Grading

The final score that you will receive for your programming assignment is generated in relation to the total points set in your programming assignment item—not the total point value in the nbgrader notebook. When calculating the final score shown to learners, the programming assignment takes the percentage of earned points vs. the total points provided by nbgrader and returns a score matching the equivalent percentage of the point value for the programming assignment.
DO NOT CHANGE VARIABLE OR METHOD SIGNATURES The autograder will not work properly if your change the variable or method signatures.

WARNING

Please refrain from using **print statements/anything that dumps large outputs(>1000 lines) to STDOUT** to avoid running to into **memory issues**. Doing so requires your entire lab to be reset which may also result in loss of progress and you will be required to reach out to Coursera for assistance with this. This process usually takes time causing delays to your submission.

Validate Button

Please note that this assignment uses nbgrader to facilitate grading. You will see a **validate button** at the top of your Jupyter notebook. If you hit this button, it will run tests cases for the lab that aren't hidden. It is good to use the validate button before submitting the lab. Do know that the labs in the course contain hidden test cases. The validate button will not let you know whether these test cases pass. After submitting your lab, you can see more information about these hidden test cases in the Grader Output.
Cells with longer execution times will cause the validate button to time out and freeze. Please know that if you run into Validate time-outs, it will not affect the final submission grading.

EDA, Simple Linear Regression

In this assignment, we will use a simplified data and create a simple linear regression model. The dataset can be downloaded from https://www.kaggle.com/harlfoxem/housesalesprediction/download (https://www.kaggle.com/harlfoxem/housesalesprediction/download).
This dataset contains house sale prices for Kings County, which includes Seattle. It includes homes sold between May 2014 and May 2015. There are several versions of the data. Some additional information about the columns is available here: https://geodacenter.github.io/data-and-lab/KingCounty-HouseSales2015/ (https://geodacenter.github.io/data-and-lab/KingCounty-HouseSales2015/), some of which are copied below.

Variable	Description
id	Identification
date	Date sold
price	Sale price
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms
sqft_liv	Size of living area in square feet
sqft_lot	Size of the lot in square feet
floors	Number of floors
waterfront	‘1’ if the property has a waterfront, ‘0’ if not.
view	An index from 0 to 4 of how good the view of the property was
condition	Condition of the house, ranked from 1 to 5
grade	Classification by construction quality which refers to the types of materials used and the quality of workmanship. Buildings of better quality (higher grade) cost more to build per unit of measure and command higher value.
sqft_above	Square feet above ground
sqft_basmt	Square feet below ground
yr_built	Year built
yr_renov	Year renovated. ‘0’ if never renovated
zipcode	5 digit zip code
lat	Latitude
long	Longitude
squft_liv15	Average size of interior housing living space for the closest 15 houses, in square feet
squft_lot15	Average size of land lost for the closest 15 houses, in square feet

In [1]:
```
import scipy as sp
import scipy.stats as stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import copy
# Set color map to have light blue background
sns.set()
import statsmodels.formula.api as smf
import statsmodels.api as sm
%matplotlib inline
```

1. Munging data [15 pts]

In this part, let's load and inspect data. We will also learn how to transform columns when needed.

> 💡***Tip:*** `pd.read_csv(<file path>)` reads a csv file and returns to pandas data frame object. It can also read files with other delimiter such as `.tsv` files. pandas also has `pd.read_excel` to read excel files. [See more in the document (https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html)](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html)

In [2]:
```
df = pd.read_csv('data/house_data_washington.csv')
```

1a) Date string to numbers [5 pts]

Now, let's overview the dataframe. Using `.head()` on the dataframe, we can see the first 5 rows of the data. You can specify number of rows as argument then it will show those number of rows. similarly, `.tail()` gives the last 5 rows by default. You can see the columns names, but not all columns are displayed if there are too many columns.

> 💡 ***Tip:*** If you want to show all columns and rows, there are [pandas command (https://www.geeksforgeeks.org/show-all-columns-of-pandas-dataframe-in-jupyter-notebook/#)](https://www.geeksforgeeks.org/show-all-columns-of-pandas-dataframe-in-jupyter-notebook/#) setting max rows and cols. Please do not submit your homework notebook with displaying large dataframe because it may crash from large memory consumption.

The column 'date' is the date sold (with some black timestamp as well), and the data is string type (Note that sometimes data tables may have date/time columns as datetime object types. In this example data, it has a string type). We will extract year and month information from the string. In the data frame `df`, let's create new features 'sales_year' and 'sales_month' using 'date' column.

> 💡 ***Tip:*** You can use either bracket (e.g. `df['date']`) or dot (e.g. `df.date`) to get the column `'date'` in the data frame `df`. A single columns object from dataframe is a pandas series object type, and you can use `.apply()` method for a transformation. `.apply()` is generic and can be applied to not only to single column (pandas series) but also to multiple columns (pandas dataframe). Here, we will apply it to a single column object and use `lambda` function inside the `.apply()` as shown below. For more examples, see the [doc (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html)](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html).

In this case, when we inspect the 'date' column, it is a string object, so we can slice the year and month from the string. Also, we'd like to convert the extracted year and month strings to ingeters.

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

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floo
0	7129300520	20141013T000000	221900	3	1.00	1180	5650	1.
1	6414100192	20141209T000000	538000	3	2.25	2570	7242	2.
2	5631500400	20150225T000000	180000	2	1.00	770	10000	1.
3	2487200875	20141209T000000	604000	4	3.00	1960	5000	1.
4	1954400510	20150218T000000	510000	3	2.00	1680	8080	1.

5 rows × 21 columns

In [4]:
```python
print(df.date)
print(type(df.date.iloc[0]))
```

```
0        20141013T000000
1        20141209T000000
2        20150225T000000
3        20141209T000000
4        20150218T000000
              ...
21608    20140521T000000
21609    20150223T000000
21610    20140623T000000
21611    20150116T000000
21612    20141015T000000
Name: date, Length: 21613, dtype: object
<class 'str'>
```

In [5]:
```python
# extract year and month info from the string
# create new features 'sales_year' and 'sales_month' in df

df['sales_year'] = df.date.apply(lambda x: int(x[:4]))
df['sales_month'] = df.date.apply(lambda x: int(x[4:6]))
```

In [6]:
```python
df.groupby('sales_month')
```

Out[6]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x780db88945d0>
```

Now, let's count how many sales occurred in each month and each year. We can use `.groupby()` function to group by 'sales_month' and 'sales_year' as shown below.

> 💡***Tip:*** `.groupby()` itself returns an object that doesn't get displayed, not a processed dataframe. It is often used with some other aggregation method, such as `.count()`, `.mean()`, `.sum()`, etc. In the below example use case, we use `.count()` to count number of sales per each group (e.g. by sales_month). `.groupby()` can also group by multiple columns. This [resource (https://realpython.com/pandas-groupby/)](https://realpython.com/pandas-groupby/) has more explanations and examples.

```
In [7]: print(df.groupby('sales_month')['id'].count())
        print(df.groupby('sales_year')['id'].count())
```

```
sales_month
1      978
2     1250
3     1875
4     2231
5     2414
6     2180
7     2211
8     1940
9     1774
10    1878
11    1411
12    1471
Name: id, dtype: int64
sales_year
2014    14633
2015     6980
Name: id, dtype: int64
```

Question 1a-1. Based on the output from above cell, which month has the most number of sales?

```
In [8]: # your code here

        # uncomment below and update the value as an integer
        most_sales_month = (df.groupby('sales_month')['id'].count()).idxmax()
        most_sales_month
```

```
Out[8]: 5
```

```
In [9]: # tests solutions for most_sales
```

Question 1a-2. Which months has the least number of sales?

```
In [10]: # your code here

         # uncomment below and update the value as an integer
         least_sales_month = (df.groupby('sales_month')['id'].count()).idxmin()
         least_sales_month
```

```
Out[10]: 1
```

```
In [11]: # tests solutions for least_sales
```

Now, let's have a look at what data type each columns has. We can use `.info()` method on the dataframe object to see the data type. You can see `int64`, `float64` and `object` in our example. `object` can be string type or something else (such as list or other types of objects).

> 💡***Tip:*** Note that sometimes raw data is not adequately formatted that you might see columns that are supposed to be numbers can be typed as strings. It is a good practice to inspect data columns's data types and clean them if necessary.

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  int64  
 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  
 21  sales_year     21613 non-null  int64  
 22  sales_month    21613 non-null  int64  
dtypes: float64(4), int64(18), object(1)
memory usage: 3.8+ MB
```

1b) Variable types [5 pts]

Inspect each feature's data type and variable type. What is the best description for the variable type of following features? Update the string to 'numeric' or 'categorical'.

Self-check Concepts

✔ What data types can be considered as a numeric variable?

✔ What is the difference between ordinal and non-ordinal categorical variables?

> 💡***Tip:*** Is binary categorical variable (Yes/No, Male/Female, True/False, Positive/Negative etc) numeric? Why or why not?
> How about a variable that has meaning of degree- such as survey/review ratings (very satisfied = 5, satisfied = 4, neutral = 3, disatisfied = 2, very disatisfied = 1)?
> Typically it is recommended to treat ordinal categorical variable (which order has meaning- e.g. degree, grades, numbers, severity etc) as numeric variable because a linear regression (or any ML) model can treat that variable (feature) as numbers and can learn a relationship to the target variable y. Also, categorical variables need to be binarized (which involves to transform the column into multiple binary columns) before used in a linear regression model. So if we treat an ordinal categorical variable to one numeric variable column instead of multiple binary columns representing categorical variable, it is more efficient for the model. Remember- a simpler model with the same information is better!

In [13]:
```
# your code here

# Uncomment the features below and update the strings with 'numeric' or 'categorical'
price = 'numeric'
bathrooms = 'numeric'
waterfront = 'categorical'
grade = 'numeric'
zipcode = 'categorical'
sales_year = 'numeric'
```

