House Price Prediction

GR5293 EODS – Project 2
Contributer: Lichun He / Yuqi Zhang / Haishan Mei / Lei Huang

Outline

- What's New
- Dataset Overview
- Data Cleaning
- Feature Selection
- Model Selection
- Model Evaluation
- Summary

What's New

Project 1

- Data Cleaning
- Feature Selection
    - LASSO
- Model Selection
    - based on LASSO

Project 2

- Data Cleaning (improved)
- Feature Selection (improved)
    - LASSO
    - Tree Based Model Feature Importance
- Model Selection (huge improved)
    - LASSO / Tree / GradiendBoost /...
    - Tuning process
- Model Evaluation

Dataset Overview

- House Prices dataset generated from Kaggle[1]
  - 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa;
  - 1 target variable - SalePrice
- Pre-split two dataset
  - Train dataset: (1460, 80) - with ‘SalePrice’
  - Test dataset: (1459, 79) - without ‘SalePrice’

```
df_train.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
Id																					
1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 80 columns

1: https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

Dataset Overview

- Mixture of numeric and categorical variables
  - Nominal variable
    - RoofStyle: Type of roof

```
Flat        Flat
Gable       Gable
Gambrel     Gabrel (Barn)
Hip         Hip
Mansard     Mansard
Shed        Shed
```

  - Ordinal variable
    - OverallQual: Rates the overall material and finish of the house

```
10          Very Excellent
9           Excellent
...
2           Poor
1           Very Poor
```

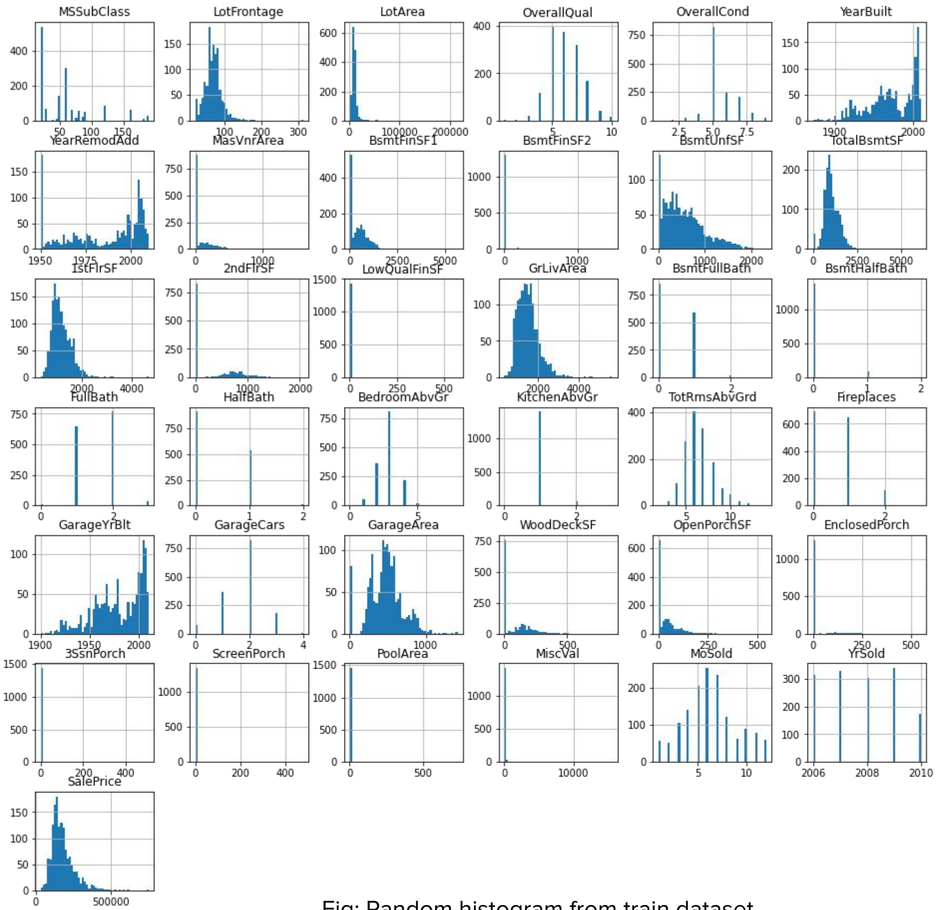


Fig: Random histogram from train dataset

Data Cleaning

Step 0: Extra target variable and drop it from train data;
Merge pre-split datasets together into our self-use train dataset

```
df_train = pd.concat((X, df_test))
df_train.shape
```

```
(2919, 79)
```

Step 1: Check duplicate data

```
df_train[df_train.duplicated(keep='first')]
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...
Id											

0 rows × 80 columns

✅ Our dataset doesn't have duplicate data.

Data Cleaning

Step 2: Check missing data

```
def Missingtable(df,missing_col_name ='Missing Data Count'):
    missing_df = pd.DataFrame(df.isnull().sum(), columns=[missing_col_name])
    return missing_df[missing_df[missing_col_name]!=0].sort_values(missing_col_name,ascending=True)

print('Number of features having missing values: {}'.format(Missingtable(df_train).shape[0]))
```

```
Number of features having missing values: 19
```

📌 We find that there exists missing datas in our datasets.

However, after taking a closer look into our datas, we find that some NA actually has its meanings. i.e. :

```
Alley: Type of alley access to property

       Grvl     Gravel
       Pave     Paved
       NA       No alley access
```

Data Cleaning

Step 2: Check missing data (cont.)

- Manually handle missing values
    - Based on the features' defination
        - Fill NA with different specific words

```
# Alley : data description says NA means "no alley access"
df_train.loc[:, "Alley"] = df_train.loc[:, "Alley"].fillna("None")
```

        - Convert some "numerical" features into categorical features

```
df_train.replace({"MSSubClass" : {20 : "SC20", 30 : "SC30", 40 : "SC40", 45 : "SC45",
                                   50 : "SC50", 60 : "SC60", 70 : "SC70", 75 : "SC75",
                                   80 : "SC80", 85 : "SC85", 90 : "SC90", 120 : "SC120",
                                   150 : "SC150", 160 : "SC160", 180 : "SC180", 190 : "SC190"},
                    "MoSold" : {1 : "Jan", 2 : "Feb", 3 : "Mar", 4 : "Apr", 5 : "May", 6 : "Jun",
                                7 : "Jul", 8 : "Aug", 9 : "Sep", 10 : "Oct", 11 : "Nov", 12 : "Dec"}
                    })
```

        - Convert some "nominal" features into ordinal features

```
df_train.replace({"Alley" : {"Grvl" : 1, "Pave" : 2},
             "BsmtCond" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5},
             "BsmtExposure" : {"No" : 0, "Mn" : 1, "Av": 2, "Gd" : 3},
             "BsmtFinType1" : {"No" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4,
                                "ALQ" : 5, "GLQ" : 6},
             "BsmtFinType2" : {"No" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4,
                                "ALQ" : 5, "GLQ" : 6},
             "BsmtQual" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd" : 4, "Ex" : 5},
```

Data Cleaning

Step 2: Check missing values (cont.)

- Create new features
    - #1: Simplifications of existing features

```
df_train["SimplOverallQual"] = df_train.OverallQual.replace({1 : 1, 2 : 1, 3 : 1, # bad
                                                             4 : 2, 5 : 2, 6 : 2, # average
                                                             7 : 3, 8 : 3, 9 : 3, 10 : 3 # good
                                                            })
```

- #2: Combinations of existing features

```
# Overall kitchen score
df_train["KitchenScore"] = df_train["KitchenAbvGr"] * df_train["KitchenQual"]
# Overall fireplace score
df_train["FireplaceScore"] = df_train["Fireplaces"] * df_train["FireplaceQu"]
```

Data Cleaning

Step 2: Check missing values (cont.)

- #3: Polynomials on the ***top 10 existing features***

```
# Find most important features relative to target
print("Find most important features relative to target")
corr = df_train.corr()
corr.sort_values(["SalePrice"], ascending = False, inplace = True)
corr.SalePrice
```

```
Find most important features relative to target
SalePrice         1.000000
OverallQual       0.819240
AllSF             0.817272
AllFlrsSF         0.729421
GrLivArea         0.718844
                    ...
LandSlope        -0.040114
SimplExterCond   -0.042183
KitchenAbvGr     -0.147891
EnclosedPorch    -0.148636
LotShape         -0.285903
Name: SalePrice, Length: 88, dtype: float64
```

```
df_train["OverallQual-s2"] = df_train["OverallQual"] ** 2
df_train["OverallQual-s3"] = df_train["OverallQual"] ** 3
df_train["OverallQual-Sq"] = np.sqrt(df_train["OverallQual"])
df_train["AllSF-2"] = df_train["AllSF"] ** 2
df_train["AllSF-3"] = df_train["AllSF"] ** 3
df_train["AllSF-Sq"] = np.sqrt(df_train["AllSF"])
df_train["AllFlrsSF-2"] = df_train["AllFlrsSF"] ** 2
df_train["AllFlrsSF-3"] = df_train["AllFlrsSF"] ** 3
df_train["AllFlrsSF-Sq"] = np.sqrt(df_train["AllFlrsSF"])
df_train["GrLivArea-2"] = df_train["GrLivArea"] ** 2
df_train["GrLivArea-3"] = df_train["GrLivArea"] ** 3
df_train["GrLivArea-Sq"] = np.sqrt(df_train["GrLivArea"])
df_train["SimplOverallQual-s2"] = df_train["SimplOverallQual"] ** 2
df_train["SimplOverallQual-s3"] = df_train["SimplOverallQual"] ** 3
df_train["SimplOverallQual-Sq"] = np.sqrt(df_train["SimplOverallQual"])
df_train["ExterQual-2"] = df_train["ExterQual"] ** 2
df_train["ExterQual-3"] = df_train["ExterQual"] ** 3
df_train["ExterQual-Sq"] = np.sqrt(df_train["ExterQual"])
df_train["GarageCars-2"] = df_train["GarageCars"] ** 2
df_train["GarageCars-3"] = df_train["GarageCars"] ** 3
df_train["GarageCars-Sq"] = np.sqrt(df_train["GarageCars"])
df_train["TotalBath-2"] = df_train["TotalBath"] ** 2
df_train["TotalBath-3"] = df_train["TotalBath"] ** 3
df_train["TotalBath-Sq"] = np.sqrt(df_train["TotalBath"])
df_train["KitchenQual-2"] = df_train["KitchenQual"] ** 2
df_train["KitchenQual-3"] = df_train["KitchenQual"] ** 3
df_train["KitchenQual-Sq"] = np.sqrt(df_train["KitchenQual"])
df_train["GarageScore-2"] = df_train["GarageScore"] ** 2
df_train["GarageScore-3"] = df_train["GarageScore"] ** 3
df_train["GarageScore-Sq"] = np.sqrt(df_train["GarageScore"])
```

Data Cleaning

Step 2: Check missing values (cont.)

- Differentiate numerical features (minus the target) and categorical features
- Inpute remaining missing values for numerical features using median

```
train_num = train_num.fillna(train_num.median())
print("Remaining NAs for numerical features in train : " + str(train_num.isnull().values.sum()))
```

```
Remaining NAs for numerical features in train : 0
```

- Create dummy features for categorical values via one-hot encoding

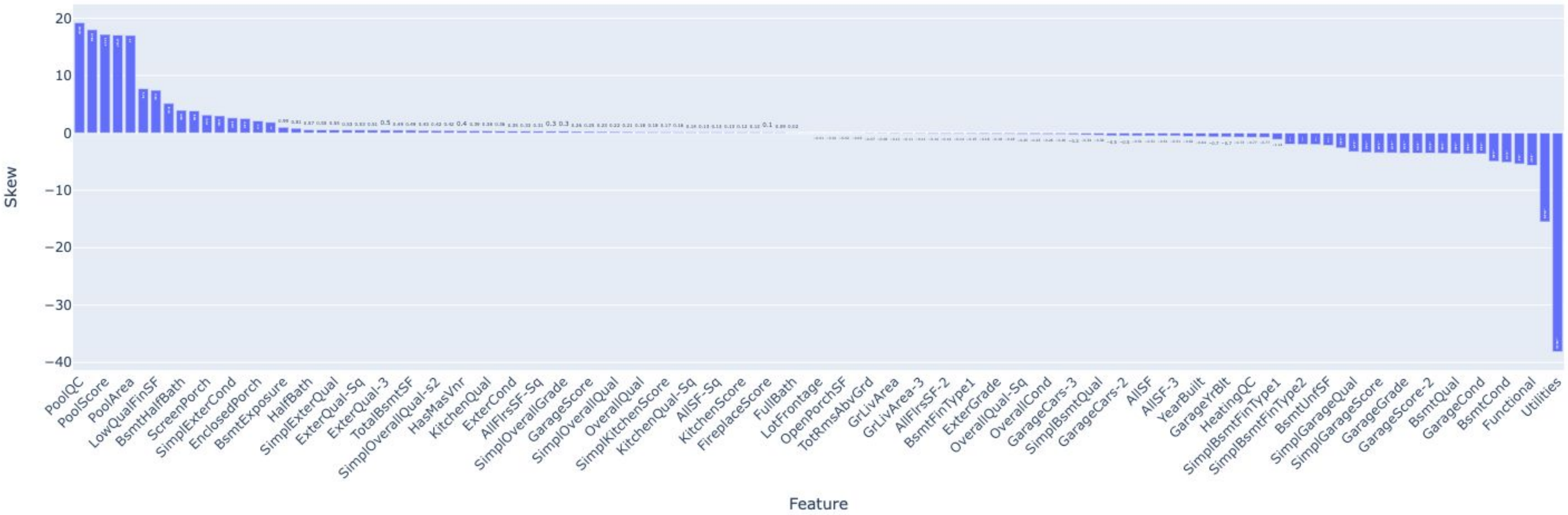
```
print("NAs for categorical features in train : " + str(train_cat.isnull().values.sum()))
train_cat = pd.get_dummies(train_cat)
print("Remaining NAs for categorical features in train : " + str(train_cat.isnull().values.sum()))
```

```
NAs for categorical features in train : 2
Remaining NAs for categorical features in train : 0
```

✅ Our dataset are ready for next step!

Data Cleaning

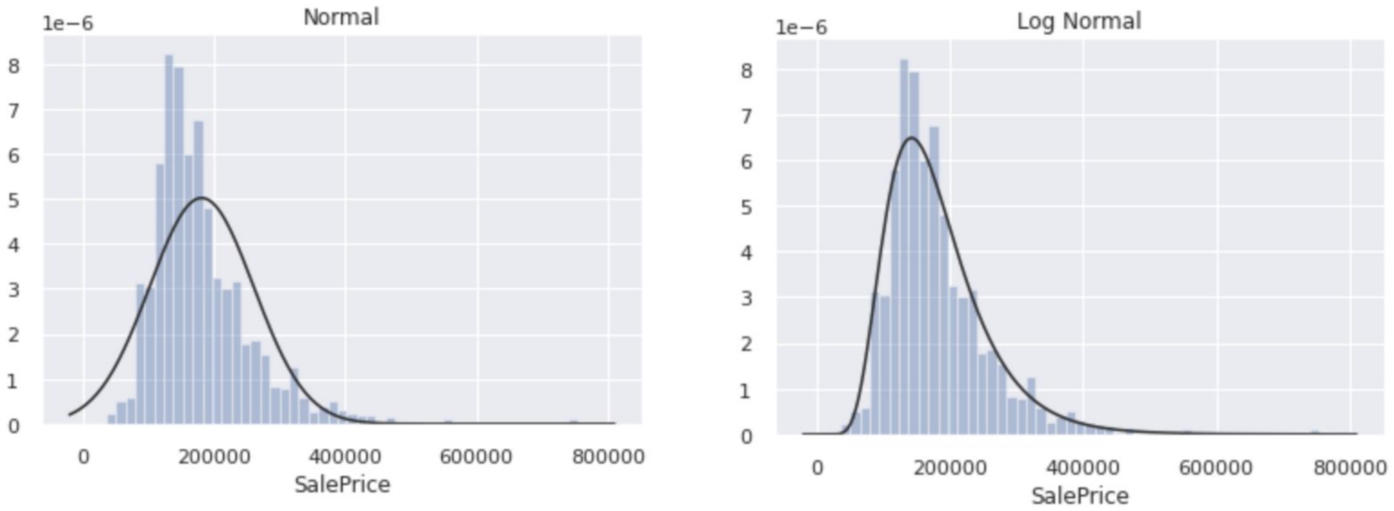
Step 3: Check skewness



Data Cleaning

Step 3: Check skewness (cont.)

```
[229] y = df_train['SalePrice']
      plt.figure(2); plt.title('Normal');
      sns.distplot(y, kde=False, fit=st.norm);
      plt.figure(3); plt.title('Log Normal');
      sns.distplot(y, kde=False, fit=st.lognorm);
```



It is apparent that SalePrice doesn't follow normal distribution, so before performing regression it has to be transformed.

Data Cleaning

Step 3: Check skewness (cont.)

- Log transform of the skewed numerical features to lessen impact of outliers
  - Inspired by Alexandru Papiu's script[1]
  - | skewness | > 0.5 is considered at least moderately skewed

```
skewness = train_num.apply(lambda x: skew(x))
skewness = skewness[abs(skewness) > 0.5]
print(str(skewness.shape[0]) + " skewed numerical features to log transform")
skewed_features = skewness.index
train_num[skewed_features] = np.log1p(train_num[skewed_features])
```

```
87 skewed numerical features to log transform
```

✅ Our dataset are ready for fitting model!

