Problem Statement :

A machine learning model is to be proposed to predict a house price based on data related to the house i.e. its area_type, availability, location, size, society, total_sqft, bath and balcony.

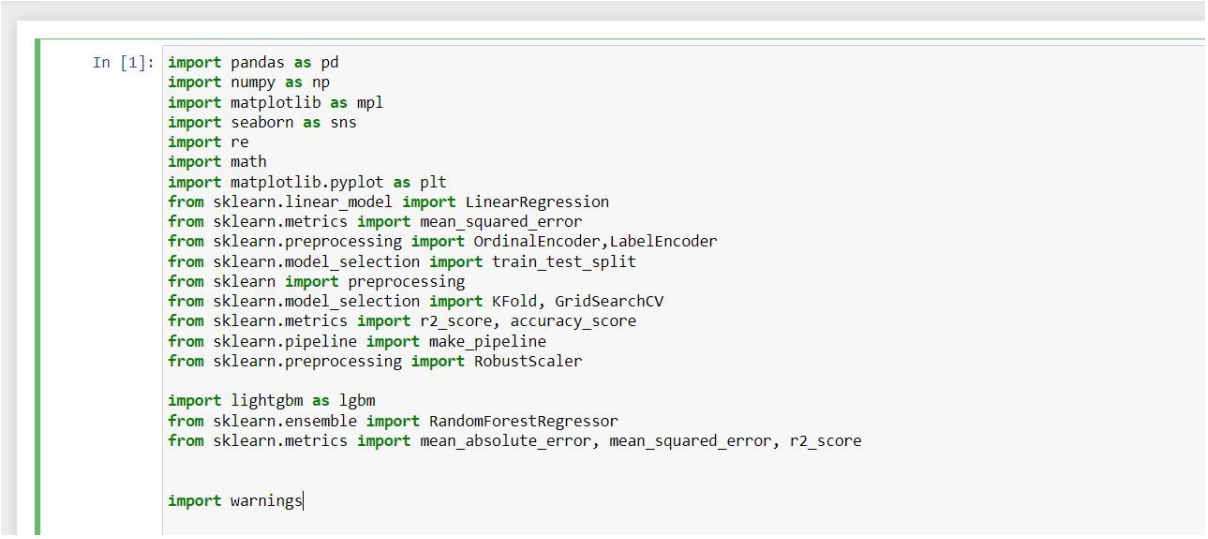
Goals of the Study:

The main objectives of this case study are as follows:

1. To apply data preprocessing and preparation techniques in order to obtain clean data (EDA).
2. To build machine learning models able to predict house price based on house features.
3. To analyze and compare models performance in order to choose the best model.

Required Libraries for Price Prediction:

Several libraries have been imported to perform price prediction. These libraries include pandas for data manipulation, numpy for numerical computing, matplotlib and seaborn for data visualization, re for regular expressions, math for mathematical functions, scikit-learn for machine learning tasks, lightgbm for the LightGBM algorithm, and warnings for managing Python interpreter warnings. These libraries provide essential functionalities for data preprocessing, modeling, evaluation, and visualization in the context of price prediction.



```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib as mpl
        import seaborn as sns
        import re
        import math
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error
        from sklearn.preprocessing import OrdinalEncoder,LabelEncoder
        from sklearn.model_selection import train_test_split
        from sklearn import preprocessing
        from sklearn.model_selection import KFold, GridSearchCV
        from sklearn.metrics import r2_score, accuracy_score
        from sklearn.pipeline import make_pipeline
        from sklearn.preprocessing import RobustScaler

        import lightgbm as lgbm
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


        import warnings
```

Data Preprocessing for Price Prediction:

1. Data Loading: The dataset "Bengaluru_House_Data.csv" is loaded into a pandas DataFrame using the `read_csv()` function from the `pandas` library.
2. Cleaning 'size' Column: The 'size' column is converted to a string data type using `.astype(str)` to ensure consistency. Then, a lambda function is applied to split the string and extract the numerical part using `(s.split()[0])`. Finally, the column is converted back to a numeric data type using `.astype(np.number)`.