In [14]:
```
# You can use below to check what unique values exist in each column.
for c in df.columns[2:]:
    print(c, df[c].unique());
```

```
price [ 221900  538000  180000 ...  610685 1007500  402101]
bedrooms [ 3  2  4  5  1  6  7  0  8  9 11 10 33]
bathrooms [1.   2.25 3.   2.   4.5  1.5  2.5  1.75 2.75 3.25 4.   3.5  0
.75 4.75
 5.   4.25 3.75 0.   1.25 5.25 6.   0.5  5.5  6.75 5.75 8.   7.5  7.75
 6.25 6.5 ]
sqft_living [1180 2570  770 ... 3087 3118 1425]
sqft_lot [ 5650  7242 10000 ...  5813  2388  1076]
floors [1.  2.  1.5 3.  2.5 3.5]
waterfront [0 1]
view [0 3 4 2 1]
condition [3 5 4 1 2]
grade [ 7  6  8 11  9  5 10 12  4  3 13  1]
sqft_above [1180 2170  770 1050 1680 3890 1715 1060 1890 1860  860 1430
1370 1810
 1980 1600 1200 1250 2330 2270 1070 2450 1710 1750 1400  790 2570 2320
 1190 1510 1090 1280  930 2360  890 2620 2600 3595 1570  920 3160  990
 2290 2165 1640 1000 2130 2830 2250 2420 3250 1850 1590 1260 2519 1540
 1110 1770 2720 2240 3070 2380 2390  880 1040  910 3450 2350 1900 1010
  960 2660 1610  765 3520 1290 1960 1160 1210 1270 1440 2190 2920 1460
 1170 1240 3140 2030 2310  700 1080 2520 2780 1560 1450 1720 2910 1620
 1360 2070 2460 1390 2140 1320 1340 1550  940 1380 3670 2370 1130  980
 3540 2500 1760 1030 1780 3400 2680 1670 2590  820 1220 2440 2090 1100
 1330 1420 1690 2150 1910 1350 1940  900 1630 2714  850 1870 1950 2760
 2020 1120 1480 1230 2280 3760 3530  830 1300 2740 1830  720 2010 3360
  800 1730  760 1700 4750 5310  580 2653 2850 2210 2630 3500 1740 1140
 2160 2650  970 2040 2180 2220 1660 3370 2690 1930 3150 3030 2050 2490
 2560 1275 2580  560 1820 1840 2990 3230 1580 3480 2510 1410 2120 3300
 3840 1500 1530 2840  833 2000 6070  950 2200 4040 1920 1490 3470 3130
 2610 3260 2260  430 3390  630 4860 3860 2810  870 3180 2770 4030 4410
 2400 1520 3040 6050 4740 1970 5403 3350 3580 1790  750 2860 2750 2340
 2870 4120 3200 2550 1805 4150 1384 2060 2110 3590 2100 2540 1880 1150
 1470 1255 1800 4370 3190 2730 4570 2470  670 2900 4670 4230 2156 1020
 2940 2640 2710 3100 3610 4270  840 3090 2300  380 2480 3460 3060 3064
 3000 1654 2790 1310 2230 2430 3680 2670 2208  810  740 1422  490 2080
 3440 5670 4475  730 3410 3010  600 2960 3570 4300 3990  780 3020 5990
  440 4460 4190 2800 2530 1650 3690 2932 3720 4250 3110 2963 4930 2950
 5000 2452 2820 1981  640 2495 2403 5320 6720  660 2341 4210 3830 3280
 2980 5153 1990 1646  610  710 5450 3504 3210 1782 2930  590 4280  680
 3880 3430 3750 4130 5710 3380 3330 4700 3220 3362 3510 3810  620 4490
 2410 3050 1008 3488 4070 3420 5770 1605  520 1088 3555 4360 3960 2700
 4340 1552 3850 2303 3270 4350 3640 2174 4160 2496 5180 5130 6350 3770
 2153 3780 2890 1714 2201 2970  992 3950 3527 2835 3915 1427 4870 3340
 3620 4310 3930 4080 5400  570 3310 6110 3320 3490 3859 3710 1798 4600
 3560 3940 3600 3800 1105 2305 3290 5050 1556 1553 4000 1657 3001 4220
  480 3120 3740  530 3700 5230 5370 3080 4140 4430 3550 1159 1288 2880
 4610 1122 3052 1479 7680 3820 1934 5080 2675 2506 5760 2154 4390 3240
 1995 1689 2782 2395 4400 6200 3526 4320 2483 4380 4580 4180 2064 3650
 1726 2019 4240 1256  500 1355 1747 1678 1833 1414 4115 3597 3170  390
 1976 5830 2601 3920 2641 5070 2518 3910 3660 3695 4020 2803 2074 2038
 4060 4890 2329 1264 1095  690 4090 1392 2844  902 4560 2811 4720 2168
 5610 2683 4900 2095 4290 4050 4260 4440 6220 1175  998 2356 4500 3900
 3831 1315 4470 4810 2286 2927 4760 8570 5140 1679 1811 2849 1676 1757
 3730 2441 2163 5250 2795 2415 3970 4200 1068 5240 1509 1954 4820 1651
 4100 1752 3630 2885 3154 1129 2632 1996 4010  550  410 6430 3790 2031
 1652 2434 3316 1899 2331 2497 2216 4170 1341 1961 5584 8860 2507 5220
 4850 5844 5530 2145  650 1982 4910 3605 1778 1463 2783 1946 1358 3870
 1864 1845 6290 3980 2382 2979 3674 2726 5440 1295 2115 6085 3265 3136
 6640 4620 3361 2245 2242 1078 2577 1329  420 4330 1975 7420 1788 2299
 1092 4225 1087 1904  470 2966 2192 2253 5550 4133 4285 1216  540 9410
```

```
 2075 5330 2166 1628 1808 1352 2557 6380 7880 2734 1363 1769 2093 1677
 2588 5190 2298 1491 2961 5020 5980 4540  844 6120 2233 4480 4110 4770
 2473  995 5160 1494 2007 1048 3002 4780 2155 2014 4980 2665 4830 4790
 5010  370 2105 3006 3004 2689 4660 1746 2678 2755 2414  901 4630 2068
 2807 2643 2181 4510 4420 1604 1435 3045 2717 2905 4940 5110 2533 6660
 3485 2659 5090 2375 1964  866 1595  944 5480  809 5040 1764 1656 1802
  460 2692 1544 2044 1212 4083 8020 3905 1502 4590  384 2092 6090 1615
 7320 1396 1484 1765 5490 1453 1643 5300 1381 4065  290 1313 5430 1397
 2793 2475 1936 3028  798 2575 3276 1584 2393 2029 3222 1072 1785 1984
  962 2423 2052 2538 2437 2789 2906 4800 7850 2196 1847 2658 2655 3855
 1728  963 2223 1611 2015 2448 1489 1116 3745 1002 3202 1347 1481 2311
 2544 2584 2217 3569 3181 1921 2612 2671 2598 3284 3266 1076 2594 2718
 1794 2481 3845 1413 1876 3148 2413 1767 5060  806 2547 1834 2024 1165
 2134 1741 2798 1852 2099 3216 1094 2891 2432 2283 2701 1658  893 2009
 1444 2744 3078 3065 1578 2815 4960 1571 6530 4640 1536 3172 6370 3223
 1608 2229 3135 1408 1763 4840 1232 2502 2424 1296 1914  988 3828 3056
 2267 1131 2796 1812 1084 2025 1564 1239 2568 1528 2628 2185 2478 2669
 1912 2828 2425 1446 3206 2406 1419 2056 1144 2456 4950 3192  828 2529
 2732 1987 3906 4073 2578 2738 3691 1061 2846 2542 1889 3336 3236 1451
 1983 2313 1824 1322 1766 2301 3274 1108 2864 2716 1572 3281 2656 2398
 1867 1613 2587 2623  894 1606 2244 2026 2238 2517 2708 2555 1405 4450
 1248 6420 2531 1333 2198 3087 3118 1425]
sqft_basement [   0  400  910 1530  730 1700  300  970  760  720  700  8
20  780  790
  330 1620  360  588 1510  410  990  600  560  550 1000 1600  500 1040
  880 1010  240  265  290  800  540  380  710  840  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
 1100  280  870  460 1400 1320  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  610  520  890 1110  150 1720  810  190 1290  670
 1800 1120 1810   60 1050  940  310  930 1390 1830 1300  510 1330 1590
  920 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 2590 1090  110 2500   90 1940 1550 2350 2490 1481 1360 1135 1520
 1850 1660 2130 2600 1690  243 1210 2620 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  894 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]
yr_built [1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 1942 1927 19
77 1900
 1979 1994 1916 1921 1969 1947 1968 1985 1941 1915 1909 1948 2005 1929
 1981 1930 1904 1996 2000 1984 2014 1922 1959 1966 1953 1950 2008 1991
 1954 1973 1925 1989 1972 1986 1956 2002 1992 1964 1952 1961 2006 1988
 1962 1939 1946 1967 1975 1980 1910 1983 1978 1905 1971 2010 1945 1924
 1990 1914 1926 2004 1923 2007 1976 1949 1999 1901 1993 1920 1997 1943
 1957 1940 1918 1928 1974 1911 1936 1937 1982 1908 1931 1998 1913 2013
 1907 1958 2012 1912 2011 1917 1932 1944 1902 2009 1903 1970 2015 1934
 1938 1919 1906 1935]
yr_renovated [   0 1991 2002 2010 1999 1992 2013 1994 1978 2005 2008 200
3 1984 1954
 2014 2011 1974 1983 1945 1990 1988 1957 1977 1981 1995 2000 1998 1970
 1989 2004 1986 2009 2007 1987 1973 2006 1985 2001 1980 1971 1979 1997
 1950 1969 1948 2015 1968 2012 1963 1951 1993 1962 1996 1972 1953 1955
```

```
 1982 1956 1940 1976 1946 1975 1958 1964 1959 1960 1967 1965 1934 1944]
zipcode [98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 980
07 98115
 98107 98126 98019 98103 98002 98133 98040 98092 98030 98119 98112 98052
 98027 98117 98058 98001 98056 98166 98023 98070 98148 98105 98042 98008
 98059 98122 98144 98004 98005 98034 98075 98116 98010 98118 98199 98032
 98045 98102 98077 98108 98168 98177 98065 98029 98006 98109 98022 98033
 98155 98024 98011 98031 98106 98072 98188 98014 98055 98039]
lat [47.5112 47.721  47.7379 ... 47.3906 47.3339 47.6502]
long [-122.257 -122.319 -122.233 -122.393 -122.045 -122.005 -122.327 -12
2.315
 -122.337 -122.031 -122.145 -122.292 -122.229 -122.394 -122.375 -121.962
 -122.343 -122.21  -122.306 -122.341 -122.169 -122.166 -122.172 -122.218
 -122.36  -122.314 -122.304 -122.11  -122.07  -122.357 -122.368 -122.157
 -122.31  -122.132 -122.362 -122.282 -122.18  -122.027 -122.347 -122.016
 -122.364 -122.175 -121.977 -122.371 -122.151 -122.301 -122.451 -122.322
 -122.189 -122.384 -122.369 -122.281 -122.29  -122.114 -122.122 -122.116
 -122.149 -122.339 -122.335 -122.344 -122.32  -122.297 -122.192 -122.215
 -122.16  -122.179 -122.287 -122.036 -122.073 -121.987 -122.125 -122.34
 -122.025 -122.008 -122.291 -122.365 -122.199 -122.194 -122.387 -122.372
 -122.391 -122.351 -122.386 -122.249 -122.277 -122.378 -121.958 -121.714
 -122.08  -122.196 -122.184 -122.133 -122.38  -122.082 -122.109 -122.053
 -122.349 -122.295 -122.253 -122.248 -122.303 -122.294 -122.226 -122.266
 -122.098 -122.212 -122.244 -122.39  -122.352 -121.85  -122.152 -122.054
 -122.072 -121.998 -122.296 -122.299 -122.381 -122.358 -122.128 -122.171
 -122.174 -122.026 -122.353 -121.943 -122.286 -122.336 -122.359 -122.162
 -122.3   -122.176 -121.996 -122.118 -122.193 -122.023 -122.224 -122.168
 -122.231 -122.331 -122.374 -122.182 -122.308 -122.307 -121.999 -122.376
 -122.2   -122.039 -122.102 -122.188 -122.379 -122.043 -122.153 -122.191
 -122.219 -122.312 -121.911 -121.994 -122.165 -122.37  -122.158 -122.047
 -122.284 -122.017 -122.275 -122.268 -122.367 -122.217 -122.373 -122.013
 -122.214 -122.034 -122.164 -121.899 -122.183 -121.95  -122.324 -122.216
 -122.395 -122.213 -122.345 -122.278 -122.111 -121.711 -122.27  -122.178
 -122.147 -121.772 -122.302 -122.438 -122.223 -122.042 -122.323 -122.255
 -122.4   -122.261 -122.071 -122.206 -122.272 -122.23  -122.144 -122.143
 -122.181 -122.154 -122.311 -122.274 -122.077 -122.    -122.298 -122.058
 -121.837 -122.333 -122.057 -122.252 -122.093 -122.012 -122.052 -122.354
 -122.22  -122.49  -121.875 -122.24  -122.078 -122.173 -121.854 -122.222
 -122.28  -122.137 -122.159 -121.974 -122.141 -122.029 -121.709 -122.19
 -121.97  -122.329 -122.195 -122.06  -121.959 -122.095 -122.148 -122.146
 -122.35  -121.901 -122.241 -122.129 -122.289 -122.305 -122.022 -122.385
 -121.779 -122.032 -122.402 -122.482 -122.227 -121.982 -122.161 -122.046
 -122.156 -122.127 -122.33  -122.197 -122.041 -122.103 -122.318 -122.382
 -122.271 -121.955 -122.211 -122.262 -122.258 -122.121 -122.221 -122.234
 -122.089 -122.123 -122.167 -121.909 -122.107 -122.064 -122.066 -122.062
 -122.264 -122.186 -122.087 -121.88  -121.864 -122.205 -122.363 -122.139
 -122.018 -122.225 -122.285 -122.084 -122.177 -122.056 -122.316 -122.021
 -122.348 -122.009 -122.131 -122.411 -122.198 -122.256 -122.117 -122.097
 -122.075 -121.845 -122.083 -122.259 -121.87  -122.015 -122.007 -121.86
 -122.409 -121.755 -121.972 -122.251 -122.317 -121.776 -122.115 -122.283
 -122.242 -122.001 -122.024 -122.309 -122.113 -121.771 -122.239 -122.273
 -122.396 -122.094 -122.267 -122.326 -122.13  -122.269 -121.853 -122.05
 -122.346 -122.076 -121.826 -122.124 -121.758 -122.202 -121.785 -121.872
 -122.006 -122.004 -122.321 -121.882 -122.101 -122.03  -122.185 -122.1
 -121.759 -121.965 -122.201 -122.366 -122.313 -122.405 -122.02  -122.279
 -122.355 -121.934 -122.15  -122.356 -121.993 -122.044 -122.134 -121.867
 -122.01  -121.991 -122.011 -121.983 -122.228 -122.033 -122.276 -122.119
 -121.937 -122.361 -122.325 -122.203 -122.136 -122.237 -122.209 -122.049
 -122.288 -122.106 -122.037 -122.207 -122.263 -121.915 -122.204 -122.09
 -122.069 -121.852 -121.787 -121.976 -122.377 -122.059 -122.383 -121.989
```

```
 -122.019 -122.208 -121.878 -122.328 -122.25  -122.338 -122.388 -122.265
 -122.332 -122.399 -122.397 -122.014 -121.956 -122.092 -122.028 -122.293
 -122.12  -122.035 -122.14  -122.04  -122.112 -121.906 -122.17  -122.238
 -122.512 -121.997 -121.89  -122.463 -121.908 -122.086 -122.389 -121.913
 -122.163 -121.918 -122.108 -122.502 -122.392 -122.236 -121.859 -121.981
 -122.342 -121.96  -121.978 -122.47  -121.91  -121.966 -122.065 -122.246
 -122.41  -121.879 -122.079 -122.099 -122.187 -121.98  -122.002 -122.138
 -121.898 -122.235 -122.126 -121.782 -121.995 -122.401 -121.858 -121.888
 -121.752 -122.063 -122.26  -121.78  -121.708 -121.721 -122.403 -121.945
 -122.243 -122.45  -121.927 -122.085 -122.088 -121.973 -122.055 -122.398
 -121.984 -121.912 -121.903 -121.946 -122.232 -122.412 -122.104 -122.048
 -122.479 -122.155 -121.833 -121.778 -122.003 -121.99  -121.926 -122.051
 -121.986 -122.245 -121.861 -122.431 -121.964 -122.142 -122.074 -122.247
 -122.497 -121.769 -121.827 -121.979 -121.871 -122.091 -121.754 -121.746
 -121.92  -121.992 -122.406 -121.359 -121.789 -121.707 -122.068 -122.404
 -122.334 -121.799 -121.774 -121.985 -121.865 -121.724 -122.415 -121.756
 -121.809 -122.135 -121.691 -122.038 -121.877 -121.94  -121.968 -121.988
 -121.315 -121.902 -122.514 -122.414 -121.883 -121.866 -121.744 -122.096
 -122.061 -121.881 -121.745 -122.461 -122.067 -121.868 -121.646 -121.93
 -122.105 -121.763 -121.718 -121.967 -121.777 -121.957 -121.823 -121.887
 -122.408 -122.462 -122.43  -122.456 -121.897 -121.932 -121.969 -121.916
 -122.081 -121.975 -121.735 -121.801 -121.761 -121.723 -121.924 -122.475
 -121.935 -122.407 -122.448 -122.453 -121.894 -121.936 -121.764 -122.416
 -121.905 -122.464 -121.768 -122.484 -121.738 -121.9   -121.82  -122.455
 -121.889 -122.496 -121.829 -122.505 -121.951 -121.847 -122.509 -121.961
 -121.417 -121.904 -122.503 -121.949 -121.874 -122.432 -121.971 -121.77
 -122.473 -121.896 -121.952 -122.254 -121.743 -121.933 -121.892 -121.749
 -121.473 -121.857 -122.465 -121.838 -121.954 -122.422 -121.931 -121.963
 -122.441 -121.925 -121.352 -122.511 -122.413 -121.876 -121.748 -121.818
 -121.8   -121.929 -121.698 -121.886 -121.802 -121.81  -121.762 -121.781
 -121.775 -122.44  -121.773 -121.819 -121.726 -122.459 -122.446 -121.855
 -121.736 -122.499 -122.46  -121.786 -122.421 -121.947 -122.439 -121.834
 -121.804 -122.443 -121.716 -121.848 -122.458 -122.515 -121.922 -121.953
 -121.783 -122.472 -121.944 -121.869 -121.828 -122.452 -121.831 -121.737
 -121.739 -121.863 -121.73  -121.856 -121.747 -121.893 -121.733 -121.846
 -121.821 -121.319 -121.765 -121.75  -122.506 -121.948 -121.921 -122.507
 -122.457 -121.914 -122.469 -121.792 -121.907 -121.841 -121.757 -121.788
 -121.731 -122.449 -121.316 -121.321 -122.504 -121.884 -121.803 -121.842
 -121.719 -121.766 -122.433 -122.519 -121.851 -121.402 -122.454 -122.467
 -121.325 -121.815 -121.676 -121.941 -122.445 -121.76  -121.885 -121.742
 -121.822 -121.895 -121.784 -121.701 -121.713 -121.727 -121.849 -121.835
 -122.435 -122.474 -122.444 -121.939 -121.48  -121.364 -121.767 -122.42
 -121.84  -122.425 -122.447 -121.797 -122.491 -121.917 -121.891 -121.942
 -121.862 -121.725 -121.873 -121.405 -122.486 -121.795 -121.734 -121.40
3]
sqft_living15 [1340 1690 2720 1360 1800 4760 2238 1650 1780 2390 2210 13
30 1370 2140
 1890 1610 1060 1280 1400 4110 2240 1220 2200 1030 1760 1860 1520 2630
 2580 1390 1460 1570 2020 1590 2160 1730 1290 2620 2470 2410 3625 1580
 3050 1228 2680  970 1190 1990 1410 1480 2730 1950 2250 2690 2960 2270
 2570 2500 1440 2750 2221 1010 3390 3530 1640 1510 2420 1940 3240 1680
  890 1130 3350 2350 1870 1720 1850 1900 1980 2520 1350 1750 1160 2550
 2370 1240 1270 2990 1380 1540 2090 2640 1830 1620 1880 2340 1710 2700
 3060 2660 1700 1970 1420 2060 2480 1550 1170 2820 1560 2230 2840 1450
 1500 3160 1200 3400 2110 2920 1770 1070 1930 3740 2260 1670 2290 1050
 2540 2190 2030 1230 2330 1300 1430 2770 1250 1630 2590 2130 1100 3836
 1320 2120 3070 1910 2080 1960 2280 1150 3430 2070 2600  830 1260 3120
 2010 1660 1600 2380 3890 4180 2653 2670 3920 2300 2310 2320 3150 1740
 2400 4550 2510 2440 2880 3860 2150 1310 1820 3080  880 2560 3470 1020
 2040 2610 1810 2860 3480 3130 3360 4050 2450 1790 3180 3600 2000 2430
```

```
 2850 4680 2360 3930 1490 2460 2077 1920 3630 3220 2100 3230 4300 3850
 2424 2530 3030 2830 2900 2950 1470  940 2740 4210 3340 3980 2180 3715
 2050 1080 2095 1000 3330 2170 1408 1530 2760 3110  950 3000 1307 2220
 4190 3440 3250 1110 2870 1210 2910 1120 4230 1708 3090 3270 2970 1180
 3100 4100 2930 3510 2688 1840 2490 4090 2810 3260 3680 3420 1654 1365
  980 1677 1140 3640 3460 3140 1502 3720 2790 2940  990 2890  860 4750
 1525 3950 5790  760 2234  960 3210 2780 2800 2305 2665 3620 2710 4320
 2650 3370 1509 1277 1981 2434 4640 2242 3040 3970 3200 4600  840 3290
 2214 1162 3010 5600 3820 3540 1975 4800  740 3990 3170 1576 1768 3310
 2980 1429 3900 3380  820 1090 4060 3910 3190 3450 3730  620 3020 3760
 3320 1132 3300 3770 3960  870 3560 4620 3520 1572 3490 1088 3159 4470
 3570 4890 3690 3280 2083 3780  920 1941 1566  850 2496 1040 3410 4240
 4670 4350 1714 5380 4330 3830 5000 2144 1494 1357  930 3580 4250 4080
 3660 1458 3736 1894 2037 1295 4170 3750 3550 4630 1439 3500 2091  900
 3880 3710 1616  720  800 2315 1564 2767 3721 4650 4020  780 1728 2027
 1264 1404 1459 2028 3639 1943 3425 2641 2114 1309 2412 2517 1802 2011
 1466 1414 3193 1845 1156 3670 1696 5340 4440 1745 1884 4690 4920 2406
 4160 3810 4480 2848 1746 2634 2049 5330 1536 2273 3056 4010 4700  910
 2125 1665 2683 3790  700 1855  750 1078 4150 4340 2344 1098 1175 1188
 3700 3840 4042 2518 3800 2488 3590 2052  810 1528 5030 4740 5070 2967
 4280 2724 3610 3940 4940 4770 1811 4830 2876 1805 1216 5170 1304 2474
 4590 4130 1492 1364 2168 4140 3543 1303 2005 3650 2583 4310 2451 1448
 2955 2142  790 1638 2554 2441 2216 4220 1961 4540  770 4200 3413 1664
 2136 3568 4510 1484 1358 2106 1834 2014 4390 4570 2175 6110 4260  710
 2112 1934 1518 1302 2622 2619 2382 4290 4560 4000 1336 3112 4070 1468
 1571 2605 1138 5110 4850 2165 4410 1678 5610 1984 4660 3870 4370  460
 4610 1914 3515 2246 1786 2109 2326 2728 4400 4950 1767 2054 5500 2555
 3674 2765 1862 1352 4030  399 2415 2901 1815 2236 2253 2004 1356 2403
 1137 1256 4930 4040 2376 4520 4490 2189 2566 2396 1282 2155 1056 2389
 2256 3618 1326 1168 4913  806 1369 2405 2875 1425 5220 1442 2333 3335
 1321 3045 1546 4730 2697 2822 2076 1757 4780  952 4270 2075 2667 1092
 1217 1716 1792 2961 1125 1463 1886  670 4460 2336 3557 5200 2258 1377
 2019 2092 4900 2615 1639 1765 1554 1381 4120 5080 1445 2793 2475  998
 2384 2575 1398 1584 2439 2197 2029 4362 1443 4420 1691 2495 2437 2547
 6210 2009 1847 1346 2578 2879 2255 2815 1608  690 2425 1481 2458 2358
 2056 1921 2419 2996 2502 1798 3087 1076 2981 2363 3191 1763 1876 1949
 2598 1979 1415 2002 2574 2166 3726 2099 2154 1522 1544 2912 2648 1658
 2755 2798 1405 2704 2738 3008 2586 2873 1232 2597 2516 1537 1128 2849
 1399 1131 1569 2381 1084 2304 4530 2297 2279 2303 2669 4225 2513 2725
 1955 2527 4443 2478 1919 1813 2533  828 2015 3078 4495 2673 2316 2647
 3402 3494 2156 3236 2612 2323 2409 2354 1285 2616 1427 1516 2456 2844
 1495 2594 2604 1268 2198 3038 2927]
sqft_lot15 [5650 7639 8062 ... 5731 1509 2007]
sales_year [2014 2015]
sales_month [10 12  2  5  6  1  4  3  7  8 11  9]
```

In [15]:

```
# test 1/6 for 1b
# This test is an example. The rest tests are hidden but are the same as this one checking each answer.
assert price == 'numeric', "Check 1b. What is the correct variable type for price?"
```

In [16]:

```
# test 2/6 for 1b
```

In [17]:

```
# test 3/6 for 1b
```

In [18]:

```
# test 4/6 for 1b
```

In [19]:
```
# test 5/6 for 1b
```

In [20]:
```
# test 6/6 for 1b
```

1c) Drop features [5 pts]

Let's drop features that are unnecessary. `id` is not a meaningful feature. `date` string has been coded to `sales_month` and `sales_year` , so we can remove `date` . `zipcode` can be also removed as it's hard to include in a linear regression model and the location info is included in the `lat` and `long` . Drop the features `id` , `date` , and `zipcode` and replace the df.

> 💡***Tip:*** .drop() function can drop one or more columns or rows. Learn how to use it in the [doc (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html)](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html)

In [21]:
```
# drop unnecessary features, replace df
# Drop unnecessary features and replace df
df = df.drop(['id', 'date', 'zipcode'], axis=1)

#df.head()
```

In [22]:
```
# tests that you droppd the features id, date, and zipcode from df
```

2. More inspection; Correlation and pair plot [5 pts]

2a) Get correlation matrix on the data frame. [5 pts]

Which feature may be the best predictor of price based on the correlation? Answer as a string value (e.g. best_guess_predictor = 'price' or best_guess_predictor = 'yr_built')

> 💡***Tip:*** .corr() finction can show correlation matrix from the dataframe. [More resource (https://www.geeksforgeeks.org/python-pandas-dataframe-corr/#)](https://www.geeksforgeeks.org/python-pandas-dataframe-corr/#)

Self-check Concepts

✔️ By looking at the correlation matrix, how do you decide which feature is the best predictor?

In [23]:
```python
df.corr()
```

Out[23]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
price	1.000000	0.308350	0.525138	0.702035	0.089661	0.256794	0.2663
bedrooms	0.308350	1.000000	0.515884	0.576671	0.031703	0.175429	-0.0065
bathrooms	0.525138	0.515884	1.000000	0.754665	0.087740	0.500653	0.0637
sqft_living	0.702035	0.576671	0.754665	1.000000	0.172826	0.353949	0.1038
sqft_lot	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005201	0.0216
floors	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000000	0.0236
waterfront	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023698	1.0000
view	0.397293	0.079532	0.187737	0.284611	0.074710	0.029444	0.4018
condition	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.0166
grade	0.667434	0.356967	0.664983	0.762704	0.113621	0.458183	0.0827
sqft_above	0.605567	0.477600	0.685342	0.876597	0.183512	0.523885	0.0720
sqft_basement	0.323816	0.303093	0.283770	0.435043	0.015286	-0.245705	0.0805
yr_built	0.054012	0.154178	0.506019	0.318049	0.053080	0.489319	-0.0261
yr_renovated	0.126434	0.018841	0.050739	0.055363	0.007644	0.006338	0.0928
lat	0.307003	-0.008931	0.024573	0.052529	-0.085683	0.049614	-0.0142
long	0.021626	0.129473	0.223042	0.240223	0.229521	0.125419	-0.0419
sqft_living15	0.585379	0.391638	0.568634	0.756420	0.144608	0.279885	0.0864
sqft_lot15	0.082447	0.029244	0.087175	0.183286	0.718557	-0.011269	0.0307
sales_year	0.003576	-0.009838	-0.026596	-0.029038	0.005468	-0.022315	-0.0041
sales_month	-0.010081	-0.001533	0.007392	0.011810	-0.002369	0.014005	0.0081

In [24]:
```python
# your code here

correlation_matrix = df.corr()

# Find the feature with the highest correlation to 'price'
best_guess_predictor = correlation_matrix['price'].drop('price').idxmax()
best_guess_predictor
```

Out[24]: 'sqft_living'

In [25]:
```python
# tests the solution for best_guess_predictor
```

2b) Display the correlation matrix as heat map [Not graded]

`seaborn.heatmap()` (https://seaborn.pydata.org/generated/seaborn.heatmap.html) can visualize a matrix as a heatmap. Visualize the correlation matrix using seaborn.heatmap(). Play with color map, text font size, decimals, text orientation etc. For example, the resulting display may look like below. If you find how to make a pretty visualization, please share in the discussion board.

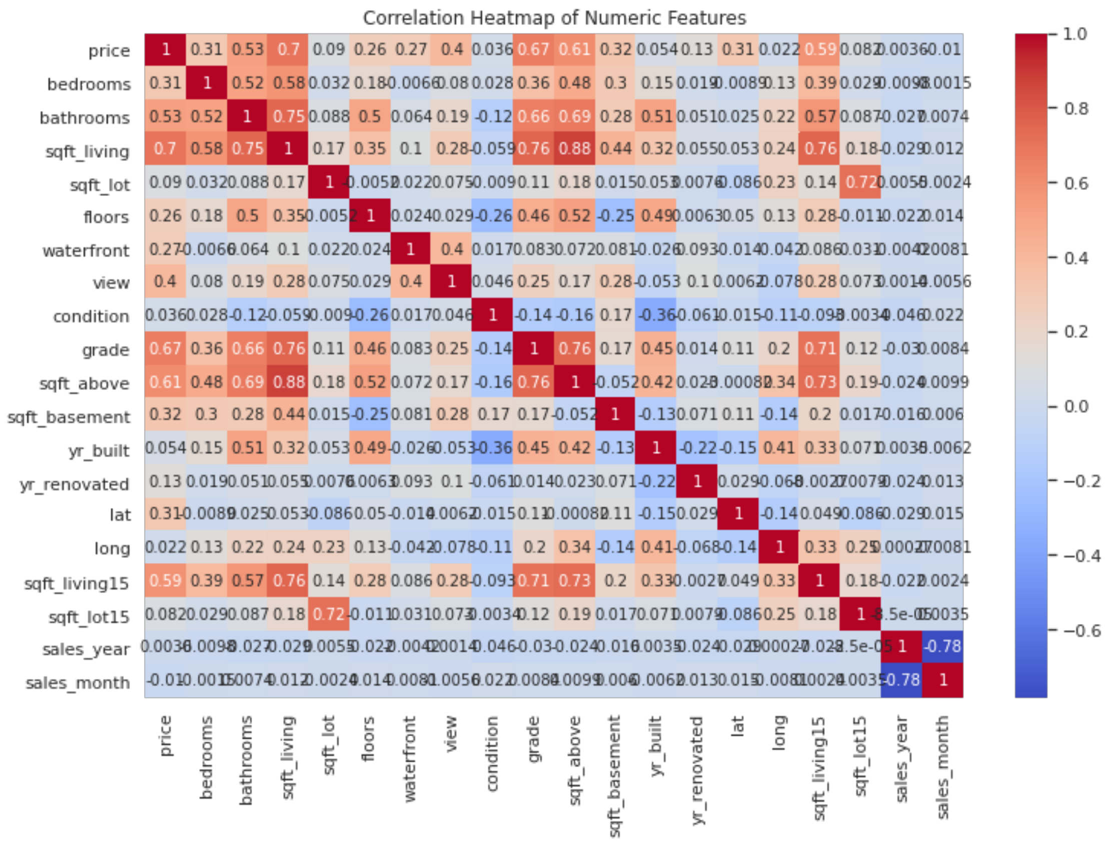
correlation matrix

Note: your code for this section may cause the Validate button to time out. If you want to run the Validate button prior to submitting, you could comment out the code in this section.

In [26]:
```
# practice visualizing correlation matrix using a heatmap
# Generate the heatmap plot
plt.figure(figsize=(12, 8))
heatmap_plot = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Numeric Features')

# Save the heatmap plot to a file using the Figure object
heatmap_plot.figure.savefig('heatmap_plot.png')

plt.show()
```



2c) Pair plot [Not graded]

Pair plot is a fast way to inspect relationships between features. Use seaborn's .pairplot() function to draw a pairplot if the first 10 columns (including price) and inspect their relationships. Set the diagonal elements to be KDE plot. The resulting plot will look like below.
pair plot

Note: your code for this section may cause the Validate button to time out. If you want to run the Validate button prior to submitting, you could comment out the code in this section.

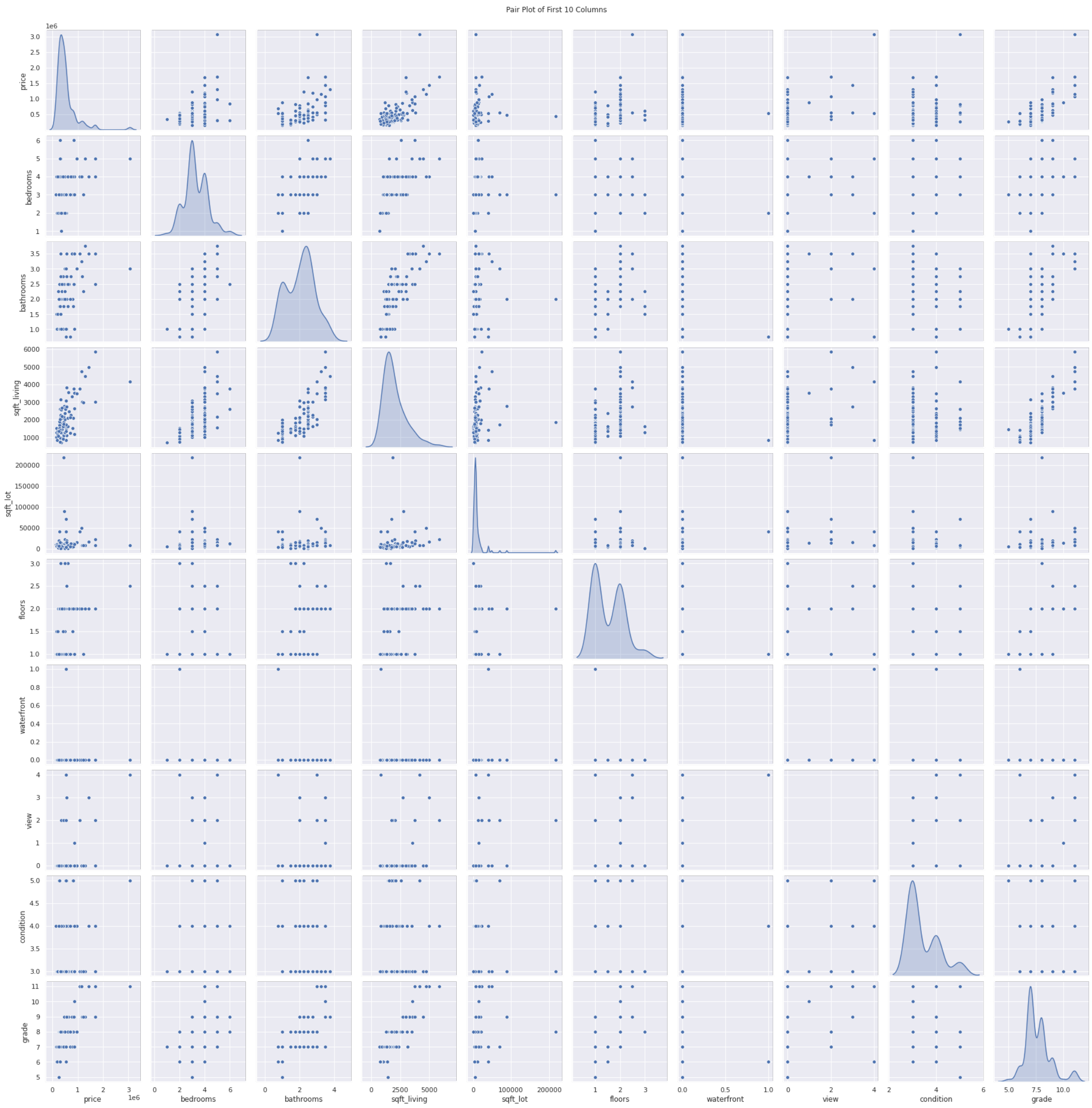
In [27]:

```
# practice inspecting relationships between features using a pair plot.
# Plot pairplot of the first 10 columns
# Select the first 10 columns
columns_to_plot = df.columns[:10]

# Draw the pair plot
pair_plot = sns.pairplot(df[columns_to_plot].sample(100), diag_kind='kd
e')
pair_plot.fig.suptitle('Pair Plot of First 10 Columns', y=1.02)

# Save the pair plot to a file using the Figure object
pair_plot.savefig('pair_plot.png')

# Show the pair plot
plt.show()
```



3. Simple linear regression [20 pts]

3a) Data preparation [5 pts]

We will split the data to train and test datasets such that the test dataset is 20% of original data. Use `sklearn.model_selection.train_test_split` function to split the data frame to X_train and X_test. X_train is 80% of observation randomly chosen. X_test is the rest 20%. Both X_train and X_test are `pd.DataFrame` object and include 'price' in the table. Note that the train_test_split can handle data frame as well as array.

> 💡 ***Tip:*** Use sklearn.model_selecttion.train_test_split to split the data frame. We would like X_train to be 80% of the observation and X_test to be 20% of the observations. Print length of X_train and X_test.

In [28]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test = train_test_split(df, test_size=0.2, random_state=42)

X_train.shape, X_test.shape
```

Out[28]:
```
((17290, 20), (4323, 20))
```

In [29]:
```
# Testing cell for self-check
assert(len(X_train) == 17290), "Check 3a, did you split properly so X_Tra
in is 80% of the observations?"
assert(type(X_train)==type(pd.DataFrame())), "Check 3a, what type of obje
ct should X_train be?"
```

In [30]:
```
# Testing cell
```

In [31]:
```
# Testing cell
```

3b) Train a simple linear regression model [5 pts]

Use the best_guess_predictor as a single predictor and build a simple linear regression model using **`statsmodels.formula.api.ols`** function (https://www.statsmodels.org/dev/example_formulas.html [(https://www.statsmodels.org/dev/example_formulas.html)](https://www.statsmodels.org/dev/example_formulas.html)) Print out the result summary. Train on the X_train portion. What is the adjusted R-squared value?

> 💡***Tip:*** We had imported the library at the top of this notebook. So you can use the `smf` alias.
>
> ```
> import statsmodels.formula.api as smf
> ```

N.B.: It recommended that you use the statsmodel library to do the regression analysis as opposed to e.g. sklearn. The sklearn library is great for advanced topics, but it's easier to get lost in a sea of details and it's not needed for these problems.

In [32]:
```
# use best_guess_predictor as a single predictor
# build a simple linear regression model, train on the X_train portion

# Make sure to use the `statsmodels.formula.api.ols` function for buildin
g the model.
# model =

#update following value according to the result
# adj_R2 =

# your code here
import statsmodels.formula.api as smf

# Build the simple linear regression model using the best guess predictor
'sqft_living'
model = smf.ols(formula='price ~ sqft_living', data=X_train).fit()

# Print the model summary
print(model.summary())

# Extract the adjusted R-squared value from the model summary
adj_R2 = model.rsquared_adj
adj_R2
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.492
Model:                            OLS   Adj. R-squared:                  0.492
Method:                 Least Squares   F-statistic:                 1.677e+04
Date:                Thu, 15 Aug 2024   Prob (F-statistic):               0.00
Time:                        20:34:57   Log-Likelihood:            -2.3995e+05
No. Observations:               17290   AIC:                         4.799e+05
Df Residuals:                   17288   BIC:                         4.799e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept     -4.2e+04   4886.778     -8.594      0.000   -5.16e+04   -3.24e+04
sqft_living   279.5548      2.159    129.496      0.000     275.323     283.786
==============================================================================
Omnibus:                    11990.495   Durbin-Watson:                   2.030
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           483410.340
Skew:                           2.835   Prob(JB):                         0.00
Kurtosis:                      28.276   Cond. No.                     5.65e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.65e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Out[32]: 0.4923544744403926

In [33]:
```
# self test
assert len(model.params.index) == 2, 'Check 3b, Number of model parameters (including intercept) does not match. Did you make a univariate model?'
```

In [34]:
```
# hidden test for 3b
```

In [35]:
```
# hidden test for 3b
```

3c) Best predictor [10 pts]

In question 5a, we picked a best guess predictor for price based on the correlation matrix. Now we will consider whether the best_guess_predictor that we used is still the best.
Print out a list ranking all of the predictors. Then print out a list of the top three predictors in order.

Hint: Linear regression uses adjusted R squared as fit performance. So you can rank by this metric.

- What were your top three predictors?
- How did you order your list of predictors to select those as the top ones?
- Is your top predictor for this section the same as the best guess predictor you selected in question 2a?

In [36]:

```
# your code here

# uncomment and update top_three
# top_three = []

# Get list of all predictors excluding 'price'
predictors = df.columns.drop('price')

# Dictionary to store adjusted R-squared values for each predictor
adj_r2_values = {}

# Iterate through each predictor and fit a linear regression model
for predictor in predictors:
    formula = f'price ~ {predictor}'
    model = smf.ols(formula=formula, data=X_train).fit()
    adj_r2_values[predictor] = model.rsquared_adj

# Sort the predictors based on adjusted R-squared values
sorted_predictors = sorted(adj_r2_values.items(), key=lambda x: x[1], rev
erse=True)

# Print the list ranking all of the predictors
print("Ranking of all predictors based on adjusted R-squared values:")
for predictor, adj_r2 in sorted_predictors:
    print(f"{predictor}: {adj_r2}")

# Extract the top three predictors
top_three = [predictor for predictor, adj_r2 in sorted_predictors[:3]]

# Print the top three predictors
print("\nTop three predictors in order:")
for predictor in top_three:
    print(predictor)
```

```
Ranking of all predictors based on adjusted R-squared values:
sqft_living: 0.4923544744403926
grade: 0.4423169630047675
sqft_above: 0.3638799586966397
sqft_living15: 0.3394626876316138
bathrooms: 0.2772844247965518
view: 0.15369966864561202
sqft_basement: 0.10323792396142162
lat: 0.0965258412680926
bedrooms: 0.09497323480282405
floors: 0.06417008043783023
waterfront: 0.06392747342978922
yr_renovated: 0.01625994373923545
sqft_lot: 0.008230737755807738
sqft_lot15: 0.006207612669632101
yr_built: 0.002353057270307324
condition: 0.0012630921111194127
long: 0.0004937714263635318
sales_month: 0.0001251798499750656
sales_year: 6.223098183821829e-05

Top three predictors in order:
sqft_living
grade
sqft_above
```

In [37]:
```
# self test cell
assert(type(top_three) == list), "Check 3c, the top_three needs to be a l
ist."
assert(len(top_three) == 3), "Check 3c, the top_three list needs to have
three element."
```

In [38]:
```
# test cell
```

In [39]:
```
# test cell
```

In [40]:
```
# test cell
```

In [41]:
```
# test cell
```

In [42]:
```
# test cell
```

In [43]:
```
# test cell
```

In [ ]:
```

```