Python Code :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, ShuffleSplit,
cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb # Ensure xgboost is installed: pip install xgboost
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import re # For regex operations
import warnings

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None) # Show all columns

# --- 1. Load Data ---
try:
    df_full = pd.read_csv('Pune house data.csv')
    # print("Full data loaded successfully.") # Optional: Can keep or
remove
    # print("Original Shape:", df_full.shape) # Optional: Can keep or
remove

    # --- SELECT SUBSET: Use only the first 200 rows ---
    n_rows_to_use = 200 # Fixed number of rows
    if len(df_full) >= n_rows_to_use:
        df = df_full.head(n_rows_to_use).copy()
        # Removed print statement about selecting rows
    else:
        # Removed warning about fewer rows, just use the full df if
smaller
```

```
        df = df_full.copy()

except FileNotFoundError:
    print("Error: 'Pune house data.csv' not found. Make sure the file is
in the correct directory.")
    exit()
except Exception as e:
    print(f"An error occurred during data loading or subset selection:
{e}")
    exit()

# --- 2. Explore Data (Basic info printed during loading) ---

# --- 3. Data Cleaning & Preprocessing ---

if 'society' in df.columns:
    df.drop('society', axis=1, inplace=True)
    print("\nDropped 'society' column.")

if 'bath' in df.columns:
    df['bath'].fillna(df['bath'].median(), inplace=True)
if 'balcony' in df.columns:
    df['balcony'].fillna(df['balcony'].median(), inplace=True)
print("Filled missing 'bath' and 'balcony' values with median.")

if 'size' in df.columns:
    df.dropna(subset=['size'], inplace=True)
    def get_bhk(x):
        if pd.isna(x): return None
        tokens = x.split(' ')
        try:
            num_str = tokens[0]
            if 'RK' in x.upper(): return 1
            elif 'Bedroom' in x:
                 if num_str.isdigit(): return int(num_str)
                 else: return None
            elif 'BHK' in x.upper():
                 if num_str.isdigit(): return int(num_str)
                 else: return None
            else: return None
        except: return None
    df['bhk'] = df['size'].apply(get_bhk)
    # print(f"Rows with Null BHK after extraction attempt:
{df['bhk'].isnull().sum()}") # Removed row count info
    df.dropna(subset=['bhk'], inplace=True)
    df['bhk'] = df['bhk'].astype(int)
    df.drop('size', axis=1, inplace=True)
    print("Processed 'size' column into 'bhk'.")
else: print("Warning: 'size' column not found.")

if 'total_sqft' in df.columns:
    def convert_sqft_to_num(x):
        if isinstance(x, (int, float)): return float(x)
        if isinstance(x, str):
            x = x.strip()
            tokens = x.split('-')
            if len(tokens) == 2:
                try:
                    val1 = float(tokens[0].strip())
                    val2 = float(tokens[1].strip())
                    if val1 <= 0 or val2 <= 0 or val1 >= val2: return None
                    return (val1 + val2) / 2
                except: return None
            try:
                match = re.match(r"^([\d.]+)\s*(\w\.?\s*\w\.?|\w+)?$", x,
re.IGNORECASE)
                if match:
                    num = float(match.group(1))
                    unit = match.group(2)
                    if unit:
                        unit = unit.strip().lower().replace('.',
'').replace(' ', '')
                        if unit in ['sqmeter', 'sqm']: return num *
10.7639
                        elif unit in ['sqyards', 'sqyd']: return num * 9.0
                        elif unit == 'acres': return num * 43560.0
                        elif unit == 'perch': return num * 272.25
                        elif unit == 'cents': return num * 435.6
                        elif unit == 'guntha': return num * 1089.0
                        elif unit == 'sqft': return num
                        else: return None
                    else: return num
                else: return None
            except: return None
        return None
    df['total_sqft_num'] = df['total_sqft'].apply(convert_sqft_to_num)
    # print(f"Rows with Null total_sqft_num after extraction attempt:
{df['total_sqft_num'].isnull().sum()}") # Removed row count info
    df.dropna(subset=['total_sqft_num'], inplace=True)
    df = df[df['total_sqft_num'] > 0]
    df.drop('total_sqft', axis=1, inplace=True)
    print("Processed 'total_sqft' column into 'total_sqft_num'.")
else: print("Warning: 'total_sqft' column not found.")

if 'availability' in df.columns:
    df['avail_ready'] = df['availability'].apply(lambda x: 1 if
isinstance(x, str) and x == 'Ready To Move' else 0)
    df.drop('availability', axis=1, inplace=True)
    print("Processed 'availability' column into 'avail_ready'.")
else: print("Warning: 'availability' column not found.")

if 'site_location' in df.columns:
    df.dropna(subset=['site_location'], inplace=True)
    initial_unique_locs = df['site_location'].nunique()
    print(f"\nNumber of unique locations initially:
{initial_unique_locs}")
    location_stats = df['site_location'].value_counts(ascending=False)
    location_threshold = 5 # Keep threshold logic as it depends on data
distribution
    if initial_unique_locs < 20: location_threshold = 2
    print(f"Using location grouping threshold: {location_threshold}")
    location_stats_less_than_threshold = location_stats[location_stats <
location_threshold]
    df['site_location'] = df['site_location'].apply(lambda x: 'Other' if x
in location_stats_less_than_threshold else x)
    print(f"Number of unique locations after grouping:
{df['site_location'].nunique()}")
else: print("Warning: 'site_location' column not found.")

print("\n--- Missing Values (After Initial Cleaning) ---")
print(df.isnull().sum())
print(f"Shape after initial cleaning: {df.shape}") # Keep shape info as
it's useful for debugging
if df.empty:
    print("\nError: DataFrame is empty after initial cleaning. Cannot
proceed.")
    exit()

# --- 4. Feature Engineering ---
if 'price' in df.columns and 'total_sqft_num' in df.columns and 'bhk' in
df.columns:
    df['price'] = pd.to_numeric(df['price'], errors='coerce')
    df.dropna(subset=['price'], inplace=True)
    if df.empty:
        print("\nError: DataFrame is empty after handling non-numeric
prices. Cannot proceed.")
        exit()
    df['price_actual'] = df['price'] * 100000
    df['price_per_sqft'] = df['price_actual'] / df['total_sqft_num']
    df = df[df['bhk'] > 0]
    if df.empty:
        print("\nError: DataFrame is empty after removing BHK=0 rows.
Cannot proceed.")
        exit()
    # Ensure 'bath' exists before creating 'bath_per_bhk'
    if 'bath' in df.columns:
         df['bath_per_bhk'] = df['bath'] / df['bhk']
    else:
        print("Warning: 'bath' column not found, skipping 'bath_per_bhk'
feature.")
    df['sqft_per_bhk'] = df['total_sqft_num'] / df['bhk']

    print("\nCreated derived features ('price_actual', 'price_per_sqft',
etc.).")
else:
    print("\nError: Required columns for feature engineering not
present.")
    exit()

# --- 5. Outlier Removal ---
```

```
print(f"\nShape before outlier removal: {df.shape}") # Keep shape info

def remove_pps_outliers_iqr(df_in, factor=1.5):
    # Needs 'price_per_sqft' which might have been created in step 4
    if 'price_per_sqft' not in df_in.columns:
        print("Warning: 'price_per_sqft' not found. Skipping PPS outlier
removal.")
        return df_in

    df_out = pd.DataFrame()
    if 'site_location' in df_in.columns:
        for key, subdf in df_in.groupby('site_location'):
            if len(subdf) < 5:
                 df_out = pd.concat([df_out, subdf], ignore_index=True)
                 continue
            Q1 = subdf.price_per_sqft.quantile(0.25)
            Q3 = subdf.price_per_sqft.quantile(0.75)
            IQR = Q3 - Q1
            if IQR <= 0:
                df_out = pd.concat([df_out, subdf], ignore_index=True)
                continue
            lower_bound = Q1 - factor * IQR
            upper_bound = Q3 + factor * IQR
            reduced_df = subdf[(subdf.price_per_sqft > lower_bound) &
(subdf.price_per_sqft <= upper_bound)]
            df_out = pd.concat([df_out, reduced_df], ignore_index=True)
        return df_out
    else:
        print("Warning: 'site_location' not found. Applying PPS outlier
removal globally.")
        Q1 = df_in.price_per_sqft.quantile(0.25)
        Q3 = df_in.price_per_sqft.quantile(0.75)
        IQR = Q3 - Q1
        if IQR <= 0: return df_in
        lower_bound = Q1 - factor * IQR
        upper_bound = Q3 + factor * IQR
        return df_in[(df_in.price_per_sqft > lower_bound) &
(df_in.price_per_sqft <= upper_bound)]

df = remove_pps_outliers_iqr(df, factor=1.5)
```

```python
print(f"Shape after price_per_sqft outlier removal: {df.shape}") # Keep shape info
if df.empty:
    print("\nError: DataFrame is empty after PPS outlier removal. Cannot proceed.")
    exit()

# Check if 'sqft_per_bhk' exists before filtering
if 'sqft_per_bhk' in df.columns:
    initial_rows = df.shape[0]
    df = df[df['sqft_per_bhk'] >= 300]
    print(f"Removed {initial_rows - df.shape[0]} rows with < 300 sqft/bhk.")
    print(f"Shape after sqft_per_bhk lower bound removal: {df.shape}") # Keep shape info
else:
    print("Warning: 'sqft_per_bhk' not found. Skipping related outlier removal.")

if df.empty:
    print("\nError: DataFrame is empty after sqft_per_bhk outlier removal. Cannot proceed.")
    exit()

# Check if 'bath' and 'bhk' exist before filtering
if 'bath' in df.columns and 'bhk' in df.columns:
    initial_rows = df.shape[0]
    df = df[df.bath < df.bhk + 2]
    print(f"Removed {initial_rows - df.shape[0]} rows where bath >= bhk + 2.")
    print(f"Shape after bathroom outlier removal: {df.shape}") # Keep shape info
else:
    print("Warning: 'bath' or 'bhk' column not found. Skipping bathroom count outlier removal.")

if df.empty:
    print("\nError: DataFrame is empty after bathroom outlier removal. Cannot proceed.")
    exit()
```

```
# Drop intermediate columns if they exist
if 'price' in df.columns: df.drop(['price'], axis=1, inplace=True)
if 'price_per_sqft' in df.columns: df.drop(['price_per_sqft'], axis=1,
inplace=True)
print("Dropped intermediate columns ('price', 'price_per_sqft').")

# --- 6. Encoding & Final Feature Prep ---
categorical_features = df.select_dtypes(include=['object']).columns
numerical_features =
df.select_dtypes(include=np.number).drop(['price_actual'], axis=1).columns

print("\nNumerical Features:", list(numerical_features))
print("Categorical Features:", list(categorical_features))

print("\n--- Final Data Sample before Modeling ---")
print(df.head())
print(f"Final Data Shape for Modeling: {df.shape}") # Keep shape info

if df.shape[0] < 50:
    print("\nWarning: Very few data points remaining after
cleaning/outlier removal.")
    if df.shape[0] < 10:
         print("Error: Insufficient data points to proceed with
modeling.")
         exit()

# --- 7. Model Building ---

X = df.drop('price_actual', axis=1)
y = df['price_actual']

test_size_ratio = 0.2
# Adjust test size based on the *actual* number of rows remaining
current_rows = X.shape[0]
if current_rows * test_size_ratio < 5 and current_rows > 0: # Check
current_rows > 0
    test_size_ratio = max(0.1, 5 / current_rows)
    print(f"Adjusted test size to {test_size_ratio:.2f} due to small
dataset size.")
```

```
try:
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size_ratio, random_state=42)
    print(f"\nTrain set shape: {X_train.shape}, Test set shape:
{X_test.shape}") # Keep shape info
    if X_train.shape[0] == 0 or X_test.shape[0] == 0:
        print("Error: Train or test set is empty after splitting.")
        exit()
except ValueError as e:
    print(f"\nError during train/test split: {e}. Not enough data?")
    exit()

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore',
sparse_output=False), categorical_features)
    ],
    remainder='passthrough'
)

# --- Define Models ---
models_to_evaluate = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(alpha=1.0, random_state=42),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100,
random_state=42, n_jobs=-1),
    "XGBoost": xgb.XGBRegressor(objective='reg:squarederror',
n_estimators=100, random_state=42, n_jobs=-1),
    "KNN": KNeighborsRegressor(n_neighbors=5)
}

results = {}
fitted_pipelines = {} # Store fitted pipelines

# --- Optional: Cross-Validation ---
print("\n--- Evaluating Models using Cross-Validation (Optional - For
reference) ---")
```

```
n_cv_splits = 5
perform_cv = True
current_rows_cv = X.shape[0] # Use current rows for check
if current_rows_cv < 50: n_cv_splits = 3
if current_rows_cv < 20:
     print("Skipping Cross-Validation due to extremely small dataset size.")
     perform_cv = False
elif perform_cv: # Only define cv if we perform it
     cv = ShuffleSplit(n_splits=n_cv_splits, test_size=test_size_ratio, random_state=42)

cv_results = {}
if perform_cv:
    for name, model in models_to_evaluate.items():
        try:
            pipeline_cv = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', model)])
            if current_rows_cv < n_cv_splits: continue # Skip if too small

            cv_scores_r2 = cross_val_score(pipeline_cv, X, y, cv=cv, scoring='r2', n_jobs=-1)
            cv_results[name] = {'CV R2 Mean': cv_scores_r2.mean(), 'CV R2 Std': cv_scores_r2.std()}
            print(f"{name} - Cross-Val R2: {cv_results[name]['CV R2 Mean']:.4f} (+/- {cv_results[name]['CV R2 Std'] * 2:.4f})")
        except Exception as e:
            print(f"Error during Cross-Validation for {name}: {e}")
            cv_results[name] = {'CV R2 Mean': np.nan, 'CV R2 Std': np.nan}
    print("-" * 30)

# --- 8. Train and Evaluate Models on Train/Test Split ---
print("\n--- Training and Evaluating Models on Test Set (Default Parameters) ---")
for name, model in models_to_evaluate.items():
    try:
        pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', model)])
        pipeline.fit(X_train, y_train)
        fitted_pipelines[name] = pipeline
        y_pred = pipeline.predict(X_test)

        r2 = r2_score(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)

        results[name] = {'R2': r2, 'MAE': mae, 'MSE': mse, 'RMSE': rmse}
        print(f"{name}:")
        print(f"  R2 Score (Test): {r2:.4f}")
        print(f"  MAE (Test): {mae:,.2f}")
        print(f"  RMSE (Test): {rmse:,.2f}")
        print("-" * 30)
    except Exception as e:
        print(f"Error training/evaluating {name}: {e}")
        results[name] = {'R2': np.nan, 'MAE': np.nan, 'MSE': np.nan,
'RMSE': np.nan}


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, ShuffleSplit,
cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb # Ensure xgboost is installed: pip install xgboost
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import re # For regex operations
import warnings

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None) # Show all columns
```

```
# --- 1. Load Data ---
try:
    df_full = pd.read_csv('Pune house data.csv')
    # print("Full data loaded successfully.") # Optional: Can keep or
remove
    # print("Original Shape:", df_full.shape) # Optional: Can keep or
remove

    # --- SELECT SUBSET: Use only the first 200 rows ---
    n_rows_to_use = 200 # Fixed number of rows
    if len(df_full) >= n_rows_to_use:
        df = df_full.head(n_rows_to_use).copy()
        # Removed print statement about selecting rows
    else:
        # Removed warning about fewer rows, just use the full df if
smaller
        df = df_full.copy()

except FileNotFoundError:
    print("Error: 'Pune house data.csv' not found. Make sure the file is
in the correct directory.")
    exit()
except Exception as e:
    print(f"An error occurred during data loading or subset selection:
{e}")
    exit()

# --- 2. Explore Data (Basic info printed during loading) ---

# --- 3. Data Cleaning & Preprocessing ---

if 'society' in df.columns:
    df.drop('society', axis=1, inplace=True)
    print("\nDropped 'society' column.")

if 'bath' in df.columns:
    df['bath'].fillna(df['bath'].median(), inplace=True)
if 'balcony' in df.columns:
    df['balcony'].fillna(df['balcony'].median(), inplace=True)
```

```
print("Filled missing 'bath' and 'balcony' values with median.")

if 'size' in df.columns:
    df.dropna(subset=['size'], inplace=True)
    def get_bhk(x):
        if pd.isna(x): return None
        tokens = x.split(' ')
        try:
            num_str = tokens[0]
            if 'RK' in x.upper(): return 1
            elif 'Bedroom' in x:
                 if num_str.isdigit(): return int(num_str)
                 else: return None
            elif 'BHK' in x.upper():
                 if num_str.isdigit(): return int(num_str)
                 else: return None
            else: return None
        except: return None
    df['bhk'] = df['size'].apply(get_bhk)
    # print(f"Rows with Null BHK after extraction attempt:
{df['bhk'].isnull().sum()}") # Removed row count info
    df.dropna(subset=['bhk'], inplace=True)
    df['bhk'] = df['bhk'].astype(int)
    df.drop('size', axis=1, inplace=True)
    print("Processed 'size' column into 'bhk'.")
else: print("Warning: 'size' column not found.")

if 'total_sqft' in df.columns:
    def convert_sqft_to_num(x):
        if isinstance(x, (int, float)): return float(x)
        if isinstance(x, str):
            x = x.strip()
            tokens = x.split('-')
            if len(tokens) == 2:
                try:
                    val1 = float(tokens[0].strip())
                    val2 = float(tokens[1].strip())
                    if val1 <= 0 or val2 <= 0 or val1 >= val2: return None
                    return (val1 + val2) / 2
                except: return None
            try:
                match = re.match(r"^([\d.]+)\s*(\w\.?\s*\w\.?|\w+)?$", x,
re.IGNORECASE)
                if match:
                    num = float(match.group(1))
                    unit = match.group(2)
                    if unit:
                        unit = unit.strip().lower().replace('.',
'').replace(' ', '')
                        if unit in ['sqmeter', 'sqm']: return num *
10.7639
                        elif unit in ['sqyards', 'sqyd']: return num * 9.0
                        elif unit == 'acres': return num * 43560.0
                        elif unit == 'perch': return num * 272.25
                        elif unit == 'cents': return num * 435.6
                        elif unit == 'guntha': return num * 1089.0
                        elif unit == 'sqft': return num
                        else: return None
                    else: return num
                else: return None
            except: return None
        return None
    df['total_sqft_num'] = df['total_sqft'].apply(convert_sqft_to_num)
    # print(f"Rows with Null total_sqft_num after extraction attempt:
{df['total_sqft_num'].isnull().sum()}") # Removed row count info
    df.dropna(subset=['total_sqft_num'], inplace=True)
    df = df[df['total_sqft_num'] > 0]
    df.drop('total_sqft', axis=1, inplace=True)
    print("Processed 'total_sqft' column into 'total_sqft_num'.")
else: print("Warning: 'total_sqft' column not found.")

if 'availability' in df.columns:
    df['avail_ready'] = df['availability'].apply(lambda x: 1 if
isinstance(x, str) and x == 'Ready To Move' else 0)
    df.drop('availability', axis=1, inplace=True)
    print("Processed 'availability' column into 'avail_ready'.")
else: print("Warning: 'availability' column not found.")

if 'site_location' in df.columns:
    df.dropna(subset=['site_location'], inplace=True)
    initial_unique_locs = df['site_location'].nunique()
    print(f"\nNumber of unique locations initially: {initial_unique_locs}")
    location_stats = df['site_location'].value_counts(ascending=False)
    location_threshold = 5 # Keep threshold logic as it depends on data distribution
    if initial_unique_locs < 20: location_threshold = 2
    print(f"Using location grouping threshold: {location_threshold}")
    location_stats_less_than_threshold = location_stats[location_stats < location_threshold]
    df['site_location'] = df['site_location'].apply(lambda x: 'Other' if x in location_stats_less_than_threshold else x)
    print(f"Number of unique locations after grouping: {df['site_location'].nunique()}")
else: print("Warning: 'site_location' column not found.")

print("\n--- Missing Values (After Initial Cleaning) ---")
print(df.isnull().sum())
print(f"Shape after initial cleaning: {df.shape}") # Keep shape info as it's useful for debugging
if df.empty:
    print("\nError: DataFrame is empty after initial cleaning. Cannot proceed.")
    exit()

# --- 4. Feature Engineering ---
if 'price' in df.columns and 'total_sqft_num' in df.columns and 'bhk' in df.columns:
    df['price'] = pd.to_numeric(df['price'], errors='coerce')
    df.dropna(subset=['price'], inplace=True)
    if df.empty:
        print("\nError: DataFrame is empty after handling non-numeric prices. Cannot proceed.")
        exit()
    df['price_actual'] = df['price'] * 100000
    df['price_per_sqft'] = df['price_actual'] / df['total_sqft_num']
    df = df[df['bhk'] > 0]
    if df.empty:
        print("\nError: DataFrame is empty after removing BHK=0 rows. Cannot proceed.")
        exit()
    # Ensure 'bath' exists before creating 'bath_per_bhk'
    if 'bath' in df.columns:
         df['bath_per_bhk'] = df['bath'] / df['bhk']
    else:
        print("Warning: 'bath' column not found, skipping 'bath_per_bhk'
feature.")
    df['sqft_per_bhk'] = df['total_sqft_num'] / df['bhk']

    print("\nCreated derived features ('price_actual', 'price_per_sqft',
etc.).")
else:
    print("\nError: Required columns for feature engineering not
present.")
    exit()

# --- 5. Outlier Removal ---
print(f"\nShape before outlier removal: {df.shape}") # Keep shape info

def remove_pps_outliers_iqr(df_in, factor=1.5):
    # Needs 'price_per_sqft' which might have been created in step 4
    if 'price_per_sqft' not in df_in.columns:
        print("Warning: 'price_per_sqft' not found. Skipping PPS outlier
removal.")
        return df_in

    df_out = pd.DataFrame()
    if 'site_location' in df_in.columns:
        for key, subdf in df_in.groupby('site_location'):
            if len(subdf) < 5:
                 df_out = pd.concat([df_out, subdf], ignore_index=True)
                 continue
            Q1 = subdf.price_per_sqft.quantile(0.25)
            Q3 = subdf.price_per_sqft.quantile(0.75)
            IQR = Q3 - Q1
            if IQR <= 0:
                df_out = pd.concat([df_out, subdf], ignore_index=True)
                continue
            lower_bound = Q1 - factor * IQR
            upper_bound = Q3 + factor * IQR
            reduced_df = subdf[(subdf.price_per_sqft > lower_bound) & (subdf.price_per_sqft <= upper_bound)]
            df_out = pd.concat([df_out, reduced_df], ignore_index=True)
        return df_out
    else:
        print("Warning: 'site_location' not found. Applying PPS outlier removal globally.")
        Q1 = df_in.price_per_sqft.quantile(0.25)
        Q3 = df_in.price_per_sqft.quantile(0.75)
        IQR = Q3 - Q1
        if IQR <= 0: return df_in
        lower_bound = Q1 - factor * IQR
        upper_bound = Q3 + factor * IQR
        return df_in[(df_in.price_per_sqft > lower_bound) & (df_in.price_per_sqft <= upper_bound)]

df = remove_pps_outliers_iqr(df, factor=1.5)
print(f"Shape after price_per_sqft outlier removal: {df.shape}") # Keep shape info
if df.empty:
    print("\nError: DataFrame is empty after PPS outlier removal. Cannot proceed.")
    exit()

# Check if 'sqft_per_bhk' exists before filtering
if 'sqft_per_bhk' in df.columns:
    initial_rows = df.shape[0]
    df = df[df['sqft_per_bhk'] >= 300]
    print(f"Removed {initial_rows - df.shape[0]} rows with < 300 sqft/bhk.")
    print(f"Shape after sqft_per_bhk lower bound removal: {df.shape}") # Keep shape info
else:
    print("Warning: 'sqft_per_bhk' not found. Skipping related outlier removal.")

if df.empty:
    print("\nError: DataFrame is empty after sqft_per_bhk outlier removal. Cannot proceed.")
    exit()
```

```python
# Check if 'bath' and 'bhk' exist before filtering
if 'bath' in df.columns and 'bhk' in df.columns:
    initial_rows = df.shape[0]
    df = df[df.bath < df.bhk + 2]
    print(f"Removed {initial_rows - df.shape[0]} rows where bath >= bhk + 2.")
    print(f"Shape after bathroom outlier removal: {df.shape}") # Keep shape info
else:
    print("Warning: 'bath' or 'bhk' column not found. Skipping bathroom count outlier removal.")

if df.empty:
    print("\nError: DataFrame is empty after bathroom outlier removal. Cannot proceed.")
    exit()

# Drop intermediate columns if they exist
if 'price' in df.columns: df.drop(['price'], axis=1, inplace=True)
if 'price_per_sqft' in df.columns: df.drop(['price_per_sqft'], axis=1, inplace=True)
print("Dropped intermediate columns ('price', 'price_per_sqft').")

# --- 6. Encoding & Final Feature Prep ---
categorical_features = df.select_dtypes(include=['object']).columns
numerical_features = df.select_dtypes(include=np.number).drop(['price_actual'], axis=1).columns

print("\nNumerical Features:", list(numerical_features))
print("Categorical Features:", list(categorical_features))

print("\n--- Final Data Sample before Modeling ---")
print(df.head())
print(f"Final Data Shape for Modeling: {df.shape}") # Keep shape info

if df.shape[0] < 50:
    print("\nWarning: Very few data points remaining after cleaning/outlier removal.")
    if df.shape[0] < 10:
         print("Error: Insufficient data points to proceed with
modeling.")
         exit()

# --- 7. Model Building ---

X = df.drop('price_actual', axis=1)
y = df['price_actual']

test_size_ratio = 0.2
# Adjust test size based on the *actual* number of rows remaining
current_rows = X.shape[0]
if current_rows * test_size_ratio < 5 and current_rows > 0: # Check
current_rows > 0
    test_size_ratio = max(0.1, 5 / current_rows)
    print(f"Adjusted test size to {test_size_ratio:.2f} due to small
dataset size.")

try:
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size_ratio, random_state=42)
    print(f"\nTrain set shape: {X_train.shape}, Test set shape:
{X_test.shape}") # Keep shape info
    if X_train.shape[0] == 0 or X_test.shape[0] == 0:
        print("Error: Train or test set is empty after splitting.")
        exit()
except ValueError as e:
    print(f"\nError during train/test split: {e}. Not enough data?")
    exit()

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore',
sparse_output=False), categorical_features)
    ],
    remainder='passthrough'
)

# --- Define Models ---
```

```
models_to_evaluate = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(alpha=1.0, random_state=42),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100,
random_state=42, n_jobs=-1),
    "XGBoost": xgb.XGBRegressor(objective='reg:squarederror',
n_estimators=100, random_state=42, n_jobs=-1),
    "KNN": KNeighborsRegressor(n_neighbors=5)
}

results = {}
fitted_pipelines = {} # Store fitted pipelines

# --- Optional: Cross-Validation ---
print("\n--- Evaluating Models using Cross-Validation (Optional - For
reference) ---")
n_cv_splits = 5
perform_cv = True
current_rows_cv = X.shape[0] # Use current rows for check
if current_rows_cv < 50: n_cv_splits = 3
if current_rows_cv < 20:
     print("Skipping Cross-Validation due to extremely small dataset
size.")
     perform_cv = False
elif perform_cv: # Only define cv if we perform it
     cv = ShuffleSplit(n_splits=n_cv_splits, test_size=test_size_ratio,
random_state=42)

cv_results = {}
if perform_cv:
    for name, model in models_to_evaluate.items():
        try:
            pipeline_cv = Pipeline(steps=[('preprocessor', preprocessor),
('regressor', model)])
            if current_rows_cv < n_cv_splits: continue # Skip if too small

            cv_scores_r2 = cross_val_score(pipeline_cv, X, y, cv=cv,
scoring='r2', n_jobs=-1)
```

```
            cv_results[name] = {'CV R2 Mean': cv_scores_r2.mean(), 'CV R2 Std': cv_scores_r2.std()}
            print(f"{name} - Cross-Val R2: {cv_results[name]['CV R2 Mean']:.4f} (+/- {cv_results[name]['CV R2 Std'] * 2:.4f})")
        except Exception as e:
            print(f"Error during Cross-Validation for {name}: {e}")
            cv_results[name] = {'CV R2 Mean': np.nan, 'CV R2 Std': np.nan}
    print("-" * 30)

# --- 8. Train and Evaluate Models on Train/Test Split ---
print("\n--- Training and Evaluating Models on Test Set (Default Parameters) ---")
for name, model in models_to_evaluate.items():
    try:
        pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('regressor', model)])
        pipeline.fit(X_train, y_train)
        fitted_pipelines[name] = pipeline
        y_pred = pipeline.predict(X_test)

        r2 = r2_score(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)

        results[name] = {'R2': r2, 'MAE': mae, 'MSE': mse, 'RMSE': rmse}
        print(f"{name}:")
        print(f"  R2 Score (Test): {r2:.4f}")
        print(f"  MAE (Test): {mae:,.2f}")
        print(f"  RMSE (Test): {rmse:,.2f}")
        print("-" * 30)
    except Exception as e:
        print(f"Error training/evaluating {name}: {e}")
        results[name] = {'R2': np.nan, 'MAE': np.nan, 'MSE': np.nan, 'RMSE': np.nan}


print("\n--- Model Performance Summary (Test Set, Default Params) ---")
best_model_name = None # Initialize
if results: # Check if results dict is populated
```

```
    results_df = pd.DataFrame(results).T.sort_values(by='R2',
ascending=False)
    # Display relevant columns nicely
    print(results_df[['R2', 'MAE', 'RMSE']])

    # Identify best model based on Test R2
    valid_results = results_df.dropna(subset=['R2']) # Ensure we only
consider models that ran successfully
    if not valid_results.empty:
        best_model_name = valid_results.index[0]
        best_r2 = valid_results.iloc[0]['R2']
        print(f"\nBest performing model on Test Set (default params):
{best_model_name} (R2 = {best_r2:.4f})")

        # --- ADD GRAPHING CODE HERE ---
        print("\n--- Generating R2 Score Comparison Graph ---")
        try:
            plt.figure(figsize=(10, 6))
            # Use the valid results DataFrame for plotting
            plot_data = valid_results['R2']
            bars = sns.barplot(x=plot_data.values, y=plot_data.index,
palette="viridis", orient='h') # Use plot_data index for y-axis

            # Add R2 scores as labels on the bars
            # Use ax.bar_label (requires matplotlib >= 3.4)
            ax = plt.gca() # Get current axes
            ax.bar_label(ax.containers[0], fmt='%.4f', padding=3) # Add
labels to the bars

            # Alternative for older matplotlib:
            # for index, value in enumerate(plot_data):
            #     plt.text(value + 0.01, index, f'{value:.4f}',
va='center') # Adjust position slightly

            plt.title('Comparison of Model R2 Scores (Test Set)')
            plt.xlabel('R2 Score')
            plt.ylabel('Model')
            plt.xlim(min(0, plot_data.min() - 0.05), max(plot_data.max() +
0.05 , 1.0)) # Adjust x-axis limits
            plt.tight_layout()
            plt.show()
            print("Graph generated successfully.")

        except Exception as e:
            print(f"Could not generate R2 comparison graph: {e}")
        # --- END OF GRAPHING CODE ---

        # --- 10. Feature Importance for the Best Default Model ---
        # (Feature importance code remains here, using 'best_model_name')
        if best_model_name: # Check if best_model_name was successfully
identified
             try:
                best_pipeline = fitted_pipelines.get(best_model_name)
                if best_pipeline:
                    final_regressor =
best_pipeline.named_steps['regressor']
                    if hasattr(final_regressor, 'feature_importances_'):
                        fitted_preprocessor =
best_pipeline.named_steps['preprocessor']
                        ohe_feature_names = []
                        cat_transformer =
fitted_preprocessor.named_transformers_.get('cat')
                        if cat_transformer is not None and
isinstance(cat_transformer, OneHotEncoder) and len(categorical_features) >
0:
                             if hasattr(cat_transformer, 'categories_'):
                                 ohe_feature_names =
cat_transformer.get_feature_names_out(categorical_features)

                        current_numerical_features =
list(X_train.select_dtypes(include=np.number).columns) # Get numerical
features from training data columns
                        feature_names = current_numerical_features +
list(ohe_feature_names)
                        importances = final_regressor.feature_importances_

                        if len(feature_names) == len(importances):
                            importance_series = pd.Series(importances,
index=feature_names).sort_values(ascending=False)
                            n_top_features = min(20, len(feature_names))
                            plt.figure(figsize=(10, max(6, n_top_features
* 0.4)))

sns.barplot(x=importance_series[:n_top_features],
y=importance_series.index[:n_top_features])
                            plt.title(f'Top {n_top_features} Feature
Importances for {best_model_name} (Default Params)')
                            plt.xlabel('Importance Score')
                            plt.ylabel('Features')
                            plt.tight_layout()
                            plt.show()
                        else:
                            print(f"\nWarning: Mismatch between feature
names ({len(feature_names)}) and importances ({len(importances)}).
Skipping feature importance plot.")
                            print("Feature Names Found:", feature_names)
                    else:
                        print(f"\nThe best model ({best_model_name}) does
not support feature_importances_.")
                else:
                     print(f"\nCould not retrieve the fitted pipeline for
the best model ({best_model_name}). Skipping feature importance plot.")
             except Exception as e:
                print(f"\nCould not generate feature importance plot:
{e}")
        else:
            print("\nSkipping feature importance plot as best model could
not be determined.")
        # --- End of Feature Importance ---

    else:
        print("\nNo valid models found after evaluation to determine the
best performer or plot comparison.")

else:
    print("\nNo model evaluation results generated.")


print("\n--- Conclusion ---")
# Removed sentence about number of rows used
```

```python
print("Models were trained with default parameters.")
print("Performance metrics reflect model behavior on the processed data.")
```

Results :

```
Dropped 'society' column.
Filled missing 'bath' and 'balcony' values with median.
Processed 'size' column into 'bhk'.
Processed 'total_sqft' column into 'total_sqft_num'.
Processed 'availability' column into 'avail_ready'.

Number of unique locations initially: 97
Using location grouping threshold: 5
Number of unique locations after grouping: 1

--- Missing Values (After Initial Cleaning) ---
area_type         0
bath              0
balcony           0
price             0
site_location     0
bhk               0
total_sqft_num    0
avail_ready       0
dtype: int64
Shape after initial cleaning: (200, 8)

Created derived features ('price_actual', 'price_per_sqft', etc.).

Shape before outlier removal: (200, 12)
Shape after price_per_sqft outlier removal: (181, 12)
Removed 5 rows with < 300 sqft/bhk.
Shape after sqft_per_bhk lower bound removal: (176, 12)
Removed 1 rows where bath >= bhk + 2.
Shape after bathroom outlier removal: (175, 12)
Dropped intermediate columns ('price', 'price_per_sqft').

Numerical Features: ['bath', 'balcony', 'bhk', 'total_sqft_num',
'avail_ready', 'bath_per_bhk', 'sqft_per_bhk']
Categorical Features: ['area_type', 'site_location']

--- Final Data Sample before Modeling ---
              area_type  bath  balcony site_location  bhk  total_sqft_num  
\
0  Super built-up  Area   2.0      1.0         Other    2          1056.0
1            Plot  Area   5.0      3.0         Other    4          2600.0
2        Built-up  Area   2.0      3.0         Other    3          1440.0
3  Super built-up  Area   3.0      1.0         Other    3          1521.0
4  Super built-up  Area   2.0      1.0         Other    2          1200.0

   avail_ready  price_actual  bath_per_bhk  sqft_per_bhk
0            0     3907000.0      1.000000         528.0
1            1    12000000.0      1.250000         650.0
2            1     6200000.0      0.666667         480.0
3            1     9500000.0      1.000000         507.0
4            1     5100000.0      1.000000         600.0
Final Data Shape for Modeling: (175, 10)

Train set shape: (140, 9), Test set shape: (35, 9)

--- Evaluating Models using Cross-Validation (Optional - For reference)
---
Linear Regression - Cross-Val R2: 0.7818 (+/- 0.1606)
Lasso - Cross-Val R2: 0.7818 (+/- 0.1606)
Decision Tree - Cross-Val R2: 0.6675 (+/- 0.3195)
Random Forest - Cross-Val R2: 0.7962 (+/- 0.1080)
XGBoost - Cross-Val R2: 0.7761 (+/- 0.0880)
KNN - Cross-Val R2: 0.6270 (+/- 0.1897)
------------------------------

--- Training and Evaluating Models on Test Set (Default Parameters) ---
Linear Regression:
  R2 Score (Test): 0.7626
  MAE (Test): 2,437,157.18
  RMSE (Test): 3,396,172.02
------------------------------
Lasso:
  R2 Score (Test): 0.7626
  MAE (Test): 2,437,158.58
  RMSE (Test): 3,396,182.55
------------------------------
Decision Tree:
  R2 Score (Test): 0.7456
  MAE (Test): 2,476,271.43
  RMSE (Test): 3,515,399.17
------------------------------
Random Forest:
  R2 Score (Test): 0.7965
  MAE (Test): 2,228,317.56
  RMSE (Test): 3,144,004.22
------------------------------
```

```
XGBoost:
  R2 Score (Test): 0.7996
  MAE (Test): 2,185,530.29
  RMSE (Test): 3,120,276.55
------------------------------
KNN:
  R2 Score (Test): 0.6313
  MAE (Test): 2,898,580.00
  RMSE (Test): 4,232,196.17
------------------------------

Dropped 'society' column.
Filled missing 'bath' and 'balcony' values with median.
Processed 'size' column into 'bhk'.
Processed 'total_sqft' column into 'total_sqft_num'.
Processed 'availability' column into 'avail_ready'.

Number of unique locations initially: 97
Using location grouping threshold: 5
Number of unique locations after grouping: 1

--- Missing Values (After Initial Cleaning) ---
area_type         0
bath              0
balcony           0
price             0
site_location     0
bhk               0
total_sqft_num    0
avail_ready       0
dtype: int64
Shape after initial cleaning: (200, 8)

Created derived features ('price_actual', 'price_per_sqft', etc.).

Shape before outlier removal: (200, 12)
Shape after price_per_sqft outlier removal: (181, 12)
Removed 5 rows with < 300 sqft/bhk.
Shape after sqft_per_bhk lower bound removal: (176, 12)
Removed 1 rows where bath >= bhk + 2.
Shape after bathroom outlier removal: (175, 12)
Dropped intermediate columns ('price', 'price_per_sqft').

Numerical Features: ['bath', 'balcony', 'bhk', 'total_sqft_num',
'avail_ready', 'bath_per_bhk', 'sqft_per_bhk']
Categorical Features: ['area_type', 'site_location']
```

```
--- Final Data Sample before Modeling ---
              area_type  bath  balcony site_location  bhk  total_sqft_num  
\
0  Super built-up  Area   2.0      1.0         Other    2          1056.0   
1             Plot  Area   5.0      3.0         Other    4          2600.0   
2         Built-up  Area   2.0      3.0         Other    3          1440.0   
3  Super built-up  Area   3.0      1.0         Other    3          1521.0   
4  Super built-up  Area   2.0      1.0         Other    2          1200.0   

   avail_ready  price_actual  bath_per_bhk  sqft_per_bhk  
0            0     3907000.0      1.000000         528.0  
1            1    12000000.0      1.250000         650.0  
2            1     6200000.0      0.666667         480.0  
3            1     9500000.0      1.000000         507.0  
4            1     5100000.0      1.000000         600.0  
Final Data Shape for Modeling: (175, 10)

Train set shape: (140, 9), Test set shape: (35, 9)

--- Evaluating Models using Cross-Validation (Optional - For reference)
---
Linear Regression - Cross-Val R2: 0.7818 (+/- 0.1606)
Lasso - Cross-Val R2: 0.7818 (+/- 0.1606)
Decision Tree - Cross-Val R2: 0.6675 (+/- 0.3195)
Random Forest - Cross-Val R2: 0.7962 (+/- 0.1080)
XGBoost - Cross-Val R2: 0.7761 (+/- 0.0880)
KNN - Cross-Val R2: 0.6270 (+/- 0.1897)
------------------------------

--- Training and Evaluating Models on Test Set (Default Parameters) ---
Linear Regression:
  R2 Score (Test): 0.7626
  MAE (Test): 2,437,157.18
  RMSE (Test): 3,396,172.02
------------------------------
Lasso:
  R2 Score (Test): 0.7626
  MAE (Test): 2,437,158.58
  RMSE (Test): 3,396,182.55
------------------------------
Decision Tree:
  R2 Score (Test): 0.7456
  MAE (Test): 2,476,271.43
  RMSE (Test): 3,515,399.17
------------------------------
Random Forest:
  R2 Score (Test): 0.7965
  MAE (Test): 2,228,317.56
  RMSE (Test): 3,144,004.22
------------------------------
XGBoost:
  R2 Score (Test): 0.7996
  MAE (Test): 2,185,530.29
  RMSE (Test): 3,120,276.55
------------------------------
KNN:
  R2 Score (Test): 0.6313
  MAE (Test): 2,898,580.00
  RMSE (Test): 4,232,196.17
------------------------------

--- Model Performance Summary (Test Set, Default Params) ---
                         R2           MAE          RMSE
XGBoost            0.799584  2.185530e+06  3.120277e+06
Random Forest      0.796524  2.228318e+06  3.144004e+06
Linear Regression  0.762576  2.437157e+06  3.396172e+06
Lasso              0.762574  2.437159e+06  3.396183e+06
Decision Tree      0.745613  2.476271e+06  3.515399e+06
KNN                0.631296  2.898580e+06  4.232196e+06

Best performing model on Test Set (default params): XGBoost (R2 = 0.7996)
```