

Data Analytics report of House Price

Siheng Huang

I Data source and dataset Introduction

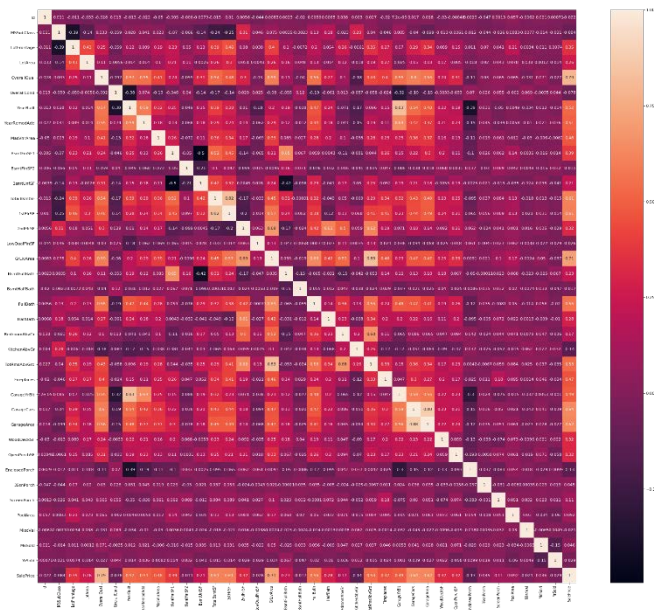
As a home buyer want to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But the analytics of this dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

This dataset contains 79 features, and the types of them are numeric, character-type discrete values. With analyzing the dataset, we can predict the price of house which only has some information.

II Exploration, Statistics, and Visualization

a. about the numeric values

Considering to draw their heat map



```
corrmat = Data_num.corr()  
plt.subplots(figsize=(30,30))  
sns.heatmap(corrmat,annot=True)  
plt.show()
```

By analyzing the color of heat map, we can find out the relationship with the dependent variable (House Price)

According to the heat map, we can find some variables that are related to each other, such as OverallQual and SalePrice have strong relation with the corr between them is 0.795 and TotRmsAbvGrd and GlivArea with the corr is 0.83, which means that we can predict the SalePrice of some house with signal feature owing to their strong relationship with HousePrice, and we can also make combination of some certain features because the combination of them may have strong relationship with our dependent variable.

b. about the character-type discrete values

The charts are in [appendix](#)

From the 1st chart, before 1990 the SalePrice do not change much over time by YearBuild and YearRemoveAdd, but after 1990 the influence increases tremendous. From 2nd chart to final charts show the SalePrice is affected by serious discrete variable, especially assessment type variable, and some charts show the effect is not linear.

III Data Cleaning

a. Firstly, find features which have missing data

```
X=pd.concat([Train,Test],axis=0)  
  
X1=pd.DataFrame(X.isnull().sum()[X.isnull().any()],columns=["values"])  
X2=pd.DataFrame(X[X1.index].dtypes,columns=["types"])  
lost_values2=pd.concat([X1,X2],axis=1).sort_values(by="values",ascending=False)  
lost_values2
```

the consequence is:

	values	types			
PoolQC	2909	object	MasVnrType	24	object
MiscFeature	2814	object	MasVnrArea	23	float64
Alley	2721	object	MSZoning	4	object
Fence	2348	object	Utilities	2	object
SalePrice	1459	float64	Functional	2	object
FireplaceQu	1420	object	BsmtHalfBath	2	float64
LotFrontage	486	float64	BsmtFullBath	2	float64
GarageFinish	159	object	GarageCars	1	float64
GarageQual	159	object	Exterior2nd	1	object
GarageYrBlt	159	float64	KitchenQual	1	object
GarageCond	159	object	Exterior1st	1	object
GarageType	157	object	Electrical	1	object
BsmtCond	82	object	BsmtUnfSF	1	float64
BsmtExposure	82	object	BsmtFinSF2	1	float64
BsmtQual	81	object	BsmtFinSF1	1	float64
BsmtFinType2	80	object	SaleType	1	object
BsmtFinType1	79	object	TotalBsmtSF	1	float64
			GarageArea	1	float64

