Final Project Submission

Please fill out:

- Student name: Cassidy Exum
- Student pace: self paced
- Scheduled project review date/time: Undetermined
- Instructor name: Morgan Jones
- Blog post URL: https://exumexaminesdata.blogspot.com/2022/08/creating-fantasy-sports-app-for-pll.html

Determining the impact of different features on house price

We are going to use linear regression and other statistical tools to determine what features affect the sale price of a house, and create models that can predict the sale price after learning from those freatures.

Stakeholder: Blackrock

Blackrock is an American investment managment company that has recently been buying up a ton of real estate. They have tasked us with helping them determine the key features of a houses sale price and generating models to predict prices for them.

What questions are we going to solve?

Create the best model for predicting the price of a home

Start with the most correlated feature. Then expand until our model is most accurate

We want Blackrock to be able to predict house prices and make smart offers when purchasing real estate

1: Imports and Data Cleaning

Lets start by importing the required packages and inspecting the data

In [1]:

```
# Standard imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
data = pd.read_csv('data/kc_house_data.csv')
```

In [3]:

```
# Next few blocks will be some data info
data.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	N
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	

5 rows × 21 columns

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 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   waterfront     19221 non-null  object
 9   view           21534 non-null  object
 10  condition      21597 non-null  object
 11  grade          21597 non-null  object
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  object
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

In [5]:

```
#price info
data['price'].describe()
```

Out[5]:

```
count    2.159700e+04
mean     5.402966e+05
```

```
std      3.673681e+05
min      7.800000e+04
25%      3.220000e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

In [6]:

```
# 3 columns arent the same length as everything else
data.isna().sum()
```

Out[6]:

```
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
```

Are these columns useful? What are they?

- `waterfront` - Whether the house is on a waterfront
  - Includes Duwamish, Elliott Bay, Puget Sound, Lake Union, Ship Canal, Lake Washington, Lake Sammamish, other lake, and river/slough waterfronts
- `view` - Quality of view from house
  - Includes views of Mt. Rainier, Olympics, Cascades, Territorial, Seattle Skyline, Puget Sound, Lake Washington, Lake Sammamish, small lake / river / creek, and other
- `yr_renovated` - Year when house was renovated

In [7]:

```
# Check counts
data['waterfront'].value_counts()
```

Out[7]:

```
NO     19075
YES      146
Name: waterfront, dtype: int64
```

Waterfront is binary, so NA values we can assume are "NO"

In [8]:

```
# fill NAN values
data['waterfront'].fillna('NO', inplace=True)
```

In [9]:

```
#map waterfront view
water_map = {'NO': 0, 'YES': 1}
data['waterfront'] = data['waterfront'].map(water_map)
```

In [10]:

```
# Check counts
data['view'].value_counts()
```

Out[10]:

```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

We can also assume that NAN values in View would just be None

In [11]:

```
# fill NAN values
data['view'].fillna('NONE', inplace=True)
```

In [12]:

```
#map view column
view_map = {'NONE': 0, 'FAIR': 1, 'AVERAGE': 2, 'GOOD': 3, 'EXCELLENT':4}
data['view'] = data['view'].map(view_map)
```

Last column that has issues, 'year renovated'

In [13]:

```
# Check counts
data['yr_renovated'].value_counts()
```

Out[13]:

```
0.0       17011
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

Ok, good to note that 0.0 is a placeholder. It probably indicates that the house was never renovated and NAN values represent that we do not know if it was ever renovated. I don't want to fill NA with anything here, so we probably won't use this column. Lets just drop it.

In [14]:

```
# drop yr_renovated
data.drop('yr_renovated', axis=1, inplace=True)
```

Lets check the data again and ensure everything worked

In [15]:

```
data.isna().sum()
```

Out[15]:

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
```

```
sqft_basement    0
yr_built         0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Perfect. The last column that i was to check right at the beginning is the price column, as that is our target and we can't have any weird placeholder values

In [16]:

```
#check counts
data['price'].value_counts()
```

Out[16]:

```
350000.0    172
450000.0    172
550000.0    159
500000.0    152
425000.0    150
           ... 
870515.0      1
336950.0      1
386100.0      1
176250.0      1
884744.0      1
Name: price, Length: 3622, dtype: int64
```

In [17]:

```
#erase string characters in grade column
def joiner(x):
    return ''.join(filter(str.isdigit, x))

grade = data['grade'].apply(joiner)
grade = pd.to_numeric(grade)
data['grade'] = grade
```

In [18]:

```
#map condition column
cond_map = {'Poor': 0, 'Fair': 1, 'Average': 2, 'Good': 3, 'Very Good':4}
data['condition'] = data['condition'].map(cond_map)
```

In [19]:

```
#replace ? with 0
data['sqft_basement'].replace(to_replace='?', value=0, inplace=True)
```

In [20]:

```
#convert type
data['sqft_basement'] = data['sqft_basement'].astype('float64')
```

In [21]:

```
#check counts
data['sqft_basement'].value_counts()
```

Out[21]:

```
0.0       13280
600.0       217
500.0       209
700.0       208
800.0       201
          ... 
915.0         1
295.0         1
1281.0        1
2130.0        1
906.0         1
Name: sqft_basement, Length: 303, dtype: int64
```

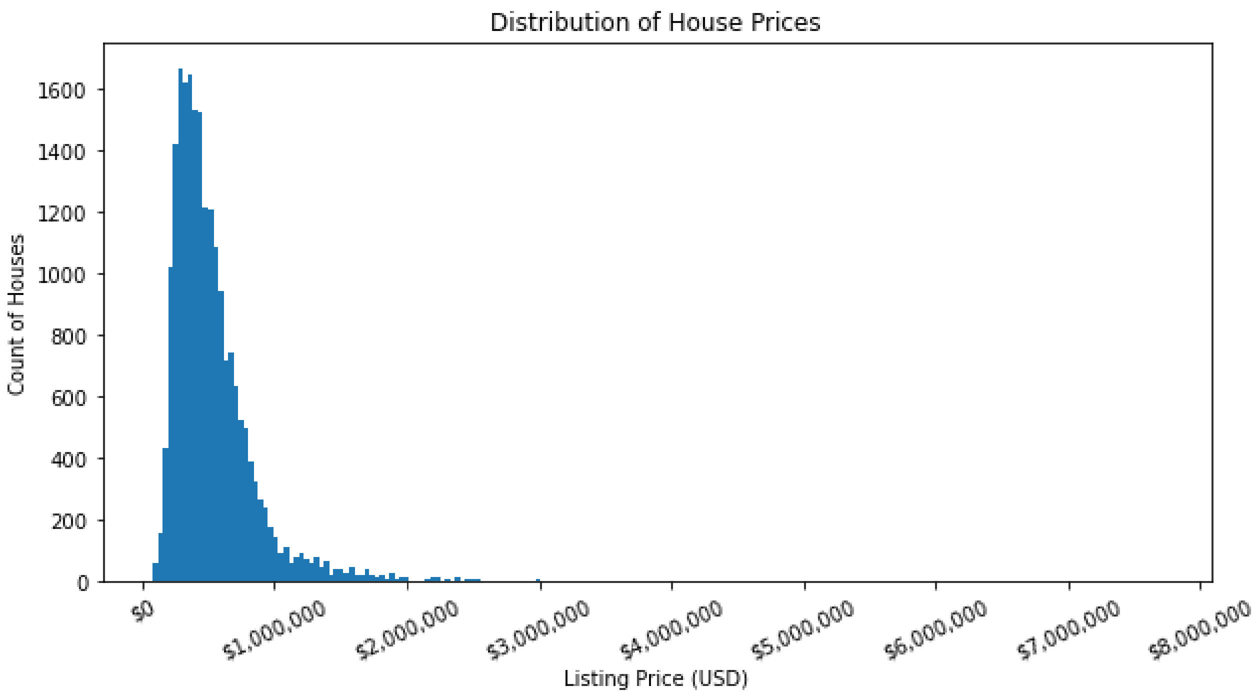
In [22]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 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   waterfront     21597 non-null  int64  
 9   view           21597 non-null  int64  
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  float64
 14  yr_built       21597 non-null  int64  
 15  zipcode        21597 non-null  int64  
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(13), object(1)
memory usage: 3.3+ MB
```

In [23]:

```
# Create a graph of sale price

import matplotlib.ticker as mtick

fig, ax = plt.subplots(figsize=(10, 5))

ax.hist(data['price'], bins=200)

ax.set_xlabel("Listing Price (USD)")
ax.set_ylabel("Count of Houses")
ax.set_title("Distribution of House Prices");

fmt = '${x:,.0f}'
plt.ticklabel_format(style='plain')
tick = mtick.StrMethodFormatter(fmt)
ax.xaxis.set_major_formatter(tick)
plt.xticks(rotation=25)

plt.show();
```



Looks good. Lets start building models.

2: Starting the modeling

Lets set up a train test split for our data

In [24]:

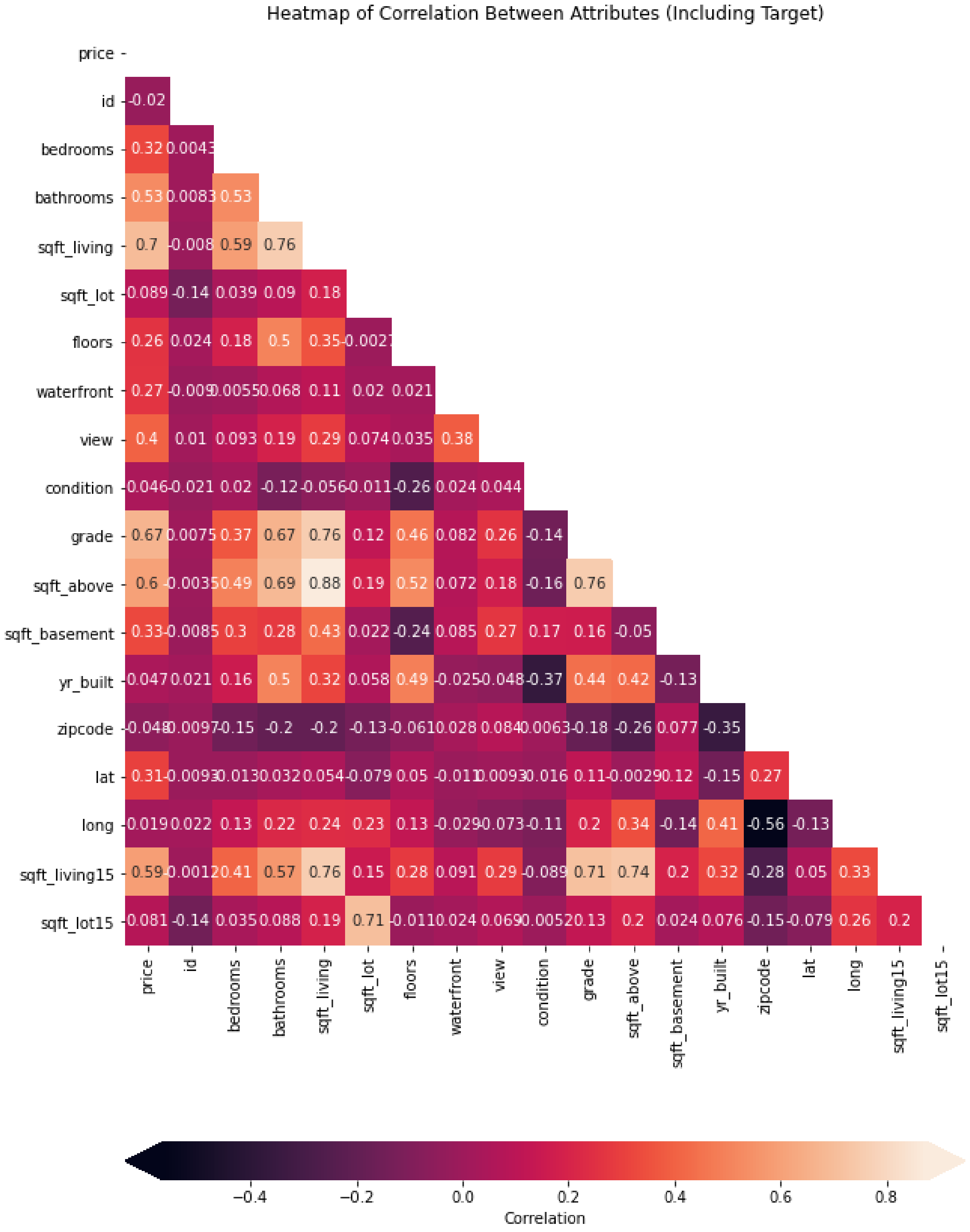
```
from sklearn.model_selection import train_test_split
```

In [25]:

```
X = data.drop(['price'], axis=1)
y = data['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random
```

In [27]:

```
#Create a correlation heatmap

# Create a df with the target as the first column,
# then compute the correlation matrix
heatmap_data = pd.concat([y_train, X_train], axis=1)
corr = heatmap_data.corr()

# Set up figure and axes
fig, ax = plt.subplots(figsize=(10, 16))

# Plot a heatmap of the correlation matrix, with both
# numbers and colors indicating the correlations
sns.heatmap(
    # Specifies the data to be plotted
    data=corr,
    # The mask means we only show half the values,
    # instead of showing duplicates. It's optional.
    mask=np.triu(np.ones_like(corr, dtype=bool)),
    # Specifies that we should use the existing axes
    ax=ax,
    # Specifies that we want labels, not just colors
    annot=True,
    # Customizes colorbar appearance
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .15, "
)

# Customize the plot appearance
ax.set_title("Heatmap of Correlation Between Attributes (Including Target)");
```

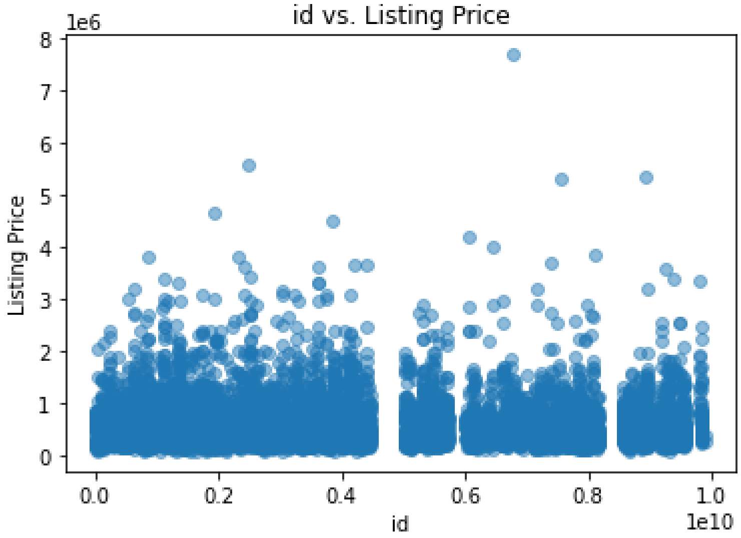


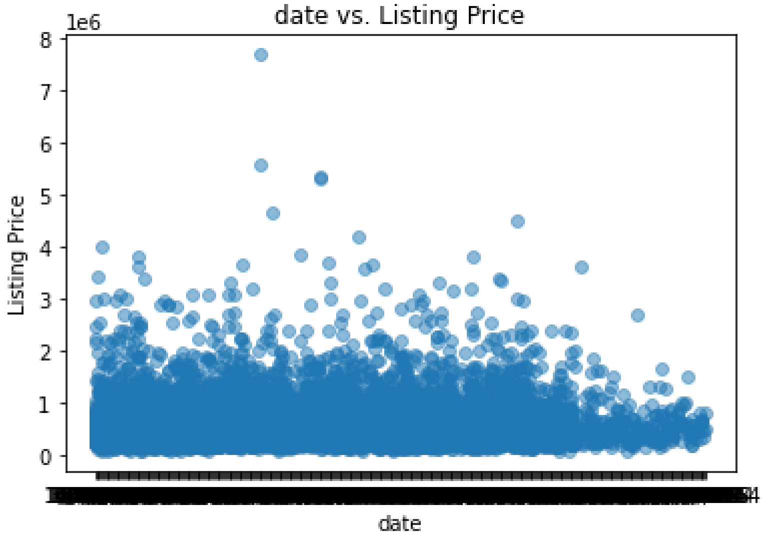
Based on the heatmap. It looks like Bedrooms, Bathrooms, sqft_living, sqft_above, and sqft_living15 are the best predictors. If we wanted to make a full model that uses all of these to predict, it would be the most robust model.

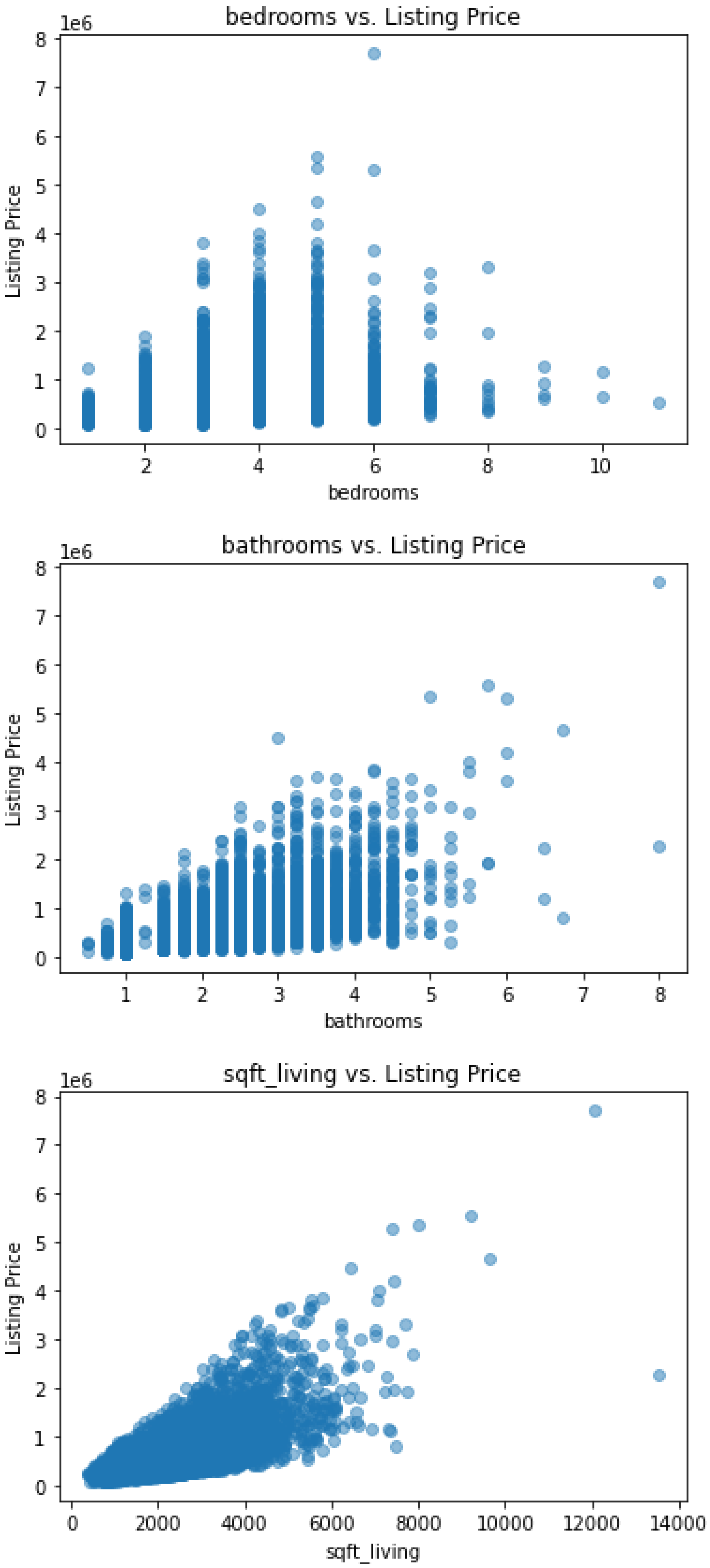
The first model we will create is just a model using Bedroom and Bathrooms because based on user feedback and interaction, those a common features that people sort by

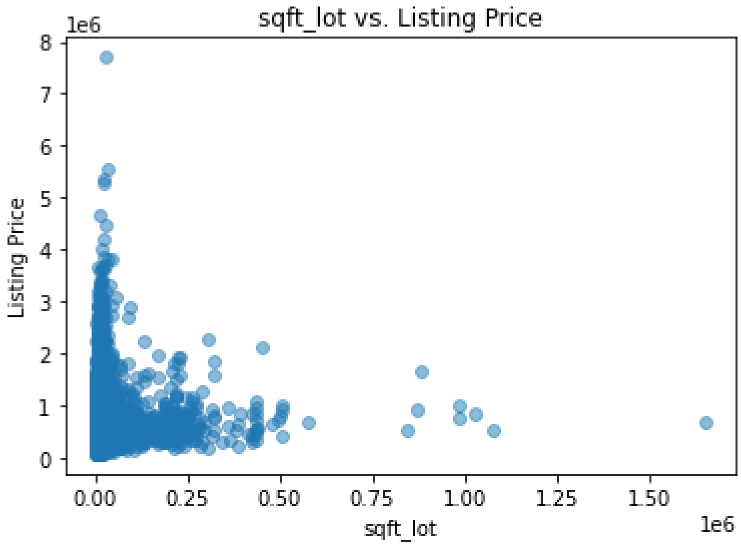
In [28]:
```
most_correlated_feature = 'sqft_living'
```

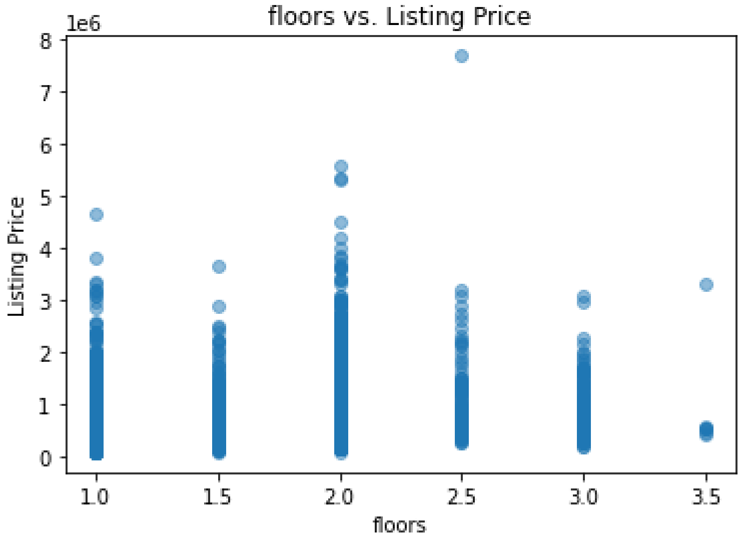
In [29]:
```
#Create graphs for all features vs price

for col in list(X_train.columns):
    fig, ax = plt.subplots()

    ax.scatter(X_train[col], y_train, alpha=0.5)
    ax.set_xlabel(col)
    ax.set_ylabel("Listing Price")
    ax.set_title("{} vs. Listing Price".format(col));
```

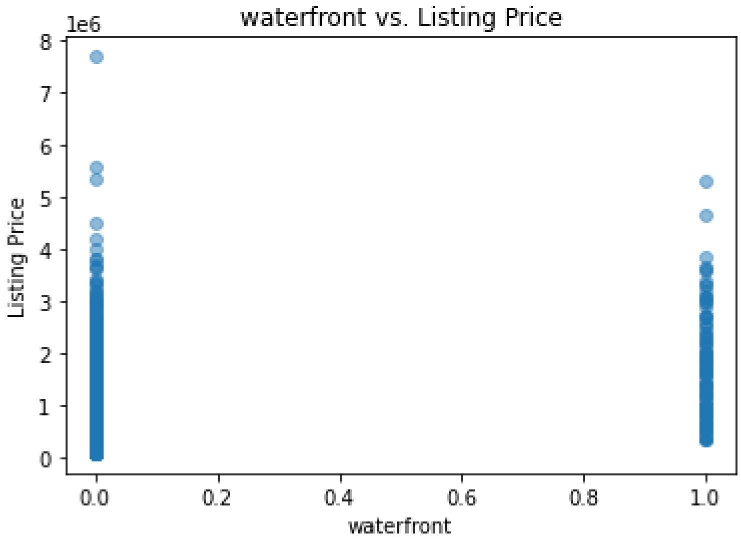


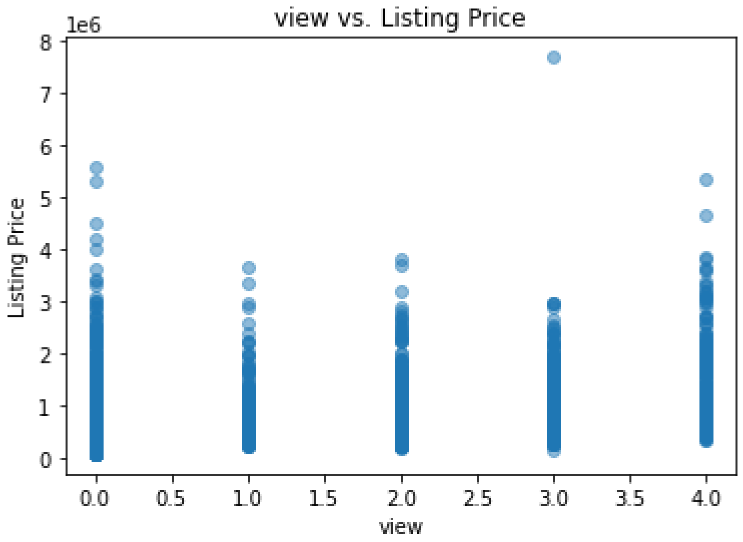


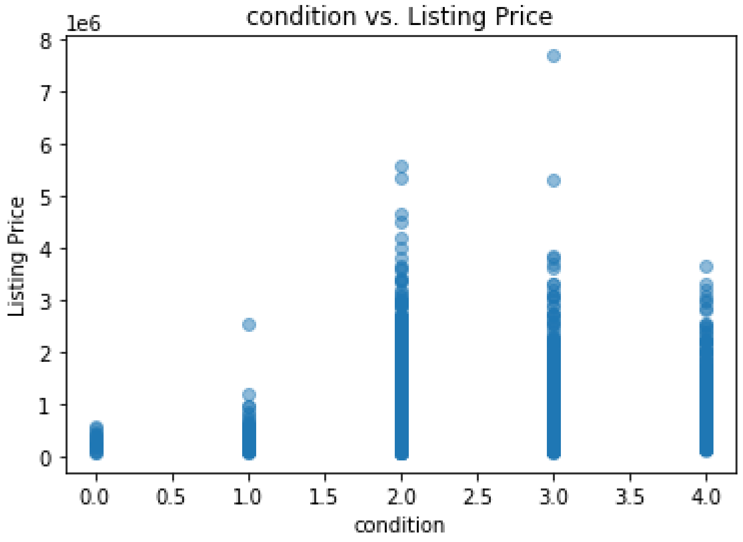

bedrooms vs. Listing Price
1e6
Listing Price
0
1
2
3
4
5
6
7
8
2
4
6
8
10
bedrooms
bathrooms vs. Listing Price
1e6
Listing Price
0
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
bathrooms
sqft_living vs. Listing Price
1e6
Listing Price
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000
10000
12000
14000
sqft_living

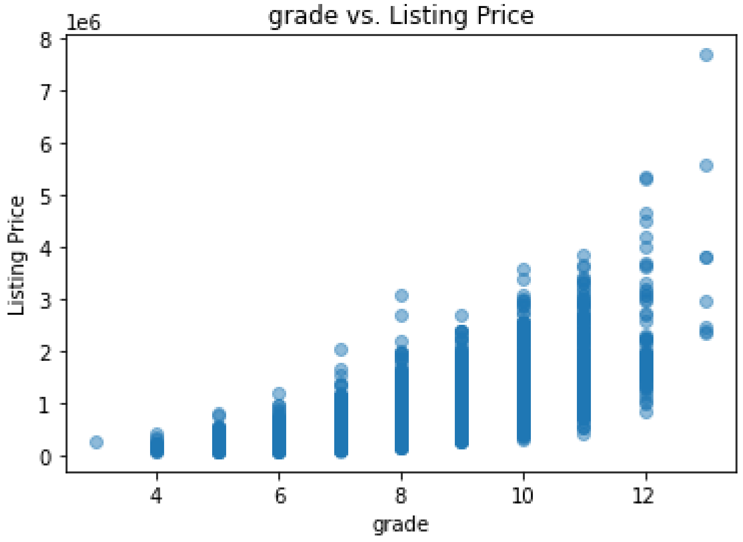

sqft_lot vs. Listing Price
1e6
8
7
6
5
4
3
2
1
0
Listing Price
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1e6
sqft_lot

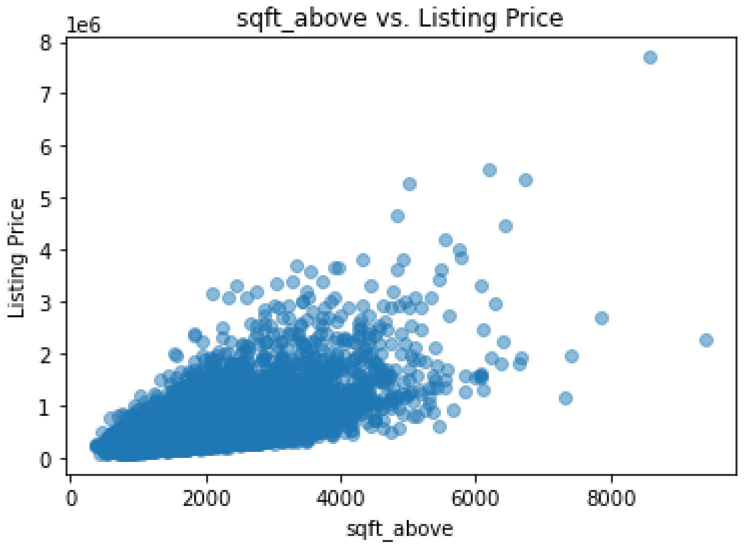

floors vs. Listing Price
1e6
8
7
6
5
4
3
2
1
0
Listing Price
1.0
1.5
2.0
2.5
3.0
3.5
floors

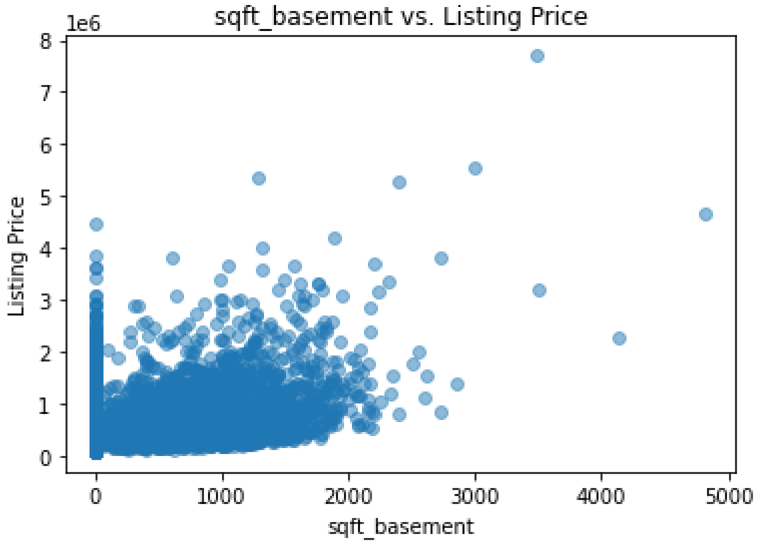

waterfront vs. Listing Price
1e6
8
7
6
5
4
3
2
1
0
Listing Price
0.0
0.2
0.4
0.6
0.8
1.0
waterfront

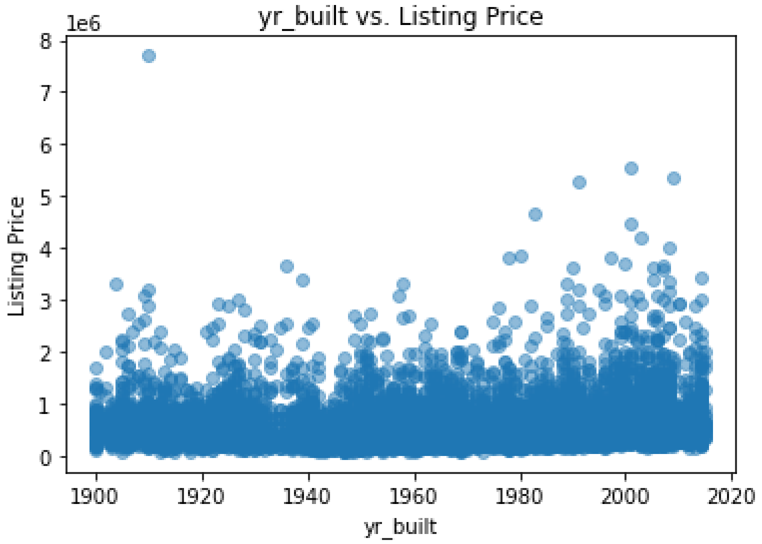

view vs. Listing Price
1e6
Listing Price
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
view

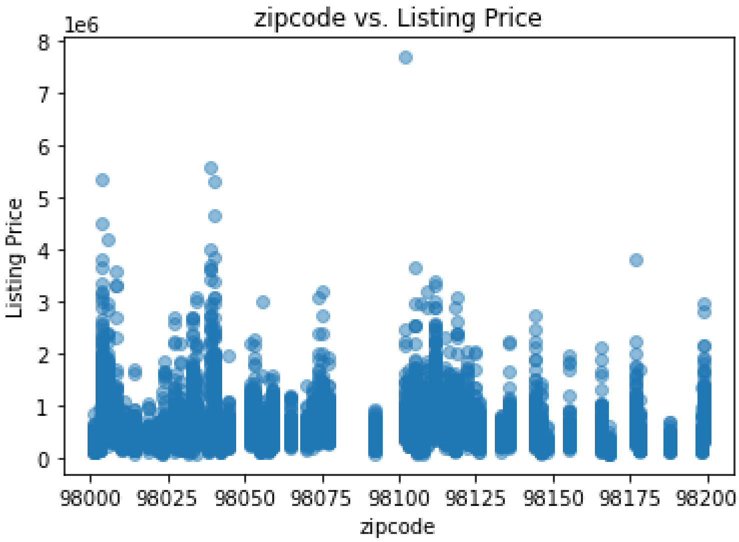

condition vs. Listing Price
1e6
Listing Price
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
condition

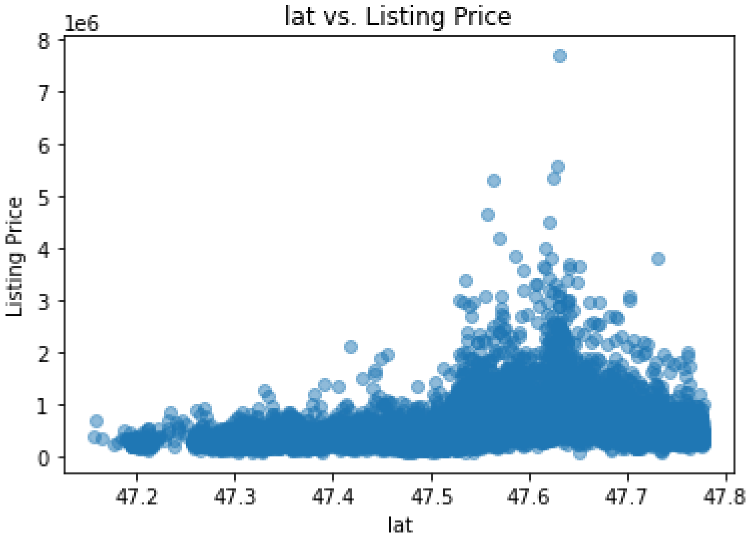

grade vs. Listing Price
1e6
Listing Price
0
1
2
3
4
5
6
7
8
4
6
8
10
12
grade

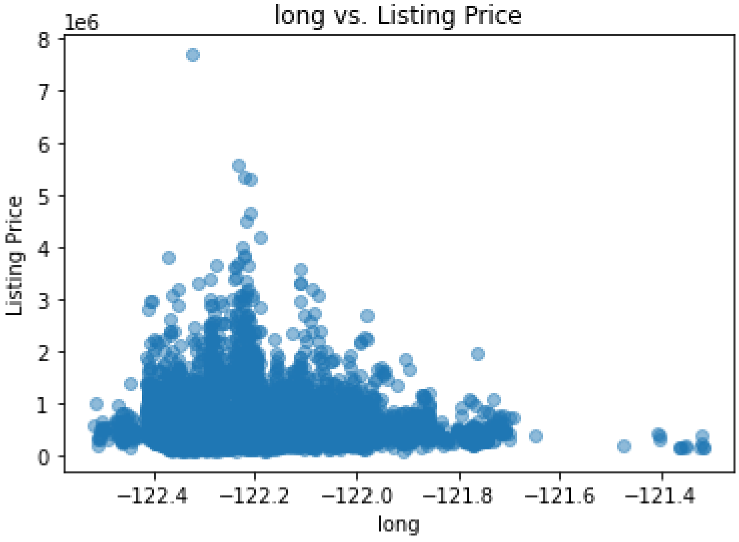

sqft_above vs. Listing Price
1e6
8
7
6
5
4
3
2
1
0
Listing Price
0
2000
4000
6000
8000
sqft_above

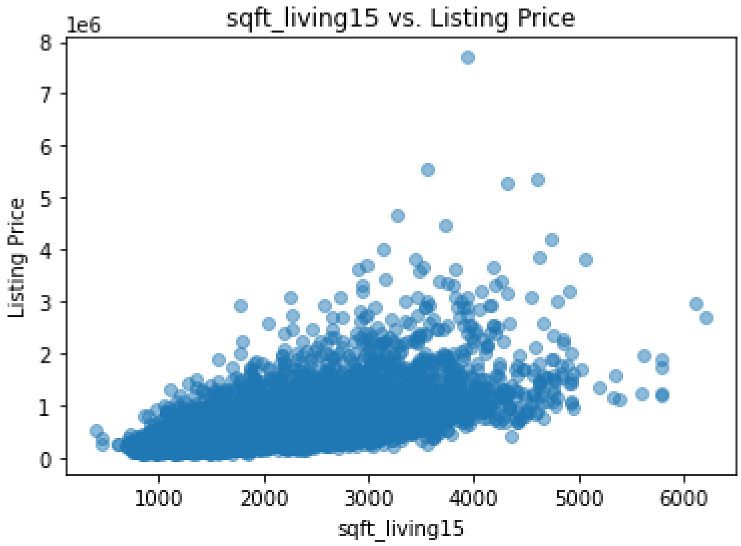

sqft_basement vs. Listing Price
1e6
8
7
6
5
4
3
2
1
0
Listing Price
0
1000
2000
3000
4000
5000
sqft_basement

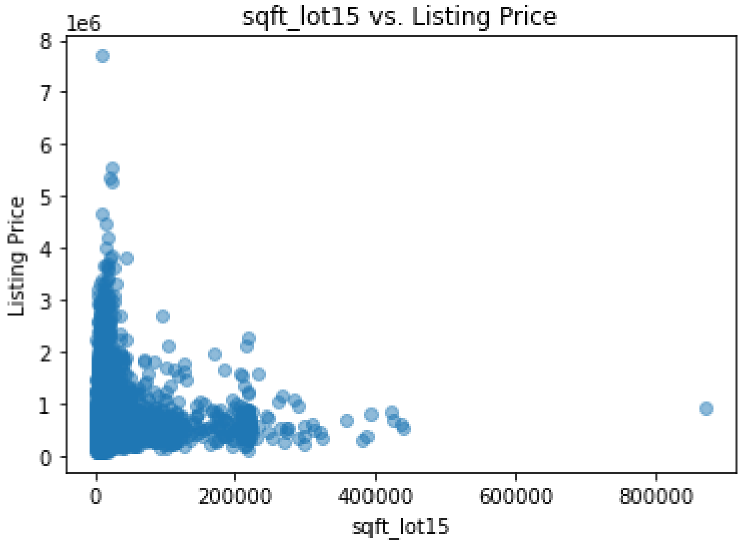

yr_built vs. Listing Price
1e6
8
7
6
5
4
3
2
1
0
Listing Price
1900
1920
1940
1960
1980
2000
2020
yr_built


zipcode vs. Listing Price


lat vs. Listing Price


long vs. Listing Price





Bathrooms, sqft_living, sqft_above, and sqft_living15 are the most linear.

We are going to ignore sqft_living15 because it may overlap too much with sqft_living.

To start, the baseline model will use sqft_living because it is the most correlated feature, then we will start adding others to try and make the model more precise.

In [30]:

```
from sklearn.linear_model import LinearRegression

baseline_model = LinearRegression()
```

In [31]:

```
from sklearn.model_selection import cross_validate, ShuffleSplit

splitter = ShuffleSplit(n_splits=3, test_size=0.25, random_state=0)

baseline_scores = cross_validate(
    estimator=baseline_model,
    X=X_train[[most_correlated_feature]],
    y=y_train,
    return_train_score=True,
    cv=splitter
)
```

```
print("Train score:      ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Train score:      0.4895269677689762
Validation score: 0.4935530672243642
```

In [32]:

```
#R^2 value
baseline_model.fit(X_train[[most_correlated_feature]], y_train)
r_sq = baseline_model.score(X_train[[most_correlated_feature]], y_train)
print('R Squared = ', r_sq)
```

```
R Squared =  0.49055555791820316
```

Ok, the model is accurate only 50% of the time. We will deffinitly need to increase that.

In [33]:

```
#build a model with all numeric features

X_train_numeric = X_train.select_dtypes(include=['float64', 'int64'])

X_train_numeric
```

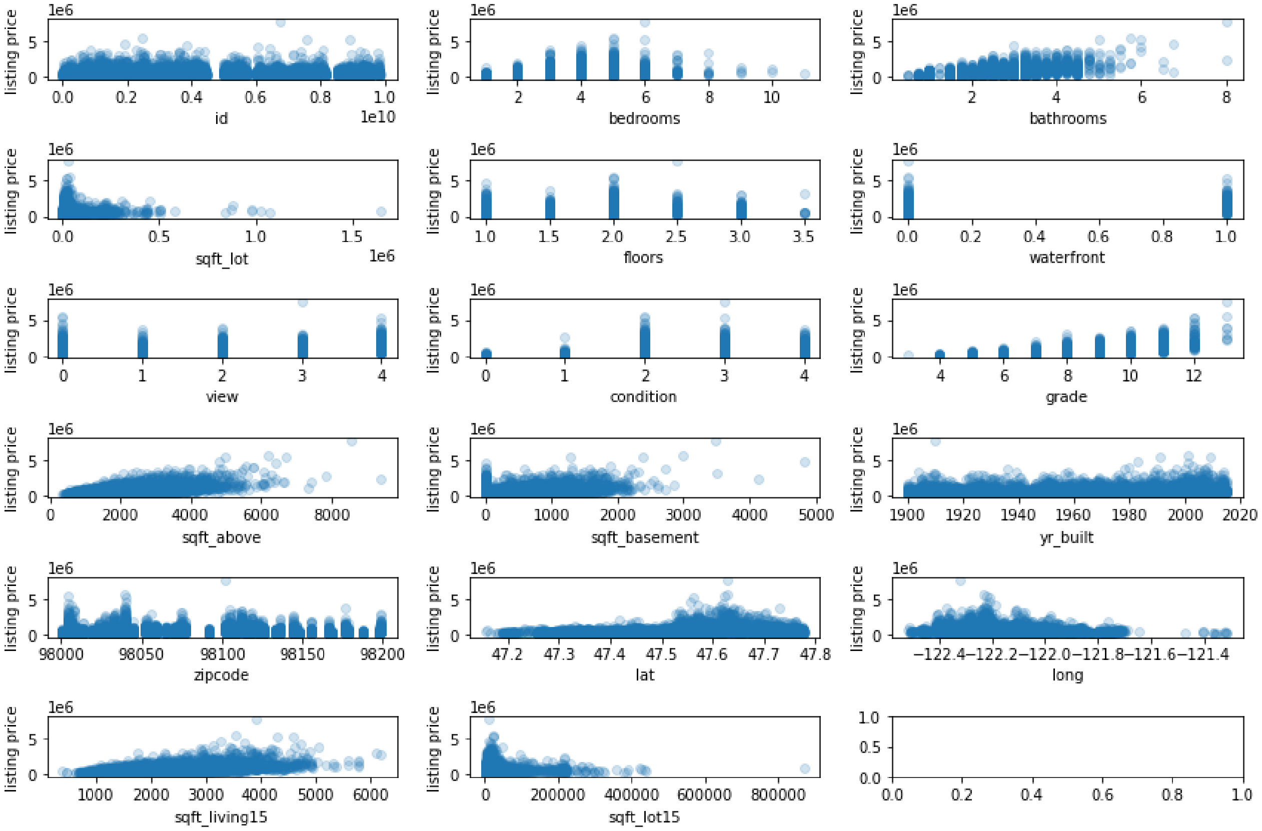
Out[33]:

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condit
6405	3905080280	3	2.50	1880	4499	2.0	0	0	
937	5466420030	3	2.50	2020	6564	1.0	0	0	
19076	2623069010	5	4.00	4720	493534	2.0	0	0	
15201	4443800545	2	2.00	1430	3880	1.0	0	0	
13083	9485930120	3	2.25	2270	32112	1.0	0	0	
...	...	...	...	...	...	...	...	...	
11964	7853230570	3	2.50	2230	5800	2.0	0	0	
21575	4140940150	4	2.75	2770	3852	2.0	0	0	
5390	8658300480	4	1.50	1530	9000	1.0	0	0	
860	1723049033	1	0.75	380	15000	1.0	0	0	
15795	8567450080	4	2.50	2755	11612	2.0	0	0	

16197 rows × 18 columns

In [34]:

```
scatterplot_data = X_train_numeric.drop("sqft_living", axis=1)

fig, axes = plt.subplots(ncols=3, nrows=6, figsize=(12, 8))
fig.set_tight_layout(True)

for index, col in enumerate(scatterplot_data.columns):
    ax = axes[index//3][index%3]
    ax.scatter(X_train_numeric[col], y_train, alpha=0.2)
    ax.set_xlabel(col)
    ax.set_ylabel("listing price")
```



In [35]:

```python
to_drop = ['id', 'lat', 'long', 'yr_built', 'zipcode']
```

In [36]:

```python
X_train_second_model = X_train_numeric.drop(to_drop, axis=1)

X_train_second_model
```

Out[36]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sc
6405	3	2.50	1880	4499	2.0	0	0	2	8	
937	3	2.50	2020	6564	1.0	0	0	2	7	
19076	5	4.00	4720	493534	2.0	0	0	4	9	
15201	2	2.00	1430	3880	1.0	0	0	3	7	
13083	3	2.25	2270	32112	1.0	0	0	3	8	
...	...	...	...	...	...	...	...	...	...	
11964	3	2.50	2230	5800	2.0	0	0	2	7	
21575	4	2.75	2770	3852	2.0	0	0	2	8	
5390	4	1.50	1530	9000	1.0	0	0	3	6	
860	1	0.75	380	15000	1.0	0	0	2	5	
15795	4	2.50	2755	11612	2.0	0	0	2	8	

16197 rows × 13 columns

In [37]:

```python
second_model = LinearRegression()

second_model_scores = cross_validate(
    estimator=second_model,
    X=X_train_second_model,
    y=y_train,
    return_train_score=True,
    cv=splitter
)

print("Current Model")
print("Train score:      ", second_model_scores["train_score"].mean())
print("Validation score:", second_model_scores["test_score"].mean())
print()
print("Baseline Model")
print("Train score:      ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Current Model
Train score:      0.6136456182503751
Validation score: 0.6056878401945676

Baseline Model
Train score:      0.4895269677689762
Validation score: 0.4935530672243642
```

In [38]:

```
second_model.fit(X_train_second_model, y_train)
r_sq = second_model.score(X_train_second_model, y_train)
print('R Squared = ', r_sq)
```

```
R Squared =  0.6119439798013717
```

Ok, better.

In [39]:

```
import statsmodels.api as sm

sm.OLS(y_train, sm.add_constant(X_train_second_model)).fit().summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.612
Model:	OLS	**Adj. R-squared:**	0.612
Method:	Least Squares	**F-statistic:**	1963.
Date:	Mon, 29 Aug 2022	**Prob (F-statistic):**	0.00
Time:	18:14:13	**Log-Likelihood:**	-2.2282e+05
No. Observations:	16197	**AIC:**	4.457e+05
Df Residuals:	16183	**BIC:**	4.458e+05
Df Model:	13		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-6.716e+05	1.87e+04	-35.965	0.000	-7.08e+05	-6.35e+05
bedrooms	-3.623e+04	2576.607	-14.062	0.000	-4.13e+04	-3.12e+04
bathrooms	-1.352e+04	4039.744	-3.346	0.001	-2.14e+04	-5597.122
sqft_living	146.5681	24.942	5.876	0.000	97.679	195.457
sqft_lot	-0.0097	0.063	-0.155	0.877	-0.133	0.113

floors	810.2210	4575.537	0.177	0.859	-8158.338	9778.780
waterfront	6.253e+05	2.36e+04	26.500	0.000	5.79e+05	6.72e+05
view	5.86e+04	2729.978	21.465	0.000	5.32e+04	6.39e+04
condition	5.789e+04	2894.260	20.001	0.000	5.22e+04	6.36e+04
grade	1.048e+05	2727.723	38.425	0.000	9.95e+04	1.1e+05
sqft_above	29.3689	24.873	1.181	0.238	-19.385	78.123
sqft_basement	70.2350	24.716	2.842	0.004	21.788	118.682
sqft_living15	17.3979	4.418	3.938	0.000	8.738	26.058
sqft_lot15	-0.8017	0.096	-8.340	0.000	-0.990	-0.613

Omnibus:	11198.733	**Durbin-Watson:**	1.996
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	537124.795
Skew:	2.760	**Prob(JB):**	0.00
Kurtosis:	30.666	**Cond. No.**	6.56e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.56e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [40]:

```
significant_features = ['bedrooms',
                        'bathrooms',
                        'sqft_living',
                        'waterfront',
                        'view',
                        'floors',
                        'sqft_above',
                        'sqft_lot15',
                        'grade',
                        'condition',
                        'sqft_basement']
```

In [41]:

```
third_model = LinearRegression()
X_train_third_model = X_train[significant_features]

third_model_scores = cross_validate(
    estimator=third_model,
    X=X_train_third_model,
    y=y_train,
    return_train_score=True,
    cv=splitter
)

print("Current Model")
print("Train score:     ", third_model_scores["train_score"].mean())
print("Validation score:", third_model_scores["test_score"].mean())
print()
print("Second Model")
print("Train score:      ", second_model_scores["train_score"].mean())
print("Validation score:", second_model_scores["test_score"].mean())
print()
print("Baseline Model")
print("Train score:      ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Current Model
Train score:      0.6132435248894742
Validation score: 0.6054904093503676

Second Model
Train score:      0.6136456182503751
Validation score: 0.6056878401945676

Baseline Model
Train score:      0.4895269677689762
Validation score: 0.4935530672243642
```

In [42]:

```
second_model.fit(X_train_third_model, y_train)
r_sq = second_model.score(X_train_third_model, y_train)
print('R Squared = ', r_sq)
```

```
R Squared =  0.6115684314181618
```

Not much better....

In [43]:

```
from sklearn.feature_selection import RFECV
from sklearn.preprocessing import StandardScaler

# Importances are based on coefficient magnitude, so
# we need to scale the data to normalize the coefficients
X_train_for_RFECV = StandardScaler().fit_transform(X_train_second_model)

model_for_RFECV = LinearRegression()

# Instantiate and fit the selector
selector = RFECV(model_for_RFECV, cv=splitter)
selector.fit(X_train_for_RFECV, y_train)

# Print the results
print("Was the column selected?")
for index, col in enumerate(X_train_second_model.columns):
    print(f"{col}: {selector.support_[index]}")
```

```
Was the column selected?
bedrooms: True
bathrooms: True
sqft_living: True
sqft_lot: False
floors: False
waterfront: True
view: True
condition: True
grade: True
sqft_above: True
sqft_basement: True
sqft_living15: True
sqft_lot15: True
```

Lets create a model using what the feature selection tells us is best

In [44]:

```
fs_chosen_features = ['bedrooms',
                      'bathrooms',
                      'sqft_living',
                      'waterfront',
                      'view',
                      'condition',
                      'grade',
                      'sqft_above',
                      'sqft_basement',
                      'sqft_living15',
                      'sqft_lot15',
                     ]
```

In [46]:

```
fourth_model = LinearRegression()
X_train_fourth_model = X_train[fs_chosen_features]

fourth_model_scores = cross_validate(
    estimator=fourth_model,
    X=X_train_fourth_model,
    y=y_train,
    return_train_score=True,
    cv=splitter
)

print("Current (Fourth) Model")
print("Train score:     ", fourth_model_scores["train_score"].mean())
print("Validation score:", fourth_model_scores["test_score"].mean())
print()
print("Third Model")
print("Train score:     ", third_model_scores["train_score"].mean())
print("Validation score:", third_model_scores["test_score"].mean())
print()
print("Second Model")
print("Train score:     ", second_model_scores["train_score"].mean())
print("Validation score:", second_model_scores["test_score"].mean())
print()
print("Baseline Model")
print("Train score:     ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Current (Fourth) Model
Train score:      0.6136335994144271
Validation score: 0.6057737894191472

Third Model
Train score:      0.6132435248894742
Validation score: 0.6054904093503676

Second Model
Train score:      0.6136456182503751
Validation score: 0.6056878401945676

Baseline Model
Train score:      0.4895269677689762
Validation score: 0.4935530672243642
```

In [47]:

```
X_test_final = X_test[fs_chosen_features]
```

In [49]:

```
# Fit the model on X_train_final and y_train
fourth_model.fit(X_train_fourth_model, y_train)
```

```
# Score the model on X_test_final and y_test
# (use the built-in .score method)
fourth_model.score(X_test_final, y_test)
```

```
Out[49]: 0.5924243699025344
```

```
In [50]: from sklearn.metrics import mean_squared_error

         mean_squared_error(y_test, fourth_model.predict(X_test_final), squared=False)
```

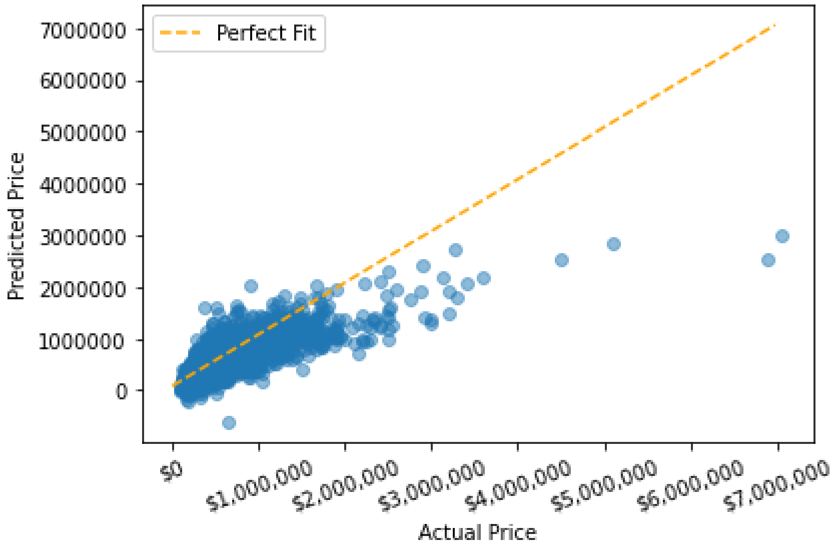
```
Out[50]: 236472.5321385657
```

Our model is off by $236,415 on average.

```
In [51]: print(pd.Series(fourth_model.coef_, index=X_train_fourth_model.columns, name="Co
         print()
         print("Intercept:", fourth_model.intercept_)
```

```
bedrooms          -36245.048784
bathrooms         -13252.857756
sqft_living          146.228602
waterfront        625330.244612
view               58595.293495
condition          57817.957608
grade             104890.368993
sqft_above            29.751469
sqft_basement         70.215330
sqft_living15         17.321045
sqft_lot15            -0.813212
Name: Coefficients, dtype: float64

Intercept: -671173.2200228107
```

```
In [52]: preds = fourth_model.predict(X_test_final)
         fig, ax = plt.subplots()

         perfect_line = np.arange(y_test.min(), y_test.max())
         ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
         ax.scatter(y_test, preds, alpha=0.5)
         ax.set_xlabel("Actual Price")
         ax.set_ylabel("Predicted Price")
         fmt = '${x:,.0f}'
         plt.ticklabel_format(style='plain')
         tick = mtick.StrMethodFormatter(fmt)
         ax.xaxis.set_major_formatter(tick)
         plt.xticks(rotation = 20)
         ax.legend();
```



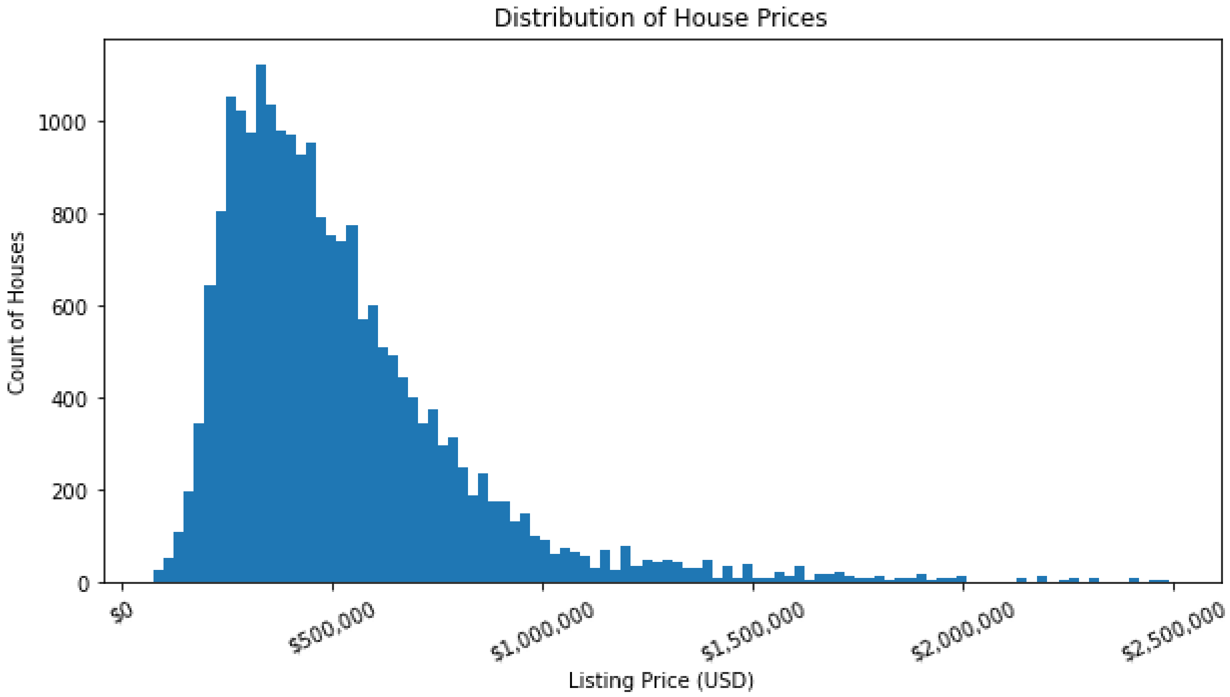
Testing eliminating outliars (Houses > $2,500,000)

```
In [53]: df_clean = data.loc[data['price']<2500000]
```

```
In [54]: df_clean['price'].describe()
```

```
Out[54]: count    2.149500e+04
         mean     5.274094e+05
         std      3.101881e+05
         min      7.800000e+04
         25%      3.200000e+05
         50%      4.500000e+05
         75%      6.400000e+05
         max      2.490000e+06
         Name: price, dtype: float64
```

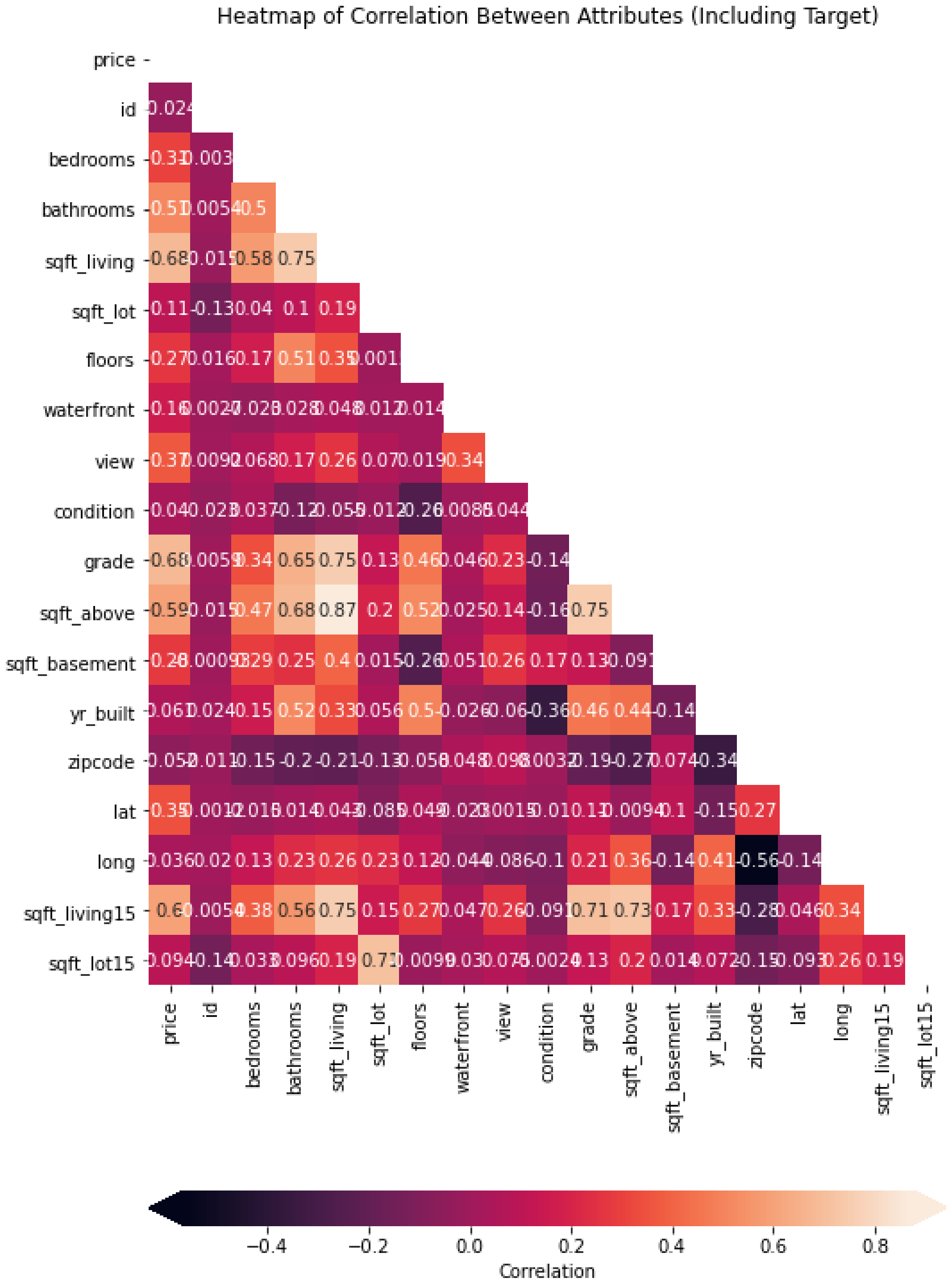
```
In [55]: fig, ax = plt.subplots(figsize=(10, 5))

         ax.hist(df_clean['price'], bins=100)

         ax.set_xlabel("Listing Price (USD)")
         ax.set_ylabel("Count of Houses")
         ax.set_title("Distribution of House Prices");

         fmt = '${x:,.0f}'
         plt.ticklabel_format(style='plain')
         tick = mtick.StrMethodFormatter(fmt)
         ax.xaxis.set_major_formatter(tick)
         plt.xticks(rotation=25)

         plt.show();
```



In [56]:
```
X = df_clean.drop(['price'], axis=1)
y = df_clean['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random
```

In [57]:
```
X_train = X_train.select_dtypes(include=['float64', 'int64'])
```

In [58]:
```
#Create a correlation heatmap

# Create a df with the target as the first column,
# then compute the correlation matrix
heatmap_data = pd.concat([y_train, X_train], axis=1)
corr = heatmap_data.corr()

# Set up figure and axes
fig, ax = plt.subplots(figsize=(8, 14))

# Plot a heatmap of the correlation matrix, with both
# numbers and colors indicating the correlations
sns.heatmap(
    # Specifies the data to be plotted
    data=corr,
    # The mask means we only show half the values,
    # instead of showing duplicates. It's optional.
    mask=np.triu(np.ones_like(corr, dtype=bool)),
    # Specifies that we should use the existing axes
    ax=ax,
    # Specifies that we want labels, not just colors
    annot=True,
    # Customizes colorbar appearance
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .15, "
)

# Customize the plot appearance
ax.set_title("Heatmap of Correlation Between Attributes (Including Target)");
```



In [59]:

```
# Importances are based on coefficient magnitude, so
# we need to scale the data to normalize the coefficients
X_train_for_RFECV = StandardScaler().fit_transform(X_train)

model_for_RFECV = LinearRegression()

# Instantiate and fit the selector
selector = RFECV(model_for_RFECV, cv=splitter)
selector.fit(X_train_for_RFECV, y_train)

# Print the results
print("Was the column selected?")
for index, col in enumerate(X_train.columns):
    print(f"{col}: {selector.support_[index]}")
```

```
Was the column selected?
id: True
bedrooms: True
bathrooms: True
sqft_living: True
sqft_lot: True
floors: True
waterfront: True
view: True
condition: True
grade: True
sqft_above: True
sqft_basement: True
yr_built: True
zipcode: True
lat: True
long: True
sqft_living15: True
sqft_lot15: True
```

In [60]:

```
# Best features from the first try
selected_feats = ['bedrooms',
                  'bathrooms',
                  'sqft_living',
                  'floors',
                  'yr_built',
                  'zipcode',
                  'lat',
                  'waterfront',
                  'view',
                  'condition',
                  'grade',
                  'long',
                  'sqft_lot15',
                  'sqft_basement',
                  'sqft_living15']
```

In [61]:

```
X_train_final = X_train[selected_feats]
X_test_final = X_test[selected_feats]
```

In [ ]:

In [64]:

```
final_model = LinearRegression()


final_model_scores = cross_validate(
    estimator=final_model,
    X=X_train_final,
    y=y_train,
    return_train_score=True,
    cv=splitter
)

print("Current (Final) Model")
print("Train score:     ", final_model_scores["train_score"].mean())
print("Validation score:", final_model_scores["test_score"].mean())


# Fit the model on X_train_final and y_train
final_model.fit(X_train_final, y_train)
```

```
# Score the model on X_test_final and y_test
r_sq = final_model.score(X_test_final, y_test)
print('R Squared = ', r_sq)
```

```
Current (Fourth) Model
Train score:      0.7071926968115884
Validation score: 0.7055658747368838
R Squared =  0.7183857539053213
```

Finally! We have a model scoring over 70%! We could probably get even better if we cut the data down even further. Say, only houses between $0and$1,000,000.

In [65]:

```
mean_squared_error(y_test, final_model.predict(X_test_final), squared=False)
```

Out[65]:

```
165439.472512914
```

An error of $165,000 is much much better than what we previously had.

In [67]:

```
print(pd.Series(final_model.coef_, index=X_train_final.columns, name="Coefficien
print()
print("Intercept:", fourth_model.intercept_)
```

```
bedrooms          -21016.715028
bathrooms          35539.570929
sqft_living          123.451515
floors             22004.216294
yr_built           -2516.661579
zipcode             -494.025867
lat               593842.191021
waterfront        317955.846862
view               54597.398038
condition          24353.840348
grade              96752.929805
long             -159670.419550
sqft_lot15            -0.068957
sqft_basement        -18.257592
sqft_living15         40.380357
Name: Coefficients, dtype: float64

Intercept: -671173.2200228107
```

These are the coefficients for each feature

The two most correlated features are sqft_living and grade. Every 1 square foot you increase in the living space of the house, the price will go up by $123. For every grade you increase in the house (i.e. increase the design, look, construction), the price will go up by \$96,752

Testing reducing max from $2,500,000 to \$1,000,000

In [68]:

```
df_clean_2 = data.loc[data['price']<1000000]
```

In [69]:

```
X = df_clean_2.drop(['price'], axis=1)
y = df_clean_2['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random

X_train = X_train.select_dtypes(include=['float64', 'int64'])
```

```
In [70]: X_train = X_train[selected_feats]
         X_test = X_test[selected_feats]
```

```
In [71]: max_one_mill_model = LinearRegression()

         # Fit the model on X_train_final and y_train
         max_one_mill_model.fit(X_train, y_train)

         # Score the model on X_test_final and y_test
         max_one_mill_model.score(X_test, y_test)
```

```
Out[71]: 0.6902043348747593
```

Why did it not get better? Am I using worse predictors for this model?

```
In [72]: X_train_for_RFECV = StandardScaler().fit_transform(X_train)

         model_for_RFECV = LinearRegression()

         # Instantiate and fit the selector
         selector = RFECV(model_for_RFECV, cv=splitter)
         selector.fit(X_train_for_RFECV, y_train)

         # Print the results
         print("Was the column selected?")
         for index, col in enumerate(X_train.columns):
             print(f"{col}: {selector.support_[index]}")
```

```
Was the column selected?
bedrooms: True
bathrooms: True
sqft_living: True
floors: True
yr_built: True
zipcode: True
lat: True
waterfront: True
view: True
condition: True
grade: True
long: True
sqft_lot15: True
sqft_basement: True
sqft_living15: True
```

Weird. I don't know why this is worse than the previous model.

All in all, I'm happy with obtaining a model that can accuratly predict near 70%

Conclusion

Final Model Discussion

The best model and the one we will present to Blackrock had a score of 70% and a mean squares error of $165,000. We could've got to this model faster if we immedietly eliminated outliars, but we chose to leave them in hoping they would provide the model accuracy at higher price values. Instead, these values only muddied the predictions on the bulk of the data. Once the outliars were eliminated, the new model was able to increase its score from 52 to 64.5 percent.

The final model used all of the numeric features in the dataFrame. The heatmap we generated showed that even the best 3 features were only had around .60 - .70 correlation, but all of the features ended up being useful. We used a selecter library to choose the features for this model.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Testing - If a rennovation increases the grade of the home by 1, how much would the price of the house increase?

```
In [73]: data['grade'].value_counts()
```

```
Out[73]: 7     8974
         8     6065
         9     2615
         6     2038
         10    1134
         11     399
         5      242
         12      89
         4       27
         13      13
         3        1
         Name: grade, dtype: int64
```

```
In [74]: df_grade_test = data.copy()
```

Process forward:

Train on

- Key value: Price (before rennovation)
- Variable(s): grade

predict on

- Key value: Price
- Variable(s): grade+1

```
In [75]: relevant_cols = ['price', 'grade']
```

```
In [76]: df_grade_test = df_grade_test[relevant_cols]
```

In [77]:

```
df_grade_test['grade'].value_counts()
```

Out[77]:

```
7     8974
8     6065
9     2615
6     2038
10    1134
11     399
5      242
12      89
4       27
13      13
3        1
Name: grade, dtype: int64
```

In [78]:

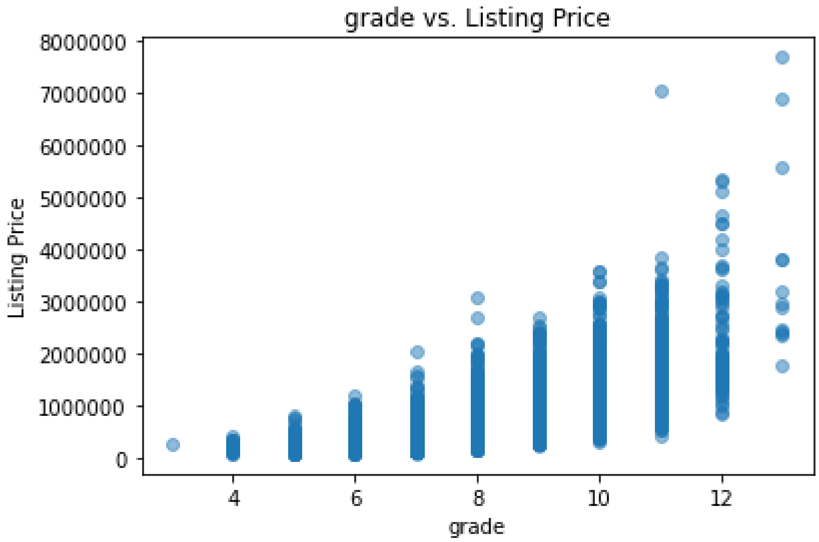
```
df_grade_test.sort_values(by = 'grade', ascending=True, inplace=True)
```

In [ ]:

```

```

In [79]:

```
fig, ax = plt.subplots()
ax.scatter(df_grade_test['grade'], df_grade_test['price'], alpha=0.5)
ax.set_xlabel('grade')
ax.set_ylabel("Listing Price")
ax.set_title("{} vs. Listing Price".format('grade'))
ax.ticklabel_format(useOffset=False, style='plain')
```



In [80]:

```
grading_model = LinearRegression()
```

In [81]:

```
x = df_grade_test.drop('price', axis=1)
y = df_grade_test['price']
```

In [82]:

```
grading_model.fit(x, y)
```

Out[82]:

```
LinearRegression()
```

In [83]:

```
grading_scores = cross_validate(
    estimator=grading_model,
    X=x,
    y=y,
    return_train_score=True,
    cv=splitter
)
```

In [84]:

```
print("grading_scores Model")
print("Train score:      ", grading_scores["train_score"].mean())
print("Validation score:", grading_scores["test_score"].mean())
```

```
grading_scores Model
Train score:      0.44150939682519513
Validation score: 0.46105647482608525
```

In [85]:

```
mean_squared_error(y, grading_model.predict(x), squared=False)
```

Out[85]:

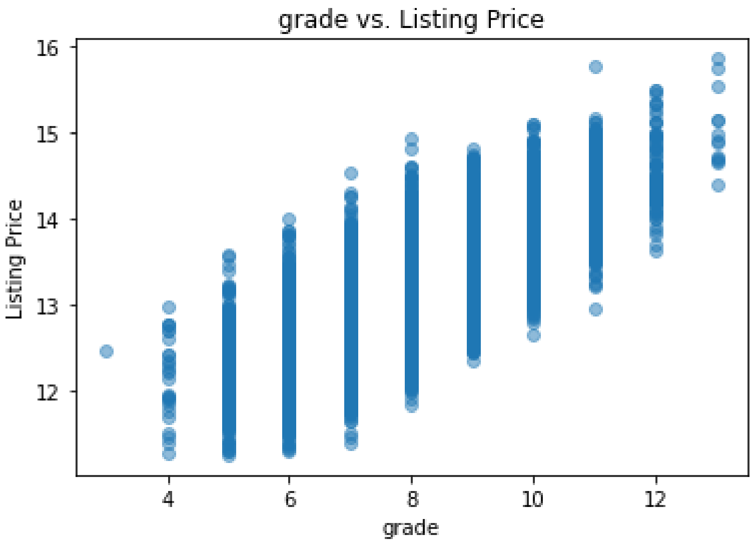
```
273391.0461744017
```

In [86]:

```
df_grade_test['price'] = df_grade_test['price'].apply(np.log)
```

In [87]:

```
fig, ax = plt.subplots()
ax.scatter(df_grade_test['grade'], df_grade_test['price'], alpha=0.5)
ax.set_xlabel('grade')
ax.set_ylabel("Listing Price")
ax.set_title("{} vs. Listing Price".format('grade'))
ax.ticklabel_format(useOffset=False, style='plain')
```



In [88]:

```
grading_model.fit(x, y)
```

Out[88]:

```
LinearRegression()
```

In [89]:

```
grading_scores = cross_validate(
    estimator=grading_model,
    X=x,
    y=y,
    return_train_score=True,
    cv=splitter
)
```

In [90]:

```
print("grading_scores Model")
print("Train score:      ", grading_scores["train_score"].mean())
print("Validation score:", grading_scores["test_score"].mean())
```

```
grading_scores Model
```

```
Train score:      0.4949088163897493
Validation score: 0.4954648224459261
```

Testing 2 - Does rennovation recency have a strong relationship and predictive relationship with sale price

How to do this? Create groups of houses with the yr_renovated every 5 or 10 years. Then use sqft as another predictor because we want to compare similar houses. Then run a regression model

In [91]:
```
data = pd.read_csv('data/kc_house_data.csv')
data['yr_renovated'].fillna(0, inplace=True)
```

In [92]:
```
data['yr_renovated'] = data['yr_renovated'].astype('int64')
```

In [93]:
```
data['yr_renovated'].value_counts()
```

Out[93]:
```
0       20853
2014       73
2003       31
2013       31
2007       30
        ...  
1976        1
1953        1
1951        1
1946        1
1944        1
Name: yr_renovated, Length: 70, dtype: int64
```

Out of 21597, 20853 have not been renovated.

In [94]:
```
total_reno = 21597-20853
total_reno
```

Out[94]:
```
744
```

In [95]:
```
df_reno = data.copy()
```

In [96]:
```
df_reno = df_reno[df_reno.yr_renovated !=0]
```

In [97]:
```
df_reno['yr_renovated']
```

Out[97]:
```
1        1991
35       2002
95       1991
103      2010
125      1992
         ... 
19602    2004
20041    2006
20428    2009
20431    2014
20946    2007
Name: yr_renovated, Length: 744, dtype: int64
```

In [98]:
```
df_reno['yr_renovated'].value_counts()
```

Out[98]:

```
2014    73
2013    31
2003    31
2007    30
2000    29
        ..
1953     1
1954     1
1959     1
1976     1
1934     1
Name: yr_renovated, Length: 69, dtype: int64
```

In [99]:

```
df_reno['yr_renovated'].describe()
```

Out[99]:

```
count     744.000000
mean     1995.928763
std        15.599946
min      1934.000000
25%      1987.000000
50%      2000.000000
75%      2007.250000
max      2015.000000
Name: yr_renovated, dtype: float64
```

In [100…

```
sum(df_reno['yr_renovated'] >= 2000)
```

Out[100…

```
379
```

In [101…

```
744/379
```

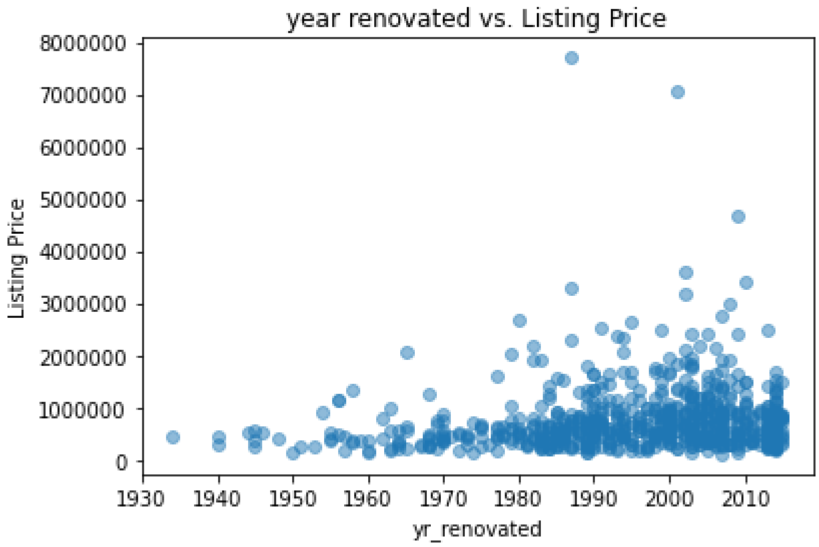
Out[101…

```
1.963060686015831
```

nearly half before/after 2000

In [102…

```
fig, ax = plt.subplots()
ax.scatter(df_reno['yr_renovated'], df_reno['price'], alpha=0.5)
ax.set_xlabel('yr_renovated')
ax.set_ylabel("Listing Price")
ax.set_title("{} vs. Listing Price".format('year renovated'))
ax.ticklabel_format(useOffset=False, style='plain')
```



In [103…

```
rel_cols = ['price', 'yr_renovated', 'sqft_living']
df_reno = df_reno[rel_cols]
```

In [104…

```
df_reno
```

Out[104…

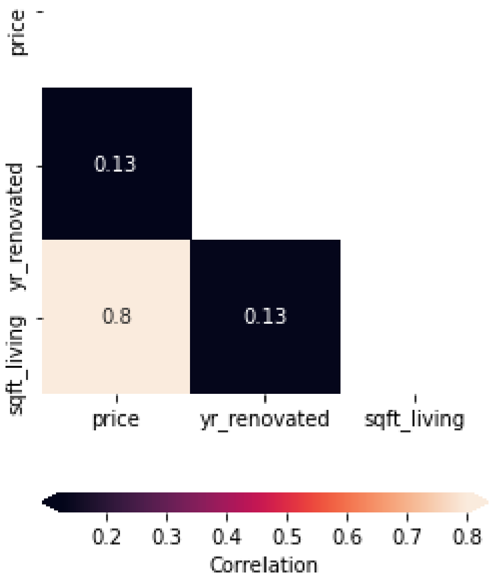
```
           price  yr_renovated  sqft_living
    1   538000.0          1991         2570
   35   696000.0          2002         2300
   95   905000.0          1991         3300
  103  1090000.0          2010         2920
  125  1450000.0          1992         2750
  ...        ...           ...          ...
19602   451000.0          2004          900
20041   434900.0          2006         1520
20428   500012.0          2009         2400
20431   356999.0          2014         1010
20946   110000.0          2007          828
```

744 rows × 3 columns

In [105…

```
#Create a correlation heatmap

# Create a df with the target as the first column,
# then compute the correlation matrix
heatmap_data = pd.concat([df_reno], axis=1)
corr = heatmap_data.corr()

# Set up figure and axes
fig, ax = plt.subplots(figsize=(4, 6))

# Plot a heatmap of the correlation matrix, with both
# numbers and colors indicating the correlations
sns.heatmap(
    # Specifies the data to be plotted
    data=corr,
    # The mask means we only show half the values,
    # instead of showing duplicates. It's optional.
    mask=np.triu(np.ones_like(corr, dtype=bool)),
    # Specifies that we should use the existing axes
    ax=ax,
    # Specifies that we want labels, not just colors
    annot=True,
    # Customizes colorbar appearance
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .15, "
)

# Customize the plot appearance
ax.set_title("Heatmap of Correlation Between Attributes (Including Target)");
```

Heatmap of Correlation Between Attributes (Including Target)



In [106…

```
testing_2_model = LinearRegression()
```

In [107…

```
x = df_reno.drop('price', axis=1)
y = df_reno['price']
```

In [108…

```
testing_2_model.fit(x, y)
```

Out[108…

```
LinearRegression()
```

In [109…

```
testing_2_scores = cross_validate(
    estimator=testing_2_model,
    X=x,
    y=y,
    return_train_score=True,
    cv=splitter
)
```

In [110…

```
print("grading_scores Model")
print("Train score:      ", testing_2_scores["train_score"].mean())
print("Validation score:", testing_2_scores["test_score"].mean())
```

```
grading_scores Model
Train score:      0.6610814476155816
Validation score: 0.535085344003929
```

In [ ]: