House Price Predict Paper

Cc 2021-04-08

House Price Prediction

**Business origin introduction**: When we have house attribute data and sales price data, if we can relate to each other, this is very important for real estate websites and house buyers.

In addition, this model can help the recommendation system fuction more usefully: Knowing the predicted price is very important for house recommendations for price-sensitive groups.

we have dataset that each row in the data set describes the characteristics of the house. Our goal is to predict the sales price based on characteristics of houses.

The evaluation model is based on the root mean square error (RMSE) between the sales price predicted by the model and the actual sales price. Convert RMSE to a logarithmic scale to ensure that the error in predicting expensive houses and cheap houses has the same score impact. Models: each cross-validation fits many models (including lasso, ridge, svr, ker, ela, bay, etc.) All trained models overfit the training data to varying degrees. Therefore, in order to make the final prediction, I mixed their predictions together to get a more reliable prediction, by using Stacking.

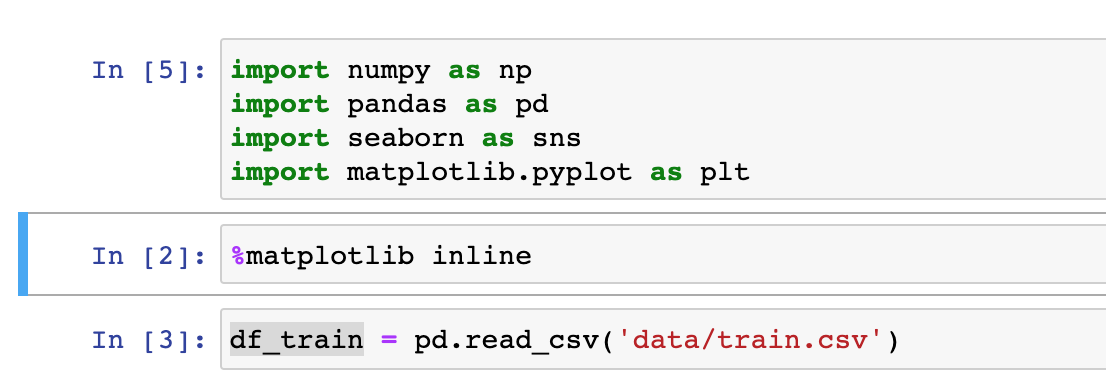
In order to improve RMSE, I use following methods:

Feature Engineering: Mainly assign values to discrete variables, feature combination and PCA

Model fusion: mainly weighted average and stacking

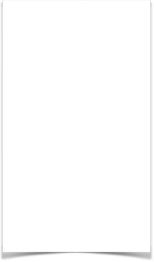
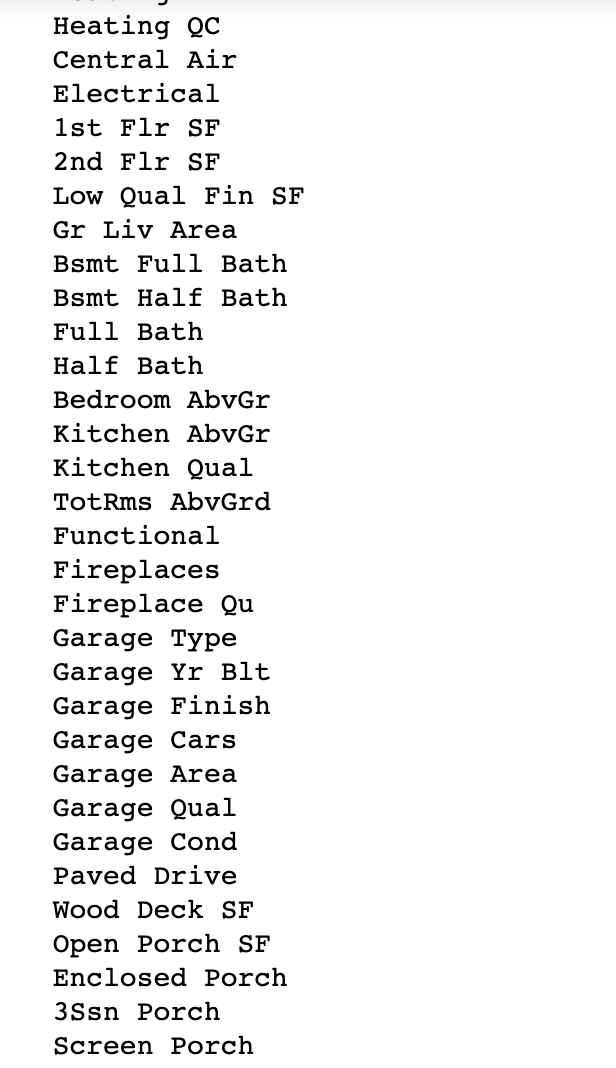
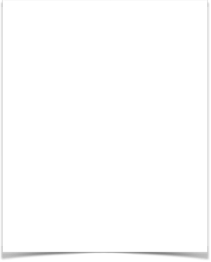
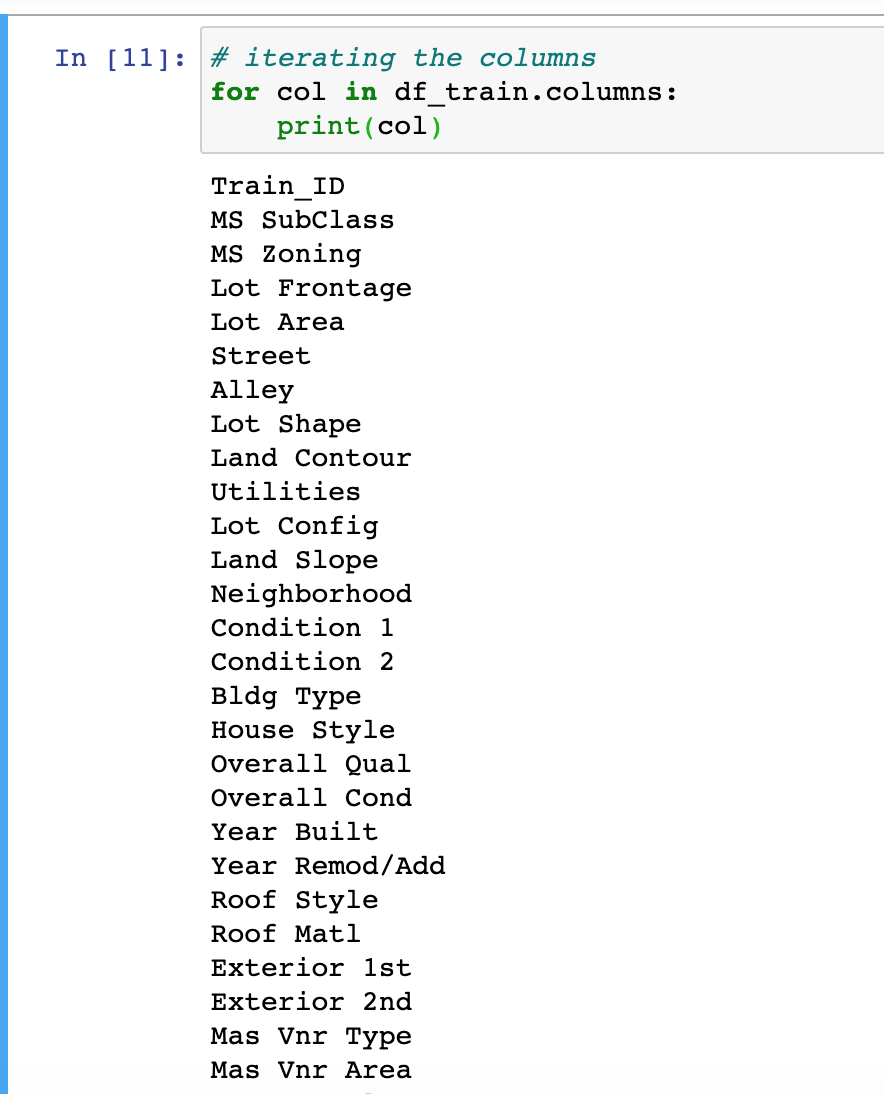
**Import necessary libraries**

We are using numpy,pandas, matplotlib.pyplot, seaborn… this libs to explore What attributes and data are needed for our predict.

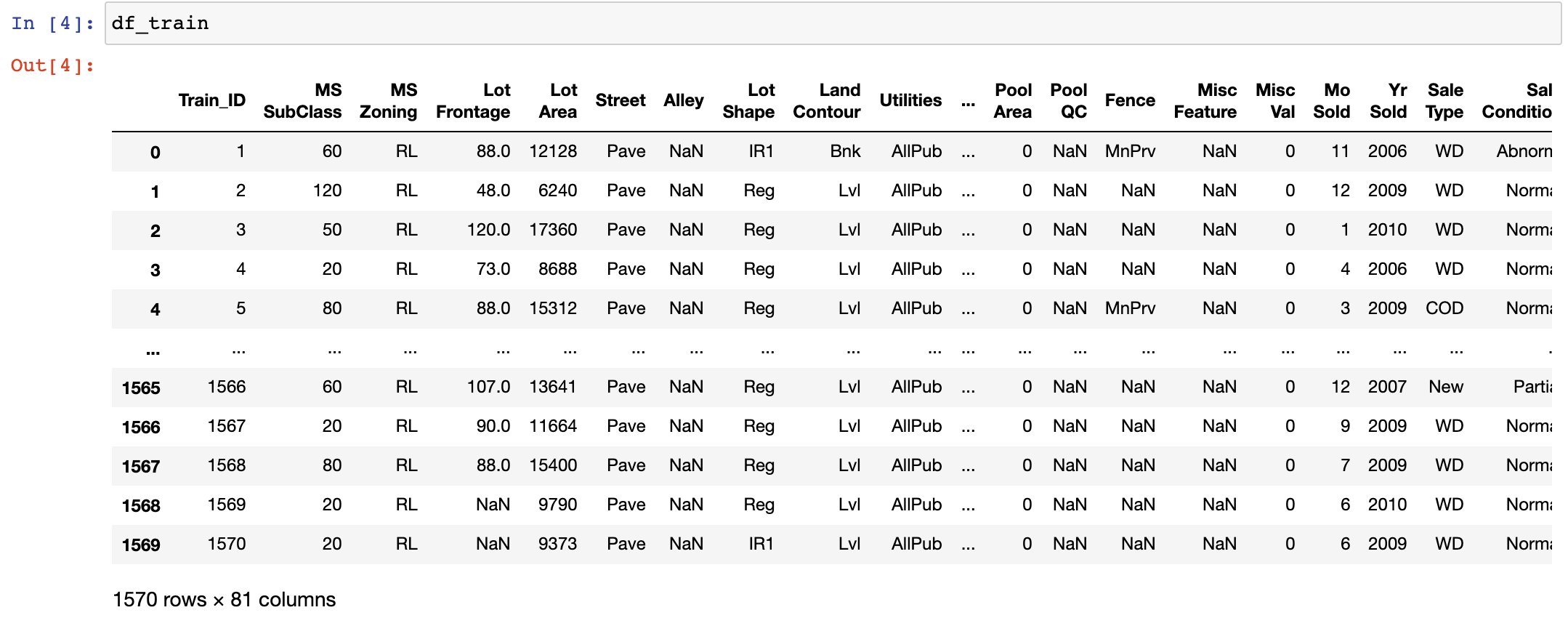


**Import Data**

We explore all column names , read Kaggle\_House\_Data\_Description’s details and explainations.

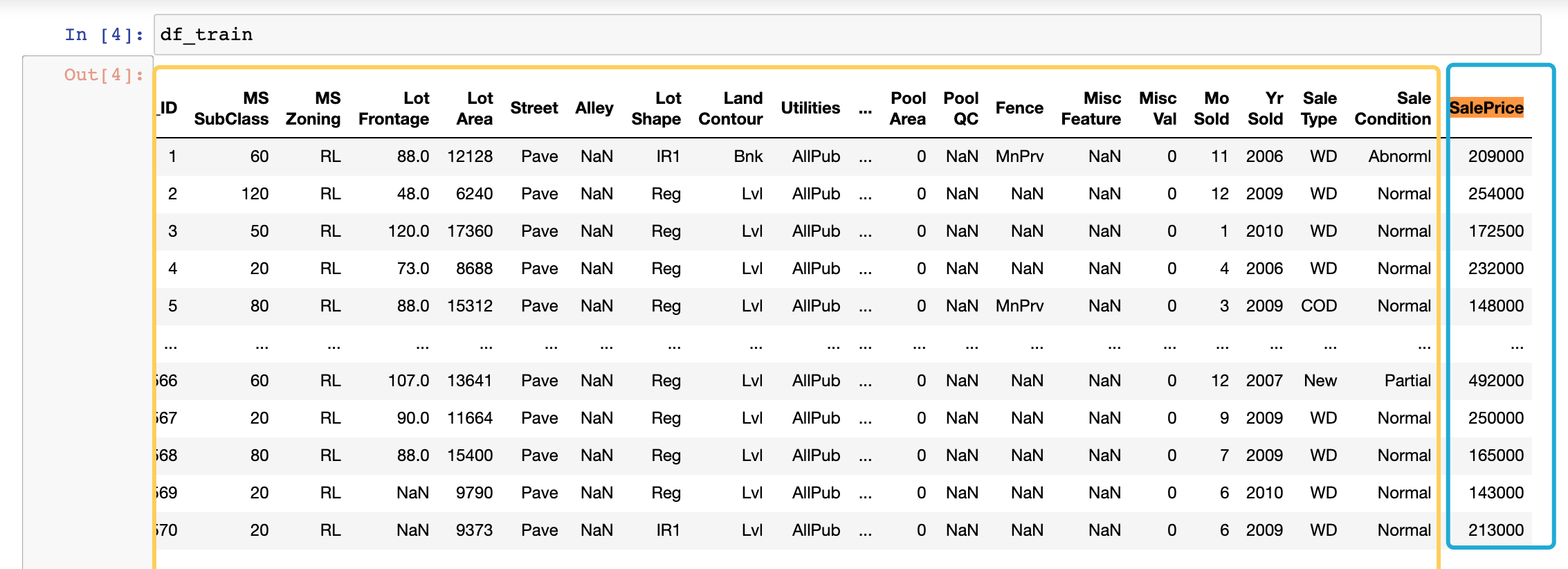


Get all column names



Get data shape and some records

A quick and simple look at the values and column names of the 81 columns, we can know that there are 80 features, including 1 predicted label, and 79 feature labels. The specific classification is as follows



Carefully read all the column names as features and the interpretation of the column data.

For example:

MSSubClass: Identifies the type of dwelling involved in the sale.

MSZoning: Identifies the general zoning classification of the sale.

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Alley: Type of alley access to property

…

Fence: Fence quality

MiscFeature: Miscellaneous feature not covered in other categories

MiscVal: $Value of miscellaneous feature

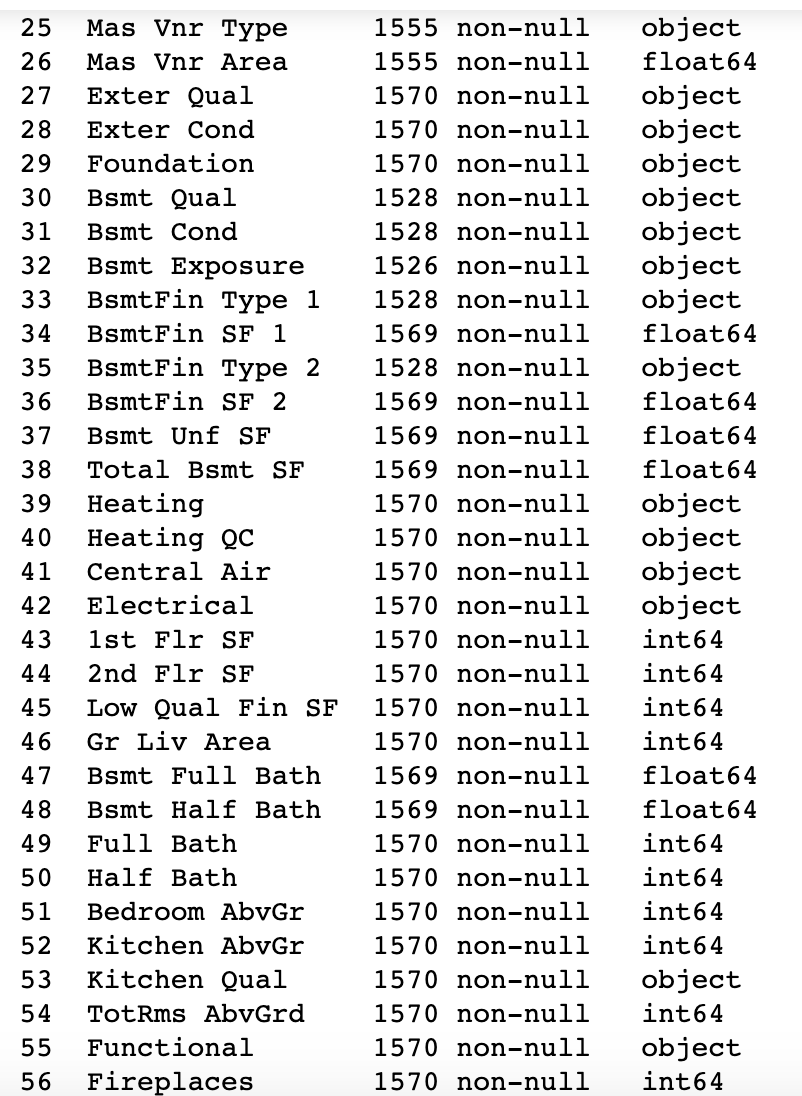
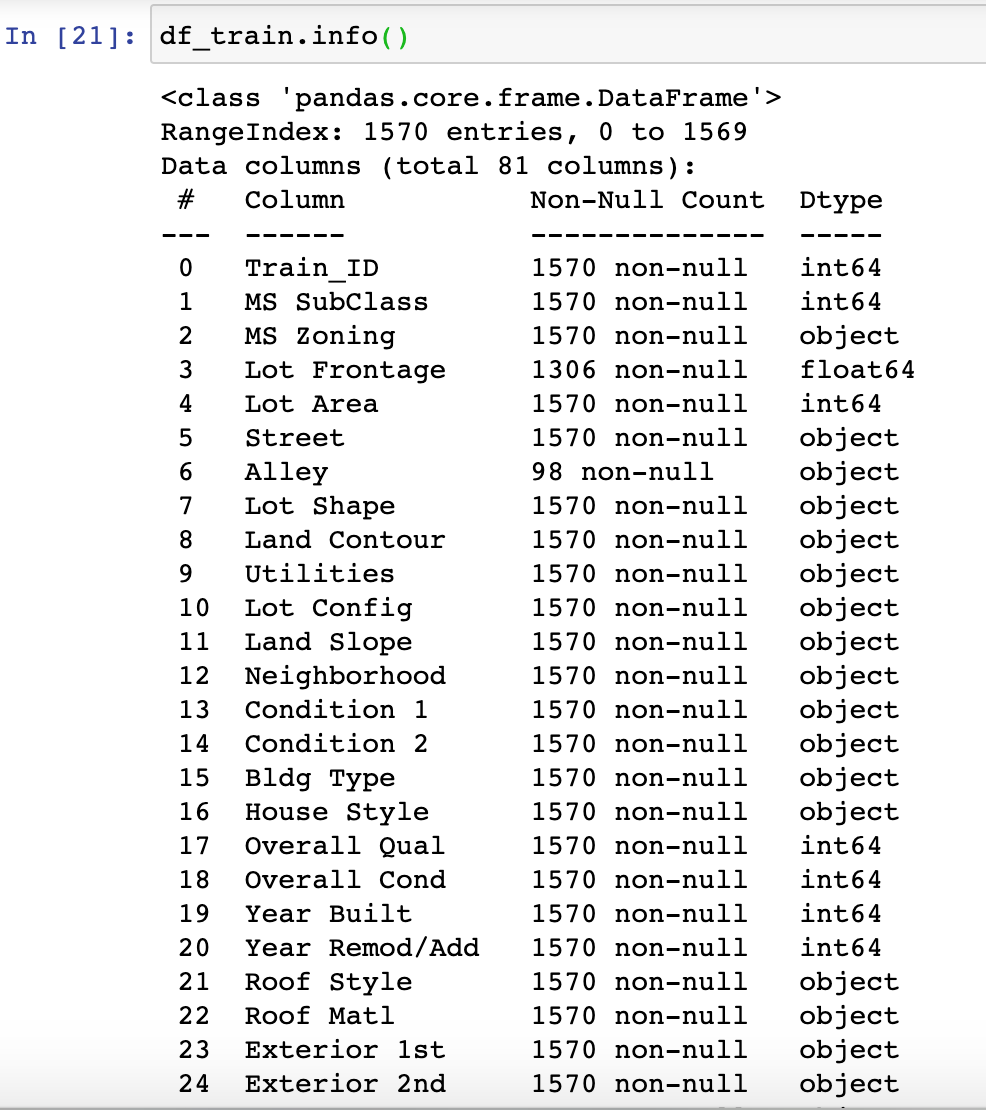
MoSold: Month Sold (MM)

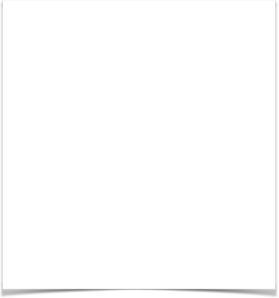
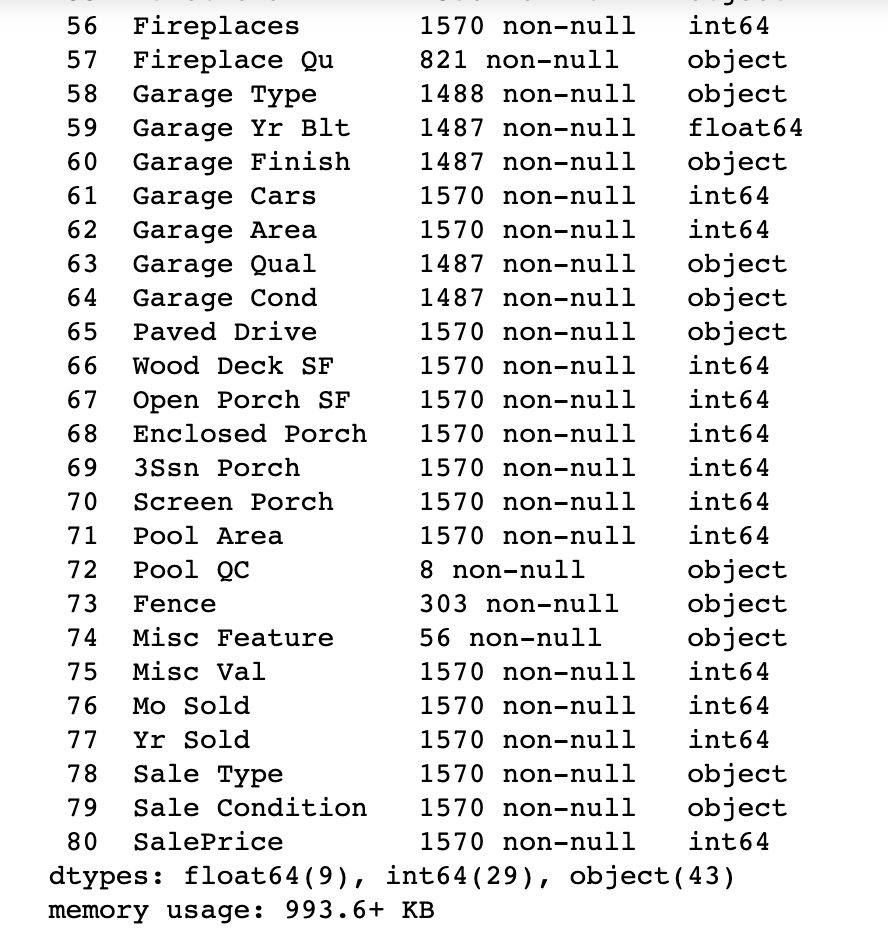
YrSold: Year Sold (YYYY)

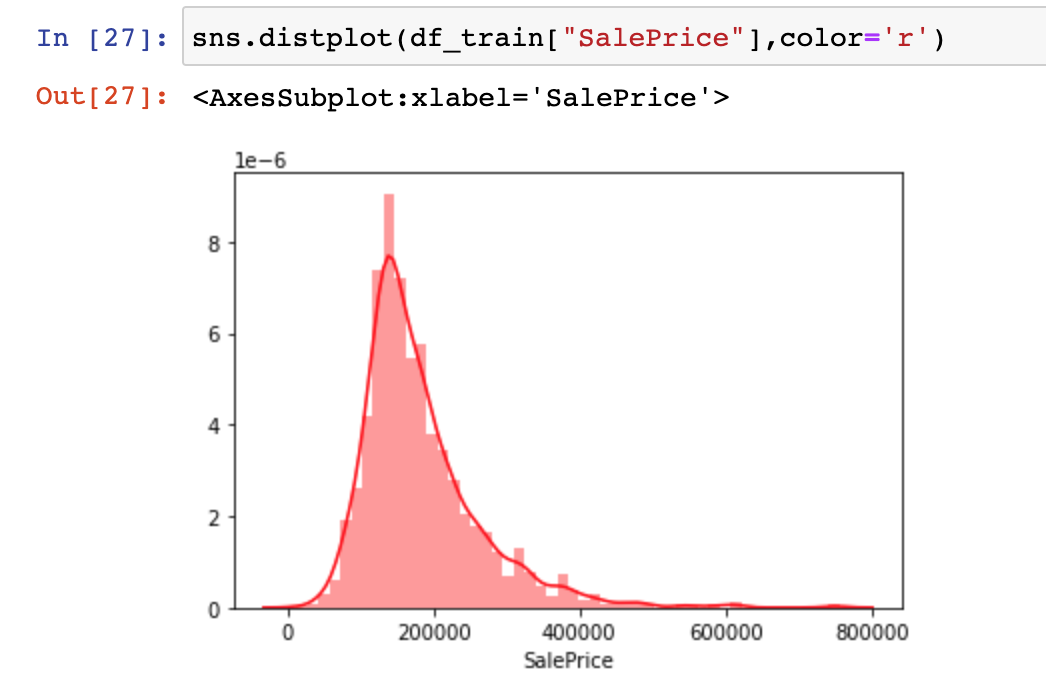
SaleType: Type of sale

SaleCondition: Condition of sale

**Look at the statistics**

View the default data of each data column in the data set, and see which columns have too few data and need to be removed 



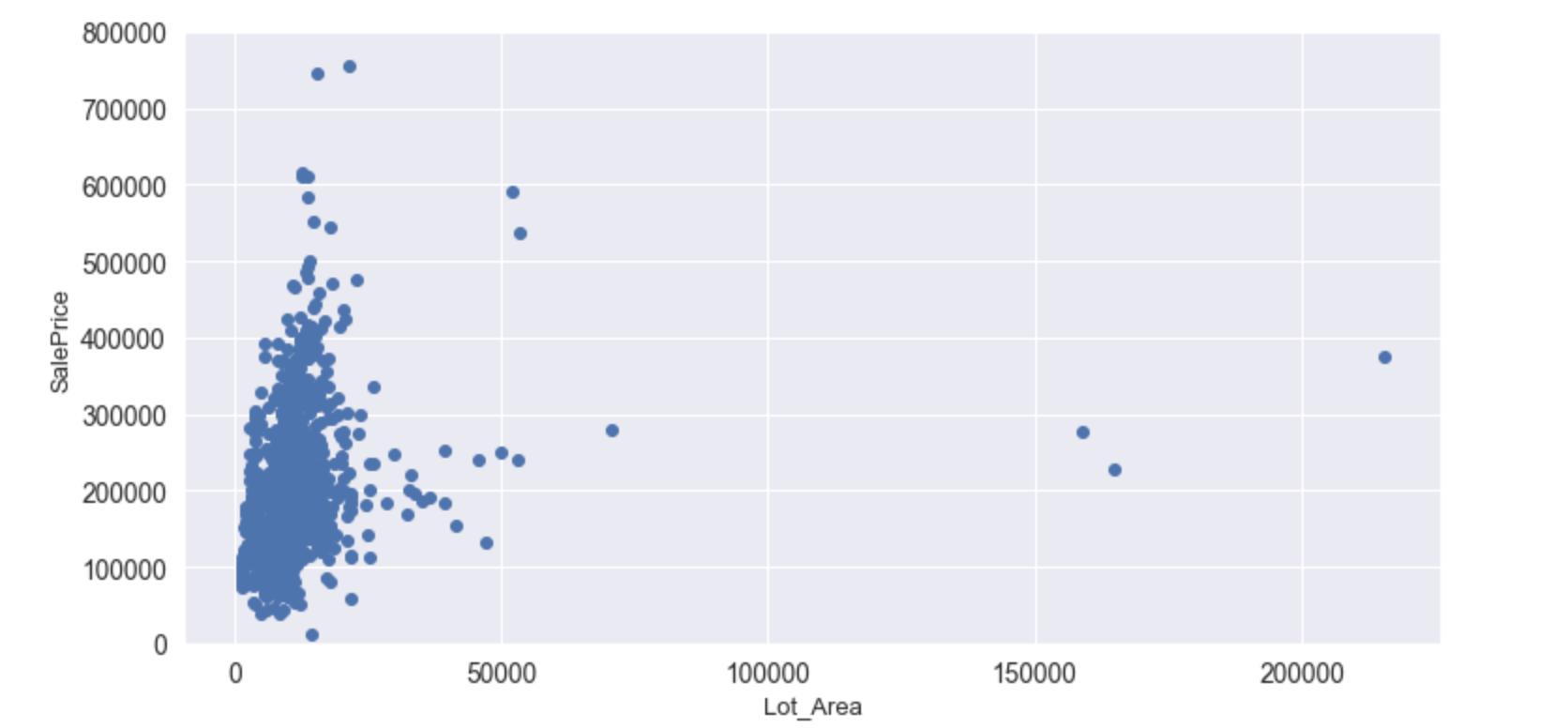
Through visualization, analyze the statistical law of the SalePrice itself

By visualizing the SalesPrice column’s distribution, we found that most of the house prices are within $400,000, and the data distribution is slightly leftward

We need to find the most relevant columns to salesprice, we do test, first find top 20, then top 10

We can see 20 columns ’s relevant hotmap’s too much, so we narroize into 10 columns as following:

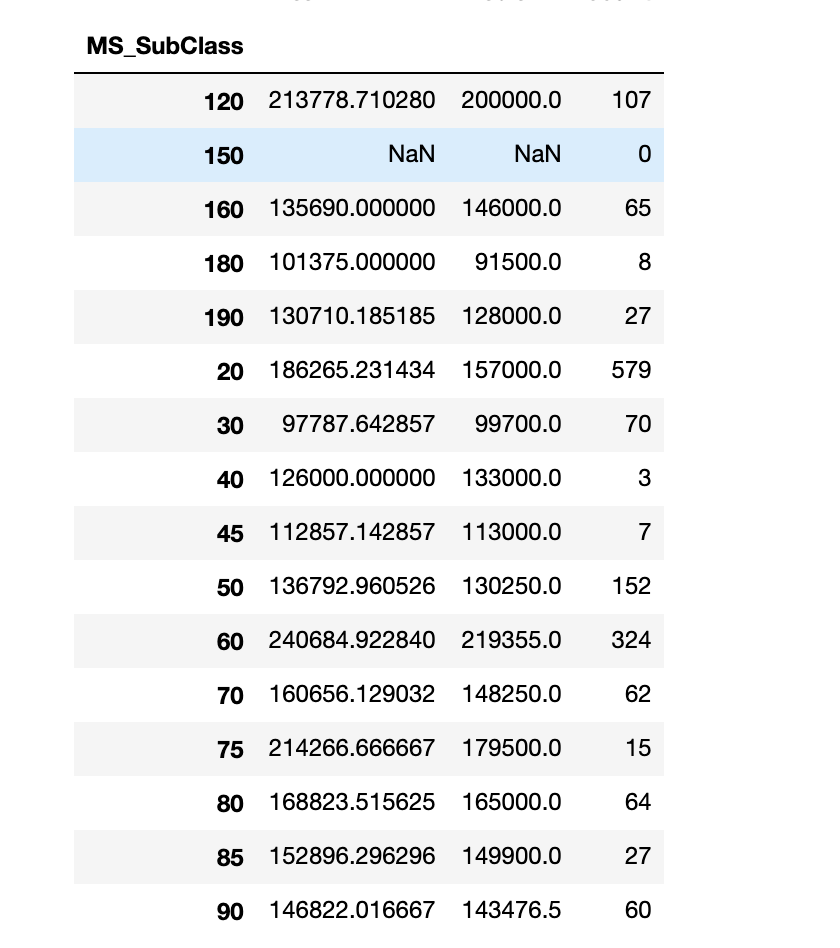


The more relevant the saleprice, the lighter the color, By observation: It can be seen from the figure that Overall Qual, Gr Live Area, Total Bsmt SF, 1st Flr SF, Garage Cars. are all relatively high in correlation coefficients.

When we do visualization of price and columns we want to observe, we can find outlier points : right-bottom-corner points are not wanted. so we use :

train.drop(train[(train[“Lot\_Area"]>50000)&(train["SalePrice"]<400000)].index,inplace=True) to filter outlier data out.

**Feature Engineering**

For discrete features, get\_dummies in pandas is generally used for digitization, but this may not be enough in this Kaggle competition, so the method I use below is to group by feature and calculate the average of SalePrice for each value of the feature And the median, and then sort and assign values based on this, as an example:

Take MSSubClass as example : the feature of MSSubClass represents the type of house, and the data is grouped by it.

So We do the mapping as following:

MS\_SubClass.map({'180':1,

'30':2, '45':2,

'190':3, '50':3, '90':3,

'85':4, '40':4, '160':4,

'70':5, '20':5, '75':5, '80':5, '150':5,

'120': 6, '60':6})

Similar to MS\_SubClass.map, a total of more than 20 features are roughly ranked, as following:

def map\_into\_new\_values():

full["oMSSubClass"] = full.MS\_SubClass.map({'180':1,

'30':2, '45':2,

'190':3, '50':3, '90':3,

'85':4, '40':4, '160':4,

'70':5, '20':5, '75':5, '80':5, '150':5,

'120': 6, '60':6})

full["oMSZoning"] = full.MS\_Zoning.map({'C (all)':1, 'RH':2, 'RM':2, 'RL':3, 'FV':4})

full["oNeighborhood"] = full.Neighborhood.map({'MeadowV':1,

'IDOTRR':2, 'BrDale':2,

'OldTown':3, 'Edwards':3, 'BrkSide':3,

'Sawyer':4, 'Blueste':4, 'SWISU':4, 'NAmes':4,

'NPkVill':5, 'Mitchel':5,

'SawyerW':6, 'Gilbert':6, 'NWAmes':6,

'Blmngtn':7, 'CollgCr':7, 'ClearCr':7, 'Crawfor':7,

'Veenker':8, 'Somerst':8, 'Timber':8,

'StoneBr':9,

'NoRidge':10, 'NridgHt':10})

full["oCondition1"] = full.Condition\_1.map({'Artery':1,

'Feedr':2, 'RRAe':2,

'Norm':3, 'RRAn':3,

'PosN':4, 'RRNe':4,

'PosA':5 ,'RRNn':5})

full["oBldgType"] = full.Bldg\_Type.map({'2fmCon':1, 'Duplex':1, 'Twnhs':1, '1Fam':2, 'TwnhsE':2})

full["oHouseStyle"] = full.House\_Style.map({'1.5Unf':1,

'1.5Fin':2, '2.5Unf':2, 'SFoyer':2,

'1Story':3, 'SLvl':3,

'2Story':4, '2.5Fin':4})

full["oExterior1st"] = full.Exterior\_1st.map({'BrkComm':1,

'AsphShn':2, 'CBlock':2, 'AsbShng':2,

'WdShing':3, 'Wd Sdng':3, 'MetalSd':3, 'Stucco':3, 'HdBoard':3,

'BrkFace':4, 'Plywood':4,

'VinylSd':5,

'CemntBd':6,

'Stone':7, 'ImStucc':7})

full["oMasVnrType"] = full.Mas\_Vnr\_Type.map({'BrkCmn':1, 'None':1, 'BrkFace':2, 'Stone':3})

full["oExterQual"] = full.Exter\_Qual.map({'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

full["oFoundation"] = full.Foundation.map({'Slab':1,

'BrkTil':2, 'CBlock':2, 'Stone':2,

'Wood':3, 'PConc':4})

full["oBsmtQual"] = full.Bsmt\_Qual.map({'Fa':2, 'None':1, 'TA':3, 'Gd':4, 'Ex':5})

full["oBsmtExposure"] = full.Bsmt\_Exposure.map({'None':1, 'No':2, 'Av':3, 'Mn':3, 'Gd':4})

full["oHeating"] = full.Heating.map({'Floor':1, 'Grav':1, 'Wall':2, 'OthW':3, 'GasW':4, 'GasA':5})

full["oHeatingQC"] = full.Heating\_QC.map({'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})

full["oKitchenQual"] = full.Kitchen\_Qual.map({'Fa':1, 'TA':2, 'Gd':3, 'Ex':4})

full["oFunctional"] = full.Functional.map({'Maj2':1, 'Maj1':2, 'Min1':2, 'Min2':2, 'Mod':2, 'Sev':2, 'Typ':3})

full["oFireplaceQu"] = full.Fireplace\_Qu.map({'None':1, 'Po':1, 'Fa':2, 'TA':3, 'Gd':4, 'Ex':5})

full["oGarageType"] = full.Garage\_Type.map({'CarPort':1, 'None':1,

'Detchd':2,

'2Types':3, 'Basment':3,

'Attchd':4, 'BuiltIn':5})

full["oGarageFinish"] = full.Garage\_Finish.map({'None':1, 'Unf':2, 'RFn':3, 'Fin':4})

full["oPavedDrive"] = full.Paved\_Drive.map({'N':1, 'P':2, 'Y':3})

full["oSaleType"] = full.Sale\_Type.map({'COD':1, 'ConLD':1, 'ConLI':1, 'ConLw':1, 'Oth':1, 'WD':1,

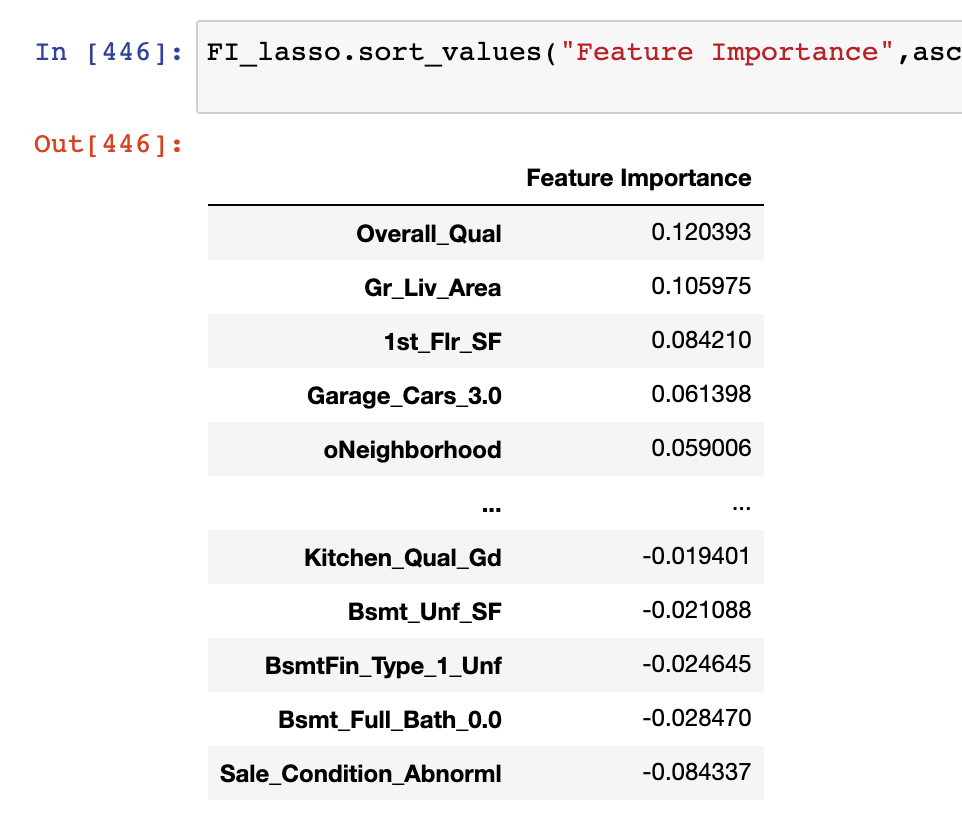
'CWD':2, 'Con':3, 'New':3})

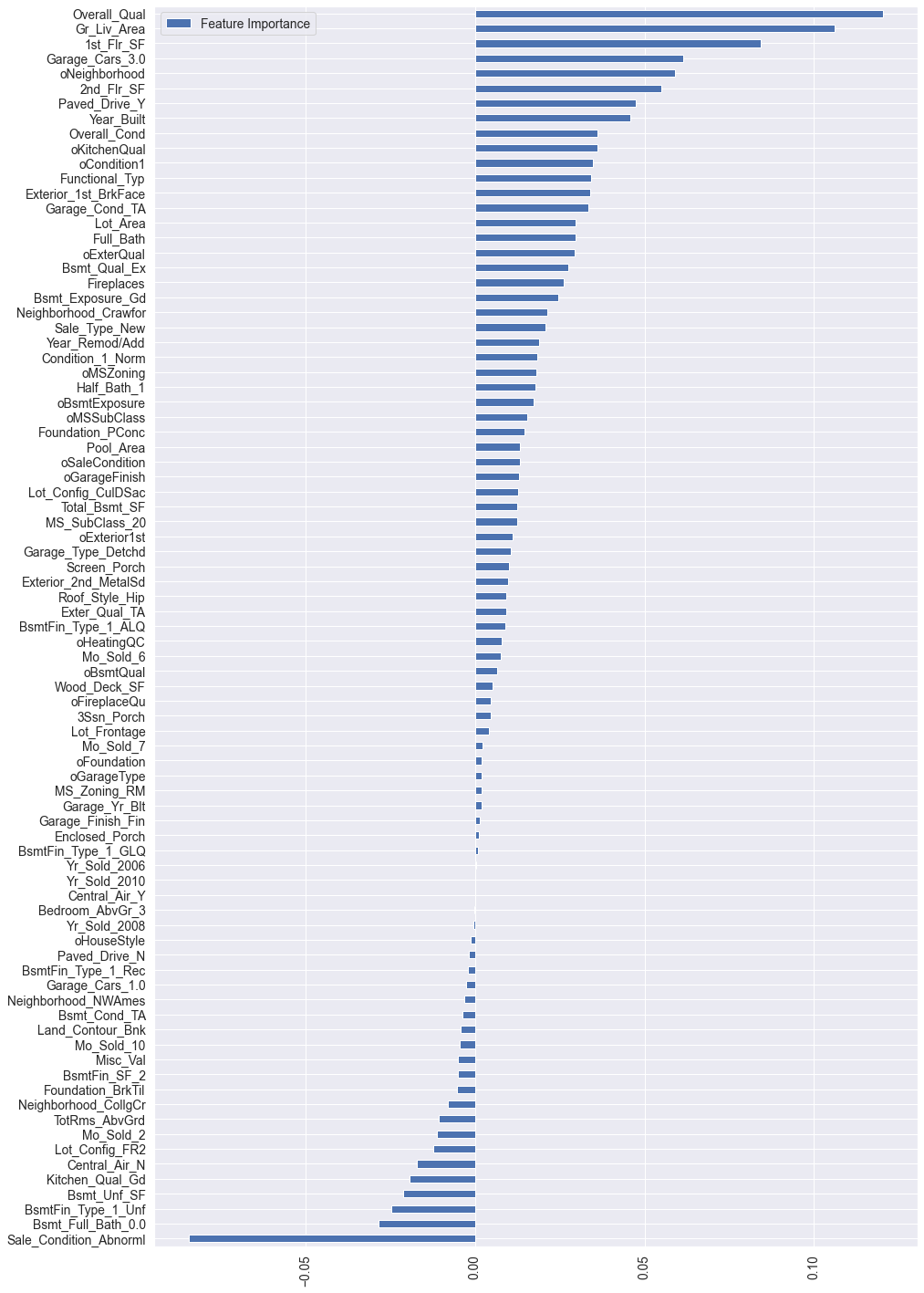
full["oSaleCondition"] = full.Sale\_Condition.map({'AdjLand':1, 'Abnorml':2, 'Alloca':2, 'Family':2, 'Normal':3, 'Partial':4})

I have converted the values of dozens of columns in total, in order to facilitate classification, you can refer to the code for details

**Feature combination**

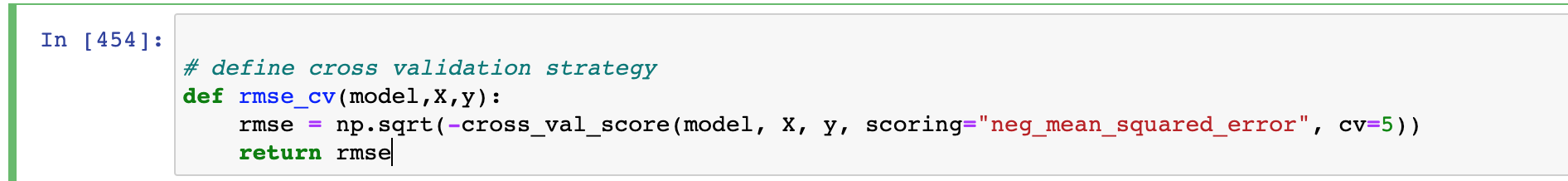
Combining the original features can usually produce unexpected results. However, there are many original features in this data set, and it is impossible to combine all of them one by one, so here we use Lasso for feature screening first, and select some of the more important features for combination





Finally, these features were added, which also included many other attempts. For details, please refer to the functions and classes related to add\_feature

PCA is a very important part, which greatly improves the final score. Because these new features I added are highly correlated with the original features, this may lead to strong multicollinearity (Multicollinearity), and PCA can just decorate the correlation. Because the purpose of using PCA here is not to reduce dimensionality, n\_components uses dimensions that are similar to the original. This is the result of my multi-party experiment, that is, add XX features in the front, and then reduce to XX dimensions.

First define the cross-validation evaluation index of RMSE

**model selection**：12 algorithms and 5-fold cross-validation are used to evaluate the baseline effect:

**LinearRegression**

**Ridge**

**Lasso**

**Random Forrest**

**Gradient Boosting Tree**

**Support Vector Regression**

**Linear Support Vector Regression**

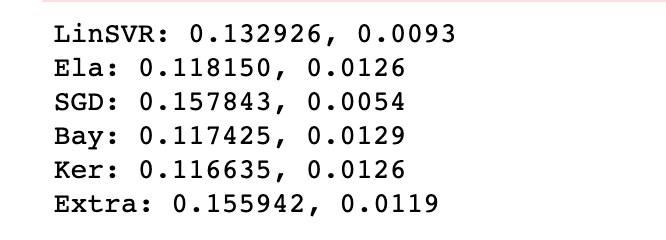
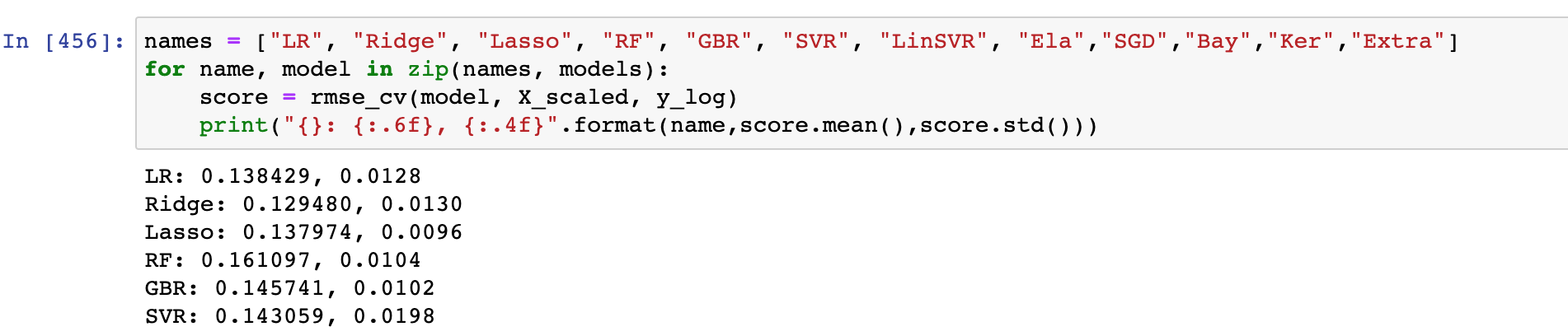
**ElasticNet**

**Stochastic Gradient Descent**

**BayesianRidge**

**KernelRidge**

**ExtraTreesRegressor**



Except the test results , **my rationale thoughts**: the tree model is generally inferior to the linear model, maybe because of the data sparsity brought by get\_dummies / or the dataset is not large enough for the the tree model to have the best .

However, these models parameters have not been adjusted. Before making a choice, we should establish a method for adjusting parameters. We should always keep in mind that the evaluation indicator is RMSE.

method to adjust parameter , for all model to use:

grid\_search = GridSearchCV(self.model,param\_grid,cv=5, scoring="neg\_mean\_squared\_error")

grid\_search.fit(X,y)

print(grid\_search.best\_params\_, np.sqrt(-grid\_search.best\_score\_))

grid\_search.cv\_results\_['mean\_test\_score'] = np.sqrt(-grid\_search.cv\_results\_['mean\_test\_score'])

print(pd.DataFrame(grid\_search.cv\_results\_)[['params','mean\_test\_score','std\_test\_score']])

SVR

grid(SVR()).grid\_get(X\_scaled,y\_log,{'C':[11,13,15],'kernel':["rbf"],"gamma":[0.0003,0.0004],"epsilon":[0.008,0.009]})

results stablized at:

std\_test\_score

0 0.001553

1 0.001609

2 0.001555

….

11 0.001617

Kernel

param\_grid={'alpha':[0.2,0.3,0.4], 'kernel':["polynomial"], 'degree':[3],'coef0':[0.8,1]}

grid(KernelRidge()).grid\_get(X\_scaled,y\_log,param\_grid)

results stablized at:

std\_test\_score

0 0.001209

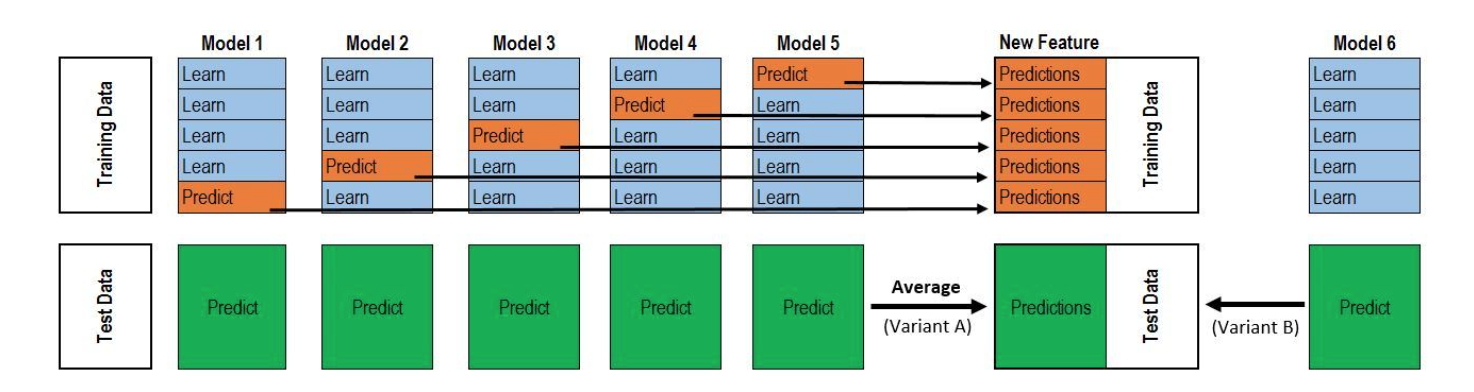
1 0.001243

2 0.001189

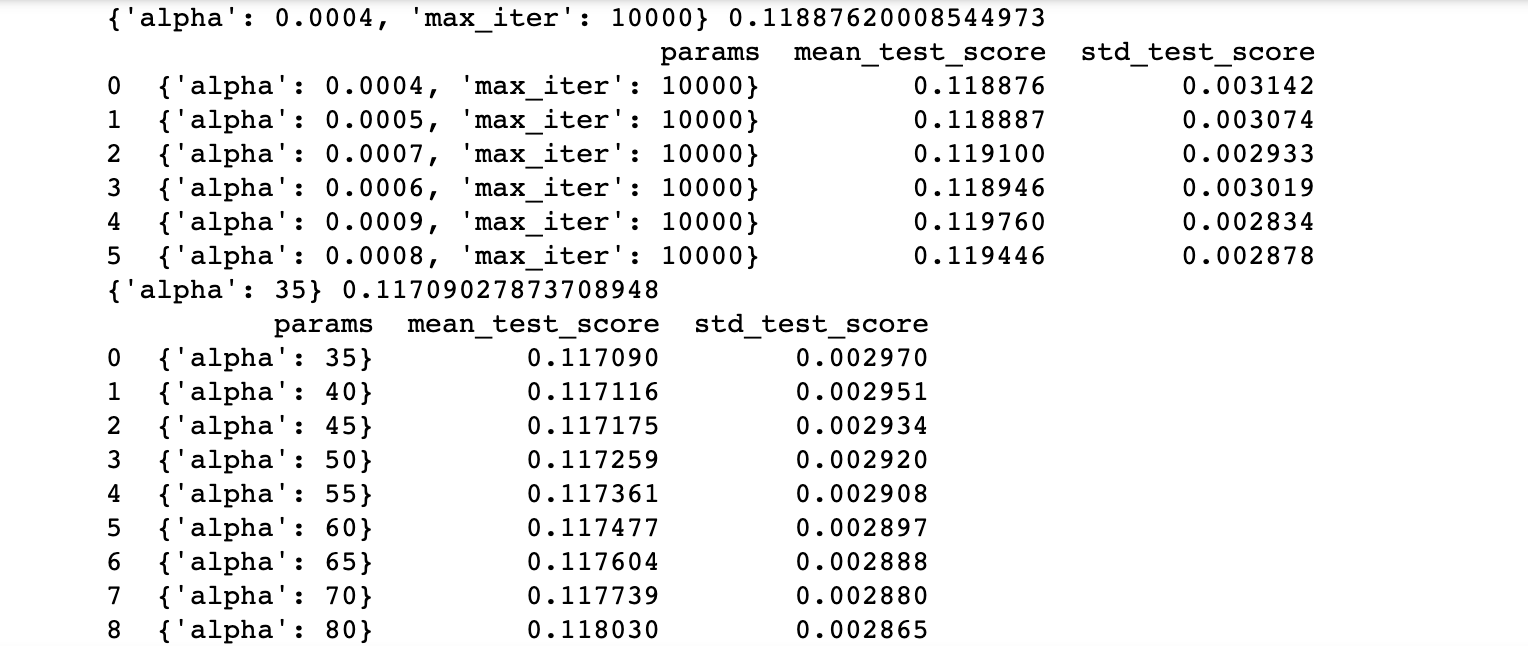
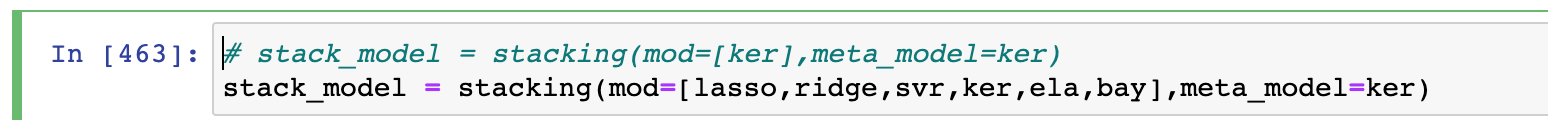
3 0.001210

4 0.001181

5 0.001191   
 After many long rounds of testing among all the models, **SVR, KER** are my best 2 models. If I use single model, I will choose from SVR and Kernel Ridge two models.

But I want to combine Weighted average and stacking all 5 models.

In the two-layer stacking as shown in the figure, there are 5 models in the first layer and 1 meta-model in the second layer. The function of the first layer model is to train to obtain a [formula] feature matrix for input to the second layer model training, where n is the number of training data rows and m is the number of the first layer model.

After many long rounds of testing, these six models were finally selected

**Final analysis, conclusion, limitations**

It is important to understand the data, and it is important to clean and transform the data. There are both discrete and continuous features, and there are a lot of missing values. Fortunately, the contestant provided the file data\_description.txt, which describes the meaning of each feature. After understanding the content, most of the missing values can be smoothly interpolated.

If I have more time and more data, I want to try the neural network method. In the training process, as long as there are enough input x and output y, a better neural network model can be trained. This model is In similar housing price prediction problems, more accurate results can be obtained.