In [ ]:

```
import pandas as pd
import numpy as np
from pandas import DataFrame
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
```

In [ ]:

```
data=pd.read_csv('Bengaluru_House_Data.csv')
```

In [ ]:

```
data.head()
```

In [ ]:

```
data.shape
```

In [ ]:

```
data.info()
```

In [ ]:

```
for column in data.columns:
    print(data[column].value_counts())
    print("*" * 20)
```

In [ ]:

```
print(data.isna().sum())
```

In [ ]:

```
# Dropping unnecessary columns
data.drop(columns=['area_type', 'availability', 'society', 'balcony'], inplace=True)
```

In [ ]:

```
print(data.describe())
```

In [ ]:

```
# Fill missing values
data['location'] = data['location'].fillna('Sarjapur Road')
data['size'] = data['size'].fillna('2 BHK')
data['bath'] = data['bath'].fillna(data['bath'].median())
```

In [ ]:

```
# Convert BHK to an integer
data['bhk'] = data['size'].str.split().str.get(0).astype(int)
```

In [ ]:

```
# Handle total_sqft, convert ranges to average and remove anomalies
def convertRange(x):
    temp = x.split('-')
    if len(temp) == 2:
        return (float(temp[0]) + float(temp[1])) / 2
    try:
        return float(x)
    except:
        return None
data['total_sqft'] = data['total_sqft'].apply(convertRange)
```

In [ ]:

```
# Price per square foot calculation
data['price_per_sqft'] = data['price'] * 1000000 / data['total_sqft']
```

In [ ]:

```
# Clean location names by stripping whitespaces
data['location'] = data['location'].apply(lambda x: x.strip())

# Handling rare locations
location_count = data['location'].value_counts()
location_count_less_10 = location_count[location_count <= 10]
data['location'] = data['location'].apply(lambda x: 'other' if x in location_count_less_
10 else x)
```

In [ ]:

```
# Remove outliers based on total_sqft per BHK
data = data[((data['total_sqft'] / data['bhk']) >= 300)]
```

In [ ]:

```
# Remove outliers based on price per sqft within each location
def remove_outliers_sqft(df):
    df_output = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        gen_props = subdf[(subdf.price_per_sqft > (m - st)) & (subdf.price_per_sqft <= (
m + st))]
        df_output = pd.concat([df_output, gen_props], ignore_index=True)
    return df_output
data = remove_outliers_sqft(data)
```

In [ ]:

```
# Remove BHK outliers
def bhk_outlier_remover(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                'std': np.std(bhk_df.price_per_sqft),
                'count': bhk_df.shape[0]
            }
        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk - 1)
            if stats and stats['count'] > 5:
                exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_sqf
t < stats['mean']].index.values)
    return df.drop(exclude_indices, axis='index')

data = bhk_outlier_remover(data)
```

In [ ]:

```
# Dropping unnecessary columns
data.drop(columns=['size', 'price_per_sqft'], inplace=True)
```

In [ ]:

```
# Selecting only numerical features
```

```python
# Selecting only numerical features
features = ['total_sqft', 'bath', 'bhk']
X = data[features]
y = data['price']  # Replace 'price' with the target column in your dataset
```

In [ ]:

```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print(X_train.shape)
print(X_test.shape)
```

In [ ]:

```python
# Feature Scaling for numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:

```python
from sklearn.neighbors import KNeighborsRegressor
```

In [ ]:

```python
# Initialize and train the KNN Regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)  # You can tune n_neighbors
knn_regressor.fit(X_train_scaled, y_train)
```

In [ ]:

```python
# Make predictions on the test set
y_pred = knn_regressor.predict(X_test_scaled)
```

In [ ]:

```python
# Evaluate the performance of the KNN Regressor
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [ ]:

```python
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R^2 Score: {r2:.2f}")
```