MACHINE LEARNING USING PYTHON

SUMMER TRAINING REPORT 2020

Submitted in partial fulfillment of the

Requirements for the award of the degree

of

Bachelor of Technology in Electronics and Communication Engineering

By:

Name: ROHAN BANSAL

University Enrollment no.: (41476802817/ECE3/2017)

Department of Electronics and Communication Engineering

Guru Tegh Bahadur Institute of Technology

Guru Gobind Singh Indraprastha University

Dwarka, New Delhi

Year 2017-2021

DECLARATION

I hereby declare that all the work presented in the Summer Training Report 2020 entitled "MACHINE LEARNING USING PYTHON" in the partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering, Guru Tegh Bahadur Institute of Technology, Guru Govind Singh Indraprastha University, New Delhi is an authentic record of my own work.

Name: ROHAN BANSAL

University Enrollment no.: (41476802817/ECE3/2017)

CERTIFICATE

Verifiable Link: https://www.skyfilabs.com/verify-certificate/60091266

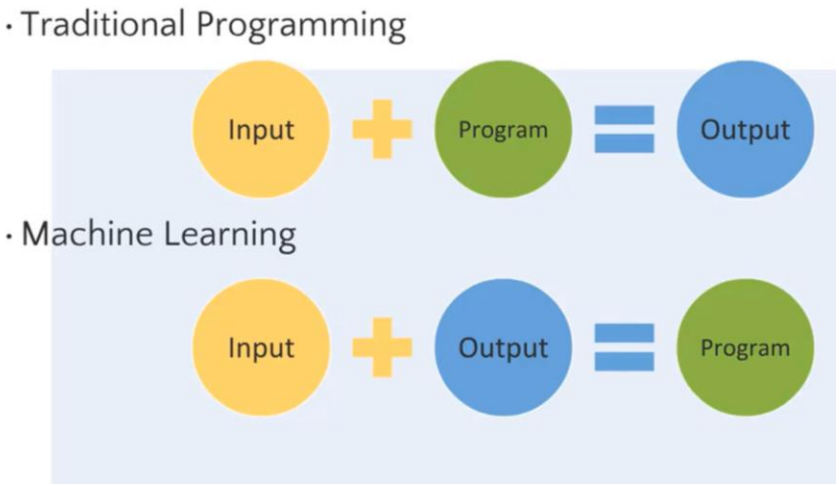
INDEX

1. MACHINE LEARNING

1.1 Overview

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed.
As it is evident from the name, it gives the computer that makes it more similar to humans: The ability to learn.

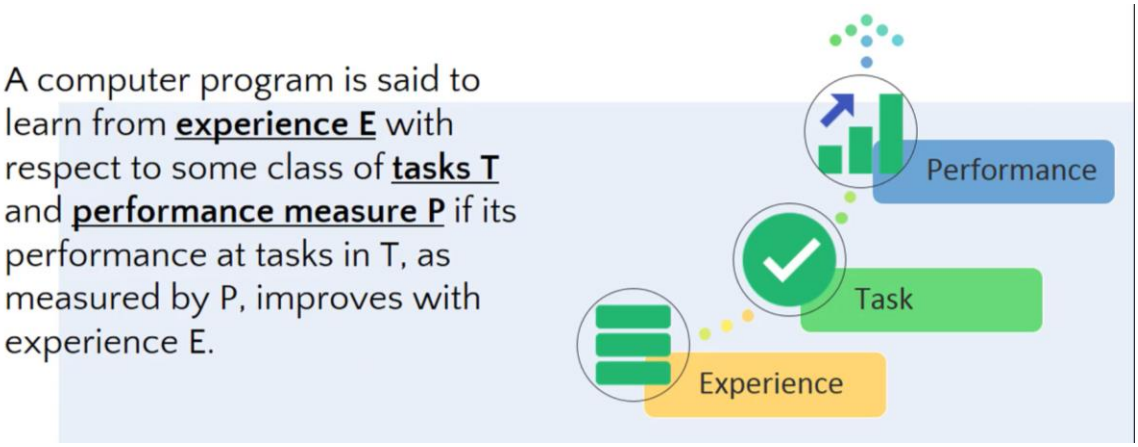
How it is different from traditional programming



1.1.1. History of Machine Learning

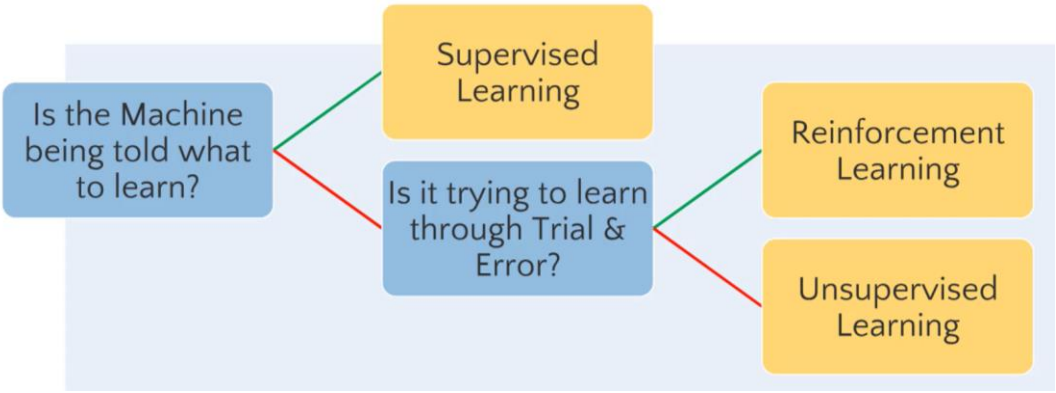
The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence and stated that "it gives computers the ability to learn without being explicitly programmed".
And in 1997, Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Formal definition of machine learning.



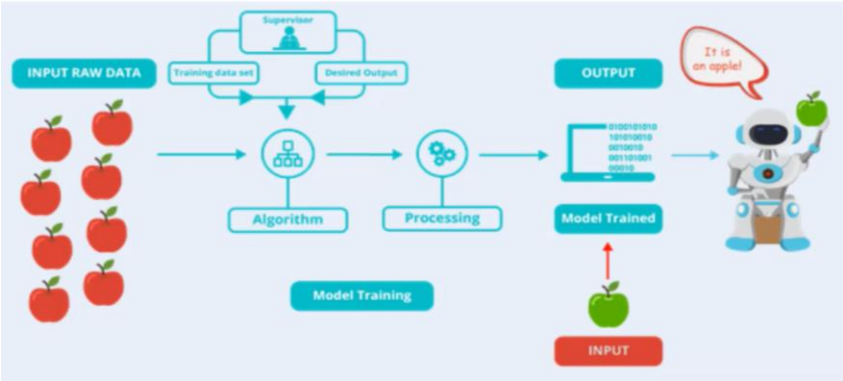
1.1.2. Applications of machine learning

1.1.3. Types of machine learning

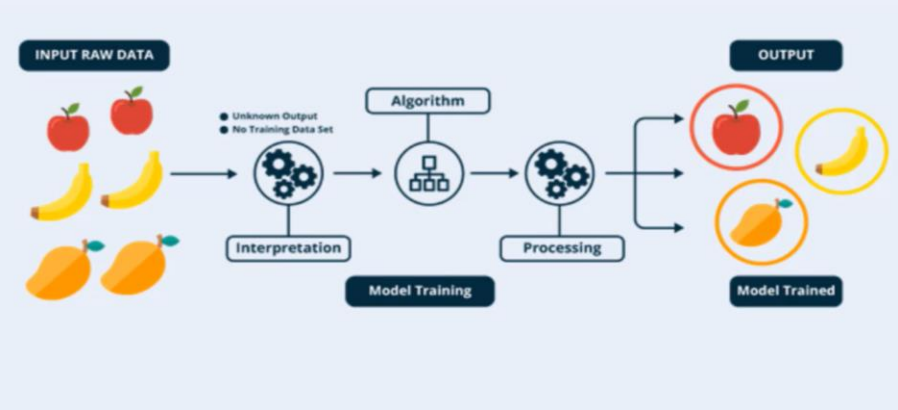


Category	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Format of the data	Input & Output Pairs	Input Only	Input & Reinforcement
How it learns	Task Driven	Data Driven	Reward/Punishment Driven

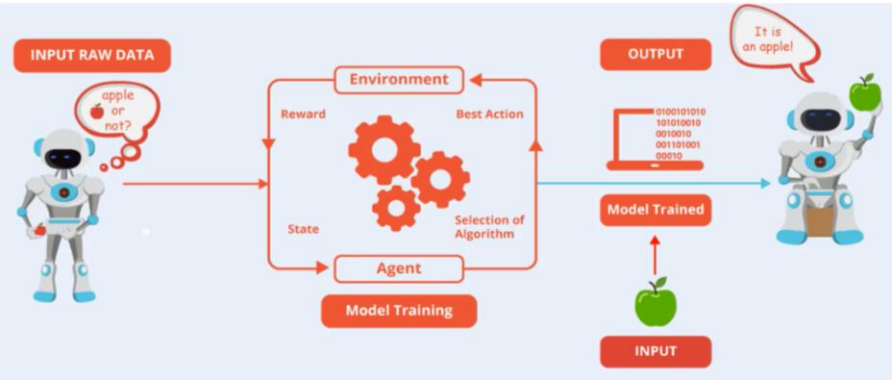
- Supervised Learning



- Unsupervised Learning

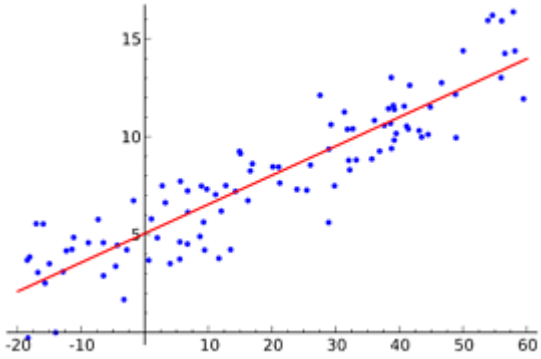


- Reinforcement Learning

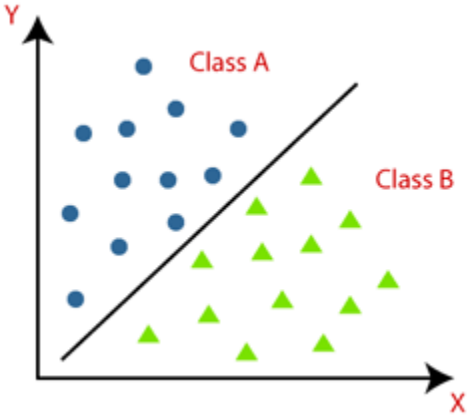


1.2. Types of Output

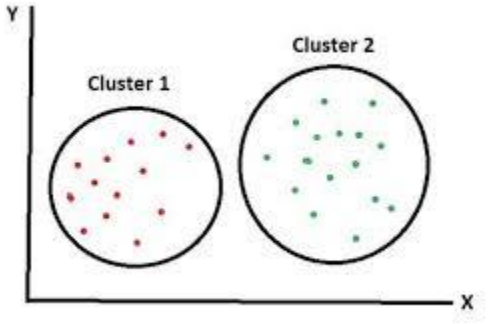
- Regression - It has Limited real-world implementation.



- Classification - it is used mainly in photos classification

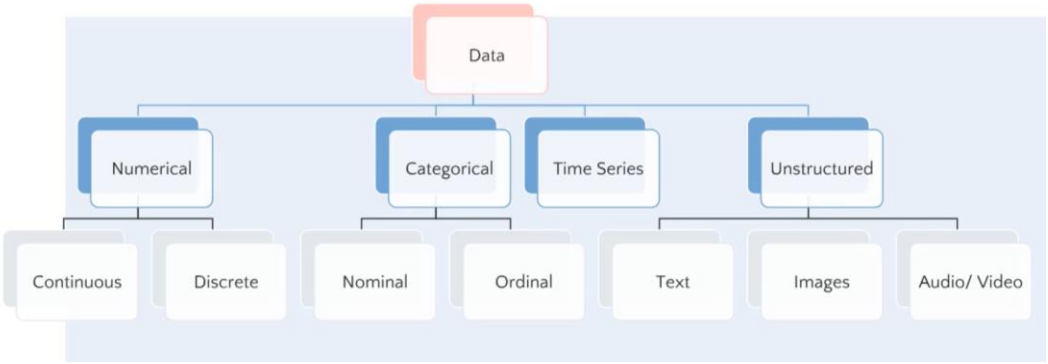


- Clustering- it is a technique where the sample is placed in a cluster where all the samples in cluster are similar to each other. It is supervised and unsupervised type of machine learning. it is generally used in anomalies detection and fraud detection

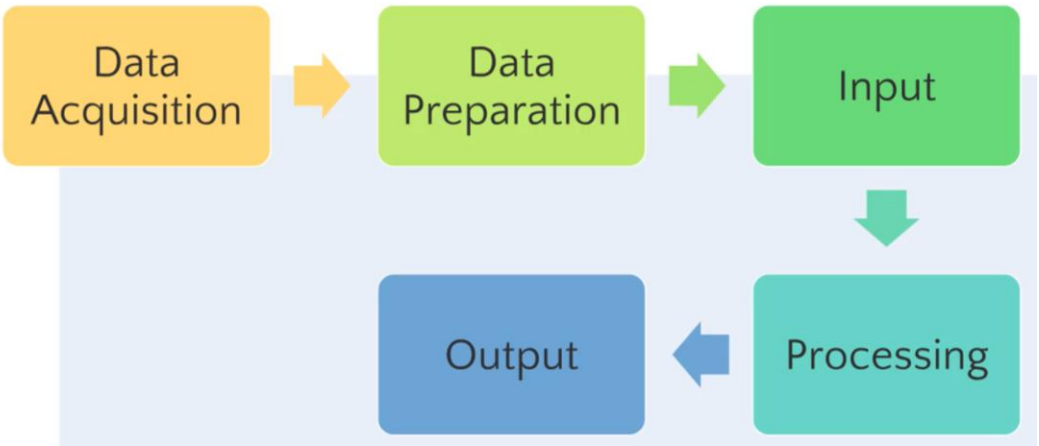


1.3. Data

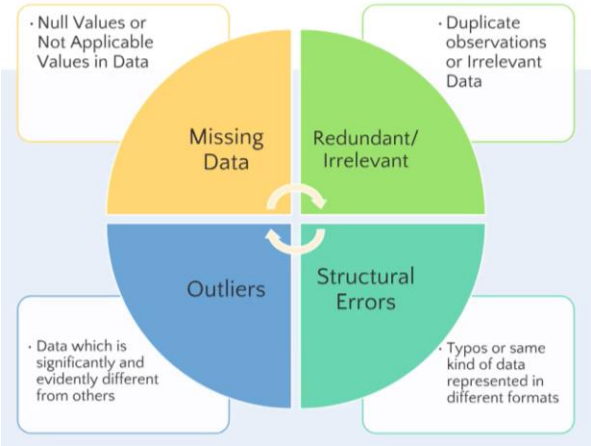
1.3.1 Types of data



1.3.2. Data processing pipeline



1.3.3. Data processing preparation



1.4. Terminologies is used in machine learning

a) Features
- measure property of data object
- used as inputVariable
- choosing distinguishing and independent features is crucial
- example colour size test extra

b) Target
- Expected output of given feature
- value to be predicted using machine learning
- example category/ name of fruit yield of crop

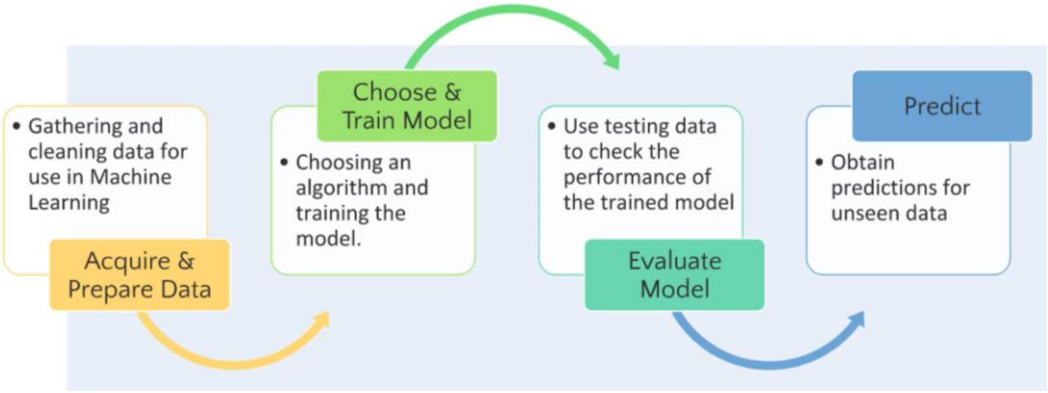
c) Label - one type of target values

d) Model - hypothesis that defines relationship between features and target

e) Training
- The process layer model use data to learn and to predict the output
- data set divided into two parts training data set which obtain standardise model ll and testing data set that test how how well model performs
- expose model to feature and expected targets
- laws relationship between labour and features

f) Prediction
- Apply model to unseen data
- the target / label based on the data

1.5. Machine Learning Pipeline



1.6. Tools used for machine learning

1.6.1. Programming tools

- Python3
  - High-Level
  - Object Oriented Programming
  - Beginner Friendly
  - Data Handling is simpler
  - Open Source Packages specifically for ML
- Anaconda
  - Free & Open Source distribution of Python
  - Packages for Scientific Computing
  - ML & Data Science Libraries out of the box
- Jupyter
  - Web Interface for Python/R Programming
  - Literate Programming □ Easy to read text with code blocks

1.6.2. Data handling in Python

- Numpy
  - Math Library for Python
  - Multi-dimensional array as a data type
  - High Level Mathematical Functions & Array Operations
- Pandas
  - Python Library to handle large datasets
  - Data structures & built-in functions to handle data operations
  - Reading & Writing Data from and to spreadsheets
- Matplotlib
  - Python library for plotting operations
  - Easy to Generate a variety of graphs

1.6.3. Machine learning libraries

- Scikit Learn
  - Python based ML library
  - Libraries for Supervised & Unsupervised ML algorithms
  - Easy to implement ML Pipelines as all the required functions are in-built
  - Built on Numpy, Scipy & Matplotlib

- TensorFlow
  - ML Framework for high-end ML problems
  - Excellent functionality for Deep Learning
  - Parallel Neural Network Training

- Keras
  - High level API built on TensorFlow
  - Rapid NN Prototyping
  - Lacks as many advanced operations as TensorFlow

2. BOSTON HOUSING MACHINE LEARNING MODEL

2.1. Description of problem

- Prediction of Price of houses in various places in Boston
- data set has 506 data points
- 14 columns in which 13 features and one is target price

2.2. Strategy for price prediction

2.3. Import Libraries

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib as mpl
import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from IPython.display import HTML
```

2.4. Steps involved in solving problem with machine learning techniques

i. Data preprocessing
ii. Splitting of data in two parts training data set and testing data set
iii. Define Error
iv. Train the Model
v. Prediction

3. STEP 1: DATA PREPROCESSING

3.1. Load Dataset

In [2]:
```
#load the dataset
boston = load_boston()

#description of dataset
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

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

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 24
4-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference
of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

3.1.1. Put the Data into Pandas Dataframe

```
In [3]:
#Put the Data into Pandas Dataframe
features = pd.DataFrame(boston.data, columns = boston.feature_names)
features
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

3.1.2. Generate a target dataset

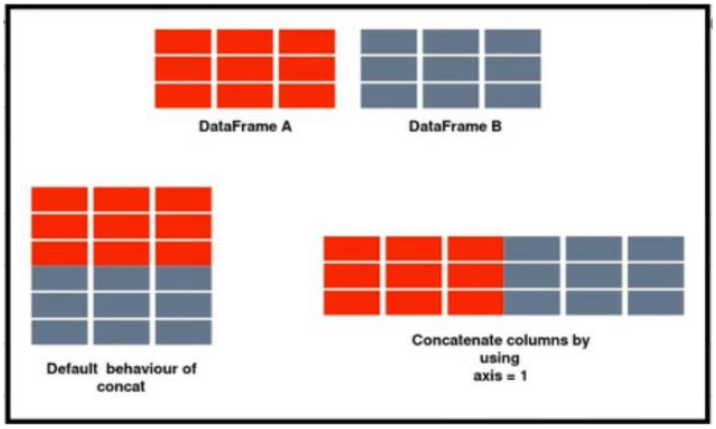
```
In [4]:
#Generate a target dataset
target = pd.DataFrame(boston.target, columns = ['target'])
target
```

Out[4]:

	target
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
...	...
501	22.4
502	20.6
503	23.9
504	22.0
505	11.9

506 rows × 1 columns

3.1.3. Concatenate features and target into a single DataFrame and print it




```
In [7]:
#concatenate features and target into a single DataFrame
#axis 1 makes it concatenate column wise
df = pd.concat([features,target], axis = 1)
#to print the concatenated dataframe
df
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

3.2. Data Visualization of BH dataset

Description is usually the first step that we must perform after importing the data to gather basic insights regarding the data so that we can understand reach off of each variable. Also to understand the overview of distribution of variable values use describe to generate a summary of data set.

'Dataset.describe()' function generate descriptive stats that summaries the central tendency dispersion and shape of data set distribution.

The describe() method computeS the following parameters for each column:

- count - number of rows
- mean - mean of column
- std - standard deviation in column

- max -maximum value in column
- min - minimum value of in Kollam
- 25% - 25 percentile
- 50% - 50 percentile
- 75% - 75 percentile

Percentile or centile is a measure used in statistics indicating the value below which a given percentage of observations in a group of observation Falls

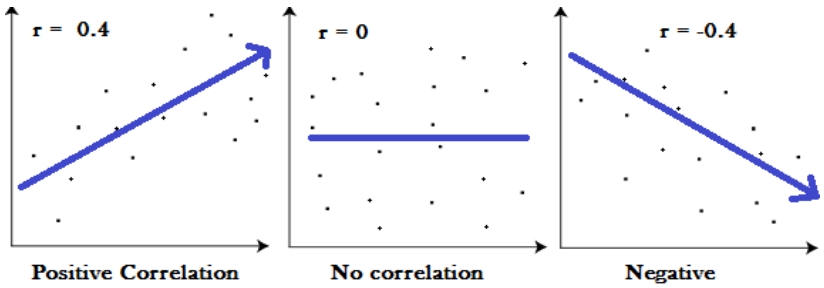
```
In [8]:
1 #use round (decimals=2) to set the precision to 2 decimal places
2 df.describe().round(decimals = 2)
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
count	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00
mean	3.61	11.36	11.14	0.07	0.55	6.28	68.57	3.80	9.55	408.24	18.46	356.67	12.65	22.53
std	8.60	23.32	6.86	0.25	0.12	0.70	28.15	2.11	8.71	168.54	2.16	91.29	7.14	9.20
min	0.01	0.00	0.46	0.00	0.38	3.56	2.90	1.13	1.00	187.00	12.60	0.32	1.73	5.00
25%	0.08	0.00	5.19	0.00	0.45	5.89	45.02	2.10	4.00	279.00	17.40	375.38	6.95	17.02
50%	0.26	0.00	9.69	0.00	0.54	6.21	77.50	3.21	5.00	330.00	19.05	391.44	11.36	21.20
75%	3.68	12.50	18.10	0.00	0.62	6.62	94.07	5.19	24.00	666.00	20.20	396.22	16.96	25.00
max	88.98	100.00	27.74	1.00	0.87	8.78	100.00	12.13	24.00	711.00	22.00	396.90	37.97	50.00

3.3. Correlation between target and attributes

In order to perform linear regression, we want to know which of the feature can be used to predict the target variable there are multiple techniques available to implement this we will do this by exam meaning the statistical metric called correlation.
Correlation describes how closely the value in are dependent on the values of other column basically it is a relationship between two columns .



Positive Correlation-

- change in values of y also shows similar change in values of x
- if x increases y increases
- if x decreases by decreases

Negative correlation

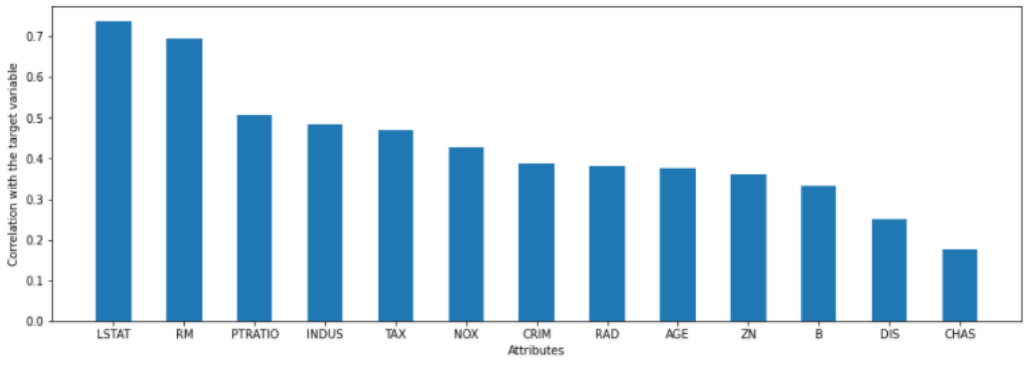
- if x increases by decreases
- if x decreases by increases

Whichever attribute has higher absolute correlation with target that is the attribute we will choose as the independent variable to perform linear regression

Note : an existing of correlation between two variables doesn't mean linear dependence between the variable

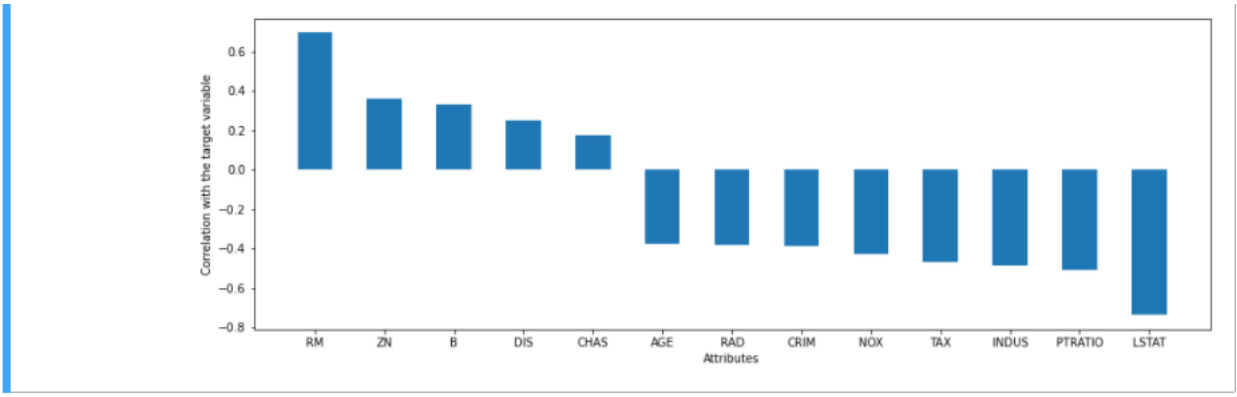
In [9]:

```
#calculate correlation between every column of data
corr = df.corr('pearson')

#take absolute values of correlation
corrs = [ abs(corr[attr]['target']) for attr in list(features)]

#make a list of pairs {(corr,features)}
l = list(zip(corrs, list(features)))

#Sort the list pairs in Reverse/Descending order
#with correlation value as the key of sorting
l.sort(key = lambda x : x[0], reverse = True )


#"nzip" pairs of tool list
#zip(*l) take a list that looks like {[a, b, c], [d, e, f], [g, h, i]}
#add returns {[a, b,c], [d, e, f], [g, h, i]}
corrs,labels = list(zip((*l)))

#Plot correlation with respect to the target variable as a bar graph
index = np.arange(len(labels))
plt.figure(figsize = (15,5))
plt.bar(index, corrs, width= 0.5)
plt.xlabel('Attributes')
plt.ylabel('Correlation with the target variable')
plt.xticks(index, labels)
plt.show()
```



- We observed from the bar graph above, that LSTAT and RM features have highest absolute correlation value with the target.

- If we take Correaltion between Target and Attribures without taking absolute values



- We observed from the bar graph above, that LSTAT and RM features have highest absolute correlation value with the target but LSTAT has negative value and RM has positive.

3.4. Normalization of BH data

Goal of normalization is to change the values such that after the transformation all the values like in a common scale that is 0 to 1 in our case.
Without normalisation it is sometimes difficult to interpret the data.
Normalisation brings all the values in a common scale without distorting the values the data becomes easier to interpret.

Before normalisation values are in arbitrary range
"sklearn" library provides MinMaxScaler method that takes a list of values in any arbitrary range and give them in a list in which values are between 0 to 1.

MinMaxScaler object stores the parameter required to normalise the values therefore we have to use separate scalar objects to perform scaling of different columns so that it we can then use to stored parameters to obtain the scaled values in original representation.
MinMaxScaler provides a method called inverse transform to obtain the values in original representation so that we can compare the predicted values and True values.

The fit transform function computes the minimum and maximum, transform the values and return normalised values. expects values column wise instead of horizontal it has to be vertical.

Therefore she shape values with numpy function reshape to have Values in one column.
We do not have to know it before and we can pass -1 for one of the dimensions and correct value is used for the dimension now pass the reshaped values as a parameter to the fit transform method and get normalised value stored in the variable x

In [11]:
```
#prepare the data for linear regression algorithm
#select the columns to use for x and y respectively
X = df['LSTAT'].values
Y = df['target'].values
```

In [12]:
```
#before normalization
print(Y[:5])
```

```
[24.  21.6 34.7 33.4 36.2]
```

In [13]:
```
#performing normalization
x_scaler = MinMaxScaler()
X = x_scaler.fit_transform(X.reshape(-1,1))
X = X[:, -1]
y_scaler = MinMaxScaler()
Y = y_scaler.fit_transform(Y.reshape(-1,1))
Y = Y[:, -1]
```

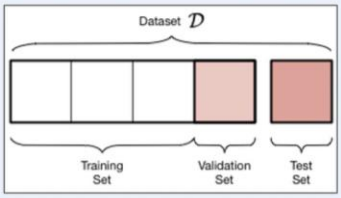
In [14]:
```
#after normalization
print(Y[:5])
```

```
[0.42222222 0.36888889 0.66       0.63111111 0.69333333]
```

4. STEP 2: SPLITTING THE DATASET

4.1. Splitting Dataset

- Training Dataset
  - Sample of data used for learning
  - Create a model that fits the data
- Validation Dataset
  - Sample of data used to evaluate a model fit and tune the model, if necessary
  - Model "sees" the data, but does not "learn" from it
- Test Dataset
  - Sample of data to evaluate the final model fit
  - Unbiased evaluation, unlike in the case of validation dataset



The model is initially fitted on training data set that is a set of examples used in in to ft the parameters of the model. the model is trained on training data set using a supervised learning method.
In practice the training data set consists of pair of set of features and corresponding target label. the current model Run with training data set and produces a result which is then compared with the target. based on the result of comparison and specific learning algorithm being used the parameters of the model are adjusted.
The fitted model is used to make prediction based on second data set called validation data set. it provides an unbiased evaluation of a model fit on training data set by tuning the model. hence the model occasionally sees this data but never does it learn from this. we use validation data set result and update higher level parameters of model. it affects the model but in a indirect way.
Test data set is used to provide unbiased evaluation of the final model with on the train data set. it is only used once the model is fully trained using training and validation data set.

The training set contains on own output and the model long from this data in order to be generalized to other data later on.
We have the test data set in order to test our models prediction on the subset

Training data set is usually divided in 2 parts- validation data set which usually is 10 to 15% of the total data set and training data set which is the rest.

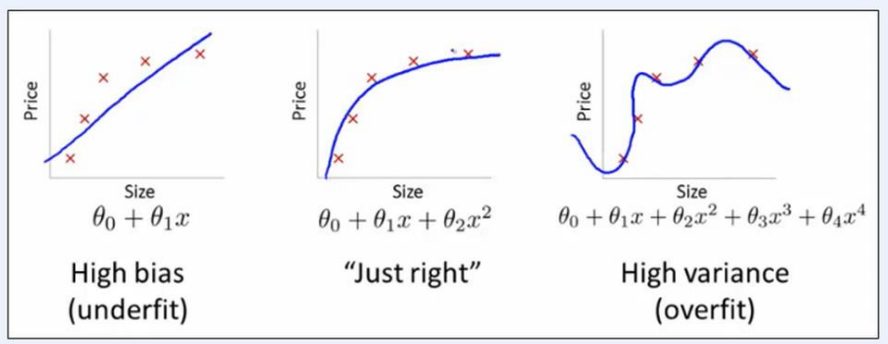
4.2. Overfitting and Underfitting

Overfitting

- When a model Learns the detail and noise in the training data set to the extent that it negatively impact the performance of the model to the new data
- model that fits the data too well
- Results in excessively complicated model
- doesn't apply to new data

Underfitting

- When the machine learning algorithm can capture the trend of the data
- model doesn't fill well enough
- often results in a accessibility simple model

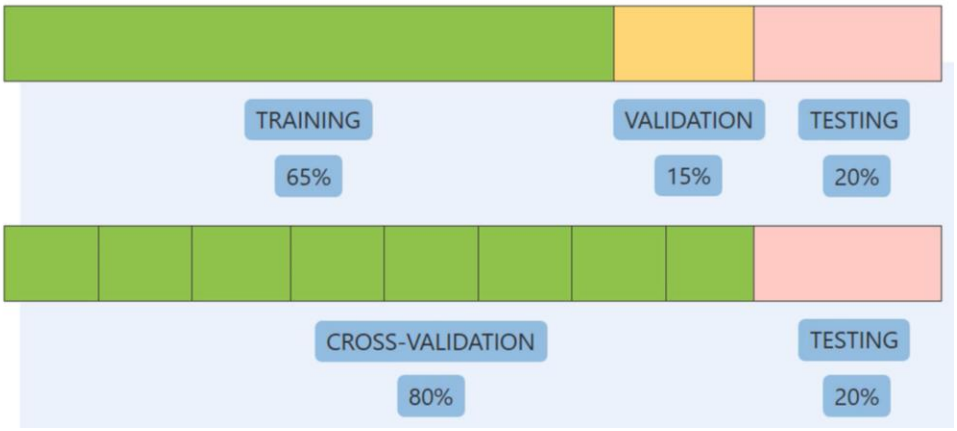


4.3. Steps To avoid overfitting

a) using validation
b) fitting multiple models
c) using Cross Validation-

- Similar to train test split but it is applied to more subsets

- meaning that we split our data into k subsets and train ke -1 subset and we hold last subset for test data set



Split Testing data set randomly choosing certain percentage of samples from the data set and exclude them from the data shown them during training phase that is training data
Note important to choose randomly to ensure that data is uniformly distributed and we don't introduce any bias case.

'sklearn' provides a function called train test split
the parameter to this function are data set and fraction of data set to be considered for the testing data set
It will return four values-

- training set of features X train
- Testing set of features X test
- training set of target by train
- testing set of target buy test

```
In [15]:
1 #splitting and sepeartion of testing dataset
2 #0.2 indicates 20% of the data is randomly sampled as testing data
3 xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size = 0.2)
```

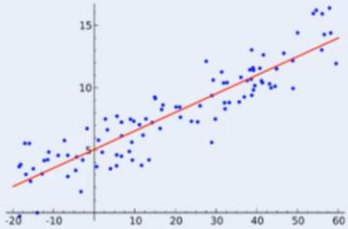
5. LINEAR REGRESSION IN MACHINE LEARNING

5.1. Overview

- Origin in statistics
- understand relationship between input and output numerical variable
- supervised learning
- predicted output is continuous rather than discrete

- Linear Model → Assumes a linear relationship between the input variables and the single output variable
- Target/Label = Linear Combination of Features
- Hypothesis:

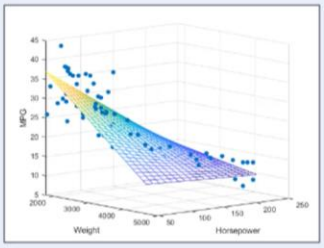
$$Y = m * X + c$$



5.2. Types of linear regression

- Simple Regression
  - Target/Label is dependent on a single feature

$$Y = m * X + c$$



- Multivariable Regression
  - Target/Label is dependent on multiple features

$$f(x, y, z) = w_1 * x + w_2 * y + w_3 * z + b$$

Learn

5.3. Learning in Linear regression

5.4. Prediction in linear regression

- Plug in Inputs in the Linear Model to obtain the Target value

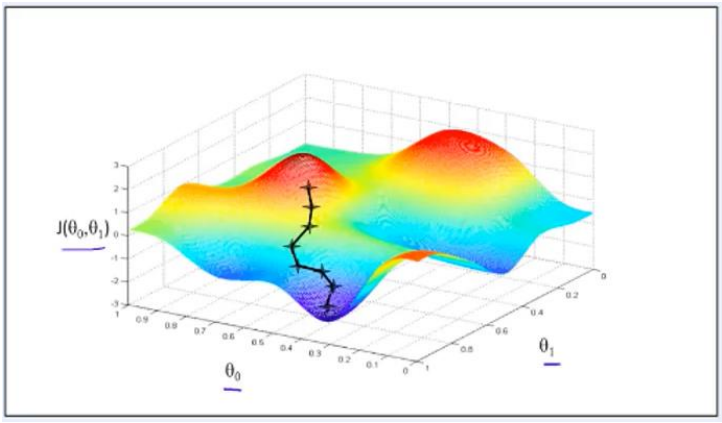
$$Y' = M * X + C$$

6. GRADIENT DESCENT IN MACHINE LEARNING

6.1. Concepts and implementation

- Optimising algorithm to minimise function

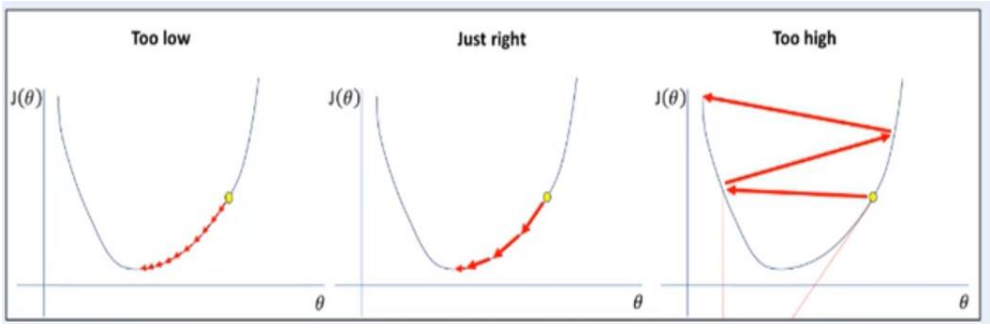
- iteratively move in the direction of steepest descent
- steepest descent - largest negative gradient
- impossible to visualise gradient in a space consisting more than three dimensions



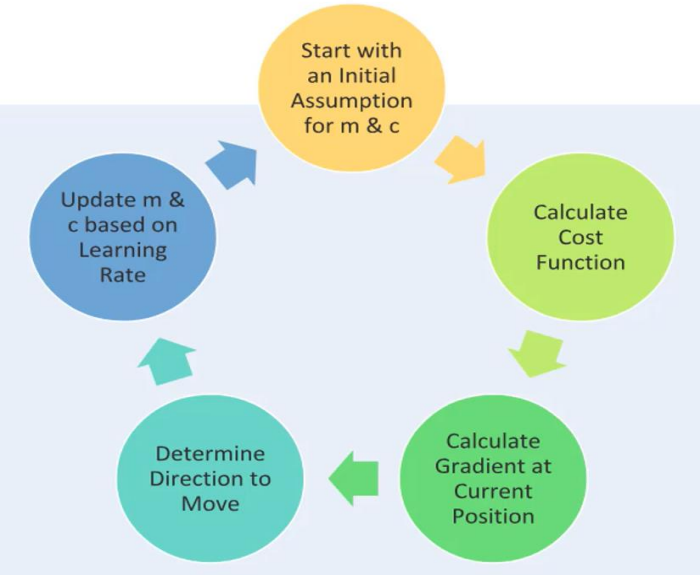
Learning rate- size of iterative step

- too low - require many steps although accurate
- Too high - might lead to divergence

Cost function- the function to be minimized in machine learning



6.2. Steps To compute gradient descent



6.3. Features of gradient descent

- Model Hypothesis → Linear Fit

$$Y = m * X_i + c$$

- Cost Function → Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y - Y_i)^2$$

- Apply Gradient Descent to

$$f(m,c) = \frac{1}{n}\sum_{i=1}^{n}((m * X_i + c) - Y_i)^2$$

- Gradient Calculation

$$f'(m,c) = \begin{bmatrix} \frac{\partial f}{\partial m} \\ \frac{\partial f}{\partial c} \end{bmatrix} = \begin{bmatrix} \frac{\Delta m}{\Delta c} \end{bmatrix} = \begin{bmatrix} \frac{1}{n}\sum_{i=1}^{n} 2 * X_i * ((m * X_i + c) - Y_i) \\ \frac{1}{n}\sum_{i=1}^{n} 2 * ((m * X_i + c) - Y_i) \end{bmatrix}$$

Updated Values of m & c based on Learning Rate

$$m = m - \Delta m * \alpha$$

$$c = c - \Delta c * \alpha$$

6.4. Programming logic of gradient descent

there are three functions which have to be defined

- Update function
- error function
- gradient Descent function

6.4.1. Update function

- Purpose
  - Calculate Gradient at Current Position
  - Update Values of m & c
- Parameters
  - Current Values of m & c
  - Attribute Values (x)
  - Target Values (t)
  - Learning Rate
- Return
  - Updated Values of m & c

In [16]:

```
#Define Update Function
def update (m, x, c, t, learning_rate):
    grad_m = sum(2 * ((m * x + c) - t) * x)
    grad_c = sum(2 * ((m * x + c) - t))
    m = m - grad_m * learning_rate
    c = c - grad_c * learning_rate
    return m, c
```

6.4.2. Error function

- Purpose
  - Calculate the cost function at the current position
- Parameters
  - Current Values of m & c
  - Attribute Values (x)
  - Target Values (t)
- Return
  - Return the error value

In [17]:
```
#define Error function
def error (m, x, c, t):
    N = x.size
    e = sum(((m * x + c) - t) ** 2)
    return e * 1 / (2 * N)
```

6.4.3. Gradient Descent function

- Purpose
  - Apply Gradient Descent on the Cost Function
  - Obtain the best fit values for m & c
- Parameters
  - Initial Values of m & c
  - Attribute Values (x)
  - Target Values (t)
  - Learning Rate
  - Number of Iterations
  - Error Threshold
- Return
  - Final Values of m & c, Lists of all intermittent error, m & c values

In [18]:
```
#Define Gradient Descent function
def gradient_descent(init_m, init_c, x, t, learning_rate, iterations, error_threshold):
    m = init_m
    c = init_c
    error_values = list()
    mc_values = list()

    for i in range(iterations):
        e = error(m, x, c, t)
        if e < error_threshold:
            print('Error threshold less than threshold. Stop the gradient descent.')
            break
        error_values.append(e)
        m, c = update(m, x, c, t, learning_rate)
        mc_values.append((m,c))
    return m, c, error_values, mc_values
```

6.5. Running Gradient Descent function

First we have to define the Hyperparameters.
Hyper parameters are the parameters that may change and varied to observe the computation versus accuracy traders

- Learning rate- Increasing the learning rate reduces the convergence time but if the learning rate is too high the model will overshoot the minima for this problem set the value below 0.0025 otherwise it will cause overflow in weight values
- Iterations- number of iterations must be large enough to allow the model to converge to a minima but if it's too large then the model become too specific to the training data thus causing overfitting that the model memorizes the data instead of burning the data
- Errors threshold- this value can be set to a maximum value of error that is acceptable when the error values goes below the threshold the gradient Descent is stopped
- Initial values for this problem where our objective is to determine the line which gives the least error and does not matter what the initial values you provide but for non convex Optimisation problems initial value affects the learning rate.

In [19]:
```
%%time
init_m = 0.9
init_c = 0
learning_rate = 0.001
iterations = 250
error_threshold = 0.001

m, c, error_values, mc_values =
 gradient_descent(init_m, init_c, xtrain, ytrain, learning_rate, iterations, error_threshold)
```

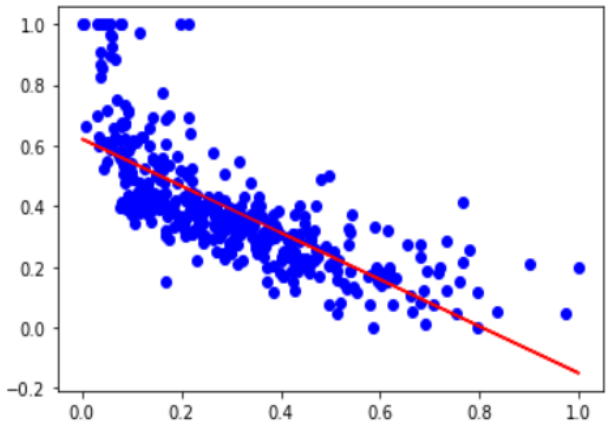
```
Wall time: 167 ms
```

7. VISUALIZATION OF THE LEARNING PROCESS

Use error values array to plot the values.
This is to observe how error changes during training model is getting better the error values must fall over regression line

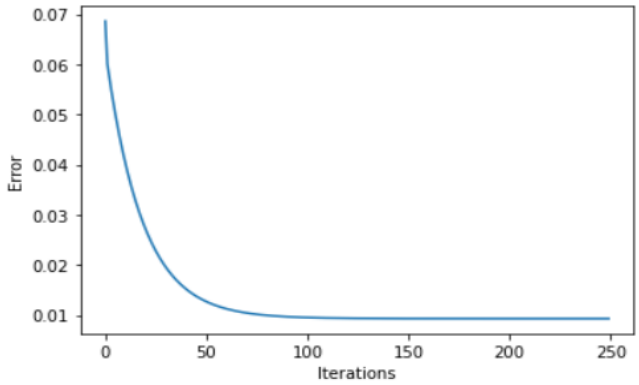
i. Plot Regression line- against the training data set to visualize what the line looks like for the training data set.

```
In [20]:
1 #Plotting Regression Line upon the traing dataset
2 plt.scatter(xtrain, ytrain, color = 'b')
3 plt.plot(xtrain, (m * xtrain + c), color = 'r')

Out[20]: [<matplotlib.lines.Line2D at 0x1955fc1d970>]
```



ii. Plot Error Values- shows how the error drops over time

```
In [21]:
1 #Plotting and Visualization of Error values
2 plt.plot(np.arange(len(error_values)), error_values)
3 plt.ylabel('Error')
4 plt.xlabel('Iterations')

Out[21]: Text(0.5, 0, 'Iterations')
```



8. STEP 4: MODEL TRAINING VISUALIZATION

As the number of iterations increases, changes in the line are less noticable
In order to reduce the processing time for the animation, it is advised to choose values

```
In [22]:
#As the number of iterations increases, changes in the line are less noticable
#In order to reduce the processing time for the animation, it is advised to choose values
mc_values_anim = mc_values[0 : 250 : 5]
```

8.1. Initialize the variables

```
In [23]:
#initialize figures and axis object
fig, ax = plt.subplots()

#initialize line object ( the same object is used by init and update function)
ln, = plt.plot([], [], 'ro-', animated = True)
```

X and Y coordinates are empty list
generating and animation requires to function Ban to initialise the state of the graph and other to update each function with new data
update function is called before drawing each frame
inside the update function we will send new endpoints for the line

8.2. Defining the Init function

- Purpose
  - The callback function initialize the graph.
  - Draw the training data such that it is in the background and set the limits of the x and y axes
- Parameters:
  - None. This function does not take any parameters
- Returns:
  - The line object. To be returned because the FuncAnimation class requires that this function returns the line object

```
#define the init function
def init():
    plt.scatter(xtrain, ytrain, color = 'g')
    ax.set_xlim(0, 1.0)
    ax.set_ylim(0, 1.0)
    return ln,
```

8.3. Defining the update function

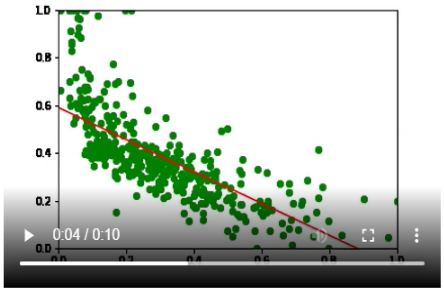
- Purpose
  - To update each frame.
  - Using the m and c values we compute the coordinates for the end points of the regression line
- Parameters
  - Frame number. The frame number will be used to index the mc_values array to access m and c values to compute the coordinates for the new frame.
- Return
  - The line object. To be returned because the FuncAnimation class requires that this function returns the line object

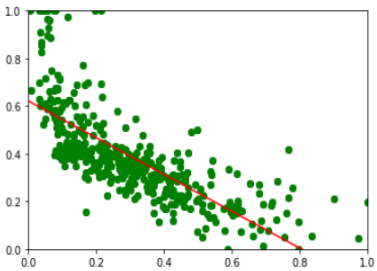
```
#define the update function
def update_frame(frame):
    m, c = mc_values_anim[frame]
    x1, y1 = -0.5, m * -.5 + c
    x2, y2 = 1.5, m * 1.5 + c
    ln.set_data([x1,x2], [y1,y2])
    return ln,
```

8.4. Declare the animated object and Generate the video of the animation

```
#Declare the animated object
#Set the callbacks and number of frames
anim = FuncAnimation(fig, update_frame, frames = range(len(mc_values_anim)), init_func = init, blit = True)

#Generate the video of the animation
HTML(anim.to_html5_video())
```

Out[23]:





9. STEP5- PREDICTION OF PRICES

9.1. Steps to be performed

1. *Compute the predicted values for each x value using the line equation with the m and c values learned from gradient descent.*
2. *Reshape predicted, xtest, and ytest arrays from a row to a column*
3. *Scale the predicted, xtest and ytest arrays and store then as predicted_scaled, xtest_scaled, ytest_scaled*
4. *Scaling adds an extra dimension to each of the arrays. Therefore remove it with a slicing operation.*

9.2. Calculate the predicted value on the test set as a vectorized operation

In [24]:
```
#Calculate the predicted value on the test set as a vectorized operation
predicted  = (m * xtest) + c
```

9.3. Compute MSE for the predicted values on the training set

In [25]:
```
#Compute MSE for the predicted values on the training set
mean_squared_error(ytest, predicted)
```

Out[25]:
```
0.020440768433733922
```

9.4. Put xtest, ytest and predicted values into a single DataFrame , so that we can see the predicted values alongside the testing data

In [26]:
```
#Put xtest, ytest and predicted values into a single DataFrame
#so that we can see the predicted values alongside the testing data
p = pd.DataFrame(list(zip(xtest, ytest, predicted)),
                 columns = ['x', 'target_y', 'predicted_y'])
p.head()
```

Out[26]:

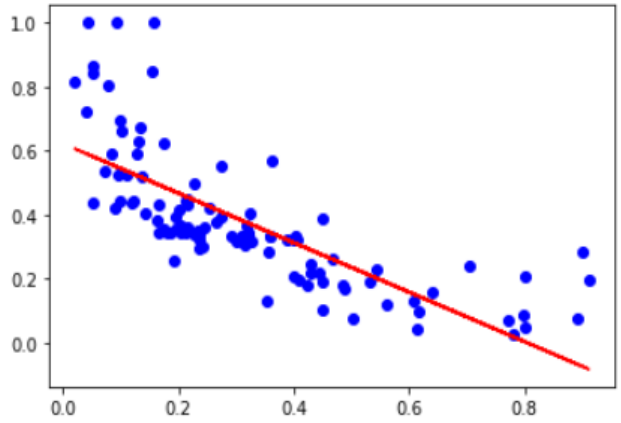
	x	target_y	predicted_y
0	0.212472	0.344444	0.457499
1	0.127759	0.591111	0.522896
2	0.215508	0.446667	0.455156
3	0.161976	0.384444	0.496482
4	0.891004	0.075556	-0.066315

9.5. Plot the predicted values against the target values

- Predicted values are represented by red colour line
- Target values are represented by blue colour line

In [27]:
```
1 plt.scatter(xtest, ytest, color = 'b')
2 plt.plot(xtest, predicted, color = 'r')
```

Out[27]: [<matplotlib.lines.Line2D at 0x1955feff490>]



9.6. Revert normalisation to obtain the predicted price of house in 1000 dollars. And find the final prediction prices

The predicted value are in range of 0-1. This is not very useful to us when we want to obtain the price. Use inverse transform to scale the values back to the original representation.

In [28]:

```
#Reshape  To shape  that is required by the scalar
#i.e. in vertical orientation(column) instead of horizontal orientation(row)
xtest = xtest.reshape(-1, 1)
ytest = ytest.reshape(-1, 1)
predicted = predicted.reshape(-1, 1)

#Perform inverse transformation function
xtest_scaled = x_scaler.inverse_transform(xtest)
ytest_scaled = y_scaler.inverse_transform(ytest)
predicted_scaled = y_scaler.inverse_transform(predicted)

#This is to remove the extra dimension
xtest_scaled = xtest_scaled[:, -1]
ytest_scaled = ytest_scaled[:, -1]
predicted_scaled = predicted_scaled[:, -1]

p = pd.DataFrame(list(zip(xtest_scaled, ytest_scaled, predicted_scaled)),
                 columns = ['x', 'target_y', 'predicted_y'])
p = p.round(decimals = 2)
p.head()
```

Out[28]:

	x	target_y	predicted_y
0	9.43	20.5	25.59
1	6.36	31.6	28.53
2	9.54	25.1	25.48
3	7.60	22.3	27.34
4	34.02	8.4	2.02

REFERENCES

- www.skyfilabs.com

Github link for Project code-

https://github.com/rohan1110/Machine-Learning-project-using-python-BostonHousing-