In [4]:

```
# 1. Explore this dataset using what you have learned in data preprocessing and data visual
import numpy as np #linear algebra
import pandas as pd #datapreprocessing, CSV file I/O
import seaborn as sns #for plotting graphs
import matplotlib.pyplot as plt
df=pd.read_csv('kc_house_data.csv')
df.head()
```

Out[4]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0

5 rows × 21 columns

In [2]:

```
#some general information about the data columns and values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [3]:

```
#Finding missing values
df.isnull().sum()
```

Out[3]:

```
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 [5]:

```
#Finding the count of no of bedrooms
df['bedrooms'].value_counts()
```

Out[5]:

```
3     9824
4     6882
2     2760
5     1601
6      272
1      199
7       38
8       13
0       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

In [6]:

```
#Finding the count of grade
df['grade'].value_counts()
```

Out[6]:

```
7     8981
8     6068
9     2615
6     2038
10    1134
11     399
5      242
12      90
4       29
13      13
3        3
1        1
Name: grade, dtype: int64
```

In [7]:

```
#Finding the count of condition
df['condition'].value_counts()
```

Out[7]:

```
3    14031
4     5679
5     1701
2      172
1       30
Name: condition, dtype: int64
```

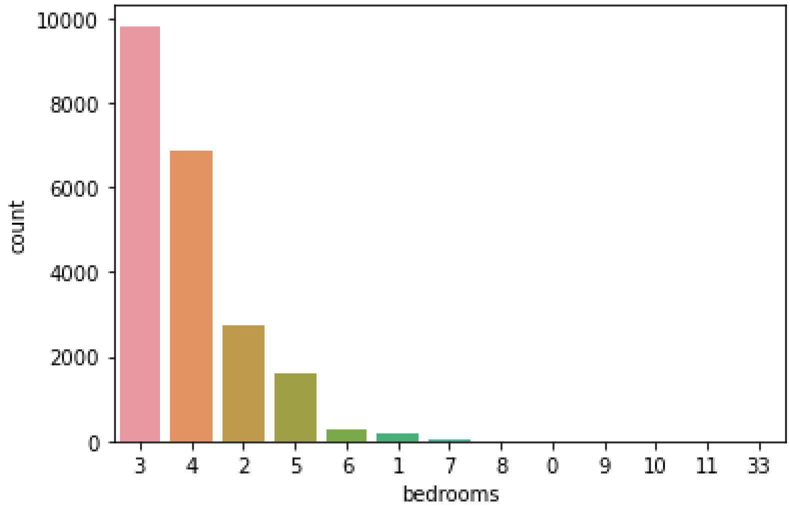
In [8]:

```
#A countplot is plotted for bedrooms
sns.countplot(df.bedrooms,order=df['bedrooms'].value_counts().index)
```

```
C:\Users\pc-H\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[8]:

```
<AxesSubplot:xlabel='bedrooms', ylabel='count'>
```



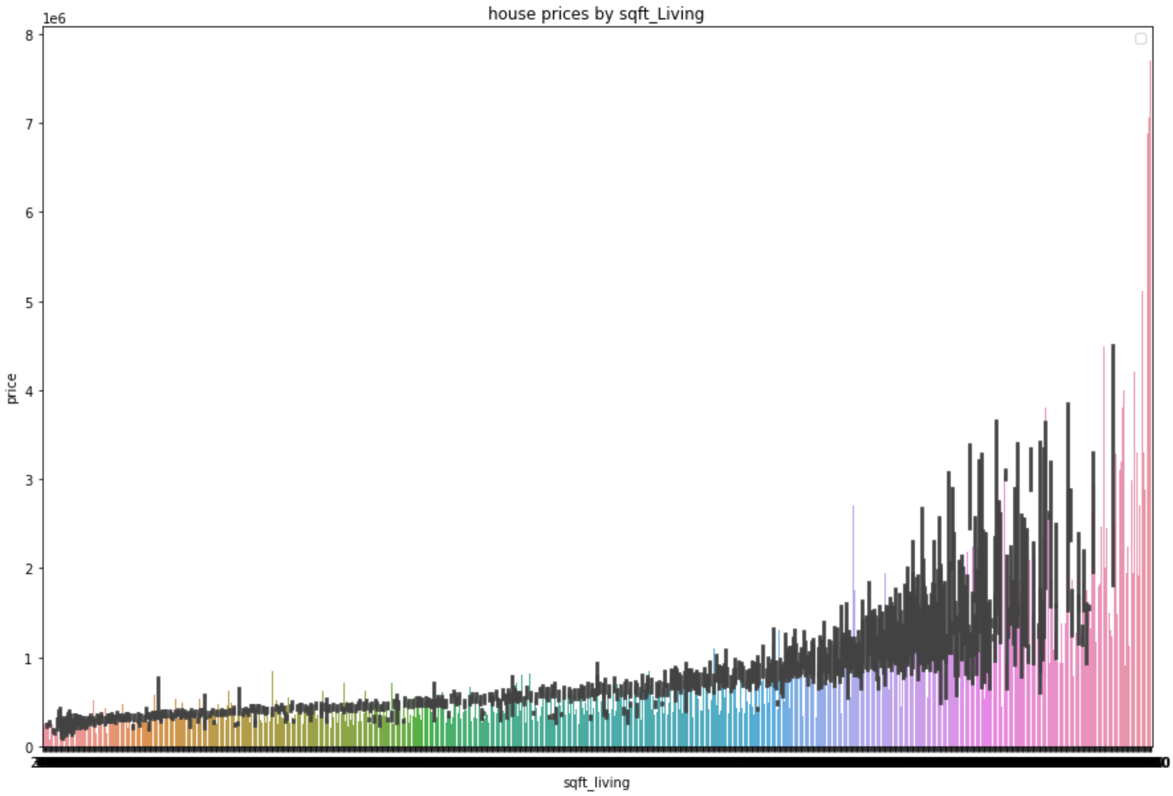
In [10]:

```
#A barplot is plotted between sqft living and prices to get an overview of how the price ch
fig,axes=plt.subplots(nrows=1,ncols=1,figsize=(15,10))
plt.title("house prices by sqft_Living")
plt.xlabel('sqft_living')
plt.ylabel('house prices')
plt.legend()
sns.barplot(x='sqft_living',y='price',data=df)
```

No handles with labels found to put in legend.

Out[10]:

<AxesSubplot:title={'center':'house prices by sqft_Living'}, xlabel='sqft_li
ving', ylabel='price'>



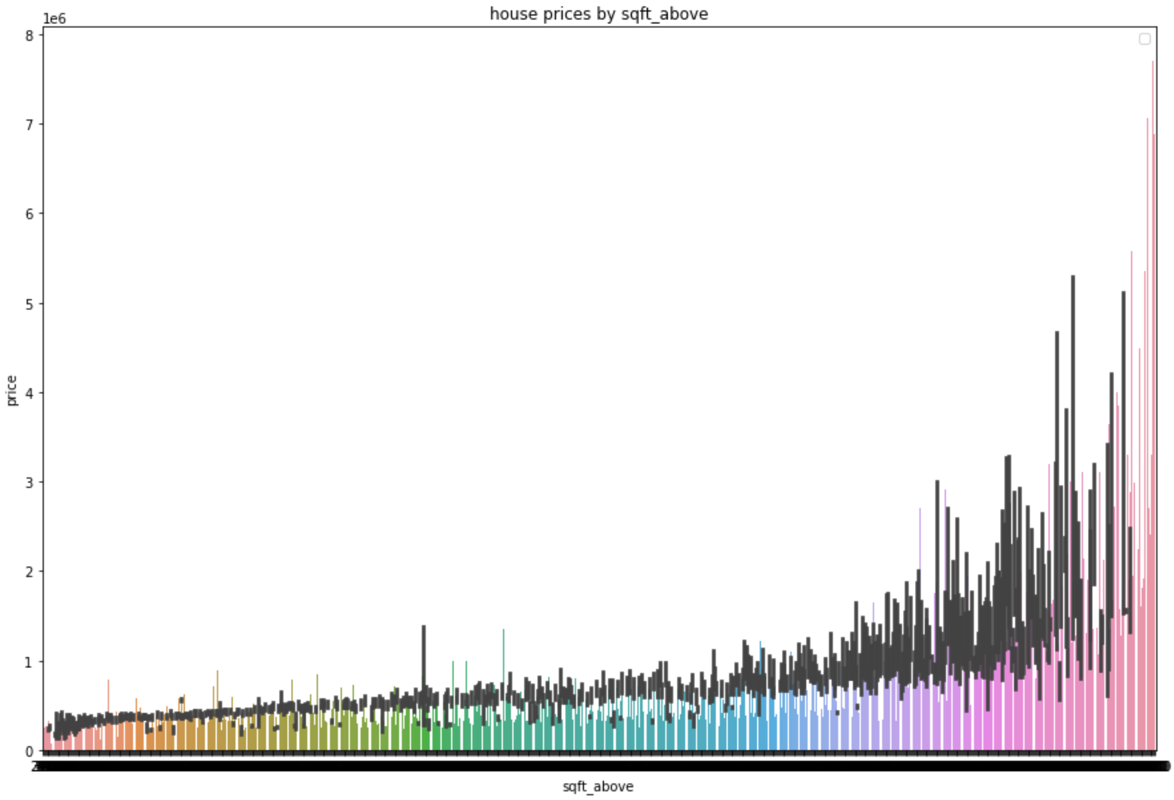
In [11]:

```
#A barplot is plotted between the sqft above and prices to see how the price changes with t
fig,axes=plt.subplots(nrows=1,ncols=1,figsize=(15,10))
plt.title("house prices by sqft_above")
plt.xlabel('sqft_above')
plt.ylabel('house prices')
plt.legend()
sns.barplot(x='sqft_above',y='price',data=df)
```

No handles with labels found to put in legend.

Out[11]:

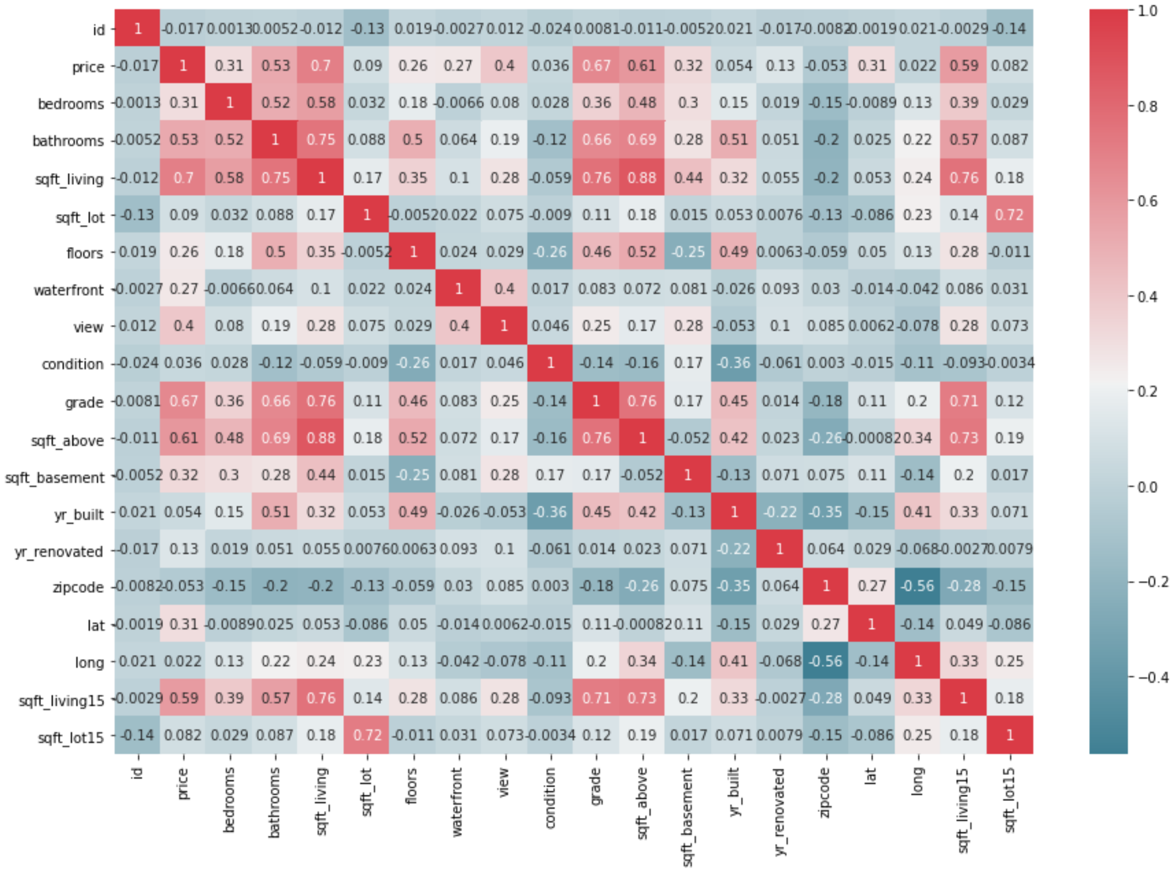
<AxesSubplot:title={'center':'house prices by sqft_above'}, xlabel='sqft_above', ylabel='price'>



In [12]:

```
def correlation_heatmap(df1):
    _,ax=plt.subplots(figsize=(15,10))
    colormap=sns.diverging_palette(220,10,as_cmap=True)
    sns.heatmap(df.corr(),annot=True,cmap=colormap)

correlation_heatmap(df)
```

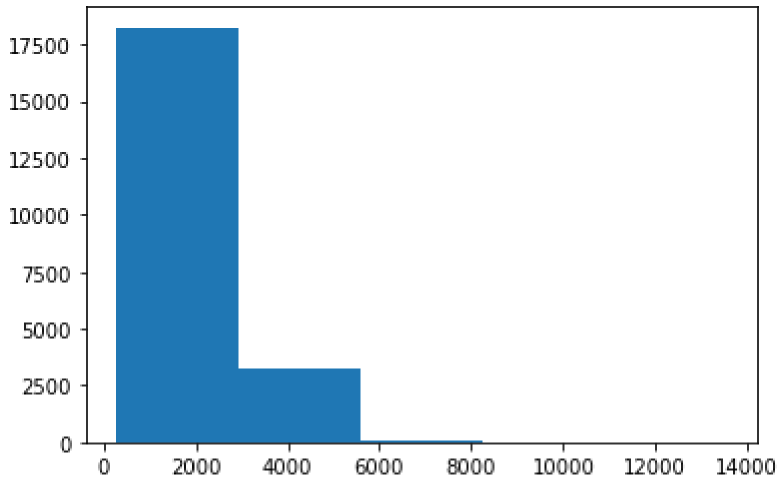


In [13]:

```
#A histogram is plotted for sqft living
plt.hist('sqft_living',data=df,bins=5)
```

Out[13]:

```
(array([1.825e+04, 3.255e+03, 1.010e+02, 5.000e+00, 2.000e+00]),
 array([  290.,  2940.,  5590.,  8240., 10890., 13540.]),
 <BarContainer object of 5 artists>)
```



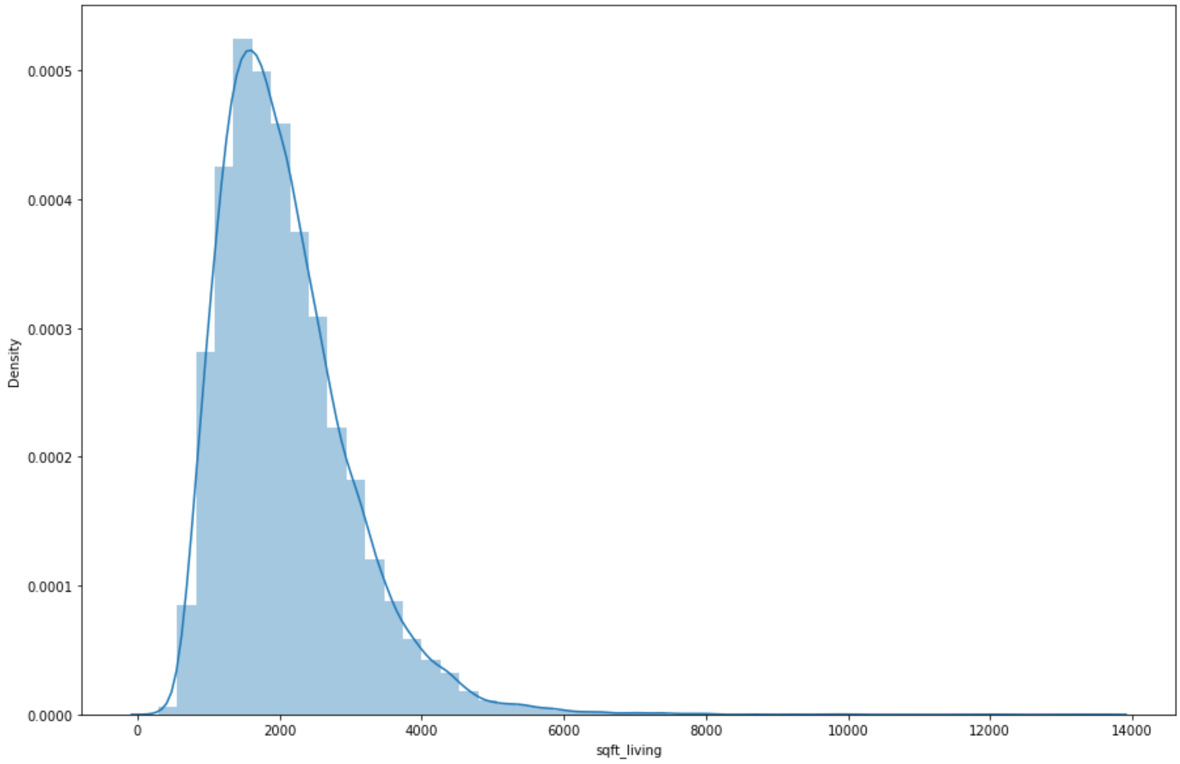
In [14]:

```
#A distplot is plotted for sqft living to see if the data is skewed or not
fig,axes=plt.subplots(nrows=1,ncols=1,figsize=(15,10))
sns.distplot(df['sqft_living'],hist=True,kde=True,rug=False,label='sqft_living',norm_hist=T
```

```
C:\Users\pc-H\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[14]:

```
<AxesSubplot:xlabel='sqft_living', ylabel='Density'>
```



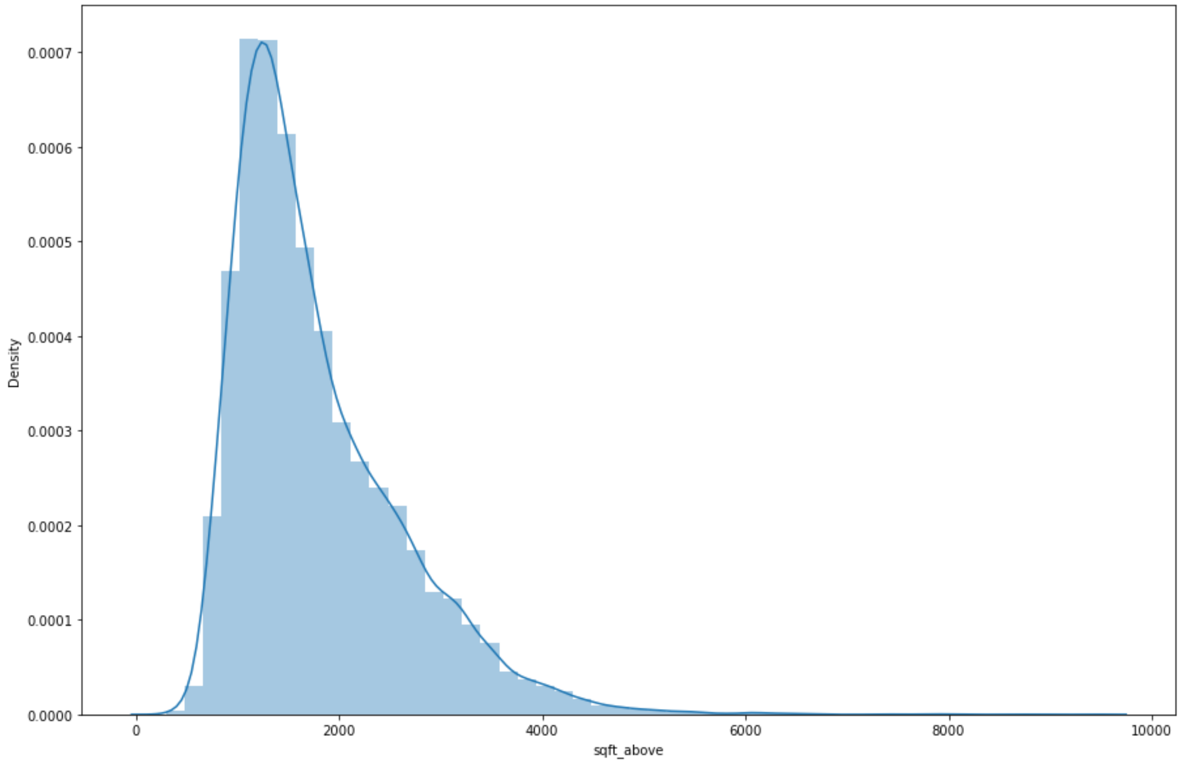
In [15]:

```
#A distplot is plotted for sqft above to see if the data is skewed or not
fig,axes=plt.subplots(nrows=1,ncols=1,figsize=(15,10))
sns.distplot(df['sqft_above'],hist=True,kde=True,rug=False,label='sqft_above',norm_hist=Tru
```

```
C:\Users\pc-H\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[15]:

```
<AxesSubplot:xlabel='sqft_above', ylabel='Density'>
```



In [16]:

```
#Finding the mean, mode and median of sqft living.
print('Mean',round(df['sqft_living'].mean(),2))
print('Median',df['sqft_living'].median())
print('Mode',df['sqft_living'].mode()[0])
```

```
Mean 2079.9
Median 1910.0
Mode 1300
```

In [17]:

```
#Through graphs we observe that the sqft living=1300 has more values.
len(df[df['sqft_living']==1300])
```

Out[17]:

```
138
```

In [18]:

```
#3. Split your dataset into a training set and a testing set.
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

In [19]:

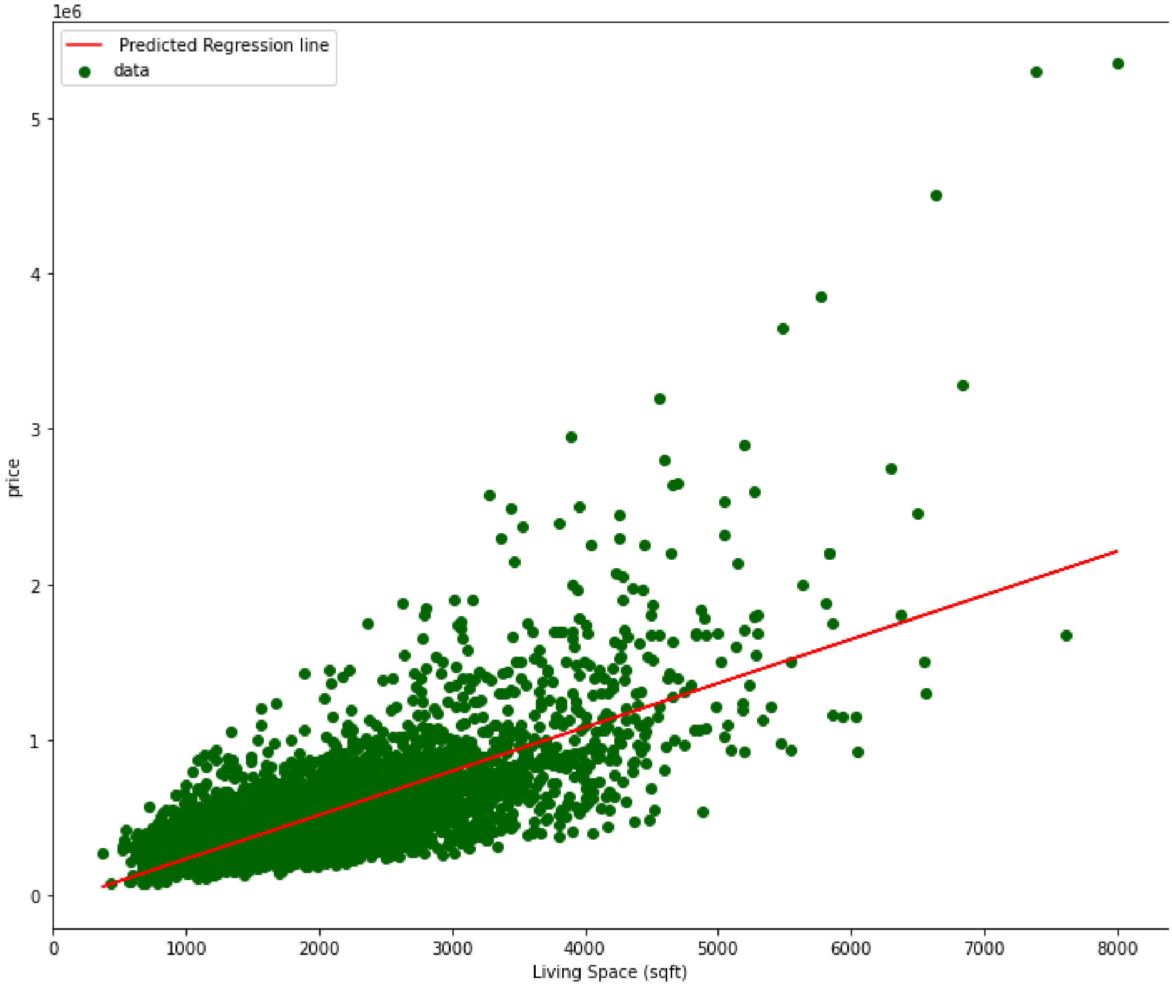
```
train_data,test_data=train_test_split(df,train_size=0.8,random_state=3)
reg=linear_model.LinearRegression()
x_train=np.array(train_data['sqft_living']).reshape(-1,1)
y_train=np.array(train_data['price']).reshape(-1,1)
reg.fit(x_train,y_train)

x_test=np.array(test_data['sqft_living']).reshape(-1,1)
y_test=np.array(test_data['price']).reshape(-1,1)
pred=reg.predict(x_test)
print('linear model')
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('Sqaured mean error', round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(x_train,y_train),3))
print('R sqaured testing',round(reg.score(x_test,y_test),3) )
print('intercept',reg.intercept_)
print('coefficient',reg.coef_)
```

```
linear model
Sqaured mean error 254289.15
R squared training 0.492
R sqaured testing 0.496
intercept [-47235.8113029]
coefficient [[282.2468152]]
```

In [20]:

```
_, ax = plt.subplots(figsize= (12, 10))
plt.scatter(x_test, y_test, color= 'darkgreen', label = 'data')
plt.plot(x_test, reg.predict(x_test), color='red', label= ' Predicted Regression line')
plt.xlabel('Living Space (sqft)')
plt.ylabel('price')
plt.legend()
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```



In [21]:

```
train_data,test_data=train_test_split(df,train_size=0.8,random_state=3)
reg=linear_model.LinearRegression()
x_train=np.array(train_data['grade']).reshape(-1,1)
y_train=np.array(train_data['price']).reshape(-1,1)
reg.fit(x_train,y_train)

x_test=np.array(test_data['grade']).reshape(-1,1)
y_test=np.array(test_data['price']).reshape(-1,1)
pred=reg.predict(x_test)
print('linear model')
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('squared mean error',round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(x_train,y_train),3))
print('R squared testing',round(reg.score(x_test,y_test),3))
print('intercept',reg.intercept_)
print('coeeficient',reg.coef_)
```

```
linear model
squared mean error 263387.61
R squared training 0.442
R squared testing 0.46
intercept [-1061459.62144314]
coeeficient [[209225.48270386]]
```

In [ ]:

```
#Multi-linear Regression Code
```

In [23]:

```
features1=['bedrooms','grade','sqft_living','sqft_above']
reg=linear_model.LinearRegression()
reg.fit(train_data[features1],train_data['price'])
pred=reg.predict(test_data[features1])
print('complex_model 1')
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('mean squared error(MSE)', round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(train_data[features1],train_data['price']),3))
print('R squared training', round(reg.score(test_data[features1],test_data['price']),3))
print('Intercept: ', reg.intercept_)
print('Coefficient:', reg.coef_)
```

```
complex_model 1
mean squared error(MSE) 239014.4
R squared training 0.548
R squared training 0.555
Intercept:  -523645.7841467742
Coefficient: [-4.33050242e+04  1.03455986e+05  2.73023590e+02 -8.38875593e+0
1]
```

In [24]:

```
features2 = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view','
reg= linear_model.LinearRegression()
reg.fit(train_data[features1],train_data['price'])
pred = reg.predict(test_data[features1])
print('Complex Model_2')
mean_squared_error = metrics.mean_squared_error(y_test, pred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(reg.score(train_data[features1], train_data['price']),
print('R-squared (testing) ', round(reg.score(test_data[features1], test_data['price']), 3)
print('Intercept: ', reg.intercept_)
print('Coefficient:', reg.coef_)
```

```
Complex Model_2
Mean Squared Error (MSE)  239014.4
R-squared (training)  0.548
R-squared (testing)  0.555
Intercept:  -523645.7841467742
Coefficient: [-4.33050242e+04  1.03455986e+05  2.73023590e+02 -8.38875593e+0
1]
```

In [ ]:

```
#Polynomial Regression
```

In [25]:

```
#For degree=2, the linear modelis built. The mean squared error is calculated and r squared
polyfeat=PolynomialFeatures(degree=2)
xtrain_poly=polyfeat.fit_transform(train_data[features1])
xtest_poly=polyfeat.fit_transform(test_data[features1])

poly=linear_model.LinearRegression()
poly.fit(xtrain_poly,train_data['price'])
polypred=poly.predict(xtest_poly)

print('Complex Model_3')
mean_squared_error = metrics.mean_squared_error(test_data['price'], polypred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(poly.score(xtrain_poly, train_data['price']), 3))
print('R-squared (testing) ', round(poly.score(xtest_poly, test_data['price']), 3))
```

```
Complex Model_3
Mean Squared Error (MSE)  221965.07
R-squared (training)  0.614
R-squared (testing)  0.616
```

In [ ]:

```
#Desciption:
#the most important features selected are: price,bedrooms','grade','sqft_living','sqft_abov
#A heatmap is a two-dimensional graphical representation of data where the individual value
#Scikit-learn provides a range of supervised and unsupervised learning algorithms via a con
#we splitting the data into 80:20 ratio of which train_size is 80%, test_size is 20%. train
#Polynomial Regression is a form of linear regression in which the relationship between the
#For degree=2, the linear modelis built. The mean squared error is calculated and r squared
#Polynomial Regression provides the best approximation of the relationship between the depe
```