Assignment of Machine Learning

The bostan dataset collected by the U.S. Census Service concerning house of https://en.wikipedia.org/wiki/Data_set

crim:per capita crime rate

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

black: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

lstat: lower status of the population

medv:median value of owner in $1000's

In [1]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("D:\sandip sir 3rd sem lab\Boston.csv")
df.head()
```

Out[1]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	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
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [2]:
```
print(np.shape(df))
```

```
(506, 15)
```

In [3]:
```
df.drop(columns=['Unnamed: 0'],inplace=True)
```

In [4]:
```
#To know about the dataset
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  black    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 [5]:
```
df.describe()
```

Out[5]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

In [6]:
```
# For cheacking the missing value
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
black      0
lstat      0
medv       0
dtype: int64
```

In [7]:
```
print(np.shape(df))
```

```
(506, 14)
```

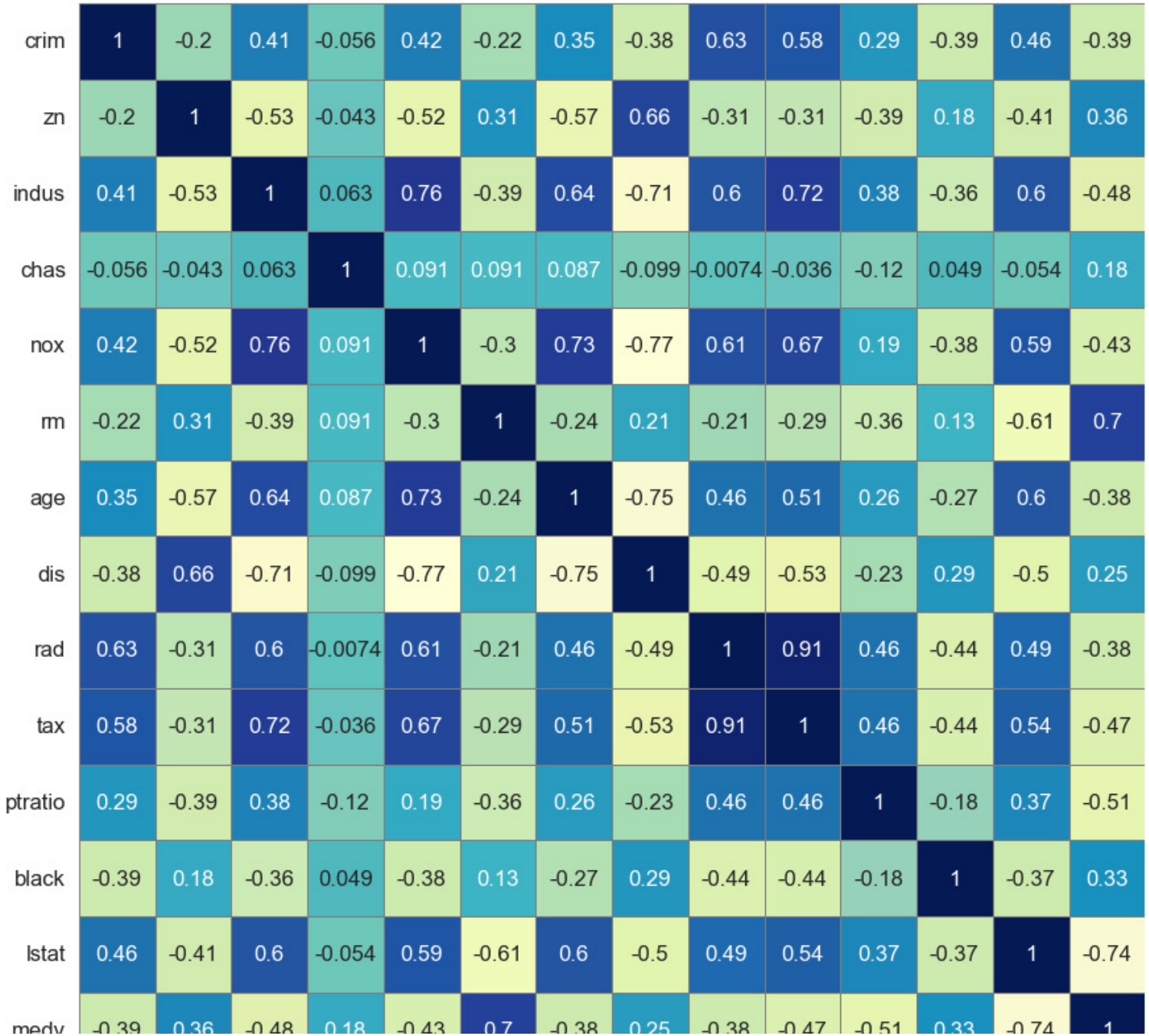
Data preprocessing

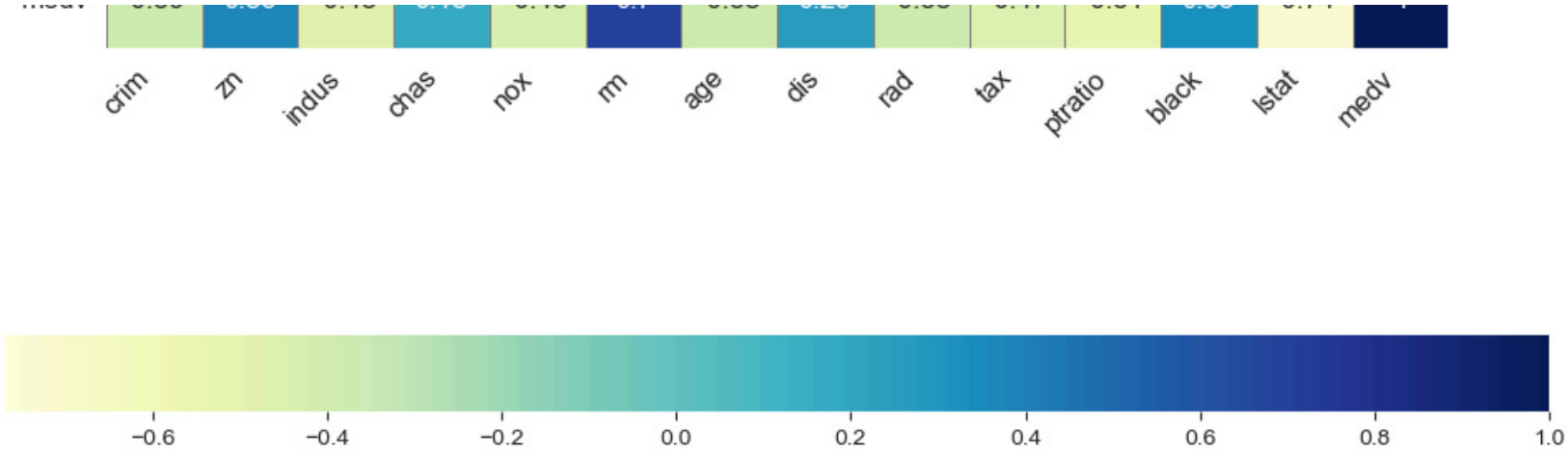
To know the feature variable how they are highly correlated

In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Customized heatmap with Seaborn
```

```
plt.figure(figsize=(15, 15))  # Increase figure size for better readability
sns.set_style("whitegrid")    # Set background style

# Heatmap with custom colors, font scaling, and rounded corners
sns.heatmap(
    df.corr(),                # Correlation matrix of the DataFrame
    annot=True,               # Show values on cells
    cmap="YlGnBu",            # Custom color map (Yellow-Green-Blue)
    annot_kws={"size": 12},   # Adjust font size for annotations
    linewidths=0.5,           # Add space between cells
    linecolor="gray",         # Color of cell borders
    cbar_kws={"shrink": 0.8, "orientation": "horizontal"},  # Custom color bar
    square=True               # Makes cells square-shaped
)

# Add title and axis labels
plt.title("Correlation Heatmap", fontsize=16, fontweight="bold", color="teal", pad=20)
plt.xticks(rotation=45, ha="right", fontsize=12)  # Rotate x labels for better readability
plt.yticks(rotation=0, fontsize=12)               # Rotate y labels for better readability

plt.show()
```

Correlation Heatmap

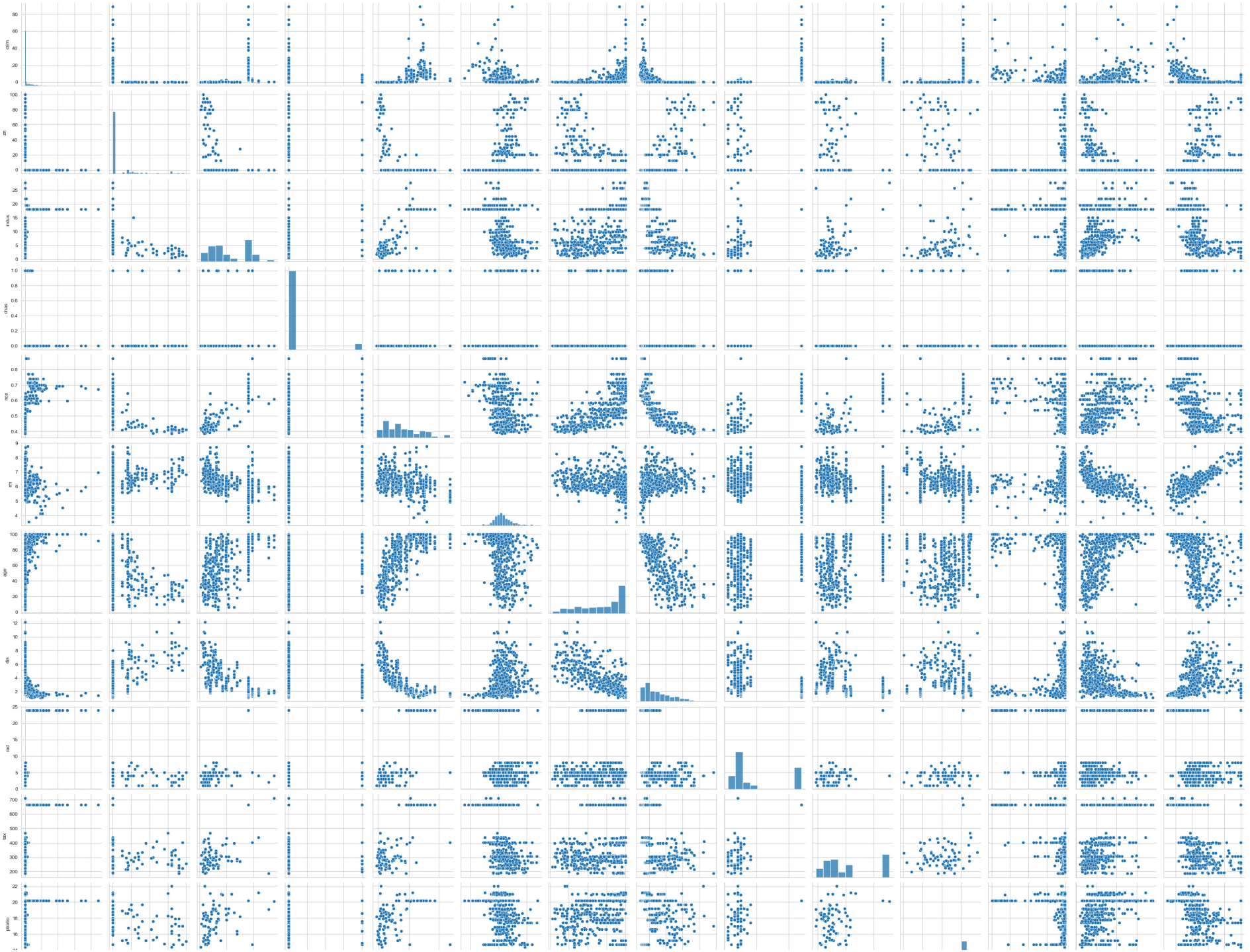


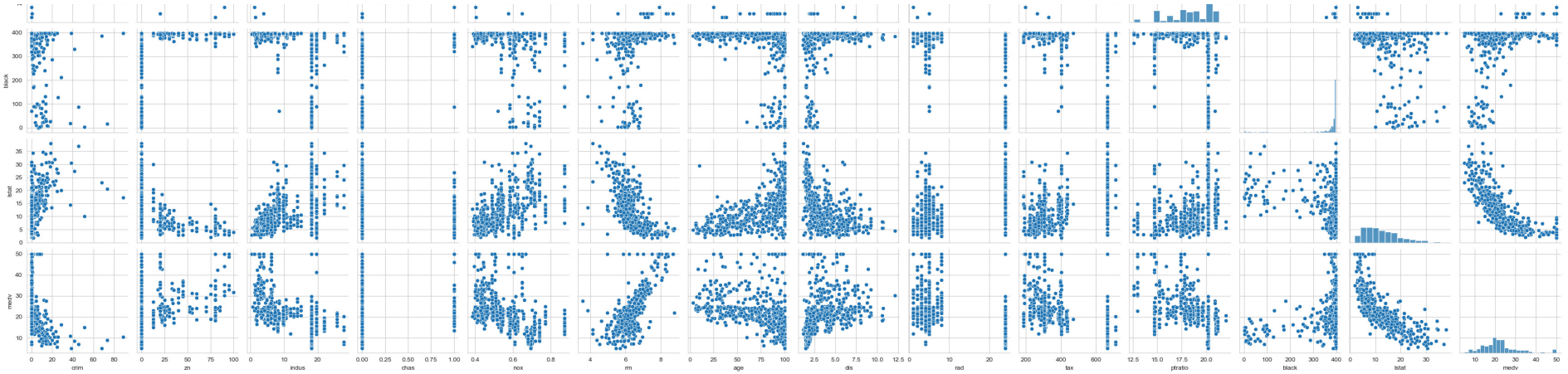


rad and tax are highly correlated.some other like above 50% correlated like zn and dis,indus and (lstat,tax,rad,age,nox) nox and lstat ,rm and medv are positive correlated.

In [9]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(df)
plt.show()
```

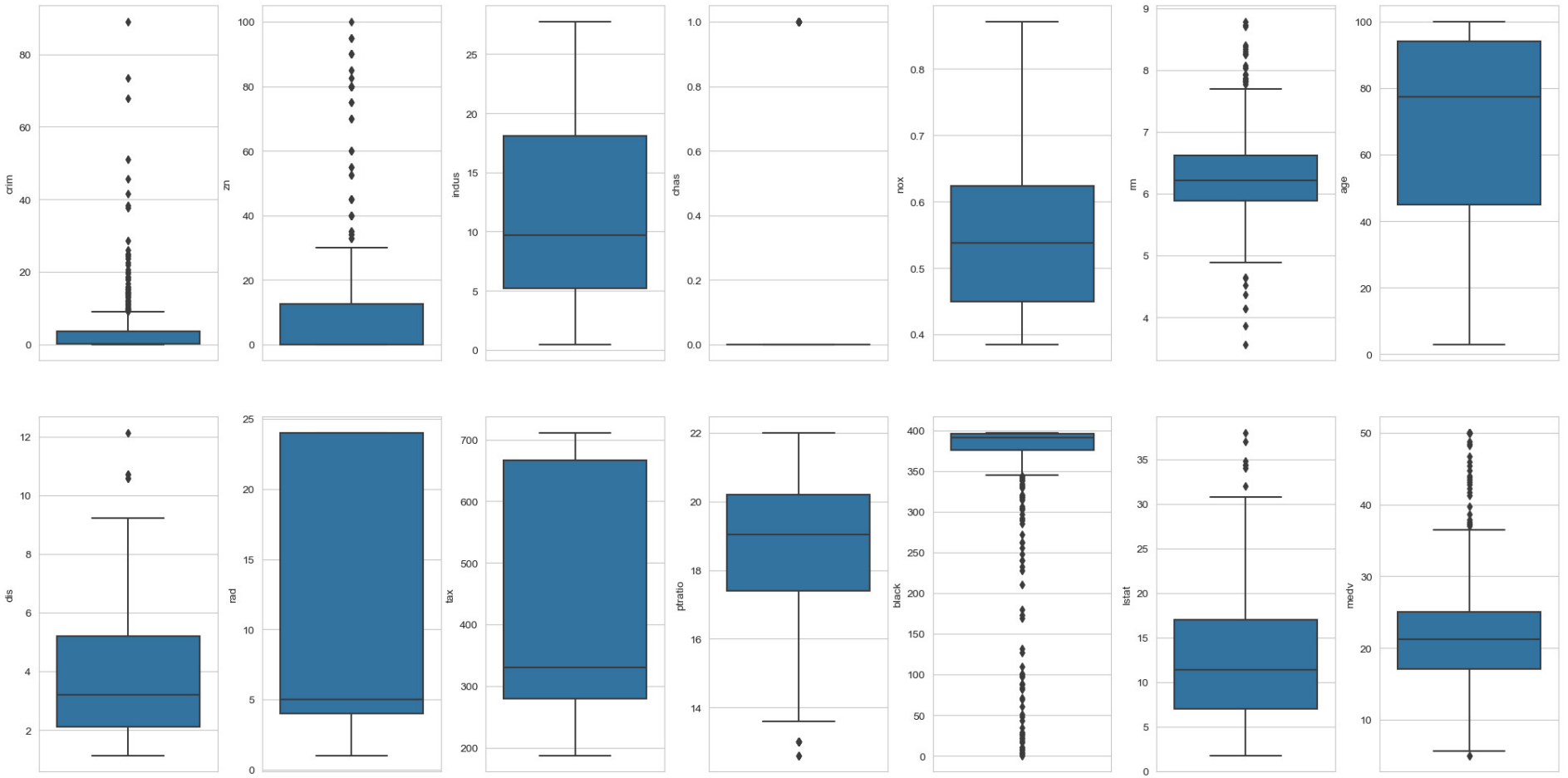
```
C:\Users\Admin\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```


crim
zn
indus
chas
nox
rm
age
dis
rad
tax
ptratio

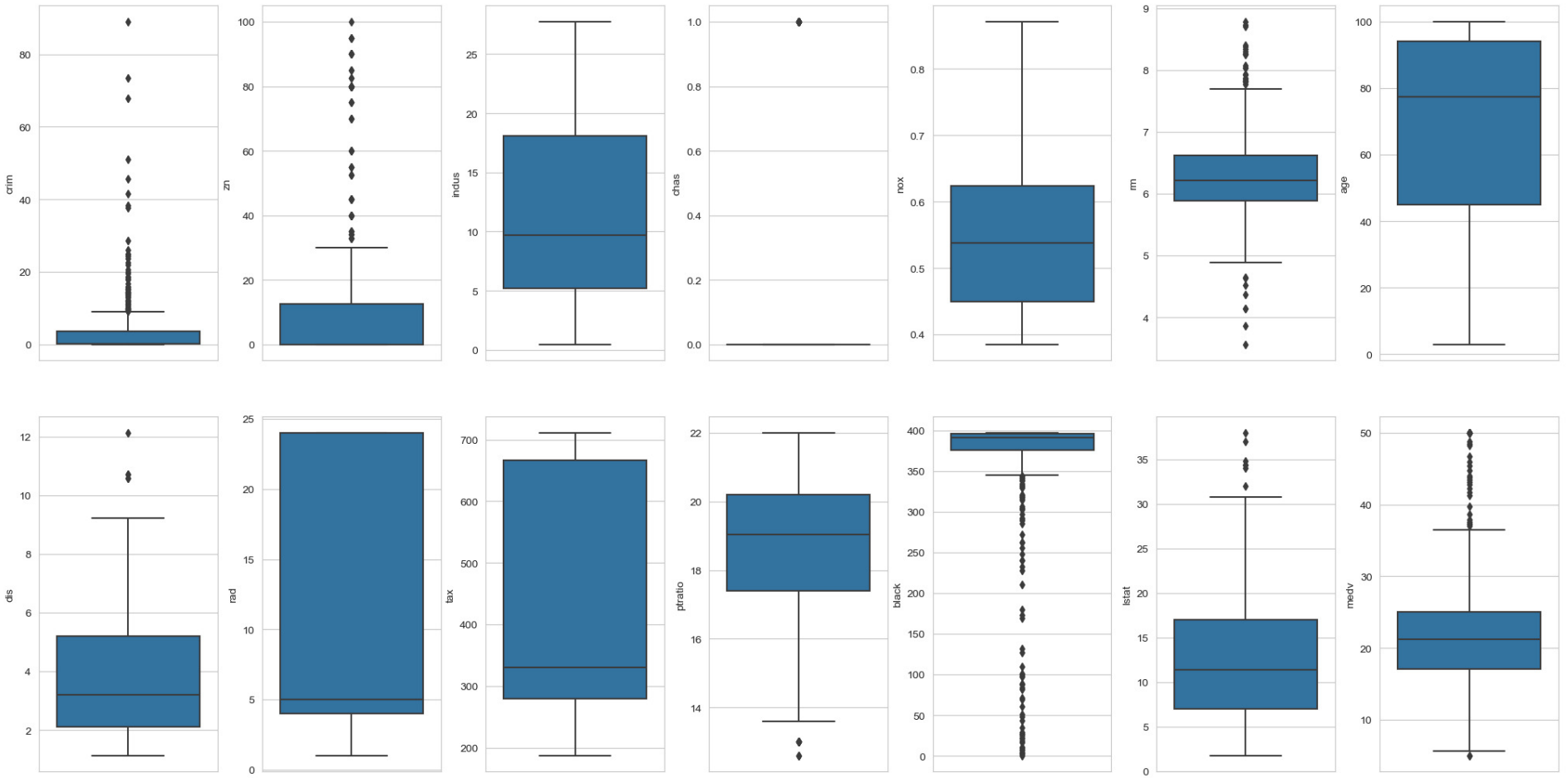


In [10]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in df.items():
    sns.boxplot(y=k, data=df, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



In [11]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in df.items():
    sns.boxplot(y=k, data=df, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



crim, zn,rm,dis,black have outlier,medv and lstat also.

In [12]:
```
# outiler Detetection and treatment it with IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [13]:
```
for k, v in df.items():
        q1 = v.quantile(0.25)
        q3 = v.quantile(0.75)
        irq = q3 - q1
        v_col = v[(v <= q1 - 1.5 * irq) | (v >= q3 + 1.5 * irq)]
        perc = np.shape(v_col)[0] * 100.0 / np.shape(df)[0]
        print("Column %s outliers = %.2f%%" % (k, perc))
```

```
Column crim outliers = 16.79%
Column zn outliers = 100.00%
Column indus outliers = 0.00%
Column chas outliers = 100.00%
Column nox outliers = 2.61%
Column rm outliers = 2.61%
Column age outliers = 0.00%
Column dis outliers = 0.00%
Column rad outliers = 15.67%
Column tax outliers = 14.18%
Column ptratio outliers = 0.00%
Column black outliers = 7.09%
Column lstat outliers = 3.36%
Column medv outliers = 5.22%
```

if above the 50% percent of total data that have impact is called outlier,so remove it.

In [14]:
```
df = df[~(df['medv'] >= 50.0)]
print(np.shape(df))
```

```
(268, 14)
```

Feature Engineering

In [15]:
```
# Feature Engineering: Log transformation of CRIM if needed
df['crim'] = np.log1p(df['crim'])
```

Encoding categorical variables:One- hot encoding for CHAS

In [16]:
```
# Encoding Categorical Variables: One-hot encoding for CHAS
df = pd.get_dummies(df, columns=['chas'], drop_first=True)
```

In [17]:
```
# Feature and Target Variable Separation
X = df.drop("medv", axis=1)  # Features
y = df["medv"]               # Target variable
```

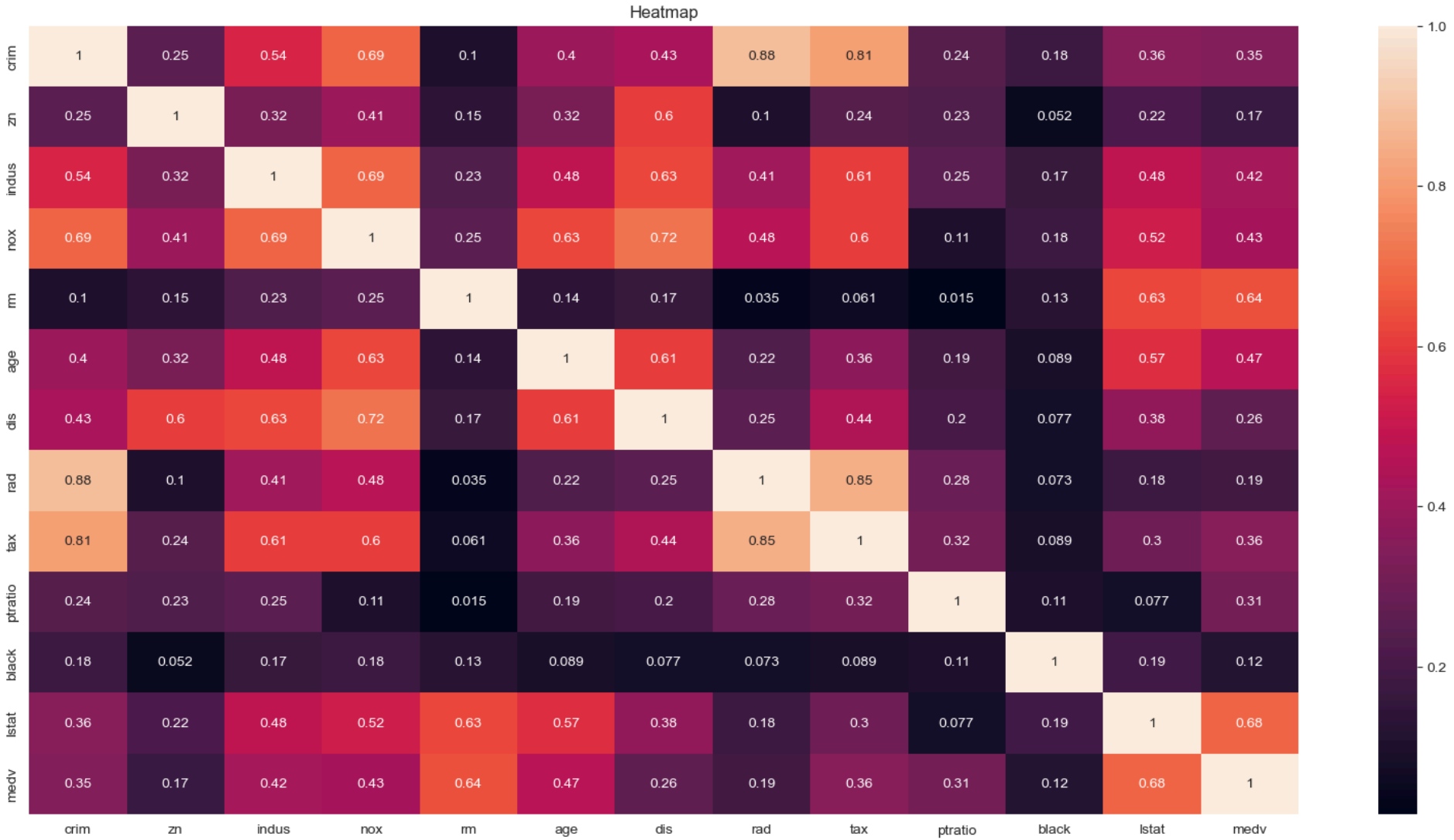
In [18]:
```
# To remove the skewness  of the bostan data.
y =  np.log1p(y)
for col in X.columns:
    if np.abs(X[col].skew()) > 0.3:
        X[col] = np.log1p(X[col])
        print(X[col].head())
```

```
0    0.006280
1    0.026587
2    0.026568
3    0.031360
4    0.064636
Name: crim, dtype: float64
0    2.944439
1    0.000000
2    0.000000
3    0.000000
4    0.000000
Name: zn, dtype: float64
0    1.196948
1    2.088153
2    2.088153
3    1.156881
4    1.156881
Name: indus, dtype: float64
0    0.430483
1    0.384582
2    0.384582
3    0.377066
4    0.377066
Name: nox, dtype: float64
0    4.192680
1    4.380776
2    4.128746
3    3.845883
4    4.010963
Name: age, dtype: float64
0    1.627278
1    1.786261
2    1.786261
3    1.954757
4    1.954757
Name: dis, dtype: float64
0    0.693147
1    1.098612
2    1.098612
3    1.386294
4    1.386294
Name: rad, dtype: float64
0    5.693732
1    5.493061
2    5.493061
3    5.407172
4    5.407172
Name: tax, dtype: float64
0    2.791165
1    2.933857
2    2.933857
```

```
3    2.980619
4    2.980619
Name: ptratio, dtype: float64
0    5.986201
1    5.986201
2    5.975919
3    5.980479
4    5.986201
Name: black, dtype: float64
0    1.788421
1    2.316488
2    1.615420
3    1.371181
4    1.845300
Name: lstat, dtype: float64
```

most of feature normal and bilnormal distributed and except chas as discreate variable.

In [19]:
```
plt.figure(figsize=(20, 10))
sns.heatmap(df.corr().abs(),  annot=True)
plt.title('Heatmap')
plt.show()
```



Linear regression

In [20]:
```
import numpy as np
import pandas as pd
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
```

In [21]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size =0.2,
                                                    random_state = 42)

print("xtrain shape : ", x_train.shape)
print("xtest shape  : ", x_test.shape)
print("ytrain shape : ", y_train.shape)
print("ytest shape  : ", y_test.shape)
```

```
xtrain shape :  (214, 12)
xtest shape  :  (54, 12)
ytrain shape :  (214,)
ytest shape  :  (54,)
```

In [22]:
```
from sklearn.preprocessing import StandardScaler
# Feature Scaling (Standardization)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

for easier handling scale array to Dataframe

In [23]:
```
# Convert scaled arrays back to DataFrame for easier handling
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)
```

In [24]:
```
# Display shapes of datasets and first few rows of scaled training set
print("Training set shape:", X_train_scaled.shape)
print("Test set shape:", X_test_scaled.shape)
print(X_train_scaled.head())
```

```
Training set shape: (214, 12)
Test set shape: (54, 12)
       crim        zn     indus       nox        rm       age       dis  \
0 -0.686867 -0.477190 -1.868057 -0.463348 -0.072538  0.140053 -0.773372
1 -0.403235  2.123006 -0.646316 -1.113862 -1.347218  0.331390  1.819096
2  1.856652 -0.477190  1.181516  0.567377  0.324765 -0.145537  0.157528
3  0.712220 -0.477190 -0.134098  0.087080 -0.566802 -1.040118  0.432503
4 -0.585984 -0.477190  0.485110  0.461835  1.464645  0.712889 -0.943516

        rad       tax   ptratio     black     lstat
0 -0.759261 -1.595809 -0.555718  0.704369 -0.304654
1  0.358772 -0.077564  0.208981  0.005551  1.204285
2  2.196657  1.913585  0.818350 -0.040803 -0.055467
3 -0.399335 -0.282224  1.241984 -0.202151 -1.083900
4 -1.877294 -0.614631  1.241984  0.395782 -1.116131
```

In [25]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
```

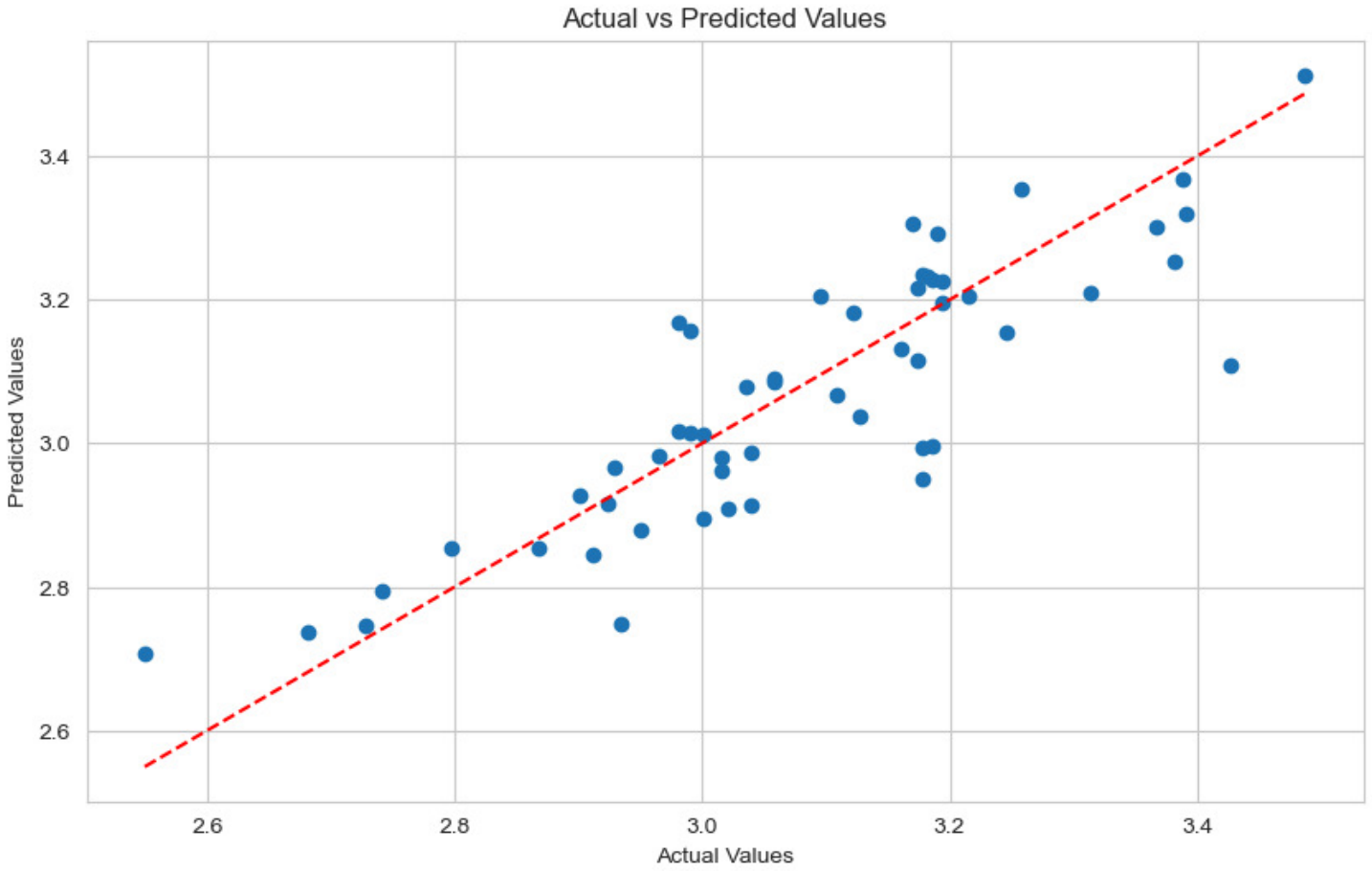
```
# Assuming X and y are already defined
model = LinearRegression()
model.fit(X_train_scaled, y_train)
y_pred=model.predict(X_test_scaled)
# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2=r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test,y_pred)
print("Mean Square Error : ", mse)
print("r2 score : ", r2)
print("Mean Absolute Error : ", mae)
```

```
Mean Square Error :  0.010081799562837217
r2 score :  0.72102308543735
Mean Absolute Error :  0.07722438914681241
```

Here, MSE approximately 0.01 suggest that model's predictions deviate from the actual values by a small amount.this is good depends on the scale of target variable('medv')

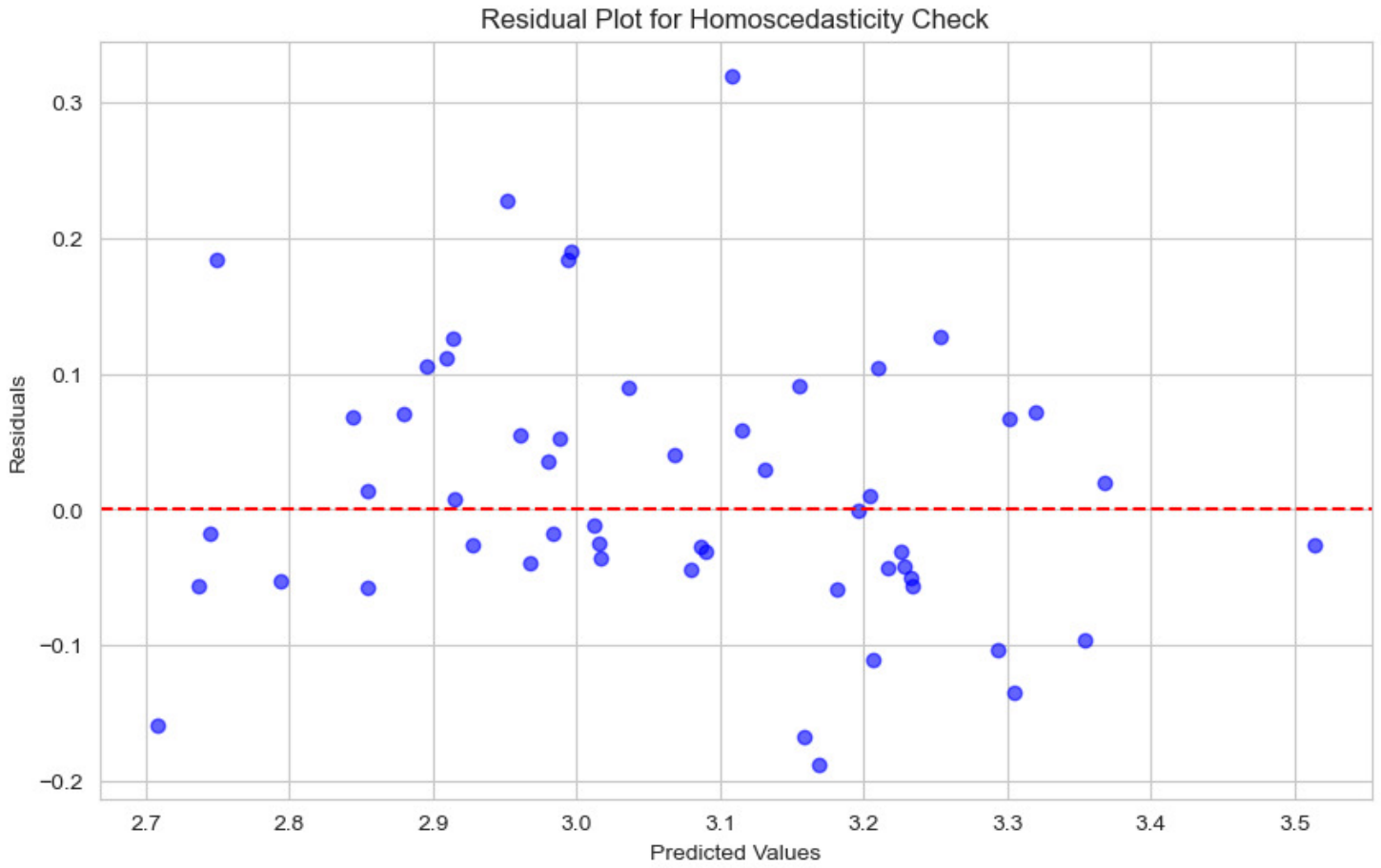
R2 approximately 0.72 means that 72% of the variance in housing prices can be explained by the features used in your model.there's still a significant portion (about 28%) of variance that is not explained, suggesting that other factors not included in your model might influence housing prices.

In [26]:
```
# Visualizing Actual vs Predicted Values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')  # Diagonal line
plt.show()
```



Homoscedasticity

In [27]:
```
import matplotlib.pyplot as plt
import seaborn as sns
residuals=y_test-y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color="blue", alpha=0.6)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot for Homoscedasticity Check")
plt.show()
```



To remove the heteroscedasticity is Detected

In [28]:
```
import numpy as np
#  Log transformation on y
y_log = np.log(y)
```

Weighted Least Square(WLS)

less weight to observations with larger variance to account for heteroscedasticity

In [29]:
```
import statsmodels.api as sm
```

```python
# # Weighted Least Squares (WLS)
weights = 1 / np.var(residuals)  # Inverse of variance for weights
wls_model = sm.WLS(y, X, weights=weights).fit()
print(wls_model.summary())
```

```
                            WLS Regression Results
==============================================================================
Dep. Variable:                   medv   R-squared (uncentered):                   0.998
Model:                            WLS   Adj. R-squared (uncentered):              0.998
Method:                 Least Squares   F-statistic:                          1.408e+04
Date:                Sun, 24 Nov 2024   Prob (F-statistic):                        0.00
Time:                        20:21:57   Log-Likelihood:                         187.15
No. Observations:                 268   AIC:                                    -350.3
Df Residuals:                     256   BIC:                                    -307.2
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
crim          -0.1284      0.052     -2.484      0.014      -0.230      -0.027
zn            -0.0158      0.008     -1.947      0.053      -0.032       0.000
indus         -0.0065      0.022     -0.295      0.768      -0.050       0.037
nox           -0.3033      0.274     -1.105      0.270      -0.844       0.237
rm             0.1803      0.025      7.188      0.000       0.131       0.230
age           -0.0469      0.018     -2.628      0.009      -0.082      -0.012
dis           -0.1156      0.043     -2.685      0.008      -0.200      -0.031
rad            0.1322      0.027      4.862      0.000       0.079       0.186
tax           -0.1944      0.043     -4.567      0.000      -0.278      -0.111
ptratio       -0.4981      0.093     -5.371      0.000      -0.681      -0.315
black          0.8911      0.070     12.807      0.000       0.754       1.028
lstat         -0.1633      0.032     -5.038      0.000      -0.227      -0.099
==============================================================================
Omnibus:                       34.203   Durbin-Watson:                   1.170
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              118.472
Skew:                          -0.464   Prob(JB):                     1.88e-26
Kurtosis:                       6.122   Cond. No.                         457.
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Omnibus (34.203): A test statistic for normality of residuals; a high value here suggests non-normality,but here is not.

Prob(Omnibus): Probability associated with the Omnibus test; a low value here indicates significant non-normality.

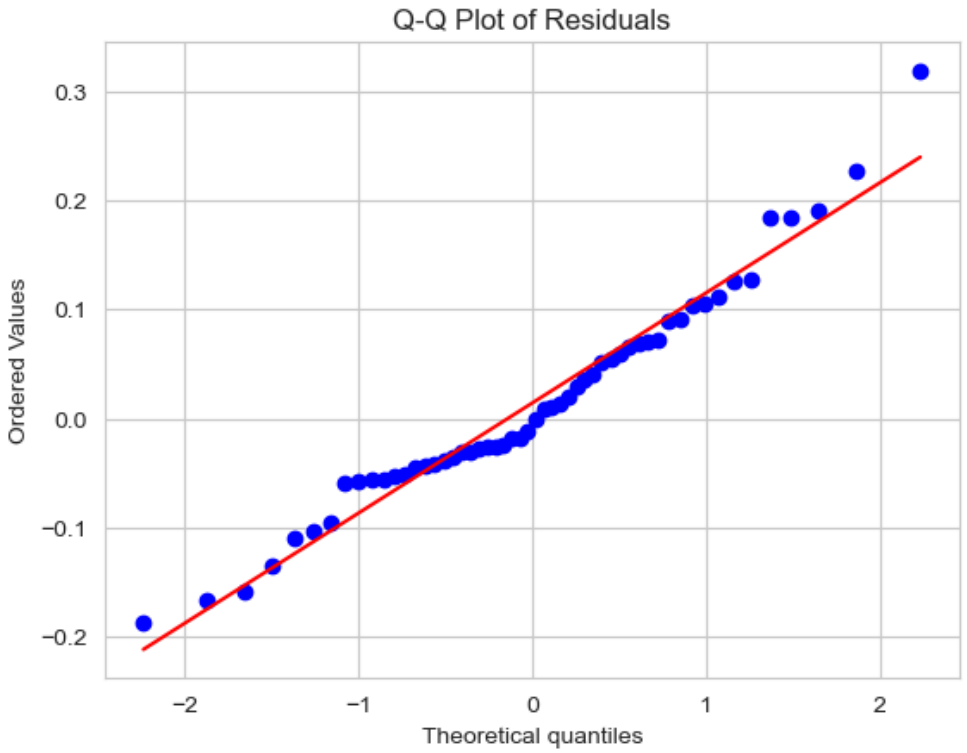
Jarque-Bera (118.472): Another test for normality, where a large value suggests the residuals are not normally distributed,but here is normaly distributed.

Skew (-0.464): Measures the asymmetry of the residual distribution. A value > 0 indicates positive skew.but here is negative skewed.

Kurtosis (6.122): Measures the "tailedness" of the residual distribution. A high kurtosis indicates more extreme values than a normal distribution, implying potential issues with outliers.but here is low value.

Normality of errors

In [30]:
```
import scipy.stats as stats
import matplotlib.pyplot as plt

# # Q-Q plot
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals")
plt.show()
```



In this plot no non-normality distributed.

Polynomial Regression

Assumptions of Polynomial Regression

Relationship of Regression:

In polynomial regression, the relationship can be non-linear, but it follows a polynomial form between the independent and target variables.

Independence of Errors:

The errors between data points should be independent.

Constant Variance of Errors (Homoscedasticity):

The variance of residuals (errors) should be constant across all levels of the independent variable.

Normality of Errors:

The errors (residuals) should follow a normal distribution, especially in cases of smaller sample sizes.

Avoid Overfitting:

Be cautious about the polynomial degree. Higher degrees can make the model overly complex and lead to overfitting.

Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y | x).

when the relationship between the independent variable (input) and the dependent variable (output) is not linear.

In [31]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [33]:
```
from sklearn.preprocessing import StandardScaler
# Feature Scaling (Standardization)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

In [34]:
```
# Convert scaled arrays back to DataFrame for easier handling
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
```

```
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)
```

In [35]:
```
poly = PolynomialFeatures(degree=2)
X_poly_train = poly.fit_transform(X_train_scaled)
X_poly_test = poly.transform(X_test_scaled)
```

In [36]:
```
model = LinearRegression()
model.fit(X_poly_train, y_train)
```

Out[36]:
```
▾ LinearRegression

LinearRegression()
```

In [37]:
```
y_pred_train = model.predict(X_poly_train)
y_pred = model.predict(X_poly_test)
print(y_pred)
print(y_test.head())
```

```
[3.01877393 3.01984438 2.74219833 3.08052632 2.9875288  3.22222928
 3.06036168 3.23174665 3.07992094 3.38912785 3.0231647  3.44773519
 3.26362095 3.025377   2.83750228 3.23352299 3.23539937 3.47557147
 3.11500115 2.76845598 2.95647121 3.02229435 3.0533946  3.21985372
 3.1622872  2.84774481 2.75980802 3.08972596 3.24096474 2.70838389
 3.27473463 3.26059878 3.12162926 3.05178236 3.56072972 3.293317
 2.78519654 3.06517008 3.05277315 3.08982714 2.59823075 2.97767919
 2.98475524 2.98071994 3.23530851 3.0704652  3.39928684 3.05890465
 3.04483715 3.21365975 3.08111658 2.98465688 3.18666548 3.02149757]
132    3.178054
340    2.980619
23     2.740840
171    3.000720
135    2.949688
Name: medv, dtype: float64
```

In [38]:
```
# On Training Data
train_mse = mean_squared_error(y_train, y_pred_train)
train_r2 = r2_score(y_train, y_pred_train)

# On Testing Data
test_mse = mean_squared_error(y_test, y_pred)
test_r2 = r2_score(y_test, y_pred)

print("Train MSE:", train_mse)
print("Train R2 Score:", train_r2)
print("Test MSE:", test_mse)
print("Test R2 Score:", test_r2)
```

```
Train MSE: 0.004509351596183737
Train R2 Score: 0.9044010688892903
Test MSE: 0.009942207952517946
Test R2 Score: 0.7248857725006052
```

Conclusion

A higher test MSE compared to training MSE can be a sign of overfitting, where the model learns noise and details from the training data that do not apply to new data.

score is still relatively good, it is significantly lower than the training R^2 score of 90%. This discrepancy suggests that your model may not be generalizing well to new data.

Logistic Regression

Independent observations:

Each observation is independent of the other. meaning there is no correlation between any input variables.

Binary dependent variables:

It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories SoftMax functions are used.

Linearity relationship between independent variables and log odds:

The relationship between the independent variables and the log odds of the dependent variable should be linear.

No outliers:

There should be no outliers in the dataset.

Large sample size:

The sample size is sufficiently large

In [39]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

In [40]:
```
# Set a threshold, e.g., median or any value you find suitable
threshold = df['medv'].median()

# Binning the target variable
df['medv_category'] = (df['medv'] > threshold).astype(int)  # 1 for high, 0 for low

# Check that the target variable is now binary
print("Unique values in medv_category:", df['medv_category'].unique())  # Should print [0, 1]
```

```
Unique values in medv_category: [1 0]
```

In [41]:
```
X = df.drop(['medv', 'medv_category'], axis=1)  # Drop original and new target from features
y = df['medv_category']
```

In [42]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [43]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Assuming X_train, X_test, y_train, y_test are already defined

# Fit logistic regression
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)

# Predict probabilities for the positive class (e.g., class 1)
y_prob = model.predict_proba(X_test)
```

In [44]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,precision_score,recall_score,f1_score
# Assuming y_pred contains continuous predictions (e.g., probabilities)
y_pred_binary = (y_pred >= 0.5).astype(int)  # Convert probabilities to binary labels
# # Calculate accuracy and other metrics
accuracy = accuracy_score(y_test, y_pred_binary)
conf_matrix = confusion_matrix(y_test, y_pred_binary)
class_report = classification_report(y_test, y_pred_binary)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.48148148148148145
Confusion Matrix:
 [[ 0 28]
 [ 0 26]]
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

```
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Admin\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [45]:

```
# Calculate MSE and RMSE on probabilities
mse = mean_squared_error(y_test, y_pred_binary)
rmse = np.sqrt(mse)
print("MSE:", mse)
print("RMSE:", rmse)

# R² can be calculated but may not be meaningful here
r2 = r2_score(y_test, y_pred_binary)
print("R²:", r2)
```

```
MSE: 0.5185185185185185
RMSE: 0.7200822998230956
R²: -1.0769230769230766
```

In [46]:

```
import numpy as np
import pandas as pd
df=pd.read_csv("D:\sandip sir 3rd sem lab\Boston.csv")
df.head()
```

Out[46]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	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
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [47]:
```
df.drop(columns=['Unnamed: 0'],inplace=True)
df
```

Out[47]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
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
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [48]:
```
X=df.iloc[:,0:13]
print(X)
y=df['medv']
print(y)
```

```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio   black  lstat
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: medv, Length: 506, dtype: float64
```

In [49]:

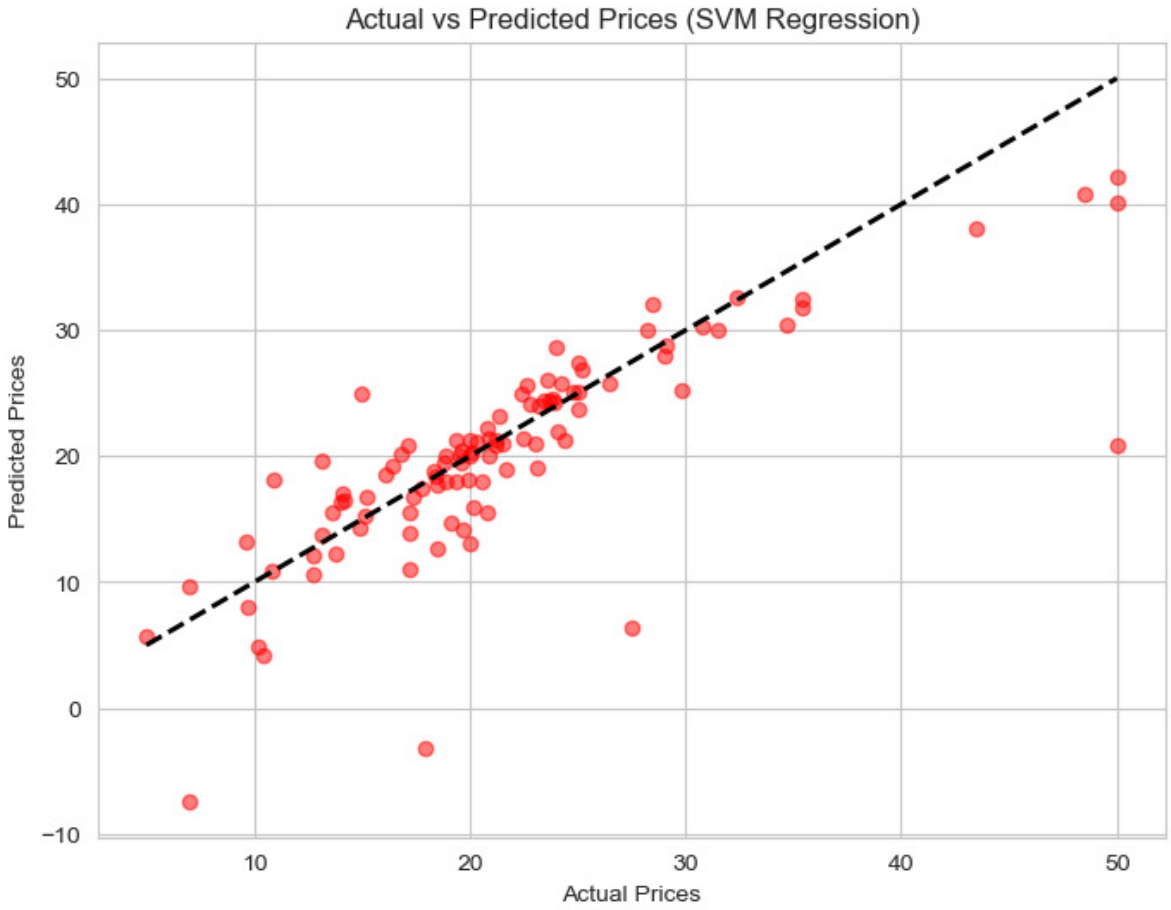
```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import numpy as np

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SVM regression model
svm_regressor = SVR(kernel='linear')  # You can try different kernels like 'linear', 'rbf', etc.
```

```
svm_regressor.fit(X_train, y_train)


# Initialize DecisionTreeRegressor Model
dt_regressor = DecisionTreeRegressor(random_state=42)

# Fit the model
dt_regressor.fit(X_train, y_train)
# Predict on the test data
y_pred = svm_regressor.predict(X_test)

# Calculate MSE, RMSE, and R²
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Print the results
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 29.435701924289845
Root Mean Squared Error (RMSE): 5.4254678991115455
R-squared (R²): 0.5986065268181071
```

In [50]:

```
import matplotlib.pyplot as plt
from sklearn.svm import SVR
import numpy as np
# Apply SVM regression model
svm_regressor = SVR(kernel='linear')
svm_regressor.fit(X_train, y_train)

# Predict on the test data
y_pred = svm_regressor.predict(X_test)

# Scatter plot for Actual vs Predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='red', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices (SVM Regression)")
plt.show()
```



Decision Tree

Interpretability:

Decision trees are easy to understand and interpret. They visually represent decisions in a flowchart-like structure, making it simple to see how decisions are made.

Non-parametric:

They do not assume any underlying distribution of the data, which makes them flexible for various types of data.

Handling Missing Values:

Decision trees can handle missing values and can work with both categorical and continuous data.

Feature Importance:

It provide insights into feature importance, helping to identify which variables are most influential in making predictions.

In [51]:
```
# Required Libraries

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error,r2_score
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize DecisionTreeRegressor Model
dt_regressor = DecisionTreeRegressor(random_state=42)

# Fit the model
dt_regressor.fit(X_train, y_train)

# Make Predictions
y_pred = dt_regressor.predict(X_test)

# Evaluate Model Performance
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Score:{rmse}")
print(f"r2 value: {r2}")
```

```
Mean Squared Error: 10.416078431372549
Root Mean Score:3.2273949915330395
r2 value: 0.8579634380978161
```

Random forest

Ensemble Method:

Random Forest is an ensemble of multiple decision trees, which improves accuracy by reducing overfitting that can occur with single decision trees.

Robustness:

It is robust against noise and overfitting due to its averaging mechanism, which helps in generalizing better on unseen data.

Versatility:

Random Forest can be used for both classification and regression tasks effectively.

Feature Selection:

It can automatically handle feature selection and provide estimates of feature importance.

In [52]:
```
from sklearn.ensemble import RandomForestRegressor
# Initialize and Train Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)
rf_regressor.fit(X_train, y_train)

# Predictions
y_pred = rf_regressor.predict(X_test)

# Evaluation Metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f"Random Forest Regression - MSE: {mse}, RMSE: {rmse}, R-squared: {r2}")
```

```
Random Forest Regression - MSE: 7.901513892156864, RMSE: 2.8109631609391226, R-squared: 0.8922527442109116
```

K-Nearest Neighbors(KNN)

Simplicity:

KNN is simple to implement and understand. It classifies a data point based on how its neighbors are classified.

No Training Phase:

KNN does not require a training phase; it simply stores the training dataset and makes predictions based on the distance to the nearest neighbors.

Adaptability:

It can be used for both classification and regression tasks and works well with small datasets.

Non-parametric:

Like decision trees, KNN does not assume any specific distribution for the data.

In [53]:
```
from sklearn.neighbors import KNeighborsRegressor,KNeighborsClassifier
# Initialize KNeighborsRegressor with k=5 (you can tune k as needed)
knn_regressor = KNeighborsRegressor(n_neighbors=5)

# Fit the model
knn_regressor.fit(X_train, y_train)

# Predictions
y_pred_regressor = knn_regressor.predict(X_test)

# Evaluation Metrics
mse = mean_squared_error(y_test, y_pred_regressor)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_regressor)

print("KNN Regression Results:")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²): {r2}")
```

```
KNN Regression Results:
Mean Squared Error (MSE): 25.860125490196076
Root Mean Squared Error (RMSE): 5.0852851926117255
R-squared (R²): 0.6473640882039258
```

Regression type data in term of Classification

In [54]:
```
import numpy as np
import pandas as pd
df=pd.read_csv("D:\sandip sir 3rd sem lab\Boston.csv")
df.head()
```

Out[54]:

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	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
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [55]:
```
df.drop(columns=['Unnamed: 0'],inplace=True)
```

In [56]:
```
X=df.iloc[:,0:13]
print(X)
```

```
y=df['medv']
print(y)
```

```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio   black  lstat
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: medv, Length: 506, dtype: float64
```

In [57]:
```
# Calculate the median value of MEDV
median_value = df['medv'].median()
print(median_value)
# Create a new binary target column
df['MEDV_Class'] = np.where(df['medv'] >= median_value, 1, 0)
print(df[['medv', 'MEDV_Class']].head())
```

```
21.2
   medv  MEDV_Class
0  24.0           1
1  21.6           1
2  34.7           1
3  33.4           1
4  36.2           1
```

In [58]:

```
X=df.iloc[:,0:13]
y=df['MEDV_Class']
print(X)
print(y)
```

```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio   black  lstat
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
0      1
1      1
2      1
3      1
4      1
      ..
501    1
502    0
503    1
504    1
505    0
Name: MEDV_Class, Length: 506, dtype: int32
```

In [59]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [60]:
```
from sklearn.linear_model import LinearRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report
```

```
# Initialize models
linear_model = LinearRegression()  # Linear regression is used here to predict probability and threshold
naive_bayes_model = GaussianNB()
decision_tree_model = DecisionTreeClassifier()
random_forest_model = RandomForestClassifier()
knn_model = KNeighborsClassifier()
```

Naive Bayes

Naive Bayes classification

It is highly used in text classification. It is used in spam filtering, sentiment detection, rating classification etc. It is fast and making prediction is easy with high dimension of data. The “Bayes” part of the name refers to Reverend Thomas Bayes, an 18th-century statistician and theologian who formulated Bayes’ theorem.

Assumption of Naive Bayes

The fundamental Naive Bayes assumption is that each feature makes an:

Feature independence:

The features of the data are conditionally independent of each other, given the class label.

Continuous features are normally distributed:

If a feature is continuous, then it is assumed to be normally distributed within each class.

Discrete features have multinomial distributions:

If a feature is discrete, then it is assumed to have a multinomial distribution within each class.

Features are equally important:

All features are assumed to contribute equally to the prediction of the class label.

No missing data:

The data should not contain any missing values.

With relation to our dataset, this concept can be understood as:

We assume that no pair of features are dependent. For example, the temperature being 'Hot' has nothing to do with the humidity or the outlook being 'Rainy' has no effect on the winds. Hence, the features are assumed to be independent.

Secondly, each feature is given the same weight(or importance). For example, knowing only temperature and humidity alone can't predict the outcome accurately. None of the attributes is irrelevant and assumed to be contributing equally to the outcome.

In [61]:
```
# Train Naive Bayes
naive_bayes_model.fit(X_train, y_train)
naive_bayes_pred = naive_bayes_model.predict(X_test)
naive_bayes_accuracy = accuracy_score(y_test, naive_bayes_pred)
print(f"Naive Bayes Accuracy: {naive_bayes_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Naive Bayes Accuracy: 0.7549019607843137
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

Decision Tree

In [62]:
```
# Train Decision Tree
decision_tree_model.fit(X_train, y_train)
decision_tree_pred = decision_tree_model.predict(X_test)
decision_tree_accuracy = accuracy_score(y_test, decision_tree_pred)
print(f"Decision Tree Accuracy: {decision_tree_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Decision Tree Accuracy: 0.8529411764705882
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

Random Forest

In [63]:
```
# Train Random Forest
random_forest_model.fit(X_train, y_train)
random_forest_pred = random_forest_model.predict(X_test)
random_forest_accuracy = accuracy_score(y_test, random_forest_pred)
print(f"Random Forest Accuracy: {random_forest_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Random Forest Accuracy: 0.8725490196078431
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

K- Nearest Neighbors(KNN)

In [64]:
```
# Train KNN
knn_model.fit(X_train, y_train)
knn_pred = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_pred)
```

```
print(f"KNN Accuracy: {knn_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
KNN Accuracy: 0.8333333333333334
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

SVM Classification on Boston Dataset

Effective in High Dimensions:

SVM is effective in high-dimensional spaces, it makes it suitable for text classification and other applications with many features.

Margin Maximization:

SVM aims to find the hyperplane that maximizes the margin between different classes, leading to better generalization on unseen data.

Kernel Trick:

SVM can use kernel functions to transform data into higher dimensions, allowing it to model complex relationships that are not linearly separable.

Robustness to Overfitting:

With proper regularization, SVM is robust against overfitting, especially in high-dimensional spaces.

Support Vector Machine(SVM)

A Support Vector Machine (SVM) is a powerful machine learning algorithm widely used for both linear and nonlinear classification, as well as regression and outlier detection tasks. SVMs are highly adaptable, making them suitable for various applications such as text classification, image classification, spam detection, handwriting identification,

gene expression analysis, face detection, and anomaly detection.

Advantages and Disadvantages of Support Vector Machine (SVM)

Advantages of Support Vector Machine (SVM)

High-Dimensional Performance:

SVM excels in high-dimensional spaces, making it suitable for image classification and gene expression analysis.

Nonlinear Capability:

Utilizing kernel functions like RBF and polynomial, SVM effectively handles nonlinear relationships.

Outlier Resilience:

The soft margin feature allows SVM to ignore outliers, enhancing robustness in spam detection and anomaly detection.

Binary and Multiclass Support:

SVM is effective for both binary classification and multiclass classification, suitable for applications in text classification.

Memory Efficiency:

SVM focuses on support vectors, making it memory efficient compared to other algorithms. Disadvantages of Support Vector Machine (SVM)

Slow Training:

SVM can be slow for large datasets, affecting performance in SVM in data mining tasks.

Parameter Tuning Difficulty:

Selecting the right kernel and adjusting parameters like C requires careful tuning, impacting SVM algorithms.

Noise Sensitivity:

SVM struggles with noisy datasets and overlapping classes, limiting effectiveness in real-world scenarios.

Limited Interpretability:

The complexity of the hyperplane in higher dimensions makes SVM less interpretable than other models.

Feature Scaling Sensitivity:

Proper feature scaling is essential; otherwise, SVM models may perform poorly.

In [65]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize and train SVM model
svm_model = SVC(kernel='linear')  # You can also try 'rbf', 'poly', or other kernels
svm_model.fit(X_train, y_train)

# Predict on test data
svm_pred = svm_model.predict(X_test)

# Evaluate model
svm_accuracy = accuracy_score(y_test, svm_pred)
print(f"SVM Accuracy: {svm_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
SVM Accuracy: 0.8921568627450981
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

In [66]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize and train SVM model
svm_model = SVC(kernel='rbf')  # You can also try 'rbf', 'poly', or other kernels
svm_model.fit(X_train, y_train)

# Predict on test data
svm_pred = svm_model.predict(X_test)
```

```
# Evaluate model
svm_accuracy = accuracy_score(y_test, svm_pred)
print(f"SVM Accuracy: {svm_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
SVM Accuracy: 0.6862745098039216
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

In [67]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize and train SVM model
svm_model = SVC(kernel='poly')  # You can also try 'rbf', 'poly', or other kernels
svm_model.fit(X_train, y_train)

# Predict on test data
svm_pred = svm_model.predict(X_test)

# Evaluate model
svm_accuracy = accuracy_score(y_test, svm_pred)
print(f"SVM Accuracy: {svm_accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
SVM Accuracy: 0.6666666666666666
Confusion Matrix:
[[ 0 28]
 [ 0 26]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        28
           1       0.48      1.00      0.65        26

    accuracy                           0.48        54
   macro avg       0.24      0.50      0.33        54
weighted avg       0.23      0.48      0.31        54
```

Decision Trees and Random Forests are great for interpretability and handling complex datasets with mixed types. KNN is useful for its simplicity and effectiveness in smaller datasets. SVM excels in high-dimensional spaces with complex boundaries. The choice of algorithm depends on the specific characteristics of Boston dataset

Logistics Accuracy Accuracy: 0.48148148148148145

Naive Bayes Accuracy: 0.7549019607843137

Decision Tree Accuracy: 0.8431372549019608

Random Forest Accuracy: 0.8725490196078431

KNN Accuracy: 0.8333333333333334

SVM Accuracy: 0.8921568627450981 (linear)

SVM Accuracy: 0.6862745098039216 (rbf)

SVM Accuracy: 0.6666666666666666 (poly)

Higher accuracy indicate model explain more about the data here is SVM is high,then Random forest.