Prediction for House prices

Regression Analysis

Hailey (Hwy-Kyung) Song

Process

001

Linear

Random Forest

XGBoost

Decision Tree

Light GBM

Gradient Boosting

NGBoost

Model Evaluation

: Evaluate base Models

002

Categorical vs Numeric

EDA : CountPlot / ScatterPlot

Heatmap / DistPlot / Probplot

Discretization (이산화)

Duplicate Data (중복) 제거

Missing Data (결측치)

Outlier (이상치) 대체

Nomalization (정규화)

Multicollinearity (변수 선택/차원 축소)

Aggregation (데이터 요약 / 파생 변수)

Data Analysis with EDA

: Preprocessing

003

X Train y Train X Test y Test

Train Models (X) → Predict y

Variety of Algorithms

Model Tuning

Ensemble

- Random Forest
- XGBoost
- NGBoost

회귀 모델 성능 평가 (지표)

Modeling

: Make / Select Models

004

Model. Predict (X)

?

Performance Evaluation

: Measure Accuracy

Data.Columns

id : 데이터 고유 id

OverallQual : 전반적 재료와 마감 품질

이산화

KitchenQual : 부엌 품질

ExterQual : 외관 재료 품질

BsmtQual : 지하실 높이

FullBath : 지상층 화장실 개수

GarageCars: 차고 자리 개수

VS

GarageYrBlt : 차고 완공 연도

GarageArea: 차고 면적

YearBuilt : 완공 연도

YearRemodAdd : 리모델링 연도

TotalBsmtSF : 지하실 면적

1stFlrSF : 1층 면적

GrLivArea : 지상층 생활 면적

target : 집값(달러 단위)

Analytical Thinking

[Categorical Data]

6 / 13

Overall Qual | Garage Cars | Full Bath | Exter Qual(외관품질) | Kitchen Qual | Bsmt Qual(지하실 높이)

[Numeric Data]

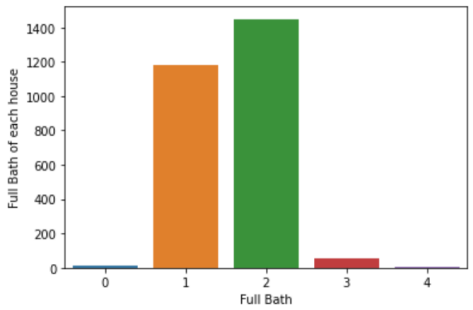
7 / 13

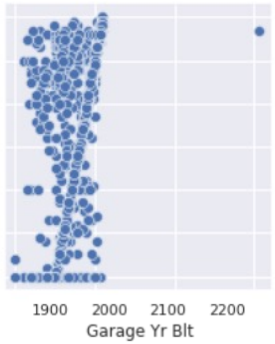
Gr Liv Area | Garage Area | Total Bsmt SF | 1st Flr SF | Year Built | Year Remod/Add | Garage Yr Blt

```
train[['Overall Qual','Exter Qual','Kitchen Qual','Bsmt Qual']].describe()
```

	Overall Qual	Exter Qual	Kitchen Qual	Bsmt Qual
count	1349.000000	1349.000000	1349.000000	1349.000000
mean	6.208302	3.426242	3.556709	3.606375
std	1.338338	0.573856	0.663209	0.694395
min	2.000000	2.000000	2.000000	1.000000
25%	5.000000	3.000000	3.000000	3.000000
50%	6.000000	3.000000	3.000000	4.000000
75%	7.000000	4.000000	4.000000	4.000000
max	10.000000	5.000000	5.000000	5.000000

```
Text(0, 0.5, 'Full Bath of each house')
```






```
all[all['Garage Yr Blt'] > 2022]
```

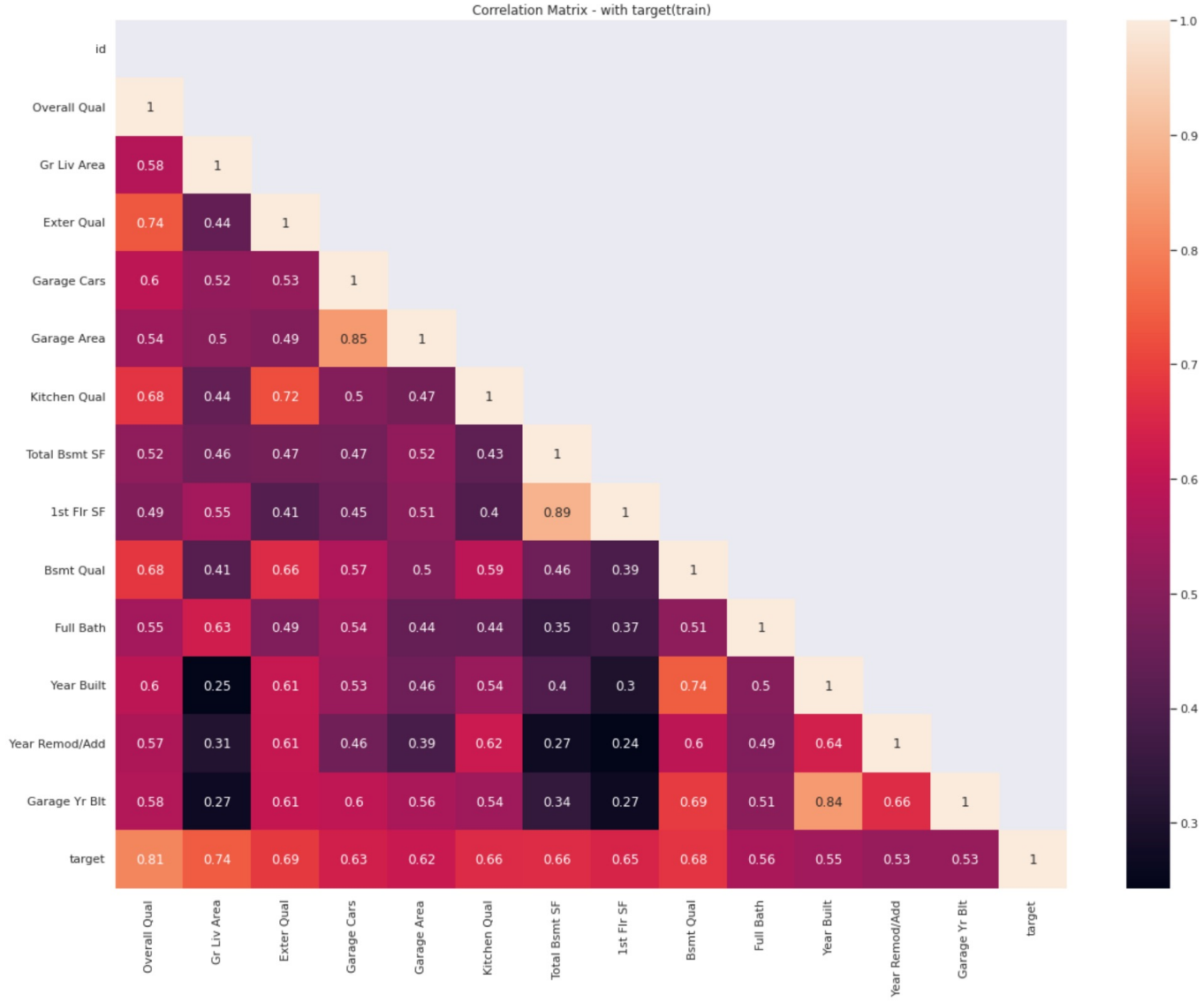
	id	Overall Qual	Gr Liv Area	Exter Qual	Garage Cars	Garage Area	Kitchen Qual	Total Bsmt SF	1st Flr SF	Bsmt Qual	Full Bath	Year Built	Year Remod/Add	Garage Yr Blt	target
254	255	8	1564	4	2	502	5	1546	1564	4	2	2006	2007	2207	267300.0

```
all[all['Year Built'] > all['Garage Yr Blt']] #(주차장 먼저 세워짐)
```

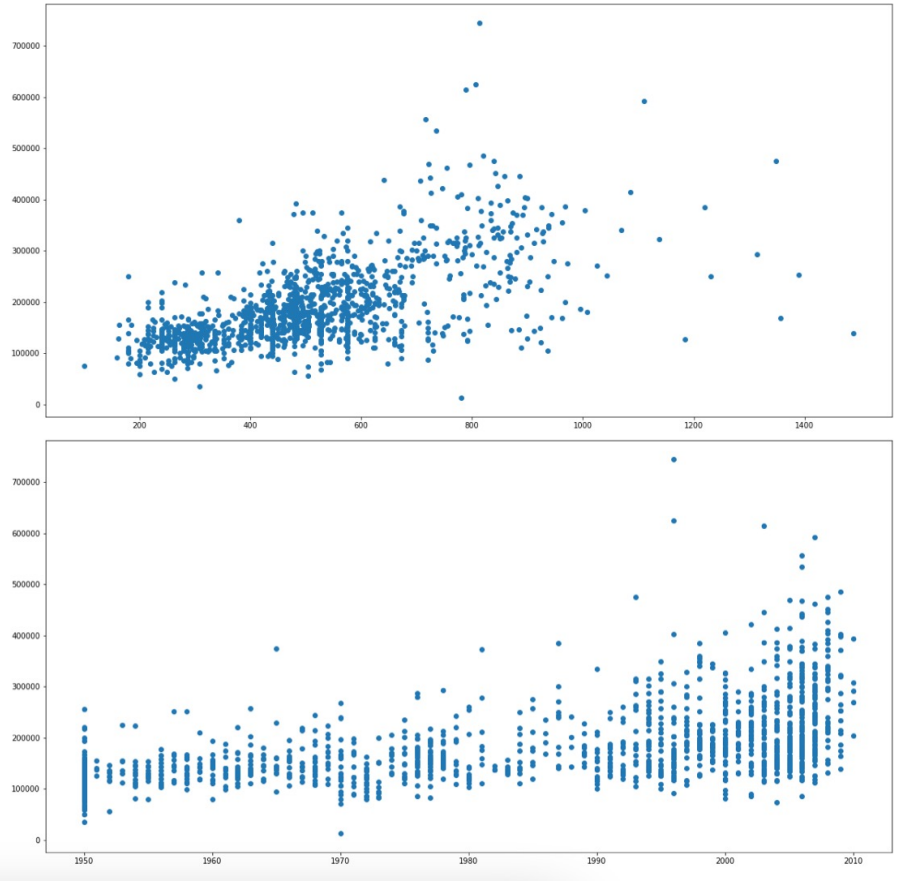
	id	Overall Qual	Gr Liv Area	Exter Qual	Garage Cars	Garage Area	Kitchen Qual	Total Bsmt SF	1st Flr SF	Bsmt Qual	Full Bath	Year Built	Year Remod/Add	Garage Yr Blt	target
13	14	6	1242	3	1	180	3	583	647	3	1	1935	1950	1926	105000.0
198	199	6	1194	3	2	539	3	655	1194	3	1	1959	1959	1954	159500.0
203	204	6	1378	3	1	162	4	901	861	3	1	1935	1998	1920	128900.0
415	416	8	1904	4	2	736	5	1058	1058	4	2	2005	2005	2003	275000.0
438	439	4	520	3	1	240	2	520	520	3	1	1927	1950	1920	68500.0
442	443	6	790	3	1	160	3	768	790	4	1	1930	1950	1925	91000.0
560	561	5	1298	3	1	256	3	720	854	3	1	1941	1950	1940	122000.0
699	700	8	1500	4	2	674	4	1489	1500	4	2	2006	2006	2005	212999.0
732	733	9	2338	5	3	1110	4	2660	2338	5	2	2006	2007	2005	591587.0
1087	1088	7	2978	4	2	564	4	710	1898	4	2	1967	2007	1961	242000.0
1119	1120	6	1848	3	2	370	4	833	1053	3	1	1923	2000	1922	207000.0

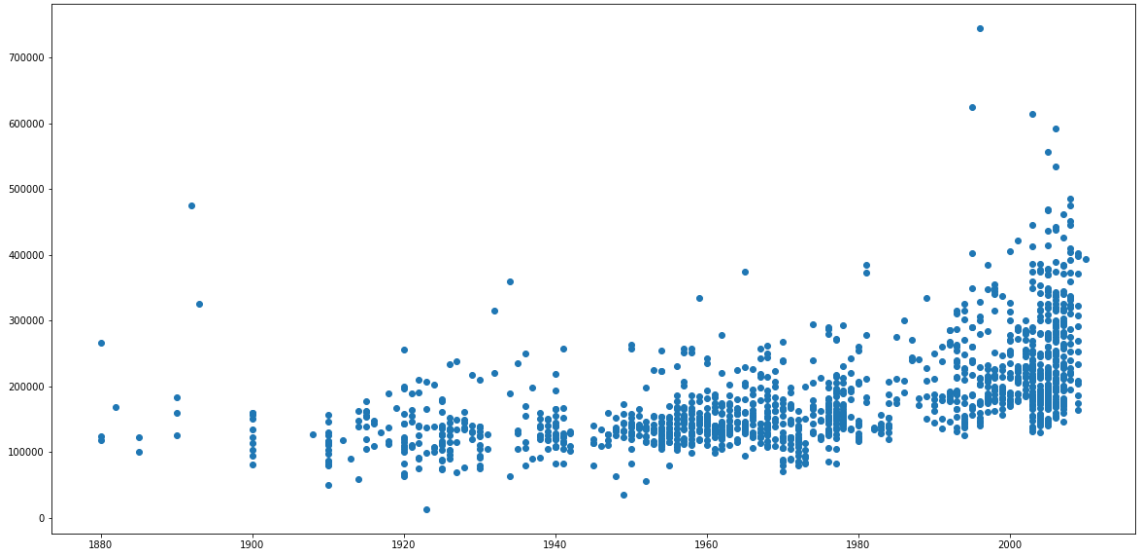
HeatMap

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f036bb79190>
```



Scatter Plot





Data Preprocessing

<table>
<tr><th>Missing/Duplicate Data</th><th>Outliers</th><th>Derived variables
(Correlations)</th><th>Standardization
/ Conversion
(continuous/asymmetric data)</th></tr>
<tr><td>df.isnull.sum()</td><td>Rows with extremely large or very small values in the graph.
Parking lot establishment year 2207 -> 2007</td><td>KitchenQual
ExterQual
Bsmt Qual</td><td>Robust Scaler
Standard Scaler</td></tr>
<tr><td>df.duplicated().sum()</td><td>Houses without bathroom
(Full Bath == 0)</td><td>Garage Area
Garage Car</td><td rowspan="2">Gr Liv Area / Garage Area
Total Bsmt SF / 1st Flr SF
Year Built / Year Remod/Add
Garage Yr Blt</td></tr>
<tr><td rowspan="3">No missing data
Removed 1 duplicate value</td><td>Remodeling Year < House Establishment Year</td><td>1st Flr SF
Gr Liv Area</td></tr>
<tr><td>Parking Lot Establishment Year < House Establishment Year</td><td rowspan="2">Summerize Quality variable
Parking area per car
(Number of floors ,except 1st floor) / Area</td><td>Log transformation</td></tr>
<tr><td>Remove or Replace (To average value)</td><td>target</td></tr>
</table>

Code for Preprocessing

Summary

```
[9]  train.reset_index(inplace=True)
     test.reset_index(inplace=True)

     kind = ['Exter Qual', 'Kitchen Qual', 'Bsmt Qual']
     Qual = ['None', 'Po', 'Fa', 'TA', 'Gd', 'Ex']

     def disc(df):
       for k in range(0,3):
          for q in range(0,6):
            df = df.replace({kind[k]:{Qual[q]:q}})
       return df

     train = disc(train)
     test = disc(test)
```

Discretization

```
tmp = all[all['Year Built'] <= all['Garage Yr Blt']]
print((tmp['Garage Yr Blt'].sum() - tmp['Year Built'].sum()) / 1350)
```

```
5.15962962962963
```

```
[49] train[train['Garage Yr Blt'] > 2022]
     train['Garage Yr Blt'] = np.where(train['Garage Yr Blt'] > 2021, 2007, train['Garage Yr Blt'])
```

```
[50] # Full Bath - Gr Liv Area 상관관계 | Full Bath == 0
     train['Full Bath'] = np.where((train['Full Bath'] == 0), (train['Gr Liv Area']/875).astype(int), train['Full Bath'])
     # Year Built' > Garage Yr Blt
     train['Garage Yr Blt'] = np.where(train['Year Built'] > train['Garage Yr Blt'], train['Year Built']+5, train['Garage Yr Blt'])
     # Year Remod/Add < Year Built
     train['Year Remod/Add'] = np.where(train['Year Remod/Add'] < train['Year Built'], train['Year Remod/Add']+12, train['Year Remod/Add'])
```

Outliers

```
# kitchen Qual / Bsmt Qual / Exter Qual 병합
# 1개 Car Area = Garage Area / Garage Cars
# 1층 외 층 여부

def drived_var(df):
  df['EKB_Qual'] = df[["Exter Qual", "Kitchen Qual", "Bsmt Qual"]].sum(axis=1)
  df['Car Area'] = df['Garage Area']/df['Garage Cars']
  df['A_flr SF'] = df['Gr Liv Area'] - df['1st Flr SF']
  df['A_flr'] = (df['A_flr SF']).apply(lambda x : 1 if x > 0 else 0)
  return df

train = drived_var(train)
test = drived_var(test)
```

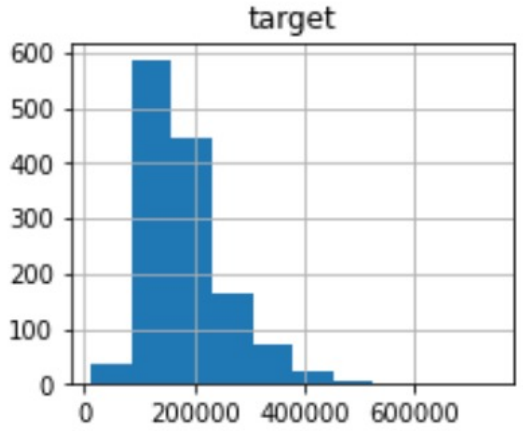
Aggregation | Drived Variables

```
# 로그 변환
train['target'] = np.log1p(train['target'])
```

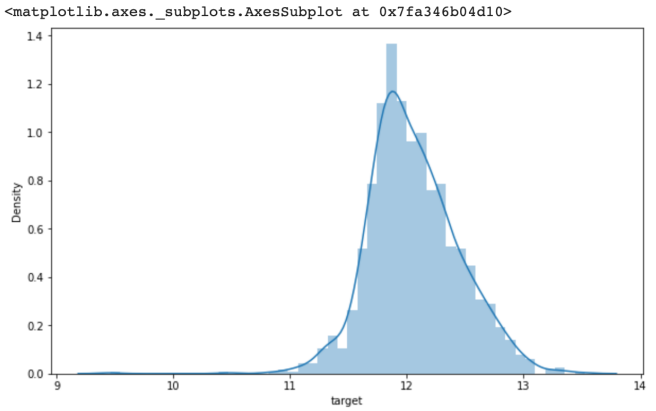
```
from sklearn.preprocessing import StandardScaler, RobustScaler
Scaler = RobustScaler()
df = train[['Gr Liv Area', 'Garage Area', 'Total Bsmt SF', '1st Flr SF', 'Year Built', 'Year Remod/Add', 'Garage Yr Blt']]
Scaler.fit(df)
Scaled_df = Scaler.transform(df)
Scaled_df = pd.DataFrame(data = Scaled_df)
train_ = train.copy()
c = 0
for i in df.columns:
  train[i] = Scaled_df[c]
  c += 1
```

Standardization / Transformation

Transformation Standardization



np.log1p



Standardization for Continuous variables

	Gr Liv Area	Garage Area	Total Bsmt SF	1st Flr SF	Year Built	Year Remod/Add	Garage Yr Blt
0	1.505564	2.200000	2.801619	2.539062	0.574468	0.277778	0.609756
1	-0.147854	-0.081818	0.696356	0.507812	0.638298	0.388889	0.682927
2	-0.866455	-0.890909	-0.291498	-0.375000	-0.191489	-0.722222	-0.268293
3	-0.430843	0.418182	-0.663968	-0.804688	-1.617021	0.361111	0.536585
4	0.815580	2.054545	0.036437	-0.128906	0.617021	0.333333	0.658537
...	...	...	...	...	...	...	...
1344	0.494436	-0.281818	-0.275304	-0.398438	0.425532	0.111111	0.439024
1345	2.071542	1.663636	1.704453	1.480469	0.638298	0.361111	0.682927
1346	-0.367250	-0.754545	0.417004	0.238281	-0.191489	-0.722222	-0.268293
1347	-0.875994	-0.200000	-0.291498	-0.386719	-0.042553	-0.527778	-0.097561
1348	-0.855326	-0.640909	-0.204453	-0.361328	0.042553	-0.416667	0.000000

1349 rows × 7 columns

Modeling - Random Forest, Gradient Boosting, Linear, Ridge, Rasso, XGBoost, NGBoost, LGBM

```
!pip install ngboost
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from xgboost import XGBRegressor
from ngboost import NGBRegressor
from lightgbm import LGBMRegressor

# 8가지 모델
# 알고리즘 선택/튜닝/앙상블

r = RandomForestRegressor(random_state = 100, criterion = 'mae')
r.fit(X_train, y_train)
r_pred = r.predict(X_test)

g = GradientBoostingRegressor(random_state = 100, criterion = 'mae')
g.fit(X_train, y_train)
g_pred = g.predict(X_test)

x = XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                 learning_rate=0.05, max_depth=3,
                 min_child_weight=1.7817, n_estimators=2200,
                 reg_alpha=0.4640, reg_lambda=0.8571,
                 subsample=0.5213, silent=1,
                 random_state =7, nthread = -1)
x.fit(X_train, y_train)
x_pred = x.predict(X_test)
```

```
ln = LinearRegression(normalize=True)
ln.fit(X_train, y_train)
ln_pred = ln.predict(X_test)

rg = Ridge(alpha = 0.0005, random_state=1)
rg.fit(X_train, y_train)
rg_pred = rg.predict(X_test)

ls = Lasso(alpha = 0.0005, random_state=1)
ls.fit(X_train, y_train)
ls_pred = ls.predict(X_test)

l = LGBMRegressor(objective='regression',num_leaves=5,
                              learning_rate=0.05, n_estimators=720,
                              max_bin = 55, bagging_fraction = 0.8,
                              bagging_freq = 5, feature_fraction = 0.2319,
                              feature_fraction_seed=9, bagging_seed=9,
                              min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
l.fit(X_train, y_train)
l_pred = l.predict(X_test)

n = NGBRegressor(random_state = 42, n_estimators = 10000, verbose = 0, learning_rate = 0.03)
n.fit(X_train, y_train)
n_pred = n.predict(X_test)
```

Default Predictive Value

	Random Forest	Grident Boosting	Linear Regression	Ridge	Lasso	XGB	NGB	LGB	target
841	11.930211	11.906294	11.910997	11.910997	11.911970	11.916280	11.915751	11.918088	11.813037
184	12.148831	12.133517	12.241466	12.241466	12.238744	12.141668	12.177420	12.195820	12.205578
526	12.075568	12.076956	12.118633	12.118633	12.120391	12.080716	12.098311	12.090504	12.146859
1282	12.192187	12.178577	12.152280	12.152280	12.152322	12.103448	12.139229	12.116682	12.179220
979	12.362395	12.437160	12.460544	12.460544	12.459218	12.427813	12.437232	12.408230	12.834684
...	...	...	...	...	...	...	...	...	...
619	12.291149	12.235258	12.319168	12.319168	12.321812	12.250286	12.264847	12.351826	12.307595
122	11.776711	11.685675	11.658988	11.658988	11.664424	11.638024	11.655877	11.641650	11.492733
1014	11.362341	11.295154	11.395896	11.395897	11.402447	11.288092	11.269088	11.119206	11.373675
989	12.066314	12.145036	12.084347	12.084347	12.086267	12.080530	12.089825	12.123954	12.289959
982	11.950364	11.971503	12.051677	12.051677	12.049710	11.994369	11.923354	12.020426	11.867104

405 rows × 9 columns

Dacon

Competetion for house prising regression

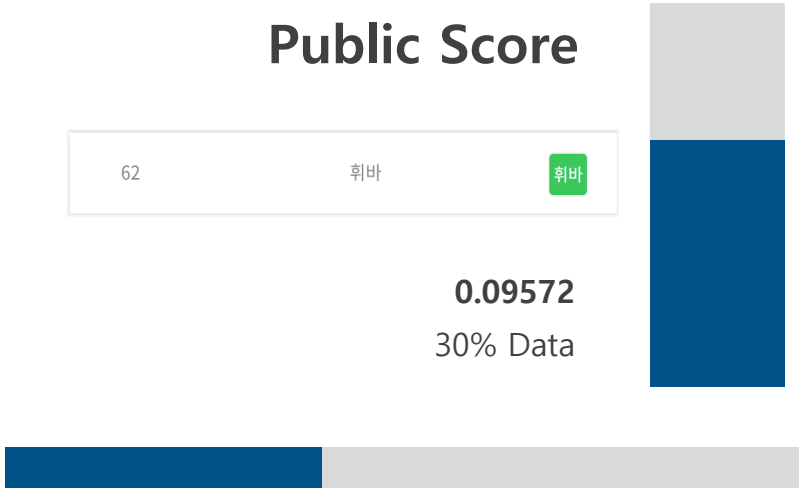
Final Scores

Public Score

62 휘바 휘바

0.09572

30% Data

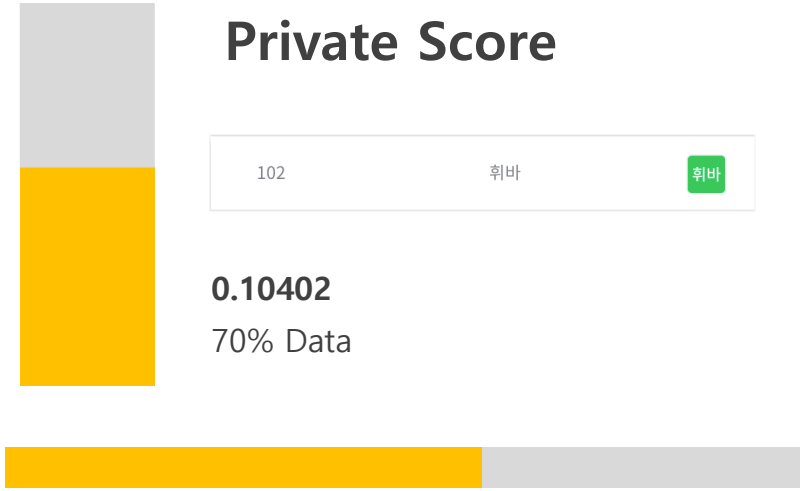


Private Score

102 휘바 휘바

0.10402

70% Data



1. More Data
2. PreProcessing
3. Feature Selection / Extraction (PCA)
4. Multiple algorithms
5. Algorithm Tuning (Grid Search)
6. Cross Validation (K-fold)
7. Ensemble methods (Hyper parameters)

Evaluation Metrics [For Regression]

```
!pip install catboost
from sklearn.metrics import make_scorer
from sklearn.model_selection import KFold
from catboost import Pool

def NMAE(real, pred):
    mae = np.mean(np.abs(real-pred))
    score = mae / np.mean(np.abs(real))
    return score
nmae_score = make_scorer(NMAE, greater_is_better=False)
```

```
0.09825028
0.09774097
0.09841115
0.09657213
0.09483226
0.09660786
0.10013898
```

```
mlist = []
preds = [r_pred, g_pred, ln_pred, rg_pred, ls_pred, x_pred, n_pred, l_pred]
for i in range(0,len(preds)):
  mlist.append(NMAE(np.expm1(y_test), np.expm1(preds[i])))
```

```
for m in mlist :
  print("{:.8f}".format(np.mean(m))) # Gradient / Lasso / x_pred / n_pred
```

Cross Validation (using K-fold)

```
x_train = train.drop(['target'], axis = 1)
y_train = train['target']

x = np.array(x)
y = np.array(y)
test = np.array(test)
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.ensemble import VotingRegressor

kfold = KFold(n_splits = 5, shuffle = True, random_state = 11)

nmae_mean= []
c = 0
rf_result = np.zeros(test.shape[0])

for train_idx, val_idx in kfold.split(x, y):

    x_train, x_val = x[train_idx], x[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    VR.fit(x_train, y_train)
    pred_rf = VR.predict(x_val)

    c += 1
    nmae = NMAE(np.expm1(y_val), np.expm1(pred_rf))

    print('교차 검증+앙상블{0}: {1}'.format(c, nmae))
    nmae_mean.append(nmae)
    rf_result += VR.predict(test)/5

print('=> 총 교차 검증 평균:', np.mean(nmae_mean))
rf_result
```

```
교차 검증+앙상블1: 0.10993029697849206
교차 검증+앙상블2: 0.08818759028716033
교차 검증+앙상블3: 0.08497116990699974
교차 검증+앙상블4: 0.08795038255052674
교차 검증+앙상블5: 0.1045424063312304
=> 총 교차 검증 평균: 0.09511636921088185
array([13.16439108, 12.39026802, 12.58739295, ..., 12.24553341,
       12.55809101, 12.38133247])
```

Grid Search

```
[79] #GridSearchCV by Dacon
     from sklearn.model_selection import GridSearchCV

     params = []
     params_rf = {'n_estimators' : [90, 150, 200, 1000],
                  'min_samples_split' : [1,2,3,4]}
     rf = RandomForestRegressor()
     params.append(params_rf)
     estimators = []
     estimators.append(rf)


     from tqdm.auto import tqdm
     def gridSearchCV(models,params):
         best_models=[]
         for i in tqdm(range(0,len(models))):
             model_grid = GridSearchCV(models[i], params[i],n_jobs = -1, verbose=1, cv=5)
             model_grid.fit(x_train,y_train)
             best_models.append(model_grid.best_estimator_)
         return best_models

     best_model_list = gridSearchCV(estimators,params)
     best_model_list
```

```
100% 1/1 [01:42<00:00, 102.58s/it]
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[RandomForestRegressor(min_samples_split=4, n_estimators=1000)]
```

```
# jujukwakwkak님 코드 참조
import plotly.express as px
from sklearn.decomposition import PCA

train_x = train.drop(['id'], axis=1)

train_x = disc(train_x)

Scaler.fit(train_x)
Scaled_df = Scaler.transform(train_x)
train_x = pd.DataFrame(data = Scaled_df)

pca = PCA()
pca.fit(train_x)
exp_var_cumul = np.cumsum(pca.explained_variance_ratio_)

px.area(x=range(1, exp_var_cumul.shape[0] + 1),
        y=exp_var_cumul,
        labels={'x':'# Components', 'y':'Explained Variance'})
```

```
pca = PCA(n_components = 8)
pca.fit_transform(x_train)
```

PCA (Select variables)

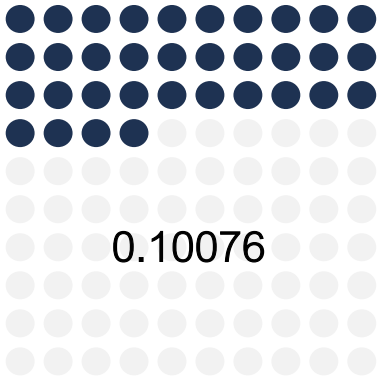
Ensemble

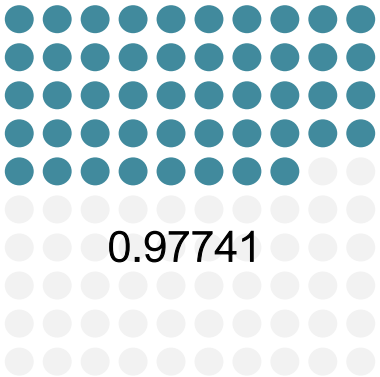
```
f1 = n.predict(test)
f2 = x.predict(test)
f3 = ls.predict(test)
```

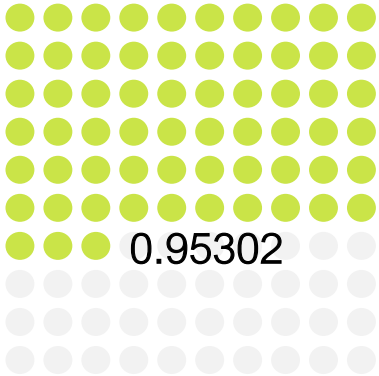
```
sub = pd.read_csv('sample_data/sample_submission.csv')
sub['target'] = np.expm1((f1 + f2 + f3) / 3)
sub.to_csv(('submission.csv'), index=False)
```

Preprocessing / Modeling

Forcatsting accuracy trend (NMAE)







34%

Basic Modeling

48%

After Preprocessing

63%

Advanced Modeling

Thank you for your attention