King County Housing Analysis

Motivation

Property adviser in a real estate company

Objectives

What are the important key factors to predict the price of the property? What are important features to increase the value of the property? How geographic features affect property values? How home interior can affect the house value?

Data loading

In [2]:

```
#Import Libaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats
import scipy.stats as stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_fac

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score


```

In [3]:

```
#Load data and check correct format
### Convert ? to  NaN
df= pd.read_csv('kc_house_data.csv', na_values='?')
```

#Data abbreviations description

Column names

id - unique identified for a house date - Date house was sold price - Price is prediction target bedrooms - Number of Bedrooms/House bathrooms - Number of bathrooms/bedrooms sqft_living - Square footage of the home sqft_lot - Square footage of the lot floors - Total floors (levels) in house waterfront - House which has a view to a waterfront view - If the house has a view or not 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 - zipcode of the house 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

In [1]:

```
#features = df.iloc[:,:-1]
#target=df['price']
#features.head(10)
```

In [8]:
```
df.head(20)
```

Out[8]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
5	7237550310	5/12/2014	1230000.0	4	4.50	5420	101930	1.0	
6	1321400060	6/27/2014	257500.0	3	2.25	1715	6819	2.0	
7	2008000270	1/15/2015	291850.0	3	1.50	1060	9711	1.0	
8	2414600126	4/15/2015	229500.0	3	1.00	1780	7470	1.0	
9	3793500160	3/12/2015	323000.0	3	2.50	1890	6560	2.0	
10	1736800520	4/3/2015	662500.0	3	2.50	3560	9796	1.0	
11	9212900260	5/27/2014	468000.0	2	1.00	1160	6000	1.0	
12	114101516	5/28/2014	310000.0	3	1.00	1430	19901	1.5	
13	6054650070	10/7/2014	400000.0	3	1.75	1370	9680	1.0	
14	1175000570	3/12/2015	530000.0	5	2.00	1810	4850	1.5	
15	9297300055	1/24/2015	650000.0	4	3.00	2950	5000	2.0	
16	1875500060	7/31/2014	395000.0	3	2.00	1890	14040	2.0	
17	6865200140	5/29/2014	485000.0	4	1.00	1600	4300	1.5	
18	16000397	12/5/2014	189000.0	2	1.00	1200	9850	1.0	
19	7983200060	4/24/2015	230000.0	3	1.00	1250	9774	1.0	

20 rows × 21 columns

#Information about Grade column Represents the construction quality of improvements. Grades run from grade 1 to 13. Generally defined as:

1-3 = Falls short of minimum building standards. Normally cabin or inferior structure

4 = Generally older, low quality construction. Does not meet code.

5 = Low construction costs and workmanship. Small, simple design.

6 = Lowest grade currently meeting building code. Low quality materials and simple designs.

7 = Average grade of construction and design. Commonly seen in plats and older sub-divisions.

8 = Just above average in construction and design. Usually better materials in both the exterior and interior finish work.

9 = Better architectural design with extra interior and exterior design and quality.

10 = Homes of this quality generally have high quality features. Finish work is better and more design quality is seen in the floor plans. Generally have a larger square footage.

11 = Custom design and higher quality finish work with added amenities of solid woods, bathroom fixtures and more luxurious options.

12 = Custom design and excellent builders. All materials are of the highest quality and all conveniences are present.

13 = Generally custom designed and built. Mansion level. Large amount of highest quality cabinet work, wood trim, marble, entry ways etc.

Information derived from:
https://blue.kingcounty.com/Assessor/eRealProperty/ResidentialGlossary.aspx?idx=viewall&Parcel=7960900070&AreaReport=http://www.KingCounty.gov/depts/Assessor/Reports/areports/2019/residential-northeast/033.aspx#BuildingGrade (https://blue.kingcounty.com/Assessor/eRealProperty/ResidentialGlossary.aspx?idx=viewall&Parcel=7960900070&AreaReport=http://www.KingCounty.gov/depts/Assessor/Reports/areports/2019/residential-northeast/033.aspx#BuildingGrade)

In [353]:
```
df.shape
```

Out[353]: (21597, 21)

Data cleaning

In [272]:

```
df.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  21143 non-null  float64
 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(9), int64(11), object(1)
memory usage: 3.5+ MB
```

Columns with missing values: waterfront, view, sqft_basement, yr_renovated

In [9]:
```
#Checking for missing values
df.isna().any()
```

Out[9]:
```
id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront        True
view              True
condition        False
grade            False
sqft_above       False
sqft_basement     True
yr_built         False
yr_renovated      True
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

In [9]:
```
df.isna().sum()
```

Out[9]:
```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2376
view              63
condition          0
grade              0
sqft_above         0
sqft_basement    454
yr_built           0
yr_renovated    3842
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

In [11]:
```
df.sqft_basement.value_counts()
```

Out[11]:
```
0.0       12826
600.0       217
500.0       209
700.0       208
800.0       201
          ...
915.0         1
295.0         1
1281.0        1
2130.0        1
906.0         1
Name: sqft_basement, Length: 303, dtype: int64
```

In [10]:
```
df.sqft_basement.unique()
```

Out[10]:
```
array([   0.,  400.,  910., 1530.,   nan,  730., 1700.,  300.,  970.,
        760.,  720.,  700.,  820.,  780.,  790.,  330., 1620.,  360.,
        588., 1510.,  410.,  990.,  600.,  560.,  550., 1000., 1600.,
        500., 1040.,  880., 1010.,  240.,  265.,  290.,  800.,  540.,
        710.,  840.,  380.,  770.,  480.,  570., 1490.,  620., 1250.,
       1270.,  120.,  650.,  180., 1130.,  450., 1640., 1460., 1020.,
       1030.,  750.,  640., 1070.,  490., 1310.,  630., 2000.,  390.,
        430.,  850.,  210., 1430., 1950.,  440.,  220., 1160.,  860.,
        580., 2060., 1820., 1180.,  200., 1150., 1200.,  680.,  530.,
       1450., 1170., 1080.,  960.,  280.,  870., 1100.,  460., 1400.,
        660., 1220.,  900.,  420., 1580., 1380.,  475.,  690.,  270.,
        350.,  935., 1370.,  980., 1470.,  160.,  950.,   50.,  740.,
       1780., 1900.,  340.,  470.,  370.,  140., 1760.,  130.,  520.,
        890., 1110.,  150., 1720.,  810.,  190., 1290.,  670., 1800.,
       1120., 1810.,   60., 1050.,  940.,  310.,  930., 1390.,  610.,
       1830., 1300.,  510., 1330., 1590.,  920., 1320., 1420., 1240.,
       1960., 1560., 2020., 1190., 2110., 1280.,  250., 2390., 1230.,
        170.,  830., 1260., 1410., 1340.,  590., 1500., 1140.,  260.,
        100.,  320., 1480., 1060., 1284., 1670., 1350., 2570., 1090.,
        110., 2500.,   90., 1940., 1550., 2350., 2490., 1481., 1360.,
       1135., 1520., 1850., 1660., 2130., 2600., 1690.,  243., 1210.,
       1024., 1798., 1610., 1440., 1570., 1650.,  704., 1910., 1630.,
       2360., 1852., 2090., 2400., 1790., 2150.,  230.,   70., 1680.,
       2100., 3000., 1870., 1710., 2030.,  875., 1540., 2850., 2170.,
        506.,  906.,  145., 2040.,  784., 1750.,  374.,  518., 2720.,
       2730., 1840., 3480., 2160., 1920., 2330., 1860., 2050., 4820.,
       1913.,   80., 2010., 3260., 2200.,  415., 1730.,  652., 2196.,
       1930.,  515.,   40., 2080., 2580., 1548., 1740.,  235.,  861.,
       1890., 2220.,  792., 2070., 4130., 2250., 2240., 1990.,  768.,
       2550.,  435., 1008., 2300., 2610.,  666., 3500.,  172., 1816.,
       2190., 1245., 1525., 1880.,  862.,  946., 1281.,  414., 2180.,
        276., 1248.,  602.,  516.,  176.,  225., 1275.,  266.,  283.,
         65., 2310.,   10., 1770., 2120.,  295.,  207.,  915.,  556.,
        417.,  143.,  508., 2810.,   20.,  274.,  248.])
```

In [3]:
```
df[df.bedrooms ==33]
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	w
15856	2402100895	6/25/2014	640000.0	33	1.75	1620	6000	1.0	

1 rows × 21 columns

In [4]:
```
#Change 33 bedrooms house to 3 bedrooms house
df['bedrooms'].replace(to_replace=33, value=3, inplace= True)
```

In [4]:
```
##Replace 0 with difference between sqft_above and sqft_living if any?
#df['sqft_basement'].replace(to_replace='0', value= abs(df['sqft_living
```

In [355]:
```
df['sqft_basement'].nunique()
```

Out[355]: 303

Missing values

In [5]:
```
##Replace missing values
features1=['waterfront', 'view']
features2=['yr_renovated', 'sqft_basement']

for value in features1:
    mode=df[value].mode()[0]
    df[value].fillna(mode, inplace=True)

for value in features2:
    median=df[value].median()
    df[value].fillna(median, inplace=True)
```

In [5]:
```
df.isna().any()
```

Out[5]:
```
id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

In [20]:
```
df.isna().sum()
```

Out[20]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [9]:
```
df['view'].isnull()
```

Out[9]:
```
0        False
1        False
2        False
3        False
4        False
         ...
21592    False
21593    False
21594    False
21595    False
21596    False
Name: view, Length: 21597, dtype: bool
```

In [22]:
```
df['view'].value_counts()
```

Out[22]:
```
0.0    19485
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

In [32]:
```
df['view'].nunique()
```

Out[32]:
```
5
```

In [30]:
```
df['waterfront'].value_counts()
```

Out[30]:
```
0.0    21451
1.0      146
Name: waterfront, dtype: int64
```

In [31]:
```
df['waterfront'].nunique()
```

Out[31]:
```
2
```

In [6]:
```
#transform date dtype 'object' to date fromat
df['date']=pd.to_datetime(df['date'])
```

In [11]:

```
df.info()
```

```
 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     21597 non-null  float64
 9   view           21597 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  float64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 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: datetime64[ns](1), float64(9), int64(11)
memory usage: 3.5 MB
```

In [7]:

```
#Drop id column
df=df.drop(['id'], axis=1)
```

In [27]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           21597 non-null  datetime64[ns]
 1   price          21597 non-null  float64
 2   bedrooms       21597 non-null  int64
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64
 5   sqft_lot       21597 non-null  int64
 6   floors         21597 non-null  float64
 7   waterfront     21597 non-null  float64
 8   view           21597 non-null  float64
 9   condition      21597 non-null  int64
 10  grade          21597 non-null  int64
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21597 non-null  float64
 13  yr_built       21597 non-null  int64
 14  yr_renovated   21597 non-null  float64
 15  zipcode        21597 non-null  int64
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64
 19  sqft_lot15     21597 non-null  int64
dtypes: datetime64[ns](1), float64(9), int64(10)
memory usage: 3.3 MB
```

In [28]:
```
df.head(10)
```

Out[28]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
0	2014-10-13	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	
1	2014-12-09	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	
2	2015-02-25	180000.0	2	1.00	770	10000	1.0	0.0	0.0	
3	2014-12-09	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	
4	2015-02-18	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	
5	2014-05-12	1230000.0	4	4.50	5420	101930	1.0	0.0	0.0	
6	2014-06-27	257500.0	3	2.25	1715	6819	2.0	0.0	0.0	
7	2015-01-15	291850.0	3	1.50	1060	9711	1.0	0.0	0.0	
8	2015-04-15	229500.0	3	1.00	1780	7470	1.0	0.0	0.0	
9	2015-03-12	323000.0	3	2.50	1890	6560	2.0	0.0	0.0	

In [33]:
```
#Double check if anything is missing
df.isna().sum().any().any()
```

Out[33]: False

In [11]:
```
df.describe().round(2)
```

Out[11]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vi
count	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597.00	21597
mean	540296.57	3.37	2.12	2080.32	15099.41	1.49	0.01	0
std	367368.14	0.90	0.77	918.11	41412.64	0.54	0.08	0
min	78000.00	1.00	0.50	370.00	520.00	1.00	0.00	0
25%	322000.00	3.00	1.75	1430.00	5040.00	1.00	0.00	0
50%	450000.00	3.00	2.25	1910.00	7618.00	1.50	0.00	0
75%	645000.00	4.00	2.50	2550.00	10685.00	2.00	0.00	0
max	7700000.00	11.00	8.00	13540.00	1651359.00	3.50	1.00	4

Data exploration

#check if the distributions of continuous variables are normal. We can check if there is a high skewness or/and kurtosis.

In [348]:

```
#Quick check for the distributions if it is normal or not
df.hist(figsize=(20, 15));
```

Most of the distributions are skewed and do not obey normal distribution.

Outliers

There are a few of vey expensive homes as outliers. I decided to get rid of 0.1 % of the data

In [13]:

```
#Check for outliers
fig, ax=plt.subplots(figsize=(15,5))
sns.boxplot(x='price', data=df, orient='h', width=0.1, showmeans=True,

plt.show()
```

It looks like outliers represent homes that are more than 6,000000$

In [7]:

```
#Get rid of outliers
outliers=df[(df['price'])>=7500000].index
df.drop(outliers, inplace=True)
```

In [8]:

```
for i in range(80,100):
    q = i/100
    print("{} percentile: {}".format(q, df.price.quantile(q=q)))
```

```
0.8 percentile: 700435.9999999998
0.81 percentile: 718000.0
0.82 percentile: 730000.72
0.83 percentile: 749950.0
0.84 percentile: 760003.2
0.85 percentile: 779721.9999999991
0.86 percentile: 799000.0
0.87 percentile: 815000.0
0.88 percentile: 836739.9999999998
0.89 percentile: 859967.6
0.9 percentile: 887000.0
0.91 percentile: 919993.6
0.92 percentile: 950000.0
0.93 percentile: 997964.0000000002
0.94 percentile: 1060000.0
0.95 percentile: 1160000.0
0.96 percentile: 1260000.0
0.97 percentile: 1390000.0
0.98 percentile: 1600000.0
0.99 percentile: 1970000.0
```

In [9]:

```
orig_tot = len(df)
df = df[df.price <1970000.0] # Subsetting to remove extreme outliers
print('Percent removed:', (orig_tot -len(df))/orig_tot)
```

```
Percent removed: 0.010047691809047552
```

In [19]:

```
#Plot again histograms for all variables in the dataset
df.hist(figsize=(20, 18), bins='auto')
```

Out[19]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000219986FF688
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002199873C508
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998AA7CC8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998AE0708
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021998B17F88
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998B51908
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998B8A2C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998BC0CC8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021998BCA8C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998C02A88
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021998C6F048
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999C780C8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021999CB01C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999CE8308
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999D21408
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999D5B508
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000021999D935C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999DCC6C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999E047C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000021999E3C908
>]],
      dtype=object)
```

Most of these distributions are skewed, one way to ameloirate for skewed distribution is log transformations.

We can also distiguish continuous variables for categorials.

In [11]:

```
#check for linearity and plot some single variable regression plots of
sns.jointplot('price','sqft_living', data=df, kind='reg');
sns.jointplot('price','sqft_above', data=df, kind='reg');
sns.jointplot('price','sqft_lot', data=df, kind='reg');
sns.jointplot('price','sqft_basement', data=df, kind='reg');
sns.jointplot('price','lat', data=df, kind='reg');
sns.jointplot('price','long', data=df, kind='reg');
sns.jointplot('price','yr_built', data=df, kind='reg');
sns.jointplot('price','yr_renovated', data=df, kind='reg');
sns.jointplot('price','zipcode', data=df, kind='reg');
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py in kdensityfft(X, kernel, bw, weights, gridsize, adjust, clip, cut, retgrid)
    450     try:
--> 451         bw = float(bw)
    452     except:

ValueError: could not convert string to float: 'scott'

During handling of the above exception, another exception occurred:

RuntimeError                              Traceback (most recent call last)
<ipython-input-11-16dc0eaab04a> in <module>
      8 sns.jointplot('price','long', data=df, kind='reg');
      9 sns.jointplot('price','yr_built', data=df, kind='reg');
```

It looks like from initial investigations, sqft_living, sqft_above are linear.

Identifying linearity

The linearity assumptions requires that there is a linear relationship between the response variable (Y) and predictor (X). Linear means that the change in Y by 1-unit change in X, is constant.

In [362]:

```
df_plot = df[['price', 'bathrooms','bedrooms' ,'sqft_living', 'sqft_lot
              'sqft_basement', 'yr_built','waterfront', "yr_renovated",
sns.pairplot(df_plot)
plt.show()
```

In [364]:

```
#Remove categorial variables for better visualization and linearity
df_plot = df[['price', 'bathrooms', 'sqft_living', 'sqft_lot', 'sqft_ab
              'sqft_basement', 'yr_built', 'sqft_living15', 'sqft_lot15
sns.pairplot(df_plot)
plt.show()
```

Linear relationship

When we check the linearity of parameters we see that 'price' has a linear relationship with other columns except 'yr_built'. There is a strong linear relationship between the price and sqft_living, sqft_above, sqft_basement, bathrooms, sqft_living15. We need to check the distributions one by one to avoid multicollinearity. We can eliminate these columns by checking their p values.

Identifying multicollinearity

Multicollinearity is the occurrence of high intercorrelations among two or more independent variables in a multiple regression model. Multicollinearity is a problem because it undermines the statistical significance of an independent variable. Other things being equal, the larger the standard error of a regression coefficient, the less likely it is that this coefficient will be statistically significant.

In [367]:

```
#Identifying multicollinearity
features = df.drop(['price'], axis=1)
target=df['price']
features.head(10)

```

Out[367]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grad
0	2014-10-13	3	1.00	1180	5650	1.0	0.0	0.0	3	
1	2014-12-09	3	2.25	2570	7242	2.0	0.0	0.0	3	
2	2015-02-25	2	1.00	770	10000	1.0	0.0	0.0	3	
3	2014-12-09	4	3.00	1960	5000	1.0	0.0	0.0	5	
4	2015-02-18	3	2.00	1680	8080	1.0	0.0	0.0	3	
5	2014-05-12	4	4.50	5420	101930	1.0	0.0	0.0	3	1
6	2014-06-27	3	2.25	1715	6819	2.0	0.0	0.0	3	
7	2015-01-15	3	1.50	1060	9711	1.0	0.0	0.0	3	
8	2015-04-15	3	1.00	1780	7470	1.0	0.0	0.0	3	
9	2015-03-12	3	2.50	1890	6560	2.0	0.0	0.0	3	

Detection of correlated parameters

In [368]:

```
#Select only significant correlations greater than 0.75
abs(features.corr()) > 0.75
```

Out[368]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
bedrooms	True	False	False	False	False	False	False	False
bathrooms	False	True	False	False	False	False	False	False
sqft_living	False	False	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
view	False	False	False	False	False	False	True	False
condition	False	False	False	False	False	False	False	True
grade	False	False	False	False	False	False	False	False
sqft_above	False	False	True	False	False	False	False	False
sqft_basement	False	False	False	False	False	False	False	False
yr_built	False	False	False	False	False	False	False	False
yr_renovated	False	False	False	False	False	False	False	False
zipcode	False	False	False	False	False	False	False	False
lat	False	False	False	False	False	False	False	False
long	False	False	False	False	False	False	False	False
sqft_living15	False	False	True	False	False	False	False	False
sqft_lot15	False	False	False	False	False	False	False	False

In [369]:

```
pd.plotting.scatter_matrix(features,figsize  = [9, 9]);
plt.show()
```

In [16]:

```
features.corr()
```

Out[16]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
bedrooms	1.000000	0.520266	0.596904	0.033559	0.176499	-0.023221	0.063171
bathrooms	0.520266	1.000000	0.742046	0.085805	0.505652	0.016053	0.148946
sqft_living	0.596904	0.742046	1.000000	0.175440	0.353499	0.032656	0.236321
sqft_lot	0.033559	0.085805	0.175440	1.000000	-0.007385	0.020010	0.074192
floors	0.176499	0.505652	0.353499	-0.007385	1.000000	0.006936	0.014150
waterfront	-0.023221	0.016053	0.032656	0.020010	0.006936	1.000000	0.334788
view	0.063171	0.148946	0.236321	0.074192	0.014150	0.334788	1.000000
condition	0.020488	-0.133265	-0.067549	-0.007640	-0.268944	0.015470	0.040966
grade	0.354474	0.649476	0.748010	0.111272	0.459541	0.029890	0.209485
sqft_above	0.485556	0.668753	0.867559	0.184498	0.528760	0.014983	0.122372
sqft_basement	0.295411	0.250454	0.399329	0.011542	-0.261020	0.037216	0.243743
yr_built	0.164972	0.523841	0.338997	0.051817	0.496309	-0.036316	-0.059422
yr_renovated	0.015194	0.040291	0.038531	0.004892	0.001986	0.073972	0.085053
zipcode	-0.158676	-0.205942	-0.204762	-0.131171	-0.060486	0.043883	0.093153
lat	-0.016904	0.015291	0.041677	-0.087774	0.046495	-0.024307	-0.004155
long	0.141103	0.236582	0.263107	0.229421	0.129084	-0.049498	-0.077737
sqft_living15	0.396702	0.555616	0.754853	0.146153	0.276571	0.041711	0.247899
sqft_lot15	0.031123	0.085040	0.185815	0.713360	-0.012970	0.028138	0.070479

In [370]:

```
#include stack and zip to create a more robust solution that will retur
#the variable pairs from the correlation matrix that have correlations
df=features.corr().abs().stack().reset_index().sort_values(0, ascending

# zip the variable name columns (Which were only named level_0 and leve
df['pairs'] = list(zip(df.level_0, df.level_1))

# set index to pairs
df.set_index(['pairs'], inplace = True)

#d rop level columns
df.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df.columns = ['cc']

# drop duplicates. This could be dangerous if you have variables perfec
# for the sake of exercise, kept it in.
df.drop_duplicates(inplace=True)
```

In [371]:

```
df[(df.cc>.75) & (df.cc <1)]
```

Out[371]:

pairs	cc
(sqft_living, sqft_above)	0.867559
(sqft_living15, sqft_living)	0.754853

Data decision to address colinearity

There are three sets of variales that are highly correlated. we can remove sqft_living15 and sqft_above from data set df.drop(columns=['sqft_living15',"sqft_above"], inplace=True). For now I will keep all the features and test the performance of the model

In [374]:

```
#Heat map
plt.figure(figsize=(20, 20))
ax = sns.heatmap(features.corr(), center=0, linewidths=.5, annot=True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```

Identify categorial and continuous variables

In [384]:

```
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(16,15), sharey=True

for ax, column in zip(axes.flatten(), df.columns):
    ax.scatter(df[column], df['price'] , label=column, alpha=.1)
    ax.set_title(f'Sale Price vs {column}')
    ax.set_xlabel(column)
    ax.set_ylabel('Sale Price in $')

fig.tight_layout()
```

Log transformation and data normalization

Log transformation is an effective technique we can use to improve the performance of linear regression models. We make the distributions look muck normal and improve our models' predictive performance.

In [456]:

```
#Transformation non-normal features
#take out sqft_basement,yr_renovated, these two features after log and
#or infinity values I tried diffrent normalization methods and still sa
#I decided to remove sqft_basement,yr_renovated sicnce the correlation

continuous = ['price', 'sqft_living', 'sqft_lot', 'sqft_above',
              'yr_built', 'sqft_living15', 'sqft_lot15' ]


categoricals= ['floors', 'condition','bedrooms', 'view', 'waterfront',

#for feat in non_normal:
    #df[feat]=df[feat].map(lambda x: np.log(x))


kc_cont = df[continuous]

#log features
log_names = [f'{column}_log' for column in kc_cont.columns]

kc_log = np.log(kc_cont)
kc_log.columns = log_names



# normalize (subract mean and divide by std)

def normalize(feature):
    return (feature - feature.mean()) / feature.std()

kc_log_norm =kc_log.apply(normalize)
```

One-Hot Encoding Categorical Columns

Categorical columns: Bedrooms, bathrooms,floors, waterfront, view, grade, condition and zipcode. We change their type to 'category' and make new columns by using pandas get_dummies() function.

In [457]:

```
#Categorial features
df['bedrooms'] = df['bedrooms'].astype('category')
df['floors'] = df['floors'].astype('category')
df['waterfront'] = df['waterfront'].astype('category')
df['view'] = df['view'].astype('category')
df['condition'] = df['condition'].astype('category')
df['grade'] = df['grade'].astype('category')
df['zipcode']=df['zipcode'].astype('category')
df['bathrooms']=df['bathrooms'].astype('category')

df_dummy = pd.get_dummies(df[categoricals], prefix=categoricals, drop_f
```

In [42]:

```
#Data normalization
#def normalize(feature):
   # return (feature - feature.mean())/feature.std()


#for feat in ['bathrooms', 'sqft_living', 'sqft_lot', 'sqft_above', 'sq
            # 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15'
    #df[feat] = norm_feat(df[feat])
```

In [458]:

```
#Combine categorial and continuous features
preprocessed = pd.concat([kc_log_norm, df_dummy], axis=1)
preprocessed.head()
```

Out[458]:

	price_log	sqft_living_log	sqft_lot_log	sqft_above_log	yr_built_log	sqft_living15_log	sqft_
0	-1.444792	-1.127694	-0.381175	-0.746778	-0.538117	-1.031533	
1	0.330701	0.747609	-0.105607	0.705282	-0.675344	-0.312979	
2	-1.864331	-2.156133	0.252602	-1.764245	-1.296368	1.160623	
3	0.562685	0.094808	-0.516846	-1.024990	-0.196274	-0.985659	
4	0.223550	-0.276573	0.015941	0.095264	0.549696	-0.117724	

5 rows × 135 columns

In [43]:

```
# Dealing with Categorial columns
#Make sure Python recognizes these strings as category.

#df['bedrooms'] = df['bedrooms'].astype('category')
#df['floors'] = df['floors'].astype('category')
#df['waterfront'] = df['waterfront'].astype('category')
#df['view'] = df['view'].astype('category')
#df['condition'] = df['condition'].astype('category')
#df['grade'] = df['grade'].astype('category')
#df['bathrooms']=df['bathrooms'].astype('category')
#df['zipcode'] = df['zipcode'].astype('category')
#df['date'] = df['date'].astype('category')
```

In [12]:

```
#Create dummy variables for the categorial features
#feats = [ 'bedrooms', 'bathrooms',  'floors',
        # 'waterfront', 'view', 'condition', 'grade']
#df_feats = df[feats]
#df_feats_zipcode = df_feats.copy()
#df_feats = pd.get_dummies(df_feats, drop_first=True)
```

In [44]:

```
#floor_dummies = pd.get_dummies(df['floors'], prefix='floors', drop_fir
#condition_dummies = pd.get_dummies(df['condition'], prefix='con', drop
#grade_dummies = pd.get_dummies(df['grade'], prefix='grade', drop_first
#bedrooms_dummies = pd.get_dummies(df['bedrooms'], prefix='bed', drop_f
#bathrooms_dummies = pd.get_dummies(df['bathrooms'], prefix='bath', dro
#waterfront_dummies = pd.get_dummies(df['waterfront'], prefix='water',
#view_dummies = pd.get_dummies(df['view'], prefix='view', drop_first=Tr
```

In [45]:

```
#df = df.drop(['floors', 'condition','bedrooms', 'view', 'waterfront',
```

In [14]:

```
#df = pd.concat([df, view_dummies, bathrooms_dummies, waterfront_dummie
                      #floor_dummies,
                      #condition_dummies,
                     # grade_dummies,
                      #bedrooms_dummies
                      #], axis=1)
#df.head()
```

In [32]:
```
df.dtypes
```

Out[32]:
```
date           datetime64[ns]
price                 float64
sqft_living           float64
sqft_lot              float64
sqft_above            float64
                     ...
bed_7                    uint8
bed_8                    uint8
bed_9                    uint8
bed_10                   uint8
bed_11                   uint8
Length: 75, dtype: object
```

In [388]:
```
df.keys()
```

Out[388]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lon
g',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

Linear regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. We will start modelling our data using two variables: the target (price) and other features to check if there is a linear relationship. The weakness of the linear regression model Weaknessesis when it performs poorly when there are non-linear relationships.

In [389]:

```
#Single linear regression
import statsmodels.formula.api as smf
f = 'price~sqft_above'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_above': [df.sqft_above.min(), df.sqft_above
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_above', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_above", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.34062439602857164
Intercept     147248.157357
sqft_above       210.060588
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
8: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
```

In [14]:

```
f = 'price~sqft_lot'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_lot': [df.sqft_lot.min(), df.sqft_lot.max()
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_lot', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_lot", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.009728480794731209
Intercept    508026.025566
sqft_lot          0.685566
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

In [390]:

```
f = 'price~sqft_living'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_living': [df.sqft_living.min(), df.sqft_liv
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_living', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_living", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.46050140020301855
Intercept      57988.127483
sqft_living      224.444098
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launche
r.py:16: UserWarning: Matplotlib is currently using module://ipykerne
l.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
  app.launch_new_instance()
```

In [392]:

```
f = 'price~bathrooms'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'bathrooms': [df.bathrooms.min(), df.bathrooms.ma
preds = model.predict(X_new)
df.plot(kind='scatter', x='bathrooms', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "bathrooms", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.25765215401678143
Intercept    111508.817838
bathrooms    193845.888862
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

In [394]:

```
f = 'price~sqft_basement'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'sqft_basement': [df.sqft_basement.min(), df.sqft
preds = model.predict(X_new)
df.plot(kind='scatter', x='sqft_basement', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "sqft_basement", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.07939243950679753
Intercept        466034.390734
sqft_basement       187.596030
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

In [396]:

```
f = 'price~zipcode'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'zipcode': [df.zipcode.min(), df.zipcode.max()]})
preds = model.predict(X_new)
df.plot(kind='scatter', x='zipcode', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "zipcode", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.002083310448225628
Intercept    2.435655e+07
zipcode     -2.430538e+02
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launche
r.py:16: UserWarning: Matplotlib is currently using module://ipykerne
l.pylab.backend_inline, which is a non-GUI backend, so cannot show the
figure.
  app.launch_new_instance()
```

In [398]:

```
f = 'price~bedrooms'
model = smf.ols(formula=f, data=df).fit()
print ('R-Squared:',model.rsquared)
print (model.params)
X_new = pd.DataFrame({'bedrooms': [df.bedrooms.min(), df.bedrooms.max()
preds = model.predict(X_new)
df.plot(kind='scatter', x='bedrooms', y='price');
plt.plot(X_new, preds, c='red', linewidth=2);
plt.show()
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "bedrooms", fig=fig)
plt.show()

residuals = model.resid
fig = sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=Tru
fig.show()
```

```
R-Squared: 0.10403493835924071
Intercept    174215.931999
bedrooms     102348.574257
dtype: float64
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: UserWarning: Matplotlib is currently using module://ipykernel.pylab.back
end_inline, which is a non-GUI backend, so cannot show the figure.
  app.launch_new_instance()
```

Preliminary analysis

As preliminary analysis, sqft_living, sqft_above, bathrooms are among the features best performing for linear regression model. Other variables had non-linear relationship explaned by low r-squared.

In [16]:

```
preprocessed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21380 entries, 0 to 21596
Columns: 135 entries, price_log to zipcode_98199
dtypes: float64(7), uint8(128)
memory usage: 3.9 MB
```

Multiple linear regression model

Multiple linear regression is a model for predicting the value of one dependent variable based on two or more independent variables.

Baseline model

In this first modeling attempt we push all parameters to the model. We see how it works. Then we are going to drop some parameters according to the p-values and variance inflation factors.

In [399]:

```
#Multiple linear regression model with statsmodels

X = preprocessed.drop('price_log', axis=1)
y = preprocessed['price_log']
X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

Out[399]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.874
Method:	Least Squares	**F-statistic:**	1111.
Date:	Tue, 09 Mar 2021	**Prob (F-statistic):**	0.00
Time:	18:07:07	**Log-Likelihood:**	-8096.7
No. Observations:	21380	**AIC:**	1.646e+04
Df Residuals:	21245	**BIC:**	1.754e+04
Df Model:	134		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.3096	0.406	-3.224	0.001	-2.106	-0.513

In [400]:

```
#plt.scatter(model.predict(X), model.resid, alpha=0.5)
#plt.plot(model.predict(X), [0 for i in range(len(X))])
#plt.gca().set(xlabel='Fitted', ylabel='Residuals',
               #title='Scatter Plot of the Residuals vs. Fitted model ')
#plt.xticks(fontsize=15)
#plt.yticks(fontsize=14)
#plt.show()
```

In [78]:

```
len(preprocessed.columns)
```

Out[78]: 138

In [80]:

```
len(X.columns)
```

Out[80]: 137

In [401]:

```
#Run same model using scikit-learn
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X, y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normaliz
# coefficients
linreg.coef_

```

Out[401]:

```
array([ 2.42104035e-01,  1.27291987e-01,  1.10105047e-01, -1.68902857e-02,
        9.25541674e-02, -3.28480363e-02,  1.32476770e-02, -1.81703642e-02,
       -3.59323358e-03, -1.49007086e-01, -1.61566109e-01,  3.03192215e-01,
        5.77514618e-01,  6.48681575e-01,  7.74165807e-01, -2.75326586e-02,
       -7.07196203e-02, -7.66632413e-02, -1.19951687e-01, -1.80804889e-01,
       -4.27339288e-01, -3.73206695e-01, -2.58242424e-01, -5.26433787e-01,
        2.40275875e-01,  2.21106677e-01,  2.06446134e-01,  3.45362274e-01,
        5.49739759e-01,  8.37190459e-01,  3.90601973e-01,  3.52414476e-01,
        3.00078691e-01,  3.48594115e-01,  3.95206782e-01,  3.94670882e-01,
        4.30344986e-01,  4.49809004e-01,  4.86621766e-01,  4.80531134e-01,
        5.50091619e-01,  5.58282107e-01,  6.19466712e-01,  5.84598909e-01,
        6.62223546e-01,  5.53847450e-01,  6.77390742e-01,  7.00012293e-01,
        6.78408737e-01,  8.36833707e-01,  4.90529498e-01, -3.44752538e-01,
        6.47669860e-01,  7.68250447e-02, -2.90431598e-01, -1.04969668e+00,
       -1.06197395e+00, -9.13013830e-01, -7.56573900e-01, -5.85907779e-01,
       -3.60122102e-01, -2.14119779e-01, -3.69869324e-02,  1.57255753e-01,
        1.11586476e+00, -1.43119949e-03,  4.32273815e-02,  2.23292447e+00,
        1.47320764e+00,  1.28631289e+00,  1.32842474e+00,  1.32169978e+00,
        4.85701394e-01,  8.97549947e-01,  6.00380541e-01,  6.34352978e-01,
        9.67854418e-02, -4.06516993e-02,  8.58023759e-01,  1.03173840e+00,
        8.34684185e-01,  1.20848335e+00,  1.02638987e-01,  1.47125850e-01,
       -3.04228367e-02,  1.57902847e+00,  1.08993222e+00,  3.36989404e-01,
        2.53245405e+00,  1.76872920e+00,  1.34696657e-01,  6.48956346e-01,
        1.28273186e+00,  1.17366359e+00,  3.04082517e-01,  6.54638198e-01,
        3.33999038e-01,  6.87413189e-01,  8.13611012e-01,  6.28246781e-01,
        9.54236051e-01,  1.10772805e+00,  1.12598573e+00,  8.76083374e-01,
        4.09996463e-02,  2.00706712e+00,  1.75563234e+00,  1.95921582e+00,
        7.70250101e-01,  1.80019664e+00,  7.81687890e-01,  2.06194649e+00,
        2.11337695e+00,  1.70641094e+00,  1.60097802e+00,  1.71256749e+00,
        9.93287459e-01,  2.02307316e+00,  1.71233610e+00,  1.18334583e+00,
        1.18366491e+00,  9.77000083e-01,  1.43859912e+00,  1.42865055e+00,
        6.02044846e-01,  3.52551011e-01,  8.87629644e-01,  6.50537707e-01,
        1.74631980e-01,  1.19867929e+00,  3.33890721e-01,  2.15123042e-01,
        1.47028221e-01,  1.77523671e+00])
```

In [20]:

```
# coefficients
linreg.intercept_
```

Out[20]: -1.3096455646917788

Step-wise selection for model refinement

In [18]:

```
#Stepwise selection using p-values
#stepwise selection, you start with an empty model (which only includes
#and each time, the variable that has an associated parameter estimate
#The algorithm stops when no variables can be added or removed given th
def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    """
    Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_ou
        verbose - whether to print the sequence of inclusions and exclu
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    See https://en.wikipedia.org/wiki/Stepwise_regression for the detai
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_featu

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).f
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feat
        if not changed:
            break
    return included
```

In [19]:

```


result = stepwise_selection(X, y, verbose = True)
print('resulting features:')
print(result)
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\ipykernel_launcher.py:2
9: DeprecationWarning: The default dtype for empty Series will be 'object'
instead of 'float64' in a future version. Specify a dtype explicitly to sil
ence this warning.

Add  grade_7                        with p-value 0.0
Add  sqft_living15_log              with p-value 0.0
Add  sqft_living_log                with p-value 0.0
Add  yr_built_log                   with p-value 0.0
Add  grade_6                        with p-value 3.9122e-309
Add  zipcode_98023                  with p-value 3.64094e-229
Add  zipcode_98004                  with p-value 4.18601e-182
Add  grade_8                        with p-value 7.45721e-171
Add  grade_5                        with p-value 0.0
Add  sqft_lot15_log                 with p-value 1.87527e-159
Add  zipcode_98042                  with p-value 6.26087e-123
Add  zipcode_98092                  with p-value 3.13684e-127
Add  zipcode_98003                  with p-value 5.77604e-133
Add  zipcode_98058                  with p-value 3.07303e-103
Add  zipcode_98030                  with p-value 7.13028e-110
Add  zipcode_98031                  with p-value 2.98337e-113
Add  zipcode_98038                  with p-value 1.55686e-116
Add  zipcode_98168                  with p-value 8.54474e-119
Add  zipcode_98198                  with p-value 1.30982e-126
Add  zipcode_98002                  with p-value 4.10201e-128
Add  zipcode_98055                  with p-value 2.23014e-128
Add  zipcode_98032                  with p-value 1.51037e-122
Add  zipcode_98178                  with p-value 2.83577e-116
Add  waterfront_1.0                 with p-value 3.24632e-107
Add  zipcode_98022                  with p-value 2.11608e-94
Add  zipcode_98188                  with p-value 1.85252e-88
Add  grade_9                        with p-value 1.9679e-88
Add  zipcode_98033                  with p-value 4.22321e-79
Add  zipcode_98040                  with p-value 6.94993e-82
Add  grade_4                        with p-value 9.3666e-75
Add  zipcode_98115                  with p-value 1.04377e-70
Add  zipcode_98117                  with p-value 2.2105e-76
Add  zipcode_98103                  with p-value 1.51862e-90
Add  zipcode_98112                  with p-value 1.83738e-97
Add  zipcode_98199                  with p-value 2.71622e-82
Add  zipcode_98105                  with p-value 1.20755e-90
Add  zipcode_98119                  with p-value 6.74881e-89
Add  zipcode_98107                  with p-value 5.95452e-98
Add  zipcode_98116                  with p-value 6.60211e-84

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-982b9c2708b5> in <module>
----> 1 result = stepwise_selection(X, y, verbose = True)
      2 print('resulting features:')
```

```
      3 print(result)

<ipython-input-18-5413f9156bd7> in stepwise_selection(X, y, initial_list, t
hreshold_in, threshold_out, verbose)
     47             changed=True
     48             worst_feature = pvalues.argmax()
---> 49             included.remove(worst_feature)
     50             if verbose:
     51                 print('Drop {:30} with p-value {:.6}'.format(worst_
feature, worst_pval))

ValueError: list.remove(x): x not in list
```

In [402]:

```
#import statsmodels.api as sm
#X_fin = X[result]
#X_with_intercept = sm.add_constant(X_fin)
#model = sm.OLS(y,X_with_intercept).fit()
#model.summary()
```

Stepwise slection did not work, tried several ways to fix it. I will use feature selection with Sckit-leearn

Scikit-learn and feature ranking for model refinement

In [423]:

```
#Feature ranking with recursive feature elimination
#Scikit-learn also provides a few functionalities for feature selection
#Elimination selects the pre-specified $n$ most important features.
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
selector = RFE(linreg, n_features_to_select=10)
selector = selector.fit(X, preprocessed['price_log'])

selector.support_
```

Out[423]:

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
        True,  True, False, False,  True,  True, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False])
```

In [411]:

```
pd.DataFrame(data=selector.support_.reshape(1,-1), columns= X.columns)
```

Out[411]:

	sqft_living_log	sqft_lot_log	sqft_above_log	yr_built_log	sqft_living15_log	sqft_lot15_log	f
0	False	False	False	False	False	False	

1 rows × 134 columns

In [435]:

```
selector.ranking_
```

Out[435]:

```
array([ 35,  99, 105, 122, 107, 118, 123, 120, 124,  72,  71,  95,  69,
        68,  67, 119, 115, 114, 113, 110,  60,  36,  98,  34,  61,  63,
        62,  52,   1,  25,  82,  83,  93,  92,  90,  91,  89,  87,  84,
        85,  80,  81,  77,  78,  76,  79,  74,  73,  75,   1,  86,  31,
        88, 116,  33,  58,  55,  56,  57,  59,  66,  94, 109,   1,   1,
       125, 111,   1,   1,   7,  13,  18,  53,  37,  46,  49, 108, 117,
        27,  17,  42,  14, 106, 101, 121,   8,  29,  64,   1,   1, 102,
        44,  12,  11,  96,  48,  65,  45,  43,  40,  24,   9,   4,  10,
       112,   5,  19,   2,  51,  21,  50,   1,   1,  16,  20,  22,  39,
         3,  15,  30,  32,  38,  28,  26,  54,  70,  41,  47, 103,  23,
        97, 100, 104,   6])
```

In [ ]:

```

```

In [436]:

```
estimators = selector.estimator_
print(estimators.coef_)
print(estimators.intercept_)
```

```
[1.52168123 2.18160211 2.15387835 2.83006428 1.82448471 1.08002728
 2.35388747 1.67048148 1.1487964  1.38862804]
-0.10062918562769398
```

In [426]:

```

pd.DataFrame(data=estimators.coef_.reshape(1,-1), columns= X.columns[se
```

Out[426]:

	view_4.0	bathrooms_5.5	grade_12	grade_13	zipcode_98004	zipcode_98005	zipcode_9803
0	1.521681	2.181602	2.153878	2.830064	1.824485	1.080027	2.35388

In [434]:

```
#Refine the model and fit  the linear regression model again using the
selected_columns = X.columns[selector.support_ ]
linreg.fit(X[selected_columns],y)

```

Out[434]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [433]:

```
#predict yhat using your model. You can use .predict() in scikit-learn.
yhat = linreg.predict(X[selected_columns])
```

In [429]:

```
SS_Residual = np.sum((y-yhat)**2)
SS_Total = np.sum((y-np.mean(y))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
adjusted_r_squared = 1 - (1-r_squared)*(len(y)-1)/(len(y)-X[selected_co
```

In [430]:

```
r_squared
```

Out[430]:

```
0.15702488067239706
```

In [431]:

```
adjusted_r_squared
```

Out[431]:

```
0.15663039561491776
```

I have chosen 10 features to for my multiple linear regression model. Most important feature: zipcode, bathrooms5.5, viiew4, grade12 and 13. However, this model had a low r-squared of 0.15. Another approach for modelling is needed.

Baseline model refinement

I will refine the baseline model, by selecting manually only features with p-value less than 0.05.

In [437]:

```
#X = ['sqft_living_log', 'sqft_above_log', 'yr_built_log',
       'sqft_living15_log', 'sqft_lot15_log', 'zipcode_log', 'floors_1.
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'condition_3', 'condition_4', 'condition_5', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7', 'view_1.
       'view_3.0', 'view_4', 'waterfront_1.0', 'bathrooms_0.75',
       'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.5', 'bathrooms_1
       'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2
       'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3
       'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'bathrooms_4
       'bathrooms_5.0', 'bathrooms_5.25', 'bathrooms_5.5', 'bathrooms_5
       'bathrooms_6.0', 'bathrooms_6.25', 'bathrooms_6.5', 'bathrooms_7
       'grade_9', 'grade_10',   'grade_11', 'grade_12', 'grade_13']
#X=pd.DataFrame(['sqft_living_log', 'sqft_above_log', 'yr_built_log',
       'sqft_living15_log', 'sqft_lot15_log', 'zipcode_log', 'floors_1.
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'condition_3', 'condition_4', 'condition_5', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7', 'view_1.
       'view_3.0', 'view_4', 'waterfront_1.0', 'bathrooms_0.75',
       'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.5', 'bathrooms_1
       'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.5', 'bathrooms_2
       'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.5', 'bathrooms_3
       'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.5', 'bathrooms_4
       'bathrooms_5.0', 'bathrooms_5.25', 'bathrooms_5.5', 'bathrooms_5
       'bathrooms_6.0', 'bathrooms_6.25', 'bathrooms_6.5', 'bathrooms_7
       'grade_9', 'grade_10',   'grade_11', 'grade_12', 'grade_13'])

#arr=X.to_numpy()
```

```
  File "<ipython-input-437-4f2f77319db6>", line 2
    'sqft_living15_log', 'sqft_lot15_log', 'zipcode_log', 'floors_1.5',
    ^
IndentationError: unexpected indent
```

In [86]:

```
#arr=arr.reshape(-1,1)
```

In [29]:

```
#print(arr)
```

In [126]:

```
# Make sure you have all the columns you need
#arr = ['sqft_living_log', 'sqft_above_log', 'yr_built_log',
       'sqft_living15_log', 'sqft_lot15_log', 'zipcode_log', 'floors_1.
       'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.5',
       'condition_3', 'condition_4', 'condition_5', 'bedrooms_3',
       'bedrooms_4', 'bedrooms_5', 'bedrooms_6', 'bedrooms_7', 'view_1.
        'view_3.0', 'view_4.0', 'waterfront_1.0',
       'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_
       'bathrooms_1.75',  'bathrooms_2.25', 'bathrooms_2.5',
       'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_
       'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_
       'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25', 'bathrooms_
       'bathrooms_5.75', 'bathrooms_6.0', 'bathrooms_6.25', 'bathrooms_
        'bathrooms_7.75',
        'grade_9',
       'grade_10', 'grade_11', 'grade_12', 'grade_13'
      ]
```

In [127]:

```
#preprocessed.columns
```

Out[127]:

```
Index(['price_log', 'sqft_living_log', 'sqft_lot_log', 'sqft_above_log',
       'yr_built_log', 'sqft_living15_log', 'sqft_lot15_log', 'zipcode_lo
g',
       'floors_1.5', 'floors_2.0', 'floors_2.5', 'floors_3.0', 'floors_3.
5',
       'condition_2', 'condition_3', 'condition_4', 'condition_5',
       'bedrooms_2', 'bedrooms_3', 'bedrooms_4', 'bedrooms_5', 'bedrooms_
6',
       'bedrooms_7', 'bedrooms_8', 'bedrooms_9', 'bedrooms_10', 'bedrooms_1
1',
       'view_1.0', 'view_2.0', 'view_3.0', 'view_4.0', 'waterfront_1.0',
       'bathrooms_0.75', 'bathrooms_1.0', 'bathrooms_1.25', 'bathrooms_1.
5',
       'bathrooms_1.75', 'bathrooms_2.0', 'bathrooms_2.25', 'bathrooms_2.
5',
       'bathrooms_2.75', 'bathrooms_3.0', 'bathrooms_3.25', 'bathrooms_3.
5',
       'bathrooms_3.75', 'bathrooms_4.0', 'bathrooms_4.25', 'bathrooms_4.
5',
       'bathrooms_4.75', 'bathrooms_5.0', 'bathrooms_5.25', 'bathrooms_5.
5',
       'bathrooms_5.75', 'bathrooms_6.0', 'bathrooms_6.25', 'bathrooms_6.
5',
       'bathrooms_6.75', 'bathrooms_7.5', 'bathrooms_7.75', 'bathrooms_8.
0',
       'grade_4', 'grade_5', 'grade_6', 'grade_7', 'grade_8', 'grade_9',
       'grade_10', 'grade_11', 'grade_12', 'grade_13'],
      dtype='object')
```

In [128]:
```
#X = preprocessed[arr]
#X = preprocessed.to_numpy()
#X
```

Out[128]:
```
array([[-1.40199803, -1.12555595, -0.38842975, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.27993836,  0.7094463 , -0.1132408 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.79943049, -2.13189317,  0.24447519, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-0.2730055 , -1.46906114, -1.97537803, ...,  0.        ,
         0.        ,  0.        ],
       [-0.28295461, -0.40774444, -1.34311378, ...,  0.        ,
         0.        ,  0.        ],
       [-0.67729031, -1.46906114, -2.22685757, ...,  0.        ,
         0.        ,  0.        ]])
```

In [129]:
```
#y = preprocessed.price_log.to_numpy().reshape(-1,1)

```

Out[129]:
```
array([[-1.40199803],
       [ 0.27993836],
       [-1.79943049],
       ...,
       [-0.2730055 ],
       [-0.28295461],
       [-0.67729031]])
```

In [130]:
```
#linreg = LinearRegression()
#linreg.fit(X, y)
```

Out[130]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [438]:
```
#LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normali
# coefficients
#linreg.coef_, linreg.intercept_
```

In [30]:

```
#All the features in model_summary, select only those with p_value <0.6
z=['sqft_living_log',
'sqft_lot_log',
'sqft_above_log',
'yr_built_log',
'sqft_living15_log',
'sqft_lot15_log',
'floors_1.5',
'floors_2.0',
'floors_2.5',
'floors_3.0',
'floors_3.5',
'condition_2',
'condition_3',
'condition_4',
'condition_5',
'bedrooms_2',
'bedrooms_3',
'bedrooms_4',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_9',
'bedrooms_10',
'bedrooms_11',
'view_1.0',
'view_2.0',
'view_3.0',
'view_4.0',
'waterfront_1',
'bathrooms_0.75',
'bathrooms_1.0',
'bathrooms_1.25',
'bathrooms_1.5',
'bathrooms_1.75',
'bathrooms_2.0',
'bathrooms_2.25',
'bathrooms_2.5',
'bathrooms_2.75',
'bathrooms_3.0',
'bathrooms_3.25',
'bathrooms_3.5',
'bathrooms_3.75',
'bathrooms_4.0',
'bathrooms_4.25',
'bathrooms_4.5',
'bathrooms_4.75',
'bathrooms_5.0',
'bathrooms_5.25',
'bathrooms_5.5',
'bathrooms_5.75',
'bathrooms_6.0',
'bathrooms_6.25',
'bathrooms_6.5',
'bathrooms_6.75',
```

```
'bathrooms_7.5',
'bathrooms_7.75',
'bathrooms_8.0',
'grade_4',
'grade_5',
'grade_6',
'grade_7',
'grade_8',
'grade_9',
'grade_10',
'grade_11',
'grade_12',
'grade_13',
'zipcode_98002',
'zipcode_98003',
'zipcode_98004',
'zipcode_98005',
'zipcode_98006',
'zipcode_98007',
'zipcode_98008',
'zipcode_98010',
'zipcode_98011',
'zipcode_98014',
'zipcode_98019',
'zipcode_98022',
'zipcode_98023',
'zipcode_98024',
'zipcode_98027',
'zipcode_98028',
'zipcode_98029',
'zipcode_98030',
'zipcode_98031',
'zipcode_98032',
'zipcode_98033',
'zipcode_98034',
'zipcode_98038',
'zipcode_98039',
'zipcode_98040',
'zipcode_98042',
'zipcode_98045',
'zipcode_98052',
'zipcode_98053',
'zipcode_98055',
'zipcode_98056',
'zipcode_98058',
'zipcode_98059',
'zipcode_98065',
'zipcode_98070',
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
```

```
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199']
```

In [31]:

```
# Run Multiple linear regression model with statsmodels again tp produc
X = preprocessed.drop('price_log', axis=1)
y = preprocessed['price_log']
X_int = sm.add_constant(X)
model = sm.OLS(y,X_int).fit()
model.summary()
```

Out[31]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.874
Method:	Least Squares	**F-statistic:**	1111.
Date:	Mon, 08 Mar 2021	**Prob (F-statistic):**	0.00
Time:	18:57:41	**Log-Likelihood:**	-8096.7
No. Observations:	21380	**AIC:**	1.646e+04
Df Residuals:	21245	**BIC:**	1.754e+04
Df Model:	134		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.3096	0.406	-3.224	0.001	-2.106	-0.513

Model refinement 1

In [33]:

```
#Select features with p-value <0.05
z_keep=['sqft_living_log',
'sqft_lot_log',
'sqft_above_log',
'yr_built_log',
'sqft_living15_log',
'sqft_lot15_log',
'floors_2.0',
'floors_3.0',
'condition_2',
'condition_3',
'condition_4',
'condition_5',
'bedrooms_3',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_10',
'view_1.0',
'view_2.0',
'view_3.0',
'view_4.0',
'bathrooms_0.75',
'bathrooms_1.0',
'bathrooms_1.25',
'bathrooms_1.5',
'bathrooms_1.75',
'bathrooms_2.0',
'bathrooms_2.25',
'bathrooms_2.5',
'bathrooms_2.75',
'bathrooms_3.0',
'bathrooms_3.25',
'bathrooms_3.5',
'bathrooms_3.75',
'bathrooms_4.0',
'bathrooms_4.25',
'bathrooms_4.5',
'bathrooms_4.75',
'bathrooms_5.0',
'bathrooms_5.25',
'bathrooms_5.5',
'bathrooms_5.75',
'grade_4',
'grade_5',
'grade_6',
'grade_7',
'grade_13',
'zipcode_98004',
'zipcode_98005',
'zipcode_98006',
'zipcode_98007',
'zipcode_98008',
'zipcode_98010',
'zipcode_98011',
```

```
'zipcode_98014',
'zipcode_98019',
'zipcode_98022',
'zipcode_98024',
'zipcode_98027',
'zipcode_98028',
'zipcode_98029',
'zipcode_98030',
'zipcode_98031',
'zipcode_98033',
'zipcode_98034',
'zipcode_98038',
'zipcode_98039',
'zipcode_98040',
'zipcode_98042',
'zipcode_98045',
'zipcode_98052',
'zipcode_98053',
'zipcode_98055',
'zipcode_98056',
'zipcode_98058',
'zipcode_98059',
'zipcode_98065',
'zipcode_98070',
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
```

```
'zipcode_98199']
z_filtered=preprocessed[z_keep]

y = preprocessed['price_log']

```

In [36]:

```
X_int = sm.add_constant(z_filtered)
model2 = sm.OLS(y,X_int).fit()
model2.summary()
```

Out[36]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.865
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	1208.
Date:	Mon, 08 Mar 2021	**Prob (F-statistic):**	0.00
Time:	19:25:24	**Log-Likelihood:**	-8914.3
No. Observations:	21380	**AIC:**	1.806e+04
Df Residuals:	21266	**BIC:**	1.897e+04
Df Model:	113		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.8644	0.142	-13.106	0.000	-2.143	-1.586

Model refinement 2

In [37]:

```
z_keep2=['sqft_living_log',
'sqft_lot_log',
'sqft_above_log',
'sqft_living15_log',
'sqft_lot15_log',
'floors_3.0',
'condition_2',
'condition_3',
'condition_4',
'condition_5',
'bedrooms_3',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_10',
'view_1.0',
'view_2.0',
'view_3.0',
'view_4.0',
'bathrooms_0.75',
'bathrooms_1.0',
'bathrooms_1.25',
'bathrooms_1.5',
'bathrooms_1.75',
'bathrooms_2.0',
'bathrooms_2.25',
'bathrooms_2.5',
'bathrooms_2.75',
'bathrooms_3.0',
'bathrooms_3.25',
'bathrooms_3.5',
'bathrooms_3.75',
'bathrooms_4.0',
'bathrooms_4.25',
'bathrooms_4.5',
'bathrooms_4.75',
'bathrooms_5.0',
'bathrooms_5.25',
'bathrooms_5.5',
'bathrooms_5.75',
'grade_4',
'grade_5',
'grade_6',
'grade_7',
'grade_13',
'zipcode_98004',
'zipcode_98005',
'zipcode_98006',
'zipcode_98007',
'zipcode_98008',
'zipcode_98010',
'zipcode_98011',
'zipcode_98014',
'zipcode_98019',
'zipcode_98022',
```

```
'zipcode_98024',
'zipcode_98027',
'zipcode_98028',
'zipcode_98029',
'zipcode_98030',
'zipcode_98031',
'zipcode_98033',
'zipcode_98034',
'zipcode_98038',
'zipcode_98039',
'zipcode_98040',
'zipcode_98042',
'zipcode_98045',
'zipcode_98052',
'zipcode_98053',
'zipcode_98055',
'zipcode_98056',
'zipcode_98058',
'zipcode_98059',
'zipcode_98065',
'zipcode_98070',
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199']
```

In [38]:

```
z_filtered2=preprocessed[z_keep2]

y = preprocessed['price_log']

```

In [39]:

```
X_int = sm.add_constant(z_filtered2)
model3 = sm.OLS(y,X_int).fit()
model3.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.865
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	1241.
Date:	Mon, 08 Mar 2021	**Prob (F-statistic):**	0.00
Time:	19:28:12	**Log-Likelihood:**	-8915.6
No. Observations:	21380	**AIC:**	1.805e+04
Df Residuals:	21269	**BIC:**	1.894e+04
Df Model:	110		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.8591	0.142	-13.094	0.000	-2.137	-1.581

In [41]:

```
#Check for multicollinearity
#multicollinearity which is a relation of 3 or more variables.
#One test for this is the variance inflation factor. Typically,
#variables with a vif of 5 or greater (or more definitively 10 or great

vif = [variance_inflation_factor(z_filtered2.values, i) for i in range(
list(zip(z_keep2, vif))
```

Out[41]:

```
[('sqft_living_log', 7.092202330910084),
 ('sqft_lot_log', 6.92340343225942),
 ('sqft_above_log', 5.241619722142487),
 ('sqft_living15_log', 3.0945456956560338),
 ('sqft_lot15_log', 7.008400535337593),
 ('floors_3.0', 1.3172980631721456),
 ('condition_2', 5.501879733913908),
 ('condition_3', 373.0665255850722),
 ('condition_4', 151.84778422273806),
 ('condition_5', 45.80838947753708),
 ('bedrooms_3', 2.2047087487060106),
 ('bedrooms_5', 1.2943642047364359),
 ('bedrooms_6', 1.0995129367014953),
 ('bedrooms_7', 1.045019905021494),
 ('bedrooms_8', 1.0171119198953704),
 ('bedrooms_10', 1.0439586331947595),
 ('view_1.0', 1.0679401108537938),
 ('view_2.0', 1.144706997816453),
 ('view_3.0', 1.118825322318213),
```

Model refinement 3

Select features with vif < 5.

In [42]:

```
z_keep3=[
'yr_built_log',
'sqft_living15_log',
'floors_2.0',
'floors_3.0',
'bedrooms_3',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_10',
'view_1.0',
'view_2.0',
'view_3.0',
'view_4.0',
'bathrooms_0.75',
'bathrooms_1.25',
'bathrooms_1.5',
'bathrooms_3.75',
'bathrooms_4.0',
'bathrooms_4.25',
'bathrooms_4.5',
'bathrooms_4.75',
'bathrooms_5.0',
'bathrooms_5.25',
'bathrooms_5.5',
'bathrooms_5.75',
'grade_4',
'grade_5',
'grade_6',
'grade_7',
'grade_13',
'zipcode_98004',
'zipcode_98005',
'zipcode_98006',
'zipcode_98007',
'zipcode_98008',
'zipcode_98010',
'zipcode_98011',
'zipcode_98014',
'zipcode_98019',
'zipcode_98022',
'zipcode_98024',
'zipcode_98027',
'zipcode_98028',
'zipcode_98029',
'zipcode_98030',
'zipcode_98031',
'zipcode_98033',
'zipcode_98034',
'zipcode_98038',
'zipcode_98039',
'zipcode_98040',
'zipcode_98042',
'zipcode_98045',
'zipcode_98052',
```

```
'zipcode_98053',
'zipcode_98055',
'zipcode_98056',
'zipcode_98058',
'zipcode_98059',
'zipcode_98065',
'zipcode_98070',
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199']

z_filtered3=preprocessed[z_keep3]

y = preprocessed['price_log']

```

In [43]:

```
X_int = sm.add_constant(z_filtered3)
model4 = sm.OLS(y,X_int).fit()
model4.summary()
```

Out[43]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.799
Model:	OLS	**Adj. R-squared:**	0.798
Method:	Least Squares	**F-statistic:**	880.7
Date:	Mon, 08 Mar 2021	**Prob (F-statistic):**	0.00
Time:	19:33:33	**Log-Likelihood:**	-13191.
No. Observations:	21380	**AIC:**	2.658e+04
Df Residuals:	21283	**BIC:**	2.735e+04
Df Model:	96		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.7736	0.014	-55.152	0.000	-0.801	-0.746

In [44]:
```
vif = [variance_inflation_factor(z_filtered3.values, i) for i in range(
list(zip(z_keep3, vif))
```

Out[44]:
```
[('yr_built_log', 2.549952738155994),
 ('sqft_living15_log', 2.2615600757045438),
 ('floors_2.0', 2.782653632157758),
 ('floors_3.0', 1.5102513564857356),
 ('bedrooms_3', 1.9910440759872903),
 ('bedrooms_5', 1.197714697301399),
 ('bedrooms_6', 1.0695223631797595),
 ('bedrooms_7', 1.039798031379654),
 ('bedrooms_8', 1.0131302160054432),
 ('bedrooms_10', 1.0426469988552465),
 ('view_1.0', 1.0636540683738802),
 ('view_2.0', 1.1339360230814306),
 ('view_3.0', 1.1063947505391323),
 ('view_4.0', 1.0798662521645823),
 ('bathrooms_0.75', 1.1272642767225578),
 ('bathrooms_1.25', 1.0064006166729709),
 ('bathrooms_1.5', 1.1178306911548526),
 ('bathrooms_3.75', 1.0338734616517145),
 ('bathrooms_4.0', 1.0403125669636504),
 ('bathrooms_4.25', 1.0214072907570482),
 ('bathrooms_4.5', 1.0417931516304224),
 ('bathrooms_4.75', 1.0092469540666005),
 ('bathrooms_5.0', 1.0206101885042045),
 ('bathrooms_5.25', 1.048050815733056),
 ('bathrooms_5.5', 1.0132863115403825),
 ('bathrooms_5.75', 1.0166421198962796),
 ('grade_4', 1.1196524733540665),
 ('grade_5', 1.1514110640269983),
 ('grade_6', 1.877384949285298),
 ('grade_7', 2.755076394948924),
 ('grade_13', 1.002952737727065),
 ('zipcode_98004', 1.0656011031039823),
 ('zipcode_98005', 1.029314250547672),
 ('zipcode_98006', 1.1160199262015873),
 ('zipcode_98007', 1.030951573572533),
 ('zipcode_98008', 1.0486059906797984),
 ('zipcode_98010', 1.0267008599216987),
 ('zipcode_98011', 1.048132757839573),
 ('zipcode_98014', 1.039564384959711),
 ('zipcode_98019', 1.0632388857690607),
 ('zipcode_98022', 1.0667495531815443),
 ('zipcode_98024', 1.0226839974564288),
 ('zipcode_98027', 1.0911353605503664),
 ('zipcode_98028', 1.0580687593072846),
 ('zipcode_98029', 1.0936292444854774),
 ('zipcode_98030', 1.0643104609083558),
 ('zipcode_98031', 1.0678930691254336),
 ('zipcode_98033', 1.0899298118124618),
 ('zipcode_98034', 1.1190945025118542),
 ('zipcode_98038', 1.2031679715639154),
 ('zipcode_98039', 1.0104142041618775),
 ('zipcode_98040', 1.079711747199037),
```

```
('zipcode_98042', 1.1440506681993956),
('zipcode_98045', 1.0554805409177108),
('zipcode_98052', 1.1124217663149552),
('zipcode_98053', 1.1031442866215027),
('zipcode_98055', 1.0620205348399192),
('zipcode_98056', 1.1000966743906817),
('zipcode_98058', 1.0819472506627372),
('zipcode_98059', 1.1307739652130635),
('zipcode_98065', 1.1343252027844615),
('zipcode_98070', 1.0459584618624276),
('zipcode_98072', 1.0569652237931255),
('zipcode_98074', 1.1103117033972134),
('zipcode_98075', 1.1292418761529415),
('zipcode_98077', 1.0590282637763508),
('zipcode_98092', 1.07903842324831),
('zipcode_98102', 1.0404121980494476),
('zipcode_98103', 1.2760147708879066),
('zipcode_98105', 1.0785095210413556),
('zipcode_98106', 1.1046770008848252),
('zipcode_98107', 1.1058240030308535),
('zipcode_98108', 1.050363246977627),
('zipcode_98109', 1.0435906804368056),
('zipcode_98112', 1.0891003570253297),
('zipcode_98115', 1.1558551893691158),
('zipcode_98116', 1.0974150184302847),
('zipcode_98117', 1.1579691438539512),
('zipcode_98118', 1.1574379458828472),
('zipcode_98119', 1.069491239947576),
('zipcode_98122', 1.101641246045518),
('zipcode_98125', 1.0999180593760451),
('zipcode_98126', 1.1112386772762086),
('zipcode_98133', 1.1193271443686477),
('zipcode_98136', 1.0788028180225686),
('zipcode_98144', 1.1040303558652729),
('zipcode_98146', 1.0860688517740478),
('zipcode_98148', 1.016336460062314),
('zipcode_98155', 1.0893775957425043),
('zipcode_98166', 1.0625950328844198),
('zipcode_98168', 1.090482520442282),
('zipcode_98177', 1.0514715354802207),
('zipcode_98178', 1.080344717224715),
('zipcode_98188', 1.0326954736934888),
('zipcode_98198', 1.0763827975561229),
('zipcode_98199', 1.0698295573439058)]
```

Model refinement 4

All feactures have now vif < 5, select only features with p_value less than 0.05

In [45]:

```
z_keep4=[
'yr_built_log',
'sqft_living15_log',
'floors_2.0',
'floors_3.0',
'bedrooms_3',
'bedrooms_5',
'bedrooms_6',
'view_1.0',
'view_2.0',
'view_3.0',
'view_4.0',
'bathrooms_0.75',
'bathrooms_1.5',
'bathrooms_3.75',
'bathrooms_4.0',
'bathrooms_4.25',
'bathrooms_4.5',
'bathrooms_4.75',
'bathrooms_5.0',
'bathrooms_5.25',
'bathrooms_5.5',
'bathrooms_5.75',
'grade_4',
'grade_5',
'grade_6',
'grade_7',
'grade_13',
'zipcode_98004',
'zipcode_98005',
'zipcode_98006',
'zipcode_98007',
'zipcode_98008',
'zipcode_98010',
'zipcode_98011',
'zipcode_98014',
'zipcode_98019',
'zipcode_98022',
'zipcode_98024',
'zipcode_98027',
'zipcode_98028',
'zipcode_98029',
'zipcode_98030',
'zipcode_98031',
'zipcode_98033',
'zipcode_98034',
'zipcode_98038',
'zipcode_98039',
'zipcode_98040',
'zipcode_98042',
'zipcode_98045',
'zipcode_98052',
'zipcode_98053',
'zipcode_98055',
'zipcode_98056',
'zipcode_98058',
```

```
'zipcode_98059',
'zipcode_98065',
'zipcode_98070',
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199']
z_filtered4=preprocessed[z_keep4]

y = preprocessed['price_log']
```

In [46]:

```
X_int = sm.add_constant(z_filtered4)
model5 = sm.OLS(y,X_int).fit()
model5.summary()
```

Out[46]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.799
Model:	OLS	**Adj. R-squared:**	0.798
Method:	Least Squares	**F-statistic:**	918.9
Date:	Mon, 08 Mar 2021	**Prob (F-statistic):**	0.00
Time:	21:12:36	**Log-Likelihood:**	-13193.
No. Observations:	21380	**AIC:**	2.657e+04
Df Residuals:	21287	**BIC:**	2.731e+04
Df Model:	92		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.7737	0.014	-55.169	0.000	-0.801	-0.746

In [47]:

```
#Check for normality
#check to ensure that our residuals are normally distributed
fig = sm.graphics.qqplot(model5.resid, dist=stats.norm, line='45', fit=
plt.xlabel('Theoretical quantiles', fontsize=16)
plt.ylabel('Sample quantiles', fontsize=16)
plt.title('Sample vs. Theoretical quantiles', fontsize=16)
plt.xticks(fontsize=15)
plt.yticks(fontsize=14)
plt.show()
```

In [51]:

```
#Checking for Homoscedasticity
# check that your errors do not display heteroscedasticity; if the erro
#then the model does not meet the initial assumptions.

plt.scatter(model5.predict(z_filtered4), model5.resid, alpha=0.5)
plt.plot(model5.predict(z_filtered4), [0 for i in range(len(z_filtered4
plt.gca().set(xlabel='Fitted', ylabel='Residuals',
              title='Scatter Plot of the Residuals vs. Fitted model ')
plt.xticks(fontsize=15)
plt.yticks(fontsize=14)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-7b64b974e536> in <module>
      3 #then the model does not meet the initial assumptions.
      4
----> 5 plt.scatter(model5.predict(z_filtered4), model5.resid, alpha=0.5)
      6 plt.plot(model5.predict(z_filtered4), [0 for i in range(len(z_filte
red4))])
      7 plt.gca().set(xlabel='Fitted', ylabel='Residuals',

~\anaconda3\lib\site-packages\statsmodels\base\model.py in predict(self, ex
og, transform, *args, **kwargs)
   1098
   1099         predict_results = self.model.predict(self.params, exog, *ar
gs,
-> 1100                                              **kwargs)
   1101
   1102         if exog_index is not None and not hasattr(predict_results,

~\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in pre
dict(self, params, exog)
    378             exog = self.exog
    379
--> 380         return np.dot(exog, params)
    381
    382     def get_distribution(self, params, scale, exog=None, dist_class
=None):

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (21380,98) and (99,) not aligned: 98 (dim 1) != 99 (dim
 0)
```

Comments

Our model after 4 refinements have r-squared of 0.799. All features have p-value less than 0.05. This model is good to predict the price of the property. Important features in the model: yr_built, sqft_living15, floors, bedrooms, bathrooms, view, grade, zipcode. There are selected features that can increase the price and others decresse the price. Features with high positive coefficients:

sqftliving 15, grade 13, view 4, bathrooms 5.5, and all zipcodes(particularly 98112, 98109, 98040, 98004). These features can increase the price of the house. Note that these features were selected in the model used before with feature selection usinh Ski-learn.

Train, test split

The train-test split is a technique for evaluating the performance of a machine learning algorithm. I will approach modelling the data by splitting it into train and test and working only on train data. After modelling refinments, I will examine the refined model on the test data to predict the price.

In [116]:

```
#train, test split
X =preprocessed.drop('price_log', axis=1)
y =preprocessed['price_log']
X_train,X_test,y_train, y_test= train_test_split(X,y, test_size=0.2)
```

In [117]:

```
X_train.head()
```

Out[117]:

	sqft_living_log	sqft_lot_log	sqft_above_log	yr_built_log	sqft_living15_log	sqft_lot15_l
19153	-0.276573	0.383351	0.095264	-0.128115	-0.187309	0.5557
6581	0.373102	-1.083903	-0.127655	1.053592	0.503639	-1.0510
19247	2.064836	0.868989	0.759578	-0.401171	1.091555	0.4059
18297	-1.210778	-2.376869	-1.288762	1.453998	-1.478192	-2.1733
18025	0.261025	0.072324	0.627128	0.751710	0.403515	0.1232

5 rows × 134 columns

In [118]:

```
# Fitting the actual model; baseline model

X_int = sm.add_constant(X_train)
model = sm.OLS(y_train,X_int).fit()
model.summary()
```

Out[118]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.874
Method:	Least Squares	**F-statistic:**	903.4
Date:	Tue, 09 Mar 2021	**Prob (F-statistic):**	0.00
Time:	12:11:52	**Log-Likelihood:**	-6429.9
No. Observations:	17104	**AIC:**	1.313e+04
Df Residuals:	16971	**BIC:**	1.416e+04
Df Model:	132		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.7375	0.198	-8.791	0.000	-2.125	-1.350

Train data model refinement 1

In [119]:

```
# Extract the p-value table from the summary with p-value < 0.05 and us
summary = model.summary()
p_table = summary.tables[1]
p_table = pd.DataFrame(p_table.data)
p_table.columns = p_table.iloc[0]
p_table = p_table.drop(0)
p_table = p_table.set_index(p_table.columns[0])
p_table['P>|t|'] = p_table['P>|t|'].astype(float)
x_cols = list(p_table[p_table['P>|t|'] < 0.05].index)
x_cols.remove('const')
print(len(p_table), len(x_cols))
print(x_cols[:5])
p_table.head()
```

```
135 118
['sqft_living_log', 'sqft_lot_log', 'sqft_above_log', 'yr_built_log', 'sqft
_living15_log']
```

Out[119]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.7375	0.198	-8.791	0.000	-2.125	-1.350
sqft_living_log	0.2417	0.008	31.144	0.000	0.226	0.257
sqft_lot_log	0.1220	0.007	16.792	0.000	0.108	0.136
sqft_above_log	0.1169	0.007	15.710	0.000	0.102	0.131
yr_built_log	-0.0158	0.005	-3.163	0.002	-0.026	-0.006

In [122]:

```
x_cols
```

Out[122]:

```
['sqft_living_log',
 'sqft_lot_log',
 'sqft_above_log',
 'yr_built_log',
 'sqft_living15_log',
 'sqft_lot15_log',
 'floors_2.0',
 'floors_3.0',
 'condition_2',
 'condition_3',
 'condition_4',
 'condition_5',
 'bedrooms_2',
 'bedrooms_3',
 'bedrooms_4',
 'bedrooms_5',
 'bedrooms_6',
 'bedrooms_7',
 'bedrooms_8',
```

In [123]:
```
x_cols=X_train[x_cols]
```

In [124]:
```

X_int = sm.add_constant(x_cols)
model = sm.OLS(y_train,X_int).fit()
model.summary()
```

Out[124]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.874
Method:	Least Squares	**F-statistic:**	1009.
Date:	Tue, 09 Mar 2021	**Prob (F-statistic):**	0.00
Time:	12:13:35	**Log-Likelihood:**	-6444.9
No. Observations:	17104	**AIC:**	1.313e+04
Df Residuals:	16985	**BIC:**	1.405e+04
Df Model:	118		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.5082	0.119	-12.709	0.000	-1.741	-1.276

Train data model refinement 2

In [125]:

```
#Select features with p-value < 0.05.
summary = model.summary()
p_table = summary.tables[1]
p_table = pd.DataFrame(p_table.data)
p_table.columns = p_table.iloc[0]
p_table = p_table.drop(0)
p_table = p_table.set_index(p_table.columns[0])
p_table['P>|t|'] = p_table['P>|t|'].astype(float)
x_cols = list(p_table[p_table['P>|t|'] < 0.05].index)
x_cols.remove('const')
print(len(p_table), len(x_cols))
print(x_cols[:5])
p_table.head()
```

```
119 117
['sqft_living_log', 'sqft_lot_log', 'sqft_above_log', 'yr_built_log', 'sqft
_living15_log']
```

Out[125]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.5082	0.119	-12.709	0.0	-1.741	-1.276
sqft_living_log	0.2402	0.008	31.223	0.0	0.225	0.255
sqft_lot_log	0.1219	0.007	16.814	0.0	0.108	0.136
sqft_above_log	0.1200	0.007	17.056	0.0	0.106	0.134
yr_built_log	-0.0174	0.005	-3.566	0.0	-0.027	-0.008

In [126]:

```
x_cols=X_train[x_cols]

X_int = sm.add_constant(x_cols)
model = sm.OLS(y_train,X_int).fit()
model.summary()
```

Out[126]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.874
Method:	Least Squares	**F-statistic:**	1018.
Date:	Tue, 09 Mar 2021	**Prob (F-statistic):**	0.00
Time:	12:14:21	**Log-Likelihood:**	-6446.4
No. Observations:	17104	**AIC:**	1.313e+04
Df Residuals:	16986	**BIC:**	1.404e+04
Df Model:	117		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.4606	0.115	-12.657	0.000	-1.687	-1.234

Train data model refinement 3

In [127]:

```
#Select features with vif < 5, and input into the model
vif = [variance_inflation_factor(x_cols.values, i) for i in range(x_col
list(zip(x_cols, vif))
```

Out[127]:

```
[('sqft_living_log', 8.03276383219178),
 ('sqft_lot_log', 7.109345665842094),
 ('sqft_above_log', 6.7465344850684374),
 ('yr_built_log', 3.237850433319822),
 ('sqft_living15_log', 3.1740219831049132),
 ('sqft_lot15_log', 6.99679554488095),
 ('floors_2.0', 4.475982494625265),
 ('floors_3.0', 1.699173519341505),
 ('condition_2', 4.294138504906648),
 ('condition_3', 283.46012192113943),
 ('condition_4', 115.5750764221029),
 ('condition_5', 34.58690156935162),
 ('bedrooms_2', 16.42641936363698),
 ('bedrooms_3', 58.77262118932047),
 ('bedrooms_4', 42.67223479176196),
 ('bedrooms_5', 10.681308789785183),
 ('bedrooms_6', 2.718069176069429),
 ('bedrooms_7', 1.2424463855616992),
 ('bedrooms_8', 1.0780304430969176),
```

In [128]:

```
vif_scores = list(zip(x_cols, vif))
x_cols = [x for x,vif in vif_scores if vif < 5]
print(len(vif_scores), len(x_cols))
```

```
117 95
```

In [129]:

```
x_cols=X_train[x_cols]

X_int = sm.add_constant(x_cols)
model = sm.OLS(y_train,X_int).fit()
model.summary()
```

Out[129]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.793
Model:	OLS	**Adj. R-squared:**	0.792
Method:	Least Squares	**F-statistic:**	687.5
Date:	Tue, 09 Mar 2021	**Prob (F-statistic):**	0.00
Time:	12:15:53	**Log-Likelihood:**	-10756.
No. Observations:	17104	**AIC:**	2.170e+04
Df Residuals:	17008	**BIC:**	2.245e+04
Df Model:	95		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.0712	0.013	-84.168	0.000	-1.096	-1.046

In [130]:

```
fig = sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=T
```

In [134]:

```
#Code does nor work , tried hard to fix it. I need help
#plt.scatter(model.predict(x_cols), model.resid)
#plt.plot(model.predict(x_cols), [0 for i in range(len(X_train))])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-134-f094c9f3f0fd> in <module>
----> 1 plt.scatter(model.predict(x_cols), model.resid)
      2 plt.plot(model.predict(x_cols), [0 for i in range(len(x_cols))])

~\anaconda3\lib\site-packages\statsmodels\base\model.py in predict(self, ex
og, transform, *args, **kwargs)
   1098
   1099         predict_results = self.model.predict(self.params, exog, *ar
gs,
-> 1100                                              **kwargs)
   1101
   1102         if exog_index is not None and not hasattr(predict_results,

~\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py in pre
dict(self, params, exog)
    378             exog = self.exog
    379
--> 380         return np.dot(exog, params)
    381
    382     def get_distribution(self, params, scale, exog=None, dist_class
=None):

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (17104,95) and (96,) not aligned: 95 (dim 1) != 96 (dim
 0)
```

Train data model refinement 4

In [135]:

```
#Select features with p-value < 0.05
summary = model.summary()
p_table = summary.tables[1]
p_table = pd.DataFrame(p_table.data)
p_table.columns = p_table.iloc[0]
p_table = p_table.drop(0)
p_table = p_table.set_index(p_table.columns[0])
p_table['P>|t|'] = p_table['P>|t|'].astype(float)
x_cols = list(p_table[p_table['P>|t|'] < 0.05].index)
x_cols.remove('const')
print(len(p_table), len(x_cols))
print(x_cols[:5])
p_table.head()
```

```
96 90
['yr_built_log', 'sqft_living15_log', 'floors_2.0', 'floors_3.0', 'conditio
n_2']
```

Out[135]:

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.0712	0.013	-84.168	0.000	-1.096	-1.046
yr_built_log	-0.0291	0.005	-5.308	0.000	-0.040	-0.018
sqft_living15_log	0.3597	0.005	71.602	0.000	0.350	0.370
floors_2.0	0.1742	0.010	18.161	0.000	0.155	0.193
floors_3.0	-0.0636	0.025	-2.526	0.012	-0.113	-0.014

In [136]:

```
x_cols=X_train[x_cols]

X_int = sm.add_constant(x_cols)
model = sm.OLS(y_train,X_int).fit()
model.summary()
```

Out[136]:

OLS Regression Results

Dep. Variable:	price_log	**R-squared:**	0.793
Model:	OLS	**Adj. R-squared:**	0.792
Method:	Least Squares	**F-statistic:**	725.7
Date:	Tue, 09 Mar 2021	**Prob (F-statistic):**	0.00
Time:	12:17:42	**Log-Likelihood:**	-10758.
No. Observations:	17104	**AIC:**	2.170e+04
Df Residuals:	17013	**BIC:**	2.240e+04
Df Model:	90		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.0639	0.012	-88.513	0.000	-1.087	-1.040

In [137]:

```
#Check and make sure all features have vif < 5
vif = [variance_inflation_factor(x_cols.values, i) for i in range(x_col
list(zip(x_cols, vif))
```

Out[137]:

```
[('yr_built_log', 2.4833275063287523),
 ('sqft_living15_log', 2.045359133449084),
 ('floors_2.0', 2.730906714124321),
 ('floors_3.0', 1.5066678551363992),
 ('condition_2', 1.0317677136329875),
 ('bedrooms_6', 1.0674263544290077),
 ('view_1.0', 1.0659287824073067),
 ('view_2.0', 1.1374476626881367),
 ('view_3.0', 1.1174177935776861),
 ('view_4.0', 1.441920863078984),
 ('waterfront_1.0', 1.4384841927316847),
 ('bathrooms_0.75', 1.12512412914068),
 ('bathrooms_3.75', 1.050911569439537),
 ('bathrooms_4.0', 1.047737314213251),
 ('bathrooms_4.25', 1.0347927046685437),
 ('bathrooms_4.5', 1.058571194008421),
 ('bathrooms_4.75', 1.0109726937125496),
 ('bathrooms_5.0', 1.0272058228223178),
 ('bathrooms_5.25', 1.0167651402300861),
 ('bathrooms_5.5', 1.0120836937790199),
 ('grade_4', 1.115545884662136),
 ('grade_5', 1.0977007400776424),
 ('grade_6', 1.441662269464835),
 ('grade_8', 1.6167602586323844),
 ('grade_10', 1.3671615358384757),
 ('grade_11', 1.2126071786567973),
 ('grade_12', 1.0698522916097866),
 ('zipcode_98004', 1.039894043704299),
 ('zipcode_98005', 1.0221601790607715),
 ('zipcode_98006', 1.0844518810397987),
 ('zipcode_98007', 1.0117991380031905),
 ('zipcode_98008', 1.0190200453474452),
 ('zipcode_98010', 1.0101914895146515),
 ('zipcode_98011', 1.0164006797451008),
 ('zipcode_98014', 1.021632464613699),
 ('zipcode_98019', 1.0292504915225036),
 ('zipcode_98022', 1.0243381393138247),
 ('zipcode_98024', 1.0113471277984878),
 ('zipcode_98027', 1.0521819407159907),
 ('zipcode_98028', 1.0213856717387744),
 ('zipcode_98029', 1.07514356900405),
 ('zipcode_98031', 1.0161972578626537),
 ('zipcode_98033', 1.0428820030187516),
 ('zipcode_98034', 1.0181608831815279),
 ('zipcode_98038', 1.083786327916407),
 ('zipcode_98039', 1.0087526898918477),
 ('zipcode_98040', 1.0584858633691856),
 ('zipcode_98042', 1.0452331343220054),
 ('zipcode_98045', 1.022163454825985),
 ('zipcode_98052', 1.0610863543377929),
 ('zipcode_98053', 1.0587987854540961),
```

```
('zipcode_98055', 1.0261143985376249),
('zipcode_98056', 1.0428099066026808),
('zipcode_98058', 1.0345075451040238),
('zipcode_98059', 1.0563196185768378),
('zipcode_98065', 1.0662356096589214),
('zipcode_98070', 1.0755369384873672),
('zipcode_98072', 1.0232211917949618),
('zipcode_98074', 1.071264151428341),
('zipcode_98075', 1.1107244257997295),
('zipcode_98077', 1.0476350043594016),
('zipcode_98102', 1.0381657210629407),
('zipcode_98103', 1.2428242161504484),
('zipcode_98105', 1.061926264194351),
('zipcode_98106', 1.0707621364685016),
('zipcode_98107', 1.1020891048606258),
('zipcode_98108', 1.0303740324977368),
('zipcode_98109', 1.0449356128764693),
('zipcode_98112', 1.0832358523820729),
('zipcode_98115', 1.106791626356546),
('zipcode_98116', 1.0834863551028515),
('zipcode_98117', 1.1300656540664447),
('zipcode_98118', 1.1012751813891932),
('zipcode_98119', 1.0692044150833826),
('zipcode_98122', 1.0970202801344864),
('zipcode_98125', 1.0501955603487876),
('zipcode_98126', 1.1014054914578417),
('zipcode_98133', 1.0579557627338805),
('zipcode_98136', 1.067108836766147),
('zipcode_98144', 1.0737682167000058),
('zipcode_98146', 1.0619836672631362),
('zipcode_98148', 1.0051840900608757),
('zipcode_98155', 1.0361417858278148),
('zipcode_98166', 1.0406704123321402),
('zipcode_98168', 1.057731831978476),
('zipcode_98177', 1.0399440443368635),
('zipcode_98178', 1.0448599800602643),
('zipcode_98188', 1.0098851125103088),
('zipcode_98198', 1.033062940665175),
('zipcode_98199', 1.058360549792428)]
```

Check the residuals of our refined model

In [138]:
```
fig = sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=T
```

Comments

Ater modelling on the train data and refinements. The r-squared is 0.793. Important features: yr_built, sqft_living15, bathrooms, view, waterfront, bedroom (only one, i.e. bedrooms 6), condition (only one; condition 2), zipcode, grade. Again there are features could decrease the price like yr_built and other increase like waterfront.

Ski-learn model to predict on test data

In [140]:
```
# Use Ski-learn to fit the train  data based on feature of train data m

modelfinal=linreg.fit(x_cols, y_train)


```

In [141]:
```
y_hat_train = modelfinal.predict(x_cols)
train_residuals = y_hat_train - y_train

```

Train Mean Squarred Error

In [142]:
```
train_mse = mean_squared_error(y_train, y_hat_train)
print('Train Mean Squarred Error:', train_mse)
```

```
Train Mean Squarred Error: 0.20599535009660247
```

In [ ]:
```
r2_score(y_train, y_hat_train)
```

K-fold cross validation

In [144]:
```

# train {K} linear regression models on the data, with each linear mode
#and all other sections combined as the training set.
# then average the individual results frome each of these linear models
#perform 5-fold cross-validation to get the mean squared error through

from sklearn.metrics import mean_squared_error, make_scorer

mse = make_scorer(mean_squared_error)


cv_5_results = cross_val_score(modelfinal, x_cols, y_train, cv=5, scori

cv_5_results.mean()

```

Out[144]:
```
0.2091602906290056
```

In [145]:
```
cv_5_results.std()
```

Out[145]:
```
0.009885110913921077
```

In [148]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(modelfinal, x_cols, y_train)

scores.mean(), scores.std()
```

Out[148]: (0.7901129123274145, 0.009001678578128986)

In [69]:
```
#vertical distance between the points and the line denote the errors
plt.figure(figsize=(8, 5))
plt.scatter(y_train, y_hat_train, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

Comments

The score mean after cross validation is 0.79 with standard deviation of 0.01. The model score is good that we can start predicting on the test data.

Problem with test data

I cant not examine and predict through my testing data in this model. Tried to figure it out, but it took me longer than expected. This is why I will try modelling with a diffrent approach

In [440]:
```
df.columns
```

Out[440]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lon
g',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

Train data modelling using Ski-learn

Befoe modelling, I will split he data into train and test. Model using only the train data. Test the model on test data. For model refinement, I will select features with hight coefficents that can negatively or positively influence the price. Also select features which performed well in other refined models (see above).

In [459]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
def filter_columns(df):
    '''Return DataFrame with only certain columns
    '''
    columns_to_keep = ['price',
        'bedrooms',
        'bathrooms',
        'sqft_living',
        'sqft_lot',
        'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abc
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', '
       'sqft_living15', 'sqft_lot15'
    ]

    return df[columns_to_keep]

```

In [460]:
```
df_filtered = filter_columns(df)
df_filtered.head()
```

Out[460]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0.0	0.0	3	
1	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	3	
2	180000.0	2	1.00	770	10000	1.0	0.0	0.0	3	
3	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	5	
4	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	3	

In [201]:
```
df_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21380 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21380 non-null  float64
 1   bedrooms       21380 non-null  category
 2   bathrooms      21380 non-null  category
 3   sqft_living    21380 non-null  int64  
 4   sqft_lot       21380 non-null  int64  
 5   floors         21380 non-null  category
 6   waterfront     21380 non-null  category
 7   view           21380 non-null  category
 8   condition      21380 non-null  category
 9   grade          21380 non-null  category
 10  sqft_above     21380 non-null  int64  
 11  sqft_basement  21380 non-null  float64
 12  yr_built       21380 non-null  int64  
 13  yr_renovated   21380 non-null  float64
 14  zipcode        21380 non-null  category
 15  lat            21380 non-null  float64
 16  long           21380 non-null  float64
 17  sqft_living15  21380 non-null  int64  
 18  sqft_lot15     21380 non-null  int64  
dtypes: category(8), float64(5), int64(6)
memory usage: 2.1 MB
```

In [461]:
```
target = df_filtered['price']
features = df_filtered.drop('price', axis=1)
```

In [196]:
```
features.head()
```

Out[196]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft
0	3	1.00	1180	5650	1.0	0.0	0.0	3	7	
1	3	2.25	2570	7242	2.0	0.0	0.0	3	7	
2	2	1.00	770	10000	1.0	0.0	0.0	3	6	
3	4	3.00	1960	5000	1.0	0.0	0.0	5	7	
4	3	2.00	1680	8080	1.0	0.0	0.0	3	8	

In [462]:
```
#Split the data into training and test sets. Set the seed to 42 and the
X_train , X_test, y_train, y_test = train_test_split(features, target,
```

In [463]:
```
# Transform with MinMaxScaler, I used the MinMaxScaler to scale the tra
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

In [464]:
```
#Transform the test data (X_test) using the same scaler:
X_test_scaled = scaler.transform(X_test)
```

Model 1

In [465]:
```
#Fit a regression model to the training data
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[465]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [466]:
```
linreg.coef_
```

Out[466]:
```
array([-177534.16001614,  212546.62793383,  456457.40389517,
        353445.31345925,   70007.61378   ,  304679.74801362,
        187564.84208087,  114947.86171879,  819454.68134789,
        325533.1706828 ,  116172.06258861, -261647.03175563,
         49463.90321689,  -89655.58395639,  360854.40707749,
       -185063.18602069,  261758.41506393, -237703.04748898])
```

In [467]:
```
linreg.intercept_
```

Out[467]:
```
-251320.09667274018
```

In [483]:
```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[483]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	-177534.160016	212546.627934	456457.403895	353445.313459	70007.61378	304679.748014

Model to make predictions on both the training and test sets:

In [468]:

```
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
```

In [469]:

```
# vertical distance between the points and the line denote the errors
#Plot predictions for the training set against the actual data:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [470]:
```
#Plot predictions for the test set against the actual data:

plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```

In [471]:
```
# Make a prediction with CV
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[471]: (0.7167860536786433, 0.008310682108915357)

In [481]:
```
mean_squared_error(y_train, lm_train_predictions)
r2_score(y_train, lm_train_predictions)
```

Out[481]: 0.7178885833452935

In [207]:
```
#Bias
def bias(y, y_hat):
    return np.mean(y_hat - y)
```

In [210]:
```
#Variance
def variance(y_hat):
    return np.mean([yi**2 for yi in y_hat]) - np.mean(y_hat)**2
```

In [211]:

```
#Calculate bias and variance
# Bias and variance for training set
b = bias(y_train, lm_train_predictions)
v =variance(lm_train_predictions)
print('Train bias: {} \nTrain variance: {}'.format(b, v))
```

```
Train bias: 7.000165469630842e-11
Train variance: 57967332005.25546
```

In [212]:

```
# Bias and variance for test set
b = bias(y_test, lm_test_predictions)
v = variance(lm_test_predictions)
print('Test bias: {} \nTest variance: {}'.format(b, v))
```

```
Test bias: -831.4950545169168
Test variance: 59263212959.64935
```

Model 1 comments

This model fit scores is good of 0.72 with standard deviation of 0.008. The r_squared is 0.72 which indicates a good fit. However, there are too many features and it needs refinement. I will try polynomial regression and make predictions on training and test data.

Polynomial regression

I wil account for interactions between two variables by including interaction effects in my model.

Model 2

In [472]:

```
poly = PolynomialFeatures(3)

X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.fit_transform(X_test_scaled)
```

In [473]:

```
polyreg = LinearRegression()
polyreg.fit(X_train_poly, y_train)
```

Out[473]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [474]:
```
#Use the model to make predictions on both the training and test sets:
# Training set predictions
poly_train_predictions = polyreg.predict(X_train_poly)

# Test set predictions
poly_test_predictions = polyreg.predict(X_test_poly)
```

In [479]:
```
from sklearn.metrics import mean_squared_error, r2_score
mean_squared_error(y_train, poly_train_predictions)
```

Out[479]: 9935161431.902498

In [480]:
```
r2_score(y_train, poly_train_predictions)
```

Out[480]: 0.8769593369415614

In [475]:
```
# Vertical distance between the points and the line denote the errors
plt.figure(figsize=(8, 5))
plt.scatter(y_train, poly_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [476]:

```
#Vertical distance between the points and the line denote the errors
plt.figure(figsize=(8, 5))
plt.scatter(y_test, poly_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```

In [218]:

```
# Bias and variance for training set
b = bias(y_train, poly_train_predictions)
v = variance(poly_train_predictions)
print('Train bias: {} \nTrain variance: {}'.format(b, v))
```

```
Train bias: 1.9515307548672824
Train variance: 70803715282.44638
```

In [219]:

```
# Bias and variance for test set
b = bias(y_test, poly_test_predictions)
v = variance(poly_test_predictions)
print('Test bias: {} \nTest variance: {}'.format(b, v))
```

```
Test bias: -582452183478.1647
Test variance: 1.1483280792142907e+28
```

Model 2 comment

Model 2 comment

After polynomial regression fit with degree 3. The r_squared is even better of 0.87. However, it looks like our model has a problem of overfit due to high number of test bias. I will seek other types of refinments for my model.

Model 3

In [518]:

```
#Model refinement
#Remove all the features that negatively influence the price such as: b
def filter_columns(df):
    '''Return DataFrame with only certain columns
    '''
    columns_to_keep = ['price',

        'bathrooms',
        'sqft_living',
        'sqft_lot',
        'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abc
       'sqft_basement', 'yr_renovated' , 'lat',
       'sqft_living15',
    ]

    return df[columns_to_keep]

```

In [488]:

```
df_filtered = filter_columns(df)
df_filtered.head()
```

Out[488]:

	price	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_a
0	221900.0	1.00	1180	5650	1.0	0.0	0.0	3	7	
1	538000.0	2.25	2570	7242	2.0	0.0	0.0	3	7	
2	180000.0	1.00	770	10000	1.0	0.0	0.0	3	6	
3	604000.0	3.00	1960	5000	1.0	0.0	0.0	5	7	
4	510000.0	2.00	1680	8080	1.0	0.0	0.0	3	8	

In [519]:
```
target = df_filtered['price']
features = df_filtered.drop('price', axis=1)
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[519]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [520]:
```
linreg.coef_
```

Out[520]:
```
array([-76937.46130085, 552139.49658762,  67265.8268193 ,  20918.20079466,
       301883.82210249, 224811.84538003, 215671.47962053, 743876.16456317,
       238041.42306573, 121316.68393492, 130386.17706071, 398452.87605619,
       220415.71360589])
```

In [521]:
```
linreg.intercept_
```

Out[521]:
```
-505039.5746224045
```

In [522]:
```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[522]:

	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	
0	-76937.461301	552139.496588	67265.826819	20918.200795	301883.822102	224811.84538	21

In [492]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[492]:
```
(0.6835296394322727, 0.009648681573611644)
```

In [493]:

```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [243]:
```
#Calculate bias and variance
# Bias and variance for training set
b = bias(y_train, lm_train_predictions)
v =variance(lm_train_predictions)
print('Train bias: {} \nTrain variance: {}'.format(b, v))
```

```
Train bias: 7.058972086830603e-11
Train variance: 55280650225.89365
```

In [494]:
```
mean_squared_error(y_train, lm_train_predictions)
```

Out[494]: 25466325488.824196

In [495]:
```
r2_score(y_train, lm_train_predictions)
```

Out[495]: 0.6846157362128611

Model 4

In [523]:
```
#Model refinement for only features that increase the value of the hous

def filter_columns(df):
    '''Return DataFrame with only certain columns
    '''
    columns_to_keep = ['price',


        'sqft_living',
        'sqft_lot',
        'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abc
       'sqft_basement', 'yr_renovated' , 'lat',
       'sqft_living15',
    ]

    return df[columns_to_keep]
```

In [524]:
```
df_filtered = filter_columns(df)
target = df_filtered['price']
features = df_filtered.drop('price', axis=1)
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[524]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [525]:
```
linreg.coef_
```

Out[525]:
```
array([498745.63660044,  74348.21530439,   9561.89761544, 302319.99010399,
       226242.40149451, 216634.66687809, 734390.75131896, 255928.50584803,
       121576.44753121, 129528.12052171, 400323.48181672, 221558.48235302])
```

In [247]:
```
linreg.intercept_
```

Out[247]:
```
-509158.97714942484
```

In [526]:
```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[526]:

	sqft_living	sqft_lot	floors	waterfront	view	condition	
0	498745.6366	74348.215304	9561.897615	302319.990104	226242.401495	216634.666878	734

In [248]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[248]:
```
(0.6833272523288318, 0.009637338452933634)
```

In [249]:

```
features.corr()
```

Out[249]:

	sqft_living	sqft_lot	sqft_above	sqft_basement	yr_renovated	lat	sqf
sqft_living	1.000000	0.175440	0.867559	0.399329	0.038531	0.041677	
sqft_lot	0.175440	1.000000	0.184498	0.011542	0.004892	-0.087774	
sqft_above	0.867559	0.184498	1.000000	-0.101192	0.010494	-0.013061	
sqft_basement	0.399329	0.011542	-0.101192	1.000000	0.057147	0.106767	
yr_renovated	0.038531	0.004892	0.010494	0.057147	1.000000	0.026236	
lat	0.041677	-0.087774	-0.013061	0.106767	0.026236	1.000000	
sqft_living15	0.754853	0.146153	0.727922	0.170557	-0.011813	0.041585	

In [268]:

```
#Check for multicollinearity
db=features.corr().abs().stack().reset_index().sort_values(0, ascending

# zip the variable name columns (Which were only named level_0 and leve
db['pairs'] = list(zip(db.level_0, db.level_1))

# set index to pairs
db.set_index(['pairs'], inplace = True)

#d rop level columns
db.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
db.columns = ['cc']

# drop duplicates. This could be dangerous if you have variables perfec
# for the sake of exercise, kept it in.
db.drop_duplicates(inplace=True)
```

In [269]:

```
db[(df.cc>.75) & (db.cc <1)]
```

Out[269]:

	cc
pairs	
(sqft_above, sqft_living)	0.867559
(sqft_living15, sqft_living)	0.754853

In [ ]:
```
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
```

Model 5

In [527]:
```
#Drop  sqft_living
#Model refinement
columns_to_keep = [ 'floors', "sqft_lot", 'waterfront', 'view', 'condit
       'sqft_basement', 'yr_renovated' , 'lat', "sqft_above","sqft_livi




features=df[columns_to_keep]
target=df['price']
```

In [264]:
```
df.columns
```

Out[264]:
```
Index(['cc'], dtype='object')
```

In [265]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 22 entries, ('sqft_living', 'sqft_living') to ('yr_renovated', 'sqft
_lot')
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   cc      22 non-null     float64
dtypes: float64(1)
memory usage: 352.0+ bytes
```

In [528]:
```

X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[528]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [529]:
```
linreg.coef_
```

Out[529]:
```
array([  8221.2842681 ,  75148.24663041, 301026.08041798, 227761.68495954,
       218058.23053148, 738202.96806059, 338000.47891605, 129799.18325082,
       400579.61534675, 728843.52650362, 223935.79200731])
```

In [530]:
```
linreg.intercept_
```

Out[530]:
```
-510891.9076626918
```

In [531]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[531]:
```
(0.6830013687202937, 0.009689226298935578)
```

In [532]:
```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [533]:
```
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
#adjusted_r_squared = 1 - (1-r_squared)*(len(y)-1)/(len(y)-X[selected_c
```

In [534]:
```
r_squared
```

Out[534]: 0.9287236381555507

In [535]:
```

linreg.coef_.reshape(-1,1)

pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns

```

Out[535]:

	floors	sqft_lot	waterfront	view	condition	grade	sqft
0	8221.284268	75148.24663	301026.080418	227761.68496	218058.230531	738202.968061	338

Model 5 comments

This model fit is excellent with r-squared of 0.92,fit score of 0.68, standard deviation of 0.009. All features have a high coefficients which significantly influnce the price i.e. increase it.

Model 6

In [506]:
```
#Keep the most powerful features
#Model refinement
columns_to_keep = [  'waterfront', 'view', 'condition', 'grade',
       'sqft_basement', "sqft_above", 'lat']





features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[506]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [507]:
```
display(linreg.coef_)
display(linreg.intercept_)
```

```
array([311813.44413174, 251029.46718552, 205712.5266061 , 786336.49229933,
       383064.51355757, 881267.30086629, 403107.37742595])

-493095.1026380424
```

In [508]:
```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[508]:

	waterfront	view	condition	grade	sqft_basement	sqft_above
0	311813.444132	251029.467186	205712.526606	786336.492299	383064.513558	881267.300866

In [509]:
```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
```

In [510]:
```
r_squared
```

Out[510]:
```
0.9266310571054966
```

In [511]:

```

scores = cross_val_score( linreg, X_train, y_train, cv=10, scoring="neg


rmse_scores = np.sqrt(-scores)
```

In [512]:

```

display(rmse_scores)
display(rmse_scores.mean())
display(rmse_scores.std())
```

```
array([169021.99084308, 158022.6097752 , 163341.3392364 , 166531.42714475,
       164657.47632172, 161048.11813711, 167396.54163493, 167391.80745401,
       155088.63958919, 149050.97732074])

162155.0927457144

6073.113480861099
```

In [513]:

```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[513]:

```
(0.6740023939316101, 0.010130824764803283)
```

In [514]:

```
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [515]:
```
# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
```

In [516]:
```
resid = lm_test_predictions - y_test
resid
```

Out[516]:
```
10391    -44364.238949
12559    114856.049878
7540       7485.286483
10860     -1909.987753
15427      2233.592748
             ...
3974     160634.408667
10220     23989.587608
945      -14935.720684
9906    -240347.895120
18760    117099.176376
Name: price, Length: 5345, dtype: float64
```

In [517]:
```
resid_train = lm_train_predictions - y_train
resid_train
```

Out[517]:
```
11447    -20649.571921
7294     167985.100066
5566     142613.254815
9818      15740.832435
2032      15514.832121
             ...
11387    -23009.977172
12074     74244.140968
5448     210367.241958
872      116964.422000
15945    161539.403052
Name: price, Length: 16035, dtype: float64
```

Model 6 comments

It is an excellent model with good r-squared of 0.92, score fit 0.67 and standard deviation of 0.01.

Model 7

In [536]:

```
#Select 2 features only, with the higest coefficients
#Model refinement
columns_to_keep = [
        'grade', 'sqft_above'
       ]









features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[536]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [537]:

```
display(linreg.coef_)
display(linreg.intercept_)
```

```
array([1243322.25862534,  439941.07064347])

-206917.28384848905
```

In [538]:

```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[538]:

	grade	sqft_above
0	1.243322e+06	439941.070643

In [539]:

```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[539]:

```
0.8813990616630507
```

In [542]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[542]:
```
(0.47394975409498324, 0.007267013589548912)
```

In [543]:
```
# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
```

In [544]:
```
resid = lm_test_predictions - y_test
resid
```

Out[544]:
```
10391     38532.077323
12559     38807.929152
7540      67520.120768
10860     50021.125245
15427   -193693.744978
             ...
3974     418819.550881
10220    195059.101042
945      138788.271602
9906    -252086.477279
18760   -232945.586520
Name: price, Length: 5345, dtype: float64
```

Model 7 comment

This is a good model with 2 features grade and sqft_above with r_squared of 0.88 and fit score of 0.47.

Model 8 for home interior

In [320]:

```
#Model for the home interior. Does home interior features affect the pr
#Model refinement
columns_to_keep = [ 'bedrooms',

        'bathrooms',
        'sqft_living',
        'sqft_lot',
        'floors',  'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_renovated' ,
       'sqft_living15']









features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[320]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [321]:

```
display(linreg.coef_)
display(linreg.intercept_)
```

```
array([-216319.76299238, -173222.22197325,  976721.57725119,
       -186106.03920302,   70322.57841228,  228589.67966804,
        914296.63954696, -210548.71081039,  134551.68741319,
        169580.13462549,  297090.26808955])

-294873.6692269234
```

In [322]:

```
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

Out[322]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	conditior
0	-216319.762992	-173222.221973	976721.577251	-186106.039203	70322.578412	228589.67966

In [323]:
```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[323]: 0.6787749871964879

In [324]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[324]: (0.5686331279480826, 0.0074397781820215525)

Model 9 for home interior

In [325]:
```
#Model refinement
columns_to_keep = [
        'sqft_living',

        'floors',  'condition', 'grade',
       'sqft_basement', 'yr_renovated' ,
       'sqft_living15']

features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[325]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [326]:
```
display(linreg.coef_)
display(linreg.intercept_)
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([544821.96935612,  44536.42516853, 226998.26508233, 922949.04599301,
       214566.17245625, 169873.52689259, 307175.56762895])

-338709.85249458184
```

Out[326]:

	sqft_living	floors	condition	grade	sqft_basement	yr_renovated
0	544821.969356	44536.425169	226998.265082	922949.045993	214566.172456	169873.526893

In [327]:
```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[327]: 0.674750537233293

In [328]:
```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[328]: (0.56356634663979, 0.007649311877272741)

In [329]:
```
features.corr()
```

Out[329]:

	sqft_living	sqft_basement	yr_renovated	sqft_living15
sqft_living	1.000000	0.399329	0.038531	0.754853
sqft_basement	0.399329	1.000000	0.057147	0.170557
yr_renovated	0.038531	0.057147	1.000000	-0.011813
sqft_living15	0.754853	0.170557	-0.011813	1.000000

Model 10 for interior

In [548]:

```
#Model refinement, keep 2 home interior features with the highest coeff

columns_to_keep = [
        'sqft_living',

          'grade',

      ]

features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[548]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [549]:

```
display(linreg.coef_)
display(linreg.intercept_)
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([937246.96873716, 883375.60203995])
```

```
-148422.63626300823
```

Out[549]:

	sqft_living	grade
0	937246.968737	883375.60204

In [550]:

```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[550]:

```
0.8932371674088216
```

In [551]:

```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[551]:

```
(0.5262447018503691, 0.005727569627331013)
```

In [552]:

```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [553]:

```
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```

In [554]:
```
resid_train = lm_train_predictions - y_train
resid_train
```

Out[554]:
```
11447    -57987.503981
7294     332433.594515
5566      40255.668695
9818     123584.485404
2032      56670.129436
             ...
11387    -69905.557964
12074    -10489.352989
5448      78670.129436
872     -128766.263167
15945     79571.209895
Name: price, Length: 16035, dtype: float64
```

In [555]:
```
resid_test = lm_test_predictions - y_test
resid_test
```

Out[555]:
```
10391     40440.127495
12559     69623.116426
7540      32042.535505
10860      6740.544178
15427   -157524.560360
             ...
3974     390158.428571
10220    117307.849591
945      166537.472393
9906    -290046.322631
18760    -89383.795693
Name: price, Length: 5345, dtype: float64
```

In [556]:
```
train_mse = mean_squared_error(y_train, lm_train_predictions)
test_mse = mean_squared_error(y_test,lm_test_predictions)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)

```

```
Train Mean Squarred Error: 38223288503.34336
Test Mean Squarred Error: 38288073770.42306
```

In [ ]:
```
Model 10 for home interior comments
This is an excellent model with 2 features grade and sqft_living with r
```

Model 11 for exterior geographic features

In [566]:

```
#Model refinement
columns_to_keep =['sqft_lot',



         'waterfront', 'view',
        'lat', 'long', 'zipcode'

    ]

features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[566]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [567]:

```
display(linreg.coef_)
display(linreg.intercept_)
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([ 892239.97404054,  203911.08433423,  540445.76370594,
        533075.79964048,   11389.73447875, -194062.59306779])

206692.99672719132
```

Out[567]:

	sqft_lot	waterfront	view	lat	long	zipcode
0	892239.974041	203911.084334	540445.763706	533075.79964	11389.734479	-194062.593068

In [568]:

```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[568]:

```
0.8411489570591084
```

In [569]:

```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[569]:

```
(0.295163468428391, 0.016983628134734672)
```

In [562]:

```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [563]:

```
resid_train = lm_train_predictions - y_train
resid_train
```

Out[563]:

```
11447     23192.793598
7294     -39178.975689
5566     264083.681661
9818      70815.683867
2032     -68056.422751
              ...
11387    -43346.335983
12074    252059.312413
5448     187024.562580
872       97450.964659
15945    354980.290757
Name: price, Length: 16035, dtype: float64
```

In [564]:
```
resid_test = lm_test_predictions - y_test
resid_test
```

Out[564]:
```
10391    -86468.804198
12559    -22707.130425
7540     274822.948077
10860    284618.212930
15427    135758.122770
             ...
3974     -50494.650215
10220    100982.519587
945       77492.067783
9906    -103524.489178
18760     35427.477571
Name: price, Length: 5345, dtype: float64
```

In [565]:
```
train_mse = mean_squared_error(y_train, lm_train_predictions)
test_mse = mean_squared_error(y_test,lm_test_predictions)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)

```

```
Train Mean Squarred Error: 56871938445.44349
Test Mean Squarred Error: 58552332164.204056
```

Model 11 for home exterior geographic features comments

This model fit did not perform well due to low fit score of 0.29. It is no the best model to work with. The most important geographic features that can influence the price sale: zipcode, view and sqft_lot.

Model 12

Select the best performing features from all above models (i.e.geographic models, interior features models).

In [604]:

```
#Model refinement
columns_to_keep =['sqft_living',  "view",


        "grade", "lat", "long"

    ]

features=df[columns_to_keep]
target=df['price']
X_train , X_test, y_train, y_test = train_test_split(features, target,
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

Out[604]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fa
lse)
```

In [605]:

```
display(linreg.coef_)
display(linreg.intercept_)
pd.DataFrame(data=linreg.coef_.reshape(1,-1), columns= features.columns
```

```
array([ 973587.25925568,  281858.92019982,  741057.4463518 ,
        384606.77339154, -193580.35465052])

-300558.78247597837
```

Out[605]:

	sqft_living	view	grade	lat	long
0	973587.259256	281858.9202	741057.446352	384606.773392	-193580.354651

In [606]:

```
lm_train_predictions = linreg.predict(X_train_scaled)
SS_Residual = np.sum((y_train-lm_train_predictions )**2)
SS_Total = np.sum((y-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
r_squared
```

Out[606]:

```
0.9239086051530769
```

In [607]:

```
scores = cross_val_score(linreg, X_train, y_train)

scores.mean(), scores.std()
```

Out[607]:

```
(0.6621766373583389, 0.008463757597811452)
```

In [608]:

```
#model to make predictions on both the training and test sets:
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```

In [611]:

```
# vertical distance between the points and the line denote the errors
#model to make predictions on both the training and test sets:
plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```

In [612]:
```
resid_train = lm_train_predictions - y_train
resid_train
resid_test = lm_test_predictions - y_test
resid_test
```

Out[612]:
```
10391    -11142.368450
12559     31865.791186
7540      47030.155479
10860     38141.131330
15427     83107.381723
             ...
3974     206190.606928
10220    -36184.363599
945       39534.438915
9906    -213952.657782
18760    137203.134093
Name: price, Length: 5345, dtype: float64
```

In [613]:
```
train_mse = mean_squared_error(y_train, lm_train_predictions)
test_mse = mean_squared_error(y_test,lm_test_predictions)
print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 27242283360.84888
Test Mean Squarred Error: 27971006598.201912
```

Model 12 comments

This is an excellent model with r_squared of 0.92 and fit score of 0.66 and standard deviation of 0.008. Most important features to predict sale price: sqft_living, grade, location and view.

Data interpretation

Sale price prediction

The most important two features to predict the price sale are: square feet of living space of the house (sqft_living), grade (how good and well maintained is the house condition). Other impotant features: location (lat, long), view (does the home have a geographic view), condition of the house, and the number of floors.

Visualization for the house square footage

In [657]:

```
new_figure, (ax1, ax2) = plt.subplots(figsize=(12,6), ncols=2)
new_figure.set_tight_layout(True)

ax1.scatter(x, y, color='black', linewidth=0.5, linestyle = ':')

ax2.scatter(z,y, color='green', linewidth=0.5, linestyle = '-.')

ax1.set_xlabel('Square footage of the home')
ax1.set_ylabel('Price ($)')
ax1.set_title ('Sale price vs house square footage')

ax2.set_xlabel('Square footage of the house apart from basement')
ax2.set_ylabel('Price ($)')
ax2.set_title ('Price sale vs square footage of the house excluding bas
```

Visualization for the house grade

In [721]:

```
fig=plt.figure(figsize=(12,8))
sns.swarmplot(x='grade', y='price', data=df);
plt.title('Grade vs. Price of the houses sold', fontsize=15)
plt.xlabel("Grade", fontsize=15)
plt.ylabel('Sale Price ($)', fontsize=15)
```

Out[721]: Text(0, 0.5, 'Sale Price ($)')

Important geographic features vs price sale.

The most two important features are square foot of the whole house lot and view (if the home has a beautiful landscape view). By descending order: square foot of the whole house lot, view (if the home has a beautiful landscape view),location, and waterfront (if the home is facing a lake, river..ect).

Visualization for the square footage of the house

In [689]:
```
sns.lmplot(x="sqft_living", y="price",data=df,)
sns.lmplot(x="sqft_above", y="price", data=df)
```

Out[689]: <seaborn.axisgrid.FacetGrid at 0x1f52b64b4c8>

Visualization for the location of the house

In [688]:

```
plt.figure(figsize=(18,10))
plt.scatter(df['long'], df['lat'], c=df['price']/df['sqft_living'], cma
cbar = plt.colorbar()
cbar.set_label('Price per square footage')
plt.grid(which='both')
plt.title('Price sold per square footage based on location')
plt.show
```

Out[688]: <function matplotlib.pyplot.show(*args, **kw)>

Visualization for house grade

In [11]:

```
fig, ax = plt.subplots(figsize=(12,4))
sns.boxplot(y = 'view',
            x = 'price',
            data = df,
            width = 0.8,
            orient = 'h',
            showmeans = True,
            fliersize = 3,
            ax = ax)
plt.xlabel("Sale Price ($)", fontsize=15)
plt.ylabel('View', fontsize=15)
plt.title('View vs. Price of the houses sold', fontsize=15)
plt.show()
```

Important house interior features vs price sale

The most important two features are grade (overall grade of the housing unit) and square footage of the home. Therefore, the most important features to predict the price sale based on home interior features are by descending order: Grade (overall grade of the housing unit), square footage of the home, total floors in the house, the square footage of interior housing living space for the nearest 15 neighbors, how good is the overall conditon of the house, square footage of the basement and the year when the house is renovated.

Visualization for the house grade

In [12]:

```
fig, ax = plt.subplots(figsize=(12,4))
sns.boxplot(y = 'grade',
            x = 'price',
            data = df,
            width = 0.8,
            orient = 'h',
            showmeans = True,
            fliersize = 3,
            ax = ax)
plt.title('Grade vs. Price of the houses sold', fontsize=15)
plt.xlabel("Sale Price ($)", fontsize=15)
plt.ylabel('Grade', fontsize=15)
plt.show()
```

Visualization of the square footage of the house

In [647]:
```
sns.lmplot(x="sqft_living", y="price",data=df,)
sns.lmplot(x="sqft_above", y="price", data=df)
```

Out[647]: <seaborn.axisgrid.FacetGrid at 0x1f51f204dc8>

Vsualization of year renovation of the house

In [686]:

```
x=df['yr_renovated']
y=df['price']

new_figure = plt.figure(figsize=(10, 5))
ax = new_figure.add_subplot(111)

plt.scatter(x, y, label='Sample Data')
ax.set_xlim(1930, 2020), ax.set_ylim(0,2500000)
ax.set_xlabel('The year house renovated')
ax.set_ylabel('Price sold ($)')
ax.set_title('Year house was renovated vs price sold');
plt.show()
```

Important features which increase the price sale

The most two important features are quare footage of house apart from basement and overall grade given to the housing unit, based on King County grading system. Therefore, the most important features which can potentially increase the price sale: square footage of house apart from basement, overall grade given to the housing unit, based on King County grading system, latitude coordinate (location) Other features: square footage of the basement, house which has a view to a waterfront, If the house has a view or not, How good is the overall condition.

Visualization of the square footage of the house

In [690]:

```
x=df['sqft_living']
y=df['price']
z=df['sqft_above']
new_figure, (ax1, ax2) = plt.subplots(figsize=(12,6), ncols=2)
new_figure.set_tight_layout(True)

ax1.scatter(x, y, color='black', linewidth=0.5, linestyle = ':')

ax2.scatter(z,y, color='green', linewidth=0.5, linestyle = '-.')

ax1.set_xlabel('Square footage of the home')
ax1.set_ylabel('Price ($)')
ax1.set_title ('Sale price vs house square footage')

ax2.set_xlabel('Square footage of the house apart from basement')
ax2.set_ylabel('Price ($)')
ax2.set_title ('Price sale vs square footage of the house excluding bas
```

Visualization of the house grade

In [717]:

```
fig = plt.figure(figsize=(15, 8))
sns.barplot(data=df, y='price', x='grade')
plt.title('House grade vs. price of the house sold', fontsize=15)
plt.xlabel("Grade", fontsize=15)
plt.ylabel('Sale Price ($)', fontsize=15)
plt.show()
```

Visualization of the house latitude location

In [716]:

```
fig = plt.figure(figsize=(15, 8))
sns.scatterplot(data=df, y='price', x='lat')
plt.title('House latitude (lat) position vs. price of the houses sold',
plt.xlabel("Latitude", fontsize=15)
plt.ylabel('Sale Price ($)', fontsize=15)
plt.show()
```

Visualization of the house view

In [714]:

```
fig = plt.figure(figsize=(10, 5))
sns.barplot(data=df, y='price', x='view')
plt.title('House landscape view vs. price of the houses sold', fontsize
plt.xlabel("View", fontsize=12)
plt.ylabel('Sale Price ($)', fontsize=12)
plt.show()
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\pandas\io\formats\forma
t.py:1403: FutureWarning: Index.ravel returning ndarray is deprecated; in a
future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```

Visualization of the house waterfront

In [713]:

```

fig = plt.figure(figsize=(5, 5))
sns.barplot(data=df, y='price', x='waterfront')
plt.title('Waterfront view vs. price of the houses sold ', fontsize=12)
plt.xlabel("Waterfront", fontsize=12)
plt.ylabel('Sale Price ($)', fontsize=12)
plt.show()
```

```
C:\Users\mirnamamaranda\anaconda3\lib\site-packages\pandas\io\formats\forma
t.py:1403: FutureWarning: Index.ravel returning ndarray is deprecated; in a
future version this will return a view on self.
  for val, m in zip(values.ravel(), mask.ravel())
```

Conclusion

The most important features to predict the sale price of the house in King County , Seattle are square footage and the grade of the house. Other features that can increase the sale price i.e. location (whic region the house is located in King County), waterfront for the house and a natural view for the house (i.e. overlooking a mountain, hill, valley, forest..ect). Overall, according to our models to predict the price, the most precise features that can increase the sale price are: square footage of the house of the 15 closet neighbors, a house with two floors, 4 views, 1 waterfront, 5.25 bathrooms, grade 12 or 13 located in certion locations (zipcode 9814, 98011, 9802, 98033, 98102...ect).

More data in different years is needed to have better prediction in order to have better visualozations for the chage of the sale price with time. If time data is provided, we can predict if sale price increase/decrease by time and estimate the value increase or decrease of the properties with time in certain areas. This information is vital because it allows real estate investors to invest in the right properties that will have profits.

Other features are missing from the data, like school district, the education quality of the schooling system, house upgrades (wood flooring, marble countertops, wood decoration, stone facade...ect). These features can influentially affect the sale price and thir incoporation into the model in essential for future consideration.