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 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 $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, k = 1, 2, \ldots, K$

2. For k=1 to K:

a. Compute $r_{ikm} = y_{ik} - p_k(x_i), i = 1, 2, \ldots, N$

b. it a regression tree to the targets r_{ikm}, i=1,2,…,N, giving terminal regions R_{jim},j=1,2,…,J_m

c. Compute $\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} (r_{ikm})}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1-|r_{ikm})}, j = 1, 2, \ldots, J_m.$

d. Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm}).$

Output $\hat{f}_k(x) = f_{kM}(x), k = 1, 2, \ldots, K$

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.

I. GBMs will continue improving to minimize all errors. This can overemphasize outliers and cause overfitting. Must use cross-validation to neutralize.
II. Computationally expensive - GBMs often require many trees (>1000) which can be time and memory exhaustive.
III. 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.).
IV. 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 of RMSLE, "RMSLE penalizes an under-predicted estimate greater than an over-predicted estimate."

The formula of RMLSE is shown as below.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(x_i+1) - \log(y_i+1))^2}$$

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

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
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

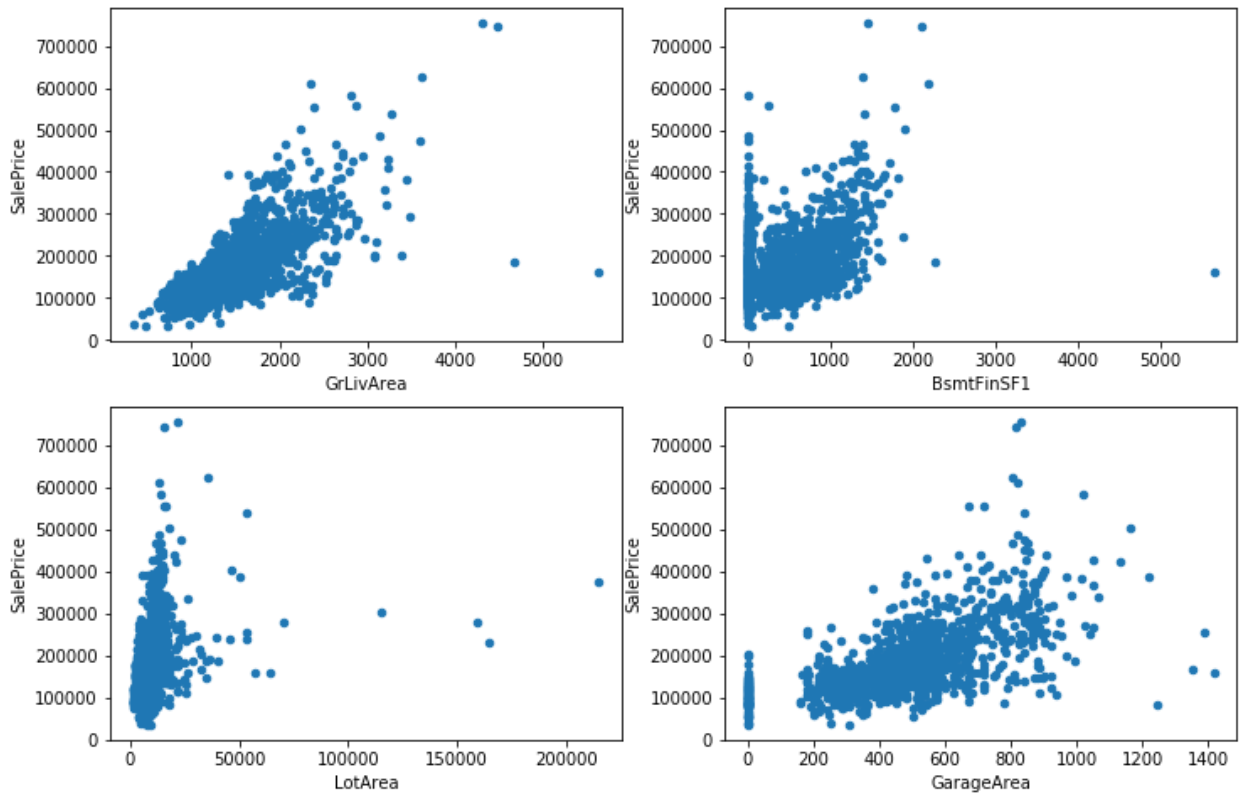
Remove Outliers

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>
```



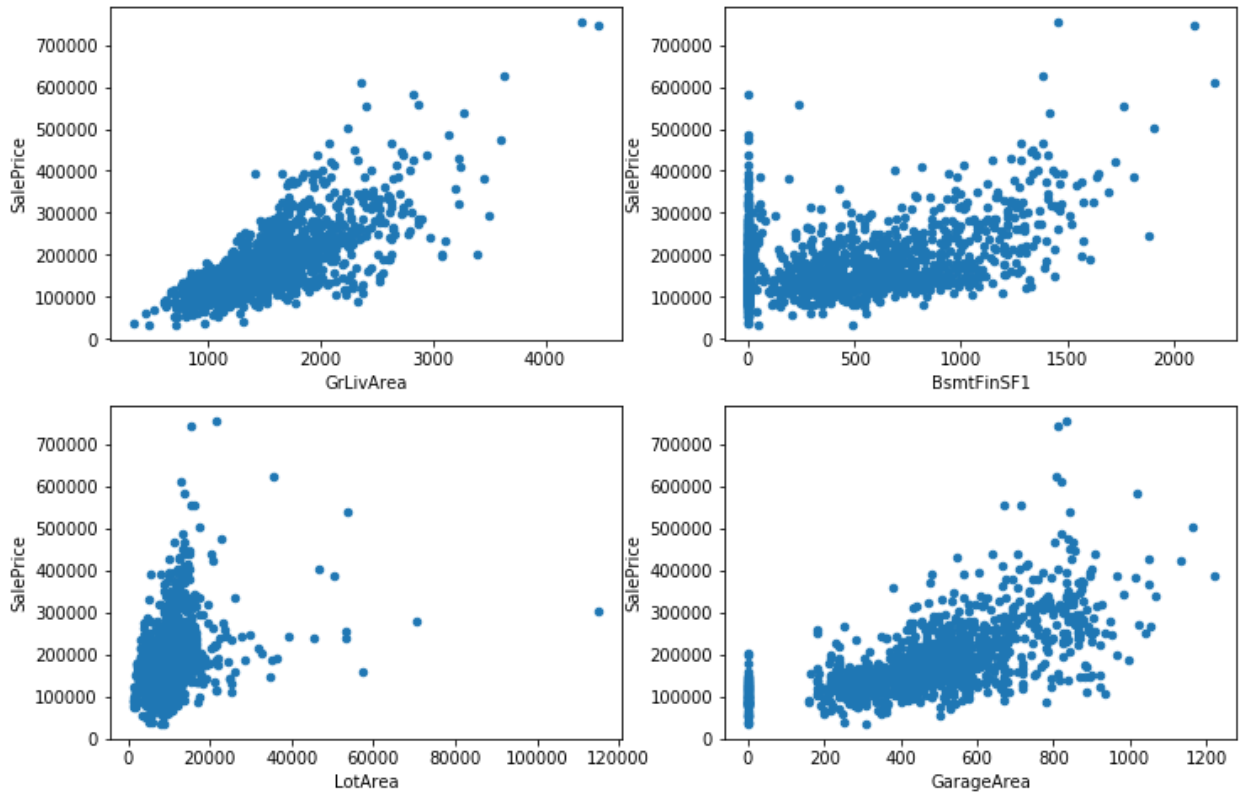
In [41]:

```
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]:

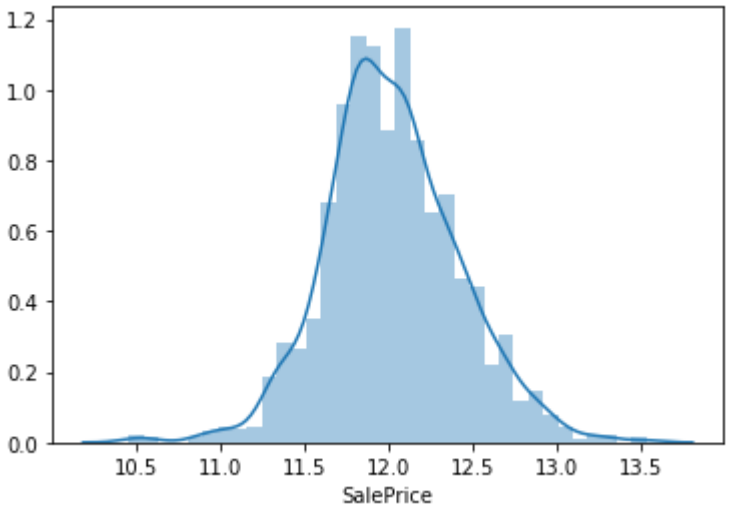
```
<matplotlib.axes._subplots.AxesSubplot at 0x107ce0e9388>
```



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

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition
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

	MSZoning	LotFrontage	Utilities	Exterior1st	Exterior2nd	MasVnrType	MasVnrArea	BsmtQual	BsmtCond	BsmtExposure	...	KitchenQual	Functional	GarageType	GarageYrBlt	GarageFinish	GarageCars	GarageArea	GarageQual	GarageCond	SaleType
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	WdSdng	WdShng	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

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
		60			00															

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.

1. 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.