CAPSTONE PROJECT 1: FINAL PROJECT

PREDICTING HOUSE PRICE

MEHRETEAB KIDANE
Springboard
June, 2019

INTRODUCTION

Over the past few decades, the housing price has shown increments as times goes by. It is no doubt that the longer the period of time the higher the price of housing. This is a big issue to be taken into consideration. It is considered an important issue because house is one of the important basic human needs. Therefore, being concerned about the housing price is a must and the factors that contribute to the increase of housing price need to be determined.

The following are the objectives of this project:
1. To determine the causes of increase in housing price.
2. To predict the price of housing price in the future.
3. To identify the effect of increase in housing price.

OVERVIEW OF DATA

This project illustrates different approaches to predict house prices using machine learning tools and forecasting algorithms to uncover what really influences the value of a house and achieve the high degree of accuracy in my model. The original dataset can be found Link , in the Kaggle website. This dataset will allow us to learn more about the Housing market and to explore more deeply the most popular machine learning techniques, as well as learning more about the necessary steps to follow in a data science project.

SIGNIFICANCE OF THE PROBLEM

By doing prediction on the housing price, the factors that lead to the increase of housing price can be determined. The housing price in the future can be forecasted as well. The importance of doing this proposal on the housing price is that, it will give positive impact to the government. It can help the government to predict or estimates the price of housing in the future. These can wider the awareness of the price of housing which is now in high rate. This research will also can be an advantage for the housing developers in determining the housing price by identification of the factors that lead to the housing price estimation. Moreover, this prediction is also important to individuals who will purchase house in the future. This will give awareness to the individual regarding the housing price and help them to figure out the best decision in purchasing a house.

DELIVERABLE

The deliverables will be the codes and visualization techniques on GitHub in the form of Jupyter Notebooks, and a slide desk. This will include a report and I intend to write a documentation explaining the code and the results.

DATA CLEANSING AND ANALYSIS

Let's analyze the dataset and take a closer look at its content. The aim here is to find details like the number of columns and other metadata which will help us to gauge size and other properties such as the range of values in the columns of the dataset.

Read the data into a data frame
data = pd.read_csv('. ./input/kc_house_data.csv')

To see the number of rows and columns in the dataset we can use the "shape" attribute as shown below:
data.shape
In the output you should see (21613, 21) which means that our algorithm has 21613 rows and 21 columns.
The description for the features is given below:

id :- It is the unique numeric number assigned to each house being sold.
date :- It is the date on which the house was sold out.
price: - It is the price of house which we have to predict so this is our target variable and apartment from it are our features.
bedrooms: - It determines number of bedrooms in a house.
bathrooms :- It determines number of bathrooms in a bedroom of a house.
sqft_living :- It is the measurement variable which determines the measurement of house in square foot.
sqft_lot : It is also the measurement variable which determines square foot of the lot.
floors: It determines total floors means levels of house.
waterfront: This feature determines whether a house has a view to waterfront 0 means no 1 means yes.
view: This feature determines whether a house has been viewed or not 0 means no 1 means yes.
condition: It determines the overall condition of a house on a scale of 1 to 5.
grade: It determines the overall grade given to the housing unit, based on King County grading system on a scale of 1 to 11.
sqft_above : It determines square footage of house apart from basement.
sqft_basement : It determines square footage of the basement of the house.
yr_built : It determines the date of building of the house.
yr_renovated : It determines year of renovation of house.
zipcode: It determines the zip code of the location of the house.
lat: It determines the latitude of the location of the house.
long: It determines the longitude of the location of the house.
sqft_living15: Living room area in 2015(implies-- some renovations)
sqft_lot15: lotSize area in 2015(implies-- some renovations)

Let's see how the dataset actually looks. To do so, we can use the "head()" function as shown below:
data.head()

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated
0	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	0
1	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	1991
2	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	0
3	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	0
4	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	0

To check if there are any null values in the dataset

Now it’s time for some general data inspection. Let's first examine to see if there are any nulls in the dataframe as well as look at the type of the data (i.e whether it is a string or numeric)

```
# Looking for nulls
print(data.isnull().any())
# Inspecting type
print(data.dtypes)
```

The data is pretty clean. There are no pesky nulls which we need to treat and most of the features are in numeric format. Let's go ahead and drop the "id" and "date" columns as these 2 features will not be used in this analysis.

MISSING DATA

The answer to these questions is important for practical reasons because missing data can imply a reduction of the sample size. This can prevent us from proceeding with the analysis. Moreover, from a substantive perspective, we need to ensure that the missing data process is not biased and hiding an inconvenient truth. The good news is our data set contains no missing values.

```
#missing data
total = df_house.isnull().sum().sort_values(ascending=False)
percent =
(df_house.isnull().sum()/df_house.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

OUTLIERS DETECTION

Outliers were detected and analyzed using the outlier box plots. From the outlier box plot I inferred that the data consist of many outliers for the target and price variables. However the outliers for the price variable corresponded to outliers for Numbers of bedrooms, number of bathrooms and square feet living.

FINDING CORRELATION

In this step we check by finding correlation of all the features with respect to target variable i.e., price to see whether they are positively correlated or negatively correlated to find if they help in prediction process in model building process or not. But this is also one of the most important step as it also involves domain knowledge of the field of the data means you cannot simply remove the feature from your prediction process just because it is negatively correlated because it may contribute in future prediction for this you should take help of some domain knowledge personnel.

As **id** and **date** columns are not important to predict price so we are discarding it for finding correlation

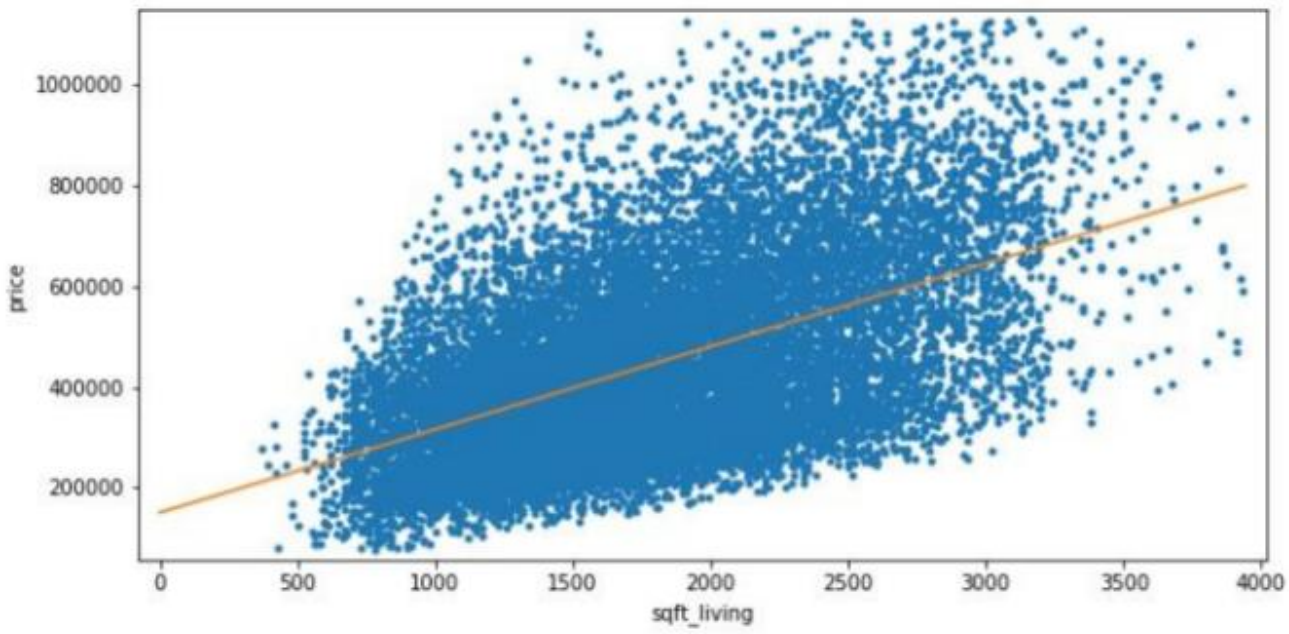
In [151]:
```
# Finding Correlation of price with other variables to see
#how many variables are strongly correlated with price
correlations = {}
for f in features:
    data_temp = data[[f,target]]
    x1 = data_temp[f].values
    x2 = data_temp[target].values
    key = f + ' vs ' + target
    correlations[key] = pearsonr(x1,x2)[0]
```

In [149]:
```
# Printing all the correlated features value with respect to price which is target variable
data_correlations = pd.DataFrame(correlations, index=['Value']).T
data_correlations.loc[data_correlations['Value'].abs().sort_values(ascending=False).index]
```

Out[149]:

	Value
sqft_living vs price	0.702035
grade vs price	0.667434
sqft_above vs price	0.605567
sqft_living15 vs price	0.585379
bathrooms vs price	0.525138
view vs price	0.397293
sqft_basement vs price	0.323816
bedrooms vs price	0.308350
lat vs price	0.307003
waterfront vs price	0.266369
floors vs price	0.256794
yr_renovated vs price	0.126434
sqft_lot vs price	0.089661
sqft_lot15 vs price	0.082447
yr_built vs price	0.054012
zipcode vs price	-0.053203
condition vs price	0.036362
long vs price	0.021626

As zipcode is negatively correlated with sales price, so we can discard it for sales price prediction.



Price vs sqft_living plot also shows there is a positive correlation.

EXPLORATORY DATA ANALYSIS INFERENTIAL STATISTICS

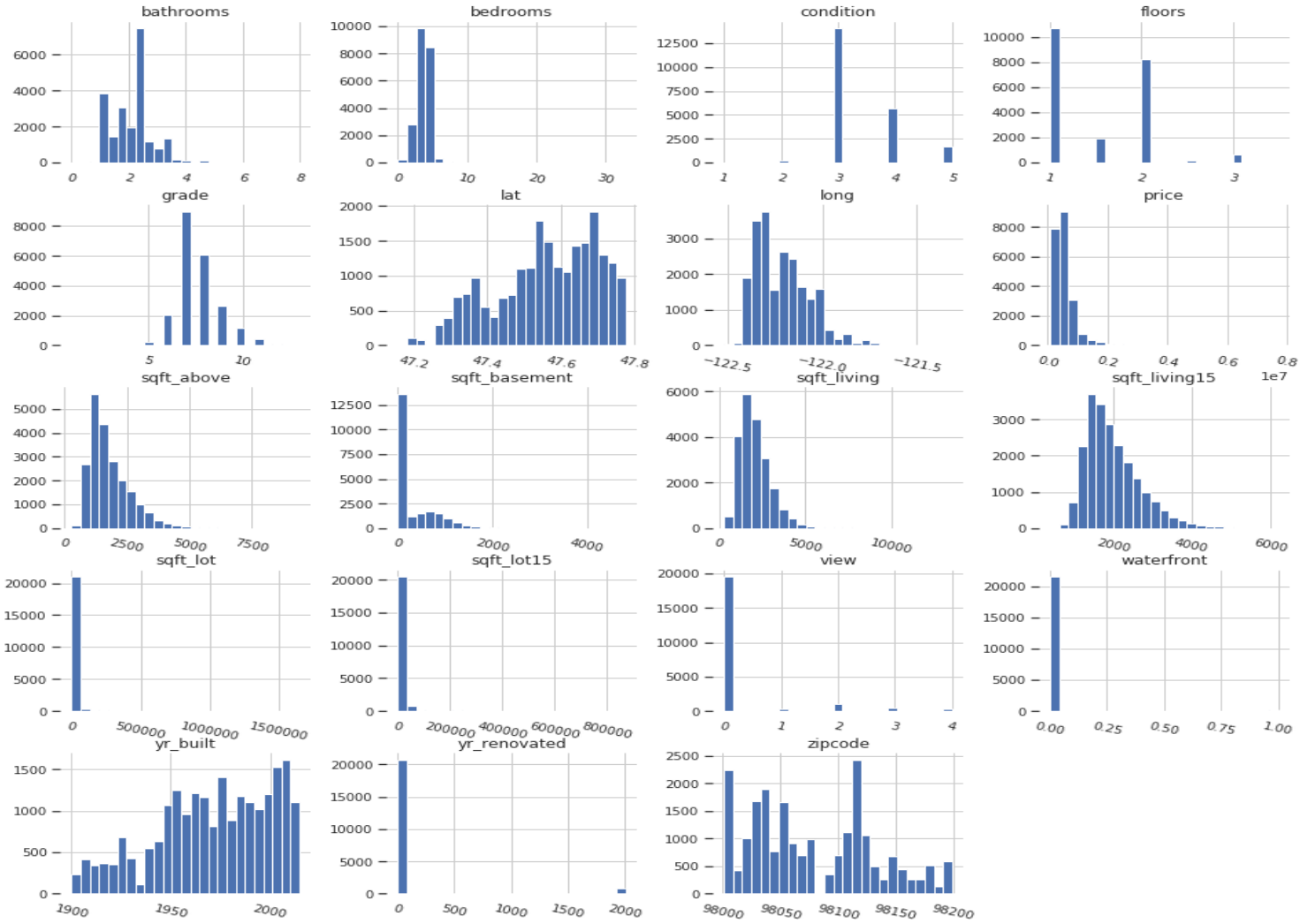
From the dataset the most expensive house is priced at 7,700,000 and the least expensive house is priced 75,000. The average or mean price of a house is 540,088.14 and the median price of a house is 450,000. I plotted the histogram of price and it is right skewed. This indicates that there might be some outliers in the dataset. I also plotted a boxplot of price to check for outliers and found there are some outliers. Then I checked for outliers for other features like the number of bedrooms, the number of bathrooms, sqft_living, sqft_lot, floors, condition, grade, sqft_above, sqft_basement. Except for the column number of floors all other columns show that there might be some outliers. Then I converted the date column from string to datetime data type. Then I checked how many houses were sold in 2014 and 2015. There were 14633 houses sold in the year 2014 and 6980 houses sold in the year 2015.

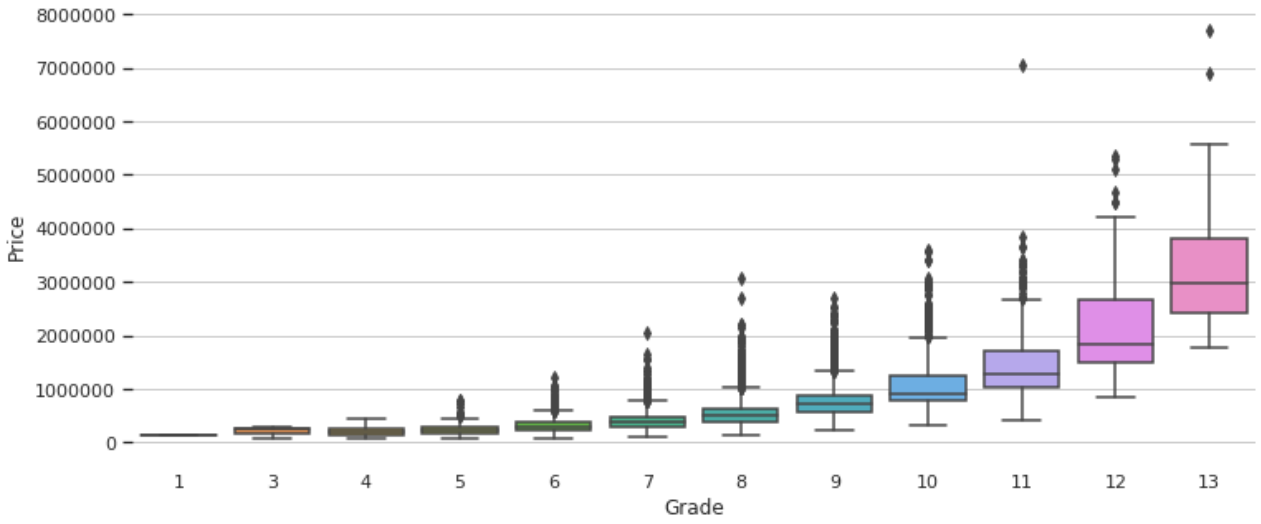
To gain a sense of the relationship of the features with each other and with house prices, the target variable, I tried to use a diverse set of data visualization tools, including the following boxplots, histogramplots, scatter plots and correlation plots. The first EDA I performed was to examine the distribution of the home sale prices. The price of a house is dependent on various factors like size of area, how many bedrooms, location and many other factors. I conducted a hypothesis test to check if there is no significant correlation between a number of bedrooms and price. The p-value for the hypothesis test is less than the level of significance 0.05, so I rejected the null hypothesis. So I support that there is a correlation between a number of bedrooms and price. I also conducted a hypothesis test to check the correlation between a number of bathrooms and price. The p-value for the hypothesis test is less than the level of significance 0.05, so I reject the null hypothesis and suggest that there is a correlation between a number of bathrooms and price.

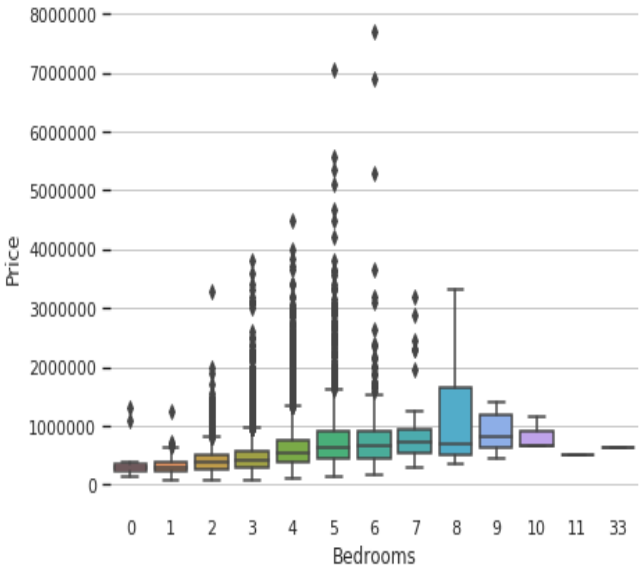
I started with the histograms of dataframe

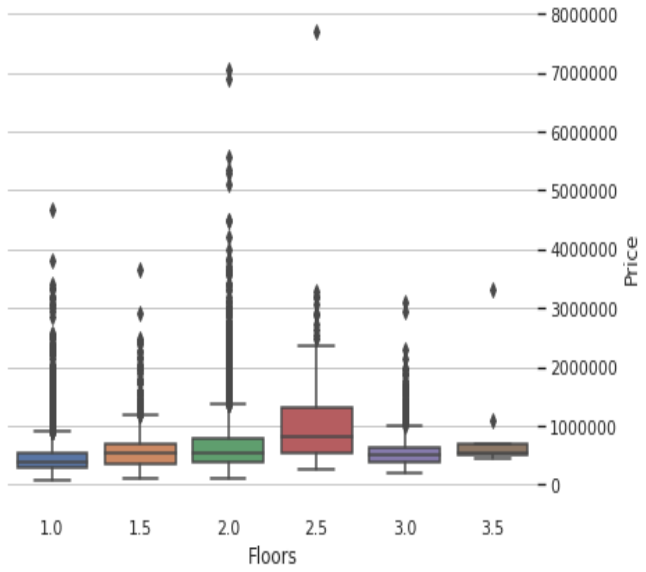
```
df=df[['price', 'bedrooms', 'bathrooms', 'sqft_living',
    'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
    'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
    'lat', 'long', 'sqft_living15', 'sqft_lot15']]
h = df.hist(bins=25,figsize=(16,16),xlabelsize='10',ylabelsize='10',xrot=-15)
sns.despine(left=True, bottom=True)
[x.title.set_size(12) for x in h.ravel()];
[x.yaxis.tick_left() for x in h.ravel()];
```

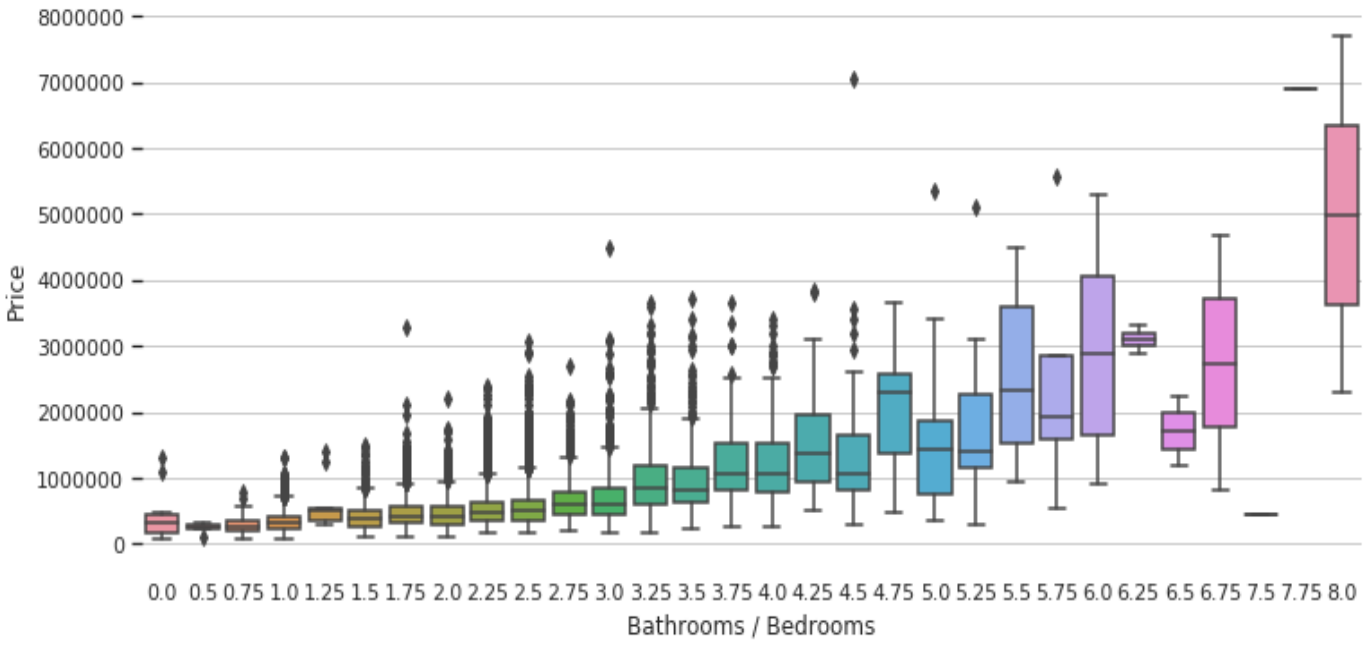
To determine bedrooms, floors or bathrooms/bedrooms vs price, I preferred **boxplot** because we have numerical data but they are not continuous as 1, 2... bedrooms, 2.5, 3... floors

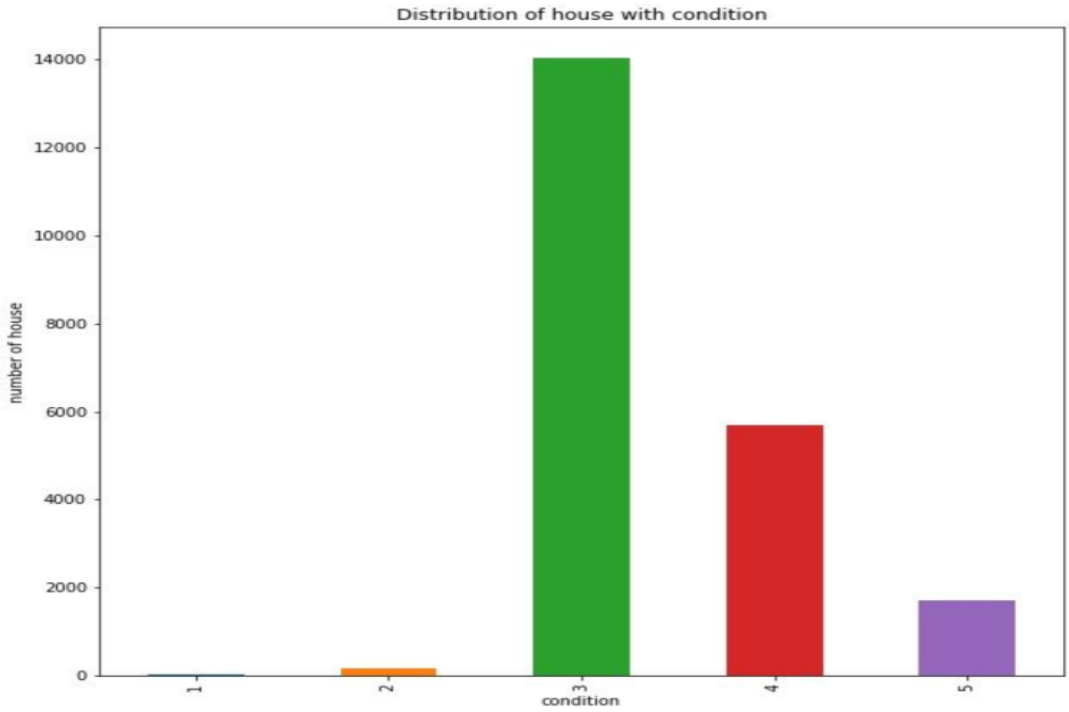



8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
Price
1
3
4
5
6
7
8
9
10
11
12
13
Grade

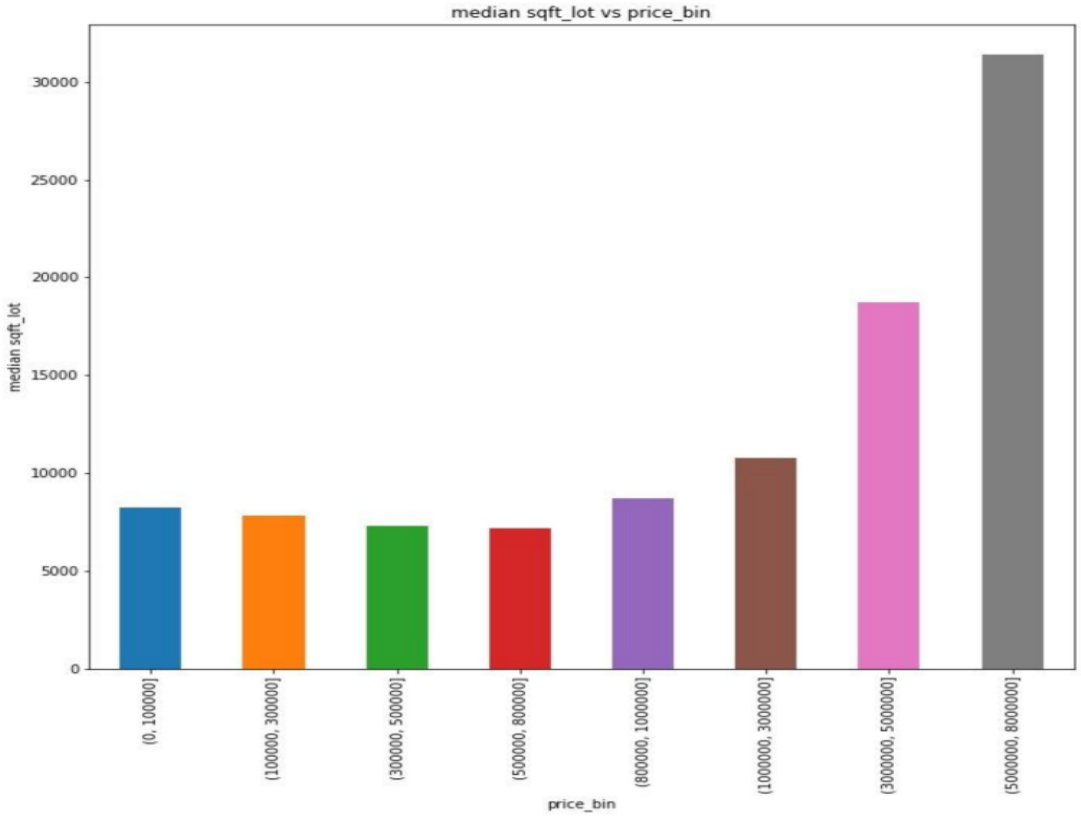

8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
Price
0
1
2
3
4
5
6
7
8
9
10
11
33
Bedrooms


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
Price
1.0
1.5
2.0
2.5
3.0
3.5
Floors





Grouping the house by condition tells us that most of the house has 3 points for a condition.

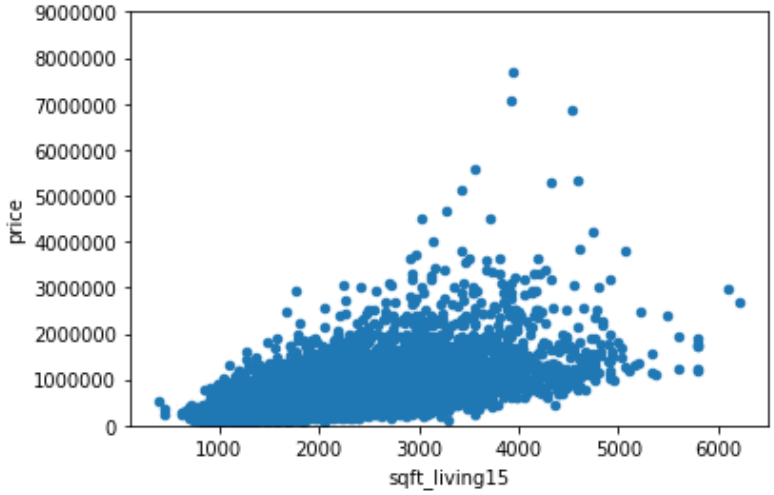


Sqft_lot vs price_bin plot shows that houses priced 0 to 500,000 have an inverse relation with median sqft_lot and houses priced 500,000 to 8,000,000 has a direct relationship with median sqft_lot.

PRICE VS. SQUARE FEET LIVING

The below scatterplots is indicative of house price increase with increase in square foot living and number of bathrooms. Home prices below 500,000 dollars have less than 2500 Square foot living area and have less than 3 bathrooms.

```
var = 'sqft_living15'
data = pd.concat([data['price'], data[var]], axis=1)
data.plot.scatter(x=var, y='price', ylim=(3,9000000));
```



Similarly, I conducted a hypothesis test to check the correlation between sqft_living and price. The p-value for the hypothesis test is less than the level of significance 0.05, so I reject the null hypothesis and suggest that there is a correlation between sqft_living and price. I also conducted a hypothesis test to check if there is a correlation between grade and price. The test suggests that there is correlation between grade and price. I also conducted a hypothesis test to check if there is no statistical importance between mean house price and a number of bedrooms less than 3 and greater than 3. The p-value for the test is greater than the level of significance 0.05, so I fail to reject the null hypothesis. This suggests us that there is no statistical importance between mean house price and a number of bedrooms less than 3 and greater than 3.

PREDICTIVE MODELING

There are different machine learning algorithms to predict the house prices. In this project I will use four regression algorithms to train machine learning models. The models I used are Linear Regression and Random Forest Regression, Gradient Boosting and Decision tree Regressor.

The motivations for choosing those regression algorithms are they can accurately predict the trends of feature data. There are many factors affect house prices, such as numbers of bedrooms and bathrooms. In addition, choosing different combinations of parameters in Linear Regression will also affect the predictions greatly.

SPLITTING DATA INTO TRAINING AND TESTING SET

The training dataset and test dataset must be similar, usually have the same predictors or variables. They differ on the observations and specific values in the variables. If you fit the model on the training dataset, then you implicitly minimize error or find correct responses. The fitted model provides a good prediction on the training dataset. Then you test the model on the test dataset. If the model predicts good also on the test dataset, you have more confidence. You have more confidence since the test dataset is similar to the training dataset, but not the same nor seen by the model. It means the model transfers prediction or learning in real sense.

So, by splitting dataset into training and testing subset, I can efficiently measure my trained model since it never sees testing data before. Thus it's possible to prevent overfitting.

I am just splitting dataset into 20% of test data and remaining 80% will used for training the model.

```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y
,test_size = 0.2)
```

FEATURE IMPORTANCE

After splitting let’s identify the most important fractures in our dataset that contribute most to my prediction variable. Irrelevant features would decrease accuracy especially for linear algorithms like linear regression.

Reduce Overfitting : Less redundant data means less opportunity to make decisions based on noise
Improve accuracy : Less misleading data means modeling accuracy improves
Reduce Training Time: Less data means that algorithms train faster

APPLYING MACHINE LEARNING MODEL

I. Linear Regression

Linear regression is a statistical approach for modeling relationship between a dependent variable with a given set of independent variables.

```
In [138]: from sklearn.linear_model import LinearRegression
          linear=LinearRegression()
          linear.fit(X_train, y_train)
          lin=linear.score(X_test,y_test)
          linpredict =linear.predict(X_test)
          exp_lin = explained_variance_score(linpredict,y_test)
```

II. Random Forest Regressor

Different kinds of models have different advantages. The random forest model is very good at handling tabular data with numerical features, or categorical features with fewer than hundreds of categories. Unlike linear models, random forests are able to capture non-linear interaction between the features and the target. Execute the following script to train the machine learning model using Random Forest Regressor.

```
In [125]: from sklearn.ensemble import RandomForestRegressor
          rand_regr = RandomForestRegressor(n_estimators=400,random_state=0)
          rand_regr.fit(X_train, y_train)

          random=rand_regr.score(X_test,y_test)
          predictions = rand_regr.predict(X_test)
          exp_rand = explained_variance_score(predictions,y_test)
```

III. Gradient Boosting Regressor

Gradient Boosting is one of the most powerful techniques for machine learning problems. It is an ensemble learning algorithm which combines the prediction of several base estimators in order to improve robustness over a single estimator.

```
In [140]: from sklearn.ensemble import GradientBoostingRegressor
          est=GradientBoostingRegressor(n_estimators=400, max_depth=5, loss='ls',
                                        min_samples_split=2,learning_rate=0.1).fit(X_train, y_train)
          gradient=est.score(X_test,y_test)
          pred = est.predict(X_test)
          exp_est = explained_variance_score(pred,y_test)
```

IV. Decision Tree Regressor

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete.

In [134]:
```
from sklearn.tree  import DecisionTreeRegressor
decision=DecisionTreeRegressor()
decision.fit(X_train, y_train)
decc=decision.score(X_test,y_test)
decpredict = decision.predict(X_test)
exp_dec = explained_variance_score(decpredict,y_test)
```

COMPARING MODELS

In [139]:
```
# Comparing Models on the basis of Model's Accuracy Score and Explained Variance Score of different models
models_cross = pd.DataFrame({
    'Model': ['Gradient Boosting','Linear Regression','Random Forest','Decision Tree'],
    'Score': [gradient,lin,random,decc],
     'Variance Score': [exp_est,exp_lin,exp_rand,exp_dec]})

models_cross.sort_values(by='Score', ascending=False)
```

Out[139]:

	Model	Score	Variance Score
0	Gradient Boosting	0.888406	0.878725
2	Random Forest	0.885575	0.868755
3	Decision Tree	0.769722	0.782054
1	Linear Regression	0.695223	0.524425

METRICS TO EVALUATE THE MODELS

I also used metrics to evaluate the performance of the model. The metrics used for evaluation of the models are root-mean-square error **(RMSE)**, mean squared error **(MSE)**, mean absolute error **(MAE)** and **R-squared** score.

The output looks like this:

According to the R-squared value, Gradient Boosting Regression model is the best performing model.

Model	Mean Absolute Error(MAE)	Mean Squared Error(MSE)	Root Mean Square Error(RMSE)	R-squared
Linear Regression	882888.95	13276038923.51	115221.69	0.63
Random Forest regression	54840.62	6302252436.45	79386.73	0.826
Gradient Boosting Regressor	56373.65	6228482868.33	78920.74	0.828
Decision TreeRegressor	76605.84	12025421297.16	109660.48	0.67

CONCLUTION

So, I have seen that **accuracy** of gradient boosting is around **88.84%** and also achieved decent **variance score** of **0.87** which is very close to 1. And all the metrics suggests that Gradient Boosting model has better performance. Therefore, it is inferred that **Gradient Boosting** is the suitable model for this dataset.

.