By analyzing all the data of the dataset, we can find some rules to fill the missing data:

- (1) To some numeric data, we can use 0 to fill them, for the reason is that the meaning of missing data of some features is that its value is empty, these features are as follows: LotFrontage, MasVnrArea, BsmtFullBath, BsmtHalfBath, GarageCars, BsmtFinSF1 and BsmtFinSF2;
- (2) To some data, we can use their mode to fill them, for the reason is that the meaning of missing data of some features is that its value is not empty, and it should be existing, for example, date. These features are as follows: PoolQC, BsmtQual, BsmtCond, FireplaceQu, GarageFinish, GarageQual, BsmtExposure, Electrical, MSZoning, Exterior1st, Exterior2nd, KitchenQual and SaleType.
- (3) And about other data, we cannot know about the reasons for them to have missing data. So, we use “missing” to fill them.

```
type_2=["PoolQC","BsmtQual", "BsmtCond", "FireplaceQu", "GarageFinish","GarageQual","BsmtExposure",
        "Electrical", "MSZoning", "Exterior1st", "Exterior2nd", "KitchenQual","SaleType"]
type_3=['Alley','MasVnrType','GarageType','GarageCond','Fence','Street','LotShape','LandContour','BsmtFinType1',
        'BsmtFinType2','CentralAir','MiscFeature','Utilities',"Functional"]

def fill_missings(res):
    for type2 in type_2:
        res[type2] = res[type2].fillna(res[type2].mode()[0])
    for type3 in type_3:
        res[type3] = res[type3].fillna("missing")

    flist = ['LotFrontage','LotArea','MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF',
            'TotalBsmtSF','1stFlrSF','2ndFlrSF','LowQualFinSF','GrLivArea', 'BsmtFullBath',
            'BsmtHalfBath','FullBath','HalfBath','BedroomAbvGr', 'KitchenAbvGr',
            'TotRmsAbvGrd', 'Fireplaces','GarageCars','GarageArea', 'WoodDeckSF',
            'OpenPorchSF', 'EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea','MiscVal']

    for fl in flist:
        res[fl] = res[fl].fillna(0)
    #using 0 to replace
    res['TotalBsmtSF'] = res['TotalBsmtSF'].apply(lambda x: np.exp(6) if x <= 0.0 else x)
    res['2ndFlrSF'] = res['2ndFlrSF'].apply(lambda x: np.exp(6.5) if x <= 0.0 else x)
    res['GarageArea'] = res['GarageArea'].apply(lambda x: np.exp(6) if x <= 0.0 else x)
    res['GarageCars'] = res['GarageCars'].apply(lambda x: 0 if x <= 0.0 else x)
    res['LotFrontage'] = res['LotFrontage'].apply(lambda x: np.exp(4.2) if x <= 0.0 else x)
    res['MasVnrArea'] = res['MasVnrArea'].apply(lambda x: np.exp(4) if x <= 0.0 else x)
    res['BsmtFinSF1'] = res['BsmtFinSF1'].apply(lambda x: np.exp(6.5) if x <= 0.0 else x)

    return res
```

b. Then, clean up outliers

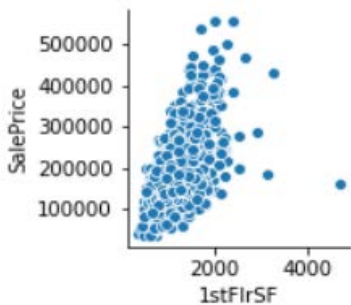
(1) Delete the SalePrice data which is beyond $[data.mean-5*data.std, data.mean+5*data.std]$

```
Data1=X[X.SalePrice <= X["SalePrice"].mean()+5*X["SalePrice"].std()]
```

(2) Use rational graph

Using the scatter diagram to Elimination of outliers

```
for column in num:
    figure=plt.figure()
    sns.pairplot(x_vars=[column],y_vars=['SalePrice'],data=Data1,dropna=True)
plt.show()
```



Taking “1stFlrSf” as example, we can find that there are a spot diverge obviously, so we have to delete this data to reduce the variance .

```
value = 1
Data1=Data1.sort_values(ascending=False)[value:]
```

And finally, I select 16 features to do this process.

IV Feature Engineering

a. Add new features

After understanding every feature, I choose to add some new features to the dataset

```
Data["TotalFlrSF"] = Data["1stFlrSF"]+Data["2ndFlrSF"]
Data["TotalPorch"] = Data["3SsnPorch"]+Data["EnclosedPorch"]+Data["OpenPorchSF"]
                    +Data["ScreenPorch"]
Data["TotalBath"] = Data["HalfBath"]+Data["FullBath"]
Data['YearsSinceRemodel'] = Data['YrSold'].astype(int) - Data['YearRemodAdd'].astype(int)
```

b. Transform some features' type to series

Some data which is object originally, they can be quantifiably represent. So I use the numeric value to represent its degree, for example, ExterQual, it has 6 kinds of values—“Ex”, “Gd”, “TA”, “Fa”, “Po” and “missing”, and each values means different degree, so we use 0 to 6 represent them.

```
def QualToInt(data):
    if (data == "Ex"):
        score=5
    elif (data == "Gd"):
        score=4
    elif (data == "TA"):
        score=3
    elif (data == "Fa"):
```

```

        score=2
    elif (data == "Po"):
        score=1
    else:
        score=0
    return score

```

c. standardization and Logarithmic

To make the calculated amount reduce and make the model fit better later, we Standardize the data.

```

for column in Data_.columns:
    if Data_[column].dtypes != "object":
        Data_[column] = (Data_[column]-Data_[column].mean())/Data_[column].std()

```

And then, after standardization, the value will be between 0 to 1, then I use Logarithmic to deal with the data after standardization, and it can magnify the absolute value of the number between 0 and 1, and compress the number greater than 1.

```

def addlogs(res, ls):
    m = res.shape[1]
    for l in ls:
        res = res.assign(newcol=pd.Series(np.log(1.01+res[l])).values)
        res.columns.values[m] = l + '_log'
        m += 1
    return res
loglist=skewness[abs(skewness)>0.15].index.tolist()
Data_ = addlogs(Data_, loglist)

```

d. One-Hot

For many machine learning model cannot deal with discrete variable, so using the method get_dummies can make object variable transform to numeric variable.

```

Data_pre=pd.get_dummies(Data_,columns=Data_.select_dtypes(include=["object"]).columns)

```

e. Reduce Dimension--PCA

After the last step, the dataset's features increase to 313, so we have to use PCA to reduce some features and at the meanwhile, the information of the remaining feature should contains most of the information.

```

from sklearn.decomposition import PCA
x = PCA(n_components=0.975,svd_solver="full").fit_transform(Data)

```

And finally, the number of features is 83, far less than the formal number—313.

V Model Building

According to the dataset, I choose 10 models to fit the data.

At the beginning, spilt the data.

```

Xtrain,Xtest,Ytrain,Ytest = train_test_split(X_1,y,test_size=0.15,random_state=666)

```

I Signal Model

a. Linear Regression

```
reg = LinearRegression().fit(Xtrain,Ytrain)
```

And the score of it is **0.8905497**, the rmse is **22170.46107**.

b. Lasso

While using Lasso, we first have to decide its alpha, and we can use learning curve.

```
alp = np.logspace(-10,0,200,base=10)
lasso_ = LassoCV(alphas=alp,cv=5).fit(Xtrain,Ytrain)
lasso_.alpha_
```

After that, we know that the alpha should be **1.0** to make the score best.

And finally the score of it is **0.8905497**, the rmse is **22170.46107**.

c. ElasticNet

```
enet_ = ElasticNetCV(alphas=alp,cv=5).fit(Xtrain,Ytrain)
yhat = enet_.predict(Xtest)
print(enet_.alpha_, enet_.score(Xtest,Ytest), mean_squared_error(yhat,Ytest)**0.5)
```

After that, we know that the alpha should be **1e-10** to make the score best.

And finally the score of it is **0.84934**, the rmse is **22172.34383**.

d. RandomForestClassifier

```
rfc=RandomForestClassifier(random_state=666).fit(Xtrain,Ytrain)
rfc.score(Xtest,Ytest)
```

Its score is **0.013953**, so we may not consider this model

e. XGboost

```
n_estimators=range(50,500,50)
score=[]

for i in n_estimators:
    xg = XGBR(n_estimators=i).fit(Xtrain,Ytrain)
    score.append(xg.score(Xtest,Ytest))

n_estimators[score.index(max(score))]
```

And we can find that the best n_estimators is 350, the score of it is **0.91299**, the rmse is **20125.796783**

f. RandomForestRegressor

To find the best n_estimators, using the same way as XGboost

```
regressor = RandomForestRegressor(n_estimators=400).fit(Xtrain,Ytrain)
```

And we can find that the best n_estimators is 400, the score of it is **0.914076**, the rmse is **20194.34745**

g. AdaBoostRegressor

After the learning curve, we could find the best parameters are: n_estimators=300, learning_rate=0.7934096665797492, loss="square"

And the score of it is **0.860575762**, the rmse is **25724.2263**

h. BayesianRidge

the score of it is **0.89788**, the rmse is **22015.060348**

i. GradientBoostingRegressor

After the learning curve, we could find the best parameters are: loss="huber", n_estimators=350

the score of it is **0.920411**, the rmse is **19427.81368**

j. MLPRegressor

the score of it is 0.920411, the rmse is 184661.3173,

II Stacking


Above all of the models, I will use stacking to combine these models, and the first floor is LinearRegression, Lasso, ElasticNet, RandomForestRegressor, AdaboostRegressor, BayesianRidge and GradientBoostingRegressor, MLPRegressor; the second floor is XGboost MLPRegressor.

```
rgc = LinearRegression()
la = Lasso(alpha = 1.0)
ela = ElasticNet(alpha = 1e-10)
rfr = RandomForestRegressor(n_estimators=400)
ada = AdaBoostRegressor(n_estimators=50, learning_rate=0.560716993820547, loss="linear")
bay = BayesianRidge()
gbr = GradientBoostingRegressor(loss="huber", n_estimators=350)
mlpr = MLPRegressor()
xg = XGBR(n_estimators=350)

stacked = StackingCVRegressor(regressors=(rgc, la, ela, rfr, ada, bay, mlpr, gbr), meta_regressor=xg,
                              use_features_in_secondary=True)
```

VI Evaluating and Result

a. Result

1461	Alphonse		0.12978	4	~10s
------	----------	--	---------	---	------

b. Evaluating

I think the error mainly comes from the process of filling missing value, feature engineering and PCA.

(1) While filling missing value, I use "0" and mode to fill them, however this way will take the inaccuracy to the model building and finally make the error great.

(2) And about the feature engineering, while using numeric value to represent the degrees of some discrete variables, it cannot easily replace the variable with the continuous number because we cannot do quantification accurate, for example, in this case, we use "5" to represent "Excellent", use "4" to represent "Good", use "3" to represent "Average", but it may have some incorrect for the reason that the gap between "Excellent" and "Good", and between "Good" and "Average" may not be the same, but if we use 5 4 and 3 to represent them, it will mean that we default they are in the same.

(3) After the process PCA, the new features will just contains 97.5% information of the formal, this will also make the error great.

APPENDIX

I exploration graph

Uploaded to Tableau (need VPN to look)

https://us-west-2b.online.tableau.com/#/site/sihenghuang/workbooks/191294?origin=card_share_link

II code

Uploaded to GitHub (need VPN to look)

https://github.com/Alphonse-HUANG/HOUSE_PRICE-report/tree/master