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

In [2]:
```
%matplotlib inline
import warnings
warnings.filterwarnings(action="ignore")
df=pd.read_csv("/home/student/Downloads/housing.csv")
df.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	Na

In [3]:
```
df.shape
```

Out[3]: (506, 14)

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```


In [5]:
```
df.isnull().sum()
```

Out[5]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

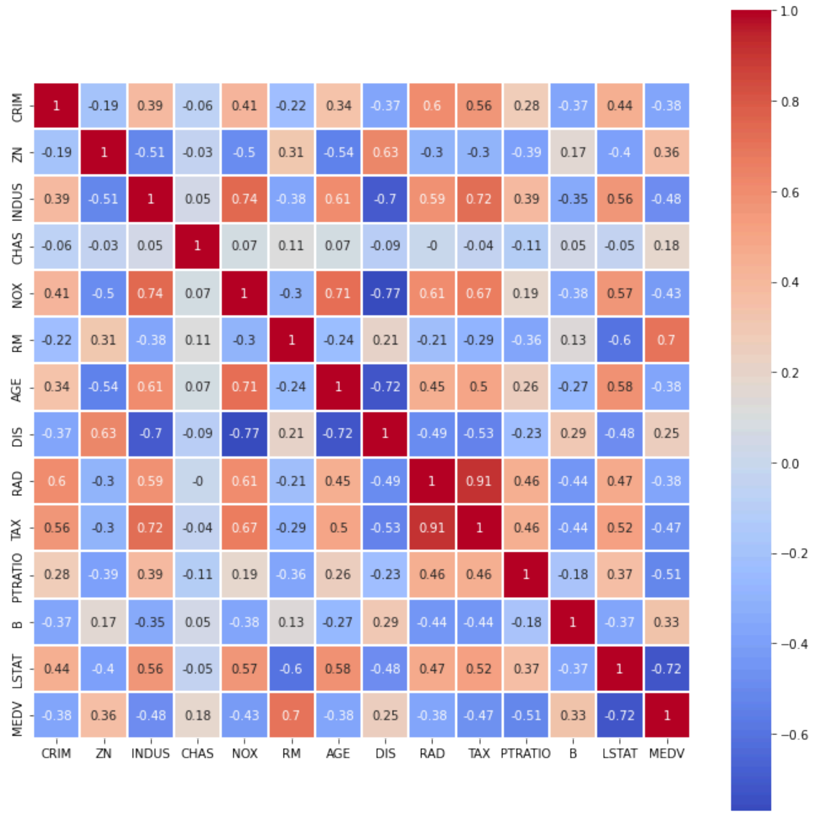
In [6]:
```
name=["CRIM","ZN","INDUS","CHAS","NOX","RM","AGE","DIS","RAD","TAX","P
for i in name:
    df[i].fillna(df[i].median(),inplace=True)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [8]:
```
plt.figure(figsize=(12,12))
sns.heatmap(data=df.corr().round(2),annot=True,cmap='coolwarm',linewid
```

Out[8]: <Axes: >



In [9]:
```
df1=df[['RM','LSTAT','PTRATIO','TAX','MEDV']]
df1
```

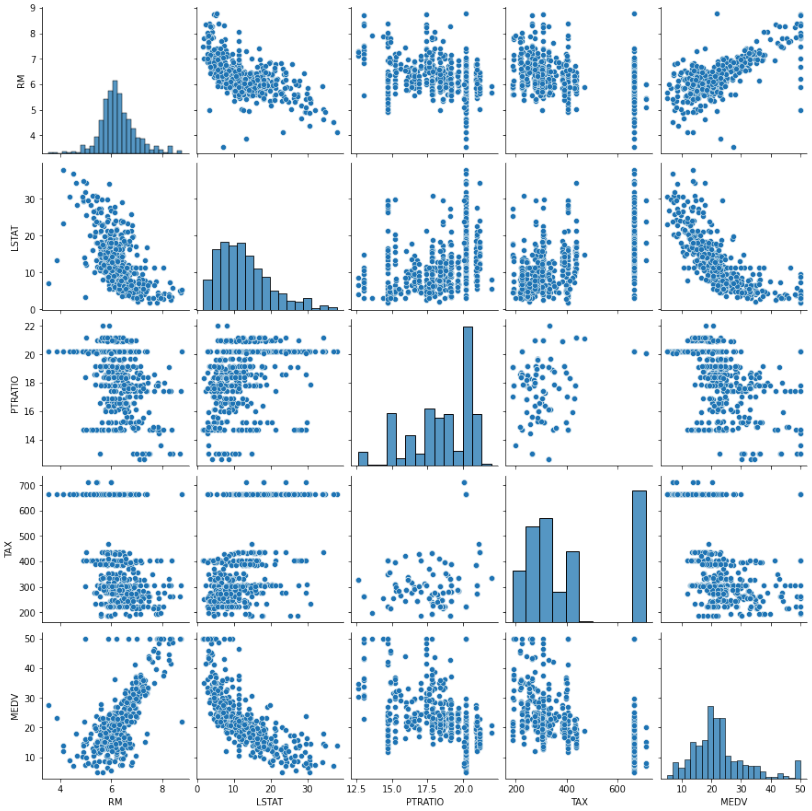
Out[9]:

	RM	LSTAT	PTRATIO	TAX	MEDV
0	6.575	4.98	15.3	296	24.0
1	6.421	9.14	17.8	242	21.6
2	7.185	4.03	17.8	242	34.7
3	6.998	2.94	18.7	222	33.4
4	7.147	11.43	18.7	222	36.2
...	...	...	...	...	...
501	6.593	11.43	21.0	273	22.4
502	6.120	9.08	21.0	273	20.6
503	6.976	5.64	21.0	273	23.9
504	6.794	6.48	21.0	273	22.0
505	6.030	7.88	21.0	273	11.9

506 rows × 5 columns

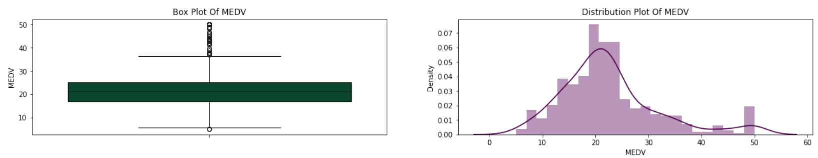
In [10]:
```
sns.pairplot(df1)
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x7c0deabdf040>



In [11]:
```
d=df1.describe()
```

In [12]:
```
plt.figure(figsize=(20,3))

plt.subplot(1,2,1)
sns.boxplot(df1.MEDV,color='#005030')
plt.title('Box Plot Of MEDV')

plt.subplot(1,2,2)
sns.distplot(a=df1.MEDV,color='#500050')
plt.title('Distribution Plot Of MEDV')

plt.show()
```



In [13]:
```
MEDV_Q3=d['MEDV']['75%']
MEDV_Q3
```

Out[13]: 25.0

In [14]:
```
MEDV_Q1=d['MEDV']['25%']
MEDV_Q1
```

Out[14]: 17.025

In [15]:
```
MEDV_IQR=MEDV_Q3-MEDV_Q1
MEDV_IQR
```

Out[15]: 7.975000000000001

In [16]:
```
MEDV_UV=MEDV_Q3+1.25*MEDV_IQR
MEDV_UV
```

Out[16]: 34.96875

In [17]:
```
MEDV_NV=MEDV_Q1-1.25*MEDV_IQR
MEDV_NV
```

Out[17]: 7.056249999999997

In [22]:
```
df1[df1['MEDV']>MEDV_UV].sort_values(by=['MEDV','RM'])
```


Out[22]:

	RM	LSTAT	PTRATIO	TAX	MEDV
279	6.812	4.85	14.9	216	35.1
273	7.691	6.58	18.6	223	35.2
281	6.968	4.59	14.9	216	35.4
55	7.249	4.81	17.9	226	35.4
258	7.333	7.79	13.0	264	36.0
304	7.236	6.93	18.4	222	36.1
181	6.144	9.45	17.8	193	36.2
4	7.147	11.43	18.7	222	36.2
192	7.178	2.87	15.2	398	36.4
264	7.206	8.10	13.0	264	36.5
190	6.951	5.10	15.2	398	37.0
179	6.980	5.04	17.8	193	37.2
291	7.148	3.56	19.2	245	37.3
226	8.040	11.43	17.4	307	37.6
182	7.155	4.82	17.8	193	37.9
97	8.069	4.21	18.0	276	38.7
180	7.765	7.56	17.8	193	39.8
157	6.943	4.59	14.7	403	41.3
232	8.337	2.47	17.4	307	41.7
202	7.610	3.11	14.7	348	42.3
253	8.259	3.54	19.1	330	42.8
261	7.520	7.26	13.0	264	43.1
268	7.470	3.16	13.0	264	43.5
98	7.820	3.57	18.0	276	43.8
256	7.454	3.11	15.9	244	44.0
224	8.266	4.14	17.4	307	44.8
280	7.820	3.76	14.9	216	45.4
282	7.645	3.01	14.9	216	46.0
228	7.686	11.43	17.4	307	46.7
233	8.247	3.95	17.4	307	48.3
203	7.853	3.81	14.7	224	48.5
262	8.398	5.91	13.0	264	48.8
368	4.970	3.26	20.2	666	50.0
372	5.875	8.88	20.2	666	50.0
371	6.216	9.53	20.2	666	50.0
369	6.683	3.73	20.2	666	50.0
370	7.016	2.96	20.2	666	50.0
161	7.489	1.73	14.7	403	50.0
162	7.802	1.92	14.7	403	50.0

	RM	LSTAT	PTRATIO	TAX	MEDV
186	7.831	4.45	17.8	193	50.0
195	7.875	2.97	14.4	255	50.0
283	7.923	3.16	13.6	198	50.0
166	7.929	3.70	14.7	403	50.0
204	8.034	2.88	14.7	224	50.0
267	8.297	7.44	13.0	264	50.0
163	8.375	3.32	14.7	403	50.0
257	8.704	5.12	13.0	264	50.0
225	8.725	4.63	17.4	307	50.0

In [21]:
```
print(f'Shape of thae dataset before removing outliers:{df1.shape}')
df2=df1[~(df1['MEDV']==50)]
print(f'Shape of thae dataset after removing outliers:{df2.shape}')
```

```
Shape of thae dataset before removing outliers:(506, 5)
Shape of thae dataset after removing outliers:(490, 5)
```

In [23]:
```
MEDV_Q3 = d['MEDV']['75%']
MEDV_Q1 = d['MEDV']['25%']
MEDV_IQR = MEDV_Q3 - MEDV_Q1
MEDV_UV = MEDV_Q3 + 1.5*MEDV_IQR
MEDV_LV = MEDV_Q1 - 1.5*MEDV_IQR

df1[df1['MEDV']<MEDV_LV]
```

Out[23]:

	RM	LSTAT	PTRATIO	TAX	MEDV
398	5.453	30.59	20.2	666	5.0
405	5.683	22.98	20.2	666	5.0

In [24]:
```
df1[df1['MEDV']>MEDV_UV].sort_values(by=['MEDV','RM'])
```

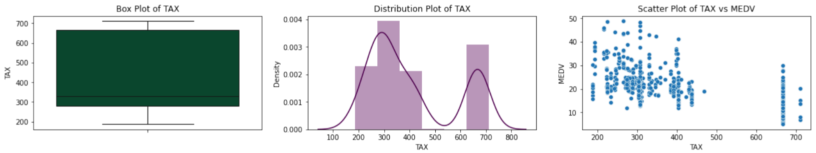
Out[24]:

	RM	LSTAT	PTRATIO	TAX	MEDV
190	6.951	5.10	15.2	398	37.0
179	6.980	5.04	17.8	193	37.2
291	7.148	3.56	19.2	245	37.3
226	8.040	11.43	17.4	307	37.6
182	7.155	4.82	17.8	193	37.9
97	8.069	4.21	18.0	276	38.7
180	7.765	7.56	17.8	193	39.8
157	6.943	4.59	14.7	403	41.3
232	8.337	2.47	17.4	307	41.7
202	7.610	3.11	14.7	348	42.3
253	8.259	3.54	19.1	330	42.8
261	7.520	7.26	13.0	264	43.1
268	7.470	3.16	13.0	264	43.5
98	7.820	3.57	18.0	276	43.8
256	7.454	3.11	15.9	244	44.0
224	8.266	4.14	17.4	307	44.8
280	7.820	3.76	14.9	216	45.4
282	7.645	3.01	14.9	216	46.0
228	7.686	11.43	17.4	307	46.7
233	8.247	3.95	17.4	307	48.3
203	7.853	3.81	14.7	224	48.5
262	8.398	5.91	13.0	264	48.8
368	4.970	3.26	20.2	666	50.0
372	5.875	8.88	20.2	666	50.0
371	6.216	9.53	20.2	666	50.0
369	6.683	3.73	20.2	666	50.0
370	7.016	2.96	20.2	666	50.0
161	7.489	1.73	14.7	403	50.0
162	7.802	1.92	14.7	403	50.0
186	7.831	4.45	17.8	193	50.0
195	7.875	2.97	14.4	255	50.0
283	7.923	3.16	13.6	198	50.0
166	7.929	3.70	14.7	403	50.0
204	8.034	2.88	14.7	224	50.0
267	8.297	7.44	13.0	264	50.0
163	8.375	3.32	14.7	403	50.0
257	8.704	5.12	13.0	264	50.0
225	8.725	4.63	17.4	307	50.0

In [25]:
```
print(f'Shape of dataset before remving Outliers: {df1.shape}')
df2 = df1[~(df1['MEDV']==50)]
print(f'Shape of dataset after remving Outliers: {df2.shape}')
```

```
Shape of dataset before remving Outliers: (506, 5)
Shape of dataset after remving Outliers: (490, 5)
```

In [27]:
```
#Box Plot, Distribution Plot and Scatter Plot for TAX
plt.figure(figsize=(20,3))

plt.subplot(1,3,1)
sns.boxplot(df2.TAX,color='#005030')
plt.title('Box Plot of TAX')

plt.subplot(1,3,2)
sns.distplot(a=df2.TAX,color='#500050')
plt.title('Distribution Plot of TAX')

plt.subplot(1,3,3)
sns.scatterplot(x=df2.TAX,y=df2.MEDV)
plt.title('Scatter Plot of TAX vs MEDV')

plt.show()
```



In [28]:
```
temp_df = df2[df1['TAX']>600].sort_values(by=['RM','MEDV'])
temp_df.shape
```

Out[28]: (132, 5)

In [29]:
```
temp_df.describe()
```

Out[29]:

	RM	LSTAT	PTRATIO	TAX	MEDV
count	132.000000	132.000000	132.000000	132.000000	132.000000
mean	6.000689	18.828864	20.196212	667.704545	14.994697
std	0.712621	6.590380	0.019163	8.623365	5.405825
min	3.561000	5.290000	20.100000	666.000000	5.000000
25%	5.674250	14.175000	20.200000	666.000000	10.900000
50%	6.139500	17.910000	20.200000	666.000000	14.100000
75%	6.407250	23.052500	20.200000	666.000000	19.200000
max	8.780000	37.970000	20.200000	711.000000	29.800000

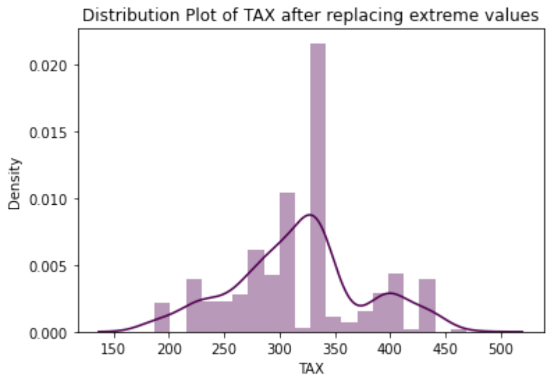
In [30]:
```
TAX_10 = df2[(df2['TAX']<600) & (df2['LSTAT']>=0) & (df2['LSTAT']<10)]
TAX_20 = df2[(df2['TAX']<600) & (df2['LSTAT']>=10) & (df2['LSTAT']<20)
TAX_30 = df2[(df2['TAX']<600) & (df2['LSTAT']>=20) & (df2['LSTAT']<30)
TAX_40 = df2[(df2['TAX']<600) & (df2['LSTAT']>=30)]['TAX'].mean()

indexes = list(df2.index)
for i in indexes:
    if df2['TAX'][i] > 600:
        if (0 <= df2['LSTAT'][i] < 10):
            df2.at[i,'TAX'] = TAX_10
        elif (10 <= df2['LSTAT'][i] < 20):
            df2.at[i,'TAX'] = TAX_20
        elif (20 <= df2['LSTAT'][i] < 30):
            df2.at[i,'TAX'] = TAX_30
        elif (df2['LSTAT'][i] >30):
            df2.at[i,'TAX'] = TAX_40

print('Values imputed successfully')
```

```
Values imputed successfully
```

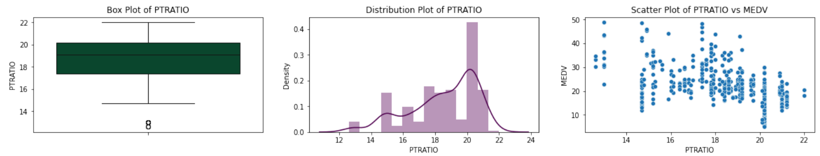
In [31]:
```
#This show all those extreme TAX values are replaced successfully
df2[df2['TAX']>600]['TAX'].count()
```

Out[31]: 0

In [32]:
```
sns.distplot(a=df2.TAX,color='#500050')
plt.title('Distribution Plot of TAX after replacing extreme values')
plt.show()
```



In [34]:
```
#Box Plot, Distribution Plot and Scatter Plot for PTRATIO
plt.figure(figsize=(20,3))

plt.subplot(1,3,1)
sns.boxplot(df2.PTRATIO,color='#005030')
plt.title('Box Plot of PTRATIO')

plt.subplot(1,3,2)
sns.distplot(a=df2.PTRATIO,color='#500050')
plt.title('Distribution Plot of PTRATIO')

plt.subplot(1,3,3)
sns.scatterplot(x=df2.PTRATIO,y=df2.MEDV)
plt.title('Scatter Plot of PTRATIO vs MEDV')

plt.show()
```

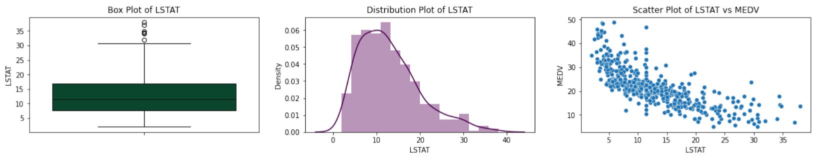


In [35]:
```
df2[df2['PTRATIO']<14].sort_values(by=['LSTAT','MEDV'])
```

Out[35]:

	RM	LSTAT	PTRATIO	TAX	MEDV
268	7.470	3.16	13.0	264.0	43.5
196	7.287	4.08	12.6	329.0	33.3
262	8.398	5.91	13.0	264.0	48.8
198	7.274	6.62	12.6	329.0	34.6
259	6.842	6.90	13.0	264.0	30.1
261	7.520	7.26	13.0	264.0	43.1
258	7.333	7.79	13.0	264.0	36.0
264	7.206	8.10	13.0	264.0	36.5
197	7.107	8.61	12.6	329.0	30.3
260	7.203	9.59	13.0	264.0	33.8
265	5.560	10.45	13.0	264.0	22.8
263	7.327	11.25	13.0	264.0	31.0
266	7.014	14.79	13.0	264.0	30.7

In [37]:
```
#Box Plot, Distribution Plot and Scatter Plot for LSTAT
plt.figure(figsize=(20,3))

plt.subplot(1,3,1)
sns.boxplot(df2.LSTAT,color='#005030')
plt.title('Box Plot of LSTAT')

plt.subplot(1,3,2)
sns.distplot(a=df2.LSTAT,color='#500050')
plt.title('Distribution Plot of LSTAT')

plt.subplot(1,3,3)
sns.scatterplot(x=df2.LSTAT,y=df2.MEDV)
plt.title('Scatter Plot of LSTAT vs MEDV')

plt.show()
```

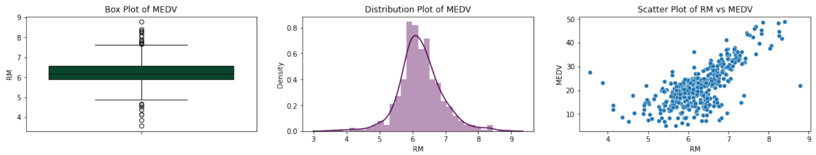


In [39]:
```
LSTAT_Q3 = d['LSTAT']['75%']
LSTAT_Q1 = d['LSTAT']['25%']
LSTAT_IQR = LSTAT_Q3 - LSTAT_Q1
LSTAT_UV = LSTAT_Q3 + 1.5*LSTAT_IQR
LSTAT_LV = LSTAT_Q1 - 1.5*LSTAT_IQR

df2[df2['LSTAT']>LSTAT_UV].sort_values(by='LSTAT')
```

Out[39]:

	RM	LSTAT	PTRATIO	TAX	MEDV
398	5.453	30.59	20.2	335.0	5.0
388	4.880	30.62	20.2	335.0	10.2
384	4.368	30.63	20.2	335.0	8.8
48	5.399	30.81	17.9	233.0	14.4
385	5.277	30.81	20.2	335.0	7.2
387	5.000	31.99	20.2	335.0	7.4
438	5.935	34.02	20.2	335.0	8.4
412	4.628	34.37	20.2	335.0	17.9
141	5.019	34.41	21.2	437.0	14.4
373	4.906	34.77	20.2	335.0	13.8
414	4.519	36.98	20.2	335.0	7.0
374	4.138	37.97	20.2	335.0	13.8

In [40]:
```
#Box Plot, Distribution Plot and Scatter Plot for RM
plt.figure(figsize=(20,3))

plt.subplot(1,3,1)
sns.boxplot(df2.RM,color='#005030')
plt.title('Box Plot of MEDV')

plt.subplot(1,3,2)
sns.distplot(a=df2.RM,color='#500050')
plt.title('Distribution Plot of MEDV')

plt.subplot(1,3,3)
sns.scatterplot(x=df2.RM,y=df2.MEDV)
plt.title('Scatter Plot of RM vs MEDV')

plt.show()
```



In [42]:
```
RM_Q3 = d['RM']['75%']
RM_Q1 = d['RM']['25%']
RM_IQR = RM_Q3 - RM_Q1
RM_UV = RM_Q3 + 1.5*RM_IQR
RM_LV = RM_Q1 - 1.5*RM_IQR

df2[df2['RM']<RM_LV].sort_values(by=['RM','MEDV'])
```

Out[42]:

	RM	LSTAT	PTRATIO	TAX	MEDV
365	3.561	7.12	20.2	294.139785	27.5
367	3.863	13.33	20.2	330.770270	23.1
406	4.138	23.34	20.2	338.636364	11.9
374	4.138	37.97	20.2	335.000000	13.8
384	4.368	30.63	20.2	335.000000	8.8
414	4.519	36.98	20.2	335.000000	7.0
412	4.628	34.37	20.2	335.000000	17.9
386	4.652	28.28	20.2	338.636364	10.5

In [43]:
```
print(f'Shape of dataset before removing data points: {df2.shape}')
df3 = df2.drop(axis=0,index=[365,367])
print(f'Shape of dataset before removing data points: {df3.shape}')
```

```
Shape of dataset before removing data points: (490, 5)
Shape of dataset before removing data points: (488, 5)
```

In [44]:
```
df3[df3['RM']>RM_UV].sort_values(by=['RM','MEDV'])
```

Out[44]:

	RM	LSTAT	PTRATIO	TAX	MEDV
180	7.765	7.56	17.8	193.000000	39.8
98	7.820	3.57	18.0	276.000000	43.8
280	7.820	3.76	14.9	216.000000	45.4
203	7.853	3.81	14.7	224.000000	48.5
226	8.040	11.43	17.4	307.000000	37.6
97	8.069	4.21	18.0	276.000000	38.7
233	8.247	3.95	17.4	307.000000	48.3
253	8.259	3.54	19.1	330.000000	42.8
224	8.266	4.14	17.4	307.000000	44.8
232	8.337	2.47	17.4	307.000000	41.7
262	8.398	5.91	13.0	264.000000	48.8
364	8.780	5.29	20.2	294.139785	21.9

In [45]:
```
print(f'Shape of dataset before removing data points: {df3.shape}')
df3 = df3.drop(axis=0,index=[364])
print(f'Shape of dataset before removing data points: {df3.shape}')
```

```
Shape of dataset before removing data points: (488, 5)
Shape of dataset before removing data points: (487, 5)
```

In [46]:
```
#Applying linear Regression
#Now will split our dataset into Dependent variable and Independent va

X = df3.iloc[:,0:4].values
y = df3.iloc[:,-1:].values  #MEDV
```

In [47]:
```
print(f"Shape of Dependent Variable X = {X.shape}")
print(f"Shape of Independent Variable y = {y.shape}")
```

```
Shape of Dependent Variable X = (487, 4)
Shape of Independent Variable y = (487, 1)
```

In [48]:
```

def FeatureScaling(X):
    """
    is function takes an array as an input, which needs to be scaled d
    Apply Standardization technique to it and scale down the features 

    Input <- 2 dimensional numpy array
    Returns -> Numpy array after applying Feature Scaling
    """
    mean = np.mean(X,axis=0)
    std = np.std(X,axis=0)
    for i in range(X.shape[1]):
        X[:,i] = (X[:,i]-mean[i])/std[i]
    return X
```


In [49]:
```
X = FeatureScaling(X)
```

In [50]:
```
m,n = X.shape
X = np.append(arr=np.ones((m,1)),values=X,axis=1)
```

In [51]:
```
#Now we will spit our data into Train set and Test Set

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,

print(f"Shape of X_train = {X_train.shape}")
print(f"Shape of X_test = {X_test.shape}")
print(f"Shape of y_train = {y_train.shape}")
print(f"Shape of y_test = {y_test.shape}")
```

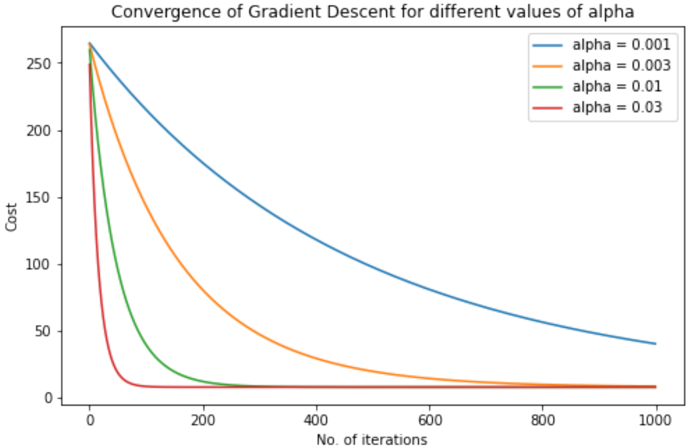
```
Shape of X_train = (389, 5)
Shape of X_test = (98, 5)
Shape of y_train = (389, 1)
Shape of y_test = (98, 1)
```

In [52]:
```
#ComputeCost function determines the cost (sum of squared errors)

def ComputeCost(X,y,theta):
    """
    This function takes three inputs and uses the Cost Function to det
    actual values)
    Cost Function: Sum of square of error in predicted values divided
    J = 1/(2*m) *  Summation(Square(Predicted values - Actual values))

    Input <- Take three numoy array X,y and theta
    Return -> The cost calculated from the Cost Function
    """
    m=X.shape[0] #number of data points in the set
    J = (1/(2*m)) * np.sum((X.dot(theta) - y)**2)
    return J
```

In [53]:
```
#Gradient Descent Algorithm to minimize the Cost and find best paramet

def GradientDescent(X,y,theta,alpha,no_of_iters):
    m=X.shape[0]
    J_Cost = []
    for i in range(no_of_iters):
        error = np.dot(X.transpose(),(X.dot(theta)-y))
        theta = theta - alpha * (1/m) * error
        J_Cost.append(ComputeCost(X,y,theta))

    return theta, np.array(J_Cost)
```

In [54]:
```
iters = 1000

alpha1 = 0.001
theta1 = np.zeros((X_train.shape[1],1))
theta1, J_Costs1 = GradientDescent(X_train,y_train,theta1,alpha1,iters

alpha2 = 0.003
theta2 = np.zeros((X_train.shape[1],1))
theta2, J_Costs2 = GradientDescent(X_train,y_train,theta2,alpha2,iters

alpha3 = 0.01
theta3 = np.zeros((X_train.shape[1],1))
theta3, J_Costs3 = GradientDescent(X_train,y_train,theta3,alpha3,iters

alpha4 = 0.03
theta4 = np.zeros((X_train.shape[1],1))
theta4, J_Costs4 = GradientDescent(X_train,y_train,theta4,alpha4,iters
```

In [55]:
```
plt.figure(figsize=(8,5))
plt.plot(J_Costs1,label = 'alpha = 0.001')
plt.plot(J_Costs2,label = 'alpha = 0.003')
plt.plot(J_Costs3,label = 'alpha = 0.01')
plt.plot(J_Costs4,label = 'alpha = 0.03')
plt.title('Convergence of Gradient Descent for different values of alp
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.legend()
plt.show()
```



In [56]:
```
theta4 #Primts values of bias,rmstat,ptraio,tax,lstat in sequence
```

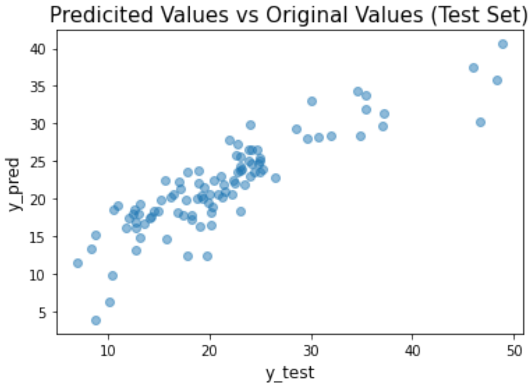
Out[56]:
```
array([[21.73546011],
       [ 2.86076194],
       [-2.87113215],
       [-1.96051041],
       [-1.078047  ]])
```


In [57]:
```
def Predict(X,theta):
    """
    This function predicts the result for the unseen data
    """
    y_pred = X.dot(theta)
    return y_pred
```

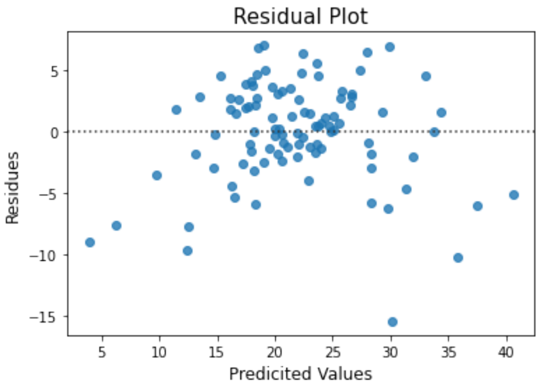
In [58]:
```
y_pred = Predict(X_test,theta4)
y_pred[:5]
```

Out[58]:
```
array([[23.54777745],
       [28.09399088],
       [16.16554384],
       [18.97965458],
       [17.66976105]])
```

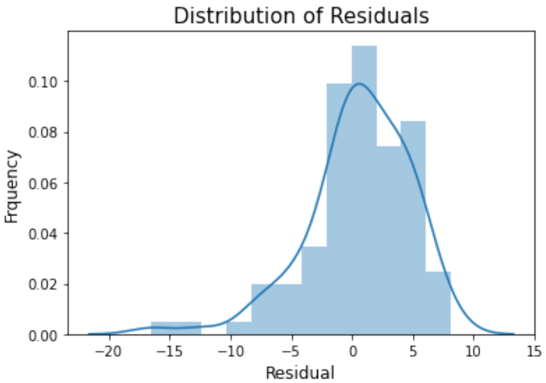
In [59]:
```
plt.scatter(x=y_test,y=y_pred,alpha=0.5)
plt.xlabel('y_test',size=12)
plt.ylabel('y_pred',size=12)
plt.title('Predicited Values vs Original Values (Test Set)',size=15)
plt.show()
```



In [62]:
```
sns.residplot(x=y_pred,y=(y_pred-y_test))
plt.xlabel('Predicited Values',size=12)
plt.ylabel("Residues",size=12)
plt.title('Residual Plot',size=15)
plt.show()
```



In [63]:
```
sns.distplot(y_pred-y_test)
plt.xlabel('Residual',size=12)
plt.ylabel('Frquency',size=12)
plt.title('Distribution of Residuals',size=15)
plt.show()
```



In [64]:
```
from sklearn import metrics
r2= metrics.r2_score(y_test,y_pred)  #Differnent Error rates
N,p = X_test.shape
adj_r2 = 1-((1-r2)*(N-1))/(N-p-1)
print(f'R^2 = {r2}')
print(f'Adjusted R^2 = {adj_r2}')
```

```
R^2 = 0.7467879874493435
Adjusted R^2 = 0.7330264650281122
```


In [65]:
```
from sklearn import metrics
mse = metrics.mean_squared_error(y_test,y_pred)
mae = metrics.mean_absolute_error(y_test,y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
print(f'Mean Squared Error: {mse}',f'Mean Absolute Error: {mae}',f'Roo
```

```
Mean Squared Error: 18.27759183945454
Mean Absolute Error: 3.24036598158337
Root Mean Squared Error: 4.275230033513348
```

In [66]:
```
#coefficients of regression model
coeff=np.array([y for x in theta4 for y in x]).round(2)
features=['Bias','RM','TAX','PTRATIO','LSTAT']
eqn = 'MEDV = '
for f,c in zip(features,coeff):
    eqn+=f" + ({c} * {f})";

print(eqn)
```

```
MEDV =  + (21.74 * Bias) + (2.86 * RM) + (-2.87 * TAX) + (-1.96 * PTR
ATIO) + (-1.08 * LSTAT)
```

In [67]:
```
sns.barplot(x=features,y=coeff)
plt.ylim([-5,25])
plt.xlabel('Coefficient Names',size=12)
plt.ylabel('Coefficient Values',size=12)
plt.title('Visualising Regression Coefficients',size=15)
plt.show()
```