Work Integrated Learning Programmes Division
M.Tech (Data Science and Engineering)
Data Visualization And Interpretation

Assignment -Python – PS1 [Weightage 13%]

1. Do not change the name of the data file that is shared with the problem statement.
2. If intermediate data files are created, retain in the present working directory and attach them during submission.
3. Retain the data file in the same directory as that of this workbook.
4. Retain the Visualizations that is produced in the file. Don't clear them away.
5. Submit only the .IPYNB file. Intermediate files to be attached as mentioned in (2).
6. All the visuals should adhere to the visualization principles learnt in the Course and must be presentation ready. Most effective visuals would fetch maximum credits
7. Submissions done via means other than CANAVAS will strictly be NOT graded.

Group No: (mention your group number here)

Full Name	BITS ID
Deepak Kajla	2023cs04003
Harsha K	2023cs04018
Sahitya Srinivasan	2023cs04028
Santhosh Bhat	2023cs04041

Objective

To find best players from each positions with their age, nationality, club based on their Potential Scores

Download and Prep the Data: 1 Marks

Import the libraries needed

In [1]:
```
# Check for required libraries for visualisation!
# Alternatively, output can also be seen on git link directly: https://github.com/HarshaK1997/SEM2-DV/blob/master/Part2/dvi-Python-ps1.ipynb

# ! pip install seaborn
# ! pip install plotly
```

In [2]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio
import warnings

# # If you are running on vscode, use renderer as vscode. (OOPS moment: "Run all cells" again, if first run didn't give the graph)
# renderer='vscode'

# # If you are running on colab, use renderer as colab. (OOPS moment: "Run all cells" again, if first run didn't give the graph)
# renderer='colab'

# If you wanted to create html, use renderer as notebook. (OOPS moment: "Run all cells" again, if first run didn't give the graph)
renderer='notebook'

pio.renderers.default=renderer
warnings.filterwarnings("ignore")
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
```

Load data and store in dataframe

In [3]:

```python
# Load the dataframe from a CSV file
# We used the Housing dataset for this assignment which is attached for reference.
df = pd.read_csv('housing_data_set.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8129 entries, 0 to 8128
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   8129 non-null   object 
 1   price_tnd            6421 non-null   float64
 2   price_eur            6421 non-null   float64
 3   location             8129 non-null   object 
 4   city                 6813 non-null   object 
 5   governorate          8129 non-null   object 
 6   Area                 7944 non-null   float64
 7   pieces               6940 non-null   float64
 8   room                 7720 non-null   float64
 9   bathroom             7470 non-null   float64
 10  age                  3984 non-null   object 
 11  state                7029 non-null   float64
 12  latt                 8094 non-null   float64
 13  long                 8094 non-null   float64
 14  distance_to_capital  8094 non-null   float64
 15  garage               8129 non-null   int64  
 16  garden               8129 non-null   int64  
 17  concierge            8129 non-null   int64  
 18  beach_view           8129 non-null   int64  
 19  mountain_view        8129 non-null   int64  
 20  pool                 8129 non-null   int64  
 21  elevator             8129 non-null   int64  
 22  furnished            8129 non-null   int64  
 23  equipped_kitchen     8129 non-null   int64  
 24  central_heating      8129 non-null   int64  
 25  air_conditioning     8129 non-null   int64  
dtypes: float64(10), int64(11), object(5)
memory usage: 1.6+ MB
```

Find out what type of variable you are dealing with. This will help you find the right visualization method for that variable.

In [4]:
```
### Data Exploration Functions ###

# Get the number of rows and columns in the data set
def display_dataframe_shape(df):
    rows, columns = df.shape
    print("There are {} rows and {} columns in the data set".format(rows, columns))
```

In [5]:
```
# Explore the data set variables

# Display the first 5 rows of the dataframe
display(df.head())
```

	id	price_tnd	price_eur	location	city	governorate	Area	pieces	room	bathroom	age	state	latt	long	distance_to_capital	garage	garden	concierge	beach_view	mountain_view	pool	elevator	furnished	equipped_kitchen	central_heating	air_conditi
0	b9e1c759-d149-46e8-9765-d8c198a13ff0	NaN	NaN	Cité El Bassatine Ancien	Boumhel Bassatine	Ben Arous	NaN	27.0	8.0	NaN	NaN	NaN	36.577240	10.34246	30.815266	0	0	0	0	0	0	0	0	0	0	0
1	863e62e5-0bfe-49f3-ad97-e0ae91be68e9	3250000.0	1007500.0	El Kantaoui	Hammam Sousse	Sousse	1000.0	26.0	16.0	14.0	30-50	1.0	35.898175	10.58025	108.792932	1	0	1	1	0	0	1	1	1	1	1
2	0048e6da-9aec-4ebe-8ee1-1ad7cd0015e6	2000000.0	620000.0	Sousse Corniche	Sousse Ville	Sousse	932.0	24.0	24.0	10.0	NaN	1.0	35.827291	10.63390	118.317747	0	0	0	0	0	0	0	1	1	1	1
3	032f818f-1b38-4d1a-a000-753e235ccf54	2000000.0	620000.0	Sousse Corniche	Sousse Ville	Sousse	932.0	24.0	24.0	NaN	NaN	NaN	35.827300	10.63390	118.316886	0	0	0	0	0	0	0	1	1	1	1
4	2272576f-fb3b-4c82-8a0e-a00fe2e7c154	2000000.0	620000.0	Sousse Corniche	Sousse Ville	Sousse	932.0	24.0	24.0	10.0	NaN	1.0	35.827300	10.63390	118.316886	0	0	0	0	0	0	0	1	1	1	1

In [6]:
```python
# Display the last 5 rows of the dataframe
display(df.tail())
```

	id	price_tnd	price_eur	location	city	governorate	Area	pieces	room	bathroom	age	state	latt	long	distance_to_capital	garage	garden	concierge	beach_view	mountain_view	pool	elevator	furnished	equipped_kitchen	central_heating	air_conditi
8124	d6533c0a-666a-4fe4-8c20-6a97561bfb24	NaN	NaN	Sahloul	Sousse Ville	Sousse	NaN	NaN	2.0	2.0	0	2.0	35.830000	10.620000	117.401561	1	0	0	0	0	0	1	0	1	1	1
8125	3c3ac15d-12a9-46bb-9227-8c2c98de4007	NaN	NaN	El Kantaoui	Hammam Sousse	Sousse	NaN	NaN	NaN	NaN	NaN	NaN	35.856900	10.597200	113.681036	0	0	0	0	0	1	1	0	0	0	0
8126	d3b30b43-9377-45c1-8e22-626f11a29638	NaN	NaN	Carthage	Carthage	tunis	NaN	NaN	NaN	3.0	NaN	NaN	36.869432	10.316405	16.505765	1	0	0	0	0	0	1	0	0	1	1
8127	a970dfc9-4d39-4d7e-bd10-0fdcfc27ab04	NaN	NaN	La Marsa	NaN	tunis	NaN	NaN	NaN	1.0	NaN	NaN	36.876389	10.325278	17.722190	0	0	0	0	0	0	0	0	0	0	0
8128	a9ee278d-7e1c-4e2c-a3f3-d54c40ca2772	NaN	NaN	Les Berges Du Lac 2	La Marsa	tunis	NaN	NaN	3.0	NaN	NaN	2.0	36.847471	10.270151	10.829218	0	0	0	0	0	1	0	0	0	0	0

In [7]:

```
# Display the shape of the dataframe
display_dataframe_shape(df)

# Display the data types of each column
display(df.dtypes)
```

```
There are 8129 rows and 26 columns in the data set
```

```
id                        object
price_tnd                float64
price_eur                float64
location                  object
city                      object
governorate               object
Area                     float64
pieces                   float64
room                     float64
bathroom                 float64
age                       object
state                    float64
latt                     float64
long                     float64
distance_to_capital      float64
garage                     int64
garden                     int64
concierge                  int64
beach_view                 int64
mountain_view              int64
pool                       int64
elevator                   int64
furnished                  int64
equipped_kitchen           int64
central_heating            int64
air_conditioning           int64
dtype: object
```

Visualisation Questions - 2 X 5 = 10 Marks

Question 1

Fill the missing value for the continous variables with Mean(average) for proper data visualization

Preprocess height - convert data in format xx'xx to xx.xx Remove "nan" with Mode and convert the column to numerical

Preprocess weight - convert data in format xxlbs to xx Remove "nan" with Mode and convert the column to numerical

Do Univariate anlaysis for outliers detection for height and weight

Write the python code in the below cell to create appropriate visual to perform the above task.

Answer in markdown cells below the visual

1. Summarise your findings from the visual
2. The reason for selecting the chart type you did
3. Mention the pre-attentive attributes used.(atleast 2)
4. Mention the gestalt principles used.(atleast 2)

In [8]:
```
#determine the numerical features and describe them
df.describe()
```

Out[8]:

	price_tnd	price_eur	Area	pieces	room	bathroom	state	latt	long	distance_to_capital	garage	garden	concierge	beach_view	mountain_view	pool	elevator	furnished	equipped_kitchen	central_heating	air_conditioning
count	6.421000e+03	6.421000e+03	7934.000000	6940.000000	7020.000000	7070.000000	7029.000000	8094.000000	8094.000000	8094.000000	8129.000000	8129.0	8129.000000	8129.000000	8129.000000	8129.000000	8129.000000	8129.000000	8129.000000	8129.000000	8129.000000
mean	6.812867e+05	2.112588e+05	369.742954	4.318300	3.325389	1.997724	1.386826	36.421537	10.396046	65.495742	0.471276	0.0	0.245295	0.104072	0.082544	0.231886	0.262271	0.048099	0.568582	0.558248	0.550129
std	8.825321e+05	2.737262e+05	955.113562	2.270759	2.020898	1.237548	0.574206	1.144072	0.409603	126.346464	0.499205	0.0	0.430288	0.305373	0.275209	0.422062	0.439896	0.213989	0.495305	0.496626	0.497511
min	6.500000e+03	2.015000e+03	20.000000	1.000000	1.000000	1.000000	0.000000	0.000000	-0.428052	0.003560	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.600000e+05	8.060000e+04	117.000000	3.000000	2.000000	1.000000	1.000000	36.401080	10.195560	11.345082	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	4.200000e+05	1.302000e+05	178.000000	4.000000	3.000000	2.000000	1.000000	36.818810	10.325278	17.722190	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000
75%	7.800000e+05	2.418000e+05	371.000000	5.000000	4.000000	2.000000	2.000000	36.876389	10.614570	72.825068	1.000000	0.0	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000
max	1.900000e+07	5.890000e+06	26600.000000	27.000000	50.000000	20.000000	2.000000	40.565098	11.331493	4224.947771	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [9]:

```
# Drop garden as all records have only single value
df.drop('garden', axis=1, inplace=True)
```

In [10]:

```
df.describe().columns
```

Out[10]:

```
Index(['price_tnd', 'price_eur', 'Area', 'pieces', 'room', 'bathroom', 'state',
       'latt', 'long', 'distance_to_capital', 'garage', 'concierge',
       'beach_view', 'mountain_view', 'pool', 'elevator', 'furnished',
       'equipped_kitchen', 'central_heating', 'air_conditioning'],
      dtype='object')
```

In [11]:

```
#leftover features are categorical
categorical_cols = df.columns.difference(df.describe().columns).to_list()
print(categorical_cols)
for col in categorical_cols:
    counts = df[col].value_counts(dropna=False).reset_index()
    counts.columns = [col, "count"]
    percentages = df[col].value_counts(dropna=False, normalize=True).reset_index()
    percentages.columns = [col, "proportion"]
    result = counts.merge(percentages, on=col)
    result["proportion"] = result["proportion"].apply(lambda x: x * 100)
    print(f"{col}: Top 5 and bottom 5 records")
    print(pd.concat([result.head(5), result.tail(5)]).drop_duplicates())
    print("\n")
```

```
['age', 'city', 'governorate', 'id', 'location']
age: Top 5 and bottom 5 records
             age  count  proportion
0            NaN   4145   50.990282
1              0   1471   18.095707
2            1-5    826   10.161151
3           5-10    671    8.254398
4          10-20    557    6.852011
6          10,20    124    1.525403
7          30-50    104    1.279370
8          50-70     22    0.270636
9   Plus de 100       9    0.110715
10        70-100      7    0.086111


city: Top 5 and bottom 5 records
                city  count  proportion
0                NaN   1316   16.188953
1           La Marsa   1095   13.470292
2           Hammamet   1093   13.445688
3          La Soukra    628    7.725427
4       Ariana Ville    553    6.802805
66              Douz      2    0.024603
67      Borj El Amri      1    0.012302
68  Kalaat Landalous      1    0.012302
69          Tebourba      1    0.012302
70         El Battan      1    0.012302


governorate: Top 5 and bottom 5 records
     governorate  count  proportion
0          tunis   2303   28.330668
1         Nabeul   1884   23.176282
2         Ariana   1440   17.714356
3      Ben Arous    721    8.869480
4         Sousse    670    8.242096
18     Tataouine      9    0.110715
19       Siliana      9    0.110715
20         Gafsa      7    0.086111
21        Le Kef      1    0.012302
22  El Kasserine      1    0.012302


id: Top 5 and bottom 5 records
                                        id  count  proportion
0     b9e1c759-d149-46e8-9765-d8c198a13ff0      1    0.012302
1     ecb224f7-a95b-4e1d-8a96-72ecb3c1a64f      1    0.012302
2     21a72c14-cf23-4d25-b96b-91bc88c92347      1    0.012302
3     2e1aa417-568e-4ad4-98a0-d9ce3c32ed90      1    0.012302
4     b7966e73-37e9-427c-9476-9fb5ec33cc80      1    0.012302
8124  3768197f-d7b3-407d-b18a-e63ccff1eeb9      1    0.012302
8125  a273b3b4-a0fd-4920-ba14-d059e53898a0      1    0.012302
8126  b91803c9-3c4c-4985-91a2-f80f7d13b496      1    0.012302
8127  c11cf606-4a88-4b9c-8dcc-d3a62e69e1f3      1    0.012302
8128  a9ee278d-7e1c-4e2c-a3f3-d54c40ca2772      1    0.012302


location: Top 5 and bottom 5 records
                    location  count  proportion
0                   Hammamet    507    6.236930
```

```
1                  La Soukra    374    4.600812
2               Hammamet Nord    349    4.293271
3     Les Jardins de Carthage    283    3.481363
4                    La Marsa    266    3.272235
463             El Halfaouine      1    0.012302
464          Kalaat Landalous      1    0.012302
465            Cité Ghouzaila      1    0.012302
466               Sidi Abbes       1    0.012302
467               Hedi Chaker      1    0.012302
```

In [12]:
```
# Delete "id" columns as its unique for each row and does not provide any useful information
df.drop('id', axis=1, inplace=True)
categorical_cols.remove('id')
```

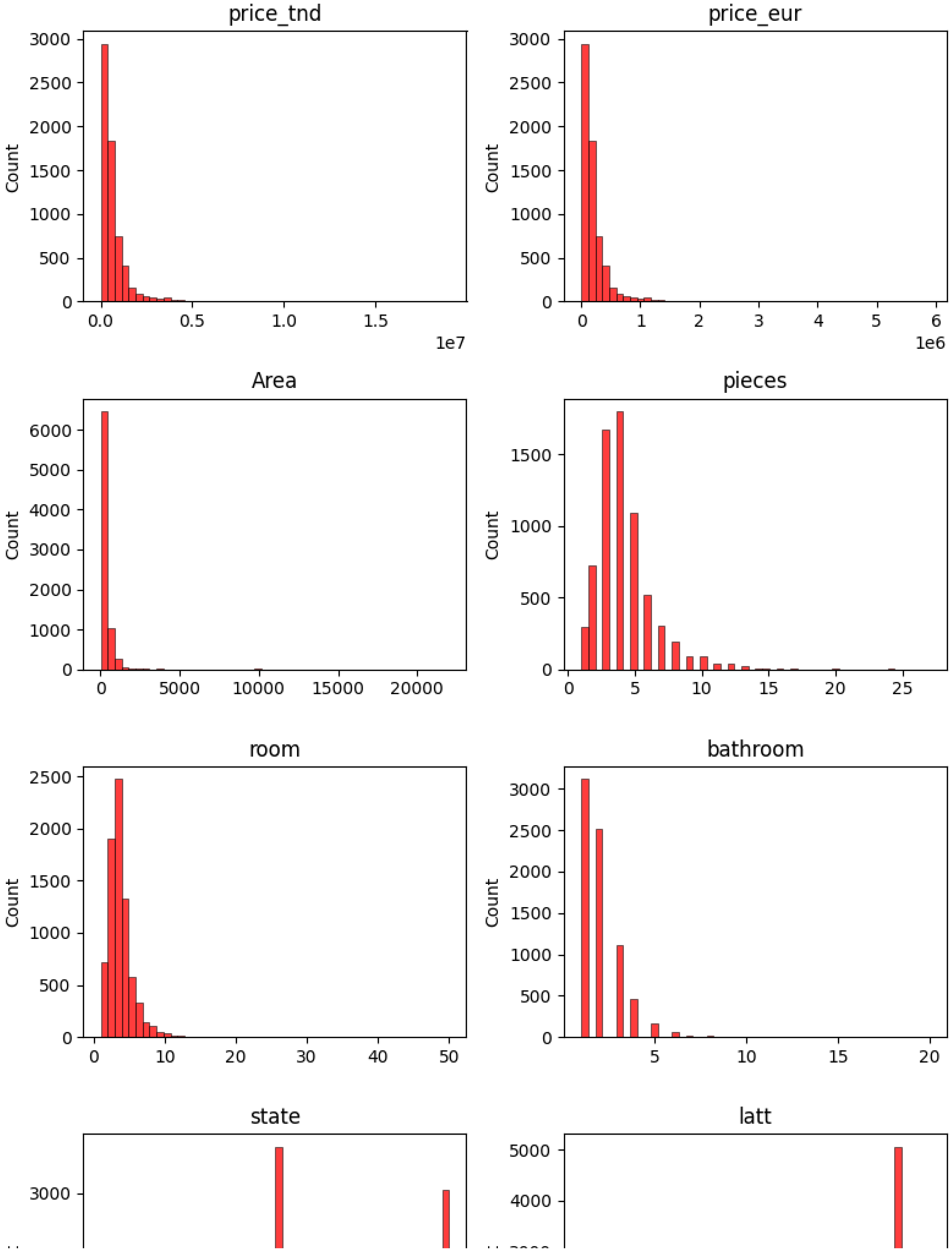
In [13]:
```
df.columns
categorical_cols
```

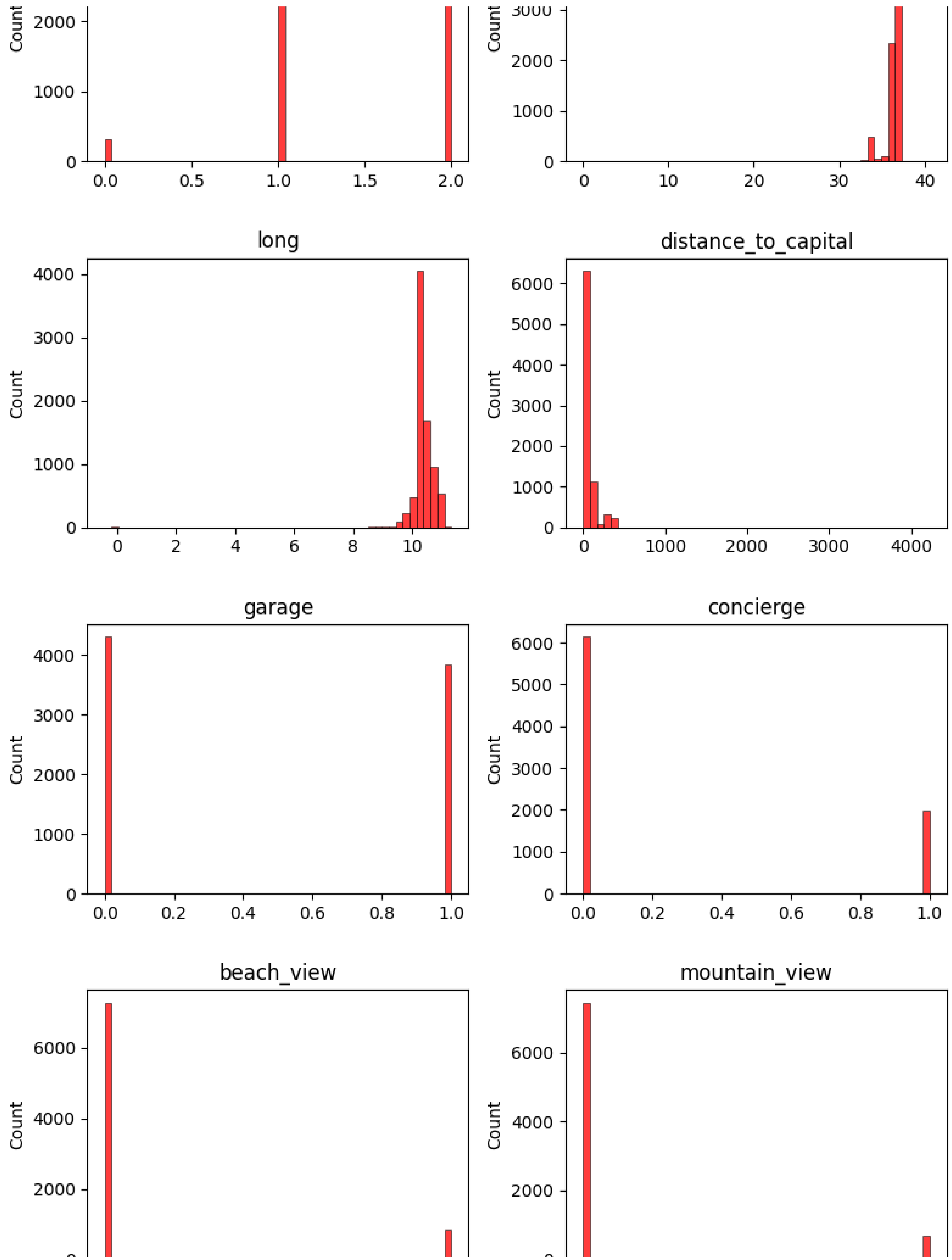
Out[13]:
```
['age', 'city', 'governorate', 'location']
```

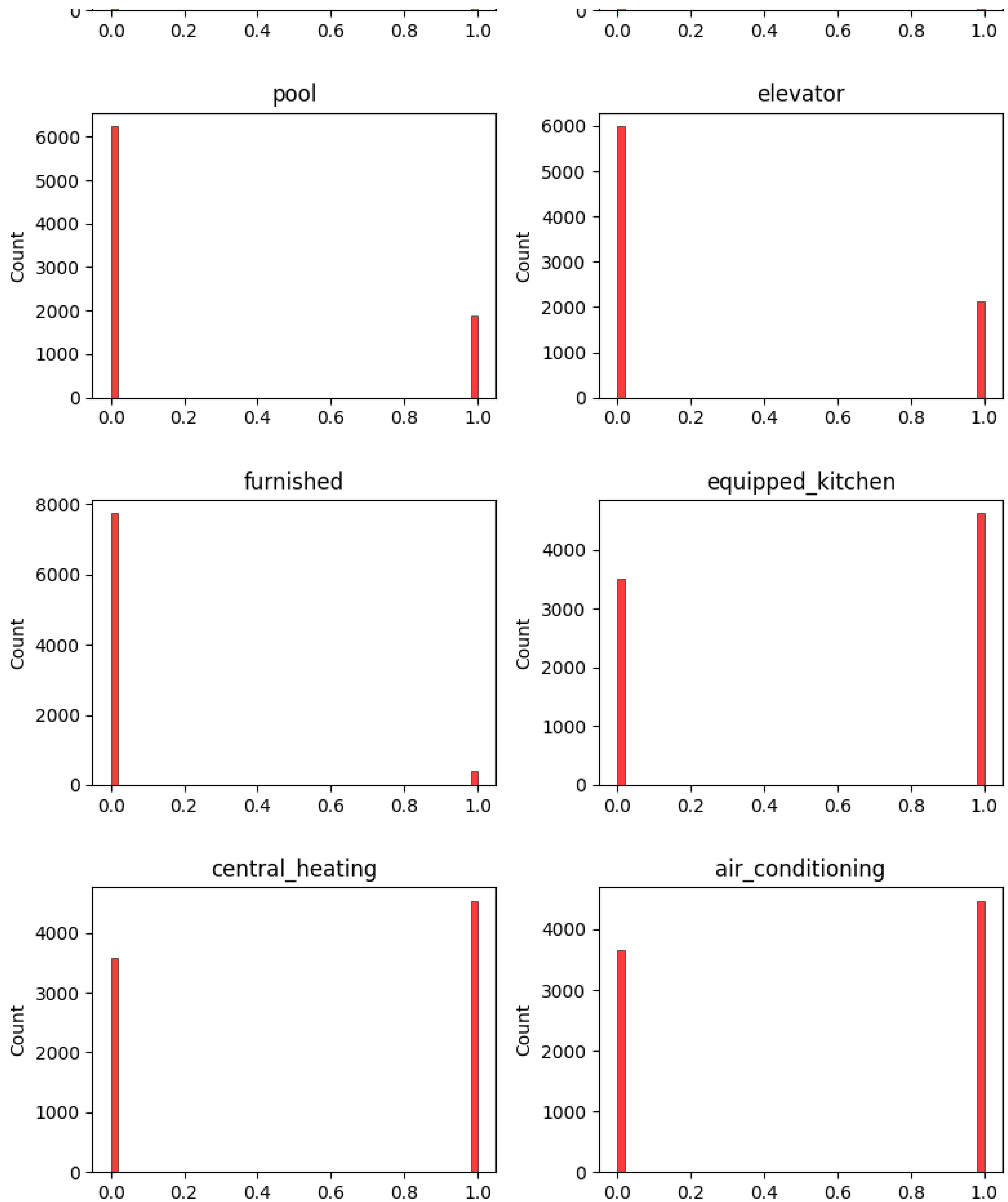
In [14]:
```
categorical_cols = ['location','city','governorate']
ordinal_cols = ['age']
numerical_cols = df.describe().columns.to_list()
print(categorical_cols)
print(ordinal_cols)
print(numerical_cols)
```

```
['location', 'city', 'governorate']
['age']
['price_tnd', 'price_eur', 'Area', 'pieces', 'room', 'bathroom', 'state', 'latt', 'long', 'distance_to_capital', 'garage', 'concierge', 'beach_view', 'mountain_view', 'pool',
'elevator', 'furnished', 'equipped_kitchen', 'central_heating', 'air_conditioning']
```

In [15]:
```
# visualize distribution of numerical columns
fig, ax = plt.subplots(int(np.ceil(len(numerical_cols)/2)), 2, figsize=(8, 30))
ax = ax.flatten()
# Iterate over the numerical columns and corresponding axes
for i, col in enumerate(numerical_cols):
    sns.histplot(df[col], bins=50, ax=ax[i], color='red')
    ax[i].set_title(col)
    ax[i].set_xlabel('')

# Remove any empty subplots (if number of subplots is more than columns)
for j in range(i+1, len(ax)):
    fig.delaxes(ax[j])

fig.tight_layout()
fig.show()
```


price_tnd
Count
3000
2500
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
1e7
price_eur
Count
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
1e6
Area
Count
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
pieces
Count
1500
1000
500
0
0
5
10
15
20
25
room
Count
2500
2000
1500
1000
500
0
0
10
20
30
40
50
bathroom
Count
3000
2500
2000
1500
1000
500
0
5
10
15
20
state
3000
latt
5000
4000


long
distance_to_capital
garage
concierge
beach_view
mountain_view
Count



Interpretations:

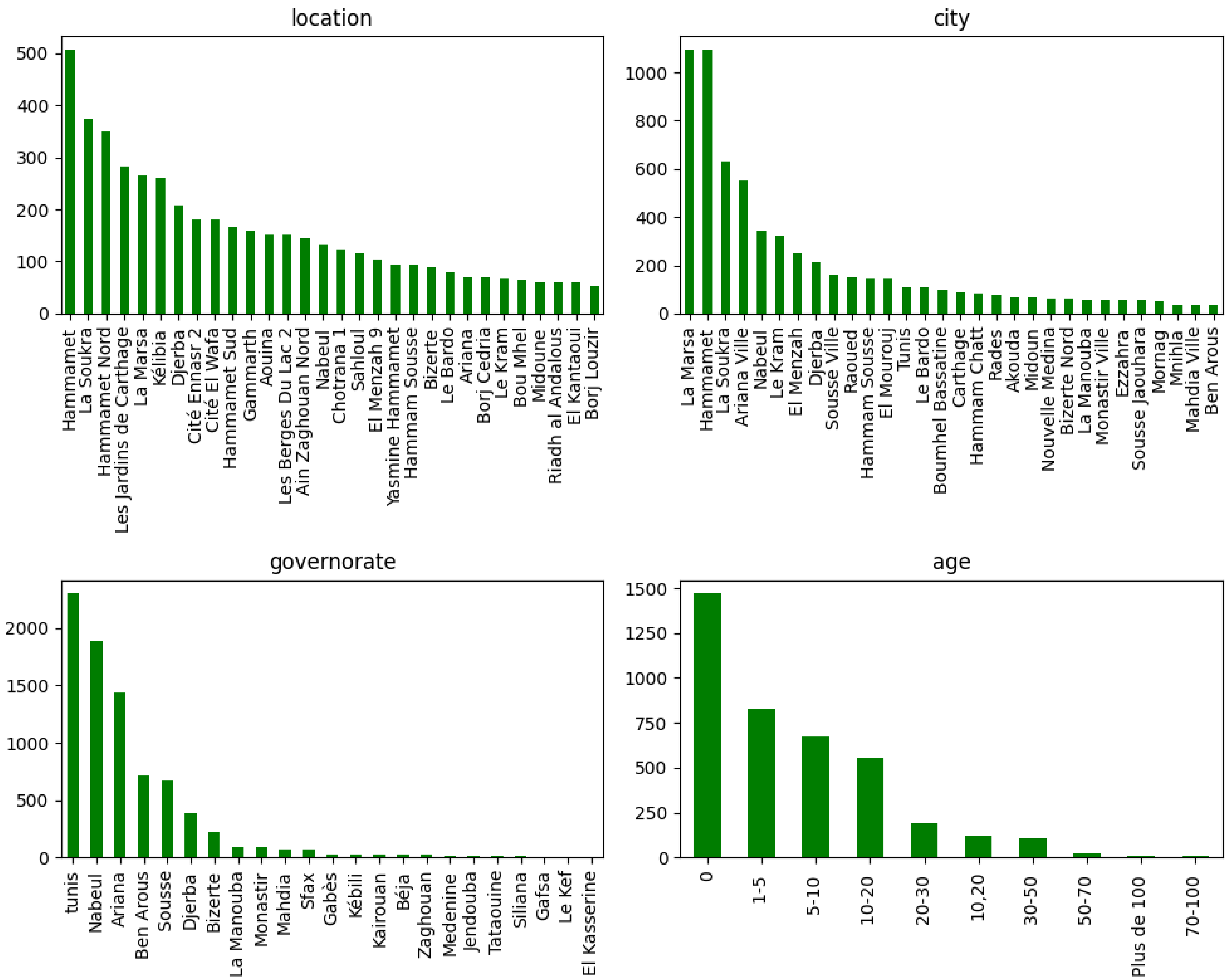
1. Summarise your findings from the visual

- garage, concierge, beach_view, mountain_view, pool, elevator, furnished, equipped_kitchen, central_heating and air_conditioning are boolean values (0 or 1)
- price_eur, price_tnd, area, pieces, room, bathroom and distance_to_capital are right-skewed while latt and long are left-skewed

2. The reason for selecting the chart type you did
    - Since we want to see the distribution of numerical data, we chose histogram chart

In [16]:

```
# visualize distribution of nominal and ordinal categorical columns
display_cols = categorical_cols + ordinal_cols
fig, ax = plt.subplots(int(np.ceil(len(display_cols)/2)), 2, figsize=(10, 8))
ax = ax.flatten()
# Iterate over the numerical columns and corresponding axes
for i, col in enumerate(display_cols):
    df[col].value_counts().sort_values(ascending=False).head(30).plot(kind='bar', ax=ax[i], color='green')
    ax[i].set_title(col)
    ax[i].set_xlabel('')

# Remove any empty subplots (if number of subplots is more than columns)
for j in range(i+1, len(ax)):
    fig.delaxes(ax[j])

fig.tight_layout()
fig.show()
```



Interpretations:

1. Summarise your findings from the visual
    - location and city have data spread across large number of values
    - age had ordinal data, and can be converted to numerical also
2. The reason for selecting the chart type you did
    - Since we want to see the distribution of categorical data, we chose bar chart

In [17]:
```
### Data Preprocessing ###
```

```python
# Drop duplicates
df.drop_duplicates(inplace=True)
display_dataframe_shape(df)
```

```
There are 7971 rows and 24 columns in the data set
```

In [18]:
```python
### Data Cleaning ###

# Strip the leading and trailing whitespaces from the columns
# also convert to lower case for uniformity
df = df.map(lambda x: x.strip().lower() if isinstance(x, str) else x)
df.head()
```

Out[18]:

	price_tnd	price_eur	location	city	governorate	Area	pieces	room	bathroom	age	state	latt	long	distance_to_capital	garage	concierge	beach_view	mountain_view	pool	elevator	furnished	equipped_kitchen	central_heating	air_conditioning
0	NaN	NaN	cité el bassatine ancien	boumhel bassatine	ben arous	NaN	27.0	8.0	NaN	NaN	NaN	36.577240	10.342463	0.815266	0	0	0	0	0	0	0	0	0	0
1	3250000.0	1007500.0	el kantaoui	hammam sousse	sousse	1000.0	26.0	16.0	14.0	30-50	1.0	35.898175	10.580251	108.792932	1	1	1	0	0	1	1	1	1	1
2	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	10.0	NaN	1.0	35.827291	10.633901	118.317747	0	0	0	0	0	0	1	1	1	1
3	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	NaN	NaN	NaN	35.827300	10.633901	118.316886	0	0	0	0	0	0	1	1	1	1
4	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	10.0	NaN	1.0	35.827300	10.633901	118.316886	0	0	0	0	0	0	1	1	1	1

In [19]:
```python
# We can convert the ordinal value to numerical based on the mean of the ranges
# This is better than simple substitution with numbers as it retains more of the original information
print("\nage:")
unique_values = df['age'].unique()
mean_map = {
    '0': 0,
    '1-5': 3,
    '5-10': 7.5,
    '10-20': 15,
    '10,20': 15,
    '20-30': 25,
    '30-50': 40,
    '50-70': 60,
    '70-100': 85,
    'plus de 100': 100
}
print("Before conversion: ", unique_values)
df['mean_age'] = df['age'].map(mean_map)
unique_values = df['mean_age'].unique()
print("After conversion: ", unique_values)
df.drop('age', axis=1, inplace=True)
numerical_cols.append('mean_age')
df.head()
```

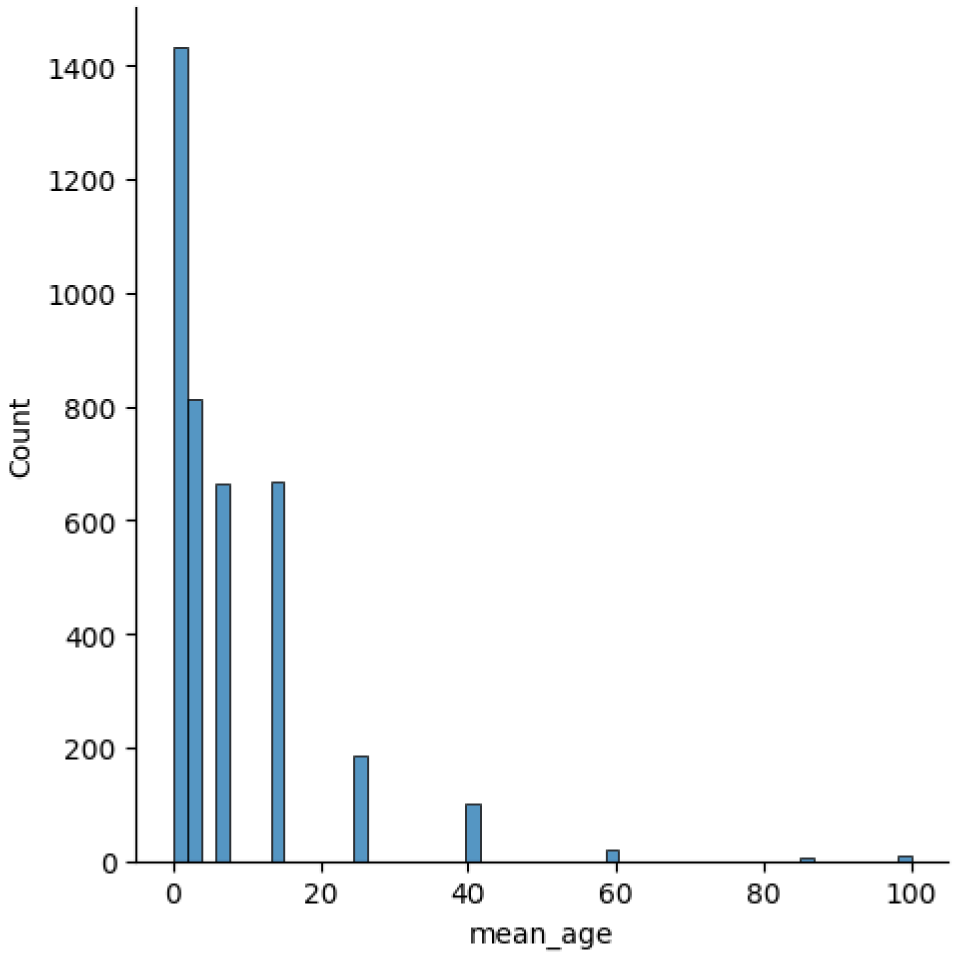
```
age:
Before conversion:  [nan '30-50' '1-5' '0' '5-10' '10-20' '20-30' '70-100' '10,20'
 'plus de 100' '50-70']
After conversion:  [  nan  40.    3.    0.    7.5  15.   25.   85.  100.   60. ]
```

Out[19]:

	price_tnd	price_eur	location	city	governorate	Area	pieces	room	bathroom	state	latt	long	distance_to_capital	garage	concierge	beach_view	mountain_view	pool	elevator	furnished	equipped_kitchen	central_heating	air_conditioning	mean_age
0	NaN	NaN	cité el bassatine ancien	boumhel bassatine	ben arous	NaN	27.0	8.0	NaN	NaN	36.577240	10.342463	0.815266	0	0	0	0	0	0	0	0	0	0	NaN
1	3250000.0	1007500.0	el kantaoui	hammam sousse	sousse	1000.0	26.0	16.0	14.0	1.0	35.898175	10.580251	08.792932	1	1	1	0	0	1	1	1	1	1	40.0
2	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	10.0	1.0	35.827291	10.633901	18.317747	0	0	0	0	0	0	1	1	1	1	NaN
3	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	NaN	NaN	35.827300	10.633901	18.316886	0	0	0	0	0	0	1	1	1	1	NaN
4	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	10.0	1.0	35.827300	10.633901	18.316886	0	0	0	0	0	0	1	1	1	1	NaN

In [20]:
```
sns.displot(df['mean_age'])
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x1557ff980>



In [21]:
```
# Check for missing values
display(df.isna().sum())
```

```
price_tnd              1668
price_eur              1668
location                  0
city                   1229
governorate               0
Area                    177
pieces                 1154
room                    406
bathroom                650
state                  1086
latt                     35
long                     35
distance_to_capital      35
garage                    0
concierge                 0
beach_view                0
mountain_view             0
pool                      0
elevator                  0
furnished                 0
equipped_kitchen          0
central_heating           0
air_conditioning          0
mean_age               4067
dtype: int64
```

In [22]:

```python
# Filling missing values: city column

before_processing = df['city'].isna().sum()
print("Initial number of missing values in city column: ", before_processing)

# Fetch all the location in each city, to fill missing values of city column more meaningfully.
location_city = {}
for local in df['location'].dropna().unique():
    d1 = df[df['location'] == local]['city'].dropna()
    if len(d1) > 0:
        location_city[local] = d1.mode()[0]

# Fill missing values in city column using location column
df.loc[df['city'].isna(), 'city'] = df[df['city'].isna()]['location'].map(location_city)

nan_values = df['city'].isna().sum()
filled_values = before_processing - nan_values

print("Number of city records filled smartly: ", filled_values)
print("Number of city records remaining which are still empty (will be removed): ", nan_values)

# Remove city records if the location is not available
df.dropna(subset=['city'], inplace=True)
```

```
Initial number of missing values in city column:  1229
Number of city records filled smartly:  404
Number of city records remaining which are still empty (will be removed):  825
```

In [23]:

```python
# Filling missing values: latt, long, distance_to_capital

cols = ['latt', 'long', 'distance_to_capital']

for col in cols:
    before_processing = df['city'].isna().sum()
```

```python
    if before_processing > 0:
        print(f"\nInitial number of missing values in {col} column: ", before_processing)

        # Fetch all the location in each city, to fill missing values of city column more meaningfully.
        location_map = {}
        for local in df['location'].dropna().unique():
            d1 = df[df['location'] == local][col].dropna()
            if len(d1) > 0:
                location_map[local] = d1.median()

        # Fill missing values in city column using location column
        df.loc[df[col].isna(), col] = df[df[col].isna()]['location'].map(location_map)

        nan_values = df[col].isna().sum()
        filled_values = before_processing - nan_values

        print(f"Number of {col} records filled smartly: ", filled_values)
        print(f"Number of {col} records remaining which are still empty (will be removed): ", nan_values)

        # Remove city records if the location is not available
        df.dropna(subset=[col], inplace=True)
    else:
        print(f"\nNo missing values in {col} column")
```

```
No missing values in latt column

No missing values in long column

No missing values in distance_to_capital column
```

In [24]:

```python
# price is target, so we drop na values
df.dropna(subset=['price_eur'], inplace=True)
df.shape
```

Out[24]: (5689, 24)

In [25]:

```python
# Filling remaining numerical values with median

for col in numerical_cols:
    df[col].fillna(df[col].median(), inplace=True)
# Check for missing values
display(df.isna().sum())
```

```
price_tnd              0
price_eur              0
location               0
city                   0
governorate            0
Area                   0
pieces                 0
room                   0
bathroom               0
state                  0
latt                   0
long                   0
distance_to_capital    0
garage                 0
concierge              0
beach_view             0
mountain_view          0
pool                   0
elevator               0
furnished              0
equipped_kitchen       0
central_heating        0
air_conditioning       0
mean_age               0
dtype: int64
```

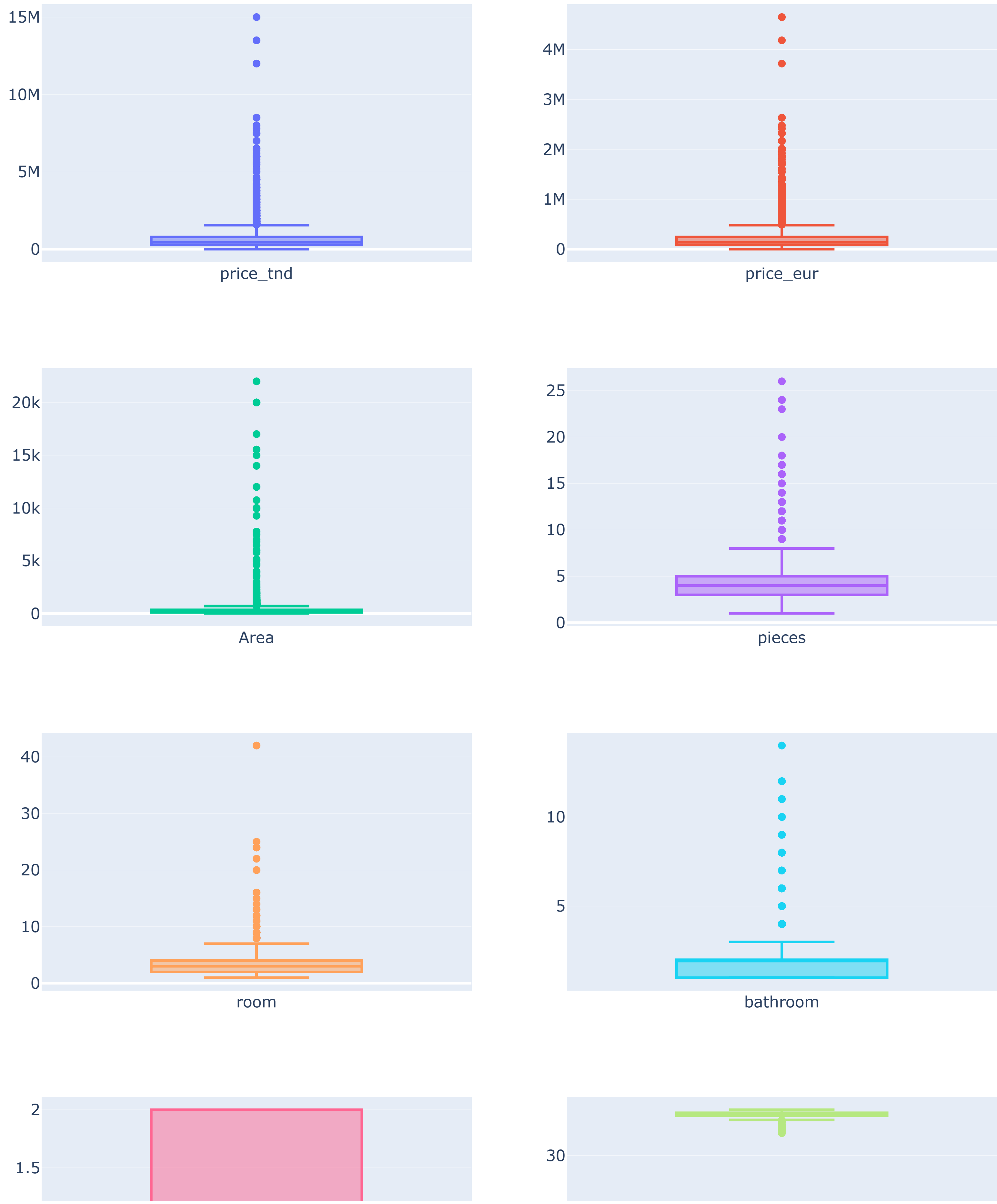
In [26]:

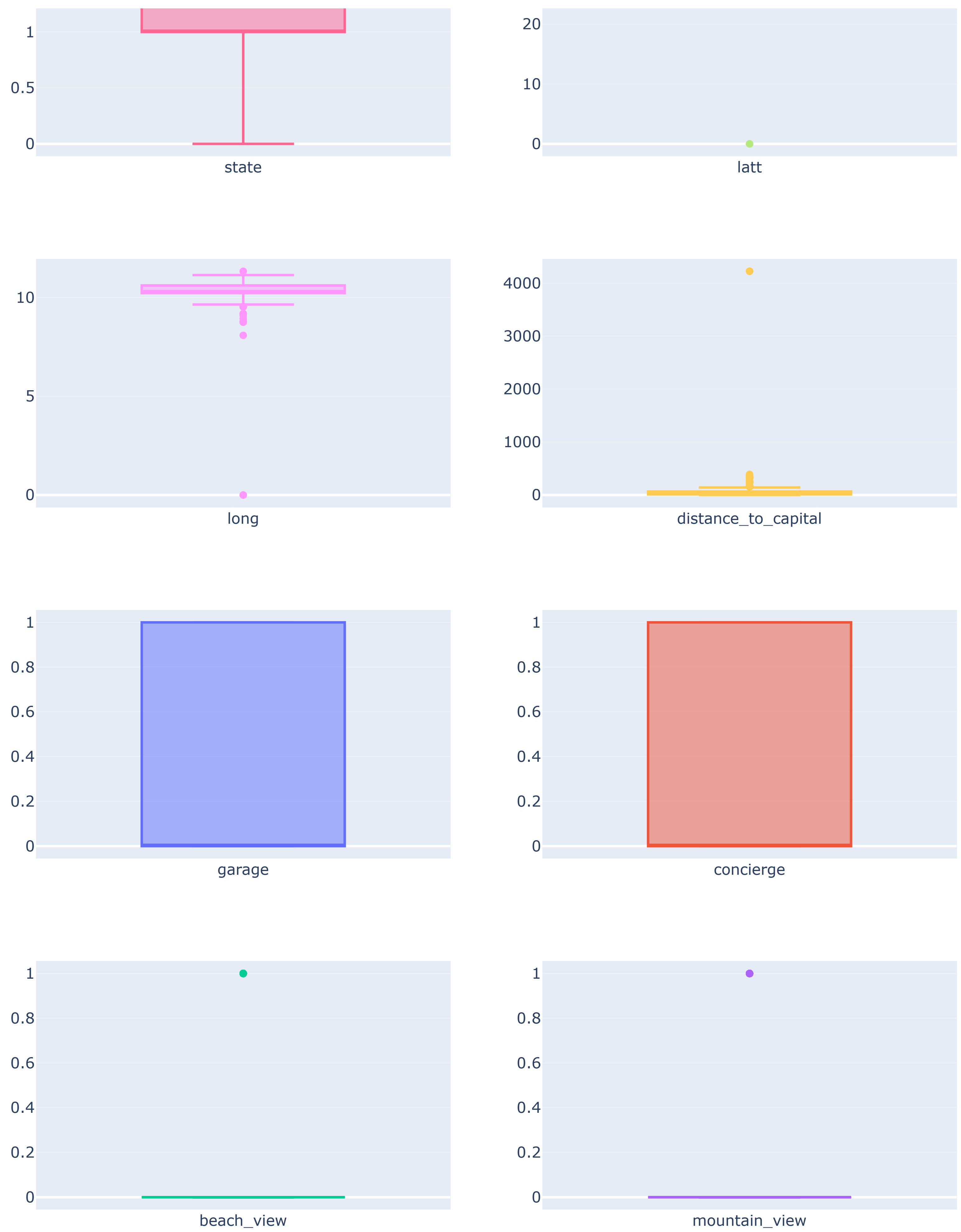
```
print("Total number of records before final duplicates removal: ", df.shape[0])

# Drop duplicates if any after filling missing values
df.drop_duplicates(inplace=True)

print("Total number of records after final duplicates removal: ", df.shape[0])

# Check for missing values after filling and converting
df.info()
```

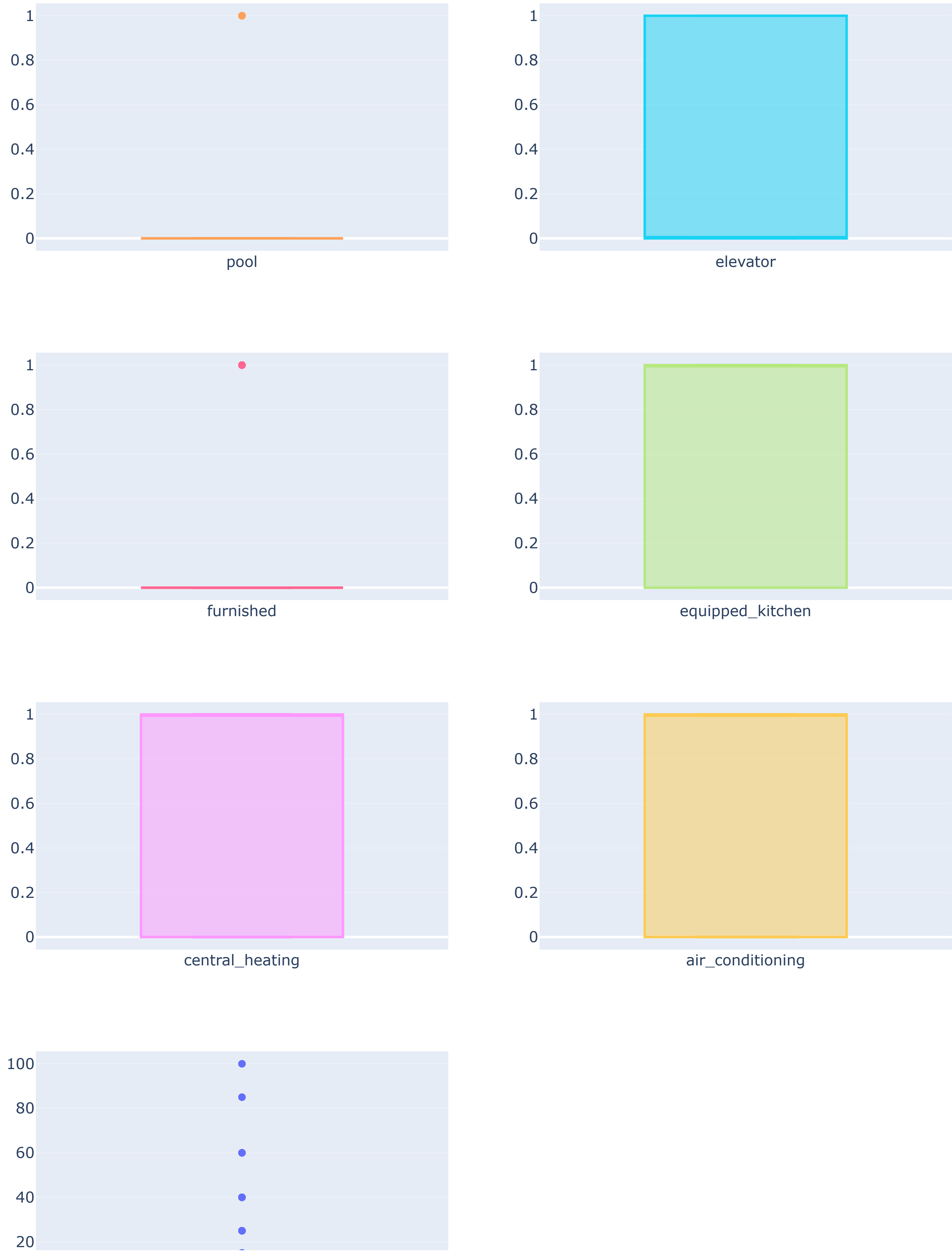
```
Total number of records before final duplicates removal:  5689
Total number of records after final duplicates removal:  5688
<class 'pandas.core.frame.DataFrame'>
Index: 5688 entries, 1 to 8104
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   price_tnd            5688 non-null   float64
 1   price_eur            5688 non-null   float64
 2   location             5688 non-null   object 
 3   city                 5688 non-null   object 
 4   governorate          5688 non-null   object 
 5   Area                 5688 non-null   float64
 6   pieces               5688 non-null   float64
 7   room                 5688 non-null   float64
 8   bathroom             5688 non-null   float64
 9   state                5688 non-null   float64
 10  latt                 5688 non-null   float64
 11  long                 5688 non-null   float64
 12  distance_to_capital  5688 non-null   float64
 13  garage               5688 non-null   int64  
 14  concierge            5688 non-null   int64  
 15  beach_view           5688 non-null   int64  
 16  mountain_view        5688 non-null   int64  
 17  pool                 5688 non-null   int64  
 18  elevator             5688 non-null   int64  
 19  furnished            5688 non-null   int64  
 20  equipped_kitchen     5688 non-null   int64  
 21  central_heating      5688 non-null   int64  
 22  air_conditioning     5688 non-null   int64  
 23  mean_age             5688 non-null   float64
dtypes: float64(11), int64(10), object(3)
memory usage: 1.1+ MB
```

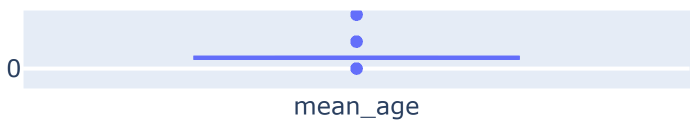
In [27]:

```
# checking numerical columns for outliers using boxplot
fig = go.Figure()
fig = make_subplots(rows=int(np.ceil(len(numerical_cols)/2)), cols=2)
for col in numerical_cols:
    r_num = int(numerical_cols.index(col)/2) + 1
    c_num = int(numerical_cols.index(col)%2) + 1
    fig.add_trace(go.Box(y=df[col], name=col, showlegend=False), row=r_num, col=c_num)
fig.update_layout(height=3200, width=900)
fig.show()
```


15M
10M
5M
0
price_tnd
4M
3M
2M
1M
0
price_eur
20k
15k
10k
5k
0
Area
25
20
15
10
5
0
pieces
40
30
20
10
0
room
10
5
bathroom
2
1.5
30


state
latt
long
distance_to_capital
garage
concierge
beach_view
mountain_view
0
0.5
1
10
20
5
1000
2000
3000
4000
0.2
0.4
0.6
0.8


1
0.8
0.6
0.4
0.2
0
pool
elevator
furnished
equipped_kitchen
central_heating
air_conditioning
100
80
60
40
20



Interpretations:

1. Summarise your findings from the visual
   - price_eur, price_tnd, area, pieces, room and bathroom have large number of records considered as outliers using IQR formula (box plot), but changing all these records can cause huge loss of information from data
   - latt, long and distance_to_capital are ideal for dealing with outlier values
2. The reason for selecting the chart type you did
   - Since we want to see the outliers in numerical data, we have used boxplot which shows IQR analysis
3. Mention the pre-attentive attributes used.(atleast 2)
   - Color: Using colour of chart to distinguish different charts in a close space
   - Position: The position of elements along the x-axis immediately conveys information about the distribution, such as where data is concentrated or how it is spread.
4. Mention the gestalt principles used.(atleast 2)
   - Proximity: Grouping elements that are close to each other helps viewers
   - Similarity: Using consistent styles, colors, or shapes for similar elements

In [28]:
```
# Function to replace outliers with the median
def replace_outliers_with_median(column):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)

    # Calculate IQR
    IQR = Q3 - Q1

    # Define the lower and upper bound for detecting outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Calculate the median
    median = column.median()

    # Replace outliers with the median
    return column.apply(lambda x: median if x < lower_bound or x > upper_bound else x)
```

In [29]:
```
outlier_cols = [
 'latt',
 'long',
 'distance_to_capital']

df[outlier_cols] = df[outlier_cols].apply(replace_outliers_with_median)
df.head()
```

Out[29]:

	price_tnd	price_eur	location	city	governorate	Area	pieces	room	bathroom	state	latt	long	distance_to_capital	garage	concierge	beach_view	mountain_view	pool	elevator	furnished	equipped_kitchen	central_heating	air_conditioning	mean_age
1	3250000.0	1007500.0	el kantaoui	hammam sousse	sousse	1000.0	26.0	16.0	14.0	1.0	35.898175	10.58025	108.792932	1	1	1	0	0	1	1	1	1	1	40.0
2	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	10.0	1.0	35.827291	10.63390	118.317747	0	0	0	0	0	0	1	1	1	1	3.0
3	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	2.0	1.0	35.827300	10.63390	118.316886	0	0	0	0	0	0	1	1	1	1	3.0
4	2000000.0	620000.0	sousse corniche	sousse ville	sousse	932.0	24.0	24.0	10.0	1.0	35.827300	10.63390	118.316886	0	0	0	0	0	0	1	1	1	1	3.0
5	1000000.0	310000.0	sousse riadh	sousse riadh	sousse	1000.0	23.0	16.0	9.0	2.0	35.814330	10.63392	119.603211	1	0	0	0	0	0	0	0	0	0	3.0

Question 2

Do Bi-Variate anlaysis for outliers detection for height and weight

Write the python code in the below cell to create appropriate visual to perform the above task.

Answer in markdown cells below the visual

1. Summarise your findings from the visual
2. The reason for selecting the chart type you did
3. Mention the pre-attentive attributes used.(atleast 2)
4. Mention the gestalt principles used.(atleast 2)

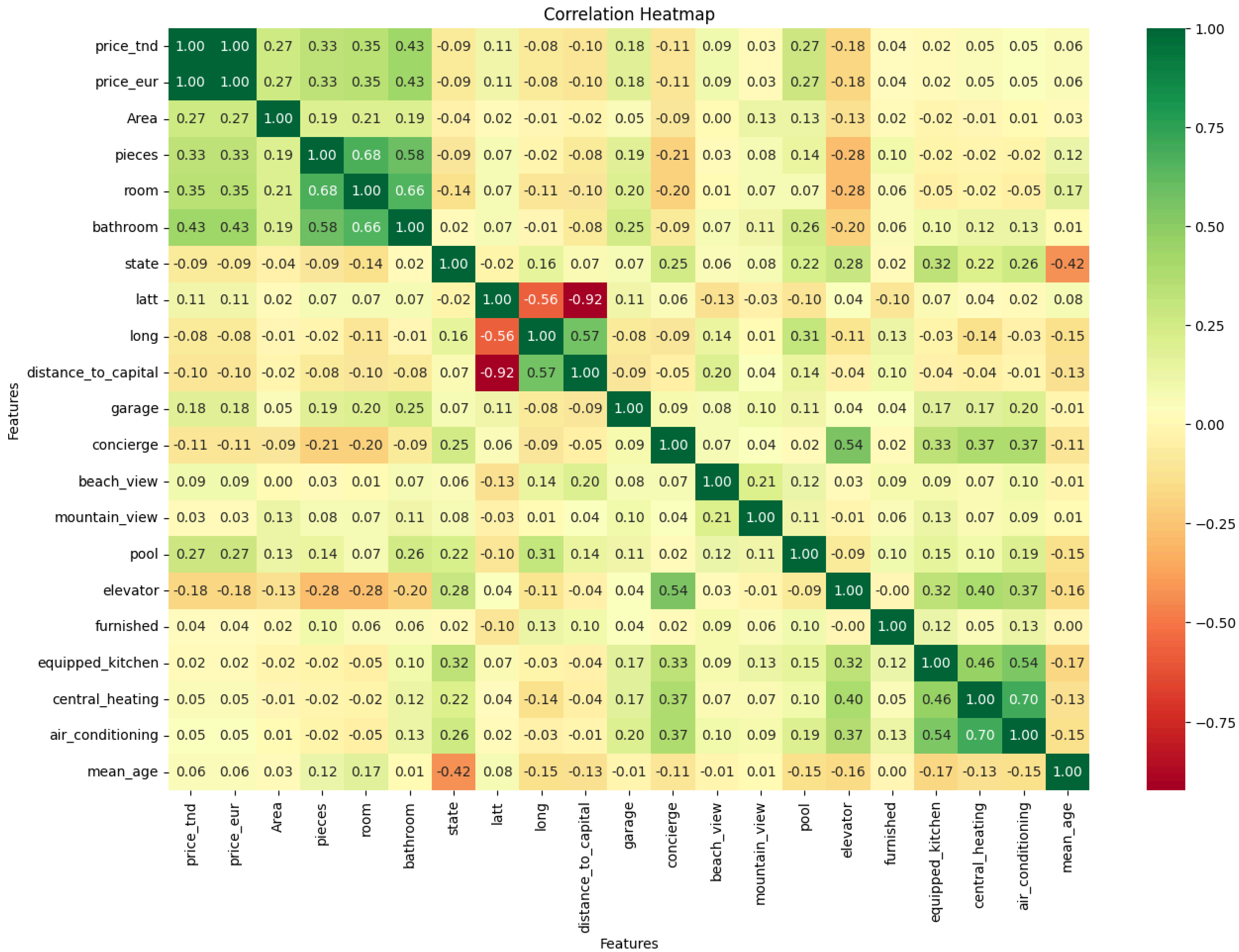
In [30]:

```
print("categorical columns:", categorical_cols)
print("numerical columns:", numerical_cols)
```

```
categorical columns: ['location', 'city', 'governorate']
numerical columns: ['price_tnd', 'price_eur', 'Area', 'pieces', 'room', 'bathroom', 'state', 'latt', 'long', 'distance_to_capital', 'garage', 'concierge', 'beach_view', 'mount
ain_view', 'pool', 'elevator', 'furnished', 'equipped_kitchen', 'central_heating', 'air_conditioning', 'mean_age']
```

In [31]:

```
# test for correlation between numerical columns
plt.figure(figsize=(15, 10))
corr = df[numerical_cols].corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap="RdYlGn")
plt.title("Correlation Heatmap")
plt.xlabel("Features")
plt.ylabel("Features")
plt.show()
```



Interpretations:

1. Summarise your findings from the visual
   - price_eur and price_tnd, central_heating and air_conditioning, room with bathroom, and room with pieces are highly correlated
   - distance_to_capital and latt are inversely correlated
2. The reason for selecting the chart type you did
   - Since we want to see correlation between 2 numerical features, we chose heatmap
3. Mention the pre-attentive attributes used.(atleast 2)
   - Color: Using colour of chart to highlight different parts of the chart
   - Position: By positioning the variables in a logical order, viewers can more easily interpret the correlations between related variables.
4. Mention the gestalt principles used.(atleast 2)
   - Continuity: A clear gradient from one color to another helps viewers easily understand the range of correlation values across the matrix.
   - Figure-Ground: By using contrasting colors or shades for the cells compared to the overall background of the plot, making it easier to focus on specific correlations.

In [32]:
```
# price_tnd and price_eur are correlated
df.drop('price_tnd', axis=1, inplace=True)
# latt is inversely correlated to distance_to_capital
df.drop('latt', axis=1, inplace=True)
# room is correlated to pieces and bathroom are
df.drop('bathroom', axis=1, inplace=True)
df.drop('pieces', axis=1, inplace=True)
# air_conditioning and central_heating are correlated
df.drop('air_conditioning', axis=1, inplace=True)
numerical_cols.remove('price_tnd')
numerical_cols.remove('latt')
numerical_cols.remove('bathroom')
numerical_cols.remove('pieces')
numerical_cols.remove('air_conditioning')
```

In [33]:
```
# test for correlation between categorical columns
from scipy.stats import chi2_contingency

results = []

# Iterate over all combinations of categorical columns
for i in range(len(categorical_cols)):
    for j in range(i + 1, len(categorical_cols)):
        cat_col1 = categorical_cols[i]
        cat_col2 = categorical_cols[j]

        # Create a contingency table
        contingency_table = pd.crosstab(df[cat_col1], df[cat_col2])

        # Perform the Chi-Square test
        chi2, p_value, dof, expected = chi2_contingency(contingency_table)

        # Store the results
        result = {
            "categorical_1": cat_col1,
            "categorical_2": cat_col2,
            "chi2": chi2,
            "p_value": p_value,
            "degrees_of_freedom": dof
        }
        results.append(result)

# Convert results to DataFrame for easier viewing
```

```
results_df = pd.DataFrame(results)
print(results_df)
```

```
  categorical_1 categorical_2           chi2  p_value  degrees_of_freedom
0      location          city  369344.305084      0.0               20636
1      location   governorate   81001.081002      0.0                4928
2          city   governorate   82042.918608      0.0                1072
```

In [34]:
```
# city, location and governorate are highly correlated (p < 0.05)
df.drop(['city', 'governorate'], axis=1, inplace=True)
categorical_cols.remove('city')
categorical_cols.remove('governorate')
```

In [35]:
```
# test for correlation between categorical and numerical columns
from scipy.stats import f_oneway

results = []

# Iterate over all combinations of categorical and numerical columns
for cat_col in categorical_cols:
    for num_col in numerical_cols:

        # Perform ANOVA
        groups = [df[num_col][df[cat_col] == cat] for cat in df[cat_col].unique()]
        f_statistic, p_value = f_oneway(*groups)

        result = {
            "categorical": cat_col,
            "numerical": num_col,
            "F-statistic": f_statistic,
            "p_value": p_value
        }
        results.append(result)

# Convert results to DataFrame for easier viewing
results_df = pd.DataFrame(results)
print(results_df)
```

```
   categorical            numerical  F-statistic        p_value
0     location            price_eur     6.286032  9.016327e-191
1     location                 Area    10.569026   0.000000e+00
2     location                 room     4.276339  3.483282e-107
3     location                state     5.181974  7.973623e-145
4     location                 long   480.254814   0.000000e+00
5     location  distance_to_capital   559.374609   0.000000e+00
6     location               garage     2.079821   1.968504e-23
7     location            concierge     3.981165   4.729040e-95
8     location           beach_view     3.727408   1.006256e-84
9     location        mountain_view     3.144282   1.490370e-61
10    location                 pool     6.423087  1.919545e-196
11    location             elevator     6.211928  1.061153e-187
12    location            furnished     2.023864   1.077187e-21
13    location     equipped_kitchen     3.695483   1.966083e-83
14    location      central_heating     4.919308  7.124259e-134
15    location             mean_age     8.609623  4.063989e-285
```

In [36]:
```
# Remove location as it is highly correlated to all numerical columns
df.drop('location', axis=1, inplace=True)
```

In [37]:
```
display_dataframe_shape(df)
df.drop_duplicates(inplace=True)
display_dataframe_shape(df)
```

```
There are 5688 rows and 16 columns in the data set
There are 5671 rows and 16 columns in the data set
```

Question 3

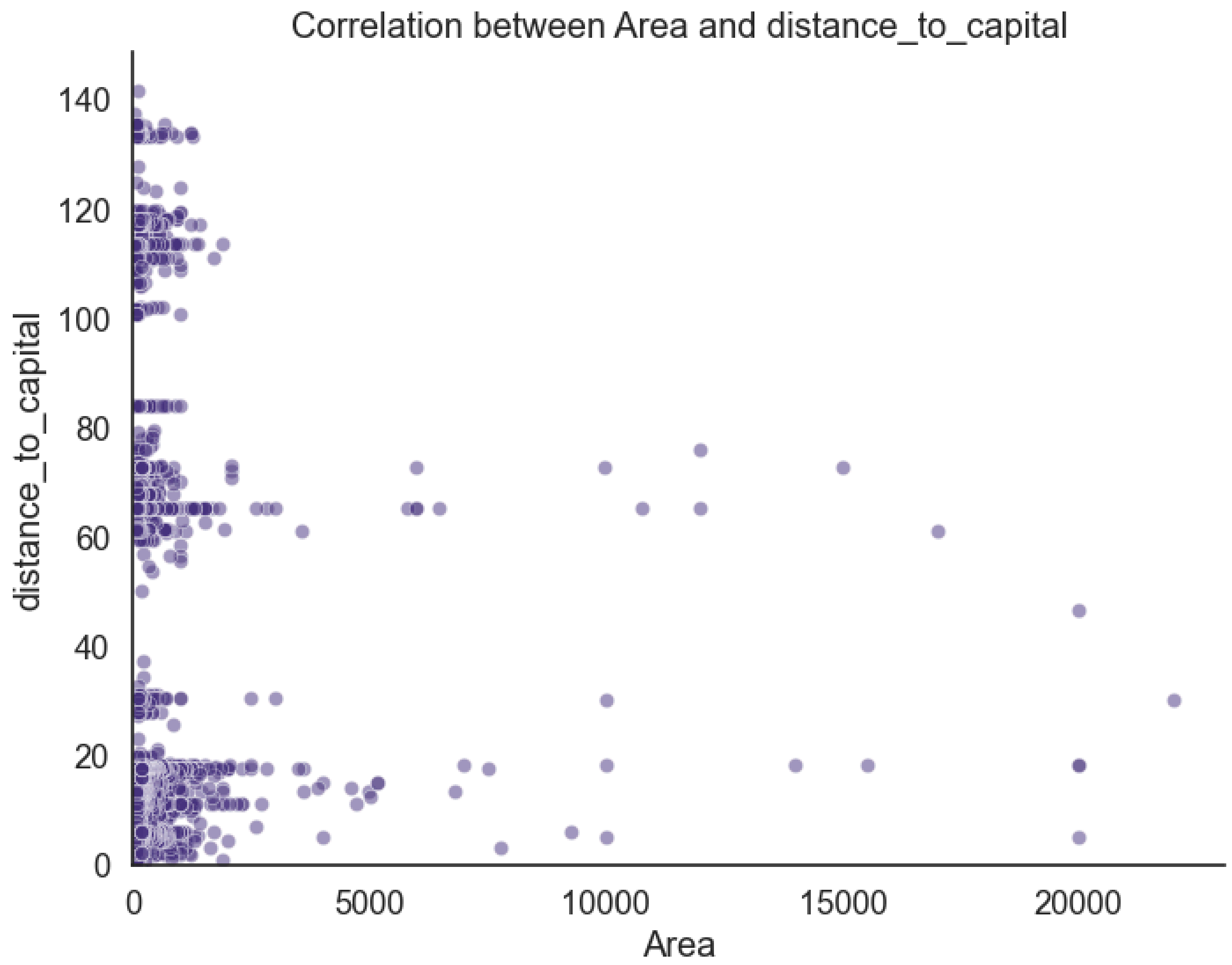
What kind of co-relation exists between Area and distance_to_capital

Write the python code in the below cell to create appropriate visual to perform the above task.

Answer in markdown cells below the visual

1. Summarise your findings from the visual
2. The reason for selecting the chart type you did
3. Mention the pre-attentive attributes used.(atleast 2)
4. Mention the gestalt principles used.(atleast 2)

In [38]:
```
# Scatter plot
sns.set(style="white", context="notebook", palette="viridis", font_scale=1.2)
fig, ax = plt.subplots(figsize=(8,6))
plt.title('Correlation between Area and distance_to_capital')
sns.scatterplot(data=df, x="Area", y="distance_to_capital", ax=ax, alpha=0.5)
sns.despine()
# Set axis limits to include the origin (0,0)
ax.set_xlim(left=0)
ax.set_ylim(bottom=0)

plt.show()
```



Interpretations:

1. Summarise your findings from the visual
   - All of the bigger Area housing locations are within 80 miles distance from its capital.
   - More than one-third of total smaller Area housing locations are within 40 miles distance from its capital.
   - More than 90% of the housing records has the Area less than 5000 square units.
2. The reason for selecting the chart type you did
   - We need to visualise two quantitative feature namely, 'Area' & 'distance_to_capital'.
   - Scatterplot is one of the best way to represent for this purpose.
3. Mention the pre-attentive attributes used.(atleast 2)
   - Visual variable color gradient is used to represent density of data variable records.
   - Visual variable position is used to represent (Area, distance_to_capital) data variable.
4. Mention the gestalt principles used.(atleast 2)
   - Gestalt Law of Proximity - 3 separate groups (0-40, 60-80, 100-140) can be visualised in the visual.
   - Gestalt Law of Closure - Although there is no frame around the plot, the axis lines and the labels are enough to define a closed space.

Question 4

What kind of relation exists between Age and (potential vs Overall). Create an appropriate visual to compare potential vs Overall with respect to age in one single visual.

Write the python code in the below cell to create appropriate visual to perform the above task.

Answer in markdown cells below the visual

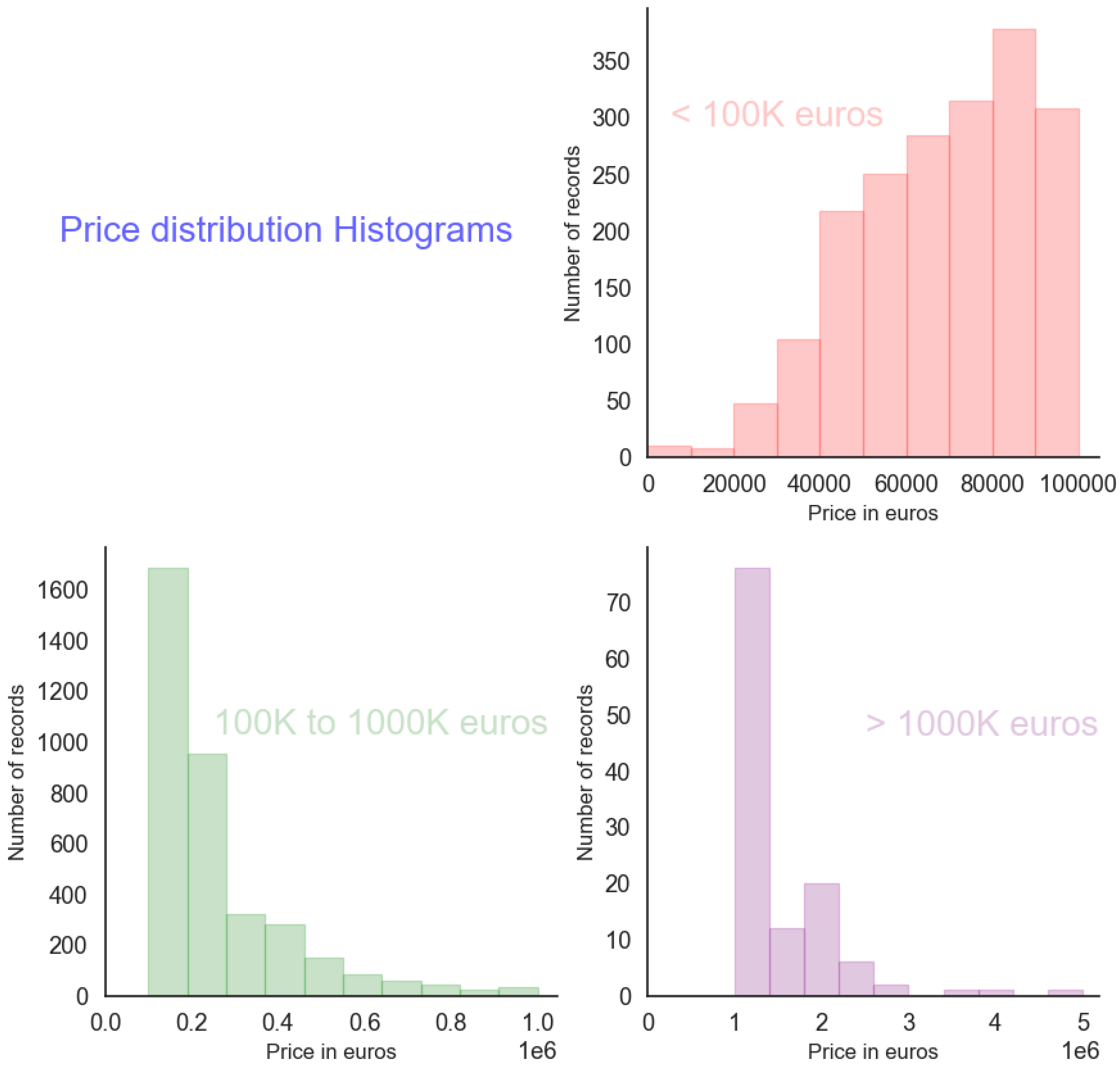
1. Summarise your findings from the visual
2. The reason for selecting the chart type you did
3. Mention the pre-attentive attributes used.(atleast 2)
4. Mention the gestalt principles used.(atleast 2)

In [39]:

```python
# Histogram:

fig = plt.figure(figsize=(10, 10))
fig.suptitle('Price Distribution Overview', fontsize=16)

# Subplot 1
ax0 = fig.add_subplot(221)
# Add text to the first subplot
ax0.text(0.4, 0.5, 'Price distribution Histograms',
         fontsize=20,
         color='blue',
         ha='center',
         va='center',
         alpha=0.6)
# Remove borders
for spine in ax0.spines.values():
    spine.set_visible(False)
# Remove x and y ticks
ax0.set_xticks([])
ax0.set_yticks([])

# Subplot 2
ax1 = fig.add_subplot(222)
ax1.hist(df['price_eur'],
         edgecolor='red',
         facecolor='red',
         bins=10,
         range=(0, 100000),
         alpha=0.2)
ax1.text(30000, 300, '< 100K euros',
         fontsize=20,
         color='red',
         ha='center',
         va='center',
         alpha=0.2)
ax1.set_xlabel('Price in euros', size=12)
ax1.set_ylabel('Number of records', size=12)
sns.despine(ax=ax1)
# Set axis limits to include the origin (0,0)
ax1.set_xlim(left=0)
ax1.set_ylim(bottom=0)

# Subplot 3
ax2 = fig.add_subplot(223)
ax2.hist(df['price_eur'],
         edgecolor='green',
         facecolor='green',
         bins=10,
         range=(100000, 1000000),
```

```
         alpha=0.2)
ax2.text(250000, 1000, '100K to 1000K euros',
         fontsize=20,
         color='green',
         ha='left',
         va='bottom',
         alpha=0.2)
ax2.set_xlabel('Price in euros', size=12)
ax2.set_ylabel('Number of records', size=12)
sns.despine(ax=ax2)
# Set axis limits to include the origin (0,0)
ax2.set_xlim(left=0)
ax2.set_ylim(bottom=0)

# Subplot 4
ax3 = fig.add_subplot(224)
ax3.hist(df['price_eur'],
         edgecolor='purple',
         facecolor='purple',
         bins=10,
         range=(1000000, 5000000),
         alpha=0.2)
ax3.text(2500000, 45, '> 1000K euros',
         fontsize=20,
         color='purple',
         ha='left',
         va='bottom',
         alpha=0.2)
ax3.set_xlabel('Price in euros', size=12)
ax3.set_ylabel('Number of records', size=12)
sns.despine(ax=ax3)
# Set axis limits to include the origin (0,0)
ax3.set_xlim(left=0)
ax3.set_ylim(bottom=0)

plt.show()
```

Price Distribution Overview



Interpretations:

1. Summarise your findings from the visual

- Highest number of houses are in price range 100K - 200K.
- Middle class buyers may target for wider spread of the price ranging 40K to 100K.

2. The reason for selecting the chart type you did
   - We need to visualise only one quantitative measure namely, 'price_eur'.
   - Histogram is one of the best way to represent for this purpose.
3. Mention the pre-attentive attributes used.(atleast 2)
   - Distribution of quantitative 'price_eur' data variable is represented as series of bars as visual variable.
   - Each bar/bin represents a range of data variable values and its height (visual variable) indicates how many data points fall within that range.
4. Mention the gestalt principles used.(atleast 2)
   - Gestalt Law of Similarity - Text and the histogram plots are of same color.
   - Gestalt Law of Closure - Although there is no frame around each plot, the axis lines and the labels are enough to define a closed space.

Question 5

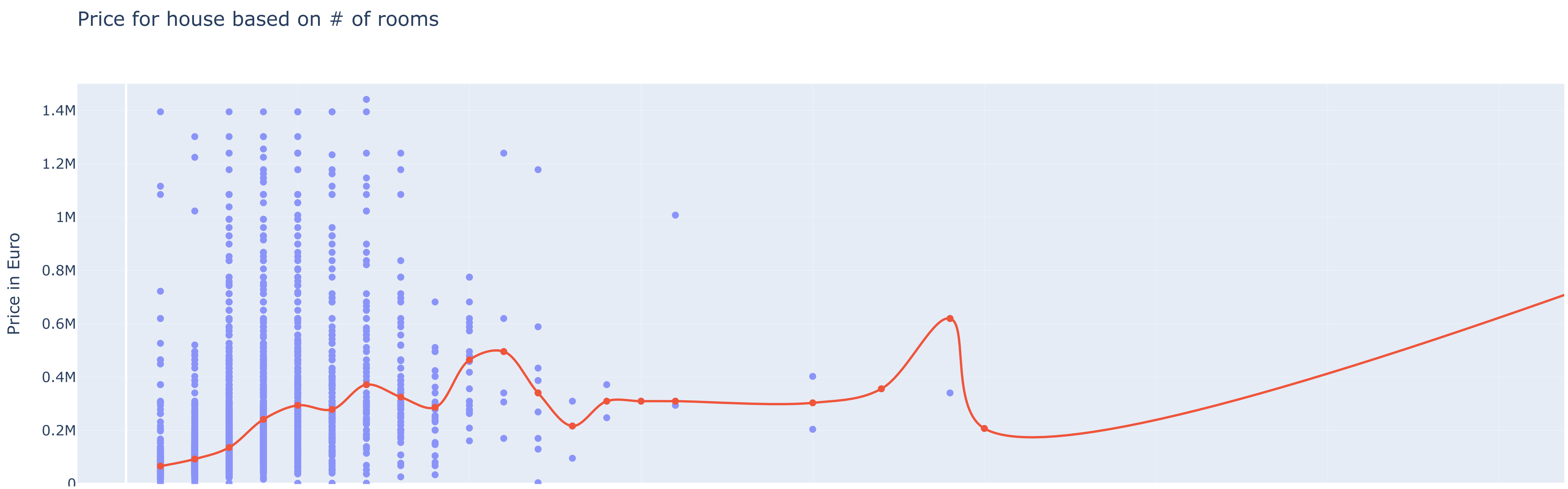
What kind of relation exists between number of rooms in the house and its price. Create an appropriate visual to show any kind of relation that exists between room count and price of the house in one single visual.

Write the python code in the below cell to create appropriate visual to perform the above task.

Answer in markdown cells below the visual

1. **Summarise your findings from the visual**
2. **The reason for selecting the chart type you did**
3. **Mention the pre-attentive attributes used.(atleast 2)**
4. **Mention the gestalt principles used.(atleast 2)**

In [40]:
```
fig_df = df.groupby('room')['price_eur'].agg('median').reset_index()

fig = go.Figure()
fig.add_trace(go.Scatter(x=df['room'], y=df['price_eur'],
                    mode='markers',
                    name='Price',
                    opacity=0.7))
fig.add_trace(go.Scatter(x=fig_df['room'], y=fig_df['price_eur'],
                    mode='lines+markers',
                    name='Median price',
                    line_shape='spline'))
fig.update_layout(title='Price for house based on # of rooms',
                   xaxis_title='No. of rooms',
                   yaxis_title='Price in Euro')
fig.update_yaxes(range=[-50, 1500000])
fig.show()
```



Interpretations:

1. Summarise your findings from the visual
   - The average price tends to increase with the number of rooms, with some fluctuations, particularly between 5 to 20 rooms.
   - There is a noticeable spread of individual prices, especially for houses with fewer rooms (1-10 rooms), indicating variability in pricing. As the number of rooms increases beyond 20, prices generally rise more consistently.
2. The reason for selecting the chart type you did
   - A line chart is ideal for displaying trends over a continuous variable (number of rooms).
   - Combining line chart with scatter plot allows for a clear comparison between the average trend and the spread of individual data points.
3. Mention the pre-attentive attributes used.(atleast 2)
   - Color: Different colors are used to distinguish between the average price (red) and individual prices (blue). This contrast helps viewers quickly differentiate between the two data series.
   - Position: The x and y positions of the points are used to convey the number of rooms and the price, respectively. This spatial arrangement makes it easy to understand the relationship between the variables.
4. Mention the gestalt principles used.(atleast 2)
   - Proximity: The data points for individual prices are grouped closely along the number of rooms, indicating their relationship. The proximity helps viewers understand that these prices are all associated with a particular room count.
   - Continuity: The line showing the average price follows a smooth, continuous path. This principle helps viewers perceive the trend as a single, unified object, making the overall pattern easier to follow.

Group's choice-2 Marks

Frame 1 (more) question which will help in the EDA(Exploratory Data Analysis) of the given data set and answer the same using the best visual.

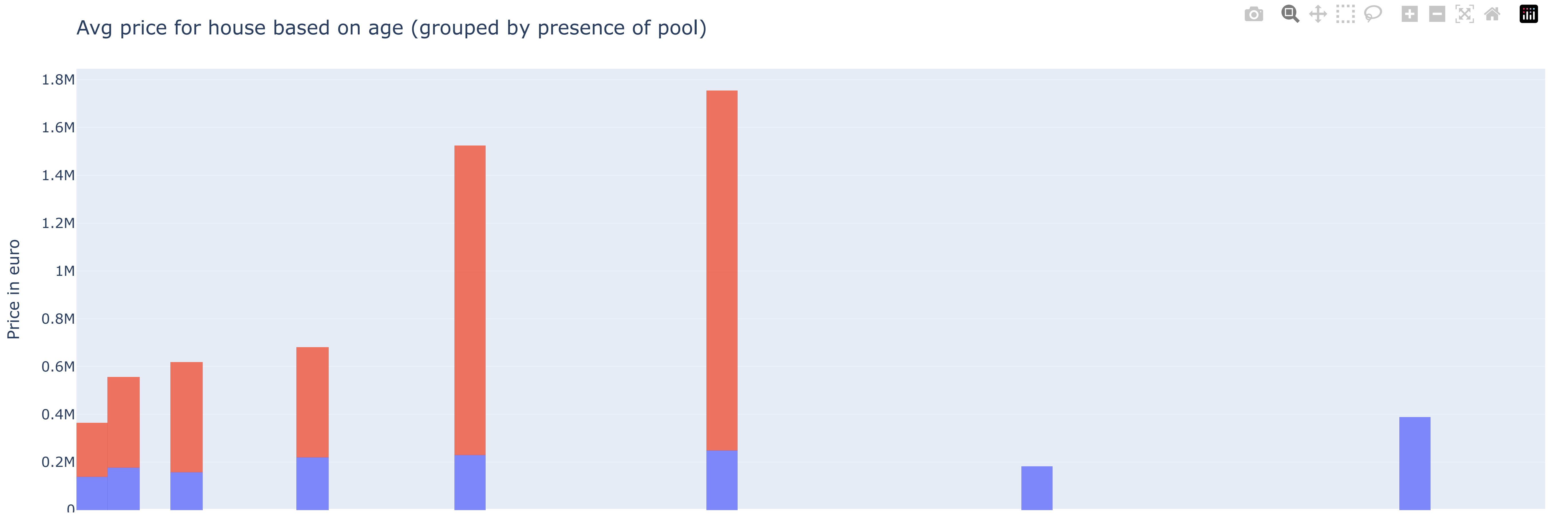
1. Write the question in a markdown cell
2. Below the question,in a coding cell,write the python code to create the visual to answer the question

Answer in markdown cells below the visual

1.Summarise your findings from the visual.
2.The reason for selecting the chart type you did
3.Mention the pre-attentive attributes used.(atleast 2)
4.Mention the gestalt principles used.(atleast 2)

What is the effect of age of the house on price of the house for cases where a pool is present?

In [41]:
```
df['pool'] = df['pool'].astype(bool)
fig = px.histogram(df, x="mean_age", histfunc='avg', y="price_eur", color="pool", opacity=0.8)
fig.update_layout(title='Avg price for house based on age (grouped by presence of pool)',
                  xaxis_title='Avg age in years',
                  yaxis_title='Price in euro')
fig.show()
```



Interpretations:

1. Summarise your findings from the visual
    - Houses with pool have significantly higher price than those without, especially in the 20-40 year age.

- House prices have an increasing trend as the age increases
- Houses over 40 years old do not have pools

2. The reason for selecting the chart type you did
   - A stacked histogram chart was chosen because it effectively compares average house prices across different age groups.
3. Mention the pre-attentive attributes used.(atleast 2)
   - Color: Red bars represent houses with a pool (True), while blue bars represent houses without a pool (False). This color distinction helps viewers quickly identify the two categories.
   - Form: The length of bars visually represents the magnitude of house prices, allowing immediate comparison across different categories.
4. Mention the gestalt principles used.(atleast 2)
   - Similarity: Consistent color coding (red and blue) groups related data (houses with/without pools) together, aiding quick comprehension.
   - Proximity: Related items (red and blue bars) are placed close to each other within the same age group, suggesting they should be compared as a set.

************ END OF ASSIGNMENT ****************