1) As the data are stored in Datafile, thus need load into Jupyter notebook as dataframe first before converting to numpy.

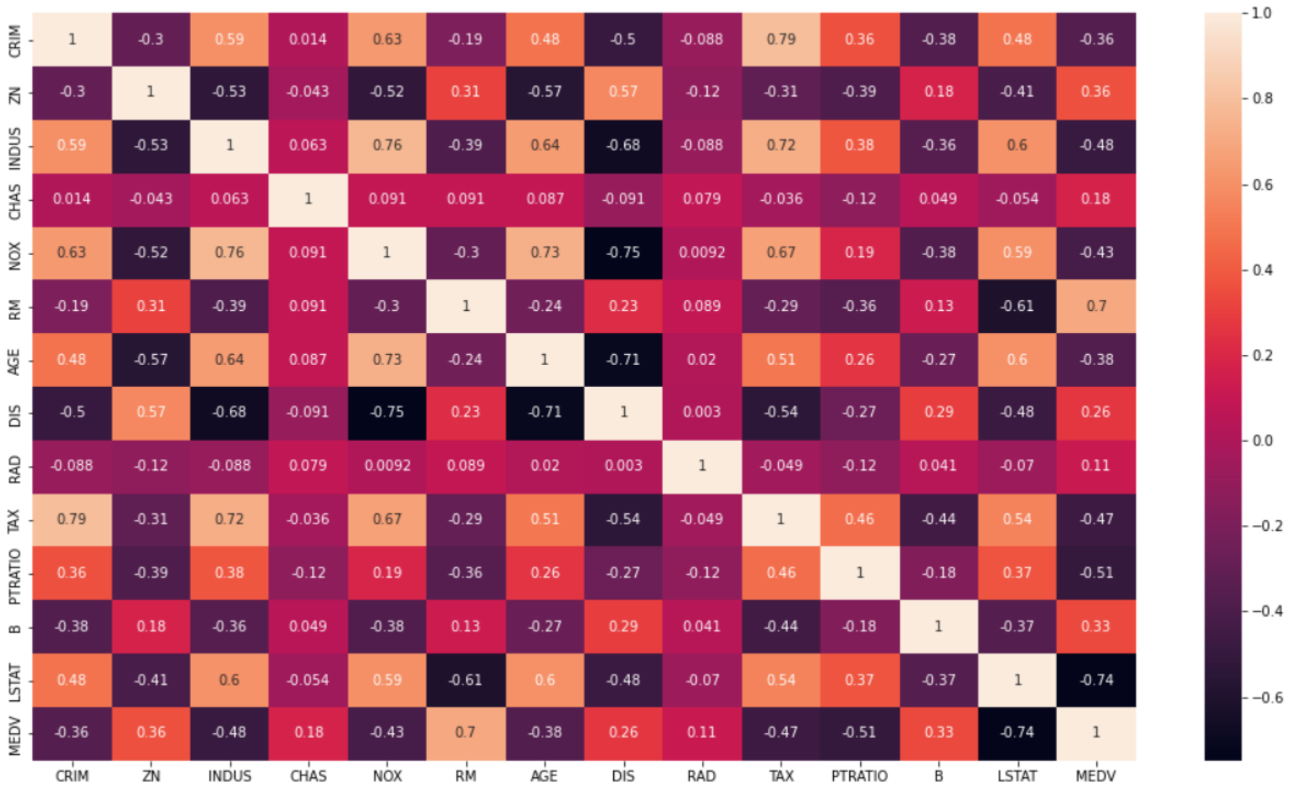
   There are no missing data for this dataset and thus, original data will be used for this case and no data modification is to be make.

2) linear regression equation:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + b_9x_9 + b_{10}x_{10} + b_{11}x_{11} + b_{12}x_{12} + b_{13}x_{13}$$

   As since there are 13 numerical input variables, thus 13 parameters are needed to determine the output(y), which is the predicted median value of owner-occupied homes in Boston.

3) 1st: plot the correlation between the output vs the 13 parameters:



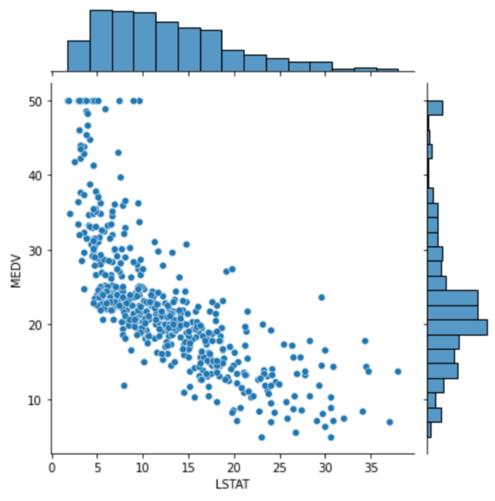
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	1	-0.3	0.59	0.014	0.63	-0.19	0.48	-0.5	-0.088	0.79	0.36	-0.38	0.48	-0.36
ZN	-0.3	1	-0.53	-0.043	-0.52	0.31	-0.57	0.57	-0.12	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.59	-0.53	1	0.063	0.76	-0.39	0.64	-0.68	-0.088	0.72	0.38	-0.36	0.6	-0.48
CHAS	0.014	-0.043	0.063	1	0.091	0.091	0.087	-0.091	0.079	-0.036	-0.12	0.049	-0.054	0.18
NOX	0.63	-0.52	0.76	0.091	1	-0.3	0.73	-0.75	0.0092	0.67	0.19	-0.38	0.59	-0.43
RM	-0.19	0.31	-0.39	0.091	-0.3	1	-0.24	0.23	0.089	-0.29	-0.36	0.13	-0.61	0.7
AGE	0.48	-0.57	0.64	0.087	0.73	-0.24	1	-0.71	0.02	0.51	0.26	-0.27	0.6	-0.38
DIS	-0.5	0.57	-0.68	-0.091	-0.75	0.23	-0.71	1	0.003	-0.54	-0.27	0.29	-0.48	0.26
RAD	-0.088	-0.12	-0.088	0.079	0.0092	0.089	0.02	0.003	1	-0.049	-0.12	0.041	-0.07	0.11
TAX	0.79	-0.31	0.72	-0.036	0.67	-0.29	0.51	-0.54	-0.049	1	0.46	-0.44	0.54	-0.47
PTRATIO	0.36	-0.39	0.38	-0.12	0.19	-0.36	0.26	-0.27	-0.12	0.46	1	-0.18	0.37	-0.51
B	-0.38	0.18	-0.36	0.049	-0.38	0.13	-0.27	0.29	0.041	-0.44	-0.18	1	-0.37	0.33
LSTAT	0.48	-0.41	0.6	-0.054	0.59	-0.61	0.6	-0.48	-0.07	0.54	0.37	-0.37	1	-0.74
MEDV	-0.36	0.36	-0.48	0.18	-0.43	0.7	-0.38	0.26	0.11	-0.47	-0.51	0.33	-0.74	1

2nd: from the heatmap above, the 5 top attributes are:

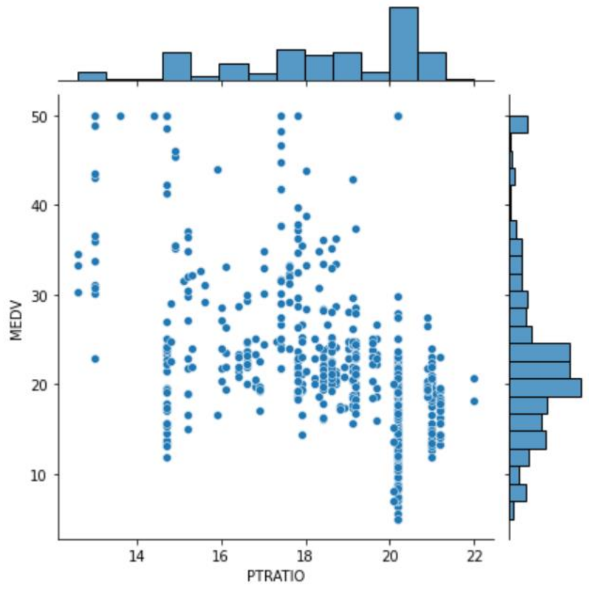
- LSTAT
- RM
- PTRATIO
- TAX
- INDUS

Below are the following plots between the 5 top attributes and the output respectively:

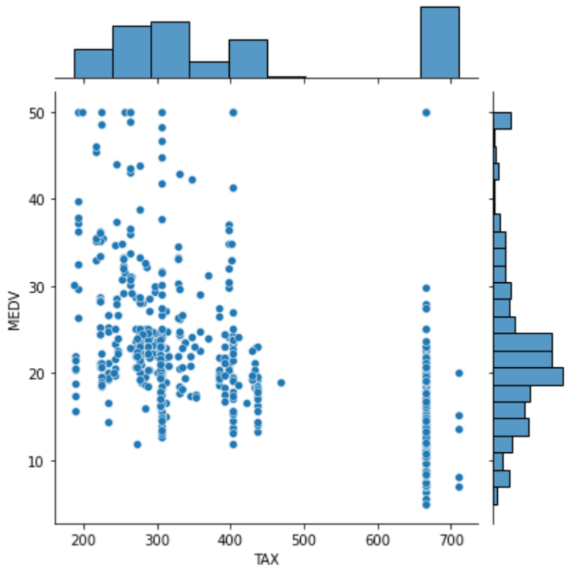
⇨ % Lower status of the population vs Median value of owner-occupied homes in $1000's



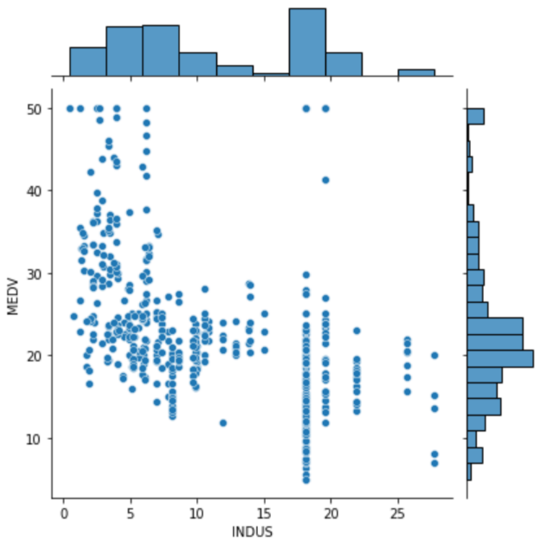
⇨ pupil-teacher ratio by town vs Median value of owner-occupied homes in $1000's



⇨ full-value property-tax rate per $10,000 vs Median value of owner-occupied homes in $1000's



⇨ proportion of non-retail business acres per town vs Median value of owner-occupied homes in $1000's



4) Root Mean Square Error (RMSE) is to show how concentrated the data is around the line of best fit. The lower the RMSE, the better a given model can "fit" a dataset.
Below is the formula:

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}}$$

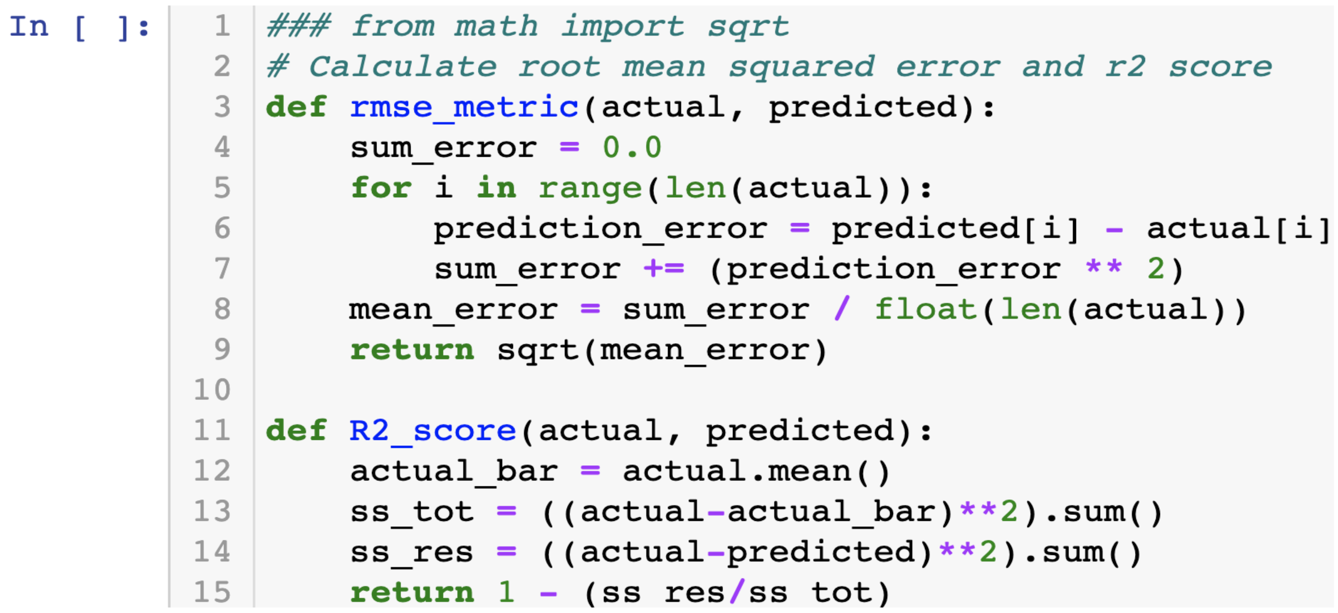
RMSD	= root-mean-square deviation
i	= variable i
N	= number of non-missing data points
x_i	= actual observations time series
$\hat{x}_i$	= estimated time series

R2-Score is the proportion of the variation in the dependent variable that is predictable from the independent variable(s). Generally, a higher R2-Score indicates a better fit for the model.
Below is the formula:

$$R^2 = 1 - \frac{RSS}{TSS}$$

R^2	= coefficient of determination
RSS	= sum of squares of residuals
TSS	= total sum of squares

Below is the python code for these 2 functions above:

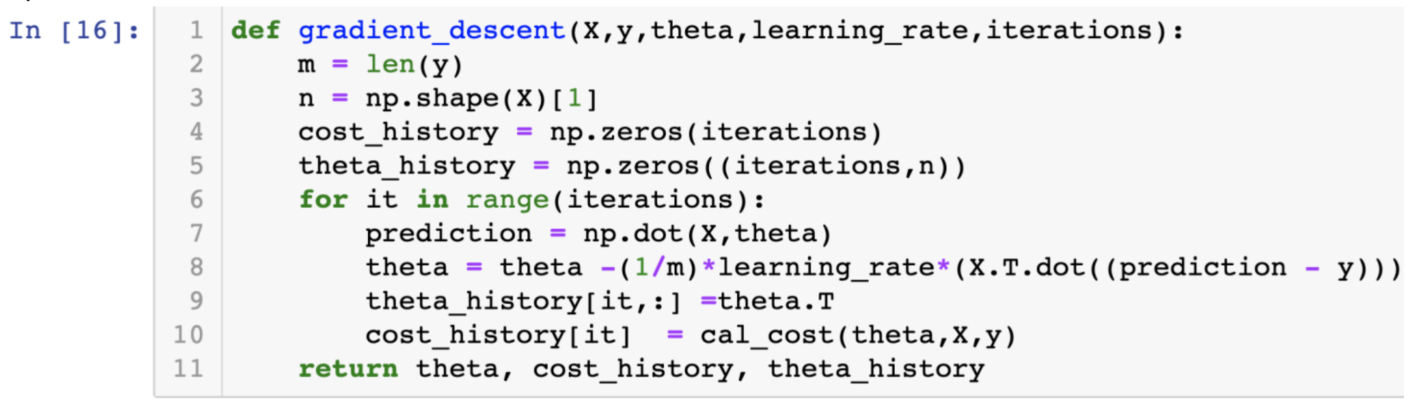


In [ ]:

```python
### from math import sqrt
# Calculate root mean squared error and r2 score
def rmse_metric(actual, predicted):
    sum_error = 0.0
    for i in range(len(actual)):
        prediction_error = predicted[i] - actual[i]
        sum_error += (prediction_error ** 2)
    mean_error = sum_error / float(len(actual))
    return sqrt(mean_error)

def R2_score(actual, predicted):
    actual_bar = actual.mean()
    ss_tot = ((actual-actual_bar)**2).sum()
    ss_res = ((actual-predicted)**2).sum()
    return 1 - (ss res/ss tot)
```

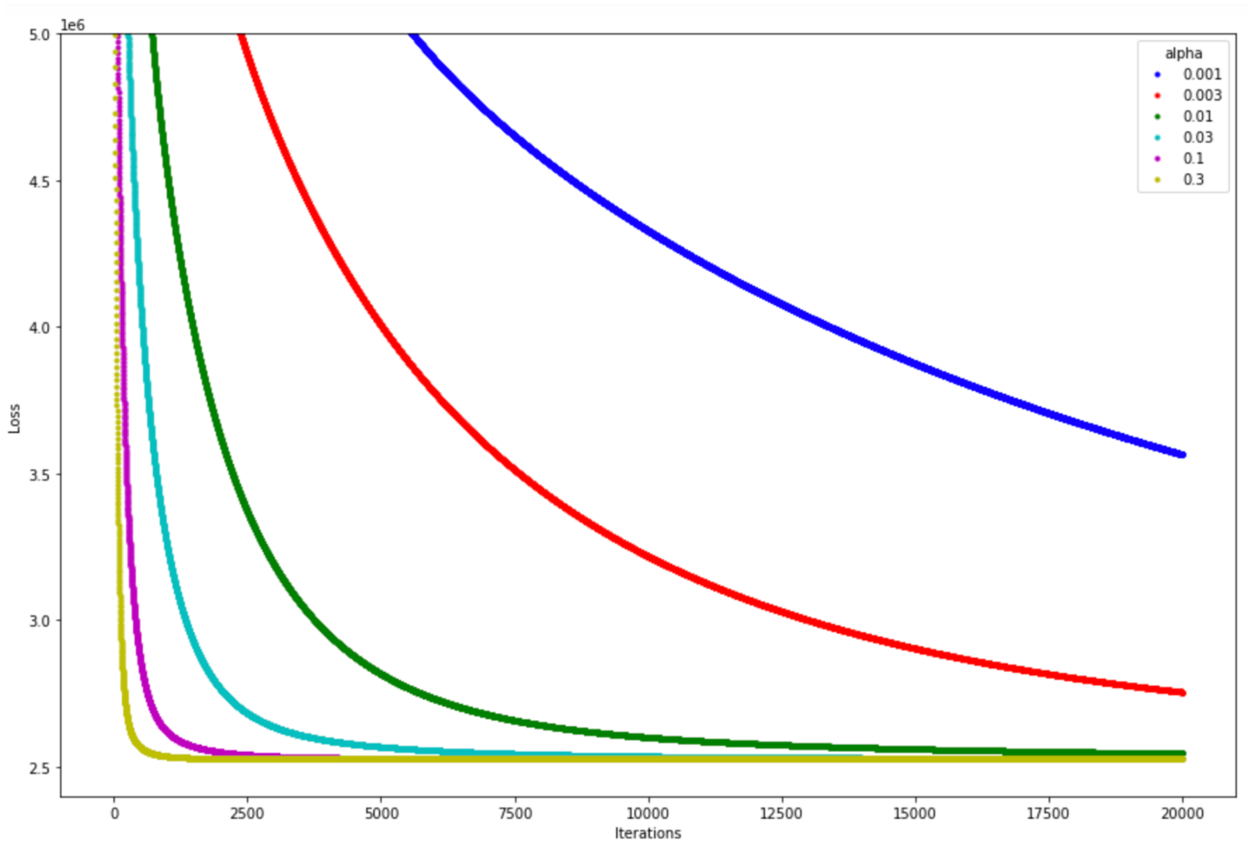
5) Gradient Descent is to estimate the model parameters that minimize the error of the model.
Python code:



In [16]:

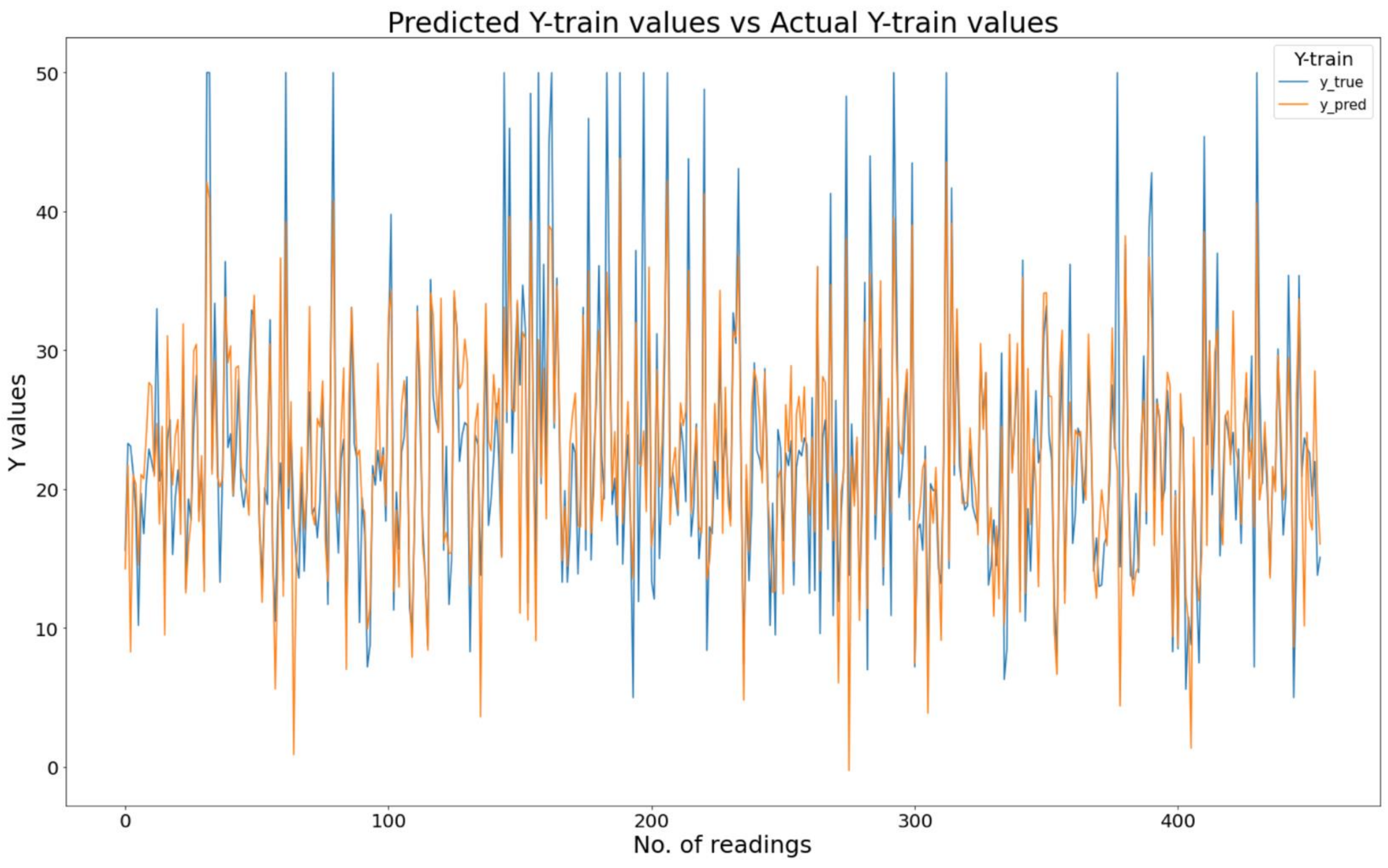
```python
def gradient_descent(X,y,theta,learning_rate,iterations):
    m = len(y)
    n = np.shape(X)[1]
    cost_history = np.zeros(iterations)
    theta_history = np.zeros((iterations,n))
    for it in range(iterations):
        prediction = np.dot(X,theta)
        theta = theta -(1/m)*learning_rate*(X.T.dot((prediction - y)))
        theta_history[it,:] =theta.T
        cost_history[it]  = cal_cost(theta,X,y)
    return theta, cost_history, theta_history
```

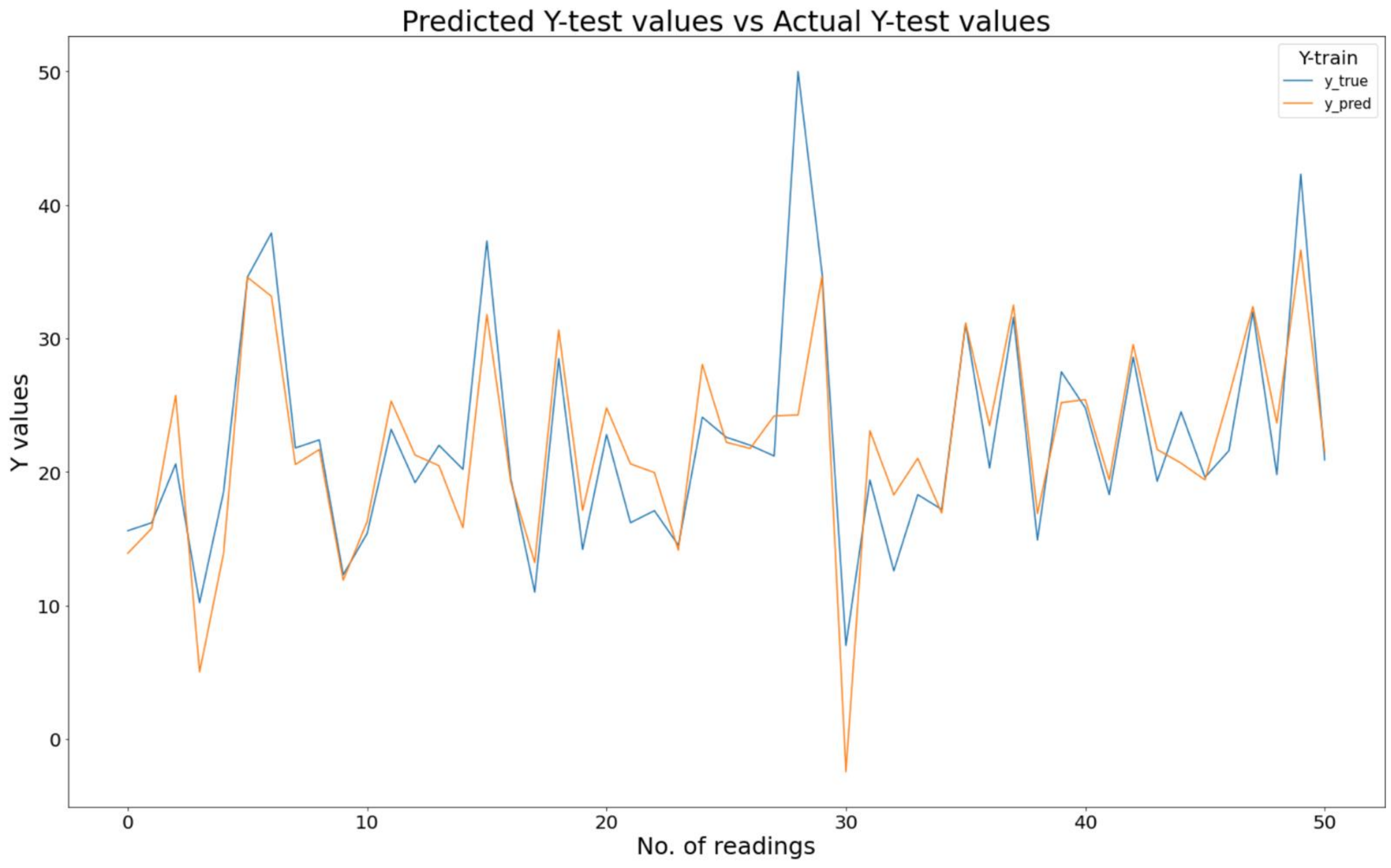
Below is the plot for different learning rate, α at the number of iteration=20000:



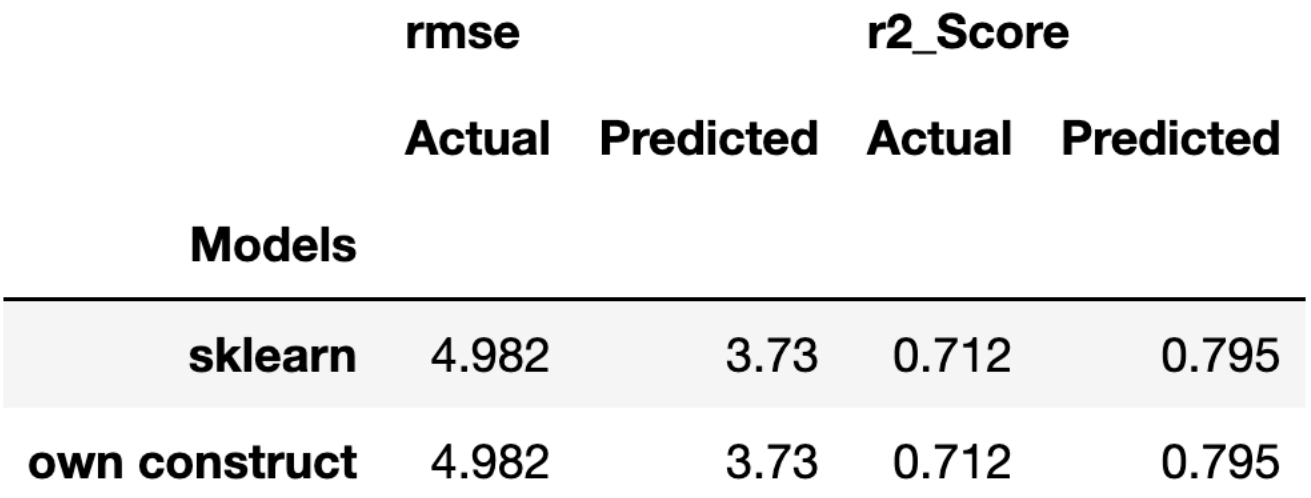
Based on the graph above, the best learning rate is α=0.3 since it generates the least loss at the end of iterations.

Qn8)





Below is the comparison between sklearn and constructed model in terms of RMSE and R2 scores:



	rmse		r2_Score	
	Actual	Predicted	Actual	Predicted
Models				
sklearn	4.982	3.73	0.712	0.795
own construct	4.982	3.73	0.712	0.795

Based on the graphs and rmse and r2_score, this can conclude that the constructed model is accurate as sklearn model.