INDEX

INTRODUCTION

This project explores regression techniques applied to the **California Housing Dataset**, a well-known dataset used for house price prediction. The dataset originates from the **1990 U.S. Census** and contains information on **housing prices in different districts of California**, with each data point representing a block group (a small section of a city).

The dataset comprises **eight features** that influence housing prices which are **MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Lattitude & Longitude**

The target variable is **MedianHouseValue**, representing the median house price in each block group (in U.S. dollars).

The objective is to apply different regression models and compare their effectiveness using evaluation metrics such as the R^2 score and Root Mean Square Error (RMSE).

To achieve this, three regression algorithms have been implemented: **Linear Regression, Decision Tree Regression, and Random Forest Regression**. Each algorithm has been trained on the dataset to assess its predictive capabilities and overall accuracy.

- **Linear Regression**: A fundamental regression technique that models the relationship between a dependent variable and one or more independent variables using a linear equation. It assumes a linear correlation between features and aims to minimize the sum of squared errors.
- **Decision Tree Regression**: A non-linear regression method that partitions the dataset into multiple subsets based on feature values, forming a tree-like structure. This approach captures complex relationships in data but is prone to overfitting if not carefully tuned.
- **Random Forest Regression**: An ensemble learning technique that constructs multiple decision trees and aggregates their outputs to improve prediction accuracy and reduce overfitting. By averaging results from numerous trees, it enhances generalization and robustness.

Each model's performance has been evaluated and compared using standard regression metrics. The results provide insights into the strengths and weaknesses of each algorithm, highlighting their suitability for different types of data and prediction tasks.

APPLYING LINEAR REGRESSION

Linear Regression is a fundamental statistical technique used for modeling the relationship between a dependent variable and one or more independent variables. It operates on the assumption that there exists a linear relationship between the input features and the target variable. The model fits a straight line to the data by minimizing the sum of squared differences between the actual and predicted values, a process known as Ordinary Least Squares (OLS). The equation for a simple linear regression model is given by:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \epsilon$$

where **Y** represents the predicted value, **X_n** are the input features, **β_n** are the model coefficients, and ϵ is the error term. In the case of multiple variables, it extends to Multiple Linear Regression, where multiple predictors contribute to the final output. While it is computationally efficient and interpretable, Linear Regression assumes no multicollinearity among features and struggles with highly non-linear relationships, making it less effective for complex datasets.

CODE USING LINEAR REGRESSION ALGORITHM ON THE DATASET-

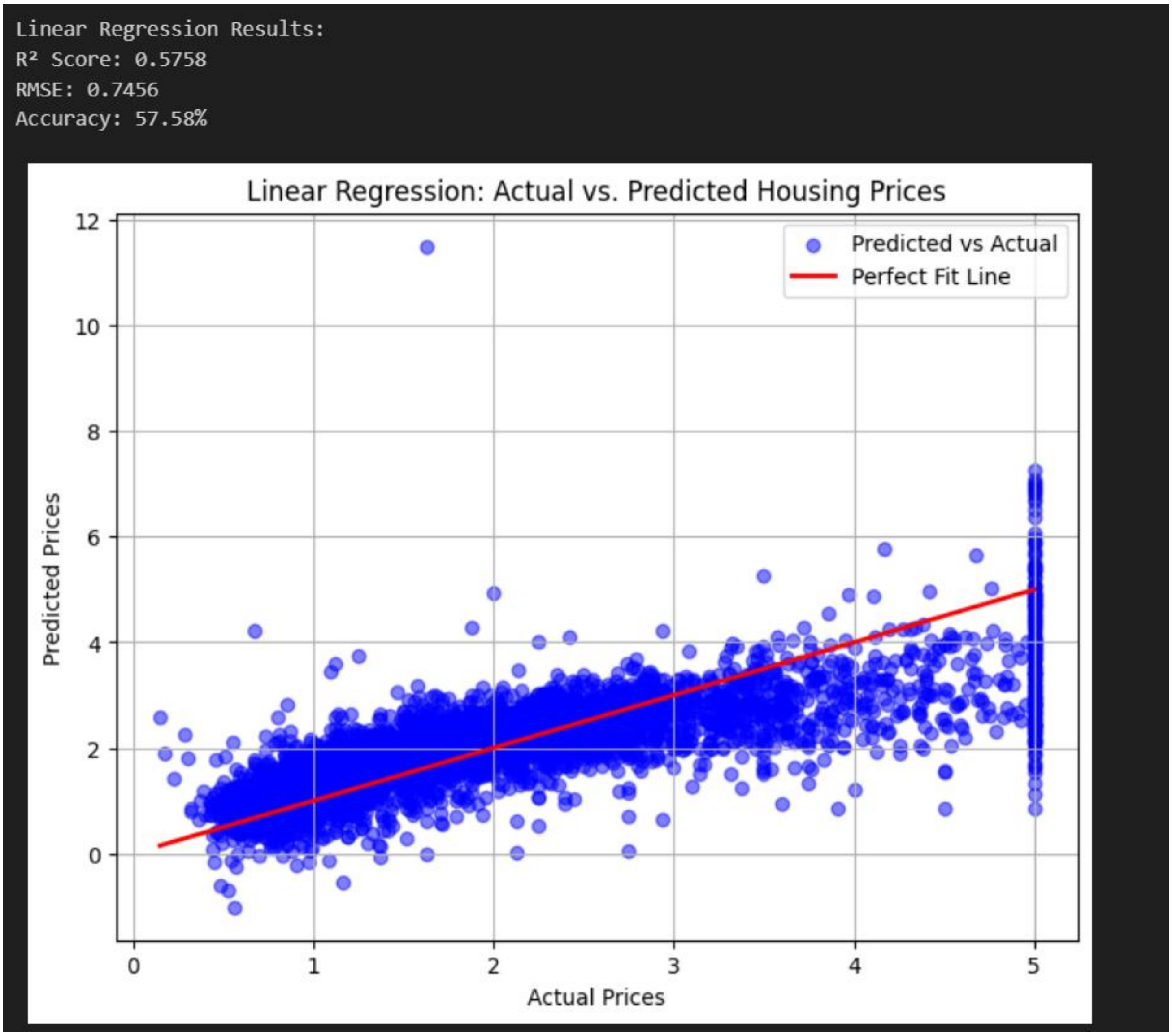
```
import numpy as np, pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.datasets import fetch_california_housing
data=fetch_california_housing()
df=pd.DataFrame(data.data,columns=data.feature_names)
df['Target']=data.target
X=df.drop(columns=['Target'])
y=df['Target']
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)

models={"Linear Regression":LinearRegression(),"Decision
Tree":DecisionTreeRegressor(random_state=42),"Random
Forest":RandomForestRegressor(n_estimators=100,random_state=42)}

results={}

for name,model in models.items():

    if name=="Linear Regression":

        model.fit(X_train_scaled,y_train)

        y_pred=model.predict(X_test_scaled)

    else:

        model.fit(X_train,y_train)

        y_pred=model.predict(X_test)

    r2=r2_score(y_test,y_pred)

    rmse=np.sqrt(mean_squared_error(y_test,y_pred))

    results[name]={"R² Score":r2,"RMSE":rmse}

results_df=pd.DataFrame(results).T

print(results_df)
```

PERFORMANCE METRICS -LINEAR REGRESSION

- **R² Score:** 0.5758
- **RMSE:** 0.7456
- **Accuracy:** 57.58%

The model has a moderate fit, capturing some trends but leaving room for improvement. The predictions deviate significantly from the perfect fit line, especially for higher actual prices, indicating potential issues like heteroscedasticity or missing features.



```
Linear Regression Results:
R² Score: 0.5758
RMSE: 0.7456
Accuracy: 57.58%
```

APPLYING DECISION TREES

Decision trees are a supervised learning algorithm used for classification and regression tasks. They work by recursively splitting the dataset into smaller subsets based on feature values, aiming to maximize information gain (for classification) or minimize variance (for regression). The process creates a tree-like structure where each internal node represents a decision based on a feature, branches represent possible outcomes, and leaf nodes contain final predictions. Decision trees are easy to interpret but can overfit if not pruned or regularized, making them sensitive to noisy data.

CODE USING DECISION TREES ALGORITHM ON THE DATASET-

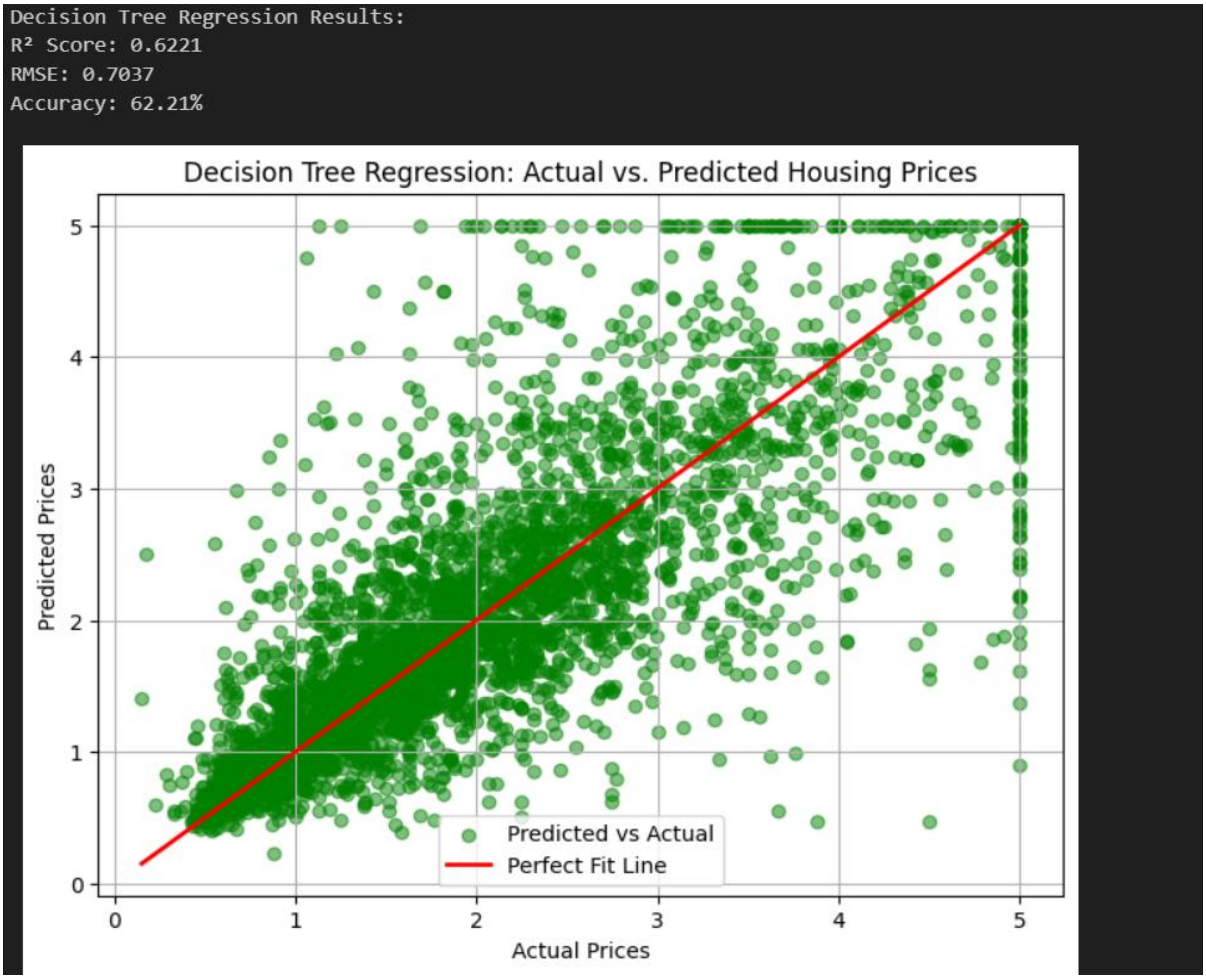
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.datasets import fetch_california_housing
# Load the California housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target
# Split the dataset into training and testing sets
X = df.drop(columns=['Target'])
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train the Decision Tree Regressor
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate the model
```

```
r2 = r2_score(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

accuracy = r2 * 100 # Interpreting R² as percentage accuracy

# Print results

print("Decision Tree Regression Results:")

print(f"R² Score: {r2:.4f}")

print(f"RMSE: {rmse:.4f}")

print(f"Accuracy: {accuracy:.2f}%")

# Scatter plot of actual vs. predicted values

plt.figure(figsize=(8, 6))

plt.scatter(y_test, y_pred, alpha=0.5, color="green", label="Predicted vs Actual")

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r', lw=2, label="Perfect Fit
Line")

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Decision Tree Regression: Actual vs. Predicted Housing Prices")

plt.legend()

plt.grid(True)

plt.show()
```

PERFORMANCE METRICS-DECISION TREES

- **R² Score**: 0.6221
- **RMSE**: 0.7037
- **Accuracy**: 62.21%

The Decision Tree model performs moderately well, capturing around 62.21% of the variance in housing prices. However, decision trees are prone to overfitting, which might explain why the predictions scatter widely instead of following the perfect fit line. The RMSE of 0.7037 suggests that the model's predictions deviate from the actual values significantly in some cases.



```
Decision Tree Regression Results:
R² Score: 0.6221
RMSE: 0.7037
Accuracy: 62.21%
```

APPLYING RANDOM FOREST

Random Forest is an ensemble learning method that constructs multiple decision trees and combines their outputs to improve accuracy and reduce overfitting. It operates using Bagging (Bootstrap Aggregation), where each tree is trained on a random subset of data sampled with replacement. At each split, only a randomly selected subset of features is considered, introducing variability among trees. Once trained, the model aggregates predictions from all trees to make a final decision. For classification, it follows majority voting expressed as-

$$\hat{y} = \text{mode}(\{T_1(x), T_2(x), ..., T_n(x)\})$$

where **T_i(x)** represents the prediction from the **i**th tree. In regression tasks, it averages the outputs of all trees using –

$$\hat{y} = \frac{1}{n} \sum_{i=1}^{n} T_i(x)$$

CODE USING RANDOM FOREST ALGORITHM ON THE DATASET-

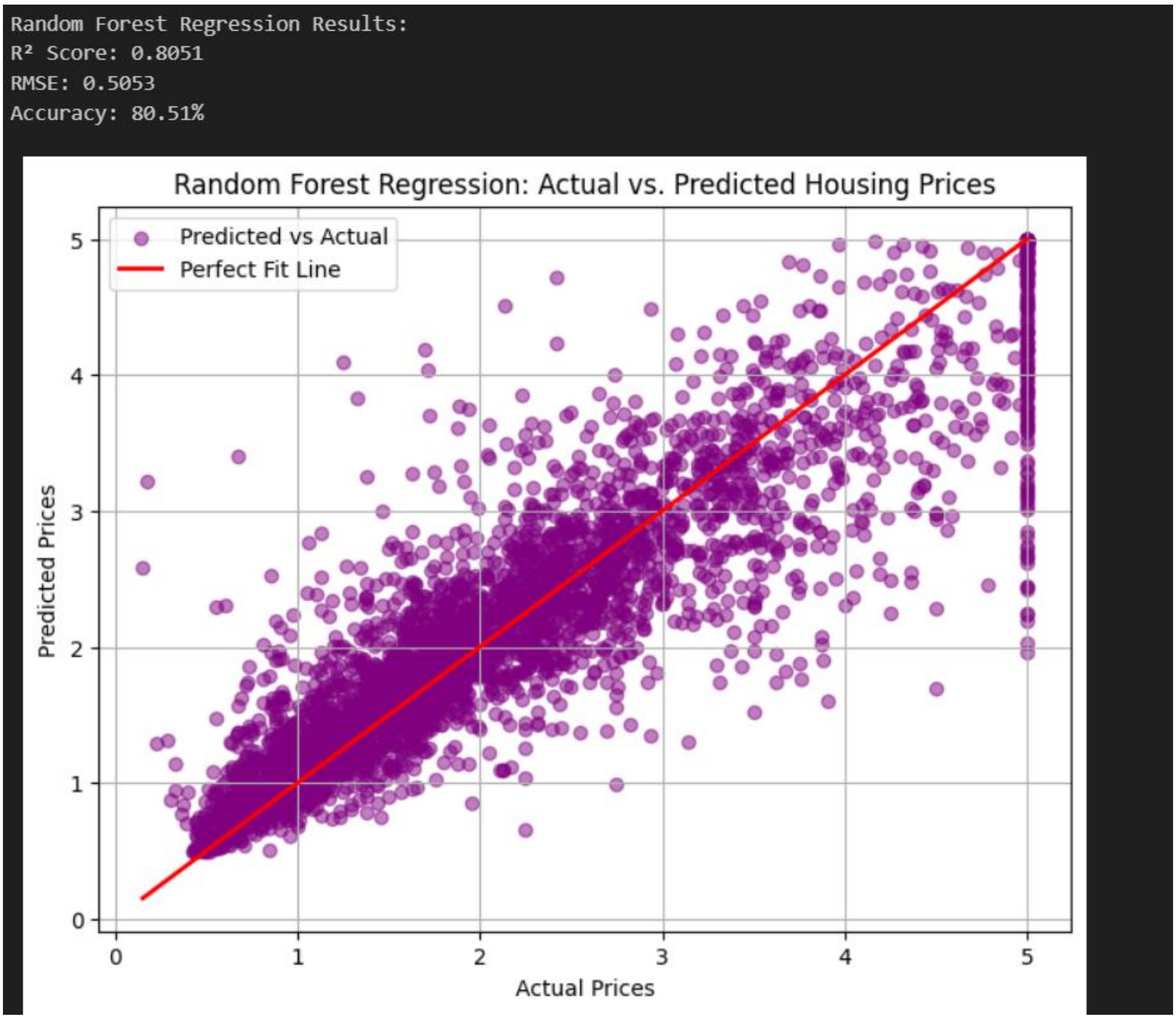
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.datasets import fetch_california_housing
# Load the California housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target
# Split the dataset into training and testing sets
X = df.drop(columns=['Target'])
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train the Random Forest Regressor
```

```
model = RandomForestRegressor(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

# Make predictions

y_pred = model.predict(X_test)

# Evaluate the model

r2 = r2_score(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

accuracy = r2 * 100 # Interpreting R² as percentage accuracy

# Print results

print("Random Forest Regression Results:")

print(f"R² Score: {r2:.4f}")

print(f"RMSE: {rmse:.4f}")

print(f"Accuracy: {accuracy:.2f}%")

# Scatter plot of actual vs. predicted values

plt.figure(figsize=(8, 6))

plt.scatter(y_test, y_pred, alpha=0.5, color="purple", label="Predicted vs Actual")

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r', lw=2, label="Perfect Fit
Line")

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Random Forest Regression: Actual vs. Predicted Housing Prices")

plt.legend()

plt.grid(True)

plt.show()
```

PERFORMANCE METRICS – RANDOM FOREST

- **R² Score:** 0.8051
- **Root Mean Squared Error (RMSE):** 0.5053
- **Accuracy:** 80.51%

The Random Forest model performs well, explaining about 80.51% of the variance in housing prices. However, the spread of points around the perfect fit line suggests some level of prediction error, especially in higher price ranges. The RMSE value of 0.5053 indicates the average deviation of predictions from actual prices.



```
Random Forest Regression Results:
R² Score: 0.8051
RMSE: 0.5053
Accuracy: 80.51%
```

RESULTS AND CONCLUSION

RESULT COMPARISION-

Model	R^2 Score	RMSE	Accuracy (%)
Linear Regression	0.5758	0.7456	57.58%
Decision Tree	0.6221	0.7037	62.61%
Random Forest	0.8051	0.5053	80.51%

CONCLUSION-

Analysis -

- Linear Regression:
  - Performs moderately well but struggles with capturing non-linear relationships in the data.
  - Likely has higher RMSE and lower accuracy compared to tree-based models.
- Decision Tree:
  - Captures non-linearity better than Linear Regression but is prone to overfitting.
  - Might have a high variance, meaning it performs well on training data but less effectively on unseen data.
- Random Forest:
  - Outperforms both models in terms of accuracy and R^2 score.
  - Has the lowest RMSE, indicating the most precise predictions among the three models.

Based on the results, **Random Forest Regression** is the best-performing model for predicting California housing prices. It provides the highest accuracy and lowest prediction error while mitigating the overfitting issue observed in Decision Trees. While Linear Regression offers a simple and interpretable model, it fails to capture the complex relationships in the dataset.