Rolando_01_Cleaning

December 7, 2022

1 CS3300 Lab 1: Data Cleaning

Jackson Rolando

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

# function for printing dictionaries so the notebook PDF isn't a novel
def print_dict(dict, n):
    keys = list(dict.keys())
    vals = list(dict.values())
    i = 0
    while (i < n and i < len(keys)):
        print(str(keys[i]) + ": " + str(vals[i]))
        i += 1
    if(i < len(keys)):
        print('...')
```

1.1 1. Loading the Data

[2]:
```
df_rets = pd.read_csv('./Sacramentorealestatetransactions.csv')
print(df_rets.head())
print(df_rets.info())
```

```
            street        city    zip state  beds  baths  sq__ft  \
0     3526 HIGH ST  SACRAMENTO  95838    CA     2      1     836
1      51 OMAHA CT  SACRAMENTO  95823    CA     3      1    1167
2   2796 BRANCH ST  SACRAMENTO  95815    CA     2      1     796
3  2805 JANETTE WAY  SACRAMENTO  95815    CA     2      1     852
4   6001 MCMAHON DR  SACRAMENTO  95824    CA     2      1     797

          type                     sale_date  price   latitude   longitude
0  Residential  Wed May 21 00:00:00 EDT 2008  59222  38.631913 -121.434879
1  Residential  Wed May 21 00:00:00 EDT 2008  68212  38.478902 -121.431028
2  Residential  Wed May 21 00:00:00 EDT 2008  68880  38.618305 -121.443839
3  Residential  Wed May 21 00:00:00 EDT 2008  69307  38.616835 -121.439146
4  Residential  Wed May 21 00:00:00 EDT 2008  81900  38.519470 -121.435768
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   street     985 non-null    object
 1   city       985 non-null    object
 2   zip        985 non-null    int64
 3   state      985 non-null    object
 4   beds       985 non-null    int64
 5   baths      985 non-null    int64
 6   sq__ft     985 non-null    int64
 7   type       985 non-null    object
 8   sale_date  985 non-null    object
 9   price      985 non-null    int64
 10  latitude   985 non-null    float64
 11  longitude  985 non-null    float64
dtypes: float64(2), int64(5), object(5)
memory usage: 92.5+ KB
None
```

According to the info() call, there are no fields in any of the rows with a null value. This could not be the case, since there are zeros in some of the numerical columns, which won't make sense and will need to be addressed.

Columns like state, baths, bed, zip, and city could be use more categorically, while square footage will probably be used as a numerical variable.

The columns with types labelled "object" in the info output are assumed to be strings.

1.2 2. Representing Categorical Variables

This function turns a column into a dictionary, with the series's elements as the keys, and the counts of each of those elements as the value.

[3]:
```python
def makeCountDict(series):
    out_dict = {}
    for value in series:
        out_dict.setdefault(value, 0)
        out_dict[value] += 1
    return out_dict
```

This isolates the street names, getting rid of the address numbers and apartment numbers for a more accurate count of each street name, then uses the previous function to count

[4]:
```python
def isolate_street_name(address: str):
    start = address.find(' ') + 1
    end = address.find('Unit')
    return address[(start):(None if end == -1 else end - 1)]
```

```
dirty_street_names =  df_rets['street'].map(lambda address:␣
 ↪isolate_street_name(address))
streets_dict = makeCountDict(dirty_street_names)
print("Counts of unique street names:")
print_dict(streets_dict, 15)
```

```
Counts of unique street names:
HIGH ST: 3
OMAHA CT: 1
BRANCH ST: 1
JANETTE WAY: 1
MCMAHON DR: 1
PEPPERMILL CT: 1
OGDEN NASH WAY: 1
19TH AVE: 1
TRINITY RIVER DR: 1
10TH ST: 1
MORRISON AVE: 3
FAWN CIR: 1
LA ROSA RD: 1
KIRK WAY: 1
LOCH HAVEN WAY: 2
…
```

Counting unique zip codes:

[5]:
```
print_dict(makeCountDict(df_rets['zip']), 15)
```

