In [ ]:
```
from sklearn import datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [ ]:
```
boston= datasets.load_boston ()
df = pd.DataFrame(boston.data ,columns = boston.feature_names)
df['price']=boston.target
```

```
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning:
Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

Attribute Information (in order):

```
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    - 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
    - price     Median value of owner-occupied homes in $1000's
```

In [ ]:
```
df.head()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

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

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

In [ ]:
```
df.head().to_csv("head_boston.csv")
```

In [ ]:
```
df_head = pd.read_csv("head_boston.csv")
```

In [ ]:
```
df_head.head()
```

Out[ ]:

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
0	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	24.0
1	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	21.6
2	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	34.7
3	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	33.4
4	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	36.2

In [ ]:

```
df_head.drop(df_head.columns[[0]], axis=1, inplace=True)
```

Here we dropped the unnecessery indexing column as we have an indexing column already

In [ ]:

```
df_head
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

In [ ]:

```
df_head.iloc[0:5, [0, 1, 3,9,10,11,13]] += 44
```

In [ ]:

```
df_head.iloc[0:5, [2, 4, 5,6,7,11,12]] += .44
```

We added 44 to the first 5 rows in the columns 0, 1, 3,9,10,11,13 and added 0.44 in the columns 2, 4, 5,6,7,11,12

In [ ]:

```
df_head
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
0	44.00632	62.0	2.75	44.0	0.978	7.015	65.64	4.5300	1.0	340.0	59.3	441.34	5.42	68.0
1	44.02731	44.0	7.51	44.0	0.909	6.861	79.34	5.4071	2.0	286.0	61.8	441.34	9.58	65.6
2	44.02729	44.0	7.51	44.0	0.909	7.625	61.54	5.4071	2.0	286.0	61.8	437.27	4.47	78.7
3	44.03237	44.0	2.62	44.0	0.898	7.438	46.24	6.5022	3.0	266.0	62.7	439.07	3.38	77.4
4	44.06905	44.0	2.62	44.0	0.898	7.587	54.64	6.5022	3.0	266.0	62.7	441.34	5.77	80.2

In [ ]:

```
df_new = df.append(df_head,ignore_index=True)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_30944\3573584504.py:1: FutureWarning: The frame.append method is deprecated
and will be removed from pandas in a future version. Use pandas.concat instead.
  df_new = df.append(df_head,ignore_index=True)
```

We tried to append the newly created rows with our main dataset

In [ ]:

```
df_new
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.20	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.90	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.10	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.80	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.20	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
506	44.00632	62.0	2.75	44.0	0.978	7.015	65.64	4.5300	1.0	340.0	59.3	441.34	5.42	68.0
507	44.02731	44.0	7.51	44.0	0.909	6.861	79.34	5.4071	2.0	286.0	61.8	441.34	9.58	65.6
508	44.02729	44.0	7.51	44.0	0.909	7.625	61.54	5.4071	2.0	286.0	61.8	437.27	4.47	78.7
509	44.03237	44.0	2.62	44.0	0.898	7.438	46.24	6.5022	3.0	266.0	62.7	439.07	3.38	77.4
510	44.06905	44.0	2.62	44.0	0.898	7.587	54.64	6.5022	3.0	266.0	62.7	441.34	5.77	80.2

511 rows × 14 columns

In [ ]:

```
df_new.shape
```

Out[ ]:

```
(511, 14)
```

In [ ]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 511 entries, 0 to 510
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     511 non-null    float64
 1   ZN       511 non-null    float64
 2   INDUS    511 non-null    float64
 3   CHAS     511 non-null    float64
 4   NOX      511 non-null    float64
 5   RM       511 non-null    float64
 6   AGE      511 non-null    float64
 7   DIS      511 non-null    float64
 8   RAD      511 non-null    float64
 9   TAX      511 non-null    float64
 10  PTRATIO  511 non-null    float64
 11  B        511 non-null    float64
 12  LSTAT    511 non-null    float64
 13  price    511 non-null    float64
dtypes: float64(14)
memory usage: 56.0 KB
```

In [ ]:
```
df_new.describe()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
count	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	5
mean	4.009012	11.718200	11.072838	0.499022	0.558254	6.294620	68.505479	3.813386	9.477495	407.068493	18.878278	3
std	9.440396	23.491693	6.860968	4.335840	0.120785	0.707026	28.040781	2.104803	8.694994	168.142730	4.772575	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	
25%	0.082325	0.000000	5.130000	0.000000	0.449000	5.887500	45.250000	2.102150	4.000000	279.000000	17.400000	3
50%	0.263630	0.000000	9.690000	0.000000	0.538000	6.211000	77.000000	3.262800	5.000000	330.000000	19.100000	3
75%	3.805910	17.750000	18.100000	0.000000	0.631000	6.630500	93.950000	5.222850	24.000000	666.000000	20.200000	3
max	88.976200	100.000000	27.740000	44.000000	0.978000	8.780000	100.000000	12.126500	24.000000	711.000000	62.700000	4

In [ ]:
```
df_new.isnull()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
506	False	False	False	False	False	False	False	False	False	False	False	False	False	False
507	False	False	False	False	False	False	False	False	False	False	False	False	False	False
508	False	False	False	False	False	False	False	False	False	False	False	False	False	False
509	False	False	False	False	False	False	False	False	False	False	False	False	False	False
510	False	False	False	False	False	False	False	False	False	False	False	False	False	False

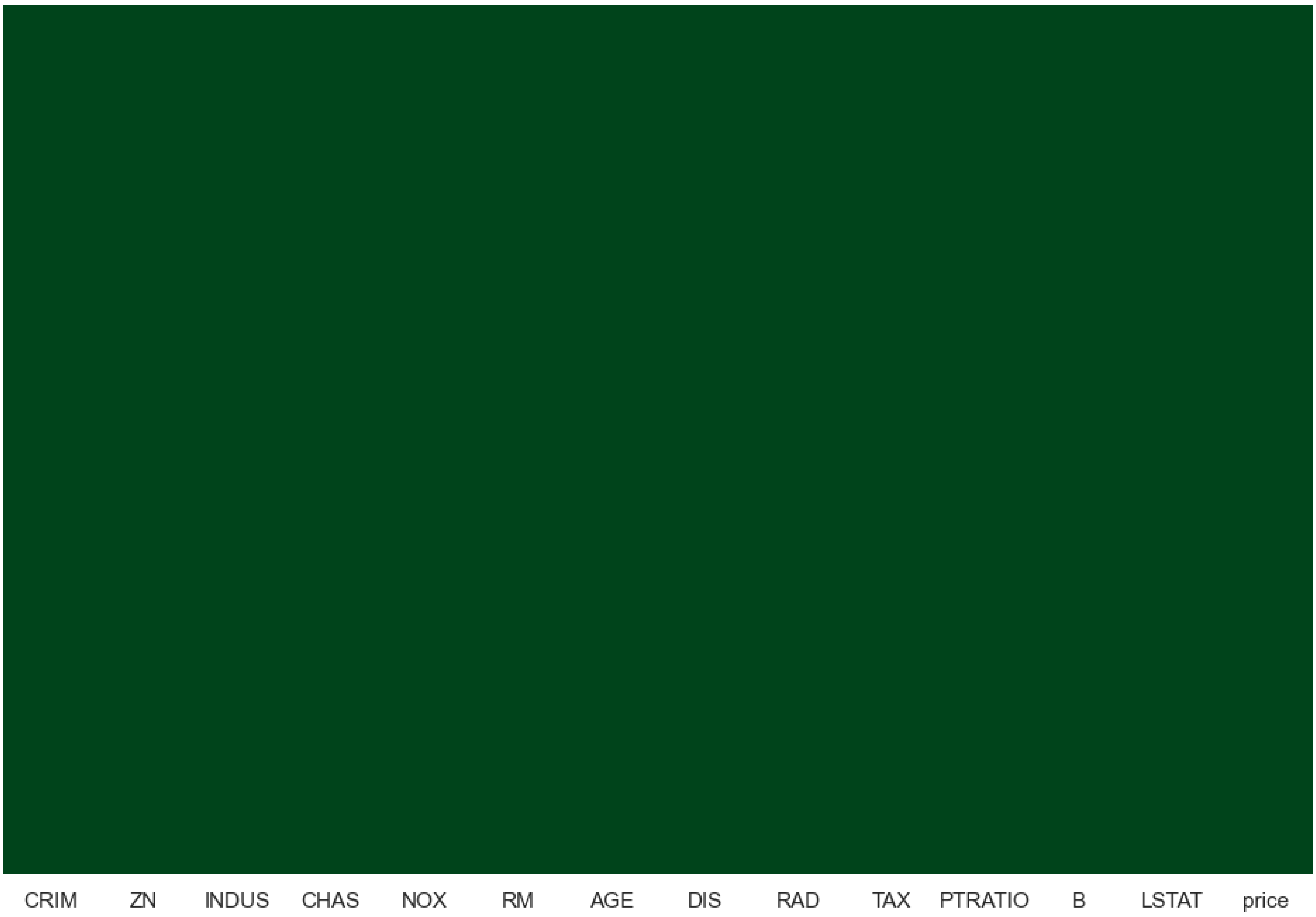
511 rows × 14 columns

In [ ]:
```
df_new.isnull().sum()
```

Out[ ]:
```
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
price      0
dtype: int64
```

In [ ]:
```
sns.heatmap(df_new.isnull(),yticklabels=False, cbar = False, cmap = "Greens_r")
```

Out[ ]:
```
<AxesSubplot:>
```



There are no null values in this dataset

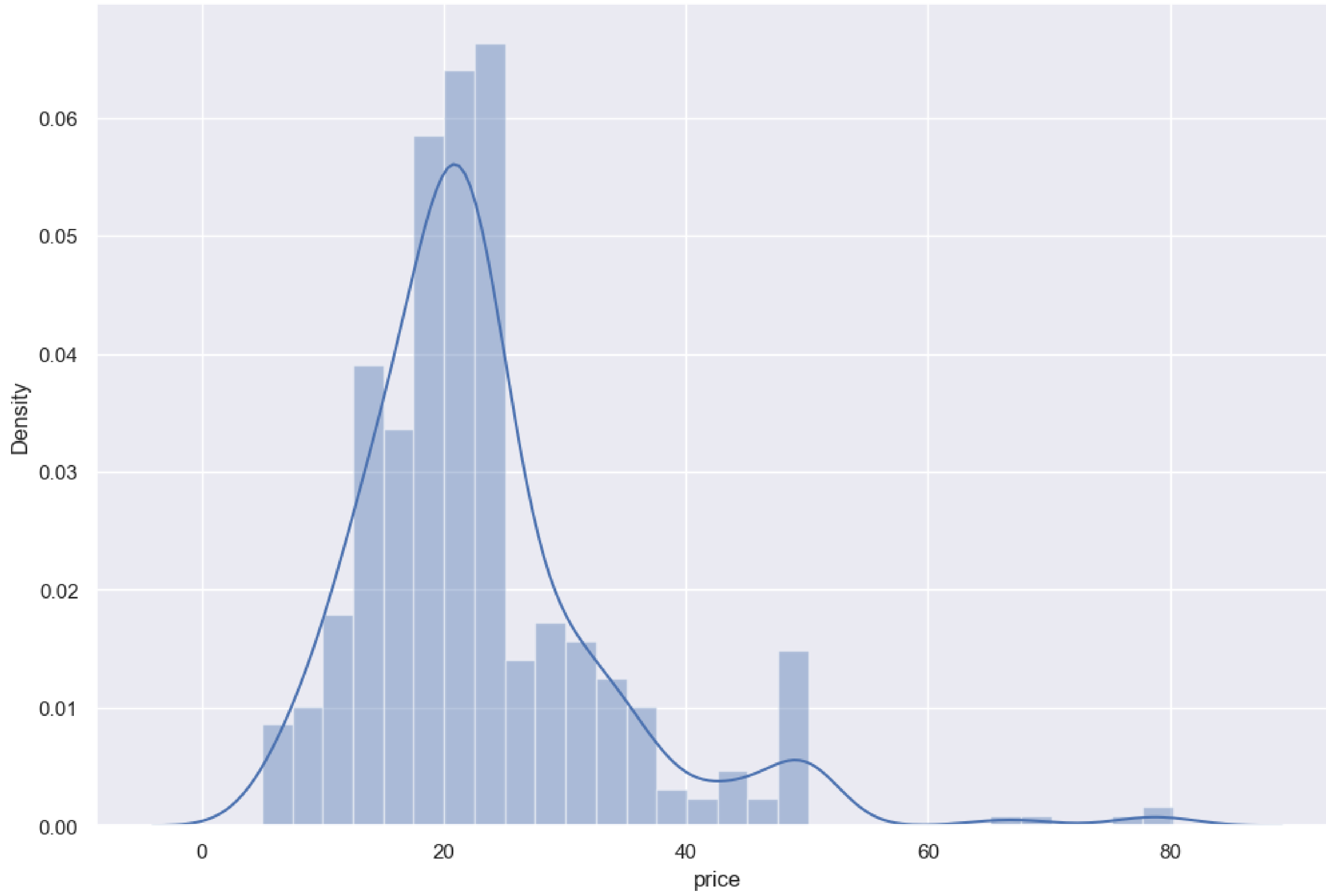
In [ ]:
```
####Calculate some measures####

minimum_price = df_new['price'].min()
maximum_price = df_new['price'].max()
mean_price = df_new['price'].mean()
median_price = df_new['price'].median()
std_price = df_new['price'].std()
print("Statistics for Boston housing dataset:\n")
print("Minimum price: ${}".format(minimum_price))
print("Maximum price: ${}".format(maximum_price))
print("Mean price: ${}".format(mean_price))
print("Median price ${}".format(median_price))
print("Standard deviation of prices: ${}".format(std_price))
```

```
Statistics for Boston housing dataset:

Minimum price: $5.0
Maximum price: $80.2
Mean price: $23.036203522504895
Median price $21.2
Standard deviation of prices: $10.478691752915296
```

Let's plot the distribution of the target variable Price. We will use the `distplot` function from the `seaborn` library.

In [ ]:
```
sns.set(rc={'figure.figsize':(12,8)})
sns.distplot(df_new["price"], bins=30)
plt.show()
```

```
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `
distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displ
ot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



We can see that the price (target) is normally distributed with some outliers .

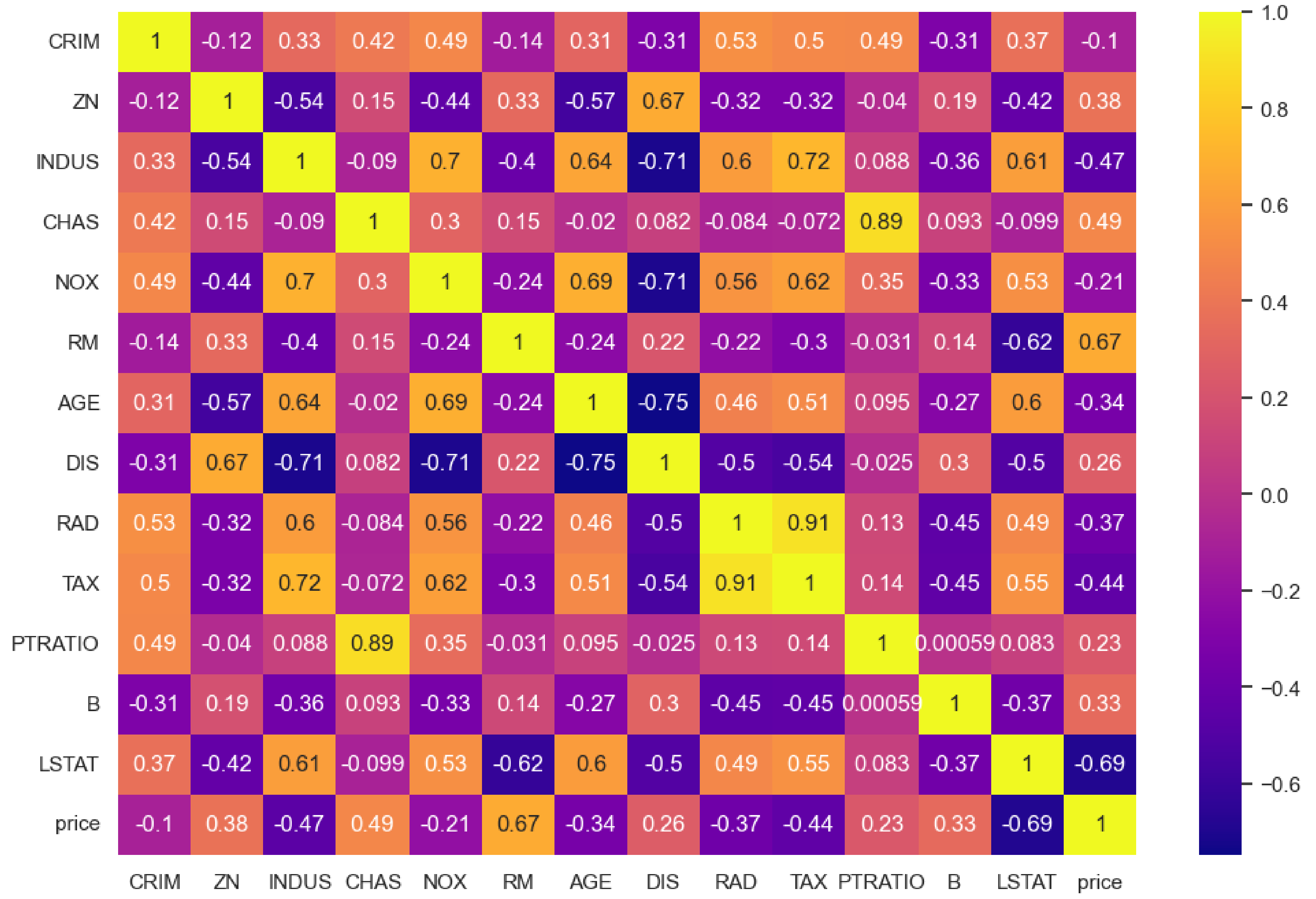
In [ ]:
```
##Lets see the correlations and plot them by using Seaborn

corr= df_new.corr()
corr
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
CRIM	1.000000	-0.115452	0.327200	0.418180	0.489527	-0.136571	0.308947	-0.305663	0.530002	0.497481	0.494938	-0.309702	0.
ZN	-0.115452	1.000000	-0.539408	0.149266	-0.441386	0.325793	-0.565615	0.665960	-0.319961	-0.320140	-0.039846	0.186406	-0.
INDUS	0.327200	-0.539408	1.000000	-0.090035	0.697333	-0.398991	0.644099	-0.709929	0.597819	0.721837	0.088488	-0.362223	0.
CHAS	0.418180	0.149266	-0.090035	1.000000	0.301258	0.147241	-0.019850	0.081852	-0.083568	-0.071938	0.887236	0.092762	-0.
NOX	0.489527	-0.441386	0.697333	0.301258	1.000000	-0.243646	0.690458	-0.705766	0.556769	0.615710	0.345414	-0.334599	0.
RM	-0.136571	0.325793	-0.398991	0.147241	-0.243646	1.000000	-0.242176	0.215468	-0.218405	-0.298405	-0.031192	0.138893	-0.
AGE	0.308947	-0.565615	0.644099	-0.019850	0.690458	-0.242176	1.000000	-0.746835	0.455797	0.506626	0.095239	-0.274347	0.
DIS	-0.305663	0.665960	-0.709929	0.081852	-0.705766	0.215468	-0.746835	1.000000	-0.497649	-0.537329	-0.025342	0.296921	-0.
RAD	0.530002	-0.319961	0.597819	-0.083568	0.556769	-0.218405	0.455797	-0.497649	1.000000	0.910392	0.134971	-0.448556	0.
TAX	0.497481	-0.320140	0.721837	-0.071938	0.615710	-0.298405	0.506626	-0.537329	0.910392	1.000000	0.144650	-0.445172	0.
PTRATIO	0.494938	-0.039846	0.088488	0.887236	0.345414	-0.031192	0.095239	-0.025342	0.134971	0.144650	1.000000	0.000593	0.
B	-0.309702	0.186406	-0.362223	0.092762	-0.334599	0.138893	-0.274347	0.296921	-0.448556	-0.445172	0.000593	1.000000	-0.
LSTAT	0.370701	-0.420555	0.607250	-0.098561	0.532909	-0.618444	0.602068	-0.500973	0.492422	0.546613	0.082654	-0.371353	1.
price	-0.103407	0.383667	-0.466270	0.491848	-0.213572	0.671791	-0.342725	0.261161	-0.372087	-0.442586	0.232216	0.333577	-0.

In [ ]:
```
plt.figure (figsize=(12,8))
sns.heatmap(corr, annot = True, cmap = 'plasma')
```

Out[ ]:
```
<AxesSubplot:>
```

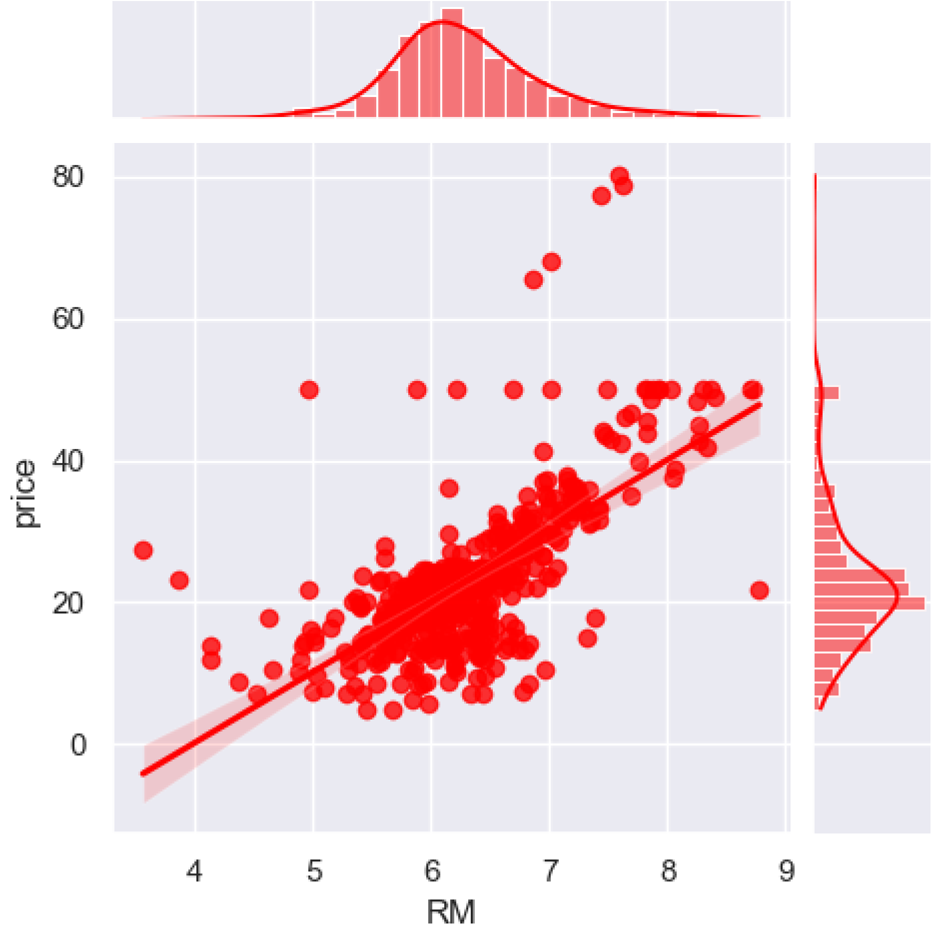


We can see here is the Positive relationship between **Price** and **B,PTRATIO,DIS,RM,CHAS** and **ZN**

and Negative relationship between **Price** and **LSTAT,TAX,RAD,AGE,NOX,INDUS** and **CRIM**

In [ ]:
```
###Correlation between RM and Price

plt.figure (figsize=(15,9))
sns.jointplot(x = df_new["RM"], y = df_new["price"], data = df_new, kind = 'reg', color = 'red', height = 5)
```

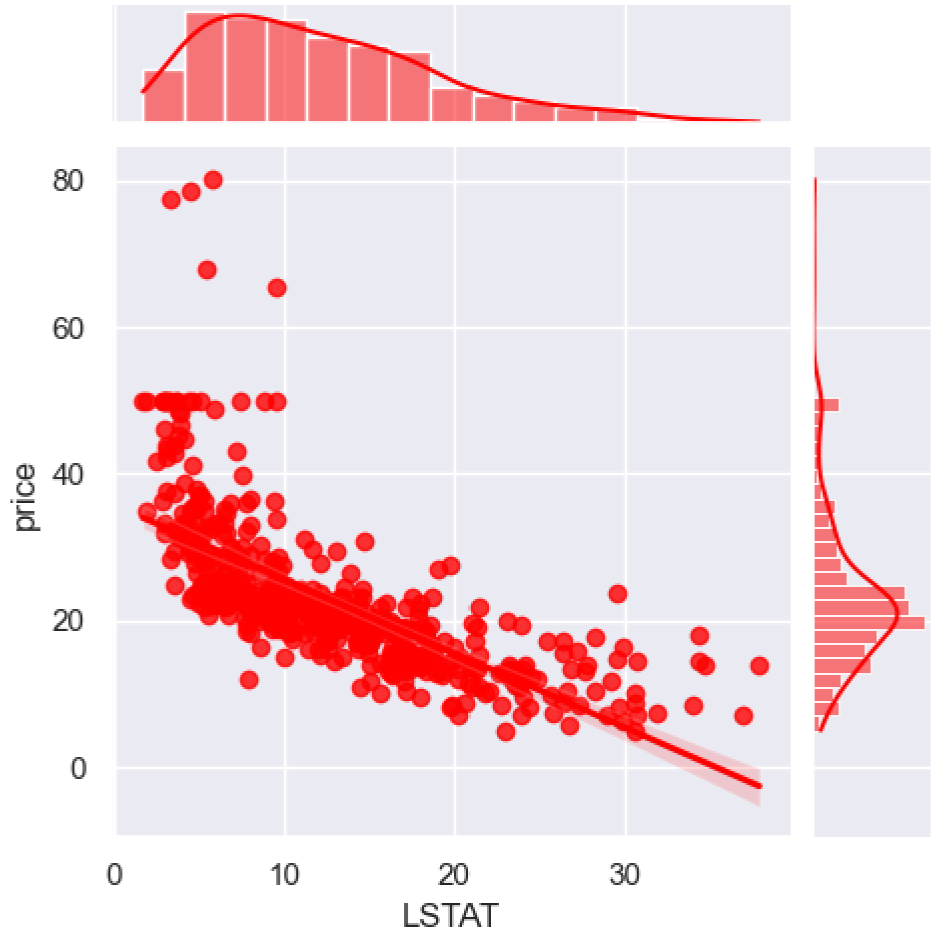
Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x26566b9d150>
```
```
<Figure size 1500x900 with 0 Axes>
```



Here can see the Positive correlation between Price and RM in the dataset and data is normally distributed here.

In [ ]:
```
###Correlation between LSTAT and Price
plt.figure (figsize=(15,9))
sns.jointplot(x = df_new["LSTAT"], y = df_new["price"], data = df_new, kind = 'reg', color = 'red', height = 5)
```

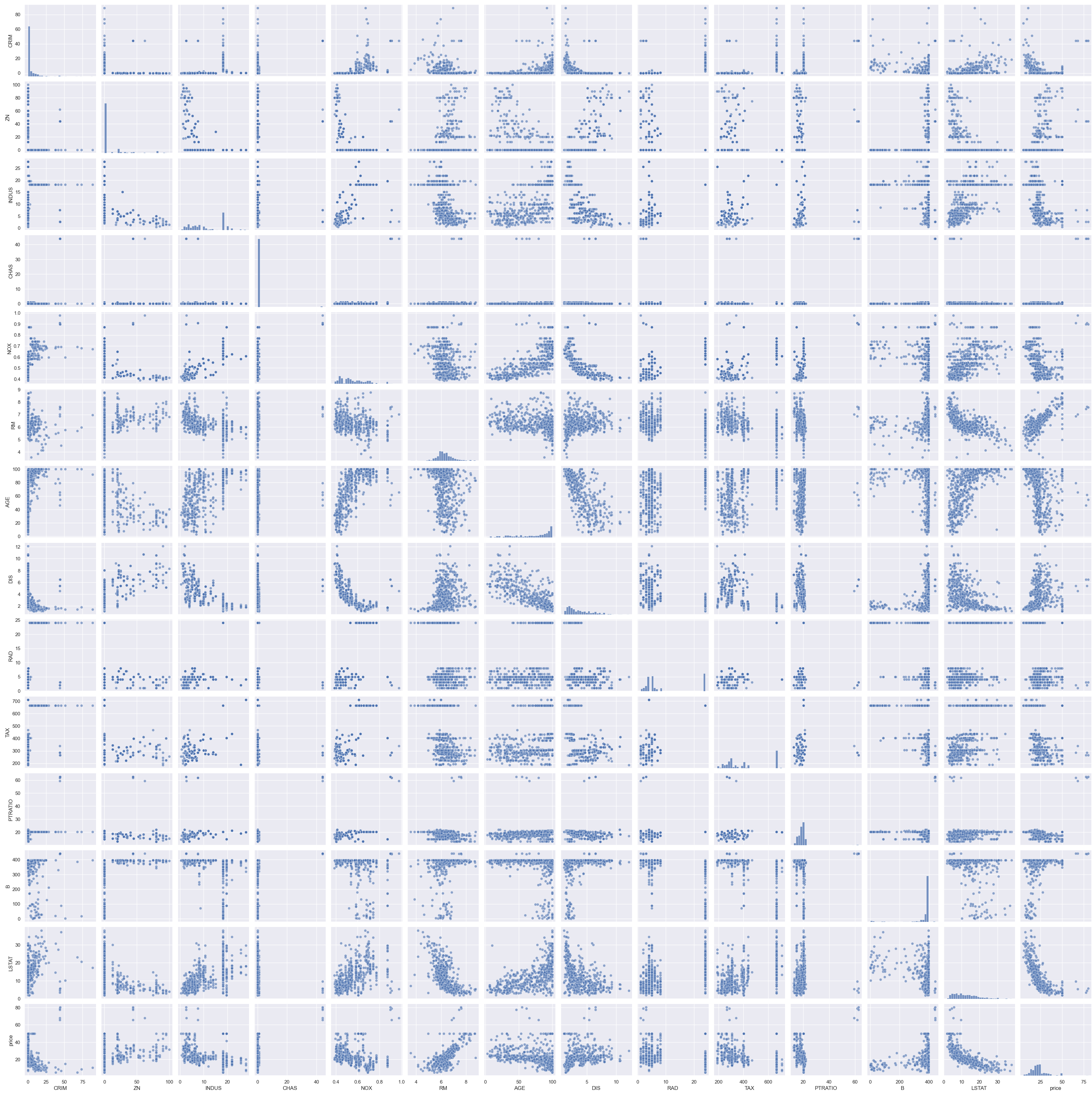
Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x26565d118a0>
```
```
<Figure size 1500x900 with 0 Axes>
```



Here can see the Negative correlation between Price and LSTAT in the dataset.

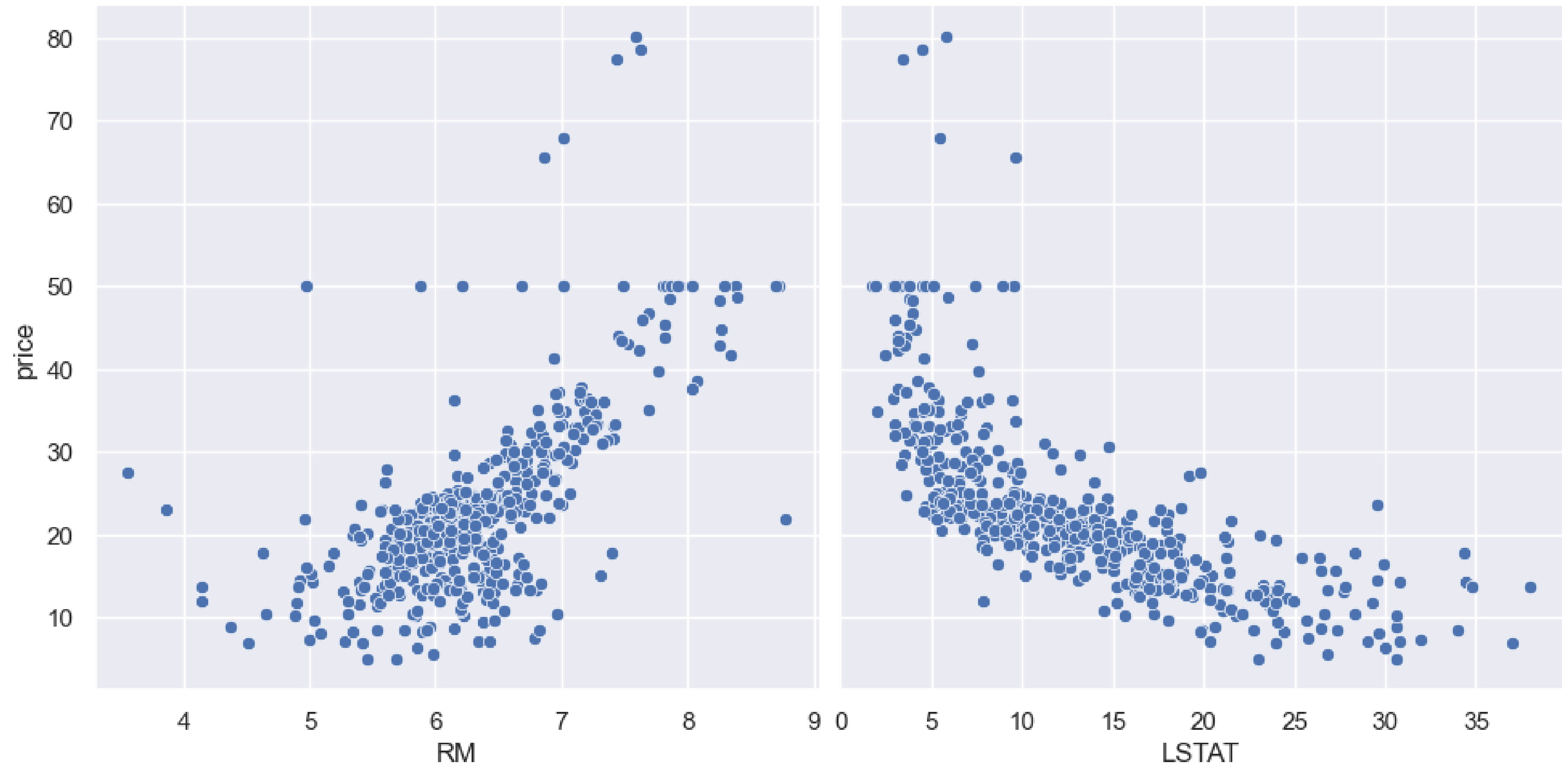
In [ ]:
```
sns.pairplot(df_new,plot_kws={'alpha': 0.6},diag_kws={'bins': 30})
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x26566762c50>



In [ ]:
```
sns.pairplot(df_new,x_vars=["RM","LSTAT"],y_vars=["price"],height=5)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x26566b9efe0>



In [ ]:
```
df_new["price_boolian"] = df_new["price"]>20

df_new["Price_seg"]= (df_new["price_boolian"].map({True:"1", False:"0"})).astype(float)
```

In [ ]:
```
df_new.head(10)
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	price	price_boolian	Price_seg
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	24.0	True	1.0
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	21.6	True	1.0
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	34.7	True	1.0
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	2.94	33.4	True	1.0
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	36.2	True	1.0
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7	True	1.0
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9	True	1.0
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1	True	1.0
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5	False	0.0
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9	False	0.0

Here we added two new variables for Price which is > 20 and <= 20 to make categories and perform some ML Algorithms

In [ ]:
```
#############Logistic Regression using Statsmodels############

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import statsmodels.api as sm
cols=['CRIM', 'ZN', 'INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B'     ,'LSTAT']
X = df_new[cols]
y = df_new['Price_seg']
logitmodel=sm.Logit(y,X)
result=logitmodel.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.265164
         Iterations 10
                           Logit Regression Results
==============================================================================
Dep. Variable:              Price_seg   No. Observations:                  511
Model:                          Logit   Df Residuals:                      498
Method:                           MLE   Df Model:                           12
Date:                Mon, 05 Dec 2022   Pseudo R-squ.:                  0.6104
Time:                        01:01:17   Log-Likelihood:                -135.50
converged:                       True   LL-Null:                       -347.75
Covariance Type:            nonrobust   LLR p-value:                 2.428e-83
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
CRIM          -0.0831      0.074     -1.127      0.260      -0.228       0.061
ZN             0.0204      0.015      1.396      0.163      -0.008       0.049
INDUS          0.1156      0.042      2.722      0.006       0.032       0.199
CHAS           1.6345      0.694      2.354      0.019       0.273       2.996
NOX           -4.8148      2.464     -1.954      0.051      -9.645       0.015
RM             2.0206      0.337      6.001      0.000       1.361       2.681
AGE           -0.0401      0.011     -3.725      0.000      -0.061      -0.019
DIS           -0.5294      0.160     -3.308      0.001      -0.843      -0.216
RAD            0.1651      0.052      3.202      0.001       0.064       0.266
TAX           -0.0084      0.003     -3.046      0.002      -0.014      -0.003
PTRATIO       -0.1817      0.084     -2.161      0.031      -0.347      -0.017
B              0.0084      0.003      2.836      0.005       0.003       0.014
LSTAT         -0.2932      0.048     -6.051      0.000      -0.388      -0.198
==============================================================================
```

In [ ]:
```
X= sm.add_constant(X)
logitmodel2=sm.Logit(y,X)
result2=logitmodel2.fit()
print(result2.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.248371
         Iterations 10
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.635
Dependent Variable: Price_seg        AIC:              281.8352
Date:               2022-12-05 01:01 BIC:              341.1443
No. Observations:   511              Log-Likelihood:   -126.92
Df Model:           13               LL-Null:          -347.75
Df Residuals:       497              LLR p-value:      3.4450e-86
Converged:          1.0000           Scale:            1.0000
No. Iterations:     10.0000
------------------------------------------------------------------
             Coef.    Std.Err.     z      P>|z|    [0.025    0.975]
------------------------------------------------------------------
const        15.6632    3.8381   4.0810  0.0000    8.1407  23.1856
CRIM         -0.0803    0.0674  -1.1900  0.2340   -0.2124   0.0519
ZN            0.0293    0.0163   1.8017  0.0716   -0.0026   0.0611
INDUS         0.1266    0.0460   2.7517  0.0059    0.0364   0.2167
CHAS          1.7204    0.7219   2.3831  0.0172    0.3055   3.1353
NOX         -10.9039    2.9880  -3.6492  0.0003  -16.7603  -5.0474
RM            0.9575    0.3854   2.4848  0.0130    0.2023   1.7128
AGE          -0.0357    0.0111  -3.2066  0.0013   -0.0576  -0.0139
DIS          -0.7627    0.1777  -4.2914  0.0000   -1.1110  -0.4143
RAD           0.2252    0.0565   3.9862  0.0001    0.1145   0.3359
TAX          -0.0096    0.0029  -3.2878  0.0010   -0.0153  -0.0039
PTRATIO      -0.3632    0.0993  -3.6568  0.0003   -0.5579  -0.1685
B             0.0056    0.0027   2.0249  0.0429    0.0002   0.0110
LSTAT        -0.3646    0.0546  -6.6788  0.0000   -0.4716  -0.2576
=================================================================
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100)
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)
y_pred=logmodel.predict(X_test)
```

```
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:444: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [ ]:
```
probs = logmodel.predict_proba(X_test)

preds = probs[:,1]
```

In [ ]:
```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.81      0.81        62
         1.0       0.87      0.88      0.88        92

    accuracy                           0.85       154
   macro avg       0.85      0.84      0.84       154
weighted avg       0.85      0.85      0.85       154

[[50 12]
 [11 81]]
0.8506493506493507
```

In [ ]:
```
####Decision Tree#######
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100)
from sklearn import tree
DTclf=tree.DecisionTreeClassifier()
DTclf.fit(X_train, y_train)
#####Predict probabilities for the test data.
probsDT = DTclf.predict_proba(X_test)
####Keep Probabilities of the positive class only.
probsDT = probsDT[:, 1]
```

In [ ]:
```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.81      0.81        62
         1.0       0.87      0.88      0.88        92

    accuracy                           0.85       154
   macro avg       0.85      0.84      0.84       154
weighted avg       0.85      0.85      0.85       154

[[50 12]
 [11 81]]
0.8506493506493507
```

In [ ]:
```
################ RF ############################################
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
RFclf=RandomForestClassifier(n_estimators=900)
RFclf.fit(X_train, y_train)
y_pred=RFclf.predict(X_test)

#####Predict probabilities for the test data.
probsRF = RFclf.predict_proba(X_test)

####Keep Probabilities of the positive class only.
probsRF = probsRF[:, 1]
```

In [ ]:
```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.87      0.83      0.85        54
         1.0       0.91      0.93      0.92       100

    accuracy                           0.90       154
   macro avg       0.89      0.88      0.88       154
weighted avg       0.90      0.90      0.90       154

[[45  9]
 [ 7 93]]
0.8961038961038961
```

In [ ]:
```
###################### SVM #####################################
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
SVclf2 = SVC(kernel='rbf', C=10, gamma='auto')

#(kernel='poly', degree=4), kernel='linear', Gaussian kernel: kernel = 'rbf', kernel='sigmoid'
SVclf2.fit(X_train, y_train)
y_pred=SVclf2.predict(X_test)
probsSV = SVclf2.fit(X_train, y_train).decision_function(X_test)
```

In [ ]:
```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.28      0.42        54
         1.0       0.72      0.98      0.83       100

    accuracy                           0.73       154
   macro avg       0.80      0.63      0.62       154
weighted avg       0.77      0.73      0.69       154

[[15 39]
 [ 2 98]]
0.7337662337662337
```

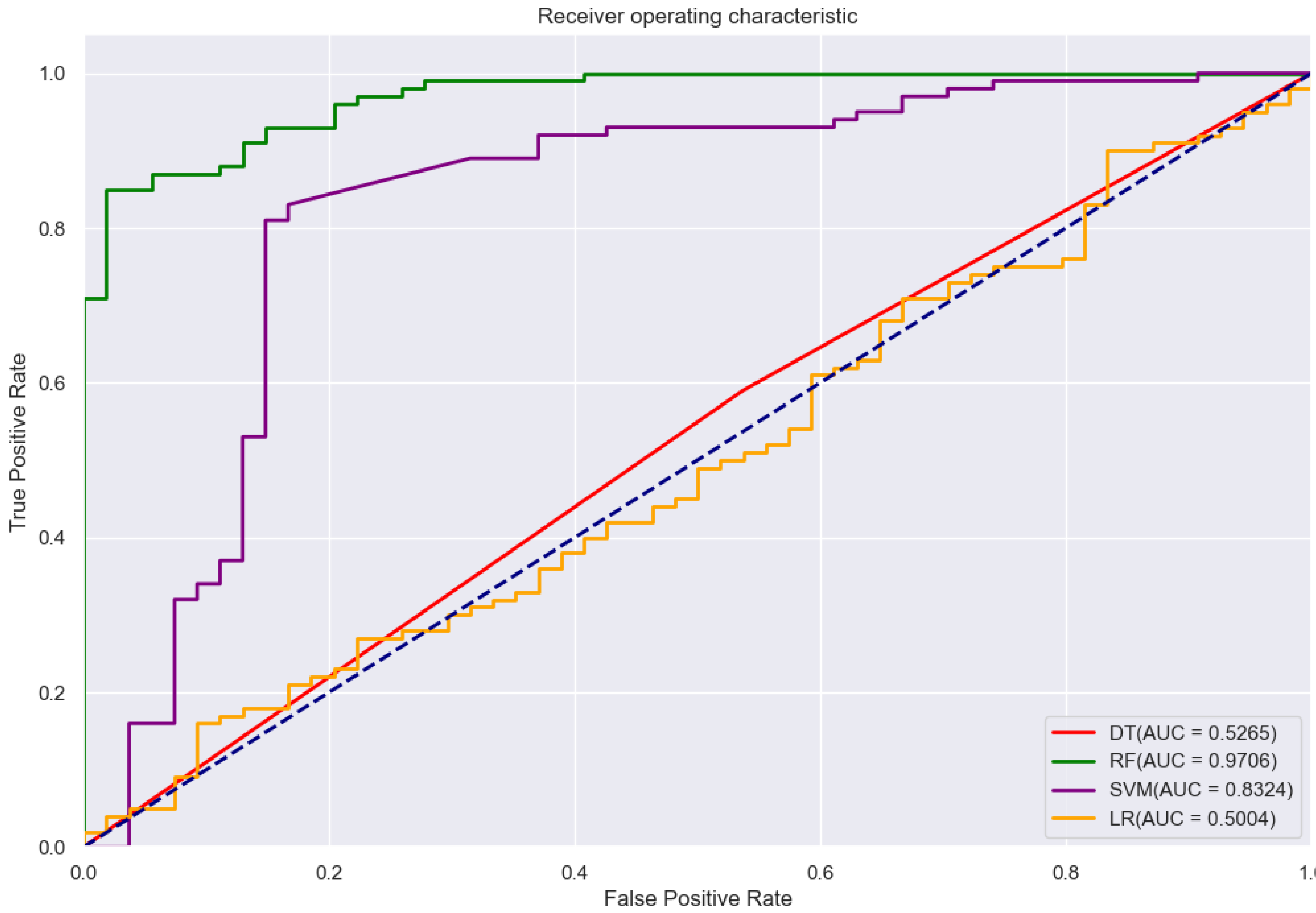
In [ ]:

```python
###Compute the AUC Score.
from sklearn.metrics import roc_curve, roc_auc_score

auc = roc_auc_score(y_test, probsDT)

auc2 = roc_auc_score(y_test, probsRF)

auc3 = roc_auc_score(y_test, probsSV)

auc4 = roc_auc_score(y_test, preds)

print('DT AUC:', auc)

print('RF AUC2:', auc2)

print('SVM AUC3:', auc3)

print('LR AUC4:', auc4)
```

```
DT AUC: 0.5264814814814814
RF AUC2: 0.9705555555555555
SVM AUC3: 0.8324074074074074
LR AUC4: 0.5003703703703704
```

In [ ]:

```python
###Get the ROC Curve

fpr, tpr, thresholds = roc_curve(y_test, probsDT)

fpr2, tpr2, thresholds2 = roc_curve(y_test, probsRF)

fpr3, tpr3, thresholds3 = roc_curve(y_test, probsSV)

fpr4, tpr4, thresholds5 = roc_curve(y_test, preds)

####Plot ROC Curve

plt.figure()

lw = 2

plt.plot(fpr, tpr, color='red',lw=lw, label='DT(AUC = %0.4f)' % auc)

plt.plot(fpr2, tpr2, color='green',lw=lw, label='RF(AUC = %0.4f)' % auc2)

plt.plot(fpr3, tpr3, color='purple',lw=lw, label='SVM(AUC = %0.4f)' % auc3)

plt.plot(fpr4, tpr4, color='orange',lw=lw, label='LR(AUC = %0.4f)' % auc4)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

plt.show()
```



Here we can see Random Forest is performing better among all the other tests.