case_6.4

May 29, 2020

1 How much should properties be worth in Milwaukee, Wisconsin? (Part II)

1.1 Introduction (5 mts)

Business Context. You are the same property developer from the previous case, with the same goal. Although the previous model you built was a good start, it did not incorporate all the variables you wished to include, and you are skeptical of how well it might work on data that it was not trained on; i.e. **out-of-sample data**.

Business Problem. Your task is to **improve your model to predict property prices in the city of Milwaukee, Wisconsin**.

Analytical Context. Again, the dataset consists of property sales (commercial and residential) in Milwaukee, Wisconsin from 2002 to 2018. focused on diagnosing and fixing possible issues arising in linear regression. In the previous case, We learned how to visually analyze residuals and to detect outliers and heteroscedasticity. We showed that variable transformation can improve some of these issues, while explicit removal of outliers *explainable by external factors* could improve things further.

This case is structured as follows: you will 1) learn how categorical variables are actually handled in regression models; 2) dive into the predictive ability of the model and learn how to assess and improve it; and finally 3) look at **collinearity**, an issue that arises when fitting regression models with highly correlated or too many predictors and how to deal with it.

1.2 Preparing our data (5 mts)

We will pick up were we left things off at the end of the last case. We will load the same packages and data, and fit the same model. The one difference is that we will remove from the dataset all properties with a sale price below $2,000, which may not correspond to real market prices, per our analysis of residuals in the last exercise of the previous case:

[43]:
```
### Load relevant packages
import pandas                 as pd
import numpy                  as np
import matplotlib.pyplot      as plt
import seaborn                as sns
import statsmodels.formula.api as smf
```

```
import statsmodels.api         as sm
import scipy

%matplotlib inline
plt.style.use('ggplot')
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

[44]:
```
data = pd.read_csv("2002-2018-property-sales-data.csv",
    dtype = { # indicate categorical variables
        "PropType": "category",
        "District": "category",
        "Extwall": "category",
        "Nbhd": "category",
        "Style": "category",
    },
    parse_dates=["Sale_date"], # the Sale_date column is parsed as a date
)
def remove_unused_categories(data):
    """ The `remove_unused_categories` method in pandas
        removes categories from a Series if there are no
        elements of that category.

        This function is a convenience function that removes
        unused categories for all categorical columns
        of a data frame.

        The reason this is useful is that when we
        fit a linear regression, `statsmodels` will
        create a coefficient for every category in a column,
        and so unused categories pollute the results.
    """
    for cname in data:
        col = data[cname]
        if pd.api.types.is_categorical_dtype(col):
            data[cname] = col.cat.remove_unused_categories()
    return data

clean = np.where(
    (data["Sale_price"] > 2000) & # this is the only change!
    (data["Year_Built"] > 1800) &
    (data["Fin_sqft"] > 0) & # must have non-zero finished square feet
    (data["Lotsize"] > 0)  & # must have non-zero lot size
    (data["PropType"] == "Residential")
    )
data_clean = data.iloc[clean].copy()
remove_unused_categories(data_clean).head()
```

[44]:

	PropType	Taxkey	Address	CondoProject	District	Nbhd
10	Residential	3080013000	3033 N 35TH ST	NaN	7	2960
51	Residential	3190434000	1908 E WEBSTER PL	NaN	3	3170
67	Residential	3891722000	812 N 25TH ST	NaN	4	3040
116	Residential	3880628000	959 N 34TH ST	NaN	4	2300
134	Residential	3880406000	3209 W WELLS ST	NaN	4	2300

	Style	Extwall	Stories	Year_Built	Nr_of_rms	Fin_sqft
10	AP 1	Frame	2.0	1913	0	3476
51	Rm or Rooming House	Frame	2.0	1897	0	1992
67	Rm or Rooming House	Frame	2.0	1907	0	2339
116	AP 1	Frame	2.0	1890	0	2329
134	Mansion	Stone	2.5	1891	0	7450

	Units	Bdrms	Fbath	Hbath	Lotsize	Sale_date	Sale_price
10	4	9	1	0	5040	2002-02-01	42000
51	4	2	2	0	2880	2002-05-01	145000
67	6	0	1	0	3185	2002-06-01	30000
116	4	4	1	0	5781	2002-10-01	66500
134	2	7	6	0	15600	2002-11-01	150500

1.3 Training vs. test sets (25 mts)

As we discussed in the introduction, one of our chief concerns is whether or not the model we have built works just as well on out-of-sample data as it does on in-sample data. This is a *very* common problem in model-building, as is known as **overfitting**. You will learn more about overfitting in future cases, but here we will discuss a simple method for mitigating it.

The idea is to randomly split the data into a training set and a test set. The **training set** is the one on which we train and fit our multiple linear regression model. We then run our fitted model on the **test set** and compared its predictions against the actual test set response variable data to evaluate its performance.

1.3.1 Exercise 1: (10 mts)

Write code to split the data into training and test sets (an 80-20 split is a good starting point). Fit a linear regression model on the training set, with the logarithm of `Sale_price` as the response variable and `District`, `Units`, and the logarithm of `Fin_sqft` as the predictor variables.

Answer. First we set the *random seed*, which ensures that everyone who runs this notebook will get the same random numbers, and therefore the same results in the proceeding analysis. We then randomly choose 80% of the indices between 1 and the number of rows in the data. Extracting these rows from the data yields the training set, while all other rows form the test set:

[45]:
```
np.random.seed(2019) # a seed makes the analysis reproducible
                     # so everyone will get the same results
ndata = len(data_clean)
```

```
# Randomly choose 0.8n indices between 1 and n
idx_train = np.random.choice(range(ndata),int(0.8*ndata),replace=False)
# The test set is comprised from all the indices that were
# not selected in the training set:
idx_test  = np.asarray(list(set(range(ndata)) - set(idx_train)))
train     = data_clean.iloc[idx_train] # the training data set
test      = data_clean.iloc[idx_test]  # the test data set
print(train.shape) # 19,312 rows and 19 columns
print(test.shape)  #  4,829 rows and 19 columns
```

```
(19556, 19)
(4889, 19)
```

[46]:
```
model_log = smf.ols(formula = "np.log(Sale_price) ~ District + Units"
                            "+ np.log(Fin_sqft)",
                    data = train).fit()
model_log.summary()
```

[46]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(Sale_price)   R-squared:                       0.608
Model:                            OLS   Adj. R-squared:                  0.608
Method:                 Least Squares   F-statistic:                     1894.
Date:                Fri, 15 Nov 2019   Prob (F-statistic):               0.00
Time:                        08:00:20   Log-Likelihood:                -7614.7
No. Observations:               19556   AIC:                         1.526e+04
Df Residuals:                   19539   BIC:                         1.540e+04
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      5.2460      0.063     83.895      0.000       5.123       5.369
District[T.10] 0.6199      0.015     42.085      0.000       0.591       0.649
District[T.11] 0.8121      0.014     57.327      0.000       0.784       0.840
District[T.12] -0.0162     0.023     -0.704      0.482      -0.061       0.029
District[T.13] 0.7927      0.015     54.029      0.000       0.764       0.821
```

```
District[T.14]       0.8558      0.015     58.297      0.000       0.827
0.885
District[T.15]      -0.4857      0.022    -22.245      0.000      -0.528
-0.443
District[T.2]        0.3082      0.016     18.793      0.000       0.276
0.340
District[T.3]        1.0123      0.017     60.808      0.000       0.980
1.045
District[T.4]       -0.2320      0.034     -6.853      0.000      -0.298
-0.166
District[T.5]        0.6815      0.014     48.208      0.000       0.654
0.709
District[T.6]       -0.1870      0.020     -9.322      0.000      -0.226
-0.148
District[T.7]       -0.0817      0.018     -4.566      0.000      -0.117
-0.047
District[T.8]        0.1358      0.019      7.149      0.000       0.099
0.173
District[T.9]        0.5709      0.017     33.096      0.000       0.537
0.605
Units               -0.3476      0.007    -50.304      0.000      -0.361
-0.334
np.log(Fin_sqft)     0.8677      0.009     95.361      0.000       0.850
0.886
==============================================================================
Omnibus:                     4333.113   Durbin-Watson:                   2.009
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            26728.932
Skew:                          -0.923   Prob(JB):                         0.00
Kurtosis:                       8.422   Cond. No.                         185.
==============================================================================

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

1.3.2 Categorical features under the hood (10 mts)

You may have noticed that there are over a dozen coefficients for `District` above. This is because `District` is a categorical variable, and for categorical features, one coefficient is obtained for all but one of the categories. If there are only two categories (e.g. gender), this is intuitive: the feature is converted into a column of zeros and ones before feeding it into the regression, where it is treated like a regular numerical variable.

When there are more than two categories, one category is designated the "reference" or "baseline" category, and "dummy" columns of ones and zeros are created for all other categories. Let's take a dummy example with three categories and five rows:

Dummy
A
B
C
C
A

Before the linear regression is fitted, the `Dummy` column is transformed into **two** "dummy" columns. The first column is 1 if the district is `B`, and 0 otherwise, whereas the second column is 1 if the district is `C`, and 0 otherwise. We get:

B	C
0	0
1	0
0	1
0	1
0	0

The dummy columns are fed into the linear regression and treated like regular numerical variables. This technique is called **one-hot encoding** and can be done manually with the pandas function `pd.get_dummies()`.

Just like in the binary case, the choice of baseline changes the coefficients and their interpretation; the `District[T.3]` coefficient of 0.9797 should be interpreted as the difference in predicted outcomes between districts 3 and 1. But the baseline choice does not affect predictions and model performance, which is why most software will arbitrarily pick one category as the baseline without warning (often the first alphabetically).

1.3.3 Exercise 2: (5 mts)

Add `Style` to our last regression model. According to this model, which property style is the most desirable?

Answer. One possible solution is given below:

[47]:
```
model_style = smf.ols(formula = "np.log(Sale_price) ~ Style + District + Units"
                              "+ np.log(Fin_sqft)",
                      data = train).fit()
model_style.summary()
```

[47]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(Sale_price)   R-squared:                       0.638
```

Model:	OLS	Adj. R-squared:	0.637
Method:	Least Squares	F-statistic:	1041.
Date:	Fri, 15 Nov 2019	Prob (F-statistic):	0.00
Time:	08:01:23	Log-Likelihood:	-6843.8
No. Observations:	19556	AIC:	1.376e+04
Df Residuals:	19522	BIC:	1.402e+04
Df Model:	33		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4.4567	0.166	26.829	0.000	4.131	4.782
Style[T.Bi-Level]	0.1019	0.133	0.767	0.443	-0.159	0.363
Style[T.Cape Cod]	0.2236	0.124	1.797	0.072	-0.020	0.467
Style[T.Colonial]	0.2869	0.125	2.304	0.021	0.043	0.531
Style[T.Cottage]	-0.0827	0.126	-0.658	0.510	-0.329	0.164
Style[T.Dplx Bungalow]	-0.2505	0.105	-2.376	0.018	-0.457	-0.044
Style[T.Duplex N/S]	-0.1309	0.106	-1.239	0.215	-0.338	0.076
Style[T.Duplex O/S]	-0.3237	0.105	-3.074	0.002	-0.530	-0.117
Style[T.Duplex-Cottage]	-0.5049	0.109	-4.630	0.000	-0.719	-0.291
Style[T.Mansion]	0.2897	0.132	2.191	0.028	0.031	0.549
Style[T.Milwaukee Bungalow]	0.0743	0.124	0.597	0.550	-0.170	0.318
Style[T.Ranch]	0.2899	0.124	2.329	0.020	0.046	0.534
Style[T.Residence O/S]	0.0438	0.124	0.353	0.724	-0.200	0.287
Style[T.Rm or Rooming House]	-0.2530	0.166	-1.529	0.126	-0.577	0.071
Style[T.Split Level]	0.1646	0.128	1.283	0.199	-0.087	0.416
Style[T.Townhouse]	-0.1553	0.108	-1.434	0.152	-0.367	0.057
Style[T.Triplex]	-0.3130	0.095	-3.309	0.001		

```
-0.498      -0.128
Style[T.Tudor]                  0.3118      0.127      2.460      0.014
0.063       0.560
District[T.10]                  0.6644      0.014     46.331      0.000
0.636       0.693
District[T.11]                  0.7570      0.014     54.953      0.000
0.730       0.784
District[T.12]                  0.1557      0.024      6.607      0.000
0.109       0.202
District[T.13]                  0.7520      0.014     53.007      0.000
0.724       0.780
District[T.14]                  0.9364      0.014     64.880      0.000
0.908       0.965
District[T.15]                 -0.3955      0.021    -18.463      0.000
-0.437      -0.354
District[T.2]                   0.2522      0.016     15.801      0.000
0.221       0.284
District[T.3]                   1.1061      0.017     65.719      0.000
1.073       1.139
District[T.4]                  -0.1960      0.033     -5.901      0.000
-0.261      -0.131
District[T.5]                   0.6250      0.014     45.333      0.000
0.598       0.652
District[T.6]                  -0.0513      0.020     -2.562      0.010
-0.091      -0.012
District[T.7]                  -0.0584      0.017     -3.374      0.001
-0.092      -0.024
District[T.8]                   0.2757      0.019     14.524      0.000
0.238       0.313
District[T.9]                   0.4946      0.017     29.193      0.000
0.461       0.528
Units                          -0.0415      0.030     -1.394      0.163
-0.100       0.017
np.log(Fin_sqft)                0.9082      0.011     85.881      0.000
0.888       0.929
==============================================================================
Omnibus:                     4282.563   Durbin-Watson:                   2.008
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            23159.718
Skew:                          -0.950   Prob(JB):                         0.00
Kurtosis:                       7.981   Cond. No.                     1.47e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.47e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

"""

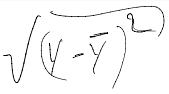
The mansion style is the most desirable according to this model, since it has the highest positive coefficient.

1.4 Evaluating predictive performance (30 mts)

Recall that for any given data point, the residual is the difference between our model's prediction for that point and the actual value of that point. The **mean absolute error (MAE)** is a metric which summarizes the model's holistic performance on the entire dataset. The MAE is calculated by taking the absolute value of each residual, then taking the mean of all of those absolute values. In essence, the MAE describes the typical magnitude of the residuals; the lower the MAE, the better.

[28]:
```
np.round(
    np.mean(                                                    # Mean
        np.abs(                                                 # Absolute
            model_log.predict(test) - np.log(test.Sale_price)   # Error
        )
    ),
2) # round to two digits
```

[28]: 0.26



Another very popular choice for evaluating regression is the **root mean squared error (RMSE)**. This is computed by taking the square of each residual, averaging them, and then taking the square root. Below is the implementation in python, which is possibly easier to understand than the above explanation:

[29]:
```
np.round(
    np.sqrt(                                                    # Root
        np.mean(                                                # Mean
            np.square(                                          # Squared
                model_log.predict(test) - np.log(test.Sale_price)   # Error
            )
        )
    ),
    decimals=3 # round to 3 decimals
)
```

[29]: 0.367

Finally, there is the **mean absolute percentage error**. This takes the absolute value of each residual and divides it by the actual value of that point to obtain a percentage, then averaging across all percentages.

[30]:
```
np.round(
    np.mean(                                    # Mean
        np.abs(                                 # Absolute
            np.exp(model_log.predict(test))
            - test.Sale_price                   # Error
            )
        * 100 / test.Sale_price                 # Percentage
    ),
    decimals=1 # round to 1 decimal
)
```

[30]: 29.5

1.4.1 Question: (5 mts)

Why would we not just use MAE instead of having to deal with three different metrics?

Although RMSE may seem needlessly complicated compared to MAE, it is much more commonly used. The reason is that the RMSE metric is the same as the one being minimized on the training data by a standard linear regression model (also called an **ordinary least squares (OLS)** regression). This makes it a "natural" choice using the same metric to evaluate the out-of-sample (test) performance.

Furthermore, the RMSE puts much more weight on outliers, since the errors are squared before being averaged. In cases where outliers are especially bad and need to be punished, the RMSE is a better choice.

The MAPE has a nice interpretation in that we can say that a model's predictions are on average wrong by a certain percentage. For example, our house price model could be on average off by 25.4%. Such a model is often described as being "74.6% accurate".

For those who are mathematically inclined, the equations below define the three error metrics above:

$$\begin{aligned} MAE &= \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \\ RMSE &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \\ MAPE &= 100\%\frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \end{aligned}$$

1.4.2 Exercise 3: (10 mts)

3.1 Compute the MAE for the first model (`model_lin` without the log transform). Can you compare the MAEs of the two models directly?

Answer. Shown below:

[48]:
```
model_lin = smf.ols(formula = "Sale_price ~ District + Units+ Fin_sqft",
                    data = train).fit()
np.round(
    np.mean(                                                    # Mean
        np.abs(                                                 # Absolute
            model_lin.predict(test) - test.Sale_price           # Error
        )
    ))
```

[48]:
```
31777.0
```

No, the metric is different if the error is computed on the sale price or its logarithm. To compare the two models, we need to either take the logarithm of the predictions of `model_lin`, or exponentiate the predictions of `model_log`.

[49]:
```
np.round(
    np.mean(                                                    # Mean
        np.abs(                                                 #␣
 ↪Absolute
            np.log(model_lin.predict(test)) - np.log(test.Sale_price)     #␣
 ↪Error
        )
    ),
2) # round to two digits
```

```
/Users/haris.jaliawala/anaconda3/envs/w6/lib/python3.7/site-
packages/pandas/core/series.py:856: RuntimeWarning: invalid value encountered in
log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[49]:
```
0.27
```

But there's a problem! The model for the untransformed model sometimes predicts negative prices. This is nonsensical, as there are no negative prices in the original data. When this happens, the logarithm is undefined, which prompts the python warning above (the invalid values are then ignored when computing the mean, which is highly misleading). This gives a further argument for modeling the logarithm of sale price:

[50]:
```
np.round(
    np.mean(                                                    # Mean
        np.abs(                                                 # Absolute
            np.exp(model_log.predict(test)) - test.Sale_price     # Error
        )
    ))
```

[50]:
```
30235.0
```

There is no such problem when the predictions from the second model are exponentiated, and we

also find that the MAE is lower (which is better).

3.2 Compute the RMSE, MAE and MAPE for the two sets of errors below.

[51]:
```
errors_outlier = np.array([0.1, -0.1, 0.2, 1.2])
errors_nooutly = np.array([0.4, -0.4, 0.5, 0.7])
true_values    = np.array([2.0,  1.8, 2.3, 2.8])
```

Answer. The RMSE more heavily penalizes the errors with outliers:

[52]:
```
print("RMSE with outlier {:5.3f} vs. {:5.3f} without outlier".format(
    np.sqrt(np.mean(np.square(errors_outlier))),
    np.sqrt(np.mean(np.square(errors_nooutly)))))
print(" MAE with outlier {:5.3f} vs. {:5.3f} without outlier".format(
    np.mean(np.abs(errors_outlier)),
    np.mean(np.abs(errors_nooutly))))
print("MAPE with outlier {:4.1f}% vs. {:4.1f}% without outlier".format(
    np.mean(np.abs(errors_outlier)/true_values)*100,
    np.mean(np.abs(errors_nooutly)/true_values)*100))
```

```
RMSE with outlier 0.612 vs. 0.515 without outlier
 MAE with outlier 0.400 vs. 0.500 without outlier
MAPE with outlier 15.5% vs. 22.2% without outlier
```

1.5 Adding additional predictors to improve our model (30 mts)

With large datasets, the predictive performance of a model can often be improved by adding more predictors. `model_feat` below adds features for the number of stories and bedrooms, the year built, neighborhood (instead of simply district), type of external wall, number of full and half baths, lot size and sale year. This last predictor is an example of **feature engineering** – remember we are not limited to the predictors as they are given to us in the raw data. Often, it is beneficial to transform and combine predictors, as we did here by extracting the year from the sale date and treating it as a categorical variable.

1.5.1 Exercise 4: (30 mts)

4.1 Fit a linear regression of the logarithm of the sale price with the following features:

1. number of stories
2. number of bedrooms
3. number of units in the property
4. neighborhood code
5. style
6. type of external wall
7. number of full baths
8. number of half baths
9. the logarithm of square footage

10. the logarithm of the lot size
11. year the property was *built* as a *numerical* variable
12. year the property was *sold* as a *categorical* variable

Give the resulting fitted model the variable name `model_feat`.

(Hints: 1) write a function to extract the year from the sale date – `train.Sale_date.iloc[0].year` gives the year of the first sale; 2) use `C()` around a term in the formula to make the term categorical)

Answer. One possible solution is shown below:

[53]:
```
def year_from_date(dt):
    """ function to extract the year from a Sale date """
    return dt.dt.year

model_feat = smf.ols(formula = "np.log(Sale_price) ~ "
                               "Stories + Year_Built + Nbhd"
                               "+ np.log(Fin_sqft)"
                               "+ Style"
                               "+ Extwall  + Units + Bdrms"
                               "+ Fbath + Hbath + np.log(Lotsize)"
                               "+ C(year_from_date(Sale_date))",
                     data = train).fit()
model_feat.summary()
```

[53]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(Sale_price)   R-squared:                       0.813
Model:                            OLS   Adj. R-squared:                  0.811
Method:                 Least Squares   F-statistic:                     445.7
Date:                Fri, 15 Nov 2019   Prob (F-statistic):               0.00
Time:                        08:07:50   Log-Likelihood:                -372.59
No. Observations:               19555   AIC:                             1125.
Df Residuals:                   19365   BIC:                             2623.
Df Model:                         189
Covariance Type:            nonrobust
================================================================================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
Intercept                        0.9107      0.376      2.422      0.015
0.174       1.648
Nbhd[T.1040]                     0.2860      0.035      8.059      0.000
0.216       0.356
Nbhd[T.1140]                    -0.3183      0.035     -9.003      0.000
```

-0.388	-0.249				
Nbhd[T.1150]		-0.4187	0.041	-10.298	0.000
-0.498	-0.339				
Nbhd[T.1160]		-0.4862	0.044	-10.938	0.000
-0.573	-0.399				
Nbhd[T.1200]		-0.0088	0.044	-0.203	0.839
-0.094	0.077				
Nbhd[T.1220]		0.4315	0.049	8.847	0.000
0.336	0.527				
Nbhd[T.1340]		-0.3423	0.041	-8.267	0.000
-0.423	-0.261				
Nbhd[T.1380]		0.0085	0.039	0.216	0.829
-0.069	0.086				
Nbhd[T.1400]		0.1194	0.035	3.377	0.001
0.050	0.189				
Nbhd[T.1440]		-0.5311	0.043	-12.391	0.000
-0.615	-0.447				
Nbhd[T.1460]		0.2815	0.035	8.119	0.000
0.214	0.349				
Nbhd[T.1470]		0.2154	0.035	6.113	0.000
0.146	0.284				
Nbhd[T.1480]		0.0592	0.035	1.679	0.093
-0.010	0.128				
Nbhd[T.1490]		0.0875	0.037	2.356	0.018
0.015	0.160				
Nbhd[T.1500]		-0.5240	0.048	-10.918	0.000
-0.618	-0.430				
Nbhd[T.1560]		-0.1224	0.048	-2.542	0.011
-0.217	-0.028				
Nbhd[T.1620]		-0.2779	0.040	-6.944	0.000
-0.356	-0.199				
Nbhd[T.1680]		-0.3772	0.042	-9.036	0.000
-0.459	-0.295				
Nbhd[T.1740]		-0.2748	0.044	-6.298	0.000
-0.360	-0.189				
Nbhd[T.1780]		-0.5533	0.047	-11.673	0.000
-0.646	-0.460				
Nbhd[T.1820]		-0.0526	0.042	-1.248	0.212
-0.135	0.030				
Nbhd[T.1830]		0.0743	0.041	1.814	0.070
-0.006	0.155				
Nbhd[T.1840]		0.1581	0.041	3.838	0.000
0.077	0.239				
Nbhd[T.1850]		0.1855	0.039	4.788	0.000
0.110	0.261				
Nbhd[T.1860]		-0.1371	0.041	-3.310	0.001
-0.218	-0.056				

Nbhd[T.1870]	0.1039	0.045	2.334	0.020	
0.017	0.191				
Nbhd[T.1880]	-0.1017	0.093	-1.090	0.276	
-0.284	0.081				
Nbhd[T.1940]	0.1137	0.039	2.947	0.003	
0.038	0.189				
Nbhd[T.1980]	0.6183	0.037	16.899	0.000	
0.547	0.690				
Nbhd[T.2040]	0.3191	0.034	9.421	0.000	
0.253	0.385				
Nbhd[T.2080]	0.5396	0.033	16.493	0.000	
0.475	0.604				
Nbhd[T.2100]	0.4632	0.032	14.270	0.000	
0.400	0.527				
Nbhd[T.2120]	0.5512	0.037	14.925	0.000	
0.479	0.624				
Nbhd[T.2140]	0.5439	0.044	12.411	0.000	
0.458	0.630				
Nbhd[T.2160]	-0.4760	0.053	-9.063	0.000	
-0.579	-0.373				
Nbhd[T.2180]	-0.0581	0.048	-1.219	0.223	
-0.152	0.035				
Nbhd[T.2300]	-0.0762	0.048	-1.593	0.111	
-0.170	0.018				
Nbhd[T.2380]	-0.8188	0.043	-18.986	0.000	
-0.903	-0.734				
Nbhd[T.240]	0.3066	0.035	8.651	0.000	
0.237	0.376				
Nbhd[T.2400]	-0.7856	0.057	-13.781	0.000	
-0.897	-0.674				
Nbhd[T.2420]	-0.7120	0.064	-11.101	0.000	
-0.838	-0.586				
Nbhd[T.2500]	0.1370	0.053	2.594	0.010	
0.033	0.241				
Nbhd[T.2510]	-0.2417	0.044	-5.486	0.000	
-0.328	-0.155				
Nbhd[T.2520]	-0.2642	0.043	-6.191	0.000	
-0.348	-0.181				
Nbhd[T.2540]	0.5370	0.034	15.569	0.000	
0.469	0.605				
Nbhd[T.2580]	0.8181	0.050	16.240	0.000	
0.719	0.917				
Nbhd[T.2600]	0.5765	0.037	15.722	0.000	
0.505	0.648				
Nbhd[T.2610]	0.2349	0.047	4.961	0.000	
0.142	0.328				
Nbhd[T.2660]	0.5680	0.044	13.008	0.000	

```
0.482       0.654
Nbhd[T.2680]                    0.7727      0.044     17.718      0.000
0.687       0.858
Nbhd[T.2700]                    0.4613      0.039     11.716      0.000
0.384       0.539
Nbhd[T.2710]                    0.4031      0.035     11.573      0.000
0.335       0.471
Nbhd[T.2720]                    0.6083      0.035     17.287      0.000
0.539       0.677
Nbhd[T.2760]                    0.5649      0.038     14.723      0.000
0.490       0.640
Nbhd[T.2800]                    0.4003      0.048      8.407      0.000
0.307       0.494
Nbhd[T.2810]                    0.5311      0.045     11.764      0.000
0.443       0.620
Nbhd[T.2820]                   -0.7654      0.065    -11.727      0.000
-0.893      -0.637
Nbhd[T.2840]                   -0.8635      0.049    -17.768      0.000
-0.959      -0.768
Nbhd[T.2850]                   -1.0631      0.052    -20.349      0.000
-1.165      -0.961
Nbhd[T.2860]                   -0.7609      0.046    -16.550      0.000
-0.851      -0.671
Nbhd[T.2870]                   -1.2887      0.054    -23.717      0.000
-1.395      -1.182
Nbhd[T.2880]                    0.6701      0.040     16.851      0.000
0.592       0.748
Nbhd[T.2890]                    0.6543      0.040     16.315      0.000
0.576       0.733
Nbhd[T.2900]                   -0.8344      0.050    -16.743      0.000
-0.932      -0.737
Nbhd[T.2910]                    0.7915      0.047     16.967      0.000
0.700       0.883
Nbhd[T.2920]                   -0.2866      0.046     -6.242      0.000
-0.377      -0.197
Nbhd[T.2930]                   -0.7840      0.050    -15.590      0.000
-0.883      -0.685
Nbhd[T.2940]                   -1.0642      0.051    -20.857      0.000
-1.164      -0.964
Nbhd[T.2950]                   -1.2675      0.044    -29.069      0.000
-1.353      -1.182
Nbhd[T.2960]                   -0.9971      0.050    -19.935      0.000
-1.095      -0.899
Nbhd[T.3000]                   -0.3939      0.047     -8.439      0.000
-0.485      -0.302
Nbhd[T.3040]                   -0.4187      0.045     -9.289      0.000
-0.507      -0.330
```

```
Nbhd[T.3060]                    1.1756      0.036     32.561      0.000
1.105       1.246
Nbhd[T.3150]                    1.0207      0.040     25.462      0.000
0.942       1.099
Nbhd[T.3160]                    1.0833      0.037     29.605      0.000
1.012       1.155
Nbhd[T.3170]                    1.0795      0.041     26.095      0.000
0.998       1.161
Nbhd[T.3190]                    1.0827      0.041     26.662      0.000
1.003       1.162
Nbhd[T.3240]                    1.1444      0.041     28.094      0.000
1.065       1.224
Nbhd[T.3320]                    0.7380      0.039     18.814      0.000
0.661       0.815
Nbhd[T.3330]                    0.8177      0.048     17.119      0.000
0.724       0.911
Nbhd[T.360]                     0.3189      0.040      7.883      0.000
0.240       0.398
Nbhd[T.380]                     0.1685      0.048      3.539      0.000
0.075       0.262
Nbhd[T.40]                      0.1303      0.046      2.813      0.005
0.040       0.221
Nbhd[T.4000]                    0.2206      0.035      6.264      0.000
0.152       0.290
Nbhd[T.4020]                    0.2913      0.050      5.807      0.000
0.193       0.390
Nbhd[T.4040]                   -0.1392      0.055     -2.518      0.012
-0.248      -0.031
Nbhd[T.4050]                   -0.1243      0.045     -2.773      0.006
-0.212      -0.036
Nbhd[T.4060]                    0.0984      0.036      2.743      0.006
0.028       0.169
Nbhd[T.4100]                   -0.1096      0.042     -2.593      0.010
-0.192      -0.027
Nbhd[T.4120]                   -0.1042      0.037     -2.809      0.005
-0.177      -0.032
Nbhd[T.4160]                    0.0317      0.038      0.842      0.400
-0.042       0.105
Nbhd[T.4180]                    0.2174      0.039      5.564      0.000
0.141       0.294
Nbhd[T.4240]                    0.4917      0.033     14.979      0.000
0.427       0.556
Nbhd[T.4260]                    0.4647      0.045     10.296      0.000
0.376       0.553
Nbhd[T.4280]                    0.5167      0.046     11.207      0.000
0.426       0.607
Nbhd[T.4310]                    0.5759      0.039     14.711      0.000
```

0.499	0.653				
Nbhd[T.4320]		0.5361	0.035	15.476	0.000
0.468	0.604				
Nbhd[T.4330]		0.5124	0.037	13.689	0.000
0.439	0.586				
Nbhd[T.4340]		0.5725	0.033	17.438	0.000
0.508	0.637				
Nbhd[T.4350]		0.5220	0.036	14.315	0.000
0.451	0.593				
Nbhd[T.4360]		0.4314	0.036	12.021	0.000
0.361	0.502				
Nbhd[T.4380]		0.4712	0.037	12.803	0.000
0.399	0.543				
Nbhd[T.440]		0.3304	0.038	8.613	0.000
0.255	0.406				
Nbhd[T.4400]		0.4314	0.037	11.541	0.000
0.358	0.505				
Nbhd[T.4410]		0.5446	0.039	14.054	0.000
0.469	0.621				
Nbhd[T.4420]		0.5318	0.033	16.092	0.000
0.467	0.597				
Nbhd[T.4425]		0.4903	0.069	7.099	0.000
0.355	0.626				
Nbhd[T.4430]		0.5818	0.058	10.035	0.000
0.468	0.695				
Nbhd[T.4500]		0.8799	0.038	23.285	0.000
0.806	0.954				
Nbhd[T.4510]		0.9417	0.046	20.394	0.000
0.851	1.032				
Nbhd[T.4520]		0.4150	0.033	12.543	0.000
0.350	0.480				
Nbhd[T.4540]		1.0206	0.037	27.776	0.000
0.949	1.093				
Nbhd[T.4560]		1.1939	0.045	26.653	0.000
1.106	1.282				
Nbhd[T.4580]		0.7400	0.034	21.812	0.000
0.673	0.806				
Nbhd[T.4600]		0.9702	0.037	25.918	0.000
0.897	1.044				
Nbhd[T.4610]		0.8238	0.038	21.705	0.000
0.749	0.898				
Nbhd[T.4620]		0.6390	0.033	19.131	0.000
0.574	0.704				
Nbhd[T.4660]		0.4896	0.035	13.982	0.000
0.421	0.558				
Nbhd[T.4700]		0.5886	0.034	17.378	0.000
0.522	0.655				

Nbhd[T.4720]	0.5769	0.036	15.873	0.000	
0.506	0.648				
Nbhd[T.4740]	0.4990	0.040	12.491	0.000	
0.421	0.577				
Nbhd[T.4780]	0.5293	0.035	15.023	0.000	
0.460	0.598				
Nbhd[T.480]	0.2623	0.035	7.584	0.000	
0.194	0.330				
Nbhd[T.4800]	0.5036	0.036	13.828	0.000	
0.432	0.575				
Nbhd[T.4840]	0.5767	0.039	14.891	0.000	
0.501	0.653				
Nbhd[T.4860]	0.4905	0.040	12.389	0.000	
0.413	0.568				
Nbhd[T.4910]	0.8428	0.035	23.955	0.000	
0.774	0.912				
Nbhd[T.4920]	0.5689	0.039	14.682	0.000	
0.493	0.645				
Nbhd[T.50]	0.2372	0.058	4.103	0.000	
0.124	0.351				
Nbhd[T.520]	0.2443	0.054	4.538	0.000	
0.139	0.350				
Nbhd[T.560]	0.0151	0.038	0.396	0.692	
-0.060	0.090				
Nbhd[T.600]	0.1435	0.045	3.210	0.001	
0.056	0.231				
Nbhd[T.660]	0.1093	0.047	2.314	0.021	
0.017	0.202				
Nbhd[T.700]	0.0783	0.045	1.743	0.081	
-0.010	0.166				
Nbhd[T.780]	0.3486	0.037	9.452	0.000	
0.276	0.421				
Nbhd[T.800]	0.3121	0.057	5.494	0.000	
0.201	0.423				
Nbhd[T.820]	0.1716	0.042	4.091	0.000	
0.089	0.254				
Nbhd[T.900]	-0.3424	0.037	-9.158	0.000	
-0.416	-0.269				
Nbhd[T.960]	-0.3023	0.041	-7.417	0.000	
-0.382	-0.222				
Nbhd[T.980]	-0.1327	0.039	-3.383	0.001	
-0.210	-0.056				
Style[T.Bi-Level]	-0.6946	0.130	-5.339	0.000	
-0.950	-0.440				
Style[T.Cape Cod]	-0.5440	0.125	-4.344	0.000	
-0.789	-0.298				
Style[T.Colonial]	-0.5110	0.125	-4.077	0.000	

-0.757	-0.265				
Style[T.Cottage]		-0.7310	0.125	-5.827	0.000
-0.977	-0.485				
Style[T.Dplx Bungalow]		-0.6963	0.116	-6.008	0.000
-0.924	-0.469				
Style[T.Duplex N/S]		-0.7288	0.116	-6.269	0.000
-0.957	-0.501				
Style[T.Duplex O/S]		-0.7388	0.116	-6.387	0.000
-0.966	-0.512				
Style[T.Duplex-Cottage]		-0.8648	0.118	-7.345	0.000
-1.096	-0.634				
Style[T.Mansion]		-0.5230	0.129	-4.051	0.000
-0.776	-0.270				
Style[T.Milwaukee Bungalow]		-0.5215	0.125	-4.169	0.000
-0.767	-0.276				
Style[T.Ranch]		-0.5461	0.125	-4.355	0.000
-0.792	-0.300				
Style[T.Residence O/S]		-0.5643	0.125	-4.519	0.000
-0.809	-0.320				
Style[T.Rm or Rooming House]		-0.2160	0.129	-1.681	0.093
-0.468	0.036				
Style[T.Split Level]		-0.6106	0.127	-4.790	0.000
-0.860	-0.361				
Style[T.Townhouse]		-0.8891	0.118	-7.542	0.000
-1.120	-0.658				
Style[T.Triplex]		-0.7687	0.112	-6.883	0.000
-0.988	-0.550				
Style[T.Tudor]		-0.4815	0.126	-3.808	0.000
-0.729	-0.234				
Extwall[T.Block]		-0.0479	0.023	-2.058	0.040
-0.093	-0.002				
Extwall[T.Brick]		0.0234	0.005	4.700	0.000
0.014	0.033				
Extwall[T.Fiber-Cement]		0.1319	0.027	4.814	0.000
0.078	0.186				
Extwall[T.Frame]		-0.0611	0.006	-9.729	0.000
-0.073	-0.049				
Extwall[T.Masonry / Frame]		-0.0029	0.011	-0.260	0.795
-0.025	0.019				
Extwall[T.Prem Wood]		0.0423	0.036	1.174	0.241
-0.028	0.113				
Extwall[T.Stone]		0.0558	0.010	5.339	0.000
0.035	0.076				
Extwall[T.Stucco]		-0.0217	0.014	-1.542	0.123
-0.049	0.006				
C(year_from_date(Sale_date))[T.2003]		0.1907	0.167	1.139	0.255
-0.137	0.519				

```
C(year_from_date(Sale_date))[T.2004]     0.0502      0.177      0.284      0.777
-0.296       0.397
C(year_from_date(Sale_date))[T.2005]     0.3697      0.143      2.579      0.010
0.089       0.651
C(year_from_date(Sale_date))[T.2006]     0.7535      0.143      5.275      0.000
0.474       1.034
C(year_from_date(Sale_date))[T.2007]     0.4765      0.162      2.933      0.003
0.158       0.795
C(year_from_date(Sale_date))[T.2008]     0.7862      0.276      2.846      0.004
0.245       1.328
C(year_from_date(Sale_date))[T.2009]     0.4484      0.121      3.712      0.000
0.212       0.685
C(year_from_date(Sale_date))[T.2010]     0.4308      0.121      3.564      0.000
0.194       0.668
C(year_from_date(Sale_date))[T.2011]     0.3329      0.121      2.753      0.006
0.096       0.570
C(year_from_date(Sale_date))[T.2012]     0.3109      0.121      2.573      0.010
0.074       0.548
C(year_from_date(Sale_date))[T.2013]     0.2975      0.121      2.462      0.014
0.061       0.534
C(year_from_date(Sale_date))[T.2014]     0.3429      0.121      2.838      0.005
0.106       0.580
C(year_from_date(Sale_date))[T.2015]     0.3433      0.121      2.842      0.004
0.107       0.580
C(year_from_date(Sale_date))[T.2016]     0.4070      0.121      3.370      0.001
0.170       0.644
C(year_from_date(Sale_date))[T.2017]     0.4818      0.121      3.990      0.000
0.245       0.718
C(year_from_date(Sale_date))[T.2018]     0.5531      0.121      4.581      0.000
0.316       0.790
Stories                                  0.0049      0.009      0.517      0.605
-0.014       0.023
Year_Built                               0.0029      0.000     16.287      0.000
0.003       0.003
np.log(Fin_sqft)                         0.5342      0.011     47.522      0.000
0.512       0.556
Units                                   -0.0885      0.022     -4.050      0.000
-0.131      -0.046
Bdrms                                9.501e-06      0.000      0.077      0.938
-0.000       0.000
Fbath                                    0.1132      0.004     25.747      0.000
0.105       0.122
Hbath                                    0.0690      0.004     16.425      0.000
0.061       0.077
np.log(Lotsize)                          0.1245      0.008     16.568      0.000
0.110       0.139
==============================================================================
```

```
Omnibus:                       6984.310   Durbin-Watson:                   1.998
Prob(Omnibus):                    0.000   Jarque-Bera (JB):            78372.858
Skew:                            -1.393   Prob(JB):                         0.00
Kurtosis:                        12.403   Cond. No.                     5.98e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.98e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

4.2 How should the coefficient `Extwall[T.Brick]` of 0.0236 be interpreted? Select the appropriate choice below:

"A property with brick external walls is predicted to have a log price that is…"

(a) 0.0236 higher than a property with aluminum / vinyl external walls that is otherwise identical

(b) increased by 0.0236, all else being equal

(c) 0.0236 higher than the average property in the training dataset

Answer. (a) is the correct interpretation of the coefficient. Categorical coefficients in a linear regression are always interpreted relative to the reference category, which by default is the first category alphabetically, in this case "Aluminum / Vinyl".

4.3 Compute the mean absolute error for `model_feat`, and compare it to `model_log`. Does `model_feat` perform better or worse?

Answer. `model_feat` performs better than `model_log`:

[54]:
```
np.round(
    np.mean(                                                    # Mean
        np.abs(                                                 # Absolute
            model_feat.predict(test) - np.log(test.Sale_price)  # Error
        )
    ),
2) # round to two digits
```

[54]:
```
0.18
```

4.4 What story does this model tell us about the Milwaukee housing market during the last 15 years?

(a) The prices increased until 2007, when there was a housing crash. Since then, prices have gone back up to their pre-crash level

(b) Overall, housing prices steadily increased over time

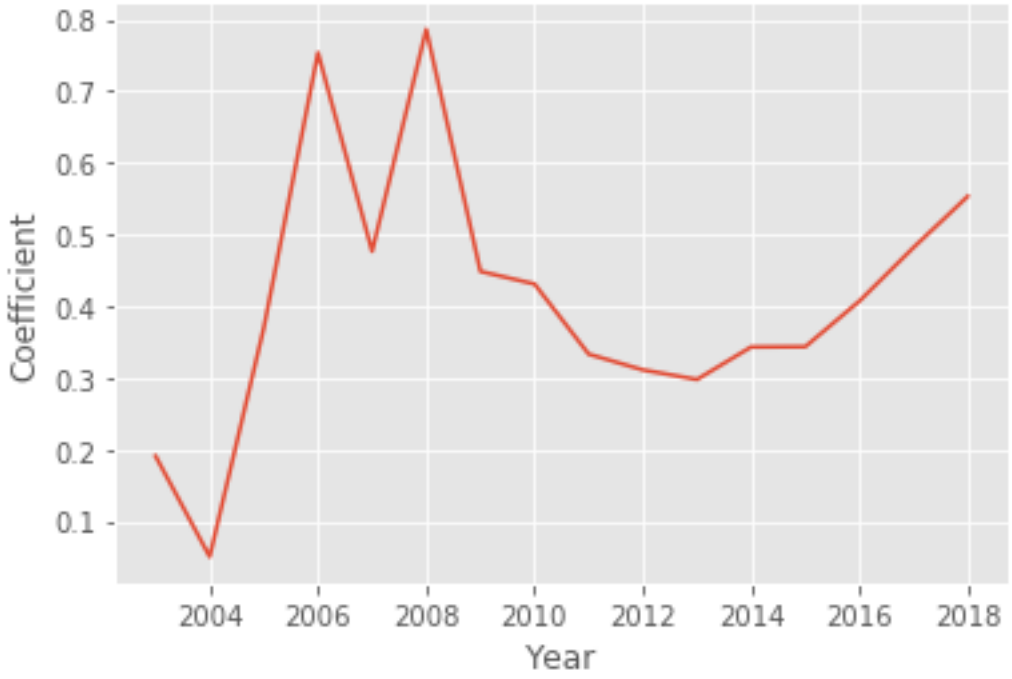
(c) The prices increased until 2007, then subsequently went down

(d) The prices increased until 2007, when there was a housing crash. Since then, prices have slowly risen, but they have not yet fully recovered

Answer. (d). The coefficient for 2018 is still lower than 2006 and 2008. This can also be seen from the plot of coefficients below:

[55]:
```
year = range(2003,2019)
year_coef = model_feat.params["C(year_from_date(Sale_date))[T.2003]":
 ↪"C(year_from_date(Sale_date))[T.2018]"]
plt.plot(year, year_coef)
plt.xlabel("Year")
plt.ylabel("Coefficient")
```

[55]:
```
Text(0, 0.5, 'Coefficient')
```



4.5 A 2-story townhouse is on the market for $150,000 with the following characteristics:

1. Neighborhood 4320 in District 11
2. 1200 finished square feet
3. 7200 square feet lot size
4. 3 bedrooms
5. Vinyl external wall

6. built in 1987
7. 1 full bath and 1 half-bath
8. 1 unit

Is this a good price according to our model? What assumptions do you have to make to answer this question?

Answer. Our model predicts (see below) that this house should sell for about \$130,000. Therefore \$150,000 is too high of a price unless the house has positive features that are not captured in the data.

One factor that we did not account for is that the sale took place in 2019. Since we have no 2019 data, we had to assume that prices remained flat after 2018. It is possible that real estate prices have gone up significantly in 2019, which could justify the \$150,000 asking price.

[56]:
```
d = {"Stories": 2,
     "Style": "Townhouse",
     "Extwall": "Aluminum / Vinyl",
     "Nbhd": '4320',
     "Fin_sqft": 1200.0,
     "Lotsize": 7200.0,
     "Sale_date": pd.datetime(2018,10,1), # there is no training data for 2019,
                                          # so we have to assume prices stayed␣
 ↪flat
                                          # since 2018
     "Year_Built":  1987,
     "Fbath": 1,
     "Hbath": 1,
     "Units": 1,
     "Bdrms": 3}
d_df = test.copy()
d_df = d_df.append(d, ignore_index=True)
pred = model_feat.predict(d_df.iloc[-1:]).iloc[0]
print("This property is predicted to sell for ${:,.0f}.".format(np.exp(pred)))
```

```
This property is predicted to sell for $130,370.
```

1.6 Collinearity and standard errors (20 mts)

However, adding a large dump of new features to a model and then leaving it to its own devices is not a prudent strategy. As we discussed in previous cases, new features could add almost no predictive power to our model, and we generally want a model that achieves a good level of predictiveness without sacrificing simplicity. The problem is not merely one of interpretability and convenience though; linear regression can actually be **destabilized** when there are too many predictors that do not add value. This means that small changes in the training data fed into the model results in large changes in the fitted coefficients; overfitting is a common consequence of this phenomenon.

1.6.1 Question: (5 mts)

What do you think causes the above to happen?

As one adds more predictors, they begin to contribute redundant information (which explains why every additional predictor contributes zero marginal information). Mathematically, this means that new predictors can often be expressed to a good extent as linear combinations (weighted sums) of some of the other predictors. When this happens, a model is said to exhibit **collinearity** or **multicollinearity**.

Since predictors become mostly redundant, this means that nearly identical predictions can be obtained by vastly different coefficient values, by "trading off" between the coefficients of the predictors which exhibit collinearity. Consequently, the model has no way to decide from the data alone which of these possible models with identical predictions to choose, which translates into inflated standard errors and a high risk of overfitting.

To demonstrate this effect, we will introduce an engineered feature: `log(Fin_sqft/Lotsize + 0.00001)`. The small constant is there to avoid making the new feature *perfectly* collinear with log(`Fin_sqft`) and log(`Lotsize`) (which would create a "divide-by-zero" error in the background regression function code):

[57]:
```
# base model without engineered collinear feature
model_finlot = smf.ols(formula = "np.log(Sale_price) ~ "
                               "Stories + Year_Built"
                               "+ np.log(Fin_sqft)"
                               "+ np.log(Lotsize)"
                               "+ Style"
                               "+ Extwall  + Units + Bdrms"
                               "+ Fbath + Hbath + Nbhd"
                               "+ C(year_from_date(Sale_date))",
                    data = train).fit()
model_finlot.summary()
```

[57]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(Sale_price)   R-squared:                       0.813
Model:                            OLS   Adj. R-squared:                  0.811
Method:                 Least Squares   F-statistic:                     445.7
Date:                Fri, 15 Nov 2019   Prob (F-statistic):               0.00
Time:                        08:18:13   Log-Likelihood:                -372.59
No. Observations:               19555   AIC:                             1125.
Df Residuals:                   19365   BIC:                             2623.
Df Model:                         189
Covariance Type:            nonrobust
==============================================================================
========================
                                    coef    std err          t      P>|t|
```

```
[0.025      0.975]
------------------------------------------------------------------------------
-----------------------
Intercept                           0.9107      0.376      2.422      0.015
0.174       1.648
Style[T.Bi-Level]                  -0.6946      0.130     -5.339      0.000
-0.950      -0.440
Style[T.Cape Cod]                  -0.5440      0.125     -4.344      0.000
-0.789      -0.298
Style[T.Colonial]                  -0.5110      0.125     -4.077      0.000
-0.757      -0.265
Style[T.Cottage]                   -0.7310      0.125     -5.827      0.000
-0.977      -0.485
Style[T.Dplx Bungalow]             -0.6963      0.116     -6.008      0.000
-0.924      -0.469
Style[T.Duplex N/S]                -0.7288      0.116     -6.269      0.000
-0.957      -0.501
Style[T.Duplex O/S]                -0.7388      0.116     -6.387      0.000
-0.966      -0.512
Style[T.Duplex-Cottage]            -0.8648      0.118     -7.345      0.000
-1.096      -0.634
Style[T.Mansion]                   -0.5230      0.129     -4.051      0.000
-0.776      -0.270
Style[T.Milwaukee Bungalow]        -0.5215      0.125     -4.169      0.000
-0.767      -0.276
Style[T.Ranch]                     -0.5461      0.125     -4.355      0.000
-0.792      -0.300
Style[T.Residence O/S]             -0.5643      0.125     -4.519      0.000
-0.809      -0.320
Style[T.Rm or Rooming House]       -0.2160      0.129     -1.681      0.093
-0.468       0.036
Style[T.Split Level]               -0.6106      0.127     -4.790      0.000
-0.860      -0.361
Style[T.Townhouse]                 -0.8891      0.118     -7.542      0.000
-1.120      -0.658
Style[T.Triplex]                   -0.7687      0.112     -6.883      0.000
-0.988      -0.550
Style[T.Tudor]                     -0.4815      0.126     -3.808      0.000
-0.729      -0.234
Extwall[T.Block]                   -0.0479      0.023     -2.058      0.040
-0.093      -0.002
Extwall[T.Brick]                    0.0234      0.005      4.700      0.000
0.014       0.033
Extwall[T.Fiber-Cement]             0.1319      0.027      4.814      0.000
0.078       0.186
Extwall[T.Frame]                   -0.0611      0.006     -9.729      0.000
-0.073      -0.049
```

Extwall[T.Masonry / Frame]	-0.0029	0.011	-0.260	0.795	
-0.025	0.019				
Extwall[T.Prem Wood]	0.0423	0.036	1.174	0.241	
-0.028	0.113				
Extwall[T.Stone]	0.0558	0.010	5.339	0.000	
0.035	0.076				
Extwall[T.Stucco]	-0.0217	0.014	-1.542	0.123	
-0.049	0.006				
Nbhd[T.1040]	0.2860	0.035	8.059	0.000	
0.216	0.356				
Nbhd[T.1140]	-0.3183	0.035	-9.003	0.000	
-0.388	-0.249				
Nbhd[T.1150]	-0.4187	0.041	-10.298	0.000	
-0.498	-0.339				
Nbhd[T.1160]	-0.4862	0.044	-10.938	0.000	
-0.573	-0.399				
Nbhd[T.1200]	-0.0088	0.044	-0.203	0.839	
-0.094	0.077				
Nbhd[T.1220]	0.4315	0.049	8.847	0.000	
0.336	0.527				
Nbhd[T.1340]	-0.3423	0.041	-8.267	0.000	
-0.423	-0.261				
Nbhd[T.1380]	0.0085	0.039	0.216	0.829	
-0.069	0.086				
Nbhd[T.1400]	0.1194	0.035	3.377	0.001	
0.050	0.189				
Nbhd[T.1440]	-0.5311	0.043	-12.391	0.000	
-0.615	-0.447				
Nbhd[T.1460]	0.2815	0.035	8.119	0.000	
0.214	0.349				
Nbhd[T.1470]	0.2154	0.035	6.113	0.000	
0.146	0.284				
Nbhd[T.1480]	0.0592	0.035	1.679	0.093	
-0.010	0.128				
Nbhd[T.1490]	0.0875	0.037	2.356	0.018	
0.015	0.160				
Nbhd[T.1500]	-0.5240	0.048	-10.918	0.000	
-0.618	-0.430				
Nbhd[T.1560]	-0.1224	0.048	-2.542	0.011	
-0.217	-0.028				
Nbhd[T.1620]	-0.2779	0.040	-6.944	0.000	
-0.356	-0.199				
Nbhd[T.1680]	-0.3772	0.042	-9.036	0.000	
-0.459	-0.295				
Nbhd[T.1740]	-0.2748	0.044	-6.298	0.000	
-0.360	-0.189				
Nbhd[T.1780]	-0.5533	0.047	-11.673	0.000	

```
-0.646      -0.460
Nbhd[T.1820]                -0.0526      0.042     -1.248      0.212
-0.135       0.030
Nbhd[T.1830]                 0.0743      0.041      1.814      0.070
-0.006       0.155
Nbhd[T.1840]                 0.1581      0.041      3.838      0.000
0.077       0.239
Nbhd[T.1850]                 0.1855      0.039      4.788      0.000
0.110       0.261
Nbhd[T.1860]                -0.1371      0.041     -3.310      0.001
-0.218      -0.056
Nbhd[T.1870]                 0.1039      0.045      2.334      0.020
0.017       0.191
Nbhd[T.1880]                -0.1017      0.093     -1.090      0.276
-0.284       0.081
Nbhd[T.1940]                 0.1137      0.039      2.947      0.003
0.038       0.189
Nbhd[T.1980]                 0.6183      0.037     16.899      0.000
0.547       0.690
Nbhd[T.2040]                 0.3191      0.034      9.421      0.000
0.253       0.385
Nbhd[T.2080]                 0.5396      0.033     16.493      0.000
0.475       0.604
Nbhd[T.2100]                 0.4632      0.032     14.270      0.000
0.400       0.527
Nbhd[T.2120]                 0.5512      0.037     14.925      0.000
0.479       0.624
Nbhd[T.2140]                 0.5439      0.044     12.411      0.000
0.458       0.630
Nbhd[T.2160]                -0.4760      0.053     -9.063      0.000
-0.579      -0.373
Nbhd[T.2180]                -0.0581      0.048     -1.219      0.223
-0.152       0.035
Nbhd[T.2300]                -0.0762      0.048     -1.593      0.111
-0.170       0.018
Nbhd[T.2380]                -0.8188      0.043    -18.986      0.000
-0.903      -0.734
Nbhd[T.240]                  0.3066      0.035      8.651      0.000
0.237       0.376
Nbhd[T.2400]                -0.7856      0.057    -13.781      0.000
-0.897      -0.674
Nbhd[T.2420]                -0.7120      0.064    -11.101      0.000
-0.838      -0.586
Nbhd[T.2500]                 0.1370      0.053      2.594      0.010
0.033       0.241
Nbhd[T.2510]                -0.2417      0.044     -5.486      0.000
-0.328      -0.155
```

Nbhd[T.2520]	-0.2642	0.043	-6.191	0.000	
-0.348	-0.181				
Nbhd[T.2540]	0.5370	0.034	15.569	0.000	
0.469	0.605				
Nbhd[T.2580]	0.8181	0.050	16.240	0.000	
0.719	0.917				
Nbhd[T.2600]	0.5765	0.037	15.722	0.000	
0.505	0.648				
Nbhd[T.2610]	0.2349	0.047	4.961	0.000	
0.142	0.328				
Nbhd[T.2660]	0.5680	0.044	13.008	0.000	
0.482	0.654				
Nbhd[T.2680]	0.7727	0.044	17.718	0.000	
0.687	0.858				
Nbhd[T.2700]	0.4613	0.039	11.716	0.000	
0.384	0.539				
Nbhd[T.2710]	0.4031	0.035	11.573	0.000	
0.335	0.471				
Nbhd[T.2720]	0.6083	0.035	17.287	0.000	
0.539	0.677				
Nbhd[T.2760]	0.5649	0.038	14.723	0.000	
0.490	0.640				
Nbhd[T.2800]	0.4003	0.048	8.407	0.000	
0.307	0.494				
Nbhd[T.2810]	0.5311	0.045	11.764	0.000	
0.443	0.620				
Nbhd[T.2820]	-0.7654	0.065	-11.727	0.000	
-0.893	-0.637				
Nbhd[T.2840]	-0.8635	0.049	-17.768	0.000	
-0.959	-0.768				
Nbhd[T.2850]	-1.0631	0.052	-20.349	0.000	
-1.165	-0.961				
Nbhd[T.2860]	-0.7609	0.046	-16.550	0.000	
-0.851	-0.671				
Nbhd[T.2870]	-1.2887	0.054	-23.717	0.000	
-1.395	-1.182				
Nbhd[T.2880]	0.6701	0.040	16.851	0.000	
0.592	0.748				
Nbhd[T.2890]	0.6543	0.040	16.315	0.000	
0.576	0.733				
Nbhd[T.2900]	-0.8344	0.050	-16.743	0.000	
-0.932	-0.737				
Nbhd[T.2910]	0.7915	0.047	16.967	0.000	
0.700	0.883				
Nbhd[T.2920]	-0.2866	0.046	-6.242	0.000	
-0.377	-0.197				
Nbhd[T.2930]	-0.7840	0.050	-15.590	0.000	

-0.883	-0.685				
Nbhd[T.2940]		-1.0642	0.051	-20.857	0.000
-1.164	-0.964				
Nbhd[T.2950]		-1.2675	0.044	-29.069	0.000
-1.353	-1.182				
Nbhd[T.2960]		-0.9971	0.050	-19.935	0.000
-1.095	-0.899				
Nbhd[T.3000]		-0.3939	0.047	-8.439	0.000
-0.485	-0.302				
Nbhd[T.3040]		-0.4187	0.045	-9.289	0.000
-0.507	-0.330				
Nbhd[T.3060]		1.1756	0.036	32.561	0.000
1.105	1.246				
Nbhd[T.3150]		1.0207	0.040	25.462	0.000
0.942	1.099				
Nbhd[T.3160]		1.0833	0.037	29.605	0.000
1.012	1.155				
Nbhd[T.3170]		1.0795	0.041	26.095	0.000
0.998	1.161				
Nbhd[T.3190]		1.0827	0.041	26.662	0.000
1.003	1.162				
Nbhd[T.3240]		1.1444	0.041	28.094	0.000
1.065	1.224				
Nbhd[T.3320]		0.7380	0.039	18.814	0.000
0.661	0.815				
Nbhd[T.3330]		0.8177	0.048	17.119	0.000
0.724	0.911				
Nbhd[T.360]		0.3189	0.040	7.883	0.000
0.240	0.398				
Nbhd[T.380]		0.1685	0.048	3.539	0.000
0.075	0.262				
Nbhd[T.40]		0.1303	0.046	2.813	0.005
0.040	0.221				
Nbhd[T.4000]		0.2206	0.035	6.264	0.000
0.152	0.290				
Nbhd[T.4020]		0.2913	0.050	5.807	0.000
0.193	0.390				
Nbhd[T.4040]		-0.1392	0.055	-2.518	0.012
-0.248	-0.031				
Nbhd[T.4050]		-0.1243	0.045	-2.773	0.006
-0.212	-0.036				
Nbhd[T.4060]		0.0984	0.036	2.743	0.006
0.028	0.169				
Nbhd[T.4100]		-0.1096	0.042	-2.593	0.010
-0.192	-0.027				
Nbhd[T.4120]		-0.1042	0.037	-2.809	0.005
-0.177	-0.032				

```
Nbhd[T.4160]                    0.0317      0.038       0.842       0.400
-0.042       0.105
Nbhd[T.4180]                    0.2174      0.039       5.564       0.000
0.141       0.294
Nbhd[T.4240]                    0.4917      0.033      14.979       0.000
0.427       0.556
Nbhd[T.4260]                    0.4647      0.045      10.296       0.000
0.376       0.553
Nbhd[T.4280]                    0.5167      0.046      11.207       0.000
0.426       0.607
Nbhd[T.4310]                    0.5759      0.039      14.711       0.000
0.499       0.653
Nbhd[T.4320]                    0.5361      0.035      15.476       0.000
0.468       0.604
Nbhd[T.4330]                    0.5124      0.037      13.689       0.000
0.439       0.586
Nbhd[T.4340]                    0.5725      0.033      17.438       0.000
0.508       0.637
Nbhd[T.4350]                    0.5220      0.036      14.315       0.000
0.451       0.593
Nbhd[T.4360]                    0.4314      0.036      12.021       0.000
0.361       0.502
Nbhd[T.4380]                    0.4712      0.037      12.803       0.000
0.399       0.543
Nbhd[T.440]                     0.3304      0.038       8.613       0.000
0.255       0.406
Nbhd[T.4400]                    0.4314      0.037      11.541       0.000
0.358       0.505
Nbhd[T.4410]                    0.5446      0.039      14.054       0.000
0.469       0.621
Nbhd[T.4420]                    0.5318      0.033      16.092       0.000
0.467       0.597
Nbhd[T.4425]                    0.4903      0.069       7.099       0.000
0.355       0.626
Nbhd[T.4430]                    0.5818      0.058      10.035       0.000
0.468       0.695
Nbhd[T.4500]                    0.8799      0.038      23.285       0.000
0.806       0.954
Nbhd[T.4510]                    0.9417      0.046      20.394       0.000
0.851       1.032
Nbhd[T.4520]                    0.4150      0.033      12.543       0.000
0.350       0.480
Nbhd[T.4540]                    1.0206      0.037      27.776       0.000
0.949       1.093
Nbhd[T.4560]                    1.1939      0.045      26.653       0.000
1.106       1.282
Nbhd[T.4580]                    0.7400      0.034      21.812       0.000
```

```
0.673       0.806
Nbhd[T.4600]                    0.9702      0.037     25.918      0.000
0.897       1.044
Nbhd[T.4610]                    0.8238      0.038     21.705      0.000
0.749       0.898
Nbhd[T.4620]                    0.6390      0.033     19.131      0.000
0.574       0.704
Nbhd[T.4660]                    0.4896      0.035     13.982      0.000
0.421       0.558
Nbhd[T.4700]                    0.5886      0.034     17.378      0.000
0.522       0.655
Nbhd[T.4720]                    0.5769      0.036     15.873      0.000
0.506       0.648
Nbhd[T.4740]                    0.4990      0.040     12.491      0.000
0.421       0.577
Nbhd[T.4780]                    0.5293      0.035     15.023      0.000
0.460       0.598
Nbhd[T.480]                     0.2623      0.035      7.584      0.000
0.194       0.330
Nbhd[T.4800]                    0.5036      0.036     13.828      0.000
0.432       0.575
Nbhd[T.4840]                    0.5767      0.039     14.891      0.000
0.501       0.653
Nbhd[T.4860]                    0.4905      0.040     12.389      0.000
0.413       0.568
Nbhd[T.4910]                    0.8428      0.035     23.955      0.000
0.774       0.912
Nbhd[T.4920]                    0.5689      0.039     14.682      0.000
0.493       0.645
Nbhd[T.50]                      0.2372      0.058      4.103      0.000
0.124       0.351
Nbhd[T.520]                     0.2443      0.054      4.538      0.000
0.139       0.350
Nbhd[T.560]                     0.0151      0.038      0.396      0.692
-0.060       0.090
Nbhd[T.600]                     0.1435      0.045      3.210      0.001
0.056       0.231
Nbhd[T.660]                     0.1093      0.047      2.314      0.021
0.017       0.202
Nbhd[T.700]                     0.0783      0.045      1.743      0.081
-0.010       0.166
Nbhd[T.780]                     0.3486      0.037      9.452      0.000
0.276       0.421
Nbhd[T.800]                     0.3121      0.057      5.494      0.000
0.201       0.423
Nbhd[T.820]                     0.1716      0.042      4.091      0.000
0.089       0.254
```

```
Nbhd[T.900]                                -0.3424      0.037     -9.158      0.000
-0.416      -0.269
Nbhd[T.960]                                -0.3023      0.041     -7.417      0.000
-0.382      -0.222
Nbhd[T.980]                                -0.1327      0.039     -3.383      0.001
-0.210      -0.056
C(year_from_date(Sale_date))[T.2003]         0.1907      0.167      1.139      0.255
-0.137       0.519
C(year_from_date(Sale_date))[T.2004]         0.0502      0.177      0.284      0.777
-0.296       0.397
C(year_from_date(Sale_date))[T.2005]         0.3697      0.143      2.579      0.010
0.089       0.651
C(year_from_date(Sale_date))[T.2006]         0.7535      0.143      5.275      0.000
0.474       1.034
C(year_from_date(Sale_date))[T.2007]         0.4765      0.162      2.933      0.003
0.158       0.795
C(year_from_date(Sale_date))[T.2008]         0.7862      0.276      2.846      0.004
0.245       1.328
C(year_from_date(Sale_date))[T.2009]         0.4484      0.121      3.712      0.000
0.212       0.685
C(year_from_date(Sale_date))[T.2010]         0.4308      0.121      3.564      0.000
0.194       0.668
C(year_from_date(Sale_date))[T.2011]         0.3329      0.121      2.753      0.006
0.096       0.570
C(year_from_date(Sale_date))[T.2012]         0.3109      0.121      2.573      0.010
0.074       0.548
C(year_from_date(Sale_date))[T.2013]         0.2975      0.121      2.462      0.014
0.061       0.534
C(year_from_date(Sale_date))[T.2014]         0.3429      0.121      2.838      0.005
0.106       0.580
C(year_from_date(Sale_date))[T.2015]         0.3433      0.121      2.842      0.004
0.107       0.580
C(year_from_date(Sale_date))[T.2016]         0.4070      0.121      3.370      0.001
0.170       0.644
C(year_from_date(Sale_date))[T.2017]         0.4818      0.121      3.990      0.000
0.245       0.718
C(year_from_date(Sale_date))[T.2018]         0.5531      0.121      4.581      0.000
0.316       0.790
Stories                                      0.0049      0.009      0.517      0.605
-0.014       0.023
Year_Built                                   0.0029      0.000     16.287      0.000
0.003       0.003
np.log(Fin_sqft)                             0.5342      0.011     47.522      0.000
0.512       0.556
np.log(Lotsize)                              0.1245      0.008     16.568      0.000
0.110       0.139
Units                                      -0.0885      0.022     -4.050      0.000
```

```
-0.131      -0.046
Bdrms                              9.501e-06      0.000       0.077       0.938
-0.000       0.000
Fbath                                 0.1132      0.004      25.747       0.000
0.105       0.122
Hbath                                 0.0690      0.004      16.425       0.000
0.061       0.077
==============================================================================
Omnibus:                     6984.310   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            78372.858
Skew:                          -1.393   Prob(JB):                         0.00
Kurtosis:                      12.403   Cond. No.                     5.98e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.98e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[58]:
```
model_coll = smf.ols(formula = "np.log(Sale_price) ~ "
                           "Stories + Year_Built"
                           "+ np.log(Fin_sqft)"
                           "+ np.log(Lotsize)"
                           "+ np.log(Fin_sqft/Lotsize + 0.0001)"
                           "+ Style"
                           "+ Extwall  + Units + Bdrms"
                           "+ Fbath + Hbath"
                           "+ Nbhd"
                           "+ C(year_from_date(Sale_date))",
                 data = train).fit()
model_coll.summary()
```

[58]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(Sale_price)   R-squared:                       0.813
Model:                            OLS   Adj. R-squared:                  0.811
Method:                 Least Squares   F-statistic:                     443.8
Date:                Fri, 15 Nov 2019   Prob (F-statistic):               0.00
Time:                        08:18:15   Log-Likelihood:                -363.38
No. Observations:               19555   AIC:                             1109.
Df Residuals:                   19364   BIC:                             2614.
Df Model:                         190
Covariance Type:            nonrobust
```

```
==============================================================================
                                coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
Intercept                      1.0152      0.377      2.696      0.007
0.277       1.753
Style[T.Bi-Level]             -0.7032      0.130     -5.407      0.000
-0.958      -0.448
Style[T.Cape Cod]             -0.5496      0.125     -4.390      0.000
-0.795      -0.304
Style[T.Colonial]             -0.5210      0.125     -4.158      0.000
-0.767      -0.275
Style[T.Cottage]              -0.7390      0.125     -5.893      0.000
-0.985      -0.493
Style[T.Dplx Bungalow]        -0.6994      0.116     -6.037      0.000
-0.926      -0.472
Style[T.Duplex N/S]           -0.7344      0.116     -6.319      0.000
-0.962      -0.507
Style[T.Duplex O/S]           -0.7428      0.116     -6.424      0.000
-0.969      -0.516
Style[T.Duplex-Cottage]       -0.8676      0.118     -7.372      0.000
-1.098      -0.637
Style[T.Mansion]              -0.5299      0.129     -4.107      0.000
-0.783      -0.277
Style[T.Milwaukee Bungalow]   -0.5278      0.125     -4.221      0.000
-0.773      -0.283
Style[T.Ranch]                -0.5502      0.125     -4.390      0.000
-0.796      -0.305
Style[T.Residence O/S]        -0.5719      0.125     -4.581      0.000
-0.817      -0.327
Style[T.Rm or Rooming House]  -0.2180      0.128     -1.697      0.090
-0.470       0.034
Style[T.Split Level]          -0.6181      0.127     -4.850      0.000
-0.868      -0.368
Style[T.Townhouse]            -0.8978      0.118     -7.618      0.000
-1.129      -0.667
Style[T.Triplex]              -0.7691      0.112     -6.890      0.000
-0.988      -0.550
Style[T.Tudor]                -0.4882      0.126     -3.862      0.000
-0.736      -0.240
Extwall[T.Block]              -0.0477      0.023     -2.051      0.040
-0.093      -0.002
Extwall[T.Brick]               0.0234      0.005      4.693      0.000
0.014       0.033
Extwall[T.Fiber-Cement]        0.1355      0.027      4.943      0.000
```

```
0.082       0.189
Extwall[T.Frame]                -0.0611      0.006     -9.734      0.000
-0.073      -0.049
Extwall[T.Masonry / Frame]      -0.0021      0.011     -0.188      0.851
-0.024       0.020
Extwall[T.Prem Wood]             0.0465      0.036      1.290      0.197
-0.024       0.117
Extwall[T.Stone]                 0.0563      0.010      5.383      0.000
0.036       0.077
Extwall[T.Stucco]               -0.0221      0.014     -1.569      0.117
-0.050       0.006
Nbhd[T.1040]                     0.2852      0.035      8.039      0.000
0.216       0.355
Nbhd[T.1140]                    -0.3194      0.035     -9.037      0.000
-0.389      -0.250
Nbhd[T.1150]                    -0.4204      0.041    -10.345      0.000
-0.500      -0.341
Nbhd[T.1160]                    -0.4881      0.044    -10.985      0.000
-0.575      -0.401
Nbhd[T.1200]                    -0.0092      0.044     -0.210      0.834
-0.095       0.076
Nbhd[T.1220]                     0.4287      0.049      8.793      0.000
0.333       0.524
Nbhd[T.1340]                    -0.3442      0.041     -8.316      0.000
-0.425      -0.263
Nbhd[T.1380]                     0.0072      0.039      0.183      0.855
-0.070       0.084
Nbhd[T.1400]                     0.1182      0.035      3.346      0.001
0.049       0.188
Nbhd[T.1440]                    -0.5327      0.043    -12.433      0.000
-0.617      -0.449
Nbhd[T.1460]                     0.2798      0.035      8.073      0.000
0.212       0.348
Nbhd[T.1470]                     0.2136      0.035      6.063      0.000
0.145       0.283
Nbhd[T.1480]                     0.0576      0.035      1.635      0.102
-0.011       0.127
Nbhd[T.1490]                     0.0854      0.037      2.300      0.021
0.013       0.158
Nbhd[T.1500]                    -0.5259      0.048    -10.962      0.000
-0.620      -0.432
Nbhd[T.1560]                    -0.1257      0.048     -2.610      0.009
-0.220      -0.031
Nbhd[T.1620]                    -0.2788      0.040     -6.969      0.000
-0.357      -0.200
Nbhd[T.1680]                    -0.3799      0.042     -9.105      0.000
-0.462      -0.298
```

```
Nbhd[T.1740]                    -0.2768      0.044     -6.347      0.000
-0.362      -0.191
Nbhd[T.1780]                    -0.5535      0.047    -11.684      0.000
-0.646      -0.461
Nbhd[T.1820]                    -0.0538      0.042     -1.277      0.202
-0.136       0.029
Nbhd[T.1830]                     0.0735      0.041      1.796      0.073
-0.007       0.154
Nbhd[T.1840]                     0.1582      0.041      3.842      0.000
0.078       0.239
Nbhd[T.1850]                     0.1852      0.039      4.782      0.000
0.109       0.261
Nbhd[T.1860]                    -0.1363      0.041     -3.292      0.001
-0.217      -0.055
Nbhd[T.1870]                     0.1029      0.045      2.312      0.021
0.016       0.190
Nbhd[T.1880]                    -0.1032      0.093     -1.107      0.268
-0.286       0.080
Nbhd[T.1940]                     0.1121      0.039      2.906      0.004
0.036       0.188
Nbhd[T.1980]                     0.6167      0.037     16.864      0.000
0.545       0.688
Nbhd[T.2040]                     0.3165      0.034      9.346      0.000
0.250       0.383
Nbhd[T.2080]                     0.5371      0.033     16.421      0.000
0.473       0.601
Nbhd[T.2100]                     0.4609      0.032     14.204      0.000
0.397       0.524
Nbhd[T.2120]                     0.5480      0.037     14.842      0.000
0.476       0.620
Nbhd[T.2140]                     0.5404      0.044     12.335      0.000
0.455       0.626
Nbhd[T.2160]                    -0.4694      0.053     -8.938      0.000
-0.572      -0.366
Nbhd[T.2180]                    -0.0561      0.048     -1.177      0.239
-0.150       0.037
Nbhd[T.2300]                    -0.0742      0.048     -1.551      0.121
-0.168       0.020
Nbhd[T.2380]                    -0.8141      0.043    -18.878      0.000
-0.899      -0.730
Nbhd[T.240]                      0.3126      0.035      8.816      0.000
0.243       0.382
Nbhd[T.2400]                    -0.7796      0.057    -13.677      0.000
-0.891      -0.668
Nbhd[T.2420]                    -0.7065      0.064    -11.017      0.000
-0.832      -0.581
Nbhd[T.2500]                     0.1364      0.053      2.583      0.010
```

```
0.033       0.240
Nbhd[T.2510]                    -0.2405      0.044     -5.461      0.000
-0.327      -0.154
Nbhd[T.2520]                    -0.2624      0.043     -6.151      0.000
-0.346      -0.179
Nbhd[T.2540]                     0.5381      0.034     15.608      0.000
0.471       0.606
Nbhd[T.2580]                     0.8157      0.050     16.198      0.000
0.717       0.914
Nbhd[T.2600]                     0.5771      0.037     15.745      0.000
0.505       0.649
Nbhd[T.2610]                     0.2337      0.047      4.939      0.000
0.141       0.326
Nbhd[T.2660]                     0.5644      0.044     12.929      0.000
0.479       0.650
Nbhd[T.2680]                     0.7698      0.044     17.657      0.000
0.684       0.855
Nbhd[T.2700]                     0.4586      0.039     11.652      0.000
0.381       0.536
Nbhd[T.2710]                     0.4017      0.035     11.538      0.000
0.333       0.470
Nbhd[T.2720]                     0.6061      0.035     17.229      0.000
0.537       0.675
Nbhd[T.2760]                     0.5623      0.038     14.660      0.000
0.487       0.637
Nbhd[T.2800]                     0.4023      0.048      8.452      0.000
0.309       0.496
Nbhd[T.2810]                     0.5366      0.045     11.886      0.000
0.448       0.625
Nbhd[T.2820]                    -0.7676      0.065    -11.765      0.000
-0.895      -0.640
Nbhd[T.2840]                    -0.8645      0.049    -17.797      0.000
-0.960      -0.769
Nbhd[T.2850]                    -1.0611      0.052    -20.318      0.000
-1.163      -0.959
Nbhd[T.2860]                    -0.7616      0.046    -16.571      0.000
-0.852      -0.671
Nbhd[T.2870]                    -1.2835      0.054    -23.626      0.000
-1.390      -1.177
Nbhd[T.2880]                     0.6728      0.040     16.925      0.000
0.595       0.751
Nbhd[T.2890]                     0.6589      0.040     16.431      0.000
0.580       0.738
Nbhd[T.2900]                    -0.8334      0.050    -16.730      0.000
-0.931      -0.736
Nbhd[T.2910]                     0.7989      0.047     17.121      0.000
0.707       0.890
```

```
Nbhd[T.2920]                    -0.2864      0.046     -6.242      0.000
-0.376      -0.196
Nbhd[T.2930]                    -0.7840      0.050    -15.598      0.000
-0.883      -0.686
Nbhd[T.2940]                    -1.0634      0.051    -20.851      0.000
-1.163      -0.963
Nbhd[T.2950]                    -1.2649      0.044    -29.019      0.000
-1.350      -1.179
Nbhd[T.2960]                    -0.9957      0.050    -19.915      0.000
-1.094      -0.898
Nbhd[T.3000]                    -0.3916      0.047     -8.394      0.000
-0.483      -0.300
Nbhd[T.3040]                    -0.4144      0.045     -9.196      0.000
-0.503      -0.326
Nbhd[T.3060]                     1.1784      0.036     32.649      0.000
1.108       1.249
Nbhd[T.3150]                     1.0242      0.040     25.557      0.000
0.946       1.103
Nbhd[T.3160]                     1.0875      0.037     29.723      0.000
1.016       1.159
Nbhd[T.3170]                     1.0855      0.041     26.237      0.000
1.004       1.167
Nbhd[T.3190]                     1.0917      0.041     26.860      0.000
1.012       1.171
Nbhd[T.3240]                     1.1527      0.041     28.279      0.000
1.073       1.233
Nbhd[T.3320]                     0.7374      0.039     18.807      0.000
0.661       0.814
Nbhd[T.3330]                     0.8189      0.048     17.153      0.000
0.725       0.913
Nbhd[T.360]                      0.3238      0.040      8.004      0.000
0.244       0.403
Nbhd[T.380]                      0.1761      0.048      3.697      0.000
0.083       0.269
Nbhd[T.40]                       0.1434      0.046      3.090      0.002
0.052       0.234
Nbhd[T.4000]                     0.2213      0.035      6.284      0.000
0.152       0.290
Nbhd[T.4020]                     0.3000      0.050      5.979      0.000
0.202       0.398
Nbhd[T.4040]                    -0.1332      0.055     -2.411      0.016
-0.242      -0.025
Nbhd[T.4050]                    -0.1214      0.045     -2.710      0.007
-0.209      -0.034
Nbhd[T.4060]                     0.1005      0.036      2.805      0.005
0.030       0.171
Nbhd[T.4100]                    -0.1062      0.042     -2.513      0.012
```

-0.189	-0.023				
Nbhd[T.4120]		-0.0992	0.037	-2.674	0.008
-0.172	-0.026				
Nbhd[T.4160]		0.0344	0.038	0.914	0.361
-0.039	0.108				
Nbhd[T.4180]		0.2182	0.039	5.586	0.000
0.142	0.295				
Nbhd[T.4240]		0.4897	0.033	14.926	0.000
0.425	0.554				
Nbhd[T.4260]		0.4621	0.045	10.241	0.000
0.374	0.551				
Nbhd[T.4280]		0.5132	0.046	11.136	0.000
0.423	0.604				
Nbhd[T.4310]		0.5738	0.039	14.661	0.000
0.497	0.650				
Nbhd[T.4320]		0.5341	0.035	15.424	0.000
0.466	0.602				
Nbhd[T.4330]		0.5101	0.037	13.631	0.000
0.437	0.583				
Nbhd[T.4340]		0.5709	0.033	17.396	0.000
0.507	0.635				
Nbhd[T.4350]		0.5209	0.036	14.291	0.000
0.449	0.592				
Nbhd[T.4360]		0.4294	0.036	11.968	0.000
0.359	0.500				
Nbhd[T.4380]		0.4722	0.037	12.836	0.000
0.400	0.544				
Nbhd[T.440]		0.3406	0.038	8.866	0.000
0.265	0.416				
Nbhd[T.4400]		0.4297	0.037	11.500	0.000
0.356	0.503				
Nbhd[T.4410]		0.5429	0.039	14.016	0.000
0.467	0.619				
Nbhd[T.4420]		0.5297	0.033	16.035	0.000
0.465	0.594				
Nbhd[T.4425]		0.4918	0.069	7.123	0.000
0.356	0.627				
Nbhd[T.4430]		0.5795	0.058	10.000	0.000
0.466	0.693				
Nbhd[T.4500]		0.8819	0.038	23.346	0.000
0.808	0.956				
Nbhd[T.4510]		0.9413	0.046	20.395	0.000
0.851	1.032				
Nbhd[T.4520]		0.4138	0.033	12.512	0.000
0.349	0.479				
Nbhd[T.4540]		1.0225	0.037	27.838	0.000
0.950	1.094				

```
Nbhd[T.4560]                    1.1963      0.045     26.716      0.000
1.109       1.284
Nbhd[T.4580]                    0.7391      0.034     21.797      0.000
0.673       0.806
Nbhd[T.4600]                    0.9713      0.037     25.959      0.000
0.898       1.045
Nbhd[T.4610]                    0.8236      0.038     21.711      0.000
0.749       0.898
Nbhd[T.4620]                    0.6365      0.033     19.062      0.000
0.571       0.702
Nbhd[T.4660]                    0.4883      0.035     13.950      0.000
0.420       0.557
Nbhd[T.4700]                    0.5868      0.034     17.329      0.000
0.520       0.653
Nbhd[T.4720]                    0.5750      0.036     15.828      0.000
0.504       0.646
Nbhd[T.4740]                    0.5001      0.040     12.523      0.000
0.422       0.578
Nbhd[T.4780]                    0.5304      0.035     15.059      0.000
0.461       0.599
Nbhd[T.480]                     0.2625      0.035      7.594      0.000
0.195       0.330
Nbhd[T.4800]                    0.5024      0.036     13.801      0.000
0.431       0.574
Nbhd[T.4840]                    0.5770      0.039     14.905      0.000
0.501       0.653
Nbhd[T.4860]                    0.4975      0.040     12.561      0.000
0.420       0.575
Nbhd[T.4910]                    0.8464      0.035     24.060      0.000
0.777       0.915
Nbhd[T.4920]                    0.5688      0.039     14.686      0.000
0.493       0.645
Nbhd[T.50]                      0.2364      0.058      4.091      0.000
0.123       0.350
Nbhd[T.520]                     0.2436      0.054      4.527      0.000
0.138       0.349
Nbhd[T.560]                     0.0200      0.038      0.522      0.601
-0.055       0.095
Nbhd[T.600]                     0.1457      0.045      3.262      0.001
0.058       0.233
Nbhd[T.660]                     0.1092      0.047      2.314      0.021
0.017       0.202
Nbhd[T.700]                     0.0825      0.045      1.837      0.066
-0.006       0.171
Nbhd[T.780]                     0.3551      0.037      9.626      0.000
0.283       0.427
Nbhd[T.800]                     0.3108      0.057      5.474      0.000
```

```
0.200       0.422
Nbhd[T.820]                                0.1756      0.042      4.186      0.000
0.093       0.258
Nbhd[T.900]                               -0.3451      0.037     -9.232      0.000
-0.418      -0.272
Nbhd[T.960]                               -0.3036      0.041     -7.452      0.000
-0.383      -0.224
Nbhd[T.980]                               -0.1340      0.039     -3.417      0.001
-0.211      -0.057
C(year_from_date(Sale_date))[T.2003]        0.1884      0.167      1.126      0.260
-0.140       0.516
C(year_from_date(Sale_date))[T.2004]        0.0498      0.177      0.282      0.778
-0.297       0.396
C(year_from_date(Sale_date))[T.2005]        0.3741      0.143      2.611      0.009
0.093       0.655
C(year_from_date(Sale_date))[T.2006]        0.7570      0.143      5.301      0.000
0.477       1.037
C(year_from_date(Sale_date))[T.2007]        0.4708      0.162      2.899      0.004
0.153       0.789
C(year_from_date(Sale_date))[T.2008]        0.7935      0.276      2.873      0.004
0.252       1.335
C(year_from_date(Sale_date))[T.2009]        0.4505      0.121      3.732      0.000
0.214       0.687
C(year_from_date(Sale_date))[T.2010]        0.4332      0.121      3.585      0.000
0.196       0.670
C(year_from_date(Sale_date))[T.2011]        0.3349      0.121      2.771      0.006
0.098       0.572
C(year_from_date(Sale_date))[T.2012]        0.3133      0.121      2.594      0.010
0.077       0.550
C(year_from_date(Sale_date))[T.2013]        0.2997      0.121      2.481      0.013
0.063       0.536
C(year_from_date(Sale_date))[T.2014]        0.3452      0.121      2.858      0.004
0.108       0.582
C(year_from_date(Sale_date))[T.2015]        0.3456      0.121      2.862      0.004
0.109       0.582
C(year_from_date(Sale_date))[T.2016]        0.4092      0.121      3.390      0.001
0.173       0.646
C(year_from_date(Sale_date))[T.2017]        0.4841      0.121      4.010      0.000
0.247       0.721
C(year_from_date(Sale_date))[T.2018]        0.5555      0.121      4.603      0.000
0.319       0.792
Stories                                     0.0096      0.010      1.011      0.312
-0.009       0.028
Year_Built                                  0.0028      0.000     15.702      0.000
0.002       0.003
np.log(Fin_sqft)                           91.6567     21.320      4.299      0.000
49.867     133.446
```

```
np.log(Lotsize)                          -90.9934     21.319     -4.268      0.000
-132.781     -49.206
np.log(Fin_sqft / Lotsize + 0.0001)      -91.1666     21.330     -4.274      0.000
-132.976     -49.357
Units                                     -0.0884      0.022     -4.047      0.000
-0.131      -0.046
Bdrms                                  8.151e-06      0.000      0.066      0.947
-0.000       0.000
Fbath                                     0.1130      0.004     25.721      0.000
0.104       0.122
Hbath                                     0.0680      0.004     16.159      0.000
0.060       0.076
==============================================================================
Omnibus:                     6985.029   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            78569.997
Skew:                          -1.392   Prob(JB):                         0.00
Kurtosis:                      12.417   Cond. No.                     4.05e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.05e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

1.6.2 Exercise 5: (5 mts)

5.1 What happened to the coefficients and standard errors of `log(Fin_sqft)` and `log(Lotsize)` as the engineered feature was introduced?

Answer. The coefficient of `log(Fin_sqft)` went up by a large number while the coeffient of `log(Lotsize)` went down by a similarly large number. The coefficient of `log(Lotsize)` is now an absurdly large *negative* number. The coefficients are destabilized, which is reflected in the much higher standard errors (which went from about 0.01 to 18).

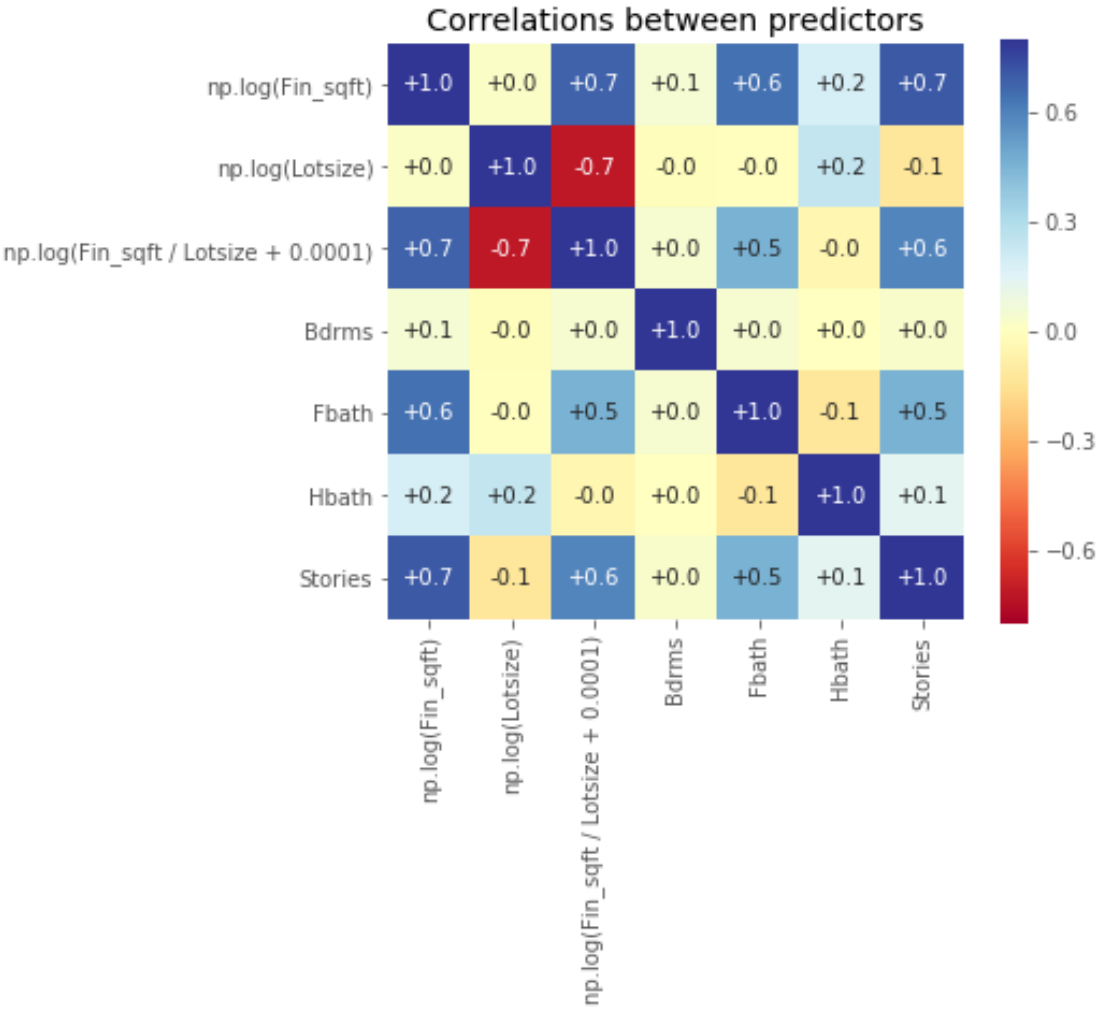
5.2: What happens when you change the 0.0001 constant (say, to 0.00001 or 0.001) in the engineered feature?

Answer. As the constant grows, the collinearity weakens.

To detect and investigate collinearities, it is useful to look at correlations between predictors within a model, shown below. Notice the high correlations (positive *or* negative) between the engineered feature `np.log(Fin_sqft / Lotsize + 0.0001)` and `np.log(Fin_sqft)/np.log(Lotsize)`. Note also the other high correlations with bathrooms and number of stories in the matrix. These can destabilize the model and make it difficult to make meaningful statements about the incremental value of adding extra bathrooms and stories.

[59]:
```
M = model_coll.model.data.exog # the design matrix X containing all
                               # predictors including dummy variables
                               # for categorical variables
param_names = model_coll.model.data.param_names # parameter names
M_df = pd.DataFrame(M, columns=param_names) # as a pandas data frame
# subset of predictors to plot:
M_sub = M_df[['np.log(Fin_sqft)','np.log(Lotsize)',
              'np.log(Fin_sqft / Lotsize + 0.0001)',
              'Bdrms','Fbath','Hbath','Stories']]
plt.figure(figsize=(6,5))
sns.heatmap(M_sub.corr(), cmap="RdYlBu",
    annot=True, square=True,
    vmin=-0.8, vmax=0.8, fmt="+.1f")
plt.title("Correlations between predictors")
```

[59]:
```
Text(0.5, 1, 'Correlations between predictors')
```



This illustrates the danger of just throwing additional features into a model. Not only is it redundant, but it can also lead to highly unstable and nonsensical values for the coefficients, which can lead to overfitting in production systems. One way of dealing with this is what we discussed in previous cases – using the AIC criterion to iteratively evaluate each new feature and determine if it is really adding enough incremental value on top of a penalty for the additional complexity. Another is to examine correlation matrices like the one we created above to identify any values close to -1 or 1. In future cases on **cross-validation** and **regularization**, you will learn additional ways to deal with overly complex and potentially overfit models.

1.7 Conclusions (5 mts)

In this case, we had a closer look at categorical predictors. We then focused on predictions, and discussed a few different metrics for the predictive power of a regression model. Each had their own advantages and disadvantages depending on what the business problem at hand needed to prioritize.

To improve the predictive power of our model, we added additional predictors. We learned how to **engineer features** to capture information that was not directly available as a column of the data set. However, we saw how adding so many features at once could lead to **collinearity** and therefore destabilize a model, making coefficients difficult to interpret and increasing the chance of overfit predictions in production.

1.8 Takeaways (5 mts)

More complex models can provide better predictions, but they also make it harder to interpret the model. Furthermore, more complex models can sometimes actually destabilize performance. In many such situations, complexity creates collinearity between variables which actually makes our models highly unstable and prone to overfitting. Since much of the business world is about applying trained models on future, currently unknown data, overfitting can lead to serious, negative impacts on the bottom line.