3. Cleaning 'total_sqft' Column: The 'total_sqft' column is cleaned by removing specific substrings such as 'Sq. Meter', 'Sq. Yards', 'Acres', 'Cents', 'Perch', 'Grounds', 'Meter', 'Guntha', and 'Yards'. Multiple `.str.replace()` functions are used to replace these substrings with an empty string.
4. Handling Range Values in 'total_sqft' Column: The 'total_sqft' column may contain range values specified as 'x-y'. To handle this, the column is split into two columns using `.str.split('-', n=1, expand=True)`. Then, the split values are converted to floats using `pd.to_numeric()` with the `errors='coerce'` parameter to handle non-numeric values. Finally, the mean of the numeric values in the range is calculated using `.mean(axis=1)` and assigned back to the 'total_sqft' column.
5. Handling Missing Values: Missing values in the 'size', 'balcony', and 'bath' columns are filled using the mean value of each respective column. This ensures that there are no missing values that could affect the subsequent modeling process.

These preprocessing steps are performed to clean and transform the data into a suitable format for training a price prediction model. By removing unnecessary substrings, handling range values, and imputing missing values, the dataset is prepared for further analysis and modeling.

Sample Code Image: An image of the code used for data preprocessing is provided below for reference.

```
In [2]: df = pd.read_csv("Bengaluru_House_Data.csv")
df["size"] = df["size"].astype(str).apply(lambda s: (s.split()[0]))
df["size"] = df["size"].astype(np.number)

df['total_sqft'] = df['total_sqft'].str.replace('Sq. Meter', '').str.replace('Sq. Yards', '')
df['total_sqft'] = df['total_sqft'].str.replace('Acres', '').str.replace('Cents', '').str.replace('Perch', '')
df['total_sqft'] = df['total_sqft'].str.replace('Grounds', '').str.replace('Sq. Meter', '').str.replace('Guntha','')
df['total_sqft'] = df['total_sqft'].str.replace('Sq. Yards', '').str.replace('Meter', '')
df['total_sqft'] = df['total_sqft'].str.replace('Sq. Yards', '').str.replace('Meter', '')
df['total_sqft'] = df['total_sqft'].str.replace('Sq.', '').str.replace('Yards', '')
df['total_sqft'] = df['total_sqft'].str.replace('Acres', '').str.replace('Cents', '')
df['total_sqft'] = df['total_sqft'].str.replace('Grounds', '').str.replace('Meter', '')
df['total_sqft'] = df['total_sqft'].str.replace('Sq. Meter', '').str.replace('Sq. Yards', '')
df['total_sqft'] = df['total_sqft'].str.replace('Sq.', '').str.replace('Yards', '')
df['total_sqft'] = df['total_sqft'].str.replace('Acres', '').str.replace('Cents', '')
df['total_sqft'] = df['total_sqft'].str.replace('Grounds', '').str.replace('Meter', '')
df['total_sqft'] = df['total_sqft'].str.replace('Sq. Meter', '').str.replace('Sq. Yards', '')
df['total_sqft'] = df['total_sqft'].str.replace('Sq.', '').str.replace('Yards', '')
df['total_sqft'] = df['total_sqft'].str.replace('Acres', '').str.replace('Cents', '')
df['total_sqft'] = df['total_sqft'].str.replace('Grounds', '').str.replace('Meter', '')

split_values = df['total_sqft'].str.split('-', n=1, expand=True)

# Convert the split values to floats, excluding non-numeric values
numeric_values = split_values.apply(pd.to_numeric, errors='coerce')
# Calculate the average of the numeric values in the range
df['total_sqft'] = numeric_values.mean(axis=1)
df["size"]=df["size"].fillna(df['size'].mean())
df["balcony"]=df["balcony"].fillna(df['balcony'].mean())
df["bath"]=df["bath"].fillna(df["size"])
```

Price Distribution Analysis:

A histogram plot is created using sns.distplot() to visualize the distribution of house prices. The 'price' column from the DataFrame is used as the input data, and the color 'salmon' is chosen for the histogram bars. This analysis helps in understanding the range and distribution of house prices, which is essential for predicting and evaluating price values accurately.

Sample Code Image: An image of the code used for Price Distribution Analysis is provided below for reference.

```
In [4]: plt.figure(figsize=(10,5))
        sns.distplot(df['price'],color='salmon')
```

Outlier Detection and Removal:

A strip plot is created using `sns.stripplot()` to analyze the relationship between the 'size' and 'price' columns. The 'size' values are plotted on the x-axis, and the corresponding 'price' values are plotted on the y-axis. Additionally, the hue parameter is used to differentiate the data points based on the 'size' category. By examining the strip plot, outliers can be identified and investigated further.

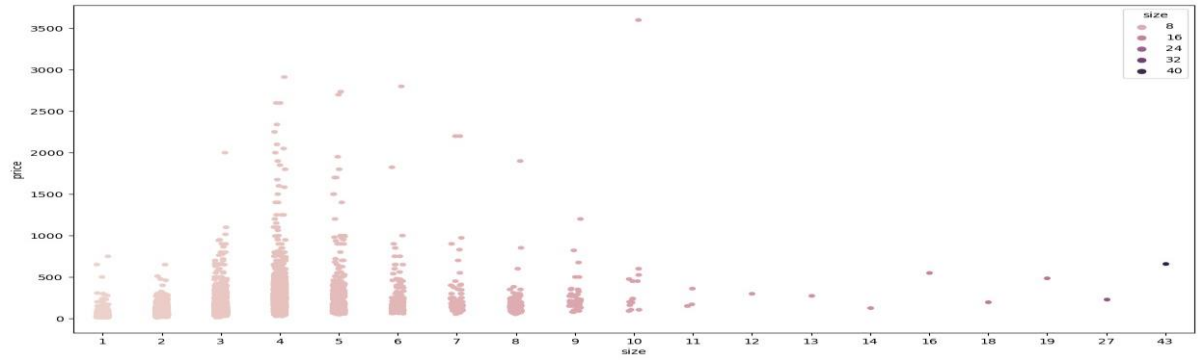
In this case, it has been observed that there are outliers where the 'size' is greater than 11 and the 'price' is greater than 1500. These outliers are represented by individual data points or dots in the strip plot. To address these outliers, they are removed from the dataset using the `drop()` function. The resulting dataset, named 'train', excludes the outlier data points based on the specified conditions. This step ensures that the outliers do not adversely affect the accuracy and performance of the price prediction model.

By identifying and removing outliers, the data is refined, leading to a more reliable and accurate prediction model. This outlier detection and removal process helps in improving the overall quality of the dataset and enhances the effectiveness of subsequent analyses and predictions.

Sample Code Image: An image of the code used for outlier detection and removal is provided below for reference.

```
In [5]: df["size"]=df["size"].astype(int)
        plt.figure(figsize=(15,8))
        sns.stripplot(x=df['size'],y=df['price'],hue=df['size'])
        plt.show()
        # df["bath"]=df["bath"].astype(int)
```

```
In [9]: # Removing Outliers
        train = df.drop(df[(df['price']>1500)
                           & (df['size']>11)].index).reset_index(drop=True)
```



Categorical Variable Encoding for Price Prediction:

To prepare the dataset for price prediction, the categorical variable 'area_type' is encoded using the OrdinalEncoder from scikit-learn.

1. Reshaping Input Data: The 'area_type' column from the 'train' dataset is extracted and reshaped using `.values.reshape(-1, 1)`. This reshaping is done to ensure that the input data is in the appropriate shape expected by the OrdinalEncoder.
2. Ordinal Encoding: The OrdinalEncoder is initialized with specified categories ['Super built-up Area', 'Built-up Area', 'Plot Area', 'Carpet Area'] using the `categories` parameter. This specifies the order in which the categories should be encoded.
3. Dropping Unnecessary Columns: The 'area_type', 'availability', and 'location' columns are dropped from the 'train' dataset using the `drop()` function. These columns are removed as they may not contribute significantly to the price prediction task.
4. Applying Encoding: The ordinal encoding is applied to the reshaped 'area_type' column using `oe.fit_transform(inputval)`. The encoded values are assigned back to the 'area_type' column of the 'train' dataset.

By encoding the categorical variable 'area_type', the data is transformed into a numerical representation that can be used as input for machine learning models. Ordinal encoding is chosen in this case to maintain the inherent order of the categories. This preprocessing step ensures that the categorical variable is appropriately represented in a format that is compatible with price prediction models.

```
In [15]: inputval = train['area_type'].values.reshape(-1,1)
         oe = OrdinalEncoder(categories=[['Super built-up  Area','Built-up  Area','Plot  Area','Carpet  Area']])
         train.drop(["area_type", "availability", "location"], axis=1, inplace= True)
         train['area_type'] = oe.fit_transform(inputval)
         train.head()
```

Data Splitting into Training and Testing Sets:

To prepare the data for building a price prediction model, the dataset is split into training and testing sets using the `train_test_split()` function from scikit-learn.

1. Feature and Target Separation: The 'price' column is separated from the 'train' dataset and assigned to the target variable `y`. The remaining columns are assigned to the feature matrix `X` using the `drop()` function, where the 'price' column is dropped along the 'axis=1'.
2. Train-Test Split: The `train_test_split()` function is used to split the data into training and testing sets. The feature matrix `X` and the target variable `y` are provided as input, along with the specified `test_size` of 0.3, indicating that 30% of the data will be allocated for testing purposes. Additionally, the `random_state` parameter is set to 0 to ensure reproducibility of the split.

3. Split Results: The split data is assigned to variables `x_train`, `x_test`, `y_train`, and `y_test`, representing the feature matrices and target variables for the training and testing sets, respectively.

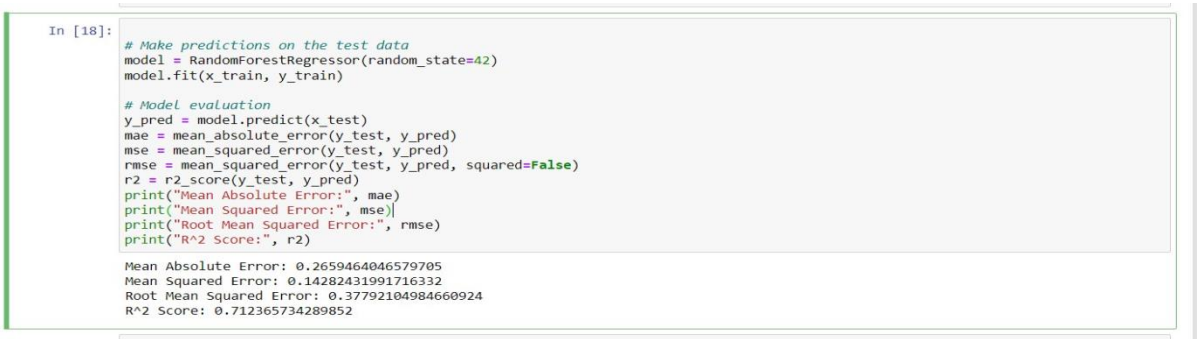
By splitting the data into training and testing sets, we can evaluate the performance of the price prediction model on unseen data. The training set, consisting of a majority portion of the data, is used for model training, while the testing set allows for assessing the model's generalization and performance on new data.

Random Forest Regression Model and Evaluation:

To predict house prices, a Random Forest Regressor model is trained and evaluated using the training and testing data.

1. Model Initialization: The Random Forest Regressor is initialized with the `random_state` parameter set to 42 to ensure reproducibility of results.
2. Model Training: The model is trained on the training data using the `fit()` method, where the feature matrix `x_train` and target variable `y_train` are provided as input.
3. Prediction on Test Data: The trained model is used to make predictions on the test data using the `predict()` method. The feature matrix `x_test` is passed as input, and the predicted values are assigned to `y_pred`.
4. Evaluation Metrics: Several evaluation metrics are computed to assess the performance of the model on the test data. The Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 Score are calculated using functions from the scikit-learn library. These metrics provide insights into the accuracy, precision, and goodness-of-fit of the model predictions.
5. Printing the Evaluation Metrics: The calculated evaluation metrics, including MAE, MSE, RMSE, and R^2 Score, are printed to the console for easy interpretation and analysis.

By training the Random Forest Regressor model and evaluating its performance using various metrics, we gain an understanding of how well the model predicts house prices. The evaluation metrics help quantify the accuracy and reliability of the model's predictions, providing valuable insights for further analysis and model improvement.



Advantages of Using Random Forest Regression for House Price Prediction:

1. Handling Nonlinearity: Random Forest Regression can effectively capture nonlinear relationships between input features and the target variable. This is important in predicting house prices since there may be complex interactions and non-linear patterns in the data.
2. Robustness to Outliers: Random Forest Regression is robust to outliers and noise in the data. It aggregates predictions from multiple decision trees, reducing the impact of individual outliers on the overall model performance.
3. Handling High-Dimensional Data: Random Forest Regression can handle datasets with a large number of features. It automatically selects a subset of features at each split, making it suitable for datasets with high dimensionality.
4. Feature Importance: Random Forest Regression provides a measure of feature importance, indicating which features contribute the most to the prediction. This information can help identify the key factors influencing house prices.
5. Handling Missing Values: Random Forest Regression can handle missing values in the data without the need for imputation. It can utilize the available information from other features to make accurate predictions even when certain features have missing values.
6. Reduced Overfitting: Random Forest Regression utilizes bagging and random feature selection, which reduces the risk of overfitting. It helps in generalizing the model and improving its performance on unseen data.

Evaluation Metrics for House Price Prediction:

Suppose we are building a machine learning model to predict house prices based on various features like size, number of bedrooms, and location. We have a dataset of 100 houses, and we split it into a training set (70 houses) and a test set (30 houses).

1. Mean Absolute Error (MAE): Let's say we make price predictions for the test set using our model. The MAE tells us the average difference between our predicted prices and the actual prices of the houses. For example, if the MAE is 10, it means, on average, our predictions have an absolute difference of $10,000 from the actual prices.
2. Mean Squared Error (MSE): Similar to MAE, MSE measures the average squared difference between our predicted prices and the actual prices. It gives higher weightage to larger errors. For instance, if the MSE is 100, it means, on average, our predictions have a squared difference of $100,000 from the actual prices.
3. Root Mean Squared Error (RMSE): RMSE is the square root of MSE, which gives us a measure of the average difference between our predicted prices and the actual prices in the original units. If the RMSE is 10, it means, on average, our predictions have a difference of $10,000 from the actual prices.
4. R^2 Score: R^2 score indicates the proportion of the variance in the target variable (house prices) that can be explained by our model. It ranges from 0 to 1, where 1 represents a perfect fit and 0 indicates that the model does not explain any variance. For example, if the R^2 score is 0.75, it means our model explains 75% of the variability in house prices.

In summary, these evaluation metrics provide insights into how well our model is performing in predicting house prices. The lower the MAE, MSE, and RMSE, and the higher the R^2 score, the better our model's predictions align with the actual house prices.

LightGBM for House Price Prediction:

1. K-Fold Cross Validation: K-Fold Cross Validation is used to evaluate the performance of the LightGBM model. It helps to estimate how the model will perform on unseen data by splitting the training data into K subsets (in this case, 10 subsets) and performing training and validation on different combinations of these subsets.
2. Grid Search: Grid Search is employed to find the best combination of hyperparameters for the LightGBM model. It allows us to specify different values for hyperparameters like learning_rate, feature_fraction, and num_leaves. The GridSearchCV function performs an exhaustive search over the specified parameter values to find the optimal configuration that minimizes the mean squared error.
3. Faster Training Speed: LightGBM is known for its fast training speed. It uses a histogram-based algorithm to build decision trees, which enables efficient and parallel computation. This makes it particularly useful for large datasets with a high number of features, as it can significantly reduce training time compared to other algorithms.
4. Lower Memory Usage: LightGBM is optimized for lower memory usage. It uses a leaf-wise tree growth strategy, which allows it to build trees in a more memory-efficient manner. This makes it feasible to train models on machines with limited memory resources.
5. Handling High-Dimensional Data: LightGBM can handle datasets with a large number of features. It automatically selects a subset of features at each split based on their importance, which helps to reduce overfitting and improve model performance. This is particularly advantageous in predicting house prices, where there can be a wide range of features influencing the target variable.
6. Improved Accuracy: LightGBM uses a gradient boosting algorithm, which iteratively improves the model by minimizing the loss function. It can effectively capture complex relationships and non-linear patterns in the data, leading to improved accuracy in predicting house prices.
7. Handling Missing Values: LightGBM can handle missing values in the data without the need for imputation. It treats missing values as a separate category during the tree construction process, allowing it to utilize the available information from other features and make accurate predictions even when certain features have missing values.

```
In [18]: warnings.filterwarnings(action='ignore')
kfold = KFold(n_splits=10, random_state = 77, shuffle = True)
# LightGBM Grid Search
params = {
    'task' : 'train',
    'objective' : 'regression',
    'subsample' : 0.8,
    'max_depth' : 7
}

param_grid = {
    'learning_rate': [0.1],
    'feature_fraction' : [0.5, 0.8],
    'num_leaves':[31, 63, 127]
}

lgbm_model = lgbm.LGBMRegressor(**params, verbose=-1)

lgbm_grid  = GridSearchCV(lgbm_model,
                          param_grid,
                          cv=kfold,
                          scoring='neg_mean_squared_error',
                          return_train_score=True)

lgbm_grid.fit(x_train, y_train)

r2_score(lgbm_grid.predict(x_train), y_train)

lgbm_model.fit(x_train,y_train)
```