House_Price_Prediction_v1

January 19, 2020

1 House Price Prediction --- version 1

Charles Zhang **Jan 19 2020**

```
In [3]: import numpy as np
        import seaborn as sns
        import pandas as pd
        import matplotlib.pyplot as plt
        from scipy import stats
        %matplotlib inline
        from sklearn.linear_model import Ridge
        from sklearn.model_selection import cross_val_score
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.linear_model import Ridge
        from sklearn.ensemble import BaggingRegressor
        from sklearn.model_selection import cross_val_score
        from sklearn.ensemble import AdaBoostRegressor
        from xgboost import XGBRegressor
```

2 1. Process Data

```
In [8]: test_df = pd.read_csv("https://raw.githubusercontent.com/zcczhang/House_Price_Predicti
        train_df = pd.read_csv("https://raw.githubusercontent.com/zcczhang/House_Price_Predict
        train = pd.read_csv("https://raw.githubusercontent.com/zcczhang/House_Price_Prediction_
```

```
In [9]: train_df.head()
```

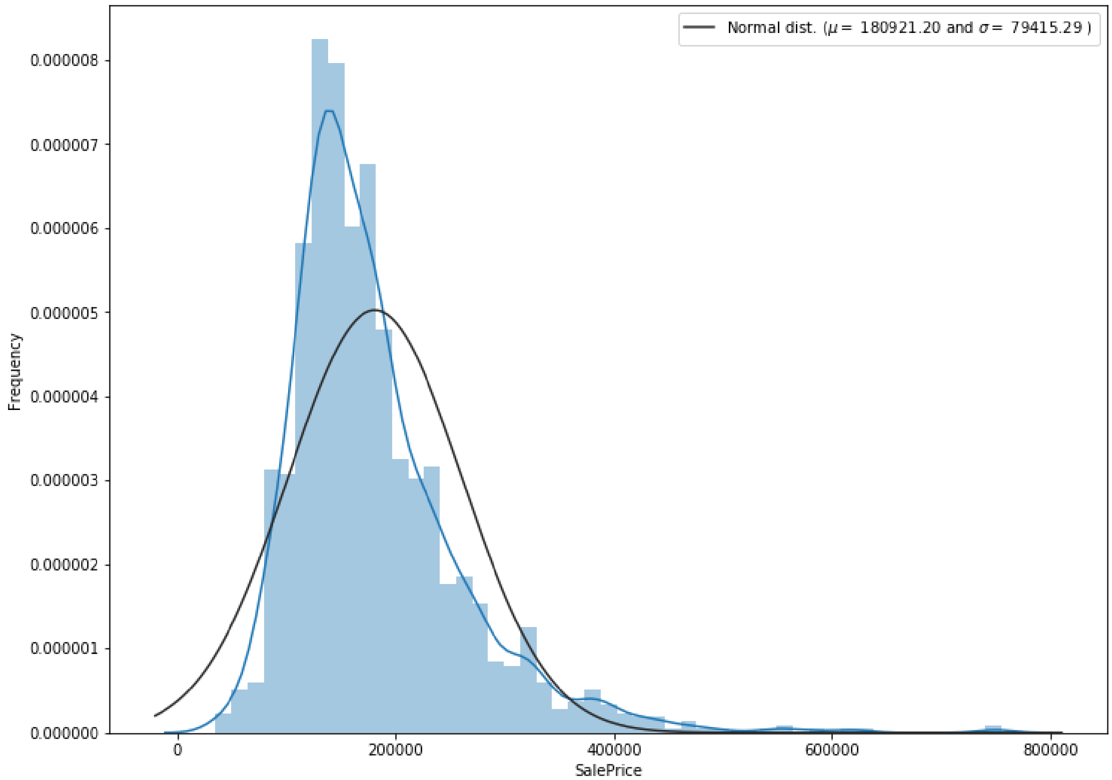
```
Out[9]:    MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
        Id
        1          60       RL         65.0     8450   Pave   NaN      Reg
        2          20       RL         80.0     9600   Pave   NaN      Reg
        3          60       RL         68.0    11250   Pave   NaN      IR1
        4          70       RL         60.0     9550   Pave   NaN      IR1
        5          60       RL         84.0    14260   Pave   NaN      IR1

          LandContour Utilities LotConfig    ...     PoolArea PoolQC Fence  \
        Id                                   ...
```

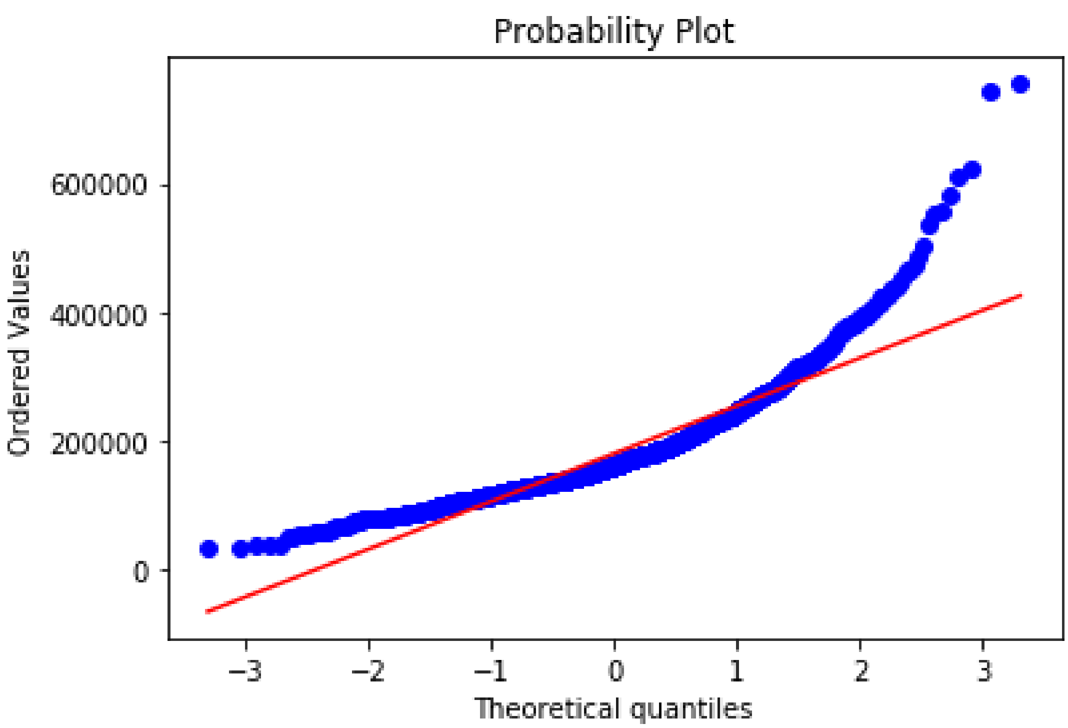
```
1          Lvl    AllPub    Inside    ...           0    NaN    NaN
2          Lvl    AllPub       FR2    ...           0    NaN    NaN
3          Lvl    AllPub    Inside    ...           0    NaN    NaN
4          Lvl    AllPub    Corner    ...           0    NaN    NaN
5          Lvl    AllPub       FR2    ...           0    NaN    NaN

   MiscFeature MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
Id
1          NaN       0      2    2008        WD         Normal     208500
2          NaN       0      5    2007        WD         Normal     181500
3          NaN       0      9    2008        WD         Normal     223500
4          NaN       0      2    2006        WD        Abnorml     140000
5          NaN       0     12    2008        WD         Normal     250000

[5 rows x 80 columns]
```

In [10]:
```
#shape of train data
train_df.shape
```

Out[10]: (1460, 80)

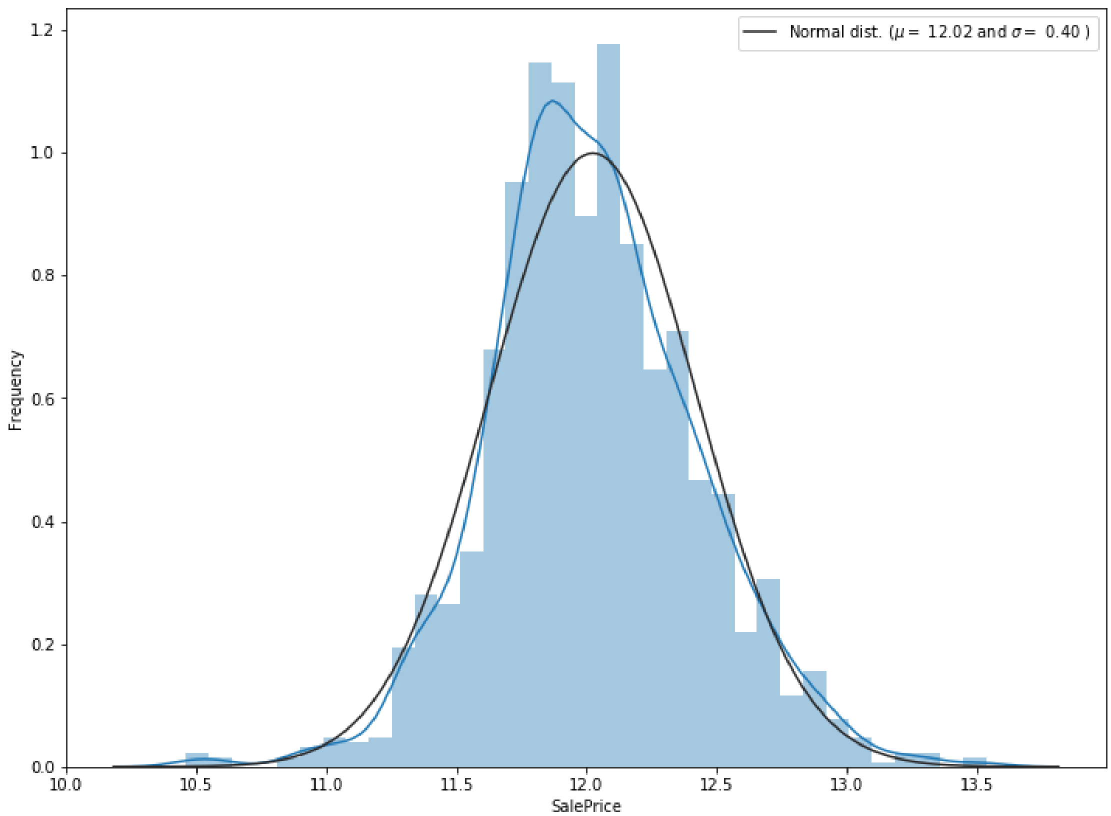
In [11]:
```
plt.subplots(figsize=(12,9))
sns.distplot(train['SalePrice'], fit=stats.norm)

# Get the fitted parameters used by the function

(mu, sigma) = stats.norm.fit(train['SalePrice'])

# plot with the distribution

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], 
plt.ylabel('Frequency')

#Probablity plot

fig = plt.figure()
stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```

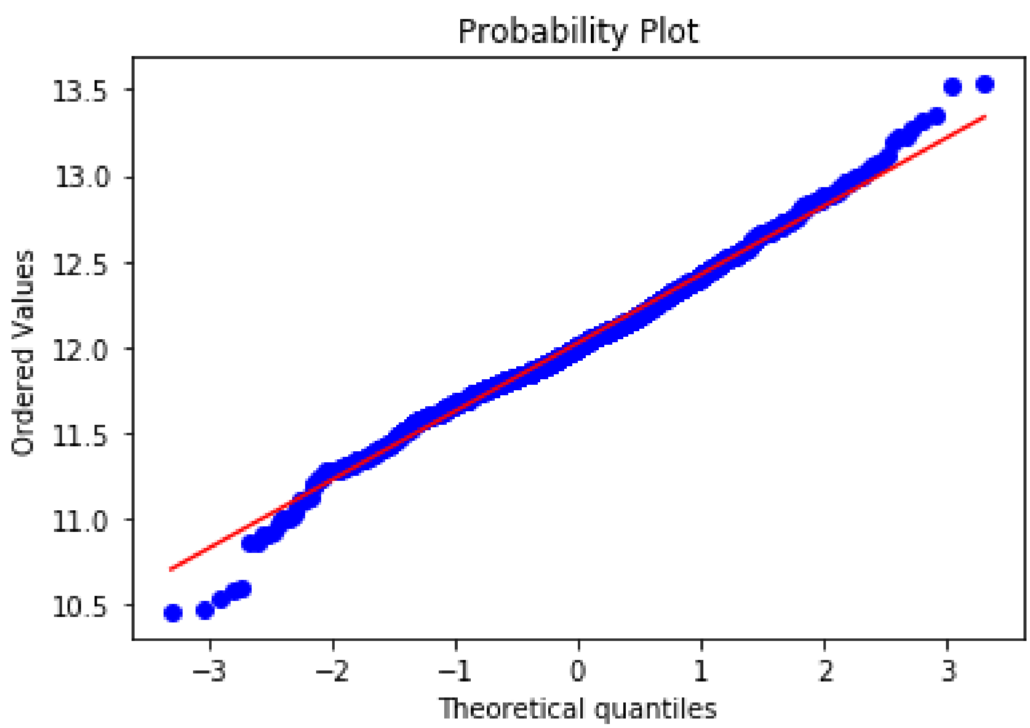

Normal dist. ($\mu =$ 180921.20 and $\sigma =$ 79415.29)
0.000008
0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0.000000
Frequency
0
200000
400000
600000
800000
SalePrice


Probability Plot
600000
400000
200000
0
Ordered Values
−3
−2
−1
0
1
2
3
Theoretical quantiles

This target varibale is right skewed. Now, we need to tranform this variable and make it normal distribution.

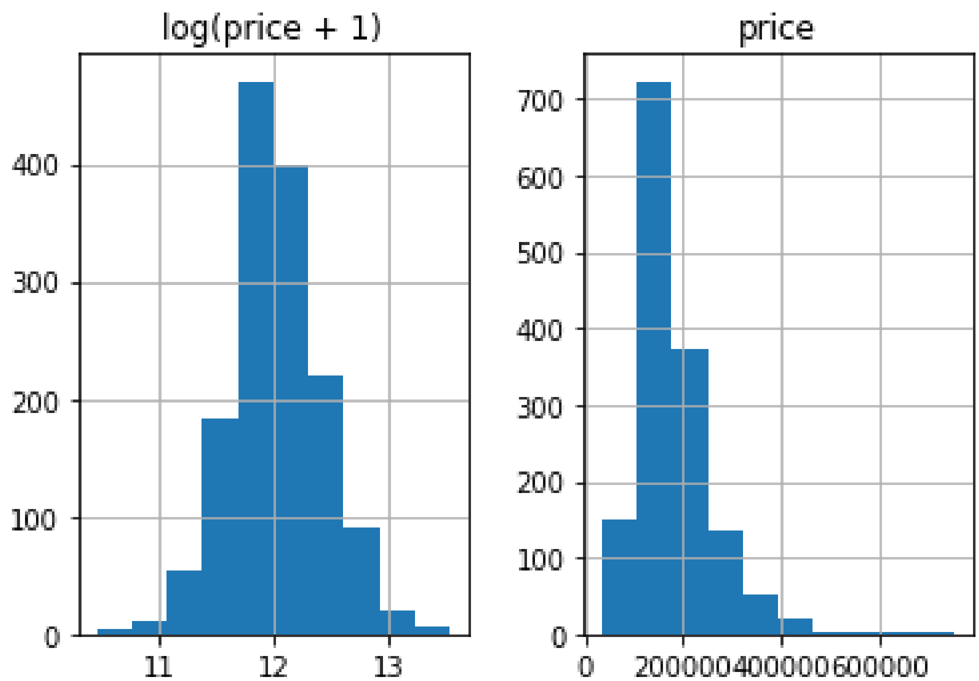
```
In [12]: #we use log function which is in numpy
         train['SalePrice'] = np.log1p(train['SalePrice'])

         #Check again for more normal distribution

         plt.subplots(figsize=(12,9))
         sns.distplot(train['SalePrice'], fit=stats.norm)

         # Get the fitted parameters used by the function

         (mu, sigma) = stats.norm.fit(train['SalePrice'])

         # plot with the distribution

         plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], l
         plt.ylabel('Frequency')

         #Probablity plot

         fig = plt.figure()
         stats.probplot(train['SalePrice'], plot=plt)
         plt.show()
```


Normal dist. ($\mu =$ 12.02 and $\sigma =$ 0.40)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
10.0
10.5
11.0
11.5
12.0
12.5
13.0
13.5
SalePrice


Probability Plot
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Ordered Values
−3
−2
−1
0
1
2
3
Theoretical quantiles

Simple Version

```
In [13]: prices = pd.DataFrame({"price":train_df["SalePrice"], "log(price + 1)":np.log1p(train_
         prices.hist()
         y_train = np.log1p(train_df.pop('SalePrice'))
```



3 2. Transform Variables

```
In [14]: # connect
         all_df = pd.concat((train_df, test_df), axis=0)
         all_df.shape

Out[14]: (2919, 79)
```

change variables attributes

```
In [15]: all_df['MSSubClass'].dtypes

Out[15]: dtype('int64')

In [16]: all_df['MSSubClass'] = all_df['MSSubClass'].astype(str)
         all_dummy_df = pd.get_dummies(all_df)
         all_dummy_df.head()
```

```
Out[16]:     LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  YearRemodAdd  \
        Id
        1          65.0     8450            7            5       2003          2003
        2          80.0     9600            6            8       1976          1976
        3          68.0    11250            7            5       2001          2002
        4          60.0     9550            7            5       1915          1970
        5          84.0    14260            8            5       2000          2000

            MasVnrArea  BsmtFinSF1  BsmtFinSF2  BsmtUnfSF            ...             \
        Id                                                           ...
        1        196.0       706.0         0.0      150.0            ...
        2          0.0       978.0         0.0      284.0            ...
        3        162.0       486.0         0.0      434.0            ...
        4          0.0       216.0         0.0      540.0            ...
        5        350.0       655.0         0.0      490.0            ...

            SaleType_ConLw  SaleType_New  SaleType_Oth  SaleType_WD  \
        Id
        1                0             0             0            1
        2                0             0             0            1
        3                0             0             0            1
        4                0             0             0            1
        5                0             0             0            1

            SaleCondition_Abnorml  SaleCondition_AdjLand  SaleCondition_Alloca  \
        Id
        1                       0                      0                     0
        2                       0                      0                     0
        3                       0                      0                     0
        4                       1                      0                     0
        5                       0                      0                     0

            SaleCondition_Family  SaleCondition_Normal  SaleCondition_Partial
        Id
        1                      0                     1                      0
        2                      0                     1                      0
        3                      0                     1                      0
        4                      0                     0                      0
        5                      0                     1                      0

        [5 rows x 303 columns]
```

use get_dummies method to use One-Hot method to represent the categories, dividing into 12 classes, true=1 while false=0

Check the Missing Value

```
In [17]: all_dummy_df.isnull().sum().sort_values(ascending=False).head(10)
```

```
Out[17]: LotFrontage     486
         GarageYrBlt     159
         MasVnrArea       23
         BsmtHalfBath      2
         BsmtFullBath      2
         BsmtFinSF2        1
         GarageCars        1
         TotalBsmtSF       1
         BsmtUnfSF         1
         GarageArea        1
         dtype: int64
```

```
In [18]: # use means to fill
         mean_cols = all_dummy_df.mean()
         all_dummy_df = all_dummy_df.fillna(mean_cols)
```

3.0.1 normalization

```
In [19]: numeric_cols = all_df.columns[all_df.dtypes != 'object']
         numeric_col_means = all_dummy_df.loc[:, numeric_cols].mean()
         numeric_col_std = all_dummy_df.loc[:, numeric_cols].std()
         # Standard distribution (X-X')/s(or use log)
         all_dummy_df.loc[:, numeric_cols] = (all_dummy_df.loc[:, numeric_cols] - numeric_col_r
```

```
In [20]: all_dummy_df.isnull().sum().sum()
```

```
Out[20]: 0
```

4 3. Build Models

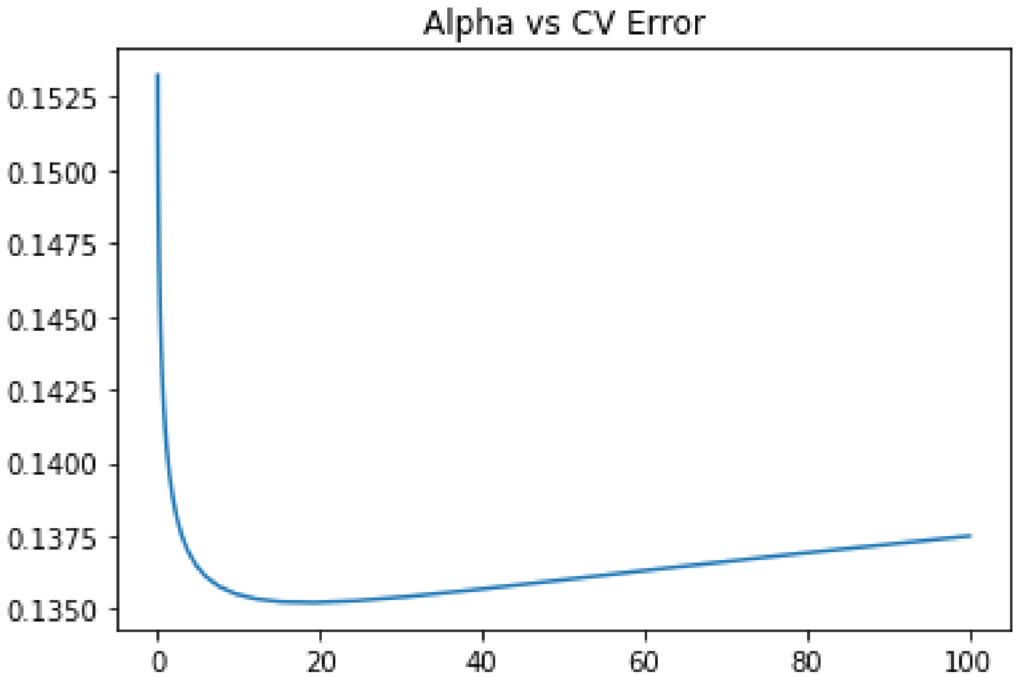
```
In [21]: dummy_train_df = all_dummy_df.loc[train_df.index]
         dummy_test_df = all_dummy_df.loc[test_df.index]
         dummy_train_df.shape, dummy_test_df.shape
```

```
Out[21]: ((1460, 303), (1459, 303))
```

- ### Ridge Regression

```
In [22]: X_train = dummy_train_df.values
         X_test = dummy_test_df.values

         alphas = np.logspace(-3, 2, 50)
         test_scores = []
         for alpha in alphas:
             clf = Ridge(alpha)
             test_score = np.sqrt(-cross_val_score(clf, X_train, y_train, cv=10, scoring='neg_r
             test_scores.append(np.mean(test_score))
         # choose and see the best alpha
```

```
plt.plot(alphas, test_scores)
plt.title("Alpha vs CV Error");
```

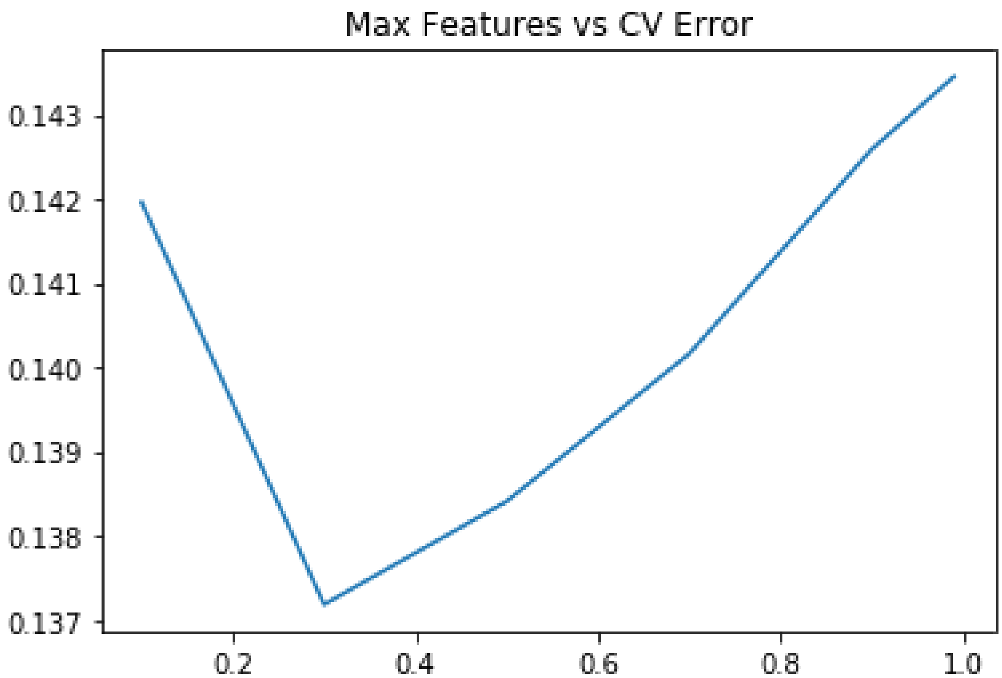


when alpha = 10-20, teh score could be around 0.135

-

4.0.1 Random Forest

```
In [23]: max_features = [.1, .3, .5, .7, .9, .99]
         test_scores = []
         for max_feat in max_features:
             clf = RandomForestRegressor(n_estimators=200, max_features=max_feat)
             test_score = np.sqrt(-cross_val_score(clf, X_train, y_train, cv=5, scoring='neg_me
             test_scores.append(np.mean(test_score))
         plt.plot(max_features, test_scores)
         plt.title("Max Features vs CV Error");
```



use alpha = 18 as the best parameter to ensemble

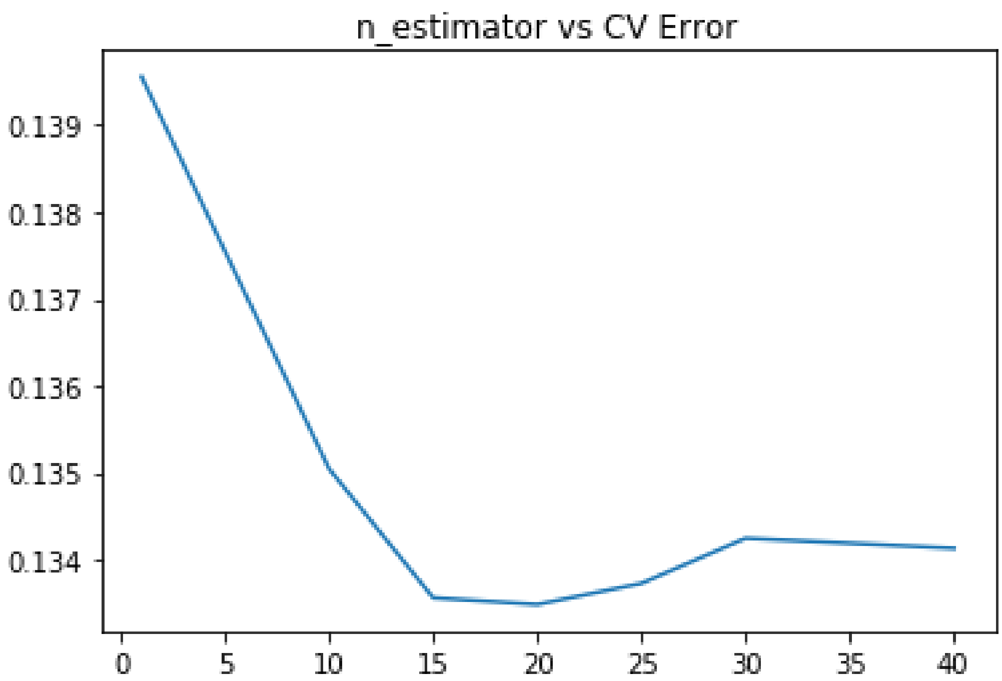
```
In [24]: ridge = Ridge(15.5)
```

-

4.0.2 Bagging

```
In [25]: params = [1, 10, 15, 20, 25, 30, 40]
         test_scores = []
         for param in params:
             clf = BaggingRegressor(n_estimators=param, base_estimator=ridge)
             test_score = np.sqrt(-cross_val_score(clf, X_train, y_train, cv=10, scoring='neg_
             test_scores.append(np.mean(test_score))
```

```
In [26]: plt.plot(params, test_scores)
         plt.title("n_estimator vs CV Error");
```

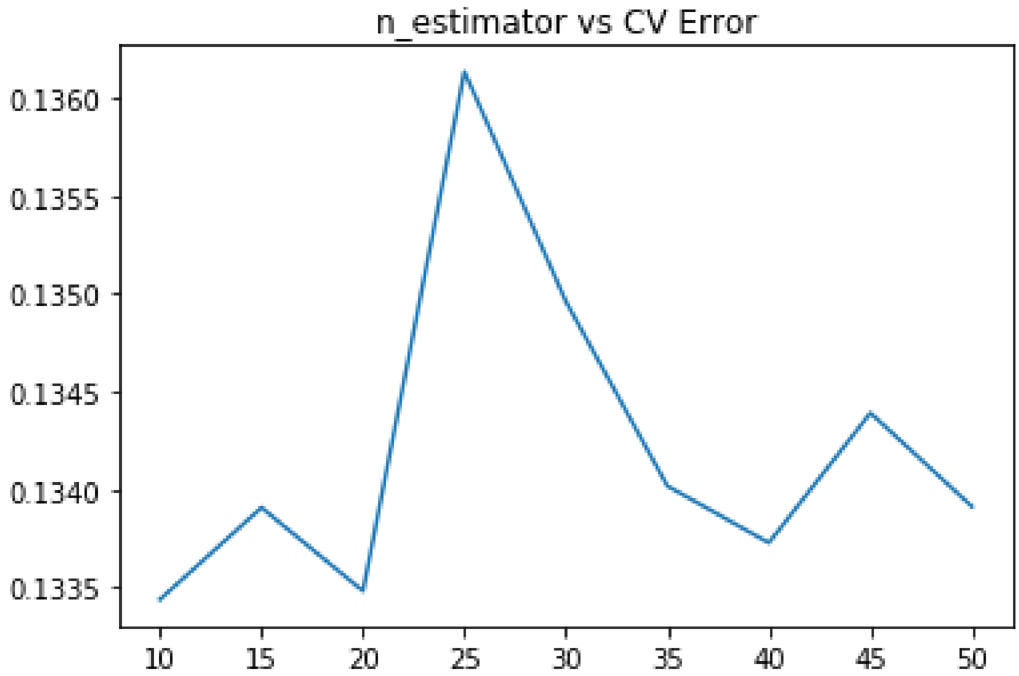


around 0.133

-

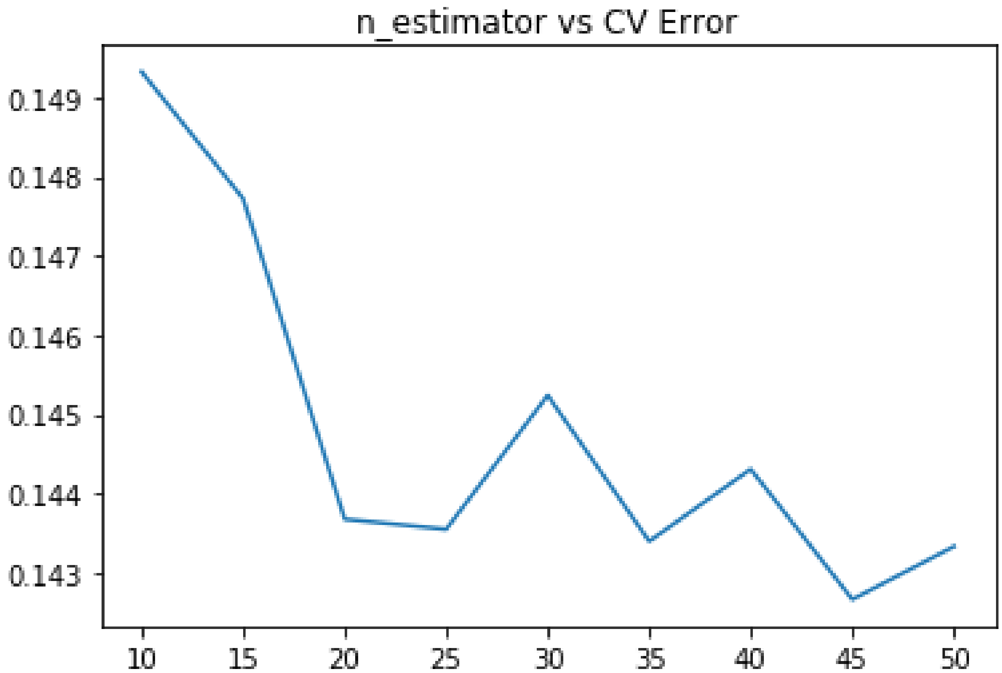
4.0.3 Boosting

In [27]:
```
params = [10, 15, 20, 25, 30, 35, 40, 45, 50]
test_scores = []
for param in params:
    clf = BaggingRegressor(n_estimators=param, base_estimator=ridge)
    test_score = np.sqrt(-cross_val_score(clf, X_train, y_train, cv=10, scoring='neg_m
    test_scores.append(np.mean(test_score))
plt.plot(params, test_scores)
plt.title("n_estimator vs CV Error");
```



Adaboost+Ridge

```
In [28]: params = [10, 15, 20, 25, 30, 35, 40, 45, 50]
         test_scores = []
         for param in params:
             clf = BaggingRegressor(n_estimators=param)
             test_score = np.sqrt(-cross_val_score(clf, X_train, y_train, cv=10, scoring='neg_m
             test_scores.append(np.mean(test_score))
         plt.plot(params, test_scores)
         plt.title("n_estimator vs CV Error");
```

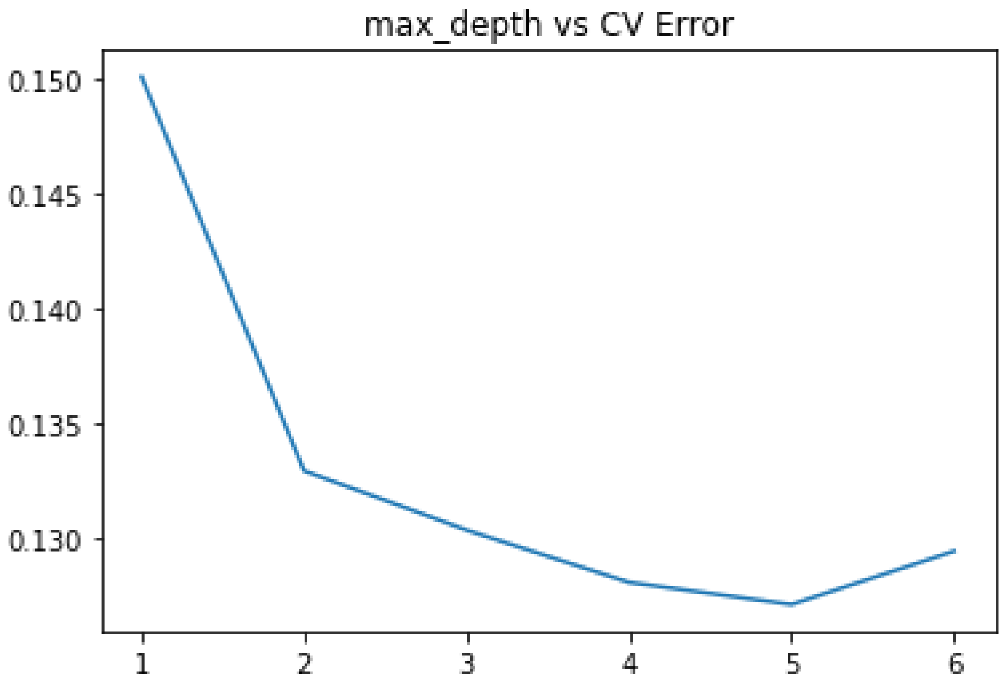


with default DT

-

4.0.4 XGboost

```
In [29]: params = [1,2,3,4,5,6]
         test_scores = []
         for param in params:
             clf = XGBRegressor(max_depth=param)
             test_score = np.sqrt(-cross_val_score(clf, X_train, y_train, cv=10, scoring='neg_m
             test_scores.append(np.mean(test_score))

In [30]: plt.plot(params, test_scores)
         plt.title("max_depth vs CV Error");
```



score is around 0.127, which is the best for now

In [31]:
```
xg = XGBRegressor(n_estimators=500, max_features=.3)
```

In [32]:
```
ridge.fit(X_train, y_train)
xg.fit(X_train, y_train)
```

Out[32]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, max_features=0.3, min_child_weight=1, missing=None,
       n_estimators=500, n_jobs=1, nthread=None, objective='reg:linear',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

In [33]:
```
y_ridge = np.expm1(ridge.predict(X_test))
y_xg = np.expm1(xg.predict(X_test))
```

In [34]:
```
y_final = (y_ridge + y_xg) / 2
```

In [35]:
```
submission_df = pd.DataFrame(data= {'Id' : test_df.index, 'SalePrice': y_final})
```

In [36]:
```
submission_df.head(10)
```

Out[36]:

	Id	SalePrice
0	1461	118750.320098
1	1462	154573.147591

```
2  1463  178348.981084
3  1464  193365.913566
4  1465  189740.249516
5  1466  171885.553885
6  1467  181790.259736
7  1468  163427.517004
8  1469  189001.784912
9  1470  123092.090870
```

In [42]:
```
submission_df.to_csv('submission.csv')
```