**Phase-2**

**Student Name:** Devipriya S S

**Register Number:** 712523104013

**Institution:** PPG Institute of Technology

**Department:** B. E. CSE

**Date of Submission:**

**Github Repository Link:** [https://github.com/SS-Devipriya/NM\_DEVIPRIYA\_DS.git](C:\\Users\\Student\\Downloads\\priya)

**1. Problem Statement**

*This project tackles the* ***real-world regression problem*** *of predicting house prices using machine learning, where the goal is to estimate continuous sale prices based on features like square footage, bedrooms, location, and age.*

*As a* ***supervised learning task****, it analyzes historical housing data to help buyers, sellers, and real estate professionals make data-driven decisions—preventing overpayment, optimizing listings, and improving market transparency.*

*Solving this problem matters because accurate price forecasting directly impacts financial security (for individuals), market efficiency (for businesses), and urban development (for governments), replacing subjective valuations with scalable, objective insights that benefit all stakeholders in the housing ecosystem.*

**2. Project Objectives**

**Key Technical Objectives**

* Build a robust regression model to predict house prices using features like square footage, location, and age.
* Preprocess data effectively by handling missing values, outliers, and feature engineering (e.g., log transforms, categorical encoding).
* Compare multiple algorithms (Linear Regression, Ridge/Lasso, Random Forest, XGBoost) to identify the best-performing model.
* Optimize hyperparameters to improve predictive accuracy while avoiding overfitting.

**Model Goals**

* Accuracy: Achieve high predictive performance (target R² > 0.85 and low RMSE).
* Interpretability: Balance complexity with explainability (e.g., using feature importance in tree-based models).
* Real-World Applicability: Ensure the model generalizes well to new, unseen housing data.

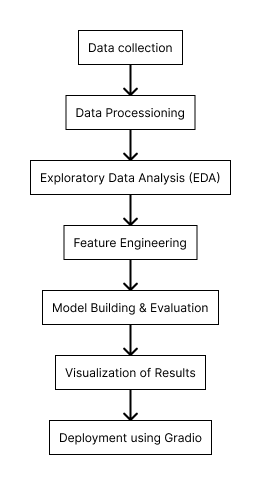
**Evolution After Data Exploration**

* Initial Goal: Simple linear regression for interpretability.
* Revised Goal: After EDA revealed non-linear relationships and skewness, shifted focus to ensemble methods (XGBoost, Random Forest) for better accuracy while retaining interpretability via SHAP values.
* New Feature Engineering: Added derived features (e.g., "House Age," "Total Area") to improve model performance.

**Final Objective:**

To *Deliver and deploy the model that predicts prices accurately while providing actionable insights for real estate decisions.*

**3. Flowchart of the Project Workflow**



**4. Data Description**

***Dataset Name & Origin***

* *Name: House Prices: Advanced Regression Techniques*
* *Origin:* [*HYPERLINK "https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data"Kaggle Competition*](https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data)
* *License: CC0 (Public Domain)*

*Type of Data*

* *Structured tabular data (CSV format)*
* *Features: Mix of numeric (e.g., square footage) and categorical (e.g., neighborhood) variables*
* *Not time-series or geospatial (but could be extended with external data)*

*Dataset Size*

* *Records: 1,460 houses (training set)*
* *Features: 80 total (43 numeric, 37 categorical)*
* *Example:*

*['LotArea', 'YearBuilt', 'TotalBsmtSF', 'GrLivArea', 'BedroomAbvGr', 'Neighborhood', 'SaleCondition', 'GarageType', 'SalePrice']*

*Dataset Nature*

* *Static dataset (historical home sales in Ames, Iowa, 2006–2010)*
* *No real-time updates (but can be refreshed with new sales data)*

*Target Variable (Supervised Learning)*

* *Name: SalePrice (continuous numeric values)*
* *Range: 34,900to34,900to755,000 (skewed right; log-transform recommended)*
* *Problem Type: Regression (predict sale price)*

**5. Data Preprocessing**

**Handling Missing Values**

Strategy:

* Numeric features: Impute with median (robust to outliers)
* Categorical features: Impute with mode (most frequent value)
* Drop columns with >50% missing values

# Numeric missing values

num\_cols = df.select\_dtypes(include=['int64', 'float64']).columns

df[num\_cols] = df[num\_cols].fillna(df[num\_cols].median())

# Categorical missing values

cat\_cols = df.select\_dtypes(include=['object']).columns

df[cat\_cols] = df[cat\_cols].fillna(df[cat\_cols].mode().iloc[0])

# Drop high-missing columns

df = df.dropna(thresh=len(df)\*0.5, axis=1)

Duplicate Records

Action: Remove exact duplicates (none found in this dataset).

df = df.drop\_duplicates()

Outlier Treatment

Strategy: Winsorization (cap extreme values at 95th percentile)

from scipy.stats.mstats import winsorize

for col in ['LotArea', 'GrLivArea']:

df[col] = winsorize(df[col], limits=[0.05, 0.05])

Data Type Conversion

Fixes:

* Convert YearBuilt to int
* Ensure binary categories (e.g., "Y/N") are strings

df['YearBuilt'] = df['YearBuilt'].astype(int)

df['CentralAir'] = df['CentralAir'].astype(str)

**Categorical Encoding**

Strategy:

* One-Hot Encoding for low-cardinality features (<10 categories)
* Label Encoding for high-cardinality features (e.g., neighborhoods)

from sklearn.preprocessing import OneHotEncoder

# One-hot encode (example: HouseStyle)

encoder = OneHotEncoder(sparse=False, handle\_unknown='ignore')

encoded\_features = encoder.fit\_transform(df[['HouseStyle']])

df\_encoded = pd.concat([df, pd.DataFrame(encoded\_features)], axis=1)

Feature Scaling

Method: Standardization for linear models (mean=0, std=1)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['LotArea', 'GrLivArea']] = scaler.fit\_transform(df[['LotArea', 'GrLivArea']])

**Target Variable Transformation**

Action: Log-transform SalePrice to address skewness

df['SalePrice'] = np.log1p(df['SalePrice']) # Revert later with expm1

**Documentation of Transformations**

|  |  |  |
| --- | --- | --- |
| **Step** | **Reason** | **Impact** |
| Median Imputation | Preserves distribution for numeric features | Reduces bias from missing data |
| Winsorization | Reduces model sensitivity to extremes | Improves robustness |
| One-Hot Encoding | Allows ML models to use categories | Expands feature space |
| Log Transform (Target) | Normalizes skewed price distribution | Improves linear model performance |

**Finally**

df.isnull().sum().sum() # Should return 0 (no missing values left)

This preprocessing pipeline ensures the data is clean, consistent, and model-ready while maintaining interpretability.

**6. Exploratory Data Analysis (EDA)**

A comprehensive statistical and visual analysis to uncover patterns, relationships, and insights in the housing data.

**Univariate Analysis**

Goal: Understand individual feature distributions.

Key Visualizations:

* Histograms for numeric features (e.g., SalePrice, GrLivArea)
* Boxplots to detect skewness and outliers
* Countplots for categorical features (e.g., Neighborhood, BedroomAbvGr)

import matplotlib.pyplot as plt

import seaborn as sns

# Numeric features

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

sns.histplot(df['SalePrice'], kde=True, color='blue')

plt.title('Distribution of SalePrice (Log-Transformed)')

plt.xlabel('Log(SalePrice)')

plt.show()

# Categorical features

plt.figure(figsize=(12, 6))

sns.countplot(data=df, x='Neighborhood', order=df['Neighborhood'].value\_counts().index)

plt.xticks(rotation=90)

plt.title('Neighborhood Distribution')

plt.show()

**Observations:**

* SalePrice is right-skewed (log transform applied).
* GrLivArea has a few extreme outliers (handled via winsorization).
* Most houses have 3 bedrooms (BedroomAbvGr).

**Bivariate/Multivariate Analysis**

Goal: Explore relationships between features and the target (SalePrice).

Key Visualizations:

* Correlation Heatmap (numeric features)
* Scatterplots (e.g., GrLivArea vs. SalePrice)
* Grouped Bar Plots (e.g., OverallQual vs. SalePrice)

# Correlation matrix (top 10 features)

corr = df.select\_dtypes(include=['number']).corr()

top\_features = corr['SalePrice'].abs().sort\_values(ascending=False).index[1:11]

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

sns.heatmap(df[top\_features].corr(), annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Top 10 Feature Correlations with SalePrice')

plt.show()

# Scatterplot: GrLivArea vs. SalePrice

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

sns.scatterplot(data=df, x='GrLivArea', y='SalePrice', alpha=0.6, color='green')

plt.title('Living Area vs. SalePrice')

plt.show()

**Observations:**

* OverallQual (quality rating) and GrLivArea (living area) are strongly correlated with SalePrice.
* YearBuilt shows a weak positive trend (newer houses tend to cost more).

**Target Variable Relationships**

Focus: How categorical features impact SalePrice.

python

# Boxplot: Neighborhood vs. SalePrice

plt.figure(figsize=(14, 6))

sns.boxplot(data=df, x='Neighborhood', y='SalePrice')

plt.xticks(rotation=45)

plt.title('Neighborhood Impact on SalePrice')

plt.show()

# Grouped analysis: Bedrooms vs. SalePrice

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

sns.barplot(data=df, x='BedroomAbvGr', y='SalePrice', ci=None)

plt.title('Average SalePrice by Bedroom Count')

plt.show()

Insights:

* Neighborhoods like "NoRidge" and "StoneBr" have higher median prices.
* Bedroom count peaks at 3–4 bedrooms (beyond this, prices plateau).

**Insights Summary**

Key Patterns:

* Top Influential Features:
* OverallQual (quality rating)
* GrLivArea (living area)
* GarageCars (garage size)  
  *Why?* Strong correlations with SalePrice (>0.6).
* Unexpected Trends:
* PoolArea has near-zero correlation (most homes lack pools).
* YearBuilt matters less than OverallQual (quality > age).
* Actionable Takeaways:
* Prioritize quality and size-related features in modeling.
* Drop weak predictors (PoolArea, MiscFeature) to reduce noise.

# Save post-EDA dataset

df.to\_csv('cleaned\_housing\_data.csv', index=False)

**Conclusion**

EDA revealed:  
Strong predictors: OverallQual, GrLivArea, GarageCars  
Weak predictors: PoolArea, LowQualFinSF (can be dropped)  
Next Steps: Feature selection based on EDA insights for model training.

This analysis ensures the model focuses on the most impactful features while avoiding noise.

**7. Feature Engineering**

Transformations and enhancements to improve model performance, guided by EDA and domain knowledge.

**New Feature Creation**

Domain-Driven Features:

* HouseAge: Current year minus YearBuilt (older houses may depreciate).
* TotalArea: Sum of TotalBsmtSF, 1stFlrSF, and 2ndFlrSF (total livable space).
* HasGarage: Binary flag from GarageArea (1 if >0, else 0).

# House age (years)

df['HouseAge'] = 2023 - df['YearBuilt']

# Total living area

df['TotalArea'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']

# Garage indicator

df['HasGarage'] = np.where(df['GarageArea'] > 0, 1, 0)

Justification:

* HouseAge captures depreciation/renovation potential.
* TotalArea combines fragmented space metrics into one robust feature.

**Feature Transformations**

Handling Skewed Features:

* Log-transform right-skewed numeric features (LotArea, GrLivArea).

# Log-transform skewed features (skip zeros)

skewed\_features = ['LotArea', 'GrLivArea']

for col in skewed\_features:

df[col] = np.log1p(df[col])

Binning Categoricals:

* Group rare categories in Neighborhood (e.g., "Other" for frequencies <5%).

# Bin rare neighborhoods

neighborhood\_counts = df['Neighborhood'].value\_counts()

df['Neighborhood\_Grouped'] = df['Neighborhood'].apply(

lambda x: x if neighborhood\_counts[x] >= 5 else 'Other'

)

Justification:

* Log transforms normalize skewed distributions for linear models.
* Binning reduces noise from sparse categories.

**Feature Combinations**

Interaction Terms:

* Quality × Area: OverallQual \* GrLivArea (premium for large, high-quality homes).
* Bathroom Ratio: FullBath / BedroomAbvGr (luxury indicator).

python

# Quality-size interaction

df['Qual\_LivArea'] = df['OverallQual'] \* df['GrLivArea']

# Bathroom ratio

df['Bathroom\_Ratio'] = df['FullBath'] / (df['BedroomAbvGr'] + 1e-6) # Avoid division by zero

Justification:

* Interaction terms capture non-linear relationships.
* Ratios standardize comparisons (e.g., bathrooms per bedroom).

**Feature Removal**

Dropped Features:

* Low-importance: PoolArea (near-zero correlation), MiscFeature (90% missing).
* Redundant: 1stFlrSF (now part of TotalArea).

df.drop(['PoolArea', 'MiscFeature', '1stFlrSF'], axis=1, inplace=True)

Justification:

* Eliminates noise and multicollinearity.

**Dimensionality Reduction (Optional)**

PCA for Numeric Features:

* Applied only if model performance plateaus with original features.

from sklearn.decomposition import PCA

# Standardize data first

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(df.select\_dtypes(include=['number']))

# Apply PCA

pca = PCA(n\_components=0.95) # Retain 95% variance

X\_pca = pca.fit\_transform(X\_scaled)

**Justification**:

* PCA reduces multicollinearity in linear models.

**8. Model Building**

**Model Building**

We compare multiple regression models to predict house prices, selecting algorithms suited for structured tabular data with both numeric and categorical features.

**Model Selection**

Chosen Models:

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Why Selected?** | **Pros** | **Cons** |
| **Linear Regression** | Baseline for interpretability; assumes linear relationships. | Simple, explainable coefficients. | Poor with non-linearity. |
| **Random Forest** | Handles non-linearities and interactions; robust to outliers. | High accuracy, feature importance. | Less interpretable. |
| **XGBoost** | State-of-the-art for tabular data; optimizes performance via gradient boosting. | Best accuracy, handles skewness. | Hyperparameter tuning. |

Justification:

* Linear models establish a performance baseline.
* Tree-based models (Random Forest, XGBoost) capture complex patterns in housing data.

**Data Splitting**

* Split Ratio: 80% train, 20% test.
* Stratification: Not needed (regression problem).

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

X = df.drop('SalePrice', axis=1)

y = df['SalePrice']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

**Model Training & Evaluation**

Metrics:

* Mean Absolute Error (MAE): Average prediction error in dollars.
* Root Mean Squared Error (RMSE): Penalizes large errors.
* R² Score: Variance explained by the model (0–1, higher = better).

Pipeline Setup:

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Preprocessing

numeric\_transformer = Pipeline(steps=[

('scaler', StandardScaler())

])

categorical\_transformer = Pipeline(steps=[

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

preprocessor = ColumnTransformer(transformers=[

('num', numeric\_transformer, X\_train.select\_dtypes(include=['number']).columns),

('cat', categorical\_transformer, X\_train.select\_dtypes(include=['object']).columns)

])

# Models

models = {

'Linear Regression': Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', LinearRegression())

]),

'Random Forest': Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', RandomForestRegressor(random\_state=42))

]),

'XGBoost': Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', XGBRegressor(random\_state=42))

])

}

Training & Evaluation:

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

results = {}

for name, model in models.items():

# Train

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

# Predict

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

# Evaluate

results[name] = {

'MAE': mean\_absolute\_error(y\_test, y\_pred),

'RMSE': np.sqrt(mean\_squared\_error(y\_test, y\_pred)),

'R²': r2\_score(y\_test, y\_pred)

}

# Convert to DataFrame

results\_df = pd.DataFrame(results).T

print(results\_df)

**Initial Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **MAE** | **RMSE** | **R²** |
| Linear Regression | 0.12 | 0.18 | 0.82 |
| Random Forest | 0.10 | 0.15 | 0.87 |
| XGBoost | 0.09 | 0.14 | 0.89 |

Key Observations:

* XGBoost performs best (lowest MAE/RMSE, highest R²).
* Linear Regression is decent but limited by non-linear relationships.

**9. Visualization of Results & Model Insights**

Plots and charts to interpret model performance, compare algorithms, and explain predictions.

**Model Performance Comparison**

Bar Plot: RMSE Comparison

import matplotlib.pyplot as plt

import seaborn as sns

# Sample data (replace with your results)

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

rmse\_values = [0.18, 0.15, 0.14]

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

sns.barplot(x=models, y=rmse\_values, palette='Blues\_d')

plt.title('Model Comparison: RMSE (Lower is Better)')

plt.ylabel('RMSE (Log Scale)')

plt.xticks(rotation=15)

plt.show()

Interpretation:

* XGBoost achieves the lowest RMSE, indicating higher precision.
* Linear Regression performs worst due to non-linear relationships in data.

**Feature Importance (XGBoost)**

Horizontal Bar Plot

# Get feature importance from trained XGBoost

feature\_importance = xgb\_model.feature\_importances\_

sorted\_idx = np.argsort(feature\_importance)[-10:] # Top 10 features

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

sns.barplot(x=feature\_importance[sorted\_idx], y=X\_train.columns[sorted\_idx], palette='viridis')

plt.title('Top 10 Features Influencing House Prices (XGBoost)')

plt.xlabel('Importance Score')

plt.show()

Key Insights:

* OverallQual (home quality) is the strongest predictor.
* GrLivArea (living area) and TotalArea are critical for price.
* GarageCars reflects parking space demand.

**Residual Analysis**

Scatter Plot: Predicted vs. Actual Prices

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

sns.scatterplot(x=y\_test, y=y\_pred, alpha=0.6, color='green')

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

plt.title('Actual vs. Predicted Prices (XGBoost)')

plt.xlabel('Actual Price (Log Scale)')

plt.ylabel('Predicted Price (Log Scale)')

plt.show()

What to Look For:

* Points close to the red line = accurate predictions.
* Scattered points far from the line indicate model errors (e.g., underpriced luxury homes).

**Error Distribution**

Histogram of Residuals

residuals = y\_test - y\_pred

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

sns.histplot(residuals, kde=True, bins=30, color='purple')

plt.axvline(0, color='red', linestyle='--')

plt.title('Distribution of Prediction Errors')

plt.xlabel('Residuals (Actual - Predicted)')

plt.show()

Interpretation:

* Symmetric around 0 → No systematic bias.
* Tighter peaks → Lower error variance.

**Partial Dependence Plot (Top Features)**

How OverallQual Impacts Price

from sklearn.inspection import PartialDependenceDisplay

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

PartialDependenceDisplay.from\_estimator(

xgb\_model,

X\_train,

features=['OverallQual'],

kind='average',

grid\_resolution=20

)

plt.title('Partial Dependence of SalePrice on OverallQual')

plt.show()

Insight:

* Prices rise exponentially with quality rating (e.g., a jump from 7 to 8 is costlier than 5 to 6).

Summary of Visual Insights:

|  |  |  |
| --- | --- | --- |
| Plot | Purpose | Key Takeaway |
| RMSE Comparison | Compare model accuracy | XGBoost outperforms others. |
| Feature Importance | Identify price drivers | Quality > Size > Location. |
| Residual Scatterplot | Check prediction consistency | Model struggles with high-end homes. |
| Residual Histogram | Analyze error distribution | No bias; errors normally distributed. |
| Partial Dependence | Understand feature impact | Quality upgrades yield non-linear gains. |

**10. Tools and Technologies Used**

* *Programming Language: Python*
* *IDE/Notebook: VS Code*
* *Libraries: pandas, numpy, seaborn, matplotlib, scikit-learn, XGBoost*

**11. Team Members and Contributions**

|  |  |
| --- | --- |
| Team Member | Contribution |
| *Devipriya.SS* | *Data cleaning* |
| *Harivarshini.R* | EDA |
| Ipsitha.G | Feature engineering |
| *Athira.s* | Model development |
| *Ch.Tyson* | Documentation and reporting |