DATA Cleaning

Now that we have a better understanding of our data set, let's try to get a cleaner data set.

Table of Contents

Step 1: Dealing with NaN values

We noticed during our EDA that specific data variables contain NaN values :

- yr_renovated
- sqft_basement
- view
- waterfront

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

Out[1]:

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

5 rows × 21 columns

Year Renovated

In [2]:
```
# Fill NaN Year Renovated with Yr Built
cleaned_data = data.copy()
cleaned_data['yr_renovated'] = cleaned_data['yr_renovated'].fillna(cleaned_
```

In [3]:
```
# Fill 0 Year renovated with Yr Built
cleaned_data.loc[cleaned_data['yr_renovated'] == 0, 'yr_renovated'] = clean
cleaned_data.head()
```

Out[3]:

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

5 rows × 21 columns

In [4]:
```
cleaned_data.shape
```

Out[4]:
```
(21597, 21)
```

Sqft Basement

In [5]:
```
# Remove non numerical values and set them to null
print('sqft_basement - Count non numeric values : ', len(cleaned_data.loc[c
cleaned_data.loc[cleaned_data['sqft_basement'] == '?', 'sqft_basement'] = 0
cleaned_data.head()
```

```
sqft_basement - Count non numeric values :  454
```

Out[5]:

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

5 rows × 21 columns

View

In [6]:
```
# Check different values for view
print(cleaned_data['view'].value_counts())
print('view - Total of missing values : ', cleaned_data['view'].isna().sum(
```

```
0.0    19422
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
view - Total of missing values :  63
```

In [7]:
```
# Set missing values to null
cleaned_data['view'].fillna(0, inplace=True)
cleaned_data.isna().sum()
```

Out[7]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
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
```

Waterfront

In [8]:
```
# Check different values for waterfront
print(cleaned_data['waterfront'].value_counts())
print('Waterfront - Total of missing values : ', cleaned_data['waterfront']
```

```
0.0    19075
1.0      146
Name: waterfront, dtype: int64
Waterfront - Total of missing values :  2376
```

In [9]:
```
# Set missing values to null
cleaned_data['waterfront'].fillna(0, inplace=True)
cleaned_data.isna().sum()
```

Out[9]:
```
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
```

Conclusion During the NaN/Null values analysis we made the choice of keeping as max as possible entries given values. We did it by assigning default value to the most commonly value in dataset (or mode value). Next step, now that we have cleaned our missing data, let's fix the datatypes.

Step 2: Converting data types

As shown previously, we have deal with NaN & null values. It is time to convert our dataset with correct datatype.

Looking back to our EDA, we need to convert the following data to the correct datatype:

- date to datetime
- waterfront to bool
- yr_built to datetime
- yr_renovated to datetime

Transforming date, yr_built, yr_renovated to datetime

In [10]:
```
# Transform date to datetime using pd.to_datetime()
cleaned_data['date'] = pd.to_datetime(cleaned_data['date'], format='%m/%d/%
cleaned_data.dtypes
```

Out[10]:
```
id                        int64
date             datetime64[ns]
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 [11]:
```
# Transform yr_built to datetime
cleaned_data['yr_built'] = pd.to_datetime(cleaned_data['yr_built'], format=
cleaned_data.dtypes
```

Out[11]:
```
id                        int64
date             datetime64[ns]
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         datetime64[ns]
yr_renovated            float64
zipcode                   int64
lat                     float64
long                    float64
sqft_living15             int64
sqft_lot15                int64
dtype: object
```

In [12]:
```
# Transform yr_renovated to datetime in 2 steps. First converting into int
cleaned_data['yr_renovated'] = cleaned_data['yr_renovated'].apply(lambda x:
cleaned_data['yr_renovated'] = pd.to_datetime(cleaned_data['yr_renovated'],
cleaned_data.dtypes
```

Out[12]:
```
id                        int64
date             datetime64[ns]
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         datetime64[ns]
yr_renovated     datetime64[ns]
zipcode                   int64
lat                     float64
long                    float64
```

Transforming waterfront to boolean

In [13]:
```
# Check waterfront values
cleaned_data['waterfront'] = cleaned_data['waterfront'].astype(bool)
cleaned_data.dtypes
```

Out[13]:
```
id                        int64
date             datetime64[ns]
price                   float64
bedrooms                  int64
bathrooms               float64
sqft_living               int64
sqft_lot                  int64
floors                  float64
waterfront                 bool
view                    float64
condition                 int64
grade                     int64
sqft_above                int64
sqft_basement            object
yr_built         datetime64[ns]
yr_renovated     datetime64[ns]
zipcode                   int64
lat                     float64
long                    float64
```

Step 3: Categorical variables

Do we have categorical variables ? We have some insights from our EDA about which are the right candidates but let's dig in.

In [14]:
```
# Spotting Categorical Variables
cleaned_data.describe()
```

Out[14]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.

At first glance, we can already spot some categorical variables : condition, grade, bedrooms, bathrooms, floors and maybe view / zipcode

In [15]:
```
# Check uniqueness
categorical_variables = ['waterfront', 'condition', 'grade', 'bedrooms', 'b

for categorical_variable in categorical_variables:
    print('{} unique values : '. format(categorical_variable), cleaned_data
```

```
waterfront unique values :  2
condition unique values :  5
grade unique values :  11
bedrooms unique values :  12
bathrooms unique values :  29
floors unique values :  6
view unique values :  5
zipcode unique values :  70
```

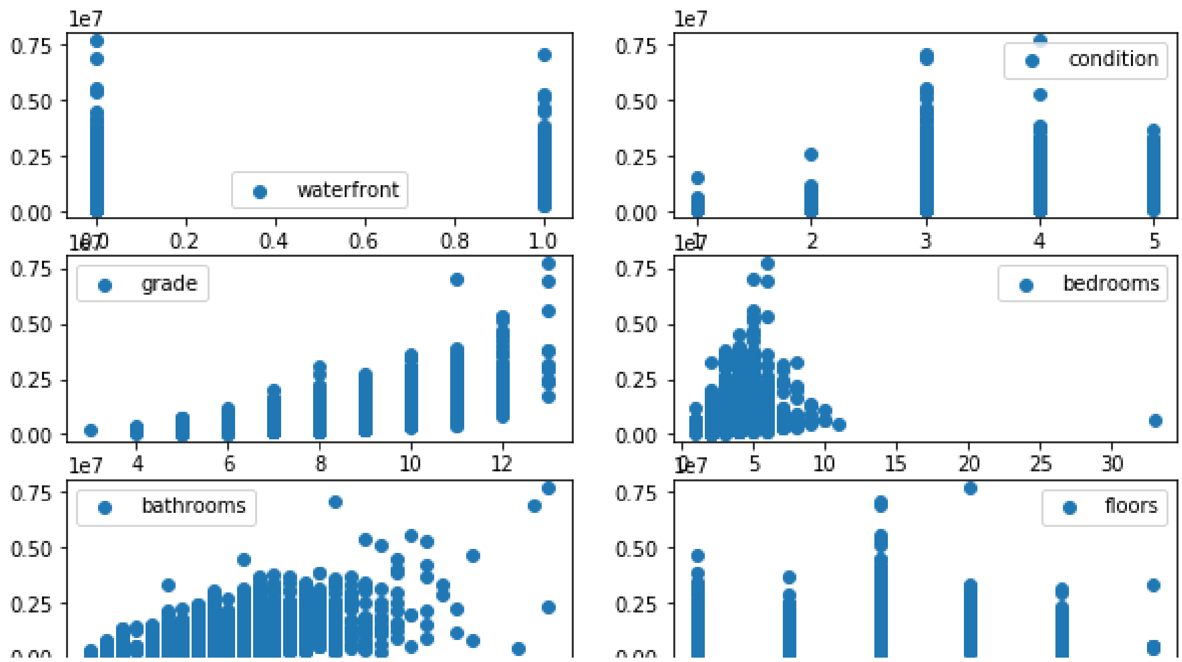
We identify that the suspected categorical variables have a limited range. So there is a good chance that they actually are.

Let's plot the price against those variables if we can see any pattern.

In [16]:
```
#
import matplotlib.pyplot as plt
%matplotlib inline

num_lines = len(categorical_variables)//2 + 1
fig = plt.figure(figsize=(10, 10))

for index, xcol in enumerate(categorical_variables):
    plt.subplot(num_lines, 2, index + 1)
    plt.scatter(data=cleaned_data, x=xcol, y='price', label=xcol);
    plt.legend();
```



Conclusion : Plotting them allow us to clearly indentifty categorical data and especially for the zipcode when you look at zipcode > 98150

In [17]:
```
# Transform  Categorical Variables using one-hot-encoding
categorical_variables = ['waterfront', 'condition', 'grade', 'bedrooms', 'b

water_dummies = pd.get_dummies(cleaned_data['waterfront'], prefix="water")
cond_dummies = pd.get_dummies(cleaned_data['condition'], prefix="cond")
grade_dummies = pd.get_dummies(cleaned_data['grade'], prefix="grade")
bed_dummies = pd.get_dummies(cleaned_data['bedrooms'], prefix='bed')
bath_dummies = pd.get_dummies(cleaned_data['bathrooms'], prefix='bath')
floor_dummies = pd.get_dummies(cleaned_data['floors'], prefix='floor')
view_dummies = pd.get_dummies(cleaned_data['view'], prefix='view')
zipcode_dummies = pd.get_dummies(cleaned_data['zipcode'], prefix='zip')
```

In [18]:
```
cleaned_data = cleaned_data.drop(categorical_variables, axis=1)

cleaned_data = pd.concat([
                    cleaned_data,
                    water_dummies,
                    cond_dummies,
                    grade_dummies,
                    bed_dummies,
                    bath_dummies,
                    floor_dummies,
                    view_dummies,
                    zipcode_dummies],
                    axis=1)
cleaned_data.head()
```

Out[18]:

	id	date	price	sqft_living	sqft_lot	sqft_above	sqft_basement	yr_built	yr_renovat
0	7129300520	2014-10-13	221900.0	1180	5650	1180	0.0	1955-01-01	1955-01-
1	6414100192	2014-12-09	538000.0	2570	7242	2170	400.0	1951-01-01	1991-01-
2	5631500400	2015-02-25	180000.0	770	10000	770	0.0	1933-01-01	1933-01-
3	2487200875	2014-12-09	604000.0	1960	5000	1050	910.0	1965-01-01	1965-01-
4	1954400510	2015-02-18	510000.0	1680	8080	1680	0.0	1987-01-01	1987-01-

5 rows × 153 columns

Step 4: Save our cleaned dataset for reusability

Let's save our cleaned dataset to a distinct csv file

In [19]:
```
# Save clean dataset to cleaned_kc_house_data.csv file
cleaned_data.to_csv('cleaned_kc_house_data.csv', index=False)

# test loading the cleaned dataset file and see if we kept our data cleanir
tested_dataset = pd.read_csv('cleaned_kc_house_data.csv')
tested_dataset.head()
```

Out[19]:

	id	date	price	sqft_living	sqft_lot	sqft_above	sqft_basement	yr_built	yr_renovat
0	7129300520	2014-10-13	221900.0	1180	5650	1180	0.0	1955-01-01	1955-01-
1	6414100192	2014-12-09	538000.0	2570	7242	2170	400.0	1951-01-01	1991-01-
2	5631500400	2015-02-25	180000.0	770	10000	770	0.0	1933-01-01	1933-01-
3	2487200875	2014-12-09	604000.0	1960	5000	1050	910.0	1965-01-01	1965-01-
4	1954400510	2015-02-18	510000.0	1680	8080	1680	0.0	1987-01-01	1987-01-

5 rows × 153 columns

In [20]:
```
# Check datatypes
tested_dataset.dtypes
```

Out[20]:
```
id                 int64
date              object
price            float64
sqft_living        int64
sqft_lot           int64
sqft_above         int64
sqft_basement    float64
yr_built          object
yr_renovated      object
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
water_False        int64
water_True         int64
cond_1             int64
cond_2             int64
cond_3             int64
cond_4             int64
```

In [21]:
```
# We can see that the datetime format is not kept. Let's fix it
date_columns = ['date', 'yr_built', 'yr_renovated']

for date_column in date_columns:
    tested_dataset[date_column] = pd.to_datetime(tested_dataset[date_column

tested_dataset.dtypes
```

Out[21]:
```
id                          int64
date               datetime64[ns]
price                     float64
sqft_living                 int64
sqft_lot                    int64
sqft_above                  int64
sqft_basement             float64
yr_built           datetime64[ns]
yr_renovated       datetime64[ns]
lat                       float64
long                      float64
sqft_living15               int64
sqft_lot15                  int64
water_False                 int64
water_True                  int64
cond_1                      int64
cond_2                      int64
cond_3                      int64
cond_4                      int64
```

In [26]:
```
tested_dataset.columns
```

Out[26]:
```
Index(['id', 'date', 'price', 'sqft_living', 'sqft_lot', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'lat',
       ...
       'zip_98146', 'zip_98148', 'zip_98155', 'zip_98166', 'zip_98168',
       'zip_98177', 'zip_98178', 'zip_98188', 'zip_98198', 'zip_98199'],
      dtype='object', length=153)
```

Conclusion

All looks good. We managed to clean our data and implement a way to reuse it once we will work on regression.