Tema1

Getting api key

Post request at *http://141.85.224.171:8000/key* to get the api key and use it later to get the data.

Store it in json file: *__api_key.json__*.

In [ ]:
```
import requests
import json

url = 'http://141.85.224.171:8000/key'
identifier = ''
body = {'identifier': identifier}

response = requests.post(url, json = body)

if response.status_code == 200:
    print("API key retrieved successfully!")

    # Extract the API key from the response body
    api_key = response.json()["key"]

    # Store the API key in a JSON file
    with open("api_key.json", "w") as f:
        json.dump({"api_key": api_key}, f)

    api_key
else:
    print("Error retrieving key. Status code:", response.status_code)
```

```
API key retrieved successfully!
```

Getting data

Post request at *http://141.85.224.171:8001/dmda/a1/data* with the key in the body and retrieve the data.

Decode it and store it in json file: *__data.json__*.

In [ ]:
```
import json
import base64
import gzip

# Step 1: Load API key
with open("api_key.json", "r") as f:
    api_key = json.load(f)["api_key"]

# Step 2: Make POST request
url = 'http://141.85.224.171:8001/dmda/a1/data'
body = {
    'identity': {
        'identifier_value': identifier,
```

```
        'key': api_key
    }
}

response = requests.post(url, json = body)

# Step 3: Check if request was successful
if response.status_code == 200:
    response_body = json.loads(response.text)
    data = response_body["data"]
    data_format = response_body["format"]
    metadata = response_body["metadata"]

    # Decode data
    if metadata.get("byte_encoding", "base64"):
        data = base64.b64decode(data)

    # If data is compressed, uncompress it
    if metadata.get("compressed", True):
        # Add other decompression formats here if necessary
        if metadata["compression_format"] == "gzip":
            data = gzip.decompress(data)

    # Parse the data according to its format
    # Add other formats here if necessary
    if data_format == "json":
        # Process the parsed data here as needed
        data = json.loads(data)

    # Step 4: Write the unencrypted data to the file
    with open("data.json", "w") as f:
        json.dump(data, f)

    data
else:
    print("Error: Request unsuccessful. Status code:", response.status_code)
```

Saving data as csv and format some columns

Loading the data from json file.

Clean some columns values such as *price*, *Suprafaţă utilă:*, *Suprafaţă utilă totală:*, *Suprafaţă construită:*, *last_update*, *An construcţie:* and *Nr. balcoane:*.

Write the rows in csv format in ***data.csv***.

RUN FROM HERE

Make sure you have Python 3.10.10

In [ ]:
```
import json, csv, re
import numpy as np

# Load the previously retrieved data
with open('data.json') as json_data:
    apartments = json.load(json_data)
```

```
# header
fields = apartments[0].keys()
print(fields)

try:
    with open('data.csv', 'w', encoding="utf-8") as csv_file:
        csv_writer = csv.DictWriter(csv_file, fieldnames=fields)
        csv_writer.writeheader()

        for apartment in apartments:
            # Clean data
            for column in fields:
                if column in apartment:
                    if column == 'price':
                        apartment[column] = apartment[column].replace('.', '')
                        apartment[column] = apartment[column].replace(',', '')
                    elif column == 'Suprafață utilă:':
                        apartment[column] = apartment[column].replace(',', '.').
                    elif column == 'Suprafață utilă totală:':
                        apartment[column] = apartment[column].replace(',', '.').
                    elif column == 'Suprafață construită:':
                        apartment[column] = apartment[column].replace(',', '.').
                    elif column == 'last_update':
                        if any(map(apartment[column].__contains__, ['azi', 'Azi'
                            apartment[column] = '20.03.2023'
                        elif any(map(apartment[column].__contains__, ['ieri', 'I
                            apartment[column] = '19.03.2023'
                        elif 'Actualizat în' in apartment[column]:
                            apartment[column] = apartment[column].removeprefix('
                    elif column == 'An construcție:':
                        years = re.findall("\d{4}", apartment[column])
                        if (len(years) == 1):
                            apartment[column] = int(years[0])
                        else:
                            # get the mean value between the 2 years
                            apartment[column] = int(np.mean(np.array(list(map(in
                    elif column == 'Nr. balcoane:':
                        apartment[column] = re.findall(r'\b\d+\b', apartment[col

            csv_writer.writerow(dict(zip(fields, [None if column not in apartmen
except IOError:
    print("I/O error")
```

```
dict_keys(['id', 'url', 'location', 'price', 'last_update', 'Nr. camere:', 'Sup
rafață utilă:', 'Suprafață utilă totală:', 'Suprafață construită:', 'Compartime
ntare:', 'Etaj:', 'Nr. bucătării:', 'Nr. băi:', 'An construcție:', 'Structură r
ezistență:', 'Tip imobil:', 'Regim înălțime:', 'Nr. balcoane:', 'Utilitati gene
rale', 'Sistem incalzire', 'Acces internet', 'Ferestre cu geam termopan', 'Usa
intrare', 'Izolatii termice', 'Podele', 'Pereti', 'Alte spatii utile', 'Contori
zare', 'Dotari imobil', 'Servicii imobil', 'Amenajare strazi', 'agentName', 'ag
entCompany'])
```

Loading data from csv

In [ ]:

```
import pandas as pd
import numpy as np

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

```
# df
```

Cleaning data & fill gaps

First we see what column types are and we enforce a new type to better control the values (Will do this also on upcoming column changes but for now we will make it on the most obvious ones).

Fill nan values, considering that *Suprafaţă utilă totală* is *Suprafaţă utilă* for that rows (normally the *Suprafaţă utilă* totală would be without balcony and have the same value as *Suprafaţă utilă*, ***but Romania*** :))

Then we filter the data and keep the ones that are after 2022 updated. Before that year we will consider outdated. Also drop the duplicates, the rows that have multiple key columns, at once, empty, the rows that don't have *Suprafaţă utilă* or *Structură rezistenţă* (regarding the latter we observed that if they don't have it, they don't have also many other columns).

Get the *location* splitted for maybe we can see a plot about price against location later.

Add a new column, ***price_sqm***, to better understand the value of our dataset

In [ ]:
```
# let's take a look at what the object columns are
print(df.select_dtypes('object').columns)

# Clean
df['price'] = pd.to_numeric(df['price'])
df['last_update'] = pd.to_datetime(df['last_update'], format='%d.%m.%Y')
df['An construcţie:'] = pd.to_numeric(df['An construcţie:'])
df['Suprafaţă utilă:'] = pd.to_numeric(df['Suprafaţă utilă:'])
df['Suprafaţă utilă totală:'] = pd.to_numeric(df['Suprafaţă utilă totală:'])
df['Suprafaţă construită:'] = pd.to_numeric(df['Suprafaţă construită:'])
df['Nr. balcoane:'] = pd.to_numeric(df['Nr. balcoane:'])

# Fill nan values, considering that Suprafaţă utilă totală is Suprafaţă utilă fo
df['Suprafaţă utilă totală:'].fillna(df['Suprafaţă utilă:'], inplace=True)

print(len(df.index))
# Remove data updated before 2022
df = df.drop(df[df['last_update'] < '2022-01-01'].index)
# Remove duplicate rows
df = df.drop_duplicates(subset=['id'])
# Remove rows that have multiple key columns at once empty
subset = ['Nr. camere:', 'Suprafaţă utilă:', 'Suprafaţă utilă totală:', 'Suprafa
df = df.dropna(subset=subset, how='all')
# Remove rows that don't have surface
df = df.dropna(subset=['Suprafaţă utilă:'])
# Remove rows that don't have structura rezistenta because they lack many more c
df = df.dropna(subset=['Structură rezistenţă:'])

#split location
df[['loc', 'z']] = df['location'].str.split(', zona', expand=True)
df[['o', 's']] = df['loc'].str.split(', ', expand=True)
df[['sect', 's']] = df['s'].str.split(' ', expand=True)
df.insert(loc=2, column='oras', value=df['o'])
```

```
df.insert(loc=3, column='sector', value=df['s'])
df['sector'] = pd.to_numeric(df['sector'])
df.insert(loc=4, column='zona', value=df['z'])
df = df.drop(columns=['loc', 'location', 'o', 's', 'z', 'sect'])

print(len(df.index))
#Add new column price/mp
df.insert(loc=6, column='price_sqm', value=np.where(df['price'] < 20000, df['pri

#Save new df
df.to_csv('data.csv', index=False)
```

```
Index(['id', 'url', 'location', 'last_update', 'Compartimentare:', 'Etaj:',
       'Structură rezistență:', 'Tip imobil:', 'Regim înălțime:',
       'Utilitati generale', 'Sistem incalzire', 'Acces internet',
       'Ferestre cu geam termopan', 'Usa intrare', 'Izolatii termice',
       'Podele', 'Pereti', 'Alte spatii utile', 'Contorizare', 'Dotari imobil',
       'Servicii imobil', 'Amenajare strazi', 'agentName', 'agentCompany'],
      dtype='object')
9935
7644
```

Show stats

Regarding ny nan values are for each column, percentage and number

And to understand some calculations on numerical columns

In [ ]:
```
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
for i in np.arange(df.shape[1]):
    n = df.iloc[:,i].isnull().sum()
    if n > 0:
        print(list(df.columns.values)[i] + ': ' + str(n) + ' nans')

print(df.describe())
```

```
id - 0%
url - 0%
oras - 0%
sector - 48%
zona - 0%
price - 0%
price_sqm - 0%
last_update - 0%
Nr. camere: - 0%
Suprafață utilă: - 0%
Suprafață utilă totală: - 0%
Suprafață construită: - 5%
Compartimentare: - 2%
Etaj: - 0%
Nr. bucătării: - 12%
Nr. băi: - 0%
An construcție: - 1%
Structură rezistență: - 0%
Tip imobil: - 0%
Regim înălțime: - 1%
Nr. balcoane: - 16%
Utilitati generale - 6%
Sistem incalzire - 11%
Acces internet - 40%
Ferestre cu geam termopan - 26%
Usa intrare - 20%
Izolatii termice - 35%
Podele - 15%
Pereti - 15%
Alte spatii utile - 64%
Contorizare - 35%
Dotari imobil - 27%
Servicii imobil - 57%
Amenajare strazi - 59%
agentName - 0%
agentCompany - 7%
sector: 3701 nans
Suprafață construită:: 370 nans
Compartimentare:: 127 nans
Nr. bucătării:: 905 nans
Nr. băi:: 17 nans
An construcție:: 40 nans
Regim înălțime:: 71 nans
Nr. balcoane:: 1257 nans
Utilitati generale: 461 nans
Sistem incalzire: 842 nans
Acces internet: 3089 nans
Ferestre cu geam termopan: 1973 nans
Usa intrare: 1500 nans
Izolatii termice: 2660 nans
Podele: 1121 nans
Pereti: 1142 nans
Alte spatii utile: 4910 nans
Contorizare: 2650 nans
Dotari imobil: 2046 nans
Servicii imobil: 4335 nans
Amenajare strazi: 4547 nans
agentName: 1 nans
agentCompany: 550 nans
            sector         price     price_sqm  Nr. camere:  Suprafață utilă:
```

```
\
count  3943.000000  7.644000e+03   7644.000000  7644.000000       7644.000000
mean      3.158002  1.477409e+05   2190.279858     2.512297         71.627471
std       1.705080  1.661111e+05   1689.161047     0.925021         36.591473
min       1.000000  4.000000e+01     40.000000     1.000000          1.000000
25%       2.000000  7.200000e+04   1332.912046     2.000000         51.000000
50%       3.000000  1.140600e+05   1770.988806     2.000000         63.000000
75%       4.000000  1.732500e+05   2441.850476     3.000000         81.000000
max       6.000000  3.499000e+06  19775.000000    10.000000        600.000000

       Suprafață utilă totală:  Suprafață construită:  Nr. bucătării:  \
count              7644.000000            7274.000000     6739.000000
mean                 74.172080              87.819764        1.010387
std                  40.213568              53.122021        0.231554
min                   1.000000               1.187000        1.000000
25%                  52.000000              60.000000        1.000000
50%                  64.000000              75.000000        1.000000
75%                  83.000000              98.000000        1.000000
max                 605.000000             800.000000       11.000000

          Nr. băi:  An construcție:  Nr. balcoane:
count  7627.000000      7604.000000    6387.000000
mean      1.449849      2003.145055       1.233130
std       0.656182        24.621809       0.538506
min       1.000000      1874.000000       1.000000
25%       1.000000      1983.000000       1.000000
50%       1.000000      2019.000000       1.000000
75%       2.000000      2022.000000       1.000000
max      11.000000      2025.000000      11.000000
```

Dropping rows

Drop some columns that have nearly and over 50% empty cells and not relevant for now

For example, *Alte spatii utile*, that has info about if the property has backyard or not, which can be obvious that will increase the price if it is present.

Also dropping *agentName* and *agentCompany* as they don't have relevant info and are very wrong (some *agentNames* are on *agentCompany* columns and agentName has many nan values)

In [ ]:
```
df = df.drop(columns=['Suprafață utilă:', 'Suprafață construită:', 'Acces intern
```

Fill nan values

As the stats say, we can fill those values for the columns *Nr. bucătării:* and *Nr. balcoane:* with mean value, as this is probaby the real value. A property might have 1 kitchen rather than 2.

In column *Compartimentare:* most of them are decomandat and because there are 2% nan values we can assume thta the rest are decomandat and fill in the gaps

For number of bathrooms it is clear that depends on the number of rooms of the property

When it comes to *Structură rezistență:*, it depends on the majority. In this case, a block of flats is likely to be constructed with beton and a house with caramida

In [ ]:
```
# Fill missing values with the mean for nr. buc, nr. balcoane
df['Nr. bucătării:'] = df['Nr. bucătării:'].fillna(round(df['Nr. bucătării:'].me
df['Nr. balcoane:'] = df['Nr. balcoane:'].fillna(round(df['Nr. balcoane:'].mean(

print(df['Compartimentare:'].value_counts())
df['Compartimentare:'] = df['Compartimentare:'].fillna('decomandat')

m1 = (df['Nr. camere:'] < 3)
m2 = (df['Nr. camere:'] > 2)
df.loc[m1, 'Nr. băi:'] = df.loc[m1, 'Nr. băi:'].fillna(1)
df.loc[m2, 'Nr. băi:'] = df.loc[m2, 'Nr. băi:'].fillna(2)

m = (df['Tip imobil:'] == 'bloc de apartamente')
print(df.loc[m, 'Structură rezistență:'].value_counts())
df.loc[m, 'Structură rezistență:'] = df.loc[m, 'Structură rezistență:'].fillna('
m = (df['Tip imobil:'] == 'casa/vila')
print(df.loc[m, 'Structură rezistență:'].value_counts())
df.loc[m, 'Structură rezistență:'] = df.loc[m, 'Structură rezistență:'].fillna('
```

```
decomandat        5348
semidecomandat    2043
circular            67
nedecomandat        54
vagon                5
Name: Compartimentare:, dtype: int64
beton       6392
altele       627
caramida     469
bca           23
lemn           1
Name: Structură rezistență:, dtype: int64
caramida    67
beton       50
altele      15
Name: Structură rezistență:, dtype: int64
```

Sort of OneHotEncoding

For the floor I transformed the string values between 0 and 1 as follows based on personal preference, personal market research and inquiries and what a market research said (i can't find the source anymore :( ): | Demisol & Etaj x / x (as in last floor) | Ultimele 2 | Parter | Mansarda | Etaj | | ---------- | ---------- | ---------- | ---------- | ---------- | | 0 | 0.25 | 0.5 | 0.75 | 1 |

For the partitioning and structural resistence I encoded the values in numbers from 1 to 5.

For type I encoded in a simple OneHot column encoding such that is the type is apartment it is 1, house a 0.

And for the height regime i encoded it as a sum, based on the letters and numbers every row had: If it contained S, D, P, M it means +1 and on a floor, for example 11E, +11. Empty cells were defaulted to 0.

In [ ]:
```
# for floors
df.insert(loc=14, column='Etaj_cod', value=df['Etaj:'].astype(str).apply(lambda
for i in range(2, 26):
    etaj = 'Etaj ' + str(i) + ' / ' + str(i)
    df['Etaj_cod'] = df['Etaj_cod'].astype(str).apply(lambda x: 0 if etaj in x e

df['Etaj_cod'] = df['Etaj_cod'].astype(str).apply(lambda x: 0 if 'Demisol' in x
df['Etaj_cod'] = df['Etaj_cod'].astype(str).apply(lambda x: 0.25 if 'Ultimele 2'
df['Etaj_cod'] = df['Etaj_cod'].astype(str).apply(lambda x: 0.5 if 'Parter' in x
df['Etaj_cod'] = df['Etaj_cod'].astype(str).apply(lambda x: 0.75 if 'Mansarda' i
df['Etaj_cod'] = df['Etaj_cod'].astype(str).apply(lambda x: 1 if 'Etaj' in x els
df['Etaj_cod'] = pd.to_numeric(df['Etaj_cod'])
df = df.drop(columns=['Etaj:'])

# for partitioning
df.insert(loc=13, column='Compartimentare_cod', value=df['Compartimentare:'].ast
df['Compartimentare_cod'] = df['Compartimentare_cod'].astype(str).apply(lambda x
df['Compartimentare_cod'] = df['Compartimentare_cod'].astype(str).apply(lambda x
df['Compartimentare_cod'] = df['Compartimentare_cod'].astype(str).apply(lambda x
df['Compartimentare_cod'] = df['Compartimentare_cod'].astype(str).apply(lambda x
df['Compartimentare_cod'] = pd.to_numeric(df['Compartimentare_cod'])
df = df.drop(columns=['Compartimentare:'])

# for structural resistence
df.insert(loc=20, column='Struct_rezistenta_cod', value=df['Structură rezistență
df['Struct_rezistenta_cod'] = df['Struct_rezistenta_cod'].astype(str).apply(lamb
df['Struct_rezistenta_cod'] = df['Struct_rezistenta_cod'].astype(str).apply(lamb
df['Struct_rezistenta_cod'] = df['Struct_rezistenta_cod'].astype(str).apply(lamb
df['Struct_rezistenta_cod'] = df['Struct_rezistenta_cod'].astype(str).apply(lamb
df['Struct_rezistenta_cod'] = pd.to_numeric(df['Struct_rezistenta_cod'])
df = df.drop(columns=['Structură rezistență:'])

# for type
df.insert(loc=22, column='tip_imobil_cod', value=df['Tip imobil:'].astype(str).a
df['tip_imobil_cod'] = pd.to_numeric(df['tip_imobil_cod'])

#pt regim inaltime
df['Regim înălțime:'] = df['Regim înălțime:'].fillna('0')
```

```
def encode_regim(x):
    parts = x.split('+')
    code = 0
    for part in parts:
        match part:
            case 'S':
                code += 1
            case 'D':
                code += 1
            case 'P':
                code += 1
            case 'M':
                code += 1
            case other:
                code += int(part.split('E')[0])
    return str(code)

df['Regim înălțime:'] = df['Regim înălțime:'].apply(lambda x: np.where(x != '0',
df['Regim înălțime:'] = pd.to_numeric(df['Regim înălțime:'])
```

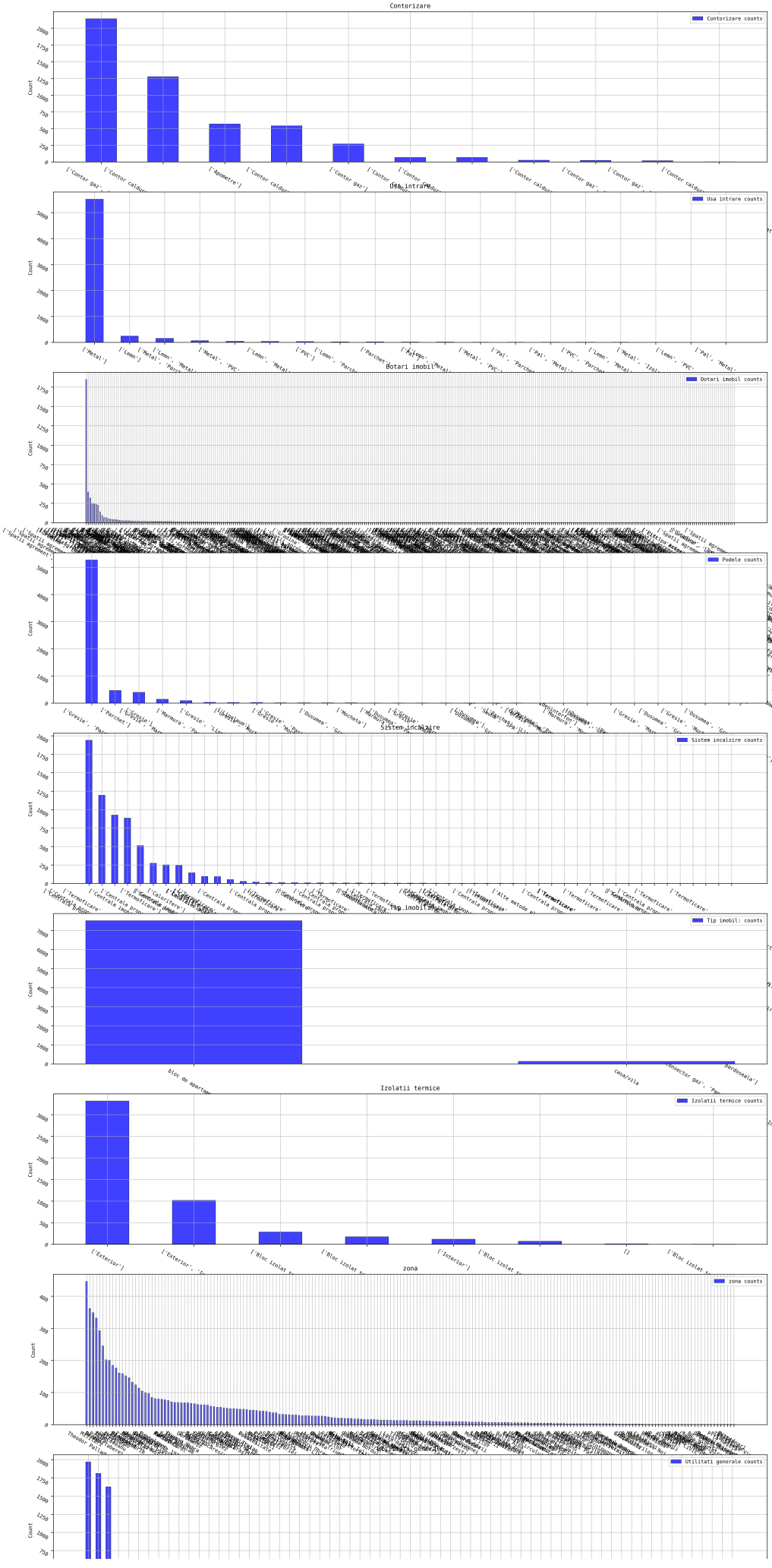
Plots

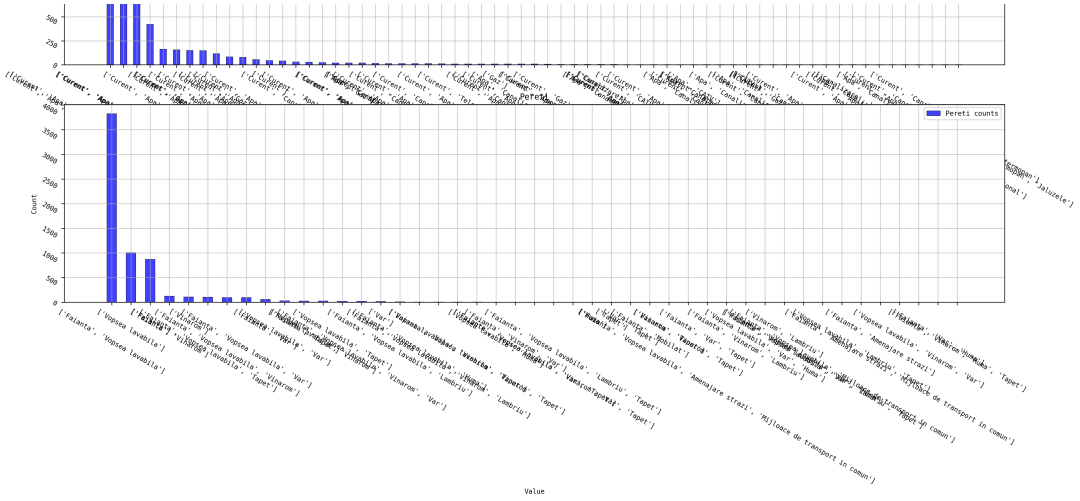
Understand the data

Let's see how is our data distributed

In [ ]:
```
import matplotlib.pyplot as plt

# plot for some the categorical features
cols = df.select_dtypes('object').drop(columns=['id', 'url', 'oras', ]).columns
cols = np.random.choice(cols, 10, replace=False)
fig, ax = plt.subplots(len(cols), 1, figsize=(20, 50), dpi=300)
plt.rcParams['font.family'] = 'monospace'
plt.rcParams['font.size'] = 8
for idx, col in enumerate(cols):
    counts = df[col].value_counts()
    ax[idx].bar(counts.index, counts.values, alpha=0.75, label=f'{col} counts',
    ax[idx].set_title(col)
    ax[idx].set_ylabel('Count')
    ax[idx].set_xlabel('Value')
    ax[idx].tick_params(labelrotation=-30)
    ax[idx].grid(alpha=0.75)
    ax[idx].legend()
plt.tight_layout()
plt.show()
```

```
C:\Users\grecu\AppData\Local\Temp\ipykernel_2832\1893728354.py:18: UserWarning:
Tight layout not applied. tight_layout cannot make axes height small enough to
accommodate all axes decorations.
  plt.tight_layout()
```


Contorizare
Contorizare counts
Count
Usa intrare
Usa intrare counts
Count
Dotari imobil
Dotari imobil counts
Count
Podele
Podele counts
Count
Sistem incalzire
Sistem incalzire counts
Count
Tip imobil
Tip imobil counts
Count
bloc de apartamente
casa/vila
Izolatii termice
Izolatii termice counts
Count
zona
zona counts
Count
Utilitati generale
Utilitati generale counts
Count



Encodare coloane in some sort of one hot

In [ ]:
```
#corectare coloanele lista
df['Utilitati generale'] = df['Utilitati generale'].fillna("['Curent', 'Apa', 'C
df['Sistem incalzire'] = df['Sistem incalzire'].fillna("[]")
df['Contorizare'] = df['Contorizare'].fillna("[]")
df['Ferestre cu geam termopan'] = df['Ferestre cu geam termopan'].fillna("[]")

def funct(x):
    return eval(x)
df['Utilitati generale'] = df['Utilitati generale'].apply(lambda x: funct(x))
df['Sistem incalzire'] = df['Sistem incalzire'].apply(lambda x: funct(x))
df['Contorizare'] = df['Contorizare'].apply(lambda x: funct(x))
df['Ferestre cu geam termopan'] = df['Ferestre cu geam termopan'].apply(lambda x
```

In [ ]:
```
#daca exista item in coloana cu fereastra cu termopan, atunci trebuie introdusa
def add_termopan(x):
    if len(x['Ferestre cu geam termopan']) != 0 and 'Ferestre cu geam termopan'
        x['Utilitati generale'].append('Ferestre cu geam termopan')
    return x['Utilitati generale']
df['Utilitati generale'] = df.apply(lambda x: np.where(len(x['Ferestre cu geam t

#daca exista 'Centrala imobil' sau 'Centrala proprie' in coloana 'Sistem incalzi
def add_gaz(x):
    sist = len(x['Sistem incalzire']) != 0 and ('Centrala imobil' in x['Sistem i
    contor = len(x['Contorizare']) != 0 and 'Contor gaz' in x['Contorizare']
    if (sist or contor) and 'Gaz' not in x['Utilitati generale']:
        x['Utilitati generale'] = np.append(x['Utilitati generale'], 'Gaz')
    return x['Utilitati generale']
df['Utilitati generale'] = df.apply(lambda x: np.where(((len(x['Sistem incalzire

#drop la coloanele 'Ferestre cu geam termopan', 'Sistem incalzire', 'Izolatii te
df = df.drop(columns=['Tip imobil:', 'Usa intrare', 'Ferestre cu geam termopan',

df = df.drop('Utilitati generale', 1).join(pd.get_dummies(pd.DataFrame(df['Utili
```

```
C:\Users\grecu\AppData\Local\Temp\ipykernel_2832\1957549454.py:20: FutureWarnin
g: In a future version of pandas all arguments of DataFrame.drop except for the
argument 'labels' will be keyword-only.
  df = df.drop('Utilitati generale', 1).join(pd.get_dummies(pd.DataFrame(df['Ut
ilitati generale'].tolist(), df.index).stack()).astype(int).groupby(level=0).su
m(), how = 'left', lsuffix='left', rsuffix='right')
```

Removing the rows where we don't have data on many columns

Exactly 2 columns

In [ ]:
```
df = df.dropna(subset=['Apa'])
```

Save the df to csv

In [ ]:
```
#Save new df
df.to_csv('data.csv', index=False)
```

Rerun stats

In [ ]:
```
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{} - {}%'.format(col, round(pct_missing*100)))
print(df.isnull().sum())
print(df.describe())
for i in np.arange(df.shape[1]):
    n = df.iloc[:,i].isnull().sum()
    if n > 0:
        print(list(df.columns.values)[i] + ': ' + str(n) + ' nans')
```

```
id - 0%
url - 0%
oras - 0%
sector - 48%
zona - 0%
price - 0%
price_sqm - 0%
last_update - 0%
Nr. camere: - 0%
Suprafaţă utilă totală: - 0%
Nr. bucătării: - 0%
Nr. băi: - 0%
Compartimentare_cod - 0%
Etaj_cod - 0%
An construcţie: - 1%
Regim înălţime: - 0%
Nr. balcoane: - 0%
Struct_rezistenta_cod - 0%
tip_imobil_cod - 0%
Apa - 0%
CATV - 0%
Canalizare - 0%
Curent - 0%
Ferestre cu geam termopan - 0%
Gaz - 0%
Jaluzele - 0%
Telefon - 0%
Telefon international - 0%
id                              0
url                             0
oras                            0
sector                       3701
zona                            0
price                           0
price_sqm                       0
last_update                     0
Nr. camere:                     0
Suprafaţă utilă totală:         0
Nr. bucătării:                  0
Nr. băi:                        0
Compartimentare_cod             0
Etaj_cod                        0
An construcţie:                40
Regim înălţime:                 0
Nr. balcoane:                   0
Struct_rezistenta_cod           0
tip_imobil_cod                  0
Apa                             0
CATV                            0
Canalizare                      0
Curent                          0
Ferestre cu geam termopan       0
Gaz                             0
Jaluzele                        0
Telefon                         0
Telefon international           0
dtype: int64
            sector         price     price_sqm  Nr. camere:  \
count  3941.000000  7.642000e+03   7642.000000  7642.000000
mean      3.157067  1.477643e+05   2190.608305     2.512170
```

```
std       1.705007  1.661263e+05   1689.257810     0.924967
min       1.000000  4.000000e+01     40.000000     1.000000
25%       2.000000  7.200000e+04   1333.333333     2.000000
50%       3.000000  1.140830e+05   1771.144279     2.000000
75%       4.000000  1.732500e+05   2441.890410     3.000000
max       6.000000  3.499000e+06  19775.000000    10.000000

       Suprafață utilă totală:  Nr. bucătării:     Nr. băi:  \
count              7642.000000     7642.000000  7642.000000
mean                 74.173695        1.009160     1.449359
std                  40.215778        0.217467     0.655871
min                   1.000000        1.000000     1.000000
25%                  52.000000        1.000000     1.000000
50%                  64.000000        1.000000     1.000000
75%                  83.000000        1.000000     2.000000
max                 605.000000       11.000000    11.000000

       Compartimentare_cod     Etaj_cod  An construcție:  ...  tip_imobil_cod
\
count          7642.000000  7642.000000      7602.000000  ...     7642.000000
mean              3.693143     0.767960      2003.152065  ...        0.982727
std               0.529884     0.388807        24.621241  ...        0.130295
min               1.000000     0.000000      1874.000000  ...        0.000000
25%               3.000000     0.500000      1983.000000  ...        1.000000
50%               4.000000     1.000000      2019.000000  ...        1.000000
75%               4.000000     1.000000      2022.000000  ...        1.000000
max               5.000000     1.000000      2025.000000  ...        1.000000

              Apa         CATV   Canalizare       Curent  \
count  7642.000000  7642.000000  7642.000000  7642.000000
mean      0.950406     0.575111     0.915991     0.991494
std       0.217120     0.494358     0.277420     0.091839
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     1.000000     1.000000
50%       1.000000     1.000000     1.000000     1.000000
75%       1.000000     1.000000     1.000000     1.000000
max       1.000000     1.000000     1.000000     1.000000

       Ferestre cu geam termopan          Gaz     Jaluzele      Telefon  \
count                7642.000000  7642.000000  7642.000000  7642.000000
mean                    0.746925     0.934310     0.000262     0.373332
std                     0.434802     0.247755     0.016176     0.483721
min                     0.000000     0.000000     0.000000     0.000000
25%                     0.000000     1.000000     0.000000     0.000000
50%                     1.000000     1.000000     0.000000     0.000000
75%                     1.000000     1.000000     0.000000     1.000000
max                     1.000000     1.000000     1.000000     1.000000

       Telefon international
count            7642.000000
mean                0.070270
std                 0.255617
min                 0.000000
25%                 0.000000
50%                 0.000000
75%                 0.000000
max                 1.000000

[8 rows x 23 columns]
```

```
sector: 3701 nans
An construcție:: 40 nans
```

More plots

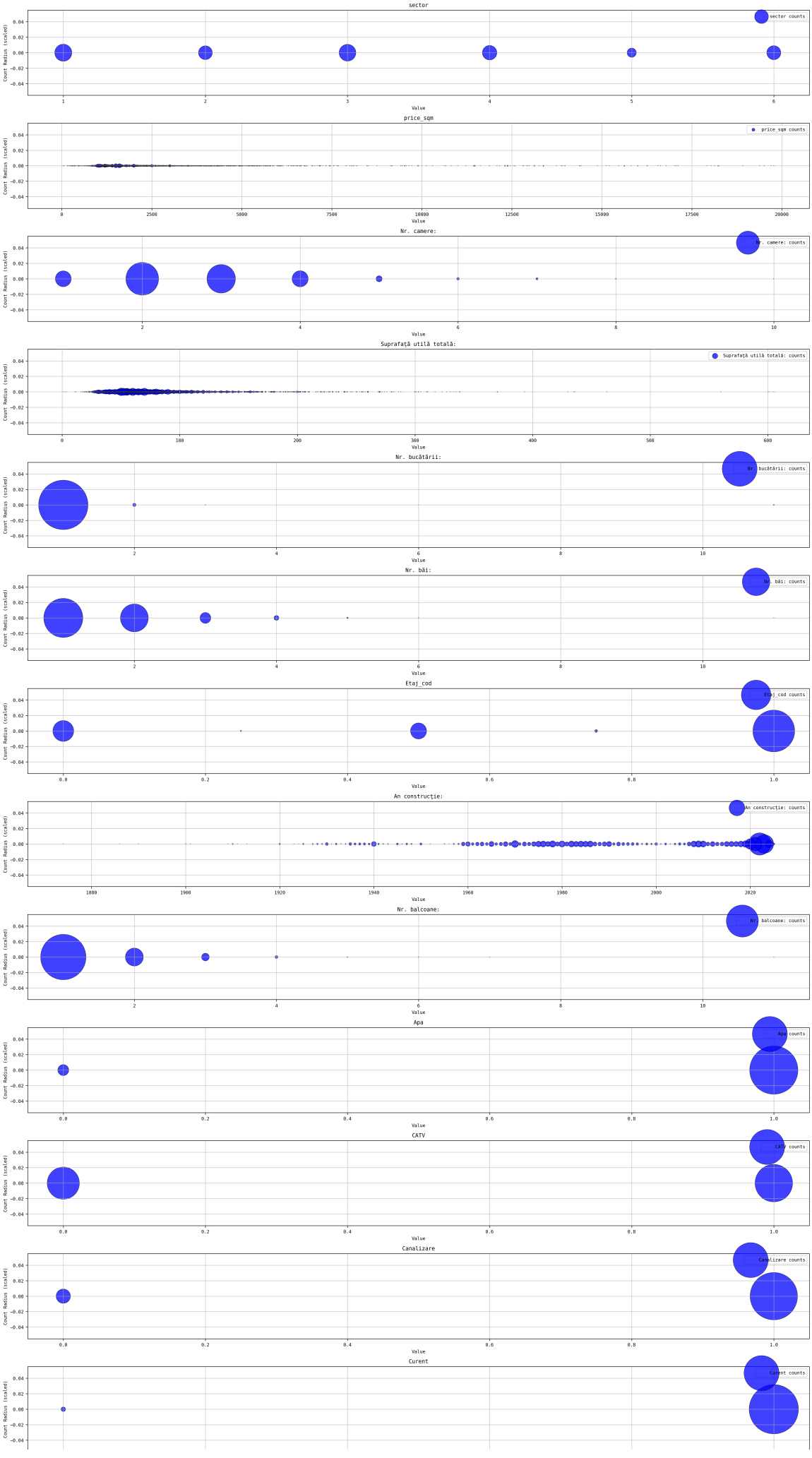
This time to understand the numerical columns

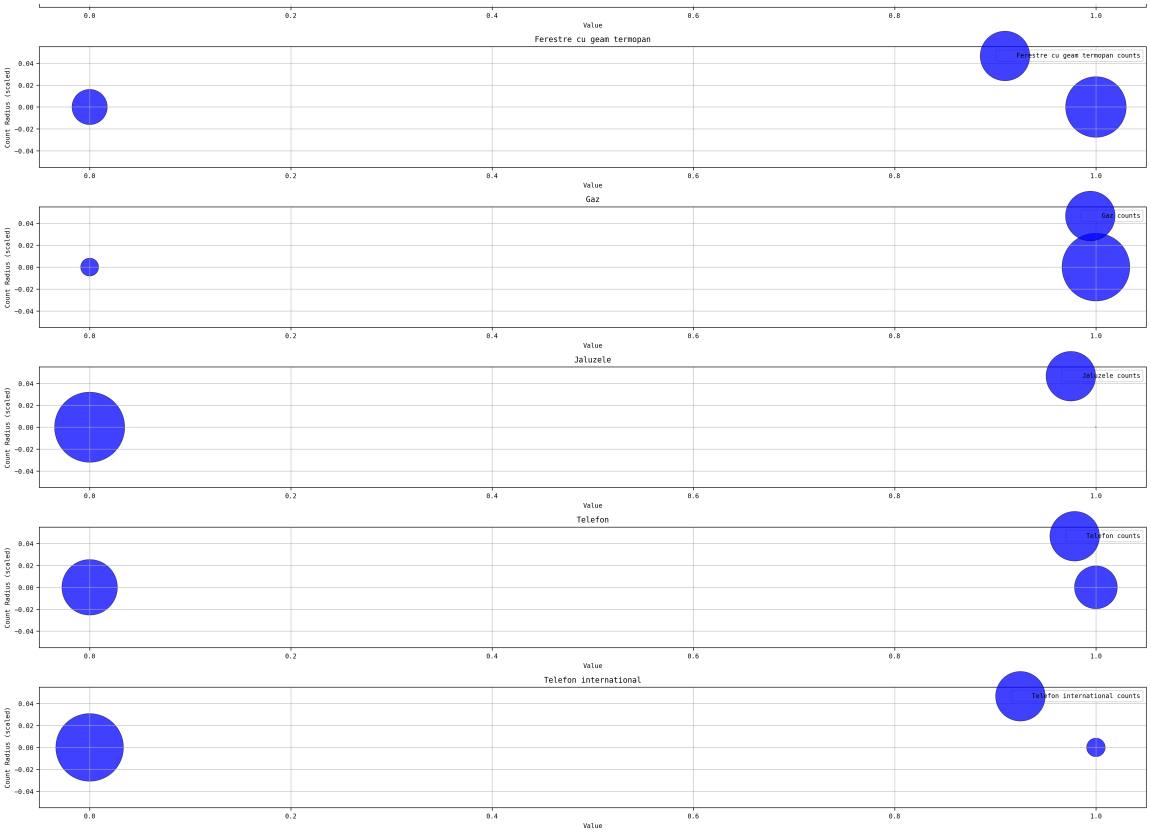
Data usually has dominant patterns and uneven distributions.

I managed to extract these key infos out of these plots:

1. Most common *sector* is 1
2. 2 rooms are the most common
3. Most of the adds are for places at a intermediary floor
4. Most of the adds are for places build after 2020
5. Most of them have *Gaz*, *Telefon* and *Ferestre cu geam termopan*

In [ ]:
```
# lets do the same for the int64 columns
# ball charts
cols = df.select_dtypes('float64').columns
fig, ax = plt.subplots(len(cols), 1, figsize=(20, 50), dpi=300)
# set the font to monospace
plt.rcParams['font.family'] = 'monospace'
# make the font size a bit smaller
plt.rcParams['font.size'] = 8
for idx, col in enumerate(cols):
    counts = df[col].value_counts()
    # draw the ball chart
    ax[idx].scatter(counts.index, np.zeros(len(counts)), alpha=0.75, label=f'{co
    ax[idx].set_title(col)
    ax[idx].set_ylabel('Count Radius (scaled)')
    ax[idx].set_xlabel('Value')
    ax[idx].grid(alpha=0.75)
    ax[idx].legend()
    # remove the ball from the legend
    handles, labels = ax[idx].get_legend_handles_labels()
plt.tight_layout()
```


sector
sector counts
price_sqm
price_sqm counts
Nr. camere:
Nr. camere: counts
Suprafață utilă totală:
Suprafață utilă totală: counts
Nr. bucătării:
Nr. bucătării: counts
Nr. băi:
Nr. băi: counts
Etaj_cod
Etaj_cod counts
An construcție:
An construcție: counts
Nr. balcoane:
Nr. balcoane: counts
Apa
Apa counts
CATV
CATV counts
Canalizare
Canalizare counts
Curent
Curent counts
Count Radius (scaled)
Value

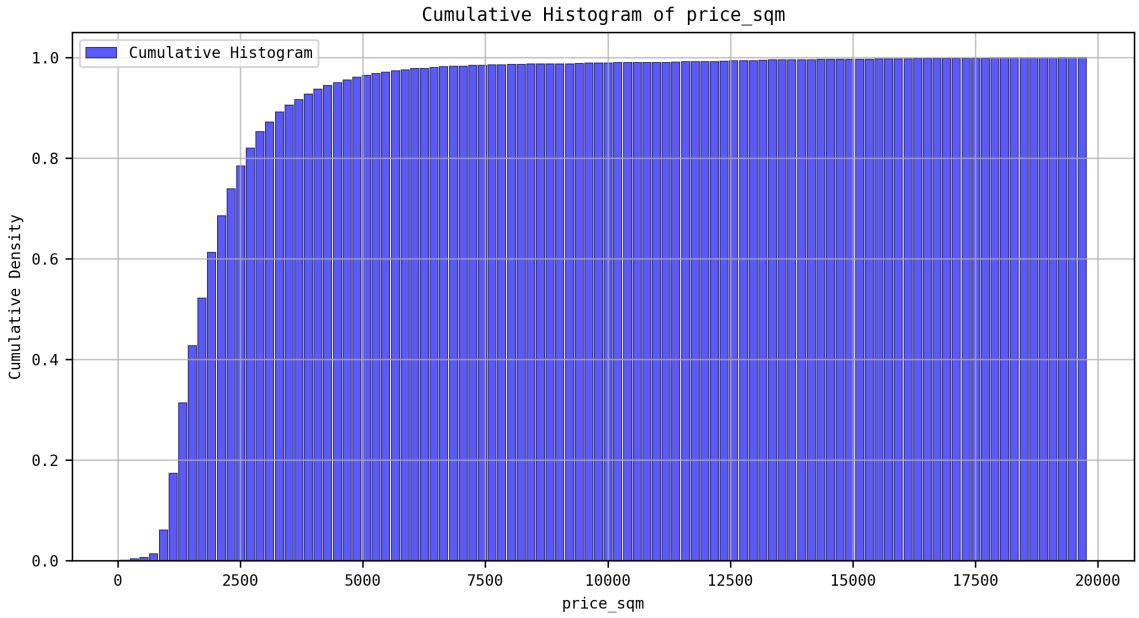


Management wants a feature

Cumulative histogram

- Given the properties an owner inputs, we want our platform to propose them a listing price.
- Based on this listing price owners can see similar properties and choose a better price such that they can close a faster deal
- At the same time, buyers can asses faster if the property is listed fairly given the current market

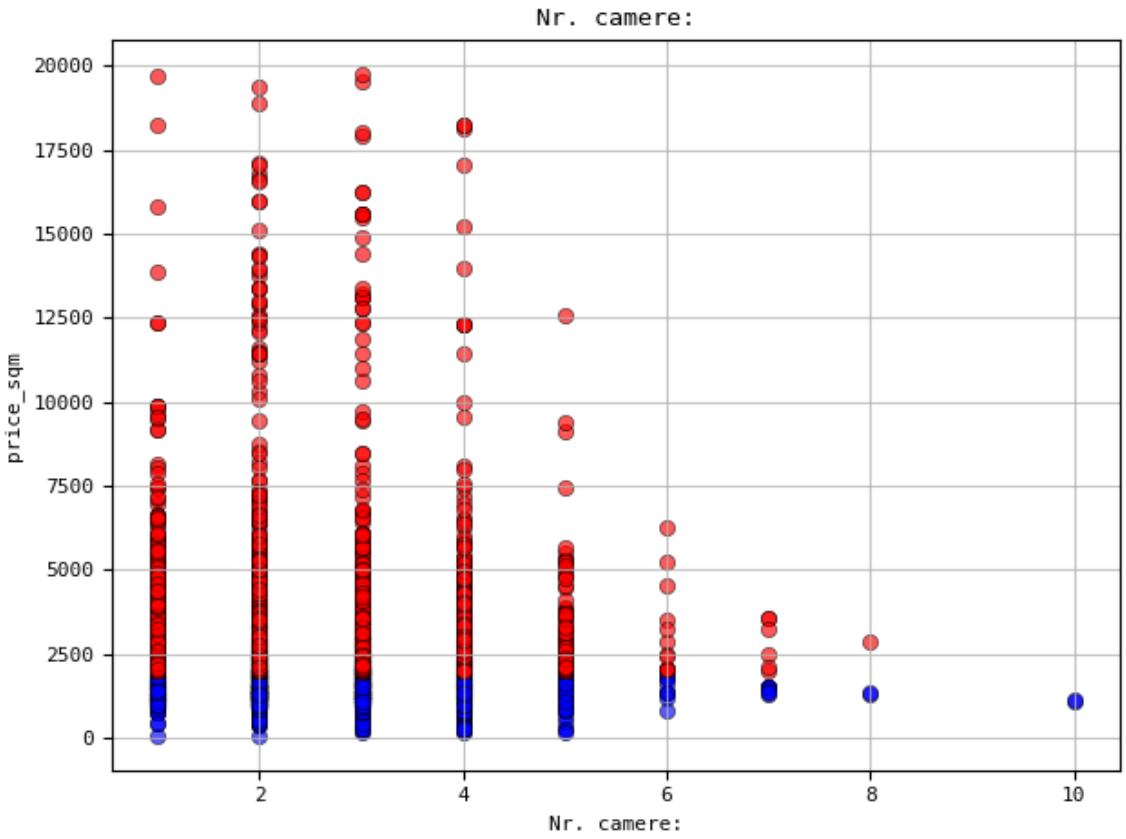
In [ ]:
```
# let's take a look at the cumulative histogram of the `SalePrice` column
fig, ax = plt.subplots(1, 1, figsize=(10, 5), dpi=300)
plt.hist(df['price_sqm'], bins=100, cumulative=True, density=True, alpha=0.65, l
plt.title('Cumulative Histogram of price_sqm')
plt.ylabel('Cumulative Density')
plt.xlabel('price_sqm')
plt.grid(alpha=0.75)
plt.legend()
plt.show()
```

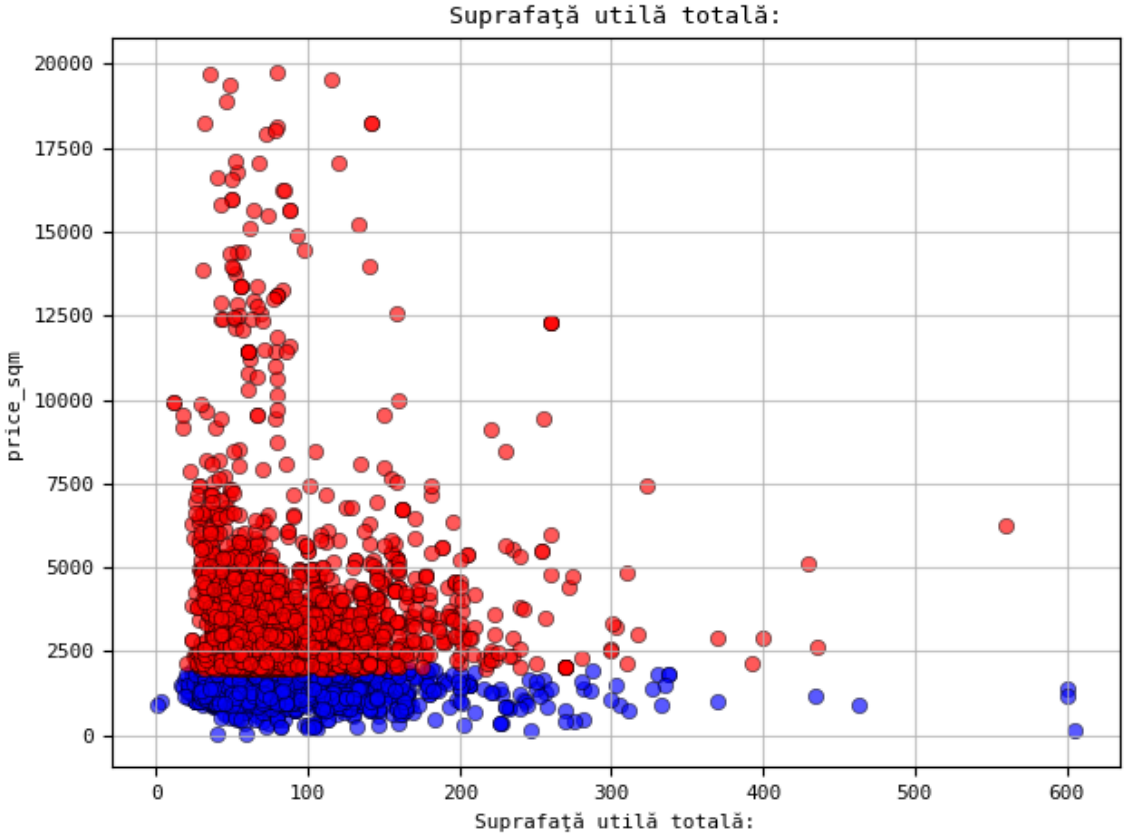


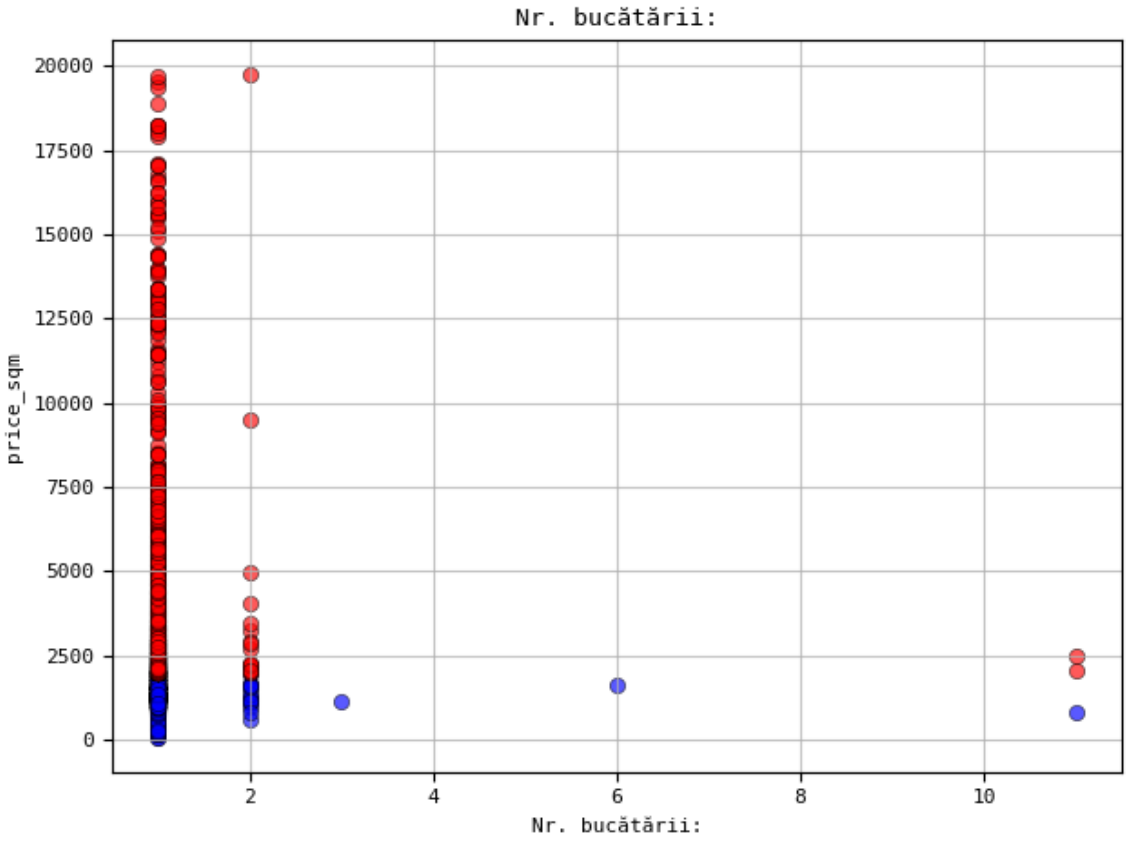
Scatter plots

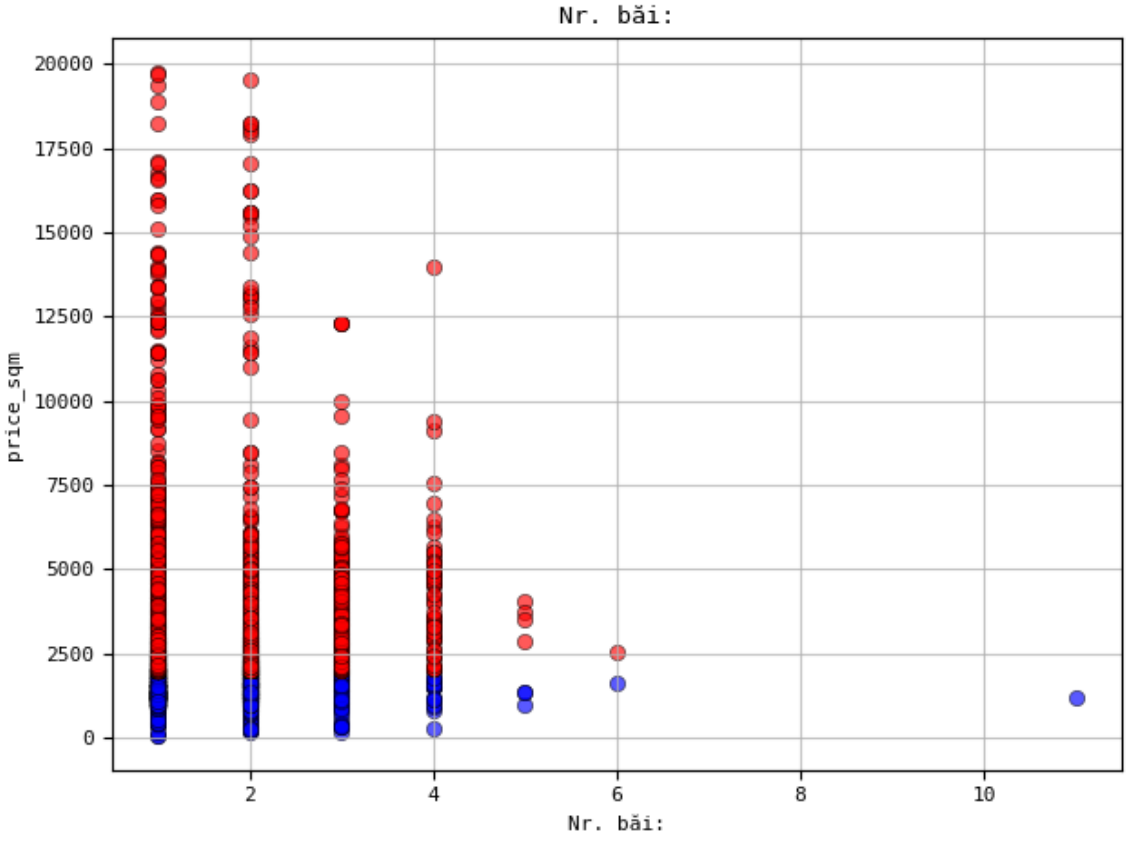
We want to see if there is a linear relationship between the numerical variables and *price_sqm*

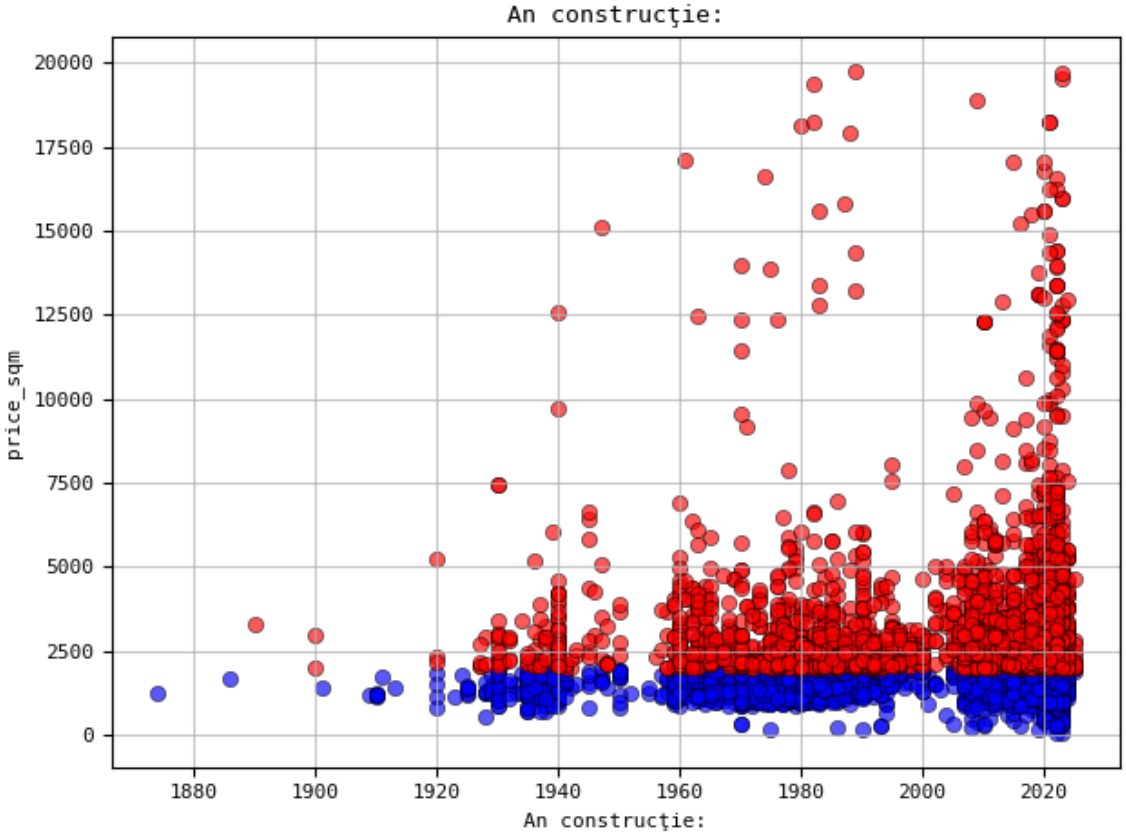
In [ ]:
```
# select some numerical variables and do a scatter plot of the variable vs `pric
# we also want to see if there are any outliers
low_prices = df[df['price_sqm'] < 2000]
high_prices = df[df['price_sqm'] >= 2000]

numerical_variables = ['Nr. camere:', 'Suprafață utilă totală:', 'Nr. bucătării:
df2 = df[numerical_variables]
numerical_variables = df2.select_dtypes(include=['number']).columns.tolist()

# do a scatter plot of the variable vs `price_sqm`
# in blue for the low prices and in red for the high prices
# on X-axis we have the price and on Y-axis we have the variable
for i, variable in enumerate(numerical_variables):
  plt.scatter(low_prices[variable], low_prices['price_sqm'], alpha=0.65, color='
  plt.scatter(high_prices[variable], high_prices['price_sqm'], alpha=0.65, color
  plt.title(variable)
  plt.xlabel(variable)
  plt.ylabel('price_sqm')
  plt.grid(alpha=0.75)

  plt.rcParams['font.family'] = 'monospace'
  plt.tight_layout()
  plt.show()
```

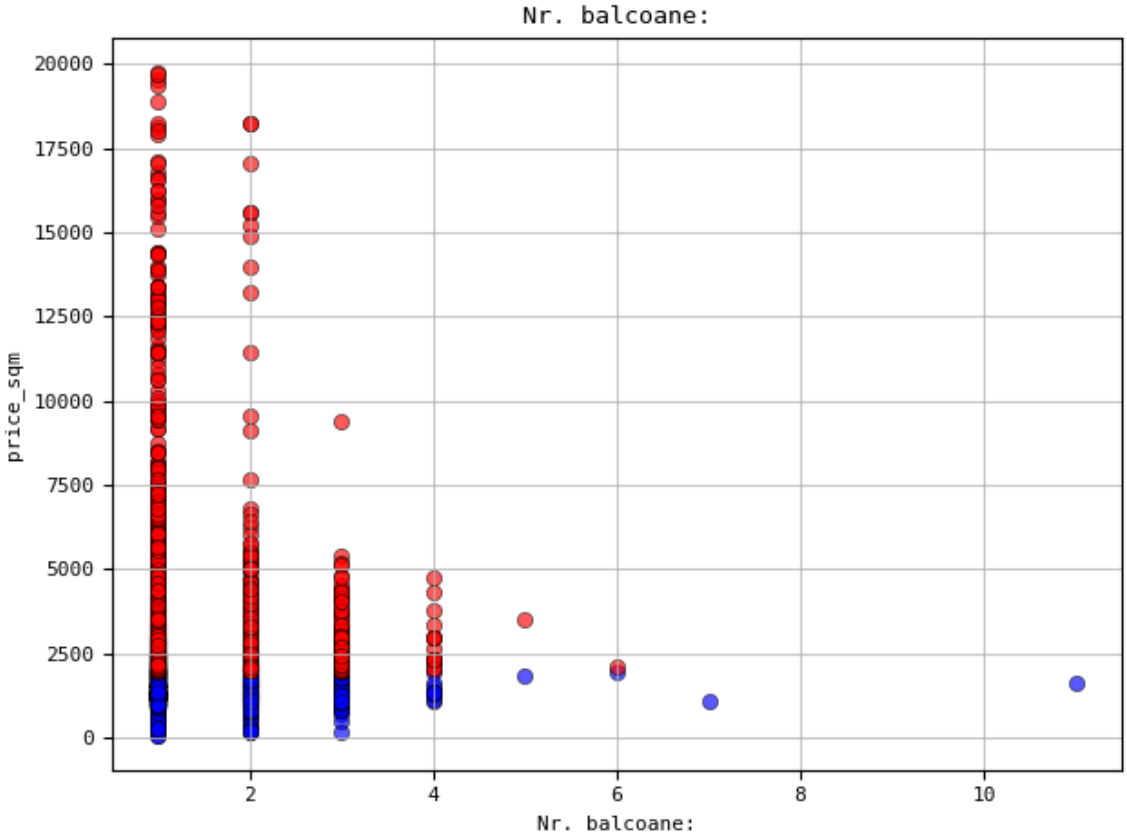

Nr. camere:
price_sqm
Nr. camere:
0
2500
5000
7500
10000
12500
15000
17500
20000
2
4
6
8
10

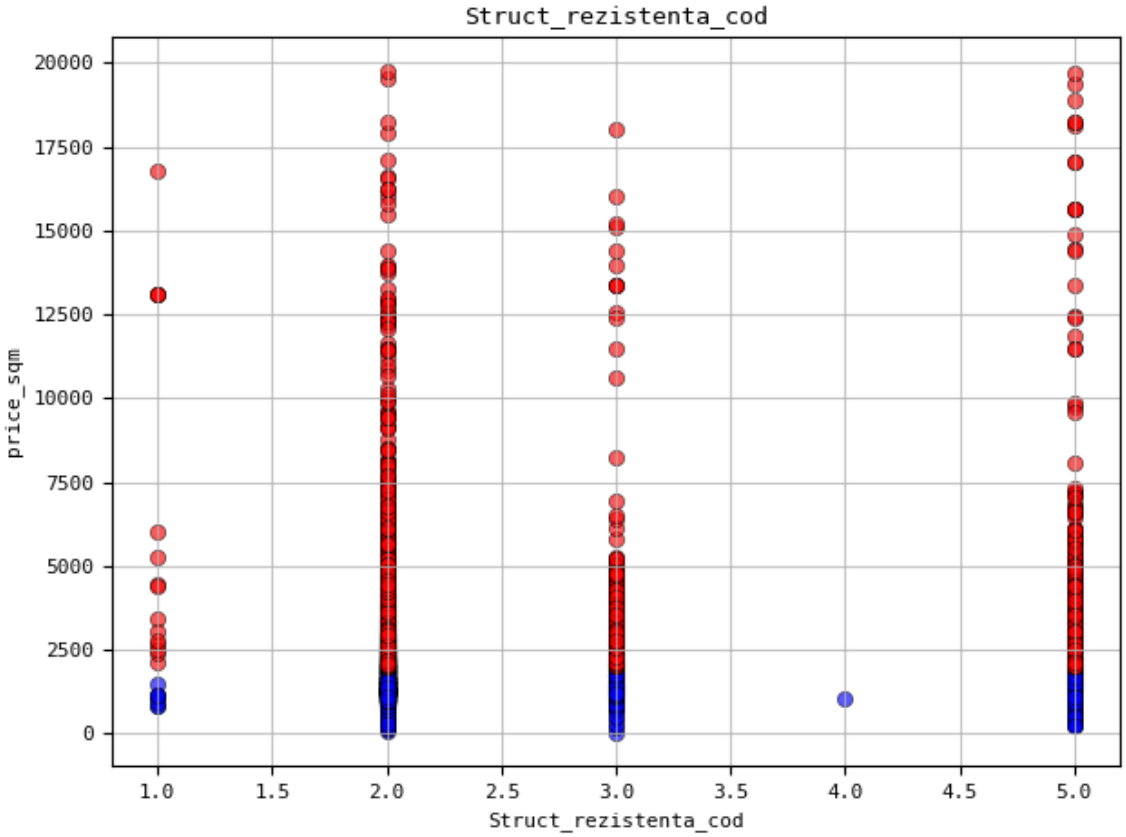

Suprafaţă utilă totală:
price_sqm
Suprafaţă utilă totală:
0
2500
5000
7500
10000
12500
15000
17500
20000
0
100
200
300
400
500
600

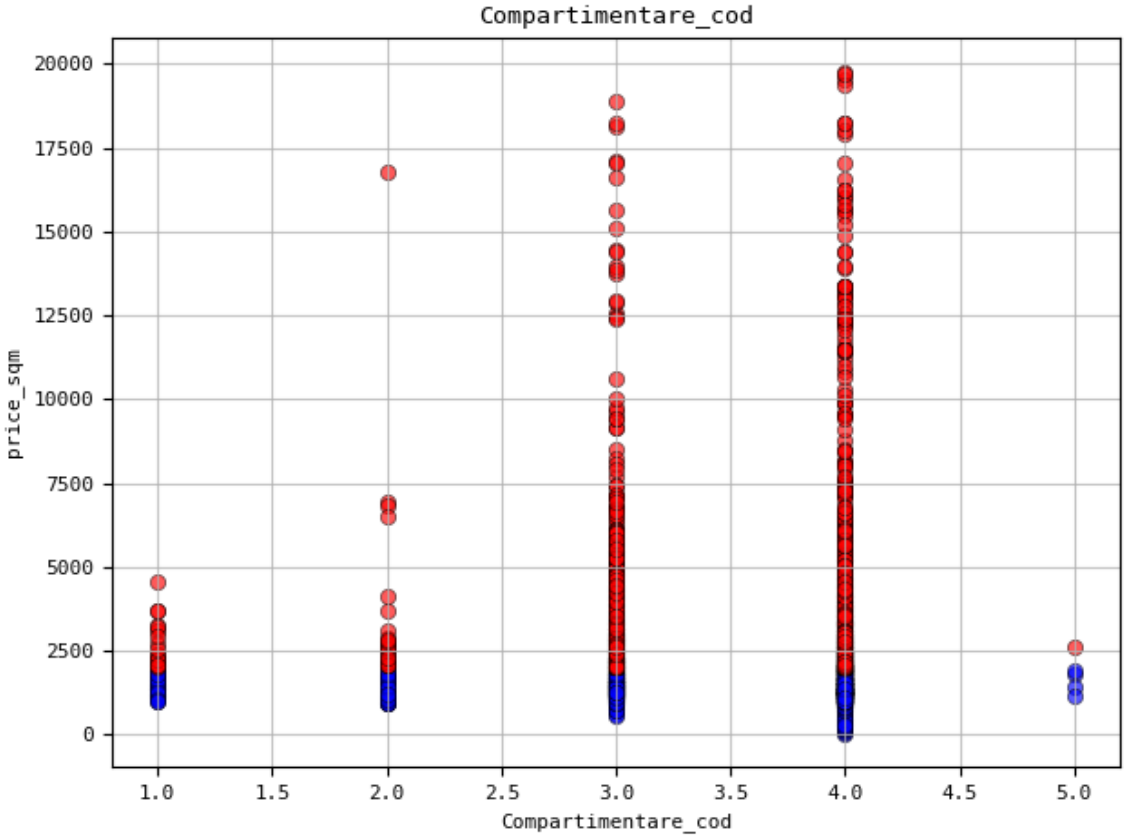

Nr. bucătării:
price_sqm
Nr. bucătării:
0
2500
5000
7500
10000
12500
15000
17500
20000
2
4
6
8
10

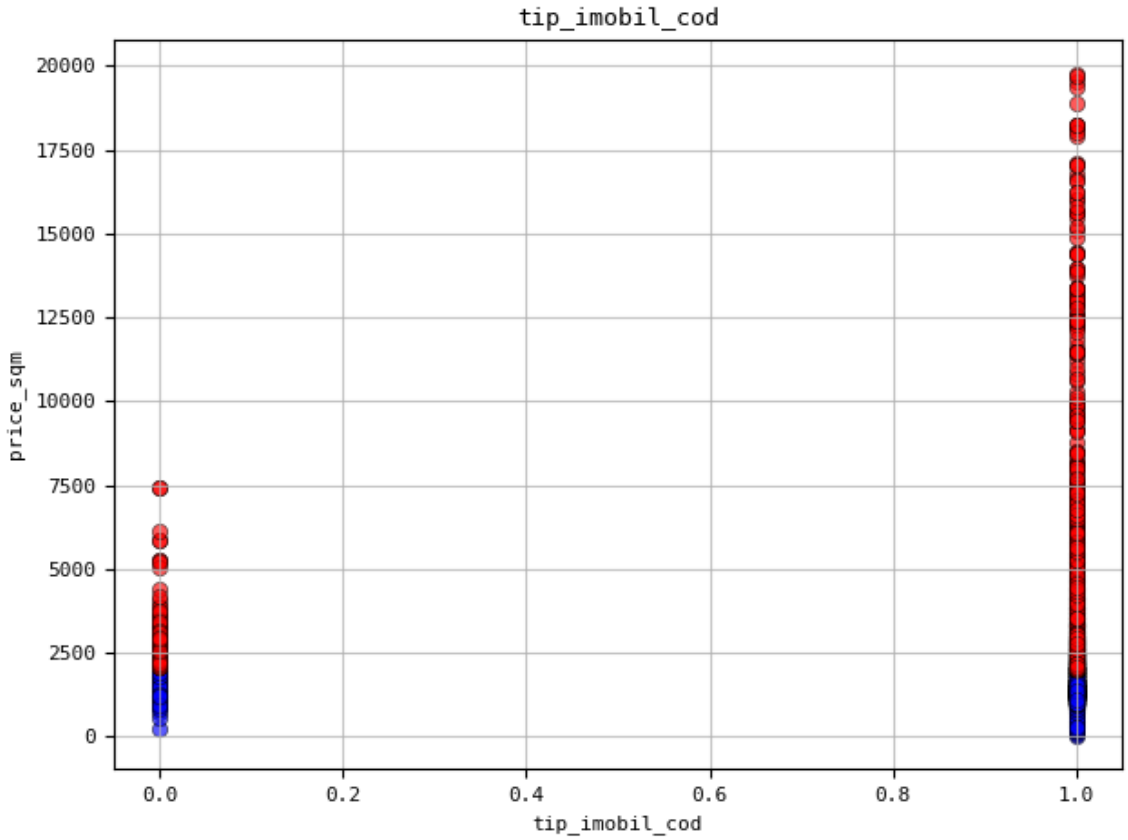

Nr. băi:
price_sqm
Nr. băi:
0
2500
5000
7500
10000
12500
15000
17500
20000
2
4
6
8
10


An construcţie:
price_sqm
An construcţie:
20000
17500
15000
12500
10000
7500
5000
2500
0
1880
1900
1920
1940
1960
1980
2000
2020


Nr. balcoane:
price_sqm
Nr. balcoane:
20000
17500
15000
12500
10000
7500
5000
2500
0
2
4
6
8
10


Struct_rezistenta_cod
price_sqm
Struct_rezistenta_cod
0
2500
5000
7500
10000
12500
15000
17500
20000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Compartimentare_cod
price_sqm
Compartimentare_cod
0
2500
5000
7500
10000
12500
15000
17500
20000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0



More histograms

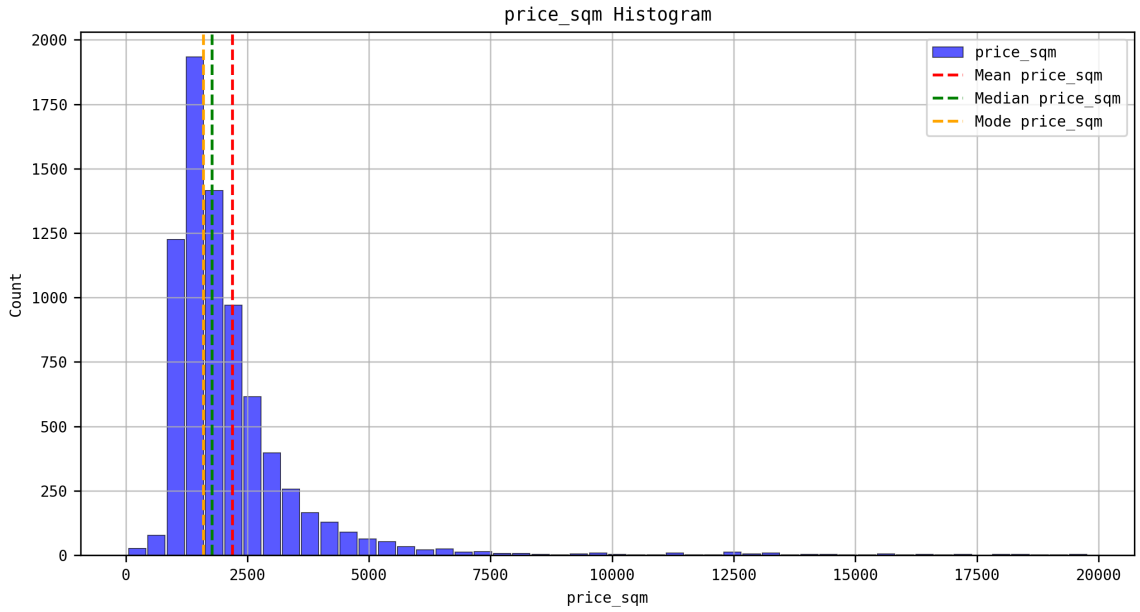
To understand the distribution of price per sq meter

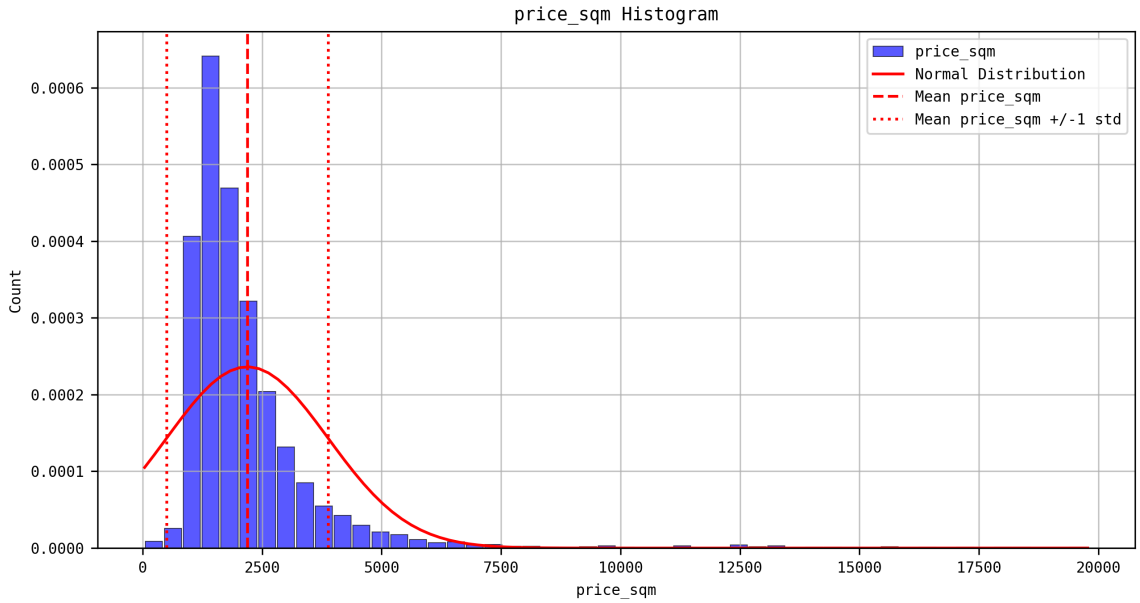
We can see that the majority of them are under 3500

In [ ]:
```
# let's take a look at the histogram of `price_sqm`
fig, ax = plt.subplots(1, 1, figsize=(10, 5), dpi=300)
plt.rcParams['font.family'] = 'monospace'
plt.hist(df['price_sqm'], bins=50, alpha=0.65, color='blue', edgecolor='black',
# add a vertical line at the mean
plt.axvline(df['price_sqm'].mean(), color='red', linestyle='--', label='Mean pri
# add a vertical line at the median
plt.axvline(df['price_sqm'].median(), color='green', linestyle='--', label='Medi
# add a vertical line at the mode
plt.axvline(df['price_sqm'].mode()[0], color='orange', linestyle='--', label='Mo
plt.title('price_sqm Histogram')
plt.xlabel('price_sqm')
plt.ylabel('Count')
plt.grid(alpha=0.75)
plt.legend()

# let's do the same plot whilst also plotting the density curve
fig, ax = plt.subplots(1, 1, figsize=(10, 5), dpi=300)
plt.rcParams['font.family'] = 'monospace'
plt.hist(df['price_sqm'], bins=50, alpha=0.65, color='blue', edgecolor='black',
# plot the density curve
x = np.linspace(df['price_sqm'].min(), df['price_sqm'].max(), 100)
y = stats.norm.pdf(x, df['price_sqm'].mean(), df['price_sqm'].std())
# renormalize the curve to the histogram
plt.plot(x, y, color='red', linestyle='-', label='Normal Distribution')
# add a vertical line at the mean
plt.axvline(df['price_sqm'].mean(), color='red', linestyle='--', label='Mean pri
```

```
# add two vertical lines at the mean +/- 1 std
plt.axvline(df['price_sqm'].mean() + df['price_sqm'].std(), color='red', linesty
plt.axvline(df['price_sqm'].mean() - df['price_sqm'].std(), color='red', linesty
# place horizontal lines between the mean +/- 1 std
plt.title('price_sqm Histogram')
plt.xlabel('price_sqm')
plt.ylabel('Count')
plt.grid(alpha=0.75)
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x19782e8ece0>





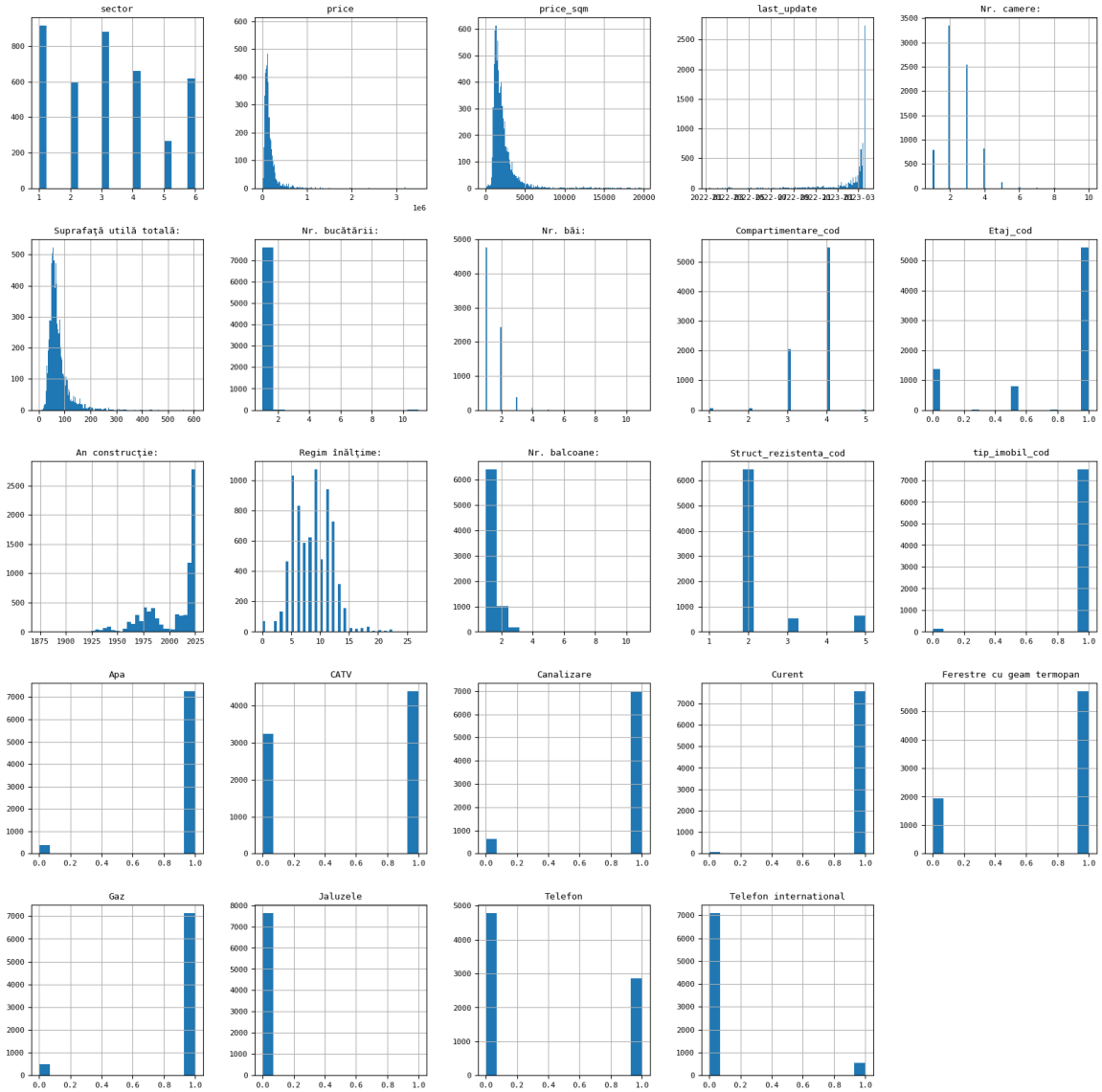
Plots for numerical values

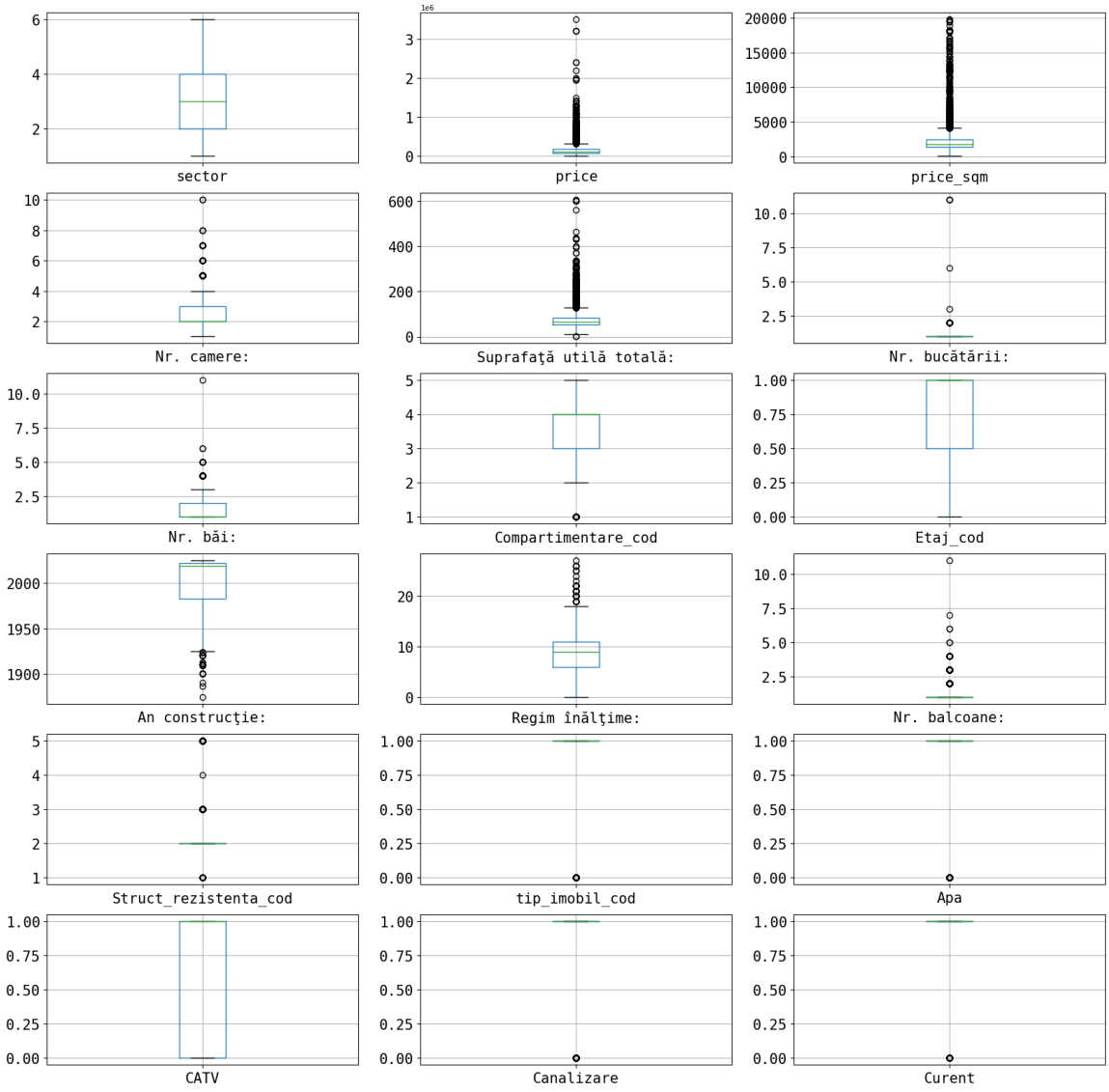
What we can take from here is that most of the households have *Compartimentare_cod, decomandat*.

The rest of them we discovered earlier and some of them are kinda obvious

In [ ]:
```
import seaborn as sns
```

```
# plots to see data
df.hist(bins = "auto", figsize = (20,20))

df2 = df.drop(columns=['last_update'])
not_object = df2.select_dtypes(exclude = object).columns
fig, axes = plt.subplots(nrows = 6, ncols = 3, figsize = (20,20))
# counter
index = 0

for row in range(len(axes)):
    for col in range(len(axes[row])):
        if index == len(not_object):
            break
        ax = axes[row][col]
        var = df2[[not_object[index]]]
        var.boxplot(fontsize = 15, ax = ax)
        index += 1
```





Correlation matrix

Conclusions from this:

1. Price is correlated with surface, no. of rooms and bathrooms
2. We can see that no. of rooms is obviosly very correlated with surface, and also no. of bathrooms
3. We can also see that no. of bathrooms is more correlated with no. of rooms and surface, than no. of balcony is. So it means that it is more likely that a household has more bathrooms than balconies and it is more likely to take into consideration builiding one bathroom than a new balcony, with surface growth.
4. Where is water there is also sewage
5. Construction year is also correlated with *Compartimentare* and *tip_imobil*
6. There is also a strong correlation between *CATV* and *Telefon*

In [ ]:
```
# corelație

f = plt.figure(figsize=(14, 14))
corr_df = df.corr()
corr_df = corr_df - np.diag(np.diag(corr_df))
corr_df = corr_df[corr_df > 0.2]
plt.matshow(corr_df, fignum=f.number)
```

```
plt.xticks(range(df.select_dtypes(['number']).shape[1]), df.select_dtypes(['numb
plt.yticks(range(df.select_dtypes(['number']).shape[1]), df.select_dtypes(['numb
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16)
```

```
C:\Users\grecu\AppData\Local\Temp\ipykernel_2832\3914941515.py:4: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a futu
re version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  corr_df = df.corr()
```

Out[ ]: Text(0.5, 1.0, 'Correlation Matrix')