In [3]:

```python
import pandas as pd
```

In [4]:

```python
import matplotlib.pyplot as plt
import numpy as np
```

In [5]:

```python
df=pd.read_csv("kc_house_data.csv")
```

In [6]:

```python
df
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180.0	0	1955	0	98178	47.5112	-1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170.0	400	1951	1991	98125	47.7210	-1
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770.0	0	1933	0	98028	47.7379	-1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050.0	910	1965	0	98136	47.5208	-1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680.0	0	1987	0	98074	47.6168	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530.0	0	2009	0	98103	47.6993	-1
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310.0	0	2014	0	98146	47.5107	-1
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020.0	0	2009	0	98144	47.5944	-1
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600.0	0	2004	0	98027	47.5345	-1
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020.0	0	2008	0	98144	47.5941	-1

21613 rows × 21 columns

In [7]:

```python
df.shape
```

Out[7]:

```
(21613, 21)
```

In [38]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21611 non-null  float64
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(6), int64(14), object(1)
memory usage: 3.5+ MB
```

In [48]:

```python
df['date'] = pd.to_datetime(df['date'])
df['Month'] = df['date'].apply(lambda date: date.month)
df['Year'] = df['date'].apply(lambda date: date.year)
```

In [107…

```python
Y = df['price'].values
```

In [108…

```python
print(X.shape)
print(Y.shape)
```

```
(21611,)
(21611,)
```

In [109…

```python
df.head()
```

Out[109…

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	...	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	7129300520	221900.0	3	1.00	1180	5650	1.0	0	0	3	...	0	1955	0	98178	47.5112	-122.257	1340	5650
1	6414100192	538000.0	3	2.25	2570	7242	2.0	0	0	3	...	400	1951	1991	98125	47.7210	-122.319	1690	7639
2	5631500400	180000.0	2	1.00	770	10000	1.0	0	0	3	...	0	1933	0	98028	47.7379	-122.233	2720	8062
3	2487200875	604000.0	4	3.00	1960	5000	1.0	0	0	5	...	910	1965	0	98136	47.5208	-122.393	1360	5000
4	1954400510	510000.0	3	2.00	1680	8080	1.0	0	0	3	...	0	1987	0	98074	47.6168	-122.045	1800	7503

5 rows × 22 columns

In [110…

```python
df.isnull().sum()
```

Out[110…

```
id                 0
price              0
bedrooms           0
```

```
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
Month            0
Year             0
dtype: int64
```

In [111…

```
df.dropna(inplace=True)
```

In [112…

```
df.isnull().sum()
```

Out[112…

```
id               0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
Month            0
Year             0
dtype: int64
```

In [113…

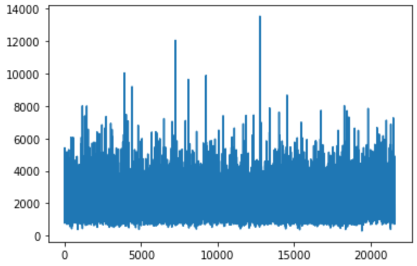
```
sqft=df['sqft_living']
```

In [114…

```
sqft.plot()
```

Out[114…

```
<AxesSubplot:>
```



In [167…

```
X = df[['bedrooms','bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view','condition',
       'grade','sqft_above','sqft_basement','sqft_living15','sqft_lot15']].values
y = df['price'].values
```

In [168…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=50)
```

Normalize the data.

In [169…

```
std = StandardScaler()
X = std.fit_transform(X)
```

In [170…

```
from sklearn.ensemble import RandomForestRegressor
```

In [171…

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,mean_squared_error,mean_absolute_error
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
```

In [172…

```
std = StandardScaler()
X = std.fit_transform(X)
```

In [173…

```
rfr = RandomForestRegressor(n_estimators=200)
rfr.fit(X_train,y_train)
```

Out[173…

```
RandomForestRegressor(n_estimators=200)
```

In [174…

```
score_rfr = rfr.score(X_train,y_train)
prev_rfr = rfr.predict(X_test)
              an_absolute_error(y_test,prev_rfr)
```

```
mse_rfr = mean_squared_error(y_test,prev_rfr)
rmse_rfr = np.sqrt(mean_squared_error(y_test,prev_rfr))
```

In [175…

```
print('Mae: ',mae_rfr)
print('Mse: ',mse_rfr)
print('Rmse: ',rmse_rfr)
```

```
Mae:  123126.961817964
Mse:  37677739823.719635
Rmse:  194107.5470550273
```

LINEAR REGRESSION

In [176…

```
lr = LinearRegression()
lr.fit(X_train,y_train)
```

Out[176…

```
LinearRegression()
```

In [177…

```
pred_lr = lr.predict(X_test)
score_lr = lr.score(X_train,y_train)
```

In [178…

```
mae_lr = mean_absolute_error(y_test,pred_lr)
mse_lr = mean_squared_error(y_test,pred_lr)
rmse_lr = np.sqrt(mse_lr)
```

