

# oo\_Data\_Wrangling

October 03, 2023

## 1 flats-in-cracow data wrangling

### 1.1 Imports

```
[1]: import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from collections import Counter
from IPython.display import display
from sklearn.impute import KNNImputer
from pylab import rcParams
from pathlib import Path
```

### 1.2 Setup

```
[2]: # Create directory for images
Path("img").mkdir(parents=True, exist_ok=True)

# Set default figure size
rcParams['figure.figsize'] = (4, 4)

# Tell pandas how to display floats
pd.options.display.float_format = "{:,.2f}".format
```

### 1.3 Goal

I scraped listings of properties for sale in Cracow. We would like to create a model to predict flat prices.

### 1.4 Data source

Data has been scraped from a website with listings. The data has undergone small transformations along the way. The goal of these transformations was to get the data into a usable state not to check it's validity.

## 1.5 Data loading

```
[3]: path = '../flats-data/raw_data.csv'
```

```
[4]: data = pd.read_csv(path, lineterminator='\n')
```

```
[5]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60604 entries, 0 to 60603
Data columns (total 24 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Date             60434 non-null  object
1   City             46536 non-null  object
2   District         33403 non-null  object
3   Amount           60375 non-null  float64
4   Currency         60375 non-null  object
5   Property         60023 non-null  object
6   Seller           60269 non-null  object
7   Area             60118 non-null  float64
8   Rooms            59423 non-null  float64
9   Bathrooms        38847 non-null  float64
10  Parking          26133 non-null  object
11  Garden           60604 non-null  bool
12  Balcony           60604 non-null  bool
13  Terrace           60604 non-null  bool
14  Floor            60604 non-null  bool
15  New              60604 non-null  bool
16  Estate           60604 non-null  bool
17  Townhouse        60604 non-null  bool
18  Apartment         60604 non-null  bool
19  Land             60604 non-null  bool
20  Studio           60604 non-null  bool
21  Title            60434 non-null  object
22  Description       52855 non-null  object
23  Link             60604 non-null  object
dtypes: bool(10), float64(4), object(10)
memory usage: 7.1+ MB
```

First we sort the data in from newest to oldest, forcing rows with missing Date values to be last.

```
[6]: data = data.sort_values(by='Date',
                             ascending=False,
                             na_position='last',
                             ignore_index=True)
```

Next we assume that the Title column uniquely identifies a listing.

```
[7]: data = data.drop_duplicates(['Title'], keep='first')
```

After this the shape of the data is:

```
[8]: print(data.shape)
```

(10484, 24)

## 1.6 Data exploration

We check for missing values that we will have to deal with.

```
[9]: missing = data.isnull().sum(axis=0)
missing.name = 'Missing'
missing = missing.to_frame()
missing = missing[missing['Missing'] > 0]
missing.sort_values('Missing', ascending=False)
```

```
[9]:
```

	Missing
Parking	6604
District	4392
Bathrooms	4187
Description	1900
City	1688
Rooms	232
Area	129
Seller	88
Property	83
Amount	9
Currency	9
Date	1
Title	1

### 1.6.1 Check numeric columns

We see that we have 24 columns at our disposal. We inspect the numeric columns to see what we are dealing with. In the Amount column we note there is a property for sale that costs 1PLN, clearly a erroneous value. Next we note that the enourmous maximum in the Amount column. That is quite a lot of money and could be considered a potential outlier. The maximum and minimum of the Area column also indicate the existance of outliers. These values are clearly too large. The data will need to undergo a filtering process.

```
[10]: data.describe()
```

```
[10]:
```

	Amount	Area	Rooms	Bathrooms
count	10,475.00	10,355.00	10,252.00	6,297.00
mean	722,001.88	132.23	2.92	1.32
std	5,139,332.07	3,562.58	1.32	0.63

min	100.00	1.00	1.00	1.00
25%	395,000.00	43.00	2.00	1.00
50%	499,400.00	56.00	3.00	1.00
75%	720,000.00	80.00	4.00	1.00
max	521,290,000.00	320,000.00	6.00	4.00

### 1.6.2 Check binary columns

We inspect the data to see if binary columns are properly populated and check for imbalances.

```
[11]: binary = data.select_dtypes(bool).columns.to_list()
```

```
for col in binary:
    tmp = data[[col, 'Amount']]
    tmp = tmp.fillna('NaN')
    tmp = tmp.groupby(col, as_index=False)
    tmp = tmp.count()
    tmp = tmp.rename(columns={'Amount': 'Count'})
    tmp = tmp.sort_values('Count', ascending=False)
    tmp = tmp.reset_index(drop=True)
    display(tmp)
```

	Garden	Count
0	False	8407
1	True	2077

	Balcony	Count
0	False	6816
1	True	3668

	Terrace	Count
0	False	9237
1	True	1247

	Floor	Count
0	False	6398
1	True	4086

	New	Count
0	False	7090
1	True	3394

	Estate	Count
0	False	8947
1	True	1537

	Townhouse	Count
0	False	9576
1	True	908

	Apartment	Count
0	False	8960
1	True	1524

	Land	Count
0	False	8047
1	True	2437

	Studio	Count
0	False	9788
1	True	696

### 1.6.3 Check categorical columns

We inspect categorical columns to assert that they contain “valid” values. Most of these columns were generated by a script during the scraping and etl phase of the project.

```
[12]: categorical = data.select_dtypes('object').columns
categorical = categorical.to_list()
omit = ['Title', 'Link', 'Description', 'Date']

for col in categorical:
    if col not in omit:
        tmp = data[['Amount', col]].copy()
        tmp = tmp.fillna('NaN')
        tmp = tmp.groupby(col, as_index=False)
        tmp = tmp.count()
        tmp = tmp.rename(columns={'Amount': 'Count'})
        tmp = tmp.sort_values('Count', ascending=False)
        tmp = tmp.reset_index(drop=True)
        display(tmp)
```

	City	Count
0	kraków	8796
1	NaN	1688

	District	Count
0	NaN	4392
1	krowodrza	813
2	stare miasto	696

3	podgorze	641
4	nowa huta	455
5	debniki	442
6	bronowice	435
7	pradnik bialy	426
8	pradnik czerwony	323
9	biezanow	318
10	grzegorzki	306
11	czyzyny	235
12	mistrzejowice	203
13	lagiewniki	171
14	zwierzyniec	151
15	podgorze duchackie	132
16	bienczyce	120
17	swoszowice	106
18	prokocim	62
19	borek falecki	34
20	wzgorza krzeslawickie	23

	Currency	Count
0	pln	10475
1	NaN	9

	Property	Count
0	flat	9015
1	house	1386
2	NaN	83

	Seller	Count
0	realtor	9598
1	owner	798
2	NaN	88

	Parking	Count
0	NaN	6604
1	street	1519
2	garage	1516
3	no parking	651
4	covered	194

#### 1.6.4 Check text columns

We search for keywords in the data.

```
[13]: # text = data[data['Description'].isna() == False].copy()
# text = text['Description'].to_list()
# text = ' '.join(text)
# text = text.split(' ')
# text = [x for x in text if x.isalpha()]
# text = [x for x in text if len(x) > 3]
```

```
[14]: # for i in range(5, len(text)-5):
#     if 'piętro' in text[i]:
#         s = text[i-5:i+5]
#         s = ' '.join(s)
#         print(s)
```

## 1.7 Data cleaning

We assume that if we know the district, the City is kraków.

```
[15]: mask = (data['City'].isna() == True) & (data['District'].isna() == False)
data.loc[mask, 'City'] = 'kraków'
```

We extract more Parking information from the property description.

```
[16]: def extract_parking(x):
    if ('garaż' in x or 'garaz' in x or 'parking' in x) and 'podziemny' in x:
        return 'covered'
    elif ('garaż' in x or 'garaz' in x) and 'podziemny' not in x:
        return 'garage'
    elif 'parking' in x and 'podziemny' not in x:
        return 'street'
    else:
        return 'no parking'
```

```
[17]: mask = (data['Parking'].isna() == True) & (data['Description'].isna() == False)
data.loc[mask, ['Parking', 'Description']] = data.loc[mask, 'Description'].
    ↪.apply(extract_parking)
```

```
[18]: mask = data['Parking'].isna() == True
data.loc[mask, 'Parking'] = 'no parking'
```

We confirm that we have dealt with all the NaNs in the Parking column.

```
[19]: print(data['Parking'].isna().sum())
```

0

### 1.7.1 Filtering

Next we filter the data according to these rules:

```
[20]: data = data[data['City'] == 'kraków']
data = data[data['Currency'] == 'pln']
data = data[data['Property'] == 'flat']
data = data[(data['Amount'] >= data['Amount'].quantile(0.025))]
data = data[(data['Amount'] <= data['Amount'].quantile(0.975))]
data = data[(data['Area'] >= data['Area'].quantile(0.01))]
data = data[(data['Area'] <= data['Area'].quantile(0.99))]
data = data[data['District'] != 'unknown']
data = data[data['District'].isna() == False]
data = data[data['Seller'].isna() == False]
data = data[data['Description'].isna() == False]
```

```
[21]: data = data.reset_index(drop=True)
```

```
[22]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4592 entries, 0 to 4591
Data columns (total 24 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Date            4592 non-null   object
1   City            4592 non-null   object
2   District        4592 non-null   object
3   Amount          4592 non-null   float64
4   Currency        4592 non-null   object
5   Property        4592 non-null   object
6   Seller          4592 non-null   object
7   Area            4592 non-null   float64
8   Rooms           4536 non-null   float64
9   Bathrooms       2238 non-null   float64
10  Parking         4592 non-null   object
11  Garden          4592 non-null   bool
12  Balcony         4592 non-null   bool
13  Terrace         4592 non-null   bool
14  Floor           4592 non-null   bool
15  New             4592 non-null   bool
16  Estate          4592 non-null   bool
17  Townhouse       4592 non-null   bool
18  Apartment       4592 non-null   bool
19  Land            4592 non-null   bool
20  Studio          4592 non-null   bool
21  Title           4592 non-null   object
22  Description     4592 non-null   object
23  Link            4592 non-null   object
dtypes: bool(10), float64(4), object(10)
memory usage: 547.2+ KB
```



### 1.7.2 Impute missing values

The next step is to fill in missing values for numeric columns Amount Area Rooms and Bathrooms. We use the KNNImputer to accomplish this.

```
[23]: numeric = list(data.select_dtypes('number').columns)

[24]: mask = (data['Bathrooms'].isna() == True | data['Rooms'].isna())
      missing = data[numeric]

      imputer = KNNImputer(n_neighbors=5)
      imputer.fit(missing)

      missing = imputer.transform(missing)
      missing = pd.DataFrame(missing, columns=numeric)

      for col in numeric:
          data[col] = missing[col]

      for col in numeric:
          data[col] = data[col].apply(lambda x: round(x))
```

```
[25]: print(data.shape)
```

(4592, 24)

### 1.8 Save data

Verify that there are no NaNs in data.

```
[26]: data.isnull().sum().sum()
```

[26]: 0

Remove columns that will not be used further.

```
[27]: data = data.drop(['Title',
                        'Description',
                        'Link',
                        'Property',
                        'City',
                        'Currency',
                        'Date'], axis=1)
```

Take a last peek at the data.

```
[28]: data.head()
```

```
[28]:
```

	District	Amount	Seller	Area	Rooms	Bathrooms	Parking	Garden \
0	krowodrza	595000	realtor	78	4	2	no parking	False
1	podgorze	449000	realtor	61	3	1	no parking	False
2	nowa huta	449000	realtor	58	3	1	no parking	False
3	krowodrza	595000	realtor	78	4	2	no parking	False
4	krowodrza	430000	realtor	48	2	1	garage	False

  

	Balcony	Terrace	Floor	New	Estate	Townhouse	Apartment	Land	Studio
0	True	False	False	False	False	False	False	False	False
1	True	False	True	False	False	False	False	False	False
2	True	False	False	True	False	False	False	False	False
3	True	False	False	False	False	False	False	False	False
4	True	False	True	False	True	False	False	False	False

```
[29]: data.describe()
```

```
[29]:
```

	Amount	Area	Rooms	Bathrooms
count	4,592.00	4,592.00	4,592.00	4,592.00
mean	535,522.19	55.93	2.61	1.10
std	222,331.92	20.25	0.99	0.33
min	214,000.00	22.00	1.00	1.00
25%	390,000.00	41.00	2.00	1.00
50%	470,000.00	53.00	3.00	1.00
75%	618,775.00	66.00	3.00	1.00
max	1,525,000.00	135.00	6.00	4.00

Save it for further analysis.

```
[30]: data.to_csv('../flats-data/cleaned_data.csv', index=False)
```