Project Title: **Predicting Housing Prices in Boston**

Team: Ying Lin Zhao (yz2433), Melanie Jalbert (mj496), Israel Davidson (izd3), Fiona Huang (xh393), David Park (yp358)

Objective

In this project, the goal is to predict housing prices in Boston using a variety of features from the Boston housing dataset. The dataset includes key characteristics such as crime rates, average number of rooms, distance to employment centers, and other socio-economic factors that potentially influence the price of homes. The focus is on answering the following questions:

- How can we use the data to predict housing prices?
- Which factors are the best predictors of housing prices?

The analysis aims to provide actionable insights for Urban Vision, a nonprofit focused on affordable housing and community development, to better understand the housing landscape in Boston and make informed decisions about future housing projects.

Data Overview

The Boston housing dataset contains information on various predictors for the median house value (medv) across different census tracts in the city. The dataset includes both quantitative and qualitative variables, such as:

- **crim**: Crime rate by town
- **zn**: Proportion of residential land zoned for large lots
- **indus**: Proportion of non-retail business acres per town
- **chas**: Dummy variable indicating proximity to the Charles River
- **nox**: Nitric oxides concentration
- **rm**: Average number of rooms per dwelling
- **age**: Proportion of owner-occupied units built before 1940
- **dis**: Weighted distances to employment centers
- **rad**: Index of accessibility to radial highways
- **tax**: Property tax rate
- **ptratio**: Pupil-teacher ratio by town
- **lstat**: Percentage of lower status of the population
- **medv**: Median house value in $1000s (target variable)

In [1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_validate, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.pipeline import Pipeline
from math import sqrt


df = pd.read_csv("data/Boston.csv",usecols=range(1,14))
```

In [2]:
```
# Understanding the size of the dataframe and ensuring there is no NaNs
print (df.shape)
```

```python
np.sum(pd.isna(df))
```

```
(506, 13)
```

Out[2]:
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
lstat      0
medv       0
dtype: int64
```

From above, we can see that there are no missing data and there are 506 observations with 13 features. In the snapshot of the dataframe below and the feature descriptions, we can also see that the unit of observations is by town. However, there are only approximately 141 cities and towns in the Greater Boston area, so we would need clarification on the unit of observation.

In [3]:
```python
df.head()
```

Out[3]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2

Methodology

1. **Data Preprocessing and Exploration**
   - **Summary Statistics**: We began by generating summary statistics of the dataset to understand the distributions and key characteristics of the features using `describe()` method.
   - **Train-Test Split**: The dataset was split into training (80%) and testing (20%) sets using `train_test_split` to ensure that model performance could be evaluated properly and to avoid overfitting.
   - **Scaling**: We scaled the numerical features using `StandardScaler` to normalize the data. This ensures that features with large numerical ranges don't dominate the model.
2. **Modeling**
   - **Linear Regression Model**: We employed a linear regression model to predict housing prices. This was chosen due to the continuous nature of the target variable (`medv`), and the assumption that relationships between predictors and the target variable may be approximately linear.
   - **Model Fitting**: The model was trained using the scaled features from the training set.
3. **Evaluation**
   - We used multiple performance metrics to assess the model, including:
     - **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in predictions.
     - **Root Mean Squared Error (RMSE)**: The square root of MSE, providing a more interpretable metric in terms of the unit of the target variable.
     - **R-squared (R2)**: Indicates the proportion of variance in the dependent variable that is predictable from the independent variables.

4. **Visualization**

- **Predicted vs. Actual Plot**: A scatterplot was created to visualize the relationship between the predicted and actual median house values. A red dashed line at 45 degrees indicates perfect predictions.
- **Residual Plot**: A histogram of residuals (the difference between the actual and predicted values) was plotted to assess the distribution of prediction errors and check for any patterns that might indicate model inadequacy.

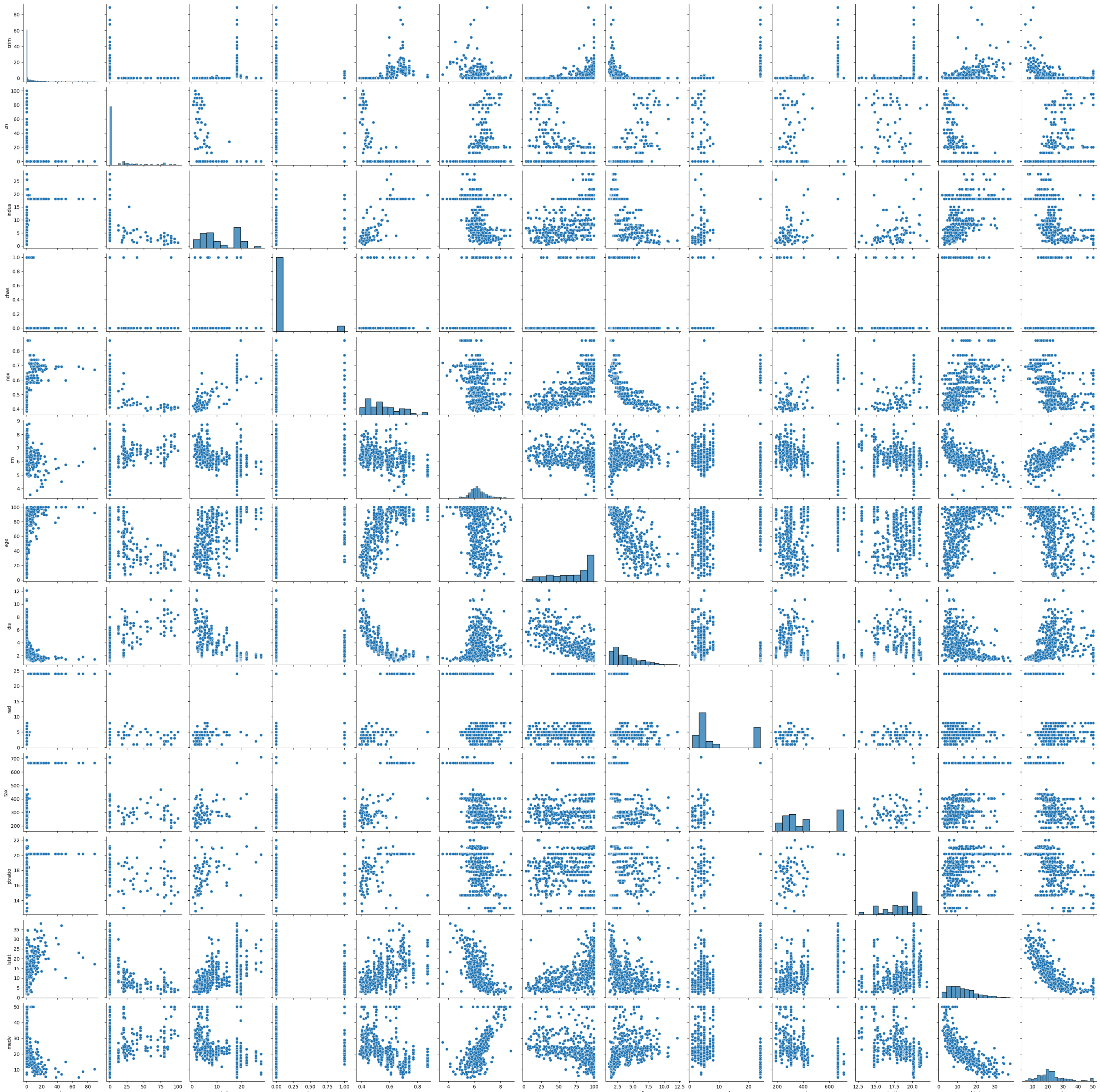
In [4]:
```
df.describe()
```

Out[4]:

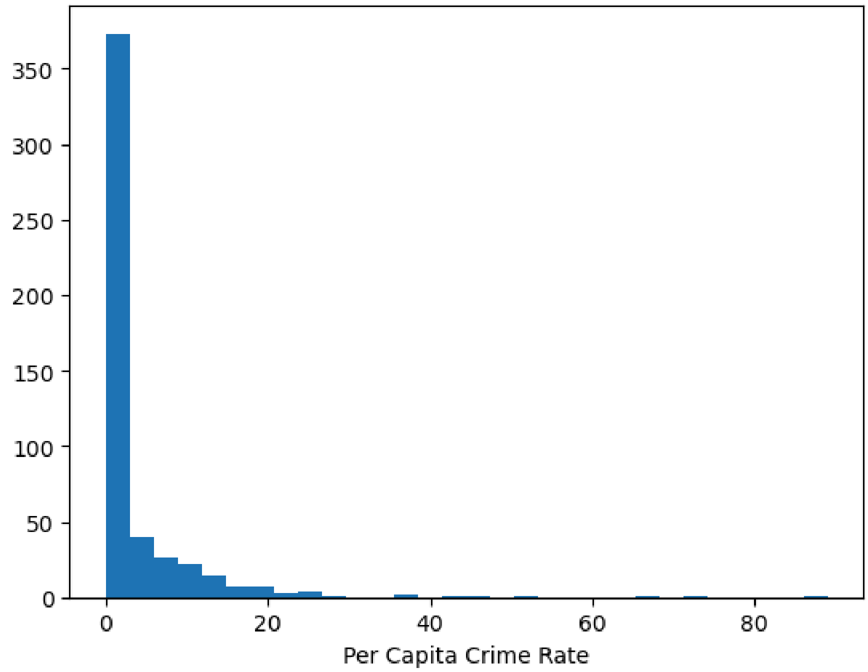
	crim	zn	indus	chas	nox	rm	age	dis	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	2
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	2

In [5]:
```
sns.pairplot(df)
```

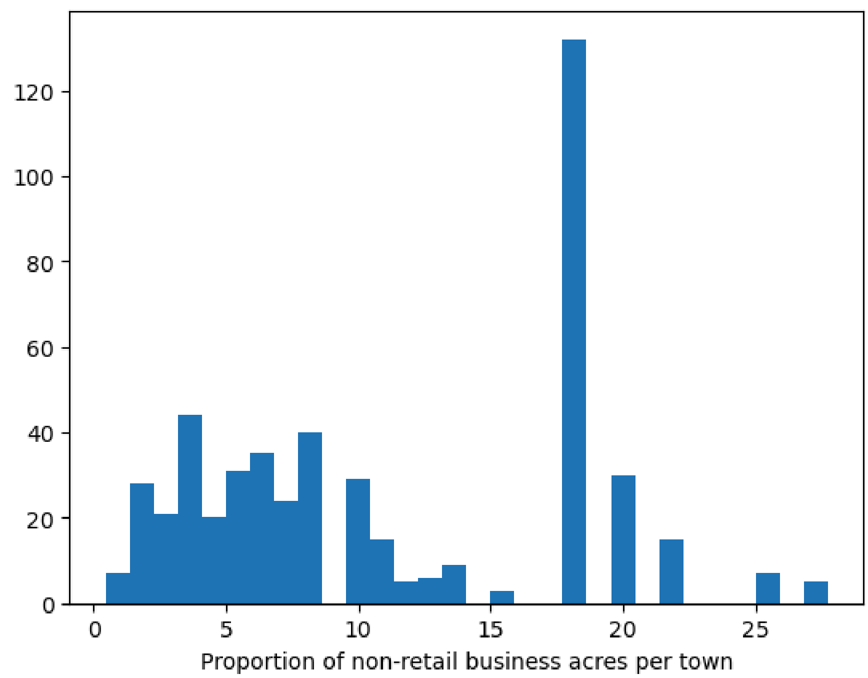
Out[5]:
```
<seaborn.axisgrid.PairGrid at 0x169566550>
```



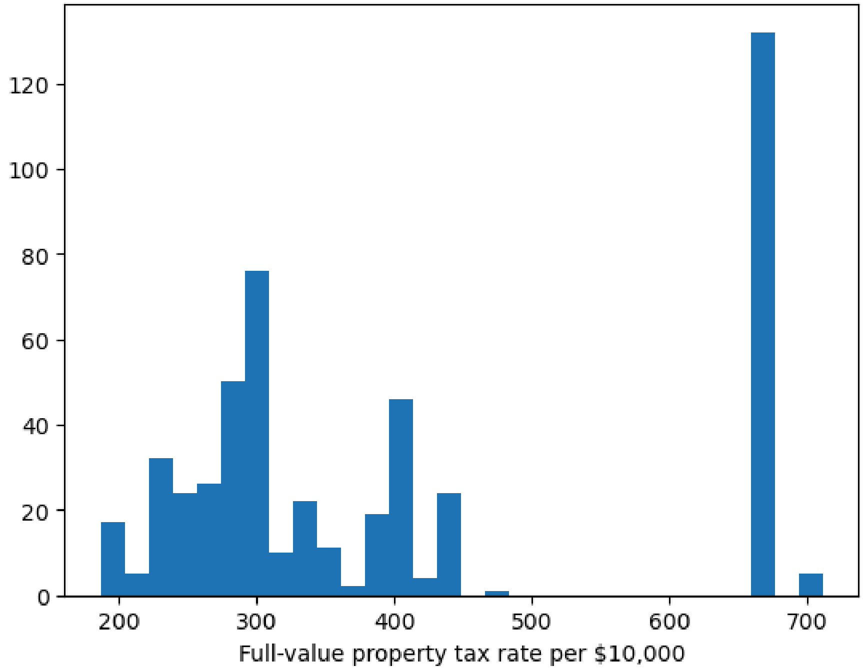
In [6]:
```
plt.hist(df["crim"], bins=30)
plt.xlabel("Per Capita Crime Rate")
plt.show()
```



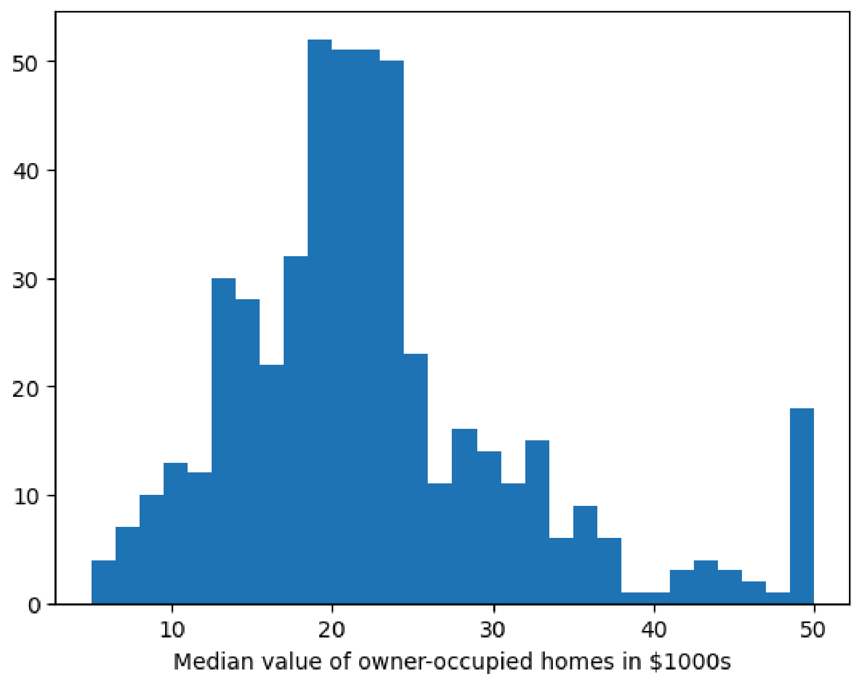
In [7]:
```
plt.hist(df["indus"], bins=30)
plt.xlabel("Proportion of non-retail business acres per town")
plt.show()
```



In [8]:
```
plt.hist(df["tax"], bins=30)
plt.xlabel("Full-value property tax rate per $10,000")
plt.show()
```



In [9]:
```
plt.hist(df["medv"], bins=30)
plt.xlabel("Median value of owner-occupied homes in $1000s")
plt.show()
```



Ying addition -- see what we want to keep

Looking at the histograms of the variables, we can see a right skewed distribution for factors including crim, zn, dis, and lstat and a left skewed distribution for factors including age and ptratio. This indicate a potential need to tranform these features to satisfy assumptions of the linear regression.

Prior to building models, we can further explore relationships between the features and the y-variable, medv.

In [10]:
```
df.corr()
```

Out[10]:

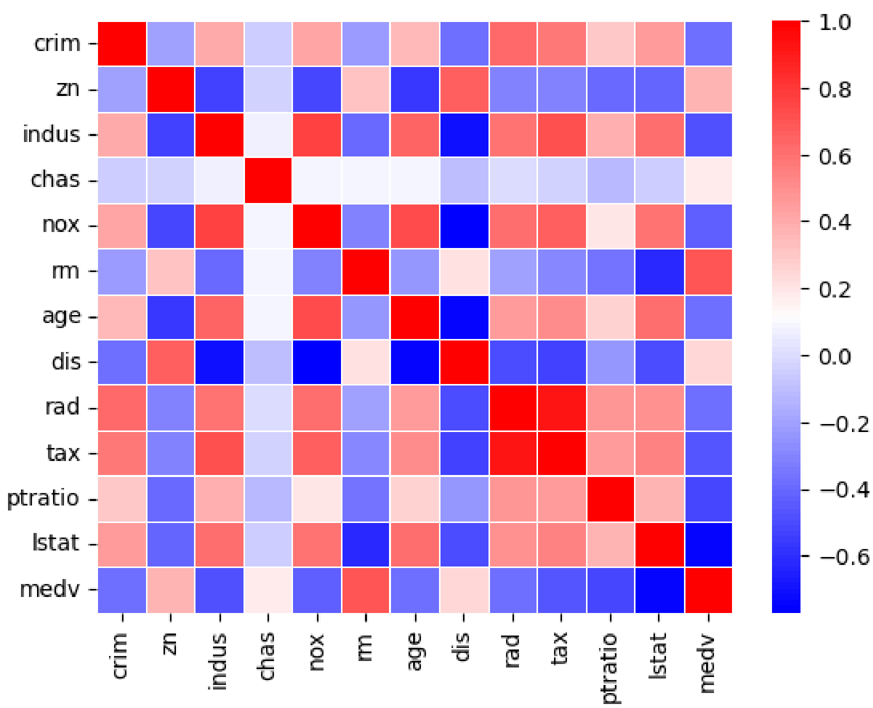
	crim	zn	indus	chas	nox	rm	age	dis	rad
crim	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505
zn	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948
indus	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129
chas	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368
nox	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441
rm	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847
age	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022
dis	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588
rad	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000
tax	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228
ptratio	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741
lstat	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676
medv	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.381626

Looking at the last row of scatterplots which shows the other features against the median value of owner-occupied homes in $1000s, we can see that factors such as crim, zn, nox, rm, rad, tax, ptratio, and lstat have correlations with median values of homes. Since our goal is to assist issues of housing affordability, a beneficial step would be identifying the characteristics of areas that have high prices and comparing to the characteristics of areas with low prices.

In [11]:

```
corr_matrix = df.corr()
sns.heatmap(corr_matrix, cmap='bwr', linewidth=.5)
```

Out[11]: <Axes: >



In [12]:

```
high_prices = df[(df["medv"]>df["medv"].quantile(0.80))]
high_prices.describe()
```

Out [12]:

	crim	zn	indus	chas	nox	rm	age	dis	
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.00
mean	0.631501	28.782178	5.564554	0.118812	0.490435	7.192990	54.821782	4.394886	5.66
std	1.595083	32.860875	5.117536	0.325181	0.080829	0.622245	28.146075	2.154758	5.00
min	0.009060	0.000000	0.460000	0.000000	0.394000	4.970000	6.800000	1.129600	1.00
25%	0.037680	0.000000	2.460000	0.000000	0.433000	6.812000	31.900000	2.847000	3.00
50%	0.081870	20.000000	3.780000	0.000000	0.464000	7.135000	53.600000	3.838400	5.00
75%	0.511830	45.000000	6.200000	0.000000	0.507000	7.520000	80.800000	5.960400	5.00
max	9.232300	100.000000	19.580000	1.000000	0.668000	8.725000	100.000000	12.126500	24.00

In [13]:

```
low_prices = df[(df["medv"]<df["medv"].quantile(0.20))]
low_prices.describe()
```

Out [13]:

	crim	zn	indus	chas	nox	rm	age	dis	
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000
mean	12.853867	0.123762	17.445842	0.009901	0.669901	5.882832	93.914851	2.119956	18.87
std	15.116099	1.243796	4.070455	0.099504	0.079363	0.622606	7.212245	0.952734	8.76
min	0.047410	0.000000	6.910000	0.000000	0.448000	4.138000	59.700000	1.137000	1.000
25%	3.321050	0.000000	18.100000	0.000000	0.614000	5.520000	92.400000	1.580400	5.000
50%	9.329090	0.000000	18.100000	0.000000	0.693000	5.957000	96.000000	1.822600	24.000
75%	15.177200	0.000000	18.100000	0.000000	0.713000	6.343000	98.900000	2.198000	24.000
max	88.976200	12.500000	27.740000	1.000000	0.871000	7.313000	100.000000	6.346700	24.000

Extracting the town data for the towns with the top 20th percentile median value and the towns with the lowest 20th percentile median value, we can observe that compared to towns with the highest housing value, towns with cheapest housing value have:

- higher crime rate
- lower proportion of residential land
- higher proportion of non-retail business (factories)
- lower access to transportion and scenary as demonstrated by the lower chas and rad values
- higher nitric oxide (indicator of air pollution)
- older housing and higher property tax
- higher percentage of lower-status population.

These observations are not suprising, for town prices are impacted by safety levels, health risk levels, access to transportation, and amount of resources circulating in the neighborhood.

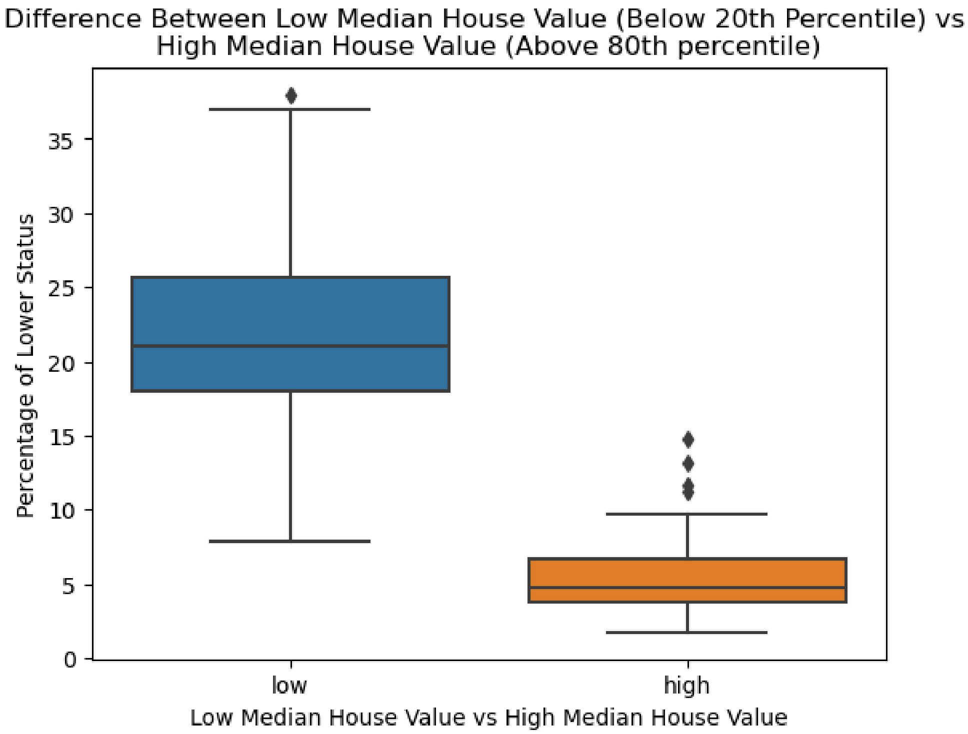
Based on the correlation matrix, we chose top 7 variables with correlation values that has highest magnitude.

In [14]:

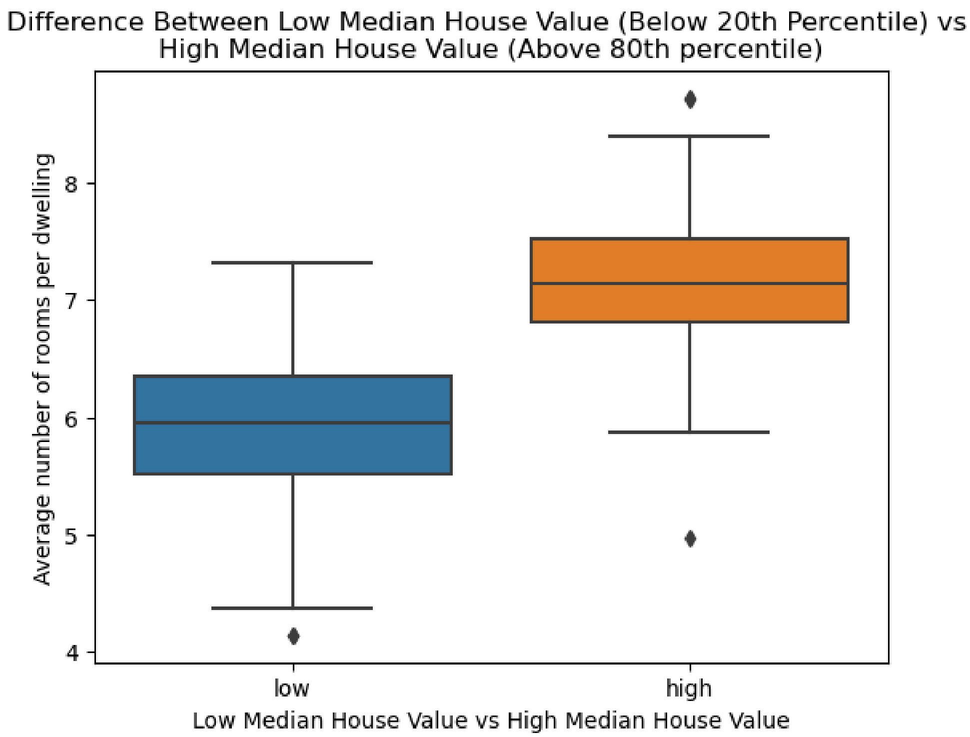
```
low_prices_df = pd.DataFrame(low_prices)
low_prices_df.reset_index(inplace=True)
low_prices_df["price_cat"] = "low"

high_prices_df = pd.DataFrame(high_prices)
high_prices_df.reset_index(inplace=True)
```

```
high_prices_df["price_cat"] = "high"
combined_df = pd.concat([low_prices_df, high_prices_df])
```

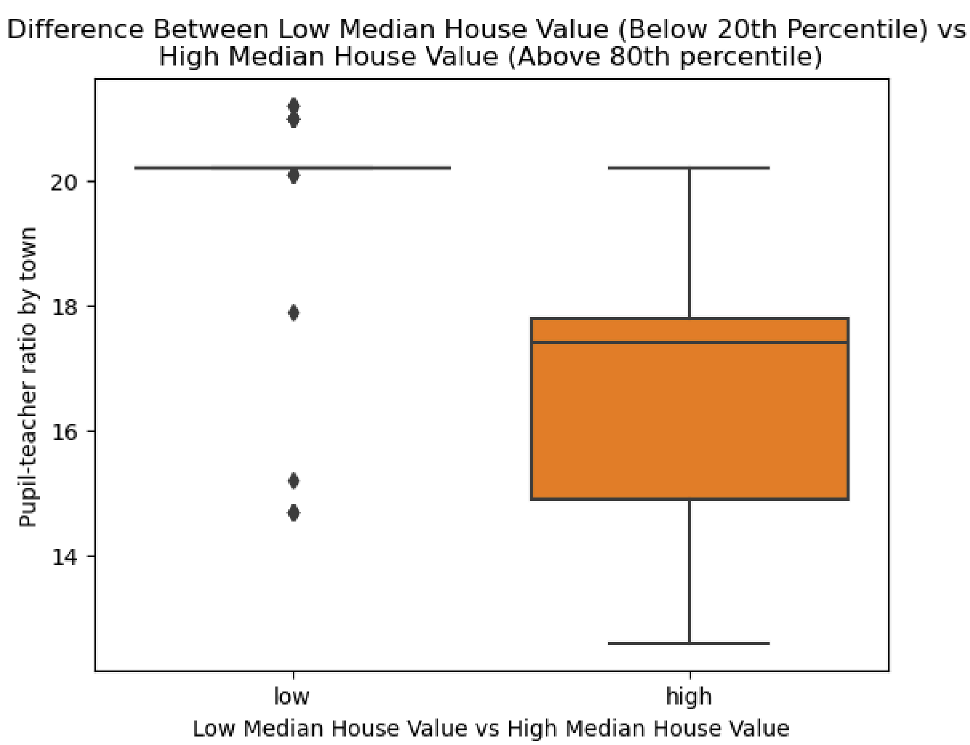
In [15]:
```
sns.boxplot(x="price_cat", y="lstat", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Percentage of Lower Status")
plt.show()
```



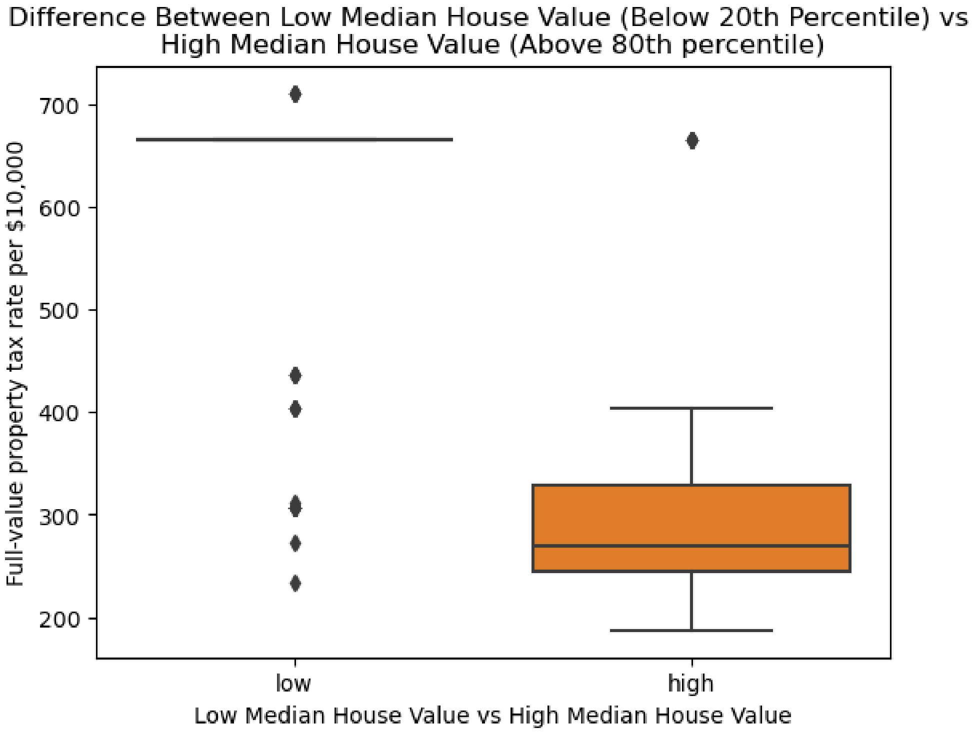
In [16]:
```
sns.boxplot(x="price_cat", y="rm", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Average number of rooms per dwelling")
plt.show()
```



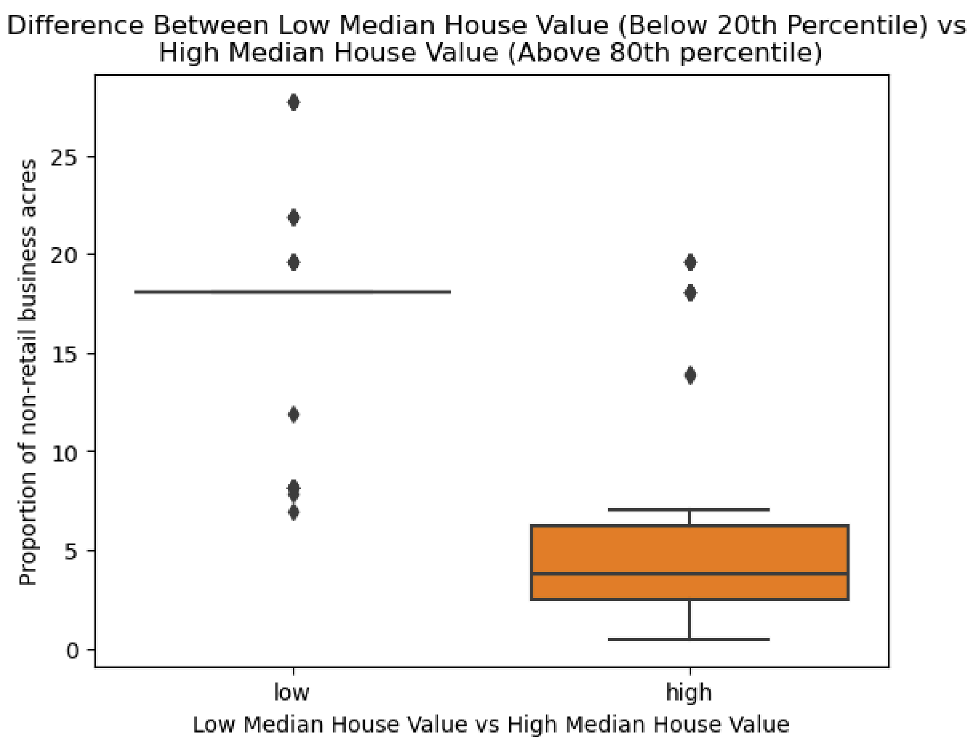
In [17]:
```
sns.boxplot(x="price_cat", y="ptratio", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Pupil-teacher ratio by town")
plt.show()
```



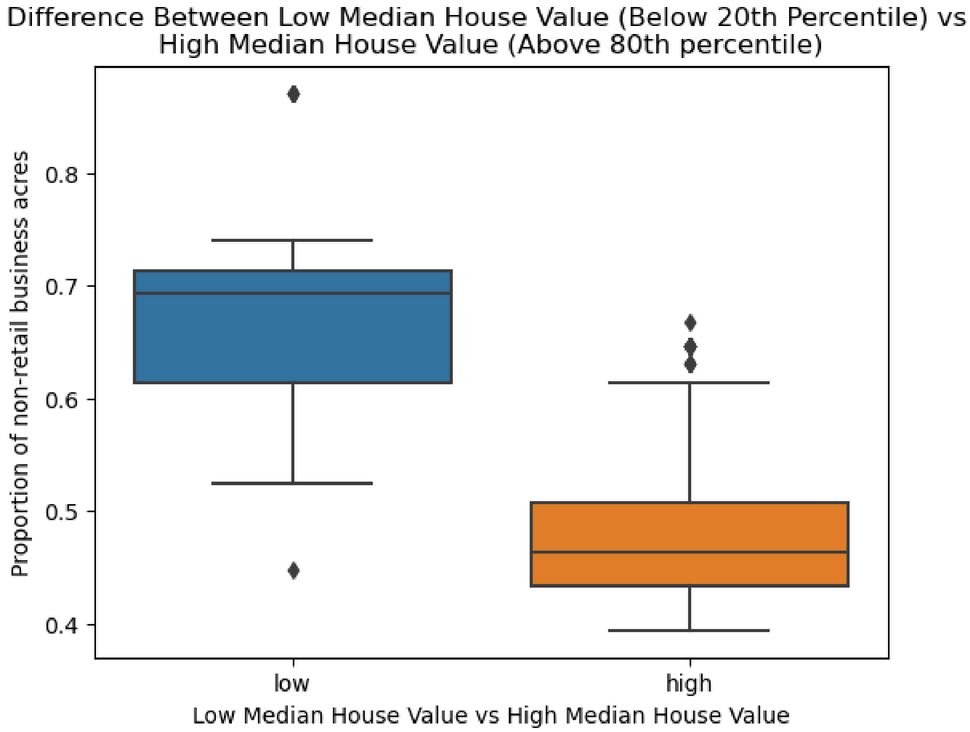
In [18]:
```
sns.boxplot(x="price_cat", y="tax", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Full-value property tax rate per $10,000")
plt.show()
```



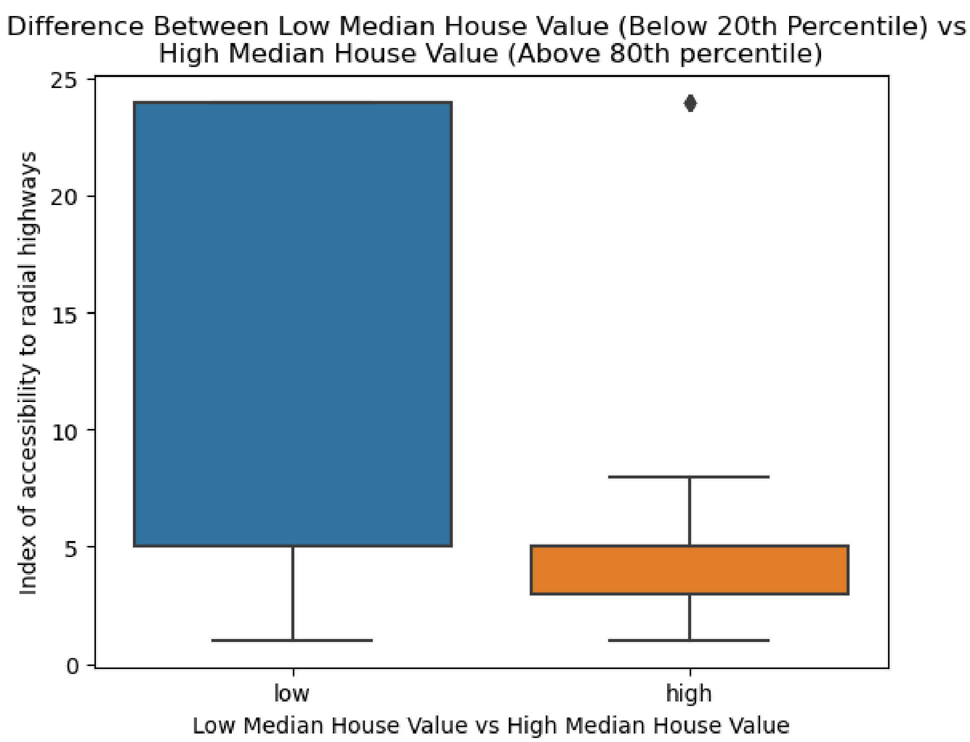
In [19]:
```
sns.boxplot(x="price_cat", y="indus", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Proportion of non-retail business acres")
plt.show()
```



In [20]:
```
sns.boxplot(x="price_cat", y="nox", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Proportion of non-retail business acres")
plt.show()
```



In [21]:
```
sns.boxplot(x="price_cat", y="rad", data=combined_df)
plt.title("Difference Between Low Median House Value (Below 20th Percentile) vs \n" \
"High Median House Value (Above 80th percentile)")
plt.xlabel("Low Median House Value vs High Median House Value")
plt.ylabel("Index of accessibility to radial highways")
plt.show()
```



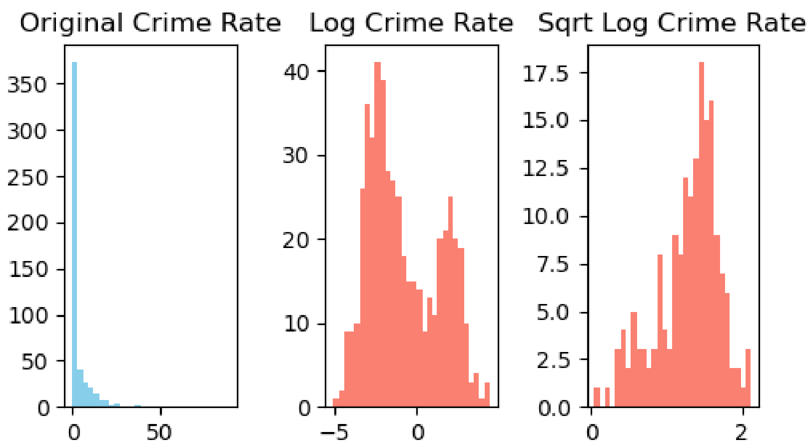
After exploring some characteristics of towns with low median house values and towns with high median house values, we will now try to measure the significance or impact of these factors.

As mentioned in the analysis of the pair plot, these factors are skewed and will need some tranformation to ensure an approximately normal distribution.

In particular, we can see a right skewed distribution for factors including crim, zn, dis, and lstat and a left skewed distribution for factors including age and ptratio. This indicate a potential need to tranform these features to satisfy assumptions of the linear regression.

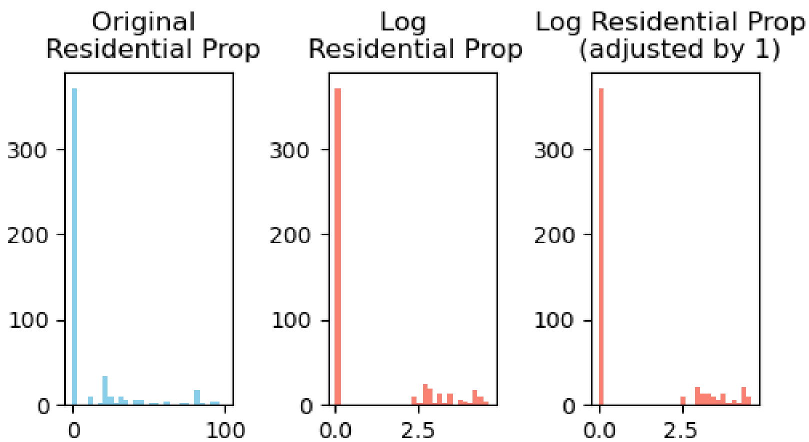
In [22]:
```
df_transformed = df.copy()

df_transformed["logCrim"] = np.log(df_transformed["crim"])
df_transformed["sqrtCrim"] = np.sqrt(df_transformed["logCrim"])
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(5, 3))
axes[0].hist(df_transformed["crim"], bins=30, color='skyblue')
axes[0].set_title("Original Crime Rate")
axes[1].hist(df_transformed["logCrim"], bins=30, color='salmon')
axes[1].set_title("Log Crime Rate")
axes[2].hist(df_transformed["sqrtCrim"], bins=30, color='salmon')
axes[2].set_title("Sqrt Log Crime Rate")
fig.tight_layout()

plt.show()
```

```
/Users/yzhao/anaconda3/envs/info2950/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

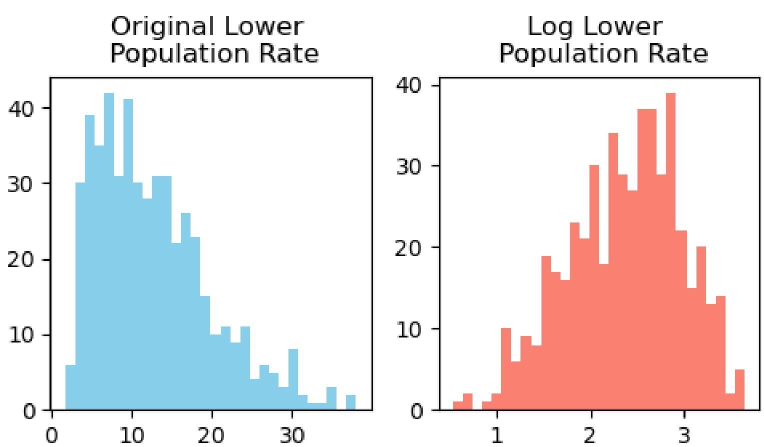


Since there are a lot of 0 values for zn, it will return -infinity when the log transformation is taken. As a result, we tried using the square root transformation and log tranformation after adjusting the values by 1, but the distribution is not normal after the transformation.

In [23]:
```
df_transformed["logZn"] = np.log(df_transformed["zn"]+1)
df_transformed["sqrtZn"] = np.cbrt(df_transformed["zn"])
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(5, 3))
axes[0].hist(df_transformed["zn"], bins=30, color='skyblue')
axes[0].set_title("Original \n Residential Prop")
axes[1].hist(df_transformed["sqrtZn"], bins=30, color='salmon')
axes[1].set_title("Log  \n Residential Prop")
axes[2].hist(df_transformed["logZn"], bins=30, color='salmon')
axes[2].set_title("Log Residential Prop \n (adjusted by 1)")
fig.tight_layout()

plt.show()
```



In [24]:
```
df_transformed["logLstat"] = np.log(df_transformed["lstat"])
#df_transformed["sqrtCrim"] = np.sqrt(df_transformed["logCrim"])
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(5, 3))
axes[0].hist(df_transformed["lstat"], bins=30, color='skyblue')
axes[0].set_title("Original Lower \n Population Rate")
axes[1].hist(df_transformed["logLstat"], bins=30, color='salmon')
axes[1].set_title("Log Lower \n Population Rate")
fig.tight_layout()

plt.show()
```



The log tranformation for this feature did not seem to assist with normality.

In [25]:
```
# Split data into training and testing sets
X = df.drop('medv', axis=1)
y = df['medv']

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

In [26]:
```
# Create a pipeline with scaling and regression model
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LinearRegression())
])

# Define cross-validation strategy
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Define scoring metrics
scoring = {
    'r2': 'r2',
    'mse': 'neg_mean_squared_error',
```

```
    'mae': 'neg_mean_absolute_error'
}
```

In [27]:
```
# Perform cross-validation
cv_results = cross_validate(pipeline, X_train, y_train, cv=kf, scoring=scoring, return_train_score=
```

Results

- **Model Performance**: The model produced the following evaluation metrics:
  - **MAE**: Mean Absolute Error
  - **RMSE**: Root Mean Squared Error
  - **R²**: R-squared (Proportion of variance explained)
- **Key Insights from Visualizations**:
  - The predicted vs. actual plot showed that the linear regression model generally performs well for most of the data points. However, there are some outliers where the predictions are less accurate.
  - The residual plot does not display a random distribution, therefore, we believe that the normality assumption of the linear regression is not fulfilled.

In [28]:
```
# Extract mean scores from cross-validation
mean_r2 = np.mean(cv_results['test_r2'])
mean_rmse = sqrt(-np.mean(cv_results['test_mse']))  # Convert to positive
mean_mae = -np.mean(cv_results['test_mae'])  # Convert to positive

print(f"Cross-validated R²: {mean_r2:.2f}")
print(f"Cross-validated RMSE: {mean_rmse:.2f}")
print(f"Cross-validated MAE: {mean_mae:.2f}")
```

```
Cross-validated R²: 0.71
Cross-validated RMSE: 5.03
Cross-validated MAE: 3.56
```

In [29]:
```
# Train model on full training set
pipeline.fit(X_train, y_train)

# Make predictions on test set
y_pred = pipeline.predict(X_test)

# Evaluate on test set
test_r2 = r2_score(y_test, y_pred)
test_rmse = sqrt(mean_squared_error(y_test, y_pred))
test_mae = mean_absolute_error(y_test, y_pred)

print(f"Test R²: {test_r2:.2f}")
print(f"Test RMSE: {test_rmse:.2f}")
print(f"Test MAE: {test_mae:.2f}")
```

```
Test R²: 0.69
Test RMSE: 4.77
Test MAE: 3.11
```

Feature Selection using Forward Feature Selection with Various Model Evaluation Metrics

In [30]:
```
sc = StandardScaler()

def training_test_result_MSE(df):
    X = df.drop(columns=["medv"]).values
    X = sc.fit_transform(X)
    X = sm.add_constant(X)
    y = df['medv'].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=123)

    model_sc = sm.OLS(y_train, X_train).fit()
    outsample_predictions_sc = model_sc.predict(X_test)
    test_metrics = mean_squared_error(y_test, outsample_predictions_sc)
```

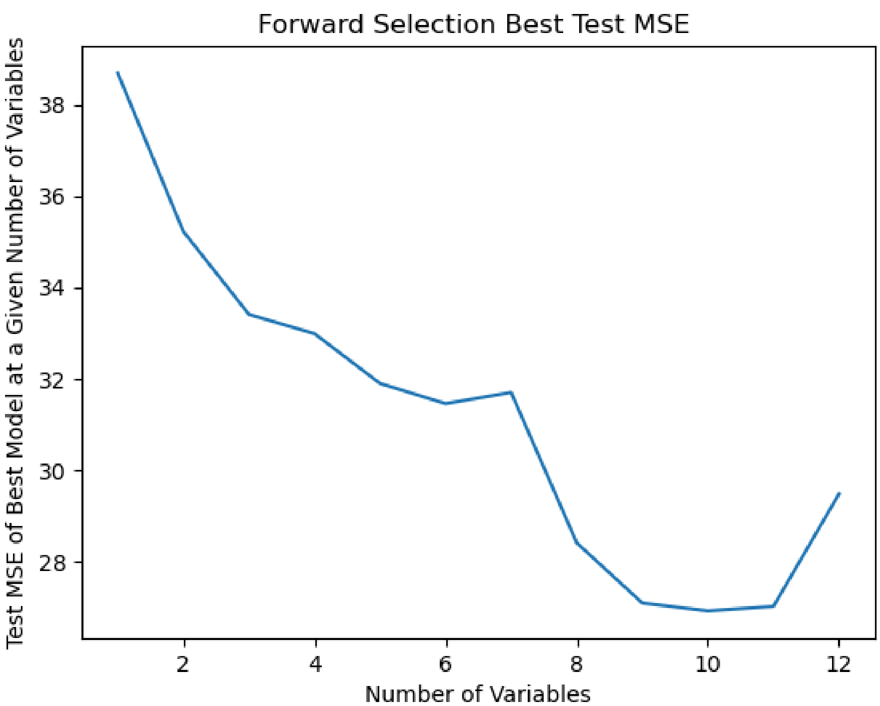
```
    return (test_metrics)

def training_test_result_R2(df):
    X = df.drop(columns=["medv"]).values
    X = sc.fit_transform(X)
    X = sm.add_constant(X)
    y = df['medv'].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=123)

    model_sc = sm.OLS(y_train, X_train).fit()
    outsample_predictions_sc = model_sc.predict(X_test)
    r2 = r2_score(y_test, outsample_predictions_sc)
    n = len(y_test)
    p = X_test.shape[1] - 1
    adj_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
    return (adj_r2)

def model_eval_AIC(df):
    X = df.drop(columns=["medv"]).values
    X = sc.fit_transform(X)
    X = sm.add_constant(X)
    y = df['medv'].values
    model_sc = sm.OLS(y, X).fit()
    test_metrics = model_sc.aic
    return (test_metrics)

def model_eval_BIC(df):
    X = df.drop(columns=["medv"]).values
    X = sc.fit_transform(X)
    X = sm.add_constant(X)
    y = df['medv'].values
    model_sc = sm.OLS(y, X).fit()
    test_metrics = model_sc.bic
    return (test_metrics)

def forwardSelection (df, function):
    """To be used for metrics where having a smaller value is preferred"""
    all_features = list(df.drop(columns=['medv']).columns)
    selected_features = []
    best_test_mse_list = []
    for numvar in range (1,len(all_features)+1):
        temp_best_test_mse = 1000000
        for feature in all_features:
            if (feature in selected_features):
                next;
            else:
                temp_features = selected_features +  [feature]
                temp_test_mse = function(df[temp_features + ['medv']])
                if temp_test_mse < temp_best_test_mse:
                    temp_best_test_mse = temp_test_mse
                    temp_best_feature = feature
        # add the best feature for that given round (considering models of i features)
        selected_features.append(temp_best_feature)
        best_test_mse_list.append(temp_best_test_mse)
        # print (f"For model with {numvar} variables, the next best feature is {temp_best_feature}"
    print (best_test_mse_list)
    print (selected_features)
    return (best_test_mse_list)

def forwardSelection_R2 (df, function):
    """To be used for metrics where having a larger value is preferred"""
    all_features = list(df.drop(columns=['medv']).columns)
    selected_features = []
    best_test_mse_list = []
    for numvar in range (1,len(all_features)+1):
        temp_best_test_mse = 0
        for feature in all_features:
            if (feature in selected_features):
                next;
            else:
```

```
                temp_features = selected_features +  [feature]
                temp_test_mse = function(df[temp_features + ['medv']])
                if temp_test_mse > temp_best_test_mse:
                    temp_best_test_mse = temp_test_mse
                    temp_best_feature = feature
        # add the best feature for that given round (considering models of i features)
        selected_features.append(temp_best_feature)
        best_test_mse_list.append(temp_best_test_mse)
        # print (f"For model with {numvar} variables, the next best feature is {temp_best_feature}"
    print (best_test_mse_list)
    print (selected_features)
    return (best_test_mse_list)
```

In [31]:
```
test_score = forwardSelection(df, training_test_result_MSE)
test_results = pd.DataFrame({"numVar":np.arange(1, len(test_score) + 1), "testMSE":test_score})
plt.plot(test_results["numVar"], test_results["testMSE"])
plt.xlabel("Number of Variables")
plt.ylabel("Test MSE of Best Model at a Given Number of Variables")
plt.title("Forward Selection Best Test MSE ")
```

```
[38.68755662089717, 35.22536624931015, 33.40659016862141, 32.98632693603778, 31.895854616620444, 3
1.45497569145293, 31.69884051483146, 28.410251987691787, 27.09292852261487, 26.92006231393658, 27.0
18176606001195, 29.477285596923664]
['lstat', 'ptratio', 'chas', 'rad', 'tax', 'crim', 'nox', 'dis', 'zn', 'age', 'indus', 'rm']
```

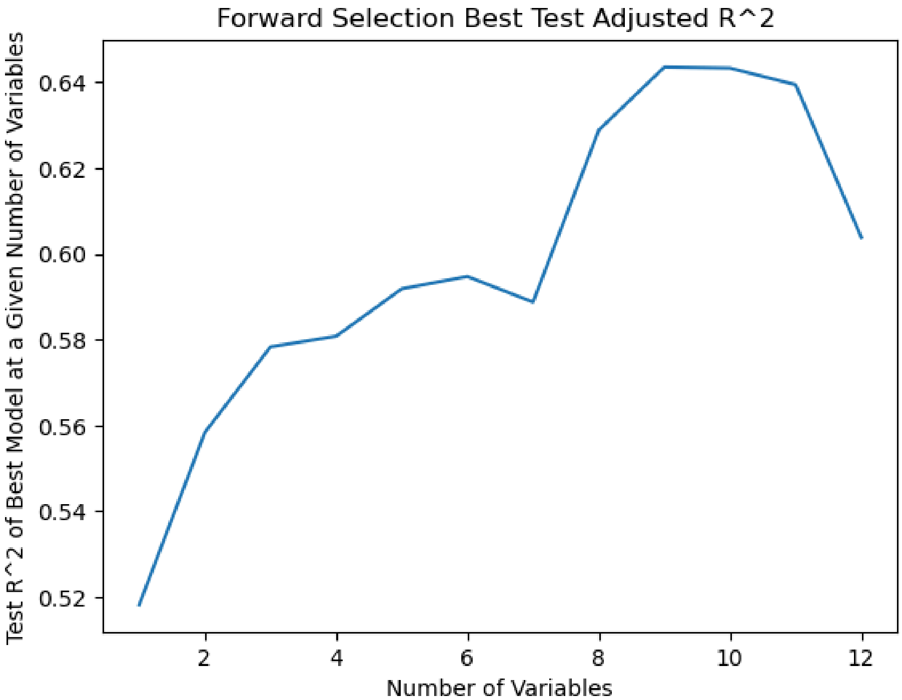
Out[31]: Text(0.5, 1.0, 'Forward Selection Best Test MSE ')



In [32]:
```
test_score = forwardSelection_R2(df, training_test_result_R2)
test_results = pd.DataFrame({"numVar":np.arange(1, len(test_score) + 1), "TestR2":test_score})
plt.plot(test_results["numVar"], test_results["TestR2"])
plt.xlabel("Number of Variables")
plt.ylabel("Test R^2 of Best Model at a Given Number of Variables")
plt.title("Forward Selection Best Test Adjusted R^2 ")
```

```
[0.5181690813279849, 0.5583442684503259, 0.5783180070825542, 0.5807903742071965, 0.591872358702366
5, 0.5947379158074502, 0.5887598521081245, 0.628846406075602, 0.6435634521603673, 0.643325903222437
4, 0.6394689923222951, 0.6038247771460132]
['lstat', 'ptratio', 'chas', 'rad', 'tax', 'crim', 'nox', 'dis', 'zn', 'age', 'indus', 'rm']
```

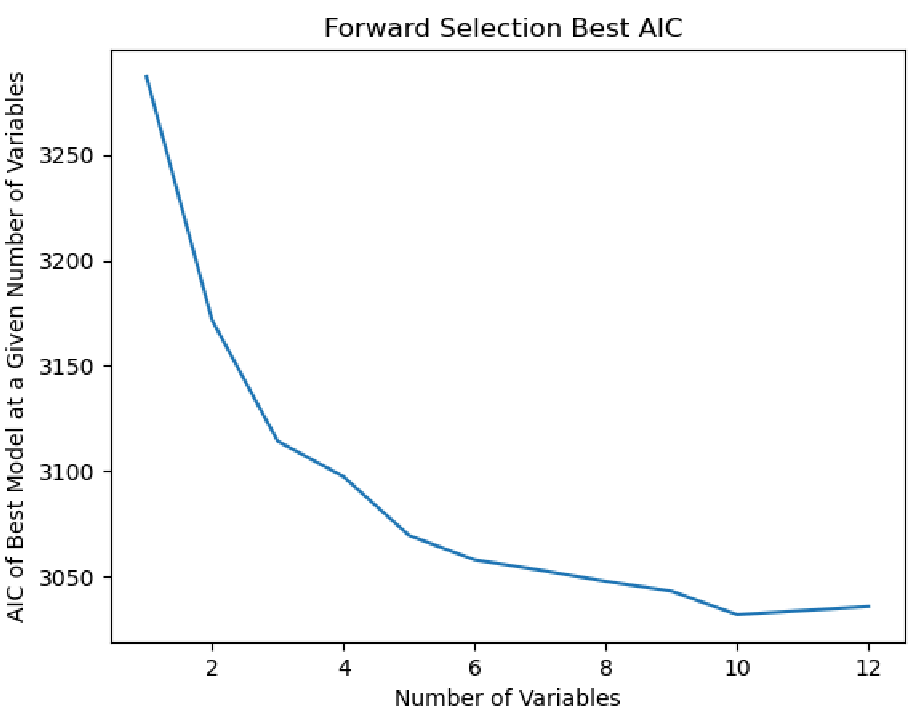
Out[32]: Text(0.5, 1.0, 'Forward Selection Best Test Adjusted R^2 ')



In [33]:
```
test_score = forwardSelection(df, model_eval_AIC)
test_results = pd.DataFrame({"numVar":np.arange(1, len(test_score) + 1), "AIC":test_score})
plt.plot(test_results["numVar"], test_results["AIC"])
plt.xlabel("Number of Variables")
plt.ylabel("AIC of Best Model at a Given Number of Variables")
plt.title("Forward Selection Best AIC ")
```

```
[3286.974956900157, 3171.5423142992013, 3114.0972674193326, 3097.359044862759, 3069.4386331672167,
3057.9390497191152, 3053.040329856005, 3047.7679231226593, 3043.0803164878553, 3031.9440576705483,
3033.86888104232, 3035.8206794796506]
['lstat', 'rm', 'ptratio', 'dis', 'nox', 'chas', 'zn', 'crim', 'rad', 'tax', 'age', 'indus']
```

Out[33]:
```
Text(0.5, 1.0, 'Forward Selection Best AIC ')
```



In [34]:
```
original_columns = ['lstat', 'ptratio', 'chas', 'rad', 'tax', 'crim']
X_reduced = df[original_columns]
```

```python
y = df['medv'].values

X_train, X_test, y_train_r, y_test_r = train_test_split(X_reduced, y, test_size=0.2, random_state=4

X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)

# Re-wrap into DataFrames with column names
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=original_columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=original_columns)

# --- STEP 4: Add constant term for intercept ---
X_train_scaled_df = sm.add_constant(X_train_scaled_df)
X_test_scaled_df = sm.add_constant(X_test_scaled_df)

# --- STEP 5: Fit model using training data only ---
model = sm.OLS(y_train_r, X_train_scaled_df).fit()

model.summary()
```

Out[34]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.632
Model:	OLS	**Adj. R-squared:**	0.626
Method:	Least Squares	**F-statistic:**	113.6
Date:	Tue, 08 Apr 2025	**Prob (F-statistic):**	5.70e-83
Time:	22:43:35	**Log-Likelihood:**	-1273.2
No. Observations:	404	**AIC:**	2560.
Df Residuals:	397	**BIC:**	2588.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	22.7965	0.284	80.319	0.000	22.239	23.355
lstat	-5.7012	0.345	-16.519	0.000	-6.380	-5.023
ptratio	-2.5189	0.330	-7.634	0.000	-3.168	-1.870
chas	0.9820	0.288	3.405	0.001	0.415	1.549
rad	3.0034	0.737	4.073	0.000	1.554	4.453
tax	-2.3748	0.718	-3.306	0.001	-3.787	-0.963
crim	-0.8898	0.367	-2.426	0.016	-1.611	-0.169

Omnibus:	88.084	**Durbin-Watson:**	2.118
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	163.288
Skew:	1.211	**Prob(JB):**	3.49e-36
Kurtosis:	4.958	**Cond. No.**	6.14

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [35]:
```python
# --- STEP 6: Predict on test data ---
y_pred_reducded = model.predict(X_test_scaled_df)
```

In [36]:
```python
# Evaluate on test set
test_rmse_r = sqrt(mean_squared_error(y_test_r, y_pred_reducded))
```

```
test_mae_r = mean_absolute_error(y_test_r, y_pred_reducded)

print(f"Test RMSE: {test_rmse_r:.2f}")
print(f"Test MAE: {test_mae_r:.2f}")
```

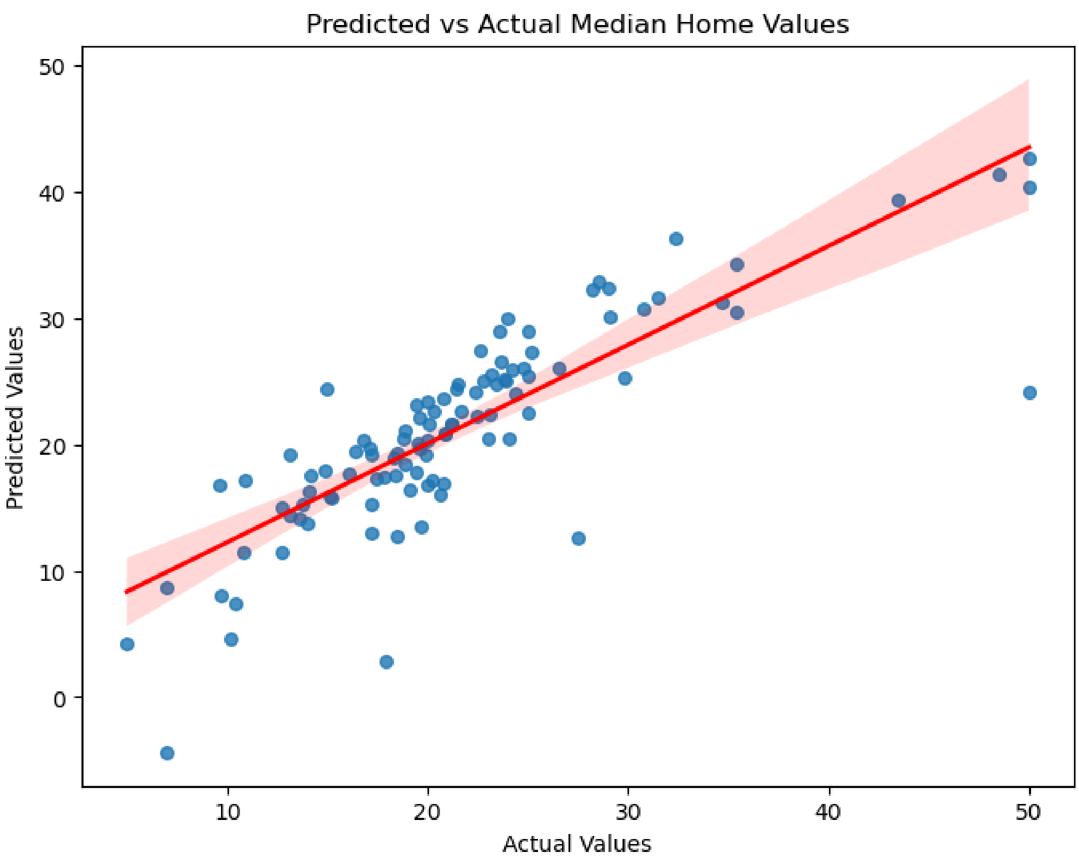
```
Test RMSE: 5.06
Test MAE: 3.40
```

Visualizations

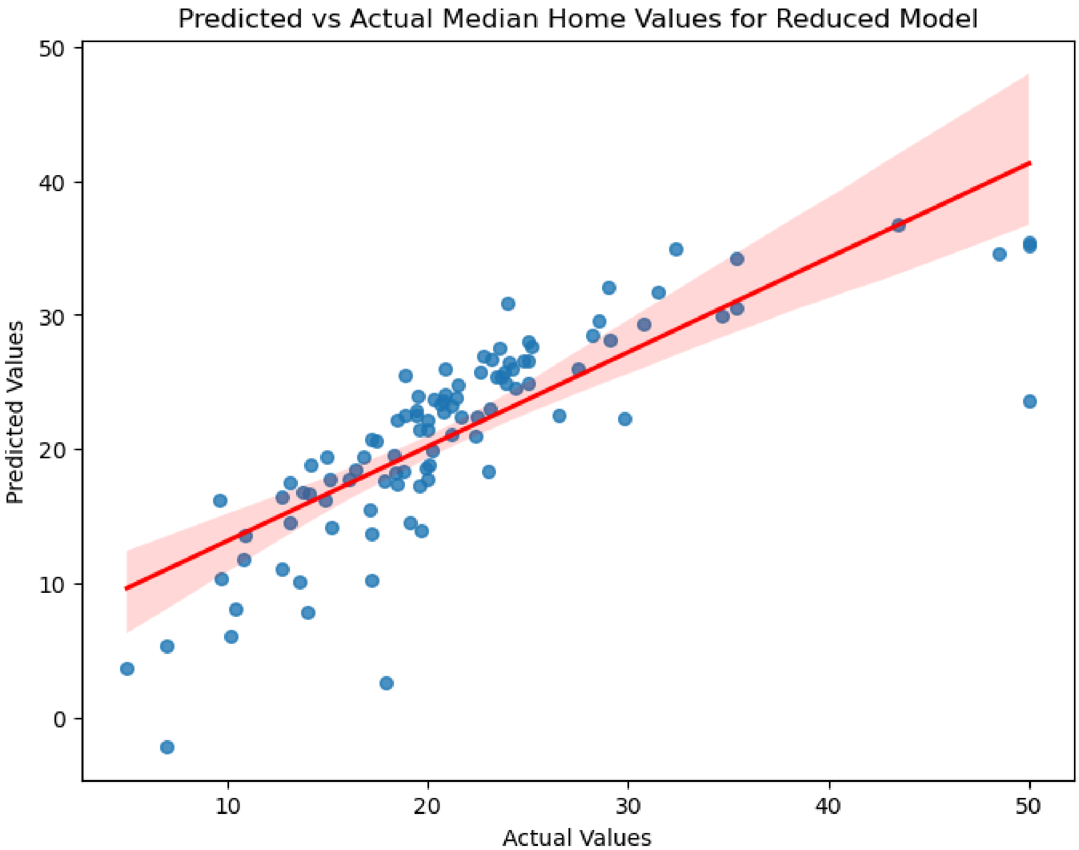
Predicted vs. Actual Plot

A scatter plot showing the relationship between predicted and actual median home values. The red dashed line represents perfect prediction

In [37]:
```
plt.figure(figsize=(8,6))
sns.regplot(x=y_test, y=y_pred, scatter_kws={'s': 30}, line_kws={'color': 'red', 'lw': 2})
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Predicted vs Actual Median Home Values")
plt.show()
```

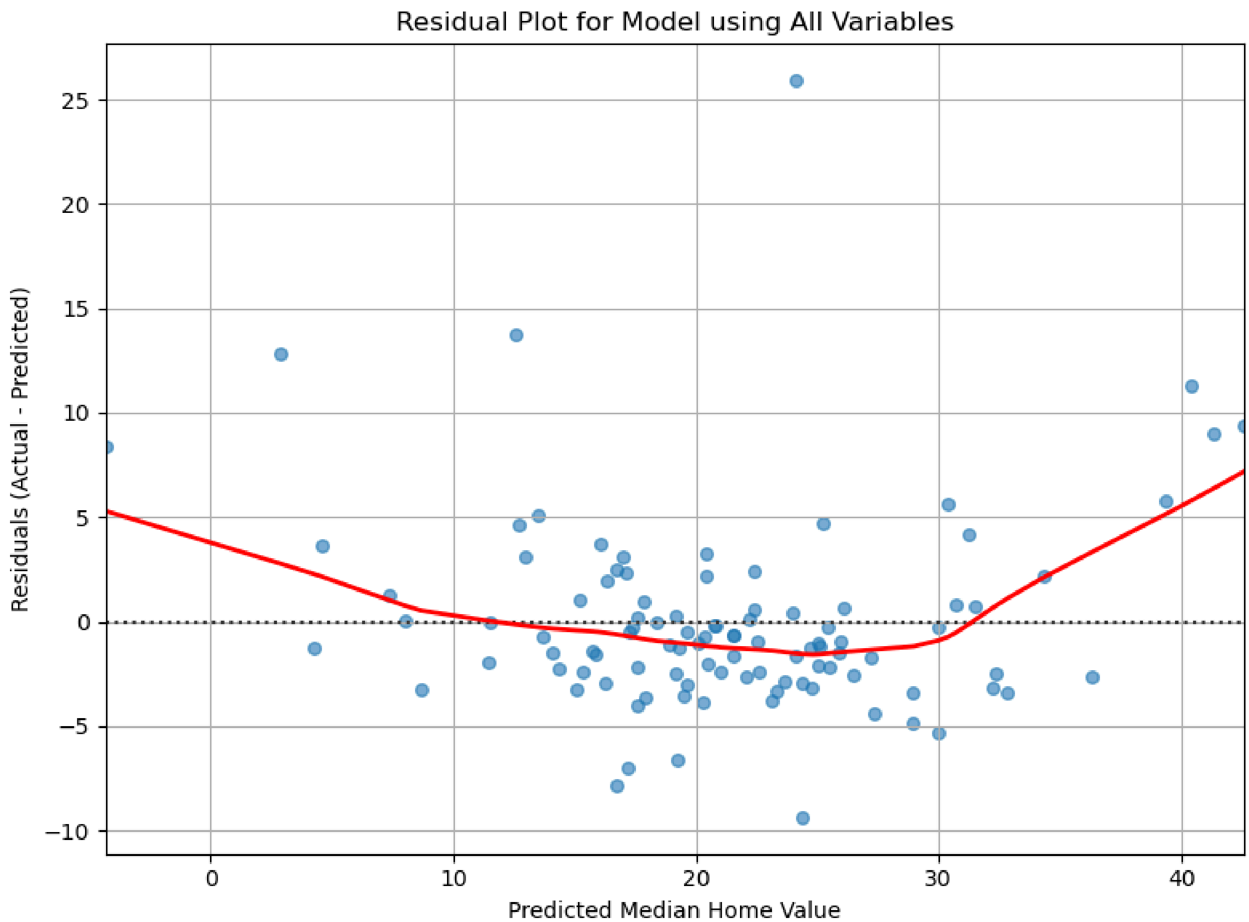


In [38]:
```
plt.figure(figsize=(8,6))
sns.regplot(x= y_test_r , y=y_pred_reducded , scatter_kws={'s': 30}, line_kws={'color': 'red', 'lw'
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Predicted vs Actual Median Home Values for Reduced Model")
plt.show()
```

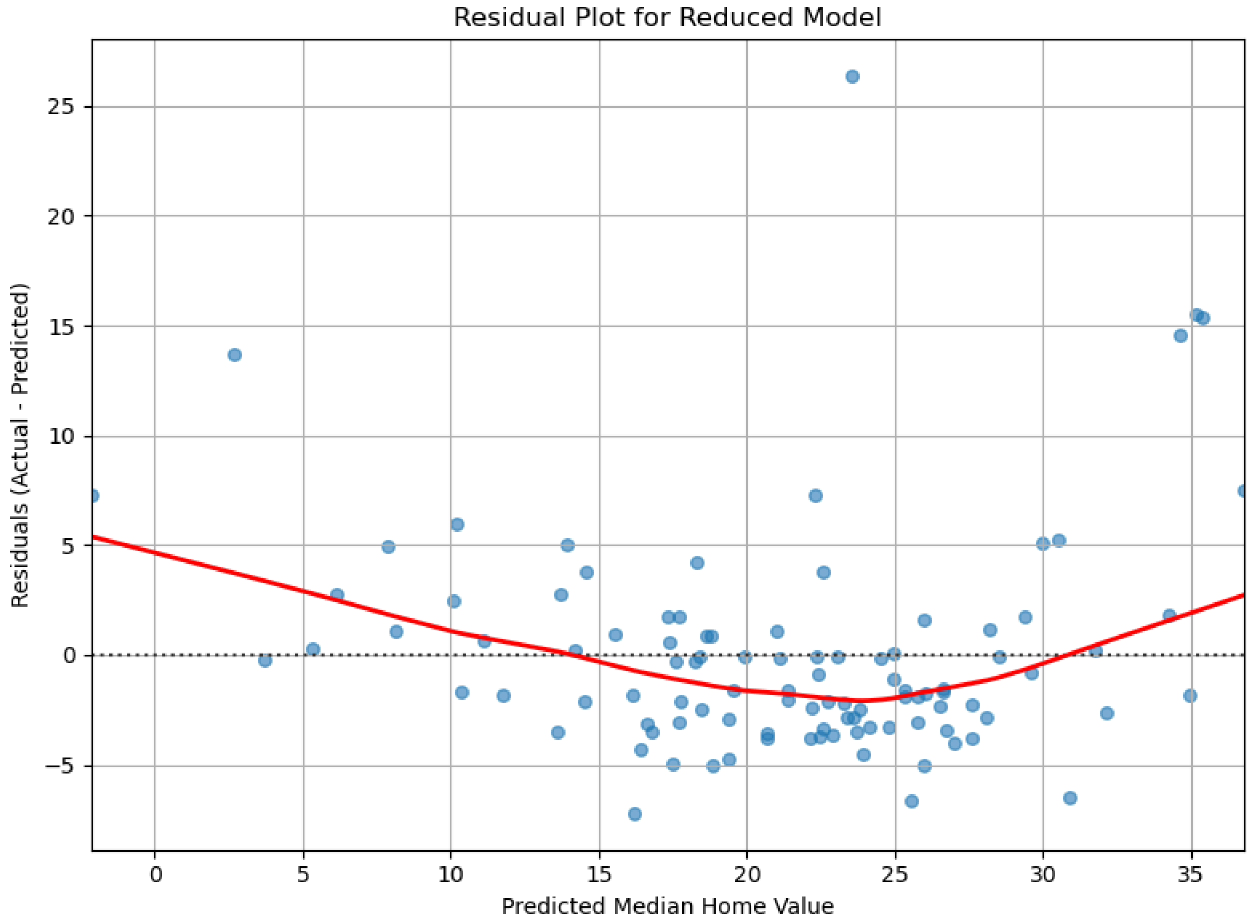


Residual Plot

In [39]:
```
residuals = y_test - y_pred
plt.figure(figsize=(8,6))
sns.residplot(x=y_pred, y=residuals, lowess=True,
              scatter_kws={'alpha': 0.6, 's': 30},
              line_kws={'color': 'red', 'lw': 2})
plt.xlabel("Predicted Median Home Value")
plt.ylabel("Residuals (Actual - Predicted)")
plt.title("Residual Plot for Model using All Variables")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [40]:
```
residuals_r = y_test_r - y_pred_reducded
plt.figure(figsize=(8,6))
sns.residplot(x=y_pred_reducded, y=residuals_r, lowess=True,
              scatter_kws={'alpha': 0.6, 's': 30},
              line_kws={'color': 'red', 'lw': 2})
plt.xlabel("Predicted Median Home Value")
plt.ylabel("Residuals (Actual - Predicted)")
plt.title("Residual Plot for Reduced Model")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Residual Plot for Reduced Model
Residuals (Actual - Predicted)
Predicted Median Home Value
25
20
15
10
5
0
−5
0
5
10
15
20
25
30
35