Linear Regression and its Applications

Objective:

The goal of this assignment is to help you understand and implement Linear Regression using Python. You will analyze a dataset, perform data preprocessing, train a regression model, and evaluate its performance.

In [18]:
```
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

Step 1: Data Loading and Exploration

In [20]:
```
# Import libraries
import pandas as pd
import numpy as np

# Load the dataset
url = "https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv"
df = pd.read_csv(url)

# Explore the dataset
print("First 5 rows of the dataset:")
print(df.head())

# Check for missing values
print("\nMissing values:")
print(df.isnull().sum())

# Describe the dataset
print("\nDataset description:")
print(df.describe())
```

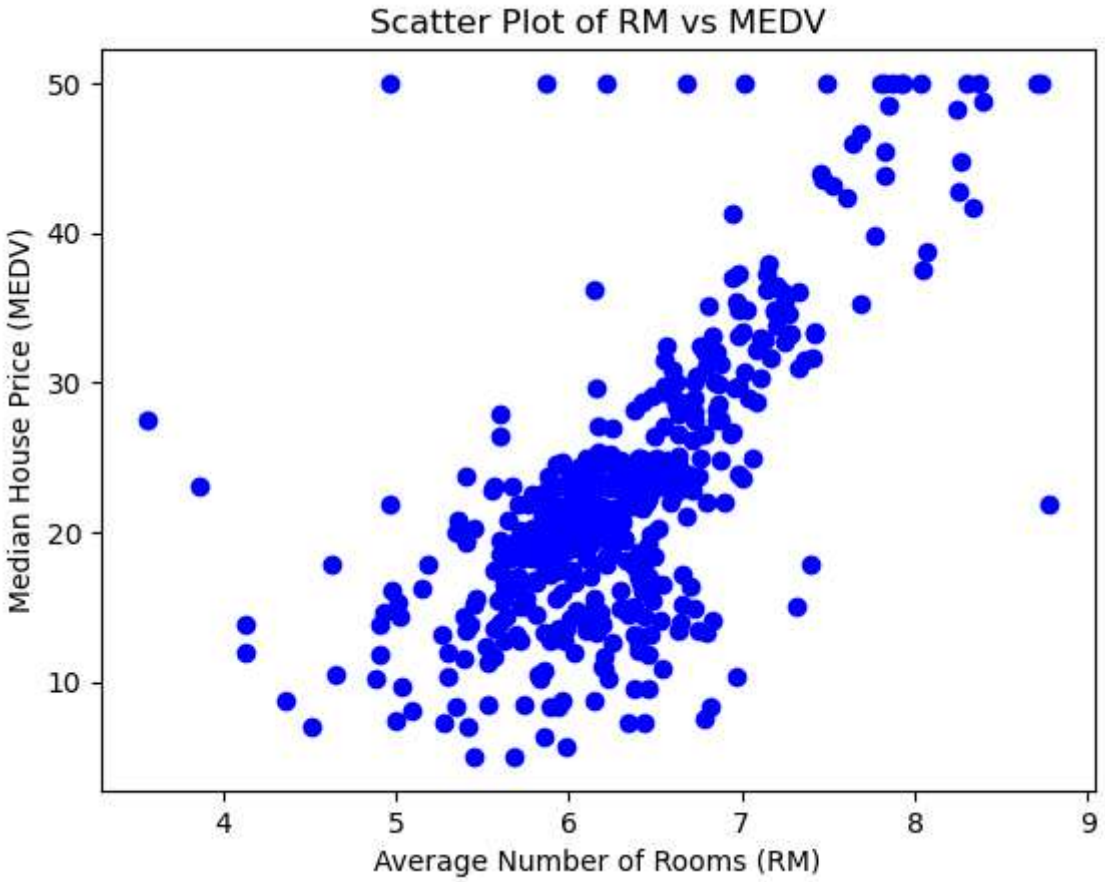
```
First 5 rows of the dataset:
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2

Missing values:
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

Dataset description:
             crim          zn       indus        chas         nox          rm  \
count  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
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000
```

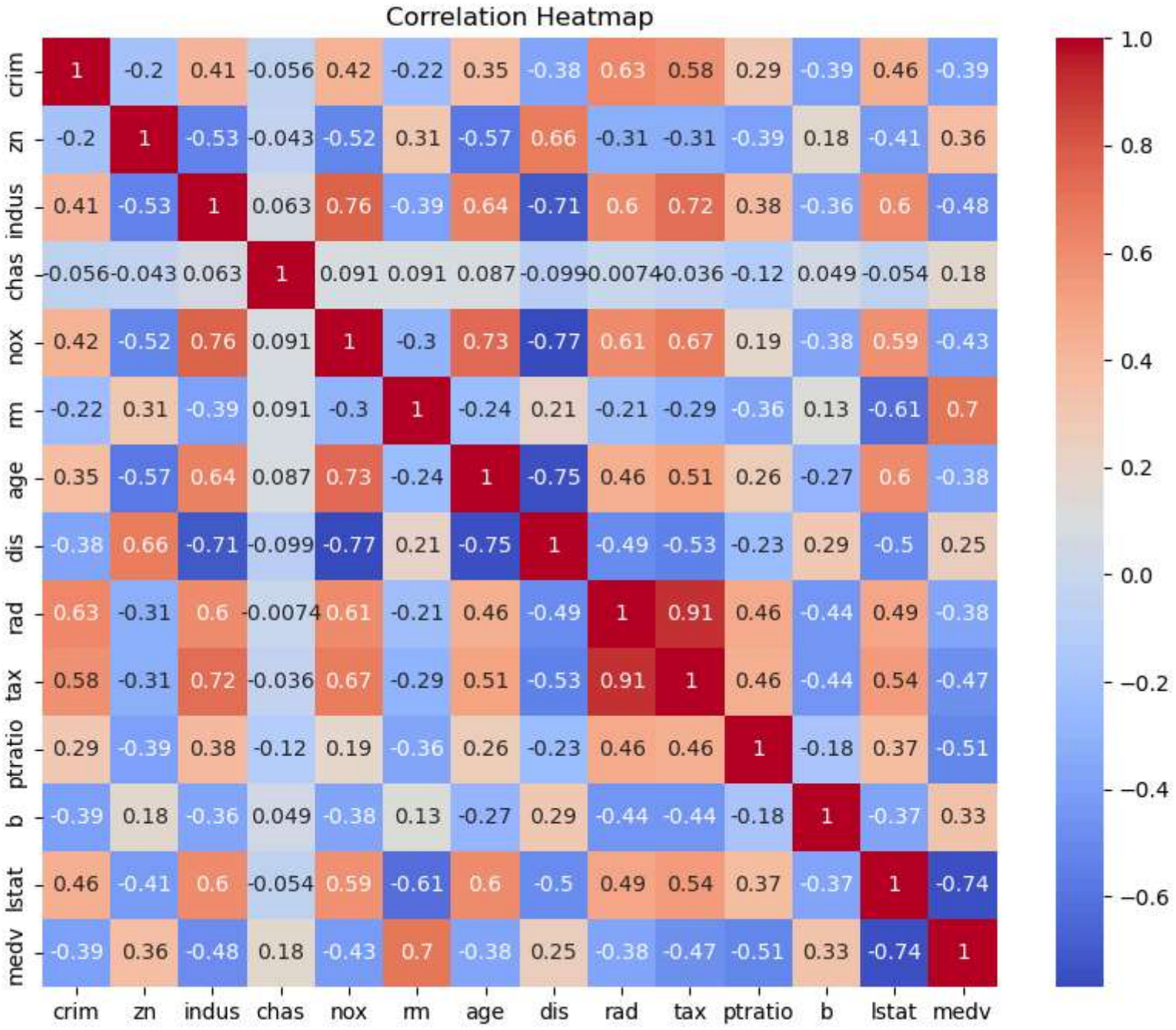
```
              age         dis         rad         tax     ptratio           b  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            lstat        medv
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Step 2: Data Visualization

In [22]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Scatter plot
plt.scatter(df['rm'], df['medv'], color='blue')
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median House Price (MEDV)")
plt.title("Scatter Plot of RM vs MEDV")
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```


Scatter Plot of RM vs MEDV
Median House Price (MEDV)
Average Number of Rooms (RM)
50
40
30
20
10
4
5
6
7
8
9



Step 3: Data Preprocessing

In [24]:
```
from sklearn.model_selection import train_test_split

# Independent and dependent variables
X = df[['rm']]
y = df['medv']

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

Step 4: Simple Linear Regression

In [26]:
```
from sklearn.linear_model import LinearRegression

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Display the intercept and coefficient
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

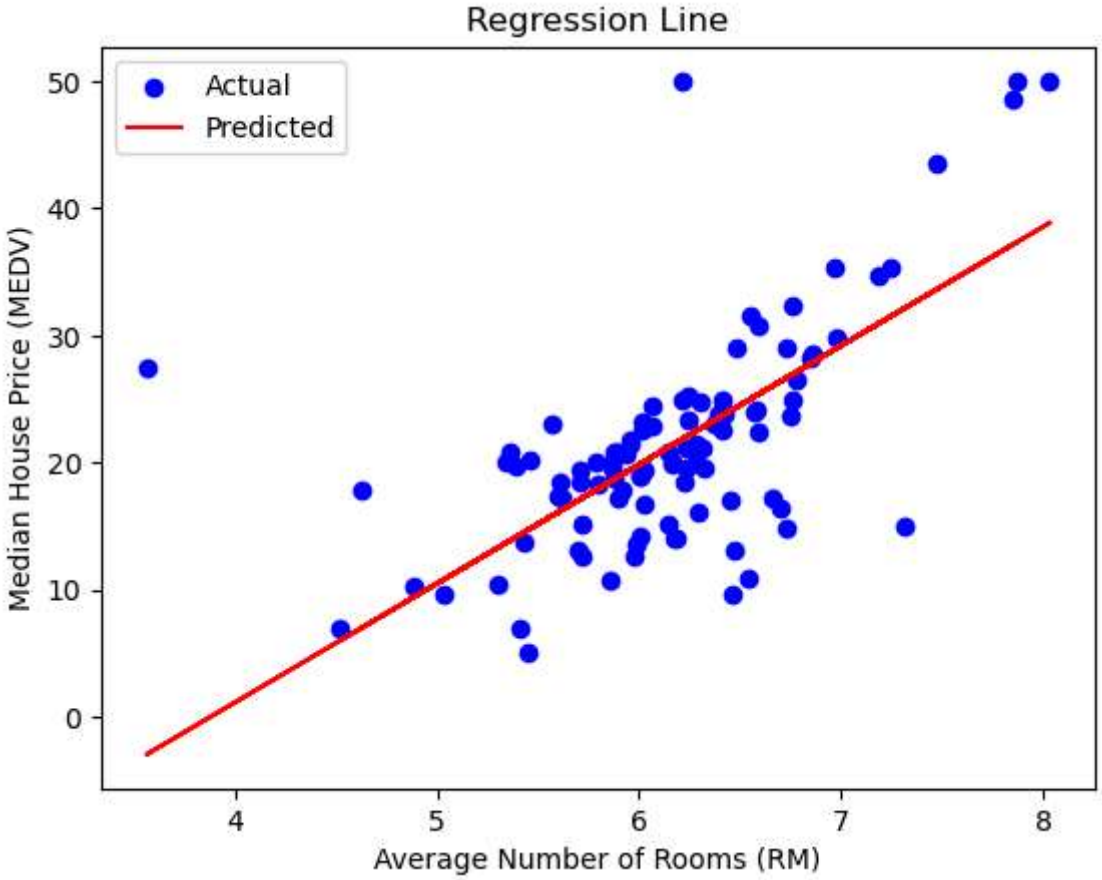
```
Intercept: -36.24631889813795
Coefficient: [9.34830141]
```

Step 5: Evaluate the Model

In [28]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Predictions
y_pred = model.predict(X_test)

# Evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared Score:", r2)

# Visualize regression line
plt.scatter(X_test, y_test, color='blue', label='Actual')
```

```
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median House Price (MEDV)")
plt.title("Regression Line")
plt.legend()
plt.show()
```

```
Mean Squared Error: 46.144775347317264
R-squared Score: 0.3707569232254778
```



Step 6: Multiple Linear Regression

In [30]:
```
# Use all features except MEDV
X_multi = df.drop('medv', axis=1)
y_multi = df['medv']
```

```
# Split the dataset
X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_split(X_multi, y_multi, test_size=0.2, random_s

# Train the model
multi_model = LinearRegression()
multi_model.fit(X_train_multi, y_train_multi)

# Predictions
y_pred_multi = multi_model.predict(X_test_multi)

# Evaluation metrics
mse_multi = mean_squared_error(y_test_multi, y_pred_multi)
r2_multi = r2_score(y_test_multi, y_pred_multi)

print("Multiple Linear Regression - Mean Squared Error:", mse_multi)
print("Multiple Linear Regression - R-squared Score:", r2_multi)
```

```
Multiple Linear Regression - Mean Squared Error: 24.291119474973613
Multiple Linear Regression - R-squared Score: 0.6687594935356307
```

In [ ]: