# PROJECT TITLE:-
# PREDICTION OF HOUSE PRICES USING MACHINE LEARNING ALGORITHMS

**PREPARED BY**-SHASHI NANDAN PRASAD
**INSTITUTE-**NIT JAMSHEDPUR
**BRANCH**-PRODUCTION AND INDUSTRIAL ENGINEERING
**ROLL NO**-2017UGPI015

## CONTENTS

## 1. INTRODUCTION

The aim of this project is to predict the house prices i.e "price" which is the target variable.The dataset used here is of "kc-house-data" which is picked from kaggle repository.In this dataset, input variables are 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.

The ouput variable is "price" which is predicted on these parameters.

In this paper,I am going to explain the steps I had taken for predicting house prices using different Regression techniques and among them which is the best and optimum Regression technique for this prediction of house prices.

## 2.)PROBLEM STATEMENT

Our objective is to predict the house prices using various Machine Learning Algorithms by applying various Regression Techniques.

## 3.)MULTIPLE LINEAR REGRESSION MODEL

Regression analysis is widely used for prediction and forecasting. It

is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships.

Many techniques for carrying out regression analysis have been developed. Familiar methods such as linear regression and ordinary least squares regression are parametric, in that the regression function is defined in terms of a finite number of unknown parameters that are estimated from the data.

Multiple linear regression is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

The Formula for Multiple linear regression is-

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon$$

where, for i=n observations:

$y_i$=dependent variable

$x_i$=explanatory variables

$\beta_0$=y-intercept (constant term)

$\beta_p$=slope coefficients for each explanatory variable

$\epsilon$=the model's error term (also known as the residuals)

The multiple regression model is based on the following assumptions:

- There is a linear relationship between the dependent variables and the independent variables.
- The independent variables are not too highly correlated with each other.
- $y_i$ observations are selected independently and randomly from the population.
- Residuals should be normally distributed with a mean of 0 and variance $\sigma$.

## 4.POLYNOMIAL REGRESSION

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial**.** Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x).

In general, we can model the expected value of *y* as an *n*th degree polynomial, yielding the general polynomial regression model.

The mathematical expression for polynomial regression is:-

$$Y_i = a + b_1X_i + b_2X_i^2 + b_3X_i^3 + \ldots + b_kX_i^k + e_i$$

| Variable | Meaning |
|---|---|
| a | Constant |
| $b_j$ | The coefficient for the independent variable to the $j^{th}$ power |
| $e_i$ | Random error term |

## Assumptions for polynomial Regression is:-

the relationship between the dependent variable y and any independent variable $x_i$ is linear or curvilinear (specifically polynomial), the independent variables $x_i$ are independent of each other.

# 5.DECISION TREE

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs and utility.

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

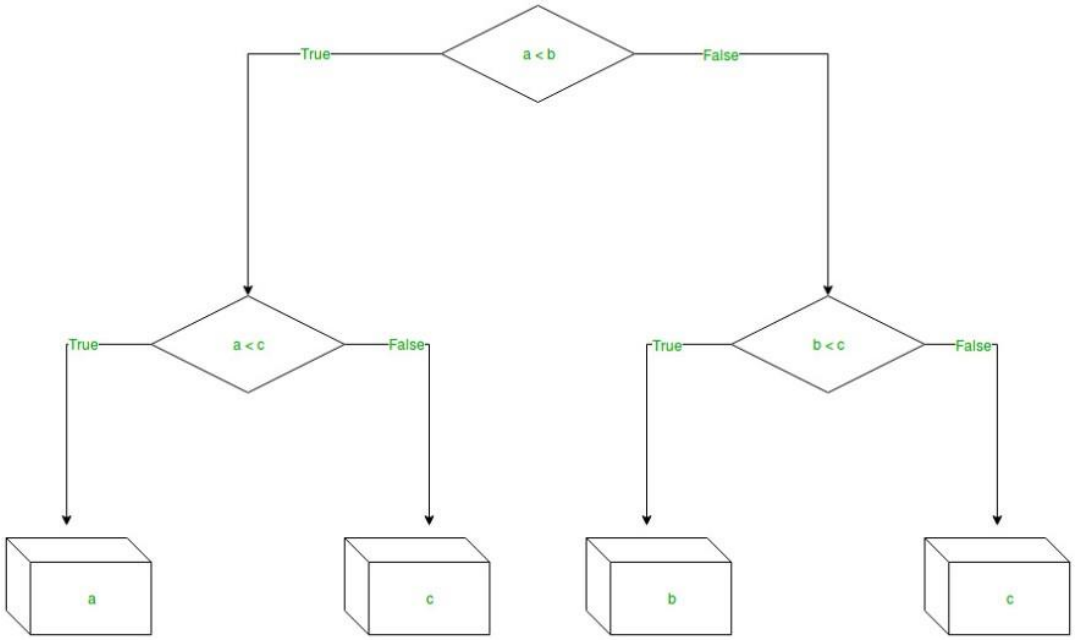



Fig: Decision Tree evaluating smallest of three numbers

## 6.RANDOM FOREST

Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

Random forests are used for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

It consists of an arbitrary number of simple trees, which are used to determine the final outcome. In the regression problem, their responses are averaged to obtain an estimate of the dependent variable.

## 7.K-NEAREST NEIGHBORS

K-nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition.

A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the distance functions.

**Distance functions**

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

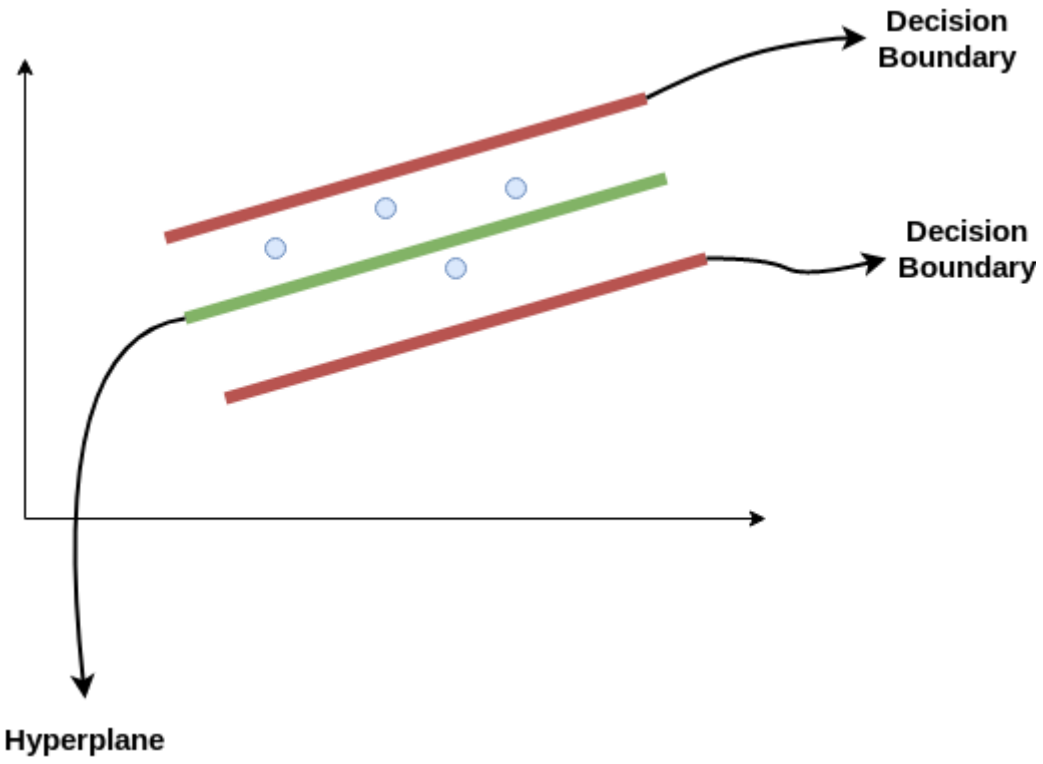
Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise; however, the compromise is that the distinct boundaries within the feature space are blurred.

## 8.)SUPPORT VECTOR REGRESSION

### The Idea Behind Support Vector Regression:-

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. So let's now dive deep and understand how SVR works actually.



Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we'll understand is what is the decision boundary (the danger red line above!). Consider these lines as being at any distance, say 'a', from the hyperplane. So, these are the lines that we draw at distance '+a' and '-a' from the hyperplane. This 'a' in the text is basically referred to as epsilon.

# 9.)XGBOOST REGRESSION

## System Features:-

The library provides a system for use in a range of computing environments, not least:

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithm to make best use of hardware.

## Algorithm Features

The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model. Some key algorithm implementation features include:

- **Sparse Aware** implementation with automatic handling of missing data values.
- **Block Structure** to support the parallelization of tree construction.
- **Continued Training** so that you can further boost an already fitted model on new data.

XGBoost is free open source software available for use under the permissive Apache-2 license.

## Why Use XGBoost?

The two reasons to use XGBoost are also the two goals of the project:

1. Execution Speed.
2. Model Performance.

## 1. XGBoost Execution Speed

Generally, XGBoost is fast. Really fast when compared to other implementations of gradient boosting.

Szilard Pafka performed some objective benchmarks comparing the performance of XGBoost to other implementations of gradient boosting and bagged decision trees. He wrote up his results in May 2015 in the blog post titled "Benchmarking Random Forest Implementations".

He also provides all the code on GitHub and a more extensive report of results with hard numbers.

### 2. XGBoost Model Performance

XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems.

The evidence is that it is the go-to algorithm for competition winners on the Kaggle competitive data science platform.

## 10.MACHINE LEARNING MODEL

For designing a model following steps were taken-

**a) Dataset Source(Kaggle):-**

b) https://www.kaggle.com/sawayaka/kc-house

**b)Data Preparation:** The data was combined in a single csv file for further processing. Our aim was to predict the "**prices**" of house, so it became our dependent variable. A correlation matrix was checked to find the relation among the parameters. The following variables were considered for model making-

a. Bedrooms
b. Bathrooms
c. sqft_living
d. sqft_lot
e. floors
f. waterfront
g. view
h. condition
i. grade
j. sqft_above
k. sqft_basement
l. yr_built
m. yr_renovated
n. zipcode
o. Lat

p. Long
q. sqft_living15
r. sqft_lot15

There were null values in the dataset which needed to be removed for an effective model. First, the null values of '**sqft_above'** were deleted along with entire row.
Now the data is prepared for splitting it into training and test data.

c) **Choosing a model:** Different models were trained on the train set and its R-squared value was found out:-

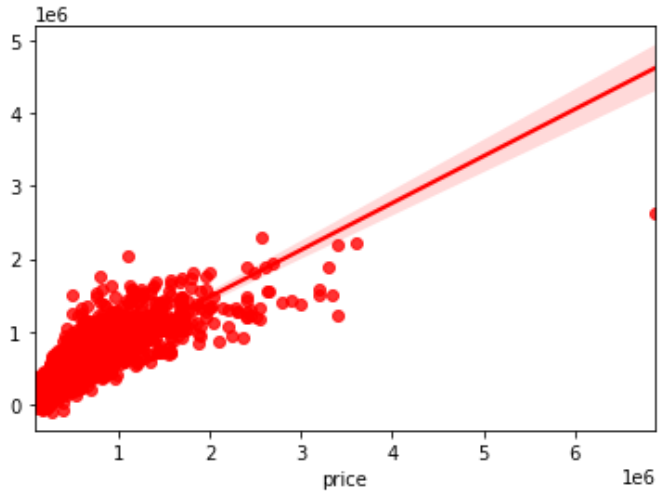
**a. Multiple Linear Regression**
b. $R^2$= 0.6486563096296043
**c. Polynomial Regression**
**$R^2$ = 0.8299003357092205**
**d. Decision Tree**
$R^2$ = 0.7166880349500175
**e. Random Forest**
$R^2$ = 0.8148145883428144
**f. K-Nearest Neighbors**
$R^2$=0.7234590333376416
**g. Support Vector Regression**
$R^2$=0.09105772694551195
**h. Xgboost Regression**
$R^2$=0.8125868031622345

## 11. RESULT AND CONCLUSION

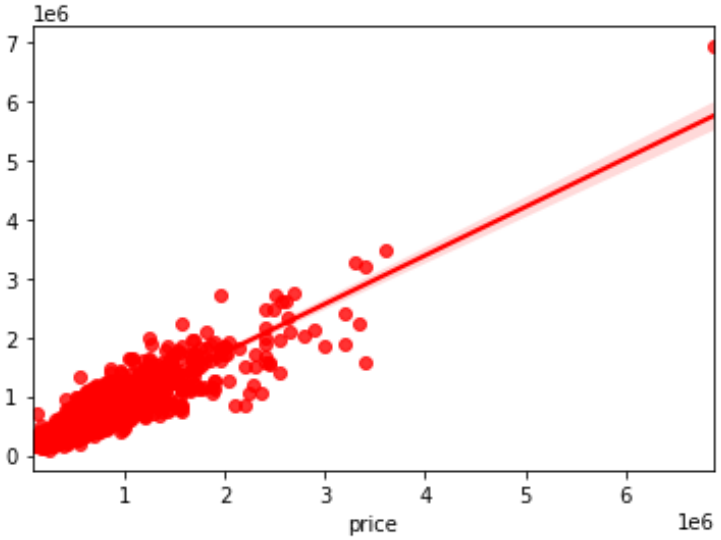
The best fit is **82.99%** for **Polynomial Regression,** which is the best regressive model as compared to other models. Hence, this model can be utilized for the prediction of house prices.

A graph of y_test vs y_pred was plotted for different models-

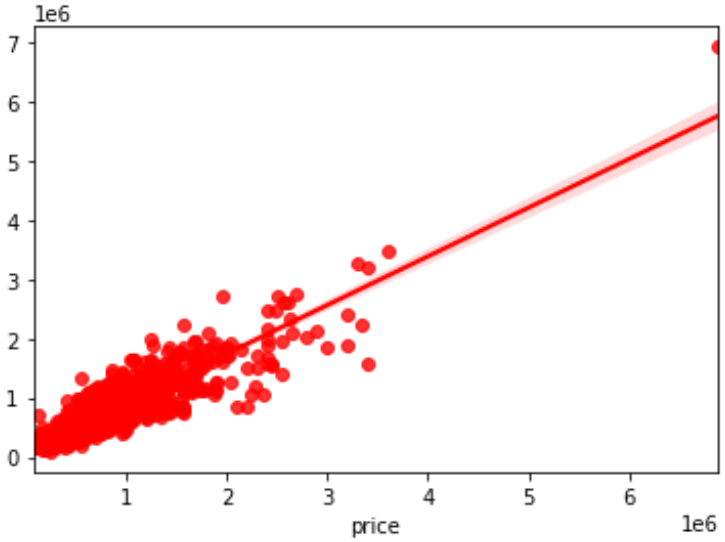
a. Multiple Linear Regression



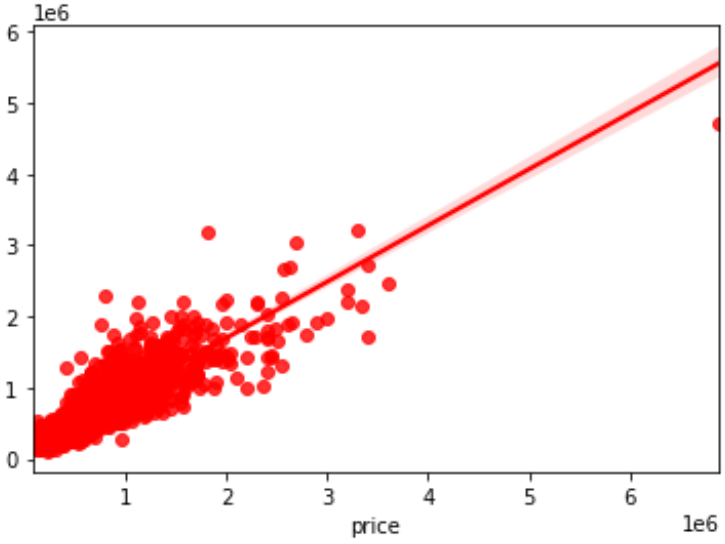

b. Polynomial Regression


1e6
7
6
5
4
3
2
1
0
1
2
3
4
5
6
price
1e6

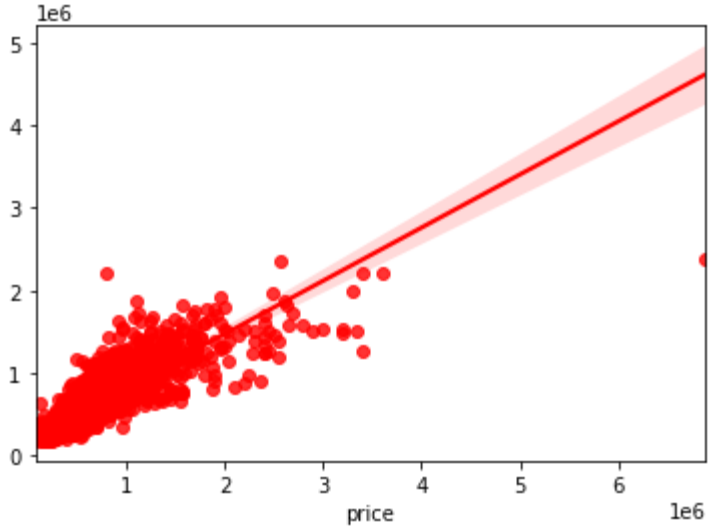
c. Decision Tree



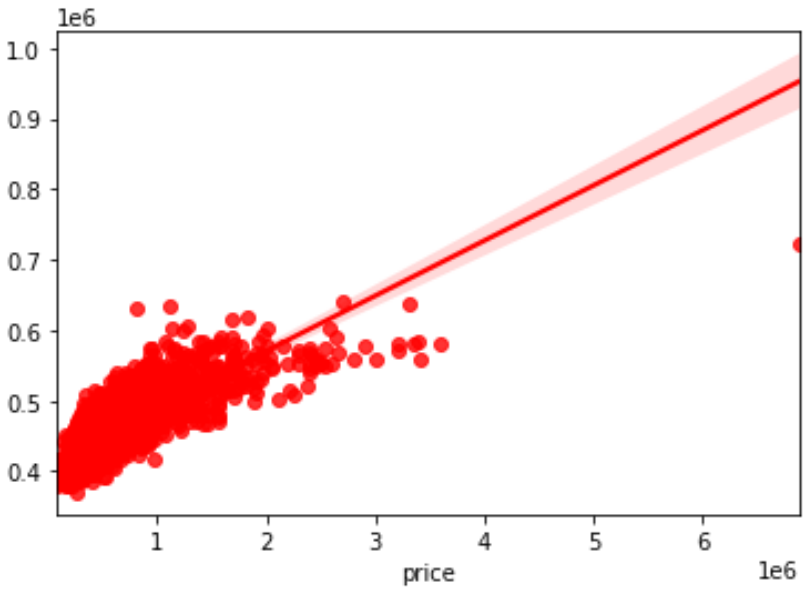

d. Random Forest



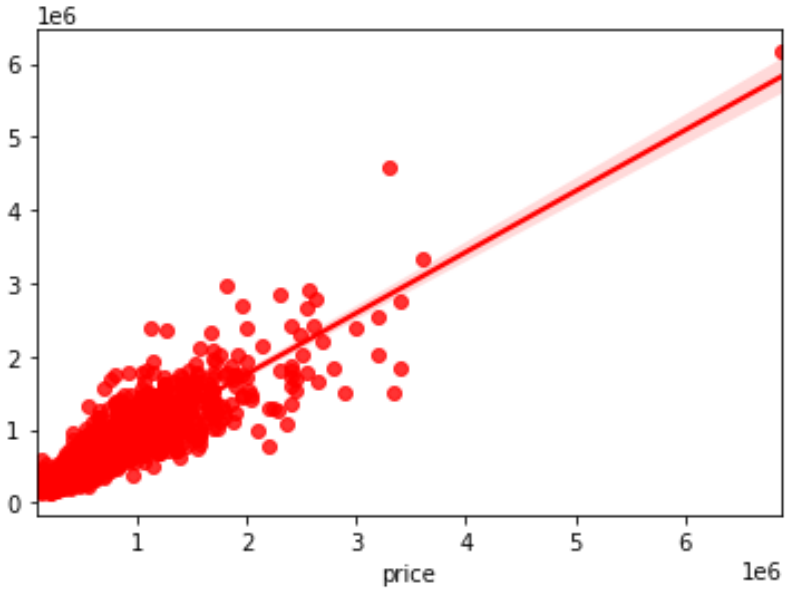

e. K-Nearest Neighbors




f. Support Vector Regression




g. Xgboost Regression




# 12.Annexure

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
pd.options.display.max_rows = 1000
pd.options.display.max_columns=None
df=pd.read_csv('../input/kc-house-data/kc_house_data.csv')
df
```

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7129300520 | 20141013T000000 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 | 0 | 0 | 3 | 7 | 1180.0 | 0 | 1955 | 0 | 98178 | 47.5112 | -122.257 | 1340 | 5650 |
| 1 | 6414100192 | 20141209T000000 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 | 0 | 0 | 3 | 7 | 2170.0 | 400 | 1951 | 1991 | 98125 | 47.7210 | -122.319 | 1690 | 7639 |
| 2 | 5631500400 | 20150225T000000 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 | 0 | 0 | 3 | 6 | 770.0 | 0 | 1933 | 0 | 98028 | 47.7379 | -122.233 | 2720 | 8062 |
| 3 | 2487200875 | 20141209T000000 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 | 0 | 0 | 5 | 7 | 1050.0 | 910 | 1965 | 0 | 98136 | 47.5208 | -122.393 | 1360 | 5000 |
| 4 | 1954400510 | 20150218T000000 | 510000.0 | 3 | 2.00 | 1680 | 8080 | 1.0 | 0 | 0 | 3 | 8 | 1680.0 | 0 | 1987 | 0 | 98074 | 47.6168 | -122.045 | 1800 | 7503 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 21608 | 263000018 | 20140521T000000 | 360000.0 | 3 | 2.50 | 1530 | 1131 | 3.0 | 0 | 0 | 3 | 8 | 1530.0 | 0 | 2009 | 0 | 98103 | 47.6993 | -122.346 | 1530 | 1509 |
| 21609 | 6600060120 | 20150223T000000 | 400000.0 | 4 | 2.50 | 2310 | 5813 | 2.0 | 0 | 0 | 3 | 8 | 2310.0 | 0 | 2014 | 0 | 98146 | 47.5107 | -122.362 | 1830 | 7200 |
| 21610 | 1523300141 | 20140623T000000 | 402101.0 | 2 | 0.75 | 1020 | 1350 | 2.0 | 0 | 0 | 3 | 7 | 1020.0 | 0 | 2009 | 0 | 98144 | 47.5944 | -122.299 | 1020 | 2007 |
| 21611 | 291310100 | 20150116T000000 | 400000.0 | 3 | 2.50 | 1600 | 2388 | 2.0 | 0 | 0 | 3 | 8 | 1600.0 | 0 | 2004 | 0 | 98027 | 47.5345 | -122.069 | 1410 | 1287 |
| 21612 | 1523300157 | 20141015T000000 | 325000.0 | 2 | 0.75 | 1020 | 1076 | 2.0 | 0 | 0 | 3 | 7 | 1020.0 | 0 | 2008 | 0 | 98144 | 47.5941 | -122.299 | 1020 | 1357 |

21613 rows × 21 columns

```
df=df.drop(["id","date", ], axis=1)
df
```

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 | 0 | 0 | 3 | 7 | 1180.0 | 0 | 1955 | 0 | 98178 | 47.5112 | -122.257 | 1340 | 5650 |
| 1 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 | 0 | 0 | 3 | 7 | 2170.0 | 400 | 1951 | 1991 | 98125 | 47.7210 | -122.319 | 1690 | 7639 |
| 2 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 | 0 | 0 | 3 | 6 | 770.0 | 0 | 1933 | 0 | 98028 | 47.7379 | -122.233 | 2720 | 8062 |
| 3 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 | 0 | 0 | 5 | 7 | 1050.0 | 910 | 1965 | 0 | 98136 | 47.5208 | -122.393 | 1360 | 5000 |
| 4 | 510000.0 | 3 | 2.00 | 1680 | 8080 | 1.0 | 0 | 0 | 3 | 8 | 1680.0 | 0 | 1987 | 0 | 98074 | 47.6168 | -122.045 | 1800 | 7503 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 21608 | 360000.0 | 3 | 2.50 | 1530 | 1131 | 3.0 | 0 | 0 | 3 | 8 | 1530.0 | 0 | 2009 | 0 | 98103 | 47.6993 | -122.346 | 1530 | 1509 |
| 21609 | 400000.0 | 4 | 2.50 | 2310 | 5813 | 2.0 | 0 | 0 | 3 | 8 | 2310.0 | 0 | 2014 | 0 | 98146 | 47.5107 | -122.362 | 1830 | 7200 |
| 21610 | 402101.0 | 2 | 0.75 | 1020 | 1350 | 2.0 | 0 | 0 | 3 | 7 | 1020.0 | 0 | 2009 | 0 | 98144 | 47.5944 | -122.299 | 1020 | 2007 |
| 21611 | 400000.0 | 3 | 2.50 | 1600 | 2388 | 2.0 | 0 | 0 | 3 | 8 | 1600.0 | 0 | 2004 | 0 | 98027 | 47.5345 | -122.069 | 1410 | 1287 |
| 21612 | 325000.0 | 2 | 0.75 | 1020 | 1076 | 2.0 | 0 | 0 | 3 | 7 | 1020.0 | 0 | 2008 | 0 | 98144 | 47.5941 | -122.299 | 1020 | 1357 |

21613 rows × 19 columns

#Checking of missing values
df.isnull().sum()

```
price          0
bedrooms       0
bathrooms      0
sqft_living    0
sqft_lot       0
floors         0
waterfront     0
view           0
condition      0
grade          0
sqft_above     2
sqft_basement  0
yr_built       0
yr_renovated   0
zipcode        0
lat            0
long           0
sqft_living15  0
sqft_lot15     0
dtype: int64
```

df = df.dropna(subset=["sqft_above"])

# Correlation Matrix

corr_matrix = df.corr()
corr_matrix

| | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| price | 1.000000 | 0.308366 | 0.525150 | 0.702055 | 0.089661 | 0.256814 | 0.266371 | 0.397299 | 0.036379 | 0.667434 | 0.605567 | 0.323842 | 0.054006 | 0.126436 | -0.053182 | 0.307010 | 0.021613 | 0.585377 | 0.082448 |
| bedrooms | 0.308366 | 1.000000 | 0.515974 | 0.576763 | 0.031710 | 0.175418 | -0.006581 | 0.079537 | 0.028433 | 0.356998 | 0.477616 | 0.303251 | 0.154248 | 0.018844 | -0.152717 | -0.008950 | 0.129516 | 0.391670 | 0.029252 |
| bathrooms | 0.525150 | 0.515974 | 1.000000 | 0.754684 | 0.087730 | 0.500712 | 0.063743 | 0.187735 | -0.124917 | 0.664981 | 0.685363 | 0.283737 | 0.505968 | 0.050733 | -0.203825 | 0.024619 | 0.222987 | 0.568626 | 0.087163 |
| sqft_living | 0.702055 | 0.576763 | 0.754684 | 1.000000 | 0.172841 | 0.354048 | 0.103829 | 0.284647 | -0.058689 | 0.762727 | 0.876644 | 0.434925 | 0.318066 | 0.055377 | -0.199342 | 0.052530 | 0.240187 | 0.756440 | 0.183301 |
| sqft_lot | 0.089661 | 0.031710 | 0.087730 | 0.172841 | 1.000000 | -0.005206 | 0.021602 | 0.074705 | -0.008951 | 0.113617 | 0.183511 | 0.015301 | 0.053061 | 0.007640 | -0.129583 | -0.085673 | 0.229519 | 0.144605 | 0.718556 |
| floors | 0.256814 | 0.175418 | 0.500712 | 0.354048 | -0.005206 | 1.000000 | 0.023695 | 0.029432 | -0.263808 | 0.458208 | 0.523899 | -0.245634 | 0.489361 | 0.006330 | -0.059181 | 0.049628 | 0.125446 | 0.279907 | -0.011275 |
| waterfront | 0.266371 | -0.006581 | 0.063743 | 0.103829 | 0.021602 | 0.023695 | 1.000000 | 0.401857 | 0.016655 | 0.082775 | 0.072074 | 0.080618 | -0.026172 | 0.092883 | 0.030283 | -0.014270 | -0.041913 | 0.086463 | 0.030702 |
| view | 0.397299 | 0.079537 | 0.187735 | 0.284647 | 0.074705 | 0.029432 | 0.401857 | 1.000000 | 0.045995 | 0.251320 | 0.167648 | 0.277051 | -0.053474 | 0.103912 | 0.084819 | 0.006172 | -0.078408 | 0.280440 | 0.072569 |
| condition | 0.036379 | 0.028433 | -0.124917 | -0.058689 | -0.008951 | -0.263808 | 0.016655 | 0.045995 | 1.000000 | -0.144647 | -0.158206 | 0.174273 | -0.361384 | -0.060617 | 0.002967 | -0.014965 | -0.106453 | -0.092795 | -0.003397 |
| grade | 0.667434 | 0.356998 | 0.664981 | 0.762727 | 0.113617 | 0.458208 | 0.082775 | 0.251320 | -0.144647 | 1.000000 | 0.755924 | 0.168375 | 0.446958 | 0.014412 | -0.184842 | 0.114102 | 0.198349 | 0.713197 | 0.119243 |
| sqft_above | 0.605567 | 0.477616 | 0.685363 | 0.876644 | 0.183511 | 0.523899 | 0.072074 | 0.167648 | -0.158206 | 0.755924 | 1.000000 | -0.051976 | 0.423915 | 0.023283 | -0.261192 | -0.000810 | 0.343800 | 0.731871 | 0.194048 |
| sqft_basement | 0.323842 | 0.303251 | 0.283737 | 0.434925 | 0.015301 | -0.245634 | 0.080618 | 0.277051 | 0.174273 | 0.168375 | -0.051976 | 1.000000 | -0.133195 | 0.071365 | 0.075076 | 0.110546 | -0.144904 | 0.200341 | 0.017292 |
| yr_built | 0.054006 | 0.154248 | 0.505968 | 0.318066 | 0.053061 | 0.489361 | -0.026172 | -0.053474 | -0.361384 | 0.446958 | 0.423915 | -0.133195 | 1.000000 | -0.224913 | -0.346885 | -0.148069 | 0.409325 | 0.326214 | 0.070936 |
| yr_renovated | 0.126436 | 0.018844 | 0.050733 | 0.055377 | 0.007640 | 0.006330 | 0.092883 | 0.103912 | -0.060617 | 0.014412 | 0.023283 | 0.071365 | -0.224913 | 1.000000 | 0.064352 | 0.029408 | -0.068378 | -0.002675 | 0.007849 |
| zipcode | -0.053182 | -0.152717 | -0.203825 | -0.199342 | -0.129583 | -0.059181 | 0.030283 | 0.084819 | 0.002967 | -0.184842 | -0.261192 | 0.075076 | -0.346885 | 0.064352 | 1.000000 | 0.267066 | -0.564062 | -0.279016 | -0.147230 |
| lat | 0.307010 | -0.008950 | 0.024619 | 0.052530 | -0.085673 | 0.049628 | -0.014270 | 0.006172 | -0.014965 | 0.114102 | -0.000810 | 0.110546 | -0.148069 | 0.029408 | 0.267066 | 1.000000 | -0.135492 | 0.048874 | -0.086408 |
| long | 0.021613 | 0.129516 | 0.222987 | 0.240187 | 0.229519 | 0.125446 | -0.041913 | -0.078408 | -0.106453 | 0.198349 | 0.343800 | -0.144904 | 0.409325 | -0.068378 | -0.564062 | -0.135492 | 1.000000 | 0.334586 | 0.254448 |
| sqft_living15 | 0.585377 | 0.391670 | 0.568626 | 0.756440 | 0.144605 | 0.279907 | 0.086463 | 0.280440 | -0.092795 | 0.713197 | 0.731871 | 0.200341 | 0.326214 | -0.002675 | -0.279016 | 0.048874 | 0.334586 | 1.000000 | 0.183188 |
| sqft_lot15 | 0.082448 | 0.029252 | 0.087163 | 0.183301 | 0.718556 | -0.011275 | 0.030702 | 0.072569 | -0.003397 | 0.119243 | 0.194048 | 0.017292 | 0.070936 | 0.007849 | -0.147230 | -0.086408 | 0.254448 | 0.183188 | 1.000000 |

# **Feature Selection**

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
X = df.iloc[:,1:]  #independent columns
Y = df.iloc[:,0] #target column
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=f_regression, k=10)
fit = bestfeatures.fit(X,Y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Features','Score']  #naming the
dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
         Features         Score
2     sqft_living  21002.360772
8           grade  17359.011804
9      sqft_above  12512.889029
16  sqft_living15  11264.733778
1       bathrooms   8228.716198
6            view   4050.218305
10  sqft_basement   2531.734429
0        bedrooms   2270.708755
14            lat   2248.709114
5      waterfront   1650.332397
```

**x = df[["sqft_living", "grade", "sqft_above", "sqft_living15", "bathrooms", "view", "sqft_basement", "bedrooms", "lat","waterfront" ]]**
**y = df[["price"]]**

# Initialise the Scaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = scaler.fit_transform(x)

# Splitting the Data set in training and test data
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=0)
from sklearn.preprocessing import StandardScaler
```

## # Multiple Linear Regression

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

#R-Square value

from sklearn.metrics import r2_score
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

print(regressor.intercept_)
print(regressor.coef_)
```

```
[539552.95503109]
[[ 1.22076261e+18  9.36982316e+04 -1.10074394e+18  2.96516830e+03
  -1.23387743e+04  4.97145260e+04 -5.88152046e+17 -2.22580319e+04
   9.24187467e+04  5.52890668e+04]]
```

```
# Original vs Predicted
import seaborn as sns
sns.regplot(y_test, y_pred, color = 'red')
plt.savefig('mlr.jpg')
```

## #Polynomial Regression

```
#fitting polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg=PolynomialFeatures(degree=4)
x_poly=poly_reg.fit_transform(x)
lin_reg_2=LinearRegression()
lin_reg_2.fit(x_poly,y)

y_pred=lin_reg_2.predict(poly_reg.fit_transform(x_test))
#R-Square value
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

# Original vs Predicted
sns.regplot(y_test,y_pred,color = 'red')
plt.savefig('pr.jpg')
```

```
y_pred_all = lin_reg_2.predict(poly_reg.fit_transform(x))
df["y_pred_all"] = y_pred_all
print(df.iloc[:,[0,-1]])
```

```
          price     y_pred_all
0      221900.0  325965.581202
1      538000.0  595530.815943
2      180000.0  276970.594019
3      604000.0  378686.541559
4      510000.0  569668.438379
...         ...            ...
21608  360000.0  499856.621973
21609  400000.0  506833.488550
21610  402101.0  355485.893702
21611  400000.0  436262.112329
21612  325000.0  355170.114893

[21611 rows x 2 columns]
```

**print ("Coefficient of Determination : ",r2_score(df['price'],y_pred_all))**

```
Coefficient of Determination :  0.8426432977552519
```

## #Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
regressor_dt = DecisionTreeRegressor(max_depth=10)
regressor_dt.fit(x_train,y_train)
y_pred = regressor_dt.predict(x_test)

#R-Square value
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

# Original vs Predicted
sns.regplot(y_test, y_pred, color = 'red')
plt.savefig('dtr.jpg')
```

## #Random Forest

```
from sklearn.ensemble import RandomForestRegressor

regressor_rfr = RandomForestRegressor(n_estimators = 10)
regressor_rfr.fit(x_train,y_train)
y_pred = regressor_rfr.predict(x_test)

#R-Square value
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

# Original vs Predicted
sns.regplot(y_test, y_pred, color = 'red')
plt.savefig('rfr.jpg')
```

**#K-Nearest Neighbors**

```
from sklearn.neighbors import KNeighborsRegressor

# Create KNN regressor
knn = KNeighborsRegressor(n_neighbors = 50)
# Fit the classifier to the data
knn.fit(x_train, y_train)

y_pred = knn.predict(x_test)

#R-Square value
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

# Original vs Predicted
sns.regplot(y_test, y_pred, color = 'red')
```

```
plt.savefig('knnr.jpg')
```

**#Support Vector Regression**

```
#fitting svr to the dataset
from sklearn.svm import SVR
regressor_svr= SVR(kernel="linear")
regressor_svr.fit(x_train,y_train)
y_pred = regressor_svr.predict(x_test)

#R-Square value
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

# Original vs Predicted
sns.regplot(y_test, y_pred, color = 'red')
plt.savefig('svr.jpg')
```

**#Xgboost Regression**

```
from xgboost import XGBRegressor
regressor_xgb=XGBRegressor()
regressor_xgb.fit(x_train,y_train)

y_pred = regressor_xgb.predict(x_test)

#R-Square value
coefficient_of_dermination = r2_score(y_test, y_pred)
print("Score: {}".format(coefficient_of_dermination))

# Original vs Predicted
```

```
sns.regplot(y_test, y_pred, color = 'red')
plt.savefig('xgbr.jpg')
```