University of St.Gallen

Using regression and machine learning algorithm for housing price prediction in Ames, Iowa

Term paper of Econometrics II

Ao Sun 18-624-007

Xiaoqian Zhu 18-624-056

May 5, 2019

Contents

1 Introduction

Real estate industry is the basic guarantee of people's life, and houses is one of the most common way to accumulating wealth. In the last decade, housing prices in most cities have been on the rise. Whether the prices will continue to rise in the future, and to what level they can reach, it's highly concerned among the government, developers, and ordinary citizens.

Liu (2014) pointed that accurate forecasting for future housing price is very important for socioeconomic development and national lives and helps in establishing real estate policies.

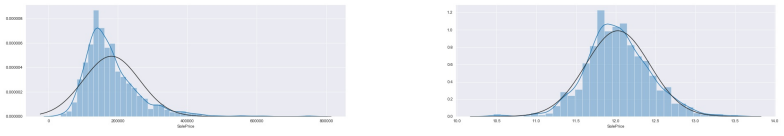
In this paper we used four different methods to fit the data, OLS, LASSO and Ridge regression and Random Forest. After that, we used bootstrap to get the performance of the estimators under each model. It turns out that Random forest is superior to the others on both test and train data.

The analysis of housing price has two principal trends: one is called hedonic-based regression, which is more concentrating on the market fundamentals.

Bin(2004) used a semi-parametric method to estimate the hedonic price function and find it performances better than conventional parametric model. There are lots of studies use machine learning method to do the prediction. Park and Bae(2015) developed a prediction model of the Fairfax County of Virgina based on naïve Bayesian, Adaboost methods, and found that ML algorithm enhance the predictability and significantly contribute on the correct evaluation of real estate price. Gu, Zhu and Jiang(2011) suggested a hybrid model of genetic algorithm and support vector machine(G-SVM) model to predict the housing price and found that this model has higher forecast accuracy.

2 Data

We get 79 explanatory variables describing almost every aspect of residential homes with the *SalePrice* in Ames of Iowa from Kaggle Competition. After the visualization of the *SalePrice*, we found that it fits the log-normal distribution well.



(a) Normal Distribution Fitting (b) Log-Normal Distribution Fitting

Figure 1: Distribution visualization of Sale Price

There are several missing values, we drop those variables with missing values more than 10% of the number observations. And transformed non-numerical

labels (as long as they are comparable) to numerical labels.
From the plot we can observe outliers, we delete several unreasonable outliers of variables *GrLivArea* and *TotalBsmtSF*.
In order to compare the performance of each model, we split 20% of the observations as test data, and do the fitting on the rest (train) data.
Until now still remain many explanatory variables, so we get the correlation matrix and use *SelectKBest* in python to get the first fifteen important features. The correlations as follows:

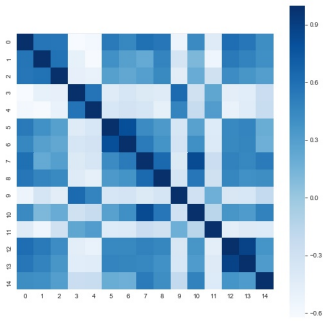


Figure 2: Correlation matrix

3 Methodology

3.1 Identification strategy and estimation methods

In this paper, we selected four estimation methods. They are OLS, Ridge(Hoerl, Kennard, 1970), Lasso(Tibshirani, 1996) and Random Forest (Breiman, 2001) respectively. Since the first three model have linear expressions, we do the variance comparison via bootstrap between those three models.
Before fiting the model, we had selected 15 explanatory variables including *OverallQual*, *YearBuilt*, *YearRemodAdd* etc, to predict the log *SalePrice* of house. OLS, Ridge and Lasso are all in generalized linear model classes, which means we can get the estimated weight parameters $\hat{\beta}$ after fitting the models. While Random Forest is a non-linear model, from which we get the variables importance instead of the estimated parameters.
We use Grid Search (Bergstra, Bengio, 2012) method to select hyper parameters during fitting process.
After the model fitting, we discussed the performance of the models in test data. We chose MSE as the criterion and do bootstrap to estimate the variance of $\hat{\beta}$ in linear models and variables importance (Mean decrease impurity) in Random Forest.

3.2 Advantages and disadvantages of the different estimators

In our data, *SalePrice* is a continuous variable, there are two advantages of OLS.
Firstly, it's a simple method to explain and to understand. $\hat{\beta}$ suggests how many *SalePrice* will change with one unit change of explanatory variables.
Secondly, it's applicable in most general condition whether the *SalePrice* is continuous or discrete. If *SalePrice* is a binary variable, perhaps use probit or logit model will be better than classical OLS. But OLS model can be used in most regression tasks.
Compared to other models, the disadvantages of OLS are main in three points. Primarily, it's sensitive to outliers, so that's why we've removed the outliers in training data. Then, when the data is not normally distributed, the outcome might be unreliable, so we've used a relatively big data set to alleviate the problem. In addition, OLS has tendency to overfit data.
Lasso and Ridge can help us solve the last disadvantage of OLS. In Lasso we can use L1 regularization to shrinking some elements in β to 0. That means Lasso is useful for variables selection by which the redundant variables are discarded, while Ridge use L2 regularization to make the redundant elements relatively small. Including more variables means larger probability to meet the overfitting problem, since they can help decrease the effects of redundant variables, they become two common way to avoid overfitting. The most disadvantages of Lasso and Ridge is the bigger computation time than OLS because of the regularization.
The last selected model is Random Forest. The biggest difference is that Random Forest is a non-linear model. It has different criterion compare to linear model. Breiman (2001) pointed out some advantages of Random Forest:
Firstly, using random features can accelerate the model fitting process.
Next, Random Forest is less sensitive to outliers than OLS.
Moreover, Random Forest is an ensemble model which combine many decision trees and use bagging methods to avoid overfitting.
The biggest disadvantage of Random Forest is the curse of dimensionality. And it doesn't give much economics explanation as the above three models do.

4 Results

4.1 Table of the results

After the point estimation, we did 100 times bootstrap for variance estimation. Since Efron(1993) pointed that 50-200 times resampling is enough for variance estimation. We got result tables for all models:

	Random Forest		OLS		Ridge		Lasso	
	Mean decrease impurity	std	$\hat{\beta}$	std	$\hat{\beta}$	std	$\hat{\beta}$	std
OverallQual	182.42	11.74	113.90	9.39	107.08	8.96	108.84	8.91
YearBuilt	102.22	13.09	1.93	0.39	1.94	0.38	1.92	0.37
YearRemodAdd	40.86	9.27	3.36	0.43	3.43	0.43	3.45	0.43
ExterQual	63.59	12.62	16.37	14.87	10.74	12.77	**0.00**	4.07
BsmtQual	29.46	8.29	-15.41	8.56	-15.30	8.06	-10.16	7.52
TotalBsmtSF	71.64	9.91	0.24	0.03	0.24	0.03	0.24	0.03
1stFlrSF	62.50	7.83	0.06	0.04	0.06	0.04	0.06	0.04
GrLivArea	155.99	13.87	0.39	0.03	0.39	0.03	0.37	0.02
FullBath	52.93	11.28	-34.32	19.27	-25.81	15.21	**0.00**	12.44
KitchenQual	26.46	8.49	-25.71	9.62	-23.68	8.63	-12.87	9.61
TotRmsAbvGrd	22.09	3.03	-6.92	7.85	-7.31	7.51	-3.07	5.65
GarageType	32.14	11.07	-11.50	4.49	-11.87	4.44	-10.46	4.42
GarageCars	69.48	9.73	26.57	22.75	20.01	16.57	**0.00**	9.10
GarageArea	61.55	8.81	0.23	0.07	0.26	0.05	0.32	0.04
MeanArea	26.67	4.12	0.12	0.03	0.12	0.03	0.12	0.02
MSE in out of sample data	51.54	1.97	104.55	4.57	105.90	4.37	107.45	4.12

*All number have been multiplied with 1000 since the values are extremely small.

