In [1]:
```python
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
```

In [3]:
```python
# Load the dataset
file_path = 'BostonHousing.csv'
data = pd.read_csv(file_path)
```

In [5]:
```python
# Preview dataset
print("Initial Dataset:")
print(data.info())
print(data.head())
```

```
Initial Dataset:
<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
None
      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  
```

In [7]:
```python
# Step 1: Handle Missing Values
imputer = SimpleImputer(strategy="mean")
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
```

```python
# Step 2: Feature Scaling
scaler = StandardScaler()
scaled_features = pd.DataFrame(scaler.fit_transform(data_imputed), columns=data.columns)

# Step 3: Feature Engineering (Polynomial Features)
poly = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
poly_features = pd.DataFrame(poly.fit_transform(scaled_features), columns=poly.get_feature_names_out())

# Step 4: Outlier Detection and Removal
outlier_detector = IsolationForest(contamination=0.05, random_state=42)
outlier_labels = outlier_detector.fit_predict(poly_features)
poly_features['outlier'] = outlier_labels
cleaned_data = poly_features[poly_features['outlier'] == 1].drop(columns=['outlier'])
```

In [9]:

```python
# Align the target variable with cleaned data
target = data_imputed['medv']
cleaned_data_indices = cleaned_data.index
aligned_target = target.iloc[cleaned_data_indices]

# Step 5: Dimensionality Reduction (PCA)
pca = PCA(n_components=10)  # Reduce to 10 components
pca_features = pca.fit_transform(cleaned_data)
pca_features_df = pd.DataFrame(pca_features, columns=[f"PC{i+1}" for i in range(pca.n_components_)])

# Step 6: Split Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(
    pca_features_df, aligned_target, test_size=0.2, random_state=42
)
```

In [11]:

```python
# Display summary
print("Preprocessing completed successfully!")
print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Preprocessing completed successfully!
Training set shape: (384, 10)
Testing set shape: (96, 10)
```

In [13]:

```python
#EDA
```

In [15]:

```python
# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())
```

```
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
```

In [17]:
```python
# Check the shape of the dataset
print("\nDataset shape:", data.shape)
```

```
Dataset shape: (506, 14)
```

In [19]:
```python
# Check for missing values
print("\nMissing values:")
print(data.isnull().sum())
```

```
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
```

In [21]:
```python
# Display data types and basic info
print("\nDataset information:")
print(data.info())
```

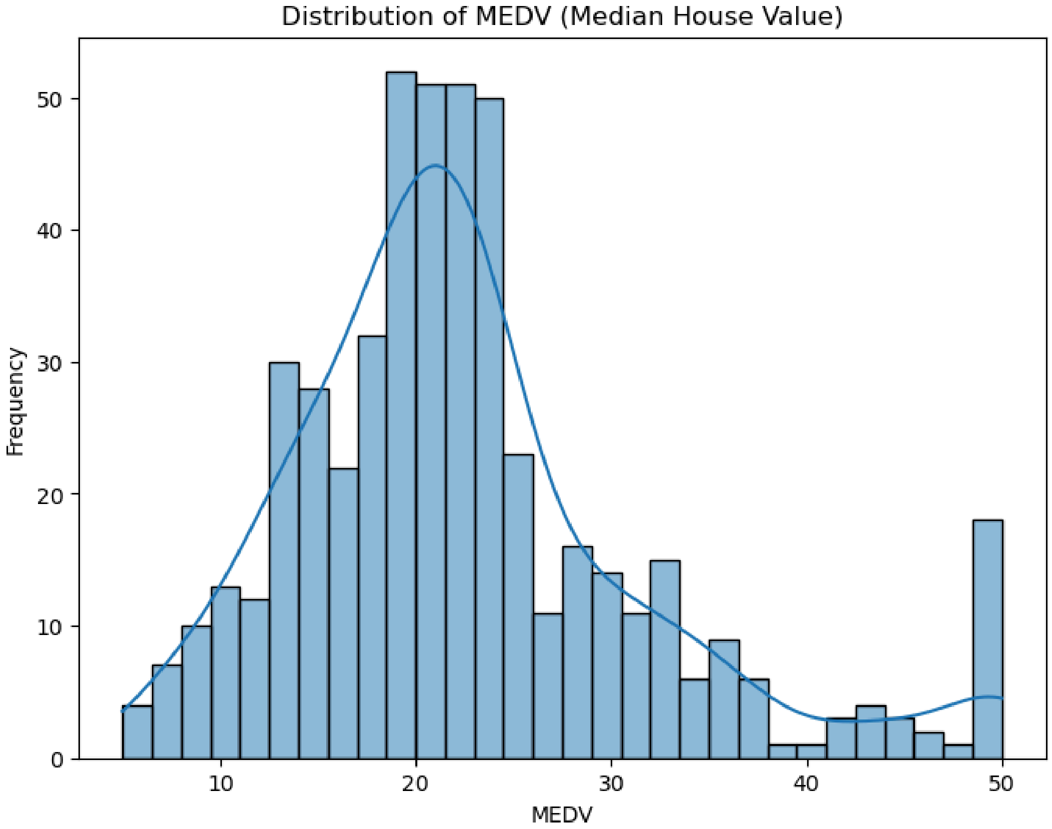
```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    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
None
```

In [23]:
```python
# Summary statistics
print("\nSummary statistics:")
print(data.describe())
```

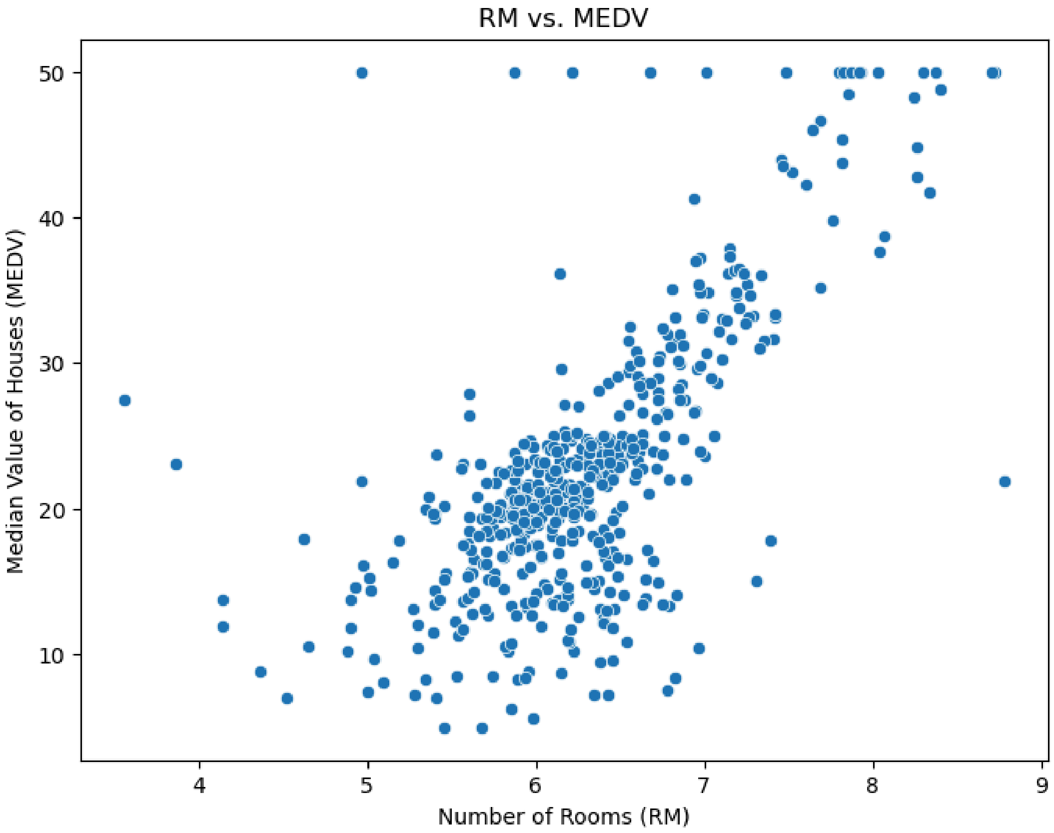
```
Summary statistics:
             crim          zn       indus        chas         nox          rm  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    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
```

In [25]:

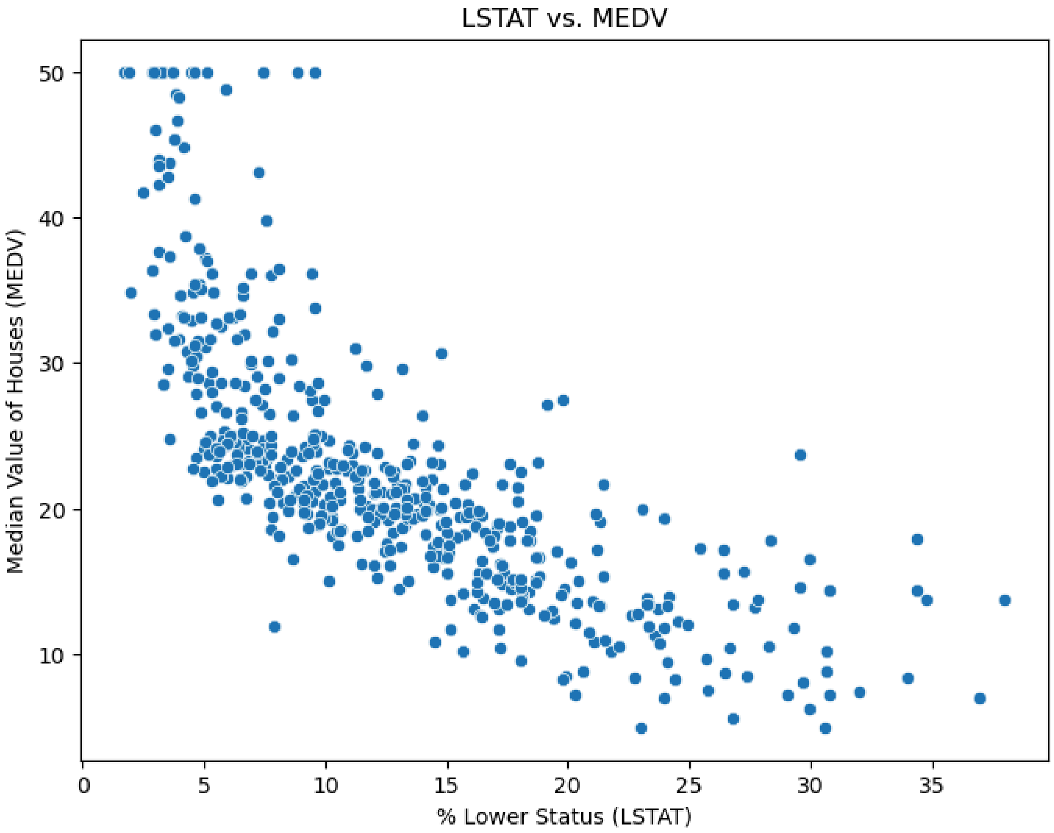
```python
# ----------- Visualization -----------
# 1. Distribution of the target variable (medv)
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.histplot(data['medv'], kde=True, bins=30)
plt.title('Distribution of MEDV (Median House Value)')
plt.xlabel('MEDV')
plt.ylabel('Frequency')
plt.show()
```



In [27]:
```
# 2. Scatterplots for feature-target relationships
# Scatterplot for RM (number of rooms) vs. MEDV
plt.figure(figsize=(8, 6))
sns.scatterplot(x='rm', y='medv', data=data)
plt.title('RM vs. MEDV')
plt.xlabel('Number of Rooms (RM)')
plt.ylabel('Median Value of Houses (MEDV)')
plt.show()
```

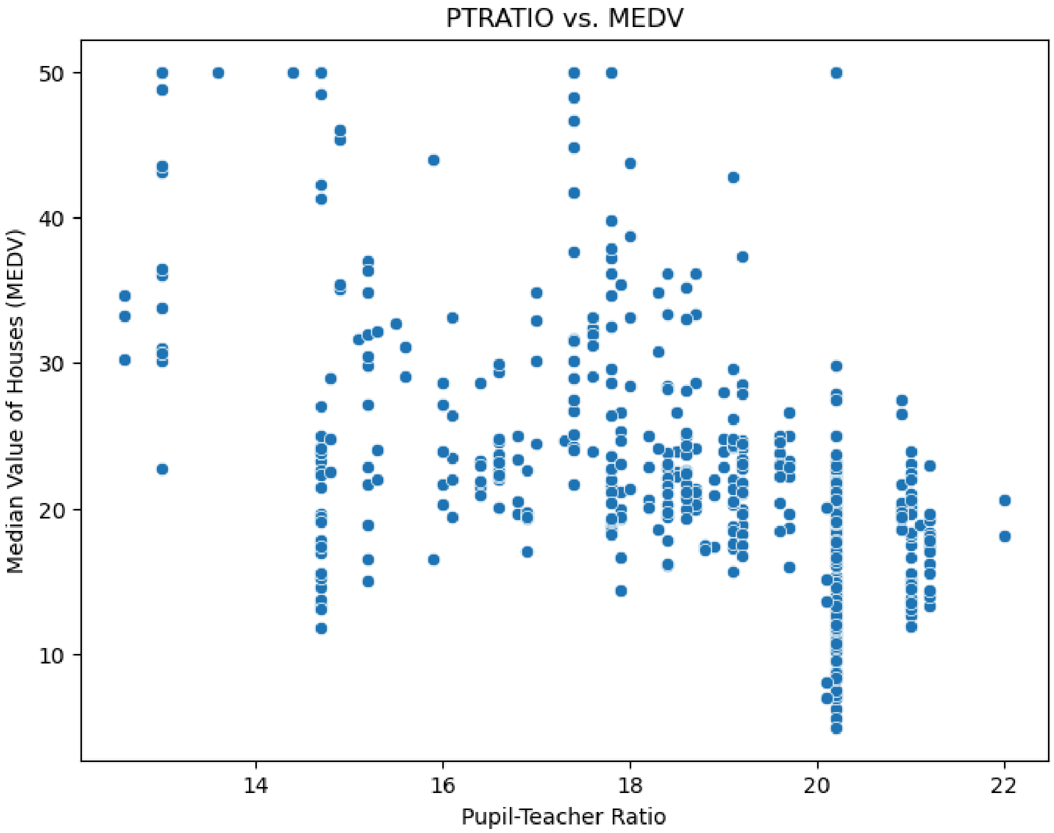


In [29]:
```
# Scatterplot for LSTAT (% lower status) vs. MEDV
plt.figure(figsize=(8, 6))
sns.scatterplot(x='lstat', y='medv', data=data)
plt.title('LSTAT vs. MEDV')
plt.xlabel('% Lower Status (LSTAT)')
plt.ylabel('Median Value of Houses (MEDV)')
plt.show()
```

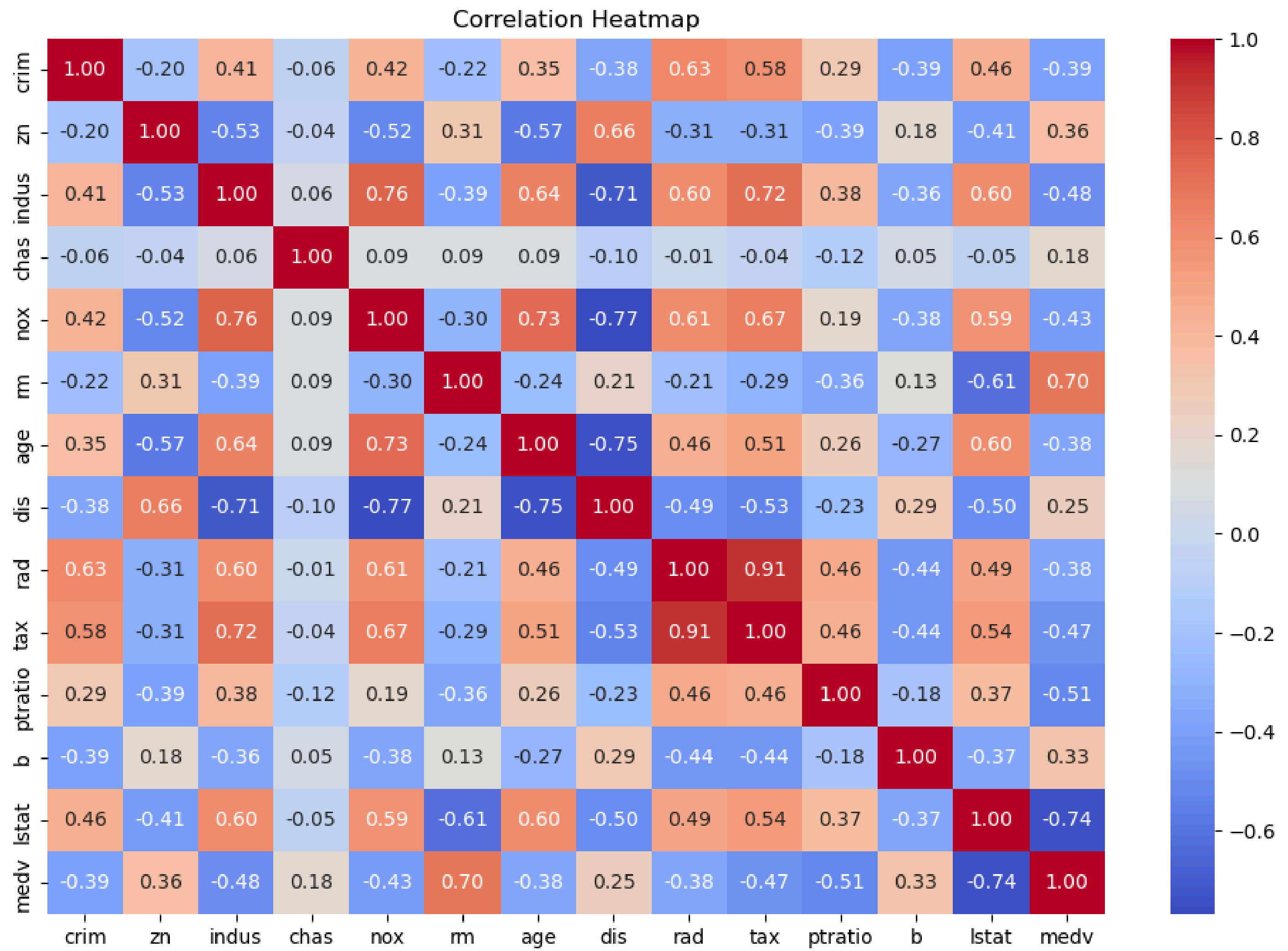


In [31]:

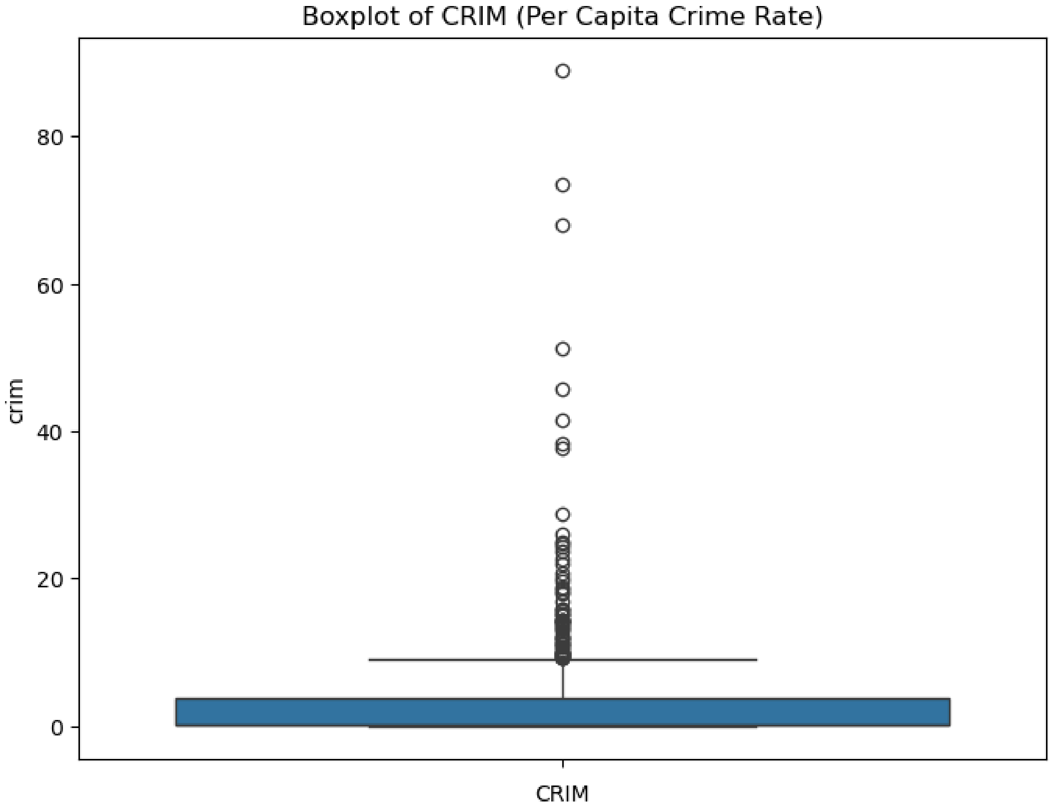
```
# Scatterplot for PTRATIO (Pupil-Teacher Ratio) vs. MEDV
plt.figure(figsize=(8, 6))
sns.scatterplot(x='ptratio', y='medv', data=data)
plt.title('PTRATIO vs. MEDV')
plt.xlabel('Pupil-Teacher Ratio')
plt.ylabel('Median Value of Houses (MEDV)')
plt.show()
```



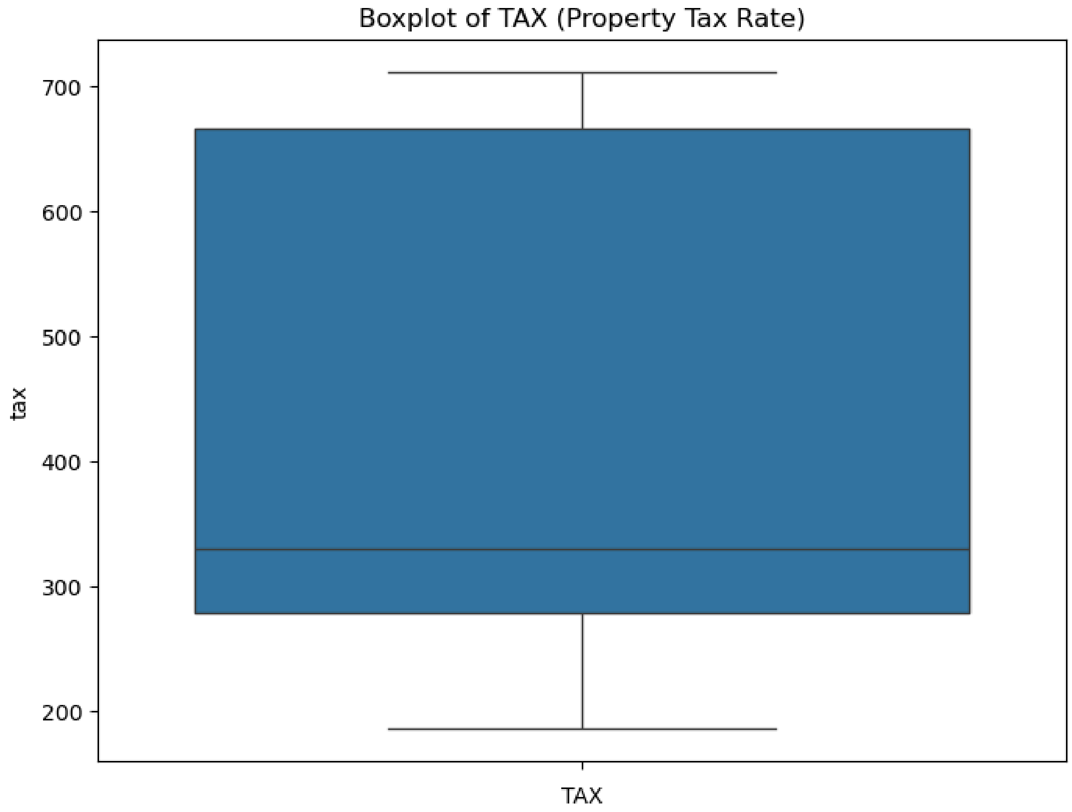
In [33]:
```
# 3. Correlation heatmap for all features
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



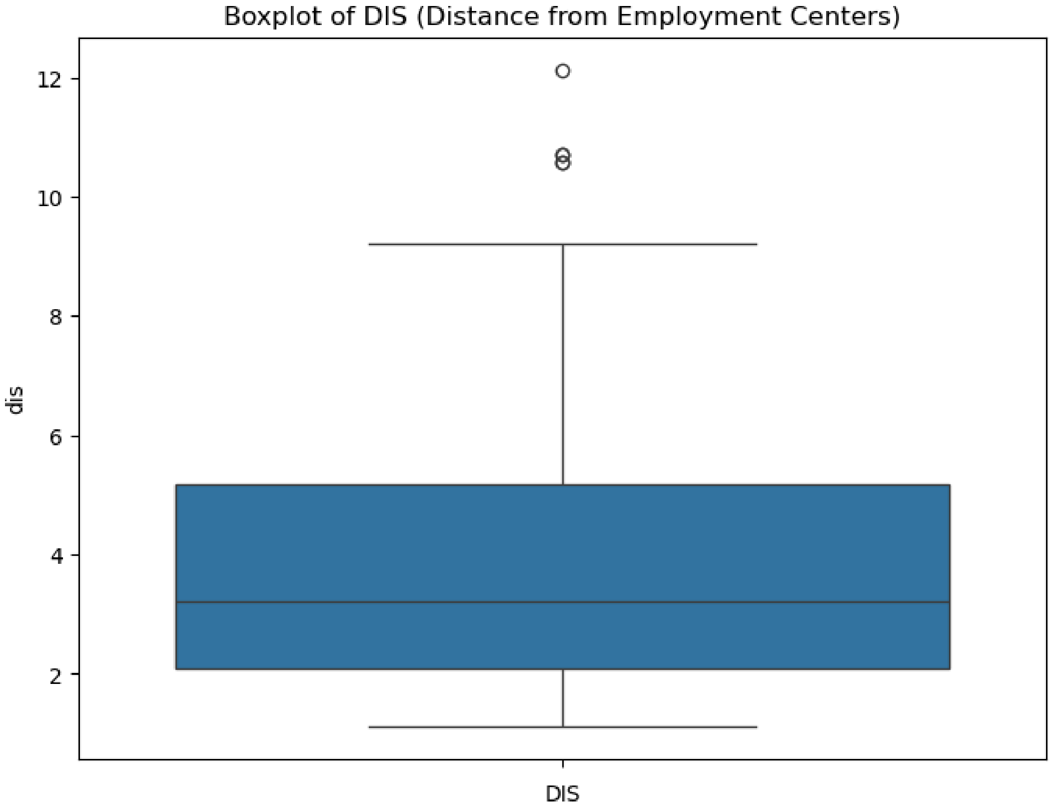
In [35]:
```
# 4. Boxplots for outlier detection
# Boxplot for CRIM (per capita crime rate)
plt.figure(figsize=(8, 6))
sns.boxplot(data['crim'])
plt.title('Boxplot of CRIM (Per Capita Crime Rate)')
plt.xlabel('CRIM')
plt.show()
```



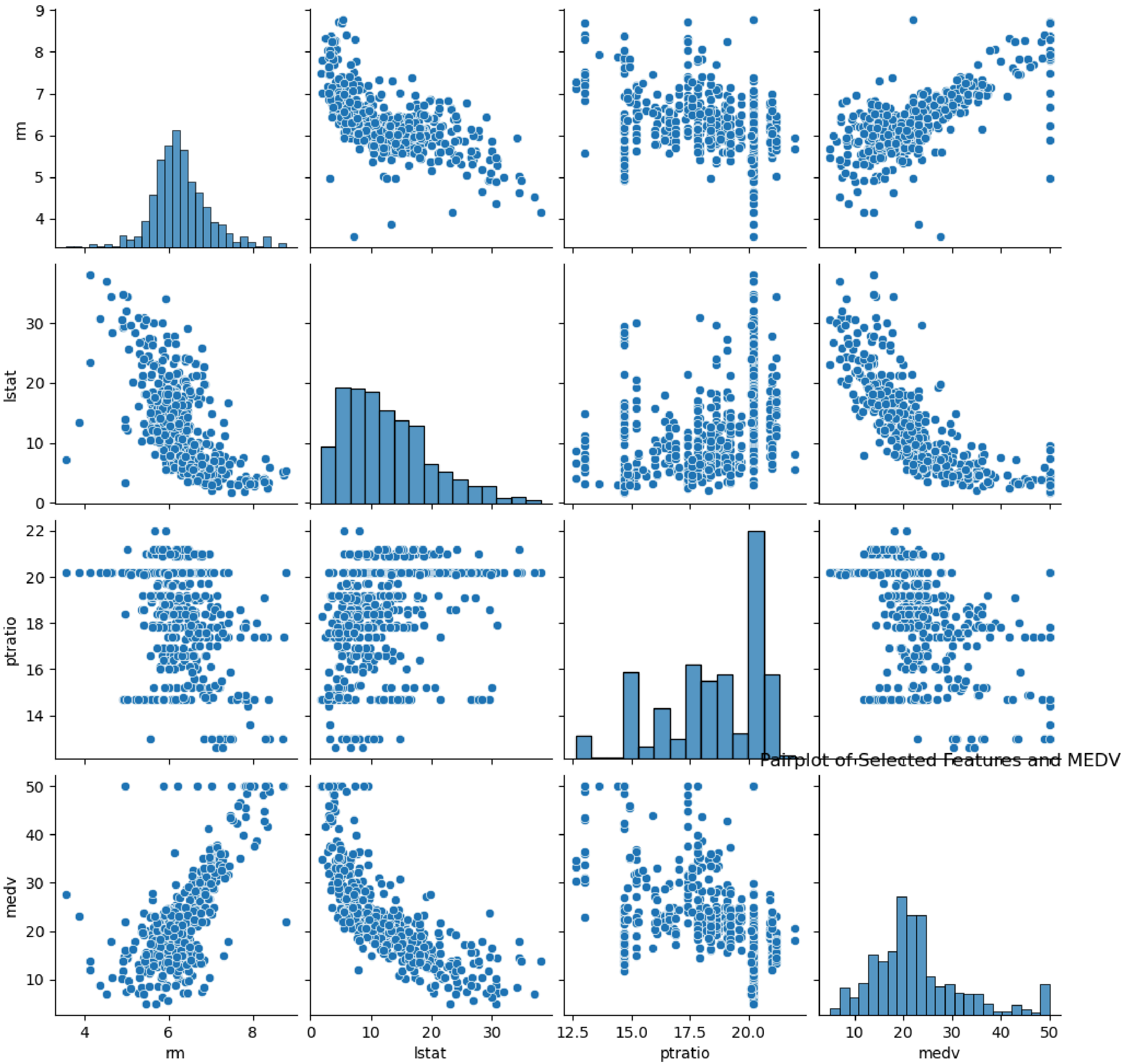
In [37]:
```
# Boxplot for TAX (property tax rate)
plt.figure(figsize=(8, 6))
sns.boxplot(data['tax'])
plt.title('Boxplot of TAX (Property Tax Rate)')
plt.xlabel('TAX')
plt.show()
```



In [39]:
```
# Boxplot for DIS (distance from employment centers)
plt.figure(figsize=(8, 6))
sns.boxplot(data['dis'])
plt.title('Boxplot of DIS (Distance from Employment Centers)')
plt.xlabel('DIS')
plt.show()
```

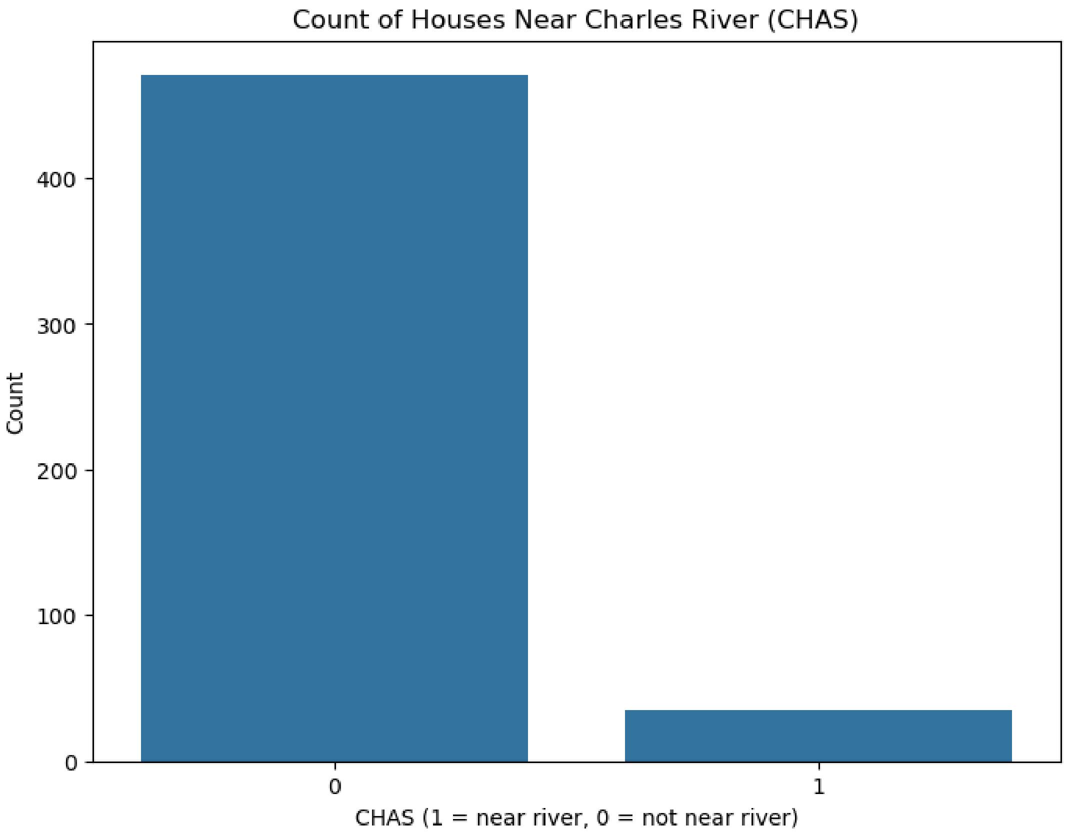


In [41]:
```
# 5. Pairplot for selected features and MEDV
selected_features = ['rm', 'lstat', 'ptratio', 'medv']
sns.pairplot(data[selected_features])
plt.title('Pairplot of Selected Features and MEDV')
plt.show()
```


Pairplot of Selected Features and MEDV
rm
lstat
ptratio
medv

In [43]:

```python
# 6. Countplot for categorical feature (CHAS)
plt.figure(figsize=(8, 6))
sns.countplot(x='chas', data=data)
plt.title('Count of Houses Near Charles River (CHAS)')
plt.xlabel('CHAS (1 = near river, 0 = not near river)')
plt.ylabel('Count')
plt.show()
```



In [45]:

```python
## Linear Regression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and fit the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predict on the test set
y_pred_linear = linear_model.predict(X_test)

# Evaluate the Linear Regression model
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print("Linear Regression Results:")
```

```
print(f"Mean Squared Error (MSE): {mse_linear:.2f}")
print(f"R-squared (R²): {r2_linear:.2f}")
```

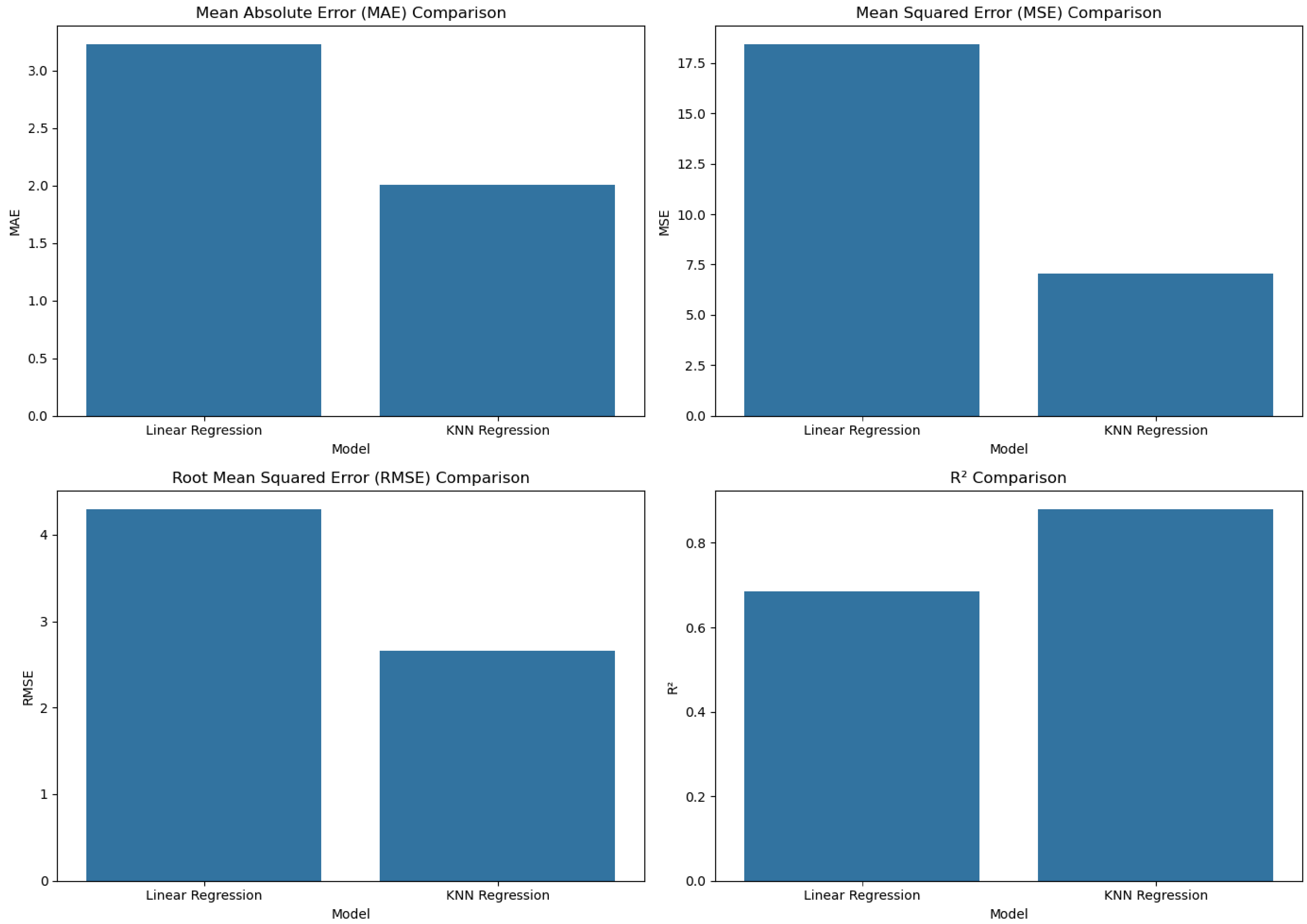
```
Linear Regression Results:
Mean Squared Error (MSE): 18.42
R-squared (R²): 0.68
```

In [47]:
```
## KNN

from sklearn.neighbors import KNeighborsRegressor

# Initialize the KNN model with 5 neighbors (can tune this hyperparameter)
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Predict on the test set
y_pred_knn = knn_model.predict(X_test)

# Evaluate the KNN Regression model
mse_knn = mean_squared_error(y_test, y_pred_knn)
r2_knn = r2_score(y_test, y_pred_knn)

print("K-Nearest Neighbors Regression Results:")
print(f"Mean Squared Error (MSE): {mse_knn:.2f}")
print(f"R-squared (R²): {r2_knn:.2f}")
```

```
K-Nearest Neighbors Regression Results:
Mean Squared Error (MSE): 7.06
R-squared (R²): 0.88
```

In [49]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Comparison of Models
models = ['Linear Regression', 'KNN Regression']
mse_knn = mean_squared_error(y_test, y_pred_knn)
mse_lr = mean_squared_error(y_test, y_pred_linear)
mae_scores = [mean_absolute_error(y_test, y_pred_linear), mean_absolute_error(y_test, y_pred_knn)]
mse_scores = [mse_lr, mse_knn]
rmse_scores = [np.sqrt(mse_lr), np.sqrt(mse_knn)]
r2_scores = [r2_linear, r2_score(y_test, y_pred_knn)]

# Creating a DataFrame for better visualization
comparison_df_v1 = pd.DataFrame({
    'Model': models,
    'MAE': mae_scores,
    'MSE': mse_scores,
    'RMSE': rmse_scores,
    'R²': r2_scores
})

# Plot MAE, MSE, RMSE, and R² for both models
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# MAE plot
sns.barplot(x='Model', y='MAE', data=comparison_df_v1, ax=axes[0, 0])
axes[0, 0].set_title('Mean Absolute Error (MAE) Comparison')

# MSE plot
sns.barplot(x='Model', y='MSE', data=comparison_df_v1, ax=axes[0, 1])
axes[0, 1].set_title('Mean Squared Error (MSE) Comparison')
```

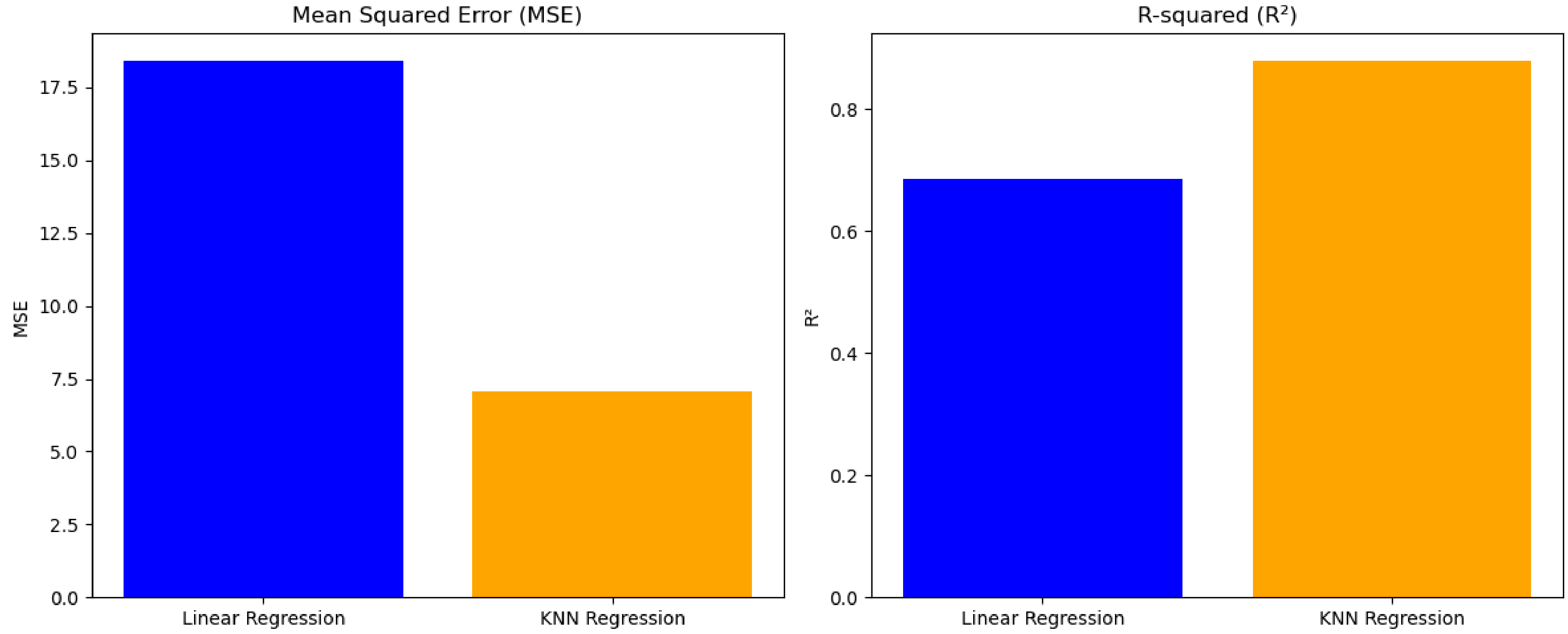
```
# RMSE plot
sns.barplot(x='Model', y='RMSE', data=comparison_df_v1, ax=axes[1, 0])
axes[1, 0].set_title('Root Mean Squared Error (RMSE) Comparison')

# R² plot
sns.barplot(x='Model', y='R²', data=comparison_df_v1, ax=axes[1, 1])
axes[1, 1].set_title('R² Comparison')

plt.tight_layout()
plt.show()

print(comparison_df_v1)
```



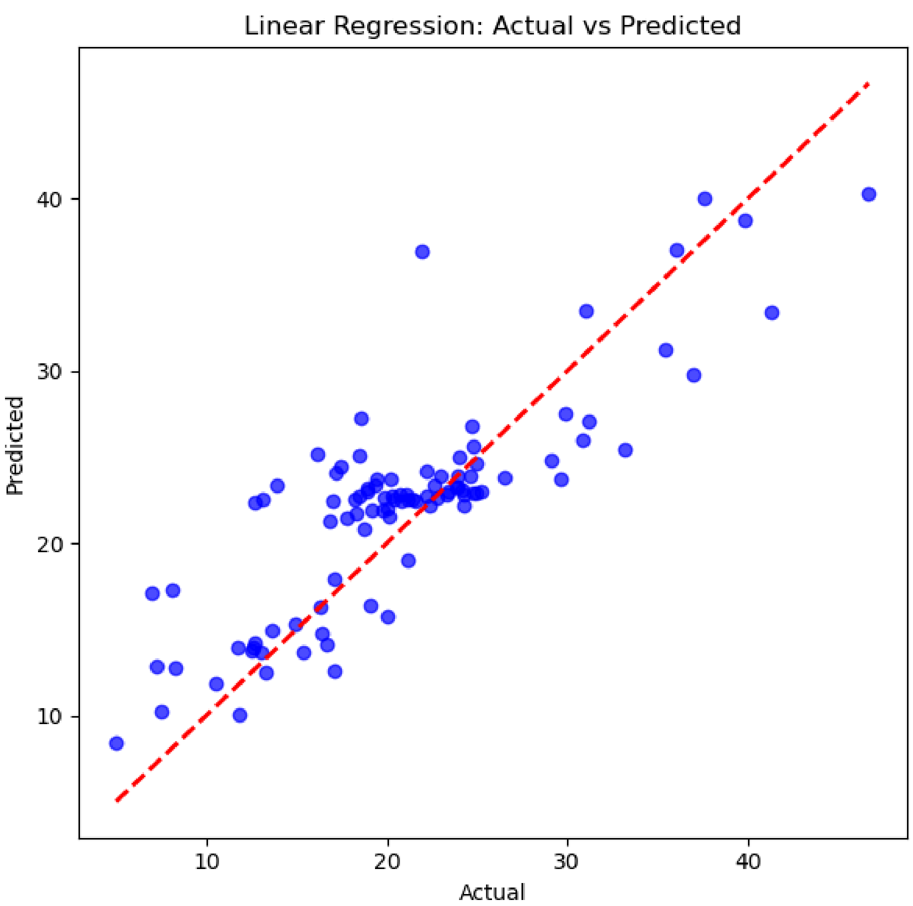
	Model	MAE	MSE	RMSE	R²
0	Linear Regression	3.226640	18.422169	4.292105	0.684924
1	KNN Regression	2.003333	7.061042	2.657262	0.879234

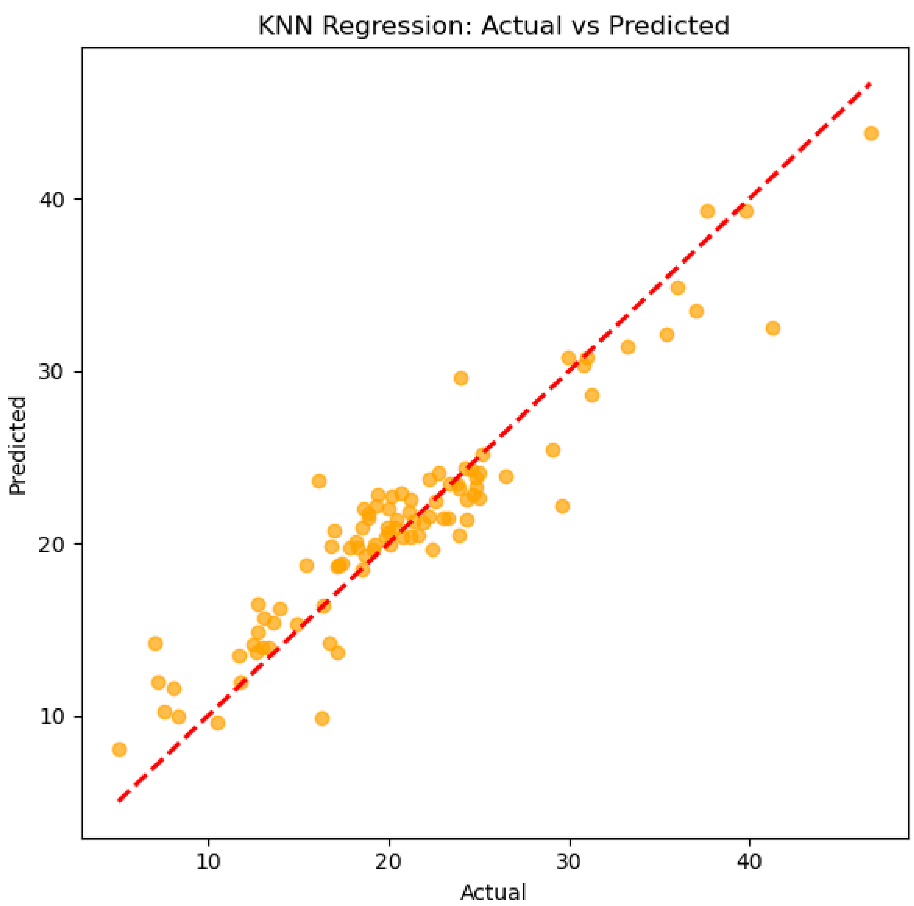
In [51]:
```python
import matplotlib.pyplot as plt
import pandas as pd

# Create a DataFrame to store the results for comparison
comparison_df = pd.DataFrame({
    'Model': ['Linear Regression', 'KNN Regression'],
    'Mean Squared Error (MSE)': [mse_linear, mean_squared_error(y_test, y_pred_knn)],
    'R-squared (R²)': [r2_linear, r2_score(y_test, y_pred_knn)]
})

# Print the comparison table
print("Model Comparison:")
print(comparison_df)

# Visualize the comparison of metrics
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Bar plot for Mean Squared Error (MSE)
axes[0].bar(comparison_df['Model'], comparison_df['Mean Squared Error (MSE)'], color=['blue', 'orange'])
axes[0].set_title('Mean Squared Error (MSE)')
axes[0].set_ylabel('MSE')

# Bar plot for R-squared (R²)
axes[1].bar(comparison_df['Model'], comparison_df['R-squared (R²)'], color=['blue', 'orange'])
axes[1].set_title('R-squared (R²)')
axes[1].set_ylabel('R²')

plt.tight_layout()
plt.show()

# Scatter plot for Actual vs Predicted values for both models
plt.figure(figsize=(12, 6))

# Linear Regression scatter plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_linear, alpha=0.7, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
plt.title('Linear Regression: Actual vs Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')

# KNN Regression scatter plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_knn, alpha=0.7, color='orange')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
plt.title('KNN Regression: Actual vs Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')

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

```
Model Comparison:
               Model  Mean Squared Error (MSE)  R-squared (R²)
0  Linear Regression                 18.422169        0.684924
1     KNN Regression                  7.061042        0.879234
```




```
/var/folders/vc/n9qx_9sd7kd7vykk06n6tslh0000gn/T/ipykernel_31561/660447161.py:37: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
/var/folders/vc/n9qx_9sd7kd7vykk06n6tslh0000gn/T/ipykernel_31561/660447161.py:45: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
```





Neural Networks and Random Forest

In [54]:
```
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor

# Apply Neural Network
nn_model = MLPRegressor(hidden_layer_sizes=(100,50), max_iter=1000, random_state=42)
nn_model.fit(X_train, y_train)
y_pred_nn = nn_model.predict(X_test)

# Calculate Neural Network Metrics
mse_nn = mean_squared_error(y_test, y_pred_nn)
r2_nn = r2_score(y_test, y_pred_nn)

# Apply Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Calculate Random Forest Metrics
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

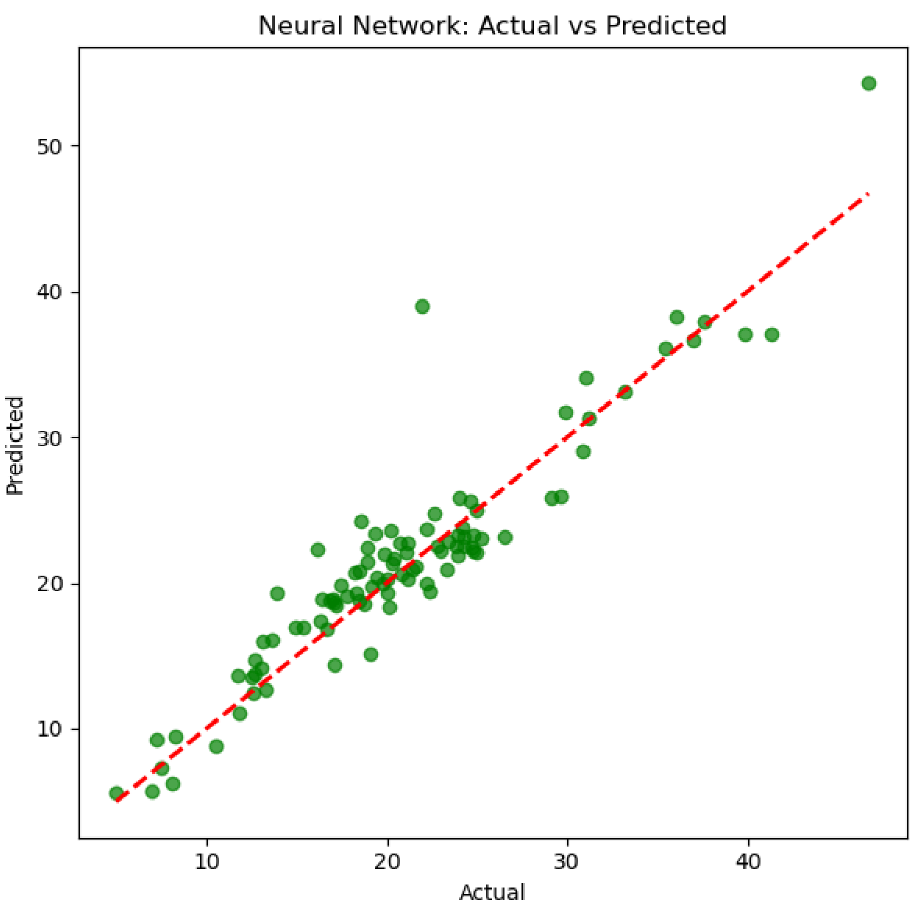
```python
# Print Metrics for Neural Network and Random Forest
print(f"Neural Network: MSE = {mse_nn:.4f}, R² = {r2_nn:.4f}")
print(f"Random Forest: MSE = {mse_rf:.4f}, R² = {r2_rf:.4f}")
```

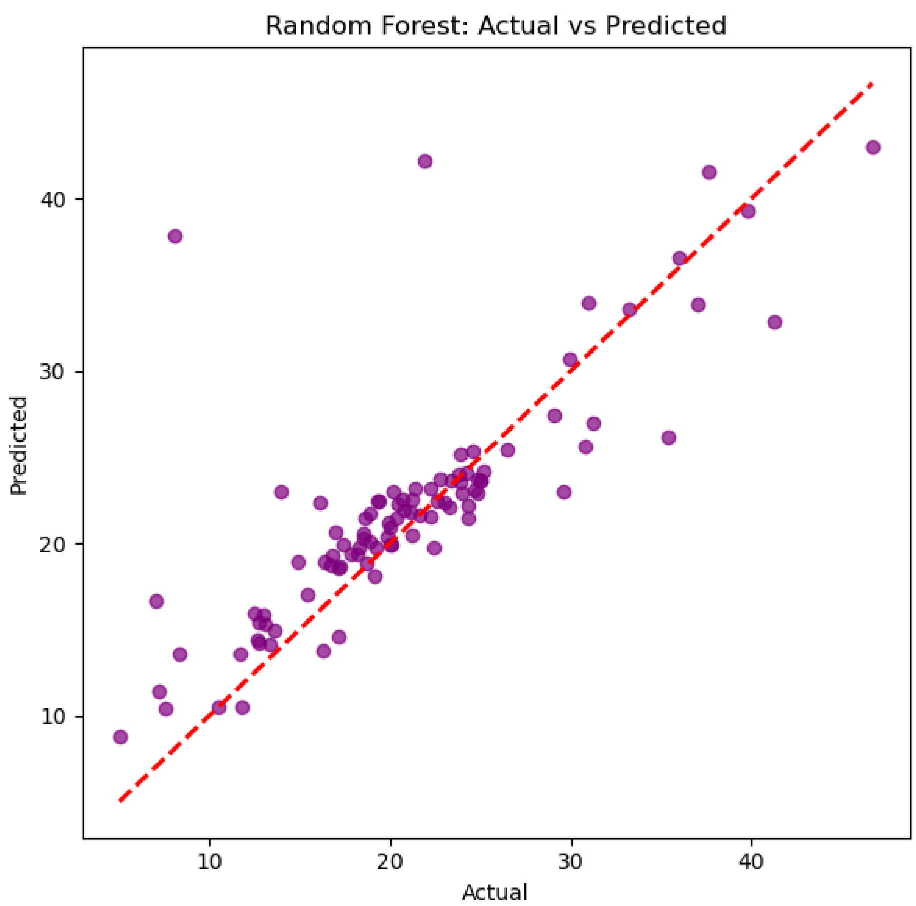
```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  warnings.warn(
Neural Network: MSE = 8.1102, R² = 0.8613
Random Forest: MSE = 21.9673, R² = 0.6243
```

In [55]:

```python
from sklearn.metrics import mean_squared_error
import numpy as np





# Scatter plots for Neural Network and Random Forest predictions
plt.figure(figsize=(12, 6))

# Neural Network scatter plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_nn, alpha=0.7, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
plt.title('Neural Network: Actual vs Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')

# Random Forest scatter plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_rf, alpha=0.7, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
plt.title('Random Forest: Actual vs Predicted')
plt.xlabel('Actual')
plt.ylabel('Predicted')

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

```
/var/folders/vc/n9qx_9sd7kd7vykk06n6tslh0000gn/T/ipykernel_31561/4127734519.py:14: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
/var/folders/vc/n9qx_9sd7kd7vykk06n6tslh0000gn/T/ipykernel_31561/4127734519.py:22: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "k--" (-> color='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2, color='red')
```

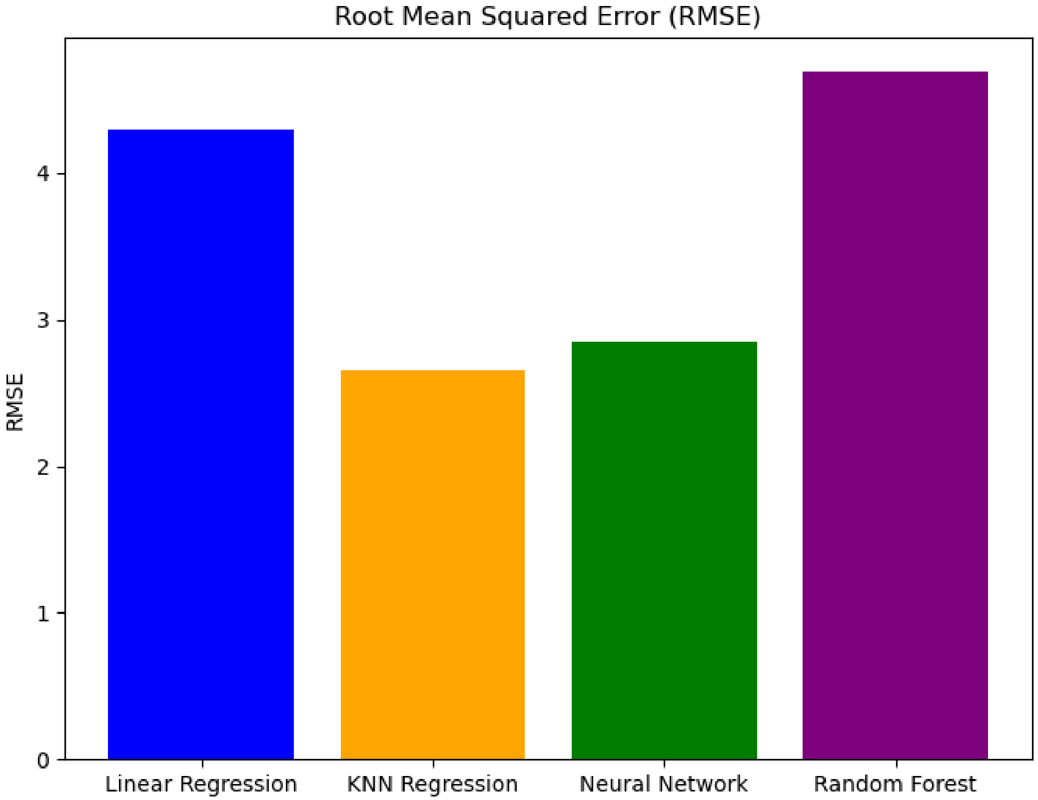




In [58]:
```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Calculate RMSE, MSE, and R² for each model
rmse_linear = np.sqrt(mse_linear)
r2_linear = r2_linear  # Assuming r2_linear is already computed
mse_linear = mse_linear

rmse_knn = np.sqrt(mean_squared_error(y_test, y_pred_knn))
r2_knn = r2_score(y_test, y_pred_knn)
mse_knn = mean_squared_error(y_test, y_pred_knn)

rmse_nn = np.sqrt(mse_nn)
r2_nn = r2_nn  # Assuming r2_nn is already computed
mse_nn = mse_nn

rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_rf  # Assuming r2_rf is already computed
mse_rf = mse_rf

# Print metrics for all models
print(f"Linear Regression: RMSE = {rmse_linear:.4f}, MSE = {mse_linear:.4f}, R² = {r2_linear:.4f}")
print(f"KNN Regression: RMSE = {rmse_knn:.4f}, MSE = {mse_knn:.4f}, R² = {r2_knn:.4f}")
print(f"Neural Network: RMSE = {rmse_nn:.4f}, MSE = {mse_nn:.4f}, R² = {r2_nn:.4f}")
```

```python
print(f"Random Forest: RMSE = {rmse_rf:.4f}, MSE = {mse_rf:.4f}, R² = {r2_rf:.4f} \n")



# Update the comparison DataFrame with RMSE
comparison_df = pd.DataFrame({
    'Model': ['Linear Regression', 'KNN Regression', 'Neural Network', 'Random Forest'],
    'Mean Squared Error (MSE)': [mse_linear, mean_squared_error(y_test, y_pred_knn), mse_nn, mse_rf],
    'Root Mean Squared Error (RMSE)': [rmse_linear, rmse_knn, rmse_nn, rmse_rf],
    'R-squared (R²)': [r2_linear, r2_score(y_test, y_pred_knn), r2_nn, r2_rf]
})


# Print the updated comparison table
print("Model Comparison:")
print(comparison_df)

# Visualize RMSE comparison
plt.figure(figsize=(8, 6))
plt.bar(comparison_df['Model'], comparison_df['Root Mean Squared Error (RMSE)'], color=['blue', 'orange', 'green', 'purple'])
plt.title('Root Mean Squared Error (RMSE)')
plt.ylabel('RMSE')
plt.show()
```

```
Linear Regression: RMSE = 4.2921, MSE = 18.4222, R² = 0.6849
KNN Regression: RMSE = 2.6573, MSE = 7.0610, R² = 0.8792
Neural Network: RMSE = 2.8478, MSE = 8.1102, R² = 0.8613
Random Forest: RMSE = 4.6869, MSE = 21.9673, R² = 0.6243

Model Comparison:
               Model  Mean Squared Error (MSE)  \
0  Linear Regression                 18.422169
1     KNN Regression                  7.061042
2     Neural Network                  8.110185
3      Random Forest                 21.967267

   Root Mean Squared Error (RMSE)  R-squared (R²)
0                        4.292105        0.684924
1                        2.657262        0.879234
2                        2.847839        0.861291
3                        4.686925        0.624292
```



In [60]:
```
!pip install shap
```

```
Requirement already satisfied: shap in /opt/anaconda3/lib/python3.12/site-packages (0.46.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages (from shap) (1.26.4)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.12/site-packages (from shap) (1.13.1)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.12/site-packages (from shap) (1.4.2)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.12/site-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /opt/anaconda3/lib/python3.12/site-packages (from shap) (4.66.4)
Requirement already satisfied: packaging>20.9 in /opt/anaconda3/lib/python3.12/site-packages (from shap) (23.2)
Requirement already satisfied: slicer==0.0.8 in /opt/anaconda3/lib/python3.12/site-packages (from shap) (0.0.8)
Requirement already satisfied: numba in /opt/anaconda3/lib/python3.12/site-packages (from shap) (0.59.1)
Requirement already satisfied: cloudpickle in /opt/anaconda3/lib/python3.12/site-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in /opt/anaconda3/lib/python3.12/site-packages (from numba->shap) (0.42.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.12/site-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.12/site-packages (from pandas->shap) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/python3.12/site-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.2.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-learn->shap) (2.2.0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.12/site-packages (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
```

In [62]:
```
import shap

# Initialize SHAP explainer and compute SHAP values for each model
# explainer_linear = shap.Explainer(linear_model.predict, X_test)
# shap_values_linear = explainer_linear(X_test)

# explainer_knn = shap.KernelExplainer(knn_model.predict, X_train)
# shap_values_knn = explainer_knn.shap_values(X_test)
```

```
explainer_nn = shap.KernelExplainer(nn_model.predict, X_train)
shap_values_nn = explainer_nn.shap_values(X_test)

# explainer_rf = shap.TreeExplainer(rf_model)
# shap_values_rf = explainer_rf(X_test)

# # SHAP Summary Plot for Linear Regression
# print("SHAP Summary Plot for Linear Regression")
# shap.summary_plot(shap_values_linear, X_test)

# # SHAP Summary Plot for KNN
# print("SHAP Summary Plot for KNN")
# shap.summary_plot(shap_values_knn, X_test)

# SHAP Summary Plot for Neural Network
print("SHAP Summary Plot for Neural Network")
shap.summary_plot(shap_values_nn, X_test)

# # SHAP Summary Plot for Random Forest
# print("SHAP Summary Plot for Random Forest")
# shap.summary_plot(shap_values_rf, X_test)
```

```
Using 384 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K samples.
  0%|          | 0/96 [00:00<?, ?it/s]
SHAP Summary Plot for Neural Network
```