Delivered To:

::: Dr Ayesha Hakim :::

In [1]:
```
import numpy as np
import pandas as pd
```

Import the dataset

In [2]:
```
df= pd.read_csv('Boston Dataset.csv')
```

In [3]:
```
df.head(50)
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	M
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.90	19.15	
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311	15.2	386.63	29.93	
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.10	
10	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	
11	0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.90	13.27	
12	0.09378	12.5	7.87	0	0.524	5.889	39.0	5.4509	5	311	15.2	390.50	15.71	
13	0.62976	0.0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21.0	396.90	8.26	
14	0.63796	0.0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21.0	380.02	10.26	
15	0.62739	0.0	8.14	0	0.538	5.834	56.5	4.4986	4	307	21.0	395.62	8.47	
16	1.05393	0.0	8.14	0	0.538	5.935	29.3	4.4986	4	307	21.0	386.85	6.58	
17	0.78420	0.0	8.14	0	0.538	5.990	81.7	4.2579	4	307	21.0	386.75	14.67	
18	0.80271	0.0	8.14	0	0.538	5.456	36.6	3.7965	4	307	21.0	288.99	11.69	
19	0.72580	0.0	8.14	0	0.538	5.727	69.5	3.7965	4	307	21.0	390.95	11.28	
20	1.25179	0.0	8.14	0	0.538	5.570	98.1	3.7979	4	307	21.0	376.57	21.02	
21	0.85204	0.0	8.14	0	0.538	5.965	89.2	4.0123	4	307	21.0	392.53	13.83	
22	1.23247	0.0	8.14	0	0.538	6.142	91.7	3.9769	4	307	21.0	396.90	18.72	
23	0.98843	0.0	8.14	0	0.538	5.813	100.0	4.0952	4	307	21.0	394.54	19.88	
24	0.75026	0.0	8.14	0	0.538	5.924	94.1	4.3996	4	307	21.0	394.33	16.30	
25	0.84054	0.0	8.14	0	0.538	5.599	85.7	4.4546	4	307	21.0	303.42	16.51	
26	0.67191	0.0	8.14	0	0.538	5.813	90.3	4.6820	4	307	21.0	376.88	14.81	
27	0.95577	0.0	8.14	0	0.538	6.047	88.8	4.4534	4	307	21.0	306.38	17.28	
28	0.77299	0.0	8.14	0	0.538	6.495	94.4	4.4547	4	307	21.0	387.94	12.80	
29	1.00245	0.0	8.14	0	0.538	6.674	87.3	4.2390	4	307	21.0	380.23	11.98	
30	1.13081	0.0	8.14	0	0.538	5.713	94.1	4.2330	4	307	21.0	360.17	22.60	
31	1.35472	0.0	8.14	0	0.538	6.072	100.0	4.1750	4	307	21.0	376.73	13.04	
32	1.38799	0.0	8.14	0	0.538	5.950	82.0	3.9900	4	307	21.0	232.60	27.71	
33	1.15172	0.0	8.14	0	0.538	5.701	95.0	3.7872	4	307	21.0	358.77	18.35	

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	M
34	1.61282	0.0	8.14	0	0.538	6.096	96.9	3.7598	4	307	21.0	248.31	20.34	
35	0.06417	0.0	5.96	0	0.499	5.933	68.2	3.3603	5	279	19.2	396.90	9.68	
36	0.09744	0.0	5.96	0	0.499	5.841	61.4	3.3779	5	279	19.2	377.56	11.41	
37	0.08014	0.0	5.96	0	0.499	5.850	41.5	3.9342	5	279	19.2	396.90	8.77	
38	0.17505	0.0	5.96	0	0.499	5.966	30.2	3.8473	5	279	19.2	393.43	10.13	
39	0.02763	75.0	2.95	0	0.428	6.595	21.8	5.4011	3	252	18.3	395.63	4.32	
40	0.03359	75.0	2.95	0	0.428	7.024	15.8	5.4011	3	252	18.3	395.62	1.98	
41	0.12744	0.0	6.91	0	0.448	6.770	2.9	5.7209	3	233	17.9	385.41	4.84	
42	0.14150	0.0	6.91	0	0.448	6.169	6.6	5.7209	3	233	17.9	383.37	5.81	
43	0.15936	0.0	6.91	0	0.448	6.211	6.5	5.7209	3	233	17.9	394.46	7.44	
44	0.12269	0.0	6.91	0	0.448	6.069	40.0	5.7209	3	233	17.9	389.39	9.55	
45	0.17142	0.0	6.91	0	0.448	5.682	33.8	5.1004	3	233	17.9	396.90	10.21	
46	0.18836	0.0	6.91	0	0.448	5.786	33.3	5.1004	3	233	17.9	396.90	14.15	
47	0.22927	0.0	6.91	0	0.448	6.030	85.5	5.6894	3	233	17.9	392.74	18.80	
48	0.25387	0.0	6.91	0	0.448	5.399	95.3	5.8700	3	233	17.9	396.90	30.81	
49	0.21977	0.0	6.91	0	0.448	5.602	62.0	6.0877	3	233	17.9	396.90	16.20	

In [4]:
```
df.tail(50)
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	M
456	4.66883	0.0	18.10	0	0.713	5.976	87.9	2.5806	24	666	20.2	10.48	19.01	
457	8.20058	0.0	18.10	0	0.713	5.936	80.3	2.7792	24	666	20.2	3.50	16.94	
458	7.75223	0.0	18.10	0	0.713	6.301	83.7	2.7831	24	666	20.2	272.21	16.23	
459	6.80117	0.0	18.10	0	0.713	6.081	84.4	2.7175	24	666	20.2	396.90	14.70	
460	4.81213	0.0	18.10	0	0.713	6.701	90.0	2.5975	24	666	20.2	255.23	16.42	
461	3.69311	0.0	18.10	0	0.713	6.376	88.4	2.5671	24	666	20.2	391.43	14.65	
462	6.65492	0.0	18.10	0	0.713	6.317	83.0	2.7344	24	666	20.2	396.90	13.99	
463	5.82115	0.0	18.10	0	0.713	6.513	89.9	2.8016	24	666	20.2	393.82	10.29	
464	7.83932	0.0	18.10	0	0.655	6.209	65.4	2.9634	24	666	20.2	396.90	13.22	
465	3.16360	0.0	18.10	0	0.655	5.759	48.2	3.0665	24	666	20.2	334.40	14.13	
466	3.77498	0.0	18.10	0	0.655	5.952	84.7	2.8715	24	666	20.2	22.01	17.15	
467	4.42228	0.0	18.10	0	0.584	6.003	94.5	2.5403	24	666	20.2	331.29	21.32	
468	15.57570	0.0	18.10	0	0.580	5.926	71.0	2.9084	24	666	20.2	368.74	18.13	
469	13.07510	0.0	18.10	0	0.580	5.713	56.7	2.8237	24	666	20.2	396.90	14.76	
470	4.34879	0.0	18.10	0	0.580	6.167	84.0	3.0334	24	666	20.2	396.90	16.29	
471	4.03841	0.0	18.10	0	0.532	6.229	90.7	3.0993	24	666	20.2	395.33	12.87	
472	3.56868	0.0	18.10	0	0.580	6.437	75.0	2.8965	24	666	20.2	393.37	14.36	
473	4.64689	0.0	18.10	0	0.614	6.980	67.6	2.5329	24	666	20.2	374.68	11.66	
474	8.05579	0.0	18.10	0	0.584	5.427	95.4	2.4298	24	666	20.2	352.58	18.14	
475	6.39312	0.0	18.10	0	0.584	6.162	97.4	2.2060	24	666	20.2	302.76	24.10	
476	4.87141	0.0	18.10	0	0.614	6.484	93.6	2.3053	24	666	20.2	396.21	18.68	
477	15.02340	0.0	18.10	0	0.614	5.304	97.3	2.1007	24	666	20.2	349.48	24.91	
478	10.23300	0.0	18.10	0	0.614	6.185	96.7	2.1705	24	666	20.2	379.70	18.03	
479	14.33370	0.0	18.10	0	0.614	6.229	88.0	1.9512	24	666	20.2	383.32	13.11	
480	5.82401	0.0	18.10	0	0.532	6.242	64.7	3.4242	24	666	20.2	396.90	10.74	
481	5.70818	0.0	18.10	0	0.532	6.750	74.9	3.3317	24	666	20.2	393.07	7.74	
482	5.73116	0.0	18.10	0	0.532	7.061	77.0	3.4106	24	666	20.2	395.28	7.01	
483	2.81838	0.0	18.10	0	0.532	5.762	40.3	4.0983	24	666	20.2	392.92	10.42	
484	2.37857	0.0	18.10	0	0.583	5.871	41.9	3.7240	24	666	20.2	370.73	13.34	
485	3.67367	0.0	18.10	0	0.583	6.312	51.9	3.9917	24	666	20.2	388.62	10.58	
486	5.69175	0.0	18.10	0	0.583	6.114	79.8	3.5459	24	666	20.2	392.68	14.98	
487	4.83567	0.0	18.10	0	0.583	5.905	53.2	3.1523	24	666	20.2	388.22	11.45	
488	0.15086	0.0	27.74	0	0.609	5.454	92.7	1.8209	4	711	20.1	395.09	18.06	
489	0.18337	0.0	27.74	0	0.609	5.414	98.3	1.7554	4	711	20.1	344.05	23.97	

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	M
490	0.20746	0.0	27.74	0	0.609	5.093	98.0	1.8226	4	711	20.1	318.43	29.68	
491	0.10574	0.0	27.74	0	0.609	5.983	98.8	1.8681	4	711	20.1	390.11	18.07	
492	0.11132	0.0	27.74	0	0.609	5.983	83.5	2.1099	4	711	20.1	396.90	13.35	
493	0.17331	0.0	9.69	0	0.585	5.707	54.0	2.3817	6	391	19.2	396.90	12.01	
494	0.27957	0.0	9.69	0	0.585	5.926	42.6	2.3817	6	391	19.2	396.90	13.59	
495	0.17899	0.0	9.69	0	0.585	5.670	28.8	2.7986	6	391	19.2	393.29	17.60	
496	0.28960	0.0	9.69	0	0.585	5.390	72.9	2.7986	6	391	19.2	396.90	21.14	
497	0.26838	0.0	9.69	0	0.585	5.794	70.6	2.8927	6	391	19.2	396.90	14.10	
498	0.23912	0.0	9.69	0	0.585	6.019	65.3	2.4091	6	391	19.2	396.90	12.92	
499	0.17783	0.0	9.69	0	0.585	5.569	73.5	2.3999	6	391	19.2	395.77	15.10	
500	0.22438	0.0	9.69	0	0.585	6.027	79.7	2.4982	6	391	19.2	396.90	14.33	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	

Shape of dataset

In [5]:
```
print('Shape of Training dataset:', df.shape)
```

```
Shape of Training dataset: (506, 14)
```

Checking null values for training dataset

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
```

The Target Variable is the last one which is called MEDV.

Here lets change ‘medv’ column name to ‘Price’

In [7]:
```
df.rename(columns={'MEDV':'PRICE'}, inplace=True)
df
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PI
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	

506 rows × 14 columns

Exploratory Data Analysis

Information about the dataset features

In [8]:
```
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     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    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

Describe

In [9]:
```
df.describe()
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00000(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79504:
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10571(
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12960(
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10017!
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20745(
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18842!
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12650(

Minimum price of the data

In [10]:
```
minimum_price = np.amin(df['PRICE'])
```

Maximum price of the data

In [11]:
```
maximum_price = np.amax(df['PRICE'])
```

Mean price of the data

In [12]:
```
mean_price = np.mean(df['PRICE'])
```

Median price of the data

In [13]:
```
median_price = np.median(df['PRICE'])
```

Standard deviation of prices of the data

In [14]:
```
std_price = np.std(df['PRICE'])
```

Show the calculated statistics

In [15]:
```
print("Statistics for Boston housing dataset:\n")
print("Minimum PRICES: ${}".format(minimum_price))
print("Maximum PRICES: ${}".format(maximum_price))
print("Mean PRICES: ${}".format(mean_price))
print("Median PRICES ${}".format(median_price))
print("Standard deviation of PRICES: ${}".format(std_price))
```

```
Statistics for Boston housing dataset:

Minimum PRICES: $5.0
Maximum PRICES: $50.0
Mean PRICES: $22.532806324110677
Median PRICES $21.2
Standard deviation of PRICES: $9.188011545278206
```

Feature Observation

Finding out the correlation between the features

In [16]:
```
corr = df.corr()
corr.shape
```

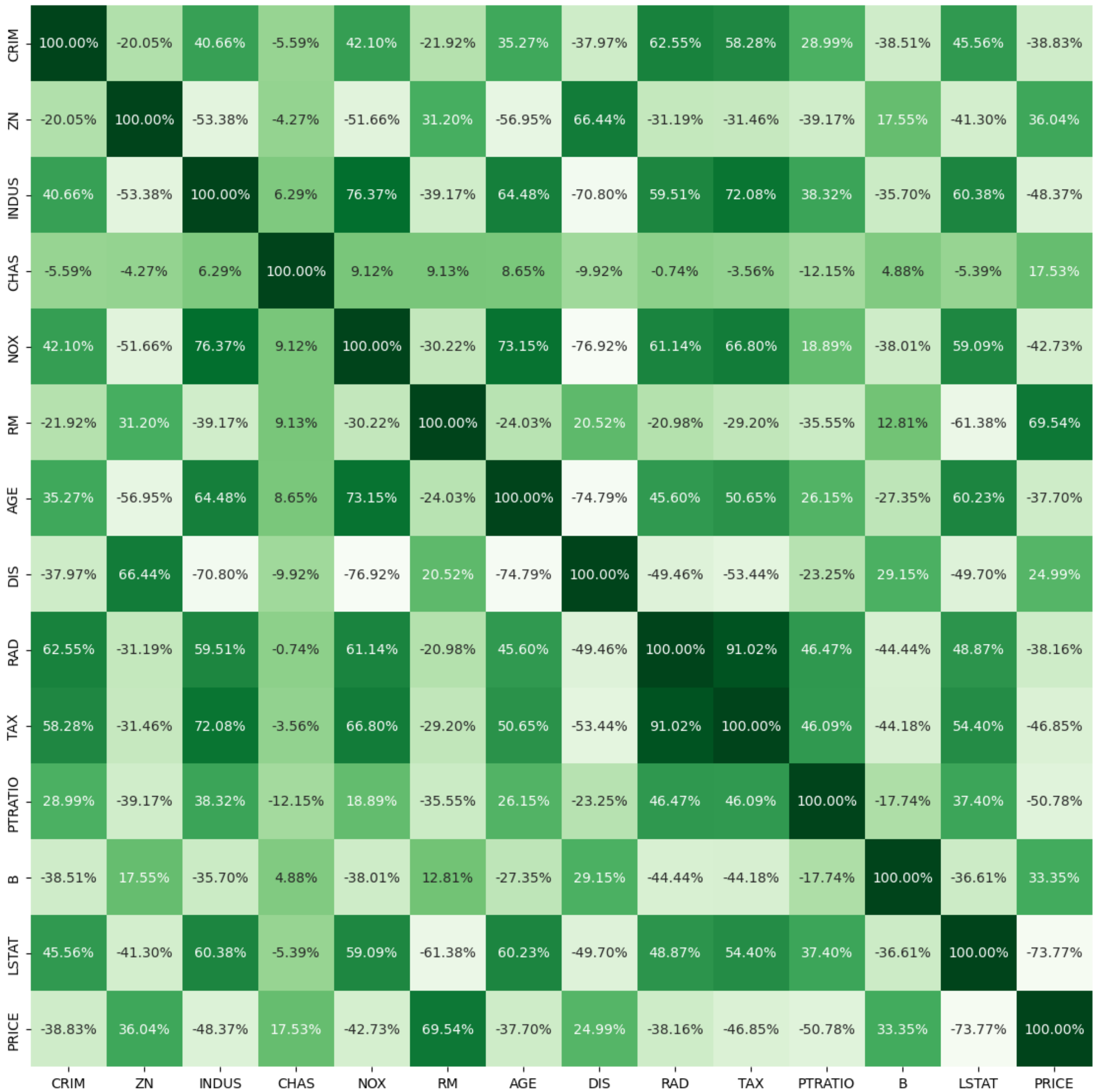
Out[16]:
```
(14, 14)
```

Plotting the heatmap of correlation between features

In [17]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [18]:
```
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar=False, square= True, fmt='.2%', annot=True, cmap='Greens')
```

Out[18]: <Axes: >



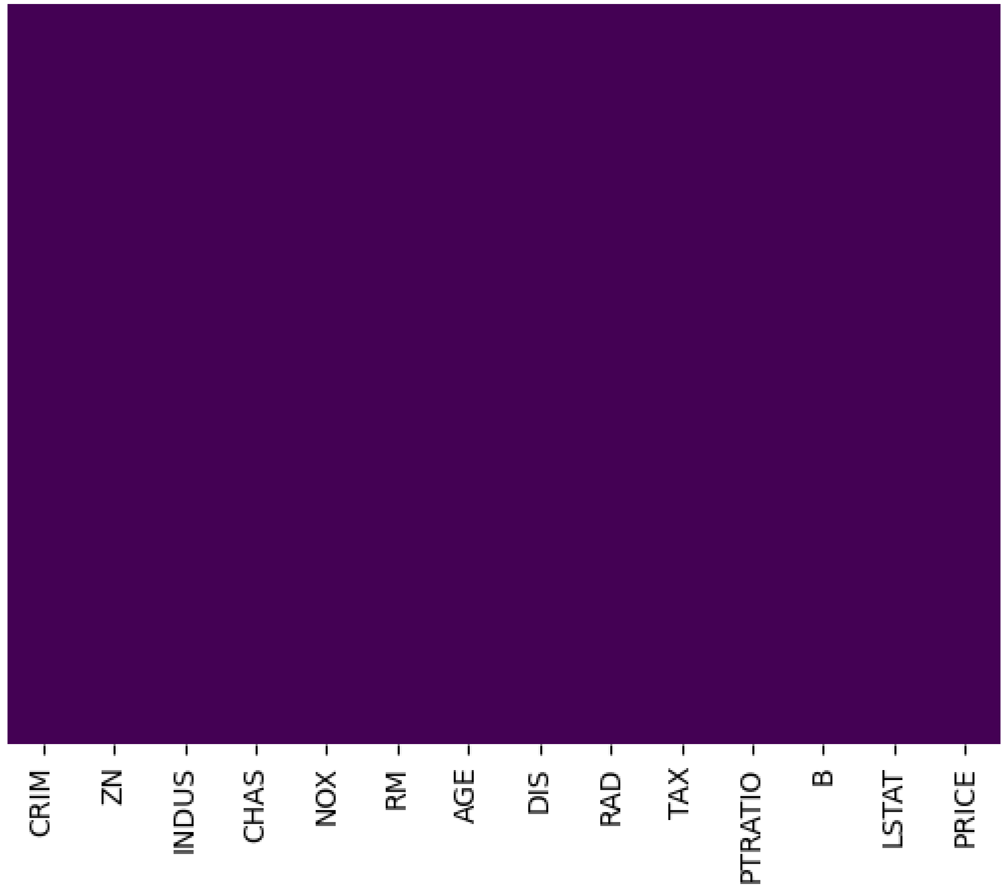
::: HeatMap :::

Checking the null values using heatmap

There is any null values are occupyed here

In [19]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

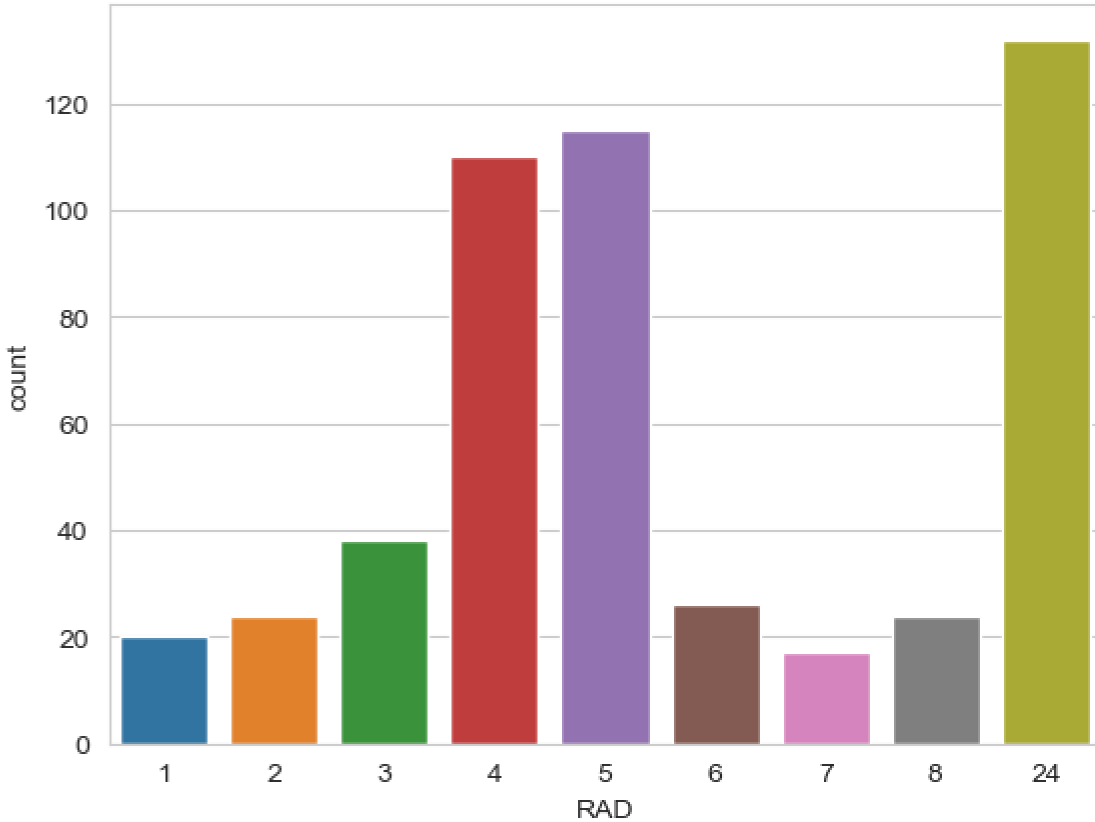
Out[19]: <Axes: >



Note: There are no null or missing values here.

In [20]:
```
sns.set_style('whitegrid')
sns.countplot(x='RAD',data=df)
```

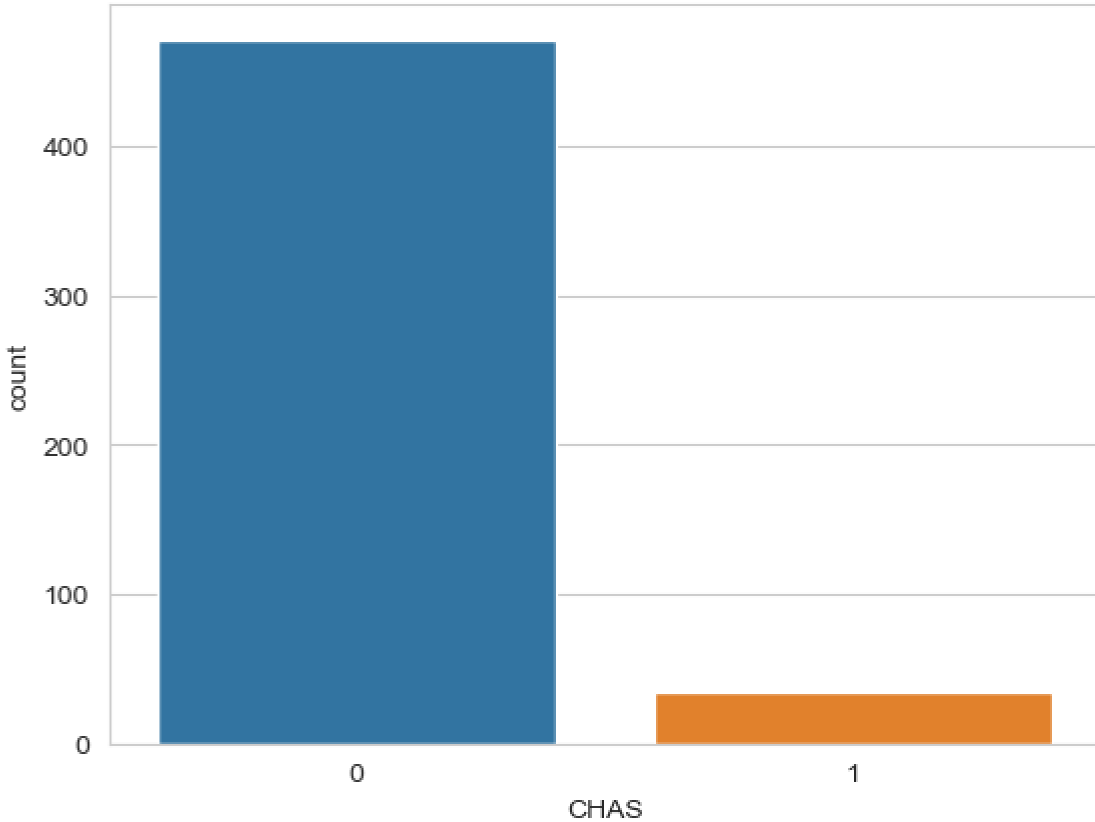
Out[20]:
```
<Axes: xlabel='RAD', ylabel='count'>
```



::: Counting For RAD Values :::

In [21]:
```
sns.set_style('whitegrid')
sns.countplot(x='CHAS',data=df)
```

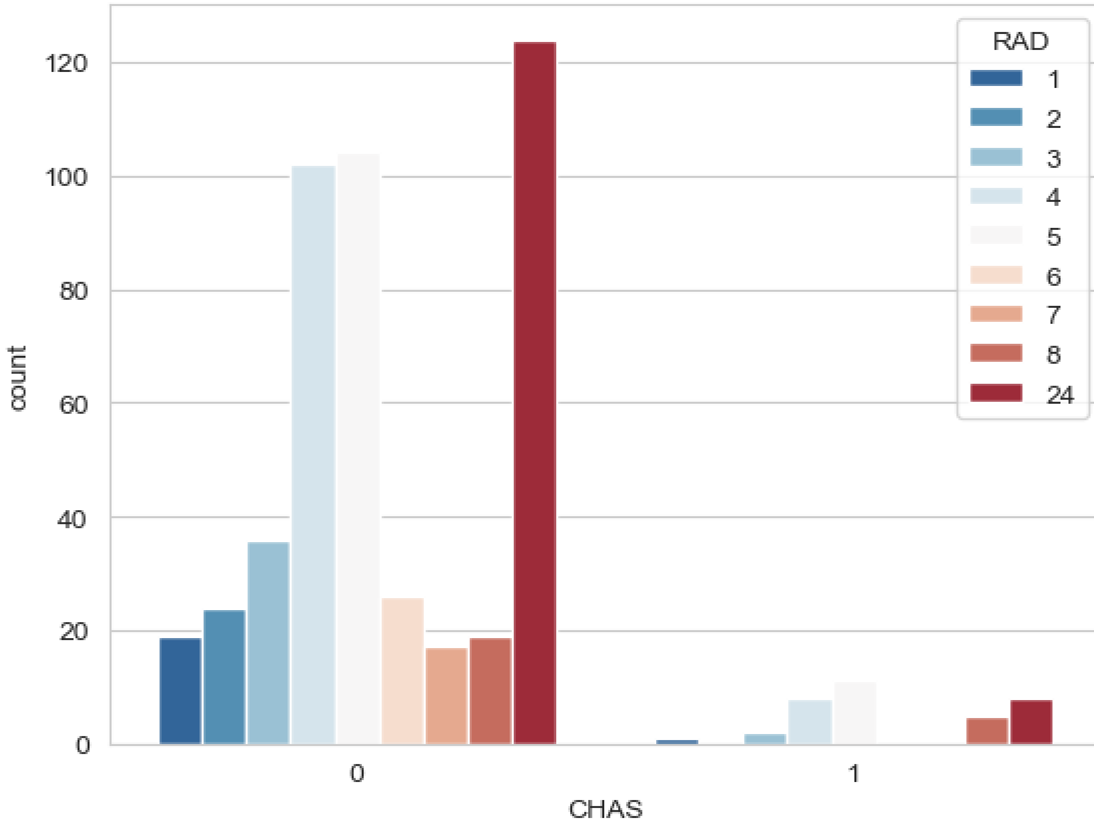
Out[21]: <Axes: xlabel='CHAS', ylabel='count'>



::: Counting For CHAS Feature :::

In [22]:
```
sns.set_style('whitegrid')
sns.countplot(x='CHAS',hue='RAD',data=df,palette='RdBu_r')
```

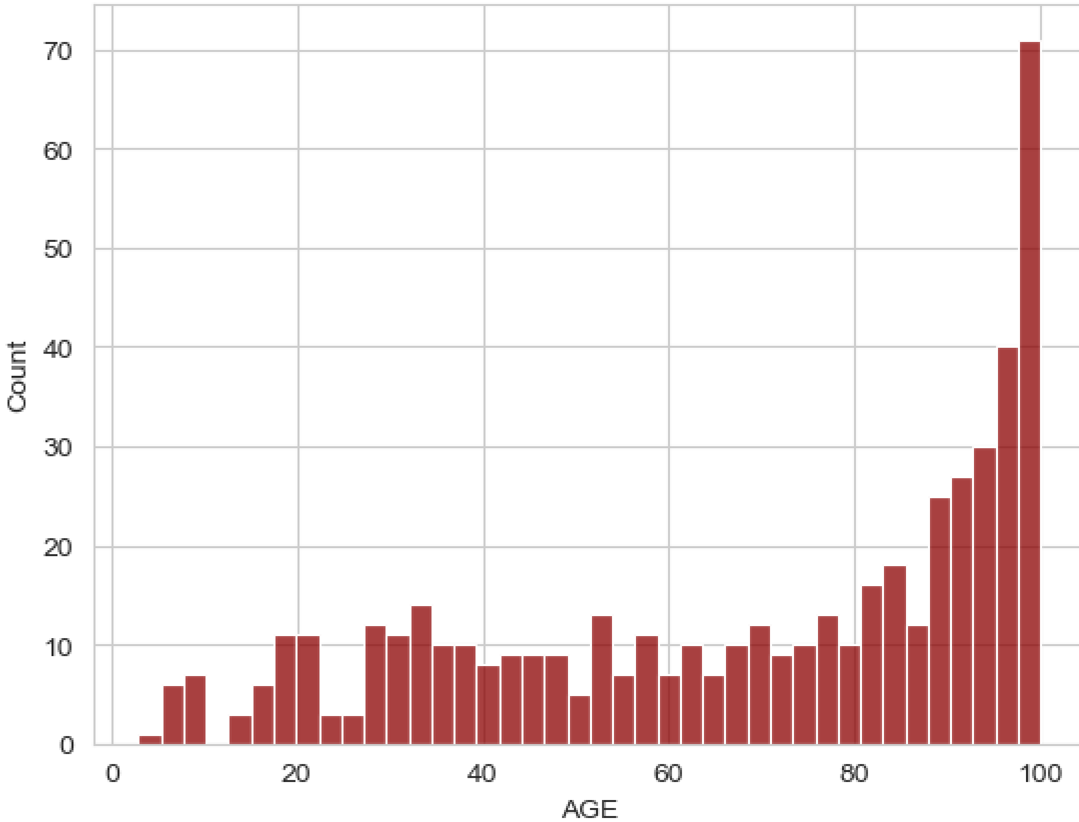
Out[22]: <Axes: xlabel='CHAS', ylabel='count'>



::: CHAS DATA :::

In [23]:
```
sns.histplot(data=df, x='AGE', color='darkred', bins=40)
```

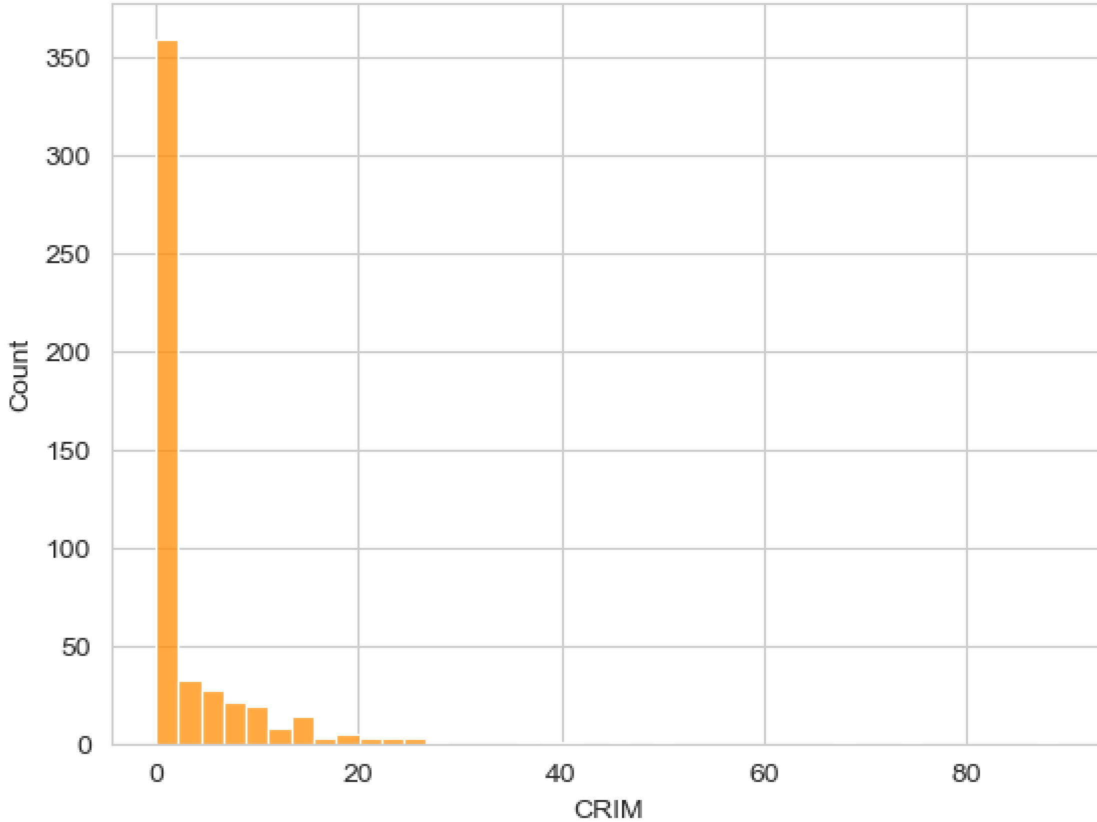
Out[23]: <Axes: xlabel='AGE', ylabel='Count'>



::: HOUSE'S AGE Features Understanding :::

In [24]:
```
sns.histplot(df['CRIM'].dropna(), kde=False, color='darkorange', bins=40)
```

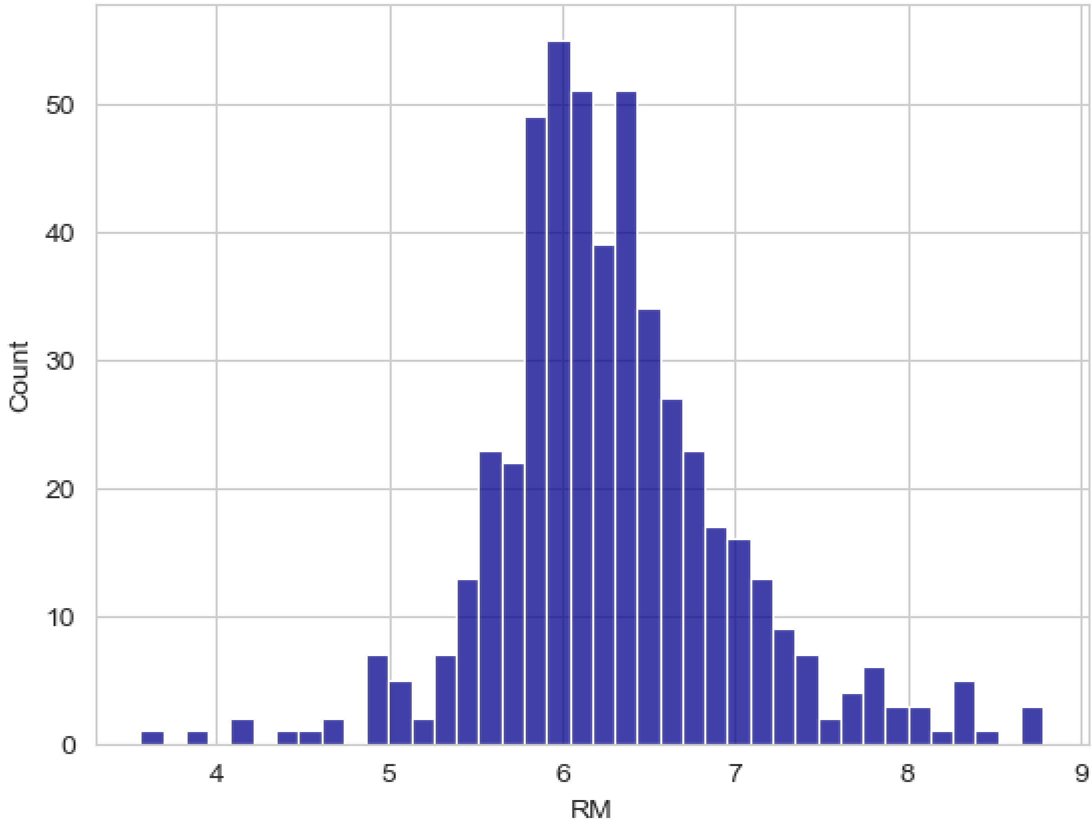
Out[24]:
```
<Axes: xlabel='CRIM', ylabel='Count'>
```



::: CRIM RATE :::

In [25]:
```
sns.histplot(df['RM'].dropna(), color='darkblue', bins=40)
```

Out[25]: <Axes: xlabel='RM', ylabel='Count'>




```
                    ::: Understanding Number of ROOMS into the
HOUSES :::
```

Feature Selection

Lets try to understand which are important feature for this dataset

In [26]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

Independent Columns

In [27]:
```
X = df.iloc[:,0:13]
```

Target Column i.e PRICE range

In [28]:
```
y = df.iloc[:,-1]
```

In [29]:
```
y = np.round(df['PRICE'])
```

Apply SelectKBest class to extract top 5 best features

In [30]:
```
bestfeatures = SelectKBest(score_func=chi2, k=5)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
```

Concat two dataframes for better visualization

In [31]:
```
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
```

Naming the dataframe Columns

In [32]:
```
featureScores.columns = ['Specs','Score']
featureScores
```

Out[32]:

	Specs	Score
0	CRIM	5503.817133
1	ZN	5937.859414
2	INDUS	873.746270
3	CHAS	59.080170
4	NOX	5.073299
5	RM	21.981504
6	AGE	2424.308937
7	DIS	163.919426
8	RAD	1445.257647
9	TAX	14817.836927
10	PTRATIO	45.692587
11	B	3340.486412
12	LSTAT	1430.549632

Print 5 best features

In [33]:
```
print(featureScores.nlargest(5,'Score'))
```

```
   Specs         Score
9    TAX  14817.836927
1     ZN   5937.859414
0   CRIM   5503.817133
11     B   3340.486412
6    AGE   2424.308937
```

Feature Importance

In [34]:
```
from sklearn.ensemble import ExtraTreesClassifier
```

In [35]:
```
model = ExtraTreesClassifier()
model.fit(X,y)
```

Out[35]:
```
▾ ExtraTreesClassifier
ExtraTreesClassifier()
```

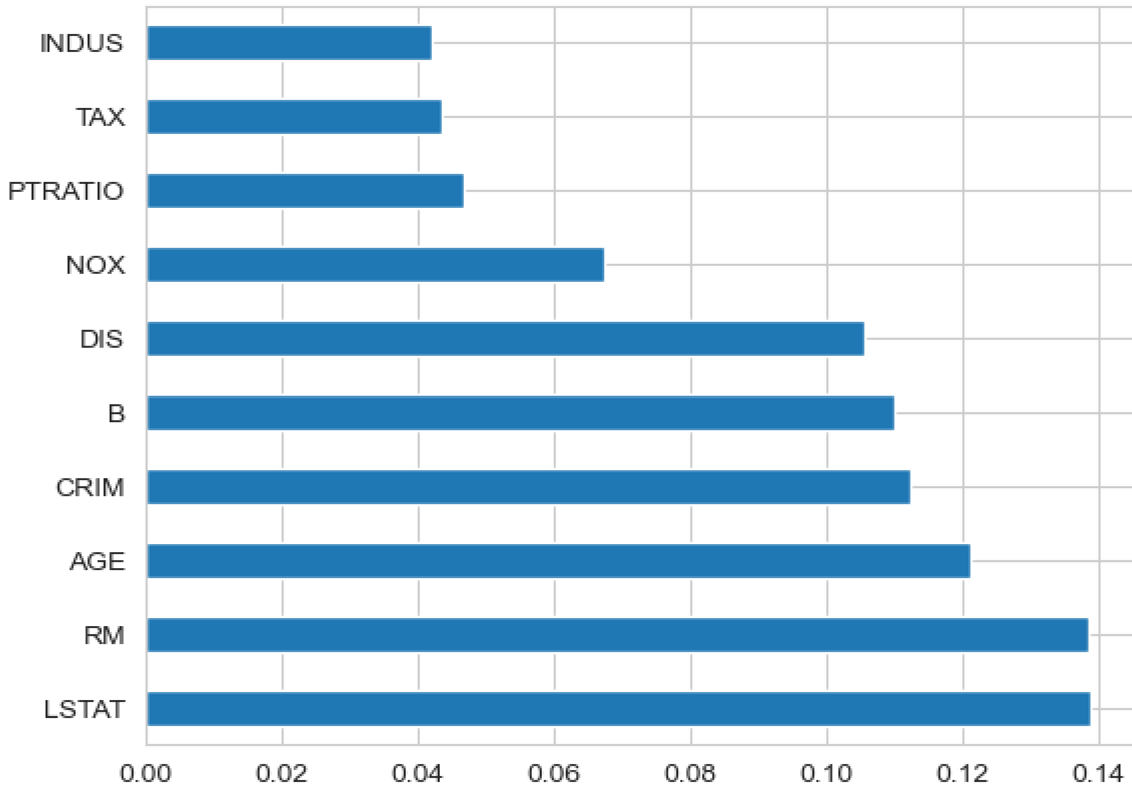
Use inbuilt class feature_importances of tree based classifiers

In [36]:
```
print(model.feature_importances_)
```

```
[0.11227324 0.02286096 0.04199342 0.01378241 0.06742125 0.13874429
 0.12121669 0.10574195 0.036542   0.04362699 0.04670181 0.11009668
 0.1389983 ]
```

Plot graph of feature importances for better visualization

In [37]:
```
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```




```
::: Important Features rated by target variable correlation :::
```

Model Fitting

Linear Regression

Value Assigning

In [38]:
```
x=df.iloc[:,0:13]
y=df.iloc[:,-1]
```

In [39]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=0)
```

In [40]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

Out[40]:
```
▾ LinearRegression
LinearRegression()
```

In [41]:
```
y_pred=model.predict(x_train)
```

In [42]:
```
print("Training Accuracy:",model.score(x_train,y_train)*100)
```

```
Training Accuracy: 77.30135569264233
```

In [43]:
```
print("Testing Accuracy:",model.score(x_test,y_test)*100)
```

```
Testing Accuracy: 58.9222384918251
```

In [44]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [45]:
```
print("Model Accuracy:",r2_score(y,model.predict(x))*100)
```

```
Model Accuracy: 73.73440319905033
```

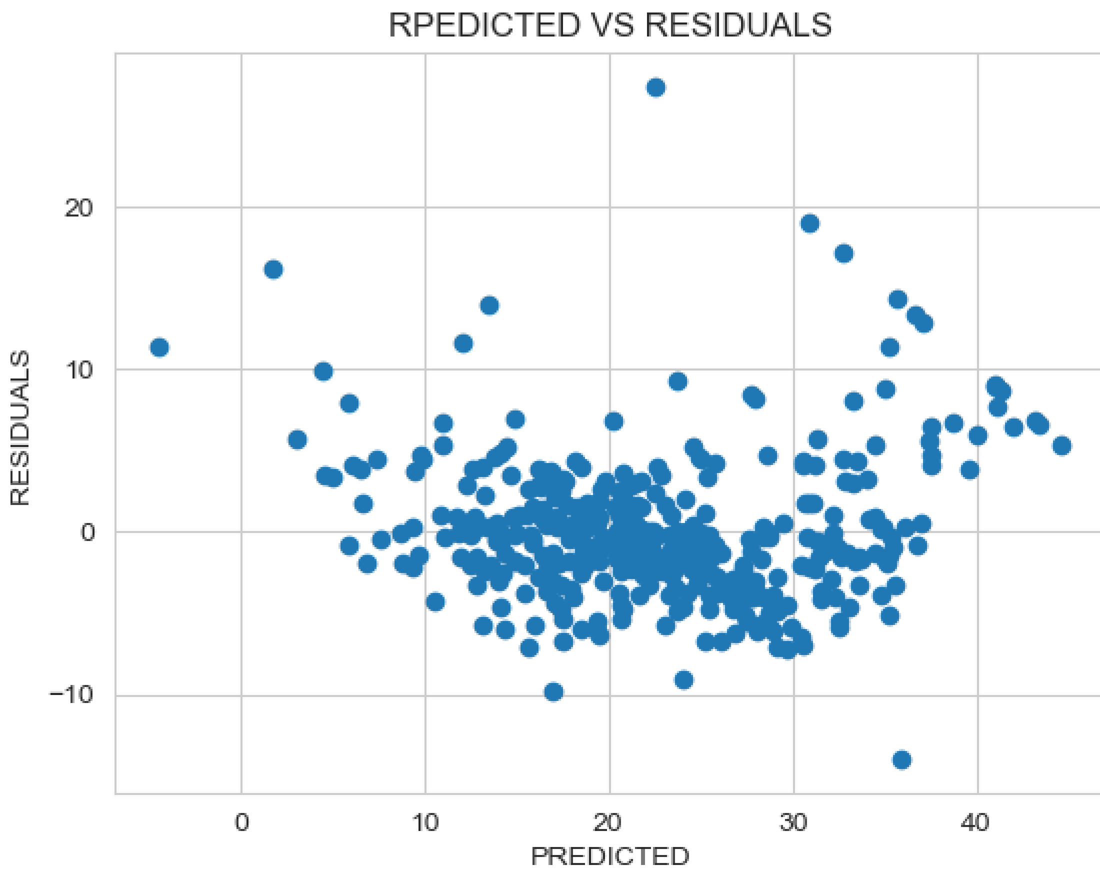
In [46]:
```
plt.scatter(y_train,y_pred)
plt.xlabel('PRICES')
plt.ylabel('PREDICTED PRICES')
plt.title('PRICES VS RPEDICTED PRICES')
plt.show()
```

::: See! how data points are predicted :::

Checking Residuals

In [47]:
```
plt.scatter(y_pred,y_train-y_pred)
plt.title('RPEDICTED VS RESIDUALS')
plt.xlabel('PREDICTED')
```

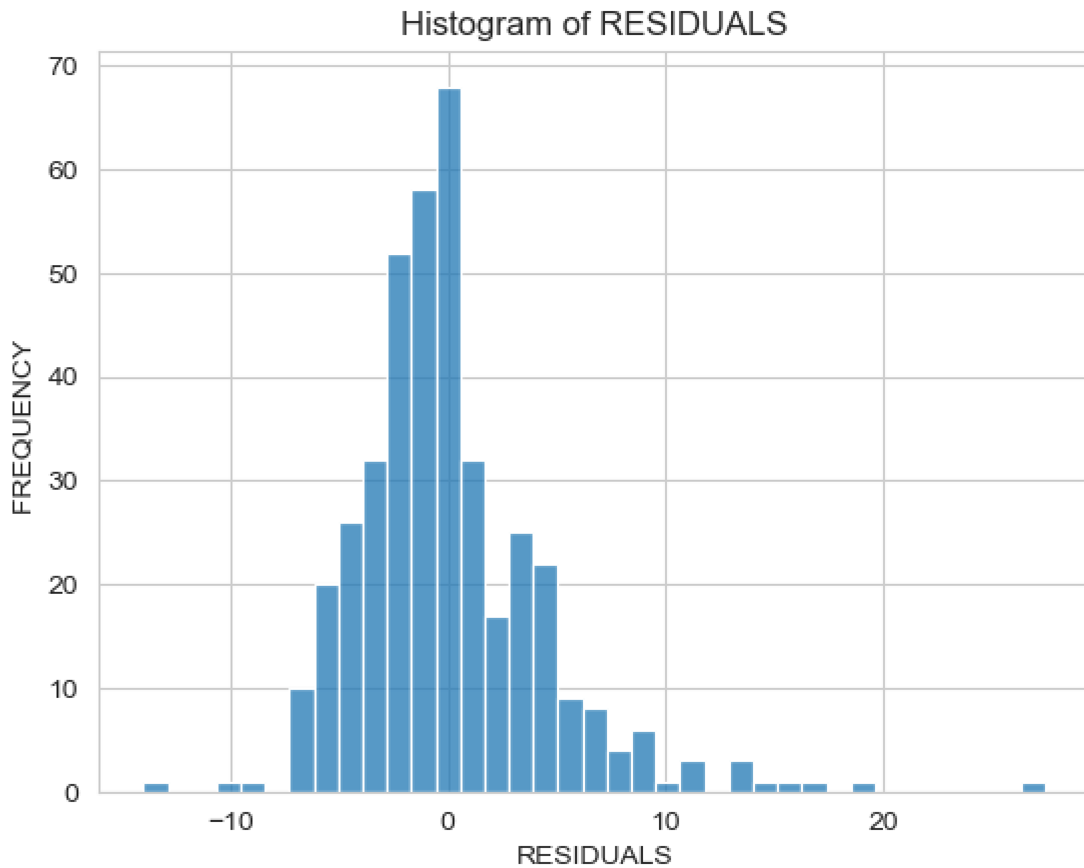
```
plt.ylabel('RESIDUALS ')
plt.show()
```



::: Predicted Vs Residuals :::

Checking Normality of of Errors

In [48]:
```
sns.histplot(y_train-y_pred)
plt.title('Histogram of RESIDUALS')
plt.xlabel('RESIDUALS')
plt.ylabel('FREQUENCY ')
plt.show()
```



::: Hist Plotting for residuals :::

Random Forest Regression

In [49]:
```
x=df.iloc[:,[-1,5,10,4,9]]
y=df.iloc[:,[-1]]
```

In [50]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=0)
```

In [51]:
```
from sklearn.ensemble import RandomForestRegressor
reg=RandomForestRegressor()
reg.fit(x_train,y_train)
```

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_9944\4176756501.py:3: DataConversionWarning:
A column-vector y was passed when a 1d array was expected. Please change the shape of
y to (n_samples,), for example using ravel().
  reg.fit(x_train,y_train)
```

Out[51]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [52]:
```
y_pred = reg.predict(x_train)
```

In [53]:
```
print("Training Accuracy:",reg.score(x_train,y_train)*100)
```

```
Training Accuracy: 99.99454861454954
```

In [54]:
```
print("Testing Accuracy:",reg.score(x_test,y_test)*100)
```

```
Testing Accuracy: 99.99119481884803
```

Visualizing the difference between actual PRICES and PREDICTED values

In [55]:
```
plt.scatter(y_train,y_pred)
plt.xlabel('PRICES')
plt.ylabel('PREDICTED PRICES')
plt.title('PRICES VS PREDICTED PRICES')
plt.show()
```


```
::: Linear Regression plotting data points :::
```

Prediction and Final Score:

Finally we made it!!!

1.Linear Regression

Training Accuracy: 77.30135569264233

Testing Accuracy: 58.9222384918251

Model Accuracy: 73.73440319905033

2. Random Forest Regressor

Training Accuracy: 99.99323673544639

Training Accuracy: 99.99323673544639

Delivered By:

```
::: M Yasir Madni :::
::: Shoaib Yaseen :::
::: Muhammad Riyan:::
::: Hassan Raza   :::
::: Raza Abbas    :::
```

In [ ]: