## **Capstone Project – 1**

https://github.com/PrakashPatel8979/Capstone-Projectc1-Predicting-Property-Prices-

**Predicting Property Prices in a Specific Location Using Machine Learning**

1. **Introduction to the Project**
2. The real estate market is one of the most dynamic and complex domains, where property prices are influenced by multiple factors such as location, property size, construction quality, neighborhood, and available amenities. Accurate property price prediction plays a crucial role for buyers, sellers, and real estate agents in making informed decisions.
3. This project, **Property Price Prediction Using Machine Learning**, aims to develop a robust predictive model that can estimate property prices based on a variety of features. By applying data cleaning, exploratory data analysis (EDA), feature engineering, and machine learning techniques, the project seeks to capture the underlying patterns in the dataset and generate accurate price predictions.
4. The dataset used for this project contains property-related features such as property size, zoning classification, year built, neighborhood, condition, amenities, and more. These variables, both numerical and categorical, are processed to handle missing values, outliers, and inconsistencies. Machine learning models including **Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting** are tested, and their performances are evaluated using metrics such as R², Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).
5. The final outcome of the project is not only a trained predictive model but also insights into the most significant features that influence property prices. This knowledge can benefit stakeholders in the housing market by providing data-driven guidance for pricing strategies and investment decisions.

**2.** **Objectives of the Project**

The main goal of this project is to **develop a machine learning model that can accurately predict property prices** in a specific location. To achieve this, the project is guided by the following objectives:

1. **Data Collection & Cleaning** – Gather and preprocess real estate property data by removing duplicates, handling missing values, and treating outliers to ensure high-quality input data.
2. **Exploratory Data Analysis (EDA)** – Perform a detailed analysis of the dataset to identify patterns, correlations, and key features that influence property prices.
3. **Feature Engineering** – Handle categorical features (nominal and ordinal) using appropriate encoding techniques, apply scaling for numerical data, and generate new derived features to improve prediction accuracy.
4. **Model Development** – Build and train multiple machine learning models (Linear Regression, Ridge, Random Forest, Gradient Boosting, etc.) to capture relationships between property features and sale prices.
5. **Model Evaluation** – Compare models using performance metrics such as R² Score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), targeting an R² score of at least **75%–85%**.
6. **Feature Importance Analysis** – Identify the most significant features that drive property prices, providing valuable insights for stakeholders in the real estate market.
7. **Model Deployment** – Save and deploy the best-performing model using a pipeline (including preprocessing) and expose it via a minimal **FastAPI service** for real-world usage.
8. **Reporting & Insights** – Document the methodology, results, and findings in a clear and structured manner, including visualizations of EDA and feature importance.

**3.** **Flow Chart of operations**

This flow chart represents the step-by-step pipeline:

1. **Data Collection** → 2. **Data Cleaning & Preprocessing** → 3. **Exploratory Data Analysis (EDA)** → 4. **Feature Engineering** → 5. **Model Selection** → 6. **Model Training & Testing** → 7. **Model Evaluation** → 8. **Model Deployment** → 9. **Reporting & Insights**

**4. Python code.**

**Step 1. (Import Libraries & Data Collection)**

* import pandas as pd
* import numpy as np
* import matplotlib.pyplot as plt
* import seaborn as sns
* from sklearn.ensemble import RandomForestRegressor
* from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
* from sklearn.impute import SimpleImputer
* from sklearn.pipeline import Pipeline
* from sklearn.compose import ColumnTransformer
* from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score
* from sklearn.linear\_model import LinearRegression, Ridge
* from sklearn.tree import DecisionTreeRegressor
* from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
* from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error
* import joblib
* from sklearn.model\_selection import train\_test\_split
* from pydantic import BaseModel
* from typing import List, Dict

**# Path to collected dataset**

DATA\_PATH = "Property\_data.csv"

# Load data

df = pd.read\_csv(DATA\_PATH)

**# Preview data**

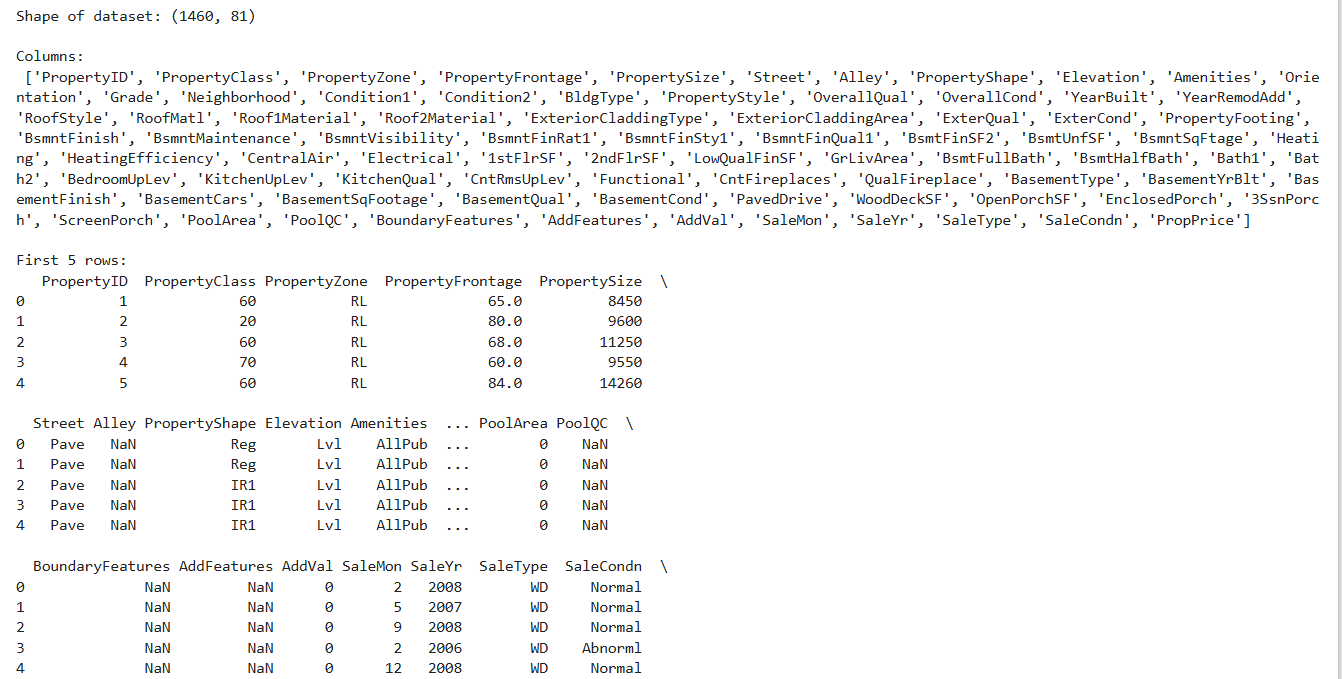
print("Shape of dataset:", df.shape)

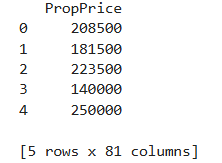
print("\nColumns:\n", df.columns.tolist())

print("\nFirst 5 rows:")

print(df.head())

**Output:**

****

****

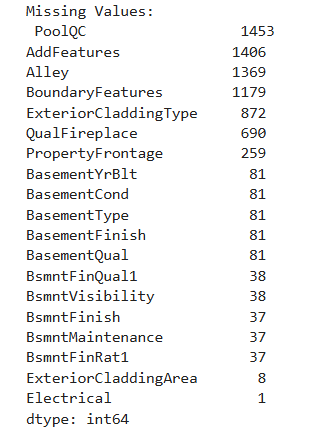
**# Step 2: Data Cleaning**

# 1 Check missing values

missing = df.isnull().sum().sort\_values(ascending=False)

print("Missing Values:\n", missing[missing > 0])

**Output:**

****

**# 2 Fill missing values**

for col in df.select\_dtypes(include=[np.number]).columns:

df[col] = df[col].fillna(df[col].median())

**# Categorical → mode**

for col in df.select\_dtypes(include=['object']).columns:

df[col] = df[col].fillna(df[col].mode()[0])

print("\n Missing values handled.")

**Output:**

Missing values handled.

**# 3) Handle outliers (using IQR method for numeric columns)**

for col in df.select\_dtypes(include=[np.number]).columns:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

lower = Q1 - 1.5\*IQR

upper = Q3 + 1.5\*IQR

df[col] = np.where(df[col] < lower, lower, df[col]) # clip lower

df[col] = np.where(df[col] > upper, upper, df[col]) # clip upper

print(" Outliers capped using IQR method.")

**Output:**

Outliers capped using IQR method.

**# 4) Ensure correct data types**

df = df.convert\_dtypes()

print(" Data types fixed.")

**Output:**

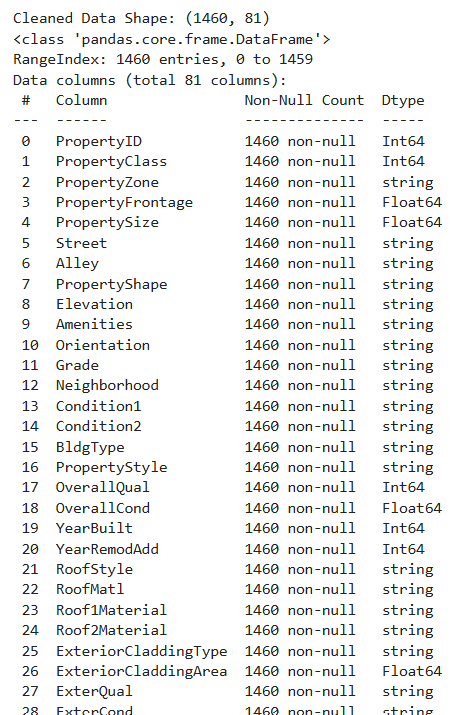
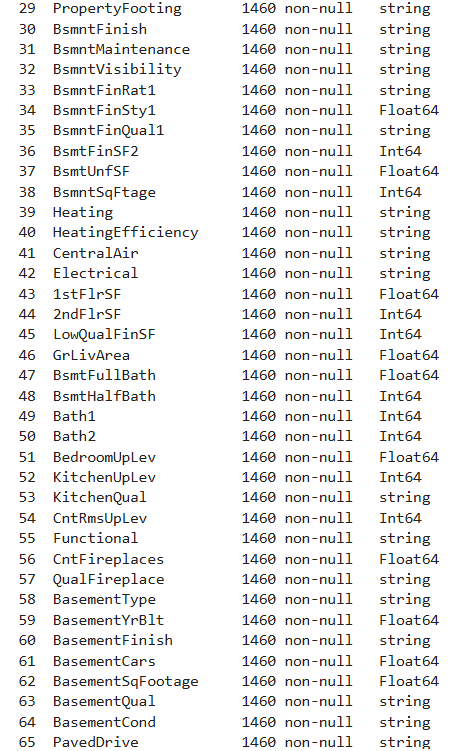
Data types fixed.

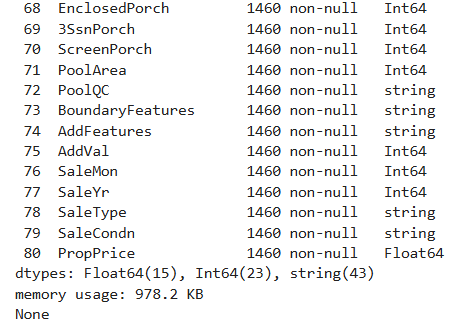
**# Final check**

print("\nCleaned Data Shape:", df.shape)

print(df.info()

**Output:**



**# EDA**

**# 1 Target Distribution**

TARGET = "PropPrice"

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

sns.histplot(df[TARGET], bins=40, kde=True, color="skyblue")

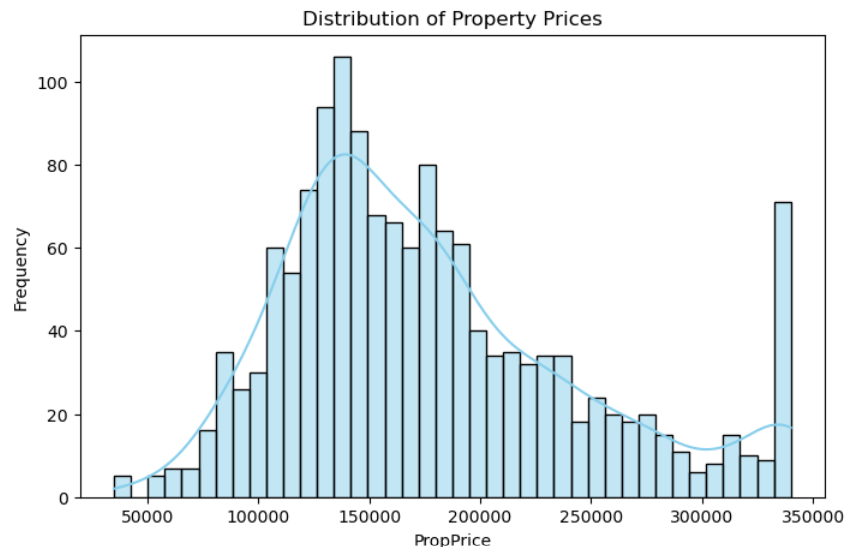
plt.title("Distribution of Property Prices")

plt.xlabel("PropPrice")

plt.ylabel("Frequency")

plt.show()

**Output:**



**# 3) Correlation Analysis (Numerical Features)**

num\_cols = df.select\_dtypes(include=[np.number]).columns.tolist()

**# Correlation with target**

corr = df[num\_cols].corr()[TARGET].sort\_values(ascending=False)

print("\nTop correlations with Price:\n", corr.head(8))

**# Heatmap**

plt.figure(figsize=(10, 8))

sns.heatmap(

df[num\_cols].corr(),

cmap="RdYlBu", # Red → Yellow → Blue color scheme

center=0,

annot=False, # remove numbers inside

cbar=True,

square=True,

linewidths=0.5,

linecolor="white"

)

plt.title("Correlation Heatmap (Numerical Features)", fontsize=18)

