Algorithm: Gradient Boosting Machines (GBM)

1.

With all the hype about deep learning and "AI", it is not well publicized that for structured/tabular data widely encountered in business applications it is actually another machine learning algorithm, the gradient boosting machine (GBM) that most often achieves the highest accuracy in supervised learning tasks.

We intend to involve Light GBM and Gradient Boosting Regressor in this case.

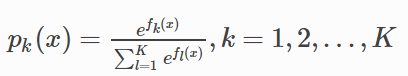
Light GBM is a fast, distributed, high-performance gradient [boosting](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=which-algorithm-takes-the-crown-light-gbm-vs-xgboost) framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks. Light GBM is becoming more and more popular because of its faster train speed and higher efficiency.

Gradient Boosting is a typical decision tree machine learning technique, which usually solves regression and classification issues. It performs greatly in prediction models.

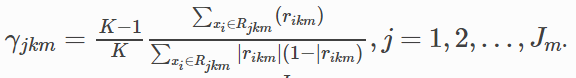
2.

Friedman (2001) proposed a Gradient Boosting algorithm to solve the minimization problem above, which works well with a variety of different loss functions

For m=1 to M:

1. Set 

2. For k=1 to K:

1. Compute 
2. it a regression tree to the targets rikm, i=1,2,…,N, giving terminal regions Rjim,j=1,2,…,Jm
3. Compute 
4. Update 

Output 

3.

Our dataset can be directly downloaded from <https://www.kaggle.com/c/house-prices-advanced-regression-techniques>. It contains more than 70 independent variables, which includes street, alley, land slope and so on, to predict sale price of houses.

Gradient Boosting usually helps to deal with classification and regression problems, and data that requires classification or prediction can accordingly adapt this algorithm. Light GBM is a special algorithm contained among Gradient Boosting.

4.

Even though Gradient Boosting Machines is powerful, there are times when it works less effective than other algorithms.

1. GBMs will continue improving to minimize all errors. This can overemphasize outliers and cause overfitting. Must use cross-validation to neutralize.
2. Computationally expensive - GBMs often require many trees (>1000) which can be time and memory exhaustive.
3. The high flexibility results in many parameters that interact and influence heavily the behavior of the approach (number of iterations, tree depth, regularization parameters, etc.).
4. Less interpretable although this is easily addressed with various tools (variable importance, partial dependence plots, LIME, etc).

5.

To run the algorithm, we adapted the following Python library:

(1) sklearn:

* In python, sklearn is a machine learning package which include a lot of machine learning algorithms.
* We are using some of its modules like train\_test\_split, DecisionTreeClassifier and DecisionTreeRegressor.

(2) NumPy:

* It is a numeric python module which provides fast maths functions for calculations.
* It is used to read data in numpy arrays and for manipulation purpose.

(3) Pandas:

* Used to read and write different files.
* Data manipulation can be done easily with dataframes.

(4) Matplotlib:

* Used to draw the comparison lines between the prediction figures and actual figures.

(5) Seaborn:

* Used to draw scatter chart between the variables

We also adapted the following functions/methods

(1) Data Import:

* To import and manipulate the data we are using the pandas package provided in python

(2) Data Slicing:

* Before training the model we have to split the dataset into the training and testing dataset
* To split the dataset for training and testing we are using the sklearn module train\_test\_split
* Random-state variable is a pseudo-random number generator state used for random sampling

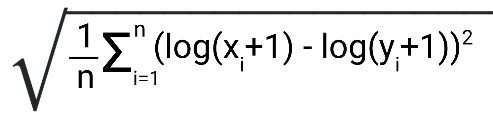
(3) Data Evaluating

* By exploring accuracy score, weighted average score of precision, weighted average score of recall, we evaluate the result.
* We used MSE to evaluate the result.

6.

We used Root Mean Squared Logarithmic Error (RMSLE) to evaluate the model. According to [Kaggle's own definition](https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError) of RMSLE, "RMSLE penalizes an under-predicted estimate greater than an over-predicted estimate."

The formula of RMLSE is shown as below.



7.

|  |  |  |
| --- | --- | --- |
| Number | Links | Notes |
| 1 | <https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full> |  |
| 2 | <https://www.youtube.com/watch?v=kho6oANGu_A> |  |
| 3 | <https://www.youtube.com/watch?v=9GCEVv94udY> |  |
| 4 | <https://lightgbm.readthedocs.io/en/latest/Python-Intro.html> |  |
| 5 | <https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc> |  |
| 6 | <https://anaconda.org/conda-forge/lightgbm> |  |
| 7 | <https://www.kaggle.com/nschneider/gbm-vs-xgboost-vs-lightgbm> |  |
| 8 | <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html> |  |
| 9 | <https://www.programcreek.com/python/example/102433/sklearn.ensemble.GradientBoostingRegressor> |  |
| 10 | <https://www.datatechnotes.com/2019/06/gradient-boosting-regression-example-in.html> |  |
| 11 | <https://rdrr.io/cran/MLmetrics/man/RMSLE.html> |  |

8. Algorithm

In [35]:

# ADA\_II

# HW 5

# Team 3

#Shiwen Chen (Leader)

#Jiahua Chen

#Qi Liu

**Set up environments¶**

In [36]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import norm, skew

from sklearn.linear\_model import LogisticRegression

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

from sklearn import metrics

from sklearn.metrics import mean\_squared\_error

from sklearn.tree import DecisionTreeClassifier

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

**Input data¶**

In [37]:

train\_data = pd.read\_csv('data\houseprice\_train.csv')

In [38]:

test\_data = pd.read\_csv('data\houseprice\_test.csv')

In [39]:

train\_data.head(5)

Out[39]:

|  | **Id** | **MSSubClass** | **MSZoning** | **LotFrontage** | **LotArea** | **Street** | **Alley** | **LotShape** | **LandContour** | **Utilities** | **...** | **PoolArea** | **PoolQC** | **Fence** | **MiscFeature** | **MiscVal** | **MoSold** | **YrSold** | **SaleType** | **SaleCondition** | **SalePrice** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 60 | RL | 65.0 | 8450 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 2 | 2008 | WD | Normal | 208500 |
| 1 | 2 | 20 | RL | 80.0 | 9600 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 5 | 2007 | WD | Normal | 181500 |
| 2 | 3 | 60 | RL | 68.0 | 11250 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 9 | 2008 | WD | Normal | 223500 |
| 3 | 4 | 70 | RL | 60.0 | 9550 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 2 | 2006 | WD | Abnorml | 140000 |
| 4 | 5 | 60 | RL | 84.0 | 14260 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 12 | 2008 | WD | Normal | 250000 |

5 rows × 81 columns

**Remove Outliers¶**

In [40]:

fig, axarr = plt.subplots(2, 2, figsize = (12, 8))

train\_data.plot.scatter(

x="GrLivArea",

y="SalePrice",

ax=axarr[0][0]

)

train\_data.plot.scatter(

x="BsmtFinSF1",

y="SalePrice",

ax=axarr[0][1]

)

train\_data.plot.scatter(

x="LotArea",

y="SalePrice",

ax=axarr[1][0]

)

train\_data.plot.scatter(

x="GarageArea",

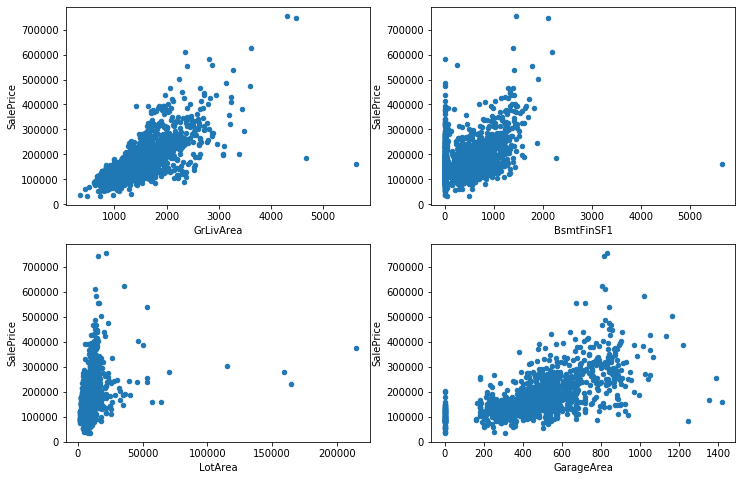
y="SalePrice",

ax=axarr[1][1]

)

Out[40]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x107cdfd4448>



In [41]:

train\_data = train\_data.drop(train\_data[(train\_data['GrLivArea']>4000) & (train\_data['SalePrice']<300000)].index)

train\_data = train\_data.drop(train\_data[(train\_data['LotArea']>150000)].index)

train\_data = train\_data.drop(train\_data[(train\_data['GarageArea']>1200) & (train\_data['SalePrice']<300000)].index)

fig, axarr = plt.subplots(2, 2, figsize = (12, 8))

train\_data.plot.scatter(

x="GrLivArea",

y="SalePrice",

ax=axarr[0][0]

)

train\_data.plot.scatter(

x="BsmtFinSF1",

y="SalePrice",

ax=axarr[0][1]

)

train\_data.plot.scatter(

x="LotArea",

y="SalePrice",

ax=axarr[1][0]

)

train\_data.plot.scatter(

x="GarageArea",

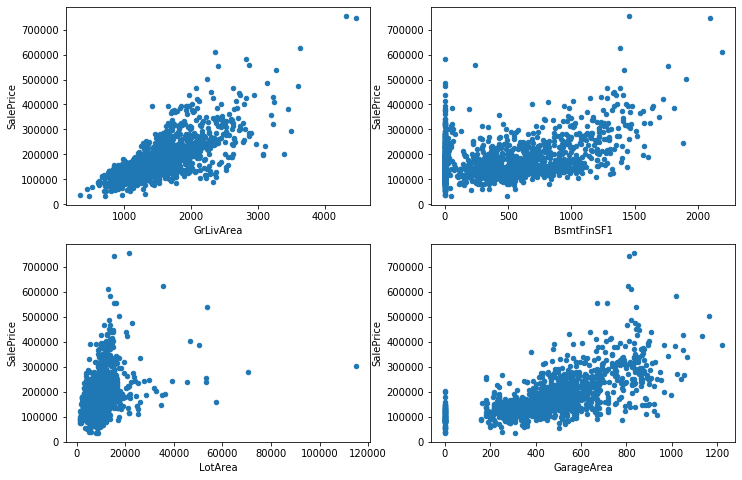
y="SalePrice",

ax=axarr[1][1]

)

Out[41]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x107ce0e9388>

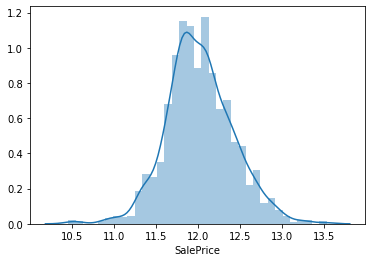


***Apply Log transfomation to SalePrice¶***

In [42]:

train\_data['SalePrice'] = np.log1p(train\_data['SalePrice'])

sns.distplot(train\_data['SalePrice']);



***Concat train data and test data¶***

In [43]:

Id = test\_data['Id']

train\_y = train\_data.SalePrice.values

all\_data = pd.concat((train\_data, test\_data), sort=False).reset\_index(drop=True)

all\_data.drop(['SalePrice'], axis=1, inplace=True)

print("all\_data size is : {}".format(all\_data.shape))

all\_data.head(5)

all\_data size is : (2911, 80)

Out[43]:

|  | **Id** | **MSSubClass** | **MSZoning** | **LotFrontage** | **LotArea** | **Street** | **Alley** | **LotShape** | **LandContour** | **Utilities** | **...** | **ScreenPorch** | **PoolArea** | **PoolQC** | **Fence** | **MiscFeature** | **MiscVal** | **MoSold** | **YrSold** | **SaleType** | **SaleCondition** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 60 | RL | 65.0 | 8450 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | 0 | NaN | NaN | NaN | 0 | 2 | 2008 | WD | Normal |
| 1 | 2 | 20 | RL | 80.0 | 9600 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | 0 | NaN | NaN | NaN | 0 | 5 | 2007 | WD | Normal |
| 2 | 3 | 60 | RL | 68.0 | 11250 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | 0 | NaN | NaN | NaN | 0 | 9 | 2008 | WD | Normal |
| 3 | 4 | 70 | RL | 60.0 | 9550 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | 0 | NaN | NaN | NaN | 0 | 2 | 2006 | WD | Abnorml |
| 4 | 5 | 60 | RL | 84.0 | 14260 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | 0 | NaN | NaN | NaN | 0 | 12 | 2008 | WD | Normal |

5 rows × 80 columns

**Dealing with missing data¶**

In [44]:

all\_data = all\_data.drop('Id', axis=1)

missing\_data = all\_data.isnull().sum()

missing\_data = missing\_data.drop(missing\_data[missing\_data == 0].index)

missing\_ratio = missing\_data / len(all\_data) \* 100

all\_data = all\_data.drop(missing\_ratio[missing\_ratio.values > 20].index, axis=1)

all\_data.head(5)

Out[44]:

|  | **MSSubClass** | **MSZoning** | **LotFrontage** | **LotArea** | **Street** | **LotShape** | **LandContour** | **Utilities** | **LotConfig** | **LandSlope** | **...** | **OpenPorchSF** | **EnclosedPorch** | **3SsnPorch** | **ScreenPorch** | **PoolArea** | **MiscVal** | **MoSold** | **YrSold** | **SaleType** | **SaleCondition** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 60 | RL | 65.0 | 8450 | Pave | Reg | Lvl | AllPub | Inside | Gtl | ... | 61 | 0 | 0 | 0 | 0 | 0 | 2 | 2008 | WD | Normal |
| 1 | 20 | RL | 80.0 | 9600 | Pave | Reg | Lvl | AllPub | FR2 | Gtl | ... | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 2007 | WD | Normal |
| 2 | 60 | RL | 68.0 | 11250 | Pave | IR1 | Lvl | AllPub | Inside | Gtl | ... | 42 | 0 | 0 | 0 | 0 | 0 | 9 | 2008 | WD | Normal |
| 3 | 70 | RL | 60.0 | 9550 | Pave | IR1 | Lvl | AllPub | Corner | Gtl | ... | 35 | 272 | 0 | 0 | 0 | 0 | 2 | 2006 | WD | Abnorml |
| 4 | 60 | RL | 84.0 | 14260 | Pave | IR1 | Lvl | AllPub | FR2 | Gtl | ... | 84 | 0 | 0 | 0 | 0 | 0 | 12 | 2008 | WD | Normal |

5 rows × 74 columns

In [16]:

missing\_data = all\_data.isnull().sum()

missing\_data = missing\_data.drop(missing\_data[missing\_data == 0].index)

missing\_ratio = missing\_data / len(all\_data) \* 100

print(missing\_ratio)

all\_data[missing\_ratio.index].head(5)

MSZoning 0.137410

LotFrontage 16.592236

Utilities 0.068705

Exterior1st 0.034352

Exterior2nd 0.034352

MasVnrType 0.824459

MasVnrArea 0.790106

BsmtQual 2.782549

BsmtCond 2.816901

BsmtExposure 2.816901

BsmtFinType1 2.713844

BsmtFinSF1 0.034352

BsmtFinType2 2.748196

BsmtFinSF2 0.034352

BsmtUnfSF 0.034352

TotalBsmtSF 0.034352

Electrical 0.034352

BsmtFullBath 0.068705

BsmtHalfBath 0.068705

KitchenQual 0.034352

Functional 0.068705

GarageType 5.393336

GarageYrBlt 5.462041

GarageFinish 5.462041

GarageCars 0.034352

GarageArea 0.034352

GarageQual 5.462041

GarageCond 5.462041

SaleType 0.034352

dtype: float64

Out[16]:

|  | **MSZoning** | **LotFrontage** | **Utilities** | **Exterior1st** | **Exterior2nd** | **MasVnrType** | **MasVnrArea** | **BsmtQual** | **BsmtCond** | **BsmtExposure** | **...** | **KitchenQual** | **Functional** | **GarageType** | **GarageYrBlt** | **GarageFinish** | **GarageCars** | **GarageArea** | **GarageQual** | **GarageCond** | **SaleType** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | RL | 65.0 | AllPub | VinylSd | VinylSd | BrkFace | 196.0 | Gd | TA | No | ... | Gd | Typ | Attchd | 2003.0 | RFn | 2.0 | 548.0 | TA | TA | WD |
| 1 | RL | 80.0 | AllPub | MetalSd | MetalSd | None | 0.0 | Gd | TA | Gd | ... | TA | Typ | Attchd | 1976.0 | RFn | 2.0 | 460.0 | TA | TA | WD |
| 2 | RL | 68.0 | AllPub | VinylSd | VinylSd | BrkFace | 162.0 | Gd | TA | Mn | ... | Gd | Typ | Attchd | 2001.0 | RFn | 2.0 | 608.0 | TA | TA | WD |
| 3 | RL | 60.0 | AllPub | Wd Sdng | Wd Shng | None | 0.0 | TA | Gd | No | ... | Gd | Typ | Detchd | 1998.0 | Unf | 3.0 | 642.0 | TA | TA | WD |
| 4 | RL | 84.0 | AllPub | VinylSd | VinylSd | BrkFace | 350.0 | Gd | TA | Av | ... | Gd | Typ | Attchd | 2000.0 | RFn | 3.0 | 836.0 | TA | TA | WD |

5 rows × 29 columns

In [17]:

all\_data["LotFrontage"] = all\_data.groupby("Neighborhood")["LotFrontage"].transform(lambda x: x.fillna(x.median()))

all\_data['Utilities'] = all\_data['Utilities'].fillna(all\_data['Utilities'].mode()[0])

all\_data['MSZoning'] = all\_data['MSZoning'].fillna(all\_data['MSZoning'].mode()[0])

all\_data['Utilities'] = all\_data['Utilities'].fillna(all\_data['Utilities'].mode()[0])

all\_data['Exterior1st'] = all\_data['Exterior1st'].fillna(all\_data['Exterior1st'].mode()[0])

all\_data['Exterior2nd'] = all\_data['Exterior2nd'].fillna(all\_data['Exterior2nd'].mode()[0])

all\_data['MasVnrType'] = all\_data['MasVnrType'].fillna(all\_data['MasVnrType'].mode()[0])

all\_data['Electrical'] = all\_data['Electrical'].fillna(all\_data['Electrical'].mode()[0])

all\_data['KitchenQual'] = all\_data['KitchenQual'].fillna(all\_data['KitchenQual'].mode()[0])

all\_data['Functional'] = all\_data['Functional'].fillna(all\_data['Functional'].mode()[0])

all\_data['SaleType'] = all\_data['SaleType'].fillna(all\_data['SaleType'].mode()[0])

all\_data['BsmtQual'] = all\_data['BsmtQual'].fillna('None')

all\_data['BsmtCond'] = all\_data['BsmtCond'].fillna('None')

all\_data['BsmtExposure'] = all\_data['BsmtExposure'].fillna('None')

all\_data['BsmtFinType1'] = all\_data['BsmtFinType1'].fillna('None')

all\_data['BsmtFinType2'] = all\_data['BsmtFinType2'].fillna('None')

all\_data['GarageType'] = all\_data['GarageType'].fillna('None')

all\_data['GarageFinish'] = all\_data['GarageFinish'].fillna('None')

all\_data['GarageQual'] = all\_data['GarageQual'].fillna('None')

all\_data['GarageCond'] = all\_data['GarageCond'].fillna('None')

all\_data['BsmtFinSF1'] = all\_data['BsmtFinSF1'].fillna(0)

all\_data['BsmtFinSF2'] = all\_data['BsmtFinSF2'].fillna(0)

all\_data['BsmtUnfSF'] = all\_data['BsmtUnfSF'].fillna(0)

all\_data['TotalBsmtSF'] = all\_data['TotalBsmtSF'].fillna(0)

all\_data['BsmtFullBath'] = all\_data['BsmtFullBath'].fillna(0)

all\_data['BsmtHalfBath'] = all\_data['BsmtHalfBath'].fillna(0)

all\_data['MasVnrArea'] = all\_data['MasVnrArea'].fillna(0)

all\_data['GarageYrBlt'] = all\_data['GarageYrBlt'].fillna(0)

all\_data['GarageCars'] = all\_data['GarageCars'].fillna(0)

all\_data['GarageArea'] = all\_data['GarageArea'].fillna(0)

In [18]:

all\_data = pd.get\_dummies(all\_data)

all\_data.head(5)

Out[18]:

|  | **MSSubClass** | **LotFrontage** | **LotArea** | **OverallQual** | **OverallCond** | **YearBuilt** | **YearRemodAdd** | **MasVnrArea** | **BsmtFinSF1** | **BsmtFinSF2** | **...** | **SaleType\_ConLw** | **SaleType\_New** | **SaleType\_Oth** | **SaleType\_WD** | **SaleCondition\_Abnorml** | **SaleCondition\_AdjLand** | **SaleCondition\_Alloca** | **SaleCondition\_Family** | **SaleCondition\_Normal** | **SaleCondition\_Partial** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 60 | 65.0 | 8450 | 7 | 5 | 2003 | 2003 | 196.0 | 706.0 | 0.0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 20 | 80.0 | 9600 | 6 | 8 | 1976 | 1976 | 0.0 | 978.0 | 0.0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 60 | 68.0 | 11250 | 7 | 5 | 2001 | 2002 | 162.0 | 486.0 | 0.0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 70 | 60.0 | 9550 | 7 | 5 | 1915 | 1970 | 0.0 | 216.0 | 0.0 | ... | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 60 | 84.0 | 14260 | 8 | 5 | 2000 | 2000 | 350.0 | 655.0 | 0.0 | ... | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

5 rows × 278 columns

***Split to train and test data¶***

In [19]:

ntrain = train\_data.shape[0]

ntest = test\_data.shape[0]

train = all\_data[:ntrain]

test = all\_data[ntrain:]

train\_x = train

print(train\_x.shape[0], train\_y.shape[0])

1452 1452

***Cross validation¶***

In [20]:

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, train\_y, scoring="neg\_mean\_squared\_error", cv = kf))

return(rmse)

**Select an algorithm¶**

We wanted to use lightbgm initially, however, after looking through a great number of related articles, we still failed to solve the problem "No module named 'lightgbm'", and we ran out of time. As a consequence, we determined to use GradientBoostingRegressor to predict the sales prices.

In [30]:

import lightgbm as lgb

**---------------------------------------------------------------------------**

**ModuleNotFoundError** Traceback (most recent call last)

**<ipython-input-30-5dacb4a27011>** in <module>

**----> 1 import** lightgbm **as** lgb

**ModuleNotFoundError**: No module named 'lightgbm'

In [31]:

from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor ( loss='huber', n\_estimators=150)

score = rmsle\_cv(model)

print("GBR score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))

GBR score: 0.1191 (0.0061)

***Mean square error validation¶***

In [34]:

def rmsle(y, y\_pred):

return np.sqrt(mean\_squared\_error(y, y\_pred))

***Train the selected model¶***

In [33]:

model.fit(train\_x, train\_y)

train\_prediction = model.predict(train)

prediction = np.expm1(model.predict(test.values))

print(rmsle(train\_y, train\_prediction))

# print(prediction)

0.08073781404963243

**Conclusion¶**

1. At my first attempt, I dropped all the columns that contain missing value. That's one way. The next attemt I tried to simply fill them with either some common value, or 0, or None.
2. Due to the limitation of time, we merely adapted GradientBoostingRegressor in this case and only evaluated this model, which means that we did not have camparison data and it is hard to tell whether the result is better than other algorithms.

**Addition Notes¶**

After trying all variety of methods, we failed to import lightgbm into Anaconda environment. It is a very useful algorithm, which is likely to be a great model to predict the sales prices.