Senior Design Project

House Price Prediction Using Ensemble Learning Techniques

Under the guidance of:-
Dr. Manomita Chakraborty

By:-
Raghunath Singh (20BCI7012)
Anuj (20BCE7055)

AGENDA

PROBLEM DEFINITION

ABOUT DATASET

METHODOLOGY

CODE

OUTPUT

SUMMARY

PROBLEM DEFINITION

To design an Machine Learning Algorithm for the accurate prediction of House Price using Random Forest, Support Vector Regression, Boosting algorithm and Ensemble models.

ABOUT DATASET

The dataset has been taken from Kaggle.

It has following features:-
ID: Unique identifier for each house listing.
Date: Date of the house listing.
Number of bedrooms: The count of bedrooms in the house.
Number of bathrooms: The count of bathrooms in the house.
Living area: Total living area of the house in square feet.
Lot area: Total area of the lot in square feet.
Number of floors: Number of floors in the house.
Waterfront present: Indicates whether the house has a waterfront view (binary: 0 for no, 1 for yes).
Number of schools nearby: Number of schools located near the house.
Distance from the airport: Distance of the house from the nearest airport in miles.
Price: Price of the house.

ABOUT DATASET

Number of views: Number of views the house has received.
Condition of the house: Condition rating of the house.
Grade of the house: Grade rating of the house.
Area of the house (excluding basement): Total area of the house excluding the basement in square feet.
Area of the basement: Area of the basement in square feet.
Built Year: Year the house was originally built.
Renovation Year: Year of the last renovation, if any.
Postal Code: Postal code of the house location.
Latitude: Latitude coordinate of the house location.
Longitude: Longitude coordinate of the house location.
Living area after renovation: Total living area of the house after renovation in square feet.
Lot area after renovation: Total area of the lot after renovation in square feet.

Kaggle Link:- https://www.kaggle.com/datasets/mohamedafsal007/house-price-dataset-of-india

Methodology

Data Exploration:
Loaded the dataset and checked for missing values.
Explored data types and basic statistics.

Visualization:
Plotted count distributions for various features.
Visualized geographical data and price distributions.
Examined correlations and relationships between variables.

Feature Engineering:
Created new features like property age and total square footage
.
Feature Selection:
Identified and removed highly correlated features

.

Methodology

Modeling:
Split data for training and testing.
Trained RandomForest, XGBoost, and CatBoost models.
Evaluated models using R-squared scores.
Combined predictions to create an ensemble model.

Model Selection:
Selected the best-performing model based on R-squared scores.

Deployment:
Saved the selected model for future use.

Code

```
import pandas as pd
import numpy as np
df=pd.read_csv('House Price India.csv’)
df.head(5)
pd.isnull(df).head()
df.isna().sum()
df.info()
# returns a dataframe with column names of the dataset
pd.DataFrame(list(df.columns), columns=['Column Name’])
from matplotlib import pyplot as plt
import seaborn as sns
# used to return a countplot with null title and figsize to be (15, 8) as default

def countplot(dataframe, x_val, plot_title='', figsize=(15,8)):
    plt.figure(figsize=figsize)
    plt.title(plot_title)
    sns.countplot(data=df, x=x_val)
    plt.show()
```

Code

```
# used to return a barplot
def barplot(dataframe, x_val, y_val):
    sns.barplot(data=dataframe, x=x_val, y=y_val)
    plt.title(x_val.title() + ' vs ' + y_val.title())
    plt.show()
df.shape
#Univariate Analysis
# returns the countplot of bedrooms
countplot(dataframe=df, x_val='number of bedrooms', plot_title='Count of Number of Bedrooms’)
# returns the countplot of bathrooms
countplot(dataframe=df, x_val='number of bathrooms', plot_title='Count of Number of Bathrooms’)
# returns the countplot of floors
countplot(dataframe=df, x_val='number of floors', figsize=(8,5), plot_title='Count of Number of Floors’)
# returns the countplot of number water present or not
countplot(dataframe=df, x_val='waterfront present', figsize=(8,6), plot_title='Water present (1) or not (0)')
```

Code

```
# returns the countplot of views
countplot(dataframe=df, x_val='number of views', figsize=(8,5), plot_title='Number of Views')
plt.figure(figsize=(10,10))
sns.jointplot(x=df.Lattitude.values, y=df.Longitude.values,size=10)
plt.ylabel('Longitude',fontsize=12)
plt.xlabel('Latitude',fontsize=12)
plt.show()
sns.despine
# returns the countplot of House Condition
countplot(dataframe=df, x_val='condition of the house', figsize=(8,5), plot_title='House Condition')
# returns the countplot of each grade of house
countplot(dataframe=df, x_val='grade of the house', figsize=(8,5), plot_title='Grade of the House')
# returns the countplot of Scools nearby
countplot(dataframe=df, x_val='Number of schools nearby', figsize=(8,5), plot_title='Number of Schools Nearby')
sns.displot(df['Price'])
plt.title('Price Distribution')
plt.show()
```

Code

```
# Visualize correlation matrix using a heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
# plt.title('Correlation Matrix')
plt.suptitle('Correlation Matrix', y=1.02)
plt.tight_layout()
plt.show()
# Pairplot to visualize relationships between numerical variables
sns.pairplot(df[['number of bedrooms', 'living area', 'lot area', 'number of floors', 'Price']])
plt.suptitle('Pairplot of Numerical Variables', y=1.02)  # Adjust the y-coordinate to prevent overlapping
plt.tight_layout()  # Automatically adjusts subplot parameters for better layout
plt.show()
# Boxplot to identify patterns and outliers
plt.figure(figsize=(15, 8))
sns.boxplot(x='number of bedrooms', y='Price', data=df)
plt.title('Boxplot of Price by Number of Bedrooms')
plt.show()
```

Code

```
from datetime import datetime
# from math import radians, sin, cos, sqrt, atan2
# Current year
current_year = datetime.now().year
# Age of Property
df['age_of_property'] = current_year - df['Built Year']
# Total Square Footage
df['total_sqft'] = df['living area'] + df['lot area']
# Renovation Age
# Calculate age of renovation, substituting Built Year if Renovation Year is 0
df['age_of_renovation'] = current_year - df.apply(lambda row: row['Renovation Year'] if row['Renovation Year'] != 0 else row['Built Year'], axis=1)
# Quality of Location
df['quality_of_location'] = df['grade of the house'] + df['waterfront present']
# Basement Ratio
df['basement_ratio'] = df['Area of the basement'] / df['Area of the house(excluding basement)']
# Floor Area Ratio
df['floor_area_ratio'] = df['living area'] / df['lot area']
# Display the updated DataFrame
print(df.head())
```

Code

```
correlation_matrix = df.corr()
plt.figure(figsize=(24, 18))  # Adjusted figure size
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.suptitle('Correlation Matrix', y=0.95)  # Adjusted y position of the title
plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
# Convert 'number of bathrooms' and 'number of floors' to integers
df['number of bathrooms'] = df['number of bathrooms'].astype(int)
df['number of floors'] = df['number of floors'].astype(int)
df['floor_area_ratio']=df['floor_area_ratio'].astype(int)
df['basement_ratio']=df['basement_ratio'].astype(int)
df['age_of_renovation']=df['age_of_renovation'].astype(int)
df.info()
```

Code

```
# Calculate the correlation matrix
correlation_matrix = df[['number of bedrooms', 'number of bathrooms', 'living area', 'lot area',
                         'number of floors', 'waterfront present', 'number of views',
                         'condition of the house', 'grade of the house',
                         'Area of the house(excluding basement)', 'Area of the basement',
                         'Built Year', 'Renovation Year', 'Postal Code', 'Lattitude', 'Longitude',
                         'living_area_renov', 'lot_area_renov', 'Number of schools nearby',
                         'Distance from the airport', 'age_of_property', 'total_sqft',
                         'age_of_renovation', 'quality_of_location', 'basement_ratio',
                         'floor_area_ratio', 'Price']].corr()

# Display the correlation matrix
print(correlation_matrix)

# Filter the correlation values with respect to the target variable (Price)
price_correlation = correlation_matrix['Price'].sort_values(ascending=False)

# Display the correlation values
print(price_correlation)
```

Code

```
df = df.drop(columns=['id', 'Date'])

# Print the first few rows to verify the changes
print(df.head())
df.head()
#redunant attribute checking on full dataset column for removing unnecessary columns.
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix for the entire dataset
correlation_matrix = df.corr()

# Set the size of the figure
plt.figure(figsize=(28, 36))

# Create the heatmap with annotations and a colormap
sns.heatmap(correlation_matrix, annot=True, cmap=plt.cm.CMRmap_r)

# Show the plot
plt.show()
```

Code

```
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if corr_matrix.iloc[i, j] > threshold:
#            if corr_matrix.iloc[i, j] > threshold or corr_matrix.iloc[i, j] < -threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
corr_features=correlation(df,0.7)
len(set(corr_features))
corr_features
corr_features.discard('Price')
corr_features
# Drop the correlated features from the DataFrame
df = df.drop(corr_features,axis=1)
# Save the modified DataFrame to a new CSV file
df.to_csv("modified_dataset.csv", index=False)
```

Code

```
# Load the modified CSV file into a DataFrame
modified_df = pd.read_csv("modified_dataset.csv")
# Display the head of the modified DataFrame
modified_df.head()
modified_df = modified_df.drop(columns=['Lattitude', 'Longitude','Postal Code'])
# Save the modified DataFrame to a new CSV file
modified_df.to_csv("modified_dataset.csv", index=False)
modified_df.head()
# Separate features (X) and target variable (y)
X = modified_df.drop('Price', axis=1)
y = modified_df['Price’]
modified_df = modified_df.drop(columns=['Built Year','Renovation Year','basement_ratio','Distance from the airport','Number of schools nearby','floor_area_ratio','condition of the house','Area of the basement'])
# Save the modified DataFrame to a new CSV file
modified_df.to_csv("modified_dataset.csv", index=False)
modified_df.head()
```

Code

```
from sklearn.model_selection import train_test_split
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
from sklearn.preprocessing import StandardScaler
# Initialize the scaler for feature
scaler = StandardScaler()
# Fit the scaler to the training data and transform the training data
X_train_scaled = scaler.fit_transform(X_train)
# Transform the testing data using the scaler fitted on the training data
X_test_scaled = scaler.transform(X_test)
# Initialize the scaler for the target variable
target_scaler = StandardScaler()
# Fit and transform the training target variable
y_train_scaled = target_scaler.fit_transform(y_train.values.reshape(-1, 1)).flatten()
# Transform the testing target variable
y_test_scaled = target_scaler.transform(y_test.values.reshape(-1, 1)).flatten()
```

Code

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Define individual base models
random_forest_model = RandomForestRegressor()
xgboost_model = XGBRegressor()

# Train individual base models
random_forest_model.fit(X_train_scaled, y_train)
xgboost_model.fit(X_train_scaled, y_train)
```

Code

```
# Make predictions using individual base models
y_pred_rf = random_forest_model.predict(X_test_scaled)
y_pred_xgb = xgboost_model.predict(X_test_scaled)
y_pred_rf
y_pred_xgb
# Evaluate Random Forest model
r2_rf = r2_score(y_test, y_pred_rf)
# Evaluate XGBoost model
r2_xgb = r2_score(y_test, y_pred_xgb)
print("Random Forest Model:")
print("R-squared:", r2_rf)
print("\nXGBoost Model:")
print("R-squared:", r2_xgb)
# Combine predictions using averaging
ensemble_predictions = (y_pred_rf + y_pred_xgb) / 2
# Evaluate ensemble model
r2 = r2_score(y_test, ensemble_predictions)
print("R-squared:", r2)
```

Code

```
# !pip install catboost
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


# Define individual base models
# random_forest_model = RandomForestRegressor()
xgboost_model = XGBRegressor()
catboost_model = CatBoostRegressor(verbose=0)

# Train individual base models
random_forest_model.fit(X_train, y_train)
xgboost_model.fit(X_train, y_train)
catboost_model.fit(X_train, y_train)
```

Code

```
# Make predictions using individual base models
y_pred_rf = random_forest_model.predict(X_test)
y_pred_xgb = xgboost_model.predict(X_test)
y_pred_cb = catboost_model.predict(X_test)


r_cb=r2_score(y_test,y_pred_cb)
print("catboost:")
print("R-Squared: ",r_cb)

# Combine predictions using averaging
ensemble_predictions = (y_pred_rf + y_pred_xgb + y_pred_cb) / 3

# Evaluate ensemble model
r2 = r2_score(y_test, ensemble_predictions)

print("Ensemble Model:")
print("R-squared:", r2)
```

Code

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# Define individual base models
random_forest_model = RandomForestRegressor()
xgboost_model = XGBRegressor()
catboost_model = CatBoostRegressor(verbose=0)
# Train individual base models
random_forest_model.fit(X_train_scaled, y_train)
xgboost_model.fit(X_train_scaled, y_train)
catboost_model.fit(X_train_scaled, y_train)
# Make predictions using individual base models
y_pred_rf = random_forest_model.predict(X_test_scaled)
y_pred_xgb = xgboost_model.predict(X_test_scaled)
y_pred_cb = catboost_model.predict(X_test_scaled)
r2_xg=r2_score(y_test,y_pred_xgb)
print("xgboost Model:")
print("R-squared:", r2_xg)
```

Code

```
# Combine predictions using averaging
ensemble_predictions = (y_pred_rf + y_pred_xgb + y_pred_cb) / 3
# Evaluate ensemble model
r2 = r2_score(y_test, ensemble_predictions)
print("Ensemble Model:")
print("R-squared:", r2)
# Define a threshold (e.g., mean of the target variable)
threshold = y_train.mean()
# Classify predictions
predicted_classes = (ensemble_predictions > threshold).astype(int)
actual_classes = (y_test > threshold).astype(int)
# Calculate True Positives, False Positives, False Negatives
true_positives = ((predicted_classes == 1) & (actual_classes == 1)).sum()
false_positives = ((predicted_classes == 1) & (actual_classes == 0)).sum()
false_negatives = ((predicted_classes == 0) & (actual_classes == 1)).sum()
# Calculate precision and recall
precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
print("Precision:", precision)
print("Recall:", recall)
```

Code

```
import pickle
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Define individual base models
models = {
    'Random Forest': RandomForestRegressor(),
    'XGBoost': XGBRegressor(),
    'CatBoost': CatBoostRegressor(verbose=0)
}

# Train individual base models and compute R-squared scores
r2_scores = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
print("R-squared:", r2_scores[best_model_name])
```

Code

```
print("R-squared:", r2_scores[best_model_name])
r2_scores[model_name] = r2_score(y_test, y_pred)

# Evaluate ensemble model and compute its R-squared score
y_pred_rf = random_forest_model.predict(X_test)
y_pred_xgb = xgboost_model.predict(X_test)
y_pred_cb = catboost_model.predict(X_test)
ensemble_predictions = (y_pred_rf + y_pred_xgb + y_pred_cb) / 3
ensemble_r2 = r2_score(y_test, ensemble_predictions)
r2_scores['Ensemble'] = ensemble_r2

# Select the model with the highest R-squared score
best_model_name = max(r2_scores, key=r2_scores.get)

# If the best model is the ensemble, we need to define it separately
if best_model_name == 'Ensemble':
    best_model = None  # Or you can define your ensemble model here
else:
    best_model = models[best_model_name]
```

Code

```
# The Ensemble model is the bagging of random_forest+xGradientBoosting+Catboost Algorithm
print("Best Model:", best_model_name)

# Save the selected model as a pickle file
with open('best_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)
```

Output

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [2]: df=pd.read_csv('House Price India.csv')
```

```
In [3]: df.head(5)
```

Out[3]:

	id	Date	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	condition of the house	...	Built Year	Renovation Year	Postal Code	Lattitude	Longitude	living_
0	6762810145	42491	5	2.50	3650	9050	2.0	0	4	5	...	1921	0	122003	52.8645	-114.557	
1	6762810635	42491	4	2.50	2920	4000	1.5	0	0	5	...	1909	0	122004	52.8878	-114.470	
2	6762810998	42491	5	2.75	2910	9480	1.5	0	0	3	...	1939	0	122004	52.8852	-114.468	
3	6762812605	42491	4	2.50	3310	42998	2.0	0	0	3	...	2001	0	122005	52.9532	-114.321	
4	6762812919	42491	3	2.00	2710	4500	1.5	0	0	4	...	1929	0	122006	52.9047	-114.485	

5 rows × 23 columns

In [4]:
```
pd.isnull(df).head()
```

Out[4]:

number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	condition of the house	...	Built Year	Renovation Year	Postal Code	Lattitude	Longitude	living_area_renov	lot_area_renov	Number of schools nearby
False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False

In [5]:
```
df.isna().sum()
```

Out[5]:
```
id                                         0
Date                                       0
number of bedrooms                         0
number of bathrooms                        0
living area                                0
lot area                                   0
number of floors                           0
waterfront present                         0
number of views                            0
condition of the house                     0
grade of the house                         0
Area of the house(excluding basement)      0
Area of the basement                       0
Built Year                                 0
Renovation Year                            0
Postal Code                                0
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14620 entries, 0 to 14619
Data columns (total 23 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   id                                     14620 non-null  int64  
 1   Date                                   14620 non-null  int64  
 2   number of bedrooms                     14620 non-null  int64  
 3   number of bathrooms                    14620 non-null  float64
 4   living area                            14620 non-null  int64  
 5   lot area                               14620 non-null  int64  
 6   number of floors                       14620 non-null  float64
 7   waterfront present                     14620 non-null  int64  
 8   number of views                        14620 non-null  int64  
 9   condition of the house                 14620 non-null  int64  
 10  grade of the house                     14620 non-null  int64  
 11  Area of the house(excluding basement)  14620 non-null  int64  
 12  Area of the basement                   14620 non-null  int64  
 13  Built Year                             14620 non-null  int64  
 14  Renovation Year                        14620 non-null  int64  
 15  Postal Code                            14620 non-null  int64  
 16  Lattitude                              14620 non-null  float64
 17  Longitude                              14620 non-null  float64
 18  living_area_renov                      14620 non-null  int64  
 19  lot_area_renov                         14620 non-null  int64  
 20  Number of schools nearby               14620 non-null  int64  
 21  Distance from the airport              14620 non-null  int64  
 22  Price                                  14620 non-null  int64  
dtypes: float64(4), int64(19)
memory usage: 2.6 MB
```

In [7]:
```
# returns a dataframe with column names of the dataset

pd.DataFrame(list(df.columns), columns=['Column Name'])
```

Out[7]:

	Column Name
0	id
1	Date
2	number of bedrooms
3	number of bathrooms
4	living area
5	lot area
6	number of floors
7	waterfront present
8	number of views
9	condition of the house
10	grade of the house
11	Area of the house(excluding basement)
12	Area of the basement
13	Built Year
14	Renovation Year
15	Postal Code
16	Lattitude
17	Longitude
18	living_area_renov
19	lot_area_renov

19	lot_area_renov
20	Number of schools nearby
21	Distance from the airport
22	Price

In [8]:
```
from matplotlib import pyplot as plt
import seaborn as sns
```

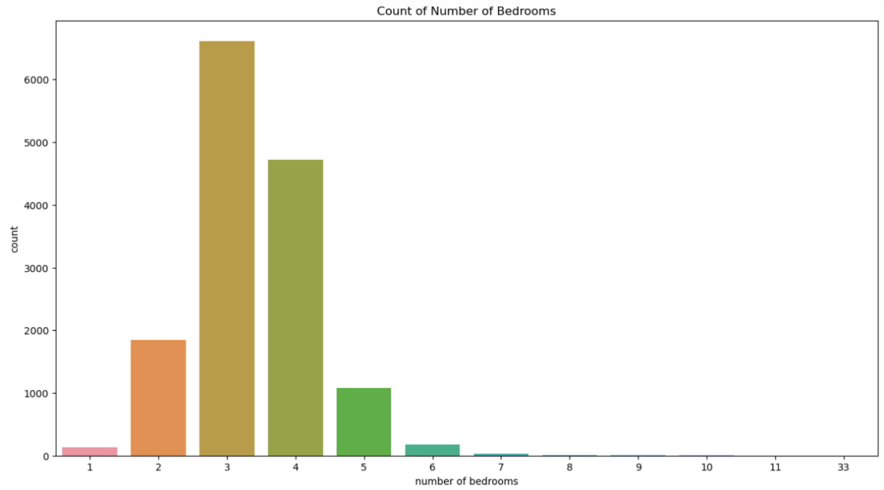
In [9]:
```
# used to return a countplot with null title and figsize to be (15, 8) as default

def countplot(dataframe, x_val, plot_title='', figsize=(15,8)):
    plt.figure(figsize=figsize)
    plt.title(plot_title)
    sns.countplot(data=df, x=x_val)
    plt.show()
```

In [10]:
```
# used to return a barplot

def barplot(dataframe, x_val, y_val):
    sns.barplot(data=dataframe, x=x_val, y=y_val)
    plt.title(x_val.title() + ' vs ' + y_val.title())
    plt.show()
```

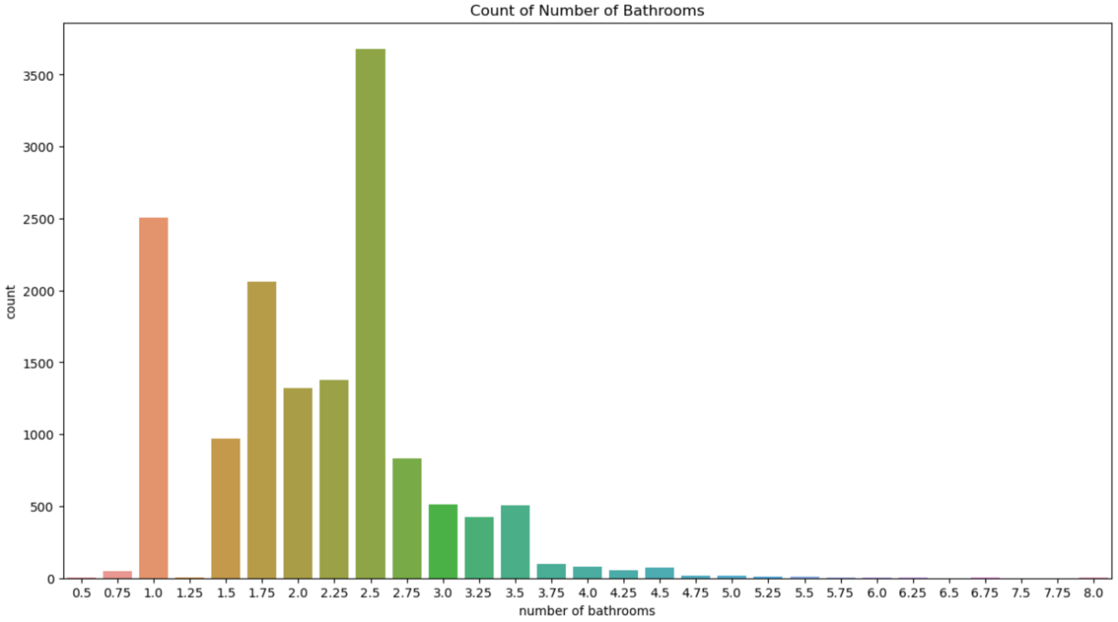
In [11]:
```
df.shape
```

Out[11]:
```
(14620, 23)
```

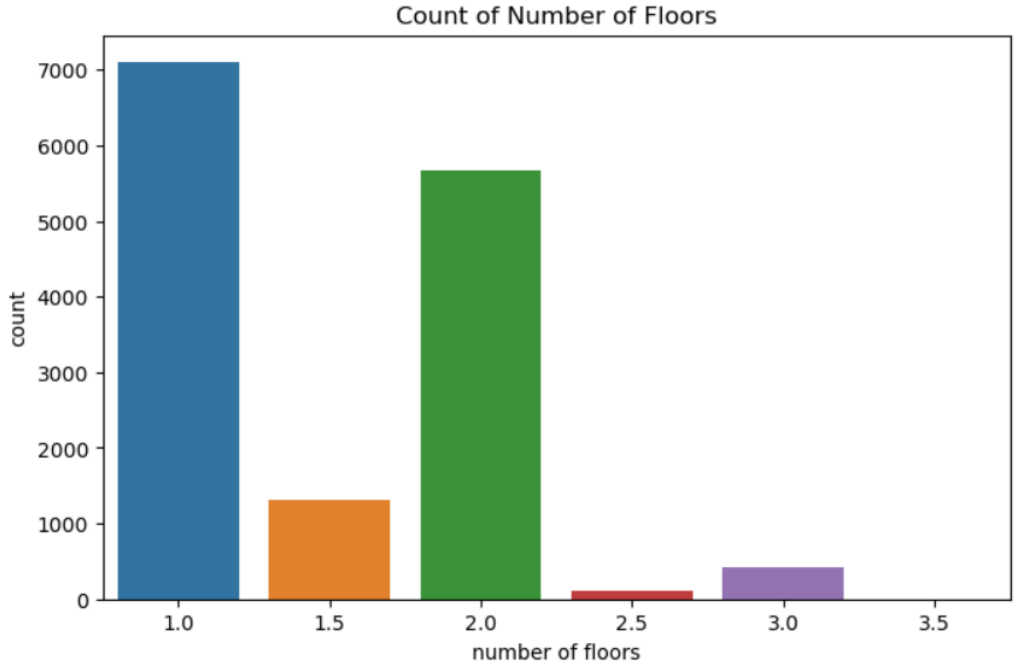
In [12]:
```
#Univariate Analysis
# returns the countplot of bedrooms

countplot(dataframe=df, x_val='number of bedrooms', plot_title='Count of Number of Bedrooms')
```



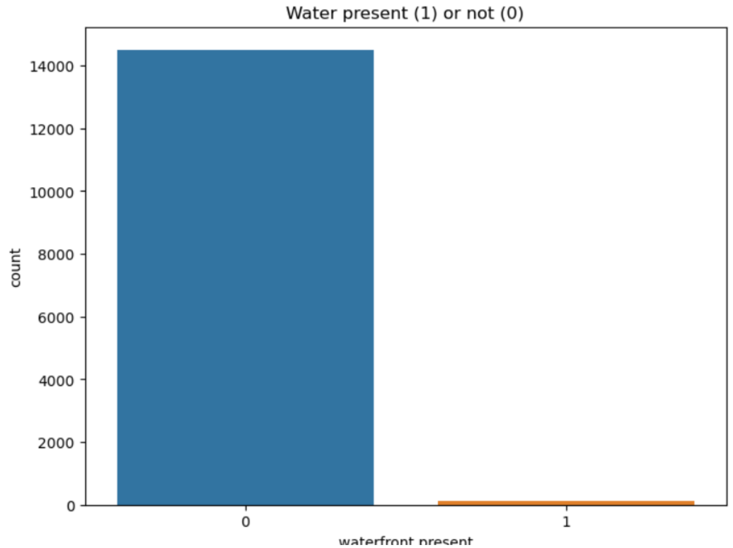
In [13]:
```
# returns the countplot of bathrooms

countplot(dataframe=df, x_val='number of bathrooms', plot_title='Count of Number of Bathrooms')
```



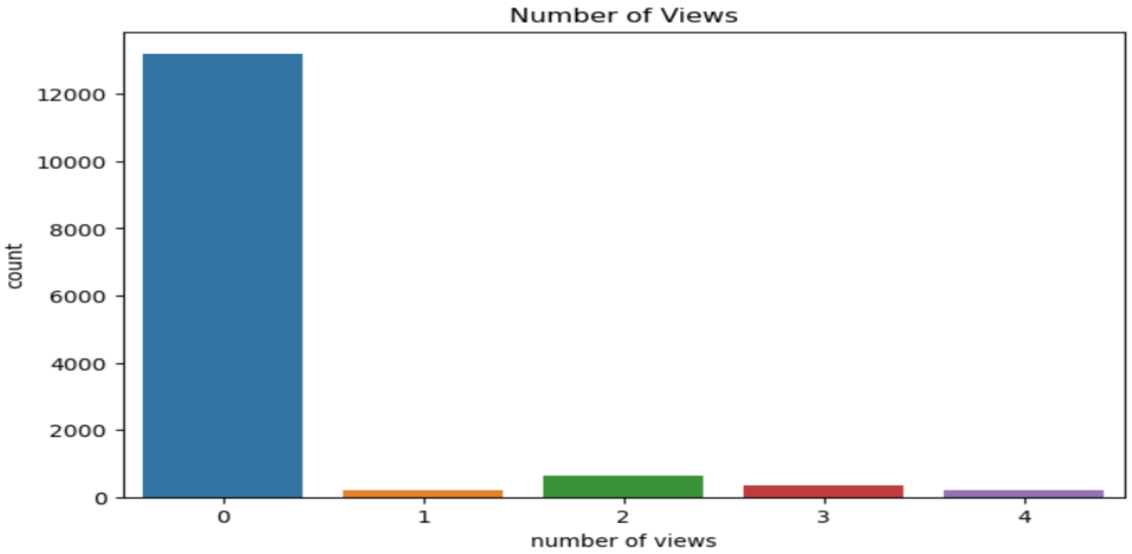
In [14]:
```python
# returns the countplot of floors

countplot(dataframe=df, x_val='number of floors', figsize=(8,5), plot_title='Count of Number of Floors')
```



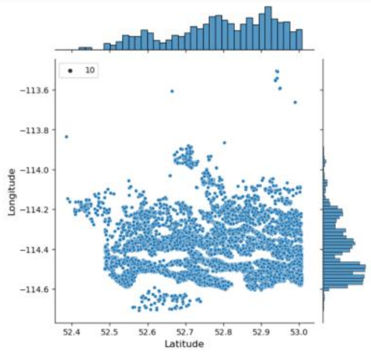
In [15]:
```python
# returns the countplot of number water present or not

countplot(dataframe=df, x_val='waterfront present', figsize=(8,6), plot_title='Water present (1) or not (0)')
```



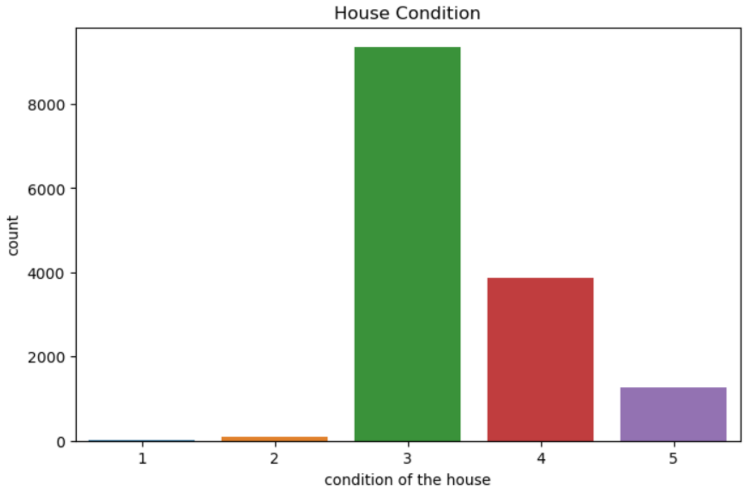
In [16]:
```
# returns the countplot of views

countplot(dataframe=df, x_val='number of views', figsize=(8,5), plot_title='Number of Views')
```



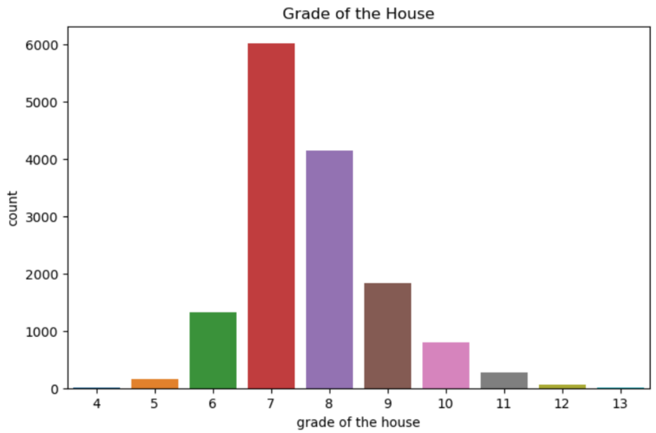
In [17]:
```
plt.figure(figsize=(10,10))
sns.jointplot(x=df.Lattitude.values, y=df.Longitude.values,size=10)
plt.ylabel('Longitude',fontsize=12)
plt.xlabel('Latitude',fontsize=12)
plt.show()
sns.despine
```



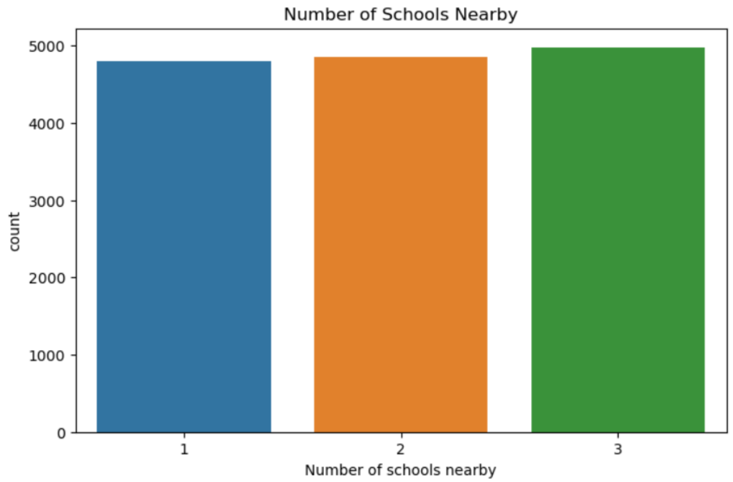
In [18]:
```
# returns the countplot of House Condition

countplot(dataframe=df, x_val='condition of the house', figsize=(8,5), plot_title='House Condition')
```



```
In [19]: # returns the countplot of each grade of house

         countplot(dataframe=df, x_val='grade of the house', figsize=(8,5), plot_title='Grade of the House')
```

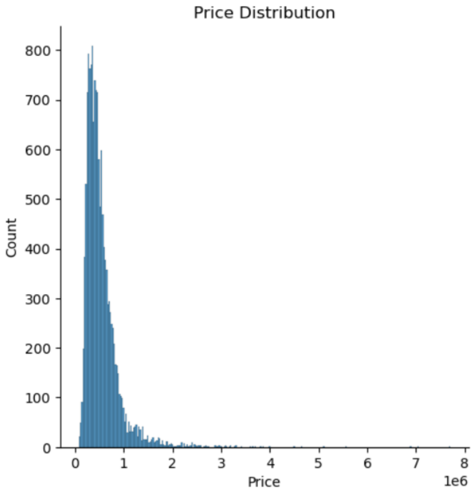



```
In [20]: # returns the countplot of Scools nearby

         countplot(dataframe=df, x_val='Number of schools nearby', figsize=(8,5), plot_title='Number of Schools Nearby')
```

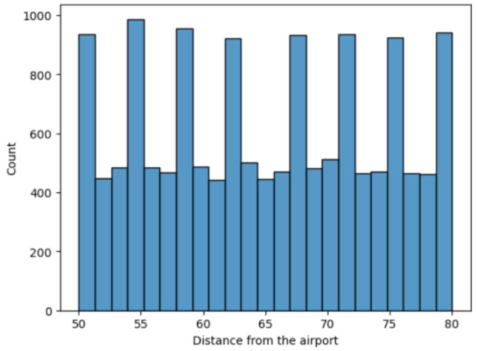


In [21]:
```
sns.displot(df['Price'])
plt.title('Price Distribution')
plt.show()
```

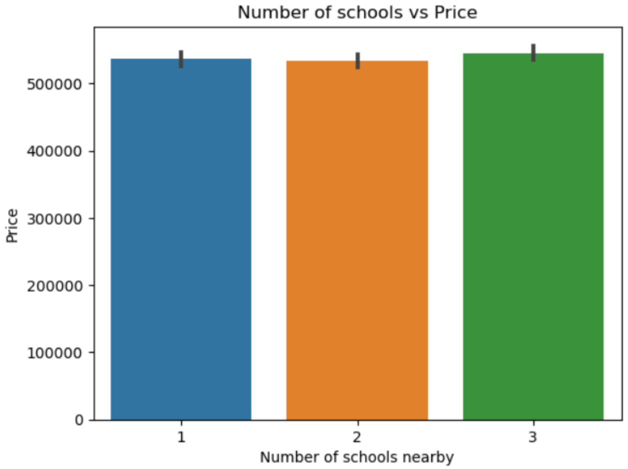
```
D:\anaconda\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



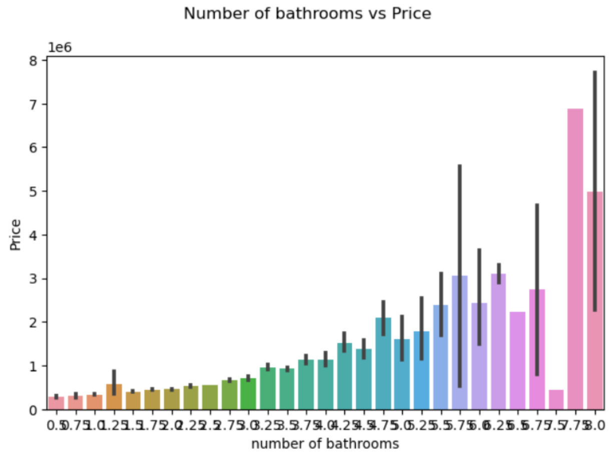
In [22]:
```
# returns the histplot of distance from airport

sns.histplot(data=df, x='Distance from the airport')
plt.show()
```



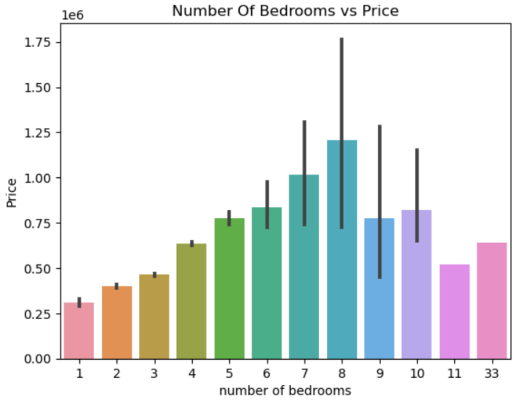
```
In [23]: #Bi - variate Analysis
sns.barplot(data=df, x='Number of schools nearby', y='Price')
plt.title('Number of schools vs Price')
plt.show()
```



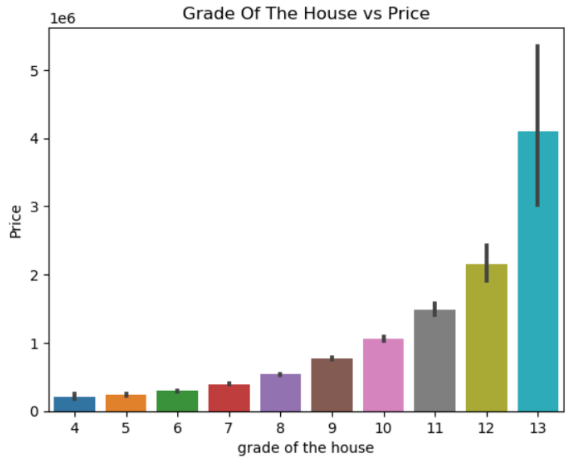

```
In [24]: sns.barplot(data=df, x='number of bathrooms', y='Price')
plt.suptitle('Number of bathrooms vs Price')
plt.tight_layout()
plt.show()
# barplot(dataframe=df, x_val='number of bathrooms', y_val='Price')
```



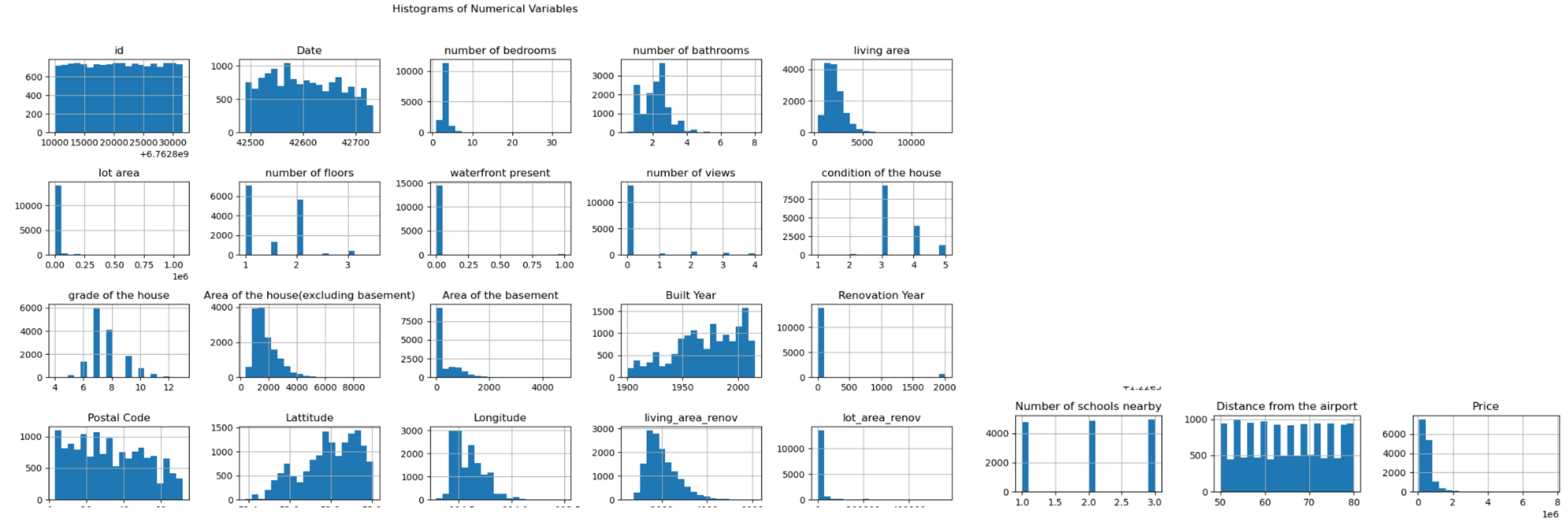
In [25]:
```
barplot(dataframe=df, x_val='number of bedrooms', y_val='Price')
```



In [26]:
```
barplot(dataframe=df, x_val='grade of the house', y_val='Price')
```

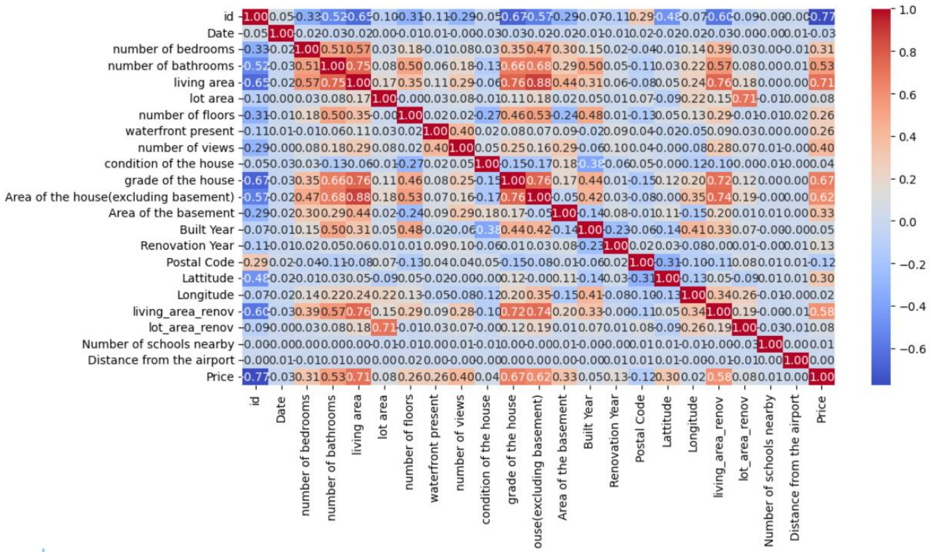


In [27]:
```
# Visualize the distribution of each numerical variable
df.hist(bins=20, figsize=(15, 10))
plt.suptitle('Histograms of Numerical Variables', y=1.02)  # Adjust the y-coordinate to prevent overlapping
plt.tight_layout()  # Automatically adjusts subplot parameters for better layout
plt.show()
```



In [28]:

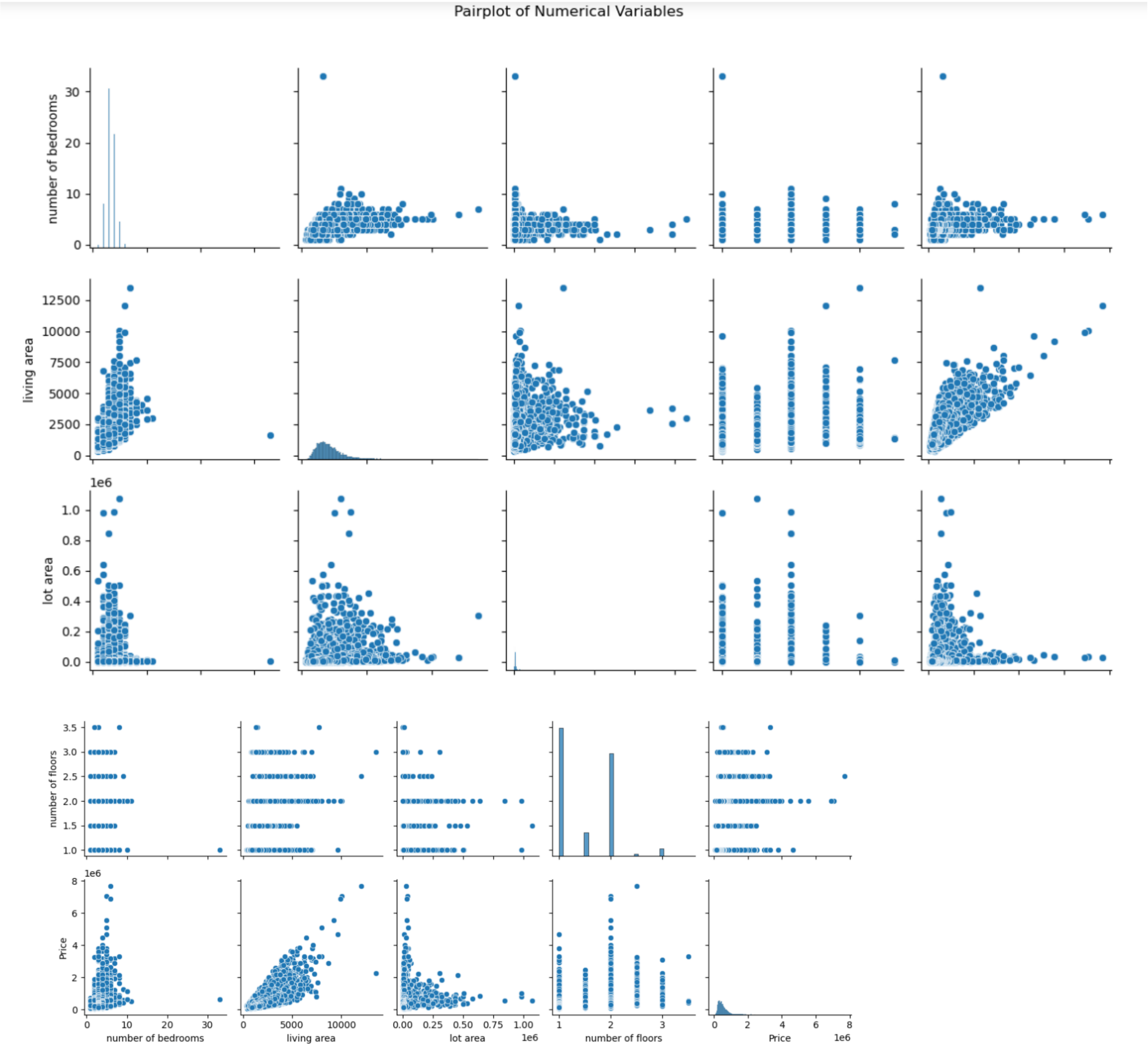
```python
# Visualize correlation matrix using a heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
# plt.title('Correlation Matrix')
plt.suptitle('Correlation Matrix', y=1.02)
plt.tight_layout()
plt.show()
```



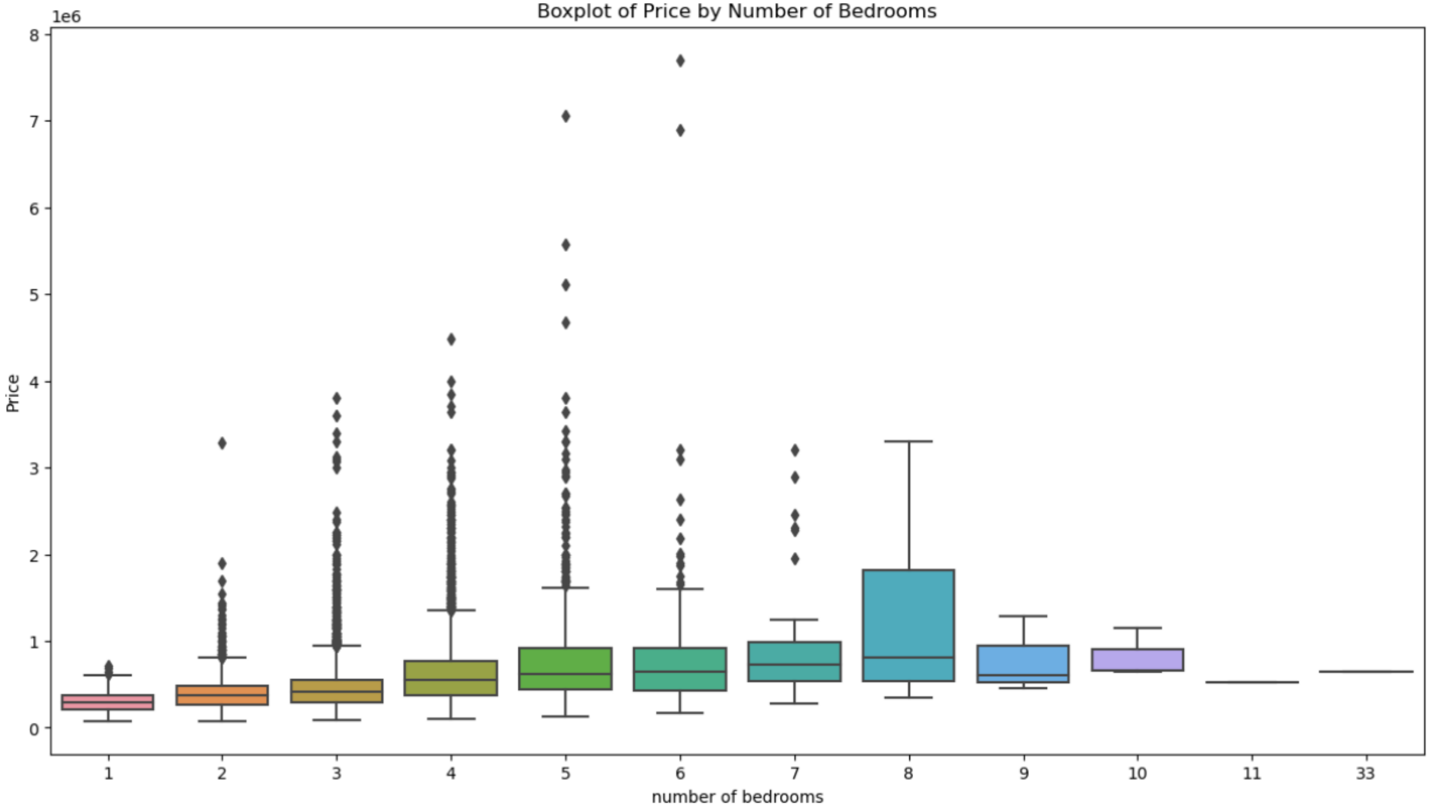
In [29]:

```python
# Pairplot to visualize relationships between numerical variables
sns.pairplot(df[['number of bedrooms', 'living area', 'lot area', 'number of floors', 'Price']])
plt.suptitle('Pairplot of Numerical Variables', y=1.02)  # Adjust the y-coordinate to prevent overlapping
plt.tight_layout()  # Automatically adjusts subplot parameters for better layout
plt.show()
```

```
D:\anaconda\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
C:\Users\RAGHUNATH SINGH\AppData\Local\Temp\ipykernel_19232\3003877325.py:4: UserWarning: The figure layout has changed to tigh
t
  plt.tight_layout()  # Automatically adjusts subplot parameters for better layout
```


Pairplot of Numerical Variables
number of bedrooms
living area
lot area
number of floors
Price
1e6

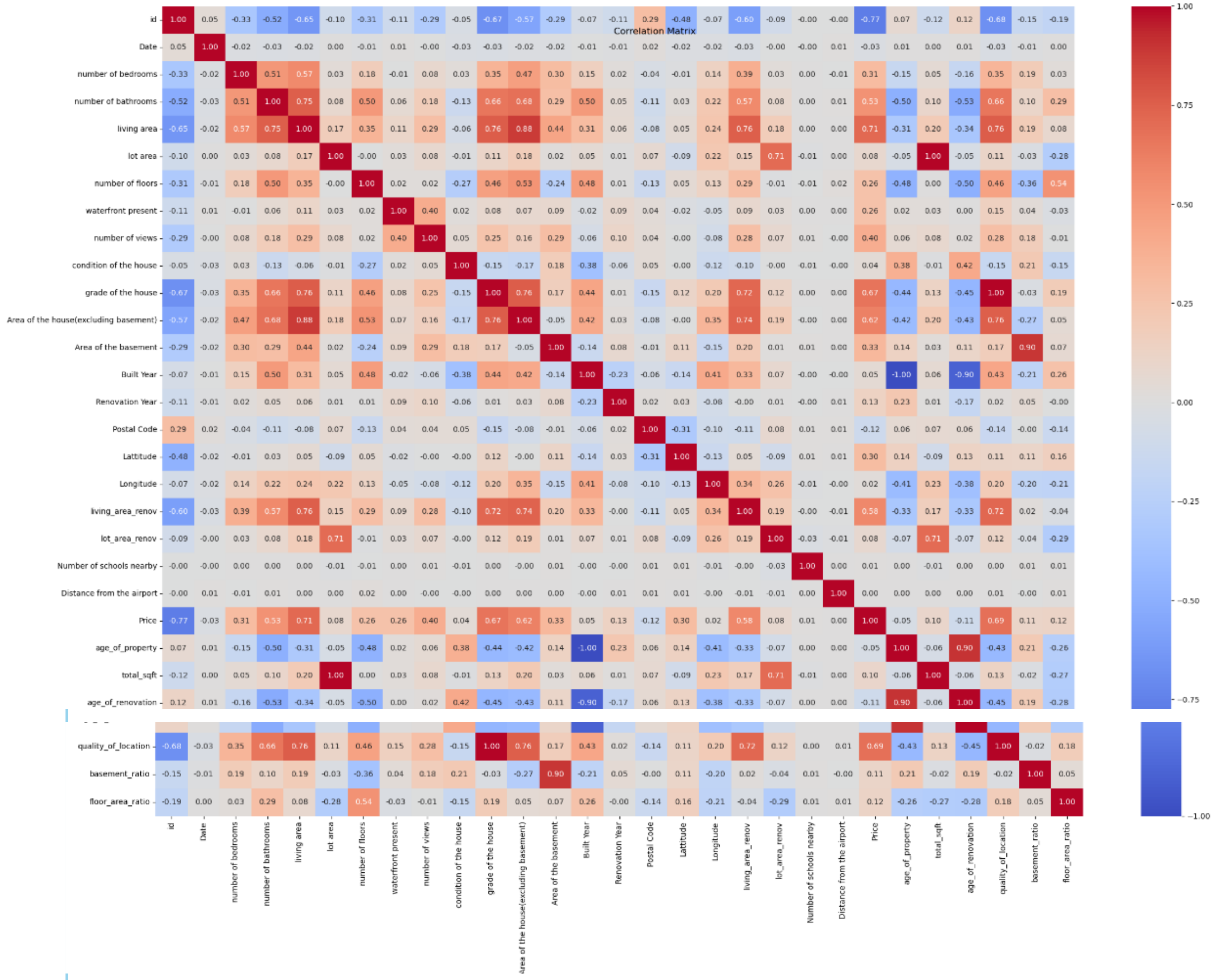
In [30]:
```
# Boxplot to identify patterns and outliers
plt.figure(figsize=(15, 8))
sns.boxplot(x='number of bedrooms', y='Price', data=df)
plt.title('Boxplot of Price by Number of Bedrooms')
plt.show()
```



In [31]:
```python
from datetime import datetime
# from math import radians, sin, cos, sqrt, atan2
```

In [32]:
```python
# Feature engineering

# Current year
current_year = datetime.now().year

# Age of Property
df['age_of_property'] = current_year - df['Built Year']

# Total Square Footage
df['total_sqft'] = df['living area'] + df['lot area']

# Renovation Age
# Calculate age of renovation, substituting Built Year if Renovation Year is 0
df['age_of_renovation'] = current_year - df.apply(lambda row: row['Renovation Year'] if row['Renovation Year'] != 0 else row['Bui

# Quality of Location
df['quality_of_location'] = df['grade of the house'] + df['waterfront present']

# Basement Ratio
df['basement_ratio'] = df['Area of the basement'] / df['Area of the house(excluding basement)']

# Floor Area Ratio
df['floor_area_ratio'] = df['living area'] / df['lot area']

# Display the updated DataFrame
print(df.head())
```

In [33]:
```python
correlation_matrix = df.corr()

plt.figure(figsize=(24, 18))  # Adjusted figure size
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.suptitle('Correlation Matrix', y=0.95)  # Adjusted y position of the title
plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```



In [37]:

```
# Convert 'number of bathrooms' and 'number of floors' to integers
df['number of bathrooms'] = df['number of bathrooms'].astype(int)
df['number of floors'] = df['number of floors'].astype(int)
df['floor_area_ratio']=df['floor_area_ratio'].astype(int)
df['basement_ratio']=df['basement_ratio'].astype(int)
df['age_of_renovation']=df['age_of_renovation'].astype(int)
```

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14620 entries, 0 to 14619
Data columns (total 29 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   id                                     14620 non-null  int64  
 1   Date                                   14620 non-null  int64  
 2   number of bedrooms                     14620 non-null  int64  
 3   number of bathrooms                    14620 non-null  int32  
 4   living area                            14620 non-null  int64  
 5   lot area                               14620 non-null  int64  
 6   number of floors                       14620 non-null  int32  
 7   waterfront present                     14620 non-null  int64  
 8   number of views                        14620 non-null  int64  
 9   condition of the house                 14620 non-null  int64  
 10  grade of the house                     14620 non-null  int64  
 11  Area of the house(excluding basement)  14620 non-null  int64  
 12  Area of the basement                   14620 non-null  int64  
 13  Built Year                             14620 non-null  int64  
 14  Renovation Year                        14620 non-null  int64  
 15  Postal Code                            14620 non-null  int64  
 16  Lattitude                              14620 non-null  float64
 17  Longitude                              14620 non-null  float64
 18  living_area_renov                      14620 non-null  int64  
 19  lot_area_renov                         14620 non-null  int64  
 20  Number of schools nearby               14620 non-null  int64  
 21  Distance from the airport              14620 non-null  int64  
 22  Price                                  14620 non-null  int64  
 23  age_of_property                        14620 non-null  int64  
 24  total_sqft                             14620 non-null  int64  
 25  age_of_renovation                      14620 non-null  int32  
 26  quality_of_location                    14620 non-null  int64  
 27  basement_ratio                         14620 non-null  int32  
 28  floor_area_ratio                       14620 non-null  int32  
dtypes: float64(2), int32(5), int64(22)
memory usage: 3.0 MB
```

[39]:
```
# Calculate the correlation matrix
correlation_matrix = df[['number of bedrooms', 'number of bathrooms', 'living area', 'lot area',
                         'number of floors', 'waterfront present', 'number of views',
                         'condition of the house', 'grade of the house',
                         'Area of the house(excluding basement)', 'Area of the basement',
                         'Built Year', 'Renovation Year', 'Postal Code', 'Lattitude', 'Longitude',
                         'living_area_renov', 'lot_area_renov', 'Number of schools nearby',
                         'Distance from the airport', 'age_of_property', 'total_sqft',
                         'age_of_renovation', 'quality_of_location', 'basement_ratio',
                         'floor_area_ratio', 'Price']].corr()

# Display the correlation matrix
print(correlation_matrix)

# Filter the correlation values with respect to the target variable (Price)
price_correlation = correlation_matrix['Price'].sort_values(ascending=False)

# Display the correlation values
print(price_correlation)
```

In [41]:
```
df.describe()
```
Out[41]:

	id	Date	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	condition of the house	...	lot_a
count	1.462000e+04	14620.000000	14620.000000	14620.000000	14620.000000	1.462000e+04	14620.000000	14620.000000	14620.000000	14620.000000	...	146
mean	6.762821e+09	42604.538646	3.379343	1.761286	2098.262996	1.509328e+04	1.453352	0.007661	0.233105	3.430506	...	127
std	6.237575e+03	67.347991	0.938719	0.736921	928.275721	3.791962e+04	0.552787	0.087193	0.766259	0.664151	...	260
min	6.762810e+09	42491.000000	1.000000	0.000000	370.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	...	6
25%	6.762815e+09	42546.000000	3.000000	1.000000	1440.000000	5.010750e+03	1.000000	0.000000	0.000000	3.000000	...	50
50%	6.762821e+09	42600.000000	3.000000	2.000000	1930.000000	7.620000e+03	1.000000	0.000000	0.000000	3.000000	...	76
75%	6.762826e+09	42662.000000	4.000000	2.000000	2570.000000	1.080000e+04	2.000000	0.000000	0.000000	4.000000	...	101
max	6.762832e+09	42734.000000	33.000000	8.000000	13540.000000	1.074218e+06	3.000000	1.000000	4.000000	5.000000	...	5606

8 rows × 29 columns

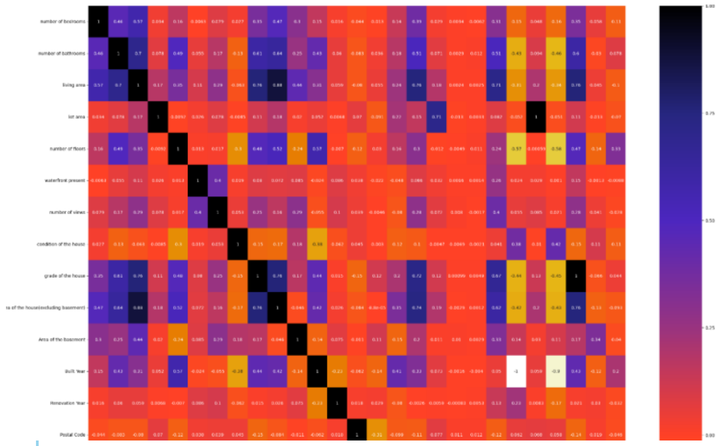
```
# Drop the 'id' and 'Date' columns from the dataset
df = df.drop(columns=['id', 'Date'])

# Print the first few rows to verify the changes
print(df.head())
```

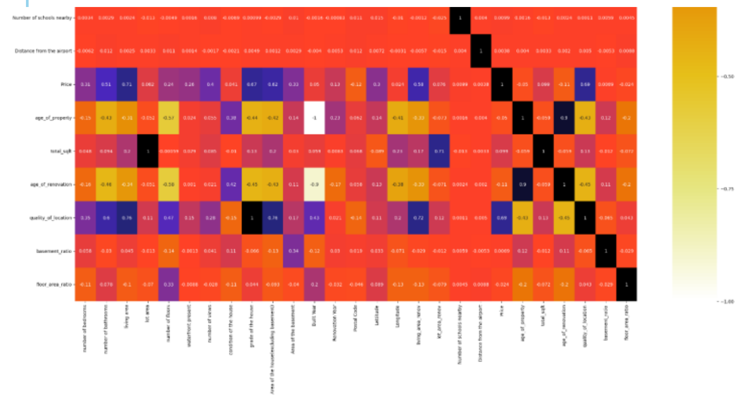
In [43]:
```
df.head()
```
Out[43]:

	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	condition of the house	grade of the house	Area of the house(excluding basement)	...	lot_area_renov	Number of schools nearby	Distance from the airport	Price	age_c
0	5	2	3650	9050	2	0	4	5	10	3370	...	5400	2	58	2380000	
1	4	2	2920	4000	1	0	0	5	8	1910	...	4000	2	51	1400000	
2	5	2	2910	9480	1	0	0	3	8	2910	...	6600	1	53	1200000	
3	4	2	3310	42998	2	0	0	3	9	3310	...	42847	3	76	838000	
4	3	2	2710	4500	1	0	0	4	8	1880	...	4500	1	51	805000	

5 rows × 27 columns

In [44]:
```python
# import seaborn as sns
# plt.figure(figsize=(12,10))
# cor=X_train.corr()
# sns.heatmap(cor,annot=True,cmap=plt.cm.CMRmap_r)
# plt.show()
#redunant attribute checking on full dataset column for removing unnecessary columns.
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix for the entire dataset
correlation_matrix = df.corr()

# Set the size of the figure
plt.figure(figsize=(28, 36))

# Create the heatmap with annotations and a colormap
sns.heatmap(correlation_matrix, annot=True, cmap=plt.cm.CMRmap_r)

# Show the plot
plt.show()
```





In [45]:
```python
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if corr_matrix.iloc[i, j] > threshold:
#               if corr_matrix.iloc[i, j] > threshold or corr_matrix.iloc[i, j] < -threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

In [46]:
```python
corr_features=correlation(df,0.7)
len(set(corr_features))
```

Out[46]: 8

In [47]:
```python
corr_features
```

Out[47]:
```
{'Area of the house(excluding basement)',
 'Price',
 'age_of_renovation',
 'grade of the house',
 'living_area_renov',
 'lot_area_renov',
 'quality_of_location',
 'total_sqft'}
```

In [48]:
```python
corr_features.discard('Price')
corr_features
```

Out[48]:
```
{'Area of the house(excluding basement)',
 'age_of_renovation',
 'grade of the house',
 'living_area_renov',
 'lot_area_renov',
 'quality_of_location',
 'total_sqft'}
```

In [49]:
```python
# Drop the correlated features from the DataFrame
df = df.drop(corr_features,axis=1)

# Save the modified DataFrame to a new CSV file
df.to_csv("modified_dataset.csv", index=False)
```

In [50]:
```python
# Load the modified CSV file into a DataFrame
modified_df = pd.read_csv("modified_dataset.csv")

# Display the head of the modified DataFrame
modified_df.head()
```

Out[50]:

	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	condition of the house	Area of the basement	Built Year	Renovation Year	Postal Code	Lattitude	Longitude	Number of schools nearby	Distance from the airport
0	5	2	3650	9050	2	0	4	5	280	1921	0	122003	52.8645	-114.557	2	58
1	4	2	2920	4000	1	0	0	5	1010	1909	0	122004	52.8878	-114.470	2	51
2	5	2	2910	9480	1	0	0	3	0	1939	0	122004	52.8852	-114.468	1	53
3	4	2	3310	42998	2	0	0	3	0	2001	0	122005	52.9532	-114.321	3	76
4	3	2	2710	4500	1	0	0	4	830	1929	0	122006	52.9047	-114.485	1	51

In [51]:
```python
modified_df = modified_df.drop(columns=['Lattitude', 'Longitude','Postal Code'])

# Save the modified DataFrame to a new CSV file
modified_df.to_csv("modified_dataset.csv", index=False)
```

In [52]:
```python
# modified_df = modified_df.drop(columns=['Postal Code'])

# # Save the modified DataFrame to a new CSV file
# modified_df.to_csv("modified_dataset.csv", index=False)
modified_df.head()
```

Out[52]:

	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	condition of the house	Area of the basement	Built Year	Renovation Year	Number of schools nearby	Distance from the airport	Price	age_of_property	ba
0	5	2	3650	9050	2	0	4	5	280	1921	0	2	58	2380000	103	
1	4	2	2920	4000	1	0	0	5	1010	1909	0	2	51	1400000	115	
2	5	2	2910	9480	1	0	0	3	0	1939	0	1	53	1200000	85	
3	4	2	3310	42998	2	0	0	3	0	2001	0	3	76	838000	23	
4	3	2	2710	4500	1	0	0	4	830	1929	0	1	51	805000	95	

In [53]:
```python
# Separate features (X) and target variable (y)
X = modified_df.drop('Price', axis=1)
y = modified_df['Price']
```

In [56]:
```python
modified_df = modified_df.drop(columns=['Built Year','Renovation Year','basement_ratio','Distance from the airport','Number of s

# Save the modified DataFrame to a new CSV file
modified_df.to_csv("modified_dataset.csv", index=False)
```

In [57]:
```python
# modified_df = modified_df.drop(columns=['Renovation Year'])

# # Save the modified DataFrame to a new CSV file
# modified_df.to_csv("modified_dataset.csv", index=False)
modified_df.head()
```

Out[57]:

	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	Price	age_of_property
0	5	2	3650	9050	2	0	4	2380000	103
1	4	2	2920	4000	1	0	0	1400000	115
2	5	2	2910	9480	1	0	0	1200000	85
3	4	2	3310	42998	2	0	0	838000	23
4	3	2	2710	4500	1	0	0	805000	95

In [58]:
```python
from sklearn.model_selection import train_test_split

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [59]:
```python
from sklearn.preprocessing import StandardScaler

# Initialize the scaler for features
scaler = StandardScaler()

# Fit the scaler to the training data and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data using the scaler fitted on the training data
X_test_scaled = scaler.transform(X_test)

# Initialize the scaler for the target variable
target_scaler = StandardScaler()

# Fit and transform the training target variable
y_train_scaled = target_scaler.fit_transform(y_train.values.reshape(-1, 1)).flatten()

# Transform the testing target variable
y_test_scaled = target_scaler.transform(y_test.values.reshape(-1, 1)).flatten()
```

In [63]:
```python
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

In [64]:
```python
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Define individual base models
random_forest_model = RandomForestRegressor()
xgboost_model = XGBRegressor()

# Train individual base models
random_forest_model.fit(X_train_scaled, y_train)
xgboost_model.fit(X_train_scaled, y_train)

# Make predictions using individual base models
y_pred_rf = random_forest_model.predict(X_test_scaled)
y_pred_xgb = xgboost_model.predict(X_test_scaled)

y_pred_rf
y_pred_xgb
```

```
Out[64]: array([ 389763.22,  396218.1 ,  404091.56, ..., 1213856.1 ,  374069.22,
                 610154.2 ], dtype=float32)
```

```
In [65]: # Evaluate Random Forest model
r2_rf = r2_score(y_test, y_pred_rf)

# Evaluate XGBoost model
r2_xgb = r2_score(y_test, y_pred_xgb)

print("Random Forest Model:")
print("R-squared:", r2_rf)

print("\nXGBoost Model:")
print("R-squared:", r2_xgb)
```

```
Random Forest Model:
R-squared: 0.6635819536414276

XGBoost Model:
R-squared: 0.6487248781534409
```

```
In [66]: # Combine predictions using averaging
ensemble_predictions = (y_pred_rf + y_pred_xgb) / 2

# Evaluate ensemble model
r2 = r2_score(y_test, ensemble_predictions)

print("R-squared:", r2)
```

```
R-squared: 0.673007199618424
```

```
In [67]: # !pip install catboost
```

```
In [73]: from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


# Define individual base models
# random_forest_model = RandomForestRegressor()
xgboost_model = XGBRegressor()
catboost_model = CatBoostRegressor(verbose=0)

# Train individual base models
random_forest_model.fit(X_train, y_train)
xgboost_model.fit(X_train, y_train)
catboost_model.fit(X_train, y_train)

# Make predictions using individual base models
y_pred_rf = random_forest_model.predict(X_test)
y_pred_xgb = xgboost_model.predict(X_test)
y_pred_cb = catboost_model.predict(X_test)


r_cb=r2_score(y_test,y_pred_cb)
print("catboost:")
print("R-Squared: ",r_cb)

# Combine predictions using averaging
ensemble_predictions = (y_pred_rf + y_pred_xgb + y_pred_cb) / 3

# Evaluate ensemble model
r2 = r2_score(y_test, ensemble_predictions)

print("Ensemble Model:")
```

```
print("R-squared:", r2)
```

```
catboost:
R-Squared:  0.6681556299940523
Ensemble Model:
R-squared: 0.6785935216543266
```

In [74]:

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score



# Define individual base models
random_forest_model = RandomForestRegressor()
xgboost_model = XGBRegressor()
catboost_model = CatBoostRegressor(verbose=0)

# Train individual base models
random_forest_model.fit(X_train_scaled, y_train)
xgboost_model.fit(X_train_scaled, y_train)
catboost_model.fit(X_train_scaled, y_train)

# Make predictions using individual base models
y_pred_rf = random_forest_model.predict(X_test_scaled)
y_pred_xgb = xgboost_model.predict(X_test_scaled)
y_pred_cb = catboost_model.predict(X_test_scaled)


r2_xg=r2_score(y_test,y_pred_xgb)
print("xgboost Model:")
print("R-squared:", r2_xg)
```

```
# Combine predictions using averaging
ensemble_predictions = (y_pred_rf + y_pred_xgb + y_pred_cb) / 3

# Evaluate ensemble model
r2 = r2_score(y_test, ensemble_predictions)

print("Ensemble Model:")
print("R-squared:", r2)
```

```
xgboost Model:
R-squared: 0.6487248781534409
Ensemble Model:
R-squared: 0.6783502201587474
```

```
In [75]: # Define a threshold (e.g., mean of the target variable)
threshold = y_train.mean()

# Classify predictions
predicted_classes = (ensemble_predictions > threshold).astype(int)
actual_classes = (y_test > threshold).astype(int)

# Calculate True Positives, False Positives, False Negatives
true_positives = ((predicted_classes == 1) & (actual_classes == 1)).sum()
false_positives = ((predicted_classes == 1) & (actual_classes == 0)).sum()
false_negatives = ((predicted_classes == 0) & (actual_classes == 1)).sum()

# Calculate precision and recall
precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)

print("Precision:", precision)
print("Recall:", recall)
```

```
Precision: 0.7691561590688651
Recall: 0.7242009132420091
```

```
In [83]: import pickle
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Define individual base models
models = {
    'Random Forest': RandomForestRegressor(),
    'XGBoost': XGBRegressor(),
    'CatBoost': CatBoostRegressor(verbose=0)
}

# Train individual base models and compute R-squared scores
r2_scores = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2_scores[model_name] = r2_score(y_test, y_pred)

# Evaluate ensemble model and compute its R-squared score
y_pred_rf = random_forest_model.predict(X_test)
y_pred_xgb = xgboost_model.predict(X_test)
y_pred_cb = catboost_model.predict(X_test)
ensemble_predictions = (y_pred_rf + y_pred_xgb + y_pred_cb) / 3
ensemble_r2 = r2_score(y_test, ensemble_predictions)
r2_scores['Ensemble'] = ensemble_r2

# Select the model with the highest R-squared score
best_model_name = max(r2_scores, key=r2_scores.get)
```

```python
# If the best model is the ensemble, we need to define it separately
if best_model_name == 'Ensemble':
    best_model = None  # Or you can define your ensemble model here
else:
    best_model = models[best_model_name]

# The Ensemble model is the bagging of random_forest+xGradientBoosting+Catboost Algorithm
print("Best Model:", best_model_name)
print("R-squared:", r2_scores[best_model_name])
```

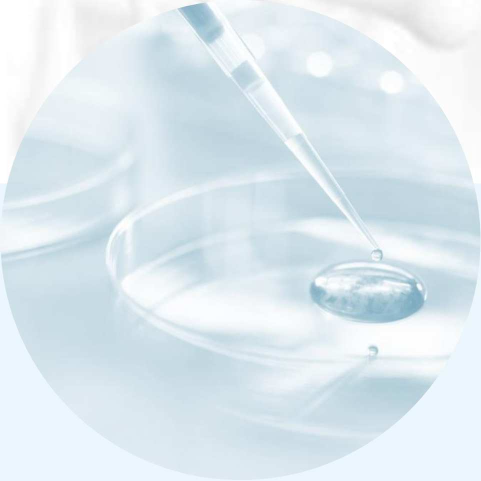
```
Best Model: CatBoost
R-squared: 0.6681556299940523
```

```
D:\anaconda\Lib\site-packages\sklearn\base.py:457: UserWarning: X has feature names, but RandomForestRegressor was fitted witho
ut feature names
  warnings.warn(
```

```python
# Save the selected model as a pickle file
with open('best_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)
```

SUMMARY

The project involved analyzing housing data to predict prices in India. After data preprocessing and exploratory data analysis, redundant features were removed, leaving 8 informative features. Three machine learning models (Random Forest, XGBoost, CatBoost) were trained individually and combined using averaging to create an ensemble model. The best model, a combination of the three, achieved high accuracy. A Flask backend was implemented to serve predictions based on user inputs from a frontend interface.



THANK YOU