Table 1: Table of results

4.2 Compare the results of different estimators

Firstly, we compared the results between linear models. And then compare the best model among linear models with Random Forest.
1. OLS has the minimum MSE in test data. Second is Ridge. Lasso performances the worst among linear models based on this criterion.
2. *FullBath* has the biggest negative value in OLS and Ridge, but in Lasso, the L1 regularization make it zero.
3. Most variables' values in Ridge are less than those in OLS.
4. The standard deviations in OLS are always bigger than those in Ridge and Lasso. But between Ridge and Lasso, the standard deviations have no obvious difference.
5. *ExterQual*, *FullBath*, *GarageCars* have been assigned to 0 in Lasso, which means Lasso model think they are redundant variables.
Then compare the Random Forest and linear models:
1. Random Forest gets better performance on test data than linear models.
2. Random Forest considers *OverallQual* has the biggest influence, too. But Random Forest think the next important variable is *GrLivArea* which has a relatively small weight in linear model.

4.3 Critically discussion of the results.

As what we've concluded above, OLS model has the biggest standard deviation among linear models in train data. Even OLS got the best out sample performance in linear models, we can't conclude the OLS is the best model for prediction still. In addition, Lasso model discarded *ExterQual* (Evaluates the quality of the material on the exterior), *FullBath* (Full bathrooms above grade) and *GarageCars* (Size of garage in

car capacity). Although Lasso got the smallest standard deviations, the discarded variables can intuitively affect house price much in reality. We thought this might be the main reason that Lasso got the worst performance in out sample.
Besides, the MSE of Random Forest in test data is only half of the linear models. Even we used non-traditional linear model, like Ridge and Lasso, Random forests still outperform. That may mean linear models have more limitations to explain the explanatory ability of variables on *SalePrice*.
We proposed 3 reason why linear model did much worse than Random Forest:
1. OLS, Lasso and Ridge are one-dimensional linear models, they are much more simpler.
2. Many variables are discrete factors rather than continuous number. For example, variable *OverallQual*(Rates the overall material and finish of the house) was assigned as integers from 1 to 10 which is more effective in Random Forest estimation.
3. The outliers affected linear models more.

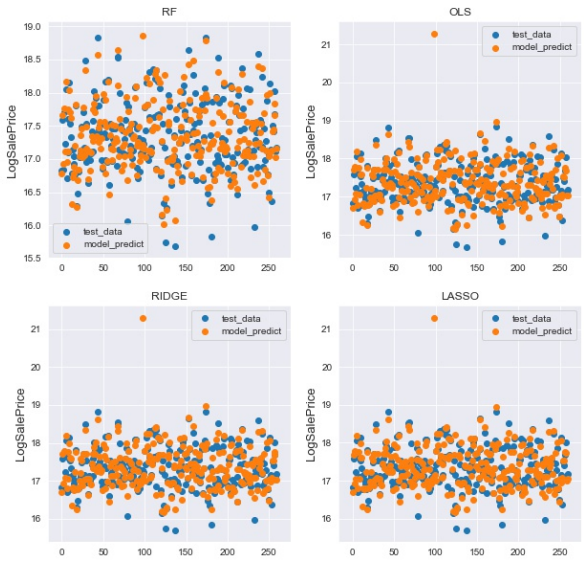


Figure 3: Comparison of test data with predicted data

From the above figure, we can see that there are some outliers in the prediction of linear model. However, outliers affected Random Forest prediction less than other models.

We thought the results is plausible. In all models, variable *OverallQual* is the primary factor affecting the house price. That outcome is in accord with our experience in reality. Moreover, in Random Forest model, the area variables affect housing price a lot and so do the linear model. Considering the lager value scale measurement of area

variables, we think the result is intuitive.

5 Conclusions

From the analysis above, we found Random Forest perform much better than linear models under out of sample MSE criterion.
Among linear models, OLS had the lowest out of sample MSE, but it had the biggest variance of estimated parameters $\hat{\beta}$. Lasso got the lowest variance, but because of discarding some variables, it had the largest out of sample MSE. Compared to OLS and Lasso, Ridge got an intermediate performance on either out of sample MSE or the variance of estimated parameters.

We think the Random Forest is the best model to do the prediction task, and we proposed three ways that probably can help increase the out of sample performance:
1. Use another criterion to select variables and test different variables combination, like information entropy.
2. Do normalization and standardization on some variables to align the scale of variables.
3. Check and remove more outliers in the training data, since we've got a lot explanatory variables.
4. Most literature now use suggest to do the Box-Cox transformation, which can be treated as a generalized log transformation, and most researchers also include higher-order terms in their regressions. Those methods could also be used for improving our model.

References

Bin, O. (2004). A prediction comparison of housing sales prices by parametric versus semi-parametric regressions. *Journal of Housing Economics*. 13(1), 68-84.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1), 281-305.

Bischoff, E., Bischoff, H., & Giuliano, F. (2005). An introduction to the bootstrap / b. efron, r.j. tibshirani.

Friedman, J. H. . (2001). Greedy function approximation: a gradient boosting machine. *The Annals of Statistics* , 29(5), 1189-1232.

Gu, J., Zhu, M., & Jiang, L. (2011). Housing price forecasting based on genetic algorithm and support vector machine. *Expert Systems with Applications* , 38(4), 3383-3386.

Hoerl, A., & Kennard, R. (1970). Ridge regression: applications to nonorthogonal problems. *Technometrics*, 12(1), 14.

Liu, J. G. , Zhang, X. L. , & Wu, W. P. . (2006). Application of fuzzy neural network for real estate prediction. *International Symposium on Advances in Neural Networks-isnn*. DBLP

Park, B., & Bae, J. K. (2015). Using machine learning algorithms for housing price prediction: the case of fairfax county, virginia housing data. *Expert Systems with Applications* , 42(6), 2928-2934.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society* , 58(1), 267-288.

Appendix

May 5, 2019

0.1 Import libraries

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from scipy import stats
        from scipy.stats import norm
        from sklearn.preprocessing import LabelEncoder
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.linear_model import LinearRegression, Lasso, Ridge
        from sklearn.metrics import mean_squared_error
        from sklearn.svm import SVR
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import f_regression
        from sklearn.model_selection import GridSearchCV
        from sklearn.utils import resample

In [2]: #### set the random seed
        np.random.seed(137)

In [3]: color = sns.color_palette()
        sns.set_style('darkgrid')
```

1 1. Import data

```
In [4]: # original_data = pd.read_csv('houspr.csv', index_col=0).drop(['Alley', 'PoolQC', 'Fen
        original_data = pd.read_csv('houspr.csv', index_col=0)
        train_data = original_data.iloc[:1200]
        test_data = original_data.iloc[1200:]

In [5]: original_data.head()

Out[5]:     MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
        Id
        1           60       RL         65.0     8450   Pave   NaN      Reg
        2           20       RL         80.0     9600   Pave   NaN      Reg
        3           60       RL         68.0    11250   Pave   NaN      IR1
```

```
4           70       RL         60.0     9550   Pave   NaN      IR1
5           60       RL         84.0    14260   Pave   NaN      IR1

   LandContour Utilities LotConfig    ...     PoolArea PoolQC Fence  \
Id                                    ...
1          Lvl    AllPub    Inside    ...            0    NaN   NaN
2          Lvl    AllPub       FR2    ...            0    NaN   NaN
3          Lvl    AllPub    Inside    ...            0    NaN   NaN
4          Lvl    AllPub    Corner    ...            0    NaN   NaN
5          Lvl    AllPub       FR2    ...            0    NaN   NaN

   MiscFeature MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
Id
1          NaN       0      2    2008        WD         Normal     208500
2          NaN       0      5    2007        WD         Normal     181500
3          NaN       0      9    2008        WD         Normal     223500
4          NaN       0      2    2006        WD        Abnorml     140000
5          NaN       0     12    2008        WD         Normal     250000

[5 rows x 80 columns]
```

1.0.1 statistics of the data

```
In [6]: original_data.dtypes
```

```
Out[6]: MSSubClass         int64
        MSZoning          object
        LotFrontage      float64
        LotArea            int64
        Street            object
        Alley             object
        LotShape          object
        LandContour       object
        Utilities         object
        LotConfig         object
        LandSlope         object
        Neighborhood      object
        Condition1        object
        Condition2        object
        BldgType          object
        HouseStyle        object
        OverallQual        int64
        OverallCond        int64
        YearBuilt          int64
        YearRemodAdd       int64
        RoofStyle         object
        RoofMatl          object
        Exterior1st       object
```

```
Exterior2nd       object
MasVnrType        object
MasVnrArea       float64
ExterQual         object
ExterCond         object
Foundation        object
BsmtQual          object
                  ...
BedroomAbvGr       int64
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
Length: 80, dtype: object
```

```
In [7]: original_data.describe(include='all')
```

Out[7]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley
count	1460.000000	1460	1201.000000	1460.000000	1460	91
unique	NaN	5	NaN	NaN	2	2
top	NaN	RL	NaN	NaN	Pave	Grvl
freq	NaN	1151	NaN	NaN	1454	50
mean	56.897260	NaN	70.049958	10516.828082	NaN	NaN

```
std       42.300571      NaN    24.284752    9981.264932    NaN   NaN
min       20.000000      NaN    21.000000    1300.000000    NaN   NaN
25%       20.000000      NaN    59.000000    7553.500000    NaN   NaN
50%       50.000000      NaN    69.000000    9478.500000    NaN   NaN
75%       70.000000      NaN    80.000000   11601.500000    NaN   NaN
max      190.000000      NaN   313.000000  215245.000000    NaN   NaN

       LotShape LandContour Utilities LotConfig      ...           PoolArea  \
count      1460        1460      1460      1460      ...        1460.000000
unique        4           4         2         5      ...                NaN
top         Reg         Lvl    AllPub    Inside      ...                NaN
freq        925        1311      1459      1052      ...                NaN
mean        NaN         NaN       NaN       NaN      ...           2.758904
std         NaN         NaN       NaN       NaN      ...          40.177307
min         NaN         NaN       NaN       NaN      ...           0.000000
25%         NaN         NaN       NaN       NaN      ...           0.000000
50%         NaN         NaN       NaN       NaN      ...           0.000000
75%         NaN         NaN       NaN       NaN      ...           0.000000
max         NaN         NaN       NaN       NaN      ...         738.000000

       PoolQC  Fence MiscFeature       MiscVal       MoSold       YrSold  \
count       7    281          54   1460.000000  1460.000000  1460.000000
unique      3      4           4           NaN          NaN          NaN
top        Gd  MnPrv        Shed           NaN          NaN          NaN
freq        3    157          49           NaN          NaN          NaN
mean      NaN    NaN         NaN     43.489041     6.321918  2007.815753
std       NaN    NaN         NaN    496.123024     2.703626     1.328095
min       NaN    NaN         NaN      0.000000     1.000000  2006.000000
25%       NaN    NaN         NaN      0.000000     5.000000  2007.000000
50%       NaN    NaN         NaN      0.000000     6.000000  2008.000000
75%       NaN    NaN         NaN      0.000000     8.000000  2009.000000
max       NaN    NaN         NaN  15500.000000    12.000000  2010.000000

       SaleType  SaleCondition      SalePrice
count      1460           1460    1460.000000
unique        9              6           NaN
top          WD         Normal           NaN
freq       1267           1198           NaN
mean        NaN            NaN  180921.195890
std         NaN            NaN   79442.502883
min         NaN            NaN   34900.000000
25%         NaN            NaN  129975.000000
50%         NaN            NaN  163000.000000
75%         NaN            NaN  214000.000000
max         NaN            NaN  755000.000000

[11 rows x 80 columns]
```

1.0.2 1.1 Fit normal distribution with original data(sale_price) and compare

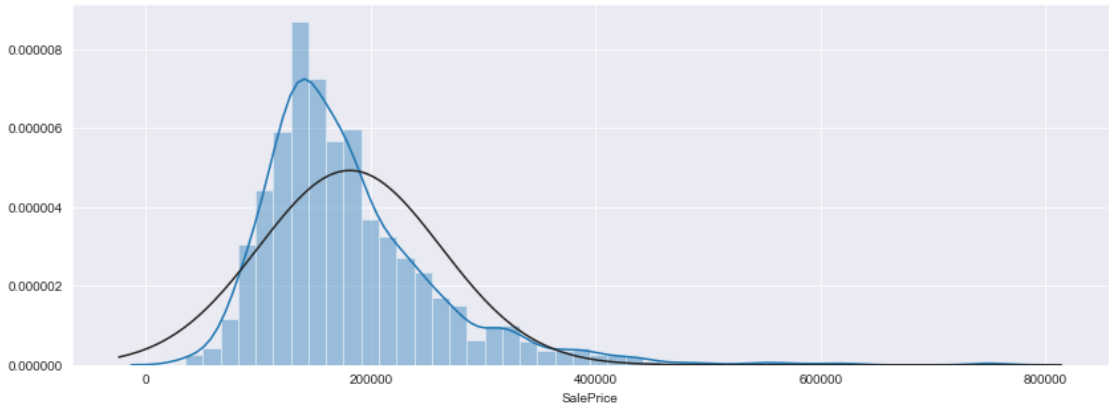
```
In [8]: # sns.distplot(train_data['SalePrice'], fit=norm)
        (mu, sigma) = norm.fit(train_data['SalePrice'])
        print("The mean of normal fitting the orginal data is {} \nThe standard deviation of no
```

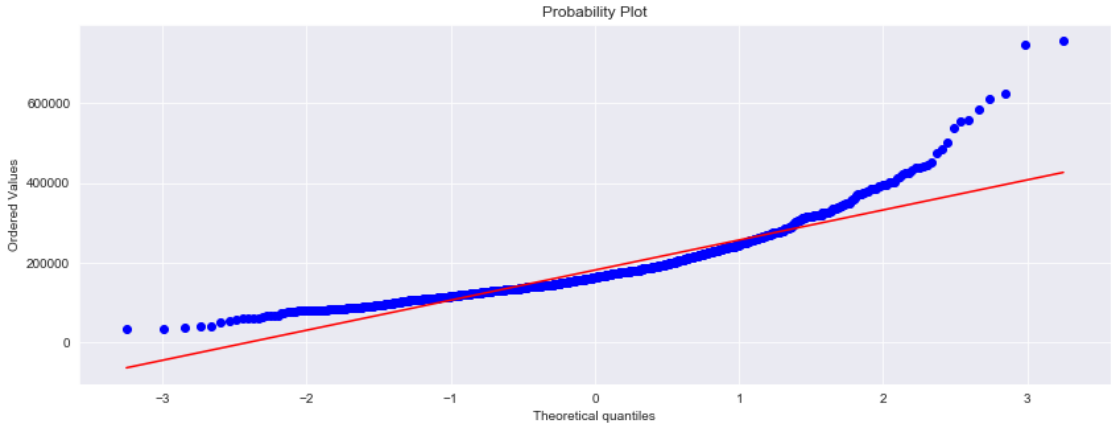
```
The mean of normal fitting the orginal data is 181414.6283
The standard deviation of normal fitting the orginal data is 81037.122
```

1.0.3 1.2 Plot the distribution of sale_price(original data)

```
In [54]: plt.figure(figsize=(14, 5))
         sns.distplot(train_data['SalePrice'], fit=norm)
         plt.savefig("normal distribution.jpg")

         # Get the QQ-plot
         plt.figure(figsize=(14, 5))
         res = stats.probplot(train_data['SalePrice'], plot=plt)

         plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tu
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





2 2. Data processing

2.1 2.1 Do the log transfromation on sale_price

```
In [56]: train_data.loc[:, "SalePrice"] = np.log1p(train_data["SalePrice"])
```

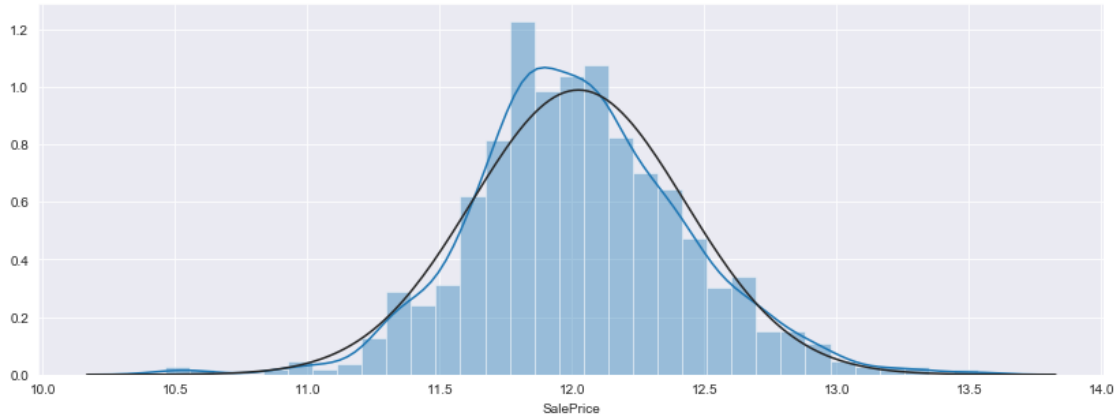
2.1.1 2.1.1 Fit normal distribution with the data after log-transformation and compare

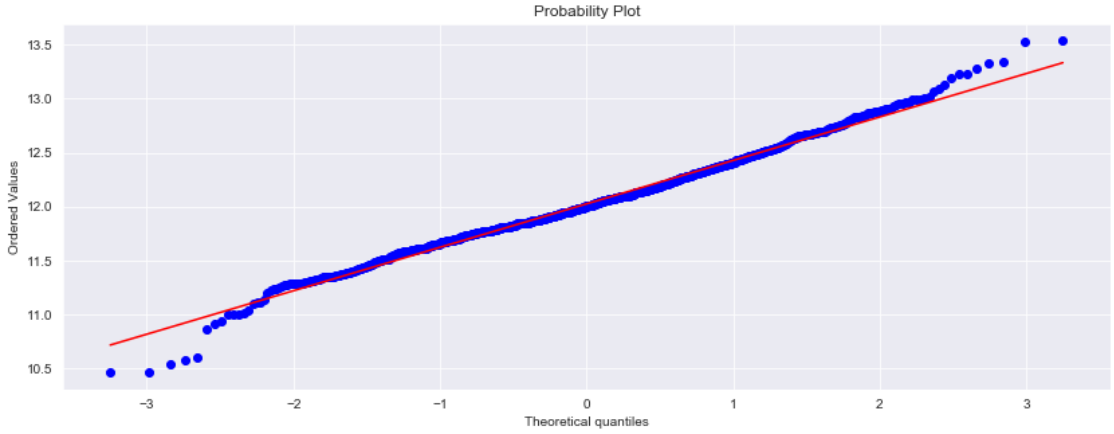
```
In [57]: (mu1, sigma1) = norm.fit(train_data['SalePrice'])
         print("The mean of normal fitting the log-data is {} \nThe standard deviation of norma
```

```
The mean of normal fitting the log-data is 12.0243
The standard deviation of normal fitting the log-data is 0.4034
```

2.1.2 2.1.2 Plot the distribution of log sale_price

```
In [58]: plt.figure(figsize=(14, 5))
         sns.distplot(train_data['SalePrice'], fit=norm)
         plt.savefig("log-normal distribution.jpg")

         # Get also the QQ-plot
         plt.figure(figsize=(14, 5))
         res = stats.probplot(train_data['SalePrice'], plot=plt)

         plt.show()
```

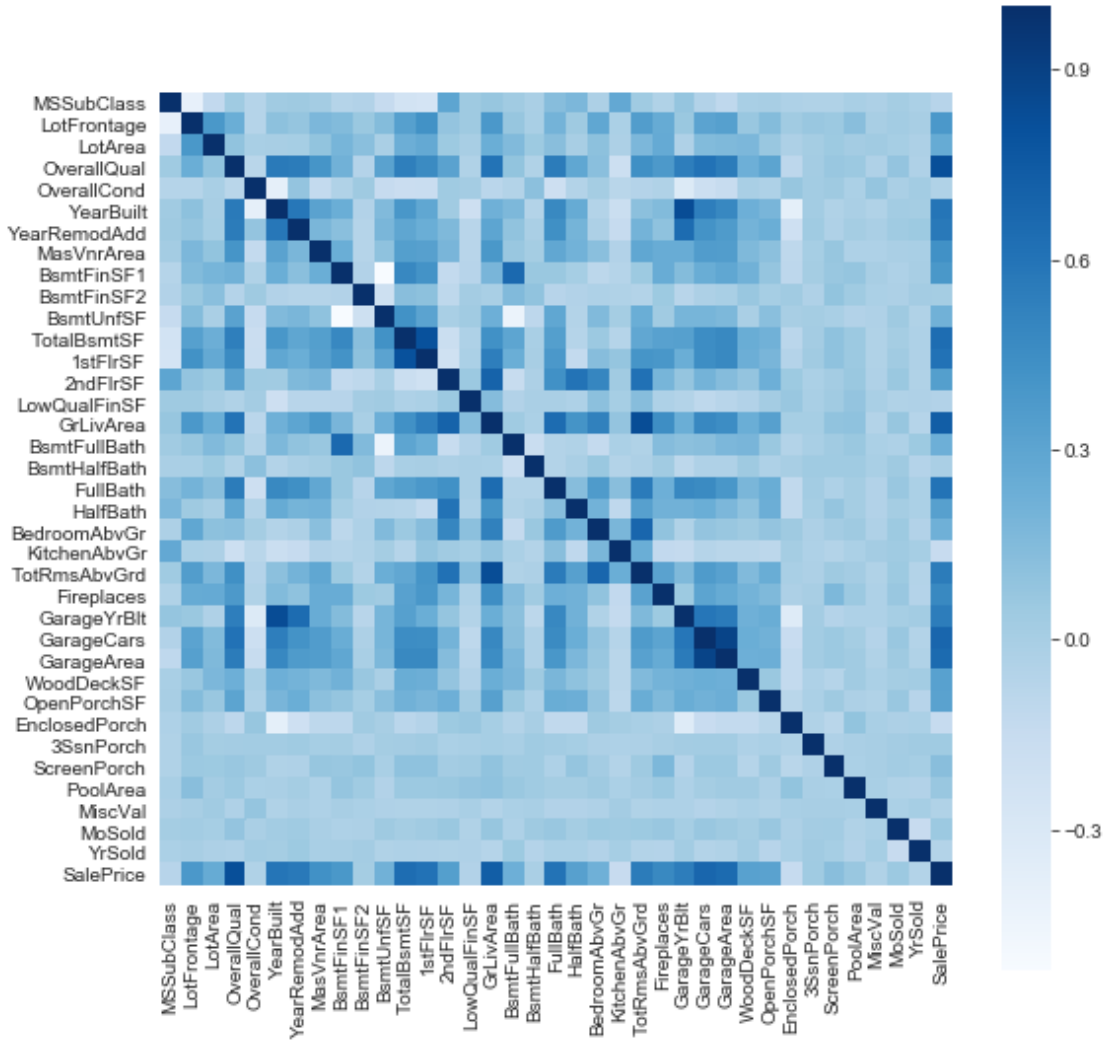




2.2 2.2 Plot the correlation matrix

```
In [19]: plt.subplots(figsize=(9, 9))
         corr_x = train_data.corr()
         sns.heatmap(corr_x, annot=False, vmax=1, square=True, cmap="Blues")
```

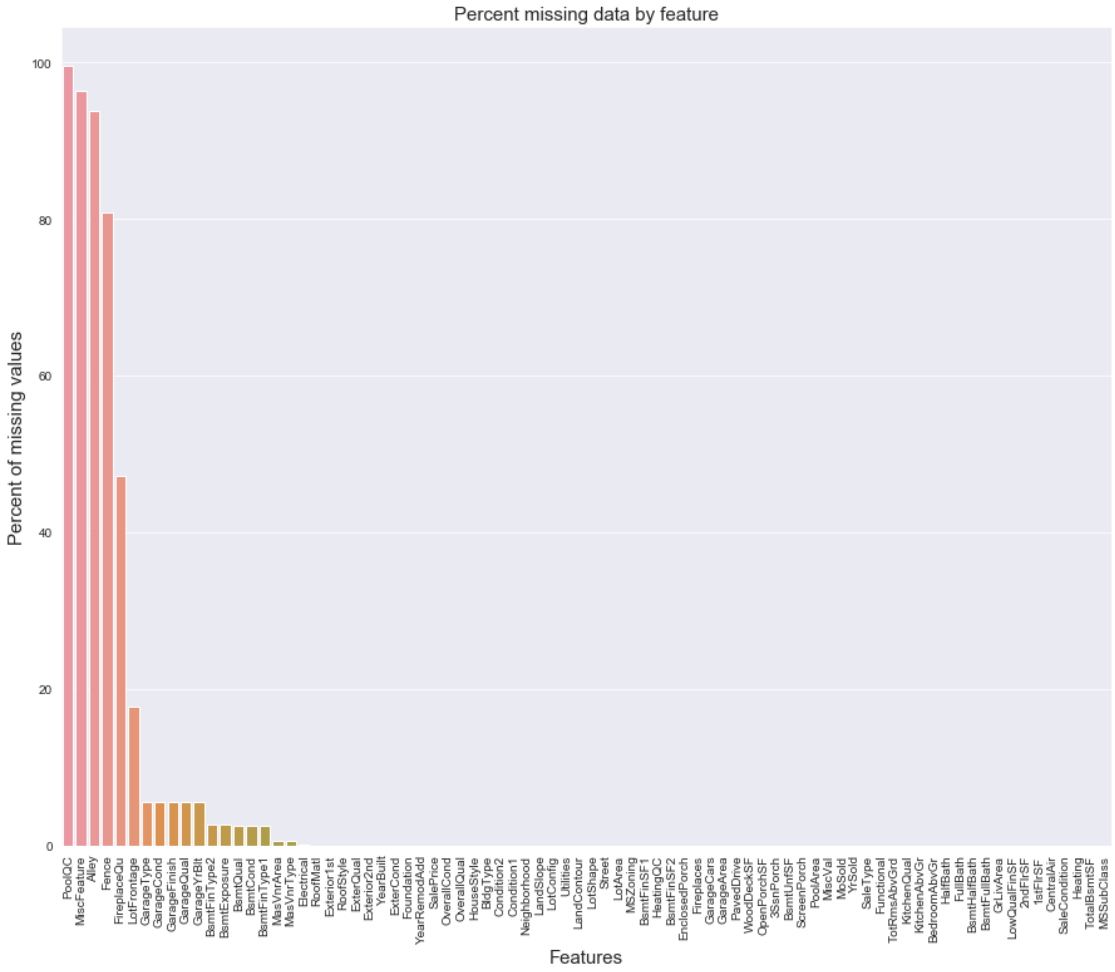
```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1a19631be0>
```



2.3 2.3 Dealing with the missing values

```
In [20]: missing_pd = (original_data.isnull().sum() / original_data.shape[0] * 100).sort_values
         missing_pd.head(20)

         f, ax = plt.subplots(figsize=(15, 12))
         plt.xticks(rotation='90')
         sns.barplot(x=missing_pd.index, y=missing_pd)
         plt.xlabel('Features', fontsize=15)
         plt.ylabel('Percent of missing values', fontsize=15)
         plt.title('Percent missing data by feature', fontsize=15)
```

```
Out[20]: Text(0.5, 1.0, 'Percent missing data by feature')
```



2.4 2.4 we drop the variables which missing data ratio exceeding 10%

```
In [21]: original_data = original_data.drop(missing_pd[missing_pd > 10].index, axis=1)
         missing_pd = (original_data.isnull().sum() / original_data.shape[0] * 100).sort_values
         missing_variables = missing_pd[missing_pd > 0].index

In [22]: # we still have 13 variables to deal with
         # Replace GarageType, GarageFinish, GarageQual and GarageCond missing value with None
         for var in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
             original_data[var] = original_data[var].fillna('None')

In [23]: # Replace GarageYrBlt missing value with 0
         original_data['GarageYrBlt'] = original_data['GarageYrBlt'].fillna(0)

In [24]: # Replace BsmtFinType2, BsmtExposure, BsmtQual, BsmtFinType1, BsmtCond missing value
         for var in ('BsmtFinType2', 'BsmtExposure', 'BsmtQual', 'BsmtFinType1', 'BsmtCond'):
             original_data[var] = original_data[var].fillna('None')
```

In [25]:
```
# Replace MasVnrArea missing with 0
original_data["MasVnrArea"] = original_data["MasVnrArea"].fillna(0)
```

In [26]:
```
# Replace MasVnrType missing with None
original_data["MasVnrType"] = original_data["MasVnrType"].fillna("None")
```

In [27]:
```
# Replace Electrical missing with most frequent value
original_data['Electrical'] = original_data['Electrical'].fillna(original_data['Electr
```

2.4.1 check the missing value

In [28]:
```
missing_pd = (original_data.isnull().sum() / original_data.shape[0] * 100).sort_values
missing_variables = missing_pd[missing_pd > 0].index
print(missing_variables.shape)
```

```
(0,)
```

2.5 2.5 Now we have 73 variables

2.5.1 2.5.1 Transform non-numerical labels to numerical labels

In [29]:
```
original_data['MSSubClass'] = original_data['MSSubClass'].astype('str')
original_data['OverallCond'] = original_data['MSSubClass'].astype('str')
original_data['YrSold'] = original_data['YrSold'].astype(str)
original_data['MoSold'] = original_data['MoSold'].astype(str)

### Laberl Encoding
s_variables = original_data.columns[(original_data.dtypes == 'object')]
for col in s_variables:
    lbl = LabelEncoder()
    lbl.fit(list(original_data[col].values))
    original_data[col] = lbl.transform(list(original_data[col].values))
```

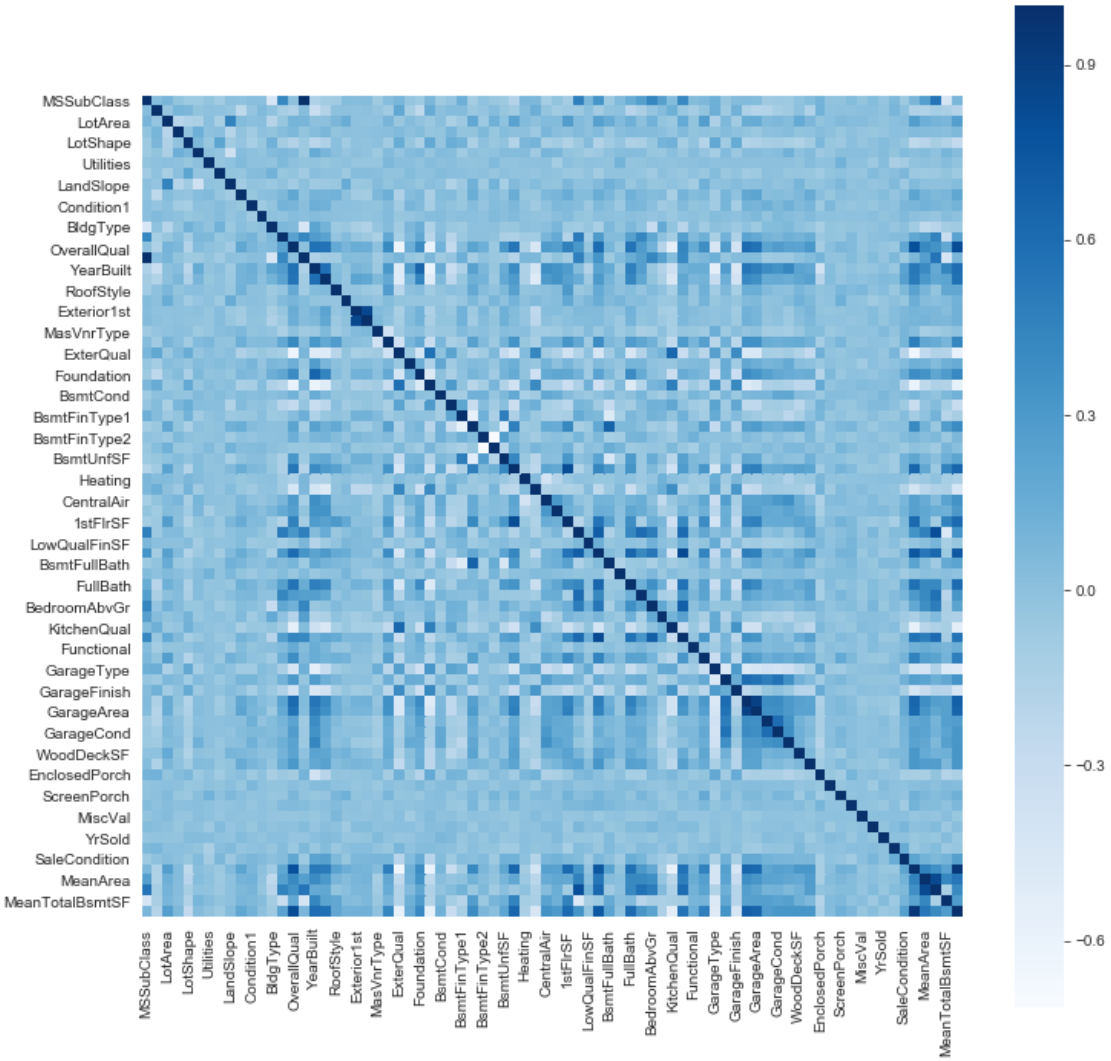
2.6 2.6 features engineering

In [30]:
```
original_data['MeanArea'] = original_data['MSSubClass'].map(
    original_data.groupby('MSSubClass')['GrLivArea', 'TotalBsmtSF'].mean().sum(1))
original_data['MeanGrLivArea'] = original_data['MSSubClass'].map(
    original_data.groupby('MSSubClass')['GrLivArea'].mean())
original_data['MeanTotalBsmtSF'] = original_data['MSSubClass'].map(
    original_data.groupby('MSSubClass')['TotalBsmtSF'].mean())
```

In [31]:
```
# generate the log sale_price
original_data['LogSalePrice'] = np.log2(original_data['SalePrice'])
```

In [32]:
```
# inspect the transformed data
train_data = original_data.iloc[:1200]

# correlation matrix
```

```
plt.subplots(figsize=(12, 12))
corr_x = train_data.corr()
sns.heatmap(corr_x, annot=False, vmax=1, square=True, cmap="Blues")

corr_x = train_data.corr()
# select the varibles that have correlation greater than 0.3
varibles = train_data.columns[abs(corr_x.LogSalePrice) > 0.4]

# extract the 21 variables data
train_data = train_data[varibles]
test_data = original_data.iloc[1200:][varibles]
```



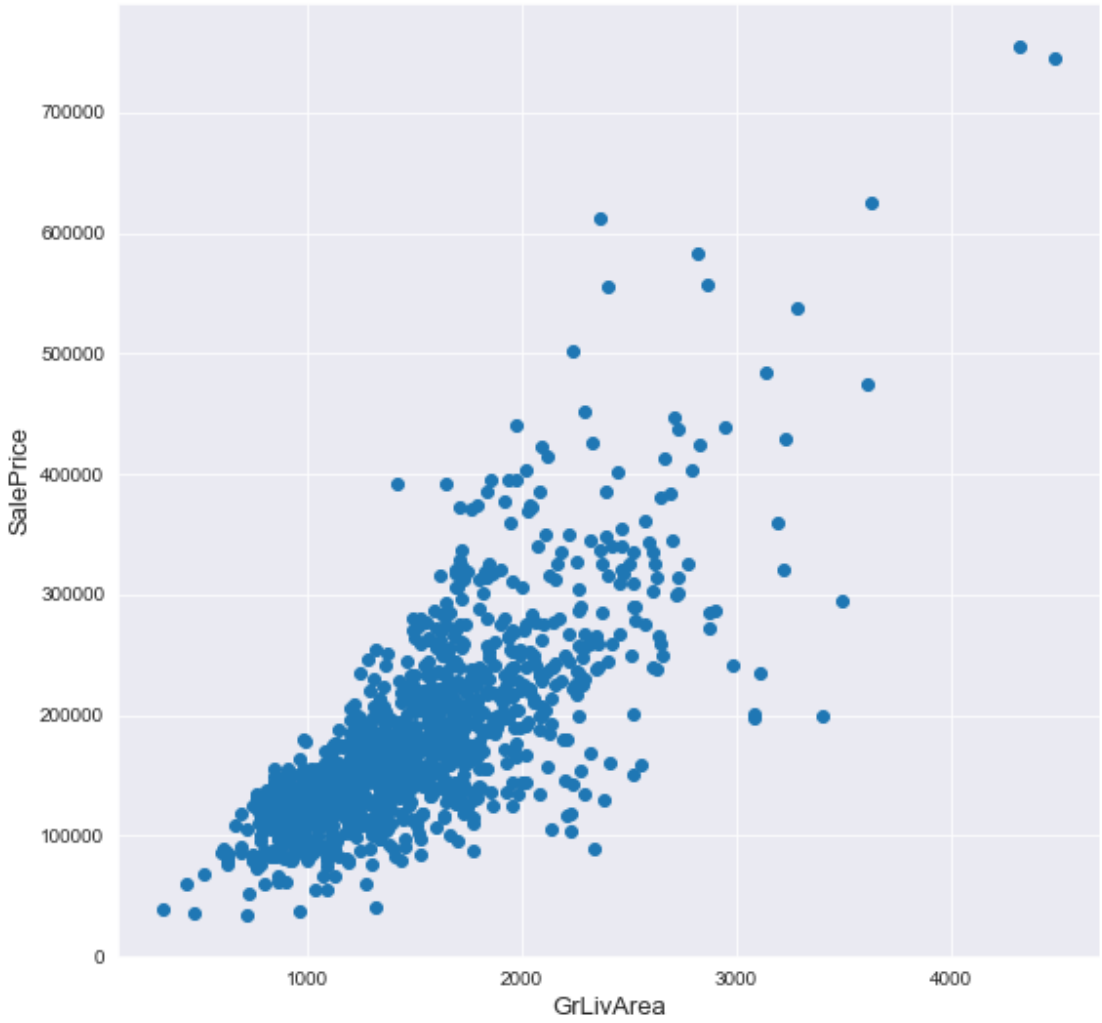
2.7 2.7 Explore the outliers

```
In [33]: fig, ax = plt.subplots(figsize=(9, 9))

         ax.scatter(x=train_data['GrLivArea'], y=train_data['SalePrice'])
         plt.ylabel('SalePrice', fontsize=13)
         plt.xlabel('GrLivArea', fontsize=13)
         plt.show()
```

2.7.1 2.7.1 Deleting outliers

```
In [36]: train_data = train_data.drop(train_data[(train_data['GrLivArea'] > 4000) & (train_data
         # Check the graphic again
         fig, ax = plt.subplots(figsize=(9, 9))
         ax.scatter(train_data['GrLivArea'], train_data['SalePrice'])
```

```
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



2.7.2 2.7.2 Explore another outliers

```
In [37]: fig, ax = plt.subplots(figsize=(9, 9))
         ax.scatter(x=train_data['TotalBsmtSF'], y=train_data['SalePrice'])
         plt.ylabel('SalePrice', fontsize=13)
         plt.xlabel('TotalBsmtSF', fontsize=13)
         plt.show()
```

2.7.3 2.7.3 Deleting outliers

```
In [38]: train_data = train_data.drop(train_data[(train_data['TotalBsmtSF'] > 3000) & (train_da
```

```
In [39]: # Check the graphic again
         fig, ax = plt.subplots(figsize=(9, 9))
         ax.scatter(train_data['TotalBsmtSF'], train_data['SalePrice'])
         plt.ylabel('SalePrice', fontsize=13)
         plt.xlabel('TotalBsmtSF', fontsize=13)
         plt.show()
```

2.8 2.8 Get traing data and test data

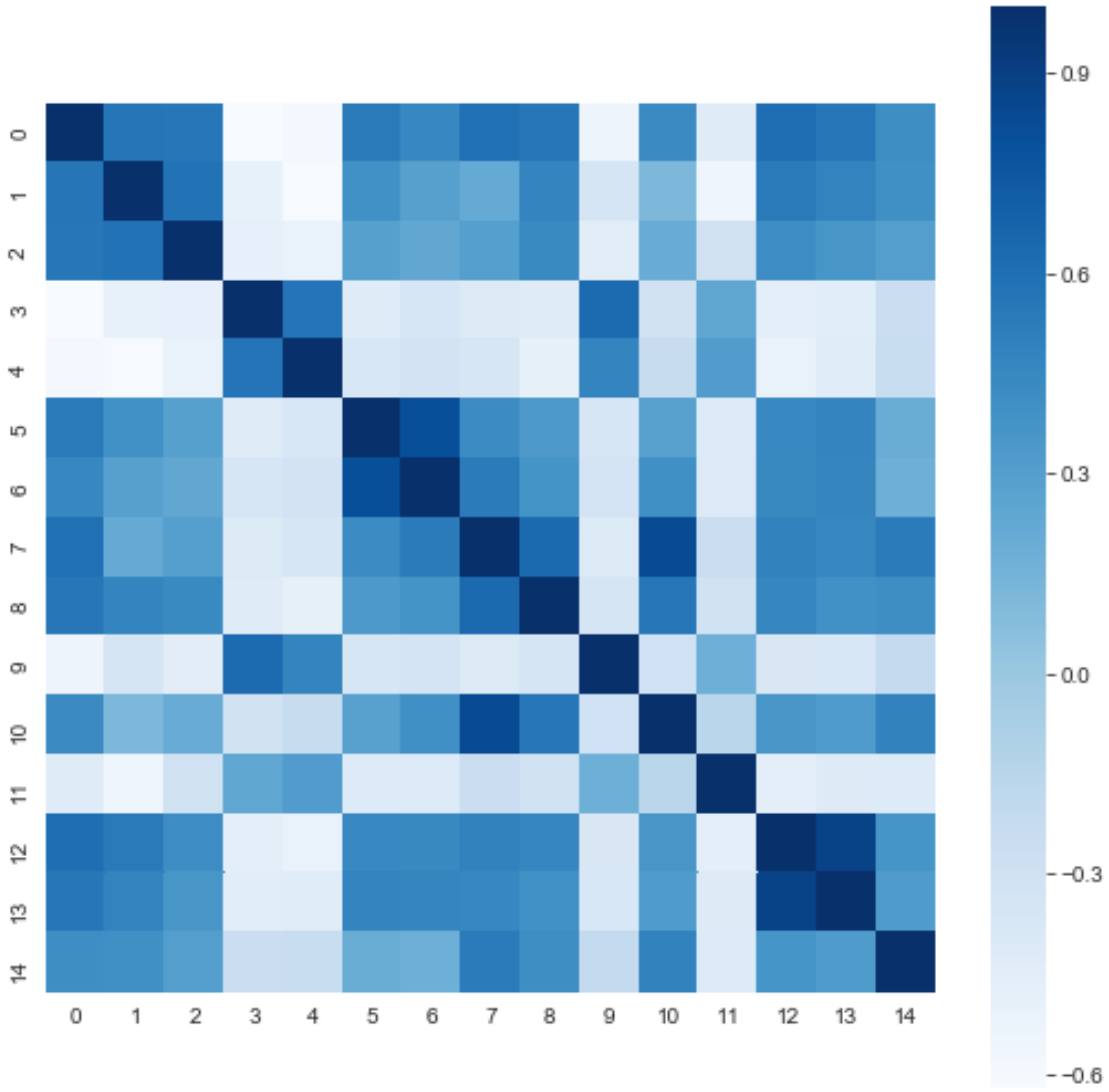
```
In [40]: train_y = train_data['LogSalePrice']
         train_x = train_data.drop(['LogSalePrice', 'SalePrice'], axis=1)

         test_y = test_data['LogSalePrice']
         test_x = test_data.drop(['LogSalePrice', 'SalePrice'], axis=1)
```

3 3. Features selection

```
In [81]: X_new = SelectKBest(f_regression, k=15).fit_transform(train_x, train_y)
         features = []
         j = 0
         for i in range(train_x.shape[1]):
```

```
            if all(X_new[:, j] == train_x.iloc[:, i]):
                features.append(train_x.columns[i])
                j += 1
                if j == 15:
                    break
        ##### now we have 15 features
```

In [84]:
```
df = pd.DataFrame(X_new)
plt.subplots(figsize=(9, 9))
correlations = df.corr()
sns.heatmap(correlations, annot=False, vmax=1, square=True, cmap="Blues")
plt.savefig("correlation.jpg")
plt.show()
```



4 4. Fit the models

In [42]:
```
train_x = train_x[features]
test_x = test_x[features]

outcome = {
    'RF': pd.DataFrame(),
    'OLS': pd.DataFrame(),
    'RIDGE': pd.DataFrame(),
    'LASSO': pd.DataFrame()
}
```

In [43]:
```
def models(train_x, train_y):

    ########################
    ##### RandomForest #####
    ########################
    # Random Forest Regressor parameters
    rf_params = {
        'n_jobs': -1,
        'n_estimators': 300,
        'warm_start': True,
        'max_depth': 15,
        'min_samples_leaf': 9,
        'max_features': 'log2',
        'verbose': 0
    }
    model = GridSearchCV(RandomForestRegressor(**rf_params),
                         {'max_depth': [10, 12, 14],
                          'min_samples_leaf': [1, 2]},
                         n_jobs=-1, cv=5)
    model_1 = model.fit(train_x, train_y)

    model = RandomForestRegressor(**rf_params)
    model = model.set_params(**model_1.best_params_)
    model_1 = model.fit(train_x, train_y)

    m1 = mean_squared_error(test_y, model_1.predict(test_x))

    df_1 = pd.Series(model_1.feature_importances_, index=train_x.columns)
    df_1['mean_squared_error'] = m1

    ###############
    ##### OLS #####
    ###############
    ols_params = {
        'copy_X': True,
        'fit_intercept': True,
```

```
    'n_jobs': None,
    'normalize': False
}
model = LinearRegression(**ols_params)
model_2 = model.fit(train_x, train_y)

m2 = mean_squared_error(test_y, model_2.predict(test_x))

df_2 = pd.Series(model_2.coef_, index=train_x.columns)
df_2['mean_squared_error'] = m2

#################
##### Ridge #####
#################
rid_params = {
    'alpha': 0.3,
    'copy_X': True,
    'fit_intercept': True,
    'normalize': False
}
model = GridSearchCV(Ridge(**rid_params),
                     {'alpha': [40, 50, 60]},
                     n_jobs=-1, cv=5)
model_3 = model.fit(train_x, train_y)

model = Ridge(**rid_params)
model = model.set_params(**model_3.best_params_)
model_3 = model.fit(train_x, train_y)

m3 = mean_squared_error(test_y, model_3.predict(test_x))

df_3 = pd.Series(model_3.coef_, index=train_x.columns)
df_3['mean_squared_error'] = m3

#################
##### Lasso #####
#################
las_params = {
    'alpha': 0.1,
    'copy_X': True,
    'fit_intercept': True,
    'normalize': False
}
model = GridSearchCV(Lasso(**las_params),
                     {'alpha': [0.004, 0.005, 0.006]},
                     n_jobs=-1, cv=5)
model_4 = model.fit(train_x, train_y)
```

```
        model = Lasso(**las_params)
        model = model.set_params(**model_4.best_params_)
        model_4 = model.fit(train_x, train_y)

        m4 = mean_squared_error(test_y, model_4.predict(test_x))

        df_4 = pd.Series(model_4.coef_, index=train_x.columns)
        df_4['mean_squared_error'] = m4

        fitted_models = [model_1, model_2, model_3, model_4]

        return [df_1, df_2, df_3, df_4], fitted_models
```

```
In [45]: dfs, fitted_models= models(train_x, train_y)
         outcome['RF'] = outcome['RF'].append(dfs[0].to_frame().T)
         outcome['OLS'] = outcome['OLS'].append(dfs[1].to_frame().T)
         outcome['RIDGE'] = outcome['RIDGE'].append(dfs[2].to_frame().T)
         outcome['LASSO'] = outcome['LASSO'].append(dfs[3].to_frame().T)
```

5 5. Use bootstrap to get the variance of the estimators of the parameters

```
In [47]: # bootstrap in model
         N = 100   # for getting the varaince, 100-200 times paths are enough, as for percenti
         for i in range(N):
             x, y = resample(train_x, train_y)
             dfs, _ = models(x, y)
             outcome['RF'] = outcome['RF'].append(dfs[0].to_frame().T)
             outcome['OLS'] = outcome['OLS'].append(dfs[1].to_frame().T)
             outcome['RIDGE'] = outcome['RIDGE'].append(dfs[2].to_frame().T)
             outcome['LASSO'] = outcome['LASSO'].append(dfs[3].to_frame().T)
```

```
In [48]: results = {}
         models_name = ['RF', 'OLS', 'RIDGE', 'LASSO']
         for m in models_name:
             results[m] = pd.concat((outcome[m].iloc[0], outcome[m].iloc[1:].std()), 1)
             results[m].columns = [['features', 'std']]
             results[m] *= 1000
             results[m] = results[m].round(2)
         result = pd.concat(results, 1)
         result[['RF', 'OLS', 'RIDGE', 'LASSO']].to_csv('result.csv')
```

```
In [86]: result
```

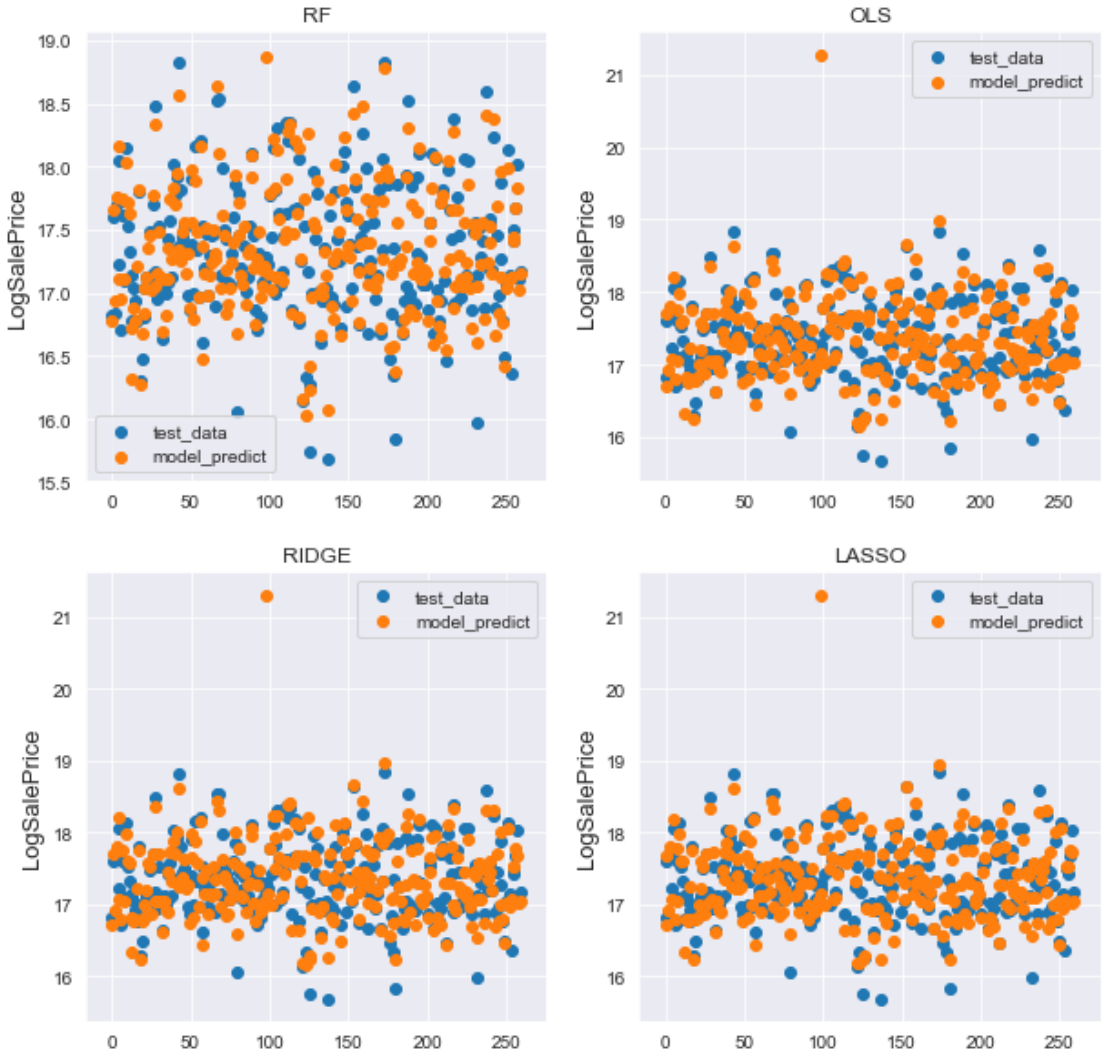
Out[86]:

	LASSO		OLS		RF		RIDGE	\
	features	std	features	std	features	std	features	
OverallQual	108.84	9.12	113.90	9.37	168.99	11.95	107.08	
YearBuilt	1.92	0.39	1.93	0.40	102.01	13.45	1.94	

YearRemodAdd	3.45	0.45	3.36	0.46	47.97	8.98	3.43
ExterQual	0.00	4.59	16.37	14.85	52.21	13.07	10.74
BsmtQual	-10.16	7.19	-15.41	7.56	34.52	7.74	-15.30
TotalBsmtSF	0.24	0.03	0.24	0.03	77.01	10.82	0.24
1stFlrSF	0.06	0.04	0.06	0.04	61.35	6.83	0.06
GrLivArea	0.37	0.02	0.39	0.03	171.67	12.89	0.39
FullBath	-0.00	13.26	-34.32	18.83	44.48	12.19	-25.81
KitchenQual	-12.87	9.37	-25.71	9.39	25.47	8.80	-23.68
TotRmsAbvGrd	-3.07	6.09	-6.92	7.82	25.49	2.73	-7.31
GarageType	-10.46	4.18	-11.50	4.20	34.53	10.78	-11.87
GarageCars	0.00	7.95	26.57	21.51	60.95	11.17	20.01
GarageArea	0.32	0.04	0.23	0.06	65.66	10.38	0.26
MeanArea	0.12	0.03	0.12	0.03	27.69	3.90	0.12
mean_squared_error	107.45	3.92	104.55	4.15	51.54	1.79	105.90

	std
OverallQual	8.84
YearBuilt	0.39
YearRemodAdd	0.45
ExterQual	12.77
BsmtQual	7.18
TotalBsmtSF	0.03
1stFlrSF	0.04
GrLivArea	0.03
FullBath	14.95
KitchenQual	8.48
TotRmsAbvGrd	7.46
GarageType	4.15
GarageCars	15.73
GarageArea	0.05
MeanArea	0.03
mean_squared_error	4.02

In [85]:
```
fig, axes = plt.subplots(2, 2, figsize=(10, 10))
for i, val in enumerate(models_name):
    axes[i//2, i%2].scatter(range(260), test_y, label="test_data")
    axes[i//2, i%2].scatter(range(260), fitted_models[i].predict(test_x), label="model
    axes[i//2, i%2].set_ylabel('LogSalePrice', fontsize=13)
    axes[i//2, i%2].set_title(val)
    axes[i//2, i%2].legend()
plt.savefig("Comparison_of_test_data_with_predicted_data.jpg")
plt.show()
```



In [ ]: