Fundamentals Of Machine Learning

PBL

[20B12CS331]

Analysis Of Regression Models For House Sales Price Prediction

Members

Pranav Gupta 19803021(B13)

Parish Bindal 19803022(B13)

Khushbo Kumari 19803003(B13)

Abhinav Verma 19103223(B7)

Submitted to

Dr Parul Aggarwal

Problem Statement

Wrong estimate or prediction of real estate/ housing properties leads to wrong budgeting and delay of payment for both customers and brokers. Hence the **problem is to have an accurate method or model to have justified and correct pricing of real estate**.

Motivation

In real estate, there is usually a discrepancy between real pricing and expected pricing. As a result, it's preferable to utilise precise and best machine learning models based on historical records to forecast sales and purchase pricing. But various models can be used for predicting the price hence a comparative examination of the various models is required. Thus we planned to compile a short simulation on **Comparative Analysis Of Various Models For House Sales Price Prediction.**

Literature

House Price Index (HPI)-The House Price Index (HPI) is a broad measure of the movement of single-family property prices in the United States. In addition to serving as a trend indicator, it also serves as an analytical tool for estimating changes in mortgage default, prepayment, and housing affordability rates [1].

Housing Price Prediction via Improved Machine Learning Techniques-Before building models, the data should be processed accordingly so that the models could learn the patterns more efficiently. Specifically, numerical values were standardised, while categorical values were one-hot-encoded[2].

Housing Price Prediction Based on Multiple Linear Regression-According to economics principles, the market price of properties is attained when the demand and supply curves intersect with each other, which is subject to various factors, both subjectively and objectively[3].

Models used/proposed(Flow Diagram)

Linear Regression:

Linear Regression is a supervised machine learning algorithm. It carries out a regression task. Based on independent variables, regression models a goal prediction value. It is mostly used in forecasting and determining the link between variables. Different regression models differ in terms of the type of relationship they evaluate between dependent and independent variables and the number of independent variables they employ.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

Hypothesis function for Linear Regression:

$$y = \theta_1 + \theta_2 . x$$

While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.

$$\theta_1 : intercept$$

$$\theta_2 : coefficient\ of\ x$$

How can the θ1 and θ2 values are updated to acquire the greatest fit line?

Cost Function (J):

The model seeks to predict the y value in such a way that the **error difference between the predicted and true value is as small as possible** by reaching the best-fit regression line. As a result, it is critical to update the θ1 and θ2 values in order to find the ideal value that minimises the difference between the predicted y value (pred) and the true y value (y).

$$minimize \frac{1}{n}\sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n}\sum_{i=1}^{n} (pred_i - y_i)^2$$

Cost function(J) of Linear Regression is the **Root Mean Squared Error (RMSE)** between predicted y value (pred) and true y value (y).

Least Squares Method

$$\textbf{Prediction Equation} \quad \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

$$\textbf{Slope} \quad \hat{\beta}_1 = \frac{SS_{xy}}{SS_{xx}} = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}$$

$$\textbf{y-intercept} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Gradient Descent Method

$$\text{repeat until convergence } \{$$
$$\left.\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)} \end{aligned}\right] \text{update } \theta_0 \text{ and } \theta_1 \text{ simultaneously}$$
$$\}$$

The model employs Gradient Descent to update θ1 and θ2 values in order to minimise Cost function (minimising RMSE value) and achieve the best fit line. The goal is to start with random θ1 and θ2 numbers and then update the values iteratively until the minimal cost is reached.

Multivariate Regression:

Multivariate Regression comes under the class of Supervised Learning Algorithms i.e when we are provided with a training dataset. In the case of multivariate linear regression, the output value is dependent on multiple input values. The relationship between input values, the format of different input values and the range of input values plays an important role in linear model creation and prediction.

For multiple input values, the hypothesis function will look like,

$$y = \theta_1 + \theta_2 * x_2 + \theta_3 * x_3 + \ldots\ldots\ldots\ldots\ldots \theta_n * x_n$$

where $x_2, x_3 \ldots x_n$ are multiple feature values

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

hθ(X)=Xθ

If we consider the house price example then the factors affecting its price like house size, no of bedrooms, location etc are nothing but input variables of the above hypothesis function.

Cost Function:

A hypothesis is a predicted value of the response variable represented by h(x). Cost function defines the cost for wrongly predicting hypotheses. It should be as small as possible. We choose a hypothesis function as a linear combination of features x.

$$J = \frac{1}{2n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Gradient Descent:

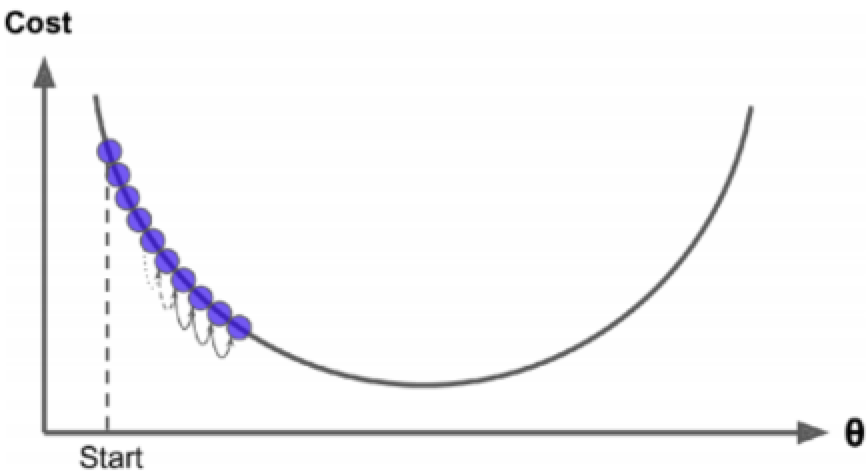
The model employs Gradient Descent to update θ1 and θ2 values in order to minimise Cost function (minimising RMSE value) and achieve the best fit line. The goal is to start with random θ1 and θ2 numbers and then update the values iteratively until the minimal cost is reached.

$$\text{repeat until convergence: } \{$$
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n}$$
$$\}$$

where, **X0 = 1**

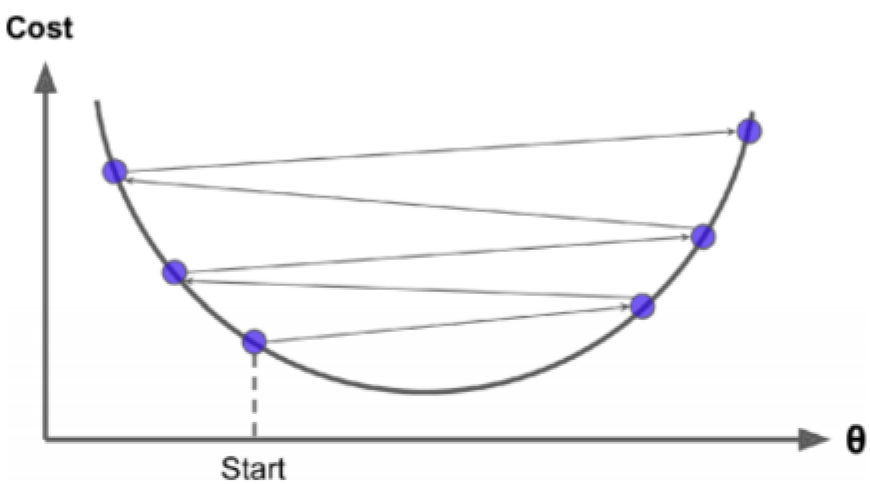
α = Learning rate

If α is too small, the gradient descent can be slow. It will require more iterations (hence time) to reach the minimum.

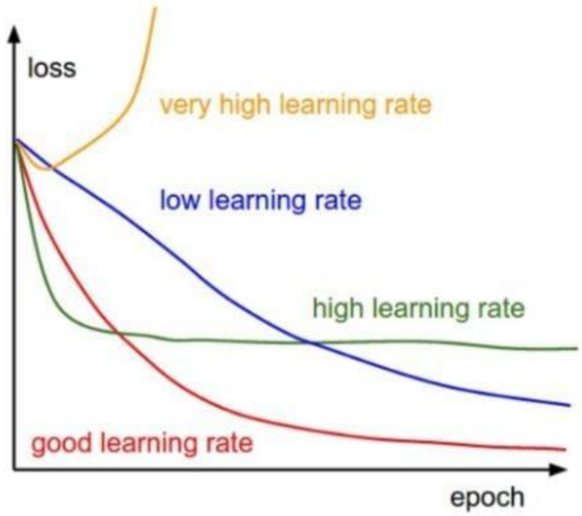


The learning rate is too small

If α is too large, the gradient descent can overshoot the minimum. In that case, it may fail to converge or even diverge. If this happens when you run your Python code, NaN values are returned for θ0 , θ1. In that case, you should decrease the value of α and run the algorithm again.



The learning rate is too large



RSS(Residual Sum of Squares):

The residual sum of squares (RSS), also known as the sum of squared residuals (SSR) or the sum of the squared estimate of errors (SSE), is the sum of the squares of residuals (deviations predicted from actual empirical values of data). It is a measure of the discrepancy between the data and an estimation model, such as linear regression. A small RSS indicates a tight fit of the model to the data.

$$\mathrm{RSS} = \sum_{i=1}^{n}(y_i - f(x_i))^2$$

R-squared(coefficient of determination):

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination. Its value lies in the range (0,1). The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.

Residual Sum of Squares

$$SS_{\mathrm{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

Total Sum of Squares

$$SS_{\mathrm{tot}} = \sum_i (y_i - \bar{y})^2$$

$$R^2 = 1 - \frac{SS_{\mathrm{res}}}{SS_{\mathrm{tot}}}$$

Dataset(https://www.kaggle.com/search?q=house+price+prediction)

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0

condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
3	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
3	7	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639
3	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
5	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
3	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503

Implementations

```
#linear regression - least square method
def estimate_coef(x, y):

    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)

    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    b_1 = SS_xy / SS_xx #slope
    b_0 = m_y - b_1*m_x #intercept

    return (b_0, b_1)
```

This function estimates the coefficient for linear regression using the least square method .

```
def get_rss(y_pred, y):
    rss = np.sum((y_pred - y) ** 2)
    return rss
#TSS is the total sum of squares, sum of y_outcome - mean_y
#RSS is the residual sum of squares, sum of y_outcome - predicted_y
#R² = (TSS - RSS) / TSS
def get_r2(y_pred, y):
    rss = np.sum((y_pred - y) ** 2)
    tss = np.sum((y - np.mean(y))** 2)
    r2=(tss-rss)/tss
    return r2
```

The residual sum of squares (RSS) measures the level of variance in the error term, or residuals, of a regression model. The smaller the residual sum of squares, the better your model fits your data; the greater the residual sum of squares, the poorer your model fits your data.

```python
#linear regression gradient descent
def Linear_GradientDescent(x, y, b1, b0, learning_rate, epochs):
    cost_list = [0] * epochs

    m=len(y)
    for epoch in range(epochs):
        z =x*b1 + b0
        loss = z - y

        weight_gradient = x.T.dot(loss) / m
        bias_gradient = np.sum(loss) / m

        b1 = b1 - learning_rate*weight_gradient
        b0 = b0 - learning_rate*bias_gradient

        cost = (np.sum(((x*b1 + b0) - y) ** 2)) / (2*m)
        cost_list[epoch] = cost

        if (epoch%100==0):
            print(f"Cost at epoch {epoch} is : ","{:.2e}".format(cost))

    plt.plot(cost_list)
    plt.show()
    return b0,b1
def linear_reg_sklearn(feature):
    y,x= data["price"],data[feature]
    x=x.to_numpy()
    y=y.to_numpy()
    x=x.reshape(-1,1)
    y=y.reshape(-1,1)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_st
ate=100)
    reg = LinearRegression().fit(x_train,y_train)
    b=[]
    slope=reg.coef_
    intercept=reg.intercept_
    b.append(intercept)
    b.append(slope)
    y_pred=reg.predict(x_test)
    print(f"slope : {slope}")
    print(f"intercept : {intercept}")
    plot_regression_line(x_train,y_train,b)
```

```
    rss=get_rss(y_pred, y_test)
    rss = "{:e}".format(rss)
    print(f"RSS :  {rss}\n")
    r2=r2_score(y_test,y_pred)
    print(f"R2 :  {r2}\n")
```

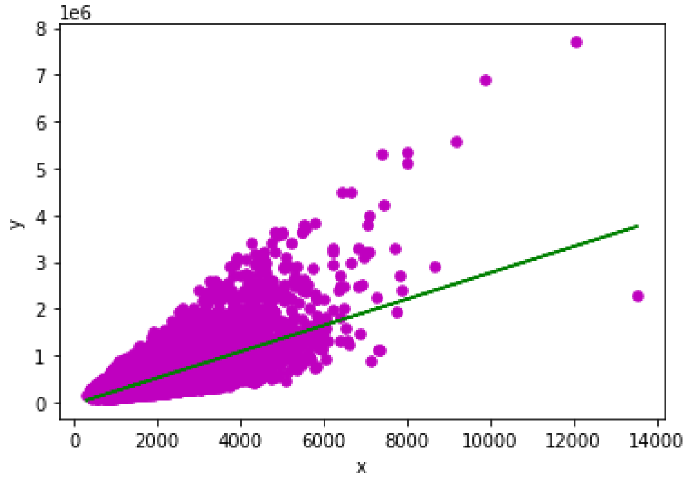
This function is used to find w and b where w : θ (1 to num of features) and b : theta naught

```
def Multivariate_GradientDescent(x, y, w, b, learning_rate, epochs):
    cost_list = [0] * epochs

    m=len(y)
    for epoch in range(epochs):
        z =np.dot(x,w) + b
        loss = z - y

        weight_gradient = x.T.dot(loss) / m
        bias_gradient = np.sum(loss) / m

        w = w - learning_rate*weight_gradient
        b = b - learning_rate*bias_gradient

        cost = Multivariate_CostFunction(x, y, w, b,m)
        cost_list[epoch] = cost

        if (epoch%200==0):
            print(f"Cost at epoch {epoch} is : ","{:.2e}".format(cost))

    #plt.plot(np.arrange(epochs),cost_list)
    plt.plot(cost_list)
    plt.show()

    return w, b
```

Results

1. Linear Regression - Least Squares Method

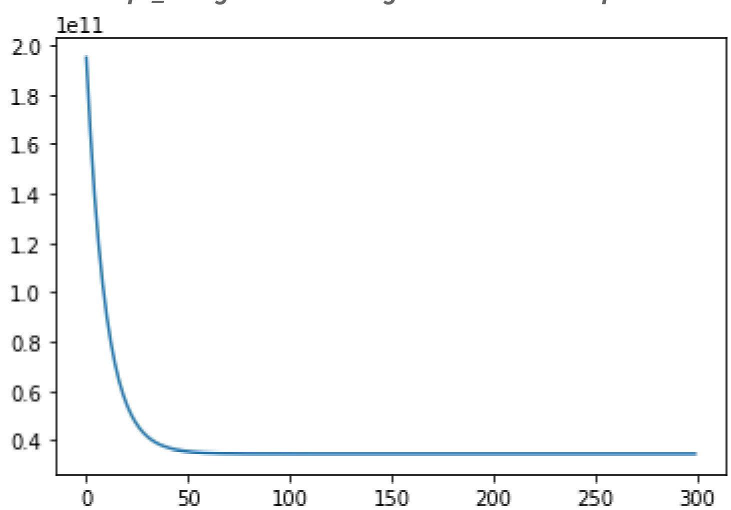
```
Estimated coefficients:
intercept = -42628.97651509475
slope = 280.68541678774295
```

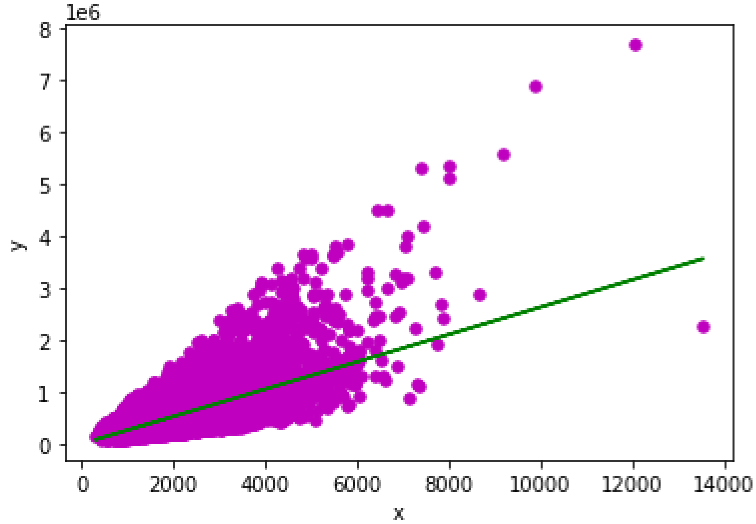



```
RSS : 2.822317e+14

R2 : 0.5155709745445146
```

2. Linear Regression - Gradient Descent Optimization

Feature: sqft_living ***Learning rate: 1e-8*** ***epochs: 300***

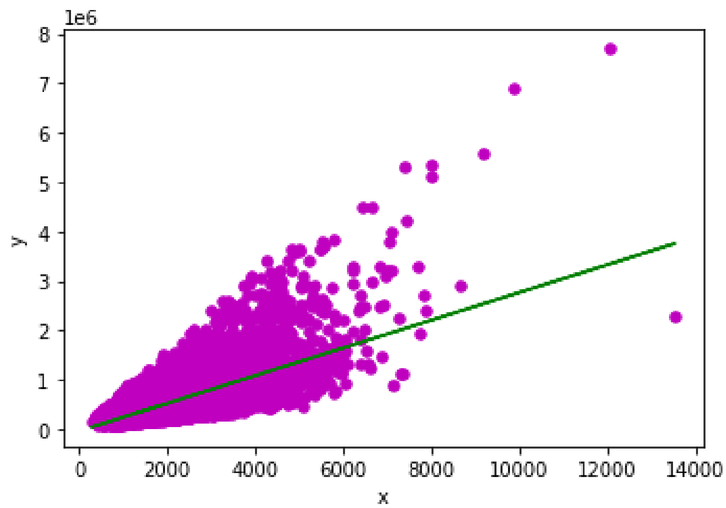





```
RSS : 2.838211e+14

R2 : 0.5128428942361627
```

3. Linear Regression - sklearn

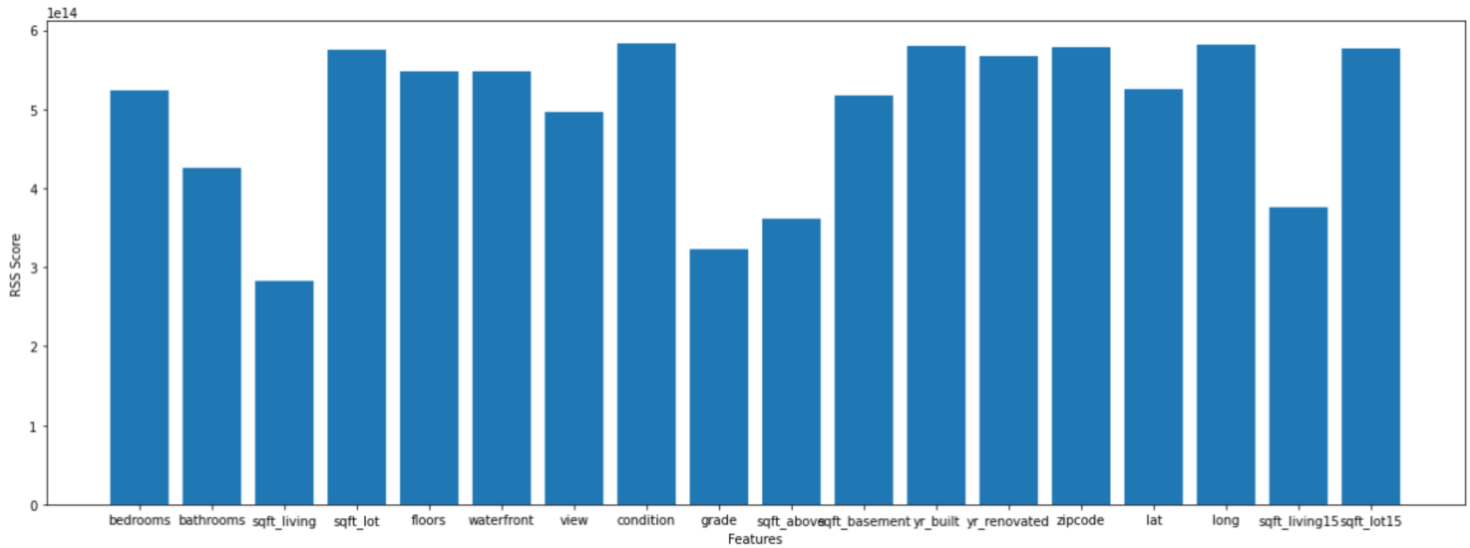
```
slope : [[280.68541679]]
intercept : [-42628.97651509]
```

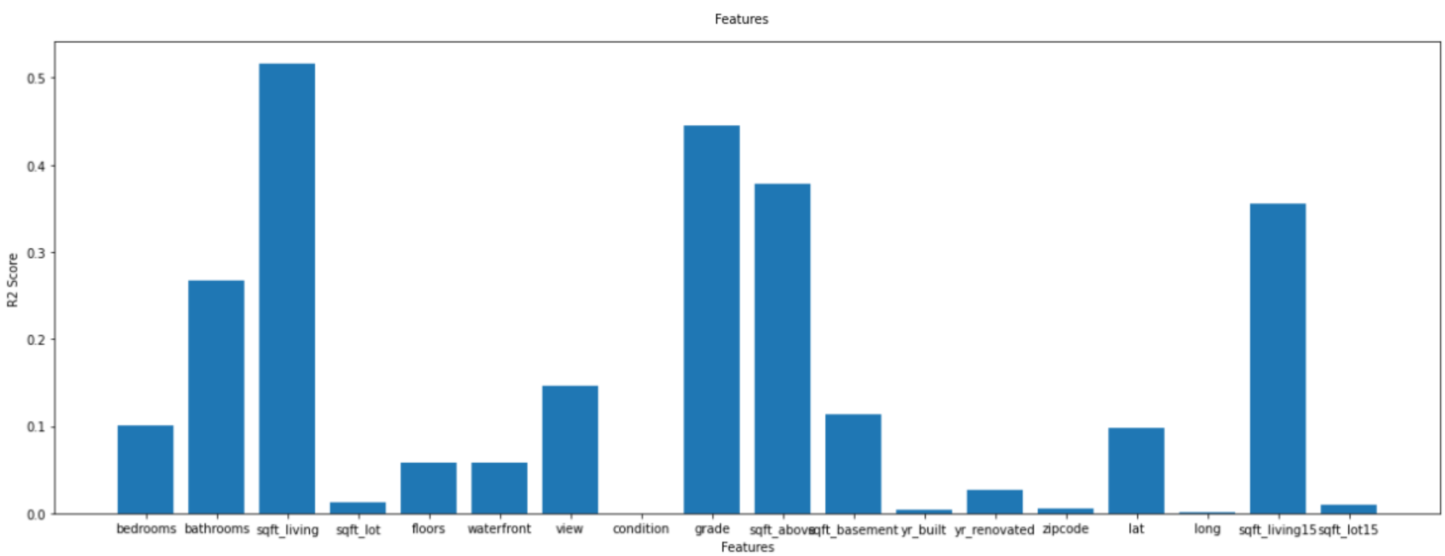



```
RSS : 2.822317e+14

R2 : 0.5155709745445147
```

4. Analysis of scores with different features - Linear Regression

```
----------LINEAR REGRESSION - Analysis of scores with different features---------
```

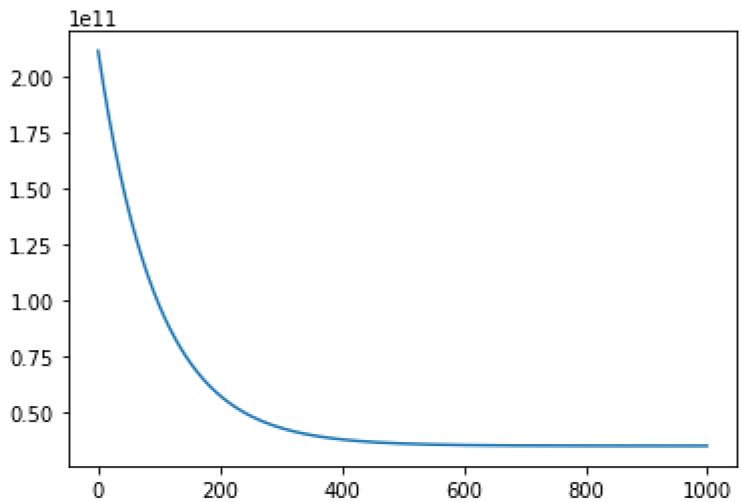




5. Multivariate Linear Regression - Gradient Descent Optimization

Learning rate : 1e-9 ***epochs : 1000***

```
Enter feature names :
sqft_living
grade
bedrooms
Cost at epoch 0 is :  2.11e+11
Cost at epoch 100 is :  9.77e+10
Cost at epoch 200 is :  5.72e+10
Cost at epoch 300 is :  4.27e+10
Cost at epoch 400 is :  3.76e+10
Cost at epoch 500 is :  3.57e+10
Cost at epoch 600 is :  3.51e+10
Cost at epoch 700 is :  3.48e+10
Cost at epoch 800 is :  3.48e+10
Cost at epoch 900 is :  3.47e+10
```




```
RSS : 2.837455e+14

R2 : 0.5129725660158259
```

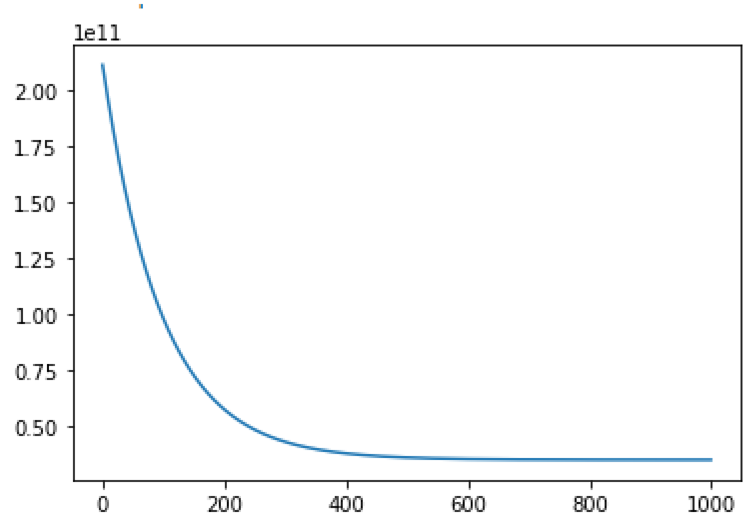
6. Multivariate Linear Regression - sklearn

```
-----------MULTIVARIATE REGRESSION - sklearn---------
features :
bedrooms        bathrooms       sqft_living     sqft_lot        floc
sqft_above      sqft_basement   yr_built        yr_renovated    zipc
Enter number of features : 3

Enter feature names :
sqft_living
grade
bedrooms
RSS : 2.573069e+14


R2 : 0.5583524018584544
```

Conclusions

Multivariate linear regression takes into consideration more no of features . Due to the fact Multivariate linear regression performs better in this case . This is well proved by the R^2 and the RSS values .

The R^2 and the RSS values for multivariate linear regression are as follows :




```
RSS : 2.837455e+14

R2 : 0.5129725660158259
```

References

1. Investopedia.
2. Housing Price Prediction via Improved Machine Learning Techniques - ScienceDirect
3. Housing Price Prediction Based on Multiple Linear Regression
4. Kumari, Khushbu & Yadav, Suniti. (2018). Linear regression analysis study. Journal of the Practice of Cardiovascular Sciences. 4. 33. 10.4103/jpcs.jpcs_8_18.
5. Manorathna, Rukshan. (2020). Linear Regression with Gradient Descent.
6. Bargiela, Andrzej & Nakashima, Tomoharu & Pedrycz, W.. (2005). Iterative gradient descent approach to multiple regression with fuzzy data. 304- 309. 10.1109/NAFIPS.2005.1548552.