# student

April 19, 2021

Please fill out: * Student name: Tamjid Ahsan * Student pace: Full Time * Scheduled project review date/time: * Instructor name: James Irving * Blog post URL:

# 1 INTRODUCTION

## 1.1 Overview

King County is located in the U.S. state of Washington. The population was 2,252,782 in the 2019 census estimate, making it the most populous county in Washington, and the 12th-most populous in the United States. The county seat is Seattle, also the state's most populous city. The county is named to honor civil rights leader Martin Luther King Jr.[1]

The county sees a USD 700K Median Listing Home Price, USD 431 Median Listing Home Price/Sq Ft, USD 766K Median Sold Home Price, with around 6000 homes listed for sale and 2800 homes for rent.[2] Homes in King County, WA sold for 2.19% above asking price on average in March 2021 with a median of 36 days days on market.

In March 2021, King County home prices were up 15.4% compared to last year. There were 3,198 homes sold in March 2021, up from 2,656 last year.[3].

This makes the county a prime market of real estate, and a paramount candidate for analysis. The namesake of Martin King Luther Jr. has a very lucrative real estate market worth exploring.

## 1.2 Business problem

Source: image generated by author using tableau public, and online gif maker.

King County Residents that want to renovate their home to increase its resale value, but don't know what factors are important for determining a home's value. While focusing on features they can renovate, I shall discuss key factors of a home's price. This factors can be both under their constrain and beyond their control.

Those include: - How to improve marketability. - Focus on which aspect of the house. - What factors to keep in mind deciding budget and required return on investment.

# 2 The imports

## 2.1 Packages

[1]:
```
# core operational packeges
import os
import warnings
# data manupulation
import pandas as pd
pd.set_option('display.max_columns',0)
import numpy as np

# plotting
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px

# Statistics
import statistics as stat
import scipy.stats as stats

# regression
## statsmodels
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from statsmodels.formula.api import ols
## scikit-learn
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import linear_model
from sklearn.linear_model import LinearRegression

# notebook styling packages
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context='notebook', ticks='True', grid='False')
## to reset to default theme
# jtplot.reset()
```

## 2.2 Functions used

### 2.2.1 Exploration

[2]:
```
def check_NaN(df):
    """
    Checks for NaN in the pandas DataFrame and spits a DataFrame of report.
    Uses df.isnull() method.
```

```
    Parameters:
    ===========
    df = pandas.DataFrame
    """
    null_checking = []
    for column in df.columns:
        not_null = df[column].isnull().value_counts()[0]
        try:
            is_null = df[column].isnull().value_counts()[1]
        except:
            is_null = 0
        temp_dict = {'name': column, 'is_null': is_null, 'not_null': not_null}
        null_checking.append(temp_dict)
    df_ = pd.DataFrame(null_checking)
    return df_
```

[3]:
```
def check_duplicates(df):
    """
    Checks for duplicates in the pandas DataFrame and splits a Dataframe of␣
 ↪report.

    Parameters:
    ===========
    df = pandas.DataFrame
    """
    dup_checking = []
    for column in df.columns:
        not_duplicated = df[column].duplicated().value_counts()[0]
        try:
            duplicated = df[column].duplicated().value_counts()[1]
        except:
            duplicated = 0
        temp_dict = {
            'name': column,
            'duplicated': duplicated,
            'not_duplicated': not_duplicated
        }
        dup_checking.append(temp_dict)
    df_ = pd.DataFrame(dup_checking)
    return df_
```

[4]:
```
def correlation_top_bottom(df):
    """
    Input a Pandas correlation matrix DataFrame (df.corr()) to get top 10␣
 ↪positive and negetively correlated features.

    Parameters:
```

```
    ===========
    df = pandas.core.frame.DataFrame; use df.corr().
    """
    corr_df_matrix_ = df.unstack().reset_index()
    corr_df_matrix_.columns = ["feature_0", 'feature_1', 'correlation']
    corr_df_matrix_['keep'] = corr_df_matrix_.apply(
        lambda x: False if x['feature_0'] == x['feature_1'] else True, axis=1)
    corr_df_matrix_['feature_combo'] = corr_df_matrix_.apply(
        lambda x: ' and '.join(set(x[['feature_0', 'feature_1']])), axis=1)
    corr_featurs = corr_df_matrix_[corr_df_matrix_.keep][[
        'feature_combo', 'correlation'
    ]].drop_duplicates().sort_values(by='correlation', ascending=False)
    print(
        f'Positive correlations:\n{corr_featurs.head(10).
↪reset_index()}\n\n{"-"*70}\nNegative correlations:\n{corr_featurs.
↪sort_values(by="correlation").head(10).reset_index()}'
    )
```

[5]:
```
def format_number(data_value, index):
    # Number formatter
    """
    Formats values to thousands, millions and billions.
    Used for modifying chart axes.
    +++ formatting helper function +++
    """
    if data_value >= 1_000_000_000:
        formatter = '${:1.1f}B'.format(data_value * 0.000_000_001)
    elif data_value >= 1_000_000:
        formatter = '${:1.0f}M'.format(data_value * 0.000_001)
    else:
        formatter = '${:1.0f}K'.format(data_value * 0.001)
    return formatter
```

[6]:
```
def feature_prob_details(x, alpha=.05):
    """
    Displays:
    =========
    Plots boxplot, density plot, cdf and probability plot with fitting of given␣
↪data.
    Decison based on both:
    Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95%␣
↪confidence interval.

    Parameters:
    ===========
    x = array OR DataFrame Series Object; target data
    alpha = float; default: .05
```

```
    Note:
    =====
    For Kolmogorov-Smirnov it uses two sided test. To change this modify source␣
↪code.
    """
    plt.close('all')
    fig = plt.figure(figsize=(14, 6))

    ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=2)
    ax2 = plt.subplot2grid((3, 3), (0, 2))
    ax3 = plt.subplot2grid((3, 3), (1, 0), colspan=2, rowspan=2)
    ax4 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)
    # ax1
    sns.boxplot(x=x, ax=ax1,
                color='gold')  #.xaxis.set_major_formatter(format_number)
    ax1.set_title('Box and Whiskers Plot')
    # ax2
    sns.kdeplot(x,
                color='gold',
                label='kde',
                palette="pastel",
                cumulative=True,
                ax=ax2)
    ax2.set_title('CDF')
    # ax3
    sns.histplot(x,
                 bins=20,
                 alpha=.5,
                 color='silver',
                 stat="density",
                 fill=True,
                 label='hist',
                 palette="pastel",
                 element='step',
                 ax=ax3)#.xaxis.set_major_formatter(format_number)
    sns.kdeplot(x, color='gold', label='kde', palette="pastel", ax=ax3)
    ax3.set_title('Density Estimation')
    ax3.legend()
    # ax4
    sm.graphics.qqplot(x,
                       fit=True,
                       line='r',
                       marker='h',
                       markerfacecolor='silver',
                       alpha=0.5,
                       ax=ax4)
```

```
    sm.qqline(line='45', fmt='g--', lw=1, alpha=.8, ax=ax4)
    ax4.set_title('Probability Plot')
    plt.suptitle(t=f'Probability details of "{x.name}" feature',
                 size=20,
                 weight=14)
    plt.tight_layout()
    plt.show()

    from scipy.stats import shapiro
    from scipy.stats import kstest
    t_stat, p_val = shapiro(x)
    t_stat1, p_val1 = kstest(x, stats.norm.cdf)
    alpha = alpha
    if (p_val or p_val1) < alpha:
        print(
            f'Distribution is NOT NORMAL based on Shapiro-Wilk Test and␣
↪Kolmogorov-Smirnov test for goodness of fit at {(1-alpha)*100}% confidence␣
↪interval.'
        )
    else:
        print(
            f'Distribution IS NORMAL based on Shapiro-Wilk Test and␣
↪Kolmogorov-Smirnov test for goodness of fit at {(1-alpha)*100}% confidence␣
↪interval.'
        )
```

[7]:
```
def silencer():
    # warnings husher
    """
    Silences user warning
    +++ formatting helper function +++
    """
    warnings.simplefilter("ignore")
    warnings.warn("UserWarning", UserWarning)
```

[8]:
```
def heatmap_DataFrame(df, annot_format='.1f'):
    """
    Return a masked heatmap of the given DataFrame

    Parameters:
    ===========
    df            = pandas.DataFrame object.
    annot_format  = str, for formatting; default: '.1f'

    Example `annot_format`:
    -----------------------
    .1e = scientific notation with 1 decimal point (standard form)
```

```
    .2f = 2 decimal places
    .3g = 3 significant figures
    .4% = percentage with 4 decimal places

    Note:
    =====
    Rounding error can happen if '.1f' is used.
    """
    with plt.style.context('dark_background'):
        plt.figure(figsize=(10, 10), facecolor='k')
        mask = np.triu(np.ones_like(df.corr(), dtype=bool))
        cmap = sns.diverging_palette(3, 3, as_cmap=True)
        sns.heatmap(df.corr(),
                    mask=mask,
                    cmap=cmap,
                    annot=True,
                    fmt=annot_format,
                    linecolor='k',
                    annot_kws={"size": 9},
                    square=True,
                    linewidths=.5,
                    cbar_kws={"shrink": .5})
        plt.title(f'Features heatmap', fontdict={"size": 20})
```

[9]:
```
def top_correlated_features(df, limit=.75, verbose=False):
    """
    Input a Pandas DataFrame to get top correlated (based on absolute value)␣
 ↪features filtered by a cutoff.

    Parameters:
    ===========
    df        = pandas.DataFrame object.
    limit     = float; default: .75
    verbose   = boolean; default: False.
        `True` returns DataFrame without filtering by cutoff.
        `False` returns DataFrame filted by cutoff.
    """
    df_corr = df.corr().abs().unstack().reset_index().sort_values(
        0, ascending=False)
    df_corr.columns = ["feature_0", 'feature_1', 'correlation']
    df_corr['keep_me'] = df_corr.apply(
        lambda x: False if x['feature_0'] == x['feature_1'] else True, axis=1)
    df_corr['feature_combo'] = df_corr.apply(
        lambda x: ' and '.join(set(x[['feature_0', 'feature_1']])), axis=1)

    corr_features = df_corr[df_corr.keep_me == True][[
        'feature_combo', 'correlation'
```

```
    ]].drop_duplicates().reset_index(drop='index')
    # features with correlation more than 75%
    if verbose == True:
        return corr_features
    else:
        return corr_features[corr_features.correlation > limit]
```

[10]:
```
def check_for_high_p_val(input_, thresh_hold=.05, sorted_=False):
    """
    Input a statsmodels.regression.linear_model.RegressionResultsWrapper to get␣
 ↪features abobe a thresh hold.

    Parameters:
    ===========
    input_ = statsmodels.regression.linear_model.RegressionResultsWrapper
    thresh_hold = float; default: .05.
    """
    x = input_.pvalues[input_.pvalues > thresh_hold].to_frame().reset_index()
    x.columns = ['features', 'p_values']

    if sorted_:
        return x.sort_values(
            by='p_values',
            ascending=False).style.set_precision(4).set_properties(
                **{
                    'background': 'black',
                    'color': 'lawngreen'
                })
    else:
        return x.style.set_precision(4).set_properties(**{
            'background': 'black',
            'color': 'lawngreen'
        }).background_gradient('Reds')
```

[11]:
```
def num_col_for_plotting(row, col=3):
    """
    Returns number of rows to plot

    Parameters:
    ===========
    row = int;
    col = int; default col: 3
    +++ formatting helper function +++
    """
    if row % col != 0:
        return (row // col) + 1
    else:
```

```
        return row // col
```

[12]:
```
def test_for_linearity(df, target='price'):
    """
    Test for the linearity assumption among features

    Parameters:
    ===========
    df      = pandas.Dataframe object
    target  = str; default target: 'price'
    """
    fig, axes = plt.subplots(nrows=num_col_for_plotting(
        len(df.drop(target, axis=1).columns)),
                             ncols=3,
                             figsize=(16, 25),
                             sharey=True)

    for ax, column in zip(axes.flatten(), df.drop(target, axis=1).columns):
        sns.scatterplot(x=df[column],
                        y=df['price'],# / 100_000,
                        markers='H',
                        palette="pastel",
                        color='silver',
                        alpha=.5,
                        ax=ax)
        ax.set_title(f'{target.title()} vs {column.title()}')
        sns.despine()
        plt.tight_layout()

    plt.suptitle(y=1,
                 t='Test for the linearity assumption',
                 va='bottom',
                 size=20,
                 weight=14)
    plt.show()
```

[13]:
```
def find_outliers_Z(data):
    """
    ### Study Group function -- not reinventing the wheel ###
    Link:

    ----------------------------------------------------------
    Use scipy to calculate absolute Z-scores
    and return boolean series where True indicates it is an outlier.

    Args:
        data (Series,or ndarray): data to test for outliers.
```

```
    Returns:
        [boolean Series]: A True/False for each row use to slice outliers.

    EXAMPLE USE:
    >> idx_outs = find_outliers_df(df['AdjustedCompensation'])
    >> good_data = df[~idx_outs].copy()
    """
    import pandas as pd
    import numpy as np
    import scipy.stats as stats
    import pandas as pd
    import numpy as np
    ## Calculate z-scores
    zs = stats.zscore(data)

    ## Find z-scores >3 awayfrom mean
    idx_outs = np.abs(zs)>3

    ## If input was a series, make idx_outs index match
    if isinstance(data,pd.Series):
        return pd.Series(idx_outs,index=data.index)
    else:
        return pd.Series(idx_outs)
```

[14]:
```
def find_outliers_IQR(data):
    """
    ### Study Group function -- not reinventing the wheel ###
    Link:

    ---------------------------------------------------------------
    Use Tukey's Method of outlier removal AKA InterQuartile-Range Rule
    and return boolean series where True indicates it is an outlier.
    - Calculates the range between the 75% and 25% quartiles
    - Outliers fall outside upper and lower limits, using a treshold of  1.
 ↪5*IQR the 75% and 25% quartiles.

    IQR Range Calculation:
        res = df.describe()
        IQR = res['75%'] -  res['25%']
        lower_limit = res['25%'] - 1.5*IQR
        upper_limit = res['75%'] + 1.5*IQR

    Args:
        data (Series,or ndarray): data to test for outliers.

    Returns:
        [boolean Series]: A True/False for each row use to slice outliers.
```

```
    EXAMPLE USE:
    >> idx_outs = find_outliers_df(df['AdjustedCompensation'])
    >> good_data = df[~idx_outs].copy()

    """
    df_b=data
    res= df_b.describe()

    IQR = res['75%'] -  res['25%']
    lower_limit = res['25%'] - 1.5*IQR
    upper_limit = res['75%'] + 1.5*IQR

    idx_outs = (df_b>upper_limit) | (df_b<lower_limit)

    return idx_outs
```

[15]:
```
def check_outliers_in_df(df,
                         chart_type='boxplot',
                         show_dfs=True,
                         turn_off_boxplot=False):
    """
    Returns value counts of the DataFrame columns and a boxplot or boxenplot.
    Use:
        Check for outliers in the dataset.

    Parameters:
    ===========
    df                = pandas.DataFrame object.
    chart_type        = str: either 'boxplot' or 'boxenplot'
    turn_off_boxplot  = boolean; default: False.
        `True` shows plot.
        `False`  does not plot.
    """
    if show_dfs:
        print('DataFrame Columns value counts.')
        for column in df:
            print(f"{'_'*60}")
            print(
                f'{column}: ({df[column].nunique(dropna=False)} unique values␣
↪including NaN)'
            )
            print(df[column].value_counts(dropna=False))
            print(f"{'_'*60}")

    if turn_off_boxplot == False:
        print('')
        print(f"{'_'*60}")
```

```
        print(f'DataFrame Columns {chart_type.title()}')
        print(f"{'_'*60}")

        # default col for chart is 3, to be able to single or 2 features
        # this cells are used
        n = 3
        w = 16
        h = 25
        if len(df.columns) < n:
            n = len(df.columns)
            w = 8
            h = 5
        if len(df.columns) == 1:
            if chart_type == 'boxplot':
                sns.boxplot(x=df.columns[0], data=df, color='gold')
            if chart_type == 'boxenplot':
                sns.boxenplot(x=df.columns[0], data=df, color='gold')
            sns.despine()
            plt.tight_layout()
        # for features more than 2
        else:
            fig, axes = plt.subplots(nrows=num_col_for_plotting(len(
                df.columns),
                                                                col=n),
                                     ncols=n,
                                     figsize=(w, h),
                                     sharey=True)
            for ax, column in zip(axes.flatten(), df):
                if chart_type == 'boxplot':
                    sns.boxplot(x=column, data=df, color='gold', ax=ax)
                if chart_type == 'boxenplot':
                    sns.boxenplot(x=column, data=df, color='gold', ax=ax)
                ax.set_title(
                    f'{chart_type.title()} of distribution of {column.title()}'
                )
                sns.despine()
                plt.tight_layout()
```

[16]:
```
def Distance(lat1, lons1, lat2, lons2):
    """
    Calculates distance in kilometere between two places given lattitude and␣
↪longitude with consideration of earths curvature.

    Source:
    https://stackoverflow.com/questions/27928/
↪calculate-distance-between-two-latitude-longitude-points-haversine-formula
    """
```

```
    import math
    a = 6378.137  #equitorial radius in km
    b = 6356.752  #polar radius in km
    lat1 = math.radians(lat1)
    lons1 = math.radians(lons1)
    R = (((((a**2) * math.cos(lat1))**2) + (((b**2) * math.sin(lat1))**2)) /
         ((a * math.cos(lat1))**2 +
          (b * math.sin(lat1))**2))**0.5  #radius of earth at lat1
    x1 = R * math.cos(lat1) * math.cos(lons1)
    y1 = R * math.cos(lat1) * math.sin(lons1)
    z1 = R * math.sin(lat1)

    lat2 = math.radians(lat2)
    lons2 = math.radians(lons2)
    R = (((((a**2) * math.cos(lat2))**2) + (((b**2) * math.sin(lat2))**2)) /
         ((a * math.cos(lat2))**2 +
          (b * math.sin(lat2))**2))**0.5  #radius of earth at lat2
    x2 = R * math.cos(lat2) * math.cos(lons2)
    y2 = R * math.cos(lat2) * math.sin(lons2)
    z2 = R * math.sin(lat2)

    return ((x1 - x2)**2 + (y1 - y2)**2 +
            (z1 - z2)**2)**0.5  # * 0.621371 for mile
```

[17]:
```
def distance_(df, lat, long):
    """
    +++ Helper Function +++
    Takes a DataFrame conatining lat and long values and calculates distance in␣
 ↪miles (rounded to two decimal place) from Seattle downtown.
    Seattle downtown latitude = 47.6062, longitude = -122.3321
    """
    p = df[lat].to_list()
    k = df[long].to_list()
    temp_list = []
    for x in zip(p, k):
        z = {
            'distance_from_downtown_mile':
            round(Distance(x[0], x[1], 47.6062, -122.3321) * 0.621371, 2)
        }
        temp_list.append(z)
    df_ = pd.DataFrame(temp_list)
    return df_
```

### 2.2.2 Plotly Charts

[18]:
```
def get_location_interactive(mapbox_style="open-street-map"):
    """
    +++ Predefined function +++
    Returns a map with markers for houses based on lat and long.

    Parameters:
    ===========
    mapbox_style = str; options are following:
        > "white-bg" yields an empty white canvas which results in no external␣
↪HTTP requests

        > "carto-positron", "carto-darkmatter", "stamen-terrain",
          "stamen-toner" or "stamen-watercolor" yield maps composed of raster␣
↪tiles
          from various public tile servers which do not require signups or␣
↪access tokens

        > "open-street-map" does work
    """
    fig = px.scatter_mapbox(df,
                            lat=df.lat,
                            lon=df.long,color='price',
                           ␣
↪color_continuous_scale=["green",'blue','red','gold'],
                            zoom=8.5,range_color=[0,df['price'].quantile(0.95)],
                            height=700,
                            title='House location',
                            opacity=.5,
                            center={
                                'lat': df.lat.mode()[0],
                                'lon': df.long.mode()[0]
                            })
    fig.update_layout(mapbox_style=mapbox_style)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    fig.show()
```

[19]:
```
def get_location_static(mapbox_style='stamen-terrain'):
    """
    +++ Predefined function +++
    Returns a map with markers for houses based on lat and long.

    Parameters:
    ===========
    mapbox_style = str; options are following:
```

```
        > "white-bg" yields an empty white canvas which results in no external␣
↪HTTP requests

        > "carto-positron", "carto-darkmatter", "stamen-terrain",
          "stamen-toner" or "stamen-watercolor" yield maps composed of raster␣
↪tiles
          from various public tile servers which do not require signups or␣
↪access tokens

        > "open-street-map" does not work
    """

    fig = px.scatter_mapbox(df,
                            lat=df.lat,
                            lon=df.long,color='price',
                           ␣
↪color_continuous_scale=["green",'blue','red','gold'],
                            zoom=8.5,range_color=[0,df['price'].quantile(0.95)],
                            height=700,
                            title='House location',
                            opacity=.5,
                            center={
                                'lat': df.lat.mode()[0],
                                'lon': df.long.mode()[0]
                            })
    fig.update_layout(mapbox_style=mapbox_style)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    import plotly.io as plyIo
    img_bytes = fig.to_image(format="png", width=1200, height=700, scale=1)
    from IPython.display import Image
    display(Image(img_bytes))
```

[20]:
```
def average_price_by_zipcode_static(mapbox_style='carto-darkmatter'):
    """
    +++ Predefined function +++
    Returns a map with box based on zipcode for average house prices.

    Parameters:
    ===========
    mapbox_style = str; options are following:
        > "white-bg" yields an empty white canvas which results in no external␣
↪HTTP requests

        > "carto-positron", "carto-darkmatter", "stamen-terrain",
          "stamen-toner" or "stamen-watercolor" yield maps composed of raster␣
↪tiles
```

```
        from various public tile servers which do not require signups or␣
↪access tokens

        > "open-street-map" does not work
    ----------------------------------------------------------------------
    Uses geodata from:
    https://github.com/OpenDataDE/State-zip-code-GeoJSON/blob/master/
↪wa_washington_zip_codes_geo.min.json
    """
    # geodata in form of geojson file
    # retrived from https://github.com/OpenDataDE/State-zip-code-GeoJSON/blob/
↪master/wa_washington_zip_codes_geo.min.json
    washinton_zip = json.load(
        open('./data/wa_washington_zip_codes_geo.min.json', 'r'))
    # extract zipcode for id matching
    for feature in washinton_zip['features']:
        feature['id'] = feature['properties']['ZCTA5CE10']

    fig = px.choropleth_mapbox(
        data_frame=df_mean_price_by_zip,
        locations='zipcode',
        geojson=washinton_zip,
        color='price',
        mapbox_style=mapbox_style,
        zoom=8.5,
        height=800,
        color_continuous_scale=['green', 'blue', 'red', 'gold'],
        title='Zipcode by Average Price',
        labels={'price': 'Average House Price'},
        opacity=.7,
        center={
            'lat': df.lat.mode()[0],
            'lon': df.long.mode()[0]
        })
    fig.update_geos(fitbounds='locations', visible=True)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    #     fig.show()
    import plotly.io as plyIo
    img_bytes = fig.to_image(format="png", width=1400, height=800, scale=1)
    from IPython.display import Image
    display(Image(img_bytes))
```

[21]:
```
def average_price_by_zipcode_interactive(mapbox_style="carto-darkmatter"):
    """
    +++ Predefined function +++
    Returns a map with box based on zipcode for average house prices.
```

```
    Parameters:
    ==========
    mapbox_style = str; options are following:
        > "white-bg" yields an empty white canvas which results in no external␣
↪HTTP requests

        > "carto-positron", "carto-darkmatter", "stamen-terrain",
          "stamen-toner" or "stamen-watercolor" yield maps composed of raster␣
↪tiles
          from various public tile servers which do not require signups or␣
↪access tokens

        > "open-street-map" does work

    ----------------------------------------------------------------------
    Uses geodata from:
    https://github.com/OpenDataDE/State-zip-code-GeoJSON/blob/master/
↪wa_washington_zip_codes_geo.min.json
    """
    # geodata in form of geojson file
    # retrived from https://github.com/OpenDataDE/State-zip-code-GeoJSON/blob/
↪master/wa_washington_zip_codes_geo.min.json
    washinton_zip = json.load(
        open('./data/wa_washington_zip_codes_geo.min.json', 'r'))
    # extract zipcode for id matching
    for feature in washinton_zip['features']:
        feature['id'] = feature['properties']['ZCTA5CE10']

    fig = px.choropleth_mapbox(
        data_frame=df_mean_price_by_zip,
        locations='zipcode',
        geojson=washinton_zip,
        color='price',
        mapbox_style=mapbox_style,
        zoom=8.5,
        height=800,
        color_continuous_scale=['green', 'blue', 'red', 'gold'],
        title='Zipcode by Average Price',
        labels={'price': 'Average House Price'},
        opacity=.7,
        center={
            'lat': df.lat.mode()[0],
            'lon': df.long.mode()[0]
        })
    fig.update_geos(fitbounds='locations', visible=True)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    fig.show()
```

[22]:
```
def average_price_per_sqft_by_zipcode_static(mapbox_style='carto-darkmatter'):
    """
    +++ Predefined function +++
    Returns a map with box based on zipcode for average house prices.

    Parameters:
    ===========
    mapbox_style = str; options are following:
        > "white-bg" yields an empty white canvas which results in no external␣
↪HTTP requests

        > "carto-positron", "carto-darkmatter", "stamen-terrain",
          "stamen-toner" or "stamen-watercolor" yield maps composed of raster␣
↪tiles
          from various public tile servers which do not require signups or␣
↪access tokens

        > "open-street-map" does not work
    ----------------------------------------------------------------------------
    Uses geodata from:
    https://github.com/OpenDataDE/State-zip-code-GeoJSON/blob/master/
↪wa_washington_zip_codes_geo.min.json
    """
    # geodata in form of geojson file
    # retrived from https://github.com/OpenDataDE/State-zip-code-GeoJSON/blob/
↪master/wa_washington_zip_codes_geo.min.json
    washinton_zip = json.load(
        open('./data/wa_washington_zip_codes_geo.min.json', 'r'))
    # extract zipcode for id matching
    for feature in washinton_zip['features']:
        feature['id'] = feature['properties']['ZCTA5CE10']

    fig = px.choropleth_mapbox(
        data_frame=df_mean_price_per_sqft_by_zip,
        locations='zipcode',
        geojson=washinton_zip,
        color='price_per_sqft',
        mapbox_style=mapbox_style,
        zoom=8.5,
        height=800,
        color_continuous_scale=['green', 'blue', 'red', 'gold'],
        title='Zipcode by Average Price',
        labels={'price': 'Average House Price'},
        opacity=.7,
        center={
            'lat': df.lat.mode()[0],
            'lon': df.long.mode()[0]
```

```
        })
    fig.update_geos(fitbounds='locations', visible=True)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    #     fig.show()
    import plotly.io as plyIo
    img_bytes = fig.to_image(format="png", width=1400, height=800, scale=1)
    from IPython.display import Image
    display(Image(img_bytes))
```

[23]:
```
def price_vs_bedroom_count():
    """
    +++ Predefined function +++
    """
    chart = df.groupby('bedrooms')[['price']].agg(['count', 'mean'])
    chart.columns = chart.columns.droplevel(0)
    chart.columns = ["count", "data"]
    chart.index.name = 'labels'
    chart = chart.reset_index()
    charts = [
        go.Bar(x=chart['labels'].values,
               y=chart['data'].values,
               marker_color='silver',
               name='Bedrooms'),
        go.Scatter(x=chart['labels'].values,
                   y=chart['count'].values,
                   yaxis='y2',
                   name='Average Price',
                   line={
                       'shape': 'spline',
                       'smoothing': 0.3
                   },
                   mode='lines',
                   marker_color='gold')
    ]
    figure = go.Figure(data=charts,
                       layout=go.Layout({
                           'barmode': 'group',
                           'legend': {
                               'xanchor': 'left',
                               'x': .9,
                               'y': 1.02,
                               'yanchor': 'bottom',
                               'orientation': 'h'
                           },
                           'title': {
                               'text': 'price(mean) Categorized by bedrooms'
                           },
```

```
                            'xaxis': {
                                'dtick': 1,
                                'title': {
                                    'text': 'bedrooms'
                                }
                            },
                            'yaxis': {
                                'side': 'left',
                                'title': {
                                    'text': 'price (mean)'
                                }
                            },
                            'yaxis2': {
                                'overlaying': 'y',
                                'side': 'right',
                                'title': {
                                    'text': 'Bedrooms Count'
                                }
                            }
                        }))
    figure.update_layout(template="plotly_dark")
    figure.show()
```

[24]:
```
def price_vs_year_built():
    """
    +++ Predefined function +++
    """
    chart = df.groupby('yr_built')[['price']].agg(['count', 'mean'])
    chart.columns = chart.columns.droplevel(0)
    chart.columns = ["count", "data"]
    chart.index.name = 'labels'
    chart = chart.reset_index()
    chart = chart
    charts = [
        go.Bar(x=chart['labels'].values,
               y=chart['data'].values,
               name='Price',
               marker_color='silver'),
        go.Scatter(x=chart['labels'].values,
                   y=chart['count'].values,
                   yaxis='y2',
                   name='Frequency',
                   mode='lines+markers',
                   marker_color='red')
    ]
    figure = go.Figure(data=charts,
                       layout=go.Layout({
```

```
                        'barmode': 'overlay',
                        'legend': {
                            'xanchor': 'left',
                            'x': .9,
                            'y': 1.02,
                            'yanchor': 'bottom',
                            'orientation': 'h'
                        },
                        'title': {
                            'text':
                            'Average price Categorized by year built'
                        },
                        'xaxis': {
                            'dtick':
                            1,
                            'title': {
                                'text': 'year built'
                            },
                            'range': [
                                chart['labels'].values.min() - 1,
                                chart['labels'].values.max() + 1
                            ]
                        },
                        'yaxis': {
                            'side': 'left',
                            'title': {
                                'text': 'Average price'
                            }
                        },
                        'yaxis2': {
                            'overlaying': 'y',
                            'side': 'right',
                            'title': {
                                'text': 'Sale Frequency'
                            }
                        }
                    }))

figure.update_layout(template="plotly_dark")
figure.show()
```

# 3 OBTAIN

Data for this analysis was provided as part of phase two project of The Flatiron School Full Time Online Data Science program. This a fork of that. The csv file named `kc_house_data.csv` contains following information in this repository located at `./data/kc_house_data.csv`.

GeoJson file used to get map is soured from here provided by Open Data Delaware. A copy of that can be found at `./data/wa_washington_zip_codes_geo.min.json'` in this repository.

### 3.1 Column Names and descriptions for Kings County Data Set

As the readme file accompanying the dataset.

- **id** - unique identified for a house
- **dateDate** - house was sold
- **pricePrice** - is prediction target
- **bedroomsNumber** - of Bedrooms/House
- **bathroomsNumber** - of bathrooms/bedrooms
- **sqft_livingsquare** - footage of the home
- **sqft_lotsquare** - footage of the lot
- **floorsTotal** - floors (levels) in house
- **waterfront** - House which has a view to a waterfront
- **view** - Has been viewed
- **condition** - How good the condition is ( Overall )
- **grade** - overall grade given to the housing unit, based on King County grading system
- **sqft_above** - square footage of house apart from basement
- **sqft_basement** - square footage of the basement
- **yr_built** - Built Year
- **yr_renovated** - Year when house was renovated
- **zipcode** - zip
- **lat** - Latitude coordinate
- **long** - Longitude coordinate
- **sqft_living15** - The square footage of interior housing living space for the nearest 15 neighbors
- **sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors

Although explanation and data contained in **view** does not make any real life sense. It is very unlikely that so many house is sold without inspection, with keeping in mind that some folks buy houses for investment purpose, still some kind of representative visits and facilitate buying process. All other features are supportive of the explanation given and data contained with some extreme outliers, e.g. one house has 33 bedroom with an disproportionate sqft.

## 4 SCRUB & EXPLORE

[25]:
```
# loading data
df = pd.read_csv('./data/kc_house_data.csv')
```

[26]:
```
# observe subset of data
df.sample(10)
```

[26]:

| | id | date | price | … | long | sqft_living15 | sqft_lot15 |
|---|---|---|---|---|---|---|---|
| 12488 | 9211500730 | 2/18/2015 | 162000.0 | … | -122.377 | 1690 | 7770 |
| 6571 | 1250202285 | 10/20/2014 | 908800.0 | … | -122.290 | 980 | 6300 |
| 20681 | 3449000200 | 5/8/2015 | 360000.0 | … | -122.147 | 1720 | 8475 |

```
9181   5100402644  11/19/2014  525000.0  … -122.319          1570        6380
4546   7165700110   5/7/2015   280000.0  … -122.282          1450        1461
2088   3380900160  4/28/2015   502000.0  … -122.359          1750        8475
19030  5631501161  4/17/2015   425000.0  … -122.235          1590        9900
20378  3362401758   9/3/2014   467000.0  … -122.348          1350        1415
20146  7967000130   4/1/2015   370228.0  … -122.275          2050        4000
12913  3293700482  1/14/2015   509000.0  … -122.353          1780        8545

[10 rows x 21 columns]
```

[27]:
```
# columns inside DataFrame
df.columns
```

[27]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

[28]:
```
df.shape
```

[28]:
```
(21597, 21)
```

[29]:
```
# checking for data type
df.dtypes
```

[29]:
```
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
```

[30]:
```
check_NaN(df)
```

[30]:

| | name | is_null | not_null |
|---|---|---|---|
| 0 | id | 0 | 21597 |
| 1 | date | 0 | 21597 |
| 2 | price | 0 | 21597 |
| 3 | bedrooms | 0 | 21597 |
| 4 | bathrooms | 0 | 21597 |
| 5 | sqft_living | 0 | 21597 |
| 6 | sqft_lot | 0 | 21597 |
| 7 | floors | 0 | 21597 |
| 8 | waterfront | 2376 | 19221 |
| 9 | view | 63 | 21534 |
| 10 | condition | 0 | 21597 |
| 11 | grade | 0 | 21597 |
| 12 | sqft_above | 0 | 21597 |
| 13 | sqft_basement | 0 | 21597 |
| 14 | yr_built | 0 | 21597 |
| 15 | yr_renovated | 3842 | 17755 |
| 16 | zipcode | 0 | 21597 |
| 17 | lat | 0 | 21597 |
| 18 | long | 0 | 21597 |
| 19 | sqft_living15 | 0 | 21597 |
| 20 | sqft_lot15 | 0 | 21597 |

[31]:
```
check_duplicates(df)
```

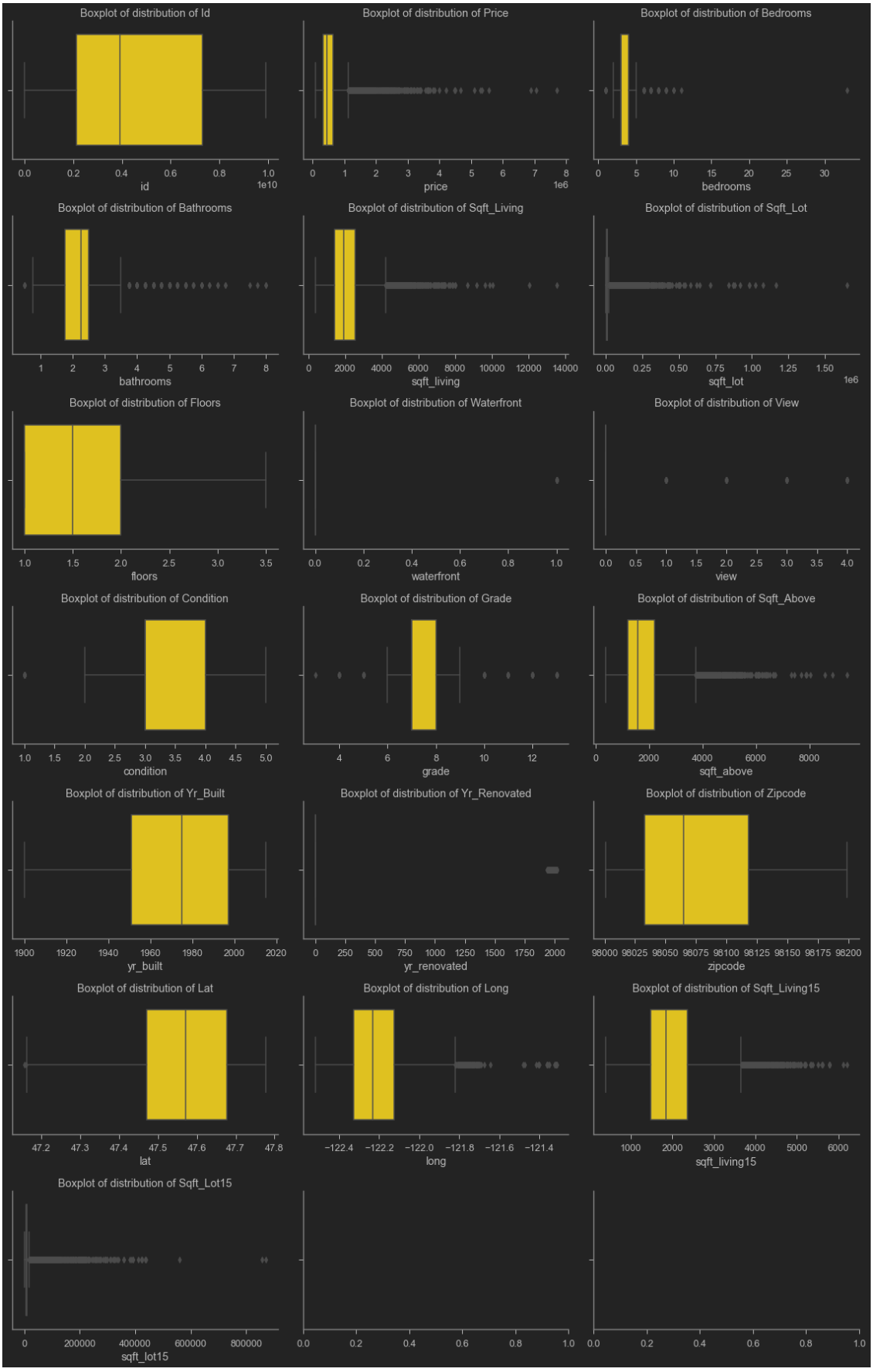
[31]:

| | name | duplicated | not_duplicated |
|---|---|---|---|
| 0 | id | 177 | 21420 |
| 1 | date | 21225 | 372 |
| 2 | price | 17975 | 3622 |
| 3 | bedrooms | 21585 | 12 |
| 4 | bathrooms | 21568 | 29 |
| 5 | sqft_living | 20563 | 1034 |
| 6 | sqft_lot | 11821 | 9776 |
| 7 | floors | 21591 | 6 |
| 8 | waterfront | 21594 | 3 |
| 9 | view | 21591 | 6 |
| 10 | condition | 21592 | 5 |
| 11 | grade | 21586 | 11 |
| 12 | sqft_above | 20655 | 942 |
| 13 | sqft_basement | 21293 | 304 |
| 14 | yr_built | 21481 | 116 |
| 15 | yr_renovated | 21526 | 71 |
| 16 | zipcode | 21527 | 70 |

```
17            lat       16564            5033
18           long       20846             751
19  sqft_living15       20820             777
20     sqft_lot15       12915            8682
```

[32]:
```
# 'date' and 'sqft_basement' both has incorrect dtype
check_outliers_in_df(df.drop(['date','sqft_basement'],axis=1),show_dfs=False)
```

```
------------------------------------------------------------
DataFrame Columns Boxplot
------------------------------------------------------------
```


Boxplot of distribution of Id
id
1e10
Boxplot of distribution of Price
price
1e6
Boxplot of distribution of Bedrooms
bedrooms
Boxplot of distribution of Bathrooms
bathrooms
Boxplot of distribution of Sqft_Living
sqft_living
Boxplot of distribution of Sqft_Lot
sqft_lot
1e6
Boxplot of distribution of Floors
floors
Boxplot of distribution of Waterfront
waterfront
Boxplot of distribution of View
view
Boxplot of distribution of Condition
condition
Boxplot of distribution of Grade
grade
Boxplot of distribution of Sqft_Above
sqft_above
Boxplot of distribution of Yr_Built
yr_built
Boxplot of distribution of Yr_Renovated
yr_renovated
Boxplot of distribution of Zipcode
zipcode
Boxplot of distribution of Lat
lat
Boxplot of distribution of Long
long
Boxplot of distribution of Sqft_Living15
sqft_living15
Boxplot of distribution of Sqft_Lot15
sqft_lot15

[33]:
```
# looking at numerical features
df.describe().transpose().style.format("{0:,.0f}")
```

[33]: <pandas.io.formats.style.Styler at 0x1c24dac7430>

### 4.1 Initial cleaning

[34]:
```
# cleaning data

# drop duplicates based on id column
df = df[~df.duplicated(['id'], keep='first')]
# reseting index
df = df.reset_index().drop('index', axis=1, errors='ignore')

# converting date to datetime
df.loc[:, 'date'] = pd.to_datetime(df['date'], infer_datetime_format=True)

# nan and error handeling
# waterfront
df.loc[:, 'waterfront'] = df['waterfront'].replace({np.nan: 0})
df.loc[:, 'waterfront'] = df['waterfront'].astype('int')
# view
df.loc[:, 'view'] = df['view'].replace({np.nan: 0})
# yr_renovated
df.loc[:, 'yr_renovated'] = df['yr_renovated'].replace({np.nan: 0})
df.loc[:, 'yr_renovated'] = df['yr_renovated'].astype('int')
# sqft_basement
df.loc[:, 'sqft_basement'] = df['sqft_basement'].replace({'?': 0})
df.loc[:, 'sqft_basement'] = pd.to_numeric(df['sqft_basement'],
                                           errors="coerce")
df.loc[:, 'sqft_basement'] = df['sqft_basement'].astype('int')
```

[35]:
```
# droping possible error in input
df = df[df['bedrooms']!=33]
# reseting index
df = df.reset_index().drop('index', axis=1, errors='ignore')
```

[36]:
```
# after cleaning
display(check_NaN(df), check_duplicates(df),
        df.describe().transpose().style.format("{0:,.0f}"))
```

| | name | is_null | not_null |
|---|---|---|---|
| 0 | id | 0 | 21419 |
| 1 | date | 0 | 21419 |
| 2 | price | 0 | 21419 |

```
3        bedrooms        0     21419
4       bathrooms        0     21419
5     sqft_living        0     21419
6        sqft_lot        0     21419
7          floors        0     21419
8      waterfront        0     21419
9            view        0     21419
10      condition        0     21419
11          grade        0     21419
12     sqft_above        0     21419
13  sqft_basement        0     21419
14       yr_built        0     21419
15   yr_renovated        0     21419
16        zipcode        0     21419
17            lat        0     21419
18           long        0     21419
19  sqft_living15        0     21419
20     sqft_lot15        0     21419
```

| | name | duplicated | not_duplicated |
|---|---|---|---|
| 0 | id | 0 | 21419 |
| 1 | date | 21047 | 372 |
| 2 | price | 17812 | 3607 |
| 3 | bedrooms | 21408 | 11 |
| 4 | bathrooms | 21390 | 29 |
| 5 | sqft_living | 20385 | 1034 |
| 6 | sqft_lot | 11643 | 9776 |
| 7 | floors | 21413 | 6 |
| 8 | waterfront | 21417 | 2 |
| 9 | view | 21414 | 5 |
| 10 | condition | 21414 | 5 |
| 11 | grade | 21408 | 11 |
| 12 | sqft_above | 20477 | 942 |
| 13 | sqft_basement | 21116 | 303 |
| 14 | yr_built | 21303 | 116 |
| 15 | yr_renovated | 21349 | 70 |
| 16 | zipcode | 21349 | 70 |
| 17 | lat | 16386 | 5033 |
| 18 | long | 20668 | 751 |
| 19 | sqft_living15 | 20642 | 777 |
| 20 | sqft_lot15 | 12737 | 8682 |

```
<pandas.io.formats.style.Styler at 0x1c24d4481f0>
```

### 4.2 Feature engineering

NOTE: by "k" i mean each house and "n" means total house.

### 4.2.1 Distance from downtown Seattle

Seattle downtown coordinate for this is considered at * 47.6062° N, * 122.3321° W

[37]:
```
df_distace = distance_(df, 'lat', 'long')
df = pd.concat([df, df_distace], axis=1, ignore_index=False)
df
```

[37]:

| | id | date | … | sqft_lot15 | distance_from_downtown_mile |
|---|---|---|---|---|---|
| 0 | 7129300520 | 2014-10-13 | … | 5650 | 7.43 |
| 1 | 6414100192 | 2014-12-09 | … | 7639 | 7.95 |
| 2 | 5631500400 | 2015-02-25 | … | 8062 | 10.19 |
| 3 | 2487200875 | 2014-12-09 | … | 5000 | 6.54 |
| 4 | 1954400510 | 2015-02-18 | … | 7503 | 13.38 |
| … | … | … | … | … | … |
| 21414 | 263000018 | 2014-05-21 | … | 1509 | 6.46 |
| 21415 | 6600060120 | 2015-02-23 | … | 7200 | 6.74 |
| 21416 | 1523300141 | 2014-06-23 | … | 2007 | 1.74 |
| 21417 | 291310100 | 2015-01-16 | … | 1287 | 13.22 |
| 21418 | 1523300157 | 2014-10-15 | … | 1357 | 1.75 |

[21419 rows x 22 columns]

### 4.2.2 Price per sqft

Ratio of price of each house over total sqft of that.

[38]:
```
df['price_per_sqft'] = round(df['price']/(df['sqft_lot']+df['sqft_living']),2)
```

[39]:
```
df.loc[:, 'price_per_sqft'] = df['price_per_sqft'].astype('str')
df.loc[:, 'price_per_sqft'] = df['price_per_sqft'].astype('float')
```

[40]:
```
df.price_per_sqft
```

[40]:
```
0         32.49
1         54.83
2         16.71
3         86.78
4         52.25
          …
21414    135.29
21415     49.24
21416    169.66
21417    100.30
21418    155.06
Name: price_per_sqft, Length: 21419, dtype: float64
```

### 4.2.3 Neighborhood

boolean feature to check that Total sqft of n is with in 40% to 60% range of average of total sqft of surrounding 15 house to detect unusual house in the neighborhood. Check for unusual homes.

[41]:
```
sum_sqft_n = (df['sqft_lot'] + df['sqft_living'])
sum_sqft_15 = (df['sqft_lot15'] + df['sqft_living15'])
df['total_sqft_larger_than_neighbours'] = ((sum_sqft_15 * .4 <= sum_sqft_n) &
                                           (sum_sqft_n <= sum_sqft_15 * .6))
df['total_sqft_larger_than_neighbours'] = df[
    'total_sqft_larger_than_neighbours'].apply(lambda x: 1 if x == True else 0)
print(df['total_sqft_larger_than_neighbours'].dtype)
df
```

```
int64
```

[41]:

| | id | date | … | price_per_sqft | total_sqft_larger_than_neighbours |
|---|---|---|---|---|---|
| 0 | 7129300520 | 2014-10-13 | … | 32.49 | 0 |
| 1 | 6414100192 | 2014-12-09 | … | 54.83 | 0 |
| 2 | 5631500400 | 2015-02-25 | … | 16.71 | 0 |
| 3 | 2487200875 | 2014-12-09 | … | 86.78 | 0 |
| 4 | 1954400510 | 2015-02-18 | … | 52.25 | 0 |
| … | … | … | … | … | … |
| 21414 | 263000018 | 2014-05-21 | … | 135.29 | 0 |
| 21415 | 6600060120 | 2015-02-23 | … | 49.24 | 0 |
| 21416 | 1523300141 | 2014-06-23 | … | 169.66 | 0 |
| 21417 | 291310100 | 2015-01-16 | … | 100.30 | 0 |
| 21418 | 1523300157 | 2014-10-15 | … | 155.06 | 0 |

[21419 rows x 24 columns]

### 4.2.4 Is Renovated

[42]:
```
print(
```

```
    f'Out of {len(df.yr_renovated)} values, {df.yr_renovated.value_counts()[0]}␣
↪are empty while only {df.yr_renovated.value_counts()[1:].sum()} rows␣
↪containing year renovated. This is not usefull for the model. So this␣
↪feature is converted to a boolean, where 1 means it is renovated and 0 means␣
↪it not renovated.'
)
```

Out of 21419 values, 20679 are empty while only 740 rows containing year renovated. This is not usefull for the model. So this feature is converted to a boolean, where 1 means it is renovated and 0 means it not renovated.

[43]:
```
df['is_renovated'] = df['yr_renovated'].apply(lambda x: 0
                                              if x == 0 else 1)
```

[44]:
```
# checking that this worked by showing value counts
df['is_renovated'].value_counts()
```

[44]:
```
0    20679
1      740
Name: is_renovated, dtype: int64
```

### 4.3 First look at the cleaned dataset

[45]:
```
df
```

[45]:
```
                id       date  …  total_sqft_larger_than_neighbours
is_renovated
0       7129300520 2014-10-13  …                                  0
0
1       6414100192 2014-12-09  …                                  0
1
2       5631500400 2015-02-25  …                                  0
0
3       2487200875 2014-12-09  …                                  0
0
4       1954400510 2015-02-18  …                                  0
0
…              …         …  …                                  …
…
21414    263000018 2014-05-21  …                                  0
0
21415   6600060120 2015-02-23  …                                  0
0
21416   1523300141 2014-06-23  …                                  0
0
21417    291310100 2015-01-16  …                                  0
0
21418   1523300157 2014-10-15  …                                  0
```

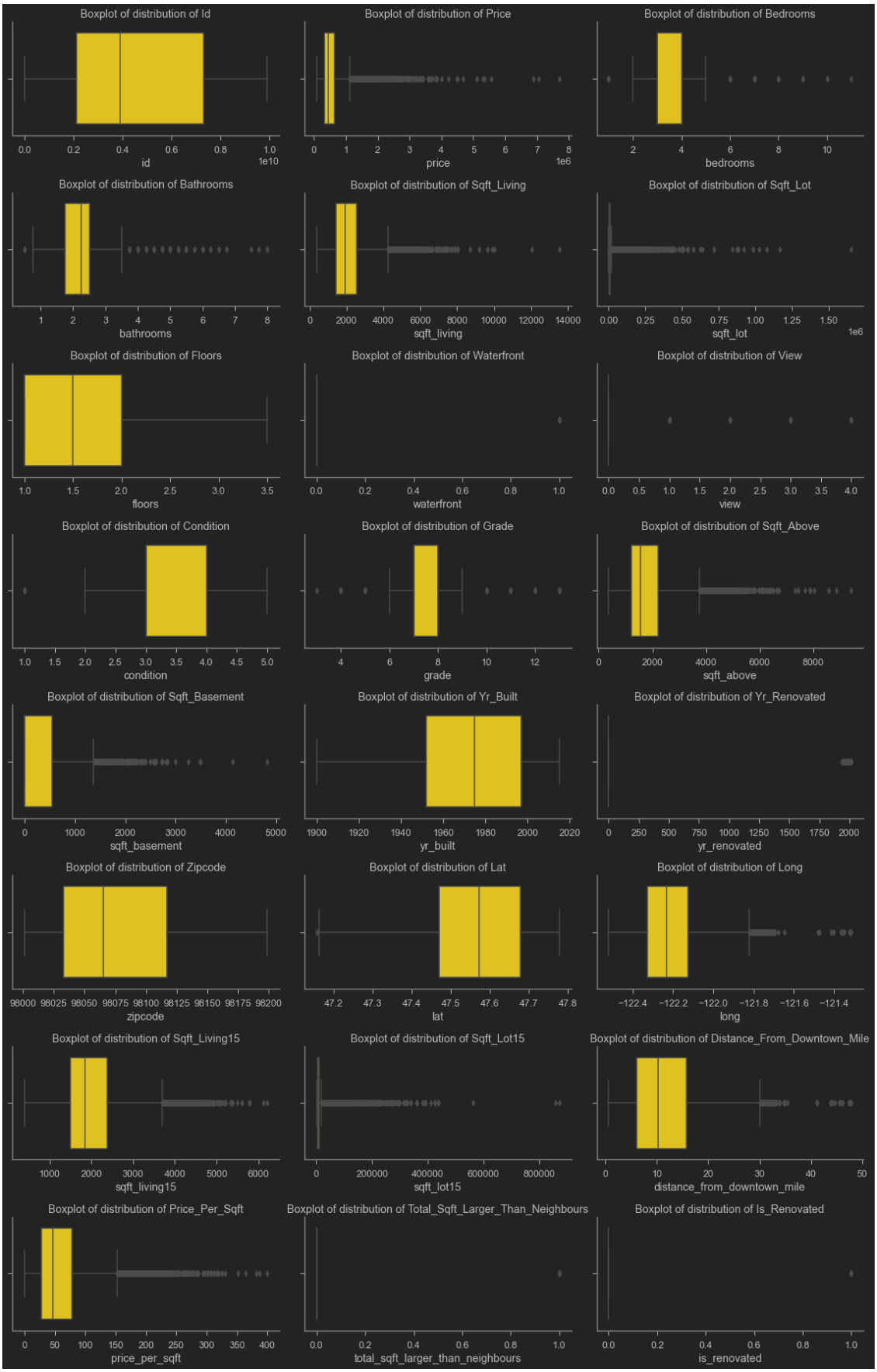
```
0

[21419 rows x 25 columns]
```

[46]:
```
fig, axes = plt.subplots(nrows=num_col_for_plotting(len(df.columns), col=3),
                         ncols=3,
                         figsize=(16, 26),
                         sharey=False)
for ax, column in zip(axes.flatten(), df):
    sns.histplot(x=column, data=df, color='gold', ax=ax, kde=True)
    ax.set_title(f'Histplot of {column.title()}')
    ax.tick_params('x', labelrotation=45)
    sns.despine()
    plt.tight_layout()
    plt.suptitle('Histogram plots of the dataset',
                 fontsize=20,
                 fontweight=3,
                 va='bottom')
```


Histogram plots of the dataset
Histplot of Id
Histplot of Date
Histplot of Price
Histplot of Bedrooms
Histplot of Bathrooms
Histplot of Sqft_Living
Histplot of Sqft_Lot
Histplot of Floors
Histplot of Waterfront
Histplot of View
Histplot of Condition
Histplot of Grade
Histplot of Sqft_Above
Histplot of Sqft_Basement
Histplot of Yr_Built
Histplot of Yr_Renovated
Histplot of Zipcode
Histplot of Lat
Histplot of Long
Histplot of Sqft_Living15
Histplot of Sqft_Lot15
Histplot of Distance_From_Downtown_Mile
Histplot of Price_Per_Sqft
Histplot of Total_Sqft_Larger_Than_Neighbours
Histplot of Is_Renovated
Count
id
date
price
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
sqft_above
sqft_basement
yr_built
yr_renovated
zipcode
lat
long
sqft_living15
sqft_lot15
distance_from_downtown_mile
price_per_sqft
total_sqft_larger_than_neighbours
is_renovated

[47]:
```
check_outliers_in_df(df.drop('date', axis=1),
                     chart_type='boxplot',
                     show_dfs=False)
```

```
------------------------------------------------------------
DataFrame Columns Boxplot
------------------------------------------------------------
```


Boxplot of distribution of Id
id
1e10
Boxplot of distribution of Price
price
1e6
Boxplot of distribution of Bedrooms
bedrooms
Boxplot of distribution of Bathrooms
bathrooms
Boxplot of distribution of Sqft_Living
sqft_living
Boxplot of distribution of Sqft_Lot
sqft_lot
1e6
Boxplot of distribution of Floors
floors
Boxplot of distribution of Waterfront
waterfront
Boxplot of distribution of View
view
Boxplot of distribution of Condition
condition
Boxplot of distribution of Grade
grade
Boxplot of distribution of Sqft_Above
sqft_above
Boxplot of distribution of Sqft_Basement
sqft_basement
Boxplot of distribution of Yr_Built
yr_built
Boxplot of distribution of Yr_Renovated
yr_renovated
Boxplot of distribution of Zipcode
zipcode
Boxplot of distribution of Lat
lat
Boxplot of distribution of Long
long
Boxplot of distribution of Sqft_Living15
sqft_living15
Boxplot of distribution of Sqft_Lot15
sqft_lot15
Boxplot of distribution of Distance_From_Downtown_Mile
distance_from_downtown_mile
Boxplot of distribution of Price_Per_Sqft
price_per_sqft
Boxplot of distribution of Total_Sqft_Larger_Than_Neighbours
total_sqft_larger_than_neighbours
Boxplot of distribution of Is_Renovated
is_renovated

## 4.4 Categorical and Numerical features

### 4.4.1 Identifying

[48]:
```
df.columns
```

[48]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15',
       'distance_from_downtown_mile', 'price_per_sqft',
       'total_sqft_larger_than_neighbours', 'is_renovated'],
      dtype='object')
```

[49]:
```
# based on information from the readme file attached to the dataset.
# House identifier
unique_feat_df = ['id']
# Catagorical features
categorical_feat_df = [
    'waterfront', 'view', 'condition', 'grade', 'is_renovated'
]
# Continuous features
numerical_continuous_feat_df = [
    'price', 'sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement',
    'sqft_living15', 'sqft_lot15', 'price_per_sqft',
    'distance_from_downtown_mile', 'total_sqft_larger_than_neighbours',
    'price_per_sqft'
]
numerical_discrete_feat_df = ['bedrooms', 'bathrooms', 'floors']
# Timing features
time_feat_df = ['date', 'yr_built', 'yr_renovated']
# Location feat
location_feat_df = ['lat', 'long', 'zipcode']
```

### 4.4.2 Classification

[50]:
```
# Although zipcode is discrete numerical column, the numbers means nothing in␣
 ↪context of this analysis.
# So zipcode is considered as a cateforical feature.
categorical_feat = categorical_feat_df + ['zipcode']
numerical_feat = numerical_continuous_feat_df + numerical_discrete_feat_df + [
    'yr_built', 'yr_renovated'
] + ['lat', 'long']
dropped = ['id' + 'date' + 'yr_renovated']
```

```
print('categorical_feat:', categorical_feat)
print('numerical_feat:', numerical_feat)
len(categorical_feat + numerical_feat + dropped) == len(df.columns)
```

```
categorical_feat: ['waterfront', 'view', 'condition', 'grade', 'is_renovated',
'zipcode']
numerical_feat: ['price', 'sqft_living', 'sqft_lot', 'sqft_above',
'sqft_basement', 'sqft_living15', 'sqft_lot15', 'price_per_sqft',
'distance_from_downtown_mile', 'total_sqft_larger_than_neighbours',
'price_per_sqft', 'bedrooms', 'bathrooms', 'floors', 'yr_built', 'yr_renovated',
'lat', 'long']
```

[50]:
```
True
```

[51]:
```
# checkpoint and fail safe measure
df.to_csv(r'./Data/kc_house_data_df.csv', index=False)
```

### 4.4.3 Dropping

In the Dataframe for initial model `id`, `date`, `yr_renovated`, and `view` is not included. My reasoning for this is following: - `id`: This is the identifier column. - `date`: Sale date is not important input for now as I am not capturing seasonality of value for this analysis. - `yr_renovated`: Already converted to boolean column. - `view`: Not sure about this feature what it really means as not much information is provided. This can also be a categorical or numeric feature.

[52]:
```
# loading data back
df = pd.read_csv('./data/kc_house_data_df.csv')
```

[53]:
```
# make sure everything works
df.columns
```

[53]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15',
       'distance_from_downtown_mile', 'price_per_sqft',
       'total_sqft_larger_than_neighbours', 'is_renovated'],
      dtype='object')
```

[54]:
```
df_model = df.drop(columns=['id', 'date','yr_renovated','view'])
```

[55]:
```
# looking at a random sample
df_model.sample(5)
```

[55]:

| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 17771 | 599950.0 | 2 | … | 0 | 0 |
| 20136 | 2350000.0 | 6 | … | 0 | 0 |
| 10168 | 1250000.0 | 3 | … | 0 | 1 |

```
14106   489900.0         4  …                                    0             0
12394   535500.0         3  …                                    0             0

[5 rows x 21 columns]
```

#### 4.4.4 Initial data for the model

[56]:
```
df_model
```

[56]:
```
          price  bedrooms  …  total_sqft_larger_than_neighbours  is_renovated
0      221900.0         3  …                                  0             0
1      538000.0         3  …                                  0             1
2      180000.0         2  …                                  0             0
3      604000.0         4  …                                  0             0
4      510000.0         3  …                                  0             0
…           …       …  …                                    …             …
21414  360000.0         3  …                                  0             0
21415  400000.0         4  …                                  0             0
21416  402101.0         2  …                                  0             0
21417  400000.0         3  …                                  0             0
21418  325000.0         2  …                                  0             0

[21419 rows x 21 columns]
```

[57]:
```
numerical_feat_model = [
    'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
    'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long', 'sqft_living15',
    'sqft_lot15', 'distance_from_downtown_mile','price_per_sqft'
]
# although all contain numbers to discribe, numbers represent categories.
categorical_feat_model = [
    'waterfront', 'condition', 'grade', 'is_renovated',␣
 ↪'zipcode','total_sqft_larger_than_neighbours'
]
len(numerical_feat_model + categorical_feat_model) == len(df_model.columns)
```

[57]:
```
True
```

### 4.5 Feature relationships

#### 4.5.1 Feature correlation and multicollinearity

[58]:
```
# correlation of features of entire dataset
correlation_top_bottom(df.corr())
```

```
Positive correlations:
   index                  feature_combo  correlation
0    359  is_renovated and yr_renovated     0.999968
```

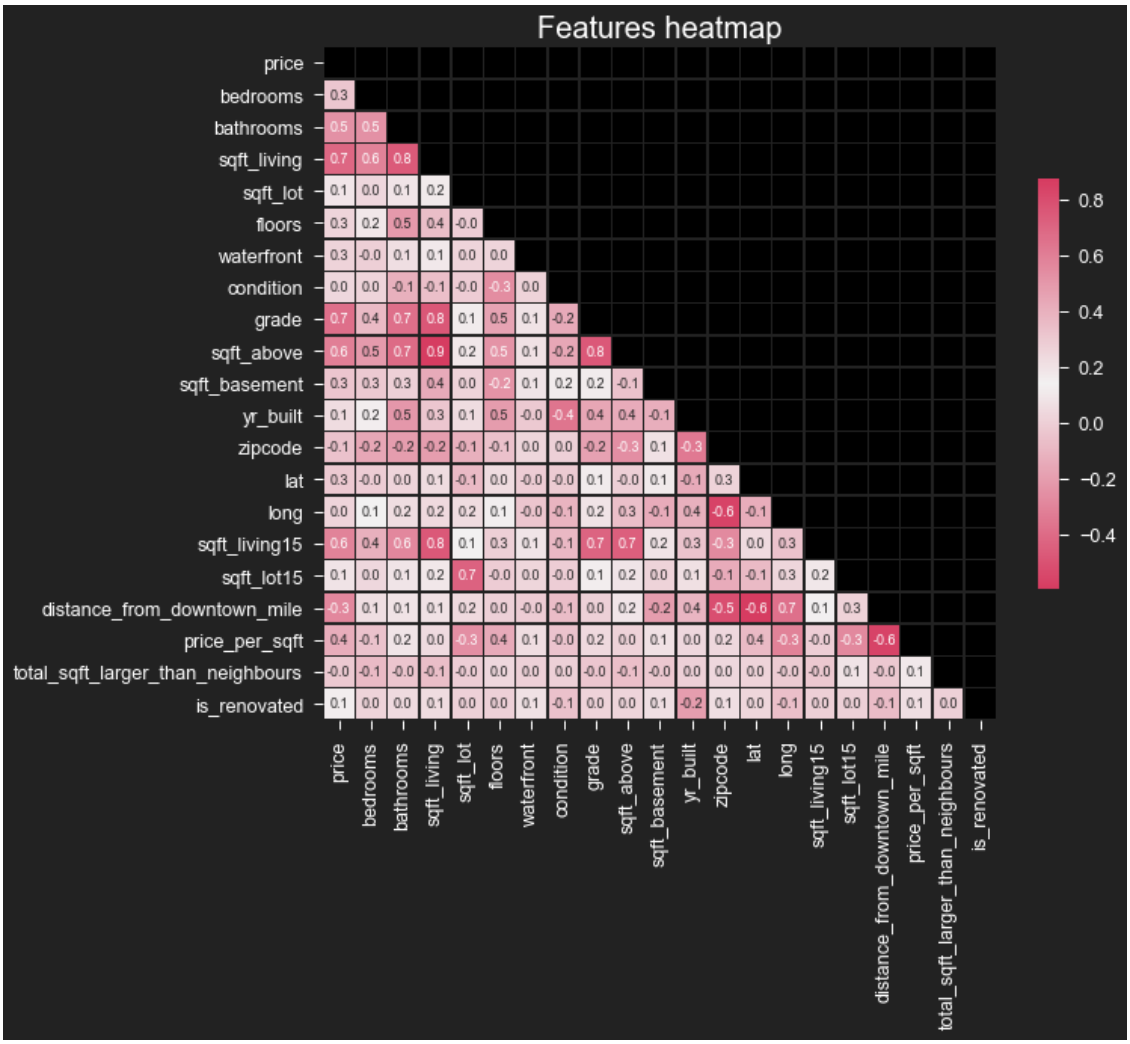
```
1    107     sqft_living and sqft_above     0.876534
2    106          grade and sqft_living     0.762474
3    251           grade and sqft_above     0.756217
4    114  sqft_living and sqft_living15     0.756184
5     76      bathrooms and sqft_living     0.755519
6    282   sqft_above and sqft_living15     0.731877
7    139        sqft_lot and sqft_lot15     0.717742
8    258        grade and sqft_living15     0.713173
9     28          sqft_living and price     0.701887

----------------------------------------------------------------------
Negative correlations:
   index                                    feature_combo  correlation
0    404            distance_from_downtown_mile and lat    -0.595645
1    377                                zipcode and long    -0.564772
2    501  price_per_sqft and distance_from_downtown_mile    -0.561403
3    380        zipcode and distance_from_downtown_mile    -0.519708
4    229                          condition and yr_built    -0.365092
5    327                            zipcode and yr_built    -0.346151
6    429                         price_per_sqft and long    -0.308950
7     44          distance_from_downtown_mile and price    -0.286815
8    477                  price_per_sqft and sqft_lot15    -0.280734
9    378                      zipcode and sqft_living15    -0.278415
```

[59]:
```
# correlation among seleted features for model
top_correlated_features(df_model,verbose=True)
```

[59]:
```
                                    feature_combo  correlation
0                       sqft_living and sqft_above     0.876534
1                            grade and sqft_living     0.762474
2                             grade and sqft_above     0.756217
3                     sqft_living and sqft_living15     0.756184
4                         bathrooms and sqft_living     0.755519
..                                              …          …
221  condition and total_sqft_larger_than_neighbours     0.002456
222                          waterfront and bedrooms     0.002133
223                               lat and sqft_above     0.002001
224                               sqft_above and lat     0.002001
225                   is_renovated and sqft_living15     0.000749

[226 rows x 2 columns]
```

### 4.5.2 Heatmap

[60]:
```
heatmap_DataFrame(df_model)
```




Only sqft features are collinear in nature. This is expected. and has a relationship with grade. Leaving it as it is right now.

## 4.6 Exploring features

### 4.6.1 Individually

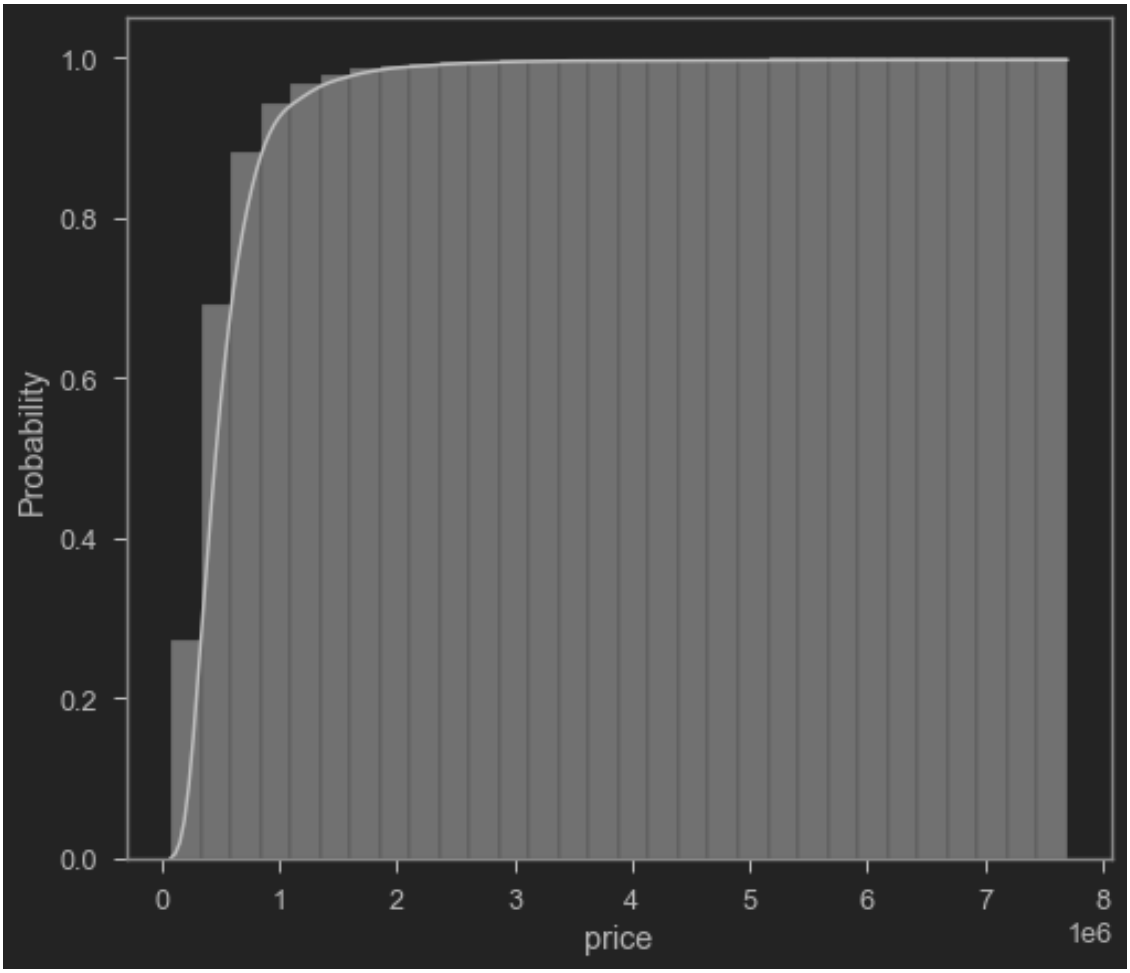
**Price**

[61]:
```
df_model['price'].describe().to_frame().style.format("{0:,.0f}")
```

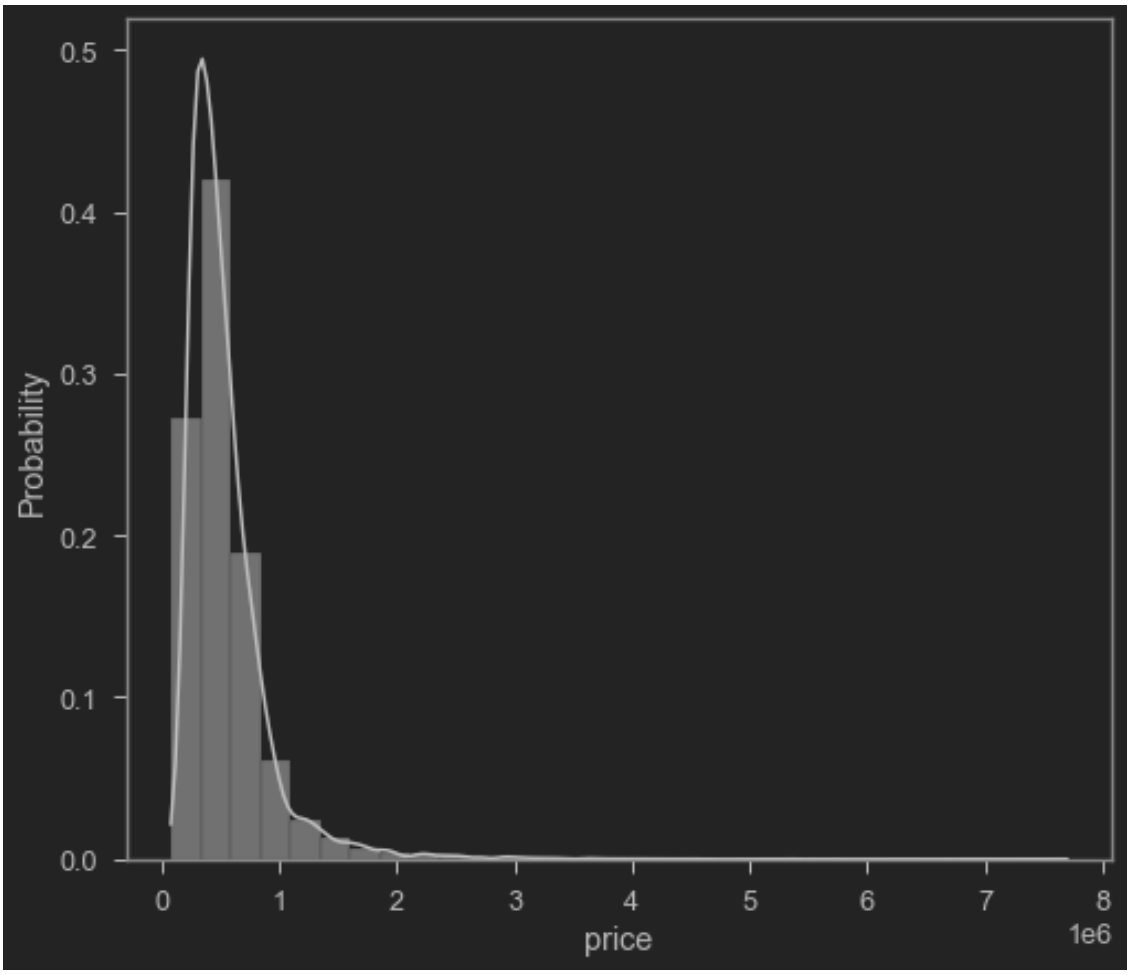
[61]:
```
<pandas.io.formats.style.Styler at 0x1c2574c2a00>
```

[62]:
```
sns.histplot(data=df_model.price,
             bins=30,
             color='silver',
             kde=True,
             stat="probability",
```

```
                cumulative=True);
```

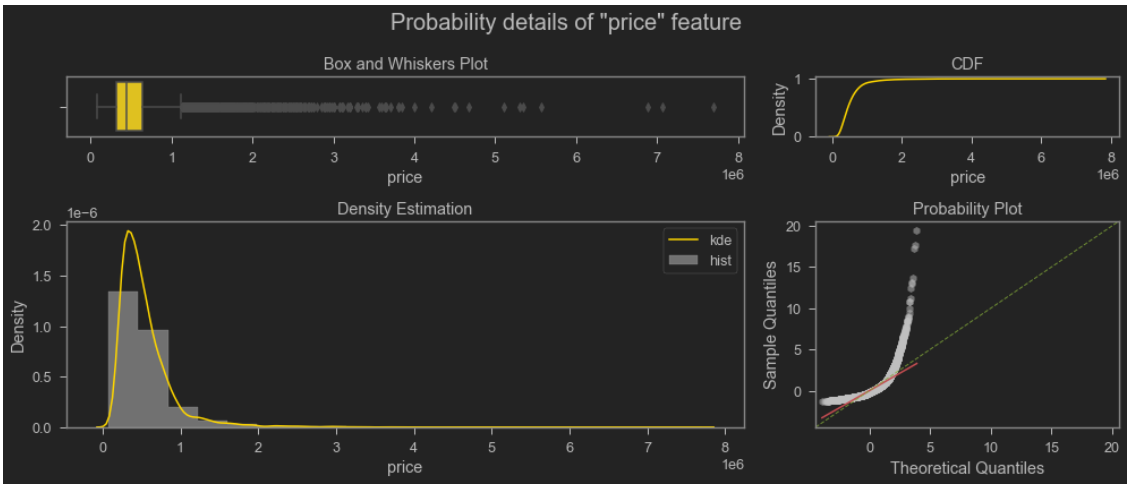



[63]:
```
sns.histplot(data=df_model.price,
             bins=30,
             color='silver',
             kde=True,
             stat="probability");
```




**probplot plot**

[64]:
```
# with supressing warning of N>5000
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    feature_prob_details(df_model.price)
```

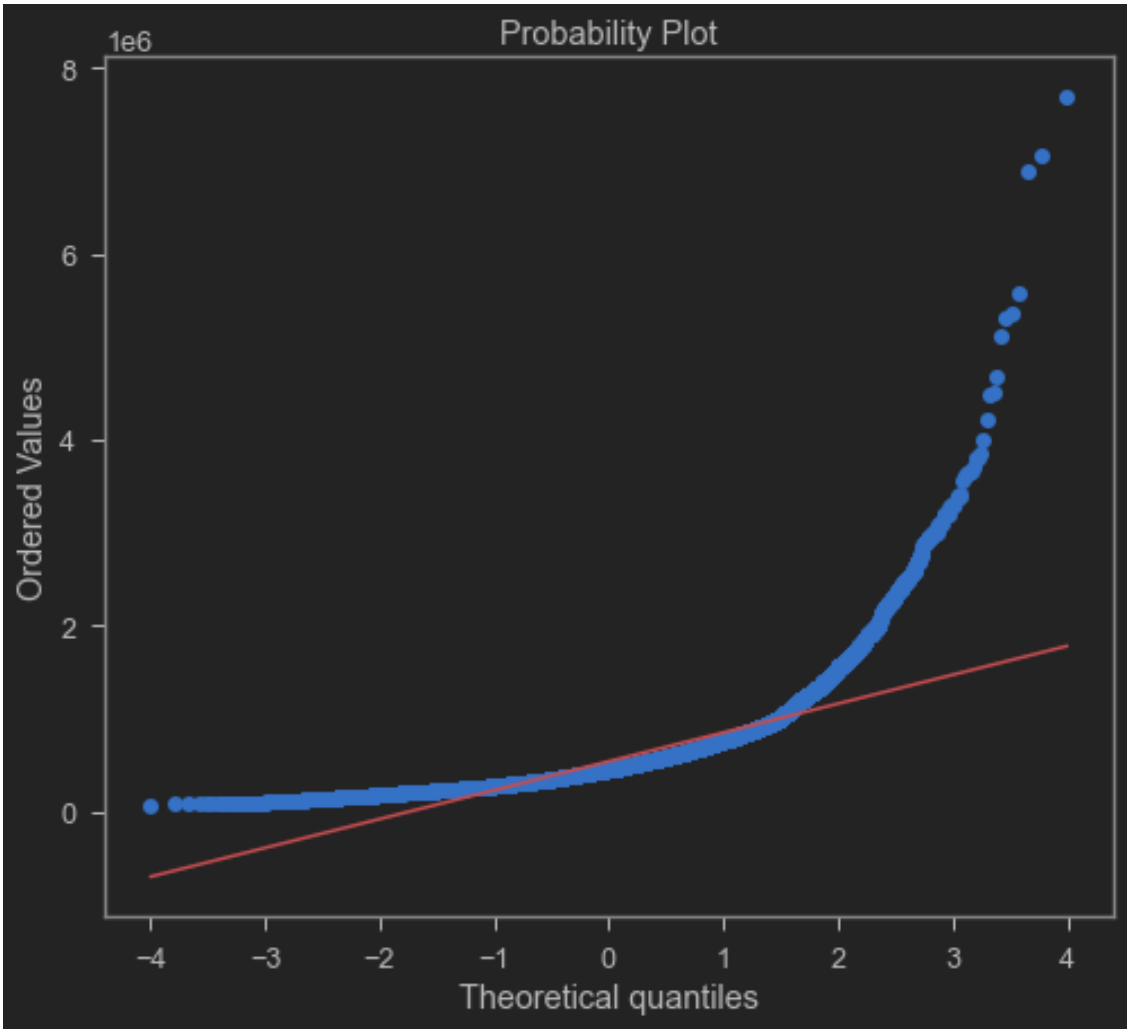



```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
```

This probability plot uses partial regression. The red line represents that. Green dotted line is the 45 degree line as a reference point. This is long tailed and highly skewed. This is expected for house price, lots of cheap house and some extravagant ones.