In [179…

```
print('Mae_lr: ',mae_lr)
print('Mse_lr: ',mse_lr)
print('Rmse_lr: ',rmse_lr)
```

```
Mae_lr:  153081.6455419886
Mse_lr:  54814955728.526276
Rmse_lr:  234125.93988818556
```

In [180…

```
import matplotlib.pyplot as plt
```

In [181…

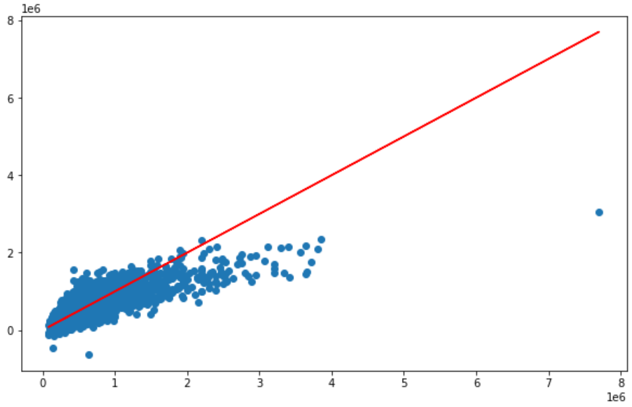
```
def resizeplot(l,a):
    plt.figure(figsize=(l,a))
```

In [195…

```
resizeplot(10,6)
plt.scatter(y_test,pred_lr)
plt.plot(y_test,y_test,color='red')
```

Out[195…

```
[<matplotlib.lines.Line2D at 0x2542230d3c8>]
```

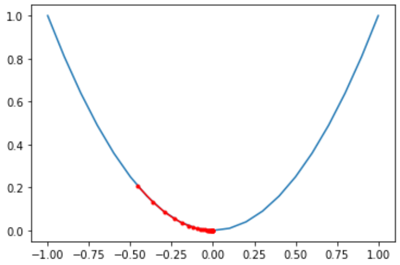


example of plotting a gradient descent search on a one-dimensional function

In [209…

```
from numpy import asarray
from numpy import arange
from numpy.random import rand
from matplotlib import pyplot

# objective function
def objective(x):
        return x**2.0

# derivative of objective function
def derivative(x):
        return x * 2.0

# gradient descent algorithm
def gradient_descent(objective, derivative, bounds, n_iter, step_size):
        # track all solutions
        solutions, scores = list(), list()
        # generate an initial point
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
        # run the gradient descent
        for i in range(n_iter):
                # calculate gradient
                gradient = derivative(solution)
                # take a step
                solution = solution - step_size * gradient
                # evaluate candidate point
                solution_eval = objective(solution)
                # store solution
                solutions.append(solution)
                scores.append(solution_eval)
                # report progress
                print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
        return [solutions, scores]

# define range for input
bounds = asarray([[-1.0, 1.0]])
# define the total iterations
n_iter = 30
# define the step size
step_size = 0.1
# perform the gradient descent search
solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_size)
# sample input range uniformly at 0.1 increments
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
# compute targets
results = objective(inputs)
# create a line plot of input vs result
pyplot.plot(inputs, results)
# plot the solutions found
pyplot.plot(solutions, scores, '.-', color='red')
# show the plot
pyplot.show()
```

```
>0 f([-0.45530131]) = 0.20730
>1 f([-0.36424104]) = 0.13267
>2 f([-0.29139284]) = 0.08491
>3 f([-0.23311427]) = 0.05434
>4 f([-0.18649141]) = 0.03478
>5 f([-0.14919313]) = 0.02226
>6 f([-0.11935451]) = 0.01425
>7 f([-0.0954836]) = 0.00912
>8 f([-0.07638688]) = 0.00583
>9 f([-0.06110951]) = 0.00373
>10 f([-0.04888761]) = 0.00239
>11 f([-0.03911008]) = 0.00153
>12 f([-0.03128807]) = 0.00098
>13 f([-0.02503045]) = 0.00063
>14 f([-0.02002436]) = 0.00040
>15 f([-0.01601949]) = 0.00026
>16 f([-0.01281559]) = 0.00016
>17 f([-0.01025247]) = 0.00011
>18 f([-0.00820198]) = 0.00007
>19 f([-0.00656158]) = 0.00004
>20 f([-0.00524927]) = 0.00003
>21 f([-0.00419941]) = 0.00002
>22 f([-0.00335953]) = 0.00001
>23 f([-0.00268762]) = 0.00001
>24 f([-0.0021501]) = 0.00000
>25 f([-0.00172008]) = 0.00000
>26 f([-0.00137606]) = 0.00000
>27 f([-0.00110085]) = 0.00000
>28 f([-0.00088068]) = 0.00000
>29 f([-0.00070454]) = 0.00000
```



In [ ]:

Loading [MathJax]/extensions/Safe.js