In [58]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
```

In [59]:
```
dataset = pd.read_csv('/Users/SUMA/Downloads/Melbourne_housing_FULL.csv')
```

Data Analysis

In [63]:
```
dataset
```

Out[63]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Dista
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	
...	...	...	...	...	...	...	...	...	
34852	Yarraville	13 Burns St	4	h	1480000.0	PI	Jas	24/02/2018	
34853	Yarraville	29A Murray St	2	h	888000.0	SP	Sweeney	24/02/2018	
34854	Yarraville	147A Severn St	2	t	705000.0	S	Jas	24/02/2018	
34855	Yarraville	12/37 Stephen St	3	h	1140000.0	SP	hockingstuart	24/02/2018	
34856	Yarraville	3 Tarrengower St	2	h	1020000.0	PI	RW	24/02/2018	

34857 rows × 21 columns

In [64]:
```
dataset.shape
```

Out[64]:
```
(34857, 21)
```

In [65]:
```
dataset.describe()
```

Out[65]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	
count	34857.000000	2.724700e+04	34856.000000	34856.000000	26640.000000	26631.000000	261
mean	3.031012	1.050173e+06	11.184929	3116.062859	3.084647	1.624798	
std	0.969933	6.414671e+05	6.788892	109.023903	0.980690	0.724212	
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	
25%	2.000000	6.350000e+05	6.400000	3051.000000	2.000000	1.000000	
50%	3.000000	8.700000e+05	10.300000	3103.000000	3.000000	2.000000	
75%	4.000000	1.295000e+06	14.000000	3156.000000	4.000000	2.000000	
max	16.000000	1.120000e+07	48.100000	3978.000000	30.000000	12.000000	

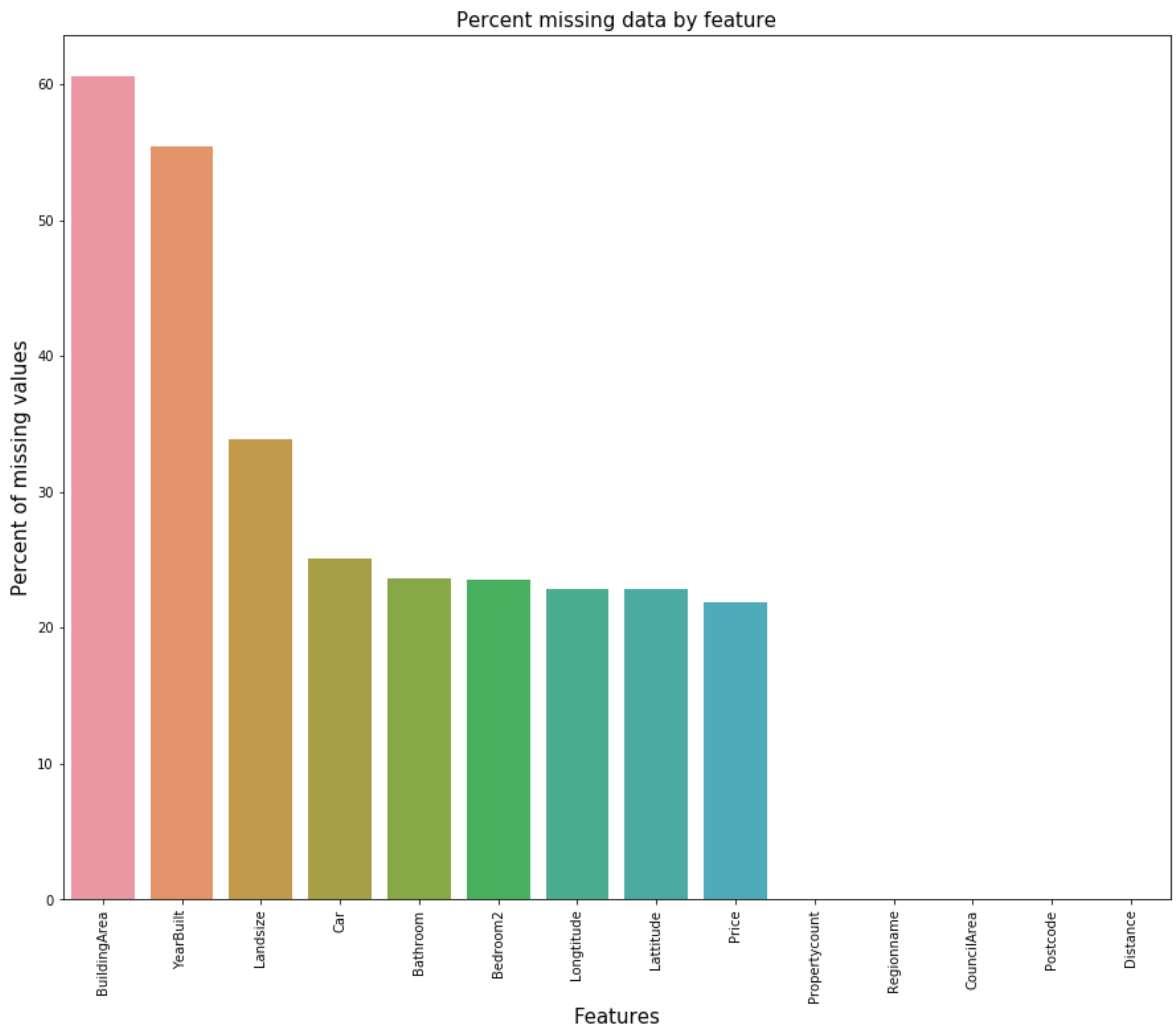
In [66]:
```
all_data_na = (dataset.isnull().sum() / len(dataset)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_value
s(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data
```

Out[66]:

	Missing Ratio
BuildingArea	60.576068
YearBuilt	55.386293
Landsize	33.881286
Car	25.039447
Bathroom	23.599277
Bedroom2	23.573457
Longtitude	22.882061
Lattitude	22.882061
Price	21.832057
Propertycount	0.008607
Regionname	0.008607
CouncilArea	0.008607
Postcode	0.002869
Distance	0.002869

In [67]:
```
f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='90')
sns.barplot(x=all_data_na.index, y=all_data_na)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Percent of missing values', fontsize=15)
plt.title('Percent missing data by feature', fontsize=15)
```

Out[67]:
```
Text(0.5, 1.0, 'Percent missing data by feature')
```



In [68]:
```
dataset = dataset[~ dataset.Price.isnull() ] ## Eliminated recores with "Pric
e" null
```

In [69]:
```
dataset.reset_index(drop=True, inplace=True)
```

In [70]:
```
from sklearn.model_selection import ShuffleSplit
shuffleSplit = ShuffleSplit(n_splits=1,test_size = 0.2 , random_state=42)

for train_index, test_index in shuffleSplit.split(dataset):
    training_set = dataset.loc[train_index]
    test_set = dataset.loc[test_index]
```

In [71]:
```
training_set.isnull().any()
```

Out[71]:
```
Suburb           False
Address          False
Rooms            False
Type             False
Price            False
Method           False
SellerG          False
Date             False
Distance          True
Postcode          True
Bedroom2          True
Bathroom          True
Car               True
Landsize          True
BuildingArea      True
YearBuilt         True
CouncilArea       True
Lattitude         True
Longtitude        True
Regionname        True
Propertycount     True
dtype: bool
```

In [72]:
```
training_set = training_set[~ training_set['Postcode'].isnull()]
```

In [73]:
```
test_set = test_set[~ test_set['Postcode'].isnull()]
```

In [74]:
```
training_set.drop(["Address", "CouncilArea", "Regionname", "Lattitude", "Subur
b", "Longtitude","Type","Method","SellerG","Date"], axis=1, inplace=True)
```

In [75]:
```
test_set.drop(["Address", "CouncilArea", "Regionname", "Lattitude", "Suburb",
"Longtitude","Type","Method","SellerG","Date"], axis=1, inplace=True)
```

In [76]:
```
training_set.select_dtypes(['float64','int64']).isnull().any()
```

Out[76]:
```
Rooms            False
Price            False
Distance         False
Postcode         False
Bedroom2          True
Bathroom          True
Car               True
Landsize          True
BuildingArea      True
YearBuilt         True
Propertycount     True
dtype: bool
```

In [77]:
```
training_set.Bedroom2.fillna(value=training_set.Bedroom2.mean(), inplace=True)
training_set.Bathroom.fillna(value=training_set.Bathroom.mode()[0], inplace=Tr
ue)
training_set.Car.fillna(value=training_set.Car.median(), inplace=True)
