1. Environment Setup and Data Overview

In [62]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [63]:

```
df = pd.read_csv('BostonHousing.csv')
df.columns = [col.upper() for col in df.columns]
df.head()
```

Out[63]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEI
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36

In [64]:

```
df.shape
```

Out[64]:

```
(506, 14)
```

In [65]:

```
df.columns.tolist()
```

Out[65]:

```
['CRIM',
 'ZN',
 'INDUS',
 'CHAS',
 'NOX',
 'RM',
 'AGE',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT',
 'MEDV']
```

In [66]:

```
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  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

In [67]:

```
df.dtypes
```

Out[67]:

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX          int64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

2. Univariate Analysis

Quick Data Summary

In [68]:

```
df.describe().T
```

Out[68]:

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762

	count	mean	std	min	25%	50%	75%	max
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.9000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
MEDV	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

In [69]:

```
from pandas_summary import DataFrameSummary
dfs = DataFrameSummary(df)
dfs.summary().T
```

Out[69]:

	count	mean	std	min	25%	50%	75%	max	counts	uniqu
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762	506	5
ZN	506.0	11.363636	23.322453	0.0	0.0	0.0	12.5	100.0	506	
INDUS	506.0	11.136779	6.860353	0.46	5.19	9.69	18.1	27.74	506	
CHAS	506.0	0.06917	0.253994	0.0	0.0	0.0	0.0	1.0	506	
NOX	506.0	0.554695	0.115878	0.385	0.449	0.538	0.624	0.871	506	
RM	506.0	6.284634	0.702617	3.561	5.8855	6.2085	6.6235	8.78	506	4
AGE	506.0	68.574901	28.148861	2.9	45.025	77.5	94.075	100.0	506	3
DIS	506.0	3.795043	2.10571	1.1296	2.100175	3.20745	5.188425	12.1265	506	4
RAD	506.0	9.549407	8.707259	1.0	4.0	5.0	24.0	24.0	506	
TAX	506.0	408.237154	168.537116	187.0	279.0	330.0	666.0	711.0	506	
PTRATIO	506.0	18.455534	2.164946	12.6	17.4	19.05	20.2	22.0	506	
B	506.0	356.674032	91.294864	0.32	375.3775	391.44	396.225	396.9	506	3
LSTAT	506.0	12.653063	7.141062	1.73	6.95	11.36	16.955	37.97	506	4
MEDV	506.0	22.532806	9.197104	5.0	17.025	21.2	25.0	50.0	506	2

In [70]:

```
df.isna().sum()
```

Out[70]:
```
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
```

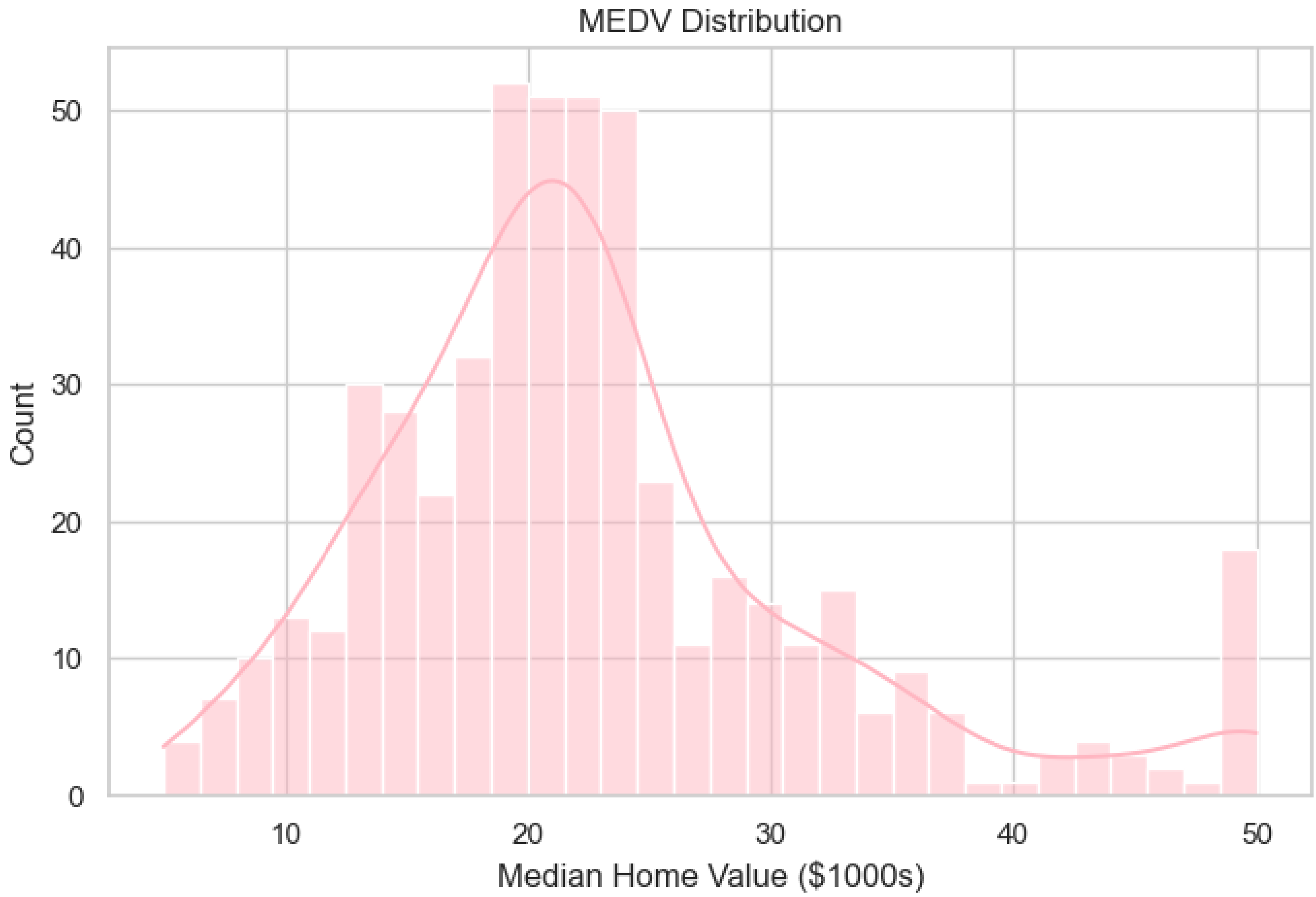
In [71]:
```
df.duplicated().sum()
```

Out[71]:
```
0
```

Target Variable Exploration

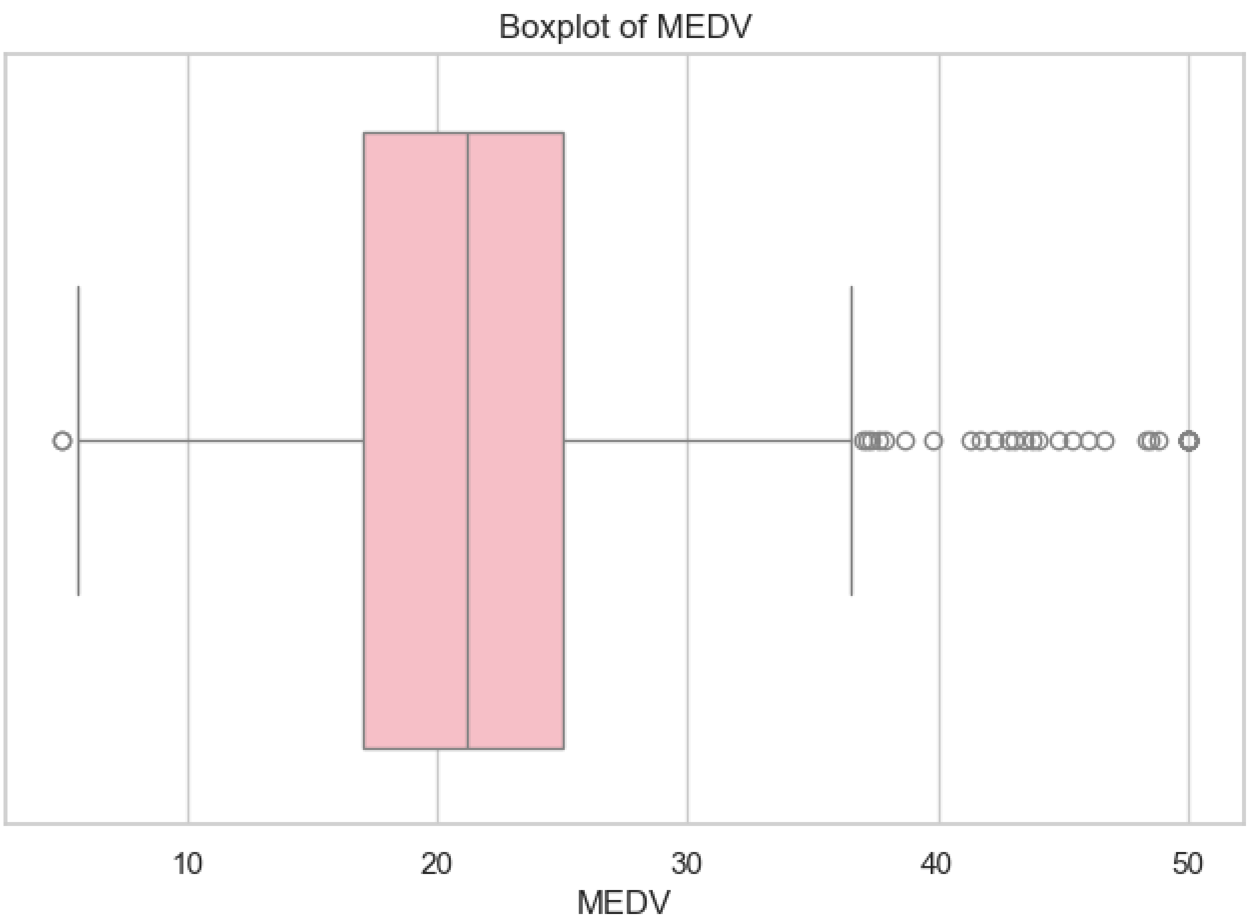
In [72]:
```
sns.set_theme(
    style="whitegrid",
    palette="pastel",
    context="notebook"
)
```

In [73]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df['MEDV'], kde=True, color='Lightpink', bins=30)
plt.title("MEDV Distribution")
plt.xlabel("Median Home Value ($1000s)")
plt.show()
```



In [74]:

```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['MEDV'],color = 'lightpink')
plt.title("Boxplot of MEDV")
plt.show()
```



In [75]:

```
df['MEDV'].skew()
```

Out[75]:

1.1080984082549072

In [76]:

```
df['MEDV'].kurtosis()
```

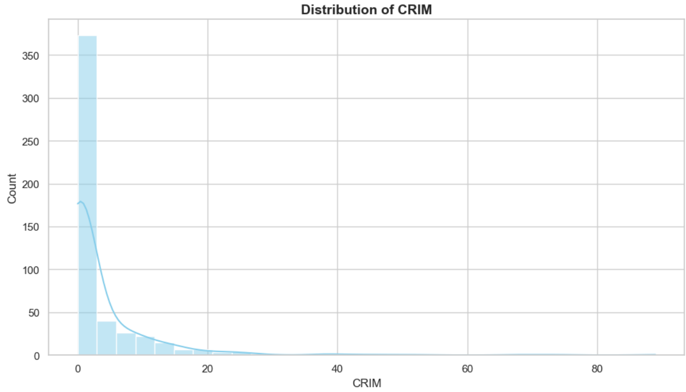
Out[76]:

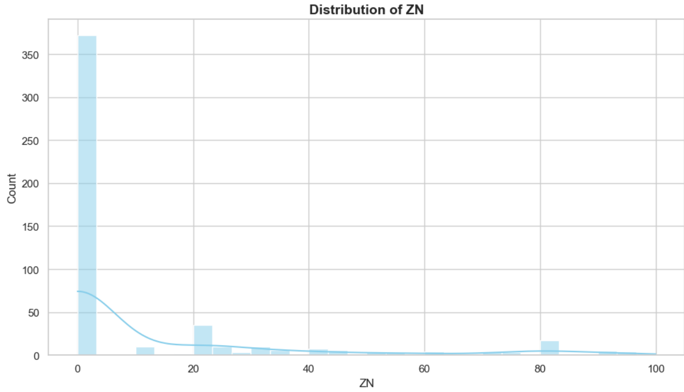
1.495196944165818

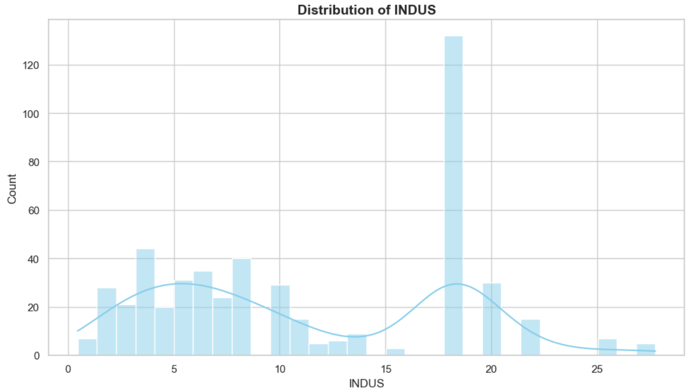
Feature Inspection

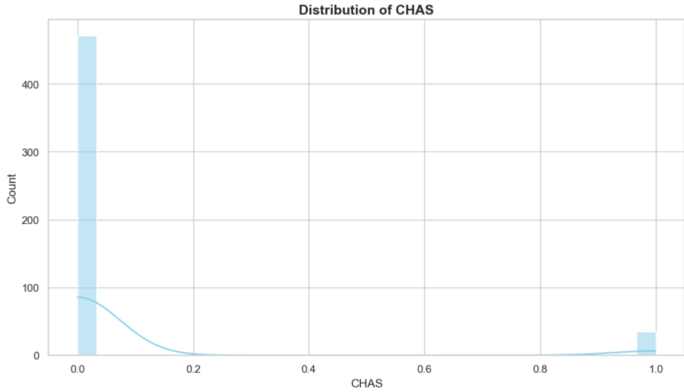
In [77]:

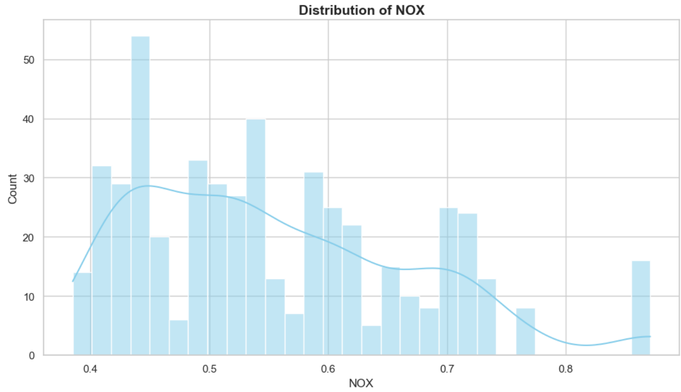
```
plt.figure(figsize=(20, 40))
for i, col in enumerate(df.columns):
    plt.subplot(7, 2, i+1)
    sns.histplot(df[col], kde=True, bins=30, color='skyblue')
    plt.title(f"Distribution of {col}",fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()
```

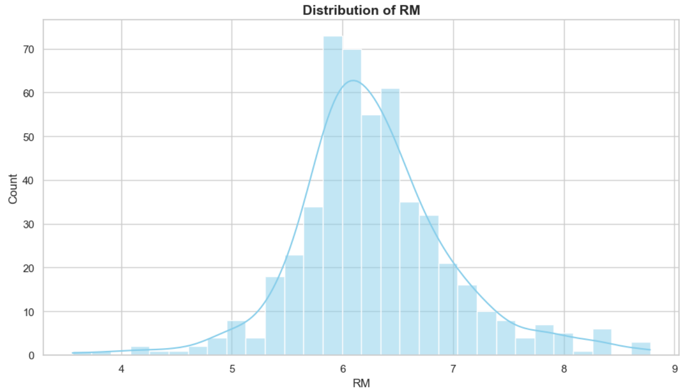

Distribution of CRIM
Count
CRIM

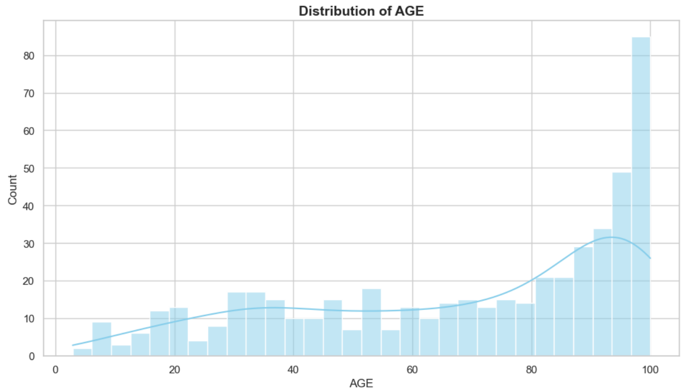

Distribution of ZN
Count
ZN

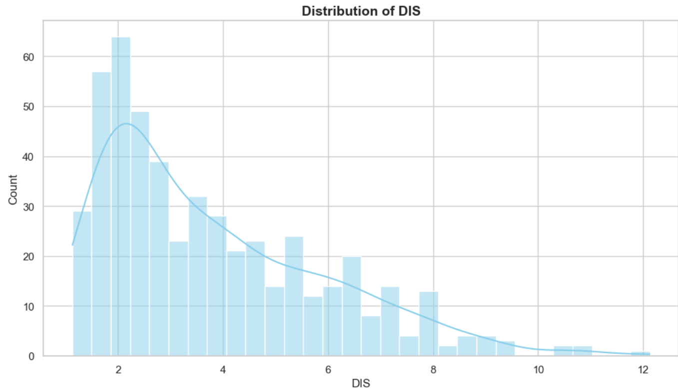

Distribution of INDUS
Count
INDUS

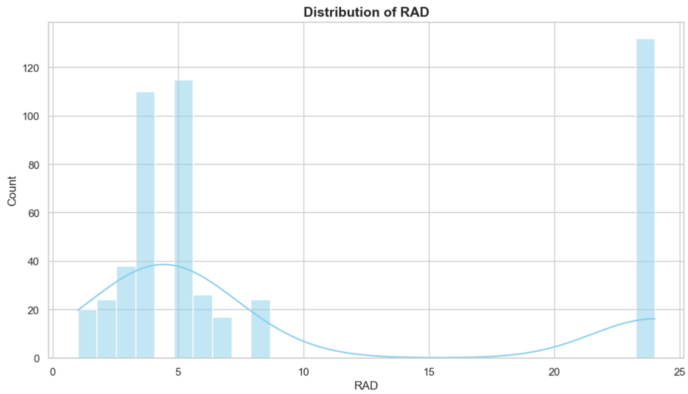

Distribution of CHAS
Count
CHAS

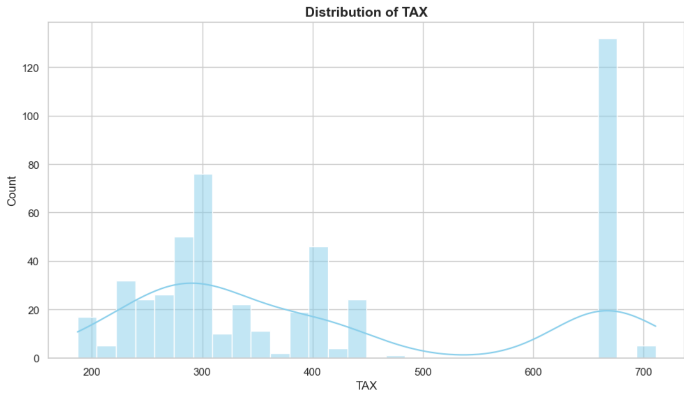

Distribution of NOX
Count
NOX


Distribution of RM
Count
RM


Distribution of AGE
Count
AGE

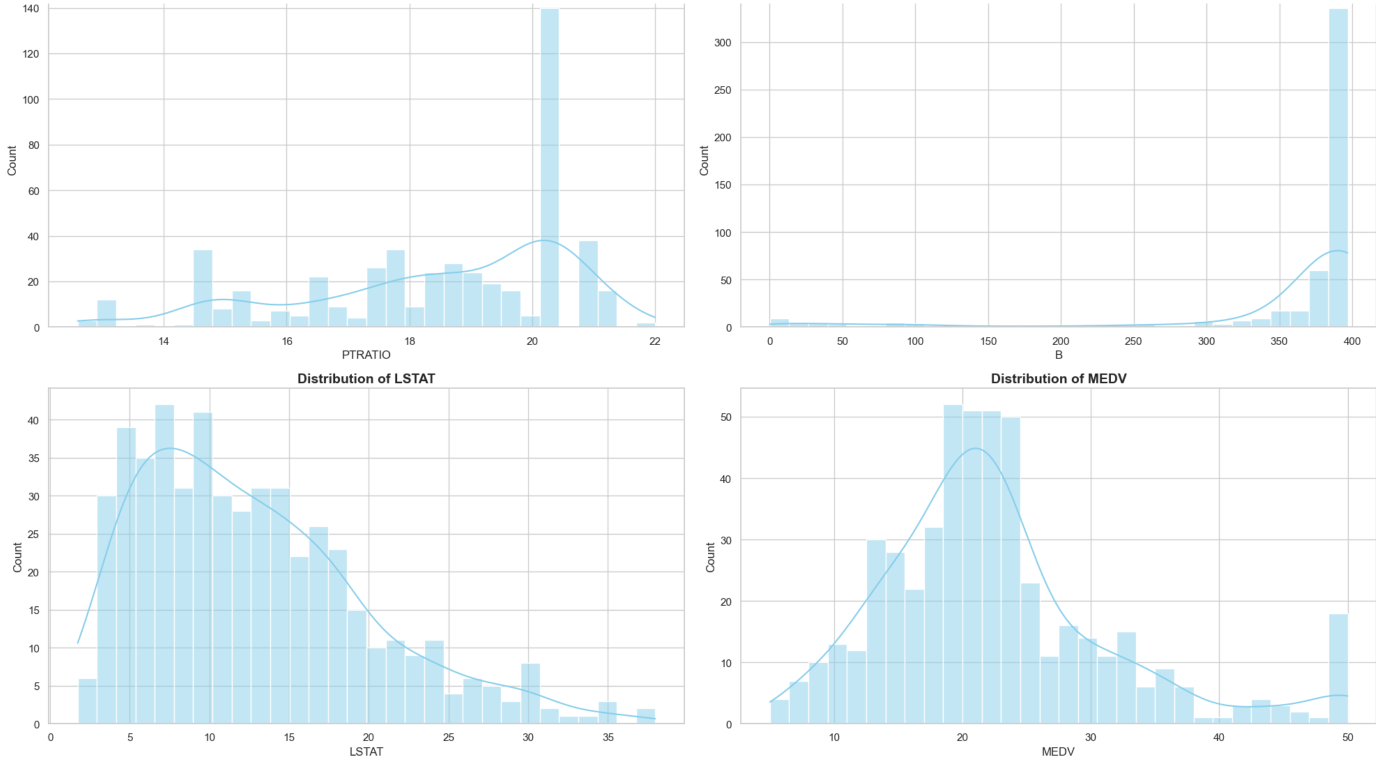

Distribution of DIS
Count
DIS


Distribution of RAD
Count
RAD


Distribution of TAX
Count
TAX

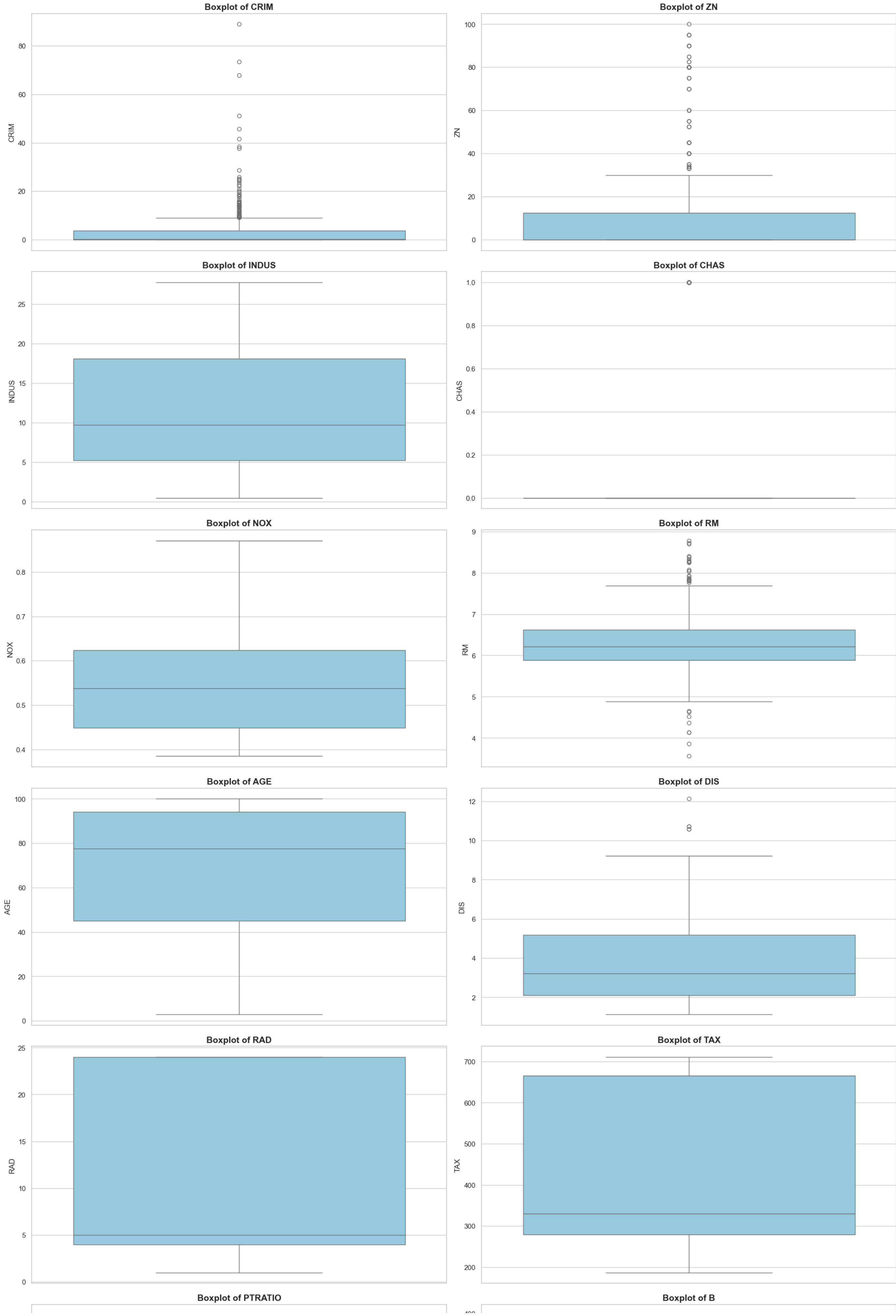
Distribution of PTRATIO

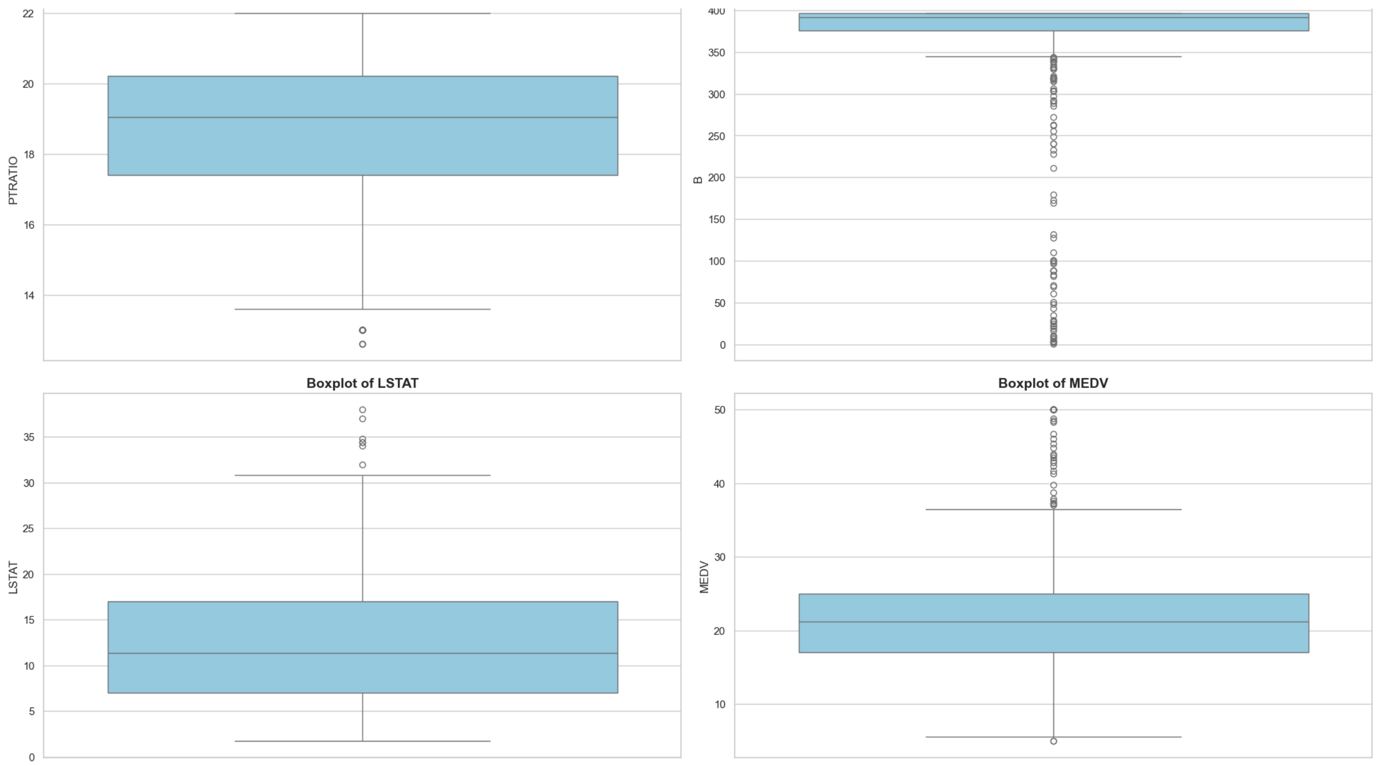

Distribution of B



In [78]:

```
plt.figure(figsize=(20, 40))
for i, col in enumerate(df.columns):
    plt.subplot(7, 2, i+1)
    sns.boxplot(df[col], color='skyblue')
    plt.title(f"Boxplot of {col}",fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()
```


Boxplot of CRIM
CRIM
Boxplot of ZN
ZN
Boxplot of INDUS
INDUS
Boxplot of CHAS
CHAS
Boxplot of NOX
NOX
Boxplot of RM
RM
Boxplot of AGE
AGE
Boxplot of DIS
DIS
Boxplot of RAD
RAD
Boxplot of TAX
TAX
Boxplot of PTRATIO
Boxplot of B



In [79]:

```
skew = df.skew().sort_values(ascending = False)
skew
```

Out[79]:

```
CRIM       5.223149
CHAS       3.405904
ZN         2.225666
MEDV       1.108098
DIS        1.011781
RAD        1.004815
LSTAT      0.906460
NOX        0.729308
TAX        0.669956
RM         0.403612
INDUS      0.295022
AGE       -0.598963
PTRATIO   -0.802325
B         -2.890374
dtype: float64
```

In [80]:

```
kurt = df.kurtosis().sort_values(ascending = False)
kurt
```

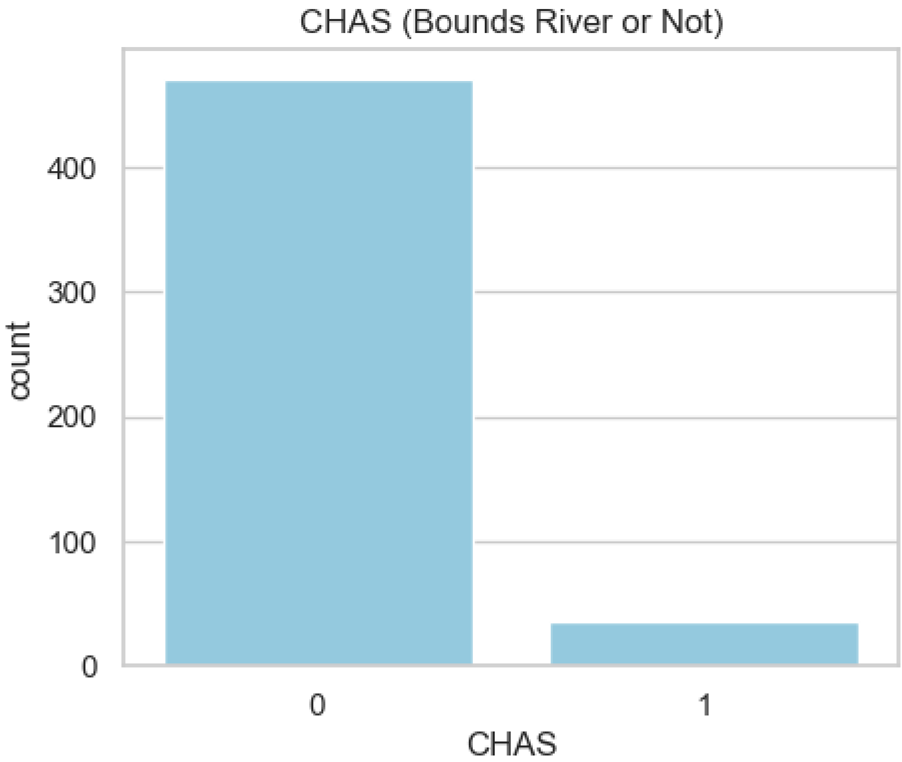
Out[80]:

```
CRIM       37.130509
CHAS        9.638264
B           7.226818
ZN          4.031510
RM          1.891500
MEDV        1.495197
LSTAT       0.493240
DIS         0.487941
NOX        -0.064667
PTRATIO    -0.285091
```

```
RAD         -0.867232
AGE         -0.967716
TAX         -1.142408
INDUS       -1.233540
dtype: float64
```

In [81]:

```
plt.figure(figsize = (5,4))
sns.countplot(x='CHAS', data=df,color = 'skyblue')
plt.title("CHAS (Bounds River or Not)")
plt.show()
```

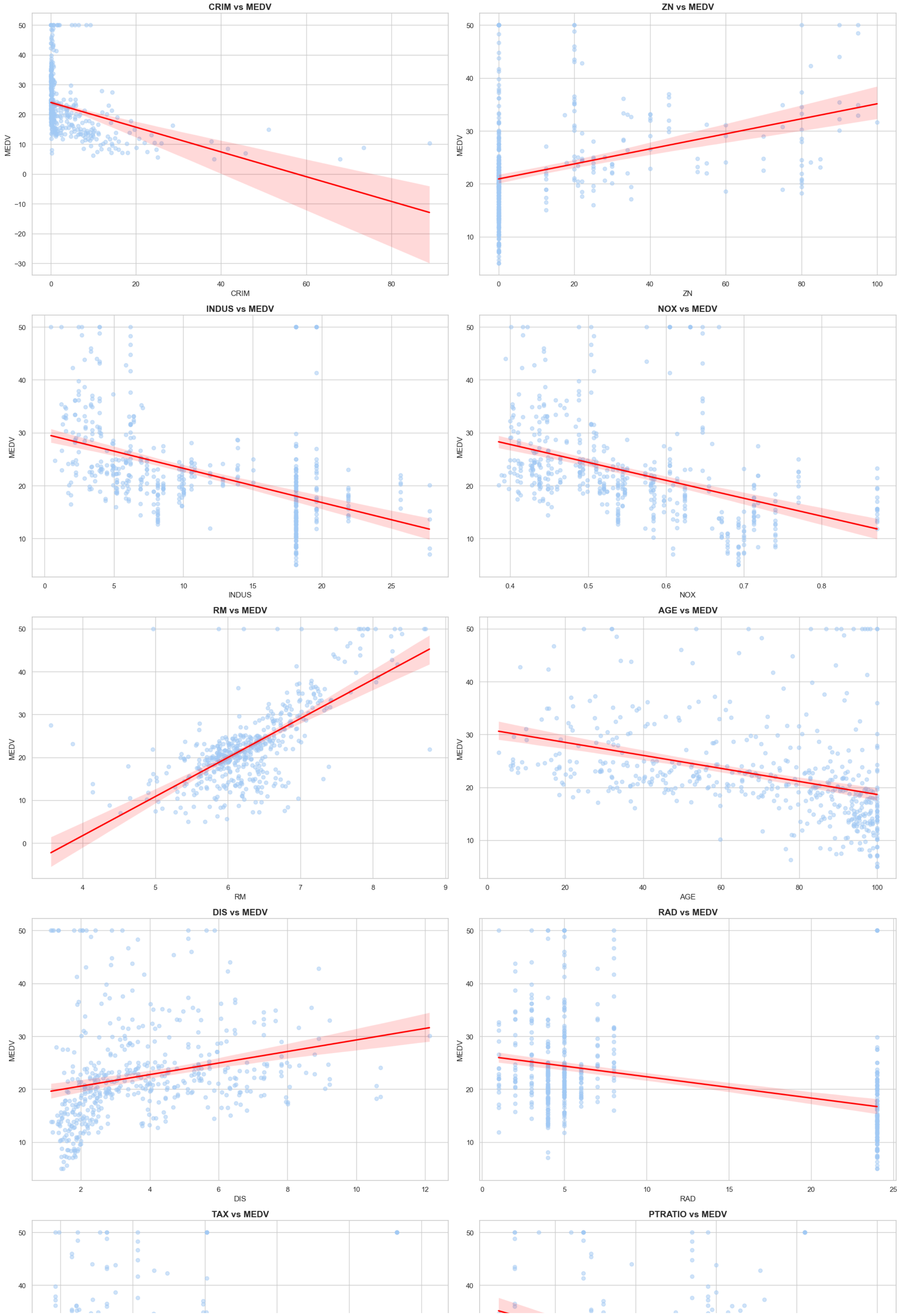


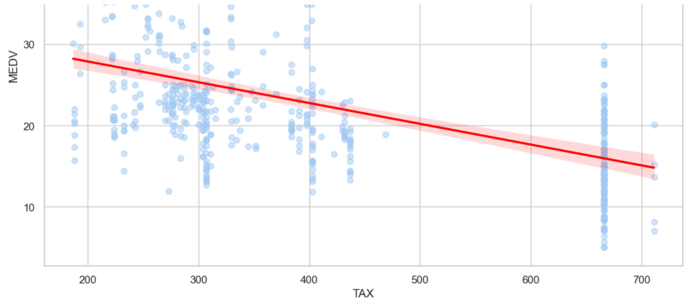
3. Bivariate Analysis

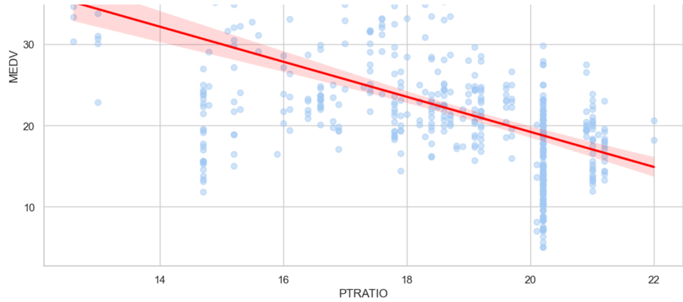
Scatter Plot Analysis with Target Variable

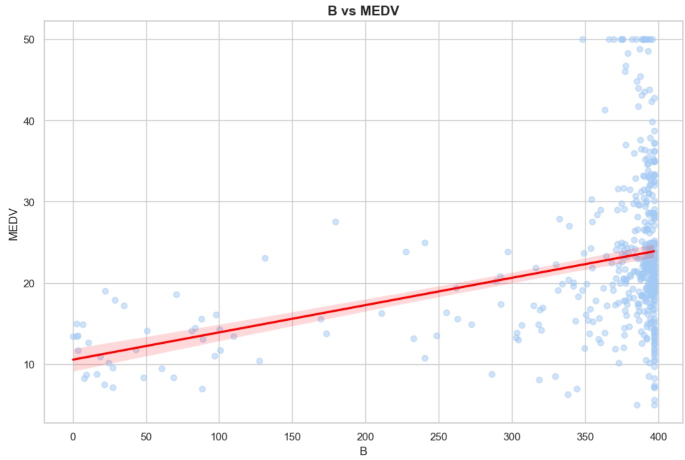
In [82]:

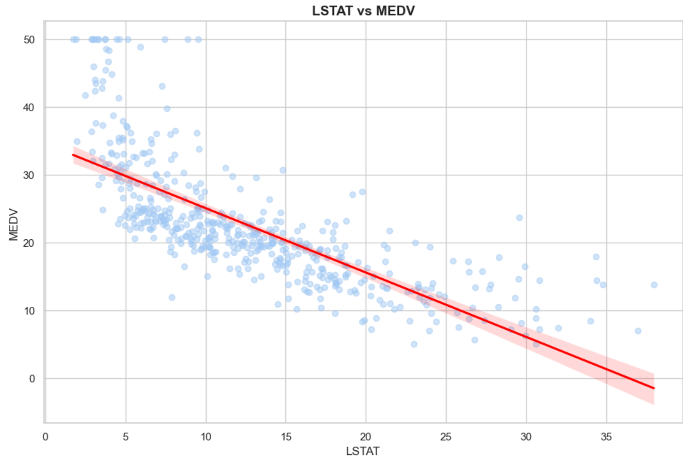
```
features = df.drop(columns=(['MEDV','CHAS'])).columns

plt.figure(figsize=(20, 40))
for i, feature in enumerate(features):
    plt.subplot(6, 2, i + 1)
    sns.regplot(x=df[feature], y=df['MEDV'], scatter_kws={'alpha': 0.5}, line_kws={"colo
    plt.title(f'{feature} vs MEDV',fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()
```


CRIM vs MEDV
MEDV
CRIM
ZN vs MEDV
MEDV
ZN
INDUS vs MEDV
MEDV
INDUS
NOX vs MEDV
MEDV
NOX
RM vs MEDV
MEDV
RM
AGE vs MEDV
MEDV
AGE
DIS vs MEDV
MEDV
DIS
RAD vs MEDV
MEDV
RAD
TAX vs MEDV
PTRATIO vs MEDV









Correlation Analysis

In [83]:

```
corr_matrix = df.corr()
corr_matrix
```

Out[83]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RA
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.6255
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.3119
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.5951
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.0073
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.6114
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.2098
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.4560
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.4945
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.0000
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.9102
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.4647
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.4444
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.4886
MEDV	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.3816

In [84]:

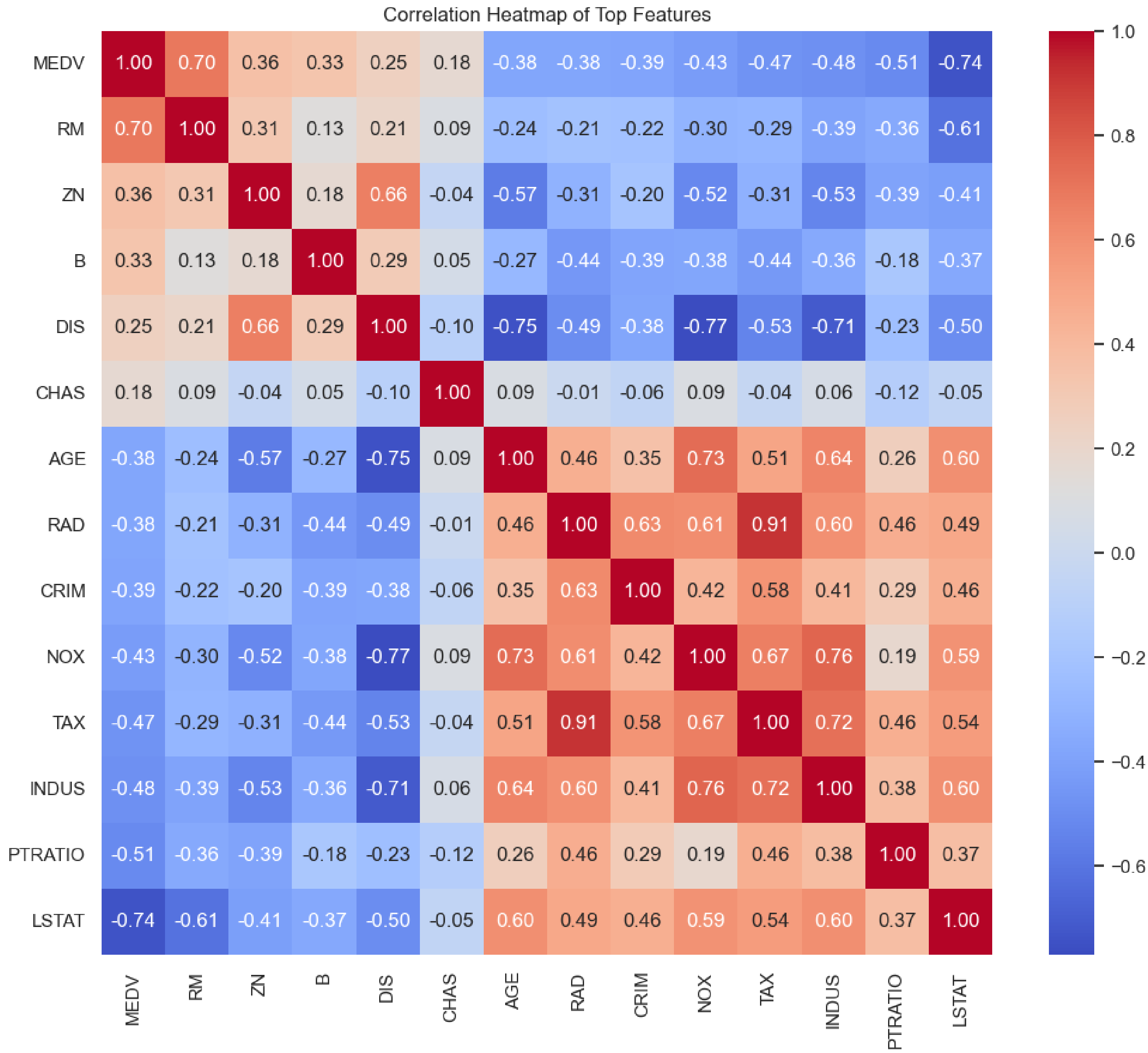
```
top_corr = corr_matrix['MEDV'].sort_values(ascending=False)
top_corr
```

Out[84]:

```
MEDV       1.000000
RM         0.695360
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
AGE       -0.376955
RAD       -0.381626
CRIM      -0.388305
NOX       -0.427321
TAX       -0.468536
INDUS     -0.483725
PTRATIO   -0.507787
LSTAT     -0.737663
Name: MEDV, dtype: float64
```

In [85]:

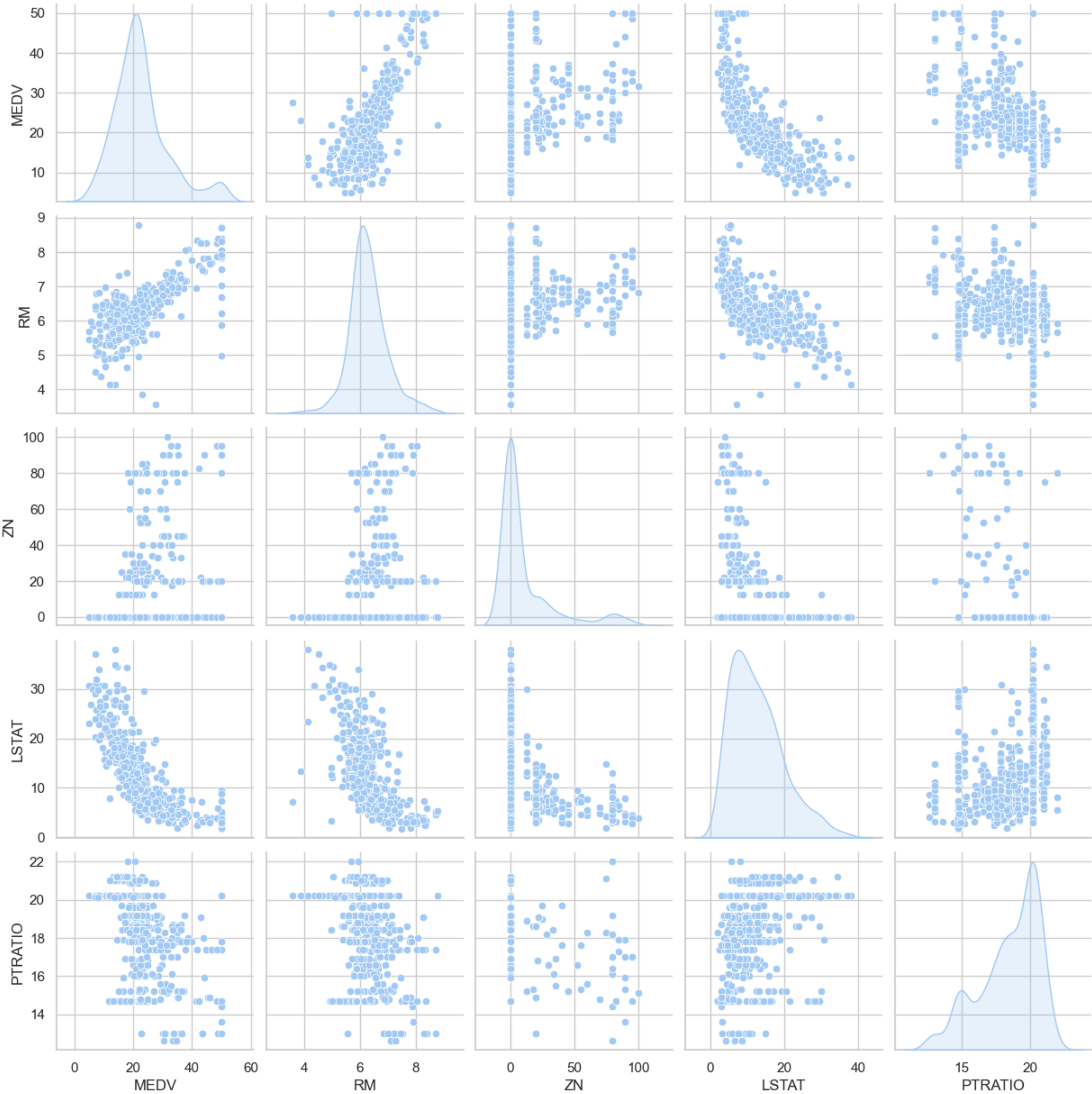
```
top_corr_features = top_corr.index.tolist()
plt.figure(figsize=(12, 10))
sns.heatmap(df[top_corr_features].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap of Top Features")
plt.show()
```



In [86]:

```
plt.figure(figsize = (12,15))
sns.pairplot(df[['MEDV','RM','ZN','LSTAT','PTRATIO']], diag_kind='kde')
plt.show()
```

```
<Figure size 1200x1500 with 0 Axes>
```



Statistical Significance Tests

T-test For Categorical Feature

In [87]:

```
from scipy.stats import ttest_ind

group0 = df[df['CHAS'] == 0]['MEDV']
group1 = df[df['CHAS'] == 1]['MEDV']

t_stat, p_val = ttest_ind(group0, group1)
print(f"t-statistic: {t_stat:.2f}, p-value: {p_val:.4f}")
```

```
t-statistic: -4.00, p-value: 0.0001
```

Pearson's Correlation Coefficient For Numerical Fetures

In [88]:

```
from scipy.stats import pearsonr

target_col = 'MEDV'
numerical_cols = df.select_dtypes(include=['number']).columns.drop(['MEDV','CHAS'])

print(f"{'Feature':<20} {'Correlation':<15} {'P-Value'}")
print("-" * 50)

for col in numerical_cols:
    corr, p_val = pearsonr(df[col], df[target_col])
    print(f"{col:<20} {corr:<15.4f} {p_val:.4f}")
```

```
Feature              Correlation     P-Value
--------------------------------------------------
CRIM                 -0.3883         0.0000
ZN                   0.3604          0.0000
INDUS                -0.4837         0.0000
NOX                  -0.4273         0.0000
RM                   0.6954          0.0000
AGE                  -0.3770         0.0000
DIS                  0.2499          0.0000
RAD                  -0.3816         0.0000
TAX                  -0.4685         0.0000
PTRATIO              -0.5078         0.0000
B                    0.3335          0.0000
LSTAT                -0.7377         0.0000
```

4. Data pre-processing

Handling Missing Values

In [89]:

```
df.isna().sum()
```

Out[89]:

```
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
```

Handling Outliers

In [90]:

```
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import StandardScaler

X1 = df.drop(columns=['MEDV'])

scaler = StandardScaler()
X_scaled1 = scaler.fit_transform(X1)

lof = LocalOutlierFactor(n_neighbors=20, contamination=0.03)  #
y_pred = lof.fit_predict(X_scaled1)

df_lof_cleaned = df[y_pred == 1]

print(f"Original shape: {df.shape}")
print(f"After removing outliers: {df_lof_cleaned.shape}")
```

```
Original shape: (506, 14)
After removing outliers: (490, 14)
```

In [91]:

```
df_lof_cleaned.head()
```

Out[91]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MED
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36

Features Transformation

In [92]:

```
df_transformed = df_lof_cleaned.copy()

df_transformed['CRIM'] = np.log1p(df_transformed['CRIM'])
df_transformed['ZN'] = np.log1p(df_transformed['ZN'])
df_transformed['DIS'] = np.log1p(df_transformed['DIS'])
df_transformed['RAD'] = np.log1p(df_transformed['RAD'])

df_transformed['B'] = np.log1p(df_transformed['B'].max() + 1 - df_transformed['B'])
```

In [93]:

```
df_transformed.head()
```

Out[93]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.006300	2.944439	2.31	0	0.538	6.575	65.2	1.627278	0.693147	296	15.3	0.693147
1	0.026944	0.000000	7.07	0	0.469	6.421	78.9	1.786261	1.098612	242	17.8	0.693147
2	0.026924	0.000000	7.07	0	0.469	7.185	61.1	1.786261	1.098612	242	17.8	1.803359
3	0.031857	0.000000	2.18	0	0.458	6.998	45.8	1.954757	1.386294	222	18.7	1.451614
4	0.066770	0.000000	2.18	0	0.458	7.147	54.2	1.954757	1.386294	222	18.7	0.693147

Feature Encoding

we don't have to do this as values in "CHAS" columns are already 0 and 1 format

Feature Scaling

In [94]:

```
x = df_transformed.drop(['MEDV'],axis=1)
y = df_transformed['MEDV']
```

In [95]:

```
x.head()
```

Out[95]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.006300	2.944439	2.31	0	0.538	6.575	65.2	1.627278	0.693147	296	15.3	0.693147
1	0.026944	0.000000	7.07	0	0.469	6.421	78.9	1.786261	1.098612	242	17.8	0.693147
2	0.026924	0.000000	7.07	0	0.469	7.185	61.1	1.786261	1.098612	242	17.8	1.803359
3	0.031857	0.000000	2.18	0	0.458	6.998	45.8	1.954757	1.386294	222	18.7	1.451614
4	0.066770	0.000000	2.18	0	0.458	7.147	54.2	1.954757	1.386294	222	18.7	0.693147

In [96]:

```
y.head()
```

Out[96]:

```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: MEDV, dtype: float64
```

In [97]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
```

In [98]:

```
X_scaled.head()
```

Out[98]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	
0	-0.786545	1.208551	-1.264745	-0.250812	-0.116444	0.402017	-0.103396	0.330157	-1.813638	-0.64
1	-0.765069	-0.599058	-0.570269	-0.250812	-0.718734	0.178124	0.384718	0.717063	-1.262664	-0.96
2	-0.765089	-0.599058	-0.570269	-0.250812	-0.718734	1.288868	-0.249474	0.717063	-1.262664	-0.96
3	-0.759957	-0.599058	-1.283711	-0.250812	-0.814752	1.016997	-0.794593	1.127119	-0.871742	-1.08
4	-0.723635	-0.599058	-1.283711	-0.250812	-0.814752	1.233621	-0.495312	1.127119	-0.871742	-1.08

In [99]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_s
```

5. Model Training and Evaluation

Model Building with Statsmodels

Full Model (All Features)

In [100]:

```
import statsmodels.api as sm

x_state =sm.add_constant(x)
```

In [101]:

```
x_state, y = x_state.align(y, join='inner', axis=0)
```

In [102]:

```
stat1 = sm.OLS(y, x_state)
results1 = stat1.fit()
results1.summary()
```

Out[102]:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.772
Model:	OLS	**Adj. R-squared:**	0.766
Method:	Least Squares	**F-statistic:**	124.1
Date:	Fri, 08 Aug 2025	**Prob (F-statistic):**	1.43e-143
Time:	19:26:40	**Log-Likelihood:**	-1407.0
No. Observations:	490	**AIC:**	2842.
Df Residuals:	476	**BIC:**	2901.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	40.0164	5.034	7.950	0.000	30.125	49.908
CRIM	-0.1143	0.548	-0.209	0.835	-1.190	0.962
ZN	0.2792	0.184	1.521	0.129	-0.082	0.640
INDUS	0.0037	0.057	0.065	0.948	-0.108	0.116
CHAS	1.8624	0.864	2.156	0.032	0.165	3.560
NOX	-21.5301	3.733	-5.768	0.000	-28.865	-14.195
RM	4.8537	0.410	11.836	0.000	4.048	5.659
AGE	-0.0127	0.013	-1.007	0.314	-0.037	0.012
DIS	-8.3814	1.073	-7.810	0.000	-10.490	-6.273
RAD	2.0516	0.641	3.202	0.001	0.793	3.311
TAX	-0.0112	0.003	-3.625	0.000	-0.017	-0.005
PTRATIO	-0.8929	0.123	-7.240	0.000	-1.135	-0.651
B	-0.1134	0.149	-0.762	0.446	-0.406	0.179
LSTAT	-0.4840	0.052	-9.320	0.000	-0.586	-0.382

Omnibus:	110.983	**Durbin-Watson:**	1.128
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	368.301
Skew:	1.026	**Prob(JB):**	1.06e-80
Kurtosis:	6.719	**Cond. No.**	1.28e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.28e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Reduced Model (Only Statistically Significant Features – p < 0.05)

In [103]:

```
new_x = x_state.drop(['CRIM','ZN','INDUS','AGE','B'],axis = 1)
new_x.head()
```

Out[103]:

	const	CHAS	NOX	RM	DIS	RAD	TAX	PTRATIO	LSTAT
0	1.0	0	0.538	6.575	1.627278	0.693147	296	15.3	4.98
1	1.0	0	0.469	6.421	1.786261	1.098612	242	17.8	9.14
2	1.0	0	0.469	7.185	1.786261	1.098612	242	17.8	4.03
3	1.0	0	0.458	6.998	1.954757	1.386294	222	18.7	2.94
4	1.0	0	0.458	7.147	1.954757	1.386294	222	18.7	5.33

In [104]:

```
stat2 = sm.OLS(y, new_x)
results2 = stat2.fit()
results2.summary()
```

Out[104]:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.770
Model:	OLS	**Adj. R-squared:**	0.766
Method:	Least Squares	**F-statistic:**	201.2
Date:	Fri, 08 Aug 2025	**Prob (F-statistic):**	3.63e-148
Time:	19:26:40	**Log-Likelihood:**	-1409.4
No. Observations:	490	**AIC:**	2837.
Df Residuals:	481	**BIC:**	2874.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	40.6238	4.933	8.235	0.000	30.931	50.317
CHAS	1.7385	0.853	2.037	0.042	0.062	3.415
NOX	-23.3781	3.457	-6.762	0.000	-30.171	-16.585
RM	4.8227	0.389	12.384	0.000	4.058	5.588
DIS	-7.4037	0.874	-8.472	0.000	-9.121	-5.687
RAD	1.9914	0.540	3.686	0.000	0.930	3.053
TAX	-0.0108	0.003	-4.075	0.000	-0.016	-0.006
PTRATIO	-0.9770	0.110	-8.879	0.000	-1.193	-0.761
LSTAT	-0.5031	0.046	-10.883	0.000	-0.594	-0.412

Omnibus:	108.751	**Durbin-Watson:**	1.125
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	344.323
Skew:	1.021	**Prob(JB):**	1.70e-75
Kurtosis:	6.563	**Cond. No.**	1.24e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

VIF-Based Model (Remove features with VIF > 5)

In [105]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data['Feature'] = x_state.columns

vif_data['VIF'] = [
    variance_inflation_factor(x_state.values, i)
    for i in range(x_state.shape[1])
]

print(vif_data)
```

```
    Feature         VIF
0     const  660.332039
1      CRIM    7.220934
2        ZN    2.331247
3     INDUS    3.968059
4      CHAS    1.082510
5       NOX    4.765554
6        RM    2.073471
7       AGE    3.253687
8       DIS    5.067188
9       RAD    5.793705
10      TAX    6.845770
11  PTRATIO    1.858438
12        B    1.301130
13    LSTAT    3.501780
```

In [106]:

```
new_x = x_state.drop(['CRIM'], axis =1)
```

In [107]:

```
stat3 = sm.OLS(y, new_x)
results3 = stat3.fit()
results3.summary()
```

Out[107]:

OLS Regression Results

Dep. Variable:	MEDV	**R-squared:**	0.772
Model:	OLS	**Adj. R-squared:**	0.766
Method:	Least Squares	**F-statistic:**	134.7
Date:	Fri, 08 Aug 2025	**Prob (F-statistic):**	1.23e-144
Time:	19:26:40	**Log-Likelihood:**	-1407.0
No. Observations:	490	**AIC:**	2840.
Df Residuals:	477	**BIC:**	2895.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

const	40.1956	4.955	8.112	0.000	30.459	49.932
ZN	0.2771	0.183	1.513	0.131	-0.083	0.637
INDUS	0.0047	0.057	0.083	0.934	-0.107	0.116
CHAS	1.8683	0.862	2.166	0.031	0.174	3.563
NOX	-21.6575	3.679	-5.887	0.000	-28.886	-14.429
RM	4.8514	0.410	11.846	0.000	4.047	5.656
AGE	-0.0126	0.013	-1.000	0.318	-0.037	0.012
DIS	-8.3450	1.058	-7.889	0.000	-10.423	-6.267
RAD	1.9868	0.560	3.548	0.000	0.887	3.087
TAX	-0.0114	0.003	-3.852	0.000	-0.017	-0.006
PTRATIO	-0.8932	0.123	-7.250	0.000	-1.135	-0.651
B	-0.1173	0.147	-0.796	0.427	-0.407	0.172
LSTAT	-0.4867	0.050	-9.675	0.000	-0.585	-0.388

Omnibus:	110.033	**Durbin-Watson:**	1.128
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	363.556
Skew:	1.018	**Prob(JB):**	1.13e-79
Kurtosis:	6.696	**Cond. No.**	1.26e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.26e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [108]:

```
vif_data = pd.DataFrame()
vif_data['Feature'] = new_x.columns

vif_data['VIF'] = [
    variance_inflation_factor(new_x.values, i)
    for i in range(new_x.shape[1])
]

print(vif_data)
```

```
    Feature         VIF
0     const  641.133335
1        ZN    2.324302
2     INDUS    3.940057
3      CHAS    1.081354
4       NOX    4.637969
5        RM    2.072018
6       AGE    3.246709
7       DIS    4.932772
8       RAD    4.433999
```

```
9        TAX    6.279230
10  PTRATIO    1.858224
11        B    1.280264
12    LSTAT    3.291567
```

Model Building with sklearn Linear Regression

In [109]:

```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
```

In [110]:

```
elastic_net = ElasticNet()
```

In [111]:

```
param_grid = {
    'alpha': [0.01, 0.1, 1.0, 10.0, 100.0],
    'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9, 1.0]
}

grid_search = GridSearchCV(estimator=elastic_net,
                           param_grid=param_grid,
                           scoring='neg_mean_squared_error',
                           cv=5,
                           n_jobs=-1)


grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Test MSE:", mse)
print("Test R2 Score:", r2)
```

```
Best Parameters: {'alpha': 0.01, 'l1_ratio': 0.1}
Test MSE: 15.629724406696106
Test R2 Score: 0.7803442899953164
```

Model Building with sklearn Random Forest Regressor

In [112]:

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state=42)

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
```

```
    'max_features': ['auto', 'sqrt', 'log2']
}

grid_search = GridSearchCV(estimator=rf,
                           param_grid=param_grid,
                           cv=5,
                           n_jobs=-1,
                           scoring='r2',
                           verbose=1)

grid_search.fit(X_train, y_train)

best_rf = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

y_pred = best_rf.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Random Forest with GridSearch Results:")
print("Test MSE:", mse)
print("Test R² Score:", r2)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best Parameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min
_samples_split': 2, 'n_estimators': 200}
Random Forest with GridSearch Results:
Test MSE: 8.79938058418366
Test R² Score: 0.8763360031484482
```

In [113]:

```
importances = best_rf.feature_importances_
feature_names = X_train.columns

feat_imp = pd.Series(importances, index=feature_names).sort_values(ascending=False)
feat_imp
```

Out[113]:

```
RM         0.289698
LSTAT      0.252946
PTRATIO    0.073926
INDUS      0.072950
CRIM       0.063911
NOX        0.055424
DIS        0.053130
TAX        0.043558
AGE        0.035192
B          0.025056
RAD        0.015712
ZN         0.014440
CHAS       0.004058
dtype: float64
```

In [114]:

```
low_importance = ['CHAS', 'ZN', 'RAD']
X_train_reduced = X_train.drop(columns=low_importance)
X_test_reduced = X_test.drop(columns=low_importance)

best_rf.fit(X_train_reduced, y_train)
```

```python
y_pred = best_rf.predict(X_test_reduced)

from sklearn.metrics import r2_score
print("R2:", r2_score(y_test, y_pred))
print('MSE',mean_squared_error(y_test, y_pred))
```

```
R2: 0.8693199890669581
MSE 9.298608974489792
```

In [ ]:

```python

```

In [ ]:

```python

```

🏡 Boston Housing Price Prediction

An end-to-end **Data Science Regression Project** built to predict housing prices using machine learning and statistical modeling. This project covers the complete data science pipeline — from **EDA and preprocessing to model building, hyperparameter tuning, and evaluation** — using the well-known **Boston Housing Dataset**.

🚀 Project Highlights

✅ Hands-on implementation of both **Statistical Models** and **Machine Learning Algorithms**
✅ Real-world **Data Preprocessing Techniques** including outlier detection using **LOF**
✅ Thorough **Exploratory Data Analysis (EDA)**: Univariate, Bivariate, and Statistical Tests
✅ Performed **Feature Engineering**, **Scaling**, and **Multicollinearity Check (VIF)**
✅ Used **Grid Search CV** for Hyperparameter Tuning
✅ Compared model performance using **MSE** and **R² Score**
✅ Feature importance analysis from **Random Forest**

📂 Table of Contents

📊 1. Dataset Overview

The Boston Housing Dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass.

Target Variable: `MEDV` (Median value of owner-occupied homes in $1000s)
Features: 14 numerical and categorical predictors like `RM` (rooms), `LSTAT` (lower status population %), `CRIM` (crime rate), etc.

🧱 2. Project Structure

```
1. Environment Setup and Data Overview
2. Univariate Analysis
3. Bivariate Analysis
4. Data Preprocessing
5. Model Training & Evaluation
```

🔍 3. Step-by-Step Workflow

📌 Step 1: Environment Setup & Data Overview

- Imported all essential libraries.
- Loaded the dataset and did initial data screening (`.info()` , `.describe()` , null check).

📌 Step 2: Univariate Analysis

- Visualized **distribution of features** and the **target** using histograms and boxplots.
- Calculated and interpreted **skewness** and **kurtosis** to understand the shape of the distributions.

📌 Step 3: Bivariate Analysis

- Created **scatter plots** between each numerical feature and the target variable to observe relationships.
- Generated **correlation heatmap** to identify strong and weak relationships.
- Performed **statistical significance tests**:
  - **Pearson correlation** for numerical vs numerical
  - **T-test** for binary categorical vs numerical

📌 Step 4: Data Preprocessing

- ✅ No missing values were found.
- 🔍 Detected **outliers** using **Local Outlier Factor (LOF)** and removed them.
- 📏 Scaled features using **StandardScaler** for linear models.

📌 Step 5: Model Training & Evaluation

✔️ StatsModels (OLS)

- Built a baseline regression model.

- Removed statistically insignificant variables.
- Checked **VIF values** to reduce multicollinearity.

✔️ Scikit-learn Linear Regression

- Trained a model using scaled features.
- Applied **GridSearchCV** to explore hyperparameters.

✔️ Random Forest Regressor

- Used **ensemble technique** to capture non-linearities.
- Tuned with **GridSearchCV**.
- Extracted **feature importances** for interpretation.

📈 Evaluation Metrics

- Used **R^2 Score** and **Mean Squared Error (MSE)** for model comparison.
- Compared models on performance and interpretability.

🧰 4. Tools and Libraries

Tool	Purpose
`Pandas`	Data manipulation and wrangling
`NumPy`	Numerical operations
`Matplotlib`	Data visualization
`Seaborn`	Statistical plots
`Scikit-learn`	Machine learning models & tools
`Statsmodels`	Statistical modeling
`SciPy`	Hypothesis testing

🏁 5. Results Summary

Model	R^2 Score	MSE
StatsModels OLS	~0.77	-
Linear Regression (SKL)	~0.78	~15.63
Random Forest Regressor	~0.86	~9.30

- 📌 `RM` (average number of rooms per dwelling) and `LSTAT` (lower status population %) were the most influential features.
- Random Forest gave the best performance, but the Linear model offers more interpretability.

📬 6. Contact

Made with ❤️ by **Mit Trivedi**
📧 Email: trivedimit04@gmail.com
🔗 LinkedIn | GitHub

⭐ If you found this useful, give it a star and share it! ⭐

In [ ]:

```

```

In [ ]:

```

```