House Price Prediction Dataset

House price prediction refers to the process of estimating the future or current values of a residential property based on various factors and historical data.The goal is to forecast the price of a house given its characteristics and the context of the real estate market.

let's install and import the libraries. We'll use the matplotlib.pyplot module for basic plots like line & bar charts. It is often imported with the alias plt. We'll use the seaborn module for more advanced plots. It is commonly imported with the alias sns.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore') # if there are any warning due to version
```

Load dataset

In [2]:
```
df=pd.read_csv('data (1).csv')
df.head()
```

Out[2]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	

Basic Visualization

In [3]:
```
df.shape # checking the n0.of rows and columns
```

Out[3]: (4600, 18)

In [4]:
```
df.head() #Display the first 5rows in the dataframe
```

Out[4]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	

In [5]:
```
df = df.drop_duplicates()
df.head()
```

Out[5]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	

Renaming Columns

In [6]:
```
df.rename(columns={'floors': 'levels'}).head(5)
```

Out[6]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	levels	wat
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	

In [7]:
```
df.rename(columns={'price': 'cost'}).head(10)
```

Out[7]:

	date	cost	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	
5	2014-05-02 00:00:00	490000.0	2.0	1.00	880	6380	1.0	
6	2014-05-02 00:00:00	335000.0	2.0	2.00	1350	2560	1.0	
7	2014-05-02 00:00:00	482000.0	4.0	2.50	2710	35868	2.0	
8	2014-05-02 00:00:00	452500.0	3.0	2.50	2430	88426	1.0	
9	2014-05-02 00:00:00	640000.0	4.0	2.00	1520	6200	1.5	

In [8]:
```
df.tail() #provides last 5 samples
```

Out[8]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	flo
4595	2014-07-09 00:00:00	308166.666667	3.0	1.75	1510	6360	
4596	2014-07-09 00:00:00	534333.333333	3.0	2.50	1460	7573	
4597	2014-07-09 00:00:00	416904.166667	3.0	2.50	3010	7014	
4598	2014-07-10 00:00:00	203400.000000	4.0	2.00	2090	6630	
4599	2014-07-10 00:00:00	220600.000000	3.0	2.50	1490	8102	

In [9]:
```
df.columns
```

Out[9]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```

In [10]:
```
df.sample()
```

Out[10]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	v
2044	2014-06-06 00:00:00	185000.0	3.0	1.0	1840	8100	1.0	

In [11]:
```
df.sample(10)
```

Out[11]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floo
384	2014-05-08 00:00:00	3.219500e+05	2.0	1.25	860	1277	2
2224	2014-06-10 00:00:00	4.030000e+05	2.0	1.00	1100	3598	1
4381	2014-05-12 00:00:00	5.719861e+05	3.0	2.50	3720	11610	2
2937	2014-06-20 00:00:00	6.659000e+05	4.0	2.25	2870	5453	2
4588	2014-07-08 00:00:00	0.000000e+00	4.0	2.25	2890	18226	3
1827	2014-06-03 00:00:00	5.750000e+05	4.0	2.75	3120	7644	2
3492	2014-06-26 00:00:00	2.195000e+05	3.0	1.00	1090	6710	1
1448	2014-05-28 00:00:00	5.550000e+05	3.0	2.50	3160	4270	2
836	2014-05-16 00:00:00	3.300000e+05	3.0	1.50	1170	4950	1
1617	2014-05-30 00:00:00	1.365000e+06	3.0	2.50	2090	6000	1

In [12]:
```
df.isnull().sum() # check for missing values
```

Out[12]:
```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

DATA CLEANING

Handle missing values,outliers,and inconsistencies in the data.This may involve imputation or removal of problematic data points.

In [13]:
```
df.info() # method prints information about the DataFrame.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           4600 non-null   object 
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64  
 5   sqft_lot       4600 non-null   int64  
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64  
 8   view           4600 non-null   int64  
 9   condition      4600 non-null   int64  
 10  sqft_above     4600 non-null   int64  
 11  sqft_basement  4600 non-null   int64  
 12  yr_built       4600 non-null   int64  
 13  yr_renovated   4600 non-null   int64  
 14  street         4600 non-null   object 
 15  city           4600 non-null   object 
 16  statezip       4600 non-null   object 
 17  country        4600 non-null   object 
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

In [14]:
```
df.describe() #calculates summary statistics for all numerical columns in th
```

Out[14]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03	4
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04	
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04	
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02	
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03	
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03	
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04	
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06	

In [15]:
```
df.describe().transpose()
```

Out[15]:

	count	mean	std	min	25%	
price	4600.0	551962.988473	563834.702547	0.0	322875.00	46094
bedrooms	4600.0	3.400870	0.908848	0.0	3.00	
bathrooms	4600.0	2.160815	0.783781	0.0	1.75	
sqft_living	4600.0	2139.346957	963.206916	370.0	1460.00	198
sqft_lot	4600.0	14852.516087	35884.436145	638.0	5000.75	768
floors	4600.0	1.512065	0.538288	1.0	1.00	
waterfront	4600.0	0.007174	0.084404	0.0	0.00	
view	4600.0	0.240652	0.778405	0.0	0.00	
condition	4600.0	3.451739	0.677230	1.0	3.00	
sqft_above	4600.0	1827.265435	862.168977	370.0	1190.00	159
sqft_basement	4600.0	312.081522	464.137228	0.0	0.00	
yr_built	4600.0	1970.786304	29.731848	1900.0	1951.00	197
yr_renovated	4600.0	808.608261	979.414536	0.0	0.00	

In [16]:
```
df["date"]=pd.to_datetime(df["date"])
```

In [17]:
```
df.dtypes
```

Out[17]:
```
date             datetime64[ns]
price                   float64
bedrooms                float64
bathrooms               float64
sqft_living               int64
sqft_lot                  int64
floors                  float64
waterfront                int64
view                      int64
condition                 int64
sqft_above                int64
sqft_basement             int64
yr_built                  int64
yr_renovated              int64
street                   object
city                     object
statezip                 object
country                  object
dtype: object
```

In [18]:
```
df.duplicated().sum()
```

Out[18]:
```
np.int64(0)
```

In [19]:
```
df.groupby("sqft_lot")[["bedrooms","bathrooms","floors","view","condition"]]
```

Out[19]:

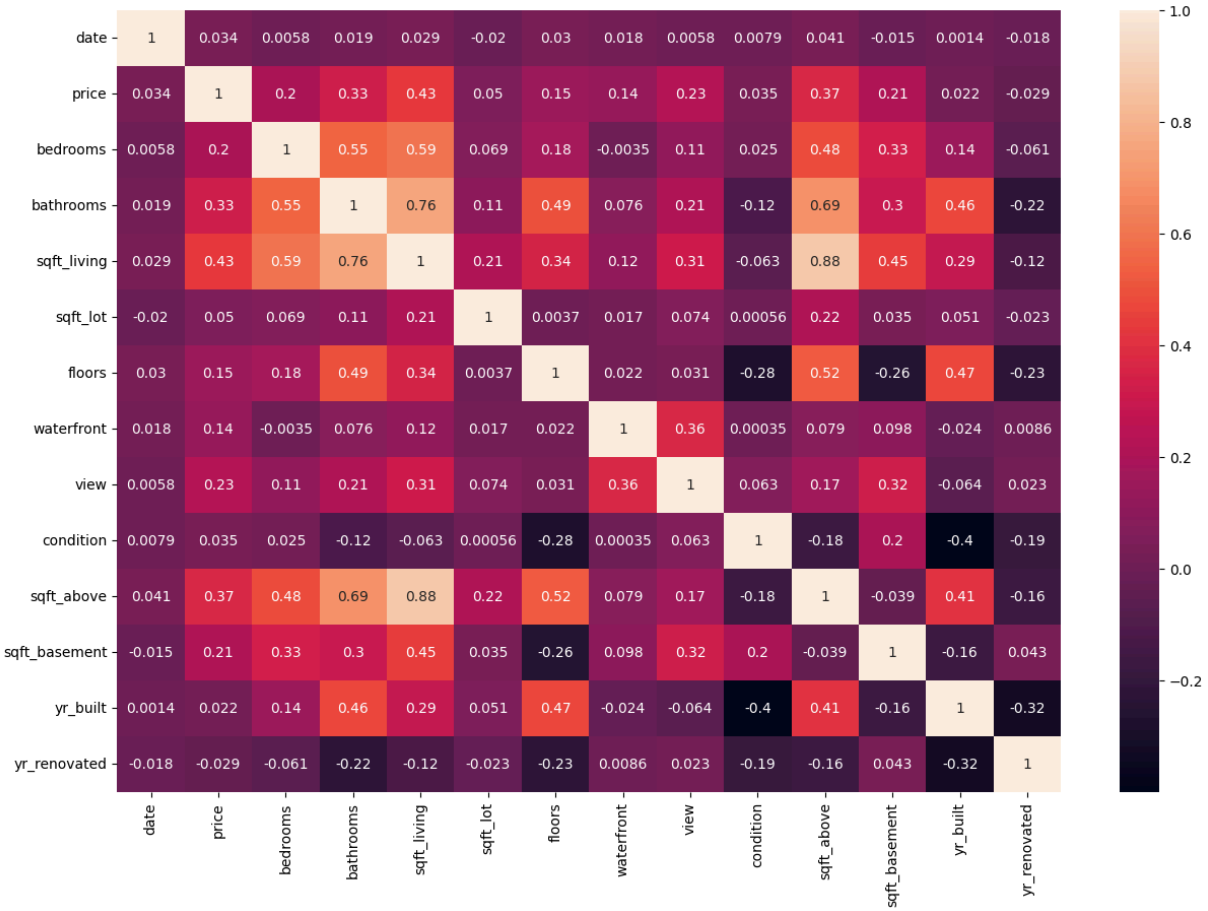
sqft_lot	bedrooms	bathrooms	floors	view	condition
1074218	5.0	3.25	1.5	0	5
641203	2.0	2.00	2.0	0	3
478288	3.0	1.75	1.5	3	4
435600	6.0	5.50	3.5	3	5
423838	2.0	1.00	1.0	2	5
389126	3.0	1.00	1.5	0	4
327135	3.0	2.50	2.0	0	3
307752	7.0	8.00	3.0	4	3
306848	3.0	1.00	1.0	0	3
284011	4.0	4.50	2.0	0	4
280962	2.0	1.75	2.0	2	3
265716	4.0	1.75	1.0	0	4
258746	5.0	3.00	1.5	0	4
251341	3.0	2.00	2.0	0	3
250470	3.0	1.75	1.0	0	4

DataFrame showing the sum of bedrooms, bathrooms, floors, view, and condition for each unique sqft_lot, sorted in descending order by sqft_lot, and displaying the top 15 rows.

Data Visualisation

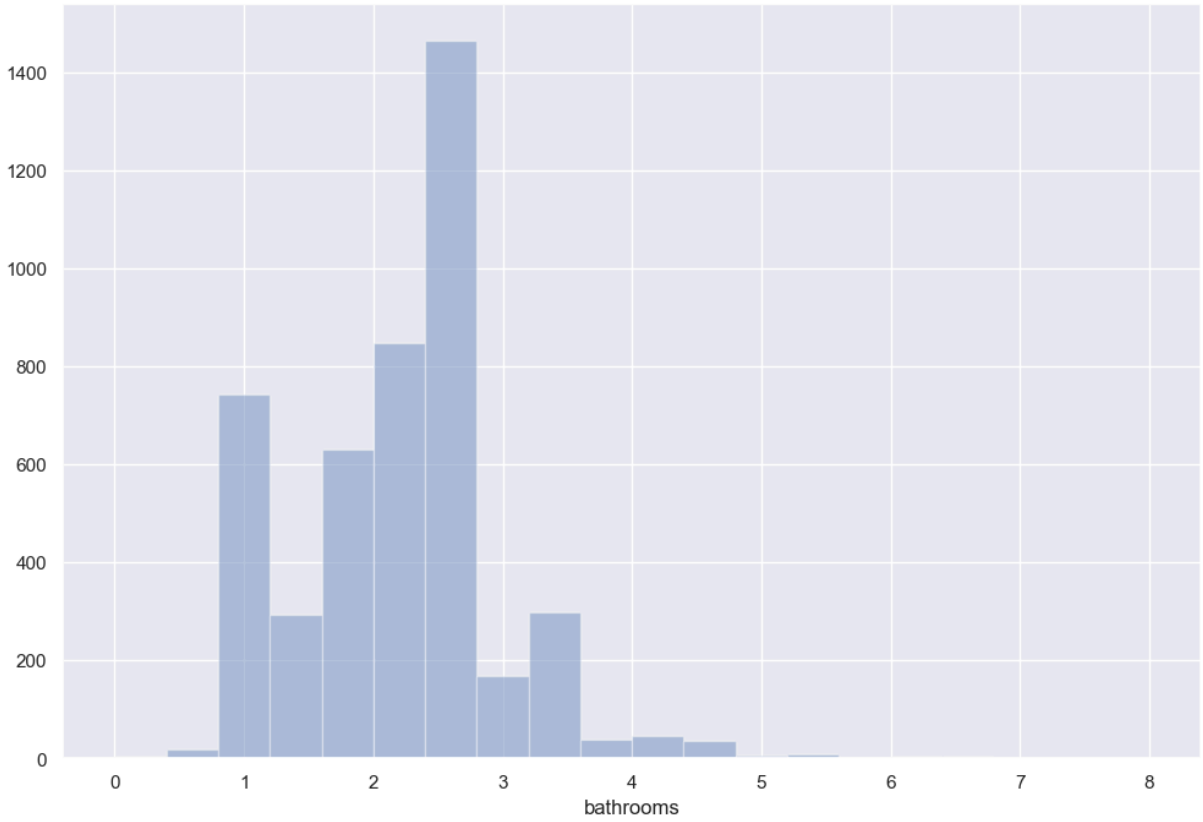
In [20]:
```
num_col=df[df.dtypes[df.dtypes != 'object'].index]
num_col

plt.figure(figsize=(15,10))
sns.heatmap(num_col.corr(),annot=True)
```

Out[20]: <Axes: >



The code generates a heatmap showing the correlation matrix of the numeric columns in the DataFrame df, with correlation coefficients annotated on the heatmap.

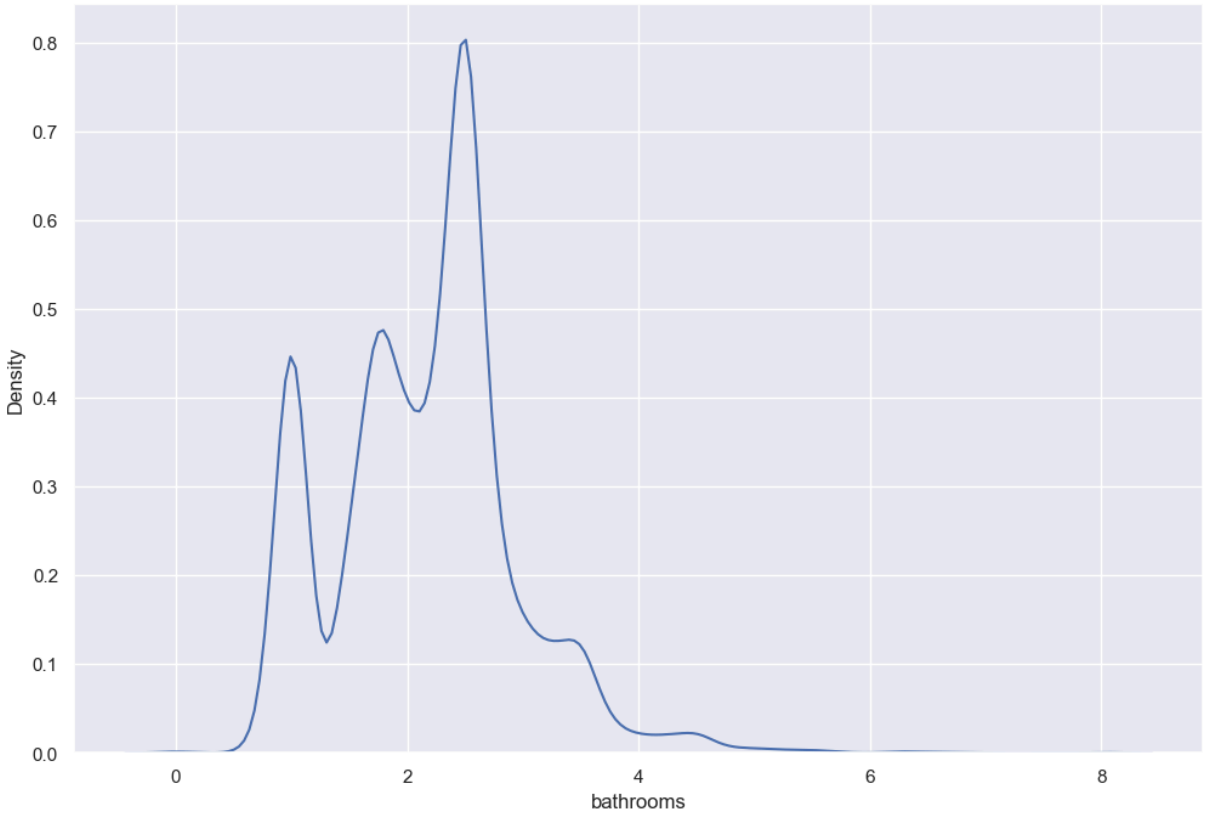
In [21]:
```
sns.set(rc={'figure.figsize':(12,8)})
sns.distplot(df['bathrooms'],kde=False,bins=20);   ##Kernal density estimati
```



his will generate and display a histogram of the bathrooms column with 20 bins, and it will not include a kernel density estimate (KDE) curve. The figure size is set to 12x8 inches, as specified.

In [22]:
```
sns.kdeplot(df['bathrooms']) # Kernal Density Estimation
```

Out[22]:
```
<Axes: xlabel='bathrooms', ylabel='Density'>
```



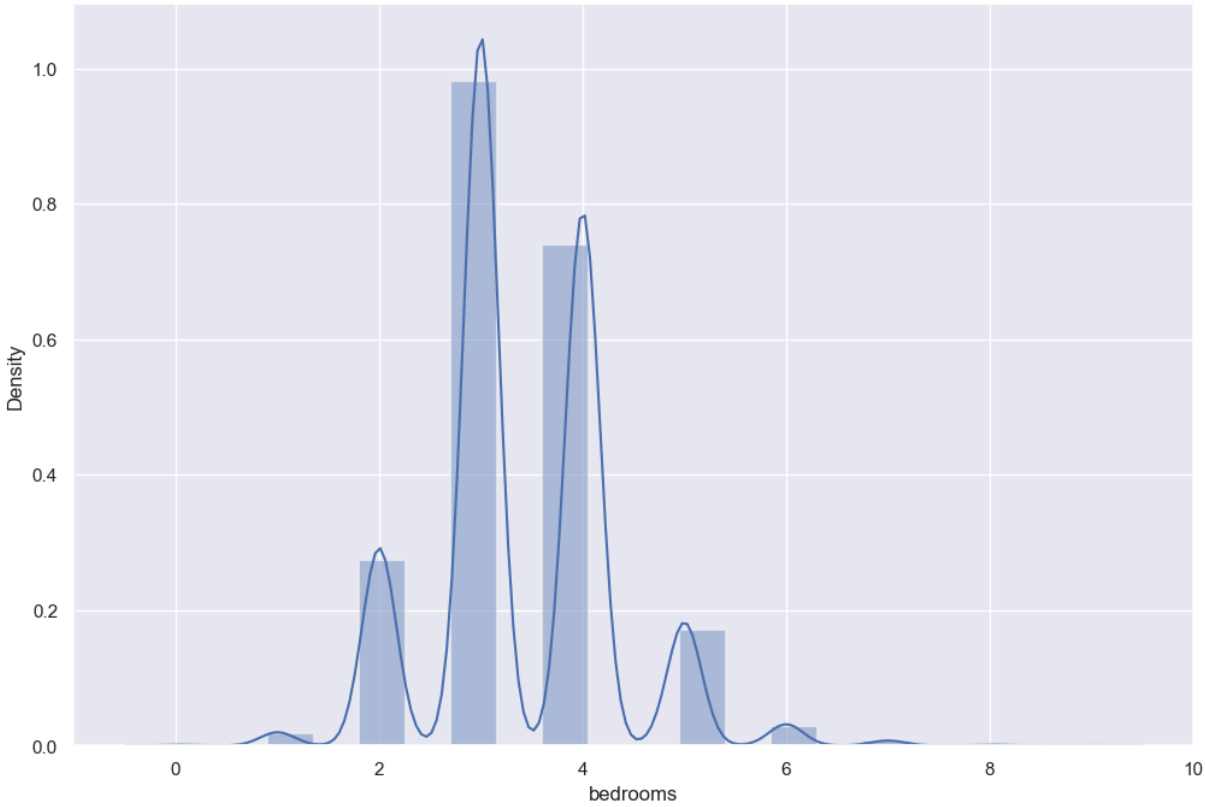
A KDE plot of the bathrooms column, illustrating the estimated probability density function of the data distribution.

In [23]:
```
df['bathrooms'].describe()
```

Out[23]:
```
count    4600.000000
mean        2.160815
std         0.783781
min         0.000000
25%         1.750000
50%         2.250000
75%         2.500000
max         8.000000
Name: bathrooms, dtype: float64
```

Summary statistics including count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum values of the bathrooms column.

In [24]:
```
sns.set(rc={'figure.figsize':(12,8)})
sns.distplot(df['bedrooms'],bins=20);
```



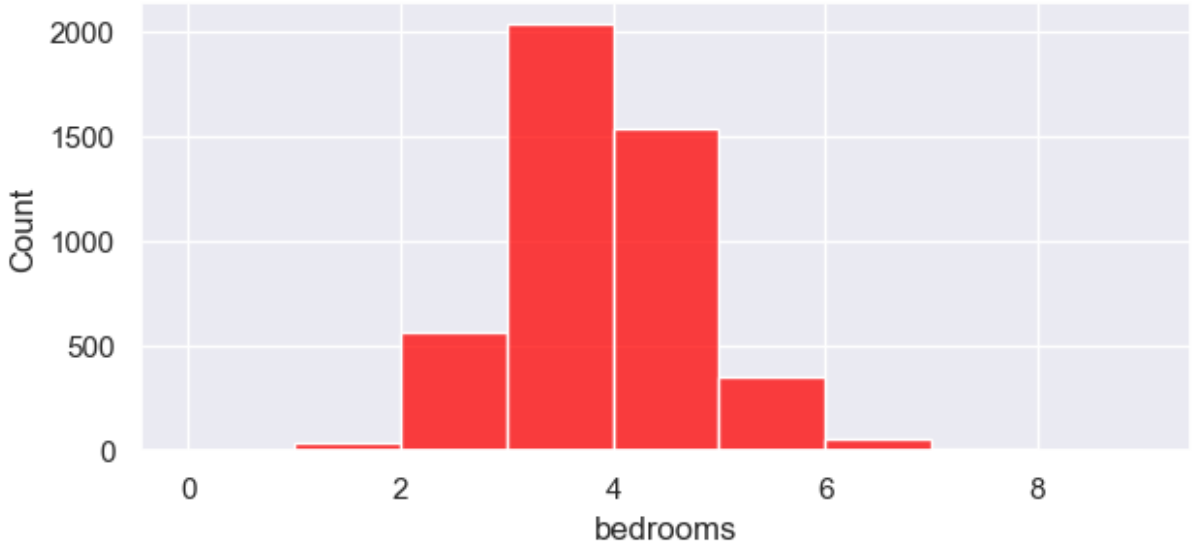
A histogram of the bedrooms column with 20 bins, displaying the distribution of the data.

Histogram¶

In exploratory data analysis(EDA),a histogram is a graphical representation that shows the distibution of a dataset.It display the frequency of data points falling within specified ranges or bins.By plotting a histogram,you can quickly see patterns,such as the distribution shape,cental tendency,and spread of the data.It helps in identifying skewness, outliers,and the overall distribution of the data,which is essential for understanding the underlying patterns and macking informed decisions.

In [25]:
```
plt.figure(figsize=(7,3))

sns.histplot(x="bedrooms",data=df,color="red",binwidth=1)
```

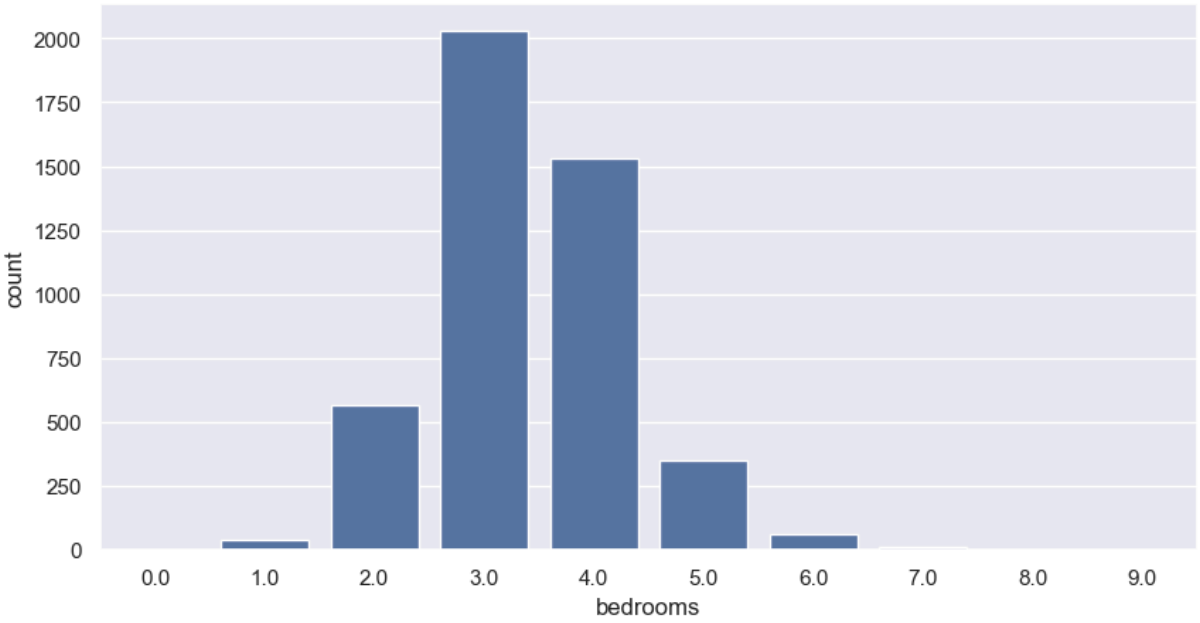
Out[25]:
```
<Axes: xlabel='bedrooms', ylabel='Count'>
```



A red histogram of the bedrooms column with bin width of 1, displayed in a 7x3 inch figure.

In [26]:
```
plt.figure(figsize=(10,5))

sns.countplot(x='bedrooms',data=df)
```

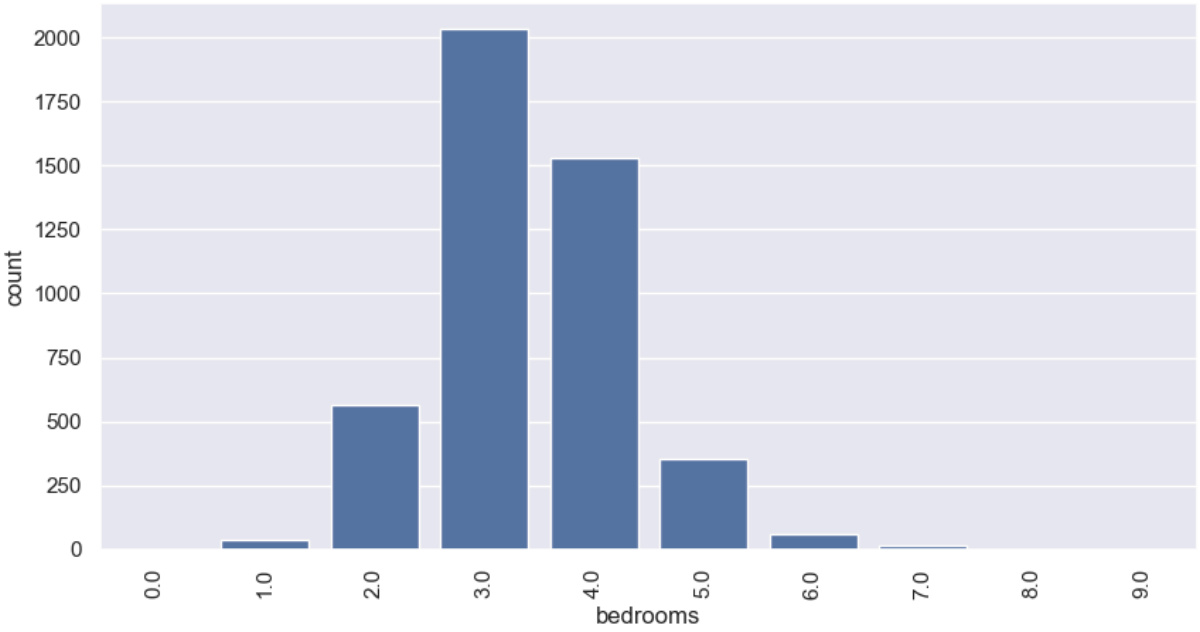
Out[26]:
```
<Axes: xlabel='bedrooms', ylabel='count'>
```



A bar plot showing the count of each unique value in the bedrooms column, displayed in a 10x5 inch figure.

In [27]:
```
plt.figure(figsize=(10,5))

sns.countplot(x='bedrooms',data=df)
plt.xticks(rotation=90)
```

Out[27]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, '0.0'),
  Text(1, 0, '1.0'),
  Text(2, 0, '2.0'),
  Text(3, 0, '3.0'),
  Text(4, 0, '4.0'),
  Text(5, 0, '5.0'),
  Text(6, 0, '6.0'),
  Text(7, 0, '7.0'),
  Text(8, 0, '8.0'),
  Text(9, 0, '9.0')])
```

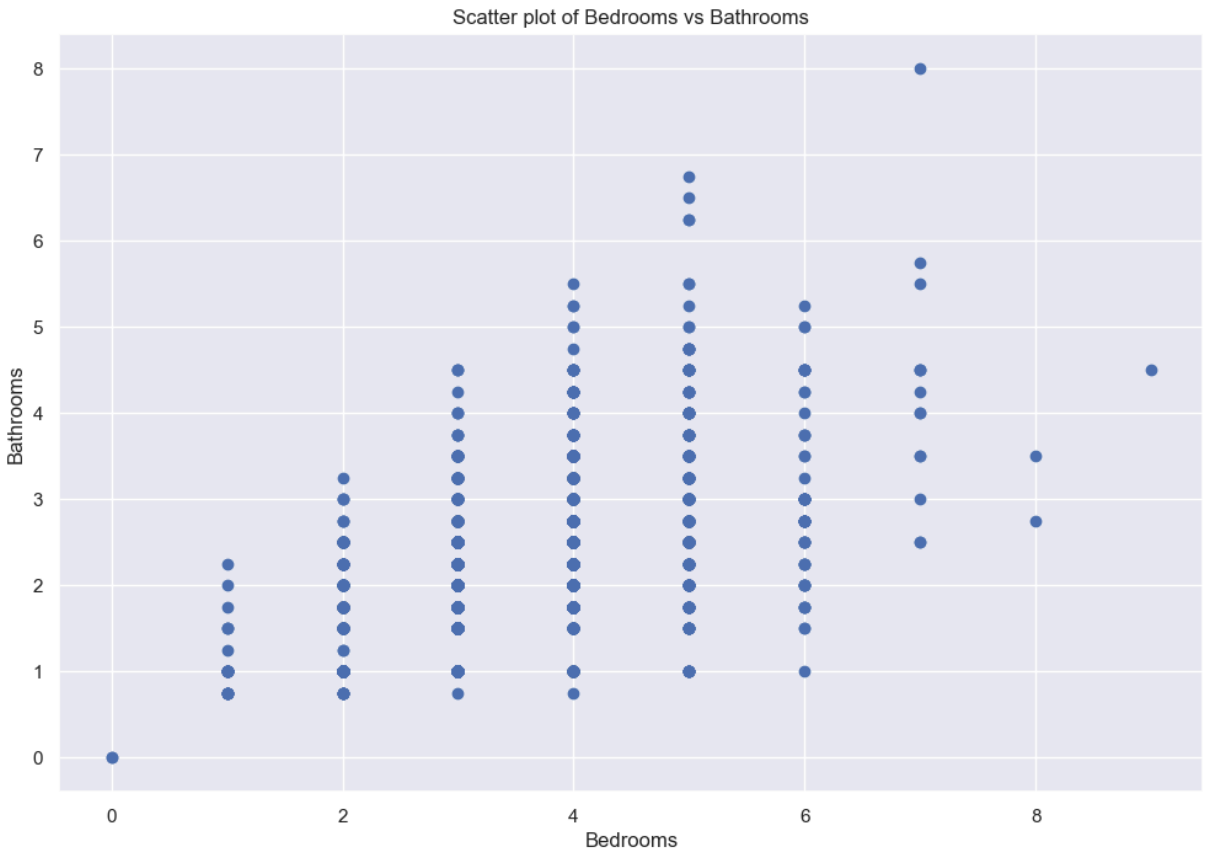


This code generates a bar plot of counts for each unique value in the bedrooms column from the DataFrame df, with the figure size set to 10x5 inches and x-axis labels rotated 90 degrees.

Scatterplot

A scatter plot is a two-dimentional data visualization that uses dots to represent the values obtained for two different variables-one is plotted along the x-axis and the other plotted along the y-axis.

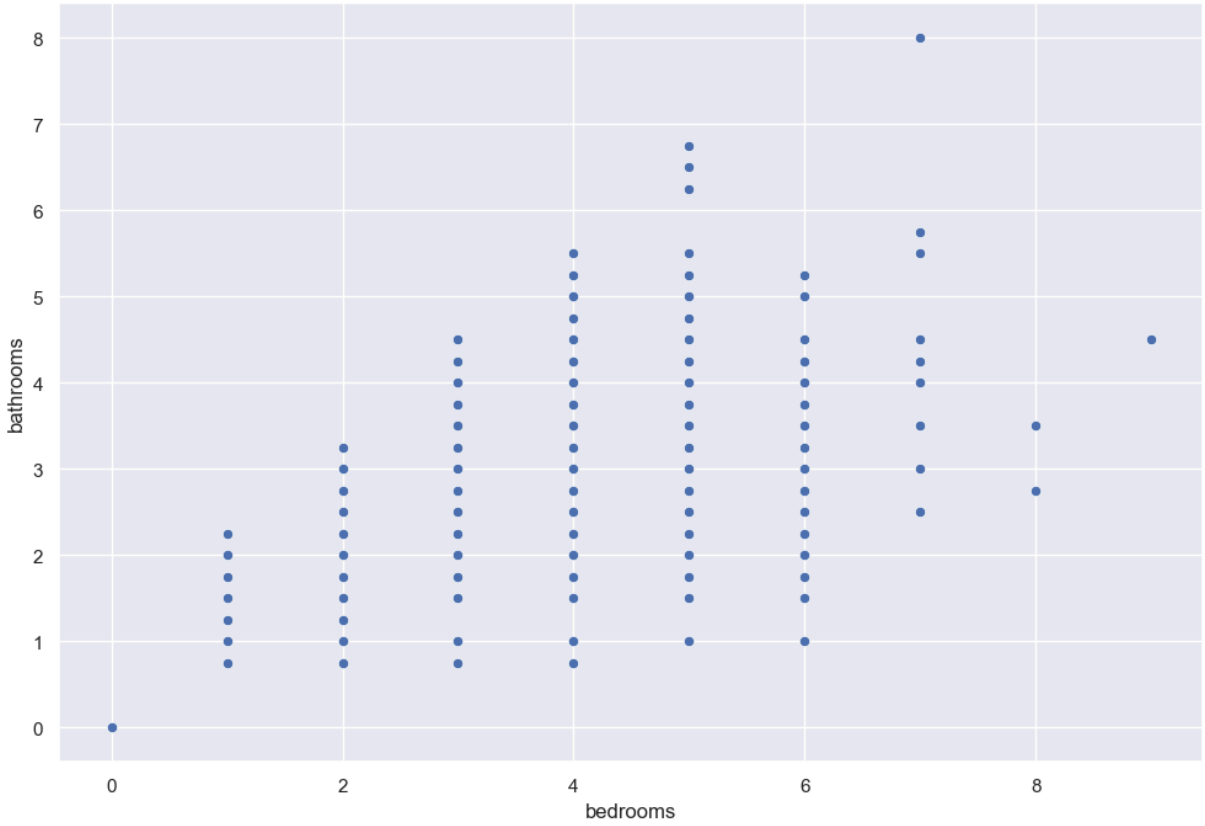
In [28]:
```
plt.scatter(x="bedrooms",y="bathrooms",data=df)
plt.xlabel("Bedrooms")
plt.ylabel("Bathrooms")
plt.title("Scatter plot of Bedrooms vs Bathrooms")
plt.show()
```



A scatter plot of bedrooms vs bathrooms with labeled axes and a title, showing the relationship between the two variables.

In [29]:
```
sns.scatterplot(x="bedrooms",y="bathrooms",data=df)
```

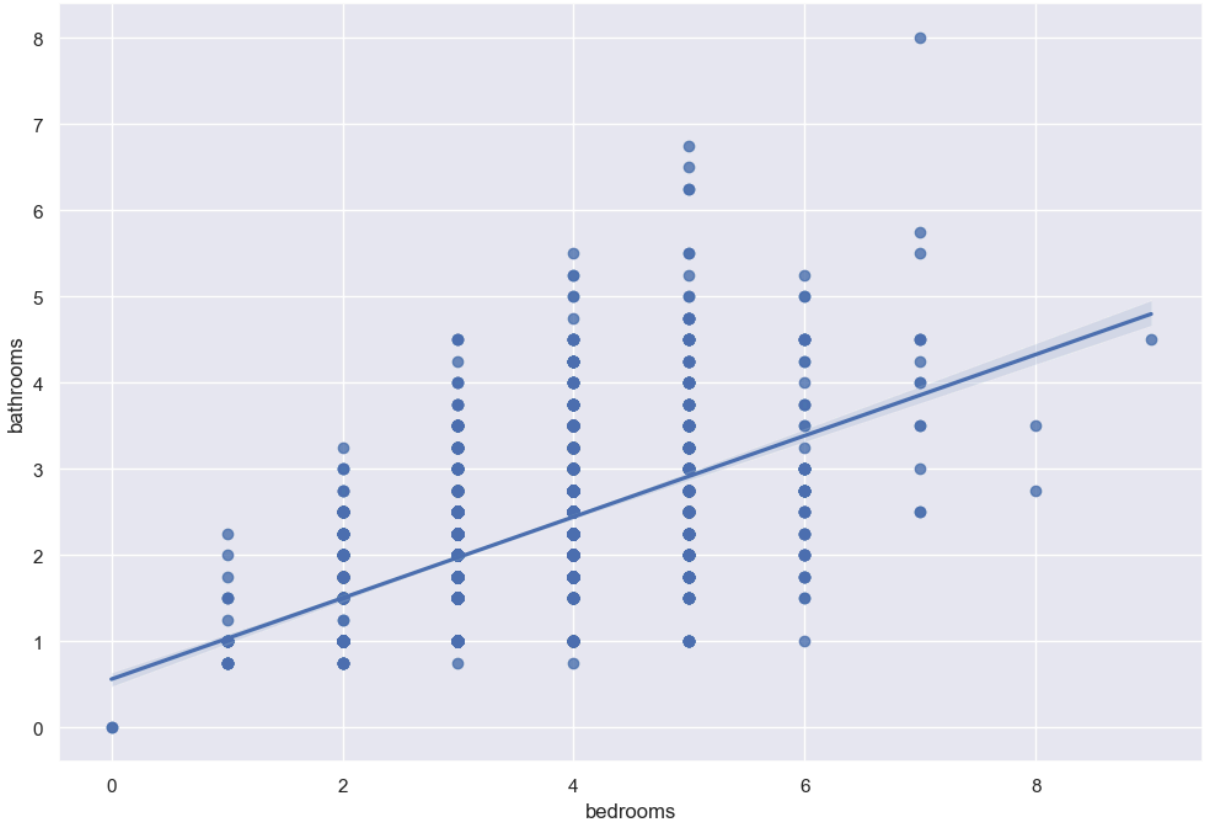
Out[29]:
```
<Axes: xlabel='bedrooms', ylabel='bathrooms'>
```



A scatter plot showing the relationship between bedrooms and bathrooms.

In [30]:
```
sns.regplot(x="bedrooms",y="bathrooms",data=df,scatter=True,fit_reg=True)
```

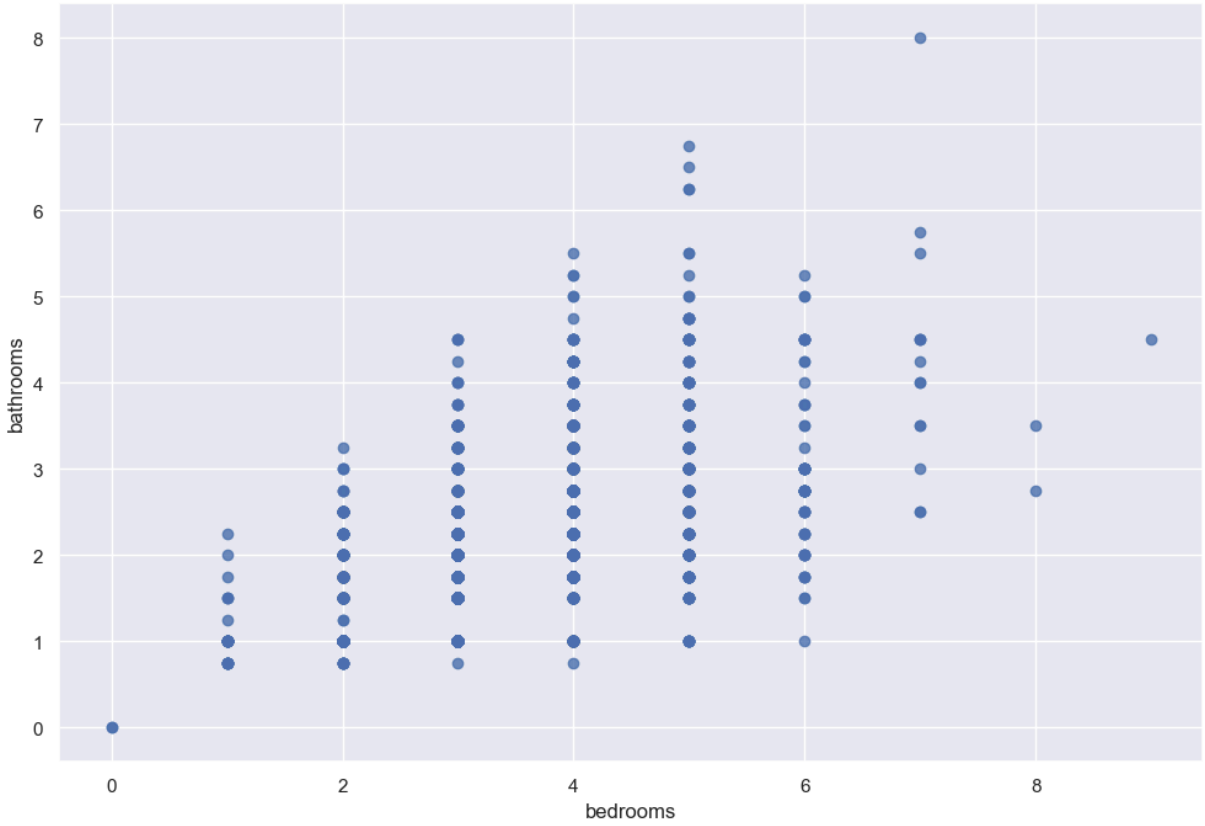
Out[30]:
```
<Axes: xlabel='bedrooms', ylabel='bathrooms'>
```



This code generates a scatter plot of bedrooms versus bathrooms with a fitted regression line, showing both the data points and the regression line.

In [31]:
```
sns.regplot(x="bedrooms",y="bathrooms",data=df,scatter=True,fit_reg=False)
```

Out[31]:
```
<Axes: xlabel='bedrooms', ylabel='bathrooms'>
```



In [32]:
```
import pandas as pd
col_names = ['sepal_length','sepal_width','petal_length','petal_width','spec

csv_url = 'data (1).csv'
flowers_df = pd.read_csv(csv_url, names  = col_names)
```

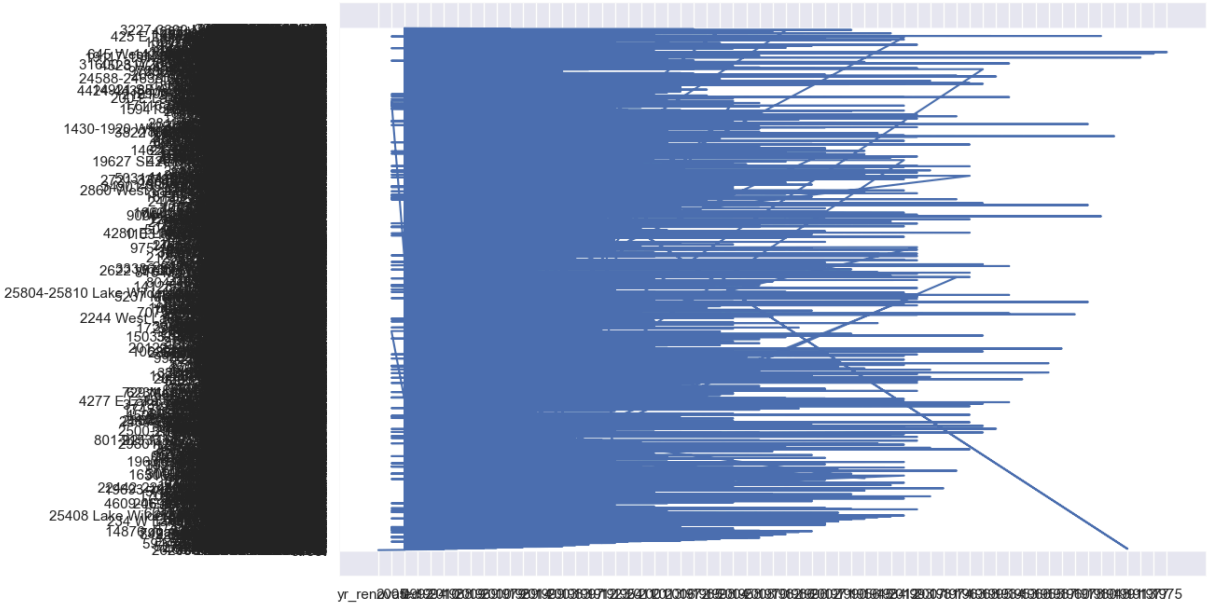
In [33]:
```
flowers_df.columns
```

Out[33]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [34]:
```
flowers_df['species'].unique()
```

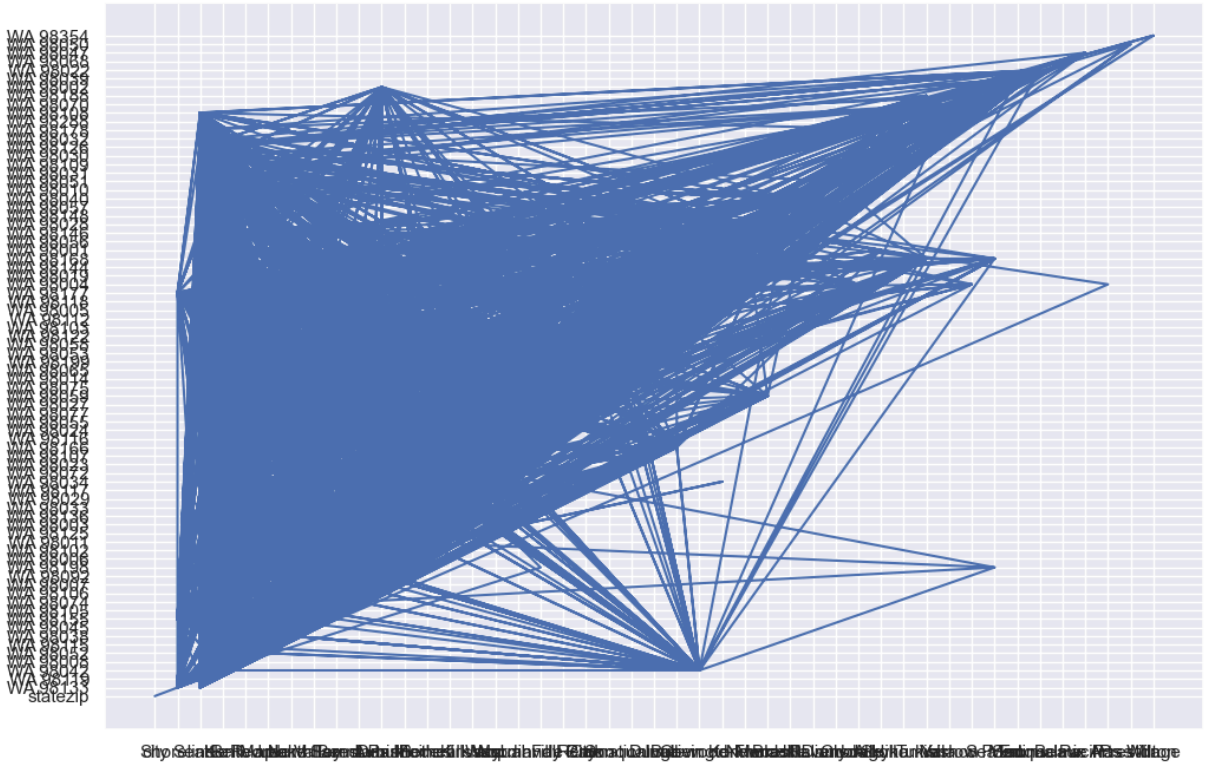
Out[34]:
```
array(['country', 'USA'], dtype=object)
```

In [35]:
```
plt.plot(flowers_df.sepal_length,flowers_df.sepal_width)
plt.show()
```



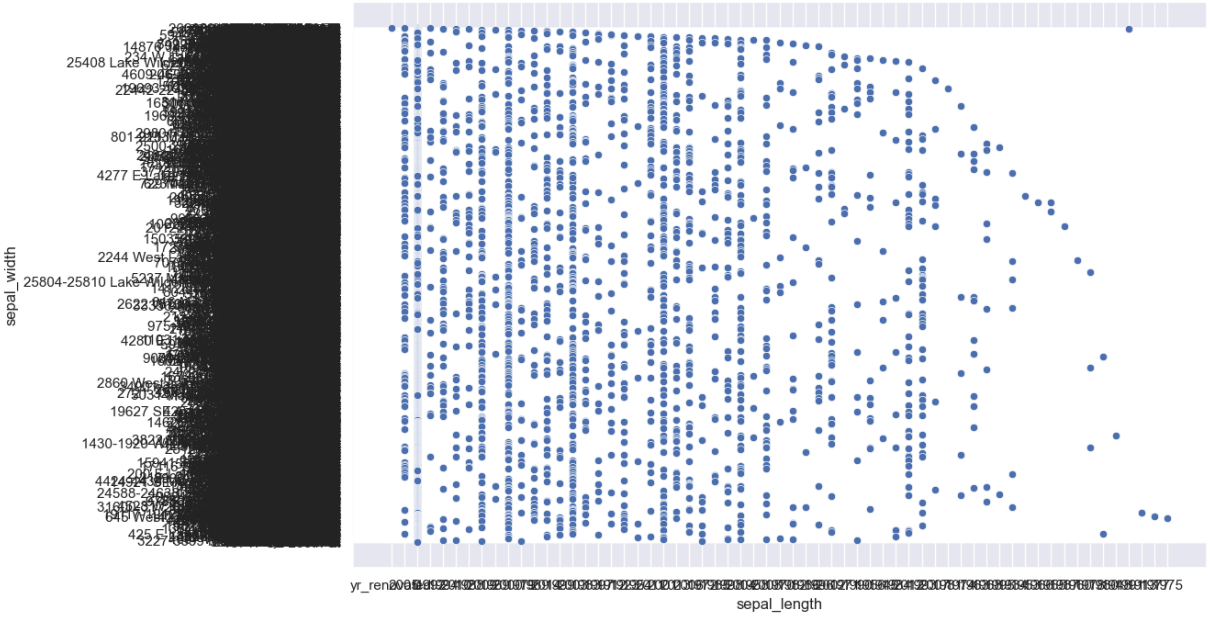
A line plot of sepal_length vs sepal_width showing the trend between these two variables.

In [36]:
```
plt.plot(flowers_df.petal_length,flowers_df.petal_width);
plt.show()
```



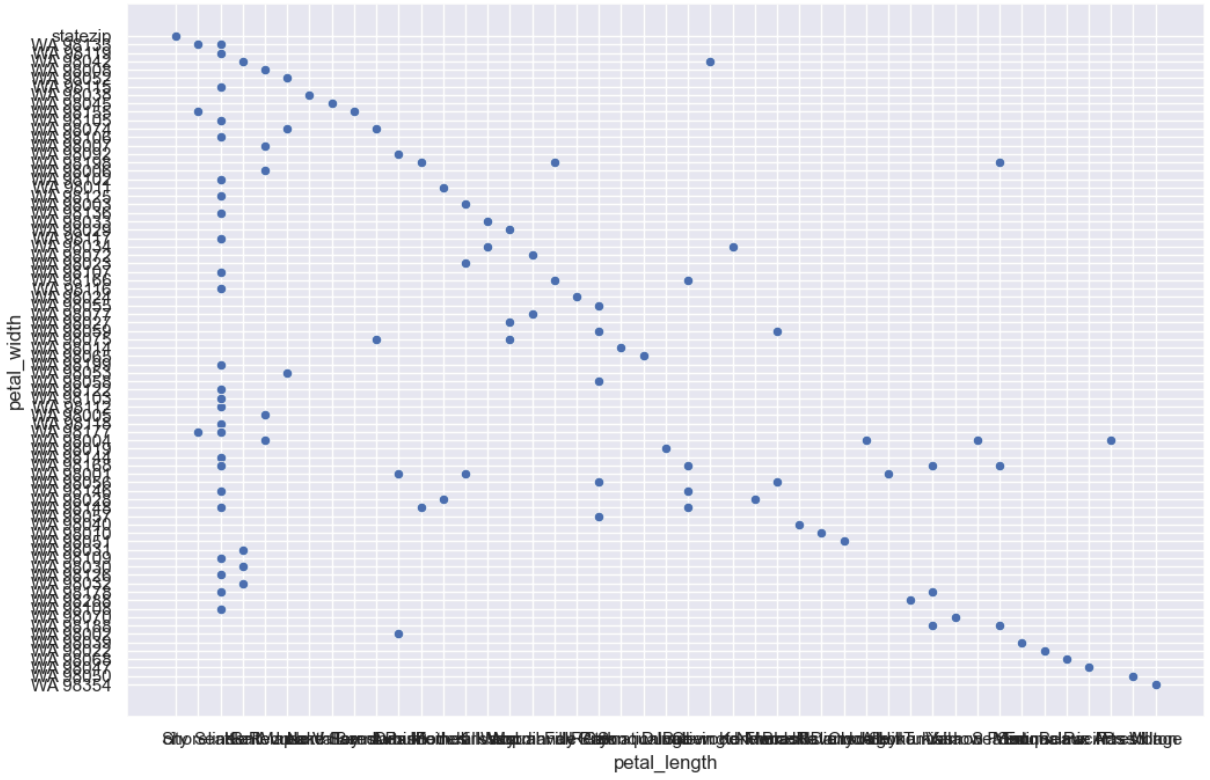
A line plot of petal_length vs petal_width, showing the trend between these two variables.

In [37]:
```
sns.scatterplot(x=flowers_df.sepal_length,y=flowers_df.sepal_width);
plt.show()
```



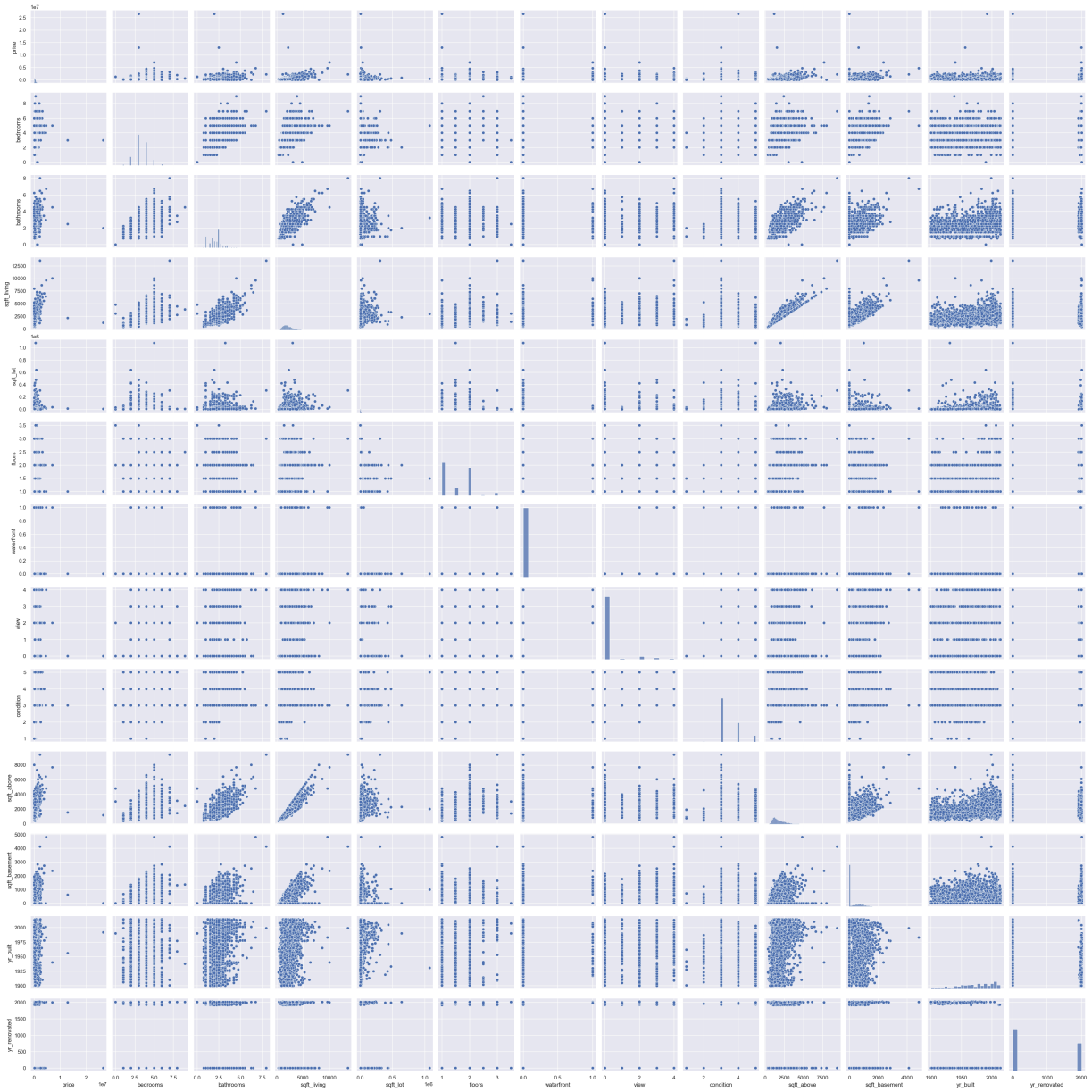
This code generates a scatter plot of sepal_length versus sepal_width from the flowers_df DataFrame, displaying the individual data points for these two variables.

In [38]:
```
sns.scatterplot(x=flowers_df.petal_length,y=flowers_df.petal_width);
plt.show()
```



A scatter plot of petal_length versus petal_width, showing the distribution of data points for these variables.

In [39]:
```
sns.pairplot(df)
plt.show()
```



A matrix of scatter plots and histograms showing pairwise relationships and distributions of all numeric variables in the DataFrame.

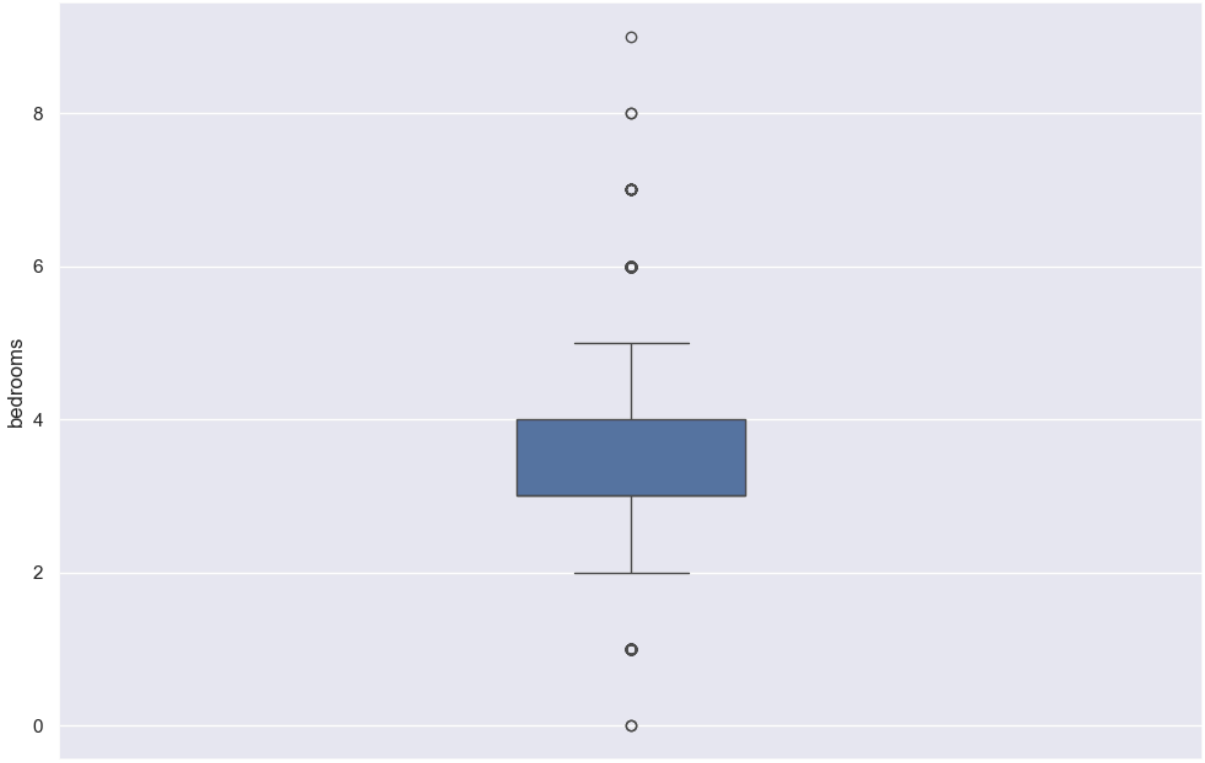
Boxplots

A boxplot,also known as a box-and-whisker plot,is a graphical representation used to visualize the distribution and summary statistics of a dataset.

In [40]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

In [41]:
```
sns.boxplot(y='bedrooms',data=df, width=0.2)
```

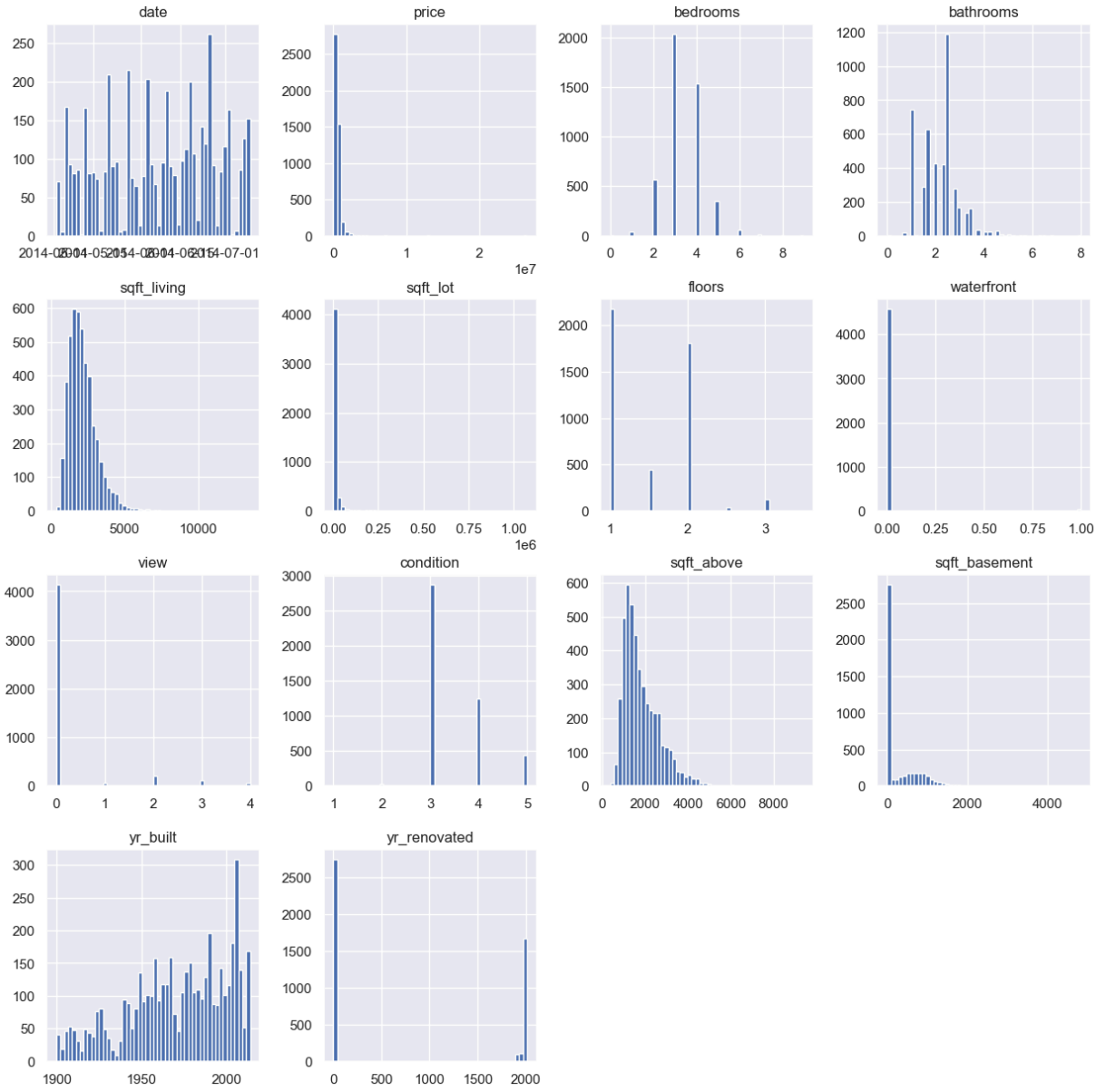
Out[41]:
```
<Axes: ylabel='bedrooms'>
```



A vertical box plot of bedrooms with a box width of 0.2, showing the distribution and summary statistics of the data.

In [42]:
```
df.hist(bins=50, figsize=(15, 15))
```

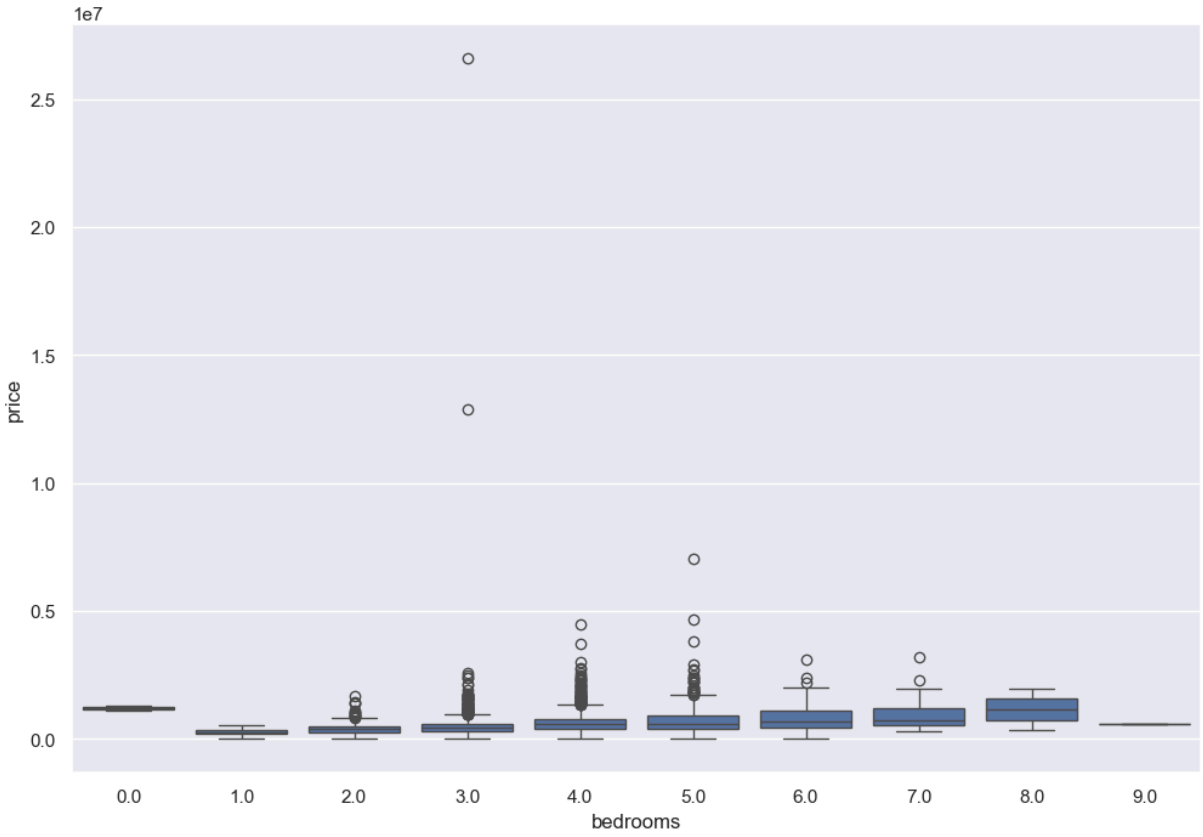
Out[42]:
```
array([[<Axes: title={'center': 'date'}>,
        <Axes: title={'center': 'price'}>,
        <Axes: title={'center': 'bedrooms'}>,
        <Axes: title={'center': 'bathrooms'}>],
       [<Axes: title={'center': 'sqft_living'}>,
        <Axes: title={'center': 'sqft_lot'}>,
        <Axes: title={'center': 'floors'}>,
        <Axes: title={'center': 'waterfront'}>],
       [<Axes: title={'center': 'view'}>,
        <Axes: title={'center': 'condition'}>,
        <Axes: title={'center': 'sqft_above'}>,
        <Axes: title={'center': 'sqft_basement'}>],
       [<Axes: title={'center': 'yr_built'}>,
        <Axes: title={'center': 'yr_renovated'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



A grid of histograms for all numeric columns in the DataFrame, with 50 bins each, displayed in a 15x15 inch figure.

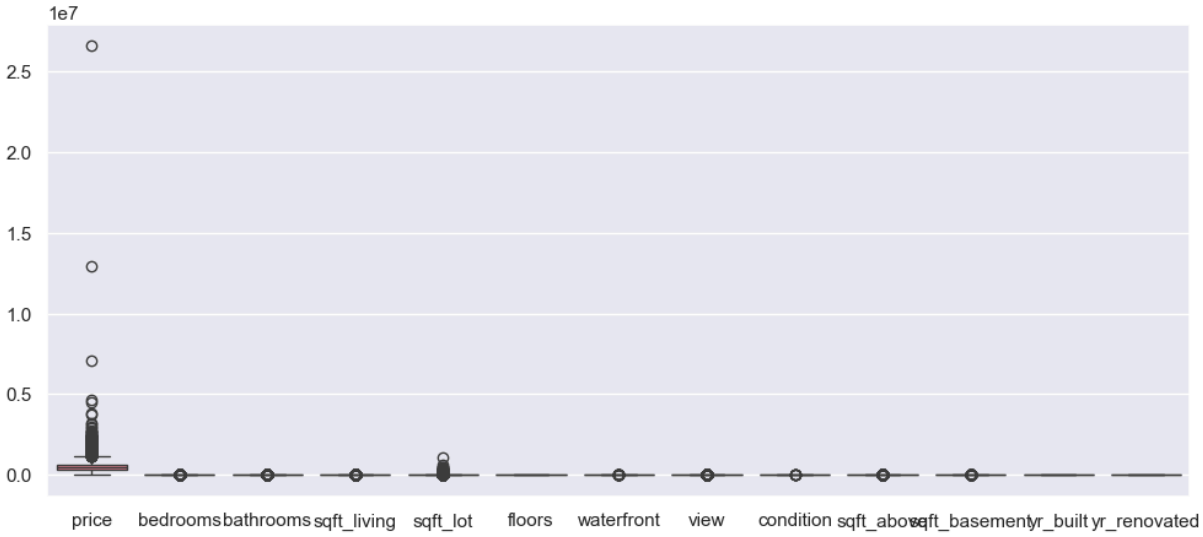
In [43]:
```
sns.boxplot(x='bedrooms',y='price',data=df)
```

Out[43]:
```
<Axes: xlabel='bedrooms', ylabel='price'>
```

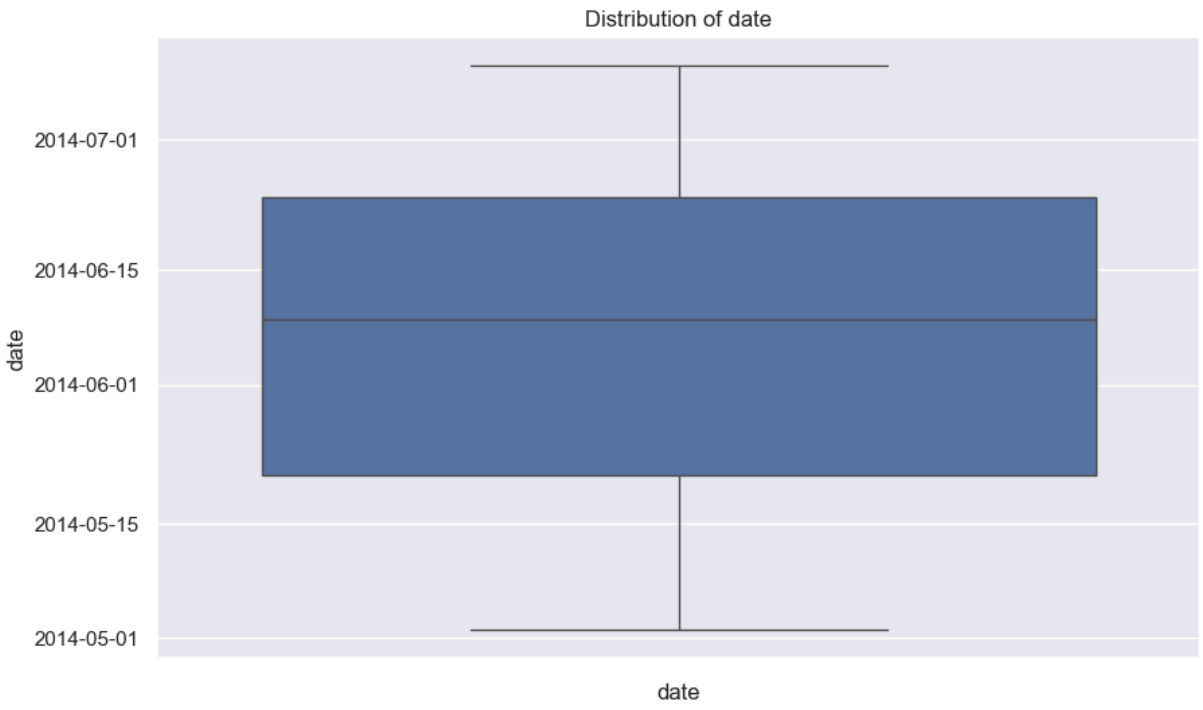


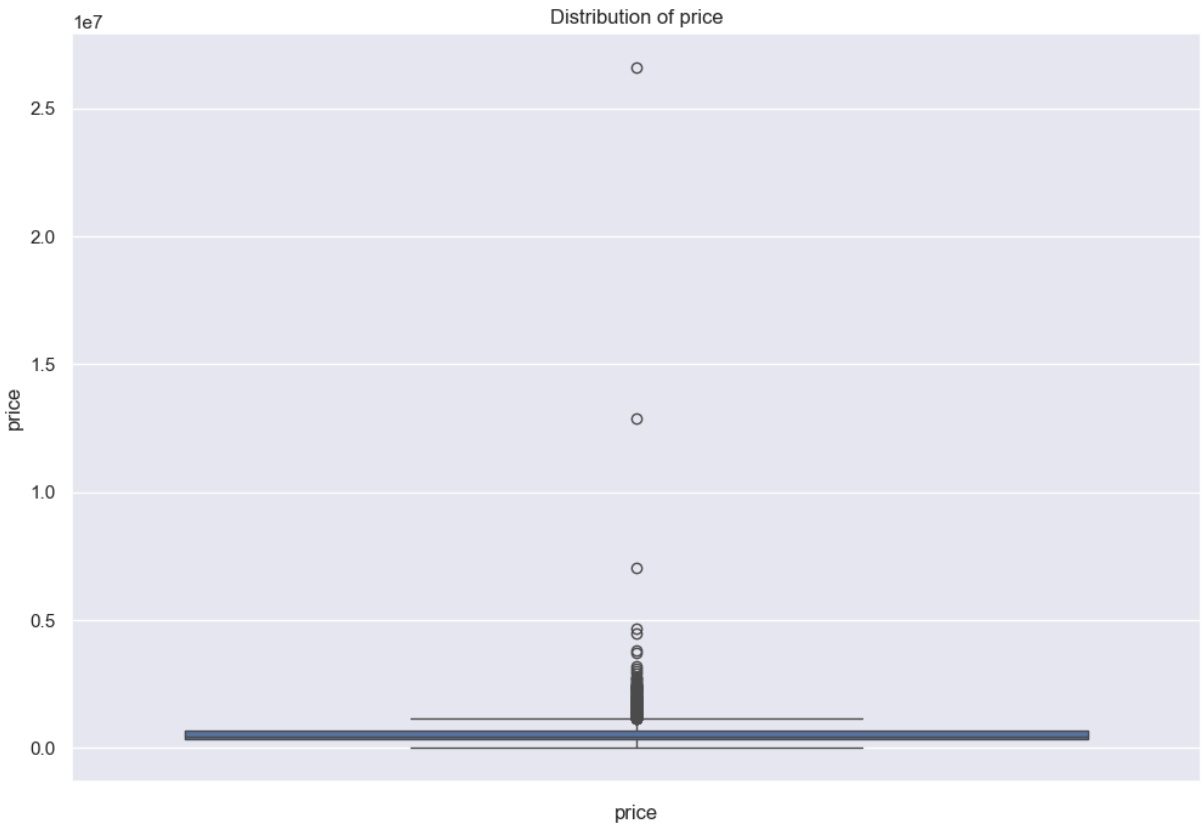
In [44]:
```
plt.figure(figsize=(12,5))
sns.boxplot(data=df)
```

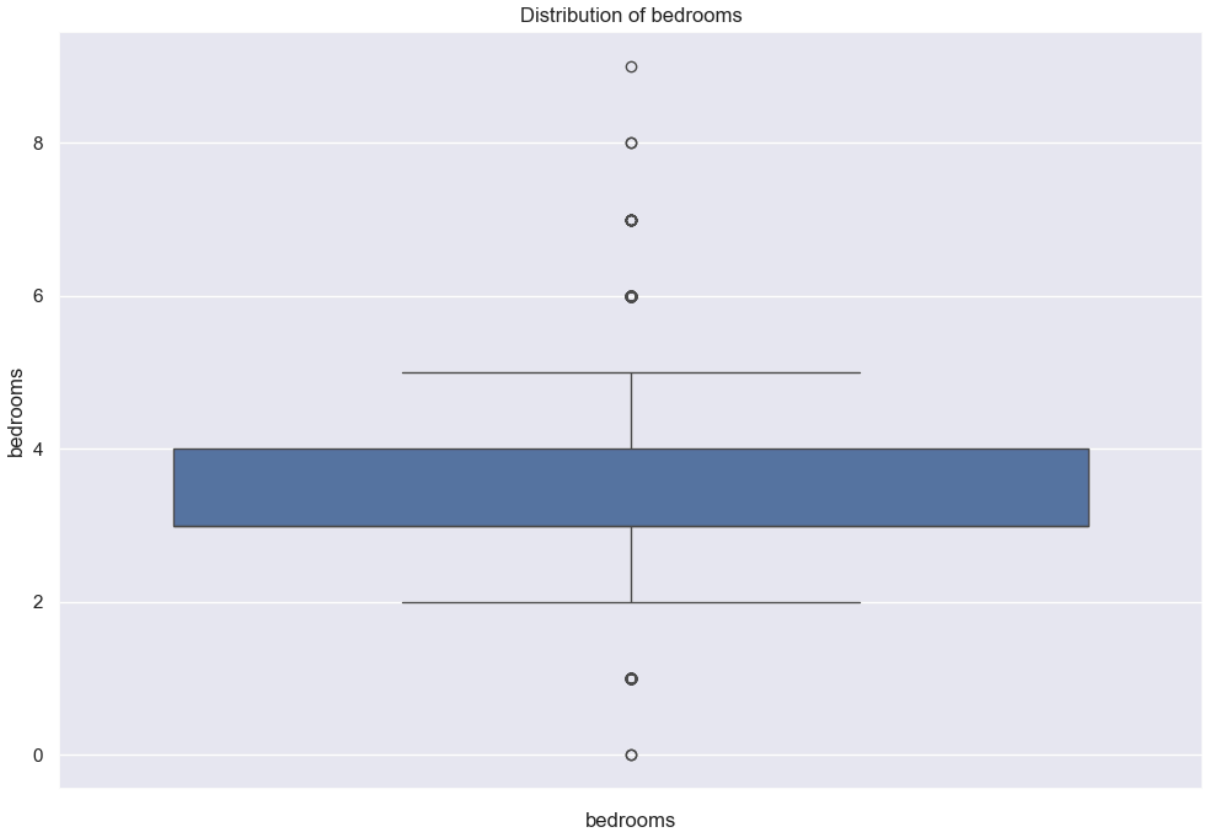
Out[44]: <Axes: >

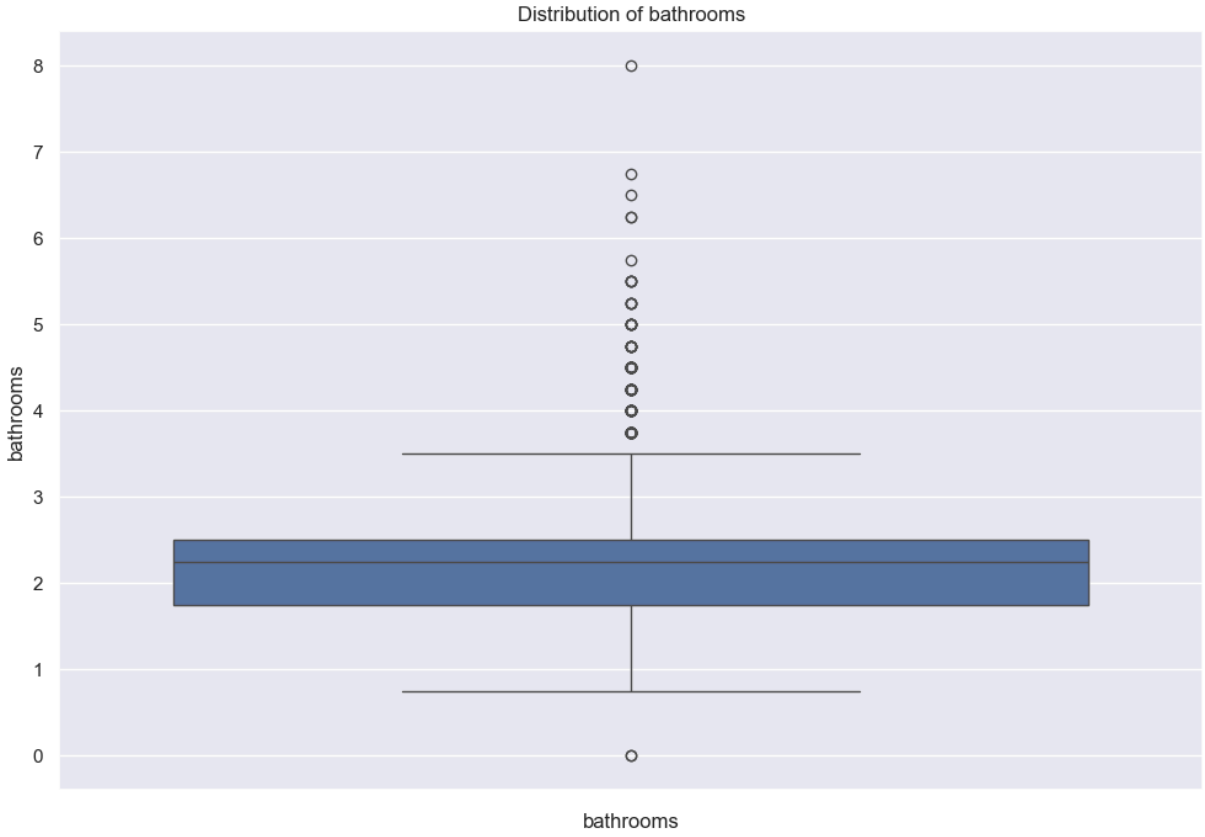


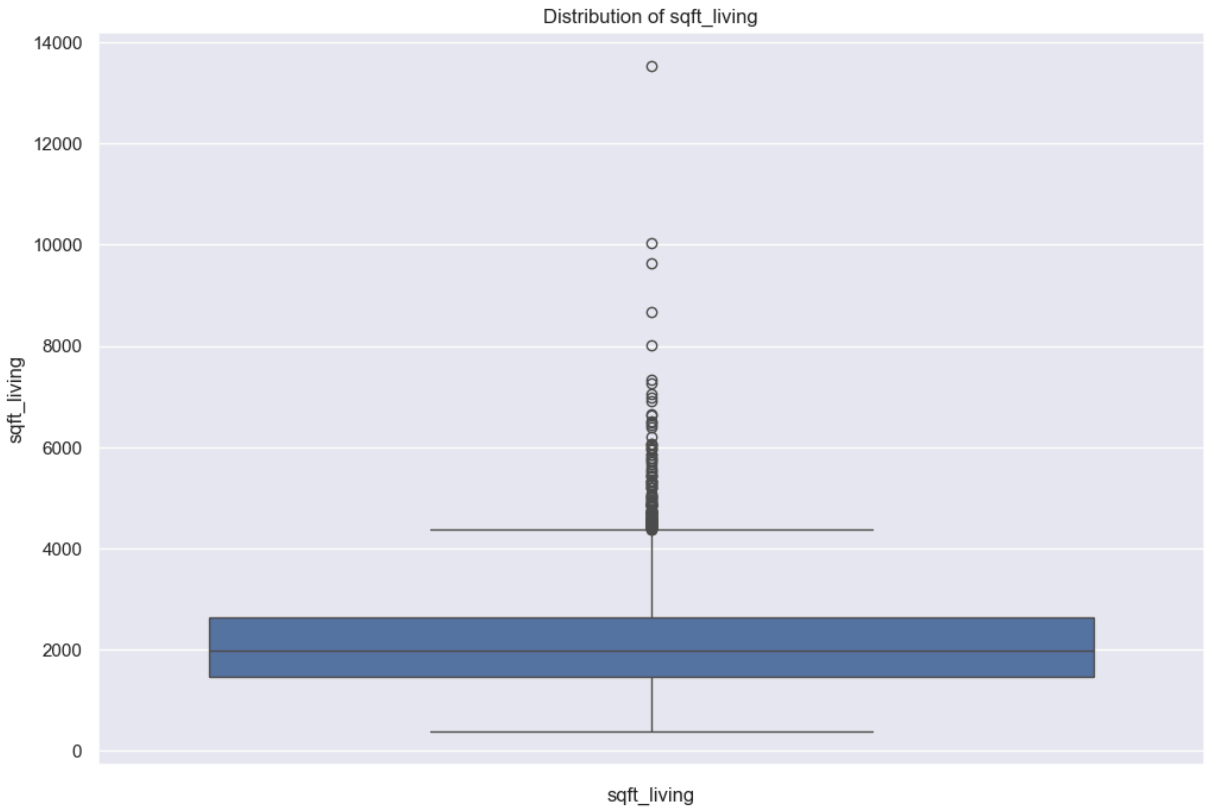
In [45]:
```
plt.figure(figsize=(10, 6))
for col in df.columns:
    sns.boxplot(df[col])
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.xticks(rotation=45)
    plt.show()
```

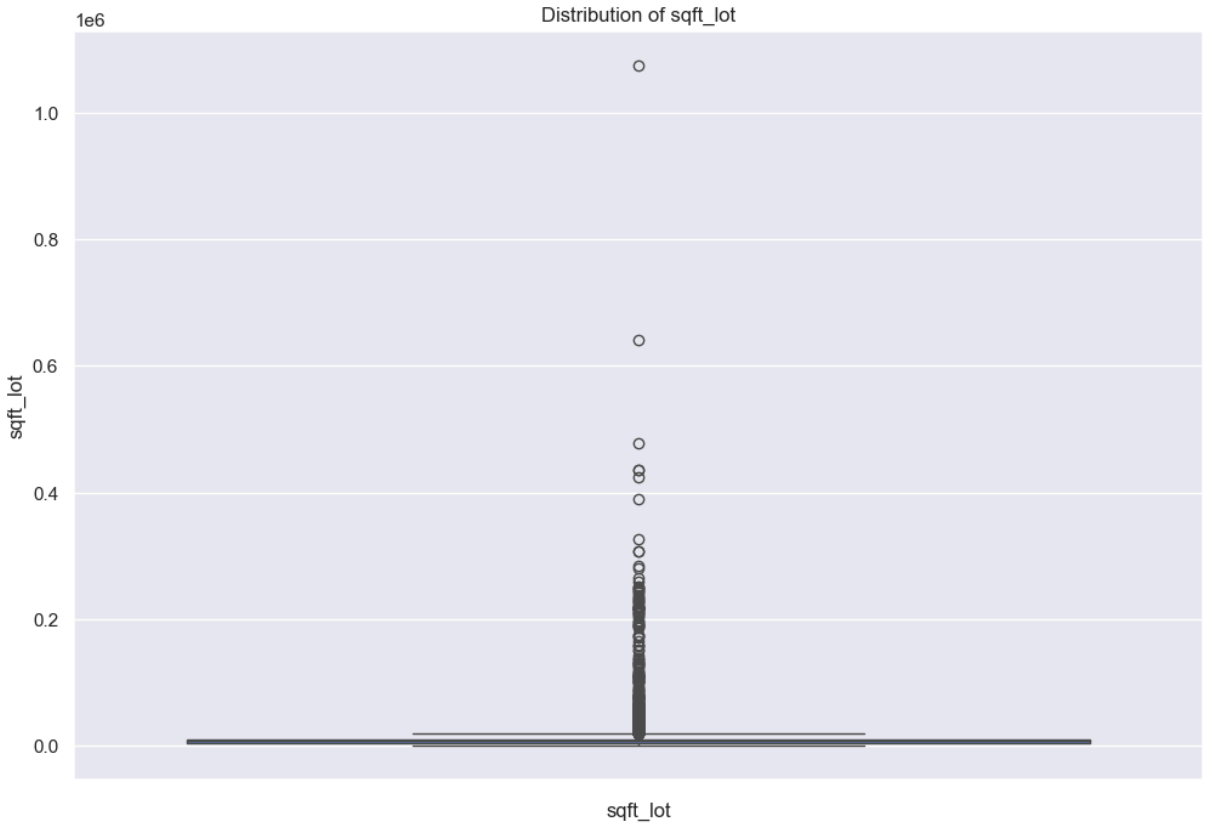

Distribution of date
2014-07-01
2014-06-15
2014-06-01
2014-05-15
2014-05-01
date
date

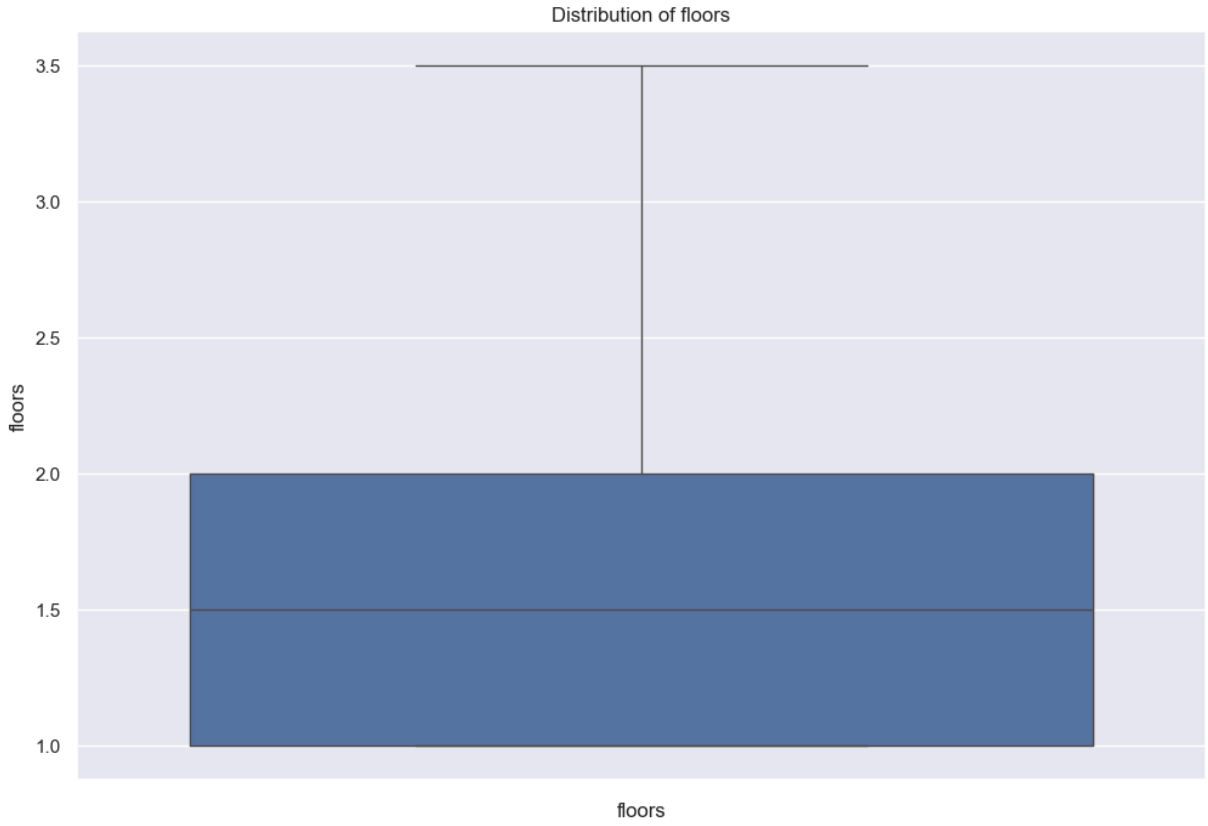

Distribution of price
1e7
2.5
2.0
1.5
1.0
0.5
0.0
price
price

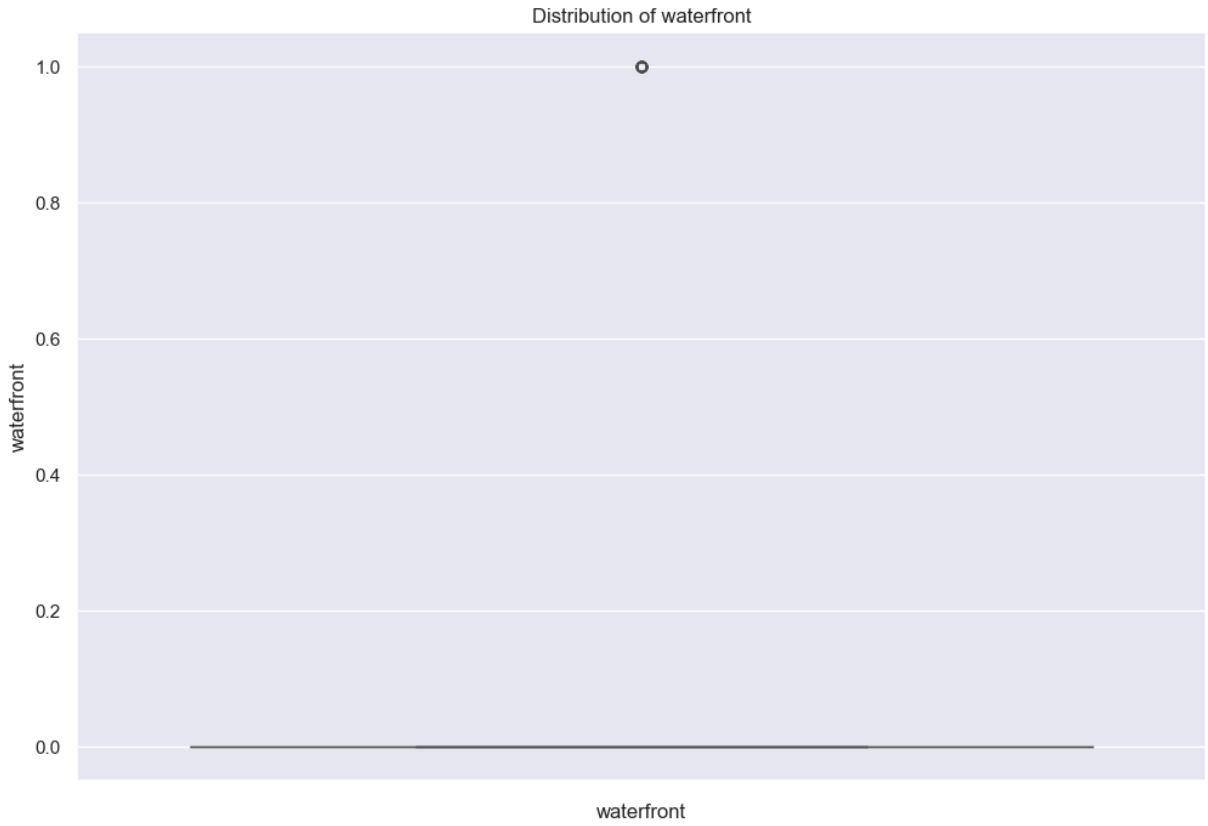

Distribution of bedrooms
bedrooms
0
2
4
6
8
bedrooms

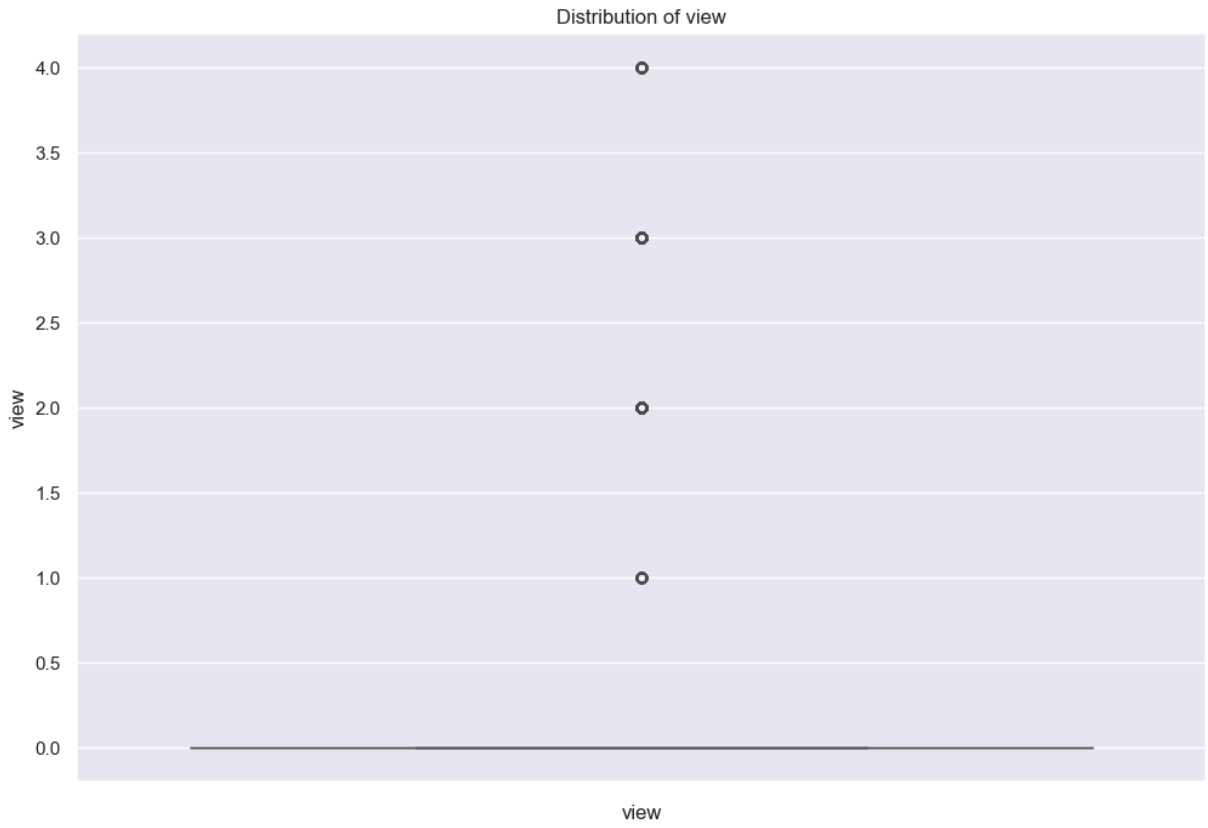

Distribution of bathrooms
bathrooms
0
1
2
3
4
5
6
7
8
bathrooms

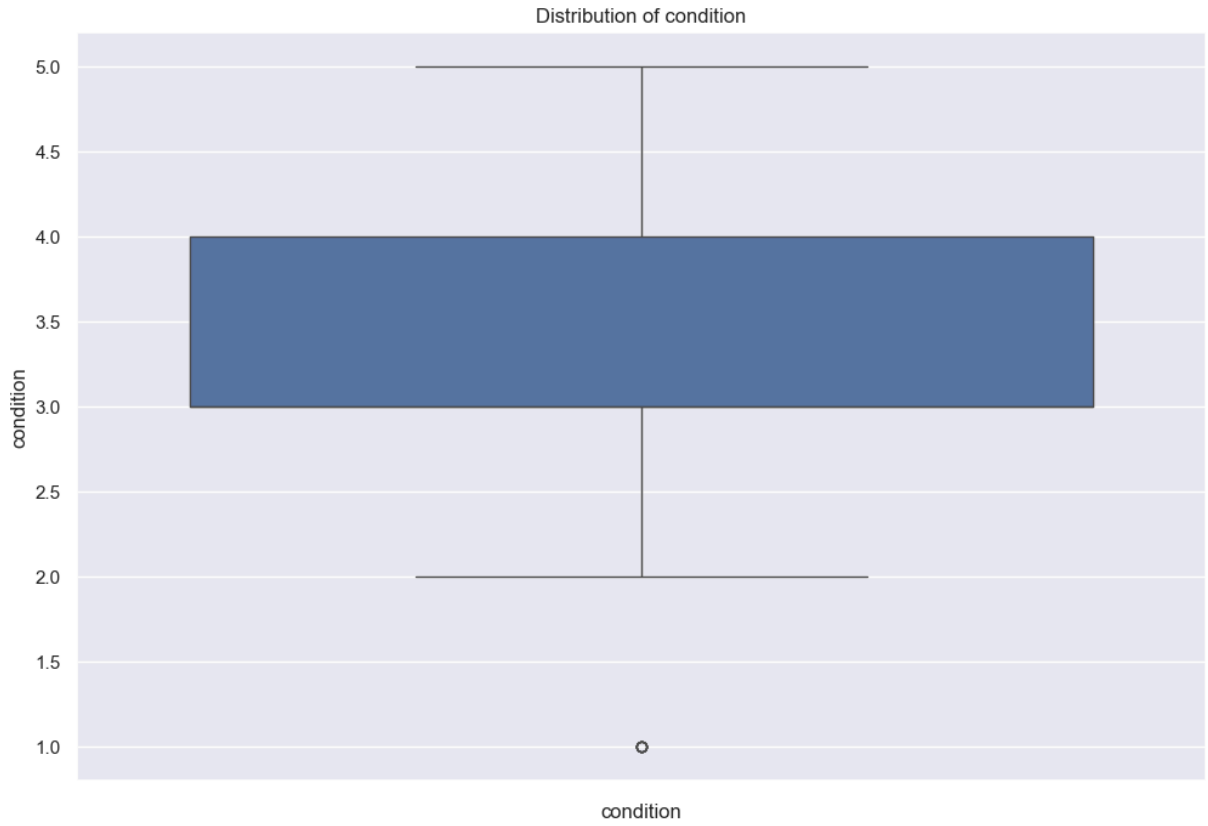

Distribution of sqft_living
14000
12000
10000
8000
6000
4000
2000
0
sqft_living
sqft_living

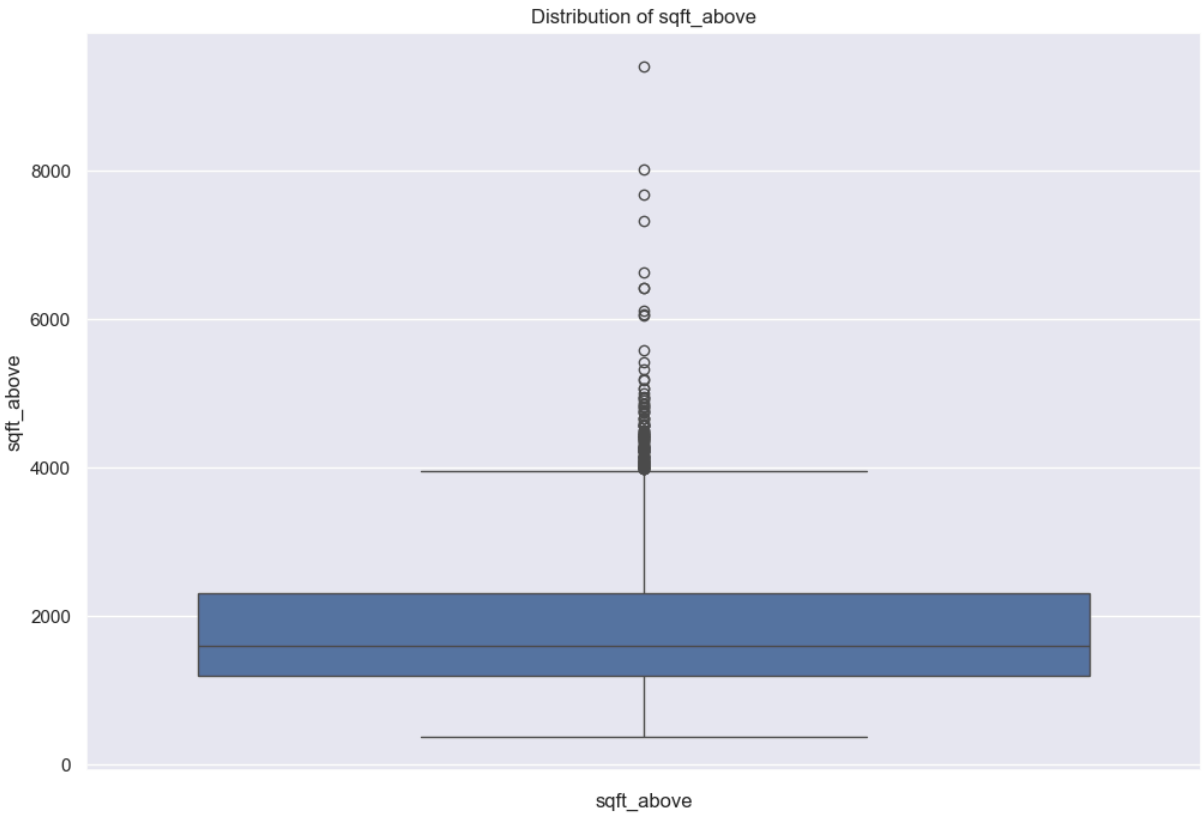

Distribution of sqft_lot
1e6
1.0
0.8
0.6
0.4
0.2
0.0
sqft_lot
sqft_lot

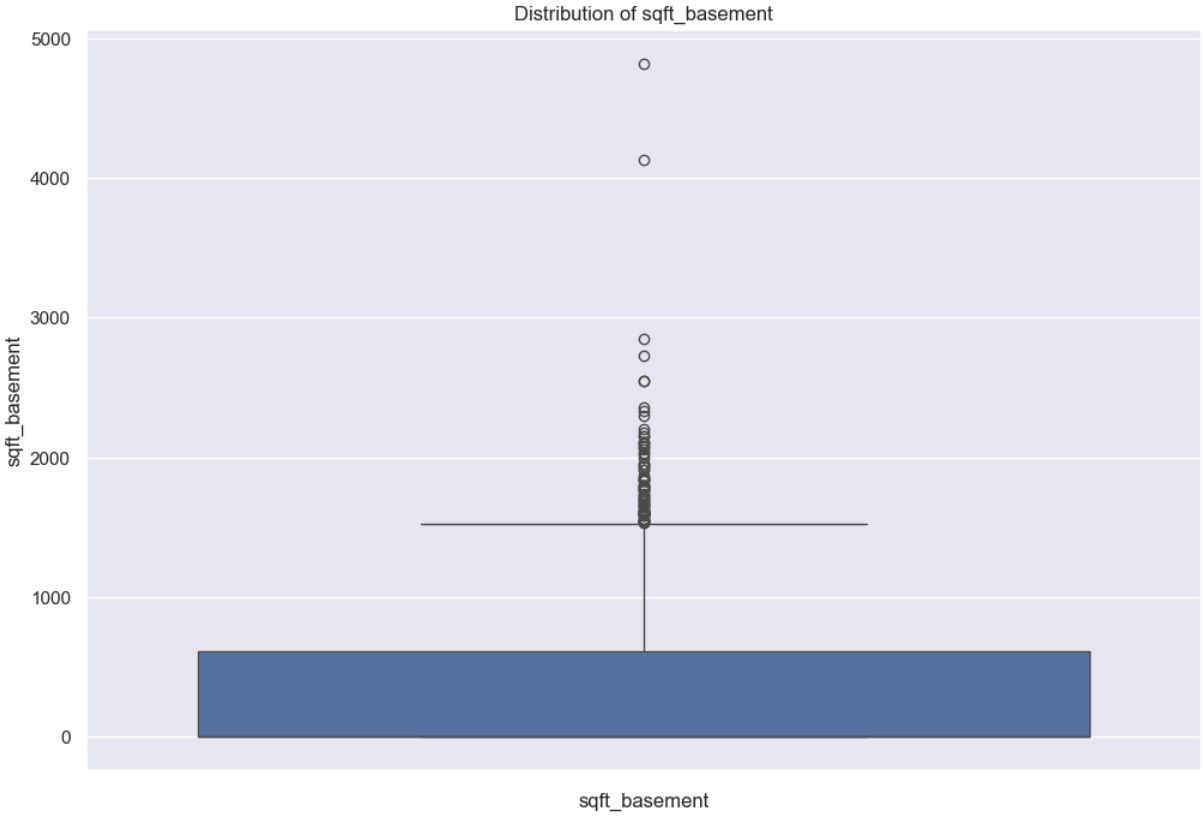

Distribution of floors
3.5
3.0
2.5
2.0
1.5
1.0
floors
floors

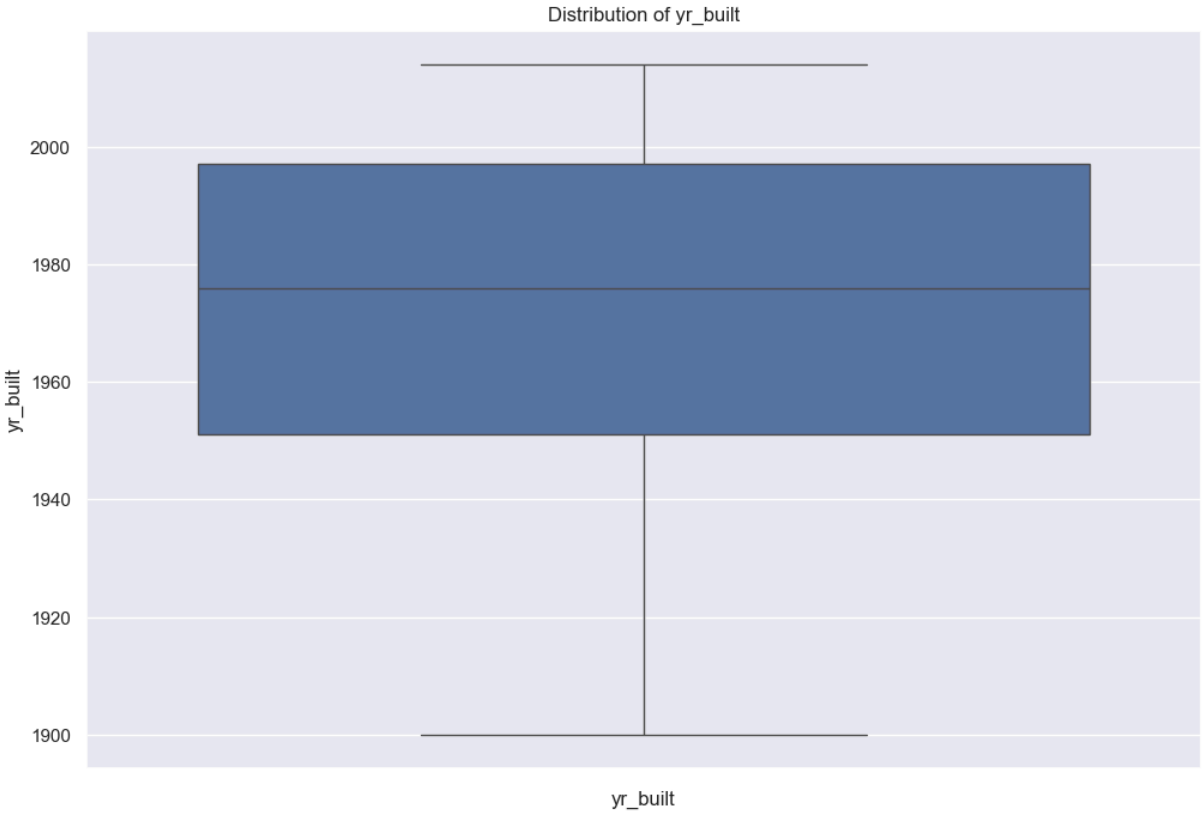

Distribution of waterfront
1.0
0.8
0.6
0.4
0.2
0.0
waterfront
waterfront


Distribution of view
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
view
view

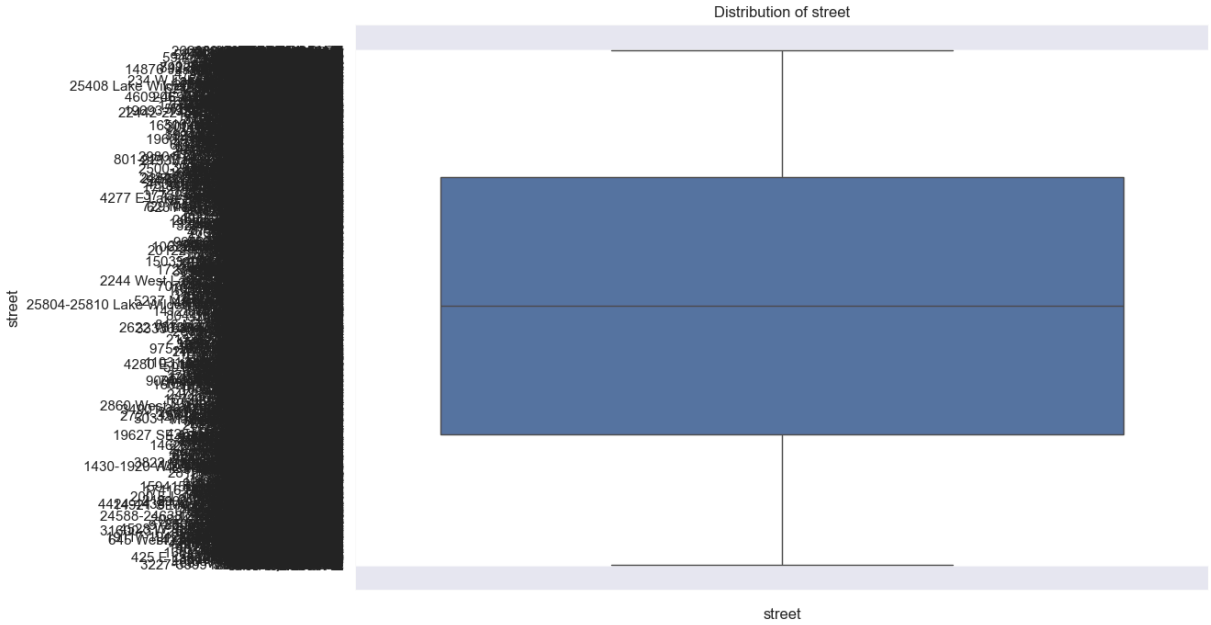

Distribution of condition
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
condition
condition

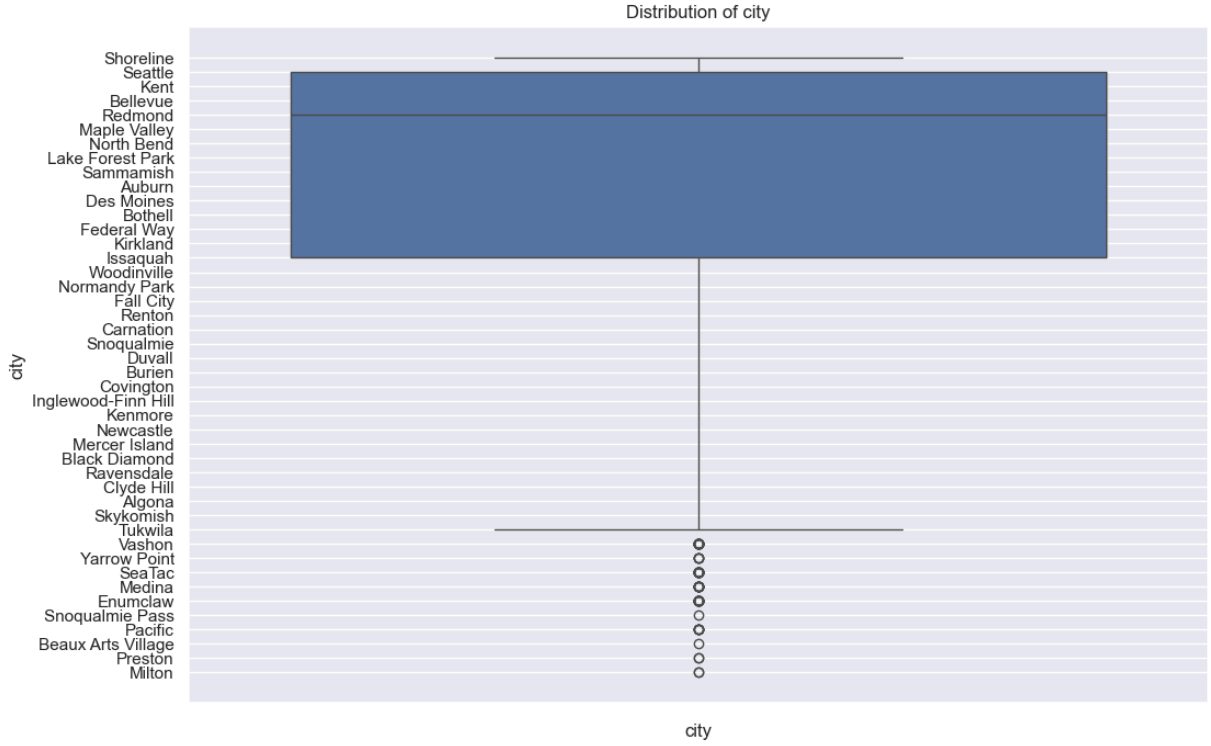

Distribution of sqft_above
8000
6000
4000
2000
0
sqft_above
sqft_above

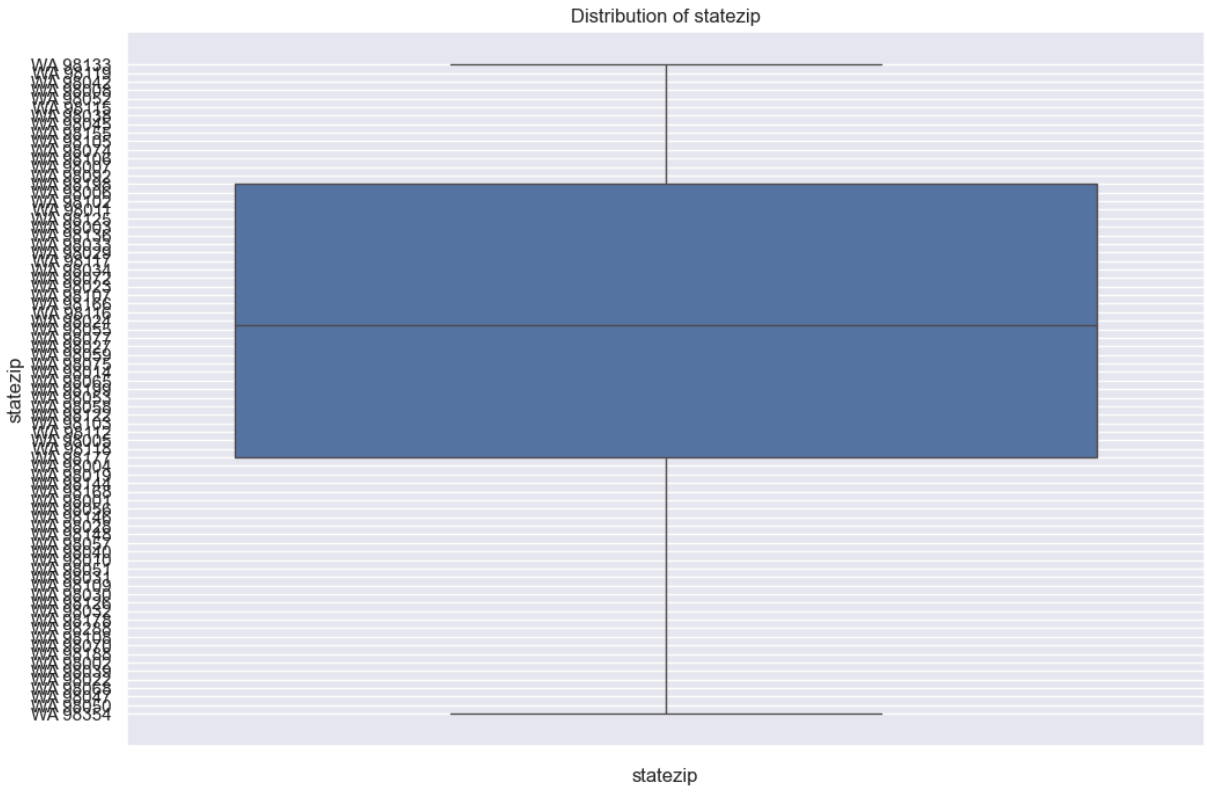

Distribution of sqft_basement
5000
4000
3000
2000
1000
0
sqft_basement
sqft_basement

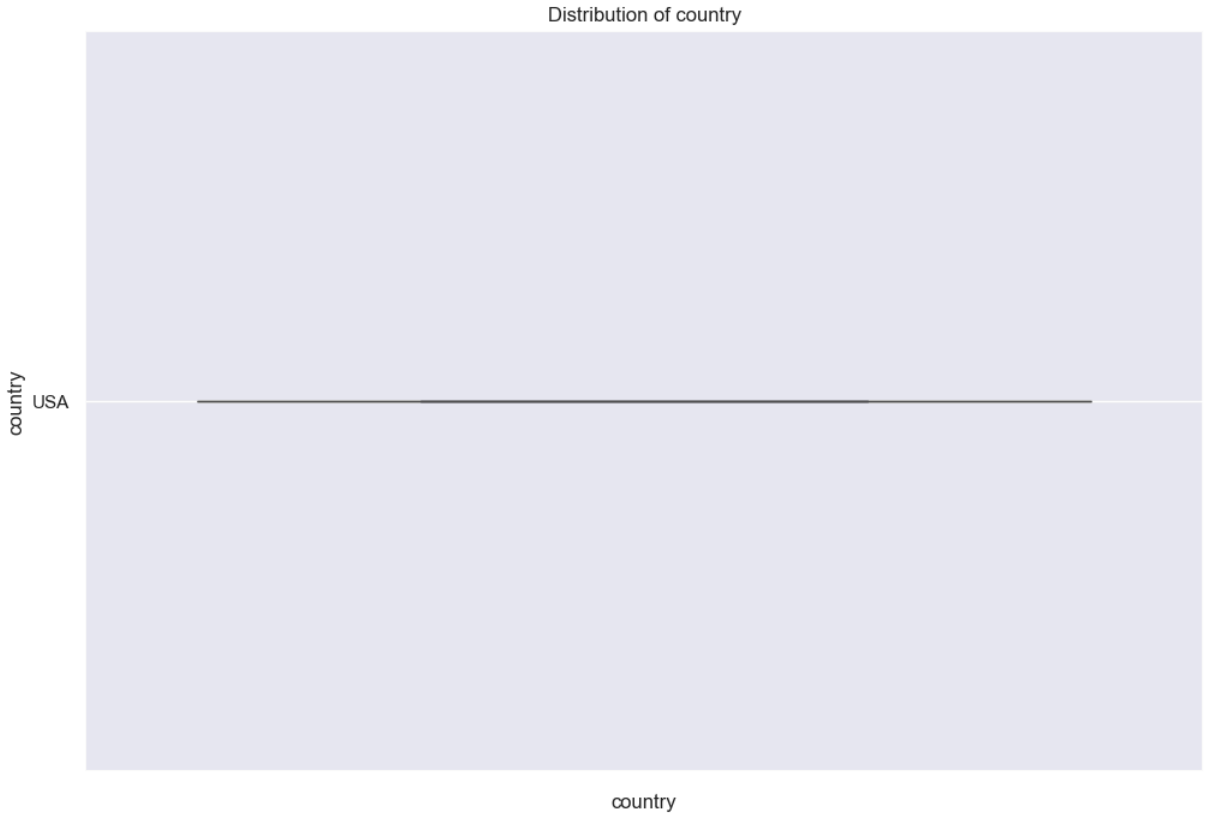

Distribution of yr_built
2000
1980
1960
1940
1920
1900
yr_built
yr_built


Distribution of yr_renovated
2000
1750
1500
1250
1000
750
500
250
0
yr_renovated
yr_renovated


Distribution of street
street
street


Distribution of city
Shoreline
Seattle
Kent
Bellevue
Redmond
Maple Valley
North Bend
Lake Forest Park
Sammamish
Auburn
Des Moines
Bothell
Federal Way
Kirkland
Issaquah
Woodinville
Normandy Park
Fall City
Renton
Carnation
Snoqualmie
Duvall
Burien
Covington
Inglewood-Finn Hill
Kenmore
Newcastle
Mercer Island
Black Diamond
Ravensdale
Clyde Hill
Algona
Skykomish
Tukwila
Vashon
Yarrow Point
SeaTac
Medina
Enumclaw
Snoqualmie Pass
Pacific
Beaux Arts Village
Preston
Milton
city
city





Individual box plots for each column in the DataFrame, with titles, labeled x-axes, and rotated x-axis labels, displayed one at a time in a 10x6 inch figure.

Conclusion of House Price Prediction Data Analysis

The house price prediction dataset enables the development of models to estimate property values based on various features and historical data.

This notebook was converted to PDF with convert.ploomber.io