WINTER INTERNSHIP REPORT

ON

<u>“AI/Machine/Deep Learning”</u>

AREA OF ONLINE INTERNSHIP	AI/Machine/Deep Learning
Intern Name	**Abhishek Kushwah**
Branch	**Electronics and Communication Engineering**
Name Of Institution	**Indian Institute of Technology, Indore**
Faculty Mentor Name	**Dr. Vimal Bhatia**
Duration	**01/01/2022 to 01/02/2022**
Date of Submission	**02/2/2022**

➢ CONTENTS:

Introduction

Machine Learning is the science of getting computers to learn without being explicitly programmed. It is closely related to computational statistics, which focuses on making prediction using computer. In its application across business problems, machine learning is also referred as predictive analysis. Machine Learning is closely related to computational statistics. Machine Learning focuses on the development of computer programs that can access data and use it to learn themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Deep learning vs. Machine learning

Machine learning and deep learning are both types of AI. In short, machine learning is AI that can automatically adapt with minimal human interference. Deep learning is a subset of machine learning that uses artificial neural networks to mimic the learning process of the human brain.

Take a look at these key differences before we dive in further.

Machine learning	Deep learning
It is a subset of AI	And it’s a subset of machine learning
Can train on smaller data sets	Requires large amounts of data
Requires more human intervention to correct and learn	Learns on its own from environment and past mistakes
Shorter training and lower accuracy	Longer training and higher accuracy
Makes simple, linear correlations	Makes non-linear, complex correlations

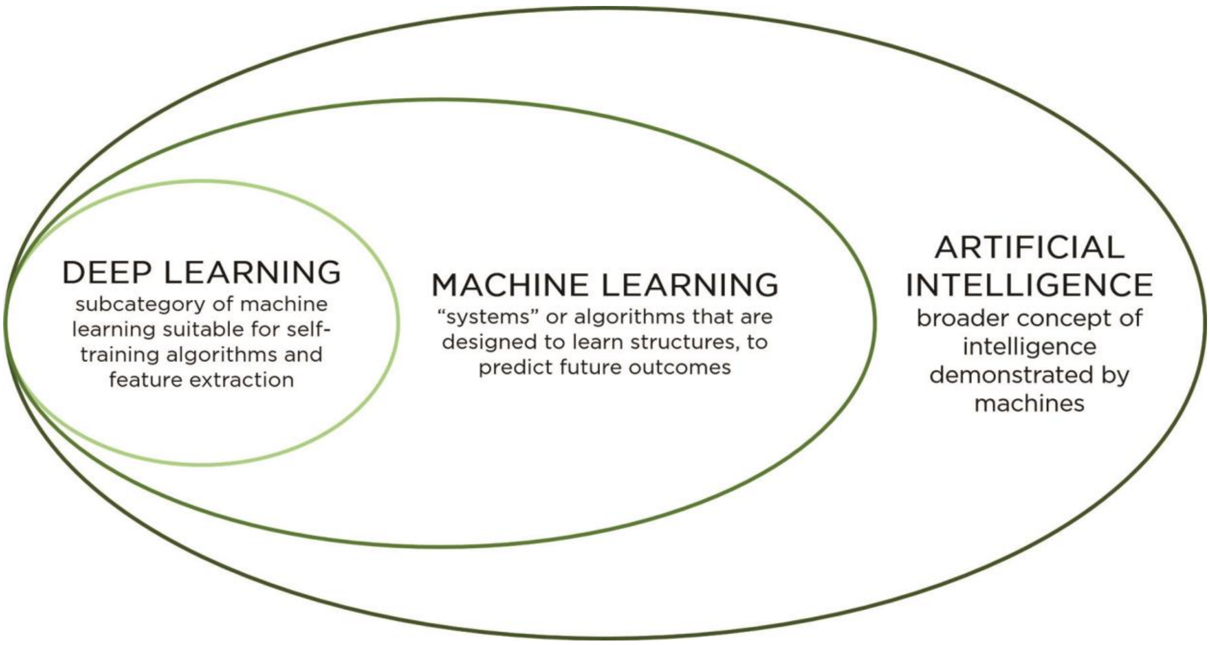
Machine learning	Deep learning
Can train on a CPU (central processing unit)	Needs a specialized GPU (graphics processing unit) to train

Artificial intelligence (AI)

At its most basic level, the field of artificial intelligence uses computer science and data to enable problem solving in machines.

While we don't yet have human-like robots trying to take over the world, we do have examples of AI all around us. These could be as simple as a computer program that can play chess, or as complex

as an algorithm that can predict the RNA structure of a virus to help develop vaccines.



Types of Machine Learning

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve. Broadly Machine Learning can be categorized into four categories.

I. Supervised Learning

II. Unsupervised Learning

III. Reinforcement Learning

IV. Semi-supervised Learning

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly.

Supervised Learning

Supervised Learning is a type of learning in which we are given a data set and we already know what are correct output should look like, having the idea that there is a relationship between the input and output.

Basically, it is learning task of learning a function that maps an input to an output based on example inputoutput pairs. It infers a function from labeled training data consisting of a set of training examples. Supervised learning problems are categorized

Unsupervised Learning

Unsupervised Learning is a type of learning that allows us to approach problems with little or no idea what our problem should look like. We can derive the structure by clustering the data based on a relationship among the variables in data. With unsupervised learning there is no feedback based on prediction result.

Basically, it is a type of self-organized learning that helps in finding previously unknown patterns in data set without pre-existing label.

Reinforcement Learning

Reinforcement learning is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best.

Semi-Supervised Learning

Semi-supervised learning fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy.

Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

Data Preprocessing, Analysis & Visualization

Machine Learning algorithms don’t work so well with processing raw data. Before we can feed such data to an ML algorithm, we must pre-process it. We must apply some transformations on it. With data pre-processing, we convert raw data into a clean data set. To perform data this, there are 7 techniques -

1. Rescaling Data

For data with attributes of varying scales, we can rescale attributes to possess the same scale. We rescale attributes into the range 0 to 1 and call it normalization. We use the MinMaxScaler class from scikitlearn. This gives us values between 0 and 1.

2. Standardizing Data

With standardizing, we can take attributes with a Gaussian distribution and different means and standard deviations and transform them into a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

3. Normalizing Data

In this task, we rescale each observation to a length of 1 (a unit norm). For this, we use the Normalizer class.

4.Binarizing Data

Using a binary threshold, it is possible to transform our data by marking the values above it 1 and those equal to or below it, 0. For this purpose, we use the Binarizer class.

5. Mean Removal

We can remove the mean from each feature to center it on zero.

6. One Hot Encoding

When dealing with few and scattered numerical values, we may not need to store these. Then, we can perform One Hot Encoding. For k distinct values, we can transform the feature into a k-dimensional vector with one value of 1 and 0 as the rest values.

7. Label Encoding

Some labels can be words or numbers. Usually, training data is labelled with words to make it readable.Label encoding converts word labels into numbers to let algorithms work on them.

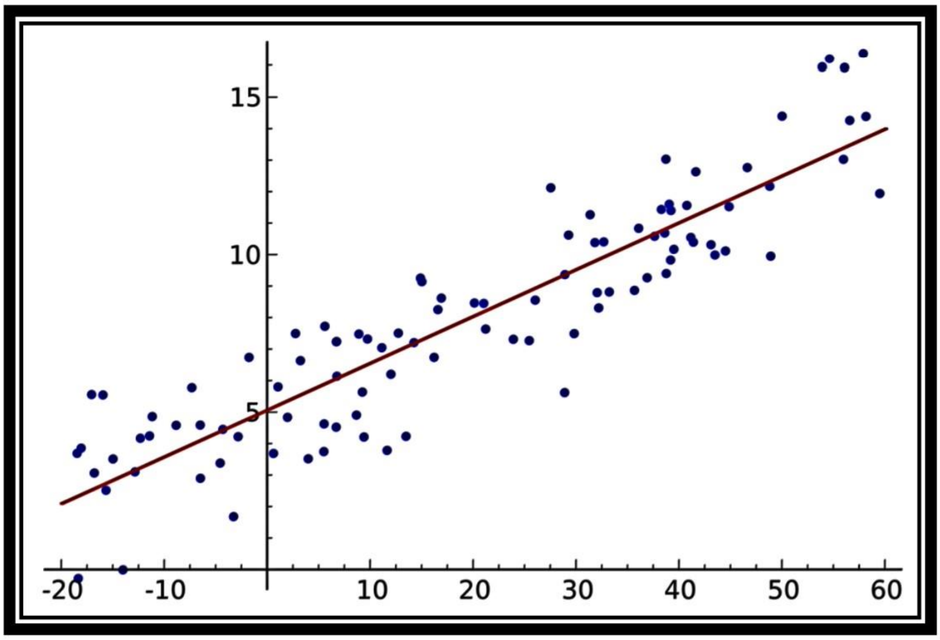
Machine Learning Algorithms

There are many types of Machine Learning Algorithms specific to different use cases. As we work with datasets, a machine learning algorithm works in two stages. We usually split the data around 20%-80% between testing and training stages. Under supervised learning, we split a dataset into a training data and test data in Python ML. Followings are the Algorithms of Python Machine Learning -

1. Linear Regression

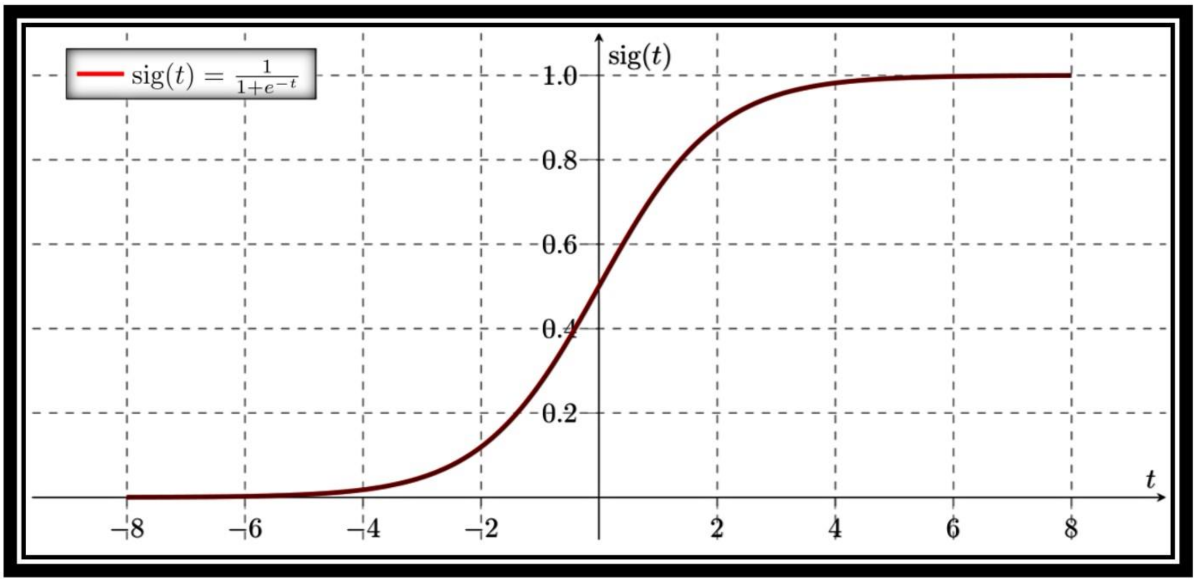
Linear regression is one of the supervised Machine learning algorithms in Python that observes continuous features and predicts an outcome. Depending on whether it runs on a single variable or on many features, we can call it simple linear regression or multiple linear regression.

This is one of the most popular Python ML algorithms and often under-appreciated. It assigns optimal weights to variables to create a line ax+b to predict the output. We often use linear regression to estimate real values like a number of calls and costs of houses based on continuous variables. The regression line is the best line that fits Y=a*X+b to denote a relationship between independent and dependent variables.

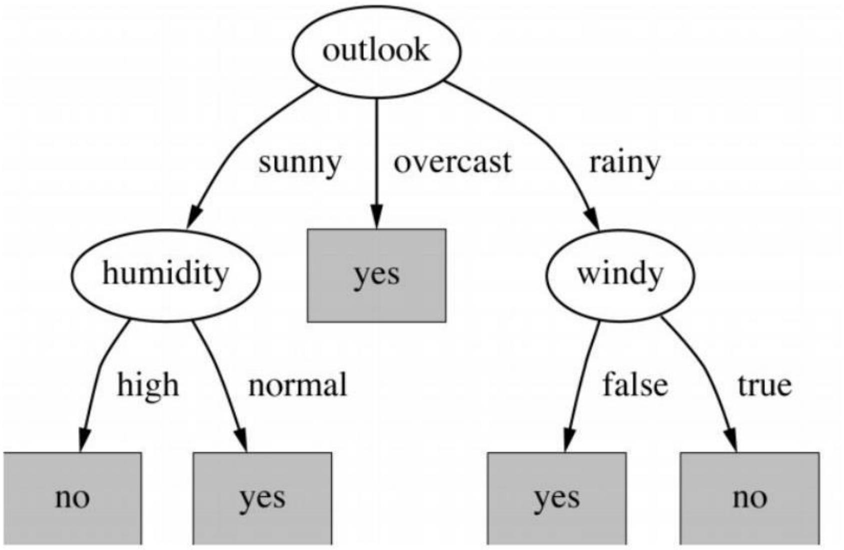


2. Logistic Regression-

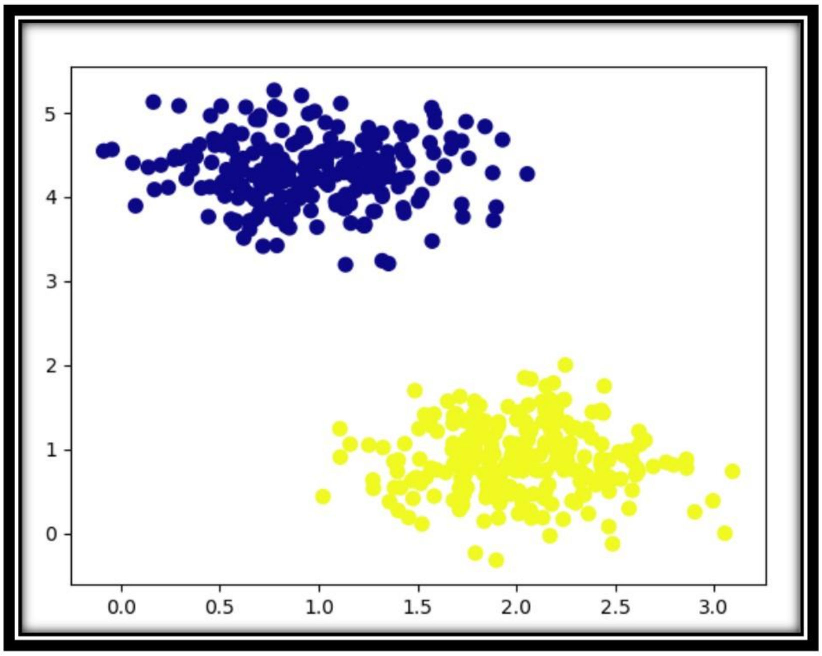
Logistic regression is a supervised classification is unique Machine Learning algorithms in Python that finds its use in estimating discrete values like 0/1, yes/no, and true/false. This is based on a given set of independent variables. We use a logistic function to predict the probability of an event and this gives us an output between 0 and 1. Although it says ‘regression’, this is actually a classification algorithm. Logistic regression fits data into a logit function and is also called logit regression.