training_set.fillna(value= training_set.mean()[["BuildingArea", "YearBuilt",
"Propertycount"]], inplace=True)
training_set["Landsize_log"] = np.log(training_set[~training_set.Landsize.isnu
ll() & training_set.Landsize > 0]['Landsize'])
Landsize_log_mean = training_set["Landsize_log"].mean()
training_set["Landsize_log"].fillna(value=Landsize_log_mean, inplace=True)
training_set["Landsize_log"] = training_set["Landsize_log"].apply(lambda x: La
ndsize_log_mean if x == 0 else x)
training_set.drop('Landsize', axis=1, inplace=True)
```

In [78]:
```
test_set.Bedroom2.fillna(value=test_set.Bedroom2.mean(), inplace=True)
test_set.Bathroom.fillna(value=test_set.Bathroom.mode()[0], inplace=True)
test_set.Car.fillna(value=test_set.Car.median(), inplace=True)
test_set.fillna(value= test_set.mean()[["BuildingArea", "YearBuilt", "Property
count"]], inplace=True)
test_set["Landsize_log"] = np.log(test_set[~test_set.Landsize.isnull() & test_
set.Landsize > 0]['Landsize'])
Landsize_log_mean = test_set["Landsize_log"].mean()
test_set["Landsize_log"].fillna(value=Landsize_log_mean, inplace=True)
test_set["Landsize_log"] = test_set["Landsize_log"].apply(lambda x: Landsize_l
og_mean if x == 0 else x)
test_set.drop('Landsize', axis=1, inplace=True)
```

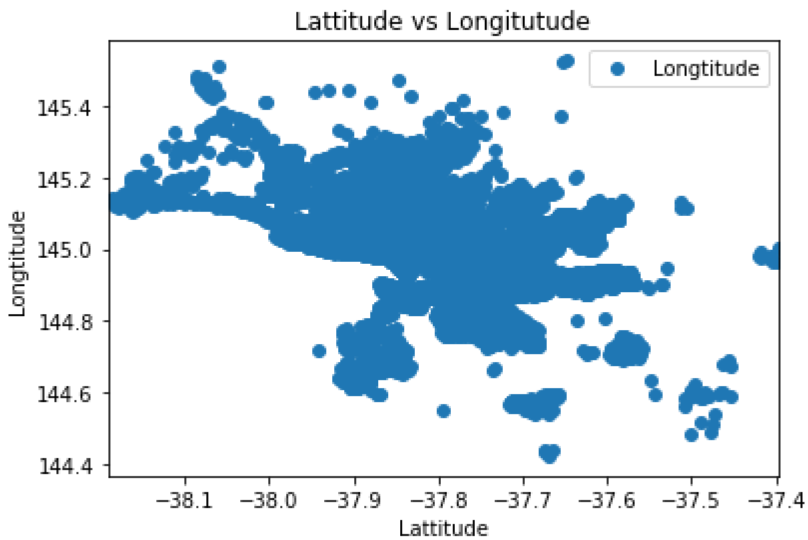
In [79]:
```
training_set.isnull().any()
```

Out[79]:
```
Rooms            False
Price            False
Distance         False
Postcode         False
Bedroom2         False
Bathroom         False
Car              False
BuildingArea     False
YearBuilt        False
Propertycount    False
Landsize_log     False
dtype: bool
```

Data visualization

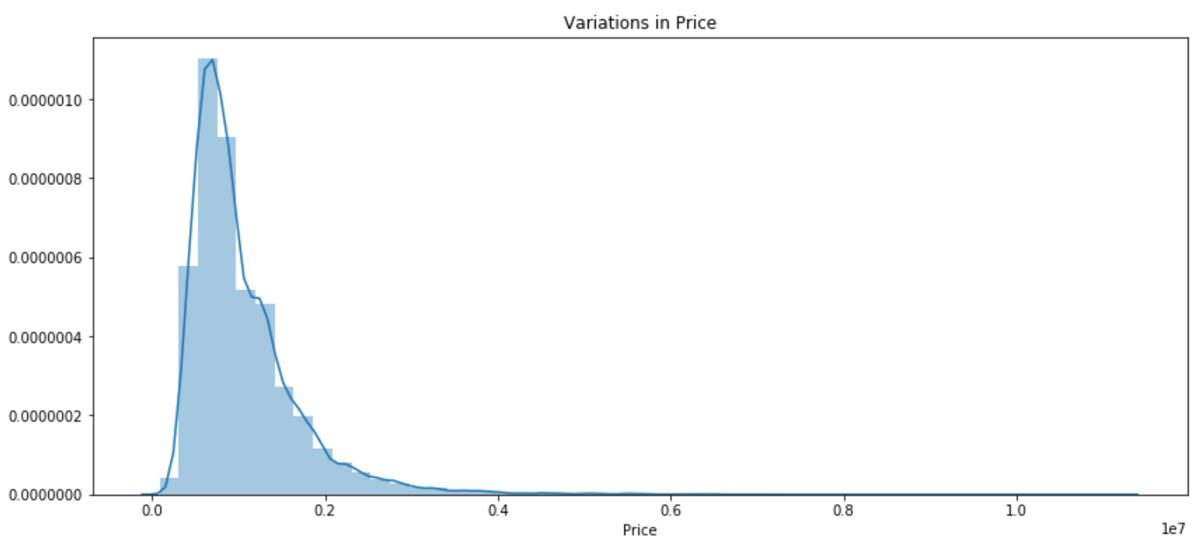
In [80]:
```
plt.figure(figsize=(14,6))
dataset.plot(x='Lattitude', y='Longtitude', style='o')
plt.title('Lattitude vs Longitutude')
plt.xlabel('Lattitude')
plt.ylabel('Longtitude')
plt.show()
```

```
<Figure size 1008x432 with 0 Axes>
```



In [82]:
```
plt.figure(figsize=(14,6))
plt.tight_layout()
plt.title('Variations in Price')
sns.distplot(dataset['Price'])
```

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x4fd39c9f88>
```

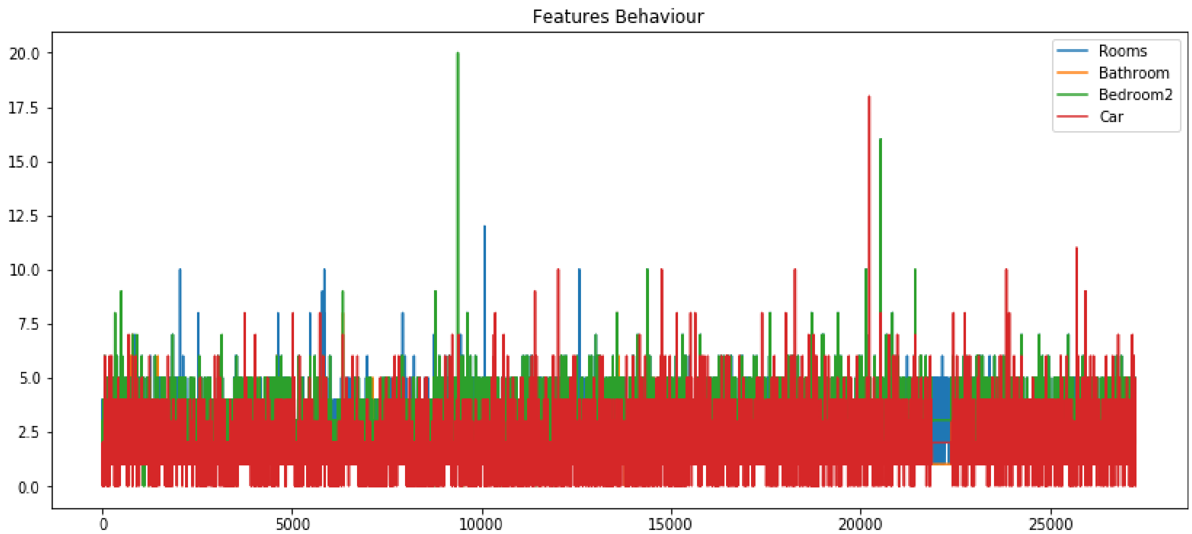


In [83]:
```python
plt.figure(figsize=(14,6))

# Add title
plt.title("Features Behaviour")


sns.lineplot(data=training_set['Rooms'], label="Rooms")
sns.lineplot(data=training_set['Bathroom'], label="Bathroom")
sns.lineplot(data=training_set['Bedroom2'], label="Bedroom2")
sns.lineplot(data=training_set['Car'], label="Car")
```

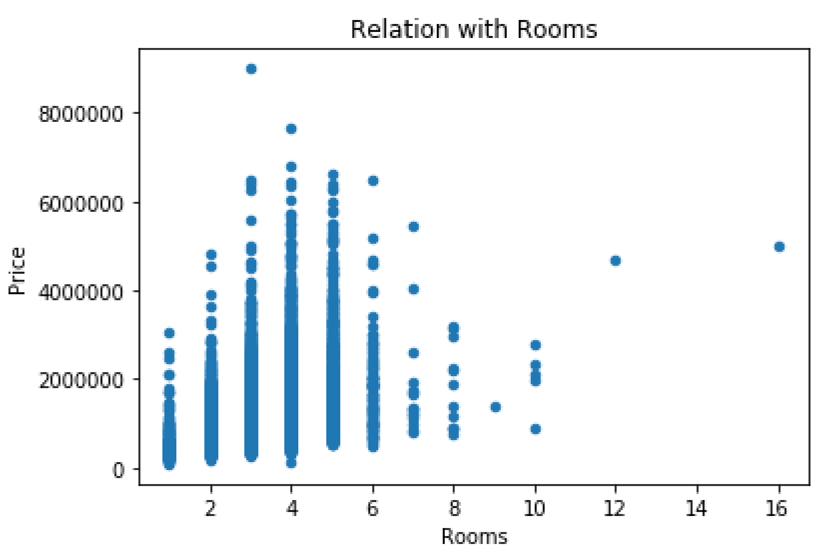
Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x4fd68a9388>



In [110]:
```python
plt.figure(figsize=(14,6))
training_set.plot.scatter(x='Rooms', y='Price')
plt.title('Relation with Rooms')
```

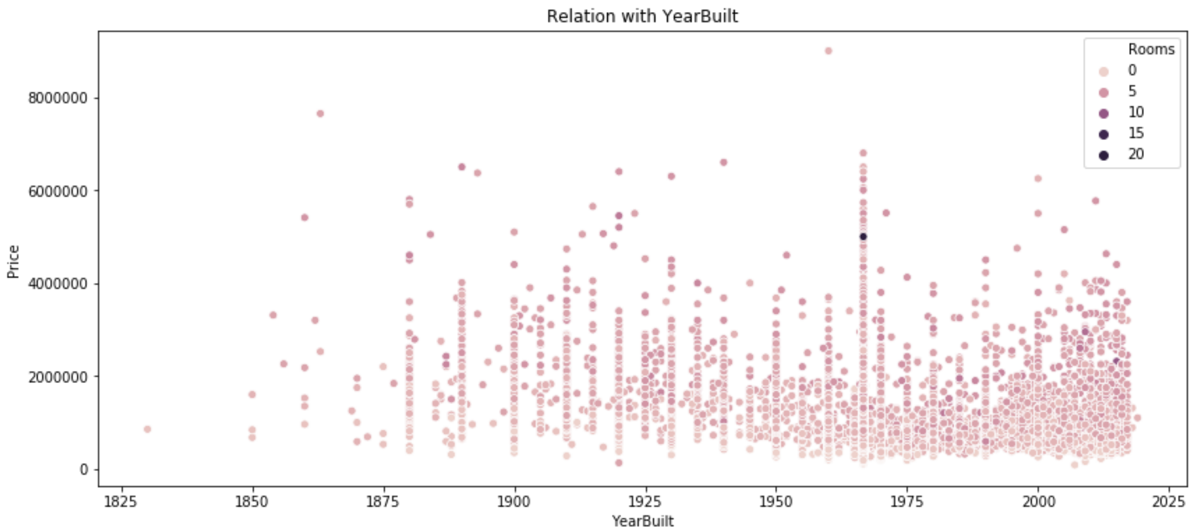
Out[110]: Text(0.5, 1.0, 'Relation with Rooms')

<Figure size 1008x432 with 0 Axes>



In [111]:
```
plt.figure(figsize=(14,6))
sns.scatterplot(x=training_set['YearBuilt'], y=training_set['Price'], hue=trai
ning_set['Rooms'])
plt.title('Relation with YearBuilt')
```

Out[111]: `Text(0.5, 1.0, 'Relation with YearBuilt')`



Identifying features and Target variable

In [86]:
```
input_features = [x for x in training_set.columns if x not in ['Price']]
input_features1 = [x for x in test_set.columns if x not in ['Price']]
```

In [87]:
```
X_train = training_set[input_features].values
y_train = training_set['Price'].values
X_test = test_set[input_features].values
y_test = test_set['Price'].values
```

LinearRegression

In [88]:
```
lr = LinearRegression()
lr_model = lr.fit(X_train,y_train)
```

In [89]:
```
y_train_pred = lr_model.predict(X_train)
```

In [98]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_train, y_train_pred)
print("Score using Linear Regression : %f " %(r2))
```

```
Score using Linear Regression : 0.455472
```

In [91]:
```
test_set.isnull().any()
```

Out[91]:
```
Rooms            False
Price            False
Distance         False
Postcode         False
Bedroom2         False
Bathroom         False
Car              False
BuildingArea     False
YearBuilt        False
Propertycount    False
Landsize_log     False
dtype: bool
```

In [92]:
```
y_test_pred = lr_model.predict(X_test)
```

In [96]:
```
r2 = r2_score(y_test, y_test_pred)
print("Score using Linear Regression : %f " %(r2))
```

```
Score using Linear Regression : 0.439035
```

DecisionTreeRegressor

In [97]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor

def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=
0)
    model.fit(X_train,y_train)
    preds = model.predict(X_train)
    r2 = r2_score(y_train, preds)
    print("Score using DecisionTreeRegressor : %f " %(r2))
    mae = mean_absolute_error(y_train, preds)
    return(mae)

for max_leaf_nodes in [5, 50, 500, 5000]:
    my_mae = get_mae(max_leaf_nodes, X_train,X_test,y_train,y_test)
    print("Max leaf nodes: %d  \t\t Mean Absolute Error:  %d" %(max_leaf_nodes
, my_mae))
```

```
Score using DecisionTreeRegressor : 0.378862
Max leaf nodes: 5                Mean Absolute Error:  354667
Score using DecisionTreeRegressor : 0.661212
Max leaf nodes: 50               Mean Absolute Error:  237270
Score using DecisionTreeRegressor : 0.828661
Max leaf nodes: 500              Mean Absolute Error:  177417
Score using DecisionTreeRegressor : 0.939374
Max leaf nodes: 5000             Mean Absolute Error:  81559
```

In [99]:
```
    model = DecisionTreeRegressor(max_leaf_nodes=500, random_state=0)
    model.fit(X_train,y_train)
    preds_val = model.predict(X_test)
    r2 = r2_score(y_test, preds_val)
    print("Score using DecisionTreeRegressor : %f " %(r2))
    mae = mean_absolute_error(y_test, preds_val)
    print(" Mean Absolute Error:  %d" %(mae))
```

```
Score using DecisionTreeRegressor : 0.614988
 Mean Absolute Error:  235248
```

GradientBoostingRegressor

In [100]:
```
from sklearn.ensemble import GradientBoostingRegressor
gbrt = GradientBoostingRegressor(max_depth=4, n_estimators=300, learning_rate=
0.1, random_state=42)
gbrt.fit(X_train, y_train)
y_pred_gbrt = gbrt.predict(X_train)
```

In [101]:
```
r2 = r2_score(y_train, y_pred_gbrt)
print("Score using GradientBoostingRegressor : %f " %(r2))
```

```
Score using GradientBoostingRegressor : 0.810393
```

In [102]:
```
y_test_pred_gbrt = gbrt.predict(X_test)
```

In [103]:
```
r2 = r2_score(y_test, y_test_pred_gbrt)
print("Score using GradientBoostingRegressor : %f " %(r2))
print("Mean Absolute Error: " + str(mean_absolute_error( y_test_pred_gbrt, y_t
est)))
```

```
Score using GradientBoostingRegressor : 0.739055
Mean Absolute Error: 203785.65279587658
```

Finetune model

In [48]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'max_depth':[6,7,8],
     'n_estimators':[300, 350],
     'learning_rate':[0.09, 0.1, 0.11, 0.12]} ]
grd_gbr_model = GradientBoostingRegressor(random_state=15)
grid_search = GridSearchCV(grd_gbr_model, param_grid, cv=3,
                           scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

Out[48]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.
1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.
0,
                                                 n_estimators=100,
                                                 n_iter_no_change=None,
                                                 presort='auto',
                                                 random_state=15, subsample=
1.0,
                                                 tol=0.0001,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=Fals
e),
             iid='warn', n_jobs=None,
             param_grid=[{'learning_rate': [0.09, 0.1, 0.11, 0.12],
                          'max_depth': [6, 7, 8], 'n_estimators': [300, 35
0]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [49]:
```
grid_search.best_params_
```

Out[49]:
```
{'learning_rate': 0.09, 'max_depth': 6, 'n_estimators': 300}
```

In [50]:
```
best_y_pred_gbrt = grid_search.best_estimator_.predict(X_train)
```

In [52]:
```
r2 = r2_score(y_train, best_y_pred_gbrt)
print(r2)
```

```
0.867960991717767
```

In [53]:
```
best_y_test_pred_gbrt = grid_search.best_estimator_.predict(X_test)
```

In [55]:
```
r2 = r2_score(y_test, best_y_test_pred_gbrt)
print(r2)
```

```
0.7395375078499166
```

Use Log transformed Y - Variable to train

In [38]:
```
y_train_log = np.log(y_train)
```

In [45]:
```
y_test_log = np.log(y_test)
```

In [41]:
```
from sklearn.model_selection import GridSearchCV
param_grid = [
    {'max_depth':[6,7],
     'n_estimators':[300],
     'learning_rate':[0.1, 0.11]} ]
grd_gbr_model = GradientBoostingRegressor(random_state=15)
grid_search = GridSearchCV(grd_gbr_model, param_grid, cv=3,
                           scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train_log)
```

Out[41]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.
1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.
0,
                                                 n_estimators=100,
                                                 n_iter_no_change=None,
                                                 presort='auto',
                                                 random_state=15, subsample=
1.0,
                                                 tol=0.0001,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=Fals
e),
             iid='warn', n_jobs=None,
             param_grid=[{'learning_rate': [0.1, 0.11], 'max_depth': [6, 7],
                          'n_estimators': [300]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [42]:
```
grid_search.best_params_
```

Out[42]:
```
{'learning_rate': 0.11, 'max_depth': 6, 'n_estimators': 300}
```

In [43]:
```python
best_y_pred_gbrt_log = grid_search.best_estimator_.predict(X_train)
```

In [105]:
```python
r2 = r2_score(y_train_log, best_y_pred_gbrt_log)
print("Score using GradientBoostingRegressor : %f " %(r2))
```

```
Score using GradientBoostingRegressor : 0.882423
```

In [46]:
```python
best_y_test_pred_gbrt_log = grid_search.best_estimator_.predict(X_test)
```

In [106]:
```python
r2 = r2_score(y_test_log, best_y_test_pred_gbrt_log)
print("Score using GradientBoostingRegressor : %f " %(r2))
```

```
Score using GradientBoostingRegressor : 0.775973
```

Changing Hyper-parameters to fix overfitting

In [39]:
```python
from sklearn.ensemble import GradientBoostingRegressor
gbrt1 = GradientBoostingRegressor(max_depth=3, n_estimators=500, learning_rate
=0.4, random_state=45,min_samples_split= 2)
gbrt1.fit(X_train, y_train)
y_pred_gbrt = gbrt1.predict(X_train)
```

In [107]:
```python
r2 = r2_score(y_train, y_pred_gbrt)
print("Score using GradientBoostingRegressor : %f " %(r2))
print("Mean Absolute Error: " + str(mean_absolute_error( y_pred_gbrt, y_train
)))
```

```
Score using GradientBoostingRegressor : 0.810393
Mean Absolute Error: 171878.88766566114
```

In [108]:
```python
y_test_pred_gbrt = gbrt1.predict(X_test)
```

In [109]:
```python
r2 = r2_score(y_test, y_test_pred_gbrt)
print("Score using GradientBoostingRegressor : %f " %(r2))
print("Mean Absolute Error: " + str(mean_absolute_error( y_test_pred_gbrt, y_t
est)))
```

```
Score using GradientBoostingRegressor : 0.717575
Mean Absolute Error: 216135.60407064192
```