In [4]:
```python
import pandas as pd

# Load the dataset with the first column as a single column
file_path = 'housing (1).csv'  # Update with your actual file path
column_names = ['Column']  # Assuming the first column is named 'Column'
df = pd.read_csv(file_path, names=column_names)

# Split the first column into 14 parts based on whitespace
# Assuming the first column contains all 14 parts in a single string separated by whitespace
first_column_parts = df['Column'].str.split(expand=True)

# Assign column names to the separated parts
first_column_parts.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                              'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

# Print the separated columns
print("Separated columns:")
print(first_column_parts)
```

```
Separated columns:
        CRIM     ZN   INDUS CHAS     NOX      RM    AGE     DIS RAD    TAX  \
0    0.00632  18.00   2.310    0  0.5380  6.5750  65.20  4.0900   1  296.0
1    0.02731   0.00   7.070    0  0.4690  6.4210  78.90  4.9671   2  242.0
2    0.02729   0.00   7.070    0  0.4690  7.1850  61.10  4.9671   2  242.0
3    0.03237   0.00   2.180    0  0.4580  6.9980  45.80  6.0622   3  222.0
4    0.06905   0.00   2.180    0  0.4580  7.1470  54.20  6.0622   3  222.0
..       ...    ...     ...  ...     ...     ...    ...     ... ..    ...
501  0.06263   0.00  11.930    0  0.5730  6.5930  69.10  2.4786   1  273.0
502  0.04527   0.00  11.930    0  0.5730  6.1200  76.70  2.2875   1  273.0
503  0.06076   0.00  11.930    0  0.5730  6.9760  91.00  2.1675   1  273.0
504  0.10959   0.00  11.930    0  0.5730  6.7940  89.30  2.3889   1  273.0
505  0.04741   0.00  11.930    0  0.5730  6.0300  80.80  2.5050   1  273.0

    PTRATIO       B LSTAT   MEDV
0     15.30  396.90  4.98  24.00
1     17.80  396.90  9.14  21.60
2     17.80  392.83  4.03  34.70
3     18.70  394.63  2.94  33.40
4     18.70  396.90  5.33  36.20
..      ...     ...   ...    ...
501   21.00  391.99  9.67  22.40
502   21.00  396.90  9.08  20.60
503   21.00  396.90  5.64  23.90
504   21.00  393.45  6.48  22.00
505   21.00  396.90  7.88  11.90

[506 rows x 14 columns]
```

In [5]:
```python
import pandas as pd

# Load the dataset with the first column as a single column
file_path = 'housing (1).csv'  # Update with your actual file path
column_names = ['Column']  # Assuming the first column is named 'Column'
df = pd.read_csv(file_path, names=column_names)

# Split the first column into 14 parts based on whitespace
# Assuming the first column contains all 14 parts in a single string separated by whitespace
first_column_parts = df['Column'].str.split(expand=True)

# Assign column names to the separated parts
first_column_parts.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                              'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

# Save the separated DataFrame to a new CSV file
output_file = 'housing_separated.csv'  # Specify the desired output file path
first_column_parts.to_csv(output_file, index=False)

print(f"Separated columns saved to '{output_file}'.")
```

```
Separated columns saved to 'housing_separated.csv'.
```

In [7]:
```python
import pandas as pd

# Load the dataset with the first column as a single column
file_path = 'housing (1).csv'  # Update with your actual file path
column_names = ['Column']  # Assuming the first column is named 'Column'
df = pd.read_csv(file_path, names=column_names)

# Split the first column into 14 parts based on whitespace
# Assuming the first column contains all 14 parts in a single string separated by whitespace
first_column_parts = df['Column'].str.split(expand=True)

# Assign column names to the separated parts
first_column_parts.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                              'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

# Convert all columns to numeric, coercing errors to NaN
for col in first_column_parts.columns:
    first_column_parts[col] = pd.to_numeric(first_column_parts[col], errors='coerce')

# Print missing values before imputation
print("Missing values before imputation:")
print(first_column_parts.isnull().sum())

# Impute missing values with median
first_column_parts = first_column_parts.fillna(first_column_parts.median())

# Print missing values after imputation
print("\nMissing values after imputation:")
print(first_column_parts.isnull().sum())

# Save the separated DataFrame with numeric and imputed values to a new CSV file
output_file = 'housing_separated_numeric_imputed.csv'  # Specify the desired output file path
first_column_parts.to_csv(output_file, index=False)

print(f"\nSeparated columns with numeric and imputed values saved to '{output_file}'.")
```

```
Missing values before imputation:
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

Missing values after imputation:
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

Separated columns with numeric and imputed values saved to 'housing_separated_numeric_imputed.csv'.
```

In [ ]:

In [9]:
```python
from sklearn.preprocessing import MinMaxScaler

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Perform Min-Max scaling on the dataset
scaled_data = scaler.fit_transform(first_column_parts)

# Convert scaled_data back to a DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=first_column_parts.columns)

# Print the scaled DataFrame
print("Min-Max scaled data:")
print(scaled_df.head())
```

```
Min-Max scaled data:
       CRIM    ZN     INDUS  CHAS       NOX        RM       AGE       DIS  \
0  0.000000  0.18  0.067815   0.0  0.314815  0.577505  0.641607  0.269203
1  0.000236  0.00  0.242302   0.0  0.172840  0.547998  0.782698  0.348962
2  0.000236  0.00  0.242302   0.0  0.172840  0.694386  0.599382  0.348962
3  0.000293  0.00  0.063050   0.0  0.150206  0.658555  0.441813  0.448545
4  0.000705  0.00  0.063050   0.0  0.150206  0.687105  0.528321  0.448545

        RAD       TAX   PTRATIO         B     LSTAT      MEDV
0  0.000000  0.208015  0.287234  1.000000  0.089680  0.422222
1  0.043478  0.104962  0.553191  1.000000  0.204470  0.368889
2  0.043478  0.104962  0.553191  0.989737  0.063466  0.660000
3  0.086957  0.066794  0.648936  0.994276  0.033389  0.631111
4  0.086957  0.066794  0.648936  1.000000  0.099338  0.693333
```

In [10]:
```python
from sklearn.model_selection import train_test_split

# Assuming first_column_parts is your processed DataFrame

# Define your features (X) and target variable (y)
X = first_column_parts.drop(columns=['MEDV'])  # Features: all columns except 'MEDV'
y = first_column_parts['MEDV']  # Target variable: 'MEDV'

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print the shapes of the training and testing sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (404, 13)
Shape of X_test: (102, 13)
Shape of y_train: (404,)
Shape of y_test: (102,)
```

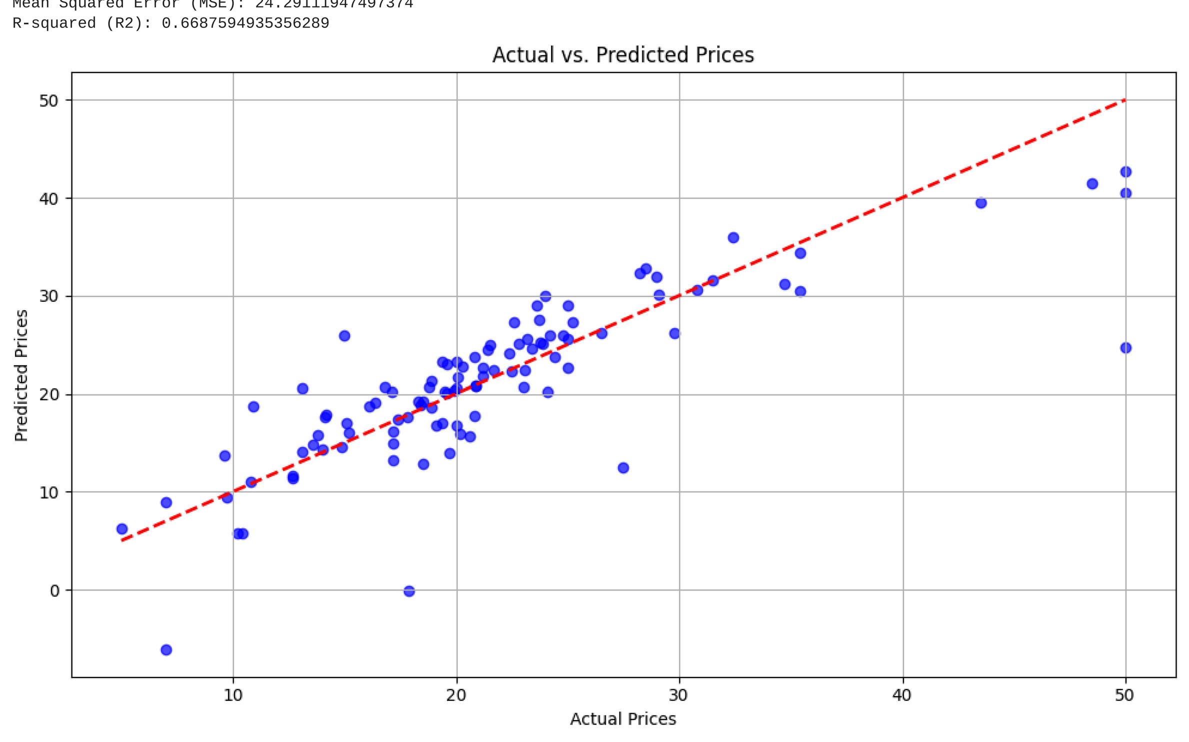
In [17]:
```python
import pandas as pd

# Load the dataset with proper parsing
file_path = 'housing (1).csv'
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

# Assuming the data is space-separated, use delimiter='\s+' for one or more spaces
df = pd.read_csv(file_path, names=column_names, delimiter='\s+', header=None)

# Display the first few rows to verify the data
print(df.head())
```

```
      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  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

```
<>:9: SyntaxWarning: invalid escape sequence '\s'
<>:9: SyntaxWarning: invalid escape sequence '\s'
C:\Users\HP\AppData\Local\Temp\ipykernel_22632\1960855550.py:9: SyntaxWarning: invalid escape sequence '\s'
  df = pd.read_csv(file_path, names=column_names, delimiter='\s+', header=None)
```

In [32]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load your dataset
file_path = 'housing_separated_numeric_imputed.csv'
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df = pd.read_csv(file_path, names=column_names, skiprows=1)  # Skip header if present

# Separate features and target variable
X = df.drop(columns=['MEDV'])  # Features
y = df['MEDV']  # Target variable
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the sizes of the split datasets
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
# Initialize the model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
# Plotting actual vs. predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red', linewidth=2)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual vs. Predicted Prices')
plt.grid(True)
plt.tight_layout()
plt.show()
# Retrieve and print model coefficients
coef_df = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef_})
print("Model Coefficients:")
print(coef_df)
```

```
X_train shape: (404, 13), y_train shape: (404,)
X_test shape: (102, 13), y_test shape: (102,)
Mean Absolute Error (MAE): 3.189091965887874
Mean Squared Error (MSE): 24.29111947497374
R-squared (R2): 0.6687594935356289
```




```
Model Coefficients:
    Feature  Coefficient
0      CRIM    -0.113056
1        ZN     0.030110
2     INDUS     0.040381
3      CHAS     2.784438
4       NOX   -17.202633
5        RM     4.438835
6       AGE    -0.006296
7       DIS    -1.447865
8       RAD     0.262430
9       TAX    -0.010647
10  PTRATIO    -0.915456
11        B     0.012351
12    LSTAT    -0.508571
```

In [33]:
```python
## UNDERSTANDING AND INSIGHTS OF ALL

Handling Missing Values and Imputation:
Initially, there were challenges with handling missing values in your dataset, leading to issues when trying to convert string values to floats for numeric operations.
The strategy involved checking for missing values and then using methods like fillna() with median values to impute missing data. This approach helps ensure all features are properly populated for modeling.


Normalization and Scaling:
There was an intention to perform feature scaling and normalization, which is crucial for models like Linear Regression to perform optimally. This step typically involves scaling numerical features to a standard range (e.g., using Sta


Mean Absolute Error (MAE) and Mean Squared Error (MSE):
MAE and MSE were used as evaluation metrics to assess the model's performance. MAE measures the average magnitude of errors, while MSE provides a measure of the average squared difference between predicted and actual values.
These metrics help gauge how well the model predicts housing prices and can indicate areas for improvement.


Visualizing Model Performance:
The intention was to visualize the model's performance using plots, particularly to compare actual housing prices against predicted prices. This visual assessment helps in understanding the accuracy and reliability of the predictions


About model coefficient
CRIM (-0.113056): For each additional unit of CRIM (per capita crime rate by town), the dependent variable decreases by approximately 0.113 units, assuming other features remain constant.
ZN (0.030110): For each additional unit of ZN (proportion of residential land zoned for lots over 25,000 sq. ft.), the dependent variable increases by 0.030 units.
INDUS (0.040381): For each additional unit of INDUS (proportion of non-retail business acres per town), the dependent variable increases by 0.040 units.
CHAS (2.784438): If the property is adjacent to the Charles River (CHAS is a binary variable: 1 if adjacent, 0 otherwise), the dependent variable increases by 2.784 units. This suggests a substantial positive impact.
NOX (-17.202633): For each additional unit of NOX (nitric oxides concentration), the dependent variable decreases by 17.203 units, indicating a strong negative impact.
RM (4.438835): For each additional room (RM), the dependent variable increases by 4.439 units, suggesting a positive relationship.
AGE (-0.006296): For each additional percentage of owner-occupied units built prior to 1940 (AGE), the dependent variable decreases by 0.006 units.
DIS (-1.447865): For each additional unit of weighted distances to five Boston employment centers (DIS), the dependent variable decreases by 1.448 units.
RAD (0.262430): For each additional index of accessibility to radial highways (RAD), the dependent variable increases by 0.262 units.
TAX (-0.010647): For each additional unit of full-value property tax rate per $10,000 (TAX), the dependent variable decreases by 0.011 units
PTRATIO (-0.915456): For each additional student-teacher ratio (PTRATIO), the dependent variable decreases by 0.915 units.
B (0.012351): For each additional unit of 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town (B), the dependent variable increases by 0.012 units.
LSTAT (-0.508571): For each additional percentage of lower status of the population (LSTAT), the dependent variable decreases by 0.509 units.


DATA SHAPES
X_train shape: (404, 13): This means that the training dataset X_train contains 404 samples, each with 13 features. These are the independent variables used to train the model.
y_train shape: (404,): The y_train array contains the 404 target values corresponding to each sample in X_train. This is the dependent variable the model is trying to predict.
X_test shape: (102, 13): The test dataset X_test contains 102 samples, each with the same 13 features. This is used to evaluate the model's performance on unseen data.
y_test shape: (102,): The y_test array contains the 102 target values for each sample in X_test. It is used to assess how well the model performs on new data.


## A lower MAE indicates a better fit to the data. In this case, on average, the model's predictions are off by about 3.19 units from the actual values.
## A lower MSE indicates a better model, but it is more sensitive to outliers than MAE. Here, the MSE of 24.29 suggests how the squared errors average out, indicating some errors might be larger.
## An R² of 0.668 means that approximately 66.8% of the variance in the dependent variable is explained by the model. This indicates a moderately strong fit, suggesting the model captures the main trends in the data but could be impro


## INSIGHT on PLOT
X-axis (Actual Prices): This axis represents the actual values of the target variable from your test dataset (y_test).
Y-axis (Predicted Prices): This axis shows the predicted values of the target variable generated by the linear regression model (y_pred).

Each blue dot on the plot represents a data point from the test dataset. The position of the dot shows the actual value on the X-axis and the predicted value on the Y-axis for that particular instance.
Ideally, if the model's predictions were perfect, all the data points would lie directly on the diagonal red line (45-degree line).

The red dashed line represents the line of perfect prediction, where the predicted values equal the actual values (y_pred = y_test).
It serves as a reference to show how close the predictions are to the actual values.
```

```
Clustered Around the Red Line: If the majority of the points are close to this line, it indicates that the model has a good predictive capability.
Scatter and Spread: The distance of the points from the line shows the error in prediction. Points further away from the line indicate larger prediction errors.
```

In [ ]: