Final Project Submission

* Student name: Sabina Bains
* Student pace: Self Paced
* Scheduled project review date / time: 1/19/2022 @ 1 pm PCT
* Instructor name: Claude Fried

Business Understanding

Business Problem:

King County Healing Association (KCHA) has reached out to you to help estimate a budget for a program that helps previously incarcerated individuals get back on their feet.

KCHA is an established nonprofit organization that has a variety of successful initiatives such as rehabilitation centers, medical assistance, career services, and temporary housing for those re-entering their communities. The organization now wants to expand on their housing opportunities by rolling out a new service to help those who are prepared to transition into a permanent residence.

Unfortunately, those with criminal records tend to be discriminated against even after serving their sentence, and often banks do not lend money to these individuals. Additionally, those previously incarcerated could have bad credit or a lack of savings. Because of all this, KCHA plans to borrow from banks themselves and provide the down payment for applicants of this program. The new tenants can then pay off their mortgage to KCHA monthly, while additionally paying back the down payment over time.

Along with requirements to ensure the applicants have a stable income and can pay off their debt, KCHA requires tenants to choose homes within the same zipcode of their parole officer (if applicable).

KCHA doesn't plan to launch this service for another 3 years, and therefore are in the preliminary stages. The organization estimates they will assist 5 families in their first year of the program, and would like our help estimating the total down payment cost based on these family's parole office location, household size, and general preferences. Ideally, they would not exceed 500k in their first year.

Data Understanding

Importing Packages and Reading in Data

In [1]:
```
# importing necessary packages
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
%matplotlib inline
import statsmodels.stats.stattools as stattools
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
from sklearn import linear_model
import scipy.stats as stats
import seaborn as sns
import numpy as np
import warnings
from math import e

warnings.filterwarnings('ignore')
style.use('dark_background')
pd.set_option("display.max_columns", 999)
pd.set_option("display.max_rows", 999)
```

In [2]:
```
# importing King County House Sales dataset
df = pd.read_csv('data/kc_house_data.csv')
```

Looking at sample # / number of predictors / basic statistics

In [3]:
```
print('''This data set has {} rows and {} columns.'''.format(df.shape[0],c
```

```
This data set has 21597 rows and 21 columns.
```

In [4]:
```
# looks like price is the dependent variable with 20 potential predictors
df.head()
```

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

In [5]:
```
# taking a look at basic stats of descriptors. price looks skewed based o
df.describe()
```

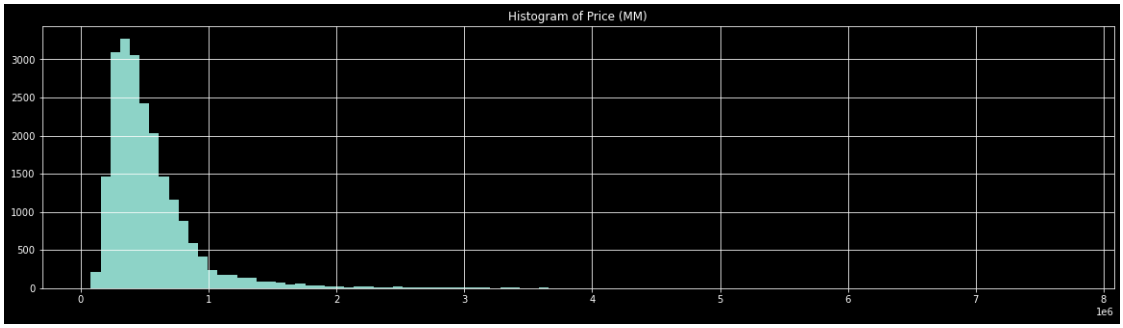
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	215
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.49
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.53
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.00
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.00
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.50
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.00
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.50

Looking at distribution of continuous variables

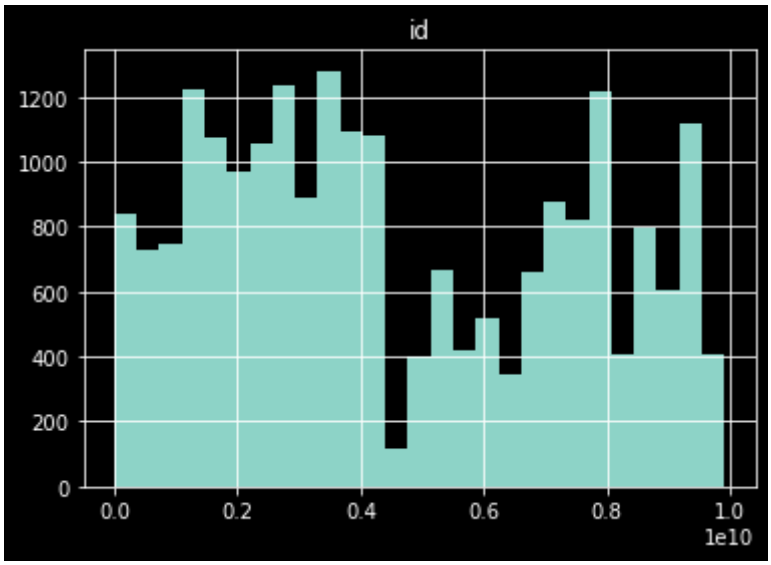
-- not an assumption

In [6]:
```
## distribution of price variable
df.hist('price',bins = 100, figsize = (20,5))
plt.title('Histogram of Price (MM)')
```

```
Text(0.5, 1.0, 'Histogram of Price (MM)')
```



In [7]:
```
# Checking Normality of Variables (not an assumption, but just for referer
# id, floors, waterfront, view, yr_built, yr_renovated, zipcode, lat, lon
for col in df.drop('price',axis=1).columns:
    if df[col].dtype == np.float64 or df[col].dtype == np.int64:
        df.hist(col,bins='auto')
```



Data Preparation

Handling N/A's

In [8]:
```
df.isna().sum()
# looks like waterfront, view, and yr_renovated columns need to be addres
```

```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view             63
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated   3842
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

In [9]:
```
## checking waterfront column
print('''Unique values are {} removing waterfront N/A's would remove {} of
are looking at affordable homes, and a waterfront property is not a priori
      .format(df.waterfront.unique(), '{:0.1%}'.format(df.waterfront.isna(
## dropping waterfront columns
df.drop('waterfront',axis=1,inplace=True)
```

```
Unique values are [nan  0.  1.] removing waterfront N/A's would remove 11.0% of the o
verall dataset. We can drop this column from the dataset since we
are looking at affordable homes, and a waterfront property is not a priority
```

In [10]:
```
## dropping views column
df.drop('view',axis=1,inplace=True)
```

In [11]:
```
## checking yr_renovated column
print('''removing view N/A's would remove {} of the overall dataset. Let's
      .format('{:0.1%}'.format(df.yr_renovated.isna().sum() / len(df))))
## filling N/A values with 0
df.yr_renovated.fillna(0,inplace=True)
```

```
removing view N/A's would remove 17.8% of the overall dataset. Let's assume N/A means
no renovations.
```

Converting data types

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   condition      21597 non-null  int64  
 9   grade          21597 non-null  int64  
 10  sqft_above     21597 non-null  int64  
 11  sqft_basement  21597 non-null  object 
 12  yr_built       21597 non-null  int64  
 13  yr_renovated   21597 non-null  float64
 14  zipcode        21597 non-null  int64  
 15  lat            21597 non-null  float64
 16  long           21597 non-null  float64
 17  sqft_living15  21597 non-null  int64  
 18  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(11), object(2)
memory usage: 3.1+ MB
```

In [13]:
```
# sqft_basement is object type, should be numeric.
df.sqft_basement.unique()[:6]
```

```
array(['0.0', '400.0', '910.0', '1530.0', '?', '730.0'], dtype=object)
```

In [14]:
```
## some rows have a "?" value. need to remove
df = df.loc[~(df.sqft_basement == '?')]
```

In [15]:
```
# changing ID and Zip columns to str type
for col in ['id', 'zipcode']:
    df[col] = df[col].astype(str)
```

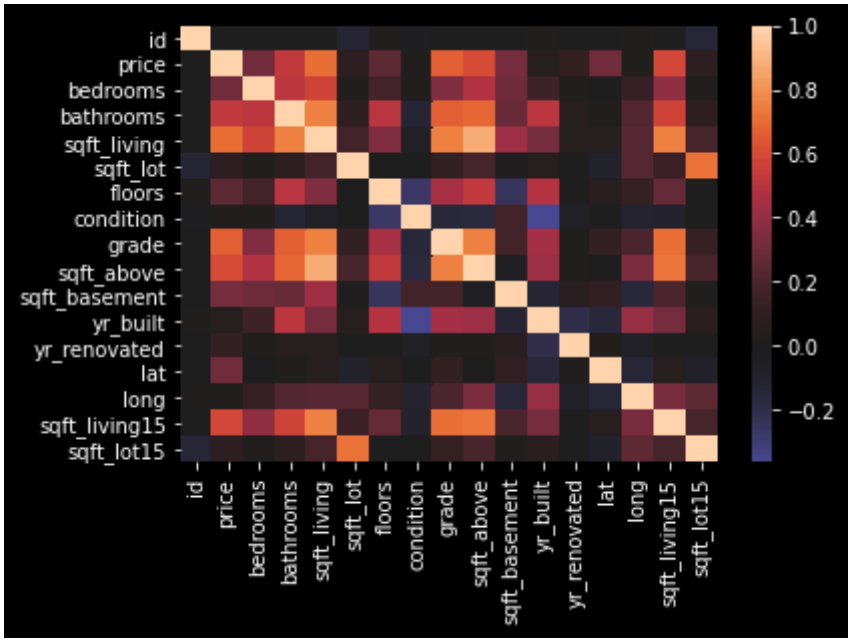
In [16]:
```
# changing columns to int type
for col in ['id', 'price', 'bedrooms', 'sqft_living',
       'sqft_lot', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated',
       'sqft_living15', 'sqft_lot15']:
    df[col] = df[col].astype(float).astype(int)
```

In [17]:
```
df.info()
# all dtypes are as they should be now
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21143 entries, 0 to 21596
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21143 non-null  int64  
 1   date           21143 non-null  object 
 2   price          21143 non-null  int64  
 3   bedrooms       21143 non-null  int64  
 4   bathrooms      21143 non-null  float64
 5   sqft_living    21143 non-null  int64  
 6   sqft_lot       21143 non-null  int64  
 7   floors         21143 non-null  float64
 8   condition      21143 non-null  int64  
 9   grade          21143 non-null  int64  
 10  sqft_above     21143 non-null  int64  
 11  sqft_basement  21143 non-null  int64  
 12  yr_built       21143 non-null  int64  
 13  yr_renovated   21143 non-null  int64  
 14  zipcode        21143 non-null  object 
 15  lat            21143 non-null  float64
 16  long           21143 non-null  float64
 17  sqft_living15  21143 non-null  int64  
 18  sqft_lot15     21143 non-null  int64  
dtypes: float64(4), int64(13), object(2)
memory usage: 3.2+ MB
```

Checking for and removing Multicollinearity

In [18]:
```
# overall view of correlation. Looks like we have some values > 0.7
sns.heatmap(df.corr(), center=0);
```



In [19]:
```
# Viewing Correlation Between Each Variable ad Checking which variables ha
df_corr = df.corr().abs().stack().reset_index()
df_corr['preds'] = df_corr['level_0']+' & '+df_corr['level_1']
df_corr = df_corr.drop(['level_0','level_1'],axis = 1).rename(columns={0:'
df_corr.loc[(df_corr['corr'] > 0.7) & (df_corr['corr'] != 1)]
```

	corr	preds
157	0.876678	sqft_above & sqft_living
77	0.876678	sqft_living & sqft_above
140	0.763101	grade & sqft_living
76	0.763101	sqft_living & grade
259	0.756389	sqft_living15 & sqft_living
83	0.756389	sqft_living & sqft_living15
161	0.756382	sqft_above & grade
145	0.756382	grade & sqft_above
71	0.755278	sqft_living & bathrooms
55	0.755278	bathrooms & sqft_living
264	0.731016	sqft_living15 & sqft_above
168	0.731016	sqft_above & sqft_living15
101	0.720649	sqft_lot & sqft_lot15
277	0.720649	sqft_lot15 & sqft_lot
263	0.713574	sqft_living15 & grade
151	0.713574	grade & sqft_living15
69	0.702328	sqft_living & price
21	0.702328	price & sqft_living

In [20]:
```
# KCHA isn't really interested in renovating details, and since this has l
df.drop(['sqft_lot15','sqft_living15'],axis=1,inplace=True)
```

In [21]:
```
## Looks like sqft_living and sqft_above is also highly correlated with o
df.drop(['sqft_above','sqft_living'],axis=1,inplace=True)
```

Splitting data to Continuous and Categorical variables

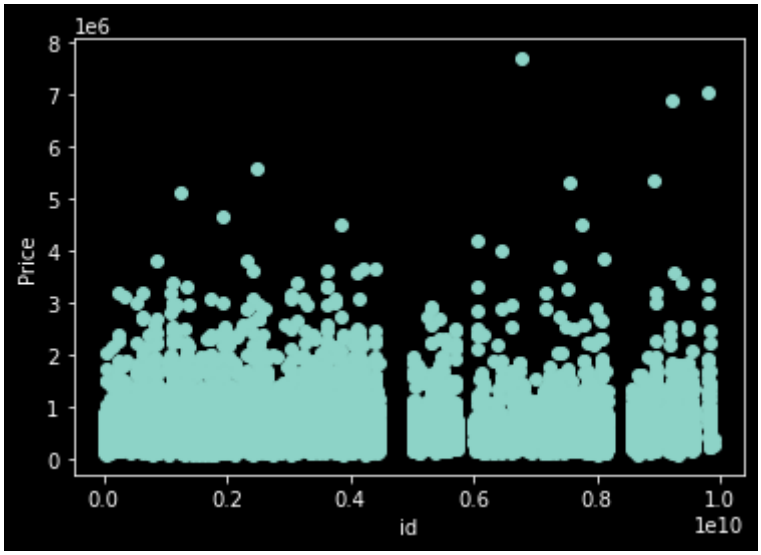
In [22]:
```
#SPLITTING INTO CONTINOUS AND DISCRETE VARIABLES to turn categorical into
cont = df[['price','id', 'bedrooms', 'bathrooms',
       'sqft_lot', 'floors', 'condition', 'grade',
       'sqft_basement', 'yr_built', 'yr_renovated', 'lat', 'long']]
cat = df[['zipcode']]
```

Checking Linearity of Features

In [23]:

```
## Checking Linearity of Features to Dependent variable
for col in cont.drop('price',axis=1).columns:
    plt.scatter(cont[col],cont['price'])
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.show()

print('''
---------- NOTES: ------------
Makes sense that ID does not have a relationship with price -- drop ID
looks like bedrooms has an outlier -- remove bedrooms > 30 from analysis.
sqft_lot does not have a linear relationship with price -- drop sqft_lot
sqft_basement seems off, drop and transform to categorical variable (has b
yr_built is does not have linear relationship with price, drop yr_built
yr_renovated does not have linear relationship with price, drop and transf
we can use grade to determine quality rather than condition since it has a
lat & long do not have a linear relationship, drop
''')
```



In [24]:

```
# removing outlier with 33 bedrooms
cont = cont.loc[~(cont.bedrooms >= 30)]
```

In [25]:
```
## creating function to turn values > 0 to 1, and values = 0 to 0.
def make_binary(row):
    if row > 0:
        return 1
    if row == 0:
        return 0


## modifying sqft_basement and yr_renovated to be binary variables
## adding new dummy vars into x_cat
cat['has_basement'] = cont['sqft_basement'].apply(make_binary)
cat['renovated'] = cont['yr_renovated'].apply(make_binary)
cat = cat.loc[cat.renovated.isin([0,1])]
```

In [26]:
```
## dropping vars that do not have a linear relationship
cont.drop(['id','sqft_lot','sqft_basement','yr_renovated','yr_built','lat'
```

Encoding Categorical Variables

In [27]:
```
# checking out how many unique values are in each cat value
for col in cat.columns:
    print(col, len(cat[col].unique()))
```

```
zipcode 70
has_basement 2
renovated 2
```

In [28]:
```
cat_dummies_list = []
for idx, col in enumerate(cat.columns):
    cat_dummies_list.append(pd.get_dummies(cat[col], prefix=col, drop_firs

cat_dummies = pd.concat(cat_dummies_list, axis=1)
```

Looking into whether we should normalize data

In [29]:
```
cont.drop('price',axis=1).describe()
```

	bedrooms	bathrooms	floors	grade
count	21142.000000	21142.000000	21142.000000	21142.000000
mean	3.371157	2.116096	1.493615	7.658310
std	0.902213	0.768545	0.539252	1.174272
min	1.000000	0.500000	1.000000	3.000000
25%	3.000000	1.750000	1.000000	7.000000
50%	3.000000	2.250000	1.500000	7.000000
75%	4.000000	2.500000	2.000000	8.000000
max	11.000000	8.000000	3.500000	13.000000

In [30]:
```
for col in cont.drop('price',axis=1).columns:
    print('range of values for '+col+": ",max(cont[col]) - min(cont[col]))
print("\nconsidering the range for all these features are not too differer
```

```
range of values for bedrooms:  10
range of values for bathrooms:  7.5
range of values for floors:  2.5
range of values for grade:  10

considering the range for all these features are not too different, we shouldn't need
to normalize the data
```

Modeling

MODEL 1

In [31]:
```
# Creating inputs for model 1, with Continuous variables only
X = cont.drop('price',axis=1)
y = cont['price']
```

In [32]:
```
# creating function to run OLS regression
def run_model(X,y):
    predictors_int = sm.add_constant(X)
    return sm.OLS(y,predictors_int).fit()
```

In [33]:

```
# running model
model_1 = run_model(X,y)
model_1.summary()
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.467
Model:	OLS	Adj. R-squared:	0.467
Method:	Least Squares	F-statistic:	4630.
Date:	Wed, 19 Jan 2022	Prob (F-statistic):	0.00
Time:	11:10:21	Log-Likelihood:	-2.9430e+05
No. Observations:	21142	AIC:	5.886e+05
Df Residuals:	21137	BIC:	5.887e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-9.874e+05	1.37e+04	-72.000	0.000	-1.01e+06	-9.61e+05
bedrooms	1.217e+04	2428.974	5.011	0.000	7410.050	1.69e+04
bathrooms	8.057e+04	3728.096	21.611	0.000	7.33e+04	8.79e+04
floors	-7.271e+04	4071.543	-17.859	0.000	-8.07e+04	-6.47e+04
grade	1.861e+05	2153.468	86.405	0.000	1.82e+05	1.9e+05

Omnibus:	18852.047	Durbin-Watson:	1.972
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1731976.726
Skew:	3.904	Prob(JB):	0.00
Kurtosis:	46.648	Cond. No.	67.3

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

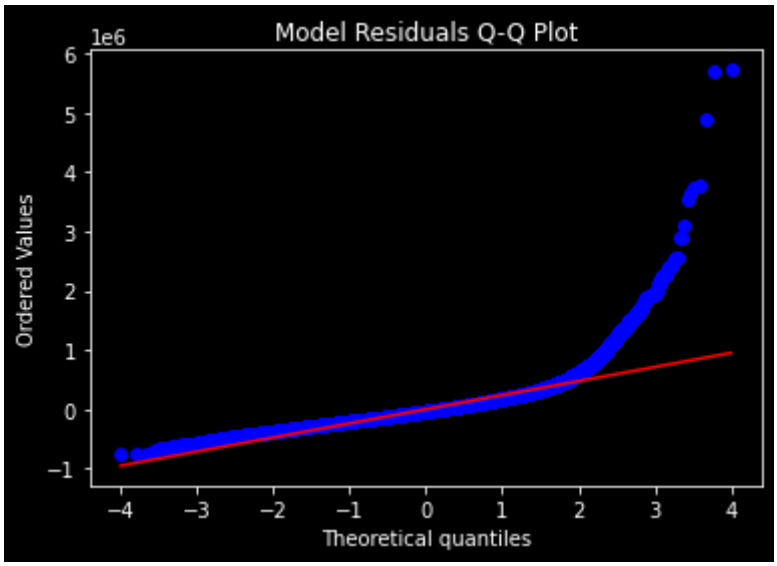
In [34]:

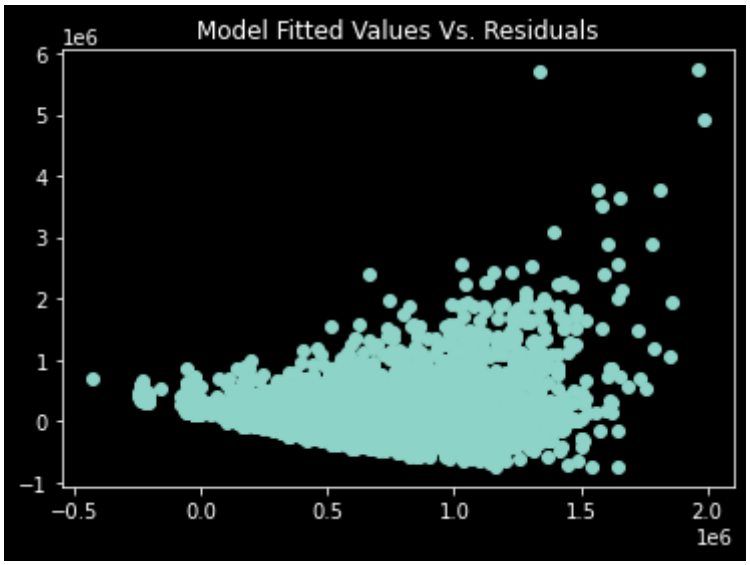
```
# Checking for normality of residuals and homoscedasticity

# running QQ Plot
stats.probplot(model_1.resid, dist="norm", plot= plt)
plt.title("Model Residuals Q-Q Plot")
plt.show()

# running fitted values vs. residuals
plt.scatter(model_1.fittedvalues,model_1.resid)
plt.title("Model Fitted Values Vs. Residuals")
plt.show()

print("residuals for this model do not follow that of a normal distributic
```





residuals for this model do not follow that of a normal distribution, and error terms for this model are heteroscedastic. We may want to log transform variables

Model 2 - Transforming Y & dropping Floors

In [35]:

```
# Log transforming Y variable
y_log = np.log(y)

# Floors having a negative coefficient doesn't seem to make sense. Let's d
X = X.drop('floors', axis = 1)

# running model 2 with log transform
model_2 = run_model(X,y_log)
model_2.summary()
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.511
Model:	OLS	Adj. R-squared:	0.511
Method:	Least Squares	F-statistic:	7355.
Date:	Wed, 19 Jan 2022	Prob (F-statistic):	0.00
Time:	11:10:22	Log-Likelihood:	-8886.3
No. Observations:	21142	AIC:	1.778e+04
Df Residuals:	21138	BIC:	1.781e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.6759	0.019	570.480	0.000	10.639	10.713
bedrooms	0.0412	0.003	12.455	0.000	0.035	0.048
bathrooms	0.0767	0.005	15.843	0.000	0.067	0.086
grade	0.2705	0.003	93.414	0.000	0.265	0.276

Omnibus:	134.197	Durbin-Watson:	1.961
Prob(Omnibus):	0.000	Jarque-Bera (JB):	136.612
Skew:	0.196	Prob(JB):	2.16e-30
Kurtosis:	3.043	Cond. No.	66.0

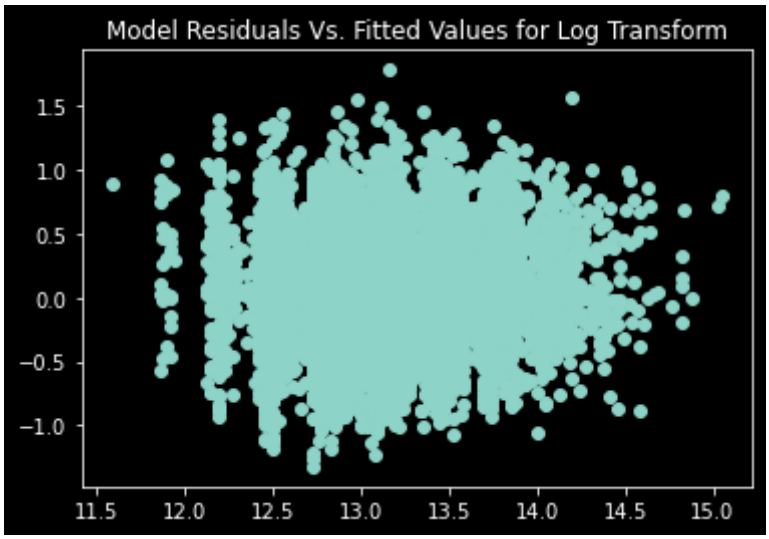
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [36]:

```
# Model 2 has an improved adj. R-squared. Let's check the residuals again.
```

In [37]:
```
stats.probplot(model_2.resid, dist="norm", plot= plt)
plt.title("Model Residuals Q-Q Plot for Log Transform")
plt.show()
plt.scatter(model_2.fittedvalues,model_2.resid)
plt.title("Model Residuals Vs. Fitted Values for Log Transform")
plt.show()
```



In [38]:
```
# These plots look much better, errors are homoscedasctic and follow a no
```

Model 3 - Adding Categorical Variables

In [39]:
```
# Adding in zipcode dummy variables to our model
X = pd.concat((X, cat_dummies), axis=1)
model_3 = run_model(X , y_log)
model_3.summary()
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.802
Model:	OLS	Adj. R-squared:	0.802
Method:	Least Squares	F-statistic:	1156.
Date:	Wed, 19 Jan 2022	Prob (F-statistic):	0.00
Time:	11:10:22	Log-Likelihood:	696.23
No. Observations:	21142	AIC:	-1242.
Df Residuals:	21067	BIC:	-645.5
Df Model:	74		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.6811	0.018	598.803	0.000	10.646	10.716
bedrooms	0.0536	0.002	24.734	0.000	0.049	0.058
bathrooms	0.0887	0.003	27.743	0.000	0.082	0.095
grade	0.1970	0.002	98.302	0.000	0.193	0.201
zipcode_98002	-0.0147	0.021	-0.705	0.481	-0.056	0.026
zipcode_98003	-0.0077	0.019	-0.408	0.683	-0.045	0.029
zipcode_98004	1.1491	0.018	62.664	0.000	1.113	1.185
zipcode_98005	0.7563	0.022	34.175	0.000	0.713	0.800
zipcode_98006	0.6818	0.017	41.139	0.000	0.649	0.714
zipcode_98007	0.6187	0.024	26.300	0.000	0.573	0.665
zipcode_98008	0.6744	0.019	35.822	0.000	0.637	0.711
zipcode_98010	0.3357	0.027	12.529	0.000	0.283	0.388
zipcode_98011	0.4441	0.021	21.098	0.000	0.403	0.485
zipcode_98014	0.4136	0.025	16.777	0.000	0.365	0.462
zipcode_98019	0.3589	0.021	16.933	0.000	0.317	0.400
zipcode_98022	0.1555	0.020	7.797	0.000	0.116	0.195
zipcode_98023	-0.0572	0.016	-3.486	0.000	-0.089	-0.025
zipcode_98024	0.5497	0.029	18.824	0.000	0.492	0.607
zipcode_98027	0.5194	0.017	30.156	0.000	0.486	0.553
zipcode_98028	0.4177	0.019	22.149	0.000	0.381	0.455
zipcode_98029	0.5238	0.018	28.513	0.000	0.488	0.560

zipcode_98030	0.0306	0.019	1.580	0.114	-0.007	0.069
zipcode_98031	0.0553	0.019	2.901	0.004	0.018	0.093
zipcode_98032	-0.0461	0.025	-1.876	0.061	-0.094	0.002
zipcode_98033	0.7885	0.017	46.545	0.000	0.755	0.822
zipcode_98034	0.5180	0.016	32.187	0.000	0.486	0.550
zipcode_98038	0.1796	0.016	11.295	0.000	0.148	0.211
zipcode_98039	1.3561	0.036	37.709	0.000	1.286	1.427
zipcode_98040	0.9566	0.019	50.217	0.000	0.919	0.994
zipcode_98042	0.0711	0.016	4.434	0.000	0.040	0.103
zipcode_98045	0.3552	0.020	17.492	0.000	0.315	0.395
zipcode_98052	0.6166	0.016	38.552	0.000	0.585	0.648
zipcode_98053	0.6487	0.017	37.570	0.000	0.615	0.682
zipcode_98055	0.1266	0.019	6.610	0.000	0.089	0.164
zipcode_98056	0.3452	0.017	20.165	0.000	0.312	0.379
zipcode_98058	0.1550	0.017	9.252	0.000	0.122	0.188
zipcode_98059	0.3693	0.017	22.188	0.000	0.337	0.402
zipcode_98065	0.4716	0.018	25.575	0.000	0.435	0.508
zipcode_98070	0.5702	0.025	22.682	0.000	0.521	0.619
zipcode_98072	0.5155	0.019	27.032	0.000	0.478	0.553
zipcode_98074	0.5543	0.017	32.666	0.000	0.521	0.588
zipcode_98075	0.6193	0.018	34.564	0.000	0.584	0.654
zipcode_98077	0.5254	0.021	24.985	0.000	0.484	0.567
zipcode_98092	0.0386	0.018	2.167	0.030	0.004	0.074
zipcode_98102	0.8238	0.027	30.875	0.000	0.772	0.876
zipcode_98103	0.7212	0.016	45.529	0.000	0.690	0.752
zipcode_98105	0.9063	0.020	45.161	0.000	0.867	0.946
zipcode_98106	0.2333	0.018	12.940	0.000	0.198	0.269
zipcode_98107	0.7074	0.019	36.890	0.000	0.670	0.745
zipcode_98108	0.2940	0.021	13.732	0.000	0.252	0.336
zipcode_98109	0.9185	0.026	35.117	0.000	0.867	0.970
zipcode_98112	0.9901	0.019	51.275	0.000	0.952	1.028
zipcode_98115	0.7680	0.016	48.035	0.000	0.737	0.799
zipcode_98116	0.7102	0.018	38.994	0.000	0.675	0.746
zipcode_98117	0.7414	0.016	45.967	0.000	0.710	0.773
zipcode_98118	0.4275	0.016	26.114	0.000	0.395	0.460
zipcode_98119	0.8883	0.022	40.830	0.000	0.846	0.931
zipcode_98122	0.6825	0.019	36.307	0.000	0.646	0.719

zipcode_98125	0.5358	0.017	31.278	0.000	0.502	0.569
zipcode_98126	0.4924	0.018	27.568	0.000	0.457	0.527
zipcode_98133	0.4026	0.016	24.475	0.000	0.370	0.435
zipcode_98136	0.6389	0.019	32.934	0.000	0.601	0.677
zipcode_98144	0.6120	0.018	34.088	0.000	0.577	0.647
zipcode_98146	0.2901	0.019	15.390	0.000	0.253	0.327
zipcode_98148	0.1207	0.034	3.548	0.000	0.054	0.187
zipcode_98155	0.4194	0.017	24.939	0.000	0.386	0.452
zipcode_98166	0.3883	0.019	19.963	0.000	0.350	0.426
zipcode_98168	0.0724	0.019	3.775	0.000	0.035	0.110
zipcode_98177	0.6355	0.019	32.704	0.000	0.597	0.674
zipcode_98178	0.1710	0.019	8.883	0.000	0.133	0.209
zipcode_98188	0.0832	0.024	3.466	0.001	0.036	0.130
zipcode_98198	0.1003	0.019	5.296	0.000	0.063	0.137
zipcode_98199	0.8107	0.018	44.117	0.000	0.775	0.847
has_basement_1.0	0.0560	0.004	15.352	0.000	0.049	0.063
renovated_1.0	0.1248	0.009	13.881	0.000	0.107	0.142

Omnibus:	1566.228	Durbin-Watson:	1.982
Prob(Omnibus):	0.000	Jarque-Bera (JB):	5951.583
Skew:	0.301	Prob(JB):	0.00
Kurtosis:	5.529	Cond. No.	578.

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [40]:
```
# adjusted R Squared value increases by a lot. some Zip's have p-Values >
```

In [41]:
```
zips_to_drop = ['zipcode_98002','zipcode_98003','zipcode_98030','zipcode_9
```

Model 4 - Removing Features with P-values > 0.05

In [42]:

```
## Dropping zips with high P-value's
X = X.drop(zips_to_drop,axis=1)
model_4 = run_model(X, y_log)
model_4.summary()
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.802
Model:	OLS	Adj. R-squared:	0.802
Method:	Least Squares	F-statistic:	1221.
Date:	Wed, 19 Jan 2022	Prob (F-statistic):	0.00
Time:	11:10:22	Log-Likelihood:	691.24
No. Observations:	21142	AIC:	-1240.
Df Residuals:	21071	BIC:	-675.4
Df Model:	70		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	10.6782	0.014	742.349	0.000	10.650	10.706
bedrooms	0.0536	0.002	24.714	0.000	0.049	0.058
bathrooms	0.0891	0.003	27.888	0.000	0.083	0.095
grade	0.1969	0.002	98.507	0.000	0.193	0.201
zipcode_98004	1.1515	0.015	76.514	0.000	1.122	1.181
zipcode_98005	0.7587	0.019	38.928	0.000	0.721	0.797
zipcode_98006	0.6843	0.013	53.291	0.000	0.659	0.709
zipcode_98007	0.6212	0.021	29.492	0.000	0.580	0.662
zipcode_98008	0.6770	0.016	43.297	0.000	0.646	0.708
zipcode_98010	0.3382	0.025	13.713	0.000	0.290	0.387
zipcode_98011	0.4466	0.018	24.460	0.000	0.411	0.482
zipcode_98014	0.4160	0.022	18.641	0.000	0.372	0.460
zipcode_98019	0.3613	0.018	19.606	0.000	0.325	0.397
zipcode_98022	0.1580	0.017	9.312	0.000	0.125	0.191
zipcode_98023	-0.0547	0.013	-4.330	0.000	-0.079	-0.030
zipcode_98024	0.5521	0.027	20.254	0.000	0.499	0.606
zipcode_98027	0.5218	0.014	38.177	0.000	0.495	0.549
zipcode_98028	0.4202	0.016	26.803	0.000	0.390	0.451
zipcode_98029	0.5261	0.015	34.838	0.000	0.497	0.556
zipcode_98031	0.0578	0.016	3.629	0.000	0.027	0.089
zipcode_98033	0.7909	0.013	59.417	0.000	0.765	0.817

zipcode_98034	0.5206	0.012	42.631	0.000	0.497	0.545
zipcode_98038	0.1820	0.012	15.200	0.000	0.158	0.205
zipcode_98039	1.3584	0.034	39.474	0.000	1.291	1.426
zipcode_98040	0.9590	0.016	60.276	0.000	0.928	0.990
zipcode_98042	0.0736	0.012	6.055	0.000	0.050	0.097
zipcode_98045	0.3576	0.017	20.557	0.000	0.324	0.392
zipcode_98052	0.6190	0.012	51.224	0.000	0.595	0.643
zipcode_98053	0.6510	0.014	47.396	0.000	0.624	0.678
zipcode_98055	0.1291	0.016	8.057	0.000	0.098	0.161
zipcode_98056	0.3477	0.014	25.687	0.000	0.321	0.374
zipcode_98058	0.1575	0.013	12.053	0.000	0.132	0.183
zipcode_98059	0.3717	0.013	28.727	0.000	0.346	0.397
zipcode_98065	0.4739	0.015	31.211	0.000	0.444	0.504
zipcode_98070	0.5727	0.023	25.064	0.000	0.528	0.617
zipcode_98072	0.5180	0.016	32.508	0.000	0.487	0.549
zipcode_98074	0.5566	0.013	41.685	0.000	0.530	0.583
zipcode_98075	0.6215	0.015	42.729	0.000	0.593	0.650
zipcode_98077	0.5278	0.018	28.934	0.000	0.492	0.564
zipcode_98092	0.0410	0.014	2.846	0.004	0.013	0.069
zipcode_98102	0.8263	0.025	33.678	0.000	0.778	0.874
zipcode_98103	0.7237	0.012	60.965	0.000	0.700	0.747
zipcode_98105	0.9089	0.017	53.129	0.000	0.875	0.942
zipcode_98106	0.2359	0.015	16.097	0.000	0.207	0.265
zipcode_98107	0.7100	0.016	44.220	0.000	0.678	0.741
zipcode_98108	0.2966	0.019	15.895	0.000	0.260	0.333
zipcode_98109	0.9211	0.024	38.440	0.000	0.874	0.968
zipcode_98112	0.9926	0.016	61.221	0.000	0.961	1.024
zipcode_98115	0.7707	0.012	63.908	0.000	0.747	0.794
zipcode_98116	0.7128	0.015	47.888	0.000	0.684	0.742
zipcode_98117	0.7440	0.012	60.758	0.000	0.720	0.768
zipcode_98118	0.4301	0.013	34.244	0.000	0.406	0.455
zipcode_98119	0.8908	0.019	46.739	0.000	0.853	0.928
zipcode_98122	0.6851	0.016	43.916	0.000	0.654	0.716
zipcode_98125	0.5385	0.014	39.761	0.000	0.512	0.565
zipcode_98126	0.4951	0.014	34.256	0.000	0.467	0.523
zipcode_98133	0.4052	0.013	31.987	0.000	0.380	0.430
zipcode_98136	0.6416	0.016	39.322	0.000	0.610	0.674

zipcode_98144	0.6146	0.015	42.190	0.000	0.586	0.643
zipcode_98146	0.2928	0.016	18.696	0.000	0.262	0.323
zipcode_98148	0.1232	0.032	3.809	0.000	0.060	0.187
zipcode_98155	0.4220	0.013	32.108	0.000	0.396	0.448
zipcode_98166	0.3909	0.016	23.868	0.000	0.359	0.423
zipcode_98168	0.0752	0.016	4.679	0.000	0.044	0.107
zipcode_98177	0.6380	0.016	39.009	0.000	0.606	0.670
zipcode_98178	0.1737	0.016	10.762	0.000	0.142	0.205
zipcode_98188	0.0858	0.022	3.973	0.000	0.043	0.128
zipcode_98198	0.1029	0.016	6.526	0.000	0.072	0.134
zipcode_98199	0.8133	0.015	53.923	0.000	0.784	0.843
has_basement_1.0	0.0557	0.004	15.288	0.000	0.049	0.063
renovated_1.0	0.1246	0.009	13.854	0.000	0.107	0.142

Omnibus:	1562.473	Durbin-Watson:	1.982
Prob(Omnibus):	0.000	Jarque-Bera (JB):	5934.891
Skew:	0.300	Prob(JB):	0.00
Kurtosis:	5.525	Cond. No.	315.

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [43]:
```
# Although the adj. R value does not increase after removal, the F-statis
```

MODEL EVALUATION

Train - Test Split to assess model 4's ability to predict

In [44]:
```
# Performing a train-test-split to assess performance, and whether the mo
# randomly split the data, 67% train 33% test.
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y_log)
```

In [45]:
```
# apply regression to training X and y data, use to predict X train and X
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [46]:
```
#looking at Mean Squared Error of train and test
# from sklearn.metrics import root_mean_squared_error

mse_train = np.sum((y_train-y_hat_train)**2)/len(y_train)
mse_test = np.sum((y_test-y_hat_test)**2)/len(y_test)
print('Train Mean Squarred Error:', '{:,.3f}'.format(mse_train))
print('Test Mean Squarred Error:', '{:,.3f}'.format(mse_test))
print("\nDifference between train and test:",'{:.1%}'.format(mse_test / ms
```

```
Train Mean Squarred Error: 0.055
Test Mean Squarred Error: 0.055

Difference between train and test: 0.3%
```

In [47]:
```
## difference can change depending on the randomly chosen sample, so lets
```

Cross Validation

In [48]:
```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
import sklearn

from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.model_selection import cross_val_score

mse = make_scorer(mean_squared_error)

avg_mse = np.mean(cross_val_score(linreg, X, y_log, cv=20, scoring=mse))

print('With an average MSE of','{:.3f}'.format(avg_mse), 'when performing
```

```
With an average MSE of 0.056 when performing cross validation with 20 folds on our te
st data, we can be more confident in our model's ability to predict new data
```

Final Evaluation

In [49]:
```
# creating dictionary of coefficient names and corresponding values for ai
coefs= dict(model_4.params)
```

In [50]:
```
# creating function to report on changes for each feature in our model
def coef_summary():
    for key, value in coefs.items():
        if key == 'const':
            print('A home in zipcode 98001, with a value of 0 for all othe
        else:
            if 'zipcode' in key:
                if value > 0:
                    print('Purchasing a home in',key,'would increase the h
                if value < 0:
                    print('Purchasing a home in',key,'would decrease the h
            else:
                if value > 0:
                    print('A one unit increase in',key,'would increase the
                if value < 0:
                    print('A one unit increase',key,'would decrease the ho
```

In [51]:
```
(e**.1032614) - 1
```

```
0.10878120660522672
```

In [52]:
```
print('''Final Evaluation:\n\nModel 4 is our strongest model for multiple
be explained by our model's features. This result has the highest F-Statis
intercept-only model would be a better predictor. All features also have a
no effect on house price. Lastly, since this model is built for preliminar
many specific features that require input, such as exact square footage of
as well as changes in sale price value for each additional unit of a featu
change, as the sale variable is log transformed.\n''')
coef_summary()
```

```
Final Evaluation:

Model 4 is our strongest model for multiple reasons. It has an adj. R squared value o
f 0.802, suggesting ~80% of variance in the data can
be explained by our model's features. This result has the highest F-Statistic, and an
associated lowest P-value of < 0.001, meaning we reject the null hypothesis that an
intercept-only model would be a better predictor. All features also have an associate
d P-value of < 0.05, meaning there is less than a 5% chance that these variables have
no effect on house price. Lastly, since this model is built for preliminary insights
only and is mainly focused on changes in price due to location, we do not have too
many specific features that require input, such as exact square footage of the home /
lot / basement. Below states the value of a home at it's base (with no features),
as well as changes in sale price value for each additional unit of a feature, assumin
g all other variables are held constant. The changes are reported as a percentage
change, as the sale variable is log transformed.

A home in zipcode 98001, with a value of 0 for all other features, would have a sale
price of $43,398
A one unit increase in bedrooms would increase the house price by 5.5%
A one unit increase in bathrooms would increase the house price by 9.3%
A one unit increase in grade would increase the house price by 21.8%
Purchasing a home in zipcode_98004 would increase the house price by 216.3%
Purchasing a home in zipcode_98005 would increase the house price by 113.6%
Purchasing a home in zipcode_98006 would increase the house price by 98.2%
Purchasing a home in zipcode_98007 would increase the house price by 86.1%
Purchasing a home in zipcode_98008 would increase the house price by 96.8%
Purchasing a home in zipcode_98010 would increase the house price by 40.2%
Purchasing a home in zipcode_98011 would increase the house price by 56.3%
Purchasing a home in zipcode_98014 would increase the house price by 51.6%
Purchasing a home in zipcode_98019 would increase the house price by 43.5%
Purchasing a home in zipcode_98022 would increase the house price by 17.1%
Purchasing a home in zipcode_98023 would decrease the house price by 5.3%
Purchasing a home in zipcode_98024 would increase the house price by 73.7%
Purchasing a home in zipcode_98027 would increase the house price by 68.5%
Purchasing a home in zipcode_98028 would increase the house price by 52.2%
Purchasing a home in zipcode_98029 would increase the house price by 69.2%
Purchasing a home in zipcode_98031 would increase the house price by 6.0%
Purchasing a home in zipcode_98033 would increase the house price by 120.5%
Purchasing a home in zipcode_98034 would increase the house price by 68.3%
Purchasing a home in zipcode_98038 would increase the house price by 20.0%
Purchasing a home in zipcode_98039 would increase the house price by 289.0%
Purchasing a home in zipcode_98040 would increase the house price by 160.9%
Purchasing a home in zipcode_98042 would increase the house price by 7.6%
Purchasing a home in zipcode_98045 would increase the house price by 43.0%
```

```
Purchasing a home in zipcode_98052 would increase the house price by 85.7%
Purchasing a home in zipcode_98053 would increase the house price by 91.7%
Purchasing a home in zipcode_98055 would increase the house price by 13.8%
Purchasing a home in zipcode_98056 would increase the house price by 41.6%
Purchasing a home in zipcode_98058 would increase the house price by 17.1%
Purchasing a home in zipcode_98059 would increase the house price by 45.0%
Purchasing a home in zipcode_98065 would increase the house price by 60.6%
Purchasing a home in zipcode_98070 would increase the house price by 77.3%
Purchasing a home in zipcode_98072 would increase the house price by 67.9%
Purchasing a home in zipcode_98074 would increase the house price by 74.5%
Purchasing a home in zipcode_98075 would increase the house price by 86.2%
Purchasing a home in zipcode_98077 would increase the house price by 69.5%
Purchasing a home in zipcode_98092 would increase the house price by 4.2%
Purchasing a home in zipcode_98102 would increase the house price by 128.5%
Purchasing a home in zipcode_98103 would increase the house price by 106.2%
Purchasing a home in zipcode_98105 would increase the house price by 148.2%
Purchasing a home in zipcode_98106 would increase the house price by 26.6%
Purchasing a home in zipcode_98107 would increase the house price by 103.4%
Purchasing a home in zipcode_98108 would increase the house price by 34.5%
Purchasing a home in zipcode_98109 would increase the house price by 151.2%
Purchasing a home in zipcode_98112 would increase the house price by 169.8%
Purchasing a home in zipcode_98115 would increase the house price by 116.1%
Purchasing a home in zipcode_98116 would increase the house price by 104.0%
Purchasing a home in zipcode_98117 would increase the house price by 110.4%
Purchasing a home in zipcode_98118 would increase the house price by 53.7%
Purchasing a home in zipcode_98119 would increase the house price by 143.7%
Purchasing a home in zipcode_98122 would increase the house price by 98.4%
Purchasing a home in zipcode_98125 would increase the house price by 71.3%
Purchasing a home in zipcode_98126 would increase the house price by 64.1%
Purchasing a home in zipcode_98133 would increase the house price by 50.0%
Purchasing a home in zipcode_98136 would increase the house price by 90.0%
Purchasing a home in zipcode_98144 would increase the house price by 84.9%
Purchasing a home in zipcode_98146 would increase the house price by 34.0%
Purchasing a home in zipcode_98148 would increase the house price by 13.1%
Purchasing a home in zipcode_98155 would increase the house price by 52.5%
Purchasing a home in zipcode_98166 would increase the house price by 47.8%
Purchasing a home in zipcode_98168 would increase the house price by 7.8%
Purchasing a home in zipcode_98177 would increase the house price by 89.3%
Purchasing a home in zipcode_98178 would increase the house price by 19.0%
Purchasing a home in zipcode_98188 would increase the house price by 9.0%
Purchasing a home in zipcode_98198 would increase the house price by 10.8%
Purchasing a home in zipcode_98199 would increase the house price by 125.5%
A one unit increase in has_basement_1.0 would increase the house price by 5.7%
A one unit increase in renovated_1.0 would increase the house price by 13.3%
```

BEGINNING DEPLOYMENT

The details of the five families being considered for this program are listed below:

Family 1 would like a 3 bedroom, 2 bathroom house in zipcode 98011. They would prefer a basement and a home quality of at least 6/13

```
Family 3 would like a 2 bedroom, 1 bathroom house in zipcode 98032.
They would prefer a home quality of 10/13
Family 3 would like a 1 bedroom, 1 bathroom house in zipcode 98045.
They would prefer a home quality of at least 8/13
Family 4 would like a 4 bedroom, 3 bathroom house in zipcode 98112.
They would prefer a basement and a home quality of at least 6/13
Family 5 would like a 1 bedroom, 2 bathroom house in zipcode 98001.
They would prefer a home quality of at least 5/13
```

In [53]:

```
# Writing a function that generates the input to predict using the model
X = X.reindex(sorted(X.columns, reverse = True), axis=1)
def hhold_details(zipcode,renovated,has_basement,grade,bedrooms,bathrooms)
    pred_list = [0] * len([x for x in X.columns if x[:3] == 'zip'])

    if zipcode == 98001 or 'zipcode_'+str(zipcode) in zips_to_drop:
        pass
    else:
        zipcode = 'zipcode_'+str(zipcode)
        zip_idx = X.columns.get_loc(zipcode)
        pred_list[zip_idx] = 1

    pred_list.append(renovated)
    pred_list.append(has_basement)
    pred_list.append(grade)
    pred_list.append(bedrooms)
    pred_list.append(bathrooms)
    return pred_list
```

In [54]:

```
# Fitting Model to final X and y.
regr = linear_model.LinearRegression()
regr.fit(X, y_log)
```

```
LinearRegression()
```

In [55]:

```
# Using function to generate predictor list for each family
fam1 = hhold_details(bedrooms = 3, bathrooms = 2, grade = 6, zipcode = 980
fam2 = hhold_details(bedrooms = 2, bathrooms = 1, grade = 10, zipcode = 98
fam3 = hhold_details(bedrooms = 1, bathrooms = 1, grade = 8, zipcode = 980
fam4 = hhold_details(bedrooms = 4, bathrooms = 3, grade = 6, zipcode = 981
fam5 = hhold_details(bedrooms = 1, bathrooms = 2, grade = 5, zipcode = 980
```

In [56]:

```
# Calculating price prediction based on house featurs, as well as the ovei
overall_spend = []
for fam in [fam1,fam2,fam3,fam4,fam5]:
    pred_fam = regr.predict([fam])
    overall_spend.append(e**pred_fam[0]*.30)

    print('The predicted cost of a home with these features and in this zi
print('\nTotal Budget Required: At least '+'$ {:,.0f}'.format(sum(overall_
```

```
The predicted cost of a home with these features and in this zipcode would be $ 328,1
12, requiring a down payment of at least $ 98,433
The predicted cost of a home with these features and in this zipcode would be $ 378,4
68, requiring a down payment of at least $ 113,541
The predicted cost of a home with these features and in this zipcode would be $ 345,9
46, requiring a down payment of at least $ 103,784
The predicted cost of a home with these features and in this zipcode would be $ 653,2
98, requiring a down payment of at least $ 195,989
The predicted cost of a home with these features and in this zipcode would be $ 146,4
78, requiring a down payment of at least $ 43,944

Total Budget Required: At least $ 555,691
```