Import all needed Libraries

```
# Import Python Libraries: Pandas and Numpy
import pandas as pd
import numpy as np

# Import Libraries for data visualization
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
import seaborn as sns


# Import scikit-Learn module for the model: Linear Regression, Decision tree
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor


# Import scikit-Learn module to split the dataset into train/ test sub-datasets
from sklearn.model_selection import train_test_split

# score the model on the new test set
from sklearn import metrics
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')
```

Load the King County dataset into a dataframe

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

# Display first three records to check if correct data is loaded
data.head(5)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	f
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	

Pre-process the Dataset

```
# Check the datatypes of all Variables
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

We can see that date is in object format, so change it to date format

```
#Change the date format
data.date = pd.to_datetime(data.date)

# Now re-check the datatypes
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   id             21613 non-null  int64         
 1   date           21613 non-null  datetime64[ns]
 2   price          21613 non-null  float64
```

```
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: datetime64[ns](1), float64(5), int64(15)
memory usage: 3.5 MB
```

▼ Now we can see that the datatype of date has been changed to datetime

```
# Now check for null values in the dataset. If we have any null values we should replace them

data.isnull().sum()
```

```
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
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

We can see that there are no null values in the dataset.

Perform Exploratory Data Analysis on the Data

```
# Get the shape ( number of rows and columns) of the dataset

data.shape
```

```
(21613, 21)
```

We have a total of 21 columns and 21,613 records in the dataset

```
# Get the first five records/rows at the top of the data set
data.head(5)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	

```
# Get the summary statistics of the numeric variables/attributes of the data set

data.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04

```
# Sort data by target variable : price

data.sort_values("price")
```

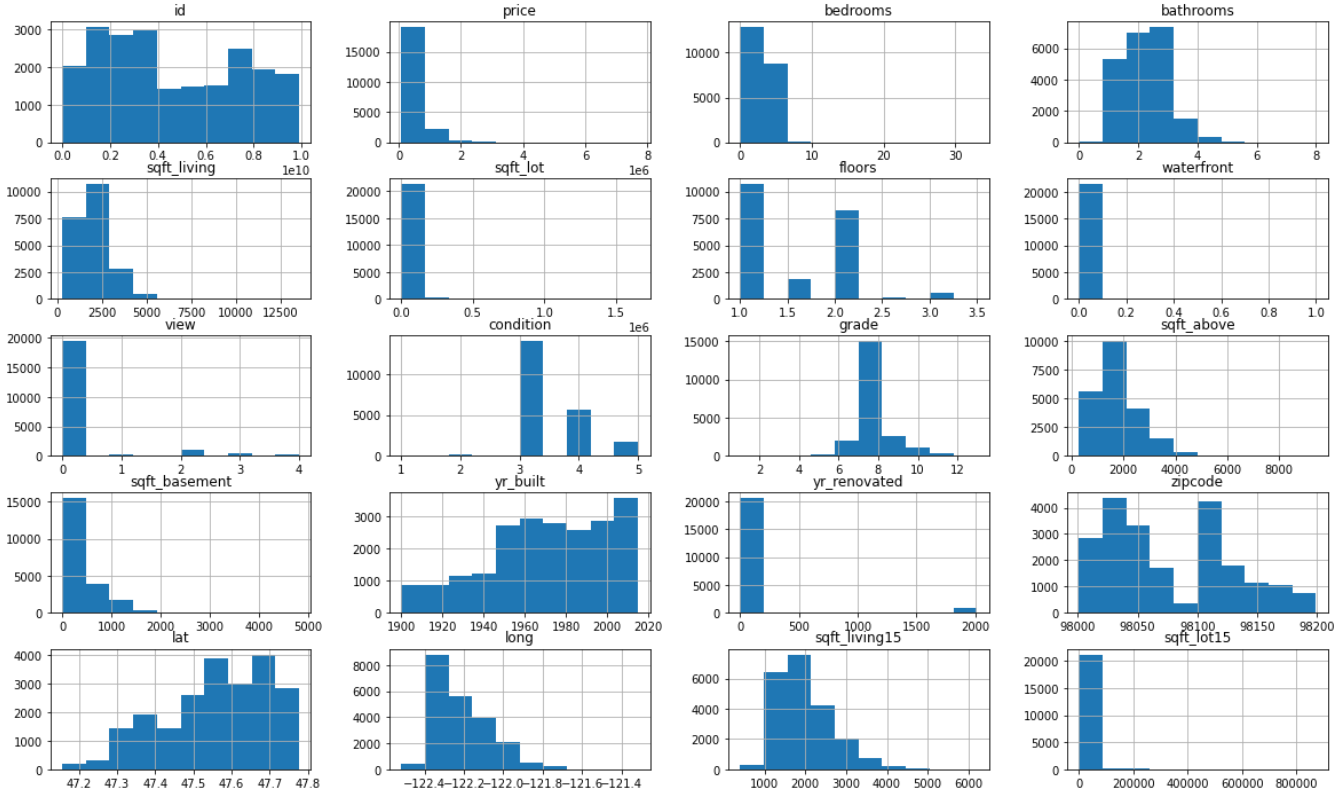
	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
1149	3421079032	2015-02-17	75000.0	1	0.00	670	43377	1.0
15293	40000362	2014-05-06	78000.0	2	1.00	780	16344	1.0
465	8658300340	2014-05-23	80000.0	1	0.75	430	5050	1.0
16198	3028200080	2015-03-24	81000.0	2	1.00	730	9975	1.0
8274	3883800011	2014-11-05	82000.0	3	1.00	860	10426	1.0
...	...	...	...	...	...	...	...	...
1448	8907500070	2015-04-13	5350000.0	5	5.00	8000	23985	2.0
4411	2470100110	2014-08-04	5570000.0	5	5.75	9200	35069	2.0
9254	9208900037	2014-09-19	6885000.0	6	7.75	9890	31374	2.0
3914	9808700762	2014-06-11	7062500.0	5	4.50	10040	37325	2.0
7252	6762700020	2014-10-13	7700000.0	6	8.00	12050	27600	2.5

21613 rows × 21 columns

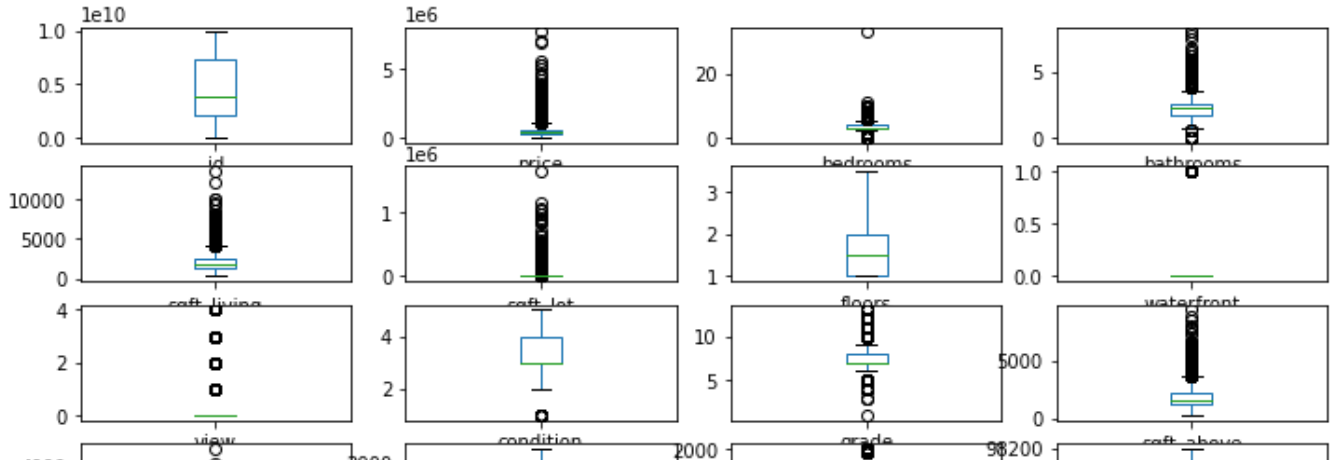
We can see the min price and max price of the houses i.e, 75,000 and 7,700,000

Data Visualization for the Dataset

```
# Plot histogram for each numeric
data.hist(figsize=(20,12))
plt.show()
```




```
# Create boxplots
data.plot(kind='box', subplots=True, layout=(6,4),sharex=False, figsize=(12,8))
plt.show()
```



We can see some major outliers in number of bedrooms and bathrooms. We can make an assumption that a residential house will have atleast 1 bedroom and 1 bathroom. And also we can see there are 33 bedrooms in one house, so may be that is a Motel/commercial property. Delete all such records from the datset

lat long sqft_living15 sqft_lot15

```
# Delete records with 0 bedrooms and bathrooms
data = data[data.bedrooms > 0]
data = data[data.bathrooms > 0]

# Delete the record with 33 bedrooms
data = data[data.bedrooms < 33]

# Shape of new dataset
data.shape
```

```
(21596, 21)
```

The datset has 21,596 records after removing the records. So we are still left with enough observations to develop a prediction model

Target Variable : Housing Price distribution

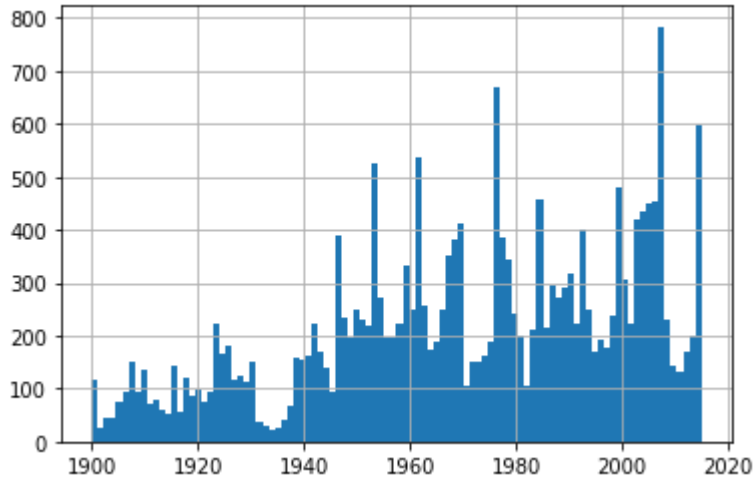
```
plt.figure(figsize = (8,6))
sns.distplot(data['price'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff16e8410>
```

We can see that most of the houses are within a million dollars, and there are very less number of houses that are more than 2 million

Relation between Price and other Categorical variables

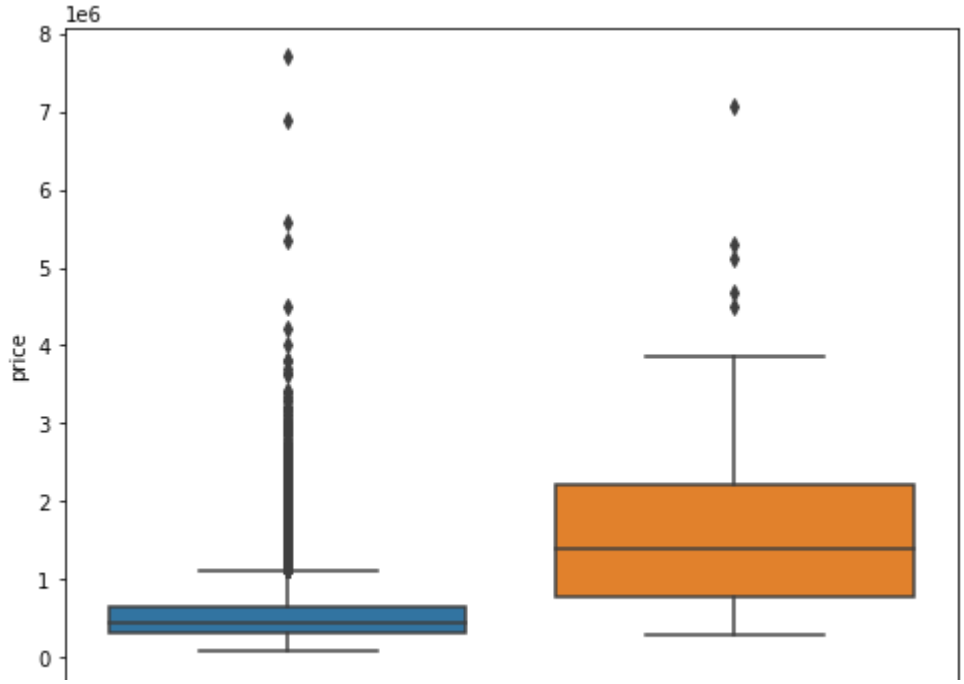

```
# Price of house and Year_built
data['yr_built'].hist(bins = 100)
plt.show()
```



We can see that the houses built after 2000 are more expensive, may be because of the realestate boom. And the dips in between are may be due to recession

```
# Price and Waterfront
plt.figure(figsize = (8,6))
sns.boxplot(x='waterfront', y='price', data=data)
```

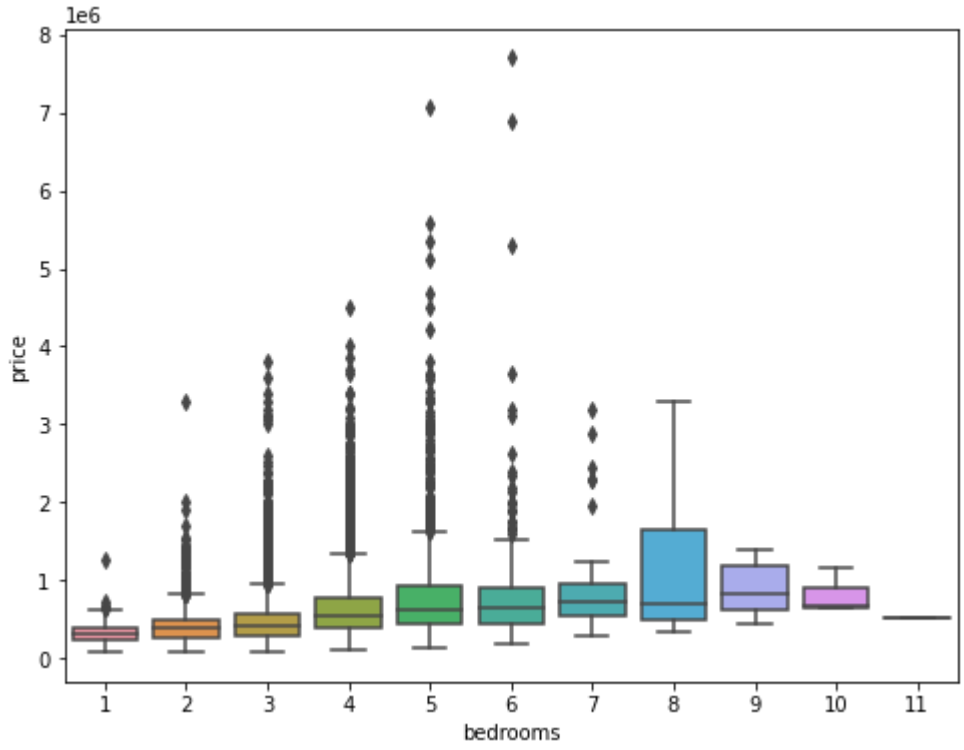
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff1681910>
```



Waterfront Property comes at a premium

```
# Price and Number of berdooms
plt.figure(figsize = (8,6))
sns.boxplot(x='bedrooms', y='price', data=data)
```

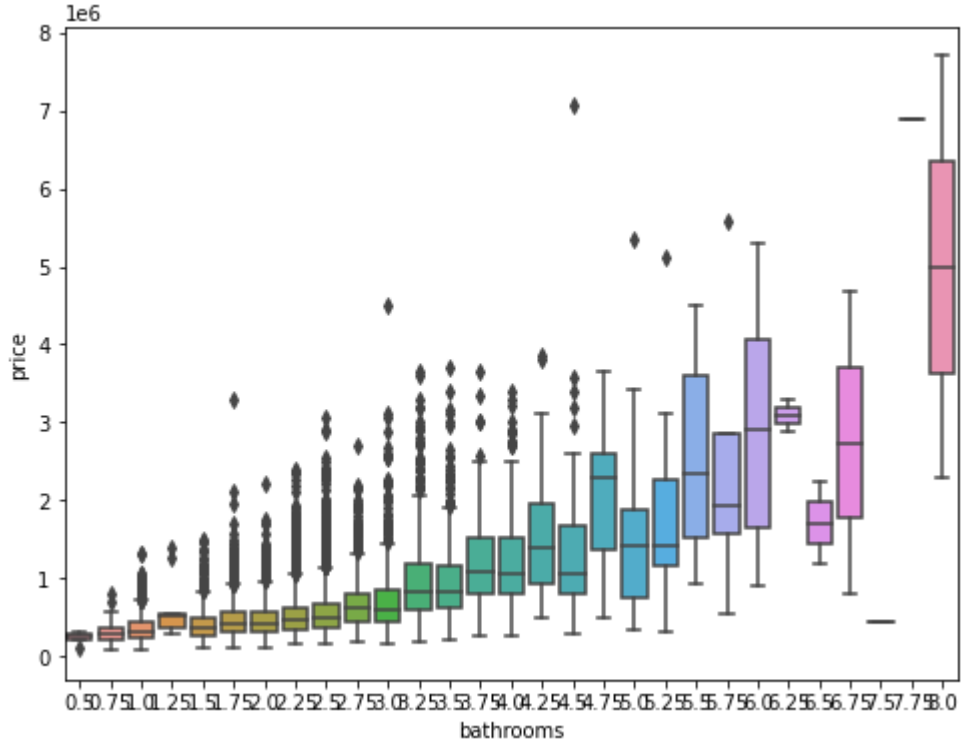
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff0a93e50>
```



We can see some increase in price with the increase in number of bedrooms

```
# Price and Number of bathrooms
plt.figure(figsize = (8,6))
sns.boxplot(x='bathrooms', y='price', data=data)
```

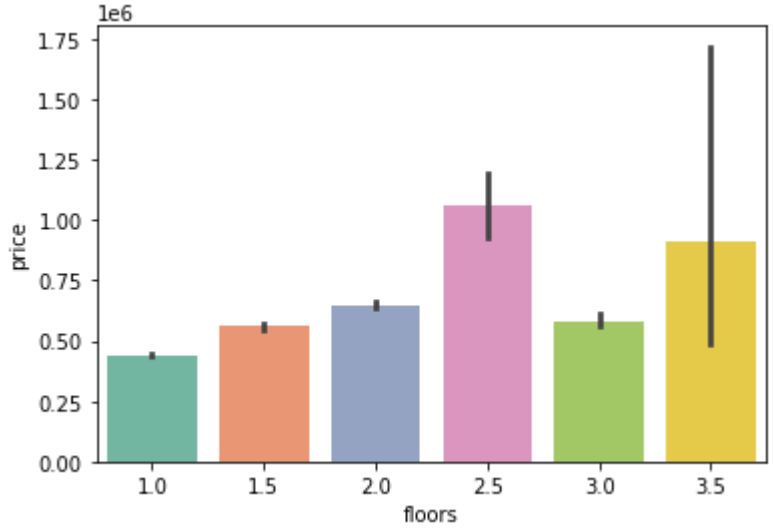
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff0f96450>
```



- We can see that with more number of bathrooms, the house price increases

```
# Price and Number of Floors
sns.barplot(x ='floors', y='price', data = data, palette= 'Set2' )
```

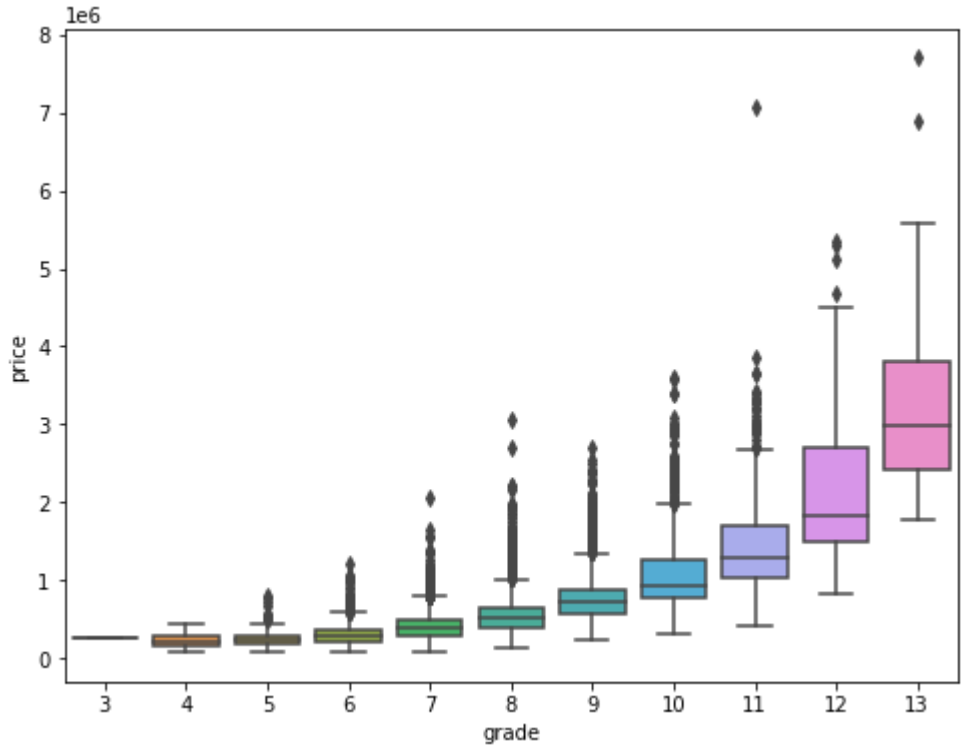
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff12532d0>
```



Houses with 2.5 floors are more expensive

```
# Price and Grade
plt.figure(figsize = (8,6))
sns.boxplot(x='grade', y='price', data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff145dd90>
```



We can see that with good construction quality/grade, the house price increases

```
# Price and Sqft_living area
plt.figure(figsize =(8,6))
sns.scatterplot(x ='sqft_living', y = 'price', data = data)
```

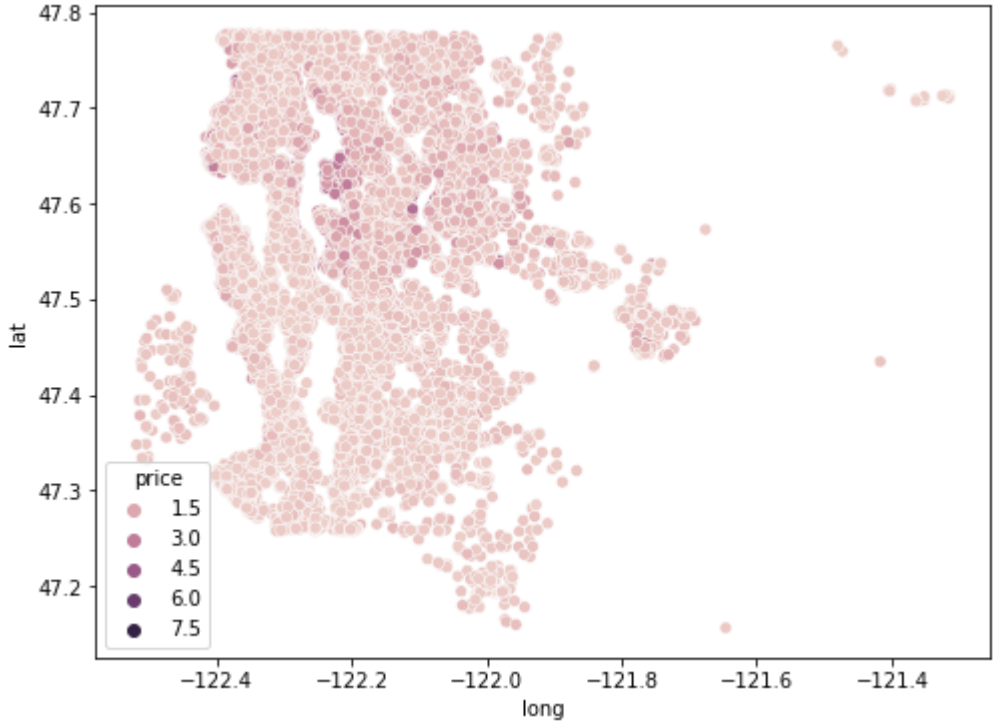
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff0a53910>
```

Price increases with increse in size of living area. We can see some outliers. For example, there is one house with living area of almost 14,000 sqft for less price (220,000). This may be because the property is in the country area ad not in the city

Location of the property

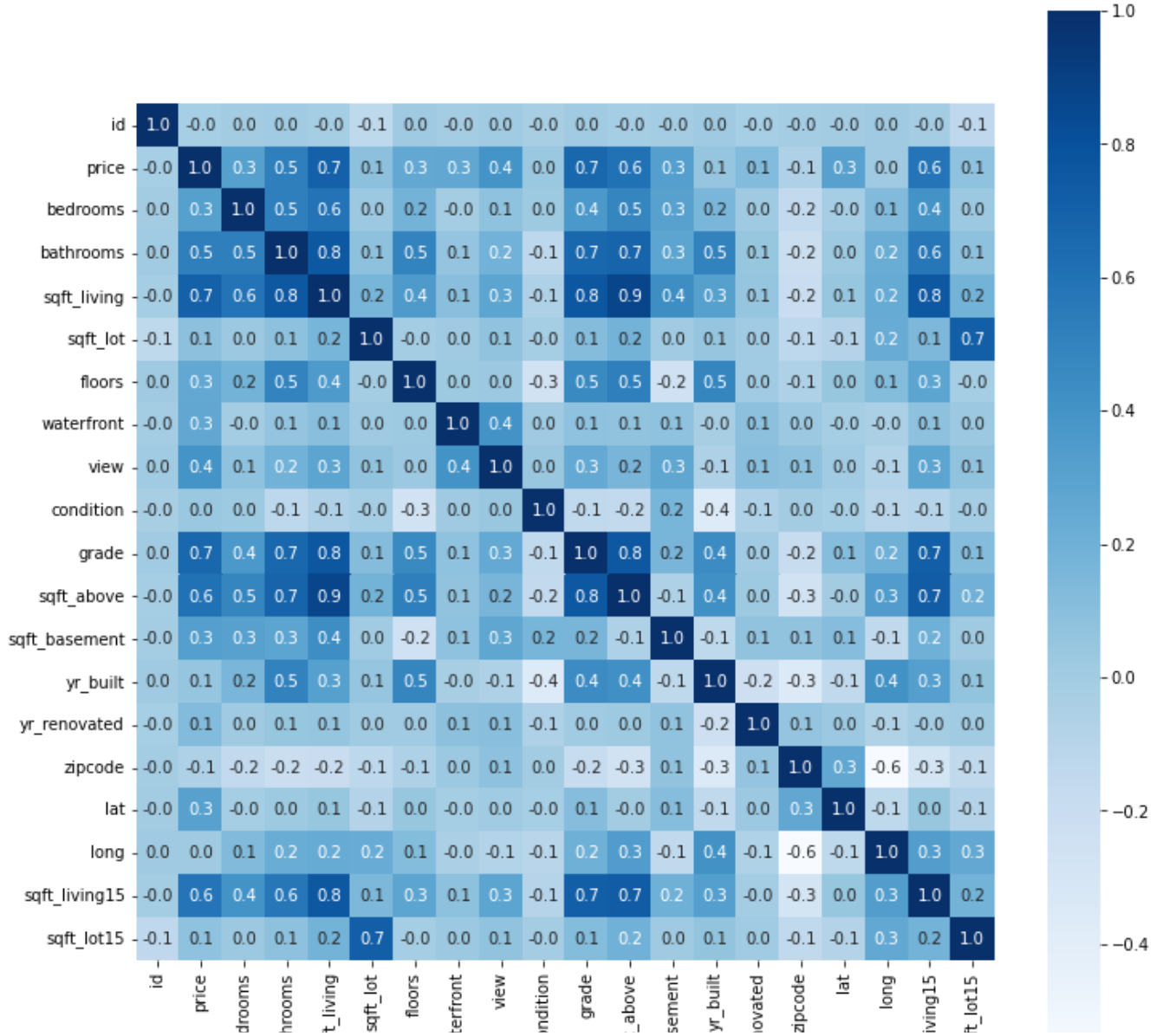
```
plt.figure(figsize =(8,6))
sns.scatterplot(x ='long', y = 'lat', data = data, hue ='price')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff10e2fd0>
```




```
# Correlation matrix
corr  = data.corr()
plt.figure(figsize = (12,12))
sns.heatmap(corr, cbar= True, square = True, fmt = '.1f', annot = True, annot_kws = {'size':1
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff12856d0>
```



Feature Engineering : Creating new Features

1.Split the Date sold column into month_sold and year_sold. This way we can understand if there are more sales in any particular month of the year

```
# Convert the Date column into month_sold and year_sold.

data['month_sold']=data['date'].apply(lambda x:x.month)
data['year_sold']= data['date'].apply(lambda x:x.year)


#Drop original Date column
data.drop(columns=['date'], axis=1, inplace=True)
```

2. Convert the Year built column into 'Age' column that allows better interpretation. For this, we subtracted the actual year built from the latest built year in the dataset('2015') to calculate age of the house.

```
# calculate age of the house as of 2015
data['age'] = 2015 - data.yr_built

# Drp the original column 'yr_built'
data = data.drop(columns=['yr_built'], axis=1)
```

3. Change the year renovated column into a binary column - '1' for the homes that are renovated within past 10 years or built within the past 5 years(so they dont need renovation yet), and '0' for homes that are not renovated within past 10 years

```
#Fill missing values
data.yr_renovated.fillna(0.0, inplace=True)

#Create renovated column
data['renovated'] = data.year_sold - data.yr_renovated

#Replace any values less than 10 with 1, and any values over 10 with 0
renovated = data.renovated.values
age = data.age.values
values = np.where(renovated <= 10, 1, 0)
data['renovated'] = np.where(age <= 5, 1, values)

#Drop yr_renovated column
data.drop(columns=['yr_renovated'], axis=1, inplace=True)
```

4. Seattle is an important city in King county, where a lot of houses are listed. So for better analysis, calculate distance of the house from specific latitude and longitude points (Seattle downtown). This can be done using geopy module in python

```
from geopy import distance
lat_long = data['lat'].astype(str) + ',' + data['long'].astype(str)
lat_long = list(map(eval, lat_long))

Seattle = (47.6062, -122.3321)
miles = []
for i in lat_long:
    miles.append(round(distance.distance(i, Seattle).miles, 1))

data['distance'] = miles
```

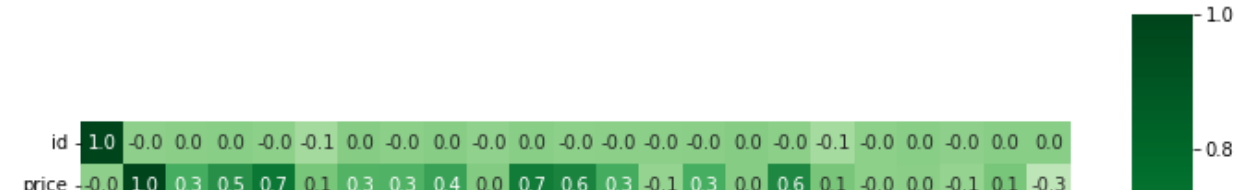
```
# After adding new features, we can see all the important columns
data.columns
```

```
Index(['id', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'zipcode', 'lat', 'long', 'sqft_living15',
       'sqft_lot15', 'month_sold', 'year_sold', 'age', 'renovated',
       'distance'],
      dtype='object')
```

Correlation matrix

```
# Correlation matrix
corr  = data.corr()
plt.figure(figsize = (12,12))
sns.heatmap(corr, cbar= True, square = True, fmt = '.1f', annot = True, annot_kws = {'size':1
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcfe38ba390>
```



Feature Selection

By looking at the correlation matrix we can understand which features are important and influence the house price

```
# Drop all unnecessary features
dCol = ['id','zipcode','month_sold','year_sold','lat','long','condition']
data.drop(dCol, axis = 1, inplace = True)

data.columns
```

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'grade', 'sqft_above', 'sqft_basement',
       'sqft_living15', 'sqft_lot15', 'age', 'renovated', 'distance'],
      dtype='object')
```

```
# Now we have 16 features
data.shape
```

```
(21596, 16)
```

Separate the dataset into Input and Output Arrays

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

Split Input/Output Arrays into Training/Testing Datasets

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.3, random_state=20)
```

Build and Train the model

```
model = LinearRegression()
#model = DecisionTreeRegressor()
model.fit(X_train,Y_train)
```

```
LinearRegression()
```

Model Prediction on Trained data

```
Y_predict = model.predict(X_train)
```

Model Evaluation

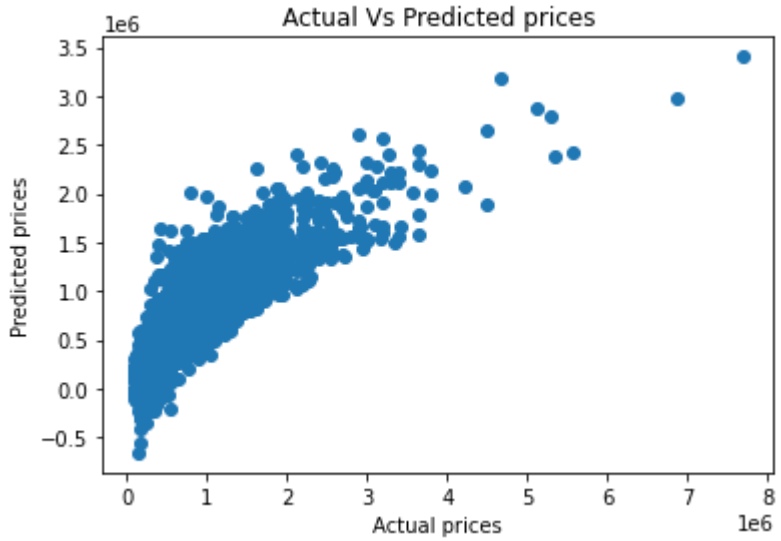
```
# Model Evaluation

print('R^2:',metrics.r2_score(Y_train,Y_predict))
print('MAE:', metrics.mean_absolute_error(Y_train,Y_predict))
print('MSE:', metrics.mean_squared_error(Y_train,Y_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(Y_train,Y_predict)))
```

```
R^2: 0.7085481351384417
MAE: 127748.50571489001
MSE: 40239112611.98932
RMSE: 200596.8908333061
```

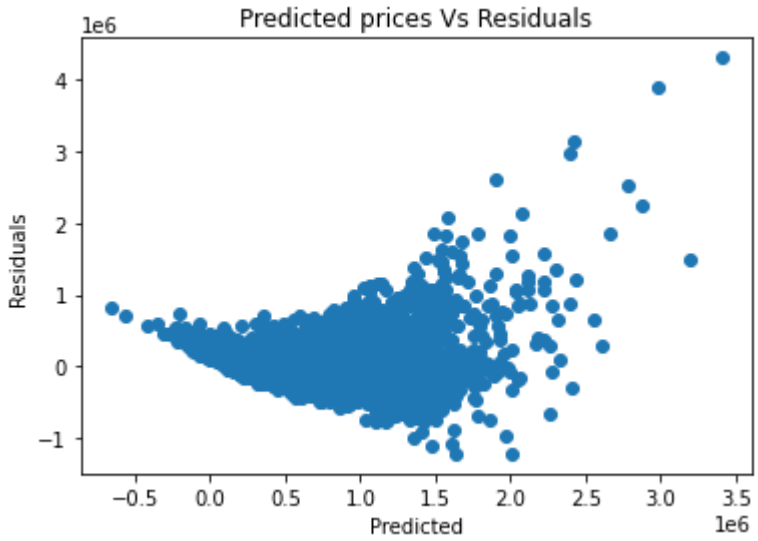
Visualizing the differences between actual and predicted values

```
plt.scatter(Y_train,Y_predict)
plt.xlabel("Actual prices")
plt.ylabel("Predicted prices")
plt.title("Actual Vs Predicted prices")
plt.show()
```



Inspecting Residuals

```
# Checking residuals
plt.scatter(Y_predict, Y_train-Y_predict)
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Predicted prices Vs Residuals")
plt.show()
```



Checking Normality of Errors

```
sns.distplot(Y_train-Y_predict)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Histogram of Residuals")
plt.show()
```

Model Evaluation

```
# Predicting test data with model
Y_test_predict = model.predict(X_test)
```

```
# Model Evaluation
act_model = metrics.r2_score(Y_test, Y_test_predict)

print('R^2:', act_model)
print('MAE:', metrics.mean_absolute_error(Y_test, Y_test_predict))
print('MSE:', metrics.mean_squared_error(Y_test, Y_test_predict))
print('RMSE:', np.sqrt(metrics.mean_squared_error(Y_test, Y_test_predict)))
```

```
R^2: 0.7111081828873063
MAE: 124755.49161090353
MSE: 36728240445.76218
RMSE: 191646.13339632546
```

Let's make a Prediction using the Linear model

```
# Predict the house price in dollars
# Arbitrary predictors/housing records are used
model.predict([[4,3,3400,7000,2,1,4,3,3000,400,2800,6000,5,0,5.5]])
```

```
array([1196974.9103683])
```

The Predicted value of the house with the above features is $1,196,975

✓ 0s completed at 8:25 PM