00_Data_Wrangling

September 24, 2020

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: 53110 entries, 0 to 53109
Data columns (total 24 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         52975 non-null  object
 1   City         40097 non-null  object
 2   District     29317 non-null  object
 3   Amount       52924 non-null  float64
 4   Currency     52924 non-null  object
 5   Property     52612 non-null  object
 6   Seller       52825 non-null  object
 7   Area         52683 non-null  float64
 8   Rooms        52071 non-null  float64
 9   Bathrooms    34026 non-null  float64
 10  Parking      22575 non-null  object
 11  Garden       53110 non-null  bool
 12  Balcony      53110 non-null  bool
 13  Terrace      53110 non-null  bool
 14  Floor        53110 non-null  bool
 15  New          53110 non-null  bool
 16  Estate       53110 non-null  bool
 17  Townhouse    53110 non-null  bool
 18  Apartment    53110 non-null  bool
 19  Land         53110 non-null  bool
 20  Studio       53110 non-null  bool
 21  Title        52975 non-null  object
 22  Description  46442 non-null  object
 23  Link         53110 non-null  object
dtypes: bool(10), float64(4), object(10)
memory usage: 6.2+ 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]:
```python
data = data.drop_duplicates(['Title'], keep='first')
```

After this the shape of the data is:

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

```
(9363, 24)
```

1.6 Data exploration

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

[9]:
```python
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	5867
District	3944
Bathrooms	3782
Description	1675
City	1565
Rooms	223
Area	121
Property	78
Seller	77
Amount	6
Currency	6
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]:
```python
data.describe()
```

[10]:

	Amount	Area	Rooms	Bathrooms
count	9,357.00	9,242.00	9,140.00	5,581.00
mean	730,605.29	138.30	2.91	1.32
std	5,435,283.74	3,770.84	1.32	0.63

min	100.00	1.00	1.00	1.00
25%	395,000.00	42.00	2.00	1.00
50%	499,900.00	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	7564
1	True	1799

	Balcony	Count
0	False	6137
1	True	3226

	Terrace	Count
0	False	8275
1	True	1088

	Floor	Count
0	False	5767
1	True	3596

	New	Count
0	False	6370
1	True	2993

	Estate	Count
0	False	8021
1	True	1342

	Townhouse	Count
0	False	8539
1	True	824

	Apartment	Count
0	False	8016
1	True	1347

	Land	Count
0	False	7207
1	True	2156

	Studio	Count
0	False	8767
1	True	596

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	7798
1	NaN	1565

	District	Count
0	NaN	3944
1	krowodrza	724
2	stare miasto	623

```
3                podgorze    591
4               nowa huta    414
5                 debniki    399
6               bronowice    388
7           pradnik bialy    360
8                biezanow    291
9        pradnik czerwony    284
10             grzegorzki    269
11                czyzyny    207
12          mistrzejowice    169
13             lagiewniki    138
14            zwierzyniec    129
15     podgorze duchackie    120
16              bienczyce    109
17             swoszowice     96
18               prokocim     59
19          borek falecki     31
20  wzgorza krzeslawickie     18
```

```
  Currency  Count
0      pln   9357
1      NaN      6
```

```
  Property  Count
0     flat   8055
1    house   1230
2      NaN     78
```

```
    Seller  Count
0  realtor   8569
1    owner    717
2      NaN     77
```

```
      Parking  Count
0         NaN   5867
1      street   1418
2      garage   1331
3  no parking    573
4     covered    174
```

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 `NaN`s 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: 4102 entries, 0 to 4101
Data columns (total 24 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         4102 non-null   object
 1   City         4102 non-null   object
 2   District     4102 non-null   object
 3   Amount       4102 non-null   float64
 4   Currency     4102 non-null   object
 5   Property     4102 non-null   object
 6   Seller       4102 non-null   object
 7   Area         4102 non-null   float64
 8   Rooms        4047 non-null   float64
 9   Bathrooms    1971 non-null   float64
 10  Parking      4102 non-null   object
 11  Garden       4102 non-null   bool
 12  Balcony      4102 non-null   bool
 13  Terrace      4102 non-null   bool
 14  Floor        4102 non-null   bool
 15  New          4102 non-null   bool
 16  Estate       4102 non-null   bool
 17  Townhouse    4102 non-null   bool
 18  Apartment    4102 non-null   bool
 19  Land         4102 non-null   bool
 20  Studio       4102 non-null   bool
 21  Title        4102 non-null   object
 22  Description  4102 non-null   object
 23  Link         4102 non-null   object
dtypes: bool(10), float64(4), object(10)
memory usage: 488.8+ 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)
```

```
(4102, 24)
```

1.8 Save data

Verify that there are no `NaN`s 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	biezanow	439082	realtor	56	3	1	covered	True	
1	podgorze	845000	realtor	132	5	2	no parking	False	
2	podgorze	360000	realtor	41	2	1	no parking	False	
3	krowodrza	1190000	realtor	81	3	1	no parking	True	
4	debniki	990000	realtor	93	4	2	street	False	

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

[29]:

```
data.describe()
```

[29]:

	Amount	Area	Rooms	Bathrooms
count	4,102.00	4,102.00	4,102.00	4,102.00
mean	532,119.80	55.62	2.60	1.10
std	221,445.75	20.06	0.99	0.32
min	210,000.00	23.00	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%	610,000.00	66.00	3.00	1.00
max	1,525,000.00	134.00	6.00	4.00

Save it for further analysis.

[30]:

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