

```
1 from google.colab import files
2
3
4 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.

```
1 #for reading the data
2 import pandas as pd
3
4 #for data visualization
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 import numpy as np
8
9 #for making linear regression model
10 from scipy.stats import norm
11 from sklearn.preprocessing import StandardScaler
12 from scipy import stats
13 import warnings
14 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

```
1 import pandas as pd
2 import io
3
4 df = pd.read_csv(io.BytesIO(uploaded['train.csv']))
5 print(df)
6
```

```
↳
```

	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]

```
1 import io
2 df_test = pd.read_excel(io.BytesIO(uploaded['test.xlsx']))
3 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

```
1 import io
2
3 %matplotlib inline
4 #bring in the six packs
5 df_train =pd.read_csv(io.BytesIO(uploaded['train.csv']))
6
7 #check the decoration
8 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')
```

```
1 #descriptive statistics summary
2 df_train['price'].describe()
3 #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.

```
1 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.

```
1 dupl_rows = df_train.duplicated().sum()
2 print(dupl_rows)
3 null_v = df_train.isnull().sum()
4 print(null_v)
```

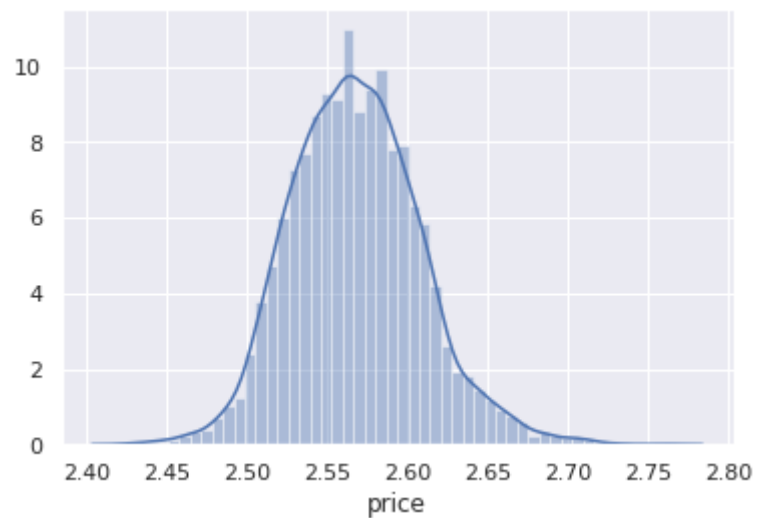
```
0  
id 0  
date 0  
price 0  
bedrooms 0  
bathrooms 0  
sqft_living 0  
sqft_lot 0  
floors 0  
waterfront 0  
view 0  
condition 0  
grade 0  
sqft_above 0  
sqft_basement 0  
yr_built 0  
yr_renovated 0  
zipcode 0  
lat 0  
long 0  
sqft_living15 0  
sqft_lot15 0  
dtype: int64
```

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.

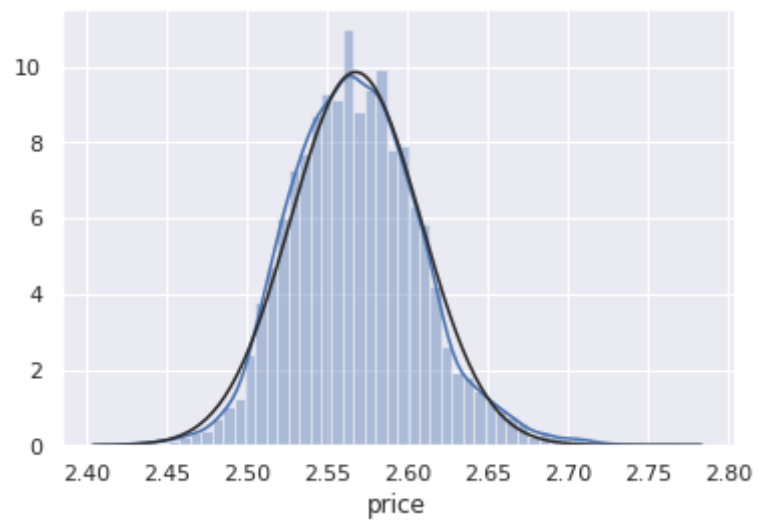
```
1 #histogram  
2 sns.distplot(df_train['price']);  
3 #plotting histogram
```



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



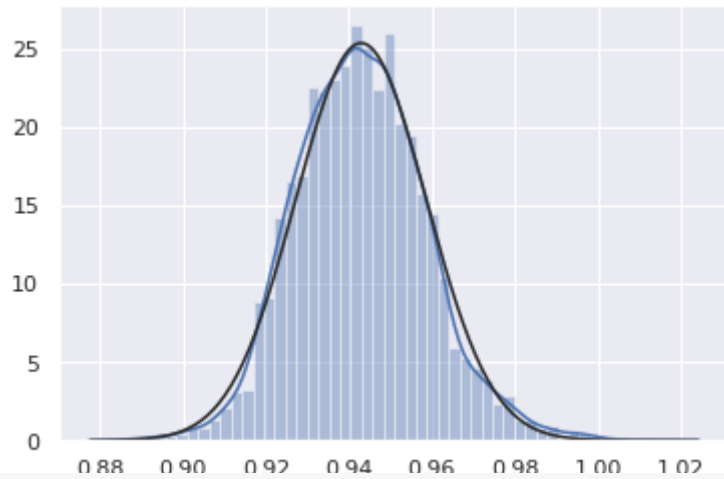
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff7893d8358>
```



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



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



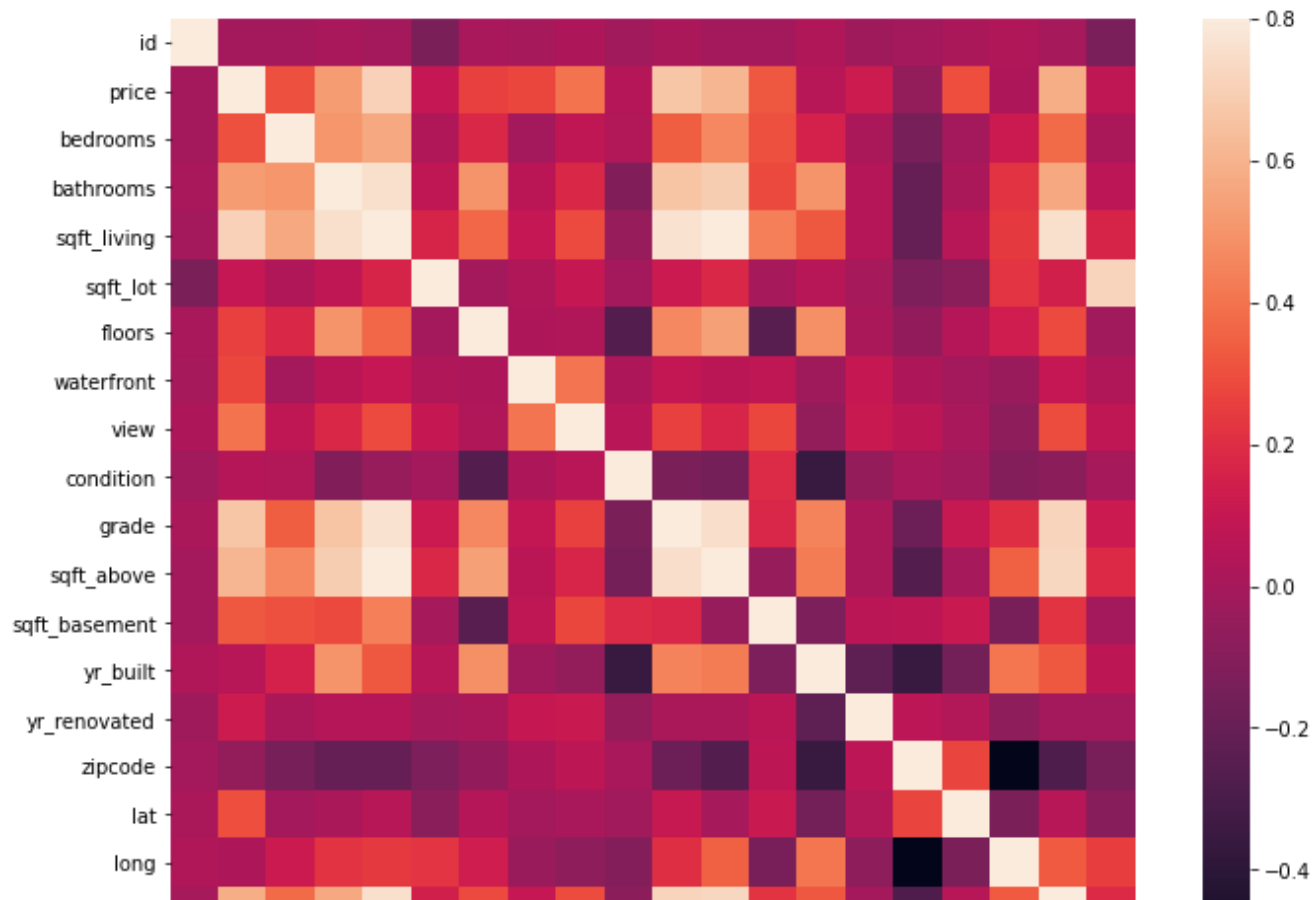
```
1 #skewness and kurtosis
2 print("Skewness: %f" % df_train['price'].skew())
3 print("Kurtosis: %f" % df_train['price'].kurt())
```

➞ Skewness: 4.296023
Kurtosis: 38.871048

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

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

➞

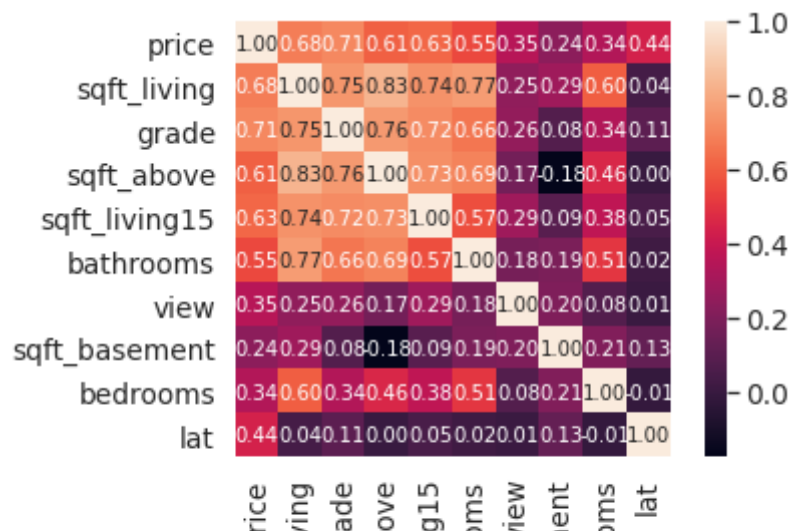


```

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

```



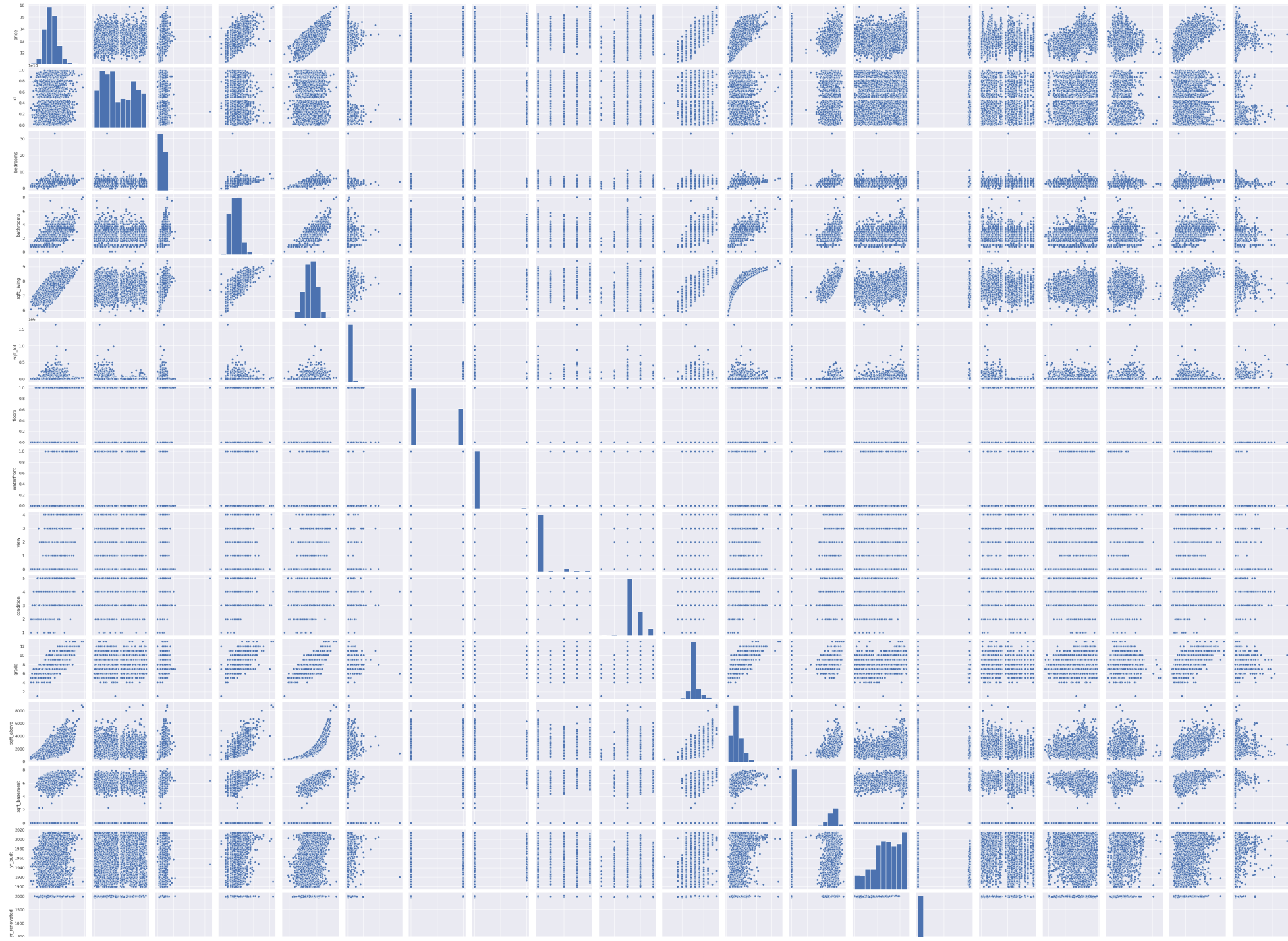


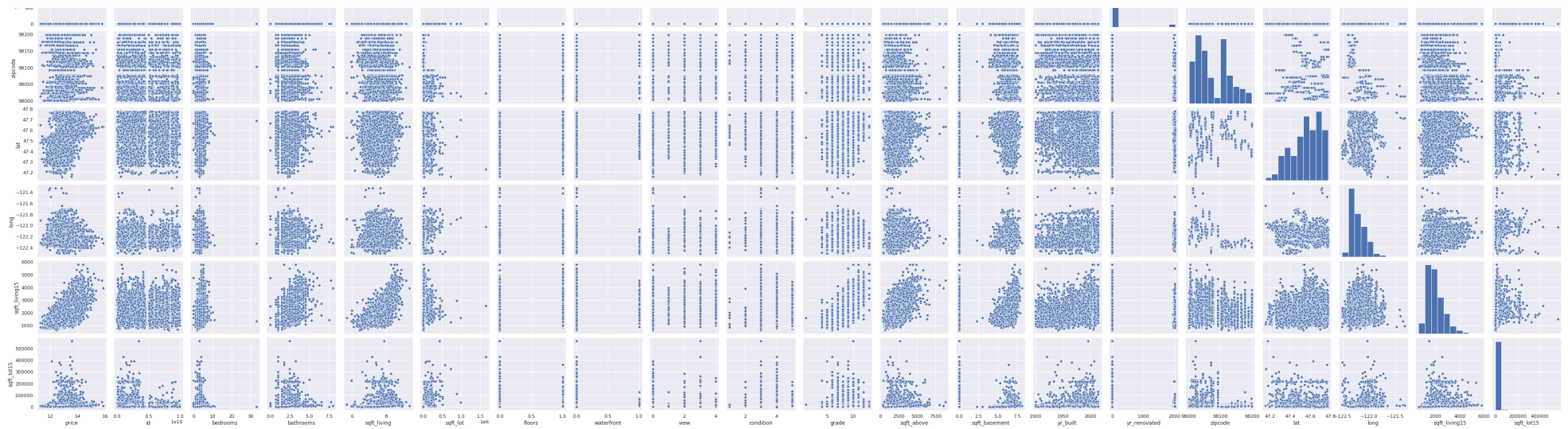
```

1 #scatterplot
2 sns.set()
3 cols = ['price', 'id', 'date', 'bedrooms', 'bathrooms', 'sqft_living',
4         'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
5         'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
6         'lat', 'long', 'sqft_living15', 'sqft_lot15']
7 # here i ve chosen the 10 most important features for prediciting house prices
8 sns.pairplot(df_train[cols], size = 2.5)
9 plt.show();

```







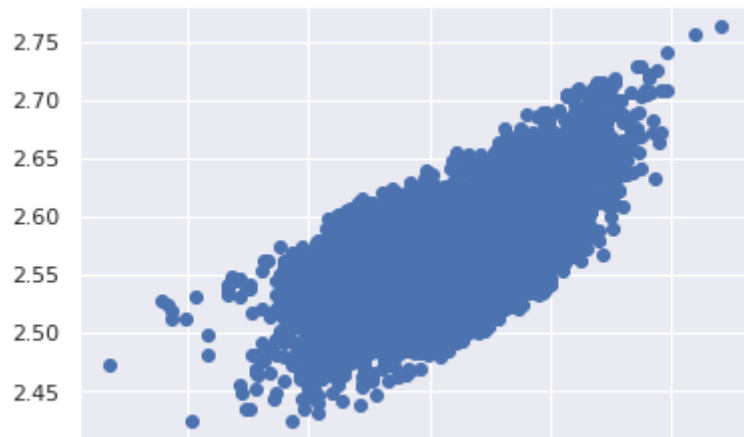
Extracting the important features

```

1 X = df_train["sqft_living"]
2 y = df_train["price"]
3 plt.scatter(X,y)
4 plt.show()

```





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

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

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

☞

price	1.000000
bedrooms	0.339246

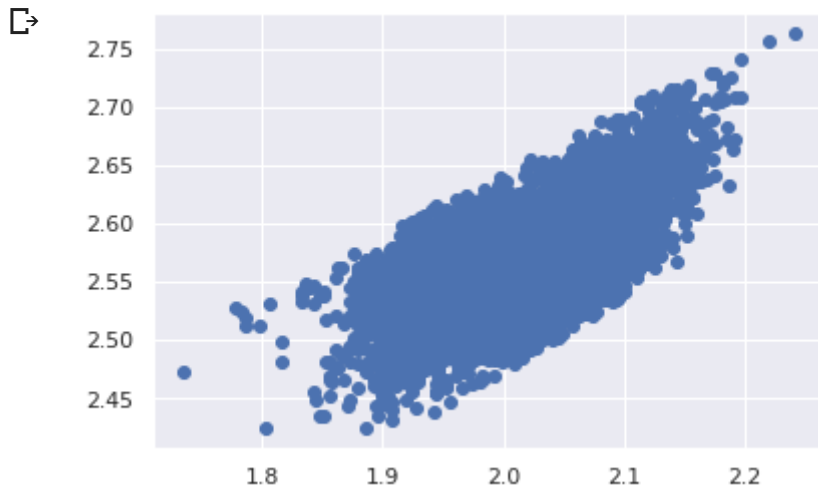
Visualizing some of the important features with respect to the target variable

