AI Computer Assignment 0

October 1, 2019

sajjad p. savoij 810195517

1 part A : data representation

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1.1 read data from csv file

To do so pandas **read_csv** is used , this attribute returns data as **pandas dataframe**

[2]:
```
data = pd.read_csv('houses.csv')
```

1.2 describe and explore data

In this part diversity of df attributes are used to describe features of data provided in houses.csv - df.head() - df.describe() - df.count() - df.info()

[3]:
```
data.head()
```

[3]:
```
   Id  MSSubClass  LotArea LotConfig  OverallQual  LotFrontage Neighborhood  \
0   1          60     8450    Inside            7         65.0      CollgCr
1   2          20     9600       FR2            6         80.0      Veenker
2   3          60    11250    Inside            7         68.0      CollgCr
3   4          70     9550    Corner            7         60.0      Crawfor
4   5          60    14260       FR2            8         84.0      NoRidge

   OverallCond  BedroomAbvGr  TotRmsAbvGrd  TotalBsmtSF  YearBuilt  SalePrice
0            5             3             8          856       2003      208.5
1            8             3             6         1262       1976      181.5
2            5             3             6          920       2001      223.5
3            5             3             7          756       1915      140.0
4            5             4             9         1145       2000      250.0
```

[4]:

```
data.describe()
```

[4]:

```
                Id    MSSubClass       LotArea  OverallQual  LotFrontage  \
count  1134.000000   1134.000000   1134.000000  1134.000000    937.00000
mean    622.062610     54.056437   9487.280423     6.065256     68.40555
std     359.623823     38.760477   3866.279692     1.294012     20.13204
min       1.000000     20.000000   1300.000000     2.000000     21.00000
25%     310.250000     20.000000   7508.750000     5.000000     59.00000
50%     623.500000     50.000000   9246.500000     6.000000     70.00000
75%     932.750000     60.000000  11250.000000     7.000000     80.00000
max    1243.000000    180.000000  39104.000000    10.000000    134.00000

       OverallCond  BedroomAbvGr  TotRmsAbvGrd  TotalBsmtSF    YearBuilt  \
count  1134.000000   1134.000000   1134.000000  1134.000000  1134.000000
mean      5.551146      2.828924      6.354497  1032.037037  1972.981481
std       1.015560      0.734241      1.441257   385.301916    28.432646
min       3.000000      1.000000      3.000000     0.000000  1885.000000
25%       5.000000      2.000000      5.000000   796.000000  1955.000000
50%       5.000000      3.000000      6.000000   990.000000  1975.000000
75%       6.000000      3.000000      7.000000  1262.000000  2001.000000
max       8.000000      5.000000     11.000000  2223.000000  2009.000000

         SalePrice
count  1134.000000
mean    174.783949
std      65.428985
min      34.900000
25%     129.925000
50%     161.875000
75%     207.500000
max     415.298000
```

[5]:

```
data.count()
```

[5]:

```
Id              1134
MSSubClass      1134
LotArea         1134
LotConfig       1134
OverallQual     1134
LotFrontage      937
Neighborhood    1134
OverallCond     1134
BedroomAbvGr    1134
TotRmsAbvGrd    1134
TotalBsmtSF     1134
YearBuilt       1134
SalePrice       1134
```

```
dtype: int64
```

[6]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1134 entries, 0 to 1133
Data columns (total 13 columns):
Id              1134 non-null int64
MSSubClass      1134 non-null int64
LotArea         1134 non-null int64
LotConfig       1134 non-null object
OverallQual     1134 non-null int64
LotFrontage     937 non-null float64
Neighborhood    1134 non-null object
OverallCond     1134 non-null int64
BedroomAbvGr    1134 non-null int64
TotRmsAbvGrd    1134 non-null int64
TotalBsmtSF     1134 non-null int64
YearBuilt       1134 non-null int64
SalePrice       1134 non-null float64
dtypes: float64(2), int64(9), object(2)
memory usage: 115.3+ KB
```

1.3 delete non-numeric fields

There are diffrent ways to handle non-numeric fields in data-frames , one obvious yet remarkable approach would be to neglect them to do so pandas data frame **drop** attribute is used

[7]:
```python
num_data = data.drop(['LotConfig','Neighborhood'] , axis = 1)
num_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1134 entries, 0 to 1133
Data columns (total 11 columns):
Id              1134 non-null int64
MSSubClass      1134 non-null int64
LotArea         1134 non-null int64
OverallQual     1134 non-null int64
LotFrontage     937 non-null float64
OverallCond     1134 non-null int64
BedroomAbvGr    1134 non-null int64
TotRmsAbvGrd    1134 non-null int64
TotalBsmtSF     1134 non-null int64
YearBuilt       1134 non-null int64
SalePrice       1134 non-null float64
dtypes: float64(2), int64(9)
memory usage: 97.6 KB
```

1.4 replace missing with mean of each column

To perform statistical analysis on data , all data cells should be filled with computable numerical values. For this matter pandas data-frame **fillna** attribute is used

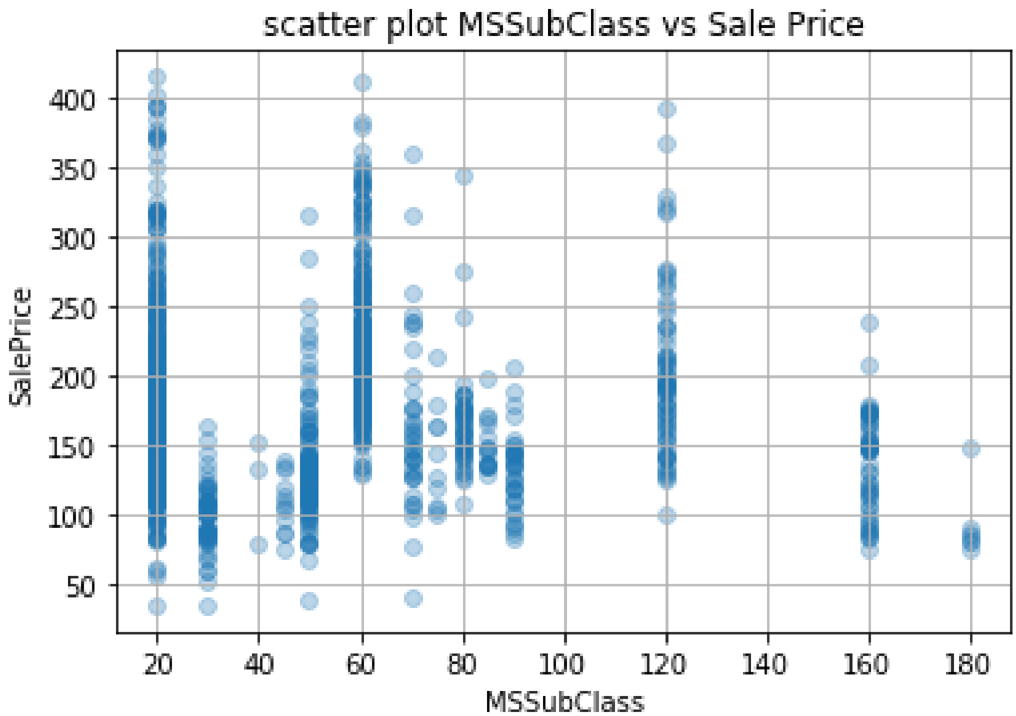
[8]:
```
num_data.fillna(num_data.mean() , inplace=True)
num_data.count()
```

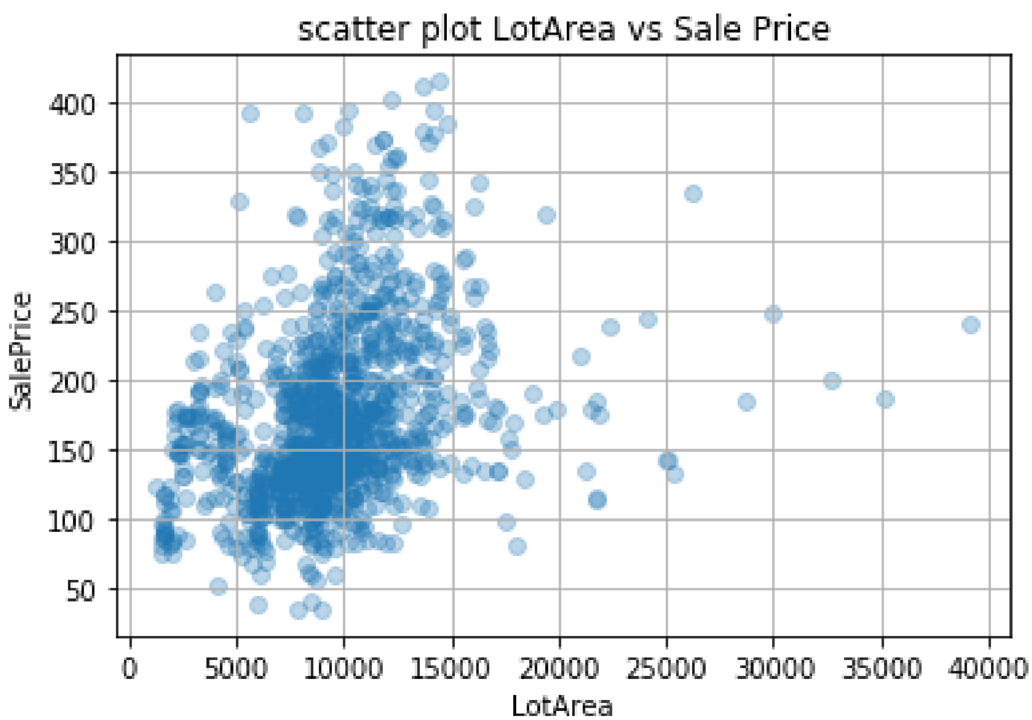
[8]:
```
Id              1134
MSSubClass      1134
LotArea         1134
OverallQual     1134
LotFrontage     1134
OverallCond     1134
BedroomAbvGr    1134
TotRmsAbvGrd    1134
TotalBsmtSF     1134
YearBuilt       1134
SalePrice       1134
dtype: int64
```

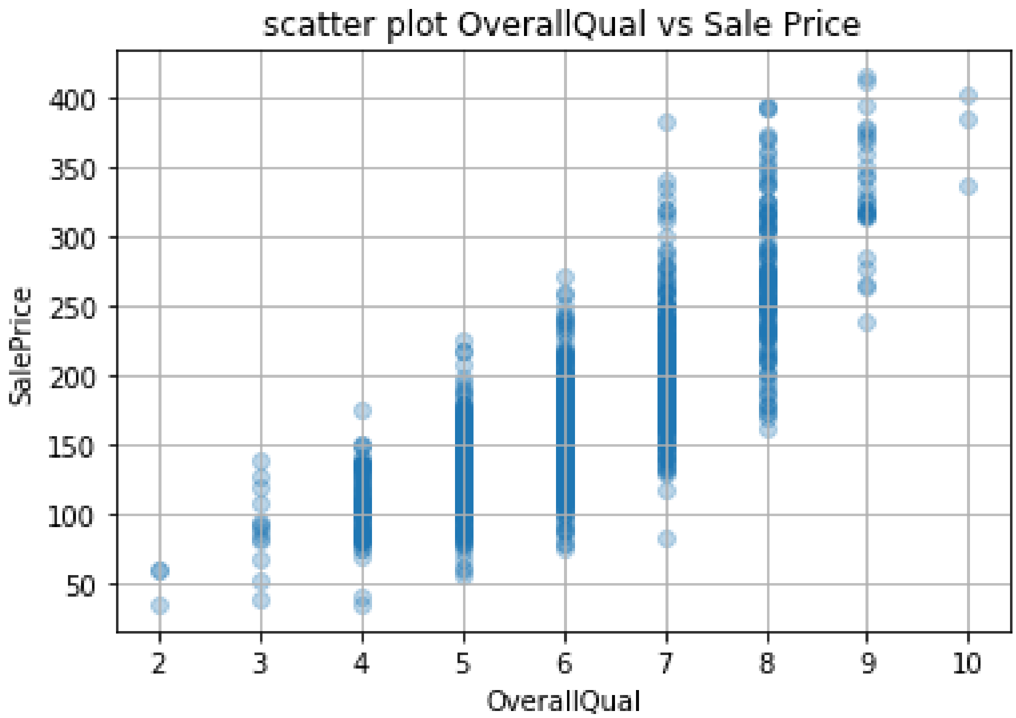
1.5 Data Visualization

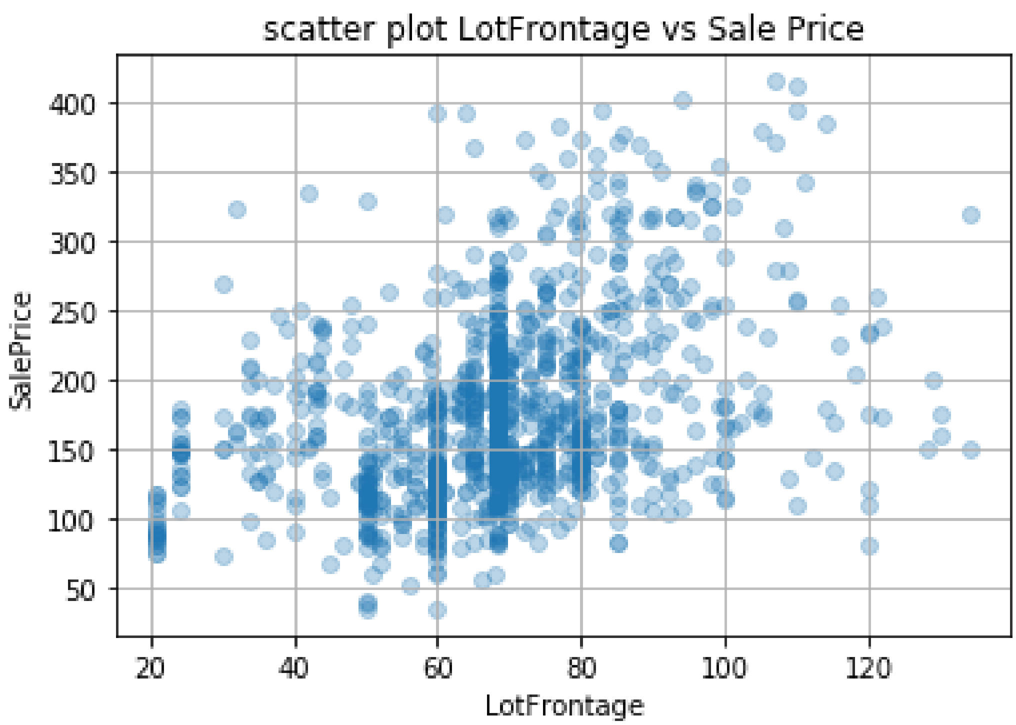
using matplotlib.pyplot.scatter

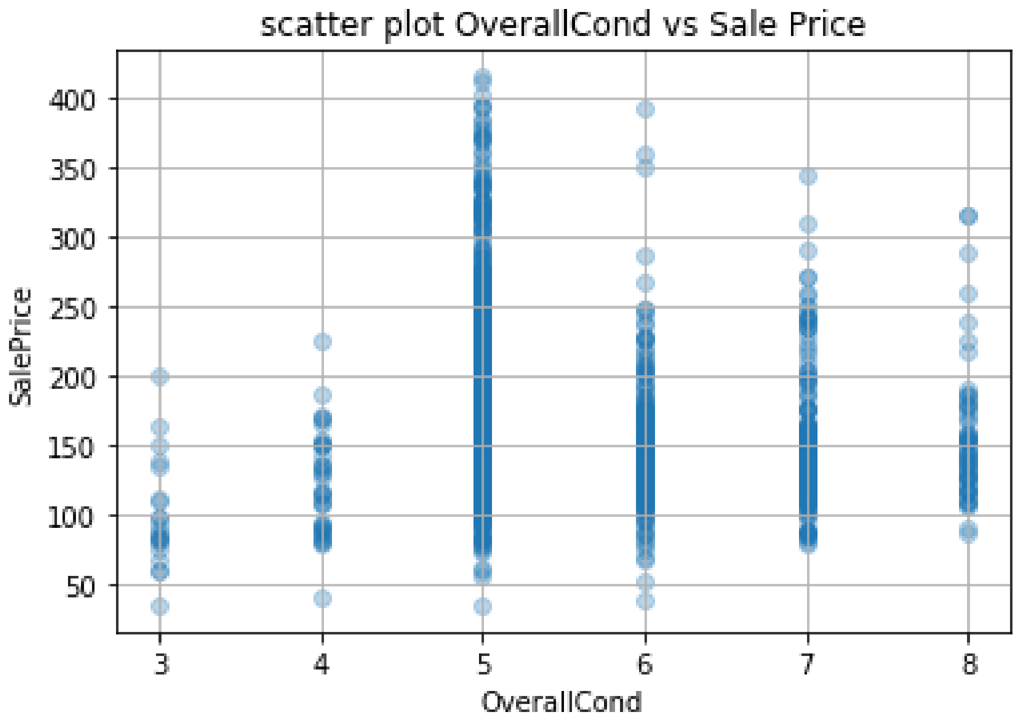
[9]:
```
for col in num_data.columns[1:10] :
    plt.scatter(num_data[col] , num_data.SalePrice , alpha = 0.3)
    plt.xlabel(col)
    plt.ylabel('SalePrice')