1: https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models

Correlation

Finding Correlation coefficients between numeric features and SalePrice.

```
[223] correlation = numeric_features.corr()
      print(correlation['SalePrice'].sort_values(ascending = False),'\n')
```

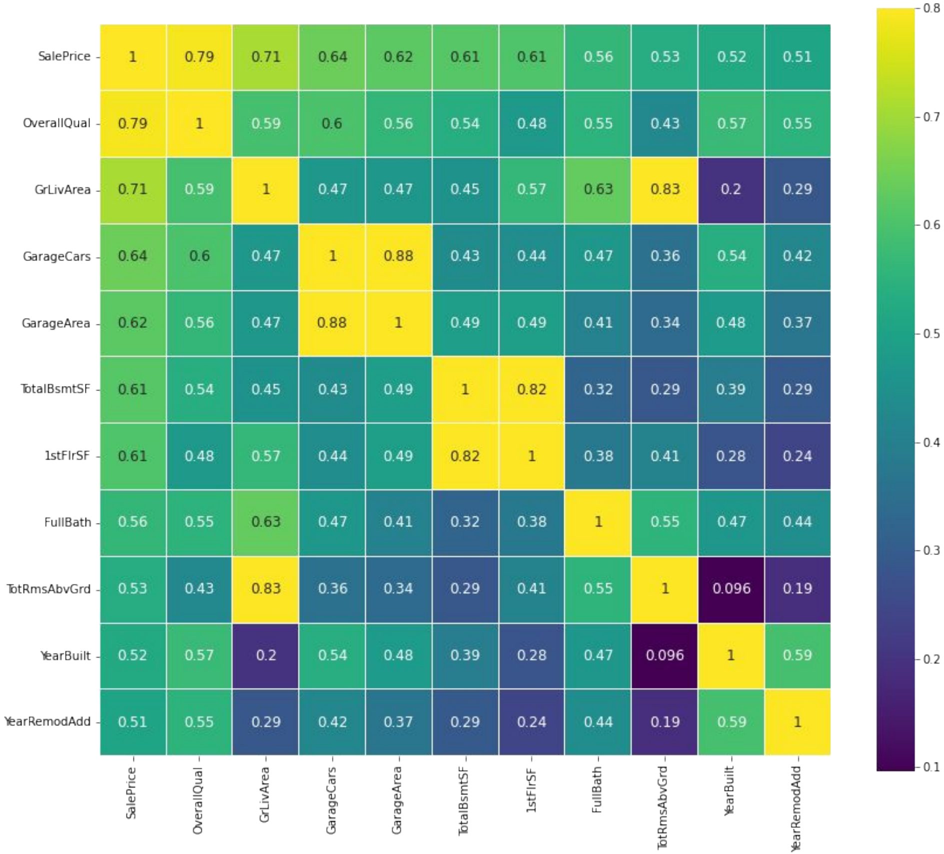
```
SalePrice        1.000
OverallQual      0.791
GrLivArea        0.709
GarageCars       0.640
GarageArea       0.623
TotalBsmtSF      0.614
1stFlrSF         0.606
FullBath         0.561
TotRmsAbvGrd     0.534
YearBuilt        0.523
YearRemodAdd     0.507
GarageYrBlt      0.486
MasVnrArea       0.477
Fireplaces       0.467
BsmtFinSF1       0.386
LotFrontage      0.352
WoodDeckSF       0.324
2ndFlrSF         0.319
OpenPorchSF      0.316
HalfBath         0.284
LotArea          0.264
BsmtFullBath     0.227
BsmtUnfSF        0.214
BedroomAbvGr     0.168
ScreenPorch      0.111
PoolArea         0.092
```

```
MoSold           0.046
3SsnPorch        0.045
BsmtFinSF2      -0.011
BsmtHalfBath    -0.017
MiscVal         -0.021
LowQualFinSF    -0.026
YrSold          -0.029
OverallCond     -0.078
MSSubClass      -0.084
EnclosedPorch   -0.129
KitchenAbvGr    -0.136
Name: SalePrice, dtype: float64
```

Correlation

Visualizing correlations

- 'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'.
- 'GarageCars' and 'GarageArea' are strongly correlated variables. It is because the number of cars that fit into the garage is a consequence of the garage area.
- 'YearBuilt' appears slightly correlated with 'SalePrice'. This required more analysis to arrive at a conclusion may be do some time series analysis.



Split Dataset

- Split cleaned data into training and testing dataset based on their Id
  - for predict target test

```
new_train = train.iloc[:1460,:]
new_test = train.iloc[1460:,:]
```

- Split train data with 80% as training set & 20% as testing set
  - for validation test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(new_train, y,
                                                    test_size = 0.2,
                                                    random_state = 123)

print("X_train : " + str(X_train.shape))
print("X_test : " + str(X_test.shape))
print("y_train : " + str(y_train.shape))
print("y_test : " + str(y_test.shape))
```

```
X_train : (1168, 321)
X_test : (292, 321)
y_train : (1168,)
y_test : (292,)
```

Feature Selection

LASSO regularization drives the coefficient of uninformative features to 0

```
from sklearn.linear_model import LogisticRegression

# Now with LASSO
logr = LogisticRegression(C=0.1, penalty="l1", solver="liblinear", random_state=123)
logr.fit(X_train, y_train.astype('int'))

sorted_tuples = sorted(list(zip(X_train.columns.values,logr.coef_[0])),key=lambda x:x[1],reverse=True)
for feature,coef in sorted_tuples:
    print(f'{feature:30s} : {coef: 0.3f}')
```

```
# which columns were kept?
X_train.columns[logr.coef_[0] != 0]
```

Feature Selection

- Feature to keep by doing Lasso:

['YearRemodAdd','BsmtFinSF1', BsmtUnfDF', 'TotalBsmtSF'. 'GarageArea', 'WoodDeckSF', 'OpenPorchDF', 'EnclosedPorch', 'MiscVal', 'YrSold'. 'OverallGrade', 'AllPorchSF', 'GrLivArea-Sq', 'GarageScore-Sq']

- What matters?

  Remodel date, the area of first-time finished basement, unfinished basement area, how big the basement is, the size of garage, wood deck area, open porch area, enclosed porch area, ...

Feature Selection

Slope, General shape of the property, overall material and finish quality, ...
Different from Lasso!

Tree Based Model Feature Importance

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=123).fit(X_train,y_train)
rf.feature_importances_
```

```
feature_importances = pd.Series(rf.feature_importances_,index=X_train.columns)
feature_importances.sort_values(ascending=False).round(3)
```

```
# which columns were kept?
X_train.columns[feature_importances != 0]
```

```
Index(['LotFrontage', 'LotArea', 'Street', 'LotShape', 'LandSlope',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
       ...
       'SaleType_ConLw', 'SaleType_New', 'SaleType_Oth', 'SaleType_WD',
       'SaleCondition_Abnorml', 'SaleCondition_AdjLand',
       'SaleCondition_Alloca', 'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=309)
```

Baseline model

- Model with full features
  - From this table, we choose three models (with score more than 0.9): **Ridge, XGBoost, GradientBoost**
  - We use **GridSearch** to find best parameters.
    - Ridge: We find the best alpha for ridge is 10.
    - XGBoost and GradientBoost: learning_rate = 0.01, max_depth =4, n_estimators = 2000, subsample = 0.5.

model	unstandard	standard
LinearRegression	0.889	0.902
KNN	0.730	0.831
SVR	0.738	0.823
Ridge	0.906	0.909
DecisionTree	0.745	0.717
ExtraTree	0.764	0.721
XGBoost	0.908	0.918
RandomForest	0.891	0.889
AdaBoost	0.831	0.825
GradientBoost	0.917	0.917
Bagging	0.871	0.874

Model Selection

- Model with full features
  - Unstandard
    - Ridge 0.906
    - **Ridge10 0.917**
    - **XGBoost_tuned 0.921**
    - **GradientBoost_tuned 0.921**
  - Standard
    - **Ridge10 0.9**
    - **XGBoost_tuned 0.921**
    - **GradientBoost_tuned 0.922**

	model	unstandard	standard
0	LinearRegression	0.889	-2.71
1	KNN	0.730	0.821
2	SVR	0.738	0.790
3	Ridge	0.906	0.896
4	Ridge10	0.917	0.900
5	DecisionTree	0.746	0.724
6	ExtraTree	0.760	0.764
7	XGBoost	0.908	0.916
8	XGBoost_tuned	0.921	0.921
9	RandomForest	0.885	0.890
10	AdaBoost	0.828	0.824
11	GradientBoost	0.916	0.916
12	GradientBoost_tuned	0.921	0.922
13	Bagging	0.877	0.868

Model Selection

- Models with lasso selected features
  - Unstandard
    - **XGBoost 0.889**
    - **GradientBoost 0.894**
  - Standard
    - **XGBoost 0.898**
    - **GradientBoost_tuned 0.895**

	model	unstandard	standard
0	LinearRegression	0.879	0.879
1	KNN	0.730	0.835
2	SVR	0.739	0.831
3	Ridge	0.879	0.879
4	Ridge10	0.879	0.877
5	DecisionTree	0.713	0.717
6	ExtraTree	0.720	0.754
7	XGBoost	0.889	0.898
8	XGBoost_tuned	0.885	0.891
9	RandomForest	0.873	0.875
10	AdaBoost	0.812	0.812
11	GradientBoost	0.894	0.894
12	GradientBoost_tuned	0.894	0.895
13	Bagging	0.867	0.867

Model Selection

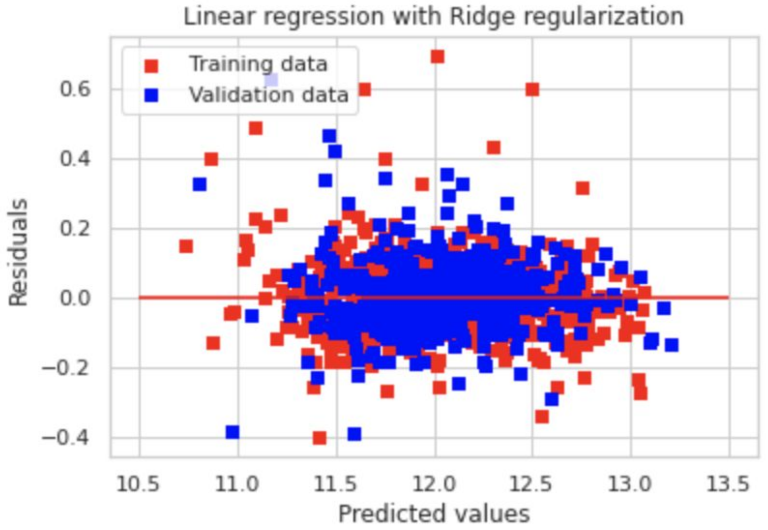
- Models with tree selected features
  - Unstandard
    - Ridge 0.906
    - **Ridge10 0.917**
    - **XGBoost_tuned 0.921**
    - **GradientBoost_tuned 0.925**
  - Standard
    - **Ridge10 0.902**
    - **XGBoost_tuned 0.921**
    - **GradientBoost_tuned 0.923**

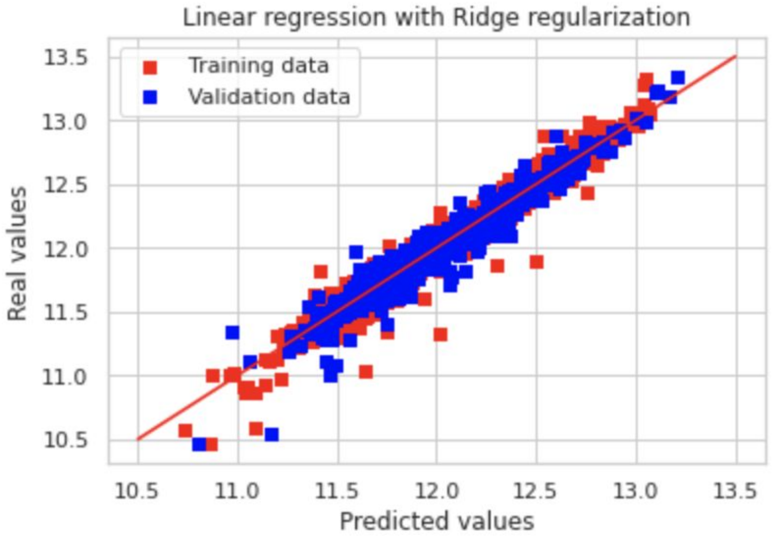
	model	unstandard_tree	standard_tree
0	LinearRegression	0.888	0.888
1	KNN	0.730	0.821
2	SVR	0.738	0.797
3	Ridge	0.906	0.897
4	Ridge10	0.917	0.902
5	DecisionTree	0.710	0.709
6	ExtraTree	0.745	0.687
7	XGBoost	0.908	0.916
8	XGBoost_tuned	0.921	0.921
9	RandomForest	0.892	0.886
10	AdaBoost	0.831	0.831
11	GradientBoost	0.917	0.916
12	GradientBoost_tuned	0.925	0.923
13	Bagging	0.872	0.881