sqft_totl	0.100203
-----------	----------

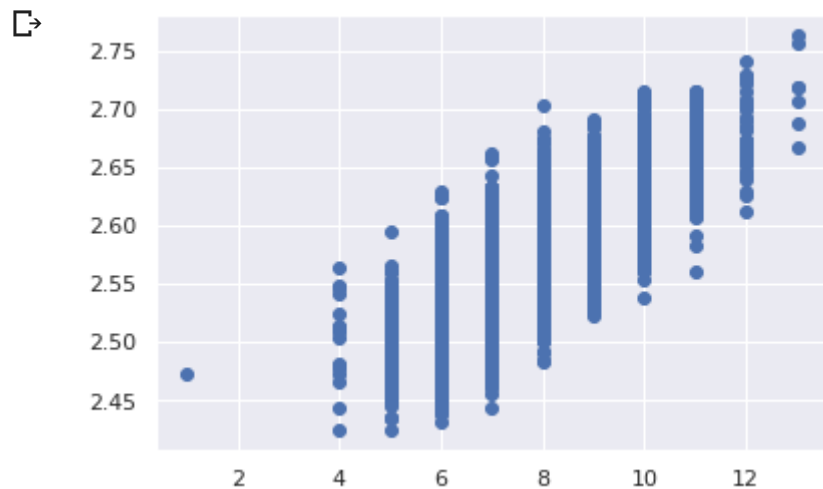
```
1 cor_mat = df_train.corr()
2 feat = 11
3 fields = cor_mat.nlargest(feat, 'price')['price'].index
4 field = list(fields)
5 field.remove('price')
6 print(field)
```

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

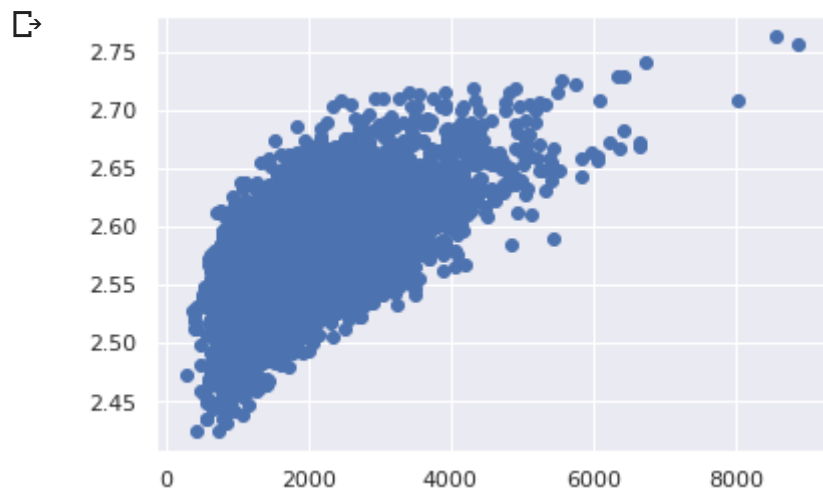
```
1 X = df_train["sqft_living"]
2 y = df_train["price"]
3 plt.scatter(X,y)
4 plt.show()
```



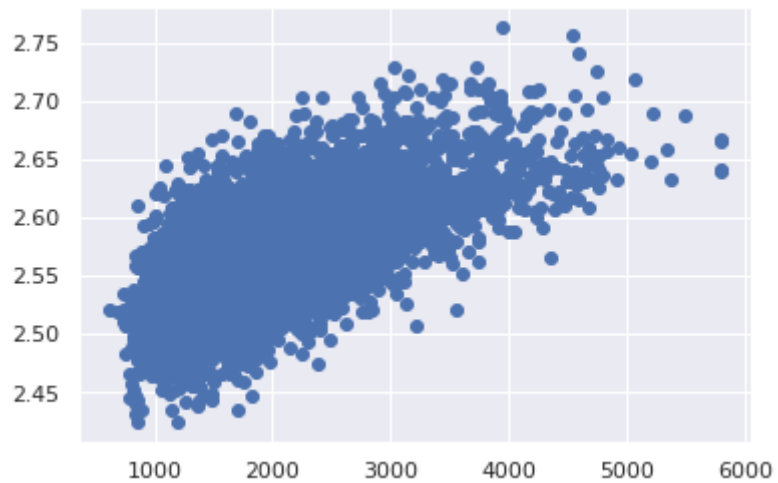
```
1 X = df_train["grade"]
2 plt.scatter(X,y)
3 plt.show()
```



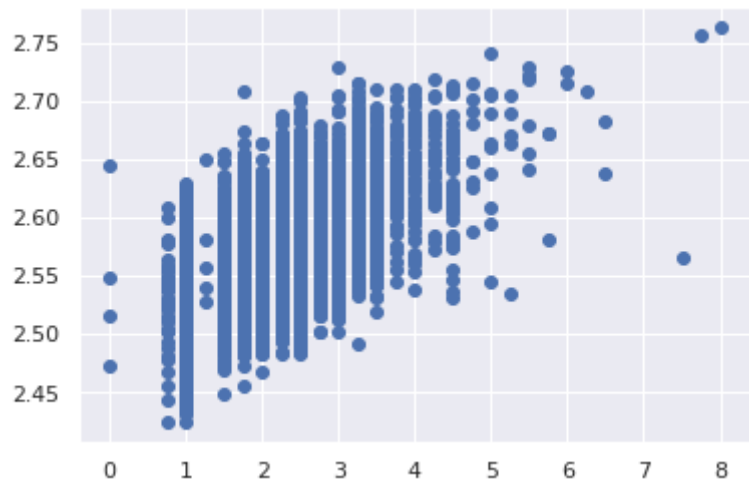
```
1 X = df_train["sqft_above"]  
2 plt.scatter(X,y)  
3 plt.show()
```



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



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



We build a polynomial regression model now

For degree 4

```
1 polyF = PolynomialFeatures(degree = 4)
2 x_poly_train = polyF.fit_transform(df_train[X1])
3 x_poly_test = polyF.fit_transform(df_test[X1])
4 LinReg = LinearRegression()
5 LinReg.fit(x_poly_train, df_train['price'])
6 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.

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

```
↳ Degree 4:
MSE: 63397114366.3
R-squared train: -8.797
R-squared test: -31300045513.66
```

For degree 3

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

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



```

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

```

```

↳ Degree 3:
MSE: 10503126196449.53
R-squared train: -0.106
R-squared test: -859098884335090.1

```

For degree 2

```

1 polyF = PolynomialFeatures(degree = 2)
2 x_poly_train = polyF.fit_transform(df_train[X1])
3 x_poly_test = polyF.fit_transform(df_test[X1])
4 LinReg = LinearRegression()
5 LinReg.fit(x_poly_train, df_train['price'])
6 yPred = LinReg.predict(x_poly_test)
7 print("Degree 2:")
8 MSE = metrics.mean_squared_error(df_test['price'], yPred)
9 print("MSE:", round(np.sqrt(MSE),2))
10 print("R-squared train: ", round(LinReg.score(x_poly_train, df_train['price']),3))
11 print("R-squared test: ", round(LinReg.score(x_poly_test, df_test['price']),3))

```

```

↳ Degree 2:
MSE: 1242268907.39
R-squared train: 1.0
R-squared test: -12018137.89

```

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.

