目录

**[一、 数据准备](#_Toc14273_WPSOffice_Level1)** **[1](#_Toc14273_WPSOffice_Level1)**

**[二、 环境准备](#_Toc24861_WPSOffice_Level1)** **[1](#_Toc24861_WPSOffice_Level1)**

**[三、 代码编写](#_Toc32493_WPSOffice_Level1)** **[2](#_Toc32493_WPSOffice_Level1)**

[1. 数据读取](#_Toc24861_WPSOffice_Level2) [2](#_Toc24861_WPSOffice_Level2)

[2. 数据分析](#_Toc32493_WPSOffice_Level2) [2](#_Toc32493_WPSOffice_Level2)

[3. 数据清洗](#_Toc30608_WPSOffice_Level2) [3](#_Toc30608_WPSOffice_Level2)

[4. 特征工程](#_Toc4385_WPSOffice_Level2) [5](#_Toc4385_WPSOffice_Level2)

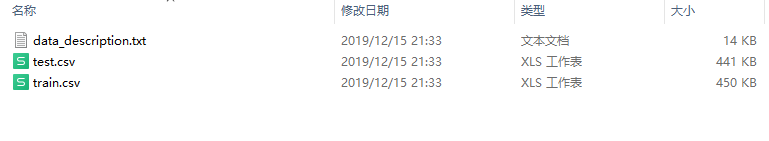
[5. 模型训练](#_Toc3317_WPSOffice_Level2) [5](#_Toc3317_WPSOffice_Level2)

**[四、生成结果](#_Toc30608_WPSOffice_Level1)** **[9](#_Toc30608_WPSOffice_Level1)**

# 数据准备

本次实验使用的是房价预测的数据集，详见：<https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data>

数据下载完成后，将其解压，将得到以下数据



# 环境准备

可使用pycharm、VSCode或者notebook进行代码的编写及测试。在编辑器中新建main.py文件。

# 代码编写

以下代码均为截取，完整代码可查看源码。

## 数据读取

使用pandas模块的read\_csv读取训练集以及测试集。

base\_dir = r"D:\ACoder\AllMyLab\大数据实训\大数据人工智能大作业\house-prices-advanced-regression-techniques"

data\_train = pd.read\_csv(path.join(base\_dir,'train.csv'))

data\_test = pd.read\_csv(path.join(base\_dir,'test.csv'))

## 数据分析

使用pandas\_profiling生成详细分析报告。此外棵通过调用dataframe.corr()函数查看各维度的相关性。

# 生成详细分析报告

profile = ProfileReport(data\_train,title="Pandas Profiling Report")

profile.to\_file("analysis.html")

#%%

num = data\_train.select\_dtypes(exclude='object')

# corr()计算各维度数据的相关性。

numcorr = num.corr()

f,ax = plt.subplots(figsize=(17,1))

sns.heatmap(numcorr.sort\_values(by=['SalePrice'],ascending=False).head(1),cmap="Blues")

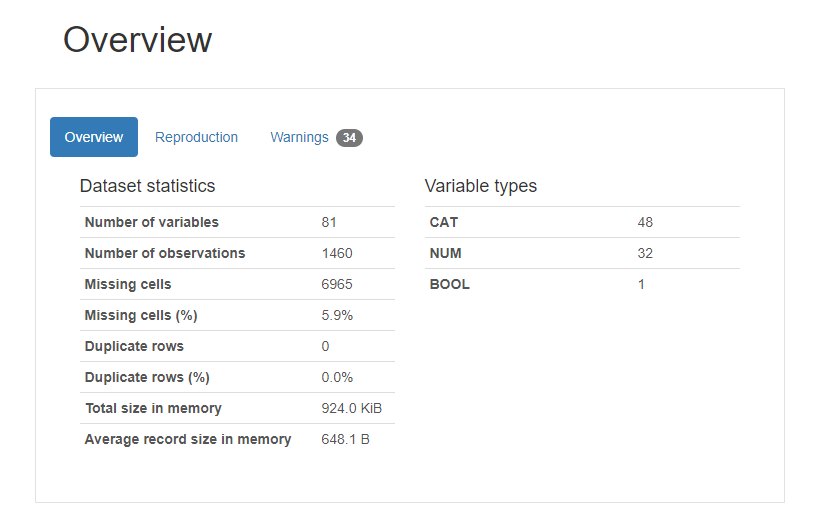
plt.title(" Numerical features correlation with the sale price", weight='bold', fontsize=18)

plt.xticks(weight='bold')

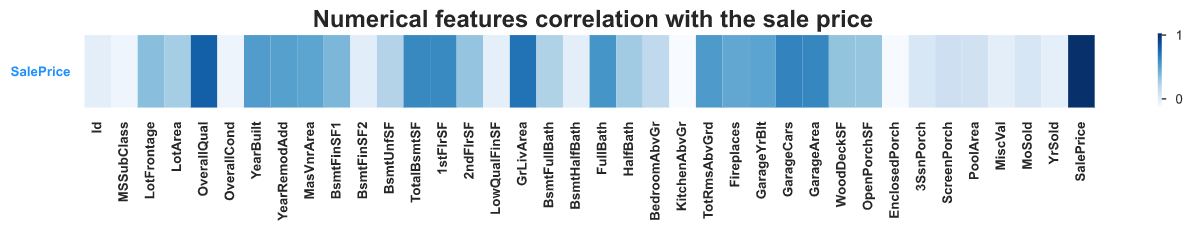
plt.yticks(weight='bold', color='dodgerblue', rotation=0)

plt.show()

运行成功后将生成analysis.html文件，浏览器打开后如下所示：



此外会生成一张热力图，如下所示：



## 数据清洗

该过程主要对于缺失值过多的特征进行删除，对于缺失值较少的特征通过平均值、临近值、零值、None值、中值等方式进行填充。

# %%数据清洗=================================================

y\_train = data\_train['SalePrice'].to\_frame()

#将训练集以及测试集合并

# pandas contact 之后，一定要记得用reset\_index去处理index,不然容易出现莫名的逻辑错误

# drop=True就是把原来的索引index列去掉，重置index。

data1 = pd.concat((data\_train,data\_test),sort=False).reset\_index(drop=True)

# inplace=True：不创建新的对象，直接对原始对象进行修改；

# inplace=False：对数据进行修改，创建并返回新的对象承载其修改结果。

data1.drop(['SalePrice'],axis=1,inplace=True)

data1.drop(['Id'],axis=1,inplace=True)

print("Total size is : ",data1.shape)

#%%

# 如果缺失值超过百分之20，即不缺失的值未达到80%，删除该特征

data = data1.dropna(thresh=len(data1)\*0.8,axis=1)

print(data1.shape[1]-data.shape[1], " features is dropped")

print('The shape of the combined dataset after dropping features with more than 80% M.V.', data.shape)

#%%

# 接下来处理缺失值未超过百分之20的特征

allna = (data.isnull().sum()/len(data))\*100

allna = allna.drop(allna[allna == 0].index).sort\_values()

NA = data[allna.index.to\_list()]

NAcat = NA.select\_dtypes(include='object')

NAnum = NA.select\_dtypes(exclude='object')

print('We have :',NAcat.shape[1],'categorical features with missing values')

print('We have :',NAnum.shape[1],'numerical features with missing values')

# %%

#MasVnrArea: Masonry veneer area in square feet, the missing data means no veneer so we fill with 0

data['MasVnrArea']=data.MasVnrArea.fillna(0)

#LotFrontage has 16% missing values. We fill with the median

data['LotFrontage']=data.LotFrontage.fillna(data.LotFrontage.median())

#GarageYrBlt:  车库建成的年份

data['GarageYrBlt']=data["GarageYrBlt"].fillna(1980)

#对于剩下的列: Bathroom, half bathroom, basement related columns and garage related columns:

#我们会把它们的缺失值填充为0，即boolean值

# 下面是一些不好人工判断填充值的列，我们把它们填充为前一个数据的对应值。

fill\_cols = ['Electrical', 'SaleType', 'KitchenQual', 'Exterior1st',

             'Exterior2nd', 'Functional', 'Utilities', 'MSZoning']

for col in data[fill\_cols]:

    data[col] = data[col].fillna(method='ffill')

# method = 'ffill’意味着用前一个数据的对应值来填充

# 对于其他列，我们直接用0填充或者用None填充

#Categorical missing values

NAcols=data.columns

for col in NAcols:

    if data[col].dtype == "object":

        data[col] = data[col].fillna("None")

#Numerical missing values

for col in NAcols:

    if data[col].dtype != "object":

        data[col]= data[col].fillna(0)

## 特征工程

该过程主要将一些相关的特征进行组合，降低模型的复杂度和计算量。

# %%特征工程=================================================

data['TotalArea'] = data['TotalBsmtSF'] + data['1stFlrSF'] + data['2ndFlrSF'] + data['GrLivArea'] +data['GarageArea']

data['Bathrooms'] = data['FullBath'] + data['HalfBath']\*0.5

data['Year average']= (data['YearRemodAdd']+data['YearBuilt'])/2

data['MSSubClass'] = data['MSSubClass'].apply(str)

data['YrSold'] = data['YrSold'].astype(str)

# 独热编码

new\_data = pd.get\_dummies(data)

print("the shape of the original dataset",data.shape)

print("the shape of the encoded dataset",new\_data.shape)

print("We have ",new\_data.shape[1]- data.shape[1], 'new encoded features')

# 重新划分训练集测试集

train = new\_data[:len(data\_train)]

test = new\_data[len(data\_train):]

# 去掉一些离群点

train=train[(train['GrLivArea'] < 4600) & (train['MasVnrArea'] < 1500)]

target = data\_train[['SalePrice']]

pos = [1298,523,279]

target.drop(target.index[pos],inplace=True)

print( 'Train: ',train.shape[0], 'rows')

print('Target:', target.shape[0],'rows')

## 模型训练

### 使用ridge回归进行训练

Ridge回归即加上L2正则化项的线性回归。

# %% 模型训练=============================================

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import RobustScaler

# %%预处理

x\_train,x\_test,y\_train,y\_test = train\_test\_split(train,target,test\_size=0.3,

    random\_state = 0)

scalar = RobustScaler()

x\_train = scalar.fit\_transform(x\_train)

x\_test = scalar.fit\_transform(x\_test)

testX = scalar.transform(test)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

# %%

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_squared\_error

from sklearn.linear\_model import Ridge

import math

ridge = Ridge()

# parameters= {'alpha':[0.0001,0.0009,0.001,0.002,0.003,0.01,0.1,1,10,100]}

# 第一次尝试可发现alpha在10处获得最优解，故移动取值范围重新进行网格搜索。

parameters = {'alpha':[x for x in range(1,101)]}

ridge\_reg = GridSearchCV(ridge,param\_grid=parameters,scoring='neg\_mean\_squared\_error',cv=15)

ridge\_reg.fit(x\_train,y\_train)

print("The best value of Alpha is: ",ridge\_reg.best\_params\_)

print("The best score achieved is: ",math.sqrt(-ridge\_reg.best\_score\_))

ridge\_pred=math.sqrt(-ridge\_reg.best\_score\_)

#%%

ridge\_mod=Ridge(alpha=20)

ridge\_mod.fit(x\_train,y\_train)

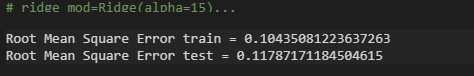
y\_pred\_train=ridge\_mod.predict(x\_train)

y\_pred\_test=ridge\_mod.predict(x\_test)

print('Root Mean Square Error train = ' + str(math.sqrt(mean\_squared\_error(y\_train, y\_pred\_train))))

print('Root Mean Square Error test = ' + str(math.sqrt(mean\_squared\_error(y\_test, y\_pred\_test))))

运行结果：



### 使用lasso回归进行训练

Lasso回归即加上L1正则化项的线性回归

# %%

from sklearn.linear\_model import Lasso

parameters= {'alpha':[0.0001,0.0009,0.001,0.002,0.003,0.01,0.1,1,10,100]}

lasso=Lasso()

lasso\_reg=GridSearchCV(lasso, param\_grid=parameters, scoring='neg\_mean\_squared\_error', cv=15)

lasso\_reg.fit(x\_train,y\_train)

print('The best value of Alpha is: ',lasso\_reg.best\_params\_)

#%%

lasso\_mod=Lasso(alpha=0.0009)

lasso\_mod.fit(x\_train,y\_train)

y\_lasso\_train=lasso\_mod.predict(x\_train)

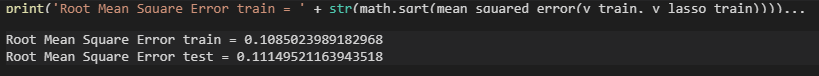
y\_lasso\_test=lasso\_mod.predict(x\_test)

#%%

print('Root Mean Square Error train = ' + str(math.sqrt(mean\_squared\_error(y\_train, y\_lasso\_train))))

print('Root Mean Square Error test = ' + str(math.sqrt(mean\_squared\_error(y\_test, y\_lasso\_test))))

运行结果：



### 使用ElasticNet进行训练

ElasticNet可简单理解为加上了L1和L2正则化项的线性回归。

# %%

from sklearn.linear\_model import ElasticNetCV

# 这里alphas是前面网格搜索的最佳结果

alphas = [0.000542555]

l1ratio = [0.1, 0.3,0.5, 0.9, 0.95, 0.99, 1]

elastic\_cv = ElasticNetCV(cv=5, max\_iter=1e7, alphas=alphas,  l1\_ratio=l1ratio)

elasticmod = elastic\_cv.fit(x\_train, y\_train.ravel())

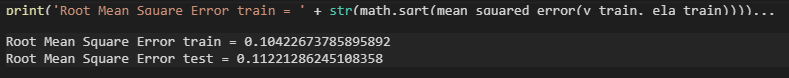
ela\_pred=elasticmod.predict(x\_test)

ela\_train = elasticmod.predict(x\_train)

print('Root Mean Square Error train = ' + str(math.sqrt(mean\_squared\_error(y\_train, ela\_pred))))

print('Root Mean Square Error test = ' + str(math.sqrt(mean\_squared\_error(y\_test, ela\_pred))))

运行结果：



### 使用XGBoost进行训练

# %%

from xgboost.sklearn import XGBRegressor

xgb= XGBRegressor(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,colsample\_bynode=1, colsample\_bytree=0.5, gamma=0,importance\_type='gain', learning\_rate=0.01, max\_delta\_step=0,max\_depth=3, min\_child\_weight=0, missing=None, n\_estimators=4000,n\_jobs=1, nthread=None, objective='reg:squarederror', random\_state=0,reg\_alpha=0.0001, reg\_lambda=0.01, scale\_pos\_weight=1, seed=None,silent=None, subsample=1, verbosity=0)

xgmod=xgb.fit(x\_train,y\_train)

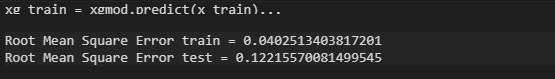
xg\_pred=xgmod.predict(x\_test)

xg\_train = xgmod.predict(x\_train)

print('Root Mean Square Error train = ' + str(math.sqrt(mean\_squared\_error(y\_train, xg\_train))))

print('Root Mean Square Error test = ' + str(math.sqrt(mean\_squared\_error(y\_test, xg\_pred))))

运行结果：



### 集成学习VotingRegressor

# %%

from sklearn.ensemble import VotingRegressor

vote\_mod = VotingRegressor([('Ridge', ridge\_mod), ('Lasso', lasso\_mod), ('Elastic', elastic\_cv),

                            ('XGBRegressor', xgb)])

vote= vote\_mod.fit(x\_train, y\_train.ravel())

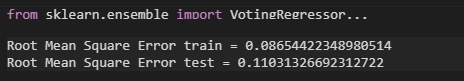
vote\_pred=vote.predict(x\_test)

vote\_train = vote.predict(x\_train)

print('Root Mean Square Error train = ' + str(math.sqrt(mean\_squared\_error(y\_train, vote\_train))))

print('Root Mean Square Error test = ' + str(math.sqrt(mean\_squared\_error(y\_test, vote\_pred))))

运行结果：



可以看到，在集成学习的模型中，在测试集上的MSE要小于其他各个独立模型的MSE。

## 四、生成结果

将结果输出到csv文件中

#%%

vote\_test = vote\_mod.predict(testX)

final=np.expm1(vote\_test)

final\_submission = pd.DataFrame({

        "Id": data\_test["Id"],

        "SalePrice": final

    })

final\_submission.to\_csv("final\_submission.csv", index=False)

final\_submission.head()

