**Real Estate Price Prediction Process :**

**1. Data Exploration**

**Objective:**

* Understand the dataset structure and identify any initial issues like missing values or inconsistent data types.

**Steps:**

* **Load the dataset**: We start by loading the real estate dataset to examine its structure and key statistics.

*import pandas as pd*

*df = pd.read\_csv('/path/to/real\_estate\_prices\_dataset.csv')*

*print(df.head())*

*print(df.info())*

* **Check for missing values**: Before proceeding, we check if any columns have missing data that need to be addressed.

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

**2. Data Preprocessing**

**Objective:**

* Clean and prepare the data for predictive modeling by transforming features into a suitable format and scaling numerical values.

**Steps:**

* **Drop irrelevant columns**: We removed features that are unlikely to contribute to the price prediction, such as HouseID and any redundant information.

*df*.drop(['HouseID'], axis=1, inplace=True)

* **Encode categorical variables**: We converted categorical features (e.g., Location, HouseType, Garage, etc.) into numerical values using one-hot encoding to make them suitable for model training.

*df = pd.get\_dummies(df, columns=['Location', 'HouseType', 'Garage', 'Garden', 'Basement'], drop\_first=True)*

* **Feature scaling**: We normalized numerical features like Age, MonthlyIncome, and DistanceFromHome using MinMaxScaler to bring all variables into a similar scale range, helping the model perform more effectively.

*from sklearn.preprocessing import MinMaxScaler*

*numeric\_cols = ['Age', 'MonthlyIncome', 'DistanceFromHome']*

*scaler = MinMaxScaler()*

*df[numeric\_cols] = scaler.fit\_transform(df[numeric\_cols])*

**3. Model Selection**

**Objective:**

* Select and train different machine learning models to predict real estate prices and evaluate their performance.

**Steps:**

* **Define features (X) and target (y)**: We split the dataset into input features X and the target variable y (price).

*X = df.drop('Price', axis=1) # Drop the target column from features*

*y = df['Price'] # Target variable*

* **Train-test split**: The data is split into training (90%) and testing (10%) sets to allow us to evaluate model performance on unseen data.

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

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.10, random\_state=42)*

**4. Model Evaluation**

**Objective:**

* Assess the performance of the models using different evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-Squared (R²).

**Steps:**

* **Train and evaluate a Linear Regression model**: We implemented a basic linear regression model to predict the prices and checked the performance.

*from sklearn.linear\_model import LinearRegression*

*lr\_model = LinearRegression()*

*lr\_model.fit(X\_train, y\_train)*

*y\_pred\_lr = lr\_model.predict(X\_test)*

**Evaluation**:

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

*print("Linear Regression Performance:")*

*print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred\_lr)}")*

*print(f"MSE: {mean\_squared\_error(y\_test, y\_pred\_lr)}")*

*print(f"R2: {r2\_score(y\_test, y\_pred\_lr)}")*

* **Train and evaluate a Random Forest Regressor**: To improve accuracy, we tried a more complex model like Random Forest, which is known for handling structured data well.

*from sklearn.ensemble*

*import RandomForestRegressor*

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

*rf\_model.fit(X\_train, y\_train)*

*y\_pred\_rf = rf\_model.predict(X\_test)*

**Evaluation**:

*print(f"Random Forest Performance:")*

*print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred\_rf)}")*

*print(f"MSE: {mean\_squared\_error(y\_test, y\_pred\_rf)}")*

*print(f"R2: {r2\_score(y\_test, y\_pred\_rf)}")*

* **Train and evaluate a Gradient Boosting Regressor**: Gradient Boosting is another powerful ensemble model that usually performs well on tabular data.

*from sklearn.ensemble import GradientBoostingRegressor*

*gb\_model = GradientBoostingRegressor(n\_estimators=100, random\_state=42)*

*gb\_model.fit(X\_train, y\_train)*

*y\_pred\_gb = gb\_model.predict(X\_test)*

**Evaluation**:

*print(f"Gradient Boosting Performance:")*

*print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred\_gb)}")*

*print(f"MSE: {mean\_squared\_error(y\_test, y\_pred\_gb)}")*

*print(f"R2: {r2\_score(y\_test, y\_pred\_gb)}")*

**5. Visualizations**

**Objective:**

* Use visualizations to understand the data and evaluate model performance effectively.

**Steps:**

* **Correlation heatmap**: Visualize the correlations between features to see which attributes might have the most significant impact on the target variable.

*import seaborn as sns*

*import matplotlib.pyplot as plt*

*plt.figure(figsize=(12, 8))*

*sns.heatmap(df.corr(), annot=True, cmap='coolwarm', center=0)*

*plt.title('Correlation Heatmap')*

*plt.show()*

* **Feature Importance (Gradient Boosting)**: Understand which features contributed the most to the prediction by plotting the feature importances from the Gradient Boosting model.

*feature\_importance = gb\_model.feature\_importances\_*

*feature\_names = X\_train.columns*

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

*sns.barplot(x=feature\_importance, y=feature\_names)*

*plt.title('Feature Importances (Gradient Boosting)')*

*plt.show()*

* **Price distribution boxplot**: Visualize the distribution of the target variable (Price) to understand its range and any outliers.

*sns.boxplot(y=y\_train)*

*plt.title('Box Plot of House Prices')*

*plt.show()*

**6. Conclusion**

The Real Estate Price Prediction project demonstrates a complete pipeline from data exploration to model evaluation using different machine learning algorithms.

* **Linear Regression**: Offers a baseline model with reasonable accuracy.
* **Random Forest**: Showed improved performance by capturing non-linear relationships in the data.
* **Gradient Boosting**: Provided the best predictive performance by leveraging ensemble learning techniques.

Further improvements could be achieved by tuning hyperparameters or introducing more advanced models, but the current results provide valuable insights into the real estate market's pricing dynamics.