Feature_Engineering

February 27, 2021

0.1 Feature Engineering with Linear Regression: Applied to the Ames Housing Data

Using the Ames Housing Data:

Dean De Cock Truman State University Journal of Statistics Education Volume 19, Number 3(2011), www.amstat.org/publications/jse/v19n3/decock.pdf

In this notebook, we will build some linear regression models to predict housing prices from this data. In particular, we will set out to improve on a baseline set of features via **feature engineering**: deriving new features from our existing data. Feature engineering often makes the difference between a weak model and a strong one.

We will use visual exploration, domain understanding, and intuition to construct new features that will be useful later in the course as we turn to prediction.

Notebook Contents

1. Simple EDA
2. One-hot Encoding variables
3. Log transformation for skewed variables
4. Pair plot for features
5. Basic feature engineering: adding polynomial and interaction terms
6. Feature engineering: categories and features derived from category aggregates

0.2 1. Simple EDA

[48]:
```
%pylab inline
%config InlineBackend.figure_formats = ['retina']

import pandas as pd
import seaborn as sns
sns.set() #this will settings defaults for seaborn outputs
```

```
Populating the interactive namespace from numpy and matplotlib
```

Load the Data, Examine and Explore

[49]:
```
## Load in the Ames Housing Data
datafile = "D:\Courses\IBM Machine Learning Professional␣
↪Certificate\Exploratory Data Analysis for Machine␣
↪Learning\data\Ames_Housing_Data.tsv"
```

```python
df = pd.read_csv(datafile, sep='\t')
```

[50]:
```python
df
```

[50]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
0         1  526301100           20        RL         141.0     31770   Pave
1         2  526350040           20        RH          80.0     11622   Pave
2         3  526351010           20        RL          81.0     14267   Pave
3         4  526353030           20        RL          93.0     11160   Pave
4         5  527105010           60        RL          74.0     13830   Pave
…       …        …            …         …             …       …    …
2925   2926  923275080           80        RL          37.0      7937   Pave
2926   2927  923276100           20        RL           NaN      8885   Pave
2927   2928  923400125           85        RL          62.0     10441   Pave
2928   2929  924100070           20        RL          77.0     10010   Pave
2929   2930  924151050           60        RL          74.0      9627   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
0      NaN       IR1          Lvl  …         0     NaN    NaN          NaN
1      NaN       Reg          Lvl  …         0     NaN  MnPrv          NaN
2      NaN       IR1          Lvl  …         0     NaN    NaN         Gar2
3      NaN       Reg          Lvl  …         0     NaN    NaN          NaN
4      NaN       IR1          Lvl  …         0     NaN  MnPrv          NaN
…      …        …          …   …         …     …     …            …
2925   NaN       IR1          Lvl  …         0     NaN  GdPrv          NaN
2926   NaN       IR1          Low  …         0     NaN  MnPrv          NaN
2927   NaN       Reg          Lvl  …         0     NaN  MnPrv         Shed
2928   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2929   NaN       Reg          Lvl  …         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0           0       5    2010       WD           Normal     215000
1           0       6    2010       WD           Normal     105000
2       12500       6    2010       WD           Normal     172000
3           0       4    2010       WD           Normal     244000
4           0       3    2010       WD           Normal     189900
…         …     …     …       …             …        …
2925        0       3    2006       WD           Normal     142500
2926        0       6    2006       WD           Normal     131000
2927      700       7    2006       WD           Normal     132000
2928        0       4    2006       WD           Normal     170000
2929        0      11    2006       WD           Normal     188000

[2930 rows x 82 columns]
```

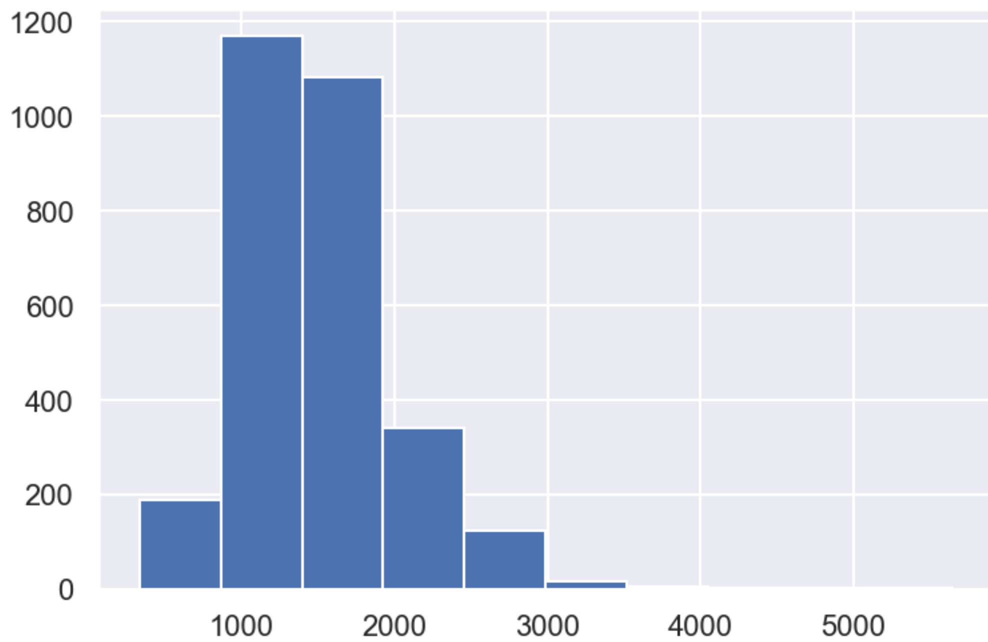
[51]:
```python
## Examine the columns, look at missing data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 82 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Order            2930 non-null   int64
 1   PID              2930 non-null   int64
 2   MS SubClass      2930 non-null   int64
 3   MS Zoning        2930 non-null   object
 4   Lot Frontage     2440 non-null   float64
 5   Lot Area         2930 non-null   int64
 6   Street           2930 non-null   object
 7   Alley            198 non-null    object
 8   Lot Shape        2930 non-null   object
 9   Land Contour     2930 non-null   object
 10  Utilities        2930 non-null   object
 11  Lot Config       2930 non-null   object
 12  Land Slope       2930 non-null   object
 13  Neighborhood     2930 non-null   object
 14  Condition 1      2930 non-null   object
 15  Condition 2      2930 non-null   object
 16  Bldg Type        2930 non-null   object
 17  House Style      2930 non-null   object
 18  Overall Qual     2930 non-null   int64
 19  Overall Cond     2930 non-null   int64
 20  Year Built       2930 non-null   int64
 21  Year Remod/Add   2930 non-null   int64
 22  Roof Style       2930 non-null   object
 23  Roof Matl        2930 non-null   object
 24  Exterior 1st     2930 non-null   object
 25  Exterior 2nd     2930 non-null   object
 26  Mas Vnr Type     2907 non-null   object
 27  Mas Vnr Area     2907 non-null   float64
 28  Exter Qual       2930 non-null   object
 29  Exter Cond       2930 non-null   object
 30  Foundation       2930 non-null   object
 31  Bsmt Qual        2850 non-null   object
 32  Bsmt Cond        2850 non-null   object
 33  Bsmt Exposure    2847 non-null   object
 34  BsmtFin Type 1   2850 non-null   object
 35  BsmtFin SF 1     2929 non-null   float64
 36  BsmtFin Type 2   2849 non-null   object
 37  BsmtFin SF 2     2929 non-null   float64
 38  Bsmt Unf SF      2929 non-null   float64
 39  Total Bsmt SF    2929 non-null   float64
 40  Heating          2930 non-null   object
 41  Heating QC       2930 non-null   object
 42  Central Air      2930 non-null   object
```

```
 43  Electrical       2929 non-null   object 
 44  1st Flr SF       2930 non-null   int64  
 45  2nd Flr SF       2930 non-null   int64  
 46  Low Qual Fin SF  2930 non-null   int64  
 47  Gr Liv Area      2930 non-null   int64  
 48  Bsmt Full Bath   2928 non-null   float64
 49  Bsmt Half Bath   2928 non-null   float64
 50  Full Bath        2930 non-null   int64  
 51  Half Bath        2930 non-null   int64  
 52  Bedroom AbvGr    2930 non-null   int64  
 53  Kitchen AbvGr    2930 non-null   int64  
 54  Kitchen Qual     2930 non-null   object 
 55  TotRms AbvGrd    2930 non-null   int64  
 56  Functional       2930 non-null   object 
 57  Fireplaces       2930 non-null   int64  
 58  Fireplace Qu     1508 non-null   object 
 59  Garage Type      2773 non-null   object 
 60  Garage Yr Blt    2771 non-null   float64
 61  Garage Finish    2771 non-null   object 
 62  Garage Cars      2929 non-null   float64
 63  Garage Area      2929 non-null   float64
 64  Garage Qual      2771 non-null   object 
 65  Garage Cond      2771 non-null   object 
 66  Paved Drive      2930 non-null   object 
 67  Wood Deck SF     2930 non-null   int64  
 68  Open Porch SF    2930 non-null   int64  
 69  Enclosed Porch   2930 non-null   int64  
 70  3Ssn Porch       2930 non-null   int64  
 71  Screen Porch     2930 non-null   int64  
 72  Pool Area        2930 non-null   int64  
 73  Pool QC          13 non-null     object 
 74  Fence            572 non-null    object 
 75  Misc Feature     106 non-null    object 
 76  Misc Val         2930 non-null   int64  
 77  Mo Sold          2930 non-null   int64  
 78  Yr Sold          2930 non-null   int64  
 79  Sale Type        2930 non-null   object 
 80  Sale Condition   2930 non-null   object 
 81  SalePrice        2930 non-null   int64  
dtypes: float64(11), int64(28), object(43)
memory usage: 1.8+ MB
```

[52]:
```
df['Gr Liv Area'].hist() #there are some outliers above 4000 entries
```

[52]:
```
<AxesSubplot:>
```



[53]:
```
# This is recommended by the data set author to remove a few outliers

df = df.loc[df['Gr Liv Area'] <= 4000,:]
print("Number of rows in the data:", df.shape[0])
print("Number of columns in the data:", df.shape[1])
data = df.copy() # Keep a copy our original data, its a good practice
```

```
Number of rows in the data: 2925
Number of columns in the data: 82
```

[54]:
```
# A quick look at the data:
df.head()
```

[54]:
```
   Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
0      1  526301100           20        RL         141.0     31770   Pave
1      2  526350040           20        RH          80.0     11622   Pave
2      3  526351010           20        RL          81.0     14267   Pave
3      4  526353030           20        RL          93.0     11160   Pave
4      5  527105010           60        RL          74.0     13830   Pave

  Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
0   NaN       IR1          Lvl  …         0     NaN    NaN          NaN
1   NaN       Reg          Lvl  …         0     NaN  MnPrv          NaN
2   NaN       IR1          Lvl  …         0     NaN    NaN         Gar2
3   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
4   NaN       IR1          Lvl  …         0     NaN  MnPrv          NaN
```

```
  Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0        0       5    2010       WD           Normal     215000
1        0       6    2010       WD           Normal     105000
2    12500       6    2010       WD           Normal     172000
3        0       4    2010       WD           Normal     244000
4        0       3    2010       WD           Normal     189900

[5 rows x 82 columns]
```

[55]:
```
len(df.PID.unique()) #this is unique value column and this wont give any␣
↪benifit for prediction
```

[55]:
```
2925
```

[56]:
```
len(df.Order.unique()) #same as PID
```

[56]:
```
2925
```

[57]:
```
#drop those unique value columns
df.drop(['PID','Order'],axis=1,inplace=True)
```

```
C:\Users\cspon\anaconda3\lib\site-packages\pandas\core\frame.py:4163:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

[58]:
```
df.head()
```

[58]:
```
   MS SubClass MS Zoning  Lot Frontage  Lot Area Street Alley Lot Shape  \
0           20        RL         141.0     31770   Pave   NaN       IR1
1           20        RH          80.0     11622   Pave   NaN       Reg
2           20        RL          81.0     14267   Pave   NaN       IR1
3           20        RL          93.0     11160   Pave   NaN       Reg
4           60        RL          74.0     13830   Pave   NaN       IR1

  Land Contour Utilities Lot Config  … Pool Area Pool QC  Fence  \
0          Lvl    AllPub     Corner  …         0     NaN    NaN
1          Lvl    AllPub     Inside  …         0     NaN  MnPrv
2          Lvl    AllPub     Corner  …         0     NaN    NaN
3          Lvl    AllPub     Corner  …         0     NaN    NaN
4          Lvl    AllPub     Inside  …         0     NaN  MnPrv

  Misc Feature Misc Val Mo Sold  Yr Sold  Sale Type  Sale Condition  SalePrice
0          NaN        0       5     2010         WD          Normal     215000
```

```
1           NaN         0        6     2010          WD           Normal     105000
2          Gar2     12500        6     2010          WD           Normal     172000
3           NaN         0        4     2010          WD           Normal     244000
4           NaN         0        3     2010          WD           Normal     189900

[5 rows x 80 columns]
```

We're going to first do some basic data cleaning on this data:

- Converting categorical variables to dummies
- Making skew variables symmetric

0.2.1 One-hot encoding for dummy variables:

[ ]:
```
# Get a Pd.Series consisting of all the string categoricals
one_hot_encode_cols = df.dtypes[df.dtypes == np.object]  # filtering by string␣
↪categoricals
one_hot_encode_cols = one_hot_encode_cols.index.tolist()  # list of categorical␣
↪fields

df[one_hot_encode_cols].head().T
```

[ ]:
```
one_hot_encode_cols
```

We're going to first do some basic data cleaning on this data:

- Converting categorical variables to dummies
- Making skew variables symmetric

One-hot encoding the dummy variables:

[ ]:
```
# Do the one hot encoding
df = pd.get_dummies(df, columns=one_hot_encode_cols, drop_first=True)
df.describe().T
```

[ ]:
```
df
```

0.2.2 Log transforming skew variables

[59]:
```
df.select_dtypes('number') #filter out only numerical values
```

[59]:
```
      MS SubClass  Lot Frontage  Lot Area  Overall Qual  Overall Cond  \
0              20         141.0     31770             6             5
1              20          80.0     11622             5             6
2              20          81.0     14267             6             6
3              20          93.0     11160             7             5
4              60          74.0     13830             5             5
…             …            …        …             …            …
2925           80          37.0      7937             6             6
2926           20           NaN      8885             5             5
```

```
2927          85          62.0     10441             5              5
2928          20          77.0     10010             5              5
2929          60          74.0      9627             7              5

      Year Built  Year Remod/Add  Mas Vnr Area  BsmtFin SF 1  BsmtFin SF 2  \
0           1960            1960         112.0         639.0           0.0
1           1961            1961           0.0         468.0         144.0
2           1958            1958         108.0         923.0           0.0
3           1968            1968           0.0        1065.0           0.0
4           1997            1998           0.0         791.0           0.0
…          …              …            …            …            …
2925        1984            1984           0.0         819.0           0.0
2926        1983            1983           0.0         301.0         324.0
2927        1992            1992           0.0         337.0           0.0
2928        1974            1975           0.0        1071.0         123.0
2929        1993            1994          94.0         758.0           0.0

      …  Wood Deck SF  Open Porch SF  Enclosed Porch  3Ssn Porch  \
0     …           210             62               0           0
1     …           140              0               0           0
2     …           393             36               0           0
3     …             0              0               0           0
4     …           212             34               0           0
…    …           …             …              …           …
2925  …           120              0               0           0
2926  …           164              0               0           0
2927  …            80             32               0           0
2928  …           240             38               0           0
2929  …           190             48               0           0

      Screen Porch  Pool Area  Misc Val  Mo Sold  Yr Sold  SalePrice
0                0          0         0        5     2010     215000
1              120          0         0        6     2010     105000
2                0          0     12500        6     2010     172000
3                0          0         0        4     2010     244000
4                0          0         0        3     2010     189900
…             …        …        …       …      …       …
2925             0          0         0        3     2006     142500
2926             0          0         0        6     2006     131000
2927             0          0       700        7     2006     132000
2928             0          0         0        4     2006     170000
2929             0          0         0       11     2006     188000

[2925 rows x 37 columns]
```

[60]:

```
df.select_dtypes('object') #filter out only categorical values
```

```
[60]:     MS Zoning Street Alley Lot Shape Land Contour Utilities Lot Config  \
0           RL   Pave   NaN       IR1          Lvl    AllPub     Corner
1           RH   Pave   NaN       Reg          Lvl    AllPub     Inside
2           RL   Pave   NaN       IR1          Lvl    AllPub     Corner
3           RL   Pave   NaN       Reg          Lvl    AllPub     Corner
4           RL   Pave   NaN       IR1          Lvl    AllPub     Inside
…         …    …   …       …          …       …         …
2925        RL   Pave   NaN       IR1          Lvl    AllPub    CulDSac
2926        RL   Pave   NaN       IR1          Low    AllPub     Inside
2927        RL   Pave   NaN       Reg          Lvl    AllPub     Inside
2928        RL   Pave   NaN       Reg          Lvl    AllPub     Inside
2929        RL   Pave   NaN       Reg          Lvl    AllPub     Inside

     Land Slope Neighborhood Condition 1  … Garage Type Garage Finish  \
0           Gtl        NAmes        Norm  …      Attchd           Fin
1           Gtl        NAmes       Feedr  …      Attchd           Unf
2           Gtl        NAmes        Norm  …      Attchd           Unf
3           Gtl        NAmes        Norm  …      Attchd           Fin
4           Gtl      Gilbert        Norm  …      Attchd           Fin
…           …          …         …  …         …             …
2925        Gtl      Mitchel        Norm  …      Detchd           Unf
2926        Mod      Mitchel        Norm  …      Attchd           Unf
2927        Gtl      Mitchel        Norm  …         NaN           NaN
2928        Mod      Mitchel        Norm  …      Attchd           RFn
2929        Mod      Mitchel        Norm  …      Attchd           Fin

     Garage Qual Garage Cond Paved Drive Pool QC  Fence Misc Feature  \
0             TA          TA           P     NaN    NaN          NaN
1             TA          TA           Y     NaN  MnPrv          NaN
2             TA          TA           Y     NaN    NaN         Gar2
3             TA          TA           Y     NaN    NaN          NaN
4             TA          TA           Y     NaN  MnPrv          NaN
…            …          …          …     …    …          …
2925          TA          TA           Y     NaN  GdPrv          NaN
2926          TA          TA           Y     NaN  MnPrv          NaN
2927         NaN         NaN           Y     NaN  MnPrv         Shed
2928          TA          TA           Y     NaN    NaN          NaN
2929          TA          TA           Y     NaN    NaN          NaN

     Sale Type Sale Condition
0           WD         Normal
1           WD         Normal
2           WD         Normal
3           WD         Normal
4           WD         Normal
…         …            …
2925        WD         Normal
```

```
2926        WD          Normal
2927        WD          Normal
2928        WD          Normal
2929        WD          Normal

[2925 rows x 43 columns]
```

[61]:
```python
df.select_dtypes('number').columns
```

[61]:
```
Index(['MS SubClass', 'Lot Frontage', 'Lot Area', 'Overall Qual',
       'Overall Cond', 'Year Built', 'Year Remod/Add', 'Mas Vnr Area',
       'BsmtFin SF 1', 'BsmtFin SF 2', 'Bsmt Unf SF', 'Total Bsmt SF',
       '1st Flr SF', '2nd Flr SF', 'Low Qual Fin SF', 'Gr Liv Area',
       'Bsmt Full Bath', 'Bsmt Half Bath', 'Full Bath', 'Half Bath',
       'Bedroom AbvGr', 'Kitchen AbvGr', 'TotRms AbvGrd', 'Fireplaces',
       'Garage Yr Blt', 'Garage Cars', 'Garage Area', 'Wood Deck SF',
       'Open Porch SF', 'Enclosed Porch', '3Ssn Porch', 'Screen Porch',
       'Pool Area', 'Misc Val', 'Mo Sold', 'Yr Sold', 'SalePrice'],
      dtype='object')
```

[62]:
```python
# Create a list of numerical colums to check for skewing
num_cols = df.select_dtypes('number').columns

skew_limit = 0.75 # define a limit above which we will log transform
skew_vals = df[num_cols].skew()
```

[63]:
```python
skew_vals #0 means no skew, >0 means positive skew, <0 means negative skew
```

[63]:
```
MS SubClass         1.356549
Lot Frontage        1.111071
Lot Area           13.200004
Overall Qual        0.171657
Overall Cond        0.572769
Year Built         -0.602475
Year Remod/Add     -0.449567
Mas Vnr Area        2.565458
BsmtFin SF 1        0.821985
BsmtFin SF 2        4.135900
Bsmt Unf SF         0.925021
Total Bsmt SF       0.399079
1st Flr SF          0.942615
2nd Flr SF          0.847517
Low Qual Fin SF    12.107629
Gr Liv Area         0.878879
Bsmt Full Bath      0.615553
Bsmt Half Bath      3.965970
Full Bath           0.164954
```

```
Half Bath          0.702966
Bedroom AbvGr      0.306912
Kitchen AbvGr      4.309573
TotRms AbvGrd      0.704992
Fireplaces         0.732312
Garage Yr Blt     -0.382039
Garage Cars       -0.219734
Garage Area        0.213681
Wood Deck SF       1.848286
Open Porch SF      2.495162
Enclosed Porch     4.010586
3Ssn Porch        11.393854
Screen Porch       3.953495
Pool Area         18.743766
Misc Val          22.225015
Mo Sold            0.195773
Yr Sold            0.132843
SalePrice          1.591072
dtype: float64
```

[64]:

```
# Showing the skewed columns
skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))

skew_cols
```

[64]:

	Skew
Misc Val	22.225015
Pool Area	18.743766
Lot Area	13.200004
Low Qual Fin SF	12.107629
3Ssn Porch	11.393854
Kitchen AbvGr	4.309573
BsmtFin SF 2	4.135900
Enclosed Porch	4.010586
Bsmt Half Bath	3.965970
Screen Porch	3.953495
Mas Vnr Area	2.565458
Open Porch SF	2.495162
Wood Deck SF	1.848286
SalePrice	1.591072
MS SubClass	1.356549
Lot Frontage	1.111071
1st Flr SF	0.942615

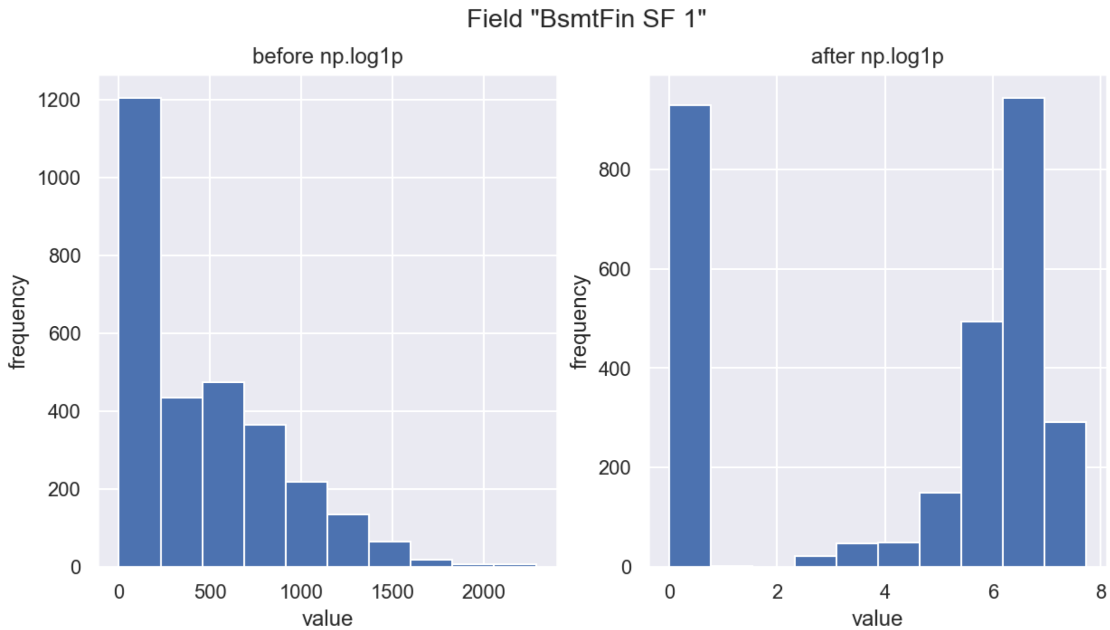
```
Bsmt Unf SF       0.925021
Gr Liv Area       0.878879
2nd Flr SF        0.847517
BsmtFin SF 1      0.821985
```

[65]:
```
#another way
skew_cols = skew_vals[abs(skew_vals)>skew_limit].sort_values(ascending=False)
skew_cols
```

[65]:
```
Misc Val           22.225015
Pool Area          18.743766
Lot Area           13.200004
Low Qual Fin SF    12.107629
3Ssn Porch         11.393854
Kitchen AbvGr       4.309573
BsmtFin SF 2        4.135900
Enclosed Porch      4.010586
Bsmt Half Bath      3.965970
Screen Porch        3.953495
Mas Vnr Area        2.565458
Open Porch SF       2.495162
Wood Deck SF        1.848286
SalePrice           1.591072
MS SubClass         1.356549
Lot Frontage        1.111071
1st Flr SF          0.942615
Bsmt Unf SF         0.925021
Gr Liv Area         0.878879
2nd Flr SF          0.847517
BsmtFin SF 1        0.821985
dtype: float64
```

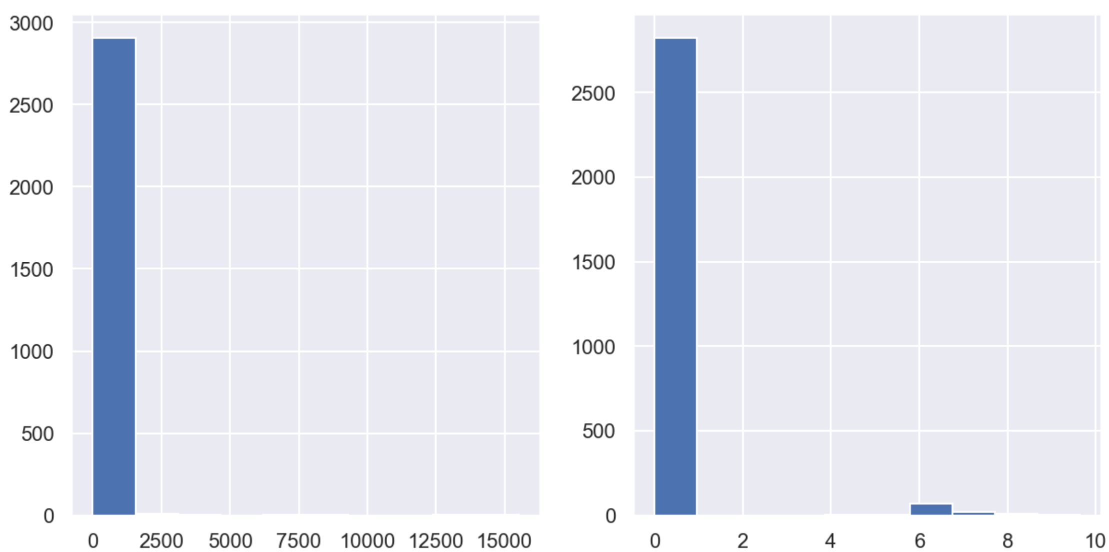
[66]:
```
# Let's look at what happens to one of these features, when we apply np.log1p␣
↪visually.

# Choose a field
field = "BsmtFin SF 1"

# Create two "subplots" and a "figure" using matplotlib
fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(10, 5))

# Create a histogram on the "ax_before" subplot
df[field].hist(ax=ax_before)

# Apply a log transformation (numpy syntax) to this column
df[field].apply(np.log1p).hist(ax=ax_after)
```

```
# Formatting of titles etc. for each subplot
ax_before.set(title='before np.log1p', ylabel='frequency', xlabel='value')
ax_after.set(title='after np.log1p', ylabel='frequency', xlabel='value')
fig.suptitle('Field "{}"'.format(field));
```



[67]:
```
fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(10, 5))
df['Misc Val'].hist(ax=ax_before)
df['Misc Val'].apply(np.log1p).hist(ax=ax_after)
```

[67]: <AxesSubplot:>



[69]:
```python
# Perform the skew transformation:

for col in skew_cols.index.values:
    if col == "SalePrice":
        continue
    df[col] = df[col].apply(np.log1p)
```

```
<ipython-input-69-aa893cc83dad>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[col] = df[col].apply(np.log1p)
```

[70]:
```python
df.head()
```

[70]:
```
   MS SubClass MS Zoning  Lot Frontage  Lot Area Street Alley Lot Shape  \
0     1.397363        RL      1.784370  2.430654   Pave   NaN       IR1
1     1.397363        RH      1.685370  2.338024   Pave   NaN       Reg
2     1.397363        RL      1.687642  2.357620   Pave   NaN       IR1
3     1.397363        RL      1.712589  2.334101   Pave   NaN       Reg
4     1.631370        RL      1.671001  2.354672   Pave   NaN       IR1

  Land Contour Utilities Lot Config … Pool Area Pool QC Fence  \
0          Lvl    AllPub     Corner …       0.0     NaN   NaN
1          Lvl    AllPub     Inside …       0.0     NaN MnPrv
2          Lvl    AllPub     Corner …       0.0     NaN   NaN
3          Lvl    AllPub     Corner …       0.0     NaN   NaN
4          Lvl    AllPub     Inside …       0.0     NaN MnPrv

  Misc Feature  Misc Val Mo Sold  Yr Sold  Sale Type  Sale Condition  \
0          NaN  0.000000       5     2010        WD           Normal
1          NaN  0.000000       6     2010        WD           Normal
2         Gar2  2.345028       6     2010        WD           Normal
3          NaN  0.000000       4     2010        WD           Normal
4          NaN  0.000000       3     2010        WD           Normal

   SalePrice
0     215000
1     105000
2     172000
3     244000
4     189900
```

```
[5 rows x 80 columns]
```

[71]:
```python
# We now have a larger set of potentially-useful features
df.shape
```

[71]:
```
(2925, 80)
```

[72]:
```python
# There are a *lot* of variables. Let's go back to our saved original data and␣
↪look at how many values are missing for each variable.
df = data
data.isnull().sum().sort_values()
```

[72]:
```
Order               0
Sale Condition      0
Heating QC          0
Central Air         0
1st Flr SF          0
                 …
Fireplace Qu     1422
Fence            2354
Alley            2727
Misc Feature     2820
Pool QC          2914
Length: 82, dtype: int64
```

Let's pick out just a few numeric columns to illustrate basic feature transformations.

[73]:
```python
smaller_df= df.loc[:,['Lot Area', 'Overall Qual', 'Overall Cond',
                      'Year Built', 'Year Remod/Add', 'Gr Liv Area',
                      'Full Bath', 'Bedroom AbvGr', 'Fireplaces',
                      'Garage Cars','SalePrice']]
```

[74]:
```python
# Now we can look at summary statistics of the subset data
smaller_df.describe().T
```

[74]:
```
                 count           mean           std      min       25%  \
Lot Area        2925.0   10103.583590   7781.999124   1300.0    7438.0
Overall Qual    2925.0       6.088205      1.402953      1.0       5.0
Overall Cond    2925.0       5.563761      1.112262      1.0       5.0
Year Built      2925.0    1971.302906     30.242474   1872.0    1954.0
Year Remod/Add  2925.0    1984.234188     20.861774   1950.0    1965.0
Gr Liv Area     2925.0    1493.978803    486.273646    334.0    1126.0
Full Bath       2925.0       1.564786      0.551386      0.0       1.0
Bedroom AbvGr   2925.0       2.853675      0.827737      0.0       2.0
Fireplaces      2925.0       0.596923      0.645349      0.0       0.0
Garage Cars     2924.0       1.765048      0.759834      0.0       1.0
SalePrice       2925.0  180411.574701  78554.857286  12789.0  129500.0
```

```
                      50%       75%       max
Lot Area           9428.0   11515.0  215245.0
Overall Qual          6.0       7.0      10.0
Overall Cond          5.0       6.0       9.0
Year Built         1973.0    2001.0    2010.0
Year Remod/Add     1993.0    2004.0    2010.0
Gr Liv Area        1441.0    1740.0    3820.0
Full Bath             2.0       2.0       4.0
Bedroom AbvGr         3.0       3.0       8.0
Fireplaces            1.0       1.0       4.0
Garage Cars           2.0       2.0       5.0
SalePrice        160000.0  213500.0  625000.0
```

[75]:
```
smaller_df.describe()
```

[75]:
```
            Lot Area  Overall Qual  Overall Cond   Year Built  Year Remod/Add  \
count    2925.000000   2925.000000   2925.000000  2925.000000     2925.000000
mean    10103.583590      6.088205      5.563761  1971.302906     1984.234188
std      7781.999124      1.402953      1.112262    30.242474       20.861774
min      1300.000000      1.000000      1.000000  1872.000000     1950.000000
25%      7438.000000      5.000000      5.000000  1954.000000     1965.000000
50%      9428.000000      6.000000      5.000000  1973.000000     1993.000000
75%     11515.000000      7.000000      6.000000  2001.000000     2004.000000
max    215245.000000     10.000000      9.000000  2010.000000     2010.000000

       Gr Liv Area    Full Bath  Bedroom AbvGr   Fireplaces  Garage Cars  \
count  2925.000000  2925.000000    2925.000000  2925.000000  2924.000000
mean   1493.978803     1.564786       2.853675     0.596923     1.765048
std     486.273646     0.551386       0.827737     0.645349     0.759834
min     334.000000     0.000000       0.000000     0.000000     0.000000
25%    1126.000000     1.000000       2.000000     0.000000     1.000000
50%    1441.000000     2.000000       3.000000     1.000000     2.000000
75%    1740.000000     2.000000       3.000000     1.000000     2.000000
max    3820.000000     4.000000       8.000000     4.000000     5.000000

           SalePrice
count    2925.000000
mean   180411.574701
std     78554.857286
min     12789.000000
25%    129500.000000
50%    160000.000000
75%    213500.000000
max    625000.000000
```

[76]:
```
smaller_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 2925 entries, 0 to 2929
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Lot Area        2925 non-null   int64
 1   Overall Qual    2925 non-null   int64
 2   Overall Cond    2925 non-null   int64
 3   Year Built      2925 non-null   int64
 4   Year Remod/Add  2925 non-null   int64
 5   Gr Liv Area     2925 non-null   int64
 6   Full Bath       2925 non-null   int64
 7   Bedroom AbvGr   2925 non-null   int64
 8   Fireplaces      2925 non-null   int64
 9   Garage Cars     2924 non-null   float64
 10  SalePrice       2925 non-null   int64
dtypes: float64(1), int64(10)
memory usage: 274.2 KB
```

[77]:

```
# There appears to be one NA in Garage Cars - we will take a simple approach␣
↪and fill it with 0
smaller_df = smaller_df.fillna(0)
```

[78]:

```
smaller_df
```

[78]:

```
      Lot Area  Overall Qual  Overall Cond  Year Built  Year Remod/Add  \
0        31770             6             5        1960            1960
1        11622             5             6        1961            1961
2        14267             6             6        1958            1958
3        11160             7             5        1968            1968
4        13830             5             5        1997            1998
…          …           …             …           …             …
2925      7937             6             6        1984            1984
2926      8885             5             5        1983            1983
2927     10441             5             5        1992            1992
2928     10010             5             5        1974            1975
2929      9627             7             5        1993            1994

      Gr Liv Area  Full Bath  Bedroom AbvGr  Fireplaces  Garage Cars  \
0            1656          1              3           2          2.0
1             896          1              2           0          1.0
2            1329          1              3           0          1.0
3            2110          2              3           2          2.0
4            1629          2              3           1          2.0
…             …          …              …          …           …
2925         1003          1              3           0          2.0
2926          902          1              2           0          2.0
2927          970          1              3           0          0.0
```

```
2928         1389           1                2             1            2.0
2929         2000           2                3             1            3.0

      SalePrice
0        215000
1        105000
2        172000
3        244000
4        189900
…          …
2925     142500
2926     131000
2927     132000
2928     170000
2929     188000

[2925 rows x 11 columns]
```

[79]:
```python
smaller_df.info()
```

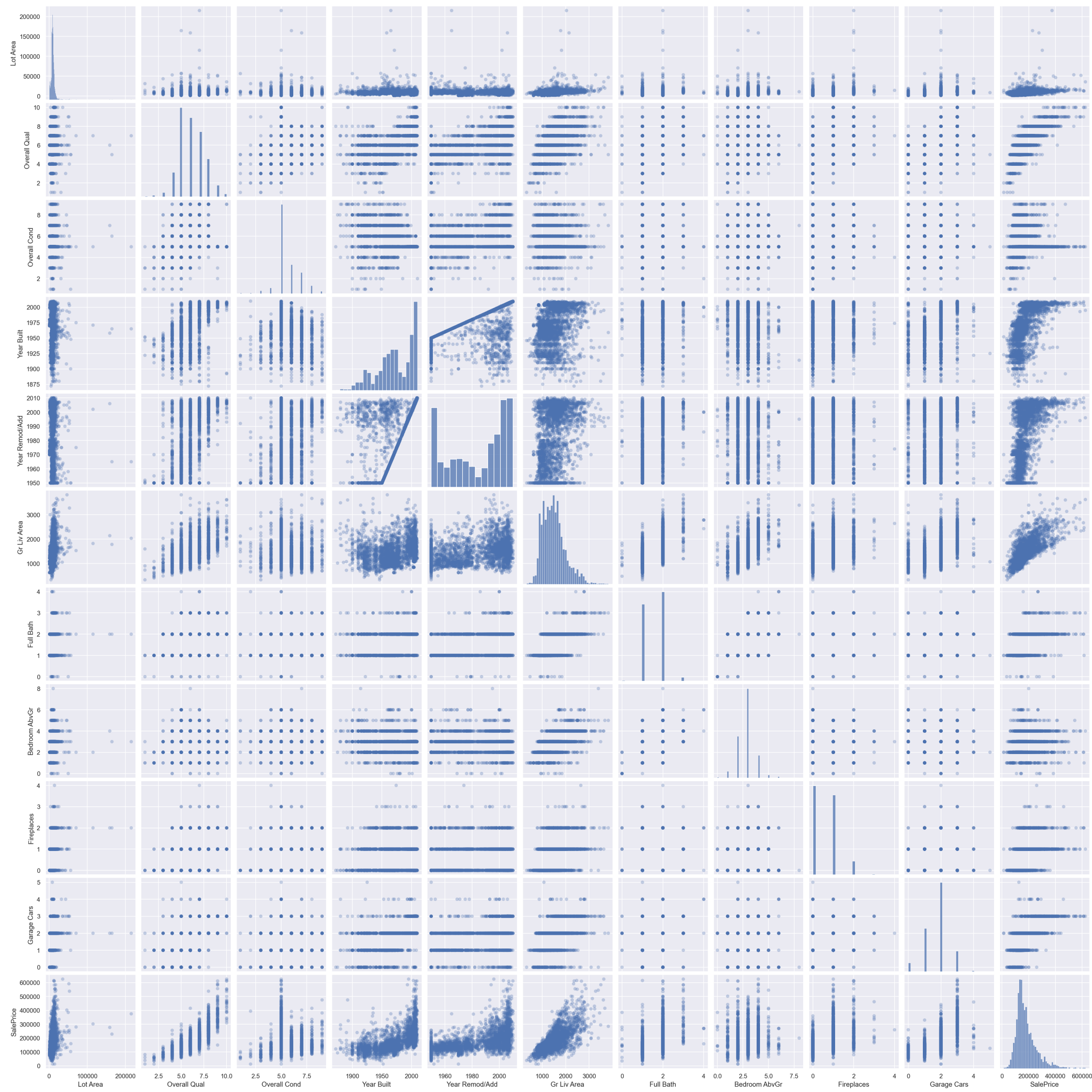
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2925 entries, 0 to 2929
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Lot Area        2925 non-null   int64
 1   Overall Qual    2925 non-null   int64
 2   Overall Cond    2925 non-null   int64
 3   Year Built      2925 non-null   int64
 4   Year Remod/Add  2925 non-null   int64
 5   Gr Liv Area     2925 non-null   int64
 6   Full Bath       2925 non-null   int64
 7   Bedroom AbvGr   2925 non-null   int64
 8   Fireplaces      2925 non-null   int64
 9   Garage Cars     2925 non-null   float64
 10  SalePrice       2925 non-null   int64
dtypes: float64(1), int64(10)
memory usage: 274.2 KB
```

0.2.3 Pair plot of features

Now that we have a nice, filtered dataset, let's generate visuals to better understand the target and feature-target relationships: pairplot is great for this!

[80]:
```python
sns.pairplot(smaller_df, plot_kws=dict(alpha=.3, edgecolor='none'))
```

[80]: <seaborn.axisgrid.PairGrid at 0x20bbacbab20>



Data Exploration Discussion:

1. What do these plots tell us about the distribution of the target?

2. What do these plots tell us about the relationship between the features and the target? Do you think that linear regression is well-suited to this problem? Do any feature transformations come to mind?

3. What do these plots tell us about the relationship between various pairs of features? Do you think there may be any problems here?

Suppose our target variable is the SalePrice. We can set up separate variables for features and target.

[81]:
```
#Separate our features from our target

X = smaller_df.loc[:,['Lot Area', 'Overall Qual', 'Overall Cond',
                      'Year Built', 'Year Remod/Add', 'Gr Liv Area',
                      'Full Bath', 'Bedroom AbvGr', 'Fireplaces',
                      'Garage Cars']]

y = smaller_df['SalePrice']
```

[82]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2925 entries, 0 to 2929
Data columns (total 10 columns):
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Lot Area        2925 non-null   int64
 1   Overall Qual    2925 non-null   int64
 2   Overall Cond    2925 non-null   int64
 3   Year Built      2925 non-null   int64
 4   Year Remod/Add  2925 non-null   int64
 5   Gr Liv Area     2925 non-null   int64
 6   Full Bath       2925 non-null   int64
 7   Bedroom AbvGr   2925 non-null   int64
 8   Fireplaces      2925 non-null   int64
 9   Garage Cars     2925 non-null   float64
dtypes: float64(1), int64(9)
memory usage: 251.4 KB
```

Now that we have feature/target data X, y ready to go, we're nearly ready to fit and evaluate a baseline model using our current feature set. We'll need to create a **train/validation split** before we fit and score the model.

Since we'll be repeatedly splitting X, y into the same train/val partitions and fitting/scoring new models as we update our feature set, we'll define a reusable function that completes all these steps, making our code/process more efficient going forward.

Great, let's go ahead and run this function on our baseline feature set and take some time to analyze the results.

0.2.4 Basic feature engineering: adding polynomial and interaction terms

One of the first things that we looked for in the pairplot was evidence about the relationship between each feature and the target. In certain features like *'Overall Qual'* and *'Gr Liv Qual'*, we notice an upward-curved relationship rather than a simple linear correspondence. This suggests that we should add quadratic **polynomial terms or transformations** for those features, allowing us to express that non-linear relationship while still using linear regression as our model.

Luckily, pandas makes it quite easy to quickly add those square terms as additional features to our original feature set. We'll do so and evaluate our model again below.

As we add to our baseline set of features, we'll create a copy of the latest benchmark so that we can continue to store our older feature sets. ### Polynomial Features

[83]:
```
X2 = X.copy()

X2['OQ2'] = X2['Overall Qual'] ** 2
X2['GLA2'] = X2['Gr Liv Area'] ** 2
```

[85]:
```
X2
```

[85]:
```
      Lot Area  Overall Qual  Overall Cond  Year Built  Year Remod/Add  \
0        31770             6             5        1960            1960
1        11622             5             6        1961            1961
```

```
2       14267           6            6        1958              1958
3       11160           7            5        1968              1968
4       13830           5            5        1997              1998
…       …            …           …        …             …
2925     7937           6            6        1984              1984
2926     8885           5            5        1983              1983
2927    10441           5            5        1992              1992
2928    10010           5            5        1974              1975
2929     9627           7            5        1993              1994

      Gr Liv Area  Full Bath  Bedroom AbvGr  Fireplaces  Garage Cars  OQ2  \
0            1656          1              3           2          2.0   36
1             896          1              2           0          1.0   25
2            1329          1              3           0          1.0   36
3            2110          2              3           2          2.0   49
4            1629          2              3           1          2.0   25
…             …          …             …          …          …  …
2925         1003          1              3           0          2.0   36
2926          902          1              2           0          2.0   25
2927          970          1              3           0          0.0   25
2928         1389          1              2           1          2.0   25
2929         2000          2              3           1          3.0   49

         GLA2
0     2742336
1      802816
2     1766241
3     4452100
4     2653641
…       …
2925  1006009
2926   813604
2927   940900
2928  1929321
2929  4000000

[2925 rows x 12 columns]
```

As is, each feature is treated as an independent quantity. However, there may be **interaction effects**, in which the impact of one feature may dependent on the current value of a different feature.

For example, there may be a higher premium for increasing *'Overall Qual'* for houses that were built more recently. If such a premium or a similar effect exists, a feature that multiplies *'Overall Qual'* by *'Year Built'* can help us capture it.

Another style of interaction term involves feature proprtions: for example, to get at something like quality per square foot we could divide *'Overall Qual'* by *'Lot Area'*.

Let's try adding both of these interaction terms and see how they impact the model results.

0.2.5 Feature interactions

[86]:
```python
X3 = X2.copy()

# multiplicative interaction
X3['OQ_x_YB'] = X3['Overall Qual'] * X3['Year Built']

# division interaction
X3['OQ_/_LA'] = X3['Overall Qual'] / X3['Lot Area']
```

[87]:
```python
X3
```

[87]:
```
      Lot Area  Overall Qual  Overall Cond  Year Built  Year Remod/Add  \
0        31770             6             5        1960            1960
1        11622             5             6        1961            1961
2        14267             6             6        1958            1958
3        11160             7             5        1968            1968
4        13830             5             5        1997            1998
…          …           …             …           …              …
2925      7937             6             6        1984            1984
2926      8885             5             5        1983            1983
2927     10441             5             5        1992            1992
2928     10010             5             5        1974            1975
2929      9627             7             5        1993            1994

      Gr Liv Area  Full Bath  Bedroom AbvGr  Fireplaces  Garage Cars  OQ2  \
0            1656          1              3           2          2.0   36
1             896          1              2           0          1.0   25
2            1329          1              3           0          1.0   36
3            2110          2              3           2          2.0   49
4            1629          2              3           1          2.0   25
…              …          …              …           …           …  …
2925         1003          1              3           0          2.0   36
2926          902          1              2           0          2.0   25
2927          970          1              3           0          0.0   25
2928         1389          1              2           1          2.0   25
2929         2000          2              3           1          3.0   49

         GLA2  OQ_x_YB   OQ_/_LA
0     2742336    11760  0.000189
1      802816     9805  0.000430
2     1766241    11748  0.000421
3     4452100    13776  0.000627
4     2653641     9985  0.000362
…         …        …         …
2925  1006009    11904  0.000756
```

```
2926   813604     9915  0.000563
2927   940900     9960  0.000479
2928  1929321     9870  0.000500
2929  4000000    13951  0.000727

[2925 rows x 14 columns]
```

Interaction Feature Exercise: What other interactions do you think might be helpful? Why?

0.2.6 Categories and features derived from category aggregates

Incorporating **categorical features** into linear regression models is fairly straightforward: we can create a new feature column for each category value, and fill these columns with 1s and 0s to indicate which category is present for each row. This method is called **dummy variables** or **one-hot-encoding**.

We'll first explore this using the *'House Style'* feature from the original dataframe. Before going straight to dummy variables, it's a good idea to check category counts to make sure all categories have reasonable representation.

[90]:
```python
data['House Style']
```

[90]:
```
0       1Story
1       1Story
2       1Story
3       1Story
4       2Story
          …
2925      SLvl
2926    1Story
2927    SFoyer
2928    1Story
```

```
2929    2Story
Name: House Style, Length: 2925, dtype: object
```

[88]:
```
data['House Style'].value_counts()
```

[88]:
```
1Story    1480
2Story     869
1.5Fin     314
SLvl       128
SFoyer      83
2.5Unf      24
1.5Unf      19
2.5Fin       8
Name: House Style, dtype: int64
```

This looks ok, and here's a quick look at how dummy features actually appear:

[92]:
```
pd.get_dummies(df['House Style'], drop_first=True).head(10)
```

[92]:

	1.5Unf	1Story	2.5Fin	2.5Unf	2Story	SFoyer	SLvl
0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	1	0	0	0	0	0
8	0	1	0	0	0	0	0
9	0	0	0	0	1	0	0

We can call `pd.get_dummies()` on our entire dataset to quickly get data with all the original features and dummy variable representation of any categorical features. Let's look at some variable values.

[94]:
```
nbh_counts = df.Neighborhood.value_counts()
nbh_counts
```

[94]:
```
NAmes      443
CollgCr    267
OldTown    239
Edwards    191
Somerst    182
NridgHt    166
Gilbert    165
Sawyer     151
NWAmes     131
SawyerW    125
```

```
Mitchel    114
BrkSide    108
Crawfor    103
IDOTRR      93
Timber      72
NoRidge     69
StoneBr     51
SWISU       48
ClearCr     44
MeadowV     37
BrDale      30
Blmngtn     28
Veenker     24
NPkVill     23
Blueste     10
Greens       8
GrnHill      2
Landmrk      1
Name: Neighborhood, dtype: int64
```

For this category, let's map the few least-represented neighborhoods to an "other" category before adding the feature to our feature set and running a new benchmark.

[97]:
```
nbh_counts[nbh_counts <= 8].index
```

[97]:
```
Index(['Greens', 'GrnHill', 'Landmrk'], dtype='object')
```

[98]:
```
other_nbhs = list(nbh_counts[nbh_counts <= 8].index)

other_nbhs
```

[98]:
```
['Greens', 'GrnHill', 'Landmrk']
```

[99]:
```
X4 = X3.copy()

X4['Neighborhood'] = df['Neighborhood'].replace(other_nbhs, 'Other')
```

[100]:
```
X4.Neighborhood.value_counts()
```

[100]:
```
NAmes      443
CollgCr    267
OldTown    239
Edwards    191
Somerst    182
NridgHt    166
Gilbert    165
Sawyer     151
NWAmes     131
```

```
SawyerW     125
Mitchel     114
BrkSide     108
Crawfor     103
IDOTRR       93
Timber       72
NoRidge      69
StoneBr      51
SWISU        48
ClearCr      44
MeadowV      37
BrDale       30
Blmngtn      28
Veenker      24
NPkVill      23
Other        11
Blueste      10
Name: Neighborhood, dtype: int64
```

Getting to fancier features Let's close out our introduction to feature engineering by considering a more complex type of feature that may work very nicely for certain problems. It doesn't seem to add a great deal over what we have so far, but it's a style of engineering to keep in mind for the future.

We'll create features that capture where a feature value lies relative to the members of a category it belongs to. In particular, we'll calculate deviance of a row's feature value from the mean value of the category that row belongs to. This helps to capture information about a feature relative to the category's distribution, e.g. how nice a house is relative to other houses in its neighborhood or of its style.

Below we define reusable code for generating features of this form, feel free to repurpose it for future feature engineering work!

[103]:
```
def add_deviation_feature(X, feature, category):

    # temp groupby object
    category_gb = X.groupby(category)[feature]

    # create category means and standard deviations for each observation
    category_mean = category_gb.transform(lambda x: x.mean())
    category_std = category_gb.transform(lambda x: x.std())

    # compute stds from category mean for each feature value,
    # add to X as new feature
    deviation_feature = (X[feature] - category_mean) / category_std
    X[feature + '_Dev_' + category] = deviation_feature
```

And now let's use our feature generation code to add 2 new deviation features, and run a final benchmark.

[104]:

```
X5 = X4.copy()
X5['House Style'] = df['House Style']
add_deviation_feature(X5, 'Year Built', 'House Style')
add_deviation_feature(X5, 'Overall Qual', 'Neighborhood')
```

[106]:

```
X5
```

[106]:

```
      Lot Area  Overall Qual  Overall Cond  Year Built  Year Remod/Add  \
0        31770             6             5        1960            1960
1        11622             5             6        1961            1961
2        14267             6             6        1958            1958
3        11160             7             5        1968            1968
4        13830             5             5        1997            1998
…          …           …            …          …               …
2925      7937             6             6        1984            1984
2926      8885             5             5        1983            1983
2927     10441             5             5        1992            1992
2928     10010             5             5        1974            1975
2929      9627             7             5        1993            1994

      Gr Liv Area  Full Bath  Bedroom AbvGr  Fireplaces  Garage Cars  OQ2  \
0            1656          1              3           2          2.0   36
1             896          1              2           0          1.0   25
2            1329          1              3           0          1.0   36
3            2110          2              3           2          2.0   49
4            1629          2              3           1          2.0   25
…             …          …              …           …           …  …
2925         1003          1              3           0          2.0   36
2926          902          1              2           0          2.0   25
2927          970          1              3           0          0.0   25
2928         1389          1              2           1          2.0   25
2929         2000          2              3           1          3.0   49

         GLA2  OQ_x_YB   OQ_/_LA Neighborhood House Style  \
0     2742336    11760  0.000189        NAmes      1Story
1      802816     9805  0.000430        NAmes      1Story
2     1766241    11748  0.000421        NAmes      1Story
3     4452100    13776  0.000627        NAmes      1Story
4     2653641     9985  0.000362      Gilbert      2Story
…        …       …        …          …          …
2925  1006009    11904  0.000756      Mitchel        SLvl
2926   813604     9915  0.000563      Mitchel      1Story
2927   940900     9960  0.000479      Mitchel      SFoyer
2928  1929321     9870  0.000500      Mitchel      1Story
2929  4000000    13951  0.000727      Mitchel      2Story

      Year Built_Dev_House Style  Overall Qual_Dev_Neighborhood
```

```
0                      -0.590334              0.857503
1                      -0.551186             -0.430205
2                      -0.668629              0.857503
3                      -0.277154              2.145211
4                       0.545208             -2.101974
…                           …                     …
2925                    0.505068              0.434947
2926                    0.310059             -0.518590
2927                    1.096487             -0.518590
2928                   -0.042269             -0.518590
2929                    0.421480              1.388483

[2925 rows x 18 columns]
```

0.3 Polynomial Features in Scikit-Learn

`sklearn` allows you to build many higher-order terms at once with `PolynomialFeatures`

[107]:
```python
from sklearn.preprocessing import PolynomialFeatures
```

[108]:
```python
#Instantiate and provide desired degree;
#Note: degree=2 also includes intercept, degree 1 terms, and cross-terms

pf = PolynomialFeatures(degree=2)
```

[109]:
```python
features = ['Lot Area', 'Overall Qual']
pf.fit(df[features])
```

[109]:
```
PolynomialFeatures()
```

[110]:
```python
pf.get_feature_names()  #Must add input_features = features for appropriate␣
↪names
```

[110]:
```
['1', 'x0', 'x1', 'x0^2', 'x0 x1', 'x1^2']
```

[111]:
```python
feat_array = pf.transform(df[features])
pd.DataFrame(feat_array, columns = pf.
↪get_feature_names(input_features=features))
```

[111]:
```
        1  Lot Area  Overall Qual    Lot Area^2  Lot Area Overall Qual  \
0     1.0   31770.0           6.0  1.009333e+09               190620.0
1     1.0   11622.0           5.0  1.350709e+08                58110.0
2     1.0   14267.0           6.0  2.035473e+08                85602.0
3     1.0   11160.0           7.0  1.245456e+08                78120.0
4     1.0   13830.0           5.0  1.912689e+08                69150.0
…     …       …             …            …                    …
2920  1.0    7937.0           6.0  6.299597e+07                47622.0
2921  1.0    8885.0           5.0  7.894322e+07                44425.0
```

```
2922  1.0   10441.0            5.0  1.090145e+08                 52205.0
2923  1.0   10010.0            5.0  1.002001e+08                 50050.0
2924  1.0    9627.0            7.0  9.267913e+07                 67389.0

      Overall Qual^2
0               36.0
1               25.0
2               36.0
3               49.0
4               25.0
…               …
2920            36.0
2921            25.0
2922            25.0
2923            25.0
2924            49.0

[2925 rows x 6 columns]
```

0.4 Recap

While we haven't yet turned to prediction, these feature engineering exercises set the stage. Generally, feature engineering often follows a sort of *Pareto principle*, where a large bulk of the predictive gains can be reached through adding a set of intuitive, strong features like polynomial transforms and interactions. Directly incorporating additional information like categorical variables can also be very helpful. Beyond this point, additional feature engineering can provide significant, but potentially diminishing returns. Whether it's worth it depends on the use case for the model.