In [11]:

```
# Required Python Packages
import pandas as pd
from math import pow


def get_headers(dataframe):
    """
    Get the headers name of the dataframe
    :param dataframe:
    :return:
    """
    return dataframe.columns.values


def cal_mean(readings):
    """
    Function to calculate the mean value of the input readings
    :param readings:
    :return:
    """
    readings_total = sum(readings)
    number_of_readings = len(readings)
    mean = readings_total / float(number_of_readings)
    return mean


def cal_variance(readings):
    """
    Calculating the variance of the readings
    :param readings:
    :return:
    """

    # To calculate the variance we need the mean value
    # Calculating the mean value from the cal_mean function
    readings_mean = cal_mean(readings)
    # mean difference squared readings
    mean_difference_squared_readings = [pow((reading - readings_mean), 2) for
reading in readings]
    variance = sum(mean_difference_squared_readings)
    return variance / float(len(readings) - 1)


def cal_covariance(readings_1, readings_2):
    """
    Calculate the covariance between two different list of readings
    :param readings_1:
    :param readings_2:
    :return:
    """
    readings_1_mean = cal_mean(readings_1)
    readings_2_mean = cal_mean(readings_2)
    readings_size = len(readings_1)
    covariance = 0.0
    for i in range(0, readings_size):
        covariance += (readings_1[i] - readings_1_mean) * (readings_2[i] - readings_2_mean)
    return covariance / float(readings_size - 1)


def cal_simple_linear_regression_coefficients(x_readings, y_readings):
    """
    Calculating the simple linear regression coefficients (B0, B1)
    :param x_readings:
    :param y_readings:
    :return:
    """
    # Coefficient B1 = covariance of x_readings and y_readings divided by variance of x_readings
    # Directly calling the implemented covariance and the variance functions
    # To calculate the coefficient B1
    b1 = cal_covariance(x_readings, y_readings) / float(cal_variance(x_readings))

    # Coefficient B0 = mean of y_readings - ( B1 * the mean of the x_readings )
    b0 = cal_mean(y_readings) - (b1 * cal_mean(x_readings))
    return b0, b1


def predict_target_value(x, b0, b1):
    """
    Calculating the target (y) value using the input x and the coefficients b0, b1
    :param x:
    :param b0:
    :param b1:
    :return:
    """
    return b0 + b1 * x


def cal_rmse(actual_readings, predicted_readings):
    """
    Calculating the root mean square error
    :param actual_readings:
    :param predicted_readings:
    :return:
    """
    square_error_total = 0.0
    total_readings = len(actual_readings)
    for i in range(0, total_readings):
        error = predicted_readings[i] - actual_readings[i]
        square_error_total += pow(error, 2)
    rmse = square_error_total / float(total_readings)
    return rmse


def simple_linear_regression(dataset):
    """
    Implementing the simple linear regression without using any python library
    :param dataset:

    :return:
    """

    # Get the dataset header names
    dataset_headers = get_headers(dataset)
    print("Dataset Headers :: ", dataset_headers)

    # Calculating the mean of the square feet and the price readings
    square_feet_mean = cal_mean(dataset[dataset_headers[0]])
    price_mean = cal_mean(dataset[dataset_headers[1]])

    square_feet_variance = cal_variance(dataset[dataset_headers[0]])
    price_variance = cal_variance(dataset[dataset_headers[1]])

    # Calculating the regression
    covariance_of_price_and_square_feet = dataset.cov()[dataset_headers[0]][da
taset_headers[1]]
    w1 = covariance_of_price_and_square_feet / float(square_feet_variance)

    w0 = price_mean - (w1 * square_feet_mean)

    # Predictions
    dataset['Predicted_Price'] = w0 + w1 * dataset[dataset_headers[0]]
    print (dataset['Predicted_Price'])

if __name__ == "__main__":

    input_path = 'input_data.csv'
    house_price_dataset = pd.read_csv(input_path)
    simple_linear_regression(house_price_dataset)
```

```
Dataset Headers ::  ['square_feet' 'price']
0     6088.297872
1     7527.127660
2     8965.957447
3    10404.787234
4    11843.617021
5    13282.446809
6    19037.765957
Name: Predicted_Price, dtype: float64
```

```
In [ ]:
```

```
#Conclusion:On Given Dataset Linear Regression Model FROM Scrath perform bette
r than sklearn model with predication house parameter.
```