EDA

Here we are gonna dig inside our data and see if we find missing values, wrong data types etc...

Table of Contents

Step 1: the shape of data

In [1]:
```
import pandas as pd
df = pd.read_csv('kc_house_data.csv')
df.head()
```

Out[1]:

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

5 rows × 21 columns

In [2]:
```
df.shape
```

Out[2]: (21597, 21)

Conclusion

All looks good

Step 2: Analyzing datatypes

Check Column naming

In [3]:
```
df.columns
```

Out[3]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grad
e',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipc
ode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

Notes Column name check - need to some of the columns have different names in column readme file, bedrooms vs bedroomsNumber, bathrooms vs bathroomsNumber, etc...

Check data types

In [4]:
```
df.dtypes
```

Out[4]:
```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront       float64
view             float64
condition          int64
grade              int64
sqft_above         int64
sqft_basement     object
yr_built           int64
yr_renovated     float64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [5]:
```
# date column needs a to_datetime (is currently string)
# bathrooms 2.25 doesn't make sense, 2.5 does - according to lindsey, 1/
4  and 3/4 baths are a thing
# floors should be int not float
# waterfront is a float, should be bool
# what do the ints from 3 to 5 mean in terms of condition? 3=good? 5-ba
d? or vice versa? condition affects price?
# same for grade, get more info, info affects price?
# sqfoot above mean above ground (not including basement)
# sqfoot basement may have impact on price, but not as much as above gro
und sq footage (is float, needs to be int).
# We can see that it is a object where it should be an integer like the
 other surface measures.
# yrbuilt - datetime? or in int ok?
# yrrenovated shouldn't be a float
# sqft living & lot based on 15 nearest neighbors
```

In [6]:
```
# sqft_basement has a '?' string as its Null value
sqft_basement = df['sqft_basement']
pd.to_numeric(sqft_basement)
```

```
-----------------------------------------------------------------------
----
ValueError                                Traceback (most recent call l
ast)
pandas/_libs/lib.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "?"

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call l
ast)
<ipython-input-6-4d05284adccc> in <module>
      1 # sqft_basement check
      2 sqft_basement = df['sqft_basement']
----> 3 pd.to_numeric(sqft_basement)

~/anaconda3/lib/python3.7/site-packages/pandas/core/tools/numeric.py in
to_numeric(arg, errors, downcast)
    133             coerce_numeric = False if errors in ('ignore', 'rai
se') else True
    134             values = lib.maybe_convert_numeric(values, set(),
--> 135                                                coerce_numeric=c
oerce_numeric)
    136 
    137     except Exception:

pandas/_libs/lib.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "?" at position 6
```

In [7]:
```
# How many empty sqft_basement measures do we have?
len(df.loc[df['sqft_basement'] == '?'])
```

Out[7]:
```
454
```

In [8]:
```
# Defines most commonly found value for sqft_basement
sqft_basement.value_counts().head()
```

Out[8]:
```
0.0      12826
?          454
600.0      217
500.0      209
700.0      208
Name: sqft_basement, dtype: int64
```

Conclusion : Let's assume that those with unknown sqft_basement are actually with no basement and set to 0

In [ ]:
```
# Yr renovated
yr_renovated = df['yr_renovated']
yr_renovated = pd.to_numeric(yr_renovated)
```

In [ ]:
```
yr_renovated.isna().sum()
```

In [ ]:
```
yr_renovated.isnull().sum()
```

In [ ]:
```
# Defines most commonly found value for yr_renovated
yr_renovated.value_counts().head()
```

Conclusion As there are lot of values that can't be ignored in yr_renovated, let's set the NaN values to the yr_built or assumes that they have never been renovated. We will perfom the datatype change to datetime after.

Step 3 : Check for duplicated entries

In [9]:
```
# Checking for duplicated entries in IDs.
df.loc[df.duplicated(['id'], keep=False)].sort_values(by=['id'], ascendi
ng=True)
```

Out[9]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
2495	1000102	4/22/2015	300000.0	6	3.00	2400	9373	2.0	
2494	1000102	9/16/2014	280000.0	6	3.00	2400	9373	2.0	
16800	7200179	10/16/2014	150000.0	2	1.00	840	12750	1.0	
16801	7200179	4/24/2015	175000.0	2	1.00	840	12750	1.0	
11422	109200390	10/20/2014	250000.0	3	1.75	1480	3900	1.0	
11421	109200390	8/20/2014	245000.0	3	1.75	1480	3900	1.0	
12406	123039336	12/8/2014	244900.0	1	1.00	620	8261	1.0	
12405	123039336	6/11/2014	148000.0	1	1.00	620	8261	1.0	
7786	251300110	1/14/2015	358000.0	3	2.25	2510	12013	2.0	
7785	251300110	7/31/2014	225000.0	3	2.25	2510	12013	2.0	
9225	302000375	8/14/2014	169100.0	3	2.00	1050	18304	1.0	
9226	302000375	5/6/2015	250000.0	3	2.00	1050	18304	1.0	
14842	324000530	3/23/2015	459000.0	3	1.00	1320	5000	1.5	
14841	324000530	7/8/2014	201500.0	3	1.00	1320	5000	1.5	
7171	526059224	9/23/2014	260000.0	4	1.75	1650	7276	1.0	
7172	526059224	2/6/2015	470000.0	4	1.75	1650	7276	1.0	
17367	641900050	8/19/2014	335000.0	4	2.25	2160	8817	1.0	
17368	641900050	2/6/2015	499950.0	4	2.25	2160	8817	1.0	
19536	643300040	11/4/2014	481000.0	4	1.75	1920	9500	1.0	
19537	643300040	3/13/2015	719521.0	4	1.75	1920	9500	1.0	
15286	705730280	4/21/2015	335000.0	3	2.50	1740	5267	2.0	
15285	705730280	8/19/2014	325000.0	3	2.50	1740	5267	2.0	
9266	722039087	9/23/2014	220500.0	2	1.00	990	57499	1.0	
9267	722039087	5/4/2015	329000.0	2	1.00	990	57499	1.0	
3781	723049156	5/23/2014	149000.0	3	1.00	1700	8645	1.0	
3782	723049156	11/12/2014	284700.0	3	1.00	1700	8645	1.0	
823	726049190	10/2/2014	287500.0	3	1.00	1810	7200	1.0	
824	726049190	2/18/2015	431000.0	3	1.00	1810	7200	1.0	
17588	795000620	9/24/2014	115000.0	3	1.00	1080	6250	1.0	
17589	795000620	12/15/2014	124000.0	3	1.00	1080	6250	1.0	
...	...	...	...	...	...	...	...	...	
717	8820903380	7/28/2014	452000.0	6	2.25	2660	13579	2.0	
718	8820903380	1/2/2015	730000.0	6	2.25	2660	13579	2.0	
6428	8832900780	10/13/2014	480000.0	5	2.00	1760	21562	1.0	

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
6429	8832900780	4/8/2015	647500.0	5	2.00	1760	21562	1.0	
12907	8910500150	5/29/2014	329932.0	3	1.50	1460	5040	1.0	
12908	8910500150	1/20/2015	539000.0	3	1.50	1460	5040	1.0	
10215	8945100320	5/6/2014	136500.0	3	1.50	1420	8580	1.0	
10216	8945100320	10/8/2014	224097.0	3	1.50	1420	8580	1.0	
2974	9136103130	12/1/2014	430000.0	2	1.50	1090	4013	1.5	
2975	9136103130	5/12/2015	685000.0	2	1.50	1090	4013	1.5	
13010	9211500620	10/8/2014	182700.0	3	2.25	1740	6650	1.0	
13011	9211500620	4/28/2015	305000.0	3	2.25	1740	6650	1.0	
6366	9222400605	4/11/2015	850000.0	5	4.00	2980	4500	1.5	
6365	9222400605	11/15/2014	842500.0	5	4.00	2980	4500	1.5	
12820	9238500040	2/10/2015	599000.0	3	2.50	2970	23100	1.0	
12819	9238500040	6/24/2014	400000.0	3	2.50	2970	23100	1.0	
16658	9250900104	4/10/2015	496000.0	5	1.75	2110	8500	1.0	
16657	9250900104	11/10/2014	300000.0	5	1.75	2110	8500	1.0	
4338	9353300600	6/24/2014	348500.0	3	1.50	1360	10726	1.0	
4339	9353300600	3/26/2015	370000.0	3	1.50	1360	10726	1.0	
2492	9407110710	2/26/2015	322000.0	3	1.75	1510	8400	1.0	
2491	9407110710	11/7/2014	195000.0	3	1.75	1510	8400	1.0	
4918	9809000020	3/13/2015	1940000.0	5	2.25	3120	16672	2.0	
4917	9809000020	5/13/2014	1900000.0	5	2.25	3120	16672	2.0	
6340	9828200460	1/6/2015	430000.0	2	1.00	700	4800	1.0	
6339	9828200460	6/27/2014	260000.0	2	1.00	700	4800	1.0	
15186	9834200305	2/10/2015	615000.0	3	1.00	1790	3876	1.5	
15185	9834200305	7/16/2014	350000.0	3	1.00	1790	3876	1.5	
1084	9834200885	7/17/2014	360000.0	4	2.50	2080	4080	1.0	
1085	9834200885	4/20/2015	550000.0	4	2.50	2080	4080	1.0	

353 rows × 21 columns

Conclusion : It seems like the properties listed have been sold multiple times during the period. It might be helpful to keep the information and not drop the duplicated values has they happen in different times.

Step 4: Null values

In [10]:
```
df.describe()
```

Out[10]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	215
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	

In [ ]:
```
# count for boolean values, can give us a better idea of distribution
# max of floors is 3.5, why .5 floors? is that the attic?
# std normalize scale factors? compare?
```

In [11]:
```
df.isna().sum()
```

Out[11]:
```
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       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

Check waterfront and view missing values

In [ ]:
```
# Check different values for waterfront
waterfront_ds = df['waterfront']
print(waterfront_ds.value_counts())
print('Waterfront - Total of missing values : ', waterfront_ds.isna().su
m())
```

In [ ]:
```
# Check different values for view
view_ds = df['view']
print(view_ds.value_counts())
print('view - Total of missing values : ', view_ds.isna().sum())
```

Conclusion As described earlier, we already identified missing data under yr_renovated and sqft_basement when checking datatype. Now we see also that the dataset contains missing data for waterfront and view. As the number of NaN for "waterfront" is huge we gonna keep the entries and replace the NaN by the most common value 0. The same strategy is used for "view" variable

In [ ]:
```
# Location is clearly important in real-estate, so we'd like to look at
 zip code and latitude/longitude,
# but to avoid multicollinearity we don't want to select more than one v
ariable.
# We will make separate models for these two measures of location and se
e which is a better predictor of value.
# The third question we'd like to pose is how well the square footage me
trics predict the price of homes.
```