Name- Chetan Mahawar
Roll No.-193010030

HOUSE PRICE PREDICTION ANALYSIS

The prediction of House Price on sale can be done on basis of number of features but the most major features considered for Price predictions are sqft_living, sqft_lot, sqft_basement, Bedrooms and Bathrooms number , waterfront, View, Grade etc. The complete analysis is done using Python language with Pandas dataframe.

Data Cleansing is the initial step for data analysis after importing file and necessary libraries for which the command are as follows:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv('kc_house_data_NaN.csv')
df.head()
df.columns
```

Number of columns or features present in dataset are:

Index(['Unnamed: 0', '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')

with total number of rows equal to 21613

For removing unnecessary columns and dropping of rows with **NaN** values following code is used,

```
column_1=df.columns[0]
column_2=df.columns[1]
df.drop([column_1,column_2],axis=1,inplace=True)
df=df.dropna()
df=df.reset_index()
```

which result into features as:

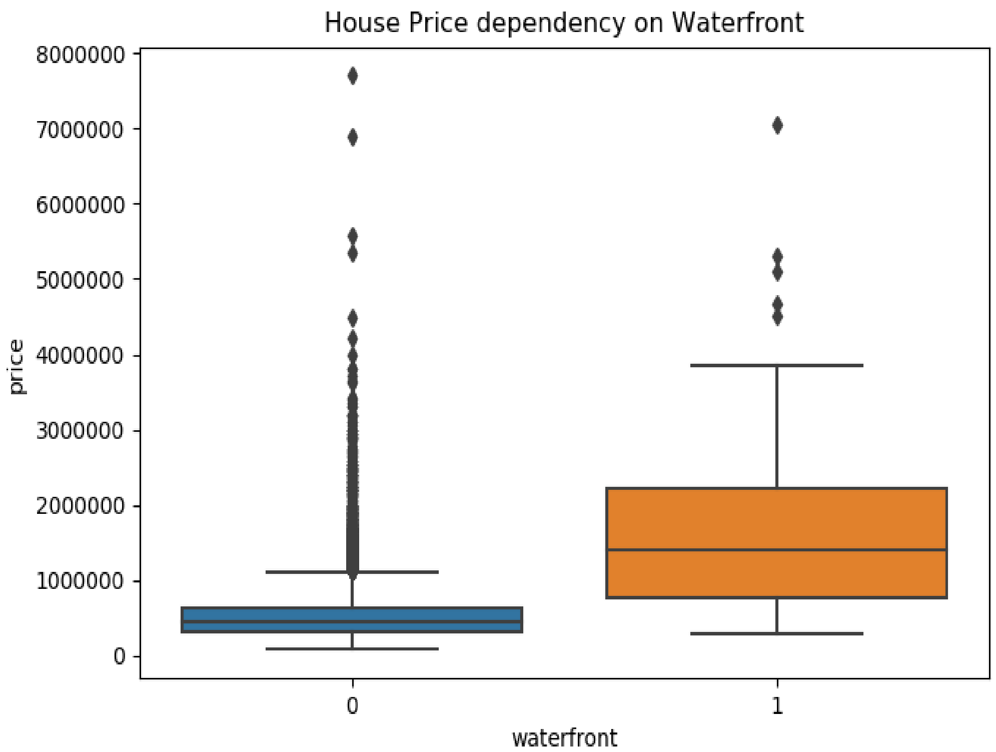
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15'],
dtype='object')

with 2 columns removed and total number of rows equal to 21597

Data Visualization and Data Exploratory Analysis

Data Visualization is done to observe trends and relations among various features and also of features with target variable. Quantitative relations among features is calculated using Statistical parameter such as Pearson Correlation coefficient and p-value etc

The following are some plots for considered dataset with inferences :



The box plot clearly indicates that prices of houses are more with waterfront than ones without waterfront but Outliers are more for houses without waterfront

Pearson corelation coefficient: 0.3
p_value : 0.0

The statistics value indicates Weak positive relation and Weak Uncertainity in results between features i.e feature "Waterfront" has less influence in predicting house prices

Code used for plot **:**

```
clf()
sns.boxplot(x=df['waterfront'],y=df['price'])
plt.show()
plt.title('House Price dependency on Waterfront')
```

Code used for finding corelation :

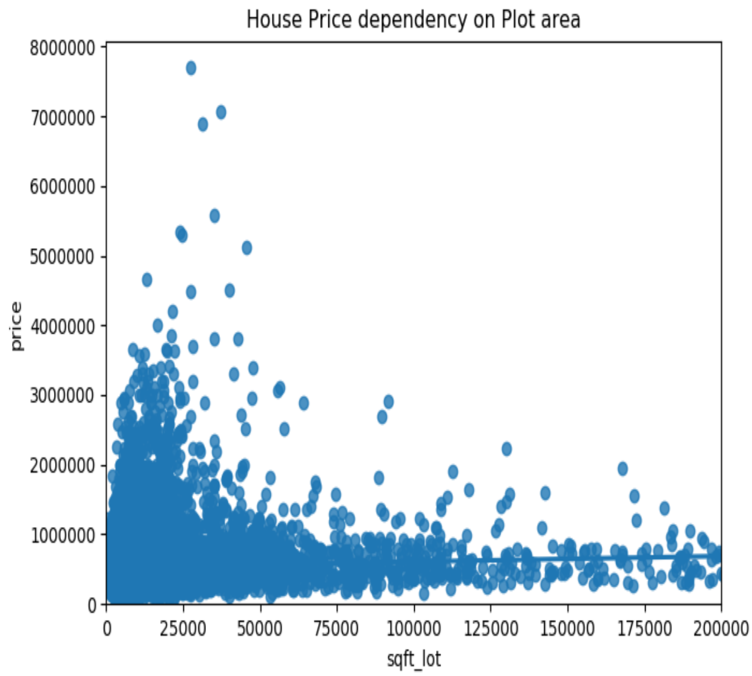
```
import scipy.stats as ss
Pearson_coef,p_value=ss.pearsonr(df['waterfront'],df['price'])
print("Pearson correlation coefficient:","{0:.1f}".format(Pearson_coef))
print("p_value :",p_value)
```

Plot clearly depicts that Flat area has major influence in predicting House prices and shows a proportional trend

The stats value indicates Moderate positive relation b/w Flat area and Price and Weak certainity in results i.e Flat area has moderate influence in predicting house price

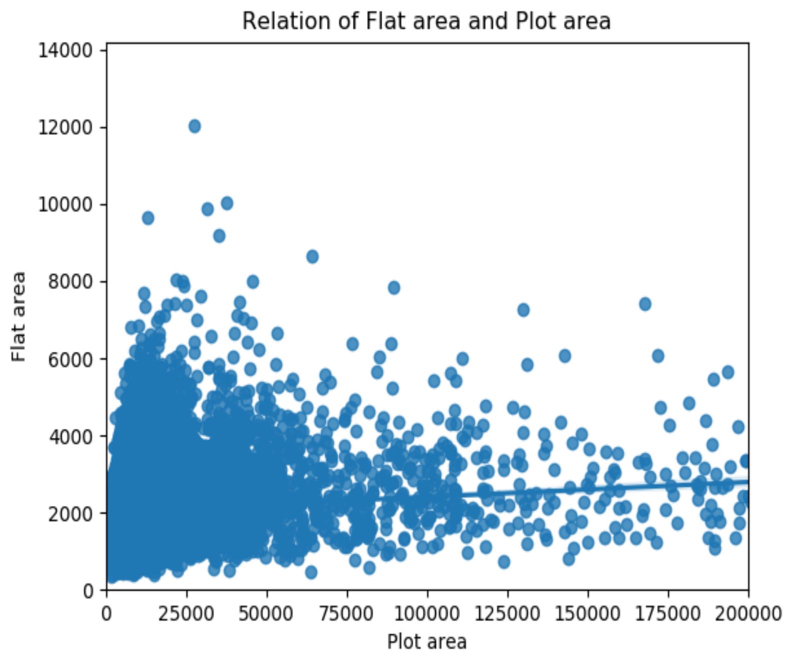
Pearson correlation coefficient: 0.7
p_value : 0.0

The Pearson Correlation value as well as plot clearly shows that Plot area is not the major feature in predicting house prices because with increase in Plot area House Prices nearly remains constant



Pearson correlation coefficient: 0.1
p_value : 5.51100044898e-40

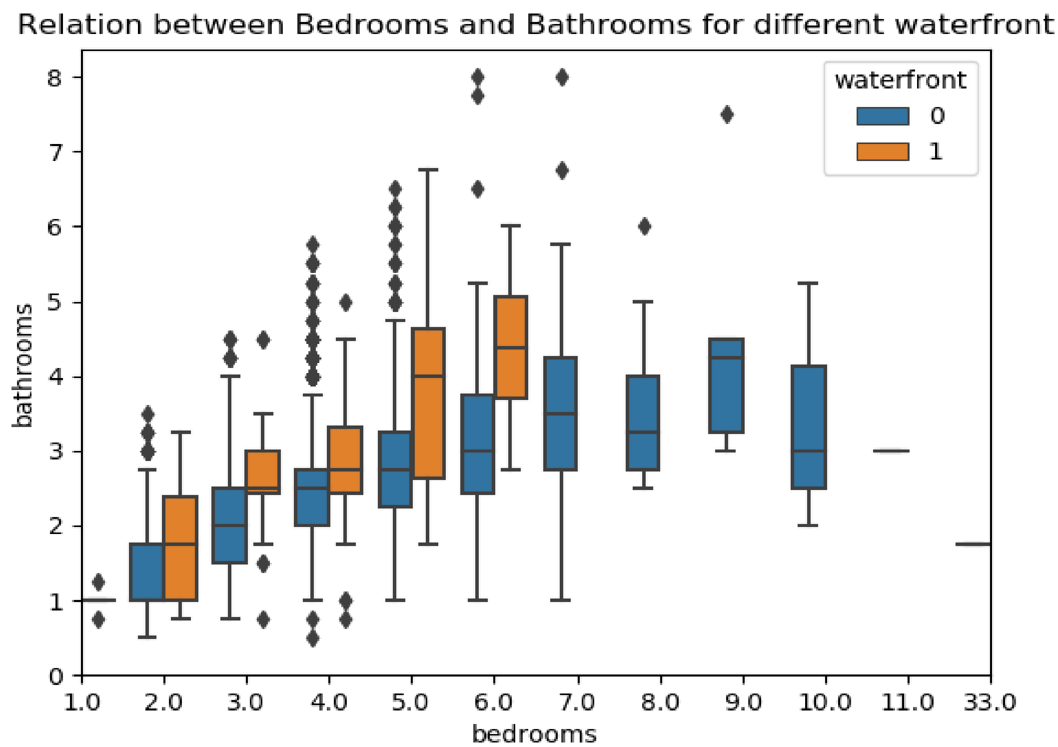
The plot area and Price has very low positive relation but high certainity in results and therefore Weak Corelation between features



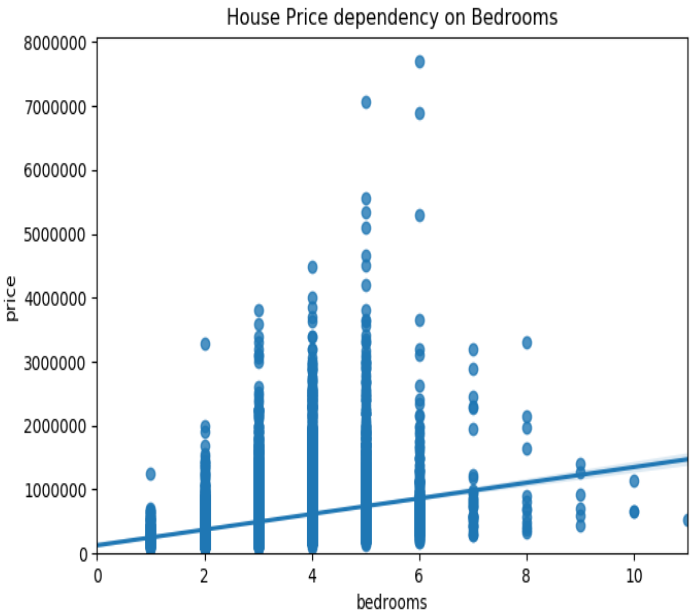
The Correlation results as well as regression plot clearly tells that there is no relation b/w Plot area and Flat area because flats in majority of cases are built on small piece of land even when large land area is available

Pearson correlation coefficient: 0.2

p_value : 1.76829128423e-145



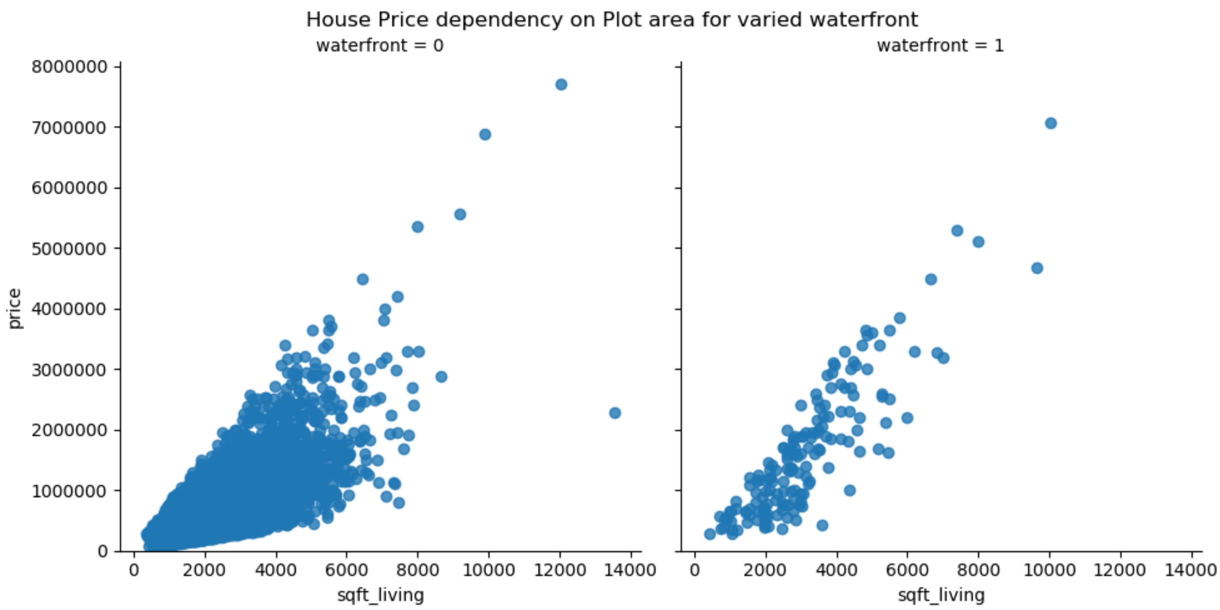
According to Box plot it can be generally concluded that with increasing number of bedrooms, number of bathrooms increases and it's also observed that corresponding to each bedroom number of bathrooms are more for houses with waterfront than houses



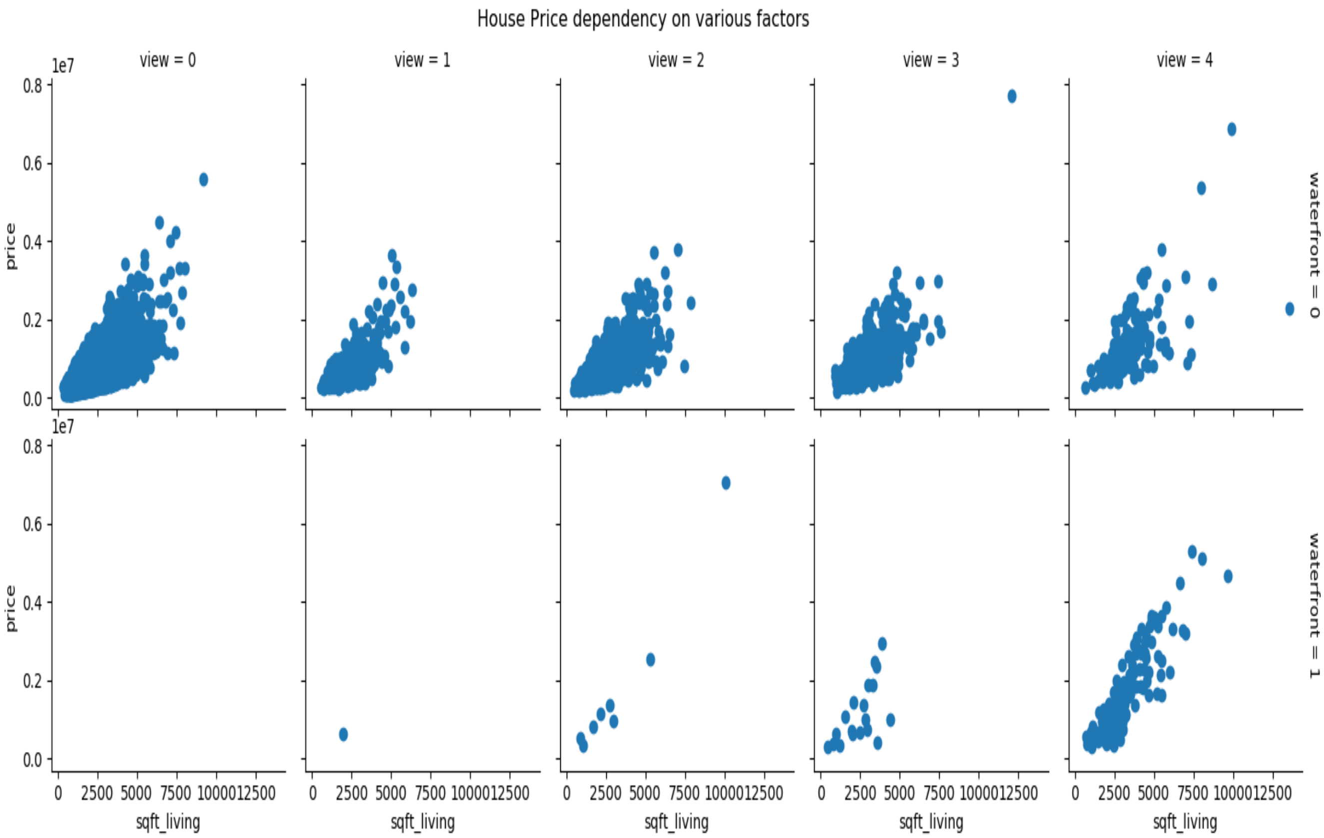
The regression plot and Correlation value indicates that larger number of bedrooms doesn't necessarily means houses with higher prices

Pearson correlation coefficient: 0.3
p_value : 0.0

There exsist low correlation between bedroom and Price since very low positive relation and Weak certainity in results



The regression plot with varied number of columns according to a categorical feature is plotted using "col" attribute which indicates the influence of categorical feature in predicting house price corresponding to another particular feature



A facetgrid regression plot provides a way to visualise the effect of more number of features together influencing the target variable and hence gives a more broader aspect for making conclusion. From the plot its clear that there are more number of houses without waterfront with a lower view than houses with waterfront and thus house prices shows a proportional trend with Flat area than with other two features.

The total plot area corresponding to each view is given by

view	sqft_lot
0	14168.931553
1	12358.918675
2	22297.290323
3	34788.884314
4	21594.987461

and is calculated using code:

df.groupby('view')['sqft_lot'].mean()

Model development is training the different types of model on given dataset by bifurcating dataset into training and test set.
Then a particular model is evaluated using "score" attribute of model which basically determines the mean square error in fitting the test dataset to particular model and then by "Predicting" target variable values corresponding to test data, we can calculate the accuracy of fitting the model to dataset and this process is termed as **Model Evaluation**
SciPy and Stats libraries are imported for Model development and Model Evaluation.

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
```

Then features corresponding to data set and target variable is decided as follows:

```
features=["floors","waterfront","lat","bedrooms","sqft_basement","view","bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
X=df[features]
Y=df['price']
```

Using complete dataset to train the model and evaluate Mean square error using

```
from sklearn.linear_model import LinearRegression
lrm = LinearRegression()
lrm.fit(X,Y)
R_sq=lrm.score(X,Y)
print('coefficient of determination by Linear Regression Model is:',"{0:.3f}".format(R_sq))
```

coefficient of determination by Linear Regression Model is: **0.658**

Dividing entire dataset into train and test set in 70:30 ratio and then use training dataset to train the model

```
X_train, X_test,Y_train,Y_test = train_test_split(X,Y,test_size=.3)
```

Then using only training dataset to fit the model and evaluating Mean square error using

```
from sklearn.linear_model import LinearRegression
lrm = LinearRegression()
lrm.fit(X_train,Y_train)
R_sq=lrm.score(X_train,Y_train)
print('coefficient of determination by Linear Regression Model is:',"{0:.3f}".format(R_sq))
```

coefficient of determination by Linear Regression Model is: **0.664**

Normalizing complete dataset using Standard Scaler module as:

```
SS=StandardScaler()
X_train_scaled=SS.fit_transform(X_train)

SS=StandardScaler()
X_test_scaled=SS.fit_transform(X_test)
```

Converting entire data set into Polynomial feature of 2^{nd} order

```
poly=PolynomialFeatures(degree=2)
X_train_poly=poly.fit_transform(X_train_scaled)

poly=PolynomialFeatures(degree=2)
X_test_poly=poly.fit_transform(X_test_scaled)
```

Then using only training dataset to fit the model and evaluating Mean square error using

```
from sklearn.linear_model import LinearRegression
lrm = LinearRegression()
lrm.fit(X_train_poly,Y_train)
R_sq=lrm.score(X_train_poly,Y_train)
print('coefficient of determination by Regression Model for Polynomial feature is:',
"{0:.3f}".format(R_sq))
```

coefficient of determination by Regression Model for Polynomial feature is: **0.755**

The increasing value of $\mathbf{R^2}$ indicates that model has been improvised to best fit to training data set

Now, predicting value for test data set and then comparing the Y_test value with Predicted value for top ten test data set values

```
predict=pd.DataFrame()
predict['Predicted']=lrm.predict(X_test_poly)
predict[0:10]
Y_test [0:10]
```

Index	Predicted	Index	price
0	-2.234234e+14	15262	513000.0
1	-4.074729e+14	8745	1175000.0
2	6.482222e+15	2747	722500.0
3	1.866879e+15	7706	340000.0
4	-4.758011e+13	2035	369950.0
5	2.199807e+16	3234	739000.0
6	6.399591e+14	19208	338000.0
7	5.627986e+14	5765	375000.0
8	1.217803e+15	10504	148226.0
9	-7.784057e+14	2917	772000.0

As we can observe from the above columns that Predicted Price and Actual Price corresponding to particular test dataset are very much different and doesn't even lies in close proximity, therefore we train and predict the value using **Ridge Regression Model** of SciPy library

Importing Ridge model from SciPy library and then using only training dataset by Normalizing and converting into Polynomial features to fit the model and evaluating Mean square error to predict the accuracy of fitting model on training data set using

Normalizing complete dataset using Standard Scaler module as:

```
SS=StandardScaler()
X_train_scaled=SS.fit_transform(X_train)

SS=StandardScaler()
```

```
X_test_scaled=SS.fit_transform(X_test)
```

Converting entire data set into Polynomial feature of 2nd order

```
poly=PolynomialFeatures(degree=2)
X_train_poly=poly.fit_transform(X_train_scaled)

poly=PolynomialFeatures(degree=2)
X_test_poly=poly.fit_transform(X_test_scaled)
```

By assuming alpha as .1 training and evaluating R^2 value for Ridge Model

```
from sklearn.linear_model import Ridge
modl =Ridge(alpha=.1)
modl.fit(X_train_poly,Y_train)
R_sq=modl.score(X_train_poly,Y_train)
print('coefficient of determination by Ridge Model for Polynomial feature is:',
"{0:.3f}".format(R_sq))
```

coefficient of determination by Ridge Model for Polynomial feature is: **0.753**

By assuming alpha as .01 training and evaluating R^2 value for Ridge Model to improvise the model

```
from sklearn.linear_model import Ridge
modl =Ridge(alpha=.01)
modl.fit(X_train_poly,Y_train)
R_sq=modl.score(X_train_poly,Y_train)
print('coefficient of determination by Ridge Model for Polynomial feature is:',
"{0:.3f}".format(R_sq))
```

coefficient of determination by Ridge Model for Polynomial feature is: **0.753**

As we can see the model cannot be improved more in terms of alpha value and therefore we fix alpha=.01 as the best parameter and Predict the target value for test data set and compare it with Actual value of target variable using code

```
predict=pd.DataFrame()
predict['Predict']=modl.predict(X_test_poly)
predict[0:10]
Y_test[0:10]
```

Index	Predicted	Index	price
0	1.719428e+05	14848	235000.0
1	2.640591e+05	18796	200000.0
2	3.196329e+05	12350	285000.0
3	1.152050e+06	11619	1250000.0
4	3.558872e+05	12702	425000.0
5	6.898388e+05	8251	787888.0
6	3.201991e+05	20253	300000.0
7	3.601924e+05	15184	426500.0
8	6.337678e+05	20422	530000.0
9	5.066479e+05	4204	369000.0

As we can observe that there is more chance of improving the model as the Predicted and Actual value are not much close to each other and therefore by using different order of Polynomial features we train the model and predict and compare the value corresponding to test dataset.

Therefore, training the model with third order Polynomial features and analysing

```
poly=PolynomialFeatures(degree=3)
```

```
X_train_poly=poly.fit_transform(X_train_scaled)

poly=PolynomialFeatures(degree=3)
X_test_poly=poly.fit_transform(X_test_scaled)
```

By using alpha as .1 for training and evaluating R^2 value for Ridge Model

```
from sklearn.linear_model import Ridge
modl =Ridge(alpha=.1)
modl.fit(X_train_poly,Y_train)
R_sq=modl.score(X_train_poly,Y_train)
print('coefficient of determination by Ridge Model for Polynomial feature is:',
"{0:.3f}".format(R_sq))
```

coefficient of determination by Ridge Model for Polynomial feature is: **0.817,** which indicates that model has improved than the earlier Ridge Model

Predicting the target value for test data set and compare it with Actual value of target variable using code

```
predict=pd.DataFrame()
predict['Predict']=modl.predict(X_test_poly)
predict[0:10]
Y_test[0:10]
```

Index	*Predicted*	*Index*	*price*
0	229660.056455	14848	235000.0
1	147362.082614	18796	200000.0
2	247176.731931	12350	285000.0
3	965949.479403	11619	1250000.0
4	388196.564093	12702	425000.0
5	709665.022639	8251	787888.0
6	273783.138718	20253	300000.0
7	265758.441608	15184	426500.0
8	623007.802033	20422	530000.0
9	449124.541114	4204	369000.0

As we can observe that Predicted and Actual value of Price are much close than earlier, therefore we can say that model has fitted to dataset to a higher extent, but we can try to Predict and compare with Actual value for higher Polynomial order features to check for the best Polynomial order and Alpha values.

Therefore, now training the model with fourth order Polynomial features and analysing

```
poly=PolynomialFeatures(degree=4)
X_train_poly=poly.fit_transform(X_train_scaled)

poly=PolynomialFeatures(degree=4)
X_test_poly=poly.fit_transform(X_test_scaled)
```

By using alpha as .1 for training and evaluating R^2 value for Ridge Model

```
from sklearn.linear_model import Ridge
modl =Ridge(alpha=.1)
modl.fit(X_train_poly,Y_train)
R_sq=modl.score(X_train_poly,Y_train)
print('coefficient of determination by Ridge Model for Polynomial feature is:',
"{0:.3f}".format(R_sq))
```

coefficient of determination by Ridge Model for Polynomial feature is: **0.862** i.e model has improved in fitting the data set but we need to check Predicted value too for checking accuracy of model because model can **Overfit** dataset too.

Predicting the target value for test data set and compare it with Actual value of target variable using code

```
predict=pd.DataFrame()
predict['Predict']=modl.predict(X_test_poly)
predict[0:10]
Y_test[0:10]
```

Index	*Predicted*	*Index*	*price*
0	233400.944190	14848	235000.0
1	192111.248105	18796	200000.0
2	253217.493252	12350	285000.0
3	823802.936531	11619	1250000.0
4	376628.706626	12702	425000.0
5	685726.141418	8251	787888.0
6	283903.639587	20253	300000.0
7	366150.368953	15184	426500.0
8	599541.965795	20422	530000.0
9	449725.781234	4204	369000.0

As we can observe that Predicted and Actual value of Price are much close than earlier, therefore we can say that model has best fitted to dataset, but we can try even with higher Polynomial order too.

Therefore, now training the model with fifth order Polynomial features and analysing

```
poly=PolynomialFeatures(degree=5)
X_train_poly=poly.fit_transform(X_train_scaled)

poly=PolynomialFeatures(degree=5)
X_test_poly=poly.fit_transform(X_test_scaled)
```

By using alpha as .1 for training and evaluating R^2 value for Ridge Model

```
from sklearn.linear_model import Ridge
modl =Ridge(alpha=.1)
modl.fit(X_train_poly,Y_train)
R_sq=modl.score(X_train_poly,Y_train)
print('coefficient of determination by Ridge Model for Polynomial feature is:',
"{0:.3f}".format(R_sq))
```

coefficient of determination by Ridge Model for Polynomial feature is: **0.904** i.e model has improved in fitting the data set but we need to check Predicted value too for checking accuracy of model because model can **Overfit** dataset.

Therefore, Predicting the target value for test data set and compare it with Actual value of target variable using code

```
predict=pd.DataFrame()
predict['Predict']=modl.predict(X_test_poly)
predict[0:10]
Y_test[0:10]
```

Index	Predicted	Index	price
0	278863.002011	14848	235000.0
1	166986.287818	18796	200000.0
2	253407.626991	12350	285000.0
3	997162.904692	11619	1250000.0
4	420897.430122	12702	425000.0
5	710784.220833	8251	787888.0
6	278701.593345	20253	300000.0
7	311316.831617	15184	426500.0
8	583416.752799	20422	530000.0
9	454947.190041	4204	369000.0

CONCLUSION

Now, as we observe that model is **Overfitting** dataset, since the Actual value and Predicted value are moving far from each other in comparison to values obtained in earlier model and therefore the **best Polynomial order** is **4** and best **alpha value is .1** for training the model to given dataset

APPENDIX

Codes for Visualisation Plots and Data Exploratory Analysis

```
#1
clf()
sns.boxplot(x=df['waterfront'],y=df['price'])
plt.show()
plt.title('House Price dependency on Waterfront')

import scipy.stats as ss
Pearson_coef,p_value=ss.pearsonr(df['waterfront'],df['price'])
print("Pearson correlation coefficient:","{0:.1f}".format(Pearson_coef))
print("p_value :",p_value)

#2
clf()
sns.regplot(x=df['sqft_living'],y=df['price'])
plt.ylim(0,)
plt.title('House Price dependency on Flat area')

Pearson_coef,p_value=ss.pearsonr(df['sqft_living'],df['price'])
print("Pearson correlation coefficient:","{0:.1f}".format(Pearson_coef))
print("p_value :",p_value)

#3
clf()
sns.regplot(x=df['sqft_lot'],y=df['price'])
xlim(0,200000)
plt.ylim(0,)
plt.title('House Price dependency on Plot area')

Pearson_coef,p_value=ss.pearsonr(df['sqft_lot'],df['price'])
print("Pearson correlation coefficient:","{0:.1f}".format(Pearson_coef))
print("p_value :",p_value)

#4
clf()
sns.regplot(y=df['sqft_living'],x=df['sqft_lot'])
xlim(0,200000)
plt.ylim(0,)
xlabel('Plot area')
ylabel('Flat area')
plt.title('Relation of Flat area and Plot area')

Pearson_coef,p_value=ss.pearsonr(df['sqft_lot'],df['sqft_living'])
```

```
print("Pearson correlation coefficient:","{0:.1f}".format(Pearson_coef))
print("p_value :",p_value)
```

#5

```
clf()
Pdf=df[['bedrooms','bathrooms','waterfront','view']]
sns.boxplot('bedrooms','bathrooms',data=Pdf,hue='waterfront')
plt.ylim(0,)
xlim(0,11)
plt.title('Relation between Bedrooms and Bathrooms for different waterfront')
```

#6

```
clf()
sns.regplot(x=df['bedrooms'],y=df['price'])
plt.ylim(0,)
xlim(0,11)
plt.title('House Price dependency on Bedrooms')

Pearson_coef,p_value=ss.pearsonr(df['bedrooms'],df['price'])
print("Pearson correlation coefficient:","{0:.1f}".format(Pearson_coef))
print("p_value :",p_value)
```

#7

```
clf()
Pdf=df[['sqft_living','price','waterfront','view']]
g=sns.lmplot('sqft_living','price',data=Pdf,fit_reg=False,col='waterfront')
plt.ylim(0,)
plt.subplots_adjust(top=.9)
g.fig.suptitle('House Price dependency on Plot area for varied waterfront')
```

#8

```
clf()
Pdf=df[['sqft_living','price','waterfront','view']]
g=sns.FacetGrid(Pdf,col='view',row='waterfront',margin_titles=True)
g.map(plt.scatter,'sqft_living','price')
plt.subplots_adjust(top=.9)
g.fig.suptitle('House Price dependency on various factors')
```