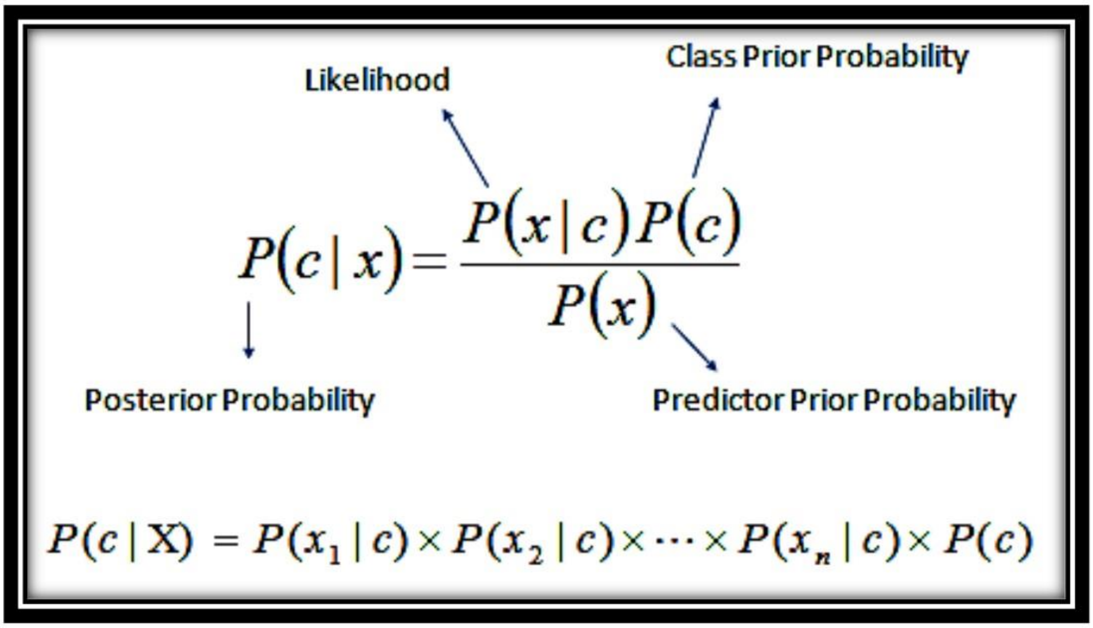
3. Decision Tree - A decision tree falls under supervised Machine Learning Algorithms in Python and comes of use for both classification and regression- although mostly for classification. This model takes an instance, traverses the tree, and compares important features with a determined conditional statement. Whether it descends to the left child branch or the right depends on the result. Usually, more important features are closer to the root. Decision Tree, a Machine Learning algorithm in Python can work on both categorical and continuous dependent variables. Here, we split a population into two or more homogeneous sets. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.



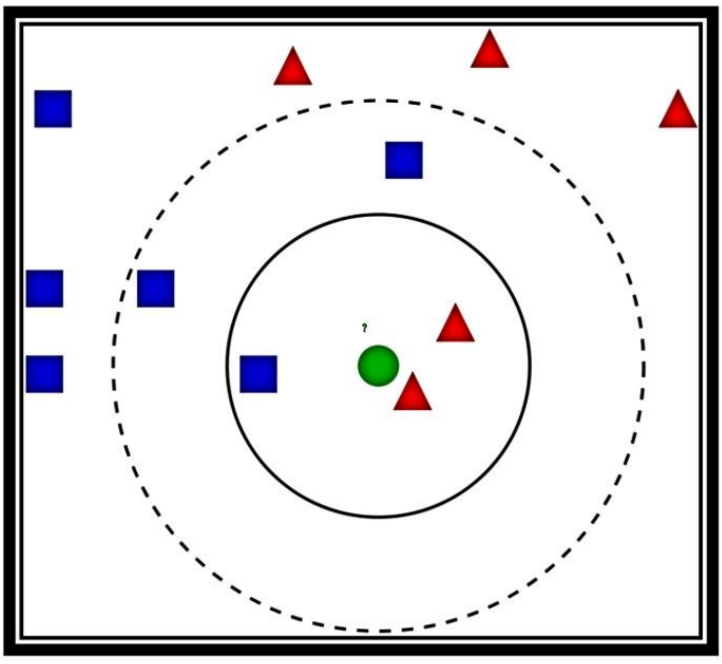
4. Support Vector Machine (SVM)- SVM is a supervised classification is one of the most important Machines Learning algorithms in Python, that plots a line that divides different categories of your data. In this ML algorithm, we calculate the vector to optimize the line. This is to ensure that the closest point in each group lies farthest from each other. While you will almost always find this to be a linear vector, it can be other than that. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. When data are unlabeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups.



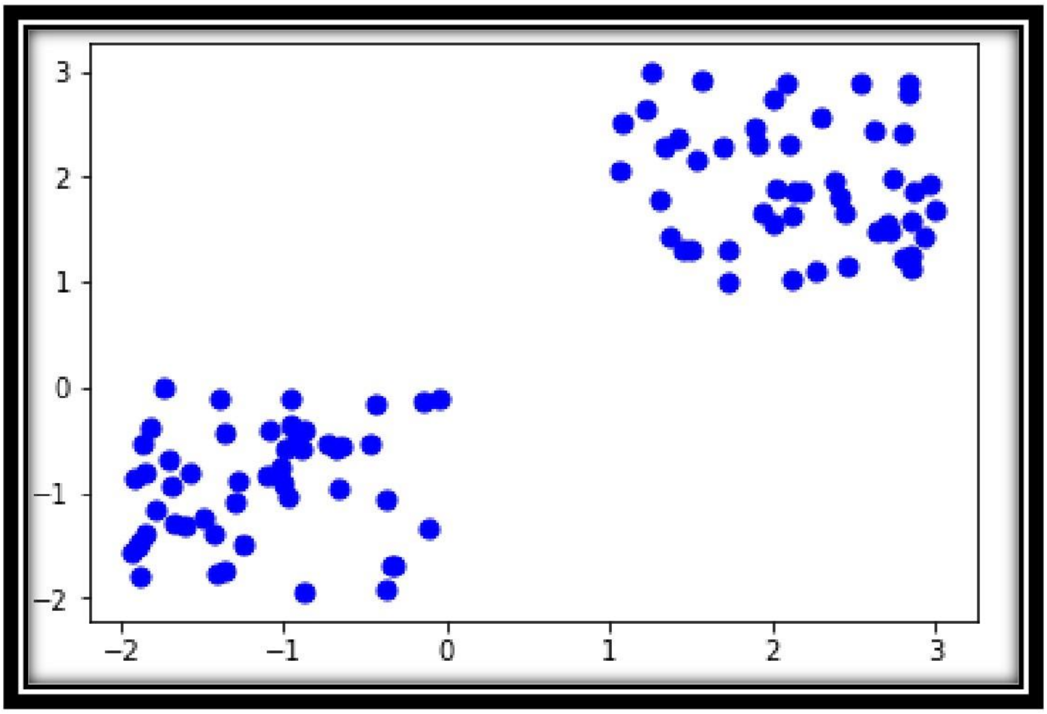
5. Naïve Bayes Algorithm - Naive Bayes is a classification method which is based on Bayes' theorem. This assumes independence between predictors. A Naive Bayes classifier will assume that a feature in a class is unrelated to any other. Consider a fruit. This is an apple if it is round, red, and 2.5 inches in diameter. A Naive Bayes classifier will say these characteristics independently contribute to the probability of the fruit being an apple. This is even if features depend on each other. For very large data sets, it is easy to build a Naive Bayesian model. Not only is this model very simple, it performs better than many highly sophisticated classification methods. Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.



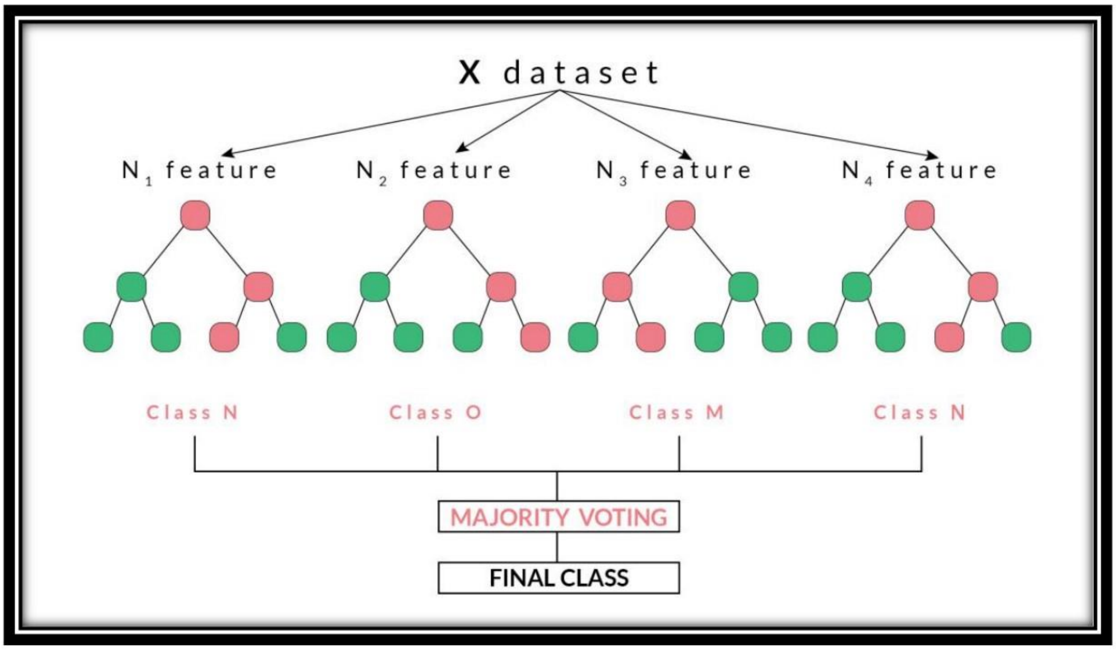
<u>6. kNN Algorithm -</u> This is a Python Machine Learning algorithm for classification and regression- mostly for classification. This is a supervised learning algorithm that considers different centroids and uses a usually Euclidean function to compare distance. Then, it analyze the results and classifies each point to the group to optimize it to place with all closest points to it. It classifies new cases using a majority vote of k of its neighbours. The case it assigns to a class is the one most common among its K nearest neighbours. For this, it uses a distance function. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. k-NN is a special case of a variablebandwidth, kernel density "balloon" estimator with a uniform kernel.



7. K-Means Algorithm - k-Means is an unsupervised algorithm that solves the problem of clustering. It classifies data using a number of clusters. The data points inside a class are homogeneous and heterogeneous to peer groups. k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. k-means clustering is rather easy to apply to even large data sets, particularly when using heuristics such as Lloyd's algorithm. It often is used as a pre-processing step for other algorithms, for example to find a starting configuration. The problem is computationally difficult (NP-hard). k-means originates from signal processing, and still finds use in this domain. In cluster analysis, the k-means algorithm can be used to partition the input data set into k partitions (clusters). k-means clustering has been used as a feature learning (or dictionary learning) step, in either (semi-)supervised learning or unsupervised learning.



8. Random Forest - A random forest is an ensemble of decision trees. In order to classify every new object based on its attributes, trees vote for class- each tree provides a classification. The classification with the most votes wins in the forest. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

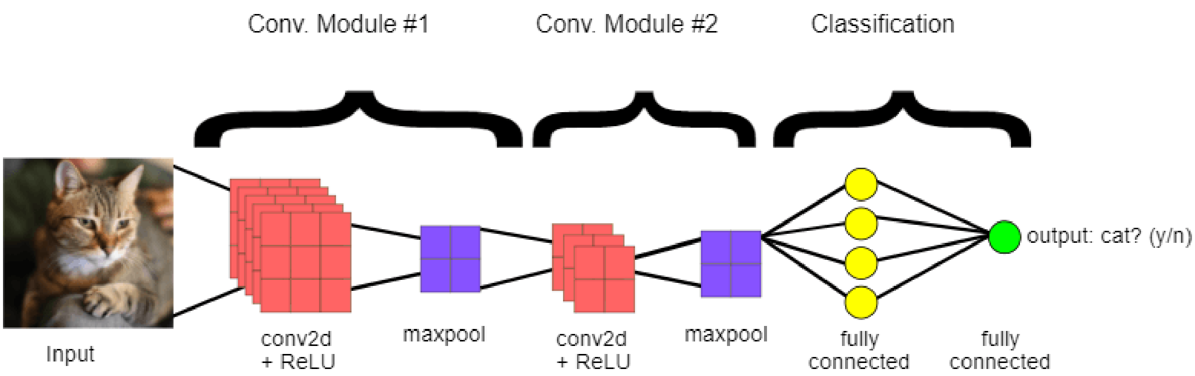

X dataset
N1 feature
N2 feature
N3 feature
N4 feature
Class N
Class O
Class M
Class N
MAJORITY VOTING
FINAL CLASS

Deep Learning Algorithms

Deep learning algorithms are dynamically made to run through several layers of neural networks, which are nothing but a set of decision-making networks that are pre-trained to serve a task. Later, each of these is passed through simple layered representations and move on to the next layer. Followings are the Algorithms of Python Deep Learning-

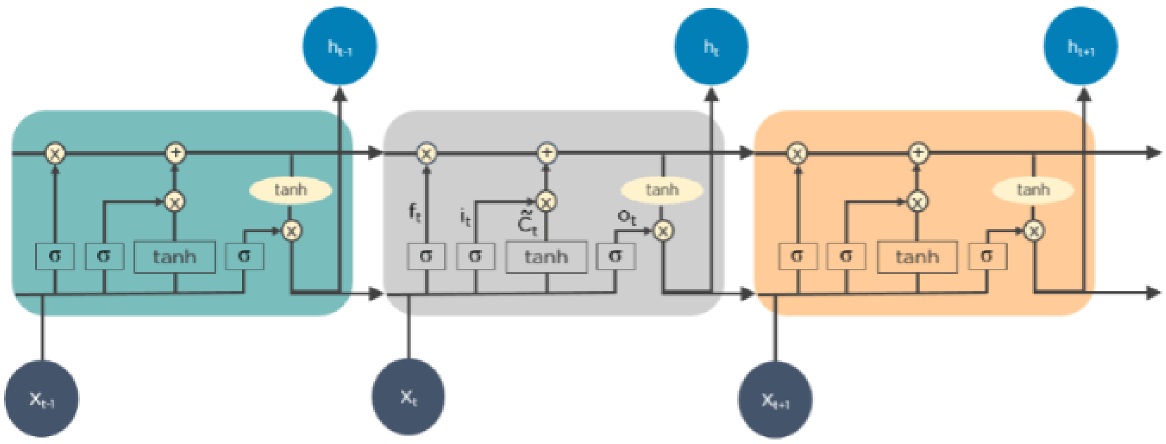
1. Convolutional Neural Networks (CNNs)

CNNs process the data by passing it through multiple layers and extracting features to exhibit convolutional operations. The Convolutional Layer consists of Rectified Linear Unit (ReLU) that outlasts to rectify the feature map. The Pooling layer is used to rectify these feature maps into the next feed. Pooling is generally a sampling algorithm that is down-sampled and it reduces the dimensions of the feature map. Later, the result generated consists of 2-D arrays consisting of single, long, continuous, and linear vector flattened in the map. The next layer i.e., called Fully Connected Layer which forms the flattened matrix or 2-D array fetched from the Pooling Layer as input and identifies the image by classifying it.



2. Long Short Term Memory Networks (LSTMs)

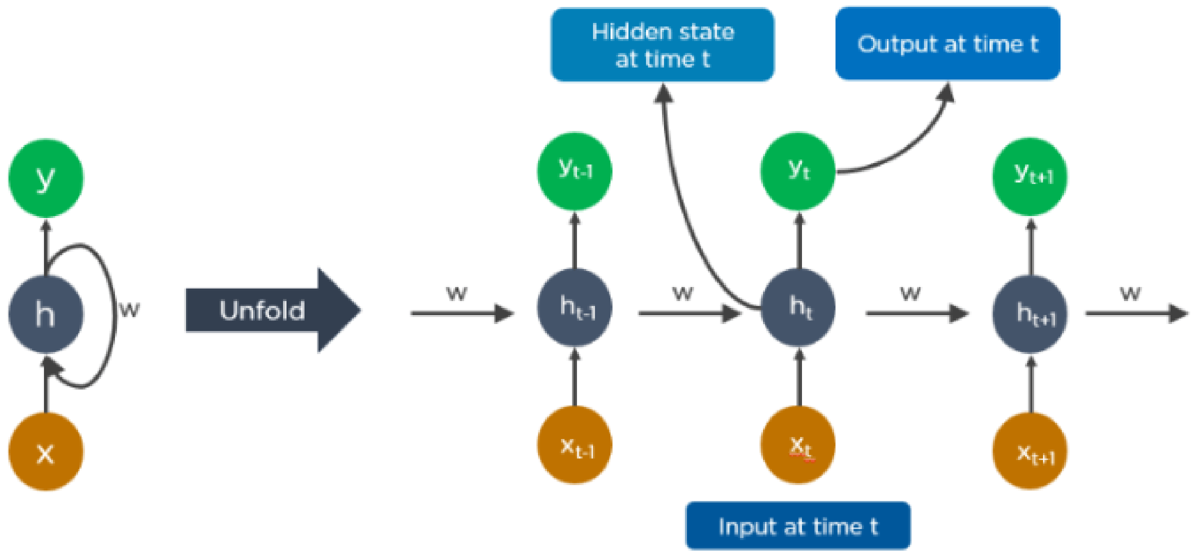
LSTMs can be defined as Recurrent Neural Networks (RNN) that are programmed to learn and adapt for dependencies for the long term. It can memorize and recall past data for a greater period and by default, it is its sole behavior. LSTMs are designed to retain over time and henceforth they are majorly used in time series predictions because they can restrain memory or previous inputs. Besides applications of time series prediction, they can be used to construct speech recognizers, development in pharmaceuticals, and composition of music loops as well. LSTM work in a sequence of events. First, they don't tend to remember irrelevant details attained in the previous state. Next, they update certain cell-state values selectively and finally generate certain parts of the cell-state as output. Below is the diagram of their operation.



3. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks or RNNs consist of some directed connections that form a cycle that allow the input provided from the LSTMs to be used as input in the current phase of RNNs. These inputs are deeply embedded as inputs and enforce the memorization ability of LSTMs lets these inputs get absorbed for a period in the internal memory. RNNs are therefore dependent on the inputs that are preserved by LSTMs and work under the synchronization phenomenon of LSTMs. RNNs are mostly used in captioning the image, time series analysis, recognizing handwritten data, and translating data to machines.

RNNs follow the work approach by putting output feeds (t-1) time if the time is defined as t. Next, the output determined by t is feed at input time t+1. Similarly, these processes are repeated for all the input consisting of any length. There's also a fact about RNNs is that they store historical information and there's no increase in the input size even if the model size is increased. RNNs look something like this when unfolded.



Project Report

on

Stock Prediction

Problem Statement - Predict the stock market price of next few days using previous stock market data (equity or indices) using machine learning or Deep learning.

1. Use News headlines as Data for prediction.

2. Use previous Equity data of Day open, close, low, high for prediction.

3. Any other stock Relative data

Overview:-

We will use some previous year stocks prices data set to Predicate next few day prices. We will use the Long Short-Term Memory (LSTM) method to create a Machine Learning model to forecast stock values. They are used to make minor changes to the information by multiplying and adding. Long-term memory (LSTM) is deep learning artificial RNN architecture. It can handle single data points (such as pictures) as well as full data sequences.

Code: - These steps are followed in code:-

- Importing the Libraries
- Read dataset
- Creating a Training Set and a Test Set
- Data Processing For LSTM
- Building the LSTM Model
- Training the Stock Market Prediction Model
- LSTM Prediction
- Comparing Predicted vs. True Adjusted Close Value
- Predicating next 30 days price

Stock price predication

(using LSTM model)

In [1]:

```
# importing libraries and DATA

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import math
from sklearn.metrics import mean_squared_error

df=pd.read_csv('NSE-TATAGLOBAL11.csv')
df.head()
```

Out[1]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	08-10-2018	208.00	222.25	206.85	216.00	215.15	4642146	10062.83
1	05-10-2018	217.00	218.60	205.90	210.25	209.20	3519515	7407.06
2	04-10-2018	223.50	227.80	216.15	217.25	218.20	1728786	3815.79
3	03-10-2018	230.00	237.50	225.75	226.45	227.60	1708590	3960.27
4	01-10-2018	234.55	234.60	221.05	230.30	230.90	1534749	3486.05

In [2]:

```
df.tail()
```

Out[2]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
1230	14-10-2013	160.85	161.45	157.70	159.3	159.45	1281419	2039.09
1231	11-10-2013	161.15	163.45	159.00	159.8	160.05	1880046	3030.76
1232	10-10-2013	156.00	160.80	155.85	160.3	160.15	3124853	4978.80
1233	09-10-2013	155.70	158.20	154.15	155.3	155.55	2049580	3204.49
1234	08-10-2013	157.00	157.80	155.20	155.8	155.80	1720413	2688.94

In [3]:

```
df=df.iloc[::-1]
```

In [4]:

```
df.head()
```

Out[4]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
1234	08-10-2013	157.00	157.80	155.20	155.8	155.80	1720413	2688.94
1233	09-10-2013	155.70	158.20	154.15	155.3	155.55	2049580	3204.49
1232	10-10-2013	156.00	160.80	155.85	160.3	160.15	3124853	4978.80
1231	11-10-2013	161.15	163.45	159.00	159.8	160.05	1880046	3030.76
1230	14-10-2013	160.85	161.45	157.70	159.3	159.45	1281419	2039.09

In [5]:

```
df.tail()
```

Out[5]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
4	01-10-2018	234.55	234.60	221.05	230.30	230.90	1534749	3486.05
3	03-10-2018	230.00	237.50	225.75	226.45	227.60	1708590	3960.27
2	04-10-2018	223.50	227.80	216.15	217.25	218.20	1728786	3815.79
1	05-10-2018	217.00	218.60	205.90	210.25	209.20	3519515	7407.06
0	08-10-2018	208.00	222.25	206.85	216.00	215.15	4642146	10062.83

In [6]:

```
df.reset_index(drop=True, inplace=True)
df
```

Out[6]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	08-10-2013	157.00	157.80	155.20	155.80	155.80	1720413	2688.94
1	09-10-2013	155.70	158.20	154.15	155.30	155.55	2049580	3204.49
2	10-10-2013	156.00	160.80	155.85	160.30	160.15	3124853	4978.80
3	11-10-2013	161.15	163.45	159.00	159.80	160.05	1880046	3030.76
4	14-10-2013	160.85	161.45	157.70	159.30	159.45	1281419	2039.09
...	...	...	...	...	...	...	...	...
1230	01-10-2018	234.55	234.60	221.05	230.30	230.90	1534749	3486.05
1231	03-10-2018	230.00	237.50	225.75	226.45	227.60	1708590	3960.27
1232	04-10-2018	223.50	227.80	216.15	217.25	218.20	1728786	3815.79
1233	05-10-2018	217.00	218.60	205.90	210.25	209.20	3519515	7407.06
1234	08-10-2018	208.00	222.25	206.85	216.00	215.15	4642146	10062.83

1235 rows × 8 columns

predication for stocks closing price

In [7]:

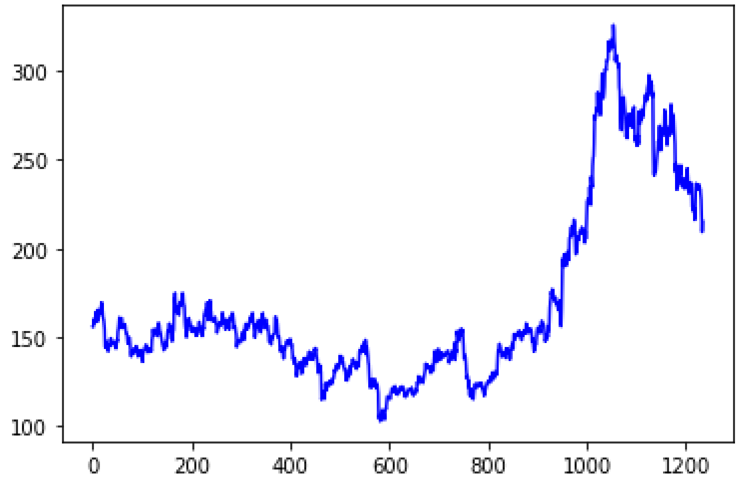
```
df=df['Close']
```

In [8]:

```
plt.plot(df,'b')
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x2159ce0ceb0>]
```



Scaling prices

In [9]:

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df=scaler.fit_transform(np.array(df).reshape(-1,1))
df
```

Out[9]:

```
array([[0.23823398],
       [0.2371134 ],
       [0.25773196],
       ...,
       [0.51792918],
       [0.47758853],
       [0.50425818]])
```

Splitting dataset into train and test split

In [10]:

```
train_size=int(len(df)*0.70)
test_size=len(df)-train_size
train_data,test_data=df[0:train_size,:],df[train_size:len(df),:1]
train_size,test_size
```

Out[10]:

```
(864, 371)
```

Convert an array of values into a dataset matrix

In [11]:

```
def create_dataset(dataset, time_step):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-- 49,50
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

In [12]:

```
# reshaping into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 100
x_train, y_train = create_dataset(train_data, time_step)
x_test, y_test = create_dataset(test_data, time_step)
```

In [13]:

```
x_train.shape,y_train.shape
```

Out[13]: ((763, 100), (763,))

In [14]:
```
x_test.shape,y_test.shape
```

Out[14]: ((270, 100), (270,))

In [15]:
```
# reshape input to be [samples, time steps, features] for LSTM
x_train =x_train.reshape(x_train.shape[0],x_train.shape[1] , 1)
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1] , 1)
```

In [16]:
```
x_train.shape
```

Out[16]: (763, 100, 1)

Using LSTM model

In [17]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,LSTM,Dropout

model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100,batch_size=64,v
```

```
Epoch 1/100
12/12 [==============================] - 7s 258ms/step - loss: 0.0100 - val_loss: 0.
1307
Epoch 2/100
12/12 [==============================] - 2s 170ms/step - loss: 0.0025 - val_loss: 0.
0468
Epoch 3/100
12/12 [==============================] - 2s 193ms/step - loss: 0.0015 - val_loss: 0.
0341
Epoch 4/100
```

```
12/12 [==============================] - 2s 173ms/step - loss: 0.0011 - val_loss: 0.
0087
Epoch 5/100
12/12 [==============================] - 2s 176ms/step - loss: 9.5998e-04 - val_los
s: 0.0054
Epoch 6/100
12/12 [==============================] - 2s 192ms/step - loss: 9.2629e-04 - val_los
s: 0.0063
Epoch 7/100
12/12 [==============================] - 2s 170ms/step - loss: 9.0178e-04 - val_los
s: 0.0079
Epoch 8/100
12/12 [==============================] - 2s 170ms/step - loss: 8.8916e-04 - val_los
s: 0.0064
Epoch 9/100
12/12 [==============================] - 2s 166ms/step - loss: 8.4978e-04 - val_los
s: 0.0050
Epoch 10/100
12/12 [==============================] - 2s 183ms/step - loss: 8.1960e-04 - val_los
s: 0.0046
Epoch 11/100
12/12 [==============================] - 2s 163ms/step - loss: 8.1709e-04 - val_los
s: 0.0041
Epoch 12/100
12/12 [==============================] - 2s 160ms/step - loss: 7.8214e-04 - val_los
s: 0.0039
Epoch 13/100
12/12 [==============================] - 2s 162ms/step - loss: 7.8733e-04 - val_los
s: 0.0052
Epoch 14/100
12/12 [==============================] - 2s 173ms/step - loss: 7.3681e-04 - val_los
s: 0.0039
Epoch 15/100
12/12 [==============================] - 2s 154ms/step - loss: 7.1752e-04 - val_los
s: 0.0039
Epoch 16/100
12/12 [==============================] - 2s 174ms/step - loss: 7.2673e-04 - val_los
s: 0.0034
Epoch 17/100
12/12 [==============================] - 2s 172ms/step - loss: 7.1868e-04 - val_los
s: 0.0041
Epoch 18/100
12/12 [==============================] - 2s 161ms/step - loss: 6.5039e-04 - val_los
s: 0.0031
Epoch 19/100
12/12 [==============================] - 2s 163ms/step - loss: 6.3992e-04 - val_los
s: 0.0031
Epoch 20/100
12/12 [==============================] - 2s 174ms/step - loss: 6.4221e-04 - val_los
s: 0.0043
Epoch 21/100
12/12 [==============================] - 2s 165ms/step - loss: 6.3678e-04 - val_los
s: 0.0028
Epoch 22/100
12/12 [==============================] - 2s 170ms/step - loss: 5.9934e-04 - val_los
s: 0.0032
Epoch 23/100
12/12 [==============================] - 2s 190ms/step - loss: 6.0142e-04 - val_los
s: 0.0026
Epoch 24/100
12/12 [==============================] - 2s 191ms/step - loss: 5.9086e-04 - val_los
s: 0.0039
Epoch 25/100
12/12 [==============================] - 2s 191ms/step - loss: 6.1280e-04 - val_los
s: 0.0029
Epoch 26/100
12/12 [==============================] - 2s 174ms/step - loss: 5.5353e-04 - val_los
s: 0.0027
Epoch 27/100
```

```
12/12 [==============================] - 2s 193ms/step - loss: 5.3018e-04 - val_los
s: 0.0027
Epoch 28/100
12/12 [==============================] - 2s 176ms/step - loss: 5.3889e-04 - val_los
s: 0.0037
Epoch 29/100
12/12 [==============================] - 2s 185ms/step - loss: 5.1437e-04 - val_los
s: 0.0023
Epoch 30/100
12/12 [==============================] - 2s 176ms/step - loss: 5.0729e-04 - val_los
s: 0.0028
Epoch 31/100
12/12 [==============================] - 2s 167ms/step - loss: 4.8881e-04 - val_los
s: 0.0021
Epoch 32/100
12/12 [==============================] - 2s 187ms/step - loss: 5.3692e-04 - val_los
s: 0.0046
Epoch 33/100
12/12 [==============================] - 2s 168ms/step - loss: 5.4835e-04 - val_los
s: 0.0021
Epoch 34/100
12/12 [==============================] - 2s 179ms/step - loss: 5.4707e-04 - val_los
s: 0.0037
Epoch 35/100
12/12 [==============================] - 2s 171ms/step - loss: 5.0232e-04 - val_los
s: 0.0019
Epoch 36/100
12/12 [==============================] - 2s 169ms/step - loss: 4.7762e-04 - val_los
s: 0.0039
Epoch 37/100
12/12 [==============================] - 2s 171ms/step - loss: 4.7378e-04 - val_los
s: 0.0029
Epoch 38/100
12/12 [==============================] - 2s 169ms/step - loss: 4.4655e-04 - val_los
s: 0.0019
Epoch 39/100
12/12 [==============================] - 2s 169ms/step - loss: 4.8253e-04 - val_los
s: 0.0048
Epoch 40/100
12/12 [==============================] - 2s 174ms/step - loss: 4.5138e-04 - val_los
s: 0.0036
Epoch 41/100
12/12 [==============================] - 2s 178ms/step - loss: 4.2361e-04 - val_los
s: 0.0029
Epoch 42/100
12/12 [==============================] - 2s 170ms/step - loss: 4.1520e-04 - val_los
s: 0.0033
Epoch 43/100
12/12 [==============================] - 2s 172ms/step - loss: 4.0880e-04 - val_los
s: 0.0025
Epoch 44/100
12/12 [==============================] - 2s 173ms/step - loss: 4.0364e-04 - val_los
s: 0.0021
Epoch 45/100
12/12 [==============================] - 2s 182ms/step - loss: 3.9081e-04 - val_los
s: 0.0054
Epoch 46/100
12/12 [==============================] - 2s 200ms/step - loss: 3.9391e-04 - val_los
s: 0.0029
Epoch 47/100
12/12 [==============================] - 2s 184ms/step - loss: 3.7565e-04 - val_los
s: 0.0020
Epoch 48/100
12/12 [==============================] - 2s 194ms/step - loss: 4.0234e-04 - val_los
s: 0.0039
Epoch 49/100
12/12 [==============================] - 2s 181ms/step - loss: 3.9950e-04 - val_los
s: 0.0026
Epoch 50/100
```

```
12/12 [==============================] - 2s 176ms/step - loss: 4.0146e-04 - val_los
s: 0.0030
Epoch 51/100
12/12 [==============================] - 2s 202ms/step - loss: 3.6901e-04 - val_los
s: 0.0045
Epoch 52/100
12/12 [==============================] - 2s 186ms/step - loss: 3.6032e-04 - val_los
s: 0.0038
Epoch 53/100
12/12 [==============================] - 2s 191ms/step - loss: 3.4338e-04 - val_los
s: 0.0026
Epoch 54/100
12/12 [==============================] - 2s 175ms/step - loss: 3.4289e-04 - val_los
s: 0.0072
Epoch 55/100
12/12 [==============================] - 2s 192ms/step - loss: 3.6659e-04 - val_los
s: 0.0029
Epoch 56/100
12/12 [==============================] - 2s 185ms/step - loss: 3.3102e-04 - val_los
s: 0.0034
Epoch 57/100
12/12 [==============================] - 2s 182ms/step - loss: 3.3199e-04 - val_los
s: 0.0032
Epoch 58/100
12/12 [==============================] - 2s 185ms/step - loss: 3.2959e-04 - val_los
s: 0.0036
Epoch 59/100
12/12 [==============================] - 2s 169ms/step - loss: 3.1508e-04 - val_los
s: 0.0030
Epoch 60/100
12/12 [==============================] - 2s 180ms/step - loss: 3.2516e-04 - val_los
s: 0.0046
Epoch 61/100
12/12 [==============================] - 2s 167ms/step - loss: 2.9853e-04 - val_los
s: 0.0036
Epoch 62/100
12/12 [==============================] - 2s 181ms/step - loss: 3.2790e-04 - val_los
s: 0.0021
Epoch 63/100
12/12 [==============================] - 2s 177ms/step - loss: 2.9796e-04 - val_los
s: 0.0025
Epoch 64/100
12/12 [==============================] - 2s 179ms/step - loss: 2.9186e-04 - val_los
s: 0.0033
Epoch 65/100
12/12 [==============================] - 2s 185ms/step - loss: 2.8922e-04 - val_los
s: 0.0040
Epoch 66/100
12/12 [==============================] - 2s 166ms/step - loss: 2.9055e-04 - val_los
s: 0.0035
Epoch 67/100
12/12 [==============================] - 2s 180ms/step - loss: 2.7458e-04 - val_los
s: 0.0022
Epoch 68/100
12/12 [==============================] - 2s 168ms/step - loss: 2.8811e-04 - val_los
s: 0.0035
Epoch 69/100
12/12 [==============================] - 2s 177ms/step - loss: 2.6262e-04 - val_los
s: 0.0039
Epoch 70/100
12/12 [==============================] - 2s 181ms/step - loss: 2.6507e-04 - val_los
s: 0.0019
Epoch 71/100
12/12 [==============================] - 2s 169ms/step - loss: 2.8545e-04 - val_los
s: 0.0042
Epoch 72/100
12/12 [==============================] - 2s 189ms/step - loss: 2.5216e-04 - val_los
s: 0.0018
Epoch 73/100
```

```
12/12 [==============================] - 2s 174ms/step - loss: 2.4758e-04 - val_los
s: 0.0030
Epoch 74/100
12/12 [==============================] - 2s 191ms/step - loss: 2.4464e-04 - val_los
s: 0.0034
Epoch 75/100
12/12 [==============================] - 2s 173ms/step - loss: 2.3547e-04 - val_los
s: 0.0030
Epoch 76/100
12/12 [==============================] - 2s 174ms/step - loss: 2.4247e-04 - val_los
s: 0.0024
Epoch 77/100
12/12 [==============================] - 2s 180ms/step - loss: 2.2801e-04 - val_los
s: 0.0021
Epoch 78/100
12/12 [==============================] - 2s 170ms/step - loss: 2.4416e-04 - val_los
s: 0.0020
Epoch 79/100
12/12 [==============================] - 2s 180ms/step - loss: 2.5509e-04 - val_los
s: 0.0060
Epoch 80/100
12/12 [==============================] - 2s 171ms/step - loss: 2.8631e-04 - val_los
s: 0.0018
Epoch 81/100
12/12 [==============================] - 2s 180ms/step - loss: 2.6004e-04 - val_los
s: 0.0039
Epoch 82/100
12/12 [==============================] - 2s 175ms/step - loss: 2.3499e-04 - val_los
s: 0.0040
Epoch 83/100
12/12 [==============================] - 2s 168ms/step - loss: 2.1618e-04 - val_los
s: 0.0024
Epoch 84/100
12/12 [==============================] - 2s 180ms/step - loss: 2.1373e-04 - val_los
s: 0.0036
Epoch 85/100
12/12 [==============================] - 2s 172ms/step - loss: 2.0939e-04 - val_los
s: 0.0020
Epoch 86/100
12/12 [==============================] - 2s 175ms/step - loss: 2.2483e-04 - val_los
s: 0.0042
Epoch 87/100
12/12 [==============================] - 2s 173ms/step - loss: 2.0648e-04 - val_los
s: 0.0032
Epoch 88/100
12/12 [==============================] - 2s 172ms/step - loss: 2.0431e-04 - val_los
s: 0.0024
Epoch 89/100
12/12 [==============================] - 2s 172ms/step - loss: 1.9300e-04 - val_los
s: 0.0024
Epoch 90/100
12/12 [==============================] - 2s 171ms/step - loss: 1.8833e-04 - val_los
s: 0.0023
Epoch 91/100
12/12 [==============================] - 2s 177ms/step - loss: 1.8687e-04 - val_los
s: 0.0024
Epoch 92/100
12/12 [==============================] - 2s 174ms/step - loss: 1.9094e-04 - val_los
s: 0.0034
Epoch 93/100
12/12 [==============================] - 2s 174ms/step - loss: 1.8662e-04 - val_los
s: 0.0017
Epoch 94/100
12/12 [==============================] - 2s 180ms/step - loss: 1.8026e-04 - val_los
s: 0.0016
Epoch 95/100
12/12 [==============================] - 2s 172ms/step - loss: 1.8938e-04 - val_los
s: 0.0013
Epoch 96/100
```

```
12/12 [==============================] - 2s 179ms/step - loss: 2.0090e-04 - val_los
s: 0.0014
Epoch 97/100
12/12 [==============================] - 2s 174ms/step - loss: 1.8186e-04 - val_los
s: 0.0017
Epoch 98/100
12/12 [==============================] - 2s 187ms/step - loss: 1.7212e-04 - val_los
s: 0.0015
Epoch 99/100
12/12 [==============================] - 2s 183ms/step - loss: 1.8260e-04 - val_los
s: 0.0020
Epoch 100/100
12/12 [==============================] - 2s 189ms/step - loss: 1.6609e-04 - val_los
s: 0.0014
```

Out[19]:
```
<keras.callbacks.History at 0x215a01daf40>
```

Prediction and checking performance metrics

In [20]:
```
train_predict=model.predict(x_train)
test_predict=model.predict(x_test)
```

In [21]:
```
#Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)

#Calculate RMSE performance metrics
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[21]:
```
141.02224963138733
```

In [22]:
```
#Test Data RMSE
math.sqrt(mean_squared_error(y_test,test_predict))
```

Out[22]:
```
254.75590886225544
```

In [23]:
```
y_predicated=model.predict(x_test)
y_predicated.shape
```

Out[23]:
```
(270, 1)
```

In [24]:
```
y_test.shape
```

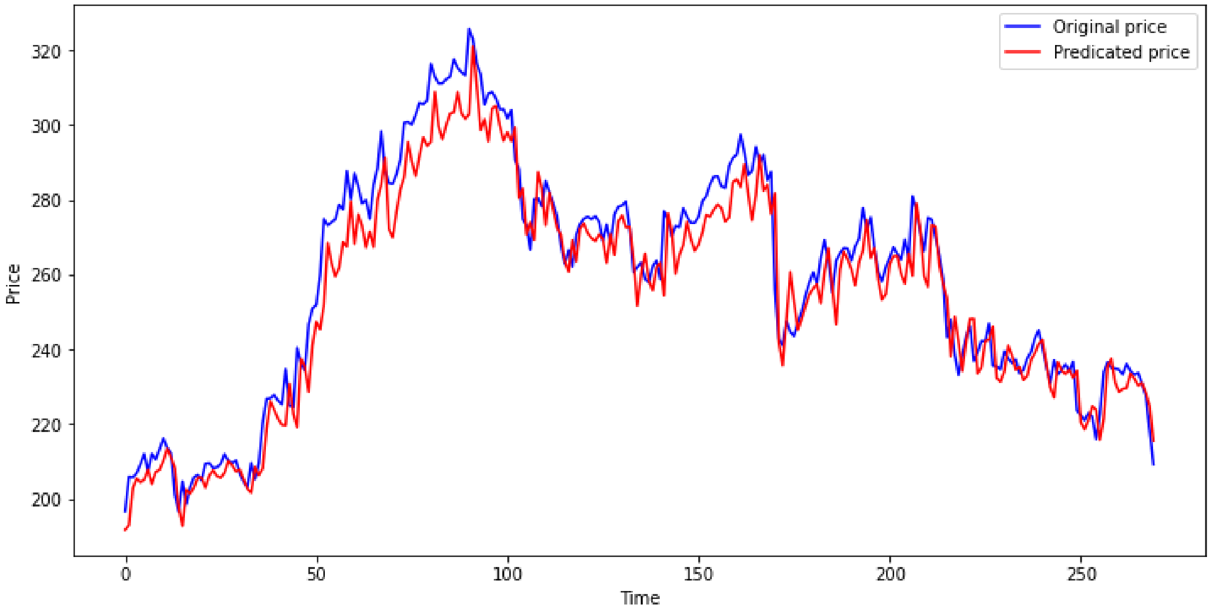
Out[24]:
```
(270,)
```

Graph b/w predicate and real price

In [25]:
```
y_predicated=scaler.inverse_transform(y_predicated)
y_test_1=scaler.inverse_transform(np.array(y_test).reshape(-1,1))
```

In [26]:
```
plt.figure(figsize=(12,6))
plt.plot(y_test_1,'b',label='Original price')
plt.plot(y_predicated,'r',label='Predicated price')
plt.xlabel('Time')
plt.ylabel('Price')
```

```
plt.legend()
plt.show()
```



Predicating future 30 Days prices

In [27]:

```
# for predication i have to take previous 100 day data
len(test_data)
```

Out[27]: 371

In [28]:

```
# 371-100=271
x_input=test_data[271:].reshape(1,-1)
x_input.shape
```

Out[28]: (1, 100)

In [35]:

```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [30]:

```
from numpy import array

lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(temp_input)>100):
        x_input=np.array(temp_input[1:])
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
```

```python
        temp_input.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1


lst_output
```

Out[30]:

```
[[0.5054447054862976],
 [0.5242617130279541],
 [0.5366925001144409],
 [0.5406090617179871],
 [0.5348264575004578],
 [0.5219063758850098],
 [0.5066055059432983],
 [0.49364638328552246],
 [0.4859609305858612],
 [0.4840588867664337],
 [0.4864457845687866],
 [0.49058300256729126],
 [0.4939187169075012],
 [0.4946891665458679],
 [0.49232831597328186],
 [0.4874247908592224],
 [0.4812997877597809],
 [0.47540947794914246],
 [0.4708256721496582],
 [0.4679659903049469],
 [0.4666072130203247],
 [0.46610400080680847],
 [0.4656883180141449],
 [0.46473637223243713],
 [0.4629335105419159],
 [0.46030569076538086],
 [0.4571375548839569],
 [0.4538259208202362],
 [0.45072850584983826],
 [0.44806137681007385]]
```

In [31]:

```python
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
```

In [32]:

```python
len(df)
```

Out[32]:

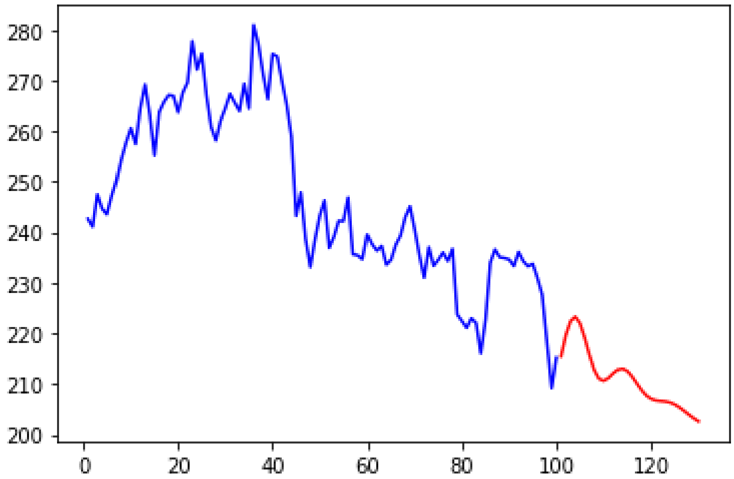
```
1235
```

In [33]:

```python
plt.plot(day_new,scaler.inverse_transform(df[1135:]),'b')
plt.plot(day_pred,scaler.inverse_transform(lst_output),'r')
```

Out[33]:

```
[<matplotlib.lines.Line2D at 0x215a9031730>]
```



Next 30 days prices

In [34]:

```
scaler.inverse_transform(lst_output)
```

Out[34]:

```
array([[215.41471379],
       [219.61278818],
       [222.38609678],
       [223.25988167],
       [221.96978267],
       [219.08731246],
       [215.67368838],
       [212.78250811],
       [211.06788361],
       [210.64353764],
       [211.17605454],
       [212.09906787],
       [212.84326574],
       [213.01515306],
       [212.48844729],
       [211.39447084],
       [210.02798265],
       [208.71385453],
       [207.69120746],
       [207.05321244],
       [206.75006922],
       [206.63780258],
       [206.54506375],
       [206.33268465],
       [205.9304662 ],
       [205.34419961],
       [204.63738849],
       [203.89856293],
       [203.20752966],
       [202.61249317]])
```

Conclusion

This training has introduced us to Machine Learning. Now, we know that Machine Learning is a technique of training machines to perform the activities a human brain can do, albeit bit faster and better than an average human-being. Today we have seen that the machines can beat human champions in games such as Chess, Mahjong, which are considered very complex. We have seen that machines can be trained to perform human activities in several areas and can aid humans in living better lives. To describe machine learning in general terms, a variety models are used to learn patterns in data and make accurate predictions based on the patterns it observes.

Machine Learning can be a Supervised or Unsupervised. If we have a lesser amount of data and clearly labelled data for training, we opt for Supervised Learning. Unsupervised Learning would generally give better performance and results for large data sets. If we have a huge data set easily available, we go for deep learning techniques. We also have learned Reinforcement Learning and Deep Reinforcement Learning. We now know what Neural Networks are, their applications and limitations. Specifically, we have developed a thought process for approaching problems that machine learning works so well at solving. We have learnt how machine learning is different than descriptive statistics.

Predicting the stock market was a time-consuming and laborious procedure a few years or even a decade ago. However, with the application of machine learning for stock market forecasts, the procedure has become much simpler. Machine learning not only saves time and resources but also outperforms people in terms of performance. It will always prefer to use a trained computer algorithm since it will advise you based only on facts, numbers, and data and will not factor in emotions or prejudice.

References

- Machine Learning Tutorial (tutorialspoint.com)
- ARTIFICIAL INTELLIGENCE (AI) – Dr Rajiv Desai (drrajivdesaimd.com)
- What is Machine Learning? - GeeksforGeeks
- Machine Learning Tutorial | Machine Learning with Python - Javatpoint
- (PDF) Stock Market Prediction Using Machine Learning Techniques (researchgate.net)