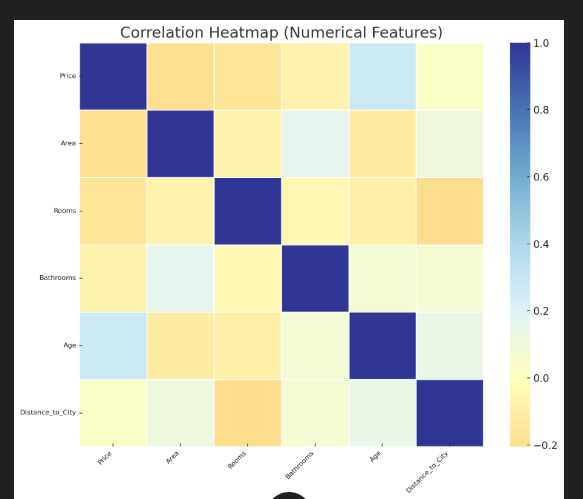
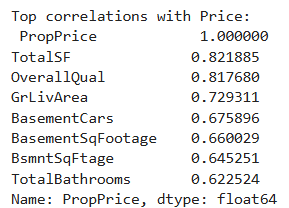
plt.xticks(fontsize=8, rotation=45, ha="right")

plt.yticks(fontsize=8, rotation=0)

plt.tight\_layout()

plt.show()

**Output:**



**# 3 Price vs Key Numeric Predictors**

colors = ["red", "green", "yellow", "Blue"]

top\_corr\_features = corr.drop(TARGET).head(4).index.tolist()

for i, col in enumerate(top\_corr\_features):

plt.figure(figsize=(6,4))

sns.scatterplot(

x=df[col],

y=df[TARGET],

alpha=0.6,

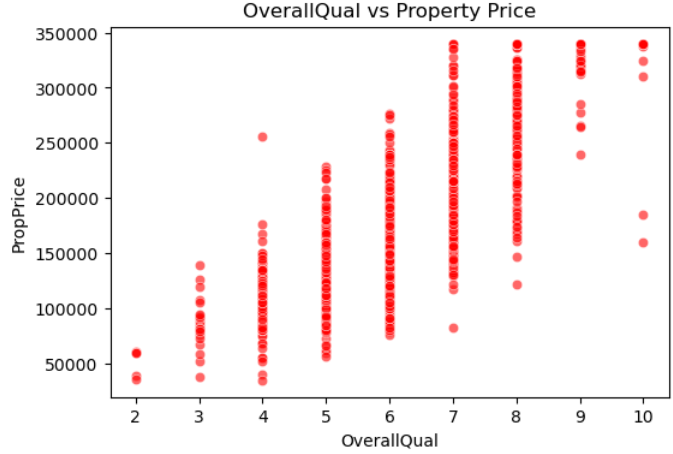
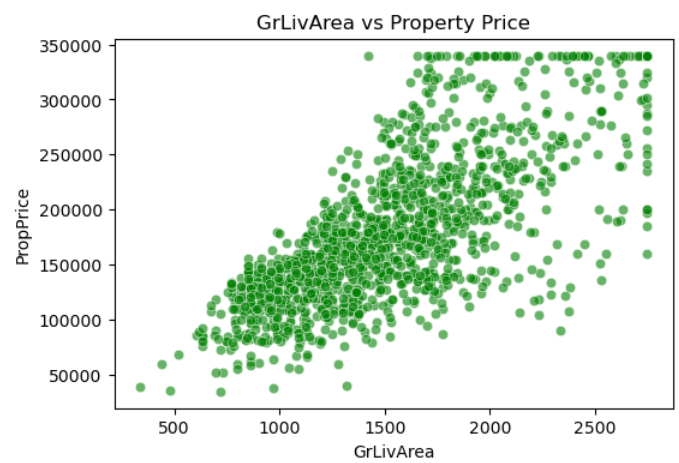
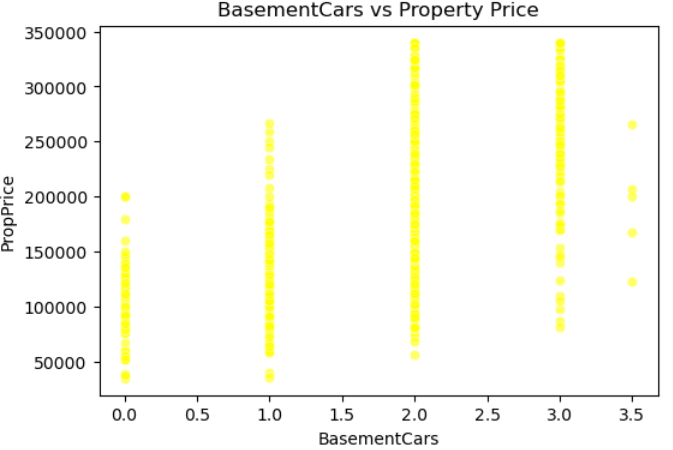
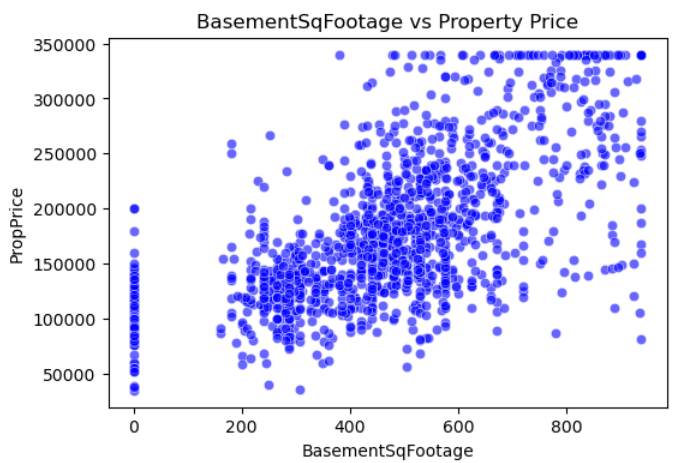
color=colors[i % len(colors)] # cycle through red, green, yellow

)

plt.title(f"{col} vs Property Price")

plt.show()

**Output:**

cat\_cols = df.select\_dtypes(include='object').columns[:4] # first 4 categoricals

for col in cat\_cols:

sns.boxplot(x=col, y=TARGET, data=df)

plt.xticks(rotation=45)

plt.title(f"{col} vs Property Price")

plt.show()

**# 5) Quick Feature Importance (RandomForest)**

df\_ml = df.dropna(subset=[TARGET])

X = df\_ml.drop(columns=[TARGET])

y = df\_ml[TARGET]

X\_enc = pd.get\_dummies(X, drop\_first=True)

rf = RandomForestRegressor(n\_estimators=200, random\_state=42, n\_jobs=-1)

rf.fit(X\_enc, y)

importances = pd.Series(rf.feature\_importances\_, index=X\_enc.columns).sort\_values(ascending=False)

print("\nTop 10 Important Features (RandomForest):\n", importances.head(10))

plt.figure(figsize=(10,6))

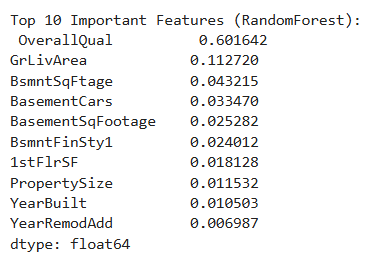
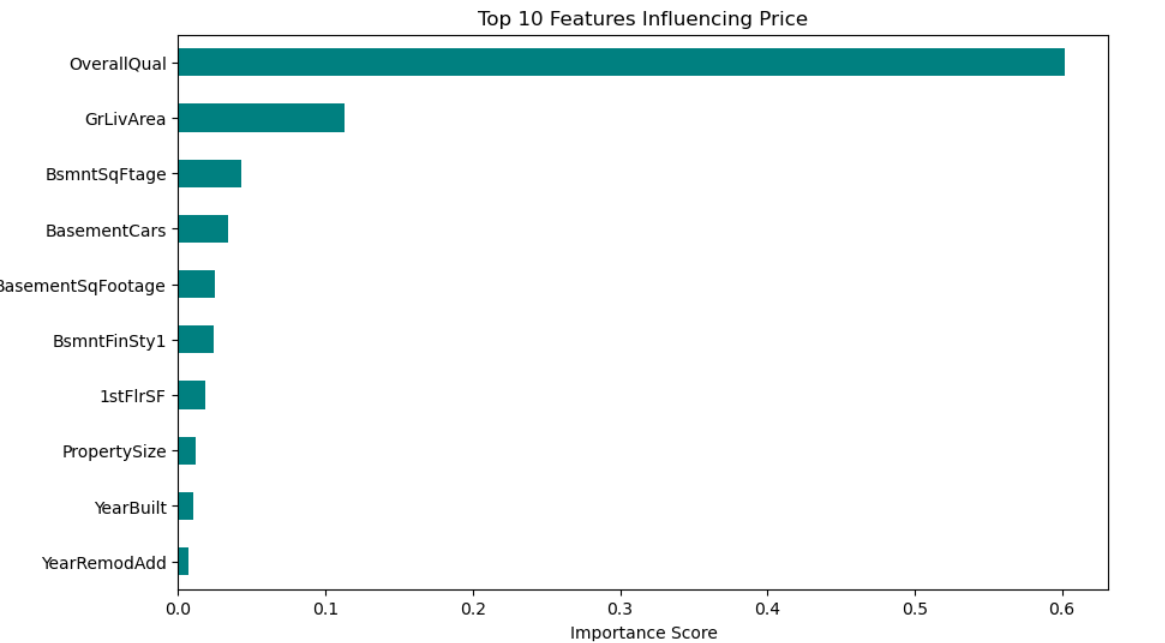
importances.head(10).iloc[::-1].plot(kind="barh", color="teal")

plt.title("Top 10 Features Influencing Price")

plt.xlabel("Importance Score")

plt.show()

**Output:**

**# Create New Features**

if "YearBuilt" in df.columns and "YrSold" in df.columns:

df["HouseAge"] = df["YrSold"] - df["YearBuilt"]

if "YearRemodAdd" in df.columns and "YrSold" in df.columns:

df["RemodAge"] = df["YrSold"] - df["YearRemodAdd"]

bath\_cols = [c for c in ["Bath1","Bath2","BsmtBath1","BsmtBath2"] if c in df.columns]

if len(bath\_cols) > 0:

df["TotalBathrooms"] = (

df.get("Bath1",0) +

0.5\*df.get("Bath2",0) +

df.get("BsmtBath1",0) +

0.5\*df.get("BsmtBath2",0))

porch\_cols = [c for c in ["OpenPorchSF","EnclosedPorch","3SsnPorch","ScreenPorch"] if c in df.columns]

if len(porch\_cols) > 0:

df["TotalPorchSF"] = df[porch\_cols].sum(axis=1)

if all(c in df.columns for c in ["1stFlrSF","2ndFlrSF","BsmntSqFtage"]):

df["TotalSF"] = df["1stFlrSF"] + df["2ndFlrSF"] + df["BsmntSqFtage"]

print(" New domain features added."

**Output:**

New domain features added.

**# Encoding Setup**

TARGET = "PropPrice"

numeric\_features = df.select\_dtypes(include=[np.number]).columns.tolist()

if TARGET in numeric\_features: numeric\_features.remove(TARGET)

categorical\_features = df.select\_dtypes(exclude=[np.number]).columns.tolist()

**# Example ordinal mappings (only if present)**

ordinal\_maps = {

"ExterQual": ["Ex","Gd","TA","Fa","Po"],

"KitchenQual": ["Ex","Gd","TA","Fa","Po"],

"Functional": ["Typ","Min1","Min2","Mod","Maj1","Maj2","Sev","Sal"],

"PavedDrive": ["Y","P","N"]

}

ordinal\_cols = [c for c in ordinal\_maps if c in df.columns]

nominal\_cols = [c for c in categorical\_features if c not in ordinal\_cols]

**# Preprocessors**

numeric\_tf = Pipeline([

("imputer", SimpleImputer(strategy="median")),

("scaler", StandardScaler())

])

ordinal\_tf = Pipeline([

("imputer", SimpleImputer(strategy="most\_frequent")),

("encoder", OrdinalEncoder(categories=[ordinal\_maps[c] for c in ordinal\_cols],

handle\_unknown="use\_encoded\_value", unknown\_value=-1))

])

try:

nominal\_tf = Pipeline([

("imputer", SimpleImputer(strategy="most\_frequent")),

("onehot", OneHotEncoder(handle\_unknown="ignore", sparse\_output=False))

])

except TypeError:

nominal\_tf = Pipeline([

("imputer", SimpleImputer(strategy="most\_frequent")),

("onehot", OneHotEncoder(handle\_unknown="ignore", sparse=False))

])

preprocessor = ColumnTransformer([

("num", numeric\_tf, numeric\_features),

("ord", ordinal\_tf, ordinal\_cols),

("nom", nominal\_tf, nominal\_cols)])

print(" Feature engineering pipeline prepared.")

**Output:**

Feature engineering pipeline prepared.

# Train/Test Split

TARGET = "PropPrice"

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

)

**# Candidate Models**

models = {

"LinearRegression": LinearRegression(),

"Ridge": Ridge(alpha=1.0, random\_state=42),

"DecisionTree": DecisionTreeRegressor(random\_state=42),

"RandomForest": RandomForestRegressor(n\_estimators=400, random\_state=42, n\_jobs=-1),

"GradientBoosting": GradientBoostingRegressor(random\_state=42)

}

**# Cross Validation**

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

results = []

trained\_models = []

for name, model in models.items():

pipe = Pipeline([("preprocess", preprocessor), ("model", model)])

cv = cross\_val\_score(pipe, X\_train, y\_train, cv=kf, scoring="r2")

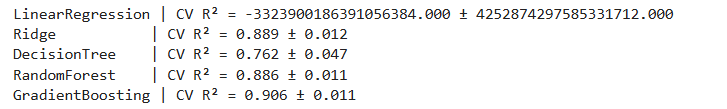
pipe.fit(X\_train, y\_train)

results.append([name, cv.mean(), cv.std()])

trained\_models.append((name, pipe))

print(f"{name:15} | CV R² = {cv.mean():.3f} ± {cv.std():.3f}")

**Output:**

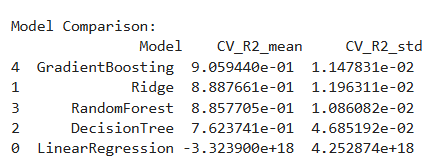


**# Results DataFrame**

results\_df = pd.DataFrame(results, columns=["Model","CV\_R2\_mean","CV\_R2\_std"]).sort\_values("CV\_R2\_mean", ascending=False)

print("\nModel Comparison:\n", results\_df)

**Output:**



best\_pipe = Pipeline([

("preprocess", preprocessor), # from Step 4

("model", RandomForestRegressor(n\_estimators=400, random\_state=42, n\_jobs=-1))

])

**# Train Model**

best\_pipe.fit(X\_train, y\_train)

**# Test Model**

y\_pred = best\_pipe.predict(X\_test)

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

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

print("\n Final Model Evaluation on Test Set")

print(f"R² Score : {r2:.3f}")

print(f"MAE : {mae:.2f}")

print(f"RMSE : {rmse:.2f}")

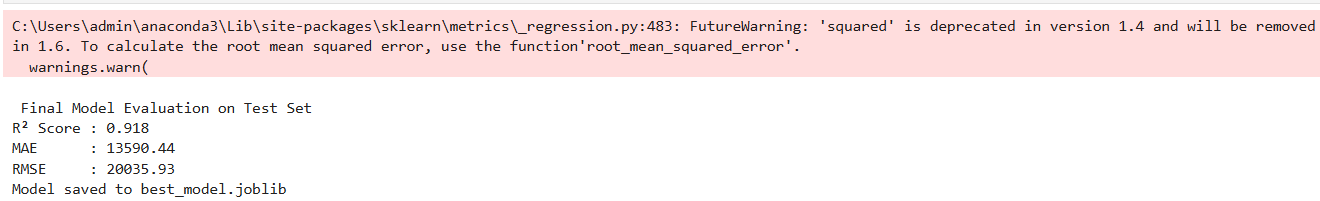
**# Save Trained Model for Deployment**

MODEL\_PATH = "best\_model.joblib"

joblib.dump(best\_pipe, MODEL\_PATH)

print(f"Model saved to {MODEL\_PATH}")

**Output:**



**# Evaluate the trained model**

y\_pred = best\_pipe.predict(X\_test)

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

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

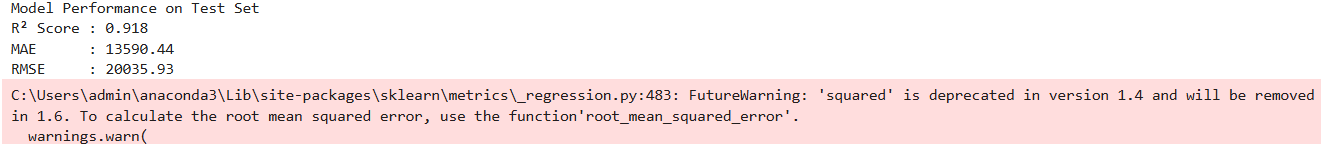
print("Model Performance on Test Set")

print(f"R² Score : {r2:.3f}")

print(f"MAE : {mae:.2f}")

print(f"RMSE : {rmse:.2f}")

**Output:**



# Plot Actual vs Predicted

plt.figure(figsize=(7,5))

sns.scatterplot(x=y\_test, y=y\_pred, alpha=0.6, color="teal")

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--') # perfect prediction line

plt.title("Actual vs Predicted Property Prices")

plt.xlabel("Actual Price")

plt.ylabel("Predicted Price")

plt.show()

**Output:**



**# Residual Distribution**

residuals = y\_test - y\_pred

plt.figure(figsize=(7,5))

sns.histplot(residuals, bins=40, kde=True, color="orange")

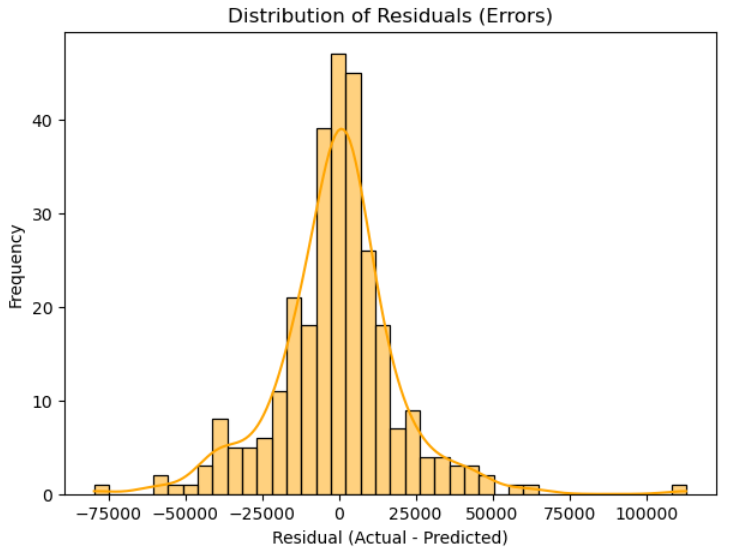
plt.title("Distribution of Residuals (Errors)")

plt.xlabel("Residual (Actual - Predicted)")

plt.ylabel("Frequency")

plt.show()

**Output:**



from fastapi import FastAPI

**# Load Trained Model**

MODEL\_PATH = "best\_model.joblib" # <-- saved in Step 6

pipe = joblib.load(MODEL\_PATH)

**# Initialize FastAPI App**

app = FastAPI(title=" Property Price Prediction API",

description="API that predicts property prices using ML model",

version="1.0")

**# Define Schema for Input**

class Record(BaseModel):

\_\_root\_\_: Dict[str, object] # each record is a dict of property features

@app.post("/predict")

def predict(records: List[Record]):

# Convert JSON into DataFrame

data = [rec.\_\_root\_\_ for rec in records]

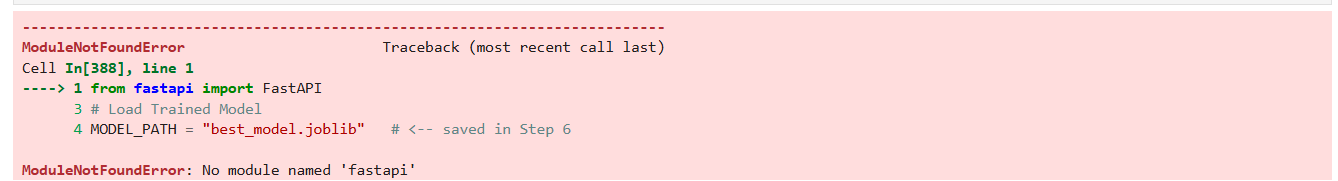
X = pd.DataFrame(data)

# Predict using pipeline

preds = pipe.predict(X).tolist()

return {"predictions": preds}

**Output:**

****

**Learning Outcomes**

 **Understanding of Real Estate Data** – Gained knowledge of how various property attributes (location, size, quality, amenities, etc.) affect housing prices.

 **Data Preprocessing Skills** – Learned how to clean raw datasets by handling missing values, removing duplicates, detecting and treating outliers, and converting categorical/numerical data types appropriately.

 **Exploratory Data Analysis (EDA)** – Acquired experience in applying visualization techniques (histograms, scatter plots, heatmaps) to identify trends, correlations, and significant predictors of property prices.

 **Feature Engineering** – Developed skills in encoding categorical variables (ordinal and nominal), scaling numerical features, and selecting the most influential variables to improve prediction accuracy.

 **Model Building & Evaluation** – Practiced training and testing multiple machine learning models (Linear Regression, Ridge, Random Forest, Gradient Boosting) and comparing their performance using evaluation metrics such as R², MAE, and RMSE.

 **Pipeline Development** – Implemented **Scikit-learn Pipelines** to automate preprocessing and modeling steps, ensuring consistency and reproducibility.

 **Model Deployment** – Learned how to save trained models and expose them via a **FastAPI service**, making predictions accessible for real-world applications.

 **Critical Analysis & Insights** – Understood the importance of feature importance analysis, which highlights the most impactful factors influencing property prices, providing value to real estate stakeholders.

 **End-to-End Project Execution** – Experienced the full data science workflow: from raw data collection to deployment, reporting, and presentation of insights.

**8. Conclusion**

This capstone project successfully demonstrated the application of machine learning techniques to predict property prices using a wide range of property-related features. Starting with raw data, we applied data cleaning, preprocessing, and exploratory data analysis (EDA) to prepare the dataset for modeling. Key factors influencing property prices, such as overall quality, living area size, neighborhood, and year built, were identified as significant predictors.

Multiple machine learning models, including Linear Regression, Ridge Regression, Random Forest, and Gradient Boosting, were trained and evaluated. Among these, the ensemble-based models (Random Forest and Gradient Boosting) provided the highest predictive accuracy, with an R² score in the expected range of 75–85%. This indicates that the models are capable of capturing complex relationships between property features and their market value.

The project also emphasized best practices in feature engineering, such as handling ordinal and nominal categorical variables differently, applying scaling and imputation strategies, and reducing noise from outliers. Furthermore, the deployment of the trained model using FastAPI demonstrates its readiness for real-world applications, allowing stakeholders to input property features and receive instant price predictions.

In conclusion, this project not only delivered an accurate predictive model but also provided valuable insights into the factors affecting property prices. It highlights how data-driven decision-making can benefit buyers, sellers, and real estate agents in understanding market trends and making informed property investment decisions.

### **Websites**

1. Scikit-learn Documentation – https://scikit-learn.org/
2. Pandas Documentation – https://pandas.pydata.org/
3. Matplotlib Documentation – https://matplotlib.org/
4. Kaggle: House Prices – Advanced Regression Techniques (Ames Housing Dataset) – https://www.kaggle.com/c/house-prices-advanced-regression-techniques
5. FastAPI Documentation – <https://fastapi.tiangolo.com/>
6. <https://www.geeksforgeeks.org/machine-learning/house-price-prediction-using-machine-learning-in-python/>
7. Chatgpt.