Group 7 - Part B - Montclair Housing Data

October 16, 2020

1 Initial Setup

[1]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm

from scipy import stats
from matplotlib.lines import Line2D
```

[2]:
```
os.chdir(r'C:\Users\Crique\Desktop\Montclair\Stats\Project')
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

2 Read in the Data

[3]:
```
df = pd.read_csv('Montclair Housing Dataset.csv')
```

2.1 Make columns a little easier to work with

[4]:
```
print(df.columns)
df.columns = ['price', 'soldDate', 'propertyType', 'beds', 'baths', 'lotSize',␣
 ↪'daysOnMarket']
df.describe()
```

```
Index(['PRICE', 'SOLD DATE', 'PROPERTY TYPE', 'BEDS', 'BATHS', 'LOT SIZE',
       'DAYS ON MARKET'],
      dtype='object')
```

[4]:

	price	beds	baths	lotSize	daysOnMarket
count	171.00	170.00	170.00	123.00	171.00
mean	794915.04	4.04	2.89	15541.07	74.65
std	455365.43	1.84	1.23	27313.17	49.13
min	107000.00	0.00	1.00	1863.00	1.00
25%	455000.00	3.00	2.00	7405.00	33.00
50%	760000.00	4.00	3.00	10018.00	62.00

```
75%     987250.00    5.00    3.50   14731.00         117.50
max    3195000.00   12.00    7.00  215186.00         180.00
```

3 Impute Missing Variables

Beds/Baths only have 1 missing value each so we'll simply fill them in with the means for each column

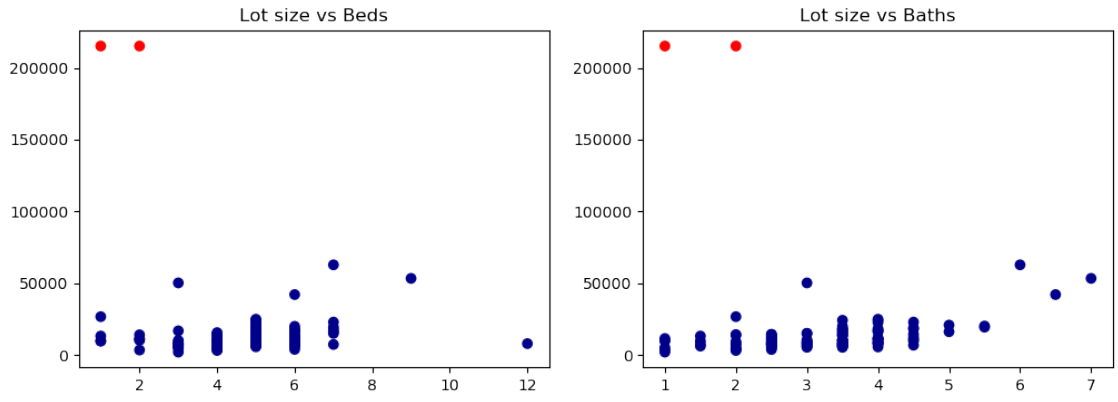
[5]:
```
df.loc[df.beds.isna(), 'beds'] = round(df.beds.mean(), 0)
df.loc[df.baths.isna(), 'baths'] = round(df.baths.mean(), 0)
```

Lot size is missing about a 1/3 of values, so we'll attempt to apply a linear regression to impute the missing values. Let's take a look at the relationship between beds/lot size and baths/lot size.

[6]:
```
dfLots = df[df.lotSize.notna()]
```

3.1 Plotting beds and baths against lot size to check the relationship

[7]:
```
fig, ax = plt.subplots(1, 2, figsize=(12,4), dpi=100)

colors = ['darkblue'] * len(dfLots)

for index, val in enumerate(dfLots.lotSize):
    if val > 100000:
        colors[index] = 'red'

bedPlot = ax[0].scatter(dfLots.beds, dfLots.lotSize, color=colors)
bathPlot = ax[1].scatter(dfLots.baths, dfLots.lotSize, color=colors)

_ = ax[0].set_title('Lot size vs Beds')
_ = ax[1].set_title('Lot size vs Baths')
```



It appears that we have two large outliers for lot size. Let's take a further look.

[8]:
```
dfLots[dfLots.lotSize > 50000].sort_values(by='lotSize', ascending=False)
```

[8]:

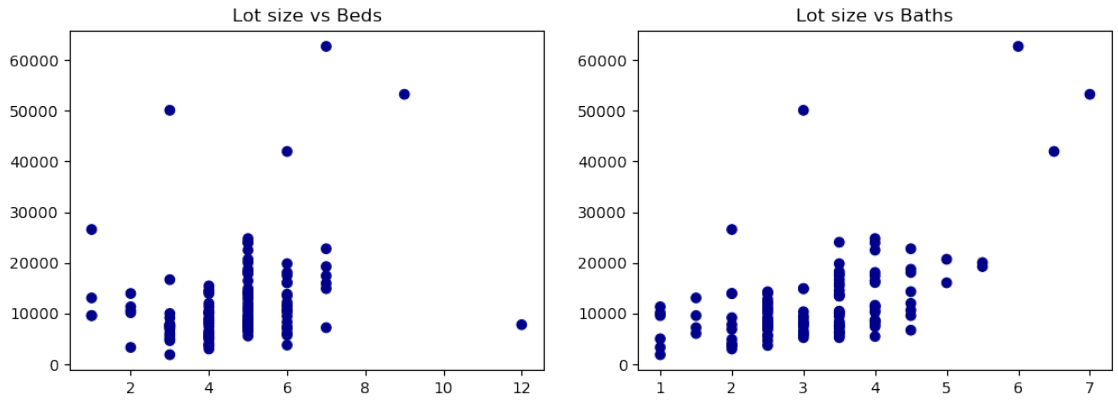
	price	soldDate	propertyType	beds	baths	lotSize	daysOnMarket
43	270000	May-7-2020	Single Family Residential	2.00	2.00	215186.00	146
154	107000	September-1-2020	Single Family Residential	1.00	1.00	215186.00	29
29	2037250	July-15-2020	Single Family Residential	7.00	6.00	62700.00	77
90	1126000	July-6-2020	Single Family Residential	9.00	7.00	53250.00	86
86	605000	April-3-2020	Single Family Residential	3.00	3.00	50093.00	180

Looking at the price of homes for all lot sizes over 50k it's clear that the two lot sizes that are greater than 200k are not valid so we'll see their sizes to N/A for now.

[9]:
```
df.loc[df.lotSize > 200000, 'lotSize'] = np.nan
dfLots = df[df.lotSize.notna()]
```

Let's replot the graphs again.

[10]:
```
fig, ax = plt.subplots(1, 2, figsize=(12,4), dpi=100)

colors = ['darkblue'] * len(dfLots)

bedPlot = ax[0].scatter(dfLots.beds, dfLots.lotSize, color=colors)
bathPlot = ax[1].scatter(dfLots.baths, dfLots.lotSize, color=colors)

_ = ax[0].set_title('Lot size vs Beds')
_ = ax[1].set_title('Lot size vs Baths')
```



I think it's a close call for beds, but thinking about it logically, the exponential seems like a good fit because lot size is going to grow much faster when we have huge houses compared to beds. Baths looks pretty clear cut to be growing exponentially. Let's check the correlation as well.

[11]:
```
dfLots.corr()
```

[11]:

	price	beds	baths	lotSize	daysOnMarket
price	1.00	0.50	0.70	0.47	-0.05
beds	0.50	1.00	0.56	0.27	0.01
baths	0.70	0.56	1.00	0.56	0.02
lotSize	0.47	0.27	0.56	1.00	0.16
daysOnMarket	-0.05	0.01	0.02	0.16	1.00

We can also see that we have some positive correlation between beds/baths on lot size. That's enough for us go ahead and create a regression model to impute the missing values. From our scatter plots, it appears both variables are exponential in nature, so we're going to transform our model as such: ln(lotSize) = b0 + b1 * beds + b2 * baths. I will test out a normal linear regression model first before transforming it.

4 Running Models

4.1 Normal Linear Model with beds/baths

[12]:
```
y = dfLots.lotSize
x = sm.add_constant(dfLots[['beds', 'baths']].values)
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

[12]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                lotSize   R-squared:                       0.314
Model:                            OLS   Adj. R-squared:                  0.302
Method:                 Least Squares   F-statistic:                     26.98
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           2.25e-10
Time:                        19:26:46   Log-Likelihood:                -1251.7
No. Observations:                 121   AIC:                             2509.
Df Residuals:                     118   BIC:                             2518.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
const      -1611.9826   2399.474     -0.672      0.503   -6363.594    3139.629
x1          -363.2134    534.837     -0.679      0.498   -1422.337     695.910
x2          4884.0384    758.582      6.438      0.000    3381.839    6386.238
==============================================================================
Omnibus:                       85.207   Durbin-Watson:                   1.957
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              550.963
Skew:                           2.423   Prob(JB):                    2.29e-120
Kurtosis:                      12.263   Cond. No.                         21.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

After running our initial model we see that we get an Adjusted R-Squred of only .302. Let's see if the exponential model is any better.

4.2 Exponential Model with beds/baths

[13]:
```
e = 2.718282
y = np.log(dfLots.lotSize)
x = sm.add_constant(dfLots[['beds', 'baths']].values)
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

[13]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                lotSize   R-squared:                       0.328
Model:                            OLS   Adj. R-squared:                  0.317
Method:                 Least Squares   F-statistic:                     28.82
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           6.45e-11
Time:                        19:26:46   Log-Likelihood:                -80.340
No. Observations:                 121   AIC:                             166.7
Df Residuals:                     118   BIC:                             175.1
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          8.3344      0.150     55.595      0.000       8.038       8.631
x1            -0.0219      0.033     -0.655      0.514      -0.088       0.044
```

```
x2             0.3143      0.047      6.632      0.000       0.220       0.408
==============================================================================
Omnibus:                        2.475   Durbin-Watson:                   1.840
Prob(Omnibus):                  0.290   Jarque-Bera (JB):                2.020
Skew:                           0.304   Prob(JB):                        0.364
Kurtosis:                       3.177   Cond. No.                         21.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

In both of our models, beds has a p-value of approximately .5, so let's remove it from the models

