DSO 545 Project Main Codebook

April 4, 2021

[1]:
```
# Import relevant libraries
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
```

[2]:
```
# Import property data
data = pd.read_excel('Property Data Compiled.xlsx', index_col = 0)
```

[3]:
```
#  Check rows and column numbers in data
data.shape
```

[3]:
```
(20363, 211)
```

[4]:
```
# List of Columns
data.columns.tolist()
```

[4]:
```
['PropID',
 'Property Address',
 'Property Name',
 'Star Rating',
 'Energy Star',
 'LEED Certified',
 'Building Status',
 'Secondary Type',
 'Market Name',
 'Submarket Name',
 'City',
 'State',
 'Zip',
 'County Name',
 'For Sale Price',
 'For Sale Status',
 'Land Area (AC)',
 'Number Of Stories',
 'Style',
 'Number Of Units',
```

```
'$Price/Unit',
'Cap Rate',
'Vacancy %',
'Avg Unit SF',
'Avg Asking/Unit',
'Avg Asking/SF',
'Avg Effective/Unit',
'Avg Effective/SF',
'Avg Concessions %',
'% Studios',
'% 1-Bed',
'% 2-Bed',
'% 3-Bed',
'% 4-Bed',
'Rent Type',
'Affordable Type',
'Market Segment',
'Parking Spaces/Unit',
'Number Of Parking Spaces',
'Days On Market',
'Amenities',
'Number Of 1 Bedrooms',
'Number Of 2 Bedrooms',
'Number Of 3 Bedrooms',
'Number Of 4 Bedrooms',
'Architect Name',
'Building Class',
'Building Park',
'Closest Transit Stop',
'Closest Transit Stop Dist (mi)',
'Closest Transit Stop Walk Time (min)',
'Construction Material',
'Developer Name',
'Features',
'Four Bedroom Asking Rent/SF',
'Four Bedroom Asking Rent/Unit',
'Four Bedroom Avg SF',
'Four Bedroom Concessions %',
'Four Bedroom Effective Rent/SF',
'Four Bedroom Effective Rent/Unit',
'Four Bedroom Vacancy %',
'Four Bedroom Vacant Units',
'Number Of Studios',
'One Bedroom Asking Rent/SF',
'One Bedroom Asking Rent/Unit',
'One Bedroom Avg SF',
'One Bedroom Concessions %',
```

```
'One Bedroom Effective Rent/SF',
'One Bedroom Effective Rent/Unit',
'One Bedroom Vacancy %',
'One Bedroom Vacant Units',
'Owner Contact',
'Owner Name',
'Percent Leased',
'Property Manager Address',
'Property Manager City State Zip',
'Property Manager Contact',
'Property Manager Name',
'Property Manager Phone',
'PropertyID',
'PropertyType',
'Serial',
'Studio Asking Rent/SF',
'Studio Asking Rent/Unit',
'Studio Avg SF',
'Studio Concessions %',
'Studio Effective Rent/SF',
'Studio Effective Rent/Unit',
'Studio Vacancy %',
'Studio Vacant Units',
'Submarket Cluster',
'Three Bedroom Asking Rent/SF',
'Three Bedroom Asking Rent/Unit',
'Three Bedroom Avg SF',
'Three Bedroom Concessions %',
'Three Bedroom Effective Rent/SF',
'Three Bedroom Effective Rent/Unit',
'Three Bedroom Vacancy %',
'Three Bedroom Vacant Units',
'Total Buildings',
'Two Bedroom Asking Rent/SF',
'Two Bedroom Asking Rent/Unit',
'Two Bedroom Avg SF',
'Two Bedroom Concessions %',
'Two Bedroom Effective Rent/SF',
'Two Bedroom Effective Rent/Unit',
'Two Bedroom Vacancy %',
'Two Bedroom Vacant Units',
'Year Built',
'Year Renovated',
'Zoning',
'Owner Address',
'Owner City State Zip',
'RBA',
```

```
'Parcel Number 1(Min)',
'Parcel Number 2(Max)',
'Last Sale Date',
'Last Sale Price',
'Latitude',
'Longitude',
'Acq Notes (my data)',
'2010 Avg Age(1m)',
'2010 Med Age(1m)',
'2010 Pop Age 0-4(1m)',
'2010 Pop Age 10-14(1m)',
'2010 Pop Age 15-19(1m)',
'2010 Pop Age 20-24(1m)',
'2010 Pop Age 45-49(1m)',
'2010 Pop Age 50-54(1m)',
'2010 Pop Age 55-59(1m)',
'2010 Pop Age 5-9(1m)',
'2010 Pop Age 60-64(1m)',
'2010 Pop Age 65+(1m)',
'2010 Pop Age 85+(1m)',
'2019 Avg Age(1m)',
'2019 Avg Age&#044; Female(1m)',
'2019 Avg Age&#044; Male(1m)',
'2019 HH Age 15-24(1m)',
'2019 HH Age 25-34(1m)',
'2019 HH Age 35-44(1m)',
'2019 HH Age 45-54(1m)',
'2019 HH Age 55-64(1m)',
'2019 HH Age 65-74(1m)',
'2019 HH Age 75-84(1m)',
'2019 HH Age 85+(1m)',
'2019 Med Age(1m)',
'2019 Med Age&#044; Female(1m)',
'2019 Med Age&#044; Male(1m)',
'2019 Median HH Age(1m)',
'2019 Pop Age <19(1m)',
'2019 Pop Age 0-4(1m)',
'2019 Pop Age 10-14(1m)',
'2019 Pop Age 15-19(1m)',
'2019 Pop Age 20-24(1m)',
'2019 Pop Age 20-64(1m)',
'2019 Pop Age 25-29(1m)',
'2019 Pop Age 30-34(1m)',
'2019 Pop Age 35-39(1m)',
'2019 Pop Age 40-44(1m)',
'2019 Pop Age 45-49(1m)',
'2019 Pop Age 50-54(1m)',
```

```
'2019 Pop Age 55-59(1m)',
'2019 Pop Age 5-9(1m)',
'2019 Pop Age 60-64(1m)',
'2019 Pop Age 65+(1m)',
'2019 Pop Age 65-69(1m)',
'2019 Pop Age 70-74(1m)',
'2019 Pop Age 75-79(1m)',
'2019 Pop Age 80-84(1m)',
'2019 Pop Age 85+(1m)',
'2024 Avg Age(1m)',
'2024 Avg Female Age(1m)',
'2024 Avg Male Age(1m)',
'2024 HH Age 15-24(1m)',
'2024 HH Age 25-34(1m)',
'2024 HH Age 35-44(1m)',
'2024 HH Age 45-54(1m)',
'2024 HH Age 55-64(1m)',
'2024 HH Age 65-74(1m)',
'2024 HH Age 75-84(1m)',
'2024 HH Age 85+(1m)',
'2024 Med Age(1m)',
'2024 Median HH Age(1m)',
'2024 Pop Age <19(1m)',
'2024 Pop Age 0-4(1m)',
'2024 Pop Age 10-14(1m)',
'2024 Pop Age 15-19(1m)',
'2024 Pop Age 20-24(1m)',
'2024 Pop Age 20-64(1m)',
'2024 Pop Age 25-29(1m)',
'2024 Pop Age 30-34(1m)',
'2024 Pop Age 35-39(1m)',
'2024 Pop Age 40-44(1m)',
'2024 Pop Age 45-49(1m)',
'2024 Pop Age 50-54(1m)',
'2024 Pop Age 55-59(1m)',
'2024 Pop Age 5-9(1m)',
'2024 Pop Age 60-64(1m)',
'2024 Pop Age 65+(1m)',
'2024 Pop Age 65-69(1m)',
'2024 Pop Age 70-74(1m)',
'2024 Pop Age 75-79(1m)',
'2024 Pop Age 80-84(1m)',
'2024 Pop Age 85+(1m)',
'Situs_Num',
'Situs_Num_Remainder',
'SITUS_DIR',
'SITUS_NAM',
```

```
'SCP',
'SCSitus_NumNam',
'SCAPN']
```

0.1 1. Data Cleaning

Create a subset of original data based on EDA to reduce calculation load. Drop duplicates. Get 5 digit zip & Fix State typo.

[5]:
```
cols = ['PropertyID',
        # rent fields
        'Avg Effective/SF', 'Avg Concessions %',
        'Studio Effective Rent/SF', 'One Bedroom Effective Rent/SF', 'Two␣
↪Bedroom Effective Rent/SF',
        'Three Bedroom Effective Rent/SF', 'Four Bedroom Effective Rent/SF',
        # unit fields
        'Studio Avg SF', 'Number Of Studios', 'Studio Vacant Units', 'Studio␣
↪Vacancy %',
        'One Bedroom Avg SF','Number Of 1 Bedrooms', 'One Bedroom Vacant␣
↪Units', 'One Bedroom Vacancy %',
        'Two Bedroom Avg SF', 'Number Of 2 Bedrooms', 'Two Bedroom Vacant␣
↪Units', 'Two Bedroom Vacancy %',
        'Three Bedroom Avg SF', 'Number Of 3 Bedrooms', 'Three Bedroom Vacant␣
↪Units', 'Three Bedroom Vacancy %',
        'Four Bedroom Avg SF', 'Number Of 4 Bedrooms', 'Four Bedroom Vacant␣
↪Units', 'Four Bedroom Vacancy %',
        # location fields
        'State', 'Market Name', 'City', 'Zip', 'County Name',
        'Closest Transit Stop Dist (mi)', 'Latitude', 'Longitude',
        # property fields
        'Star Rating', 'Building Status', 'Land Area (AC)', 'Number Of Stories',
        'Style', 'Number Of Units', 'Vacancy %', 'Avg Unit SF', 'RBA',
        '% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
        'Rent Type', 'Affordable Type', 'Construction Material',
        'Amenities', 'Owner Name', 'Year Built', 'Year Renovated',
        # demographic fields
        '2019 Avg Age(1m)', '2019 Pop Age <19(1m)', '2019 Pop Age␣
↪20-64(1m)','2019 Pop Age 65+(1m)']
sub = data.copy()[cols]
sub.drop_duplicates(subset='PropertyID', inplace = True)
sub['Zip5'] = sub['Zip'].str[:5]
sub['State'] = sub['State'].map({'TX':'TX',
                                 'FL':'FL',
                                 'GA':'GA',
                                 'NC':'NC',
                                 'Fl':'FL',
                                 'NC ':'NC'})
```

```python
sub.shape
```

[5]: (20300, 62)

[6]:
```python
# Export unique zip codes
pd.DataFrame(sub['Zip5'].unique()).to_csv('Zip5.csv',index=False)
```

[7]:
```python
# Check data type and missing values for each of the variables
sub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20300 entries, 1 to 20363
Data columns (total 62 columns):
PropertyID                         20300 non-null int64
Avg Effective/SF                   16751 non-null float64
Avg Concessions %                  20300 non-null float64
Studio Effective Rent/SF           2475 non-null float64
One Bedroom Effective Rent/SF      15360 non-null float64
Two Bedroom Effective Rent/SF      16207 non-null float64
Three Bedroom Effective Rent/SF    10031 non-null float64
Four Bedroom Effective Rent/SF     1401 non-null float64
Studio Avg SF                      2934 non-null float64
Number Of Studios                  3041 non-null float64
Studio Vacant Units                2588 non-null float64
Studio Vacancy %                   2588 non-null float64
One Bedroom Avg SF                 16910 non-null float64
Number Of 1 Bedrooms               17326 non-null float64
One Bedroom Vacant Units           15815 non-null float64
One Bedroom Vacancy %              15815 non-null float64
Two Bedroom Avg SF                 17451 non-null float64
Number Of 2 Bedrooms               17734 non-null float64
Two Bedroom Vacant Units           16526 non-null float64
Two Bedroom Vacancy %              16526 non-null float64
Three Bedroom Avg SF               10767 non-null float64
Number Of 3 Bedrooms               10964 non-null float64
Three Bedroom Vacant Units         10222 non-null float64
Three Bedroom Vacancy %            10222 non-null float64
Four Bedroom Avg SF                1569 non-null float64
Number Of 4 Bedrooms               1604 non-null float64
Four Bedroom Vacant Units          1510 non-null float64
Four Bedroom Vacancy %             1510 non-null float64
State                              20300 non-null object
Market Name                        20300 non-null object
City                               20300 non-null object
Zip                                20300 non-null object
County Name                        20300 non-null object
Closest Transit Stop Dist (mi)     18448 non-null float64
```

```
Latitude                          20300 non-null float64
Longitude                         20300 non-null float64
Star Rating                       20300 non-null int64
Building Status                   20300 non-null object
Land Area (AC)                    18108 non-null float64
Number Of Stories                 19291 non-null float64
Style                             19462 non-null object
Number Of Units                   20300 non-null int64
Vacancy %                         17021 non-null float64
Avg Unit SF                       18705 non-null float64
RBA                               20195 non-null float64
% Studios                         3187 non-null float64
% 1-Bed                           17383 non-null float64
% 2-Bed                           17782 non-null float64
% 3-Bed                           10997 non-null float64
% 4-Bed                           1546 non-null float64
Rent Type                         19796 non-null object
Affordable Type                   3735 non-null object
Construction Material             14185 non-null object
Amenities                         16797 non-null object
Owner Name                        19612 non-null object
Year Built                        19745 non-null float64
Year Renovated                    2217 non-null float64
2019 Avg Age(1m)                  20271 non-null float64
2019 Pop Age <19(1m)              20271 non-null float64
2019 Pop Age 20-64(1m)            20271 non-null float64
2019 Pop Age 65+(1m)              20271 non-null float64
Zip5                              20300 non-null object
dtypes: float64(46), int64(3), object(13)
memory usage: 9.8+ MB
```

0.1.1 1.1 Drop properties that are proposed, under construction or demolished

[8]:
```
# Number of rows for each building status
sub['Building Status'].value_counts()
```

[8]:
```
Existing              17934
Proposed                964
Under Construction      811
Demolished              365
Under Renovation        226
Name: Building Status, dtype: int64
```

[9]:
```
# Remove rows belonging to proposed, under construction and demolished␣
↪categories
row_initial = sub.shape[0]
sub = sub[sub['Building Status'].isin(['Existing','Under Renovation'])].copy()
```

```
print('Records dropped:', row_initial - sub.shape[0])
print('Current size:', sub.shape)
```

```
Records dropped: 2140
Current size: (18160, 62)
```

0.1.2 1.2 Drop properties that are missing Avg Effective/SF (which is our output variable i.e. final rent price/sqft)

Almost all records missing Avg Effective/SF are missing Effective Rent/SF for specifi unit types too, so cannot be calculated and filled and have to be dropped.

[10]:
```
# Calculate how much data is missing for avg effective/sf
print('Percentage of data missing Avg Effective/SF:',1-sub['Avg Effective/SF'].
↪count()/sub.shape[0])
```

```
Percentage of data missing Avg Effective/SF: 0.08838105726872247
```

[11]:
```
# When overal rent/SF is missing, detailed rents by unit types are usually␣
↪missing too
# The first can't be calculated and filled from the latter
sub[sub['Avg Effective/SF'].isnull()][['Studio Effective Rent/SF', 'One Bedroom␣
↪Effective Rent/SF',
                                       'Two Bedroom Effective Rent/SF','Three␣
↪Bedroom Effective Rent/SF',
                                       'Four Bedroom Effective Rent/SF']].
↪count()
```

[11]:
```
Studio Effective Rent/SF         0
One Bedroom Effective Rent/SF    1
Two Bedroom Effective Rent/SF    2
Three Bedroom Effective Rent/SF  1
Four Bedroom Effective Rent/SF   0
dtype: int64
```

[12]:
```
# Drop the rows where avg effective/SF is missing
row_initial = sub.shape[0]
sub = sub[sub['Avg Effective/SF'].notnull()].copy()
print('Records dropped:', row_initial - sub.shape[0])
print('Current size:', sub.shape)
```

```
Records dropped: 1605
Current size: (16555, 62)
```

0.1.3 1.3 Validate property unit mix

0.1.4 1.3.1 Fix wrong data in % 4-Bed

Most records have % room types that don't add up to 100. Reason is that Column '% 4-Bed' is not consistent with other % room type columns. Multiply all '% 4-Bed' by 100 to make it consistent.

[13]:
```
# Multiply % 4-Bed by 100 to make data consistent
# Create a % total column to see if all % columns add up to 100
sub['% 4-Bed'] = sub['% 4-Bed']*100
sub['% tot'] = sub[['% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed']].
↪sum(axis=1)
sub['% tot'] = sub['% tot'].apply(round)
sub['% tot'].value_counts()
```

[13]:
```
100    16518
0         37
Name: % tot, dtype: int64
```

Some % tot still add up to suggesting missing data. Let's dig deeper into different units and understand this.

0.1.5 1.3.2 Fill missing value in % Studios/1-Bed/2-Bed/3-Bed/4-Bed

Some records have Number Of Studios value but no % Studios value. Fill missing value with Number Of Studios divided by Number Of Units. Do the same for 1/2/3/4 bedrooms.

[14]:
```
# Fill missing values in % of unit types with values calculated from other␣
↪fields
unit_type = {'Studios':'Studios', '1 Bedrooms':'1-Bed',
             '2 Bedrooms':'2-Bed', '3 Bedrooms':'3-Bed', '4 Bedrooms':'4-Bed'}

for utype in unit_type.keys():
    sub['% {}'.format(unit_type[utype])] =\
    sub[['% {}'.format(unit_type[utype]), 'Number Of {}'.format(utype), 'Number␣
↪Of Units']].\
    apply(lambda x: x[0] if np.isnan(x[0])==False else(\
    x[0] if np.isnan(x[1]) else x[1]/x[2]*100), axis=1)
```

0.1.6 1.3.3 Fill missing value in Number Of Studios/1-Bed/2-Bed/3-Bed/4-Bed

Some records have % Studios values but no Number Of Studios value. Fill missing value with number of total unit X % Studios. Do the same for 1/2/3/4 bedrooms.

[15]:
```
# Fill missing values in numbers of unit types with values calculated from␣
↪other fields
for utype in unit_type.keys():
    sub['Number Of {}'.format(utype)] = sub[['Number Of {}'.format(utype), '%␣
↪{}'.format(unit_type[utype]),
```

```
                                                    'Number Of Units']].\
                                          apply(lambda x: x[1] * x[2]/100 if np.
↪isnan(x[0]) else x[0], axis=1)
```

0.1.7 1.3.4 Check if % unit types add up to 100 , fix wrong numbers

Let's come back to % tot to validate number of different units add up.

[16]:
```
# Recheck % total to see if there are still non-100s
sub['% tot'] = sub[['% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed']].
↪sum(axis=1)
sub['% tot'] = sub['% tot'].apply(round)
sub['% tot'].value_counts()
```

[16]:
```
100    16544
101        3
145        2
200        1
71         1
166        1
86         1
164        1
112        1
Name: % tot, dtype: int64
```

[17]:
```
# Find rows that don't add up to 100
wrong = sub['% tot'].value_counts().index.tolist()
wrong.remove(100)
sub[sub['% tot'].isin(wrong)][['Studio Effective Rent/SF', 'One Bedroom␣
↪Effective Rent/SF',
                               'Two Bedroom Effective Rent/SF','Three Bedroom␣
↪Effective Rent/SF',
                               'Four Bedroom Effective Rent/SF',
                               '% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '%␣
↪4-Bed',
                               'Number Of Studios', 'Number Of 1 Bedrooms',␣
↪'Number Of 2 Bedrooms',
                               'Number Of 3 Bedrooms', 'Number Of 4 Bedrooms',␣
↪'Number Of Units']].head(11)
```

[17]:
```
       Studio Effective Rent/SF  One Bedroom Effective Rent/SF  \
Row
835                         NaN                            NaN
924                        0.51                           1.02
941                         NaN                            NaN
1677                        NaN                           0.73
4064                       1.71                           1.99
```

5497	NaN	1.62
6137	1.34	NaN
8089	NaN	NaN
11073	NaN	0.95
11401	NaN	1.11
12945	1.59	1.39

Row	Two Bedroom Effective Rent/SF	Three Bedroom Effective Rent/SF \
835	NaN	NaN
924	0.78	0.82
941	NaN	NaN
1677	0.51	0.49
4064	1.96	NaN
5497	1.37	1.24
6137	0.65	0.44
8089	1.43	1.59
11073	0.92	0.80
11401	0.88	1.11
12945	1.15	1.20

Row	Four Bedroom Effective Rent/SF	% Studios	% 1-Bed	% 2-Bed \
835	0.64	NaN	NaN	NaN
924	NaN	1.123596	44.380000	46.630000
941	1.04	NaN	NaN	NaN
1677	NaN	NaN	11.594203	59.420000
4064	NaN	22.540000	45.081967	54.920000
5497	NaN	100.000000	28.571429	57.142857
6137	NaN	45.000000	45.000000	45.000000
8089	1.65	NaN	86.710000	10.490000
11073	NaN	NaN	63.960000	36.040000
11401	NaN	NaN	43.480000	55.650000
12945	NaN	63.398693	89.740000	10.260000

Row	% 3-Bed	% 4-Bed	Number Of Studios	Number Of 1 Bedrooms \
835	NaN	71.282051	NaN	NaN
924	8.990000	NaN	4.0	158.0
941	NaN	85.789474	NaN	NaN
1677	40.580000	NaN	NaN	16.0
4064	22.540000	NaN	55.0	110.0
5497	14.285714	NaN	336.0	96.0
6137	10.000000	NaN	45.0	45.0
8089	2.800000	65.734266	NaN	30.0
11073	0.900901	NaN	NaN	71.0
11401	0.869565	0.870000	NaN	50.0

```
12945    0.653595        NaN               97.0                  51.0

       Number Of 2 Bedrooms  Number Of 3 Bedrooms  Number Of 4 Bedrooms  \
Row
835                     NaN                   NaN              139.0000
924                   166.0               28.0000                   NaN
941                     NaN                   NaN              163.0000
1677                   82.0               40.0000                   NaN
4064                   79.0               54.9976                   NaN
5497                  192.0               48.0000                   NaN
6137                   45.0               10.0000                   NaN
8089                   15.0                4.0000               94.0000
11073                  39.0                1.0000                   NaN
11401                  64.0                1.0000                1.0005
12945                   4.0                1.0000                   NaN

       Number Of Units
Row
835                195
924                356
941                190
1677               138
4064               244
5497               336
6137               100
8089               143
11073              111
11401              115
12945              153
```

Row 11401 can be easily fixed by removing number of 4 bedrooms and % 4-Bed.

[18]:
```
# Fix row 11401
sub.loc[11401, 'Number Of 4 Bedrooms']=np.nan
sub.loc[11401, '% 4-Bed']=np.nan
```

Find the rows where number of units add up and fix their % accordingly. Drop the rows where number of units don't add up.

