CHAPTER 1

INTRODUCTION

1.1 Introduction

Stock price forecasting is a popular and important topic in the field of Finance because of its volatility. It attracts researchers to capture the unpredictability and predict its next moves. Stock brokers and market analysts continuously study the pattern of stocks to plan their buy or sell strategies. Internal to Apple, it is important for them to know what triggers the stock price and what downfalls it. This influence various business decisions taken in Apple. Stock market produces huge amount of data every day, so it's not an easy task to predict the future stock price considering all the current and past information. There are two main approaches to predict the stock prices, one being Technical analysis and other being Fundamental analysis. Technical analysis used the past prices and volume to predict the future trend whereas Fundamental analysis uses other financial and economic factors which influence the stock prices. In this paper, we have collected both technical and fundamental data from online sources for Apple to forecast Apple stock price. The popularity of Apple and availability of heaps of information online for our research, motivated us to choose Apple stock price forecasting.

1.2 Existing System

Mining time series data and the information from textual documents is currently an emerging topic in the data mining society. Increasingly many researchers are focusing their studies on this topic. The relationships between different stocks and influences they can have on one another are often ignored. However, this is valuable information. For example, if the price of Honda's stocks goes down, that may trigger a movement of stock prices of Toyota.

1.3 Disadvantages

Stock price prediction is based on the implications of Efficient Market Hypothesis (EMH) that stock price reacts instantaneously to day-to-day news. Work has been done to predict the direction of stock price index movement using soft computing-based models such as Other approaches like Fuzzy Logic, Genetic Algorithm, Machine Learning algorithms were also implemented. Hidden Markov Models (HMM) approach was used to forecast stock price for interrelated markets. This approach helped in pattern recognition and classification problems because of its ability for dynamic system modeling.

1.4 Proposed System

We are single handedly focusing on just predicting Apple's stock price using a very fast and amazing framework called scikit-learn which is far better than the previous frameworks and this approach saves us lots of computations and python provides us compatibility and easiness of extension

1.5 Advantages

Our project mainly aims to improve the predictive accuracy of Apple stock prices and determine if Time series forecasting and svm models can be used to offer insights regarding the future stock prices and how other external events and factors are impacting the stock prices. On a broad perspective, we target to build a model to predict the trading indication(rise range and fall range) of the stock price of Apple that would help the company for better planning. Thus, our problem can be Regression. Our team decided to implement the same using Machine learning, Data sources used and the work corresponding to each of the data sources.

1.6 Problem Definition

Stock value gauging is a mainstream and significant point in the field of Finance in light of its instability. It pulls in scientists to catch the unconventionality and foresee its best courses of action. Stock dealers and market examiners ceaselessly study the example of stocks to design their purchase or sell techniques. Inner to Apple, it is significant for them to understand what triggers the stock cost and what destructions it. This impact different business choices taken in Apple. Securities exchange produces enormous measure of information consistently, so it is anything but a simple assignment to foresee the future stock cost thinking about all the current and past data.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1 System Requirement:

PROCESSOR : I3 OR ABOVE

STORAGE : 320 GB OR MORE

RAM : 4GB OR MORE

OS : WINDOWS 10 OR ABOVE

2.2 Software Requirement:

INTERPRETER : PYTHON 3

OS : WINDOWS 10 OR ABOVE OS

2.3 Functional Requirement:

Our undertaking chiefly expects to improve the prescient precision of Apple stock costs and decide whether Time arrangement estimating and svm models can be utilized to offer bits of knowledge with respect to the future stock costs and how other outer occasions and factors are affecting the stock costs.

2.4 Technologies Used:

PYTHON PROGRAMMING

MACHINE LEARNING

2.5 Programming Language Used:

PYTHON PROGRAMMING

IDE USED:

IDLE, VS CODE OR JUPYTER NOTEBOOK

CHAPTER 3

DESIGN ANALYSIS

3.1 Data Flow Diagram:

In Data flow diagram we are using the Linear Regression Algorithm for the price prediction from the Training Data Model.

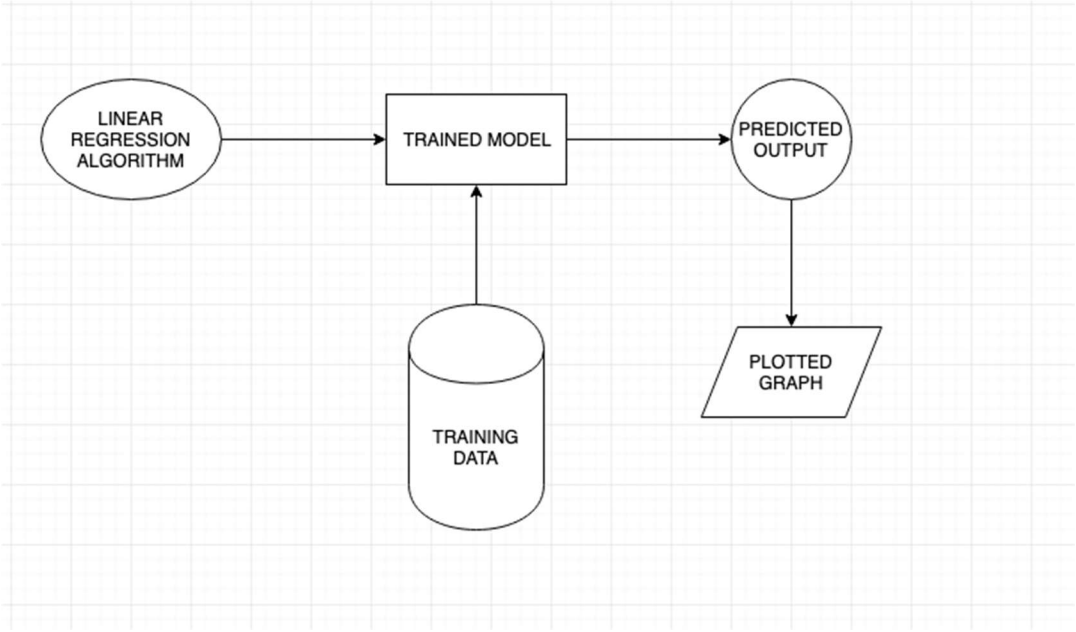

Figure No. 3.1 Data Flow Diagram

The above figure describes the data flow of the training model for the price prediction.

3.2 System Architecture

The Architecture represent the Input Algorithm for the trained model to get the desired output.

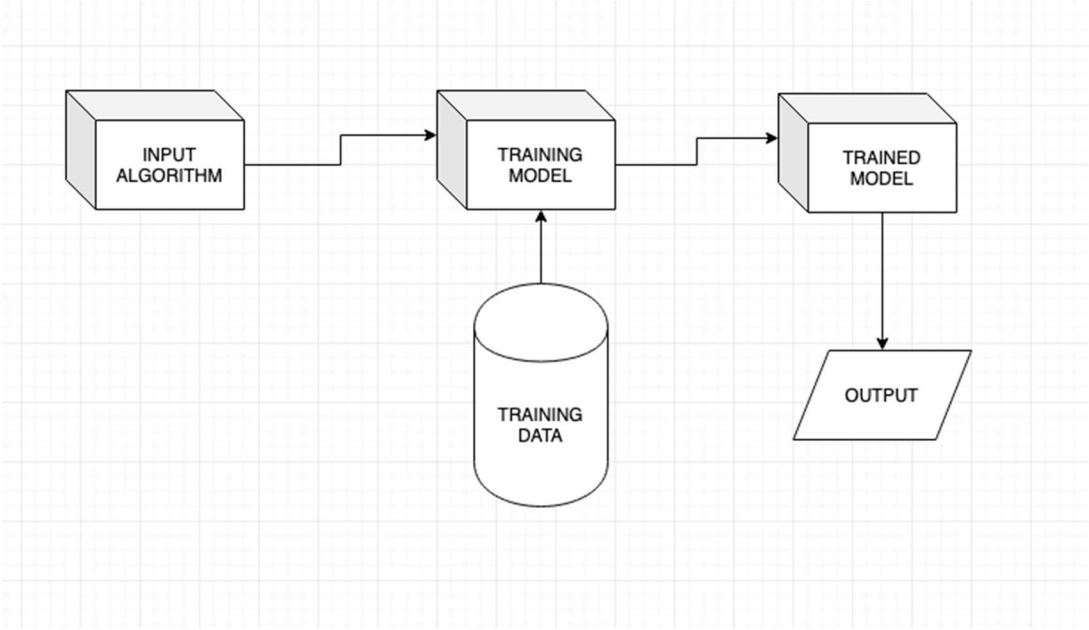

Figure No. 3.2 System Architecture

The above figure describes the System Architecture the training model for the price prediction.

3.3 Class Diagram

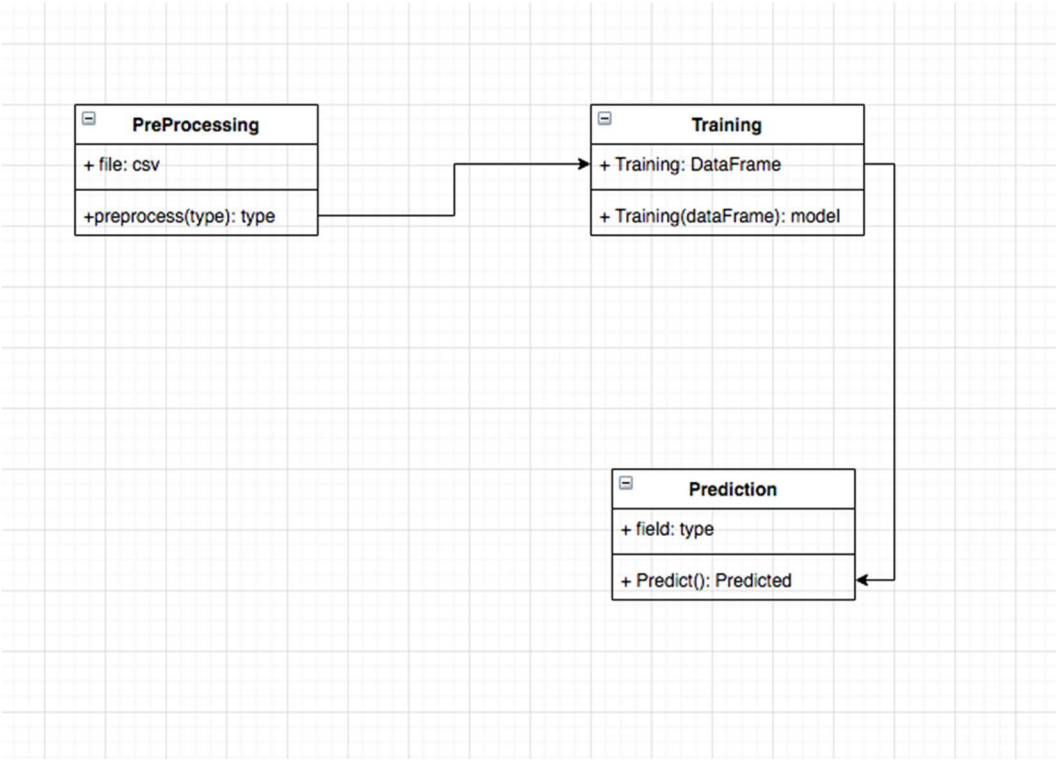

Figure No. 3.3 Class Diagram

The above figure represents the System (UML) Class Diagram

3.4 Component Diagram

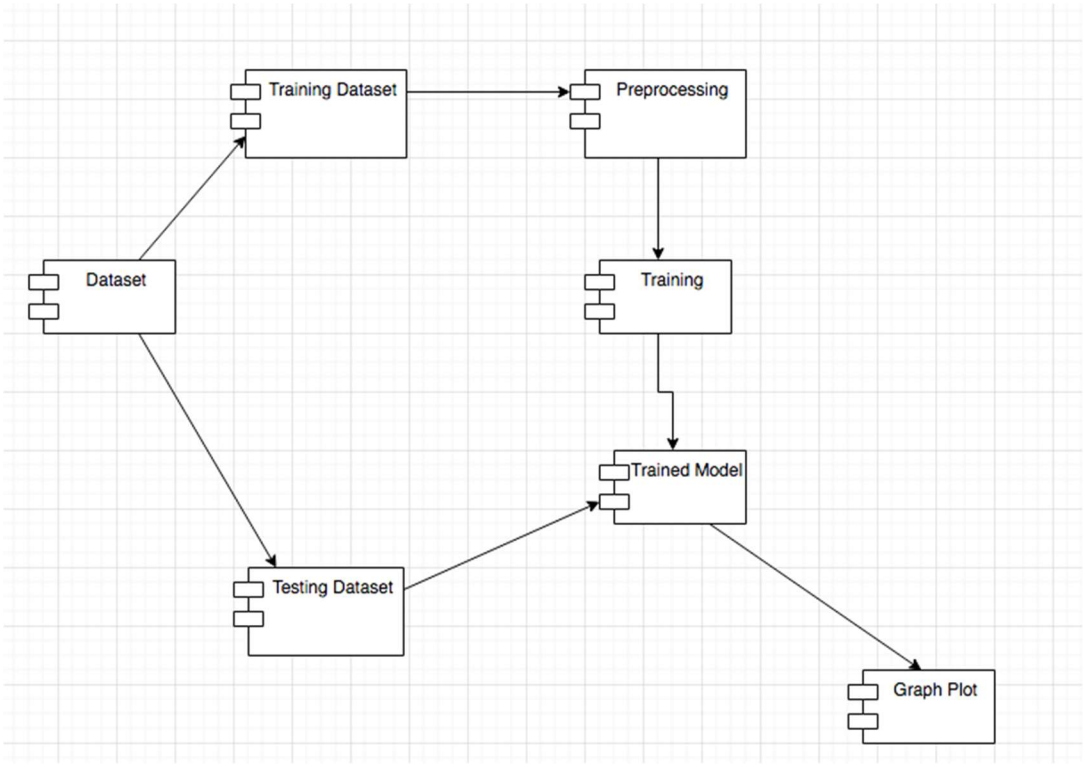

Figure No. 3.4 Component Diagram

The above figure represents the System (UML) Component Diagram

CHAPTER 4

ALGORITHM & MODULES

4.1 Algorithm

4.1.1 Linear regression

linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

4.1.2 Regression vs. classification

A regression model predicts continuous values. For example, regression models make predictions that answer questions like the following:

- What will be the weather tomorrow?
- What is the probability that a user will click on this ad?

A classification model predicts discrete values. For example, classification models make predictions that answer questions like the following:

- Is a given email message spam or not spam?
- Is this an image of a dog, a cat, or a hamster?

It has long been known that crickets (an insect species) chirp more frequently on hotter days than on cooler days. For decades, professional and amateur scientists have cataloged data on chirps-per-minute and temperature. If you want to learn a model to predict this relationship. Using this data, you want to explore this relationship.

First, examine the data by plotting it:

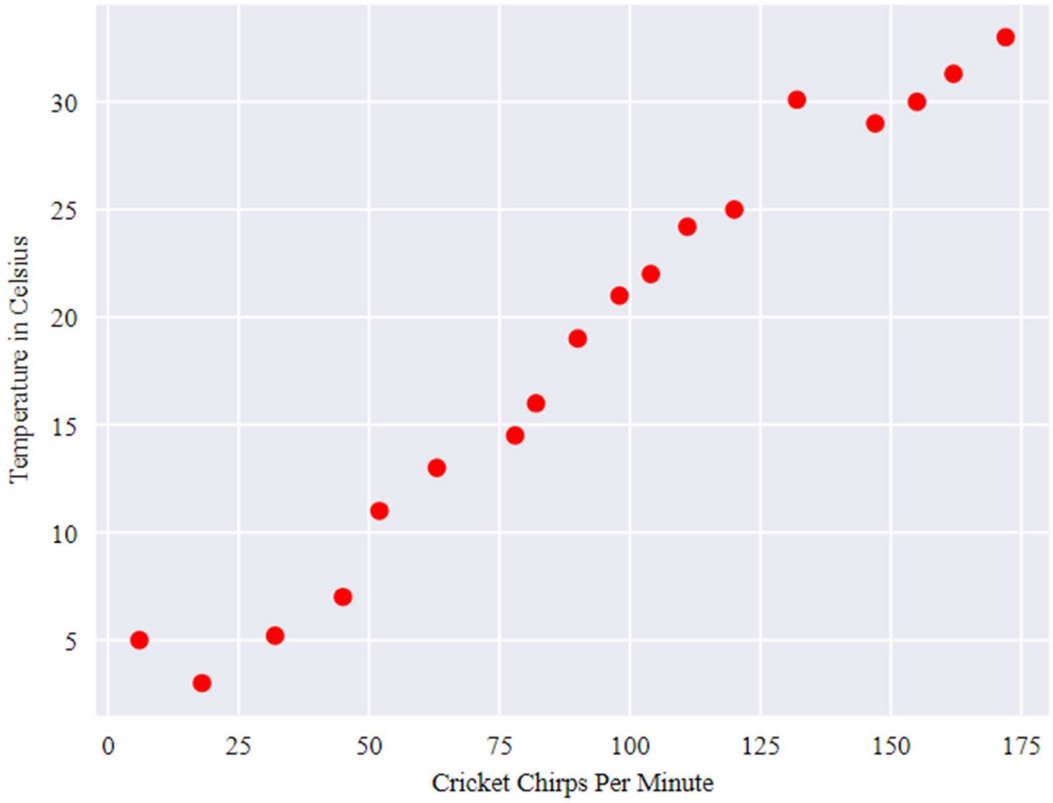

Figure No 4.1 Cricket Chirps per Minute

As expected, the plot shows the temperature rising with the number of chirps. Is this relationship between chirps and temperature linear? Yes, you could draw a single straight line like the following to approximate this relationship:

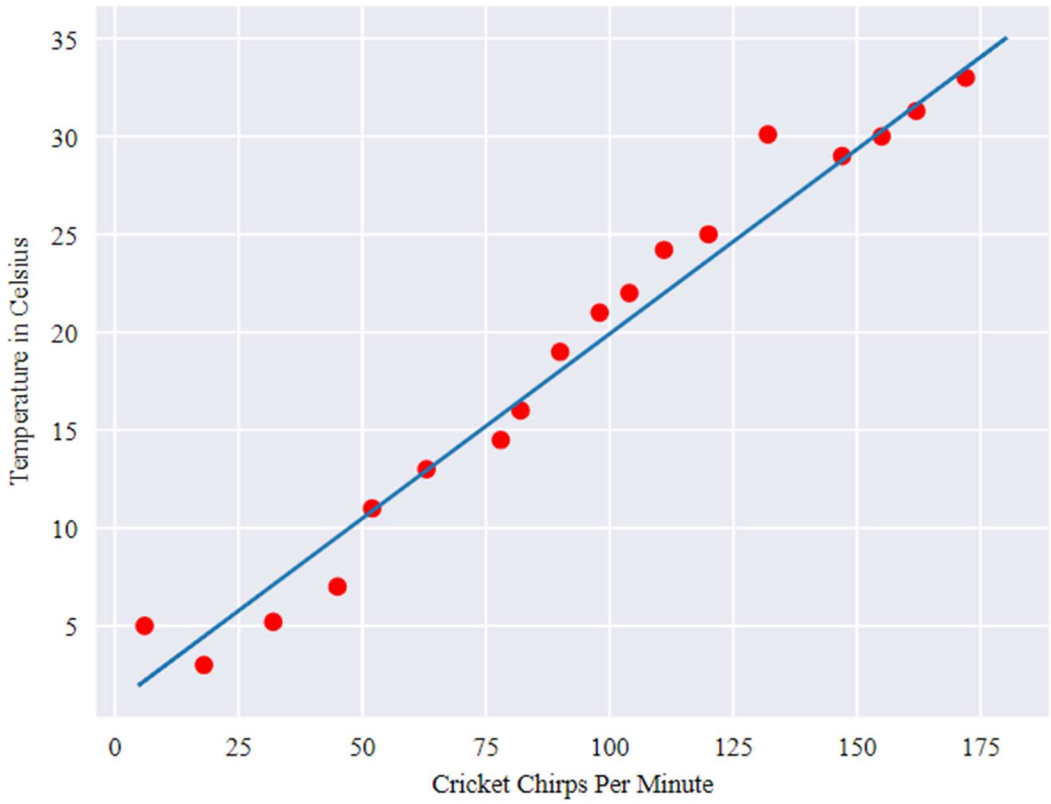

Figure No 4.2 Cricket Chirps Per Minute Temperature in Celsius.

True, the line doesn't pass through every dot, but the line does clearly show the relationship between chirps and temperature. Using the equation for a line, you could write down this relationship as follows:

y=mx+b

where:

- y is the temperature in Celsius—the value we're trying to predict.
- m is the slope of the line.
- x is the number of chirps per minute—the value of our input feature.
- b is the y-intercept.

By convention in machine learning, you'll write the equation for a model slightly differently:

$y'=b+w_1x_1$

where:

- y' is the predicted label (a desired output).
- b is the bias (the y-intercept), sometimes referred to as w_0.
- w_1 is the weight of feature 1. Weight is the same concept as the " slope" m in the traditional equation of a line.
- x_1 is a feature (a known input).

To **infer** (predict) the temperature y' for a new chirps-per-minute value x_1, just substitute the x_1 value into this model.

Although this model uses only one feature, a more sophisticated model might rely on multiple features, each having a separate weight (w_1, w_2, etc.). For example, a model that relies on three features might look as follows:

$y'=b+w_1x_1+w_2x_2+w_3x_3$

Finding the best slope

$$m = \frac{\overline{x} \cdot \overline{y} - \overline{xy}}{(\overline{x})^2 - \overline{x^2}}$$

from statistics import mean

import numpy as np

```
xs = np.array([1,2,3,4,5], dtype=np.float64)

ys = np.array([5,4,6,5,6], dtype=np.float64)

def best_fit_slope(xs,ys):

    m = (((mean(xs)*mean(ys)) - mean(xs*ys)) /

         ((mean(xs)**2) - mean(xs**2)))

    return m

m = best_fit_slope(xs,ys)

print(m)
```

finding the best bias

$$b = \overline{y} - m\overline{x}$$

```
def best_fit_slope_and_intercept(xs,ys):

    m = (((mean(xs)*mean(ys)) - mean(xs*ys)) /

         ((mean(xs)*mean(xs)) - mean(xs*xs)))

    b = mean(ys) - m*mean(xs)

    return m, b
```

4.2 Modules

4.2.1 Numpy:-

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric,[4] also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called Numarray was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Traits

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

4.2.2 Pandas:-

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects

- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets

4.2.3 Seaborn:-

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

An introduction to seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

Here is some of the functionality that seaborn offers:

- A dataset-oriented API for examining relationships between multiple variables
- Specialized support for using categorical variables to show observations or aggregate statistics
- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds dependent variables
- Convenient views onto the overall structure of complex datasets

- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

A few things have happened here. Let's go through them one by one:

1. We import seaborn, which is the only library necessary for this simple example.

```
importseabornassns
```

Behind the scenes, seaborn uses matplotlib to draw plots. Many tasks can be accomplished with only seaborn functions, but further customization might require using matplotlib directly. This is explained in more detail below. For interactive work, it's recommended to use a Jupyter/IPython interface in matplotlib mode, or else you'll have to call matplotlib.pyplot.show when you want to see the plot.

2. We apply the default default seaborn theme, scaling, and color palette.

```
sns.set()
```

This uses the matplotlib rcParam system and will affect how all matplotlib plots look, even if you don't make them with seaborn. Beyond the default theme, there are several other options, and you can independently control the style and scaling of the plot to quickly translate your work between presentation contexts (e.g., making a plot that will have readable fonts when projected during a talk). If you like the matplotlib defaults or prefer a different theme, you can skip this step and still use the seaborn plotting functions.

3. We load one of the example datasets.

```
tips = sns.load_dataset("tips")
```

Most code in the docs will use the **load_dataset()** function to get quick access to an example dataset. There's nothing particularly special about these datasets; they are just pandas dataframes, and we could have loaded them with pandas.read_csv or build them by hand. Many examples use the "tips" dataset, which is very boring but quite useful for demonstration. The tips dataset illustrates the "tidy" approach to organizing a dataset. You'll get the most out of seaborn if your datasets are organized this way, and it is explained in more detail below.

4. We draw a faceted scatter plot with multiple semantic variables.

```
sns.relplot(x="total_bill", y="tip", col="time",
        hue="smoker", style="smoker", size="size",
        data=tips)
```

Statistical estimation and error bars

Often we are interested in the average value of one variable as a function of other variables. Many seaborn functions can automatically perform the statistical estimation that is neccesary to answer these questions:

```
fmri = sns.load_dataset("fmri")
sns.relplot(x="timepoint", y="signal", col="region",
        hue="event", style="event",
        kind="line", data=fmri);
```

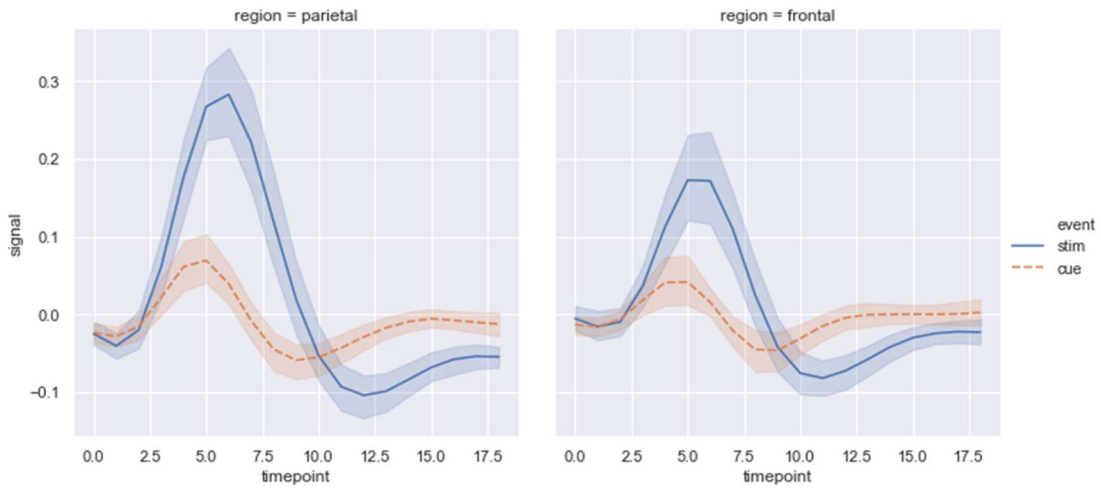

Figure No 4.3 Time point

When statistical values are estimated, seaborn will use bootstrapping to compute confidence intervals and draw error bars representing the uncertainty of the estimate.

Statistical estimation in seaborn goes beyond descriptive statisitics. For example, it is also possible to enhance a scatterplot to include a linear regression model (and its uncertainty) using **lmplot()**:

```
sns.lmplot(x="total_bill", y="tip", col="time", hue="smoker",
        data=tips);
```

Specialized categorical plots

Standard scatter and line plots visualize relationships between numerical variables, but many data analyses involve categorical variables. There are several specialized plot types in seaborn that are optimized for visualizing this kind of data. They can be accessed through **catplot()**. Similar to **relplot()**, the idea of **catplot()** is that it exposes a common dataset-oriented API that generalizes over different representations of the relationship between one numeric variable and one (or more) categorical variables.

These representations offer different levels of granularity in their presentation of the underlying data. At the finest level, you may wish to see every observation by drawing a

scatter plot that adjusts the positions of the points along the categorical axis so that they don't overlap:

```
sns.catplot(x="day", y="total_bill", hue="smoker",
        kind="swarm", data=tips);
```

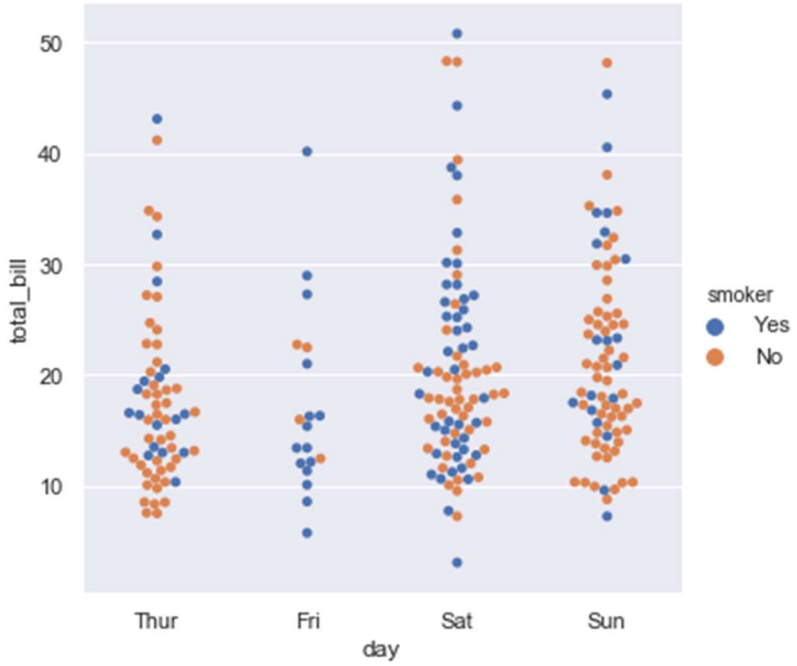

Figure No 4.4 Day Total Bill

Alternately, you could use kernel density estimation to represent the underlying

```
sns.catplot(x="day", y="total_bill", hue="smoker",
        kind="bar", data=tips);
```

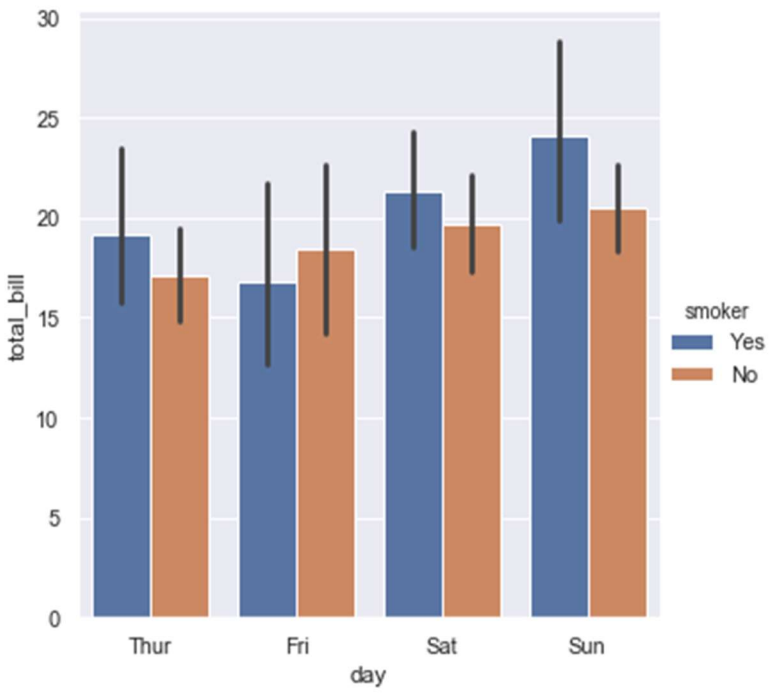

Figure No 4.5 level and axes-level functions

How do these tools work? It's important to know about a major distinction between seaborn plotting functions. All of the plots shown so far have been made with "figure-level" functions. These are optimized for exploratory analysis because they set up the matplotlib figure containing the plot(s) and make it easy to spread out the visualization across multiple axes. They also handle some tricky business like putting the legend outside the axes. To do these things, they use a seaborn FacetGrid.

Each different figure-level plot kind combines a particular "axes-level" function with the **FacetGrid** object. For example, the scatter plots are drawn using the **scatterplot()** function, and the bar plots are drawn using the barplot() function. These functions are called "axes-level" because they draw onto a single matplotlib axes and don't otherwise affect the rest of the figure.

The upshot is that the figure-level function needs to control the figure it lives in, while axes-level functions can be combined into a more complex matplotlib figure with other axes that may or may not have seaborn plots on them:

Controling the size of the figure-level functions works a little bit differently than it does for other matplotlib figures. Instead of setting the overall figure size, the figure-level functions are parameterized by the size of each facet. And instead of setting the height and width of each facet, you control the height and *aspect* ratio (ratio of width to height). This parameterization makes it easy to control the size of the graphic without thinking about exactly how many rows and columns it will have, although it can be a source of confusion:

```
sns.relplot(x="time", y="firing_rate", col="align",
            hue="choice", size="coherence", style="choice",
            height=4.5, aspect=2/3,
            facet_kws=dict(sharex=False),
            kind="line", legend="full", data=dots);
```

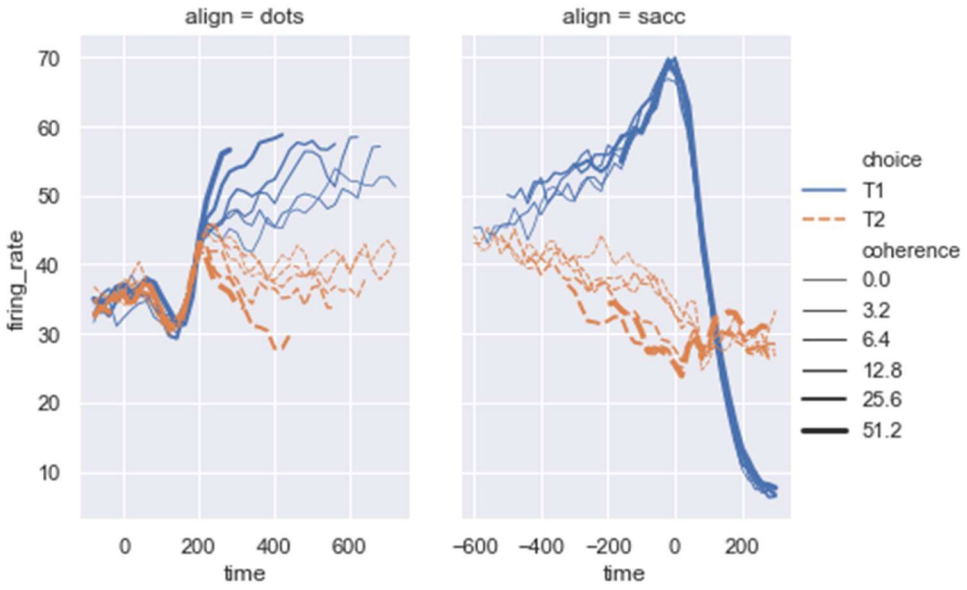

Figure No 4.6 Time rate

The way you can tell whether a function is “figure-level” or “axes-level” is whether it takes an ax= parameter. You can also distinguish the two classes by their output type:

axes-level functions return the matplotlib axes, while figure-level functions return the FacetGrid

Loading an example dataset

scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the boston house prices dataset for regression.

In the following, we start a Python interpreter from our shell and then load the iris and digits datasets. Our notational convention is that $ denotes the shell prompt while >>> denotes the Python interpreter prompt:

```
$ python
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the .data member, which is a n_samples, n_features array. In the case of supervised problem, one or more response variables are stored in the .target member. More details on the different datasets can be found in the dedicated section.

4.2.4 Matplotlib:-

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like

OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Matplotlib was originally written by John D. Hunter, since then it has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

Matplotlib requires the following dependencies:

- Python (>= 3.5)
- FreeType (>= 2.3)
- libpng (>= 1.2)
- NumPy (>= 1.10.0)
- setuptools
- cycler (>= 0.10.0)
- dateutil (>= 2.1)
- kiwisolver (>= 1.0.0)
- pyparsing

Optionally, you can also install a number of packages to enable better user interface toolkits. See What is a backend? for more details on the optional Matplotlib backends and the capabilities they provide.

- tk (>= 8.3, != 8.6.0 or 8.6.1): for the Tk-based backends;
- PyQt4 (>= 4.6) or PySide (>= 1.0.3): for the Qt4-based backends;
- PyQt5: for the Qt5-based backends;
- PyGObject or pgi (>= 0.0.11.2): for the GTK3-based backends;
- wxpython (>= 4): for the WX-based backends;
- cairocffi (>= 0.8) or pycairo: for the cairo-based backends;
- Tornado: for the WebAgg backend;

For better support of animation output format and image file formats, LaTeX, etc., you can install the following:

- ffmpeg/avconv: for saving movies;
- ImageMagick: for saving animated gifs;
- Pillow (>= 3.4): for a larger selection of image file formats: JPEG, BMP, and TIFF image files;
- LaTeX and GhostScript (>=9.0) : for rendering text with LaTeX.

Matplotlib 2.0.x supports Python versions 2.7 through 3.6. Python 3 support started with Matplotlib 1.2. Matplotlib 1.4 is the last version to support Python 2.6 Matplotlib has pledged to not support Python 2 past 2020 by signing the Python 3 Statement.

Pyplot is a Matplotlib module which provides a MATLAB-like interface.] Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source.

The **matplotlib** Python library, developed by John Hunter and many other contributors, is used to create high-quality graphs, charts, and figures. The library is extensive and capable of changing very minute details of a figure. Some basic concepts and functions provided in matplotlib are:

1. Figure and axes

The entire illustration is called a figure and each plot on it is an axes (do not confuse *Axes* with *Axis*). The figure can be thought of as a canvas on which several plots can be drawn. We obtain the figure and the axes using the subplots() function:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
```

2. Plotting

The very first thing required to plot a graph is data. A dictionary of key-value pairs can be declared, with keys and values as the x and y values. After that, scatter(), bar(), and pie(), along with tons of other functions, can be used to create the plot:

```
# Create data:
data = {
    "France": 65.4,
    "Germany": 82.4,
    "Italy": 59.2,
    "UK": 66.9
}

# Use keys and values as x and y axis values:
x_axis_data = data.keys()
y_axis_data = data.values()

# Plotting a bar graph:
ax.bar(x_axis_data, y_axis_data)
```

3. Axis

The figure and axes obtained using subplots() can be used for modification. Properties of the x-axis and y-axis (labels, minimum and maximum values, etc.) can be changed using Axes.set():

```
# Setting properties of axes:
```

```
ax.set(ylim=[0, 100],
  ylabel='Population (in million)',
  xlabel='Country',
  title='European Countries by Population')
```

Some properties of a plot.

Lastly, pyplot.show() is used to display the graph.

Code

The following code demonstrates all of the points discussed above:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

# Create data:
data = {
    "France": 65.4,
    "Germany": 82.4,
    "Italy": 59.2,
    "UK": 66.9
}
```

```
# Use keys and values as x and y axis values:

x_axis_data = data.keys()

y_axis_data = data.values()



# Plotting a bar graph:

ax.bar(x_axis_data, y_axis_data)



# Setting properties of axes:

ax.set(ylim=[0, 100],

  ylabel='Population (in million)',

  xlabel='Country',

  title='European Countries by Population')



# Displaying the graph:

plt.show()
```

4.2.5 Scikit-learn

Defining scikit learn, it is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.Later Matthieu Brucher joined the project and started to use it as a part of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.The project now has more than 30 active contributors and has had paid sponsorship from INRIA, Google, Tinyclues and the Python Software Foundation.

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features.

Learning problems fall into a few categories:

- supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page).This problem can be either:
    - classification: samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data. An example of a classification problem would be handwritten digit recognition, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited

number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

- regression: if the desired output consists of one or more continuous variables, then the task is called regression. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

- unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization (Click here to go to the Scikit-Learn unsupervised learning page).

Training set and testing set

Machine learning is about learning some properties of a data set and then testing those properties against another data set. A common practice in machine learning is to evaluate an algorithm by splitting a data set into two. We call one of those sets the training set, on which we learn some properties; we call the other set the testing set, on which we test the learned properties.

Loading an example dataset

scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the boston house prices dataset for regression.

In the following, we start a Python interpreter from our shell and then load the iris and digits datasets. Our notational convention is that $ denotes the shell prompt while >>> denotes the Python interpreter prompt:

```
$ python
```

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the .data member, which is a n_samples, n_features array. In the case of supervised problem, one or more response variables are stored in the .target member. More details on the different datasets can be found in the dedicated section.

For instance, in the case of the digits dataset, digits.data gives access to the features that can be used to classify the digits samples:

```
>>>
```

```
>>>print(digits.data)
[[ 0.   0.   5. ...   0.   0.   0.]
 [ 0.   0.   0. ...  10.   0.   0.]
 [ 0.   0.   0. ...  16.   9.   0.]
 ...
 [ 0.   0.   1. ...   6.   0.   0.]
 [ 0.   0.   2. ...  12.   0.   0.]
 [ 0.   0.  10. ...  12.   1.   0.]]
```

Learning and predicting

In the case of the digits dataset, the task is to predict, given an image, which digit it represents. We are given samples of each of the 10 possible classes (the digits zero

through nine) on which we fit an estimator to be able to predict the classes to which unseen samples belong.

In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).

An example of an estimator is the class sklearn.svm.SVC, which implements support vector classification. The estimator's constructor takes as arguments the model's parameters.

For now, we will consider the estimator as a black box:

```
>>>
```

```
>>> fromsklearnimport svm
>>>clf = svm.SVC(gamma=0.001, C=100.)
```

Choosing the parameters of the model

In this example, we set the value of gamma manually. To find good values for these parameters, we can use tools such as grid search and cross validation.

CHAPTER 5

IMPLEMENTATION

5.1 Implementation

The Implementation phase describes about price prediction accuracy performance based on recorded data with algorithms by training the data model in more number of times to achieve the more accuracy.

5.2 Source Code

```
import numpy as np

xs = np.array([1,2,3,4,5], dtype=np.float64)

ys = np.array([5,4,6,5,6], dtype=np.float64)

def best_fit_slope_and_intercept(xs,ys):

    m = (((mean(xs)*mean(ys)) - mean(xs*ys)) /

         ((mean(xs)*mean(xs)) - mean(xs*xs)))

    b = mean(ys) - m*mean(xs)

    return m, b

m, b = best_fit_slope_and_intercept(xs,ys)

print(m,b)
```

Finding best fit(Least Square Method)

$$m = \frac{\sum_{i=1}^{n}(x_i - \overline{X})(y_i - \overline{Y})}{\sum_{i=1}^{n}(x_i - \overline{X})^2}$$

Finding best bias

$$b = \overline{Y} - m\overline{X}$$

CHAPTER 6

TESTING & VALIDATION

6.1 Testing Methodologies

The testing phase is done with different methodologies to check functional and non-functional statements of the program. The validation is for validating the required input to get the desired output of the program.

The following are the Testing Methodologies:

- Unit Testing.
- Integration Testing.
- User Acceptance Testing.
- Output Testing.
- Validation Testing.

6.2 Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

6.3 Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

1. Top Down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

2. Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

- The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.
- A driver (i.e.) the control program for testing is written to coordinate test case input and output.
- The cluster is tested.\\
- Drivers are removed and clusters are combined moving upward in the program structure

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

6.4 User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

6.5 Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

6.6 Validation Checking

Validation checks are performed on the following fields.

Text Field:

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used in the program the existence of any program

defect is inferred from the output. The testing should be planned so that all the requirements are individually tested.

A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

Preparation of Test Data

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

6.7 Using Live Test Data

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact typical, such data generally will not test all combinations or formats that can enter the system. This bias toward typical values then does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

6.8 Using Artificial Test Data

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which

can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

The package "Virtual Private Network" has satisfied all the requirements specified as per software requirement specification and was accepted.

6.9 Testing Strategy

A strategy for system testing integrates system test cases and design techniques into a well-planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

CHAPTER 7

SCREENSHOTS

7.1 Screenshots

7.1.1 Stock Data

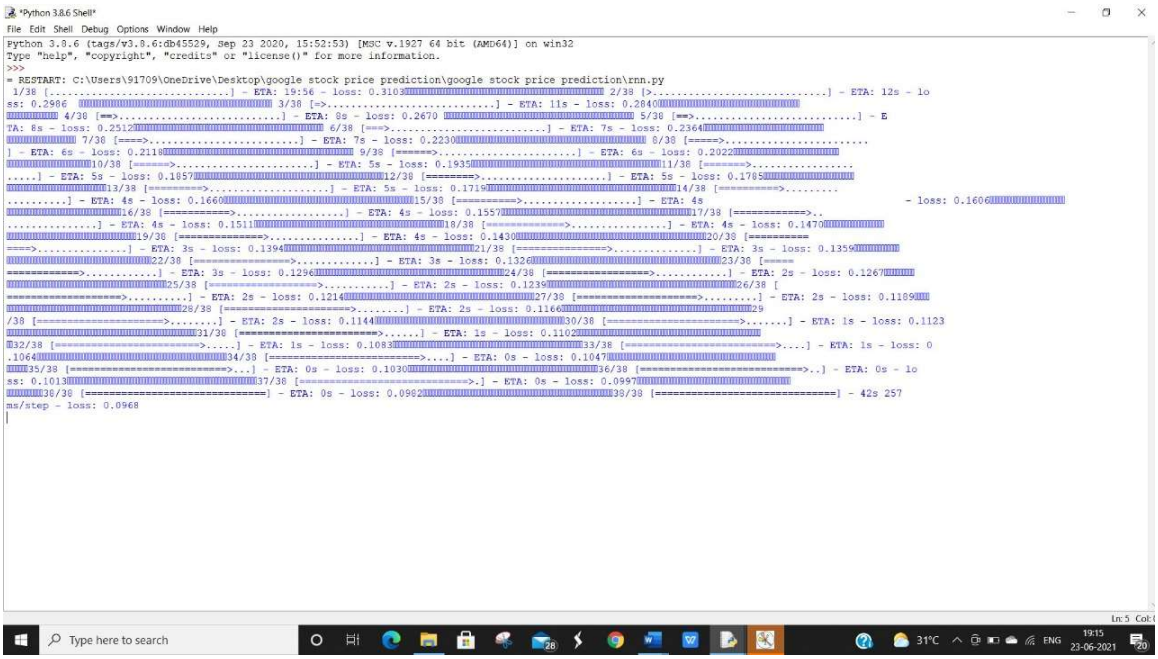

Figure No. 7.1 Stock Data

The Above Screenshot describes about the Stock Prediction Data

7.1.2 Graph Representation of Stock

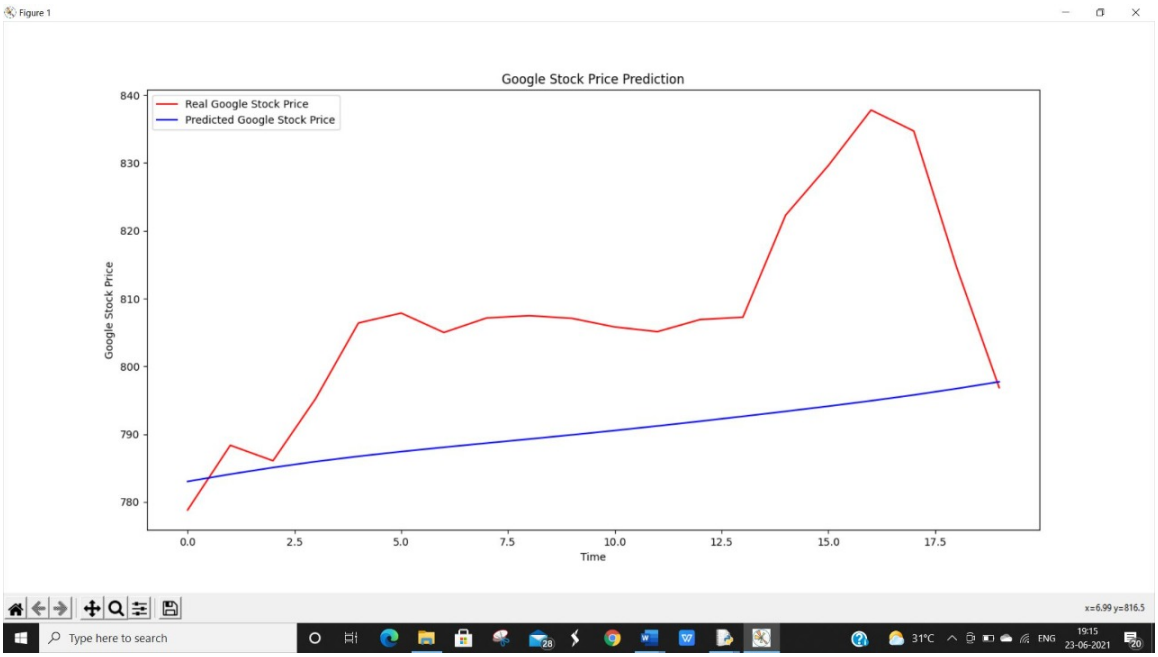

Figure No. 7.2 Graph Representation of Stock

The Above Screenshot describes about the Graph Representation of Stock

CHAPTER 8

CONCLUSION

8.1 Conclusion

No human or computer can perfectly predict the volatile stock market • Under "normal" conditions, in most cases, a good neural network will outperform most other current stock market predictors and be a very worthwhile, and potentially profitable aid to investors

• Should be used as an aid only!

8.2 Limitation

According to the above reviewed papers, it can be inferred that study on stock market prediction is still being raised among researchers. Also, it seems that hybrid methods are the permanent approach used in different researches. Considering the acceptance of ANN-based methods, the focus is to enhance the performance of ANN through some metaheuristics.

- Doesn't work well for nonlinear time series
- Requires more data
- Takes a long time processing for a large dataset
- Sensitive to noise
- Actual performance based on initial values
- Slow convergent speed
- Easily converging to a local minimum
- Unstable even when the training data are small changed
- The number of clusters must be specified in advance
- Sensitive to noise
- Only spherical shapes can be determined as clusters

- Unable to handle long time series effectively because of poor scalability

8.3 Future Scope

Time series is a main method which is used for the prediction of share prices. Time series analysis deals with analyzing a series of data gathered during time. Time series are common in different fields of economy, finance, healthcare, etc (Bisgaard and Kulahci 2011). This method tries to forecast future by assuming that the previously observed pattern can be considered as the foundation to extract future behavior (Shin 2017). Heuristic algorithms are another set of methods being used for prediction. Heuristic algorithms are often used as an alternative optimization algorithm, instead on exact methods that usually deal with finding a good feasible solution without any assurance of being optimal (Kaveh and Ghazaan 2018).

REFERENCES

Zamora, E., & Sossa, H. (2017). Dendrite morphological neurons trained by stochastic gradient descent. Neurocomputing, 260 , 420–431.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 , .

Zhan, X., Li, Y., Li, R., Gu, X., Habimana, O., & Wang, H. (2018). Stock price prediction using time convolution long short-term memory network.

In W. Liu, F. Giunchiglia, & B. Yang (Eds.), Knowledge Science, Engineering and Management (pp. 461–468). Cham: Springer International Publishing.

Zhang, J., Rong, W., Liang, Q., Sun, H., & Xiong, Z. (2017a). Data augmentation based stock trend prediction using self-organising map. In D. Liu, S. Xie, Y. Li, D. Zhao, & E.-S. M. El-Alfy (Eds.), Neural Information Processing (pp. 903–912).

Cham: Springer International Publishing. Zhang, K., Zhong, G., Dong, J., Wang, S., & Wang, Y. (2019a).

Stock market prediction based on generative adversarial network. Procedia Computer Science, 147 , 400 – 406. URL: http: //www.sciencedirect.com/science/article/pii/S1877050919302789. doi:https://doi.org/10.1016/j.procs.2019.01.256. 2018 International Conference on Identification, Information and Knowledge in the Internet of Things.

Zhang, L., Aggarwal, C., & Qi, G.-J. (2017b). Stock price prediction via discovering multi-frequency trading patterns. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD ’17 (pp. 2141–2149).

New York, NY, USA: ACM. URL: http:// doi.acm.org/10.1145/3097983.3098117. doi:10.1145/3097983.3098117. Zhang, Z., Zohren, S., & Roberts, S. (2019b).

Deeplob: Deep convolutional neu96 ral networks for limit order books. IEEE Transactions on Signal Processing, 67 , 3001–3012.