House Price Prediction

November 24, 2019

1 Problem Statement

Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

- You can use sklearn, numpy pandas, matplotlib
- Apply linear regression, random forest regressor
- Steps include
  - Read the data using pandas datafram
  - Do feature engineering and separate target value price
  - Remove those features which are not contributing
  - Apply train test split on given data
  - Apply ML algorithm using sklearn on training data and do prediction of test set
  - Evaluation matric should be RMSE

Follow the Data Science pipeline and build a Machine Learning model that will predict House Prices based on given features. The pipeline is as follows:

1. Data Wrangling and Preprocessing
2. Exploratory Data Analysis
3. Feature Selection
4. Model Training
5. Testing and Optimization

Bonus part: Do detail Exploratory data analysis to get good features.

```
In [17]: # If you want to install any missing packages, then uncomment the lines given below a
         # to ensure that you have all the dependencies you need to run the notebook.
         #import sys
         #!{sys.executable} -m pip install xgboost
         from scipy import stats
```

```
In [2]: # Libraries
        import pandas as pd
        import numpy as np
        import missingno as mno
        import seaborn as sns
        import matplotlib.pyplot as plt
```

1.1 Data Description

The first thing you need to do before solving any Data Science problem is getting familiar with the dataset. Get to know your data by printing out some stats, checking its dimensions and checking data types of features.

In [4]:
```
# Load training data
data = pd.read_csv('house_data.csv')
data.head()
```

Out[4]:
```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0         3       1.00         1180
1  6414100192  20141209T000000  538000.0         3       2.25         2570
2  5631500400  20150225T000000  180000.0         2       1.00          770
3  2487200875  20141209T000000  604000.0         4       3.00         1960
4  1954400510  20150218T000000  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view     ...      grade  sqft_above  \
0      5650     1.0           0     0     ...          7        1180
1      7242     2.0           0     0     ...          7        2170
2     10000     1.0           0     0     ...          6         770
3      5000     1.0           0     0     ...          7        1050
4      8080     1.0           0     0     ...          8        1680

   sqft_basement  yr_built  yr_renovated  zipcode      lat     long  \
0              0      1955             0    98178  47.5112 -122.257
1            400      1951          1991    98125  47.7210 -122.319
2              0      1933             0    98028  47.7379 -122.233
3            910      1965             0    98136  47.5208 -122.393
4              0      1987             0    98074  47.6168 -122.045

   sqft_living15  sqft_lot15
0           1340        5650
1           1690        7639
2           2720        8062
3           1360        5000
4           1800        7503

[5 rows x 21 columns]
```

In [5]:
```
# Dimensions of training data
data.shape
```

Out[5]:
```
(21613, 21)
```

In [6]:
```
# Explore columns
data.columns
```

Out[6]:
```
Index(['id', 'date', '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'],
      dtype='object')
```

In [7]:
```python
# Description
data.describe()
```

Out[7]:

```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.161300e+04  2.161300e+04  21613.000000  21613.000000  21613.000000
mean   4.580302e+09  5.400881e+05      3.370842      2.114757   2079.899736
std    2.876566e+09  3.671272e+05      0.930062      0.770163    918.440897
min    1.000102e+06  7.500000e+04      0.000000      0.000000    290.000000
25%    2.123049e+09  3.219500e+05      3.000000      1.750000   1427.000000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1910.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000  13540.000000

           sqft_lot        floors    waterfront          view     condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  21613.000000
mean   1.510697e+04      1.494309      0.007542      0.234303      3.409430
std    4.142051e+04      0.539989      0.086517      0.766318      0.650743
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000
50%    7.618000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.068800e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000  21613.000000
mean       7.656873   1788.390691     291.509045   1971.005136     84.402258
std        1.175459    828.090978     442.575043     29.373411    401.679240
min        1.000000    290.000000       0.000000   1900.000000      0.000000
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000

            zipcode           lat          long  sqft_living15     sqft_lot15
count  21613.000000  21613.000000  21613.000000   21613.000000   21613.000000
mean   98077.939805     47.560053   -122.213896    1986.552492   12768.455652
std       53.505026      0.138564      0.140828     685.391304   27304.179631
min    98001.000000     47.155900   -122.519000     399.000000     651.000000
25%    98033.000000     47.471000   -122.328000    1490.000000    5100.000000
50%    98065.000000     47.571800   -122.230000    1840.000000    7620.000000
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000
```

In [8]:
```python
# Check Datatypes
data.dtypes
```

```
Out[8]: id                 int64
        date              object
        price            float64
        bedrooms           int64
        bathrooms        float64
        sqft_living        int64
        sqft_lot           int64
        floors           float64
        waterfront         int64
        view               int64
        condition          int64
        grade              int64
        sqft_above         int64
        sqft_basement      int64
        yr_built           int64
        yr_renovated       int64
        zipcode            int64
        lat              float64
        long             float64
        sqft_living15      int64
        sqft_lot15         int64
        dtype: object
```

1.2 Data Wrangling and Preprocessing

We have to preprocess our data in order to make it useful for data analysis and model training. Although, the steps involved vary depending on the problem and the dataset but here we have provided a roughly generic approach which is applicable for most problems. The steps involved are as follows:

1. Look for Null or Missing Values
2. Change data type of features, if required
3. Encode data of categorical features
4. Deal with Null or Missing values

```
In [9]: # Check for any null or missing values
        data.isnull().values.any()
```

```
Out[9]: False
```

```
In [10]: # Check missing values in each column of training data
         data.isnull().sum()
```

```
Out[10]: 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
```

Thus we can see that there are no missing values in the data.

In [14]:
```
# Convert object type to date type
data['date'] = pd.to_datetime(data['date'])
data.head()
```

Out[14]:
```
           id       date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520 2014-10-13  221900.0         3       1.00         1180
1  6414100192 2014-12-09  538000.0         3       2.25         2570
2  5631500400 2015-02-25  180000.0         2       1.00          770
3  2487200875 2014-12-09  604000.0         4       3.00         1960
4  1954400510 2015-02-18  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view     ...      grade  sqft_above  \
0      5650     1.0           0     0     ...          7        1180
1      7242     2.0           0     0     ...          7        2170
2     10000     1.0           0     0     ...          6         770
3      5000     1.0           0     0     ...          7        1050
4      8080     1.0           0     0     ...          8        1680

   sqft_basement  yr_built  yr_renovated  zipcode      lat     long  \
0              0      1955             0    98178  47.5112 -122.257
1            400      1951          1991    98125  47.7210 -122.319
2              0      1933             0    98028  47.7379 -122.233
3            910      1965             0    98136  47.5208 -122.393
4              0      1987             0    98074  47.6168 -122.045

   sqft_living15  sqft_lot15
0           1340        5650
1           1690        7639
2           2720        8062
```

```
3             1360          5000
4             1800          7503

[5 rows x 21 columns]
```

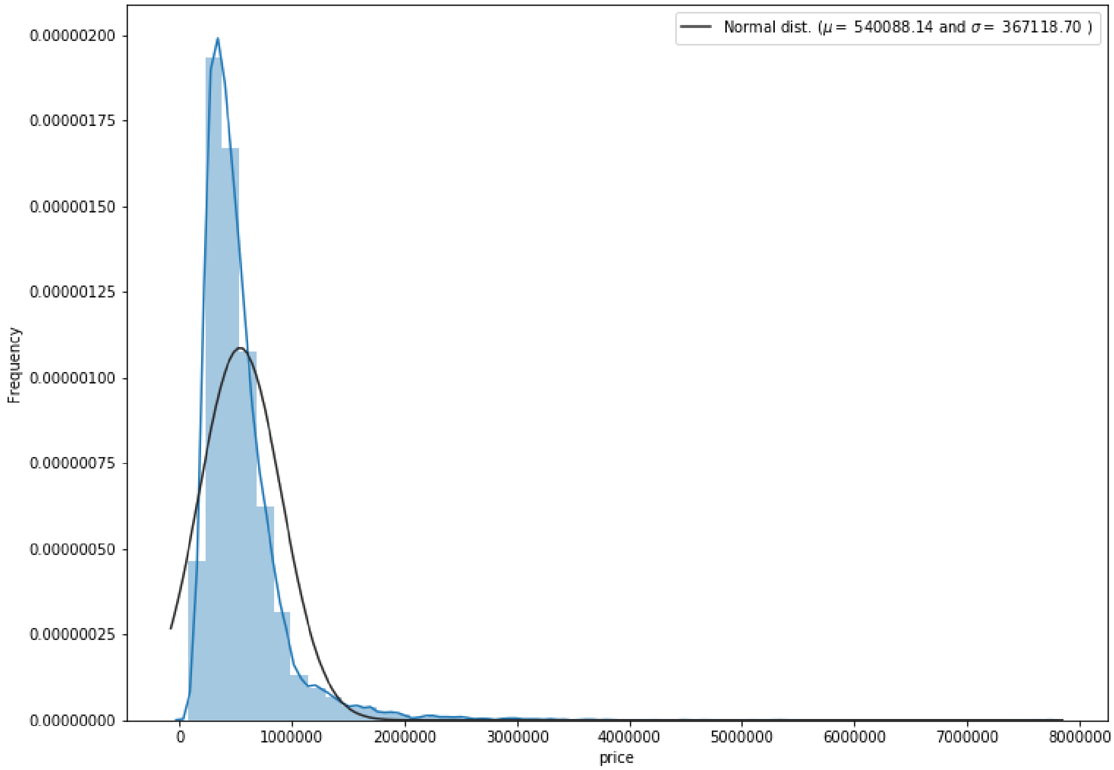
1.3 Data Analysis and Visualizations

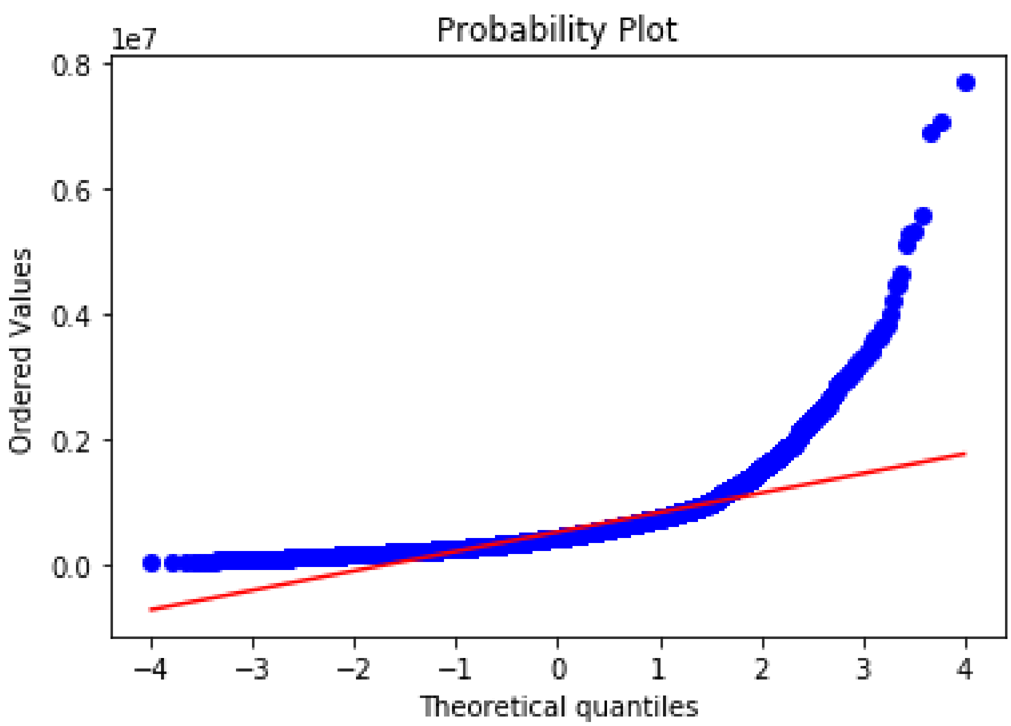
Performing a detailed analysis of the data helps you understand which features are important, what's their correlation with each other which features would contribute in predicting the target variable. Different types of visualizations and plots can help you acheive that. These include:

1. Bar Plots
2. Joint Plots
3. Box Plots
4. Correlation Heatmap
5. Distribution Plot
6. PCA Bi-plot

The 'Target Variable' is this data is the price column. We will now perform some analysis on the target variable to get a better insight into what we are working on.

In [18]:
```
plt.subplots(figsize=(12,9))
sns.distplot(data['price'], fit=stats.norm)

# Get the fitted parameters used by the function

(mu, sigma) = stats.norm.fit(data['price'])

# plot with the distribution

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], l
plt.ylabel('Frequency')

#Probablity plot

fig = plt.figure()
stats.probplot(data['price'], plot=plt)
plt.show()
```

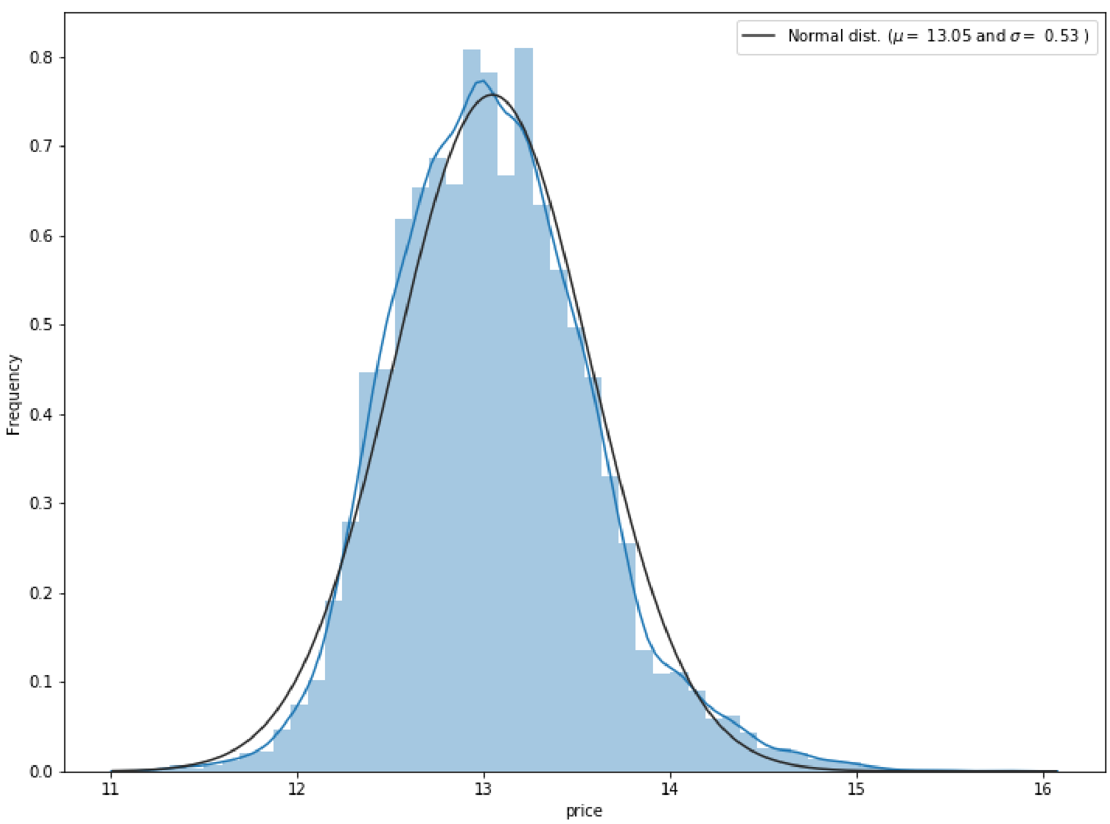

Normal dist. (μ = 540088.14 and σ = 367118.70)
Frequency
price
0.00000200
0.00000175
0.00000150
0.00000125
0.00000100
0.00000075
0.00000050
0.00000025
0.00000000
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000

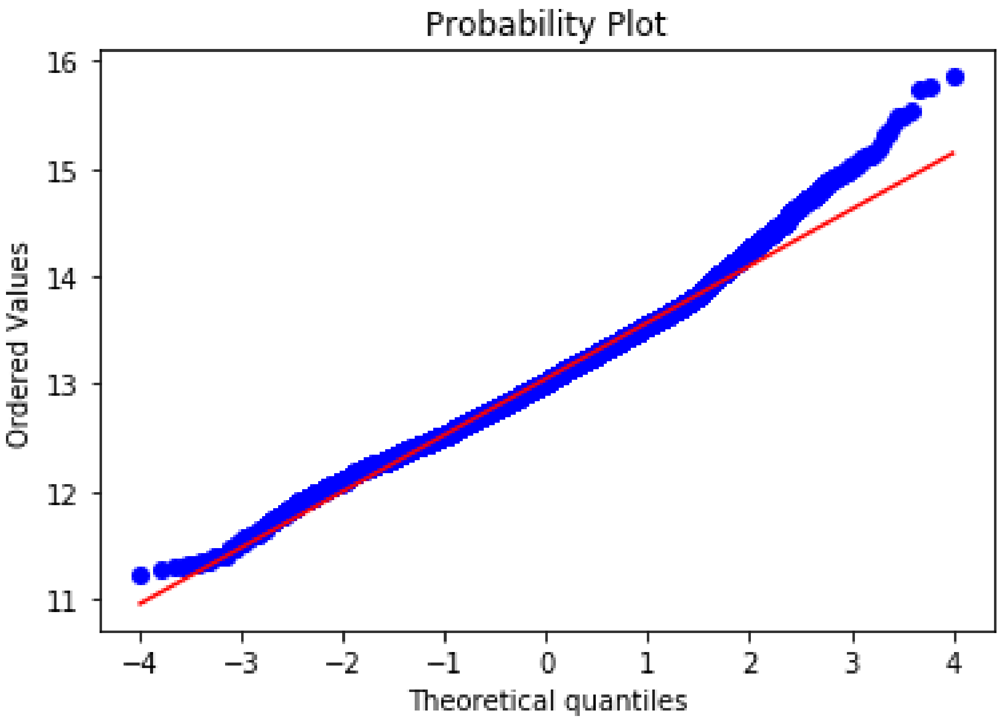

Probability Plot
1e7
Ordered Values
Theoretical quantiles
0.8
0.6
0.4
0.2
0.0
−4
−3
−2
−1
0
1
2
3
4

This target varibale is right skewed. Now, we need to tranform this variable and make it normal distribution.

Here we use log for target variable to make more normal distribution

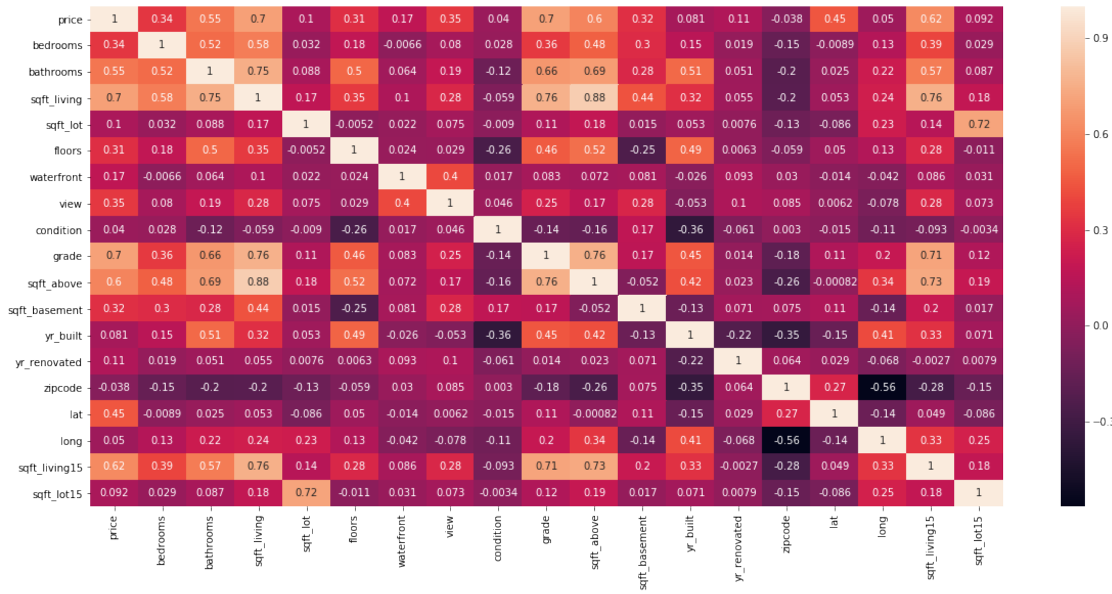
In [19]:
```
#we use log function which is in numpy
data['price'] = np.log1p(data['price'])

#Check again for more normal distribution
plt.subplots(figsize=(12,9))
sns.distplot(data['price'], fit=stats.norm)

# Get the fitted parameters used by the function
(mu, sigma) = stats.norm.fit(data['price'])

# plot with the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], 
plt.ylabel('Frequency')

#Probablity plot
fig = plt.figure()
stats.probplot(data['price'], plot=plt)
plt.show()
```





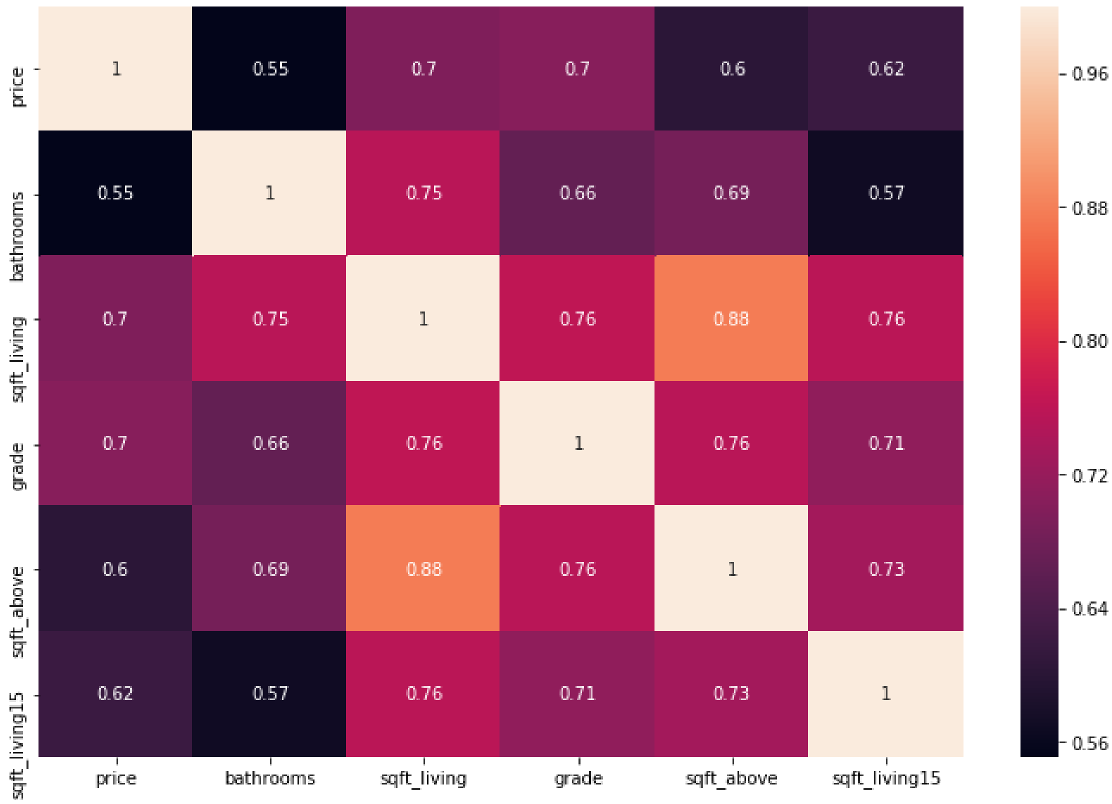
Correlation between train attributes

```
In [20]: # Separate variable into new dataframe from original dataframe which has only numerica
         # There are 20 variables of numerical type
         train_corr = data.select_dtypes(include=[np.number])

In [21]: train_corr.shape

Out[21]: (21613, 20)

In [22]: # Delete Id because that is not need for correlation plot
         del train_corr['id']

In [23]: # Correlation plot
         corr = train_corr.corr()
         plt.subplots(figsize=(20,9))
         sns.heatmap(corr, annot=True)

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1c85d41e4a8>
```



Top 50% Correlation train attributes with price

```
In [24]: top_feature = corr.index[abs(corr['price']>0.5)]
         plt.subplots(figsize=(12, 8))
         top_corr = data[top_feature].corr()
         sns.heatmap(top_corr, annot=True)
         plt.show()
```



In [34]:
```
print("Find most important features relative to target")
corr = data.corr()
corr.sort_values(['price'], ascending=False, inplace=True)
corr.price
```

```
Find most important features relative to target
```

Out[34]:
```
price            1.000000
grade            0.703634
sqft_living      0.695341
sqft_living15    0.619312
sqft_above       0.601802
bathrooms        0.550802
lat              0.449174
view             0.346522
bedrooms         0.343561
sqft_basement    0.316970
floors           0.310558
waterfront       0.174586
yr_renovated     0.114498
sqft_lot         0.099622
```

```
sqft_lot15       0.091592
yr_built         0.080654
long             0.049942
condition        0.039558
id              -0.003819
zipcode         -0.038306
Name: price, dtype: float64
```

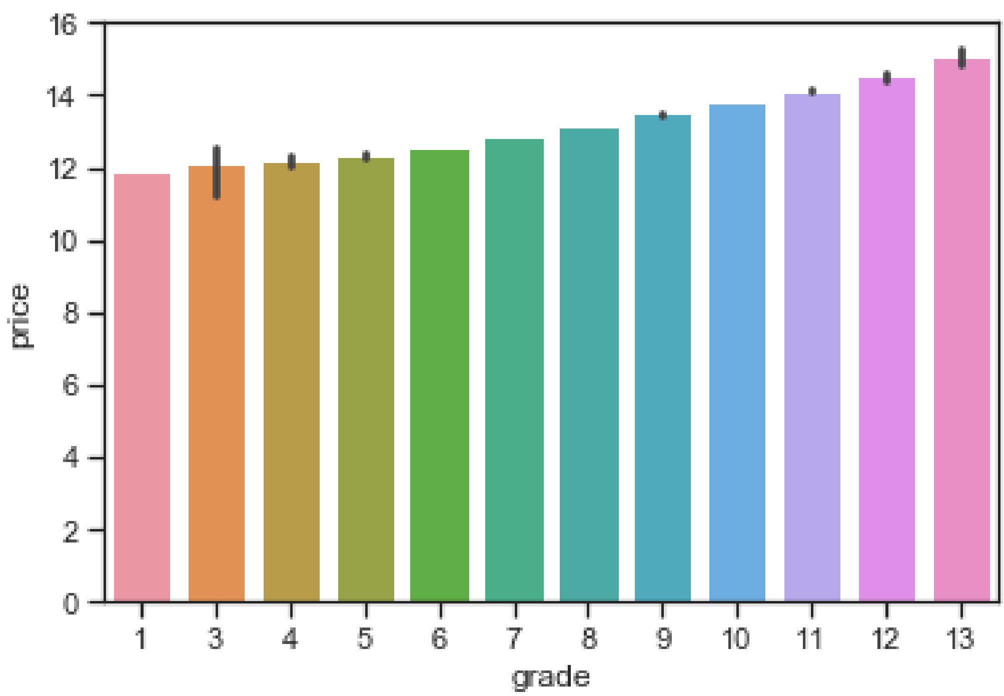
Here grade is highly correlated with target feature of price by 71%

In [30]:
```python
#unique value of sqft_living
data.grade.unique()
```

Out[30]:
```
array([ 7,  6,  8, 11,  9,  5, 10, 12,  4,  3, 13,  1], dtype=int64)
```

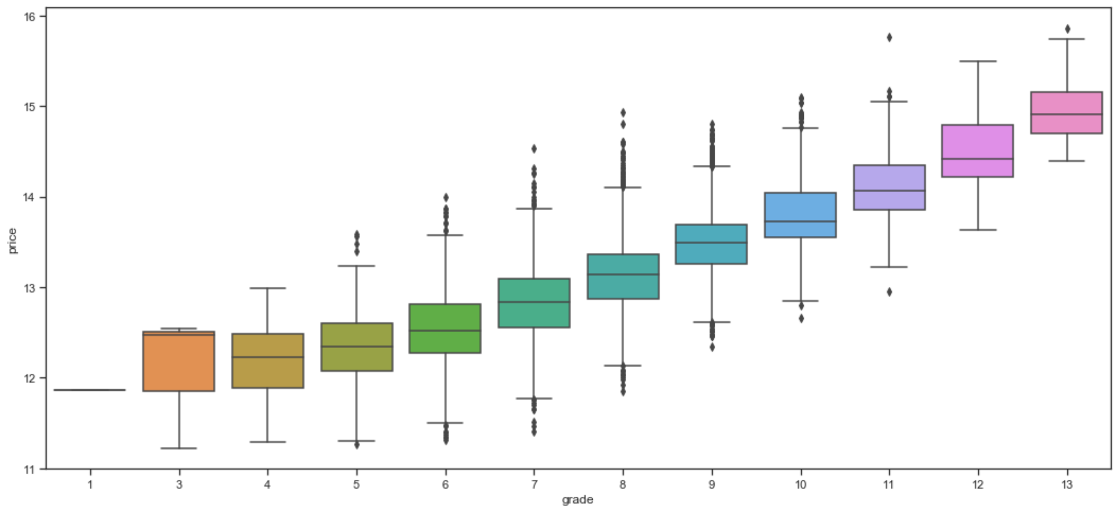
In [31]:
```python
sns.barplot(data.grade, data.price)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c86d6eee80>
```



In [32]:
```python
#boxplot
plt.figure(figsize=(18, 8))
sns.boxplot(x=data.grade, y=data.price)
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c86d615b00>
```

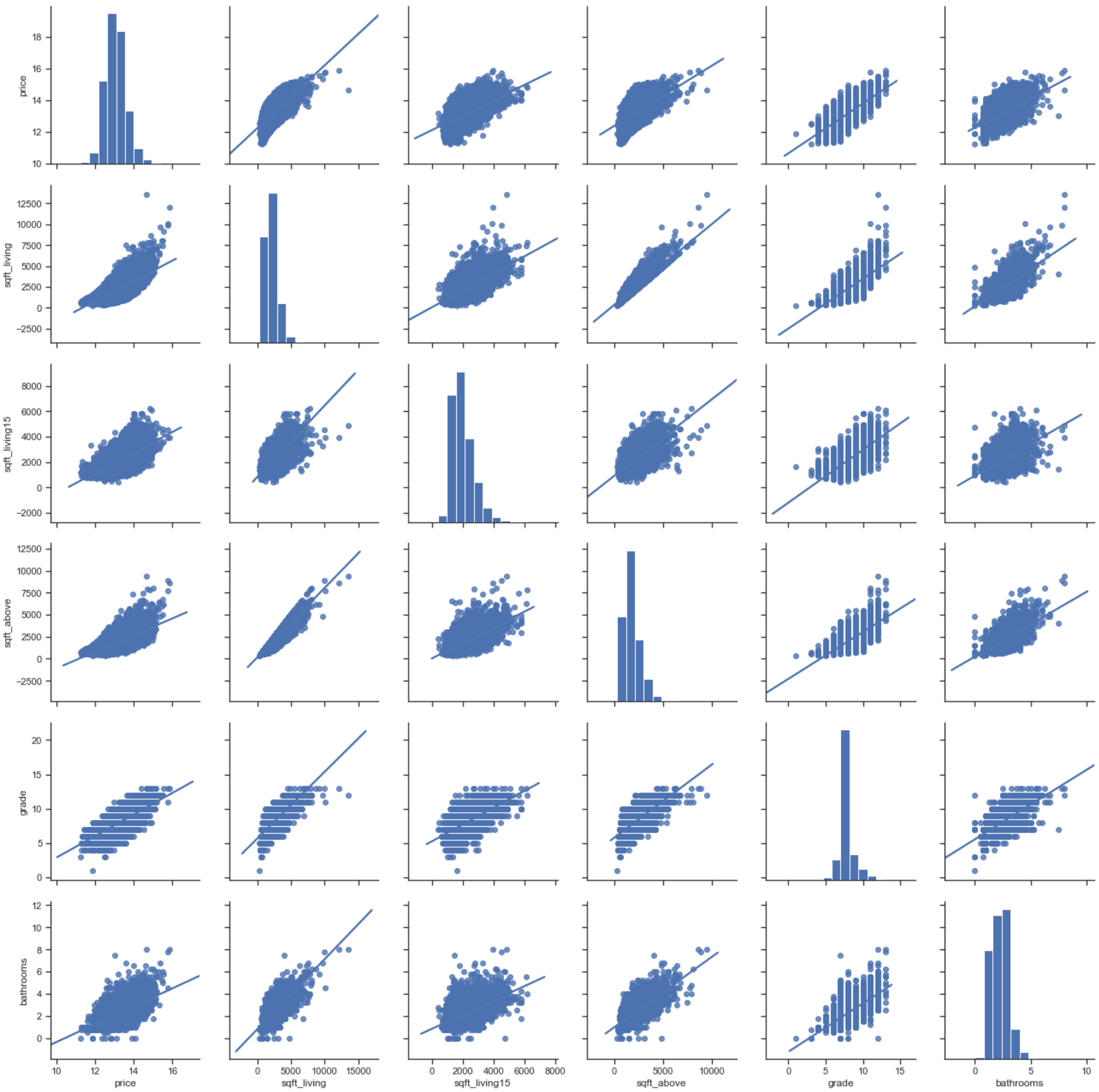


Evaluation of the top_features which are contributing to the price of the house.

```
In [28]: col = ['price', 'sqft_living', 'sqft_living15', 'sqft_above', 'grade', 'bathrooms']
         sns.set(style='ticks')
         sns.pairplot(data[col], size=3, kind='reg')
```

```
C:\Users\alina\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2065: UserWarning: The `size` pa
  warnings.warn(msg, UserWarning)
```

```
Out[28]: <seaborn.axisgrid.PairGrid at 0x1c85e039588>
```

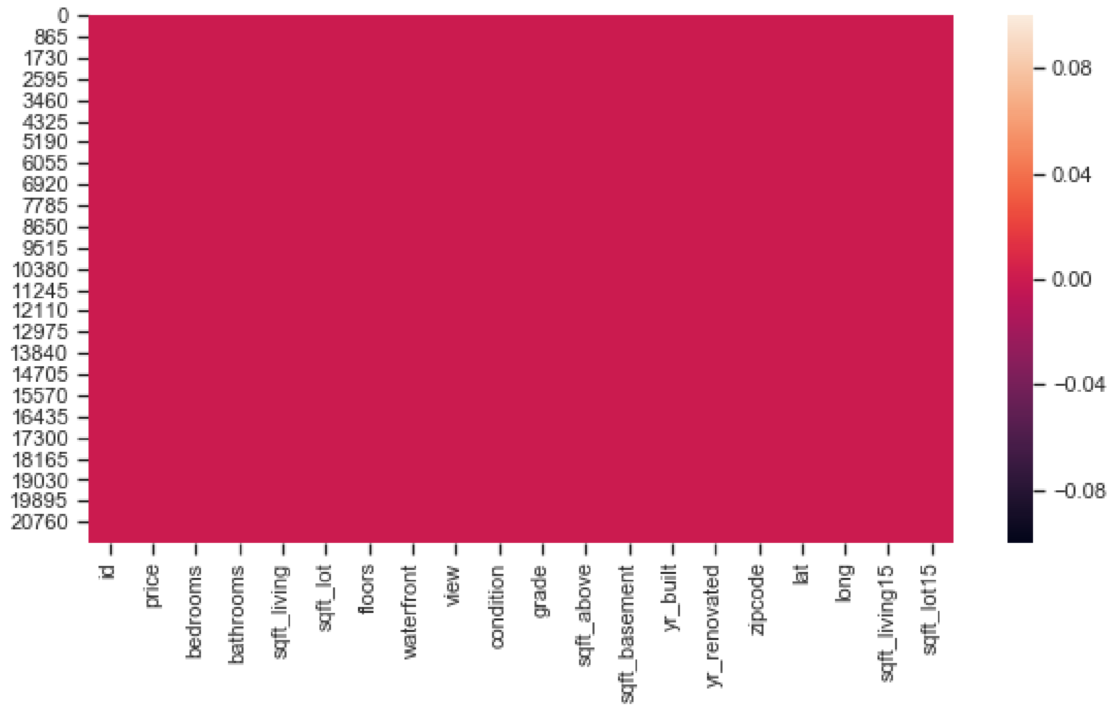


Imputing Values

In [35]:
```
#There is no need of date
data = data.drop(['date'], axis=1)
```

In [36]:
```
#Checking there is any null value or not
plt.figure(figsize=(10, 5))
sns.heatmap(data.isnull())
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x1c86ddf10f0>



There are no missing values in the data

1.4 Model Training

This is a Regression problem since we are predicting house prices which is a continous random variable. The steps involved are as follows:

1. Standardize or Normalize Training Data
2. Train Test Split
3. Train Model
4. Evaluation based on RMSE

Note: We will be implementing two models, Linear Regression and Random Forest Regressor

```
In [37]: # Take target variable into y
         y = data['price']
```

```
In [38]: # Delete the price
         del data['price']
```

```
In [39]: #Take their values in X and y
         X = data.values
         y = y.values
```

In [40]:
```
# Split data into train and test formate
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

Evaluation Metric

In [43]:
```
def get_rmse(y_pred, y_target):
    return np.sqrt(np.mean(np.square(y_pred.reshape(-1,) - y_target.reshape(-1,))))
```

Linear Regression

In [41]:
```
#Train the model
from sklearn import linear_model
model = linear_model.LinearRegression()
```

In [42]:
```
#Fit the model
model.fit(X_train, y_train)
```

Out[42]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [44]:
```
y_pred = model.predict(X_test)
```

In [46]:
```
# Score/Accuracy
print("Accuracy --> ", model.score(X_test, y_test)*100)
```

```
Accuracy -->  75.90091673275158
```

In [45]:
```
get_rmse(y_pred, y_test)
```

Out[45]:
```
0.25540977262992337
```

Random Forest Regression

In [47]:
```
#Train the model
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=1000)
```

In [48]:
```
#Fit the model
model.fit(X_train, y_train)
```

Out[48]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=1000,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [49]:
```
#Score/Accuracy
print("Accuracy --> ", model.score(X_test, y_test)*100)
```

```
Accuracy -->  88.56592044444415
```

```
In [50]: y_pred = model.predict(X_test)

In [51]: get_rmse(y_pred, y_test)

Out[51]: 0.17592917014179624
```