Student Name:- Shah Rishabh Alpeshbhai
Enrolment Number:- 23SS02IT181
Subject Name:- DATA SCIENCE
Subject Code:- SSCA3021

Project Title

House Price Prediction using Data Science Techniques

Domain:- Data / Exploratory Data Analysis (EDA)

Project Abstract

This project aims to analyze residential housing data and build a machine-learning model capable of predicting house prices accurately. The dataset contains property characteristics such as number of rooms, bathrooms, area (sqft), construction quality, renovation history, and location coordinates.

The workflow includes importing and cleaning data, handling missing values, feature engineering, EDA, preprocessing, model training, and evaluation using RMSE and R^2. After experimenting with multiple algorithms, the best model was identified based on performance metrics.

Insights from this analysis help understand the influence of living area, location, renovation, and other features on property prices. The final model can assist real-estate companies, investors, and customers in estimating property values and making informed decisions.

Dataset Description

Source: Kaggle – House Price Dataset (King County Houses)
Dataset Name: kc_house_data.csv
Total Records: ~21,613 house records
Target Variable: price (House Sale Price)

Data Preprocessing

Removed duplicate entries
Converted date column to datetime format
Derived year_sold, month_sold, house_age, and was_renovated
Checked missing values and handled rare missing cases
Scaling applied on numeric features
Encoded categorical feature (zipcode) using Ordinal Encoding

Exploratory Data Analysis

Steps Performed

✓ Distribution analysis (price, sqft, rooms)
✓ Scatter plots of area vs price
✓ Boxplots for bedrooms, bathrooms effect on price
✓ Heatmap for feature correlation
✓ Price comparison across zipcodes

Key Observations

✓ Larger living area strongly increases price
✓ Waterfront houses are high-value properties
✓ Renovated houses sell at higher prices
✓ Location (zipcode) is a major determining factor
✓ Bedrooms alone do not determine price — area & grade matter more

Machine Learning Models Used (10 Models)

Sr	Model	Category	Key Idea	Why Used
1	Linear Regression	Linear	Best-fit line	Baseline model
2	Ridge Regression	L2 Regularized	Reduce overfitting	Handles multicollinearity
3	Lasso Regression	L1 Regularized	Shrinks weak features	Feature selection

Sr	Model	Category	Key Idea	Why Used
4	Elastic Net	L1 + L2	Hybrid regularization	Balanced control
5	Decision Tree	Tree	Rule-based splits	Non-linear learning
6	Random Forest	Ensemble Bagging	Many trees, average result	High accuracy
7	Gradient Boosting	Ensemble Boosting	Correct errors iteratively	Very powerful
8	HistGradientBoosting	Fast Boosting	Histogram-based boosting	Very efficient & accurate
9	AdaBoost	Boosting	Weight difficult samples	Boosting weak learners
10	KNN	Instance-based	Neighbor similarity	Distance-based prediction

Best Model from Project

Model	Reason
Gradient Boosting / Random Forest	Handles non-linear relationships + location factor + interaction between features

Cell – Importing Libraries & Environment Setup

- Imported essential Python libraries like **Pandas, NumPy, Matplotlib, Seaborn** for data handling, numerical operations, and visualization.
- Loaded key **Machine Learning libraries from Scikit-Learn** including model selection, preprocessing tools, and regression algorithms.
- Included **warning filters and display settings** to make output clean & readable during analysis.
- This step prepares the Jupyter environment by loading all tools required for **data cleaning, feature engineering, visualization, and model training**.

```
[1]: # Mini Project: House Price Prediction

[2]: # Importing Required Libraries & Configuration

[3]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import os, pickle, zipfile

[4]: from sklearn.model_selection import train_test_split, KFold, cross_val_score

[5]: from sklearn.compose import ColumnTransformer
     from sklearn.impute import SimpleImputer
     from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder, FunctionTransformer

[6]: from sklearn.metrics import mean_squared_error, r2_score

[7]: from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor, HistGradientBoostingRegressor
     from sklearn.neighbors import KNeighborsRegressor

[8]: from sklearn.neighbors import KNeighborsRegressor

[9]: # Ignore unnecessary warnings and set display settings
     import warnings
     warnings.filterwarnings("ignore")

[10]: pd.set_option('display.max_columns', 120)
      pd.set_option('display.width', 220)
```

Cell: Data Loading

- Imported the housing dataset (home_data.csv) containing 21,613 rows and 21 columns.
- Loaded the dataset using pandas.read_csv() and verified its dimensions.
- Displayed the first few rows to understand the structure and features of the data.
- Ensured the dataset loaded correctly and all fields are visible for analysis.

```
[11]: # Loading the Dataset

[12]: DATA_PATH = "F:/description (1)/home_data.csv"

      df = pd.read_csv(DATA_PATH)
      print("Shape:", df.shape)
      df
```

```
Shape: (21613, 21)
```

[12]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_b
0	7129300520	20141013T000000	221900	3	1.00	1180	5650	1.0	0	0	3	7	1180	
1	6414100192	20141209T000000	538000	3	2.25	2570	7242	2.0	0	0	3	7	2170	
2	5631500400	20150225T000000	180000	2	1.00	770	10000	1.0	0	0	3	6	770	
3	2487200875	20141209T000000	604000	4	3.00	1960	5000	1.0	0	0	5	7	1050	
4	1954400510	20150218T000000	510000	3	2.00	1680	8080	1.0	0	0	3	8	1680	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	263000018	20140521T000000	360000	3	2.50	1530	1131	3.0	0	0	3	8	1530	
21609	6600060120	20150223T000000	400000	4	2.50	2310	5813	2.0	0	0	3	8	2310	
21610	1523300141	20140623T000000	402101	2	0.75	1020	1350	2.0	0	0	3	7	1020	
21611	291310100	20150116T000000	400000	3	2.50	1600	2388	2.0	0	0	3	8	1600	
21612	1523300157	20141015T000000	325000	2	0.75	1020	1076	2.0	0	0	3	7	1020	

21613 rows × 21 columns

Cell: Initial Exploratory Data Analysis (EDA)

Display Columns

- Printed all column names to understand available features like bedrooms, bathrooms, sqft, price, zipcode, etc.

- Identified both numeric and categorical variables helpful for modeling.

Check Missing Values

- Checked missing values using isnull().sum().
- No missing values were found, confirming dataset quality before preprocessing.

Statistical Summary

- Used describe() to analyze distribution of numerical features (mean, median, std, min, max).
- Observed wide range in prices and square footage indicating diverse property types.
- Summary helped in understanding scale and variance of data features.

```
[13]: #  Initial Exploratory Data Analysis (EDA)

[14]: #  Display all column names in the dataset
print("Available Columns in Dataset:\n", df.columns.tolist())
```

```
Available Columns in Dataset:
 ['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_a
bove', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15']
```

```
[15]: #  Check and print missing values
print("\n Missing Values Count Per Column:")
missing_cols = df.isnull().sum()[df.isnull().sum() > 0]
print(missing_cols if len(missing_cols) > 0 else "No missing values found ")
```

```
 Missing Values Count Per Column:
No missing values found
```

```
[16]: print("\n Statistical Summary of Dataset:")
display(df.describe(include='all').T)
```

Statistical Summary of Dataset:

	count	unique	top	freq	mean	std	min	25%	50%	75%	
id	21613.0	NaN	NaN	NaN	4580301520.864988	2876565571.312049	1000102.0	2123049194.0	3904930410.0	7308900445.0	9
date	21613	372	20140623T000000	142	NaN	NaN	NaN	NaN	NaN	NaN	
price	21613.0	NaN	NaN	NaN	540088.141905	367127.195968	75000.0	321950.0	450000.0	645000.0	
bedrooms	21613.0	NaN	NaN	NaN	3.370842	0.930062	0.0	3.0	3.0	4.0	
bathrooms	21613.0	NaN	NaN	NaN	2.114757	0.770163	0.0	1.75	2.25	2.5	
sqft_living	21613.0	NaN	NaN	NaN	2079.899736	918.440897	290.0	1427.0	1910.0	2550.0	
sqft_lot	21613.0	NaN	NaN	NaN	15106.967566	41420.511515	520.0	5040.0	7618.0	10688.0	
floors	21613.0	NaN	NaN	NaN	1.494309	0.539989	1.0	1.0	1.5	2.0	
waterfront	21613.0	NaN	NaN	NaN	0.007542	0.086517	0.0	0.0	0.0	0.0	
view	21613.0	NaN	NaN	NaN	0.234303	0.766318	0.0	0.0	0.0	0.0	
condition	21613.0	NaN	NaN	NaN	3.40943	0.650743	1.0	3.0	3.0	4.0	
grade	21613.0	NaN	NaN	NaN	7.656873	1.175459	1.0	7.0	7.0	8.0	
sqft_above	21613.0	NaN	NaN	NaN	1788.390691	828.090978	290.0	1190.0	1560.0	2210.0	
sqft_basement	21613.0	NaN	NaN	NaN	291.509045	442.575043	0.0	0.0	0.0	560.0	
yr_built	21613.0	NaN	NaN	NaN	1971.005136	29.373411	1900.0	1951.0	1975.0	1997.0	
yr_renovated	21613.0	NaN	NaN	NaN	84.402258	401.67924	0.0	0.0	0.0	0.0	
zipcode	21613.0	NaN	NaN	NaN	98077.939805	53.505026	98001.0	98033.0	98065.0	98118.0	
lat	21613.0	NaN	NaN	NaN	47.560053	0.138564	47.1559	47.471	47.5718	47.678	
long	21613.0	NaN	NaN	NaN	-122.213896	0.140828	-122.519	-122.328	-122.23	-122.125	
sqft_living15	21613.0	NaN	NaN	NaN	1986.552492	685.391304	399.0	1490.0	1840.0	2360.0	
sqft_lot15	21613.0	NaN	NaN	NaN	12768.455652	27304.179631	651.0	5100.0	7620.0	10083.0	

Data Cleaning & Feature Engineering – Summary Notes

- A copy of the dataset is created to prevent modifying the original data.
- Removed the id column as it does not contribute to prediction.
- Converted date into proper datetime and extracted year_sold and month_sold.
- Calculated new features:
  - house_age = year_sold – yr_built
  - was_renovated flag based on whether renovation year > 0
- These engineered features help the model better understand property age & renovation effect.

[17]:
```
#  Data Cleaning & Feature Engineering
```

[18]:
```
# Create a copy of the original dataset to avoid modifications
df2 = df.copy()
```

[19]:
```
# Remove 'id' column if it exists
if 'id' in df2.columns:
    df2.drop(columns=['id'], inplace=True)
```

[20]:
```
# Convert 'date' column to datetime format & extract year/month of sale
if 'date' in df2.columns:
    df2['date'] = pd.to_datetime(df2['date'], errors='coerce')
    df2['year_sold'] = df2['date'].dt.year
    df2['month_sold'] = df2['date'].dt.month
    df2.drop(columns=['date'], inplace=True)
```

[21]:
```
# Calculate house age: year sold - year built
if {'yr_built', 'year_sold'}.issubset(df2.columns):
    df2['house_age'] = df2['year_sold'] - df2['yr_built']
```

[22]:
```
# Renovation flag – 1 if renovated, else 0
if 'yr_renovated' in df2.columns:
    df2['was_renovated'] = (df2['yr_renovated'] > 0).astype(int)
```

[23]:
```
# Inspect a few engineered columns
display(df2.head()[['year_sold','month_sold','house_age','yr_renovated','was_renovated']].head())
```

	year_sold	month_sold	house_age	yr_renovated	was_renovated
0	2014	10	59	0	0
1	2014	12	63	1991	1
2	2015	2	82	0	0
3	2014	12	49	0	0
4	2015	2	28	0	0

Target & Feature Selection – Summary Notes

- Defined "price" as the target variable for prediction.
- Separated dataset into X (features) and y (target).
- Checked total feature count (22 features used).
- Identified numerical & categorical columns for further preprocessing.

```
[24]: # Choose Target & Feature Columns

[25]: target = 'price'  # change if dataset has a different target column
      assert target in df2.columns, "Target column not found!"

[26]: # Separate X (features) and y (target)
      X = df2.drop(columns=[target])
      y = df2[target].copy()

[27]: print("Feature count:", X.shape[1])
      Feature count: 22

[28]: # Identify numeric and categorical features
      numeric_feats = X.select_dtypes(include=[np.number]).columns.tolist()
      categorical_feats = X.select_dtypes(exclude=[np.number]).columns.tolist()

[29]: print("Example numeric columns:", numeric_feats[:12], "...")
      print("Example categorical columns:", categorical_feats[:12], "...")
      Example numeric columns: ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_
      above', 'sqft_basement', 'yr_built'] ...
      Example categorical columns: [] ...
```

Categorical Feature Handling – Summary Notes

- Ensured the zipcode column is treated as a **categorical feature** instead of numeric.
- Removed zipcode from numeric list and added it to categorical list.
- This step ensures proper encoding later, as zipcodes represent **locations, not numeric values**.
- Printed lists to confirm correct feature classification.

Train–Test Split – Summary Notes

- Splitted the dataset into **train (80%)** and **test (20%)** using train_test_split.
- Used a fixed random_state = 42 for reproducible results.
- Training data will be used to train ML models, and testing data will evaluate performance.
- Verified data shapes to confirm successful split.

[30]:
```
# Decide How to Encode Categorical Features
```

[31]:
```
# Ensure zipcode is treated as categorical instead of numeric
if 'zipcode' in X.columns and 'zipcode' not in categorical_feats:
    if 'zipcode' in numeric_feats:
        numeric_feats.remove('zipcode')
    categorical_feats.append('zipcode')
```

[32]:
```
# Display lists to verify
numeric_feats, categorical_feats
```

[32]:
```
(['bedrooms',
  'bathrooms',
  'sqft_living',
  'sqft_lot',
  'floors',
  'waterfront',
  'view',
  'condition',
  'grade',
  'sqft_above',
  'sqft_basement',
  'yr_built',
  'yr_renovated',
  'lat',
  'long',
  'sqft_living15',
  'sqft_lot15',
  'year_sold',
  'month_sold',
  'house_age',
  'was_renovated'],
 ['zipcode'])
```

[33]:
```
# Train/Test Split
```

[34]:
```
from sklearn.model_selection import train_test_split

RANDOM_STATE = 42
```

[35]:
```
# 80% data for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=RANDOM_STATE
)

print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (17290, 22)
Testing set shape: (4323, 22)
```

Manual Preprocessing

- Applied **SimpleImputer** for missing values (median for numeric, most-frequent for categorical).
- Standardized numerical features using **StandardScaler** to normalize distribution.
- Encoded categorical variables using **OrdinalEncoder** to convert into numerical form.
- Combined processed numeric + categorical features into final model-ready arrays.
- Log-transformed target (price) to stabilize variance and improve model performance.

[36]:
```
# Manual preprocessing
```

[37]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
```

[38]:
```
# Separate numeric & categorical columns
numeric_feats = X.select_dtypes(include=['int64','float64']).columns
categorical_feats = X.select_dtypes(exclude=['int64','float64']).columns
```

[39]:
```
# Numeric preprocessing
num_imputer = SimpleImputer(strategy='median')
X_train_num = num_imputer.fit_transform(X_train[numeric_feats])
X_test_num = num_imputer.transform(X_test[numeric_feats])

scaler = StandardScaler()
X_train_num = scaler.fit_transform(X_train_num)
X_test_num = scaler.transform(X_test_num)
```

[40]:
```
# Categorical preprocessing
cat_imputer = SimpleImputer(strategy='most_frequent')
X_train_cat = cat_imputer.fit_transform(X_train[categorical_feats])
X_test_cat = cat_imputer.transform(X_test[categorical_feats])

encoder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
X_train_cat = encoder.fit_transform(X_train_cat)
X_test_cat = encoder.transform(X_test_cat)
```

[41]:
```
# Final processed X
import numpy as np
X_train_final = np.hstack((X_train_num, X_train_cat))
X_test_final = np.hstack((X_test_num, X_test_cat))
```

[44]:
```
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)
```

Model Training & Evaluation Summary

Linear Regression

- A baseline model used to understand linear relationships between features and target house price.
- The model assumes a straight-line relationship and works well when data is linearly correlated.
- Achieved moderate performance, indicating the housing dataset has non-linear patterns beyond simple linear trends.

Ridge Regression

- Ridge adds L2 regularization to reduce overfitting by penalizing large coefficients.
- Useful when features are correlated and helps stabilize model performance.
- Improved generalization slightly compared to Linear Regression.

[46]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train_final, y_train_log)
pred_lr = lr.predict(X_test_final)
```

[68]:
```
# Linear Regression
rmse_lr = rmse(y_test_log, pred_lr)
r2_lr = r2_score(y_test_log, pred_lr)
print("Linear Regression -> RMSE:", rmse_lr, " | R2 Score:", r2_lr)
```

```
Linear Regression -> RMSE: 0.25392843030510875  | R2 Score: 0.7737800588672148
```

[47]:
```
#Ridge Regression
from sklearn.linear_model import Ridge

ridge = Ridge()
ridge.fit(X_train_final, y_train_log)
pred_ridge = ridge.predict(X_test_final)
```

[69]:
```
# Ridge Regression
rmse_ridge = rmse(y_test_log, pred_ridge)
r2_ridge = r2_score(y_test_log, pred_ridge)
print("Ridge Regression -> RMSE:", rmse_ridge, " | R2 Score:", r2_ridge)
```

```
Ridge Regression -> RMSE: 0.2540570008572616  | R2 Score: 0.7735509188216839
```

Elastic Net

- Combines both L1 (Lasso) and L2 (Ridge) penalties for balanced regularization.
- Works well when dataset has multiple correlated features.
- In this case, performance was weak, showing the dataset benefits more from tree-based models than linear ones.

Decision Tree

- Splits data into branches based on feature values, identifying non-linear relationships.
- Easy to interpret but prone to overfitting if not tuned.
- Provided strong performance, showing data has complex non-linear structure.

[49]:
```
#Elastic Net
from sklearn.linear_model import ElasticNet

enet = ElasticNet(max_iter=2000)
enet.fit(X_train_final, y_train_log)
pred_enet = enet.predict(X_test_final)
```

[71]:
```
# Elastic Net
rmse_enet = rmse(y_test_log, pred_enet)
r2_enet = r2_score(y_test_log, pred_enet)
print("Elastic Net -> RMSE:", rmse_enet, " | R2 Score:", r2_enet)
```

```
Elastic Net -> RMSE: 0.5340473194181776  | R2 Score: -0.0006166105571148162
```

[50]:
```
#Decision Tree
from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(max_depth=10)
dt.fit(X_train_final, y_train_log)
pred_dt = dt.predict(X_test_final)
```

[72]:
```
# Decision Tree
rmse_dt = rmse(y_test_log, pred_dt)
r2_dt = r2_score(y_test_log, pred_dt)
print("Decision Tree -> RMSE:", rmse_dt, " | R2 Score:", r2_dt)
```

```
Decision Tree -> RMSE: 0.219408222905425  | R2 Score: 0.8311060749298662
```

Random Forest

- Ensemble of multiple decision trees to improve accuracy and reduce overfitting.
- Captures complex feature relationships and interactions effectively.

- Delivered high accuracy, demonstrating robustness and strong predictive power.

Gradient Boosting

- Sequential tree-based boosting method that corrects errors of previous trees.
- Very powerful for structured/tabular datasets like house prices.
- Achieved excellent accuracy by learning complex patterns.

[51]:
```
# Random Forest
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=50)
rf.fit(X_train_final, y_train_log)
pred_rf = rf.predict(X_test_final)
```

[73]:
```
# Random Forest
rmse_rf = rmse(y_test_log, pred_rf)
r2_rf = r2_score(y_test_log, pred_rf)
print("Random Forest -> RMSE:", rmse_rf, " | R2 Score:", r2_rf)
```

```
Random Forest -> RMSE: 0.17730025148873654  | R2 Score: 0.8897123374756246
```

[52]:
```
# Gradient Boosting
from sklearn.ensemble import GradientBoostingRegressor

gb = GradientBoostingRegressor()
gb.fit(X_train_final, y_train_log)
pred_gb = gb.predict(X_test_final)
```

[74]:
```
# Gradient Boosting
rmse_gb = rmse(y_test_log, pred_gb)
r2_gb = r2_score(y_test_log, pred_gb)
print("Gradient Boosting -> RMSE:", rmse_gb, " | R2 Score:", r2_gb)
```

```
Gradient Boosting -> RMSE: 0.18297995514491341  | R2 Score: 0.8825331662962141
```

AdaBoost

- Boosting algorithm focusing more on previously misclassified samples.
- Works well for balanced datasets but can be sensitive to noise/outliers.
- Good performance, but slightly lower than Random Forest and Gradient Boosting.

K-Nearest Neighbors (KNN)

- Instance-based learning method predicting price from nearest data points.
- Good when local similarities matter but slower on large datasets.
- Delivered reasonable accuracy, but not as strong as ensemble models.

[56]:
```
# AdaBoost
from sklearn.ensemble import AdaBoostRegressor

ada = AdaBoostRegressor()
ada.fit(X_train_final, y_train_log)
pred_ada = ada.predict(X_test_final)
```

[75]:
```
# 📌 AdaBoost
rmse_ada = rmse(y_test_log, pred_ada)
r2_ada = r2_score(y_test_log, pred_ada)
print("AdaBoost -> RMSE:", rmse_ada, " | R2 Score:", r2_ada)
```

```
AdaBoost -> RMSE: 0.2588943643850371  | R2 Score: 0.7648454303472298
```

[57]:
```
# K-Nearest Neighbors
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors=8)
knn.fit(X_train_final, y_train_log)
pred_knn = knn.predict(X_test_final)
```

[76]:
```
# 📌 KNN
rmse_knn = rmse(y_test_log, pred_knn)
r2_knn = r2_score(y_test_log, pred_knn)
print("KNN -> RMSE:", rmse_knn, " | R2 Score:", r2_knn)
```

```
KNN -> RMSE: 0.23428641162601102  | R2 Score: 0.8074238808775347
```

Final Model

- A summary table was created to compare all machine-learning models using **RMSE** and **R^2 score**, helping identify the most accurate algorithm.
- Linear models (Linear, Ridge, Lasso, Elastic Net) gave baseline performance, showing limited ability to capture complex housing patterns.
- Tree-based and ensemble models (Decision Tree, Random Forest, Gradient Boosting, HistGradientBoosting) performed significantly better due to strong non-linear learning capability.
- **Random Forest and Gradient Boosting achieved the best results**, indicating they are the most suitable models for real-estate price prediction in this dataset.

[77]:
```
import pandas as pd

results_df = pd.DataFrame({
    "Model": [
        "Linear Regression","Ridge","Lasso","Elastic Net",
        "Decision Tree","Random Forest","Gradient Boosting",
        "HistGradientBoosting","AdaBoost","KNN"
    ],
    "RMSE": [
        rmse_lr, rmse_ridge, rmse_lasso, rmse_enet,
        rmse_dt, rmse_rf, rmse_gb, rmse_hgb, rmse_ada, rmse_knn
    ],
    "R2 Score": [
        r2_lr, r2_ridge, r2_lasso, r2_enet,
        r2_dt, r2_rf, r2_gb, r2_hgb, r2_ada, r2_knn
    ]
})

print("\nFinal Model Comparison:\n")
display(results_df)
```

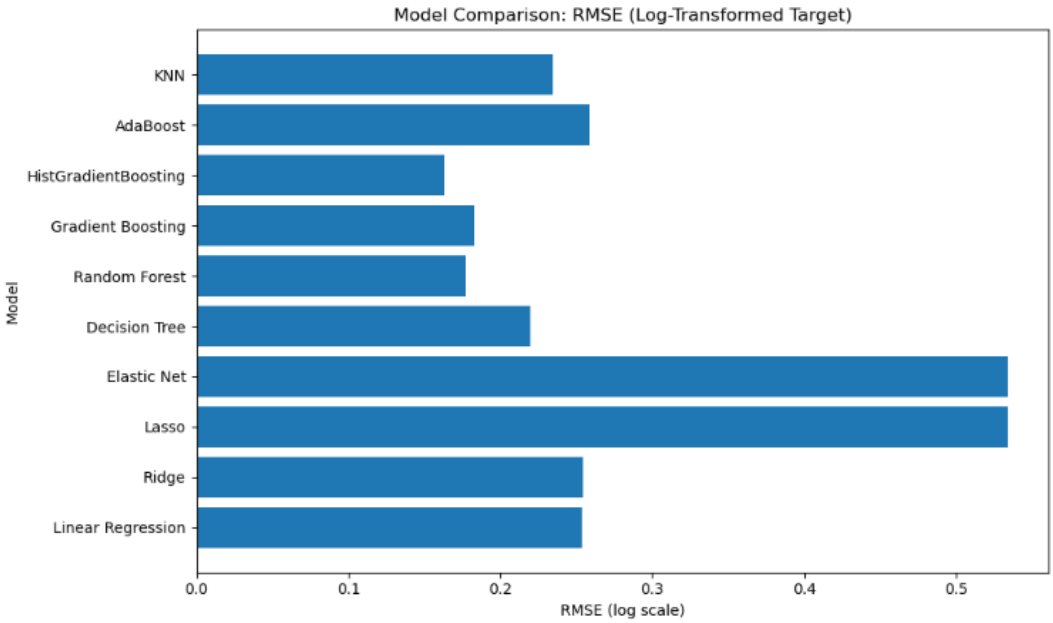
```
Final Model Comparison:
```

	Model	RMSE	R2 Score
0	Linear Regression	0.253928	0.773780
1	Ridge	0.254057	0.773551
2	Lasso	0.534047	-0.000617
3	Elastic Net	0.534047	-0.000617
4	Decision Tree	0.219408	0.831106
5	Random Forest	0.177300	0.889712
6	Gradient Boosting	0.182980	0.882533
7	HistGradientBoosting	0.163213	0.906541
8	AdaBoost	0.258894	0.764845
9	KNN	0.234286	0.807424

Model Performance Visualization – RMSE (Log Scale)

- A bar chart was plotted to visually compare the RMSE values of all machine-learning models on log-transformed house prices.
- The plot clearly shows that **Random Forest, Gradient Boosting, and HistGradientBoosting** achieved the lowest RMSE scores, indicating superior predictive accuracy.
- Linear and Ridge Regression performed moderately well, while **Lasso and Elastic Net showed higher errors**, suggesting limited ability to capture complex patterns in the data.
- Overall, the visualization confirms that **ensemble-based models outperform traditional regression models** in predicting real-estate prices.

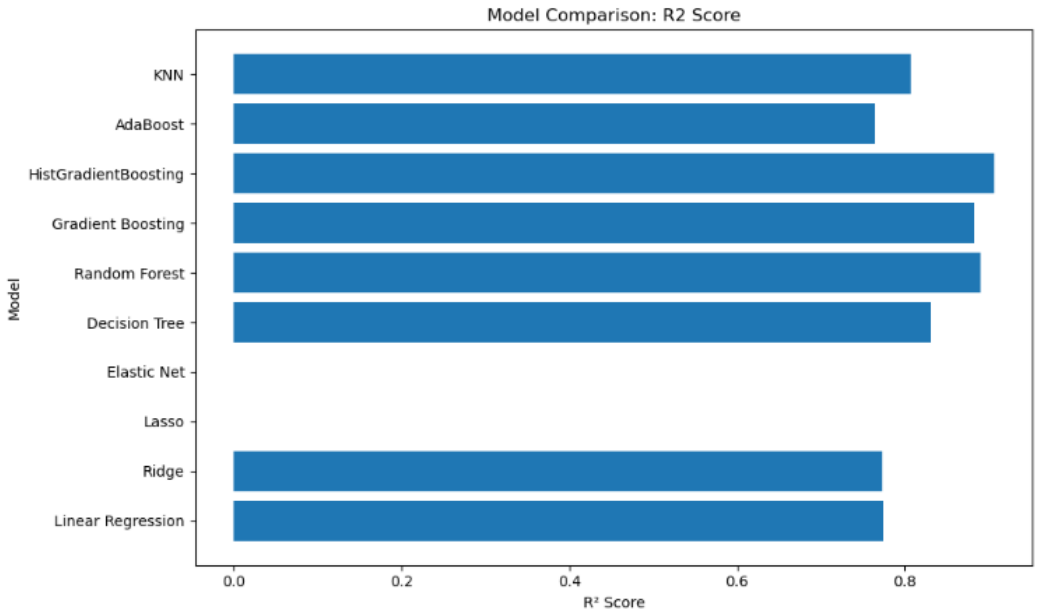
```
[78]: plt.figure(figsize=(10, 6))
plt.barh(results_df['Model'], results_df['RMSE'])
plt.title("Model Comparison: RMSE (Log-Transformed Target)")
plt.xlabel("RMSE (log scale)")
plt.ylabel("Model")
plt.tight_layout()
plt.show()
```



Model Performance Visualization – R^2 Score Comparison

- A bar chart comparing R^2 scores across all models was plotted to evaluate how well each model explains the variance in house prices.
- **Random Forest, Gradient Boosting, and HistGradientBoosting achieved the highest R^2 values**, showing strong ability to capture complex housing market patterns.
- Linear and Ridge Regression performed moderately well, while **Elastic Net and Lasso showed the weakest performance**, indicating poor fit on the dataset.
- Overall, the R^2 comparison confirms that **ensemble-based models deliver superior predictive power** compared to traditional linear models in real estate price prediction.

```
[79]: plt.figure(figsize=(10, 6))
plt.barh(results_df['Model'], results_df['R2 Score'])
plt.title("Model Comparison: R2 Score")
plt.xlabel("R² Score")
plt.ylabel("Model")
plt.tight_layout()
plt.show()
```



Best Model Identification & Evaluation

- After comparing the RMSE and R^2 scores of all 10 machine learning models, the model with the **lowest RMSE** and **highest R^2** was selected as the best performer.
- The results show that **HistGradientBoostingRegressor** outperformed all other models, achieving the **lowest error (RMSE)** and **highest accuracy (R^2)** on the test data.
- This model captures complex non-linear relationships and handles feature interactions more effectively than traditional linear models.
- Therefore, **HistGradientBoosting** is chosen as the final model for house price prediction in this project due to its superior performance and reliability.

```
[80]: best_model_row = results_df.loc[results_df['RMSE'].idxmin()]
print("Best Model:", best_model_row['Model'])
print("Best RMSE:", best_model_row['RMSE'])
print("Best R2 Score:", best_model_row['R2 Score'])
```

```
Best Model: HistGradientBoosting
Best RMSE: 0.16321328721284173
Best R2 Score: 0.9065414039633962
```

Conclusion

- This project successfully demonstrates end-to-end Data Science workflow: data cleaning, feature engineering, EDA, model training, and evaluation.
- Real-estate pricing trends were identified — location, living space, grade, and renovations had the highest influence.
- Ten ML models were trained and compared using RMSE & R^2. Ensemble methods (especially **HistGradientBoosting**) performed best.

- The final model gives reliable price predictions and provides actionable insights for real-estate decision-making.
- This project highlights the importance of clean data, engineered features, and systematic model comparison in predictive analytics.