# Module 2 – Linear Regression

## Objectives

At the end of this module, you should be able to:

- Work with Linear Regression
- Build Univariate Linear Regression Model
- Understand Gradient Descent Algorithm
- Implement Linear Regression with sklearn
- Multivariate Linear Regression
- Dummy Variables
- Feature Scaling
- Boston Housing Prizes Prediction
- Cross Validation
- Regularization: Lasso & Ridge

# Linear Regression

Before we continue to focus topic i.e. “Linear Regression” lets first know what we mean by Regression. Regression is a statistical way to establish a relationship between a dependent variable and a set of independent variable(s). e.g., if we say that

**Age = 5 + Height * 10 + Weight * 13**

Here we are establishing a relationship between Height & Weight of a person with his/ Her Age. This is a very basic example of Regression.

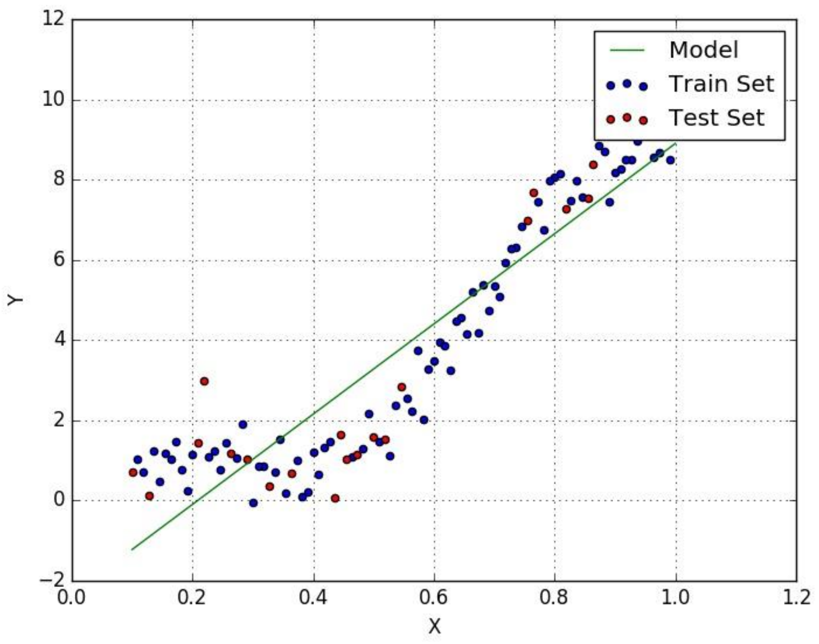
## What is Linear Regression?

A Supervised Learning Algorithm that learns from a set of training samples

“Linear Regression” can be defined as a statistical method to regress the data with dependent variable having continuous values whereas independent variables can have either continuous or categorical values.

“Linear Regression” can also be called a method to predict dependent variable (Y) based on values of independent variables (X).

In other words, we can say It simply provides estimation of relationship between a dependent variable (target/label) and one or more independent variable (predictors).



## Assumptions of Linear Regression

Not a single size fits or all, the same is true for Linear Regression as well. In order to fit a linear regression line data should satisfy few basic but important assumptions. If your data doesn't follow the assumptions, your results may be wrong as well as misleading.

i. Linearity & Additive: There should be a linear relationship between dependent and independent variables and the impact of change in independent variable values should have additive impact on dependent variable.

ii. Normality of error distribution: Distribution of differences between Actual & Predicted values (Residuals) should be normally distributed.

iii. Homoscedasticity: Variance of errors should be constant versus,
   - Time
   - The predictions
   - Independent variable values

iv. Statistical independence of errors: The error terms (residuals) should not have any correlation among themselves. E.g., In case of time series data there shouldn't be any correlation between consecutive error terms.

## Objective of Linear Regression

- Establish If there is a relationship between two variables.

  Examples – relationship between housing process and area of house, no of hours of study and the marks obtained, income and spending etc.

- Prediction of new possible values

  Based on the area of house predicting the house prices in a particular month; based on number of hour studied predicting the possible marks. Sales in next 3months etc.

## Linear Regression Use cases

- To model residential home prices as a function of the home's living area, bathrooms, number of bedrooms, lot size.

- To analyze the effect of a proposed radiation treatment on reducing tumor sizes based on patient attributes such as age or weight.

- To predict demand for goods and services. For example, restaurant chains can predict the quantity of food depending on weather.

- To predict company's sales based on previous month's sales and stock prices of a company.

## Regression Types

Univariate Linear Regression

$$y = m_1x_1\text{+c}$$

Multiple Linear Regression

$$y = m_1x_1 + m_2x_2 + m_3x_3 + ...+m_nx_n\text{+c}$$

Polynomial Linear Regression

$$y = m_1x_1 + m_2x_1^2 + m_3x_1^3 + ...+m_nx_1^n\text{+c}$$

## Simple Linear Regression

In simple linear regression, we predict scores on one variable from the scores on a second variable.

- The variable we are predicting is called the criterion variable and is referred to as Y.
- The variable we are basing our predictions on is called the predictor variable and is referred to as X.

When there is only one predictor variable, the prediction method is called simple regression.

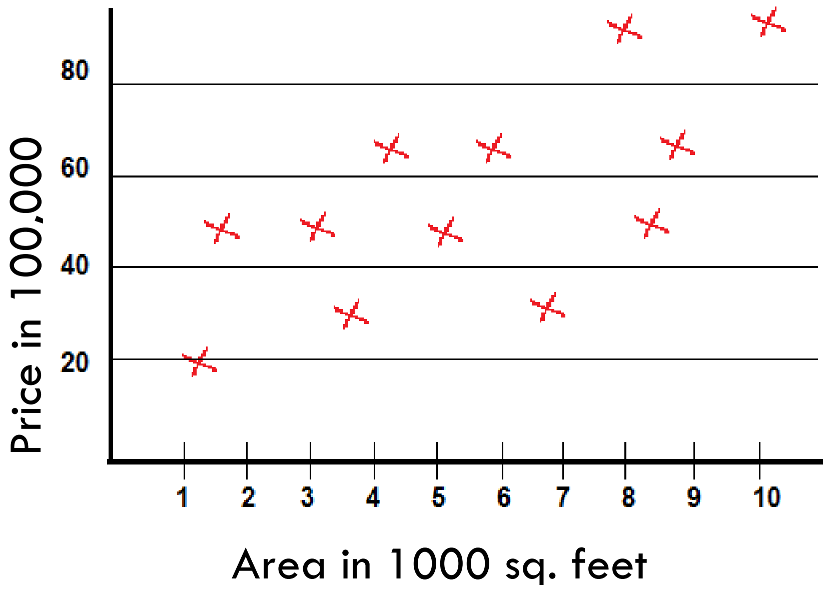
In simple linear regression, the topic of this section, the predictions of Y when plotted as a function of X form a straight line.

