Data analysis report

孙晨昕 19336117

中山大学数学学院统计系

Contents

1 Introduction **2**

1.1 Background 2

1.2 Goal 2

1.3 Methods 2

2 Data pre-processing **2**

2.1 process on target variable 2

2.1.1 Log-transformation of the target variable 3

2.2 Process on Missing values 5

2.2.1 Transforming on some categorical variables 6

2.3 Check the skew of all numerical features 6

2.4 Getting dummy categorical features 7

3 Base Model **7**

3.1 Elastic Net 7

3.2 Gradient Boosting 9

3.3 Xgboost 9

3.3.1 Interpretation of the model 10

4 Stacking model **12**

5 Conclusion **12**

5.1 Limitations 12

5.2 What i have gained 12

6 Appendix **13**

1 Introduction

1.1 Background

Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

1.2 Goal

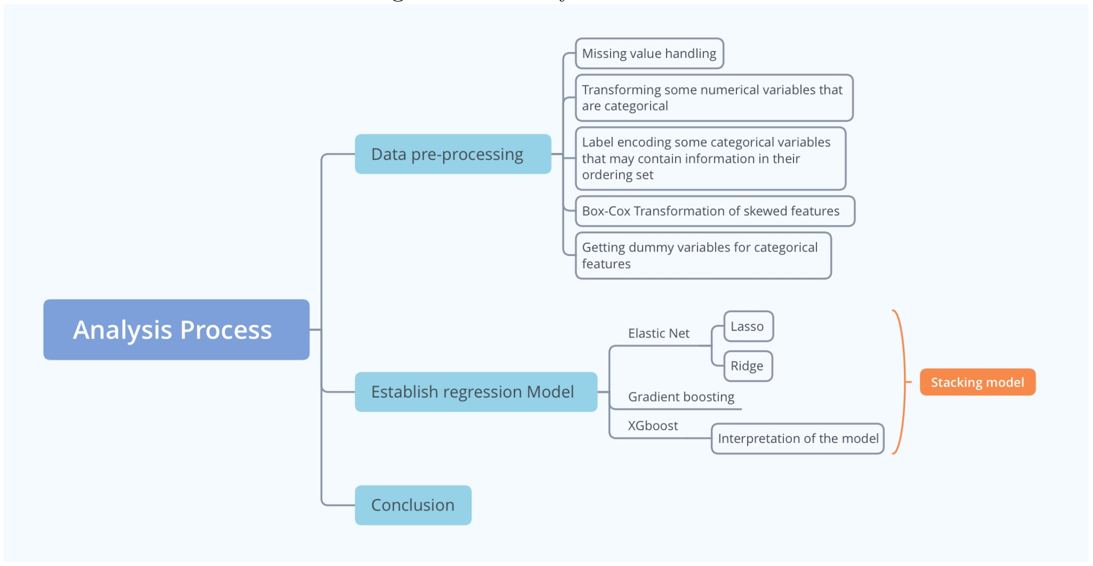
With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, we want to predict the final price of each home.

we have a data_description.txt containing full description of each column(variables).

1.3 Methods

In this analysis, we compare different regression models in predicting the house price and present a great prediction performance using stacking model along with an awesome interpretation using SHAP. The XGboost model shows that the gross area, the overall material used, the total square feet of basement area are the most important features that influence the house price.

Figure 1-1 Analysis flow chart

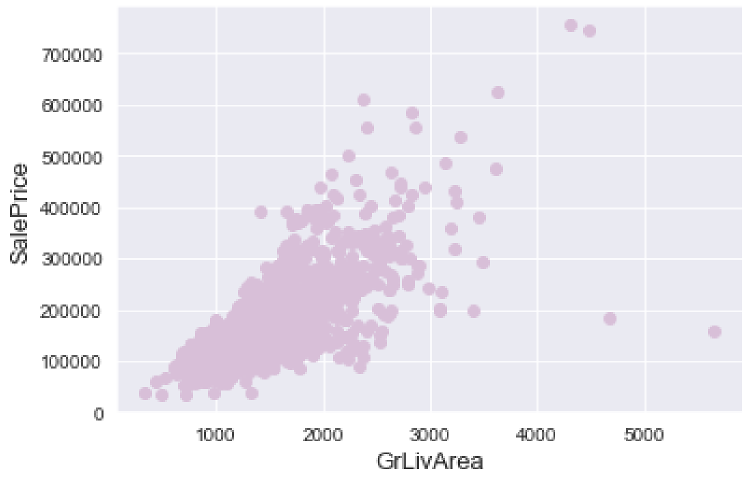

Analysis Process
Data pre-processing
Missing value handling
Transforming some numerical variables that are categorical
Label encoding some categorical variables that may contain information in their ordering set
Box-Cox Transformation of skewed features
Getting dummy variables for categorical features
Establish regression Model
Elastic Net
Lasso
Ridge
Gradient boosting
XGboost
Interpretation of the model
Stacking model
Conclusion

2 Data pre-processing

2.1 process on target variable

In our cognition, housing area is the decisive factor of housing price, so we first draw a scatter plot of the "SalePrice" with respect to "GrLivArea"

Figure 2-2 price.area.scatter.plot



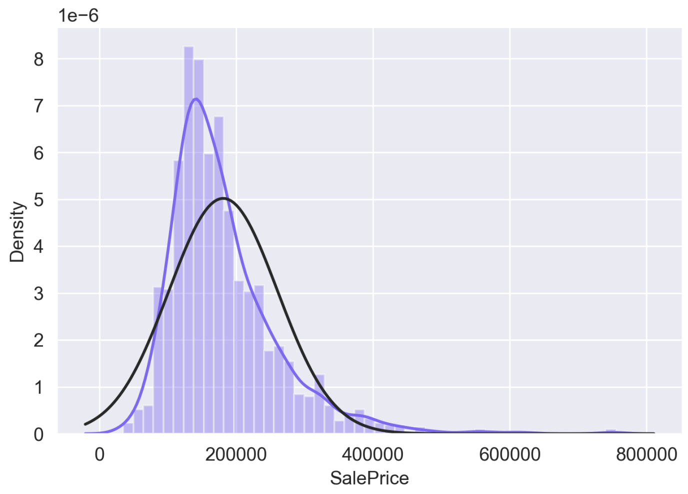
Here we observe 2 outliers with big(>4000) 'GrLivArea' and low 'SalePrice',which is apparantly impossible, so we delete them.

Now the Price and area present positive correlation, which we should expect.

2.1.1 Log-transformation of the target variable

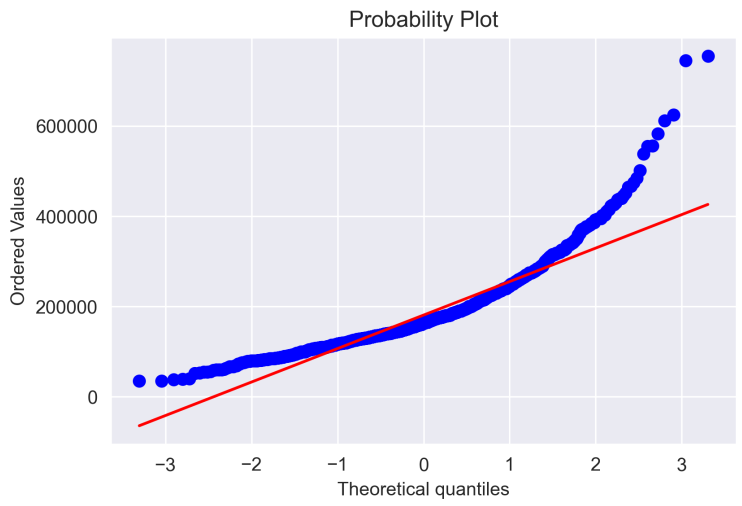
Then, let's look at the distribution that the y variable obeys

Figure 2-3 price.distribution



and the Q-Q plot

Figure 2-4 Q-Q plot

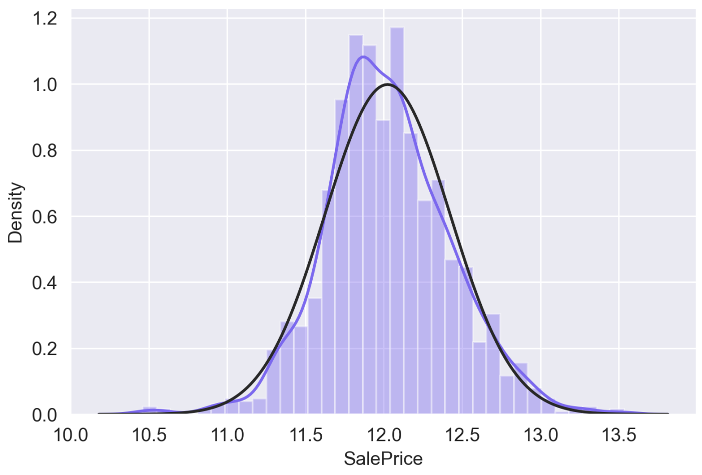


target variable is right skewed. As (linear) models love normally distributed data , we need to transform this variable and make it more normally distributed.

Solution:

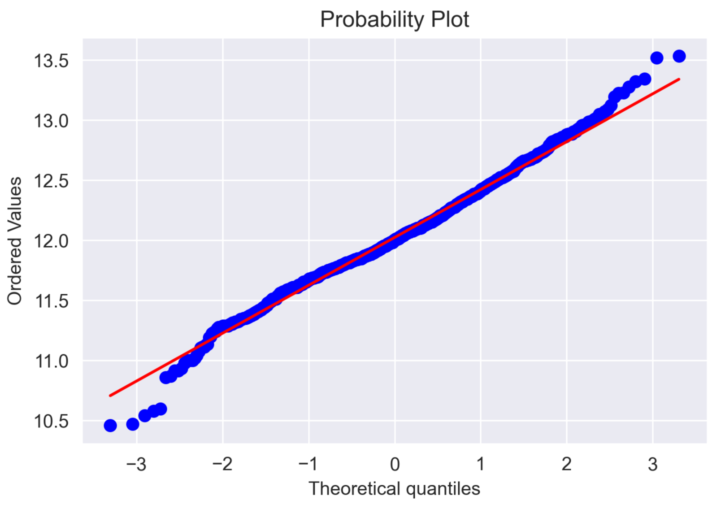
We use the numpy fuction log1p which applies log(1+x) to all elements of the column then we check the distribution again:

Figure 2-5 price.distribution



and Q-Q plot

Figure 2-6 price.distribution



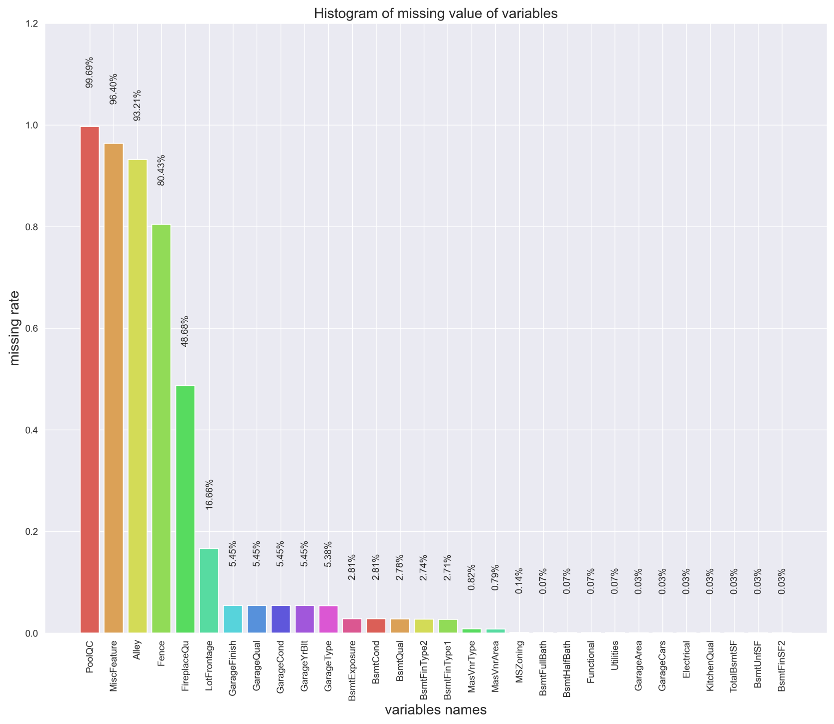
The skew seems now corrected and the data appears more normally distributed.

2.2 Process on Missing values

In order to preprocess the variables of the two datasets 'test' and 'train' in the same way, we first concatenate them into one 'all_data'

Then we calculate the missing ratio of each variable withing missing values, the result is visualized.

Figure 2-7 price.distribution



Now let's fill the NA values:

- **PoolQC** : data description says NA means "No Pool". That make sense, given the huge ratio of missing value (+99%) and majority of houses have no Pool at all in general.
- Similarly, we can do the same for "MiscFeature","Alley","Fence","FireplaceQu"...See appendix for specific operations
- **LotFrontage** : Since the area of each street connected to the house property most likely have a similar area to other houses in its neighborhood , we can fill in missing values by the median LotFrontage of the neighborhood.
- **MSZoning** (The general zoning classification) : 'RL' is by far the most common value. So we can fill in missing values with 'RL'

2.2.1 Transforming on some categorical variables

Transforming some numerical variables that are really categorical

Label Encoding some categorical variables that may contain information in their ordering set

2.3 Check the skew of all numerical features

After obtaining the variables with skewness, we want to eliminate skewness and other distributional features that complicate analysis. Our goal is to find a simple transformation that leads to normality.

Solution:

We use the scipy function boxcox1p which computes the Box-Cox transformation of 1+x .

2.4 Getting dummy categorical features

Since the R package *glmnet* that would be used can only accept numerical matrices as model inputs, we get the dummy variables and obtain the new 'train' and 'test' data.

3 Base Model

3.1 Elastic Net

For linear models, complexity is directly related to the number of variables of the model, and the more variables, the higher the complexity of the model. More variables can often give a seemingly better model when fitting, but at the same time, it is also faced with the danger of over-fitting.

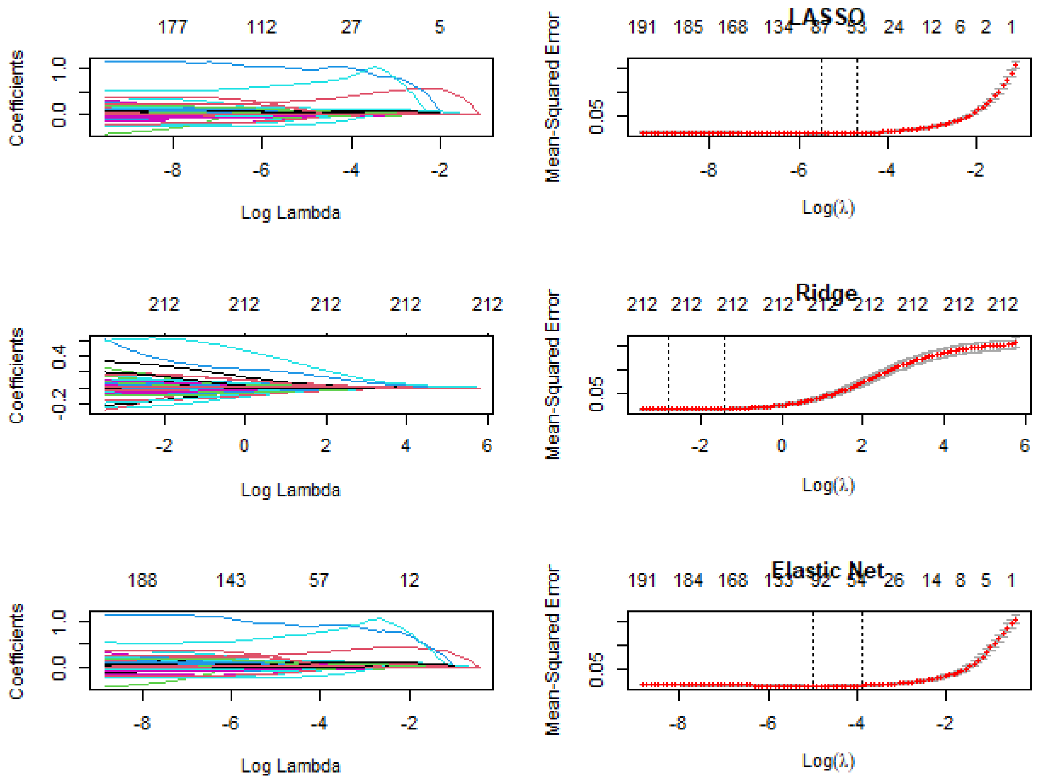
Model complexity is controlled by parameter λ, the larger λ is, the more punishment will be imposed on the linear model with more variables, thus ultimately obtaining a model with fewer variables.

Another parameter, α, controls for traits in models dealing with highly correlated data.

LASSO regression: $\alpha = 1$,The Ridge regression： $\alpha = 0$,and General Elastic Net model:$0 < \alpha < 1$

- We randomly split 'train' data into train (2/3) and test (1/3) sets
- **Cross validation** is used to fit and select the model, and a more accurate estimate of the performance of the model is obtained.
- Set the target parameter to be minimized as MSE when selecting the cross-validation model.
- **Parallel computation** is used here to enhance the operation efficiency.

Through R Package "glmnet", we obtain the best λ value for $\alpha = 0(Ridge), 0.1, 0.2, ..., 1(LASSO)$, and can plot solution path and cross-validated MSE as function of λ



and the MSE on test set of each α:

mse0	0.01602403
mse1	0.01425285
mse2	0.01393018
mse3	0.01370086
mse4	0.01347885
mse5	0.01395351
mse6	0.01417041
mse7	0.01364376
mse8	0.01344479
mse9	0.01407777
mse10	0.01358188

Table 3-1 mse'i' means $\alpha = i/10$

We can see that LASSO regression got a good result, but not the best with the min MSE on $\alpha = 8/10 = 0.8$
Also chech the R^2 for each fit:

	Rsquare for each fit
fit0	0.7141264
fit1	0.7161421
fit2	0.6909838
fit3	0.6847667
fit4	0.6916455
fit5	0.6798777
fit6	0.6924022
fit7	0.6977265
fit8	0.6925958
fit9	0.6926444
fit10	0.6805612

Table 3-2 Rsquare

We are still not satisfied with this R^2 values, so we move on.

3.2 Gradient Boosting

Boosting algorithm is one of the ensembling learning, which is composed of weak learners and passes multiple weak learners.

Predictions are made based on the results of multiple weak learners

The underlying idea of ensemble learning is that even if one weak classifier gets a wrong prediction, other weak classifiers can correct the error.

As a decision tree algorithm based on iterative superposition, GBDT mainly combines a number of different weak learners, and the corresponding prediction result is regarded as the final prediction result.

Since MAE is hard to calculate while MSE is sensitive to outliers, we here choose a 'Huber loss' (which has the advantages of both MSE and MAE that reduces the sensitivity of outliers and realizes the function of differentiating everywhere) to evaluate the model.

Let's see how this model performs on the data by evaluating the cross-validation mse error and R^2.

$$GradientBoostingscore : 0.01385$$
$$\text{R square: } 0.985$$

Wow, we can see that the R^2 has been significantly increased,which means that the fitting degree of the regression line to the observed value is higher.

3.3 Xgboost

Xgboost refers to eXtreme Gradient Boosting, which belongs to the improved gradient lifting algorithm after optimization .

Perform a second order Taylor expansion on the cost function and explicitly add regular terms to control the complexity of a model, then we get the Xgboost

Let's see how it performs:

$$Xgboost score : 0.0138$$
R square: 0.961

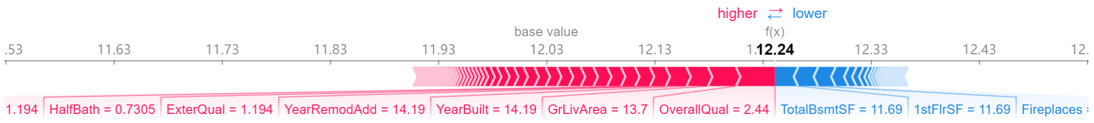
3.3.1 Interpretation of the model

We want to see which variables have a big impact on housing prices, and how.

Solution:

We utilize SHAP value to calculate the feature importance shown below:

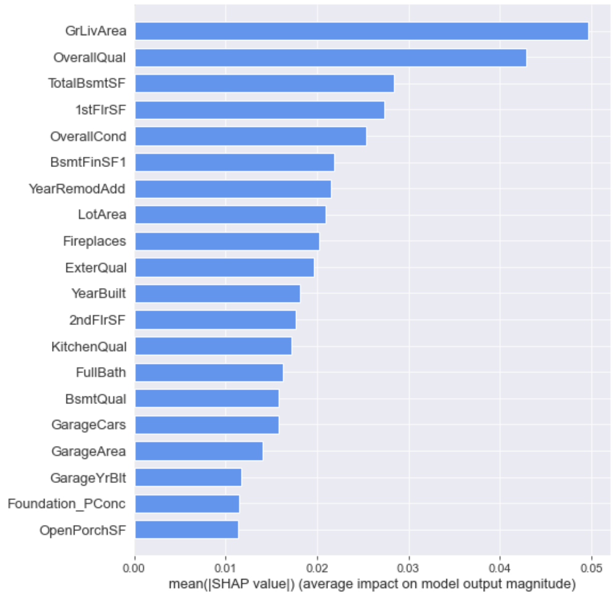
Figure 3-8 Explanation of single prediction



The figure above shows that each feature has its own contribution, which pushes the prediction result of the model from the base value to the final model output. Features that pushed the forecast higher are shown in red, and features that pushed the forcast lower are shown in blue .

and it is very clear to observe the degree of influence of each variable on the house price ranking by barplot.

Figure 3-9 Feature contribution ranking



Here we can clearly see that this following factors are playing a great role on house price determination.

- **GrLivArea** : Above grade (ground) living area square feet, the determining factor of house price, which is consistent with common sense.
- **OverallQual**:Rates the overall material and finish of the house
- **TotalBsmtSF**: Total square feet of basement area

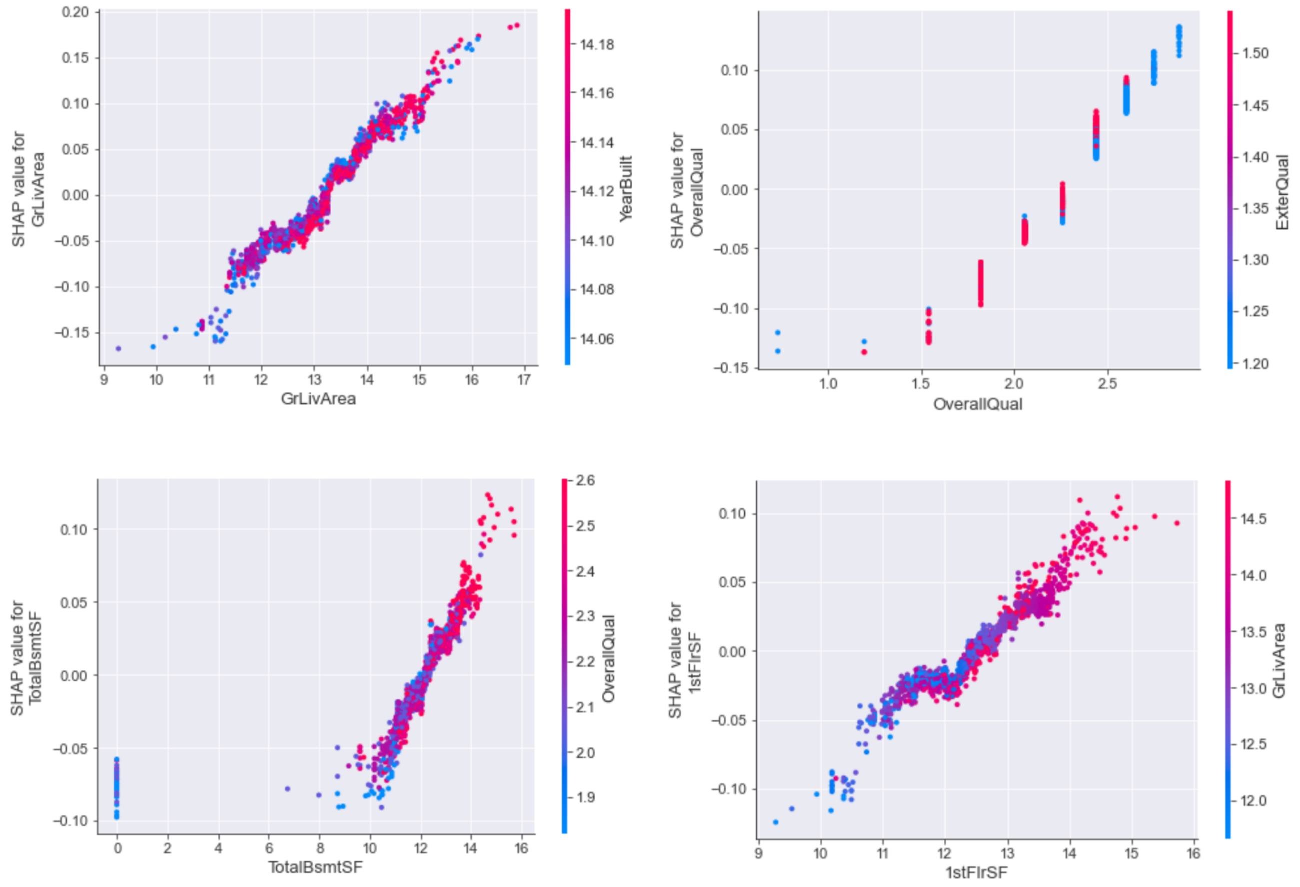
- **1stFlrSF**: First Floor square feet
- **OverallCond**: Rates the overall condition of the house

We can also get some interesting results,like

- The basement really counts, the square feet and rating of it have a high degree of discretion over house prices
- The grade of **Bath** and **Kitchen**, the condition of **Garage** are also vital.
- The square feet of first floot is more significant than second floor.

Then we choose some top important varibale to investigate the dependence of the model on each feature by dependence plot which will show the interaction of this variable with other.

Figure 3-10 Dependence plot of the top 4 important variable



we can see that

- With the change of times, the size of the house is getting bigger and bigger
- A larger basement always means the higher grade of the overall material
- The above grade (ground) living area square feet is mainly determined by the ratio of first floor area.

4 Stacking model

The Stacking method is the practice of training one model to combine other models. First we train a number of different models, and then train a model with the outputs of the previously trained models as inputs to get a final output.

We now try to use a simplest stacking approach : Averaging base models to predict the house price.

We just average four models here ElasticNet, XGBoost, KRR and lasso. And we obtain the mse of this averaged model.

$$Averagedbasemodelsscore : 0.0119$$

R square: 0.960

Fantastic! We can see that this simplest stacking approach really improve the performance.

Therefore, we choose this model to obtain our final prediction.

5 Conclusion

In this analysis we examined four different models to predict the house price given many features and Stacking model achieved the best performance. Using Shapley Value, we present awesome interpretation of XGBoost about how each feature influence the house price.

5.1 Limitations

Due to my poor knowledge of Machine Learning , this report only realize the simplest stacking approach, attempt would be made in the following days to fit more complex stacking models.

Also, the unfamiliarity of Python and data analysis report writing also lead to those constantly emerging mistakes and inappropriate expression in this report, which i am really sorry for.

5.2 What i have gained

The baseline is imitated from a notebook released on kaggle.

However, learning from the baseline, exloring and coding on my own while searching answering from the Internet really benefitted me a lot. Through this report, here is the brief summary of what i learned.

- How to pre-process (include visualizing, handling missing values, normalizing and processing classified data)the data, and make them available for the model we need to build next.
- How to fit regression models like lasso,Ridge,during this process, also have a general idea of how the Elastic Nets work, how to use R to obtain the best λ value.
- What **Gradient Boosting** is and how it works by obtaining a function estimating the residuals between the present model and the known target variable, what the difference and relationship between GBDT and XGboost are.
- How to use **SHAP** to interpret the model and how to plot them.
- The basic idea and realization method of **stacking model**.
- How to use **grid search** to find optimal hyperparameters.

6 Appendix

# R codes

```
library(xlsx)
library(MASS)  # Package needed to generate correlated precictors
library(glmnet)  # Package to fit ridge/lasso/elastic net models
library(doParallel) #Package to parellel computation
#d is the train data
d=read.xlsx('C:/Users/10048/Desktop/导出结果.xlsx',sheetIndex =1)
d=d[,-1]
#new is the test data
new=read.xlsx('C:/Users/10048/Desktop/test.xlsx',sheetIndex=1)
new=new[,-1]

n=length(d[,1])
x<- d[,1:219 ]
y<- d[,220]

#split the data into train and test
 train_rows <- sample(1:n, .66*n)
x.train <- x[train_rows, ]
x.test <- x[-train_rows, ]

y.train <- y[train_rows]
y.test <- y[-train_rows]

x_matrices <- glmnet::makeX(train = x.train,test=x.test)

#use parallel computation
 cl<-makeCluster(14)
 registerDoParallel(cl)
 for (i in 0:10)
   {
   assign(paste("fit", i, sep=""), cv.glmnet(x_matrices$x, y.train,
       type.measure="mse", alpha=i/10,family="gaussian",paralle=T))
   }
 stopCluster(cl)

#plot
 par(mfrow=c(3,2))
 plot(fit.lasso, xvar="lambda")
 plot(fit10, main="LASSO")

plot(fit.ridge, xvar="lambda")
plot(fit0, main="Ridge")

plot(fit.elnet, xvar="lambda")
plot(fit5, main="Elastic␣Net")

x.test=x_matrices$xtest

#predict on test set
yhat0 <- predict(fit0, s=fit0$lambda.1se, newx=x.test)
yhat1 <- predict(fit1, s=fit1$lambda.1se, newx=x.test)
yhat2 <- predict(fit2, s=fit2$lambda.1se, newx=x.test)
yhat3 <- predict(fit3, s=fit3$lambda.1se, newx=x.test)
yhat4 <- predict(fit4, s=fit4$lambda.1se, newx=x.test)
yhat5 <- predict(fit5, s=fit5$lambda.1se, newx=x.test)
yhat6 <- predict(fit6, s=fit6$lambda.1se, newx=x.test)
yhat7 <- predict(fit7, s=fit7$lambda.1se, newx=x.test)
yhat8 <- predict(fit8, s=fit8$lambda.1se, newx=x.test)
yhat9 <- predict(fit9, s=fit9$lambda.1se, newx=x.test)
yhat10 <- predict(fit10, s=fit10$lambda.1se, newx=x.test)

#calculate mse of each alpha
mse0 <- mean((y.test - yhat0)^2)
mse1 <- mean((y.test - yhat1)^2)
mse2 <- mean((y.test - yhat2)^2)
mse3 <- mean((y.test - yhat3)^2)
mse4 <- mean((y.test - yhat4)^2)
mse5 <- mean((y.test - yhat5)^2)
mse6 <- mean((y.test - yhat6)^2)
mse7 <- mean((y.test - yhat7)^2)
mse8 <- mean((y.test - yhat8)^2)
mse9 <- mean((y.test - yhat9)^2)
mse10 <- mean((y.test - yhat10)^2)

mse=rbind(mse0,mse1,mse2,mse3,mse4,mse5,mse6,mse7,mse8,mse9,mse10)

#can see the rsquare of each model
mse=rbind(mse0,mse1,mse2,mse3,mse4,mse5,mse6,mse7,mse8,mse9,mse10)
sse=mse*n
SSTO=sum((y.test-mean(y.test))^2)
SSR=SSTO-sse
rsquare=SSR/SSTO

```

```
#can see the coefficent of each variables in Lasso regression
coef(fit10)

#the prediction of new data(exactly test data)
x_new<- glmnet::makeX(train = new)

yhat=predict(fit10, s=fit10$lambda.1se, newx=x_new)
pre=exp(yhat)-1

#show the final prediction result of the test data
print(pre)
```

Python codes

[1]:
```
#import some necessary librairies

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
%matplotlib inline
import matplotlib.pyplot as plt  # Matlab-style plotting
import seaborn as sns
color = sns.color_palette('hls',12)
sns.set_style('darkgrid')
import warnings
def ignore_warn(*args, **kwargs):
    pass
warnings.warn = ignore_warn #ignore annoying warning (from sklearn and seaborn)
from scipy import stats
from scipy.stats import norm, skew #for some statistics


pd.set_option('display.float_format', lambda x: '{:.3f}'.format(x)) #Limiting␣
 ↪floats output to 3 decimal point
```

[2]:
```
train=pd.read_csv('D:/SYSU/21-22third_grade/ / /HW/HW6/input/train.csv')
test=pd.read_csv('D:/SYSU/21-22third_grade/ / /HW/HW6/input/test.csv')
```

[3]:
```
train.head(5)
```

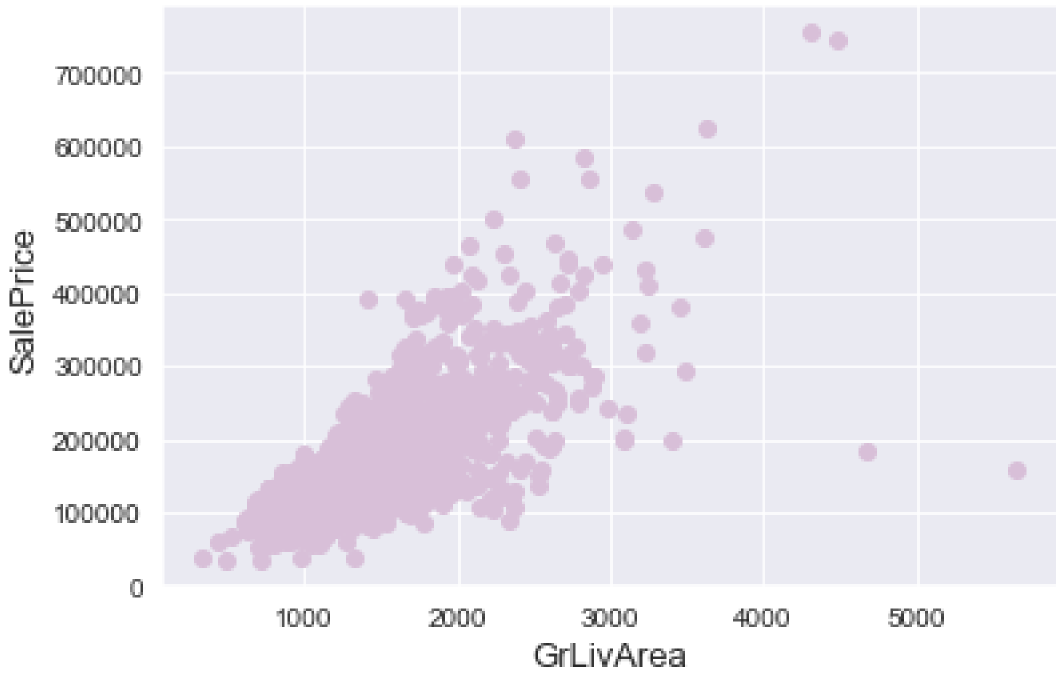
[3]:
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL       65.000     8450   Pave   NaN      Reg
1   2          20       RL       80.000     9600   Pave   NaN      Reg
2   3          60       RL       68.000    11250   Pave   NaN      IR1
3   4          70       RL       60.000     9550   Pave   NaN      IR1
4   5          60       RL       84.000    14260   Pave   NaN      IR1

  LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000
```

```
[5 rows x 81 columns]
```

[4]:
```
train_ID=train['Id']
test_ID=test['Id']
```

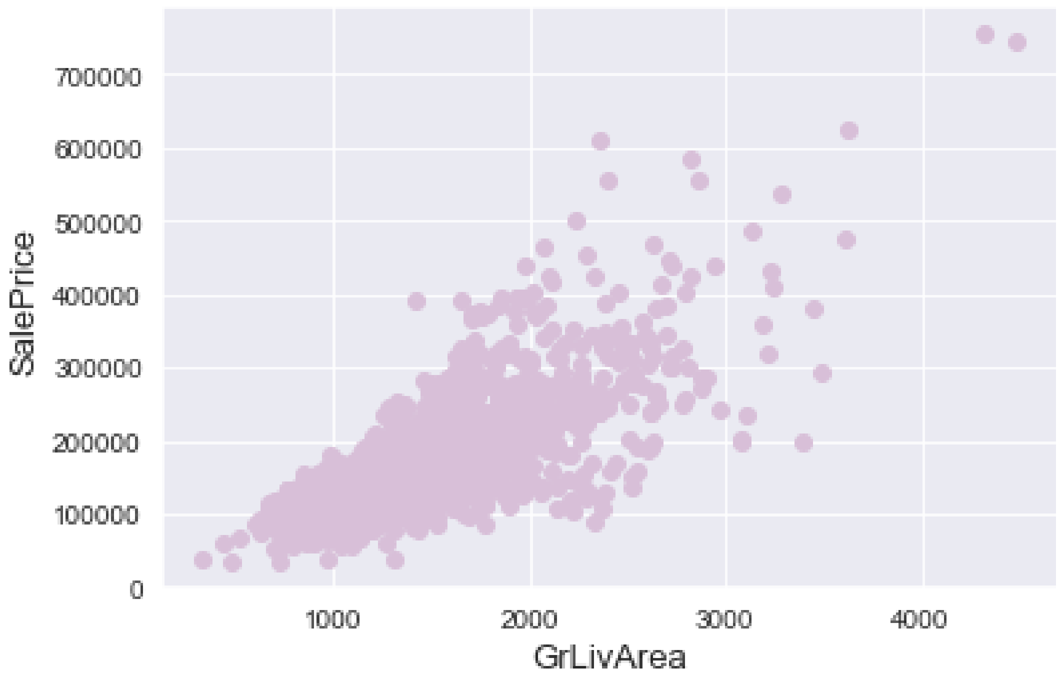
[5]:
```
train.drop("Id",axis=1,inplace=True)
test.drop("Id",axis=1,inplace=True)
#drop the "Id" column since it is unnecessary for the prediction process.
```

[6]:
```
fig,ax=plt.subplots()
ax.scatter(x=train['GrLivArea'],y=train['SalePrice'],color="#D8BFD8")
plt.ylabel('SalePrice',fontsize=13)
plt.xlabel('GrLivArea',fontsize=13)
plt.show()
```



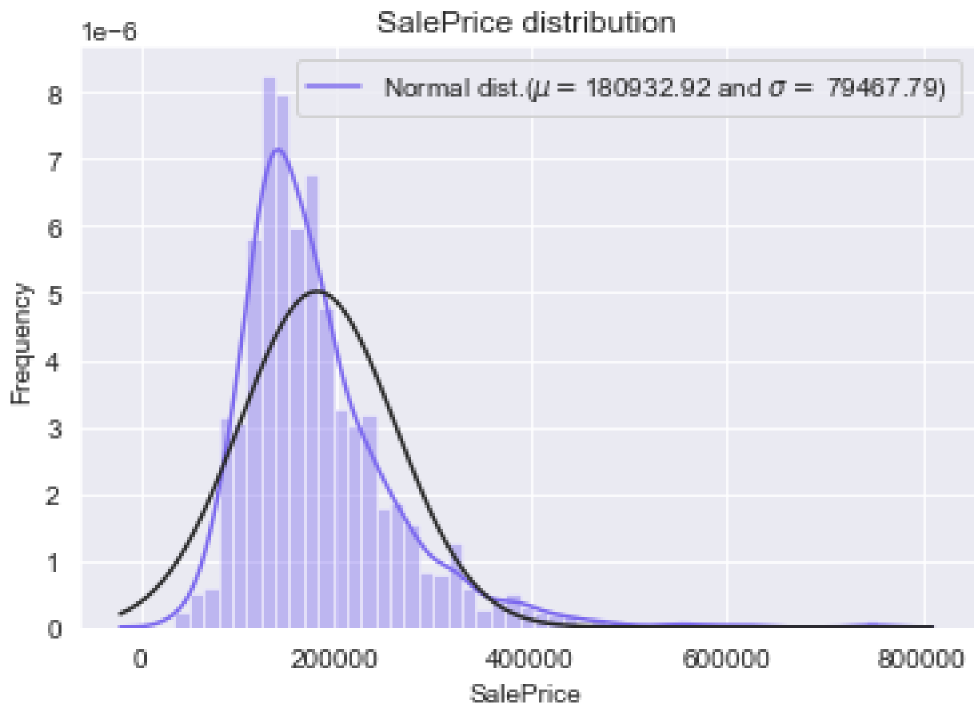
[7]:
```
train=train.drop(train[(train['GrLivArea']>4000)&(train['SalePrice']<300000)].
↪index)
```

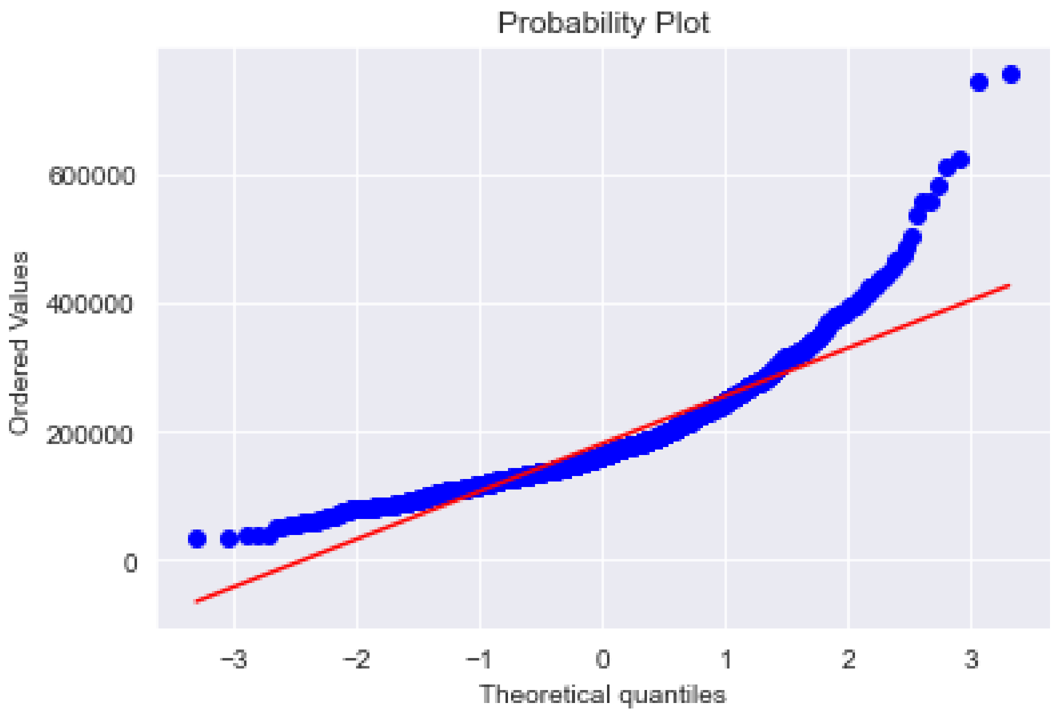
[8]:
```
fig,ax=plt.subplots()
ax.scatter(x=train['GrLivArea'],y=train['SalePrice'],color="#D8BFD8")
plt.ylabel('SalePrice',fontsize=13)
plt.xlabel('GrLivArea',fontsize=13)
plt.show()
```



[9]:
```
ax=sns.distplot(train['SalePrice'],fit=norm,color="#7B68EE")
hist_fig = ax.get_figure()
hist_fig.savefig(r'D:\SYSU\21-22third_grade\ \HW\HW6\price_distribution.png',␣
 ↪dpi=300)

(mu,sigma)=norm.fit(train['SalePrice'])
print('\n mu= {:.2f} and sigma={:.2f}\n'.format(mu,sigma))

plt.legend(['Normal dist.($\mu=${:.2f} and $\sigma=$ {:.2f})'.
 ↪format(mu,sigma)],loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig=plt.figure()
res=stats.probplot(train['SalePrice'],plot=plt)
plt.show()
fig.savefig(r'D:\SYSU\21-22third_grade\ \HW\HW6\price_area_scatter_plot.png',␣
 ↪dpi=300)
```

```
mu= 180932.92 and sigma=79467.79
```


SalePrice distribution
1e−6
Normal dist.(μ = 180932.92 and σ = 79467.79)
Frequency
SalePrice
0
1
2
3
4
5
6
7
8
0
200000
400000
600000
800000

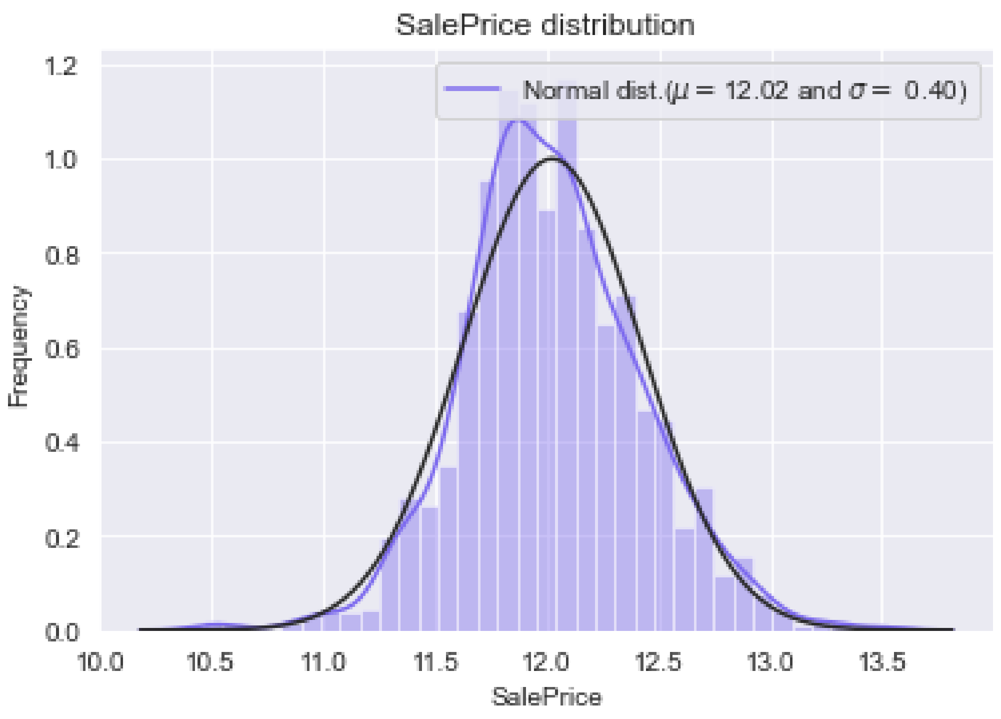

Probability Plot
Ordered Values
Theoretical quantiles
0
200000
400000
600000
−3
−2
−1
0
1
2
3

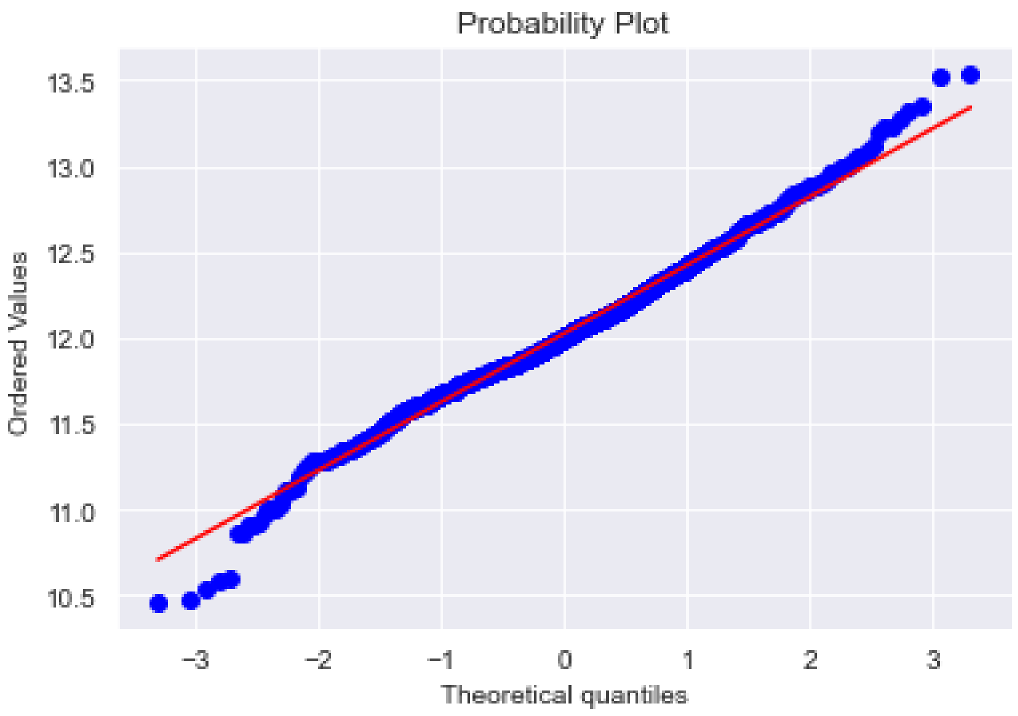
[10]:
```
fig.savefig(r'D:\SYSU\21-22third_grade\ \HW\HW6\probability_plot.png',␣
 ↪dpi=300)
```

[11]:
```
train['SalePrice']=np.log1p(train['SalePrice'])
```

[12]:
```
ax=sns.distplot(train['SalePrice'],fit=norm,color="#7B68EE")
hist_fig = ax.get_figure()
hist_fig.savefig(r'D:\SYSU\21-22third_grade\ \HW\HW6\price_distribution2.
 ↪png', dpi=300)
(mu,sigma)=norm.fit(train['SalePrice'])
print('\n mu= {:.2f} and sigma={:.2f}\n'.format(mu,sigma))

plt.legend(['Normal dist.($\mu=${:.2f} and $\sigma=$ {:.2f})'.
 ↪format(mu,sigma)],loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig=plt.figure()
res=stats.probplot(train['SalePrice'],plot=plt)
plt.show()

fig.savefig(r'D:\SYSU\21-22third_grade\ \HW\HW6\fig2.png', dpi=300)
```

```
mu= 12.02 and sigma=0.40
```





```
[13]: ntrain=train.shape[0]
      ntest=test.shape[0]
      y_train=train.SalePrice.values
      all_data=pd.concat((train,test)).reset_index(drop=True)
      all_data.drop(['SalePrice'],axis=1,inplace=True)
      #  test train data        all_data
```

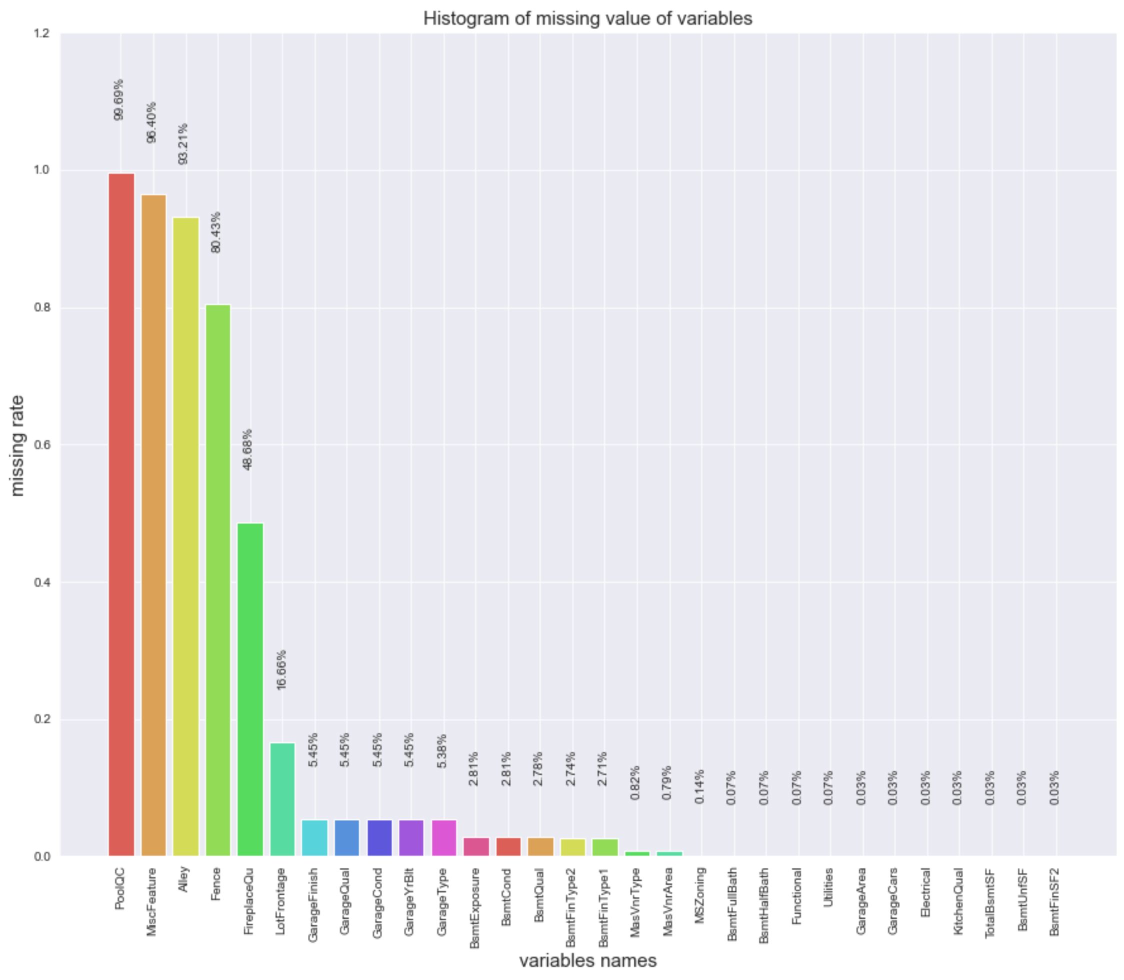
```
[14]: missing=all_data.isnull().sum().reset_index().rename(columns={0:'missNum'})
```

```
[15]: missing['missRate']=missing['missNum']/all_data.shape[0]
      miss_analy=missing[missing.missRate>0].sort_values(by='missRate',␣
       ↪ascending=False)[:30].reset_index(drop=True)
      miss_analy.head(20)
```

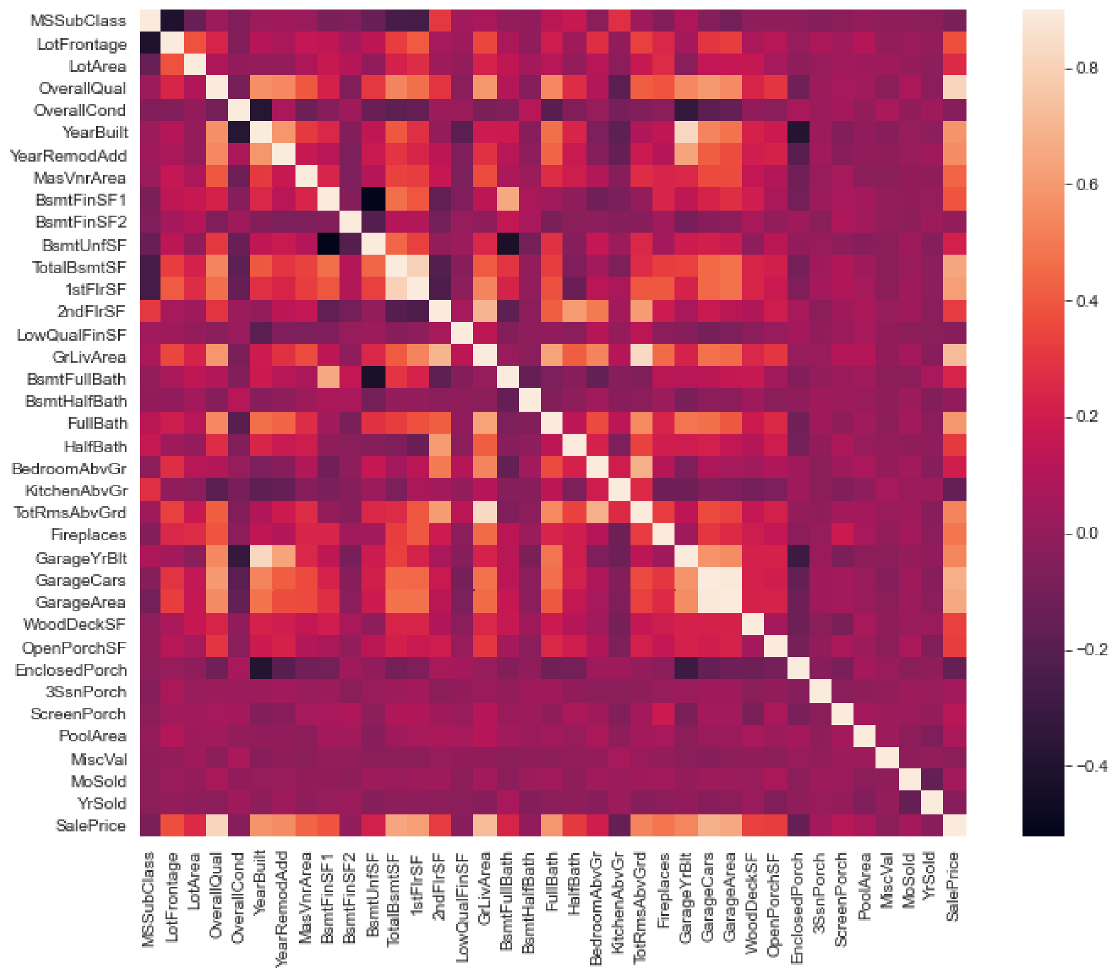
```
[15]:           index  missNum  missRate
      0        PoolQC     2908     0.997
      1   MiscFeature     2812     0.964
      2         Alley     2719     0.932
      3         Fence     2346     0.804
      4   FireplaceQu     1420     0.487
      5   LotFrontage      486     0.167
```

```
6   GarageFinish      159     0.055
7     GarageQual      159     0.055
8     GarageCond      159     0.055
9    GarageYrBlt      159     0.055
10    GarageType      157     0.054
11  BsmtExposure       82     0.028
12      BsmtCond       82     0.028
13      BsmtQual       81     0.028
14  BsmtFinType2       80     0.027
15  BsmtFinType1       79     0.027
16    MasVnrType       24     0.008
17    MasVnrArea       23     0.008
18      MSZoning        4     0.001
19  BsmtFullBath        2     0.001
```

[16]:
```
fig=plt.figure(figsize=(15,12))
plt.bar(np.arange(miss_analy.shape[0]),list(miss_analy.missRate.
 ↪values),align='center',color=sns.color_palette('hls',12))
font={'size':15,}
plt.title('Histogram of missing value of variables',fontsize=15)
plt.xlabel('variables names',font)
plt.ylabel('missing rate',font)
#  x    90
plt.xticks(np.arange(miss_analy.shape[0]),list(miss_analy['index']))
plt.xticks(rotation=90)
#
for x,y in enumerate(list(miss_analy.missRate.values)):
     plt.text(x,y+0.08,'{:.2%}'.format(y),ha='center',rotation='90')
plt.ylim([0,1.2])
fig.savefig(r'D:\SYSU\21-22third_grade\ \ \HW\HW6\missing_ratio.png', dpi=300)
```



```
[17]: #Correlation map to see how features are correlated with SalePrice
      corrmat = train.corr()
      plt.subplots(figsize=(12,9))
      sns.heatmap(corrmat, vmax=0.9, square=True)

[17]: <AxesSubplot:>
```



```
[18]: all_data['PoolQC']=all_data['PoolQC'].fillna('None')
```

```
[19]: all_data['MiscFeature']=all_data['MiscFeature'].fillna('None')
```

```
[20]: all_data["Alley"] = all_data["Alley"].fillna("None")
```

```
[21]: all_data["Fence"] = all_data["Fence"].fillna("None")
```

```
[22]: all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
```

```
[23]: all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFrontage"].
       ↪transform(lambda x: x.fillna(x.median()))
      #transform--          df     na    median
```

```
[24]: for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
          all_data[col] = all_data[col].fillna('None')
```

[25]:
```
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)
```

[26]:
```
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF',␣
 ↪'BsmtFullBath', 'BsmtHalfBath'):
    all_data[col] = all_data[col].fillna(0)
```

[27]:
```
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',␣
 ↪'BsmtFinType2'):
    all_data[col] = all_data[col].fillna('None')
```

[28]:
```
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)
```

[29]:
```
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].
 ↪mode()[0])
#RL   mode()
```

[30]:
```
all_data['Utilities'].value_counts()
```

[30]:
```
AllPub    2914
NoSeWa       1
Name: Utilities, dtype: int64
```

[31]:
```
all_data['Utilities'][ntrain:].value_counts()
#      allpub,    Nosewa train
```

[31]:
```
AllPub    1457
Name: Utilities, dtype: int64
```

[32]:
```
all_data = all_data.drop(['Utilities'], axis=1)
#   drop
```

[33]:
```
all_data["Functional"] = all_data["Functional"].fillna("Typ")
#data description  na typical
```

[34]:
```
all_data['Electrical'].value_counts()
```

[34]:
```
SBrkr    2669
FuseA     188
FuseF      50
FuseP       8
Mix         1
Name: Electrical, dtype: int64
```

[35]:
```
all_data['Electrical']=all_data['Electrical'].fillna(all_data['Electrical'].
 ↪mode()[0])
#
```

```
#     miss value
```

[36]:
```
all_data['KitchenQual'] = all_data['KitchenQual'].
 ↪fillna(all_data['KitchenQual'].mode()[0])
```

[37]:
```
all_data['Exterior1st'] = all_data['Exterior1st'].
 ↪fillna(all_data['Exterior1st'].mode()[0])
all_data['Exterior2nd'] = all_data['Exterior2nd'].
 ↪fillna(all_data['Exterior2nd'].mode()[0])
```

[38]:
```
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleType'].
 ↪mode()[0])
```

[39]:
```
all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")
```

[40]:
```
#Check remaining missing values if any
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).
 ↪sort_values(ascending=False)
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
#   missingvalues
```

[40]:
```
Empty DataFrame
Columns: [Missing Ratio]
Index: []
```

[41]:
```
#      categorial variable
all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)
# building class
```

[42]:
```
all_data['OverallCond'] = all_data['OverallCond'].astype(str)
```

[43]:
```
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

[44]:
```
#Label Encoding some categorical variables that may contain information in␣
 ↪their ordering set
from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual',␣
 ↪'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish',␣
 ↪'LandSlope',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir',␣
 ↪'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold')
```

```
# process columns, apply LabelEncoder to categorical features
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))
#
```

[45]:
```
print('Shape all_data: {}'.format(all_data.shape))
```

```
Shape all_data: (2917, 78)
```

[46]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())).
↪sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```

Skew in numerical features:

```

[46]:
```
                 Skew
MiscVal        21.940
PoolArea       17.689
LotArea        13.109
LowQualFinSF   12.085
3SsnPorch      11.372
LandSlope       4.973
KitchenAbvGr    4.301
BsmtFinSF2      4.145
EnclosedPorch   4.002
ScreenPorch     3.945
```

[47]:
```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform".
↪format(skewness.shape[0]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    #all_data[feat] += 1
    all_data[feat] = boxcox1p(all_data[feat], lam)

#all_data[skewed_features] = np.log1p(all_data[skewed_features])
```

```
There are 58 skewed numerical features to Box Cox transform
```

[48]:
```
all_data = pd.get_dummies(all_data)
print(all_data.shape)
```

```
(2917, 219)
```

[49]:
```
train=all_data[:ntrain]
test=all_data[ntrain:]
```

[50]:
```
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge, LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error,confusion_matrix
from sklearn import metrics
import xgboost as xgb
from xgboost import XGBClassifier
from xgboost import plot_importance
```

[51]:
```
n_folds = 5

def msle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(train.
 ↪values)
    mse= -cross_val_score(model, train.values, y_train,␣
 ↪scoring="neg_mean_squared_error", cv = kf,n_jobs=-1)
    return(mse)
```

[52]:
```
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
```

[53]:
```
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9,␣
 ↪random_state=3))
```

[54]:
```
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
```

[55]:
```
GBoost = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5)
```

[56]:
```
import lightgbm as lgb
```

```
[58]: model_lgb = lgb.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)
```

```
[59]: #eg:        --
      def par(model):
          learning_rate=[0.0001,0.001,0.01,0.05,0.1,0.2]
          n_estimators=[0,1000,2000]
          param_grid=dict(learning_rate=learning_rate,n_estimators=n_estimators)
          kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(train.
       ↪values)
          ␣
       ↪grid_search=GridSearchCV(model,param_grid,scoring='neg_mean_squared_error',cv=kf,n_jobs=-1)
          grid_result=grid_search.fit(train.values,y_train)
          means = grid_result.cv_results_['mean_test_score']
          stds = grid_result.cv_results_['std_test_score']
          params = grid_result.cv_results_['params']
          for mean, stdev, param in zip(means, stds, params):
              print("%f (%f) with: %r" % (mean, stdev, param))
          return(grid_result.best_params_)
```

```
[60]: model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                                    learning_rate=0.01, max_depth=3,
                                    min_child_weight=1.7817, n_estimators=2200,
                                    reg_alpha=0.4640, reg_lambda=0.8571,
                                    subsample=0.5213,
                                    random_state =7, nthread = -1)
```

```
[61]: # lgb          0.05
      par(model_lgb)
```

```
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
nan (nan) with: {'learning_rate': 0.0001, 'n_estimators': 0}
-0.143123 (0.011982) with: {'learning_rate': 0.0001, 'n_estimators': 1000}
-0.128637 (0.011196) with: {'learning_rate': 0.0001, 'n_estimators': 2000}
nan (nan) with: {'learning_rate': 0.001, 'n_estimators': 0}
```

```
-0.062572 (0.006862) with: {'learning_rate': 0.001, 'n_estimators': 1000}
-0.034348 (0.004147) with: {'learning_rate': 0.001, 'n_estimators': 2000}
nan (nan) with: {'learning_rate': 0.01, 'n_estimators': 0}
-0.014173 (0.001768) with: {'learning_rate': 0.01, 'n_estimators': 1000}
-0.013269 (0.001653) with: {'learning_rate': 0.01, 'n_estimators': 2000}
nan (nan) with: {'learning_rate': 0.05, 'n_estimators': 0}
-0.013450 (0.002082) with: {'learning_rate': 0.05, 'n_estimators': 1000}
-0.013718 (0.002141) with: {'learning_rate': 0.05, 'n_estimators': 2000}
nan (nan) with: {'learning_rate': 0.1, 'n_estimators': 0}
-0.014649 (0.001609) with: {'learning_rate': 0.1, 'n_estimators': 1000}
-0.015098 (0.001805) with: {'learning_rate': 0.1, 'n_estimators': 2000}
nan (nan) with: {'learning_rate': 0.2, 'n_estimators': 0}
-0.016284 (0.001587) with: {'learning_rate': 0.2, 'n_estimators': 1000}
-0.016533 (0.001570) with: {'learning_rate': 0.2, 'n_estimators': 2000}
```

```
[61]: {'learning_rate': 0.01, 'n_estimators': 2000}
```

```
[62]: #
par(model_xgb)
```

```
-132.962162 (0.416191) with: {'learning_rate': 0.0001, 'n_estimators': 0}
-108.919883 (0.386046) with: {'learning_rate': 0.0001, 'n_estimators': 1000}
-89.230897 (0.357205) with: {'learning_rate': 0.0001, 'n_estimators': 2000}
-132.962162 (0.416191) with: {'learning_rate': 0.001, 'n_estimators': 0}
-18.150418 (0.164486) with: {'learning_rate': 0.001, 'n_estimators': 1000}
-2.522457 (0.053125) with: {'learning_rate': 0.001, 'n_estimators': 2000}
-132.962162 (0.416191) with: {'learning_rate': 0.01, 'n_estimators': 0}
-0.014523 (0.001439) with: {'learning_rate': 0.01, 'n_estimators': 1000}
-0.013543 (0.001489) with: {'learning_rate': 0.01, 'n_estimators': 2000}
-132.962162 (0.416191) with: {'learning_rate': 0.05, 'n_estimators': 0}
-0.013861 (0.001487) with: {'learning_rate': 0.05, 'n_estimators': 1000}
-0.013822 (0.001481) with: {'learning_rate': 0.05, 'n_estimators': 2000}
-132.962162 (0.416191) with: {'learning_rate': 0.1, 'n_estimators': 0}
-0.013986 (0.001867) with: {'learning_rate': 0.1, 'n_estimators': 1000}
-0.013971 (0.001824) with: {'learning_rate': 0.1, 'n_estimators': 2000}
-132.962162 (0.416191) with: {'learning_rate': 0.2, 'n_estimators': 0}
-0.015278 (0.001467) with: {'learning_rate': 0.2, 'n_estimators': 1000}
-0.015265 (0.001372) with: {'learning_rate': 0.2, 'n_estimators': 2000}
```

```
[62]: {'learning_rate': 0.01, 'n_estimators': 2000}
```

```
[63]: # R
GBoost.fit(train.values,y_train)
y_pred=GBoost.predict(train.values)
print('R square: {}'.format(metrics.r2_score(y_train,y_pred)))
```

```
R square: 0.9845126714151072
```

[64]:
```
# R
model_xgb.fit(train.values,y_train)
y_pred=model_xgb.predict(train.values)
print('R square: {}'.format(metrics.r2_score(y_train,y_pred)))
```

```
R square: 0.951368147166869
```

[65]:
```
# R
model_lgb.fit(train.values,y_train)
y_pred=model_lgb.predict(train.values)
print('R square: {}'.format(metrics.r2_score(y_train,y_pred)))
```

```
R square: 0.9670290028044588
```

[66]:
```
# mse
score=msle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```

Lasso score: 0.0127 (0.0017)

```

[67]:
```
# mse
score = msle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Kernel Ridge score: 0.0133 (0.0017)

```

[68]:
```
score = msle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.
 ↪std()))
```

```
Gradient Boosting score: 0.0139 (0.0019)

```

[69]:
```
score = msle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Xgboost score: 0.0135 (0.0015)

```

[70]:
```
score = msle_cv(model_lgb)
print("LGBM score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
LGBM score: 0.0134 (0.0020)

```

[71]:
```
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models
    # we define clones of the original models to fit the data in
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X, y)

        return self
    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)
```

[72]:
```
averaged_models = AveragingModels(models = (ENet, GBoost, KRR, lasso))

score = msle_cv(averaged_models)
print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(),␣
 ↪score.std()))
```

```
 Averaged base models score: 0.0119 (0.0017)
```

[73]:
```
# R
averaged_models.fit(train.values,y_train)
y_pred=averaged_models.predict(train.values)
print('R square: {}'.format(metrics.r2_score(y_train,y_pred)))
```

```
R square: 0.9598757647466997
```

[74]:
```
y=pd.DataFrame(y_train)
```

[75]:
```
out=pd.concat([train,y],axis=1)
```

[76]:
```
out.to_excel(excel_writer = r"C:\Users\10048\Desktop\  .xlsx")
```

[77]:
```
outtest=test
```

[78]:
```
outtest.to_excel(excel_writer = r"C:\Users\10048\Desktop\test.xlsx")
```

[79]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())).
 ↪sort_values(ascending=False)
```

```
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

```
[79]:                          Skew
      Condition2_RRAn     53.981
      RoofMatl_Membran    53.981
      Exterior2nd_Other   53.981
      Condition2_RRAe     53.981
      MiscFeature_TenC    53.981
      Exterior1st_ImStucc 53.981
      Electrical_Mix      53.981
      RoofMatl_Metal      53.981
      Heating_Floor       53.981
      RoofMatl_Roll       53.981
```

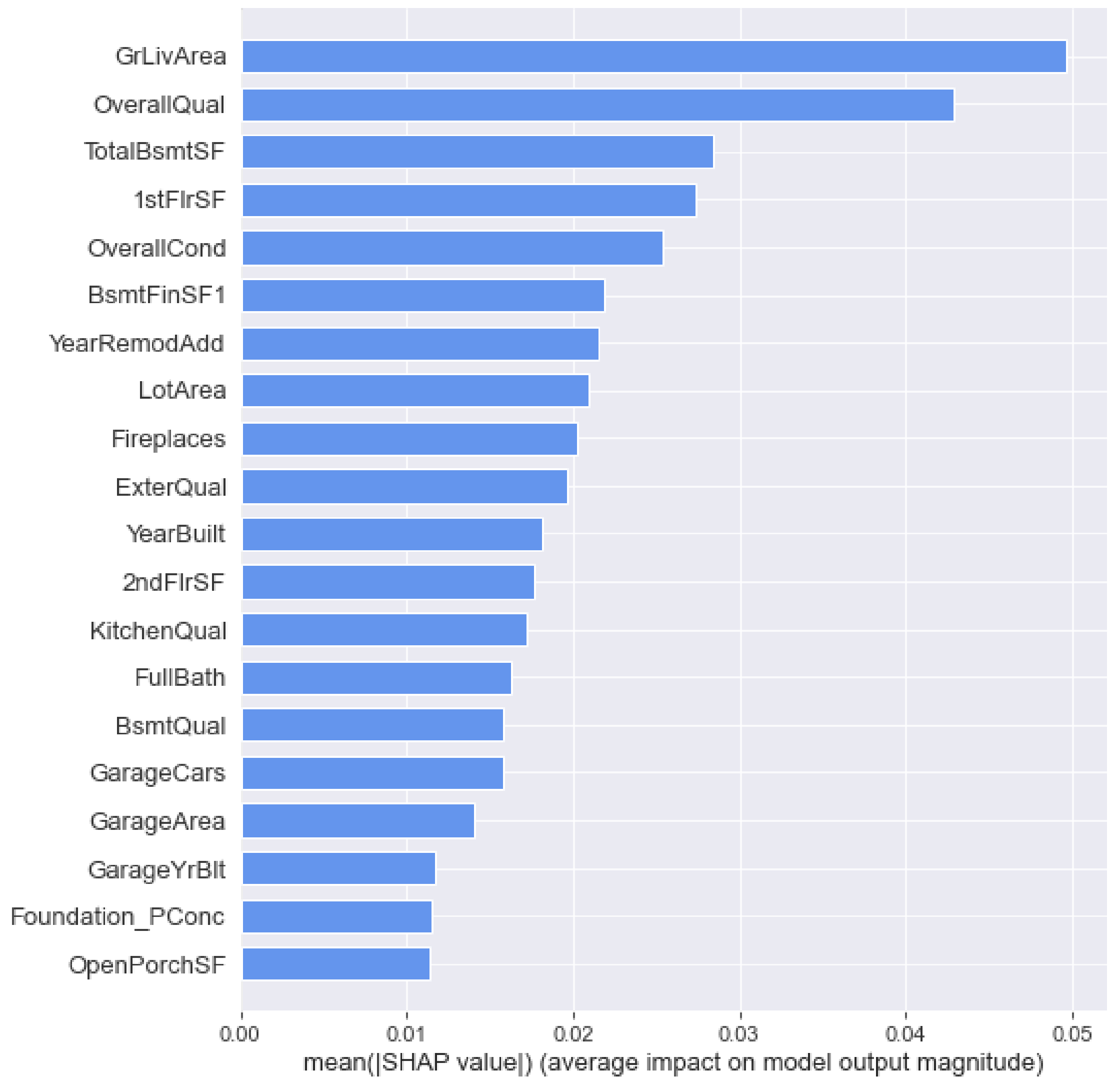
```
[80]: import shap
      shap.initjs()
```

```
<IPython.core.display.HTML object>
```

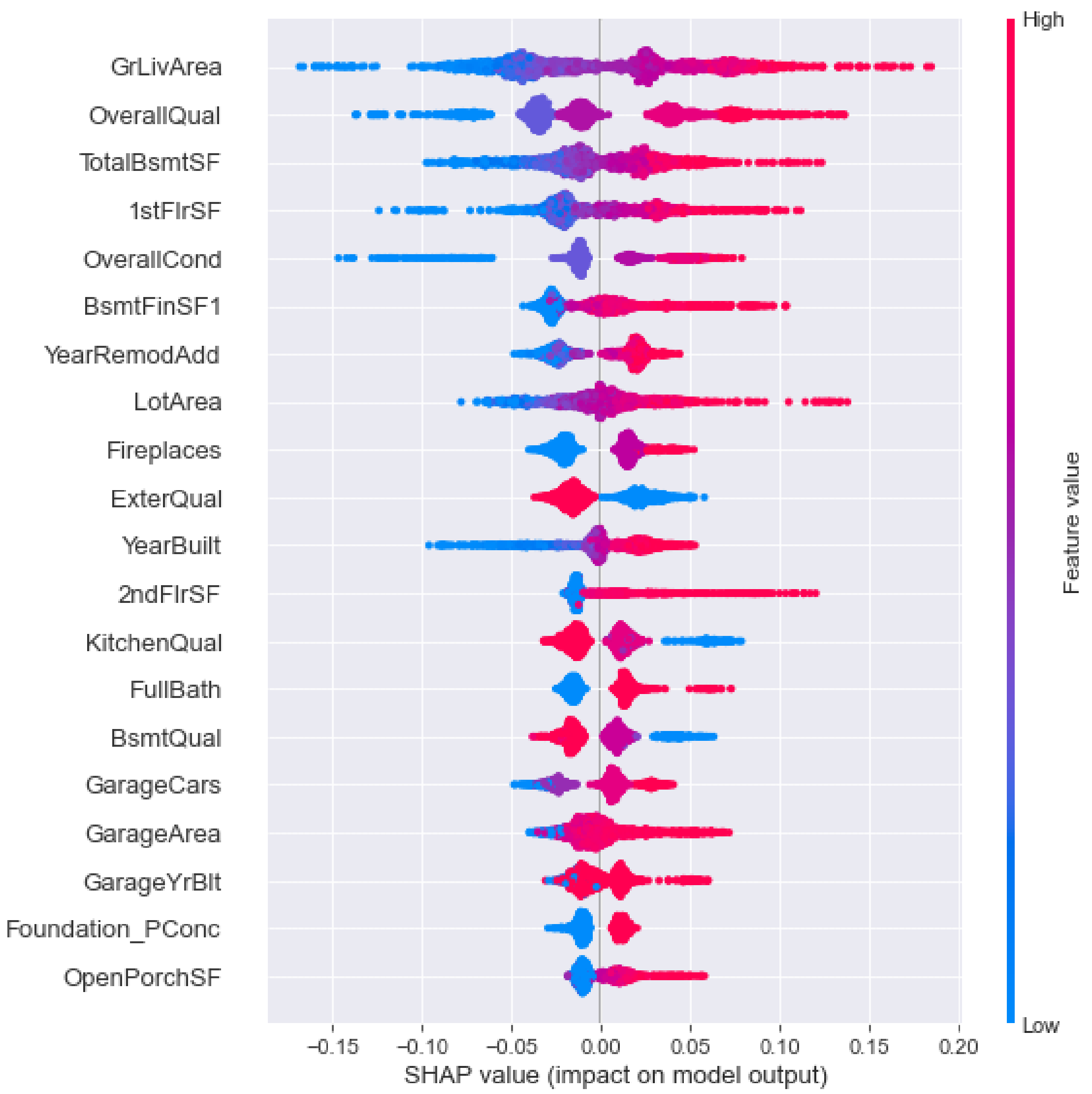
```
[81]: model=GBoost.fit(train.values,y_train)
      explainer = shap.TreeExplainer(model)
```

```
[82]: shap_values = explainer.shap_values(train.values)  #    X  SHAP
```

```
[83]: f=shap.summary_plot(shap_values, train, plot_type="bar",color= '#6495ED')
```



[84]:
```
shap.summary_plot(shap_values, train)
```

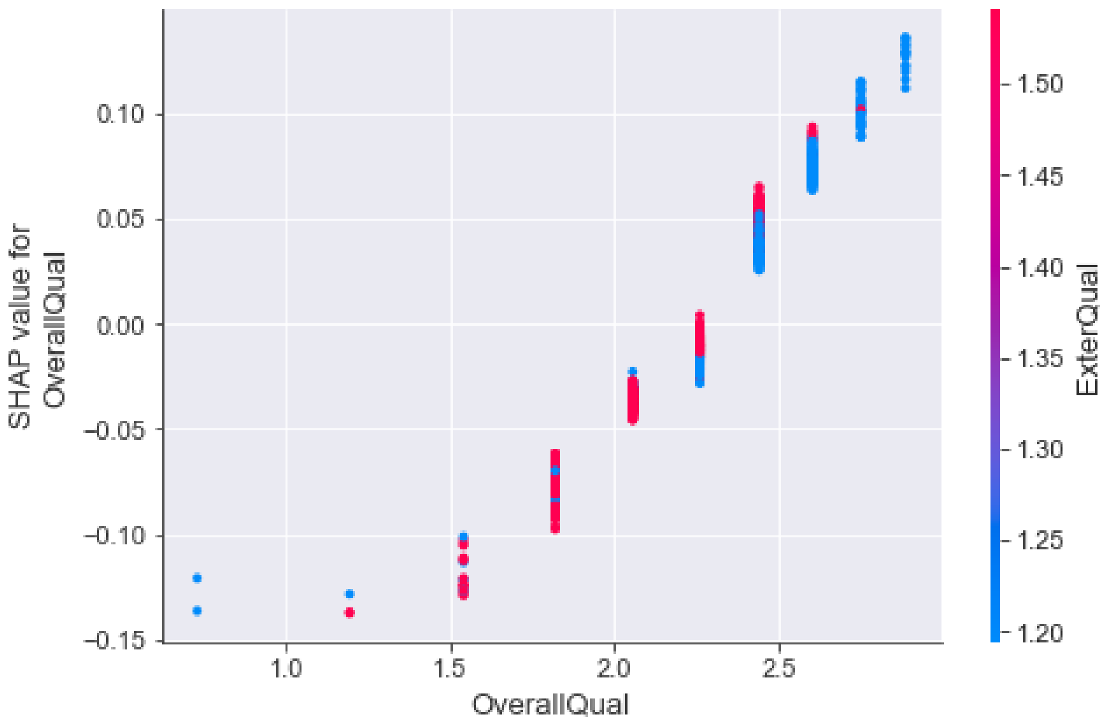


[85]:
```
shap.force_plot(explainer.expected_value, shap_values[0,:], train.iloc[0,:])
```

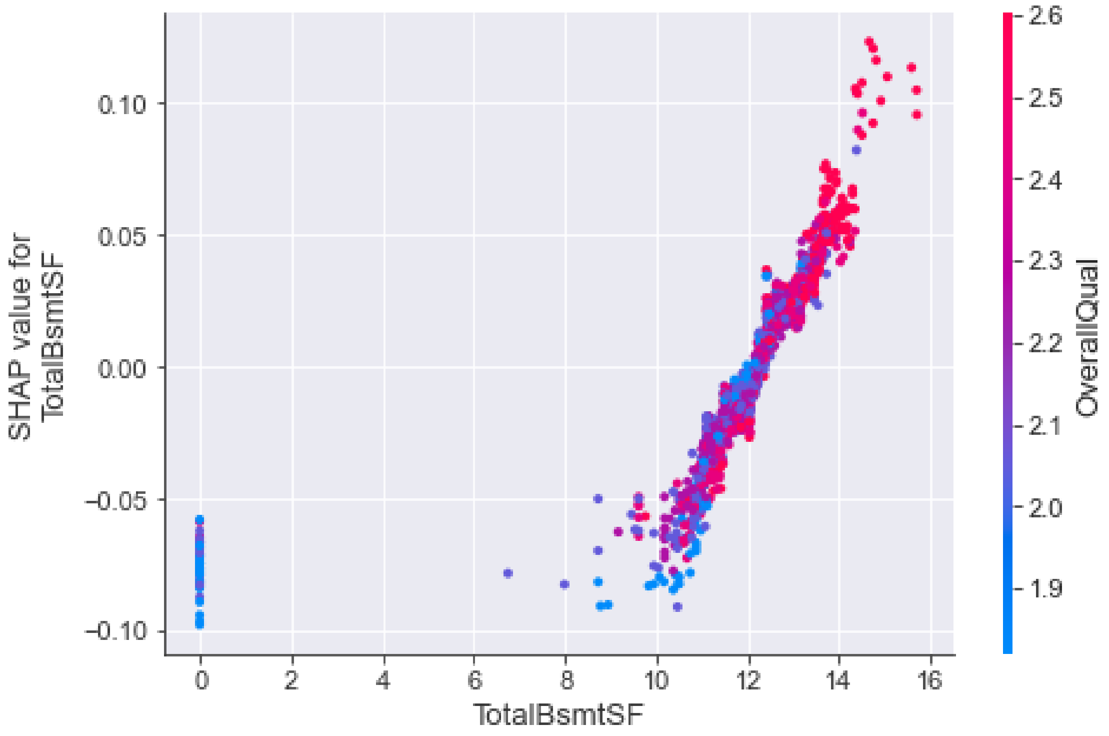
[85]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x1f7c40ccd00>
```

shap.dependence_plot(“GrLivArea”, shap_values, train)

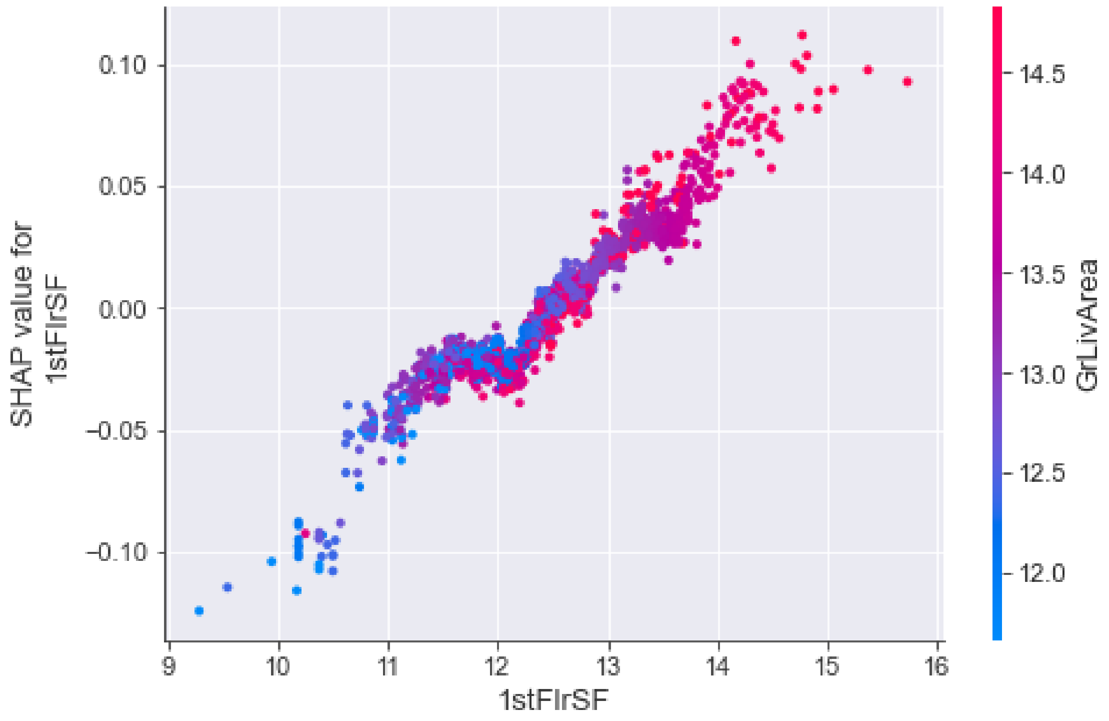
[86]:
```
shap.dependence_plot("OverallQual", shap_values, train)
```



[87]:
```
shap.dependence_plot("TotalBsmtSF", shap_values, train)
```



[88]:
```
shap.dependence_plot("1stFlrSF", shap_values, train)
```


SHAP value for 1stFlrSF
1stFlrSF
GrLivArea