    plt.title('scatter plot '+ col + ' vs Sale Price')
    plt.grid()
    plt.show()
```

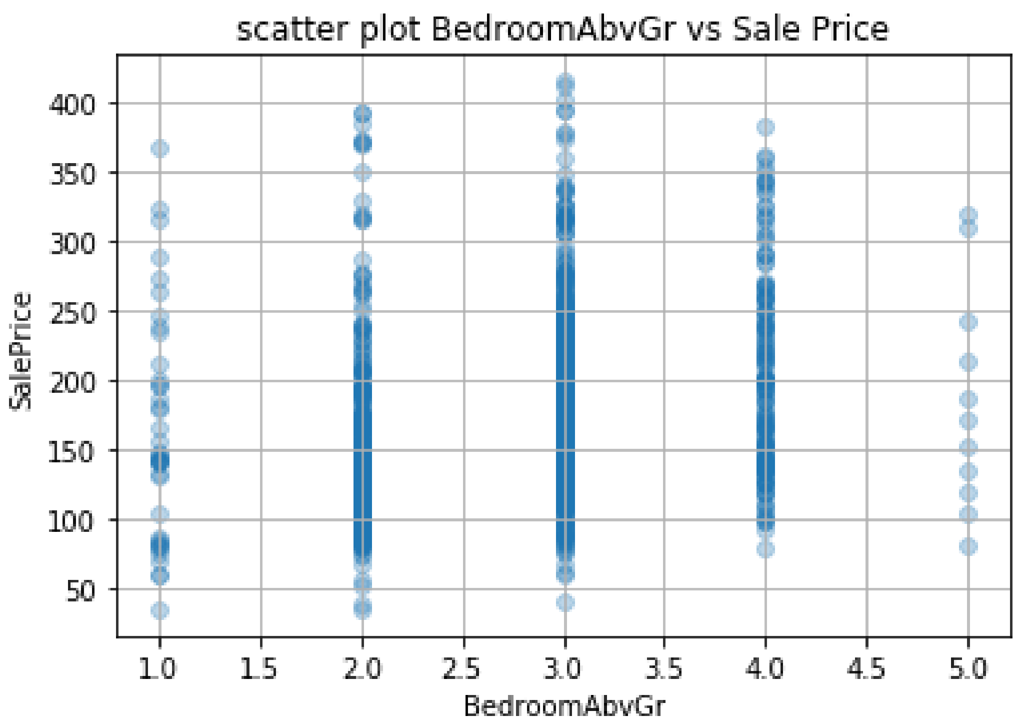

scatter plot MSSubClass vs Sale Price
SalePrice
MSSubClass
400
350
300
250
200
150
100
50
20
40
60
80
100
120
140
160
180

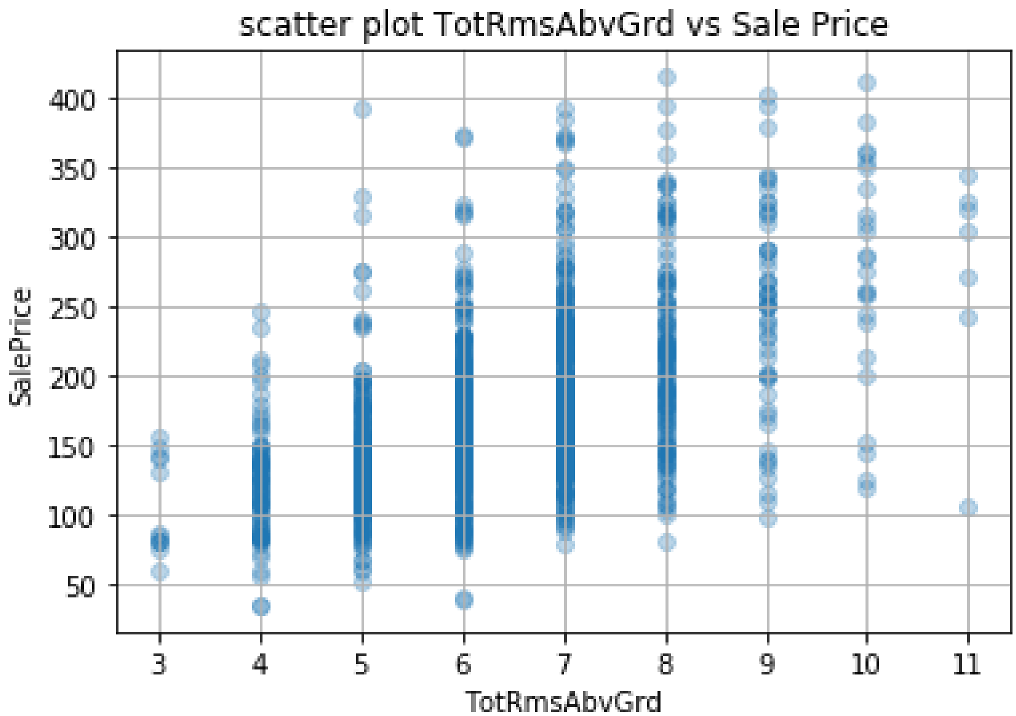

scatter plot LotArea vs Sale Price
SalePrice
LotArea
400
350
300
250
200
150
100
50
0
5000
10000
15000
20000
25000
30000
35000
40000

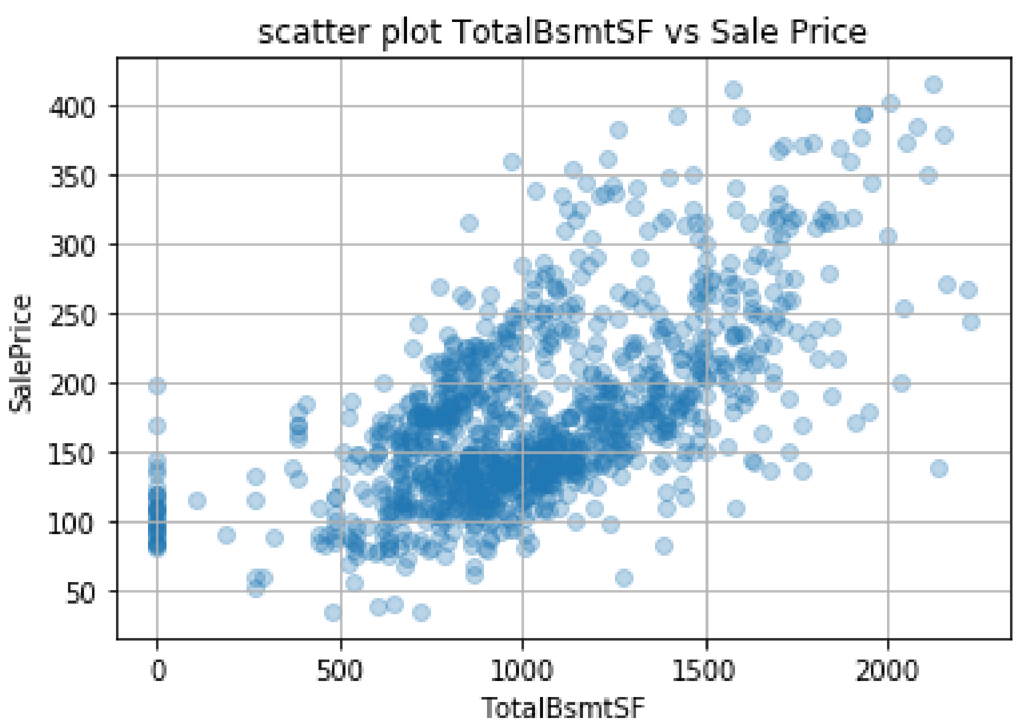

scatter plot OverallQual vs Sale Price
SalePrice
400
350
300
250
200
150
100
50
2
3
4
5
6
7
8
9
10
OverallQual

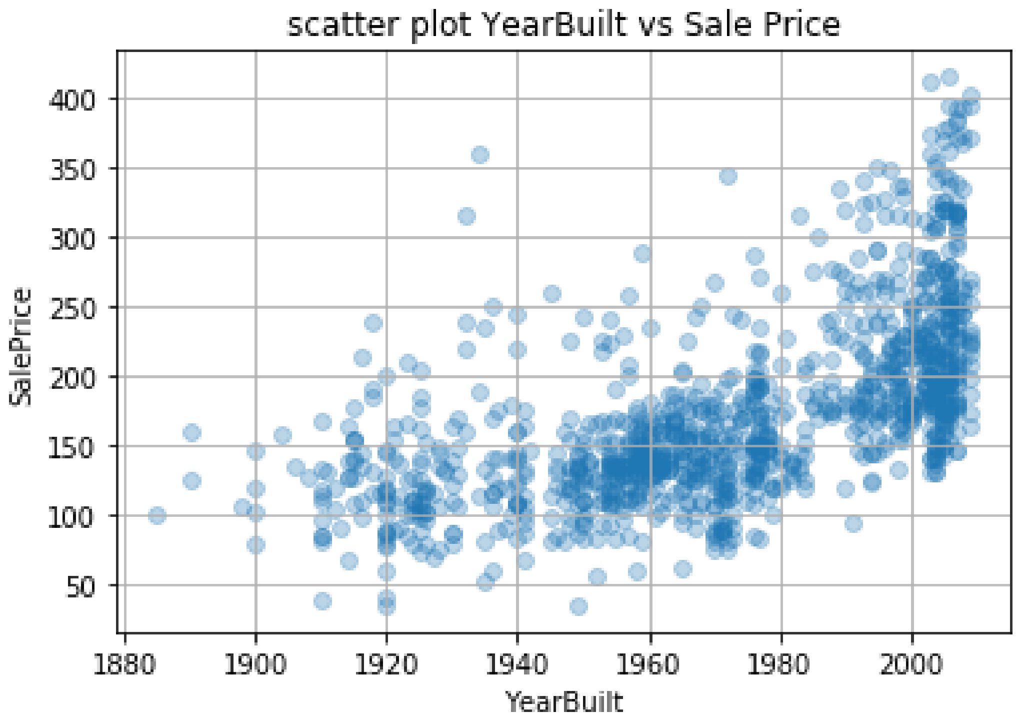

scatter plot LotFrontage vs Sale Price
SalePrice
400
350
300
250
200
150
100
50
20
40
60
80
100
120
LotFrontage


scatter plot OverallCond vs Sale Price
SalePrice
400
350
300
250
200
150
100
50
3
4
5
6
7
8
OverallCond


scatter plot BedroomAbvGr vs Sale Price
SalePrice
400
350
300
250
200
150
100
50
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
BedroomAbvGr


scatter plot TotRmsAbvGrd vs Sale Price
SalePrice
400
350
300
250
200
150
100
50
3
4
5
6
7
8
9
10
11
TotRmsAbvGrd


scatter plot TotalBsmtSF vs Sale Price
SalePrice
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TotalBsmtSF



It appeats tha OverallQual feature shows stronger linear relation with house price

2 part B : Linear Regression

A linear regression model is illustrated as below in general

$$\underline{\hat{y}} = \underline{\theta}\,\underline{x}$$

note that $\hat{y}$, θ and x are n by 1 vectors in general where n is dimension of feature vector

Our task is to find out the weight vector(θ) such that our estimation using linear model would have the least RMSE

hopefully this problem can be uniquely solved using algebric methods called ""**Norm Equation**" as I would use this method insted of random guessing the weights , derivation of norm equation is provided below

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n = \underline{\theta}\,\underline{x}$$

$$J(\theta) = \frac{1}{2m}\,(X\,\theta - Y)^T\,(X\,\theta - Y)$$

$$= \left((X\,\theta)^T - Y^T\right)(X\,\theta - Y) = (X\theta)^T(X\theta) - (X\theta)^T Y - Y^T(X\theta) + Y^T Y = \theta^T X^T X\theta - 2(X\theta)^T Y + Y^T Y$$

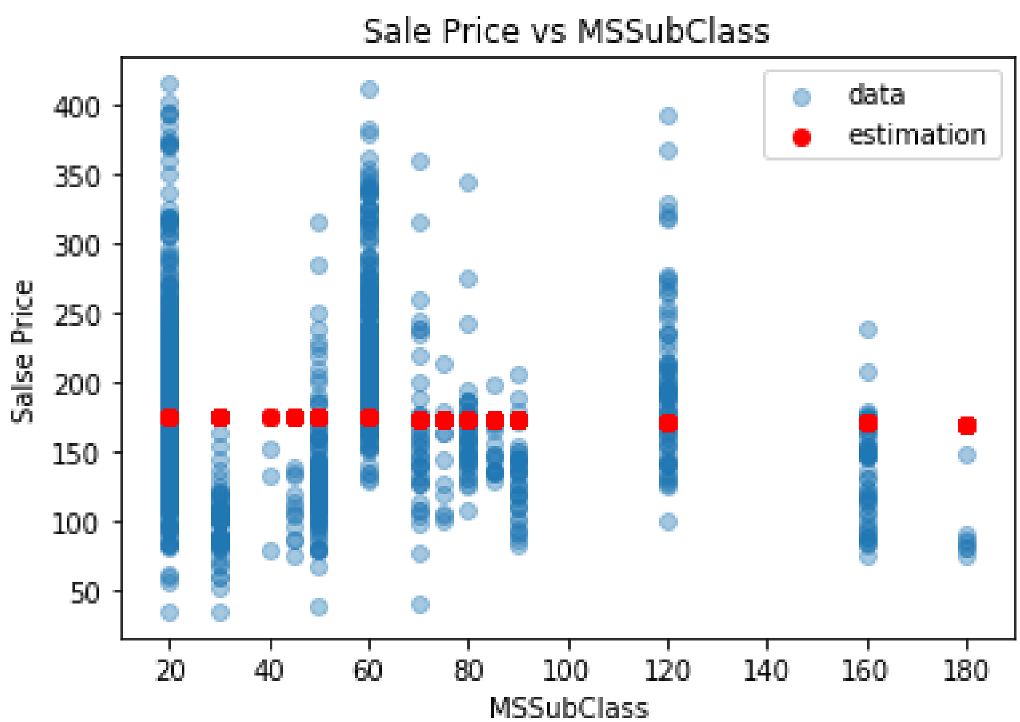
$$\frac{\partial J(\theta)}{\partial \theta} = 2X^T X\theta - 2X^T Y = 0$$

hence , $\hat{\theta} = \left(X^T X\right)^{-1} X^T Y$

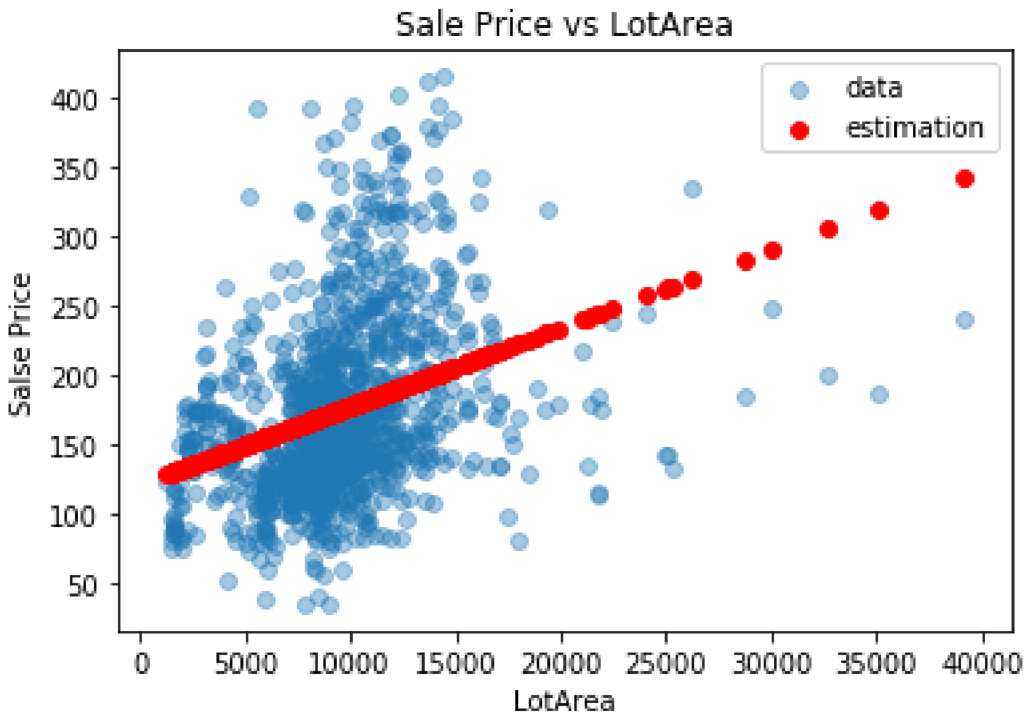
[10]:
```
class LinearReg():
    def __init__(self) :
        self.theta = 0

    def fit(self , X , y):
        """X should be mxn where m is sample size and n is size of dimension"""
        """y should be mx1 where m is sampel size  """
        s = X.shape
        X = np.hstack((np.ones((s[0],1)) , X))
        self.theta = np.dot(np.dot(np.linalg.inv(np.dot(X.T , X)),X.T),y)

    def transform(self , X):
        s = X.shape
        X = np.hstack((np.ones((s[0],1)) , X))
        return np.dot(self.theta.T,X.T).T

    def fit_transform(self , X , y):
        self.fit(X , y)
        return self.transform(X)

    def weights(self):
        return self.theta

def RMSE(x , y) :
    '''x and y should be in size nx1 where n is dimension of feature vector'''
    s = x.shape
    return np.power((np.sum(np.power((x-y),2))/s[0]),0.5)
```

[11]:
```
for col in num_data.columns :
    features = num_data[col].values.reshape(-1,1)
    targets = num_data.SalePrice.values.reshape(-1,1)
    linreg = LinearReg()
    estimation = linreg.fit_transform(features , targets)
    plt.scatter(features , targets , label = 'data' , alpha = 0.4)
    plt.scatter(features , estimation , label = 'estimation' , color = 'red')
    plt.xlabel(col)
    plt.ylabel('Salse Price')
    plt.title('Sale Price vs ' + col)
    plt.legend()
    plt.show()
    print("RMSE : ", RMSE(targets , estimation))
```

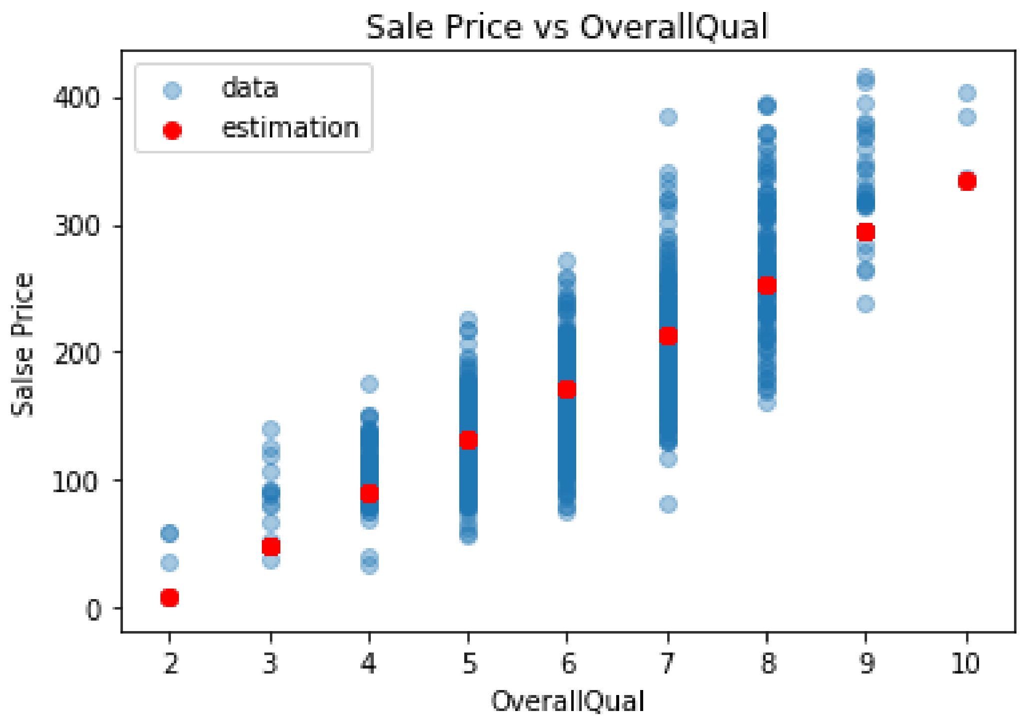
RMSE : 65.34563343158308



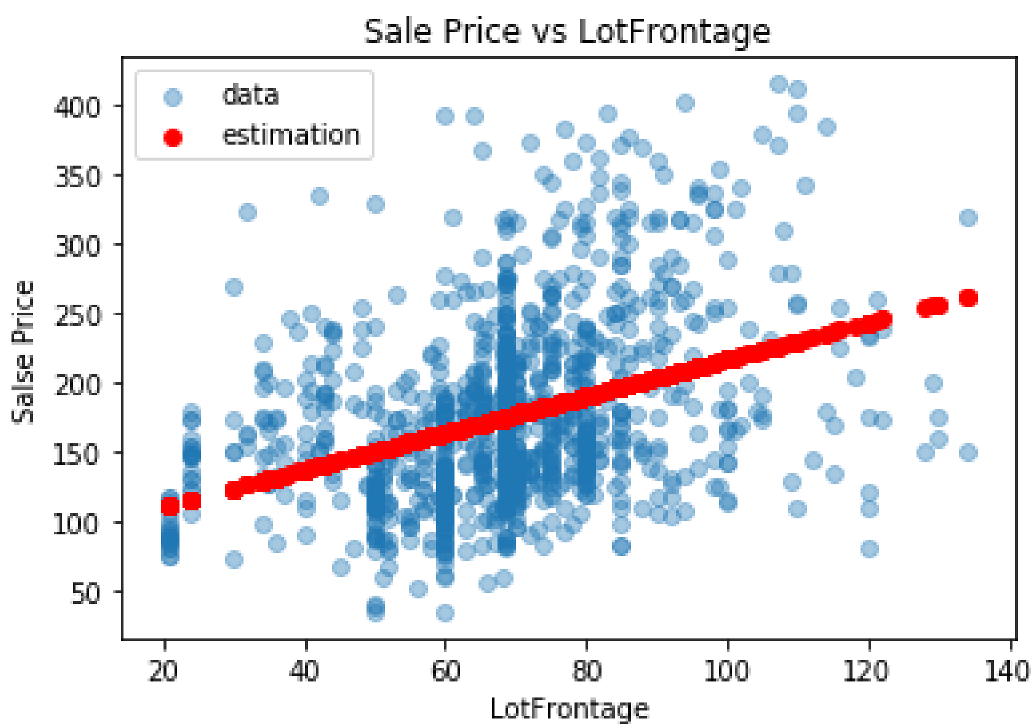
```
RMSE :  65.38398572522601
```



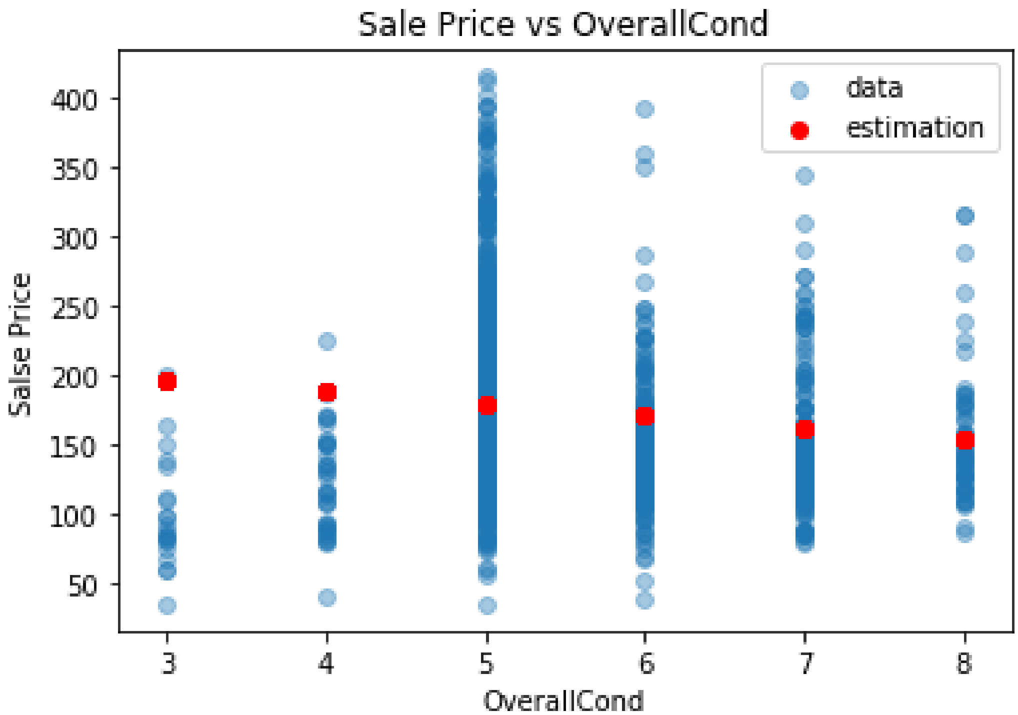

```
RMSE :  61.64965557918095
```



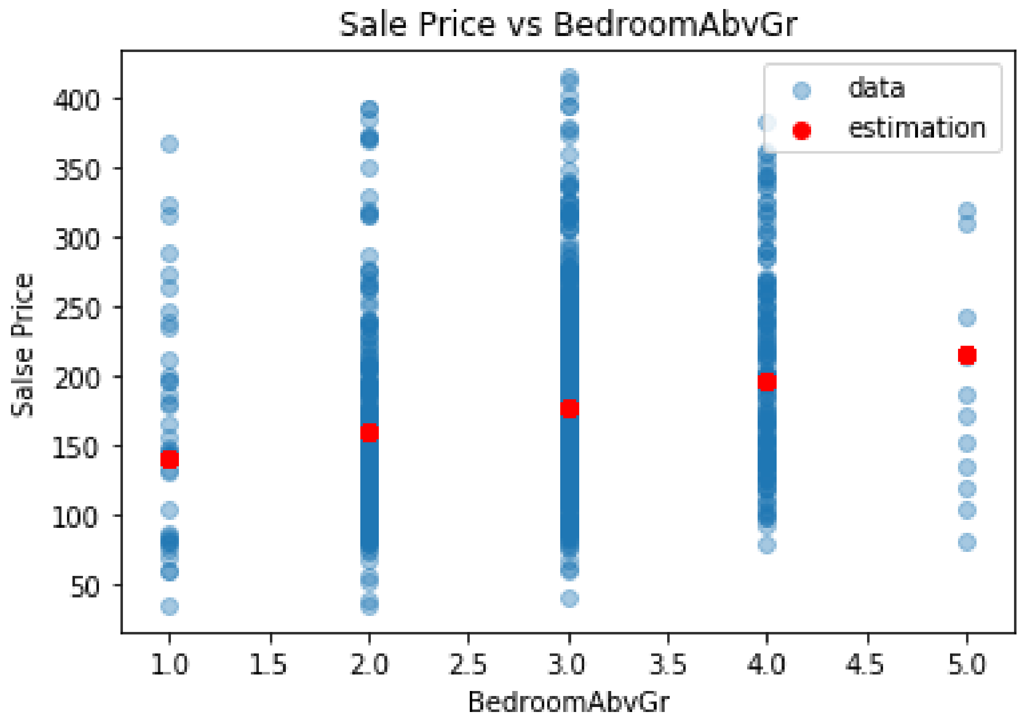
RMSE : 38.54666046317152



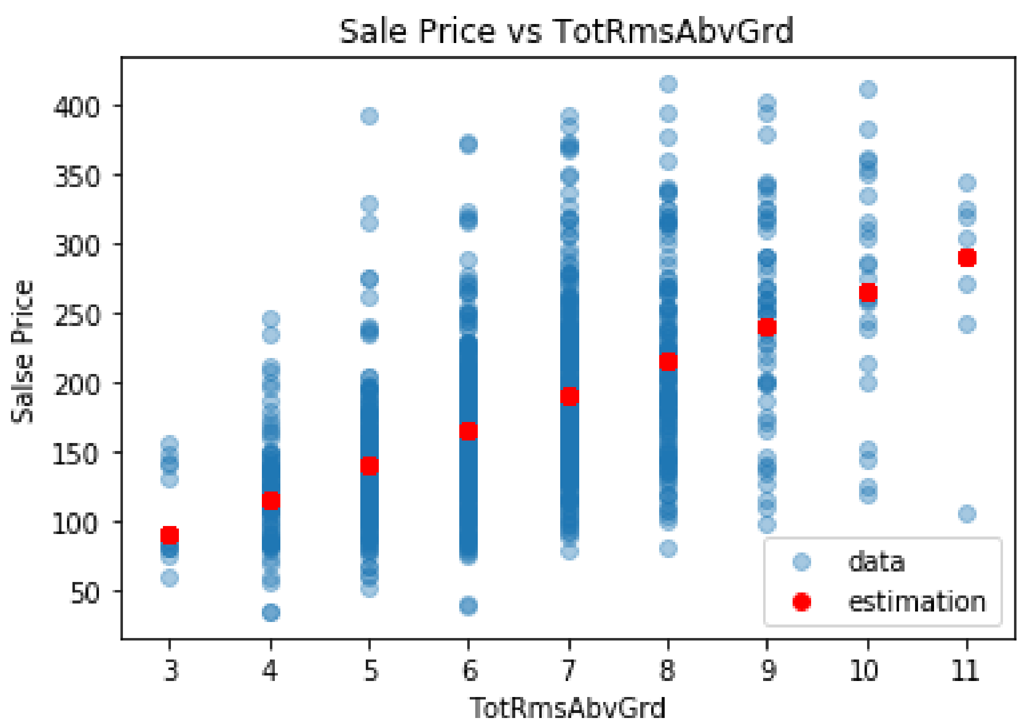
```
RMSE :  60.74279029934669
```



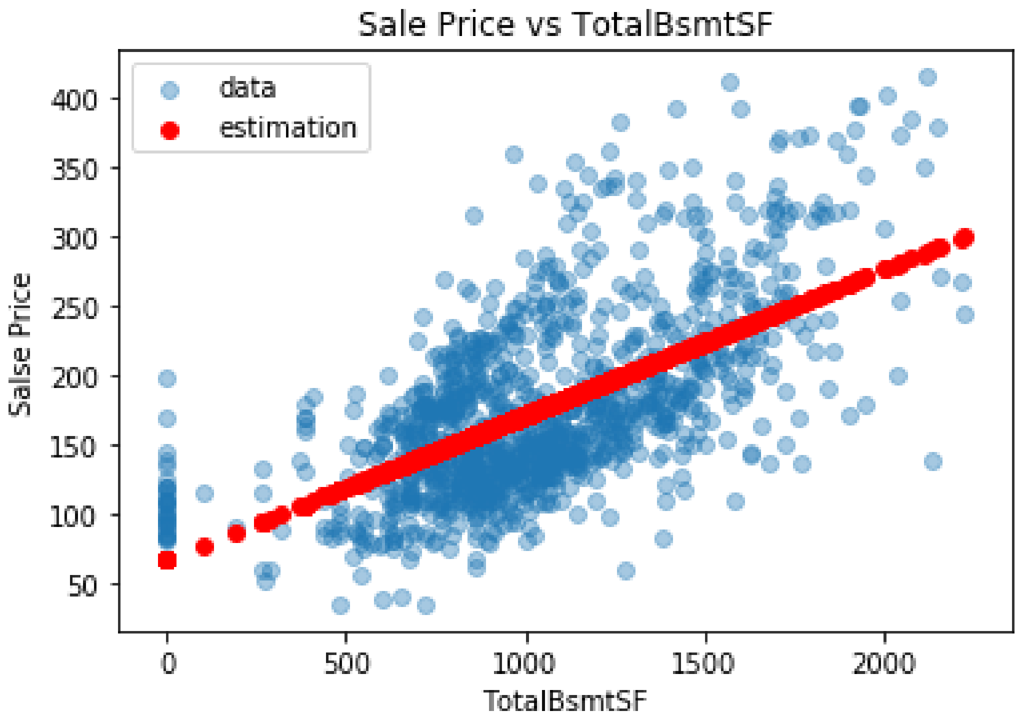

```
RMSE :  64.83939925268781
```



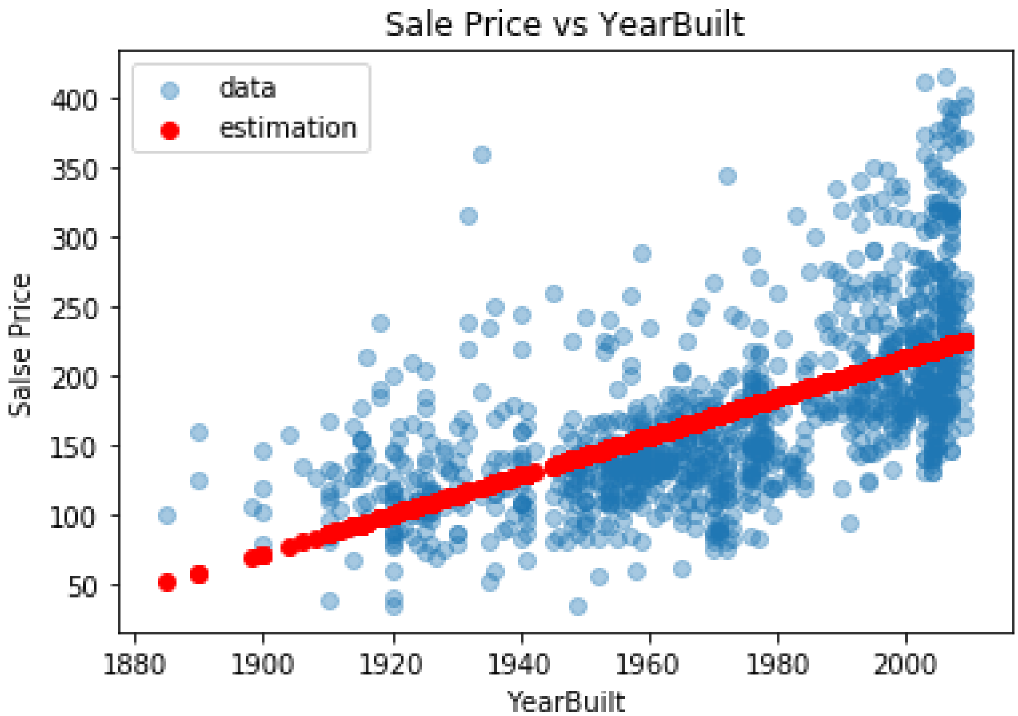

```
RMSE :  63.91833503338879
```



```
RMSE :  54.72247210447784
```




```
RMSE :  51.43001839349524
```




```
RMSE :  51.795108995469 08
```

```
RMSE :  2.2932677893343486e-13
```

which features had linear relation with sale price?

according to RMSE factor of all features , the over all qual is the best feature for single variable linear regression

[12]:
```
features = num_data.OverallQual.values.reshape(-1,1)
targets  = num_data.SalePrice.values.reshape(-1,1)
linreg   = LinearReg()
lin_estimate = linreg.fit_transform(features, targets)
print(linreg.weights())
```

```
[[-72.96290256]
 [ 40.84689292]]
```

thus the b = -72.96 and w1 = 40.84

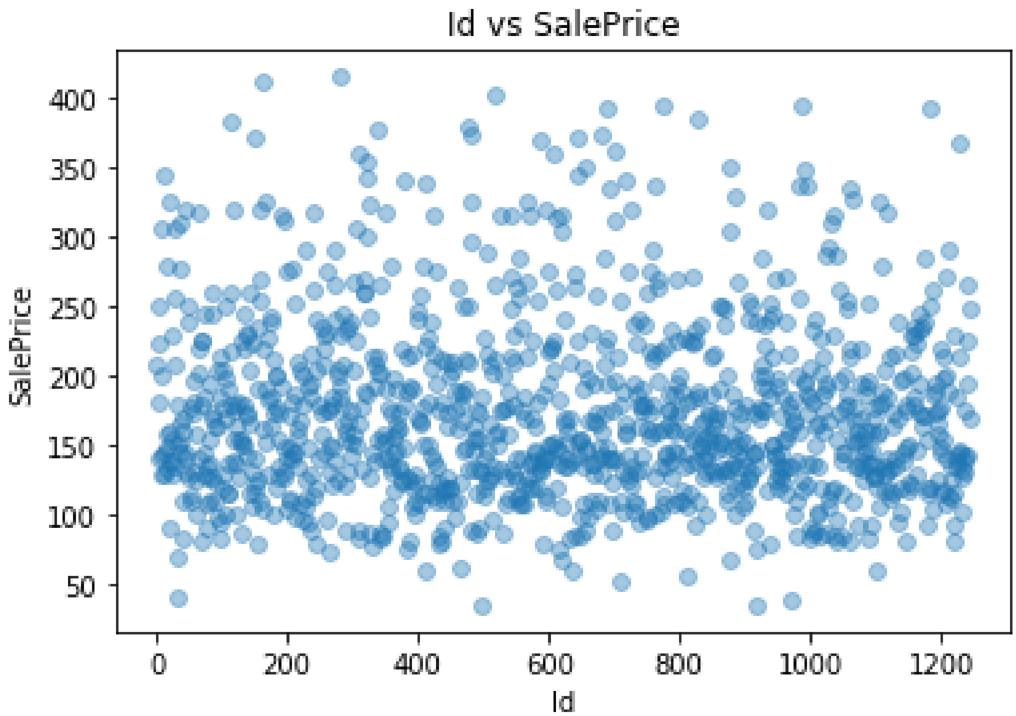
3 part C : done in vectorization.inpy

4 part D : vectorize for-loops in part B

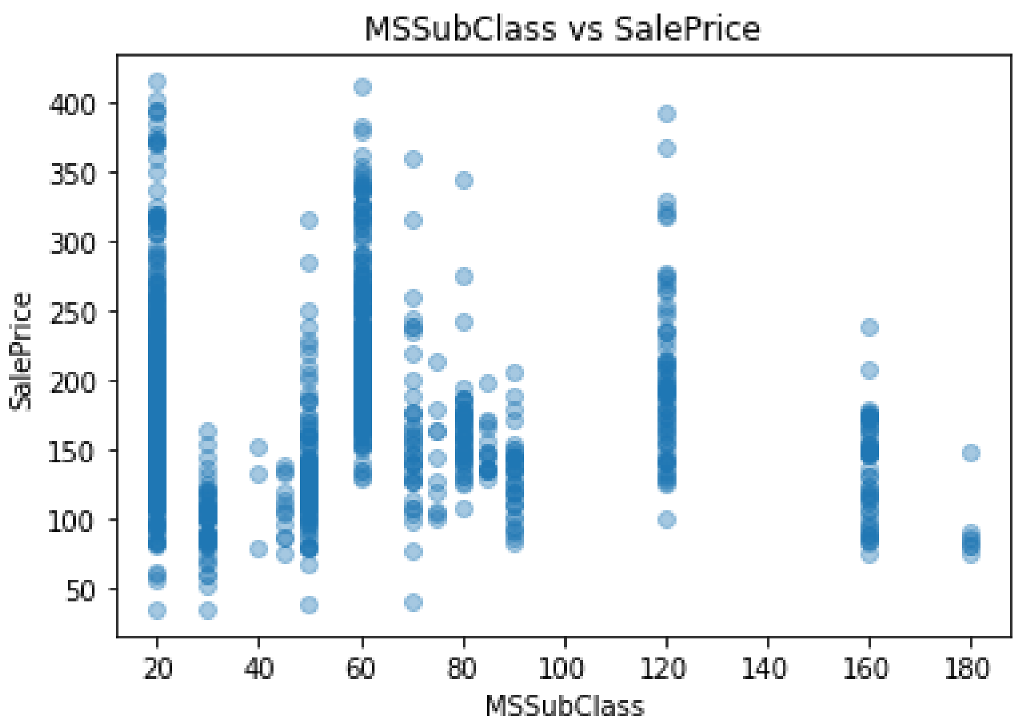
to do so I used "apply" attribute of pandas.dataframe

[13]:
```
def scatter(x , y):
    plt.scatter(x , y , alpha = 0.4)
    plt.xlabel(x.name)
    plt.ylabel(y.name)
    plt.title(x.name + " vs " + y.name)
    plt.show()
```

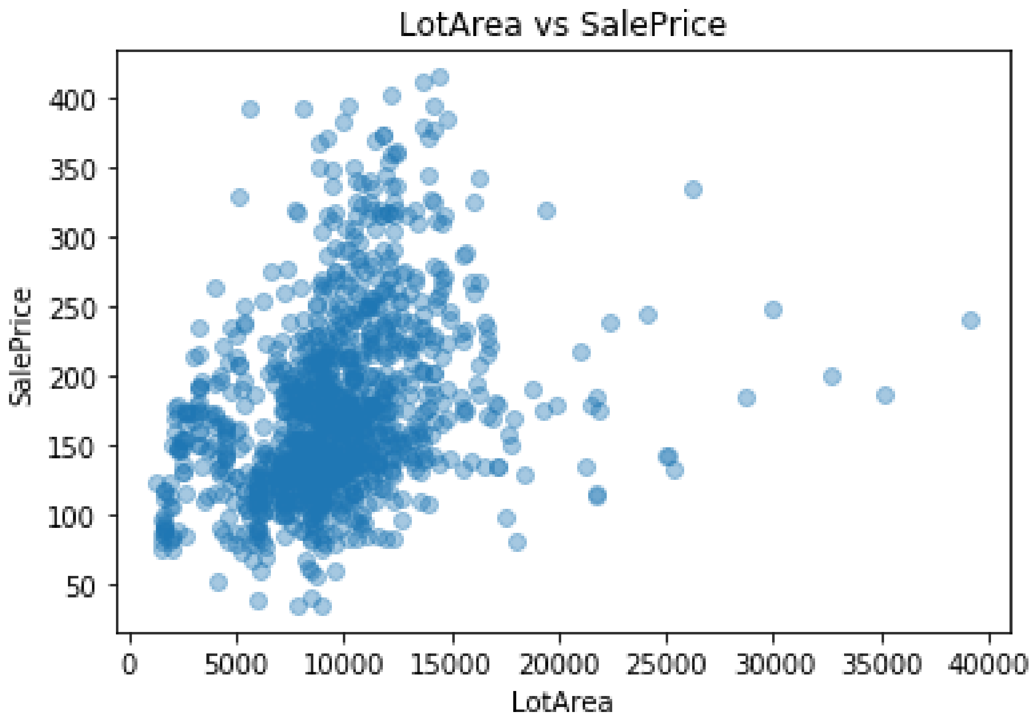
[14]:
```
num_data.apply(lambda x:scatter(x , num_data.SalePrice) , axis=0 )
```

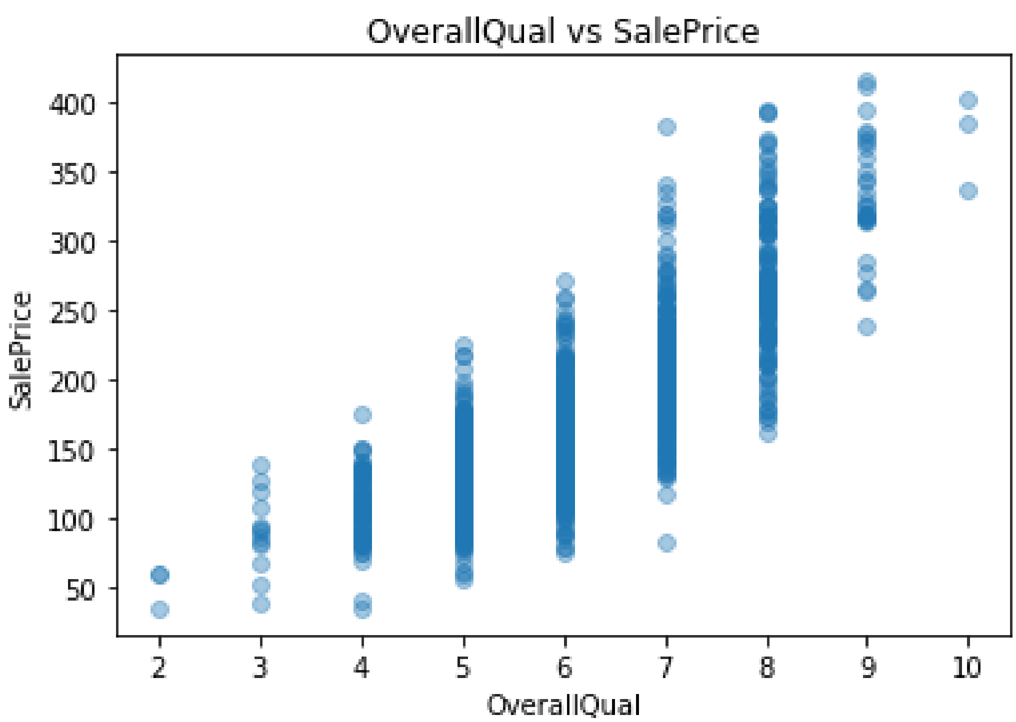


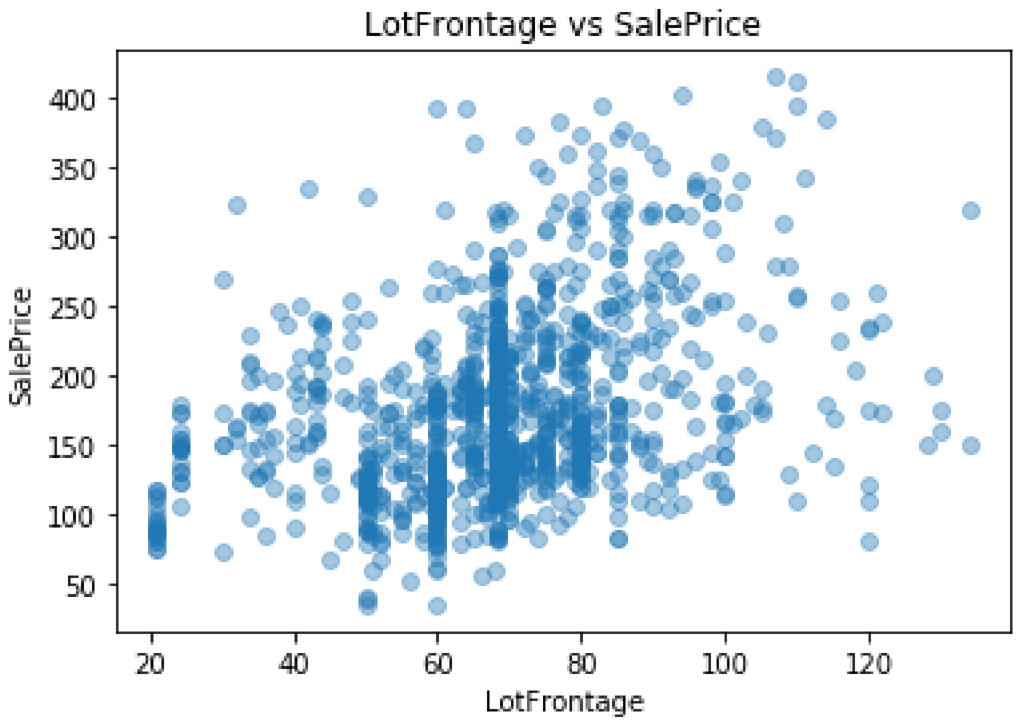
Id vs SalePrice
SalePrice
400
350
300
250
200
150
100
50
0
200
400
600
800
1000
1200
Id

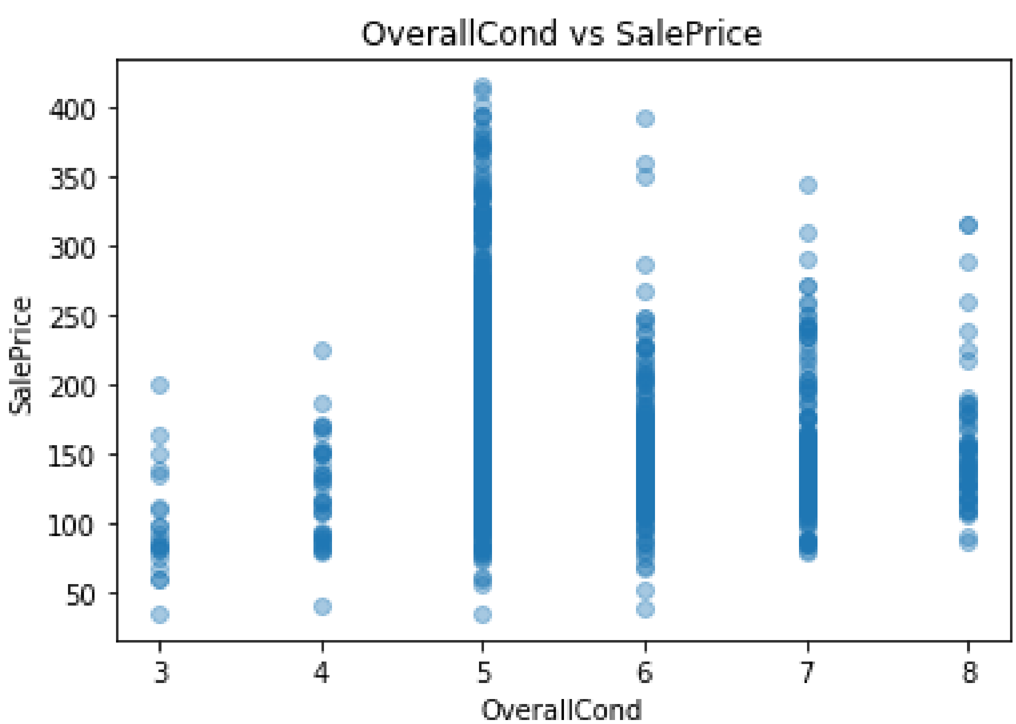


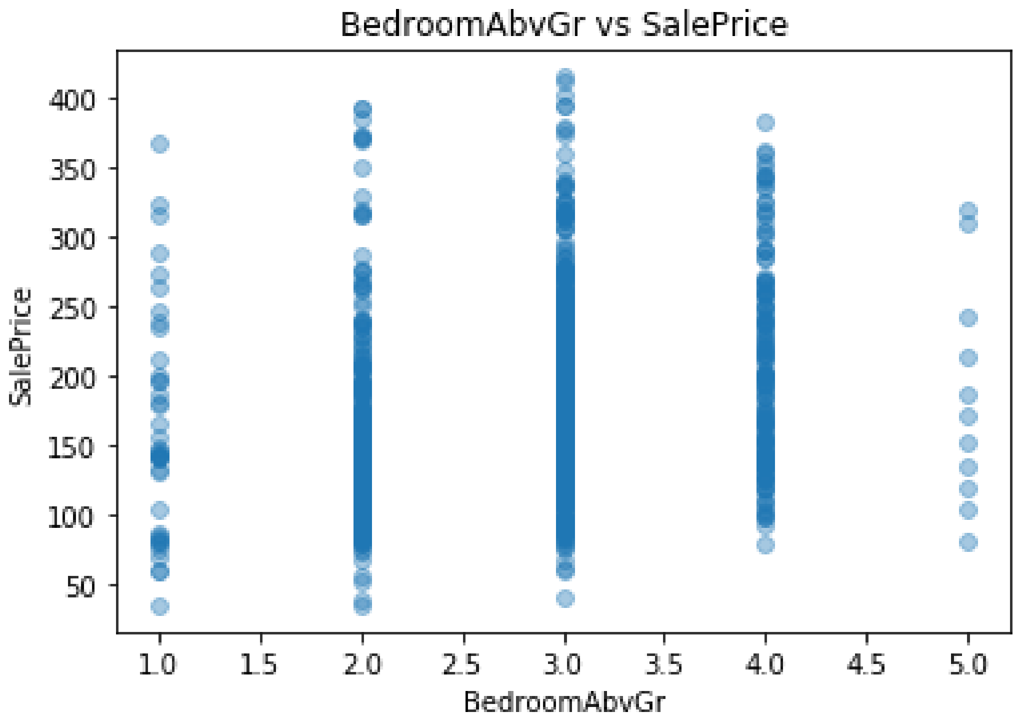
MSSubClass vs SalePrice
SalePrice
400
350
300
250
200
150
100
50
20
40
60
80
100
120
140
160
180
MSSubClass

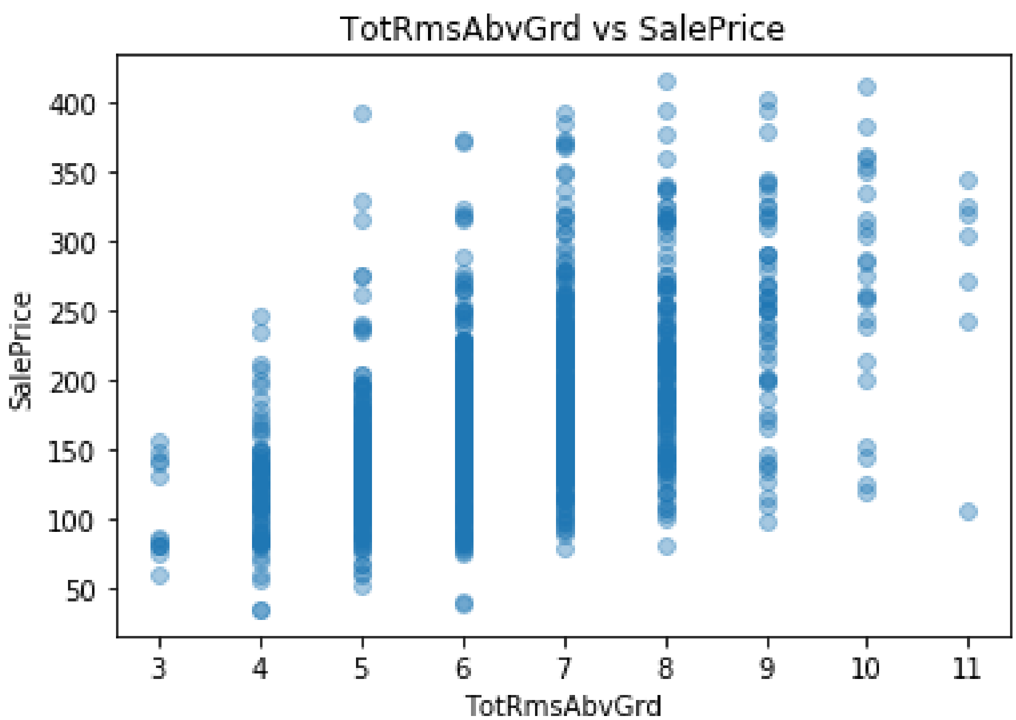

LotArea vs SalePrice
400
350
300
250
200
150
100
50
SalePrice
0
5000
10000
15000
20000
25000
30000
35000
40000
LotArea

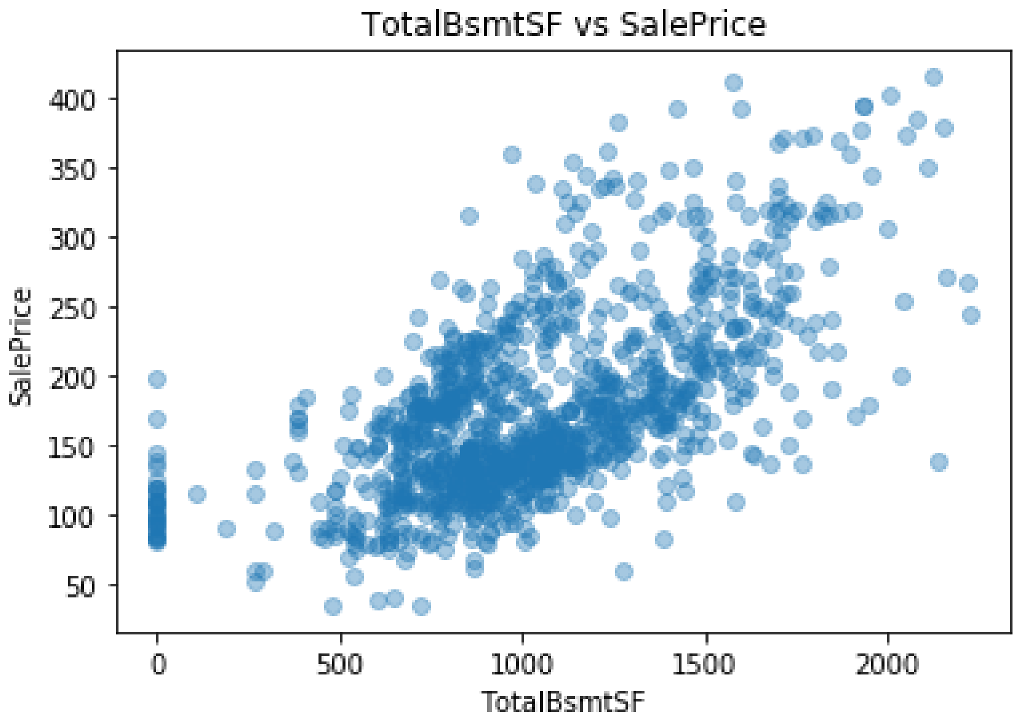

OverallQual vs SalePrice
400
350
300
250
200
150
100
50
SalePrice
2
3
4
5
6
7
8
9
10
OverallQual

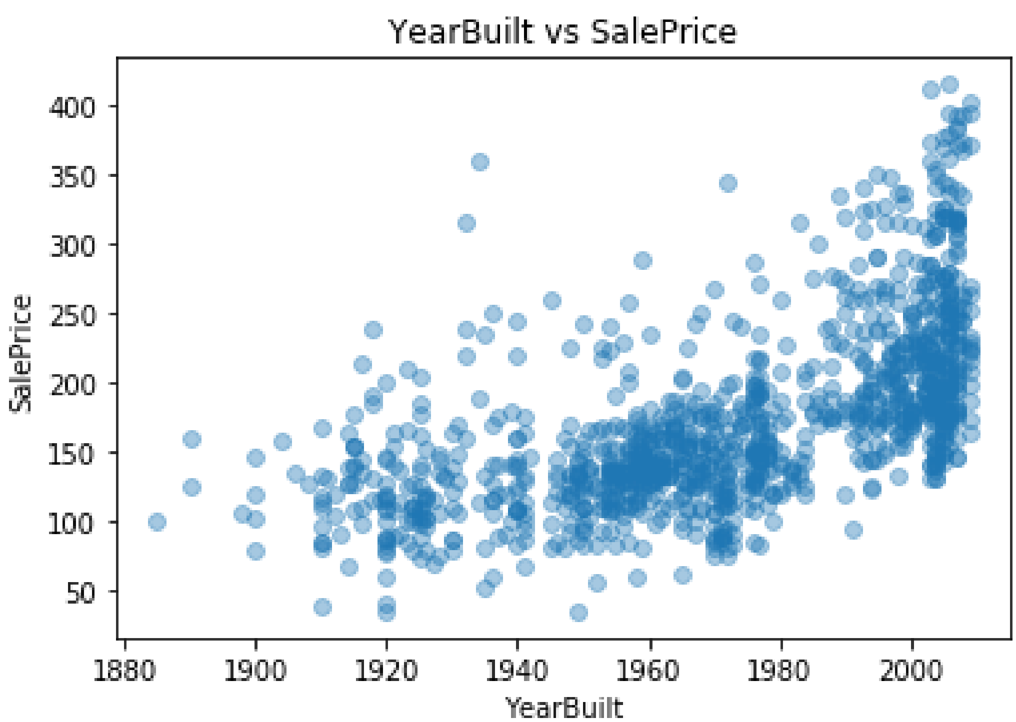

LotFrontage vs SalePrice
SalePrice
LotFrontage
400
350
300
250
200
150
100
50
20
40
60
80
100
120

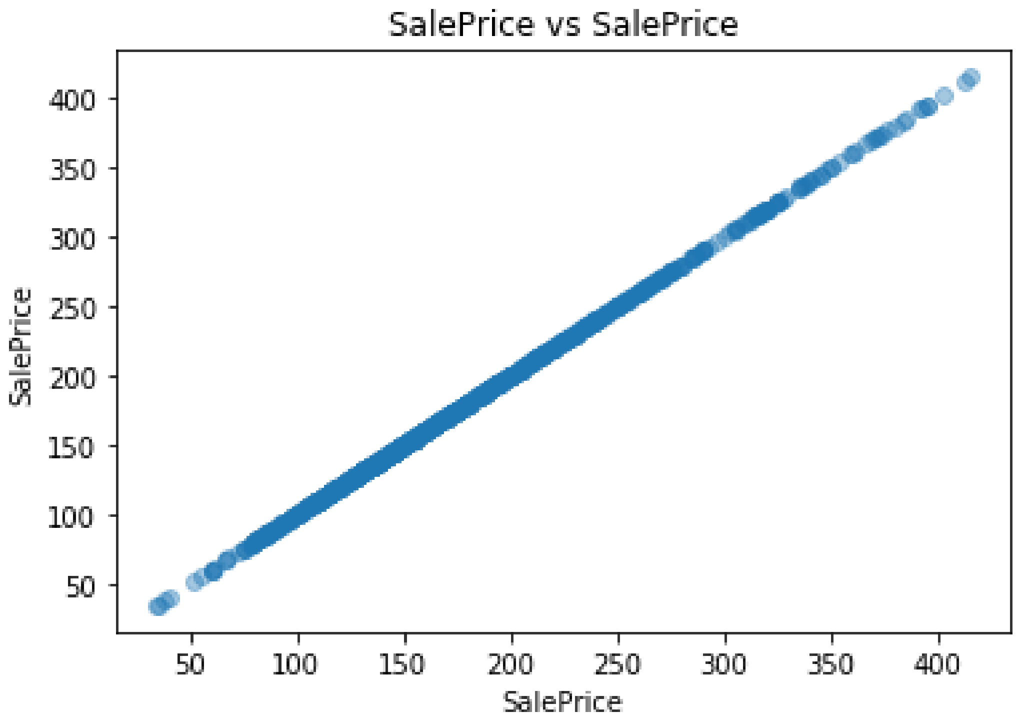

OverallCond vs SalePrice
SalePrice
OverallCond
400
350
300
250
200
150
100
50
3
4
5
6
7
8


BedroomAbvGr vs SalePrice
SalePrice
400
350
300
250
200
150
100
50
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
BedroomAbvGr


TotRmsAbvGrd vs SalePrice
SalePrice
400
350
300
250
200
150
100
50
3
4
5
6
7
8
9
10
11
TotRmsAbvGrd


TotalBsmtSF vs SalePrice
SalePrice
TotalBsmtSF
400
350
300
250
200
150
100
50
0
500
1000
1500
2000


YearBuilt vs SalePrice
SalePrice
YearBuilt
400
350
300
250
200
150
100
50
1880
1900
1920
1940
1960
1980
2000



[14]:
```
Id              None
MSSubClass      None
LotArea         None
OverallQual     None
LotFrontage     None
OverallCond     None
BedroomAbvGr    None
TotRmsAbvGrd    None
TotalBsmtSF     None
YearBuilt       None
SalePrice       None
dtype: object
```

5 part E : k nearest neighbors (KNN) non-parametric method

5.1 Standardized Data

As is explained inorder to knn works , data should be standard. The following formula is used for standardization

$$\hat{x^{(i)}} = \frac{x^{(i)} - min}{max - min}$$

[15]:
```
class KNNReg():
    def __call__(self , samples):
        return self.estimate(samples)

    def __init__(self , df , k = 10):
        self.k = k
        self.prices = df.SalePrice
        self.df = df.drop(['Id', 'SalePrice'] , axis = 1)
        self.min = self.df.mean()
        self.max = self.df.max()
        self.df = self.standardize(self.df)

    def standardize(self , df):
        return (df - self.min)/(self.max - self.min)

    def calculate_estimation(self , sample):
        indices = np.argpartition(np.linalg.norm(self.df.values - sample.values␣
↪, axis = 1)
                                  ,self.k)
        return np.mean(self.prices[indices[:self.k]])

    def estimate(self , samples):
            samples = self.standardize(samples)
            return samples.apply(lambda x:self.calculate_estimation(x) , axis =␣
↪1)
```

[16]:
```
knn = KNNReg(num_data)
```

[17]:
```
samples = num_data[:]
prices = samples.SalePrice
samples = samples.drop(['Id' , 'SalePrice'] , axis = 1)
knn_estimate = knn(samples)
```

[18]:
```
RMSE(prices , knn_estimate)
```

[18]: 25.29561802039879

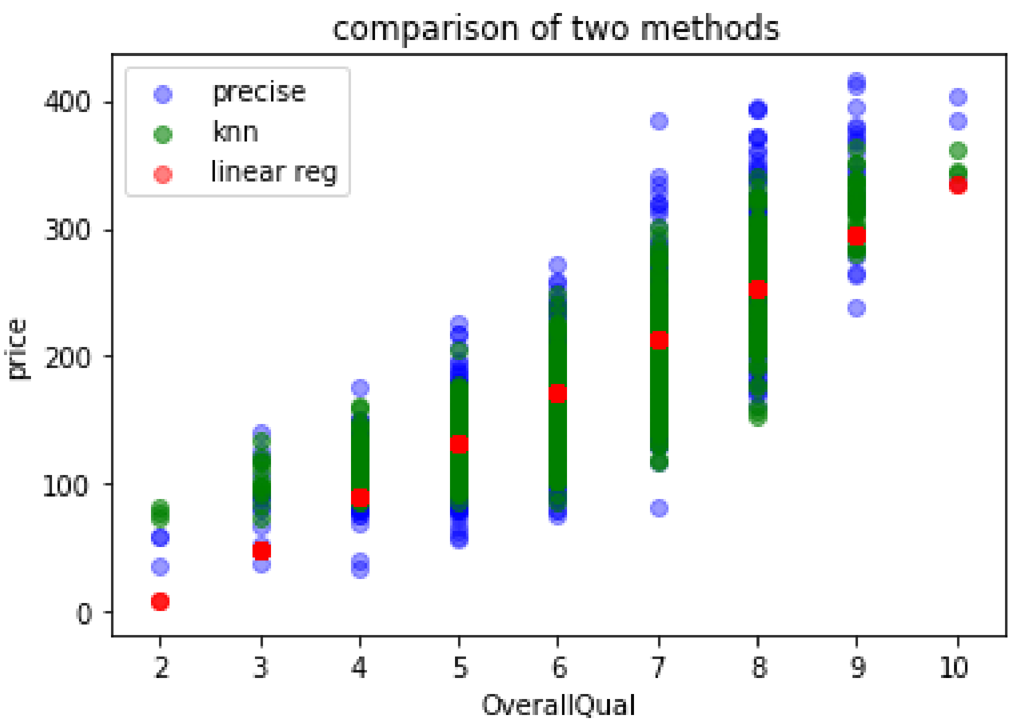
to test knnreg make an instance and call it by the sample such as above

5.2 compare two methods

In this part linear regression patametric method and KNN non-parametric method will be compared with regard to RMSE factor and in result the KNN methods worked more accurate over all neglecting the fact that knn computes more mathematical operations

5.2.1 visualization comparison

[19]:
```
features = num_data.OverallQual
true_price = num_data.SalePrice

plt.scatter(features ,true_price     , color = 'b' , alpha = 0.4 , label =␣
 ↪'precise')
plt.scatter(features , knn_estimate , color = 'g' , alpha = 0.6 , label = 'knn')
plt.scatter(features ,lin_estimate  , color = 'r' , alpha = 0.5 , label =␣
 ↪'linear reg')
plt.xlabel('OverallQual')
plt.ylabel('price')
plt.title('comparison of two methods')
plt.legend()
plt.show()
```



5.2.2 RMSE comparison

[20]:
```
print('KNN        : ', RMSE(true_price , knn_estimate))
print('linear reg : ', RMSE(true_price.values.reshape(-1,1) , lin_estimate))
```

```
KNN        :  25.29561802039879
```

```
linear reg :  38.54666046317152
```