**Detailed Explanation of Random Forest Regressor’s Superior Performance Over Linear Regression**

The **Random Forest Regressor** outperforms **Linear Regression** in many cases, particularly when the dataset exhibits **non-linearity, feature interactions, and complex patterns**. Below is a detailed explanation of why Random Forest is performing better in your analysis.

**🔹 1. Comparison Between Linear Regression & Random Forest**

| **Feature** | **Linear Regression** | **Random Forest Regressor** |
| --- | --- | --- |
| **Assumptions** | Assumes a **linear** relationship between features and target variable | No assumptions; works well with **non-linear** data |
| **Feature Interaction** | Does **not** capture interactions between features | **Automatically captures** complex feature interactions |
| **Outliers Sensitivity** | Highly sensitive to **outliers**, affecting predictions | **Robust to outliers** due to bootstrapping |
| **Overfitting** | Prone to overfitting in high-dimensional datasets | **Controlled overfitting** using ensemble averaging |
| **Feature Importance** | Assumes all features contribute equally | Provides **feature importance rankings** |
| **Interpretability** | Easy to interpret | Harder to interpret than a simple linear model |

**🔹 2. Why is Random Forest Regressor Performing Better?**

✅ **Handles Non-Linearity**

* Linear Regression assumes a **straight-line** relationship between features and target.
* Random Forest **automatically captures** complex, non-linear patterns.

✅ **Reduces Overfitting Through Bagging**

* Linear Regression **fits one equation** to all data, which may overfit when features are not properly selected.
* Random Forest uses **multiple decision trees**, averaging their outputs to avoid overfitting.

✅ **Feature Importance Helps Identify Key Predictors**

* Linear Regression assigns **coefficients** to features, assuming all features contribute **linearly**.
* Random Forest calculates **feature importance**, helping to **eliminate irrelevant features**.

✅ **Resistant to Outliers & Noise**

* Linear Regression is sensitive to **outliers**, as they pull the regression line.
* Random Forest is built using **multiple trees**, reducing the effect of extreme values.

✅ **Handles Missing Data Better**

* If missing values exist, Linear Regression requires **imputation** (mean, median, mode).
* Random Forest **can work with missing data**, making it **more robust**.

**🔹 3. Further Improving Random Forest Model**

To improve the **Random Forest Regressor’s** performance further, we can fine-tune the parameters using **GridSearchCV**.

from sklearn.model\_selection import GridSearchCV

# Hyperparameter tuning for Random Forest

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

grid\_search = GridSearchCV(RandomForestRegressor(random\_state=42), param\_grid, cv=5, n\_jobs=-1, scoring='r2')

grid\_search.fit(X\_train, y\_train)

# Best model after tuning

best\_rf = grid\_search.best\_estimator\_

y\_pred\_best\_rf = best\_rf.predict(X\_test)

# Evaluate Improved Random Forest

mse\_best\_rf = mean\_squared\_error(y\_test, y\_pred\_best\_rf)

r2\_best\_rf = r2\_score(y\_test, y\_pred\_best\_rf)

adjusted\_r2\_best\_rf = 1 - (1 - r2\_best\_rf) \* (len(y\_test) - 1) / (len(y\_test) - X\_test.shape[1] - 1)

print("\nOptimized Random Forest Performance:")

print(f'Mean Squared Error: {mse\_best\_rf}')

print(f'R-squared Score: {r2\_best\_rf}')

print(f'Adjusted R-squared Score: {adjusted\_r2\_best\_rf}')

**🔹 4. Conclusion**

✔ **Random Forest performs better because it captures non-linearity, reduces overfitting, and handles missing data & outliers effectively.**  
✔ **If interpretability is needed, feature importance can be extracted.**  
✔ **Further improvements can be made through hyperparameter tuning.**

**Boston Housing Dataset: Full Description**

The **Boston Housing Dataset** is a well-known dataset used for regression tasks, particularly in predicting housing prices. It contains **506 observations** and **13 numerical features**, along with a target variable representing the median value of owner-occupied homes.

**🔹 Columns in the Boston Dataset**

Below is a description of each feature in the dataset:

| **Column Name** | **Description** |
| --- | --- |
| **CRIM** | Per capita crime rate by town |
| **ZN** | Proportion of residential land zoned for large lots (over 25,000 sq. ft.) |
| **INDUS** | Proportion of non-retail business acres per town |
| **CHAS** | Charles River dummy variable (1 if tract bounds river, 0 otherwise) |
| **NOX** | Nitrogen oxide concentration (pollution levels, parts per 10 million) |
| **RM** | Average number of rooms per dwelling |
| **AGE** | Proportion of owner-occupied units built before 1940 |
| **DIS** | Weighted distance to five major Boston employment centers |
| **RAD** | Index of accessibility to radial highways |
| **TAX** | Property tax rate per $10,000 |
| **PTRATIO** | Pupil-teacher ratio by town |
| **B** | Proportion of Black population (1000(Bk - 0.63)² where Bk is the proportion of Black residents) |
| **LSTAT** | Percentage of lower-income population |
| **MEDV** (Target) | Median home value in $1000s |

**🔹 How the Features Affect House Prices**

* **RM (Avg. Number of Rooms):** A higher number of rooms generally increases home value.
* **LSTAT (Lower Income %):** Higher LSTAT correlates with **lower** home values.
* **CRIM (Crime Rate):** Higher crime rates usually lead to **lower** home prices.
* **PTRATIO (Pupil-Teacher Ratio):** A **higher** ratio suggests fewer teachers per student, leading to **lower** home values.
* **DIS (Distance to Employment Centers):** Houses further away tend to be **cheaper**.
* **TAX (Property Tax Rate):** Higher taxes **may** decrease demand and affect home values.
* **CHAS (Proximity to Charles River):** Being near the river **increases** home values.

**🔹 Using the Dataset in Python**

The Boston dataset was previously available in **sklearn.datasets** but has been removed. You can load it using **statmodels** or **fetch it from OpenML**:

python

CopyEdit

from sklearn.datasets import fetch\_openml

# Load dataset from OpenML

boston = fetch\_openml(name='boston', version=1, as\_frame=True)

df = boston.frame # Convert to Pandas DataFrame

# Display first few rows

print(df.head())