```
95838: 37
95823: 61
95815: 18
95824: 12
95841: 7
95842: 22
95820: 23
95670: 21
95673: 13
95822: 24
95621: 28
95833: 20
95660: 21
95834: 22
95843: 33
…
```

Counting unique beds:

[6]:
```
makeCountDict(df_rets['beds'])
```

[6]: {2: 133, 3: 413, 1: 10, 4: 258, 0: 108, 5: 59, 8: 1, 6: 3}

The street name and zip codes could be converted to numerical values, but it wouldn't be very useful. Aggregations of either set wouldn't make sense, nor would an average. Street names are arbitrarily decided, there is no scale to move along. Beds could be either. They could be categorical, but it makes sense to use them as a numerical value, as it would be a reassonable assumption that the price of a property would generally go up with the number of beds. It could be useful as a categorical value as well, since an intermediate value like 3.5 beds wouldn't make much sense.

This cell converts the city, state, zip, beds, baths, and type columns to the categorical type:

[7]:
```
cols = ['city', 'state', 'zip', 'beds', 'baths', 'type']
df_rets[cols] = df_rets[cols].astype('category')
df_rets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   street     985 non-null    object
 1   city       985 non-null    category
 2   zip        985 non-null    category
 3   state      985 non-null    category
 4   beds       985 non-null    category
 5   baths      985 non-null    category
 6   sq__ft     985 non-null    int64
 7   type       985 non-null    category
 8   sale_date  985 non-null    object
 9   price      985 non-null    int64
 10  latitude   985 non-null    float64
 11  longitude  985 non-null    float64
dtypes: category(6), float64(2), int64(2), object(2)
memory usage: 56.9+ KB
```

1.3 3. Cleaning Continuous Variables

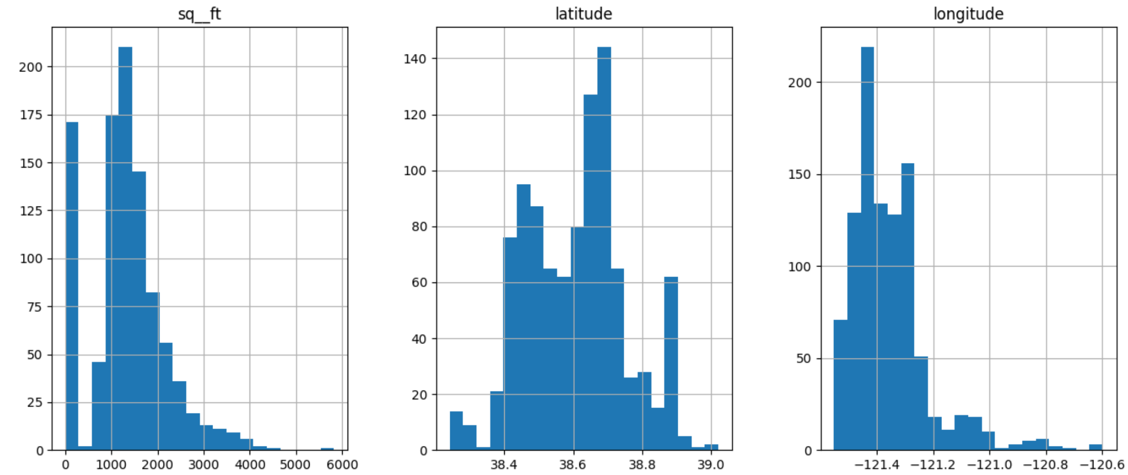
Here is a histogram of square footage, latitudes, and longitudes:

[8]:
```
df_rets.hist(column=['sq__ft', 'latitude', 'longitude'], bins=20, layout=(1,␣
↪3), ax=plt.figure(figsize = (15, 6)).gca())
```

```
/tmp/ipykernel_57606/1110506770.py:1: UserWarning: To output multiple subplots,
the figure containing the passed axes is being cleared.
  df_rets.hist(column=['sq__ft', 'latitude', 'longitude'], bins=20, layout=(1,
3), ax=plt.figure(figsize = (15, 6)).gca())
```

[8]:
```
array([[<AxesSubplot: title={'center': 'sq__ft'}>,
        <AxesSubplot: title={'center': 'latitude'}>,
```

```
<AxesSubplot: title={'center': 'longitude'}>]], dtype=object)
```



A histogram with ranged buckets to show counts makes sense here.

The square footage histogram seems to have a spike way down low. Are these just zero values?

[9]:
```
df_rets.sq__ft[df_rets.sq__ft == 0].count()
```

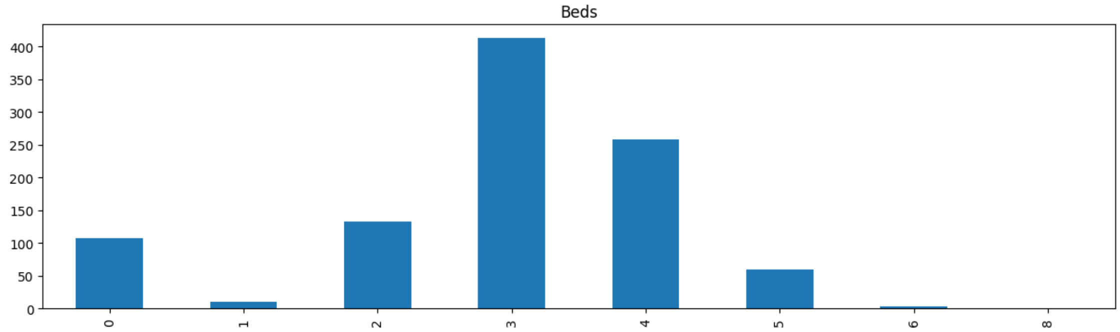
[9]: 171

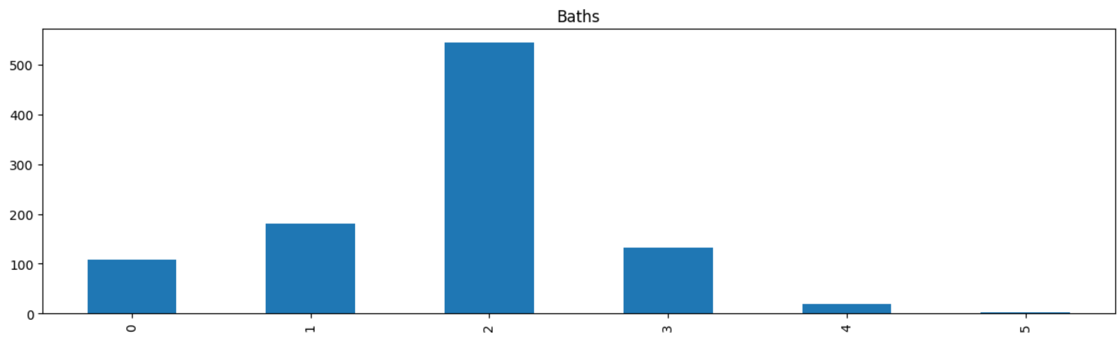
There seems to be 171 proprties whose value for square footage is 0. If nothing is built on the property this makes sense, but it could cause odd results when combined with the non-empty lots, so it'll have to be addressed for certain calculations.

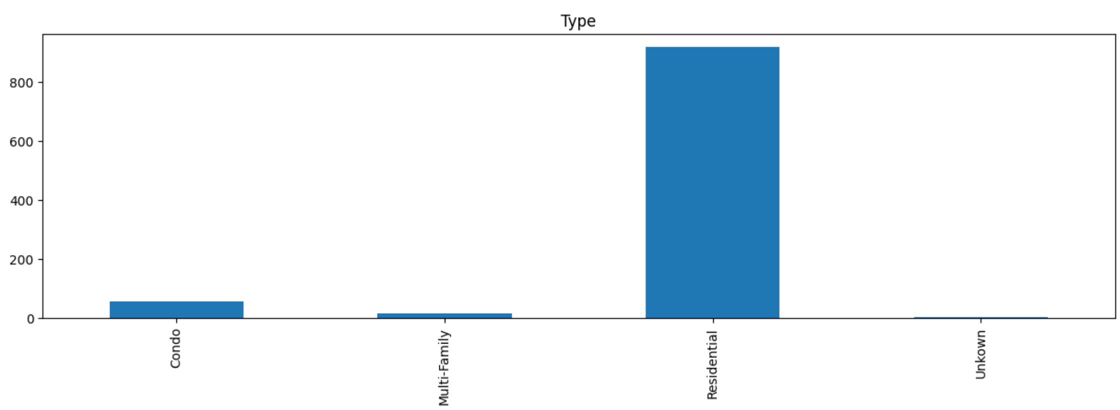
1.4 4. Cleaning Categorical Variables

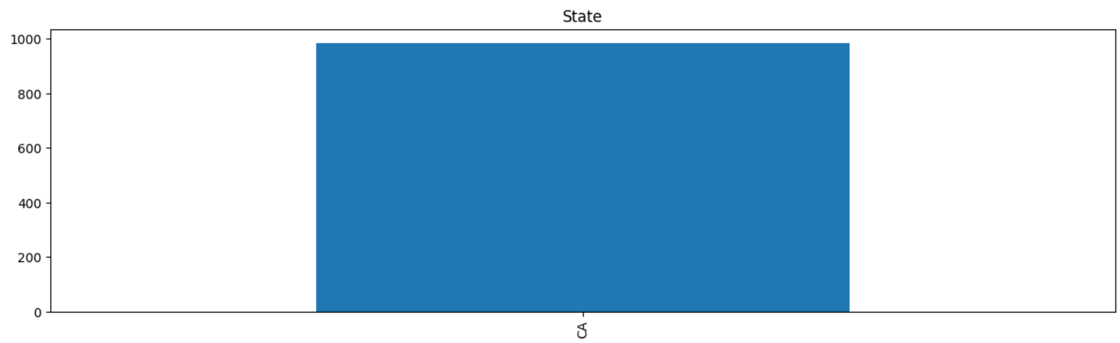
Here are histograms for the categorical columns:

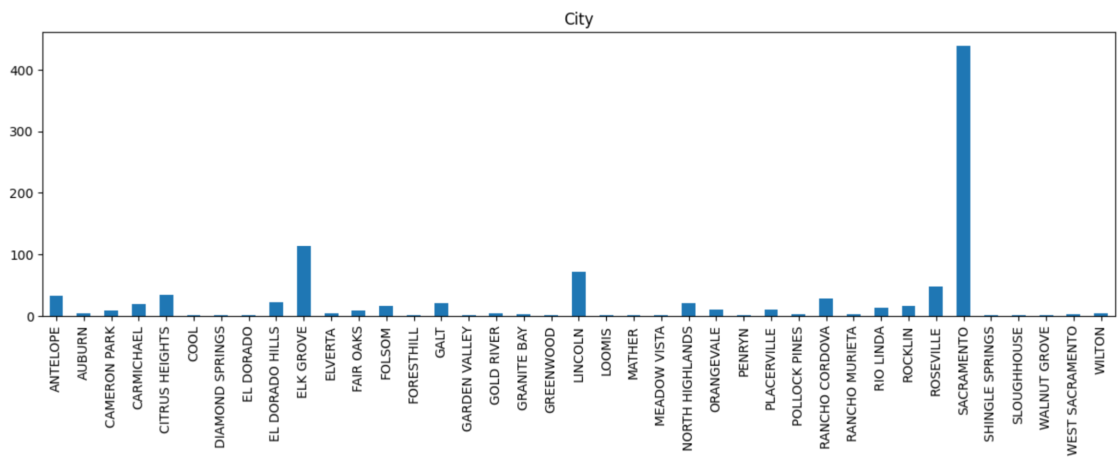
[10]:
```
cols = ['beds', 'baths', 'type', 'state', 'city', 'zip']
for col_name in cols:
    plt.figure(figsize=(15, 4))
    ax = df_rets[col_name].value_counts().sort_index().plot.bar()
    ax.set_title(col_name.capitalize())
    plt.show()
```

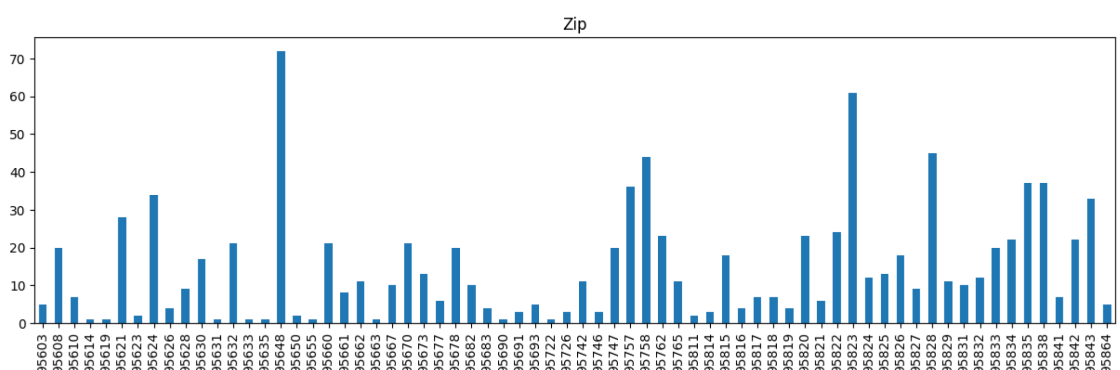

Beds
0
50
100
150
200
250
300
350
400
0
1
2
3
4
5
6
8


Baths
0
100
200
300
400
500
0
1
2
3
4
5


Type
0
200
400
600
800
Condo
Multi-Family
Residential
Unkown







There are many records with 0 beds or 0 baths. The number of both of these similar, potentially meaning that these errors stem from the same records. This could also stem from the fact that there are many empty lots in the dataset. Some of these are probably from the non-residential category, but too many 0s exist for that to be the explanation. There must be an error in the data. On an important but sensical note, the only state in the dataset is California.

1.5 5. Engineering New Variables - Part 1

This creates a boolean variable called 'empty_lot' representing just that.

[11]:
```
df_rets['empty_lot'] = df_rets['sq__ft'] == 0
df_rets.head()
```

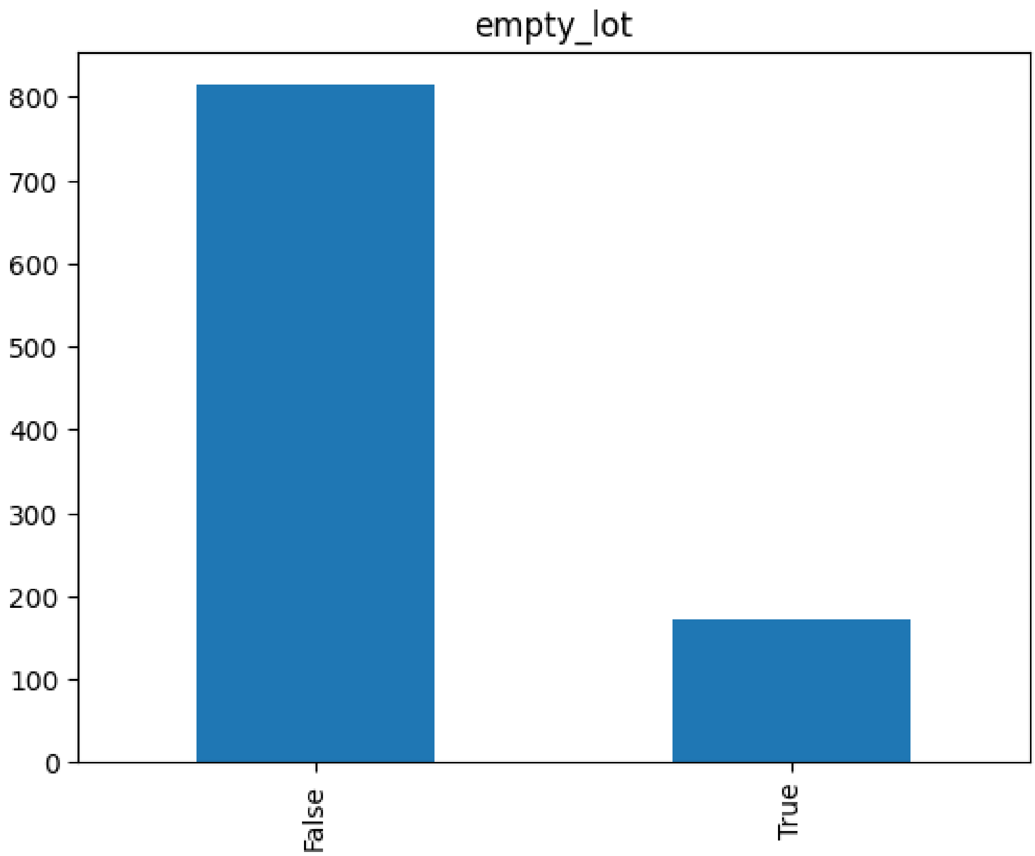
[11]:
```
              street        city    zip state  beds  baths  sq__ft         type  \
0       3526 HIGH ST  SACRAMENTO  95838    CA     2      1     836  Residential
1        51 OMAHA CT  SACRAMENTO  95823    CA     3      1    1167  Residential
2     2796 BRANCH ST  SACRAMENTO  95815    CA     2      1     796  Residential
3   2805 JANETTE WAY  SACRAMENTO  95815    CA     2      1     852  Residential
4    6001 MCMAHON DR  SACRAMENTO  95824    CA     2      1     797  Residential

                      sale_date  price   latitude   longitude  empty_lot
0  Wed May 21 00:00:00 EDT 2008  59222  38.631913 -121.434879      False
1  Wed May 21 00:00:00 EDT 2008  68212  38.478902 -121.431028      False
2  Wed May 21 00:00:00 EDT 2008  68880  38.618305 -121.443839      False
3  Wed May 21 00:00:00 EDT 2008  69307  38.616835 -121.439146      False
4  Wed May 21 00:00:00 EDT 2008  81900  38.519470 -121.435768      False
```

Here is a bar graph of the new column:

[12]:
```
ax = df_rets['empty_lot'].value_counts().plot.bar()
ax.set_title('empty_lot')
```

[12]:
```
Text(0.5, 1.0, 'empty_lot')
```



1.6 6. Engineering New Variables - Part 2

This finds the count of the unique street addresses:

[13]:
```
street_counts = makeCountDict(df_rets['street'])
print("number of unique street addresses: " + str(len(street_counts.keys())))
print("addresses with more than one occurance:\n" + str(list(filter(lambda␣
 ↪addressCount: addressCount[1] > 1, list(street_counts.items())))))
```

```
number of unique street addresses: 981
addresses with more than one occurance:
[('4734 14TH AVE', 2), ('1223 LAMBERTON CIR', 2), ('8306 CURLEW CT', 2), ('7
CRYSTALWOOD CIR', 2)]
```

Looking at the street types:

[14]:
```
df_rets['street'].head(20)
```

[14]:
```
0                      3526 HIGH ST
1                       51 OMAHA CT
2                    2796 BRANCH ST
```

```
3                     2805 JANETTE WAY
4                      6001 MCMAHON DR
5                   5828 PEPPERMILL CT
6                  6048 OGDEN NASH WAY
7                        2561 19TH AVE
8       11150 TRINITY RIVER DR Unit 114
9                         7325 10TH ST
10                   645 MORRISON AVE
11                      4085 FAWN CIR
12                     2930 LA ROSA RD
13                      2113 KIRK WAY
14                  4533 LOCH HAVEN WAY
15                     7340 HAMDEN PL
16                        6715 6TH ST
17              6236 LONGFORD DR Unit 1
18                     250 PERALTA AVE
19                     113 LEEWILL AVE
Name: street, dtype: object
```

The street types seem to be abbreviations, and in all caps, but sometimes have a Unit number as the last word, not the street type. This problem was solved eariler with isolate_street_name in part 2.

The following function isolates the street type. There were some street names in Spanish, so their street types were inferred. 'AVENIDA' was changed to 'AVE' for obvious reasons, 'VIA GRANDE" was changed to 'WAY' since its name makes it seem like it would be a larger main road. 'CONEJO' and 'VISTA DE MADERA' were defaulted to the basic 'ST', as their types couldn't really be inferred by their names. Similarly, 'BROADWAY' was also given the basic type of 'ST'. 'STATE HIGHWAY 193' could was given a fitting type of 'HWY'.

[15]:
```python
def get_street_type(address):
    street_type = ''
    if('AVENIDA' in address):
        street_type = 'AVE'
    elif('VIA GRANDE' in address):
        street_type = 'WAY'
    elif('CONEJO' in address or 'VISTA DE MADERA' in address or 'BROADWAY' in address):
        street_type = 'ST'
    elif('HIGHWAY' in address):
        street_type = 'HWY'
    else:
        str_arry = isolate_street_name(address).split()
        street_type = str_arry[-1]

    return street_type
```

Using this function, we can isolate the street and create a new categorical column in the data frame:

[16]:
```
df_rets['street_type'] = df_rets['street'].map(lambda address:␣
↪get_street_type(address)).astype('category')
print(df_rets.info())
df_rets.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   street       985 non-null    object
 1   city         985 non-null    category
 2   zip          985 non-null    category
 3   state        985 non-null    category
 4   beds         985 non-null    category
 5   baths        985 non-null    category
 6   sq__ft       985 non-null    int64
 7   type         985 non-null    category
 8   sale_date    985 non-null    object
 9   price        985 non-null    int64
 10  latitude     985 non-null    float64
 11  longitude    985 non-null    float64
 12  empty_lot    985 non-null    bool
 13  street_type  985 non-null    category
dtypes: bool(1), category(7), float64(2), int64(2), object(2)
memory usage: 59.5+ KB
None
```

[16]:
```
              street        city    zip state beds baths  sq__ft         type  \
0       3526 HIGH ST  SACRAMENTO  95838    CA    2     1     836  Residential
1        51 OMAHA CT  SACRAMENTO  95823    CA    3     1    1167  Residential
2     2796 BRANCH ST  SACRAMENTO  95815    CA    2     1     796  Residential
3   2805 JANETTE WAY  SACRAMENTO  95815    CA    2     1     852  Residential
4    6001 MCMAHON DR  SACRAMENTO  95824    CA    2     1     797  Residential

                      sale_date  price   latitude   longitude  empty_lot  \
0  Wed May 21 00:00:00 EDT 2008  59222  38.631913 -121.434879      False
1  Wed May 21 00:00:00 EDT 2008  68212  38.478902 -121.431028      False
2  Wed May 21 00:00:00 EDT 2008  68880  38.618305 -121.443839      False
3  Wed May 21 00:00:00 EDT 2008  69307  38.616835 -121.439146      False
4  Wed May 21 00:00:00 EDT 2008  81900  38.519470 -121.435768      False

  street_type
0          ST
1          CT
2          ST
3         WAY
```

```
4           DR
```

Checking the unique street types, nothing stands out after the adjustments mentioned above.

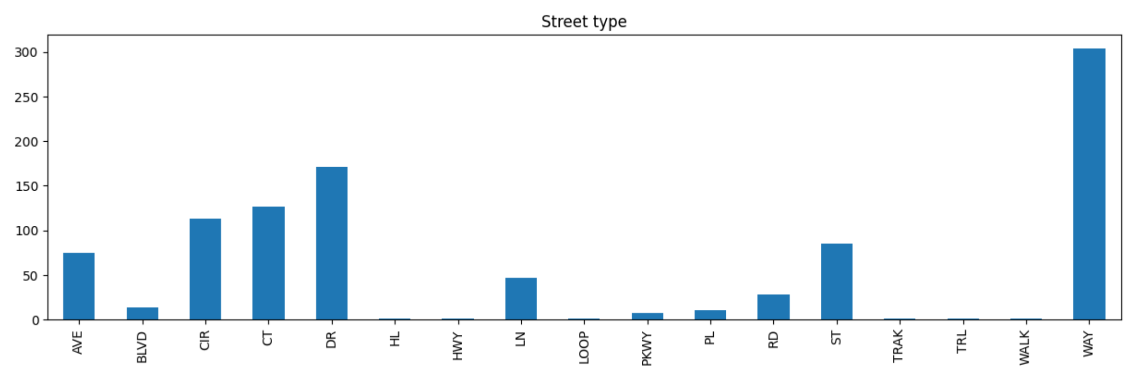
[17]:
```
makeCountDict(df_rets['street_type'])
```

[17]:
```
{'ST': 85,
 'CT': 126,
 'WAY': 304,
 'DR': 171,
 'AVE': 75,
 'CIR': 113,
 'RD': 28,
 'PL': 10,
 'LN': 47,
 'PKWY': 7,
 'BLVD': 13,
 'HWY': 1,
 'LOOP': 1,
 'WALK': 1,
 'TRAK': 1,
 'TRL': 1,
 'HL': 1}
```

Here are the street types in a bar plot:

[18]:
```
plt.figure(figsize=(15, 4))
ax = df_rets['street_type'].value_counts().sort_index().plot.bar()
ax.set_title('Street type')
```

[18]:
```
Text(0.5, 1.0, 'Street type')
```



1.7 7. Identifying Potential Dependent Variables

The variables best appropriate for regression are the (truly) numerical values that make sense to be compared to one another, like the sale date, square footage, latitude and longitute, and price. The variables suited for classification could be the number of baths, beds, zip code, city, street type, street name (not full address), property type, and whether or not the lot is empty.

A good regression output would be price. It seems like the most obvious and the most useful choice, and is almost always determined based on the other variables in the table in the real world. The number of beds could make a good dependent variable for a classification problem, as it has a small(ish) number of categories usually closely-related to the other columns.

1.8 8. Save the Cleaned Data

Here we export the cleaed data to a separate CSV file:

[19]:
```
df_rets.to_csv('clean_sacramento_real_estate.csv', index=False)
```