Summary

model	unstandard_full	standard_full	unstandard_lasso	standard_lasso	unstandard_tree	standard_tree
LinearRegression	0.889	-2.71	0.879	0.879	0.888	0.888
KNN	0.730	0.821	0.730	0.835	0.730	0.821
SVR	0.738	0.790	0.739	0.831	0.738	0.797
Ridge	0.906	0.896	0.879	0.879	0.906	0.897
Ridge10	0.917	0.900	0.879	0.877	0.917	0.902
DecisionTree	0.746	0.724	0.713	0.717	0.710	0.709
ExtraTree	0.760	0.764	0.720	0.754	0.745	0.687
XGBoost	0.908	0.916	0.889	0.898	0.908	0.916
XGBoost_tuned	0.921	0.921	0.885	0.891	0.921	0.921
RandomForest	0.885	0.890	0.873	0.875	0.892	0.886
AdaBoost	0.828	0.824	0.812	0.812	0.831	0.831
GradientBoost	0.916	0.916	0.894	0.894	0.917	0.916
GradientBoost_tuned	0.921	0.922	0.894	0.895	0.925	0.923
Bagging	0.877	0.868	0.867	0.867	0.872	0.881

Project 1 Best Model - Ridge





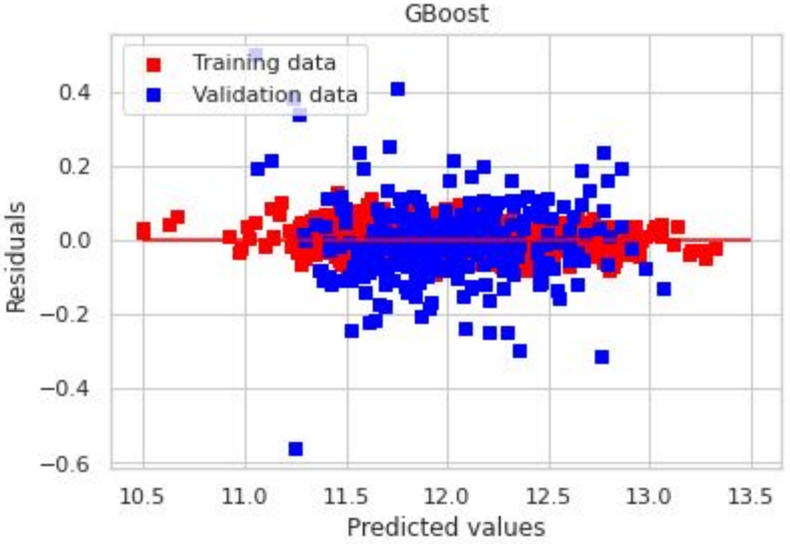
Ridge model score on train: 0.9462 On test: 0.9175

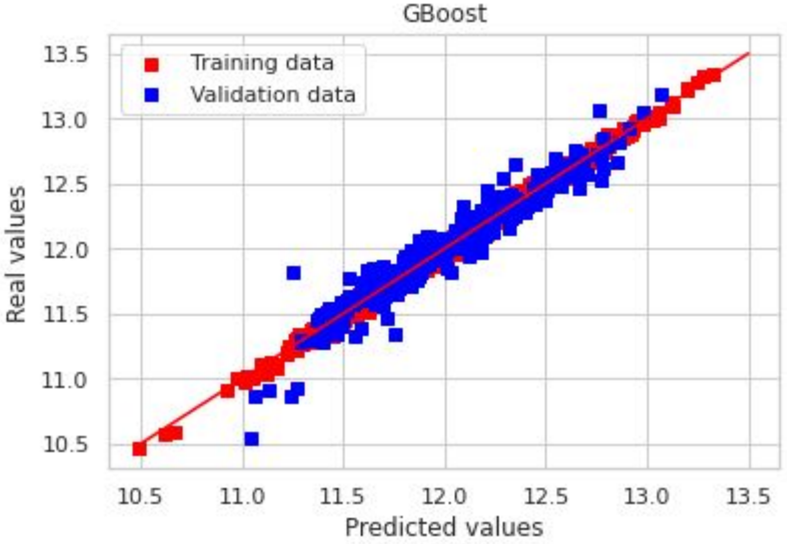
MSE_test: 0.0128 MSE_train: 0.0084

RMSE_test: 0.1133 RMSE_train: 0.0918

Final Best Model - GradientBoost

```
GBoost_model = GradientBoostingRegressor(learning_rate=0.01,
                                          max_depth = 4,
                                          n_estimators= 2000,
                                          subsample = 0.5).fit(X_train,y_train)
```





Final Best Model - GradientBoost (Cont.)

GBoost model score on train: 0.9921 On test: 0.9254

MSE_test: 0.0118 MSE_train: 0.0012

RMSE_test: 0.1088 RMSE_train: 0.0351

Improvement!

vs

Ridge model score on train: 0.9462 On test: 0.9175

MSE_test: 0.0128 MSE_train: 0.0084

RMSE_test: 0.1133 RMSE_train: 0.0918

Thank you!

image from: https://siewlinyap.github.io/portfolio/project3-analysis-on-housing-data/