Name-Surname: Phithatsanan Lertthanasiriwat

Student ID: 64070503478

CPE 393 Machine Learning

Lab Instructions: Building a California House Price Prediction Model

In this lab, students will develop a predictive model to estimate California housing prices using machine learning. This will involve data visualization, feature engineering, model building, and evaluation.

Step 1: Load the California Housing dataset and change to pandas data frame

---

+ Code + Text

```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()

import pandas as pd
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['price'] = housing.target
```

```
# Additional imports for EDA, visualizations, model building & evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For splitting, modeling, and tuning
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

Step 2: Exploratory Data Analysis (EDA)

2.1 Summary of the dataset

```
print("=== Information about the DataFrame ===")
print(df.info())
```

```
=== Information about the DataFrame ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   price       20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
None
```

```
print("\n=== Statistical Summary ===")
print(df.describe())
```

```
=== Statistical Summary ===
            MedInc      HouseAge      AveRooms     AveBedrms    Population
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude         price
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010
```

2.2 Check for missing values

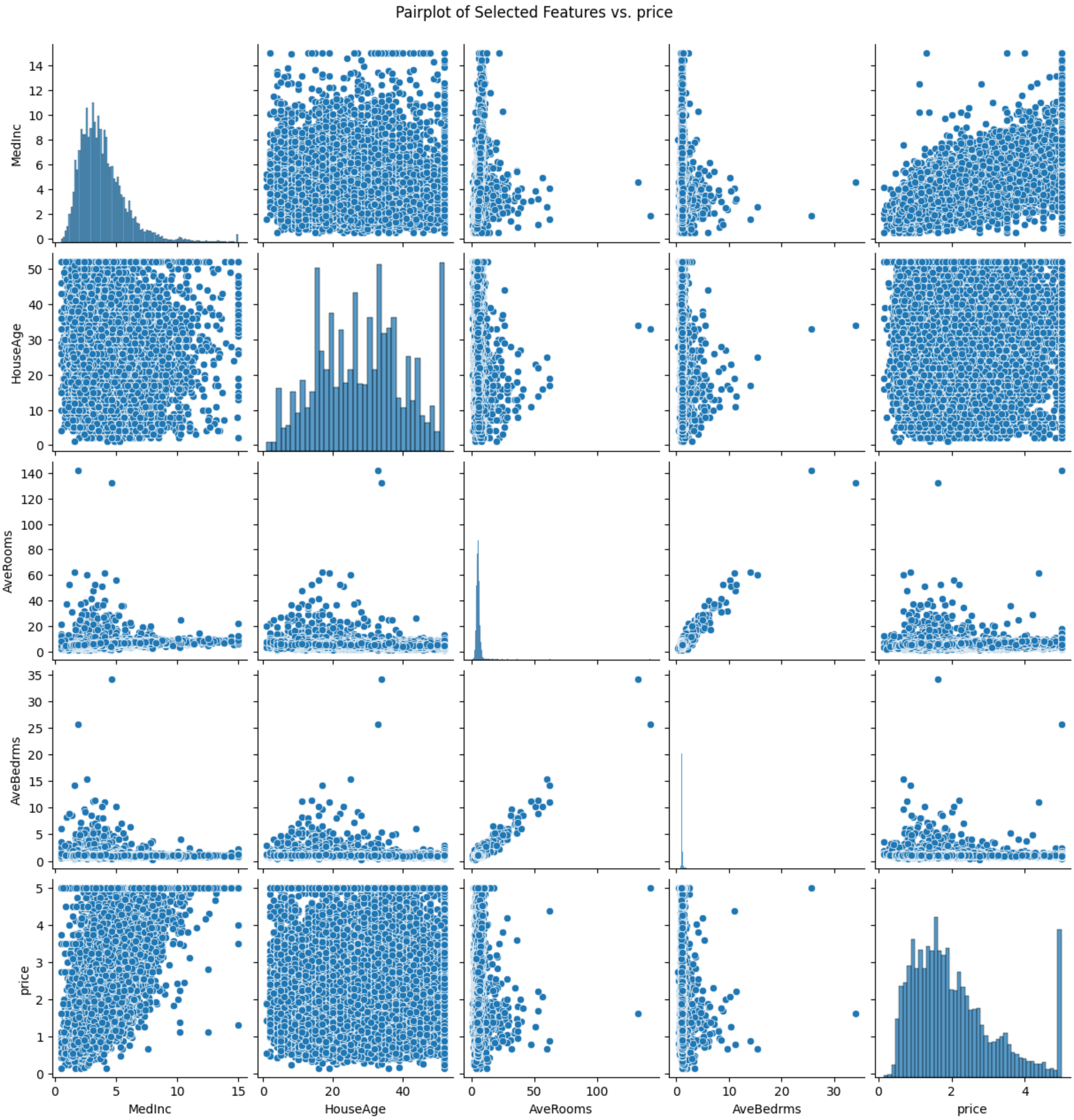
```
print("\n=== Checking Missing Values ===")
print(df.isnull().sum())
```

```
=== Checking Missing Values ===
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
price         0
dtype: int64
```

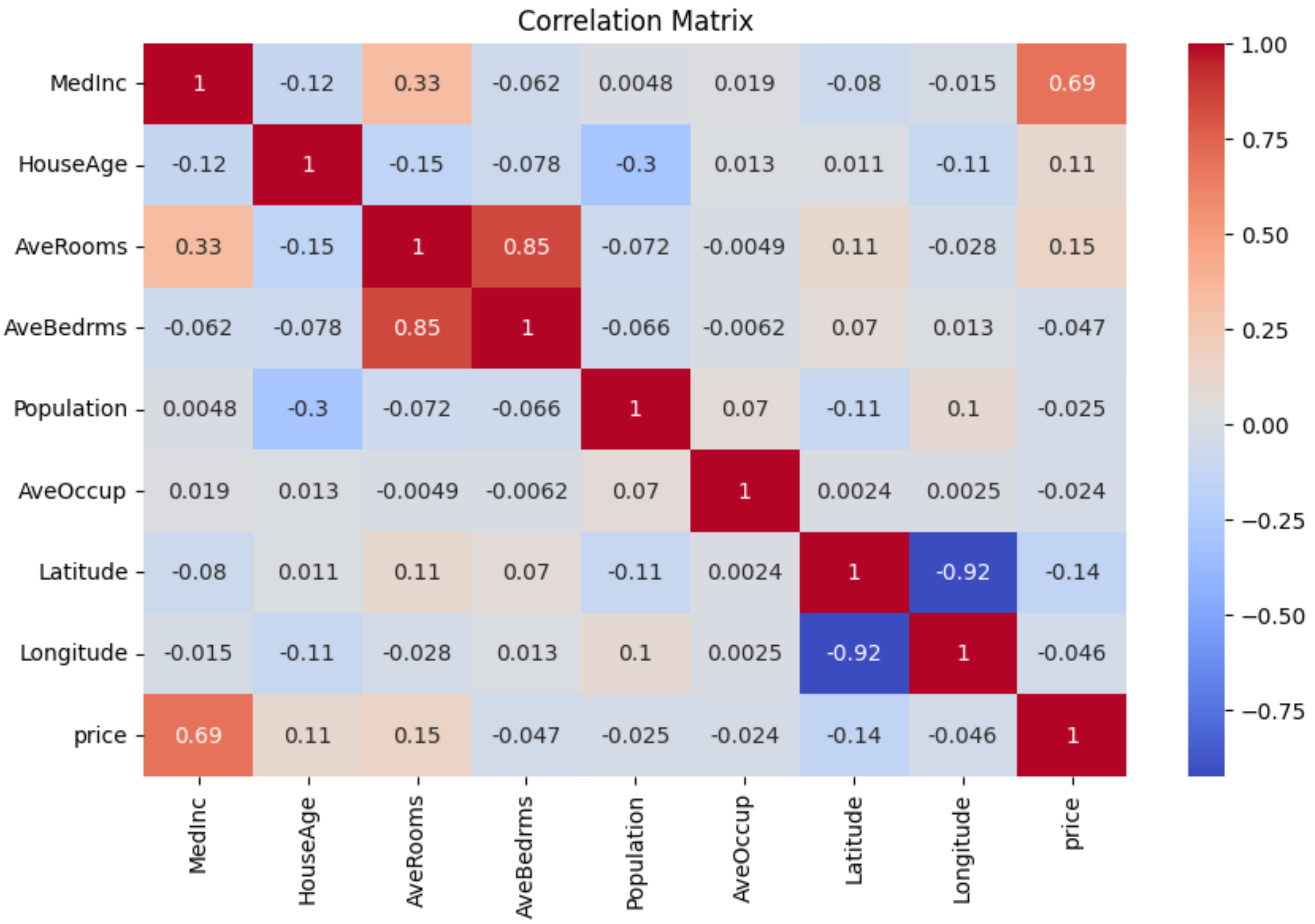
2.3 Visualize relationships

```
sns.pairplot(df[['MedInc','HouseAge','AveRooms','AveBedrms','price']])
```

```
plt.suptitle("Pairplot of Selected Features vs. price", y=1.02)
plt.show()
```



```
# Correlation Heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	price
MedInc	1	-0.12	0.33	-0.062	0.0048	0.019	-0.08	-0.015	0.69
HouseAge	-0.12	1	-0.15	-0.078	-0.3	0.013	0.011	-0.11	0.11
AveRooms	0.33	-0.15	1	0.85	-0.072	-0.0049	0.11	-0.028	0.15
AveBedrms	-0.062	-0.078	0.85	1	-0.066	-0.0062	0.07	0.013	-0.047
Population	0.0048	-0.3	-0.072	-0.066	1	0.07	-0.11	0.1	-0.025
AveOccup	0.019	0.013	-0.0049	-0.0062	0.07	1	0.0024	0.0025	-0.024
Latitude	-0.08	0.011	0.11	0.07	-0.11	0.0024	1	-0.92	-0.14
Longitude	-0.015	-0.11	-0.028	0.013	0.1	0.0025	-0.92	1	-0.046
price	0.69	0.11	0.15	-0.047	-0.025	-0.024	-0.14	-0.046	1

Step 3: Feature Engineering

3.1 Create a new feature (RoomRatio = AveRooms / AveBedrms)

```
# Add a small epsilon to avoid divide-by-zero
df['RoomRatio'] = df['AveRooms'] / (df['AveBedrms'] + 1e-5)
```

3.2 Scale numerical features (excluding the target 'price') and will do this AFTER splitting to avoid data leakage

3.3 Split into training and test sets (80% train, 20% test)

```
X = df.drop('price', axis=1)
y = df['price']

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


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)
```

Step 4: Building Machine Learning Models

4.1 Linear Regression

```
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)
y_pred_lin = lin_reg.predict(X_test_scaled)
```

4.2 Decision Tree Regressor + Hyperparameter Tuning with GridSearchCV

```
param_grid_dt = {
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}


tree = DecisionTreeRegressor(random_state=42)


grid_search_dt = GridSearchCV(
    estimator=tree,
    param_grid=param_grid_dt,
    cv=5,                         # 5-fold cross-validation
    scoring='neg_mean_squared_error',
    n_jobs=-1
)


grid_search_dt.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ best_estimator_: DecisionTreeRegressor
  ▸ DecisionTreeRegressor
```

```
print("Best parameters for Decision Tree Regressor:", grid_search_dt.best_paran
best_dt = grid_search_dt.best_estimator_
y_pred_dt = best_dt.predict(X_test)
```

```
Best parameters for Decision Tree Regressor: {'max_depth': None, 'min_sampl
```

4.3 Random Forest Regressor

```
rf = RandomForestRegressor(
    n_estimators=100, max_depth=20, random_state=42
)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

Step 5: Model Evaluation

```
def evaluate_model(y_true, y_pred, model_name="Model"):
    mae  = mean_absolute_error(y_true, y_pred)
    mse  = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2   = r2_score(y_true, y_pred)
    print(f"--- {model_name} ---")
    print(f"MAE:  {mae:.3f}")
    print(f"MSE:  {mse:.3f}")
    print(f"RMSE: {rmse:.3f}")
    print(f"R²:   {r2:.3f}\n")
```

```
# Evaluate Linear Regression
evaluate_model(y_test, y_pred_lin, "Linear Regression")
```

```
--- Linear Regression ---
MAE:  0.529
MSE:  0.530
RMSE: 0.728
R²:   0.596
```

```
# Evaluate Decision Tree
evaluate_model(y_test, y_pred_dt, "Decision Tree Regressor (Tuned)")
```

```
--- Decision Tree Regressor (Tuned) ---
MAE:  0.410
MSE:  0.369
RMSE: 0.607
R²:   0.719
```

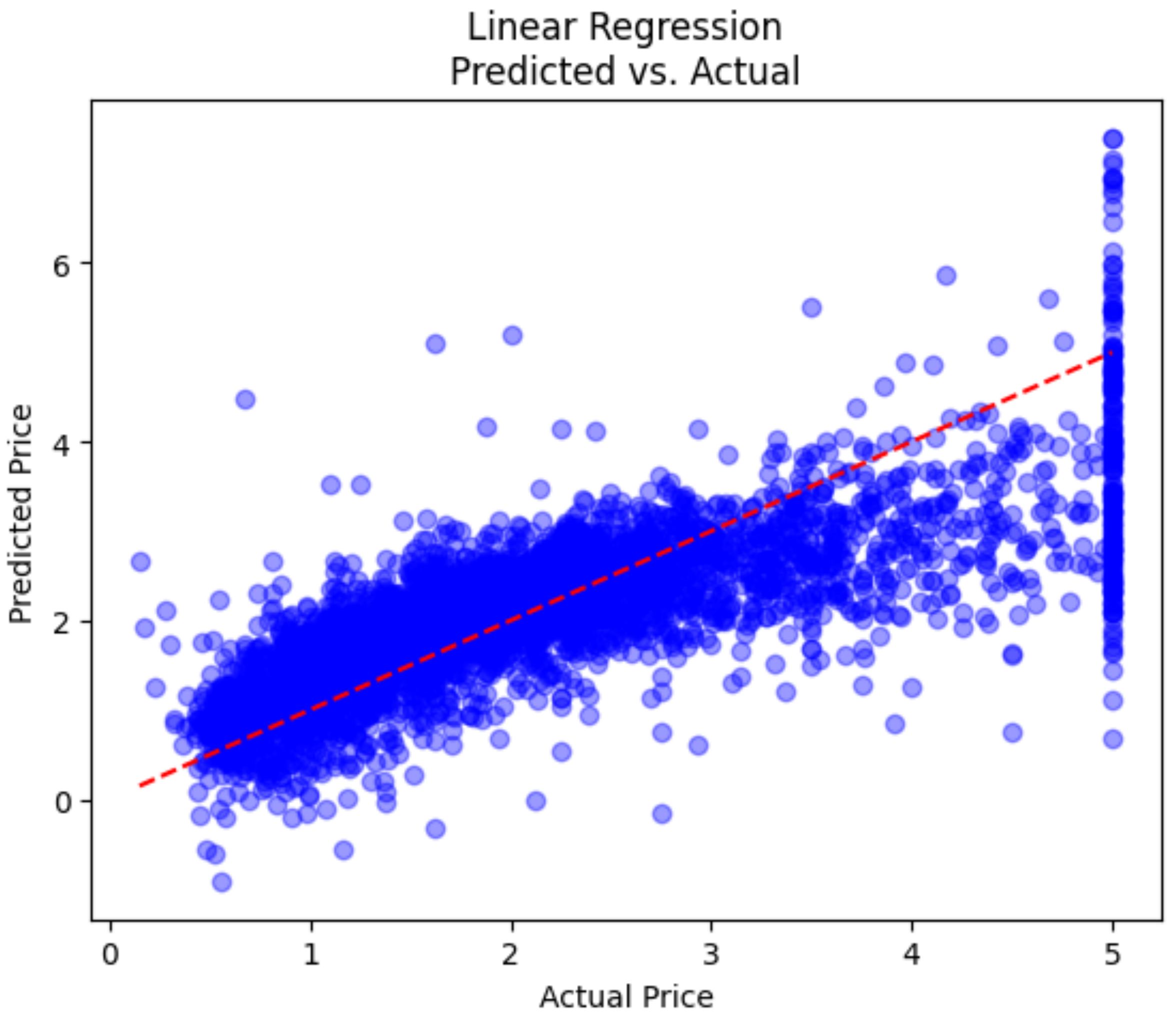
```
# Evaluate Random Forest
evaluate_model(y_test, y_pred_rf, "Random Forest Regressor")
```

```
--- Random Forest Regressor ---
MAE:  0.328
MSE:  0.256
RMSE: 0.506
R²:   0.805
```

Step 6: Visualize Predicted vs. Actual

6.1 Linear Regression

```
plt.figure(figsize=(16,5))

plt.subplot(1, 3, 1)
plt.scatter(y_test, y_pred_lin, alpha=0.4, color='blue')
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], 'r--')
plt.title("Linear Regression\nPredicted vs. Actual")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")

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

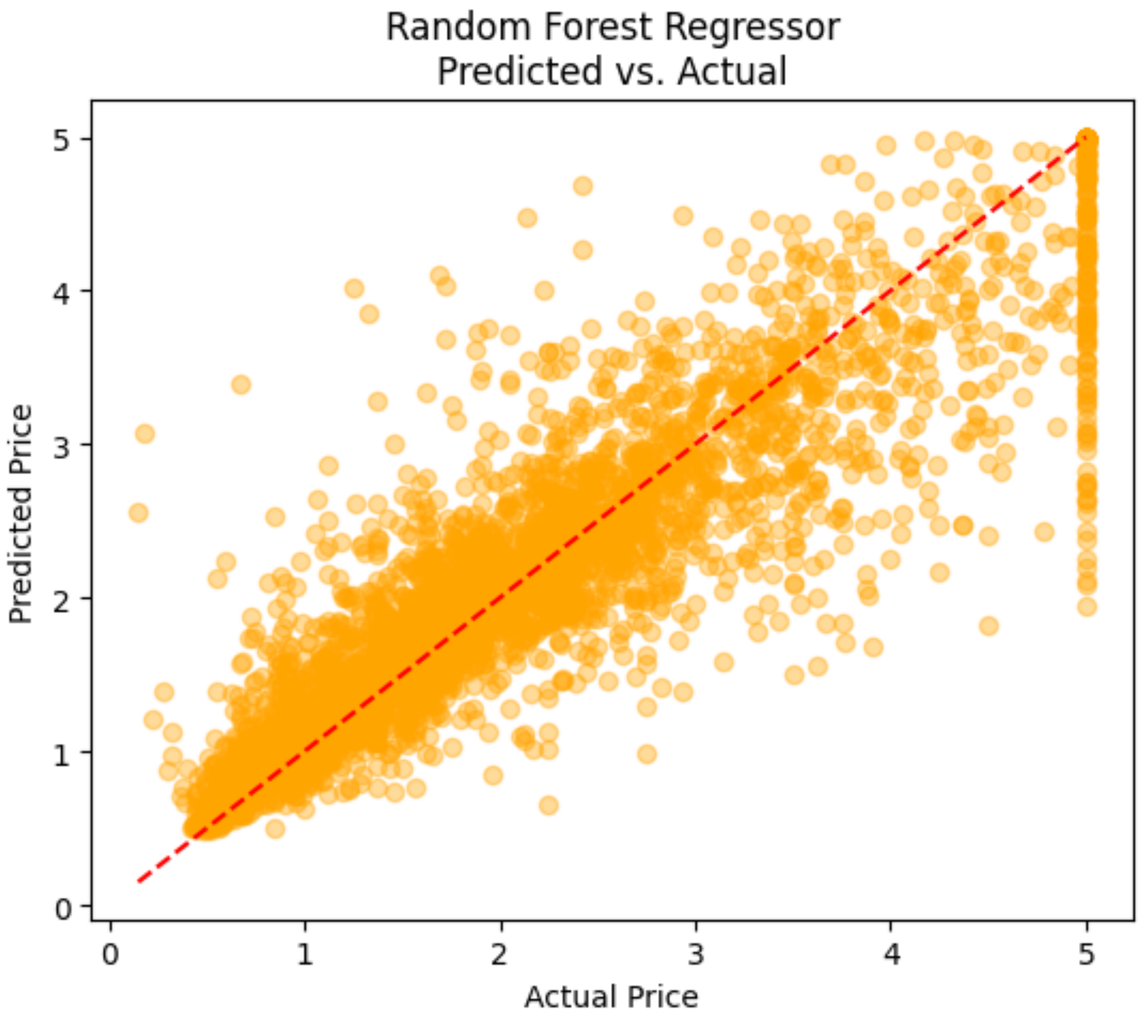


6.2 Decision Tree Regressor

```
plt.figure(figsize=(16,5))

plt.subplot(1, 3, 2)
plt.scatter(y_test, y_pred_dt, alpha=0.4, color='green')
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], 'r--')
plt.title("Decision Tree Regressor\nPredicted vs. Actual")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")

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

6.3 Random Forest Regressor

```
plt.figure(figsize=(16,5))

plt.subplot(1, 3, 3)
plt.scatter(y_test, y_pred_rf, alpha=0.4, color='orange')
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()], 'r--')
plt.title("Random Forest Regressor\nPredicted vs. Actual")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")

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



Result

Linear Regression The predictions generally follow the upward trend (more rooms, newer houses, etc. → higher price), but there is a systematic underprediction for higher actual prices. Notice how many points are below the red diagonal line when **Actual Price > 3**.

Often has higher MSE/RMSE when the relationship is non-linear or when features interact in ways that a simple linear model cannot capture.

Decision Tree Regressor The single decision tree captures more non-linearities than the linear model; however, still can see some **“blocky”** behavior. Many predictions still deviate from the diagonal line, particularly at the higher end of house prices and around certain mid-price clusters.

A well-tuned tree can outperform linear regression in capturing non-linear patterns, but it can also overfit, leading to somewhat inconsistent predictions in some regions.

Random Forest Regressor The predictions tend to cluster more tightly around the diagonal line (especially compared to Linear Regression), indicating **better overall accuracy**. There is still some scatter at the higher price ranges, but the distribution of points more closely follows the line.

By averaging many trees, Random Forest often **reduces overfitting** and captures more complex interactions among features. It generally provides **lower MAE/MSE** and higher R^2 than a single decision tree or a simple linear model.

Linear Regression shows an MAE of 0.529, MSE of 0.530, RMSE of 0.728, and R^2 of 0.596. Its scatter plot reveals a tendency to underpredict at higher actual prices, creating visible deviations from the diagonal line.

Decision Tree Regressor improves upon this, with MAE at 0.410, MSE at 0.369, RMSE at 0.607, and R^2 at 0.719; its scatter plot clusters more tightly around the diagonal, although there is still noticeable spread, especially for higher actual prices.

Random Forest Regressor delivers the lowest errors with MAE of 0.328, MSE of 0.256, RMSE of 0.506 and the highest R^2 at 0.805, and its scatter plot shows the tightest clustering around the diagonal, indicating more reliable predictions across the entire price range. From both the numeric results and the visual plots.

So, Random Forest Regressor is the top performer for this dataset.