房价预测项目报告

1.项目的选题和意义

在本学期的机器学习大作业上，我们小组选择的是 kaggle 的一个关于 Ames 的房价预测问题。数据集中用 79 个解释变量描述影响住宅的各个方面，我们通过对这些数据的研究来找到影响房价的主要因素和次要因素，然后对最终的房价进行预测，得出结论。

作为一个机器学习的入门级项目，房价预测问题在提高我们机器学习方面的能力有较大的帮助，扩宽了我们机器学习知识的视野，激发了我们对机器学习方面的兴趣。另一方面，房价预测问题有较高的现实意义，随着房地产的发展和人们对家的追求，房价的升降牵动着每一个买房者的心。同样，房价的预测对房地产厂商也有较大的意义。虽然这个题目所给的条件没有现实生活中那么复杂，但是通过不断的修正和改善，真正将其运用到生活中，一定会给人们的决策作出巨大的改变。

总而言之，房价的预测问题对个人的能力提升和为人民造福方面有重大意义。

2.实验设置

本项目以 jupyter notebook 为平台，编译语言为 python3，引用了 numpy、pandas、torch、seaborn、missingno、xgboost、optuna、LazyRegressor、sklearn 等函数库。

3..数据分析

目前在网上已经有很多研究成果，从训练数据和测试数据来说，大家都分析了数据规模、前几条数据、数据类型、缺省值及可视化。

同样，我们参考了一些网上的数据分析的方法，借鉴了很多大佬的思路。对训练集和测试集的数据维度、类型、缺省值及其相关性、分布等进行了分析，对比了训练数据和测试数

据的类型、缺失数据及各种数据的特征及相关性。挑选了几个重要的及生活中常用的评价指标作为重点分析对象。

数据维度和数据信息

```
df_train.shape
```

```
(1460, 81)
```

```
df_test.shape
```

```
(1459, 80)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

前 5 条数据信息：

```
df_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fe
0	1	60	RL	65.00	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	
1	2	20	RL	80.00	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	
2	3	60	RL	68.00	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	
3	4	70	RL	60.00	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	
4	5	60	RL	84.00	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	

5 rows × 81 columns

```
df_test.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolA
0	1461	20	RH	80.00	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	
1	1462	20	RL	81.00	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	
2	1463	60	RL	74.00	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	
3	1464	60	RL	78.00	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	
4	1465	120	RL	43.00	5005	Pave	NaN	IR1	HLS	AllPub	...	144	

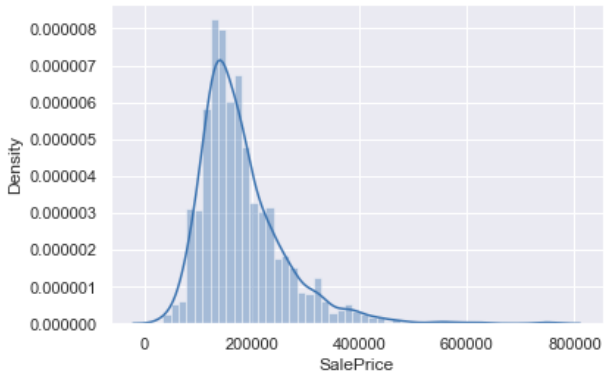
5 rows × 80 columns

研究对象是房价，因此对房价进行分析：房价平均价格 18 万、最大值最小值相差较大，

数据是偏左的正态分布，查看其偏度和峰度且偏度值较大，处理分析时需注意。

```
df_train['SalePrice'].describe()
```

```
count      1460.00
mean     180921.20
std       79442.50
min       34900.00
25%      129975.00
50%      163000.00
75%      214000.00
max      755000.00
Name: SalePrice, dtype: float64
```




```
print("峰度(kurtosis)为: {}".format(df_train['SalePrice'].kurt()))
```

峰度(kurtosis)为: 6.536281860064529

```
print("偏度(skewness)为: {}".format(df_train['SalePrice'].skew()))
```

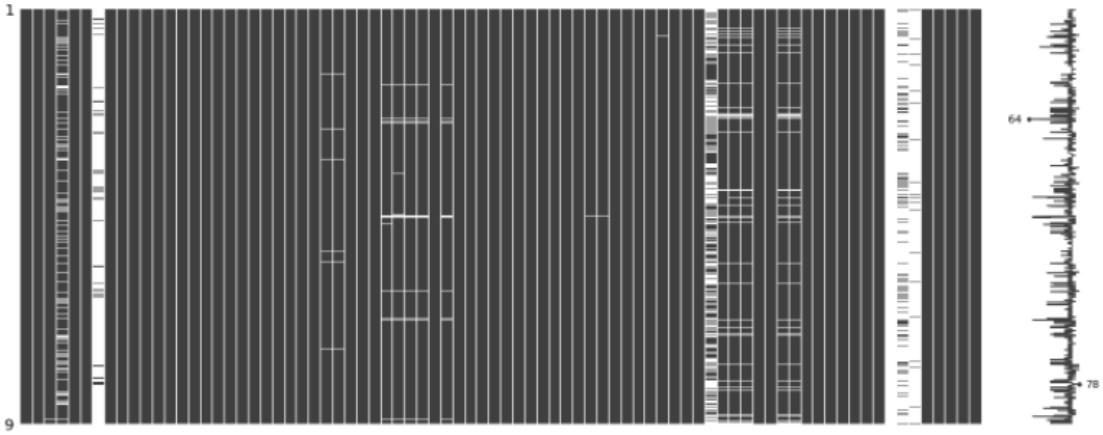
偏度(skewness)为: 1.8828757597682129

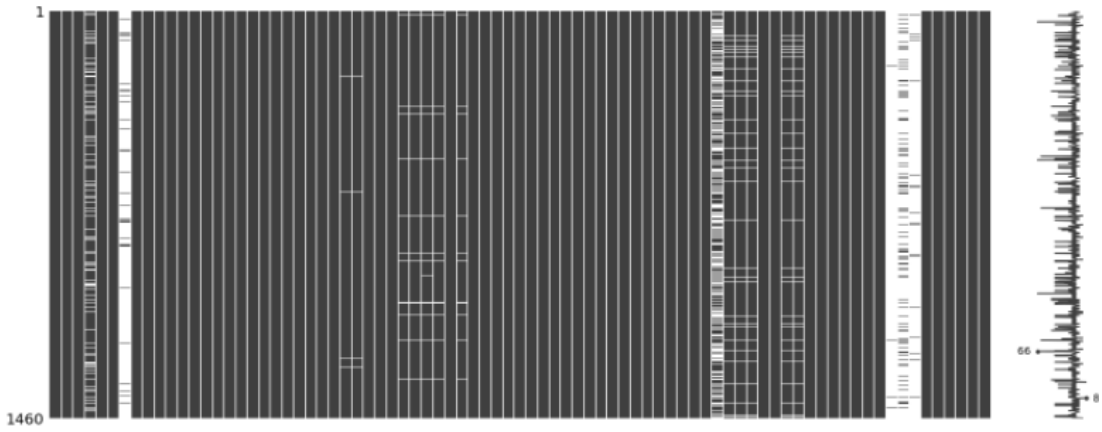
查看训练数据缺省值得前 20 个

```
#训练数据中的空值排序前20个
train_data.isnull().sum().sort_values(ascending=False).head(20)
```

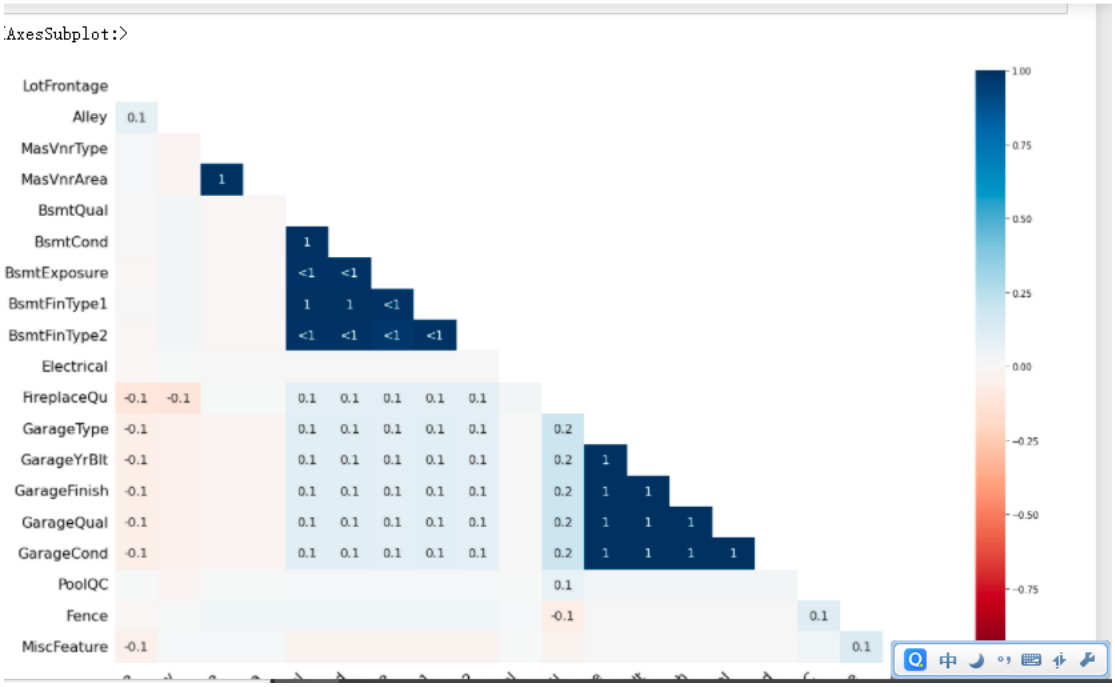
```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageCond        81
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
BsmtExposure      38
BsmtFinType2      38
BsmtFinType1      37
BsmtCond          37
BsmtQual          37
MasVnrArea         8
MasVnrType         8
Electrical         1
Utilities          0
dtype: int64
```

缺省值可视化：





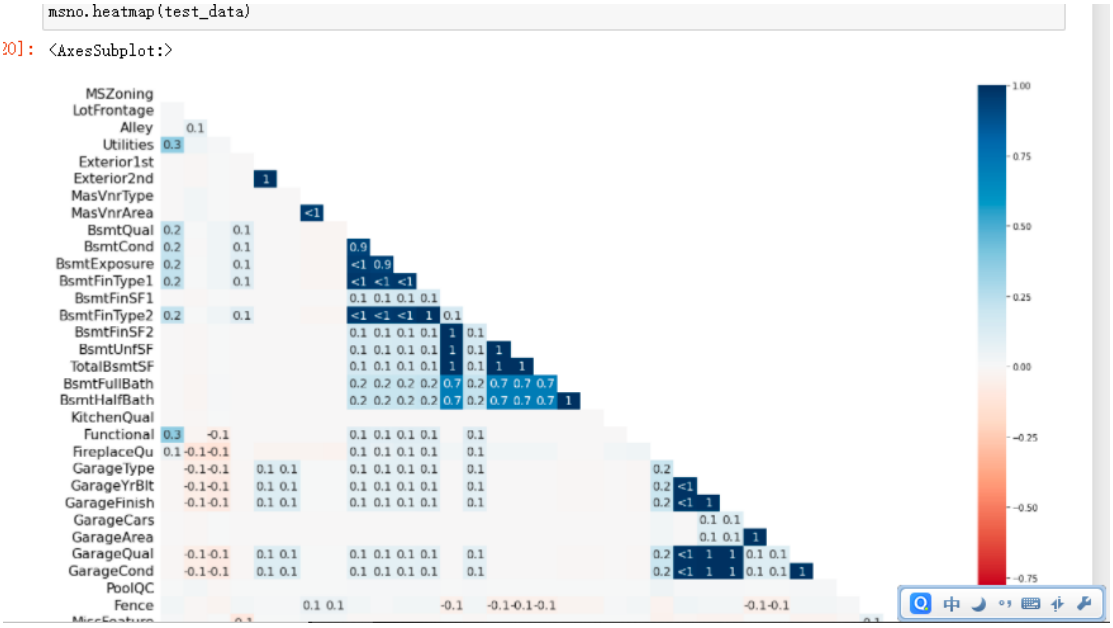
使用 misingno 查看缺失值之间的相关性



缺省值对比分析，可看出测试数据缺失更加严重，故后续处理需要去除训练集的价格，然后进行对比。

```
msno.heatmap(test_data)
```

20]: <AxesSubplot:>



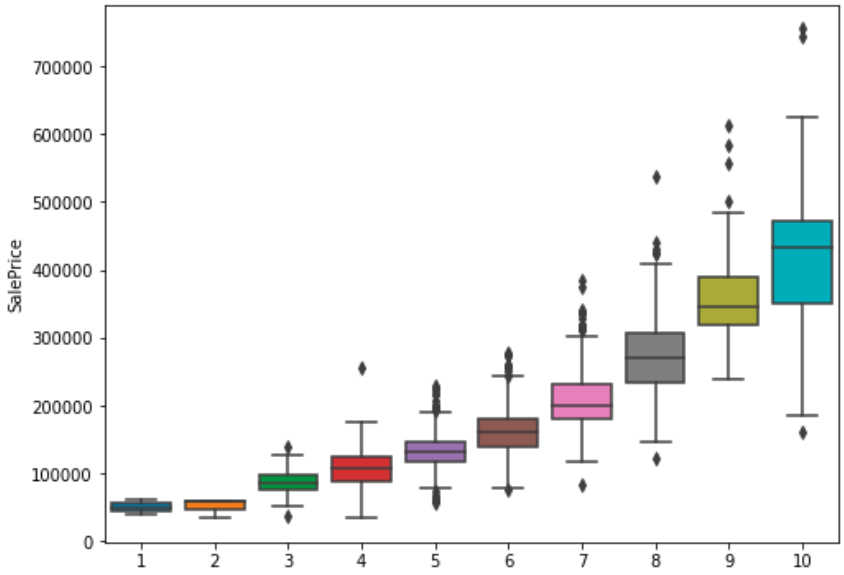

```
#缺失数据对比
null_train = train_data.isnull().sum()
null_test = test_data.isnull().sum()
null_train = null_train.drop('SalePrice')
null_comp_df = null_train.compare(null_test).sort_values(['self'],ascending=[False])
null_comp_df
```

	self	other
PoolQC	1453.0	1456.0
MiscFeature	1406.0	1408.0
Alley	1369.0	1352.0
Fence	1179.0	1169.0
FireplaceQu	690.0	730.0
LotFrontage	259.0	227.0
GarageType	81.0	76.0
GarageCond	81.0	78.0
GarageYrBlt	81.0	78.0
GarageFinish	81.0	78.0
GarageQual	81.0	78.0
BsmtFinType2	38.0	42.0
BsmtExposure	38.0	44.0

评分与房价的关系

```
overallQual_SalePrice = pd.concat([train_data['SalePrice'],train_data['OverallQual']],axis=1)
plt.figure(figsize=(8,6))
sns.boxplot(x='OverallQual',y='SalePrice',data=overallQual_SalePrice)
```

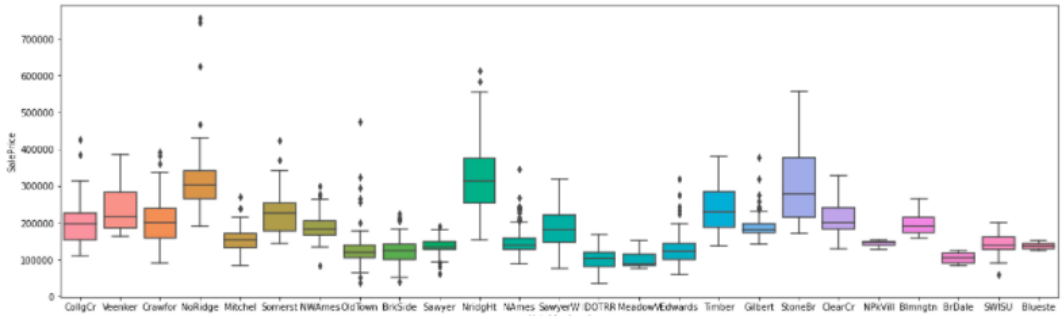
```
<AxesSubplot:xlabel='OverallQual', ylabel='SalePrice'>
```



箱线图查看离散非数值型数据的分布

```
Neighborhood_SalePrice = pd.concat([train_data['SalePrice'],train_data['Neighborhood']],axis=1)
plt.figure(figsize=(20,6))
sns.boxplot(x='Neighborhood',y='SalePrice',data=Neighborhood_SalePrice)
```

```
<AxesSubplot:xlabel='Neighborhood', ylabel='SalePrice'>
```

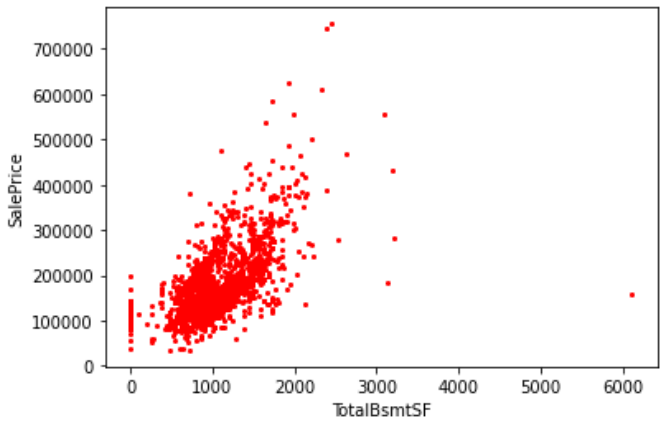


绘制和价格相关的特征的散点图

```
TotalBsmtSF_SalePrice = pd.concat([train_data['SalePrice'],train_data['TotalBsmtSF']],axis=1)
plt.figure(figsize=(8,6))
TotalBsmtSF_SalePrice.plot.scatter(x='TotalBsmtSF',y='SalePrice',s=4,c='red')
```

```
<AxesSubplot:xlabel='TotalBsmtSF', ylabel='SalePrice'>

<Figure size 576x432 with 0 Axes>
```



各种特征的数量

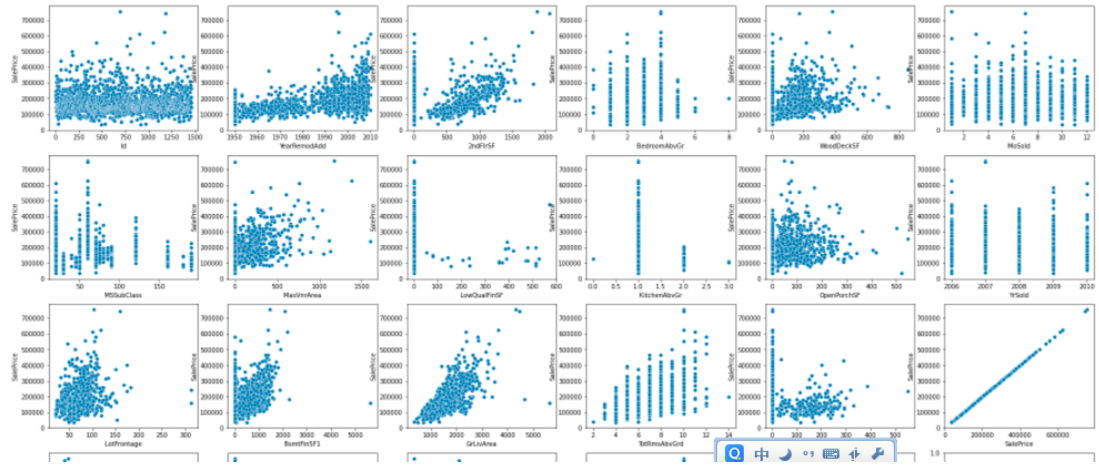
```
numerical_features = [col for col in train_data.columns if train_data[col].dtypes != 'O']
discrete_features = [col for col in numerical_features if len(train_data[col].unique()) < 25 and col not in ['Id
continuous_features = [feature for feature in numerical_features if feature not in discrete_features+['Id']]
categorical_features = [col for col in train_data.columns if train_data[col].dtype == 'O']

print("Total Number of Numerical Columns : ",len(numerical_features))
print("Number of discrete features : ",len(discrete_features))
print("No of continuous features are : ", len(continuous_features))
print("Number of non-numeric features : ",len(categorical_features))
```

```
Total Number of Numerical Columns :  38
Number of discrete features :  18
No of continuous features are :  19
Number of non-numeric features :  43
```

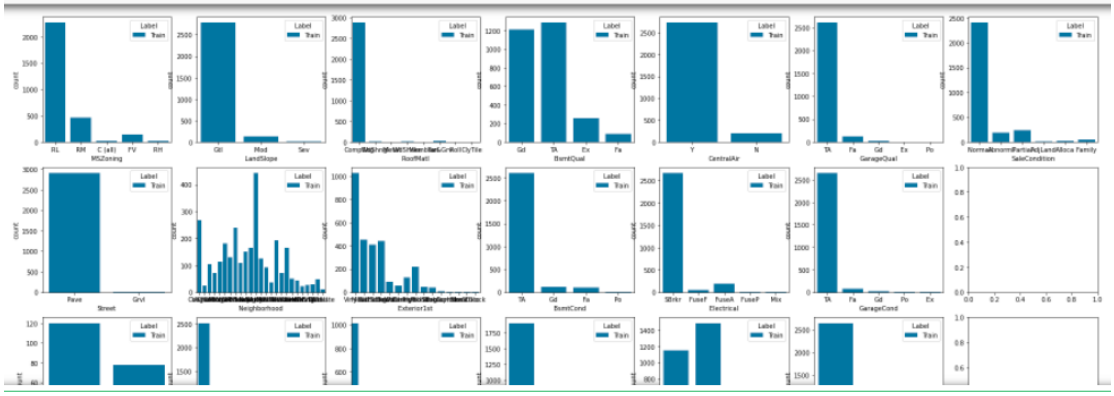
数值数据的线性分布：

```
f,axes = plt.subplots(7,6,figsize=(30,30),sharex=False)
for i,feature in enumerate(numerical_features):
    sns.scatterplot(data=combined_df,x=feature,y="SalePrice",ax=axes[i%7,i//7])
```



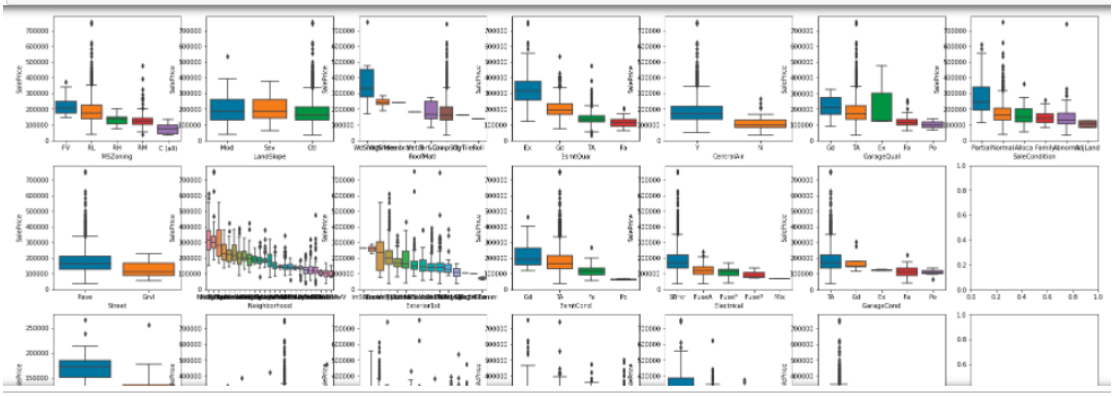
非数值数据柱状图分析

```
# 对比非数值型数据分析
f,axes = plt.subplots(7,7,figsize=(30,30),sharex=False)
for i,feature in enumerate(categorical_features):
    sns.countplot(data=combined_df,x=feature,hue="Label",ax=axes[i%7,i//7])
```



非数值数据的分布（对应价格）：箱线图，选取相关性比较强且贴合生活逻辑的特征进行分析。（例如：价格和 neighborhood：若 neighorhood 在 stoneBr 和 NridgHt 附近，价格会较高、价格和 TotalBsmtSF：若地下室面积大，价格会高）

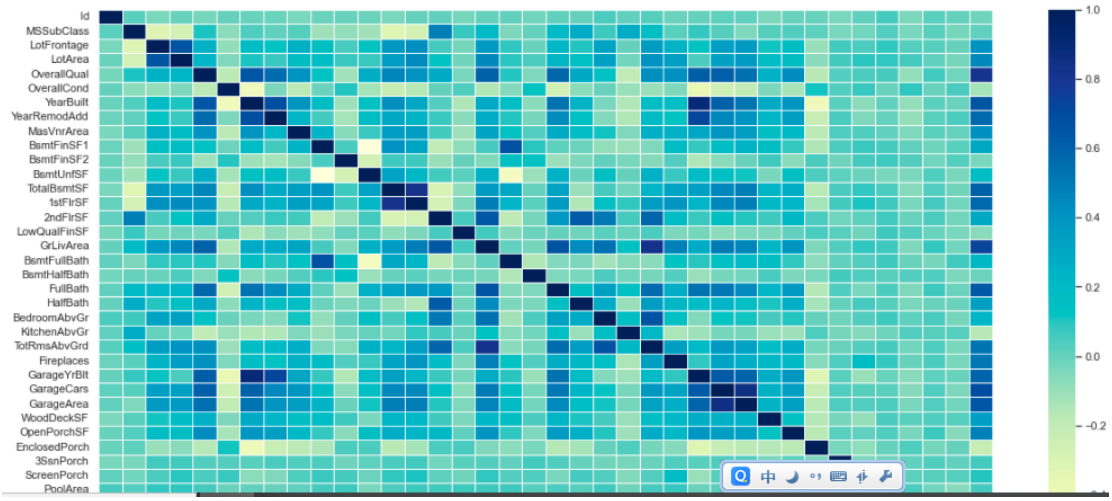
```
f, axes = plt.subplots(7,7 , figsize=(30, 30), sharex=False)
for i, feature in enumerate(categorical_features):
    sort_list = sorted(combined_df.groupby(feature)['SalePrice'].median().items(), key= lambda x:x[1], reverse = True)
    order_list = [x[0] for x in sort_list ]
    sns.boxplot(data = combined_df, x = feature, y = 'SalePrice', order=order_list, ax=axes[i%7, i//7])
plt.show()
```



热力图分析数据之间的相关性

```
# 使用热力图查看特征之间的相互关系
corrmat = train_data.corr(method='spearman') # 计算不同数据之间的相关数
plt.figure(figsize=(20,10))
sns.heatmap(corrmat,cmap="YlGnBu", linewidths=.5)
```

```
<AxesSubplot:>
```



4.数据预处理

本项目的数据处理分为两个方面：数据分析和数据预处理，前文已经概括了数据分析的大致步骤，本章主要介绍数据预处理的具体操作。

数据预处理一般包括数据审核、数据筛选和数据排序三个步骤。数据审核对于原始数据主要从完整性和准确性两个方面去审核。完整性审核主要是检查应调查的单位或个体是否有遗漏，所有的调查项目或指标是否填写齐全。准确性审核主要是包括两个方面：一是检查数据资料是否真实地反映了客观实际情况，内容是否符合实际；二是检查数据是否有错误，计算是否正确等。数据审核后，我们需要对审核过程中发现的错误应尽可能予以纠正。调查结束后，当数据发现的错误不能予以纠正，或者有些数据不符合调查的要求而又无法弥补时，就需要对数据进行筛选。数据筛选包括两方面的内容：一是将某些不符合要求的数据或有明显错误地数据予以剔除；二是将符合某种特定条件的数据筛选出来，对不符合特定条件的数据予以剔除。数据排序是按照一定顺序将数据排列，以便于研究者通过浏览数据发现一些明显的特征或趋势，找到解决问题的线索。除此之外，排序还有助于对数据检查纠错，为重新归类或分组等提供依据。

数据预处理中有数据清洗、数据集成、数据变换、数据规约等方法，本项目数据预处理主要涉及数据清洗。数据处理主要分为以下方法：

1.解决不完整数据（即值缺失）的方法

大多数情况下，缺失的值必须手工填入（即手工清理）。当然，某些缺失值可以从本数据源或其它数据源推导出来，这就可以用平均值、最大值、最小值或更为复杂的概率估计代替缺失的值，从而达到清理的目的。

2.错误值的检测及解决方法

用统计分析的方法识别可能的错误值或异常值，如偏差分析、识别不遵守分布或回归方程的值，也可以用简单规则库（常识性规则、业务特定规则等）检查数据值，或使用不同属性间的约束、外部的数据来检测和清理数据。

3.重复记录的检测及消除方法

数据库中属性值相同的记录被认为是重复记录,通过判断记录间的属性值是否相等来检测记录是否相等，相等的记录合并为一条记录（即合并/清除)。合并/清除是消重的基本方法。

4.不一致性（数据源内部及数据源之间）的检测及解决方法

从多数据源集成的数据可能有语义冲突，可定义完整性约束用于检测不一致性，也可通过分析数据发现联系，从而使得数据保持一致。开发的数据清理工具大致可分为三类。

本项目主要以上述方法为参照对数据进行处理。

数据预处理方法：主要使用 numpy、pandas、sklearn 第三方库进行数据分析，使用 matplotlib 进行图表分析。

（1） 特征删除

（2） 修改与时间相关的特征

（3） 填充缺失值

（4） 将某些数值型特征转换为非数值型特征

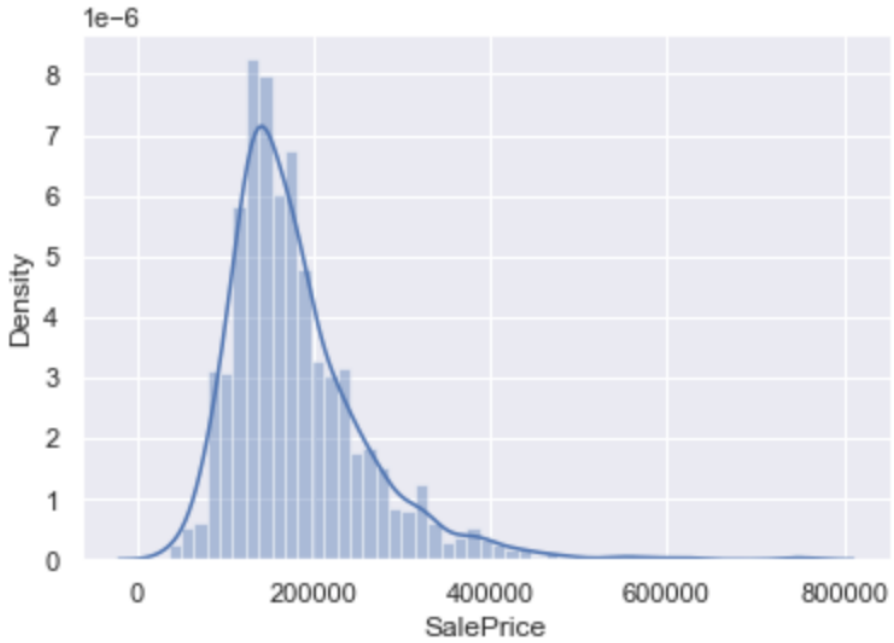
（5） 使用 sklearn 中的 PowerTransformer 使其更具高斯分布

（6） 对非数值特征中的某些特征进行融合

（7） 对非数值特征进行编码

在数据预处理中，我们根据之前数据分析得出数据集的规模、影响房价的因素数量以及房价的大致分布。

```
<AxesSubplot:xlabel='SalePrice', ylabel='Density'>
```



根据图片可以了解到房价数据类似于泊松分布。我们再进一步计算房价的峰度和偏度。

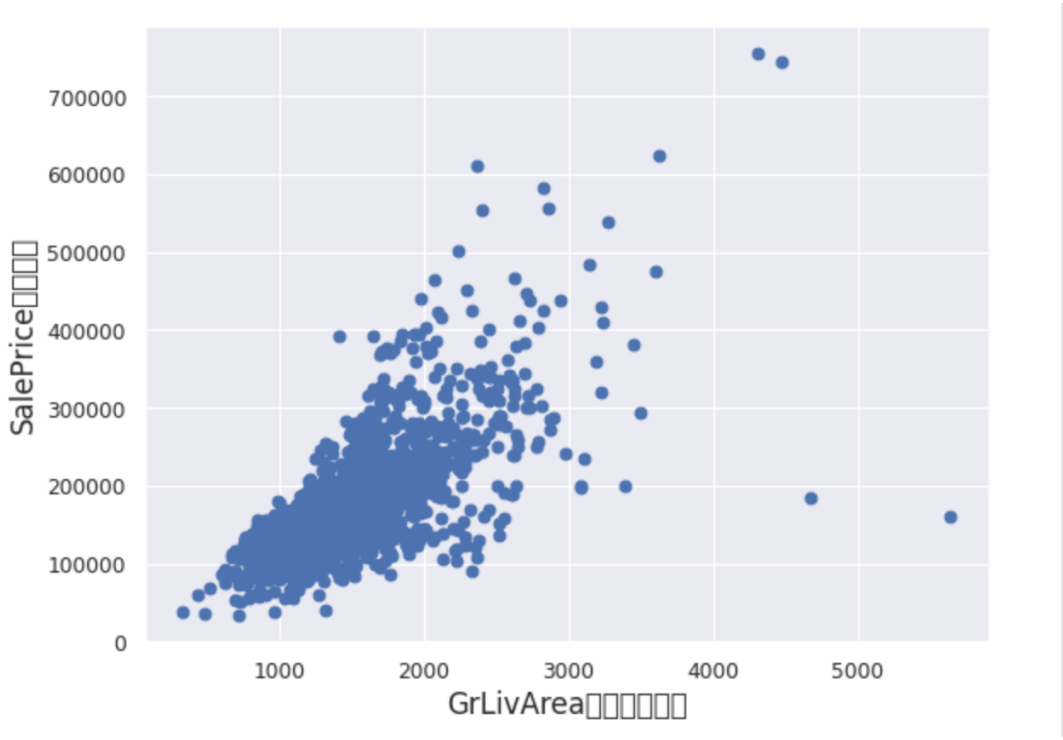
```
: print("峰度(kurtosis)为: {}".format(df_train['SalePrice'].kurt()))
```

```
峰度(kurtosis)为: 6.536281860064529
```

```
: print("偏度(skewness)为: {}".format(df_train['SalePrice'].skew()))
```

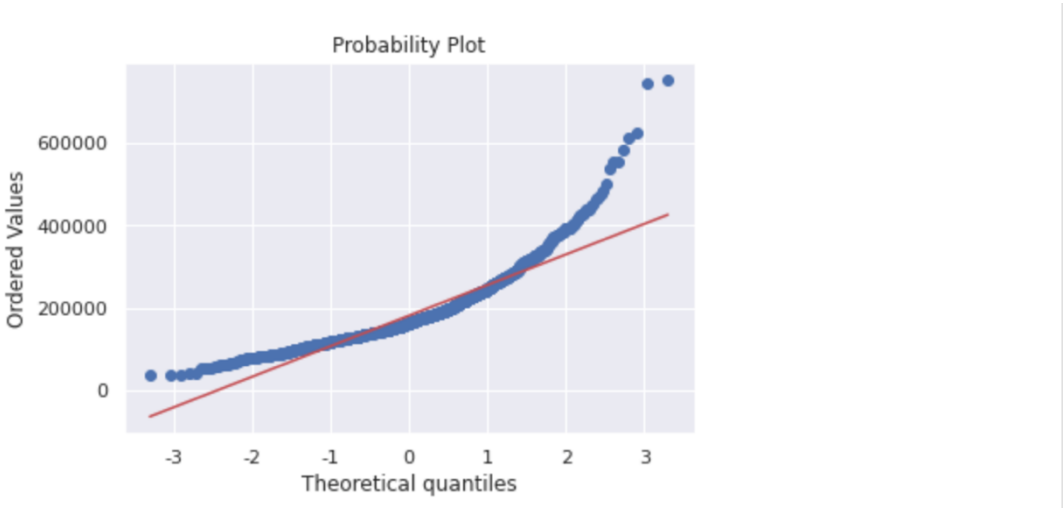
```
偏度(skewness)为: 1.8828757597682129
```

我们发现偏度较大，后期可以考虑对偏度进行调整。我们认为房屋面积与房价有一定相关性，我们尝试画出房屋面积与房价的散点图。

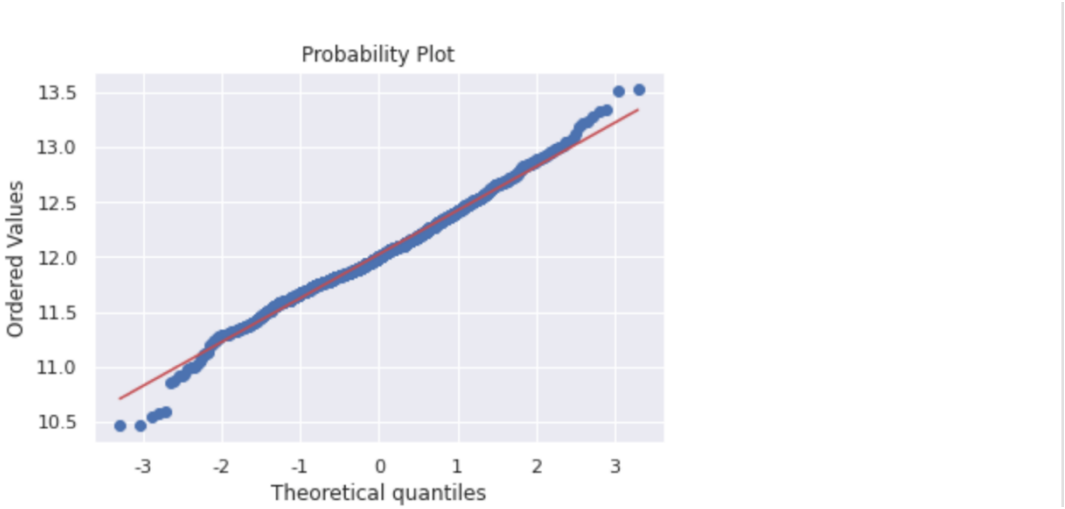


我们发现右下角有两个离群点，并将其删除。

因为房价类似于泊松分布，我们可以考虑将房价数据正态分布转换。下图为数据正态分布转换前的 Q-Q 图。



下图为正态分布转化后的 Q-Q 图。



将房价数据处理后，我们选择与房价相关性最高的前十个特征，查看其缺失值，数据缺失率，再对缺失值填充。经过一定的尝试，我们将数值型缺失值填 0，对非数值型数据进行处理，使其转换成连续的数值型变量。再对特征进行增加、倾斜等行为，然后计算其特征偏度。对偏态分布的数据进行标准化处理，使其更加服从正态分布。

5.模型建立与调整

房价预测问题作为一道经典的 kaggle 题目，网络上已经有许多成熟的模型和比较好的结果可供参考。在对众多开源的代码和方法进行研究之后，我们选取了使用较多、效果较好的几个模型对结果进行预测。

首先使用 lazypredict 库观察多个模型的大致效果（未进行调参）：

```
#调用lazypredict观察多个模型效果
x_train1, x_test1, y_train1, y_test1 = train_test_split(X_train, y_train, test_size=0.25)
reg = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric=None)
train, test = reg.fit(x_train1, x_test1, y_train1, y_test1)
test
```

```
100%|██████████| 42/42 [00:40<00:00,  1.05it/s]
```

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
LassoCV	0.78	0.91	0.12	0.64
LassoLarsCV	0.78	0.91	0.12	0.42
ElasticNetCV	0.78	0.91	0.12	0.60
LarsCV	0.77	0.91	0.13	0.83
HuberRegressor	0.76	0.91	0.13	0.31
PoissonRegressor	0.76	0.91	0.13	0.04
BayesianRidge	0.76	0.91	0.13	0.16
RidgeCV	0.76	0.90	0.13	0.60

在此选用均方根误差 rmse 定义验证函数：

```
#定义验证函数，使用均方根误差
def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X_train, y_train, scoring="neg_mean_squared_error", cv = 10))
    return (rmse)
```

根据对模型的掌握情况以及总体效果的比较，首先选择 LassoCV：

```
#Lasso
#alphas为备选的alpha，lassocv可以从中自动得到最优alpha
clf_lasso = LassoCV(alphas = [0.00005, 0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0])
clf_lasso.fit(X_train, y_train)
#保证数据有效性，防止数据太小无法显示
lasso_preds = np.expm1(clf_lasso.predict(X_test))

score_lasso = rmse_cv(clf_lasso)
#std:标准差
print("\nLasso score: {:.4f} ({:.4f})\n".format(score_lasso.mean(), score_lasso.std()))
```

```
Lasso score: 0.1099 (0.0140)
```

与网络上的众多结果进行比较，0.11 及以下的分数可以达到该竞赛提交结果中前 10%左右的水平，因此可以看出该模型拟合效果是比较好的。

接下来使用与 LassoCV 同为线性回归的 RidgeCV：

```
#岭回归
clf_ridge = RidgeCV(alphas = [0.00005, 0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0
clf_ridge.fit(X_train, y_train)
ridge_preds = np.expm1(clf_ridge.predict(X_test))

score_ridge = rmse_cv(clf_ridge)
print("\nRidge score: {:.4f} ({:.4f})\n".format(score_ridge.mean(), score_ridge.std()))
```

```
Ridge score: 0.1114 (0.0137)
```

可以看到，该模型的效果与 lasso 相差较小，也是比较好的。

再使用 ElasticNet：

In [96]:
```
#ElasticNet
clf_en = ElasticNetCV(alphas = [0.00005, 0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01], l1_ratio = [0.1, 0.2, 0.5, 0.
clf_en.fit(X_train, y_train)
elas_preds = np.expm1(clf_en.predict(X_test))

score_en = rmse_cv(clf_en)
print("\nElasticNet score: {:.4f} ({:.4f})\n".format(score_en.mean(), score_en.std()))
```

```
ElasticNet score: 0.1098 (0.0141)
```

以上三个模型的参数调节是模型自动进行的，最终得到的结果都在 0.11 左右。接下来考虑与这几个模型存在差异的 xgboost 方法和梯度提升 gbr 方法。

为了方便对这两个模型进行超参数调优，采用一个自动的超参数优化框架 optuna。optuna 的代码具有高度的模块特性，并且用户可以根据自己的希望动态构造超参数的搜索空间。

```
#超参数调整，使用optuna框架
def tune(objective):
    study = optuna.create_study(direction='maximize')
    study.optimize(objective,n_trials=100)

    params = study.best_params
    best_score = study.best_value
    print(f"Best score: {best_score} \nOptimized parameters: {params}")
    return params
```

首先使用 xgboost 方法：

```
#XGBoost
#目标函数
RANDOM_SEED = 1
def xgb_objective(trial):
    _n_estimators = trial.suggest_int("n_estimators", 50, 2000)
    _max_depth = trial.suggest_int("max_depth", 1, 20)
    _learning_rate = trial.suggest_float("learning_rate", 0.01, 1)
    _gamma = trial.suggest_float("gamma", 0.01, 1)
    _min_child_weight = trial.suggest_float("min_child_weight", 0.1, 10)
    _subsample = trial.suggest_float('subsample', 0.01, 1)
    _reg_alpha = trial.suggest_float('reg_alpha', 0.01, 10)
    _reg_lambda = trial.suggest_float('reg_lambda', 0.01, 10)


    xgbr = xgb.XGBRegressor(
        n_estimators=_n_estimators,
        max_depth=_max_depth,
        learning_rate=_learning_rate,
        gamma=_gamma,
        min_child_weight=_min_child_weight,
        subsample=_subsample,
        reg_alpha=_reg_alpha,
        reg_lambda=_reg_lambda,
        random_state=RANDOM_SEED,
    )
    score = cross_val_score(
        xgbr, X_train,y_train, cv=10, scoring="neg_root_mean_squared_error"
    ).mean()
    return score

xgb_params = tune(xgb_objective)
```

该段代码会显式呈现出 optuna 调参过程：

```
[I 2022-01-09 20:12:58,736] Trial 95 finished with value: -0.2701282497889983 and parameters: {'n_estimators': 309, 'max_depth': 10, 'learning_rate': 0.012928202492163187, 'gamma': 0.07656881427165355, 'min_child_weight': 7.767269119990805, 'subsample': 0.9766839064930225, 'reg_alpha': 2.6637394538285672, 'reg_lambda': 2.265873679710825}. Best is trial 87 with value: -0.12620168048400926.
[I 2022-01-09 20:13:10,030] Trial 96 finished with value: -0.13132584202221417 and parameters: {'n_estimators': 170, 'max_depth': 11, 'learning_rate': 0.13509880394534646, 'gamma': 0.0561813552968549, 'min_child_weight': 8.943281801955402, 'subsample': 0.9912625029270448, 'reg_alpha': 3.547377338863444, 'reg_lambda': 0.9383819116920898}. Best is trial 87 with value: -0.12620168048400926.
[I 2022-01-09 20:13:18,336] Trial 97 finished with value: -0.13896823021731924 and parameters: {'n_estimators': 105, 'max_depth': 12, 'learning_rate': 0.4685815902450403, 'gamma': 0.029267010537355836, 'min_child_weight': 8.28211337713217, 'subsample': 0.9999902366152674, 'reg_alpha': 2.161120288025211, 'reg_lambda': 1.9379543179756884}. Best is trial 87 with value: -0.12620168048400926.
[I 2022-01-09 20:13:52,992] Trial 98 finished with value: -0.12928255913917158 and parameters: {'n_estimators': 417, 'max_depth': 20, 'learning_rate': 0.06519312971039812, 'gamma': 0.14270076009506438, 'min_child_weight': 7.199983474371219, 'subsample': 0.946205465160446, 'reg_alpha': 1.8028258223218248, 'reg_lambda': 2.514471473547573}. Best is trial 87 with value: -0.12620168048400926.
[I 2022-01-09 20:14:27,644] Trial 99 finished with value: -0.12822161040506996 and parameters: {'n_estimators': 415, 'max_depth': 16, 'learning_rate': 0.09293439060349269, 'gamma': 0.13762708990016093, 'min_child_weight': 7.101619519742242, 'subsample': 0.9487938765905958, 'reg_alpha': 1.2369047987560782, 'reg_lambda': 2.5546428350270123}. Best is trial 87 with value: -0.12620168048400926.

Best score: -0.12620168048400926
Optimized parameters: {'n_estimators': 242, 'max_depth': 11, 'learning_rate': 0.063240608037147, 'gamma': 0.018400603767069035, 'min_child_weight': 7.776063920611367, 'subsample': 0.8918358575346695, 'reg_alpha': 3.5532409719812144, 'reg_lambda': 0.8675471544593278}
```

将迭代后得到的参数列表代入到模型中，得到拟合效果：

In [52]:
```
#计算效果
xgb_params = {'n_estimators': 242, 'max_depth': 11, 'learning_rate': 0.063240608037147,
clf_xgb = xgb.XGBRegressor(random_state=RANDOM_SEED, **xgb_params)
clf_xgb.fit(X_train,y_train)
xgb_preds = np.expm1(clf_xgb.predict(X_test))

score_xgb = rmse_cv(clf_xgb)
print("\nxgb score: {:.4f} ({:.4f})\n".format(score_xgb.mean(), score_xgb.std()))
```

```
xgb score: 0.1262 (0.0130)
```

对梯度提升算法采取相同的处理措施：

```
#梯度提升
def gbr_objective(trial):
    _n_estimators = trial.suggest_int("n_estimators", 50, 2000)
    _learning_rate = trial.suggest_float("learning_rate", 0.01, 1)
    _max_depth = trial.suggest_int("max_depth", 1, 20)
    _min_samples_split = trial.suggest_int("min_samples_split", 2, 20)
    _min_samples_leaf = trial.suggest_int("min_samples_leaf", 2, 20)
    _max_features = trial.suggest_int("max_features", 10, 50)

    gbr = GradientBoostingRegressor(
        n_estimators=_n_estimators,
        learning_rate=_learning_rate,
        max_depth=_max_depth,
        max_features=_max_features,
        min_samples_leaf=_min_samples_leaf,
        min_samples_split=_min_samples_split,

        random_state=RANDOM_SEED,
    )

    score = cross_val_score(
        gbr, X_train,y_train, cv=10, scoring="neg_root_mean_squared_error"
    ).mean()
    return score

gbr_params = tune(gbr_objective)
```

```
5959134859466.
[I 2022-01-09 20:53:40,735] Trial 95 finished with value: -0.11000667870913303 and parameters: {'n_estimators': 1130, 'learning_rate': 0.0
37837533105861874, 'max_depth': 3, 'min_samples_split': 11, 'min_samples_leaf': 6, 'max_features': 19}. Best is trial 93 with value: -0.10
975959134859466.
[I 2022-01-09 20:53:56,439] Trial 96 finished with value: -0.11217506678967364 and parameters: {'n_estimators': 1137, 'learning_rate': 0.0
37586005311183 89, 'max_depth': 4, 'min_samples_split': 11, 'min_samples_leaf': 6, 'max_features': 18}. Best is trial 93 with value: -0.109
75959134859466.
[I 2022-01-09 20:54:08,577] Trial 97 finished with value: -0.11776117713659082 and parameters: {'n_estimators': 1164, 'learning_rate': 0.1
3138307294020493, 'max_depth': 3, 'min_samples_split': 11, 'min_samples_leaf': 6, 'max_features': 17}. Best is trial 93 with value: -0.109
75959134859466.
[I 2022-01-09 20:54:18,829] Trial 98 finished with value: -0.11339507346791591 and parameters: {'n_estimators': 1023, 'learning_rate': 0.0
51047508559702894, 'max_depth': 3, 'min_samples_split': 11, 'min_samples_leaf': 5, 'max_features': 15}. Best is trial 93 with value: -0.10
975959134859466.
[I 2022-01-09 20:54:57,867] Trial 99 finished with value: -0.12070197243448803 and parameters: {'n_estimators': 1118, 'learning_rate': 0.0
38726389261245, 'max_depth': 13, 'min_samples_split': 14, 'min_samples_leaf': 5, 'max_features': 19}. Best is trial 93 with value: -0.1097
5959134859466.

Best score: -0.10975959134859466
Optimized parameters: {'n_estimators': 1122, 'learning_rate': 0.0362725462022589, 'max_depth': 3, 'min_samples_split': 12, 'min_samples_le
af': 6, 'max_features': 23}
```

```
In [54]: gbr_params = {'n_estimators': 1122, 'learning_rate': 0.0362725462022589, 'max_depth': 3, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max
         clf_gbr = GradientBoostingRegressor(random_state=RANDOM_SEED, **gbr_params)
         clf_gbr.fit(X_train,y_train)
         gbr_preds = np.expm1(clf_gbr.predict(X_test))

         score_gbr = rmse_cv(clf_gbr)
         print("\ngbr score: {:.4f} ({:.4f})\n".format(score_gbr.mean(), score_gbr.std()))

gbr score: 0.1098 (0.0168)
```

最终对模型进行一个简单的融合，依据每个模型效果对其分配权重：

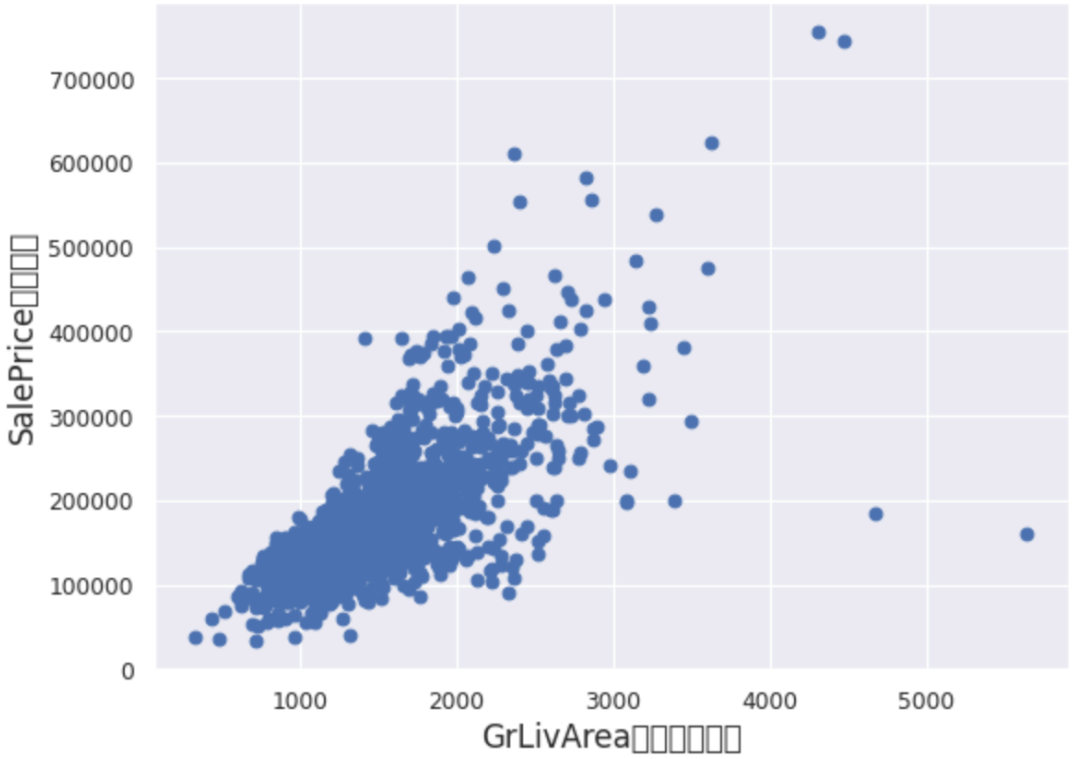
```
#模型融合
final_result = 0.3*lasso_preds + 0.4*gbr_preds + 0.1*xgb_preds +0.1*ridge_preds + 0.1*elas_preds
#solution = pd.DataFrame({"id":test.index+1461, "SalePrice":final_result}, columns=['id', 'SalePrice'])
#solution.to_csv("result.csv", index = False)
```

（由于将结果进行提交需要翻墙注册，目前用的 vpn 未能成功，因此没有最终提交到官网）

6.创新与困难

在数据预处理部分，从业务角度理解， 对于一些对房价有重要意义的变量，根据该变量和目标值之间的关系绘制散点图检查异常值点，计算偏度和峰度，对于存在的不合理的离群点，将其删除。

例如 :GrLivArea 与 SalePrice 的关系图中 ,有两个离群的 GrLivArea 值很高的数据，可以推测出现这种情况的原因，或许他们代表了农业地区，因此出现了低价。 这两个点很明显不能代表典型样例，所以我们将它们定义为异常值并删除。



在这个部分，我们没有笼统地将每个变量与目标值的关系都进行异常值检查和偏度、峰度计算，而是从实际问题出发，进行业务方面的分析，挑选出对目标重要性高的变量，进行更有针对性的处理，这样能使数据预处理的过程更有效率，避免过多冗余步骤。

模型部分的创新点在于综合使用了 lazypredict、optuna 等框架对问题进行了比较直观、方便的分析，改善了建模效果，使代码逻辑更加清晰。遇到的困难主要有调用的库较多时，不同版本之间可能会产生冲突，解决方法是认真地找出有问题的库，根据网上的一些方

案重新安装另外的版本。以及原先对于超参数调优的掌握不足，不能调出很好的参数，此时需要对此进行更多的学习和了解，最终找出了合适的方法。

7.小组分工与贡献

数据分析：赵翠玉，对训练集和测试集的数据进行分析，通过对比、画图等手段使得数据更加直观，更加易于分析。将分析结果传递给其他组员。

数据处理：陆泽洁、张蓉，接收分析结果，基于分析结果和模型需求做数据处理。

模型：薛惠文，用处理后的数据进行模型融合。

在此过程中大家通力合作，互相交流，共同完成了该项目的研究。

参考文献与开源代码

[1]https://blog.csdn.net/qq_36441393/article/details/88229814

[2]https://blog.csdn.net/chandelierds/article/details/83627060?utm_medium=distr
ibute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-
1.opensearchhbase&spm=1001.2101.3001.4242.2

[3]https://blog.csdn.net/jerry_liufeng/article/details/119360894?spm=1001.2101.3
001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-
2%7Edefault%7ECTRLIST%7Edefault-1.highlightwordscore&depth_1-
utm_source=distribute.pc_relevant.none-task-blog-
2%7Edefault%7ECTRLIST%7Edefault-1.highlightwordscore

[4]https://blog.csdn.net/u012063773/article/details/79349256?spm=1001.2101.30
01.6650.7&utm_medium=distribute.pc_relevant.none-task-blog-
2%7Edefault%7EBlogCommendFromBaidu%7Edefault-7.fixedcolumn&depth_1-
utm_source=distribute.pc_relevant.none-task-blog-
2%7Edefault%7EBlogCommendFromBaidu%7Edefault-7.fixedcolumn

[5]https://zhuanlan.zhihu.com/p/110095263

[6]https://blog.csdn.net/jerry_liufeng/article/details/119360894

[7]https://blog.csdn.net/u012063773/article/details/79349256?spm=1001.2101.30
01.6650.7&utm_medium=distribute.pc_relevant.none-task-blog-
2%7Edefault%7EBlogCommendFromBaidu%7Edefault-7.fixedcolumn&depth_1-
utm_source=distribute.pc_relevant.none-task-blog-
2%7Edefault%7EBlogCommendFromBaidu%7Edefault-7.fixedcolumn

[8]https://blog.csdn.net/jerry_liufeng/article/details/119380211