Here in the below example, considering we have a dataset in which we have the area of the several houses and their price, and we wish to build a predictive model using this dataset which can predict he price of the house based on the area fed to it.

In this example the Area will be input variable or input feature or independent variable and the price will be output variable or target or dependent variable.

| Area ( sq ft ) | Price ( INR ) |
|---|---|
| 1200 | 20,00,000 |
| 1800 | 42,00,000 |
| 3200 | 44,00,000 |
| 3800 | 25,00,000 |
| 4200 | 62,00,000 |

If we plot the data given to use in below format considering x axis as the area of the house and y axis as the price of the house.



x: Independent variable, predictor variables or regressors.

y: Dependent Variable, criterion variable, or regressand.

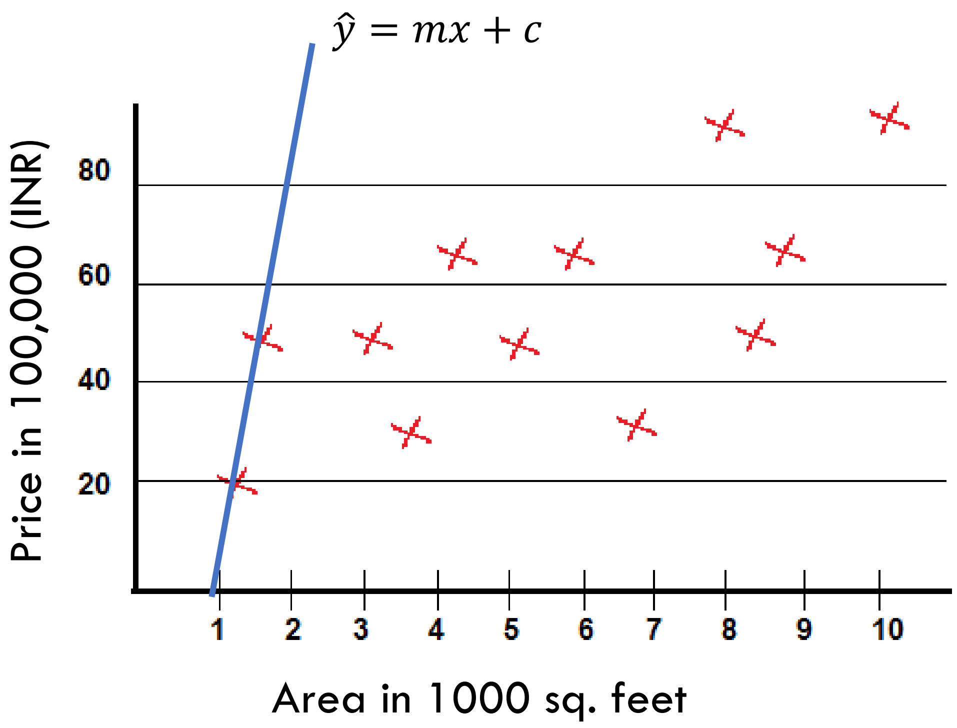
Now let us consider a scenario that I have two known people, one is my friend who is into the real estate business and have just started doing the business and done 2 deals so far and another is my uncle who is into real estate business from last 15 years, so my uncle would have consumed more data about the flats during his 15 years, Now in case I have to buy a property in the city, my uncle can be more correct in suggesting me the price compared to my friend and the reason is data, my uncle has consumed more data compared to my friend which makes him more correct. Similarly, in Machine Learning also as much as you can feed data to the algorithm that much better and effective algorithm you can build.

Consider a scenario of my friend who did only 2 deals so far, so considering that we have first two samples only ad we have to predict the price for the third value of are i.e. price of the house for 3200 sq. ft.,

How do we do it?

Probably you may say that you can draw a line between 2 existing point and extend the line to make prediction about the third point on x axis, that's actually is one of the correct and easiest way.

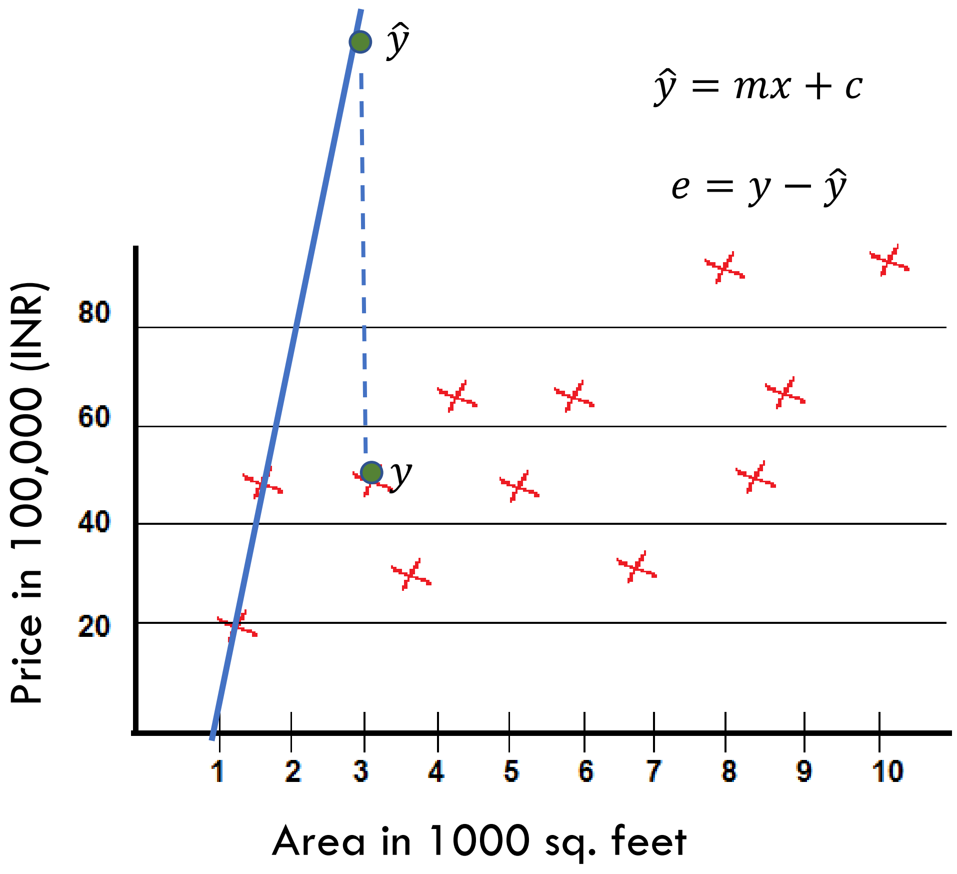
So lets consider a straight line as a predictive model.



Now in this case, we can say that the current straight line is not so correct if it makes prediction for all the known values, that means it is not fit to the pattern of our current data.

It has high error between predicted price for all the known values of x and their real/actual price.

Lets calculate he error for current straight line for x=3200 sq.ft.



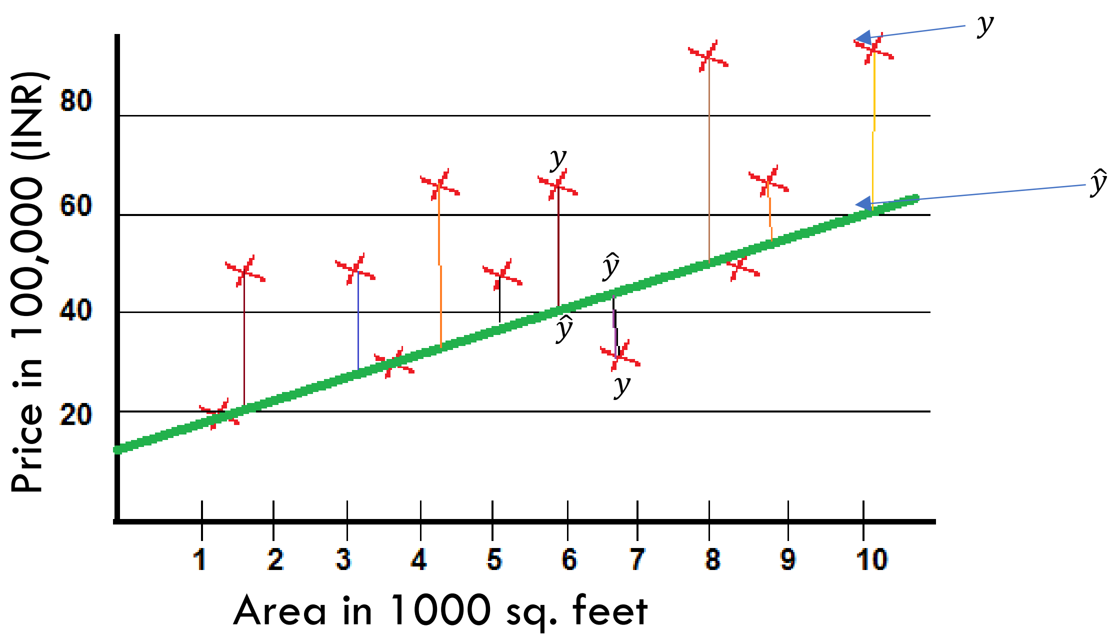
$\hat{y}$ = *Value predicted by current Algorithm*
$x$ = *input (here Area of flat)*
$y$ = *Actual Output*

So here we have depicted out error as the difference between the actual price of the house and the predicted price of the house.

Now in real world, the errors can be dual directional, means it can be positive and negative both.

Here in below image we can see in one of the case i.e. for 7th sample (which is at 6000 sq. f.t value of x approx) the error e is positive where as for 8th sample (which is at 7000 sq. ft. value of x approx) the error e is negative.

So when error is positive we have to decrease it to zero and when error is negative, we have to increase it to zero. Which makes it 2 directional optimization problem. TO resolve this issue we can take mean of square of error which is also known as MSE.



## $\boldsymbol{Cost\ Function}$

$$\boldsymbol{J = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \widehat{y_\iota})^2}$$

The final cost function can be depicted as –

$$\boldsymbol{j(m_i, c\ ) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \widehat{y_\iota})^2}$$

where n = the number of observations in the data,

$y_i$ = the actual value,

$\widehat{y_\iota}$ = the predicted value.

The constant value 1/2 is added in front for ease of computational purposes. You'll understand it in a while.

Now let us say that we are drawing another line which is green one like below one –

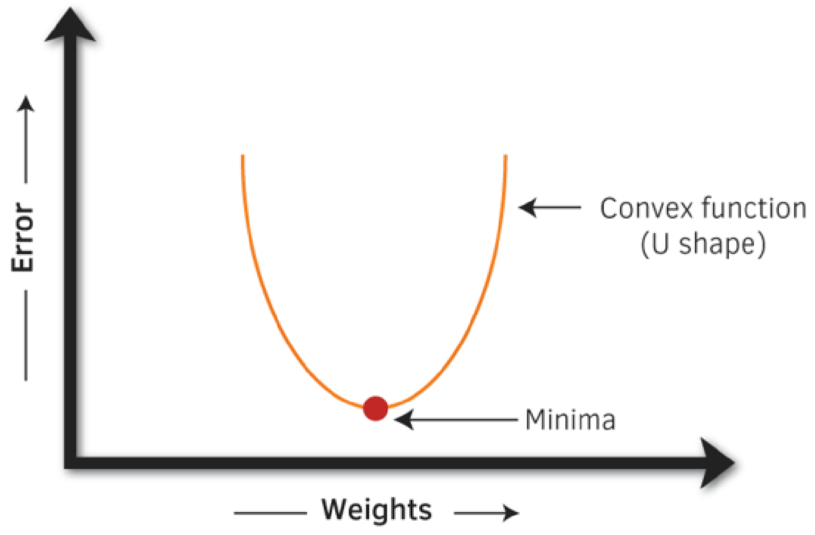
In above image we can say that green line would have less overall error compared to the red line, and how can we shift from red line to green line?

What should we change in current equation of line?

Probably you will say, change m & c. That means updating m & c in a certain direction, may reduce the cost function. So the factors affecting the value of cost function are m & c.

Machine Learning is all about learning from experiences and improving performance. Here if keep on feeding data and calculate the overall error and update m & c to minimize the error function or cost function iteration by iteration then we can say that line is learning to adapt the pattern of the data.

Lets talk more about the cost function -



This cost function in this case for Linear Regression is convex in nature. One of the important property of convex function is that it is guaranteed to provide the lowest value when differentiated at zero.

So now on a whole we can describe the final understanding of prediction of a continuous parameter in following manner -

Regression Equation: $\hat{y} = m_i x_i + c$

Parameters $m_i, c$

Cost Function: $j(m_i, c\,) = \frac{1}{2n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{y_i} - \boldsymbol{\widehat{y_\imath}})^{\boldsymbol{2}}$

Goal $\underset{m_i,c}{minimize}\, J(m_i, c)$

Now the goal is to minimize the cost function by updating the parameters m and c.

So lets take the partial differential of cost function w.r.t. m and c.

$$\Rightarrow \frac{\partial J}{\partial m} = \frac{\partial}{\partial m}\left\{\frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right\}$$

$$\Rightarrow \frac{\partial J}{\partial m} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_i$$

As you can see, the constant 1/2 got cancelled. In partial differentiation, we differentiate the entire equation with respect to one variable, keeping other variables constant. We also learn that the partial derivative of this cost function is just the difference between actual and predicted values averaged over all observations (n). To compute m & c more effectively, gradient descent comes into picture. For a particular value of m & c, gradient descent works like this:

1. First, it calculates the partial derivative of the cost function.
2. If the derivative is positive, it decreases the parameter value.
3. If the derivative is negative, it increases the parameter value.
4. The motive is to reach to the lowest point (zero) in the convex curve where the derivative is minimum.
5. It progresses iteratively using a step size (η), also called as learning rate which is defined by the user. But make sure that the step size isn't too large or too small. Too small a step size will take longer to converge, too large a step size will never reach an optimum.

## Gradient Descent Algorithm

Repeat Until converge

$$w_j := w_j - lr \frac{\partial}{\partial w} J(w_j)$$

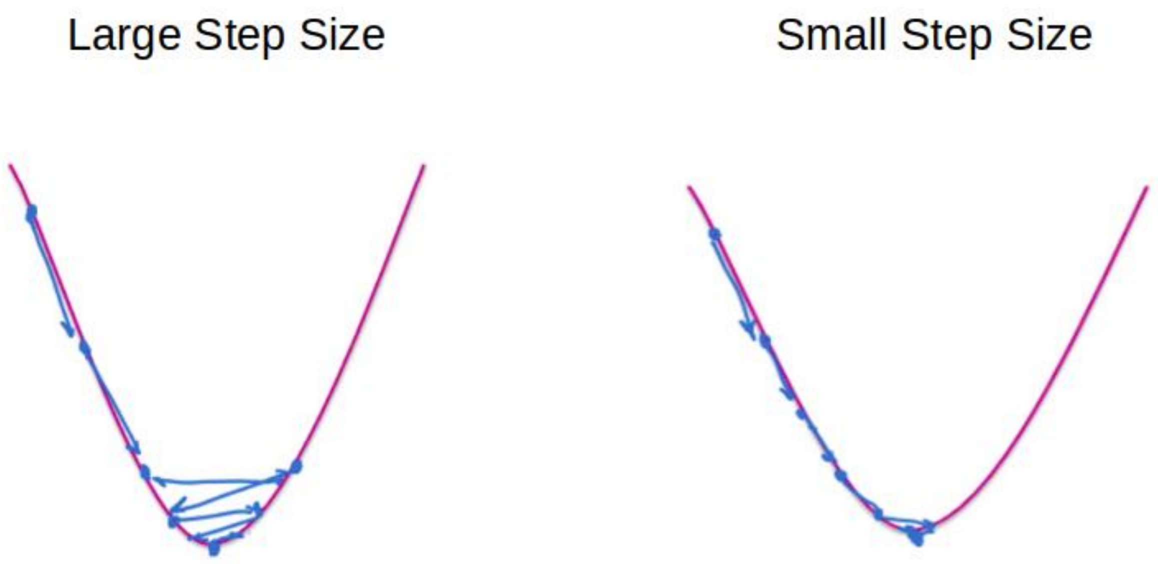
simultaneously update, $j$=0, $j$=1

where, w=parameter (coefficient & constant) (m & c)

## Learning Rate $lr$

Learning Rate $lr$ controls how big step we take while updating our parameter w.

- If $lr$ is too small, gradient descent can be slow.
- If $lr$ is too big, gradient descent can overshoot the minimum, it may fail to converge



**The above overall process is called as Linear Regression to build a linear predictive model.**

# Regression Model Performance

Once you build the model, the next logical question comes in mind is to know whether your model is good enough to predict in future or the relationship which you built between dependent and independent variables is good enough or not.

For this purpose there are various metrics which we look into-

1. **R – Square $(\boldsymbol{R^2})$**

   Formula for calculating $(R^2)$ is given by:

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

- Total Sum of Squares (TSS) : TSS is a measure of total variance in the response/ dependent variable Y and can be thought of as the amount of variability inherent in the response before the regression is performed.
- Residual Sum of Squares (RSS) : RSS measures the amount of variability that is left unexplained after performing the regression.
- (TSS – RSS) measures the amount of variability in the response that is explained (or removed) by performing the regression

  Where N is the number of observations used to fit the model, σx is the standard deviation of x, and σy is the standard deviation of y.

- $R^2$ ranges from 0 to 1.
- $R^2$ of 0 means that the dependent variable cannot be predicted from the independent variable
- $R^2$ of 1 means the dependent variable can be predicted without error from the independent variable
- An $R^2$ between 0 and 1 indicates the extent to which the dependent variable is predictable. An $R^2$ of 0.20 means that 20 percent of the variance in Y is predictable from X; an $R^2$ of 0.40 means that 40 percent is predictable; and so on.

**2. Root Mean Square Error (RMSE)**

RMSE tells the measure of dispersion of predicted values from actual values. The formula for calculating RMSE is

$$\boldsymbol{R^2 = \{\left(\frac{1}{N}\right) * \sum\left(x_i - mean(x)\right) * \left(y_i - mean(y)\right)]/(\sigma_x * \sigma_y)\}^2}$$

N : Total number of observations

Though RMSE is a good measure for errors but the issue with it is that it is susceptible to the range of your dependent variable. If your dependent variable has thin range, your RMSE will be low and if dependent variable has wide range RMSE will be high. Hence, RMSE is a good metric to compare between different iterations of a model.

3. **Mean Absolute Percentage Error (MAPE)**

To overcome the limitations of RMSE, analyst prefer MAPE over RMSE which gives error in terms of percentages and hence comparable across models. Formula for calculating MAPE can be written as:

$$\boldsymbol{RMSE = \sqrt{\frac{\sum(Y_{Actual} - Y_{Predicted})^2}{N}}}$$

N : Total number of observations

## How to improve the accuracy of a regression model?

There is little you can do when your data violates regression assumptions. An obvious solution is to use tree-based algorithms which capture non-linearity quite well. But if you are adamant at using regression, following are some tips you can implement:

1. It may be possible that all the features may not have good correlation with the label, in that case your data is suffering from non-linearity, you should **transform the Independent Variables** using sqrt, log, square, etc.

2. Sometimes it may be possible that your data is suffering from heteroskedasticity, **transform the Dependent Variable** using sqrt, log, square, etc. In such scenario you can also give a try using weighted least square method to tackle this problem.

3. In case if your data is suffering from multicollinearity, use a correlation matrix to check correlated variables. Let's say variables A and B are highly correlated. Now, instead of removing one of them, use this approach: Find the **average correlation** of A and B with the rest of the variables. Whichever variable has the higher average in comparison with other variables, remove it. Alternatively, you can use **penalized regression methods** such as lasso, ridge, elastic net, etc.

4. You can do variable selection based on **p values**. If a variable shows p value > 0.05, we can remove that variable from model since at p> 0.05, we'll always fail to reject null hypothesis.

# Multiple Linear Regression

Till now we were discussing about the scenario where we have only one independent variable. If we have more than one independent variable the procedure for fitting a best fit line is known as "Multiple Linear Regression"

Fundamentally there is no difference between 'Simple' & 'Multiple' linear regression. Both works on OLS principle and procedure to get the best line is also similar. In the case of later, regression equation will take a shape like:

$$\boldsymbol{y = m_1x_1 + m_2x_2 + m_3x_3 + \ldots + m_nx_n + c}$$

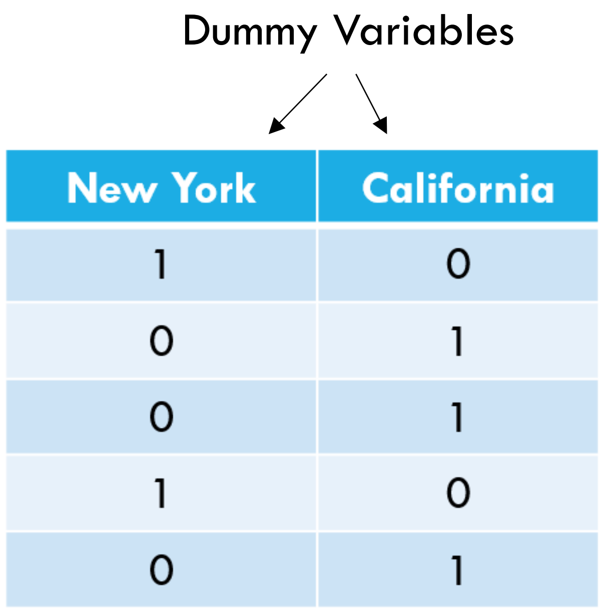
Here according to the number of independent variable the number of parameters will increase by n+1, if there are n independent variables, the geometry you can assume as n dimensional hyperplane.

## Feature Engineering for Multivariate Linear Regression

### One Hot Encoding

When some inputs are categories (e.g. gender) rather than numbers (e.g. age) we need to represent the category values as numbers so they can be used in our linear regression equations.

| Salary | Credit Score | Age | State |
|---|---|---|---|
| 192,451 | 485 | 42 | New York |
| 118,450 | 754 | 35 | California |
| 258,254 | 658 | 28 | California |
| 200,123 | 755 | 48 | New York |
| 152,485 | 654 | 52 | California |



Dummy Variables

| New York | California |
|---|---|
| 1 | 0 |
| 0 | 1 |
| 0 | 1 |
| 1 | 0 |
| 0 | 1 |

Avoiding the Dummy variable trap

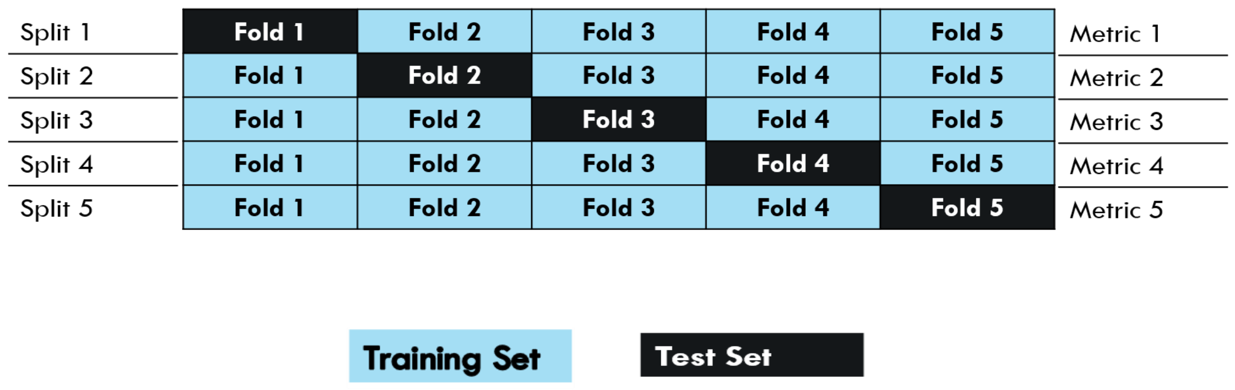
```
X=X[:,1:]
```

NOTE : if you have n dummy variables remove one dummy variable to avoid the dummy variable trap. However the linear regression model that is built in R and Python takes care of this. But there is no harm in removing it by ourselves

## Feature Scaling

| Standardization | Normalization |
|---|---|
| $X_{stand} = \frac{x - mean(x)}{standard\_deviation(x)}$ | $X_{norm} = \frac{x - min(x)}{\max(x) - \min(x)}$ |

## Cross Validation



- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k-fold CV

More folds = More computationally expensive

## Overfitting & Generalization

As we train our model with more and more data the it may start to fit the training data more and more accurately, but become worse at handling test data that we feed to it later.

This is known as “over-fitting” and results in an increased generalization error.

Large coefficients lead to overfitting

Penalizing large coefficients: Regularization

**How to minimize?**

- To minimize the generalization error we should
- Collect as much sample data as possible.
- Use a random subset of our sample data for training.
- Use the remaining sample data to test how well our model copes with data it was not trained with.

## L1 Regularization (Lasso)

(Least Absolute Shrinkage and Selection Operator)

- Having a large number of samples (n) with respect to the number of dimensionality (d) increases the quality of our model.
- One way to reduce the effective number of dimensions is to use those that most contribute to the signal and ignore those that mostly act as noise.
- L1 regularization achieves this by adding a penalty that results in the weight for the dimensions that act as noise becoming 0.
- L1 regularization encourages a sparse vector of weights in which few are non-zero and many are zero.

Depending on the regularization strength, certain weights can become zero, which makes the LASSO also useful as a supervised feature selection technique:

A limitation of the LASSO is that it selects at most n variables if m > n.

$$j(w_i\ ) = \frac{1}{2n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{y_i} - \boldsymbol{\widehat{y_\imath}})^{\boldsymbol{2}} + \boldsymbol{\lambda}\|\boldsymbol{w_i}\|$$

## L2 Regularization (Ridge)

- Another way to reduce the complexity of our model and prevent overfitting to outliers is L2 regression, which is also known as ridge regression.
- In L2 Regularization we introduce an additional term to the cost function that has the effect of penalizing large weights and thereby minimizing this skew.

Ridge regression is an L2 penalized model where we simply add the squared sum of the weights to our least-squares cost function:

By increasing the value of the hyperparameter λ , we increase the regularization strength and shrink the weights of our model.

$$j(w_i\ ) = \frac{1}{2n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{y_i} - \widehat{\boldsymbol{y_\imath}})^{\boldsymbol{2}} + \boldsymbol{\lambda}\|\boldsymbol{w_i}\|^{\boldsymbol{2}}$$

## L1 & L2 Regularization (Elastic Net)

• L1 Regularisation minimises the impact of dimensions that have low weights and are thus largely “noise”.

• L2 Regularisation minimise the impacts of outliers in our training data.

• L1 & L2 Regularisation can be used together and the combination is referred to as Elastic Net regularisation.

• Because the differential of the error function contains the sigmoid which has no inverse, we cannot solve for w and must use gradient descent.

**Lasso regression for feature selection**

● Can be used to select important features of a dataset

● Shrinks the coefficients of less important features to exactly 0.

# Multi-collinearity

Multi-collinearity tells us the strength of relationship between independent variables. If there is Multi-Collinearity in our data, our beta coefficients may be misleading. VIF (Variance Inflation Factor) is used to identify the Multi-collinearity. If VIF value is greater than 4 we exclude that variable from our model building exercise

## Iterative Models

Model building is not one step process, one need to run multiple iterations in order to reach a final model. Take care of P-Value and VIF for variable selection and R-Square & MAPE for model selection.

# Exercise 1 – Simple Linear Regression

Here we will take the salary dataset and apply simple linear regression using python on this.

## Import the libraries

```
import numpy
import matplotlib as plt
Import pandas
```

## Import dataset

```
dataset=pandas.read_csv('salary_data.csv')
X=dataset.iloc[:,:-1].values
Y=dataset.iloc[:,1].values
```

## Train test split

```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest =
train_test_split(X,Y,test_size=0.2,random_state=0)
```

## Simple Linear Regression

```
from sklearn import linear_model
alg = linear_model.LinearRegression()
alg.fit(xtrain,ytrain)
```

## Predicting the test results

```
ypred=alg.predict(xtest)
```

## Visualizing the test results

```
plt.scatter(xtest,ytest,‘g’)
plt.plot(xtest,alg.predict(xtest),’r’)
plt.title(“Test set”)
plt.xlabel(“Experience”)
plt.ylabel(“Salary”)
plt.show()
```

## Visualizing the training results

```
plt.scatter(xtrain,ytrain,‘g’)
plt.plot(xtrain,alg.predict(xtrain),’r’)
plt.title(“Training set”)
plt.xlabel(“Experience”)
plt.ylabel(“Salary”)
plt.show()
```

## Test Score (Accuracy on test data)

```
accuracy=alg.score(xtest,ytest)
print(accuracy)
```

Coefficient and intercept value

```
#for printing coefficient
print(alg.coef_)
# for printing intercept value
print(alg.intercept_)
```

Performance Analysis

```
from sklearn.metrics import mean_squared_error, r2_score
# The mean squared error
print("Mean squared error:
%.2f"%mean_squared_error(ytest,ypred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(ytest, ypred))
```

# Exercise 2 - Multiple Linear Regression Model

The dataset for this example is taken from –

https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

```
1. Title: Boston Housing Data

2. Sources:
   (a) Origin:  This dataset was taken from the StatLib library which is
                maintained at Carnegie Mellon University.
   (b) Creator:  Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the
                 demand for clean air', J. Environ. Economics &
Management,
                 vol.5, 81-102, 1978.
   (c) Date: July 7, 1993


4. Relevant Information:

   Concerns housing values in suburbs of Boston.

5. Number of Instances: 506

6. Number of Attributes: 13 continuous attributes (including "class"
                         attribute "MEDV"), 1 binary-valued attribute.

7. Attribute Information:

    1. CRIM      per capita crime rate by town
    2. ZN        proportion of residential land zoned for lots over
                 25,000 sq.ft.
    3. INDUS     proportion of non-retail business acres per town
    4. CHAS      Charles River dummy variable (= 1 if tract bounds
                 river; 0 otherwise)
    5. NOX       nitric oxides concentration (parts per 10 million)
    6. RM        average number of rooms per dwelling
    7. AGE       proportion of owner-occupied units built prior to 1940
    8. DIS       weighted distances to five Boston employment centres
    9. RAD       index of accessibility to radial highways
    10. TAX      full-value property-tax rate per $10,000
    11. PTRATIO  pupil-teacher ratio by town
    12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
                 by town
    13. LSTAT    % lower status of the population
    14. MEDV     Median value of owner-occupied homes in $1000's
```

```
8. Missing Attribute Values:  None.
```

```
In [1]: boston = pd.read_csv('boston.csv’)
In [2]: print(boston.head()
     CRIM ZN    INDUS CHAS NX     RM     AGE   DIS     RAD   TAX \
0 0.00632 18.0 2.31    0   0.538 6.575 65.2 4.0900    1    296.0
1 0.02731  0.0 7.07    0   0.469 6.421 78.9 4.9671    2    242.0
2 0.02729  0.0 7.07    0   0.469 7.185 61.1 4.9671    2    242.0
3 0.03237  0.0 2.18    0   0.458 6.998 45.8 6.0622    3    222.0
4 0.06905  0.0 2.18    0   0.458 7.147 54.2 6.0622    3    222.0

     PTRATIO          B        LSTAT            MEDV
0    15.3           396.90    4.98            24.0
1    17.8           396.90    9.14            21.6
2    17.8           392.83    4.03            34.7
3    18.7           394.63    2.94            33.4
4    18.7           396.90    5.33            36.2
```

Creating feature and target arrays

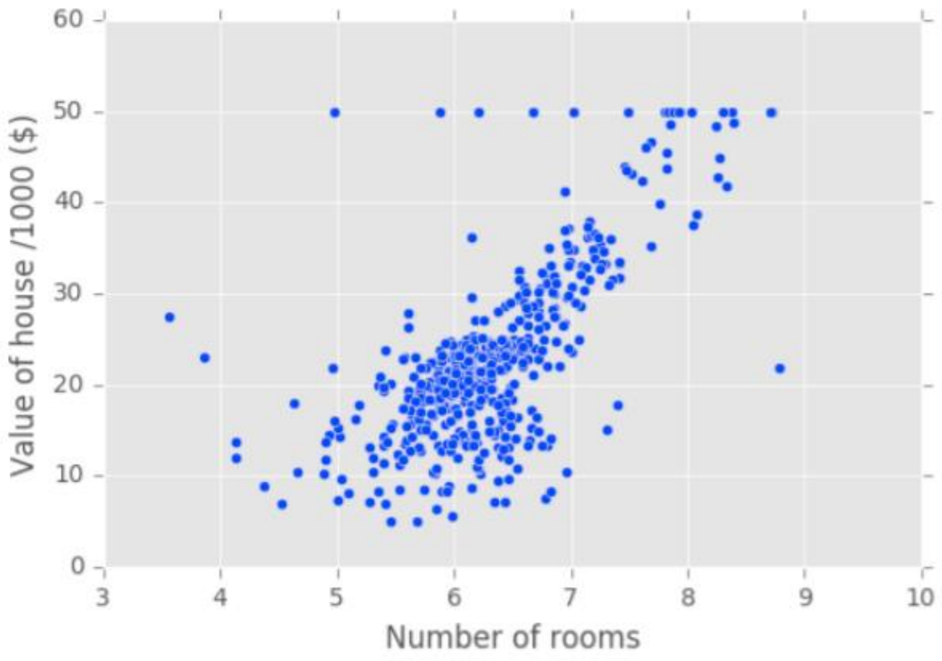
```
In [3]: X = boston.drop('MEDV', axis=1).values

In [4]: y = boston['MEDV'].values
```

Predicting house value from a single feature

```
In [5]: X_rooms = X[:,5]

In [6]: type(X_rooms), type(y)

Out[6]: (numpy.ndarray, numpy.ndarray)

In [7]: y = y.reshape(-1, 1)

In [8]: X_rooms = X_rooms.reshape(-1, 1)
```

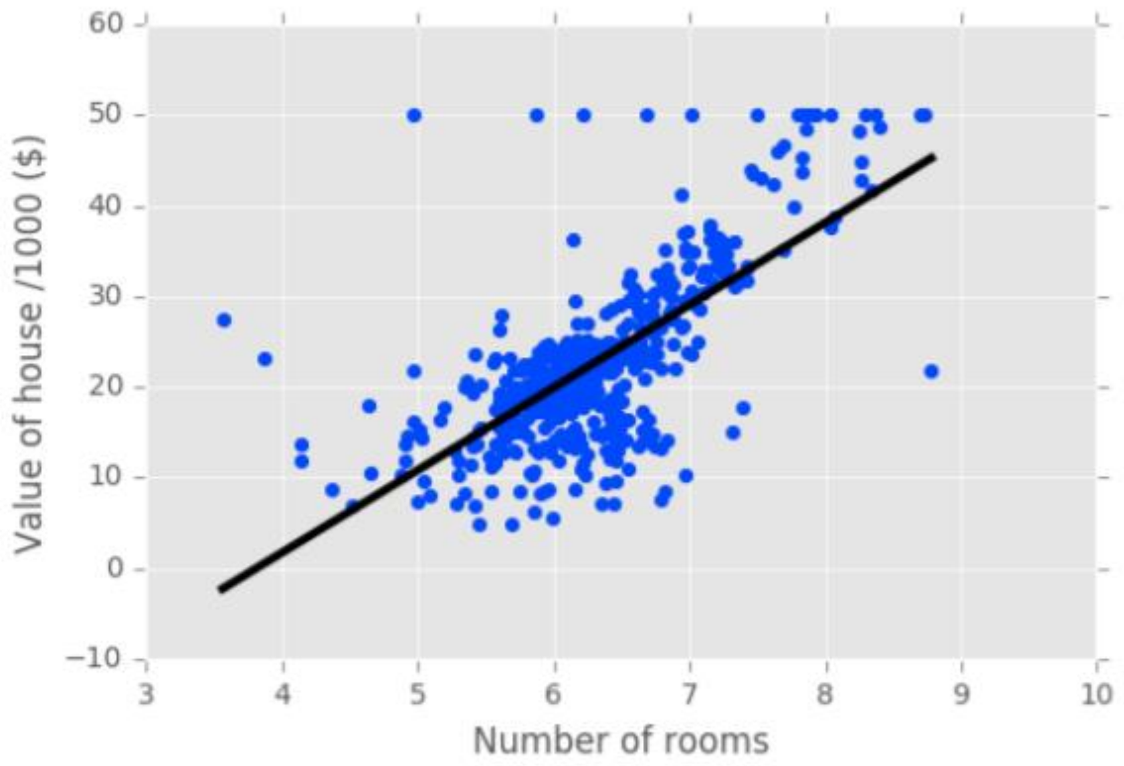
Plotting house value vs. number of rooms

```
In [9]: plt.scatter(X_rooms, y)

In [10]: plt.ylabel('Value of house /1000 ($)’)

In [11]: plt.xlabel('Number of rooms’)

In [12]: plt.show()
```



## Fitting a regression model

```
In [13]: from numpy import linspace

In [14]: from sklearn import linear_model

In [15]: alg = linear_model.LinearRegression()

In [16]: alg.fit(X_rooms, y)

In [17]: k=linspace(min(X_rooms),max(X_rooms)).reshape(-1,1)

In [18]: plt.scatter(X_rooms, y, color='blue’)

In [19]: plt.plot(k, alg.predict(k),'b', linewidth=3)

In [20]: plt.show()
```



Linear regression on all features

```
In [1]: from sklearn.model_selection import train_test_split

In [2]: X_train, X_test, y_train, y_test =
train_test_split(X, y,test_size = 0.3, random_state=42)

In [3]: alg2 = linear_model.LinearRegression()

In [4]: alg2.fit(X_train, y_train)

In [5]: y_pred = alg2.predict(X_test)

In [6]: alg2.score(X_test, y_test)

Out[6]: 0.7112260057484926
```

## Cross-validation in scikit-learn

```
In [1]: from sklearn.model_selection import cross_val_score

In [2]: alg = linear_model.LinearRegression()

In [3]: cv_results = cross_val_score(alg, X, y, cv=5)

In [4]: print(cv_results)

[ 0.63919994 0.71386698 0.58702344 0.07923081 -0.25294154]

In [5]: numpy.mean(cv_results)

Out[5]: 0.35327592439587058
```

## L1 Regularization (Lasso)

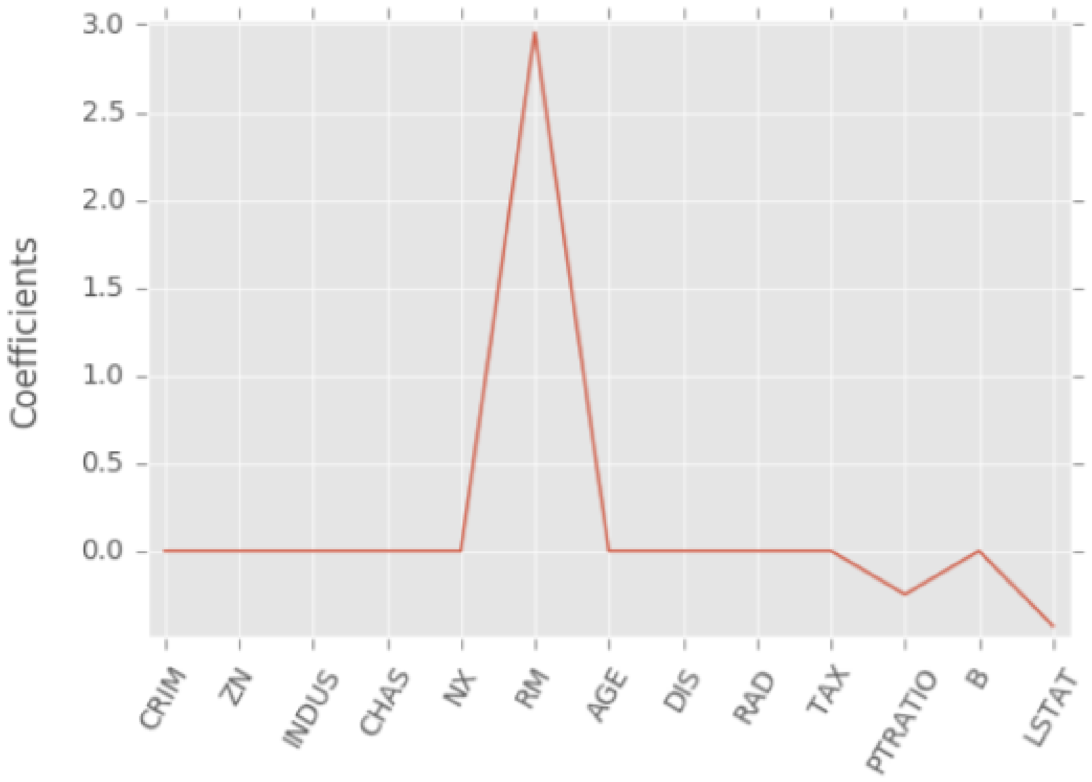
```
In [1]: from sklearn.linear_model import Lasso

In [2]: X_train, X_test, y_train, y_test =
train_test_split(X, y,test_size = 0.3, random_state=42)

In [3]: lasso = Lasso(alpha=0.1, normalize=True)

In [4]: lasso.fit(X_train, y_train)

In [5]: lasso_pred = lasso.predict(X_test)

In [6]: lasso.score(X_test, y_test)

Out[6]: 0.5950229535328551
```

L2 Regularization (Ridge)

```
In [1]: from sklearn.linear_model import Ridge

In [2]: X_train, X_test, y_train, y_test =
train_test_split(X, y,test_size = 0.3, random_state=42)

In [3]: ridge = Ridge(alpha=0.1, normalize=True)

In [4]: ridge.fit(X_train, y_train)

In [5]: ridge_pred = ridge.predict(X_test)

In [6]: ridge.score(X_test, y_test)

Out[6]: 0.69969382751273179
```

## Example 3 - Lasso regression for feature selection

```
In [1]: from sklearn.linear_model import Lasso

In [2]: names = boston.drop('MEDV', axis=1).columns

In [3]: lasso = Lasso(alpha=0.1)

In [4]: lasso_coef = lasso.fit(X, y).coef_

In [5]: plt.plot(range(len(names)), lasso_coef)

In [6]: plt.xticks(range(len(names)), names, rotation=60)

In [7]: plt.ylabel('Coefficients')

In [8]: plt.show()
```


Coefficients
3.0
2.5
2.0
1.5
1.0
0.5
0.0
CRIM
ZN
INDUS
CHAS
NX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT