Import all tools needed

In [1]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from numpy import asarray
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import mean_squared_error as mse
from scipy.stats import zscore
from scipy.stats import stats
```

Read in data frame ks_house_data

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

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	

21597 rows × 21 columns

Copy and paste the list of columns given for future reference

Column Names and descriptions for Kings County Data Set

- **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

Create new dataframe with specific column names given above name it new_df

In [3]:
```
new_df = df[['id','price', 'sqft_living', 'grade', 'sqft_above', 'sqft_living15'
new_df
```

Out[3]:

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
0	7129300520	221900.0	1180	7	1180	1340	1.00	0.0	
1	6414100192	538000.0	2570	7	2170	1690	2.25	0.0	
2	5631500400	180000.0	770	6	770	2720	1.00	0.0	
3	2487200875	604000.0	1960	7	1050	1360	3.00	0.0	
4	1954400510	510000.0	1680	8	1680	1800	2.00	0.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	360000.0	1530	8	1530	1530	2.50	0.0	
21593	6600060120	400000.0	2310	8	2310	1830	2.50	0.0	
21594	1523300141	402101.0	1020	7	1020	1020	0.75	0.0	
21595	291310100	400000.0	1600	8	1600	1410	2.50	0.0	
21596	1523300157	325000.0	1020	7	1020	1020	0.75	0.0	

21597 rows × 12 columns

Look up all the correlations of each column to each other and print to investigate

In [4]:
```
corr = new_df.corr()
corr
```

Out[4]:

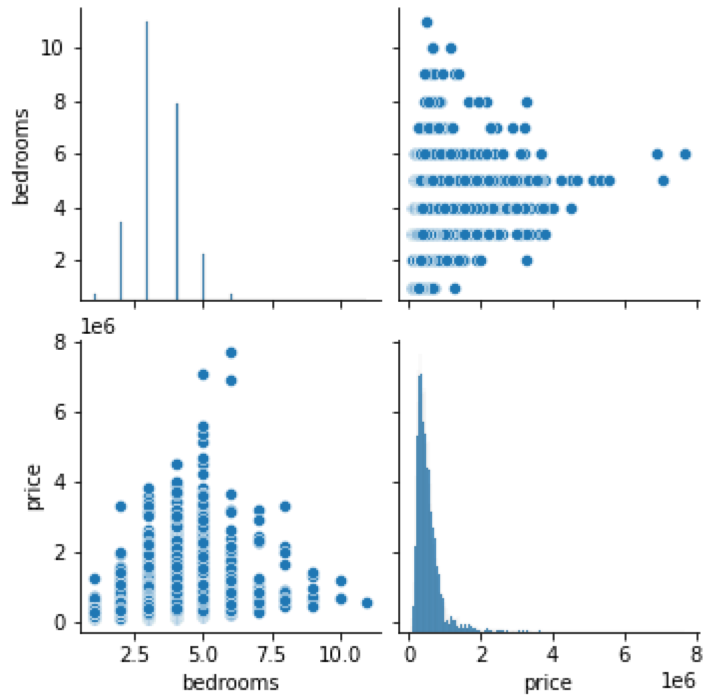
	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	
id	1.000000	-0.016772	-0.012241	0.008188	-0.010799	-0.002701	0.005162	0
price	-0.016772	1.000000	0.701917	0.667951	0.605368	0.585241	0.525906	0.
sqft_living	-0.012241	0.701917	1.000000	0.762779	0.876448	0.756402	0.755758	0.
grade	0.008188	0.667951	0.762779	1.000000	0.756073	0.713867	0.665838	0.
sqft_above	-0.010799	0.605368	0.876448	0.756073	1.000000	0.731767	0.686668	0.
sqft_living15	-0.002701	0.585241	0.756402	0.713867	0.731767	1.000000	0.569884	0
bathrooms	0.005162	0.525906	0.755758	0.665838	0.686668	0.569884	1.000000	0
view	0.011592	0.395734	0.282532	0.249727	0.166299	0.279561	0.186451	1.
bedrooms	0.001150	0.308787	0.578212	0.356563	0.479386	0.393406	0.514508	0.
lat	-0.001798	0.306692	0.052155	0.113575	-0.001199	0.048679	0.024280	0
waterfront	-0.004176	0.276295	0.110230	0.087383	0.075463	0.088860	0.067282	0.
floors	0.018608	0.256804	0.353953	0.458794	0.523989	0.280102	0.502582	0.

Drop all null values We had to drop the index 15856 because the encoder could not work with differnet a number of values in the test and training data, and since it was only a single entry, it wont affect us badly when dropped. Dropping this allowed the encoder to run and improve our model by a ton

In [5]:
```
new_df = new_df.dropna()
new_df = new_df.drop(labels=15856, axis=0)
```

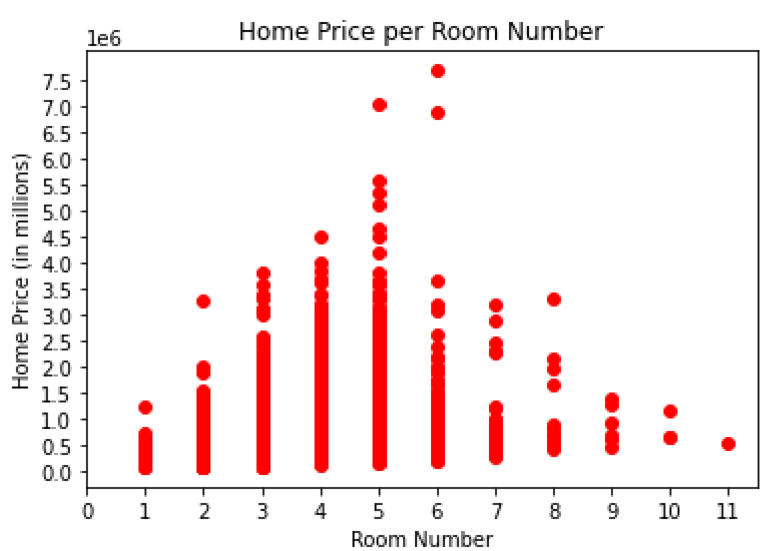
After investigating the correlations, we wanted to look at how the number of bedrooms effect the home prices. For better visualization, we created a pair plot.

In [6]:
```
import seaborn as sns

g = sns.pairplot(new_df, vars=["bedrooms", "price"])

import matplotlib.pyplot as plt
plt.show()
```



Becuase the pairplots above are not very interpretive, we created a more visually appealing scatter plot. As the plot shows, if a house has over 6 bedrooms, the price is no longer majorly affected.

In [7]:
```
plt.scatter(new_df.bedrooms, new_df.price, color='red')
plt.xlabel('Room Number')
plt.ylabel('Home Price (in millions)')
plt.title('Home Price per Room Number')
plt.xticks(np.arange(0, 12, 1))
plt.yticks(np.arange(0, 8000000, 500000))
plt.show()
plt.savefig('homepriceperroom.png')
#Create scatter plot to analayze the price differences between number of bedroom
```




```
<Figure size 432x288 with 0 Axes>
```

Call on first dataframe. Make sure to drop null values and index 15856 to ensure the encoder works correctly

In [8]:
```
df = df.dropna()
df = df.drop(labels=15856, axis=0)
df
```

Out[8]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
5	7237550310	5/12/2014	1230000.0	4	4.50	5420	101930	1.0	
6	1321400060	6/27/2014	257500.0	3	2.25	1715	6819	2.0	
...	...	...	...	...	...	...	...	...	
21591	2997800021	2/19/2015	475000.0	3	2.50	1310	1294	2.0	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	

15761 rows × 21 columns

Turn 'date' column from a string to an int to make future manipulations easier

In [9]:
```
df["date"] = df.date.apply(lambda x: x[-4:])
type(df['date'])
```

Out[9]:
```
pandas.core.series.Series
```

Turn all '?' into NaN values. Then drop all NaN values because null values decrease the models accuracy

In [10]:
```
df = df.replace('?',np.nan)

df = df.dropna()
```

Drop index 3220 because it caused issues while running train test split

In [11]:
```
df = df.drop(labels=3220, axis=0)
```

Make a linear regression model with price compared to everything else within the dataframe df (without id or price)

In [12]:
```
reg = LinearRegression()
y = df['price']
X = df.drop(['id', 'price'], axis = 1)
```

Split the data into a test and train group in order to compare how well our model functions compared to the test

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

OneHotEncode view, grade, zipcode because they are categorical values that are easier

for the model to interpret.

In [14]:
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[['view', 'grade', 'zipcode']])
transformed_train = encoder.transform(X_train[['view', 'grade', 'zipcode']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade', 'zipcode'], axis = 1
test_condition = encoder.transform(X_test[['view', 'grade', 'zipcode']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade', 'zipcode'], axis = 1),
```

Fitting and scoring the training data to see how the model works on data it has been trained for

In [15]:
```
reg.fit(X_train_encoded, y_train)
reg.score(X_train_encoded, y_train)
```

Out[15]: 0.840624332728187

Fitting and scoring the testing data to see how the model works on data its being tested against

In [16]:
```
reg.score(test_condition, y_test)
```

Out[16]: 0.8145009320259429

Getting score predictions for the training and testing data

In [17]:
```
train_preds = reg.predict(X_train_encoded)
test_preds = reg.predict(test_condition)
```

Getting the RMSE for the training data

In [18]:
```
np.sqrt(mse(y_train, train_preds))
```

Out[18]: 143663.9785123915

Getting RMSE for the testing data

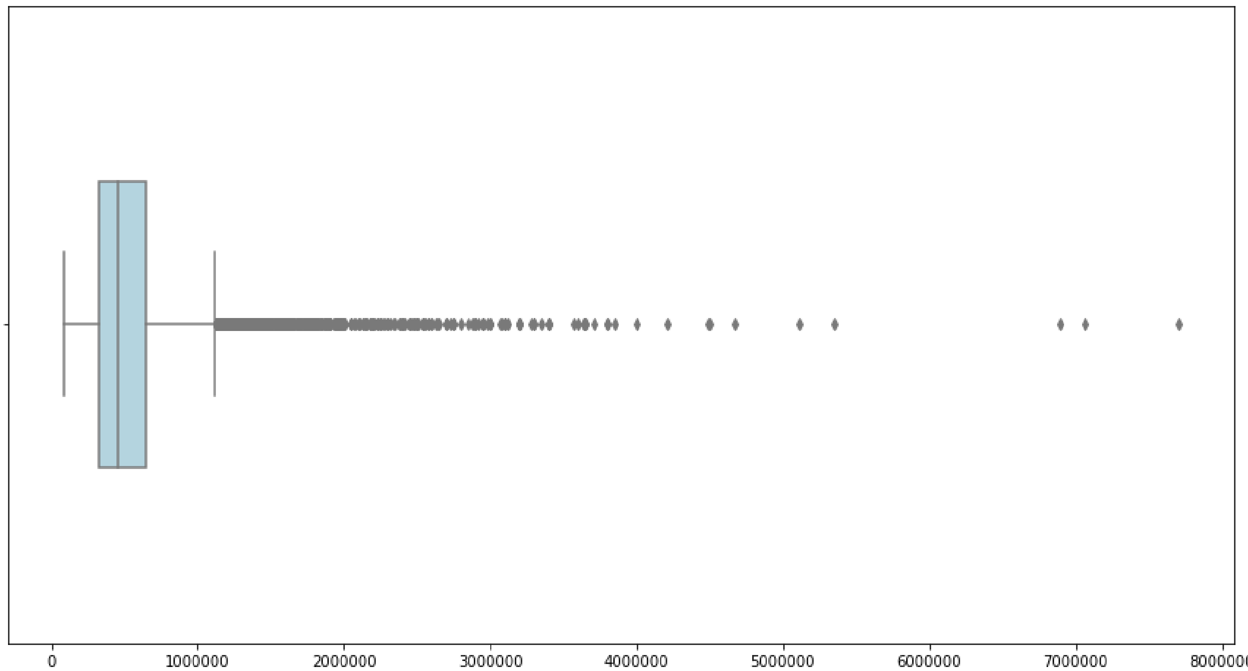
In [19]:
```
mse(y_test, test_preds, squared=False)
```

Out[19]: 176476.39588833158

Box and whiskers made for price before outliers removed

In [20]:
```
fig, ax = plt.subplots(figsize=(15,8))
plt.ticklabel_format(style="plain")
sns.boxplot(x=df['price'], width = .45, color = 'lightblue');
plt.suptitle("Before Outliers Taken Out", fontsize=20)
plt.xlabel("Price in U.S. Dollars", fontsize=14)
plt.savefig('beforeoutliers.png')
```

Before Outliers Taken Out



Making a copy of the dataframe so we have something to freely change

In [21]:
```
df_2 = df.copy()
df_2
```

Out[21]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
1	6414100192	2014	538000.0	3	2.25	2570	7242	2.0	
3	2487200875	2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2015	510000.0	3	2.00	1680	8080	1.0	
5	7237550310	2014	1230000.0	4	4.50	5420	101930	1.0	
8	2414600126	2015	229500.0	3	1.00	1780	7470	1.0	
...	...	...	...	...	...	...	...	...	
21591	2997800021	2015	475000.0	3	2.50	1310	1294	2.0	
21592	263000018	2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	2014	402101.0	2	0.75	1020	1350	2.0	
21596	1523300157	2014	325000.0	2	0.75	1020	1076	2.0	

15427 rows × 21 columns

Removing outleirs

In [22]:
```
q_low = df_2["price"].quantile(0.01)
q_hi  = df_2["price"].quantile(0.99)

df_filtered = df_2[(df_2["price"] < q_hi) & (df_2["price"] > q_low)]
```

Dropping a single entry that broke the model

In [23]:
```
df_filtered = df_filtered.drop(labels=5446, axis=0)
```

Second linear regression following the same exact steps as above with slightly different data

**The filtered price column is the new data here"

In [24]:
```
reg_2 = LinearRegression()
y = df_filtered['price']
X = df_filtered.drop(['id', 'price'], axis = 1)
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

In [26]:
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[['view', 'grade', 'zipcode']])
transformed_train = encoder.transform(X_train[['view', 'grade', 'zipcode']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade', 'zipcode'], axis = 1
test_condition = encoder.transform(X_test[['view', 'grade', 'zipcode']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade', 'zipcode'], axis = 1),
```

In [27]:
```
reg.fit(X_train_encoded, y_train)
reg.score(X_train_encoded, y_train)
```

Out[27]: 0.8492893553644953

In [28]:
```
reg.score(test_condition, y_test)
```

Out[28]: 0.8556678891117264

In [29]:
```
train_preds = reg.predict(X_train_encoded)
test_preds = reg.predict(test_condition)
```

In [30]:
```
np.sqrt(mse(y_train, train_preds))
```

Out[30]: 109665.08353490481

In [31]:
```
mse(y_test, test_preds, squared=False)
```

Out[31]: 109613.52076837461

Making a new copy again to freely change

In [32]:
```
df_3 = df_filtered.copy()
```

In [33]:
```
reg_3 = LinearRegression()
y = df_3['price']
X = df_3.drop(['id', 'price',], axis = 1)
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

In [35]:
```
encoder = OneHotEncoder(sparse=False)
```

```
encoder.fit(X_train[['view', 'grade', 'zipcode']])
transformed_train = encoder.transform(X_train[['view', 'grade', 'zipcode']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade', 'zipcode'], axis = 1
test_condition = encoder.transform(X_test[['view', 'grade', 'zipcode']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade', 'zipcode'], axis = 1),
```

In [36]:
```
reg.fit(X_train_encoded, y_train)
reg.score(X_train_encoded, y_train)
```

Out[36]: 0.8492893553644953

In [37]:
```
reg.score(test_condition, y_test)
```

Out[37]: 0.8556678891117264

In [38]:
```
train_preds = reg.predict(X_train_encoded)
test_preds = reg.predict(test_condition)
```

In [39]:
```
np.sqrt(mse(y_train, train_preds))
```

Out[39]: 109665.08353490481

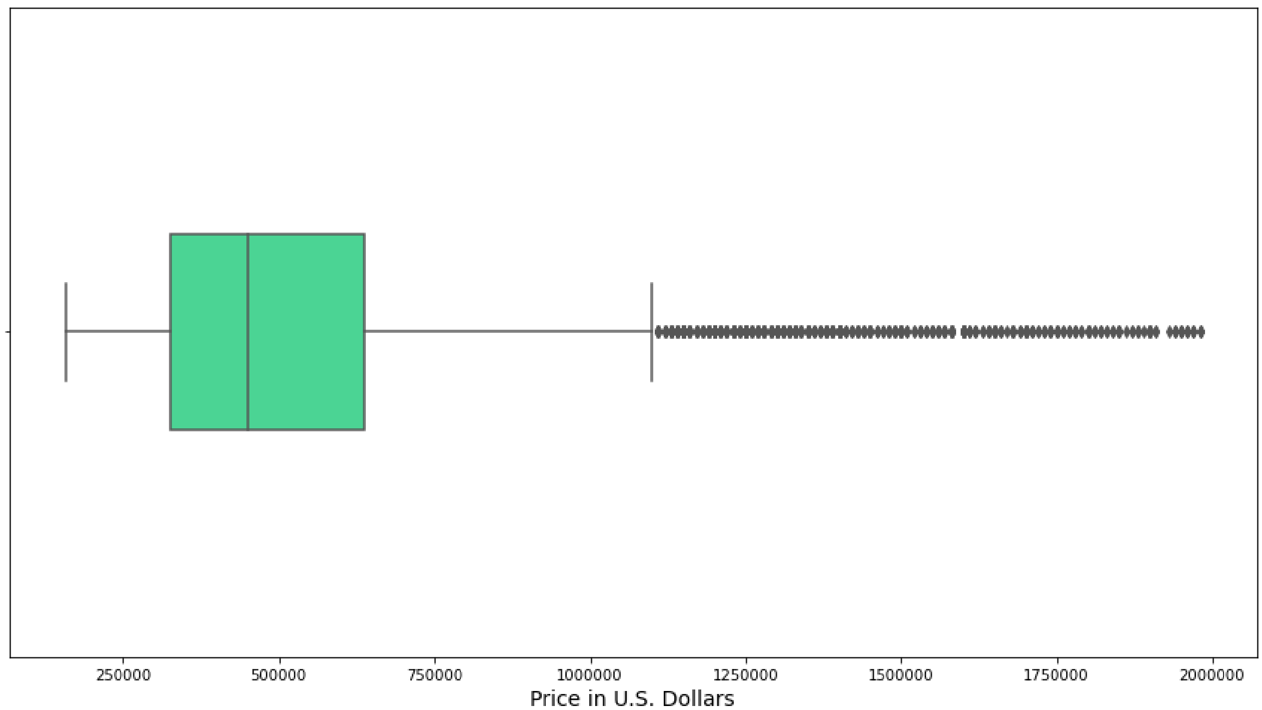
In [40]:
```
mse(y_test, test_preds, squared=False)
```

Out[40]: 109613.52076837461

Making an new graph to visulize the difference after outliers removed

In [41]:
```
fig, ax = plt.subplots(figsize=(15,8))
plt.ticklabel_format(style="plain")
sns.boxplot(x=df_3['price'], width = .3, color = '#34eb95');
plt.suptitle("After Outliers Taken Out", fontsize=25)
plt.xlabel("Price in U.S. Dollars", fontsize=14)
plt.savefig('afteroutliers.png')
```

After Outliers Taken Out



Finding percentage null values dropped

In [42]:
```
percent_yearrevn_dropped = 3824/21597
percent_yearrevn_dropped
```

Out[42]:
```
0.17706162892994398
```

In [43]:
```
percent_wat_dropped = 2376/21597
percent_wat_dropped
```

Out[43]:
```
0.11001527989998611
```

In [44]:
```
percent_view_dropped = 63/21597
percent_view_dropped
```

Out[44]:
```
0.0029170718155299346
```

In [45]:
```
df.condition.sort_values(ascending=False)
#checking the values on condition
```

Out[45]:
```
2363     5
12533    5
6390     5
6397     5
2374     5
        ..
3199     1
1440     1
16879    1
2221     1
3971     1
Name: condition, Length: 15427, dtype: int64
```

Baseline encoder

In [46]:
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[['view', 'grade']])
transformed_train = encoder.transform(X_train[['view', 'grade']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade'], axis = 1), transfor
test_condition = encoder.transform(X_test[['view', 'grade']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade',], axis = 1), test_cond
```

In [47]:
```
X_train_encoded
```

Out[47]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	sqft_abov
19551	2014	4	1.50	1200	10890	1.0	0.0	5	120
5967	2015	4	2.50	1700	6675	2.0	0.0	3	170
10100	2014	3	2.50	2090	6000	1.5	0.0	4	209
18819	2015	4	2.50	2130	9013	2.0	0.0	3	213
17635	2014	4	3.25	2420	4000	1.5	0.0	5	187
...	...	...	...	...	...	...	...	...	.
7423	2014	4	2.50	1840	4011	2.0	0.0	3	184
19158	2015	3	1.00	970	11963	1.0	0.0	4	97
7701	2014	2	2.00	1340	5350	1.5	0.0	3	134
1206	2014	4	1.75	2490	7834	1.0	0.0	4	124
10368	2014	4	2.50	2030	4080	1.5	0.0	4	173

11336 rows × 31 columns

Repeating the same linear regression steps from above

In [48]:
```
reg.fit(X_train_encoded, y_train)
```

Out[48]:
```
LinearRegression()
```

In [49]:
```
reg.score(X_train_encoded, y_train)
```

Out[49]:
```
0.7215520260287462
```

In [50]:
```
reg.score(test_condition, y_test)
```

Out[50]:
```
0.7230463137082672
```

In [51]:
```
train_preds = reg.predict(X_train_encoded)
test_preds = reg.predict(test_condition)
```

In [52]:
```
np.sqrt(mse(y_train, train_preds))
```

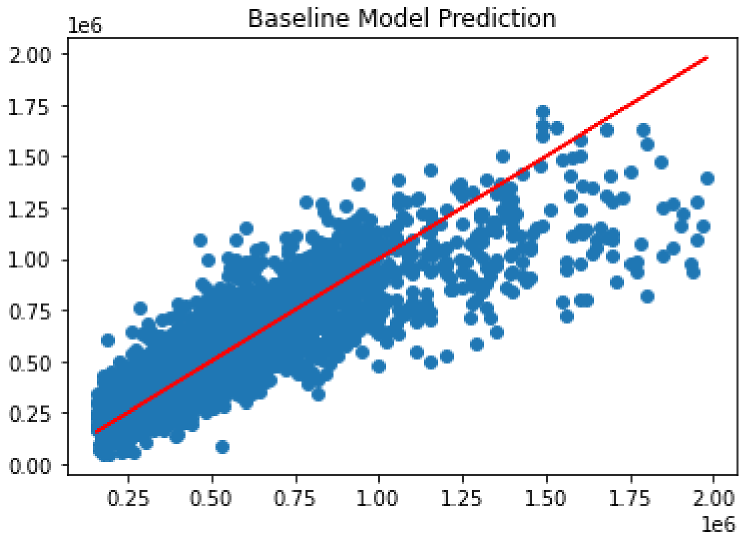
Out[52]:
```
149062.51388400592
```

In [53]:
```
mse(y_test, test_preds, squared=False)
```

Out[53]: 151840.00899197703

Making a model to show our baseline model prediction

In [54]:
```
# plot test_preds against y_test
# plot x=y line to show "perfect prediction"
x = y_test
y = test_preds
plt.scatter(x,y)
plt.title('Baseline Model Prediction')
plt.plot(y_test,y_test, color= 'red')
plt.savefig('baselinemodelprediction.png');
```



Checking list of coefs

In [55]:
```
list(zip(df.columns, reg.coef_))
```

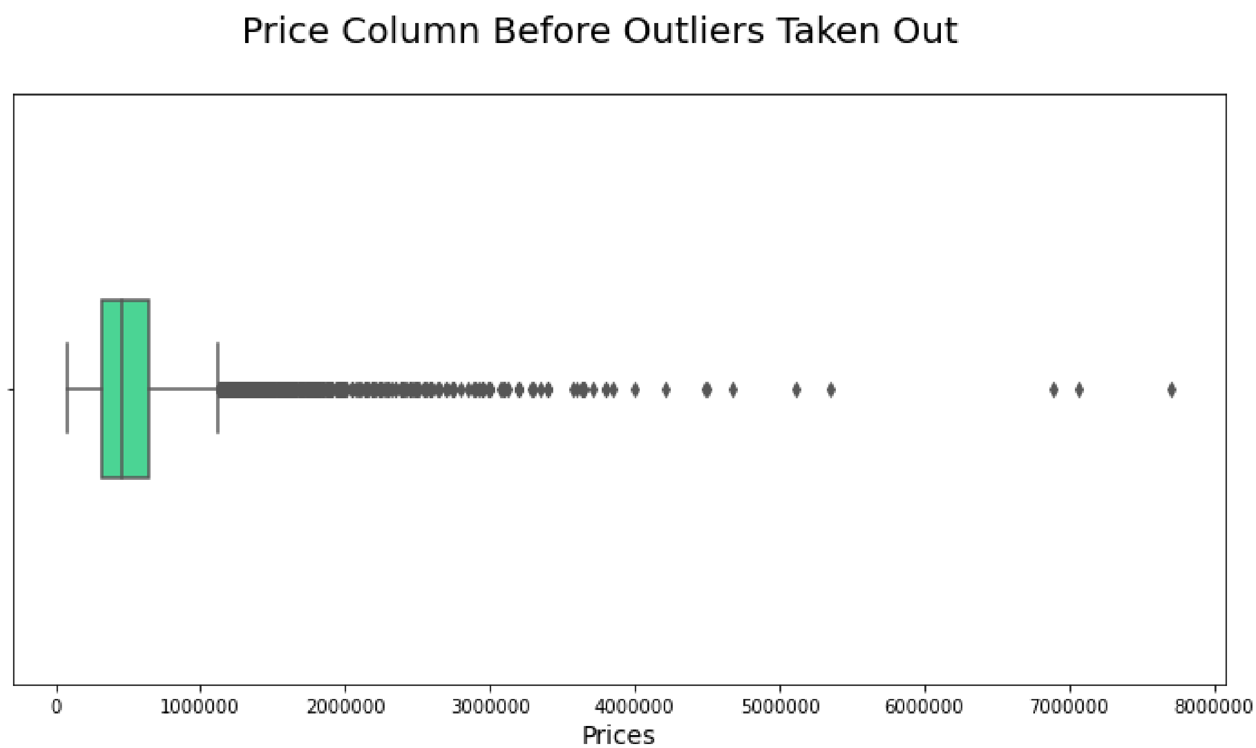
Out[55]:
```
[('id', 24030.86830464815),
 ('date', -14774.944947733886),
 ('price', 34415.09035869146),
 ('bedrooms', 66.7154081988128),
 ('bathrooms', 0.21618126139765367),
 ('sqft_living', 28312.09396977034),
 ('sqft_lot', 260139.2322417686),
 ('floors', 31161.99417888569),
 ('waterfront', 32.8710627160088),
 ('view', 33.844317375236244),
 ('condition', -1935.1438105689351),
 ('grade', 24.4783770101934),
 ('sqft_above', -470.9203008998771),
 ('sqft_basement', 597024.3327785113),
 ('yr_built', -162826.57471104764),
 ('yr_renovated', 36.46840175574007),
 ('zipcode', -0.20544624317594387),
 ('lat', -98548.97068235012),
 ('long', -4873.421737259261),
 ('sqft_living15', -19846.9817484334),
 ('sqft_lot15', 48623.88902972086)]
```

Creating a visual to show price before outliers taken out

In [56]:
```
fig, ax = plt.subplots(figsize=(12,6))
plt.ticklabel_format(style="plain")
sns.boxplot(x=df_2.price, width = .3, color = '#34eb95');
```

```
plt.suptitle("Price Column Before Outliers Taken Out", fontsize=20)
plt.xlabel("Prices", fontsize=14)
```

Out[56]: Text(0.5, 0, 'Prices')



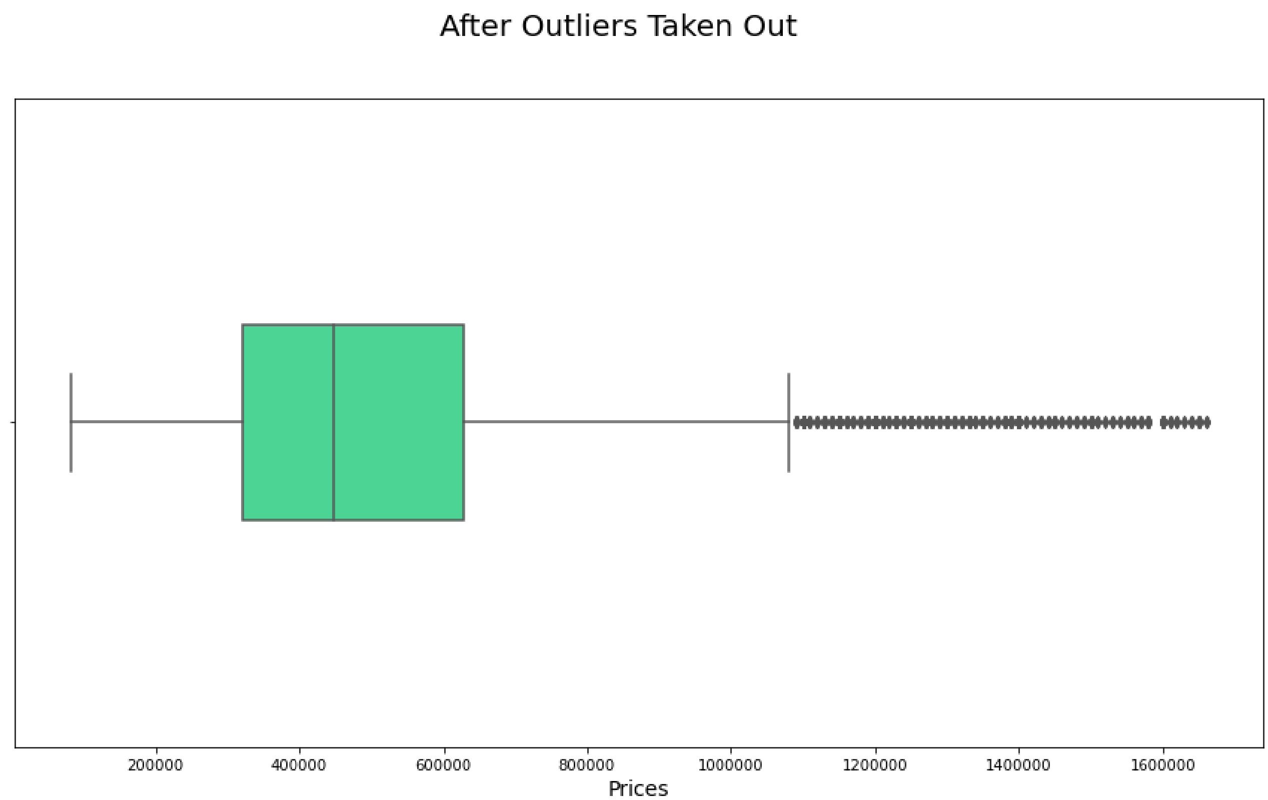
Removing outleirs

In [57]:
```
from scipy.stats import zscore
from scipy.stats import stats
z_scores = stats.zscore(df_2.price)
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3)
new_df2_price = df_2.price[filtered_entries]
new_df2_price
```

Out[57]:
```
1         538000.0
3         604000.0
4         510000.0
5        1230000.0
8         229500.0
            ...
21591     475000.0
21592     360000.0
21593     400000.0
21594     402101.0
21596     325000.0
Name: price, Length: 15151, dtype: float64
```

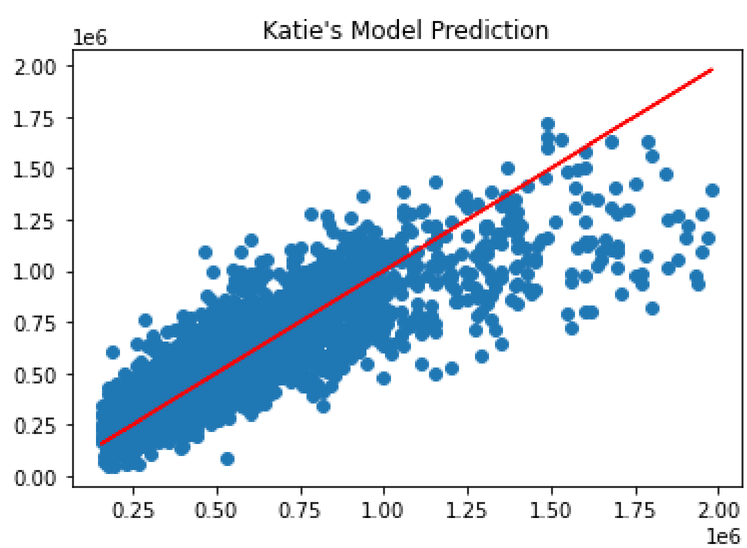
Making boxplot to visulize price after outleirs dropped

In [58]:
```
fig, ax = plt.subplots(figsize=(15,8))
plt.ticklabel_format(style="plain")
sns.boxplot(x=new_df2_price, width = .3, color = '#34eb95');
plt.suptitle("After Outliers Taken Out", fontsize=20)
plt.xlabel("Prices", fontsize=14)
plt.savefig('afteroutlierstakenout.png')
```



Katies model prediction

In [59]:
```
x = y_test
y = test_preds
plt.scatter(x,y)
plt.title("Katie's Model Prediction")
plt.plot(y_test,y_test, color= 'red');
plt.savefig('katiesmodelprediction.png')
```



Making a copy of the data frame and repeating the same steps from above on the linear regression

Price outliers dropped in this model

In [60]:
```
df2_copy = df_2[filtered_entries].copy()
```

In [61]:
```
reg_without_outliers = LinearRegression()
y = new_df2_price
X = df2_copy.drop(['id', 'price'], axis = 1)
```

In [62]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

In [63]:
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[['view', 'grade', 'zipcode']])
transformed_train = encoder.transform(X_train[['view', 'grade', 'zipcode']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade', 'zipcode'], axis = 1
test_condition = encoder.transform(X_test[['view', 'grade', 'zipcode']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade', 'zipcode'], axis = 1),
```

In [64]:
```
reg_without_outliers.fit(X_train,y_train)
```

Out[64]: LinearRegression()

In [65]:
```
reg_without_outliers.fit(X_train_encoded, y_train)
```

Out[65]: LinearRegression()

In [66]:
```
reg_without_outliers.score(X_train_encoded, y_train)
```

Out[66]: 0.8477290572835575

In [67]:
```
reg_without_outliers.score(test_condition, y_test)
```

Out[67]: 0.8473790368289617

In [68]:
```
train_preds = reg_without_outliers.predict(X_train_encoded)
test_preds = reg_without_outliers.predict(test_condition)
```

In [69]:
```
np.sqrt(mse(y_train, train_preds))
```

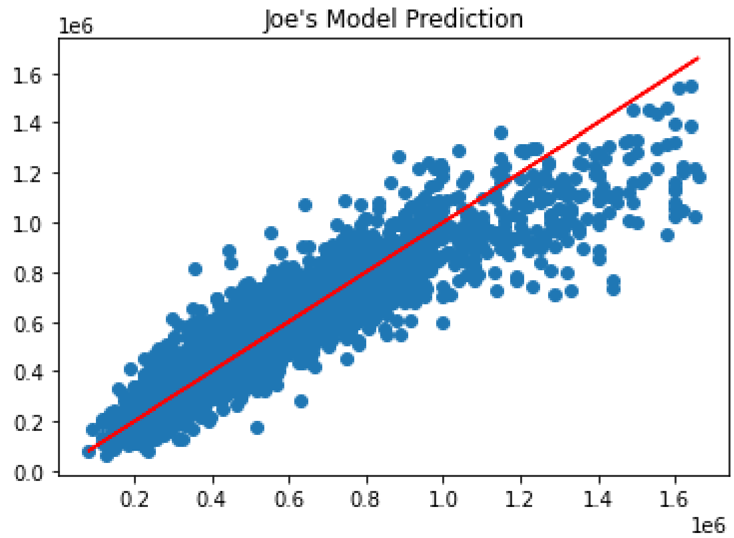
Out[69]: 101155.68187067893

In [70]:
```
mse(y_test, test_preds, squared=False)
```

Out[70]: 105763.01912913867

Joe's model prediction

In [71]:
```
x = y_test
y = test_preds
plt.scatter(x,y)
plt.title("Joe's Model Prediction")
plt.plot(y_test,y_test, color= 'red')
plt.savefig('joesmodelprediction.png');
```



Making another dataframe copy and repeating the same steps from above with new data

Dropped bedrooms in this linear regression

In [72]:
```
df3 = df2_copy
```

In [73]:
```
linreg = LinearRegression()
```

In [74]:
```
y = new_df2_price
X = df3.drop(['id', 'price', 'bedrooms'], axis = 1)
```

In [75]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

In [76]:
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[['view', 'grade', 'zipcode']])
transformed_train = encoder.transform(X_train[['view', 'grade', 'zipcode']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade', 'zipcode'], axis = 1
test_condition = encoder.transform(X_test[['view', 'grade', 'zipcode']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade', 'zipcode'], axis = 1),
```

In [77]:
```
linreg.fit(X_train,y_train)
```

Out[77]: LinearRegression()

In [78]:
```
linreg.fit(X_train_encoded, y_train)
```

Out[78]: LinearRegression()

In [79]:
```
linreg.score(X_train_encoded, y_train)
```

Out[79]: 0.8475885629639406

In [80]:
```
linreg.score(test_condition, y_test)
```

Out[80]: 0.8470426150969363

In [81]:
```
train_preds = linreg.predict(X_train_encoded)
test_preds = linreg.predict(test_condition)
```

In [82]:
```
np.sqrt(mse(y_train, train_preds))
```

Out[82]: 101202.3372659663

In [83]:
```
mse(y_test, test_preds, squared=False)
```

Out[83]: 105879.5214469478

Making another dataframe copy and repeating the same steps from above with new data

Year renovated was dropped this time

In [84]:
```
df4 = df3.copy()
```

In [85]:
```
lrm = LinearRegression()
```

In [86]:
```
y = new_df2_price
X = df4.drop(['id', 'price', 'yr_renovated'], axis = 1)
```

In [87]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

In [88]:
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[['view', 'grade', 'zipcode']])
transformed_train = encoder.transform(X_train[['view', 'grade', 'zipcode']])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(['view', 'grade', 'zipcode'], axis = 1
test_condition = encoder.transform(X_test[['view', 'grade', 'zipcode']])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(['view', 'grade', 'zipcode'], axis = 1),
```

In [89]:
```
lrm.fit(X_train,y_train)
```

Out[89]: LinearRegression()

In [90]:
```
lrm.fit(X_train_encoded, y_train)
```

Out[90]: LinearRegression()

In [91]:
```
lrm.score(test_condition, y_test)
```

Out[91]: 0.8453564587473171

In [92]:
```
train_preds = lrm.predict(X_train_encoded)
test_preds = lrm.predict(test_condition)
```

In [93]:
```
np.sqrt(mse(y_train, train_preds))
```

Out[93]: 101437.39012471554

In [94]:
```
mse(y_test, test_preds, squared=False)
```

Out[94]: 106461.51396447132

Making another dataframe copy and repeating the same steps from above with new data Bathrooms was endcoded this time

In [95]:
```
df5 = df4.copy()
```

In [96]:
```
df5 = df5.drop(labels=8537)
```

In [97]:
```
lrm = LinearRegression()
```

In [98]:
```
y = df5['price']
X = df5.drop(['id', 'price'], axis = 1)
```

In [99]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

In [100...
```
encoding_list = ['view', 'grade', 'zipcode', 'bathrooms']
```

In [101...
```
encoder = OneHotEncoder(sparse=False)
encoder.fit(X_train[encoding_list])

transformed_train = encoder.transform(X_train[encoding_list])
transformed_train = pd.DataFrame(transformed_train, columns = encoder.get_featur
X_train_encoded = pd.concat([X_train.drop(encoding_list, axis = 1), transformed_

test_condition = encoder.transform(X_test[encoding_list])
test_condition = pd.DataFrame(test_condition, columns=encoder.get_feature_names(
test_condition = pd.concat([X_test.drop(encoding_list, axis = 1), test_condition
```

In [102...
```
lrm.fit(X_train,y_train)
```

Out[102...
```
LinearRegression()
```

In [103...
```
lrm.fit(X_train_encoded, y_train)
```

Out[103...
```
LinearRegression()
```

In [104...
```
lrm.score(X_train_encoded, y_train)
```

Out[104...
```
0.8503445737189369
```

In [105...
```
lrm.score(test_condition, y_test)
```

Out[105...
```
0.846709308151527
```

In [106...
```
train_preds = lrm.predict(X_train_encoded)
test_preds = lrm.predict(test_condition)
```

In [107...
```
np.sqrt(mse(y_train, train_preds))
```

Out[107...
```
101415.64357808855
```

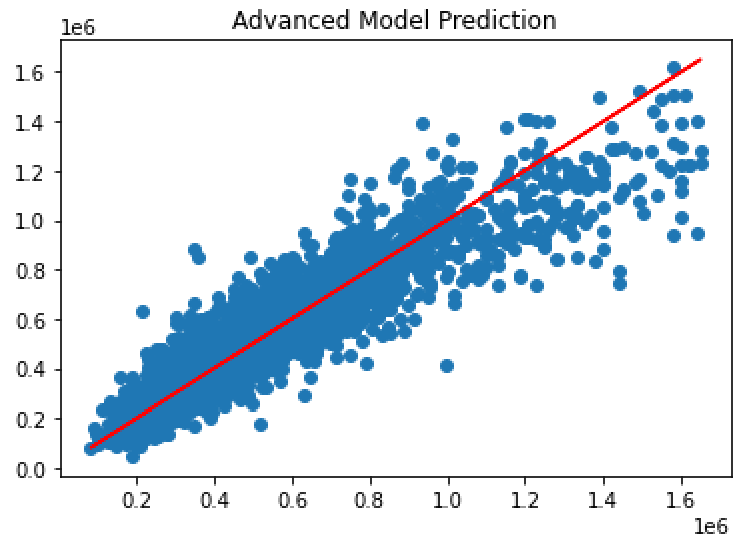
In [108...
```
mse(y_test, test_preds, squared=False)
```

Out[108...
```
102664.71233046142
```

Visulizing our advanced model prediction

In [109…

```
x = y_test
y = test_preds
plt.scatter(x,y)
plt.title('Advanced Model Prediction')
plt.plot(y_test,y_test, color= 'red')
plt.savefig('advancedmodelprediction.png');
```



Read in data again for Colette's stuff

In [110…

```
kcdf = pd.read_csv("../data/kc_house_data.csv")
kcdf
```

Out[110…

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	

21597 rows × 21 columns

We found out what the correlation is between each of the feature variables and price, then sorted the correlations from lowest to highest value

In [111…

```
corr = kcdf.corr()
```

```
corr["price"].sort_values()
```

Out[111…

```
zipcode         -0.053402
id              -0.016772
long             0.022036
condition        0.036056
yr_built         0.053953
sqft_lot15       0.082845
sqft_lot         0.089876
yr_renovated     0.129599
floors           0.256804
waterfront       0.276295
lat              0.306692
bedrooms         0.308787
view             0.395734
bathrooms        0.525906
sqft_living15    0.585241
sqft_above       0.605368
grade            0.667951
sqft_living      0.701917
price            1.000000
Name: price, dtype: float64
```

We decided to drop the columns that do not have a notable correlation with price. Below, we redefine the dataframe to only include the columns we want to keep.

In [112…

```
new_df = kcdf[['id', 'price', 'sqft_living', 'grade', 'sqft_above', 'sqft_living
new_df
```

Out[112…

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
0	7129300520	221900.0	1180	7	1180	1340	1.00	0.0	
1	6414100192	538000.0	2570	7	2170	1690	2.25	0.0	
2	5631500400	180000.0	770	6	770	2720	1.00	0.0	
3	2487200875	604000.0	1960	7	1050	1360	3.00	0.0	
4	1954400510	510000.0	1680	8	1680	1800	2.00	0.0	
...	...	...	...	...	...	...	...	...	
21592	263000018	360000.0	1530	8	1530	1530	2.50	0.0	
21593	6600060120	400000.0	2310	8	2310	1830	2.50	0.0	
21594	1523300141	402101.0	1020	7	1020	1020	0.75	0.0	
21595	291310100	400000.0	1600	8	1600	1410	2.50	0.0	
21596	1523300157	325000.0	1020	7	1020	1020	0.75	0.0	

21597 rows × 11 columns

We made sure all nulls were dropped from the dataframe

In [113…

```
new_df = new_df.dropna()
```

By subtracting squarefeet above ground from squarefeet of each house in general, we created a new column that would tell us the sqaure footage of the basement of each house.

In [114…

```
new_df = new_df.assign(sqft_basement = new_df['sqft_living'] - new_df['sqft_abov
new_df
```

Out[114…

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
1	6414100192	538000.0	2570	7	2170	1690	2.25	0.0	
2	5631500400	180000.0	770	6	770	2720	1.00	0.0	
3	2487200875	604000.0	1960	7	1050	1360	3.00	0.0	
4	1954400510	510000.0	1680	8	1680	1800	2.00	0.0	
5	7237550310	1230000.0	5420	11	3890	4760	4.50	0.0	
...	...	...	...	...	...	...	...	...	
21591	2997800021	475000.0	1310	8	1180	1330	2.50	0.0	
21592	263000018	360000.0	1530	8	1530	1530	2.50	0.0	
21593	6600060120	400000.0	2310	8	2310	1830	2.50	0.0	
21594	1523300141	402101.0	1020	7	1020	1020	0.75	0.0	
21596	1523300157	325000.0	1020	7	1020	1020	0.75	0.0	

19164 rows × 12 columns

However, we were more interested in determining whether or not each house had a basement at all. To figure this out, we turned each basement squarefootage into a boolean value and then created a new column out of these values. True means a house has a basement and False means a house has no basement.

In [115…

```
basement = []
for value in new_df["sqft_basement"]:
    if value == 0:
        basement.append("False")
    else:
        basement.append("True")

new_df["basement_bool"] = basement

new_df
```

Out[115…

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
1	6414100192	538000.0	2570	7	2170	1690	2.25	0.0	
2	5631500400	180000.0	770	6	770	2720	1.00	0.0	
3	2487200875	604000.0	1960	7	1050	1360	3.00	0.0	
4	1954400510	510000.0	1680	8	1680	1800	2.00	0.0	
5	7237550310	1230000.0	5420	11	3890	4760	4.50	0.0	
...	...	...	...	...	...	...	...	...	
21591	2997800021	475000.0	1310	8	1180	1330	2.50	0.0	
21592	263000018	360000.0	1530	8	1530	1530	2.50	0.0	
21593	6600060120	400000.0	2310	8	2310	1830	2.50	0.0	

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
21594	1523300141	402101.0	1020	7	1020	1020	0.75	0.0	
21596	1523300157	325000.0	1020	7	1020	1020	0.75	0.0	

19164 rows × 13 columns

Because the relationship between a house having or not having a basement and house price is easier to understand than the relationship between basement squarefootage and house price, we deleted the old column sqft_basement

In [116…

```
del new_df['sqft_basement']
new_df
```

Out[116…

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
1	6414100192	538000.0	2570	7	2170	1690	2.25	0.0	
2	5631500400	180000.0	770	6	770	2720	1.00	0.0	
3	2487200875	604000.0	1960	7	1050	1360	3.00	0.0	
4	1954400510	510000.0	1680	8	1680	1800	2.00	0.0	
5	7237550310	1230000.0	5420	11	3890	4760	4.50	0.0	
...	...	...	...	...	...	...	...	...	
21591	2997800021	475000.0	1310	8	1180	1330	2.50	0.0	
21592	263000018	360000.0	1530	8	1530	1530	2.50	0.0	
21593	6600060120	400000.0	2310	8	2310	1830	2.50	0.0	
21594	1523300141	402101.0	1020	7	1020	1020	0.75	0.0	
21596	1523300157	325000.0	1020	7	1020	1020	0.75	0.0	

19164 rows × 12 columns

Boolean values cannot be included in a heatmap, so we dropped the basement column for our heatmap

In [117…

```
no_bool_df = new_df.drop('basement_bool', axis='columns')
no_bool_df
```

Out[117…

	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
1	6414100192	538000.0	2570	7	2170	1690	2.25	0.0	
2	5631500400	180000.0	770	6	770	2720	1.00	0.0	
3	2487200875	604000.0	1960	7	1050	1360	3.00	0.0	
4	1954400510	510000.0	1680	8	1680	1800	2.00	0.0	
5	7237550310	1230000.0	5420	11	3890	4760	4.50	0.0	
...	...	...	...	...	...	...	...	...	
21591	2997800021	475000.0	1310	8	1180	1330	2.50	0.0	

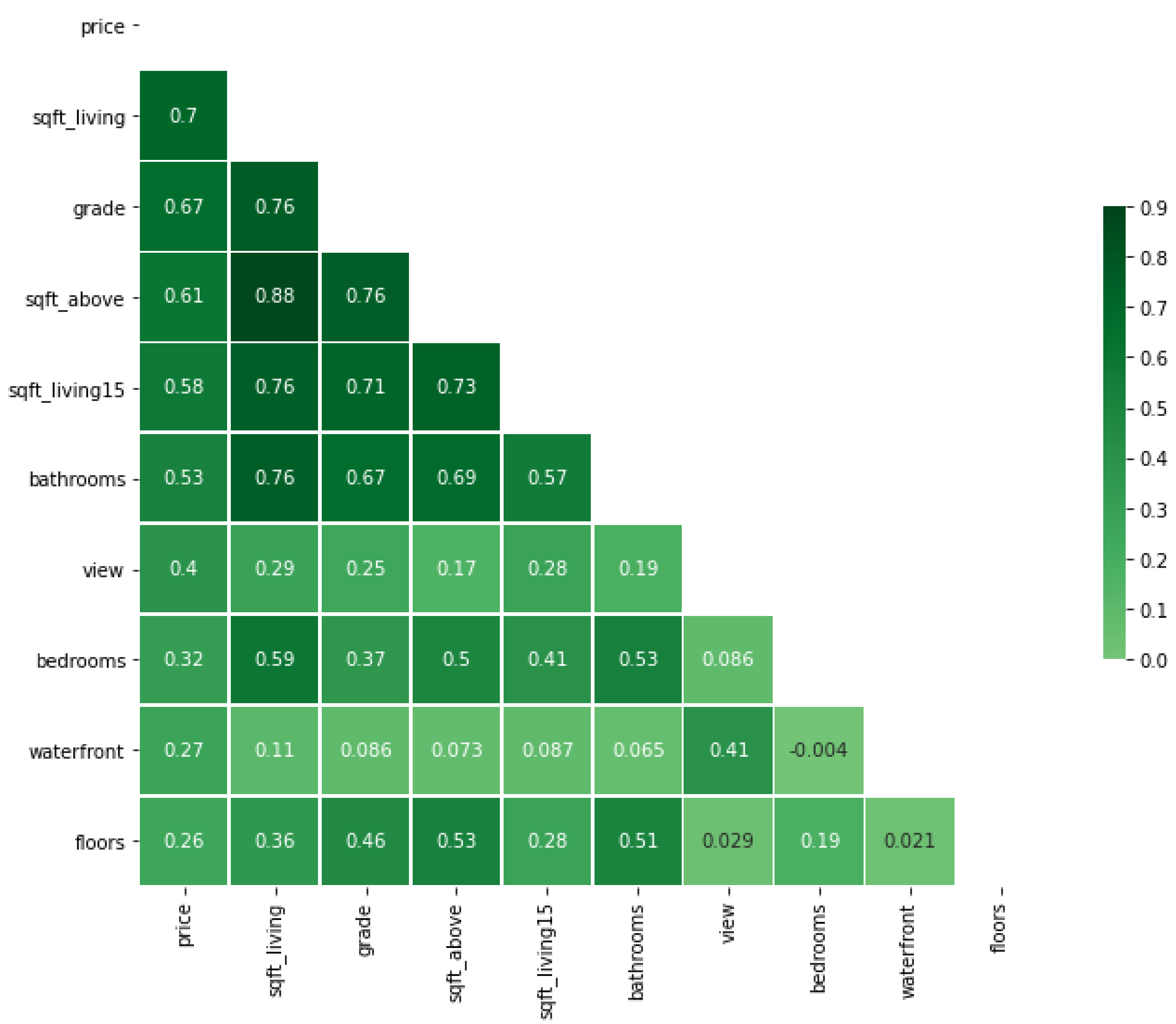
	id	price	sqft_living	grade	sqft_above	sqft_living15	bathrooms	view	be
21592	263000018	360000.0	1530	8	1530	1530	2.50	0.0	
21593	6600060120	400000.0	2310	8	2310	1830	2.50	0.0	
21594	1523300141	402101.0	1020	7	1020	1020	0.75	0.0	
21596	1523300157	325000.0	1020	7	1020	1020	0.75	0.0	

19164 rows × 11 columns

Then, we made the heatmap

In [118…

```
heatmap_df = no_bool_df.drop(columns='id')
heatmap_df = heatmap_df.dropna()
heatmap_df = heatmap_df.drop(labels=15856, axis=0)
heatmap_df.corr()
mask = np.triu(np.ones_like(heatmap_df.corr(), dtype=bool))
f, ax = plt.subplots(figsize=(11, 9))
cmap = "Greens"
sns.heatmap(heatmap_df.corr(), mask=mask, cmap=cmap, vmin=0, vmax=.9, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
plt.savefig('heatmap.png');
```



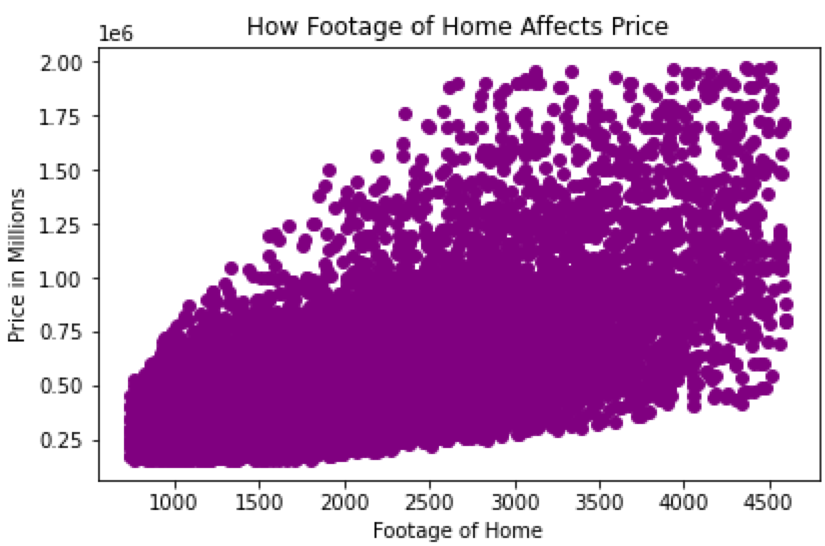
We made a visualization of the correlation between price and sqft_living because sqft_living is the variable most strongly correlated with price

In [119…

```
pricelow = heatmap_df["price"].quantile(0.01)
pricehigh  = heatmap_df["price"].quantile(0.99)

pricefiltered = heatmap_df[(heatmap_df["price"] < pricehigh) & (heatmap_df["pric

livinglow = pricefiltered["sqft_living"].quantile(0.01)
livinghigh  = pricefiltered["sqft_living"].quantile(0.99)

livingfiltered = pricefiltered[(pricefiltered["sqft_living"] < livinghigh) & (pr
```

In [120…

```
s = livingfiltered["sqft_living"]
p = livingfiltered["price"]
fig = plt.figure(figsize=(5, 3))
ax = fig.add_axes([0,0,1,1])
ax.scatter(s,p, color = "purple")
ax.set_title("How Footage of Home Affects Price")
ax.set_xlabel("Footage of Home")
ax.set_ylabel("Price in Millions")
#plt.tight_layout()
plt.savefig('how_footage_of_home_affects_price.png', bbox_inches='tight');
```



In [ ]: