```
from google.colab import files


uploaded = files.upload()
```

Choose Files 2 files

- **test.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 310807 bytes, last modified: 8/31/2020 - 100% done
- **train.csv**(application/vnd.ms-excel) - 1065981 bytes, last modified: 8/24/2020 - 100% done

```
Saving test.xlsx to test.xlsx
Saving train.csv to train (2).csv
```

Define a polynomial regression model for predicting the house price and test the model against test set.

Importing all the libraries.

```
#for reading the data
import pandas as pd

#for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

#for making linear regression model
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use
  import pandas.util.testing as tm
```

For Reading the data

```
import pandas as pd
import io

df = pd.read_csv(io.BytesIO(uploaded['train.csv']))
print(df)

```

```
              id             date  ...  sqft_living15  sqft_lot15
0     2487200875  20141209T000000  ...           1360        5000
1     7237550310  20140512T000000  ...           4760      101930
2     9212900260  20140527T000000  ...           1330        6000
3      114101516  20140528T000000  ...           1780       12697
4     6054650070  20141007T000000  ...           1370       10208
...          ...              ...  ...            ...         ...
9756  9834201367  20150126T000000  ...           1400        1230
9757  3448900210  20141014T000000  ...           2520        6023
9758  7936000429  20150326T000000  ...           2050        6200
9759  1523300141  20140623T000000  ...           1020        2007
9760  1523300157  20141015T000000  ...           1020        1357

[9761 rows x 21 columns]
```

```
import io
df_test = pd.read_excel(io.BytesIO(uploaded['test.xlsx']))
print(df_test)
```

```
              id             date   price  ...     long  sqft_living15  sqft_lot15
0     3793500160  20150312T000000  323000  ... -122.031           2390        7570
1     1175000570  20150312T000000  530000  ... -122.394           1360        4850
```

Name of columns and description of data

```
import io

%matplotlib inline
#bring in the six packs
df_train =pd.read_csv(io.BytesIO(uploaded['train.csv']))

#check the decoration
df_train.columns
```

```
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')
```

```
#descriptive statistics summary
df_train['price'].describe()
#analysing sale price first
```

```
count    9.761000e+03
mean     5.428336e+05
std      3.797779e+05
min      8.000000e+04
25%      3.200000e+05
50%      4.500000e+05
75%      6.490000e+05
max      7.700000e+06
Name: price, dtype: float64
```

To get information regarding the datatypes.

```
print(df_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9761 entries, 0 to 9760
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             9761 non-null   int64  
 1   date           9761 non-null   object 
 2   price          9761 non-null   float64
 3   bedrooms       9761 non-null   int64  
 4   bathrooms      9761 non-null   float64
 5   sqft_living    9761 non-null   float64
 6   sqft_lot       9761 non-null   int64  
 7   floors         9761 non-null   int64  
 8   waterfront     9761 non-null   int64  
 9   view           9761 non-null   int64  
 10  condition      9761 non-null   int64  
 11  grade          9761 non-null   int64  
 12  sqft_above     9761 non-null   int64  
 13  sqft_basement  9761 non-null   float64
 14  yr_built       9761 non-null   int64  
 15  yr_renovated   9761 non-null   int64  
 16  zipcode        9761 non-null   int64  
 17  lat            9761 non-null   float64
 18  long           9761 non-null   float64
 19  sqft_living15  9761 non-null   int64  
 20  sqft_lot15     9761 non-null   int64  
dtypes: float64(6), int64(14), object(1)
memory usage: 1.6+ MB
None
```

To check for duplicate or null values as a part of data preprocessing.

```
dupl_rows = df_train.duplicated().sum()
print(dupl_rows)
null_v = df_train.isnull().sum()
print(null_v)
```

```
0
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
```

From the above descriptive analysis we can say the data does not require any preprocessing.

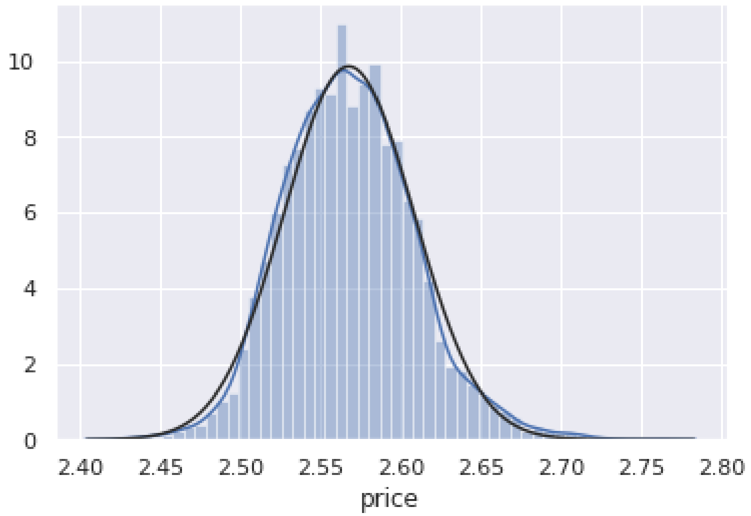
Visualization of data

Creating a distplot for the price values to see if the data is skewed in any manner and whether it might need some normalizing or not.

```
#histogram
sns.distplot(df_train['price']);
#plotting histogram
```

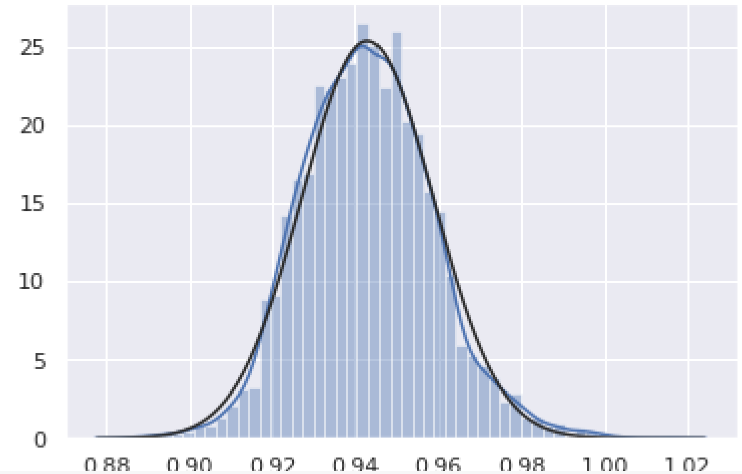

```
sns.distplot(df_train['price'], fit = norm)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff7893d8358>




```
sns.distplot(np.log(df_train['price']), fit = norm)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff7892c8240>
```

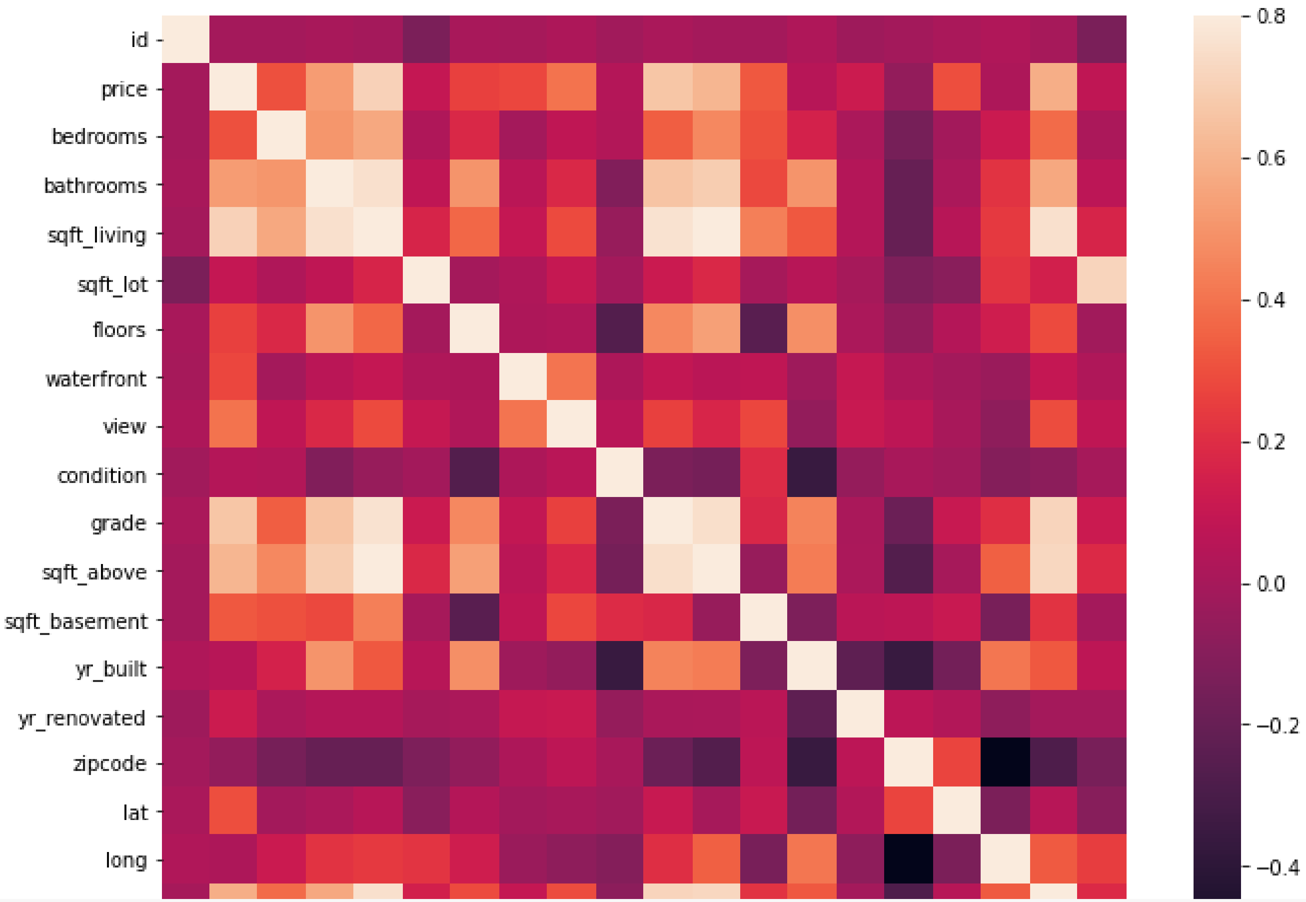



```
#skewness and kurtosis
print("Skewness: %f" % df_train['price'].skew())
print("Kurtosis: %f" % df_train['price'].kurt())
```

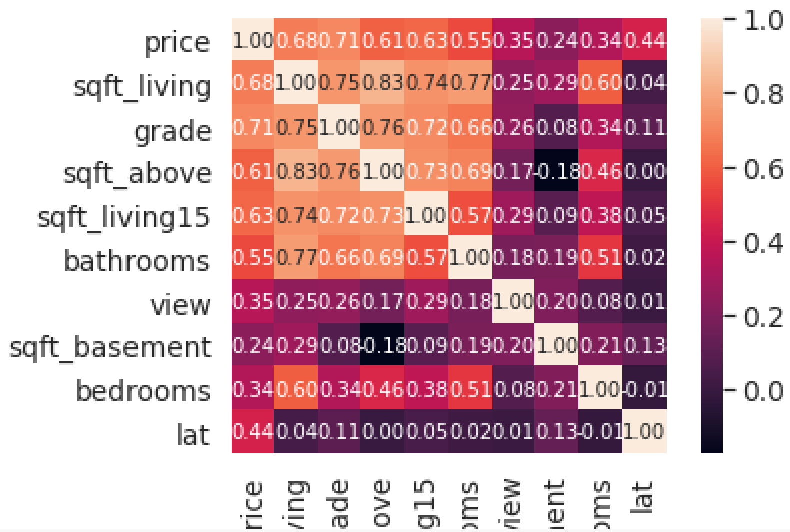
```
Skewness: 4.296023
Kurtosis: 38.871048
```

To select the important features for predicting the price on the basis of the heatmap.

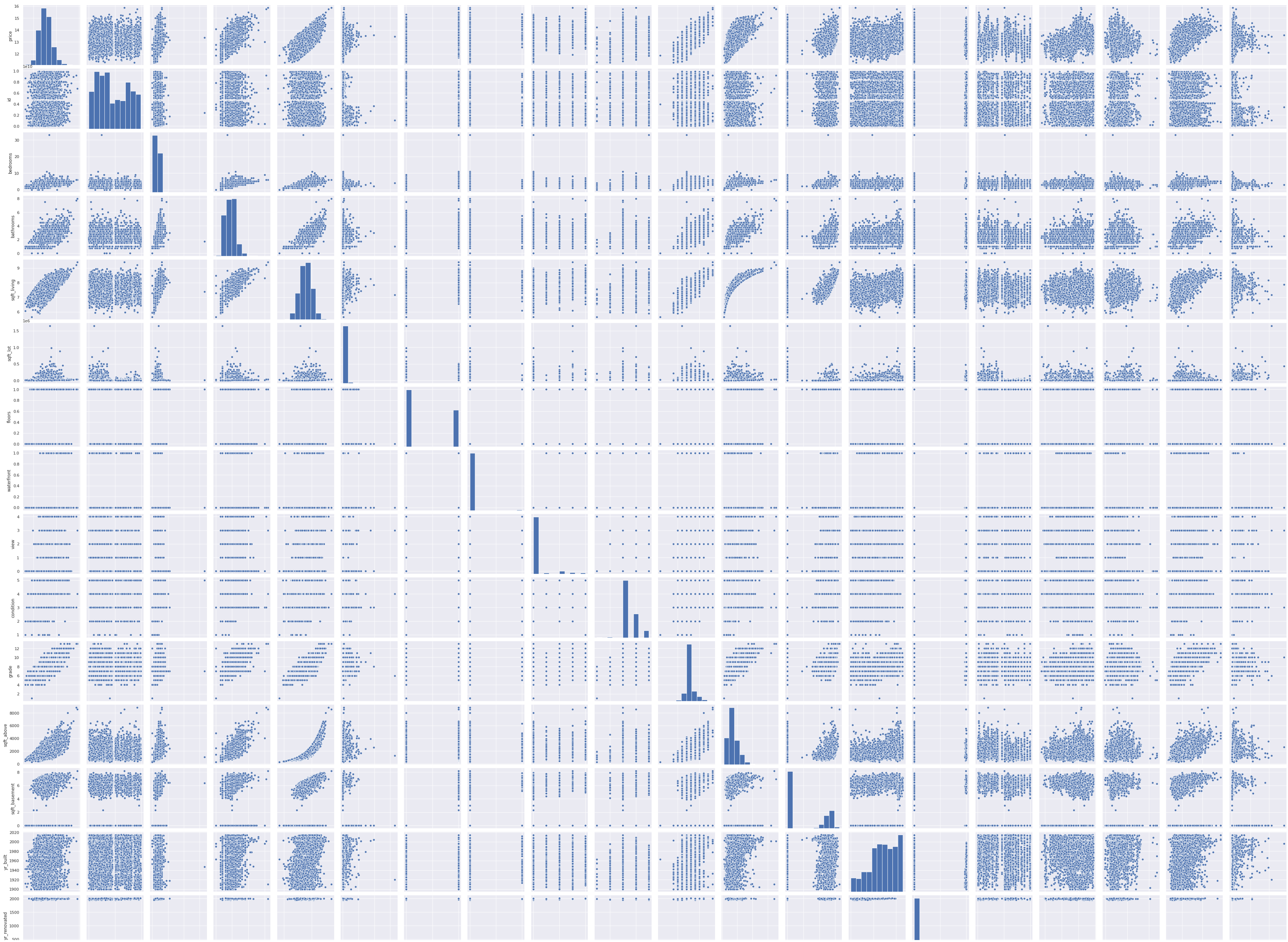
```
#correlation matrix
corrmat = df_train.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



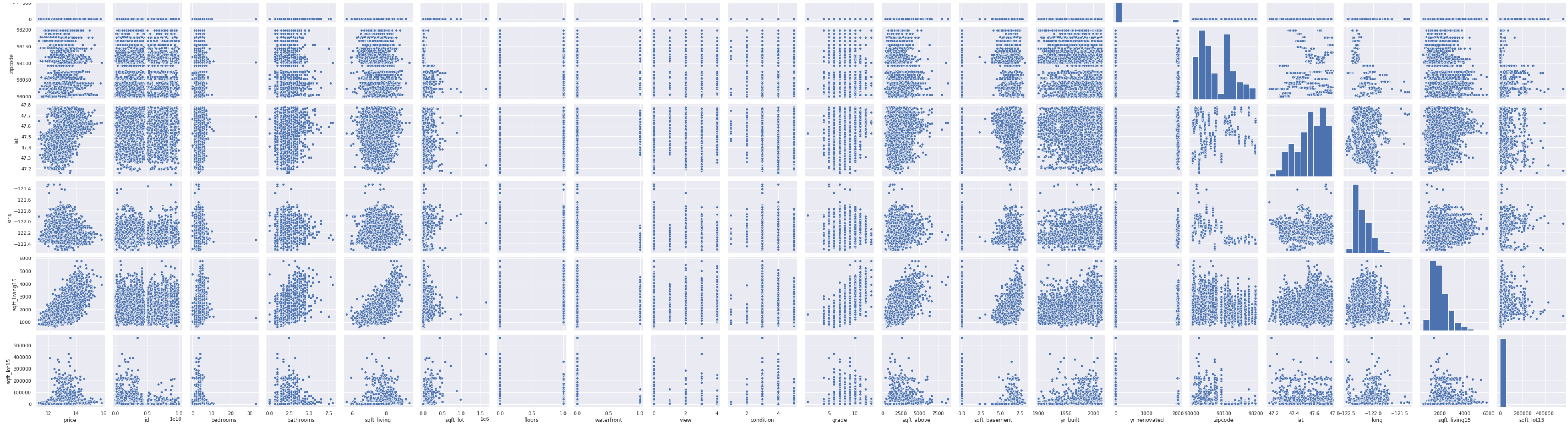

```
#price correlation matrix
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'price')['price'].index
cm = np.corrcoef(df_train[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=c
plt.show()
```




```
#scatterplot
sns.set()
cols = ['price', 'id', 'date', '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']
# here i ve chosen the 10 most important features for prediciting house prices
sns.pairplot(df_train[cols], size = 2.5)
plt.show();
```

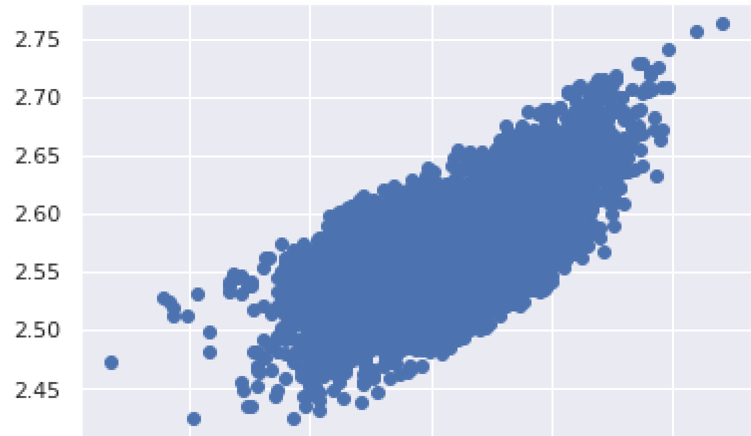


price
id
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
sqft_above
sqft_basement
yr_built
yr_renovated



Extracting the important features

```
X = df_train["sqft_living"]
y = df_train["price"]
plt.scatter(X,y)
plt.show()
```




```
X_feat = [imp_feat.index]
X1 =[]
for i in range(0,11):
  X1.append(X_feat[0][i])
print(X1)
```

```
['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'grade', 'sqft_above', 'sqft_base
```

```
target = abs(corr["price"])
imp_feat = target[target > 0.1]
print(imp_feat)
```

```
price            1.000000
bedrooms         0.339246
```

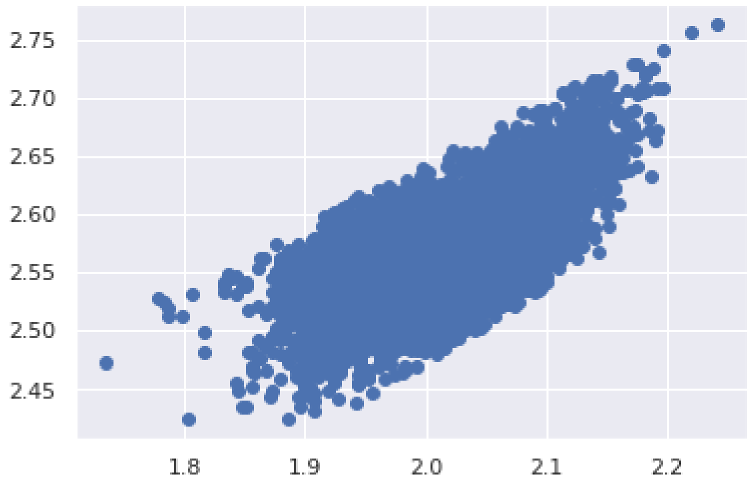
Visualizing some of the important features with respect to the target variable

```
sqft_lot         0.108285
```

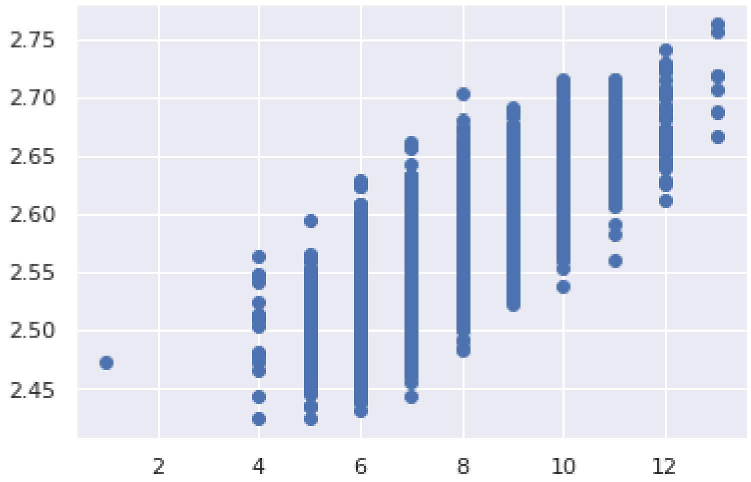
```
cor_mat = df_train.corr()
feat = 11
fields = cor_mat.nlargest(feat, 'price')['price'].index
field = list(fields)
field.remove('price')
print(field)
```

```
['grade', 'sqft_living', 'sqft_living15', 'sqft_above', 'bathrooms', 'lat', 'view', 'bedrooms', 'sqft_basement', 'floors']
```

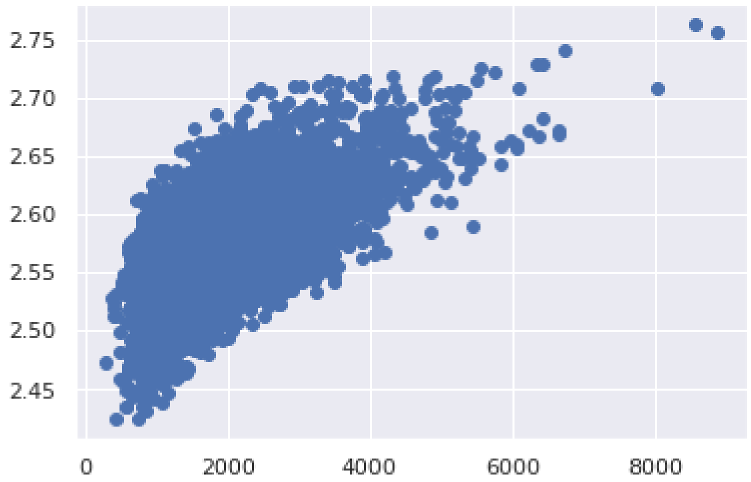
```
X = df_train["sqft_living"]
y = df_train["price"]
plt.scatter(X,y)
plt.show()
```



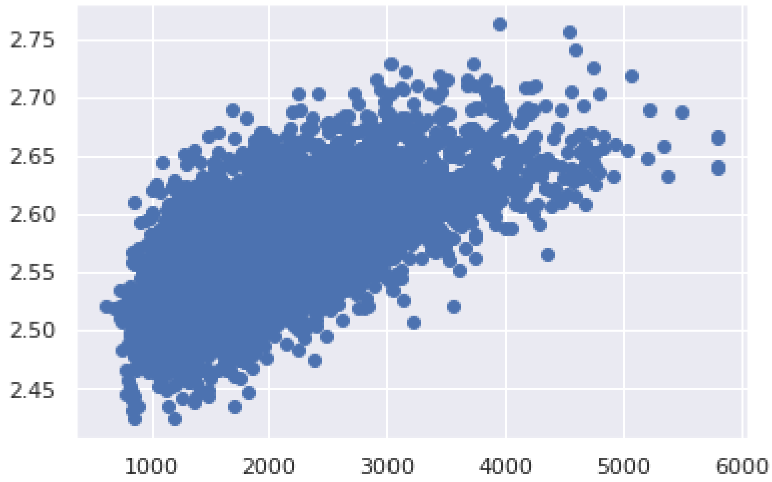

```
X = df_train["grade"]
plt.scatter(X,y)
plt.show()
```



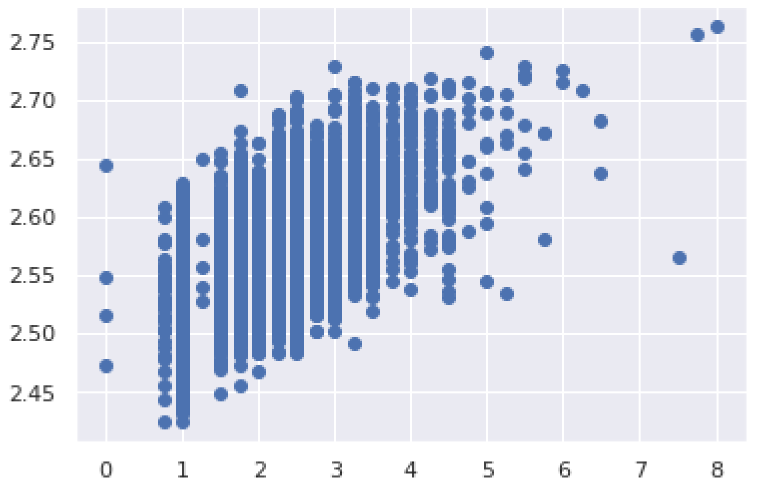

```
X = df_train["sqft_above"]
plt.scatter(X,y)
plt.show()
```




```
X = df_train["sqft_living15"]
plt.scatter(X,y)
plt.show()
```




```
X = df_train["bathrooms"]
plt.scatter(X,y)
plt.show()
```



We build a polynomial regression model now

For degree 4

```
polyF = PolynomialFeatures(degree = 4)
x_poly_train = polyF.fit_transform(df_train[X1])
x_poly_test = polyF.fit_transform(df_test[X1])
LinReg = LinearRegression()
LinReg.fit(x_poly_train, df_train['price'])
yPred = LinReg.predict(x_poly_test)
```

Since we're performing regression, we use MSE and RMSE to evaluate the models, with lower values for each indicating a more accurate model.

```
from sklearn import metrics
print("Degree 4:")
MSE = metrics.mean_squared_error(df_test['price'], yPred)
print("MSE:", round(np.sqrt(MSE),2))
print("R-squared train: ", round(LinReg.score(x_poly_train, df_train['price']),3))
print("R-squared test: ", round(LinReg.score(x_poly_test, df_test['price']),3))
```

```
Degree 4:
MSE: 539798.14
R-squared train:  0.842
R-squared test:  -1.269
```

For degree 3

```
polyF = PolynomialFeatures(degree = 3)
x_poly_train = polyF.fit_transform(df_train[X1])
x_poly_test = polyF.fit_transform(df_test[X1])
LinReg = LinearRegression()
LinReg.fit(x_poly_train, df_train['price'])
yPred = LinReg.predict(x_poly_test)
```

```
print("Degree 3:")
```

```python
MSE = metrics.mean_squared_error(df_test['price'], yPred)
print("MSE:", round(np.sqrt(MSE),2))
print("R-squared train: ", round(LinReg.score(x_poly_train, df_train['price']),3))
print("R-squared test: ", round(LinReg.score(x_poly_test, df_test['price']),3))
```

```
Degree 3:
MSE: 198454.08
R-squared train:  0.804
R-squared test:  0.693
```

For degree 2

```python
polyF = PolynomialFeatures(degree = 2)
x_poly_train = polyF.fit_transform(df_train[X1])
x_poly_test = polyF.fit_transform(df_test[X1])
LinReg = LinearRegression()
LinReg.fit(x_poly_train, df_train['price'])
yPred = LinReg.predict(x_poly_test)
print("Degree 2:")
MSE = metrics.mean_squared_error(df_test['price'], yPred)
print("MSE:", round(np.sqrt(MSE),2))
print("R-squared train: ", round(LinReg.score(x_poly_train, df_train['price']),3))
print("R-squared test: ", round(LinReg.score(x_poly_test, df_test['price']),3))
```

```
Degree 2:
MSE: 190141.76
R-squared train:  0.768
R-squared test:  0.718
```

Conclusion

From the analysis and prediction performed on the training and test datasets, it can conclusively be stated that the best Polynomial Regressor for the given problem is having a degree = 3. It is also slightly better than performing simple multivariate linear regression (so is polynomial degree = 2)as is evident from the slight improvement in the values of MSE and RMSE.