INTRODUCTION

1. Introduction

Stock Market prediction and analysis is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. Stock market is the important part of economy of the country and plays a vital role in the growth of the industry and commerce of the country that eventually affects the economy of the country. Both investors and industry are involved in stock market and wants to know whether some stock will rise or fall over certain period of time. The stock market is the primary source for any company to raise funds for business expansions. It is based on the concept of demand and supply. If the demand for a company's stock is higher, then the company share price increases and if the demand for company's stock is low then the company share price decrease.

1.1 Objective

Our project Aim is to Predict the Risk Factor in stock market.It is intended to solve the economic dilemma created in individuals who wants to invest in stock market and generates an approximate forecasting output and create a general idea for future values based on Historical data .

1.2 Problem Statement:

Stock market is very vast and difficult to understand.It is considered too uncertain to be predictable due to huge fluctuations in the market.

To overcome this situation, we use developed tool.

Vision: To suggest the investors for obtaining the optimum results.

Mission: To develop a tool which predicts risk in equities using python technologies.

1.3 Existing System

The prediction of stock market is without doubt an interesting task. In the literature there are a number of methods applied to accomplish this task. These methods use various approaches, ranging from highly informal ways (e.g. the study of a chart with the fluctuation of the market) to more formal ways.

Some of the methods are:

Analytical methods: Before the age of computers, people traded stocks and commodities primarily on intuition. As the level of investing and trading grew, people searched for tools and methods that would increase their gains while minimizing their risk. Statistics, technical analysis and fundamental analysis are all used to attempt to predict and benefit from the market's direction.

Technical Analysis:

Technical analysis is the method of predicting the appropriate time to buy or sell a stock used by those believing in the castles-in-the-air view of stock pricing.

Fundamental Analysis:

Fundamental analysis is the technique of applying the tenets of the firm foundation theory to selection of individual stocks

Other Computer Techniques:

Many other computer based techniques have been employed to forecast the stock market. They range from charting programs to sophisticated expert systems. Fuzzy logic has also been used.

Expert systems process knowledge sequentially and formulate it into rules. Expert systems are only good within their domain of knowledge and do not work well when there is missing or incomplete information.

1.4 Proposed System

An automated tool is developed by using Python along with its layout toolkit PyQt.Various widgets of PyQt are elaborately used in the forms of this tool as needed.This tool is used to analyze previous stock data with help of parameters like Previous Close ,Day Low, Day high and Previous Var . Tool is Designed To Assist Investors Using Neural Network of Tensorflow By Identifying Risk Equities Based On Historical Data.Tool is developed for Identifying Risk Factor in Equities.

SYSTEM REQUIREMENTS

2.1 Software Requirements

- 64 bit windows operating system
- Python
- Python Qt Designer for designing user interface.
- Tensorflow
- SQLite
- Pyuic for converting the layout designed user interface (UI) to python Code.

Windows Operating System 64-bit.Update it to the final version, it has many updated software compared to previous version.Microsoft Windows is a group of several graphical operating system families, all of which are developed, marketed, and sold by Microsoft. Each family caters to a certain sector of the computing industry. Active Windows families include Windows NT and Windows Embedded these may encompass subfamilies, e.g. Windows Embedded Compact (Windows CE) or Windows Server.

2.1.1 Installation of Python

To install Python and other scientific computing and machine learning packages simultaneously one should install Anaconda distribution. It is a Python implementation for Linux, Windows and OSX, and comprises various machine learning packages like numpy, scikit-learn, and matplotlib.

It also includes Jupyter Notebook, an interactive Python environment. Install Python 3.x version as per requirement.

To download the free Anaconda Python distribution from Continuum Analytics, one can do the following

Visit the official site of Continuum Analytics and its download page. Note that the installation process may take 15-20 minutes as the installer contains Python, associated packages, a code editor, and some other files. Depending on operating system, choose the installation process as explained here.

For Windows − Select the Anaconda for Windows section and look in the column with Python 2.7 or 3.x. one can find that there are two versions of the installer, one for 32-bit Windows, and one for 64-bit Windows. Choose the relevant one. Ensure that Anaconda's Python distribution installs into a single directory, and does not affect other Python installations, if any, on system.

Now, check if installation is successful. For this, go to the command line and type in the following command –

```
$ python
Python 3.5.2 |Anaconda custom (32-bit)| (default, Oct 13 2017, 14:21:34)
```

Python Language

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast.

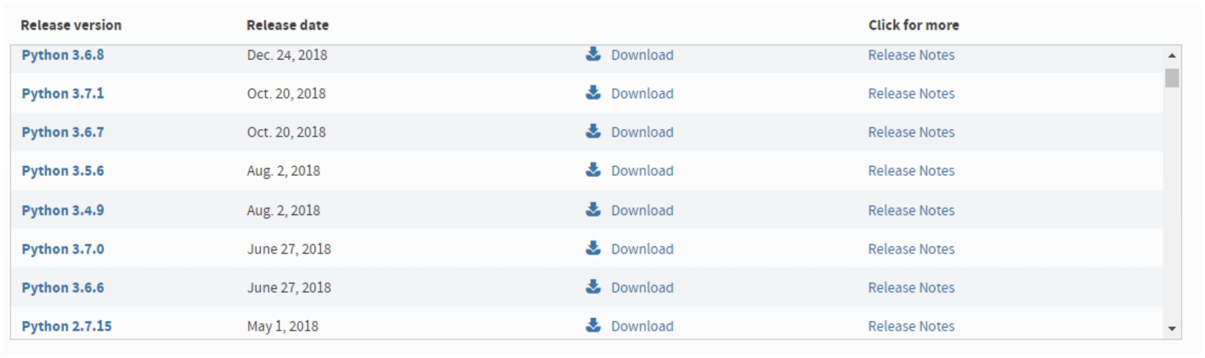
Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source.

Downloading Python

So to get started, here's how you can download the latest 64-bit Python 3.5.x if you have an older version or if you simply don't have it.

Step 1: Head over to Python 3.5.x from python.org

Step 2: Go to the Downloads page and Select the 3.5.2 download.



Release version	Release date		Click for more
Python 3.6.8	Dec. 24, 2018	Download	Release Notes
Python 3.7.1	Oct. 20, 2018	Download	Release Notes
Python 3.6.7	Oct. 20, 2018	Download	Release Notes
Python 3.5.6	Aug. 2, 2018	Download	Release Notes
Python 3.4.9	Aug. 2, 2018	Download	Release Notes
Python 3.7.0	June 27, 2018	Download	Release Notes
Python 3.6.6	June 27, 2018	Download	Release Notes
Python 2.7.15	May 1, 2018	Download	Release Notes

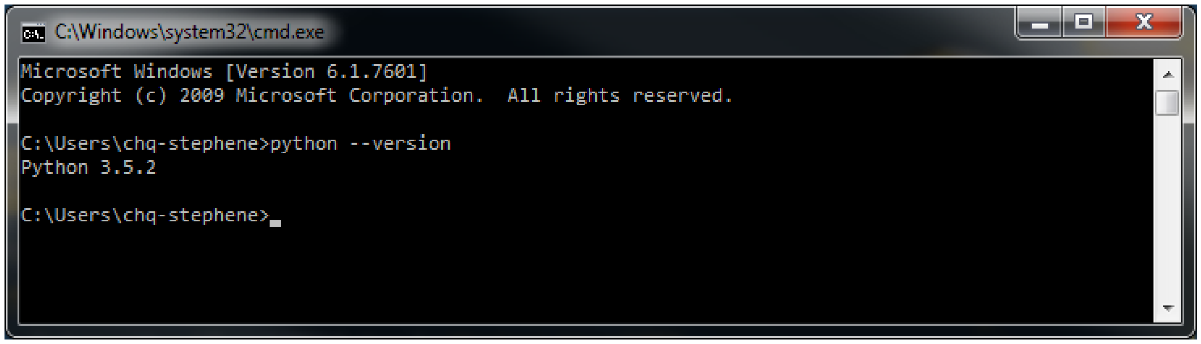
Fig 2.1: Python Downloads page

Step 3: After that you will be brought to another page, where you will need to select either the x86-64 or amd64 installer.

The one I specifically recommend for now is the Windows x86-64 executable installer.

Step 4: choose to Add Python 3.5 to PATH.

Step 5: Now you should be able to see a message saying Setup was successful. A way to confirm that it has installed successfully is to open your Command Prompt and check the version.



C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\chq-stephene>python --version
Python 3.5.2

C:\Users\chq-stephene>
```

Fig 2.2: Python version check

2.1.2 Installation of PyQt5

Qt:

Qt is designed for developing applications and user interfaces once and deploying them across several desktop and mobile operating systems.The easiest way to start application development with Qt is to download and install Qt5. It contains Qt libraries, examples, documentation, and the necessary development tools, such as the Qt Creator integrated development environment (IDE).Qt Creator provides tools for accomplishing tasks throughout the whole application development life-cycle, from creating a project to deploying the application on the target platforms. Qt Creator automates some tasks, such as creating projects, by providing wizards that guide step-by-step through the project creation process, create the necessary files, and specify settings depending on the choices we make.Also, it speeds up some tasks, such as writing code, by offering semantic highlighting, checking code syntax, code completion, refactoring actions, and other useful features.

Python Qt Designer

The PyQt installer comes with a GUI builder tool called Qt Designer. Using its simple drag and drop interface, a GUI interface can be quickly built without having to write the code. It is however, not an IDE such as Visual Studio. Hence, Qt Designer does not have the facility to debug and build the application. Creation of a GUI interface using Qt Designer starts with choosing a top level window for the application we can then drag and drop required widgets from the widget box on the left pane. we can also assign value to properties of widget laid on the form. The designed form is saved as demo.ui. This ui file contains XML representation of widgets and their properties in the design. This design is translated into Python equivalent by using pyuic4 command line utility. This utility is a wrapper for ui module. The usage of pyuic4 is as follows

Widgets & Description

QLabel

A QLabel object acts as a placeholder to display non-editable text or image, or a movie of animated GIF. It can also be used as a mnemonic key for other widgets. QLineEdit QLineEdit object is the most commonly used input field. It provides a box in 2 which one line of text can be entered. In order to enter multi-line text, QTextEdit object is required.

QPushButton

In PyQt API, the QPushButton class object presents a button which when clicked can be programmed to invoke a certain function.

The QPushButton widget provides a command button.

The push button, or command button, is perhaps the most commonly used widget in any graphical user interface. Push (click) a button to command the computer to perform some action, or to answer a question. Typical buttons are OK, Apply, Cancel, Close, Yes, No and Help. A command button is rectangular and typically displays a text label describing its action.Push buttons display a textual label, and optionally a small icon.

QLineEdit

The QLineEdit widget is a one-line text editor.

A line edit allows the user to enter and edit a single line of plain text with a useful collection of editing functions, including undo and redo, cut and paste, and drag and drop. By changing the echoMode() of a line edit, it can also be used as a "write-only" field, for inputs such as passwords.

The length of the text can be constrained to maxLength(). The text can be arbitrarily constrained using a validator() or an inputMask(), or both. When switching between a validator and an input mask on the same line edit, it is best to clear the validator or input mask to prevent undefined behavior.

A related class is QTextEdit which allows multi-line, rich text editing. we can change the text with setText() or insert(). The text is retrieved with text(); the displayed text (which may be different, see EchoMode) is retrieved with displayText(). Text can be selected with setSelection() or selectAll(), and the selection can be cut(),copy()ied and paste()d. The text can be aligned with setAlignment().When the text changes the textChanged() signal is emitted; when the text changes other than by calling setText() the textEdited() signal is emitted; when the cursor is moved the cursorPositionChanged() signal is emitted; and when the Return or Enter key is pressed the returnPressed() signal is emitted.When editing is finished, either because the line edit lost focus or Return/Enter is pressed the editingFinished() signal is emitted.

Note that if there is a validator set on the line edit, the returnPressed()/editingFinished() signals will only be emitted if the validator returns QValidator.Acceptable. By default, QLineEdits have a frame as specified by the Windows and Motif style guides; we can turn it off by calling setFrame(false).

The default key bindings are described below. The line edit also provides a context menu (usually invoked by a right mouse click) that presents some of these editing Options.

PyQt5 Libraries:

Once we have both Python and pip installed, use the following commands to install the PyQt5 Designer, tools, and packages.

```
pip install pyqt5-installer
pip install pyqt5
pip install pyqt5-tools
```

we'll find different libraries with PyQt5 folders installed at this default location: C:\Program Files (x86)\Python36-32\Lib\site-packages (on Windows OS).

2.1.3 Tensorflow Installation

TensorFlow is an open-source machine learning library for research and production. TensorFlow offers APIs for beginners and experts to develop for desktop, mobile, web, and cloud. See the sections below to get started.

Before you go on with the steps, make sure that your computer meets the requirements in order for TensorFlow to work on your computer.

The following are the necessary requirements:

- TensorFlow only supports 64-bit Python 3.5.x or Python 3.6.x on Windows.
- When you download the Python 3.5.x version, it comes with the pip3 package manager (which is the program that you are going to need in order for you use to install TensorFlow on Windows).

Now once you have downloaded the latest Python, you can now put up your finishing touches by installing your TensorFlow.

Step 1: To install TensorFlow, start a terminal. Make sure that you run the cmd as an administrator.

Here’s how you can run your cmd as an administrator:

Open up the Start menu, search for cmd and then right click on it and Run as an administrator.

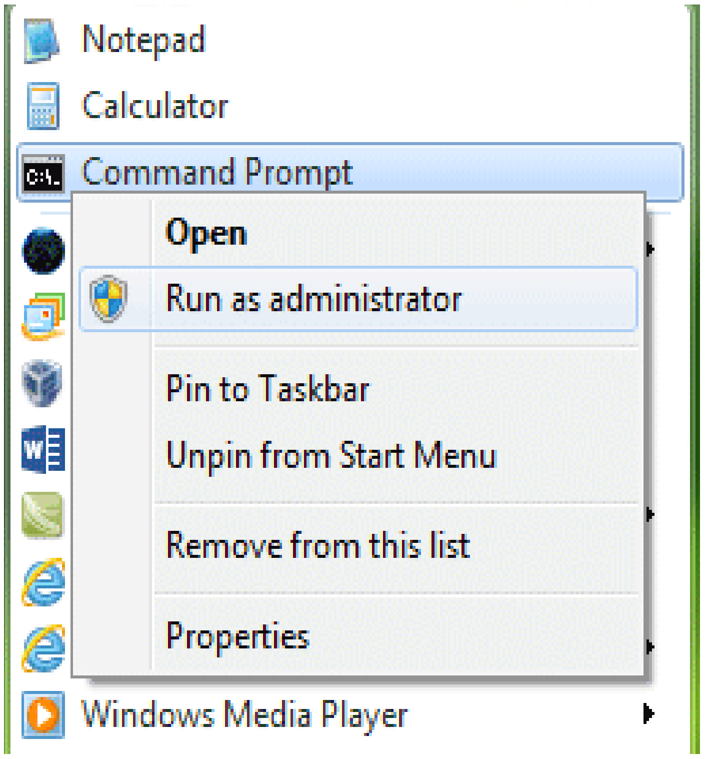


Fig 2.3: Running Command Prompt as Administrator

Step 2: Once you're done with that, now all you have to do is give just one simple little command for you to finish installing Tensorflow onto your Windows.

Just enter this command:

Create a environment named tensorflow by invoking the following command:

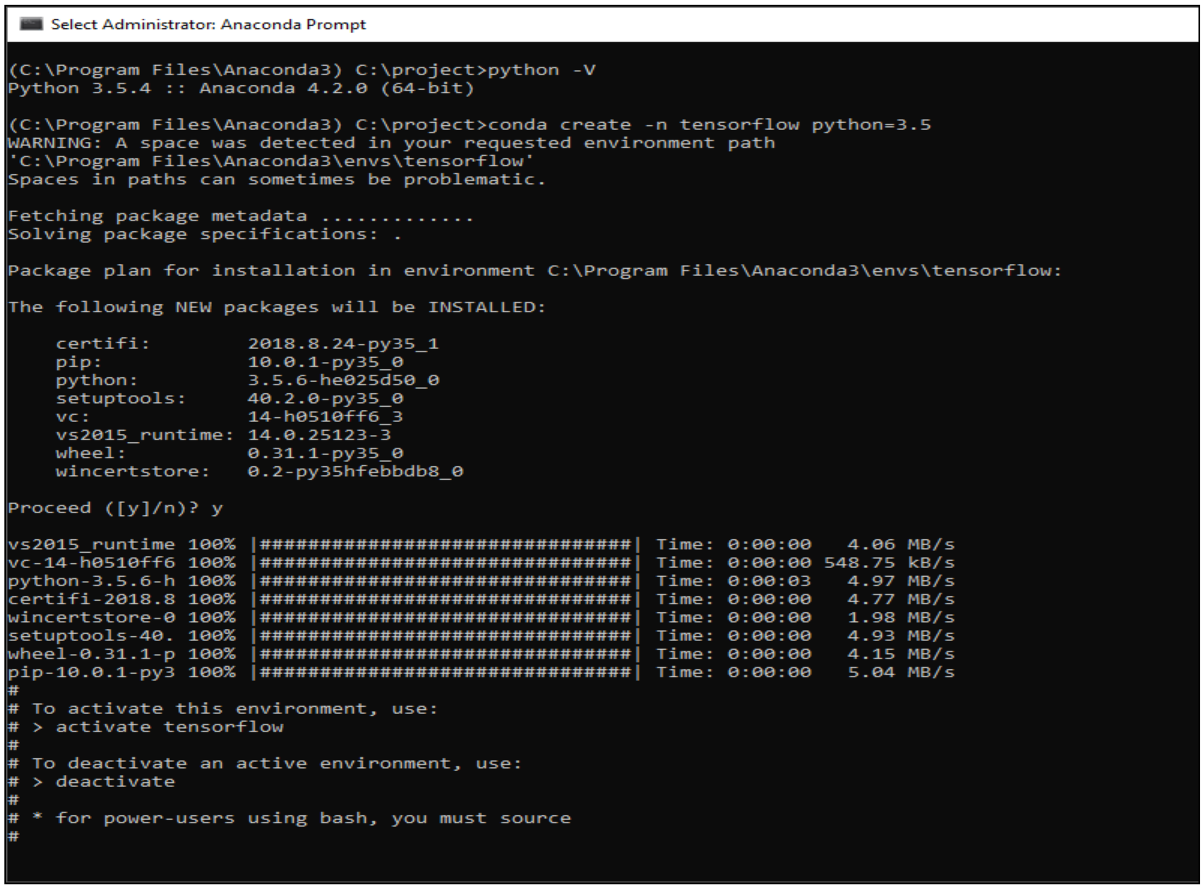
```
conda create -n tensorflow python=3.5
```

Activate the conda environment by issuing the following command:

```
activate tensorflow
```

Install TensorFlow:

```
conda install -c conda-forge tensorflow
```



```
(C:\Program Files\Anaconda3) C:\project>python -V
Python 3.5.4 :: Anaconda 4.2.0 (64-bit)

(C:\Program Files\Anaconda3) C:\project>conda create -n tensorflow python=3.5
WARNING: A space was detected in your requested environment path
'C:\Program Files\Anaconda3\envs\tensorflow'
Spaces in paths can sometimes be problematic.

Fetching package metadata .............
Solving package specifications: .

Package plan for installation in environment C:\Program Files\Anaconda3\envs\tensorflow:

The following NEW packages will be INSTALLED:

    certifi:        2018.8.24-py35_1
    pip:            10.0.1-py35_0
    python:         3.5.6-he025d50_0
    setuptools:     40.2.0-py35_0
    vc:             14-h0510ff6_3
    vs2015_runtime: 14.0.25123-3
    wheel:          0.31.1-py35_0
    wincertstore:   0.2-py35hfebbdb8_0

Proceed ([y]/n)? y

vs2015_runtime 100% |###############################| Time: 0:00:00   4.06 MB/s
vc-14-h0510ff6 100% |###############################| Time: 0:00:00 548.75 kB/s
python-3.5.6-h 100% |###############################| Time: 0:00:03   4.97 MB/s
certifi-2018.8 100% |###############################| Time: 0:00:00   4.77 MB/s
wincertstore-0 100% |###############################| Time: 0:00:00   1.98 MB/s
setuptools-40. 100% |###############################| Time: 0:00:00   4.93 MB/s
wheel-0.31.1-p 100% |###############################| Time: 0:00:00   4.15 MB/s
pip-10.0.1-py3 100% |###############################| Time: 0:00:00   5.04 MB/s
#
# To activate this environment, use:
# > activate tensorflow
#
# To deactivate an active environment, use:
# > deactivate
#
# * for power-users using bash, you must source
#
```

Fig 2.4: Tensorflow installation

After that we can check if all is correct by invoking python and trying the next program:

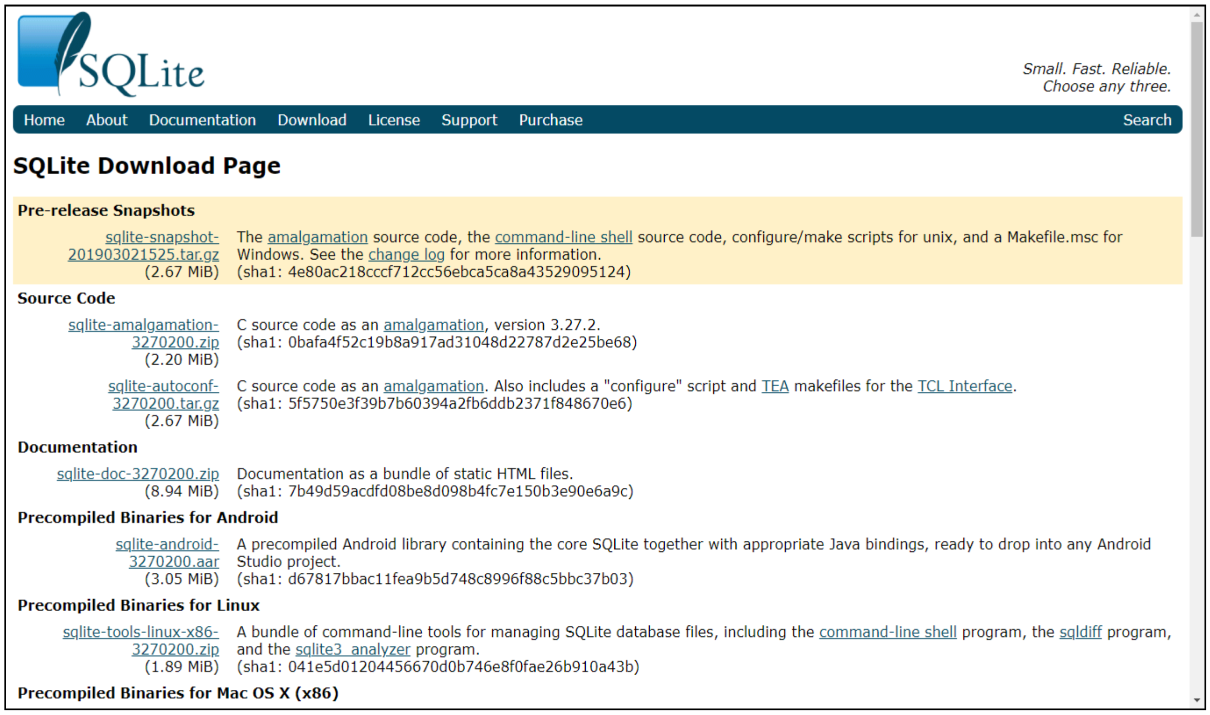
```
import tensorflow as tf

hail = tf.constant('Hello World')

session = tf.Session()

print(session.run(hail))
```

2.1.4 SQLite3 Installation

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format.

Download sqlite-tools-win32-x86-3240000.zip from official website sqlite.org.



SQLite Download Page

Pre-release Snapshots

sqlite-snapshot-201903021525.tar.gz (2.67 MiB)	The amalgamation source code, the command-line shell source code, configure/make scripts for unix, and a Makefile.msc for Windows. See the change log for more information. (sha1: 4e80ac218cccf712cc56ebca5ca8a43529095124)

Source Code

sqlite-amalgamation-3270200.zip (2.20 MiB)	C source code as an amalgamation, version 3.27.2. (sha1: 0bafa4f52c19b8a917ad31048d22787d2e25be68)
sqlite-autoconf-3270200.tar.gz (2.67 MiB)	C source code as an amalgamation. Also includes a "configure" script and TEA makefiles for the TCL Interface. (sha1: 5f5750e3f39b7b60394a2fb6ddb2371f848670e6)

Documentation

sqlite-doc-3270200.zip (8.94 MiB)	Documentation as a bundle of static HTML files. (sha1: 7b49d59acdfd08be8d098b4fc7e150b3e90e6a9c)

Precompiled Binaries for Android

sqlite-android-3270200.aar (3.05 MiB)	A precompiled Android library containing the core SQLite together with appropriate Java bindings, ready to drop into any Android Studio project. (sha1: d67817bbac11fea9b5d748c8996f88c5bbc37b03)

Precompiled Binaries for Linux

sqlite-tools-linux-x86-3270200.zip (1.89 MiB)	A bundle of command-line tools for managing SQLite database files, including the command-line shell program, the sqldiff program, and the sqlite3_analyzer program. (sha1: 041e5d01204456670d0b746e8f0fae26b910a43b)

Precompiled Binaries for Mac OS X (x86)

Fig 2.5: SQLite Download Page

Open .exe file and install it.

2.2 Hardware Requirements

- Processor - 2.1GHz
- RAM – 8GB
- 64 bit architecture
- Storage of 1TB GB

2.3 Feasibility Study

Economical Feasibility:

In Identification of risk equities in stock market we are using the open source software. We are using Windows operating system.Tensorflow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. For storing the data in the database we are using SQLite,it is an open source. It is secure, easy to use and inexpensive. We are using PyQt designer tool for designing the user interface screens. As all the above mentioned are open source. They are economically feasible.

Technical Feasibility:

A prototype of Identification of risk equities in stock market)tool was developed to verify the technical feasibility. The prototype is working successfully and hence the project is technically feasible.

LITERATURE SURVEY

3.1 Survey-1

Title	**TensorFlow Estimators: Managing Simplicity vs. Flexibility in High-Level Machine Learning Frameworks**
Authors	Heng-Tze Cheng, Zakaria Haque, Lichan Hong, Mustafa Ispir, Clemens Mewald, Illia Polosukhin, Georgios Roumpos, D Sculley, Jamie Smith, David Soergel, Yuan Tang, Philipp Tucker, Martin Wicke, Cassandra Xia, Jianwei Xie
Year of Publication	Aug 2017
Summary	In this paper a framework is presented for specifying, training, evaluating, and deploying machine learning models. Our focus is on simplifying cutting edge machine learning for practitioners in order to bring such technologies into production. Recognizing the fast evolution of the eld of deep learning, we make no attempt to capture the design space of all possible model architectures in a domain- specific language (DSL) or similar configuration language. We allow users to write code to define their models, but provide abstractions that guide developers to write models in ways conducive to productionisation . We also provide a unifying Estimator interface, making it possible to write downstream infrastructure (e.g. distributed training, hyperparameter tuning) independent of the model implementation. We balance the competing demands for flexibility and simplicity by offering APIs at different levels of abstraction, making common model architectures available out of the box, while providing a library of utilities designed to speed up experimentation with model architectures. To make out of the box models flexible and usable across a wide range of problems, these canned Estimators are parameterized not only over traditional hyperparameters, but also using feature columns, a declarative specification describing how to interpret input data.

3.2 Survey-2

Title	**Predicting Stock Trends through Technical Analysis and Nearest Neighbor Classification**
Authors	Lamartine Almeida Teixeira , Adriano Lorena Inácio ,de Oliveira Department of Computing Systems University of Pernambuco Recife, Brazil
Year of Publication	February 2009
Summary	Tech Examination is built on the philosophies of the Dow Theory and practises the past of prices to forecast upcoming actions. The method used in tech examination can be enclosed as a outline credit problem, where the ideas are resulting from the history of values and the output is an estimate of the price or an estimate of the prices trend. The most significant evidence of this type of examination is that the marketplace action reductions everything. It means the specialist believes that anything that can perhaps affect the marketplace is already reflected in the prices, as well as that all the new evidence will be directly reflected in those prices. As a import, all the technician needs is to analyse the past of prices The main gears of the tech examination are the capacity and price charts. Based on the data of values and size the tech pointers are built. Tech pointers are math formulations that are applied to the price or volume statistics of a safekeeping for demonstrating some aspect of the association of those amounts.

3.3 Survey-3

Title	**Improving N Calculation of the RSI Financial Indicator Using Neural Networks**
Author	Alejandro Rodríguez-González, Fernando Guldris-Iglesias, Ricardo Colomo-Palacios Giner Alor-Hernandez, Ruben Posada-Gomez Computer Science Department Universidad Carlos III de Madrid Leganés, Spain
Year of Publication	January 2010
Publishing details	2010 International Journal on Recent and Innovation Trends in Computing and Communication, Volume 1.
Summary	There has been growing interest in Trading Decision Support Systems in recent years. In spite of its volatility, it is not entirely random, instead, it is nonlinear and dynamic or highly complicated and volatile. Stock movement is affected by the mixture of two types of factors: determinant (e.g. gradual strength change between buying side and selling side) and random (e.g. emergent affairs or daily operation variations). There are 3 modules that are talked about in this research paper. The Neural Network Module is the responsible of provide the N values that are used to calculate RSI and decide if an investor should invest in a certain company. Trading system Module analyzes the result given by neural network module. When a query is formulated to the system, it takes the actual values of the market and builds a query to the neural network. If RSI value is higher than 70 the decision that trading system return is a sell signal. If RSI value is lower than 30 the decision that trading system return is a buy signal. The heuristic module is in charge of managing the different formulas that provide the heuristic used to generate the optimal values for RSI indicator.

METHODOLOGY

4.1 System Design

Our proposed methodology based on the DNN learning architecture for classification where the classifier is identifying Risk Factor associated with equities

The proposed methodology for classifying the Risk in Stock Market is as follows:

4.1.1 Module Description

The project is basically divided into 2 modules. The Two modules are as follows

- Data Entry Module
- Analysis Module

The first module that is used to store the details like Previous Close ,Day Low, Day high, Previous Variation and Risk category in Train data and Test data.

In the second module we Train Data Using Tensorflow

4.1.2 Proposed algorithm

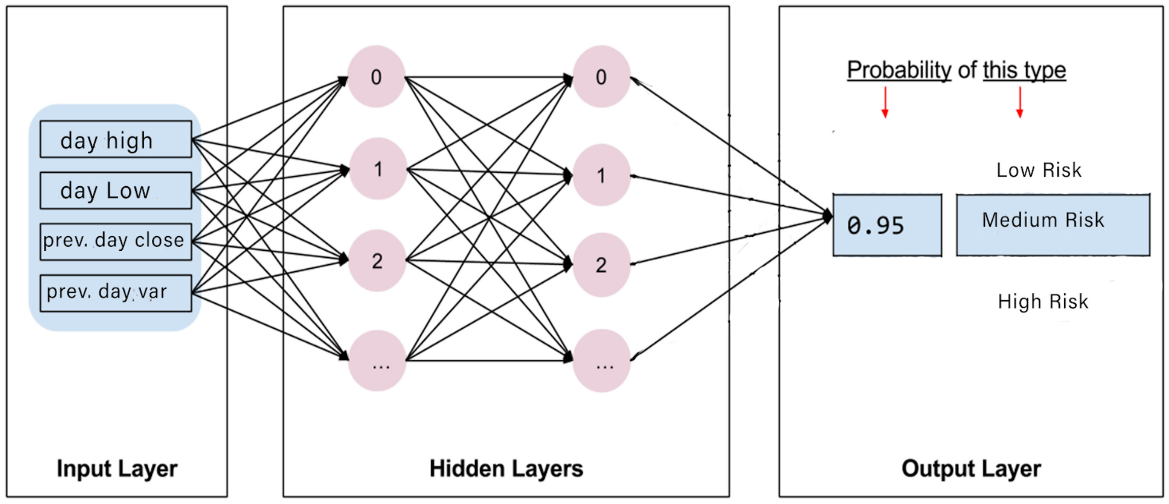


Fig 4.1: Illustration of DNN

Step 1: Get and store the following equities parameters in Test Database entity

Previous Day Close, Previous Day Variation, Day Low, Day High, Risk_Category

Step 2: Get and store the following equities parameters in Train Database entity

Previous Day Close, Previous Day Variation, Day Low, Day High, Risk_Category

Step 3: Load the train and test data into .csv files

Step 4: Provide the data files as input to the system

```
def train_input_fn(features, labels, batch_size):

    """An input function for training"""

    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))

    # Shuffle, repeat, and batch the examples.

    dataset = dataset.shuffle(1000).repeat().batch(batch_size)

    # Return the dataset.

    return dataset
```

This input function builds an input pipeline that yields batches of (features, labels) pairs, where features is a dictionary features.

```
# Instantiate a deep neural network classifier.
classifier = tf.estimator.DNNClassifier(
   feature_columns=feature_columns, # The input features to our model.
   hidden_units=[10, 10], # Two layers, each with 10 neurons.
   n_classes=3, # The number of output classes
   model_dir=PATH) # Pathname of directory where checkpoints, etc. are
stored.
```

The preceding code creates a deep neural network with the following characteristics:

- A list of feature columns. (The definitions of the feature columns are not shown in the preceding snippet.) For risk categories, the feature columns are numeric representations of four input features.
- Two fully connected layers, each having 10 neurons. A fully connected layer (also called a dense layer) is connected to every neuron in the subsequent layer.
- An output layer consisting of a one-element list.
- A directory (PATH) in which the trained model and various checkpoints will be stored.

Writing a model function

We are now ready to write the model_fn for our custom Estimator. Let's start with the function declaration:

```
def my_model_fn(
   features, # This is batch_features from input_fn
   labels,   # This is batch_labels from input_fn
   mode):    # Instance of tf.estimator.ModeKeys, see below
```

The first two arguments are the features and labels returned from the input function; that is, features and labels are the handles to the data our model will use. The mode argument indicates whether the caller is requesting training, predicting, or evaluating.

To implement a typical model function, we did following:

- Define the model's layers.
- Specify the model's behavior in three the different modes.

Define the model's layers

we defined the following three layers:

- an input layer
- one or more hidden layers
- an output layer

Use the Layers API (tf.layers) to define hidden and output layers.

Define the input layer

Call tf.feature_column.input_layer to define the input layer for a deep neural network. For example:# Create the layer of input

```
input_layer = tf.feature_column.input_layer(features, feature_columns)
```

The preceding line creates our input layer, reading our features through the input function and filtering them through the feature_columns defined earlier.

Establish Hidden Layers

we are creating a deep neural network, we must define one or more hidden layers. The Layers API provides a rich set of functions to define all types of hidden layers, including convolutional, pooling, and dropout layers. For Risk, we're simply going to call tf.layers.Dense twice to create two dense hidden layers, each with 10 neurons. By "dense," we mean that each neuron in the first hidden layer is connected to each neuron in the second hidden layer. Here's the relevant code:

```
# Definition of hidden layer: h1
# (Dense returns a Callable so we can provide input_layer as argument to it)
h1 = tf.layers.Dense(10, activation=tf.nn.relu)(input_layer)

# Definition of hidden layer: h2
# (Dense returns a Callable so we can provide h1 as argument to it)
h2 = tf.layers.Dense(10, activation=tf.nn.relu)(h1)
```

The inputs parameter to tf.layers.Dense identifies the preceding layer. The layer preceding h1 is the input layer.

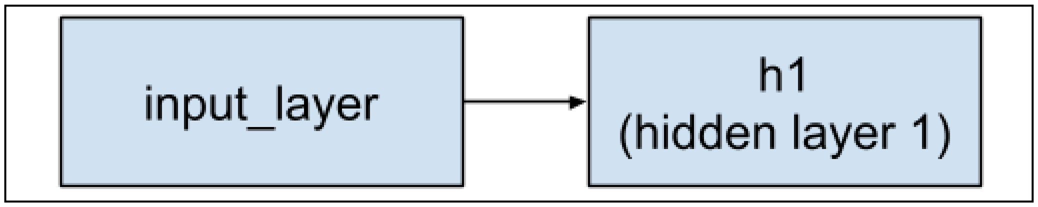


Fig 4.2: The input layer feeds into hidden layer 1.

The preceding layer to h2 is h1. So, the string of layers now looks like this:

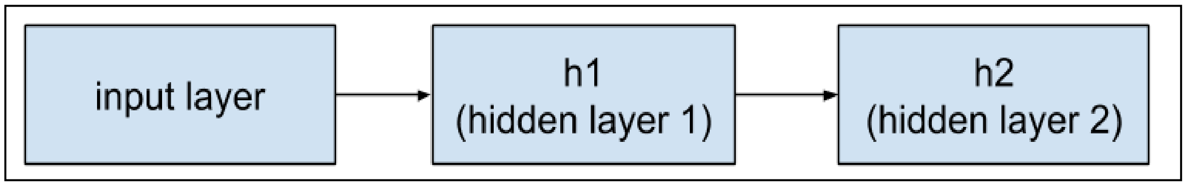


Fig 4.3: Hidden layer 1 feeds into hidden layer 2.

The first argument to tf.layers.Dense defines the number of its output neurons—10 in this case.

The activation parameter defines the activation function—Relu in this case.

Note that tf.layers.Dense provides many additional capabilities, including the ability to set a multitude of regularization parameters. For the sake of simplicity, though, we're going to simply accept the default values of the other parameters. Also, when looking at tf.layers may encounter lower-case versions (e.g. tf.layers.dense). As a general rule, we should use the class versions which start with a capital letter (tf.layers.Dense).

Output Layer

We'll define the output layer by calling tf.layers.Dense yet again:

```
# Output 'logits' layer is three numbers = probability distribution
# (Dense returns a Callable so we can provide h2 as argument to it)
logits = tf.layers.Dense(3)(h2)
```

Notice that the output layer receives its input from h2. Therefore, the full set of layers is now connected as follows:

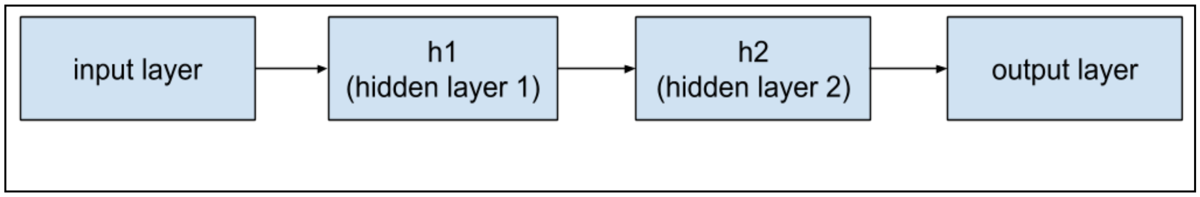


Fig 4.4: Hidden layer 2 feeds into the output layer.

When defining an output layer, the units parameter specifies the number of possible output values. So, by setting units to 3, the tf.layers.Dense function establishes a three-element logits vector. Each cell of the logits vector contains the probability of the Risk being low risk, Medium risk, or High risk respectively.

Since the output layer is a final layer, the call to tf.layers.Dense omits the optional activation parameter.

training, evaluation, and prediction

The final step in creating a model function is to write branching code that implements prediction, evaluation, and training.

The model function gets invoked whenever someone calls the Estimator's train, evaluate, or predict methods. Recall that the signature for the model function looks like this:

```
def my_model_fn(
   features, # This is batch_features from input_fn
   labels,   # This is batch_labels from input_fn
   mode):    # Instance of tf.estimator.ModeKeys, see below
```

Focus on that third argument, mode. As the following table shows, when someone calls train, evaluate, or predict, the Estimator framework invokes model function with the mod parameter set as follows:

Table 4: Values of mode.

Caller invokes custom Estimator method.	Estimator framework calls model function with the Mode parameter set to...
train()	ModeKeys.TRAIN
evaluate()	ModeKeys.EVAL
predict()	ModeKeys.PREDICT

For example, to instantiate a custom Estimator and generate an object named classifier. we made the following call

```
classifier.train(
  input_fn=lambda: my_input_fn(FILE_TRAIN, repeat_count=500, shuffle_count=256))
```

The Estimator framework then calls model function with mode set to ModeKeys.TRAIN. model function must provide code to handle all three of the mode values. For each mode value, code returns an instance of tf.estimator.EstimatorSpec, which contains the information the caller requires. Let's examine each mode.

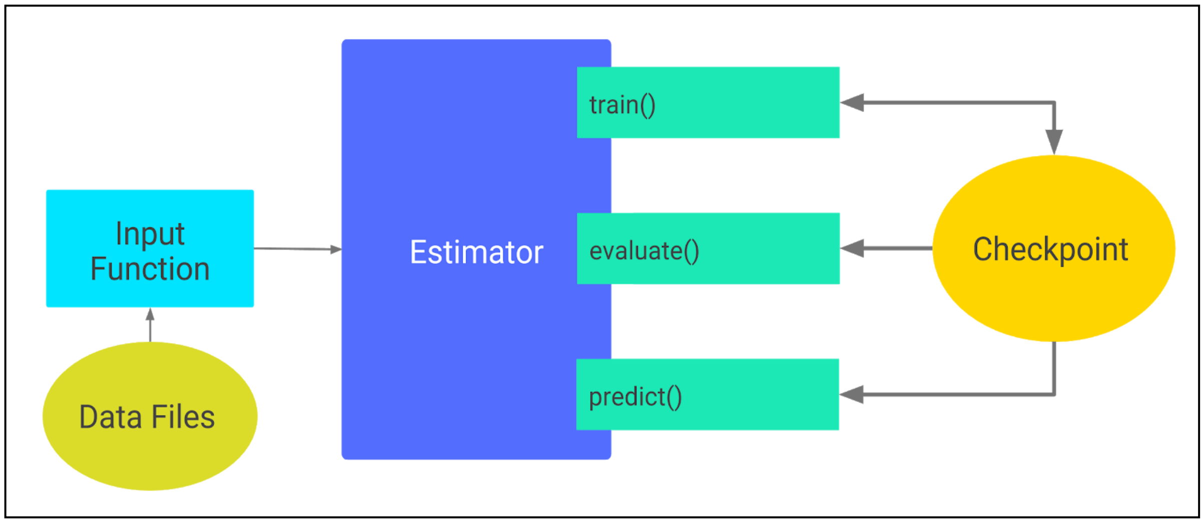


Fig 4.5: Subsequent calls to train(), evaluate(), or predict()

PREDICT

When model_fn is called with mode == ModeKeys.PREDICT, the model function must return a tf.estimator.EstimatorSpec containing the following information:

- the mode, which is tf.estimator.ModeKeys.PREDICT
- the prediction

The model must have been trained prior to making a prediction. The trained model is stored on disk in the directory established when we instantiated the Estimator.

For our case, the code to generate the prediction looks as follows:

```
# class_ids will be the model prediction for the class (Risk cat type)
# The output node with the highest value is our prediction
predictions = { 'class_ids': tf.argmax(input=logits, axis=1) }
```

```
# Return our prediction
if mode == tf.estimator.ModeKeys.PREDICT:
  return tf.estimator.EstimatorSpec(mode, predictions=predictions)
```

The block is surprisingly brief--the lines of code are simply the bucket at the end of a long hose that catches the falling predictions. After all, the Estimator has already done all the heavy lifting to make a prediction:

- The input function provides the model function with data (feature values) to infer from.
- The model function transforms those feature values into feature columns.
- The model function runs those feature columns through the previously-trained model.

The output layer is a logits vector that contains the value of each of the Risk Categories .. The tf.argmax method selects the Risk categories in that logits vector with the *highest* value. Notice that the highest value is assigned to a dictionary key named class_ids. We return that dictionary through the predictions parameter of tf.estimator.EstimatorSpec. The caller can then retrieve the prediction by examining the dictionary passed back to the Estimator's predict method.

EVAL

When model_fn is called with mode == ModeKeys.EVAL, the model function must evaluate the model, returning loss and possibly one or more metrics.

We can calculate loss by calling tf.losses.sparse_softmax_cross_entropy. Here's the complete code:

```
# To calculate the loss, we need to convert our labels
# Our input labels have shape: [batch_size, 1]
labels = tf.squeeze(labels, 1)          # Convert to shape [batch_size]
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)
```

Now let's turn our attention to metrics. Although returning metrics is optional, most custom Estimators return at least one metric. TensorFlow provides a Metrics API (tf.metrics) to calculate different kinds of metrics. For brevity's sake, we'll only return accuracy. The tf.metrics.accuracy compares our predictions against the "true labels", that is, against the labels provided by the input function. The tf.metrics.accuracy function requires the labels and predictions to have the same shape (which we did earlier). Here's the call to tf.metrics.accuracy:

```
# Calculate the accuracy between the true labels, and our predictions
accuracy = tf.metrics.accuracy(labels, predictions['class_ids'])
```

When the model is called with mode == ModeKeys.EVAL, the model function returns a tf.estimator.EstimatorSpec containing the following information:

- the mode, which is tf.estimator.ModeKeys.EVAL
- the model's loss
- typically, one or more metrics encased in a dictionary.

So, we'll create a dictionary containing our sole metric (my_accuracy). If we had calculated other metrics, we would have added them as additional key/value pairs to that same dictionary. Then, we'll pass that dictionary in the eval_metric_opsargument of tf.estimator.EstimatorSpec. Here's the block:

```
# Return our loss (which is used to evaluate our model)
# Set the TensorBoard scalar my_accurace to the accuracy
# Obs: This function only sets value during mode == ModeKeys.EVAL
# To set values during training, see tf.summary.scalar
if mode == tf.estimator.ModeKeys.EVAL:
        return tf.estimator.EstimatorSpec(mode, loss=loss,eval_metric_ops={'my_accuracy': accuracy})
```

TRAIN

When model_fn is called with mode == ModeKeys.TRAIN, the model function must train the model.

We must first instantiate an optimizer object. We picked Adagrad (tf.train.AdagradOptimizer) in the following code block only because we're mimicking the DNNClassifier, which also uses Adagrad. The tf.train package provides many other optimizers—feel free to experiment with them.

Next, we train the model by establishing an objective on the optimizer, which is simply to minimize its loss. To establish that objective, we call the minimize method.

In the code below, the optional global_step argument specifies the variable that TensorFlow uses to count the number of batches that have been processed. Setting global_step to tf.train.get_global_step will work beautifully. Also, we are calling tf.summary.scalar to report my_accuracy to TensorBoard during training. For both of these notes, please see the section on TensorBoard below for further explanation.

```
optimizer = tf.train.AdagradOptimizer(0.05)
train_op = optimizer.minimize( loss, global_step=tf.train.get_global_step())


# Set the TensorBoard scalar my_accuracy to the accuracy
tf.summary.scalar('my_accuracy', accuracy[1])
```

When the model is called with mode == ModeKeys.TRAIN, the model function must return a tf.estimator.EstimatorSpec containing the following information:

- the mode, which is tf.estimator.ModeKeys.TRAIN
- the loss
- the result of the training op

```
# Return training operations: loss and train_op
return tf.estimator.EstimatorSpec(
   mode,
   loss=loss,
   train_op=train_op)
```

Our model function is now complete!

The Custom Estimator

After creating new custom Estimator. Start by instantiating the custom Estimator through the Estimator base class as follows:

```
classifier = tf.estimator.Estimator(
   model_fn=my_model_fn,
   model_dir=PATH)  # Path to where checkpoints etc are stored
classifier.train(
  input_fn=lambda: my_input_fn(FILE_TRAIN, repeat_count=500, shuffle_count=256))
```

TensorBoard

we can view some training results in TensorBoard. To see this reporting, start TensorBoard from command-line as follows:

```
# Replace PATH with the actual path passed as model_dir
tensorboard --logdir=PATH
```

Then browse to the following URL:

localhost:6006

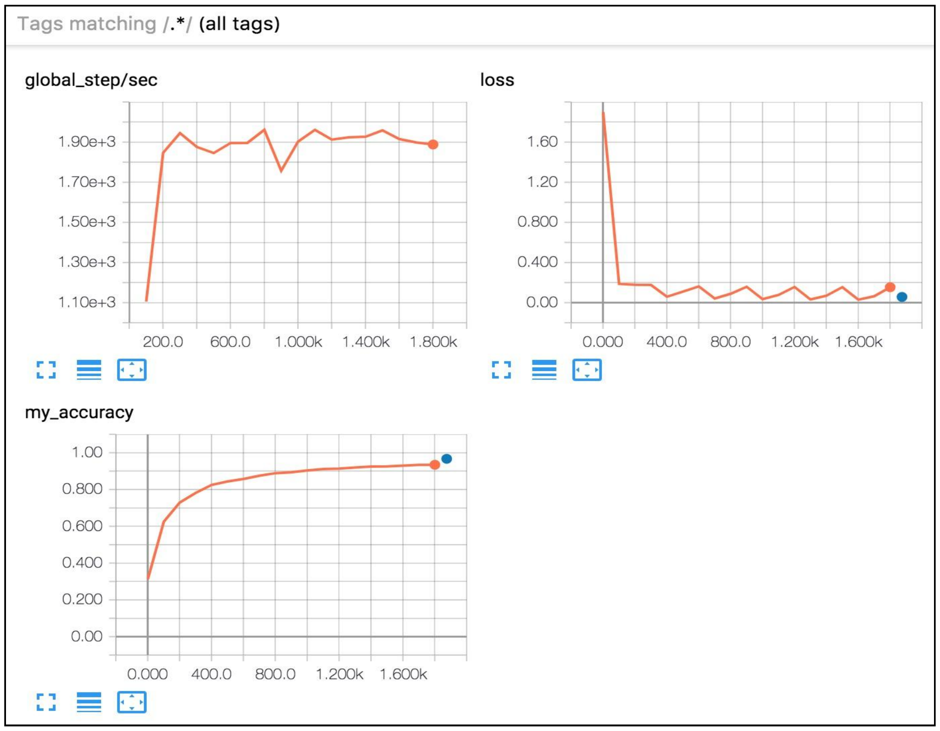


Fig 4.6: TensorBoard displays three graphs.

In brief, here's what the three graphs tell :

- **global_step/sec**: A performance indicator, showing how many batches (gradient updates) we processed per second (y-axis) at a particular batch (x-axis). In order to see this report, we provide a global_step(as we did with tf.train.get_global_step()). we also run training for a sufficiently long time, which we do by asking the Estimator train for 500 epochs when we call its train method.
- **loss**: The loss reported. The actual loss value (y-axis) doesn't mean much. The shape of the graph is what's important.
- **my_accuracy**: The accuracy recorded when we invoked both of the following.
- eval_metric_ops={**'my_accuracy'**: accuracy}), during EVAL(when returning our EstimatorSpec)
- tf.summary.scalar(**'my_accuracy'**, accuracy[1]), during TRAIN

Note the following in the my_accuracy and loss graphs:

- The orange line represents TRAIN.
- The blue dot represents EVAL.

During TRAIN, orange values are recorded continuously as batches are processed, which is why it becomes a graph spanning x-axis range. By contrast, EVAL produces only a single value from processing all the evaluation steps.As suggested in Figure, we can selectively disable/enable the reporting for training and evaluation the left side.

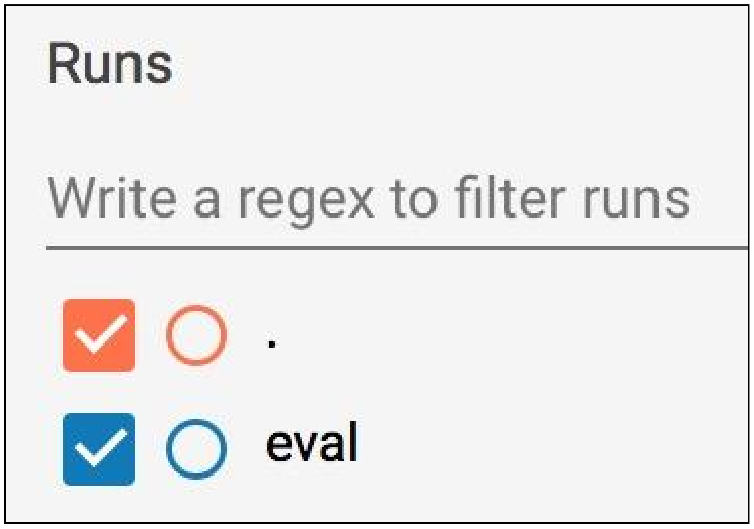


Fig 4.7: Enable or disable reporting.

In order to see the orange graph, we specify a global step. This, in combination with getting global_steps/sec reported, makes it a best practice to always register a global step by passing tf.train.get_global_step() as an argument to the optimizer.minimize call.

4.1.3 Diagrammatic representation

1.Use Case Diagram

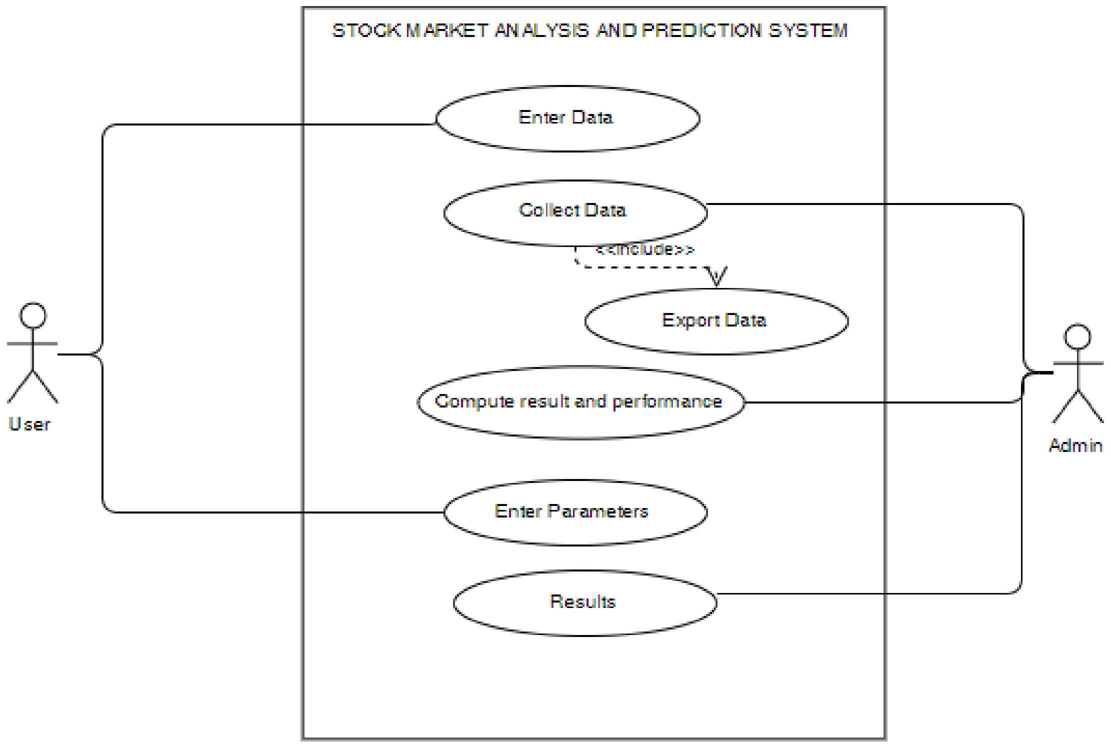


Fig 4.8: Use case diagram for stock market risk prediction system

1. Data initially Entered by User using data entry module and stored into database.
2. Admin will collect data from database and export into .csv files
3. Data is used to train the system.
4. Train model is saved.
5. User will enter parameters.
6. Prediction will be done and Results shown.

2.Sequence Diagram

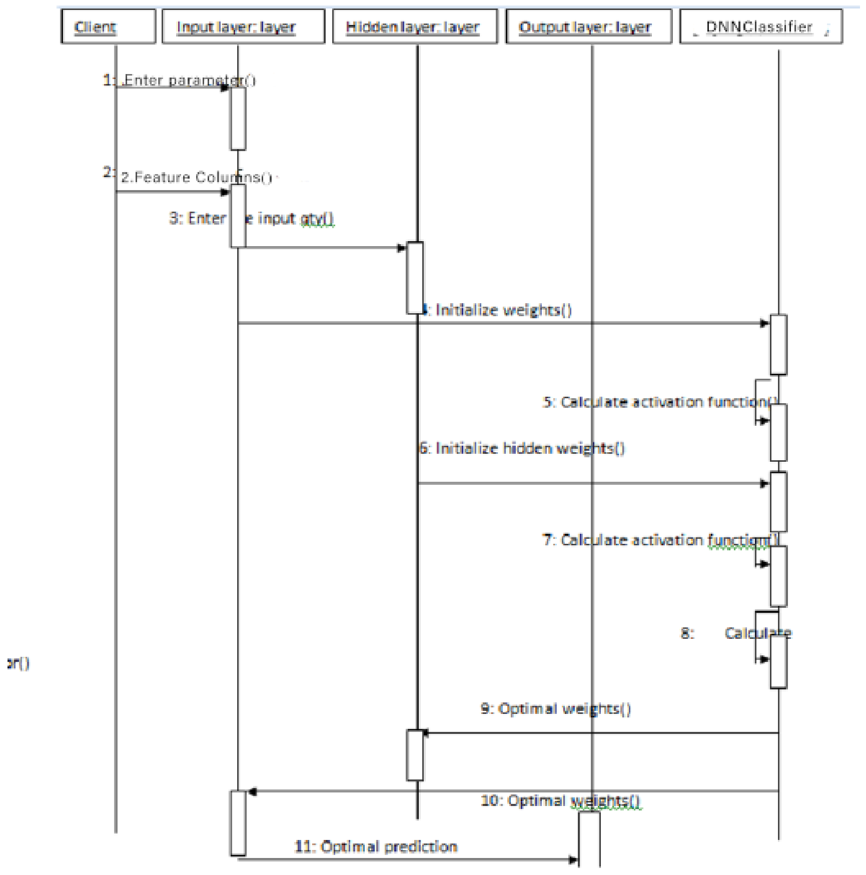


Fig 4.9: Sequence diagram for Stock market Risk prediction system

1. Input layer receives parameter entered by Client.
2. Sends input parameters to hidden layers.
3. Model will receive parameters and weights from input layer and hidden layer.
4. Performs calculations.
5. In the end optimal prediction Displayed.

4.2 Implementation of Proposed System

Database Creation using SQlite3

Now open Command Prompt and go to sqlite path folder by entering following command

```
   cd sqlite
```

Enter following commands to Create database

```
sqlite3 eqre1.db
```

Now Database is Created. For creating Tables in Database enter following Commands.

```
CREATE TABLE testdata(prevclose varchar(10),
                       daylow varchar(10),
                       dayhigh varchar(10),
                       prevvar varchar(10),
                       riskcat varchar(1)  );

CREATE TABLE traindata(prevclose varchar(10),
                       daylow varchar(10),
                       dayhigh varchar(10),
                       prevvar varchar(10),
                       riskcat varchar(1)  );

CREATE TABLE ctestdata(prevclose varchar(10),
                       daylow varchar(10),
                       dayhigh varchar(10),
                       prevvar varchar(10),
                       riskcat varchar(1)   );
```

Creation of User Interface

Start the PyQt5 Designer tool

Go to C:\Program Files (x86)\Python36-32\Lib\site-packages\pyqt5-tools and locate designer.exe. Double click to open the Qt Designer.

The Qt Designer will provide some basic templates. Select the template "Dialog with Buttons Right" as shown in the screenshot below.

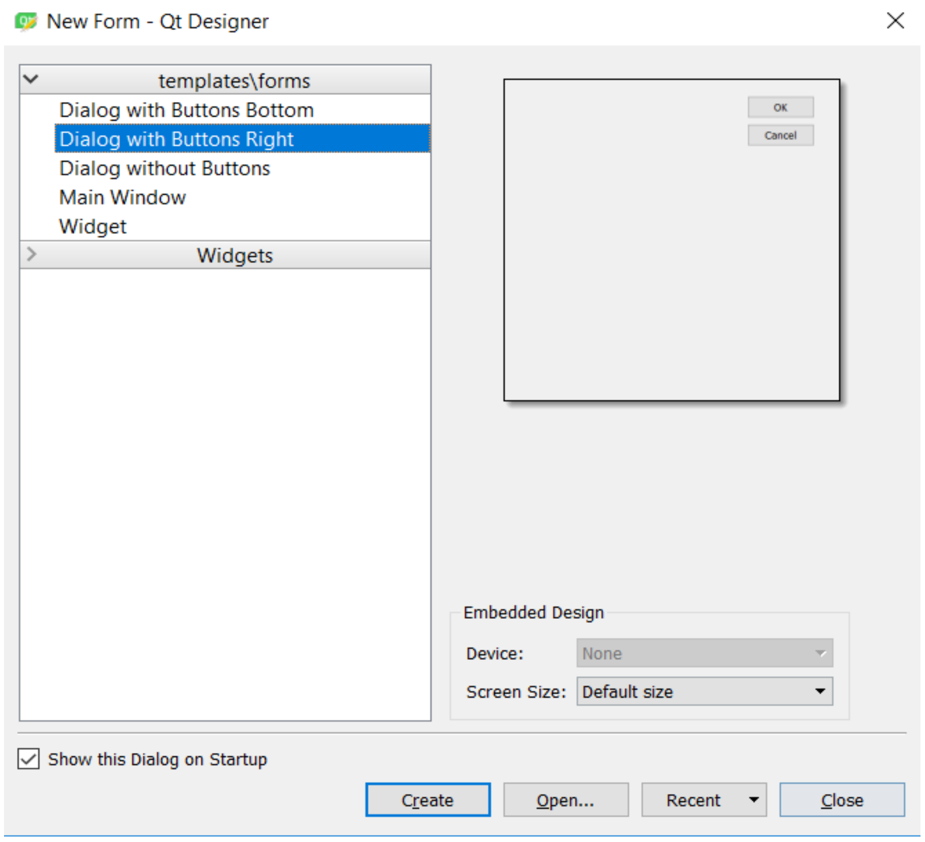


Fig 4.10: PyQt5 selection Template

We can change the dialog properties using the Property Editor.

Now, save the designed dialog as dialog.ui.

Converting dialog.ui to dialog.py

Use the command below on the command prompt.

```
pyuic5 dialog.ui > dialog.py
```

The above command will convert the dialog.ui file to dialog.py.

Next, we'll make some minor changes and execute the Python code.

This is the final code for the Dialog class generated from the dialog.ui.

The Dialog will look like this.

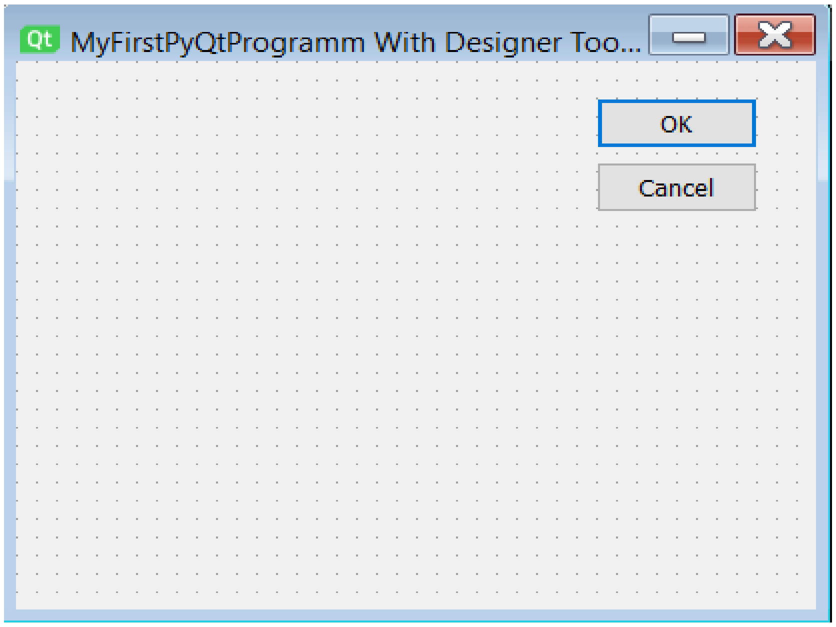


Fig 4.11: **Converting dialog.ui to dialog.py**

Creation Of Training and Testing Data

Enter Prev.Close ,Day low,Day High, Prev.Var and Risk category.

Previous Close:

Previous close is a security's closing price on the preceding day of trading. previous close can refer to the prior day's value of a stock,bond,commodity, futures or option contract, market index or any other security.

Day Low:

Day low is a security’s intraday low trading price.today’s low is the lowest price at which a stock trades over the course of a trading day.

Day High:

Day High refers to a security’s intraday high trading price.Day high is the highest price at which a stock traded during the course of the day.Day high is typically higher than the closing or opening price.More often than not this is higher than the closing price.

Previous Variation:

The daily price variation of a stock is the difference between its highest and lowest values on a given trading day. Daily price variation may also refer to the difference between one day's opening price and the next day's opening price. Daily price variation is a measure of volatility, or how much a stock's value changes. Although it is a daily measurement, average daily variations can be calculated by adding up individual daily price variations and dividing the total by the number of days to spot a more long-term trend.

Risk category:

Risk category is based on user prospective. In our project we are considering Low Risk as 0,Medium Risk as 1 and High Risk as 2.

Training Data:

Training Data also called training set,training dataset or learning set is the one on which we train and fit our model basically to fit the parameters.The training data includes both input data and the corresponding expected output.Based on this Ground truth data.the algorithm can learn how to apply technologies such as neural networks to learn and produce complex results. In Our project we are considering first four months data as Training Data.

Testing Data:

Testing Data includes only input data not the corresponding expected data,test data is used only to assess performance of model.

Enter this command in Command Prompt

```
python equitiesentry1.py
```

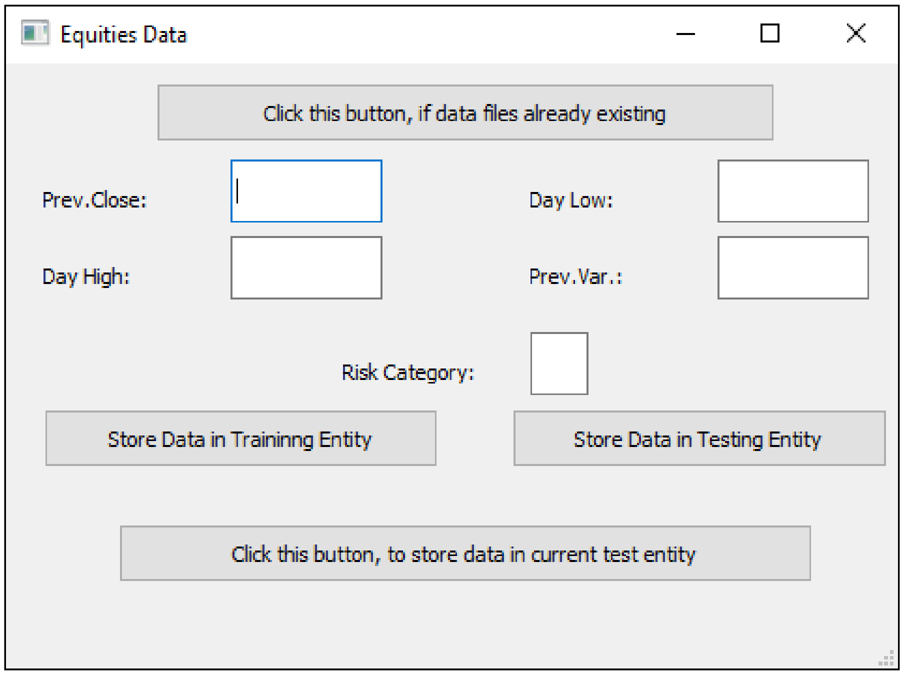
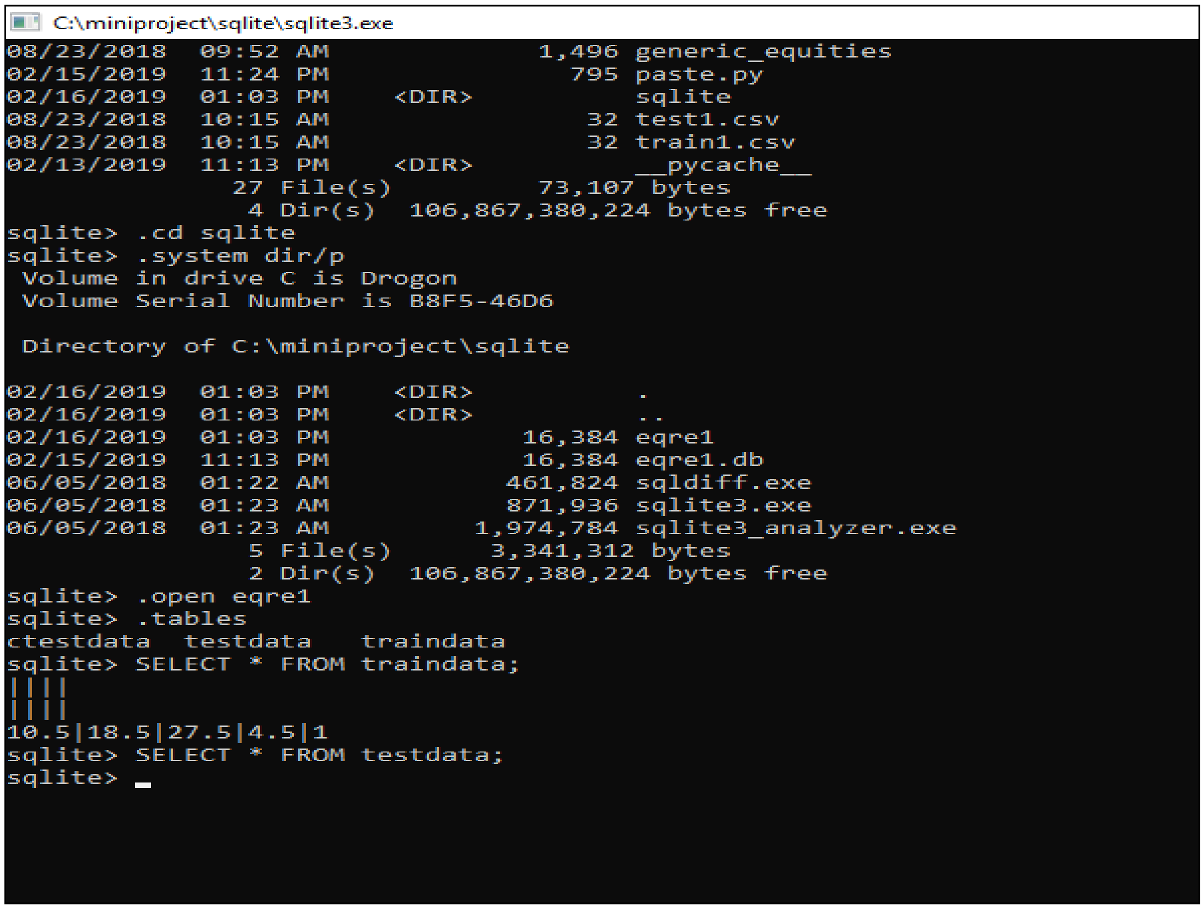

Equities Data
Click this button, if data files already existing
Prev.Close:
Day Low:
Day High:
Prev.Var.:
Risk Category:
Store Data in Traininng Entity
Store Data in Testing Entity
Click this button, to store data in current test entity

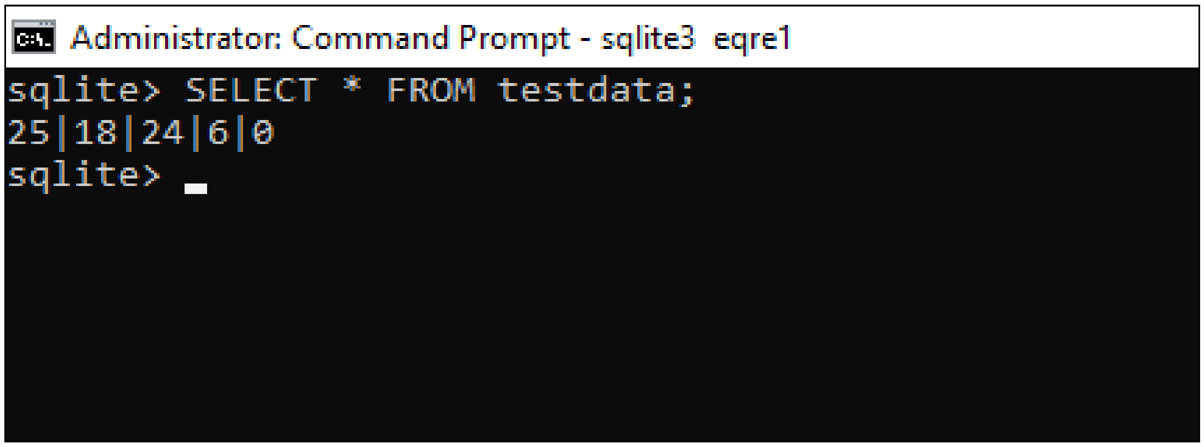
Fig 4.12: Data entry module

Store data in different Entities based on Requirements Try to maintain 80-20 or 60-40 ratio in Training and Testing Dataset.



```
C:\miniproject\sqlite\sqlite3.exe
08/23/2018  09:52 AM             1,496 generic_equities
02/15/2019  11:24 PM               795 paste.py
02/16/2019  01:03 PM    <DIR>          sqlite
08/23/2018  10:15 AM                32 test1.csv
08/23/2018  10:15 AM                32 train1.csv
02/13/2019  11:13 PM    <DIR>          __pycache__
              27 File(s)         73,107 bytes
               4 Dir(s)  106,867,380,224 bytes free
sqlite> .cd sqlite
sqlite> .system dir/p
 Volume in drive C is Drogon
 Volume Serial Number is B8F5-46D6

 Directory of C:\miniproject\sqlite

02/16/2019  01:03 PM    <DIR>          .
02/16/2019  01:03 PM    <DIR>          ..
02/16/2019  01:03 PM            16,384 eqre1
02/15/2019  11:13 PM            16,384 eqre1.db
06/05/2018  01:22 AM           461,824 sqldiff.exe
06/05/2018  01:23 AM           871,936 sqlite3.exe
06/05/2018  01:23 AM         1,974,784 sqlite3_analyzer.exe
               5 File(s)      3,341,312 bytes
               2 Dir(s)  106,867,380,224 bytes free
sqlite> .open eqre1
sqlite> .tables
ctestdata  testdata   traindata
sqlite> SELECT * FROM traindata;
||||
||||
10.5|18.5|27.5|4.5|1
sqlite> SELECT * FROM testdata;
sqlite> _
```

Fig 4.13: Data stored in Training Entity



```
Administrator: Command Prompt - sqlite3  eqre1
sqlite> SELECT * FROM testdata;
25|18|24|6|0
sqlite> _
```

Fig 4.14: Data stored in Testing Entity

To enter multiple data simultaneously

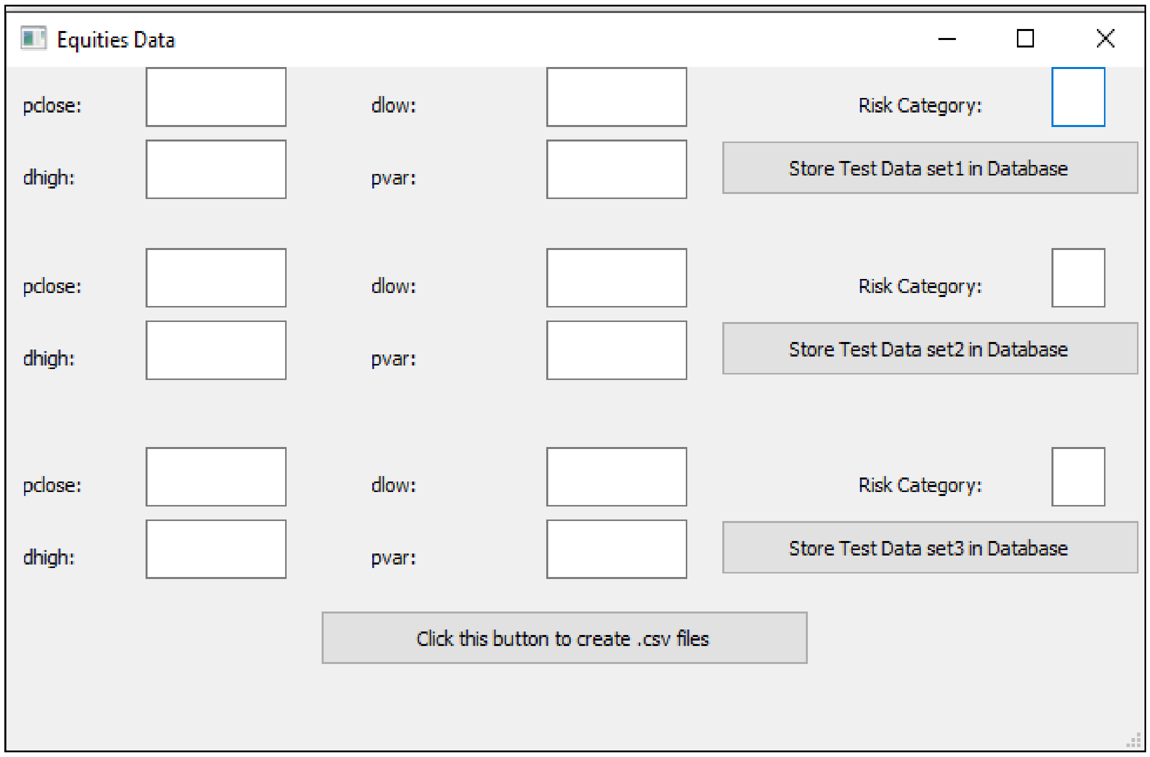

Equities Data
pclose:
dlow:
Risk Category:
dhigh:
pvar:
Store Test Data set1 in Database
pclose:
dlow:
Risk Category:
dhigh:
pvar:
Store Test Data set2 in Database
pclose:
dlow:
Risk Category:
dhigh:
pvar:
Store Test Data set3 in Database
Click this button to create .csv files

Fig 4.15: Entering multiple Data simultaneously

After Creating Entities Export entity data into .csv files by using these buttons.

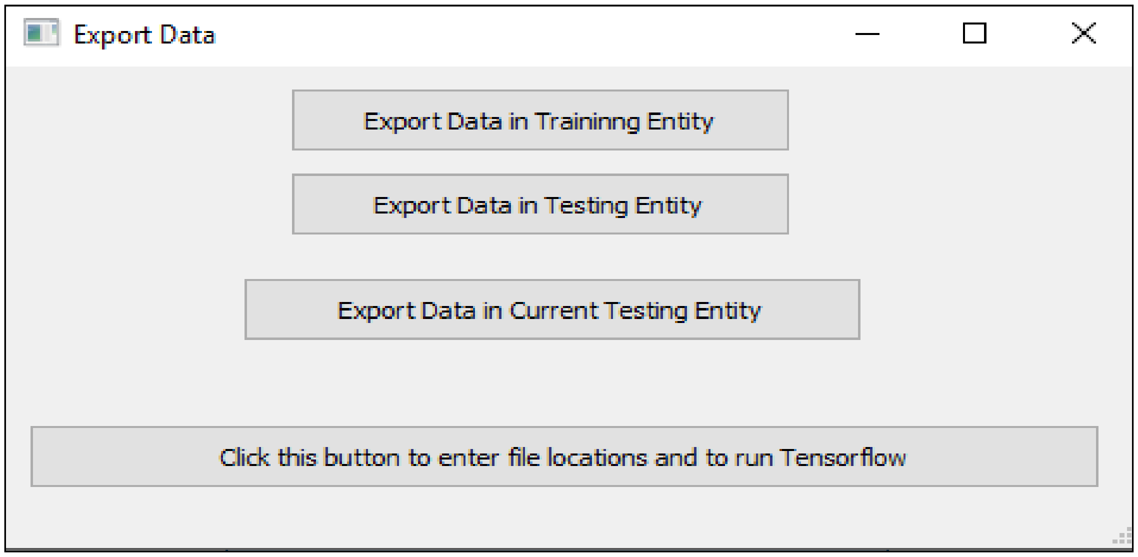

Export Data
Export Data in Traininng Entity
Export Data in Testing Entity
Export Data in Current Testing Entity
Click this button to enter file locations and to run Tensorflow

Fig 4.16: Exporting the Data

Tensorflow Execution

Activate Tensorflow by using following command before starting Tensorflow execution .

```
activate Tensorflow
```

In this module, enter the File paths of training file , testing file ,current test file.

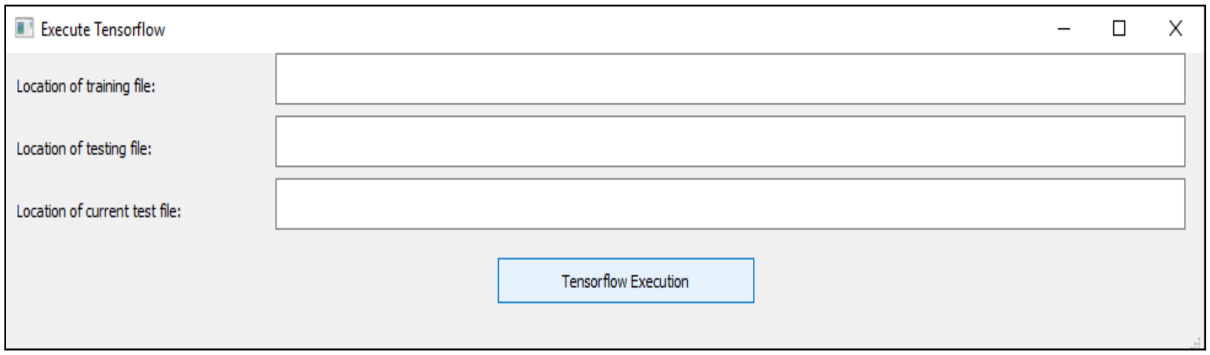
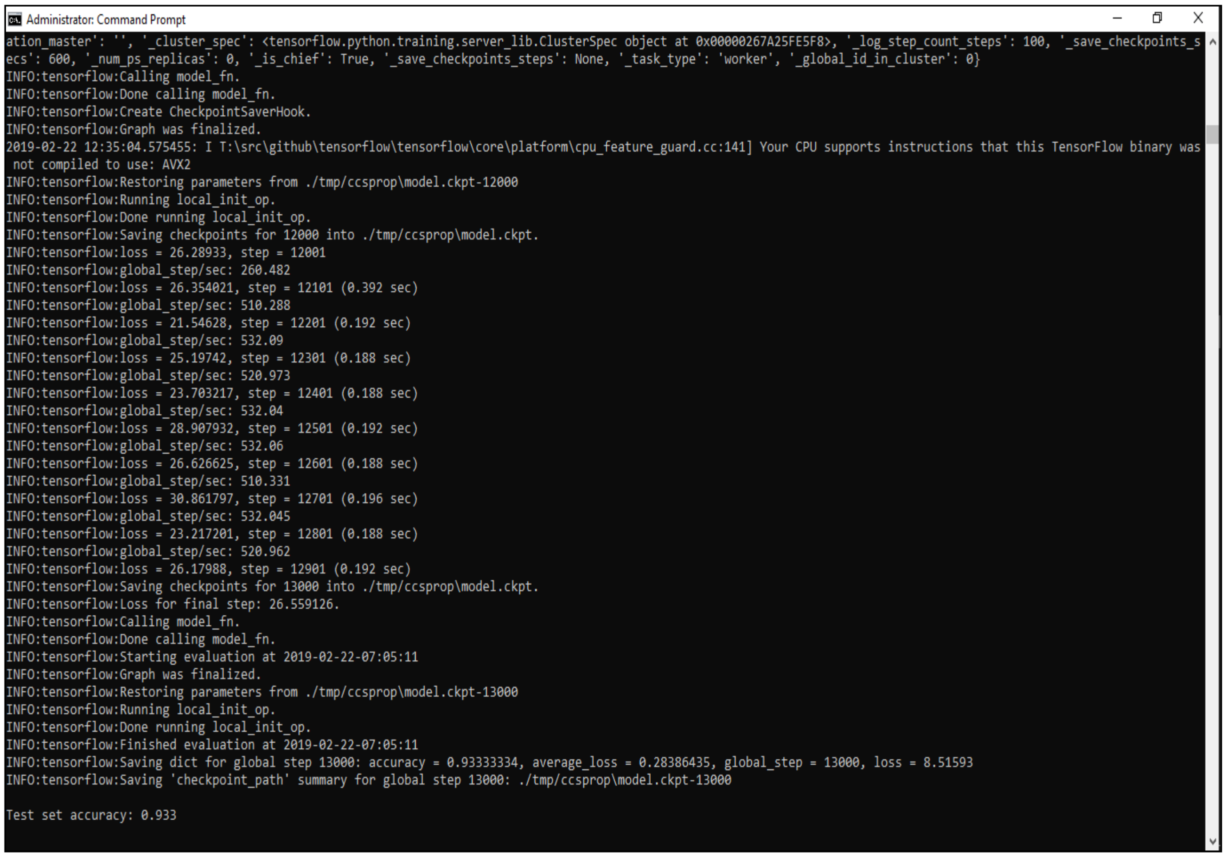

Execute Tensorflow
Location of training file:
Location of testing file:
Location of current test file:
Tensorflow Execution

Fig 4.17: Tensorflow Execution



```
ation_master': '', '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x00000267A25FE5F8>, '_log_step_count_steps': 100, '_save_checkpoints_s
ecs': 600, '_num_ps_replicas': 0, '_is_chief': True, '_save_checkpoints_steps': None, '_task_type': 'worker', '_global_id_in_cluster': 0}
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
2019-02-22 12:35:04.575455: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was
 not compiled to use: AVX2
INFO:tensorflow:Restoring parameters from ./tmp/ccsprop\model.ckpt-12000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 12000 into ./tmp/ccsprop\model.ckpt.
INFO:tensorflow:loss = 26.28933, step = 12001
INFO:tensorflow:global_step/sec: 260.482
INFO:tensorflow:loss = 26.354021, step = 12101 (0.392 sec)
INFO:tensorflow:global_step/sec: 510.288
INFO:tensorflow:loss = 21.54628, step = 12201 (0.192 sec)
INFO:tensorflow:global_step/sec: 532.09
INFO:tensorflow:loss = 25.19742, step = 12301 (0.188 sec)
INFO:tensorflow:global_step/sec: 520.973
INFO:tensorflow:loss = 23.703217, step = 12401 (0.188 sec)
INFO:tensorflow:global_step/sec: 532.04
INFO:tensorflow:loss = 28.907932, step = 12501 (0.192 sec)
INFO:tensorflow:global_step/sec: 532.06
INFO:tensorflow:loss = 26.626625, step = 12601 (0.188 sec)
INFO:tensorflow:global_step/sec: 510.331
INFO:tensorflow:loss = 30.861797, step = 12701 (0.196 sec)
INFO:tensorflow:global_step/sec: 532.045
INFO:tensorflow:loss = 23.217201, step = 12801 (0.188 sec)
INFO:tensorflow:global_step/sec: 520.962
INFO:tensorflow:loss = 26.17988, step = 12901 (0.192 sec)
INFO:tensorflow:Saving checkpoints for 13000 into ./tmp/ccsprop\model.ckpt.
INFO:tensorflow:Loss for final step: 26.559126.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2019-02-22-07:05:11
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./tmp/ccsprop\model.ckpt-13000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2019-02-22-07:05:11
INFO:tensorflow:Saving dict for global step 13000: accuracy = 0.93333334, average_loss = 0.28386435, global_step = 13000, loss = 8.51593
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 13000: ./tmp/ccsprop\model.ckpt-13000

Test set accuracy: 0.933
```

Fig 4.18: Training of the data

4.3 Sample code

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import tensorflow as tf

import equities_data

parser = argparse.ArgumentParser()
parser.add_argument('--batch_size', default=100, type=int, help='batch size')
parser.add_argument('--train_steps', default=1000, type=int,
                    help='number of training steps')

def my_model(features, labels, mode, params):
    """DNN with three hidden layers and learning_rate=0.1."""
    # Create three fully connected layers.
    net = tf.feature_column.input_layer(features, params['feature_columns'])
    for units in params['hidden_units']:
        net = tf.layers.dense(net, units=units, activation=tf.nn.relu)

    # Compute logits (1 per class).
    logits = tf.layers.dense(net, params['n_classes'], activation=None)

    # Compute predictions.
    predicted_classes = tf.argmax(logits, 1)
    if mode == tf.estimator.ModeKeys.PREDICT:
```

```
    predictions = {
        'class_ids': predicted_classes[:, tf.newaxis],
        'probabilities': tf.nn.softmax(logits),
        'logits': logits,
    }
    return tf.estimator.EstimatorSpec(mode, predictions=predictions)

# Compute loss.
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)

# Compute evaluation metrics.
accuracy = tf.metrics.accuracy(labels=labels,
                               predictions=predicted_classes,
                               name='acc_op')
metrics = {'accuracy': accuracy}
tf.summary.scalar('accuracy', accuracy[1])

if mode == tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(
        mode, loss=loss, eval_metric_ops=metrics)

# Create training op.
assert mode == tf.estimator.ModeKeys.TRAIN

optimizer = tf.train.AdagradOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
```

```
def main(argv):
    args = parser.parse_args(argv[1:])

    # Fetch the data
    (train_x, train_y), (test_x, test_y) = equities_data.load_data()

    # Feature columns describe how to use the input.
    my_feature_columns = []
    for key in train_x.keys():
        my_feature_columns.append(tf.feature_column.numeric_column(key=key))

    # Build 2 hidden layer DNN with 10, 10 units respectively.
    classifier = tf.estimator.Estimator(
        model_fn=my_model,
        params={
            'feature_columns': my_feature_columns,
            # Two hidden layers of 10 nodes each.
            'hidden_units': [10, 10],
            # The model must choose between 3 classes.
            'n_classes': 3,
        })

    # Train the Model.
    classifier.train(
        input_fn=lambda:equities_data.train_input_fn(train_x, train_y, args.batch_size),
        steps=args.train_steps)
```

```
# Evaluate the model.
    eval_result = classifier.evaluate(
        input_fn=lambda:equities_data.eval_input_fn(test_x, test_y, args.batch_size))

    print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))

    # Generate predictions from the model
    expected = [' ']
    predict_x = dict()

    value = float(input("Enter the value of 'dayhigh': "))

    predict_x['dayhigh'] = [value]

    value = float(input("Enter the value of 'daylow': "))

    predict_x['daylow'] = [value]

    value = float(input("Enter the value of 'pdclose': "))

    predict_x['pdclose'] = [value]

    value = float(input("Enter the value of 'pdvar': "))

    predict_x['pdvar'] = [value]

    predictions = classifier.predict(
        input_fn=lambda:equities_data.eval_input_fn(predict_x, labels=None,
                                     batch_size=args.batch_size))
    for pred_dict, expec in zip(predictions, expected):
        template = ('\nPrediction is "{}" ({:.1f}%),')

        class_id = pred_dict['class_ids'][0]
        probability = pred_dict['probabilities'][class_id]

        print(template.format(equities_data.RISKCAT[class_id],
                              100 * probability, expec))


if __name__ == '__main__':
    tf.logging.set_verbosity(tf.logging.INFO)
    tf.app.run(main)
```

5.RESULTS AND TESTING

5.1 System Testing

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

Test Objectives

1. All field entries must work properly.
2. The entry screen ,messages and responses must not be delayed.

Features To Be Tested

1. Verify that the entries are of the correct format.
2. Verify whether data stored in correct entity.

Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, user manuals.

Functional testing is centered on the following items:

Valid Input: identified classes of valid input must be accepted.

Invalid Input: identified classes of invalid input must be rejected.

Functions: identified functions must be exercised.

Output: identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Output 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.

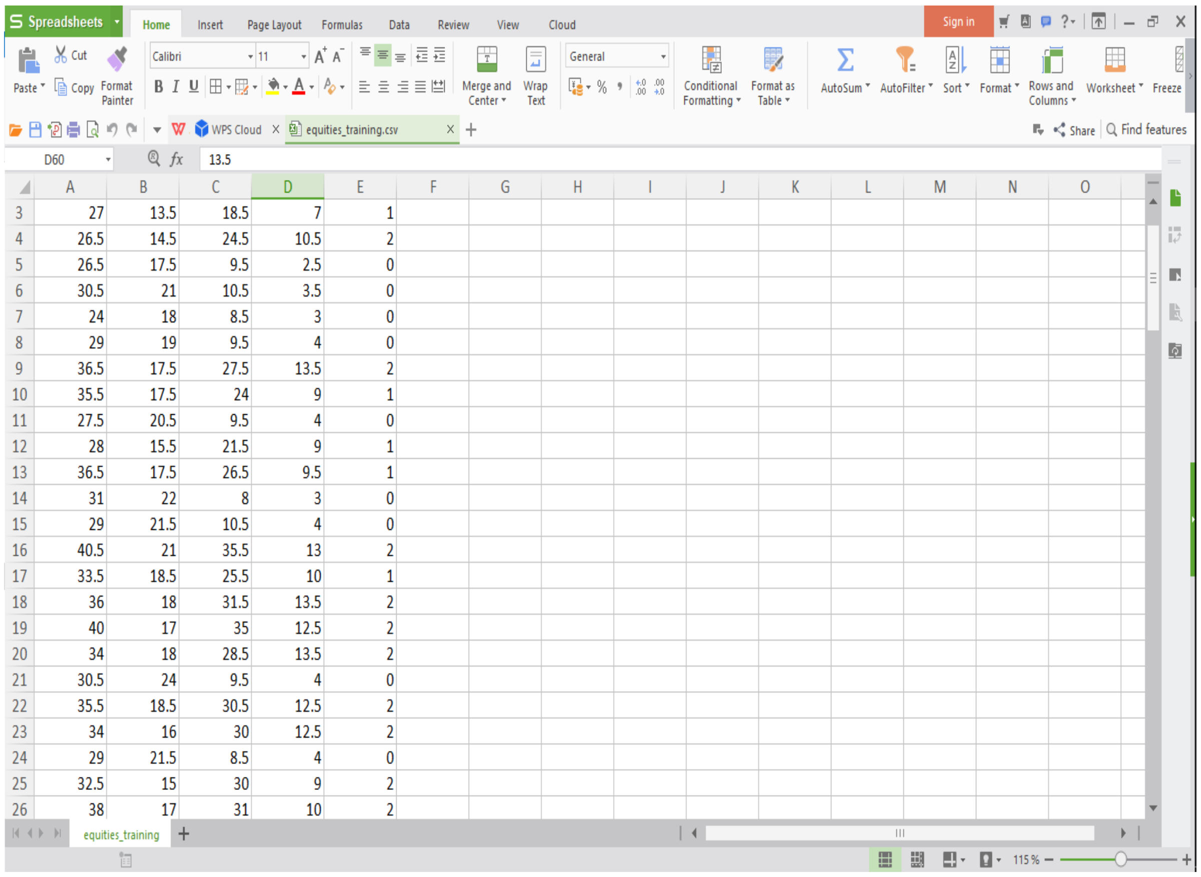
5.2 Test Cases

A Test case is a set of input data and expected results that exercises a component with the purpose of causing failure and detecting faults. Test case is an explicit set of instructions designed to detect a particular class of defect in a software system, by bringing about a failure. A Test case can give rise to many tests.

Test cases can be divided into two types. First one is positive test cases and second one is negative test cases. In positive test cases are conducted by the developer intention is to get the output. In negative test cases are conducted by the developer intention is to don't get the output.

Table 5: Test Cases

Test case ID	Test case Name	Input	Expected Result	Actual Result	Remarks
1	Enter Test Data	Previous close:32, Previous .variation:6, Day .low:29, Day .high:35 Risk category:2	Data stored into testdata file	Data stored into testdata file	Success
2	Enter Data Before creation of database	previou.c:42, Previous variation:40, Day low:39, Day high:45 Risk category:2	Table is not created. Terminate entry.	no such table: Traindata	Success
3	During Tensorflow Execution file Path not provided	Path empty	Execution terminated	Execution terminated	Success
4	Enter current test entity	p.c:24.2,p.v:4,d.l:29, d.h:35 r:2	Data saved into file	Data saved successfully	Success
5	Provide test,train,current data file	/testdata /traindata /currenttestdata	Tensorflow execution with predictions	Medium risk:96.6	Success

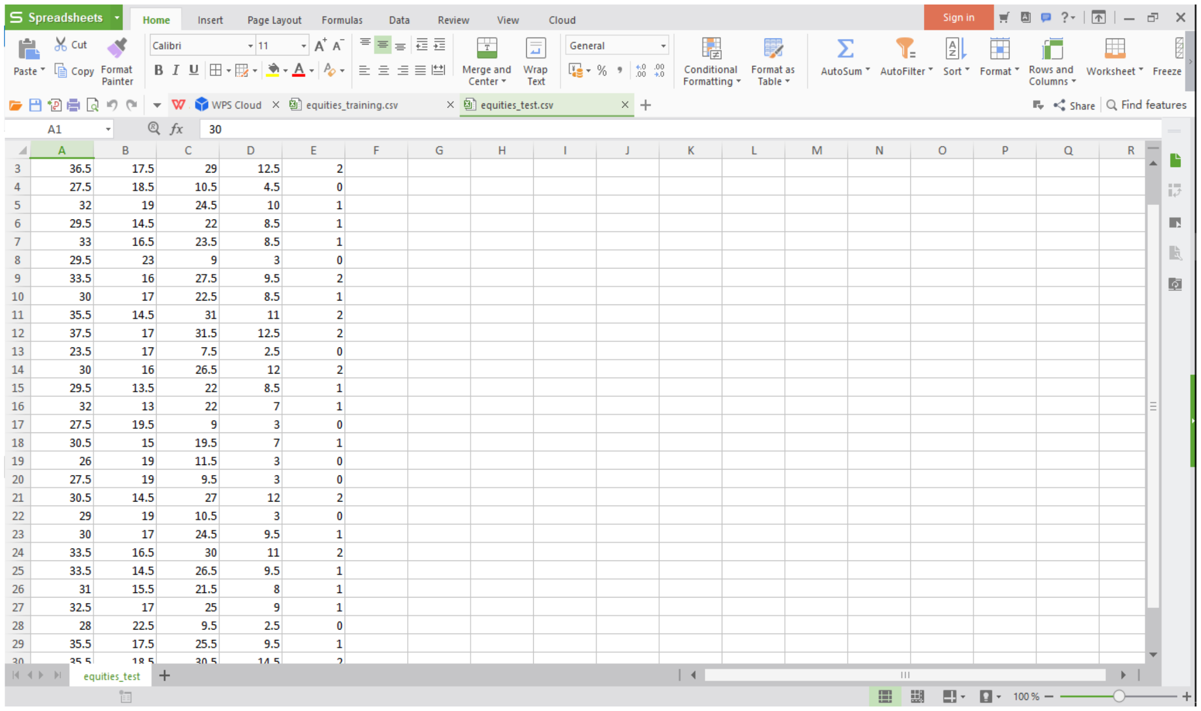


D60 13.5

	A	B	C	D	E
3	27	13.5	18.5	7	1
4	26.5	14.5	24.5	10.5	2
5	26.5	17.5	9.5	2.5	0
6	30.5	21	10.5	3.5	0
7	24	18	8.5	3	0
8	29	19	9.5	4	0
9	36.5	17.5	27.5	13.5	2
10	35.5	17.5	24	9	1
11	27.5	20.5	9.5	4	0
12	28	15.5	21.5	9	1
13	36.5	17.5	26.5	9.5	1
14	31	22	8	3	0
15	29	21.5	10.5	4	0
16	40.5	21	35.5	13	2
17	33.5	18.5	25.5	10	1
18	36	18	31.5	13.5	2
19	40	17	35	12.5	2
20	34	18	28.5	13.5	2
21	30.5	24	9.5	4	0
22	35.5	18.5	30.5	12.5	2
23	34	16	30	12.5	2
24	29	21.5	8.5	4	0
25	32.5	15	30	9	2
26	38	17	31	10	2

Fig 5.1: Training Data Set

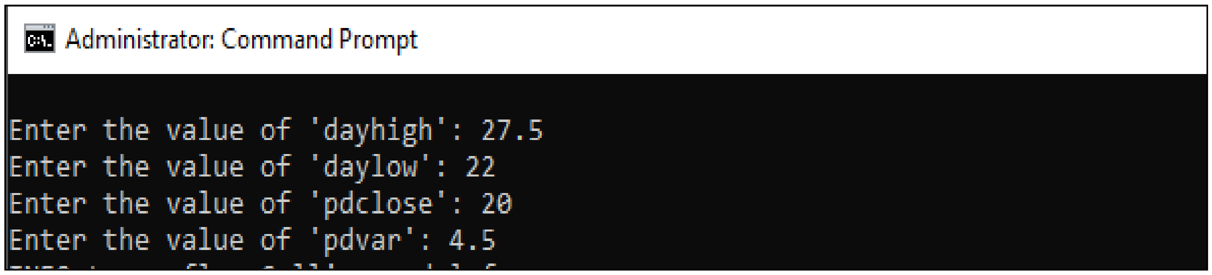
The above figure represents the training set.The data set contains values of High risk , low risk, Medium risk,previous day close,previous day variation.It represents the data of last six months.The class label 0 represents low risk,1 represent medium risk,2 represents high risk



	A	B	C	D	E
3	36.5	17.5	29	12.5	2
4	27.5	18.5	10.5	4.5	0
5	32	19	24.5	10	1
6	29.5	14.5	22	8.5	1
7	33	16.5	23.5	8.5	1
8	29.5	23	9	3	0
9	33.5	16	27.5	9.5	2
10	30	17	22.5	8.5	1
11	35.5	14.5	31	11	2
12	37.5	17	31.5	12.5	2
13	23.5	17	7.5	2.5	0
14	30	16	26.5	12	2
15	29.5	13.5	22	8.5	1
16	32	13	22	7	1
17	27.5	19.5	9	3	0
18	30.5	15	19.5	7	1
19	26	19	11.5	3	0
20	27.5	19	9.5	3	0
21	30.5	14.5	27	12	2
22	29	19	10.5	3	0
23	30	17	24.5	9.5	1
24	33.5	16.5	30	11	2
25	33.5	14.5	26.5	9.5	1
26	31	15.5	21.5	8	1
27	32.5	17	25	9	1
28	28	22.5	9.5	2.5	0
29	35.5	17.5	25.5	9.5	1
30	35.5	18.5	30.5	14.5	2

Fig 5.2: Testing Data Set

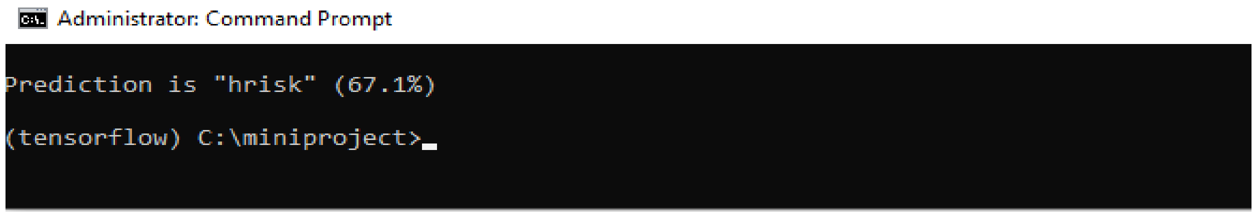
The above figure represents the training data set.20% of data is stored in training data set and rest in testing data



```
Enter the value of 'dayhigh': 27.5
Enter the value of 'daylow': 22
Enter the value of 'pdclose': 20
Enter the value of 'pdvar': 4.5
```

Fig 5.3: Input Data

The above figure represents the parameters entered by user.



```
Prediction is "hrisk" (67.1%)

(tensorflow) C:\miniproject>
```

Fig 5.4: Output Result

The above figure represents prediction of risk based on the parameters given by the user.

6.CONCLUSION AND FUTURE WORK

Conclusion

With Tensorflow, Google has created a framework that is simultaneously too low level to use comfortably for rapid prototyping, yet too high level to use comfortably in cutting edge research or in production environments that are resource constrained.

In our project,we study the use of DNN to predict financial risks in the stock market DNN is superior to all other classification methods in forecasting the risks.This is a clear message for financial forecasters and traders,which can lead to a capital gain.However,each method has its own strengths and weaknesses.

Future Work

Using neural networks to forecast stock market prices will be a continuing area of research as researchers and investors strive to outperform the market, with the ultimate goal of bettering their returns. It is unlikely that new theoretical ideas will come out of this applied work. However, interesting results and validation of theories will occur as neural networks are applied to more complicated problems. Neural networks appear to be the best modeling method currently available as they capture nonlinearities in the system without human intervention. Continued work on improving neural network performance may lead to more insights in the chaotic nature of the systems they model. However, it is unlikely a neural network will ever be the perfect prediction device that is desired because the factors in a large dynamic system, like the stock market, are too complex to be understood for a long time

7. References

[1] Heng-Tze Cheng, Zakaria Haque, TensorFlow Estimators: Managing Simplicity vs. Flexibility in High-Level Machine Learning Frameworks, Google, Inc.

[2] Lamartine Almeida Teixeira ,Adriano Lorena Inácio , Predicting Stock Trends through Technical Analysis and Nearest Neighbor Classification.

[3] Alejandro Rodríguez-González, , Improving N Calculation of the RSI Financial Indicator Using Neural Networks, International Journal on Recent and Innovation Trends in Computing and Communication.