[19]:
```
# Find rows where absolute numbers do and do not add up
test = sub[sub['% tot'].isin(wrong)]
ind = pd.DataFrame(test[['Number Of Studios', 'Number Of 1 Bedrooms', 'Number␣
 ↪Of 2 Bedrooms',
                         'Number Of 3 Bedrooms', 'Number Of 4 Bedrooms']].\
                   sum(axis=1)==test['Number Of Units']).reset_index()
ind.columns=['Row','match']
match = ind[ind['match']==True]['Row'].tolist()
unmatch = ind[ind['match']==False]['Row'].tolist()
```

```
print('Matched:', match)
print('Unmatched:', unmatch)
```

```
Matched: [924, 1677, 8089, 11073, 11401, 12945]
Unmatched: [835, 941, 4064, 5497, 6137]
```

[20]:
```
# Remove unmatched rows
# For matched ones, recalculate the % based on absolute number of units
unit_type = {'Studios':'Studios', '1 Bedrooms':'1-Bed',
             '2 Bedrooms':'2-Bed', '3 Bedrooms':'3-Bed', '4 Bedrooms':'4-Bed'}
for utype in unit_type.keys():
    sub.loc[match,'% {}'.format(unit_type[utype])] =\
    sub.loc[match, ['% {}'.format(unit_type[utype]), 'Number Of {}'.
 ↪format(utype),
                         'Number Of Units']].apply(lambda x: x[1]/x[2]*100, axis=1)
row_initial = sub.shape[0]
sub = sub.drop(unmatch)
print('Records dropped:', row_initial - sub.shape[0])
print('Current size:', sub.shape)
```

```
Records dropped: 5
Current size: (16550, 63)
```

[21]:
```
# Confirm % total all equal to 100 now
sub['% tot'] = sub[['% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed']].
 ↪sum(axis=1)
sub['% tot'] = sub['% tot'].apply(round)
sub['% tot'].value_counts()
```

[21]:
```
100    16550
Name: % tot, dtype: int64
```

0.1.8 1.3.5 Check if % and number of units match

Manually calculate number of different types of units (i.e. # Studios,1/2/3/4 bedrooms) and see if they match with the original data

[22]:
```
# Manually calculate number of unit by types from %
test = sub[['% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
                         'Number Of Studios', 'Number Of 1 Bedrooms', 'Number Of 2␣
 ↪Bedrooms',
                         'Number Of 3 Bedrooms', 'Number Of 4 Bedrooms', 'Number␣
 ↪Of Units']].copy()
test = test.fillna(0)
unit_type = {'Studios':'Studios', '1 Bedrooms':'1-Bed',
             '2 Bedrooms':'2-Bed', '3 Bedrooms':'3-Bed', '4 Bedrooms':'4-Bed'}
for utype in unit_type.keys():
```

```python
    test['# {}'.format(utype)] = test[['% {}'.format(unit_type[utype]),
                                        'Number Of Units']].apply(lambda x:
                                  round(x[0]*x[1]/100), axis=1)
test.head()
```

[22]:

```
     % Studios  % 1-Bed  % 2-Bed  % 3-Bed  % 4-Bed  Number Of Studios  \
Row
1         0.00    35.71    46.67    17.62     0.00                0.0
4        32.45    45.21    22.34     0.00     0.00               61.0
5         0.00     0.00    61.02     0.00    38.98                0.0
6        58.39    41.61     0.00     0.00     0.00               87.0
7        61.24    38.76     0.00     0.00     0.00               79.0

     Number Of 1 Bedrooms  Number Of 2 Bedrooms  Number Of 3 Bedrooms  \
Row
1                    75.0                  98.0                  37.0
4                    85.0                  42.0                   0.0
5                     0.0                  72.0                   0.0
6                    62.0                   0.0                   0.0
7                    50.0                   0.0                   0.0

     Number Of 4 Bedrooms  Number Of Units  # Studios  # 1 Bedrooms  \
Row
1                     0.0              210        0.0          75.0
4                     0.0              188       61.0          85.0
5                    46.0              118        0.0           0.0
6                     0.0              149       87.0          62.0
7                     0.0              129       79.0          50.0

     # 2 Bedrooms  # 3 Bedrooms  # 4 Bedrooms
Row
1            98.0          37.0           0.0
4            42.0           0.0           0.0
5            72.0           0.0          46.0
6             0.0           0.0           0.0
7             0.0           0.0           0.0
```

[23]:

```python
# Find cases where manually calculated numbers don't match original input
unmatch = \
test[(test['Number Of Studios']!=test['# Studios'])|\
     (test['Number Of 1 Bedrooms']!=test['# 1 Bedrooms'])|\
     (test['Number Of 2 Bedrooms']!=test['# 2 Bedrooms'])|\
     (test['Number Of 3 Bedrooms']!=test['# 3 Bedrooms'])|\
     (test['Number Of 4 Bedrooms']!=test['# 4 Bedrooms'])]
unmatch.head()
```

[23]:

```
     % Studios  % 1-Bed  % 2-Bed  % 3-Bed  % 4-Bed  Number Of Studios  \
Row
64        0.00    43.70    26.05    21.85     8.40                0.0
120       2.11    38.95    47.37    11.58     0.00                4.0
334       0.00    86.42    13.58     0.00     0.00                0.0
370       0.00    61.67     5.56    20.00    12.78                0.0
408       0.00    31.13    22.18     7.78    38.91                0.0

     Number Of 1 Bedrooms  Number Of 2 Bedrooms  Number Of 3 Bedrooms  \
Row
64                  104.0                  62.0                  52.0
120                  78.0                  89.0                  17.0
334                 210.0                  32.0                   0.0
370                 111.0                  10.0                  36.0
408                  80.0                  57.0                  20.0

     Number Of 4 Bedrooms  Number Of Units  # Studios  # 1 Bedrooms  \
Row
64                   12.0              238        0.0         104.0
120                   0.0              188        4.0          73.0
334                   0.0              242        0.0         209.0
370                  19.0              180        0.0         111.0
408                  60.0              257        0.0          80.0

     # 2 Bedrooms  # 3 Bedrooms  # 4 Bedrooms
Row
64           62.0          52.0          20.0
120          89.0          22.0           0.0
334          33.0           0.0           0.0
370          10.0          36.0          23.0
408          57.0          20.0         100.0
```

[24]:

```python
# Percentage mismatch
print("Number of rows where % and number of units don't match:", unmatch.
 ↪shape[0])
print('Percentage of unmatch:', unmatch.shape[0]/sub.shape[0])
```

```
Number of rows where % and number of units don't match: 204
Percentage of unmatch: 0.012326283987915408
```

1.2% rows unmatched are acceptable. Use percentage of units for regression model.

0.1.9 1.4 Remove rows that have missing values in Amenities & Construction Material

Amenities and construction material variables are critical and filling missing values is difficult for these categories

[25]:
```
# Check number of rows with Amenities and Construction Material information
sub[['Amenities','Construction Material']].count()
```

[25]:
```
Amenities                15759
Construction Material    13337
dtype: int64
```

[26]:
```
# Delete rows that are missing Amenities and Construction Material information
row_initial = sub.shape[0]
sub = sub[(sub['Amenities'].notnull()) &\
          (sub['Construction Material'].notnull())].copy()
print('Records dropped:', row_initial - sub.shape[0])
print('Current size:', sub.shape)
```

```
Records dropped: 3642
Current size: (12908, 63)
```

0.1.10 1.5 Check if variables related to each unit type add up

- Fill Avg SF, Vacant Unites & Vacancy % missing values
- Check if # of vacant unit matches vacancy %
- Check if there are outliers/wrong entries

0.1.11 1.5.1 Define functions to fill Vacancy% and Avg SF, and calculate Vacant Units

[27]:
```
levels = ['Zip5', 'City', 'County Name', 'Market Name']

# Function to fill vacancy % by unit type missing values with regional mean
def fill_vacancy_mean(uname1, uname2, level):
    sub.loc[sub['% {}'.format(uname1)].notnull(), '{} Vacancy %'.
 ↪format(uname2)] = \
    sub[sub['% {}'.format(uname1)].notnull()][['{} Vacancy %'.format(uname2),␣
 ↪level]].groupby(level).\
    transform(lambda x: x.fillna(x.mean()))

# Function to calculate vacant unit by type from vacancy % to fill missing␣
 ↪values
def calculate_vacant_unit(uname1, uname2, uname3):
    sub.loc[sub['% {}'.format(uname1)].notnull(),'{} Vacant Units'.
 ↪format(uname2)] =\
    sub.loc[sub['% {}'.format(uname1)].notnull(),'{} Vacant Units'.
 ↪format(uname2)].\
    fillna(sub[sub['% {}'.format(uname1)].notnull()]\
           [['Number Of {}'.format(uname3),'{} Vacancy %'.format(uname2)]].\
           apply(lambda x: round(x[0]*x[1]/100), axis=1))

# Function to fill unit size by unit type missing values with regional mean
```

```
def fill_sf_mean(uname1, uname2, level):
    sub.loc[sub['% {}'.format(uname1)].notnull(), '{} Avg SF'.format(uname2)] =␣
 ↪\
    sub[sub['% {}'.format(uname1)].notnull()][['{} Avg SF'.format(uname2),␣
 ↪level]].groupby(level).\
    transform(lambda x: x.fillna(x.mean()))
```

0.1.12 1.5.2 Studio related variables:

[28]:
```
# Check descriptive statistics of Studio related variables
sub[sub['% Studios'].notnull()][['Studio Avg SF', 'Number Of Studios', 'Studio␣
 ↪Vacant Units',
                                 'Studio Vacancy %', 'Studio Effective Rent/
 ↪SF', '% Studios']].describe()
```

[28]:

	Studio Avg SF	Number Of Studios	Studio Vacant Units
count	1842.000000	1863.000000	1856.000000
mean	512.115092	37.958137	2.990841
std	125.744971	40.772848	5.388560
min	6.000000	1.000000	0.000000
25%	436.250000	12.000000	1.000000
50%	503.000000	27.000000	2.000000
75%	582.000000	50.000000	3.000000
max	1502.000000	518.000000	67.000000

	Studio Vacancy %	Studio Effective Rent/SF	% Studios
count	1856.000000	1842.000000	1863.000000
mean	9.860991	1.964061	15.451999
std	14.523106	2.621371	15.639930
min	0.000000	0.500000	0.140000
25%	2.600000	1.410000	5.140000
50%	6.300000	1.760000	11.330000
75%	10.925000	2.197500	20.375000
max	100.000000	108.820000	100.000000

Missing values for Avg SF, Vacant Units, Vacancy % and Effective Rent/SF. Unusual small Studio Avg SF - 6. Outliers in Studio Avg SF & Studio Effective Rent/SF.

[29]:
```
# Check for outlier cases where studio avg SF is less than 200
sub[(sub['% Studios'].notnull()) & (sub['Studio Avg SF']<200)]\
[['PropertyID','Studio Effective Rent/SF', 'One Bedroom Effective Rent/SF',␣
 ↪'Two Bedroom Effective Rent/SF',
  'Three Bedroom Effective Rent/SF', 'Four Bedroom Effective Rent/SF', 'Avg␣
 ↪Effective/SF', 'Avg Unit SF',
  '% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
  'Studio Avg SF', 'Number Of Studios', 'Studio Vacant Units', 'Studio Vacancy␣
 ↪%']]
```

[29]:
```
       PropertyID  Studio Effective Rent/SF  One Bedroom Effective Rent/SF  \
Row
11548     8938828                      3.61                           1.52
11724     8943094                    108.82                           1.22

       Two Bedroom Effective Rent/SF  Three Bedroom Effective Rent/SF  \
Row
11548                           1.29                              NaN
11724                            NaN                             1.04

       Four Bedroom Effective Rent/SF  Avg Effective/SF  Avg Unit SF  \
Row
11548                             NaN              1.47        550.0
11724                             NaN              1.42        630.0

       % Studios  % 1-Bed  % 2-Bed  % 3-Bed  % 4-Bed  Studio Avg SF  \
Row
11548      20.63    22.22    57.14      NaN      NaN          154.0
11724      31.67    28.33      NaN     40.0      NaN            6.0

       Number Of Studios  Studio Vacant Units  Studio Vacancy %
Row
11548               26.0                  2.0               7.7
11724               38.0                  2.0               5.3
```

PropertyID 8943094 has wrong Studio Avg SF (6 sf is two small for a studio). Based on other columns, a reasonable fix would be 600.

[30]:
```
# Update Studio Avg SF and Studio effective rent/SF with new values for␣
↪PropertyID 8943094
sub.loc[sub.PropertyID==8943094, 'Studio Avg SF'] = sub.loc[sub.
↪PropertyID==8943094, 'Studio Avg SF'] * 100
sub.loc[sub.PropertyID==8943094, 'Studio Effective Rent/SF'] = \
sub.loc[sub.PropertyID==8943094, 'Studio Effective Rent/SF'] / 100
```

[31]:
```
# Fill missing values with predefined functions
for level in levels:
    fill_vacancy_mean('Studios', 'Studio', level)
    fill_sf_mean('Studios', 'Studio', level)
calculate_vacant_unit('Studios', 'Studio', 'Studios')

sub[sub['% Studios'].notnull()][['Studio Avg SF', 'Number Of Studios', 'Studio␣
↪Vacant Units',
                                  'Studio Vacancy %', 'Studio Effective Rent/
↪SF', '% Studios']].describe()
```

[31]:
```
       Studio Avg SF  Number Of Studios  Studio Vacant Units  \
count    1863.000000        1863.000000          1863.000000
mean      512.168385          37.958137             3.004831
std       125.346267          40.772848             5.431981
min       154.000000           1.000000             0.000000
25%       436.000000          12.000000             1.000000
50%       502.000000          27.000000             2.000000
75%       581.500000          50.000000             3.000000
max      1502.000000         518.000000            67.000000

       Studio Vacancy %  Studio Effective Rent/SF    % Studios
count       1863.000000               1842.000000  1863.000000
mean           9.857495                  1.905574    15.451999
std           14.500777                  0.816340    15.639930
min            0.000000                  0.500000     0.140000
25%            2.600000                  1.410000     5.140000
50%            6.300000                  1.760000    11.330000
75%           10.950000                  2.190000    20.375000
max          100.000000                 10.240000   100.000000
```

[32]:
```
# Check if Studio Vacant units match with Studio Vacancy %
test = sub[sub['% Studios'].notnull()].copy()
test['# Vacant Studio'] = test[['Number Of Studios', 'Studio Vacancy %']].
 ↪apply(lambda x:
                          round(x[0]*x[1]/100), axis=1)
print(test[test['Studio Vacant Units']!=test['# Vacant Studio']].shape)
#test[test['Studio Vacant Units']!=test['# Vacant Studio']][['Studio Vacant␣
 ↪Units', '# Vacant Studio']]
```

```
(0, 64)
```

Studio Vacant Units and Studio Vacancy % matches.

0.1.13 1.5.2 One Bedroom related variables:

[33]:
```
# Check descriptive statistics of 1-Bedroom related variables
sub[sub['% 1-Bed'].notnull()][['One Bedroom Avg SF','Number Of 1 Bedrooms',␣
 ↪'One Bedroom Vacant Units',
        'One Bedroom Vacancy %', 'One Bedroom Effective Rent/SF', '% 1-Bed']].
 ↪describe()
```

[33]:
```
       One Bedroom Avg SF  Number Of 1 Bedrooms  One Bedroom Vacant Units  \
count        12023.000000          12027.000000              12026.000000
mean           731.794644            119.524737                  8.285881
std            106.559897             84.496398                 13.700173
min            271.000000              1.000000                  0.000000
25%            660.000000             60.000000                  2.000000
50%            730.000000            104.000000                  5.000000
```

75%	796.000000	159.000000	10.000000
max	1502.000000	1431.000000	516.000000

	One Bedroom Vacancy %	One Bedroom Effective Rent/SF	% 1-Bed
count	12026.000000	12023.000000	12027.000000
mean	6.655987	1.361945	44.762135
std	7.873076	0.441826	19.697155
min	0.000000	0.350000	0.240000
25%	3.100000	1.100000	30.210000
50%	5.100000	1.290000	44.670000
75%	7.700000	1.520000	58.330000
max	100.000000	8.070000	100.000000

Missing values for Avg SF, Vacant Units, Vacancy % and Effective Rent/SF. Outliers in Avg SF and Effective Rent/SF.

[34]:
```
# Fill missing values with predefined functions
for level in levels:
    fill_vacancy_mean('1-Bed', 'One Bedroom', level)
    fill_sf_mean('1-Bed', 'One Bedroom', level)
calculate_vacant_unit('1-Bed', 'One Bedroom', '1 Bedrooms')

sub[sub['% 1-Bed'].notnull()][['One Bedroom Avg SF','Number Of 1 Bedrooms',␣
 ↪'One Bedroom Vacant Units',
       'One Bedroom Vacancy %', 'One Bedroom Effective Rent/SF', '% 1-Bed']].
 ↪describe()
```

[34]:

	One Bedroom Avg SF	Number Of 1 Bedrooms	One Bedroom Vacant Units \
count	12027.000000	12027.000000	12027.000000
mean	731.784468	119.524737	8.285275
std	106.545452	84.496398	13.699764
min	271.000000	1.000000	0.000000
25%	660.000000	60.000000	2.000000
50%	730.000000	104.000000	5.000000
75%	796.000000	159.000000	10.000000
max	1502.000000	1431.000000	516.000000

	One Bedroom Vacancy %	One Bedroom Effective Rent/SF	% 1-Bed
count	12027.00000	12023.000000	12027.000000
mean	6.65564	1.361945	44.762135
std	7.87284	0.441826	19.697155
min	0.00000	0.350000	0.240000
25%	3.10000	1.100000	30.210000
50%	5.10000	1.290000	44.670000
75%	7.70000	1.520000	58.330000
max	100.00000	8.070000	100.000000

[35]:
```
# Check if 1-Bedroom Vacant units match with 1-Bedroom Vacancy %
test = sub[sub['% 1-Bed'].notnull()].copy()
test['# Vacant 1-Bed'] = test[['Number Of 1 Bedrooms', 'One Bedroom Vacancy␣
↪%']].apply(lambda x:
                              round(x[0]*x[1]/100), axis=1)
#print(test[test['One Bedroom Vacant Units']!=test['# Vacant 1-Bed']].shape)
test[test['One Bedroom Vacant Units']!=test['# Vacant 1-Bed']]\
[['One Bedroom Vacant Units', '# Vacant 1-Bed']]
```

[35]:

Row	One Bedroom Vacant Units	# Vacant 1-Bed
20322	516.0	517.0

One Bedroom Vacant Units and One Bedroom Vacancy % have only one minor mismatch due to rounding and can be ignored

0.1.14 1.5.3 Two Bedroom related variables:

[36]:
```
# Check descriptive statistics of 2-Bedroom related variables
sub[sub['% 2-Bed'].notnull()][['Two Bedroom Avg SF','Number Of 2 Bedrooms',␣
↪'Two Bedroom Vacant Units',
       'Two Bedroom Vacancy %', 'Two Bedroom Effective Rent/SF', '% 2-Bed']].
↪describe()
```

[36]:

	Two Bedroom Avg SF	Number Of 2 Bedrooms	Two Bedroom Vacant Units
count	12612.000000	12618.000000	12617.000000
mean	1038.878925	114.991916	7.677261
std	152.890282	67.162461	11.796372
min	240.000000	1.000000	0.000000
25%	937.000000	72.000000	2.000000
50%	1030.000000	104.000000	5.000000
75%	1137.000000	144.000000	9.000000
max	2833.000000	1140.000000	280.000000

	Two Bedroom Vacancy %	Two Bedroom Effective Rent/SF	% 2-Bed
count	12617.000000	12612.000000	12618.000000
mean	6.614211	1.184188	45.374155
std	7.954260	0.441070	17.216202
min	0.000000	0.260000	0.300000
25%	2.800000	0.950000	34.210000
50%	5.100000	1.110000	44.875000
75%	7.700000	1.310000	54.707500
max	100.000000	15.680000	100.000000

Missing values for Avg SF, Vacant Units, Vacancy % and Effective Rent/SF. Outliers in Avg SF and Effective Rent/SF.

[37]:
```
# Fill missing values with predefined functions
for level in levels:
    fill_vacancy_mean('2-Bed', 'Two Bedroom', level)
    fill_sf_mean('2-Bed', 'Two Bedroom', level)
calculate_vacant_unit('2-Bed', 'Two Bedroom', '2 Bedrooms')

sub[sub['% 2-Bed'].notnull()][['Two Bedroom Avg SF','Number Of 2 Bedrooms',␣
↪'Two Bedroom Vacant Units',
       'Two Bedroom Vacancy %', 'Two Bedroom Effective Rent/SF', '% 2-Bed']].
↪describe()
```

[37]:
```
       Two Bedroom Avg SF  Number Of 2 Bedrooms  Two Bedroom Vacant Units  \
count        12618.000000          12618.000000              12618.000000
mean          1038.841097            114.991916                  7.676811
std            152.872694             67.162461                 11.796013
min            240.000000              1.000000                  0.000000
25%            937.000000             72.000000                  2.000000
50%           1030.000000            104.000000                  5.000000
75%           1137.000000            144.000000                  9.000000
max           2833.000000           1140.000000                280.000000

       Two Bedroom Vacancy %  Two Bedroom Effective Rent/SF        % 2-Bed
count           12618.000000                   12612.000000   12618.000000
mean                6.614069                       1.184188      45.374155
std                 7.953961                       0.441070      17.216202
min                 0.000000                       0.260000       0.300000
25%                 2.800000                       0.950000      34.210000
50%                 5.100000                       1.110000      44.875000
75%                 7.700000                       1.310000      54.707500
max               100.000000                      15.680000     100.000000
```

[38]:
```
# Check if 2-Bedroom Vacant units match with 2-Bedroom Vacancy %
test = sub[sub['% 2-Bed'].notnull()].copy()
test['# Vacant 2-Bed'] = test[['Number Of 2 Bedrooms', 'Two Bedroom Vacancy␣
↪%']].apply(lambda x:
                          round(x[0]*x[1]/100), axis=1)
print(test[test['Two Bedroom Vacant Units']!=test['# Vacant 2-Bed']].shape)
#print(test[test['Two Bedroom Vacant Units']!=test['# Vacant 2-Bed']]\
#[['Two Bedroom Vacant Units', '# Vacant 2-Bed']].count())
```

```
(0, 64)
```

Two Bedroom Vacant Units and Two Bedroom Vacancy % matches.

0.1.15 1.5.4 Three Bedroom

[39]:
```
# Check descriptive statistics of 3-Bedroom related variables
sub[sub['% 3-Bed'].notnull()][['Three Bedroom Avg SF','Number Of 3 Bedrooms',␣
↪'Three Bedroom Vacant Units',
       'Three Bedroom Vacancy %', 'Three Bedroom Effective Rent/SF', '%␣
↪3-Bed']].describe()
```

[39]:
```
       Three Bedroom Avg SF  Number Of 3 Bedrooms  Three Bedroom Vacant Units  \
count           7782.000000           7788.000000                 7785.000000
mean            1315.384477             41.302390                    2.726654
std              230.116845             36.489098                    4.733851
min              346.000000              1.000000                    0.000000
25%             1184.000000             17.000000                    1.000000
50%             1300.000000             32.000000                    2.000000
75%             1413.000000             54.000000                    3.000000
max             4099.000000            785.000000                  126.000000

       Three Bedroom Vacancy %  Three Bedroom Effective Rent/SF      % 3-Bed
count              7785.000000                      7782.000000  7788.000000
mean                  7.423353                         1.142404    16.790705
std                   9.016284                         0.436008    13.988808
min                   0.000000                         0.230000     0.120000
25%                   2.800000                         0.900000     7.155000
50%                   5.800000                         1.070000    12.800000
75%                   8.700000                         1.270000    22.452500
max                 100.000000                         9.460000   100.000000
```

Missing values for Avg SF, Vacant Units, Vacancy % and Effective Rent/SF. Outliers in Avg SF and Effective Rent/SF.

[40]:
```
# Fill missing values with predefined functions
for level in levels:
    fill_vacancy_mean('3-Bed', 'Three Bedroom', level)
    fill_sf_mean('3-Bed', 'Three Bedroom', level)
calculate_vacant_unit('3-Bed', 'Three Bedroom', '3 Bedrooms')

sub[sub['% 3-Bed'].notnull()][['Three Bedroom Avg SF','Number Of 3 Bedrooms',␣
↪'Three Bedroom Vacant Units',
       'Three Bedroom Vacancy %', 'Three Bedroom Effective Rent/SF', '%␣
↪3-Bed']].describe()
```

[40]:
```
       Three Bedroom Avg SF  Number Of 3 Bedrooms  Three Bedroom Vacant Units  \
count           7788.000000           7788.000000                 7788.000000
mean            1315.286926             41.302390                    2.726502
std              230.063005             36.489098                    4.732981
min              346.000000              1.000000                    0.000000
25%             1183.000000             17.000000                    1.000000
```

```
50%        1300.000000         32.000000                     2.000000
75%        1413.000000         54.000000                     3.000000
max        4099.000000        785.000000                   126.000000

       Three Bedroom Vacancy %  Three Bedroom Effective Rent/SF     % 3-Bed
count              7788.000000                      7782.000000  7788.000000
mean                  7.422400                         1.142404    16.790705
std                   9.014811                         0.436008    13.988808
min                   0.000000                         0.230000     0.120000
25%                   2.800000                         0.900000     7.155000
50%                   5.800000                         1.070000    12.800000
75%                   8.700000                         1.270000    22.452500
max                 100.000000                         9.460000   100.000000
```

[41]:
```
# Check if 3-Bedroom Vacant units match with 3-Bedroom Vacancy %
test = sub[sub['% 3-Bed'].notnull()].copy()
test['# Vacant 3-Bed'] = test[['Number Of 3 Bedrooms', 'Three Bedroom Vacancy␣
 ↪%']].apply(lambda x:
                        round(x[0]*x[1]/100), axis=1)
print(test[test['Three Bedroom Vacant Units']!=test['# Vacant 3-Bed']].shape)
#print(test[test['Three Bedroom Vacant Units']!=test['# Vacant 3-Bed']]\
#[['Three Bedroom Vacant Units', '# Vacant 3-Bed']].count())
```

```
(0, 64)
```

Three Bedroom Vacant Units and Three Bedroom Vacancy % matches.

0.1.16 1.5.5 Four Bedroom

[42]:
```
# Check descriptive statistics of 4-Bedroom related variables
sub[sub['% 4-Bed'].notnull()][['Four Bedroom Avg SF','Number Of 4 Bedrooms',␣
 ↪'Four Bedroom Vacant Units',
       'Four Bedroom Vacancy %', 'Four Bedroom Effective Rent/SF', '% 4-Bed']].
 ↪describe()
```

[42]:
```
       Four Bedroom Avg SF  Number Of 4 Bedrooms  Four Bedroom Vacant Units  \
count           883.000000            887.000000                 884.000000
mean           1467.035108             50.519718                   4.167421
std             335.345204             60.483560                  11.345157
min             240.000000              0.992800                   0.000000
25%            1300.000000             12.000000                   0.000000
50%            1424.000000             26.000000                   1.000000
75%            1582.000000             72.000000                   3.000000
max            3504.000000            501.000000                 180.000000

       Four Bedroom Vacancy %  Four Bedroom Effective Rent/SF     % 4-Bed
count              884.000000                      883.000000  887.000000
mean                 6.629072                        1.215764   22.955642
```

```
std            10.893822                1.168431   25.108779
min             0.000000                0.160000    0.190000
25%             0.000000                0.790000    5.560000
50%             3.950000                0.970000   12.500000
75%             8.300000                1.320000   31.925000
max           100.000000               24.410000  100.000000
```

Missing values for Avg SF, Vacant Units, Vacancy % and Effective Rent/SF. Outliers in Avg SF and Effective Rent/SF.

[43]:

```
# Fill missing values with predefined functions
for level in levels:
    fill_vacancy_mean('4-Bed', 'Four Bedroom', level)
    fill_sf_mean('4-Bed', 'Four Bedroom', level)
calculate_vacant_unit('4-Bed', 'Four Bedroom', '4 Bedrooms')

sub[sub['% 4-Bed'].notnull()][['Four Bedroom Avg SF','Number Of 4 Bedrooms',␣
 ↪'Four Bedroom Vacant Units',
        'Four Bedroom Vacancy %', 'Four Bedroom Effective Rent/SF', '% 4-Bed']].
 ↪describe()
```

[43]:

```
       Four Bedroom Avg SF  Number Of 4 Bedrooms  Four Bedroom Vacant Units  \
count           887.000000            887.000000                 887.000000
mean           1466.612534             50.519718                   4.158963
std             334.760427             60.483560                  11.327696
min             240.000000              0.992800                   0.000000
25%            1300.000000             12.000000                   0.000000
50%            1424.000000             26.000000                   1.000000
75%            1581.000000             72.000000                   3.000000
max            3504.000000            501.000000                 180.000000

       Four Bedroom Vacancy %  Four Bedroom Effective Rent/SF      % 4-Bed
count              887.000000                      883.000000   887.000000
mean                 6.619457                        1.215764    22.955642
std                 10.877986                        1.168431    25.108779
min                  0.000000                        0.160000     0.190000
25%                  0.000000                        0.790000     5.560000
50%                  4.000000                        0.970000    12.500000
75%                  8.300000                        1.320000    31.925000
max                100.000000                       24.410000   100.000000
```

[44]:

```
# Check if 4-Bedroom Vacant units match with 4-Bedroom Vacancy %
test = sub[sub['% 4-Bed'].notnull()].copy()
test['# Vacant 4-Bed'] = test[['Number Of 4 Bedrooms', 'Four Bedroom Vacancy␣
 ↪%']].apply(lambda x:
                              round(x[0]*x[1]/100), axis=1)
print(test[test['Four Bedroom Vacant Units']!=test['# Vacant 4-Bed']].shape)
```

```
(0, 64)
```

Four Bedroom Vacant Units and Four Bedroom Vacancy % matches.

0.1.17 1.6 Fill Vacancy % missing values with calculated values

Calculate total vacant units, then divide it by total number of units

[45]:
```
# Check Vacancy % missing values (if < 12908)
sub['Vacancy %'].count()
```

[45]:
```
12255
```

[46]:
```
# Fill missing values with values calculated from related fields
sub['#_Vacant_Units'] = sub[['Studio Vacant Units',
                             'One Bedroom Vacant Units',
                             'Two Bedroom Vacant Units',
                             'Three Bedroom Vacant Units',
                             'Four Bedroom Vacant Units']].fillna(0).sum(axis=1)
sub['Vacancy_%'] = sub[['#_Vacant_Units','Number Of Units']].apply(lambda x:␣
 ↪round(x[0]/x[1]*100,2), axis=1)
sub['Vacancy %'].fillna(sub['Vacancy_%'], inplace=True)
sub[['Vacancy %','Vacancy_%']].count()
```

[46]:
```
Vacancy %     12908
Vacancy_%     12908
dtype: int64
```

[47]:
```
# Check % mismatch in original vacancy% and calculated vacancy% field
print("Percentage of rows where calculated vacancy % doesn't match original␣
 ↪vacancy %:",
      sub[abs(sub['Vacancy %']-sub['Vacancy_%'])>0.05].shape[0]/sub.shape[0])
```

```
Percentage of rows where calculated vacancy % doesn't match original vacancy %:
0.051750852184691665
```

0.1.18 1.7 Fill Avg Unit SF missing values with calculated values

Calculated as weighted average of the average unit size of diffrent unit types

[48]:
```
# Check Avg Unit SF missing values (if < 12908)
sub['Avg Unit SF'].count()
```

[48]:
```
12904
```

[49]:
```
# Calculate weighted average Unit SF based on related fields
sub['Avg_Unit_SF'] = sub[['% Studios','% 1-Bed', '% 2-Bed', '% 3-Bed', '%␣
 ↪4-Bed',
```

```
                            'Studio Avg SF', 'One Bedroom Avg SF', 'Two Bedroom␣
↪Avg SF',
                            'Three Bedroom Avg SF', 'Four Bedroom Avg SF']].
↪fillna(0).\
                      apply(lambda x:␣
↪round((x[0]*x[5]+x[1]*x[6]+x[2]*x[7]+x[3]*x[8]+x[4]*x[9])/100), axis=1)
```

[50]:
```
# Fill missing values with calculated  values
sub['Avg Unit SF'].fillna(sub['Avg_Unit_SF'], inplace=True)
sub[['Avg Unit SF', 'Avg_Unit_SF']].count()
```

[50]:
```
Avg Unit SF    12908
Avg_Unit_SF    12908
dtype: int64
```

[51]:
```
# Check % mismatch in original Avg Unit SF and calculated Avg Unit SF fields
print("Percentage of rows where calculated Avg Unit SF doesn't match original␣
↪Avg Unit SF:",
      sub[abs(sub['Avg Unit SF']-sub['Avg_Unit_SF'])>10].shape[0]/sub.shape[0])
```

```
Percentage of rows where calculated Avg Unit SF doesn't match original Avg Unit
SF: 0.009683916950728231
```

0.1.19 1.8 Define functions to fill missing value with mean or median

The functions can be used at different region level (zip code, city, county, market)

[52]:
```
def fill_mean(var, level):
    sub[var] = sub[[level, var]].groupby(level).transform(lambda x: x.fillna(x.
↪mean()))

def fill_median(var, level):
    sub[var] = sub[[level, var]].groupby(level).transform(lambda x: x.fillna(x.
↪median()))
```

0.1.20 1.9 Fill 'Closest Transit Stop Dist (mi)' missing values

- Fill with regional mean
- Manually calculate the remaining missing values

[53]:
```
# Check descriptive statistics of Closest Transit Stop Dist (mi)
sub['Closest Transit Stop Dist (mi)'].describe() # mean close to median
```

[53]:
```
count    12176.000000
mean        14.540384
std          9.613232
min          0.010000
25%          6.740000
```

```
50%          13.495000
75%          20.630000
max          45.360000
Name: Closest Transit Stop Dist (mi), dtype: float64
```

[54]:
```
# Fill missing values
var = 'Closest Transit Stop Dist (mi)'
for level in levels:
    fill_mean(var, level)
sub['Closest Transit Stop Dist (mi)'].count()
```

[54]:
```
12860
```

Manually find the closest transit and calculate distance.

[55]:
```
# Find the Market Names where no closest transit stops are assigned
# Go back to the original dataset to find the closest transit stops for those␣
 ↪markets
question_markets = sub[sub['Closest Transit Stop Dist (mi)'].isnull()]['Market␣
 ↪Name'].unique().tolist()
data[data['Market Name'].isin(question_markets)][['Market Name', 'Closest␣
 ↪Transit Stop']].drop_duplicates()
```

[55]:

Row	Market Name	Closest Transit Stop
7654	Ocala	NaN
7667	Port St Lucie/Fort Pierce	NaN
7716	Ocala	Gainesville Regional
7854	Port St Lucie/Fort Pierce	Palm Beach International

Use Palm Beach International as 'Closest Transit Stop' for Port St Lucie/Fort Pierce and Gainesville Regional for Ocala. Calculate 'Closest Transit Stop Dist (mi)'.

[56]:
```
# Define a function using Latitude and Longitude information to calculate the␣
 ↪distances manually

# http://www.lat-long.com/
 ↪Latitude-Longitude-309962-Florida-Palm_Beach_International_Airport.html
PalmBeach = (26.683399, -80.095320) # lat, lon
# https://www.prokerala.com/travel/airports/united-states-of-america/
 ↪gainesville-regional-airport.html
Gainesville = (29.686, -82.2768)

def calculate_distance(p1, p2):
    lat1 = p1[0]
    lat0 = p2[0]
    lon1 = p1[1]
```

```
    lon0 = p2[1]

    # calculation: https://stackoverflow.com/questions/28994289/
↪calculate-euclidean-distance-with-google-maps-coordinates
    deglen = 69 #mile
    x = lat1 - lat0
    yy = lon1 - lon0
    y = (yy) * math.cos(lat0)
    dist = deglen * math.sqrt(x*x + y*y)
    return dist
```

[57]:
```
# Apply the distance function to fill missing values
sub.loc[sub['Closest Transit Stop Dist (mi)'].isnull(),'Closest Transit Stop␣
↪Dist (mi)']\
= sub.loc[sub['Closest Transit Stop Dist (mi)'].isnull(),
          ['Closest Transit Stop Dist (mi)', 'Market Name', 'Latitude',␣
↪'Longitude']].\
apply(lambda x: x[0] if np.isnan(x[0])==False else\
      (calculate_distance((x[2],x[3]), PalmBeach) if x[1]=='Port St Lucie/Fort␣
↪Pierce'\
       else calculate_distance((x[2],x[3]), Gainesville)), axis=1)
sub['Closest Transit Stop Dist (mi)'].count()
```

[57]: 12908

0.1.21 1.10 Fix Land Area (AC) wrong entry, fill missing values with regional median

[58]:
```
# Check descriptive statistics for Land Area (AC)
sub['Land Area (AC)'].describe() # extremely large value!!! use median due to␣
↪presence of outlier
```

[58]:
```
count    12426.000000
mean        18.515888
std        407.179133
min          0.001000
25%          6.850250
50%         11.358000
75%         18.419750
max      45318.000000
Name: Land Area (AC), dtype: float64
```

45318 acres of land area for a rental property is extremely unusual. Resaonable guess would be that it should be square feet instead of acres.

[59]:
```
# Update wrong entry
acre_to_sf = 43560
sub.loc[sub['Land Area (AC)']==45318, 'Land Area (AC)'] = 45318/acre_to_sf
```

[60]:
```python
# Check land Area extreme values
sub['Land Area (AC)'].sort_values(ascending=False).head(10)
```

[60]:
```
Row
16170    1584.0000
13395    1060.4178
10925    1000.0000
3477      820.3600
14197     481.1000
4484      410.7200
4024      329.0000
9788      272.5155
20198     263.1573
6913      246.9915
Name: Land Area (AC), dtype: float64
```

[61]:
```python
# Check 99 percentile for Land Area
sub['Land Area (AC)'].quantile(0.99)
```

[61]:
```
58.96
```

[62]:
```python
sub['Land Area (AC)'].describe()
```

[62]:
```
count    12426.000000
mean        14.868941
std         25.043607
min          0.001000
25%          6.850000
50%         11.355000
75%         18.417425
max       1584.000000
Name: Land Area (AC), dtype: float64
```

Too many outliers, very likely to mess up the model, should be excluded.

[63]:
```python
# Fill missing values with regional median
var = 'Land Area (AC)'
for level in levels:
    fill_median(var, level)
sub['Land Area (AC)'].count()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
```

```
  r = func(a, **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
```

[63]:
```
12908
```

0.1.22 1.11 Fill Number Of Stories missing values

- Fill with building style median
- Drop the rows where both number of stories and building style are missing

[64]:
```python
# Check descriptive statistics for Number of Stories
sub['Number Of Stories'].describe() # mean and median are close
```

[64]:
```
count    12892.000000
mean         3.097114
std          3.036233
min          1.000000
25%          2.000000
50%          3.000000
75%          3.000000
max         85.000000
Name: Number Of Stories, dtype: float64
```

[65]:
```python
# Find building style associated to those properties missing values in Number␣
↪of Stories
ref = sub[sub['Number Of Stories'].isnull()]['Style']
ref.head()
```

[65]:
```
Row
2205      Garden
2764    Low-Rise
4539         NaN
9570         NaN
9811      Garden
Name: Style, dtype: object
```

[66]:
```python
# Calculate median number of stories for each style
style_story = sub[['Number Of Stories','Style']].groupby('Style').median()
```

[67]:
```python
# Fill missing values with style median
for ind in ref.index:
    style = sub.loc[ind, 'Style']
    if style in style_story.index:
        sub.loc[ind, 'Number Of Stories'] = style_story.loc[style]['Number Of␣
↪Stories']
```

```
sub['Number Of Stories'].count()
```

[67]: 12903

Drop the remaining 5 rows with missing Number Of Stories value because filling it with region mean or median doesn't make sense.

[68]:
```
# Drop rows with no building style information
row_initial = sub.shape[0]
sub = sub[sub['Number Of Stories'].notnull()].copy()
print('Records dropped:', row_initial - sub.shape[0])
print('Current size:', sub.shape)
```

```
Records dropped: 5
Current size: (12903, 66)
```

0.1.23 1.12 Fill Year Built missing values with regional median

[69]:
```
# Check descriptive statistics Year Built
sub['Year Built'].describe()
```

[69]:
```
count    12875.000000
mean      1991.660272
std         16.528065
min       1881.000000
25%       1979.000000
50%       1991.000000
75%       2006.000000
max       2019.000000
Name: Year Built, dtype: float64
```

[70]:
```
# Fill missing values with median
var = 'Year Built'
for level in levels:
    fill_median(var, level)
sub['Year Built'].count()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
```

```
encountered
  r = func(a, **kwargs)
```

[70]: 12903

0.1.24 1.13 Fill Year Renovated missing values with Year Built

[71]:
```
# Check Year Renovated non null values
sub['Year Renovated'].count()
```

[71]: 1822

[72]:
```
# Replace Year Renovated missing values with Year Built
sub['Year Renovated'] = sub['Year Renovated'].fillna(sub['Year Built'])
sub['Year Renovated'].count()
```

[72]: 12903

0.1.25 1.14 Fill Average Age missing values with regional mean

[73]:
```
# Check description of 2019 Averge Age (1m) - 0 are missing values, fill with␣
↪mean
sub['2019 Avg Age(1m)'].describe()
```

[73]:
```
count    12903.000000
mean        35.845393
std          4.713318
min          0.000000
25%         33.500000
50%         35.600000
75%         38.000000
max         75.600000
Name: 2019 Avg Age(1m), dtype: float64
```

[74]:
```
# Find number of rows with missing Average age value
sub[sub['2019 Avg Age(1m)']==0].shape
```

[74]: (59, 66)

[75]:
```
# Fill missing values with regional mean
sub['2019 Avg Age(1m)'].replace(0, np.nan, inplace=True)

var = '2019 Avg Age(1m)'
for level in levels:
    fill_mean(var, level)

sub['2019 Avg Age(1m)'].describe()
```

```
[75]: count    12903.000000
      mean        36.009760
      std          4.049772
      min         21.500000
      25%         33.500000
      50%         35.600000
      75%         38.000000
      max         75.600000
      Name: 2019 Avg Age(1m), dtype: float64
```

0.1.26 1.15 Calculate regional population

- Calculate total population within 1 mile
- 59 missing values (0), fill with median (right skewed distribution)

```
[76]: # Check non null values for population variables
      sub[['2019 Pop Age <19(1m)', '2019 Pop Age 20-64(1m)','2019 Pop Age 65+(1m)']].
       ↪count()
```

```
[76]: 2019 Pop Age <19(1m)      12903
      2019 Pop Age 20-64(1m)    12903
      2019 Pop Age 65+(1m)      12903
      dtype: int64
```

```
[77]: # Calculate total population and check descriptive statistics
      sub['2019 Pop Tot'] = sub[['2019 Pop Age <19(1m)', '2019 Pop Age␣
       ↪20-64(1m)','2019 Pop Age 65+(1m)']].sum(axis=1)
      sub['2019 Pop Tot'].describe()
```

```
[77]: count    12903.000000
      mean     14131.631636
      std       8760.118586
      min          0.000000
      25%       8017.000000
      50%      12593.000000
      75%      18461.500000
      max      89490.000000
      Name: 2019 Pop Tot, dtype: float64
```

```
[78]: # Checking number of rows where total population is 0 (missing values)
      sub[sub['2019 Pop Tot']==0].shape
```

```
[78]: (59, 67)
```

```
[79]: # Fill missing values with regional median
      sub['2019 Pop Tot'].replace(0, np.nan, inplace=True)
```

```python
var = '2019 Pop Tot'
for level in levels:
    fill_median(var, level)

sub['2019 Pop Tot'].describe()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/function_base.py:3250: RuntimeWarning: All-NaN slice
encountered
  r = func(a, **kwargs)
```

[79]:
```
count    12903.000000
mean     14168.887584
std       8722.713445
min          9.000000
25%       8064.000000
50%      12613.000000
75%      18464.000000
max      89490.000000
Name: 2019 Pop Tot, dtype: float64
```

Some area has very few people (9). Since it's 1 mile radius, can be true.

1 2. External Data

1.0.1 2.1 Import Income, marriage % and male/female variables

[80]:
```python
# Import demographic data
demo = pd.read_csv('income_marriage.csv')
```

[81]:
```python
demo.head()
```

[81]:

	Zip5	MedanHHIncome(000)	married %	male/female
0	30097.0	96.9	66.029206	0.938446
1	30318.0	44.0	24.433453	1.190945
2	30309.0	78.3	32.646975	1.114291
3	30363.0	66.9	28.629997	1.202408
4	30328.0	80.8	50.854275	0.831041

[82]:
```
demo.shape
```

[82]:
```
(1925, 4)
```

[83]:
```
# Convert Zip to int type in both demo and sub
demo['Zip5'] = demo['Zip5'].astype(int)
sub['Zip5'] = sub['Zip5'].astype(int)
```

[84]:
```
sub.shape
```

[84]:
```
(12903, 67)
```

[85]:
```
# Merge demographic data into property dataset
sub = sub.merge(demo, how='left')
sub.shape
```

[85]:
```
(12903, 70)
```

[86]:
```
# Check descriptive statistics and look for missing values
sub[['MedanHHIncome(000)','married %','male/female']].describe()
```

[86]:

	MedanHHIncome(000)	married %	male/female
count	12891.000000	12895.000000	12895.000000
mean	55.647816	43.377618	0.955707
std	20.958572	10.792473	0.191937
min	13.300000	4.905718	0.579598
25%	41.300000	36.678515	0.885099
50%	50.900000	43.698003	0.934211
75%	66.200000	50.166837	0.990373
max	250.000000	76.405127	4.852663

12 missing values in MedanHHIncome(000), 8 missing values in married % and male/female each.

[87]:
```
# Fill missing values with county mean/median
fill_median('MedanHHIncome(000)', 'County Name')
fill_mean('married %', 'County Name')
fill_mean('male/female', 'County Name')
```

[88]:
```
# Check if no missing values
sub[['MedanHHIncome(000)','married %','male/female']].count()
```

[88]:
```
MedanHHIncome(000)    12903
married %             12903
male/female           12903
dtype: int64
```

1.0.2 2.2 Import Per Capita Deposit/Saving variable

[89]:
```
# Import Per Capita Deposit data
deposit = pd.read_csv('per_capita_deposit.csv')
```

[90]:
```
deposit.head()
```

[90]:
```
  State     County  Deposit (000s)  Population Est 2018  \
0    GA     Fulton       100332784              1050114
1    GA   Gwinnett        17717075               927781
2    GA       Cobb        15632932               756865
3    GA     DeKalb        12481873               756558
4    GA   Muscogee         8394232               194160

   Deposit (000s) Per Capita
0                  95.544659
1                  19.096182
2                  20.654849
3                  16.498237
4                  43.233581
```

[91]:
```
# Check descriptive statistics
print(deposit.shape)
deposit.describe()
```

```
(572, 5)
```

[91]:
```
       Deposit (000s)  Population Est 2018  Deposit (000s) Per Capita
count    5.720000e+02         5.720000e+02                 572.000000
mean     3.674213e+06         1.238987e+05                  18.570632
std      1.751788e+07         3.437498e+05                  13.667547
min      7.602000e+03         7.260000e+02                   0.928432
25%      1.679305e+05         1.195700e+04                  10.966622
50%      4.214485e+05         2.728400e+04                  15.334716
75%      1.248876e+06         8.595475e+04                  21.664140
max      2.086604e+08         4.698619e+06                 172.786415
```

[92]:
```
# Fix county name that are named differently
deposit.rename(columns={'County':'County Name'}, inplace=True)
deposit['County Name'] = deposit['County Name'].str.replace('Miami-Dade',␣
 ↪'Miami/Dade')
deposit['County Name'] = deposit['County Name'].str.replace('St. Lucie', 'St␣
 ↪Lucie')
deposit['County Name'] = deposit['County Name'].str.replace('McLennan',␣
 ↪'Mclennan')
```

[93]:
```
# Merge deposit information into property dataset
sub = sub.merge(deposit[['State', 'County Name', 'Deposit (000s) Per Capita']],
                on=['State', 'County Name',], how='left')
sub.shape
```

[93]:
```
(12903, 71)
```

[94]:
```
# Check descriptive statistics after merging
sub['Deposit (000s) Per Capita'].describe()
```

[94]:
```
count    12903.000000
mean        40.787386
std         33.086683
min          2.236086
25%         19.571416
50%         28.355298
75%         48.374164
max        172.786415
Name: Deposit (000s) Per Capita, dtype: float64
```

The geographic area that the property dataset covers in general has higher per capita saving than the four state average.

2 3. Feature Engineering

2.0.1 3.1 Create Floor Area Ratio variable

[95]:
```
# Check descriptive statistics for RBA - Rentable Building Area
sub['RBA'].describe()
```

[95]:
```
count    1.290300e+04
mean     2.671828e+05
std      1.586218e+05
min      4.160000e+03
25%      1.607900e+05
50%      2.351200e+05
75%      3.310010e+05
max      2.468016e+06
Name: RBA, dtype: float64
```

[96]:
```
# Calculate Floor Area Ratio
acre_to_sf = 43560
sub['Floor Area Ratio'] = sub['RBA']/(sub['Land Area (AC)']*acre_to_sf)
sub['Floor Area Ratio'].describe()
```

[96]:
```
count    12903.000000
mean         1.249645
```

```
std          31.926737
min           0.001937
25%           0.323446
50%           0.447185
75%           0.637774
max        3219.100092
Name: Floor Area Ratio, dtype: float64
```

2.0.2 3.2 Create Supply variables

- Calculate number of units under same zip code
- Calculate number of vacant units under same zip code

[97]:
```
sub['Supply_all'] = sub[['Number Of Units', 'Zip5']].groupby('Zip5').\
                    transform(lambda x: x.sum())
sub['Supply_vacant'] = sub[['#_Vacant_Units', 'Zip5']].groupby('Zip5').\
                       transform(lambda x: x.sum())
```

Extreme outliers exit, should be excluded.

2.0.3 3.3 Create Owner Type, Encode

- Large owner: manages >=50 properties
- Medium owner: manages [10,50) properties
- Small owner: manages <10 properties or unspecified owner
- One hot encode Owner Type

[98]:
```
sub['Owner Name'].nunique()
```

[98]: 4352

[99]:
```
# Create a datafame to store property count for each owner
owner = sub['Owner Name'].value_counts().reset_index(name='count')
```

[100]:
```
print('large owner:', owner[owner['count']>=50].shape[0])
print('medium owner:', owner[(owner['count']>=10) & (owner['count']<50)].
 ↪shape[0])
print('small owner:', owner[owner['count']<10].shape[0])
```

```
large owner: 12
medium owner: 255
small owner: 4085
```

[101]:
```
# Create owner lists by owner type
large_owner = owner[owner['count']>=50]['index'].tolist()
medium_owner = owner[(owner['count']>=10) & (owner['count']<50)]['index'].
 ↪tolist()
small_owner = owner[owner['count']<10]['index'].tolist()
```

[102]:
```
# Assign owner type to each property
# those who are missing owner info considered as small owner
sub['Owner Type'] = sub['Owner Name'].apply(lambda x: 'large' if x in␣
↪large_owner else(
                                            'medium' if x in medium_owner else␣
↪'small'))
```

[103]:
```
# Owner Type summary
sub[['Owner Type', 'Avg Effective/SF']].groupby('Owner Type').mean().
↪reset_index().\
merge(pd.DataFrame(sub['Owner Type'].value_counts().reset_index()).\
rename(columns={'index':'Owner Type', 'Owner Type':'Count'}))
```

[103]:

	Owner Type	Avg Effective/SF	Count
0	large	1.369511	940
1	medium	1.254213	4432
2	small	1.227484	7531

[104]:
```
# CreatE dummy variables
for otype in sub['Owner Type'].unique():
    sub['Owner Type_{}'.format(otype)] = sub['Owner Type'].apply(lambda x: 1 if␣
↪x==otype else 0)
```

2.0.4 3.4 Affordable Housing Encoding

- Combine Rent Stabilized and Rent Controlled into Rent Restricted, label non-affordable properties as Market
- One hot encode Affordable Type

[105]:
```
sub['Rent Type'].count()
```

[105]:
```
12903
```

[106]:
```
sub['Affordable Type'].value_counts()
```

[106]:
```
Rent Restricted     1395
Rent Subsidized      458
Affordable Units     103
Rent Stabilized        5
Rent Controlled        2
Name: Affordable Type, dtype: int64
```

[107]:
```
# Combining Rent Stabilized and Controlled as Rent Restricted and fill NULL␣
↪values as Market
sub['Affordable Type*'] = sub['Affordable Type'].fillna('Market')
sub['Affordable Type*'] = sub['Affordable Type*'].map({'Rent Stabilized':'Rent␣
↪Restricted',
```

```
                                                    'Rent Controlled':'Rent␣
↪Restricted',
                                                    'Rent Restricted':'Rent␣
↪Restricted',
                                                    'Market':'Market',
                                                    'Rent Subsidized':'Rent␣
↪Subsidized',
                                                    'Affordable Units':
↪'Affordable Units'})
```

[108]:
```python
# Affordable Type Summary
sub[['Affordable Type*', 'Avg Effective/SF']].groupby('Affordable Type*').
↪mean().reset_index().\
merge(pd.DataFrame(sub['Affordable Type*'].value_counts().reset_index()).\
rename(columns={'index':'Affordable Type*', 'Affordable Type*':'Count'})).\
sort_values('Avg Effective/SF', ascending=False)
```

[108]:

	Affordable Type*	Avg Effective/SF	Count
1	Market	1.287774	10940
0	Affordable Units	1.192039	103
3	Rent Subsidized	1.107860	458
2	Rent Restricted	0.978431	1402

[109]:
```python
# Create Dummy Variables
for atype in sub['Affordable Type*'].unique():
    sub['Affordable Type_{}'.format(atype)] = sub['Affordable Type*'].
↪apply(lambda x: 1 if x==atype else 0)
```

2.0.5 3.5 State and City Encoding

- One hot encode State
- Group cities with <= 100 properties in it into Other by state, resulting in 23 city groups
- One hot encode City

[110]:
```python
# State summary
sub[['State', 'Avg Effective/SF']].groupby('State').mean().reset_index().\
merge(pd.DataFrame(sub['State'].value_counts().reset_index()).\
rename(columns={'index':'State', 'State':'Count'})).\
sort_values('Avg Effective/SF', ascending=False)
```

[110]:

	State	Avg Effective/SF	Count
0	FL	1.325436	2993
3	TX	1.250045	6418
1	GA	1.178725	1866
2	NC	1.169047	1626

[111]:
```
# Create Dummy Variables for state
for stype in sub['State'].unique():
    sub['State_{}'.format(stype)] = sub['State'].apply(lambda x: 1 if x==stype␣
 ↪else 0)
```

[112]:
```
sub['City'].nunique()
```

[112]:
```
617
```

[113]:
```
# Check number of cities with more than 100 properties
city = sub['City'].value_counts().reset_index()
city.columns = ['City', 'Count']
city[city['Count']>100].shape
```

[113]:
```
(19, 2)
```

[114]:
```
# Cities with less than 10 properties
print('{}% cities have less than 10 properties in it'.\
      format(round(city[city['Count']<10].shape[0]/sub['City'].
 ↪nunique()*100,1)))
```

```
68.6% cities have less than 10 properties in it
```

[115]:
```
# Combine cities with fewer than 100 properties for each state
large_city = city[city['Count']>100]['City'].unique()
sub['City*'] = sub[['City','State']].apply(lambda x: x[0] if x[0] in large_city\
                                          else '{} Other'.format(x[1]), axis=1)
```

[116]:
```
sub['City*'].nunique()
```

[116]:
```
23
```

[117]:
```
# Check if the smallest cities have enough property count
sub['City*'].value_counts().tail()
```

[117]:
```
Greensboro     134
El Paso        125
Durham         120
Tallahassee    106
Plano          104
Name: City*, dtype: int64
```

[118]:
```
# Create dummy variables for cities
for ctype in sub['City*'].unique():
    sub['City_{}'.format(ctype)] = sub['City*'].apply(lambda x: 1 if x==ctype␣
 ↪else 0)
```

2.0.6 3.6 Construction Material

- Combine Steel and Metal into Steel or Metal
- One hot encode Construction Material

[119]:
```
sub['Construction Material'].value_counts()
```

[119]:
```
Wood Frame             7135
Masonry                4500
Reinforced Concrete    1074
Steel                   168
Metal                    26
Name: Construction Material, dtype: int64
```

[120]:
```
# Combine Steel and Metal
sub['Construction Material*'] = sub['Construction Material'].map({'Wood Frame':
 ↪'Wood Frame',
                                                                    'Masonry':
 ↪'Masonry',
                                                                    'Reinforced␣
 ↪Concrete':'Reinforced Concrete',
                                                                    'Steel':
 ↪'Steel or Metal',
                                                                    'Metal':
 ↪'Steel or Metal'})
```

[121]:
```
# Construction Material summary
sub[['Construction Material*', 'Avg Effective/SF']].groupby('Construction␣
 ↪Material*').mean().reset_index().\
merge(pd.DataFrame(sub['Construction Material*'].value_counts().reset_index()).\
rename(columns={'index':'Construction Material*', 'Construction Material*':
 ↪'Count'})).\
sort_values('Avg Effective/SF', ascending=False)
```

[121]:

	Construction Material*	Avg Effective/SF	Count
2	Steel or Metal	1.881340	194
1	Reinforced Concrete	1.566285	1074
0	Masonry	1.232227	4500
3	Wood Frame	1.191030	7135

[122]:
```
# Create dummy variables for construction material
for material in sub['Construction Material*'].unique():
    sub['Construction Material_{}'.format(material)] = \
    sub['Construction Material*'].apply(lambda x: 1 if x==material else 0)
```

2.0.7 3.7 Amenities w/o grouping

[123]:
```
# Parse all amenities
amenities = {}
all_amenities = sub['Amenities'].str.split(', ').tolist()
for row in all_amenities:
    #print(row)
    for item in row:
        #print(item)
        if item not in amenities.keys():
            amenities[item] = 1
        else:
            amenities[item] += 1
len(amenities)
```

[123]: 93

[124]:
```
# Calculate rent with or without a given amenity
amenity_vs_rent = pd.DataFrame()

for amenity in amenities:
    yes = sub[sub['Amenities'].str.contains(amenity)]['Avg Effective/SF'].mean()
    no = sub[-sub['Amenities'].str.contains(amenity)]['Avg Effective/SF'].mean()
    count = sub[sub['Amenities'].str.contains(amenity)]['Avg Effective/SF'].
 ↪count()
    amenity_vs_rent = amenity_vs_rent.append(pd.
 ↪DataFrame([amenity,count,yes,no,yes-no]).T,
                                              ignore_index=True)

amenity_vs_rent.columns = ['amenity','count','yes', 'no', 'diff']
amenity_vs_rent = amenity_vs_rent.sort_values('diff')
amenity_vs_rent.tail()
```

[124]:

	amenity	count	yes	no	diff
92	Walk To Campus	1	2.12	1.24694	0.873056
86	LEED Certified	43	2.17814	1.2439	0.934241
77	LEED Certified - Silver	21	2.24381	1.24539	0.998423
56	Meal Service	53	2.36264	1.24241	1.12023
91	Study Lounge	4	5.165	1.2458	3.9192

[125]:
```
# Top 15 most popular amenities
amenity_vs_rent.sort_values('count', ascending=False).head(15)
```

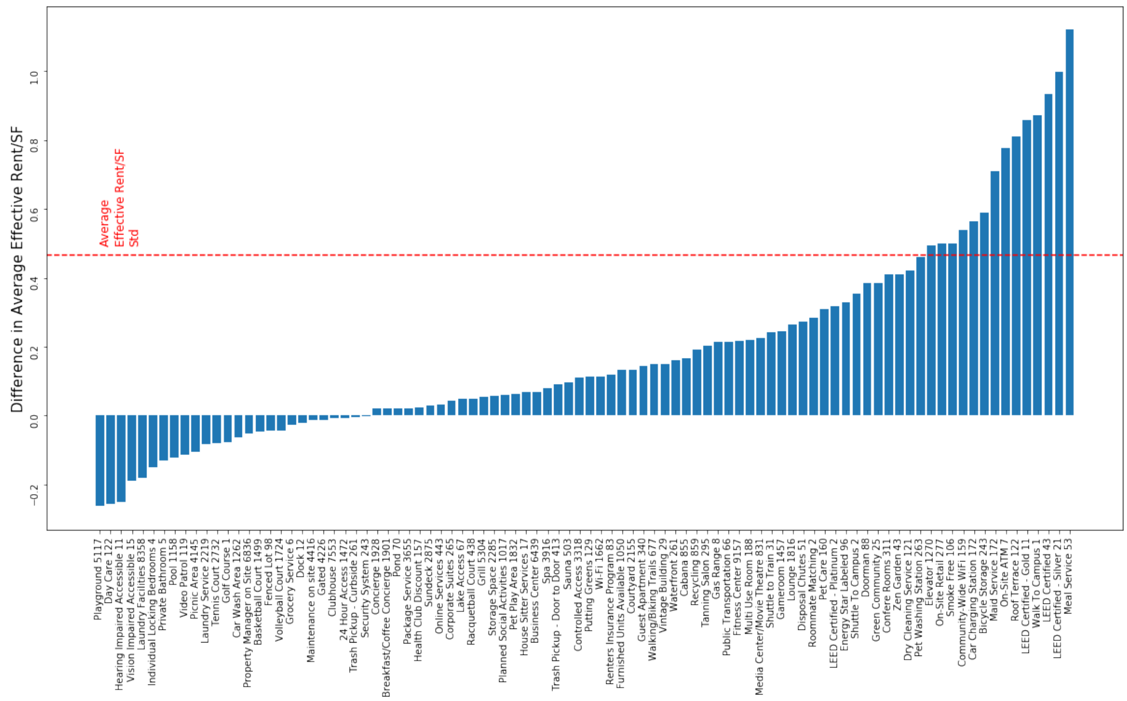
[125]:

	amenity	count	yes	no	diff
3	Fitness Center	9157	1.30969	1.0938	0.215894
4	Laundry Facilities	8358	1.18304	1.36465	-0.181611
2	Clubhouse	7553	1.2436	1.25183	-0.00822814

```
6   Property Manager on Site  6836  1.22259  1.27452  -0.0519309
0            Business Center  6439  1.28083  1.21332   0.0675079
8                      Grill  5304  1.27989  1.22407   0.0558219
5                 Playground  5117  1.08825  1.35135   -0.263099
13       Maintenance on site  4416  1.23916   1.2511  -0.0119405
18                     Gated  4226  1.23899  1.25092  -0.0119289
16               Picnic Area  4145  1.17653  1.28037   -0.103838
37                       Spa  3916  1.30311  1.22257   0.0805396
10           Package Service  3655  1.26297   1.2407   0.0222673
1          Controlled Access  3318  1.32833  1.21886    0.109464
14                   Sundeck  2875  1.27052  1.24027    0.030246
42              Tennis Court  2732  1.18383  1.26398  -0.0801542
```

[126]:

```
# Plotting rent difference with vs without amenities
test = amenity_vs_rent[amenity_vs_rent['amenity']!='Study Lounge']
xticklabel = test[['amenity','count']].apply(lambda x: x[0]+' '+str(x[1]),␣
↪axis=1)

plt.figure(figsize=(20,10))
plt.bar(np.arange(test.shape[0]),
        test['diff'])
plt.xticks(np.arange(test.shape[0]),
           xticklabel,
           rotation=90)
plt.yticks(rotation=90)
plt.axhline(sub['Avg Effective/SF'].std(),
            color='red',
            linestyle='--')
plt.annotate('Average\nEffective Rent/SF\nStd',
             (0,0.5), fontsize=12, color='red',
             rotation=90)
#plt.title('Average Effective Rent/SF With vs Without Amenity', fontsize=15)
plt.ylabel('Difference in Average Effective Rent/SF', fontsize=15)
plt.show()
```



Group Amenities

- LEED Certified - Silver, LEED Certified - Gold, LEED Certified, LEED Certified - Platinum, Energy Star Labeled
- Sports: Tennis Court, Volleyball Court, Basketball Court
- Business: Business Center, Corporate Suites, Confere Rooms, Multi Use Room
- Laundry: Laundry Facilities, Laundry Service
- Spa: Spa, Sauna
- Pet: Pet Washing Station, Pet Care, Pet Play Area
- Wifi: Community-Wide WiFi, Wi-Fi

+ Less popular high impact amenities:

- Roof Terrace
- Maid Service
- Bicycle Storage
- Car Charging Station

- On-Site Retail
- Elevator

+ Top 15 popular amenities

[127]:
```
# Total Amenities after grouping and adding popular amenities
amen = ['LEED Certified - Silver', 'LEED Certified - Gold', 'LEED Certified',␣
 ↪'LEED Certified - Platinum',
```

```
        'Energy Star Labeled',
        # Sports
        'Tennis Court', 'Volleyball Court', 'Basketball Court',
        # Business
        'Business Center', 'Corporate Suites', 'Confere Rooms', 'Multi Use␣
↪Room',
        # Laundry
        'Laundry Facilities', 'Laundry Service',
        # Spa
        'Spa', 'Sauna',
        # Pet
        'Pet Washing Station', 'Pet Care', 'Pet Play Area',
        # Wifi
        'Community-Wide WiFi', 'Wi-Fi',
        # Less popular high impact
        'Roof Terrace', 'Maid Service', 'Bicycle Storage',
        'Car Charging Station', 'On-Site Retail', 'Elevator']
top_15_amenities = amenity_vs_rent.sort_values('count', ascending=False).
↪head(15)['amenity'].tolist()

for amenity in top_15_amenities:
    if amenity not in amen:
        amen.append(amenity)

len(amen)
```

[127]: 38

[128]:
```
# One hot encode selected amenites
# Group encoded amenities and discard original ones
for amenity in amen:
    sub['Amenity_{}'.format(amenity)] = sub['Amenities'].apply(lambda x: 1 if␣
↪amenity in x.split(', ') else 0)

sub['Amenity_LEED/Energy Star'] = sub[['Amenity_LEED Certified - Silver',
                                       'Amenity_LEED Certified - Gold',
                                       'Amenity_LEED Certified',
                                       'Amenity_LEED Certified - Platinum',
                                       'Amenity_Energy Star Labeled']].
↪max(axis=1)
sub['Amenity_Sports'] = sub[['Amenity_Tennis Court',
                             'Amenity_Volleyball Court',
                             'Amenity_Basketball Court']].max(axis=1)
sub['Amenity_Business'] = sub[['Amenity_Business Center',
                               'Amenity_Corporate Suites',
                               'Amenity_Confere Rooms',
                               'Amenity_Multi Use Room']].max(axis=1)
```

```
sub['Amenity_Laundry'] = sub[['Amenity_Laundry Facilities',
                              'Amenity_Laundry Service']].max(axis=1)
sub['Amenity_Spa/Sauna'] = sub[['Amenity_Spa',
                                'Amenity_Sauna']].max(axis=1)
sub['Amenity_Pet'] = sub[['Amenity_Pet Washing Station',
                          'Amenity_Pet Care',
                          'Amenity_Pet Play Area']].max(axis=1)
sub['Amenity_Wifi'] = sub[['Amenity_Community-Wide WiFi',
                           'Amenity_Wi-Fi']].max(axis=1)
cols = ['Amenity_LEED Certified - Silver', 'Amenity_LEED Certified - Gold',␣
 ↪'Amenity_LEED Certified',
        'Amenity_LEED Certified - Platinum', 'Amenity_Energy Star Labeled',
        'Amenity_Tennis Court', 'Amenity_Volleyball Court', 'Amenity_Basketball␣
 ↪Court',
        'Amenity_Business Center', 'Amenity_Corporate Suites',
        'Amenity_Confere Rooms', 'Amenity_Multi Use Room',
        'Amenity_Laundry Facilities', 'Amenity_Laundry Service',
        'Amenity_Spa', 'Amenity_Sauna',
        'Amenity_Pet Washing Station', 'Amenity_Pet Care', 'Amenity_Pet Play␣
 ↪Area',
        'Amenity_Community-Wide WiFi', 'Amenity_Wi-Fi']
sub.drop(columns=cols, inplace=True)
```

[129]:
```
# Amenity vs. Avg effective/SF (when amenity is present vs. not present)
cols = ['Amenity_Roof Terrace', 'Amenity_Maid Service', 'Amenity_Bicycle␣
 ↪Storage',
       'Amenity_Car Charging Station', 'Amenity_On-Site Retail',␣
 ↪'Amenity_Elevator',
       'Amenity_Fitness Center', 'Amenity_Clubhouse', 'Amenity_Property Manager␣
 ↪on Site',
       'Amenity_Grill', 'Amenity_Playground', 'Amenity_Maintenance on site',␣
 ↪'Amenity_Gated',
       'Amenity_Picnic Area', 'Amenity_Package Service', 'Amenity_Controlled␣
 ↪Access',
       'Amenity_Sundeck', 'Amenity_LEED/Energy Star', 'Amenity_Sports',␣
 ↪'Amenity_Business',
       'Amenity_Laundry', 'Amenity_Spa/Sauna', 'Amenity_Pet', 'Amenity_Wifi']
amenity_summary = pd.DataFrame(sub[cols].sum()).rename(columns={0:'#␣
 ↪Properties'})
amenity_summary['% Properties'] = amenity_summary['# Properties']/sub.
 ↪shape[0]*100
amenity_summary['% Properties'] = amenity_summary['% Properties'].apply(lambda␣
 ↪x: round(x,1))
for amenity in cols:
    amenity_summary.loc[amenity, 'Avg Effective/SF w/ Amenity'] = \
    sub[sub[amenity]==1]['Avg Effective/SF'].mean()
```

```
    amenity_summary.loc[amenity, 'Avg Effective/SF w/o Amenity'] = \
    sub[sub[amenity]==0]['Avg Effective/SF'].mean()
amenity_summary['Avg Effective/SF Diff'] = amenity_summary['Avg Effective/SF w/␣
↪Amenity']-\
                                            amenity_summary['Avg Effective/SF w/
↪o Amenity']
amenity_summary = amenity_summary.sort_values('Avg Effective/SF Diff',␣
↪ascending=False)
amenity_summary
```

[129]:

```
                                   # Properties  % Properties  \
Amenity_Roof Terrace                        122           0.9
Amenity_Maid Service                        172           1.3
Amenity_Bicycle Storage                     243           1.9
Amenity_Car Charging Station                172           1.3
Amenity_On-Site Retail                      277           2.1
Amenity_Elevator                           1270           9.8
Amenity_LEED/Energy Star                    134           1.0
Amenity_Fitness Center                     9157          71.0
Amenity_Wifi                               1788          13.9
Amenity_Controlled Access                  3318          25.7
Amenity_Pet                                2003          15.5
Amenity_Spa/Sauna                          2421          18.8
Amenity_Business                           6724          52.1
Amenity_Grill                              5304          41.1
Amenity_Sundeck                            2875          22.3
Amenity_Package Service                    3655          28.3
Amenity_Clubhouse                          7553          58.5
Amenity_Gated                              4226          32.8
Amenity_Maintenance on site                4416          34.2
Amenity_Property Manager on Site           6836          53.0
Amenity_Sports                             4257          33.0
Amenity_Picnic Area                        4145          32.1
Amenity_Laundry                            8419          65.2
Amenity_Playground                         5117          39.7

                                   Avg Effective/SF w/ Amenity  \
Amenity_Roof Terrace                                  2.048852
Amenity_Maid Service                                  1.946047
Amenity_Bicycle Storage                               1.825021
Amenity_Car Charging Station                          1.804826
Amenity_On-Site Retail                                1.734549
Amenity_Elevator                                      1.692252
Amenity_LEED/Energy Star                              1.733433
Amenity_Fitness Center                                1.309690
Amenity_Wifi                                          1.373674
Amenity_Controlled Access                             1.328327
```

Amenity_Pet	1.336051
Amenity_Spa/Sauna	1.322701
Amenity_Business	1.290338
Amenity_Grill	1.279887
Amenity_Sundeck	1.270518
Amenity_Package Service	1.262971
Amenity_Clubhouse	1.243600
Amenity_Gated	1.238990
Amenity_Maintenance on site	1.239158
Amenity_Property Manager on Site	1.222594
Amenity_Sports	1.186281
Amenity_Picnic Area	1.176531
Amenity_Laundry	1.187084
Amenity_Playground	1.088251

	Avg Effective/SF w/o Amenity \
Amenity_Roof Terrace	1.239358
Amenity_Maid Service	1.237567
Amenity_Bicycle Storage	1.235917
Amenity_Car Charging Station	1.239475
Amenity_On-Site Retail	1.236316
Amenity_Elevator	1.198404
Amenity_LEED/Energy Star	1.241907
Amenity_Fitness Center	1.093796
Amenity_Wifi	1.226636
Amenity_Controlled Access	1.218863
Amenity_Pet	1.230650
Amenity_Spa/Sauna	1.229530
Amenity_Business	1.199864
Amenity_Grill	1.224065
Amenity_Sundeck	1.240272
Amenity_Package Service	1.240704
Amenity_Clubhouse	1.251828
Amenity_Gated	1.250919
Amenity_Maintenance on site	1.251098
Amenity_Property Manager on Site	1.274524
Amenity_Sports	1.276913
Amenity_Picnic Area	1.280369
Amenity_Laundry	1.359529
Amenity_Playground	1.351350

	Avg Effective/SF Diff
Amenity_Roof Terrace	0.809495
Amenity_Maid Service	0.708479
Amenity_Bicycle Storage	0.589104
Amenity_Car Charging Station	0.565350
Amenity_On-Site Retail	0.498233

```
Amenity_Elevator                              0.493848
Amenity_LEED/Energy Star                      0.491526
Amenity_Fitness Center                        0.215894
Amenity_Wifi                                  0.147038
Amenity_Controlled Access                     0.109464
Amenity_Pet                                   0.105401
Amenity_Spa/Sauna                             0.093172
Amenity_Business                              0.090474
Amenity_Grill                                 0.055822
Amenity_Sundeck                               0.030246
Amenity_Package Service                       0.022267
Amenity_Clubhouse                            -0.008228
Amenity_Gated                                -0.011929
Amenity_Maintenance on site                  -0.011941
Amenity_Property Manager on Site             -0.051931
Amenity_Sports                               -0.090632
Amenity_Picnic Area                          -0.103838
Amenity_Laundry                              -0.172445
Amenity_Playground                           -0.263099
```

3 4. Linear Regression

[130]:
```
# Final list of variables considered for regression model (Base group excluded␣
↪for analsysi)
cols = ['PropertyID', 'Avg Effective/SF',
       # Property
       'Avg Concessions %', 'Vacancy %', 'Avg Unit SF', 'Year Built', 'Year␣
↪Renovated', 'Star Rating',
       'Number Of Units', 'RBA', 'Floor Area Ratio', 'Land Area (AC)', 'Number␣
↪Of Stories',
       '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
       'Construction Material_Masonry', 'Construction Material_Reinforced␣
↪Concrete',
       'Construction Material_Steel or Metal',
       'Owner Type_large', 'Owner Type_medium',
       'Affordable Type_Rent Restricted', 'Affordable Type_Rent Subsidized',␣
↪'Affordable Type_Affordable Units',
       # Location & Demographic
       'Closest Transit Stop Dist (mi)', '2019 Avg Age(1m)', '2019 Pop Tot',
       'MedanHHIncome(000)', 'married %', 'male/female', 'Deposit (000s) Per␣
↪Capita',
       'Supply_all', 'Supply_vacant',
       'State_GA', 'State_FL', 'State_NC',
       'City_Atlanta', 'City_Dallas', 'City_Tampa', 'City_Orlando',␣
↪'City_Miami',
```

```
        'City_Jacksonville', 'City_Tallahassee', 'City_Charlotte', 'City_Durham',
        'City_Greensboro', 'City_Raleigh', 'City_Fort Worth', 'City_San Antonio',
        'City_Austin', 'City_Houston', 'City_Arlington', 'City_El Paso',␣
↪'City_Irving', 'City_Plano',
        # Amenities
        'Amenity_Roof Terrace', 'Amenity_Maid Service', 'Amenity_Bicycle␣
↪Storage',
        'Amenity_Car Charging Station', 'Amenity_On-Site Retail',␣
↪'Amenity_Elevator',
        'Amenity_Fitness Center', 'Amenity_Clubhouse', 'Amenity_Property Manager␣
↪on Site',
        'Amenity_Grill', 'Amenity_Playground', 'Amenity_Maintenance on site',␣
↪'Amenity_Gated',
        'Amenity_Picnic Area', 'Amenity_Package Service', 'Amenity_Controlled␣
↪Access',
        'Amenity_Sundeck', 'Amenity_LEED/Energy Star', 'Amenity_Sports',␣
↪'Amenity_Business',
        'Amenity_Laundry', 'Amenity_Spa/Sauna', 'Amenity_Pet', 'Amenity_Wifi']
final_onehot = sub[cols].copy()
final_onehot = final_onehot.fillna(0)
final_onehot.shape
```

[130]: (12903, 80)

3.0.1 4.1 Export for JMP

[131]:
```
final_onehot.to_csv('final_onehot_1125.csv', index=False)
```

3.0.2 4.2 Statsmodels

- The result matches JMP result
- Further variable and outlier exclusions are done in JMP

[132]:
```
test = final_onehot.copy()
```

[133]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor as␣
↪vif
from statsmodels.tools.tools import add_constant
from statsmodels.stats.outliers_influence import OLSInfluence as infl
```

Check VIF to ensure no severe multicolinearity.

[134]:
```
Xc = add_constant(test.iloc[:,2:])
vifs = [vif(Xc.values, i) for i in range(len(Xc.columns))]
pd.Series(data=vifs, index=Xc.columns).sort_values(ascending=False)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/core/fromnumeric.py:2223: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[134]:

```
const                                   47551.034684
% 1-Bed                                     9.412069
% 2-Bed                                     7.005639
Supply_vacant                               6.956090
Supply_all                                  6.868208
% 3-Bed                                     5.478698
RBA                                         4.331890
Number Of Units                             4.128782
Year Built                                  3.853151
married %                                   3.670934
Deposit (000s) Per Capita                   3.536555
State_FL                                    3.492203
City_Charlotte                              3.205380
MedanHHIncome(000)                          3.032181
Avg Unit SF                                 2.978755
% 4-Bed                                     2.934404
Star Rating                                 2.861852
State_NC                                    2.853037
Year Renovated                              2.554101
State_GA                                    2.329580
2019 Pop Tot                                2.151365
City_Houston                                1.950099
Amenity_Maintenance on site                 1.928520
City_Dallas                                 1.849001
Number Of Stories                           1.825543
City_Atlanta                                1.803963
Amenity_Fitness Center                      1.790801
Amenity_Package Service                     1.765254
2019 Avg Age(1m)                            1.663167
Amenity_Grill                               1.597699
                                              …
Construction Material_Steel or Metal        1.282758
City_Orlando                                1.276317
Amenity_Laundry                             1.231115
male/female                                 1.222513
City_Irving                                 1.218052
Vacancy %                                   1.203675
Amenity_Bicycle Storage                     1.199086
City_Tampa                                  1.197687
City_Greensboro                             1.195640
City_Durham                                 1.175744
City_Fort Worth                             1.172993
```

```
Amenity_Spa/Sauna                            1.168588
City_Tallahassee                             1.167728
Land Area (AC)                               1.167444
Amenity_Pet                                  1.161951
Amenity_Wifi                                 1.144294
Affordable Type_Rent Subsidized              1.142639
Owner Type_large                             1.139981
Avg Concessions %                            1.139287
Amenity_Roof Terrace                         1.136265
Owner Type_medium                            1.127687
Amenity_Car Charging Station                 1.115983
Amenity_On-Site Retail                       1.097522
City_El Paso                                 1.087768
City_Arlington                               1.086626
City_Plano                                   1.063534
Amenity_LEED/Energy Star                     1.035300
Amenity_Maid Service                         1.030196
Affordable Type_Affordable Units             1.029274
Floor Area Ratio                             1.007222
Length: 79, dtype: float64
```

[135]:
```
model = sm.OLS(test['Avg Effective/SF'], Xc)
```

[136]:
```
result = model.fit()
```

[137]:
```
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Avg Effective/SF   R-squared:                       0.494
Model:                            OLS   Adj. R-squared:                  0.491
Method:                 Least Squares   F-statistic:                     160.6
Date:                Sun, 01 Dec 2019   Prob (F-statistic):               0.00
Time:                        23:32:59   Log-Likelihood:                -4033.3
No. Observations:               12903   AIC:                             8225.
Df Residuals:                   12824   BIC:                             8814.
Df Model:                          78
Covariance Type:            nonrobust
================================================================================
=============================
                                                coef    std err          t
P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
-----------------------------
const                                        -8.5018      0.637    -13.348
0.000      -9.750      -7.253
Avg Concessions %                            -0.0121      0.002     -7.952
```

				0.000	-0.015	-0.009
Vacancy %	0.0036	0.000	8.652	0.000	0.003	0.004
Avg Unit SF	-0.0009	2.86e-05	-31.420	0.000	-0.001	-0.001
Year Built	0.0051	0.000	14.626	0.000	0.004	0.006
Year Renovated	7.997e-05	0.000	0.278	0.781	-0.000	0.001
Star Rating	0.0543	0.007	8.028	0.000	0.041	0.068
Number Of Units	-0.0002	4.55e-05	-5.072	0.000	-0.000	-0.000
RBA	1.953e-07	3.83e-08	5.096	0.000	1.2e-07	2.7e-07
Floor Area Ratio	-4.608e-05	9.18e-05	-0.502	0.616	-0.000	0.000
Land Area (AC)	-0.0002	0.000	-1.404	0.160	-0.000	7.13e-05
Number Of Stories	0.0208	0.001	15.969	0.000	0.018	0.023
% 1-Bed	-0.0042	0.000	-10.426	0.000	-0.005	-0.003
% 2-Bed	-0.0037	0.000	-8.800	0.000	-0.005	-0.003
% 3-Bed	-0.0025	0.001	-4.993	0.000	-0.003	-0.002
% 4-Bed	0.0046	0.001	8.069	0.000	0.003	0.006
Construction Material_Masonry	0.0100	0.007	1.445	0.148	-0.004	0.024
Construction Material_Reinforced Concrete	0.0922	0.013	7.371	0.000	0.068	0.117
Construction Material_Steel or Metal	0.0668	0.027	2.457	0.014	0.013	0.120
Owner Type_large	0.0187	0.012	1.558	0.119	-0.005	0.042
Owner Type_medium	-0.0073	0.007	-1.124	0.261	-0.020	0.005
Affordable Type_Rent Restricted	-0.3031	0.011	-27.884	0.000	-0.324	-0.282
Affordable Type_Rent Subsidized	-0.2353	0.017	-13.944	0.000	-0.268	-0.202
Affordable Type_Affordable Units	-0.1893	0.033	-5.686	0.000	-0.255	-0.124
Closest Transit Stop Dist (mi)	-0.0008	0.000	-2.274	0.023	-0.002	-0.000
2019 Avg Age(1m)	0.0088	0.001	9.493			

				0.000	0.007	0.011
2019 Pop Tot	5.071e-06	4.91e-07	10.324	0.000	4.11e-06	6.03e-06
MedanHHIncome(000)	0.0061	0.000	25.038	0.000	0.006	0.007
married %	-0.0073	0.001	-14.134	0.000	-0.008	-0.006
male/female	0.0279	0.017	1.659	0.097	-0.005	0.061
Deposit (000s) Per Capita	0.0005	0.000	3.138	0.002	0.000	0.001
Supply_all	9.681e-06	2.03e-06	4.769	0.000	5.7e-06	1.37e-05
Supply_vacant	-0.0001	2.39e-05	-6.076	0.000	-0.000	-9.82e-05
State_GA	0.0520	0.013	4.105	0.000	0.027	0.077
State_FL	0.1679	0.013	12.983	0.000	0.143	0.193
State_NC	-0.0280	0.015	-1.881	0.060	-0.057	0.001
City_Atlanta	0.0085	0.020	0.428	0.668	-0.030	0.047
City_Dallas	0.0130	0.017	0.775	0.438	-0.020	0.046
City_Tampa	-0.1506	0.022	-6.818	0.000	-0.194	-0.107
City_Orlando	-0.0935	0.020	-4.576	0.000	-0.134	-0.053
City_Miami	-0.0084	0.027	-0.307	0.759	-0.062	0.045
City_Jacksonville	-0.1890	0.022	-8.452	0.000	-0.233	-0.145
City_Tallahassee	-0.1876	0.035	-5.364	0.000	-0.256	-0.119
City_Charlotte	-0.0844	0.031	-2.717	0.007	-0.145	-0.023
City_Durham	0.0652	0.033	1.977	0.048	0.001	0.130
City_Greensboro	-0.1302	0.032	-4.134	0.000	-0.192	-0.068
City_Raleigh	0.0013	0.025	0.052	0.959	-0.048	0.051
City_Fort Worth	0.0326	0.023	1.416	0.157	-0.013	0.078
City_San Antonio	-0.0827	0.016	-5.073	0.000	-0.115	-0.051
City_Austin	0.1621	0.017	9.343			

```
0.000       0.128       0.196
City_Houston                                  -0.0176      0.013     -1.328
0.184      -0.044       0.008
City_Arlington                                 0.0955      0.028      3.384
0.001       0.040       0.151
City_El Paso                                  -0.1628      0.031     -5.235
0.000      -0.224      -0.102
City_Irving                                    0.0797      0.030      2.686
0.007       0.022       0.138
City_Plano                                     0.1308      0.034      3.883
0.000       0.065       0.197
Amenity_Roof Terrace                           0.1208      0.032      3.754
0.000       0.058       0.184
Amenity_Maid Service                           0.5319      0.026     20.575
0.000       0.481       0.583
Amenity_Bicycle Storage                        0.0763      0.024      3.243
0.001       0.030       0.122
Amenity_Car Charging Station                   0.0823      0.027      3.058
0.002       0.030       0.135
Amenity_On-Site Retail                         0.0962      0.021      4.558
0.000       0.055       0.138
Amenity_Elevator                               0.0534      0.012      4.504
0.000       0.030       0.077
Amenity_Fitness Center                         0.0562      0.009      6.527
0.000       0.039       0.073
Amenity_Clubhouse                             -0.0264      0.007     -3.888
0.000      -0.040      -0.013
Amenity_Property Manager on Site              -0.0029      0.007     -0.406
0.685      -0.017       0.011
Amenity_Grill                                  0.0046      0.008      0.617
0.537      -0.010       0.019
Amenity_Playground                            -0.0347      0.007     -4.893
0.000      -0.049      -0.021
Amenity_Maintenance on site                   -0.0218      0.009     -2.556
0.011      -0.039      -0.005
Amenity_Gated                                 -0.0196      0.007     -2.770
0.006      -0.034      -0.006
Amenity_Picnic Area                           -0.0296      0.008     -3.792
0.000      -0.045      -0.014
Amenity_Package Service                       -0.0084      0.009     -0.975
0.329      -0.025       0.008
Amenity_Controlled Access                     -0.0011      0.008     -0.132
0.895      -0.017       0.015
Amenity_Sundeck                               -0.0075      0.009     -0.845
0.398      -0.025       0.010
Amenity_LEED/Energy Star                       0.1282      0.029      4.374
0.000       0.071       0.186
Amenity_Sports                                -0.0093      0.007     -1.320
```

```
0.187      -0.023      0.005
Amenity_Business                          0.0014      0.007      0.206
0.837      -0.012      0.015
Amenity_Laundry                          -0.0236      0.007     -3.466
0.001      -0.037     -0.010
Amenity_Spa/Sauna                        -0.0021      0.008     -0.255
0.798      -0.018      0.014
Amenity_Pet                              -0.0109      0.009     -1.259
0.208      -0.028      0.006
Amenity_Wifi                              0.0149      0.009      1.649
0.099      -0.003      0.033
==============================================================================
Omnibus:                    18700.767   Durbin-Watson:                   1.919
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         18243745.463
Skew:                           8.309   Prob(JB):                         0.00
Kurtosis:                     186.461   Cond. No.                     6.78e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 6.78e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

4 5. Random Forest Regressor with Scikit-Learn

4.0.1 5.1 Exclude outliers identified from the regression model

[138]:
```python
jmp = pd.read_csv('final_onehot_1125_outlier_est_CI.dat')
```

[139]:
```python
jmp.shape
```

[139]: (12903, 85)

[140]:
```python
outlier = jmp[jmp['Outlier']==1]['PropertyID'].tolist()
len(outlier)
```

[140]: 507

[141]:
```python
cols = ['PropertyID', 'Avg Effective/SF',
       # Property
       'Avg Concessions %', 'Vacancy %', 'Avg Unit SF', 'Year Built', 'Year␣
 ↪Renovated', 'Star Rating',
       'Number Of Units', 'RBA', 'Floor Area Ratio', 'Land Area (AC)', 'Number␣
 ↪Of Stories',
       '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
       'Construction Material*', 'Owner Type', 'Affordable Type*',
```

```
        # Location & Demographic
        'Closest Transit Stop Dist (mi)', '2019 Avg Age(1m)', '2019 Pop Tot',
        'MedanHHIncome(000)', 'married %', 'male/female', 'Deposit (000s) Per␣
↪Capita',
        'Supply_all', 'Supply_vacant',
        'State', 'City', 'County Name',
        # Amenities
        'Amenity_Roof Terrace', 'Amenity_Maid Service', 'Amenity_Bicycle␣
↪Storage',
        'Amenity_Car Charging Station', 'Amenity_On-Site Retail',␣
↪'Amenity_Elevator',
        'Amenity_Fitness Center', 'Amenity_Clubhouse', 'Amenity_Property Manager␣
↪on Site',
        'Amenity_Grill', 'Amenity_Playground', 'Amenity_Maintenance on site',␣
↪'Amenity_Gated',
        'Amenity_Picnic Area', 'Amenity_Package Service', 'Amenity_Controlled␣
↪Access',
        'Amenity_Sundeck', 'Amenity_LEED/Energy Star', 'Amenity_Sports',␣
↪'Amenity_Business',
        'Amenity_Laundry', 'Amenity_Spa/Sauna', 'Amenity_Pet', 'Amenity_Wifi']
sub1 = sub[-sub['PropertyID'].isin(outlier)][cols].copy()
sub1 = sub1.fillna(0)
sub1.shape
```

[141]:
```
(12396, 56)
```

4.0.2 5.2 Regroup City, County and Affordable Type

- Make sure test subset will have enough observations in each category

[142]:
```
city = sub1['City'].value_counts().reset_index()
city.columns = ['City', 'Count']
large_city = city[city['Count']>200]['City'].unique()
sub1['City**'] = sub1[['City','State']].apply(lambda x: x[0] if x[0] in␣
↪large_city\
                                               else '{} Other'.format(x[1]), axis=1)
print('# City:', sub1['City**'].nunique())
print('Smallest city bin:', sub1['City**'].value_counts().tail(1))
```

```
# City: 15
Smallest city bin: Raleigh    229
Name: City**, dtype: int64
```

[143]:
```
county = sub1['County Name'].value_counts().reset_index()
county.columns = ['County', 'Count']
large_county = county[county['Count']>200]['County'].unique()
```

```
sub1['County**'] = sub1[['County Name','State']].apply(lambda x: x[0] if x[0]␣
↪in large_county\
                                                    else '{} Other'.format(x[1]), axis=1)
print('# County:', sub1['County**'].nunique())
print('Smallest county bin:', sub1['County**'].value_counts().tail(1))
```

```
# County: 22
Smallest county bin: Palm Beach    201
Name: County**, dtype: int64
```

[144]:
```
sub1['Construction Material*'].value_counts()
```

[144]:
```
Wood Frame              6977
Masonry                 4334
Reinforced Concrete      952
Steel or Metal           133
Name: Construction Material*, dtype: int64
```

Although Steel or Metal only have 134 observations, it's hard to reason combining it with any other categories, so will leave it as it is.

[145]:
```
sub1['Owner Type'].value_counts()
```

[145]:
```
small     7184
medium    4296
large      916
Name: Owner Type, dtype: int64
```

[146]:
```
sub1['Affordable Type*'].value_counts()
```

[146]:
```
Market              10525
Rent Restricted      1364
Rent Subsidized       418
Affordable Units       89
Name: Affordable Type*, dtype: int64
```

[147]:
```
sub1['Affordable Type**'] = sub1['Affordable Type*'].map({'Rent Restricted':
↪'Rent Restricted',
                                                          'Market':'Market',
                                                          'Rent Subsidized':
↪'Rent Subsidized/Affordable Units',
                                                          'Affordable Units':
↪'Rent Subsidized/Affordable Units'})
```

[148]:
```
sub1['Affordable Type**'].value_counts()
```

[148]: Market   10525
Rent Restricted   1364
Rent Subsidized/Affordable Units   507
Name: Affordable Type**, dtype: int64

[149]:
```
cols = ['PropertyID', 'Avg Effective/SF',
       # Property
       'Avg Concessions %', 'Vacancy %', 'Avg Unit SF', 'Year Built', 'Year␣
↪Renovated', 'Star Rating',
       'Number Of Units', 'RBA', 'Floor Area Ratio', 'Land Area (AC)', 'Number␣
↪Of Stories',
       '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
       'Construction Material*', 'Owner Type', 'Affordable Type**',
       # Location & Demographic
       'Closest Transit Stop Dist (mi)', '2019 Avg Age(1m)', '2019 Pop Tot',
       'MedanHHIncome(000)', 'married %', 'male/female', 'Deposit (000s) Per␣
↪Capita',
       'Supply_all', 'Supply_vacant',
       'State', 'City**', 'County**',
       # Amenities
       'Amenity_Roof Terrace', 'Amenity_Maid Service', 'Amenity_Bicycle␣
↪Storage',
       'Amenity_Car Charging Station', 'Amenity_On-Site Retail',␣
↪'Amenity_Elevator',
       'Amenity_Fitness Center', 'Amenity_Clubhouse', 'Amenity_Property Manager␣
↪on Site',
       'Amenity_Grill', 'Amenity_Playground', 'Amenity_Maintenance on site',␣
↪'Amenity_Gated',
       'Amenity_Picnic Area', 'Amenity_Package Service', 'Amenity_Controlled␣
↪Access',
       'Amenity_Sundeck', 'Amenity_LEED/Energy Star', 'Amenity_Sports',␣
↪'Amenity_Business',
       'Amenity_Laundry', 'Amenity_Spa/Sauna', 'Amenity_Pet', 'Amenity_Wifi']
var = sub1[cols].copy()
```

4.0.3 5.3 Separate train and test sets

- 0.7 train and 0.3 test to make sure test set have enough observations in each category

[150]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(var, test_size=0.3, random_state=0)
train = train.copy()
test = test.copy()
```

[151]:
```
print(train.shape)
print(test.shape)
```

```
(8677, 56)
(3719, 56)
```

4.0.4 5.4 Target encode six categorical features

[152]:
```
# 'Construction Material*', 'Owner Type', 'Affordable Type**', 'State',␣
↪'City**', 'County**'
```

[153]:
```
train['Construction Material_encoded'] = train[['Construction Material*','Avg␣
↪Effective/SF']].\
                                          groupby('Construction Material*').
↪transform(lambda x: x.mean())
test['Construction Material_encoded'] = test[['Construction Material*','Avg␣
↪Effective/SF']].\
                                         groupby('Construction Material*').
↪transform(lambda x: x.mean())

train['Owner Type_encoded'] = train[['Owner Type','Avg Effective/SF']].\
                              groupby('Owner Type').transform(lambda x: x.
↪mean())
test['Owner Type_encoded'] = test[['Owner Type','Avg Effective/SF']].\
                              groupby('Owner Type').transform(lambda x: x.
↪mean())

train['Affordable Type_encoded'] = train[['Affordable Type**','Avg Effective/
↪SF']].\
                              groupby('Affordable Type**').transform(lambda x:␣
↪x.mean())
test['Affordable Type_encoded'] = test[['Affordable Type**','Avg Effective/
↪SF']].\
                              groupby('Affordable Type**').transform(lambda x:␣
↪x.mean())

train['State_encoded'] = train[['State','Avg Effective/SF']].\
                              groupby('State').transform(lambda x: x.mean())
test['State_encoded'] = test[['State','Avg Effective/SF']].\
                              groupby('State').transform(lambda x: x.mean())

train['City_encoded'] = train[['City**','Avg Effective/SF']].\
                              groupby('City**').transform(lambda x: x.mean())
test['City_encoded'] = test[['City**','Avg Effective/SF']].\
                              groupby('City**').transform(lambda x: x.mean())

train['County_encoded'] = train[['County**','Avg Effective/SF']].\
                              groupby('County**').transform(lambda x: x.mean())
test['County_encoded'] = test[['County**','Avg Effective/SF']].\
                              groupby('County**').transform(lambda x: x.mean())
```

[154]:
```
cols = ['Construction Material*', 'Owner Type', 'Affordable Type**', 'State',␣
↪'City**', 'County**']
train = train.drop(columns=cols)
test = test.drop(columns=cols)
```

[155]:
```
X_train = train.iloc[:, 2:].values
y_train = train.iloc[:, 1].values
X_test = test.iloc[:, 2:].values
y_test = test.iloc[:, 1].values
```

4.0.5 5.5 Model fitting with 54 features

[156]:
```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=200, random_state=0)
regressor.fit(X_train, y_train)
y_pred_train = regressor.predict(X_train)
y_pred_test = regressor.predict(X_test)
```

[157]:
```
from sklearn import metrics

print('------------------------------TRAIN------------------------------')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_train,␣
↪y_pred_train))
print('Mean Squared Error:', metrics.mean_squared_error(y_train, y_pred_train))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train,␣
↪y_pred_train)))
print('R Squared:', metrics.r2_score(y_train, y_pred_train))
print('')
print('------------------------------TEST------------------------------')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_test))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_test))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪y_pred_test)))
print('R Squared:', metrics.r2_score(y_test, y_pred_test))
```

```
------------------------------TRAIN------------------------------
Mean Absolute Error: 0.04466652068687335
Mean Squared Error: 0.0038002139426O689
Root Mean Squared Error: 0.06164587530895226
R Squared: 0.9656867174194602

------------------------------TEST------------------------------
Mean Absolute Error: 0.12429448776552837
Mean Squared Error: 0.02851194776418393
Root Mean Squared Error: 0.16885481267699753
R Squared: 0.728666816991679
```

High Overfitting. Requires future work.

5 6. Identify Underpriced Properties

[158]:
```
pred = jmp[['PropertyID', 'Avg Effective/SF', 'Pred Formula Avg Effective/SF',
            'Lower 95% Indiv Avg Effective/SF', 'Upper 95% Indiv Avg Effective/
 ↪SF', 'Outlier']].copy()
pred.head()
```

[158]:
```
   PropertyID  Avg Effective/SF  Pred Formula Avg Effective/SF  \
0     6400737              1.46                       1.265123
1     6897678              2.11                       1.792363
2     6865125              2.11                       1.845737
3     6900122              1.03                       1.813714
4     6900121              1.63                       1.763190

   Lower 95% Indiv Avg Effective/SF  Upper 95% Indiv Avg Effective/SF  Outlier
0                          0.917724                          1.612521        0
1                          1.444436                          2.140290        0
2                          1.497319                          2.194155        0
3                          1.463738                          2.163691        1
4                          1.414693                          2.111687        0
```

[159]:
```
pred['Overpriced'] = pred[['Avg Effective/SF', 'Upper 95% Indiv Avg Effective/
 ↪SF']].\
                     apply(lambda x: 1 if x[0]>x[1] else 0, axis=1)
pred['Underpriced'] = pred[['Avg Effective/SF', 'Lower 95% Indiv Avg Effective/
 ↪SF']].\
                     apply(lambda x: 1 if x[0]<x[1] else 0, axis=1)
```

[160]:
```
pred[['Overpriced', 'Underpriced']].sum()
```

[160]:
```
Overpriced     689
Underpriced    425
dtype: int64
```

[161]:
```
pred[pred['Outlier']==0][['Overpriced', 'Underpriced']].sum()
```

[161]:
```
Overpriced     458
Underpriced    239
dtype: int64
```

[162]:
```
underpriced = pred[(pred['Outlier']==0) & (pred['Underpriced']==1)]\
              [['PropertyID', 'Pred Formula Avg Effective/SF']]
underpriced = underpriced.merge(sub[['PropertyID', 'State', 'Latitude',␣
 ↪'Longitude', 'Avg Effective/SF']])
```

```
underpriced.head()
```

[162]:
```
   PropertyID  Pred Formula Avg Effective/SF State   Latitude  Longitude  \
0     9021275                        1.261372    GA  33.688340 -84.505842
1     8325011                        1.167366    GA  33.718105 -84.370369
2     4719499                        1.118949    GA  33.667620 -84.498327
3     8077716                        1.289297    GA  33.734370 -84.428340
4     8436498                        1.173577    GA  33.737731 -84.403056

   Avg Effective/SF
0              0.68
1              0.70
2              0.74
3              0.92
4              0.81
```

[163]:
```
underpriced.to_csv('underpriced.csv',index=False)
```

DSO 545 Project External Data

April 4, 2021

[1]:
```
from bs4 import BeautifulSoup
import urllib.request
import requests
import pandas as pd
import numpy as np
```

0.0.1 1. Import Zip Codes

[2]:
```
zip5 = pd.read_csv('Zip5.csv')
zip5.columns=['Zip5']
```

[3]:
```
zip5.shape
```

[3]:
```
(1954, 1)
```

0.0.2 2. Scrape Median HH Income

sample URL: https://statisticalatlas.com/zip/30097/Household-Income

[4]:
```
df = pd.DataFrame()

for zipcode in zip5.Zip5:
    url = "https://statisticalatlas.com/zip/{}/Household-Income".format(zipcode)
    with requests.get(url) as r:
        soup = BeautifulSoup(r.text, 'lxml')
        table = soup.find_all('text', {"fill-opacity":"0.400"})[3:4]
        values = [zipcode]
        values.extend([row.text for row in table])
        df = df.append(pd.DataFrame(values).T, ignore_index=True)

cols = ['Zip5', 'MedanHHIncome']
df.columns=cols
df.head()
```

[4]:
```
    Zip5 MedanHHIncome
0  30097        $96.9k
1  30318        $44.0k
```

```
2  30309         $78.3k
3  30363         $66.9k
4  30328         $80.8k
```

[5]:
```
df.count()
```

[5]:
```
Zip5             1954
MedanHHIncome    1913
dtype: int64
```

- Drop rows where zip code is not found on the website
- Remove dollar sign, 'k', and '>' in >250k
- Remove wrong entries with '%'
- Change column data type to float and rename

[6]:
```
df1 = df.copy()
df1 = df1.dropna()
df1['MedanHHIncome'] = df1['MedanHHIncome'].str.replace('$','')
df1['MedanHHIncome'] = df1['MedanHHIncome'].str.replace('k','')
df1['MedanHHIncome'] = df1['MedanHHIncome'].str.replace('>','')
df1 = df1[-df1['MedanHHIncome'].str.contains('%')]
df1['MedanHHIncome'] = df1['MedanHHIncome'].astype(float)
df1.columns = ['Zip5', 'MedanHHIncome(000)']
```

0.0.3 3. Scrape Marital Status Info

sample URL: https://statisticalatlas.com/zip/30097/Marital-Status

[7]:
```
dfm = pd.DataFrame()

for zipcode in zip5.Zip5:
    url = "https://statisticalatlas.com/zip/{}/Marital-Status".format(zipcode)
    with requests.get(url) as r:
        soup = BeautifulSoup(r.text, 'lxml')
        table = soup.find_all('text', {"fill-opacity":"0.500"})[:8]
        values = [zipcode]
        values.extend([row.text for row in table])
        dfm = dfm.append(pd.DataFrame(values).T, ignore_index=True)

cols = ['Zip5', 'Never_Married_F', 'Never_Married_M', 'Married_F', 'Married_M',
        'Separated/Divorced_F', 'Separated/Divorced_M', 'Widowed_F',␣
 ↪'Widowed_M']
dfm.columns=cols
dfm.head()
```

[7]:
```
    Zip5 Never_Married_F Never_Married_M Married_F Married_M  \
0  30097           4,507           4,384     11.9k     12.2k
```

```
1  30318          9,604          14.2k     4,882     5,393
2  30309          5,031          6,226     3,302     3,595
3  30363            798          1,097       405       433
4  30328          5,246          4,440     7,420     7,879

  Separated/Divorced_F Separated/Divorced_M Widowed_F Widowed_M
0                1,578                  909       844       177
1                2,518                1,978     2,190     1,288
2                1,358                1,163       301       150
3                  126                   68         0         0
4                2,526                1,092     1,238       243
```

- Drop rows where zip code is not found on the website
- Change format: 11.9k to 11900
- Change column data type to integer
- Generate married % and male/female variables

[8]:
```
dfm1 = dfm.copy()
dfm1 = dfm1.dropna()
for col in dfm.columns[1:]:
    dfm1[col] = dfm1[col].str.replace(',','')
    dfm1[col] = dfm1[col].apply(lambda x: round(float(x[:-1])*1000) if x[-1:
↪]=='k' else x).astype(int)
dfm1['male'] = dfm1[['Never_Married_M', 'Married_M', 'Separated/Divorced_M',␣
↪'Widowed_M']].sum(axis=1)
dfm1['female'] = dfm1[['Never_Married_F', 'Married_F', 'Separated/Divorced_F',␣
↪'Widowed_F']].sum(axis=1)
dfm1['population'] = dfm1[['male','female']].sum(axis=1)
dfm1['married'] = dfm1[['Married_F', 'Married_M']].sum(axis=1)
dfm1['married %'] = dfm1['married']/dfm1['population']*100
dfm1['married %'].replace(0, np.nan, inplace=True)
dfm1['male/female'] = dfm1['male']/dfm1['female']
```

0.0.4 4. Merge and Export Income and Marriage data

[9]:
```
dfmg = df1.merge(dfm1[['Zip5', 'married %', 'male/female']], how='outer')
dfmg.shape
```

[9]:
```
(1925, 4)
```

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

[10]:
```
               Zip5  MedanHHIncome(000)    married %  male/female
count   1925.000000         1910.000000  1922.000000  1925.000000
mean   50132.948052           54.527068    46.575208     0.995139
std    22473.225784           21.769816    11.448893     0.822831
min    27006.000000           13.100000     0.101877     0.394516
```

```
25%     31792.000000            40.000000    40.186142     0.887395
50%     33809.000000            49.700000    47.536718     0.935197
75%     77011.000000            64.650000    54.059453     0.991362
max     79938.000000           250.000000    90.712431    33.600000
```

[11]:
```
dfmg.to_csv('income_marriage.csv', index=False)
```

0.0.5 5. Deposit

[12]:
```
deposit = pd.read_excel('FDIC Deposit.xlsx')
```

[13]:
```
print(deposit.shape)
deposit.head()
```

```
(574, 3)
```

[13]:
```
  State    County  Deposit (000s)
0    GA    Fulton       100332784
1    GA  Gwinnett        17717075
2    GA      Cobb        15632932
3    GA    DeKalb        12481873
4    GA  Muscogee         8394232
```

[14]:
```
pop = pd.read_excel('Census Population.xlsx')
```

[15]:
```
print(pop.shape)
pop.head()
```

```
(580, 2)
```

[15]:
```
                 Geography  Population Estimate (as of July 1) - 2018
0  Anderson County, Texas                                       58057
1   Andrews County, Texas                                       18128
2  Angelina County, Texas                                       87092
3   Aransas County, Texas                                       23792
4    Archer County, Texas                                        8786
```

Extract county and state from Geography.

[16]:
```
pop['State'] = pop['Geography'].apply(lambda x: x.split(', ')[1])
pop['State'] = pop['State'].map({'Texas':'TX',
                                 'Georgia':'GA',
                                 'North Carolina':'NC',
                                 'Florida':'FL'})
pop['County'] = pop['Geography'].apply(lambda x: x.split(' County')[0])
pop.columns = ['Geography', 'Population Est 2018', 'State', 'County']
pop.head()
```

```
[16]:                   Geography  Population Est 2018 State    County
      0  Anderson County, Texas                58057    TX  Anderson
      1   Andrews County, Texas                18128    TX   Andrews
      2  Angelina County, Texas                87092    TX  Angelina
      3   Aransas County, Texas                23792    TX   Aransas
      4    Archer County, Texas                 8786    TX    Archer
```

```
[17]: mg = deposit.merge(pop[['State', 'County', 'Population Est 2018']],␣
       ↪on=['State','County'])
      print(mg.shape)
      print(mg.count())
      mg.head()
```

```
(572, 4)
State                  572
County                 572
Deposit (000s)         572
Population Est 2018    572
dtype: int64
```

```
[17]:   State    County  Deposit (000s)  Population Est 2018
      0    GA    Fulton       100332784              1050114
      1    GA  Gwinnett        17717075               927781
      2    GA      Cobb        15632932               756865
      3    GA    DeKalb        12481873               756558
      4    GA  Muscogee         8394232               194160
```

Calculate per capita saving.

```
[18]: mg['Deposit (000s) Per Capita'] = mg['Deposit (000s)']/mg['Population Est 2018']
      mg.head()
```

```
[18]:   State    County  Deposit (000s)  Population Est 2018  \
      0    GA    Fulton       100332784              1050114
      1    GA  Gwinnett        17717075               927781
      2    GA      Cobb        15632932               756865
      3    GA    DeKalb        12481873               756558
      4    GA  Muscogee         8394232               194160

         Deposit (000s) Per Capita
      0                  95.544659
      1                  19.096182
      2                  20.654849
      3                  16.498237
      4                  43.233581
```

```
[19]: mg.describe()
```

[19]:

	Deposit (000s)	Population Est 2018	Deposit (000s) Per Capita
count	5.720000e+02	5.720000e+02	572.000000
mean	3.674213e+06	1.238987e+05	18.570632
std	1.751788e+07	3.437498e+05	13.667547
min	7.602000e+03	7.260000e+02	0.928432
25%	1.679305e+05	1.195700e+04	10.966622
50%	4.214485e+05	2.728400e+04	15.334716
75%	1.248876e+06	8.595475e+04	21.664140
max	2.086604e+08	4.698619e+06	172.786415

[21]:

```
mg.to_csv('per_capita_deposit.csv', index=False)
```

[ ]:

```
```

DSO 545 Project Visualization

April 4, 2021

[1]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import squarify
```

[2]:
```
data = pd.read_excel('Property Data Compiled.xlsx', index_col = 0)
```

[3]:
```
data.shape
```

[3]:
```
(20363, 211)
```

[4]:
```
cols = ['PropertyID',
        # rent fields
        'Avg Effective/SF', 'Avg Concessions %',
        'Studio Effective Rent/SF', 'One Bedroom Effective Rent/SF', 'Two␣
↪Bedroom Effective Rent/SF',
        'Three Bedroom Effective Rent/SF', 'Four Bedroom Effective Rent/SF',
        # unit fields
        'Studio Avg SF', 'Number Of Studios', 'Studio Vacant Units', 'Studio␣
↪Vacancy %',
        'One Bedroom Avg SF','Number Of 1 Bedrooms', 'One Bedroom Vacant␣
↪Units', 'One Bedroom Vacancy %',
        'Two Bedroom Avg SF', 'Number Of 2 Bedrooms', 'Two Bedroom Vacant␣
↪Units', 'Two Bedroom Vacancy %',
        'Three Bedroom Avg SF', 'Number Of 3 Bedrooms', 'Three Bedroom Vacant␣
↪Units', 'Three Bedroom Vacancy %',
        'Four Bedroom Avg SF', 'Number Of 4 Bedrooms', 'Four Bedroom Vacant␣
↪Units', 'Four Bedroom Vacancy %',
        # location fields
        'State', 'Market Name', 'City', 'Zip', 'County Name',
        'Closest Transit Stop Dist (mi)', 'Latitude', 'Longitude',
        # property fields
        'Star Rating', 'Building Status', 'Land Area (AC)', 'Number Of Stories',
        'Style', 'Number Of Units', 'Vacancy %', 'Avg Unit SF', 'RBA',
        '% Studios', '% 1-Bed', '% 2-Bed', '% 3-Bed', '% 4-Bed',
```

```
        'Rent Type', 'Affordable Type', 'Market Segment',
        'Amenities', 'Building Class', 'Construction Material', 'Owner Name',␣
↪'Property Manager Name',
        'Year Built', 'Year Renovated',
        # demographic fields
        '2019 Avg Age(1m)', '2019 Pop Age <19(1m)', '2019 Pop Age␣
↪20-64(1m)','2019 Pop Age 65+(1m)']
sub = data.copy()[cols]
sub.drop_duplicates(subset='PropertyID', inplace = True)
sub['State'] = sub['State'].replace('Fl', 'FL').replace('NC ','NC')
sub['Zip5'] = sub['Zip'].str[:5]
sub.shape
```

[4]: (20300, 65)

[5]:
```
sub['Avg Effective/SF'].describe()
```

[5]:
```
count    16751.000000
mean         1.258938
std          0.555898
min          0.190000
25%          0.970000
50%          1.160000
75%          1.400000
max         14.030000
Name: Avg Effective/SF, dtype: float64
```

[6]:
```
print('99% quantile:',sub['Avg Effective/SF'].quantile(0.99))
```

```
99% quantile: 3.47
```

1 0 Import Income, Marriage and Deposit data

[7]:
```
demo = pd.read_csv('income_marriage.csv')
```

[8]:
```
demo['Zip5'] = demo['Zip5'].apply(round).astype(str)
demo.shape
```

[8]: (1925, 4)

[9]:
```
sub = sub.merge(demo, how='left', on='Zip5')
sub.shape
```

[9]: (20300, 68)

[10]:
```
deposit = pd.read_csv('per_capita_deposit.csv')
```

[11]:
```
deposit.rename(columns={'County':'County Name'}, inplace=True)
```

[12]:
```
deposit.shape
```

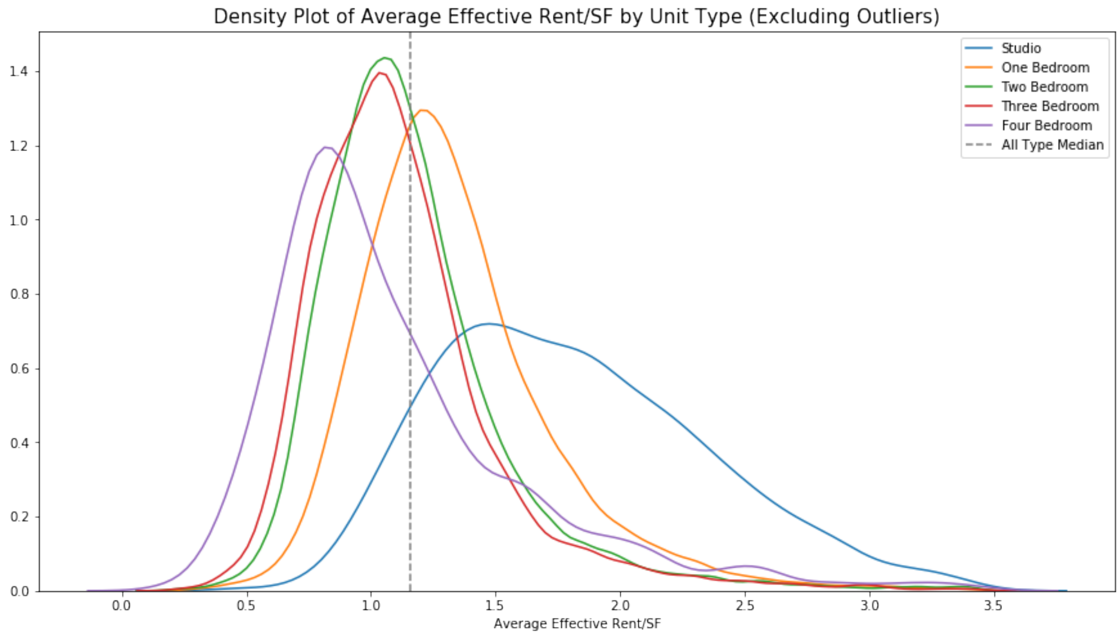
[12]: (572, 5)

[13]:
```
sub = sub.merge(deposit[['State', 'County Name', 'Deposit (000s) Per Capita']],
                on=['State', 'County Name'], how='left')
sub.shape
```

[13]: (20300, 69)

2 1 Rent by Unit Type

[14]:
```
unit_rent = ['Studio', 'One Bedroom', 'Two Bedroom',
             'Three Bedroom', 'Four Bedroom']

plt.figure(figsize=(15,8))
for unit in unit_rent:
    df = sub[sub['{} Effective Rent/SF'.format(unit)]<sub['Avg Effective/SF'].
 ↪quantile(0.99)]\
    ['{} Effective Rent/SF'.format(unit)]
    sns.distplot(df, hist=False, label=unit)
#dist.set_title('Density Plot of Average Effective Rent/SF by {} (Excluding 1%␣
 ↪Outliers)'.format(cat), fontsize=15)
#dist.set_xlabel('Average Effective Rent/SF')
#sns.distplot(sub[sub['Avg Effective/SF']<sub['Avg Effective/SF'].quantile(0.
 ↪99)]['Avg Effective/SF'],
#              hist=False, label='All Type', kde_kws=dict(linewidth=3,␣
 ↪color='black', linestyle='--'))
plt.axvline(sub['Avg Effective/SF'].median(), color='grey', linestyle='--',␣
 ↪label='All Type Median')
plt.xlabel('Average Effective Rent/SF')
plt.title('Density Plot of Average Effective Rent/SF by Unit Type (Excluding␣
 ↪Outliers)', fontsize=15)
plt.legend()
plt.show()
```



3 2 Categorical Variables

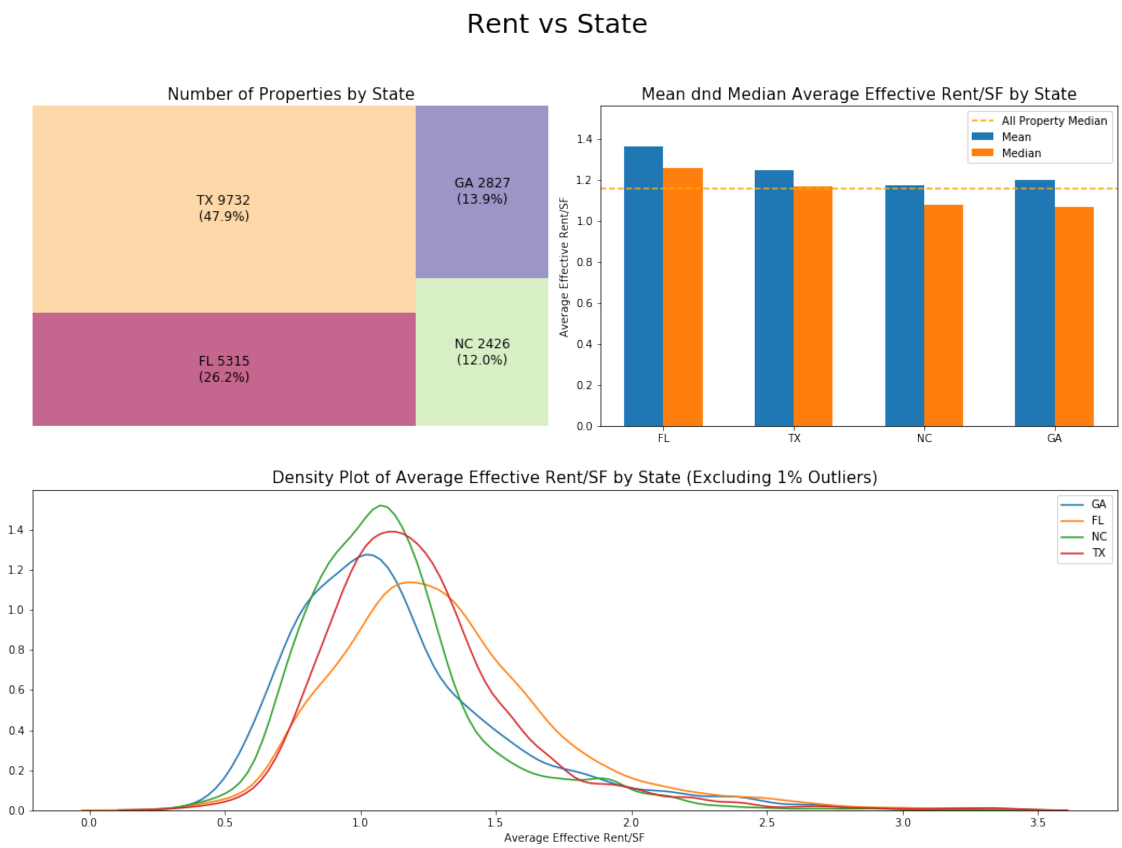
[15]:
```
def get_summary(cat):
    sub[cat] = sub[cat].fillna('Unspecified')
    summary = pd.DataFrame(sub.groupby(cat)['Avg Effective/SF'].\
                           mean().reset_index(name='Mean'))
    summary = summary.merge(pd.DataFrame(sub.groupby(cat)['Avg Effective/SF'].\
                                         median().reset_index(name='Median')))
    summary = summary.merge(pd.DataFrame(sub.groupby(cat)['PropertyID'].\
                                         count().reset_index(name='Count')))
    summary['Pct'] = summary['Count']/summary['Count'].sum()
    summary = summary.sort_values('Median', ascending=False)
    summary['label'] = summary.apply(lambda x: x[0]+'␣
 ↪'+str(x[3])+'\n('+str(round(x[4]*100,1))+'%)', axis=1)
    return summary
```

[16]:
```
def create_plot(cat):
    summary = get_summary(cat)

    fig = plt.figure(figsize=(18,12))

    # Set up grid
    grid = plt.GridSpec(2, 2, wspace = 0.1, hspace = 0.2)
    tree = fig.add_subplot(grid[0,0])
    bar = fig.add_subplot(grid[0,1])
```

```
    dist = fig.add_subplot(grid[1,0:2])

    # Tree map
    colors= plt.get_cmap('Spectral')(np.linspace(0,1,summary.shape[0]))
    squarify.plot(sizes=summary['Count'],label=summary['label'],␣
↪color=colors,alpha=0.6, ax=tree,
                  text_kwargs={'fontsize':12})
    tree.axis('off')
    tree.set_title('Number of Properties by {}'.format(cat), fontsize=15)

    # Bar chart
    pos = np.arange(summary.shape[0])
    barwidth = 0.3
    bar.bar(pos-barwidth/2, summary['Mean'], width=barwidth, label='Mean')
    bar.bar(pos+barwidth/2, summary['Median'], width=barwidth, label='Median')
    ymax = round((summary['Mean'].max()+0.2),2)
    bar.set_ylim(0,ymax)
    #bar.set_yticks(np.arange(0,1.6,0.2))
    bar.set_xticks(pos)
    bar.set_xticklabels(summary[cat])
    bar.axhline(sub['Avg Effective/SF'].median(),
                color='orange', linestyle='--',
                label='All Property Median')
    bar.legend(loc=1)
    bar.set_title('Mean dnd Median Average Effective Rent/SF by {}'.
↪format(cat), fontsize=15)
    bar.set_ylabel('Average Effective Rent/SF')

    # Density plot
    for value in sub[cat].unique():
        df = sub[(sub[cat]==value) & \
                 (sub['Avg Effective/SF']<sub['Avg Effective/SF'].quantile(0.
↪99))]['Avg Effective/SF']
        sns.distplot(df, hist=False, label=value, ax=dist)
    dist.set_title('Density Plot of Average Effective Rent/SF by {} (Excluding␣
↪1% Outliers)'.format(cat), fontsize=15)
    dist.set_xlabel('Average Effective Rent/SF')

    fig.suptitle('Rent vs {}'.format(cat), fontsize=25)

    plt.show()
```

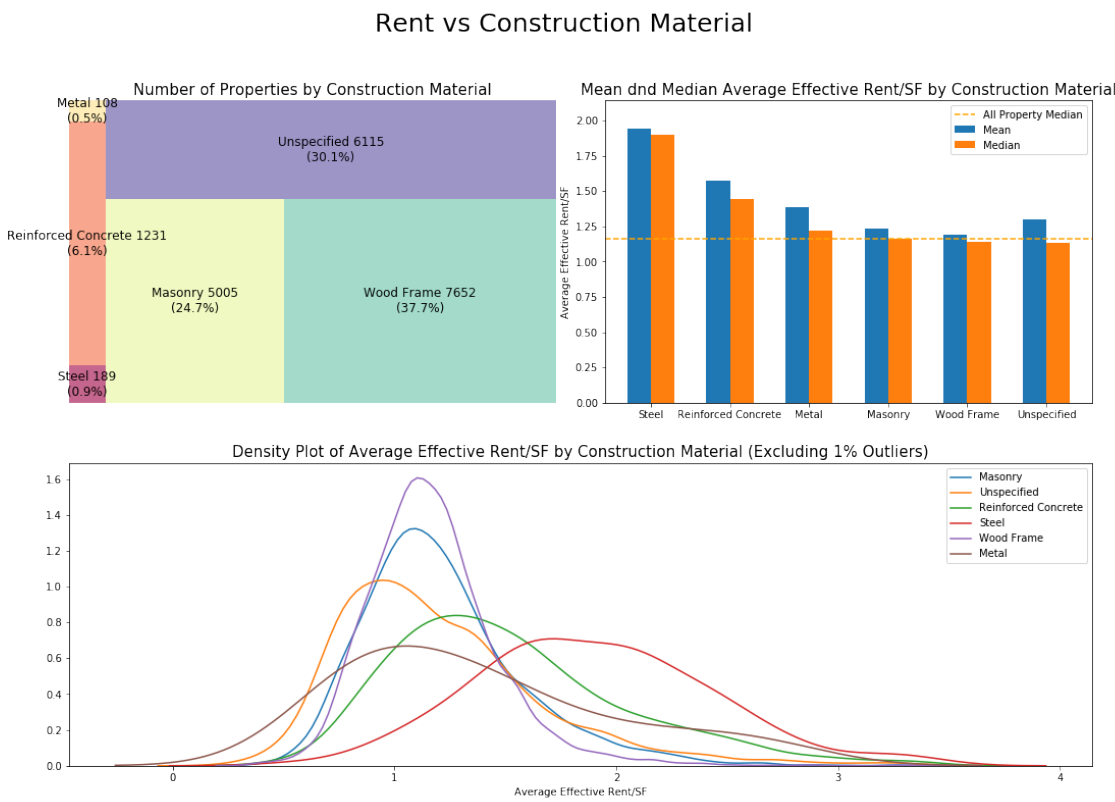
3.0.1 2.1 State

[17]:
```
cat = 'State'
create_plot(cat)
```



3.0.2 2.2 Construction Material

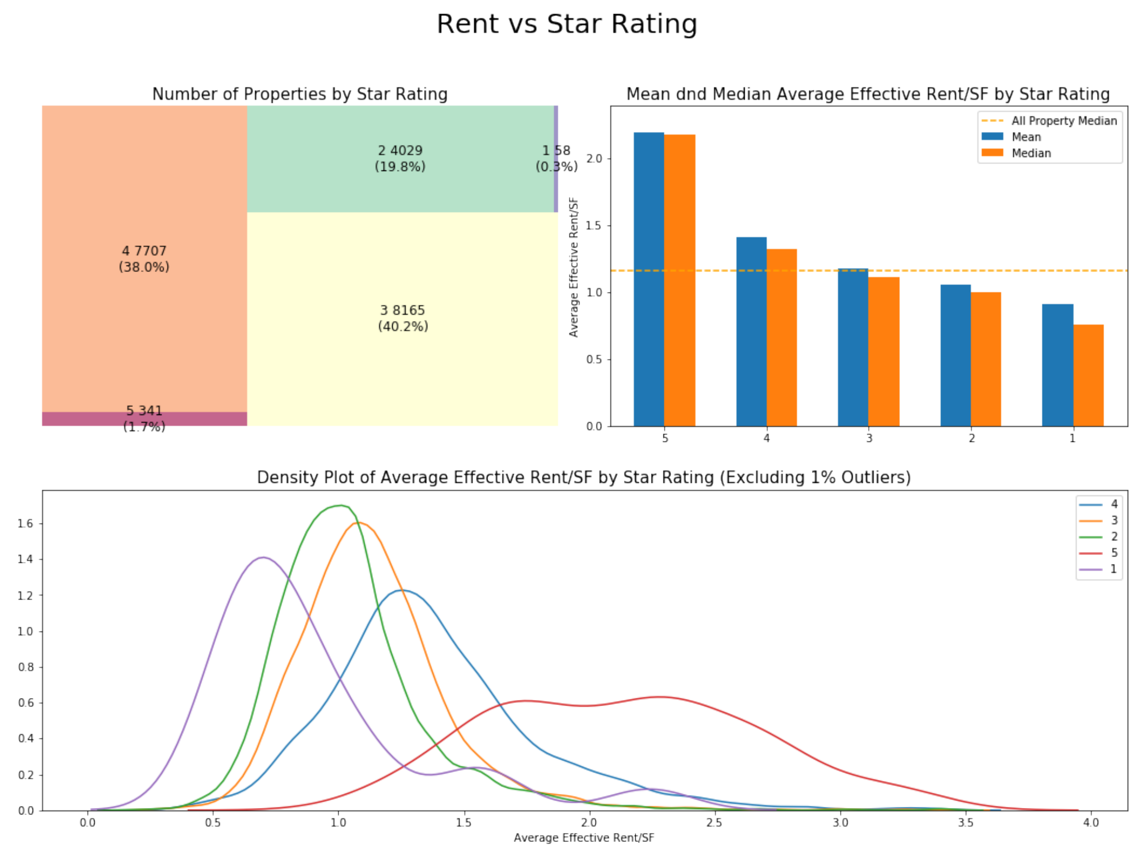
[18]:
```
cat = 'Construction Material'
create_plot(cat)
```



3.0.3 2.3 Star Rating

[19]:
```
sub['Star Rating'] = sub['Star Rating'].astype(str)
```

[20]:
```
cat = 'Star Rating'
create_plot(cat)
```



3.0.4 2.4 Building Class

[21]:
```
cat = 'Building Class'
summary = get_summary(cat)
summary = summary[summary['Building Class']!='Unspecified']
```

[22]:
```
fig = plt.figure(figsize=(18,12))

# Set up grid
grid = plt.GridSpec(2, 2, wspace = 0.1, hspace = 0.2)
tree = fig.add_subplot(grid[0,0])
bar = fig.add_subplot(grid[0,1])
dist = fig.add_subplot(grid[1,0:2])

# Tree map
colors= plt.get_cmap('Spectral')(np.linspace(0,1,summary.shape[0]))
squarify.plot(sizes=summary['Count'],label=summary['label'],␣
↪color=colors,alpha=0.6, ax=tree,
              text_kwargs={'fontsize':12})
tree.axis('off')
```

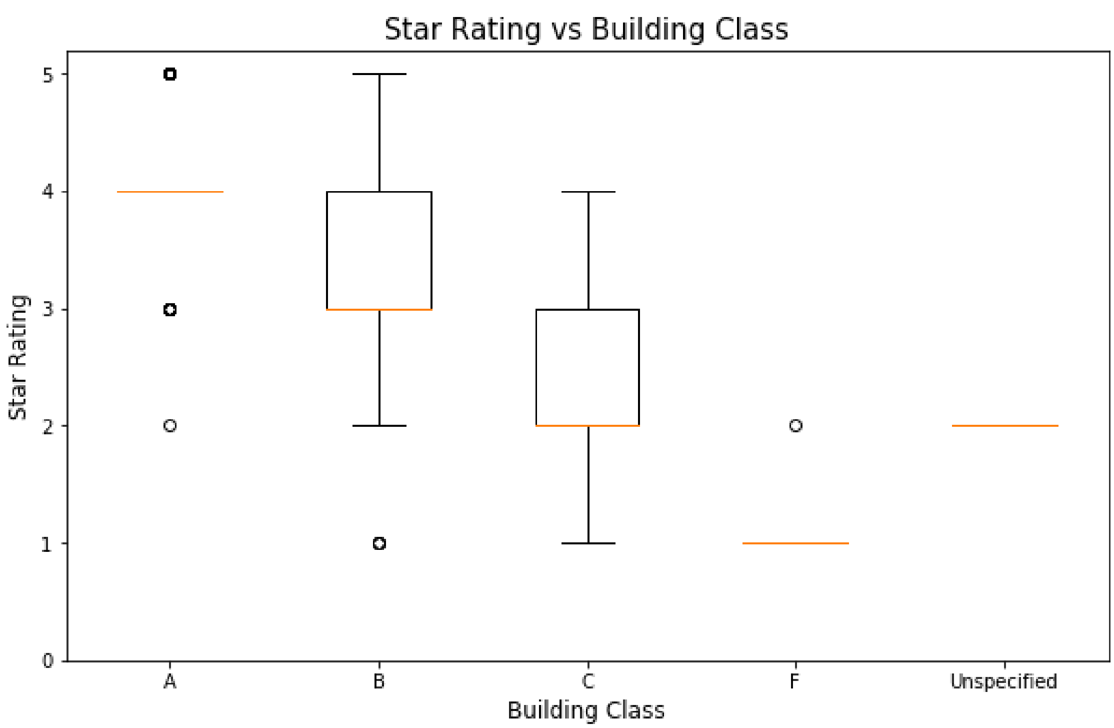
```python
tree.set_title('Number of Properties by {}'.format(cat), fontsize=15)

# Bar chart
pos = np.arange(summary.shape[0])
barwidth = 0.3
bar.bar(pos-barwidth/2, summary['Mean'], width=barwidth, label='Mean')
bar.bar(pos+barwidth/2, summary['Median'], width=barwidth, label='Median')
ymax = round((summary['Mean'].max()+0.2),2)
bar.set_ylim(0,ymax)
#bar.set_yticks(np.arange(0,1.6,0.2))
bar.set_xticks(pos)
bar.set_xticklabels(summary[cat])
bar.axhline(sub['Avg Effective/SF'].median(),
            color='orange', linestyle='--',
            label='All Property Median')
bar.legend(loc=1)
bar.set_title('Mean dnd Median Average Effective Rent/SF by {}'.format(cat),␣
↪fontsize=15)
bar.set_ylabel('Average Effective Rent/SF')

# Density plot
for value in sub[cat].unique():
    df = sub[(sub[cat]==value) & \
             (sub['Avg Effective/SF']<sub['Avg Effective/SF'].quantile(0.
↪99))]['Avg Effective/SF']
    sns.distplot(df, hist=False, label=value, ax=dist)
dist.set_title('Density Plot of Average Effective Rent/SF by {} (Excluding 1%␣
↪Outliers)'.format(cat), fontsize=15)
dist.set_xlabel('Average Effective Rent/SF')

fig.suptitle('Rent vs {}'.format(cat), fontsize=25)

plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/seaborn/distributions.py:198: RuntimeWarning: Mean of empty slice.
  line, = ax.plot(a.mean(), 0)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/core/_methods.py:85: RuntimeWarning: invalid value encountered in
double_scalars
  ret = ret.dtype.type(ret / rcount)
```

3.0.5 2.5 Star Rating vs Building Class

[23]:
```
classes = {}
for i in sub['Building Class'].unique():
    name = '{}'.format(i)
    classes[name] = sub[sub['Building Class']==i]['Star Rating'].astype(int)
```

[24]:
```
plt.figure(figsize=(10,6))
plt.boxplot(classes.values(), labels=classes.keys())
plt.title("Star Rating vs Building Class", fontsize=15)
plt.xlabel("Building Class", fontsize=12)
plt.ylabel("Star Rating",fontsize=12)
plt.yticks(np.arange(0,5.5,1))
plt.show()
```

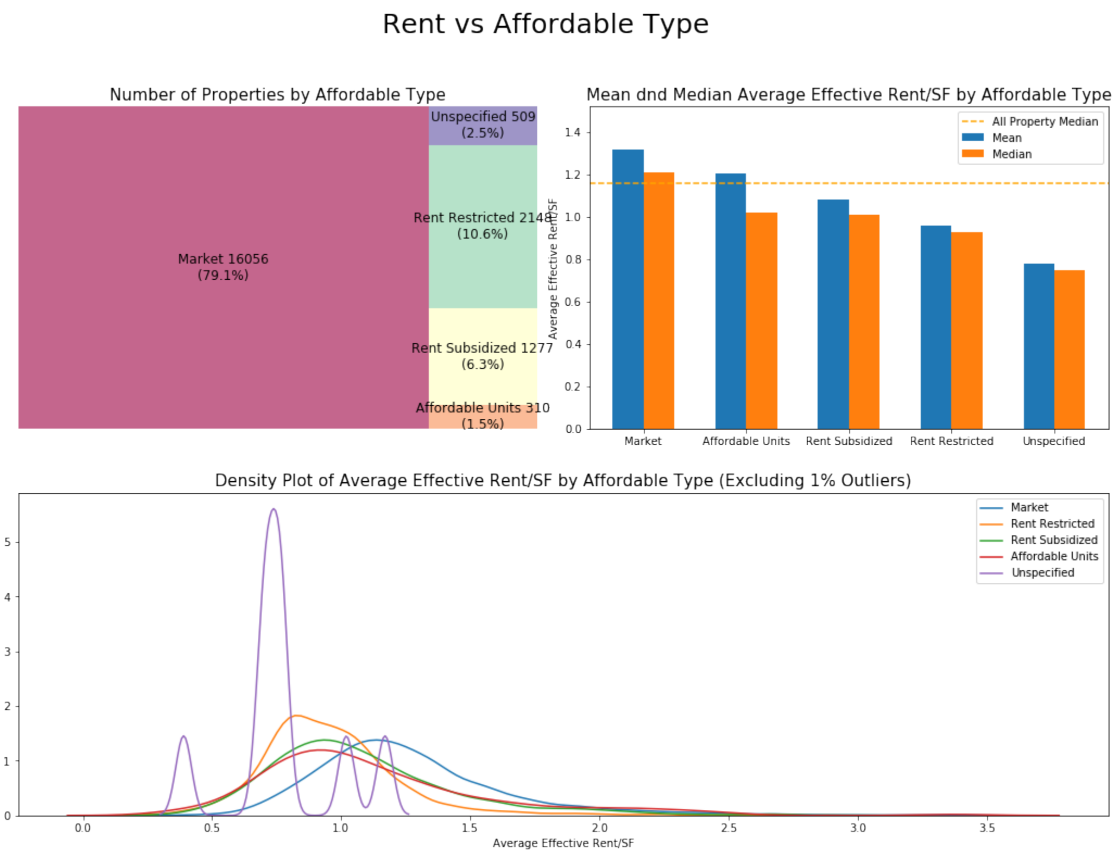


3.0.6 2.6 Affordable type

- Rent Controlled & Rent Stabilized combined in Rent Restricted

[25]:
```
sub['Rent Type'].fillna('Unspecified', inplace=True)
sub['Affordable Type'].fillna(sub['Rent Type'], inplace=True)
sub.loc[sub['Affordable Type']=='Affordable', 'Affordable Type'] = 'Unspecified'
sub.loc[sub['Affordable Type']=='Market/Affordable', 'Affordable Type'] =␣
 ↪'Unspecified'
sub.loc[sub['Affordable Type']=='Rent Stabilized', 'Affordable Type'] = 'Rent␣
 ↪Restricted'
sub.loc[sub['Affordable Type']=='Rent Controlled', 'Affordable Type'] = 'Rent␣
 ↪Restricted'
```

[26]:
```
cat = 'Affordable Type'
create_plot(cat)
```



3.0.7 2.7 Owner type

- Large owner: manages >=50 properties
- Medium owner: manages [10,50) properties
- Small owner: manages <10 properties or unspecified owner

[27]:
```
owner = sub['Owner Name'].value_counts().reset_index(name='count')
print('large owner:', owner[owner['count']>=50].shape[0])
print('medium owner:', owner[(owner['count']>=10) & (owner['count']<50)].
 ↪shape[0])
print('small owner:', owner[owner['count']<10].shape[0])
```

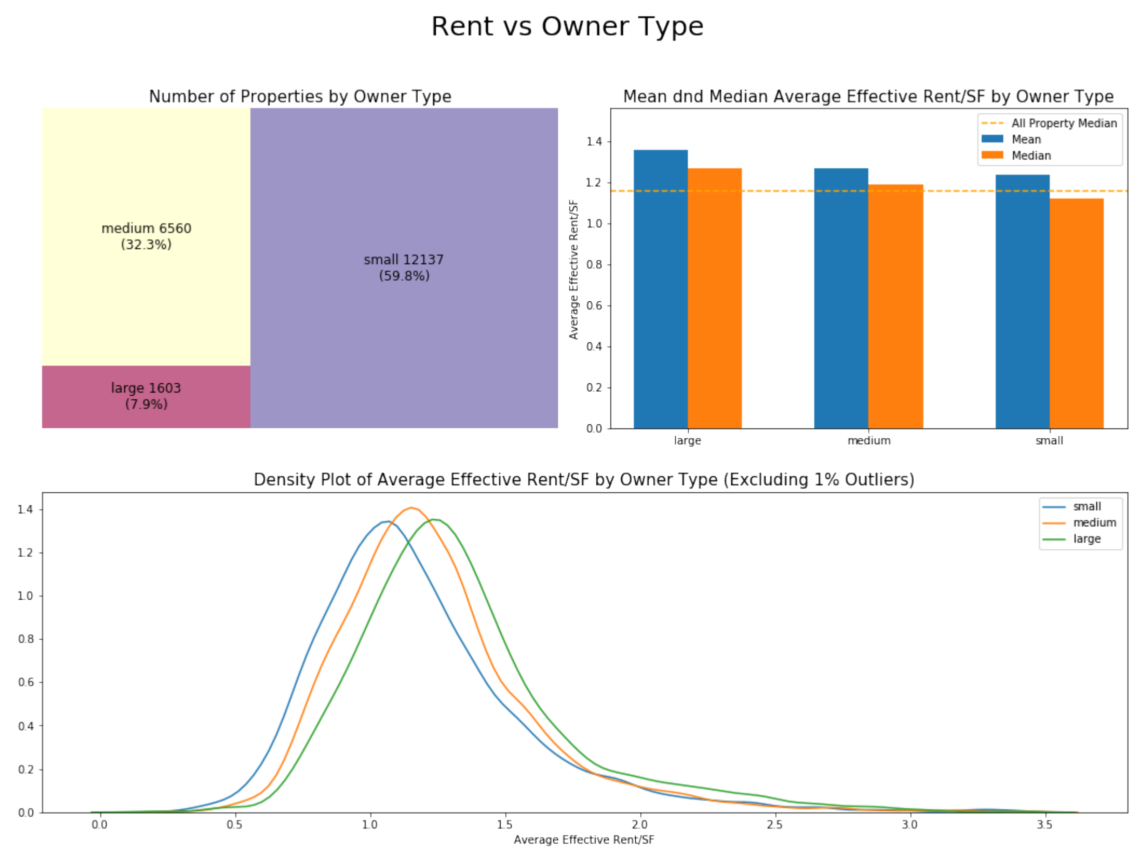
```
large owner: 20
medium owner: 363
small owner: 6733
```

[28]:
```
large_owner = owner[owner['count']>=50]['index'].tolist()
medium_owner = owner[(owner['count']>=10) & (owner['count']<50)]['index'].
 ↪tolist()
small_owner = owner[owner['count']<10]['index'].tolist()
```

```
[29]: sub['Owner Type'] = sub['Owner Name'].apply(lambda x: 'large' if x in␣
      ↪large_owner else(
                                              'medium' if x in medium_owner else␣
      ↪'small'))
      sub['Owner Type'].value_counts()
```

```
[29]: small     12137
      medium     6560
      large      1603
      Name: Owner Type, dtype: int64
```

```
[30]: cat = 'Owner Type'
      create_plot(cat)
```



3.0.8 2.8 Amenities

```
[31]: from os import path
      from PIL import Image
      from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

[32]:
```
text = " ".join(amenity for amenity in sub.Amenities.dropna())
wordcloud = WordCloud(max_font_size=150, max_words=100,␣
↪background_color="white").generate(text)
plt.figure(figsize=(10,6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

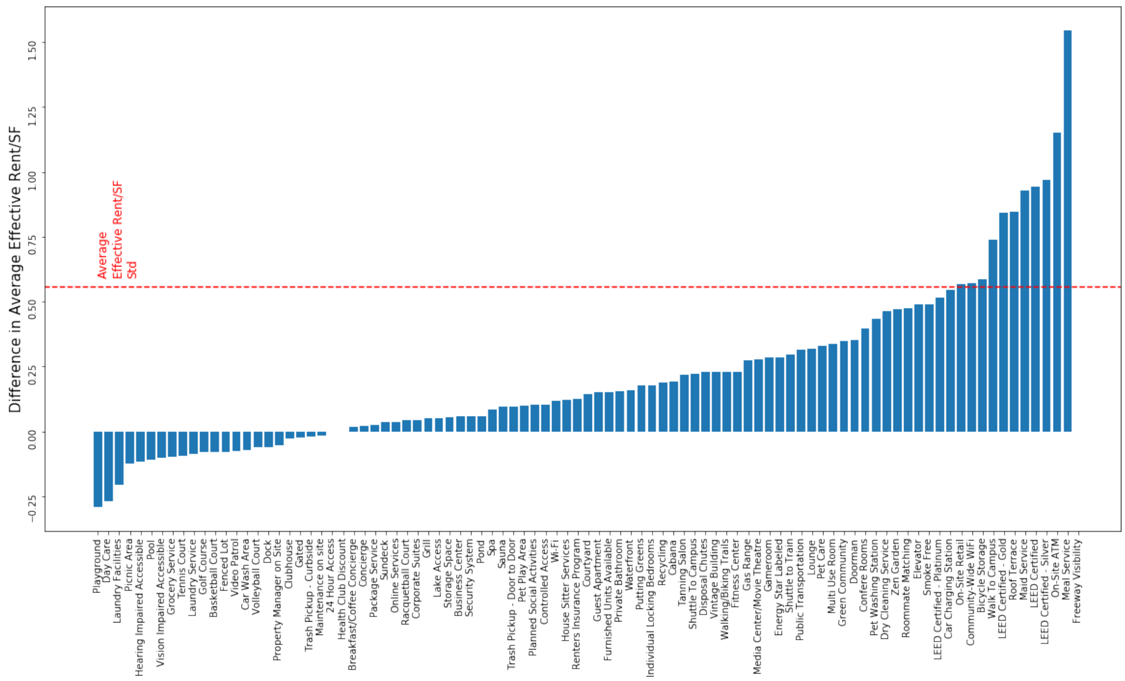
[33]:
```
amenities = {}
all_amenities = sub[sub['Amenities'].notnull()]['Amenities'].str.split(', ').
↪tolist()
for row in all_amenities:
    #print(row)
    for item in row:
        #print(item)
        if item not in amenities.keys():
            amenities[item] = 1
        else:
            amenities[item] += 1
len(amenities)
```

[33]: 94

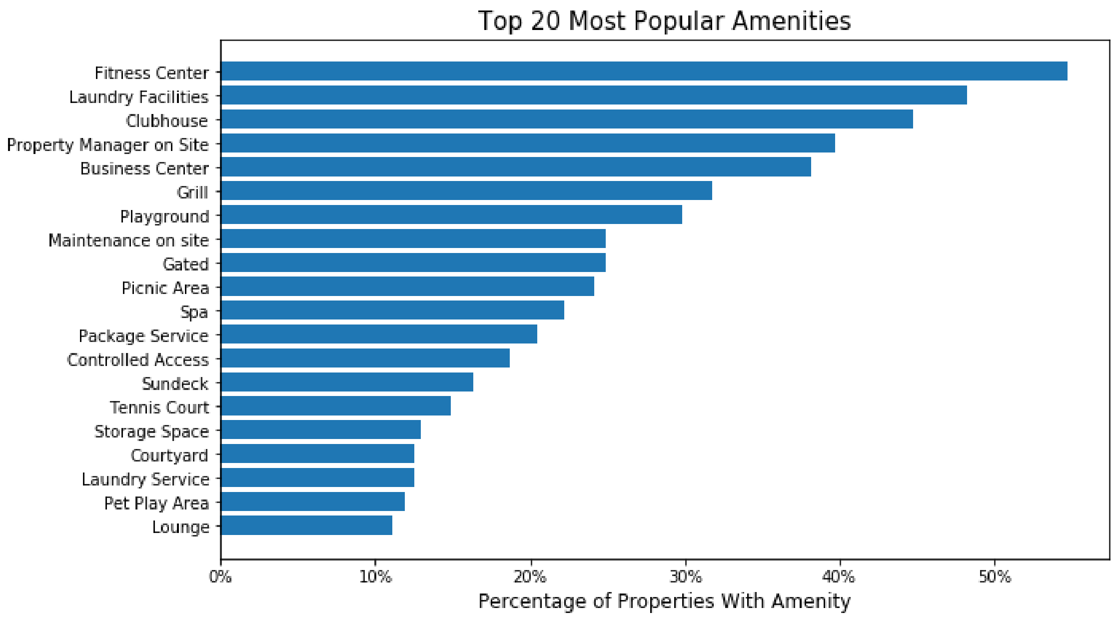
[34]:
```
amenity_vs_rent = pd.DataFrame()

sub2 = sub[sub['Amenities'].notnull()]

for amenity in amenities:
```

```
    yes = sub2[sub2['Amenities'].str.contains(amenity)]['Avg Effective/SF'].
↪mean()
    no = sub2[-sub2['Amenities'].str.contains(amenity)]['Avg Effective/SF'].
↪mean()
    count = sub2[sub2['Amenities'].str.contains(amenity)]['Avg Effective/SF'].
↪count()
    amenity_vs_rent = amenity_vs_rent.append(pd.
↪DataFrame([amenity,count,yes,no,yes-no]).T,
                                            ignore_index=True)

amenity_vs_rent.columns = ['amenity','count','yes', 'no', 'diff']
amenity_vs_rent = amenity_vs_rent.sort_values('diff')
```

[46]:
```
test = amenity_vs_rent[amenity_vs_rent['amenity']!='Study Lounge']

plt.figure(figsize=(20,10))
plt.bar(np.arange(test.shape[0]),
        test['diff'])
plt.xticks(np.arange(test.shape[0]),
           test['amenity'],
           rotation=90)
plt.yticks(rotation=90)
plt.axhline(sub['Avg Effective/SF'].std(),
            color='red',
            linestyle='--')
#plt.title('Average Effective Rent/SF With vs Without Amenity', fontsize=15)
plt.ylabel('Difference in Average Effective Rent/SF', fontsize=15)
plt.annotate('Average\nEffective Rent/SF\nStd',
             (0,0.6), fontsize=12, color='red',
             rotation=90)
plt.show()
```




```
[36]: amenity_vs_rent_short = amenity_vs_rent.sort_values('count').tail(20)
amenity_vs_rent_short['pct'] = amenity_vs_rent_short['count']/sub.shape[0]
pos = np.arange(amenity_vs_rent_short.shape[0])
plt.figure(figsize=(10,6))
plt.barh(pos, amenity_vs_rent_short['pct'])
plt.xticks(np.arange(0,0.6,0.1),
           [str(i)+'%' for i in range(0,60,10)])
plt.yticks(pos, amenity_vs_rent_short['amenity'])
plt.title('Top 20 Most Popular Amenities', fontsize=15)
plt.xlabel('Percentage of Properties With Amenity',fontsize=12)
plt.show()
```



4 3 Numerical Variables

[37]:
```
cols = ['Avg Effective/SF', 'Closest Transit Stop Dist (mi)', 'Land Area (AC)',␣
 ↪'Number Of Stories',
        'Number Of Units', 'Vacancy %', 'Avg Unit SF', 'RBA', 'Year Built',␣
 ↪'Year Renovated',
        'MedanHHIncome(000)', 'married %', 'male/female', '2019 Avg Age(1m)',␣
 ↪'Deposit (000s) Per Capita',
        '2019 Pop Age <19(1m)', '2019 Pop Age 20-64(1m)','2019 Pop Age 65+(1m)']
numerical = sub[cols]
numerical['Year Renovated'] = numerical['Year Renovated'].
 ↪fillna(numerical['Year Built'])
numerical['2019 Pop Tot'] = numerical[['2019 Pop Age <19(1m)', '2019 Pop Age␣
 ↪20-64(1m)',
                                        '2019 Pop Age 65+(1m)']].sum(axis=1)
numerical.shape
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
```

```
packages/ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

[37]:
```
(20300, 19)
```

[38]:
```python
acre_to_sf = 43560
numerical.loc[numerical['Land Area (AC)'] == numerical['Land Area (AC)'].max(),␣
 ↪'Land Area (AC)'] = \
numerical['Land Area (AC)'].max()/acre_to_sf
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/pandas/core/indexing.py:543: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

[39]:
```python
from sklearn.preprocessing import StandardScaler
```

[40]:
```python
scaler = StandardScaler()
scaler.fit(numerical)
scaled_numerical = pd.DataFrame(scaler.transform(numerical))
scaled_numerical.columns = numerical.columns
```

[41]:
```python
scaled_numerical.head()
```

[41]:

	Avg Effective/SF	Closest Transit Stop Dist (mi)	Land Area (AC) \
0	0.361699	NaN	-0.171852
1	NaN	-0.123263	-0.280836
2	NaN	0.078472	-0.367047
3	1.531013	-1.401603	-0.334229
4	1.531013	-1.417042	NaN

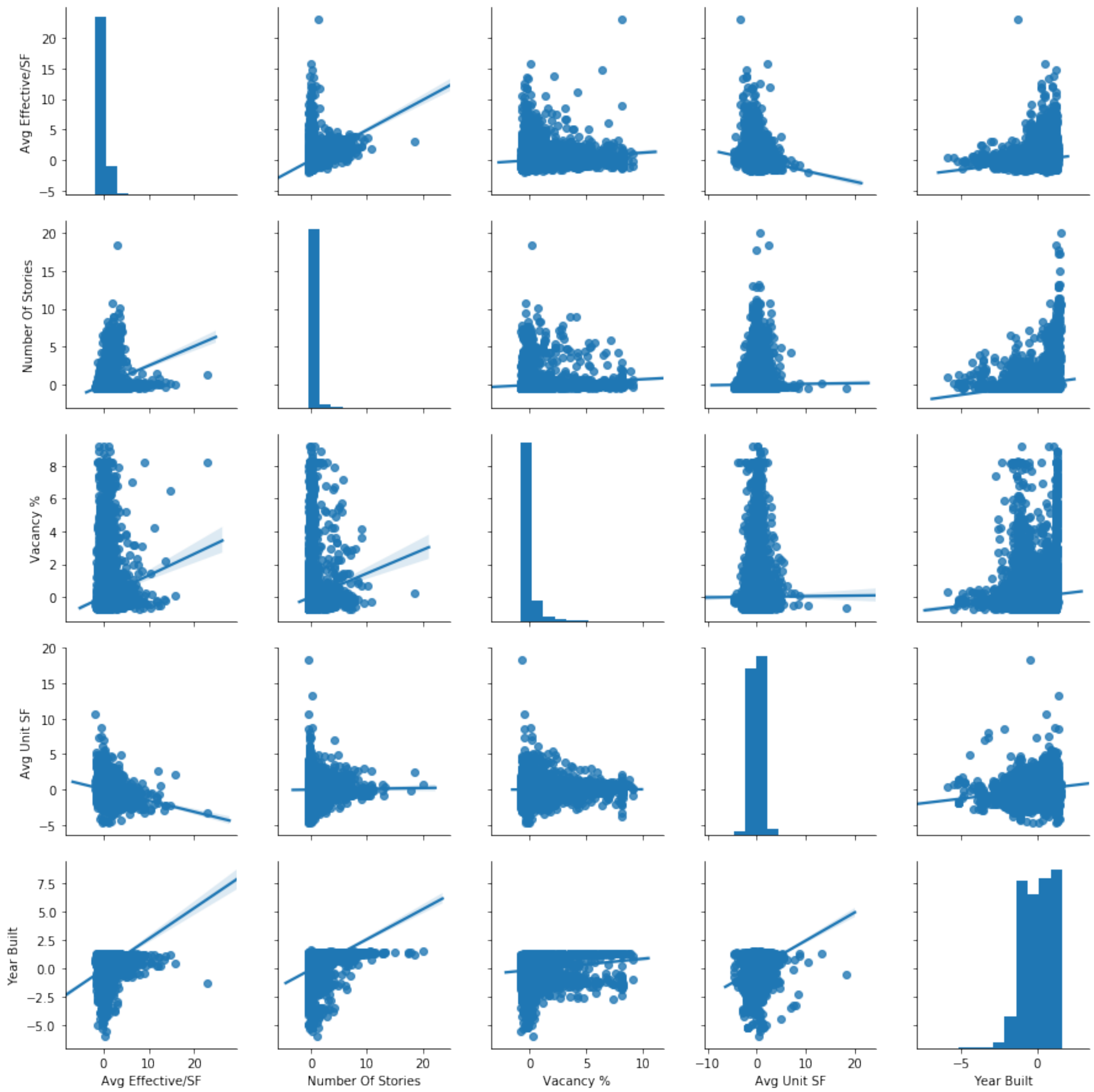
	Number Of Stories	Number Of Units	Vacancy %	Avg Unit SF	RBA \
0	-0.100175	-0.264579	-0.335072	1.243410	0.029041
1	NaN	0.083632	NaN	0.624718	0.073465
2	0.126189	-0.329062	-0.214690	NaN	-0.023871
3	0.126189	-0.406443	-0.285327	0.144081	-0.054726
4	1.258008	-0.857828	0.334493	0.788339	0.587699

```
   Year Built  Year Renovated  MedanHHIncome(000)  married %  male/female  \
0    0.650239        0.491091            1.985256   2.023719    -0.083175
1    1.438444        1.303403           -0.488398  -1.637035     0.686169
2    0.072222       -0.104605           -0.488398  -1.637035     0.686169
3    0.282410        0.112011            1.115502  -0.914180     0.452612
4   -1.346547       -1.566768            1.115502  -0.914180     0.452612

   2019 Avg Age(1m)  Deposit (000s) Per Capita  2019 Pop Age <19(1m)  \
0         -0.103248                   1.738689              0.695878
1         -1.029266                   1.738689              0.183433
2         -1.161554                   1.738689              0.565230
3          0.066837                   1.738689              0.982965
4         -0.500113                   1.738689              1.436216

   2019 Pop Age 20-64(1m)  2019 Pop Age 65+(1m)  2019 Pop Tot
0                0.466699             -0.178112      0.470115
1                1.316380             -0.656441      0.853241
2                1.836947             -0.731128      1.291466
3                2.811152              0.922045      2.264765
4                3.812098              0.390848      2.987131
```

[42]:
```
cols = ['Avg Effective/SF', 'Number Of Stories', 'Vacancy %', 'Avg Unit SF',␣
 ↪'Year Built']
scaled_num1 = scaled_numerical[cols]
sns.pairplot(scaled_num1, kind='reg')
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/histograms.py:754: RuntimeWarning: invalid value encountered
in greater_equal
  keep = (tmp_a >= first_edge)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/histograms.py:755: RuntimeWarning: invalid value encountered
in less_equal
  keep &= (tmp_a <= last_edge)
```

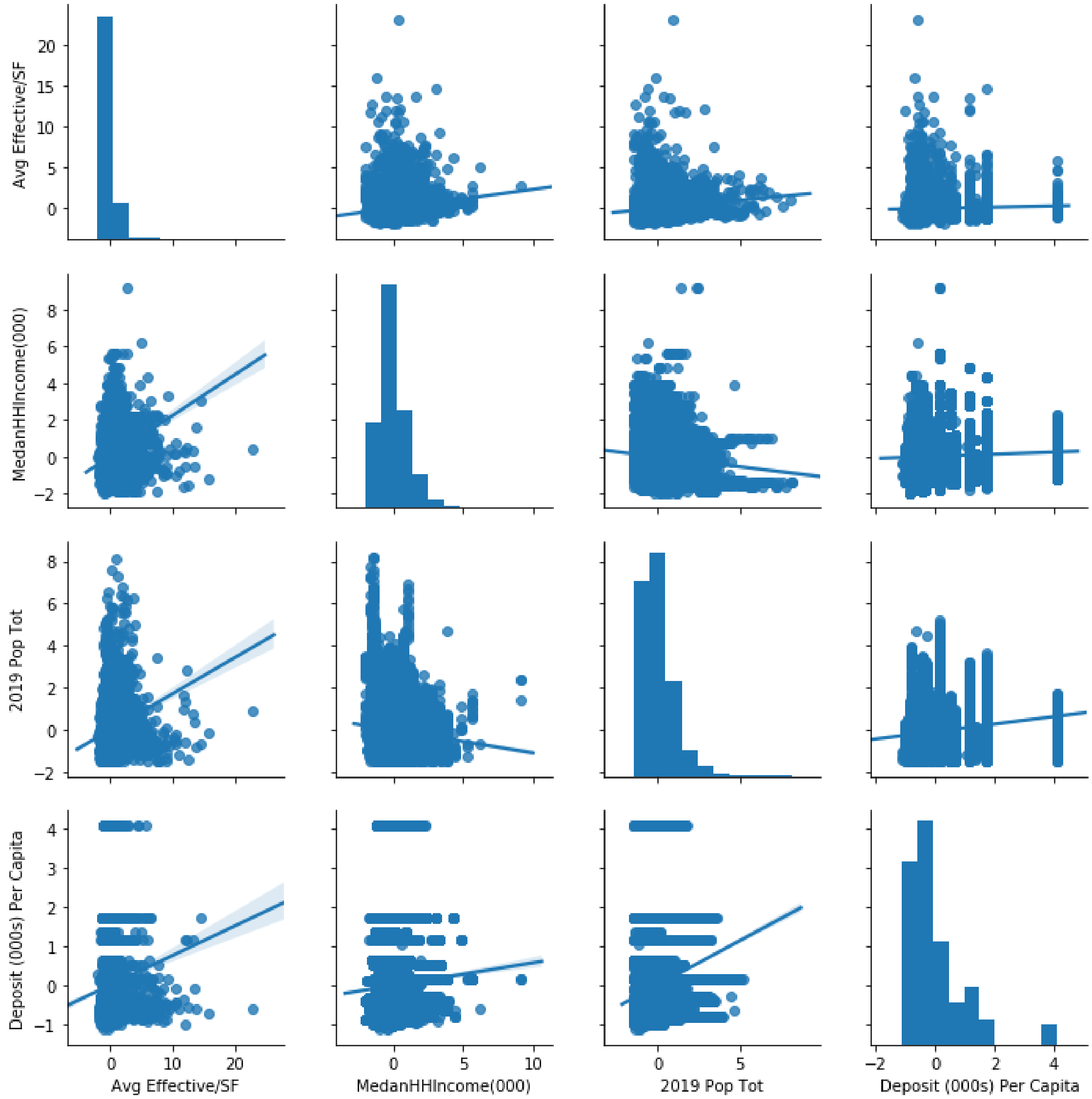
[42]:
```
<seaborn.axisgrid.PairGrid at 0x125064940>
```



[43]:
```
cols = ['Avg Effective/SF', 'MedanHHIncome(000)', '2019 Pop Tot', 'Deposit␣
↪(000s) Per Capita']
scaled_num3 = scaled_numerical[cols]
sns.pairplot(scaled_num3, kind='reg')
```

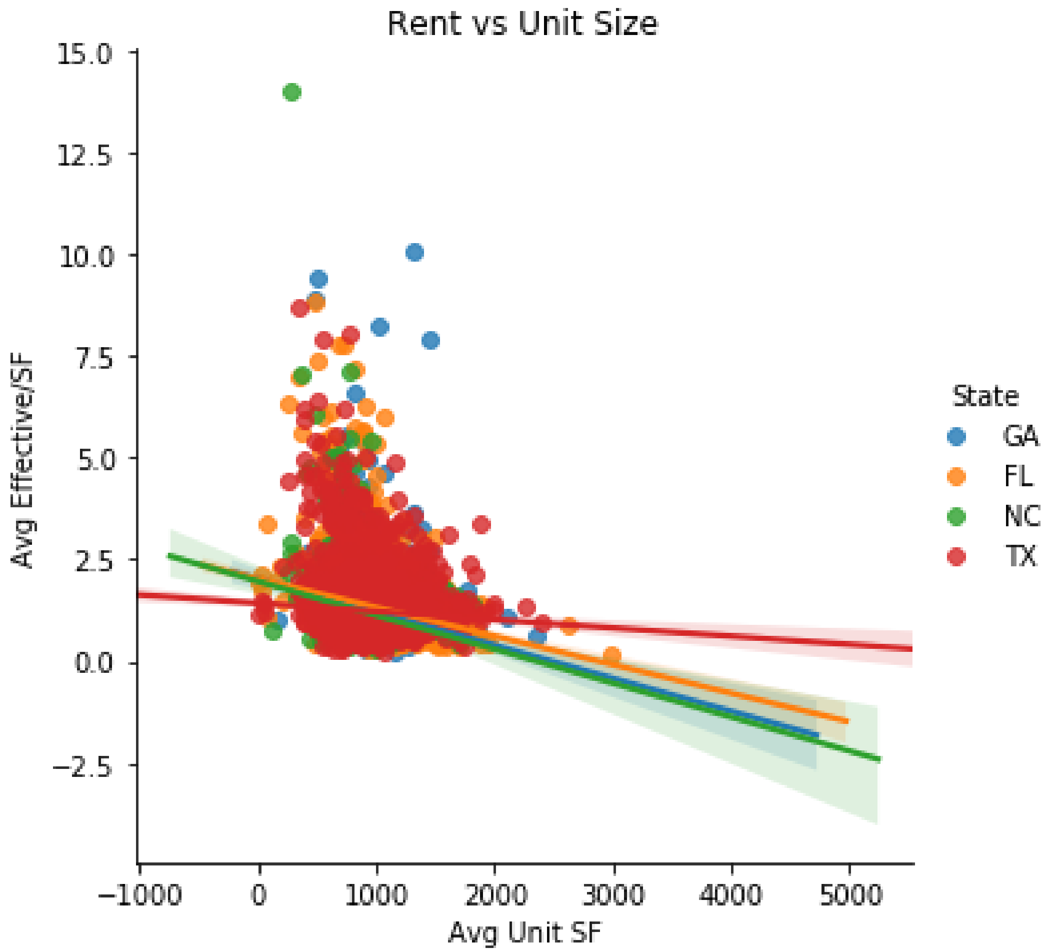
```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/histograms.py:754: RuntimeWarning: invalid value encountered
in greater_equal
  keep = (tmp_a >= first_edge)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/numpy/lib/histograms.py:755: RuntimeWarning: invalid value encountered
in less_equal
  keep &= (tmp_a <= last_edge)
```

[43]: <seaborn.axisgrid.PairGrid at 0x126fae6a0>




```
[44]: plt.figure(figsize=(10,6))
sns.lmplot('Avg Unit SF', 'Avg Effective/SF', data=sub,
           fit_reg=True, hue='State', legend=True)
plt.title("Rent vs Unit Size")
plt.show()
```

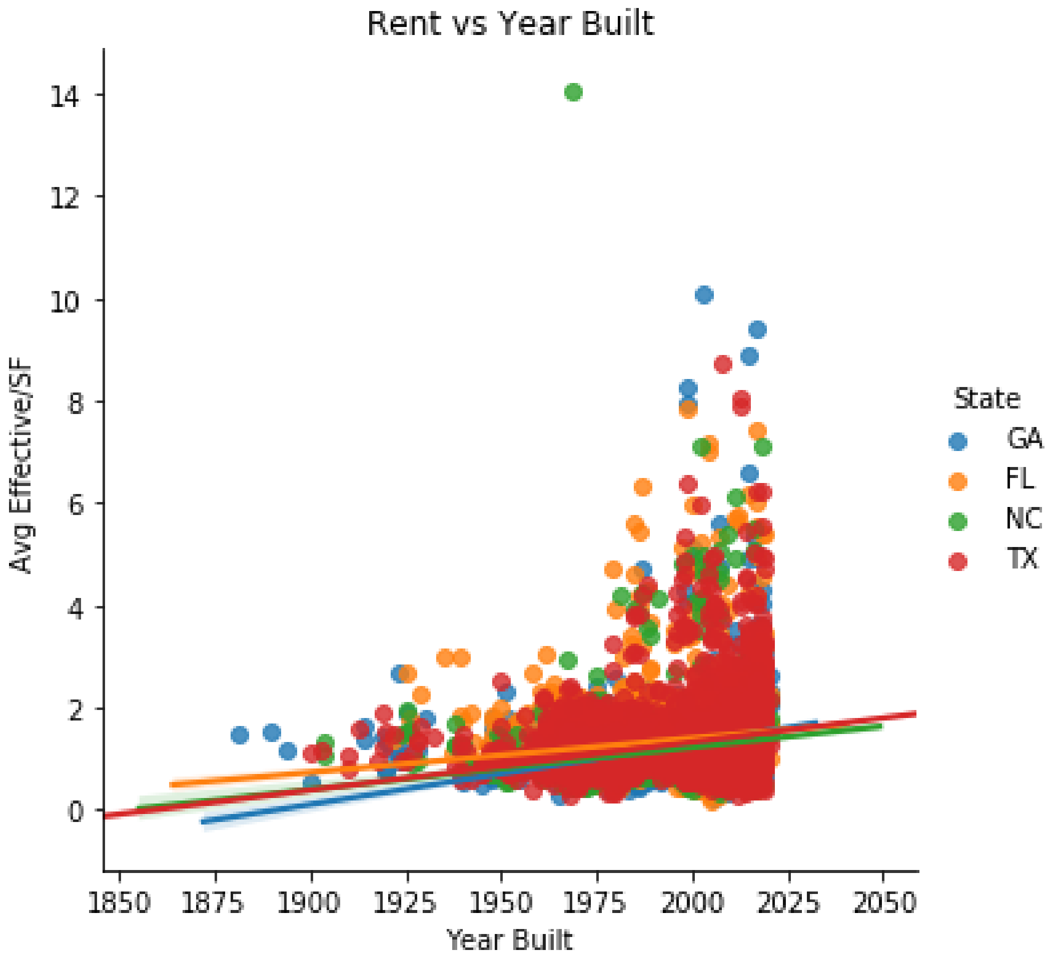
<Figure size 720x432 with 0 Axes>



[45]:

```
plt.figure(figsize=(10,6))
sns.lmplot('Year Built', 'Avg Effective/SF', data=sub,
           fit_reg=True, hue='State', legend=True)
plt.title("Rent vs Year Built")
plt.show()
```

<Figure size 720x432 with 0 Axes>


Rent vs Year Built
Avg Effective/SF
Year Built
14
12
10
8
6
4
2
0
1850
1875
1900
1925
1950
1975
2000
2025
2050
State
GA
FL
NC
TX

```
[ ]:
```