MT2022161 ADITYA.M

ASSIGNMENT -1 [AI - 511]

DATA PRE-PROCESSING

MODULES REQUIRED

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

LOADING DATA

In [2]:
```
d=pd.read_csv("boston.csv")
```

DATA OBSERVATION

In [3]:
```
d.head(10)
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311.0	15.2	386.71	17.10	18.9

In [4]:
```
d.drop_duplicates()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [5]:
```
d.shape
```

Out[5]:
```
(506, 14)
```

we can see that there are 506 unique data points with 13 features and 1 output variable

In [6]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

we can observe that all the features are either float or interger valued with no NULL values

In [7]:
```
d.describe()
```

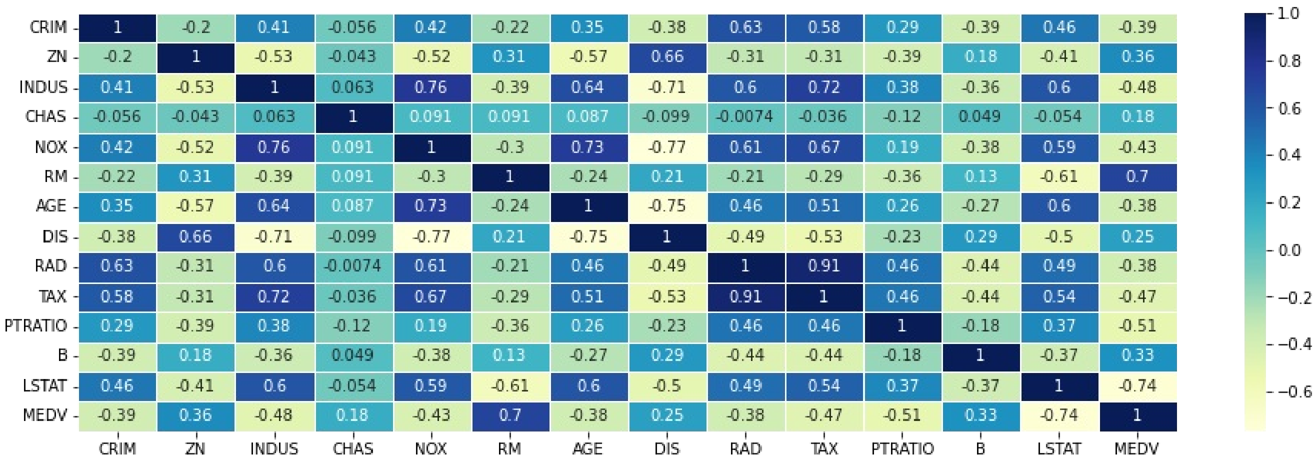
Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000

UNIVARIENT LINEAR REGRESSION

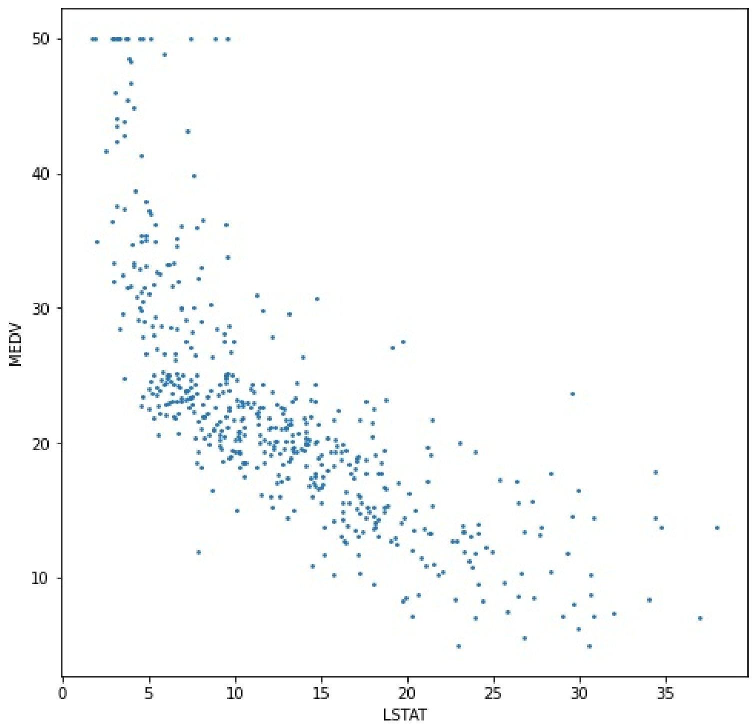
step 1: selection the feature for univarient linear regression

In [8]:
```
import seaborn as sns
cor=d.corr()
plt.figure(figsize=(16,5))
dataplot=sns.heatmap(cor,cmap="YlGnBu",annot=True,linewidths=0.5)
```

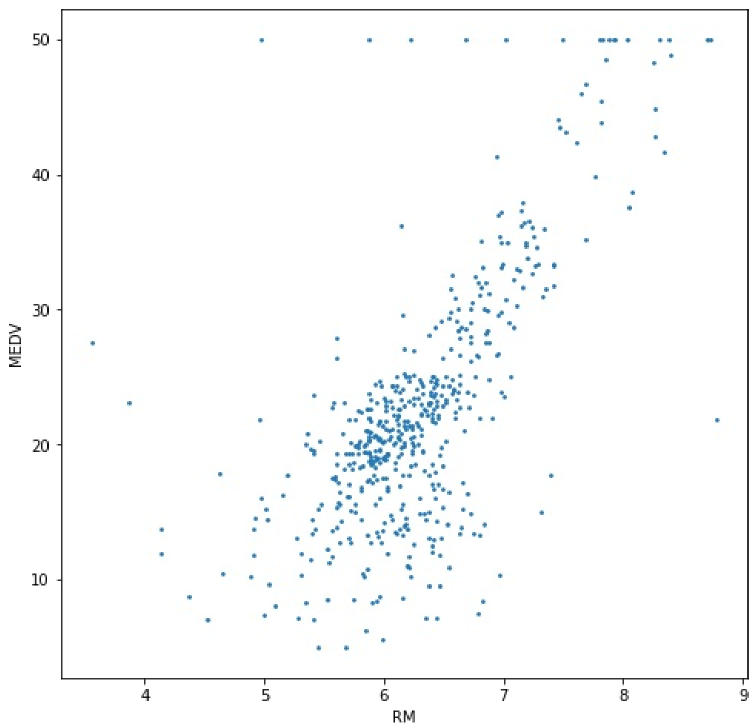


the output variable 'MEDV' has high co-relation with features 'RM' and 'LSTAT', so any 1 features can be choosen for univarient linear regression.

In [9]:
```
plt.figure(figsize=(8,8))
plt.scatter(d['LSTAT'],d['MEDV'],s=3)
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
plt.show()
```



In [10]:
```
plt.figure(figsize=(8,8))
plt.scatter(d['RM'],d['MEDV'],s=3)
plt.xlabel('RM')
plt.ylabel('MEDV')
plt.show()
```



We can take 'LSTAT' as the feature for predicting 'MEDV' in univarient linear regression.

MIN MAX normalization for bringing data values between 0 and 1 for easy visualization

In [11]:
```
#X=d['RM']
X=d['LSTAT']
Y=d['MEDV']
xmin,xmax=X.min(),X.max()
ymin,ymax=Y.min(),Y.max()
X=(X-xmin)/(xmax-xmin)
#Y=(Y-ymin)/(ymax-ymin)
```

Below code is used to split the data for training and testing in the given ratio

In [12]:
```
# splitting the data into train test
import random
def split(X,Y,ratio=0.4): #ratio is the percentage of data used for training
    n=int(ratio*len(Y))
```

```
    #train size is 40% by default
    train_index=[]
    test_index=[]

    while len(train_index)!=n:
        k=random.randrange(0,len(X))
        if k not in train_index:
            train_index.append(k)
        else:
            pass
    train_index.sort()

    for i in range(0,len(X)):
        if i not in train_index:
            test_index.append(i)

    x_train=[]
    y_train=[]
    for index in train_index:
        x_train.append(X[index])
        y_train.append(Y[index])

    x_test=[]
    y_test=[]
    for index in test_index:
        x_test.append(X[index])
        y_test.append(Y[index])

    return x_test,x_train,y_test,y_train

x_test,x_train,y_test,y_train=split(X,Y,0.7) #70% for training
```

step 2: defining the model class

In [13]:

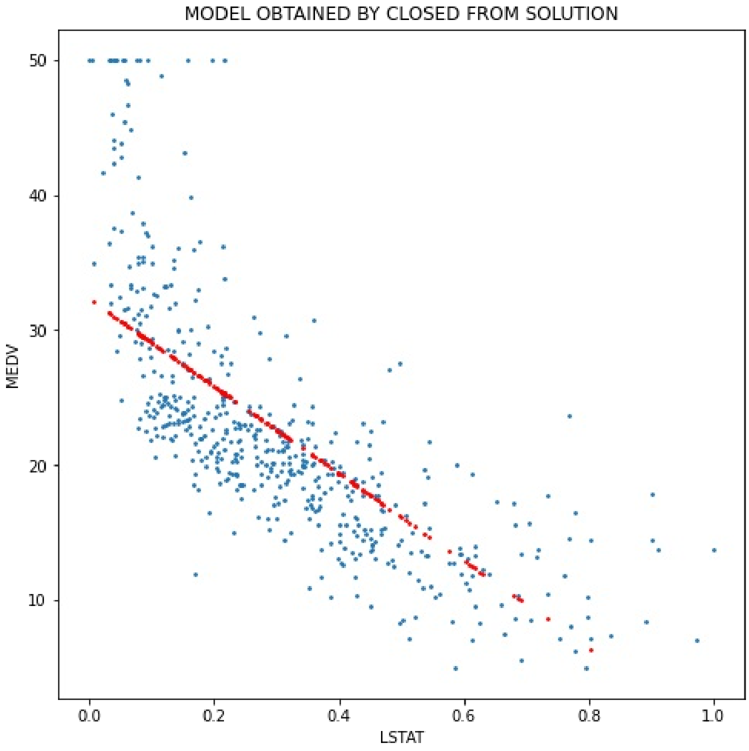
```
class univarientLinearRegression():
    def __init__(self):
        self.parameter=np.array([[0],[0]])

    def fit(self,x,y): #closed form solution
        n=len(x)
        xi=0
        xi2=0
        yi=0
        xiyi=0
        for i in range(len(x)):
            xi=xi+x[i]
            yi=yi+y[i]
            xi2=xi2+(x[i]*x[i])
            xiyi=xiyi+(x[i]*y[i])

        mat1=[[n,xi],[xi,xi2]]
        mat2=[[yi],[xiyi]]
        mat1_inv=np.linalg.inv(mat1)
        self.parameter=np.matmul(mat1_inv,mat2)


    def iterative_fit(self,x,y,lr=0.1,iter=1000): #gradient descent solution
        n=len(x)
        lis=[]
        for i in range(n):
            lis.append((1,x[i]))
        x=np.array(lis)
        y=np.array(y)
        y=y.reshape((n,1))
        x=x.reshape((n,2))
        while(iter):
            iter=iter-1
            k=np.subtract((np.matmul(x,self.parameter)),y)
            dL=np.matmul(x.T,k)
            dL=dL/n
            self.parameter=np.subtract(self.parameter,(lr*dL))

    def predict(self,x):
        y=[]
        a=self.parameter[0][0]
        b=self.parameter[1][0]
        for i in range(len(x)):
            k=a+b*x[i]
            y.append(k)
        return y

    def mse(self,y_pred,y):
        error=0
        for i in range(len(y)):
            e=(y[i]-y_pred[i])*(y[i]-y_pred[i])
            error=error+e
        error=error/len(y)
```

```
        return error
```

Step 3: Closed form solution

In [14]:
```
model=univarientLinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
mse=model.mse(y_pred,y_test)

print("mean squared error is %s"%(mse))
plt.figure(figsize=(8,8))
plt.scatter(X,Y,s=3)
plt.scatter(x_test,y_pred,s=3,color='red')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
plt.title('MODEL OBTAINED BY CLOSED FROM SOLUTION')
plt.show()
```

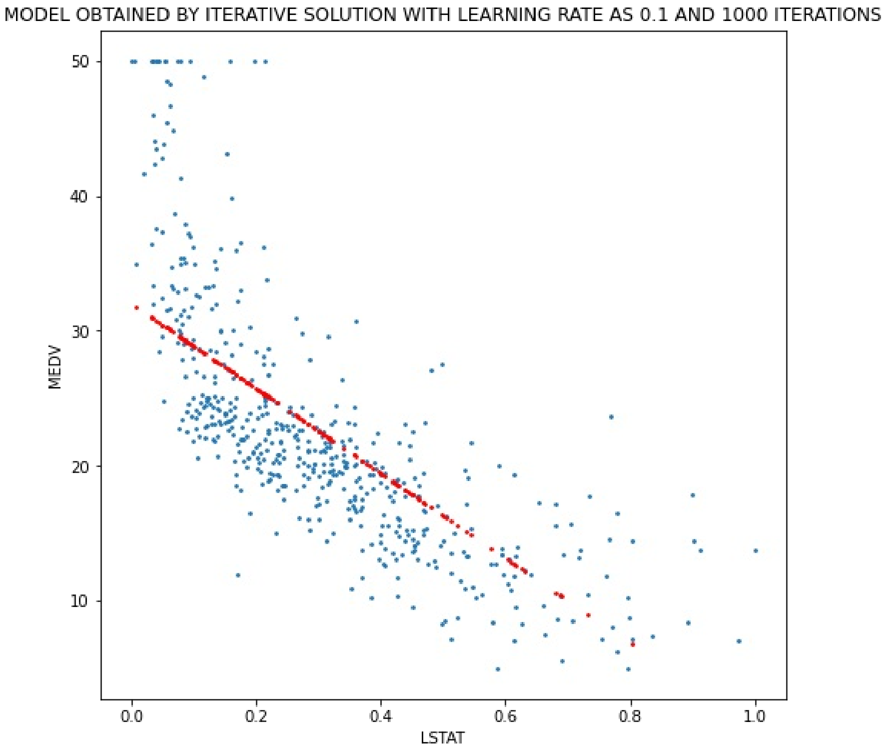
```
mean squared error is 40.76695434829298
```



step 4: gradient descent solution

In [15]:
```
model2=univarientLinearRegression()
model2.iterative_fit(x_train,y_train,0.01,10000)
y_pred2=model2.predict(x_test)
mse=model2.mse(y_pred2,y_test)

print("mean squared error is %s"%(mse))
plt.figure(figsize=(8,8))
plt.scatter(X,Y,s=3)
plt.scatter(x_test,y_pred2,s=3,color='red')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')
plt.title('MODEL OBTAINED BY ITERATIVE SOLUTION WITH LEARNING RATE AS 0.1 AND 1000 ITERATIONS')
plt.show()
```

```
mean squared error is 41.26587591491594
```



In [16]:
```
print(model.parameter)# closed form parameters
print(model2.parameter)# gradient descent parameters
```

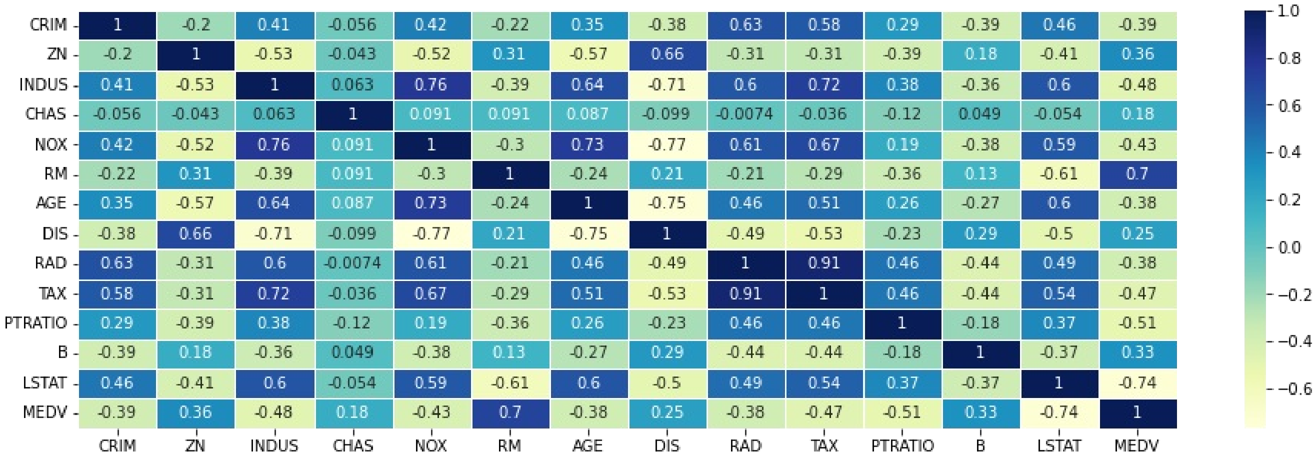
```
[[ 32.27769454]
 [-32.24427883]]
[[ 32.00490538]
 [-31.39543633]]
```

We can see that 'LSTAT' has a linear relationship with the 'median price' of the house, but in some cases other parameters are affecting the price of the house, hence we have to explore multivarient linear regression.

MULTIVARIENT LINEAR REGRESSION

step 1: selection of features for multivarient linear regression

In [17]:
```
import seaborn as sns
cor=d.corr()
plt.figure(figsize=(16,5))
dataplot=sns.heatmap(cor,cmap="YlGnBu",annot=True,linewidths=0.5)
```



'RAD' is highly co-related with 'TAX', hence we can eleminate it.

In [18]:
```
d=d.drop(['RAD'],axis=1)
#getting rid of redudant features
```

In [19]:
```
y=d.iloc[:,-1]
for feature in d.columns[:-1]:
    mean=d[feature].mean()
    std=d[feature].std()
    d[feature]=((d[feature]-mean)/(std))
x=d.iloc[:,0:-1] #standerdized values
r,c=x.shape
cons=[1 for i in range(r)]
x.insert(0,'Constant',cons,True) #inserting costants
```

In [20]:
```
X=np.array(x)
y=np.array(y)
n=len(y)
y=y.reshape((n,1))
x_test,x_train,y_test,y_train=split(X,y,0.6)
```

step 2: defining the model class

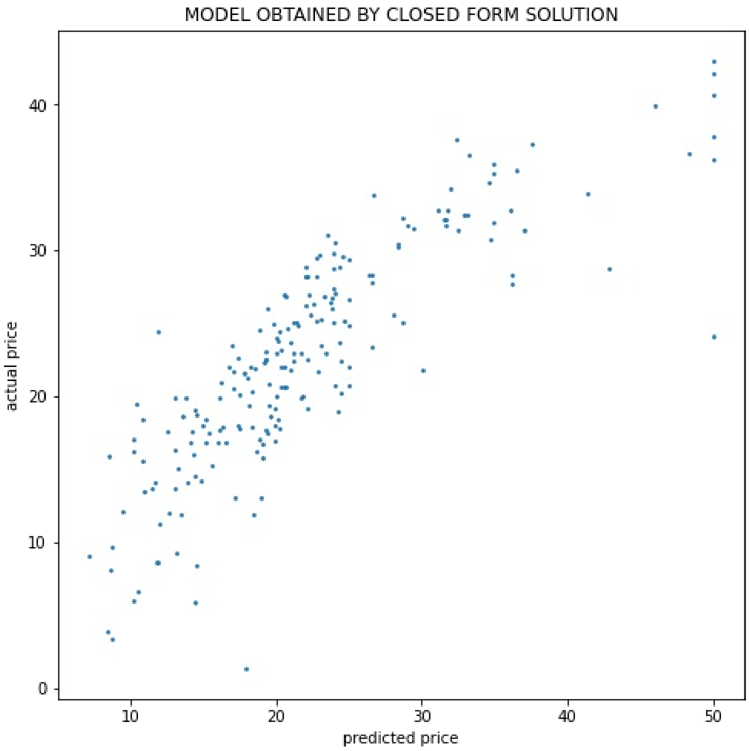
In [21]:
```
class multivarientLinearRegression():
    def __init__(self):
        self.parameter=None

    def fit(self,x,y): #closed form solution
        n,k=len(x),len(x[0])
        x=np.array(x)
        y=np.array(y)
        #n number of data points and k features
        a=np.matmul(x.T,x)
        a=np.linalg.inv(a)
        b=np.matmul(x.T,y)
        self.parameter=np.matmul(a,b)


    def iterative_fit(self,x,y,lr=0.01,iter=10000): #gradient descent solution
        n,k=len(x),len(x[0])
        x=np.array(x)
        y=np.array(y)
        self.parameter=np.ones(k)
        self.parameter=self.parameter.reshape(k,1)
        while(iter):
            iter=iter-1
            t=np.subtract((np.matmul(x,self.parameter)),y)
            dL=np.matmul(x.T,t)
            dL=dL/n
            self.parameter=np.subtract(self.parameter,(lr*dL))

    def predict(self,x):
        x=np.array(x)
        y_pred=np.matmul(x,self.parameter)
        return y_pred

    def mse(self,y_pred,y):
        y=np.array(y)
        error=0
        for i in range(len(y)):
            e=(y[i]-y_pred[i])*(y[i]-y_pred[i])
            error=error+e
        error=error/len(y)
        return int(error)
```

step 3: closed form solution

In [22]:

```
mod=multivarientLinearRegression()
mod.fit(x_train,y_train)
y_pred=mod.predict(x_test)
mse=mod.mse(y_pred,y_test)
print("mean squared error is %s"%(mse))
plt.figure(figsize=(8,8))
plt.scatter(y_test,y_pred,s=3)
plt.xlabel('predicted price')
plt.ylabel('actual price')
plt.title('MODEL OBTAINED BY CLOSED FORM SOLUTION')
plt.show()
```

```
mean squared error is 23
```



step 4: gradient descent solution

In [23]:

```
mod2=multivarientLinearRegression()
mod2.iterative_fit(x_train,y_train)
y_pred2=mod2.predict(x_test)
mse=mod2.mse(y_pred2,y_test)
print("mean squared error is %s"%(mse))
plt.figure(figsize=(8,8))
plt.scatter(y_test,y_pred2,s=3)
plt.xlabel('predicted price')
plt.ylabel('actual price')
plt.title('MODEL OBTAINED BY GRADIENT DESCENT WITH LEARNING RATE AS 0.01 AND 10000 ITERATIONS')
plt.show()
```

```
mean squared error is 23
```

MODEL OBTAINED BY GRADIENT DESCENT WITH LEARNING RATE AS 0.01 AND 10000 ITERATIONS