4.3 Linear Model with baths

[14]:
```
y = dfLots.lotSize
x = sm.add_constant(dfLots.baths.values)
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

[14]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                lotSize   R-squared:                       0.311
Model:                            OLS   Adj. R-squared:                  0.305
Method:                 Least Squares   F-statistic:                     53.73
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           3.02e-11
Time:                        19:26:46   Log-Likelihood:                -1251.9
No. Observations:                 121   AIC:                             2508.
Df Residuals:                     119   BIC:                             2513.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2380.5769   2110.997     -1.128      0.262   -6560.563    1799.409
x1          4595.3995    626.907      7.330      0.000    3354.060    5836.739
==============================================================================
Omnibus:                       86.986   Durbin-Watson:                   1.958
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              576.422
Skew:                           2.480   Prob(JB):                    6.79e-126
Kurtosis:                      12.473   Cond. No.                         11.1
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

4.4 Exponential Model with baths

[15]:
```
e = 2.718282
y = np.log(dfLots.lotSize)
x = sm.add_constant(dfLots.baths.values)
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

[15]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                lotSize   R-squared:                       0.326
Model:                            OLS   Adj. R-squared:                  0.320
Method:                 Least Squares   F-statistic:                     57.48
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           8.25e-12
Time:                        19:26:46   Log-Likelihood:                -80.559
No. Observations:                 121   AIC:                             165.1
Df Residuals:                     119   BIC:                             170.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          8.2881      0.132     62.850      0.000       8.027       8.549
x1             0.2969      0.039      7.582      0.000       0.219       0.374
==============================================================================
Omnibus:                        3.735   Durbin-Watson:                   1.821
Prob(Omnibus):                  0.154   Jarque-Bera (JB):                3.151
Skew:                           0.360   Prob(JB):                        0.207
Kurtosis:                       3.325   Cond. No.                         11.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

Although this model, has a slightly higher asjusted r-2 compared to the normal linear model, we need to calculate the r-2 differently because we transformed the our y variable.

[16]:
```
constants = [1] * len(dfLots.lotSize)
predictors = list(zip(constants, dfLots.baths))

predictions = res.predict(predictors)

errorSum = sum((predictions - y) ** 2)
standardError = np.sqrt(1/(len(dfLots.lotSize)-2)*errorSum)
```

[17]:
```
finalPredictions = np.exp(predictions + (standardError ** 2 / 2))
```

[18]:
```
np.corrcoef(finalPredictions, dfLots.lotSize)[0][1]**2
```

[18]: 0.4259823201605624

As we see here, the expoential model has a final r ^ 2 of .426, so we'll use the exponential model to impute the missing values.

4.5 Using the Model to impute the missing variables

[19]:
```
finalValues = []

for bath, size in zip(df.baths, df.lotSize):
    if pd.isnull(size):
        finalValues.append(np.exp(8.2871 + (.2973 * bath) + (standardError ** 2␣
 ↪/ 2)))
    else:
        finalValues.append(size)

df.lotSize =  finalValues
```

Now we can see that our lot size variable has been filled in nicely with the model's predicted values. We can see below that the lot size has been filled in nicely with our prediction from our baths variable. It may not be perfect, but it'll do for our model.

[20]:
```
df.tail(10)
```

[20]:

	price	soldDate	propertyType	beds	baths	lotSize	\
161	840000	May-5-2020	Single Family Residential	4.00	3.50	12586.44	
162	250000	August-17-2020	Condo/Co-op	2.00	1.00	5985.68	
163	320000	May-15-2020	Condo/Co-op	1.00	1.00	5985.68	
164	364000	April-27-2020	Condo/Co-op	2.00	1.00	5985.68	
165	500000	July-23-2020	Condo/Co-op	2.00	2.50	9349.47	
166	165000	April-28-2020	Condo/Co-op	1.00	1.00	5985.68	

```
167  619000    July-30-2020  Single Family Residential  3.00   3.50 12586.44
168  250000  August-17-2020  Single Family Residential  2.00   1.00 11325.00
169  549000     May-6-2020  Single Family Residential  5.00   2.50 10890.00
170  950000     May-5-2020  Single Family Residential  4.00   3.50 12586.44
```

```
     daysOnMarket
161           148
162            44
163           138
164           156
165            69
166           155
167            62
168            44
169           147
170           148
```

4.6 Now I need to look if there could be other variables of interest for the dataset

I'm going to convert the dates into numbers offset from the first day in the dataset. Then I will check the correlation to see if there's any value in the column.

[21]:
```
from datetime import datetime
```

[22]:
```
df.soldDate.head()

datetime.strptime(df.soldDate.loc[0], '%B-%d-%Y')
dateStamps = [datetime.strptime(date, '%B-%d-%Y') for date in df.soldDate]
minDate = min([datetime.strptime(date, '%B-%d-%Y') for date in df.soldDate])
dateOffset = [(dateStamp - minDate).days for dateStamp in dateStamps]
df['dateOffset'] = dateOffset
'{0:.2f}%'.format(float(np.corrcoef(dateOffset, df.price)[0][1]) * 100)
```

[22]:
```
'13.05%'
```

Here we see the correlation is a measly .13, but for now we'll keep it in the dataset anyway and check it out in our final model. Let's now take a look at the relationship between our numeric data columns and our dependent variable, price.

[23]:
```
df.corr()
```

[23]:
```
              price  beds  baths  lotSize  daysOnMarket  dateOffset
price          1.00  0.67   0.77     0.52         -0.13        0.13
beds           0.67  1.00   0.71     0.37         -0.11        0.11
baths          0.77  0.71   1.00     0.60         -0.12        0.12
lotSize        0.52  0.37   0.60     1.00          0.05       -0.05
daysOnMarket  -0.13 -0.11  -0.12     0.05          1.00       -1.00
```

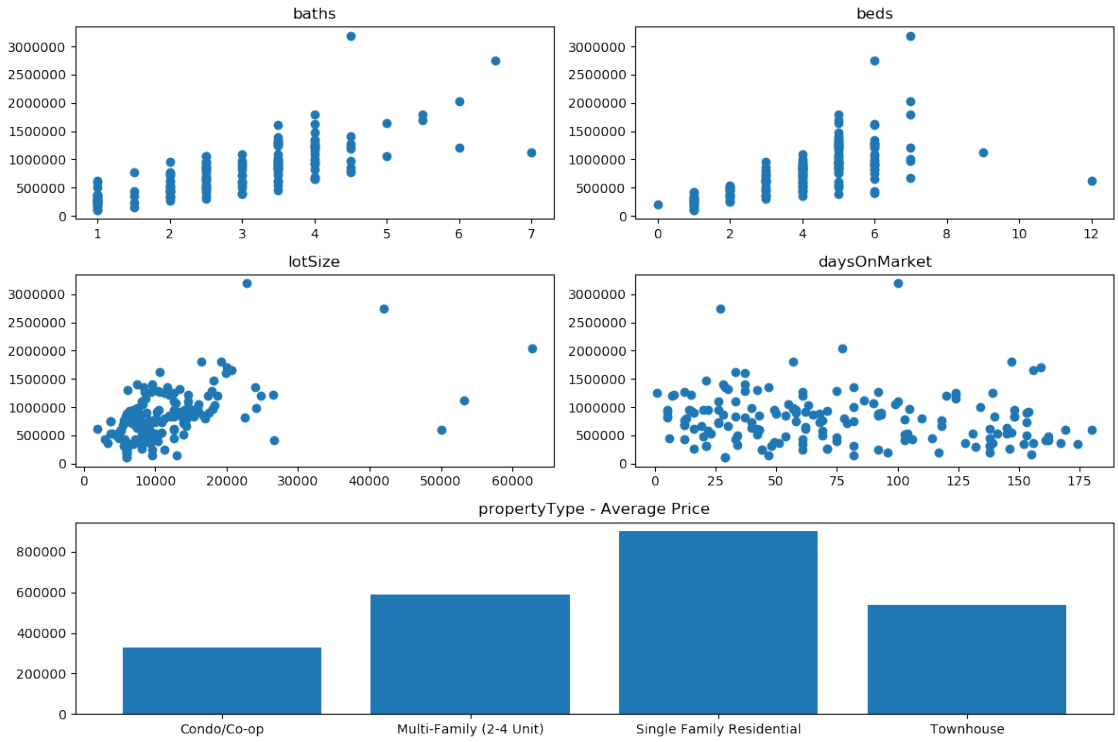
```
dateOffset      0.13   0.11    0.12     -0.05          -1.00          1.00
```

It appears we've made a mistake. The dateOffset column and daysOnMarket column violate the principle of Multicollinearity. I'm going to drop the sold Date and date Offset column that we just created.

[24]:
```
df = df[['price', 'propertyType', 'beds', 'baths', 'lotSize', 'daysOnMarket']]
```

4.7 Checking the relationship between our independent variables and our dependent variable price

[25]:
```
fig = plt.figure(figsize=(12,8), dpi=100)

plots = []
plotColumns = ['baths', 'beds', 'lotSize', 'daysOnMarket']

for i in range(1, 5):
    plots.append(fig.add_subplot(3, 2, i))
    plots[i - 1].scatter(df[plotColumns[i - 1]], df.price)
    plots[i - 1].set_title(plotColumns[i - 1])

plots.append(fig.add_subplot(3, 2, (5, 6)))

propertyAverages = df.groupby('propertyType').mean()['price']
plots[4].bar(propertyAverages.index, propertyAverages.values)
plots[4].set_title('propertyType - Average Price')

fig.tight_layout()
```



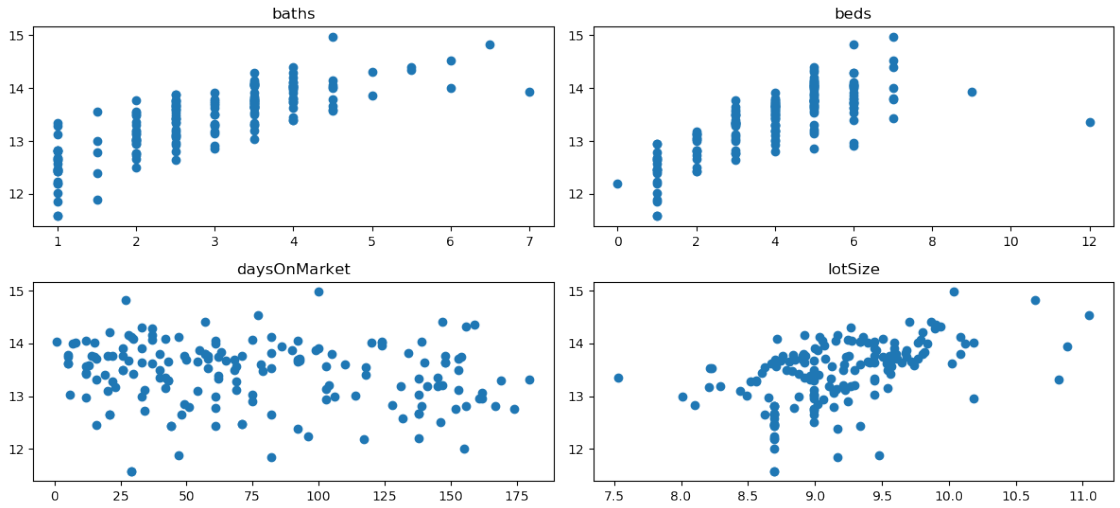
So it appears baths/beds have an expoential relationship with price. Lotsize looks like it might be linear. daysonMarket looks completely random, and we may not even want to add it to the model. Property type shows some variation between average prices, so we will add it to the model. We need to dummy encode the property type. We'll make Condo/Co-op the base case.

[26]:
```
dummies = pd.get_dummies(df.propertyType, drop_first=True)
```

I also want to quickly check to see the relationship if I log transform our Y variable. I'm going to also log transform the lot size as the relationship does not appear to be exponential with tour dependent variable

[27]:
```
fig = plt.figure(figsize=(12,8), dpi=100)

y = np.log(df.price)

plots = []
plotColumns = ['baths', 'beds', 'daysOnMarket']

for i in range(1, 4):
    plots.append(fig.add_subplot(3, 2, i))
    plots[i - 1].scatter(df[plotColumns[i - 1]], y)
    plots[i - 1].set_title(plotColumns[i - 1])
```

```
plots.append(fig.add_subplot(3, 2, 4))
plots[3].scatter(np.log(df.lotSize), y)
plots[3].set_title('lotSize')

fig.tight_layout()
```



Baths and beds appear to have a nice relationship with ln(price). DaysOnMarket is clear that there's no clear pattern. We'll try our model with lot size and see if it adds any value.

5 Creating the Final Model

We'll first create our initial model as a base with our 4 numeric columns and 1 dummy column

[28]:
```
y = df.price
x = sm.add_constant(pd.concat((df[['baths', 'beds', 'lotSize',␣
 ↪'daysOnMarket']], dummies), axis='columns'))
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

```
c:\users\crique\appdata\local\programs\python\python38\lib\site-
packages\numpy\core\fromnumeric.py:2495: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[28]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:                  price   R-squared:                       0.649
Model:                            OLS   Adj. R-squared:                  0.634
Method:                 Least Squares   F-statistic:                     43.12
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           5.32e-34
Time:                        19:26:47   Log-Likelihood:                -2380.5
No. Observations:                 171   AIC:                             4777.
Df Residuals:                     163   BIC:                             4802.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
=============
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-------------
const                     -6.153e+04   7.53e+04     -0.817      0.415
-2.1e+05    8.72e+04
baths                      1.692e+05   2.97e+04      5.692      0.000
1.11e+05    2.28e+05
beds                       7.499e+04   1.87e+04      4.012      0.000
3.81e+04    1.12e+05
lotSize                       5.9285      3.366      1.762      0.080
-0.717      12.574
daysOnMarket               -378.4360    439.824     -0.860      0.391
-1246.923     490.051
Multi-Family (2-4 Unit)    -1.85e+05   1.14e+05     -1.618      0.108
-4.11e+05    4.08e+04
Single Family Residential  5.324e+04   7.29e+04      0.730      0.466
-9.08e+04    1.97e+05
Townhouse                 -1.599e+05   2.05e+05     -0.780      0.437
-5.65e+05    2.45e+05
==============================================================================
Omnibus:                       97.014   Durbin-Watson:                   2.199
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1190.716
Skew:                           1.759   Prob(JB):                    2.75e-259
Kurtosis:                      15.439   Cond. No.                     1.37e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.37e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

For initial model we can see we have a decent model. But it looks like there's some things we can clean up. Here lotSize, daysOnMarket, and property type would all be removed at the 95% confidence level. Let's log transform our dependent variable, log transofrm lotsize, and remove daysOnMarket

[29]:
```
y = np.log(df.price)
x = sm.add_constant(pd.concat((df[['baths', 'beds']], dummies, df.lotSize),␣
 ↪axis='columns'))
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

```
c:\users\crique\appdata\local\programs\python\python38\lib\site-
packages\numpy\core\fromnumeric.py:2495: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[29]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.738
Model:                            OLS   Adj. R-squared:                  0.729
Method:                 Least Squares   F-statistic:                     77.05
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           3.67e-45
Time:                        19:26:47   Log-Likelihood:                -42.821
No. Observations:                 171   AIC:                             99.64
Df Residuals:                     164   BIC:                             121.6
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
=============
                                 coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-------------
const                          12.0758      0.074    164.163      0.000
11.931      12.221
baths                           0.2421      0.034      7.120      0.000
0.175       0.309
beds                            0.1258      0.022      5.840      0.000
0.083       0.168
Multi-Family (2-4 Unit)        -0.0750      0.132     -0.569      0.570
-0.335       0.185
Single Family Residential       0.2327      0.084      2.771      0.006
0.067       0.399
Townhouse                       0.0455      0.236      0.193      0.847
-0.421       0.511
```

```
lotSize                    -3.394e-06   3.83e-06     -0.886      0.377
-1.1e-05    4.17e-06
==============================================================================
Omnibus:                       23.454   Durbin-Watson:                   2.071
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               29.567
Skew:                          -0.866   Prob(JB):                     3.80e-07
Kurtosis:                       4.073   Cond. No.                     1.37e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.37e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

It looks like multi-family amd townhouse add no value to our model, so we'll remove them. Basically the model we'll now see if the property type is a single family residential home or not. We tried lot size with a log and no log transformation and it did not add any value so we'll remove it as well. Now we'll create the final model.

[30]:
```
y = np.log(df.price)
x = sm.add_constant(pd.concat((df[['baths', 'beds']], dummies['Single Family␣
 ↪Residential']), axis='columns'))
model = sm.OLS(y, x)
res = model.fit()
res.summary()
```

```
c:\users\crique\appdata\local\programs\python\python38\lib\site-
packages\numpy\core\fromnumeric.py:2495: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[30]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.736
Model:                            OLS   Adj. R-squared:                  0.732
Method:                 Least Squares   F-statistic:                     155.5
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           3.97e-48
Time:                        19:26:47   Log-Likelihood:                -43.386
No. Observations:                 171   AIC:                             94.77
Df Residuals:                     167   BIC:                             107.3
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
```

```
=============
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
-------------
const                         12.0637      0.068    177.047      0.000
11.929      12.198
baths                          0.2318      0.029      8.027      0.000
0.175       0.289
beds                           0.1210      0.019      6.490      0.000
0.084       0.158
Single Family Residential      0.2559      0.064      3.990      0.000
0.129       0.382
==============================================================================
Omnibus:                       27.432   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               37.587
Skew:                          -0.933   Prob(JB):                     6.89e-09
Kurtosis:                       4.340   Cond. No.                         17.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

So reading our final model we can say that:

beds: For each additional bed, we'll see a 23% increase in our price estimate.

baths: For each additional bath, we'll see 12% increase in our price estimate.

propertyType: If the property type is a single family residential home, we'll see a 25.59% increase in our price estimate.

6 Interpretation and Prediction

Final Model:

ln(price) = 12.0637 + (baths * .2318) + (beds * .1210) + (Single Family Residential * .2559)

To make a predictions we can make sense of we'll have to exp the right side of our equation and find our model SE:

price = exp(12.0637 + (baths * .2318) + (beds * .1210) + (Single Family Residential * .2559) + se^2 / 2)

Let's calculate our standard error

[31]:
```
predictions = res.predict(x.values)
se = np.sqrt(sum((y - predictions) ** 2) / (len(df) - (len(x.columns) - 1) - 1))
'{0:.4f}'.format(se)
```

[31]: '0.3156'

Here we can see our standard error is .3155 so now we can make our final predictions using this.

[32]:
```
finalPredictions = np.exp(12.0637 + (df.baths * .2318) + (df.beds * .1210) +
                          ((df.propertyType == 'Single Family Residential') * .
 ↪2559) + (se**2 / 2))
formattedPredictions = []
for prediction in finalPredictions:
    formattedPredictions.append('${:,.0f}'.format(prediction))

df['finalPredictions'] = finalPredictions
df['formattedPredictions'] = formattedPredictions
df['Residuals'] = df.price - df.finalPredictions
```

Now we can look at our final results.

[33]:
```
finalColumns = ['price', 'propertyType', 'beds', 'baths',␣
 ↪'formattedPredictions', 'Residuals']
df.head(10)[finalColumns]
```

[33]:

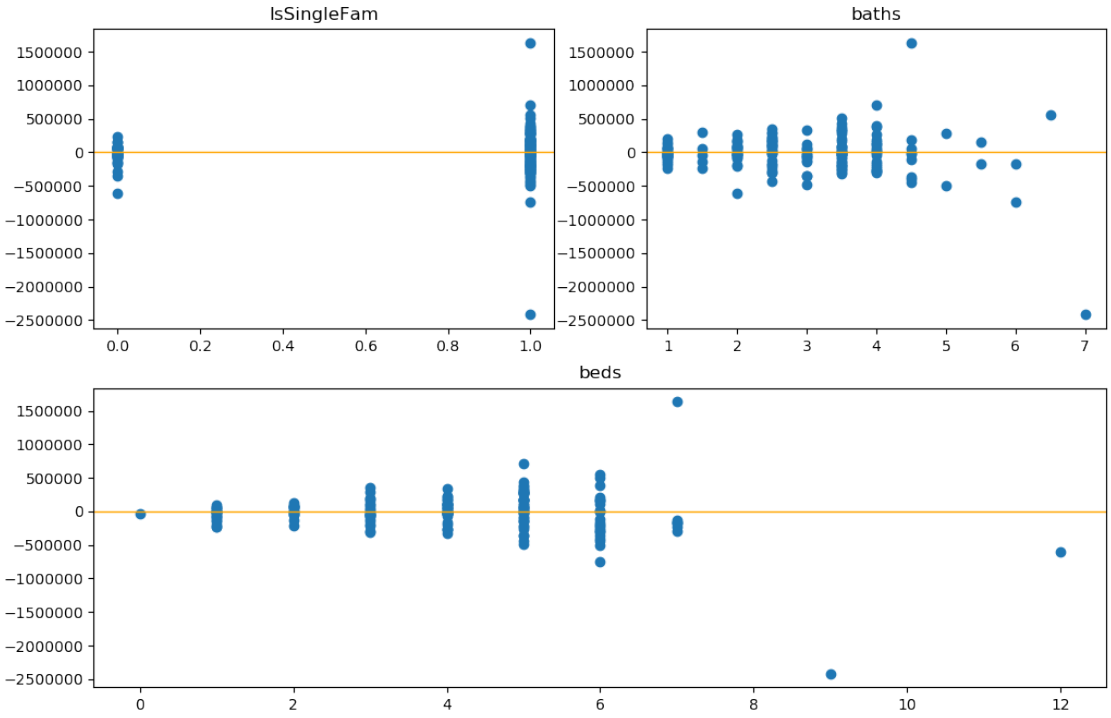
	price	propertyType	beds	baths	formattedPredictions	Residuals
0	902000	Single Family Residential	4.00	3.50	$860,003	41996.82
1	879000	Single Family Residential	4.00	3.00	$765,888	113111.89
2	801000	Single Family Residential	4.00	2.50	$682,073	118927.41
3	906750	Single Family Residential	5.00	3.00	$864,400	42349.60
4	3195000	Single Family Residential	7.00	4.50	$1,558,901	1636099.43
5	414000	Single Family Residential	4.00	2.50	$682,073	-268072.59
6	851000	Single Family Residential	3.00	3.50	$761,992	89007.97
7	1280000	Single Family Residential	5.00	3.50	$970,621	309378.98
8	730000	Single Family Residential	4.00	3.00	$765,888	
9	1060000	Single Family Residential	5.00	2.50	$769,804	

```
8  -35888.11
9  290195.88
```

There are some houses that do seem to have some pretty close estimates, but there's also many that are pretty far off. Row 4 for example is almost $1.5 million off. There's obviously crucial details we're missing such as house condition, location of property (Is it on a busy street?), and many other details. We think it's a pretty good start though in predicting a houses value. Let's check on the residual plots to confirm our findings.

6.1 Residual Plots

[34]:
```
fig = plt.figure(figsize=(12, 8), dpi=100)

plotTitles = ['IsSingleFam', 'baths', 'beds']

df['IsSingleFam'] = df.propertyType == 'Single Family Residential'

plots = []

plots.append(fig.add_subplot(2,2,1))
plots.append(fig.add_subplot(2,2,2))
plots.append(fig.add_subplot(2,2,(3,4)))

bedScatter = plots[2].scatter(df.beds, df.Residuals)
bathScatter = plots[1].scatter(df.baths, df.Residuals)
propertyScatter = plots[0].scatter(df.IsSingleFam, df.Residuals)

for plot, title in zip(plots, plotTitles):
    plot.set_title(title)
    xmin, xmax = plot.get_xlim()
    plot.add_line(Line2D((xmin, xmax), (0, 0), linewidth=1, color='orange'))
```



It definitely looks like we have a heteroskedasticity issue with our beds variable. As the number of beds increases, the wider the range of errors becomes. This is most likely due to the fact that when house grows to be that size, there's so much more information that needs to go into predicting a house than just what we have for our model. If we also go back to our original scatter plot you can also see that there's a wider variation for house prices the higher the number beds. Baths looks okay for a model, and surprisngly, the 0 group for IsSingleFam has less variation than the single family homes, even though this group encompasses different types of houses.

6.2 Confidence Interval

Now we're first going to take a look at the confidence interval for our first value in the above table where beds:

beds = 4, baths = 3.5, and Single Family Residential = 1

[35]:
```
beds = 4
baths = 3.5
SFR = 1

y = df.price
x = sm.add_constant(pd.concat((df.beds - beds, df.baths - baths, (df.
 ↪propertyType == 'Single Family Residential') - 1), axis='columns'))
model = sm.OLS(y, x)
```

```
res = model.fit()
finalSummary = res.summary()
finalSummary
```

```
c:\users\crique\appdata\local\programs\python\python38\lib\site-
packages\numpy\core\fromnumeric.py:2495: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[35]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.634
Model:                            OLS   Adj. R-squared:                  0.627
Method:                 Least Squares   F-statistic:                     96.36
Date:                Fri, 16 Oct 2020   Prob (F-statistic):           3.06e-36
Time:                        19:26:48   Log-Likelihood:                -2384.2
No. Observations:                 171   AIC:                             4776.
Df Residuals:                     167   BIC:                             4789.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       9.485e+05   2.76e+04     34.389      0.000    8.94e+05       1e+06
beds        5.838e+04   1.64e+04      3.553      0.000    2.59e+04    9.08e+04
baths        2.03e+05   2.54e+04      7.980      0.000    1.53e+05    2.53e+05
propertyType 1.392e+05   5.65e+04      2.464      0.015    2.77e+04    2.51e+05
==============================================================================
Omnibus:                      105.246   Durbin-Watson:                   2.204
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1213.676
Skew:                           2.002   Prob(JB):                    2.84e-264
Kurtosis:                      15.422   Cond. No.                         5.70
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

Our final Confidence Interval at the 95% confidence level

[36]:
```
lower, upper = res.conf_int(alpha=.05, cols=None).loc['const'].values
print('Lower: ${0:,.0f}'.format(lower))
print('Upper: ${0:,.0f}'.format(upper))
```

```
Lower: $894,008
```

```
Upper: $1,002,910
```

Now we'll find the prediction interval

[37]:
```
yse = float('2.76e+04')
se = np.sqrt(sum((df.price - finalPredictions) ** 2) / (len(df) - len(x.
 ↪columns) - 1 - 1))
variation = stats.t.ppf(.975, (len(df) - len(x.columns) - 1 - 1)) * np.sqrt(yse␣
 ↪** 2 + se ** 2)
```

[38]:
```
pointEstimate = 860003
lower = pointEstimate - variation
upper = pointEstimate + variation
print('Lower: ${0:,.0f}'.format(lower))
print('Upper: ${0:,.0f}'.format(upper))
```

```
Lower: $234,796
Upper: $1,485,210
```

Because the point estimate factors in the standard error of the estimate, we get a much broader range on our prediction interval.

7 Final Thoughts

7.1 Final Model :

ln(price) = 12.0637 + (baths * .2318) + (beds * .1210) + (Single Family Residential * .2559)

7.2 Top 10 Closest Predictions

[39]:
```
finalColumns = ['propertyType', 'beds', 'baths', 'formattedPrice',␣
 ↪'formattedPredictions', 'formattedResiduals']
df['formattedPrice'] = ['${0:,.0f}'.format(price) for price in df.price]
df['formattedResiduals'] = ['${0:,.0f}'.format(residual) for residual in df.
 ↪Residuals]
closestTen = abs(df.Residuals).sort_values()[:10].index
df.iloc[closestTen,:][finalColumns].head(10)
```

[39]:

	propertyType	beds	baths	formattedPrice
94	Single Family Residential	4.00	3.50	$860,000
104	Single Family Residential	1.00	2.00	$422,000
155	Condo/Co-op	1.00	1.00	$260,000
133	Condo/Co-op	1.00	1.00	$258,000
11	Single Family Residential	5.00	4.50	$1,220,000
19	Single Family Residential	2.00	2.50	$531,000
132	Single Family Residential	6.00	3.50	$1,100,000
89	Single Family Residential	6.00	4.00	$1,225,000

```
99   Single Family Residential  2.00   1.00       $371,900
83   Single Family Residential  4.00   2.50       $675,000

    formattedPredictions formattedResiduals
94              $860,003                $-3
104             $422,520              $-520
155             $259,443               $557
133             $259,443            $-1,443
11            $1,223,825            $-3,825
19              $535,465            $-4,465
132           $1,095,467             $4,533
89            $1,230,082            $-5,082
99              $378,205            $-6,305
83              $682,073            $-7,073
```

7.3 Top 10 Furthest Predictions

[40]:
```
worstTen = abs(df.Residuals).sort_values(ascending=False)[:10].index
df.iloc[worstTen,:][finalColumns].head(10)
```

[40]:
```
                  propertyType  beds  baths formattedPrice  \
90   Single Family Residential  9.00   7.00     $1,126,000
4    Single Family Residential  7.00   4.50     $3,195,000
100  Single Family Residential  6.00   6.00     $1,215,000
55   Single Family Residential  5.00   4.00     $1,800,000
160    Multi-Family (2-4 Unit) 12.00   2.00       $630,000
26   Single Family Residential  6.00   6.50     $2,750,000
12   Single Family Residential  6.00   3.50     $1,605,000
38   Single Family Residential  6.00   5.00     $1,054,000
17   Single Family Residential  5.00   3.00       $380,000
131  Single Family Residential  5.00   4.50       $780,000

    formattedPredictions formattedResiduals
90            $3,544,797        $-2,418,797
4             $1,558,901         $1,636,099
100           $1,955,568           $-740,568
55            $1,089,894            $710,106
160           $1,238,104           $-608,104
26            $2,195,875            $554,125
12            $1,095,467            $509,533
38            $1,550,970           $-496,970
17              $864,400           $-484,400
131           $1,223,825           $-443,825
```

As we see here the model generally had a harder time with the bigger houses.

Going forward if we wanted to turn this into a more realistic model, we'd want to include many other variables. A house that has 2 beds / 2 baths, but was updated recently, is going to be priced much differently than a similar house that hasn't been touched in 30 years. Overall, we feel like this project was a good start in understanding the relationships between key features and what determines prices of houses. It also gave us a better understanding of the key aspects that go into build regression models.