Predicting House Prices in King County, WA

Problem Statement:

This is a project for a real estate agency that helps homeowners sell their homes. I have been tasked to analyze King County housing data set to provide the agency with advice about which home renovations might increase the estimated value of a home so seller can decide which renovations to make. Therefore, in my data analysis, I will explore the most impactful features in terms of pricing.

Approach and methodology

1. Import the data
2. Clean the data
3. Explore the data
4. Model (build predictive models)
5. Interpret
6. Conclusions and Recommendations
7. Future work

Data Preprocessing

In [1]:

```
! pip install matplotlib --upgrade

# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn')

# Sklearn packages
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxS
caler
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import Lasso, Ridge
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_
error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from yellowbrick.regressor import ResidualsPlot
```

```
Requirement already up-to-date: matplotlib in /Users/alinakorsunenko/op
t/anaconda3/envs/learn-env/lib/python3.6/site-packages (3.3.4)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /
Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-p
ackages (from matplotlib) (1.2.0)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /User
s/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-packa
ges (from matplotlib) (0.10.0)
Requirement already satisfied, skipping upgrade: pillow>=6.2.0 in /User
s/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-packa
ges (from matplotlib) (8.1.0)
Requirement already satisfied, skipping upgrade: numpy>=1.15 in /Users/
alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-package
s (from matplotlib) (1.19.1)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 i
n /Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/sit
e-packages (from matplotlib) (2.8.1)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.
1.2,!=2.1.6,>=2.0.3 in /Users/alinakorsunenko/opt/anaconda3/envs/learn-
env/lib/python3.6/site-packages (from matplotlib) (2.4.7)
Requirement already satisfied, skipping upgrade: six in /Users/alinakor
sunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-packages (from
cycler>=0.10->matplotlib) (1.15.0)
```

In [2]:
```
# Import data
kc_house_data = pd.read_csv('./data/kc_house_data.csv')
kc_house_data.head()
```

Out[2]:

	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 [3]:
```
# To answer my questions, I do not need all of the columns so, I can dro
p the irrelevant ones
drop_columns = ['id',
                'date',
                'view',
                'sqft_above',
                'sqft_basement',
                'zipcode',
                'lat',
                'long',
                'sqft_living15',
                'sqft_lot15'
                ]

kc_house_data = kc_house_data.drop(columns = drop_columns)
kc_house_data.head()
```

Out[3]:

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

In [4]:
```
# Preview the data
display(kc_house_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 11 columns):
price           21597 non-null float64
bedrooms        21597 non-null int64
bathrooms       21597 non-null float64
sqft_living     21597 non-null int64
sqft_lot        21597 non-null int64
floors          21597 non-null float64
waterfront      19221 non-null float64
condition       21597 non-null int64
grade           21597 non-null int64
yr_built        21597 non-null int64
yr_renovated    17755 non-null float64
dtypes: float64(5), int64(6)
memory usage: 1.8 MB

None
```

Datatypes appear to be correct, however, there are missing vallues in `waterfront` and `yr_renovated` .

In [5]:
```
# Get number of missing values in each column
kc_house_data.isna().sum()
```

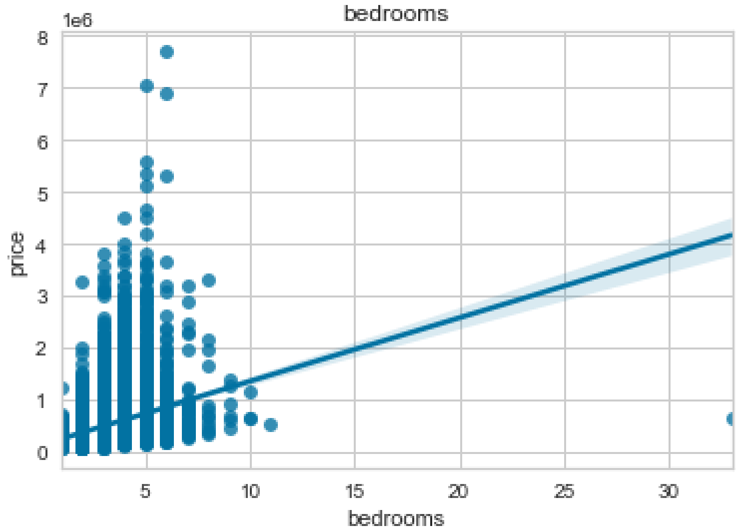
Out[5]:
```
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2376
condition          0
grade              0
yr_built           0
yr_renovated    3842
dtype: int64
```

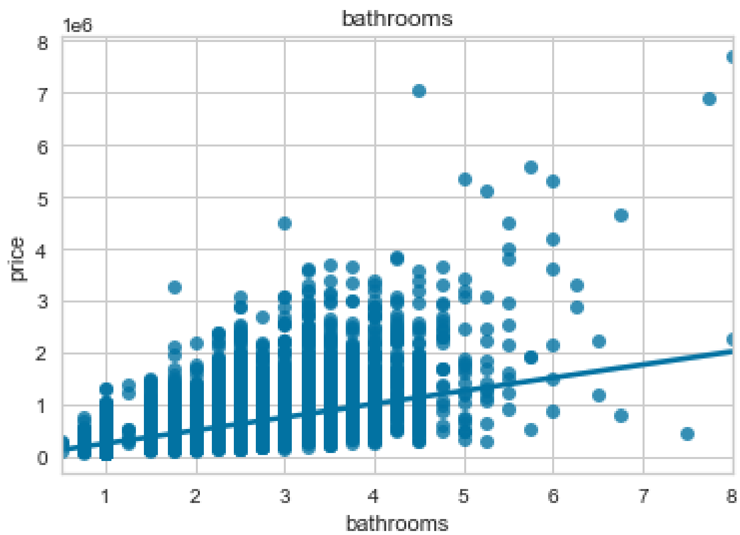
The numbers of missing values in `waterfront` and `yr_renovated` columns are significant so, I can not drop them. According to column descriptions, I can assume that missing values in `waterfront` are indicators of absence of the waterfront view and missing values in `yr_renovated` signify that a huse was not renovated.

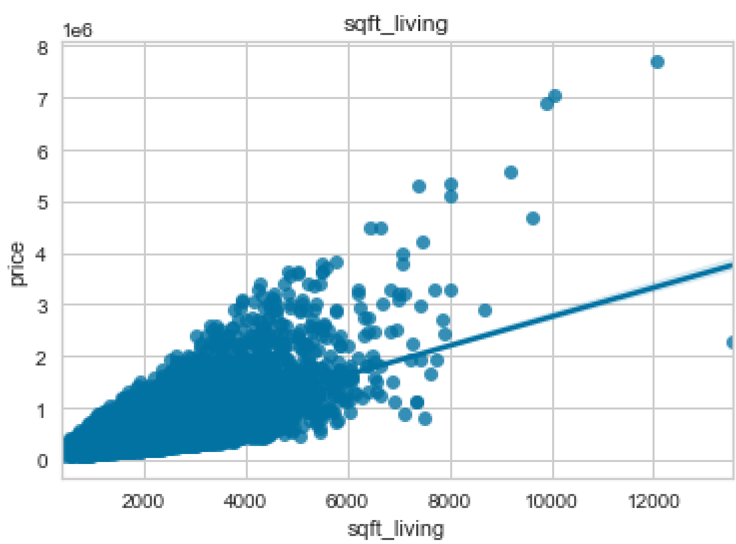
In [6]:
```
# Fill missing values for 'waterfront' with zero
kc_house_data['waterfront'] = kc_house_data['waterfront'].fillna(0)
```

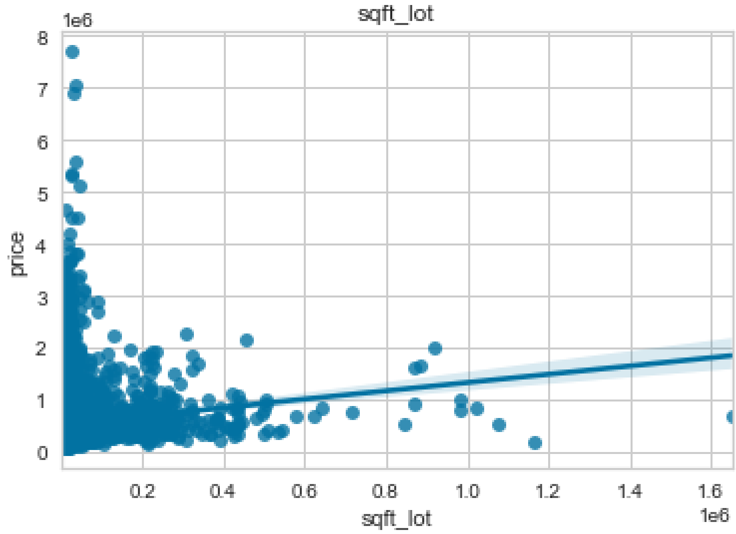
In [7]:
```
# Fill missing values for 'yr_renovated' with zero (assuming that zero m
eans - no rennovations)
kc_house_data['yr_renovated'].fillna(value = 0, inplace = True)
```

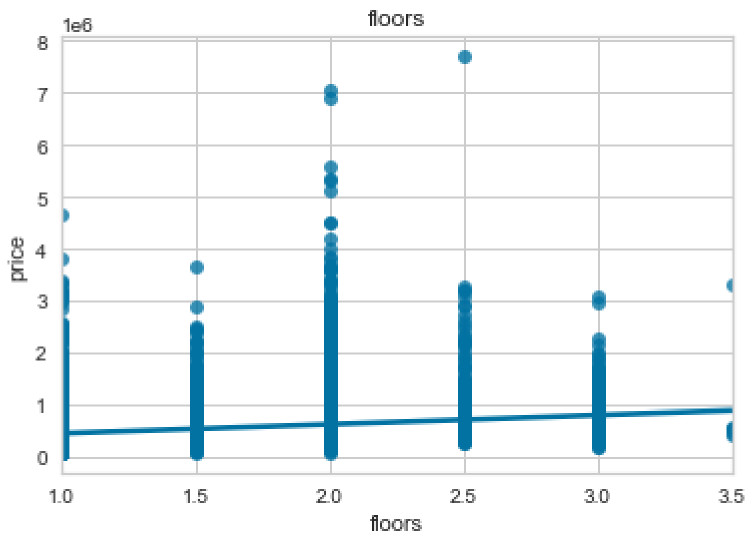
In [8]:
```
# Get an idea if there is a linear relationship between features and tar
get
X = kc_house_data.drop(columns = ['price'], axis = 1)
for col in X.columns:
    plt.subplots(1, 1)
    sns.regplot(X[col], kc_house_data.price)
    plt.title(col)
```

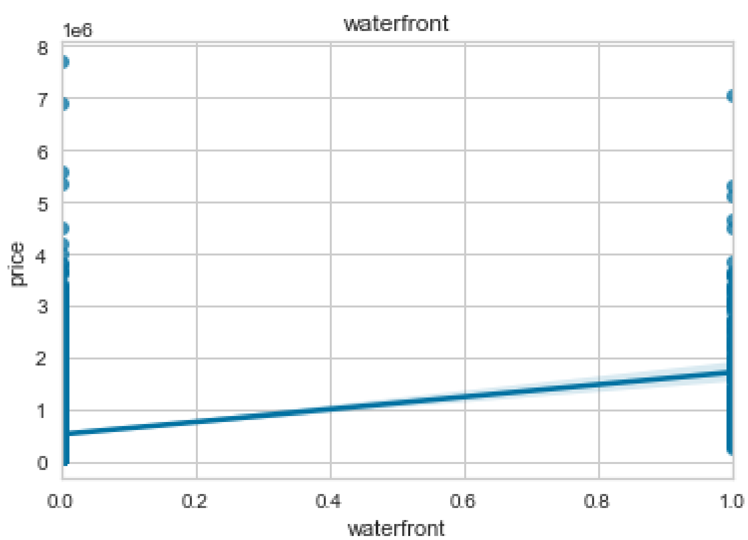

bedrooms
1e6
8
7
6
5
4
3
2
1
0
price
5
10
15
20
25
30
bedrooms

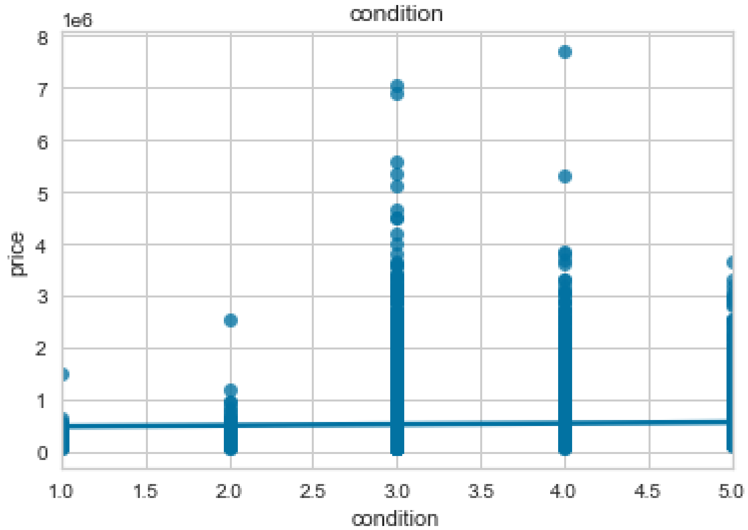

bathrooms
1e6
8
7
6
5
4
3
2
1
0
price
1
2
3
4
5
6
7
8
bathrooms

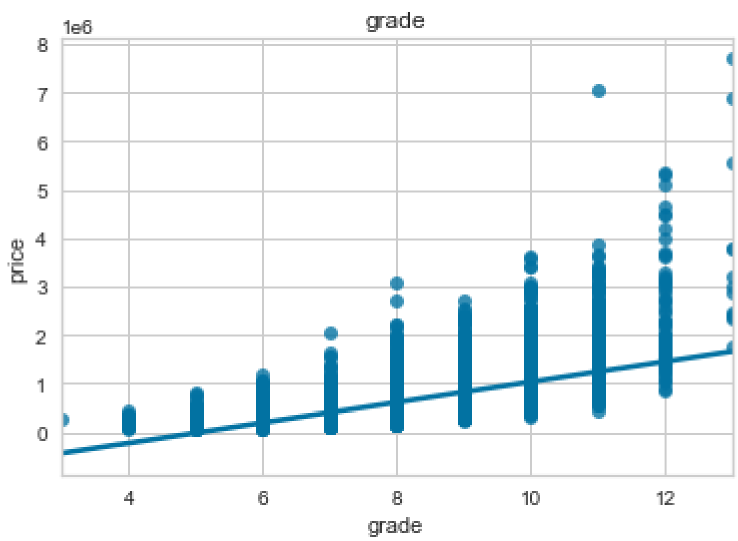

sqft_living
1e6
8
7
6
5
4
3
2
1
0
price
2000
4000
6000
8000
10000
12000
sqft_living

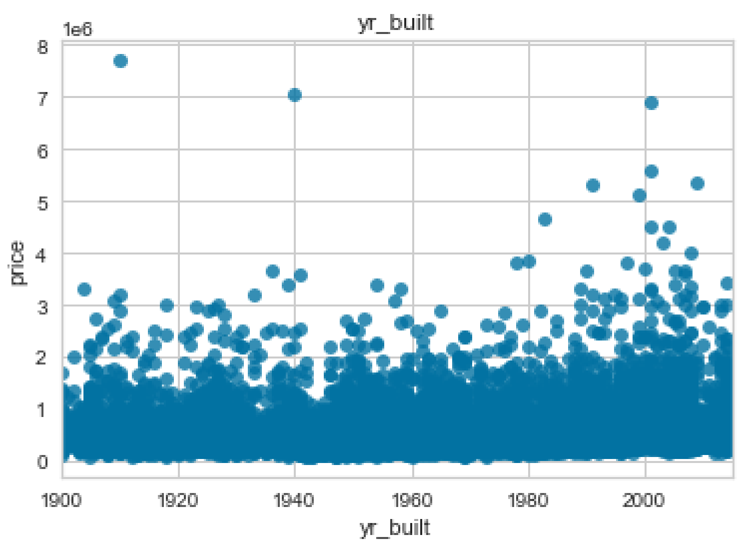

sqft_lot
1e6
8
7
6
5
4
3
2
1
0
price
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1e6
sqft_lot

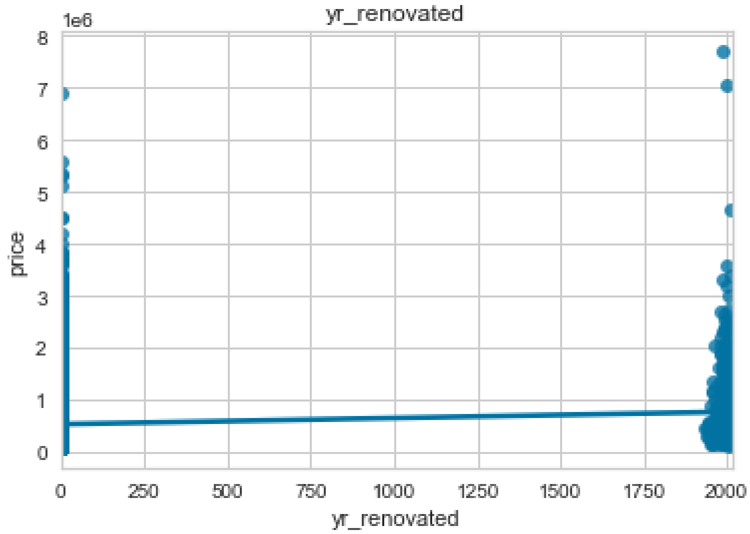

floors
1e6
8
7
6
5
4
3
2
1
0
price
1.0
1.5
2.0
2.5
3.0
3.5
floors


waterfront
1e6
8
7
6
5
4
3
2
1
0
price
0.0
0.2
0.4
0.6
0.8
1.0
waterfront


condition
1e6
price
condition


grade
1e6
price
grade

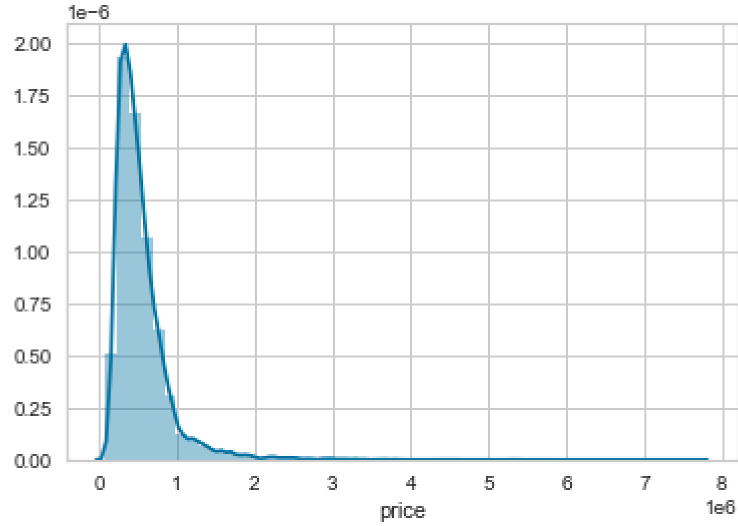

yr_built
1e6
price
yr_built



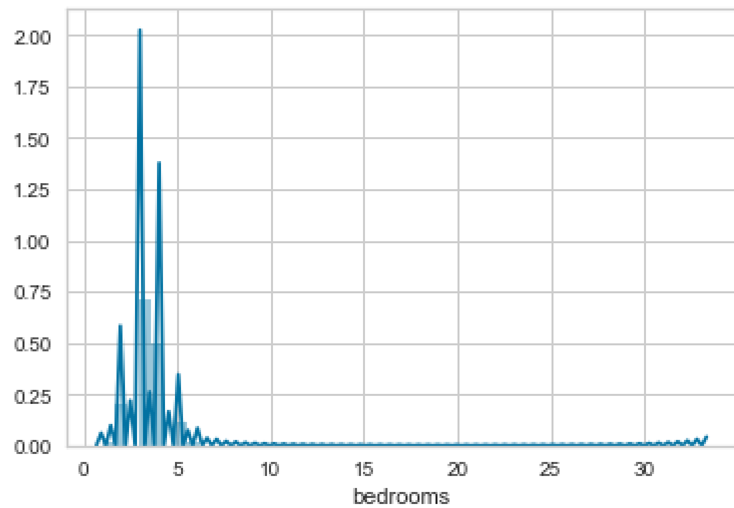
There are features which have a strong linear relationship with target: `bedrooms` , `bathrooms` , `grade` and `sqft_living` . Some features : `floors` , `condition` , `waterfront` , `yr_renovated` and others don't seem to increase the price as values increase.

Consider outliers

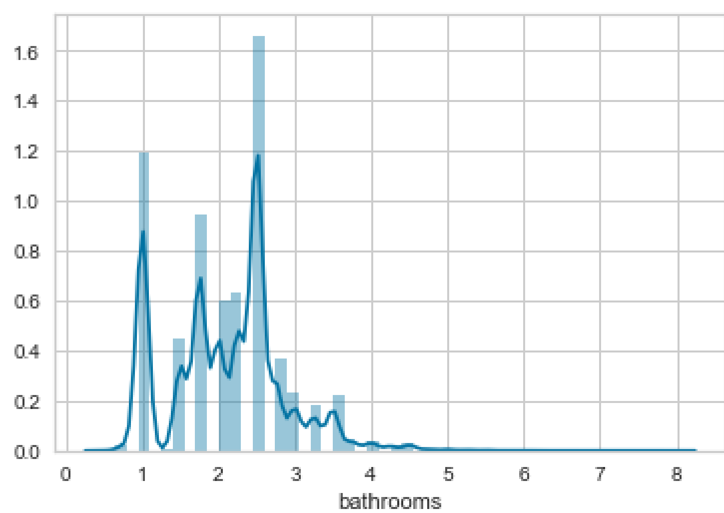
In [9]:
```
# Detect outliers
for c in kc_house_data.columns:
    plt.figure()
    try:
        if c not in ["waterfront", "yr_renovated"]:
            sns.distplot(kc_house_data[c])
        else:
            sns.distplot(kc_house_data[c], kde = False)
    except:
        print(c)
    plt.plot();
```



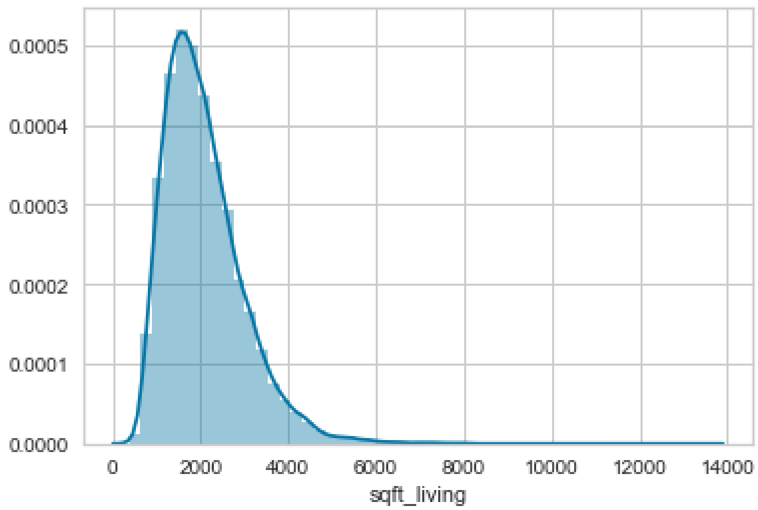
1e-6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
2
3
4
5
6
7
8
1e6
price

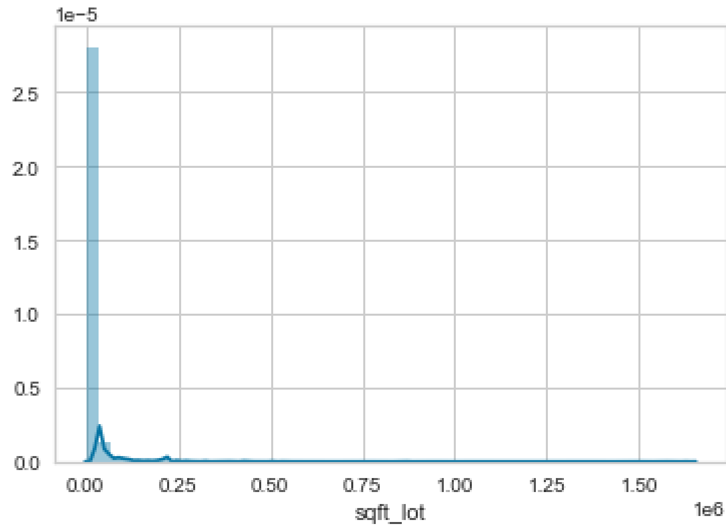


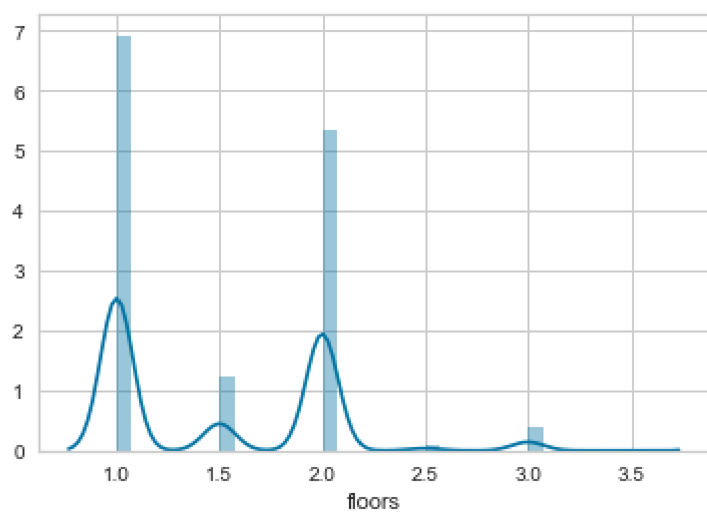
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30
bedrooms

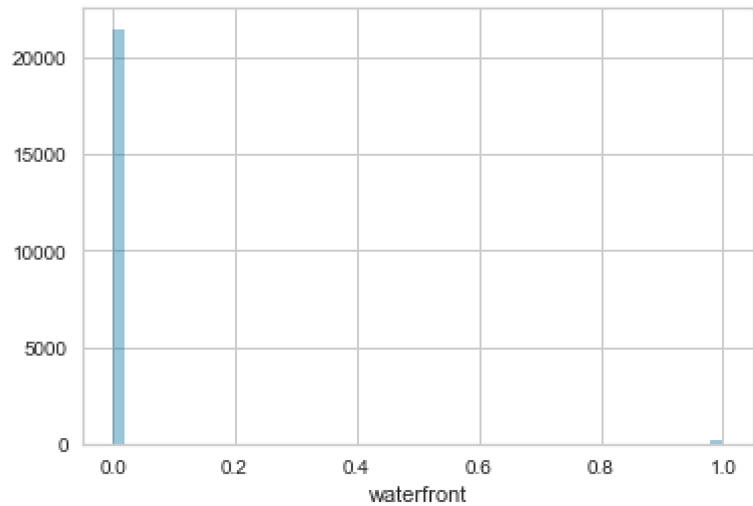


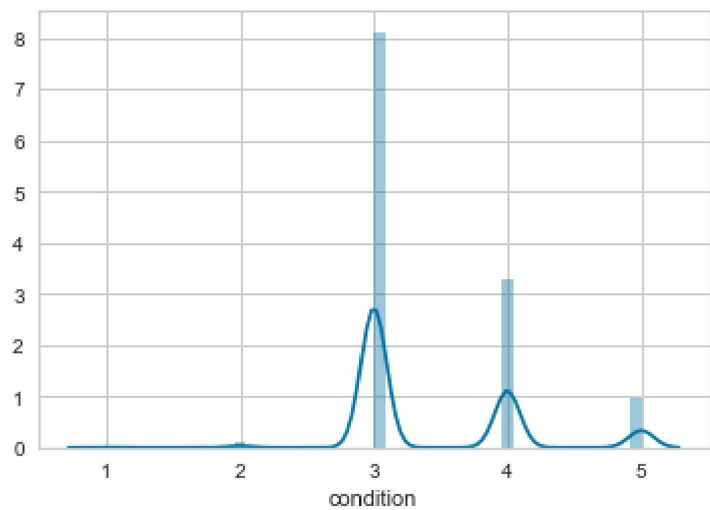
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
bathrooms

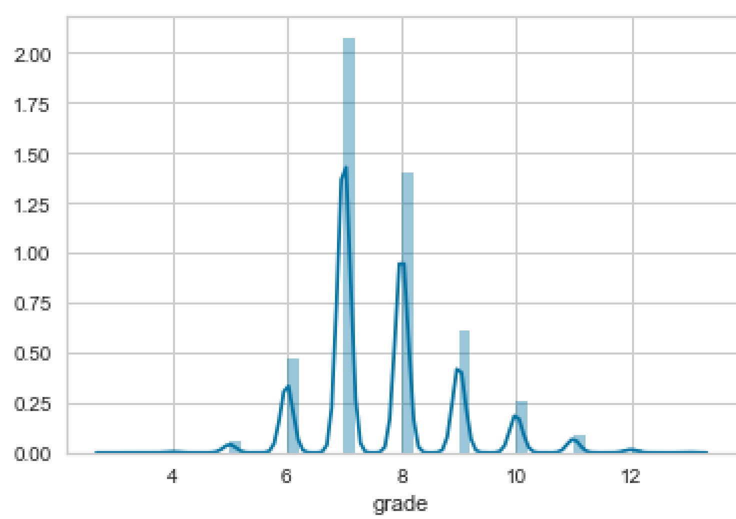

0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
2000
4000
6000
8000
10000
12000
14000
sqft_living

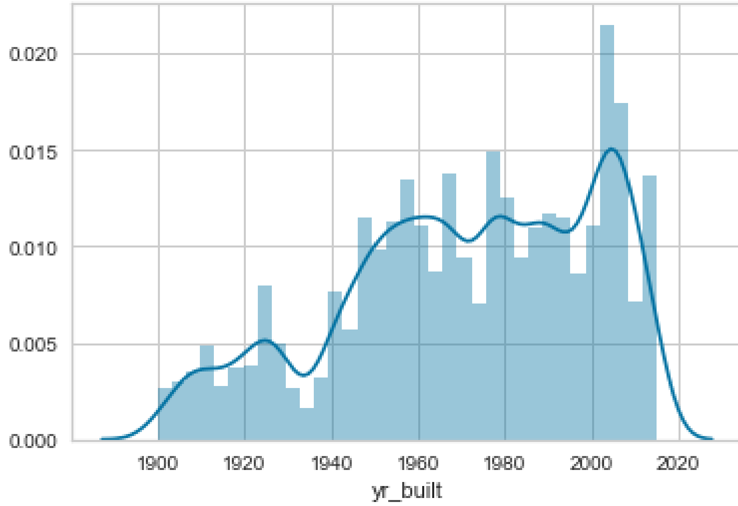

1e−5
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
sqft_lot
1e6

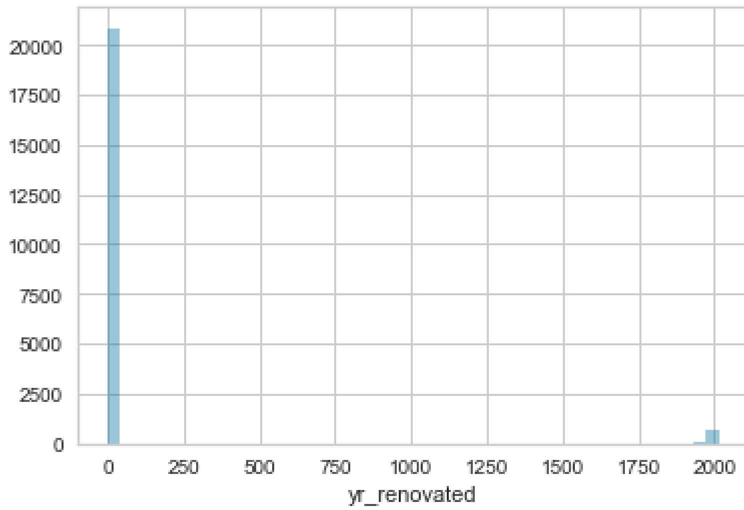

7
6
5
4
3
2
1
0
1.0
1.5
2.0
2.5
3.0
3.5
floors


20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0
waterfront


8
7
6
5
4
3
2
1
0
1
2
3
4
5
condition


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4
6
8
10
12
grade





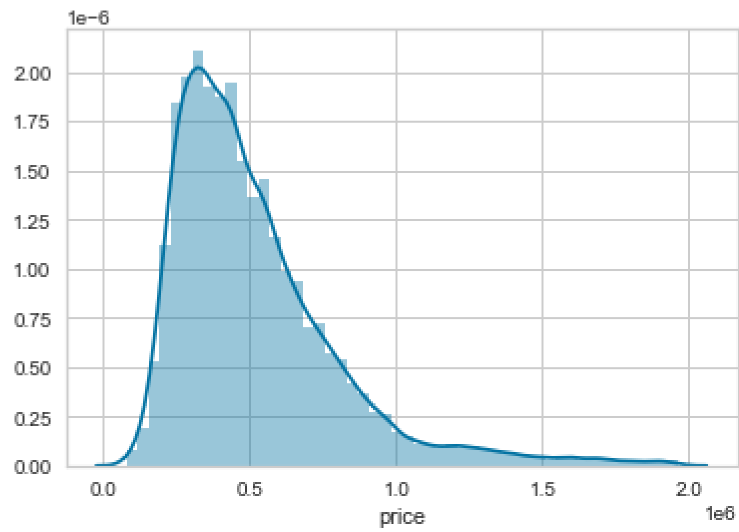
Several features have a significant outliers; for my analysis I will remove outliers for `price` and `bedrooms` .

In [10]:
```
min_q, max_q = kc_house_data['price'].quantile([0.01, 0.99])
min_q, max_q
```

Out[10]:
```
(154000.0, 1970000.0)
```

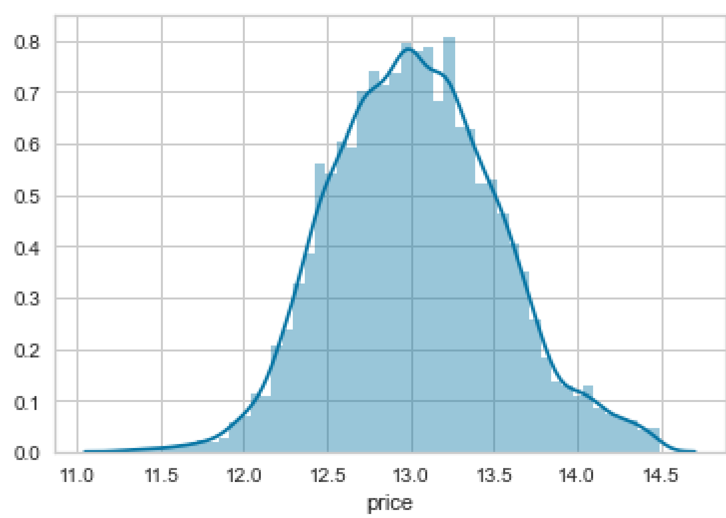
To remove outliers in `price` I can limit the dataset to properties with prices below USD1,970,000.00

In [11]:
```
# Remove outliers for target
kc_house_data = kc_house_data[kc_house_data['price'] < max_q]
sns.distplot(kc_house_data['price']);
```



When evaluating house prices, an error of $100,000.00 is more significant for cheaper houses than for expensive ones, therefore, I want to take a natural logarithm of `price` to normalize data; also, taking log of `price` will be useful for models training in my further analysis.

In [12]:
```
sns.distplot(np.log(kc_house_data['price']));
```

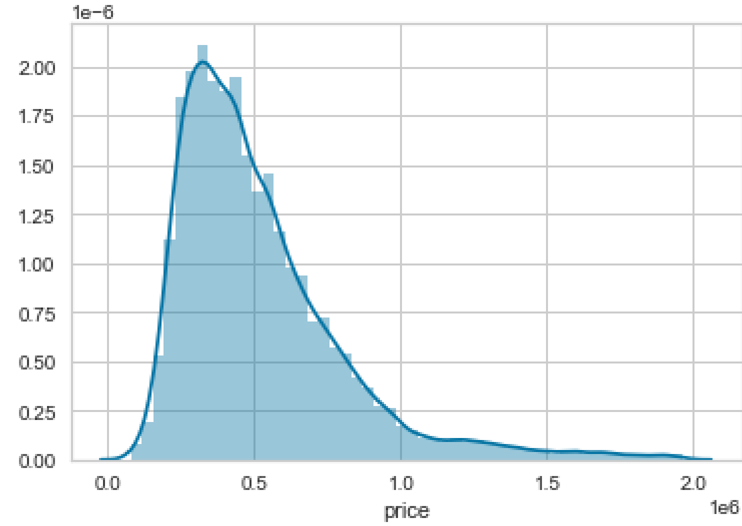


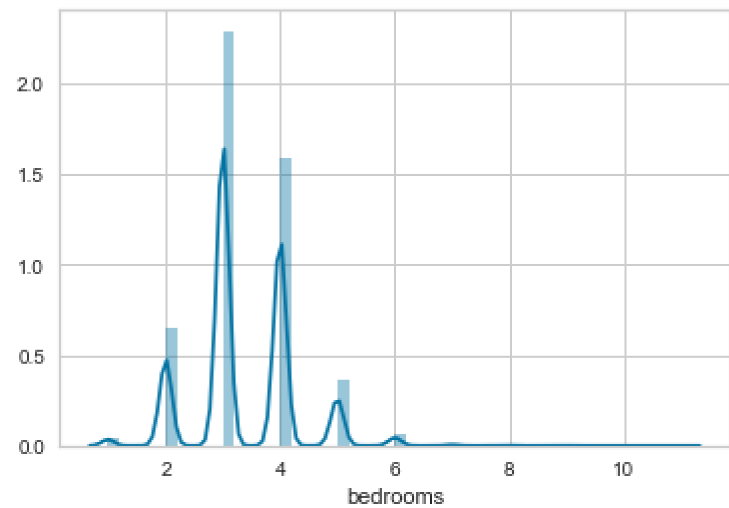
In [13]:
```
# Investigate 'bedrooms'
kc_house_data.bedrooms.value_counts()
```

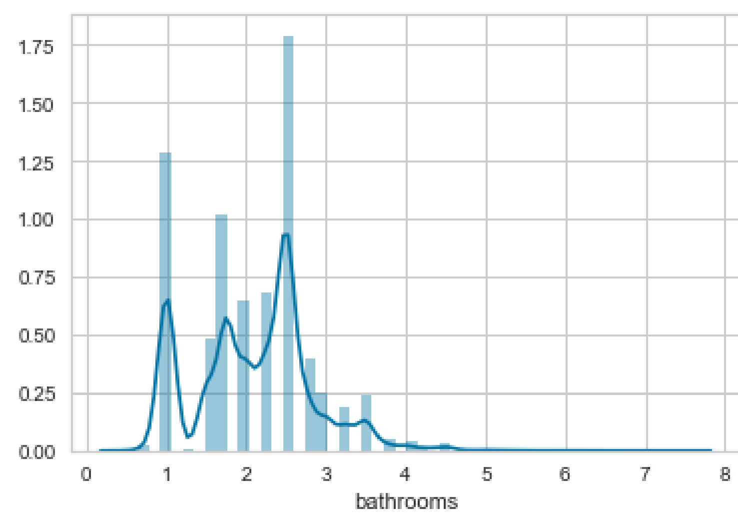
Out[13]:
```
3     9789
4     6785
2     2758
5     1539
6      259
1      196
7       33
8       10
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

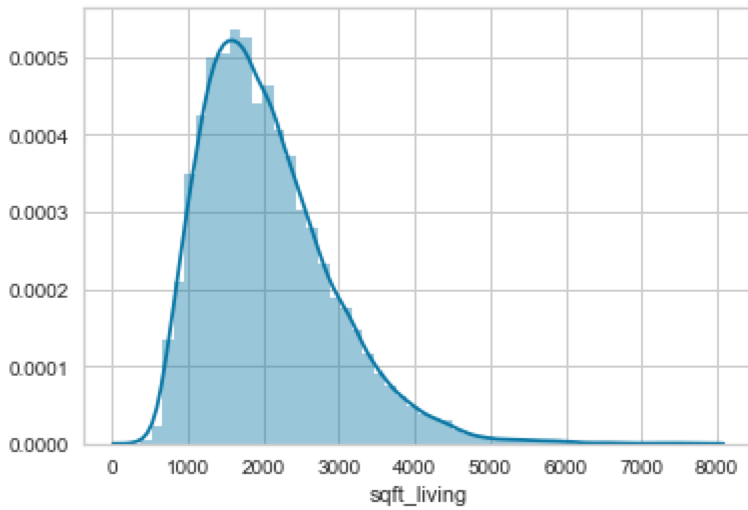
In [14]:
```
# Eliminate an oulier in 'bedrooms' column
kc_house_data = kc_house_data[kc_house_data['bedrooms'] < 32]
```

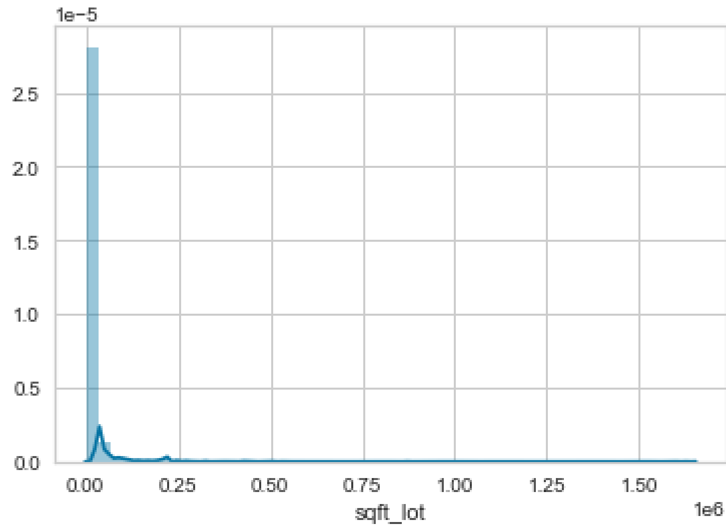
In [15]:
```
# Graphics after removing ouliers in 'price' and 'bedrooms'
for c in kc_house_data.columns:
    plt.figure()
    try:
        if c not in ["waterfront", "yr_renovated"]:
            sns.distplot(kc_house_data[c])
        else:
            sns.distplot(kc_house_data[c], kde = False)
    except:
        print(c)
    plt.plot();
```

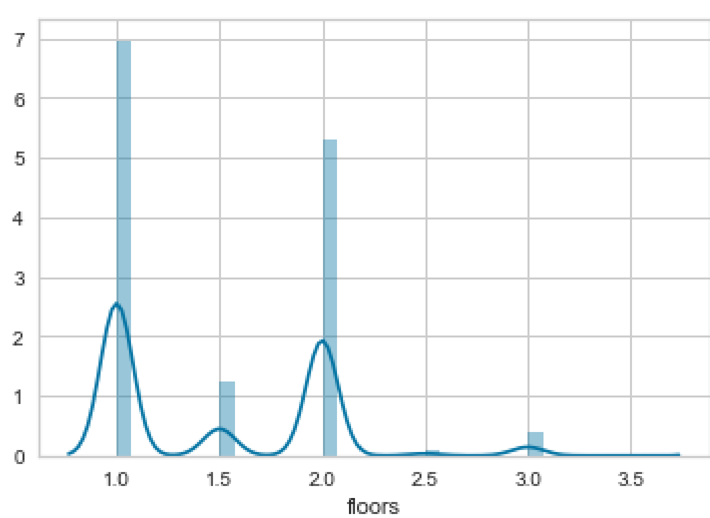

1e-6
price
1e6

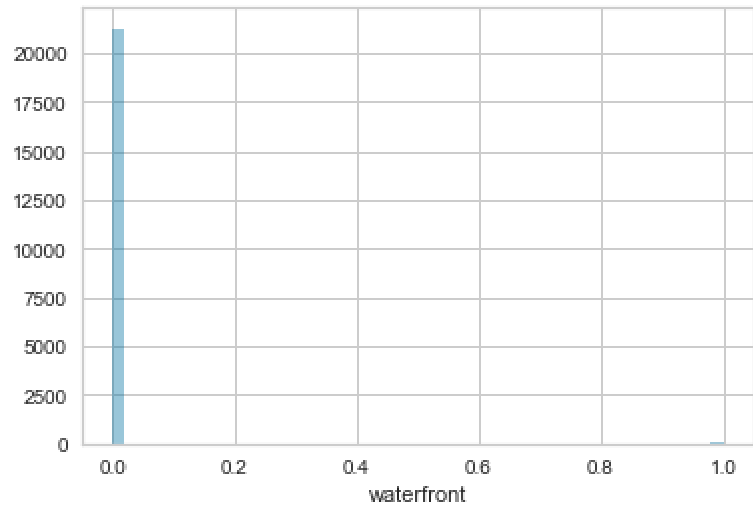

bedrooms


bathrooms

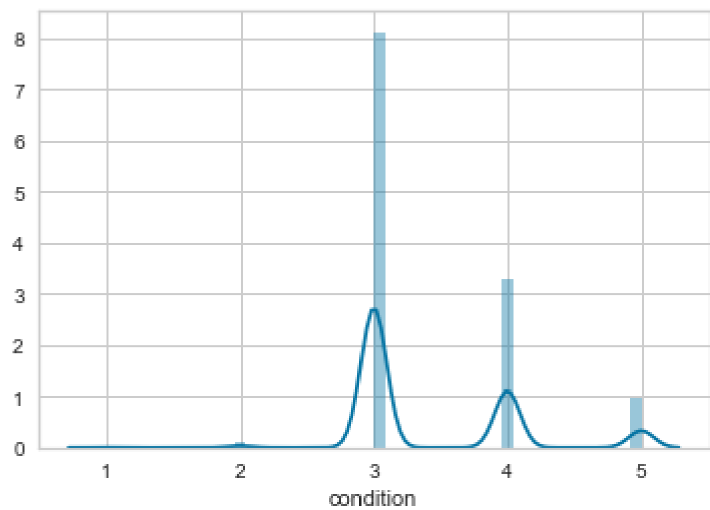

0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
1000
2000
3000
4000
5000
6000
7000
8000
sqft_living


1e−5
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
sqft_lot
1e6

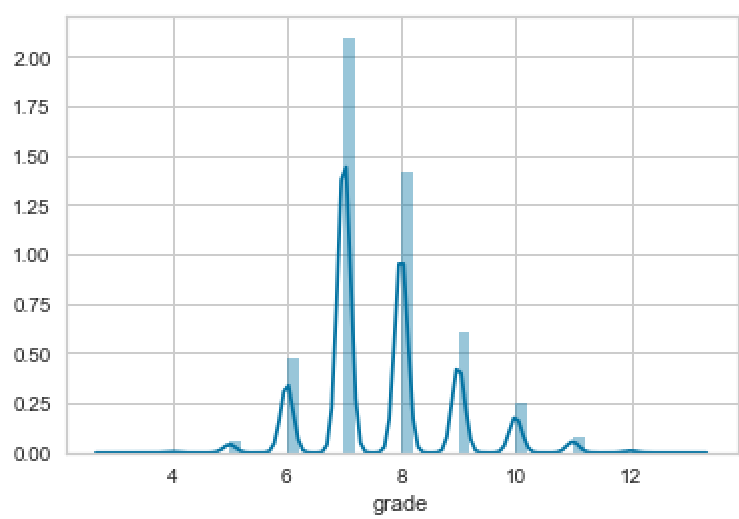

7
6
5
4
3
2
1
0
1.0
1.5
2.0
2.5
3.0
3.5
floors



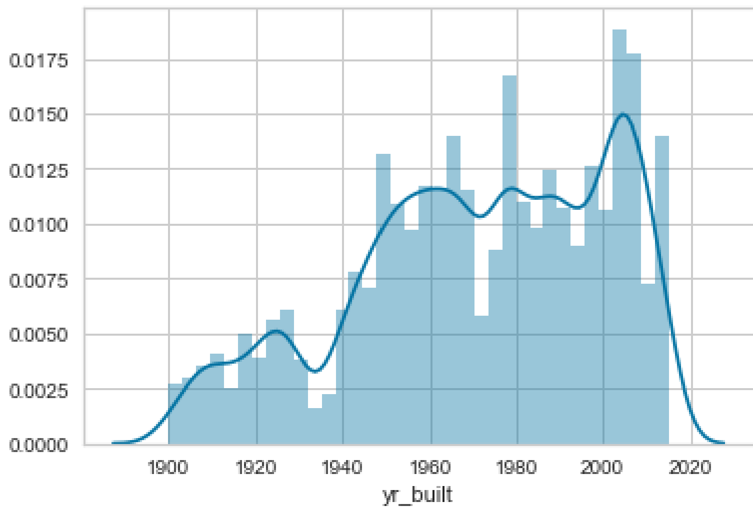
20000
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0
waterfront

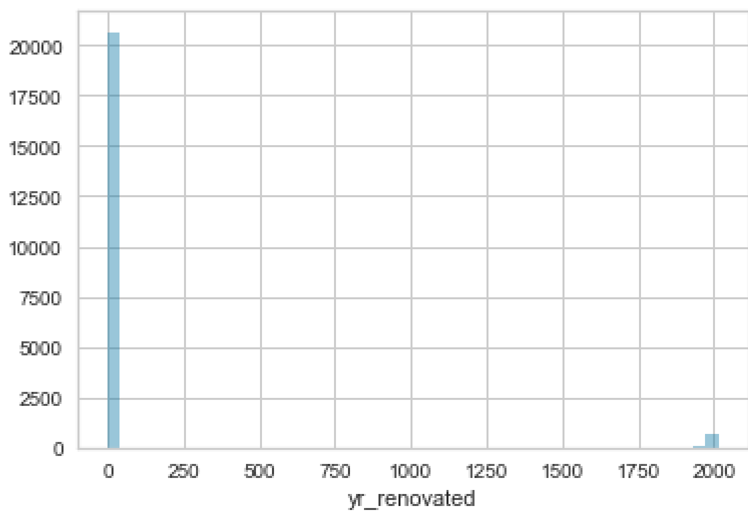


8
7
6
5
4
3
2
1
0
1
2
3
4
5
condition



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4
6
8
10
12
grade





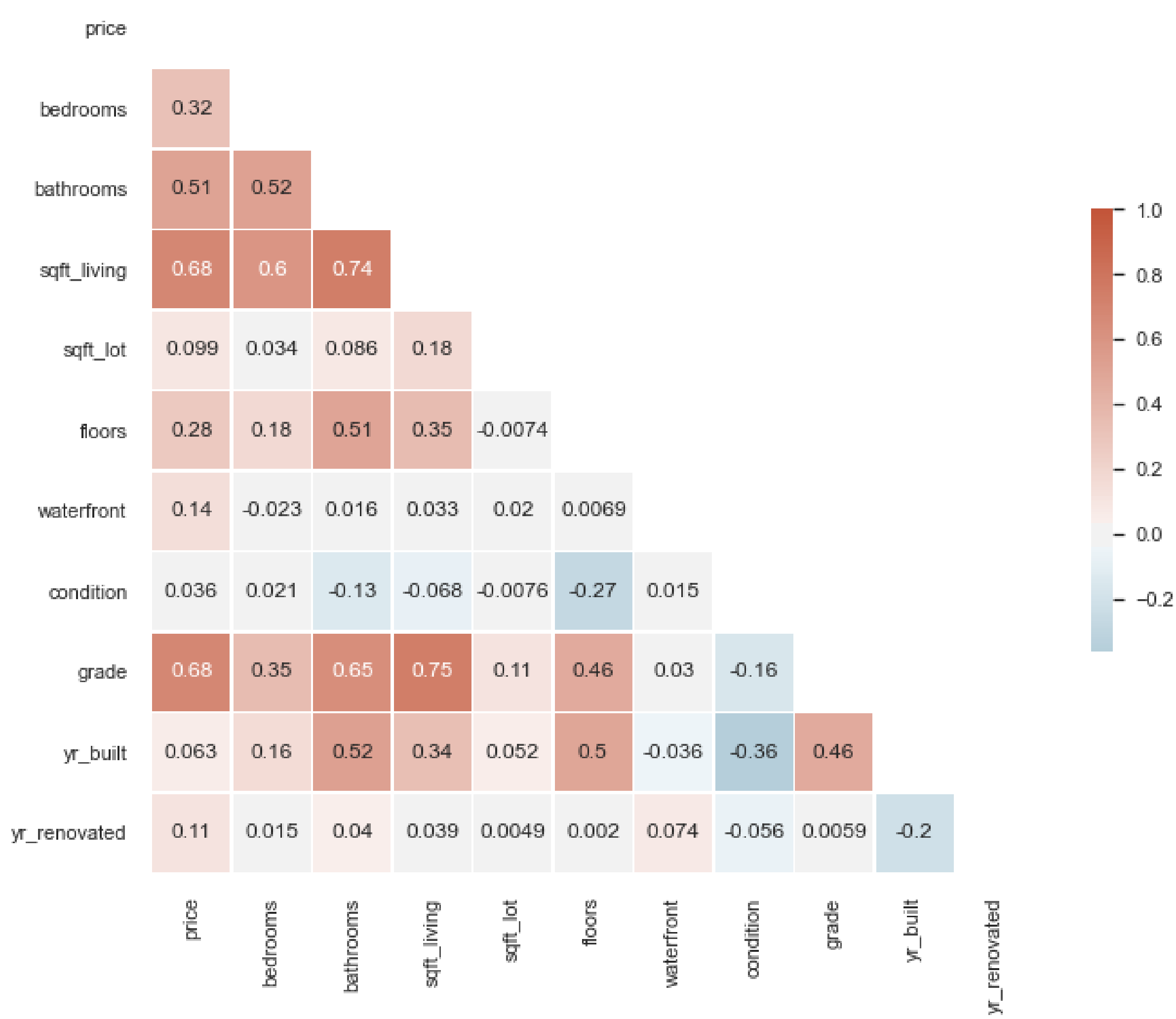
There are still some outliers in `price` , `bedrooms` , `bathrooms` and `sqft_lot` and I can eliminate them by handling outliers in `bedrooms` but for this analysis I will proceed how it is.

I can assume that a house with 33 bedroms created outliers for other features.

Diagonal correlation matrix

I want to look closer at how features correlate with target and with each other to get a better understanding of which features are important for my analysis.

In [16]:
```python
# Plotting a diagonal correlation matrix

sns.set(style = "white")

# Compute the correlation matrix
corr = kc_house_data.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype = bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize = (11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap = True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask = mask, cmap = cmap, center = 0, annot = True,
            square = True, linewidths = .5, cbar_kws = {"shrink": .5});
```



This correlation matrix gives me an idea of which features are correlated with each other (for example, `bathrooms` , `grade` and `bedrooms` have high correlation with `sgft_living`) and how these features correlate with target.

There are some features with less correlation to the target (`condition` , `yr_built` , `sqft_lot`). Also, there are features with clear correlation to the target (`sqft_living` , `grade` , `bathrooms`).

Diagonal correlation matrix answers an important question for my analysis: "Which features are the most impactful in terms of pricing?".

According to this correlation matrix, I can assume that bulding `grade` impacts `price` significantly due to the high correlation level (0.68). Also, I can state that the next features are the most impactful in terms of pricing: `sqft_living` , `grade` , `bathrooms` .

However, my analysis will be more valuable if I will use other methods to answer my research questions.

Models Training

The Linear Regression model may be a good option to show a dependency between `price` and features, however, I have decided to use Lasso and Ridge models too. Lasso and Ridge models push coefficients towards 0 and by doing this, coefficients become optimized for prediction which is helpful in avoiding overfitting and can lead to more accurate results.

Also, I will use log of target in my models training. Taking log of target is useful for making predictions; in this dataset my target variable involves money variable and as mentioned earlier an error of $100,000.00 is more significant for cheaper houses than for expensive ones.

Therefore, to make my final conclusions more reliable, I will be comparing Linear Regression, Ridge and Lasso prediction models with original target and with log of target to find the best model based on R2 score. Then, I will use additional metrics (Mean Squared Error, Mean Absolute Error and Root Mean Square Error) to measure the errors of a chosen model.

R2 score indicates how well a regression model fits a data set. However, R2 score doesn’t tell the entire story so for my future work I can try a different approach of regression analysis by building models and adjusting (remove insignificant features) them until a final model is built.

In [17]:
```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(kc_house_data.drop(c
olumns = ['price']),
                                                    kc_house_data['pric
e'],
                                                    test_size = 0.3,
                                                    random_state = 42)

# Declare the numerical features
numerical_features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot'
, 'floors',
       'waterfront', 'condition', 'grade', 'yr_built', 'yr_renovated']
categorical_features = []

# Transform numerical features by scaling each of the features to a rang
e between 0 and 1.
column_transformer = ColumnTransformer([
    ('ohe', OneHotEncoder(handle_unknown = "ignore"), categorical_featur
es),
    ('scaling', MinMaxScaler(), numerical_features)
])
```

In [18]:
```
# Define a function to get R2 scores from 6 models by training them usin
g different modes
def train_regression(model, X_train, y_train, X_test, y_test, mode) -> r
2_score:
    """
    mode: "original" or "log" of target

    """

    pipeline = Pipeline(steps = [
        ('scaling', column_transformer),
        ('regression', model)
    ])

    if mode == "log":
        pipeline.fit(X_train, np.log(y_train))
        preds = pipeline.predict(X_test)
        return r2_score(y_test, np.exp(preds))
    else:
        pipeline.fit(X_train, y_train)
        preds = pipeline.predict(X_test)
        return r2_score(y_test, preds)

# Print results for 6 models
models = [LinearRegression(), Lasso(), Ridge(), LinearRegression(), Lass
o(alpha = 0.001), Ridge()]

for i, model in enumerate(models):
    if i < 3:
        mode = 'original'
    else:
        mode =  'log'
    score = train_regression(model, X_train, y_train, X_test, y_test, mo
de)

    modelname = model.__class__.__name__
    print(f"{modelname} {mode}:".ljust(30, ' '), round(score, 6))
```

```
LinearRegression original:     0.633414
Lasso original:                0.633411
Ridge original:                0.633398
LinearRegression log:          0.56877
Lasso log:                     0.592308
Ridge log:                     0.570815
```

Th Linear Regression with original target has the highest R2 score (0.633414) therefore, I have decided to proceed with this model.

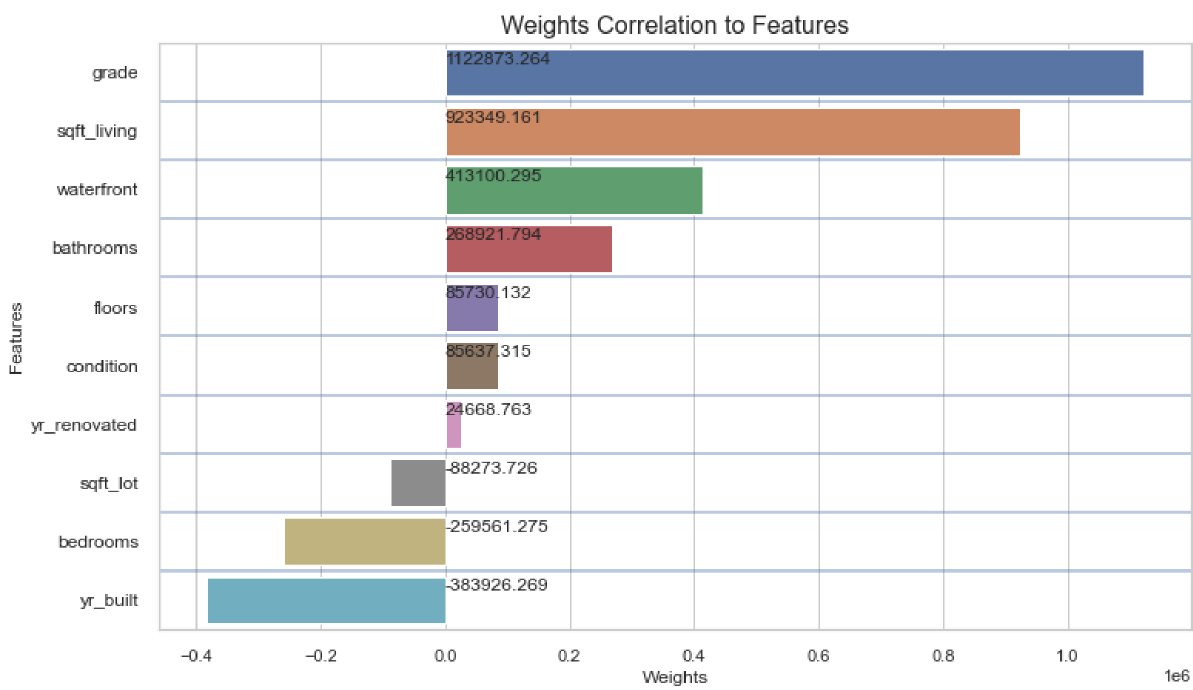
Linear Regression with Original Target

In this section I want to compare weights to define the most important features(the hieghest weights signify the most important features). However, before comparing weights I need to scale features.

In [19]:
```
# Create a pipeline
pipeline_LinearRegression = Pipeline(steps = [
    ('ohe_and_scaling', column_transformer),
    ('regression', LinearRegression())
])

pipeline_LinearRegression.fit(X_train, y_train)
preds = pipeline_LinearRegression.predict(X_test)

weights_features = list(zip(pipeline_LinearRegression['regression'].coef
_, kc_house_data.drop(columns = ['price']).columns))
```

In [20]:
```
# Sort weights
weights_features.sort(reverse = True)
```

In [21]:
```
# Plot the weights correlation to features
weights_lr = [p[0] for p in weights_features]
features_lr = [p[1] for p in weights_features]

plt.figure(figsize = (12, 7))
sns.set_style('whitegrid')
sns.barplot(weights_lr, features_lr)
plt.title('Weights Correlation to Features', fontsize = 16)
plt.xlabel('Weights')
plt.ylabel('Features')
plt.yticks(fontsize = 12)


for ind, val in enumerate(weights_lr):
    plt.text(x = -0.05, y = ind - 0.15, s = round(val, 3), fontsize = 12
)
    plt.axhline(ind - .5, alpha = 0.5)

plt.show()
```



`grade` , `sqft_living` and `waterfront` features have the greatest impact in a positive direction.

In [22]:
```
print('mean target:', np.mean(y_train))
print('std target :', np.std(y_train))
```

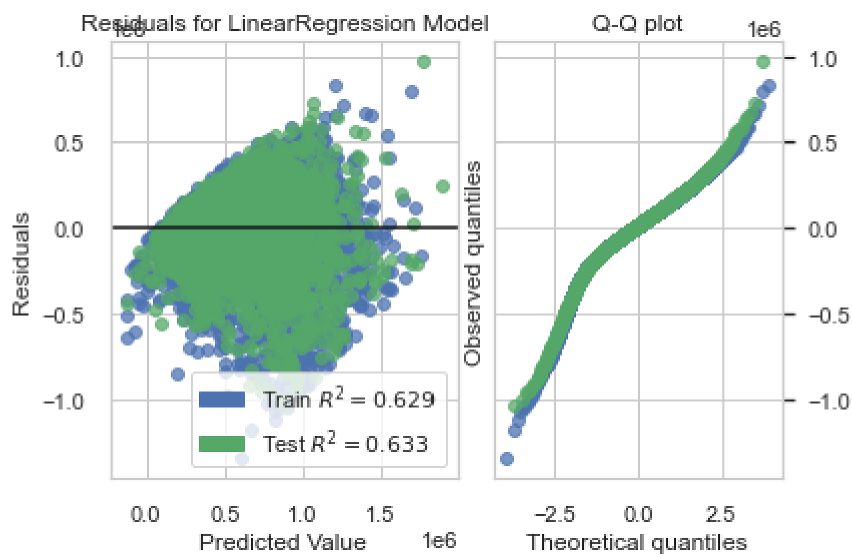
```
mean target: 518047.1454727698
std target : 284479.18992788997
```

In [23]:
```
# evaluate predictions
y_test_preds = pipeline_LinearRegression.predict(X_test)
mae = mean_absolute_error(y_test, y_test_preds)
mse = mean_squared_error(y_test, y_test_preds)
print(f"Test MAE:  {mae}")
print(f"Test MSE:  {mse}")
print(f"Test RMSE: {np.sqrt(mse)}")
```

```
Test MAE:  125240.78404847429
Test MSE:  29872976084.342712
Test RMSE: 172838.00532389487
```

The Linear Regression model is not overfitted.

In [24]:
```
# Check my residual plots to ensure trustworthy linear regression result
s
visualizer = ResidualsPlot(LinearRegression(), hist = False, qqplot = Tr
ue)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



Out[24]:
```
<AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, x
label='Predicted Value', ylabel='Residuals'>
```

There are mostly places where my model is good at predicting.

Recommendations

1. Based on the above analysis, I can assume that building `grade` has the strongest relationship with a housing `price` . I recommend stakeholders take into consideration below grade score descriptions before renovating a house. For example, to increase a home value sellers can improve exterior and interior finish work and design, check with cities municipal office to ensure sellers building plans are up-to-date and add amenities of solid woods, bathroom fixtures and more luxurious options.
2. Another important feature which has a strong relationship with a `price` is `sqft living` . I recommend stakeholders check if there is a possibility to increase living space,for example, by adding a `bathroom` and or `bedroom` .However, an additional `bedroom` does not necessarily result in a a sale price increase.
3. According to the results of `weights correlation to features` it is clear that `waterfront` feature has a positive correlation with housing `price` , however, it is something that sellers can not change to increase a house value so, I recommend stakeholders to take into consideration `yr_built` feature before selling a house. I can assume that it may be beneficial to sellers sell a house before it gets too old, however, it is important to take into consideration different factors before selling a house.

Below are the building grade score descriptions pulled from the King County site:

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.

Future work

1. Testing `date` feature to find out which months are the most popular for house sales.
2. Work on adding more insights by using an iterative approach to multiple regression modelling to determine the most impactful features .
3. Demonstrate by what exact amount the features might increase the estimated homes value.