Linear Regression (LLR)',
             'Elastic-Net Regression (ENR)','Polynomial Regression (PNR)']
EMC
```

	Train-R2	Test-R2	Train-RSS	Test-RSS	Train-MSE	Test-MSE	Train-RMSE	Test-RMSE
Multiple Linear Regression (MLR)	0.037187	3.411394e-03	2.946928e+06	7.451209e+05	411.582058	4.162687e+02	20.287485	2.040266e+01
Ridge Linear Regression (RLR)	0.037189	3.420711e-03	2.946920e+06	7.451140e+05	411.581002	4.162648e+02	20.287459	2.040257e+01
Lasso Linear Regression (LLR)	0.015140	7.117147e-03	3.014406e+06	7.423502e+05	421.006445	4.147208e+02	20.518442	2.036469e+01
Elastic-Net Regression (ENR)	0.019205	7.624465e-03	3.001964e+06	7.419709e+05	419.268776	4.145089e+02	20.476054	2.035949e+01
Polynomial Regression (PNR)	0.744584	-3.005032e+18	7.817649e+05	2.246777e+24	109.185047	1.255183e+21	10.449165	3.542856e+10

Polynomial Regression Model에서 과적합 발생
ENR이 가장 결과가 좋지만 거의 비슷함

3.7 Predictive Modeling

```
cc = Model_Evaluation_Comparison_Matrix.columns.values
s=5
plt.bar(np.arange(5), Model_Evaluation_Comparison_Matrix[cc[6]].values, width=0.3, label='RMSE (Training)')
plt.bar(np.arange(5)+0.3, Model_Evaluation_Comparison_Matrix[cc[7]].values, width=0.3, label='RMSE (Testing)')
plt.xticks(np.arange(5),EMC.index, rotation =35)
plt.legend()
plt.ylim([0,25])
plt.show()
```



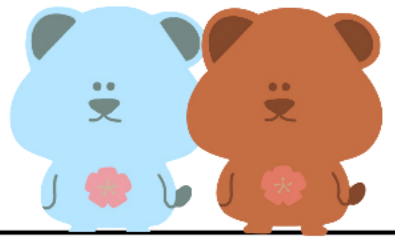
3.8 Conclusion

Feature engineering
- 결과적으로 많은 데이터가 drop되고 47.5%만을 사용

다중공선성
- VIF, RFE, PCA를 통해 해결했고
- 가장 결과가 좋은 RFE 사용

모델링
- Evaluate() 함수 미리 만들어서 각 모델 평가
- MLR, 릿지, 라쏘, 엘라스틱넷, Polynomial
- Polynomial regression은 과적합 발생

THANK YOU



EWHA EURON