Predicting House Prices

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
import folium
from folium.plugins import HeatMap
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

evaluation = pd.DataFrame({'Model': [],
                           'Details':[],
                           'Root Mean Squared Error (RMSE)':[],
                           'R-squared (training)':[],
                           'Adjusted R-squared (training)':[],
                           'R-squared (test)':[],
                           'Adjusted R-squared (test)':[],
                           '5-Fold Cross Validation':[]})

df = pd.read_csv('../input/kc_house_data.csv')
#df.describe()
#df.info()
df.head()
```

Out[1]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_abov
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	3	7	118
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	3	7	217
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	3	6	77
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	5	7	105
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	3	8	168

Defining a Function to Calculate the Adjusted R-squared

The R-squared increases when the number of features increase. Because of this, sometimes a more robust evaluator is preferred to compare the performance between different models. This evaluater is called adjusted R-squared and it only increases, if the addition of the variable reduces the MSE. The definition of the adjusted R^2 is:

$$\bar{R}^2 = R^2 - \frac{k-1}{n-k}(1 - R^2)$$

where n is the number of observations and k is the number of parameters.

In [2]:
```
def adjustedR2(r2,n,k):
    return r2-(k-1)/(n-k)*(1-r2)
```

Creating a Simple Linear Regression

When we establish a linear relationship between a response and a single explanatory variable, we term it as simple linear regression. In this case, my aim is to predict house prices, making 'price' our response variable. However, in order to create a simple model, we need to select a feature. Upon reviewing the dataset columns, the 'living area' (sqft) emerged as the most crucial feature.

Upon examining the correlation matrix, it became evident that 'price' exhibits the highest correlation coefficient with 'living area' (sqft), reinforcing my initial inclination. Therefore, I've opted to utilize 'living area' (sqft) as the primary feature. Nonetheless, if you wish to explore the relationship between 'price' and another feature, you may prefer to focus on that alternative feature.

In [3]:
```
#%%capture
train_data,test_data = train_test_split(df,train_size = 0.8,random_state=3)

lr = linear_model.LinearRegression()
X_train = np.array(train_data['sqft_living'], dtype=pd.Series).reshape(-1,1)
y_train = np.array(train_data['price'], dtype=pd.Series)
lr.fit(X_train,y_train)

X_test = np.array(test_data['sqft_living'], dtype=pd.Series).reshape(-1,1)
y_test = np.array(test_data['price'], dtype=pd.Series)

pred = lr.predict(X_test)
rmsesm = float(format(np.sqrt(metrics.mean_squared_error(y_test,pred)),'.3f'))
```

```
rtrsm = float(format(lr.score(X_train, y_train),'.3f'))
rtesm = float(format(lr.score(X_test, y_test),'.3f'))
cv = float(format(cross_val_score(lr,df[['sqft_living']],df['price'],cv=5).mean(),'.3f'))

print ("Average Price for Test Data: {:.3f}".format(y_test.mean()))
print('Intercept: {}'.format(lr.intercept_))
print('Coefficient: {}'.format(lr.coef_))

r = evaluation.shape[0]
evaluation.loc[r] = ['Simple Linear Regression','-',rmsesm,rtrsm,'-',rtesm,'-',cv]
evaluation
```

```
Average Price for Test Data: 539744.130
Intercept: -47235.811302901246
Coefficient: [282.2468152]
```

Out[3]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

I also printed the intercept and coefficient for the simple linear regression. By using these values and the below definition, we can estimate the house prices manually. The equation we use for our estimations is called hypothesis function and defined as

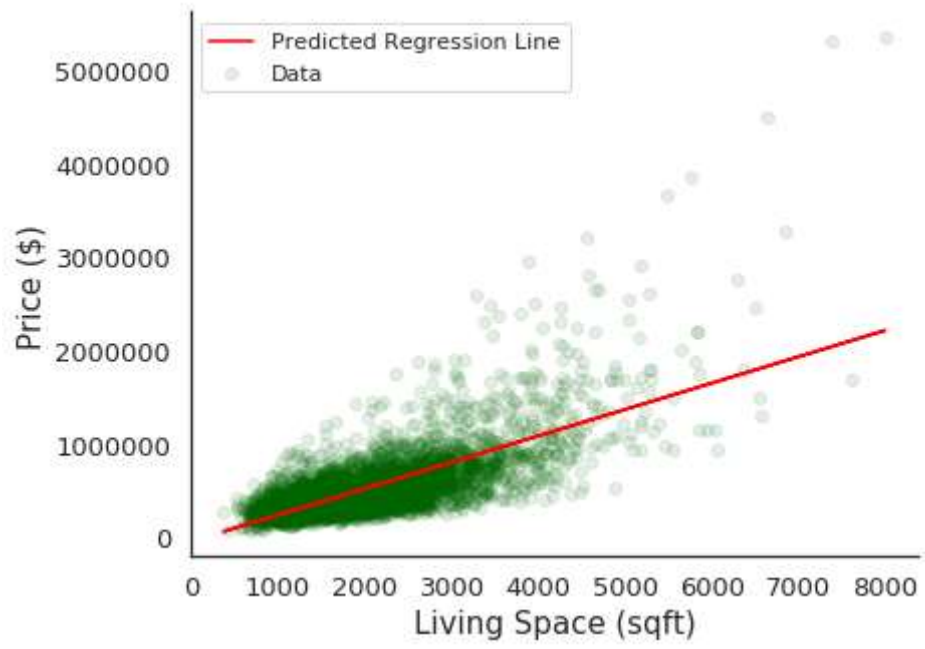
$$h_\theta(X) = \theta_0 + \theta_1 x$$

Show the Result

Because we're dealing with only two dimensions in simple regression, drawing it is straightforward. The chart below represents the outcome of the simple regression. It doesn't appear to be a perfect fit, but achieving a perfect fit with real-world datasets is often challenging

In [4]:
```
sns.set(style="white", font_scale=1)
```

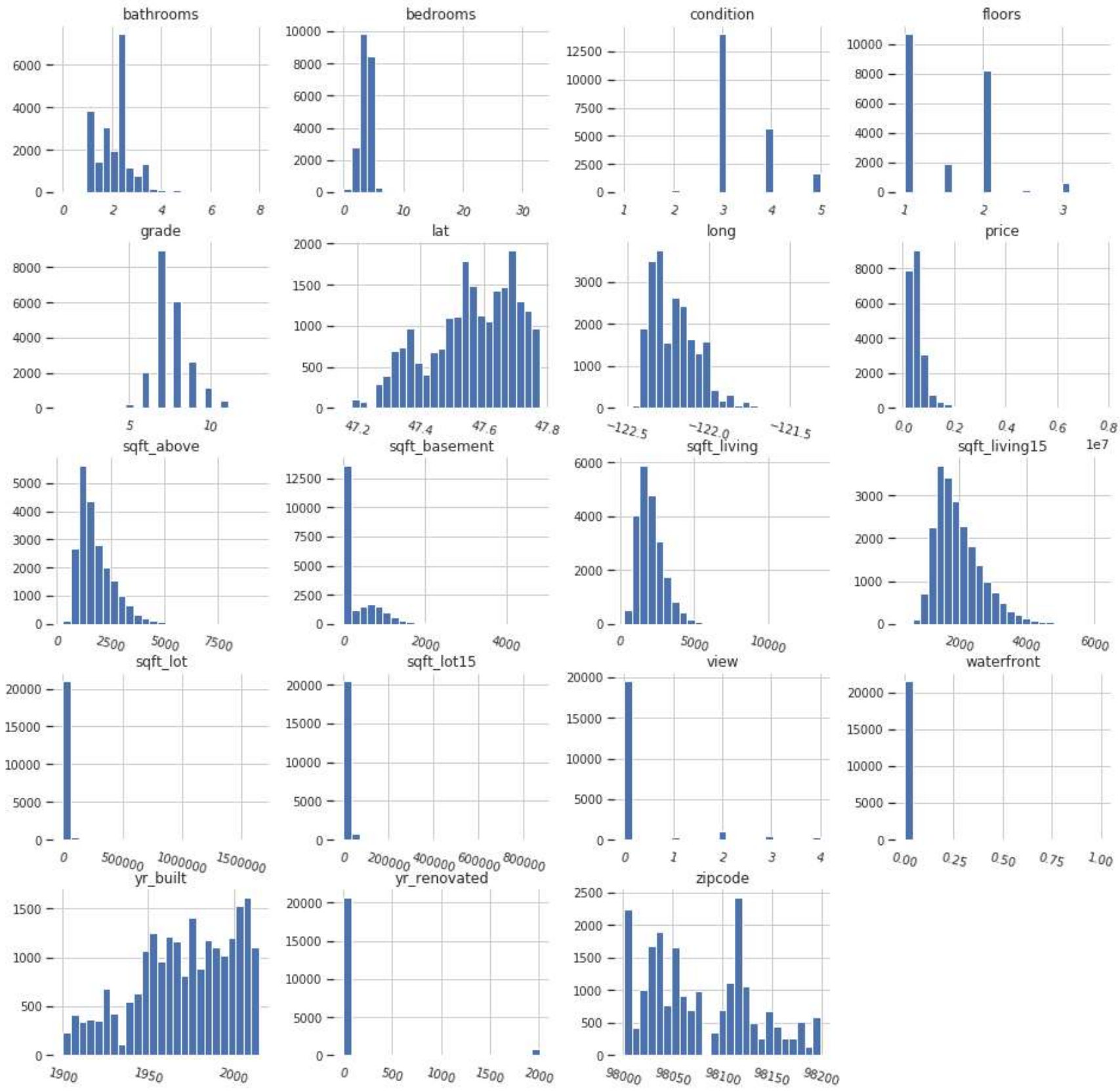
In [5]:
```
plt.figure(figsize=(6.5,5))
plt.scatter(X_test,y_test,color='darkgreen',label="Data", alpha=.1)
plt.plot(X_test,lr.predict(X_test),color="red",label="Predicted Regression Line")
plt.xlabel("Living Space (sqft)", fontsize=15)
plt.ylabel("Price ($)", fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.legend()

plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
```



Visualizing and Examining Data

Since our dataset isn't particularly large and doesn't encompass an excessive number of features, we have the opportunity to plot most of them, leading to valuable analytical insights. Engaging in the practice of drawing charts and exploring the data before implementing a model is highly beneficial. It allows us to identify potential outliers and make decisions regarding normalization. While not mandatory, familiarizing oneself with the data is always advantageous. I began by generating histograms of the dataframe.

In [6]:
```
df1=df[['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']]
h = df1.hist(bins=25,figsize=(16,16),xlabelsize='10',ylabelsize='10',xrot=-15)
sns.despine(left=True, bottom=True)
[x.title.set_size(12) for x in h.ravel()];
[x.yaxis.tick_left() for x in h.ravel()];
```

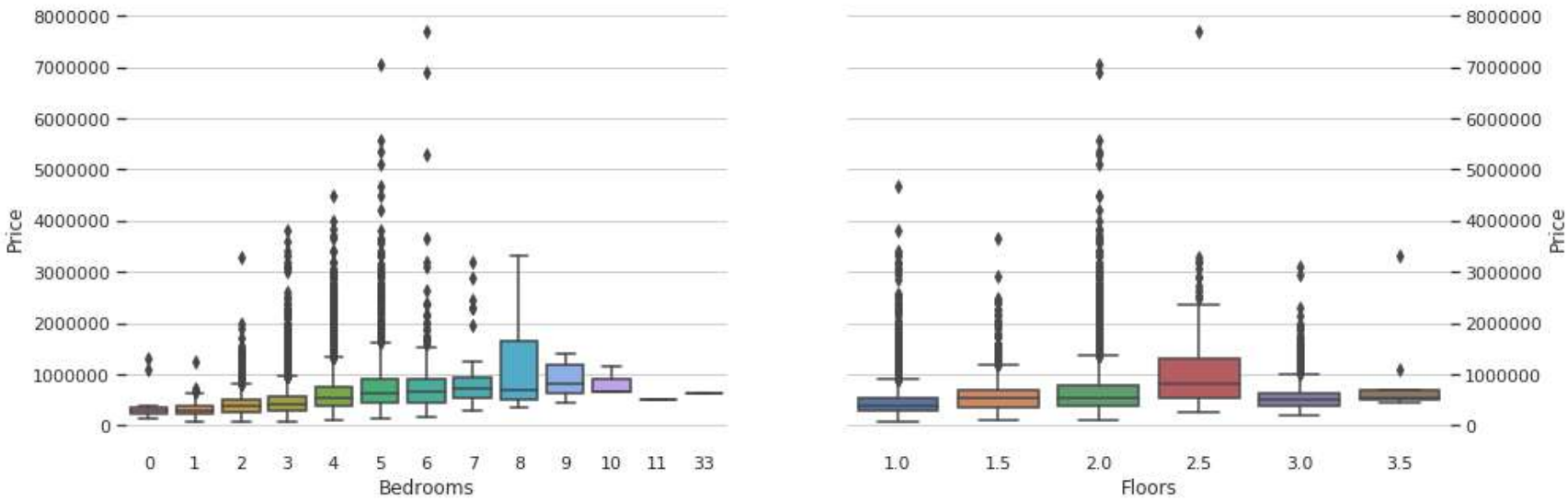


I opted for boxplots to analyze the relationship between bedrooms, floors, bathrooms/bedrooms, and price, as these features contain numerical data that isn't continuous—such as fractional values for floors or discrete values for bedrooms.

From the charts below, it's evident that there are only a handful of houses with certain features or prices significantly deviating from the rest, like those with 33 bedrooms or a price around 7000000. However, identifying their potential negative impact would require considerable time. In real-world datasets, outliers—such as luxury house prices in this dataset—are common. Therefore, I don't intend to remove these outliers.

In [7]:
```
sns.set(style="whitegrid", font_scale=1)
```

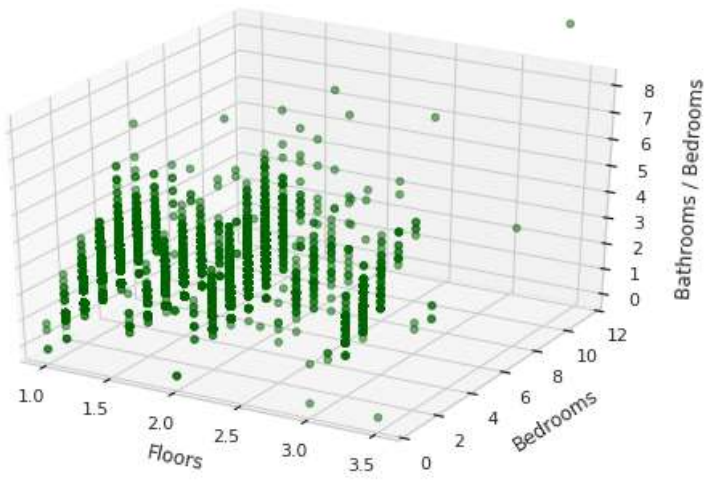
In [8]:
```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=df['bedrooms'],y=df['price'], ax=axes[0])
sns.boxplot(x=df['floors'],y=df['price'], ax=axes[1])
sns.despine(left=True, bottom=True)
axes[0].set(xlabel='Bedrooms', ylabel='Price')
axes[0].yaxis.tick_left()
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].set(xlabel='Floors', ylabel='Price')

f, axe = plt.subplots(1, 1,figsize=(12.18,5))
sns.despine(left=True, bottom=True)
sns.boxplot(x=df['bathrooms'],y=df['price'], ax=axe)
axe.yaxis.tick_left()
axe.set(xlabel='Bathrooms / Bedrooms', ylabel='Price');
```

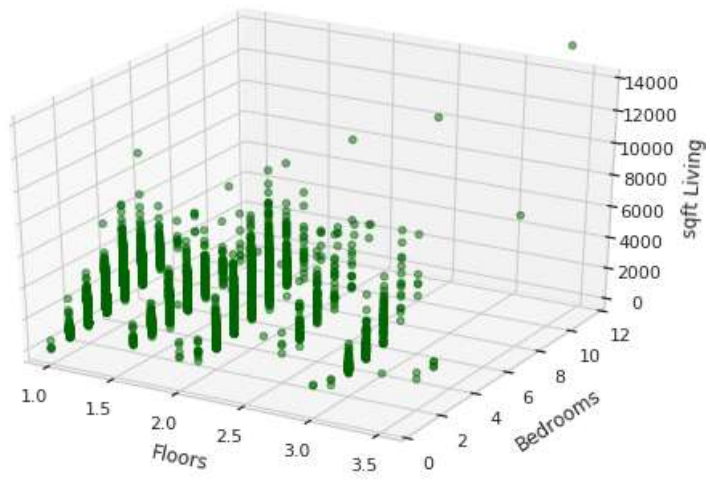


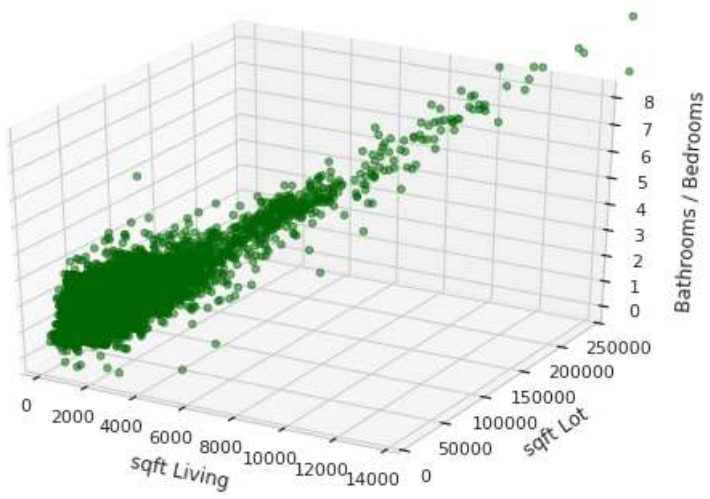
I generated plots depicting the relationship between price and various features, observing that a perfect linear relationship doesn't exist between price and these features. Curiosity led me to explore the interrelationships among the features themselves. To illustrate this, I employed 3D plots, using light green as the point color. The density of points is represented by darker shades of green, indicating areas with higher point overlap.

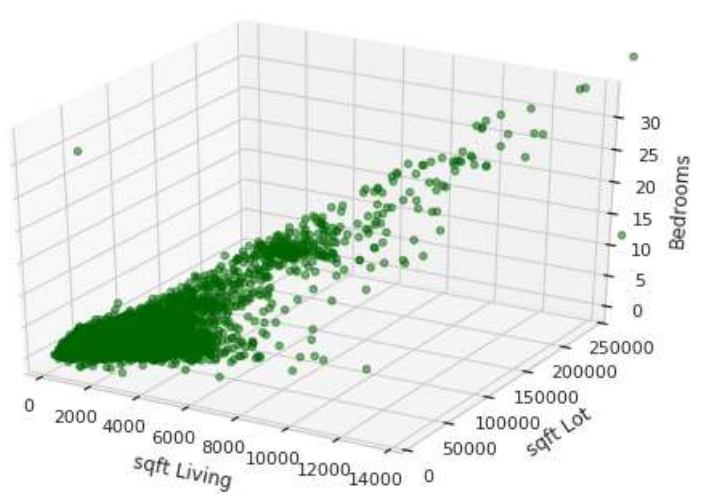
The visuals reveal that an increase in sqrt_living corresponds to an increase in sqrt_lot, bedrooms, and bathrooms/bedrooms. However, unlike these correlations, the relationship between floors, bedrooms, bathrooms/bedrooms, and sqrt_living doesn't follow a similar pattern.

In [9]:
```
fig=plt.figure(figsize=(19,12.5))
ax=fig.add_subplot(2,2,1, projection="3d")
ax.scatter(df['floors'],df['bedrooms'],df['bathrooms'],c="darkgreen",alpha=.5)
ax.set(xlabel='\nFloors',ylabel='\nBedrooms',zlabel='\nBathrooms / Bedrooms')
ax.set(ylim=[0,12])

ax=fig.add_subplot(2,2,2, projection="3d")
ax.scatter(df['floors'],df['bedrooms'],df['sqft_living'],c="darkgreen",alpha=.5)
ax.set(xlabel='\nFloors',ylabel='\nBedrooms',zlabel='\nsqft Living')
ax.set(ylim=[0,12])

ax=fig.add_subplot(2,2,3, projection="3d")
ax.scatter(df['sqft_living'],df['sqft_lot'],df['bathrooms'],c="darkgreen",alpha=.5)
ax.set(xlabel='\n sqft Living',ylabel='\nsqft Lot',zlabel='\nBathrooms / Bedrooms')
ax.set(ylim=[0,250000])

ax=fig.add_subplot(2,2,4, projection="3d")
ax.scatter(df['sqft_living'],df['sqft_lot'],df['bedrooms'],c="darkgreen",alpha=.5)
ax.set(xlabel='\n sqft Living',ylabel='\nsqft Lot',zlabel='Bedrooms')
ax.set(ylim=[0,250000]);
```



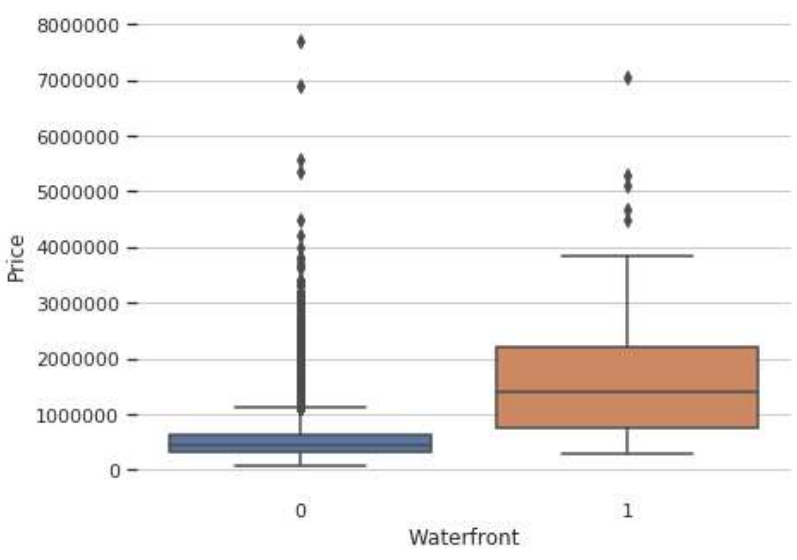


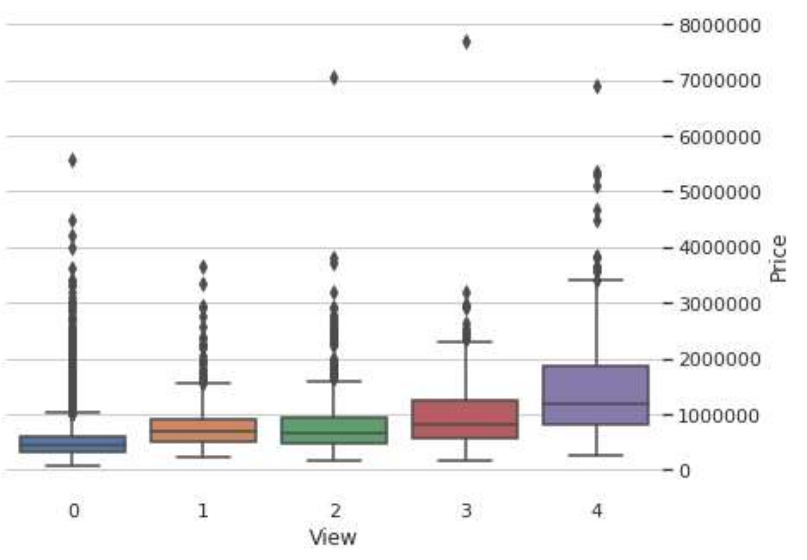


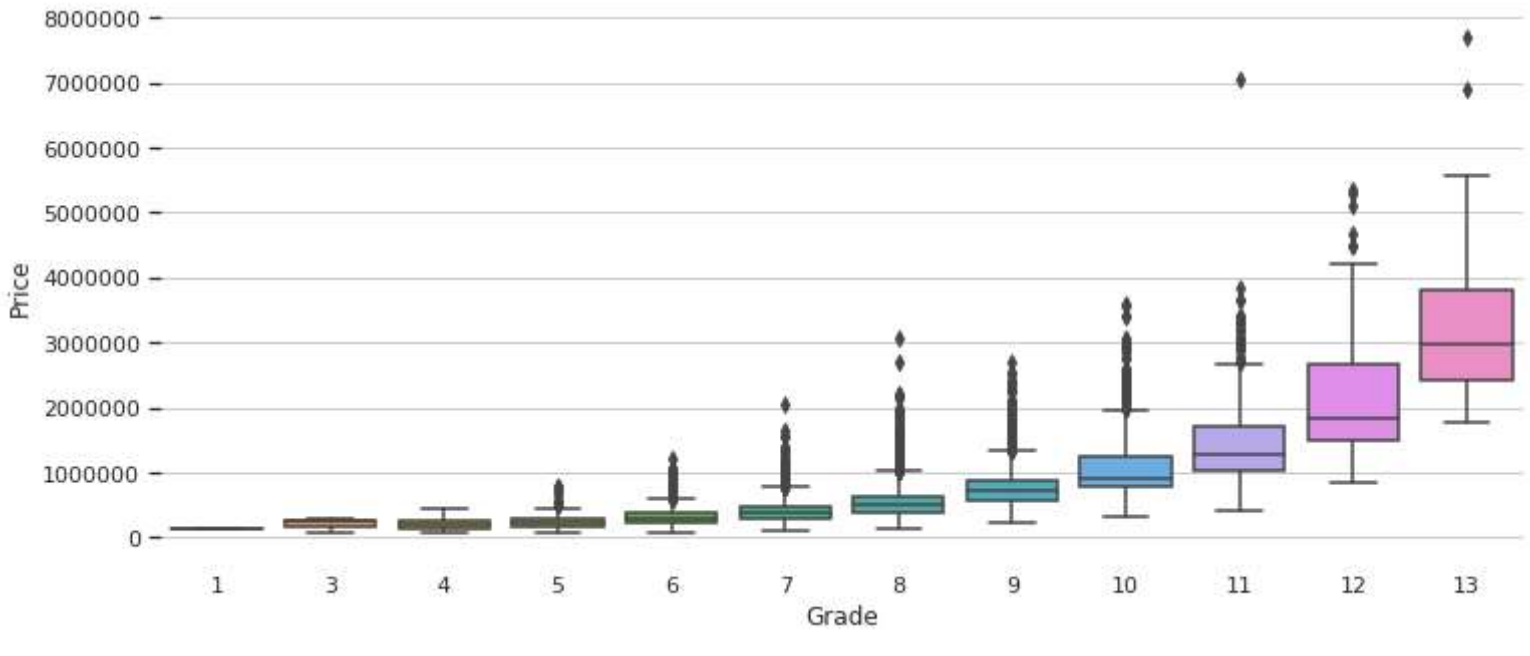


Expanding the visualization to encompass more features, the boxplots below shed light on the influence of grade and waterfront on price, showing a noticeable impact. Additionally, although to a lesser extent, the view also appears to affect price.

In [10]:
```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=df['waterfront'],y=df['price'], ax=axes[0])
sns.boxplot(x=df['view'],y=df['price'], ax=axes[1])
sns.despine(left=True, bottom=True)
axes[0].set(xlabel='Waterfront', ylabel='Price')
axes[0].yaxis.tick_left()
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].set(xlabel='View', ylabel='Price')

f, axe = plt.subplots(1, 1,figsize=(12.18,5))
sns.boxplot(x=df['grade'],y=df['price'], ax=axe)
sns.despine(left=True, bottom=True)
axe.yaxis.tick_left()
axe.set(xlabel='Grade', ylabel='Price');
```

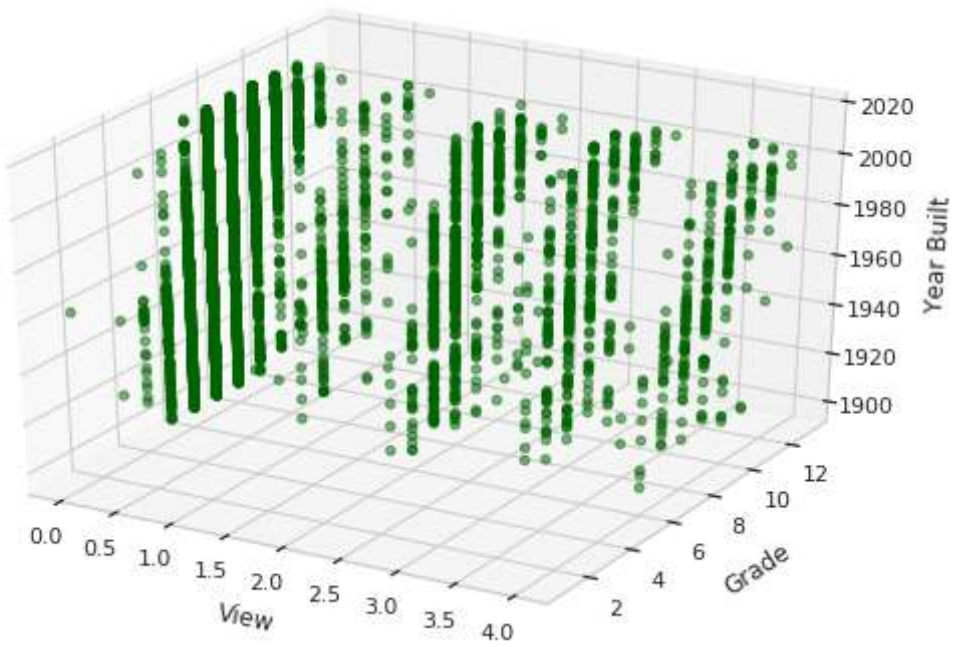






I extended the visualization by creating a 3D plot to explore the relationship between view, grade, and year built. The chart indicates that newer houses tend to have higher grades. However, regarding changes in the view, there isn't a distinct pattern discernible from the plot.

In [11]:
```
fig=plt.figure(figsize=(9.5,6.25))
ax=fig.add_subplot(1,1,1, projection="3d")
ax.scatter(train_data['view'],train_data['grade'],train_data['yr_built'],c="darkgreen",alpha=.5)
ax.set(xlabel='\nView',ylabel='\nGrade',zlabel='\nYear Built');
```

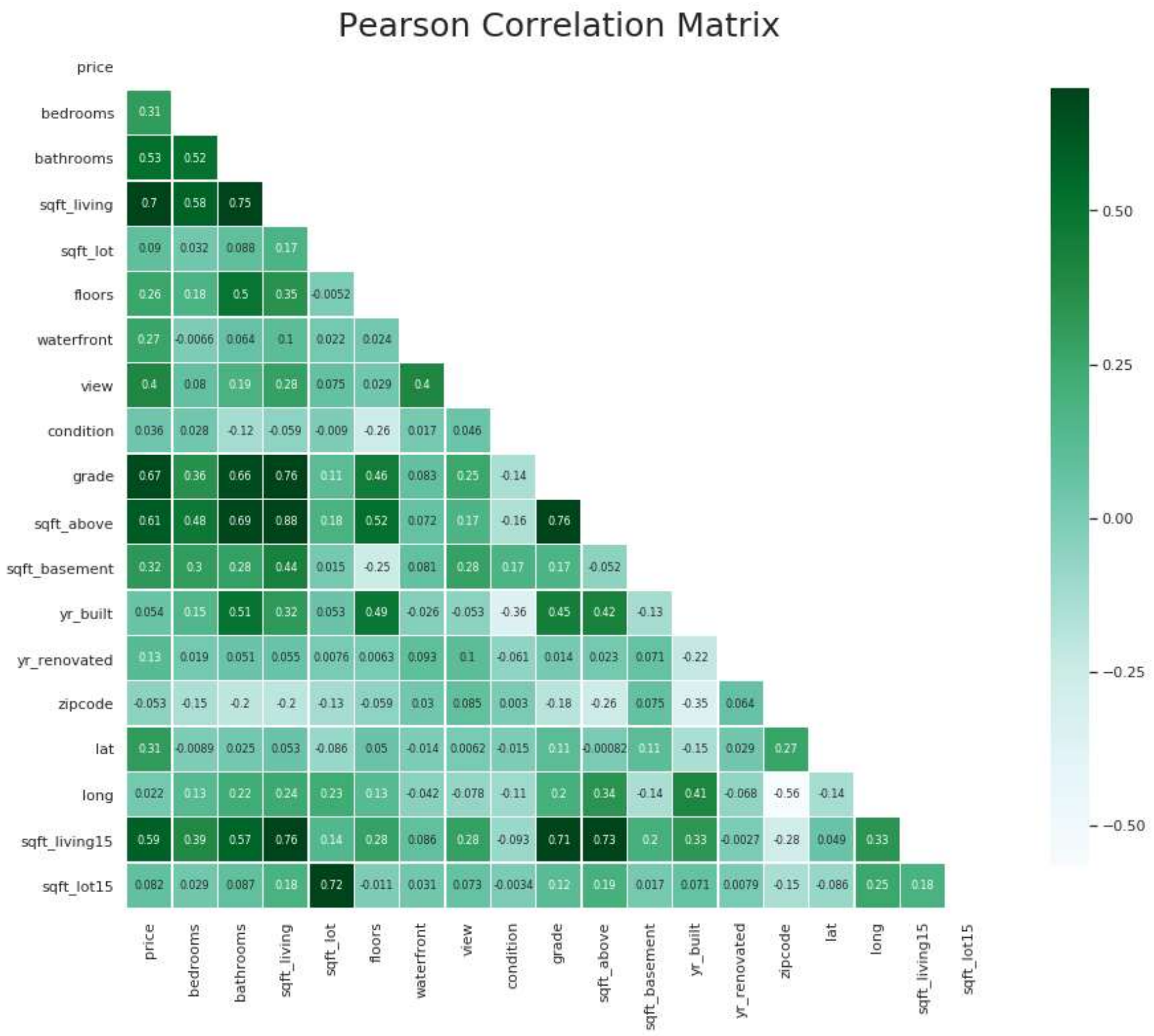


Checking Out the Correlation Among Explanatory Variables

Having an excessive number of features in a model isn't always advantageous, as it can lead to overfitting and potentially poorer performance when predicting values for new datasets. Therefore, if a feature doesn't notably enhance your model, opting not to include it might yield better results.

Another crucial consideration is correlation. When two features exhibit very high correlation, it's generally unwise to retain both, as this can contribute to overfitting. For instance, in cases of overfitting, highly correlated features like sqt_above and sqt_living might be candidates for removal due to their high correlation. This correlation can be estimated by examining the definitions in the dataset, but it's essential to confirm via the correlation matrix. However, it's not necessary to remove one of the highly correlated features in every scenario. For instance, while bathrooms and sqrt_living display high correlation, their relationship might differ significantly from that between sqt_living and sqt_above.

In [13]:
```
features = ['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']

mask = np.zeros_like(df[features].corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(16, 12))
plt.title('Pearson Correlation Matrix',fontsize=25)

sns.heatmap(df[features].corr(),linewidths=0.25,vmax=0.7,square=True,cmap="BuGn", #"BuGn_r" to reverse
            linecolor='w',annot=True,annot_kws={"size":8},mask=mask,cbar_kws={"shrink": .9});
```



Data Preprocessing

Preprocessing data can significantly enhance model accuracy and bolster its reliability. While it's not a guaranteed improvement, being mindful of the features and employing appropriate inputs often leads to more accessible outcomes. I experimented with several data mining techniques, including transformations and normalization. However, ultimately, I chose to employ binning, resulting in the creation of a new dataframe named df_dm.

In [14]:
```
df_dm=df.copy()
df_dm.describe()
```

Out[14]:

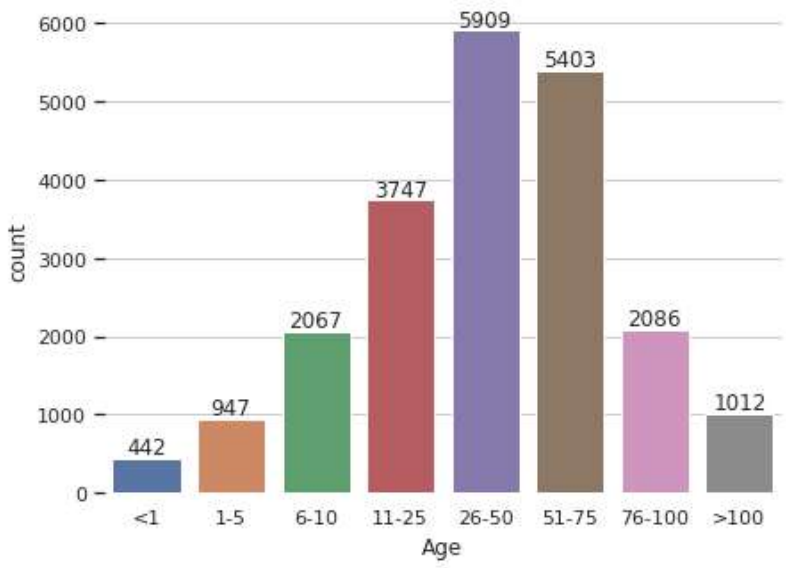
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	co
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5

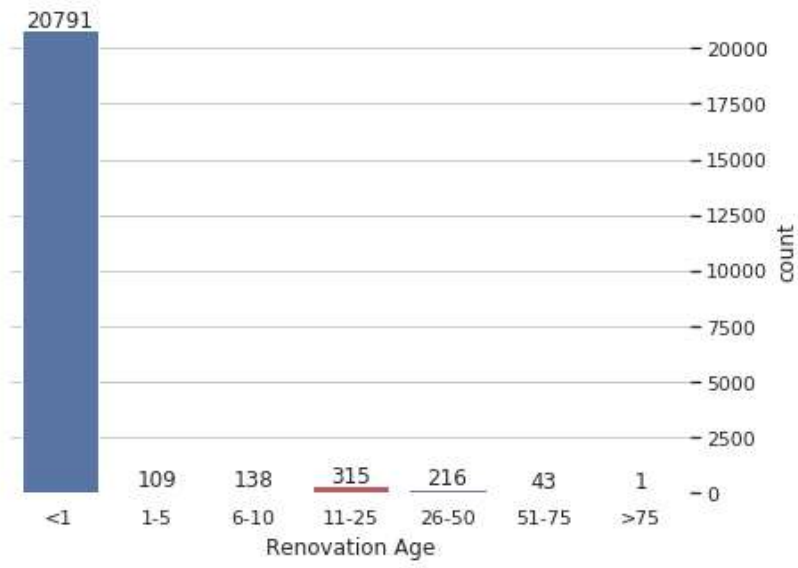
Binning

Data binning serves as a preprocessing technique aimed at mitigating the impact of minor observational errors. I found it beneficial to apply binning to columns related to 'yr_built' and 'yr_renovated' within this dataset. By doing so, I derived the ages and renovation ages of the houses at the time they were sold. Additionally, I partitioned these columns into intervals, providing a clear visualization through the histograms displayed below.

In [15]:
```
# just take the year from the date column
df_dm['sales_yr']=df_dm['date'].astype(str).str[:4]

# add the age of the buildings when the houses were sold as a new column
df_dm['age']=df_dm['sales_yr'].astype(int)-df_dm['yr_built']
# add the age of the renovation when the houses were sold as a new column
df_dm['age_rnv']=0
```

```
df_dm['age_rnv']=df_dm['sales_yr'][df_dm['yr_renovated']!=0].astype(int)-df_dm['yr_renovated'][df_dm['yr_renovated']!=0
df_dm['age_rnv'][df_dm['age_rnv'].isnull()]=0

# partition the age into bins
bins = [-2,0,5,10,25,50,75,100,100000]
labels = ['<1','1-5','6-10','11-25','26-50','51-75','76-100','>100']
df_dm['age_binned'] = pd.cut(df_dm['age'], bins=bins, labels=labels)
# partition the age_rnv into bins
bins = [-2,0,5,10,25,50,75,100000]
labels = ['<1','1-5','6-10','11-25','26-50','51-75','>75']
df_dm['age_rnv_binned'] = pd.cut(df_dm['age_rnv'], bins=bins, labels=labels)

# histograms for the binned columns
f, axes = plt.subplots(1, 2,figsize=(15,5))
p1=sns.countplot(df_dm['age_binned'],ax=axes[0])
for p in p1.patches:
    height = p.get_height()
    p1.text(p.get_x()+p.get_width()/2,height + 50,height,ha="center")
p2=sns.countplot(df_dm['age_rnv_binned'],ax=axes[1])
sns.despine(left=True, bottom=True)
for p in p2.patches:
    height = p.get_height()
    p2.text(p.get_x()+p.get_width()/2,height + 200,height,ha="center")

axes[0].set(xlabel='Age')
axes[0].yaxis.tick_left()
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].set(xlabel='Renovation Age');

# transform the factor values to be able to use in the model
df_dm = pd.get_dummies(df_dm, columns=['age_binned','age_rnv_binned'])
```





Multiple Regression

When we have **more than one** feature in a linear regression, it is defined as **multiple regression**. Then, it is time to create some complex models.

Multiple Regression - 1

I identified the features initially by reviewing previous sections and incorporated them into my initial multiple linear regression. Similar to the simple regression, I printed the coefficients utilized by the model for predictions. However, this time, if we intend to manually compute predictions, we need to use the following definition.

$$h_\theta(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

In [16]:
```
train_data_dm,test_data_dm = train_test_split(df_dm,train_size = 0.8,random_state=3)

features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','zipcode']
complex_model_1 = linear_model.LinearRegression()
complex_model_1.fit(train_data_dm[features],train_data_dm['price'])

print('Intercept: {}'.format(complex_model_1.intercept_))
print('Coefficients: {}'.format(complex_model_1.coef_))

pred = complex_model_1.predict(test_data_dm[features])
rmsecm = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred)),'.3f'))
rtrcm = float(format(complex_model_1.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm = float(format(adjustedR2(complex_model_1.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sh
rtecm = float(format(complex_model_1.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm = float(format(adjustedR2(complex_model_1.score(test_data_dm[features],test_data['price']),test_data_dm.shape[0]
cv = float(format(cross_val_score(complex_model_1,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['Multiple Regression-1','selected features',rmsecm,rtrcm,artrcm,rtecm,artecm,cv]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

```
Intercept: -57221293.13485905
Coefficients: [-5.68950279e+04  1.13310062e+04  3.18389287e+02 -2.90807628e-01
 -5.79609821e+03  5.84022824e+02]
```

Out[16]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

Multiple Regression - 2

Expanding upon the previous subsection, I augmented the features list with additional variables. Similarly, I displayed the coefficients of the model, akin to the previous subsection. Upon analyzing the evaluation metrics, there was a notable and significant improvement.

In [17]:
```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view',
            'grade','age_binned_<1', 'age_binned_1-5', 'age_binned_6-10','age_binned_11-25',
            'age_binned_26-50', 'age_binned_51-75','age_binned_76-100', 'age_binned_>100',
            'zipcode']
complex_model_2 = linear_model.LinearRegression()
complex_model_2.fit(train_data_dm[features],train_data_dm['price'])

print('Intercept: {}'.format(complex_model_2.intercept_))
print('Coefficients: {}'.format(complex_model_2.coef_))

pred = complex_model_2.predict(test_data_dm[features])
rmsecm = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred)),'.3f'))
rtrcm = float(format(complex_model_2.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm = float(format(adjustedR2(complex_model_2.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sh
rtecm = float(format(complex_model_2.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm = float(format(adjustedR2(complex_model_2.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shape
cv = float(format(cross_val_score(complex_model_2,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['Multiple Regression-2','selected features',rmsecm,rtrcm,artrcm,rtecm,artecm,cv]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

```
Intercept: 14932064.456709543
Coefficients: [-3.74523328e+04  4.83495326e+04  1.71684976e+02 -2.31081061e-01
  1.03590806e+04  5.56285921e+05  4.78399848e+04  1.24143045e+05
 -8.88123227e+04 -1.05756567e+05 -1.04723750e+05 -1.35898725e+05
 -5.37336956e+04  8.41048129e+04  1.84153081e+05  2.20667166e+05
 -1.60046391e+02]
```

Out[17]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
2	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

Multiple Regression - 3

To facilitate easy observation of differences, I constructed a model utilizing all features without any preprocessing steps. Remarkably, there was a significant improvement in the evaluation metrics once again.

In [18]:
```
features = ['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']
complex_model_3 = linear_model.LinearRegression()
complex_model_3.fit(train_data[features],train_data['price'])

print('Intercept: {}'.format(complex_model_3.intercept_))
print('Coefficients: {}'.format(complex_model_3.coef_))

pred = complex_model_3.predict(test_data[features])
rmsecm = float(format(np.sqrt(metrics.mean_squared_error(test_data['price'],pred)),'.3f'))
rtrcm = float(format(complex_model_3.score(train_data[features],train_data['price']),'.3f'))
artrcm = float(format(adjustedR2(complex_model_3.score(train_data[features],train_data['price']),train_data.shape[0],le
rtecm = float(format(complex_model_3.score(test_data[features],test_data['price']),'.3f'))
artecm = float(format(adjustedR2(complex_model_3.score(test_data[features],test_data['price']),test_data.shape[0],len(f
cv = float(format(cross_val_score(complex_model_3,df[features],df['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['Multiple Regression-3','all features, no preprocessing',rmsecm,rtrcm,artrcm,rtecm,artecm,cv]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

```
Intercept: 7580919.940385307
Coefficients: [-3.51323305e+04  4.25821114e+04  1.10705020e+02  1.13581822e-01
  6.82992716e+03  5.61794985e+05  5.28174040e+04  2.48918356e+04
  9.57708783e+04  7.01998427e+01  4.05051778e+01 -2.70948034e+03
  2.26715091e+01 -5.80427853e+02  5.98629230e+05 -2.08875497e+05
  2.32857416e+01 -3.75353459e-01]
```

Out[18]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
3	Multiple Regression-3	all features, no preprocessing	193693.989	0.698	0.697	0.708	0.707	0.695
2	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

Multiple Regression - 4

This time I used the data obtained after preprocessing step.

In [19]:
```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront',
            'view','condition','grade','sqft_above','sqft_basement','age_binned_<1',
            'age_binned_1-5', 'age_binned_6-10','age_binned_11-25', 'age_binned_26-50',
            'age_binned_51-75','age_binned_76-100', 'age_binned_>100','age_rnv_binned_<1',
            'age_rnv_binned_1-5', 'age_rnv_binned_6-10', 'age_rnv_binned_11-25',
            'age_rnv_binned_26-50', 'age_rnv_binned_51-75', 'age_rnv_binned_>75',
            'zipcode','lat','long','sqft_living15','sqft_lot15']
complex_model_4 = linear_model.LinearRegression()
complex_model_4.fit(train_data_dm[features],train_data_dm['price'])

print('Intercept: {}'.format(complex_model_4.intercept_))
print('Coefficients: {}'.format(complex_model_4.coef_))

pred = complex_model_4.predict(test_data_dm[features])
rmsecm = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred)),'.3f'))
rtrcm = float(format(complex_model_4.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm = float(format(adjustedR2(complex_model_4.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sh
rtecm = float(format(complex_model_4.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm = float(format(adjustedR2(complex_model_4.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shape
cv = float(format(cross_val_score(complex_model_4,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['Multiple Regression-4','all features',rmsecm,rtrcm,artrcm,rtecm,artecm,cv]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

```
Intercept: 8748434.76427494
Coefficients: [-3.33491904e+04  3.76549641e+04  1.10716252e+02  1.22826592e-01
 -1.26725956e+04  5.69817402e+05  5.41386091e+04  3.17275550e+04
  9.52300581e+04  7.05366154e+01  4.01794569e+01 -4.46861874e+04
 -5.43838963e+04 -7.11287025e+04 -8.93583274e+04 -5.92594439e+04
  3.70943651e+04  1.22837694e+05  1.58884499e+05 -1.97812401e+04
  9.43034022e+04  8.74164248e+04  4.14131931e+04 -1.50309593e+04
 -1.06990366e+05 -8.13304552e+04 -6.50061210e+02  6.03335811e+05
 -2.10031732e+05  2.42386074e+01 -3.08651553e-01]
```

Out[19]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
4	Multiple Regression-4	all features	191879.550	0.701	0.7	0.713	0.711	0.698
3	Multiple Regression-3	all features, no preprocessing	193693.989	0.698	0.697	0.708	0.707	0.695
2	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

Regularization

Regularization aims to combat both overfitting and underfitting issues in models. Overfitting, characterized by high variance, often results from an overly complex function that introduces unnecessary curves and angles unrelated to the data. While this function fits the training data well, it might perform poorly on the test set. Conversely, underfitting entails low variance, reflecting a very simple model that can also yield subpar results.

Remedies for these issues include manual feature adjustments or employing model selection algorithms, which can add extra workload. However, regularization presents an alternative approach where all features remain, and the model adjusts the θj coefficients. This method is particularly effective when dealing with numerous slightly useful features.

There are two widely used regularization types: Ridge and Lasso Regressions. Deciding between them depends on the nature of the variables:

Many small to medium-sized effects favor the use of Ridge regression. For scenarios with only a few variables carrying medium to large effects, Lasso regression is more suitable.

Ridge Regression

Ridge regression is called **L2 regularization** and by adding a penalty, we obtain the below equation

$$RSS_{RIDGE} = \sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \alpha \sum_{j=1}^{n} \theta_j^2$$

By changing the α value, we can control the amount of the regularization. When we increase α, regularization increases and the opposite is valid too. As a result of this, I selected different α values and used a linear regression without regularization in order to observe the differences easily.

In [20]:
```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront',
            'view','condition','grade','sqft_above','sqft_basement','age_binned_<1',
            'age_binned_1-5', 'age_binned_6-10','age_binned_11-25', 'age_binned_26-50',
            'age_binned_51-75','age_binned_76-100', 'age_binned_>100','age_rnv_binned_<1',
            'age_rnv_binned_1-5', 'age_rnv_binned_6-10', 'age_rnv_binned_11-25',
            'age_rnv_binned_26-50', 'age_rnv_binned_51-75', 'age_rnv_binned_>75',
            'zipcode','lat','long','sqft_living15','sqft_lot15']
complex_model_R = linear_model.Ridge(alpha=1)
complex_model_R.fit(train_data_dm[features],train_data_dm['price'])

pred1 = complex_model_R.predict(test_data_dm[features])
rmsecm1 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred1)),'.3f'))
rtrcm1 = float(format(complex_model_R.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm1 = float(format(adjustedR2(complex_model_R.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sl
rtecm1 = float(format(complex_model_R.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm1 = float(format(adjustedR2(complex_model_R.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shap
cv1 = float(format(cross_val_score(complex_model_R,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

complex_model_R = linear_model.Ridge(alpha=100)
complex_model_R.fit(train_data_dm[features],train_data_dm['price'])

pred2 = complex_model_R.predict(test_data_dm[features])
rmsecm2 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred2)),'.3f'))
rtrcm2 = float(format(complex_model_R.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm2 = float(format(adjustedR2(complex_model_R.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sl
rtecm2 = float(format(complex_model_R.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm2 = float(format(adjustedR2(complex_model_R.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shap
cv2 = float(format(cross_val_score(complex_model_R,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

complex_model_R = linear_model.Ridge(alpha=1000)
complex_model_R.fit(train_data_dm[features],train_data_dm['price'])

pred3 = complex_model_R.predict(test_data_dm[features])
rmsecm3 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred3)),'.3f'))
rtrcm3 = float(format(complex_model_R.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm3 = float(format(adjustedR2(complex_model_R.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sl
rtecm3 = float(format(complex_model_R.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm3 = float(format(adjustedR2(complex_model_R.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shap
cv3 = float(format(cross_val_score(complex_model_R,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['Ridge Regression','alpha=1, all features',rmsecm1,rtrcm1,artrcm1,rtecm1,artecm1,cv1]
evaluation.loc[r+1] = ['Ridge Regression','alpha=100, all features',rmsecm2,rtrcm2,artrcm2,rtecm2,artecm2,cv2]
evaluation.loc[r+2] = ['Ridge Regression','alpha=1000, all features',rmsecm3,rtrcm3,artrcm3,rtecm3,artecm3,cv3]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

Out[20]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
4	Multiple Regression-4	all features	191879.550	0.701	0.7	0.713	0.711	0.698
5	Ridge Regression	alpha=1, all features	191903.548	0.701	0.7	0.713	0.711	0.698
3	Multiple Regression-3	all features, no preprocessing	193693.989	0.698	0.697	0.708	0.707	0.695
6	Ridge Regression	alpha=100, all features	195372.495	0.694	0.693	0.703	0.701	0.691
2	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
7	Ridge Regression	alpha=1000, all features	209625.468	0.651	0.65	0.658	0.655	0.648
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

Lasso Regression

Lasso regression is called **L1 regularization** and it is defined as

$$RSS_{LASSO} = \sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \alpha \sum_{j=1}^{n} |\theta_j|$$

The main difference between ridge and lasso is the penalty but α works the same way.

In [21]:
```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront',
            'view','condition','grade','sqft_above','sqft_basement','age_binned_<1',
            'age_binned_1-5', 'age_binned_6-10','age_binned_11-25', 'age_binned_26-50',
            'age_binned_51-75','age_binned_76-100', 'age_binned_>100','age_rnv_binned_<1',
            'age_rnv_binned_1-5', 'age_rnv_binned_6-10', 'age_rnv_binned_11-25',
            'age_rnv_binned_26-50', 'age_rnv_binned_51-75', 'age_rnv_binned_>75',
            'zipcode','lat','long','sqft_living15','sqft_lot15']
complex_model_L = linear_model.Lasso(alpha=1)
complex_model_L.fit(train_data_dm[features],train_data_dm['price'])

pred1 = complex_model_L.predict(test_data_dm[features])
rmsecm1 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred1)),'.3f'))
rtrcm1 = float(format(complex_model_L.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm1 = float(format(adjustedR2(complex_model_L.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sl
rtecm1 = float(format(complex_model_L.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm1 = float(format(adjustedR2(complex_model_L.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shap
cv1 = float(format(cross_val_score(complex_model_L,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

complex_model_L = linear_model.Lasso(alpha=100)
complex_model_L.fit(train_data_dm[features],train_data_dm['price'])

pred2 = complex_model_L.predict(test_data_dm[features])
rmsecm2 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred2)),'.3f'))
rtrcm2 = float(format(complex_model_L.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm2 = float(format(adjustedR2(complex_model_L.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sl
rtecm2 = float(format(complex_model_L.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm2 = float(format(adjustedR2(complex_model_L.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shap
cv2 = float(format(cross_val_score(complex_model_L,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

complex_model_L = linear_model.Lasso(alpha=1000)
complex_model_L.fit(train_data_dm[features],train_data_dm['price'])

pred3 = complex_model_L.predict(test_data_dm[features])
rmsecm3 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred3)),'.3f'))
rtrcm3 = float(format(complex_model_L.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrcm3 = float(format(adjustedR2(complex_model_L.score(train_data_dm[features],train_data_dm['price']),train_data_dm.sl
rtecm3 = float(format(complex_model_L.score(test_data_dm[features],test_data_dm['price']),'.3f'))
artecm3 = float(format(adjustedR2(complex_model_L.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shap
cv3 = float(format(cross_val_score(complex_model_L,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['Lasso Regression','alpha=1, all features',rmsecm1,rtrcm1,artrcm1,rtecm1,artecm1,cv1]
evaluation.loc[r+1] = ['Lasso Regression','alpha=100, all features',rmsecm2,rtrcm2,artrcm2,rtecm2,artecm2,cv2]
evaluation.loc[r+2] = ['Lasso Regression','alpha=1000, all features',rmsecm3,rtrcm3,artrcm3,rtecm3,artecm3,cv3]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

Out[21]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
4	Multiple Regression-4	all features	191879.550	0.701	0.7	0.713	0.711	0.698
5	Ridge Regression	alpha=1, all features	191903.548	0.701	0.7	0.713	0.711	0.698
8	Lasso Regression	alpha=1, all features	191880.918	0.701	0.7	0.713	0.711	0.698
9	Lasso Regression	alpha=100, all features	192060.144	0.701	0.7	0.713	0.711	0.698
3	Multiple Regression-3	all features, no preprocessing	193693.989	0.698	0.697	0.708	0.707	0.695
10	Lasso Regression	alpha=1000, all features	193587.943	0.697	0.697	0.708	0.706	0.695
6	Ridge Regression	alpha=100, all features	195372.495	0.694	0.693	0.703	0.701	0.691
2	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
7	Ridge Regression	alpha=1000, all features	209625.468	0.651	0.65	0.658	0.655	0.648
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491

Polynomial Regression

The fundamental concept behind linear models is to fit a straight line to our data. Yet, when the data exhibits a quadratic distribution, opting for a quadratic function and employing polynomial transformation might yield improved results. This adjustment leads to the hypothesis function being defined as:

$$h_\theta(X) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$$

Due to numerous variations available for polynomial regression, I've presented the results in a new table, clearly demonstrating that polynomial transformation significantly enhanced the model fit. However, when employing polynomial transformations and selecting the degree, caution is crucial to prevent overfitting. As observed in the table below, certain models exhibit overfitting. Despite having very high R-squared values for the training set, these models display negative or low 5-fold cross-validation metrics, indicating potential overfitting issues.

In [22]:
```
evaluation_poly = pd.DataFrame({'Model': [],
                                'Details':[],
                                'Root Mean Squared Error (RMSE)':[],
                                'R-squared (training)':[],
                                'Adjusted R-squared (training)':[],
                                'R-squared (test)':[],
                                'Adjusted R-squared (test)':[],
                                '5-Fold Cross Validation':[]})

features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view',
            'grade','yr_built','zipcode']
polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df[features])
X_trainpoly = polyfeat.fit_transform(train_data[features])
X_testpoly = polyfeat.fit_transform(test_data[features])
poly = linear_model.LinearRegression().fit(X_trainpoly, train_data['price'])

pred1 = poly.predict(X_testpoly)
rmsepoly1 = float(format(np.sqrt(metrics.mean_squared_error(test_data['price'],pred1)),'.3f'))
rtrpoly1 = float(format(poly.score(X_trainpoly,train_data['price']),'.3f'))
rtepoly1 = float(format(poly.score(X_testpoly,test_data['price']),'.3f'))
cv1 = float(format(cross_val_score(linear_model.LinearRegression(),X_allpoly,df['price'],cv=5).mean(),'.3f'))

polyfeat = PolynomialFeatures(degree=3)
X_allpoly = polyfeat.fit_transform(df[features])
X_trainpoly = polyfeat.fit_transform(train_data[features])
X_testpoly = polyfeat.fit_transform(test_data[features])
poly = linear_model.LinearRegression().fit(X_trainpoly, train_data['price'])

pred2 = poly.predict(X_testpoly)
rmsepoly2 = float(format(np.sqrt(metrics.mean_squared_error(test_data['price'],pred2)),'.3f'))
rtrpoly2 = float(format(poly.score(X_trainpoly,train_data['price']),'.3f'))
rtepoly2 = float(format(poly.score(X_testpoly,test_data['price']),'.3f'))
cv2 = float(format(cross_val_score(linear_model.LinearRegression(),X_allpoly,df['price'],cv=5).mean(),'.3f'))

features = ['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']
polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df[features])
X_trainpoly = polyfeat.fit_transform(train_data[features])
X_testpoly = polyfeat.fit_transform(test_data[features])
poly = linear_model.LinearRegression().fit(X_trainpoly, train_data['price'])

pred3 = poly.predict(X_testpoly)
rmsepoly3 = float(format(np.sqrt(metrics.mean_squared_error(test_data['price'],pred3)),'.3f'))
rtrpoly3 = float(format(poly.score(X_trainpoly,train_data['price']),'.3f'))
rtepoly3 = float(format(poly.score(X_testpoly,test_data['price']),'.3f'))
cv3 = float(format(cross_val_score(linear_model.LinearRegression(),X_allpoly,df['price'],cv=5).mean(),'.3f'))

polyfeat = PolynomialFeatures(degree=3)
X_allpoly = polyfeat.fit_transform(df[features])
X_trainpoly = polyfeat.fit_transform(train_data[features])
X_testpoly = polyfeat.fit_transform(test_data[features])
poly = linear_model.LinearRegression().fit(X_trainpoly, train_data['price'])

pred4 = poly.predict(X_testpoly)
rmsepoly4 = float(format(np.sqrt(metrics.mean_squared_error(test_data['price'],pred4)),'.3f'))
rtrpoly4 = float(format(poly.score(X_trainpoly,train_data['price']),'.3f'))
rtepoly4 = float(format(poly.score(X_testpoly,test_data['price']),'.3f'))
cv4 = float(format(cross_val_score(linear_model.LinearRegression(),X_allpoly,df['price'],cv=5).mean(),'.3f'))


features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront',
            'view','condition','grade','sqft_above','sqft_basement','age_binned_<1',
            'age_binned_1-5', 'age_binned_6-10','age_binned_11-25', 'age_binned_26-50',
            'age_binned_51-75','age_binned_76-100', 'age_binned_>100','age_rnv_binned_<1',
            'age_rnv_binned_1-5', 'age_rnv_binned_6-10', 'age_rnv_binned_11-25',
            'age_rnv_binned_26-50', 'age_rnv_binned_51-75', 'age_rnv_binned_>75',
            'zipcode','lat','long','sqft_living15','sqft_lot15']
polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df_dm[features])
X_trainpoly = polyfeat.fit_transform(train_data_dm[features])
X_testpoly = polyfeat.fit_transform(test_data_dm[features])
poly = linear_model.LinearRegression().fit(X_trainpoly, train_data['price'])

pred5 = poly.predict(X_testpoly)
rmsepoly5 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred5)),'.3f'))
rtrpoly5 = float(format(poly.score(X_trainpoly,train_data_dm['price']),'.3f'))
rtepoly5 = float(format(poly.score(X_testpoly,test_data_dm['price']),'.3f'))
cv5 = float(format(cross_val_score(linear_model.LinearRegression(),X_allpoly,df_dm['price'],cv=5).mean(),'.3f'))

polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df_dm[features])
X_trainpoly = polyfeat.fit_transform(train_data_dm[features])
X_testpoly = polyfeat.fit_transform(test_data_dm[features])
poly = linear_model.Ridge(alpha=1).fit(X_trainpoly, train_data['price'])
```

```
pred6 = poly.predict(X_testpoly)
rmsepoly6 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred6)),'.3f'))
rtrpoly6 = float(format(poly.score(X_trainpoly,train_data_dm['price']),'.3f'))
rtepoly6 = float(format(poly.score(X_testpoly,test_data_dm['price']),'.3f'))
cv6 = float(format(cross_val_score(linear_model.Ridge(alpha=1),X_allpoly,df_dm['price'],cv=5).mean(),'.3f'))

polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df_dm[features])
X_trainpoly = polyfeat.fit_transform(train_data_dm[features])
X_testpoly = polyfeat.fit_transform(test_data_dm[features])
poly = linear_model.Ridge(alpha=50000).fit(X_trainpoly, train_data['price'])

pred7 = poly.predict(X_testpoly)
rmsepoly7 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred7)),'.3f'))
rtrpoly7 = float(format(poly.score(X_trainpoly,train_data_dm['price']),'.3f'))
rtepoly7 = float(format(poly.score(X_testpoly,test_data_dm['price']),'.3f'))
cv7 = float(format(cross_val_score(linear_model.Ridge(alpha=50000),X_allpoly,df_dm['price'],cv=5).mean(),'.3f'))

polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df_dm[features])
X_trainpoly = polyfeat.fit_transform(train_data_dm[features])
X_testpoly = polyfeat.fit_transform(test_data_dm[features])
poly = linear_model.Lasso(alpha=1).fit(X_trainpoly, train_data['price'])

pred8 = poly.predict(X_testpoly)
rmsepoly8 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred8)),'.3f'))
rtrpoly8 = float(format(poly.score(X_trainpoly,train_data_dm['price']),'.3f'))
rtepoly8 = float(format(poly.score(X_testpoly,test_data_dm['price']),'.3f'))
cv8 = float(format(cross_val_score(linear_model.Lasso(alpha=1),X_allpoly,df_dm['price'],cv=5).mean(),'.3f'))
```

```
polyfeat = PolynomialFeatures(degree=2)
X_allpoly = polyfeat.fit_transform(df_dm[features])
X_trainpoly = polyfeat.fit_transform(train_data_dm[features])
X_testpoly = polyfeat.fit_transform(test_data_dm[features])
poly = linear_model.Lasso(alpha=50000).fit(X_trainpoly, train_data['price'])

pred9 = poly.predict(X_testpoly)
rmsepoly9 = float(format(np.sqrt(metrics.mean_squared_error(test_data_dm['price'],pred9)),'.3f'))
rtrpoly9 = float(format(poly.score(X_trainpoly,train_data_dm['price']),'.3f'))
rtepoly9 = float(format(poly.score(X_testpoly,test_data_dm['price']),'.3f'))
cv9 = float(format(cross_val_score(linear_model.Lasso(alpha=50000),X_allpoly,df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation_poly.shape[0]
evaluation_poly.loc[r] = ['Polynomial Regression','degree=2, selected features, no preprocessing',rmsepoly1,rtrpoly1,'-
evaluation_poly.loc[r+1] = ['Polynomial Regression','degree=3, selected features, no preprocessing',rmsepoly2,rtrpoly2,
evaluation_poly.loc[r+2] = ['Polynomial Regression','degree=2, all features, no preprocessing',rmsepoly3,rtrpoly3,'-',r
evaluation_poly.loc[r+3] = ['Polynomial Regression','degree=3, all features, no preprocessing',rmsepoly4,rtrpoly4,'-',r
evaluation_poly.loc[r+4] = ['Polynomial Regression','degree=2, all features',rmsepoly5,rtrpoly5,'-',rtepoly5,'-',cv5]
evaluation_poly.loc[r+5] = ['Polynomial Ridge Regression','alpha=1, degree=2, all features',rmsepoly6,rtrpoly6,'-',rtep
evaluation_poly.loc[r+6] = ['Polynomial Ridge Regression','alpha=50000, degree=2, all features',rmsepoly7,rtrpoly7,'-',
evaluation_poly.loc[r+7] = ['Polynomial Lasso Regression','alpha=1, degree=2, all features',rmsepoly8,rtrpoly8,'-',rtep
evaluation_poly.loc[r+8] = ['Polynomial Lasso Regression','alpha=50000, degree=2, all features',rmsepoly9,rtrpoly9,'-',
evaluation_poly_temp = evaluation_poly[['Model','Details','Root Mean Squared Error (RMSE)','R-squared (training)','R-sq
evaluation_poly_temp.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

Out[22]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	R-squared (test)	5-Fold Cross Validation
2	Polynomial Regression	degree=2, all features, no preprocessing	151160.750	0.830	0.822	0.813
6	Polynomial Ridge Regression	alpha=50000, degree=2, all features	159872.572	0.810	0.801	0.791
8	Polynomial Lasso Regression	alpha=50000, degree=2, all features	166020.484	0.797	0.785	0.779
7	Polynomial Lasso Regression	alpha=1, degree=2, all features	166195.984	0.807	0.785	0.778
0	Polynomial Regression	degree=2, selected features, no preprocessing	191074.809	0.730	0.716	0.714
1	Polynomial Regression	degree=3, selected features, no preprocessing	188769.232	0.749	0.722	0.159
3	Polynomial Regression	degree=3, all features, no preprocessing	194010.205	0.870	0.707	-0.899
5	Polynomial Ridge Regression	alpha=1, degree=2, all features	150177.231	0.838	0.824	-3200.410
4	Polynomial Regression	degree=2, all features	151618.469	0.840	0.821	-9020.616

k-NN Regression

I introduced k-NN regression into this analysis, although I didn't anticipate achieving a strong result with it. Additionally, k-NN doesn't offer significant insight; it's a straightforward method, resembling k-NN classification. Essentially, it computes the weighted average, median, or another chosen statistic from the k-nearest instances.

The evaluation metrics for various k values across training and test sets are presented in the table below. As anticipated, k-NN didn't yield the expected level of success.

In [23]:
```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront',
            'view','condition','grade','sqft_above','sqft_basement','age_binned_<1',
            'age_binned_1-5', 'age_binned_6-10','age_binned_11-25', 'age_binned_26-50',
            'age_binned_51-75','age_binned_76-100', 'age_binned_>100','age_rnv_binned_<1',
            'age_rnv_binned_1-5', 'age_rnv_binned_6-10', 'age_rnv_binned_11-25',
            'age_rnv_binned_26-50', 'age_rnv_binned_51-75', 'age_rnv_binned_>75',
            'zipcode','lat','long','sqft_living15','sqft_lot15']
knnreg = KNeighborsRegressor(n_neighbors=15)
knnreg.fit(train_data_dm[features],train_data_dm['price'])
pred = knnreg.predict(test_data_dm[features])

rmseknn1 = float(format(np.sqrt(metrics.mean_squared_error(y_test,pred)),'.3f'))
rtrknn1 = float(format(knnreg.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrknn1 = float(format(adjustedR2(knnreg.score(train_data_dm[features],train_data_dm['price']),train_data_dm.shape[0],
rteknn1 = float(format(knnreg.score(test_data_dm[features],test_data_dm['price']),'.3f'))
arteknn1 = float(format(adjustedR2(knnreg.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shape[0],len
cv1 = float(format(cross_val_score(knnreg,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

knnreg = KNeighborsRegressor(n_neighbors=25)
knnreg.fit(train_data_dm[features],train_data_dm['price'])
pred = knnreg.predict(test_data_dm[features])

rmseknn2 = float(format(np.sqrt(metrics.mean_squared_error(y_test,pred)),'.3f'))
rtrknn2 = float(format(knnreg.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrknn2 = float(format(adjustedR2(knnreg.score(train_data_dm[features],train_data_dm['price']),train_data_dm.shape[0],
rteknn2 = float(format(knnreg.score(test_data_dm[features],test_data_dm['price']),'.3f'))
arteknn2 = float(format(adjustedR2(knnreg.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shape[0],len
cv2 = float(format(cross_val_score(knnreg,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

knnreg = KNeighborsRegressor(n_neighbors=27)
knnreg.fit(train_data_dm[features],train_data_dm['price'])
pred = knnreg.predict(test_data_dm[features])

rmseknn3 = float(format(np.sqrt(metrics.mean_squared_error(y_test,pred)),'.3f'))
rtrknn3 = float(format(knnreg.score(train_data_dm[features],train_data_dm['price']),'.3f'))
artrknn3 = float(format(adjustedR2(knnreg.score(train_data_dm[features],train_data_dm['price']),train_data_dm.shape[0],
rteknn3 = float(format(knnreg.score(test_data_dm[features],test_data_dm['price']),'.3f'))
arteknn3 = float(format(adjustedR2(knnreg.score(test_data_dm[features],test_data_dm['price']),test_data_dm.shape[0],len
cv3 = float(format(cross_val_score(knnreg,df_dm[features],df_dm['price'],cv=5).mean(),'.3f'))

r = evaluation.shape[0]
evaluation.loc[r] = ['KNN Regression','k=15, all features',rmseknn1,rtrknn1,artrknn1,rteknn1,arteknn1,cv1]
evaluation.loc[r+1] = ['KNN Regression','k=25, all features',rmseknn2,rtrknn2,artrknn2,rteknn2,arteknn2,cv2]
evaluation.loc[r+2] = ['KNN Regression','k=27, all features',rmseknn3,rtrknn3,artrknn3,rteknn3,arteknn3,cv3]
evaluation.sort_values(by = '5-Fold Cross Validation', ascending=False)
```

Out[23]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
4	Multiple Regression-4	all features	191879.550	0.701	0.7	0.713	0.711	0.698
5	Ridge Regression	alpha=1, all features	191903.548	0.701	0.7	0.713	0.711	0.698
8	Lasso Regression	alpha=1, all features	191880.918	0.701	0.7	0.713	0.711	0.698
9	Lasso Regression	alpha=100, all features	192060.144	0.701	0.7	0.713	0.711	0.698
3	Multiple Regression-3	all features, no preprocessing	193693.989	0.698	0.697	0.708	0.707	0.695
10	Lasso Regression	alpha=1000, all features	193587.943	0.697	0.697	0.708	0.706	0.695
6	Ridge Regression	alpha=100, all features	195372.495	0.694	0.693	0.703	0.701	0.691
2	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
7	Ridge Regression	alpha=1000, all features	209625.468	0.651	0.65	0.658	0.655	0.648
1	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
11	KNN Regression	k=15, all features	242834.420	0.562	0.561	0.541	0.537	0.496
0	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491
12	KNN Regression	k=25, all features	247032.235	0.529	0.528	0.525	0.521	0.487
13	KNN Regression	k=27, all features	247414.263	0.523	0.522	0.523	0.52	0.486

Evaluation Table

In [24]:
```
evaluation_temp=evaluation.append(evaluation_poly)
evaluation_temp1=evaluation_temp.sort_values(by = '5-Fold Cross Validation', ascending=False)
evaluation_temp2=evaluation_temp1.reset_index()
evaluation_f=evaluation_temp2.iloc[:,1:]
evaluation_f
```

Out[24]:

	Model	Details	Root Mean Squared Error (RMSE)	R-squared (training)	Adjusted R-squared (training)	R-squared (test)	Adjusted R-squared (test)	5-Fold Cross Validation
0	Polynomial Regression	degree=2, all features, no preprocessing	151160.750	0.830	-	0.822	-	0.813
1	Polynomial Ridge Regression	alpha=50000, degree=2, all features	159872.572	0.810	-	0.801	-	0.791
2	Polynomial Lasso Regression	alpha=50000, degree=2, all features	166020.484	0.797	-	0.785	-	0.779
3	Polynomial Lasso Regression	alpha=1, degree=2, all features	166195.984	0.807	-	0.785	-	0.778
4	Polynomial Regression	degree=2, selected features, no preprocessing	191074.809	0.730	-	0.716	-	0.714
5	Multiple Regression-4	all features	191879.550	0.701	0.7	0.713	0.711	0.698
6	Ridge Regression	alpha=1, all features	191903.548	0.701	0.7	0.713	0.711	0.698
7	Lasso Regression	alpha=1, all features	191880.918	0.701	0.7	0.713	0.711	0.698
8	Lasso Regression	alpha=100, all features	192060.144	0.701	0.7	0.713	0.711	0.698
9	Multiple Regression-3	all features, no preprocessing	193693.989	0.698	0.697	0.708	0.707	0.695
10	Lasso Regression	alpha=1000, all features	193587.943	0.697	0.697	0.708	0.706	0.695
11	Ridge Regression	alpha=100, all features	195372.495	0.694	0.693	0.703	0.701	0.691
12	Multiple Regression-2	selected features	209712.753	0.652	0.652	0.657	0.656	0.648
13	Ridge Regression	alpha=1000, all features	209625.468	0.651	0.65	0.658	0.655	0.648
14	Multiple Regression-1	selected features	248514.011	0.514	0.514	0.519	0.518	0.512
15	KNN Regression	k=15, all features	242834.420	0.562	0.561	0.541	0.537	0.496
16	Simple Linear Regression	-	254289.149	0.492	-	0.496	-	0.491
17	KNN Regression	k=25, all features	247032.235	0.529	0.528	0.525	0.521	0.487
18	KNN Regression	k=27, all features	247414.263	0.523	0.522	0.523	0.52	0.486
19	Polynomial Regression	degree=3, selected features, no preprocessing	188769.232	0.749	-	0.722	-	0.159
20	Polynomial Regression	degree=3, all features, no preprocessing	194010.205	0.870	-	0.707	-	-0.899
21	Polynomial Ridge Regression	alpha=1, degree=2, all features	150177.231	0.838	-	0.824	-	-3200.410
22	Polynomial Regression	degree=2, all features	151618.469	0.840	-	0.821	-	-9020.616

Conclusion

Upon reviewing the evaluation table, the 2nd degree polynomial model (utilizing all features without preprocessing) stands out as the best performer. However, I harbor doubts regarding its reliability. Personally, I lean towards the polynomial ridge regression (with alpha=50000, degree=2, and incorporating all features), yet other models might prove useful depending on the circumstances.

Throughout this task, I've utilized scikit-learn, benefiting from its built-in functions.