[65]:
```
fig, ax = plt.subplots()
ax = slope, intercept, r = stats.probplot(df_model['price'],
                                          dist="norm",
                                          fit=True,
                                          plot=plt)[1]
print(
    f'Slope: {round(slope,2)}, Intercept: {round(intercept,2)}, r_sq:␣
 ↪{round(r**2,4)}'
)
```

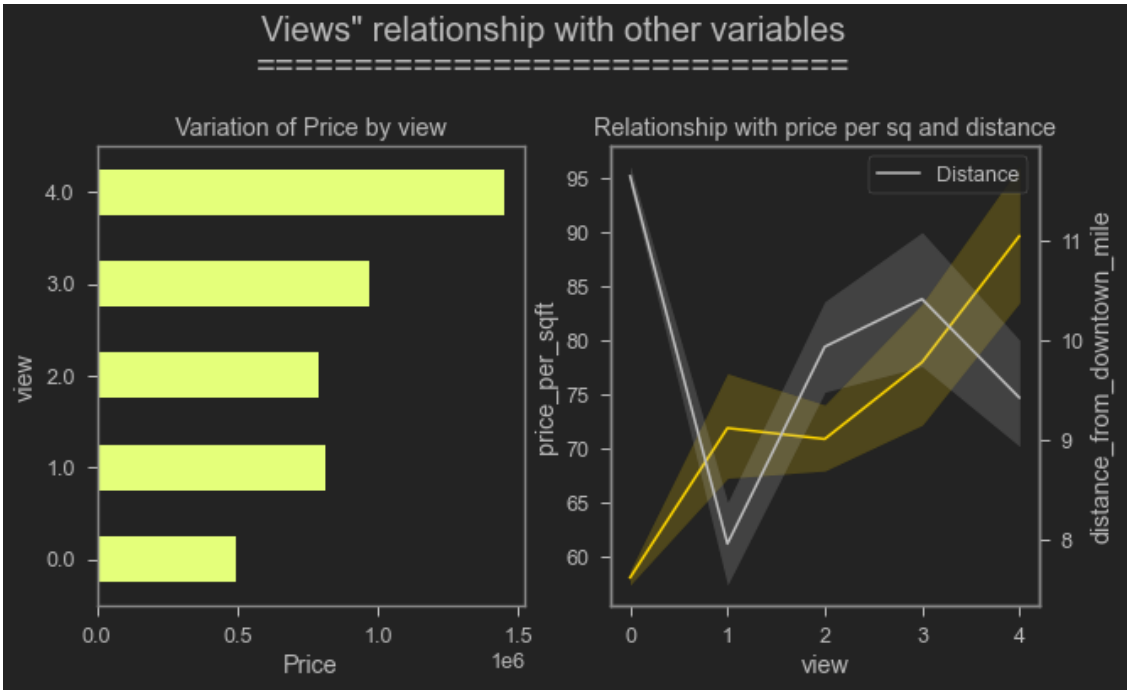
```
Slope: 310305.01, Intercept: 540734.67, r_sq: 0.711
```




**View**

[66]:
```
# without the axis lines.
fig, (ax, ax2) = plt.subplots(ncols=2, figsize=(10, 5))
ax = round(df.groupby(by='view').mean('price_per_sqft')['price'],
           2).plot(kind='barh',
                   colormap='Wistia',
                   title='Variation of Price by view',
                   ax=ax)
ax.set_xlabel('Price')
ax = sns.lineplot(y='price_per_sqft', x='view', data=df, color='gold')
ax2 = ax.twinx()
ax2 = sns.lineplot(y='distance_from_downtown_mile',
                   x='view',
                   data=df,
                   color='silver')
```

```
ax2.set_title('Relationship with price per sq and distance')
ax2.set_label("Distance")
ax2.legend(['Distance'])
plt.suptitle(f'Views" relationship with other variables\n{"="*30}',
             fontsize=20,
             va='bottom')
plt.show()
```




Now trying to make sense of `"view"`. It seems like it detects the aesthetic quality of the house. Not sure if it is something a owner can manipulate, e.g. architectural design or landscaping and decoration; or its is natural, e.g. close to a park.

Higher rated house has the greatest average price. This trend is very clear.

It seems that houses closer to the downtown has a higher price per sqft but lower view as the city landscape is not pristine but proximity to business center makes them desirable. But at the next category price drops as distance increases, people are not shelling coins for sub-par property. When the distance is high means we are out of downtown with less congested area where view is better, thus increase in price. Then at the last one we are at the suburbs, where price sees a drop because of distance.

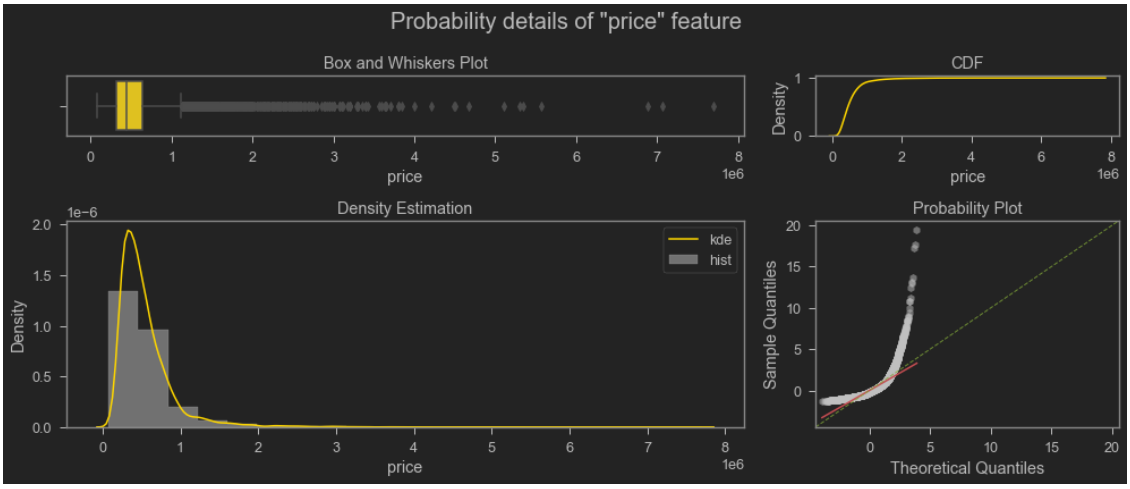
The boxed area is the highest rating lowest price.

This feature is still sufficiently ambiguous to present to the public, I might be interpreting this from a myopic view and got entirely wrong.

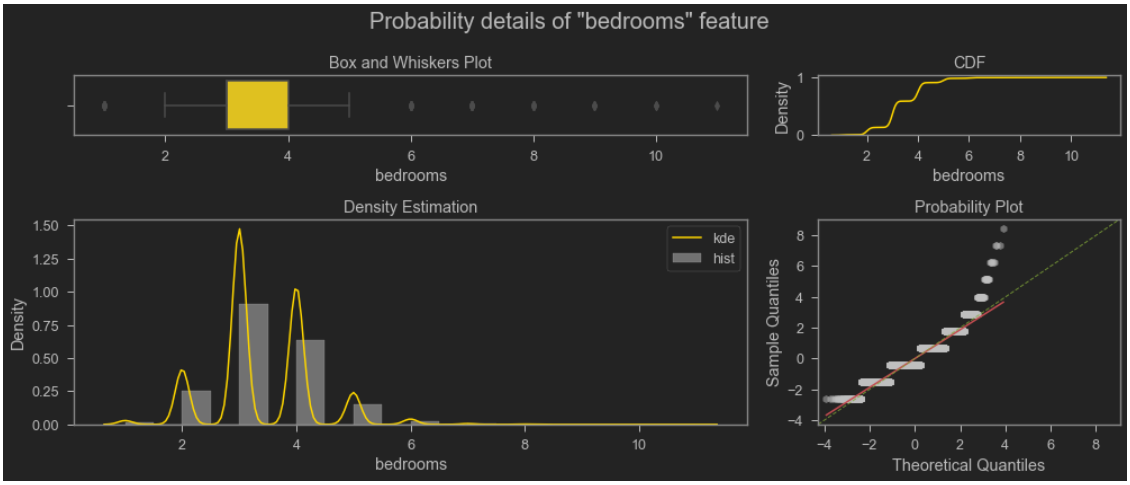
### 4.6.2 check the distribution of the features

[67]:
```
## taking a deeper look at all the features.
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    for column in df_model:
        print(f'{"="*180}')
        feature_prob_details(df_model[column])
        print(f'{"="*180}')
        plt.show()
```

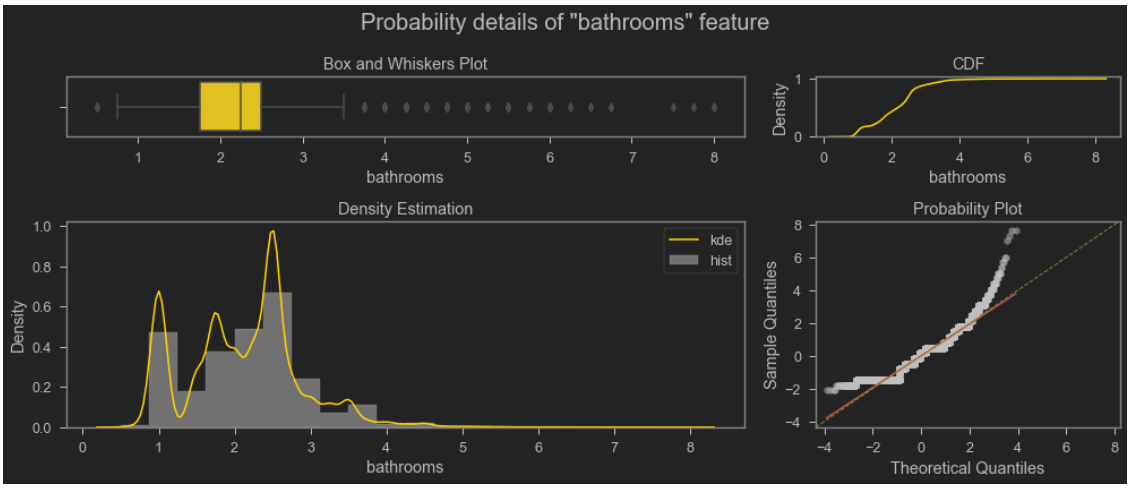
```
================================================================================
================================================================================
====================
```




```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```

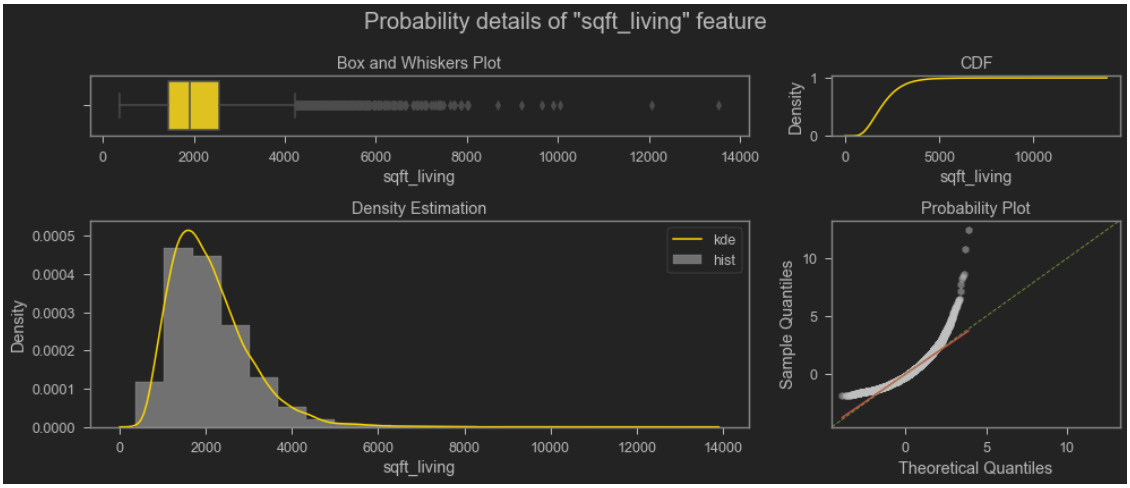



Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.




Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

==========

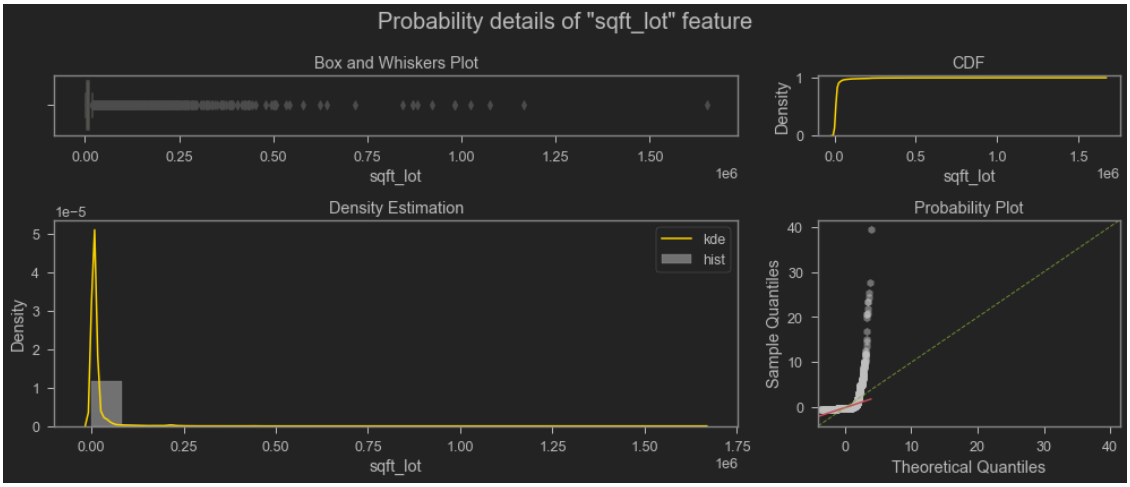



Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

==========

==========

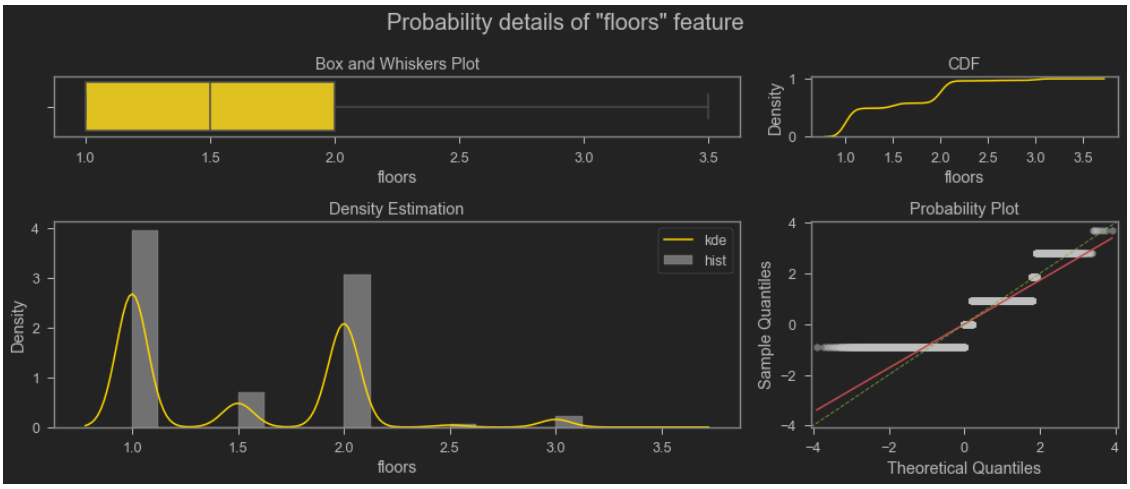
==========




Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

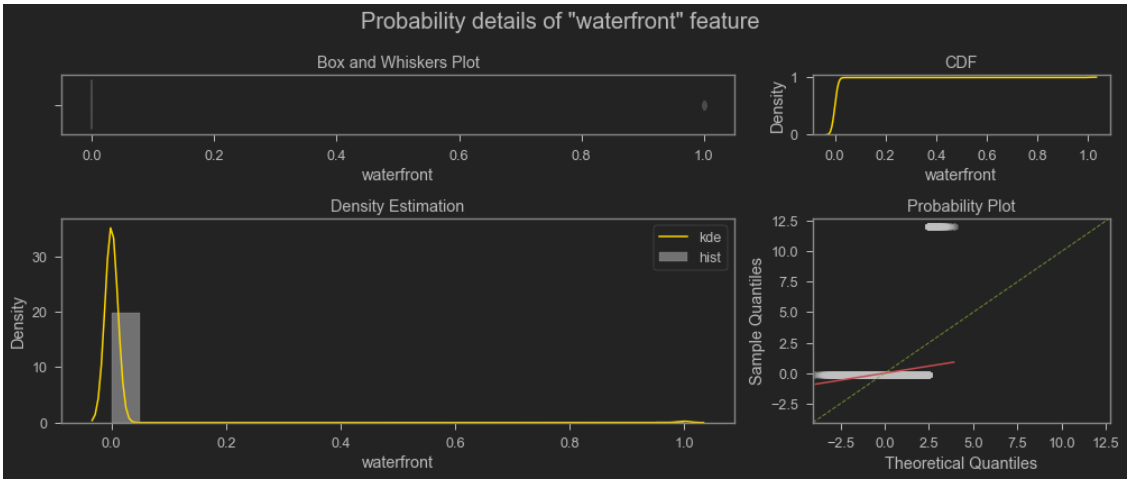
==========

==========
==========
==========
==========

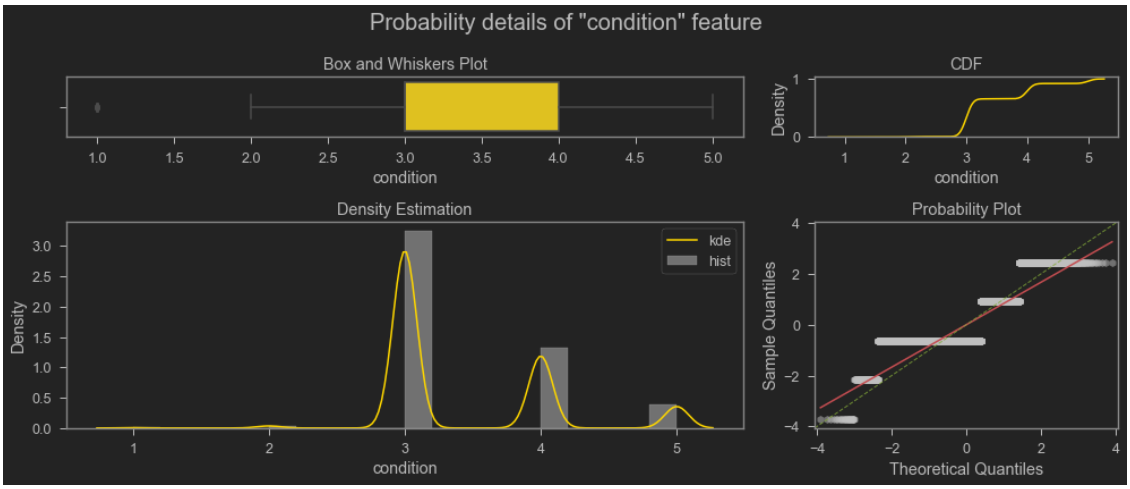



Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

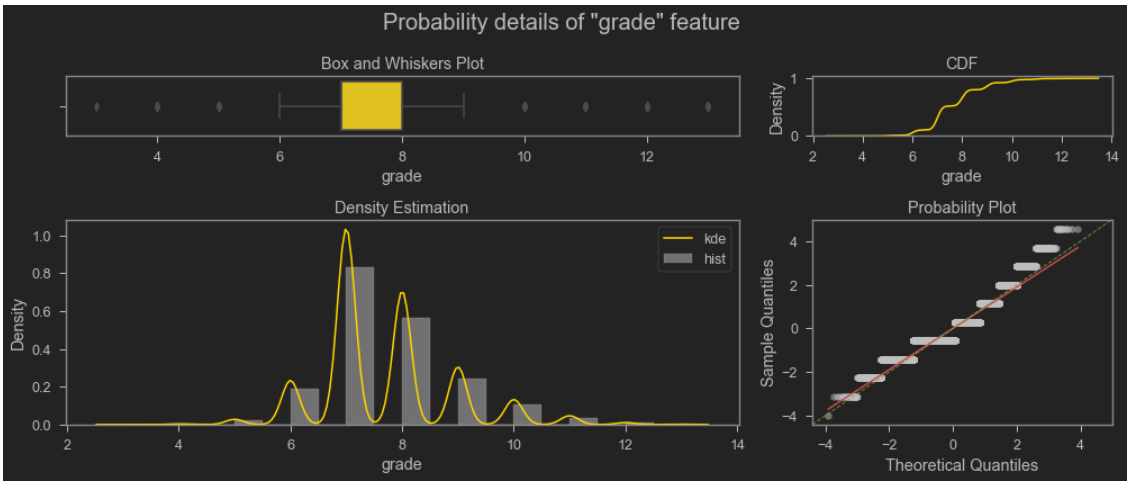
==========
==========
==========
==========



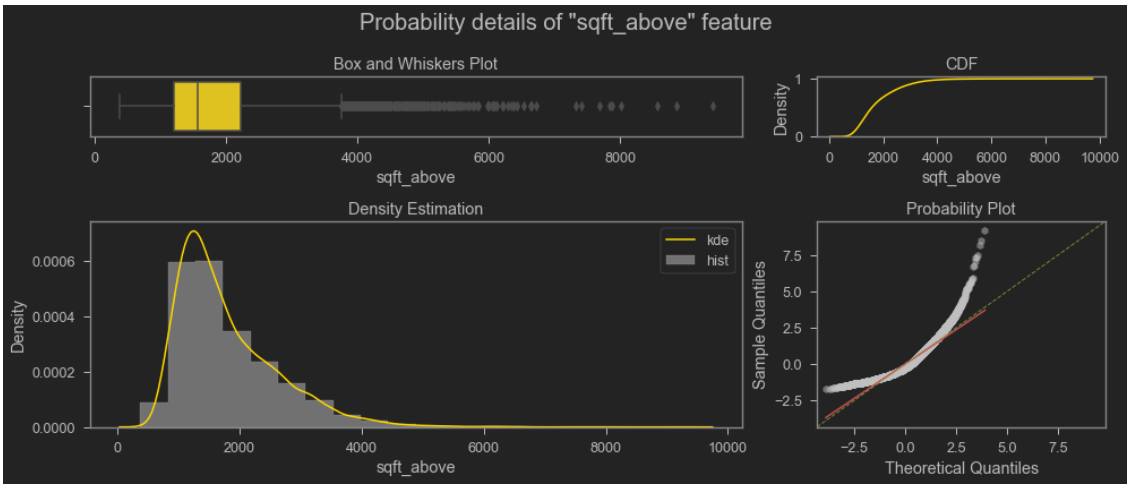
```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```



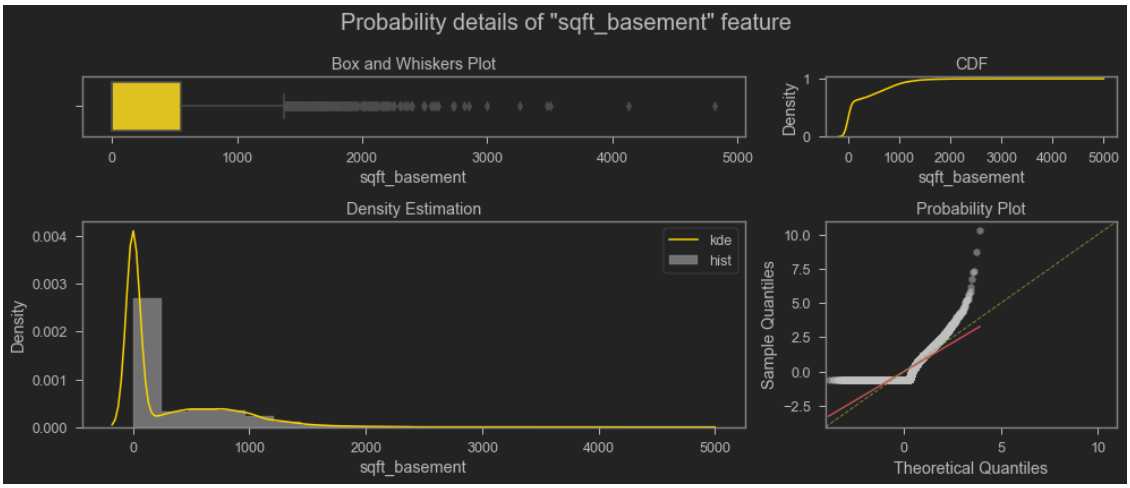

```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```



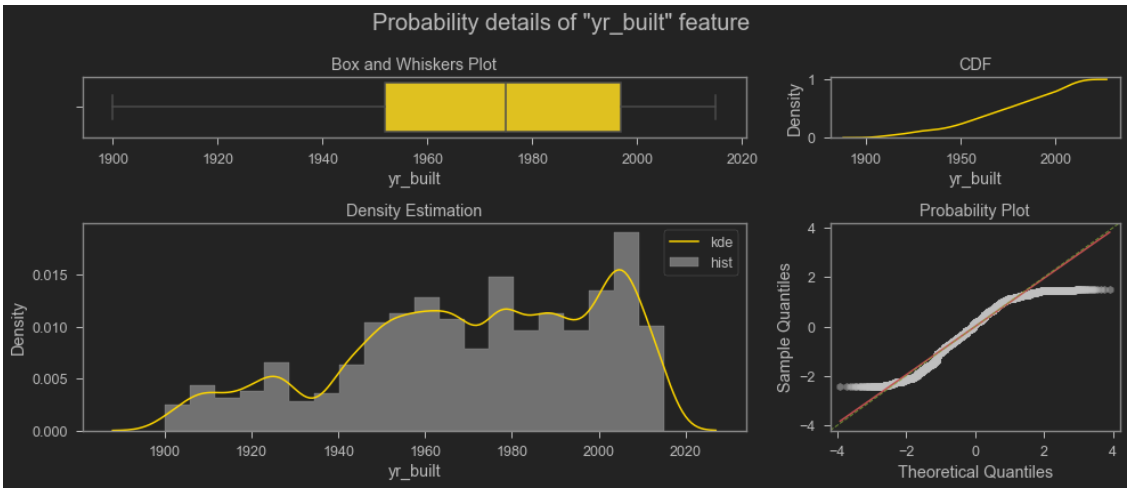
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.




Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

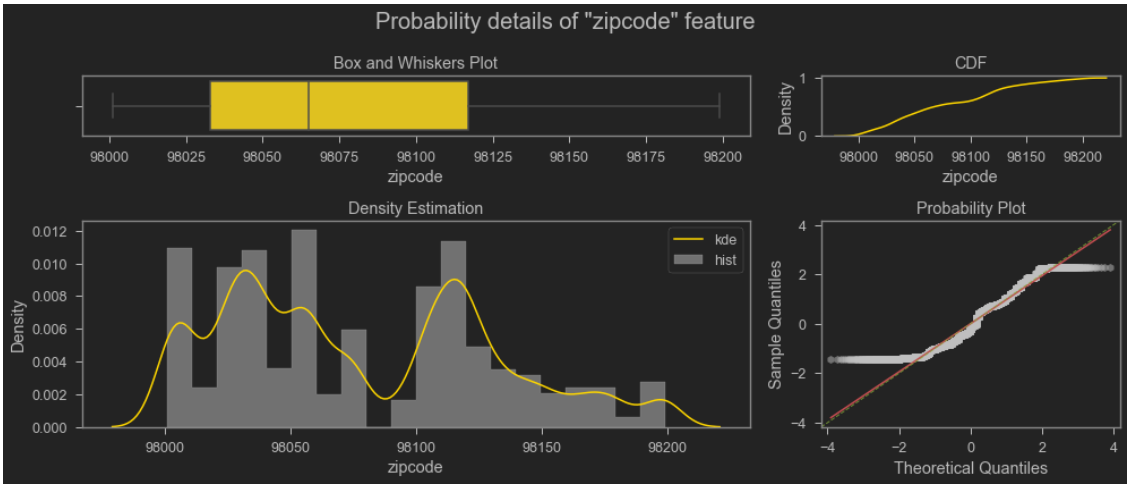



Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.




Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

==========

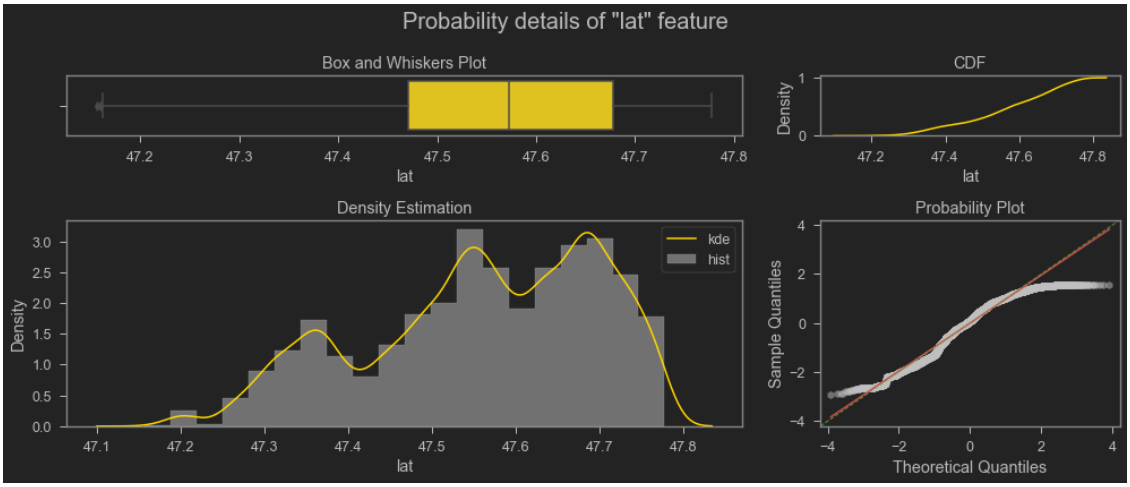



Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

==========

==========

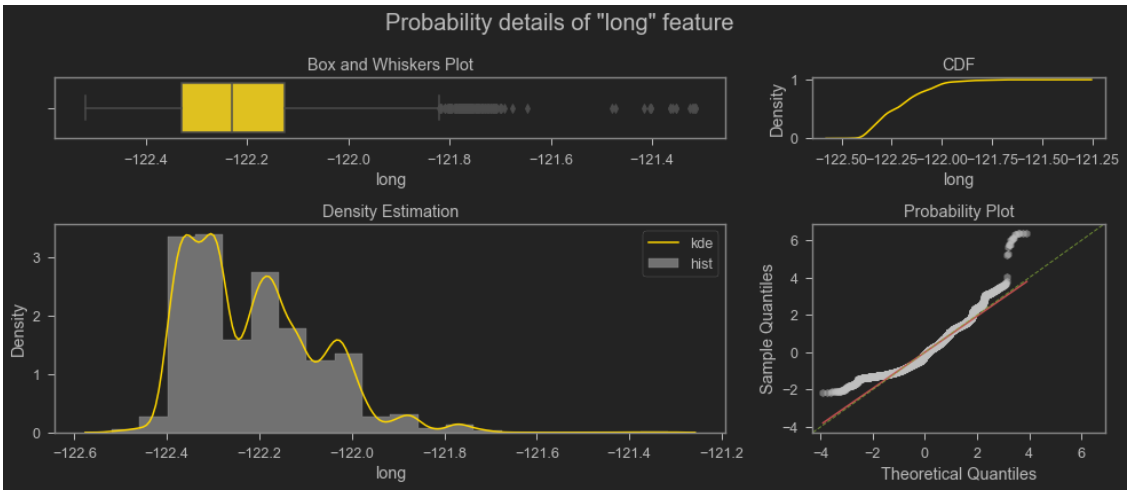
==========



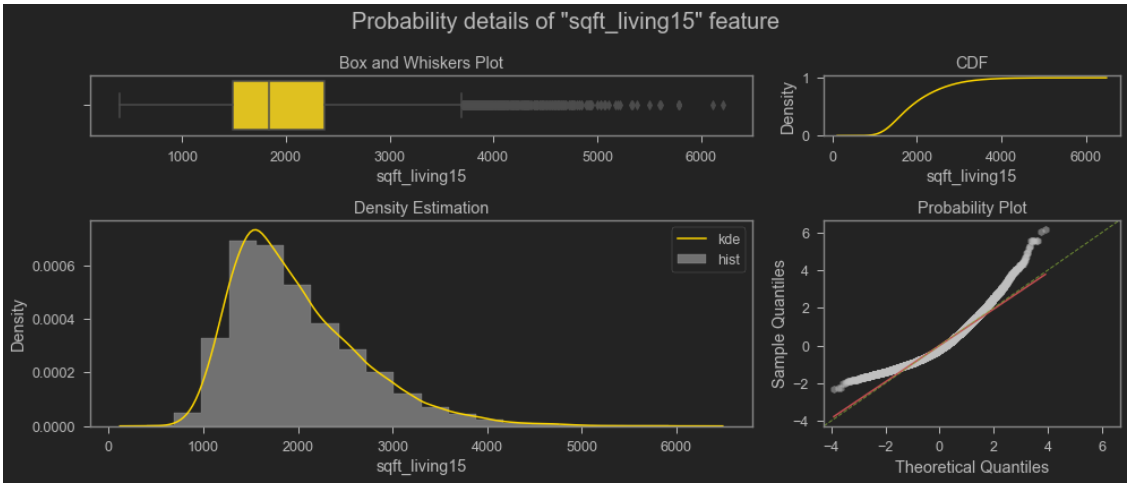

Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

==========

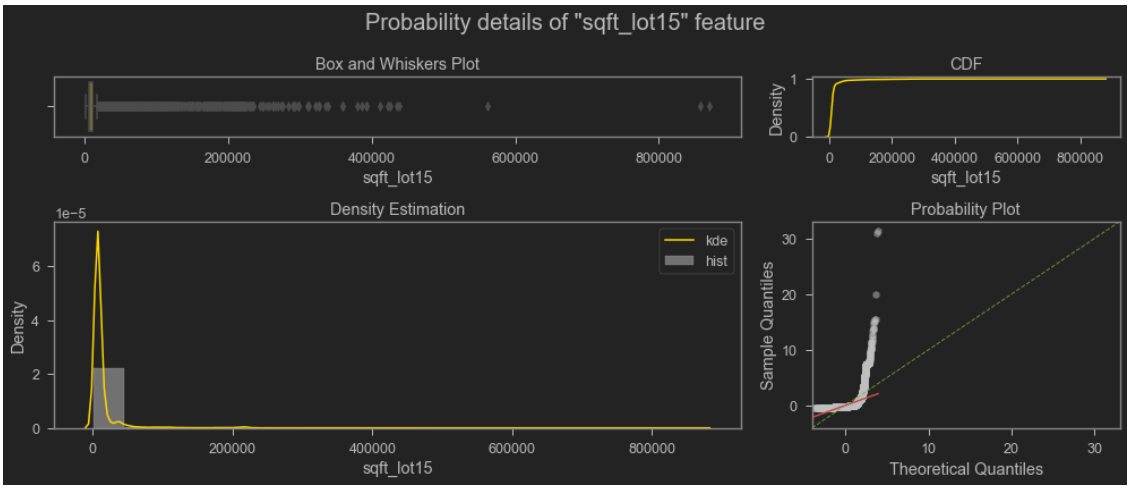
```
================================================================================
====================
================================================================================
================================================================================
====================
```



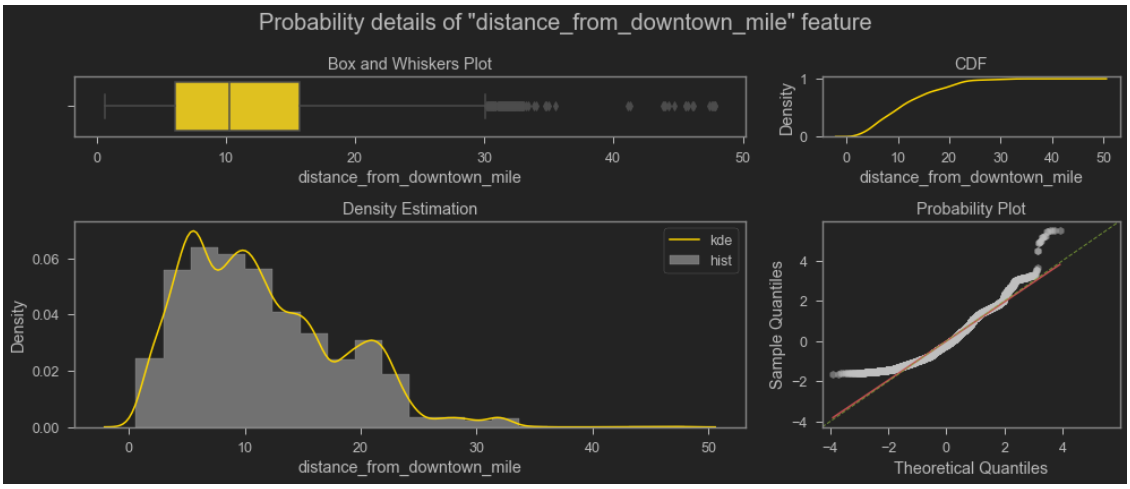

```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```



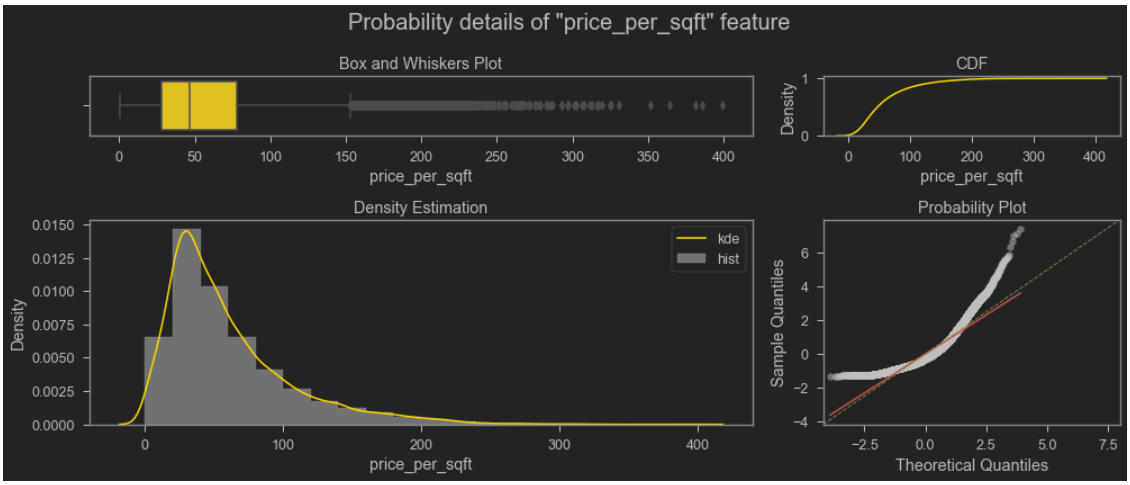
```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```



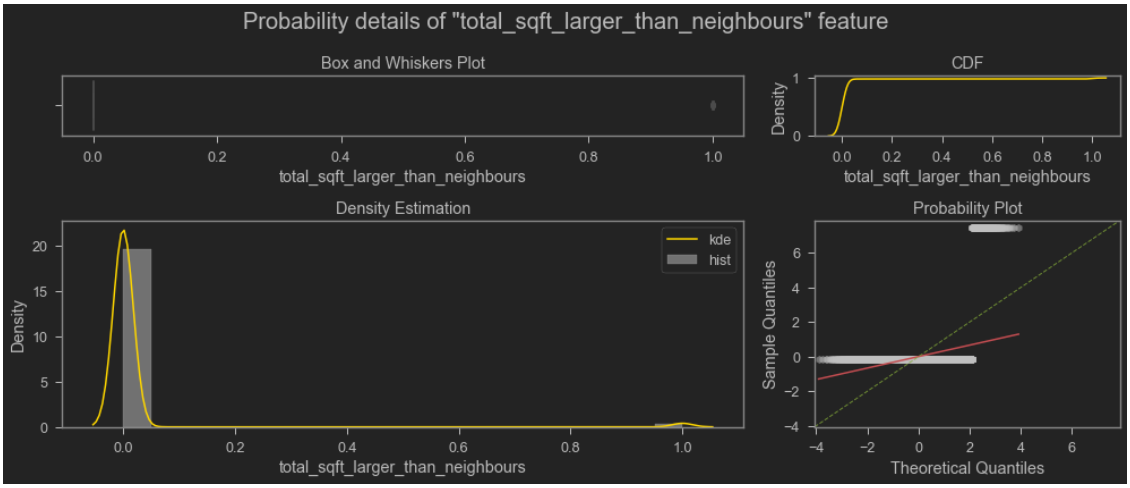

```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```



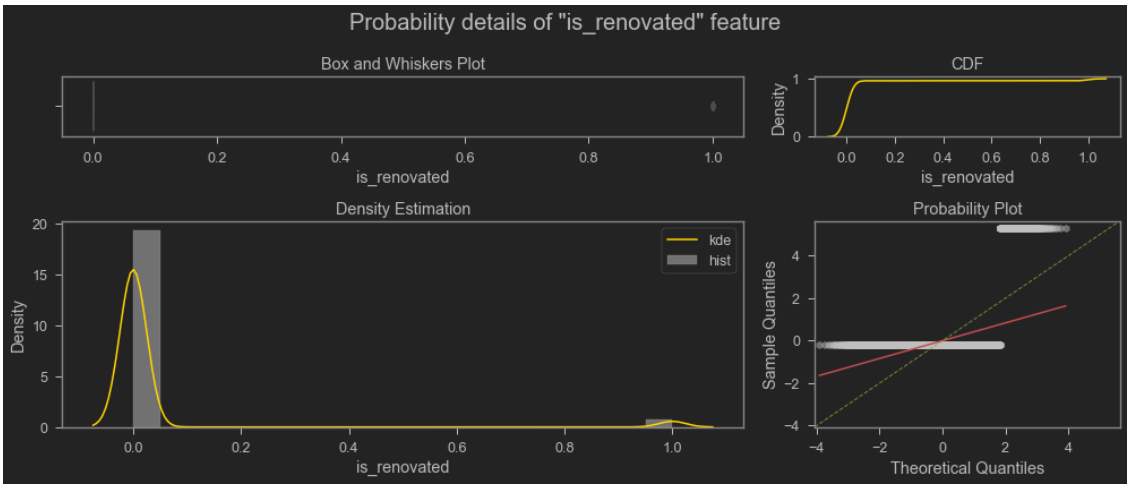
```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```




```
Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov
test for goodness of fit at 95.0% confidence interval.
================================================================================
================================================================================
====================
================================================================================
================================================================================
====================
```




Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.



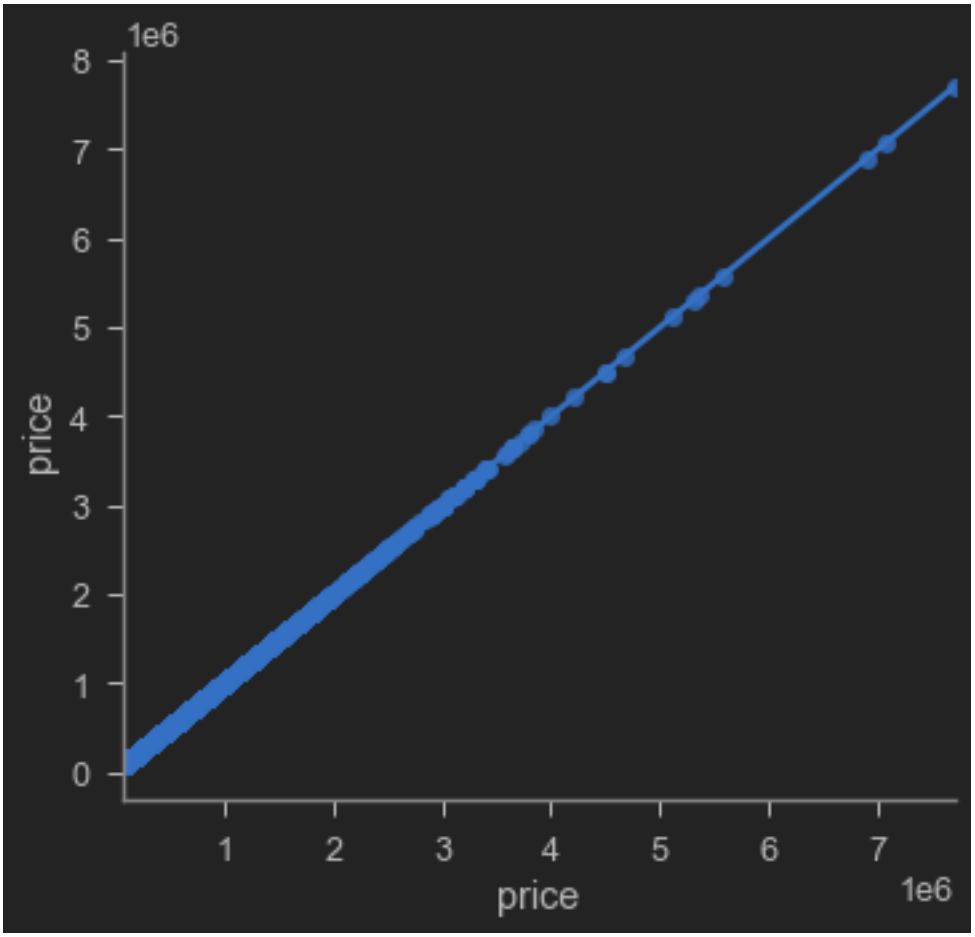

Distribution is NOT NORMAL based on Shapiro-Wilk Test and Kolmogorov-Smirnov test for goodness of fit at 95.0% confidence interval.

NONE of them are normal. Some with large outliers. This issue will be tackled in later steps.

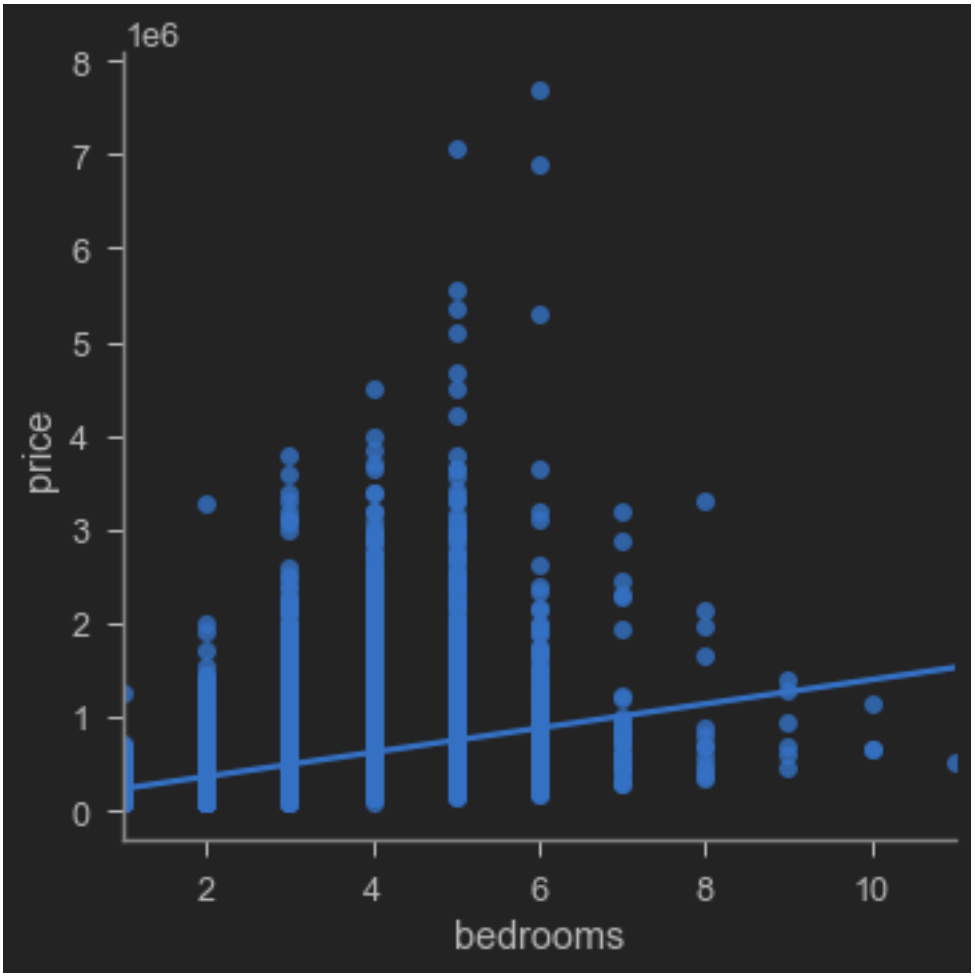
### 4.6.3 check the linearity of the features

[68]:
```
for column in df_model:
    print(column)
    sns.lmplot(y='price',x=column,data=df_model,ci=68).set_titles('ok')
    plt.show()
```

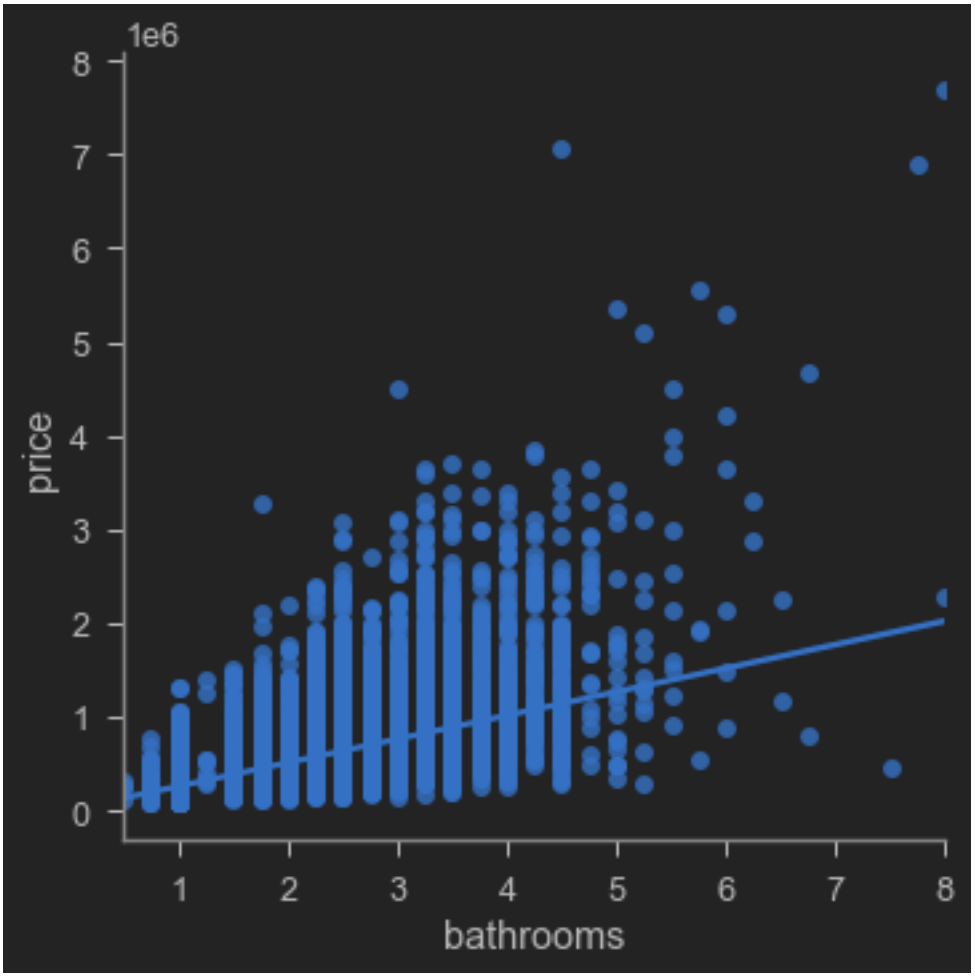
```
price
```



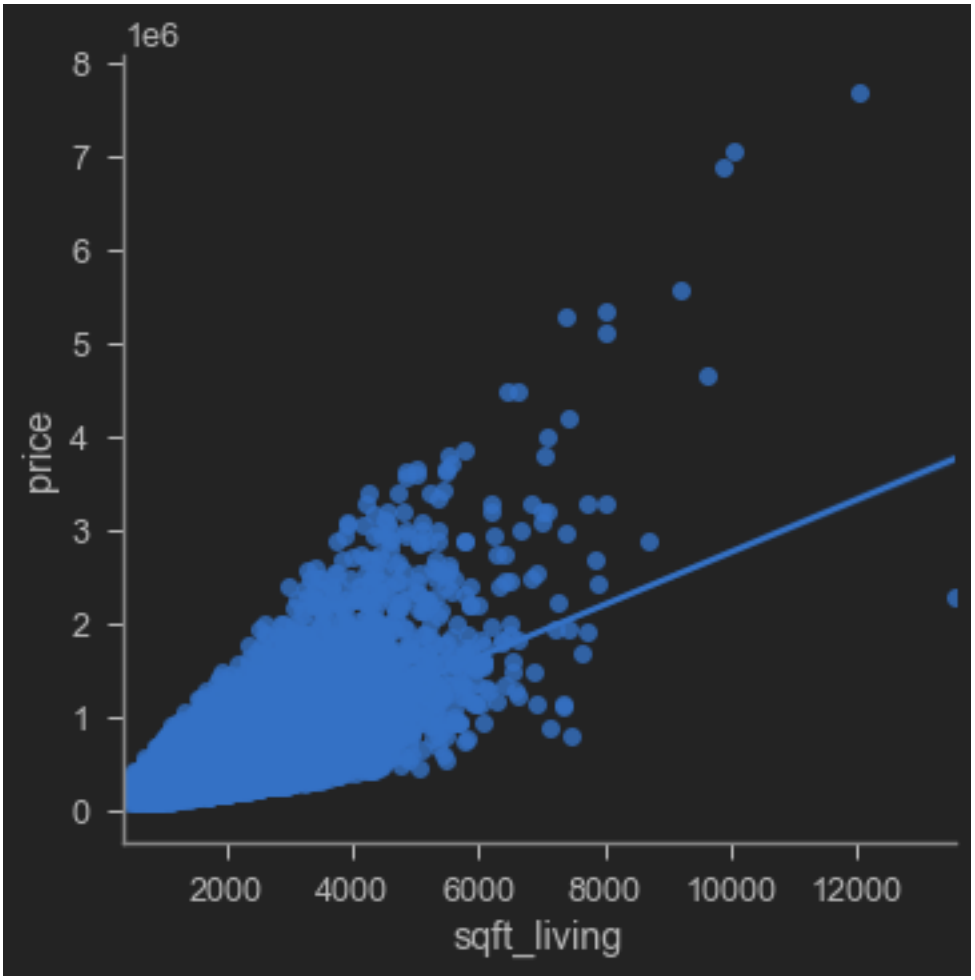

```
bedrooms
```



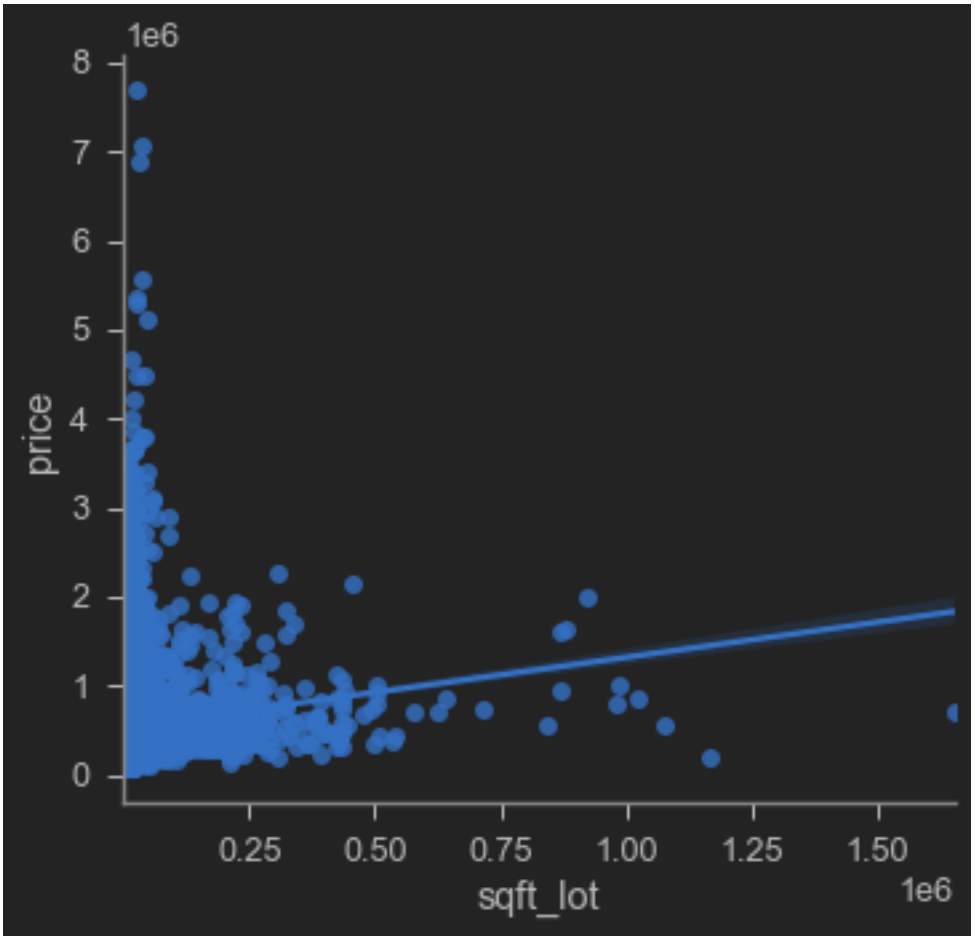

`bathrooms`



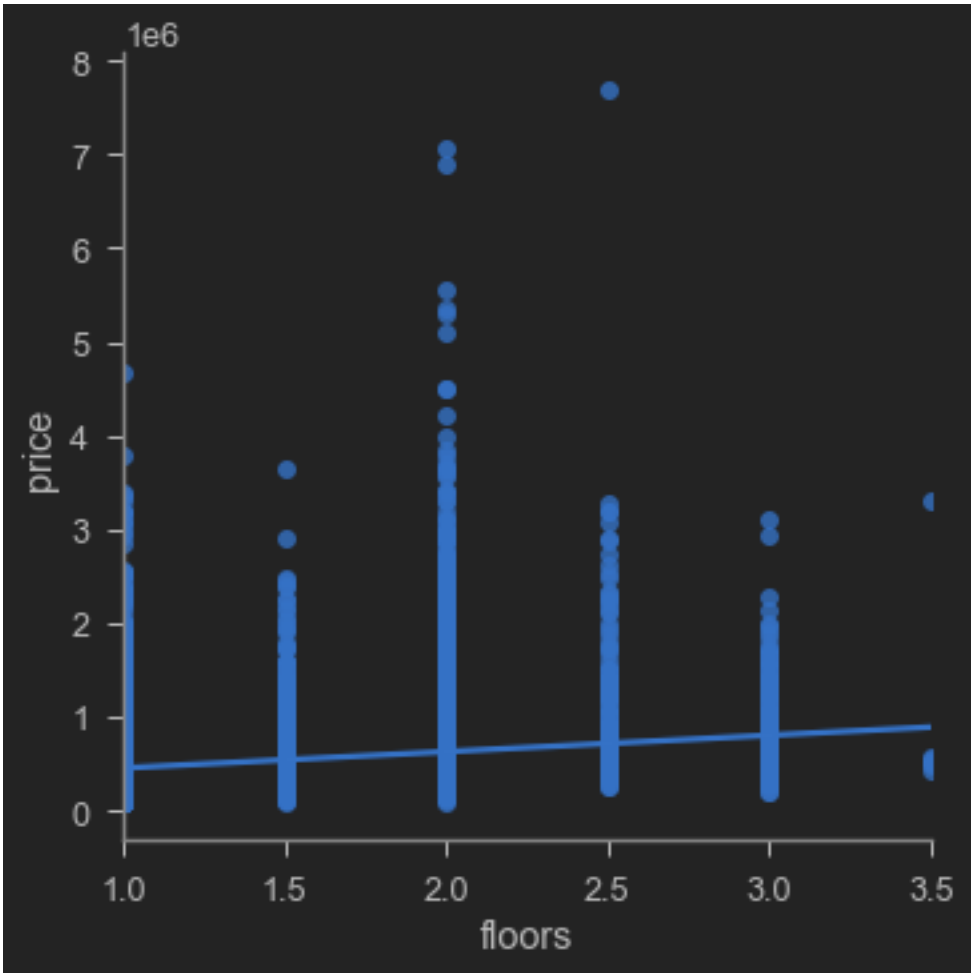

`sqft_living`


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

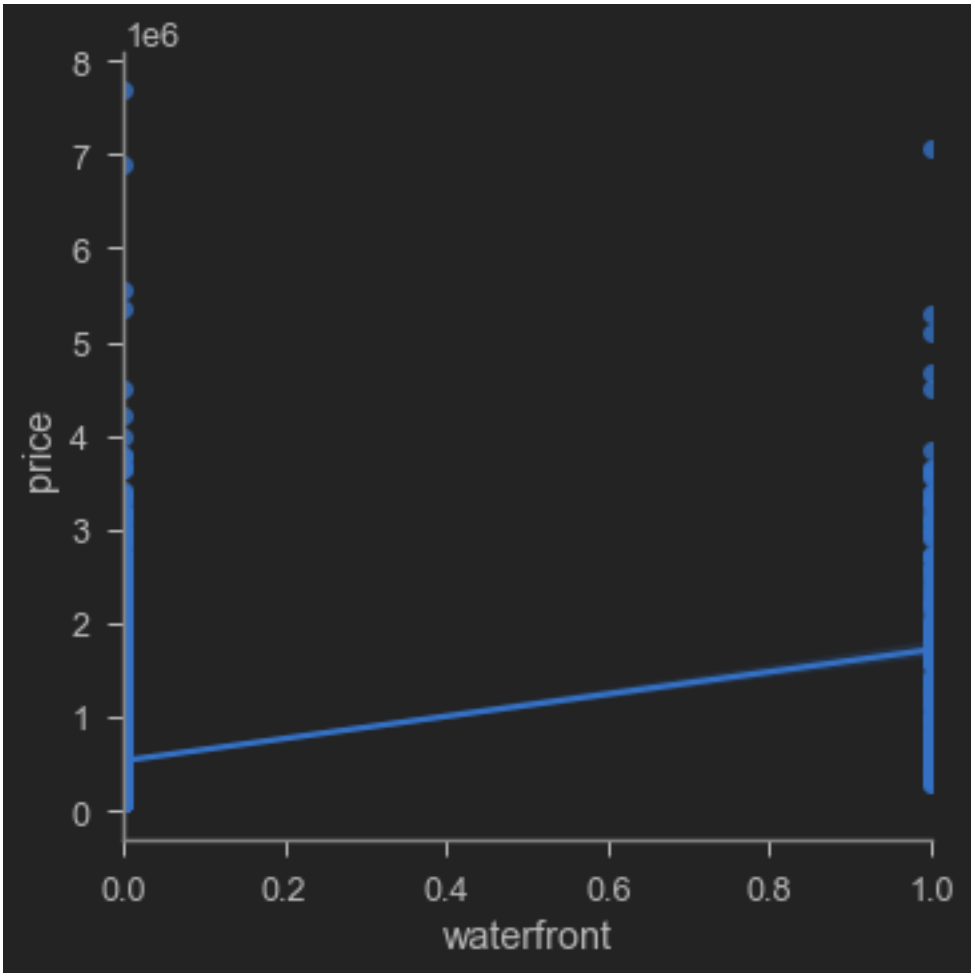
`sqft_lot`



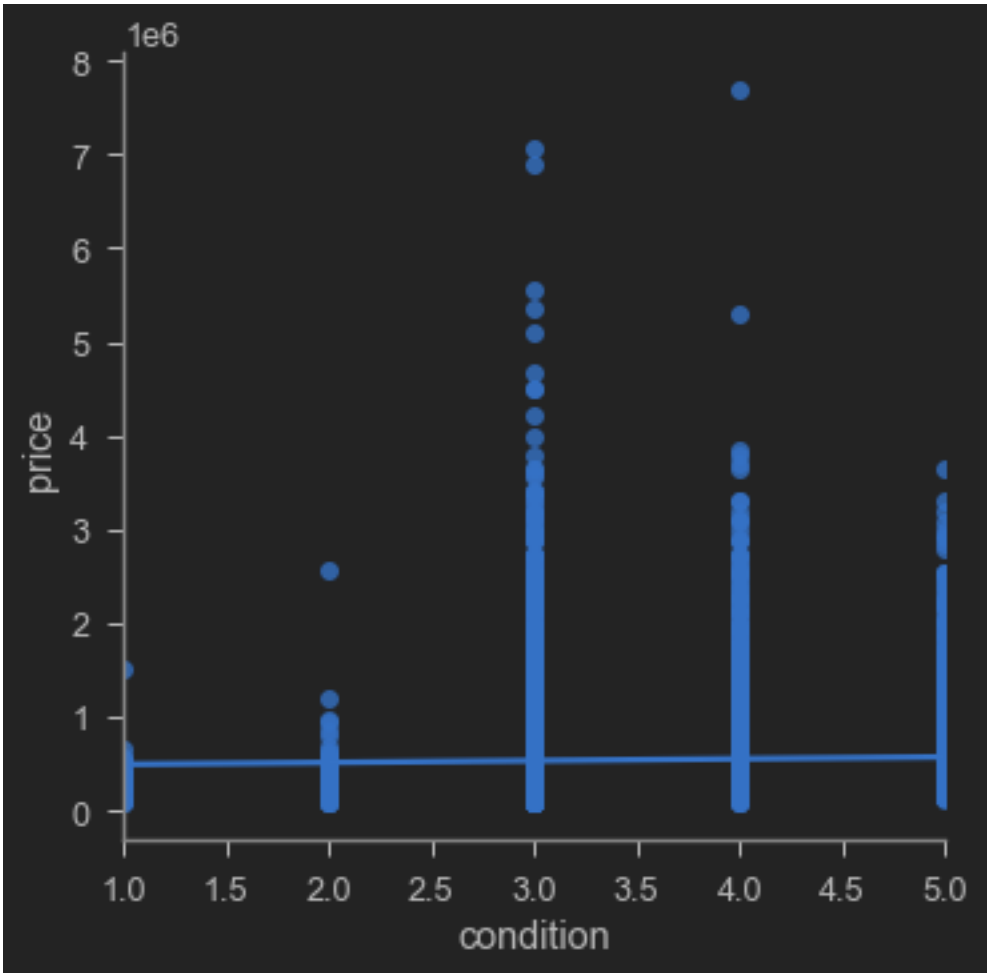

`floors`



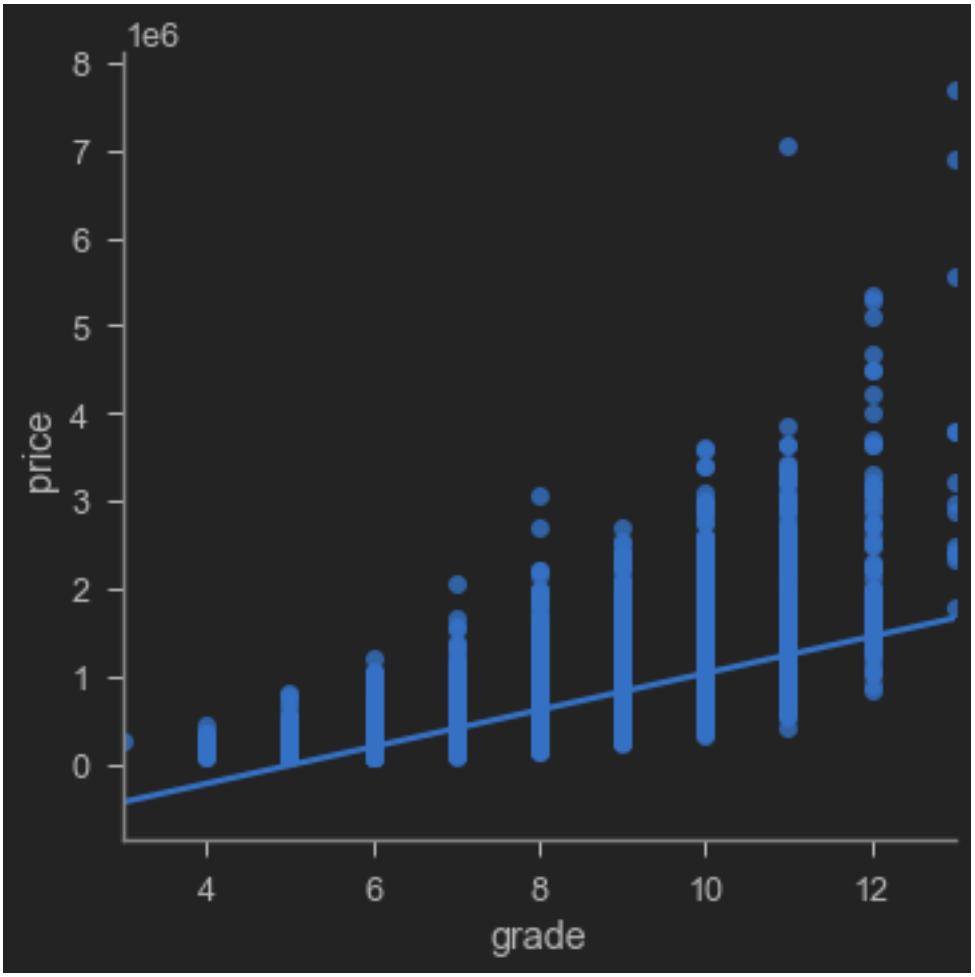

`waterfront`



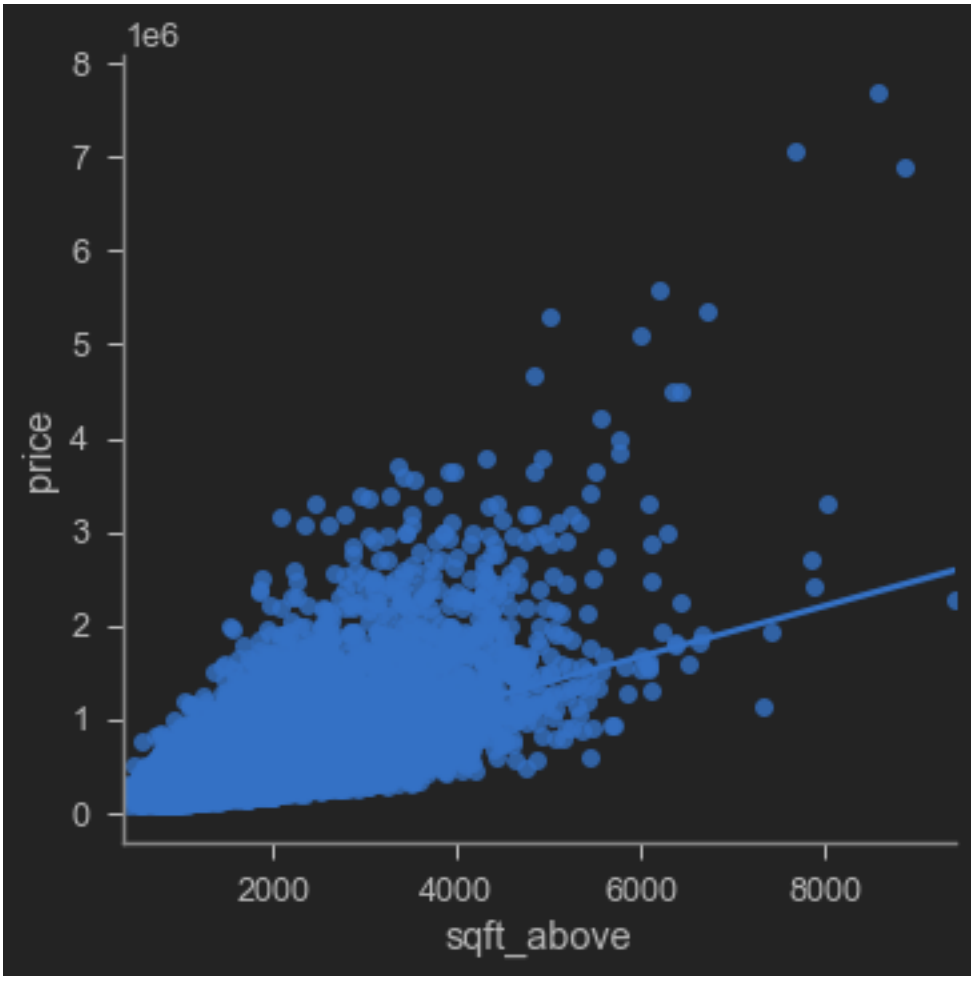

`condition`



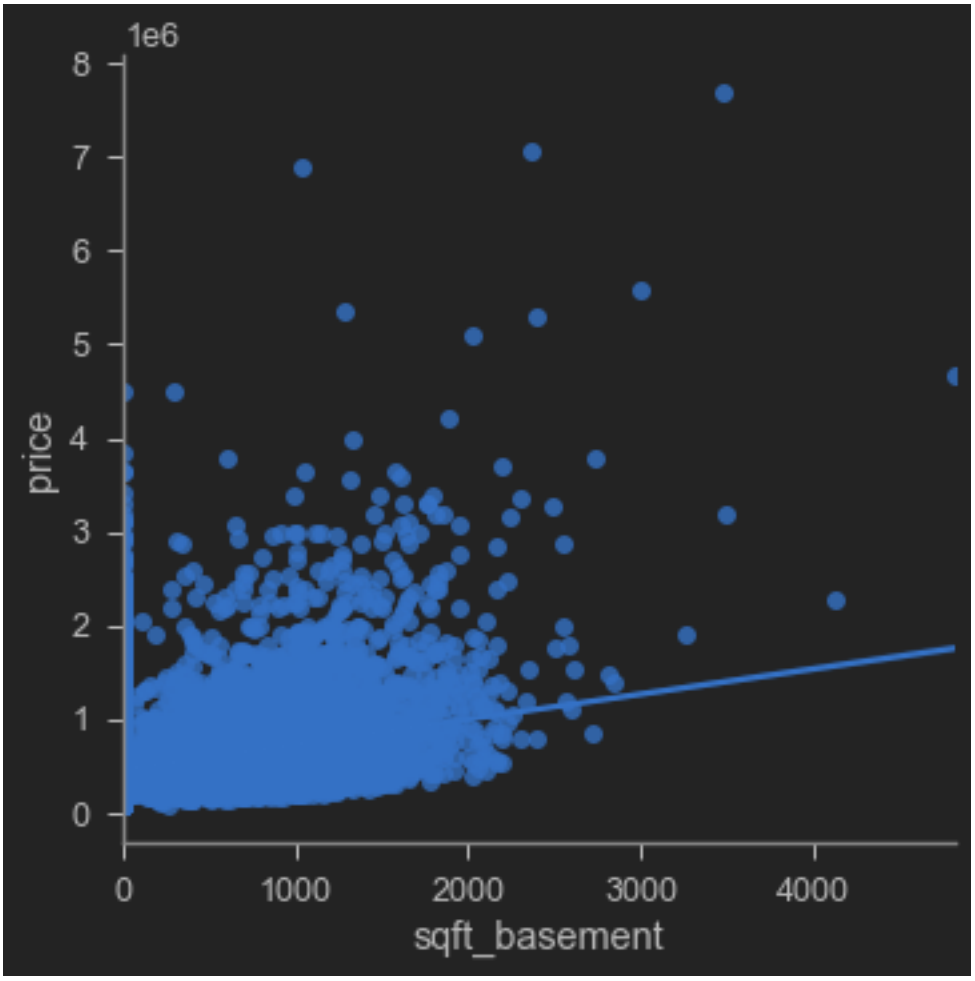

`grade`



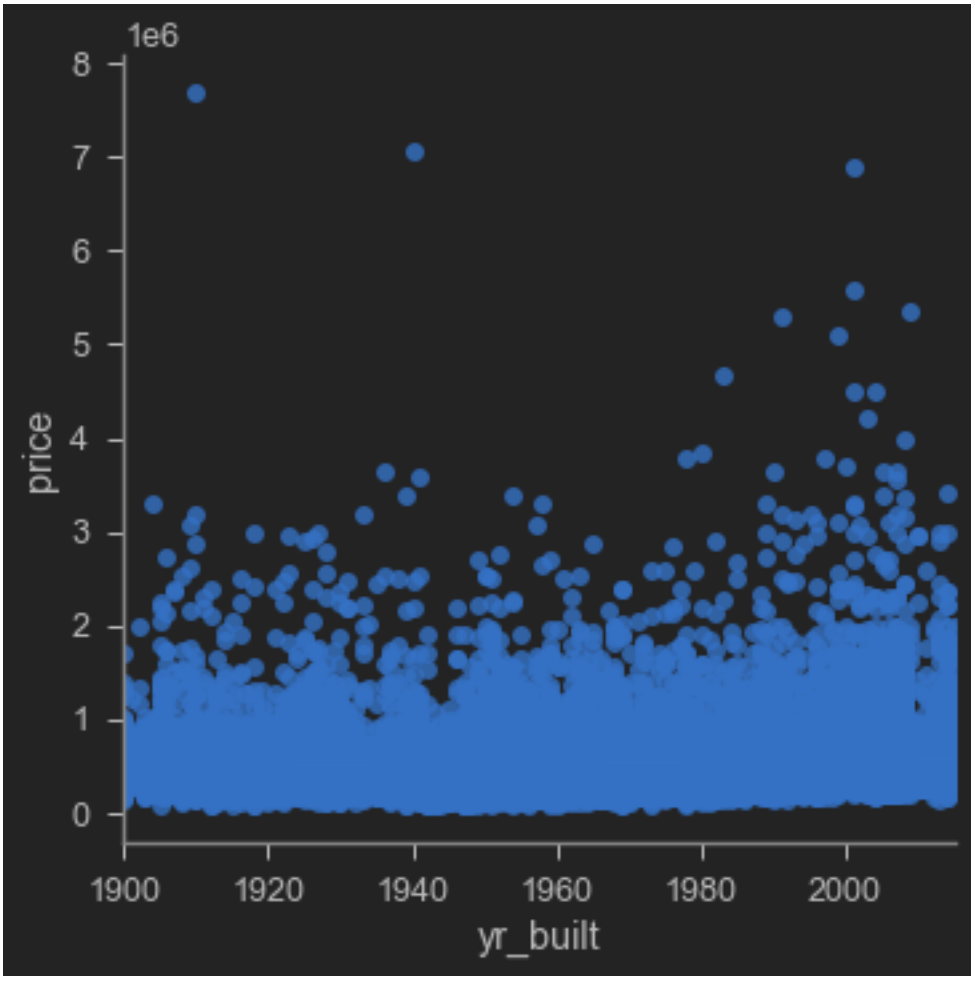

`sqft_above`



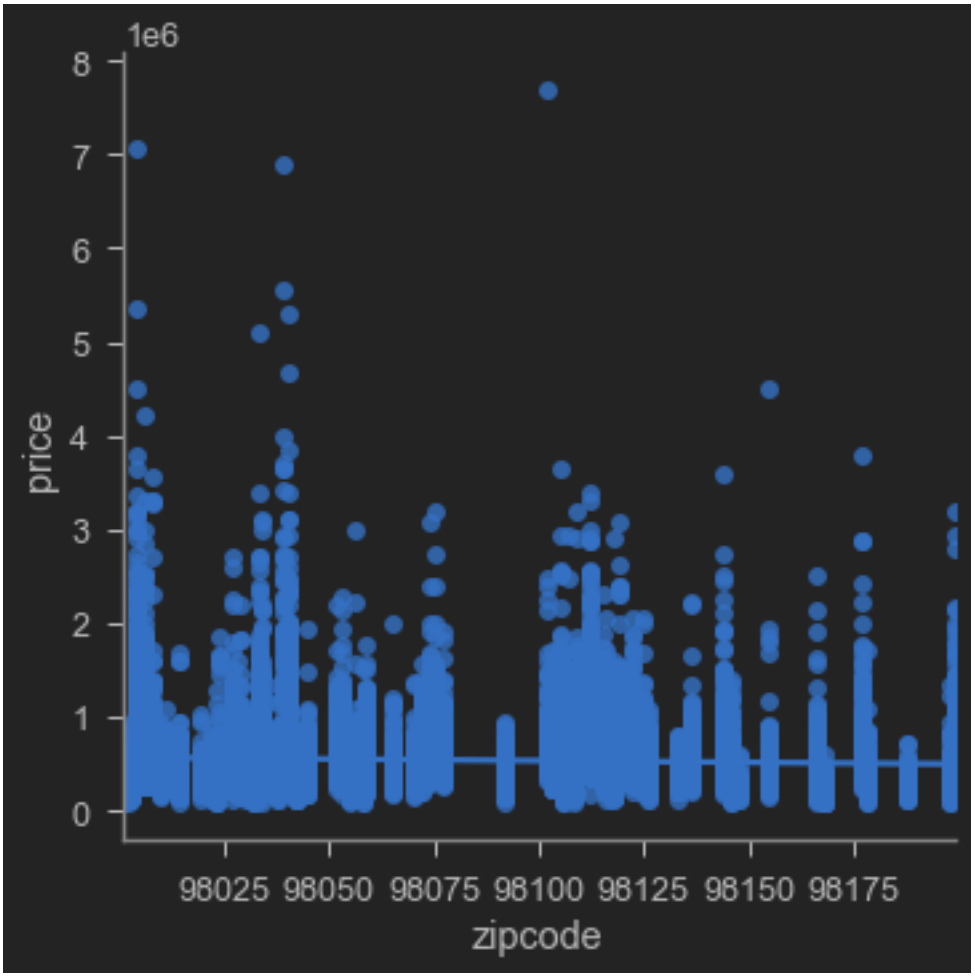

`sqft_basement`



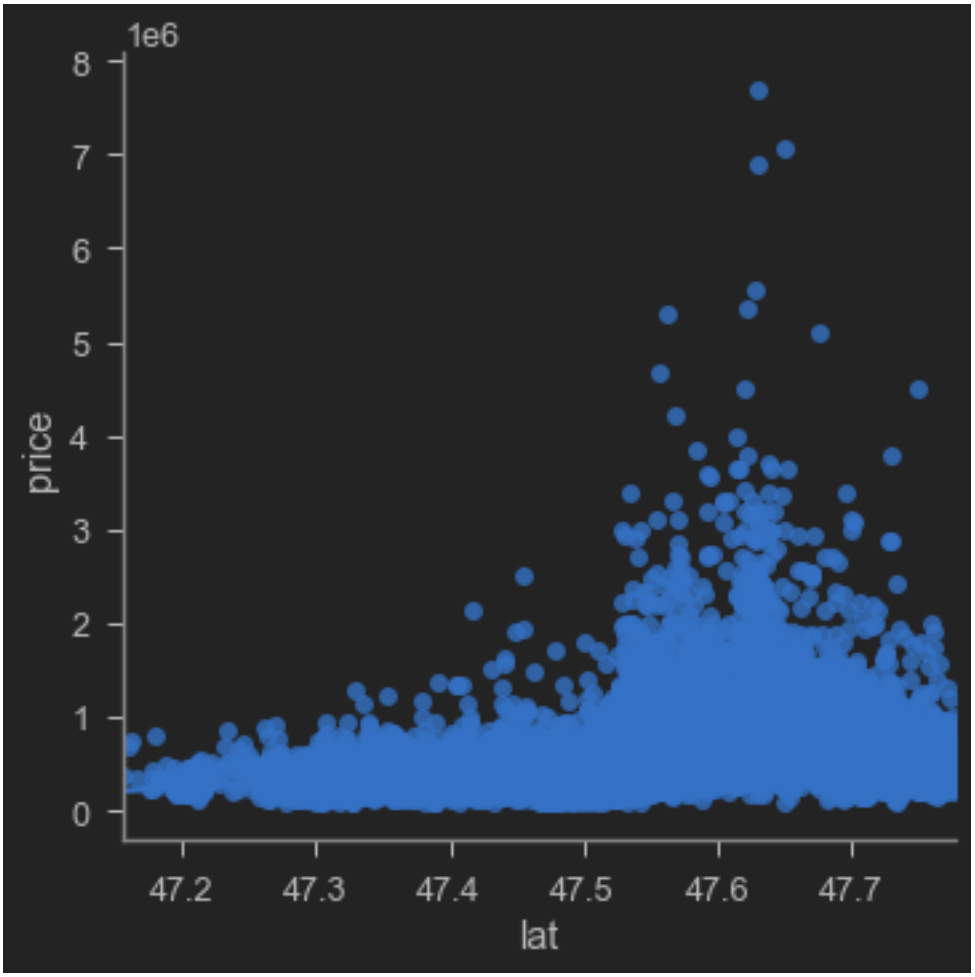

`yr_built`



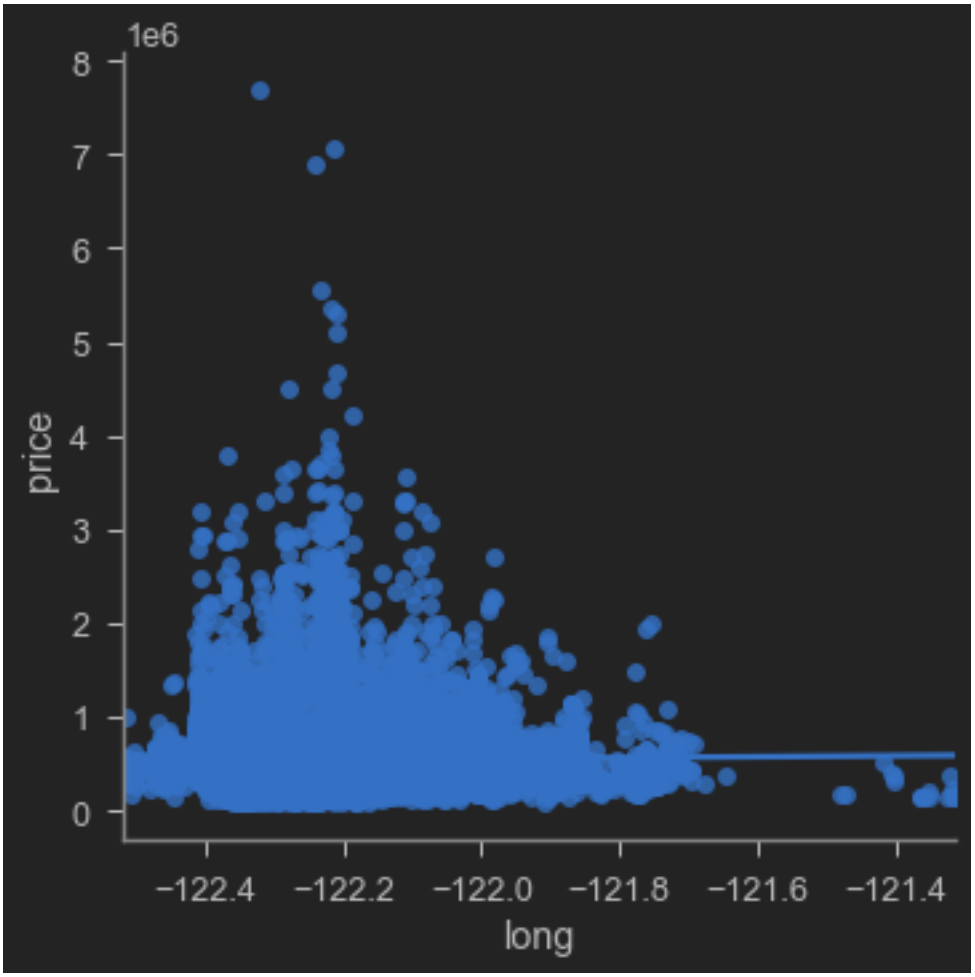

`zipcode`


1e6
8
7
6
5
4
3
2
1
0
price
98025 98050 98075 98100 98125 98150 98175
zipcode

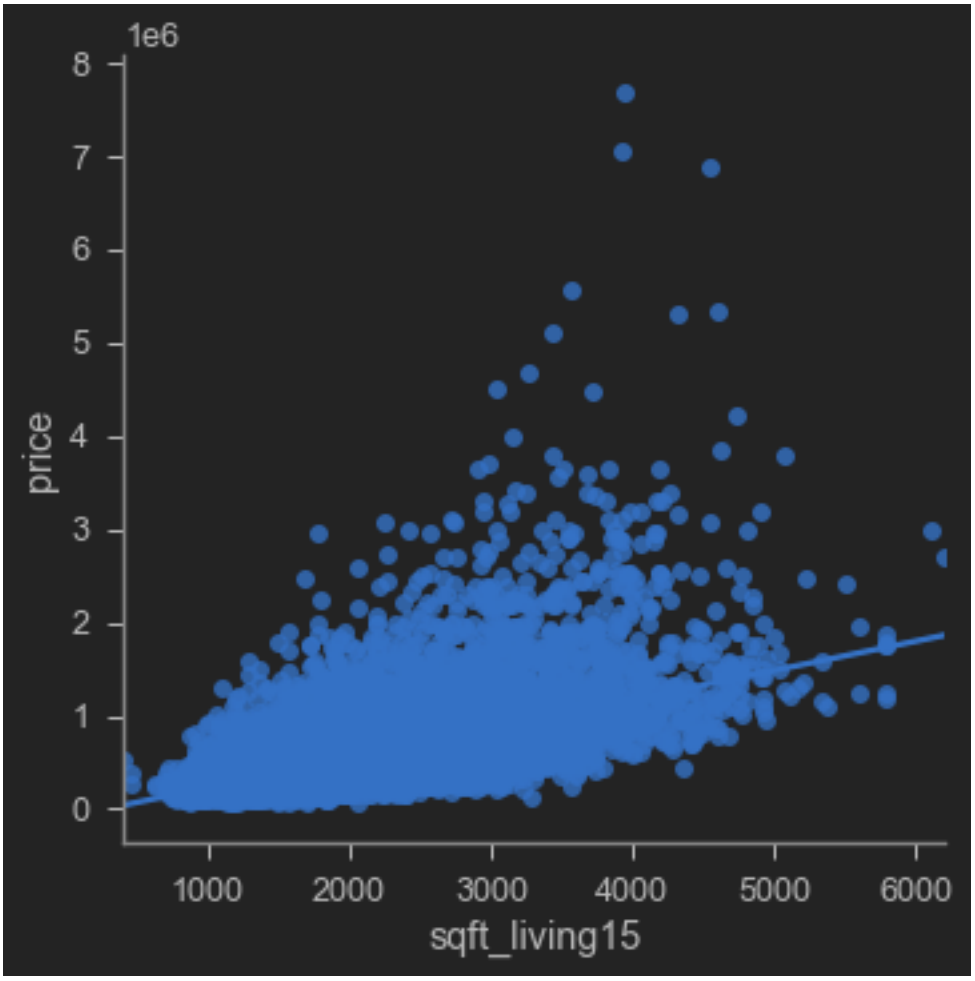
`lat`



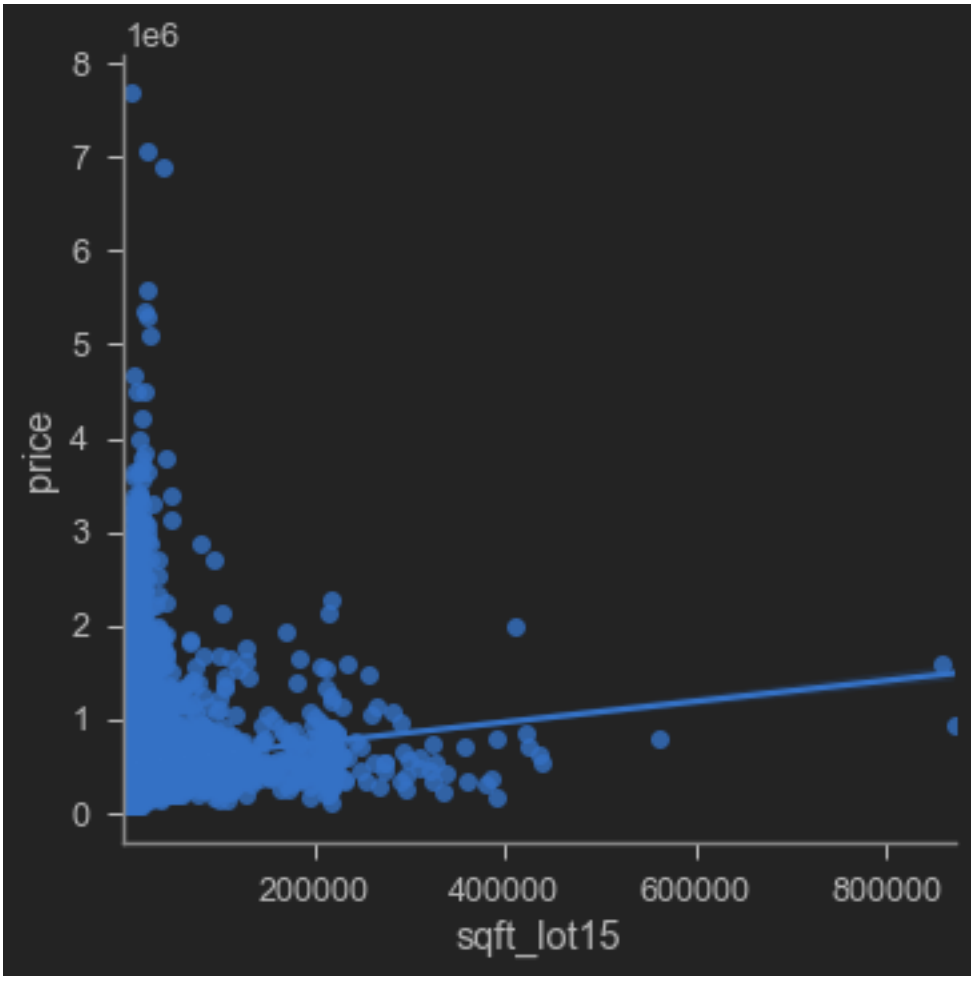

`long`



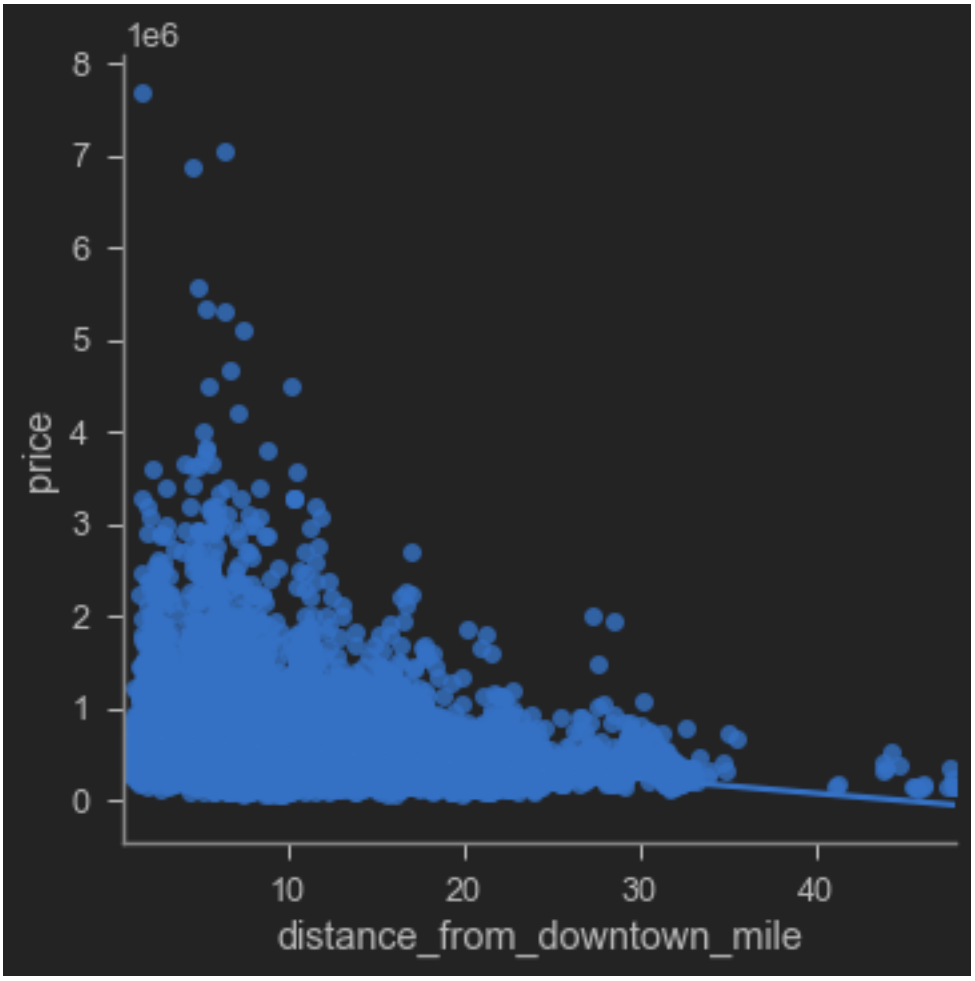

`sqft_living15`


1e6
8
7
6
5
4
3
2
1
0
price
1000
2000
3000
4000
5000
6000
sqft_living15

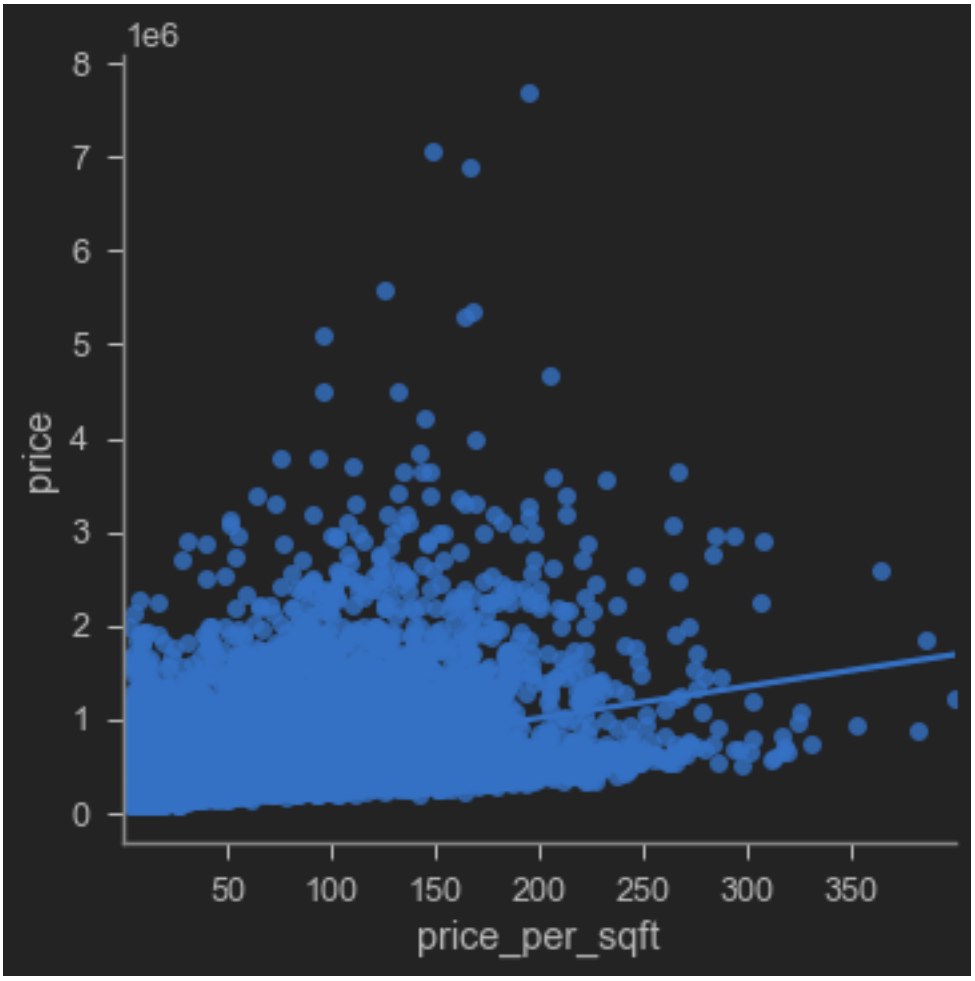
sqft_lot15



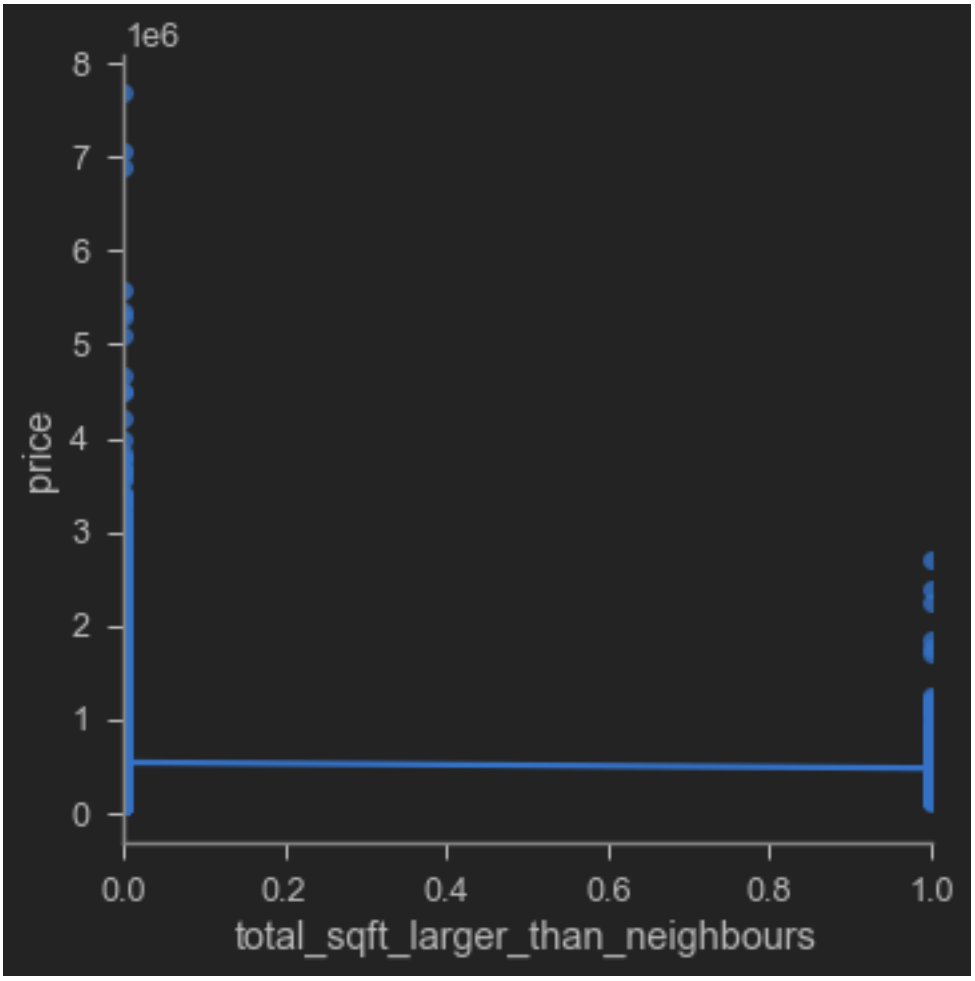

distance_from_downtown_mile



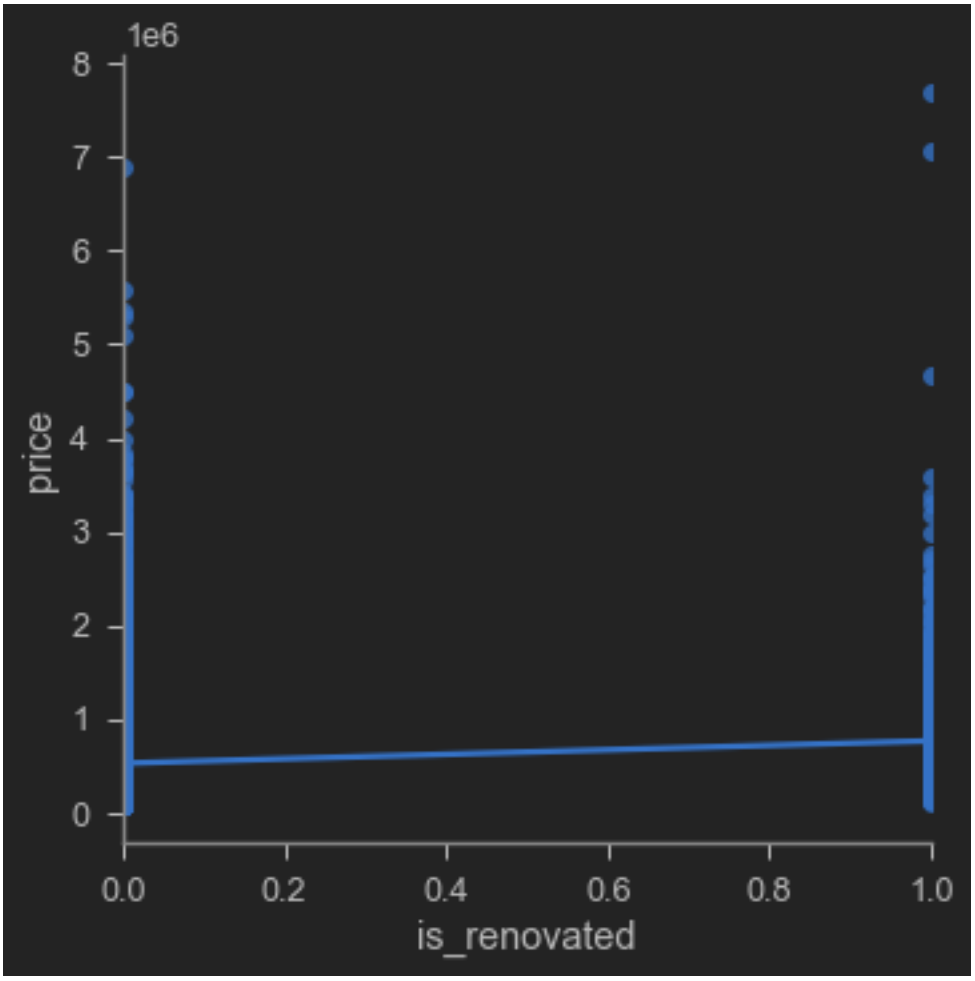

price_per_sqft



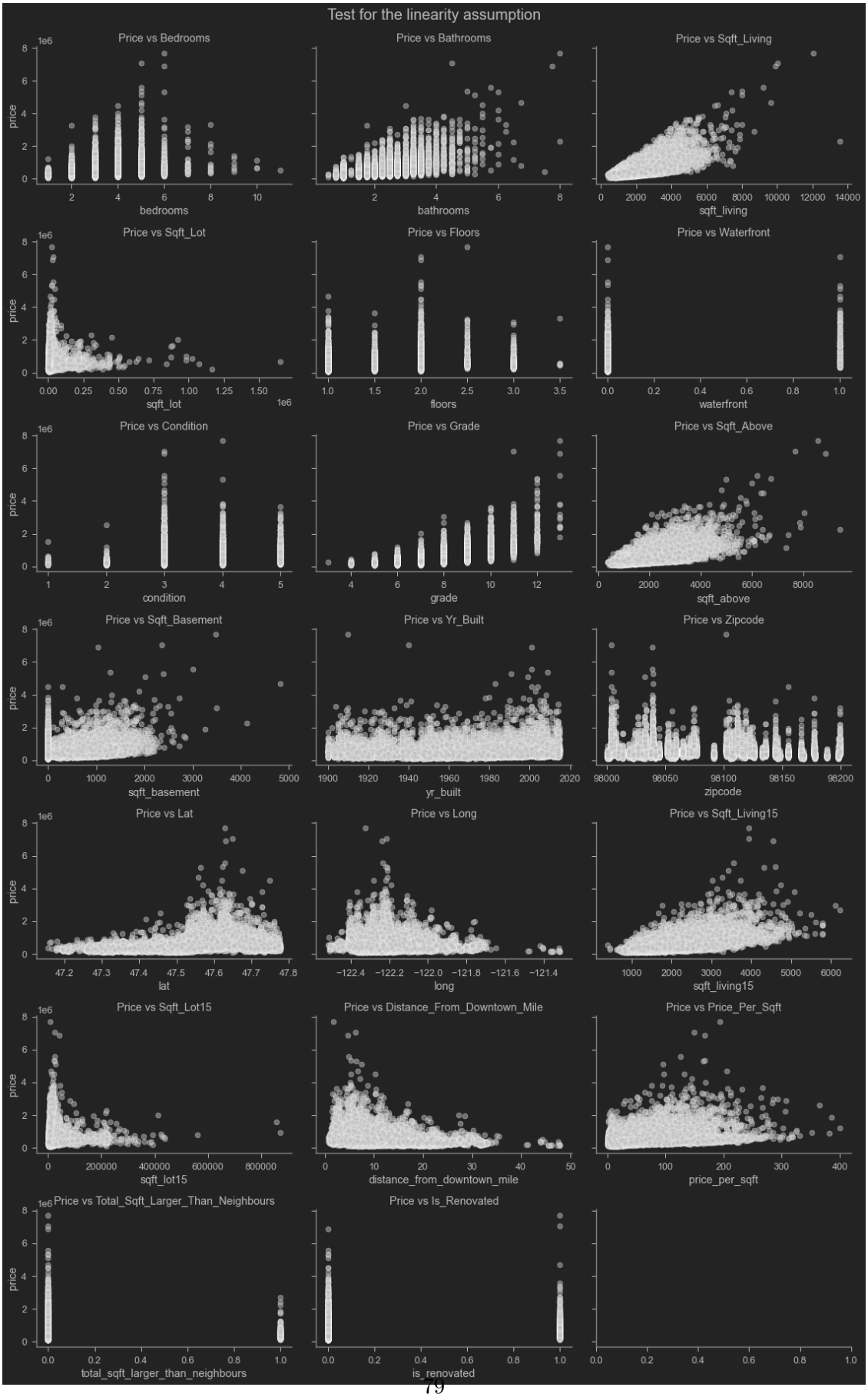

`total_sqft_larger_than_neighbours`




is_renovated




[69]:
```
test_for_linearity(df_model)
```


Test for the linearity assumption
Price vs Bedrooms
price
bedrooms
Price vs Bathrooms
bathrooms
Price vs Sqft_Living
sqft_living
Price vs Sqft_Lot
sqft_lot
Price vs Floors
floors
Price vs Waterfront
waterfront
Price vs Condition
condition
Price vs Grade
grade
Price vs Sqft_Above
sqft_above
Price vs Sqft_Basement
sqft_basement
Price vs Yr_Built
yr_built
Price vs Zipcode
zipcode
Price vs Lat
lat
Price vs Long
long
Price vs Sqft_Living15
sqft_living15
Price vs Sqft_Lot15
sqft_lot15
Price vs Distance_From_Downtown_Mile
distance_from_downtown_mile
Price vs Price_Per_Sqft
price_per_sqft
Price vs Total_Sqft_Larger_Than_Neighbours
total_sqft_larger_than_neighbours
Price vs Is_Renovated
is_renovated

Most of them have some what linear relationship with price, with some features having interesting insights. Those will be explored in later section.

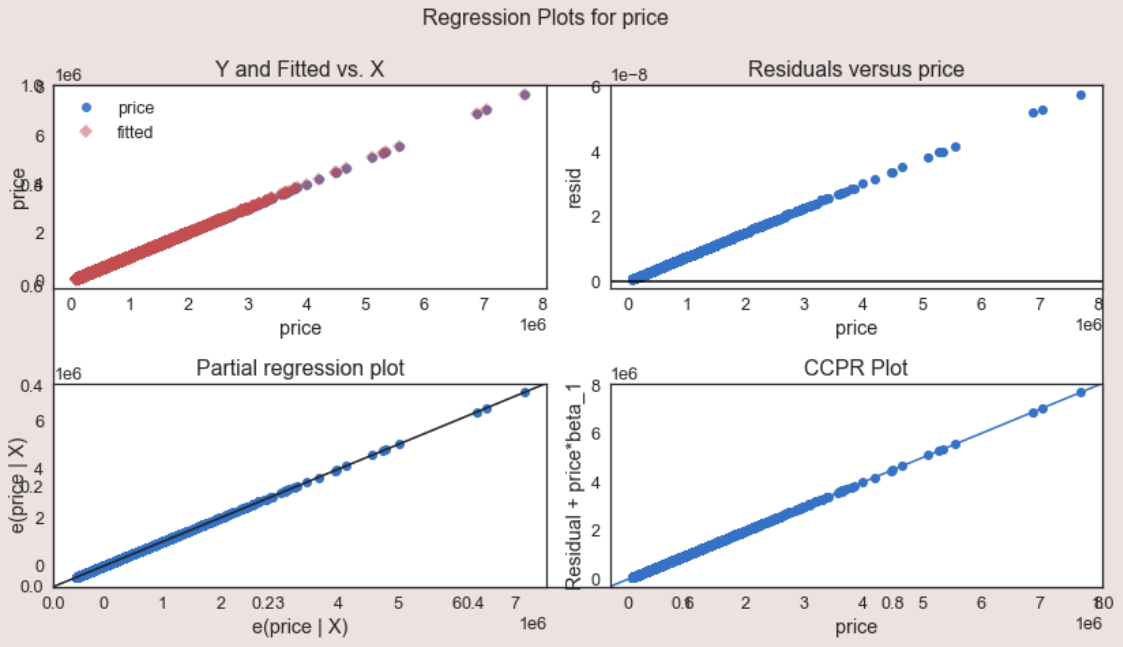
### 4.6.4 Check the normality assumptions

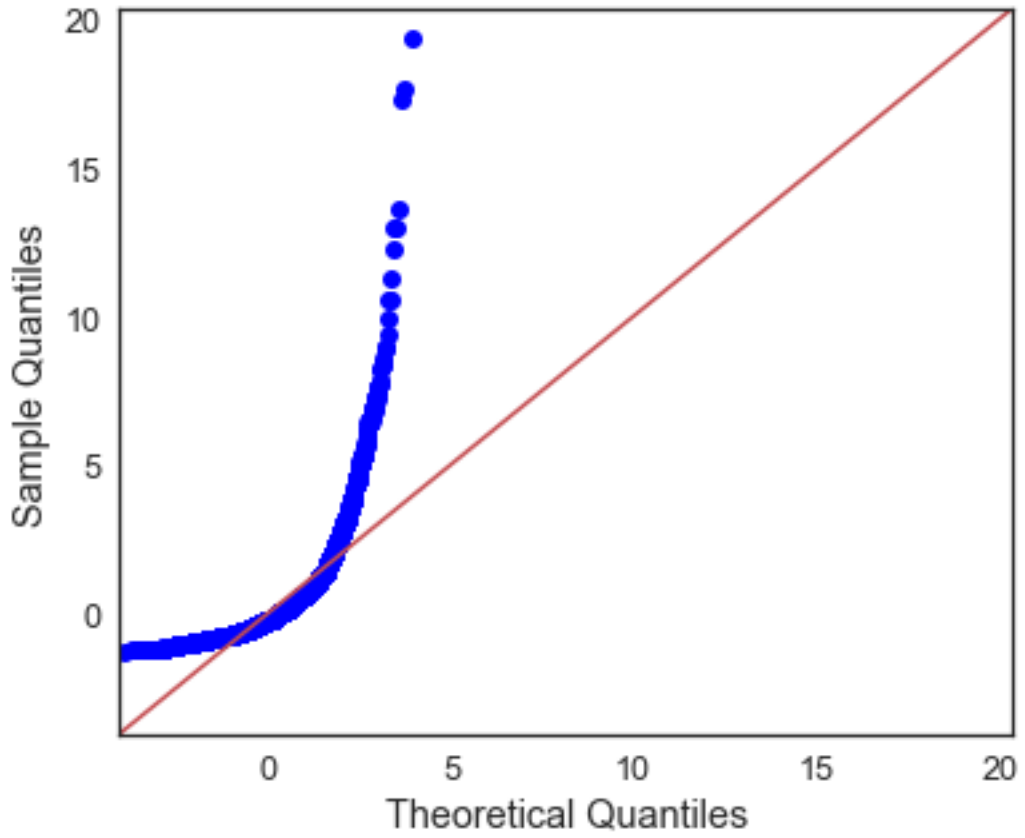
[70]:
```
# really not required, but this also gets me the simle OLS results
# What I did for price, now for ecery feature.
# checks the distribution of errors. Including price, as this will be a partial␣
 ↪regression.
# uses formula method of statsmodels for this.

## Note: A partial regression plot attempts to show the effect of adding␣
 ↪another variable to a model that already has one or more independent␣
 ↪variables
with plt.style.context('seaborn-white'):
    stat_list = []
    for idx, column in enumerate(df_model.columns):
        regression_target = 'price'
        # for dealing with categorical variables
        if column in categorical_feat_model:
            print(f'formula for regression: {f}')
            print(
                f"Regression Analysis and Diagnostics for Price with {column}")
            print(f'Plot {idx+1} of {len(df_model.columns)}')
            print(f'{"-"*90}')

            temp_df = df_model.copy()
            temp_df = temp_df[[regression_target, column]]

            f = f'{regression_target} ~ C({column})'
            f_for_plot = f'{regression_target} ~ {column}'

            model = smf.ols(formula=f, data=temp_df).fit()
            model_for_plot = smf.ols(formula=f_for_plot, data=temp_df).fit()

            fig, axes = plt.subplots(figsize=(12, 7),
                                     facecolor='#ede1e1',
                                     edgecolor='red')
            fig = sm.graphics.plot_regress_exog(model_for_plot,
                                                column,
                                                fig=fig)
            fig.tight_layout()

            sm.graphics.qqplot(model_for_plot.resid,
                               dist=stats.norm,
```

```
                            line='45',
                            fit=True).set_size_inches(6, 5)
            # to be used in later part as ANNOVA.
            temp_dict = {
                'name': column,
                'r_sq': model.rsquared,
                'intercept': model.params[0],
                'beta': model.params[1],
                'p_val': model.pvalues[1],
                'Jarque-Bera': sms.jarque_bera(model.resid)[0]
            }
            stat_list.append(temp_dict)
            plt.show()
        # same stuff for numerical features
        if column in numerical_feat_model:
            f = f'{regression_target} ~ {column}'
            print(f'formula: {f}')
            print(
                f"Regression Analysis and Diagnostics for Price with {column}")
            print(f'Plot {idx+1} of {len(df_model.columns)}')
            print(f'{"-"*90}')
            model = smf.ols(formula=f, data=df_model).fit()

            fig, axes = plt.subplots(figsize=(12, 7),
                                     facecolor='#ede1e1',
                                     edgecolor='red')
            fig = sm.graphics.plot_regress_exog(model, column, fig=fig)
            fig.tight_layout()

            sm.graphics.qqplot(model.resid,
                               dist=stats.norm,
                               line='45',
                               fit=True).set_size_inches(6, 5)
            # to be used later as ANNOVA
            temp_dict = {
                'name': column,
                'r_sq': model.rsquared,
                'intercept': model.params[0],
                'beta': model.params[1],
                'p_val': model.pvalues[1],
                'Jarque-Bera': sms.jarque_bera(model.resid)[0]
            }
            stat_list.append(temp_dict)
            plt.show()
```

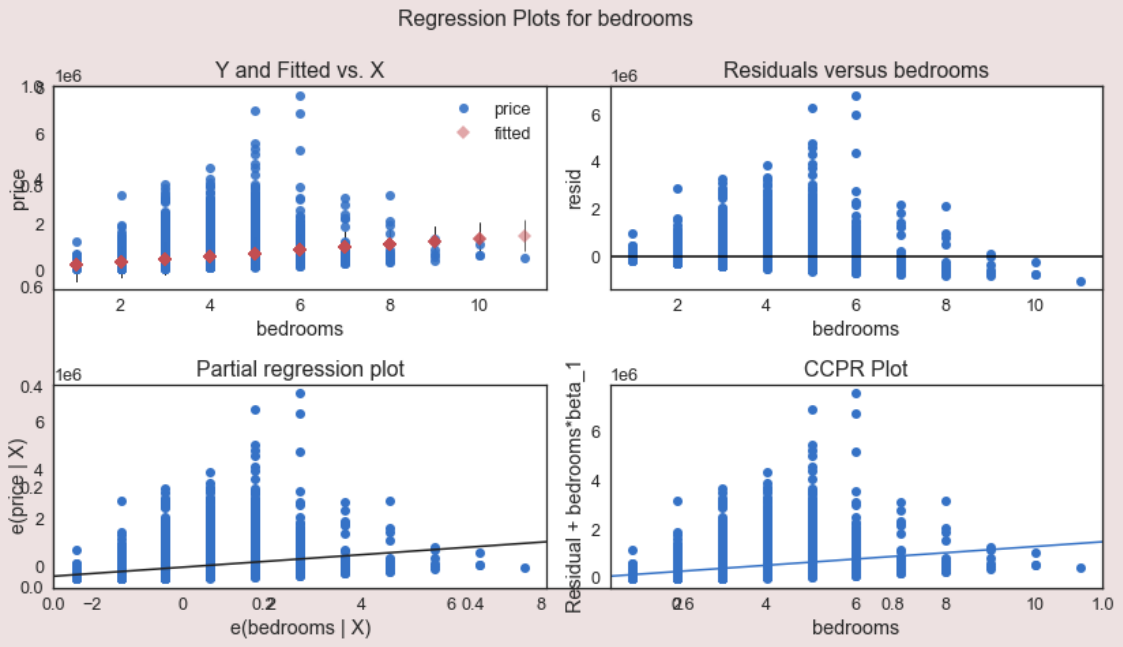
```
formula: price ~ price
Regression Analysis and Diagnostics for Price with price
Plot 1 of 21
```

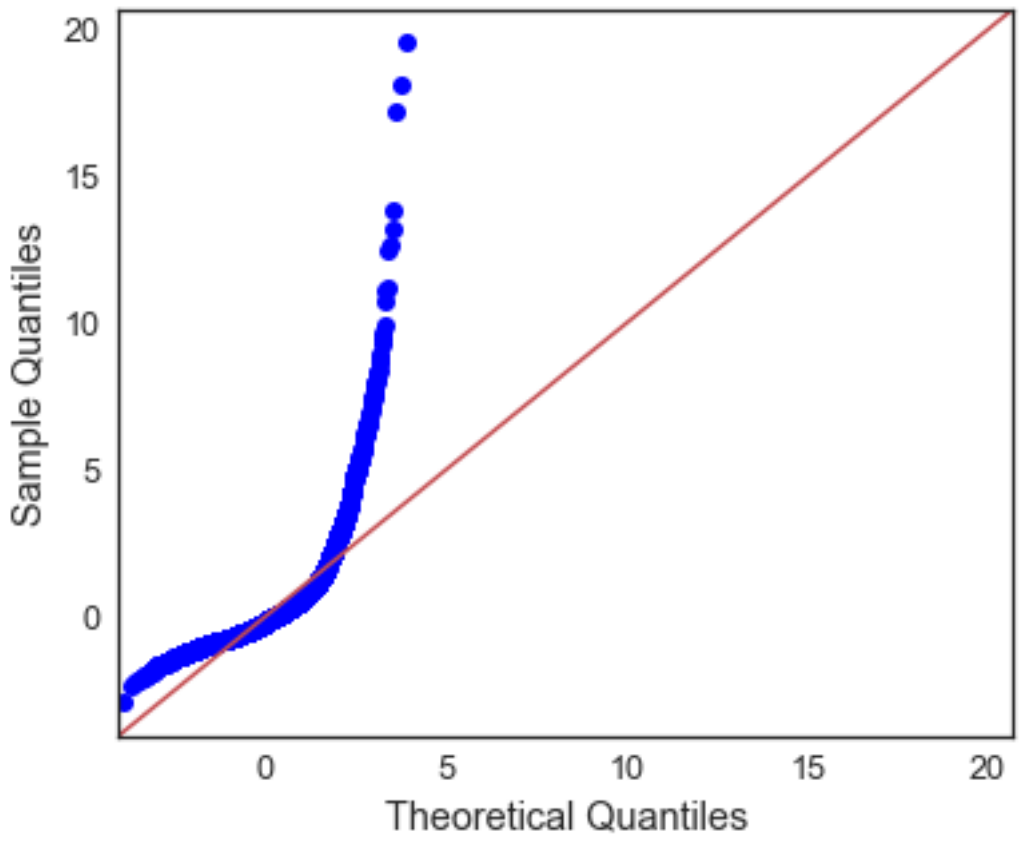
---


Regression Plots for price
Y and Fitted vs. X
price
fitted
price
1e6
Residuals versus price
resid
price
1e−8
1e6
Partial regression plot
e(price | X)
e(price | X)
1e6
CCPR Plot
Residual + price*beta_1
price
1e6

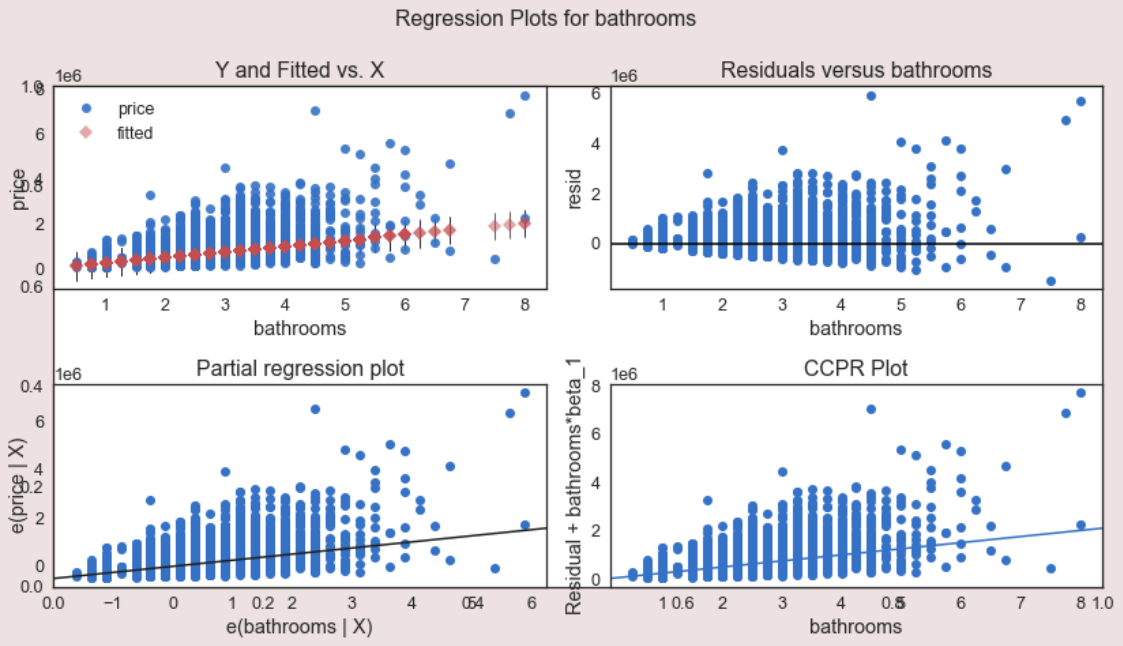

Sample Quantiles
Theoretical Quantiles
0
5
10
15
20

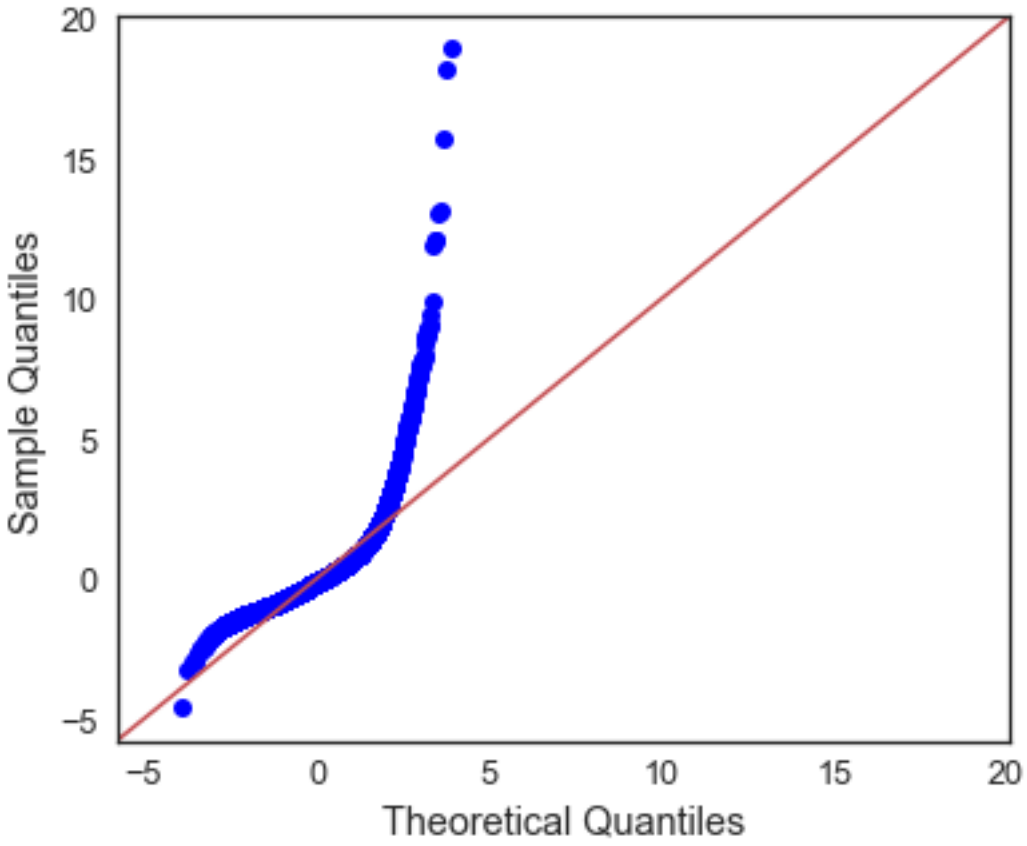
```
formula: price ~ bedrooms
Regression Analysis and Diagnostics for Price with bedrooms
Plot 2 of 21
--------------------------------------------------------------------------------
----------
```



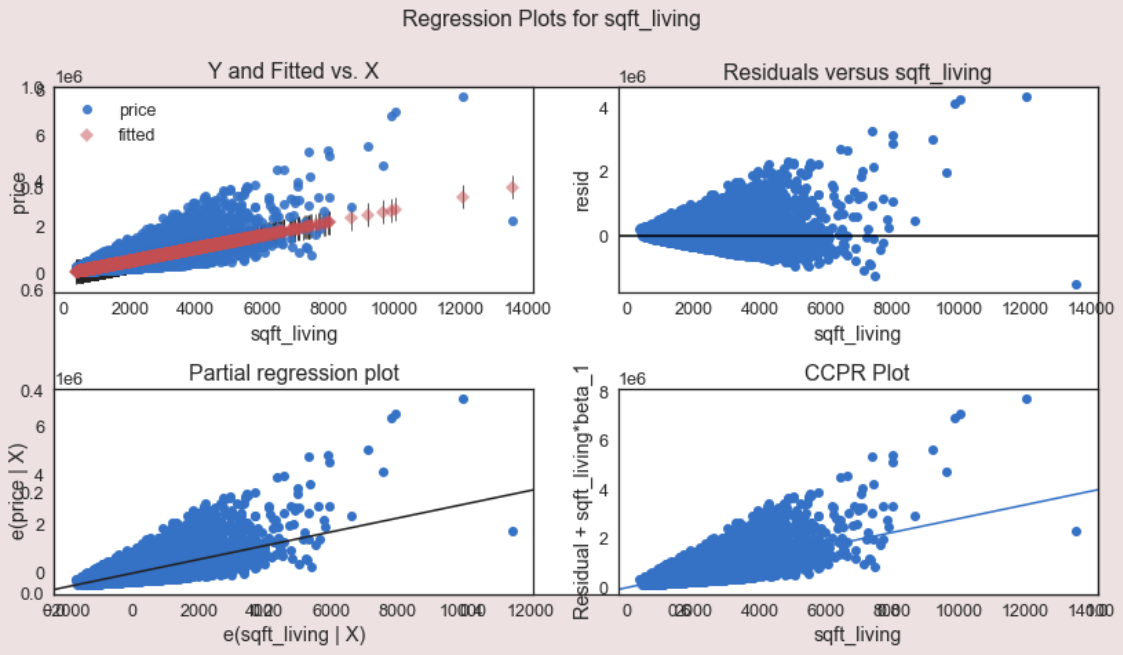


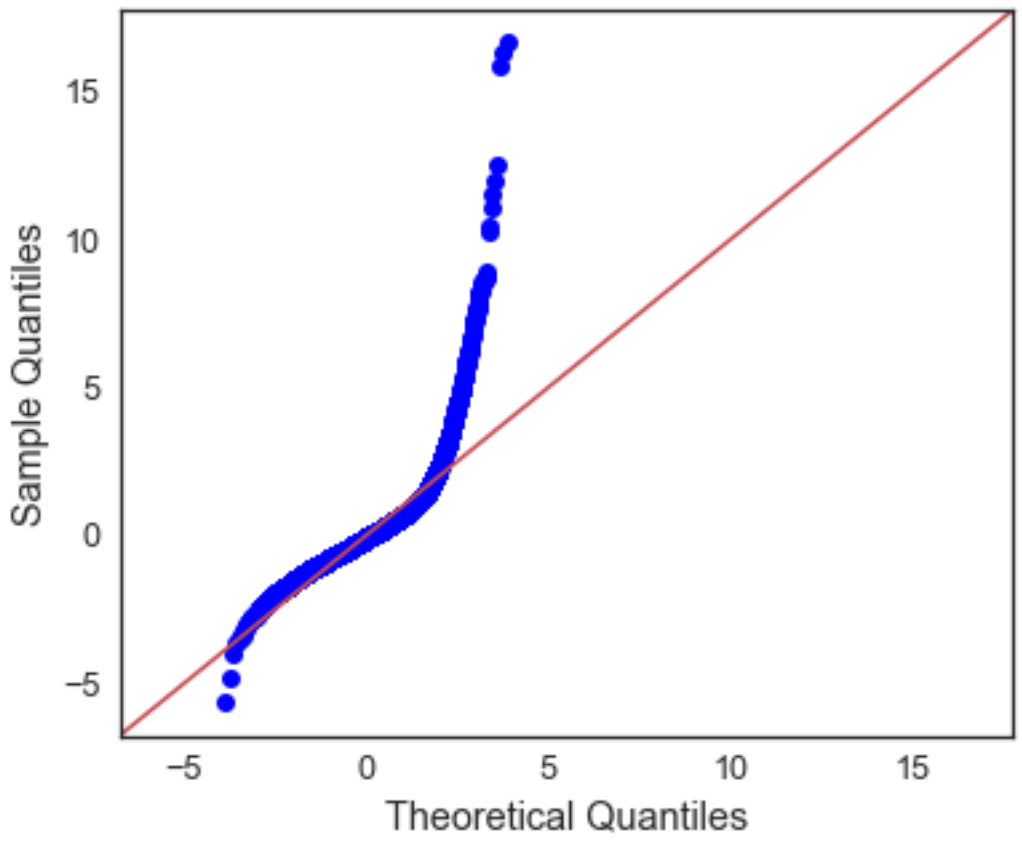

```
formula: price ~ bathrooms
Regression Analysis and Diagnostics for Price with bathrooms
Plot 3 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for bathrooms
Y and Fitted vs. X
price
fitted
bathrooms
Residuals versus bathrooms
resid
bathrooms
Partial regression plot
e(price | X)
e(bathrooms | X)
CCPR Plot
Residual + bathrooms*beta_1
bathrooms

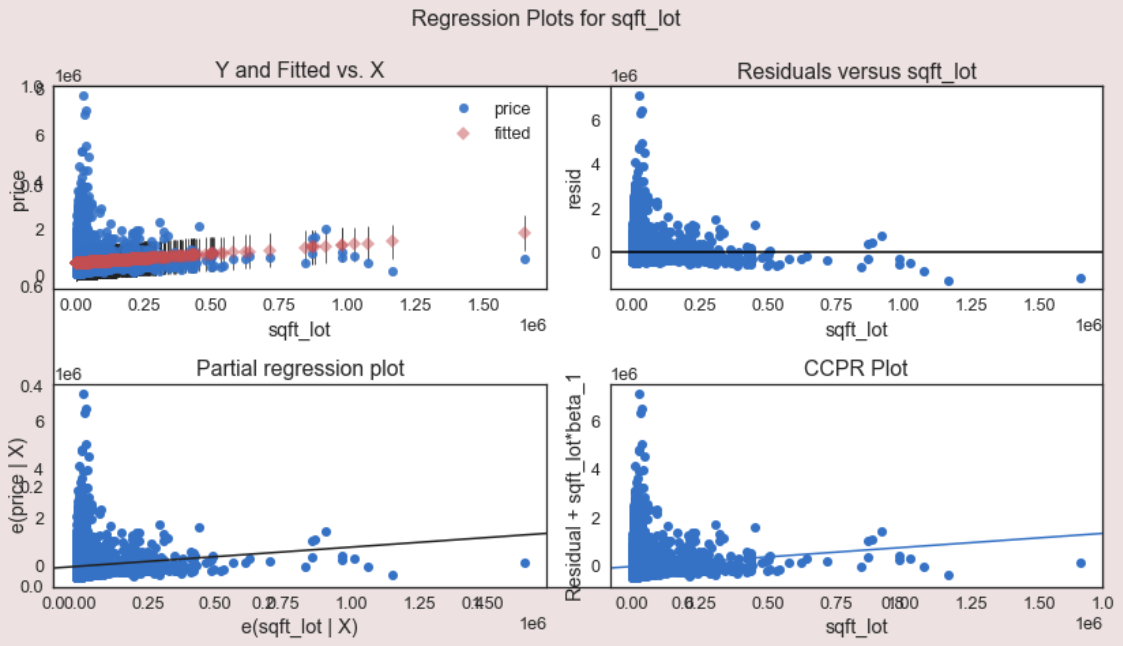

Sample Quantiles
Theoretical Quantiles

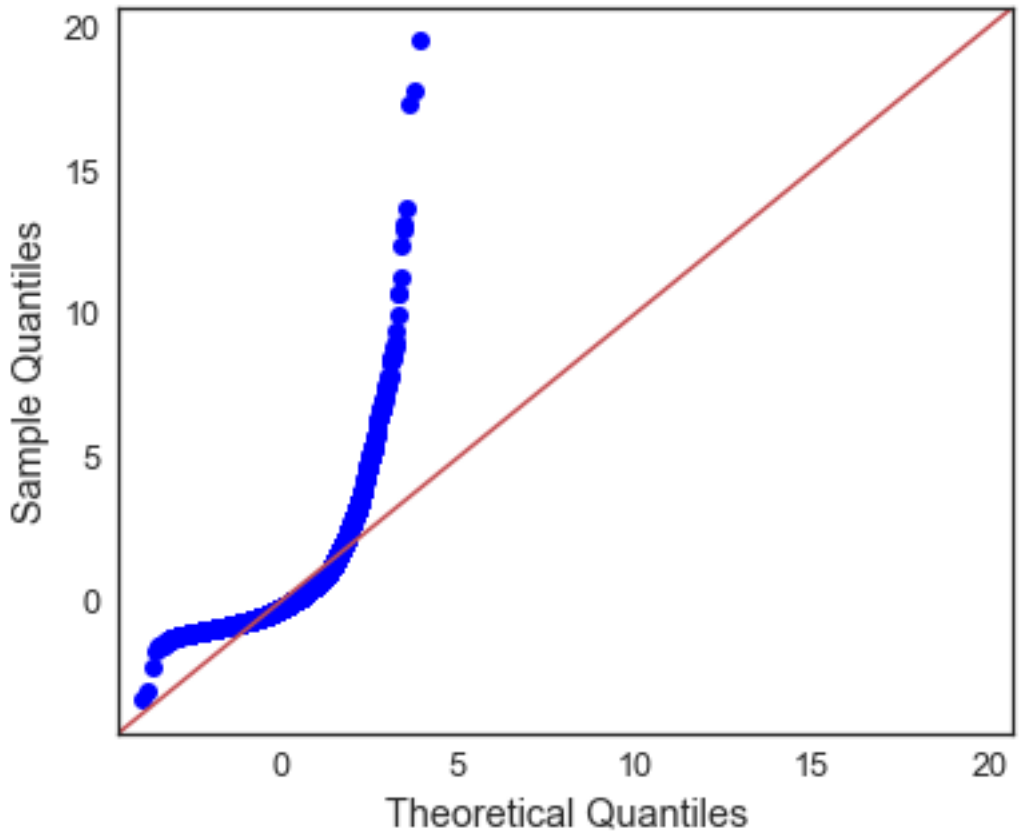
```
formula: price ~ sqft_living
Regression Analysis and Diagnostics for Price with sqft_living
Plot 4 of 21
--------------------------------------------------------------------------------
----------
```



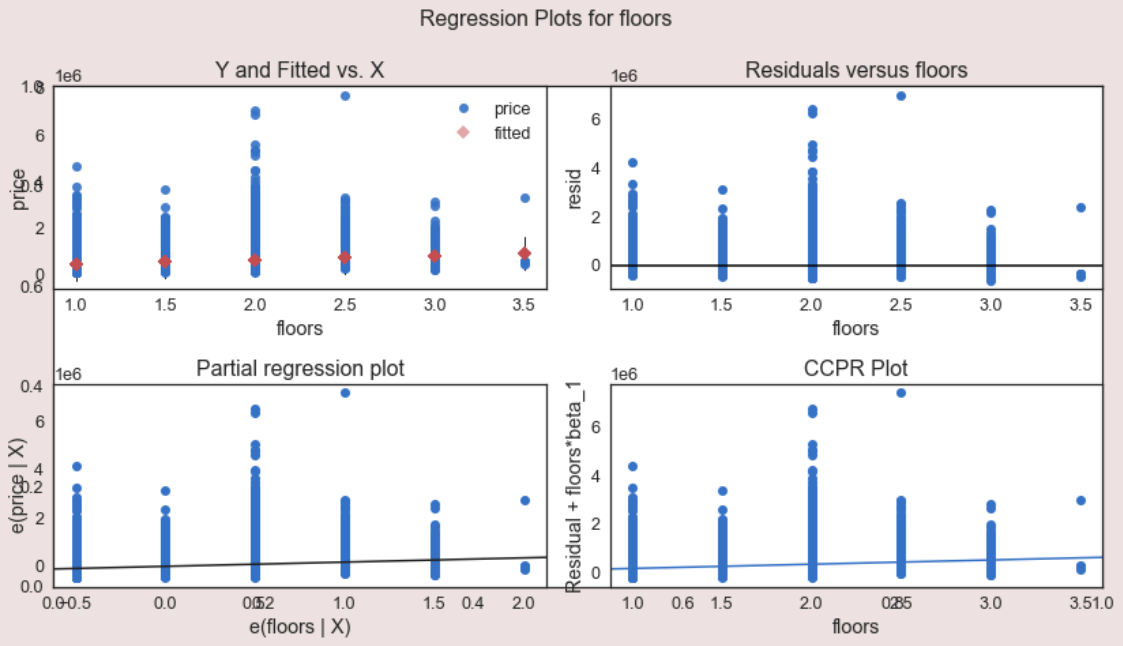


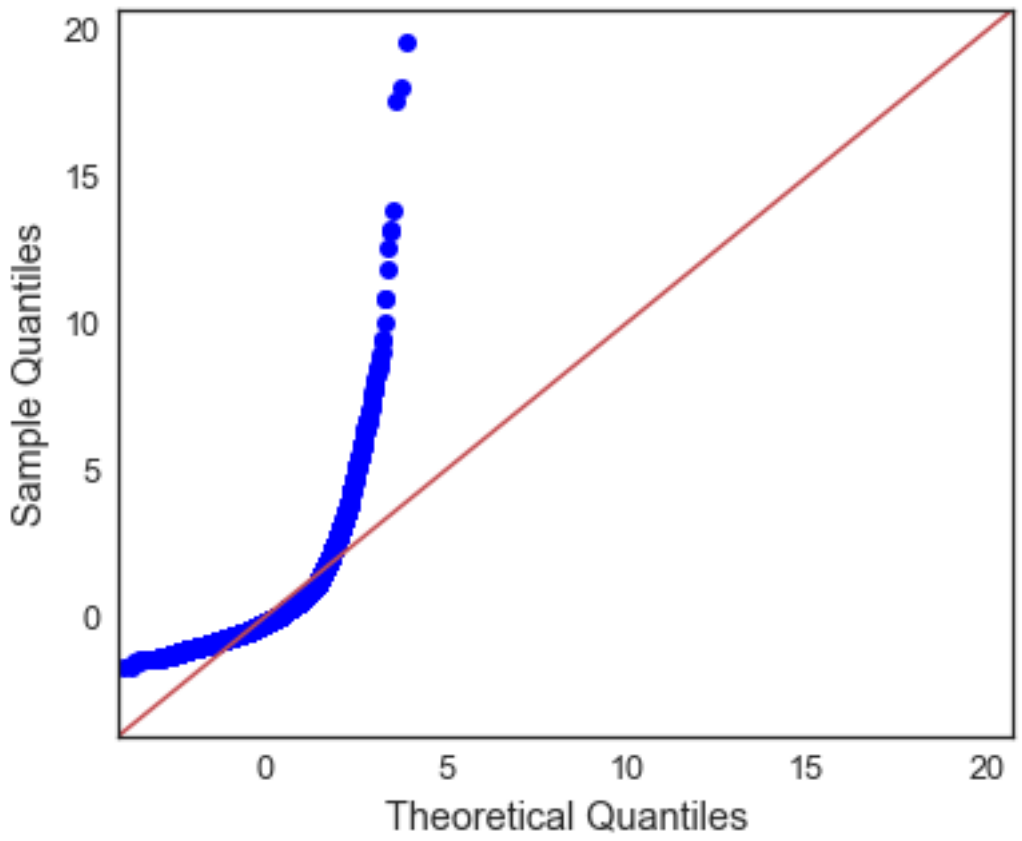

```
formula: price ~ sqft_lot
Regression Analysis and Diagnostics for Price with sqft_lot
Plot 5 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for sqft_lot
Y and Fitted vs. X
price
fitted
sqft_lot
Residuals versus sqft_lot
resid
Partial regression plot
e(price | X)
e(sqft_lot | X)
CCPR Plot
Residual + sqft_lot*beta_1

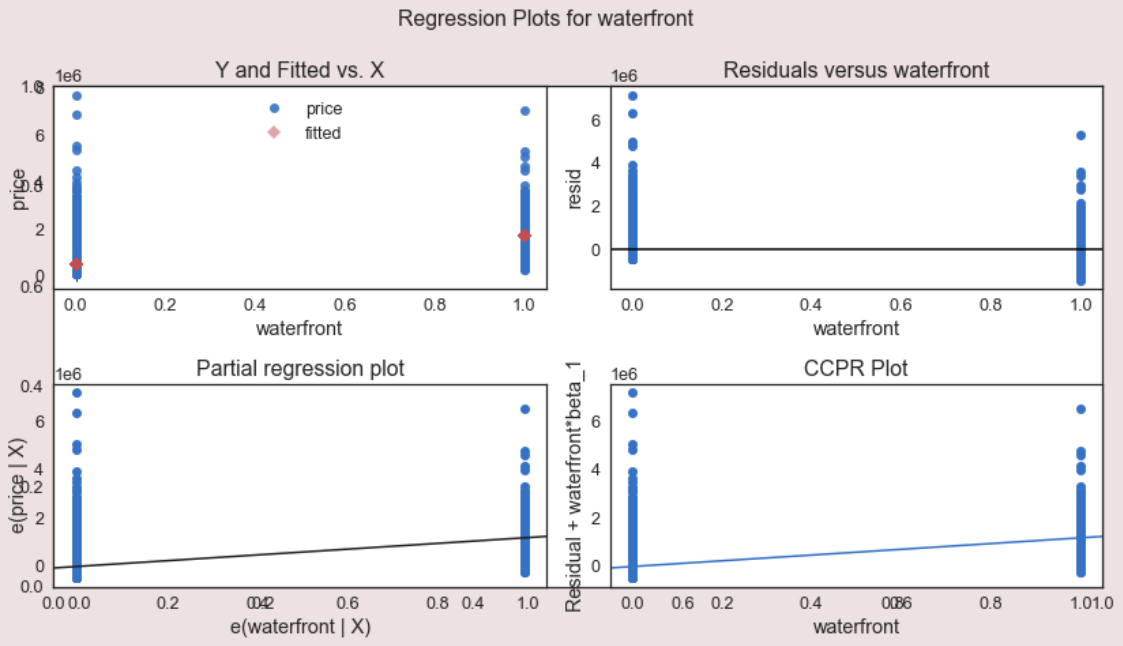

Sample Quantiles
Theoretical Quantiles

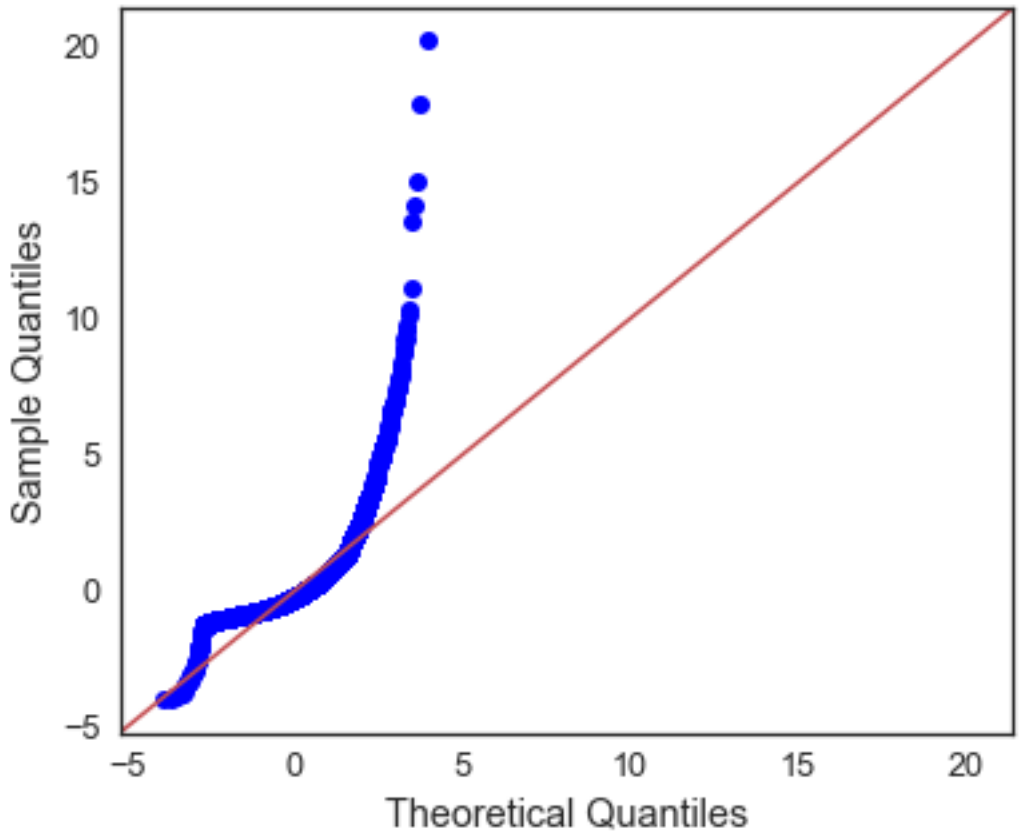
```
formula: price ~ floors
Regression Analysis and Diagnostics for Price with floors
Plot 6 of 21
--------------------------------------------------------------------------------
----------
```



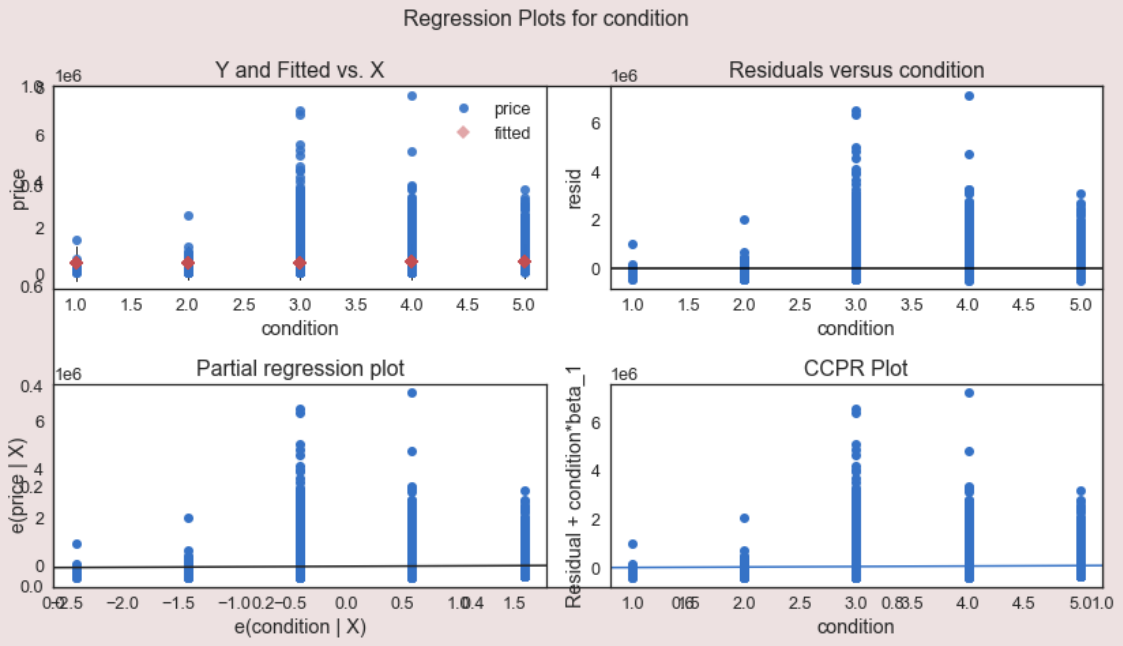


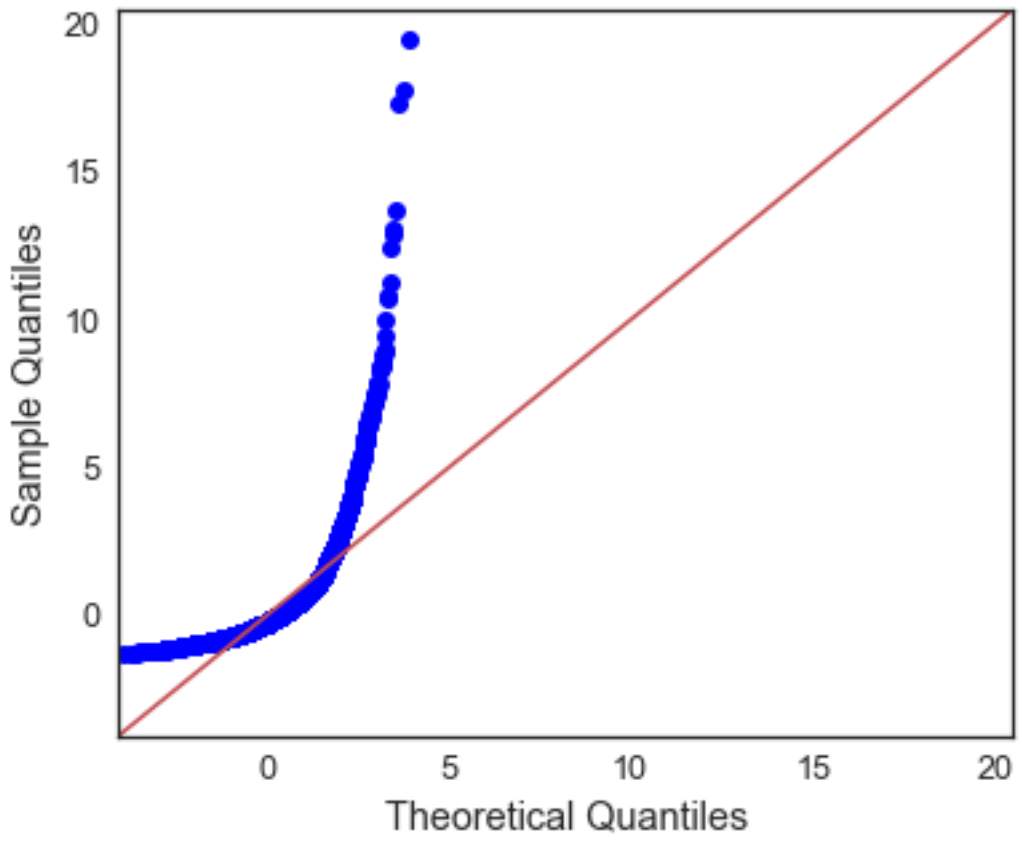

```
formula for regression: price ~ floors
Regression Analysis and Diagnostics for Price with waterfront
Plot 7 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for waterfront
Y and Fitted vs. X
price
fitted
price
waterfront
Residuals versus waterfront
resid
waterfront
Partial regression plot
e(price | X)
e(waterfront | X)
CCPR Plot
Residual + waterfront*beta_1
waterfront

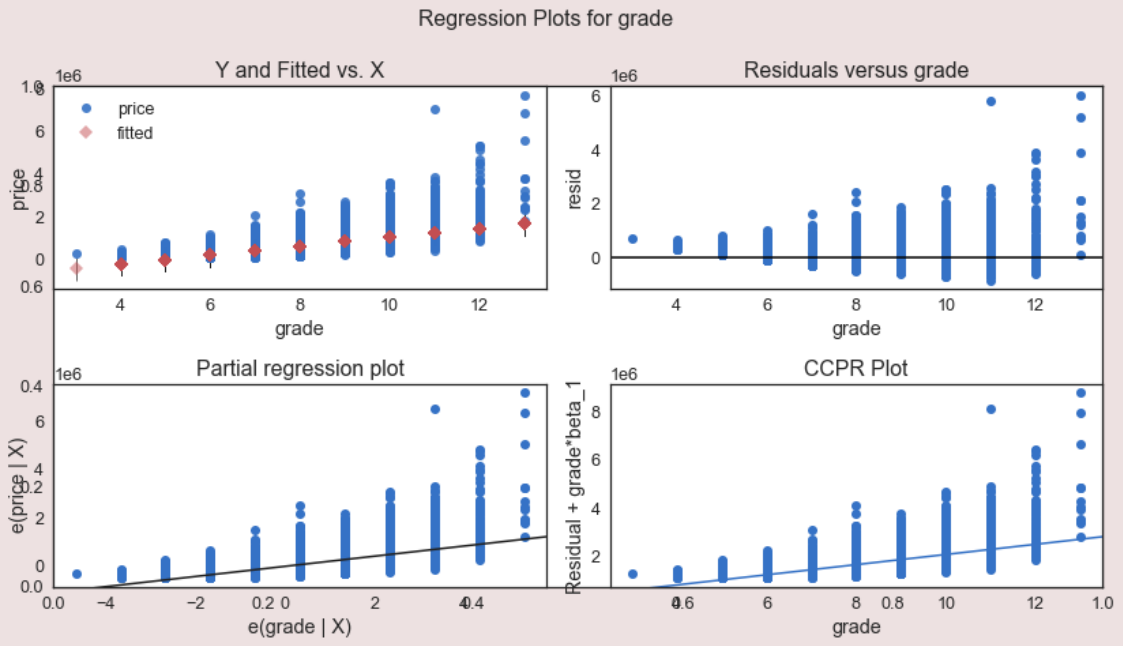

Sample Quantiles
Theoretical Quantiles

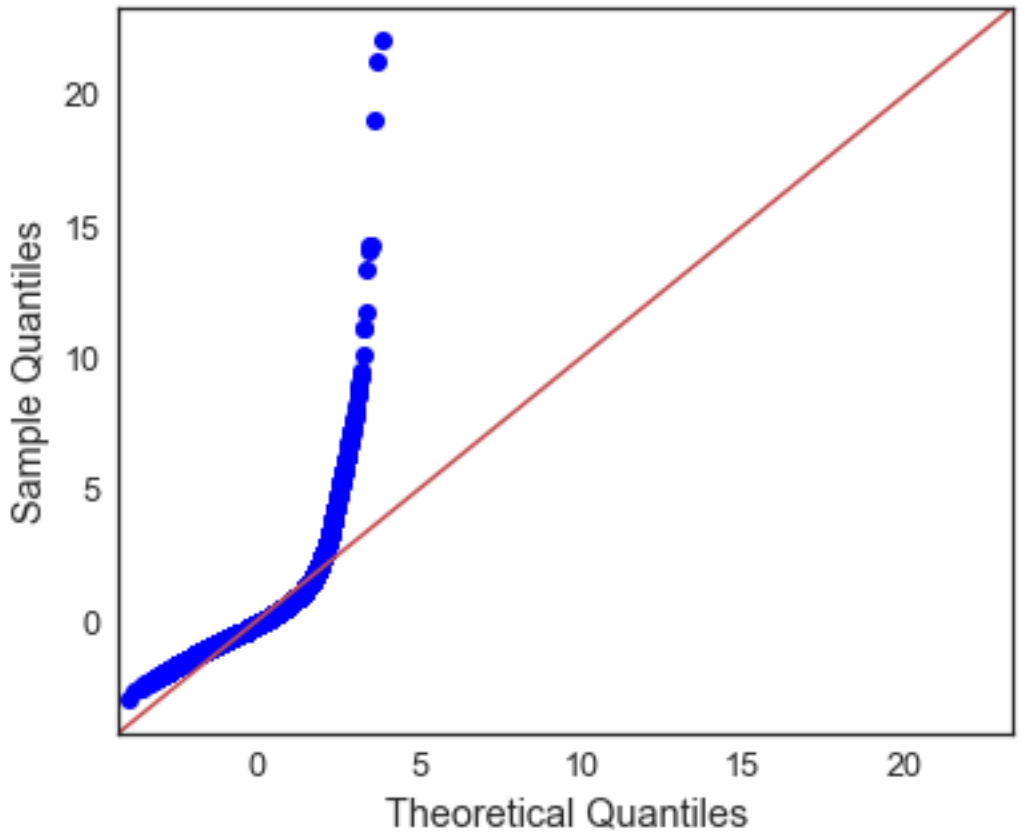
```
formula for regression: price ~ C(waterfront)
Regression Analysis and Diagnostics for Price with condition
Plot 8 of 21
--------------------------------------------------------------------------------
----------
```



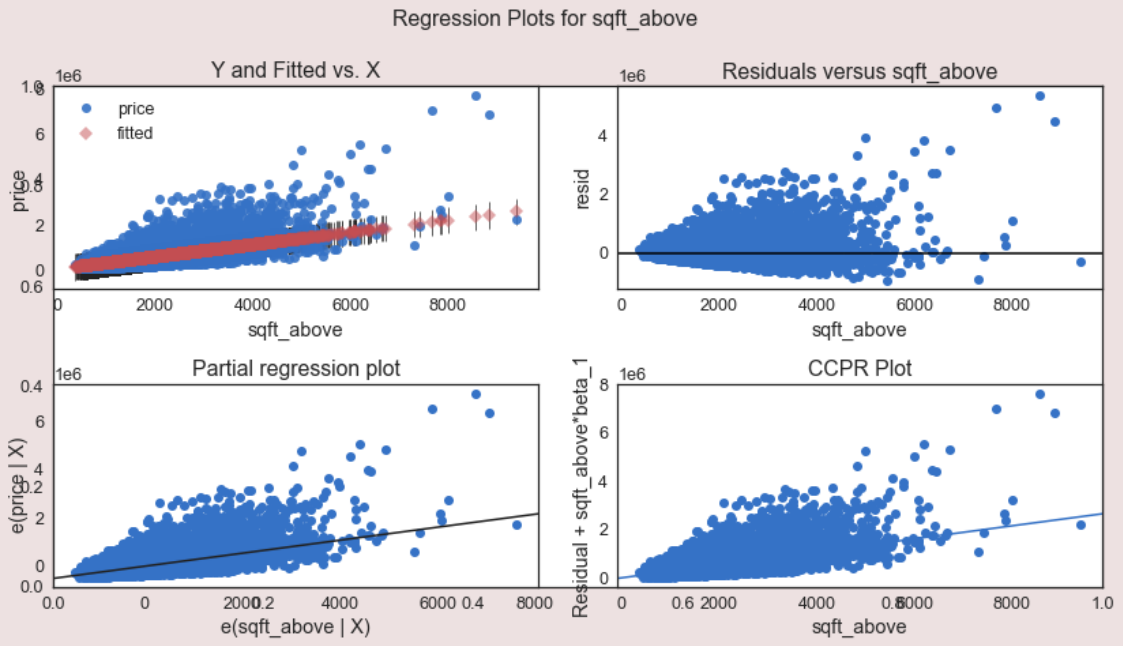


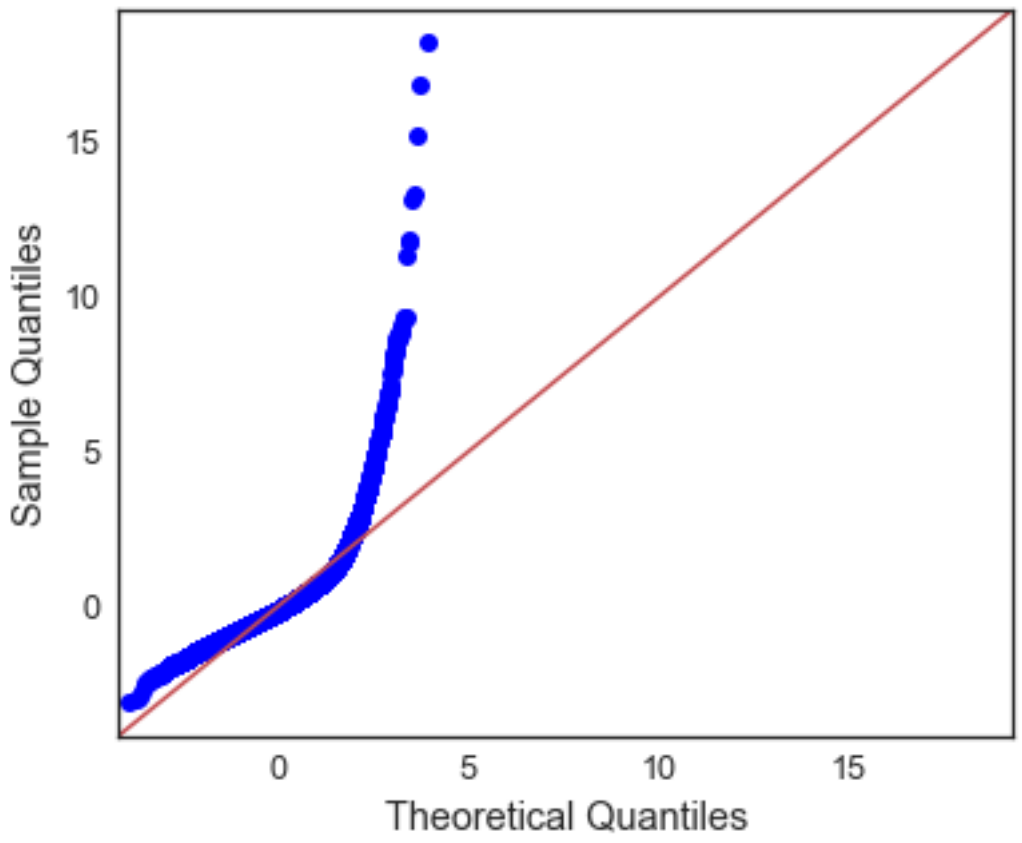

```
formula for regression: price ~ C(condition)
Regression Analysis and Diagnostics for Price with grade
Plot 9 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for grade
Y and Fitted vs. X
price
fitted
price
grade
Residuals versus grade
resid
grade
Partial regression plot
e(price | X)
e(grade | X)
CCPR Plot
Residual + grade*beta_1
grade

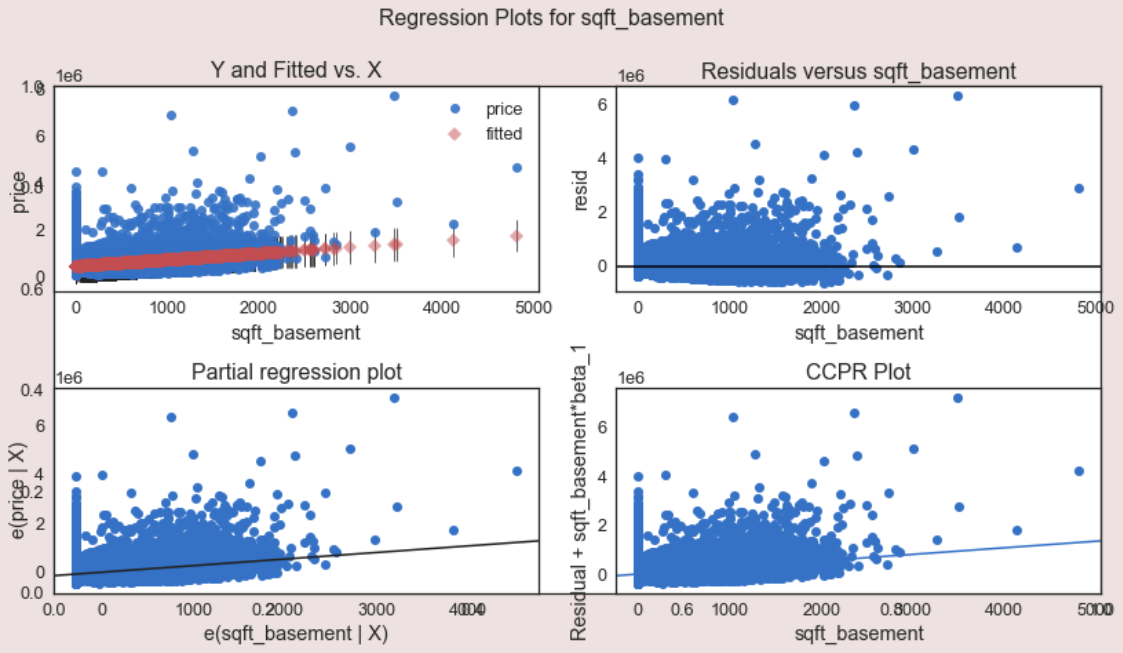

Sample Quantiles
Theoretical Quantiles

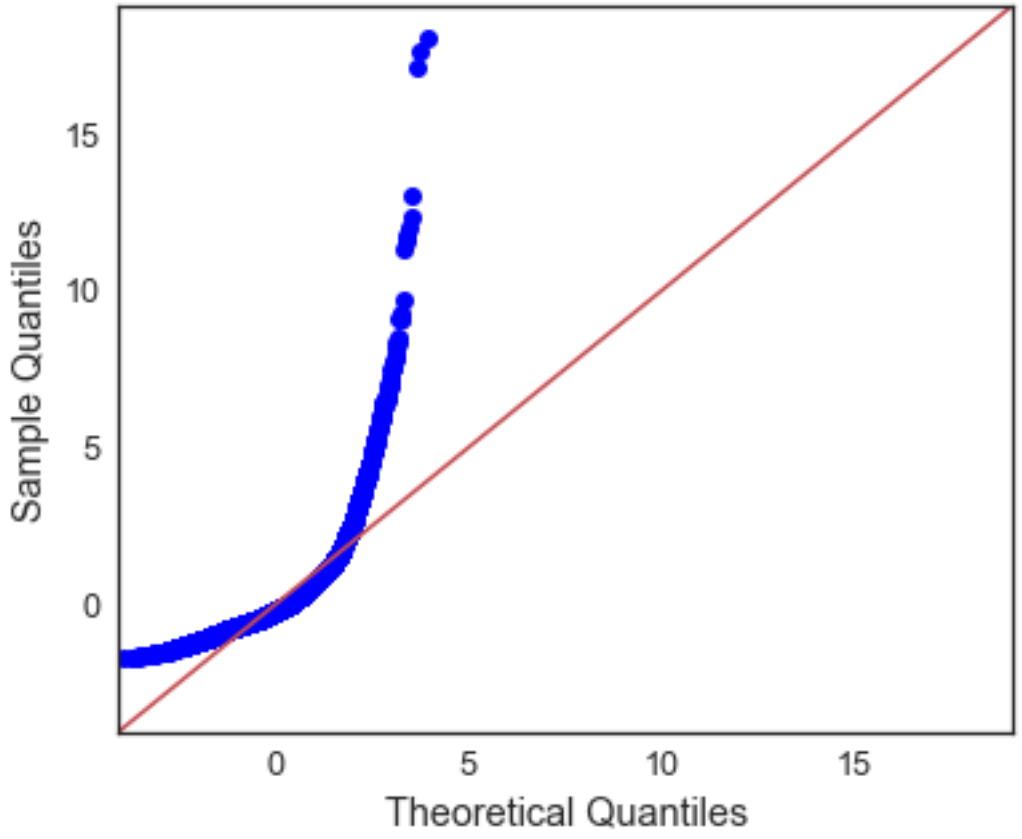
```
formula: price ~ sqft_above
Regression Analysis and Diagnostics for Price with sqft_above
Plot 10 of 21
--------------------------------------------------------------------------------
----------
```



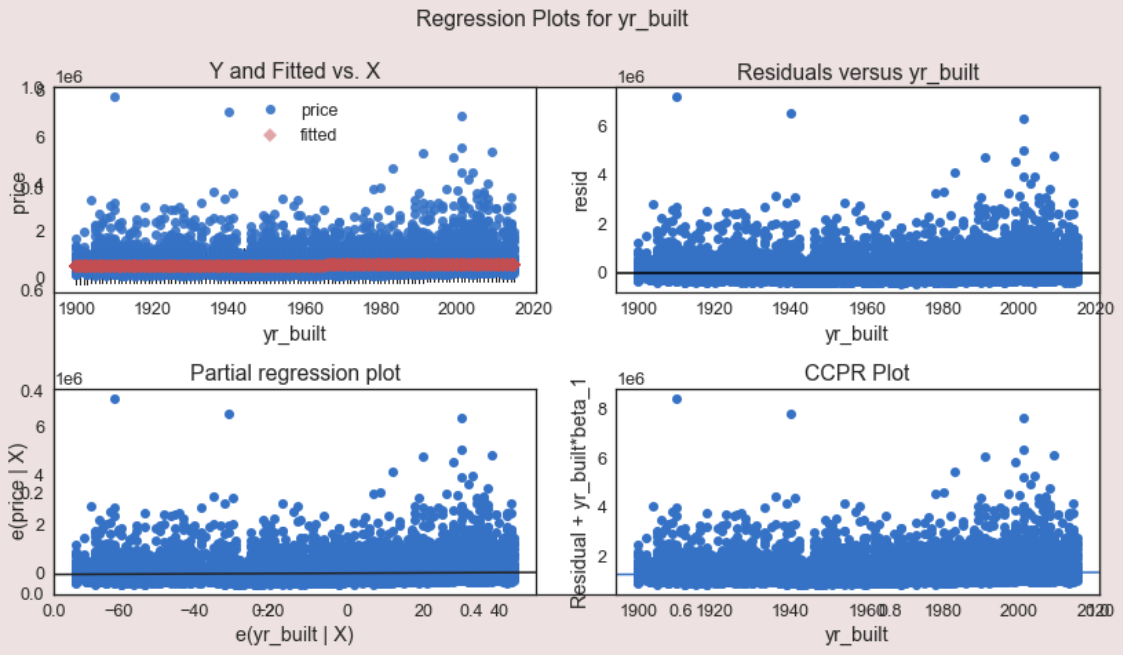


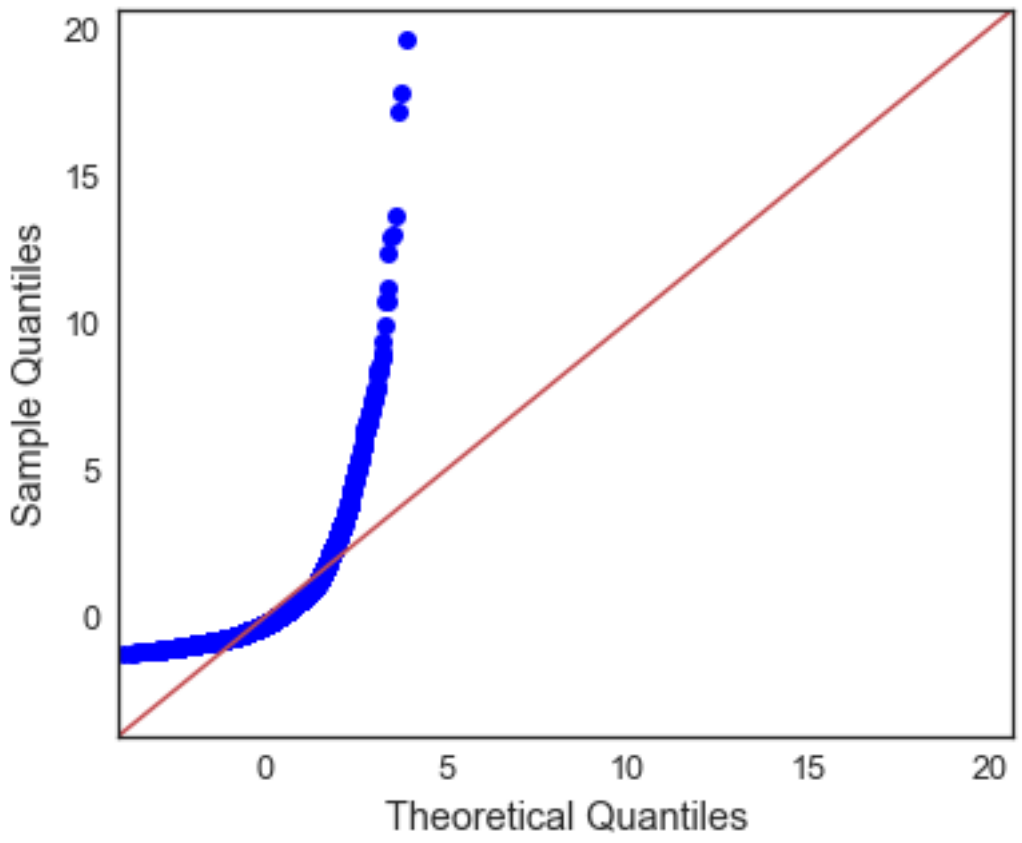

```
formula: price ~ sqft_basement
Regression Analysis and Diagnostics for Price with sqft_basement
Plot 11 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for sqft_basement
Y and Fitted vs. X
price
fitted
sqft_basement
Residuals versus sqft_basement
resid
sqft_basement
Partial regression plot
e(price | X)
e(sqft_basement | X)
CCPR Plot
Residual + sqft_basement*beta_1
sqft_basement

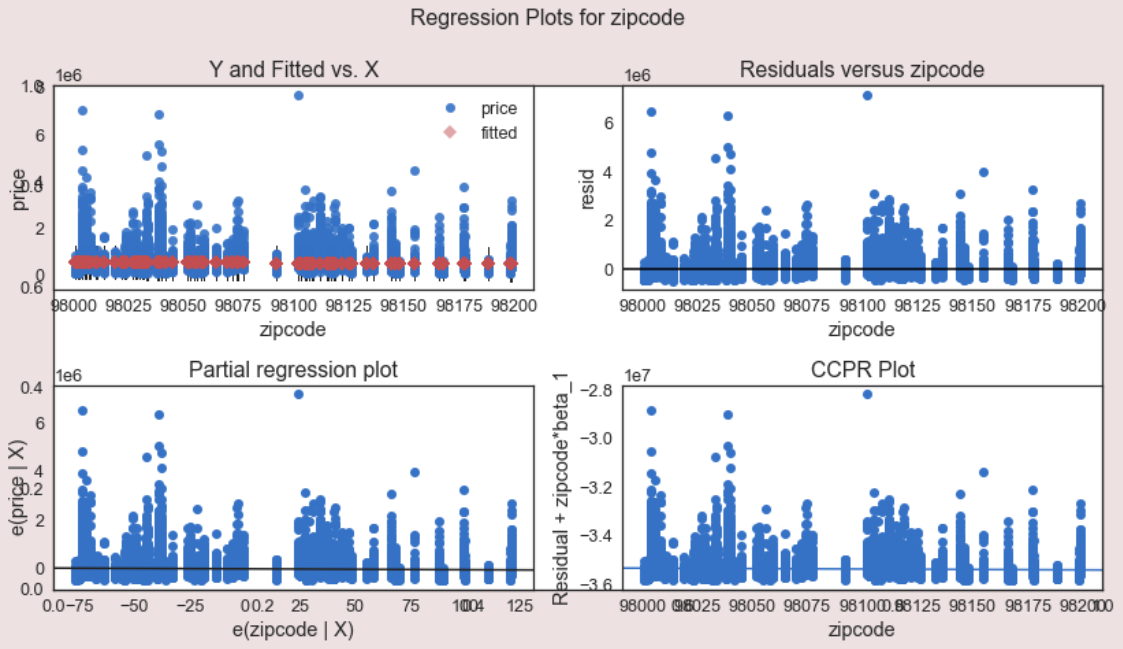

Sample Quantiles
Theoretical Quantiles

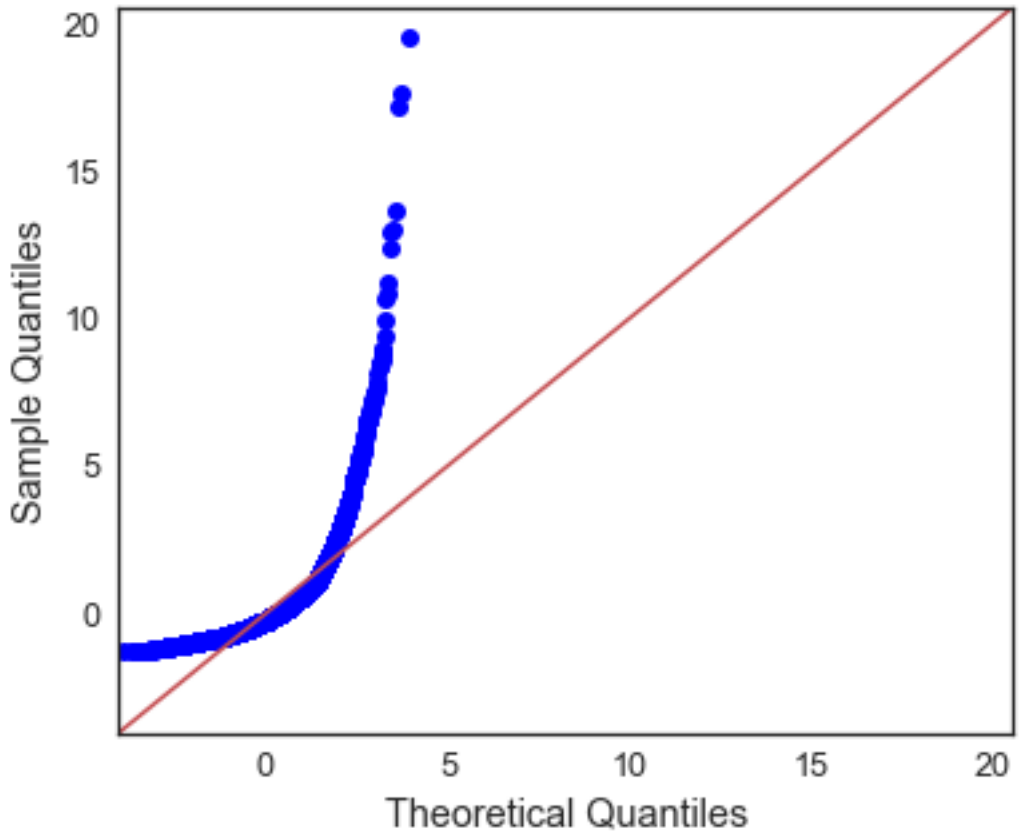
```
formula: price ~ yr_built
Regression Analysis and Diagnostics for Price with yr_built
Plot 12 of 21
--------------------------------------------------------------------------------
----------
```



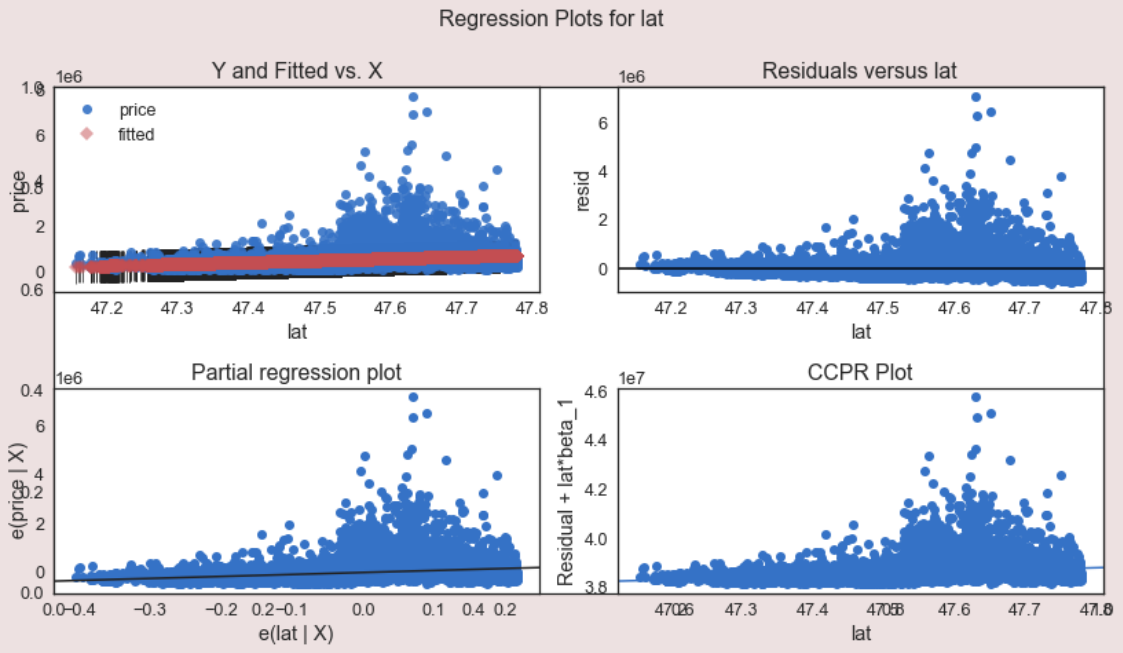


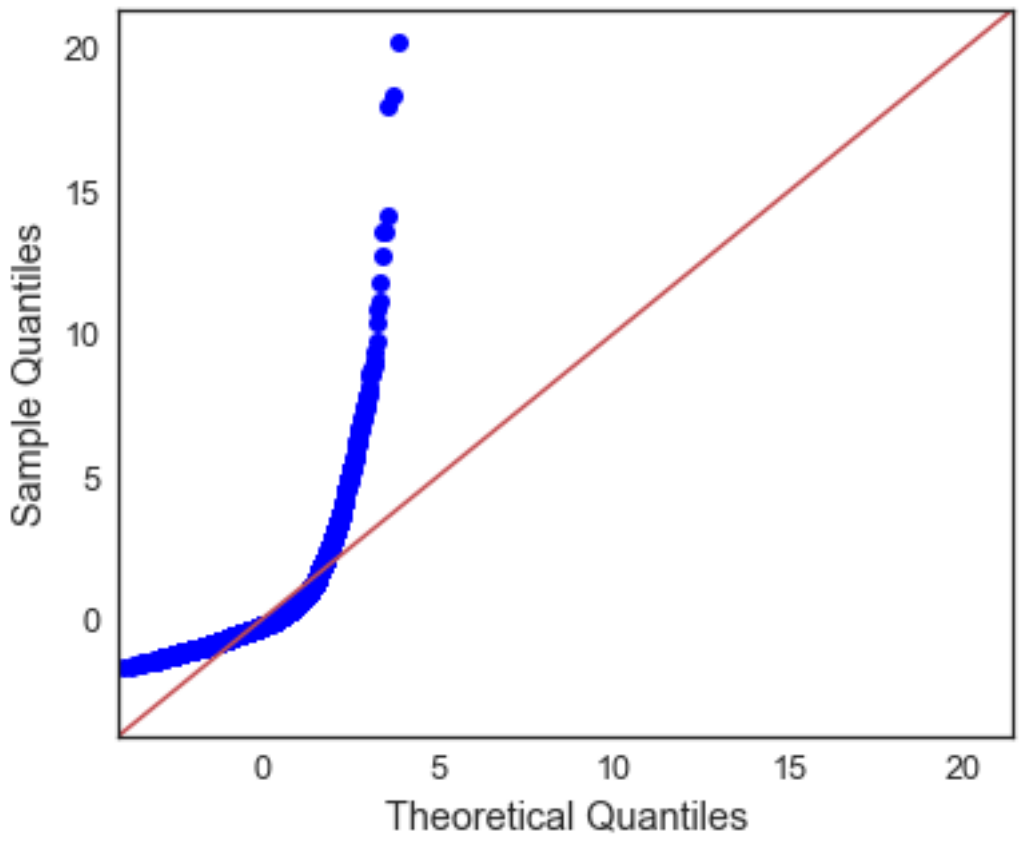

```
formula for regression: price ~ yr_built
Regression Analysis and Diagnostics for Price with zipcode
Plot 13 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for zipcode
Y and Fitted vs. X
price
fitted
zipcode
Residuals versus zipcode
resid
zipcode
Partial regression plot
e(price | X)
e(zipcode | X)
CCPR Plot
Residual + zipcode*beta_1
zipcode

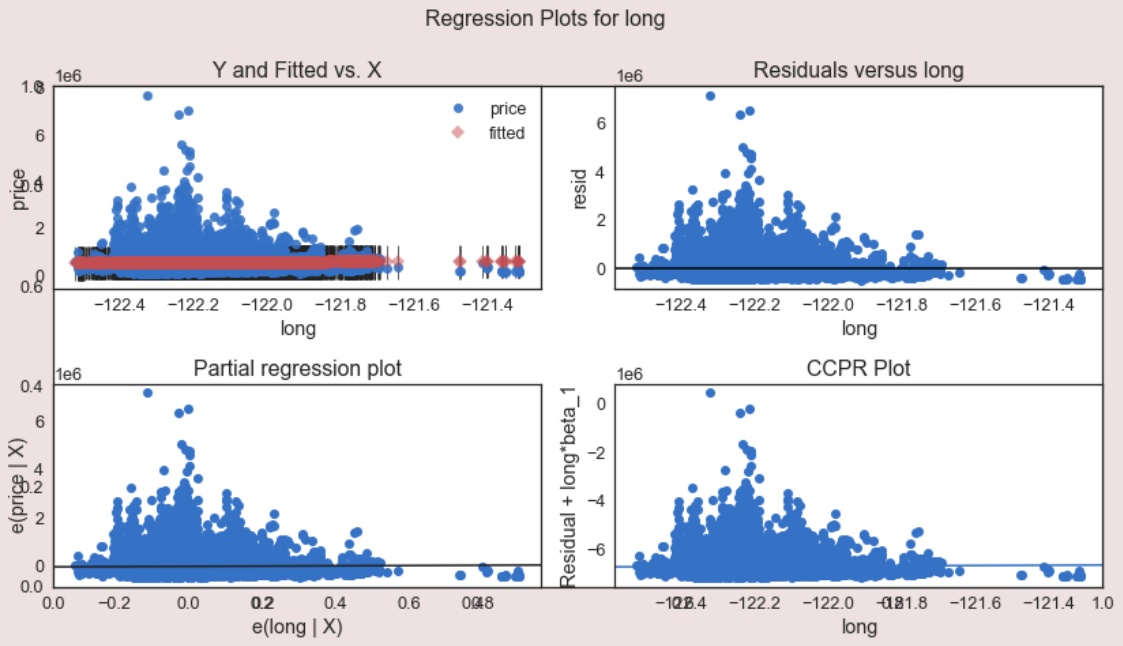

Sample Quantiles
Theoretical Quantiles

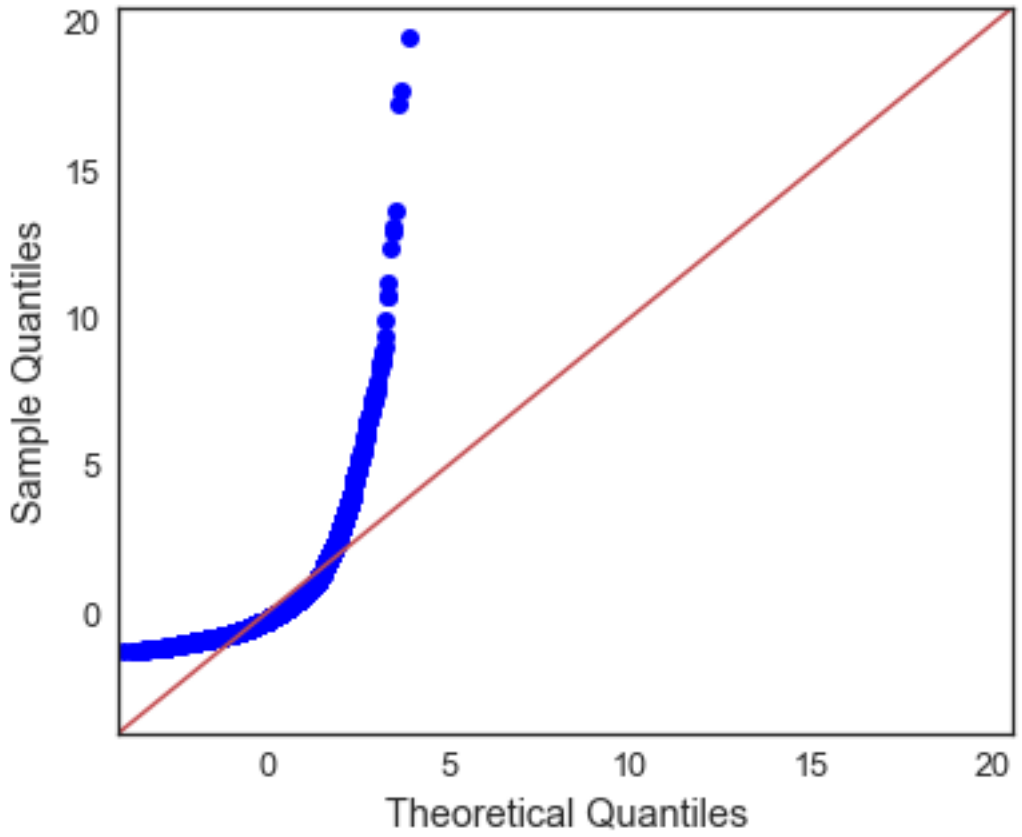
```
formula: price ~ lat
Regression Analysis and Diagnostics for Price with lat
Plot 14 of 21
--------------------------------------------------------------------------------
----------
```



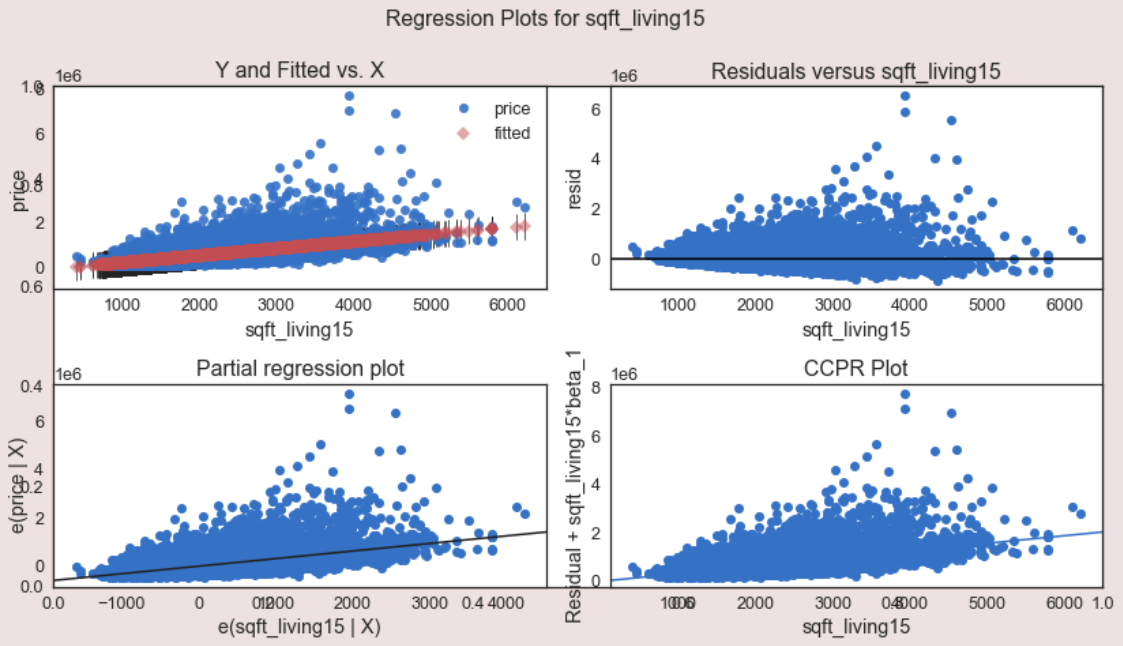


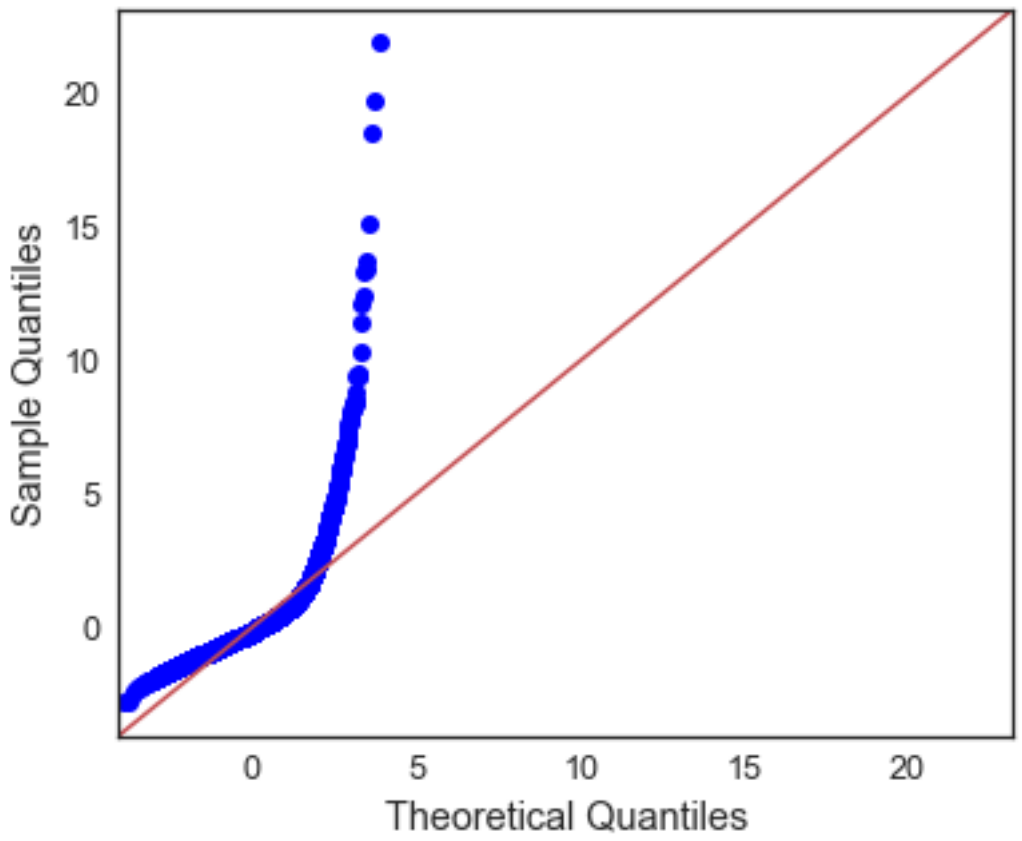

```
formula: price ~ long
Regression Analysis and Diagnostics for Price with long
Plot 15 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for long
Y and Fitted vs. X
price
fitted
Residuals versus long
resid
long
Partial regression plot
e(price | X)
e(long | X)
CCPR Plot
Residual + long*beta_1
1e6

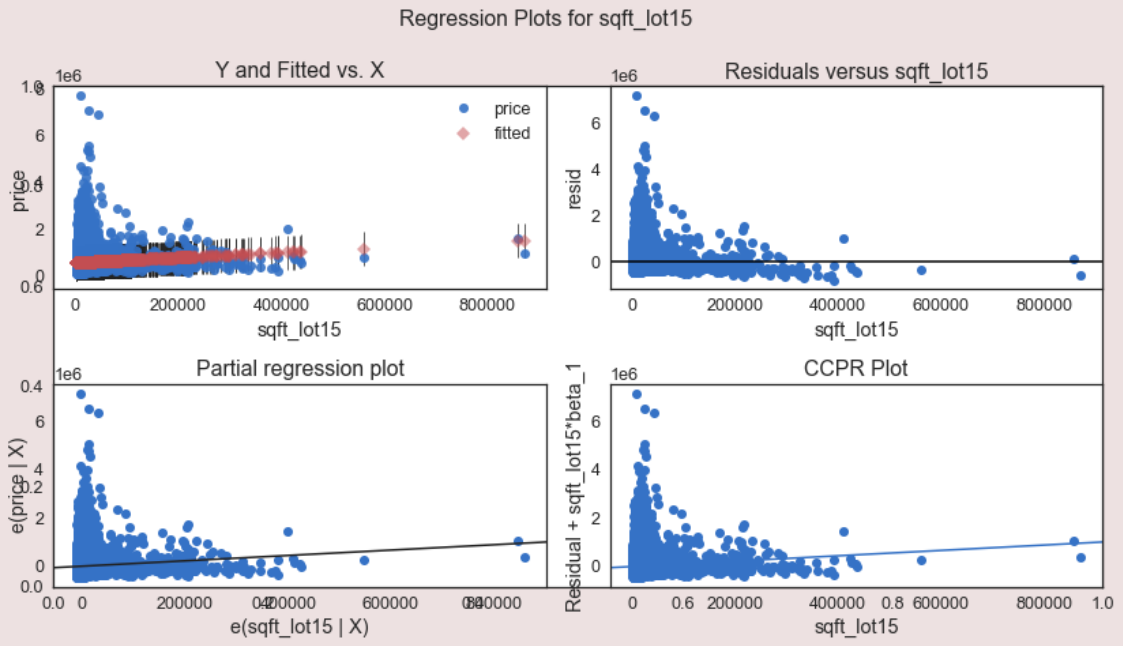

Sample Quantiles
Theoretical Quantiles

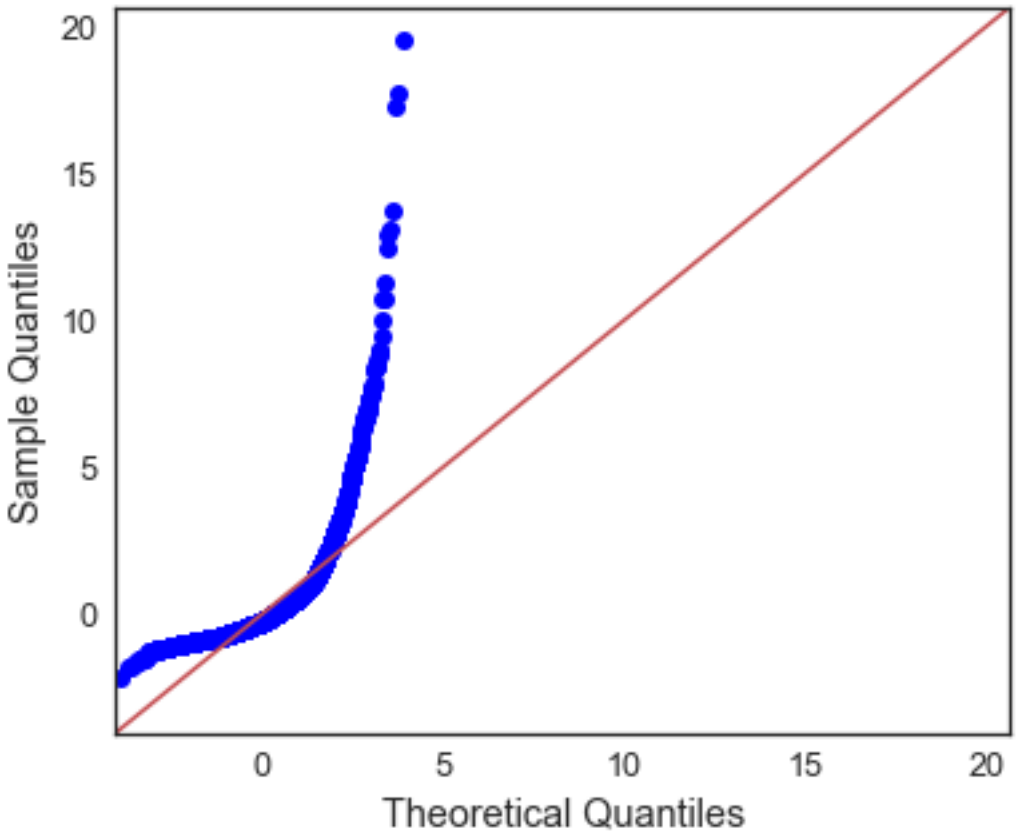
```
formula: price ~ sqft_living15
Regression Analysis and Diagnostics for Price with sqft_living15
Plot 16 of 21
--------------------------------------------------------------------------------
----------
```



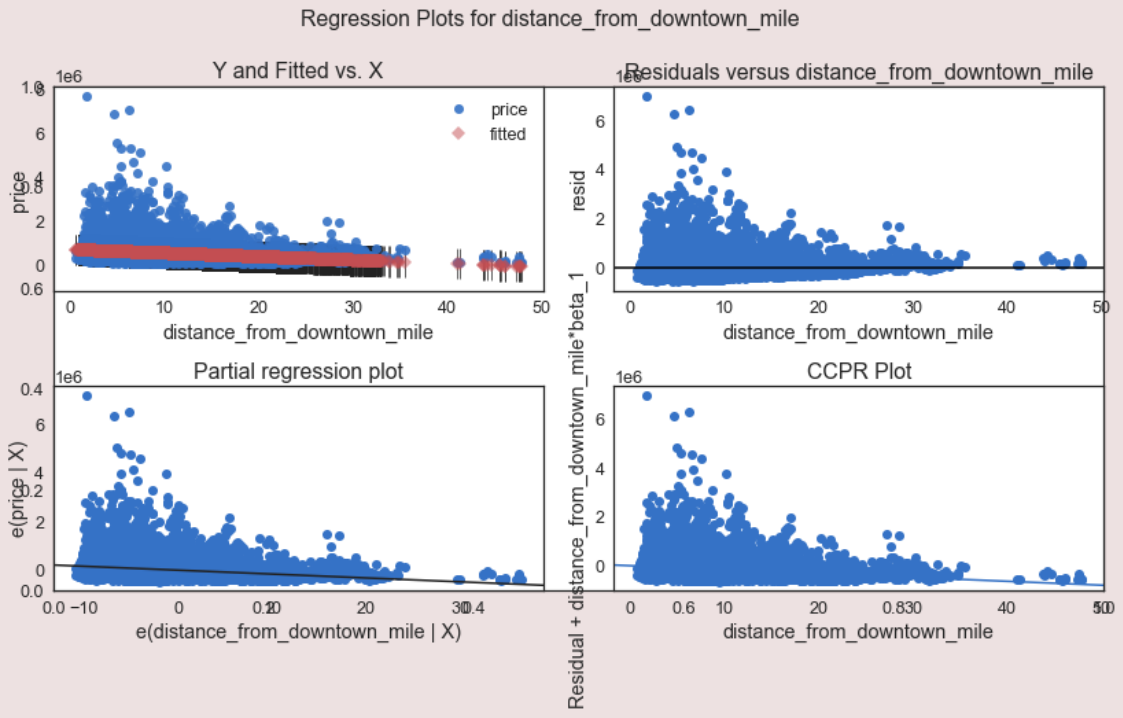


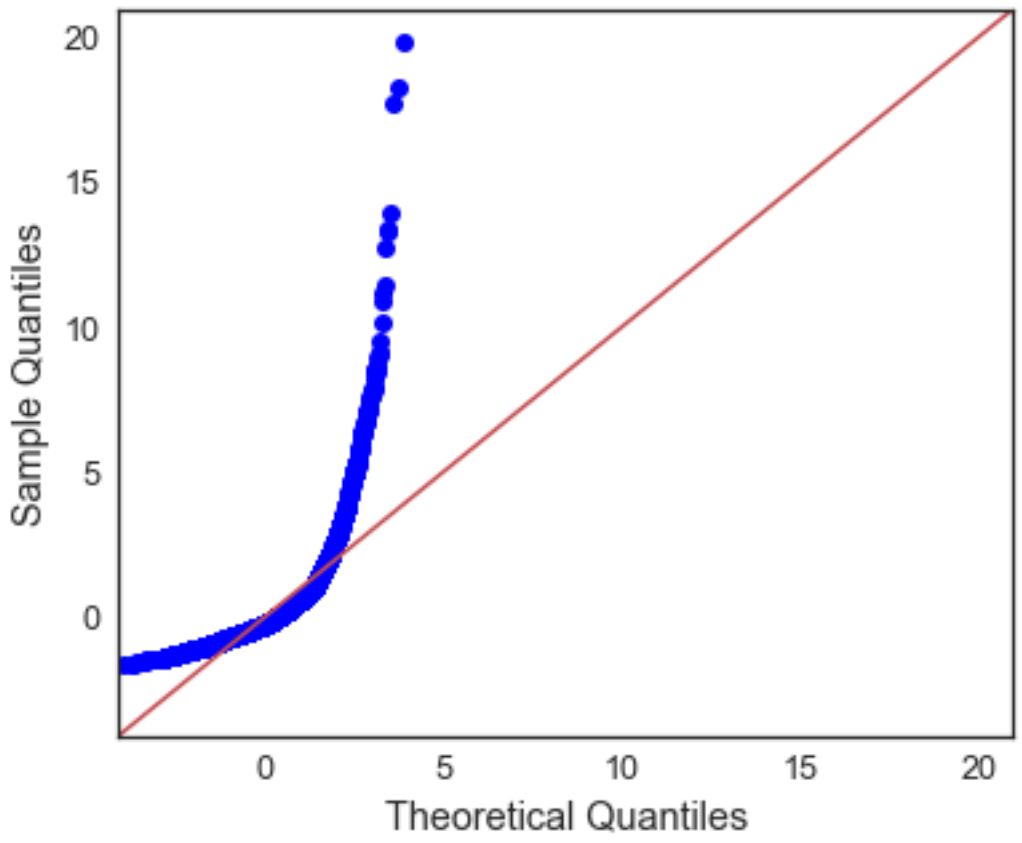

```
formula: price ~ sqft_lot15
Regression Analysis and Diagnostics for Price with sqft_lot15
Plot 17 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for sqft_lot15
Y and Fitted vs. X
price
fitted
sqft_lot15
Residuals versus sqft_lot15
resid
Partial regression plot
e(price | X)
e(sqft_lot15 | X)
CCPR Plot
Residual + sqft_lot15*beta_1

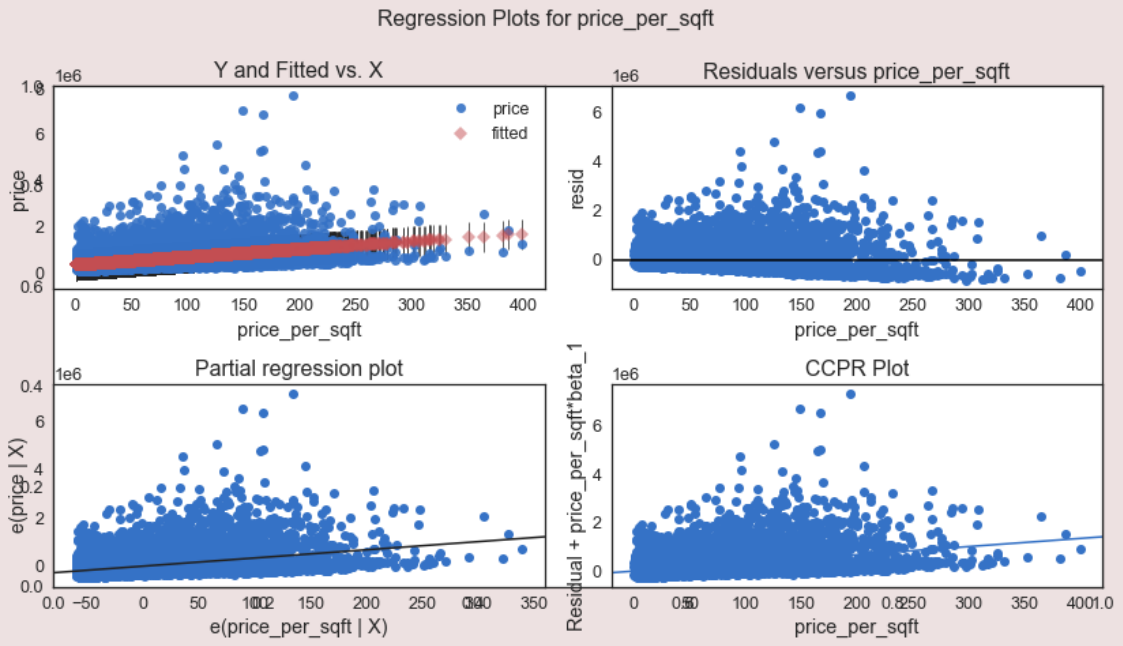

Sample Quantiles
Theoretical Quantiles

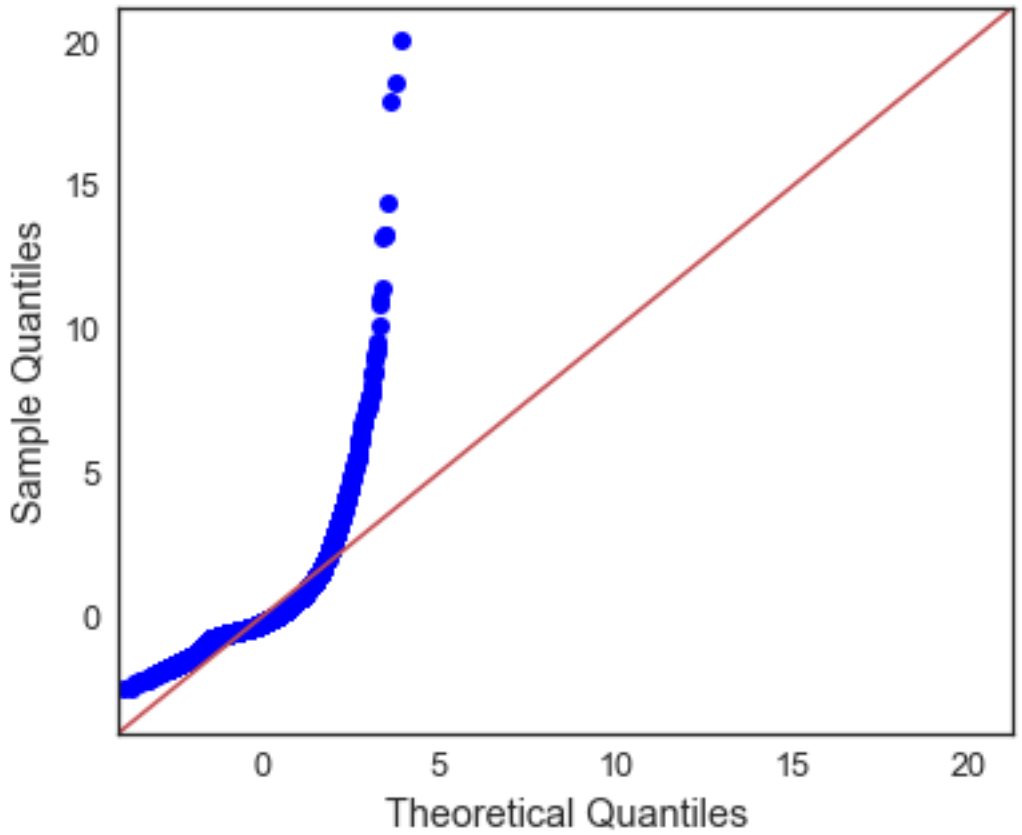
```
formula: price ~ distance_from_downtown_mile
Regression Analysis and Diagnostics for Price with distance_from_downtown_mile
Plot 18 of 21
--------------------------------------------------------------------------------
----------
```



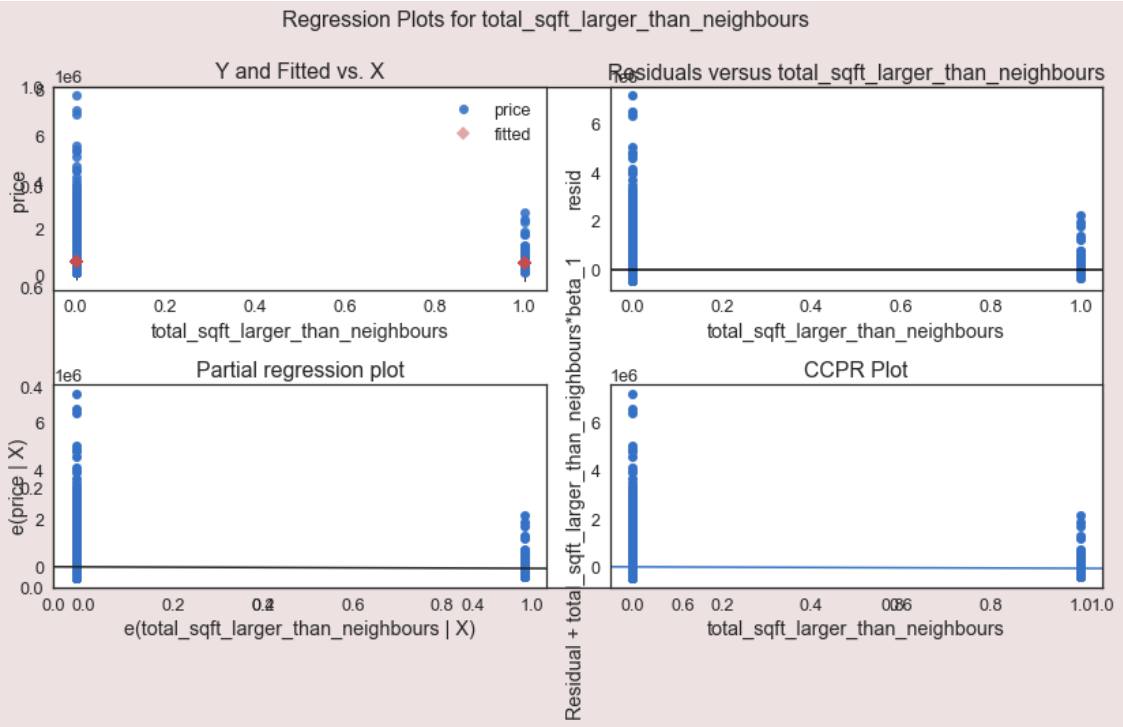


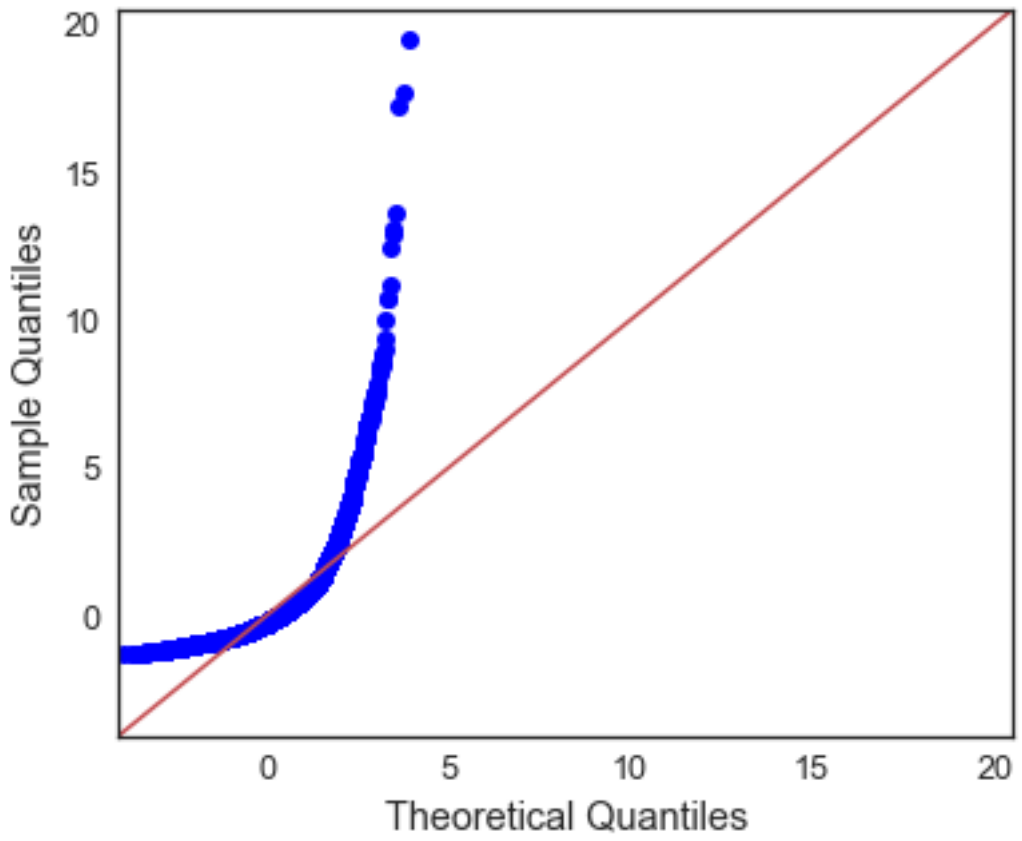

```
formula: price ~ price_per_sqft
Regression Analysis and Diagnostics for Price with price_per_sqft
Plot 19 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for price_per_sqft
Y and Fitted vs. X
price
fitted
price_per_sqft
Residuals versus price_per_sqft
resid
price_per_sqft
Partial regression plot
e(price | X)
e(price_per_sqft | X)
CCPR Plot
Residual + price_per_sqft*beta_1
price_per_sqft

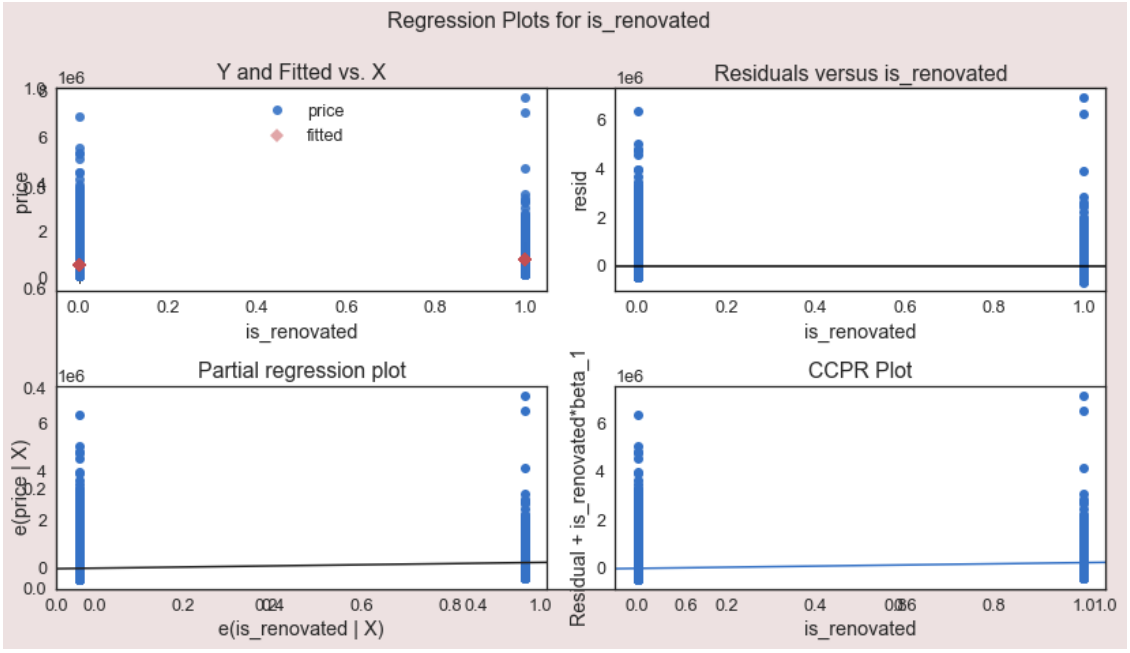

Sample Quantiles
Theoretical Quantiles

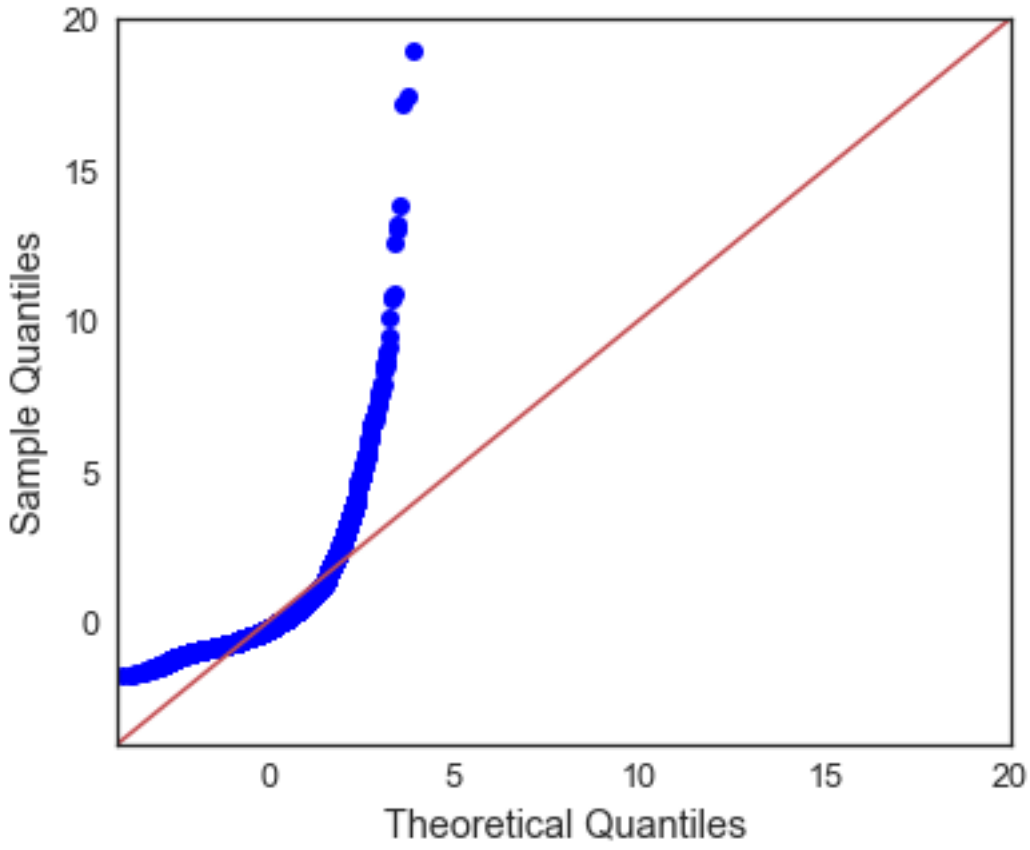
```
formula for regression: price ~ price_per_sqft
Regression Analysis and Diagnostics for Price with
total_sqft_larger_than_neighbours
Plot 20 of 21
--------------------------------------------------------------------------------
----------
```






```
formula for regression: price ~ C(total_sqft_larger_than_neighbours)
Regression Analysis and Diagnostics for Price with is_renovated
Plot 21 of 21
--------------------------------------------------------------------------------
----------
```


Regression Plots for is_renovated
Y and Fitted vs. X
price
fitted
is_renovated
Residuals versus is_renovated
resid
Partial regression plot
e(price | X)
e(is_renovated | X)
CCPR Plot
Residual + is_renovated*beta_1

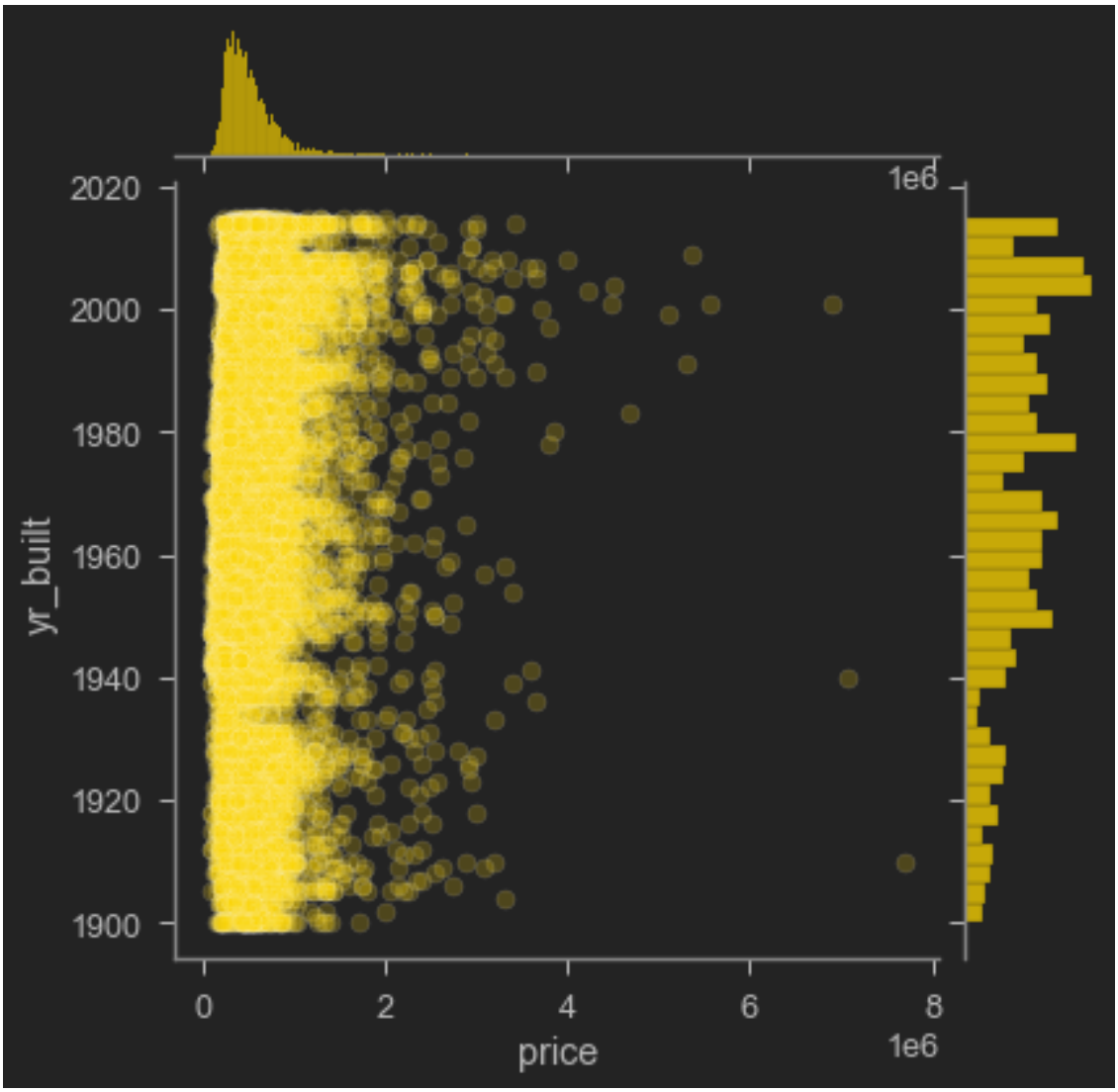

Sample Quantiles
Theoretical Quantiles

### 4.6.5 Paired Feature exploration

**Price VS Year Built**

[71]:
```
sns.jointplot(x='price', y='yr_built', data=df_model, color='gold', alpha=.2)
```

[71]:
```
<seaborn.axisgrid.JointGrid at 0x1c25ae2dd30>
```




[72]:
```
price_vs_year_built()
```

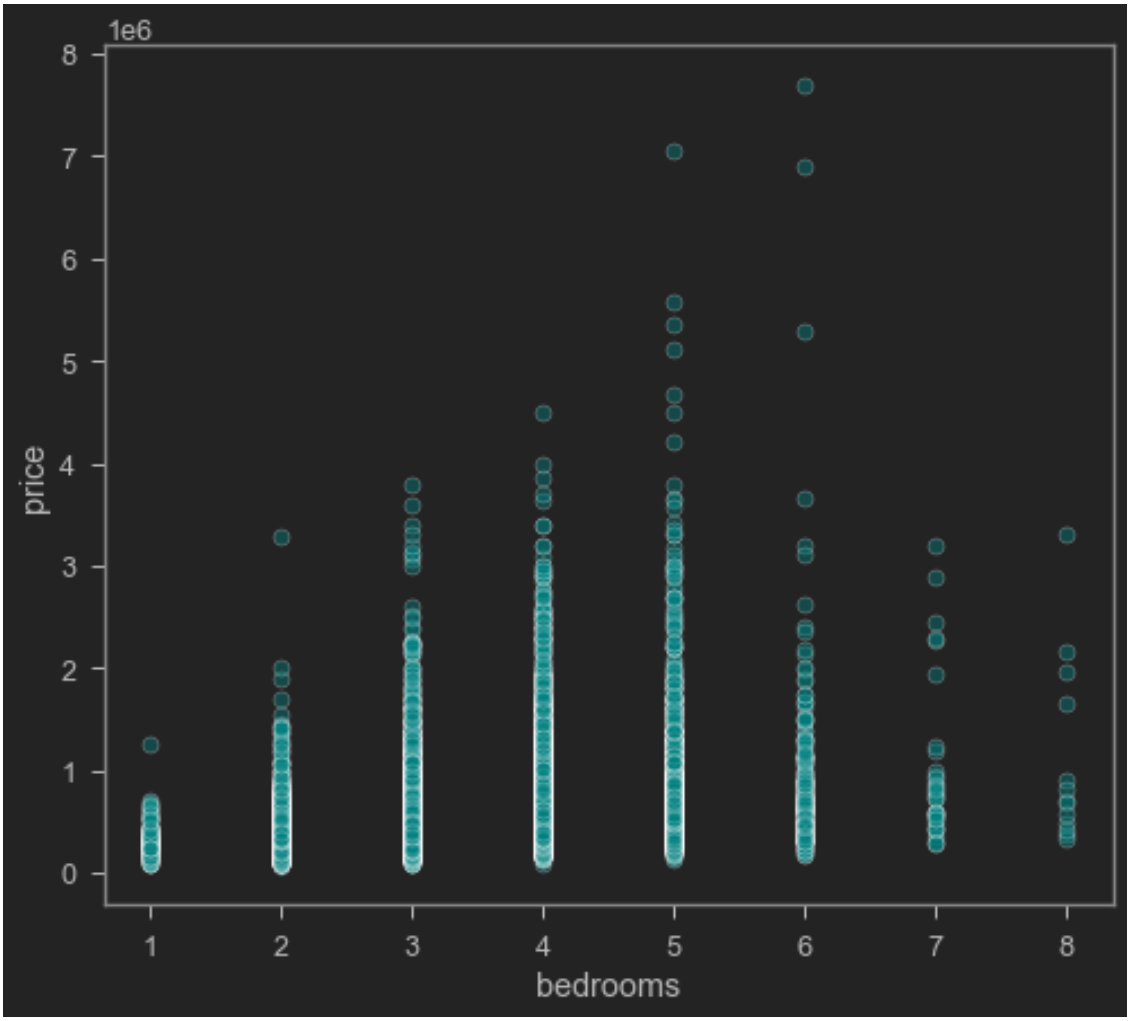
Hoses built in the 40's to 60's have good value. A weird situation for the beginning of 70's, maybe related to a quirk of the dataset or energy crisis of that time, more domain research is needed to comment on that. High sale frequency in the early 00's to a dip in 2008 because of the global financial crisis because of subprime mortgage situation. House prices was in a bubble that created the financial crisis. This data also supports that. This data of house price and sale coincides with all financial crisis. But across the board house price is stable. Have to look into sale date to make

any further comment.

### Price VS Bedroom Count

[73]:
```
sns.scatterplot(data=df_model[df_model.
↪bedrooms<=8],x='bedrooms',y='price',color='teal',alpha=.4)
```

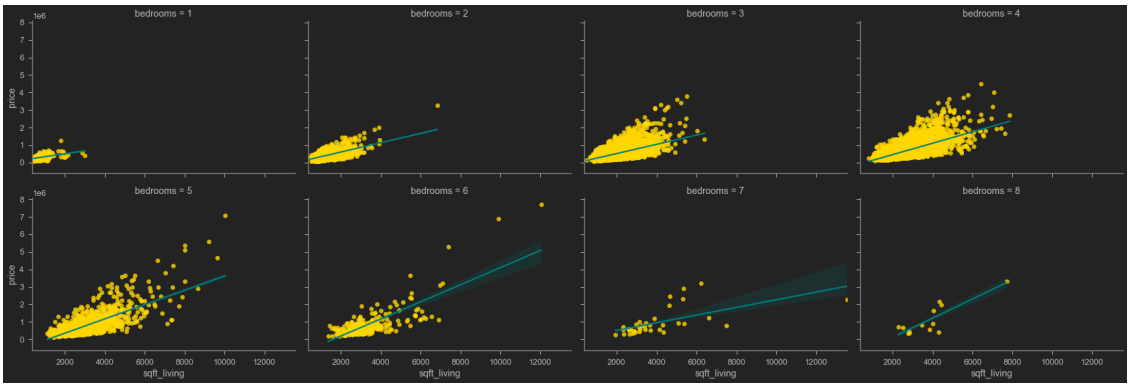
[73]:
```
<AxesSubplot:xlabel='bedrooms', ylabel='price'>
```




[74]:
```
sns.lmplot(y='price',
           x='sqft_living',
           col='bedrooms',
           col_wrap=4,
           data=df_model[df_model.bedrooms <= 8],
           palette='Set2',
           ci=68,
           height=4,
```

```
            aspect=1.5,
            scatter_kws={'color': 'gold'},
            line_kws={'color': 'teal'})  #, color='teal', alpha=.4)
```

[74]:
```
<seaborn.axisgrid.FacetGrid at 0x1c258efe3a0>
```




[75]:
```
price_vs_bedroom_count()
```

Increased bedroom count has a diminishing return after a certain thresh hold, which is 5 after that . Although more house with more bedroom has higher average sale price, this is because they are usually a larger property. I could bin house size and go down that path to pin point that, but not choosing not to for the sake of time constraint.

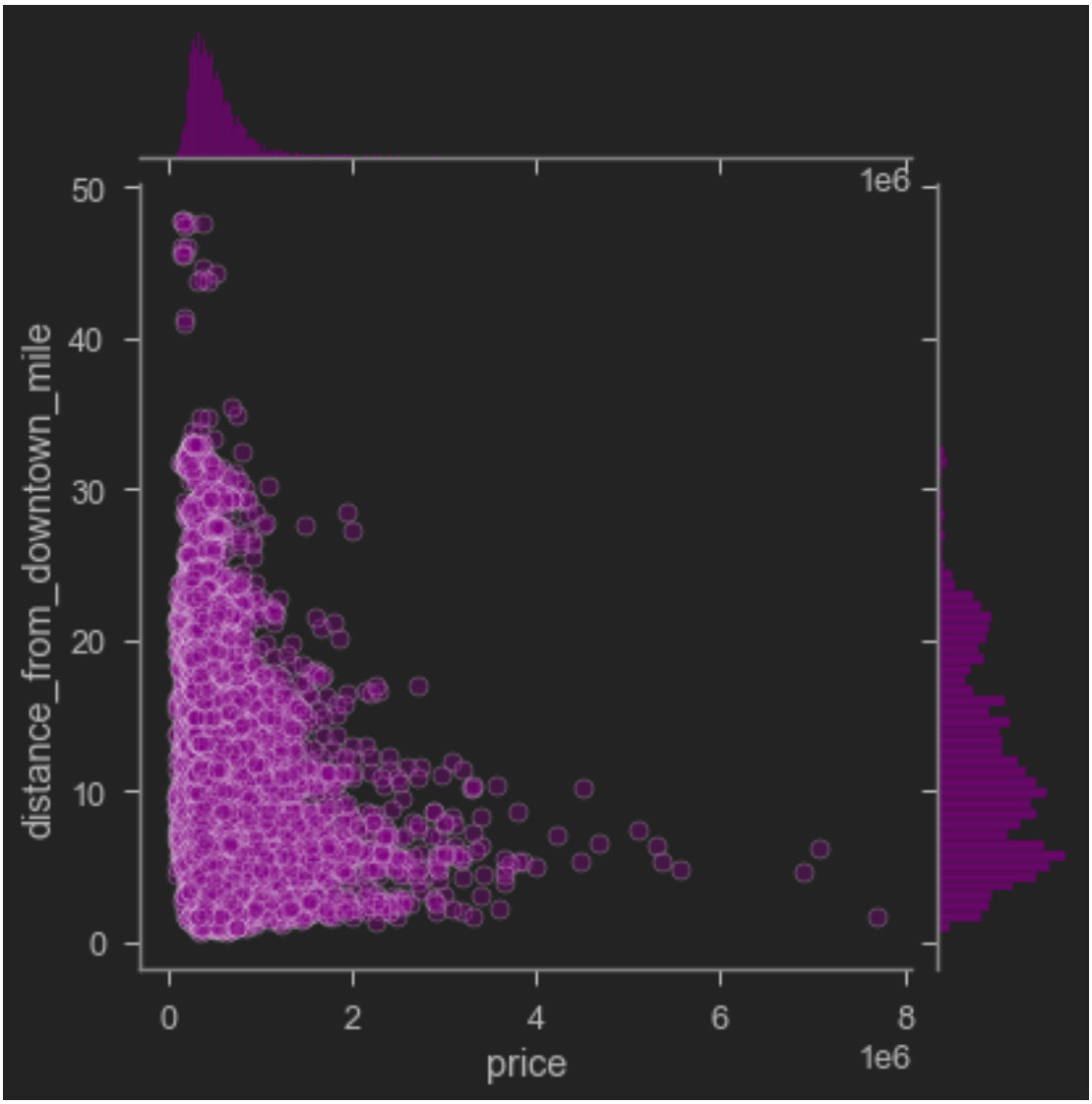
### Price VS Distance from downtown

[76]:
```
df_model.distance_from_downtown_mile
```

[76]:
```
0          7.43
1          7.95
2         10.19
3          6.54
4         13.38
          ...
21414      6.46
21415      6.74
21416      1.74
21417     13.22
21418      1.75
Name: distance_from_downtown_mile, Length: 21419, dtype: float64
```

[77]:
```
sns.jointplot(x='price',
              y='distance_from_downtown_mile',
              data=df_model,
              color='purple',
              alpha=.35)
```

[77]: <seaborn.axisgrid.JointGrid at 0x1c2548e4b20>




This is a no-brainer. But it helped me to spot some outliers. Later confirmed by looking at the map. Making a cap of arbitrary 40 miles cap there guided by looking at the above plot. I found 18 outliers.

[78]:
```
len(df_model[df_model.distance_from_downtown_mile>40])
```

[78]: 18

[79]:
```
df_model[df_model.distance_from_downtown_mile>40]
```

[79]:

| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 2564 | 134000.0 | 2 | … | 0 | 1 |

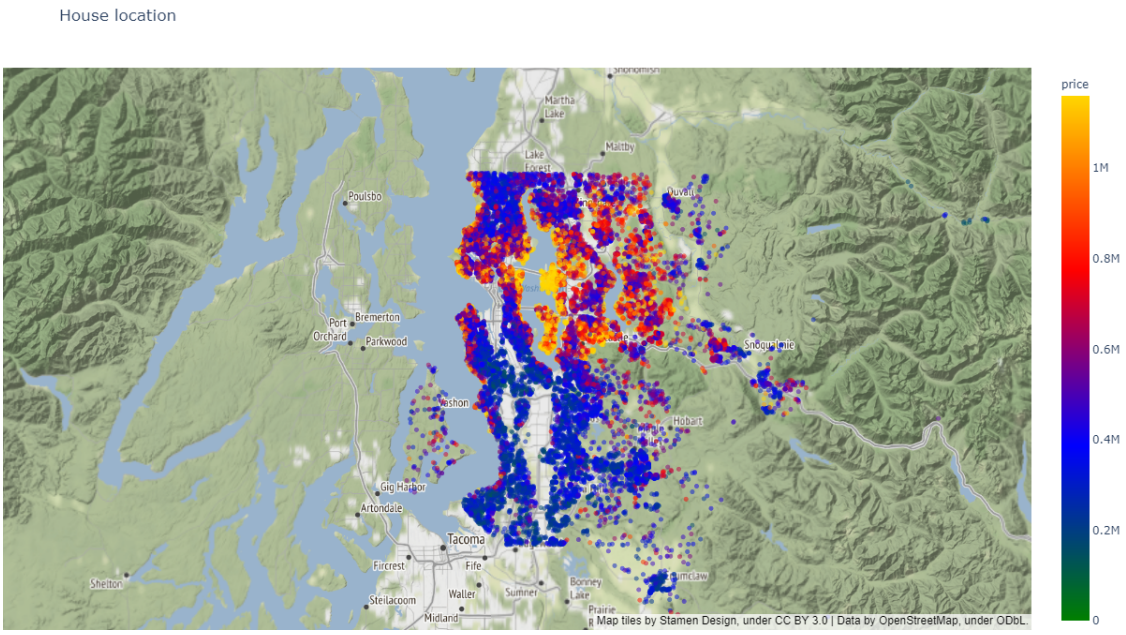
| | | | | | |
|---|---|---|---|---|---|
| 2901 | 167000.0 | 1 | … | 0 | 0 |
| 3266 | 380000.0 | 3 | … | 0 | 0 |
| 4164 | 150000.0 | 3 | … | 0 | 0 |
| 4808 | 525000.0 | 3 | … | 0 | 0 |
| 5814 | 175000.0 | 2 | … | 0 | 0 |
| 6035 | 150000.0 | 3 | … | 1 | 0 |
| 9999 | 200000.0 | 2 | … | 0 | 0 |
| 10794 | 241000.0 | 2 | … | 0 | 0 |
| 12942 | 155000.0 | 2 | … | 0 | 0 |
| 13118 | 375000.0 | 3 | … | 0 | 0 |
| 13856 | 370000.0 | 2 | … | 0 | 0 |
| 14471 | 160000.0 | 3 | … | 0 | 0 |
| 16677 | 170000.0 | 1 | … | 1 | 0 |
| 16776 | 160000.0 | 3 | … | 0 | 0 |
| 19462 | 155000.0 | 3 | … | 0 | 0 |
| 19792 | 415000.0 | 3 | … | 0 | 0 |
| 21193 | 320000.0 | 3 | … | 0 | 0 |

[18 rows x 21 columns]

### 4.6.6 Location

**House Location**

[80]:
```
get_location_static()
```




Most expensive graphs are centered around downtown Seattle.

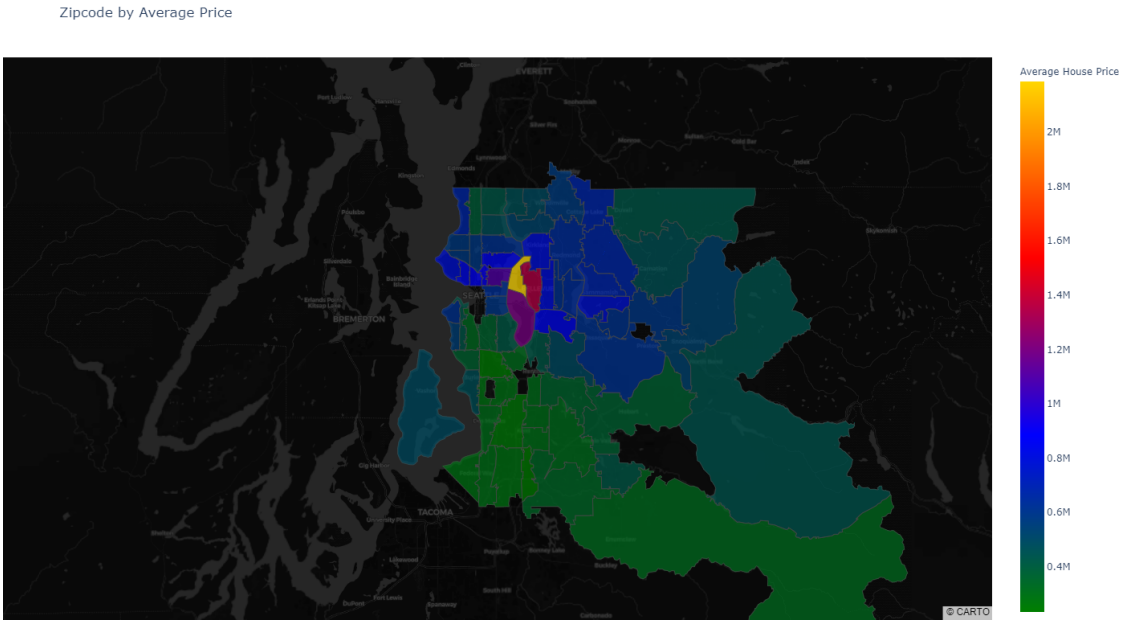
[81]:
```
# # to get a interactive version of this uncomment this and run
# get_location_interactive()
```

### Average Price by Zipcode

[82]:
```
# grouping by zipcode and aggregating by average price
df_mean_price_by_zip = df.groupby(by='zipcode').mean().reset_index()
# top 10 "rich" neighborhoods
df_mean_price_by_zip[['zipcode', 'price'
                      ]].sort_values(by='price',
                                     ascending=False).
 ↪reset_index(drop='index')[:10].style.format("{0:,.0f}")
```

[82]:
```
<pandas.io.formats.style.Styler at 0x1c24df65d00>
```

[83]:
```
average_price_by_zipcode_static()
```




[84]:
```
# # to get a interactive version of this uncomment this and run
# average_price_by_zipcode_interactive()
```

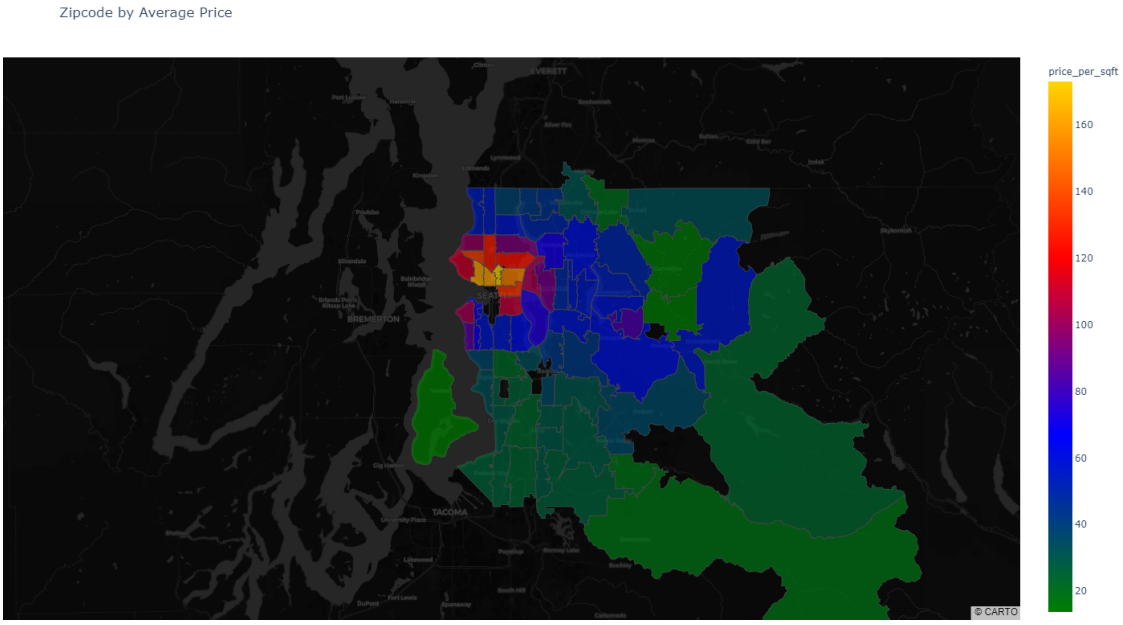
[85]:
```
df_mean_price_per_sqft_by_zip = df.groupby(by='zipcode').mean().reset_index()[[
    'zipcode', 'price_per_sqft'
]].sort_values(by='price_per_sqft', ascending=False).reset_index(drop='index')
df_mean_price_per_sqft_by_zip
```

[85]:

| | zipcode | price_per_sqft |
|---|---|---|
| 0 | 98102 | 173.003942 |

```
1     98119      161.332989
2     98109      160.549817
3     98112      152.387873
4     98107      137.061856
..      …            …
65    98010       18.455051
66    98022       18.301974
67    98014       15.488130
68    98024       14.706456
69    98070       13.748276

[70 rows x 2 columns]
```

[86]:
```
average_price_per_sqft_by_zipcode_static()
```

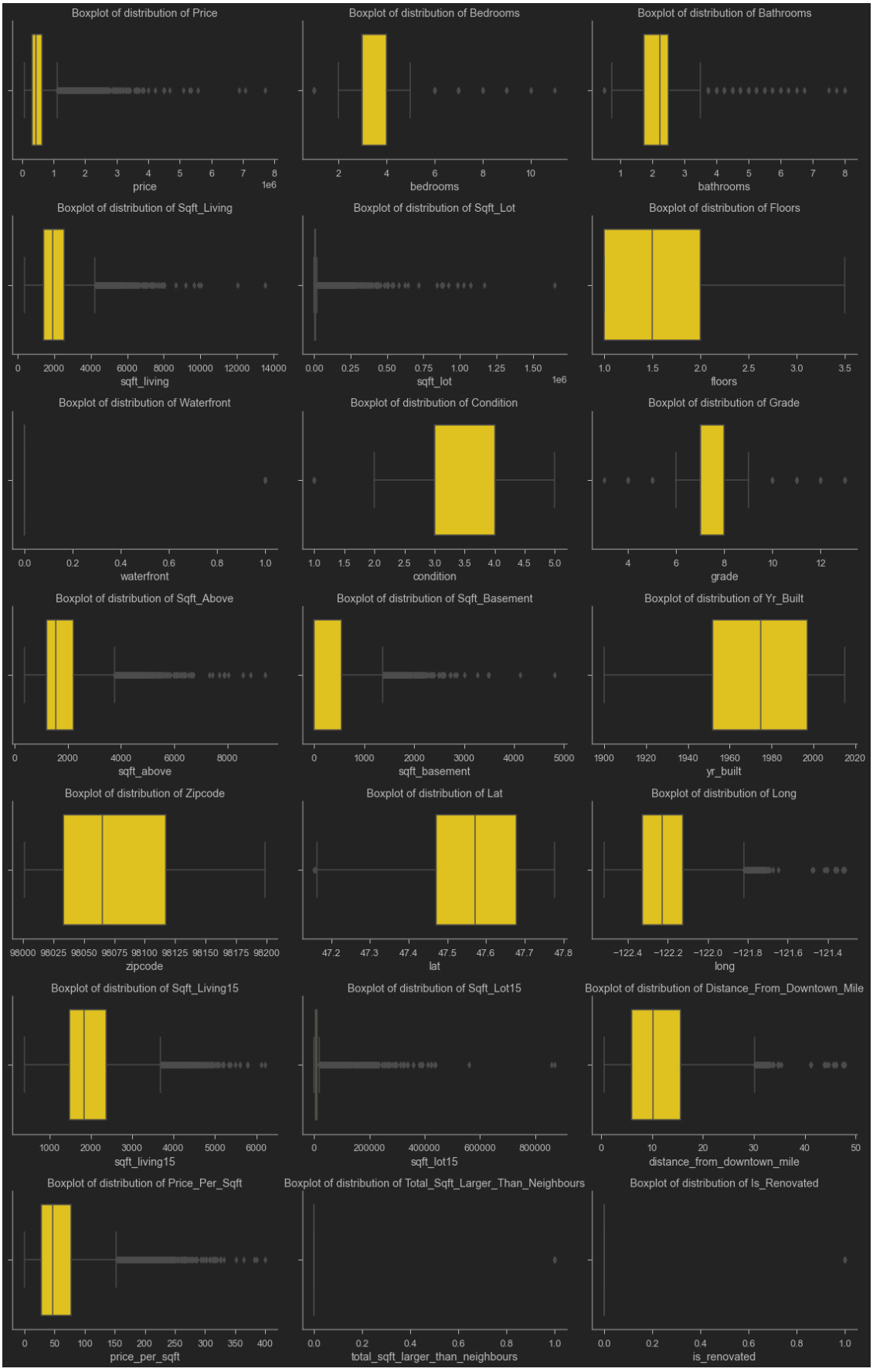



What price to expect when selling.

### 4.7 Remove outliers

[87]:
```
check_outliers_in_df(df=df_model,show_dfs=False)
```

```
------------------------------------------------------------
DataFrame Columns Boxplot
------------------------------------------------------------
```


Boxplot of distribution of Price
price
1e6
Boxplot of distribution of Bedrooms
bedrooms
Boxplot of distribution of Bathrooms
bathrooms
Boxplot of distribution of Sqft_Living
sqft_living
Boxplot of distribution of Sqft_Lot
sqft_lot
1e6
Boxplot of distribution of Floors
floors
Boxplot of distribution of Waterfront
waterfront
Boxplot of distribution of Condition
condition
Boxplot of distribution of Grade
grade
Boxplot of distribution of Sqft_Above
sqft_above
Boxplot of distribution of Sqft_Basement
sqft_basement
Boxplot of distribution of Yr_Built
yr_built
Boxplot of distribution of Zipcode
zipcode
Boxplot of distribution of Lat
lat
Boxplot of distribution of Long
long
Boxplot of distribution of Sqft_Living15
sqft_living15
Boxplot of distribution of Sqft_Lot15
sqft_lot15
Boxplot of distribution of Distance_From_Downtown_Mile
distance_from_downtown_mile
Boxplot of distribution of Price_Per_Sqft
price_per_sqft
Boxplot of distribution of Total_Sqft_Larger_Than_Neighbours
total_sqft_larger_than_neighbours
Boxplot of distribution of Is_Renovated
is_renovated

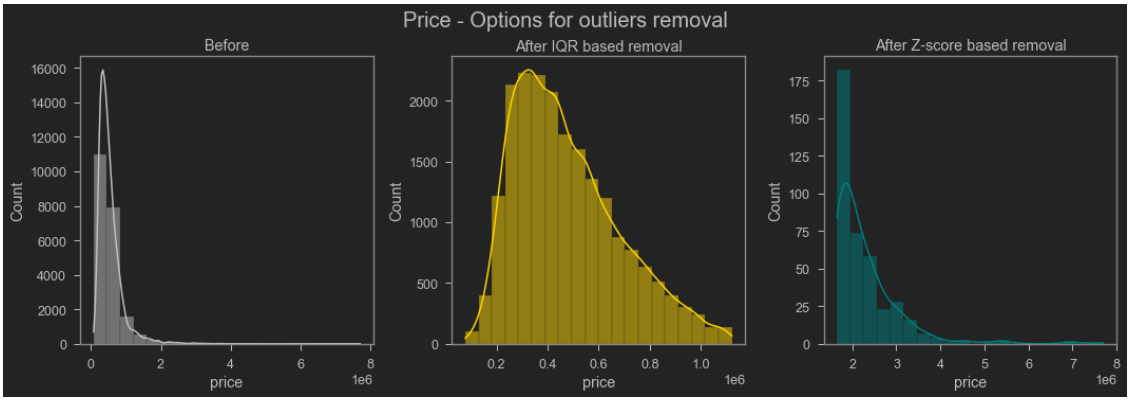
From the chart potential candidate for formula based out lier removal are: - price - bathrooms - bathrooms - sqft_living - sqft_lot - floors - condition - sqft_above - sqft_basement

And for manual outlier removal: - distance_from_downtown_mile

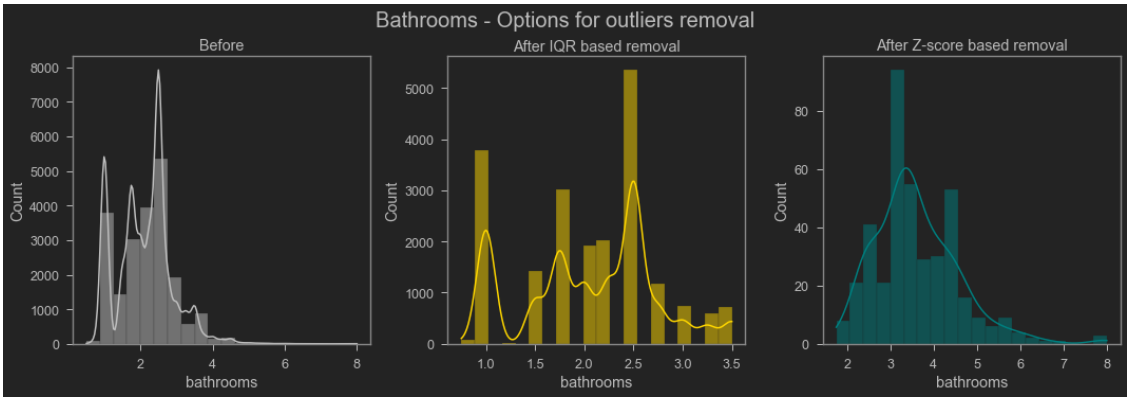
[88]:
```
to_remove_outliers_from = [
    'price', 'bathrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
    'condition', 'sqft_above', 'sqft_basement'
]
```

[89]:
```
for item in to_remove_outliers_from:
    IQR_df = df_model[find_outliers_IQR(df_model[item])]
    IQR_df_ = df_model[~find_outliers_IQR(df_model[item])]
    print(item.title(), "data loss from original dataset using IQR",
          round((len(IQR_df) / len(df_model)) * 100, 2), '%')
    Z_price_price = df_model[find_outliers_Z(df_model['price'])]
    Z_price_price_ = df_model[find_outliers_Z(df_model['price'])]
    print(item.title(), "data loss from original dataset using Z",
          round((len(Z_price_price) / len(df_model)) * 100, 2), '%')
    # left
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    sns.histplot(data=df_model, x=item, color='silver', bins=20, kde=True)
    plt.title('Before')
    # middle
    plt.subplot(1, 3, 2)
    sns.histplot(data=IQR_df_, x=item, color='gold', bins=20, kde=True)
    plt.title('After IQR based removal')
    # right
    plt.subplot(1, 3, 3)
    sns.histplot(data=Z_price_price_, x=item, color='teal', bins=20, kde=True)
    plt.title('After Z-score based removal')
    plt.tight_layout()
    plt.suptitle(f'{item.title()} - Options for outliers removal',
                 va='bottom',
                 fontsize=20)
    print(f'{"_"*160}')
    plt.show()
```

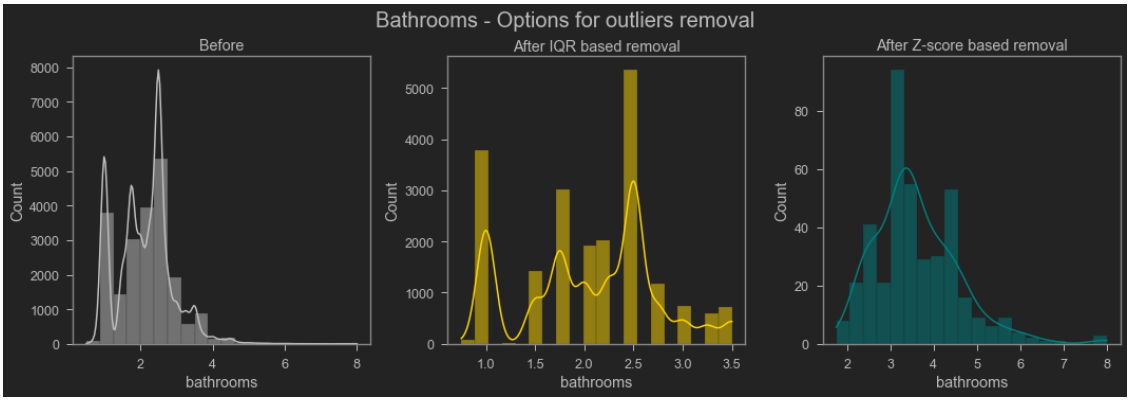
```
Price data loss from original dataset using IQR 5.37 %
Price data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



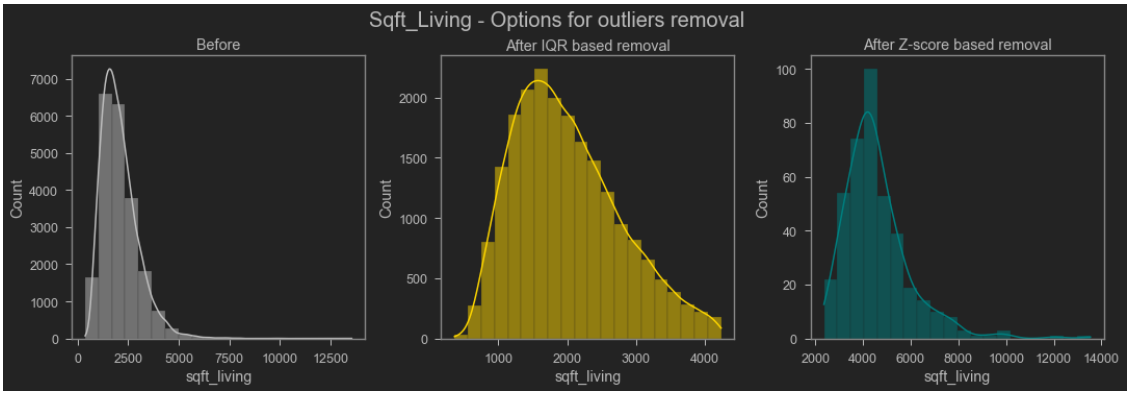

```
Bathrooms data loss from original dataset using IQR 2.61 %
Bathrooms data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



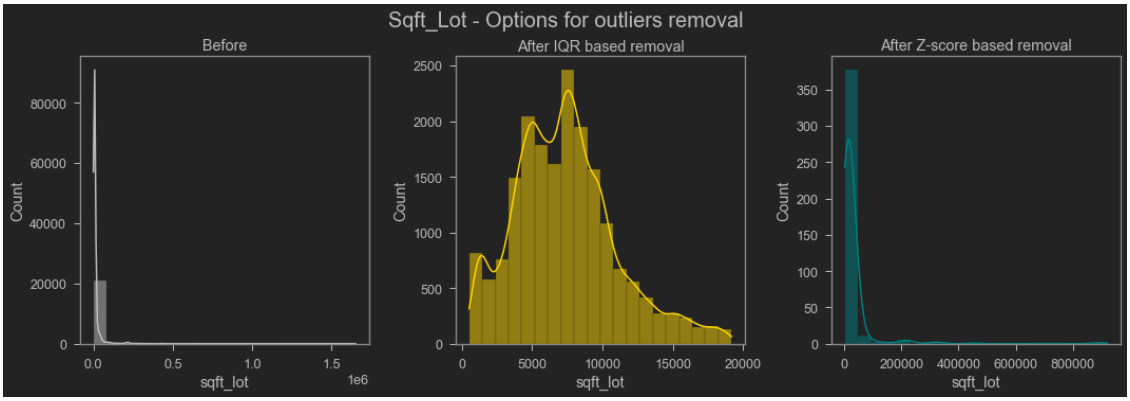

```
Bathrooms data loss from original dataset using IQR 2.61 %
Bathrooms data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



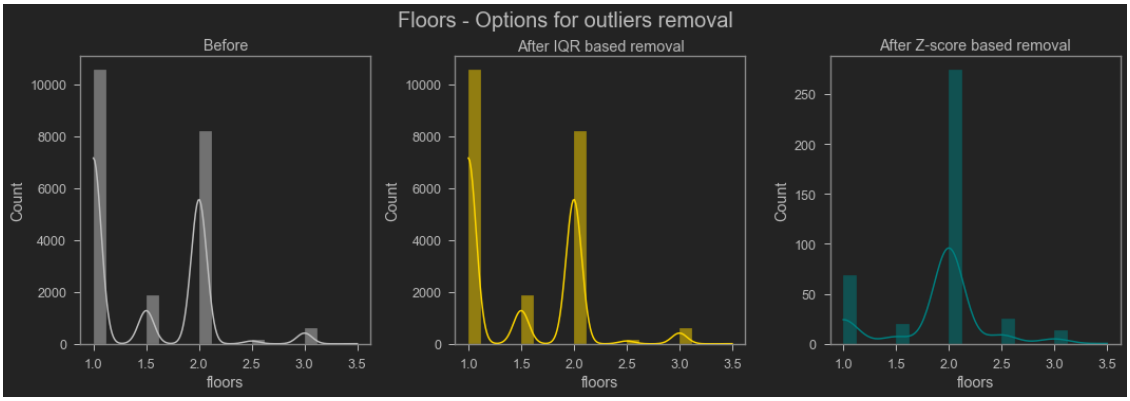

```
Sqft_Living data loss from original dataset using IQR 2.65 %
Sqft_Living data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



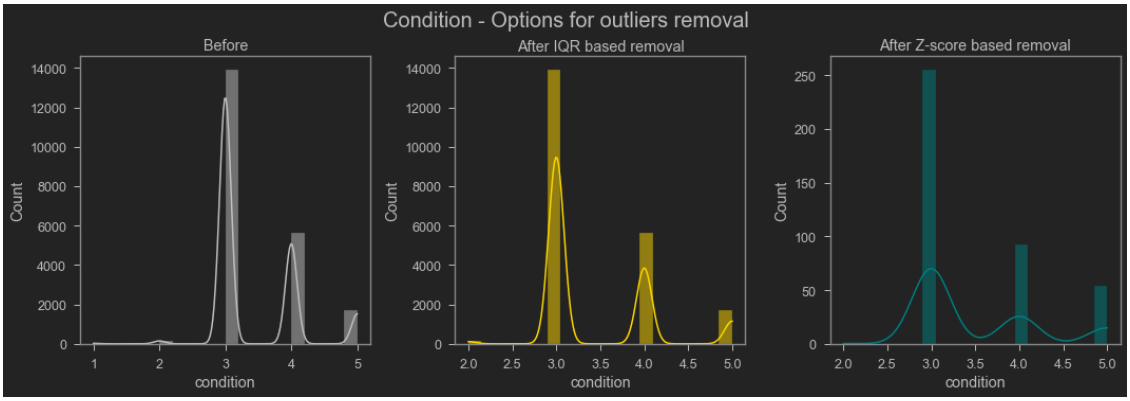

```
Sqft_Lot data loss from original dataset using IQR 11.23 %
Sqft_Lot data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



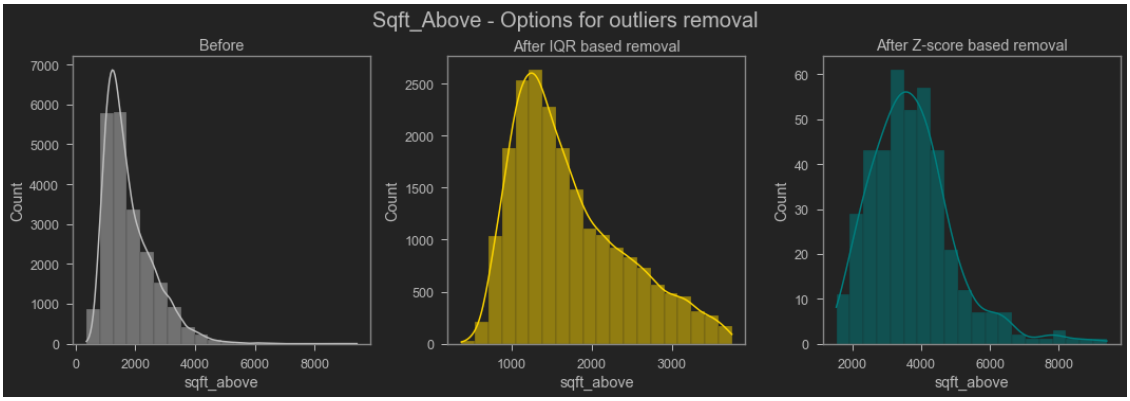

```
Floors data loss from original dataset using IQR 0.0 %
Floors data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```



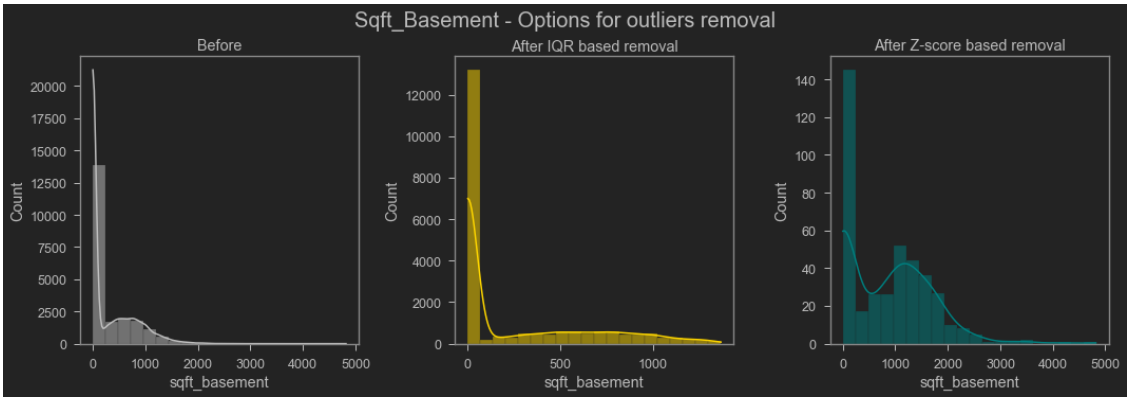

```
Condition data loss from original dataset using IQR 0.13 %
Condition data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```




```
Sqft_Above data loss from original dataset using IQR 2.8 %
Sqft_Above data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```




```
Sqft_Basement data loss from original dataset using IQR 2.6 %
Sqft_Basement data loss from original dataset using Z 1.88 %
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```




Across the board outlier removal by using IQR is the winner. Now removing those outliers. bathroom,sqft-basement can use Z-score. Maybe later.

[90]:
```
## checkpoint
df_model_2 = df_model.copy()
```

[91]:
```
df_model_2.shape
```

[91]: (21419, 21)

[92]:
```
# droping data and generating info
n = 0
while n < len(to_remove_outliers_from):
    print("Dropping Data".capitalize())
    print(f'{"_"*160}')
    for item in to_remove_outliers_from:
        print(item.capitalize())
        dropped_rows= len(df_model_2[find_outliers_IQR(df_model_2[item])])
        original_rows = len(df_model_2)

        df_model_2 = df_model_2[~find_outliers_IQR(df_model_2.price)]
        print(f'   Data dropped: {dropped_rows}')
        print(f'   Data loss : {round((dropped_rows/original_rows)*100,4)}%')

        n = n+1
    print(f'{"_"*160}')
```

```
Dropping data
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Price
   Data dropped: 1151
   Data loss : 5.3737%
Bathrooms
```

```
   Data dropped: 99
   Data loss : 0.4885%
Bathrooms
   Data dropped: 89
   Data loss : 0.4448%
Sqft_living
   Data dropped: 265
   Data loss : 1.3255%
Sqft_lot
   Data dropped: 2168
   Data loss : 10.8438%
Floors
   Data dropped: 0
   Data loss : 0.0%
Condition
   Data dropped: 27
   Data loss : 0.135%
Sqft_above
   Data dropped: 447
   Data loss : 2.2358%
Sqft_basement
   Data dropped: 523
   Data loss : 2.6159%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

[93]:
```
len_new = len(df_model_2)
len_old = len(df_model)
print('New utlier removed dataset length:', len_new)
print('Old dataset length:', len_old)
print(f'Total data loss: {round(abs((np.log(len_new/len_old)*100)),2)}%')
df_model_2
```

```
New utlier removed dataset length: 19993
Old dataset length: 21419
Total data loss: 6.89%
```

[93]:

| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 0 | 221900.0 | 3 | … | 0 | 0 |
| 1 | 538000.0 | 3 | … | 0 | 1 |
| 2 | 180000.0 | 2 | … | 0 | 0 |
| 3 | 604000.0 | 4 | … | 0 | 0 |
| 4 | 510000.0 | 3 | … | 0 | 0 |
| … | … | … | … | … | … |
| 21414 | 360000.0 | 3 | … | 0 | 0 |
| 21415 | 400000.0 | 4 | … | 0 | 0 |
| 21416 | 402101.0 | 2 | … | 0 | 0 |
| 21417 | 400000.0 | 3 | … | 0 | 0 |

```
21418  325000.0         2  …                                      0              0

[19993 rows x 21 columns]
```

There are few extremely big values in distance_from_downtown feature, now removing this. Thresh hold is more than 40 miles will be dropped.

[94]:
```
print('Outliers:
 ↪',len(df_model_2[(df_model_2['distance_from_downtown_mile']>40)]))
df_model_2[(df_model_2['distance_from_downtown_mile']>40)]
```

```
Outliers: 18
```

[94]:

| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 2564 | 134000.0 | 2 | … | 0 | 1 |
| 2901 | 167000.0 | 1 | … | 0 | 0 |
| 3266 | 380000.0 | 3 | … | 0 | 0 |
| 4164 | 150000.0 | 3 | … | 0 | 0 |
| 4808 | 525000.0 | 3 | … | 0 | 0 |
| 5814 | 175000.0 | 2 | … | 0 | 0 |
| 6035 | 150000.0 | 3 | … | 1 | 0 |
| 9999 | 200000.0 | 2 | … | 0 | 0 |
| 10794 | 241000.0 | 2 | … | 0 | 0 |
| 12942 | 155000.0 | 2 | … | 0 | 0 |
| 13118 | 375000.0 | 3 | … | 0 | 0 |
| 13856 | 370000.0 | 2 | … | 0 | 0 |
| 14471 | 160000.0 | 3 | … | 0 | 0 |
| 16677 | 170000.0 | 1 | … | 1 | 0 |
| 16776 | 160000.0 | 3 | … | 0 | 0 |
| 19462 | 155000.0 | 3 | … | 0 | 0 |
| 19792 | 415000.0 | 3 | … | 0 | 0 |
| 21193 | 320000.0 | 3 | … | 0 | 0 |

[18 rows x 21 columns]

[95]:
```
df_model_2 = df_model_2[~(df_model_2['distance_from_downtown_mile']>40)]
df_model_2
```

[95]:

| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 0 | 221900.0 | 3 | … | 0 | 0 |
| 1 | 538000.0 | 3 | … | 0 | 1 |
| 2 | 180000.0 | 2 | … | 0 | 0 |
| 3 | 604000.0 | 4 | … | 0 | 0 |
| 4 | 510000.0 | 3 | … | 0 | 0 |
| … | … | … | … | … | … |
| 21414 | 360000.0 | 3 | … | 0 | 0 |
| 21415 | 400000.0 | 4 | … | 0 | 0 |
| 21416 | 402101.0 | 2 | … | 0 | 0 |

```
21417  400000.0         3  …                                     0             0
21418  325000.0         2  …                                     0             0

[19975 rows x 21 columns]
```

[96]:
```
df_model_2
```

[96]:
```
           price  bedrooms  …  total_sqft_larger_than_neighbours  is_renovated
0       221900.0         3  …                                  0             0
1       538000.0         3  …                                  0             1
2       180000.0         2  …                                  0             0
3       604000.0         4  …                                  0             0
4       510000.0         3  …                                  0             0
…            …       …  …                                  …             …
21414   360000.0         3  …                                  0             0
21415   400000.0         4  …                                  0             0
21416   402101.0         2  …                                  0             0
21417   400000.0         3  …                                  0             0
21418   325000.0         2  …                                  0             0

[19975 rows x 21 columns]
```

[97]:
```
# after removing
# check_outliers_in_df(df=df_model_2,show_dfs=False)
```

[98]:
```
df_model_2.to_csv(r'./Data/kc_house_data_df_outlier_removed.csv', index=False)
```

[99]:
```
# checkpint: to keep a unscaled dataset in hand
df_model_3 = df_model_2.copy()
```

### 4.8 Ordinary Least Squares

[100]:
```
def OLS_sm(df,
           dependant_var='price',
           numeric_features=[],
           categorical_features=[],
           verbose=False,
           show_summary=True,
           show_plots=True,
           target_is_dollar=True):
    """
    ### Uses formula based statsmodels regression for OLS. ###

    Displays a statsmodels.iolib.summary.Summary object containing summary of␣
↪OLS analysis.
    Returns a statsmodels.regression.linear_model.RegressionResultsWrapper␣
↪which can be used to access other options available.
```

```
    Parameters:
    ===========
        df = pandas.DataFrame; no default.
                Input dataset to use for OLS.
        dependant_var = str; default: 'price'.
                Dependent variable.
        numeric_features = list; default = [].
                Identify numeric features.
        categorical_features = list; default = [].
                Identify categorical features.
        verbose = boolean; default: False.
                Shows some formula used and drop information.
                    `True` shows information.
                    `False` does not show information.
        show_summary = boolean; default: False.
                Shows summary report.
                    `True` shows information.
                    `False` does not show information.
        show_plots = boolean; default: True.
                Shows summary and Homoscedasticity information.
                    `True` shows information.
                    `False` does not show information.
        target_is_dollar = boolean; default: True.
                Modify chart axis label.
                    `True` shows information.
                    `False` does not show information.
    Under-The-Hood:
    =============
    --{Major Steps}--

        ## Regression
        cate = ' + '.join([f'C({x})' for x in categorical_features])
        nume = ' + '.join([f'{x}' for x in numeric_features])
        formula = f'{dependant_var} ~ {nume} + {cate}'

        ## plots
        # plot on the left
        sm.qqplot(multiple_regression.resid,
                  dist=stats.norm,
                  line='45',
                  fit=True,
                  ax=ax1)
        # plot on the right
        ax2.scatter(x=multiple_regression.fittedvalues,
                    y=multiple_regression.resid,
                    s=4,
```

```
                    color='gold')

    Note:
    =====
        Make sure that every column in the DataFrame has the correct dtype.
        Numeric values stored as str (i.e, object) will make stats model assume␣
↪that those are categorical variable.
        If Erros, check df to see if the passed feature is available in the␣
↪DataFrame.

    Issues:
    =======
        - Output control is not clear.

    Changelog:
    ==========
        - fixed `resid`, was using `resid_pearson`.

    -- ver: 1.2 --
    """
    cate = ' + '.join([f'C({x})' for x in categorical_features])
    nume = ' + '.join([f'{x}' for x in numeric_features])
    if len(cate)==0:
        formula = f'{dependant_var} ~ {nume}'
    else:
        formula = f'{dependant_var} ~ {nume} + {cate}'
    print('Formula for the OLS model: ', formula)
    # OLS regressor
    multiple_regression = smf.ols(formula=formula, data=df).fit()

    if verbose:
        show_summary = True
        show_plots = True

    if show_summary:
        display(multiple_regression.summary())
    if show_plots:
        # plotting
        # plot 1
        fig, (ax1,
              ax2) = plt.subplots(ncols=2,
                                  figsize=(10, 5),
                                  gridspec_kw={'width_ratios': [0.6, 0.4]})
        sm.qqplot(multiple_regression.resid,
                  dist=stats.norm,
                  line='45',
                  fit=True,
```

```
                ax=ax1)
        ax1.set_title('Q-Q Plot', fontdict={"size": 15})
        # plot 2
        # uses The predicted values for the original (unwhitened) design.
        ax2.scatter(x=multiple_regression.fittedvalues,
                    y=multiple_regression.resid,
                    s=4,
                    color='gold')
        if target_is_dollar:
            ax2.yaxis.set_major_formatter(format_number)
        ax2.set(xlabel='Predicted', ylabel='Residuals')
        ax2.axhline(y=0, c='r', lw=4, ls='--')
        ax2.set_title('Predicted VS Residuals', fontdict={"size": 15})
        plt.suptitle('Visual Check of Residuals for Homoscedasticity',
                     ha='center',
                     va='bottom',
                     fontdict={"size": 25})
        plt.tight_layout()
    if verbose == False and show_summary == False and show_plots == True:
        print('r_sq:', round(multiple_regression.rsquared, 4))
    return multiple_regression
```

Lets take a quick look at a regression.

With outlier

[101]:
```
x1 = OLS_sm(df=df_model,
              numeric_features=[
                  'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
                  'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long',
                  'sqft_living15', 'sqft_lot15', 'distance_from_downtown_mile',
                  'price_per_sqft'
              ],
              dependant_var='price',
              categorical_features=[
                  'waterfront', 'condition', 'grade', 'is_renovated',
                  'zipcode', 'total_sqft_larger_than_neighbours'
              ],
              verbose=False)
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_above + sqft_basement + yr_built + lat + long +
sqft_living15 + sqft_lot15 + distance_from_downtown_mile + price_per_sqft +
C(waterfront) + C(condition) + C(grade) + C(is_renovated) + C(zipcode) +
C(total_sqft_larger_than_neighbours)

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
```

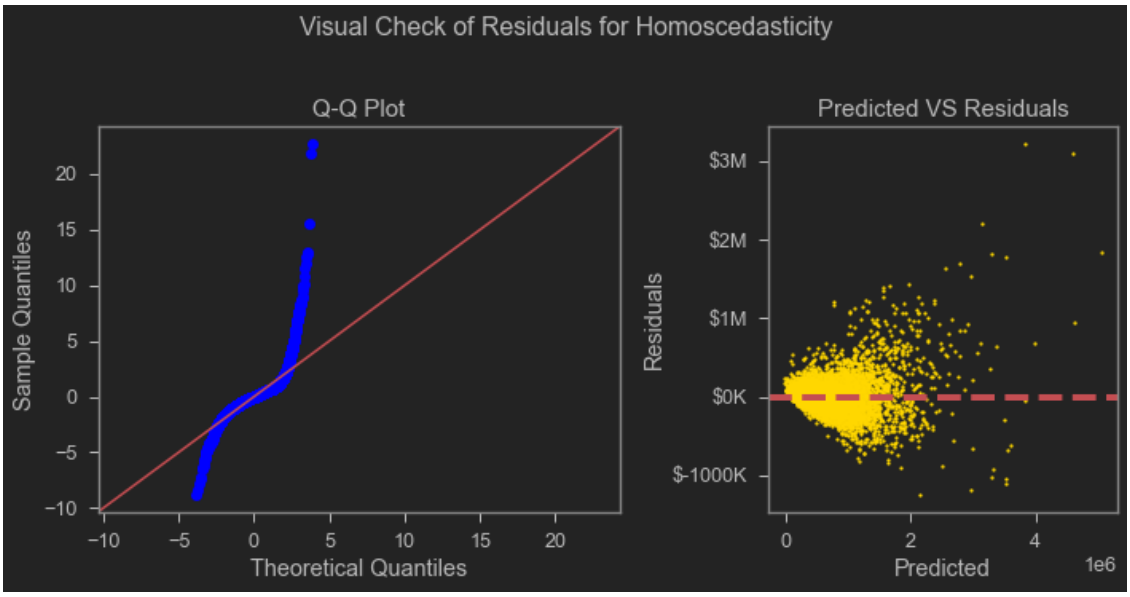
```
==============================================================================
Dep. Variable:                  price   R-squared:                       0.852
Model:                            OLS   Adj. R-squared:                  0.851
Method:                 Least Squares   F-statistic:                     1223.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:41:45   Log-Likelihood:            -2.8446e+05
No. Observations:               21419   AIC:                         5.691e+05
Df Residuals:                   21318   BIC:                         5.699e+05
Df Model:                         100
Covariance Type:            nonrobust
==============================================================================================
                                               coef    std err          t      ␣
 ↪P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Intercept                                   1.127e+07   7.74e+06      1.456      ␣
 ↪0.145   -3.91e+06    2.65e+07
C(waterfront)[T.1]                          7.249e+05   1.24e+04     58.616      ␣
 ↪0.000    7.01e+05    7.49e+05
C(condition)[T.2]                           5.079e+04   2.92e+04      1.738      ␣
 ↪0.082   -6499.167    1.08e+05
C(condition)[T.3]                           5.362e+04   2.72e+04      1.974      ␣
 ↪0.048     373.573    1.07e+05
C(condition)[T.4]                           7.731e+04   2.72e+04      2.844      ␣
 ↪0.004     2.4e+04    1.31e+05
C(condition)[T.5]                           1.037e+05   2.74e+04      3.790      ␣
 ↪0.000    5.01e+04    1.57e+05
C(grade)[T.4]                              -7.859e+04   1.45e+05     -0.542      ␣
 ↪0.588   -3.63e+05    2.06e+05
C(grade)[T.5]                              -1.269e+05   1.43e+05     -0.889      ␣
 ↪0.374   -4.07e+05    1.53e+05
C(grade)[T.6]                              -1.265e+05   1.42e+05     -0.888      ␣
 ↪0.374   -4.06e+05    1.53e+05
C(grade)[T.7]                              -1.286e+05   1.42e+05     -0.903      ␣
 ↪0.367   -4.08e+05    1.51e+05
C(grade)[T.8]                              -1.179e+05   1.43e+05     -0.827      ␣
 ↪0.408   -3.97e+05    1.61e+05
C(grade)[T.9]                               -5.76e+04   1.43e+05     -0.404      ␣
 ↪0.686   -3.37e+05    2.22e+05
C(grade)[T.10]                              5.773e+04   1.43e+05      0.405      ␣
 ↪0.686   -2.22e+05    3.37e+05
C(grade)[T.11]                              2.338e+05   1.43e+05      1.636      ␣
 ↪0.102   -4.62e+04    5.14e+05
C(grade)[T.12]                              6.467e+05   1.44e+05      4.502      ␣
 ↪0.000    3.65e+05    9.28e+05
C(grade)[T.13]                              1.682e+06   1.49e+05     11.323      ␣
 ↪0.000    1.39e+06    1.97e+06
```

```
C(is_renovated)[T.1]                    4.426e+04   5712.285      7.748      ␣
 ↪0.000     3.31e+04    5.55e+04
C(zipcode)[T.98002]                    -1.887e+04    1.3e+04     -1.455      ␣
 ↪0.146    -4.43e+04    6548.366
C(zipcode)[T.98003]                    -1.257e+04   1.16e+04     -1.088      ␣
 ↪0.277    -3.52e+04    1.01e+04
C(zipcode)[T.98004]                     4.921e+05   2.41e+04     20.409      ␣
 ↪0.000     4.45e+05    5.39e+05
C(zipcode)[T.98005]                     1.299e+05   2.52e+04      5.153      ␣
 ↪0.000     8.05e+04    1.79e+05
C(zipcode)[T.98006]                     9.943e+04   2.18e+04      4.553      ␣
 ↪0.000     5.66e+04    1.42e+05
C(zipcode)[T.98007]                     7.988e+04   2.58e+04      3.095      ␣
 ↪0.002     2.93e+04     1.3e+05
C(zipcode)[T.98008]                     9.921e+04   2.46e+04      4.034      ␣
 ↪0.000      5.1e+04    1.47e+05
C(zipcode)[T.98010]                     3.731e+04   2.04e+04      1.827      ␣
 ↪0.068    -2726.394    7.74e+04
C(zipcode)[T.98011]                     1.354e+04   2.87e+04      0.472      ␣
 ↪0.637    -4.27e+04    6.97e+04
C(zipcode)[T.98014]                     2.911e+04    3.2e+04      0.910      ␣
 ↪0.363    -3.36e+04    9.18e+04
C(zipcode)[T.98019]                     1.694e+04   3.12e+04      0.542      ␣
 ↪0.588    -4.43e+04    7.82e+04
C(zipcode)[T.98022]                     1.379e+04   1.71e+04      0.809      ␣
 ↪0.419    -1.96e+04    4.72e+04
C(zipcode)[T.98023]                    -3.072e+04   1.11e+04     -2.760      ␣
 ↪0.006    -5.25e+04   -8905.613
C(zipcode)[T.98024]                     7.164e+04   2.92e+04      2.456      ␣
 ↪0.014     1.45e+04    1.29e+05
C(zipcode)[T.98027]                     2.128e+04   2.16e+04      0.984      ␣
 ↪0.325    -2.11e+04    6.36e+04
C(zipcode)[T.98028]                     1.978e+04   2.78e+04      0.712      ␣
 ↪0.476    -3.47e+04    7.42e+04
C(zipcode)[T.98029]                     5.437e+04   2.39e+04      2.271      ␣
 ↪0.023     7453.207    1.01e+05
C(zipcode)[T.98030]                    -2.574e+04   1.36e+04     -1.886      ␣
 ↪0.059    -5.25e+04    1010.878
C(zipcode)[T.98031]                    -3.052e+04   1.48e+04     -2.058      ␣
 ↪0.040    -5.96e+04   -1449.852
C(zipcode)[T.98032]                    -3.214e+04   1.56e+04     -2.055      ␣
 ↪0.040    -6.28e+04   -1489.474
C(zipcode)[T.98033]                      1.71e+05   2.54e+04      6.735      ␣
 ↪0.000     1.21e+05    2.21e+05
C(zipcode)[T.98034]                     6.187e+04   2.62e+04      2.364      ␣
 ↪0.018     1.06e+04    1.13e+05
```

```
C(zipcode)[T.98038]                  -2658.2993   1.58e+04     -0.168  ␣
 ↪0.867   -3.37e+04    2.84e+04
C(zipcode)[T.98039]                   9.28e+05   3.08e+04     30.086  ␣
 ↪0.000    8.68e+05    9.88e+05
C(zipcode)[T.98040]                   3.096e+05   2.22e+04     13.932  ␣
 ↪0.000    2.66e+05    3.53e+05
C(zipcode)[T.98042]                  -2.493e+04   1.36e+04     -1.840  ␣
 ↪0.066   -5.15e+04    1631.556
C(zipcode)[T.98045]                   6.929e+04   2.75e+04      2.521  ␣
 ↪0.012    1.54e+04    1.23e+05
C(zipcode)[T.98052]                   7.832e+04   2.57e+04      3.043  ␣
 ↪0.002    2.79e+04    1.29e+05
C(zipcode)[T.98053]                   7.056e+04   2.72e+04      2.595  ␣
 ↪0.009    1.73e+04    1.24e+05
C(zipcode)[T.98055]                  -4.272e+04   1.73e+04     -2.472  ␣
 ↪0.013   -7.66e+04   -8852.392
C(zipcode)[T.98056]                  -1.564e+04   1.95e+04     -0.803  ␣
 ↪0.422   -5.38e+04    2.25e+04
C(zipcode)[T.98058]                  -3.857e+04   1.66e+04     -2.324  ␣
 ↪0.020   -7.11e+04   -6033.237
C(zipcode)[T.98059]                  -2.291e+04    1.9e+04     -1.207  ␣
 ↪0.227   -6.01e+04    1.43e+04
C(zipcode)[T.98065]                  -9781.9362   2.59e+04     -0.377  ␣
 ↪0.706   -6.06e+04    4.11e+04
C(zipcode)[T.98070]                   1.48e+04   1.86e+04      0.794  ␣
 ↪0.427   -2.17e+04    5.13e+04
C(zipcode)[T.98072]                   5.974e+04   2.87e+04      2.085  ␣
 ↪0.037    3581.604    1.16e+05
C(zipcode)[T.98074]                   4.552e+04   2.51e+04      1.813  ␣
 ↪0.070   -3704.279    9.48e+04
C(zipcode)[T.98075]                   3.001e+04   2.45e+04      1.225  ␣
 ↪0.221    -1.8e+04     7.8e+04
C(zipcode)[T.98077]                   2.786e+04   2.99e+04      0.933  ␣
 ↪0.351   -3.07e+04    8.64e+04
C(zipcode)[T.98092]                     -3e+04   1.17e+04     -2.568  ␣
 ↪0.010   -5.29e+04   -7097.416
C(zipcode)[T.98102]                   4.63e+04   2.72e+04      1.704  ␣
 ↪0.088   -6948.937    9.96e+04
C(zipcode)[T.98103]                   2.779e+04   2.41e+04      1.155  ␣
 ↪0.248   -1.94e+04     7.5e+04
C(zipcode)[T.98105]                   1.311e+05   2.54e+04      5.158  ␣
 ↪0.000    8.13e+04    1.81e+05
C(zipcode)[T.98106]                  -1.829e+04   1.92e+04     -0.950  ␣
 ↪0.342    -5.6e+04    1.94e+04
C(zipcode)[T.98107]                   2.342e+04   2.45e+04      0.955  ␣
 ↪0.339   -2.46e+04    7.15e+04
```

```
C(zipcode)[T.98108]                  -4.317e+04    2.2e+04     -1.964      ␣
 ↪0.050   -8.63e+04     -88.709
C(zipcode)[T.98109]                   9.13e+04   2.65e+04      3.442      ␣
 ↪0.001    3.93e+04   1.43e+05
C(zipcode)[T.98112]                   2.066e+05  2.49e+04      8.284      ␣
 ↪0.000    1.58e+05   2.55e+05
C(zipcode)[T.98115]                   7.478e+04  2.45e+04      3.049      ␣
 ↪0.002    2.67e+04   1.23e+05
C(zipcode)[T.98116]                   5.662e+04  2.07e+04      2.738      ␣
 ↪0.006    1.61e+04   9.71e+04
C(zipcode)[T.98117]                   5.627e+04  2.41e+04      2.338      ␣
 ↪0.019    9097.194   1.03e+05
C(zipcode)[T.98118]                  -1.781e+04  2.03e+04     -0.877      ␣
 ↪0.381   -5.76e+04    2.2e+04
C(zipcode)[T.98119]                   9.312e+04  2.49e+04      3.746      ␣
 ↪0.000    4.44e+04   1.42e+05
C(zipcode)[T.98122]                   1843.8601  2.45e+04      0.075      ␣
 ↪0.940   -4.61e+04   4.98e+04
C(zipcode)[T.98125]                   4.806e+04  2.56e+04      1.875      ␣
 ↪0.061   -2184.235   9.83e+04
C(zipcode)[T.98126]                   1.651e+04  1.93e+04      0.855      ␣
 ↪0.392   -2.13e+04   5.43e+04
C(zipcode)[T.98133]                   1.975e+04  2.61e+04      0.758      ␣
 ↪0.448   -3.13e+04   7.08e+04
C(zipcode)[T.98136]                   5.731e+04  1.93e+04      2.973      ␣
 ↪0.003    1.95e+04   9.51e+04
C(zipcode)[T.98144]                   1490.5305  2.32e+04      0.064      ␣
 ↪0.949   -4.39e+04   4.69e+04
C(zipcode)[T.98146]                   9070.1576  1.75e+04      0.518      ␣
 ↪0.604   -2.52e+04   4.34e+04
C(zipcode)[T.98148]                  -7451.0961  2.23e+04     -0.334      ␣
 ↪0.738   -5.11e+04   3.62e+04
C(zipcode)[T.98155]                   3.328e+04  2.71e+04      1.230      ␣
 ↪0.219   -1.98e+04   8.63e+04
C(zipcode)[T.98166]                  -1.014e+04  1.54e+04     -0.657      ␣
 ↪0.511   -4.04e+04   2.01e+04
C(zipcode)[T.98168]                  -2.913e+04  1.76e+04     -1.654      ␣
 ↪0.098   -6.37e+04   5392.000
C(zipcode)[T.98177]                   9.163e+04  2.71e+04      3.378      ␣
 ↪0.001    3.85e+04   1.45e+05
C(zipcode)[T.98178]                  -6.265e+04   1.9e+04     -3.306      ␣
 ↪0.001   -9.98e+04  -2.55e+04
C(zipcode)[T.98188]                  -2.976e+04  1.78e+04     -1.674      ␣
 ↪0.094   -6.46e+04   5089.611
C(zipcode)[T.98198]                   -2.75e+04  1.27e+04     -2.161      ␣
 ↪0.031   -5.24e+04  -2562.458
```

```
C(zipcode)[T.98199]                          1.117e+05   2.33e+04      4.798      ␣
 ↪0.000    6.61e+04    1.57e+05
C(total_sqft_larger_than_neighbours)[T.1] -5.859e+04   7541.234     -7.769      ␣
 ↪0.000   -7.34e+04   -4.38e+04
bedrooms                                  -1.031e+04   1464.808     -7.042      ␣
 ↪0.000   -1.32e+04   -7443.555
bathrooms                                  1.702e+04   2359.169      7.216      ␣
 ↪0.000    1.24e+04    2.16e+04
sqft_living                                 115.4763     12.791      9.028      ␣
 ↪0.000      90.404     140.548
sqft_lot                                      0.4399      0.034     12.839      ␣
 ↪0.000       0.373       0.507
floors                                    -8.434e+04   3003.898    -28.077      ␣
 ↪0.000   -9.02e+04   -7.85e+04
sqft_above                                   76.9522     12.829      5.998      ␣
 ↪0.000      51.806     102.098
sqft_basement                                11.2151     12.667      0.885      ␣
 ↪0.376     -13.612      36.042
yr_built                                  -1253.5786     60.323    -20.781      ␣
 ↪0.000   -1371.816   -1135.341
lat                                        5.379e+04   5.93e+04      0.907      ␣
 ↪0.364   -6.24e+04     1.7e+05
long                                       9.238e+04   5.53e+04      1.670      ␣
 ↪0.095    -1.6e+04    2.01e+05
sqft_living15                                37.7573      2.549     14.811      ␣
 ↪0.000      32.761      42.754
sqft_lot15                                    0.2575      0.054      4.734      ␣
 ↪0.000       0.151       0.364
distance_from_downtown_mile               -3836.2948   1057.881     -3.626      ␣
 ↪0.000   -5909.822   -1762.767
price_per_sqft                             2610.0242     44.511     58.638      ␣
 ↪0.000    2522.779    2697.269
==============================================================================
Omnibus:                    16124.711   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1793971.542
Skew:                           2.873   Prob(JB):                         0.00
Kurtosis:                      47.465   Cond. No.                     4.05e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
[2] The condition number is large, 4.05e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```




[102]:
```
check_for_high_p_val(x1)
```

[102]:
```
<pandas.io.formats.style.Styler at 0x1c25c71e160>
```

With outliers dropped data

[103]:
```
x2 = OLS_sm(df=df_model_2,
              numeric_features=[
                  'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
                  'sqft_above', 'sqft_basement', 'yr_built', 'lat', 'long',
                  'sqft_living15', 'sqft_lot15', 'distance_from_downtown_mile',
                  'price_per_sqft'
              ],
              dependant_var='price',
              categorical_features=[
                  'waterfront', 'condition', 'grade', 'is_renovated',
                  'zipcode', 'total_sqft_larger_than_neighbours'
              ],
              verbose=False)
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_above + sqft_basement + yr_built + lat + long +
sqft_living15 + sqft_lot15 + distance_from_downtown_mile + price_per_sqft +
C(waterfront) + C(condition) + C(grade) + C(is_renovated) + C(zipcode) +
C(total_sqft_larger_than_neighbours)

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
```

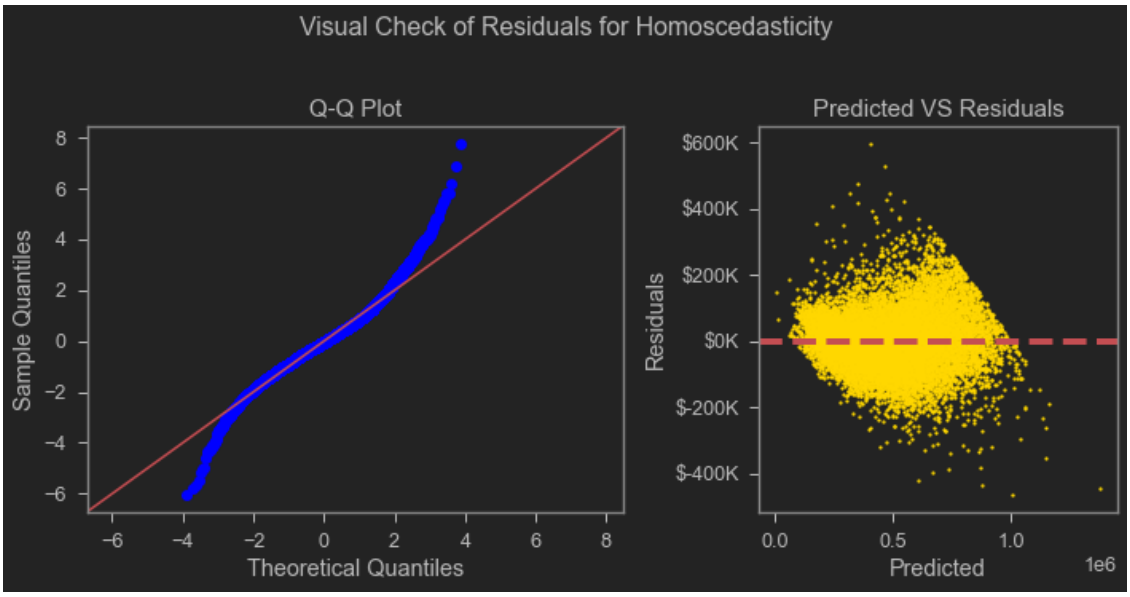
```
==============================================================================
Dep. Variable:                  price   R-squared:                       0.848
Model:                            OLS   Adj. R-squared:                  0.847
Method:                 Least Squares   F-statistic:                     1122.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:41:47   Log-Likelihood:            -2.5306e+05
No. Observations:               19975   AIC:                         5.063e+05
Df Residuals:                   19875   BIC:                         5.071e+05
Df Model:                          99
Covariance Type:            nonrobust
==============================================================================================
                                               coef    std err          t      ␣
↪P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Intercept                                  3.135e+07   4.55e+06      6.897      ␣
↪0.000    2.24e+07    4.03e+07
C(waterfront)[T.1]                          2.45e+05   1.14e+04     21.542      ␣
↪0.000    2.23e+05    2.67e+05
C(condition)[T.2]                          5.125e+04   1.61e+04      3.178      ␣
↪0.001    1.96e+04    8.29e+04
C(condition)[T.3]                          8.052e+04    1.5e+04      5.363      ␣
↪0.000    5.11e+04     1.1e+05
C(condition)[T.4]                           9.95e+04    1.5e+04      6.622      ␣
↪0.000    7.01e+04    1.29e+05
C(condition)[T.5]                          1.225e+05   1.51e+04      8.097      ␣
↪0.000    9.28e+04    1.52e+05
C(grade)[T.4]                             -9.869e+04   7.87e+04     -1.254      ␣
↪0.210   -2.53e+05    5.55e+04
C(grade)[T.5]                             -1.221e+05   7.74e+04     -1.577      ␣
↪0.115   -2.74e+05    2.97e+04
C(grade)[T.6]                             -1.103e+05   7.73e+04     -1.427      ␣
↪0.153   -2.62e+05    4.12e+04
C(grade)[T.7]                             -8.392e+04   7.73e+04     -1.086      ␣
↪0.277   -2.35e+05    6.75e+04
C(grade)[T.8]                             -4.933e+04   7.73e+04     -0.638      ␣
↪0.523   -2.01e+05    1.02e+05
C(grade)[T.9]                              9195.2162   7.73e+04      0.119      ␣
↪0.905   -1.42e+05    1.61e+05
C(grade)[T.10]                             4.201e+04   7.74e+04      0.543      ␣
↪0.587    -1.1e+05    1.94e+05
C(grade)[T.11]                             7.426e+04   7.78e+04      0.954      ␣
↪0.340   -7.82e+04    2.27e+05
C(grade)[T.12]                            -5.505e+04   8.94e+04     -0.616      ␣
↪0.538    -2.3e+05     1.2e+05
C(is_renovated)[T.1]                       2.685e+04   3405.891      7.882      ␣
↪0.000    2.02e+04    3.35e+04
```

```
C(zipcode)[T.98002]                  -1.687e+04   7102.867     -2.376      ␣
 ↪0.018   -3.08e+04   -2950.727
C(zipcode)[T.98003]                   -754.3807   6299.457     -0.120      ␣
 ↪0.905   -1.31e+04    1.16e+04
C(zipcode)[T.98004]                   2.966e+05   1.41e+04     20.997      ␣
 ↪0.000    2.69e+05    3.24e+05
C(zipcode)[T.98005]                   1.635e+05   1.41e+04     11.612      ␣
 ↪0.000    1.36e+05    1.91e+05
C(zipcode)[T.98006]                   1.189e+05   1.23e+04      9.687      ␣
 ↪0.000    9.49e+04    1.43e+05
C(zipcode)[T.98007]                     9.2e+04   1.44e+04      6.383      ␣
 ↪0.000    6.37e+04     1.2e+05
C(zipcode)[T.98008]                   9.018e+04   1.39e+04      6.498      ␣
 ↪0.000     6.3e+04    1.17e+05
C(zipcode)[T.98010]                   3.541e+04    1.2e+04      2.953      ␣
 ↪0.003    1.19e+04    5.89e+04
C(zipcode)[T.98011]                   4.075e+04   1.62e+04      2.521      ␣
 ↪0.012    9066.545    7.24e+04
C(zipcode)[T.98014]                   1.334e+04   1.85e+04      0.719      ␣
 ↪0.472    -2.3e+04    4.97e+04
C(zipcode)[T.98019]                   7701.9085   1.86e+04      0.414      ␣
 ↪0.679   -2.88e+04    4.42e+04
C(zipcode)[T.98022]                   5753.5045   1.04e+04      0.552      ␣
 ↪0.581   -1.47e+04    2.62e+04
C(zipcode)[T.98023]                  -4810.7727   6213.735     -0.774      ␣
 ↪0.439    -1.7e+04    7368.666
C(zipcode)[T.98024]                   3.783e+04   1.75e+04      2.157      ␣
 ↪0.031    3447.655    7.22e+04
C(zipcode)[T.98027]                   5.153e+04   1.25e+04      4.133      ␣
 ↪0.000    2.71e+04     7.6e+04
C(zipcode)[T.98028]                   4.196e+04   1.56e+04      2.691      ␣
 ↪0.007    1.14e+04    7.25e+04
C(zipcode)[T.98029]                   7.407e+04    1.4e+04      5.279      ␣
 ↪0.000    4.66e+04    1.02e+05
C(zipcode)[T.98030]                  -3.964e+04   7513.989     -5.275      ␣
 ↪0.000   -5.44e+04   -2.49e+04
C(zipcode)[T.98031]                  -4.882e+04   8160.577     -5.983      ␣
 ↪0.000   -6.48e+04   -3.28e+04
C(zipcode)[T.98032]                  -4.141e+04   8502.239     -4.870      ␣
 ↪0.000   -5.81e+04   -2.47e+04
C(zipcode)[T.98033]                   1.584e+05   1.43e+04     11.053      ␣
 ↪0.000     1.3e+05    1.86e+05
C(zipcode)[T.98034]                   6.514e+04   1.47e+04      4.429      ␣
 ↪0.000    3.63e+04     9.4e+04
C(zipcode)[T.98038]                  -2.001e+04   9495.244     -2.108      ␣
 ↪0.035   -3.86e+04   -1401.388
```

```
C(zipcode)[T.98039]                    4.199e+05   3.68e+04     11.418      ␣
 ↪0.000    3.48e+05    4.92e+05
C(zipcode)[T.98040]                    2.303e+05    1.3e+04     17.682      ␣
 ↪0.000    2.05e+05    2.56e+05
C(zipcode)[T.98042]                   -4.113e+04   7776.533     -5.289      ␣
 ↪0.000   -5.64e+04   -2.59e+04
C(zipcode)[T.98045]                    3.52e+04   1.78e+04      1.982      ␣
 ↪0.048     382.111       7e+04
C(zipcode)[T.98052]                    1.096e+05   1.46e+04     7.493      ␣
 ↪0.000    8.09e+04    1.38e+05
C(zipcode)[T.98053]                    1.056e+05   1.59e+04     6.624      ␣
 ↪0.000    7.44e+04    1.37e+05
C(zipcode)[T.98055]                   -5.391e+04   9479.975     -5.687      ␣
 ↪0.000   -7.25e+04   -3.53e+04
C(zipcode)[T.98056]                   -1.716e+04   1.07e+04     -1.598      ␣
 ↪0.110   -3.82e+04    3886.476
C(zipcode)[T.98058]                      -5e+04   9251.578     -5.404      ␣
 ↪0.000   -6.81e+04   -3.19e+04
C(zipcode)[T.98059]                   -1.239e+04   1.06e+04     -1.173      ␣
 ↪0.241   -3.31e+04    8307.861
C(zipcode)[T.98065]                    1.306e+04   1.61e+04     0.809      ␣
 ↪0.419   -1.86e+04    4.47e+04
C(zipcode)[T.98070]                    1.171e+05   1.08e+04     10.820      ␣
 ↪0.000    9.59e+04    1.38e+05
C(zipcode)[T.98072]                    7.676e+04   1.64e+04     4.678      ␣
 ↪0.000    4.46e+04    1.09e+05
C(zipcode)[T.98074]                    7.836e+04   1.45e+04     5.397      ␣
 ↪0.000    4.99e+04    1.07e+05
C(zipcode)[T.98075]                    9.252e+04   1.43e+04     6.473      ␣
 ↪0.000    6.45e+04    1.21e+05
C(zipcode)[T.98077]                    7.694e+04   1.74e+04     4.418      ␣
 ↪0.000    4.28e+04    1.11e+05
C(zipcode)[T.98092]                   -3.016e+04   6549.839     -4.604      ␣
 ↪0.000    -4.3e+04   -1.73e+04
C(zipcode)[T.98102]                    1.105e+05   1.55e+04     7.139      ␣
 ↪0.000    8.01e+04    1.41e+05
C(zipcode)[T.98103]                    1.131e+05   1.34e+04     8.450      ␣
 ↪0.000    8.69e+04    1.39e+05
C(zipcode)[T.98105]                    1.384e+05   1.43e+04     9.655      ␣
 ↪0.000     1.1e+05    1.67e+05
C(zipcode)[T.98106]                   -6233.7896   1.07e+04     -0.580      ␣
 ↪0.562   -2.73e+04    1.48e+04
C(zipcode)[T.98107]                    1.169e+05   1.37e+04     8.562      ␣
 ↪0.000    9.02e+04    1.44e+05
C(zipcode)[T.98108]                   -2.926e+04   1.21e+04     -2.415      ␣
 ↪0.016    -5.3e+04   -5508.330
```

```
C(zipcode)[T.98109]                    1.434e+05   1.52e+04      9.427      ␣
 ↪0.000     1.14e+05    1.73e+05
C(zipcode)[T.98112]                    1.437e+05   1.43e+04     10.037      ␣
 ↪0.000     1.16e+05    1.72e+05
C(zipcode)[T.98115]                    1.319e+05   1.36e+04      9.682      ␣
 ↪0.000     1.05e+05    1.59e+05
C(zipcode)[T.98116]                    1.218e+05   1.17e+04     10.446      ␣
 ↪0.000     9.89e+04    1.45e+05
C(zipcode)[T.98117]                    1.399e+05   1.34e+04     10.420      ␣
 ↪0.000     1.14e+05    1.66e+05
C(zipcode)[T.98118]                    3331.3620   1.12e+04      0.298      ␣
 ↪0.766    -1.86e+04    2.53e+04
C(zipcode)[T.98119]                      1.5e+05   1.41e+04     10.669      ␣
 ↪0.000     1.22e+05    1.78e+05
C(zipcode)[T.98122]                    6.034e+04   1.36e+04      4.452      ␣
 ↪0.000     3.38e+04    8.69e+04
C(zipcode)[T.98125]                    7.575e+04   1.43e+04      5.314      ␣
 ↪0.000     4.78e+04    1.04e+05
C(zipcode)[T.98126]                    5.542e+04   1.08e+04      5.122      ␣
 ↪0.000     3.42e+04    7.66e+04
C(zipcode)[T.98133]                    5.033e+04   1.45e+04      3.465      ␣
 ↪0.001     2.19e+04    7.88e+04
C(zipcode)[T.98136]                    1.115e+05   1.09e+04     10.261      ␣
 ↪0.000     9.02e+04    1.33e+05
C(zipcode)[T.98144]                    2.515e+04   1.29e+04      1.957      ␣
 ↪0.050      -41.450    5.03e+04
C(zipcode)[T.98146]                    1.768e+04   9808.419      1.802      ␣
 ↪0.071    -1546.044    3.69e+04
C(zipcode)[T.98148]                   -1.158e+04   1.22e+04     -0.950      ␣
 ↪0.342    -3.55e+04    1.23e+04
C(zipcode)[T.98155]                    5.234e+04   1.51e+04      3.457      ␣
 ↪0.001     2.27e+04     8.2e+04
C(zipcode)[T.98166]                    3.452e+04   8667.780      3.982      ␣
 ↪0.000     1.75e+04    5.15e+04
C(zipcode)[T.98168]                   -4.445e+04   9713.095     -4.577      ␣
 ↪0.000    -6.35e+04   -2.54e+04
C(zipcode)[T.98177]                    1.273e+05   1.52e+04      8.358      ␣
 ↪0.000     9.75e+04    1.57e+05
C(zipcode)[T.98178]                   -5.367e+04   1.04e+04     -5.162      ␣
 ↪0.000     -7.4e+04   -3.33e+04
C(zipcode)[T.98188]                   -3.726e+04   9716.504     -3.835      ␣
 ↪0.000    -5.63e+04   -1.82e+04
C(zipcode)[T.98198]                   -7729.3254   7005.717     -1.103      ␣
 ↪0.270    -2.15e+04    6002.465
C(zipcode)[T.98199]                    1.665e+05   1.32e+04     12.619      ␣
 ↪0.000     1.41e+05    1.92e+05
```

```
C(total_sqft_larger_than_neighbours)[T.1] -3.988e+04   4198.624     -9.497  ␣
 ↪0.000   -4.81e+04   -3.16e+04
bedrooms                                   -767.9934    840.258     -0.914  ␣
 ↪0.361   -2414.970    878.983
bathrooms                                  1.152e+04   1370.178      8.405  ␣
 ↪0.000    8830.329   1.42e+04
sqft_living                                  60.8288      7.735      7.864  ␣
 ↪0.000      45.668     75.990
sqft_lot                                      0.3635      0.020     18.631  ␣
 ↪0.000       0.325      0.402
floors                                     -4.533e+04   1734.429    -26.137  ␣
 ↪0.000   -4.87e+04   -4.19e+04
sqft_above                                   58.8240      7.741      7.599  ␣
 ↪0.000      43.652     73.996
sqft_basement                                 6.8900      7.654      0.900  ␣
 ↪0.368      -8.112     21.892
yr_built                                    -882.0364    35.038    -25.173  ␣
 ↪0.000    -950.714   -813.358
lat                                        -4175.2176   3.41e+04     -0.122  ␣
 ↪0.903    -7.1e+04   6.27e+04
long                                        2.396e+05   3.37e+04      7.109  ␣
 ↪0.000    1.74e+05   3.06e+05
sqft_living15                                43.1165      1.571     27.448  ␣
 ↪0.000      40.037     46.195
sqft_lot15                                    0.2235      0.031      7.166  ␣
 ↪0.000       0.162      0.285
distance_from_downtown_mile                -6722.5616    607.014    -11.075  ␣
 ↪0.000   -7912.359   -5532.764
price_per_sqft                              1264.6075    27.419     46.121  ␣
 ↪0.000    1210.863   1318.352
==============================================================================
Omnibus:                     1628.002   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5316.079
Skew:                           0.403   Prob(JB):                         0.00
Kurtosis:                       5.395   Cond. No.                     4.07e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
[2] The condition number is large, 4.07e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```




[104]:
```
check_for_high_p_val(x2)
```

[104]:
```
<pandas.io.formats.style.Styler at 0x1c25c545d90>
```

### 4.8.1 Consider the above one as base model

It has features with high p value, and residual plots are very abnormal.

## 4.9 Scaling

Robust statistics have good performance when distributions are not normal. These robust estimators typically have inferior statistical efficiency compared to conventional estimators for data drawn from a distribution without outliers (such as a normal distribution), but have superior efficiency for data drawn from a mixture distribution or from a heavy-tailed distribution, for which non-robust measures such as the standard deviation should not be used.

This matches the dataset characteristics.

### 4.9.1 based on mean and IQR range

[105]:
```
from sklearn.preprocessing import RobustScaler
```

[106]:
```
scaler = RobustScaler()
```

[107]:
```
# checkpoint
IQR__df = df_model_2.copy()
```

[108]:
```
# this is out of order, and added while trouble interpreting results, continue␣
↪from next block
```

```
IQR__df.reset_index(inplace=True)

X__price = IQR__df.drop(columns=['price','index'])
X_price_only = IQR__df[['price']]

# sans price
X_unscaled_pr = scaler.fit_transform(X__price)

df_for_last_step = pd.DataFrame(X_unscaled_pr,columns=X__price.columns)

df_for_last_step = pd.concat([X_price_only,df_for_last_step],axis=1)
```

[109]:
```
# back to OG steps
IQR__df = df_model_2.copy()
```

[110]:
```
# whole df
X_ = IQR__df
X = scaler.fit_transform(X_)
```

[111]:
```
scaler.center_ #means
```

[111]:
```
array([ 4.35000e+05,  3.00000e+00,  2.00000e+00,  1.85000e+03,
        7.50000e+03,  1.00000e+00,  0.00000e+00,  3.00000e+00,
        7.00000e+00,  1.51000e+03,  0.00000e+00,  1.97400e+03,
        9.80650e+04,  4.75660e+01, -1.22232e+02,  1.80000e+03,
        7.52400e+03,  1.05800e+01,  4.47500e+01,  0.00000e+00,
        0.00000e+00])
```

[112]:
```
scaler.scale_ #medians
```

[112]:
```
array([2.8000e+05, 1.0000e+00, 1.0000e+00, 1.0200e+03, 5.2600e+03,
       1.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00, 9.2000e+02,
       5.0000e+02, 4.4000e+01, 8.5000e+01, 2.2215e-01, 2.0700e-01,
       7.9000e+02, 4.7930e+03, 9.7050e+00, 4.5500e+01, 1.0000e+00,
       1.0000e+00])
```

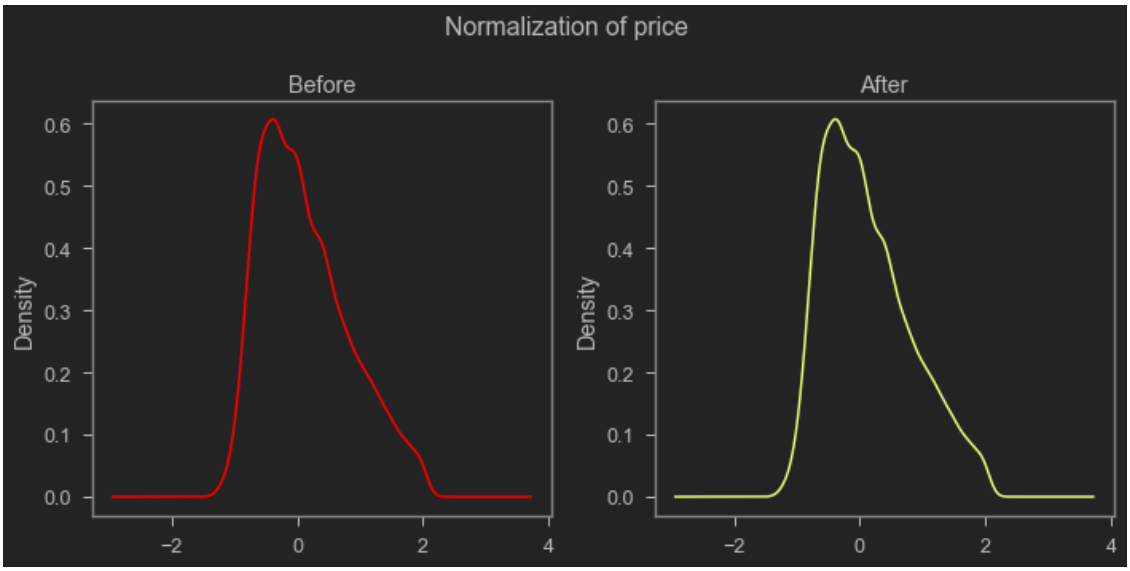
[113]:
```
X_df_ro = pd.DataFrame(X,columns=IQR__df.columns)
X_df_ro
```

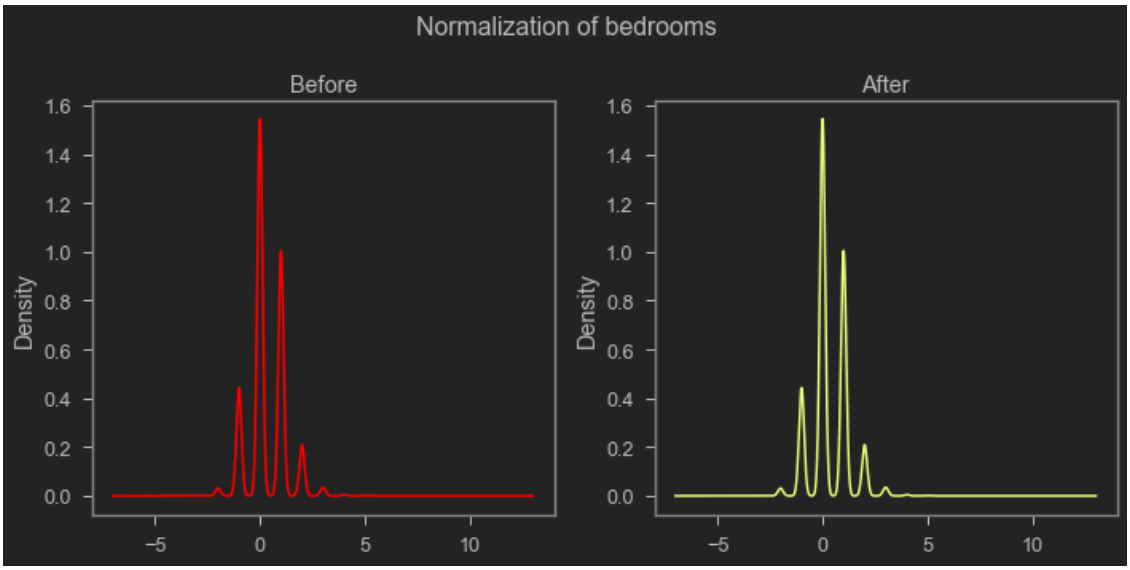
[113]:

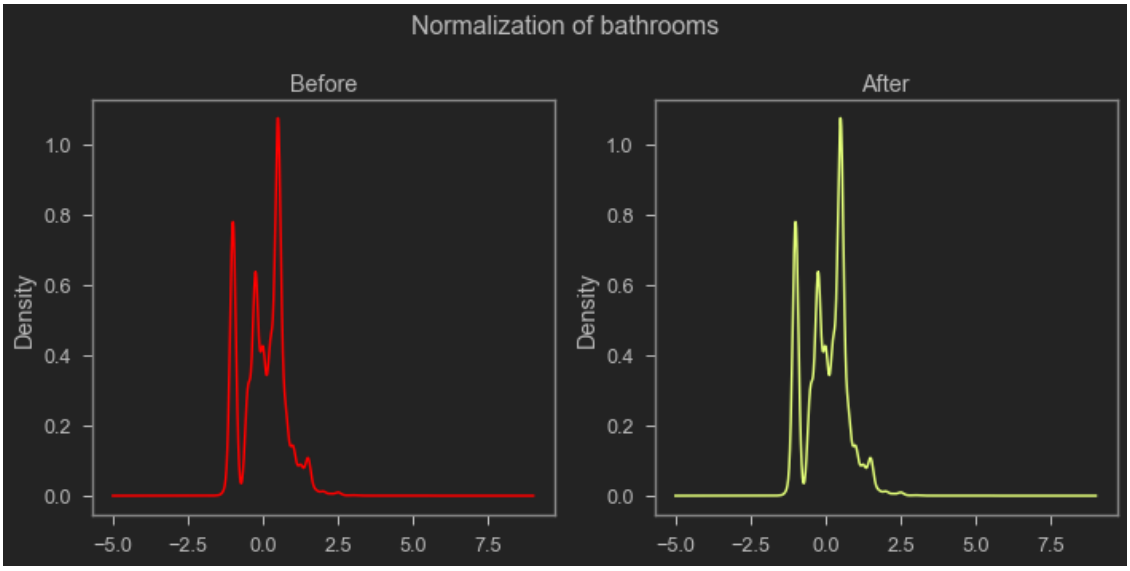
| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 0 | -0.761071 | 0.0 | … | 0.0 | 0.0 |
| 1 | 0.367857 | 0.0 | … | 0.0 | 1.0 |
| 2 | -0.910714 | -1.0 | … | 0.0 | 0.0 |
| 3 | 0.603571 | 1.0 | … | 0.0 | 0.0 |
| 4 | 0.267857 | 0.0 | … | 0.0 | 0.0 |
| … | … | … | … | … | … |
| 19970 | -0.267857 | 0.0 | … | 0.0 | 0.0 |
| 19971 | -0.125000 | 1.0 | … | 0.0 | 0.0 |

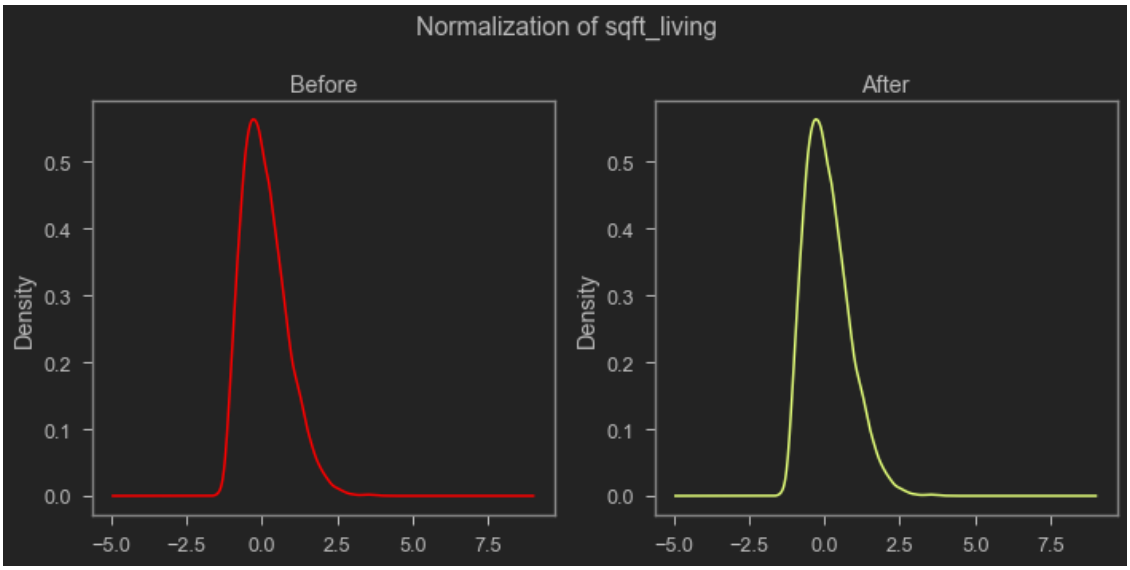
```
19972 -0.117496      -1.0  …                    0.0           0.0
19973 -0.125000       0.0  …                    0.0           0.0
19974 -0.392857      -1.0  …                    0.0           0.0

[19975 rows x 21 columns]
```

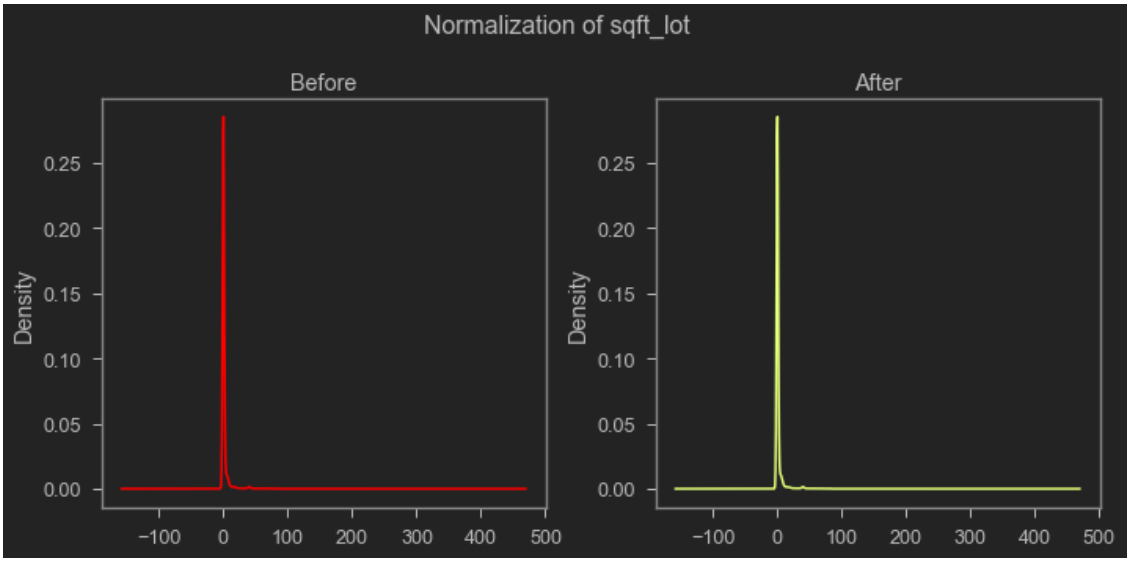
[114]:
```
for column in X_df_ro:
    fig, (ax1,ax2) = plt.subplots(ncols=2, figsize=(10,5))
    X_df_ro[column].plot(kind='kde',title='Before',colormap='hsv',ax=ax1)
    X_df_ro[column].plot(kind='kde',title='After',colormap='Wistia',ax=ax2)
    plt.suptitle(f'Normalization of {column}')
    plt.tight_layout()
    plt.show()
```

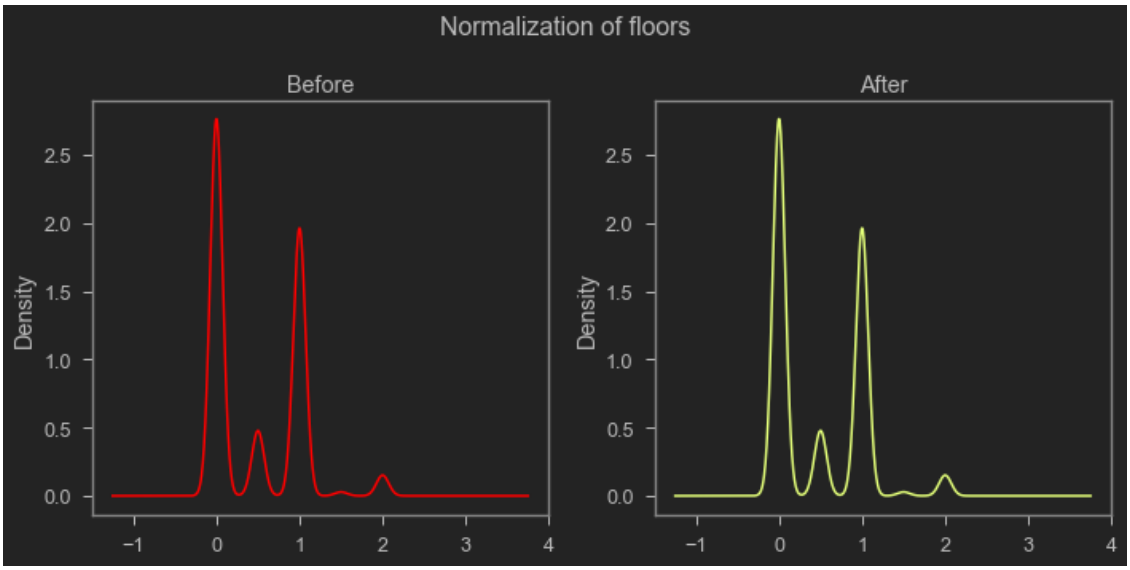


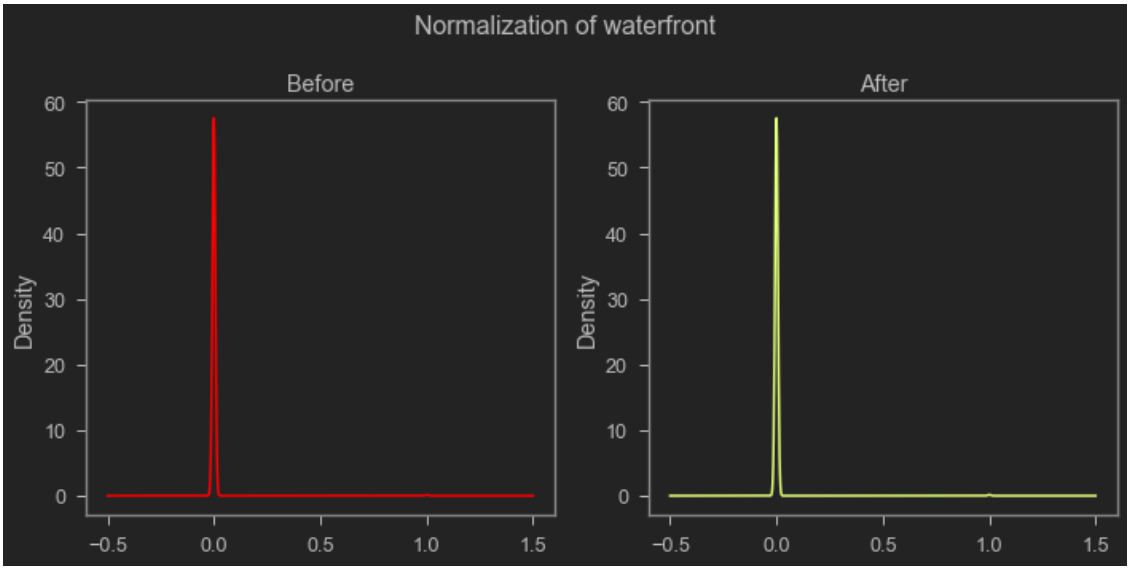

Normalization of bedrooms
Before
After
Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−5
0
5
10

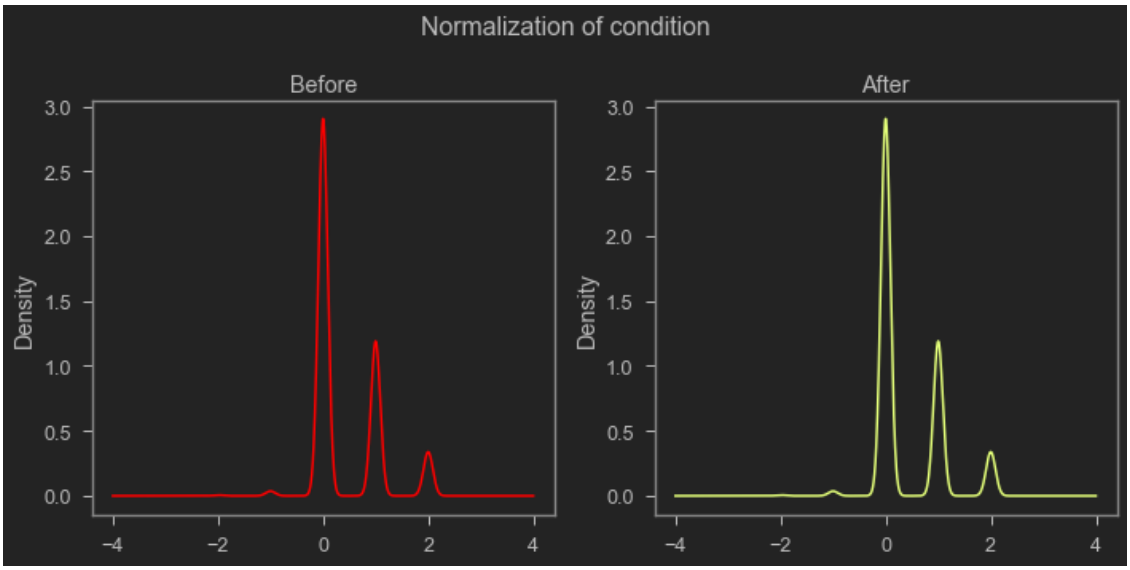

Normalization of bathrooms
Before
After
Density
1.0
0.8
0.6
0.4
0.2
0.0
−5.0
−2.5
0.0
2.5
5.0
7.5

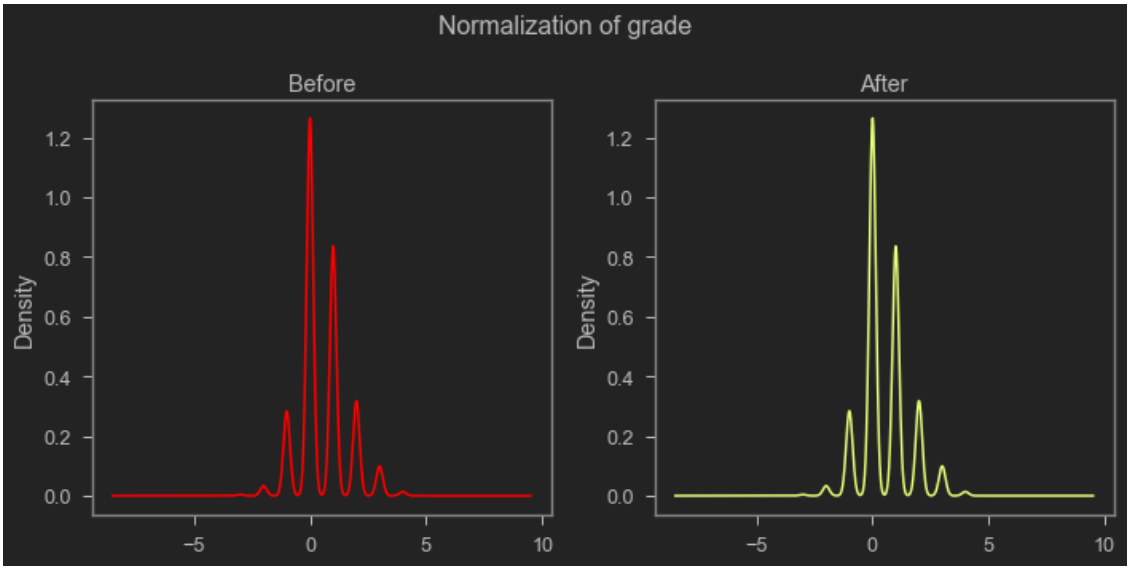

Normalization of sqft_living
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
−5.0
−2.5
0.0
2.5
5.0
7.5

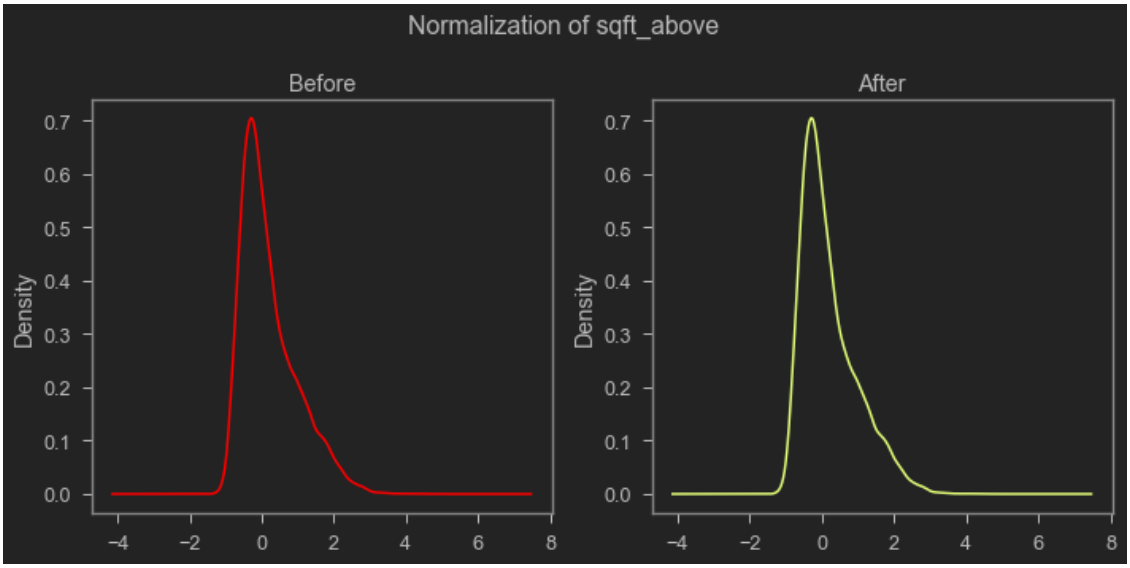

Normalization of sqft_lot
Before
After
Density
0.00
0.05
0.10
0.15
0.20
0.25
−100
0
100
200
300
400
500

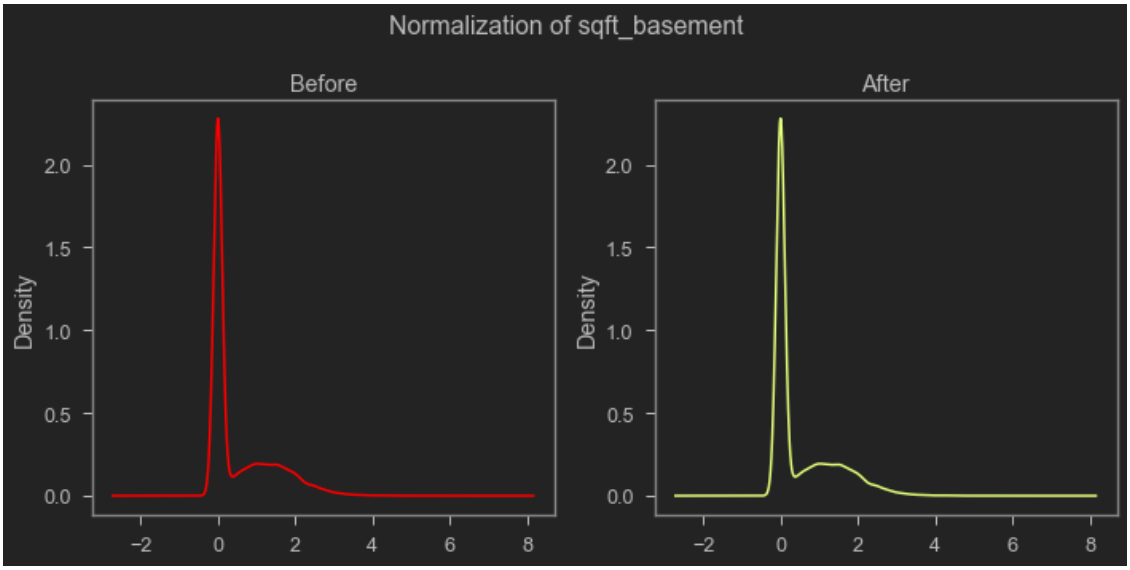

Normalization of floors
Before
After
Density
2.5
2.0
1.5
1.0
0.5
0.0
−1
0
1
2
3
4

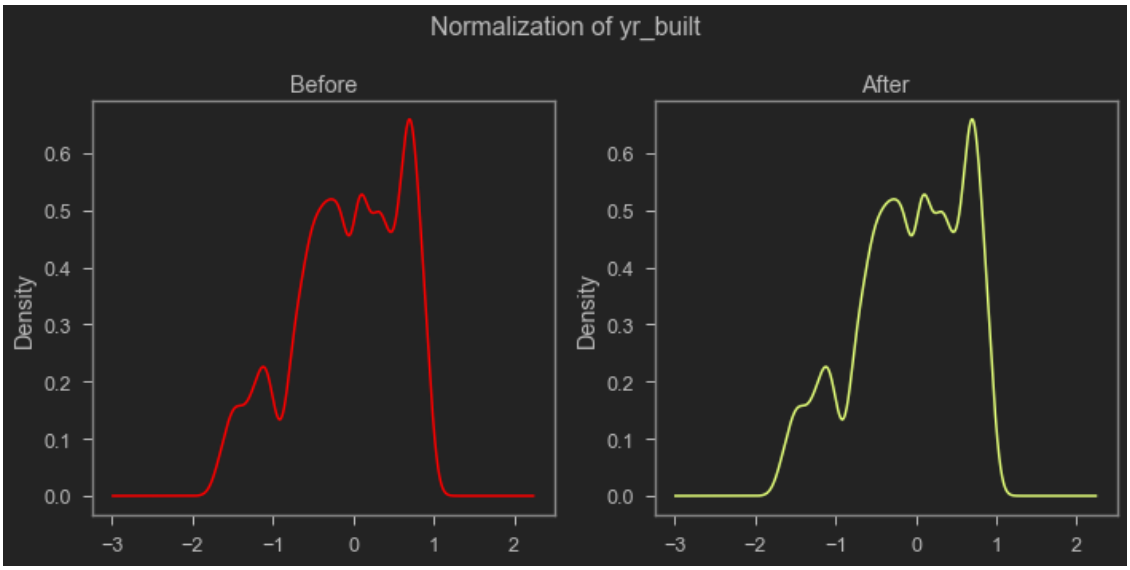

Normalization of waterfront
Before
After
Density
60
50
40
30
20
10
0
−0.5
0.0
0.5
1.0
1.5

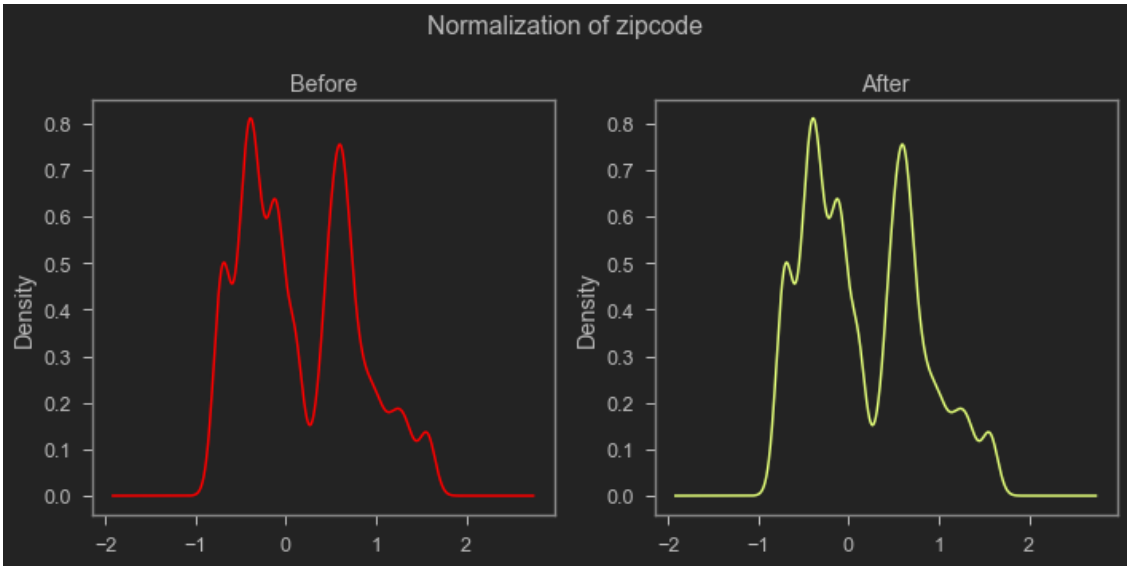

Normalization of condition
Before
After
Density
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−4
−2
0
2
4

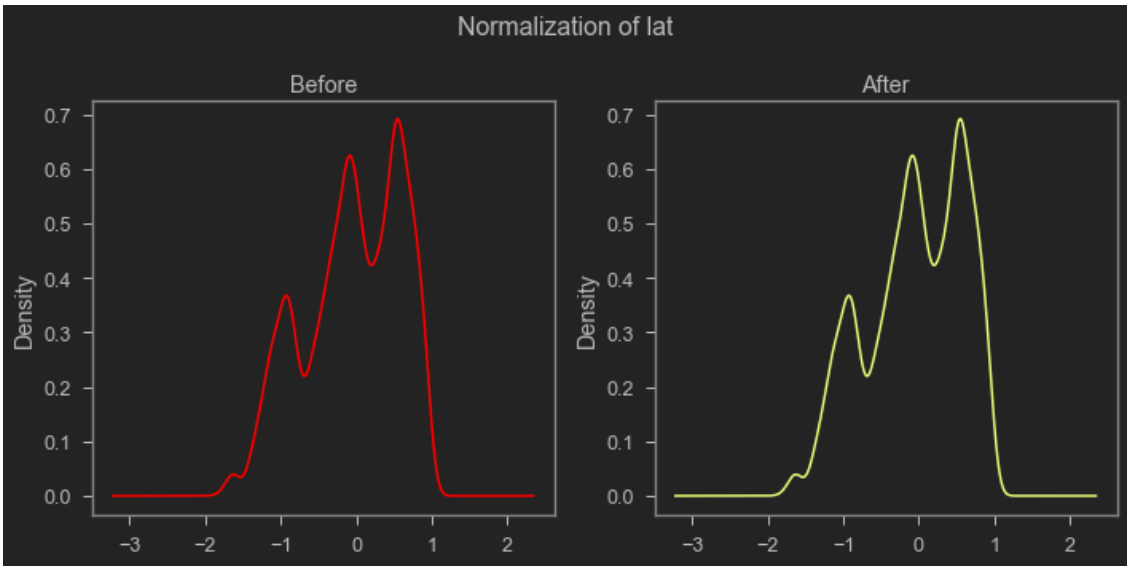

Normalization of grade
Before
After
Density
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−5
0
5
10

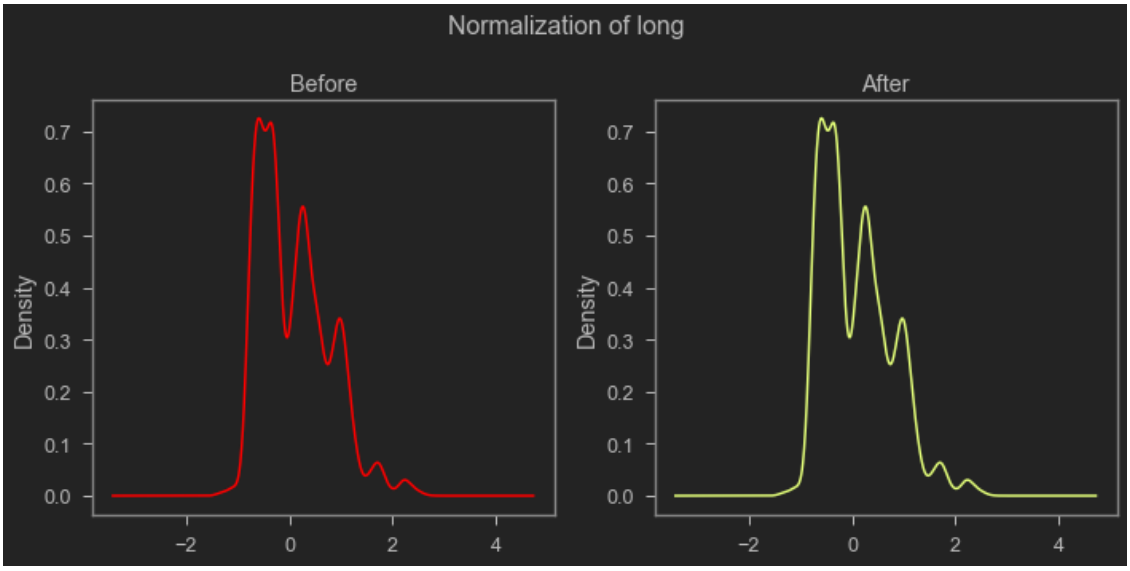

Normalization of sqft_above
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−4
−2
0
2
4
6
8

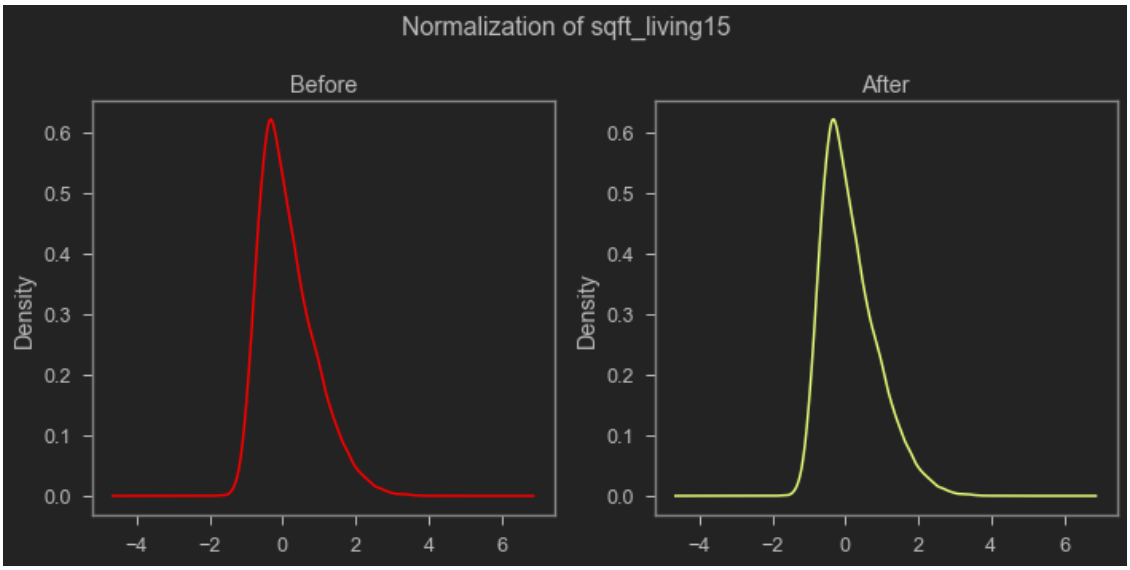

Normalization of sqft_basement
Before
After
Density
0.0
0.5
1.0
1.5
2.0
−2
0
2
4
6
8

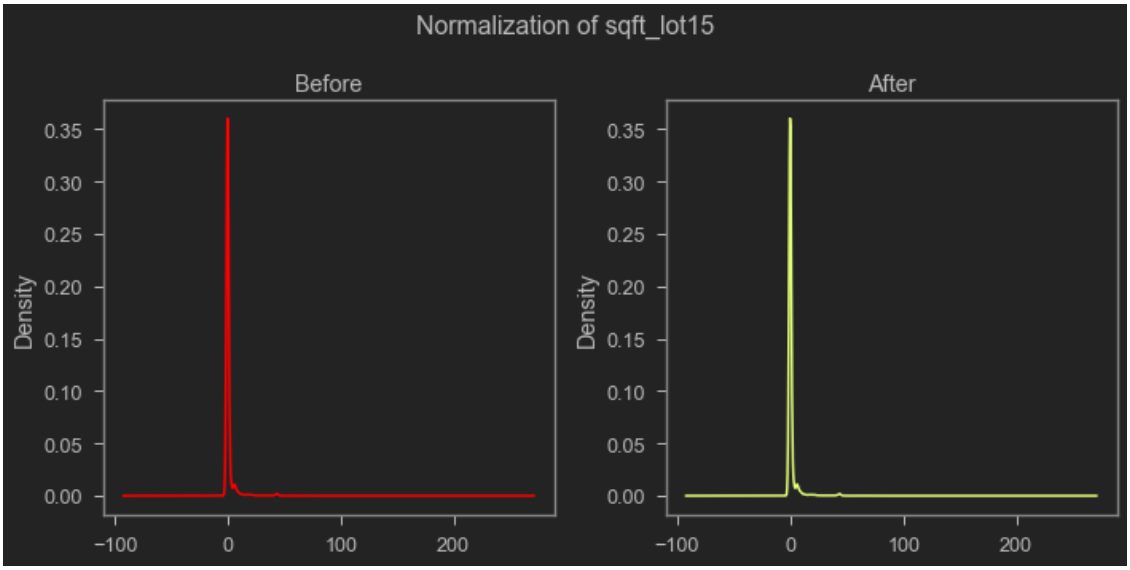

Normalization of yr_built
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−3
−2
−1
0
1
2

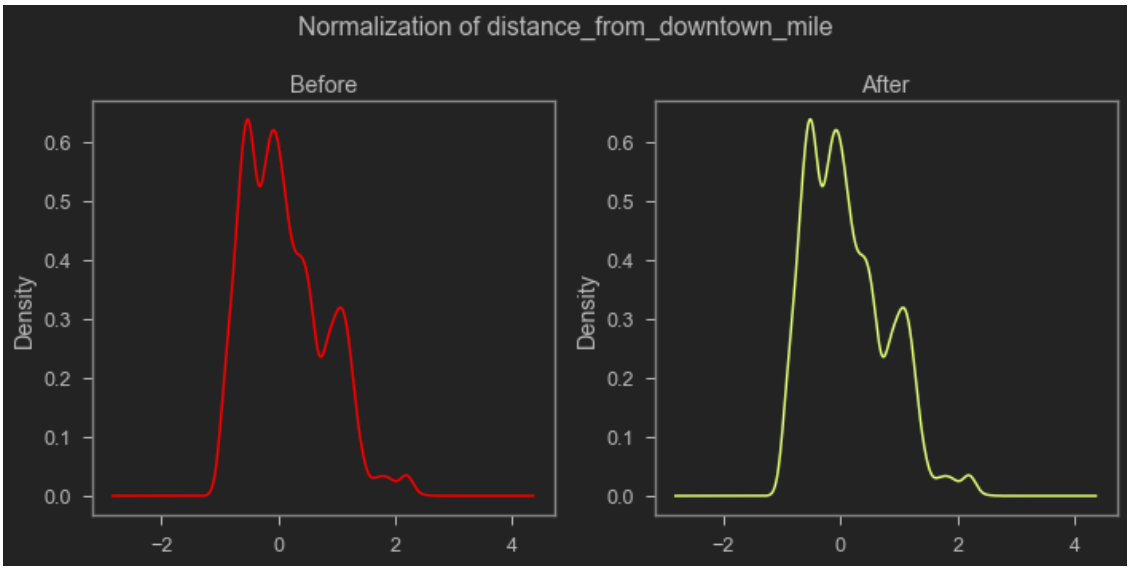

Normalization of zipcode
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−2
−1
0
1
2

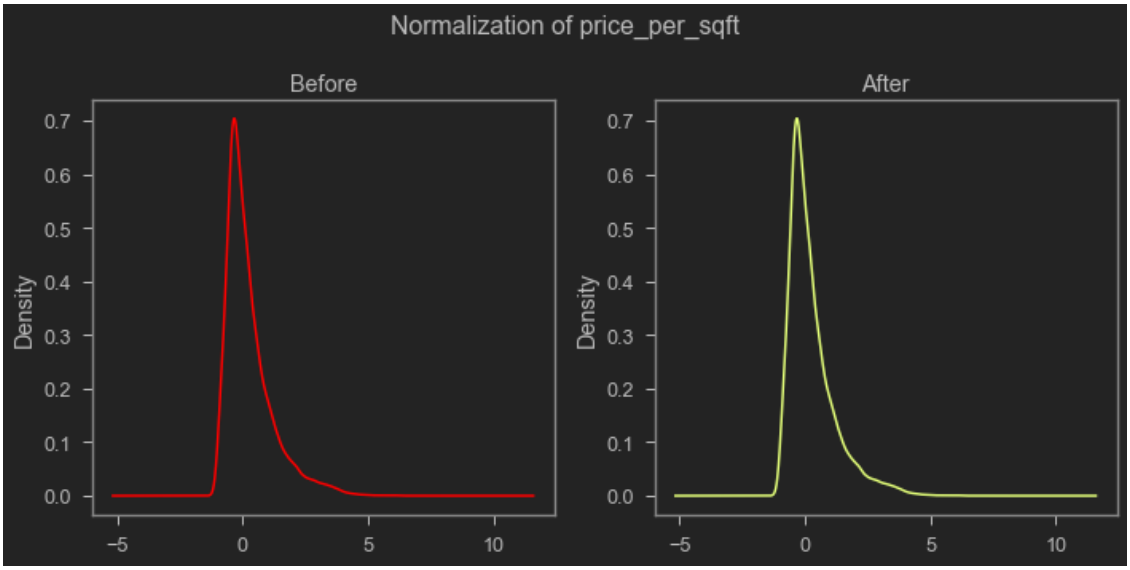

Normalization of lat
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−3
−2
−1
0
1
2

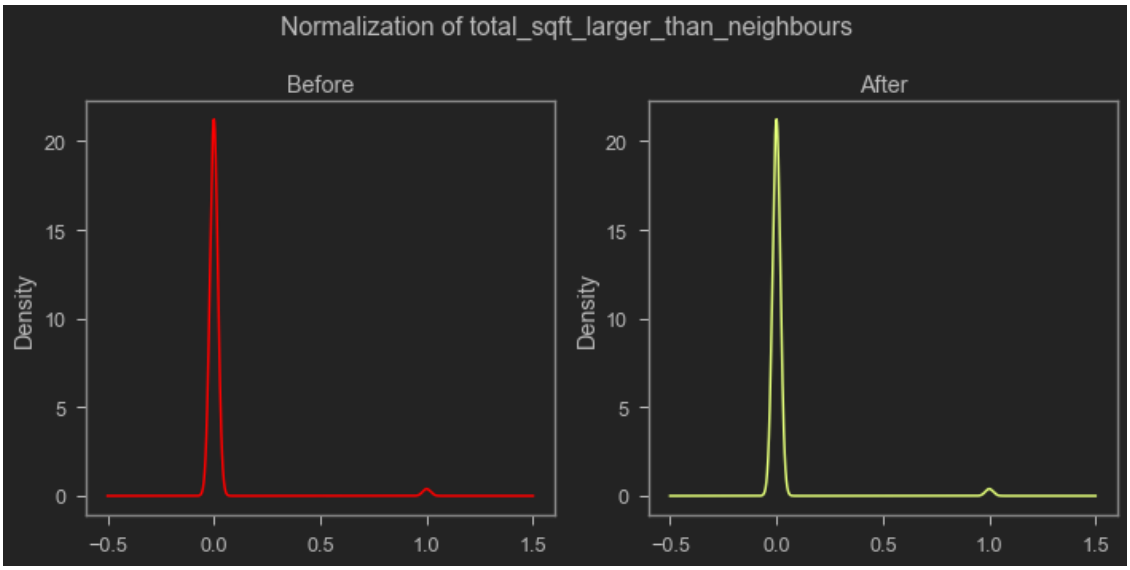

Normalization of long
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−2
0
2
4

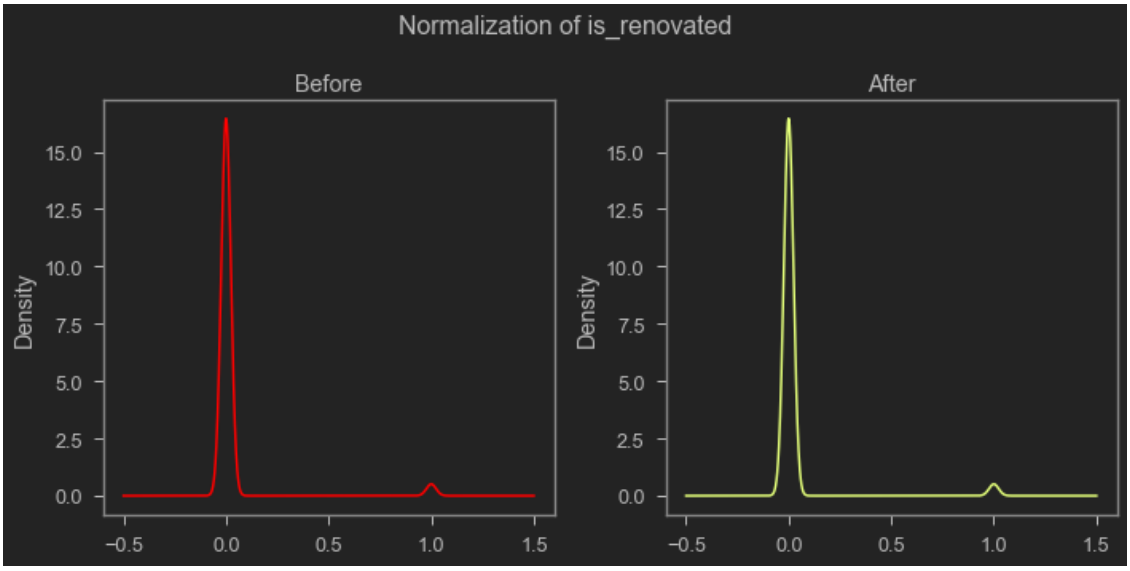

Normalization of sqft_living15
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−4
−2
0
2
4
6


Normalization of sqft_lot15
Before
After
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
−100
0
100
200


Normalization of distance_from_downtown_mile
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−2
0
2
4


Normalization of price_per_sqft
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−5
0
5
10




### 4.9.2 Based on Z score

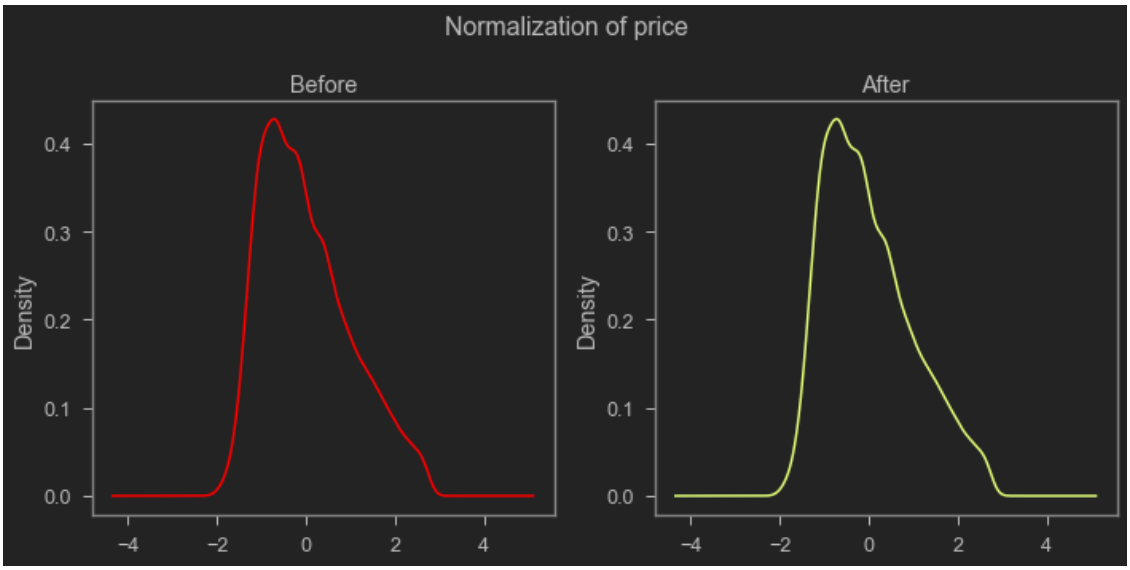
[115]:
```
Z__df = df_model_2.copy()
```

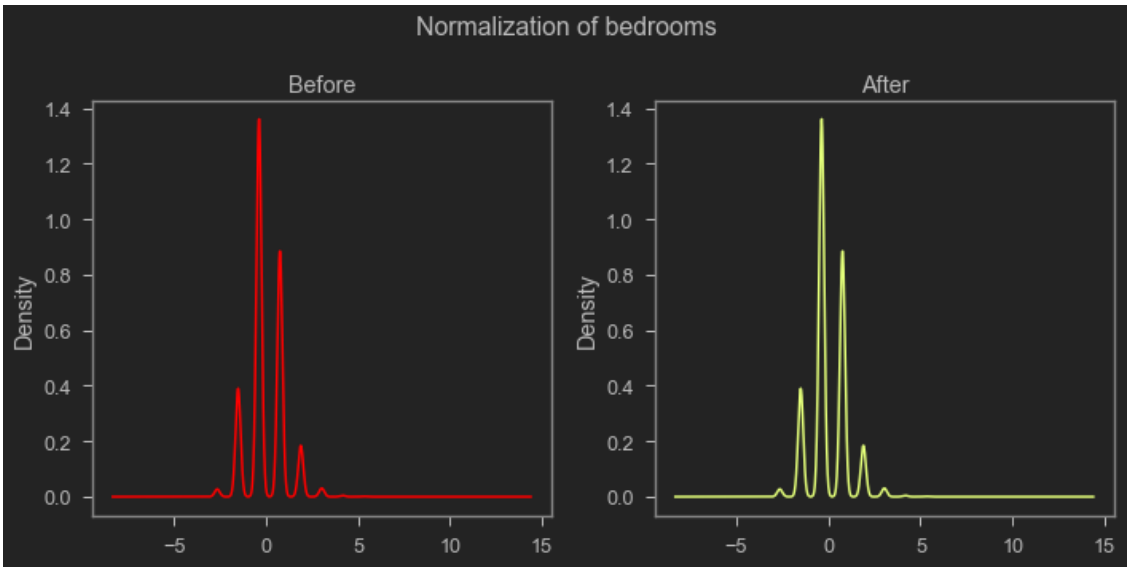
[116]:
```
from sklearn.preprocessing import StandardScaler
```

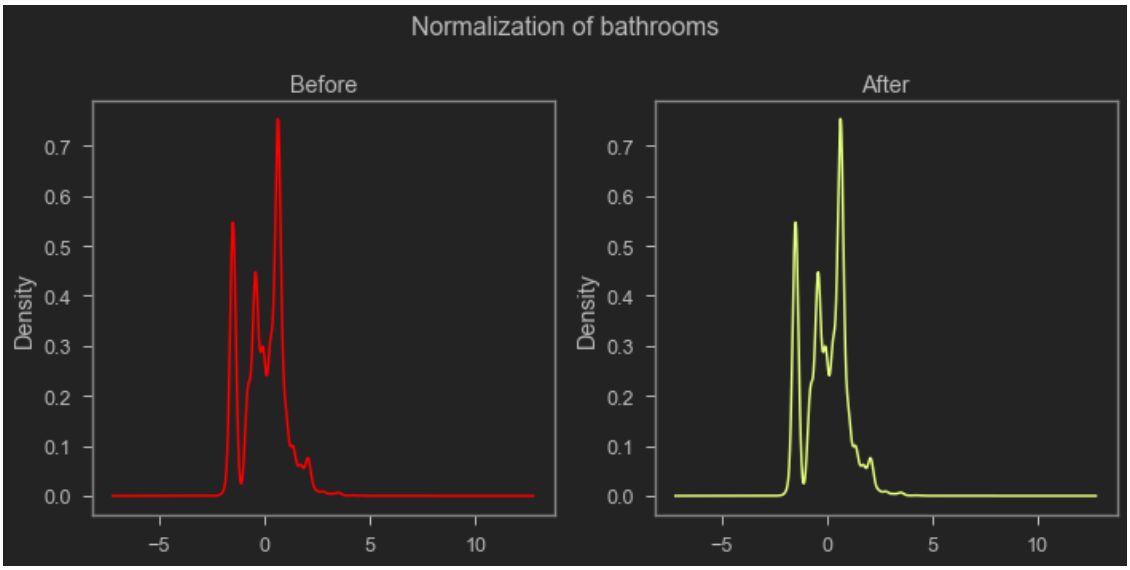
[117]:
```
scaler = StandardScaler()
X_ = Z__df
X = scaler.fit_transform(X_)
```

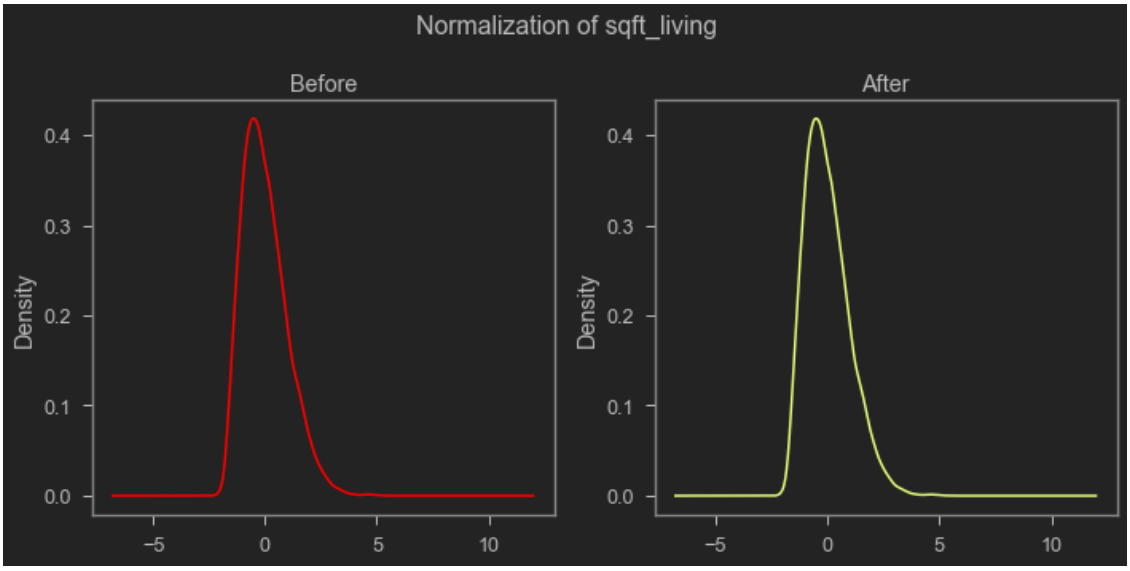
[118]:
```
X_df_Z = pd.DataFrame(X,columns=Z__df.columns)
```

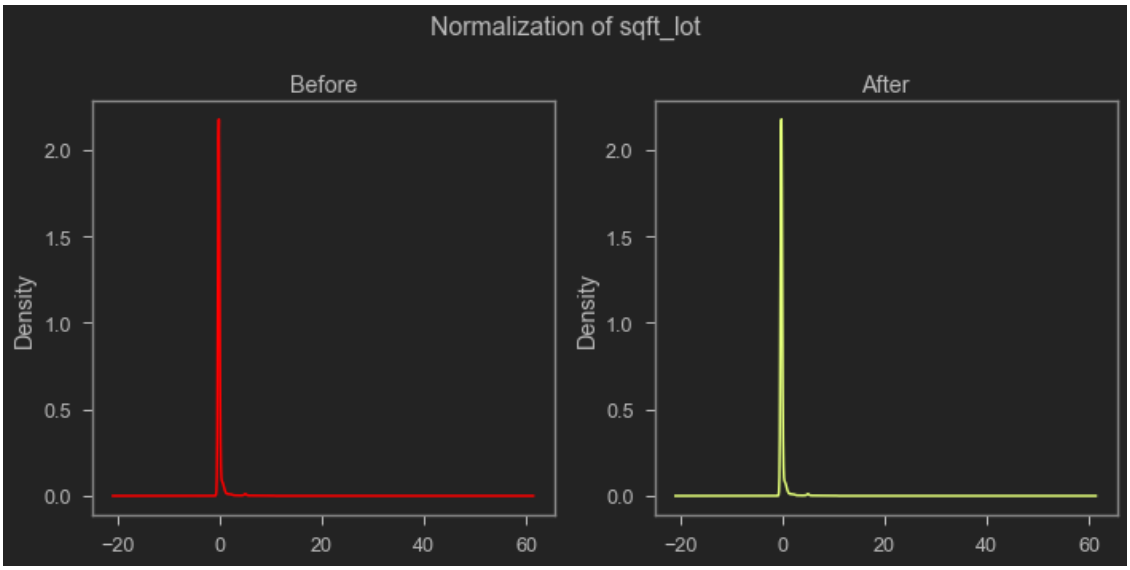
[119]:
```
for column in X_df_Z:
    fig, (ax1,ax2) = plt.subplots(ncols=2, figsize=(10,5))
    X_df_Z[column].plot(kind='kde',title='Before',colormap='hsv',ax=ax1)
    X_df_Z[column].plot(kind='kde',title='After',colormap='Wistia',ax=ax2)
    plt.suptitle(f'Normalization of {column}')
    plt.tight_layout()
    plt.show()
```

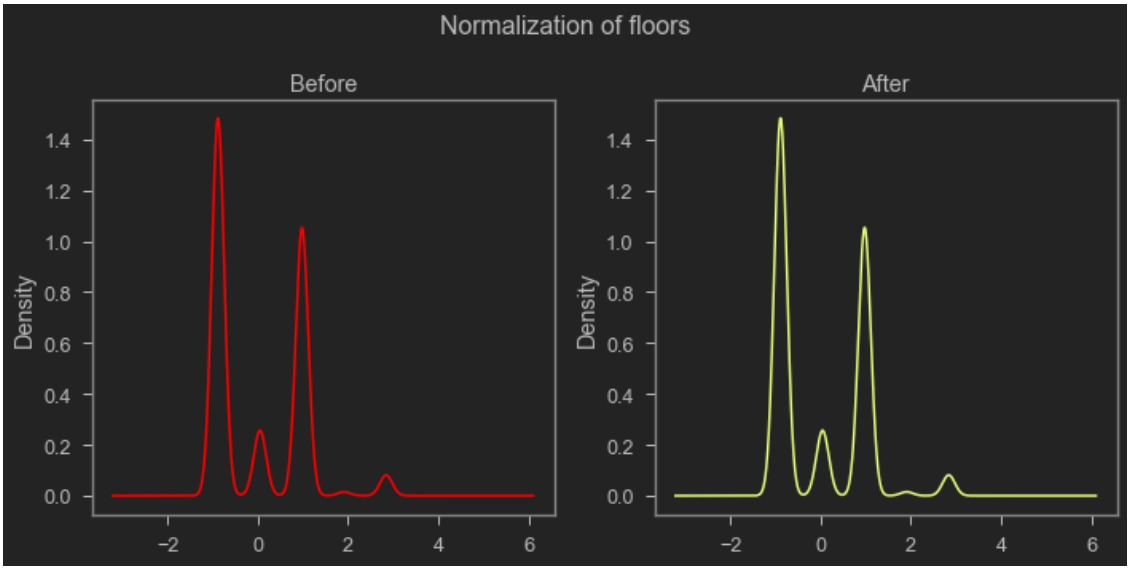


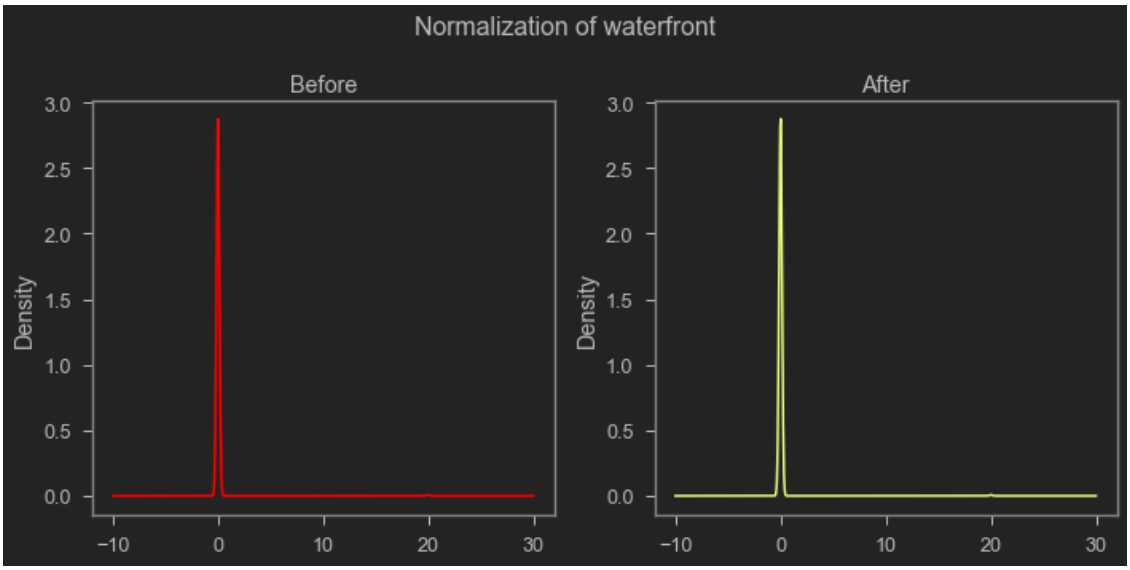



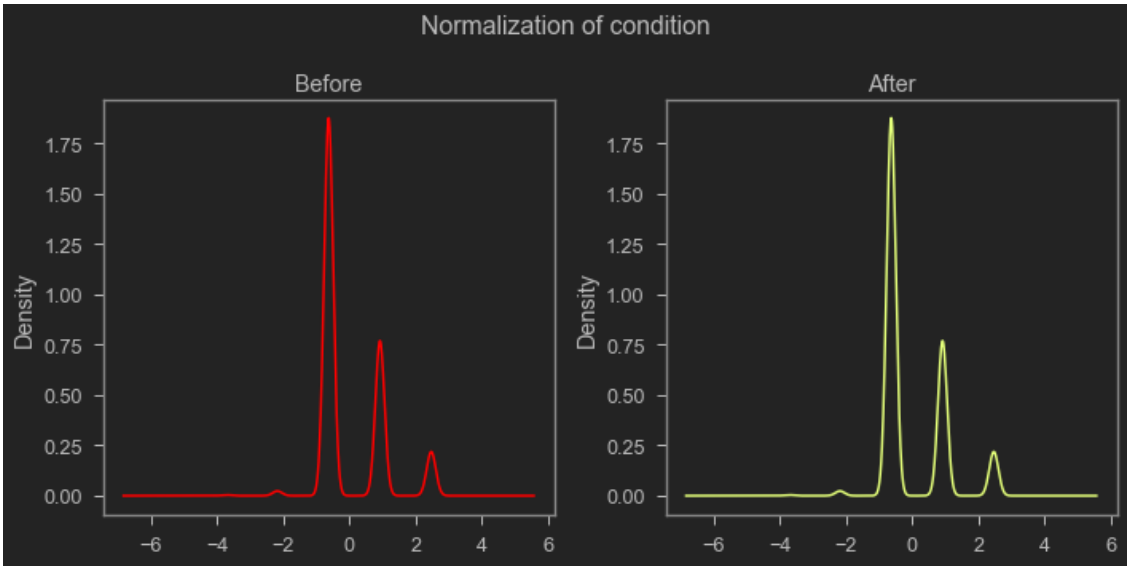

Normalization of bathrooms
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−5
0
5
10

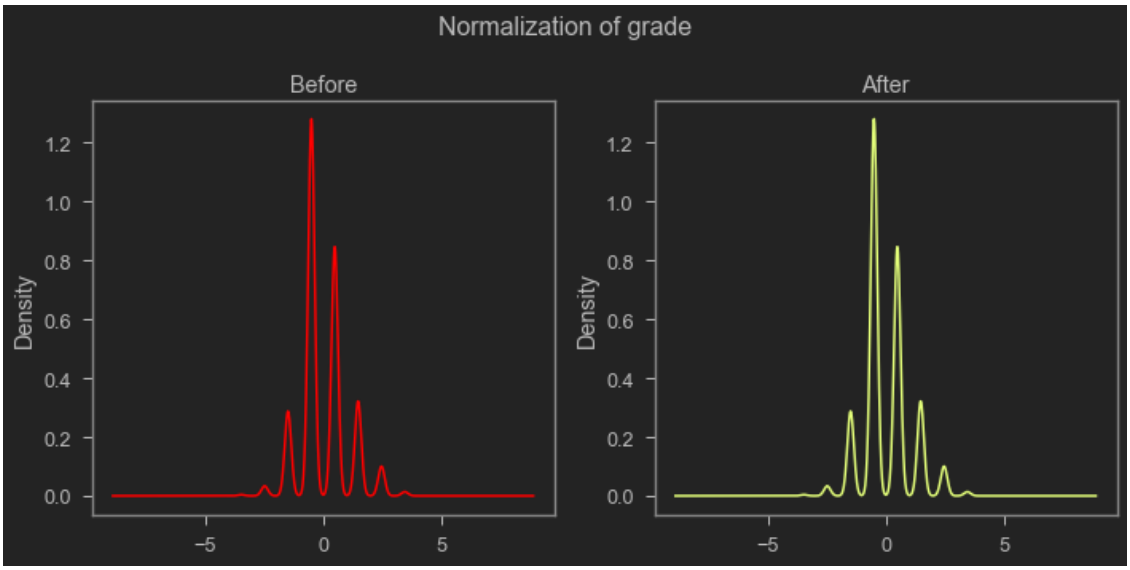

Normalization of sqft_living
Before
After
Density
0.0
0.1
0.2
0.3
0.4
−5
0
5
10

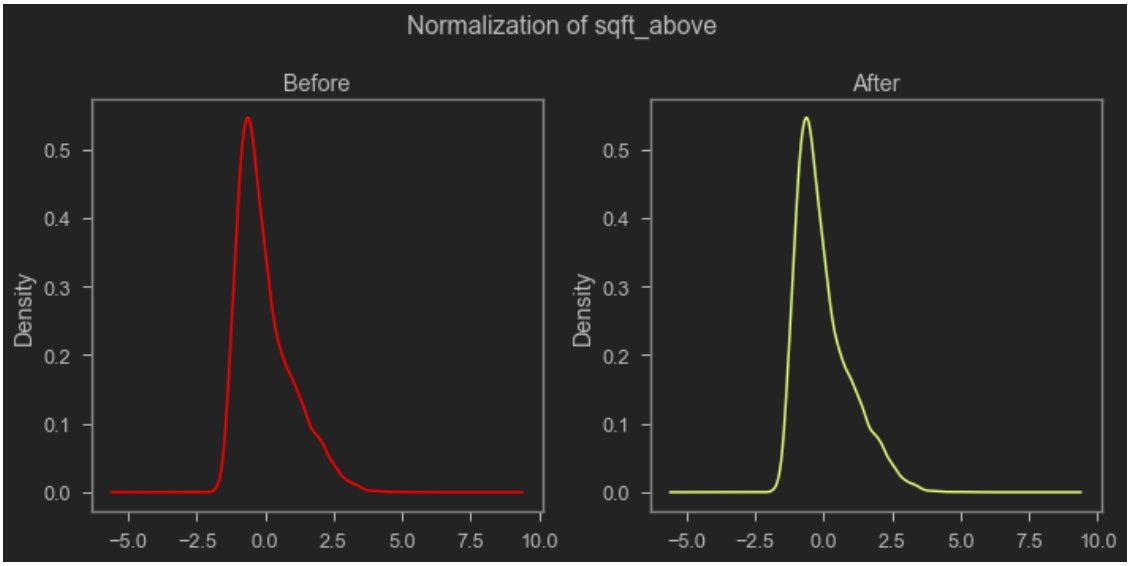

Normalization of sqft_lot
Before
After
Density
2.0
1.5
1.0
0.5
0.0
−20
0
20
40
60

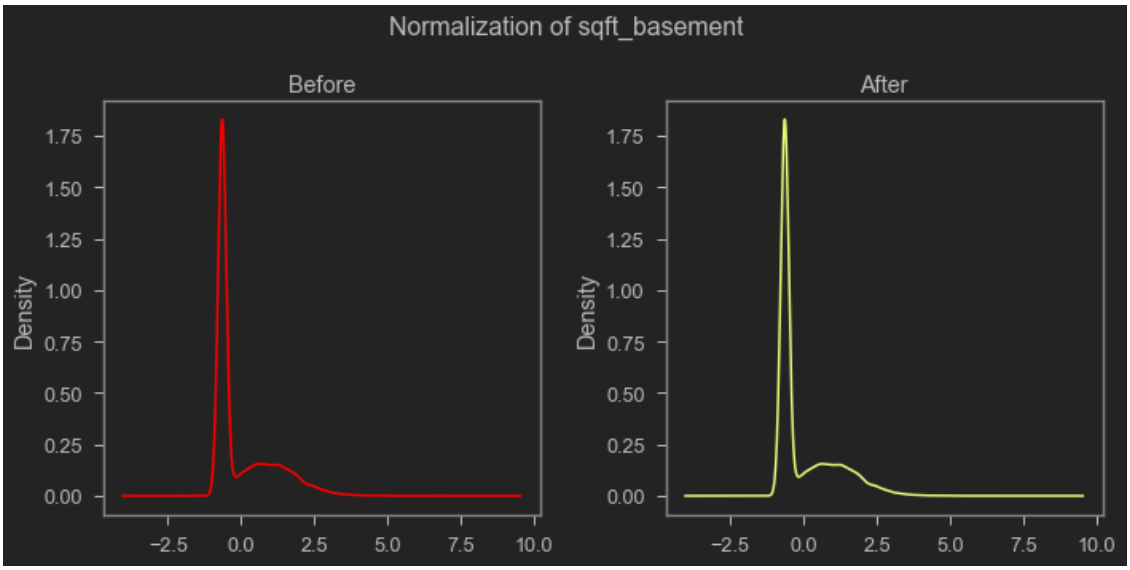

Normalization of floors
Before
After
Density
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−2
0
2
4
6

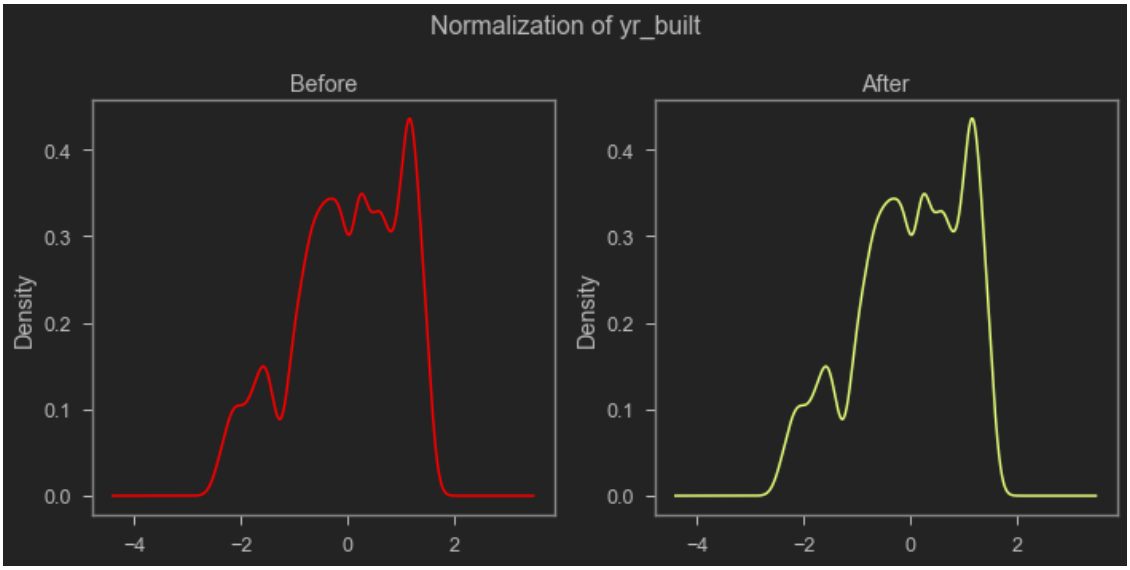

Normalization of waterfront
Before
After
Density
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−10
0
10
20
30

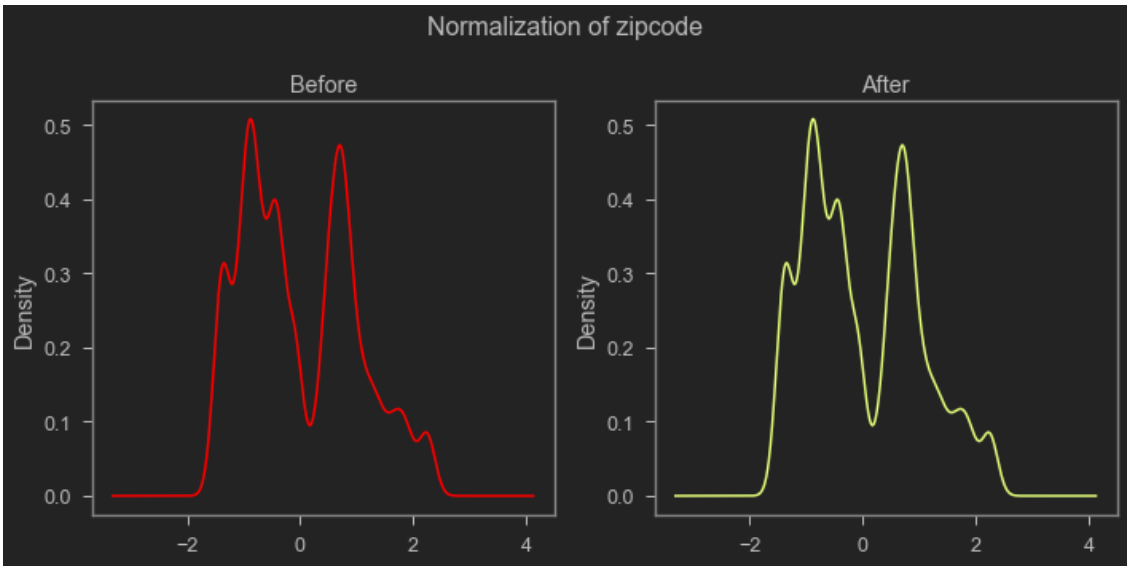

Normalization of condition
Before
After
Density
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−6
−4
−2
0
2
4
6

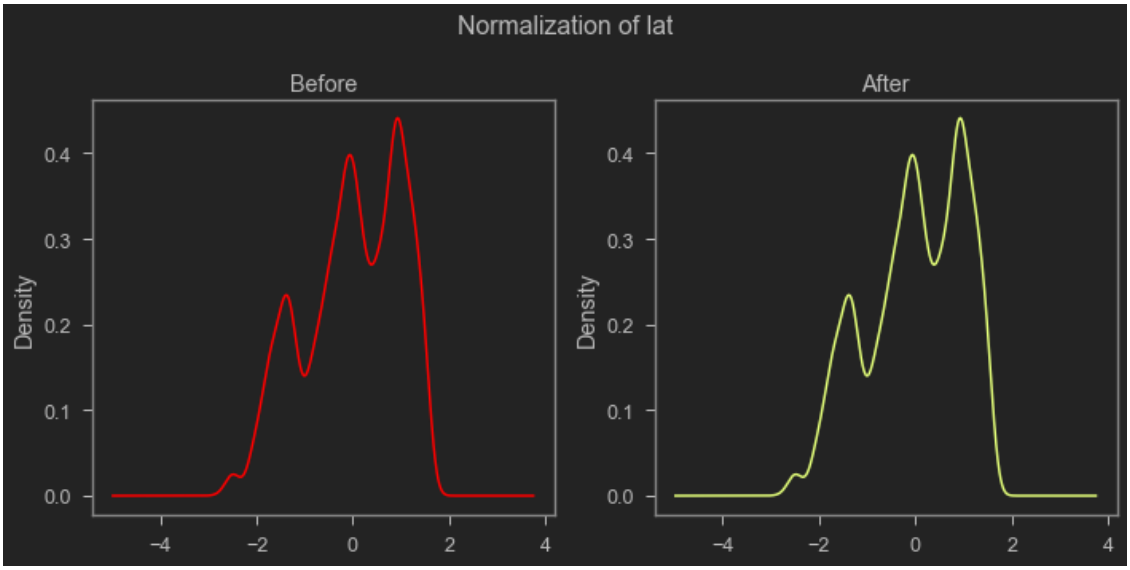

Normalization of grade
Before
After
Density
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−5
0
5

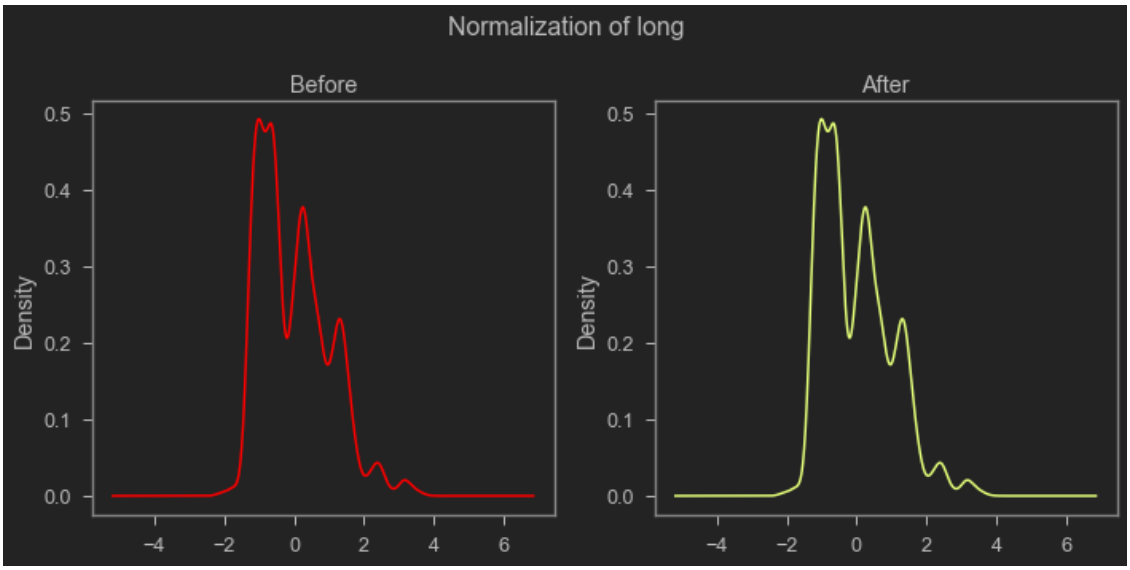

Normalization of sqft_above
Before
After
Density
0.5
0.4
0.3
0.2
0.1
0.0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

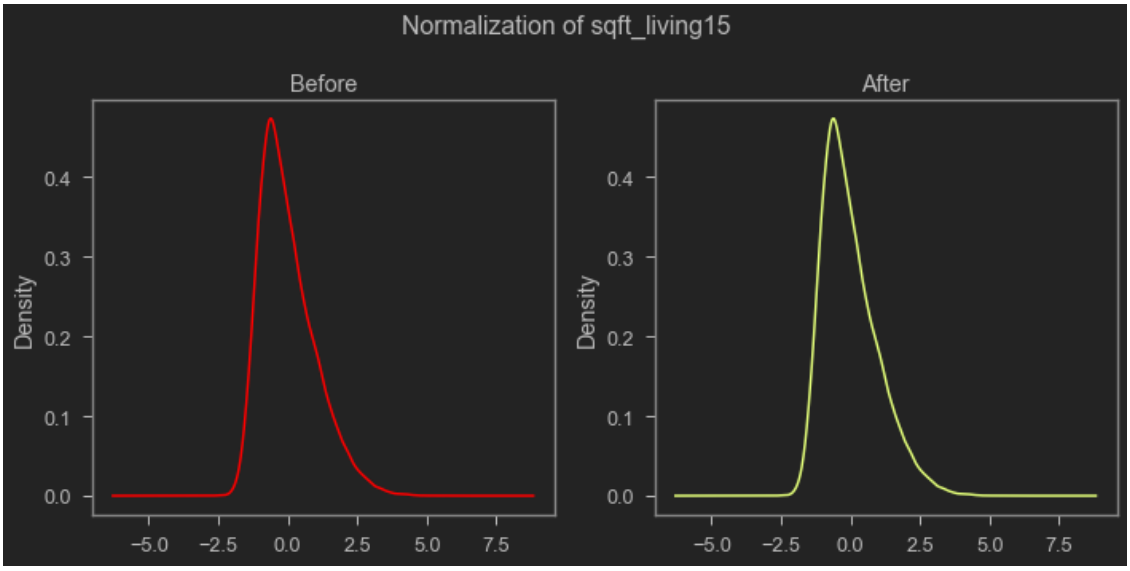

Normalization of sqft_basement
Before
After
Density
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−2.5
0.0
2.5
5.0
7.5
10.0

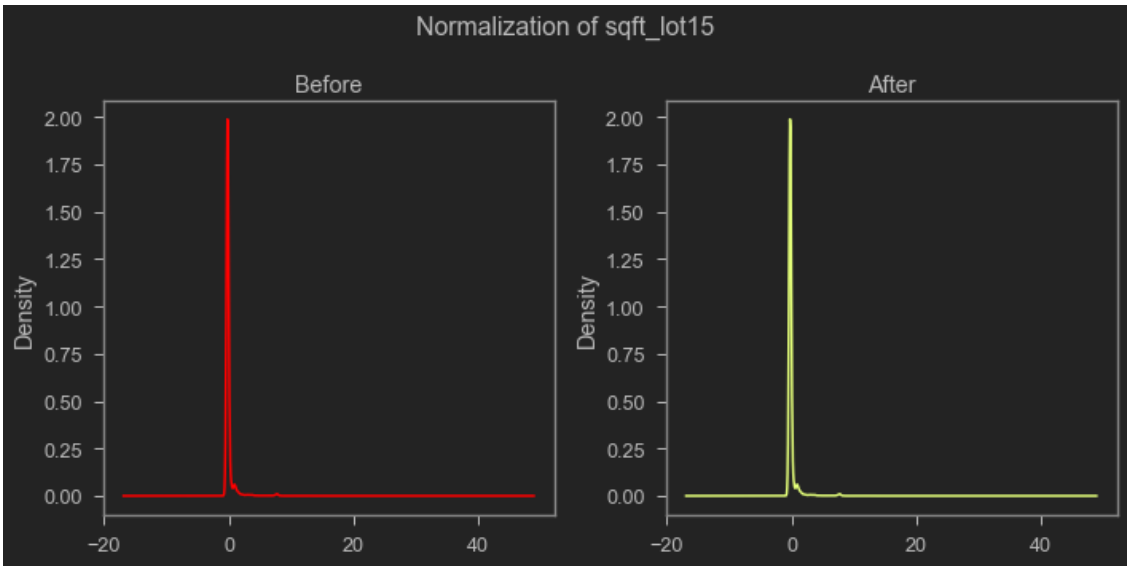

Normalization of yr_built
Before
After
Density
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2

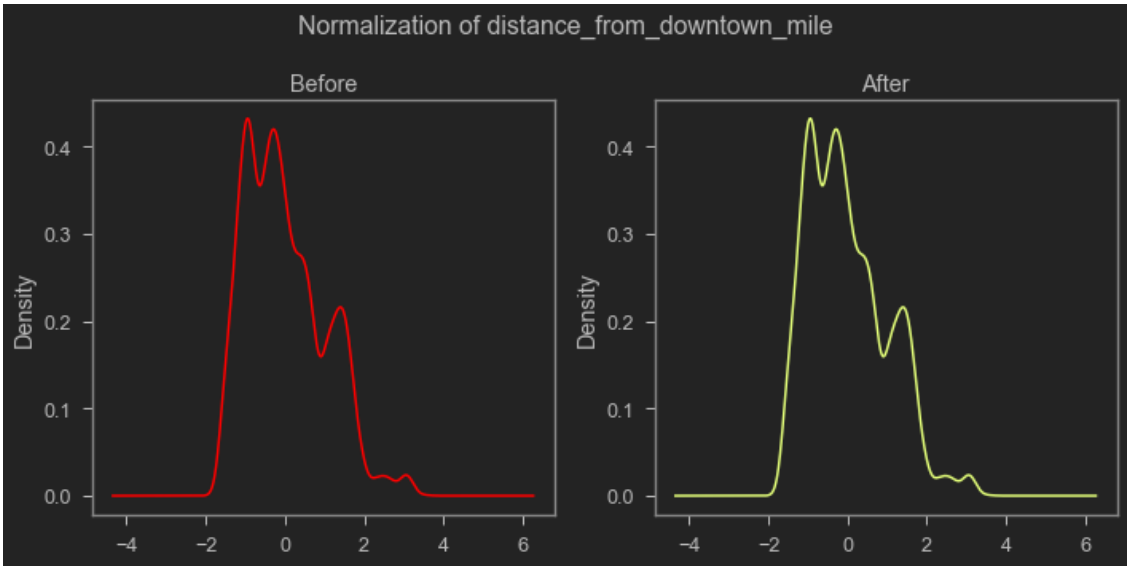

Normalization of zipcode
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
−2
0
2
4

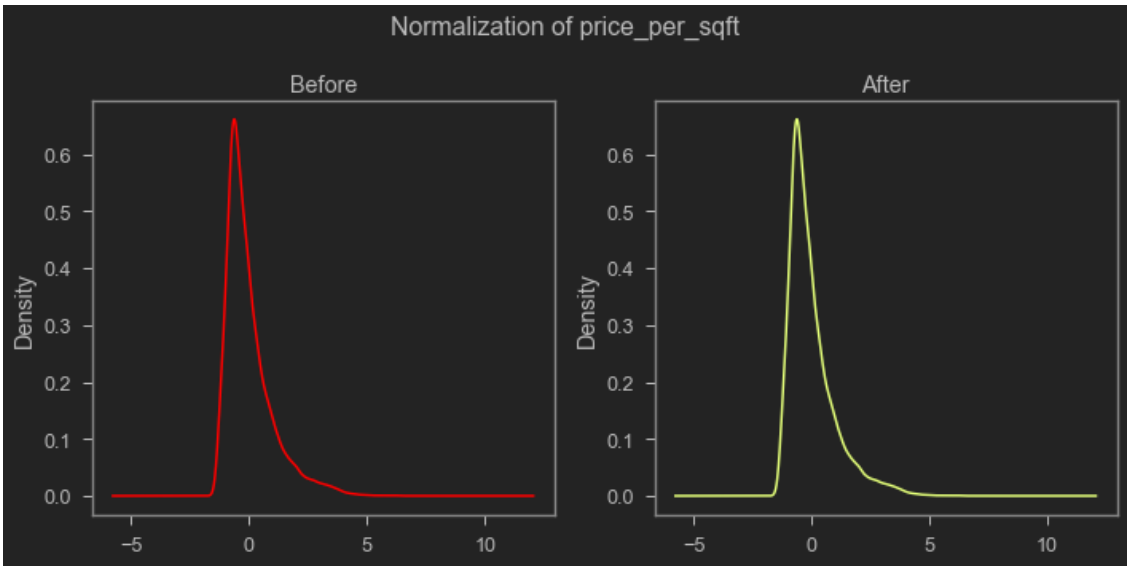

Normalization of lat
Before
After
Density
0.0
0.1
0.2
0.3
0.4
−4
−2
0
2
4

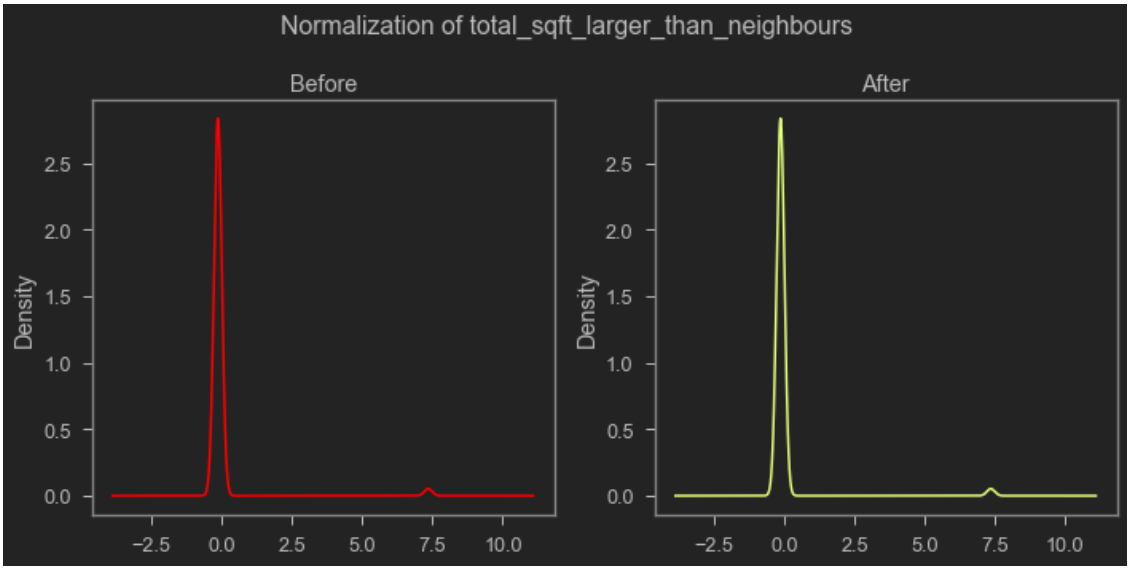

Normalization of long
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
−4
−2
0
2
4
6

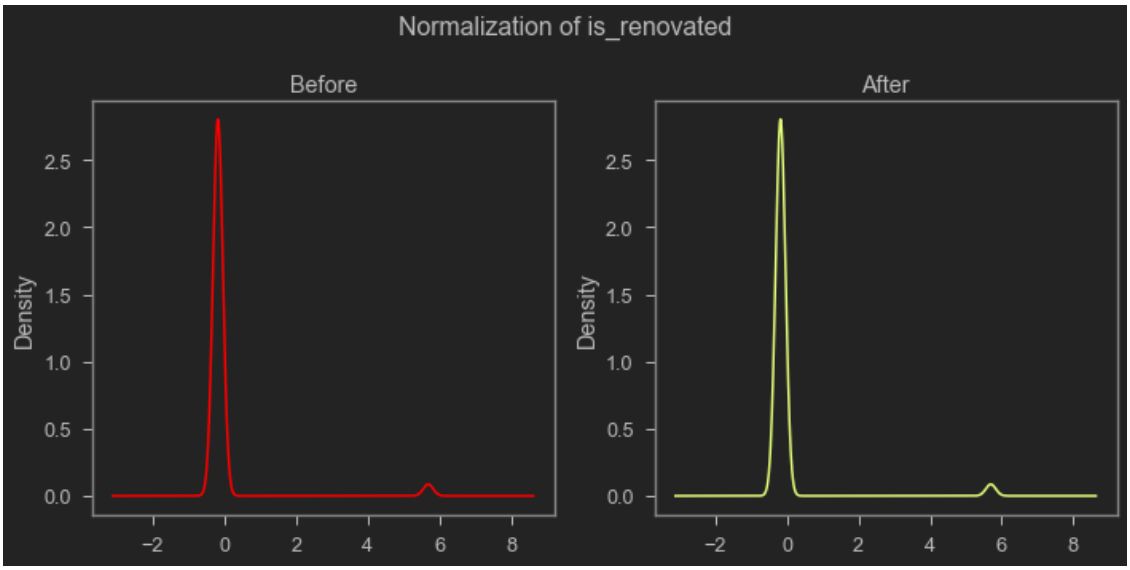

Normalization of sqft_living15
Before
After
Density
0.0
0.1
0.2
0.3
0.4
−5.0
−2.5
0.0
2.5
5.0
7.5


Normalization of sqft_lot15
Before
After
Density
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−20
0
20
40


Normalization of distance_from_downtown_mile
Before
After
Density
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4
6


Normalization of price_per_sqft
Before
After
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−5
0
5
10


Normalization of total_sqft_larger_than_neighbours
Before
After
Density
0.0
0.5
1.0
1.5
2.0
2.5
−2.5
0.0
2.5
5.0
7.5
10.0




[120]:
```
X_df_Z
```

[120]:
```
          price  bedrooms  …  total_sqft_larger_than_neighbours  is_renovated
0     -1.252728 -0.366484  …                          -0.136048     -0.175674
1      0.349503 -0.366484  …                          -0.136048      5.692364
2     -1.465108 -1.501272  …                          -0.136048     -0.175674
3      0.684040  0.768304  …                          -0.136048     -0.175674
4      0.207578 -0.366484  …                          -0.136048     -0.175674
…           …       …  …                                  …             …
19970 -0.552734 -0.366484  …                          -0.136048     -0.175674
19971 -0.349984  0.768304  …                          -0.136048     -0.175674
19972 -0.339335 -1.501272  …                          -0.136048     -0.175674
19973 -0.349984 -0.366484  …                          -0.136048     -0.175674
19974 -0.730140 -1.501272  …                          -0.136048     -0.175674

[19975 rows x 21 columns]
```

Scaling based on mean and IQR, it has better performance.

[121]:
```
df_model_2 = None
```

[122]:
```
df_model_2 = X_df_ro
```

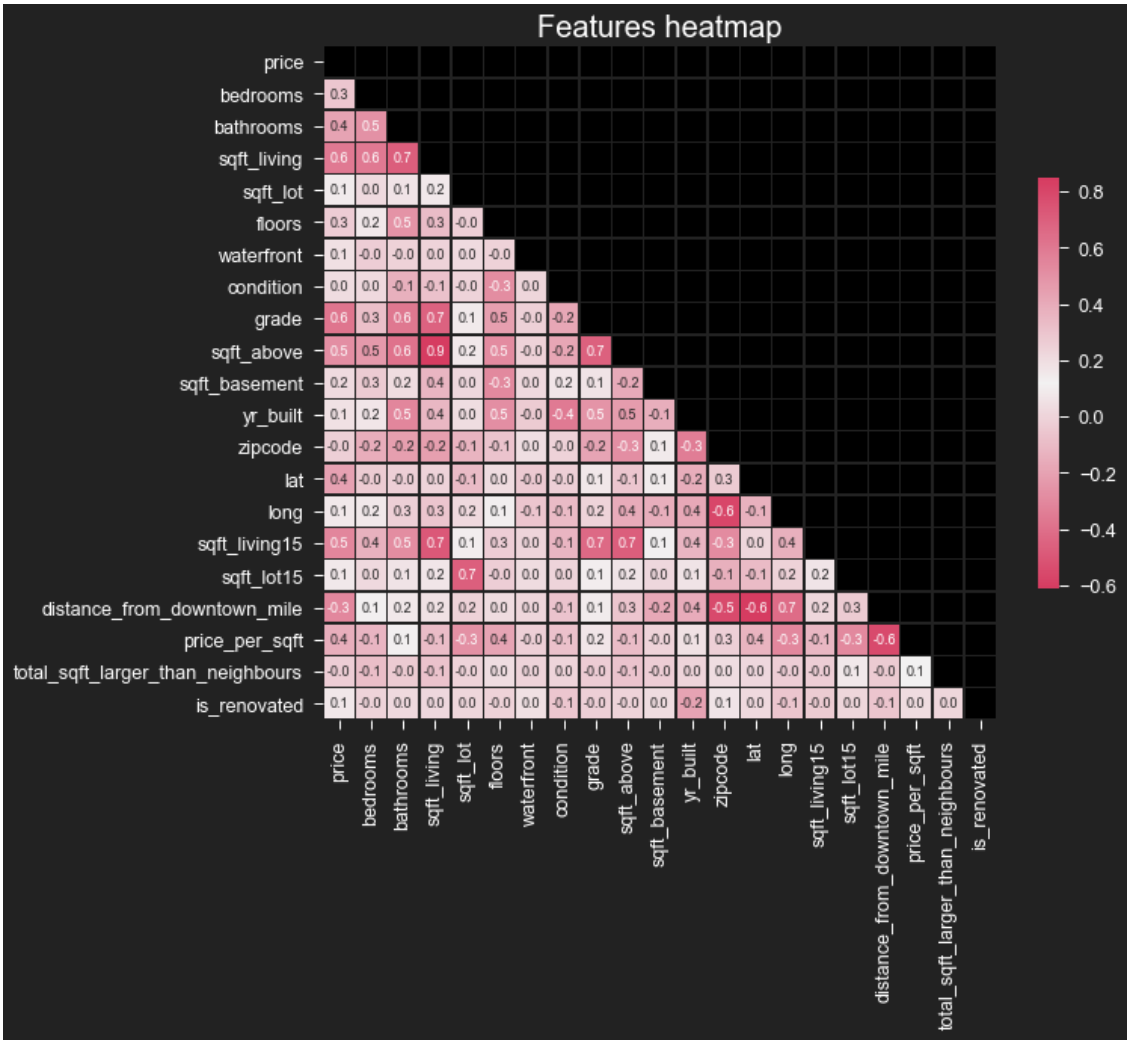
[123]:
```
df_model_2
```

[123]:
```
          price  bedrooms  …  total_sqft_larger_than_neighbours  is_renovated
0     -0.761071       0.0  …                                0.0           0.0
1      0.367857       0.0  …                                0.0           1.0
2     -0.910714      -1.0  …                                0.0           0.0
```

```
3       0.603571        1.0  …                                  0.0           0.0
4       0.267857        0.0  …                                  0.0           0.0
…            …          …    …                                  …             …
19970 -0.267857         0.0  …                                  0.0           0.0
19971 -0.125000         1.0  …                                  0.0           0.0
19972 -0.117496        -1.0  …                                  0.0           0.0
19973 -0.125000         0.0  …                                  0.0           0.0
19974 -0.392857        -1.0  …                                  0.0           0.0

[19975 rows x 21 columns]
```

### 4.10 Check for multicolinerity

[124]:
```
heatmap_DataFrame(df_model_2)
```



[125]:
```
correlation_top_bottom(df_model_2.corr())
```

```
Positive correlations:
   index                          feature_combo  correlation
0     72            sqft_living and sqft_above     0.850184
1     78         sqft_living and sqft_living15     0.731599
2    204          sqft_above and sqft_living15     0.712493
3     45             bathrooms and sqft_living     0.711160
4    100               sqft_lot and sqft_lot15     0.704609
5    177                  grade and sqft_above     0.701205
6     71                 grade and sqft_living     0.694137
7    183               grade and sqft_living15     0.662505
8    311  long and distance_from_downtown_mile     0.654559
9    371  distance_from_downtown_mile and long     0.654559

----------------------------------------------------------------------
Negative correlations:
   index                                    feature_combo  correlation
0    290             distance_from_downtown_mile and lat    -0.609103
1    266                                 zipcode and long    -0.566271
2    375  price_per_sqft and distance_from_downtown_mile    -0.550064
3    269         zipcode and distance_from_downtown_mile    -0.549053
4    158                           condition and yr_built    -0.355174
5    243                             zipcode and yr_built    -0.342231
6    312                          price_per_sqft and long    -0.304352
7     17           distance_from_downtown_mile and price    -0.304061
8    115                         sqft_basement and floors    -0.296901
9    354                   price_per_sqft and sqft_lot15    -0.296569
```

[126]:
```
def correlation_feat(df, threshold=0.75):
    feature_corr = set()  # Set of all the names of correlated columns
    corr_matrix = df.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]
                   ) > threshold:  # absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                feature_corr.add(colname)
    return feature_corr
```

[127]:
```
corr_features = correlation_feat(df_model_2, 0.75)
print('correlated features: ', len(set(corr_features)))
print('correlated features: ', corr_features)
```

```
correlated features:  1
correlated features:  {'sqft_above'}
```

[128]:
```
df_model_2.columns
```

[128]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'condition', 'grade', 'sqft_above', 'sqft_basement',
       'yr_built', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15',
       'distance_from_downtown_mile', 'price_per_sqft',
       'total_sqft_larger_than_neighbours', 'is_renovated'],
      dtype='object')
```

Dropping `sqft_above` from the data set. also dropping redundant location feature `lat` and `long` and `zipcode` and price_per_sqft. After getting impact of this, might drop it later.

[129]:
```
without_ppsq = df_model_2.
↪drop(columns=['sqft_above','lat','long','price_per_sqft','zipcode'])
```

[130]:
```
categorical_feat_ = ['waterfront', 'condition', 'grade']
numeric_feat = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
    'sqft_basement', 'yr_built', 'sqft_living15', 'sqft_lot15',
    'distance_from_downtown_mile', 'total_sqft_larger_than_neighbours',
    'is_renovated'
]
```

throwaway model

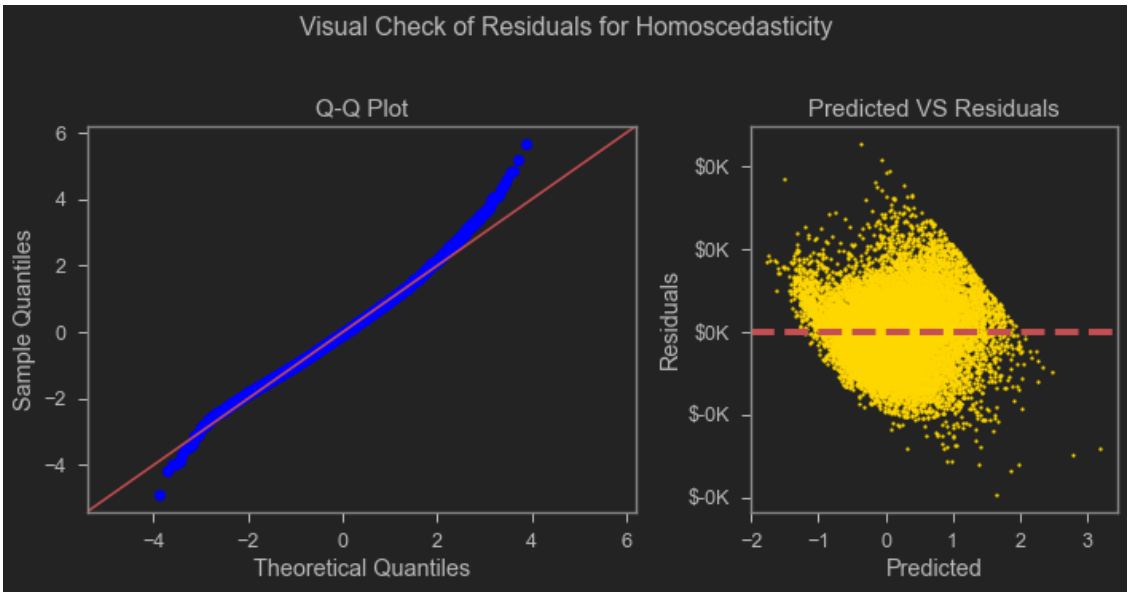
[131]:
```
OLS_sm(df=df_model_2,
       numeric_features=numeric_feat,
       categorical_features=categorical_feat_,show_plots=True,show_summary=True)
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_basement + yr_built + sqft_living15 + sqft_lot15 +
distance_from_downtown_mile + total_sqft_larger_than_neighbours + is_renovated +
C(waterfront) + C(condition) + C(grade)

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.674
Model:                            OLS   Adj. R-squared:                  0.674
Method:                 Least Squares   F-statistic:                     1586.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:42:38   Log-Likelihood:                -10157.
No. Observations:               19975   AIC:                         2.037e+04
Df Residuals:                   19948   BIC:                         2.058e+04
Df Model:                          26
Covariance Type:            nonrobust
==============================================================================================
                                    coef    std err          t      P>|t|    ␣
↪  [0.025      0.975]
----------------------------------------------------------------------------------------------
```

```
Intercept                          0.1046     0.411      0.255      0.799 ␣
 ↪ -0.700       0.910
C(waterfront)[T.1.0]               0.6684     0.057     11.675      0.000 ␣
 ↪  0.556       0.781
C(condition)[T.-1.0]               0.1642     0.084      1.953      0.051 ␣
 ↪ -0.001       0.329
C(condition)[T.0.0]                0.3398     0.078      4.343      0.000 ␣
 ↪  0.186       0.493
C(condition)[T.1.0]                0.4356     0.078      5.566      0.000 ␣
 ↪  0.282       0.589
C(condition)[T.2.0]                0.5431     0.079      6.897      0.000 ␣
 ↪  0.389       0.697
C(grade)[T.-3.0]                  -0.7826     0.410     -1.907      0.057 ␣
 ↪ -1.587       0.022
C(grade)[T.-2.0]                  -0.8246     0.404     -2.042      0.041 ␣
 ↪ -1.616      -0.033
C(grade)[T.-1.0]                  -0.7569     0.403     -1.878      0.060 ␣
 ↪ -1.547       0.033
C(grade)[T.0.0]                   -0.5029     0.403     -1.247      0.212 ␣
 ↪ -1.293       0.287
C(grade)[T.1.0]                   -0.2618     0.403     -0.649      0.516 ␣
 ↪ -1.052       0.529
C(grade)[T.2.0]                    0.0277     0.403      0.069      0.945 ␣
 ↪ -0.763       0.818
C(grade)[T.3.0]                    0.1828     0.404      0.453      0.651 ␣
 ↪ -0.609       0.974
C(grade)[T.4.0]                    0.2434     0.406      0.600      0.549 ␣
 ↪ -0.552       1.039
C(grade)[T.5.0]                   -0.1855     0.466     -0.398      0.691 ␣
 ↪ -1.099       0.729
bedrooms                          -0.0413     0.004     -9.595      0.000 ␣
 ↪ -0.050      -0.033
bathrooms                          0.0766     0.007     10.814      0.000 ␣
 ↪  0.063       0.091
sqft_living                        0.3082     0.009     34.357      0.000 ␣
 ↪  0.291       0.326
sqft_lot                           0.0065     0.001     12.144      0.000 ␣
 ↪  0.005       0.008
floors                             0.0229     0.008      2.935      0.003 ␣
 ↪  0.008       0.038
sqft_basement                     -0.0509     0.005    -10.331      0.000 ␣
 ↪ -0.061      -0.041
yr_built                          -0.1656     0.007    -23.608      0.000 ␣
 ↪ -0.179      -0.152
sqft_living15                      0.1983     0.006     32.638      0.000 ␣
 ↪  0.186       0.210
```

```
sqft_lot15                          0.0033      0.001      4.393      0.000
 ↪    0.002       0.005
distance_from_downtown_mile        -0.4363      0.005    -81.870      0.000
 ↪   -0.447      -0.426
total_sqft_larger_than_neighbours  -0.0291      0.022     -1.339      0.180
 ↪   -0.072       0.013
is_renovated                        0.1314      0.018      7.457      0.000
 ↪    0.097       0.166
==============================================================================
Omnibus:                      472.315   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              582.233
Skew:                           0.313   Prob(JB):                    3.71e-127
Kurtosis:                       3.554   Cond. No.                     4.01e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
 ↪specified.
[2] The condition number is large, 4.01e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[131]: `<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1c24e9ceb20>`




throwaway model

[132]:
```
OLS_sm(df=df_model_2,
       numeric_features=numeric_feat+['price_per_sqft'],
```

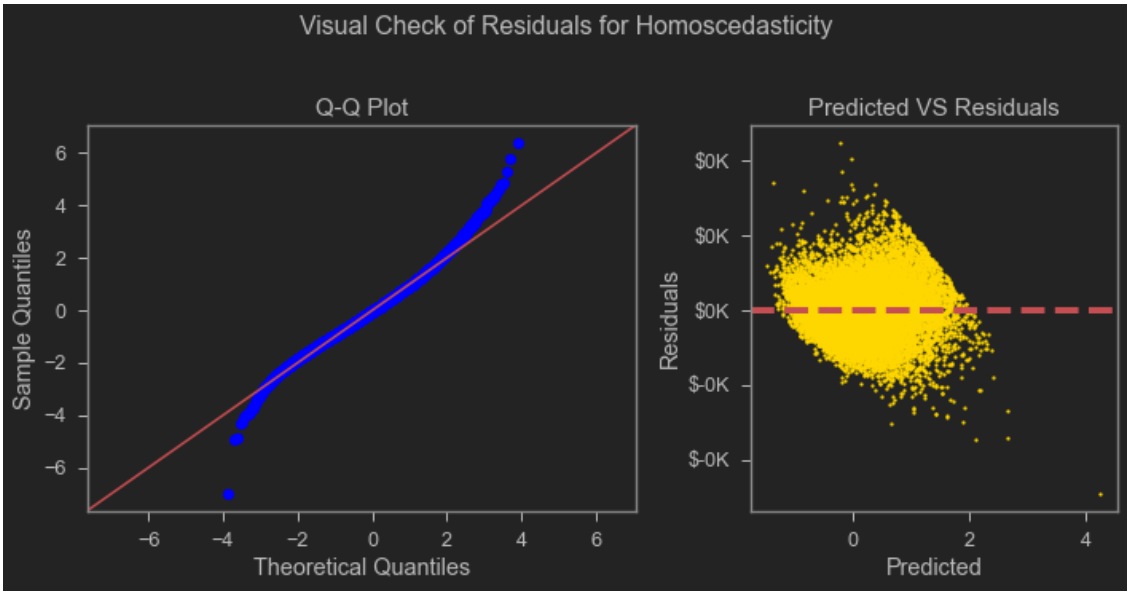
```
        categorical_features=categorical_feat_,show_plots=True,show_summary=True)
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_basement + yr_built + sqft_living15 + sqft_lot15 +
distance_from_downtown_mile + total_sqft_larger_than_neighbours + is_renovated +
price_per_sqft + C(waterfront) + C(condition) + C(grade)

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.752
Model:                            OLS   Adj. R-squared:                  0.752
Method:                 Least Squares   F-statistic:                     2244.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:42:39   Log-Likelihood:                -7411.0
No. Observations:               19975   AIC:                         1.488e+04
Df Residuals:                   19947   BIC:                         1.510e+04
Df Model:                          27
Covariance Type:            nonrobust
================================================================================================
                                     coef    std err          t      P>|t|    ␣
 ↪  [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                         0.2214      0.358      0.619      0.536    ␣
 ↪  -0.480       0.923
C(waterfront)[T.1.0]              0.7311      0.050     14.650      0.000    ␣
 ↪   0.633       0.829
C(condition)[T.-1.0]              0.1174      0.073      1.602      0.109    ␣
 ↪  -0.026       0.261
C(condition)[T.0.0]               0.2105      0.068      3.086      0.002    ␣
 ↪   0.077       0.344
C(condition)[T.1.0]               0.3092      0.068      4.531      0.000    ␣
 ↪   0.175       0.443
C(condition)[T.2.0]               0.3712      0.069      5.406      0.000    ␣
 ↪   0.237       0.506
C(grade)[T.-3.0]                 -0.5157      0.358     -1.442      0.149    ␣
 ↪  -1.217       0.185
C(grade)[T.-2.0]                 -0.6174      0.352     -1.754      0.079    ␣
 ↪  -1.307       0.073
C(grade)[T.-1.0]                 -0.5734      0.351     -1.632      0.103    ␣
 ↪  -1.262       0.115
C(grade)[T.0.0]                  -0.4281      0.351     -1.218      0.223    ␣
 ↪  -1.117       0.261
C(grade)[T.1.0]                  -0.2813      0.351     -0.800      0.424    ␣
 ↪  -0.970       0.408
```

```
C(grade)[T.2.0]                      -0.0577      0.352     -0.164      0.870
 ↪  -0.747       0.632
C(grade)[T.3.0]                       0.0972      0.352      0.276      0.782
 ↪  -0.593       0.787
C(grade)[T.4.0]                       0.1277      0.354      0.361      0.718
 ↪  -0.566       0.821
C(grade)[T.5.0]                      -0.3036      0.406     -0.747      0.455
 ↪  -1.100       0.493
bedrooms                             -0.0067      0.004     -1.774      0.076
 ↪  -0.014       0.001
bathrooms                             0.0315      0.006      5.072      0.000
 ↪   0.019       0.044
sqft_living                           0.4568      0.008     56.824      0.000
 ↪   0.441       0.473
sqft_lot                              0.0082      0.000     17.693      0.000
 ↪   0.007       0.009
floors                               -0.2504      0.008    -32.823      0.000
 ↪  -0.265      -0.235
sqft_basement                        -0.1200      0.004    -27.384      0.000
 ↪  -0.129      -0.111
yr_built                             -0.1682      0.006    -27.517      0.000
 ↪  -0.180      -0.156
sqft_living15                         0.2187      0.005     41.236      0.000
 ↪   0.208       0.229
sqft_lot15                            0.0088      0.001     13.395      0.000
 ↪   0.007       0.010
distance_from_downtown_mile          -0.2294      0.005    -43.062      0.000
 ↪  -0.240      -0.219
total_sqft_larger_than_neighbours    -0.1927      0.019    -10.110      0.000
 ↪  -0.230      -0.155
is_renovated                          0.1103      0.015      7.179      0.000
 ↪   0.080       0.140
price_per_sqft                        0.3357      0.004     79.445      0.000
 ↪   0.327       0.344
==============================================================================
Omnibus:                      730.276   Durbin-Watson:                   1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1274.369
Skew:                           0.309   Prob(JB):                    1.88e-277
Kurtosis:                       4.072   Cond. No.                     4.02e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
 ↪specified.
[2] The condition number is large, 4.02e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

"""

[132]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1c258300d60>




`price_per_sqft` is leaking information, thus dropping it for now.

[133]:
```
df_model_2 = without_ppsq.copy()
```

[134]:
```
df_model_2
```

[134]:

| | price | bedrooms | … | total_sqft_larger_than_neighbours | is_renovated |
|---|---|---|---|---|---|
| 0 | -0.761071 | 0.0 | … | 0.0 | 0.0 |
| 1 | 0.367857 | 0.0 | … | 0.0 | 1.0 |
| 2 | -0.910714 | -1.0 | … | 0.0 | 0.0 |
| 3 | 0.603571 | 1.0 | … | 0.0 | 0.0 |
| 4 | 0.267857 | 0.0 | … | 0.0 | 0.0 |
| … | … | … | … | … | … |
| 19970 | -0.267857 | 0.0 | … | 0.0 | 0.0 |
| 19971 | -0.125000 | 1.0 | … | 0.0 | 0.0 |
| 19972 | -0.117496 | -1.0 | … | 0.0 | 0.0 |
| 19973 | -0.125000 | 0.0 | … | 0.0 | 0.0 |
| 19974 | -0.392857 | -1.0 | … | 0.0 | 0.0 |

[19975 rows x 16 columns]

[135]:
```
correlation_top_bottom(df_model_2.corr())
```

```
Positive correlations:
   index                  feature_combo  correlation
```

```
0     59  sqft_living and sqft_living15     0.731599
1     35      bathrooms and sqft_living     0.711160
2     76        sqft_lot and sqft_lot15     0.704609
3     56          grade and sqft_living     0.694137
4    139        grade and sqft_living15     0.662505
5      8                 grade and price     0.618400
6     40            grade and bathrooms     0.605563
7     19       bedrooms and sqft_living     0.604535
8      3          sqft_living and price     0.604388
9     42         bathrooms and yr_built     0.549182

----------------------------------------------------------------------
Negative correlations:
   index                                 feature_combo  correlation
0    122                       condition and yr_built    -0.355174
1     13         distance_from_downtown_mile and price    -0.304061
2     89                       sqft_basement and floors    -0.296901
3     87                          condition and floors    -0.281505
4    175                     is_renovated and yr_built    -0.196537
5    120                           grade and condition    -0.178234
6    157  sqft_basement and distance_from_downtown_mile    -0.175746
7    154                     sqft_basement and yr_built    -0.149454
8     39                        bathrooms and condition    -0.142958
9    123                   condition and sqft_living15    -0.123776
```

# 5 Feature Selection

## 5.1 Exploratory Simple OLS a.k.a. Univariate OLS or ANNOVA

On all data

Dependent variable is price. A list containing all of the features were regressed. That list is:

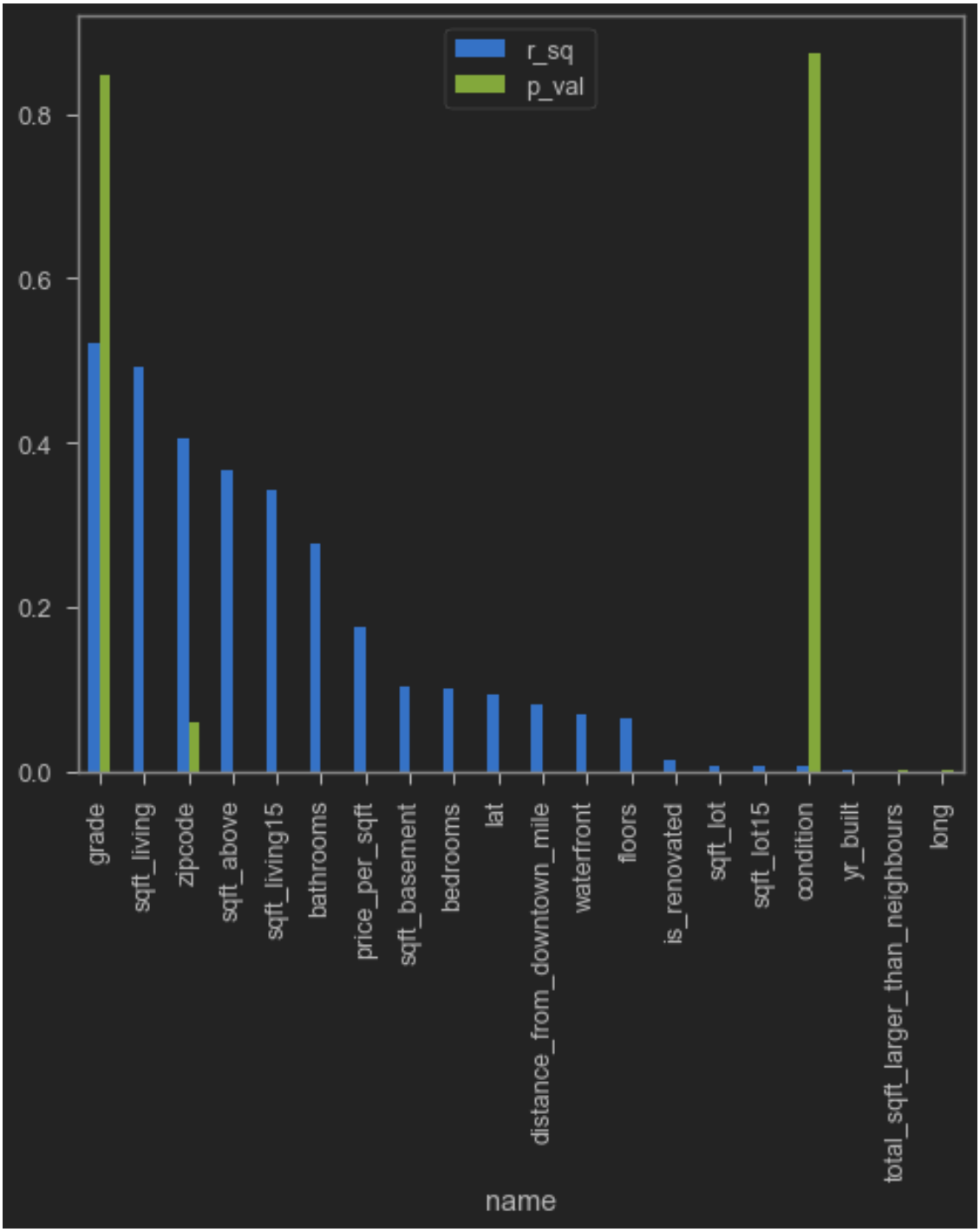
[136]:
```
df_model.columns
```

[136]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'condition', 'grade', 'sqft_above', 'sqft_basement',
       'yr_built', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15',
       'distance_from_downtown_mile', 'price_per_sqft',
       'total_sqft_larger_than_neighbours', 'is_renovated'],
      dtype='object')
```

[137]:
```
# Data from calucation for checking normality assumption code
df_stat = pd.DataFrame(stat_list).set_index('name')
df_stat.sort_values(by='r_sq',ascending=False).style.set_precision(4).
 ↪set_properties(**{'color': 'lawngreen'})
```

[137]:
```
<pandas.io.formats.style.Styler at 0x1c24e9cdc70>
```

[138]:
```
for_plot_df=df_stat.sort_values(by='r_sq',ascending=False)[1:]
ax = for_plot_df[['r_sq','p_val']].plot(kind='bar',rot=90)
ax.set_xticks(np.arange(len(for_plot_df)))
ax.set_autoscalex_on(False)
```



## 5.2 Exploratory Multiple OLS a.k.a Multivariate OLS

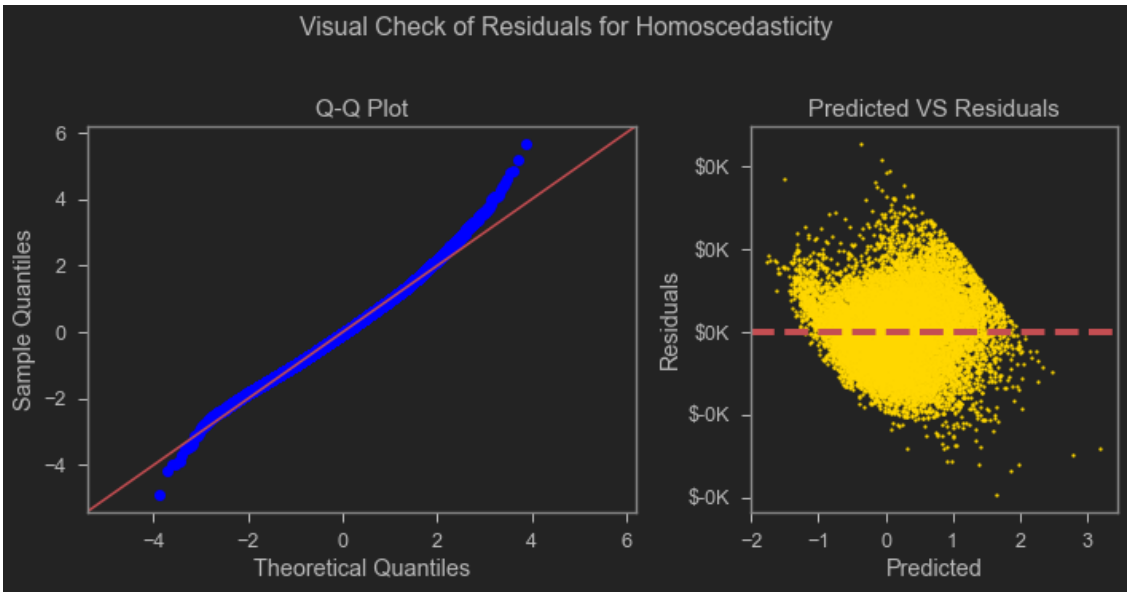
[139]:
```
OLS_sm(df=df_model_2,
       numeric_features=[
           'bedrooms',
           'bathrooms',
           'sqft_living',
           'sqft_lot',
           'floors',
           'sqft_basement',
           'yr_built',
           'sqft_living15',
           'sqft_lot15',
           'distance_from_downtown_mile',
       ],
       dependant_var='price',
       categorical_features=[
           'waterfront', 'condition', 'grade', 'is_renovated',
           'total_sqft_larger_than_neighbours'
       ],
       verbose=False)
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_basement + yr_built + sqft_living15 + sqft_lot15 +
distance_from_downtown_mile + C(waterfront) + C(condition) + C(grade) +
C(is_renovated) + C(total_sqft_larger_than_neighbours)

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.674
Model:                            OLS   Adj. R-squared:                  0.674
Method:                 Least Squares   F-statistic:                     1586.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:42:41   Log-Likelihood:                -10157.
No. Observations:               19975   AIC:                         2.037e+04
Df Residuals:                   19948   BIC:                         2.058e+04
Df Model:                          26
Covariance Type:            nonrobust
================================================================================================
                                                 coef    std err          t      ␣
 ↪ P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                                      0.1046      0.411      0.255      ␣
 ↪ 0.799      -0.700       0.910
C(waterfront)[T.1.0]                           0.6684      0.057     11.675      ␣
 ↪ 0.000       0.556       0.781
```

```
C(condition)[T.-1.0]                                0.1642      0.084      1.953  ␣
 ↪ 0.051      -0.001       0.329
C(condition)[T.0.0]                                 0.3398      0.078      4.343  ␣
 ↪ 0.000       0.186       0.493
C(condition)[T.1.0]                                 0.4356      0.078      5.566  ␣
 ↪ 0.000       0.282       0.589
C(condition)[T.2.0]                                 0.5431      0.079      6.897  ␣
 ↪ 0.000       0.389       0.697
C(grade)[T.-3.0]                                   -0.7826      0.410     -1.907  ␣
 ↪ 0.057      -1.587       0.022
C(grade)[T.-2.0]                                   -0.8246      0.404     -2.042  ␣
 ↪ 0.041      -1.616      -0.033
C(grade)[T.-1.0]                                   -0.7569      0.403     -1.878  ␣
 ↪ 0.060      -1.547       0.033
C(grade)[T.0.0]                                    -0.5029      0.403     -1.247  ␣
 ↪ 0.212      -1.293       0.287
C(grade)[T.1.0]                                    -0.2618      0.403     -0.649  ␣
 ↪ 0.516      -1.052       0.529
C(grade)[T.2.0]                                     0.0277      0.403      0.069  ␣
 ↪ 0.945      -0.763       0.818
C(grade)[T.3.0]                                     0.1828      0.404      0.453  ␣
 ↪ 0.651      -0.609       0.974
C(grade)[T.4.0]                                     0.2434      0.406      0.600  ␣
 ↪ 0.549      -0.552       1.039
C(grade)[T.5.0]                                    -0.1855      0.466     -0.398  ␣
 ↪ 0.691      -1.099       0.729
C(is_renovated)[T.1.0]                              0.1314      0.018      7.457  ␣
 ↪ 0.000       0.097       0.166
C(total_sqft_larger_than_neighbours)[T.1.0]        -0.0291      0.022     -1.339  ␣
 ↪ 0.180      -0.072       0.013
bedrooms                                           -0.0413      0.004     -9.595  ␣
 ↪ 0.000      -0.050      -0.033
bathrooms                                           0.0766      0.007     10.814  ␣
 ↪ 0.000       0.063       0.091
sqft_living                                         0.3082      0.009     34.357  ␣
 ↪ 0.000       0.291       0.326
sqft_lot                                            0.0065      0.001     12.144  ␣
 ↪ 0.000       0.005       0.008
floors                                              0.0229      0.008      2.935  ␣
 ↪ 0.003       0.008       0.038
sqft_basement                                      -0.0509      0.005    -10.331  ␣
 ↪ 0.000      -0.061      -0.041
yr_built                                           -0.1656      0.007    -23.608  ␣
 ↪ 0.000      -0.179      -0.152
sqft_living15                                       0.1983      0.006     32.638  ␣
 ↪ 0.000       0.186       0.210
```

```
sqft_lot15                                    0.0033      0.001      4.393
 ↪ 0.000       0.002       0.005
distance_from_downtown_mile                  -0.4363      0.005    -81.870
 ↪ 0.000      -0.447      -0.426
==============================================================================
Omnibus:                      472.315   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              582.233
Skew:                           0.313   Prob(JB):                    3.71e-127
Kurtosis:                       3.554   Cond. No.                     4.01e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
 ↪specified.
[2] The condition number is large, 4.01e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[139]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1c24e7c3190>




### 5.3 One Hot Encoding

[140]:
```
categorical_features=[
            'waterfront', 'condition', 'grade', 'is_renovated',
            'total_sqft_larger_than_neighbours']
```

[141]:
```
df_model_processed_ohe = pd.get_dummies(df_model_2,
                                        columns=[
                                            'waterfront', 'condition', 'grade',
                                            'is_renovated',
                                            'total_sqft_larger_than_neighbours'
                                        ],
                                        sparse=False,
                                        drop_first=True)
df_model_processed_ohe
```

[141]:

| | price | … | total_sqft_larger_than_neighbours_1.0 |
|---|---|---|---|
| 0 | -0.761071 | … | 0 |
| 1 | 0.367857 | … | 0 |
| 2 | -0.910714 | … | 0 |
| 3 | 0.603571 | … | 0 |
| 4 | 0.267857 | … | 0 |
| … | … | … | … |
| 19970 | -0.267857 | … | 0 |
| 19971 | -0.125000 | … | 0 |
| 19972 | -0.117496 | … | 0 |
| 19973 | -0.125000 | … | 0 |
| 19974 | -0.392857 | … | 0 |

[19975 rows x 27 columns]

## 5.4 Feature selection

### 5.4.1 Filter methods

Already applied filtering by correlation.

### 5.4.2 Wrapper methods

**Forward Selection using Statsmodels** Using this a guideline.

[142]:
```
def forward_selected(data, response):
    """
    Source: https://planspace.org/20150423-forward_selection_with_statsmodels/
    ---------------------------------------------------------------------------

    Linear model designed by forward selection.

    Parameters:
    -----------
    data : pandas DataFrame with all possible predictors and response.
    response: string, name of response column in data.

    Returns:
```

```
    --------
    model: an "optimal" fitted statsmodels linear model
           with an intercept
           selected by forward selection
           evaluated by adjusted R-squared
    """
    remaining = set(data.columns)
    remaining.remove(response)
    selected = []
    current_score, best_new_score = 0.0, 0.0
    while remaining and current_score == best_new_score:
        r_sq_scores_with_candidates = []
        for candidate in remaining:
            formula = "{} ~ {} + 1".format(response,
                                           ' + '.join(selected + [candidate]))
            score = smf.ols(formula, data).fit().rsquared_adj
            r_sq_scores_with_candidates.append((score, candidate))

        r_sq_scores_with_candidates.sort()
        best_new_score, best_candidate = r_sq_scores_with_candidates.pop()
        if current_score < best_new_score:
            remaining.remove(best_candidate)
            selected.append(best_candidate)
            current_score = best_new_score
    formula = "{} ~ {} + 1".format(response,
                                   ' + '.join(selected))
    model = smf.ols(formula, data).fit()

    # plotting
    # plot 1
    fig, (ax1,
          ax2) = plt.subplots(ncols=2,
                              figsize=(10, 5),
                              gridspec_kw={'width_ratios': [0.6, 0.4]})
    sm.qqplot(model.resid,
              dist=stats.norm,
              line='45',
              fit=True,
              ax=ax1)
    ax1.set_title('Q-Q Plot', fontdict={"size": 15})
    # plot 2

    ax2.scatter(x=model.fittedvalues,
                y=model.resid,
                s=4,
                color='gold')
    ax2.set(xlabel='Predicted', ylabel='Reseduals')
```

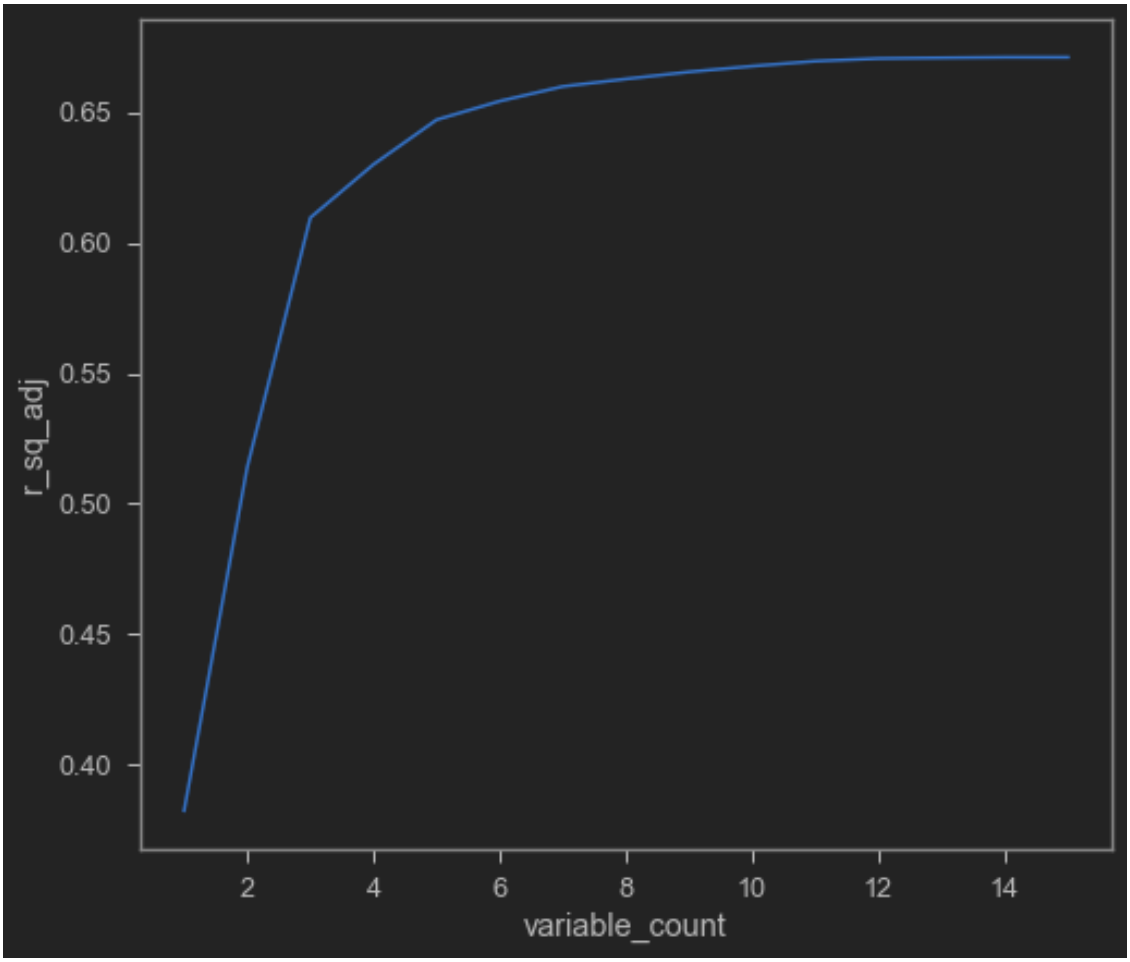
```
    ax2.axhline(y=0, c='r', lw=4, ls='--')
    ax2.set_title('Predicted VS Reseduals', fontdict={"size": 15})
    plt.suptitle('Visual Check of Reseduals for Homoscedasticity',
                 ha='center',
                 va='bottom',
                 fontdict={"size": 25})
    plt.tight_layout()
    return model
```

[143]:
```
def catch_forward_selected_steps(data, response):
    """
    Optimal feature selection helper formula.

    modified from
    Source: https://planspace.org/20150423-forward_selection_with_statsmodels/
    ----------------------------------------------------------------------------
    """
    remaining = set(data.columns)
    remaining.remove(response)
    selected = []
    current_score, best_new_score = 0.0, 0.0
    li = []
    n =0
    while remaining and current_score == best_new_score:
        r_sq_scores_with_candidates = []
        for candidate in remaining:
            formula = "{} ~ {} + 1".format(response,
                                           ' + '.join(selected + [candidate]))
            score = smf.ols(formula, data).fit().rsquared_adj
            r_sq_scores_with_candidates.append((score, candidate))
        r_sq_scores_with_candidates.sort()
        best_new_score, best_candidate = r_sq_scores_with_candidates.pop()
        if current_score < best_new_score:
            remaining.remove(best_candidate)
            selected.append(best_candidate)
            current_score = best_new_score
            n+=1
            tem_d = {'variable_count': n,'r_sq_adj':current_score}
            li.append(tem_d)
    return li
```

[144]:
```
# Without OHE, this is wrong approach
catch = pd.DataFrame(catch_forward_selected_steps(df_model_2, 'price'))
sns.lineplot(x='variable_count',y='r_sq_adj',data=catch);
catch
```

[144]:

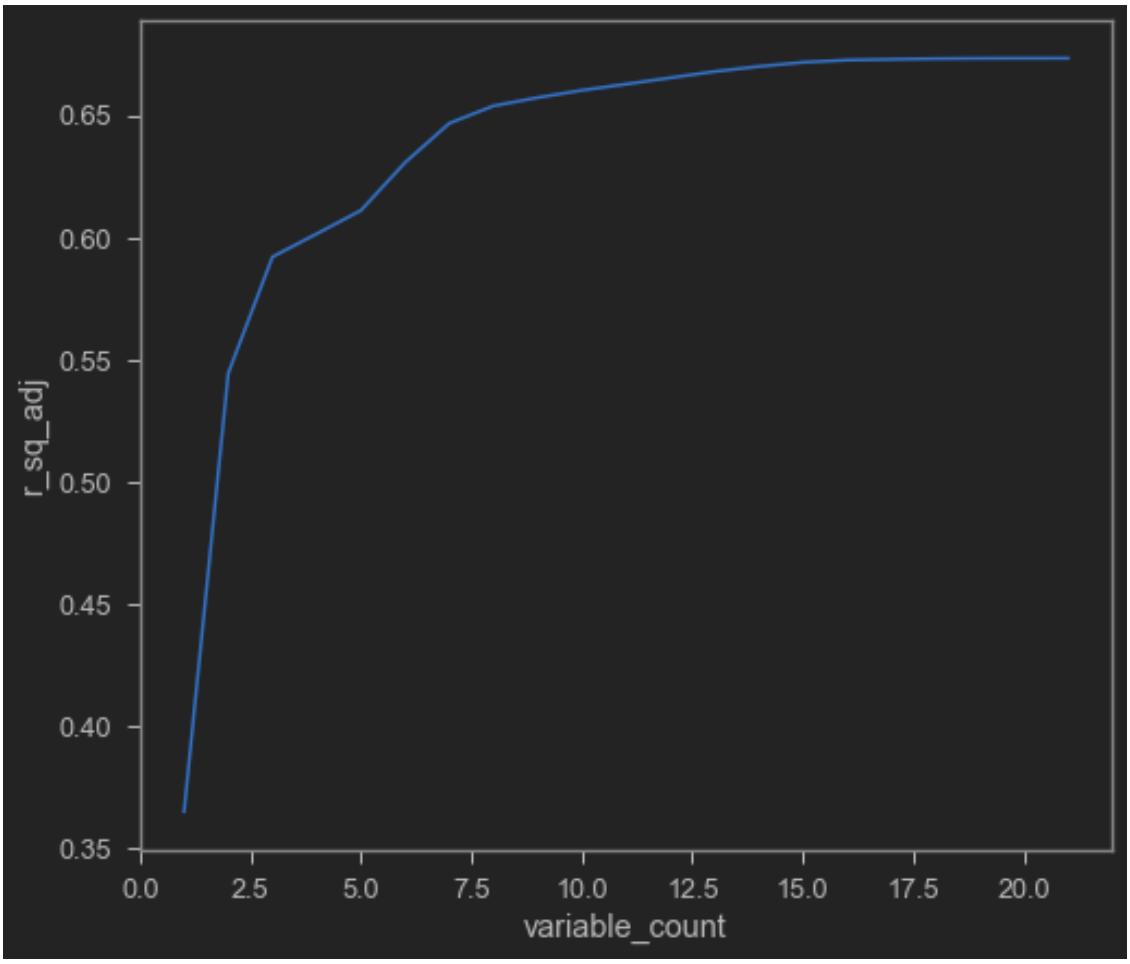
| | variable_count | r_sq_adj |
|---|---|---|
| 0 | 1 | 0.382388 |
| 1 | 2 | 0.513981 |
| 2 | 3 | 0.609593 |
| 3 | 4 | 0.629989 |
| 4 | 5 | 0.647099 |
| 5 | 6 | 0.654238 |
| 6 | 7 | 0.659856 |
| 7 | 8 | 0.662736 |
| 8 | 9 | 0.665432 |
| 9 | 10 | 0.667641 |
| 10 | 11 | 0.669555 |
| 11 | 12 | 0.670572 |
| 12 | 13 | 0.670831 |
| 13 | 14 | 0.671049 |
| 14 | 15 | 0.671054 |



[145]:
```
# with OHE
name_list = [x.split(".")[0].replace("-","") for x in␣
 ↪list(df_model_processed_ohe.columns)]
df_model_processed_ohe.columns = name_list

catch_ = pd.DataFrame(catch_forward_selected_steps(df_model_processed_ohe,␣
 ↪'price'))
sns.lineplot(x='variable_count',y='r_sq_adj',data=catch_);
catch_
```

[145]:

| | variable_count | r_sq_adj |
|---|---|---|
| 0 | 1 | 0.365253 |
| 1 | 2 | 0.544675 |
| 2 | 3 | 0.592259 |
| 3 | 4 | 0.601689 |
| 4 | 5 | 0.611253 |
| 5 | 6 | 0.630855 |
| 6 | 7 | 0.646900 |
| 7 | 8 | 0.654035 |
| 8 | 9 | 0.657412 |
| 9 | 10 | 0.660401 |
| 10 | 11 | 0.662921 |
| 11 | 12 | 0.665511 |
| 12 | 13 | 0.668086 |
| 13 | 14 | 0.670153 |
| 14 | 15 | 0.671848 |
| 15 | 16 | 0.672785 |
| 16 | 17 | 0.673084 |
| 17 | 18 | 0.673359 |
| 18 | 19 | 0.673478 |
| 19 | 20 | 0.673518 |
| 20 | 21 | 0.673531 |




Optimal number of features where diminishing return starts to occur.

[146]:
```
model = forward_selected(df_model_processed_ohe, 'price')
print(model.model.formula)
print(model.rsquared_adj)
model.summary()
```

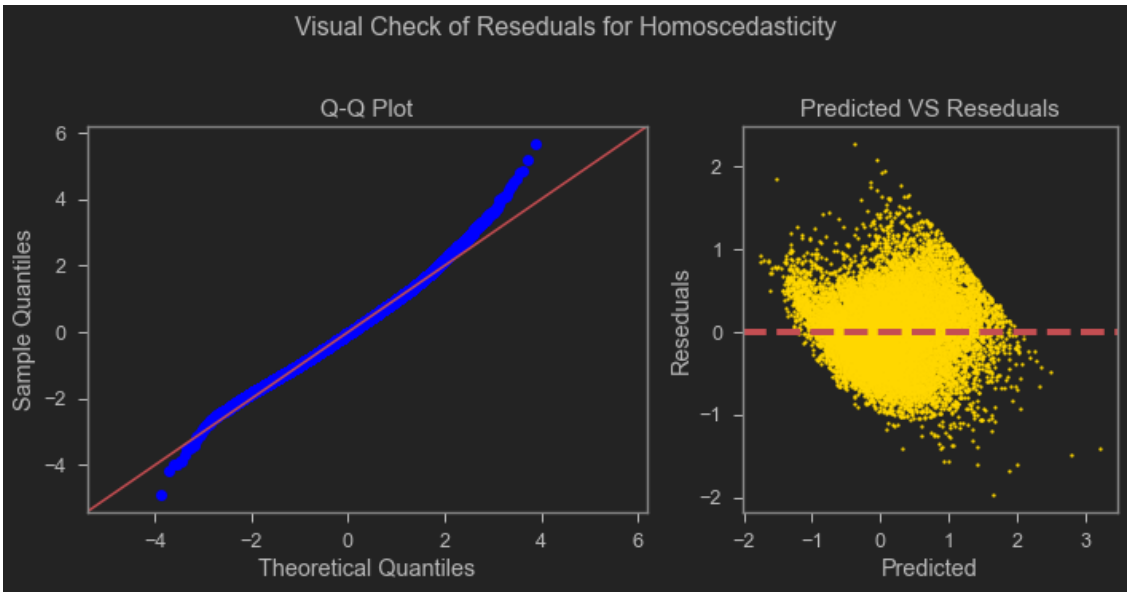
```
price ~ sqft_living + distance_from_downtown_mile + sqft_living15 + grade_2 +
grade_1 + grade_0 + yr_built + sqft_lot + condition_2 + grade_3 + condition_1 +
sqft_basement + waterfront_1 + bathrooms + bedrooms + is_renovated_1 +
condition_0 + sqft_lot15 + floors + grade_4 +
total_sqft_larger_than_neighbours_1 + 1
0.6735309838415255
```

[146]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
```

| | | | |
|---|---|---|---|
| Dep. Variable: | price | R-squared: | 0.674 |
| Model: | OLS | Adj. R-squared: | 0.674 |
| Method: | Least Squares | F-statistic: | 1649. |
| Date: | Mon, 19 Apr 2021 | Prob (F-statistic): | 0.00 |
| Time: | 19:42:58 | Log-Likelihood: | -10157. |
| No. Observations: | 19975 | AIC: | 2.037e+04 |
| Df Residuals: | 19949 | BIC: | 2.057e+04 |
| Df Model: | 25 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| Intercept | -0.0342 | 0.217 | -0.158 | 0.875 | -0.459 | 0.390 |
| sqft_living | 0.3080 | 0.009 | 34.371 | 0.000 | 0.290 | 0.326 |
| distance_from_downtown_mile | -0.4362 | 0.005 | -81.929 | 0.000 | -0.447 | -0.426 |
| sqft_living15 | 0.1983 | 0.006 | 32.636 | 0.000 | 0.186 | 0.210 |
| grade_2[0] | -0.6860 | 0.204 | -3.362 | 0.001 | -1.086 | -0.286 |
| grade_2[1] | 0.1666 | 0.202 | 0.826 | 0.409 | -0.229 | 0.562 |
| grade_1[0] | -0.6182 | 0.202 | -3.054 | 0.002 | -1.015 | -0.221 |
| grade_1[1] | -0.1229 | 0.202 | -0.609 | 0.543 | -0.519 | 0.273 |
| grade_0 | -0.3642 | 0.202 | -1.802 | 0.072 | -0.760 | 0.032 |
| yr_built | -0.1657 | 0.007 | -23.622 | 0.000 | -0.179 | -0.152 |
| sqft_lot | 0.0065 | 0.001 | 12.144 | 0.000 | 0.005 | 0.008 |
| condition_2 | 0.5431 | 0.079 | 6.898 | 0.000 | 0.389 | 0.697 |
| grade_3[0] | -0.6440 | 0.217 | -2.971 | 0.003 | -1.069 | -0.219 |
| grade_3[1] | 0.3219 | 0.202 | 1.593 | 0.111 | -0.074 | 0.718 |
| condition_1[0] | 0.1642 | 0.084 | 1.953 | 0.051 | -0.001 | 0.329 |
| condition_1[1] | 0.4356 | 0.078 | 5.566 | 0.000 | 0.282 | 0.589 |

```
sqft_basement                          -0.0509      0.005    -10.325      0.000
-0.061      -0.041
waterfront_1                            0.6683      0.057     11.675      0.000
0.556       0.781
bathrooms                               0.0766      0.007     10.810      0.000
0.063       0.090
bedrooms                               -0.0413      0.004     -9.593      0.000
-0.050      -0.033
is_renovated_1                          0.1314      0.018      7.457      0.000
0.097       0.166
condition_0                             0.3397      0.078      4.343      0.000
0.186       0.493
sqft_lot15                              0.0033      0.001      4.392      0.000
0.002       0.005
floors                                  0.0230      0.008      2.941      0.003
0.008       0.038
grade_4                                 0.3825      0.206      1.860      0.063
-0.021       0.786
total_sqft_larger_than_neighbours_1    -0.0291      0.022     -1.341      0.180
-0.072       0.013
==============================================================================
Omnibus:                      472.383   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              582.331
Skew:                           0.313   Prob(JB):                    3.53e-127
Kurtosis:                       3.554   Cond. No.                     1.91e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.91e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```



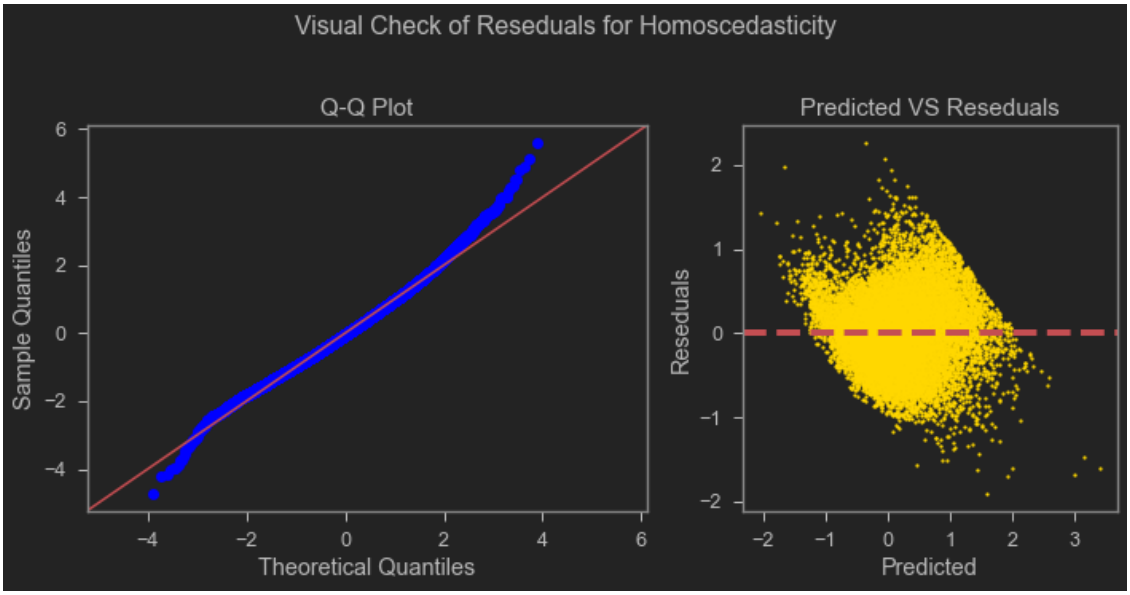

[147]:
```
# without one hot encoding, wrong approach
model1 = forward_selected(df_model_2, 'price')
print(model.model.formula)
print(model.rsquared_adj)
model.summary()
```

```
price ~ sqft_living + distance_from_downtown_mile + sqft_living15 + grade_2 +
grade_1 + grade_0 + yr_built + sqft_lot + condition_2 + grade_3 + condition_1 +
sqft_basement + waterfront_1 + bathrooms + bedrooms + is_renovated_1 +
condition_0 + sqft_lot15 + floors + grade_4 +
total_sqft_larger_than_neighbours_1 + 1
0.6735309838415255
```

[147]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.674
Model:                            OLS   Adj. R-squared:                  0.674
Method:                 Least Squares   F-statistic:                     1649.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:01   Log-Likelihood:                -10157.
No. Observations:               19975   AIC:                         2.037e+04
Df Residuals:                   19949   BIC:                         2.057e+04
Df Model:                          25
Covariance Type:            nonrobust
==============================================================================
======================
```

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| Intercept | -0.0342 | 0.217 | -0.158 | 0.875 | -0.459 | 0.390 |
| sqft_living | 0.3080 | 0.009 | 34.371 | 0.000 | 0.290 | 0.326 |
| distance_from_downtown_mile | -0.4362 | 0.005 | -81.929 | 0.000 | -0.447 | -0.426 |
| sqft_living15 | 0.1983 | 0.006 | 32.636 | 0.000 | 0.186 | 0.210 |
| grade_2[0] | -0.6860 | 0.204 | -3.362 | 0.001 | -1.086 | -0.286 |
| grade_2[1] | 0.1666 | 0.202 | 0.826 | 0.409 | -0.229 | 0.562 |
| grade_1[0] | -0.6182 | 0.202 | -3.054 | 0.002 | -1.015 | -0.221 |
| grade_1[1] | -0.1229 | 0.202 | -0.609 | 0.543 | -0.519 | 0.273 |
| grade_0 | -0.3642 | 0.202 | -1.802 | 0.072 | -0.760 | 0.032 |
| yr_built | -0.1657 | 0.007 | -23.622 | 0.000 | -0.179 | -0.152 |
| sqft_lot | 0.0065 | 0.001 | 12.144 | 0.000 | 0.005 | 0.008 |
| condition_2 | 0.5431 | 0.079 | 6.898 | 0.000 | 0.389 | 0.697 |
| grade_3[0] | -0.6440 | 0.217 | -2.971 | 0.003 | -1.069 | -0.219 |
| grade_3[1] | 0.3219 | 0.202 | 1.593 | 0.111 | -0.074 | 0.718 |
| condition_1[0] | 0.1642 | 0.084 | 1.953 | 0.051 | -0.001 | 0.329 |
| condition_1[1] | 0.4356 | 0.078 | 5.566 | 0.000 | 0.282 | 0.589 |
| sqft_basement | -0.0509 | 0.005 | -10.325 | 0.000 | -0.061 | -0.041 |
| waterfront_1 | 0.6683 | 0.057 | 11.675 | 0.000 | 0.556 | 0.781 |
| bathrooms | 0.0766 | 0.007 | 10.810 | 0.000 | 0.063 | 0.090 |
| bedrooms | -0.0413 | 0.004 | -9.593 | 0.000 | -0.050 | -0.033 |
| is_renovated_1 | 0.1314 | 0.018 | 7.457 | 0.000 | 0.097 | 0.166 |
| condition_0 | 0.3397 | 0.078 | 4.343 | 0.000 | | |

```
0.186       0.493
sqft_lot15                              0.0033      0.001       4.392       0.000
0.002       0.005
floors                                  0.0230      0.008       2.941       0.003
0.008       0.038
grade_4                                 0.3825      0.206       1.860       0.063
-0.021       0.786
total_sqft_larger_than_neighbours_1    -0.0291      0.022      -1.341       0.180
-0.072       0.013
==============================================================================
Omnibus:                      472.383   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              582.331
Skew:                           0.313   Prob(JB):                    3.53e-127
Kurtosis:                       3.554   Cond. No.                     1.91e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.91e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```



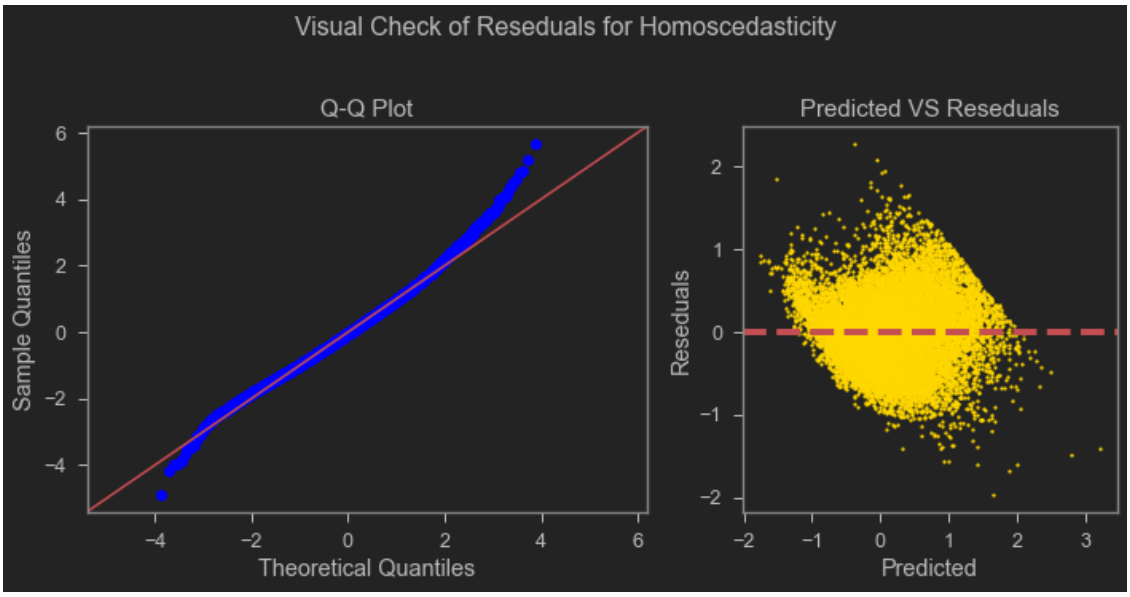

[148]:
```
# with one hot encoding, wrong approach
model1 = forward_selected(df_model_processed_ohe, 'price')
print(model.model.formula)
print(model.rsquared_adj)
```

```
model.summary()
```

```
price ~ sqft_living + distance_from_downtown_mile + sqft_living15 + grade_2 +
grade_1 + grade_0 + yr_built + sqft_lot + condition_2 + grade_3 + condition_1 +
sqft_basement + waterfront_1 + bathrooms + bedrooms + is_renovated_1 +
condition_0 + sqft_lot15 + floors + grade_4 +
total_sqft_larger_than_neighbours_1 + 1
0.6735309838415255
```

[148]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.674
Model:                            OLS   Adj. R-squared:                  0.674
Method:                 Least Squares   F-statistic:                     1649.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:08   Log-Likelihood:                -10157.
No. Observations:               19975   AIC:                         2.037e+04
Df Residuals:                   19949   BIC:                         2.057e+04
Df Model:                          25
Covariance Type:            nonrobust
==============================================================================
                                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
Intercept                                    -0.0342      0.217     -0.158      0.875
-0.459       0.390
sqft_living                                   0.3080      0.009     34.371      0.000
0.290       0.326
distance_from_downtown_mile                  -0.4362      0.005    -81.929      0.000
-0.447      -0.426
sqft_living15                                 0.1983      0.006     32.636      0.000
0.186       0.210
grade_2[0]                                   -0.6860      0.204     -3.362      0.001
-1.086      -0.286
grade_2[1]                                    0.1666      0.202      0.826      0.409
-0.229       0.562
grade_1[0]                                   -0.6182      0.202     -3.054      0.002
-1.015      -0.221
grade_1[1]                                   -0.1229      0.202     -0.609      0.543
-0.519       0.273
grade_0                                      -0.3642      0.202     -1.802      0.072
-0.760       0.032
yr_built                                     -0.1657      0.007    -23.622      0.000
```

```
-0.179      -0.152
sqft_lot                                  0.0065      0.001     12.144      0.000
0.005       0.008
condition_2                               0.5431      0.079      6.898      0.000
0.389       0.697
grade_3[0]                               -0.6440      0.217     -2.971      0.003
-1.069      -0.219
grade_3[1]                                0.3219      0.202      1.593      0.111
-0.074       0.718
condition_1[0]                            0.1642      0.084      1.953      0.051
-0.001       0.329
condition_1[1]                            0.4356      0.078      5.566      0.000
0.282       0.589
sqft_basement                            -0.0509      0.005    -10.325      0.000
-0.061      -0.041
waterfront_1                              0.6683      0.057     11.675      0.000
0.556       0.781
bathrooms                                 0.0766      0.007     10.810      0.000
0.063       0.090
bedrooms                                 -0.0413      0.004     -9.593      0.000
-0.050      -0.033
is_renovated_1                            0.1314      0.018      7.457      0.000
0.097       0.166
condition_0                               0.3397      0.078      4.343      0.000
0.186       0.493
sqft_lot15                                0.0033      0.001      4.392      0.000
0.002       0.005
floors                                    0.0230      0.008      2.941      0.003
0.008       0.038
grade_4                                   0.3825      0.206      1.860      0.063
-0.021       0.786
total_sqft_larger_than_neighbours_1      -0.0291      0.022     -1.341      0.180
-0.072       0.013
==============================================================================
Omnibus:                      472.383   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              582.331
Skew:                           0.313   Prob(JB):                    3.53e-127
Kurtosis:                       3.554   Cond. No.                     1.91e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.91e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```




### 5.4.3 Hybrid feature Elimination

**Recursive Feature Elimination Scikit-learn**

[149]:
```
from sklearn.feature_selection import RFE
```

[150]:
```
X = df_model_processed_ohe.drop(columns='price').copy()
```

[151]:
```
y = df_model_processed_ohe['price'].copy()
```

[152]:
```
linreg = LinearRegression(n_jobs=8)
selector = RFE(linreg ,n_features_to_select=6)
selector = selector.fit(X, y.values.ravel())
```

[153]:
```
selector.support_
```

[153]:
```
array([False, False,  True, False, False, False, False, False, False,
        True,  True, False, False, False, False,  True,  True,  True,
       False, False, False, False, False, False, False, False])
```

[154]:
```
selector.ranking_ #The feature ranking
```

[154]:
```
array([17, 14,  1, 20, 18, 15, 12,  7, 21,  1,  1, 11, 10,  9,  6,  1,  1,
        1,  2,  3, 16,  8,  4, 13,  5, 19])
```

[155]:
```
features_selection = pd.DataFrame(list(
    zip(X.columns.to_list(), selector.support_.tolist(),
        selector.ranking_.tolist())),
                                  columns=['Feature', 'keep', 'ranking'])
```

```
features_selection.sort_values(by="ranking", ascending=True)
```

[155]:

| | Feature | keep | ranking |
|---|---|---|---|
| 10 | waterfront_1 | True | 1 |
| 2 | sqft_living | True | 1 |
| 17 | grade_1 | True | 1 |
| 16 | grade_2 | True | 1 |
| 15 | grade_3 | True | 1 |
| 9 | distance_from_downtown_mile | True | 1 |
| 18 | grade_0 | False | 2 |
| 19 | grade_1 | False | 3 |
| 22 | grade_4 | False | 4 |
| 24 | is_renovated_1 | False | 5 |
| 14 | condition_2 | False | 6 |
| 7 | sqft_living15 | False | 7 |
| 21 | grade_3 | False | 8 |
| 13 | condition_1 | False | 9 |
| 12 | condition_0 | False | 10 |
| 11 | condition_1 | False | 11 |
| 6 | yr_built | False | 12 |
| 23 | grade_5 | False | 13 |
| 1 | bathrooms | False | 14 |
| 5 | sqft_basement | False | 15 |
| 20 | grade_2 | False | 16 |
| 0 | bedrooms | False | 17 |
| 4 | floors | False | 18 |
| 25 | total_sqft_larger_than_neighbours_1 | False | 19 |
| 3 | sqft_lot | False | 20 |
| 8 | sqft_lot15 | False | 21 |

[156]:
```
features_selection.Feature.to_list()
```

[156]:
```
['bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'sqft_basement',
 'yr_built',
 'sqft_living15',
 'sqft_lot15',
 'distance_from_downtown_mile',
 'waterfront_1',
 'condition_1',
 'condition_0',
 'condition_1',
 'condition_2',
```

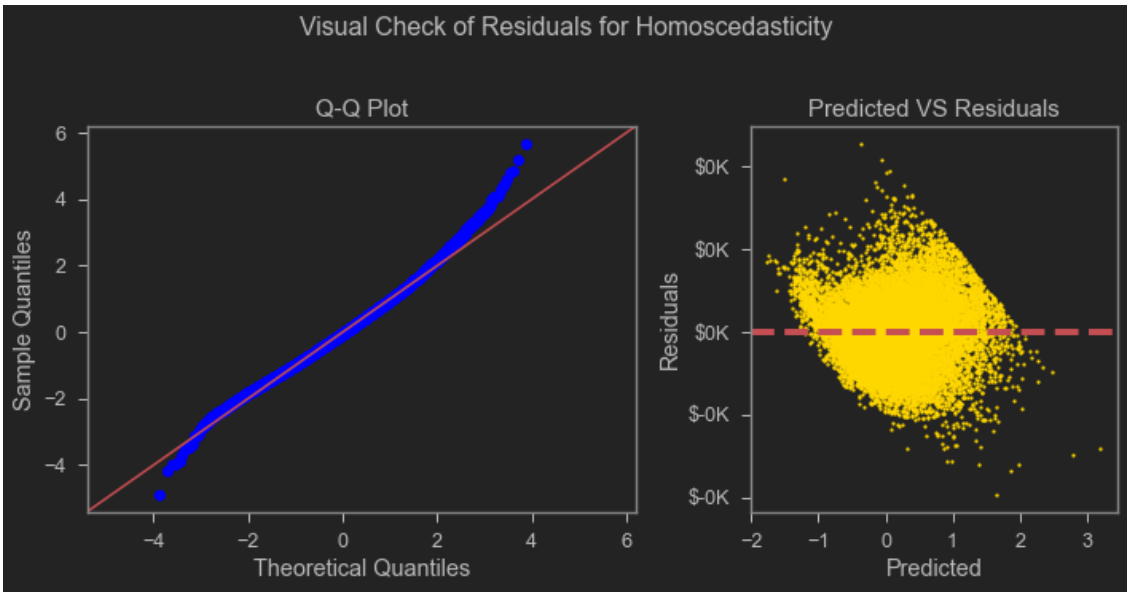
```
 'grade_3',
 'grade_2',
 'grade_1',
 'grade_0',
 'grade_1',
 'grade_2',
 'grade_3',
 'grade_4',
 'grade_5',
 'is_renovated_1',
 'total_sqft_larger_than_neighbours_1']
```

[157]:
```
fin1 = OLS_sm(df=df_model_processed_ohe,numeric_features=features_selection.
↪Feature.to_list())
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_basement + yr_built + sqft_living15 + sqft_lot15 +
distance_from_downtown_mile + waterfront_1 + condition_1 + condition_0 +
condition_1 + condition_2 + grade_3 + grade_2 + grade_1 + grade_0 + grade_1 +
grade_2 + grade_3 + grade_4 + grade_5 + is_renovated_1 +
total_sqft_larger_than_neighbours_1

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.674
Model:                            OLS   Adj. R-squared:                  0.674
Method:                 Least Squares   F-statistic:                     1586.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:09   Log-Likelihood:                -10157.
No. Observations:               19975   AIC:                         2.037e+04
Df Residuals:                   19948   BIC:                         2.058e+04
Df Model:                          26
Covariance Type:            nonrobust
================================================================================================
                                     coef    std err          t      P>|t|  ␣
 ↪    [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                         0.1046      0.411      0.255      0.799  ␣
 ↪    -0.700       0.910
bedrooms                         -0.0413      0.004     -9.595      0.000  ␣
 ↪    -0.050      -0.033
bathrooms                         0.0766      0.007     10.814      0.000  ␣
 ↪     0.063       0.091
sqft_living                       0.3082      0.009     34.357      0.000  ␣
 ↪     0.291       0.326
```

```
sqft_lot                                0.0065      0.001      12.144      0.000 ␣
 ↪     0.005       0.008
floors                                  0.0229      0.008       2.935      0.003 ␣
 ↪     0.008       0.038
sqft_basement                          -0.0509      0.005     -10.331      0.000 ␣
 ↪    -0.061      -0.041
yr_built                               -0.1656      0.007     -23.608      0.000 ␣
 ↪    -0.179      -0.152
sqft_living15                           0.1983      0.006      32.638      0.000 ␣
 ↪     0.186       0.210
sqft_lot15                              0.0033      0.001       4.393      0.000 ␣
 ↪     0.002       0.005
distance_from_downtown_mile            -0.4363      0.005     -81.870      0.000 ␣
 ↪    -0.447      -0.426
waterfront_1                            0.6684      0.057      11.675      0.000 ␣
 ↪     0.556       0.781
condition_1[0]                          0.1642      0.084       1.953      0.051 ␣
 ↪    -0.001       0.329
condition_1[1]                          0.4356      0.078       5.566      0.000 ␣
 ↪     0.282       0.589
condition_0                             0.3398      0.078       4.343      0.000 ␣
 ↪     0.186       0.493
condition_2                             0.5431      0.079       6.897      0.000 ␣
 ↪     0.389       0.697
grade_3[0]                             -0.7826      0.410      -1.907      0.057 ␣
 ↪    -1.587       0.022
grade_3[1]                              0.1828      0.404       0.453      0.651 ␣
 ↪    -0.609       0.974
grade_2[0]                             -0.8246      0.404      -2.042      0.041 ␣
 ↪    -1.616      -0.033
grade_2[1]                              0.0277      0.403       0.069      0.945 ␣
 ↪    -0.763       0.818
grade_1[0]                             -0.7569      0.403      -1.878      0.060 ␣
 ↪    -1.547       0.033
grade_1[1]                             -0.2618      0.403      -0.649      0.516 ␣
 ↪    -1.052       0.529
grade_0                                -0.5029      0.403      -1.247      0.212 ␣
 ↪    -1.293       0.287
grade_4                                 0.2434      0.406       0.600      0.549 ␣
 ↪    -0.552       1.039
grade_5                                -0.1855      0.466      -0.398      0.691 ␣
 ↪    -1.099       0.729
is_renovated_1                          0.1314      0.018       7.457      0.000 ␣
 ↪     0.097       0.166
total_sqft_larger_than_neighbours_1    -0.0291      0.022      -1.339      0.180 ␣
 ↪    -0.072       0.013
```

```
==============================================================================
Omnibus:                      472.315   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              582.233
Skew:                           0.313   Prob(JB):                    3.71e-127
Kurtosis:                       3.554   Cond. No.                     4.01e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
[2] The condition number is large, 4.01e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```




[158]:
```
check_for_high_p_val(fin1)
```

[158]:
```
<pandas.io.formats.style.Styler at 0x1c260818e80>
```

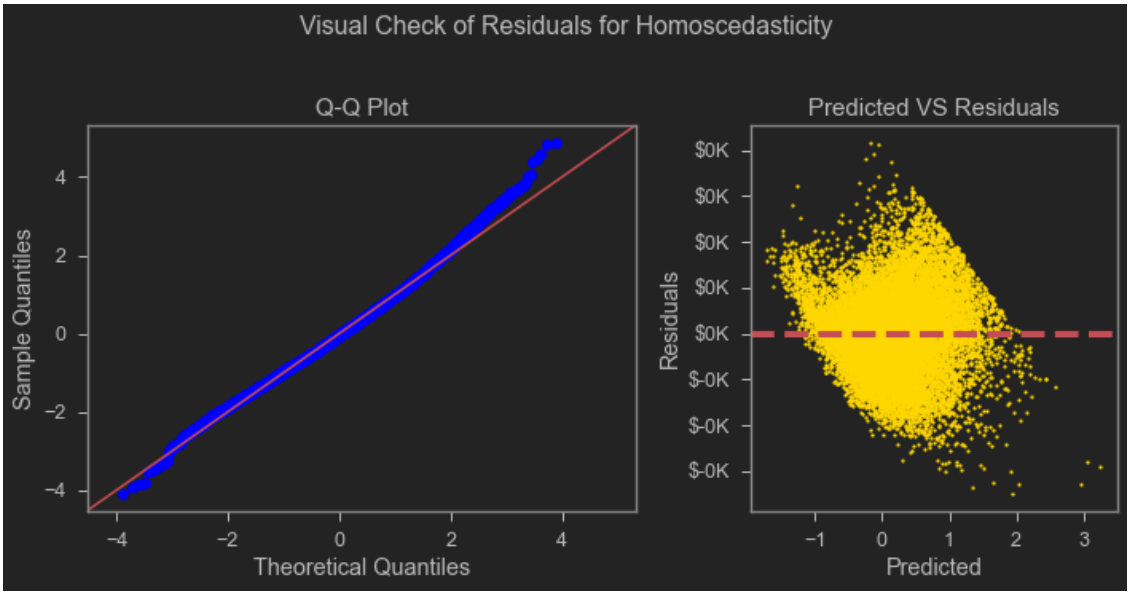
[159]:
```
fin1 = OLS_sm(df=df_model_processed_ohe,numeric_features=['bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'sqft_basement',
 'yr_built',
 'sqft_living15',
 'sqft_lot15',
 'distance_from_downtown_mile',
```

```
 'waterfront_1',
 'condition_1',
 'condition_0',
 'condition_1',
 'condition_2',
 'is_renovated_1'])
```

```
Formula for the OLS model:  price ~ bedrooms + bathrooms + sqft_living +
sqft_lot + floors + sqft_basement + yr_built + sqft_living15 + sqft_lot15 +
distance_from_downtown_mile + waterfront_1 + condition_1 + condition_0 +
condition_1 + condition_2 + is_renovated_1
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.632
Model:                            OLS   Adj. R-squared:                  0.631
Method:                 Least Squares   F-statistic:                     2139.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:09   Log-Likelihood:                -11376.
No. Observations:               19975   AIC:                         2.279e+04
Df Residuals:                   19958   BIC:                         2.292e+04
Df Model:                          16
Covariance Type:            nonrobust
================================================================================================
                                 coef    std err          t      P>|t|      [0.
 ↪025      0.975]
------------------------------------------------------------------------------------------------
Intercept                     -0.4312      0.083     -5.206      0.000      -0.
 ↪593      -0.269
bedrooms                      -0.0694      0.004    -15.529      0.000      -0.
 ↪078      -0.061
bathrooms                      0.0989      0.007     13.242      0.000       0.
 ↪084       0.114
sqft_living                    0.4608      0.009     52.588      0.000       0.
 ↪444       0.478
sqft_lot                       0.0072      0.001     12.700      0.000       0.
 ↪006       0.008
floors                         0.0545      0.008      6.647      0.000       0.
 ↪038       0.071
sqft_basement                 -0.0829      0.005    -16.249      0.000      -0.
 ↪093      -0.073
yr_built                      -0.0789      0.007    -11.022      0.000      -0.
 ↪093      -0.065
sqft_living15                  0.2860      0.006     46.480      0.000       0.
 ↪274       0.298
```

```
sqft_lot15                    0.0028      0.001      3.579      0.000        0.
 ↪001       0.004
distance_from_downtown_mile    -0.4969      0.005    -90.355      0.000       -0.
 ↪508      -0.486
waterfront_1                  0.6493      0.061     10.678      0.000        0.
 ↪530       0.769
condition_1[0]                0.2611      0.089      2.930      0.003        0.
 ↪086       0.436
condition_1[1]                0.5939      0.083      7.179      0.000        0.
 ↪432       0.756
condition_0                   0.4919      0.083      5.946      0.000        0.
 ↪330       0.654
condition_2                   0.6945      0.083      8.342      0.000        0.
 ↪531       0.858
is_renovated_1                0.1663      0.019      8.900      0.000        0.
 ↪130       0.203
==============================================================================
Omnibus:                      329.044   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              375.422
Skew:                           0.271   Prob(JB):                    3.01e-82
Kurtosis:                       3.396   Cond. No.                         548.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
"""
```



[160]:
```
check_for_high_p_val(fin1)
```

[160]:
```
<pandas.io.formats.style.Styler at 0x1c255c89a60>
```

Best by RFE but, can not take partial condition.

NOTE: Adding back other features like “zipcode” and “view” can have a huge impact. Then again not venturing that rout at this time.

# 6 MODEL

## 6.1 Some experiments

[161]:
```
RFE_recom =␣
↪['waterfront','grade','sqft_living','distance_from_downtown_mile','is_renovated','sqft_livi
```

[162]:
```
Forward_selection =␣
↪['sqft_living','distance_from_downtown_mile','sqft_living15','grade','yr_built','sqft_lot',
↪]
```

Creating a list based on the recommendations of RFE and Forward selection. Does RFE works well on categorial data liner regression? I am using these as tool for prediction rather than building a unattended pipeline. Useing them as a guide.

[163]:
```
set(RFE_recom + Forward_selection)
```

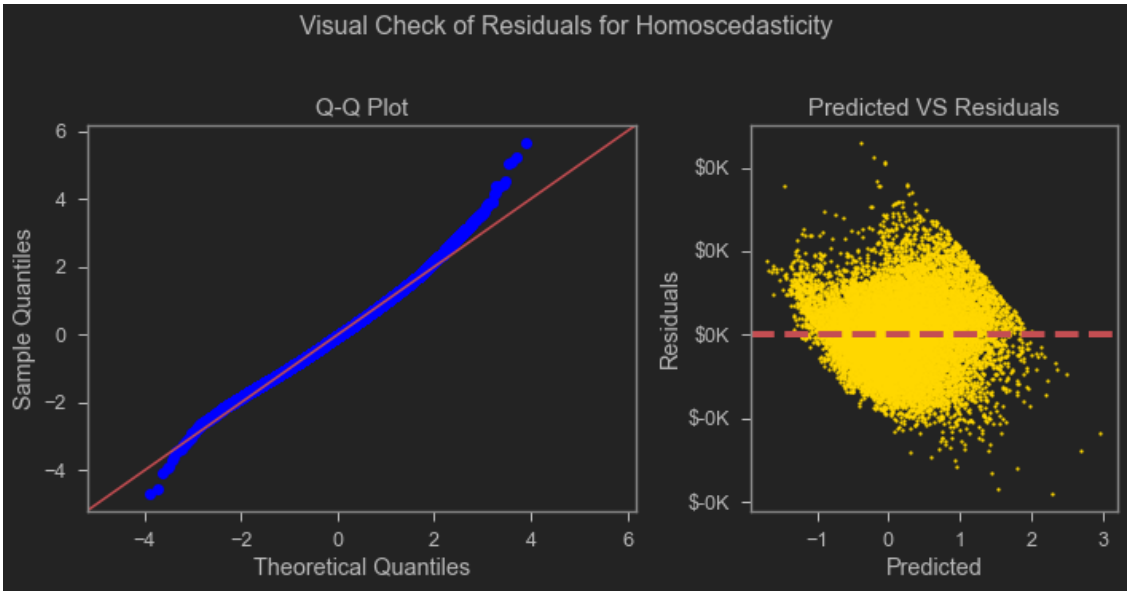
[163]:
```
{'condition',
 'distance_from_downtown_mile',
 'grade',
 'is_renovated',
 'sqft_living',
 'sqft_living15',
 'sqft_lot',
 'waterfront',
 'yr_built'}
```

[164]:
```
fin = OLS_sm(
    df=df_model_2,
    numeric_features=[
        'distance_from_downtown_mile', 'sqft_living', 'sqft_living15',
        'sqft_lot', 'yr_built'
    ],
    categorical_features=['condition', 'grade', 'is_renovated', 'waterfront'])
```

```
Formula for the OLS model:  price ~ distance_from_downtown_mile + sqft_living +
sqft_living15 + sqft_lot + yr_built + C(condition) + C(grade) + C(is_renovated)
+ C(waterfront)

<class 'statsmodels.iolib.summary.Summary'>
"""
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.668
Model:                            OLS   Adj. R-squared:                  0.667
Method:                 Least Squares   F-statistic:                     2004.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:10   Log-Likelihood:                -10350.
No. Observations:               19975   AIC:                         2.074e+04
Df Residuals:                   19954   BIC:                         2.091e+04
Df Model:                          20
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.
 ↪025      0.975]
------------------------------------------------------------------------------------------------
Intercept                        0.1125      0.415      0.271      0.786      -0.
 ↪700       0.925
C(condition)[T.-1.0]             0.1525      0.085      1.797      0.072      -0.
 ↪014       0.319
C(condition)[T.0.0]              0.3326      0.079      4.211      0.000       0.
 ↪178       0.487
C(condition)[T.1.0]              0.4141      0.079      5.242      0.000       0.
 ↪259       0.569
C(condition)[T.2.0]              0.5322      0.079      6.697      0.000       0.
 ↪376       0.688
C(grade)[T.-3.0]                -0.8018      0.414     -1.936      0.053      -1.
 ↪614       0.010
C(grade)[T.-2.0]                -0.8507      0.408     -2.087      0.037      -1.
 ↪650      -0.052
C(grade)[T.-1.0]                -0.7930      0.407     -1.949      0.051      -1.
 ↪591       0.005
C(grade)[T.0.0]                 -0.5379      0.407     -1.322      0.186      -1.
 ↪336       0.260
C(grade)[T.1.0]                 -0.2682      0.407     -0.659      0.510      -1.
 ↪066       0.530
C(grade)[T.2.0]                  0.0488      0.407      0.120      0.905      -0.
 ↪749       0.847
C(grade)[T.3.0]                  0.2305      0.408      0.566      0.572      -0.
 ↪568       1.029
C(grade)[T.4.0]                  0.3259      0.410      0.796      0.426      -0.
 ↪477       1.129
C(grade)[T.5.0]                 -0.0199      0.471     -0.042      0.966      -0.
 ↪942       0.903
C(is_renovated)[T.1.0]           0.1697      0.018      9.635      0.000       0.
 ↪135       0.204
```

```
C(waterfront)[T.1.0]            0.6866      0.058     11.894      0.000       0.
 ↪573       0.800
distance_from_downtown_mile    -0.4318      0.005    -85.822      0.000      -0.
 ↪442      -0.422
sqft_living                     0.2757      0.006     43.682      0.000       0.
 ↪263       0.288
sqft_living15                   0.2029      0.006     33.740      0.000       0.
 ↪191       0.215
sqft_lot                        0.0082      0.000     20.693      0.000       0.
 ↪007       0.009
yr_built                       -0.1181      0.006    -18.948      0.000      -0.
 ↪130      -0.106
==============================================================================
Omnibus:                      453.091   Durbin-Watson:                   1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              554.352
Skew:                           0.307   Prob(JB):                    4.21e-121
Kurtosis:                       3.537   Cond. No.                     3.48e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
[2] The condition number is large, 3.48e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```




[165]:
```
check_for_high_p_val(fin)
```

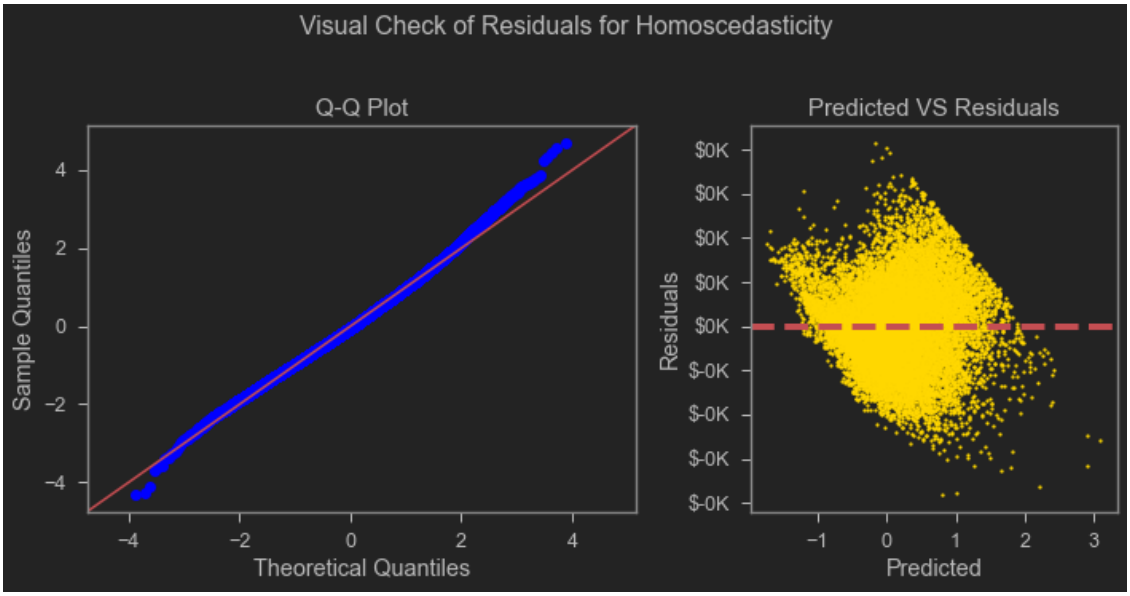
[165]: <pandas.io.formats.style.Styler at 0x1c2532803d0>

Dropping these for next iteration of model.

[166]:
```
fin = OLS_sm(
    df=df_model_2,
    numeric_features=[
        'distance_from_downtown_mile', 'sqft_living', 'sqft_living15',
        'sqft_lot', 'yr_built'
    ],
    categorical_features=[ 'is_renovated', 'waterfront'])
```

```
Formula for the OLS model:  price ~ distance_from_downtown_mile + sqft_living +
sqft_living15 + sqft_lot + yr_built + C(is_renovated) + C(waterfront)

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.607
Model:                            OLS   Adj. R-squared:                  0.607
Method:                 Least Squares   F-statistic:                     4402.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:11   Log-Likelihood:                -12026.
No. Observations:               19975   AIC:                         2.407e+04
Df Residuals:                   19967   BIC:                         2.413e+04
Df Model:                           7
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.
 ↪025      0.975]
------------------------------------------------------------------------------------------------
Intercept                        0.0707      0.003     21.108      0.000       0.
 ↪064       0.077
C(is_renovated)[T.1.0]           0.1874      0.019      9.955      0.000       0.
 ↪151       0.224
C(waterfront)[T.1.0]             0.6796      0.063     10.837      0.000       0.
 ↪557       0.803
distance_from_downtown_mile     -0.4936      0.005    -94.335      0.000      -0.
 ↪504      -0.483
sqft_living                      0.4214      0.006     66.902      0.000       0.
 ↪409       0.434
sqft_living15                    0.3049      0.006     49.709      0.000       0.
 ↪293       0.317
sqft_lot                         0.0087      0.000     20.406      0.000       0.
 ↪008       0.010
yr_built                        -0.0239      0.006     -4.248      0.000      -0.
 ↪035      -0.013
```

```
==============================================================================
Omnibus:                      271.281   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              305.272
Skew:                           0.245   Prob(JB):                     5.14e-67
Kurtosis:                       3.357   Cond. No.                         156.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
"""
```




[167]:
```
check_for_high_p_val(fin)
```

[167]:
```
<pandas.io.formats.style.Styler at 0x1c256929190>
```

I can go on doing this. But This is my final model. As there is no other significant p values left. And swapping features not going to help beyond this, as I already hit the diminishing return point.

Being said that I am throwing condition in the mix. removed waterfront as there is less control over that and removed yr_built as a result of high p_val. Most importantly it is recommended by RFE.

[168]:
```
final = OLS_sm(
    df=df_for_last_step,
    numeric_features=[
        'distance_from_downtown_mile', 'sqft_living', 'sqft_living15',
        'sqft_lot'
    ],
```

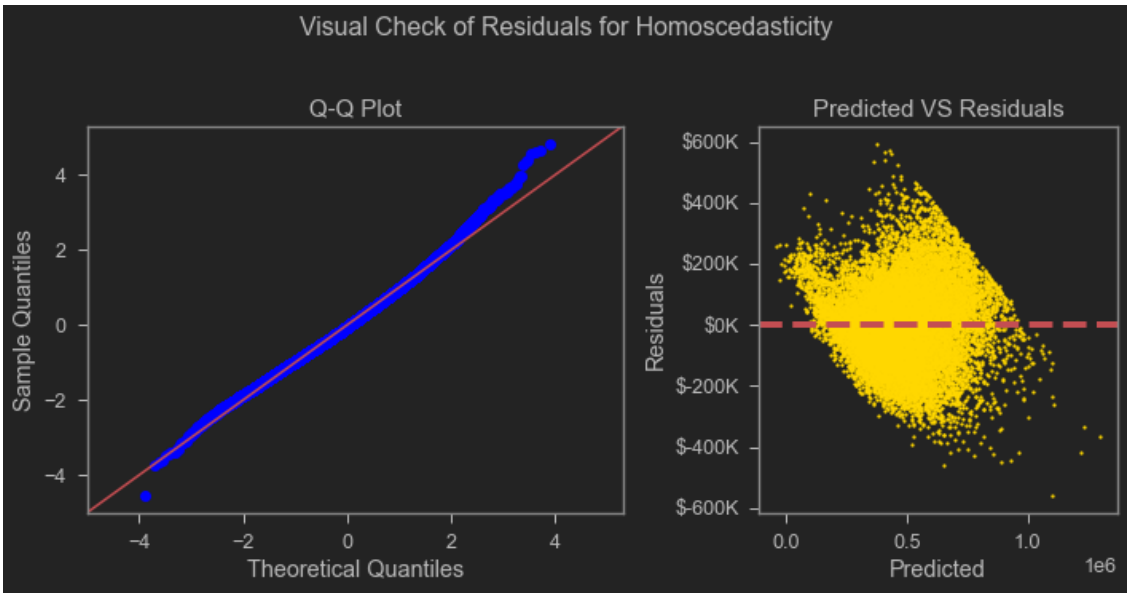
```
    categorical_features=[ 'is_renovated', 'condition'])
check_for_high_p_val(fin)
```

```
Formula for the OLS model:  price ~ distance_from_downtown_mile + sqft_living +
sqft_living15 + sqft_lot + C(is_renovated) + C(condition)
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.611
Model:                            OLS   Adj. R-squared:                  0.611
Method:                 Least Squares   F-statistic:                     3491.
Date:                Mon, 19 Apr 2021   Prob (F-statistic):               0.00
Time:                        19:43:11   Log-Likelihood:            -2.6245e+05
No. Observations:               19975   AIC:                         5.249e+05
Df Residuals:                   19965   BIC:                         5.250e+05
Df Model:                           9
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.
↪025      0.975]
------------------------------------------------------------------------------------------------
Intercept                    3.189e+05   2.37e+04     13.453      0.000    2.
↪72e+05    3.65e+05
C(is_renovated)[T.1.0]       6.502e+04   5141.670     12.645      0.000    5.
↪49e+04    7.51e+04
C(condition)[T.-1.0]         5.943e+04   2.56e+04      2.321      0.020    9235.
↪266     1.1e+05
C(condition)[T.0.0]          1.282e+05   2.37e+04      5.402      0.000    8.
↪17e+04    1.75e+05
C(condition)[T.1.0]          1.477e+05   2.38e+04      6.215      0.000    1.
↪01e+05    1.94e+05
C(condition)[T.2.0]          1.794e+05   2.39e+04      7.501      0.000    1.
↪33e+05    2.26e+05
distance_from_downtown_mile -1.394e+05   1360.809   -102.423      0.000   -1.
↪42e+05   -1.37e+05
sqft_living                  1.151e+05   1735.771     66.301      0.000    1.
↪12e+05    1.18e+05
sqft_living15                8.773e+04   1705.428     51.444      0.000    8.
↪44e+04    9.11e+04
sqft_lot                     2527.8924    118.879     21.264      0.000    2294.
↪880    2760.905
==============================================================================
Omnibus:                      356.337   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              405.760
Skew:                           0.286   Prob(JB):                     7.77e-89
```

```
Kurtosis:                       3.400   Cond. No.                         474.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
 ↪specified.
"""
```

[168]: <pandas.io.formats.style.Styler at 0x1c2657add00>




I can see a little improvement here. So this is the model I am gonna go forward with. One category of condition had a good impact on r square. Also swapped model to a one containing unscaled price. Which will help me interpret coefficients better.

[169]:
```
round(pd.DataFrame(final.params).sort_values(by=0,ascending=False),2)
```

[169]:

| | 0 |
|---|---|
| Intercept | 318876.68 |
| C(condition)[T.2.0] | 179403.32 |
| C(condition)[T.1.0] | 147676.79 |
| C(condition)[T.0.0] | 128191.14 |
| sqft_living | 115084.21 |
| sqft_living15 | 87733.31 |
| C(is_renovated)[T.1.0] | 65018.78 |
| C(condition)[T.-1.0] | 59428.20 |
| sqft_lot | 2527.89 |
| distance_from_downtown_mile | -139378.32 |

# 7 iNTERPRET

Accuracy of this model is around 61% indicated by r square. None of them have significant p value. No collinearity detected. Now looking at the betas I can see the relationships: _____ | Feature | Intercept (beta_1) | Comment | can control? | |———————————|————————|———————————————————————————————|————————| | Intercept | 330578.76 | Necessary to facilitate regression. OLS line goes through this on the y-axis | | | C(condition)[T.2.468888440743516] | 179403.32 | Positive relation, can expect to see this much improvement of value for every additional point | yes | | C(condition)[T.0.9195310230918753] | 147676.79 | Positive relation, can expect to see this much improvement of value for every additional point | yes | | C(condition)[T.-0.629826394559765] | 128191.14 | Positive relation, can expect to see this much improvement of value for every additional point | yes | | sqft_living | 85367.37 | Positive relation, can expect to see this much improvement of value for every additional sqft | yes | | sqft_living15 | 66830.29 | Positive relation, can expect to see this much improvement of value for every additional point | indirectly | | C(is_renovated)[T.5.692364186649059] | 65018.78 | Positive relation, can expect to see this much improvement of value if renovated | yes | | C(condition)[T.-2.1791838122114062] | 59428.20 | Positive relation, can expect to see this much improvement of value for every additional point | yes | | sqft_lot | 19286.77 | Positive relation, can expect to see this much improvement of value for every additional sqft | yes | | distance_from_downtown_mile | -94392.33 | Negative relation, can expect to see this much improvement of value for every additional distance | no |

---

Q-Q plot of residuals looks good, but the scatter plot is messy, indicating some bias. This is because of the nature of the data. Harsh outlier removal has impact on the result.

This is better from the base model, with every thing except r square. But the base model failed in everything except that, did not even satisfy some assumptions.

Adding more feature has a direct relation with r square. And leaving more outliers has a impact on the residuals distribution. One of the core assumptions of liner regression. Liner regression for this type of data alright, it tries to predict a specific price. In real life for this type of property use of range rather than accuracy is more appropriate. There are a lot of models out there for prediction. I dont know all of them. For that range calculation maybe a tree based model is better. I have little exposure to that right now. Liner regression is a widely use technique for prediction of price. Depending on target the r square varies.

# 8 RECOMMENDATIONS & CONCLUSIONS

Homeowners should focus to: * Condition of the house is important. Invest on improving elements that improves that. Focus on higher quality building materials. * Try to increase sqft living; consider adding to the property. * Should expect more value if closer to the city * Grade provided by the city is important. Read the guidelines and do things that improve grade. Like better materials * Dont be a outlier in the neighborhood in terms of living space. At least remodel to match that. * lot size also matters. try to strike a perfect balance here. * Bedrooms after 5 has a diminishing return. Do not focus on increasing. Instead invest on creating a better condition. * Basement sqft has a negative relation with value. Focus else where. * Renovation has a positive impact on price. * Old houses sells for less. If house is old then try to get better grade and condition

for better value.

Then they can observe a substantial improvement of their house value.

# 9 Appendix

## 9.1 Next Steps

If I had the gift of time.

I would like to, in no particular order:

- data collecting
- try a model with unscaled dependent variable; price and see the impact on model performance. That will help me better explain the data. But if not reporting those the the sign alone is enough.
- engineered more features. e.g. total house sqft, bedroom to bathroom ratio.
- process date as year, or split it to year and month and try to capture seasonality of value on listing timing.
- bin some features. e.g. year built by decades or something; bedrooms in single, typical, and large.
- make some boolean features. e.g. has basement.
- get some other monetary (e.g. average property price), and cultural and recreational (e.g. proximity to some stadium, local restaurant density, public transport, schools, parks) info about the locations. I am not sure how to get all those info.
- get some more info about view and include in the analysis. LOCATION does matter in the real estate business.
- Modeling
- try another method for RFE. e.g. tree based - random forest. RFECV - Recursive Feature Elimination with Cross Validation, RFE Hyperparameters tuning (really dont know much about this much at this time).
- use cross validation a.k.a. rotation estimation. e.g. k-fold cross-validation, Repeated random sub-sampling validation a.k.a. Monte Carlo Simulation.
- make train-test split and compare predictive power.
- use some other metric other than r square. e.g. RMSE, p value, coefficients.
- try other features in in model. go for an exhaustive search.
- Functional
- work on visuals, although wasted a lot of time doing this, mapping and what not.
- ”“functionize” EVERYTHING.” - James M. Irving. On the same note - refine docstrings.
- change naming convention of variables a bit

And the list goes on, I am stopping now. Almost nothing is perfect. The question is, is it good enough?

## 9.2 Polynomial Regression

Just to see the impact

[170]:
```
from sklearn.preprocessing import PolynomialFeatures
```

[171]:
```
features1 = [
    'sqft_lot', 'sqft_basement', 'sqft_living15', 'sqft_lot15',
    'bedrooms', 'bathrooms', 'floors', 'yr_built', 'waterfront', 'condition',
    'grade', 'is_renovated'
]
```

[172]:
```
poly_feat = PolynomialFeatures(degree=2)
X_train_poly = poly_feat.fit_transform(df_model_2[features1])
X_test_poly = poly_feat.fit_transform(df_model_2[features1])

lin_reg_sk_poly = linear_model.LinearRegression()
lin_reg_sk_poly.fit(X_train_poly, df_model_2['price'])
y_pred_poly = lin_reg_sk_poly.predict(X_test_poly)

print('Poly Model_1 ^2')
mean_squared_error = metrics.mean_squared_error(df_model_2['price'],
                                                y_pred_poly)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ',
      round(lin_reg_sk_poly.score(X_test_poly, df_model_2['price']), 3))
print('R-squared (testing) ',
      round(lin_reg_sk_poly.score(X_test_poly, df_model_2['price']), 3))
```

```
Poly Model_1 ^2
Mean Squared Error (MSE)  0.46
R-squared (training)  0.58
R-squared (testing)  0.58
```

[173]:
```
poly_feat = PolynomialFeatures(degree=3)
X_train_poly = poly_feat.fit_transform(df_model_2[features1])
X_test_poly = poly_feat.fit_transform(df_model_2[features1])

lin_reg_sk_poly = linear_model.LinearRegression()
lin_reg_sk_poly.fit(X_train_poly, df_model_2['price'])
y_pred_poly = lin_reg_sk_poly.predict(X_test_poly)

print('Poly Model_2 ^3')
mean_squared_error = metrics.mean_squared_error(df_model_2['price'],
                                                y_pred_poly)
```

```
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ',
      round(lin_reg_sk_poly.score(X_test_poly, df_model_2['price']), 3))
print('R-squared (testing) ',
      round(lin_reg_sk_poly.score(X_test_poly, df_model_2['price']), 3))
```

```
Poly Model_2 ^3
Mean Squared Error (MSE)  0.44
R-squared (training)  0.614
R-squared (testing)  0.614
```

### 9.3 Train - test split

[ ]:

#### 9.3.1 RFECV

[ ]: