Documentation for Boston House Price Prediction

Introduction

The primary objective of this internship was to leverage machine learning techniques to predict house prices in Boston. This project aimed to understand the various factors influencing house prices and to develop a predictive model that can accurately forecast future prices based on these factors. By utilizing a dataset containing various attributes related to housing and socio-economic conditions, we aimed to draw meaningful insights and build a robust predictive model.

Dataset Description

Dataset Used: HousingData.csv

The dataset used for this project is the Boston Housing Dataset, which consists of 506 observations on housing prices and various predictors. The dataset includes 14 features such as the per capita crime rate (CRIM), the proportion of residential land zoned for lots over 25,000 sq. ft. (ZN), and the number of rooms (RM), among others. Each feature represents different socio-economic factors that potentially influence the median value of owner-occupied homes (MEDV), which is the target variable.

Data Preprocessing

Data preprocessing is a crucial step in any machine learning project. In this project, we addressed missing values using various imputation techniques. For instance, missing values in the 'CRIM' feature were imputed using the median strategy, while 'ZN' and 'CHAS' were imputed using the most frequent values. The SimpleImputer class from sklearn was used for this purpose. This approach helped in maintaining the integrity of the dataset while preparing it for further analysis.

Python code:

```
from sklearn.impute import SimpleImputer

imp_mean = SimpleImputer(strategy='mean')
imp_median = SimpleImputer(strategy='median')
imp_mode = SimpleImputer(strategy='most_frequent')

data['CRIM'] = imp_median.fit_transform(data[['CRIM']])
data['ZN'] = imp_mode.fit_transform(data[['ZN']])
data['INDUS'] = imp_mean.fit_transform(data[['INDUS']])
data['CHAS'] = imp_mode.fit_transform(data[['CHAS']])
data['AGE'] = imp_median.fit_transform(data[['AGE']])
data['LSTAT'] = imp_mean.fit_transform(data[['LSTAT']])
```

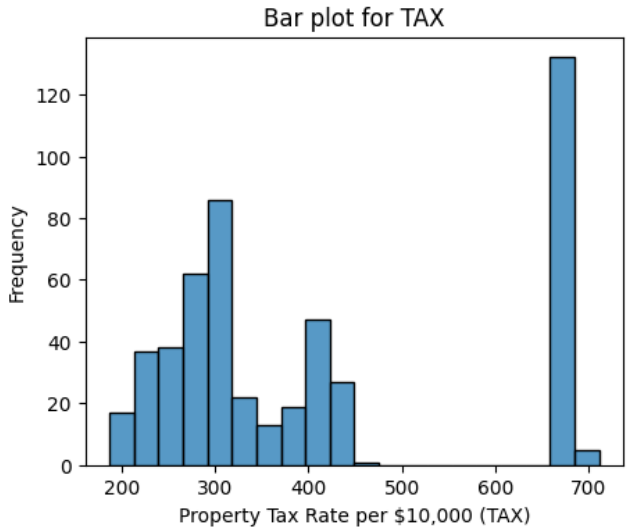
Data Visualization

Data visualization is essential for understanding the distribution and relationships between different features in the dataset. It helps in identifying patterns, trends, and outliers that could influence the predictive model.

Bar Plots : Bar plots are used to compare the frequency or count of categorical variables. They can provide insights into the distribution of categorical features.

Python code :

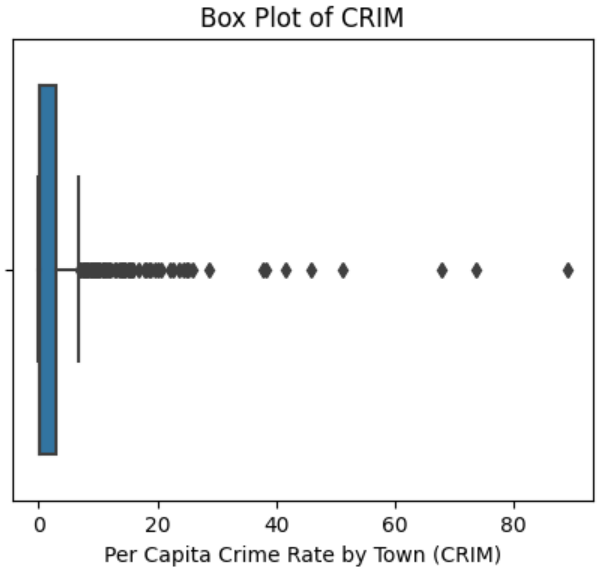
```
plt.figure(figsize=(5, 4))
sns.histplot(data['TAX'], bins=20)
plt.title('Bar plot for TAX')
plt.xlabel('Property Tax Rate per $10,000 (TAX)')
plt.ylabel('Frequency')
plt.show()
```



Box Plot : The box plot of CRIM provides insights into the distribution of crime rates across different towns. It highlights the presence of any outliers that could potentially skew the model predictions.

Python code :

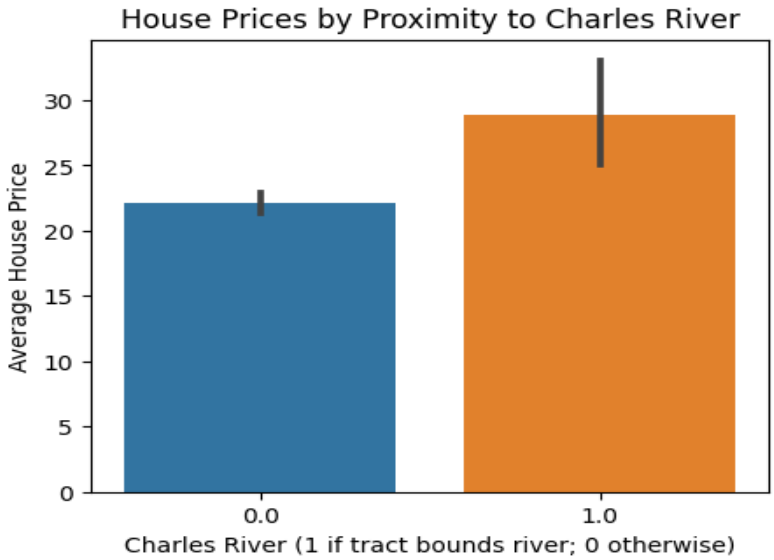
```
plt.figure(figsize=(5,4))
sns.boxplot(x=data['CRIM'])
plt.title('Box Plot of CRIM')
plt.xlabel('Per Capita Crime Rate by Town (CRIM)')
plt.show()
```



Count Plot : A countplot is a type of bar plot that represents the count of observations in each categorical bin using bars. It's particularly useful for visualizing the distribution of a categorical variable and how it compares across different categories.

Python code :

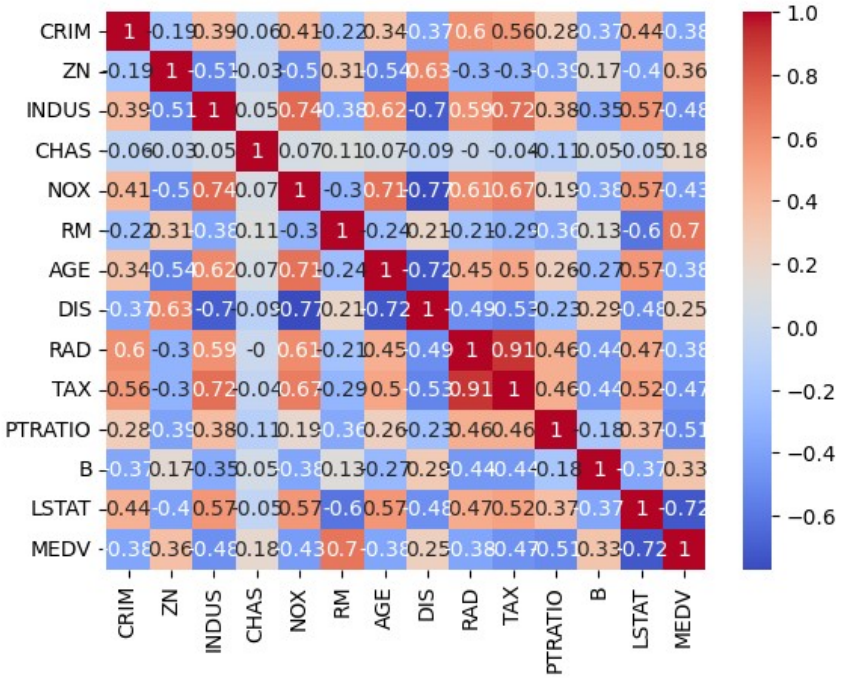
```
plt.figure(figsize=(5,4))
sns.barplot(x='CHAS', y='MEDV', data=data)
plt.title('House Prices by Proximity to Charles River')
plt.xlabel('Charles River (1 if tract bounds river; 0 otherwise)')
plt.ylabel('Average House Price')
plt.show()
```



Correlation Heatmaps : Heatmaps are a graphical representation of data where individual values are represented as colors. They are particularly useful for visualizing the correlation matrix of the dataset to understand the relationships between different features.

Python Code :

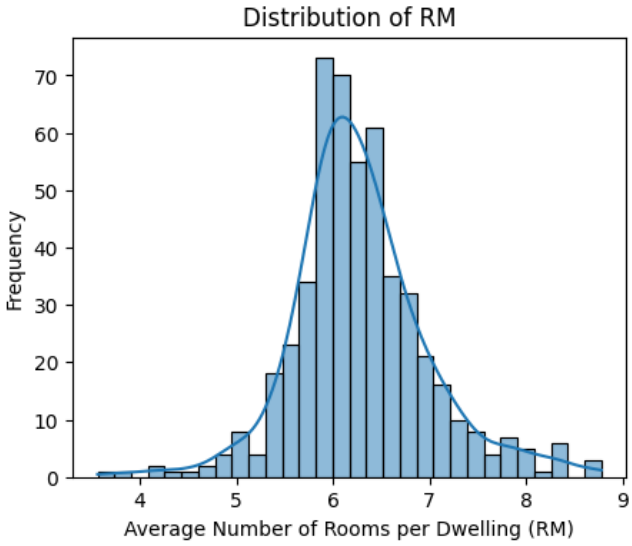
```
correlation_matrix = data.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```



Distribution Plots : Distribution plots (or histograms) show the distribution of a single variable. They help in understanding the spread and skewness of the data.

Python Code :

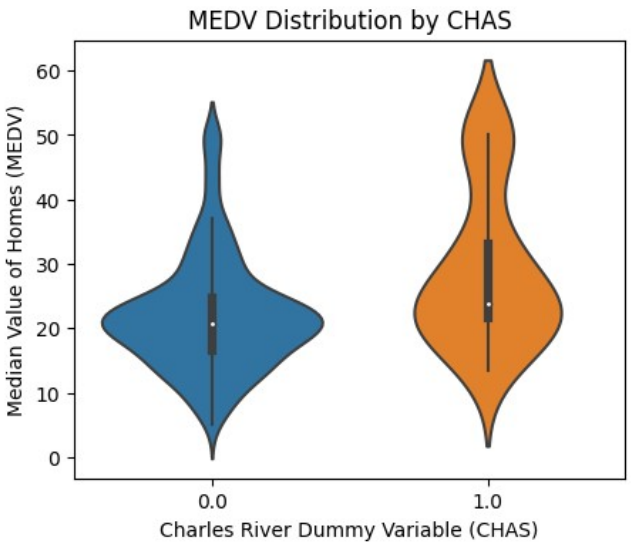
```
plt.figure(figsize=(5,4))
sns.histplot(data['RM'], bins=30, kde=True)
plt.title('Distribution of RM')
plt.xlabel('Average Number of Rooms per Dwelling (RM)')
plt.ylabel('Frequency')
plt.show()
```



The violin plot : The violin plot is a method of plotting numeric data and can be understood as a combination of a box plot and a kernel. density plot. It shows the distribution of the data across different levels of a categorical variable.

Python Code :

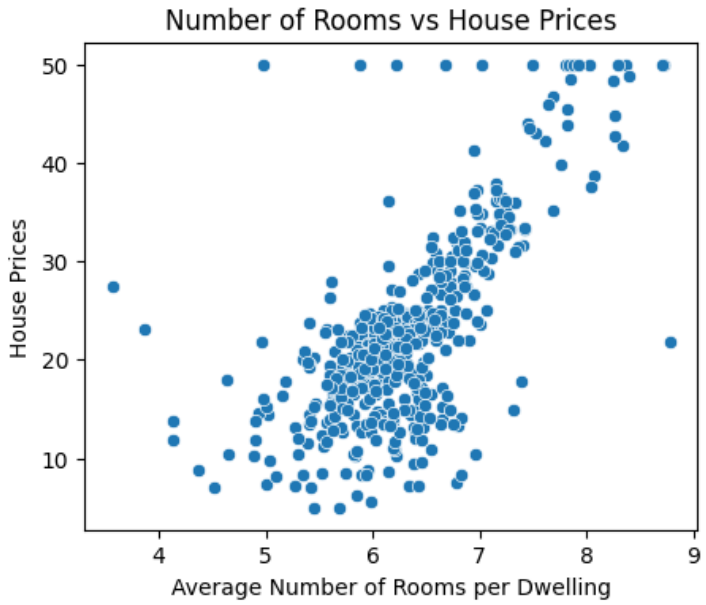
```
plt.figure(figsize=(5,4))
sns.violinplot(x='CHAS', y='MEDV', data=data)
plt.title('MEDV Distribution by CHAS')
plt.xlabel('Charles River Dummy Variable (CHAS)')
plt.ylabel('Median Value of Homes (MEDV)')
plt.show()
```



Scatter Plot: A scatterplot is a type of data visualization that displays the relationship between two continuous variables. Each point on the scatterplot represents an observation from the dataset, with the position of the point determined by the values of the two variables. This plot is particularly useful for identifying patterns, trends, correlations, and potential outliers within the data.

Python Code:

```
plt.figure(figsize=(5,4))
sns.scatterplot(x='RM', y='MEDV',data=data)
plt.title('Number of Rooms vs House Prices')
plt.xlabel('Average Number of Rooms per Dwelling')
plt.ylabel('House Prices')
plt.show()
```



Model Selection

Model selection is a critical step in the machine learning workflow where you choose the most suitable algorithm to solve a specific problem based on the characteristics of the data and the task at hand. It begins with understanding the type of problem (e.g., classification, regression, clustering) and conducting preliminary data analysis to uncover data patterns and distributions. Model is selected based on a balance of performance, complexity, and interpretability, ensuring it meets the specific needs and constraints of the project. This iterative process ensures that the selected model provides robust and reliable predictions for real-world applications.

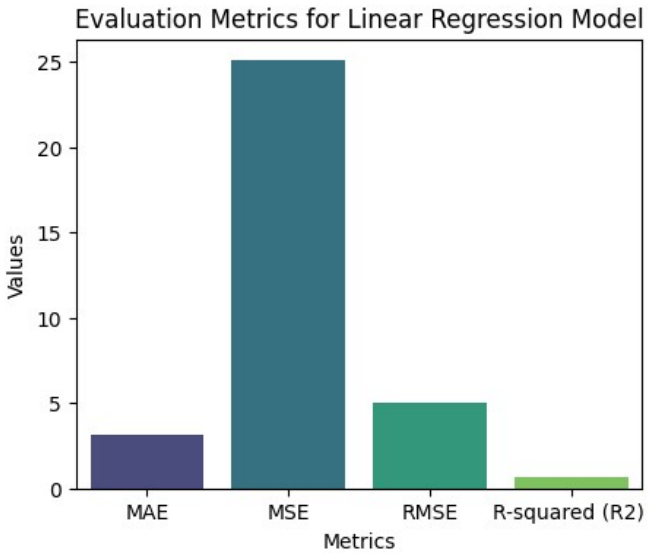
Linear Regression: Linear regression was chosen as the initial predictive model due to its simplicity and interpretability. This regression technique models the relationship between the target variable and one or more predictor variables by fitting a linear equation to observed data.

Python code :

```
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

X = data.drop('MEDV', axis=1)
y = data['MEDV']

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))
print('R2 Score:', r2_score(y_test, y_pred))
```

Output :

```
Mean Absolute Error (MAE): 3.1584994146197096
Mean Squared Error (MSE): 25.072290196306753
Root Mean Squared Error (RMSE): 5.007223801300153
R-squared (R2): 0.6581072308584777
```



Mean Absolute Error (MAE): 3.1584994146197096

This indicates that on average, the predictions made by the linear regression model are off by approximately 3.15 units from the actual values.

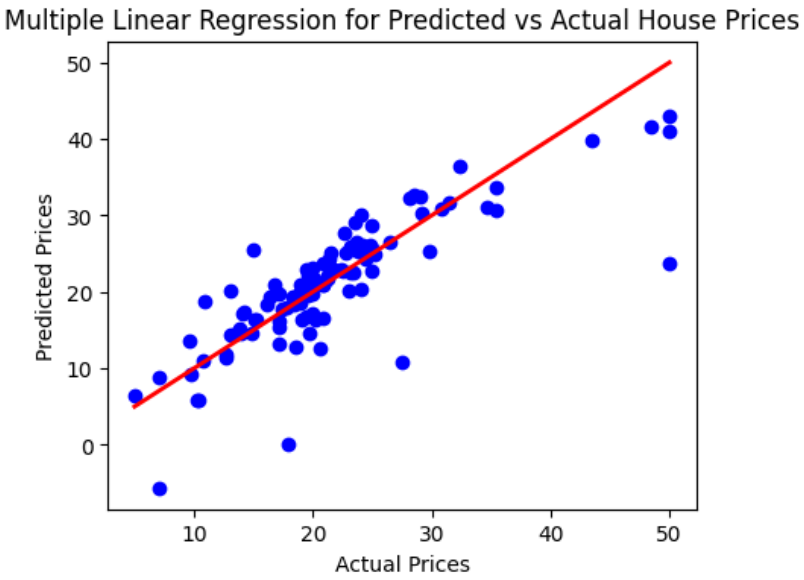
Mean Squared Error (MSE): 25.0722

This indicates that the average squared difference between the predicted and actual values is 25.07. The higher value of MSE indicates that there are some large errors in the predictions.

R^2 Score: 0.65811

This indicates that approximately 65.8% of the variance in the dependent variable is predictable from the independent variables. This shows a good fit but leaves room for improvement.

Multiple linear regression : Multiple linear regression is a statistical technique used to model the relationship between one dependent variable and multiple independent variables. It extends simple linear regression by considering more than one predictor, allowing for a more comprehensive understanding of how various factors contribute to the outcome. Multiple linear regression is widely used in various fields for predictive analysis and to gain insights into the influence of multiple factors on a single outcome.



K Neighbour Algorithm : In k-NN, the "k" represents the number of nearest neighbors considered when making predictions. For a given data point, the algorithm identifies the k closest data points in the training set based on a distance metric, typically Euclidean distance. In classification, the predicted class is determined by the majority class among the k nearest neighbors, while in regression, the predicted value is the average of the values of the k nearest neighbors.

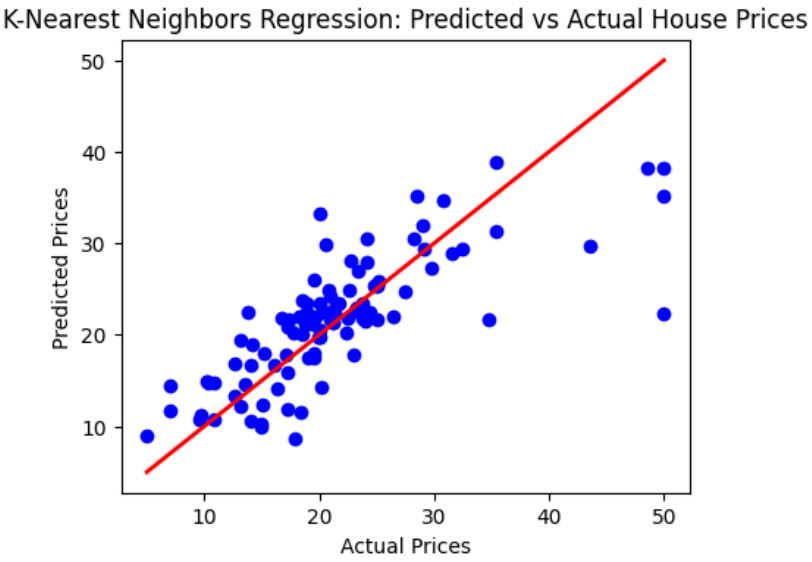
Python code :

```
from sklearn.neighbors import KNeighborsRegressor
```

```
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(x_train, y_train)
y_pred = knn_model.predict(x_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared (R2) Score:", r2_score(y_test, y_pred))
```

Output :

```
Mean Squared Error: 29.704478431372554
R-squared (R2) Score: 0.5949414151124269
```



Mean Squared Error (MSE): **29.704**

- The average squared difference between the predicted and actual values is 29.70. This value shows that the KNN model has a reasonable error rate, with some larger errors present.

R^2 Score: **0.5949**

- The R^2 score indicates that 59.5% of the variance in the dependent variable is predictable from the independent variables. This demonstrates that the KNN model fits the data well, capturing a significant portion of the variability in housing prices.

Model Evaluation

Model evaluation is a crucial phase in the machine learning process where the performance and effectiveness of a trained model are assessed to ensure it can make accurate predictions on new, unseen data. The chosen model is then validated on a separate test set to confirm its predictive capability, and interpretability tools may be used to explain the model's decisions, ensuring it meets the project's requirements for accuracy, reliability, and transparency.

Mean Absolute Error (MAE): Measures the average magnitude of errors in a set of predictions, without considering their direction.

Mean Squared Error (MSE): Measures the average of the squares of the errors, giving more weight to larger errors.

Root Mean Squared Error (RMSE): The square root of MSE, providing an error metric that is in the same units as the target variable.

R-squared (R^2): Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.

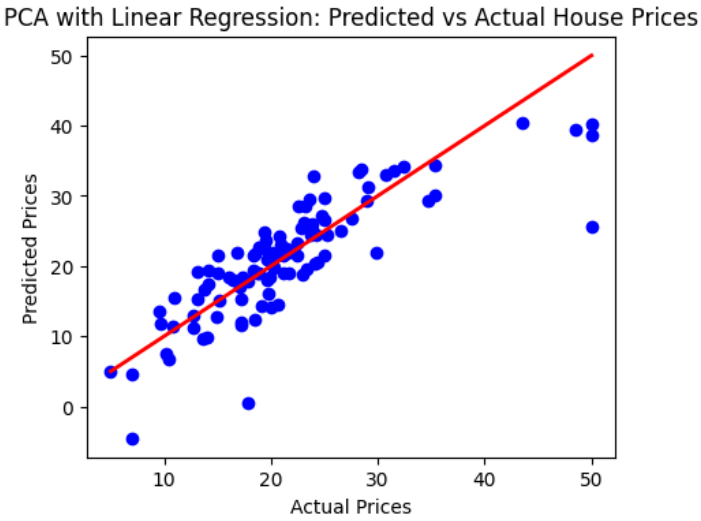
Advanced Machine Learning Techniques

Principal Component Analysis (PCA) : Principal Component Analysis (PCA) is a powerful dimensionality reduction technique used to transform a large set of variables into a smaller one that still contains most of the information in the large set. This method is particularly useful in reducing the complexity of a dataset while retaining its essential characteristics, which can lead to more efficient and effective model training. PCA works by identifying the principal components of the data, which are the directions in which the data varies the most. These directions are orthogonal to each other, meaning they are uncorrelated. By projecting the data onto these principal components, we can reduce the number of dimensions while preserving as much variance as possible.

Python code:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
x_pca = pca.fit_transform(x)
35xp_train, xp_test, yp_train, yp_test = train_test_split(x_pca,
y, test_size=0.2, random_state=42)
linear_model = LinearRegression()
linear_model.fit(xp_train, yp_train)
yp_pred = linear_model.predict(xp_test)
print("Mean Squared Error:", mean_squared_error(yp_test, yp_pred))
print("R-Squared Score:", r2_score(yp_test, yp_pred))

plt.figure(figsize=(5,4))
plt.scatter(yp_test, yp_pred, color='blue')
plt.plot([min(yp_test), max(yp_test)], [min(yp_test),
max(yp_test)],color='red', linewidth=2)
plt.title('PCA with Linear Regression: Predicted vs Actual House
Prices')
plt.xlabel('Actual Prices')
```

```
plt.ylabel('Predicted Prices')
plt.show()
```



Advanced Machine Learning Models

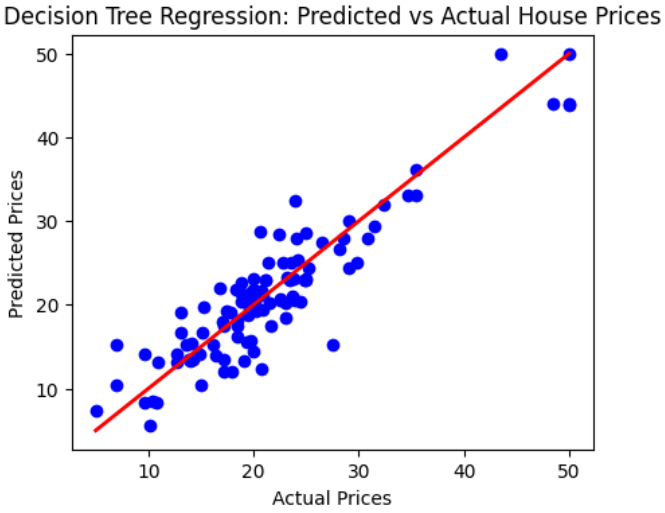
Decision Tree Algorithm: A Decision Tree is a popular supervised learning algorithm used for both regression and classification tasks in machine learning. It works by recursively partitioning the data into subsets based on the most significant attribute at each node of the tree. Here's how it operates: starting from the root node, the algorithm selects the attribute that best splits the data into distinct groups. This splitting process continues recursively for each subset, creating a tree structure where each internal node represents a decision based on an attribute, and each leaf node holds a prediction or outcome.

Python code :

```
from sklearn.tree import DecisionTreeRegressor
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(x_train, y_train)
y_pred = tree_model.predict(x_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared (R2) Score:", r2_score(y_test, y_pred))
```

Output :

```
Mean Squared Error: 12.571274509803924
R-squared (R2) Score: 0.8285745809360402
```



Mean Squared Error (MSE): 12.571

The average squared difference between the predicted and actual values is 12..57 indicating the presence of large errors.

R^2 Score: 0.828

The R^2 score indicates that 82.8% of the variance in the dependent variable is predictable from the independent variables. This shows good model performance but can be improved.

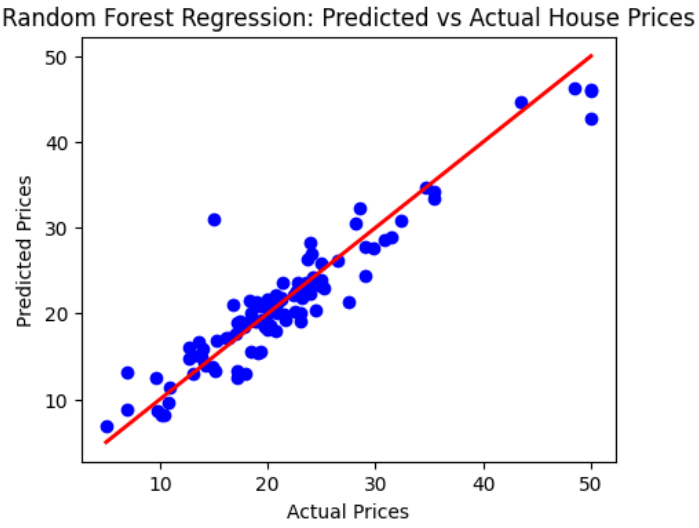
Random Forest: Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the mode of the classes (classification) or mean prediction (regression) of the individual trees. This method helps in improving the predictive accuracy and controlling overfitting.

Python code:

```
from sklearn.ensemble import RandomForestRegressor

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

print('Random Forest MAE:', mean_absolute_error(y_test,
y_pred_rf))
print('Random Forest MSE:', mean_squared_error(y_test, y_pred_rf))
print('Random Forest R2 Score:', r2_score(y_test, y_pred_rf))
```

Output :

```
Mean Squared Error: 8.278940480392155
R-squared (R2) Score: 0.8871060495775506
```



Mean Squared Error (MSE): 8.278

The average squared difference between the predicted and actual values is 8.278, which is significantly lower than the MSE of the linear regression model.

R^2 Score: 0.887

The R^2 score indicates that 88.7% of the variance in the dependent variable is predictable from the independent variables. This suggests a better fit than the linear regression model.

Gradient Boosting : Gradient Boosting is another powerful ensemble learning technique that builds models sequentially, each new model correcting errors made by the previous ones. It is particularly effective in improving predictive accuracy and handling complex data patterns.

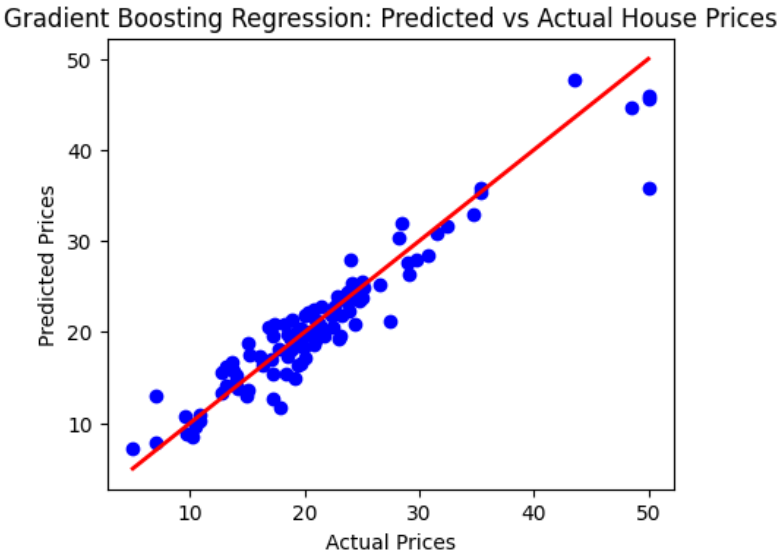
Python code:

```
from sklearn.ensemble import GradientBoostingRegressor

gb_model = GradientBoostingRegressor(n_estimators=100,
random_state=42)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)

print('Gradient Boosting MAE:', mean_absolute_error(y_test,
y_pred_gb))
print('Gradient Boosting MSE:', mean_squared_error(y_test,
y_pred_gb))
```

```
print('Gradient Boosting R2 Score:', r2_score(y_test, y_pred_gb))
```

Output :

```
Mean Squared Error: 7.155319376730123
R-squared (R2) Score: 0.9024280615512894
```



Mean Squared Error (MSE): 7.155

The average squared difference between the predicted and actual values is 7.155, indicating fewer large errors in predictions.

R^2 Score: 0.9024

The R^2 score indicates that 90.24% of the variance in the dependent variable is predictable from the independent variables. This shows excellent model performance.

Support Vector Machines (SVM): Support Vector Machines are supervised learning models that analyze data for classification and regression analysis. The strength of SVM lies in its ability to perform well in high-dimensional spaces and its effectiveness in cases where the number of dimensions exceeds the number of samples.

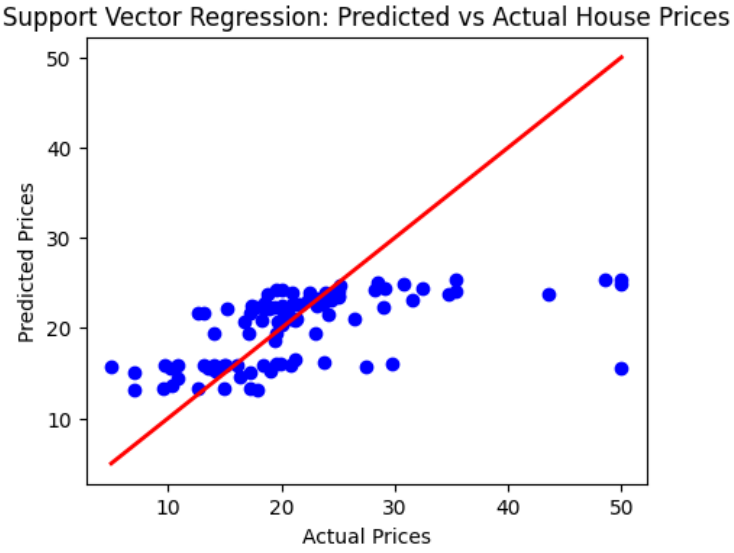
Python code :

```
from sklearn.svm import SVR

svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)

y_pred_svr = svr_model.predict(X_test)
```

```
print('SVM MAE:', mean_absolute_error(y_test, y_pred_svr))
print('SVM MSE:', mean_squared_error(y_test, y_pred_svr))
print('SVM R2 Score:', r2_score(y_test, y_pred_svr))
```

Output :

```
Mean Squared Error: 52.931033106567114
R-squared (R2) Score: 0.2782176123267932
```



Mean Squared Error (MSE): 52.93

The average squared difference between the predicted and actual values is 52.93, indicating more errors compared to the previous models.

R^2 Score: 0.2782

The R^2 score indicates that 27.82% of the variance in the dependent variable is predictable from the independent variables. This shows good model performance but not as strong as random forest and gradient boosting.

XG Booster : XGBoost, short for Extreme Gradient Boosting, is a powerful and efficient implementation of the gradient boosting algorithm. It is widely used in machine learning competitions and real-world applications due to its speed and performance. XGBoost works by building an ensemble of weak learners, typically decision trees, in a sequential manner. Clustering the Data.

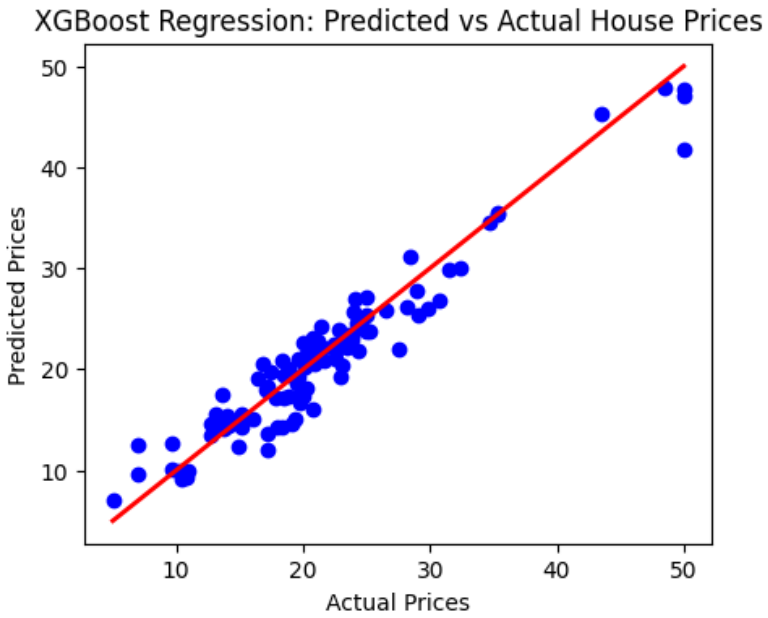
Python Code :

```
import xgboost as xgb
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
n_estimators=100,random_state=42)
```

```
xgb_model.fit(x_train, y_train)
y_pred = xgb_model.predict(x_test)
34print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-Squared Score:", r2_score(y_test, y_pred))
```

Output :

```
Mean Squared Error: 5.537024472117028
R-Squared Score: 0.9244955839791882
```



Mean Squared Error (MSE): 5.537

- The average squared difference between the predicted and actual values is 5.537. This value shows that the XGBoost model has a relatively low error rate, indicating that the model is making accurate predictions.

R^2 Score: **0.9244**

- The R^2 score indicates that 92.44% of the variance in the dependent variable is predictable from the independent variables. This demonstrates that the XGBoost model fits the data very well, capturing a significant portion of the variability in housing prices.

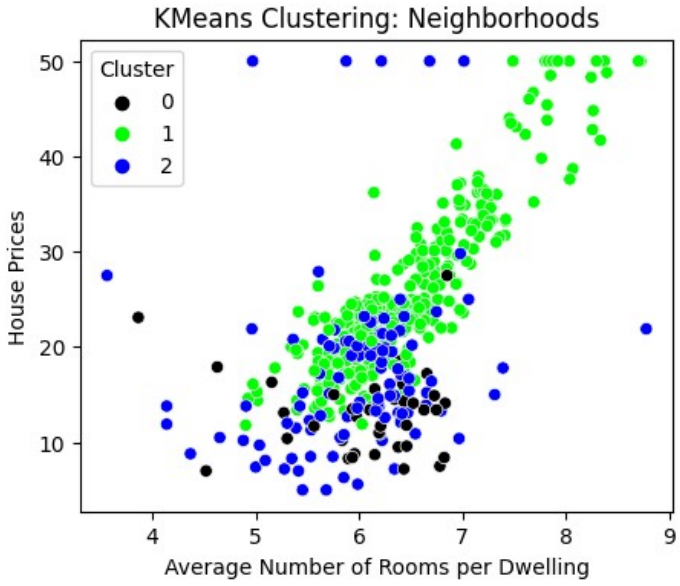
K-Means Clustering: To further analyze the data, KMeans clustering was employed to group similar observations into clusters. This unsupervised learning technique helped in identifying patterns and segmenting the data based on inherent similarities.

Python Code:

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
data['Cluster'] = kmeans.fit_predict(X)

custom_palette = [' 000000', ' 00FF00',' 0000FF']
plt.figure(figsize=(5,4))
sns.scatterplot(x='RM', y='MEDV', hue='Cluster',
palette=custom_palette, data=data)
plt.title('KMeans Clustering: Neighborhoods')
plt.xlabel('Average Number of Rooms per Dwelling')
plt.ylabel('House Prices')
plt.show()
```

Output :



Comparing an Evaluation metices between multiple models

The bar plots presented in this analysis provide a visual comparison of the performance metrics for various machine learning models used in predicting Boston housing prices. Each bar plot focuses on a specific evaluation metric: Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R2) score. These metrics help in understanding the accuracy and reliability of each model.

Mean Absolute Error (MAE)

The first bar plot displays the MAE scores for each model. MAE measures the average magnitude of errors between the predicted and actual values, without considering their direction. A lower MAE indicates a model that makes predictions closer to the actual values on average. In this

plot, the models are listed on the vertical axis, and their corresponding MAE scores are represented by the length of the horizontal bars.

Mean Squared Error (MSE)

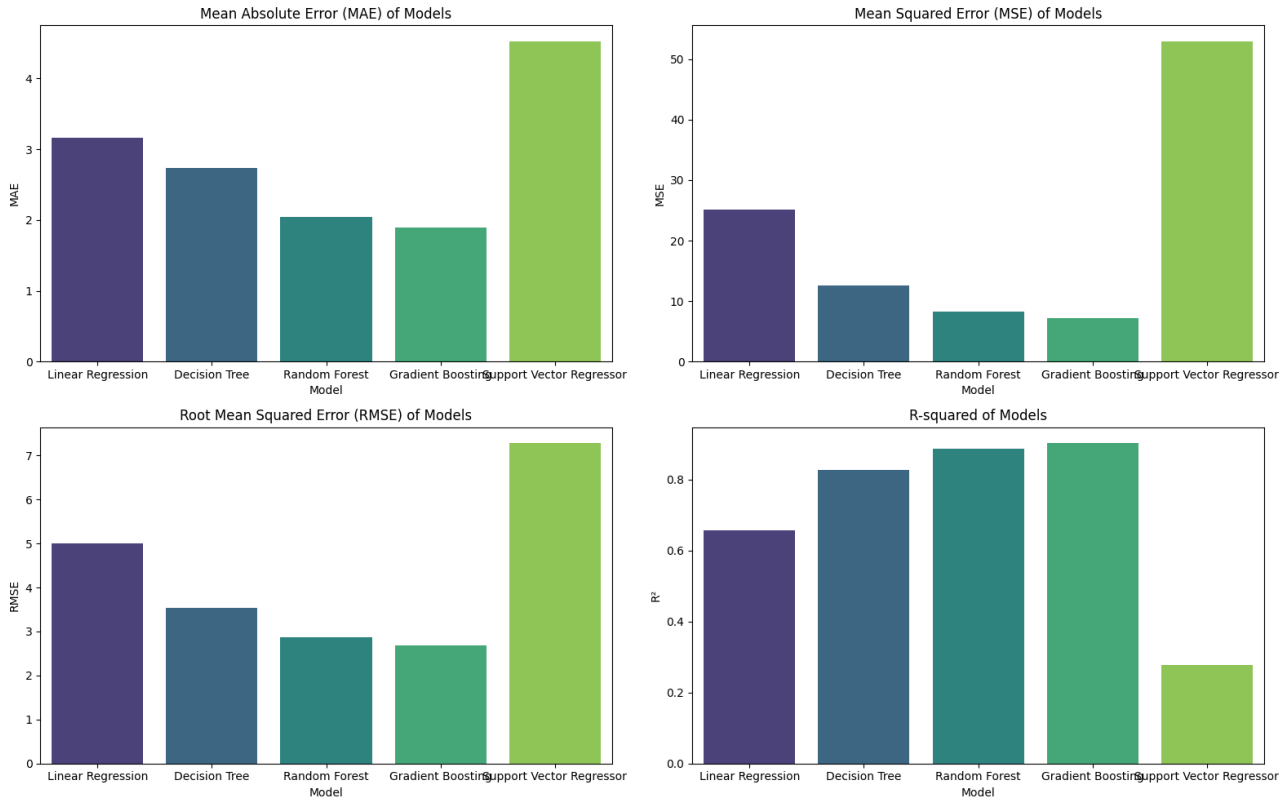
The second bar plot illustrates the MSE scores for the models. MSE is similar to MAE but squares the errors before averaging, making it more sensitive to outliers. A lower MSE signifies a model with fewer large errors. The models are again listed on the vertical axis, with their MSE scores represented by the length of the horizontal bars.

R-squared (R2) Score

The third bar plot shows the R2 scores for the models. The R2 score represents the proportion of the variance in the dependent variable that is predictable from the independent variables. A higher R2 score indicates a better fit of the model to the data.

Root Mean Square Error (RMSE):

At last plot, RMSE is another metric used to measure the average magnitude of the errors between predicted values and actual values. It is similar to MSE but differs in that it takes the square root of the average of squared differences between predicted and actual values. RMSE is expressed in the same units as the target variable, making it more interpretable in practical terms compared to MSE.



Conclusion

Summary of Findings: The analysis revealed that certain features, such as the number of rooms (RM) and the proximity to the Charles River (CHAS), have a significant impact on house prices. The linear regression model and other advanced models demonstrated a reasonable predictive capability, as evidenced by the evaluation metrics. However, there is room for improvement by exploring more complex models and feature engineering techniques.