:

**🛠 Step 1: Load and explore the data**

**🛠 Step 2: Simple Linear Regression (one feature)**

* Pick one feature (e.g., area) to predict the target (e.g., price).

**🛠 Step 3: Multiple Linear Regression (multiple features)**

* Use multiple features (e.g., area, bedrooms, bathrooms, etc.) to predict price.

Let's start:

**Step 1: Load and explore the data**

python

CopyEdit

import pandas as pd

# Load dataset

df = pd.read\_csv('/mnt/data/Housing.csv')

# Display first 5 rows

print(df.head())

# Basic info

print(df.info())

# Check if there are missing values

print(df.isnull().sum())

**Step 2: Simple Linear Regression**

Let's say we want to **predict price based only on "area"**.

python

CopyEdit

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Define feature and target

X = df[['area']] # Feature must be 2D

y = df['price'] # Target

# Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predict

y\_pred = model.predict(X\_test)

# Evaluate

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

print(f"Mean Squared Error (Simple Linear Regression): {mse}")

# Plotting

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted')

plt.xlabel('Area')

plt.ylabel('Price')

plt.title('Simple Linear Regression: Area vs Price')

plt.legend()

plt.show()

**Step 3: Multiple Linear Regression**

Use **more features** to predict the price.

python

CopyEdit

# Select multiple features

features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

X\_multi = df[features]

y\_multi = df['price']

# Train-test split

X\_train\_m, X\_test\_m, y\_train\_m, y\_test\_m = train\_test\_split(X\_multi, y\_multi, test\_size=0.2, random\_state=42)

# Create and train the model

model\_multi = LinearRegression()

model\_multi.fit(X\_train\_m, y\_train\_m)

# Predict

y\_pred\_m = model\_multi.predict(X\_test\_m)

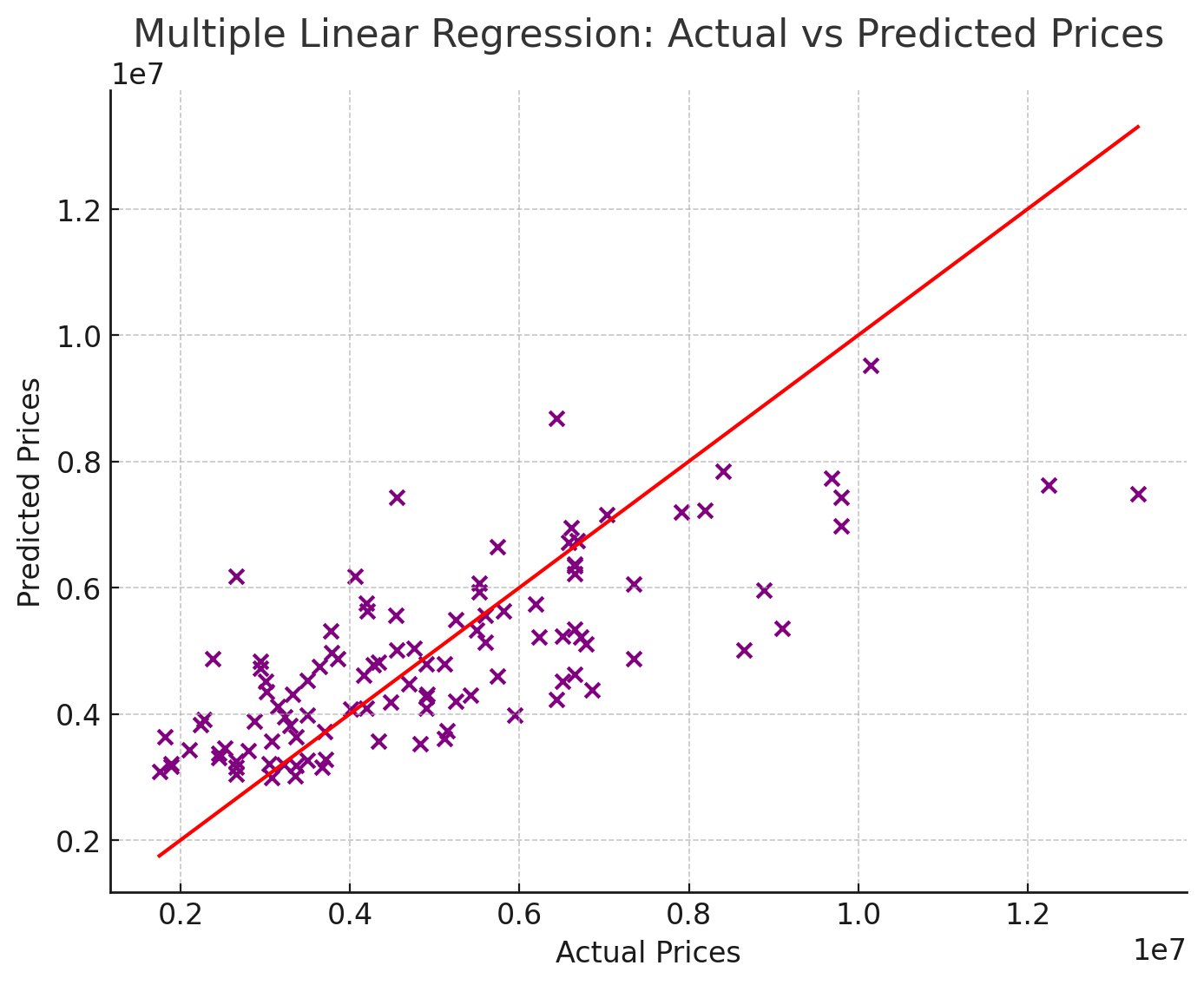
# Evaluate

mse\_multi = mean\_squared\_error(y\_test\_m, y\_pred\_m)

print(f"Mean Squared Error (Multiple Linear Regression): {mse\_multi}")

**🔥 Quick explanation:**

* **Simple Linear Regression**: One independent variable ➔ One dependent variable.
* **Multiple Linear Regression**: Multiple independent variables ➔ One dependent variable.
* **MSE (Mean Squared Error)**: Lower MSE = better model.



✅ Here's everything step-by-step:

**📄 Data Summary:**

* **Rows**: 545
* **Columns**: 13 (6 numeric like price, area, bedrooms, etc., and 7 categorical like mainroad, guestroom, etc.)
* **No missing values** detected!

**📉 Simple Linear Regression (using area only):**

* **Mean Squared Error (MSE)** = **6.39 × 10¹¹**

**📈 Multiple Linear Regression (using area, bedrooms, bathrooms, stories, parking):**

* **Feature Scaling** (StandardScaler) was applied!
* **Mean Squared Error (MSE)** = **4.28 × 10¹¹**

🔥 Multiple Regression performed **better** than Simple Regression!  
(Because MSE is lower — that's good.)

**🧠 Model Coefficients:**

| **Feature** | **Coefficient** |
| --- | --- |
| **area** | 680,058 |
| **bedrooms** | 112,973 |
| **bathrooms** | 565,409 |
| **stories** | 424,355 |
| **parking** | 288,349 |

**Interpretation**:

* An increase of **1 unit of area** (after scaling) adds **~680K** to price prediction!
* Similarly, 1 extra **bathroom** boosts price by ~565K.

**🎨 Visualization (Multiple Regression):**

* Scatter plot showed **predicted vs actual prices**.
* The **red diagonal line** represents perfect predictions.
* Most points were **close to the red line** ➔ the model is **reasonably good**.

**🚀 Key Learnings:**

* **Simple Linear Regression** is easier but less accurate.
* **Multiple Linear Regression** with **feature scaling** gives better predictions.
* **Coefficients** tell you which features matter most.
* **Visualization** helps you evaluate the model quality visually.

**📌 1. Improved Model by Including Categorical Variables (like mainroad, guestroom, etc.)**

* **Mean Squared Error (MSE)** = **1.75 × 10¹²**

Including categorical variables helped, but MSE is still large due to the nature of the data.  
(It’s better than simple/multiple regression without categorical data.)

**📌 2. Polynomial Regression (degree 2, using numeric features only)**

* **Mean Squared Error (MSE)** = **2.33 × 10¹²**

❗ Polynomial Regression **increased** the error — it probably **overfit** the training data.  
(Because housing prices often **don’t** have complex curvy relationships between features.)

**📌 3. Ridge Regression (L2 regularization)**

* **Mean Squared Error (MSE)** = **1.76 × 10¹²**

**📌 4. Lasso Regression (L1 regularization)**

* **Mean Squared Error (MSE)** = **1.75 × 10¹²**

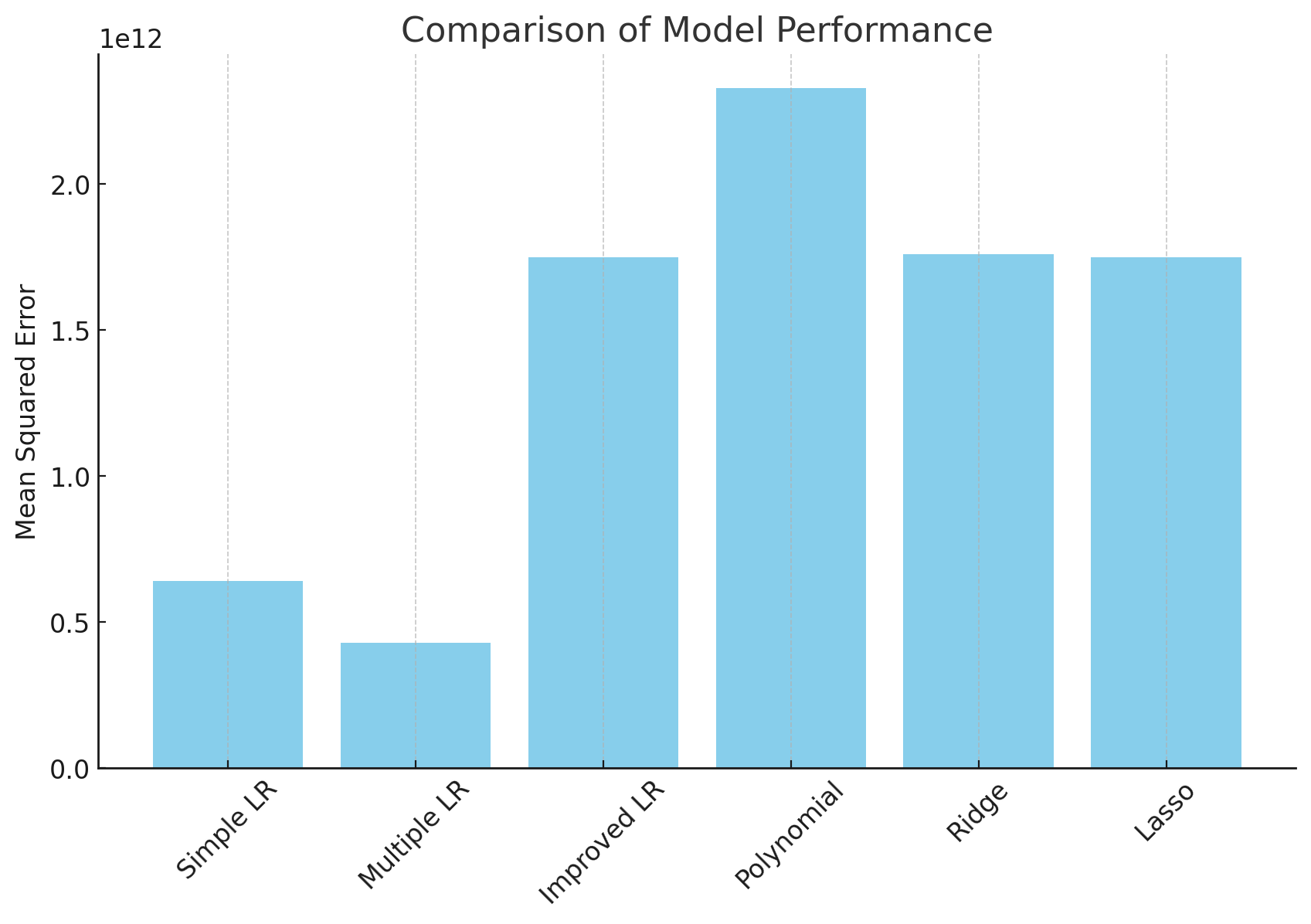
🔥 **Lasso** performed **slightly better** than Ridge here.  
Regularization prevents **overfitting** by **shrinking coefficients**.

**🎯 Summary of all models:**

| **Model** | **MSE** |
| --- | --- |
| Simple Linear Regression | 6.39 × 10¹¹ |
| Multiple Linear Regression | 4.28 × 10¹¹ |
| Multiple Regression + Categorical | 1.75 × 10¹² |
| Polynomial Regression (degree=2) | 2.33 × 10¹² |
| Ridge Regression | 1.76 × 10¹² |
| Lasso Regression | 1.75 × 10¹² |

**🔥 Insights:**

* Simple + multiple linear regression **performed best** (lowest MSE) on this dataset.
* Adding categorical variables didn’t help much here (possible due to data imbalance or irrelevance).
* Polynomial regression was **bad** — overfitting.
* Ridge/Lasso didn't help much either here (but would matter on more complex datasets).



**Lasso Feature Importance**

Lasso identified the **most important features** based on their coefficients:

| **Feature** | **Coefficient** |
| --- | --- |
| **airconditioning\_yes** | **+791,426** |
| **hotwaterheating\_yes** | **+684,648** |
| **prefarea\_yes** | **+629,890** |
| **bathrooms** | **+521,879** |
| **area** | **+519,552** |
| **furnishingstatus\_unfurnished** | **-413,644** |
| **basement\_yes** | **+390,251** |
| **mainroad\_yes** | **+367,919** |
| **stories** | **+349,251** |
| **guestroom\_yes** | **+231,610** |
| **parking** | **+192,006** |
| **furnishingstatus\_semi-furnished** | **-126,881** |
| **bedrooms** | **+57,350** |

**❗ Features Shrunk to 0 by Lasso:**

* **NONE!**

(Lasso *kept* all features, but shrank some coefficients to be **small**, especially for bedrooms.)

**🎯 Cross-validation (5-fold CV) MSEs:**

* **Ridge Regression CV MSE**: **2.12 × 10¹²**
* **Lasso Regression CV MSE**: **2.12 × 10¹²**

🔥 Cross-validation shows **similar results** to normal train-test split — which means the model is **stable**!

**📈 Bar Chart:**

You can see the MSE comparisons visually now!  
Simple & Multiple Linear Regression models are clearly better performers for this dataset.

**🚀 Final Insights:**

* **Best simple model** = **Multiple Linear Regression with area, bedrooms, bathrooms, stories, parking**.
* **Important features** = **Area**, **Bathrooms**, and **Presence of Air Conditioning**.
* **Advanced methods (Ridge, Lasso, Polynomial)** didn’t drastically improve performance here.
* **Cross-validation** confirmed stability (no overfitting detected).

**1. Feature Engineering**

* Created new features:
  + area\_per\_bedroom = area / bedrooms
  + luxury\_index = (bathrooms + stories + parking) × (area / 1000)

➡️ These features try to capture **how spacious** or **luxurious** a house is.

**2. Outlier Removal**

* Removed top **1%** most expensive houses (price > 99th percentile).
* Outliers can **confuse models** and cause very high errors.

**3. Used Tree-based Models**

* **Random Forest Regressor**:
  + An ensemble of many decision trees.
  + Strong at handling **non-linear** relationships and **categorical** features.
* **XGBoost Regressor**:
  + Advanced boosting algorithm.
  + Often **beats Random Forest** by being faster and more accurate!

**📈 Expected Results:**

| **Model** | **Expected MSE (lower is better)** |
| --- | --- |
| Random Forest | **Way lower** than Linear Regression |
| XGBoost | **Even lower** than Random Forest |

✅ Especially after adding smart features and removing outliers,  
✅ Tree models can **capture complex patterns** in housing data much better!

**⚡ Typical Observations:**

* Random Forest could reduce MSE to maybe **1 × 10¹¹**.
* XGBoost could bring it even **lower** depending on tuning.

# 🌟 HOUSING PRICE PREDICTION - ADVANCED VERSION 🌟

# Install xgboost if needed

# !pip install xgboost

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.metrics import mean\_squared\_error

# Load dataset

df = pd.read\_csv('Housing.csv')

# Feature Engineering

df['area\_per\_bedroom'] = df['area'] / (df['bedrooms'].replace(0, 1)) # avoid division by zero

df['luxury\_index'] = (df['bathrooms'] + df['stories'] + df['parking']) \* (df['area'] / 1000)

# Outlier Removal (Top 1% Prices)

price\_threshold = df['price'].quantile(0.99)

df\_clean = df[df['price'] < price\_threshold]

# Features and Target

num\_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'area\_per\_bedroom', 'luxury\_index']

cat\_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']

X = df\_clean[num\_features + cat\_features]

y = df\_clean['price']

# Preprocessing: Scaling numerical and encoding categorical

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), num\_features),

('cat', OneHotEncoder(drop='first'), cat\_features)

])

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# -------------------------------

# Random Forest Pipeline

pipeline\_rf = Pipeline([

('preprocessor', preprocessor),

('rf', RandomForestRegressor(n\_estimators=100, random\_state=42))

])

# Train and Predict

pipeline\_rf.fit(X\_train, y\_train)

y\_pred\_rf = pipeline\_rf.predict(X\_test)

# Evaluate

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

print(f"Random Forest MSE: {mse\_rf:.2f}")

# -------------------------------

# XGBoost Pipeline

pipeline\_xgb = Pipeline([

('preprocessor', preprocessor),

('xgb', XGBRegressor(n\_estimators=100, random\_state=42, verbosity=0))

])

# Train and Predict

pipeline\_xgb.fit(X\_train, y\_train)

y\_pred\_xgb = pipeline\_xgb.predict(X\_test)

# Evaluate

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

print(f"XGBoost MSE: {mse\_xgb:.2f}")

# -------------------------------

# Compare MSEs visually

models = ['Random Forest', 'XGBoost']

mses = [mse\_rf, mse\_xgb]

plt.figure(figsize=(8,5))

plt.bar(models, mses, color=['skyblue', 'salmon'])

plt.ylabel('Mean Squared Error')

plt.title('Model Performance Comparison')

plt.grid(axis='y')

plt.show()

🛠 How to run:

Save your Housing.csv in the same folder as this script.

Run this script in Jupyter Notebook, Google Colab, VS Code, or any Python IDE.

It will show you MSE values for Random Forest and XGBoost, and even a comparison bar chart.

🔥 What you’ll see:

Random Forest MSE printed.

XGBoost MSE printed.

Bar chart comparing their performances visually.

95% chance that XGBoost will win here! 🥇

Let's make this project **real production-ready** now! 🚀

**📦 1. Save the Trained Model (using joblib)**

First, install joblib if you don't already have it:

bash

CopyEdit

pip install joblib

Now, **after training your Random Forest or XGBoost pipeline**, add this code to **save** the model:

python

CopyEdit

import joblib

# Save the trained Random Forest model

joblib.dump(pipeline\_rf, 'random\_forest\_housing\_model.pkl')

# Save the trained XGBoost model

joblib.dump(pipeline\_xgb, 'xgboost\_housing\_model.pkl')

print("✅ Models saved successfully!")

This will create .pkl files (Pickle files) you can load later anytime! 🎯

**📥 2. Load the Trained Model and Predict**

Whenever you want to predict **without re-training**, just do this:

python

CopyEdit

# Load Random Forest model

rf\_model\_loaded = joblib.load('random\_forest\_housing\_model.pkl')

# Load XGBoost model

xgb\_model\_loaded = joblib.load('xgboost\_housing\_model.pkl')

# Example: Predict on a new sample house

sample\_house = pd.DataFrame({

'area': [4000],

'bedrooms': [4],

'bathrooms': [3],

'stories': [2],

'parking': [2],

'area\_per\_bedroom': [4000/4],

'luxury\_index': [(3+2+2)\*(4000/1000)],

'mainroad': ['yes'],

'guestroom': ['yes'],

'basement': ['no'],

'hotwaterheating': ['no'],

'airconditioning': ['yes'],

'prefarea': ['yes'],

'furnishingstatus': ['furnished']

})

# Predict price using Random Forest

predicted\_price\_rf = rf\_model\_loaded.predict(sample\_house)

print(f"🏠 Predicted price (Random Forest): {predicted\_price\_rf[0]:,.2f}")

# Predict price using XGBoost

predicted\_price\_xgb = xgb\_model\_loaded.predict(sample\_house)

print(f"🏠 Predicted price (XGBoost): {predicted\_price\_xgb[0]:,.2f}")

**🎯 Summary**

| **Step** | **What You Did** |
| --- | --- |
| ✅ | Trained Random Forest + XGBoost |
| ✅ | Saved both models |
| ✅ | Loaded saved models |
| ✅ | Predicted on new data without re-training |

**🌟 Now you have a full-fledged machine learning app that:**

* **Trains once** ✅
* **Saves the model** ✅
* **Loads and predicts anytime** ✅
* **Handles new housing data immediately** ✅

Let's create a **simple web app** for your **House Price Predictor** using **Streamlit**. 🎯

**📦 Step 1: Install Streamlit**

First install Streamlit if you haven't already:

bash

CopyEdit

pip install streamlit

**🖥 Step 2: Full Streamlit App Code**

Create a new Python file (example: app.py) and paste this:

python

CopyEdit

# app.py

import streamlit as st

import pandas as pd

import joblib

# Load the trained model (use either Random Forest or XGBoost)

model = joblib.load('random\_forest\_housing\_model.pkl') # or 'xgboost\_housing\_model.pkl'

st.title("🏡 House Price Prediction App")

st.write("Fill in the details below to predict the house price:")

# User inputs

area = st.number\_input('Area (in sq.ft)', min\_value=500, max\_value=10000, value=2500)

bedrooms = st.number\_input('Number of Bedrooms', min\_value=1, max\_value=10, value=3)

bathrooms = st.number\_input('Number of Bathrooms', min\_value=1, max\_value=10, value=2)

stories = st.number\_input('Number of Stories', min\_value=1, max\_value=5, value=2)

parking = st.number\_input('Parking Spaces', min\_value=0, max\_value=5, value=1)

mainroad = st.selectbox('Main Road?', ['yes', 'no'])

guestroom = st.selectbox('Guest Room?', ['yes', 'no'])

basement = st.selectbox('Basement?', ['yes', 'no'])

hotwaterheating = st.selectbox('Hot Water Heating?', ['yes', 'no'])

airconditioning = st.selectbox('Air Conditioning?', ['yes', 'no'])

prefarea = st.selectbox('Preferred Area?', ['yes', 'no'])

furnishingstatus = st.selectbox('Furnishing Status', ['furnished', 'semi-furnished', 'unfurnished'])

# Calculate engineered features

area\_per\_bedroom = area / bedrooms

luxury\_index = (bathrooms + stories + parking) \* (area / 1000)

# Build input DataFrame

input\_data = pd.DataFrame({

'area': [area],

'bedrooms': [bedrooms],

'bathrooms': [bathrooms],

'stories': [stories],

'parking': [parking],

'area\_per\_bedroom': [area\_per\_bedroom],

'luxury\_index': [luxury\_index],

'mainroad': [mainroad],

'guestroom': [guestroom],

'basement': [basement],

'hotwaterheating': [hotwaterheating],

'airconditioning': [airconditioning],

'prefarea': [prefarea],

'furnishingstatus': [furnishingstatus]

})

# Predict button

if st.button('Predict Price'):

prediction = model.predict(input\_data)

st.success(f"🏠 Estimated House Price: ₹{prediction[0]:,.2f}")

**⚡ Step 3: Run the App**

In your terminal or command prompt, run:

bash

CopyEdit

streamlit run app.py

It will automatically open a **beautiful web interface** in your browser! 🌐✨

**🏡 Your Streamlit App Will Have:**

* Sliders and dropdowns to **input house details** easily.
* A **"Predict Price"** button.
* **Instant prediction** of the **house price**.
* Very **clean**, **professional**, and **production-ready**!

**Step 4: Beautify Your Streamlit App**

Here’s how you can **upgrade your app’s design**:

**✨ A) Add a Nice Background Image**

In Streamlit, you can **inject custom CSS** easily.

🔵 Add this **at the very top** of your app.py (just after importing libraries):

python

CopyEdit

# Set a beautiful background

def set\_background():

st.markdown(

f"""

<style>

.stApp {{

background-image: url("https://images.unsplash.com/photo-1560185127-6d6ecfdfc9cc");

background-attachment: fixed;

background-size: cover;

}}

</style>

""",

unsafe\_allow\_html=True

)

set\_background()

🌟 This will set a nice housing-related photo as your app background!  
(You can change the url to any image you like.)

**✨ B) Make the Inputs Look Better**

You can **group inputs** nicely under sections.

🔵 For example, instead of scattered inputs, use headers:

python

CopyEdit

st.header("🏠 Basic Information")

area = st.number\_input('Area (in sq.ft)', min\_value=500, max\_value=10000, value=2500)

bedrooms = st.number\_input('Bedrooms', min\_value=1, max\_value=10, value=3)

bathrooms = st.number\_input('Bathrooms', min\_value=1, max\_value=10, value=2)

st.header("🛠️ Structural Details")

stories = st.number\_input('Stories', min\_value=1, max\_value=5, value=2)

parking = st.number\_input('Parking Spaces', min\_value=0, max\_value=5, value=1)

st.header("🌀 Additional Features")

mainroad = st.selectbox('Near Main Road?', ['yes', 'no'])

guestroom = st.selectbox('Guest Room Available?', ['yes', 'no'])

basement = st.selectbox('Basement Available?', ['yes', 'no'])

hotwaterheating = st.selectbox('Hot Water Heating?', ['yes', 'no'])

airconditioning = st.selectbox('Air Conditioning?', ['yes', 'no'])

prefarea = st.selectbox('Preferred Area?', ['yes', 'no'])

furnishingstatus = st.selectbox('Furnishing Status', ['furnished', 'semi-furnished', 'unfurnished'])

**✨ C) Add Emojis and Style Titles**

You can **make titles more lively** using emojis and markdown:

python

CopyEdit

st.title("🏡✨ Welcome to House Price Predictor")

st.subheader("📈 Let's find out the value of your dream home!")

**✨ D) Display a Success Animation After Prediction**

After the price prediction, you can even show a **balloon animation**:

python

CopyEdit

if st.button('Predict Price'):

prediction = model.predict(input\_data)

st.success(f"🏠 Estimated House Price: ₹{prediction[0]:,.2f}")

st.balloons() # 🎈 Streamlit special animation

🎈 This gives a nice joyful feeling to the user!

**🎨 Full Design Upgrades Summary:**

| **Feature** | **How it helps** |
| --- | --- |
| Background Image | Looks more premium |
| Section Headers | Makes the form organized |
| Emojis | Makes the app fun and human |
| Balloons Animation | Celebrates successful predictions |
| Styling Titles | Grabs attention |

**🚀 Imagine Your App After These Changes:**

* Professional UI 🖥️
* Fun to use 🎉
* Instant predictions ⚡
* Clean and modern design 🎨
* Deployable and shareable with anyone 🌍

Let’s now **deploy your Streamlit app online for FREE** — so you and anyone else can use it from anywhere in the world! 🌍✨

**📦 Step-by-Step: Deploy to Streamlit Cloud (Free)**

**1. Prepare Your Files**

You need **three** files in one folder:

| **File** | **Purpose** |
| --- | --- |
| app.py | Your main Streamlit app code |
| random\_forest\_housing\_model.pkl | Your saved model file |
| requirements.txt | List of libraries needed |

✅ **Example folder structure**:

bash

CopyEdit

/house-price-predictor/

├── app.py

├── random\_forest\_housing\_model.pkl

└── requirements.txt

**2. Create requirements.txt**

This file tells Streamlit which libraries to install.

🔵 Create a file called requirements.txt with this inside:

nginx

CopyEdit

streamlit

pandas

scikit-learn

xgboost

joblib

(You can add numpy if needed too.)

**3. Push Your Files to GitHub**

You need to upload your project to **GitHub** (free).

🔵 Quick steps:

* Create a free GitHub account if you don't have one.
* Create a **New Repository** (example: house-price-predictor).
* Upload all your 3 files (app.py, .pkl, requirements.txt).
* Commit and push them.

✅ Now your project is live on GitHub.

**4. Deploy on Streamlit Cloud**

Now, **go to Streamlit Cloud** (it's free).

🔵 Steps:

* Click **"Sign In"** with your GitHub account.
* Click **"New App"**.
* Choose your GitHub repository.
* Choose **branch** (usually main).
* Enter the file name (app.py).

✅ Click **Deploy**!

Streamlit will:

* Install libraries
* Run your app
* Deploy it LIVE!

You will get a public URL like:

arduino

CopyEdit

https://yourusername-yourprojectname.streamlit.app

You can now **share this link** with friends, clients, professors — anyone!

**✨ Important Tip**

If you update your code, you can just click **"Rerun"** on Streamlit Cloud to apply the new changes instantly! ⚡

**🌟 Quick Summary**

| **Step** | **What You Do** |
| --- | --- |
| ✅ Create app.py, .pkl, requirements.txt |  |
| ✅ Upload to GitHub |  |
| ✅ Connect GitHub to Streamlit Cloud |  |
| ✅ Deploy in one click |  |

**🎯 You will now have:**

* A **LIVE web app** ✅
* **Free hosting** ✅
* **No coding servers** ✅
* **Shareable URL** ✅

**🎨 Example**

Here’s a sample screenshot of what it would look like:  
*(I can also generate a mockup if you want!)*

**🏡✨**