本篇笔记主要是对 house price 比赛的优秀案例进行总结的笔记 - [ ] 阅读特征工程笔记[stack\_regression 阅读笔记](stack_regression%20阅读笔记.md) - [ ] 阅读[Regularized Linear Models](https://www.kaggle.com/apapiu/regularized-linear-models)

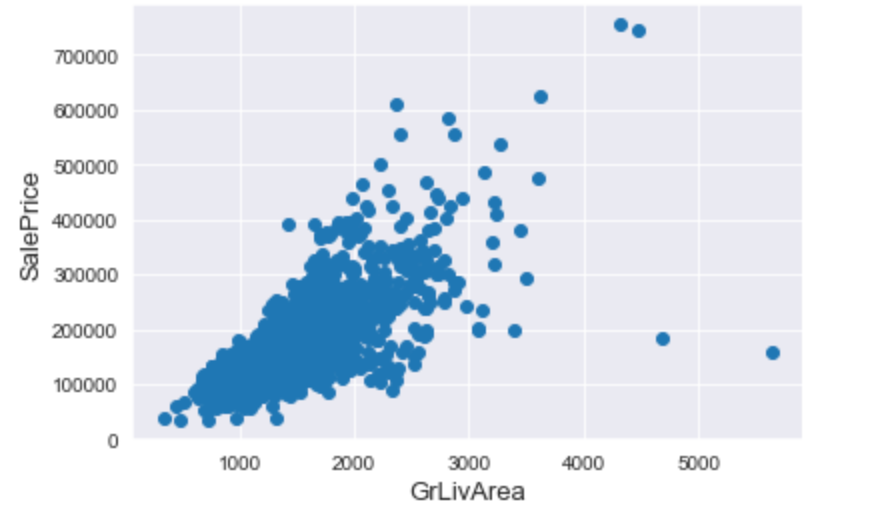
以上两篇文章都会对完成这次**回归任务**有很大的帮助

# 1. 了解数据

1. 首先**了解训**练集和测**试集**的\*\*样本数和特征数
2. 去除显而易见的无用**特**征(‘ID’)

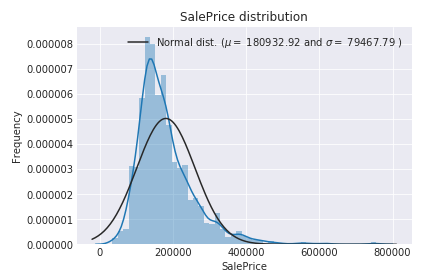
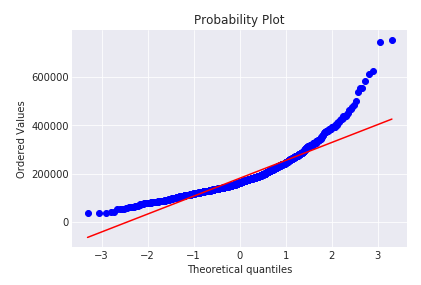
# 2. 数据处理

## 2.1 异常值

对于目标值和特征画出散点图，可以比较清晰的看出是否具有比较离谱的异常值，如下图所示：  上图可以看到，在坐标轴的右边，有两个离群点异常值（特别**极端**的异常)，可以讲这两个异常样本去除。 一般情况下，直接这样去除异常样本是很不安全的，但是对于这种特别**极端**的值可以放心的去除。

* 可能还有其他离群点在训练数据中。但是，如果在测试数据中也有离群点，则如果将它们全部删除，则可能会对我们的模型产生不良影响。因此，我们不会删除所有的离群点，而是在建模过程中，会使一些模型对它们具有鲁棒性。

## 2.2 目标变量

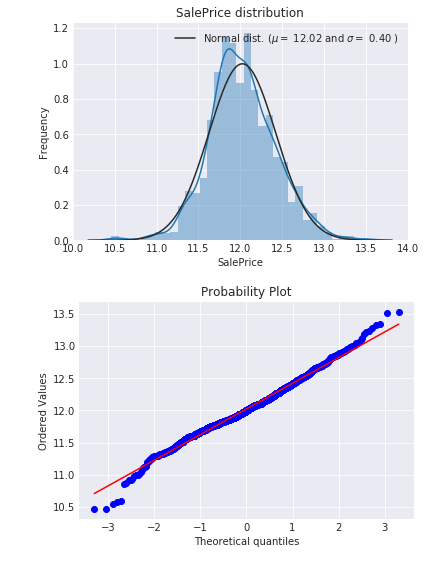
针对目标变量做一些分析 1. **偏度、峰度** 利用from scipy import stats方法可以得到某个变量的统计分布情况 (mu, sigma) = norm.fit(train['SalePrice']) 利用sns.displot(var)``方法可以得到某个变量var的**正态分布直方图以及核密度估计** python sns.distplot(train['SalePrice'] , fit=norm); # fit=norm 可以得到一条正态分布曲线  2. **QQ图**（检验样本数据概率分布，默认检验变量的正态分布） q-q 图是通过比较数据和正态分布的**分位数**是否相等来判断数据是不是符合正态分布 python stats.probplot(train['SalePrice'], plot=plt) 

可以由上面两个图看到，目标变量的分布是属于\*\*右偏态\*\*的，但是<u>一般的线性模型都要求变量符合正态分布</u>，所以我们需要使得目标变量更加符合\*\*正态分布\*\*。

## 2.3 对数变换

np.log1p(train["SalePrice"])利用numpy进行对数变换

sns.distplot(train['SalePrice'] , fit=norm);  
  
# Get the fitted parameters used by the function  
(mu, sigma) = norm.fit(train['SalePrice'])  
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))  
  
#Now plot the distribution  
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],  
 loc='best')  
plt.ylabel('Frequency')  
plt.title('SalePrice distribution')  
  
#Get also the QQ-plot  
fig = plt.figure()  
res = stats.probplot(train['SalePrice'], plot=plt)  
plt.show()

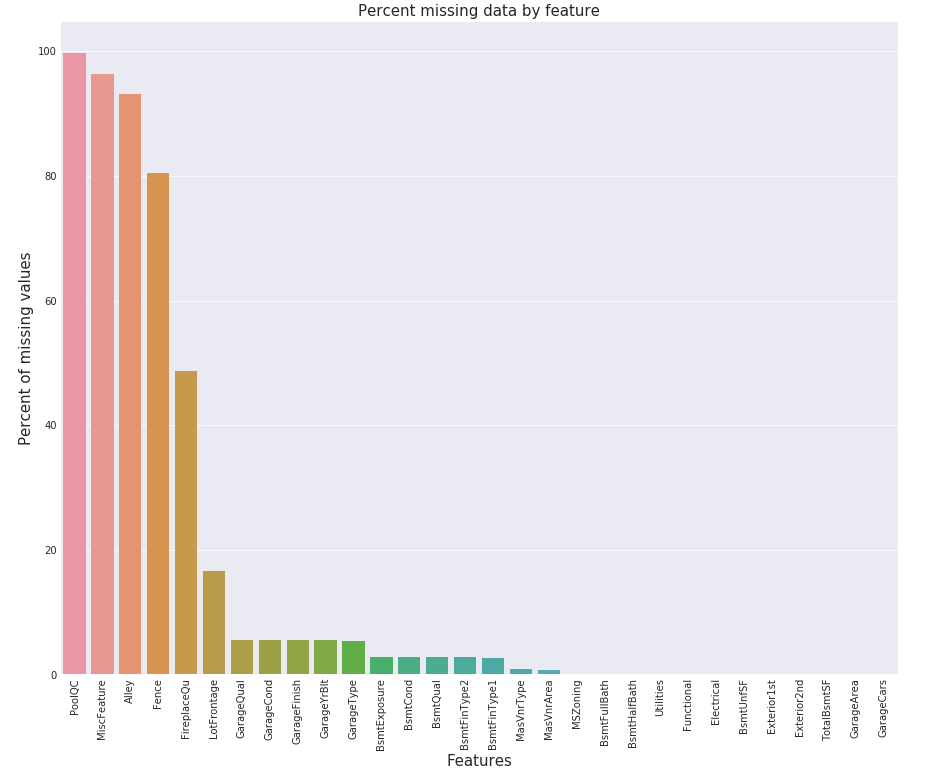
 可以看到**训练数据的目标变量**经过对数变换之后，更加符合正态分布。

# 3. 特征工程

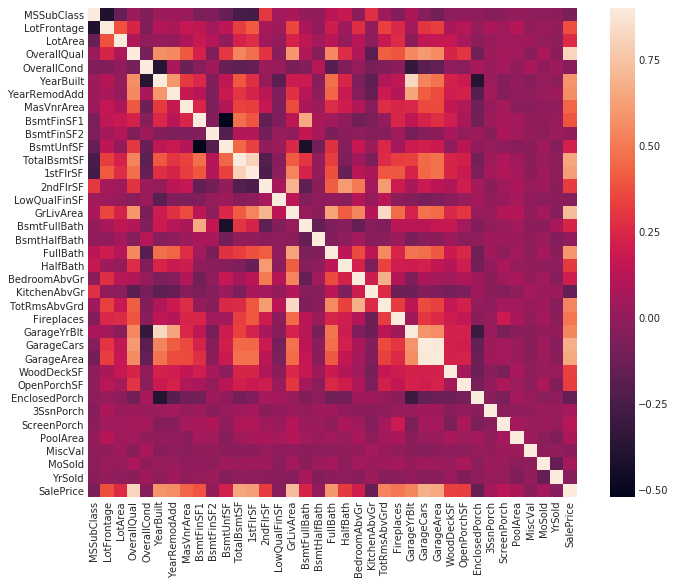
将训练数据和测试数据都concat在一起,但是去除目标变量。

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)

## 3.1 缺失值

统计出每个变量的缺失值比例，并画图  ## 3.2 特征相似 通过sns的热力图，就可以看到变量之间的相互关系（corr相关系数）

corrmat = train.corr()  
plt.subplots(figsize=(12,9))  
sns.heatmap(corrmat, vmax=0.9, square=True)

 ## 3.3 填充缺失值 通过按顺序处理缺失值的特征来填充它们（根据每个特征的特性：均值、定值、中位数、众数、None等） 注意这里不区分训练数据、测试数据

## 3.3 其他特征工程

1. **将某些数值变量转变成类别变量** ``all\_data[‘MSSubClass’].apply(str)
2. **对一些可能包含信息的分类变量进行标签编码，以反映它们的顺序集合**
3. **增加特征** 将某些特征组合（加减乘除）可以得到一个新的特征

* 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))  
    
   # shape   
   print('Shape all\_data: {}'.format(all\_data.shape))

1. **解决数据倾斜** 我们用**scipy函数boxcox1p**来计算Box-Cox转换，目标是找到一个简单的转换方式使数据规范化。
   * 首先得到哪些变量具有偏度 python 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)
   * 按序对特征进行对数变换
   * 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])

# 4. 建模

## 4.1 导入相关的库

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.metrics import mean\_squared\_error  
import xgboost as xgb  
import lightgbm as lgb

## 4.2 定义交叉验证策略

#Validation function  
n\_folds = 5  
  
def rmsle\_cv(model):  
 kf = KFold(n\_folds, shuffle=True, random\_state=42).get\_n\_splits(train.values)  
 rmse= np.sqrt(-cross\_val\_score(model, train.values, y\_train, scoring="neg\_mean\_squared\_error", cv = kf))  
 # 注意这里为负的均方误差，所以要取负号使其为正  
 return(rmse)

## 4.3 基模型

1. **Lasso 回归** lasso回归对于**异常值**特别敏感，如果希望它更加**健壮**，那么可以在pipeline中使用sklearn的**Robustscaler**方法 lasso = make\_pipeline(RobustScaler(), Lasso(alpha =0.0005, random\_state=1))
   * 关于[make\_pipeline](https://blog.csdn.net/elma_tww/article/details/88427695)
   * 关于[RobustScaler](https://scikit-learn.org.cn/view/751.html)
2. **弹性网络回归** 同样需要对于异常值进行处理 ENet = make\_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1\_ratio=.9, random\_state=3))
3. **核-岭回归**(Kernel Ridge Regression) 为什么需要引入**kernel**呢？其实，将kernel trick应用到distance-based的方法中是很直接的，因为kernel函数本身就是一个distance-based的函数。可能我们会发现，基于distance的方法，都会有对应的一个kernel版本的扩展。 此外，从实际应用来看， 因为数据可能是**非线性**的，单纯地假设真实数据服从线性关系，并用线性模型来回归真实的非线性数据，效果想必不会好。所以，引入kernel还能有一个好处，就是：引入kernel的RR，也就是KRR，能够**处理非线性数据**，即，将数据映射到某一个核空间，使得数据在这个核空间上**线性可分**。 KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
4. **Gradient Boosting Regression** [什么是huber损失？](https://zhuanlan.zhihu.com/p/358103958) 使用huber损失会使得模型对于异常值更加的健壮
5. **XGBoost**

* model\_xgb = xgb.XGBRegressor(colsample\_bytree=0.4603, gamma=0.0468,   
   learning\_rate=0.05, max\_depth=3,   
   min\_child\_weight=1.7817, n\_estimators=2200,  
   reg\_alpha=0.4640, reg\_lambda=0.8571,  
   subsample=0.5213, silent=1,  
   random\_state =7, nthread = -1)

1. **LightGBM**

* 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)

## 4.4 基础模型的表现

在交叉验证的策略下查看每个基础模型的表现状况

score = rmsle\_cv(lasso)  
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

## 4.5 stacking model

### 4.5.1 平均基模型

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)

每个模型都会被训练到，最终取他们的平均预测结果作为最终的结果输出。