BOSTON HOUSING

Import Libraries

In [ ]:

```
#Importing required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Load Dataset

In [ ]:

```
#read the dataset & save as df
df = pd.read_csv('BostonHousing.csv')
```

In [ ]:

```
#Display the Dataset
df
```

In [ ]:

```
#Check basic statistics of the data
df.describe()
```

Check for missing values in the DataFrame

In [ ]:

```
# Finding null values
df.isnull().sum()
```

In [ ]:

```
# Check the data types of each column
df.dtypes
```

In [ ]:

```
#Information of each data
df.info()
```

In [ ]:

```
#sns.pairplot(df)
```

Preparing the Independent Features & Dependent Feature

In [ ]:

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [ ]:

```
#Independent Features
X
```

In [ ]:

```
#Dependent Feature
y
```

Choosing a Model

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

In [ ]:

```
#Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0
```

In [ ]:

```
#Create and train the Multiple Linear Regression model
Model = LinearRegression()
Model.fit(X_train, y_train)
```

Predicting the Model

In [ ]:

```
#Make predictions on the test data
y_pred = Model.predict(X_test)
```

Evaluate the model's performance

In [ ]:

```
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
```

In [ ]:

```
# Calculate R-squared (coefficient of determination)
r2 = r2_score(y_test, y_pred)
```

In [ ]:

```
# Calculate the number of data points (n) and the number of features (k)
n = len(X_test)
k = X_test.shape[1]

# Calculate Adjusted R-squared
adjusted_r2 = 1 - ((1 - r2) * (n - 1) / (n - k - 1))
```

In [ ]:

```
rmse = np.sqrt(mse)
```

In [ ]:

```
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Adjusted R-squared:", adjusted_r2)
print("Root Mean Squared Error:", rmse)
```

Model is Good Fit

The MSE measures the average squared error between the model's predictions and the actual target values. the MSE is approximately 29.78. A lower MSE indicates a better fit, as it means the model's predictions are closer to the true values.

The R-squared value measures the proportion of variance in the target variable (median housing prices). The R-squared value is approximately 0.64 (or 64%). A higher R-squared value indicates a better fit.

The Adjusted R-squared value of 0.5935 indicates that approximately 59.35% of the variance in the median housing prices (MEDV) is explained by the features in the model. This suggests that the model captures a significant portion of the variability in the target variable, which is a positive sign.

In [ ]:

```
plt.figure(figsize=(8, 6))
sns.histplot(y, bins=30, kde=True)
plt.xlabel('Median Housing Prices (MEDV)')
plt.ylabel('Frequency')
plt.title('Distribution of Median Housing Prices')
plt.show()
```

In [ ]:

```
# Trained the model and have the predictions
y_pred = Model.predict(X_test)

plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual MEDV')
plt.ylabel('Predicted MEDV')
plt.title('Model Predictions vs. Actual Values')
plt.show()
```

Predicted vs. Actual Values:

The model performance plot shows a scatter plot of the actual target values ('MEDV') against the predicted target values obtained from the trained regression model. Each point in the scatter plot represents a data point from the test set.

Ideally, we want the points in this scatter plot to be as close to a diagonal line (y=x) as possible. If the model's predictions perfectly match the actual values, all points would lie on this line. Deviations from the diagonal line suggest that the model's predictions differ from the actual values.

Select a few random examples from the test data

In [ ]:

```
random_example_indices = np.random.choice(X_test.shape[0], 5, replace=False)
random_examples = X_test.iloc[random_example_indices]


# Make predictions using the trained model
predictions = Model.predict(random_examples)

# Display the predictions and corresponding actual values (if available)
for i, pred in enumerate(predictions):
    print(f"Example {i+1}:")
    print("Predicted Median Housing Price (MEDV):", pred)
    print("\n")
```