In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.datasets import load_boston
```

In [2]:
```
boston = load_boston()
```

In [3]:
```
boston
```

Out[3]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15.
,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n-------------------
```

```
--------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506 \n\n
:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute
14) is usually the target.\n\n    :Attribute Information (in order):\n        - CR
IM     per capita crime rate by town\n        - ZN       proportion of residential
land zoned for lots over 25,000 sq.ft.\n        - INDUS    proportion of non-retai
l business acres per town\n        - CHAS     Charles River dummy variable (= 1 if
tract bounds river; 0 otherwise)\n        - NOX      nitric oxides concentration
(parts per 10 million)\n        - RM       average number of rooms per dwelling\n
- AGE      proportion of owner-occupied units built prior to 1940\n        - DIS
weighted distances to five Boston employment centres\n        - RAD      index of
accessibility to radial highways\n        - TAX      full-value property-tax rate
per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n        - B        10
00(Bk - 0.63)^2 where Bk is the proportion of black people by town\n        - LSTA
T    % lower status of the population\n        - MEDV     Median value of owner-oc
cupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Ha
rrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttp
s://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset w
as taken from the StatLib library which is maintained at Carnegie Mellon Universit
y.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\npr
ices and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81
-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley,
1980.   N.B. Various transformations are used in the table on\npages 244-261 of th
e latter.\n\nThe Boston house-price data has been used in many machine learning pa
pers that address regression\nproblems.   \n     \n.. topic:: References\n\n   - B
elsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and So
urces of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining I
nstance-Based and Model-Based Learning. In Proceedings on the Tenth International
Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Mor
gan Kaufmann.\n",
 'filename': 'boston_house_prices.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [4]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attrib
ute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.f
t.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 othe
rwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Me
llon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that add
ress regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data
and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proc
eedings on the Tenth International Conference of Machine Learning, 236-243, Univer
sity of Massachusetts, Amherst. Morgan Kaufmann.
```

In [5]:
```
boston_df=pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.1
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.9
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.6
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.0
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.6
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.4
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.8

506 rows × 13 columns

In [6]:
```
# This is used to create taget values into MedV columns
boston_df["MEDV"]=boston.target
# Since boston is a dictionary .target giving us the target, .data provding us dat
```

In [7]:
```
# Medv: Target last column
boston_df.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Data Wrangling

- Cleasiing the dataset
- Handling missing values

In [8]:
```
boston_df.isnull().sum()
```

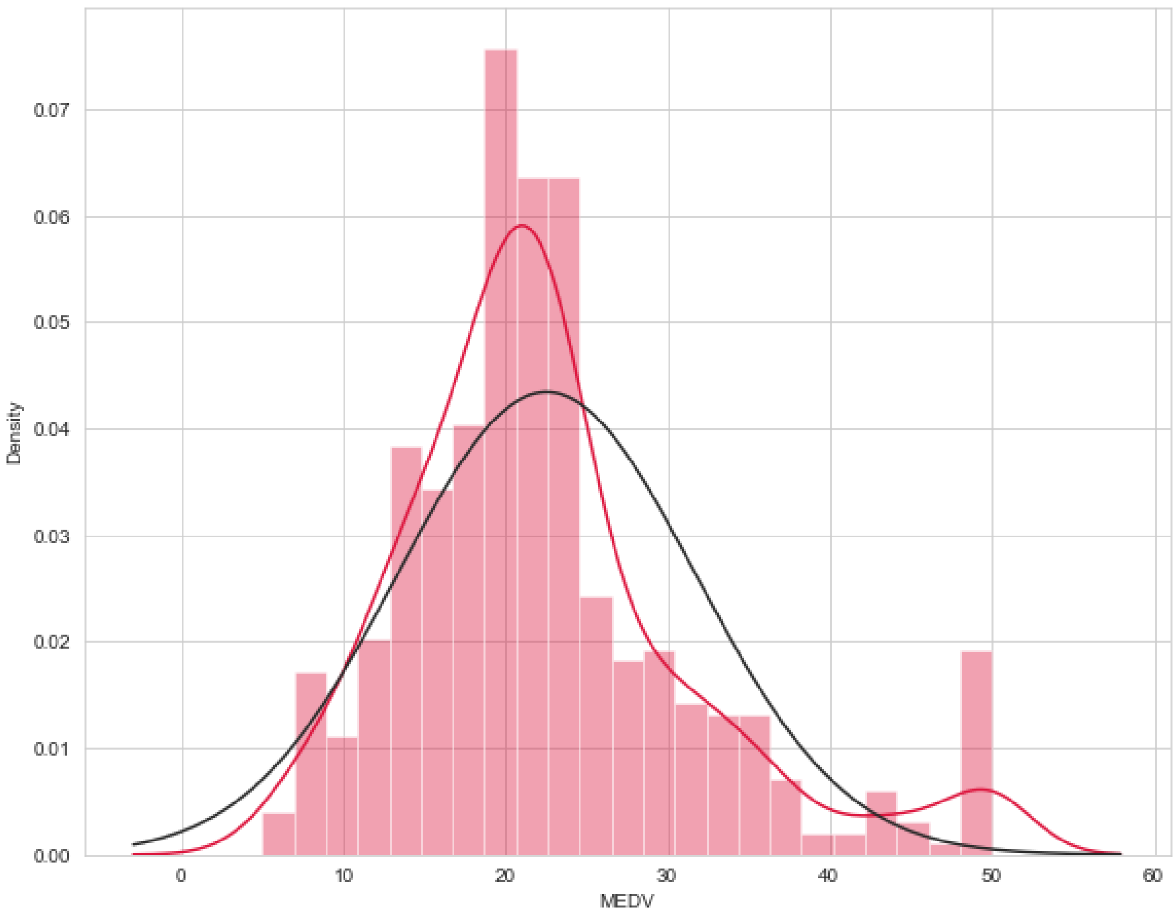
Out[8]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

Linear Regression - Assmptions

- Target (Medv) must be normally distributed

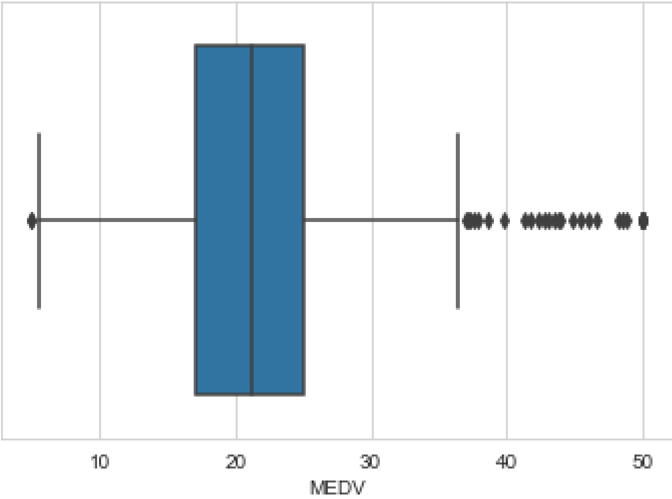
In [9]:
```
from scipy.stats import norm
sns.set_style("whitegrid")
plt.figure(figsize=(10,8))
sns.distplot(boston_df["MEDV"],fit=norm, color="crimson")
```

Out[9]:
```
<AxesSubplot:xlabel='MEDV', ylabel='Density'>
```



In [10]:
```
# For searching out the outliers, the median line is also in middle
sns.boxplot(boston_df["MEDV"])
plt.figure(figsize=(10,8))
```

Out[10]:
```
<Figure size 720x576 with 0 Axes>
```

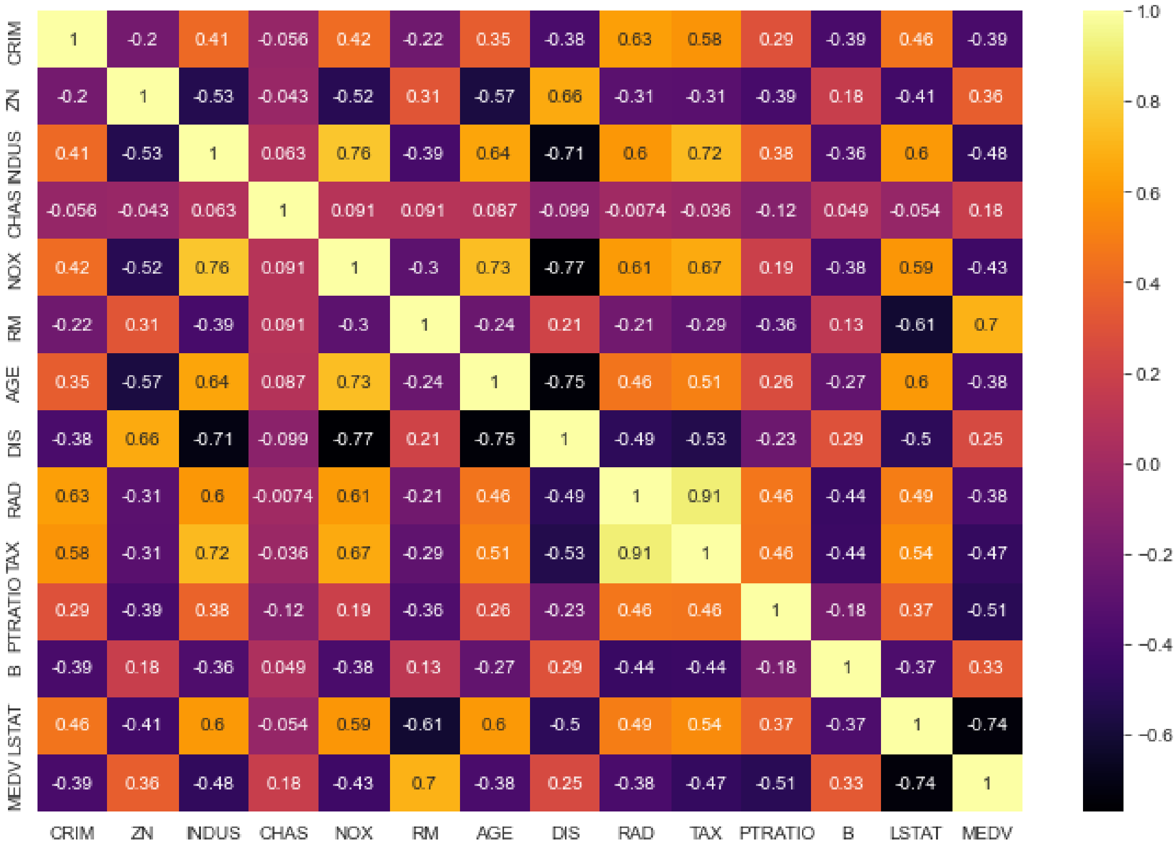



```
<Figure size 720x576 with 0 Axes>
```

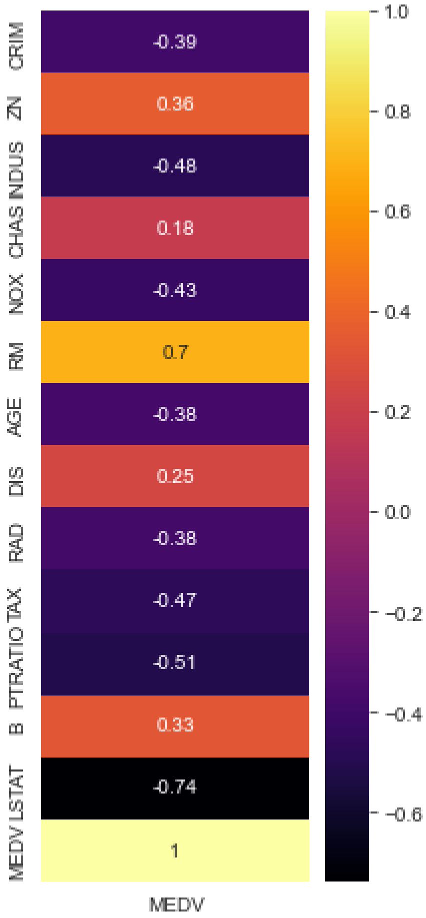
2nd feature= All the features are stastically indepedent to each other

Multi colinearlity - corelation b/w features, corelation with target columns

In [11]:
```python
plt.figure(figsize = (12,8))
sns.heatmap(boston_df.corr(), annot = True, cmap = 'inferno')
plt.show()
```



In [12]:
```python
plt.figure(figsize = (3,8))
sns.heatmap(boston_df.corr()[["MEDV"]], annot = True, cmap = 'inferno')
plt.show()
```



In [13]:
```
### We will compare any other feature with respect to target Medv except then Rad (
# is there any feature closer to 0, we will drop the feature apart from multicoline
#as of now no weakest feature
```

Feature and Target

In [14]:
```
# Feature - RAD is dropped due to muticolinearity with tax

X_feature = boston_df.drop(["RAD","MEDV"],axis=1)

# Feature

Y = boston_df["MEDV"]
```

Train and Test split

In [15]:
```
from sklearn.model_selection import train_test_split
# This function is useful in splitting the data randomly into train and test, rand
x_train,x_test,y_train,y_test = train_test_split(X_feature, Y, test_size=.2, rando
# test_size = .2 => 80% is train data , 20 % test data
```

Data Preprocessing

- Scaling Down the features range into same range.

In [16]:
```
x_train
# every column in the data are in diffrent range
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	TAX	PTRATIO	B	LSTAT
454	9.51363	0.0	18.10	0.0	0.7130	6.728	94.1	2.4961	666.0	20.2	6.68	18.71
471	4.03841	0.0	18.10	0.0	0.5320	6.229	90.7	3.0993	666.0	20.2	395.33	12.87
281	0.03705	20.0	3.33	0.0	0.4429	6.968	37.2	5.2447	216.0	14.9	392.23	4.59
477	15.02340	0.0	18.10	0.0	0.6140	5.304	97.3	2.1007	666.0	20.2	349.48	24.91
107	0.13117	0.0	8.56	0.0	0.5200	6.127	85.2	2.1224	384.0	20.9	387.69	14.09
...	...	...	...	...	...	...	...	...	...	...	...	...
440	22.05110	0.0	18.10	0.0	0.7400	5.818	92.4	1.8662	666.0	20.2	391.45	22.11
131	1.19294	0.0	21.89	0.0	0.6240	6.326	97.7	2.2710	437.0	21.2	396.90	12.26
249	0.19073	22.0	5.86	0.0	0.4310	6.718	17.5	7.8265	330.0	19.1	393.74	6.56
152	1.12658	0.0	19.58	1.0	0.8710	5.012	88.0	1.6102	403.0	14.7	343.28	12.12
362	3.67822	0.0	18.10	0.0	0.7700	5.362	96.2	2.1036	666.0	20.2	380.79	10.19

404 rows × 12 columns

In [17]:

```
# We have to fix data of every column into a single range data... we have to fix tl

from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler(feature_range=(0,1))
# it means min value will be replaced by 0 and max value by 1, we could take range
# First fit and then transfrom, fit- first get train and transform- apply changes
x_train_scaler=scaler.fit_transform(x_train)
x_test_scaler=scaler.transform(x_test)
```

In [18]:

```
x_train_scaler
```

Out[18]:

```
array([[1.06859872e-01, 0.00000000e+00, 6.46627566e-01, ...,
        8.08510638e-01, 1.60371174e-02, 4.65742025e-01],
       [4.53197194e-02, 0.00000000e+00, 6.46627566e-01, ...,
        8.08510638e-01, 9.96041152e-01, 3.03744799e-01],
       [3.45397791e-04, 2.10526316e-01, 1.05205279e-01, ...,
        2.44680851e-01, 9.88224318e-01, 7.40638003e-02],
       ...,
       [2.07272394e-03, 2.31578947e-01, 1.97947214e-01, ...,
        6.91489362e-01, 9.92031873e-01, 1.28710125e-01],
       [1.25914523e-02, 0.00000000e+00, 7.00879765e-01, ...,
        2.23404255e-01, 8.64793989e-01, 2.82940361e-01],
       [4.12712707e-02, 0.00000000e+00, 6.46627566e-01, ...,
        8.08510638e-01, 9.59377679e-01, 2.29403606e-01]])
```

Linear Regression

In [19]:

```
# Step1. Declare the ML algorithm

from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
```

In [20]:

```
# Step 2. Training of algorithm with Train data and Train label ( it means target)

lin_reg.fit (x_train_scaler,y_train)
```

Out[20]:
```
LinearRegression()
```

In [21]:
```
# Once the model is trained, let's take test dataset
# Test score (score must be as closer to 1)- R2 score (coeffiecnt of determination,
# Score function is used with features and target
# score function returns the coeffiecnet of determination r_2 of the prediction or
# The coefficient of determination (R²) is a number between 0 and 1 that measures,h

lin_reg.score(x_test_scaler,y_test)
```

Out[21]:
```
0.784004437020033
```

Regression Metrics- calculating an error score to summarize the predictive skill of a model

In [22]:
```
# Predictions
yhat = lin_reg.predict(x_test_scaler)
```

In [23]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

In [24]:
```
# compairing score of actual & predicted
r2_score(y_test, yhat)
# r2 score function will take actual- Target value and predicted value
```

Out[24]:
```
0.784004437020033
```

In [25]:
```
# Mean Absulute error

mean_squared_error(y_test, yhat)
```

Out[25]:
```
17.8746702547278
```

In [26]:
```
# Mean absulute error

mean_absolute_error(y_test, yhat)
```

Out[26]:
```
3.019989017600162
```

In [27]:
```
# RMSE= root mean sqaurd error
# RMSE value must be closer to actual y range
np.sqrt(mean_squared_error(y_test, yhat))
```

Out[27]:
```
4.2278446346487
```

Make Predictions

In [28]:
```
# scalar data is a tranform data
X_feature
```

Out[28]:

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

506 rows × 12 columns

In [29]:

```
#new prediction, copy 1st row data
X_new=[[0.00632,18.0,2.31,0.0,0.538,6.575,65.2,4.0900,296.0,15.3,396.90,4.98]]
# we can change the value, but we have to make sure it comes within the range itse
```

In [30]:

```
X_new_scaler=scaler.transform(X_new)
```

In [31]:

```
lin_reg.predict(X_new_scaler)
```

Out[31]:

```
array([31.23927307])
```

In [32]:

```
# It is indicating my predicted value of medv
```

In [ ]: