Business Problem

Real Estate firm wants a model to help them accurately price houses.

- How should we price a house based on property features such as lot, footage, bedroom #, bathroom #, and renovations?

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import seaborn as sns
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')

from statsmodels.formula.api import ols
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_err
from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import LabelEncoder,  OneHotEncoder
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Load Data

In [2]:
```
df_column_names = ('data/column_names.md')
df_house = pd.read_csv('data/kc_house_data.csv')
```

In [3]:
```
df_house.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0

5 rows × 21 columns

- id - unique identifier for a house

- date - Date house was sold
- price - Price is prediction target
- bedrooms - Number of Bedrooms/House
- bathrooms - Number of bathrooms/bedrooms
- sqft_livingsquare - footage of the home
- sqft_lotsquare - footage of the lot
- floorsTotal - floors (levels) in house
- waterfront - House which has a view to a waterfront
- view - # of views
- condition - How good the condition is (Overall)
- grade - overall grade given to the housing unit, based on King County grading system
- sqft_above - square footage of house apart from basement
- sqft_basement - square footage of the basement
- yr_built - Built Year
- yr_renovated - Year when house was renovated
- zipcode - zip
- lat - Latitude coordinate
- long - Longitude coordinate
- sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
- sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

Data Exploration

In [4]:
```
df_house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  object
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  int64
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  object
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
```

Fields with nulls: waterfront, view, yr_renovated

In [5]:
```
# print top 5 most frequent values in each column
for col in df_house.columns:
    print(col, '\n', df_house[col].value_counts(normalize=True).head(), '\n
```

```
condition
 3    0.649164
4    0.262861
5    0.078761
2    0.007871
1    0.001343
Name: condition, dtype: float64


grade
 7     0.415521
8     0.280826
9     0.121082
6     0.094365
10    0.052507
Name: grade, dtype: float64


sqft_above
 1300    0.009816
```

view is mostly zeros, yr_renovated is mostly zero, waterfront is mostly zero, sqft_basement mostly zero and '?'

In [6]:
```
df_house['waterfront'].value_counts(normalize=True)
# Make waterfront boolean where nulls are non waterfront properties
```

Out[6]:
```
0.0    0.992404
1.0    0.007596
Name: waterfront, dtype: float64
```

In [7]:
```
df_house['view'].value_counts(normalize=True)
# Make view boolean where nulls are properties without a view
```

Out[7]:
```
0.0    0.901923
2.0    0.044441
3.0    0.023591
1.0    0.015325
4.0    0.014721
Name: view, dtype: float64
```

In [8]:
```
df_house['yr_renovated'].value_counts(normalize=True)
# Make yr_renovated boolean where nulls are unrenovated properties
```

Out[8]:
```
0.0       0.958096
2014.0    0.004112
2003.0    0.001746
2013.0    0.001746
2007.0    0.001690
            ...
1946.0    0.000056
1959.0    0.000056
1971.0    0.000056
1951.0    0.000056
1954.0    0.000056
Name: yr_renovated, Length: 70, dtype: float64
```

In [9]:
```
# check all unique values for 'grade'
grade_values = list(df_house['grade'].unique())
grade_values.sort()
grade_values
```

Out[9]:
```
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```

In [10]:
```
# check all unique values for 'condition'
condition_values = list(df_house['condition'].unique())
condition_values.sort()
condition_values
```

Out[10]:
```
[1, 2, 3, 4, 5]
```

In [11]:
```
# check for identical home
sum(df_house.duplicated(subset=['id']))
```

Out[11]:
```
177
```

In [12]:
```
# check for identical home / sale date rows
sum(df_house.duplicated(subset=['id','date']))
```

Out[12]:
```
0
```

In [13]:
```
df_house['date'].value_counts(normalize=True)
```

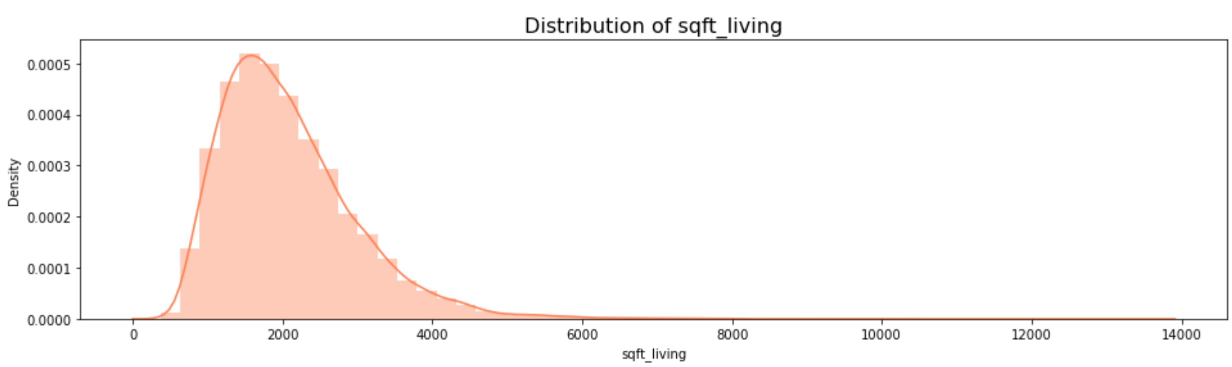
Out[13]:
```
6/23/2014     0.006575
6/25/2014     0.006066
6/26/2014     0.006066
7/8/2014      0.005880
4/27/2015     0.005834
                ...
8/30/2014     0.000046
11/30/2014    0.000046
1/31/2015     0.000046
5/17/2014     0.000046
1/17/2015     0.000046
Name: date, Length: 372, dtype: float64
```

Only houses sold in 2014 and 2015 are in our data. This will limit how successful our model will be when used to predict pricing in the 2021 market.

In [14]:
```
# Plot of the target column price
plt.figure(figsize = (16, 4))
sns.distplot(a = df_house["price"], color = "#FF7F50")
plt.title("Distribution of Price", fontsize=16);
```

skewed distribution

In [15]:
```
# Plot of the sqft_living column
plt.figure(figsize = (16, 4))
sns.distplot(a = df_house["sqft_living"], color = "#FF7F50")
plt.title("Distribution of sqft_living", fontsize=16);
```



skewed distribution

Should look at the effect of log of price and log of sqft_living on the skew in data prep

Data Preparation

Remove unecessary features

Before removing features we will want to run tests to see which features are highly correlated or important to the model. However there are four columns I believe are reasonable to remove.

Columns to remove:

- id - unique identifier for a house
  - not needed for modeling
- lat - Latitude coordinate
- long - Longitude coordinate
  - to simplify analysis we will use zipcode for location information

In [16]:
```
df_house = df_house.drop(['id', 'lat', 'long'], axis=1)
```

Missing Values

In [17]:
```
# Fields with nulls: waterfront, view,  yr_renovated
df_house.waterfront.fillna(0, inplace=True)
df_house.view.fillna(0, inplace=True)
df_house.yr_renovated.fillna(0, inplace=True)
```

Reformat Data and Feature Engineer

New Feature - season_sold

In [18]:
```
# create a year column instead of date
df_house['year_sold'] = df_house['date'].apply(lambda x: int(x[-4:]))
df_house['year_sold'].value_counts(normalize=True)
```

Out[18]:
```
2014    0.677038
2015    0.322962
Name: year_sold, dtype: float64
```

In [19]:
```python
# Create function for season of sale to account for seasonality in pricing
def season(month):
    if month == 12 or 1 <= month <= 2:
        season = 'Winter'
    elif 3 <= month <= 5:
        season = 'Spring'
    elif 6 <= month <= 8:
        season = 'Summer'
    else:
        season = 'Fall'
    return season
```

In [20]:
```python
# to use our function convert date to datetime
df_house['date'] = pd.to_datetime(df_house['date'])
# find month_sold
df_house['month_sold'] = df_house['date'].dt.month
```

In [21]:
```python
df_house['season_sold'] = df_house['month_sold'].apply(season)
df_house['season_sold'].value_counts(normalize=True)
```

Out[21]:
```
Spring    0.301801
Summer    0.293004
Fall      0.234107
Winter    0.171089
Name: season_sold, dtype: float64
```

More houses sold in Spring and Summer than Fall and Winter. This could mean that houses sell at higher prices in Spring and Summer. Intuitively this makes sense based on general knowledge of housing markets.

In [22]:
```python
df_house = df_house.drop(['date','month_sold'], axis=1)
```

New Feature - basement

In [23]:
```python
# Convert sqft_basement col to basement col which indicates whether or not
# assume '0.0' and '?' values mean no basement
df_house['sqft_basement'] = df_house['sqft_basement'].map(lambda x: 0 if x
# convert column to float
df_house['sqft_basement'] = df_house['sqft_basement'].astype('float')
# add column called basement
df_house['basement'] = df_house['sqft_basement'].map(lambda x: 1 if x > 0 e
# remove the 'sqft_basement' column
df_house = df_house.drop(['sqft_basement'], axis=1)
```

Update Feature - scale grade

In [24]:
```
# Change scale of grade to 0 - 10 so it's more intuitive
df_house['grade'] = df_house['grade'].map(lambda x: x-3)
grade_values = list(df_house['grade'].unique())
grade_values.sort()
grade_values
```

Out[24]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

New Feature - renovated_in_2000s

In [25]:
```
# define function for year renovated only counting houses renovated 2000 or
def renovation(year):
    if year >= 2000:
        return 1
    else:
        return 0
```

In [26]:
```
# apply function to the 'yr_renovated' column
df_house['yr_renovated']  = df_house['yr_renovated'].apply(renovation)

# rename column
df_house.rename({'yr_renovated': 'renovated_in_2000s'}, axis=1, inplace=Tru
df_house['renovated_in_2000s'].value_counts()
```

Out[26]:
```
0    21218
1      379
Name: renovated_in_2000s, dtype: int64
```

New Feature - viewed

In [27]:
```
# binary classification of view
def views(count):
    if count > 0:
        return 1
    else:
        return 0
```

In [28]:
```
# apply function to 'view' column
df_house['view'] = df_house['view'].apply(views)
# rename view column
df_house.rename({'view': 'viewed'}, axis=1, inplace=True)
```

In [29]:
```
df_house['viewed'].value_counts(normalize=True)
```

Out[29]:
```
0    0.902209
1    0.097791
Name: viewed, dtype: float64
```

In [30]:
```
df_house
```

3	1.00	1180	5650	1.0	0.0	0	3	4	1180	1955
3	2.25	2570	7242	2.0	0.0	0	3	4	2170	1951
2	1.00	770	10000	1.0	0.0	0	3	3	770	1933
4	3.00	1960	5000	1.0	0.0	0	5	4	1050	1965
3	2.00	1680	8080	1.0	0.0	0	3	5	1680	1987
...	...	...	...	...	...	...	...	...	...	...
3	2.50	1530	1131	3.0	0.0	0	3	5	1530	2009
4	2.50	2310	5813	2.0	0.0	0	3	5	2310	2014
2	0.75	1020	1350	2.0	0.0	0	3	4	1020	2009
3	2.50	1600	2388	2.0	0.0	0	3	5	1600	2004
2	0.75	1020	1076	2.0	0.0	0	3	4	1020	2008

ns

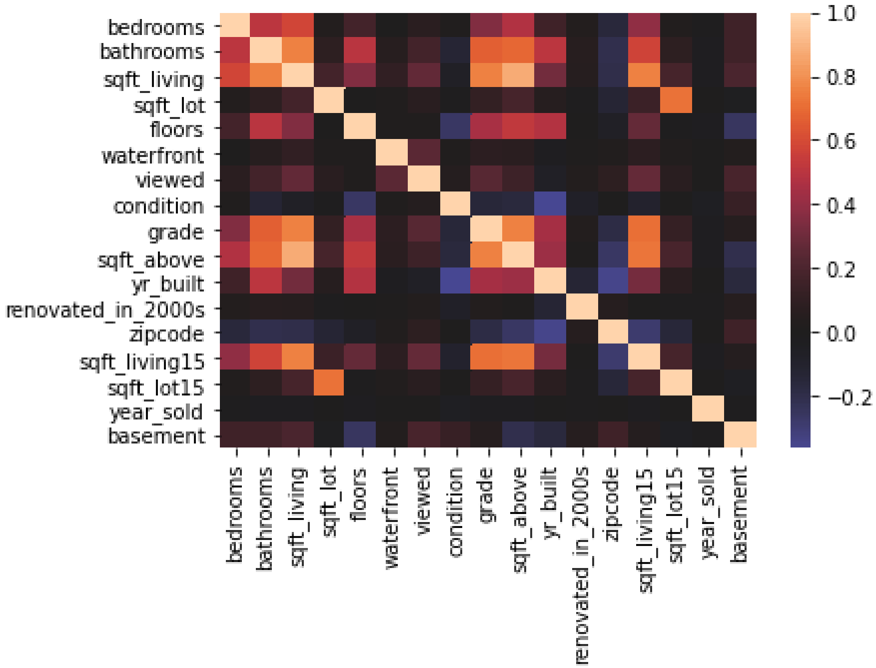
Multicollinearity

In [31]:
```
df_features = df_house.drop(['price'], axis=1)
```

In [32]:
```
fig_num = len(df_features.columns)
```

In [33]:
```
pd.plotting.scatter_matrix(df_features,figsize  = [fig_num, fig_num]);
plt.show()
```

In [34]:
```
sns.heatmap(df_features.corr(), center=0);
```



In [35]:
```
# features correlated above .7 considered highly correlated
abs(df_features.corr()) > 0.7
```

Out[35]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	viewed	condition
bedrooms	True	False	False	False	False	False	False	False
bathrooms	False	True	True	False	False	False	False	False
sqft_living	False	True	True	False	False	False	False	False
sqft_lot	False	False	False	True	False	False	False	False
floors	False	False	False	False	True	False	False	False
waterfront	False	False	False	False	False	True	False	False
viewed	False	False	False	False	False	False	True	False
condition	False	False	False	False	False	False	False	True
grade	False	False	True	False	False	False	False	False
sqft_above	False	False	True	False	False	False	False	False
yr_built	False	False	False	False	False	False	False	False
renovated_in_2000s	False	False	False	False	False	False	False	False
zipcode	False	False	False	False	False	False	False	False
sqft_living15	False	False	True	False	False	False	False	False
sqft_lot15	False	False	False	True	False	False	False	False
year_sold	False	False	False	False	False	False	False	False
basement	False	False	False	False	False	False	False	False

In [36]:
```
ve absolute value of correlation matrix as a data frame
nverts all values to absolute value
acks the row:column pairs into a multindex
set the index to set the multindex to seperate columns
rt values. 0 is the column automatically generated by the stacking

f_features.corr().abs().stack().reset_index().sort_values(0, ascending=False

p the variable name columns (Which were only named level_0 and level_1 by de
pairs'] = list(zip(df.level_0, df.level_1))

t index to pairs
et_index(['pairs'], inplace = True)

p level columns
rop(columns=['level_1', 'level_0'], inplace = True)

name correlation column as cc rather than 0
olumns = ['cc']

op duplicates. This could be dangerous if you have variables perfectly corre
rop_duplicates(inplace=True)
```

In [37]:
```
df[(df.cc>.7) & (df.cc <1)]
```

Out[37]:

pairs	cc
(sqft_living, sqft_above)	0.876448
(sqft_living, grade)	0.762779
(sqft_living15, sqft_living)	0.756402
(grade, sqft_above)	0.756073
(bathrooms, sqft_living)	0.755758
(sqft_above, sqft_living15)	0.731767
(sqft_lot, sqft_lot15)	0.718204
(sqft_living15, grade)	0.713867

- Remove: sqft_above,sqft_living15, sqft_lot15
- Keeping: grade, sqft_living, bathrooms
  - intuitively # of bathrooms, grade, and sqft_living should be important for house pricing so let's keep these for now but create a new feature binning sqft_living

In [38]:
```
# Let's go ahead and drop them from our features
df_features_clean = df_features.drop(['sqft_above', 'sqft_living15', 'sqft_
#And our original dataframe
df_house = df_house.drop(['sqft_above', 'sqft_living15', 'sqft_lot15'], axi
# We're still left with a few pairs with high correlation:
    # grade and sqft_living
    # bathrooms and sqft_living
```

In [39]:
```
# # create new column with bathrooms per square foot
# df_house['bath_per_sqft'] = df_house.apply(lambda row: row['prx'] / row['
```

Outliers

In [40]:
```
print(max(df_house['bedrooms']))
print(min(df_house['bedrooms']))
```

```
33
1
```

In [41]:
```
print(max(df_house['bathrooms']))
print(min(df_house['bathrooms']))
```

```
8.0
0.5
```

In [42]:
```
print(max(df_house['floors']))
print(min(df_house['floors']))
```

```
3.5
1.0
```

In [43]:
```
quantity_features = ['bedrooms', 'bathrooms', 'floors']
```

In [44]:
```
# remove all rows which contain value more than three standard deviations a
for var in quantity_features:
    df_house = df_house[np.abs(df_house[var]-df_house[var].mean()) <= (3*df
```

In [45]:
```
print(max(df_house['bedrooms']))
print(min(df_house['bedrooms']))
```

```
6
1
```

In [46]:
```
print(max(df_house['bathrooms']))
print(min(df_house['bathrooms']))
```

```
4.25
0.5
```

In [47]:
```
print(max(df_house['floors']))
print(min(df_house['floors']))
```

```
3.0
1.0
```

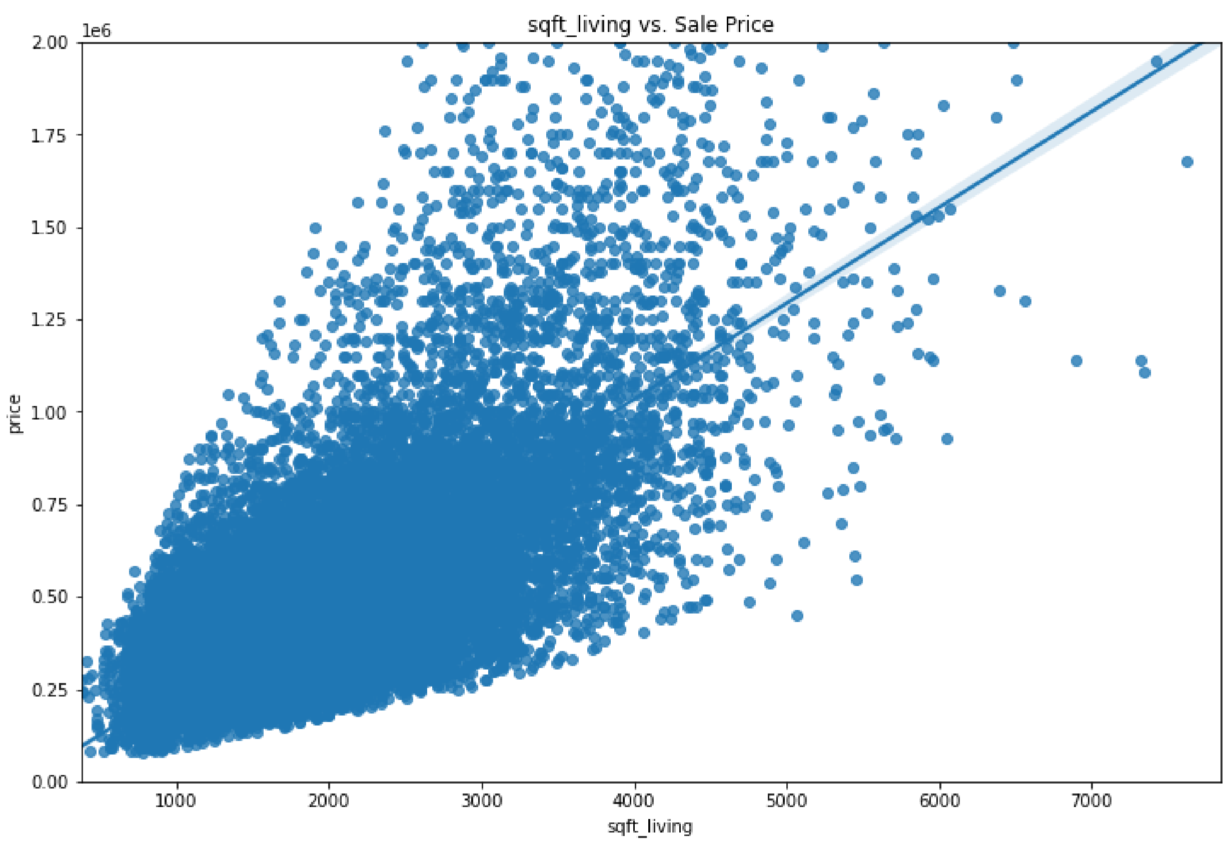
Continuous and Categorical - Feature Engineering cont'

In [48]:
```
df_features_clean.columns
```

Out[48]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'viewed', 'condition', 'grade', 'yr_built',
       'renovated_in_2000s', 'zipcode', 'year_sold', 'season_sold',
       'basement'],
      dtype='object')
```

In [49]:
```
continuous = ['sqft_living']
```

In [50]:
```
# price vs continuous variables
for variable in continuous:
    ax, figure = plt.subplots(1,1,figsize=(12,8))
    plt.ylim(0,2000000)
    sns.regplot(x=variable, y='price', data=df_house)
    plt.title("{} vs. Sale Price".format(variable))
```



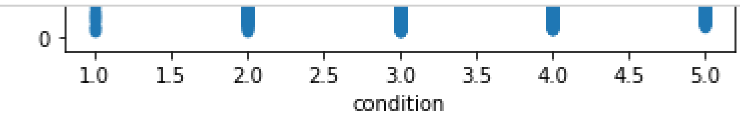
In [51]:
```
# assign categorical variables to a list
categorical = ['bedrooms', 'bathrooms', 'floors', 'waterfront', 'viewed', '
                'yr_built', 'renovated_in_2000s', 'year_sold','sea
                'basement', 'sqft_lot','zipcode']
```

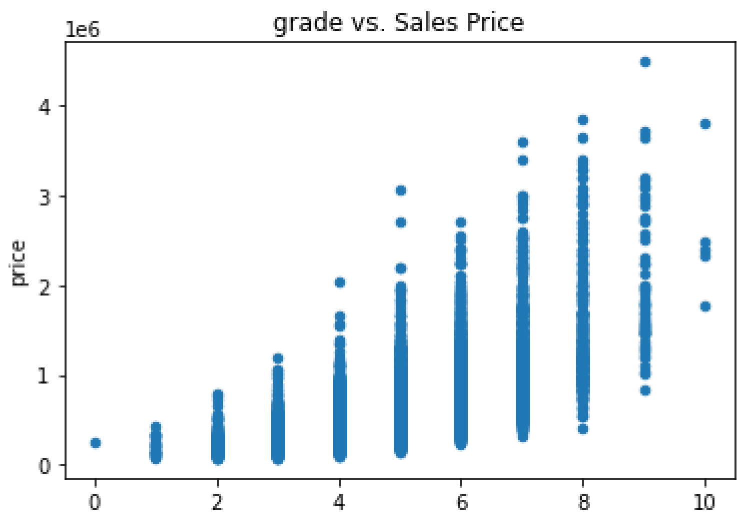
In [52]:
```
#Check that we included all features:
len(categorical) + len(continuous) == len(df_features_clean.columns)
```

Out[52]: True

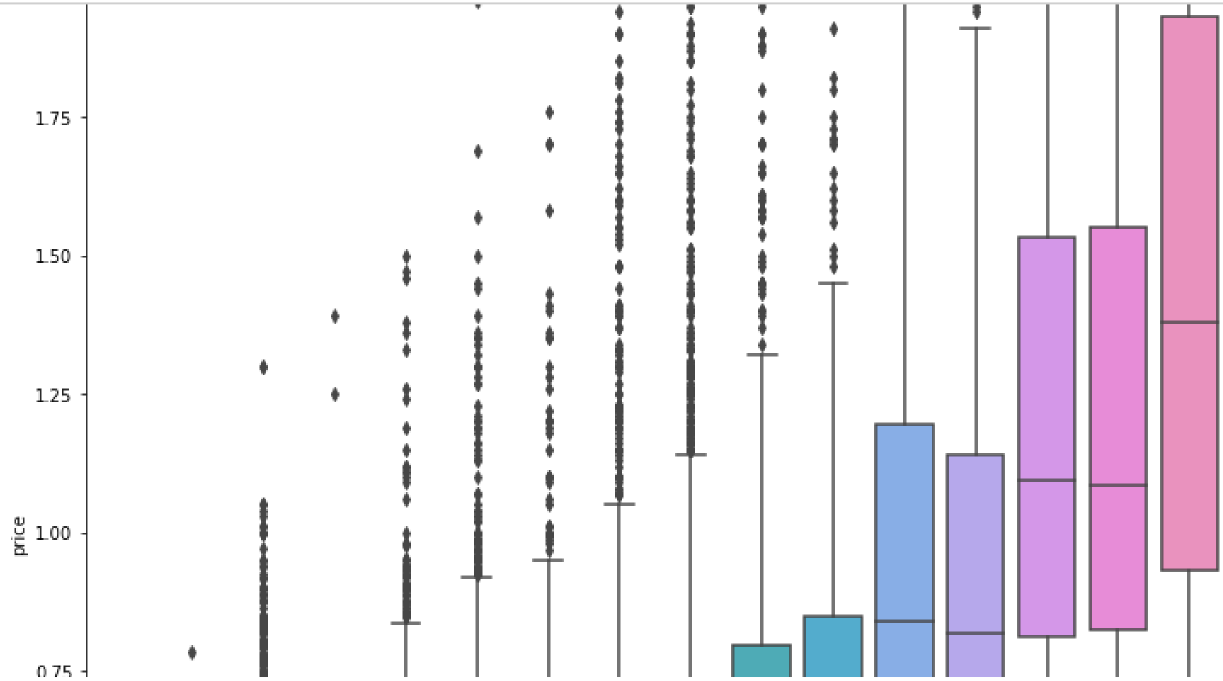
In [53]:
```
# let's take a look at the scatter plots to show how these behave more like

for column in categorical:
    df_house.plot.scatter(x=column, y='price')
    plt.title("{} vs. Sales Price".format(column))
    plt.show()
```





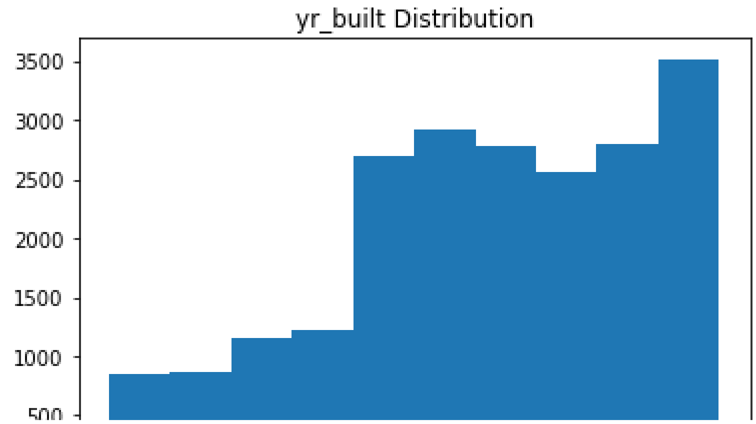
In [54]:
```
# price vs categorical variables
for variable in categorical:
    ax, figure = plt.subplots(1,1,figsize=(12,12))
    plt.ylim(0,2000000)
    sns.boxplot(x=variable, y='price', data=df_house)
    plt.title("{} vs. Sale Price".format(variable))
```



- yr_built is hard to interpret
- year_sold difference in price visually looks negligible
- season_sold Spring/Summer do have slightly higher prices than Fall/Winter

New Features - binned categoricals

In [55]:
```
# Let's look at the histograms for our categorical variables to see if we w
# I've taken out variables which are already boolean or categorized like Se
categorical_bin = ['bedrooms', 'bathrooms', 'floors', 'condition', 'grade',
for column in categorical_bin:
    plt.hist(df_house[column])
    plt.xlabel(column)
    plt.title("{} Distribution".format(column))
    plt.show()
```



In [56]:
```
# and we will also take a look at their summary statistics
categorical_bin = ['bedrooms', 'bathrooms', 'floors', 'condition', 'grade',
for column in categorical_bin:
    print(df_house[column].describe())
```

```
max          10.000000
Name: grade, dtype: float64
count    21357.000000
mean      1970.874233
std         29.343362
min       1900.000000
25%       1951.000000
50%       1975.000000
75%       1996.000000
max       2015.000000
Name: yr_built, dtype: float64
count    2.135700e+04
mean     1.490973e+04
std      4.082896e+04
min      5.200000e+02
25%      5.034000e+03
50%      7.590000e+03
75%      1.058400e+04
max      1.651359e+06
```

In [57]:
```
df_house['sqft_lot'].describe().apply(lambda x: format(x, 'f'))
```

Out[57]:
```
count      21357.000000
mean       14909.726507
std        40828.962049
min          520.000000
25%         5034.000000
50%         7590.000000
75%        10584.000000
max      1651359.000000
Name: sqft_lot, dtype: object
```

In [58]:
```
# based on some intuition and the quantile distributions, we will define ou
bedrooms_bins = [0, 2, 3, 4, 33]
bathrooms_bins = [0, 1, 2, 3, 8]
floors_bins = [0,1,2,4]# when plotted with price 2 floors looks like a high
condition_bins = [0,2,3,5] # when plotted with price condition 2 and under
grade_bins = [0,4,5,7,10]
yr_built_bins = [1900, 1950, 1975, 2000, 2015]
sqft_lot_bins = [0,2500,5000,7500,10000,10000000]
```

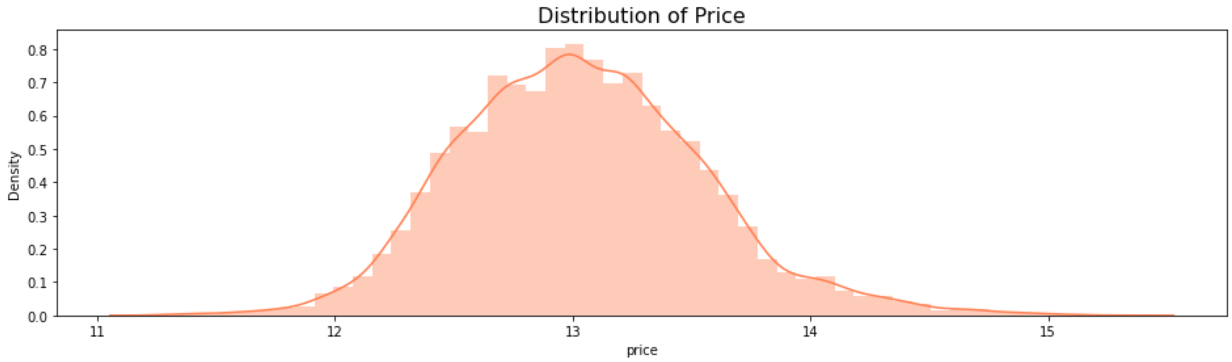
In [59]:
```
# now lets convert these columns to binned categories
# note that when binning these, the default setting is that the values on t
df_house['bedrooms'] = pd.cut(df_house['bedrooms'], bedrooms_bins)
df_house['bathrooms'] = pd.cut(df_house['bathrooms'], bathrooms_bins)
df_house['floors'] = pd.cut(df_house['floors'], floors_bins)
df_house['condition'] = pd.cut(df_house['condition'], condition_bins)
df_house['grade'] = pd.cut(df_house['grade'], grade_bins)
df_house['yr_built'] = pd.cut(df_house['yr_built'], yr_built_bins)
df_house['sqft_lot'] = pd.cut(df_house['sqft_lot'], sqft_lot_bins)
#df_house['zipcode'] = pd.cut(df_house['zipcode'], zipcode_bins)
```

In [60]:
```
# and we will also take a look at the distribution across bins
for column in categorical_bin:
    print(df_house[column].value_counts(normalize=True))
```

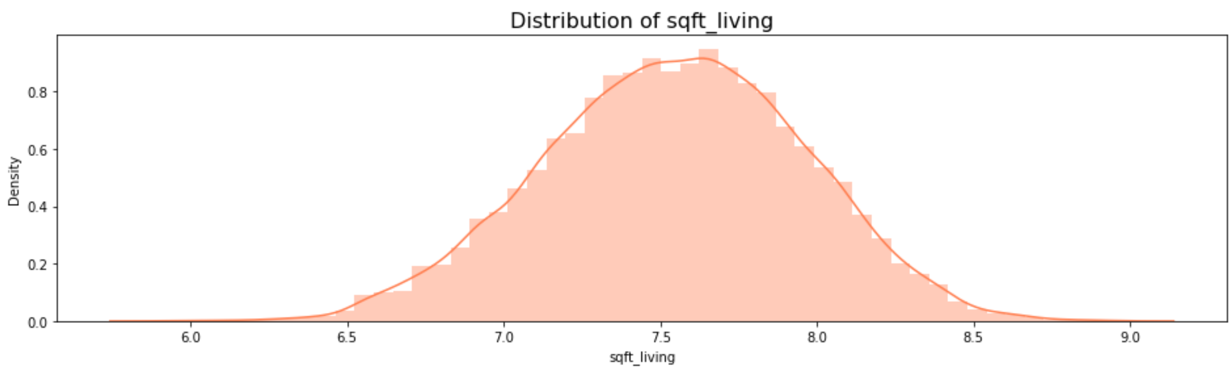
```
(2, 3]     0.459568
(3, 4]     0.319567
(0, 2]     0.138315
(4, 33]    0.082549
Name: bedrooms, dtype: float64
(2, 3]    0.437093
(1, 2]    0.300979
(0, 1]    0.183780
(3, 8]    0.078148
Name: bathrooms, dtype: float64
(0, 1]    0.498291
(1, 2]    0.466498
(2, 4]    0.035211
Name: floors, dtype: float64
(2, 3]    0.647563
(3, 5]    0.343119
(0, 2]    0.009318
Name: condition, dtype: float64
(0, 4]     0.526456
(4, 5]     0.282216
(5, 7]     0.172364
(7, 10]    0.018964
Name: grade, dtype: float64
(1975, 2000]    0.284814
(1950, 1975]    0.268500
(1900, 1950]    0.237847
(2000, 2015]    0.208839
Name: yr_built, dtype: float64
(10000, 10000000]    0.282062
(5000, 7500]         0.244885
(7500, 10000]        0.226905
(2500, 5000]         0.176570
(0, 2500]            0.069579
Name: sqft_lot, dtype: float64
```

Log Transformations

In [61]:
```
non_normal = ['price']
for feat in non_normal:
    df_house[feat] = df_house[feat].map(lambda x: np.log(x))
plt.figure(figsize = (16, 4))
sns.distplot(a = df_house["price"], color = "#FF7F50")
plt.title("Distribution of Price", fontsize=16);
```



In [62]:
```
non_normal_2 = ['sqft_living']
for feat in non_normal_2:
    df_house[feat] = df_house[feat].map(lambda x: np.log(x))
plt.figure(figsize = (16, 4))
sns.distplot(a = df_house["sqft_living"], color = "#FF7F50")
plt.title("Distribution of sqft_living", fontsize=16);
```



split data test and train

In [63]:
```
# Divide dataset into X predictors and y target
x = df_house.drop(['price'], axis=1)
y = df_house[['price']]
```

In [64]:
```
# Split the data into 80% training and 20% test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, sh
```

In [65]:
```
# Validate rows in splits look as expected
print(len(x_train), len(x_test), len(y_train), len(y_test))
```

```
17085 4272 17085 4272
```

one-hot encode categorical variables

In [66]:
```
categorical
```

Out[66]:
```
['bedrooms',
 'bathrooms',
 'floors',
 'waterfront',
 'viewed',
 'condition',
 'grade',
 'yr_built',
 'renovated_in_2000s',
 'year_sold',
 'season_sold',
 'basement',
 'sqft_lot',
 'zipcode']
```

In [67]:
```
# Convert category variables data type
# One-hot encode my categorical variables.
needs_one_hot = categorical.copy()
# These are already one-hot binary
needs_one_hot.remove('waterfront')
needs_one_hot.remove('viewed')
needs_one_hot.remove('renovated_in_2000s')
needs_one_hot.remove('basement')

x_train[needs_one_hot] = x_train[needs_one_hot].astype('category')
x_test[needs_one_hot] = x_test[needs_one_hot].astype('category')
```

In [68]:
```
## one hot encode categoricals training set
x_train_ohe = pd.get_dummies(x_train[needs_one_hot])
x_train_ohe.head()
```

Out[68]:

	bedrooms_(0, 2]	bedrooms_(2, 3]	bedrooms_(3, 4]	bedrooms_(4, 33]	bathrooms_(0, 1]	bathrooms_(1, 2]	batl
16214	0	1	0	0	0	1	
12438	0	1	0	0	0	1	
7668	0	0	0	1	0	1	
14679	0	0	1	0	0	0	
6031	0	0	0	1	0	1	

5 rows × 103 columns

In [69]:
```
## one hot encode categoricals test set
x_test_ohe = pd.get_dummies(x_test[needs_one_hot])
x_test_ohe.head()
```

Out[69]:

	bedrooms_(0, 2]	bedrooms_(2, 3]	bedrooms_(3, 4]	bedrooms_(4, 33]	bathrooms_(0, 1]	bathrooms_(1, 2]	bath
15235	0	1	0	0	1	0	
7564	0	1	0	0	0	1	
18362	1	0	0	0	1	0	
5922	1	0	0	0	0	1	
11844	0	1	0	0	1	0	

5 rows × 103 columns

In [70]:
```
## Concatenate OHE columns with original dataframe, dropping OHE original f
x_train = pd.concat([x_train, x_train_ohe], axis=1)
x_train.drop(['bedrooms', 'bathrooms', 'floors', 'waterfront', 'viewed', 'c
                          'yr_built', 'renovated_in_2000s', 'year_sold','se
                          'basement', 'sqft_lot','zipcode'], axis=1, inplac
x_train.head()
```

Out[70]:

	sqft_living	bedrooms_(0, 2]	bedrooms_(2, 3]	bedrooms_(3, 4]	bedrooms_(4, 33]	bathrooms_(0, 1]	bathroo
16214	7.339538	0	1	0	0	0	
12438	7.533694	0	1	0	0	0	
7668	7.610853	0	0	0	1	0	
14679	7.522941	0	0	1	0	0	
6031	7.138867	0	0	0	1	0	

5 rows × 104 columns

In [71]:
```
## Concatenate OHE columns with original dataframe, dropping OHE original f
x_test = pd.concat([x_test, x_test_ohe], axis=1)
x_test.drop(['bedrooms', 'bathrooms', 'floors', 'waterfront', 'viewed', 'co
                          'yr_built', 'renovated_in_2000s', 'year_sold','se
                          'basement', 'sqft_lot','zipcode'], axis=1, inplac
x_test.head()
```

Out[71]:

	sqft_living	bedrooms_(0, 2]	bedrooms_(2, 3]	bedrooms_(3, 4]	bedrooms_(4, 33]	bathrooms_(0, 1]	bathroo
15235	6.966024	0	1	0	0	1	
7564	7.012115	0	1	0	0	0	
18362	6.917706	1	0	0	0	1	
5922	7.346010	1	0	0	0	0	
11844	6.791221	0	1	0	0	1	

5 rows × 104 columns

Model Iteration Variables

In [72]:
```
# create list so that random forest model important features are named
rfm_columns = list(x_train.columns)
# create list so that ols parameters in results summary are named
xname_columns = list(x_train.columns)
```

In [73]:
```
#add const to beginning of list for ols param names
xname_columns.insert(0,'const')
```

In [74]:
```
#feature selection for second ols model
xname2_columns = xname_columns.copy()

# items to be removed
unwanted_feats = {'grade_(4, 5]', 'grade_(5, 7]','grade_(7, 10]','floors_(2
                        'yr_built_(2000, 2015]','season_sold_Fall','se
                        'zipcode_98024','zipcode_98070','zipcode_98072

xname2_columns = [ele for ele in xname2_columns if ele not in unwanted_feat

# printing modified list
print("New list after removing unwanted feats: ", xname2_columns)
```

```
New list after removing unwanted feats:  ['const', 'sqft_living', 'bedroo
ms_(0, 2]', 'bedrooms_(2, 3]', 'bedrooms_(3, 4]', 'bedrooms_(4, 33]', 'ba
throoms_(0, 1]', 'bathrooms_(1, 2]', 'bathrooms_(2, 3]', 'bathrooms_(3,
8]', 'floors_(0, 1]', 'floors_(1, 2]', 'condition_(0, 2]', 'condition_(2,
3]', 'condition_(3, 5]', 'grade_(0, 4]', 'yr_built_(1950, 1975]', 'yr_bui
lt_(1975, 2000]', 'year_sold_2014', 'year_sold_2015', 'season_sold_Sprin
g', 'season_sold_Winter', 'sqft_lot_(0, 2500]', 'sqft_lot_(2500, 5000]',
'sqft_lot_(7500, 10000]', 'sqft_lot_(10000, 10000000]', 'zipcode_98001',
'zipcode_98002', 'zipcode_98003', 'zipcode_98004', 'zipcode_98005', 'zipc
ode_98006', 'zipcode_98007', 'zipcode_98008', 'zipcode_98010', 'zipcode_9
8011', 'zipcode_98014', 'zipcode_98019', 'zipcode_98022', 'zipcode_9802
3', 'zipcode_98027', 'zipcode_98028', 'zipcode_98029', 'zipcode_98030',
'zipcode_98031', 'zipcode_98032', 'zipcode_98033', 'zipcode_98034', 'zipc
ode_98038', 'zipcode_98039', 'zipcode_98040', 'zipcode_98042', 'zipcode_9
8045', 'zipcode_98052', 'zipcode_98053', 'zipcode_98055', 'zipcode_9805
6', 'zipcode_98058', 'zipcode_98059', 'zipcode_98065', 'zipcode_98074',
'zipcode_98075', 'zipcode_98092', 'zipcode_98102', 'zipcode_98103', 'zipc
ode_98105', 'zipcode_98106', 'zipcode_98107', 'zipcode_98108', 'zipcode_9
8109', 'zipcode_98112', 'zipcode_98115', 'zipcode_98116', 'zipcode_9811
7', 'zipcode_98118', 'zipcode_98119', 'zipcode_98122', 'zipcode_98125',
'zipcode_98126', 'zipcode_98136', 'zipcode_98144', 'zipcode_98146', 'zipc
ode_98148', 'zipcode_98155', 'zipcode_98166', 'zipcode_98168', 'zipcode_9
8177', 'zipcode_98178', 'zipcode_98188', 'zipcode_98198', 'zipcode_9819
9']
```

In [75]:
```
# Features for second iteration of Random Forest Model - train
rf2_feats = ['sqft_living', 'grade_(0, 4]', 'yr_built_(1900, 1950]', 'grade
             'sqft_lot_(2500, 5000]', 'grade_(7, 10]','zipcode_98092','zipc
             'yr_built_(1950, 1975]','zipcode_98033','zipcode_98038','zipco
             'zipcode_98115','zipcode_98040']
x_train_rf2 = x_train[rf2_feats]

# Features for second iteration of Random Forest Model - test
x_test_rf2 = x_test[rf2_feats]
```

In [76]:
```
x_train.head()
```

Out[76]:

	sqft_living	bedrooms_(0, 2]	bedrooms_(2, 3]	bedrooms_(3, 4]	bedrooms_(4, 33]	bathrooms_(0, 1]	bathroo
16214	7.339538	0	1	0	0	0	
12438	7.533694	0	1	0	0	0	
7668	7.610853	0	0	0	1	0	
14679	7.522941	0	0	1	0	0	
6031	7.138867	0	0	0	1	0	

5 rows × 104 columns

In [77]:
```
x_train_ols2 = x_train.drop(['grade_(4, 5]', 'grade_(5, 7]','grade_(7, 10]'
                             'yr_built_(2000, 2015]','season_sold_Fall','se
                             'zipcode_98024','zipcode_98070','zipcode_98072
x_test_ols2 = x_test.drop(['grade_(4, 5]', 'grade_(5, 7]','grade_(7, 10]','
                             'yr_built_(2000, 2015]','season_sold_Fall','se
                             'zipcode_98024','zipcode_98070','zipcode_98072
x_test_ols2.head()
```

Out[77]:

	sqft_living	bedrooms_(0, 2]	bedrooms_(2, 3]	bedrooms_(3, 4]	bedrooms_(4, 33]	bathrooms_(0, 1]	bathroo
15235	6.966024	0	1	0	0	1	
7564	7.012115	0	1	0	0	0	
18362	6.917706	1	0	0	0	1	
5922	7.346010	1	0	0	0	0	
11844	6.791221	0	1	0	0	1	

5 rows × 90 columns

Feature Scaling

In [78]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [79]:
```
x_train_ols2 = scaler.fit_transform(x_train_ols2)
x_test_ols2 = scaler.transform(x_test_ols2)
```

Model 1

Linear Regression

In [80]:
```
linear_model = LinearRegression()
# Train the model on training data
linear_model.fit(x_train, y_train)
```

Out[80]:
```
LinearRegression()
```

Linear model prediction

In [81]:
```
# Predict on test data
predictions = linear_model.predict(x_test)
mse = mean_squared_error(y_test, predictions)
rmse = np.sqrt(mse)
print(rmse)
```

```
0.19803609919251683
```

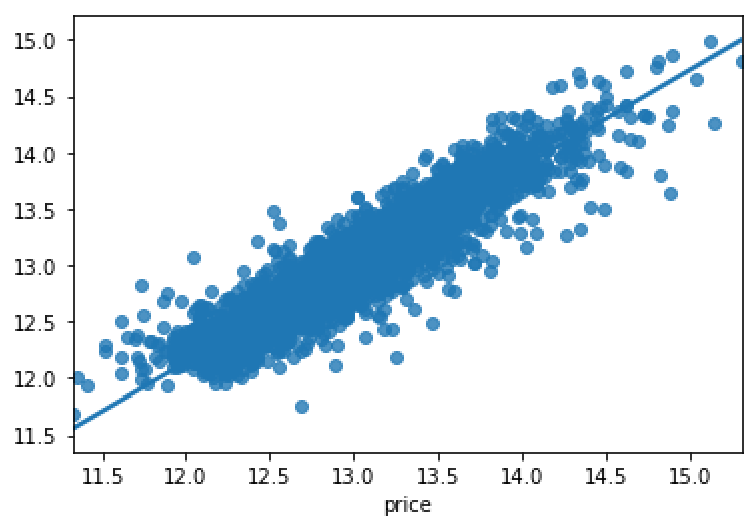
In [82]:
```
# Get how well it performed
mae_linear = mean_absolute_error(y_test, predictions)

print("Linear: {:,}".format(mae_linear))
```

```
Linear: 0.14423826491281666
```

In [83]:
```
sns.regplot(x=y_test, y=predictions)
```

Out[83]:
```
<AxesSubplot:xlabel='price'>
```



In [84]:
```
y_hat_train = linear_model.predict(x_train)
y_hat_test = linear_model.predict(x_test)
```

In [85]:
```
train_residuals = y_hat_train - y_train
test_residuals = y_hat_test - y_test
```

In [86]:
```
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 0.038139886774464414
Test Mean Squarred Error: 0.039218296583388365
```

In [87]:
```
# Train Mean Squarred Error: 0.098529840280035 61
# Test Mean Squarred Error: 0.0995161577140719
```

In [88]:
```
print('R^2 train: %.3f, test: %.3f' % (r2_score(y_train, y_hat_train),
                r2_score(y_test, y_hat_test)))
```

```
R^2 train: 0.856, test: 0.850
```

In [89]:
```
#R^2 train: 0.629, test: 0.620
```

Model 2

Decision Tree Regressor

In [90]:
```
tree_model = DecisionTreeRegressor()

# Train the model on training data
tree_model.fit(x_train, y_train)

# Predict on test data
predictions = tree_model.predict(x_test)
```

Tree model prediction

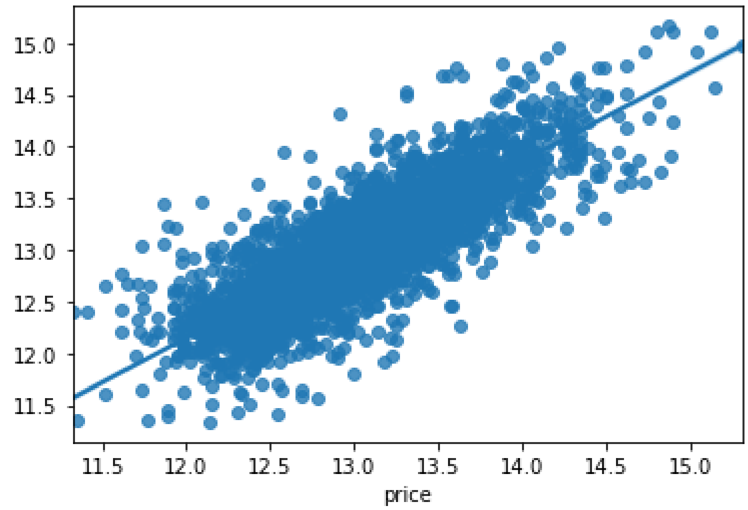
In [91]:
```
# Get how well it performed
mae_tree = mean_absolute_error(y_test, predictions)

print("Tree: {:,}".format(mae_tree))
```

```
Tree: 0.20733758793534132
```

In [92]:
```
sns.regplot(x=y_test, y=predictions)
```

Out[92]: <AxesSubplot:xlabel='price'>



In [93]:
```
y_hat_train = tree_model.predict(x_train)
y_hat_test = tree_model.predict(x_test)
```

In [94]:
```
#train_residuals = y_hat_train - y_train
#test_residuals = y_hat_test - y_test
```

In [95]:
```
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 0.00016094854427852004
Test Mean Squarred Error: 0.0848163992356105
```

In [96]:
```
print('R^2 train: %.3f, test: %.3f' % (r2_score(y_train, y_hat_train),
                    r2_score(y_test, y_hat_test)))
```

```
R^2 train: 0.999, test: 0.676
```

isssues with overfitting

Model 3

Random Forest Regressor

In [97]:
```
rf_model = RandomForestRegressor()

# Train the model on training data
rf_model.fit(x_train, y_train)

# Predict on test data
predictions = rf_model.predict(x_test)
```

Random Forest model prediction

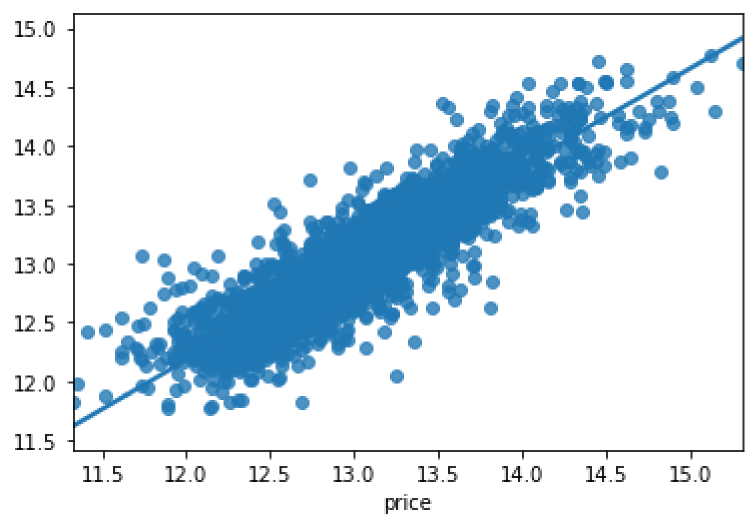
In [98]:
```
# Get how well it performed
mae_rf = mean_absolute_error(y_test, predictions)

print("Random Forest: {:,}".format(mae_rf))
```

```
Random Forest: 0.15621599370318473
```

In [99]:
```
sns.regplot(x=y_test, y=predictions)
```

Out[99]: <AxesSubplot:xlabel='price'>



In [100]:
```
y_hat_train = rf_model.predict(x_train)
y_hat_test = rf_model.predict(x_test)
```

In [101]:
```
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 0.006833147455952221
Test Mean Squarred Error: 0.047854429562705955
```

In [102]:
```
print('R^2 train: %.3f, test: %.3f' % (r2_score(y_train, y_hat_train),
                                      r2_score(y_test, y_hat_test)))
```

```
R^2 train: 0.974, test: 0.817
```

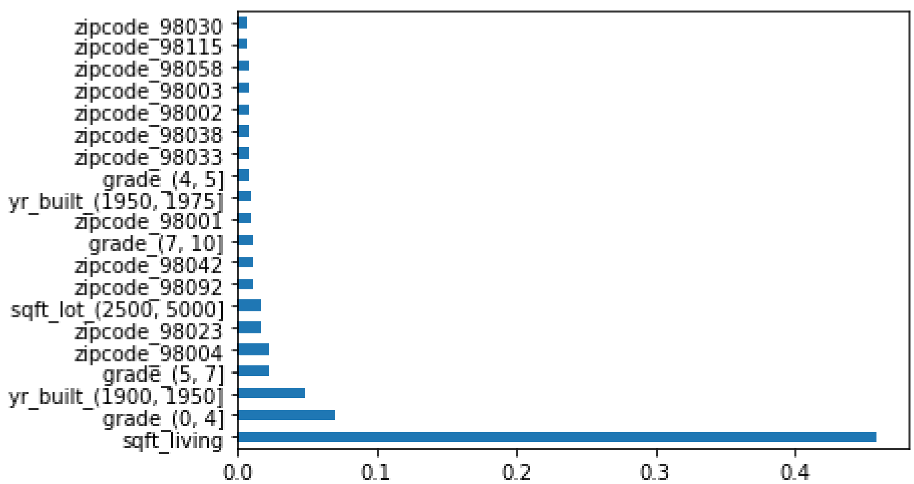
In [103]:
```
# R^2 train: 0.927, test: 0.578
```

In [104]:
```
# feats = {} # a dict to hold feature_name: feature_importance
# for feature, importance in zip(x_train.columns, model.feature_importances
#     feats[feature] = importance #add the name/value pair

# importances = pd.DataFrame.from_dict(feats, orient='index').rename(column
# importances.sort_values(by='Gini-importance').plot(kind='bar', rot=45)
```

In [105]:
```
feat_importances = pd.Series(rf_model.feature_importances_, index = rfm_col
feat_importances.nlargest(20).plot(kind='barh')
```

Out[105]: <AxesSubplot:>



In [106]:
```
feat_importances.nlargest(20)
```

Out[106]:
```
sqft_living               0.459631
grade_(0, 4]              0.070201
yr_built_(1900, 1950]     0.047959
grade_(5, 7]              0.022092
zipcode_98004             0.021954
zipcode_98023             0.016912
sqft_lot_(2500, 5000]     0.016746
zipcode_98092             0.010988
zipcode_98042             0.010501
grade_(7, 10]             0.010379
zipcode_98001             0.010089
yr_built_(1950, 1975]     0.008817
grade_(4, 5]              0.008665
zipcode_98033             0.008585
zipcode_98038             0.008201
zipcode_98002             0.008027
zipcode_98003             0.007730
zipcode_98058             0.007520
zipcode_98115             0.007229
zipcode_98030             0.006909
dtype: float64
```

Model 4

OLS

In [107]:
```
x_train1 = sm.add_constant(x_train)
```

In [108]:
```
x_test1 = sm.add_constant(x_test)
```

In [109]:
```
result = sm.OLS(y_train,x_train1).fit()
```

OLS model prediction

In [110]:
```
print(result.rsquared, result.rsquared_adj)
```

```
0.8563583837968929 0.8555466581578831
```

In [111]:
```
print(result.summary(xname= xname_columns))
```

```
-0.054      -0.048
zipcode_98003                 -0.0513      0.001    -34.376      0.000
-0.054      -0.048
zipcode_98004                  0.0779      0.002     51.703      0.000
0.075       0.081
zipcode_98005                  0.0213      0.002     14.105      0.000
0.018       0.024
zipcode_98006                  0.0295      0.002     19.478      0.000
0.027       0.032
zipcode_98007                  0.0152      0.001     10.176      0.000
0.012       0.018
zipcode_98008                  0.0242      0.002     16.056      0.000
0.021       0.027
zipcode_98010                 -0.0168      0.002    -11.176      0.000

-0.020      -0.014
zipcode_98011                 -0.0040      0.001     -2.703      0.007
-0.007      -0.001
zipcode_98014                 -0.0146      0.002     -9.699      0.000
-0.018      -0.012
```

Looking at our OLS summary all grade bins except grade_(0, 4] have high pvalue and high st err

OTHER HIGH PVALUES grade_(4, 5] grade_(5, 7] grade_(7, 10] floors_(2, 4] yr_built_(1900, 1950] yr_built_(2000, 2015] season_sold_Fall
season_sold_Summer sqft_lot_(5000, 7500]
zipcode_98024 zipcode_98070 zipcode_98072 zipcode_98077 zipcode_98133

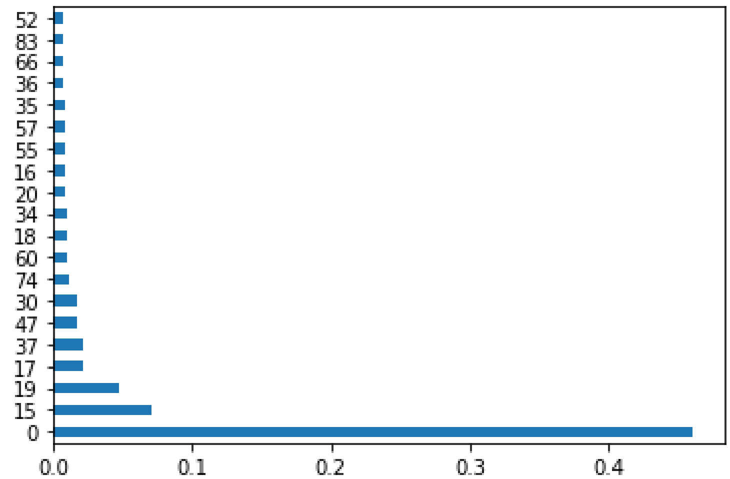
Model Reiteration and Feature Selection

Random Forest Regressor - 2

Using the Feature Importance graph let's take a look at random forest model with only the top 20 features

In [112]:
```
feat_importances = pd.Series(rf_model.feature_importances_)
feat_importances.nlargest(20).plot(kind='barh')
```

Out[112]: <AxesSubplot:>



In [113]:
```
#rf_model_2_index = [0,15,19,17,30,20,26,27,25,28,33,2,14,18,13,31,32,3,16,
```

In [114]:
```
rf2_model = RandomForestRegressor()

# Train the model on training data
rf2_model.fit(x_train_rf2, y_train)
```

Out[114]: RandomForestRegressor()

Random Forest model prediction -2

In [115]:
```
# Predict on test data
predictions = rf2_model.predict(x_test_rf2)
```

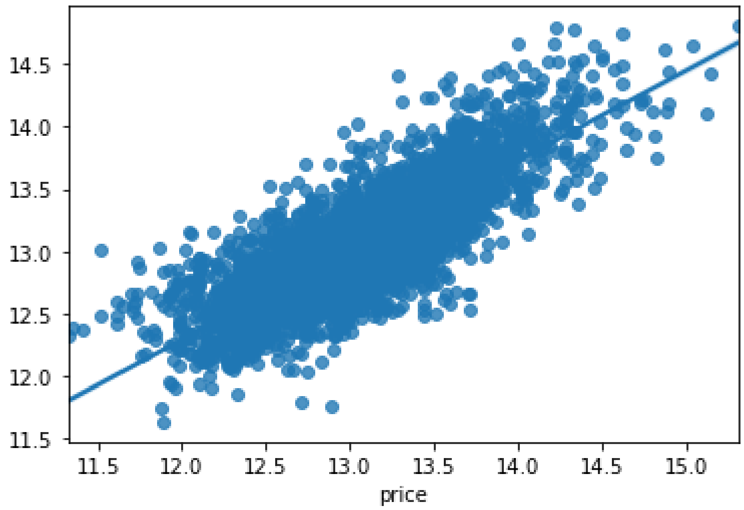
In [116]:
```
##### Get how well it performed
mae_rf = mean_absolute_error(y_test, predictions)

print("Random Forest: {:,}".format(mae_rf))
```

```
Random Forest: 0.22517524119864782
```

In [117]:
```
sns.regplot(x=y_test, y=predictions)
```

Out[117]:
```
<AxesSubplot:xlabel='price'>
```



In [118]:
```
y_hat_train = rf2_model.predict(x_train_rf2)
y_hat_test = rf2_model.predict(x_test_rf2)
```

In [119]:
```
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 0.050451008604447976
Test Mean Squarred Error: 0.08824661678762068
```

In [120]:
```
print('R^2 train: %.3f, test: %.3f' % (r2_score(y_train, y_hat_train),
                r2_score(y_test, y_hat_test)))
```

```
R^2 train: 0.810, test: 0.663
```

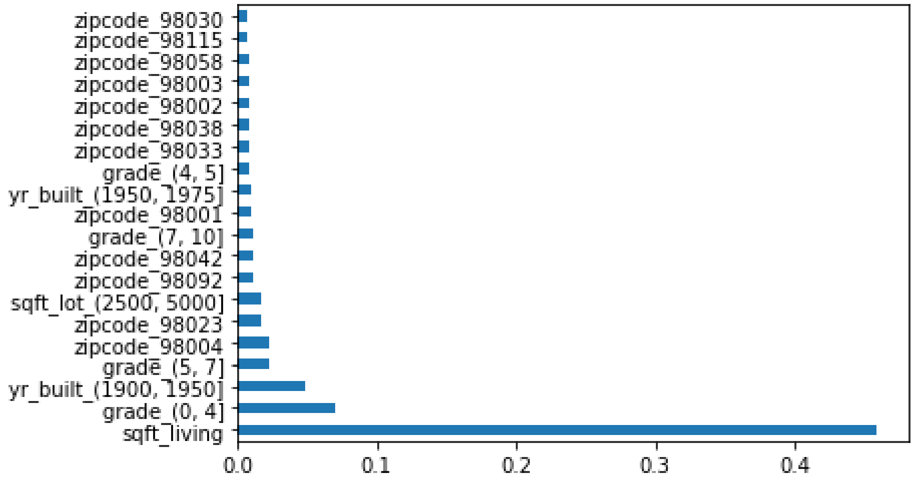
Selecting top 20 features did not improve our Random Forest Model performance

Our original Mean Squared Error: Train Mean Squarred Error: 0.006888463386013245 Test Mean Squarred Error: 0.04811152977830136

Our error has increased and our R^2 scores have gone down, this indicates that selecting features solely based on their importance is not a sound strategy.

In [121]:
```
feat_importances = pd.Series(rf_model.feature_importances_, index = rfm_col
feat_importances.nlargest(20).plot(kind='barh')
```

Out[121]: <AxesSubplot:>



OLS - 2

Using our OlS results summary let's take a look at our OLS model with features removed which have high std err and pvalues.

In [122]:
```
x_train1 = sm.add_constant(x_train_ols2)
```

In [123]:
```
x_test1 = sm.add_constant(x_test_ols2)
```

In [124]:
```
result = sm.OLS(y_train,x_train1).fit()
```

OLS model prediction - 2

In [125]:
```
print(result.rsquared, result.rsquared_adj)
```

```
0.8440335384876905 0.8432444388471411
```

In [126]:
```
print(result.summary(xname= xname2_columns))
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                  price   R-squared:
0.844
Model:                            OLS   Adj. R-squared:
0.843
Method:                 Least Squares   F-statistic:
1070.
Date:                Wed, 24 Mar 2021   Prob (F-statistic):
0.00
Time:                        13:39:20   Log-Likelihood:                    2
959.5
No. Observations:               17085   AIC:                               -
5745.
Df Residuals:                   16998   BIC:                               -
5071.
Df Model:                          86
Covariance Type:            nonrobust
==============================================================================
=====================
                           coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
---------------------
const                   13.0371      0.002   8353.077      0.000
13.034      13.040
sqft_living              0.2515      0.003     80.163      0.000
0.245       0.258
bedrooms_(0, 2]          0.0208      0.002     12.446      0.000
0.018       0.024
bedrooms_(2, 3]          0.0050      0.001      4.790      0.000
0.003       0.007
bedrooms_(3, 4]         -0.0088      0.001     -7.119      0.000
-0.011      -0.006
bedrooms_(4, 33]        -0.0201      0.002    -13.069      0.000
-0.023      -0.017
bathrooms_(0, 1]        -0.0087      0.002     -4.719      0.000
-0.012      -0.005
bathrooms_(1, 2]        -0.0124      0.001    -10.215      0.000
-0.015      -0.010
bathrooms_(2, 3]     -2.411e-05      0.001     -0.018      0.986
-0.003       0.003
bathrooms_(3, 8]         0.0338      0.002     19.217      0.000
0.030       0.037
floors_(0, 1]           -0.0137      0.005     -2.525      0.012
-0.024      -0.003
floors_(1, 2]           -0.0018      0.005     -0.367      0.714
-0.012       0.008
condition_(0, 2]        -0.0175      0.002    -11.318      0.000
-0.021      -0.015
condition_(2, 3]        -0.0090      0.001    -10.315      0.000
-0.011      -0.007
condition_(3, 5]         0.0126      0.001     14.311      0.000
0.011       0.014
```

```
grade_(0, 4]                   -0.0649      0.002    -29.535      0.000
-0.069      -0.061
yr_built_(1950, 1975]          -0.0238      0.002    -10.794      0.000
-0.028      -0.019
yr_built_(1975, 2000]          -0.0205      0.002     -9.839      0.000
-0.025      -0.016
year_sold_2014                 -0.0111      0.001     -9.575      0.000
-0.013      -0.009
year_sold_2015                  0.0111      0.001      9.575      0.000
0.009       0.013
season_sold_Spring              0.0076      0.002      3.255      0.001
0.003       0.012
season_sold_Winter             -0.0060      0.002     -3.026      0.002
-0.010      -0.002
sqft_lot_(0, 2500]             -0.0535      0.002    -23.138      0.000
-0.058      -0.049
sqft_lot_(2500, 5000]          -0.0133      0.002     -6.541      0.000
-0.017      -0.009
sqft_lot_(7500, 10000]          0.0048      0.002      2.340      0.019
0.001       0.009
sqft_lot_(10000, 10000000]      0.0355      0.002     15.718      0.000
0.031       0.040
zipcode_98001                  -0.0617      0.002    -34.866      0.000
-0.065      -0.058
zipcode_98002                  -0.0504      0.002    -29.786      0.000
-0.054      -0.047
zipcode_98003                  -0.0516      0.002    -29.738      0.000
-0.055      -0.048
zipcode_98004                   0.0834      0.002     47.359      0.000
0.080       0.087
zipcode_98005                   0.0239      0.002     14.318      0.000
0.021       0.027
zipcode_98006                   0.0361      0.002     19.398      0.000
0.032       0.040
zipcode_98007                   0.0167      0.002     10.079      0.000
0.013       0.020
zipcode_98008                   0.0261      0.002     14.933      0.000
0.023       0.029
zipcode_98010                  -0.0167      0.002    -10.266      0.000
-0.020      -0.013
zipcode_98011                  -0.0036      0.002     -2.160      0.031
-0.007      -0.000
zipcode_98014                  -0.0150      0.002     -9.092      0.000
-0.018      -0.012
zipcode_98019                  -0.0168      0.002    -10.004      0.000
-0.020      -0.014
zipcode_98022                  -0.0393      0.002    -23.043      0.000
-0.043      -0.036
zipcode_98023                  -0.0751      0.002    -40.361      0.000
-0.079      -0.071
zipcode_98027                   0.0057      0.002      3.189      0.001
0.002       0.009
zipcode_98028                  -0.0093      0.002     -5.380      0.000
-0.013      -0.006
zipcode_98029                   0.0155      0.002      8.700      0.000
0.012       0.019
zipcode_98030                  -0.0482      0.002    -28.107      0.000
```

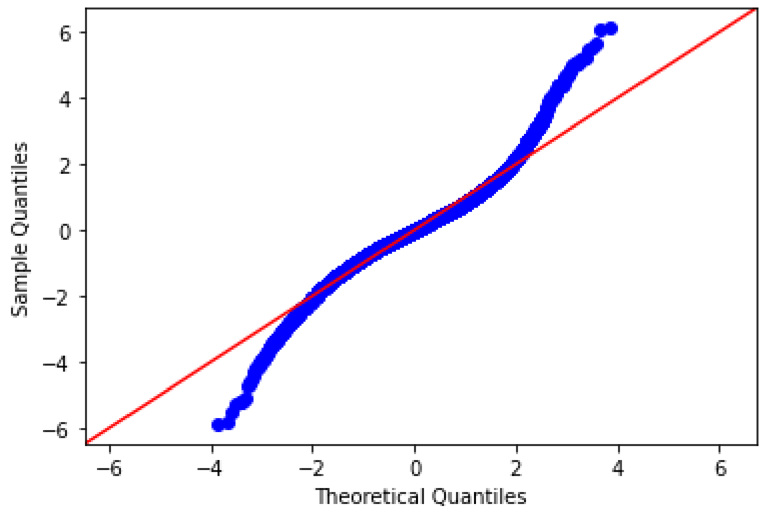
```
-0.052      -0.045
zipcode_98031                  -0.0460      0.002    -26.459      0.000
-0.049      -0.043
zipcode_98032                  -0.0390      0.002    -23.717      0.000
-0.042      -0.036
zipcode_98033                   0.0447      0.002     24.739      0.000
0.041       0.048
zipcode_98034                   0.0154      0.002      8.185      0.000
0.012       0.019
zipcode_98038                  -0.0540      0.002    -27.983      0.000
-0.058      -0.050
zipcode_98039                   0.0379      0.002     23.844      0.000
0.035       0.041
zipcode_98040                   0.0555      0.002     31.838      0.000
0.052       0.059
zipcode_98042                  -0.0653      0.002    -34.819      0.000
-0.069      -0.062
zipcode_98045                  -0.0132      0.002     -7.787      0.000
-0.016      -0.010
zipcode_98052                   0.0275      0.002     14.455      0.000
0.024       0.031
zipcode_98053                   0.0146      0.002      7.981      0.000
0.011       0.018
zipcode_98055                  -0.0382      0.002    -22.084      0.000
-0.042      -0.035
zipcode_98056                  -0.0219      0.002    -12.147      0.000
-0.025      -0.018
zipcode_98058                  -0.0470      0.002    -25.657      0.000
-0.051      -0.043
zipcode_98059                  -0.0212      0.002    -11.582      0.000
-0.025      -0.018
zipcode_98065                  -0.0113      0.002     -6.404      0.000
-0.015      -0.008
zipcode_98074                   0.0158      0.002      8.631      0.000
0.012       0.019
zipcode_98075                   0.0193      0.002     10.875      0.000
0.016       0.023
zipcode_98092                  -0.0576      0.002    -32.463      0.000
-0.061      -0.054
zipcode_98102                   0.0383      0.002     23.221      0.000
0.035       0.042
zipcode_98103                   0.0588      0.002     29.289      0.000
0.055       0.063
zipcode_98105                   0.0524      0.002     30.246      0.000
0.049       0.056
zipcode_98106                  -0.0171      0.002     -9.536      0.000
-0.021      -0.014
zipcode_98107                   0.0450      0.002     25.271      0.000
0.042       0.049
zipcode_98108                  -0.0107      0.002     -6.296      0.000
-0.014      -0.007
zipcode_98109                   0.0411      0.002     24.798      0.000
0.038       0.044
zipcode_98112                   0.0693      0.002     39.100      0.000
0.066       0.073
zipcode_98115                   0.0563      0.002     28.822      0.000
0.052       0.060
```

```
zipcode_98116                 0.0393      0.002     21.833      0.000
0.036       0.043
zipcode_98117                 0.0535      0.002     27.604      0.000
0.050       0.057
zipcode_98118                -0.0022      0.002     -1.173      0.241
-0.006       0.001
zipcode_98119                 0.0506      0.002     29.615      0.000
0.047       0.054
zipcode_98122                 0.0450      0.002     25.074      0.000
0.042       0.049
zipcode_98125                 0.0145      0.002      7.999      0.000
0.011       0.018
zipcode_98126                 0.0113      0.002      6.256      0.000
0.008       0.015
zipcode_98136                 0.0295      0.002     16.852      0.000
0.026       0.033
zipcode_98144                 0.0315      0.002     17.176      0.000
0.028       0.035
zipcode_98146                -0.0231      0.002    -13.239      0.000
-0.027      -0.020
zipcode_98148                -0.0157      0.002     -9.789      0.000
-0.019      -0.013
zipcode_98155                -0.0061      0.002     -3.335      0.001
-0.010      -0.003
zipcode_98166                -0.0125      0.002     -7.243      0.000
-0.016      -0.009
zipcode_98168                -0.0489      0.002    -28.054      0.000
-0.052      -0.045
zipcode_98177                 0.0157      0.002      9.126      0.000
0.012       0.019
zipcode_98178                -0.0335      0.002    -19.245      0.000
-0.037      -0.030
zipcode_98188                -0.0292      0.002    -17.723      0.000
-0.032      -0.026
zipcode_98198                -0.0432      0.002    -24.798      0.000
-0.047      -0.040
zipcode_98199                 0.0517      0.002     29.011      0.000
0.048       0.055
==========================================================================
=====
Omnibus:                     1322.479   Durbin-Watson:
1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              636
6.351
Skew:                           0.217   Prob(JB):
0.00
Kurtosis:                       5.959   Cond. No.                      2.1
4e+16
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is 1.69e-28. This might indicate that there a
re
strong multicollinearity problems or that the design matrix is singular.
```

Normality

In [128]:
```
fig = sm.graphics.qqplot(result.resid, dist=stats.norm, line='45', fit=True
```

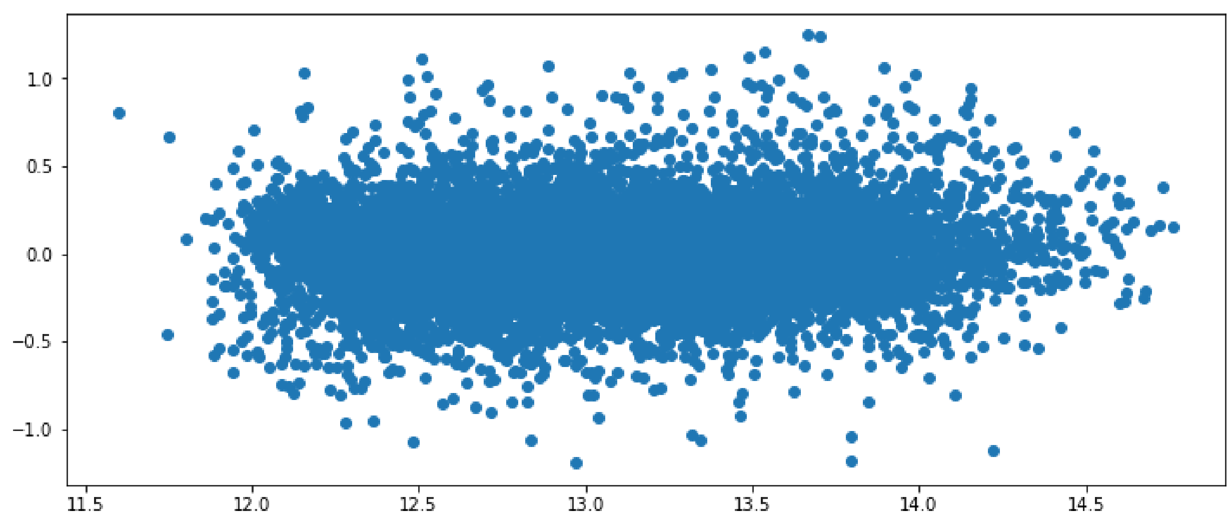


Looks like the model does best at predicting mid priced houses and doesn't perform as well on low priced or high priced houses.

Homoskedasticity

In [129]:
```
plt.figure(figsize=(12,5))
plt.scatter(result.predict(x_train1), result.resid)
```

Out[129]:
```
<matplotlib.collections.PathCollection at 0x7f8c7c6279a0>
```



Homoskedasticity looks ok

k-fold cross validation

In [130]:
```
cross_validation_5 = KFold(5, shuffle=True)
cross_validation_10 = KFold(5, shuffle=True)
cross_validation_20 = KFold(5, shuffle=True)
```

In [131]:
```
# set scoring argument to 'neg_mean_squared_error'. This negates mean squar
cv_5_results  = np.mean(cross_val_score(linear_model, x, y, cv=cross_valida
cv_10_results = np.mean(cross_val_score(linear_model, x, y, cv=cross_valida
cv_20_results = np.mean(cross_val_score(linear_model, x, y, cv=cross_valida
```

In [132]:
```
cv_5_results
```

Out[132]:
```
nan
```

In [133]:
```
cv_10_results
```

Out[133]:
```
nan
```

In [134]:
```
cv_20_results
```

Out[134]:
```
nan
```

Summary

```
Our final model has an r-squared-adjusted value of 0.84.
Our model performs about the same with the train and test data so we don't
seem to have an overfitting issue.
From the residuals in our qq plot looks like the model does best at
predicting mid priced houses and doesn't perform as well on low priced or
high priced houses.
Homoskedasticity looked good in our visualization.

Our pricing model is most effective at estimating mid range  housing
prices. It is not advisable to use this model for the low end  or high end
of the King County housing market.


Expansion of data and more develpment could better improve the model to
estimate low and high priced houses.
```

Recommendations

- Sell houses in the Spring and Summer when the market is strong. More houses are purchased during this time and they tend to sell for higher prices.
- Our most important and reliable factor for predicting a home price is sqft_living. When pricing a house the overall square footage should be the most important factor.
- Waterfront properties will have a higher price to comparable properties.
- Properties with a grade of 4 and under will have a lower price to comparable properties.
- Expand data set beyond houses sold in 2014 and 2015. Having more current data would make the model more accurate.

Further Analysis

- Expand data set beyond houses sold in 2014 and 2015. Having more current data would make the model more accurate.
- Adjust modeling to better estimate low and high priced houses.
    - Investigate what features would help us to estimate extremes