**CHAPTER 1**

**INTRODUCTION**

The introduction chapter sets the stage by providing an overview of the company, Varcons Technology, and its commitment to advancing innovative solutions in the healthcare domain. Additionally, it delineates the problem statement, focusing on the imperative need for accurate prediction of chronic diseases to facilitate early intervention and personalized healthcare strategies.

**1.1 COMPANY PROFILE**

Varcons Technology Pvt Ltd, was incorporated with a goal "To provide high quality and optimal Technological Solutions to business requirements of our clients". Every business is a different and has a unique business model and so are the technological requirements. They understand this and hence the solutions provided to these requirements are different as well. They focus on client’s requirements and provide them with tailor made technological solutions. They also understand that Reach of their Product to its targeted market or the automation of the existing process into e-client and simple process are the key features that our clients desire from Technological Solution they are looking for and these are the features that we focus on while designing the solutions for their clients.

Varcons Technology Pvt Ltd, strive to be the front runner in creativity and innovation in software development through their well-researched expertise and establish it as an out of the box software development company in Bangalore, India. As a software development company, they translate this software development expertise into value for their customers through their professional solutions.

They understand that the best desired output can be achieved only by understanding the clients demand better. Varcons Technology Pvt Ltd work with their clients and help them to define their exact solution requirement. Sometimes even they wonder that they have completely redefined their solution or new application requirement during the brainstorming session, and here they position themselves as an IT solutions consulting group comprising of high calibre consultants.

They believe that Technology when used properly can help any business to scale and achieve new heights of success. It helps Improve its efficiency, profitability, reliability; to put it in one sentence "Technology helps you to Delight your customers" and that is what we want to achieve.

**1.2** **About the company:**

Varcons Technology Pvt Ltd is a Technology Organization providing solutions for all web design and development, MYSQL, PYTHON Programming, HTML, CSS, ASP.NET and LINQ. Meeting the ever increasing automation requirements, Varcons Technology Pvt Ltd specialize in ERP, Connectivity, SEO Services, Conference Management, effective web promotion and tailor-made software products, designing solutions best suiting clients requirements. The organization where they have a right mix of professionals as a stakeholders to help us serve our clients with best of our capability and with at par industry standards. They have young, enthusiastic, passionate and creative Professionals to develop technological innovations in the field of Mobile technologies, Web applications as well as Business and Enterprise solution. Motto of our organization is to "Collaborate with our clients to provide them with best Technological solution hence creating Good Present and Better Future for our client which will bring a cascading a positive effect in their business shape as well". Providing a Complete suite of technical solutions is not just our tag line, it is Our Vision for Our Clients and for Us, We strive hard to achieve it.

**1.3 Products of Varcons Technology Pvt Ltd:**

**1. Android Apps:**

﻿It is the process by which new applications are created for devices running the Android operating system. Applications are usually developed in Java (and/or Kotlin; or other such option) programming language using the Android software development kit (SDK), but other development environments are also available, some such as Kotlin support the exact same Android APIs (and bytecode), while others such as Go have restricted API access.

The Android software development kit includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and zutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, and Windows 7 or later. As of March 2015, the SDK is not available on Android itself, but software development is possible by using specialized Android applications.

**2. Web application:**

It is a client-server computer program in which the client (including the user interface and client-side logic) runs in a web browser. Common web applications include web mail, online

retail sales, online auctions, wikis, instant messaging services and many other functions. web applications use web documents written in a standard format such as HTML and JavaScript, which are supported by a variety of web browsers. Web applications can be considered as a specific variant of client-server software where the client software is downloaded to the client machine when visiting the relevant web page, using standard procedures such as HTTP. The Client web software updates may happen each time the web page is visited. During the session, the web browser interprets and displays the pages, and acts as the universal client for any web application. The use of web application frameworks can often reduce the number of errors in a program, both by making the code simpler, and by allowing one team to concentrate on the framework while another focuses on a specified use case. In applications which are exposed to constant hacking attempts on the Internet, security- related problems can be caused by errors in the program.

Frameworks can also promote the use of best practices such as GET after POST. There are some who view a web application as a two-tier architecture. This can be a "smart" client that performs all the work and queries a "dumb" server, or a "dumb" client that relies on a "smart" server. The client would handle the presentation tier, the server would have the database (storage tier), and the business logic (application tier) would be on one of them or on both. While this increases the scalability of the applications and separates the display and the database, it still doesn't allow for true specialization of layers, so most applications will outgrow this model. An emerging strategy for application software companies is to provide web access to software previously distributed as local applications. Depending on the type of application, it may require the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software application without having to install it on a local hard drive. A company which follows this strategy is known as an application service provider (ASP), and ASPs are currently receiving much attention in the software industry.

Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application and there are some key operational areas that must be included in the development process. This includes processes for authentication, authorization, asset handling, input, and logging and auditing. Building security into the applications from the beginning can be more effective and less disruptive in the long run.

**3. Web design:**

﻿It is encompassing many different skills and disciplines in the production and maintenance of websites. The different areas of web design include web graphic design; interface design; authoring, including standardized code and proprietary software; user experience design; and search engine optimization. The term web design is normally used to describe the design process relating to the front-end (client side) design of a website including writing mark up. Web design partially overlaps web engineering in the broader scope of web development. Web designers are expected to have an awareness of usability and if their role involves creating markup then they are also expected to be up to date with web accessibility guidelines. Web design partially overlaps web engineering in the broader scope of web development.

**1.4 Services provided:**

* Core Java and Advanced Java
* Web services and development
* Dot Net Framework
* Python
* Selenium Testing
* Conference/Event Management Service
* Academic Project Guidance
* On The Job Training
* Software Training

**1.5 About Machine Learning:**

Machine learning, a branch of artificial intelligence, stands as a pivotal force shaping the contemporary technological landscape. At its core, it empowers computer systems to autonomously learn from data, discern patterns, and make informed predictions. This transformative technology has permeated diverse industries, catalyzing groundbreaking innovations. In healthcare, machine learning facilitates early disease detection and personalized treatment plans, enhancing the quality of medical care. Financial institutions leverage its capabilities for fraud detection, algorithmic trading, and risk assessment, fortifying the stability of the global economy.

Retail giants harness machine learning to recommend products, optimize supply chains, and precisely forecast consumer demand, leading to improved customer experiences and operational efficiency. Autonomous vehicles, a symbol of our future, heavily rely on machine learning for real-time decision-making, object detection, and route planning, offering the promise of safer and more efficient transportation systems. While machine learning fuels progress, it also brings forth ethical considerations, such as algorithmic bias, privacy concerns, and potential job displacement, necessitating ongoing dialogue and ethical frameworks. Looking ahead, the horizon of machine learning holds exciting trends, from explainable AI (XAI) that enhances model interpretability to the proliferation of edge computing, minimizing latency, and quantum machine learning, ushering in a new era of computational power. In essence, machine learning is not merely a technological advancement but a transformative force that continues to redefine industries, society, and the very fabric of our future.

**1.6 Problem Statement:**

In the world of financial markets, timely and accurate information is paramount for making informed investment decisions. The goal of this project is to develop a robust real-time Twitter sentiment analysis system tailored for stocks trading in Stockport. Sentiment analysis is a critical tool for gauging market sentiment and predicting future movements in stock prices based on the collective mood and opinions expressed on social media, particularly Twitter. However, existing sentiment analysis tools often suffer from limited accuracy and effectiveness in the context of financial markets. Therefore, the primary objective of this project is to understand the mechanics of sentiment analysis and improve its accuracy within the Stockport stock market. By harnessing the power of natural language processing and machine learning, we aim to provide investors, traders, and financial analysts with a reliable tool that can assist in making data-driven investment decisions and better understanding market sentiment in real time. This project seeks to bridge the gap between social media sentiment and financial market performance in Stockport, ultimately contributing to more informed and profitable investment strategies.

**1.7 Objectives:**

**1. Data Collection and Integration:** Gather and aggregate real-time Twitter data related to stocks trading in Stockport. Establish a robust data pipeline to ensure the continuous and reliable acquisition of relevant tweets, stock market data, and other contextual information.

**2. Data Preprocessing:** Clean, preprocess, and format the collected Twitter data to remove noise, irrelevant information, and duplicates. Extract relevant features such as tweet text, timestamp, user sentiment, and stock symbols.

**3. Sentiment Analysis Model Development:** Develop a state-of-the-art sentiment analysis model using natural language processing (NLP) techniques and machine learning algorithms. Train the model on a labeled dataset to accurately classify tweets as positive, negative, or neutral sentiments.

**4. Real-Time Analysis:** Implement a real-time sentiment analysis system that continuously processes incoming tweets, assigns sentiment scores, and aggregates sentiment trends over time. Ensure scalability to handle high volumes of Twitter data.

**5**. **Stock Market Data Integration:** Integrate real-time stock market data from Stockport with the sentiment analysis results. Create a unified dataset that includes stock price movements and sentiment scores.

**6. Prediction Model Development:** Build predictive models that leverage historical sentiment data and stock market performance to forecast future stock price movements. Experiment with various machine learning and time series forecasting techniques to improve prediction accuracy.

**7. Accuracy Improvement:** Continuously refine and optimize the sentiment analysis model to enhance its accuracy in capturing nuances of financial sentiment expressed on Twitter. Explore techniques such as transfer learning and domain adaptation.

**8. Visualization and Reporting**: Create intuitive data visualizations and dashboards that display real-time sentiment trends and their correlation with stock market movements in Stockport. Provide user-friendly tools for monitoring and analysis.

**9. Performance Evaluation:** Conduct rigorous performance evaluations of the sentiment analysis and prediction models using historical data. Assess their effectiveness in providing actionable insights for stock market participants.

**10. User Feedback and Iteration:** Gather feedback from users, including investors, traders, and financial analysts, and use it to iteratively improve the system's accuracy, usability, and features.

**11. Documentation and Knowledge Transfer:** Document the entire process, from data collection to model development, in a comprehensive and accessible manner. Ensure knowledge transfer to stakeholders for the system's maintenance and future enhancements.

**12. Ethical Considerations:** Address ethical concerns related to privacy, data security, and bias in sentiment analysis. Implement safeguards to ensure responsible data usage and decision-making.

**13. Scalability and Robustness:** Design the system to be scalable and robust, capable of handling increased data volumes and maintaining performance under varying conditions.

By achieving these objectives, the project aims to provide a valuable tool for market participants in Stockport, empowering them to make more informed decisions by harnessing the power of real-time sentiment analysis and predictive analytics.

**CHAPTER 2**

**SYSTEM REQUIREMENTS**

**2.1 Hardware Requirements:**

* Processor: Min. dual core processor
* Clock speed: 2.4 GHz
* RAM: Min.8 GB
* Storage: Min.256 GB free space
* Monitor: Min.10” of any resolution

**2.2 Software Requirements:**

* API: Twitter
* OS: Windows/Linux
* IDEs: Docker Desktop, Anaconda, Elastic-search, Kimbra

**2.3 Libraries/Modules Used:**

* Python 3.x
* Elasticsearch 5.x
* Kibana 5.x
* elasticsearch python module
* nltk python module
* requests python module
* tweepy python module
* beautifulsoup4 python module
* textblob python module
* vaderSentiment python module
* newspaper3k python moduleter

**CHAPTER 3**

**IMPLEMENTATION AND DESIGN ANALYSIS**

**3.1 Config.py**

In the provided code snippet, a configuration file is defined for a program or script. This file contains several key settings and variables. Notably, the configuration includes details for connecting to an Elasticsearch instance, which seems to be used for data storage and retrieval. Additionally, the file stores authentication credentials required for accessing the Twitter API, which will enable the retrieval of Twitter data related to stock market and finance topics. The configuration also encompasses natural language processing parameters, specifying relevant keywords and phrases, as well as terms to be ignored during analysis. Furthermore, there is a list of Twitter handles representing various accounts, likely including financial experts and media sources, from which tweets are to be gathered for subsequent analysis. In essence, this code serves as a foundation for a system designed to collect, process, and analyze real-time Twitter data with a focus on finance and stock market-related content.

**The pseudocode:**

elasticsearch\_host = "172.18.0.2"

elasticsearch\_port = 9200

elasticsearch\_user = "sanjeev"

elasticsearch\_password = "sanjeev2002"

consumer\_key = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

consumer\_secret = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

access\_token = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

access\_token\_secret = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

nltk\_tokens\_required = ("neuralink", "solar", "tesla", "@tesla", "#tesla", "tesla", "tsla","$tsla", "#tsla", "elonmusk", "elon", "musk", "spacex", "starlink")

nltk\_min\_tokens = 0

nltk\_tokens\_ignored = ("win", "giveaway")

twitter\_feeds = ["@elonmusk", "@cnbc", "@benzinga", "@stockwits",

"@Newsweek","@WashingtonPost","@breakoutstocks", "@bespokeinvest”,"@WSJMarkets","@stephanie\_link", "@nytimesbusiness","@IBDinvestors","@WSJDealJournal","@jimcramer", "@TheStalwart","@TruthGundlach","@Carl\_C\_Icahn", "@ReformedBroker","@bespokeinvest","@stlouisfed","@muddywatersre", "@mcuban", "@AswathDamodaran","@elerianm","@MorganStanley", "@ianbremmer","@GoldmanSachs","@Wu\_Tang\_Finance","@Schuldensuehner","@NorthmanTrader","@Frances\_Coppola","@bySamRo","@BuzzFeed","@nytimes"]

**3.2 Docker:**

This Dockerfile is a configuration file used for building a Docker container to host a Python-based application. It begins by selecting the Python 3.6 base image and sets a label for the image's maintainer. The working directory within the container is specified as "/app," where subsequent commands will be executed. To ensure the required Python dependencies are available, it copies the "requirements.txt" file from the host into the container and then uses pip to install the listed packages. Additionally, it downloads essential data files for the NLTK library, facilitating natural language processing tasks. An environment variable, PYTHONIOENCODING, is set to handle UTF-8 encoding. Lastly, the Dockerfile defines the entry point for the container, executing the "startup.sh" script using Bash. This Dockerfile streamlines the setup of a containerized environment for running the Python application and ensures the availability of necessary dependencies.

**The pseudocode:**

FROM python:3.6

LABEL maintainer="shirosai"

WORKDIR /app

COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

nltk.download('stopwords')"

COPY sentiment.py ./

COPY stockprice.py ./

COPY startup.sh ./

ENV PYTHONIOENCODING=utf8

ENTRYPOINT [ "bash", "startup.sh" ]

**3.3 Sentiment.py:**

The provided Python script is designed for real-time Twitter data collection and sentiment analysis. It utilizes various libraries and modules to process incoming tweets, extract relevant information, and perform sentiment analysis. The script extracts user details, cleans and tokenizes tweet text, and calculates sentiment scores using TextBlob and VADER Sentiment. It also handles URL extraction from tweets and analyzes linked content. The processed data is then indexed in an Elasticsearch instance. Overall, this script offers a concise yet powerful solution for real-time Twitter sentiment analysis, especially in financial and stock market contexts.

**The pseudocode:**

**i**mport sys

import json

import time

import re

import requests

import nltk

import argparse

import logging

import string

try:

import urllib.parse as urlparse

except ImportError:

import urlparse

from tweepy import API, Stream, OAuthHandler, TweepError,StreamListener

from textblob import TextBlob

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from bs4 import BeautifulSoup

from elasticsearch import Elasticsearch

from random import randint, randrange

from datetime import datetime

from newspaper import Article, ArticleException

# import elasticsearch host, twitter keys and tokens

from config import \*

STOCKSIGHT\_VERSION = '0.1-b.12'

\_\_version\_\_ = STOCKSIGHT\_VERSION

IS\_PY3 = sys.version\_info >= (3, 0)

if not IS\_PY3:

print("Sorry, stocksight requires Python 3.")

sys.exit(1)

# sentiment text-processing url

sentimentURL = 'http://text-processing.com/api/sentiment/'

# tweet id list

tweet\_ids = []

# file to hold twitter user ids

twitter\_users\_file = './twitteruserids.txt'

prev\_time = time.time()

sentiment\_avg = [0.0,0.0,0.0]

class TweetStreamListener(StreamListener):

def \_\_init\_\_(self):

self.count = 0

self.count\_filtered = 0

self.filter\_ratio = 0

# on success

def on\_data(self, data):

try:

self.count+=1

# decode json

dict\_data = json.loads(data)

print("\n------------------------------> (tweets: %s, filtered: %s, filter-ratio: %s)" \

% (self.count, self.count\_filtered, str(round(self.count\_filtered/self.count\*100,2))+"%"))

logger.debug('tweet data: ' + str(dict\_data))

text = dict\_data["text"]

if text is None:

logger.info("Tweet has no relevant text, skipping")

self.count\_filtered+=1

return True

# grab html links from tweet

tweet\_urls = []

if args.linksentiment:

tweet\_urls = re.findall(r'(https?://[^\s]+)', text)

# clean up tweet text

textclean = clean\_text(text)

# check if tweet has no valid text

if textclean == "":

logger.info("Tweet does not cotain any valid text after cleaning, not adding")

self.count\_filtered+=1

return True

# get date when tweet was created

created\_date = time.strftime(

'%Y-%m-%dT%H:%M:%S', time.strptime(dict\_data['created\_at'], '%a %b %d %H:%M:%S +0000 %Y'))

# store dict\_data into vars

screen\_name = str(dict\_data.get("user", {}).get("screen\_name"))

location = str(dict\_data.get("user", {}).get("location"))

language = str(dict\_data.get("user", {}).get("lang"))

friends = int(dict\_data.get("user", {}).get("friends\_count"))

followers = int(dict\_data.get("user", {}).get("followers\_count"))

statuses = int(dict\_data.get("user", {}).get("statuses\_count"))

text\_filtered = str(textclean)

tweetid = int(dict\_data.get("id"))

text\_raw = str(dict\_data.get("text"))

# output twitter data

print("\n<------------------------------")

print("Tweet Date: " + created\_date)

print("Screen Name: " + screen\_name)

print("Location: " + location)

print("Language: " + language)

print("Friends: " + str(friends))

print("Followers: " + str(followers))

print("Statuses: " + str(statuses))

print("Tweet ID: " + str(tweetid))

print("Tweet Raw Text: " + text\_raw)

print("Tweet Filtered Text: " + text\_filtered)

# create tokens of words in text using nltk

text\_for\_tokens = re.sub(

r"[\%|\$|\.|\,|\!|\:|\@]|\(|\)|\#|\+|(``)|('')|\?|\-", "", text\_filtered)

tokens = nltk.word\_tokenize(text\_for\_tokens)

# convert to lower case

tokens = [w.lower() for w in tokens]

# remove punctuation from each word

table = str.maketrans('', '', string.punctuation)

stripped = [w.translate(table) for w in tokens]

# remove remaining tokens that are not alphabetic

tokens = [w for w in stripped if w.isalpha()]

# filter out stop words

stop\_words = set(nltk.corpus.stopwords.words('english'))

tokens = [w for w in tokens if not w in stop\_words]

# remove words less than 3 characters

tokens = [w for w in tokens if not len(w) < 3]

print("NLTK Tokens: " + str(tokens))

# check for min token length

if len(tokens) < 5:

logger.info("Tweet does not contain min. number of tokens, not adding")

self.count\_filtered+=1

return True

# do some checks before adding to elasticsearch and crawling urls in tweet

if friends == 0 or \

followers == 0 or \

statuses == 0 or \

text == "" or \

tweetid in tweet\_ids:

logger.info("Tweet doesn't meet min requirements, not adding")

self.count\_filtered+=1

return True

# check ignored tokens from config

for t in nltk\_tokens\_ignored:

if t in tokens:

logger.info("Tweet contains token from ignore list, not adding")

self.count\_filtered+=1

return True

# check required tokens from config

tokenspass = False

tokensfound = 0

for t in nltk\_tokens\_required:

if t in tokens:

tokensfound += 1

if tokensfound == nltk\_min\_tokens:

tokenspass = True

break

if not tokenspass:

logger.info("Tweet does not contain token from required list or min required, not adding")

self.count\_filtered+=1

return True

# clean text for sentiment analysis

text\_clean = clean\_text\_sentiment(text\_filtered)

# check if tweet has no valid text

if text\_clean == "":

logger.info("Tweet does not cotain any valid text after cleaning, not adding")

self.count\_filtered+=1

return True

print("Tweet Clean Text (sentiment): " + text\_clean)

# get sentiment values

polarity, subjectivity, sentiment = sentiment\_analysis(text\_clean)

# add tweet\_id to list

tweet\_ids.append(dict\_data["id"])

# get sentiment for tweet

if len(tweet\_urls) > 0:

tweet\_urls\_polarity = 0

tweet\_urls\_subjectivity = 0

for url in tweet\_urls:

res = tweeklink\_sentiment\_analysis(url)

if res is None:

continue

pol, sub, sen = res

tweet\_urls\_polarity = (tweet\_urls\_polarity + pol) / 2

tweet\_urls\_subjectivity = (tweet\_urls\_subjectivity + sub) / 2

if sentiment == "positive" or sen == "positive":

sentiment = "positive"

elif sentiment == "negative" or sen == "negative":

sentiment = "negative"

else:

sentiment = "neutral"

# calculate average polarity and subjectivity from tweet and tweet links

if tweet\_urls\_polarity > 0:

polarity = (polarity + tweet\_urls\_polarity) / 2

if tweet\_urls\_subjectivity > 0:

subjectivity = (subjectivity + tweet\_urls\_subjectivity) / 2

logger.info("Adding tweet to elasticsearch")

# add twitter data and sentiment info to elasticsearch

es.index(index=args.index,

doc\_type="tweet",

body={"author": screen\_name,

"location": location,

"language": language,

"friends": friends,

"followers": followers,

"statuses": statuses,

"date": created\_date,

"message": text\_filtered,

"tweet\_id": tweetid,

"polarity": polarity,

"subjectivity": subjectivity,

"sentiment": sentiment})

# randomly sleep to stagger request time

time.sleep(randrange(2,5))

return True

except Exception as e:

logger.warning("Exception: exception caused by: %s" % e)

raise

# on failure

def on\_error(self, status\_code):

logger.error("Got an error with status code: %s (will try again later)" % status\_code)

# randomly sleep to stagger request time

time.sleep(randrange(2,30))

return True

# on timeout

def on\_timeout(self):

logger.warning("Timeout... (will try again later)")

# randomly sleep to stagger request time

time.sleep(randrange(2,30))

return True

**3.4 Stockprice.py:**

This Python script fetches real-time stock price data using a provided URL template, replacing a placeholder with the actual stock symbol. It continuously updates and indexes this data into Elasticsearch for analysis. The script accepts command-line arguments for customization, including Elasticsearch index settings, stock symbol, update frequency, and verbosity level for output messages. It provides basic error handling and logging. Overall, it offers a simple and efficient way to monitor and store stock price data for later analysis.

**The pseudocode:**

if \_\_name\_\_ == '\_\_main\_\_':

# parse cli args

parser = argparse.ArgumentParser()

parser.add\_argument("-i", "--index", metavar="INDEX", default="stocksight",

help="Index name for Elasticsearch (default: stocksight)")

parser.add\_argument("-d", "--delindex", action="store\_true",

help="Delete existing Elasticsearch index first")

parser.add\_argument("-s", "--symbol", metavar="SYMBOL",

help="Stock symbol to use, example: TSLA")

parser.add\_argument("-f", "--frequency", metavar="FREQUENCY", default=120, type=int,

help="How often in seconds to retrieve stock data (default: 120 sec)")

parser.add\_argument("-v", "--verbose", action="store\_true",

help="Increase output verbosity")

parser.add\_argument("--debug", action="store\_true",

help="Debug message output")

parser.add\_argument("-q", "--quiet", action="store\_true",

help="Run quiet with no message output")

parser.add\_argument("-V", "--version", action="version",

version="stocksight v%s" % STOCKSIGHT\_VERSION,

help="Prints version and exits")

args = parser.parse\_args()

# set up logging

logger = logging.getLogger('stocksight')

logger.setLevel(logging.INFO)

eslogger = logging.getLogger('elasticsearch')

eslogger.setLevel(logging.WARNING)

requestslogger = logging.getLogger('requests')

requestslogger.setLevel(logging.WARNING)

logging.addLevelName(

logging.INFO, "\033[1;32m%s\033[1;0m"

% logging.getLevelName(logging.INFO))

logging.addLevelName(

logging.WARNING, "\033[1;31m%s\033[1;0m"

% logging.getLevelName(logging.WARNING))

logging.addLevelName(

logging.ERROR, "\033[1;41m%s\033[1;0m"

% logging.getLevelName(logging.ERROR))

logging.addLevelName(

logging.DEBUG, "\033[1;33m%s\033[1;0m"

% logging.getLevelName(logging.DEBUG))

logformatter = '%(asctime)s [%(levelname)s][%(name)s] %(message)s'

loglevel = logging.INFO

logging.basicConfig(format=logformatter, level=loglevel)

if args.verbose:

logger.setLevel(logging.INFO)

eslogger.setLevel(logging.INFO)

requestslogger.setLevel(logging.INFO)

if args.debug:

logger.setLevel(logging.DEBUG)

eslogger.setLevel(logging.DEBUG)

requestslogger.setLevel(logging.DEBUG)

if args.quiet:

logger.disabled = True

eslogger.disabled = True

requestslogger.disabled = True

# print banner

if not args.quiet:

c = randint(1, 4)

if c == 1:

color = '31m'

elif c == 2:

color = '32m'

elif c == 3:

color = '33m'

elif c == 4:

color = '35m'

# set up elasticsearch mappings and create index

mappings = {

"mappings": {

"stock": {

"properties": {

"symbol": {

"type": "keyword"

},

"price\_last": {

"type": "float"

},

"date": {

"type": "date"

},

"change": {

"type": "float"

},

"price\_high": {

"type": "float"

},

"price\_low": {

"type": "float"

},

"vol": {

"type": "integer"

}

}

}

}

}

if args.symbol is None:

print("No stock symbol, see -h for help.")

sys.exit(1)

if args.delindex:

logger.info('Deleting existing Elasticsearch index ' + args.index)

es.indices.delete(index=args.index, ignore=[400, 404])

logger.info('Creating new Elasticsearch index or using existing ' + args.index)

es.indices.create(index=args.index, body=mappings, ignore=[400, 404])

# create instance of GetStock

stockprice = GetStock()

try:

# get stock price

stockprice.get\_price(symbol=args.symbol, url=url)

except Exception as e:

logger.warning("Exception: Failed to get stock data caused by: %s" % e)

except KeyboardInterrupt:

print("Ctrl-c keyboard interrupt, exiting...")

sys.exit(0)

**3.5 Design Analysis:**

The provided code snippets collectively constitute a comprehensive solution for real-time Twitter sentiment analysis and stock price tracking. The system is designed to gather data from Twitter, perform sentiment analysis, retrieve stock price information, and store the data in an Elasticsearch database for subsequent analysis.

The Twitter data collection component, represented by the `TweetStreamListener` class, continuously listens to incoming tweets and processes them in real time. It extracts essential information from tweets, such as user details, text content, and metadata, while also performing natural language processing (NLP) tasks like text cleaning and sentiment analysis. Additionally, the script handles URL extraction and sentiment analysis of linked content, enhancing the depth of sentiment analysis.

On the other hand, the stock price tracking component, encapsulated within the `GetStock` class, fetches real-time stock price data using a URL template that incorporates the stock symbol. This data is then formatted and indexed into Elasticsearch, allowing for historical price data storage and retrieval. The script provides flexibility through command-line arguments, enabling users to specify Elasticsearch index settings, stock symbols, update frequency, and verbosity levels.

Both components implement logging and error handling to ensure robustness and reliability. They interact with Elasticsearch for data storage, offering a centralized repository for both Twitter sentiment and stock price data.

In summary, this integrated system offers a valuable platform for monitoring real-time Twitter sentiment in the context of financial markets, while also providing essential tools for tracking and analyzing stock price data. The combination of sentiment analysis and stock data can provide valuable insights for traders, analysts, and researchers interested in understanding market trends and sentiment-driven influences on stock prices.

**CHAPTER 4**

**RESULTS**

This chapter contains the snapshots of the project.

Docker Window:

The docker window shows if the project container is running or not.

A screenshot of a computer

Description automatically generated

**Figure 4.1: Docker Window**

Execution Window:

Windows command prompt is used for execution.

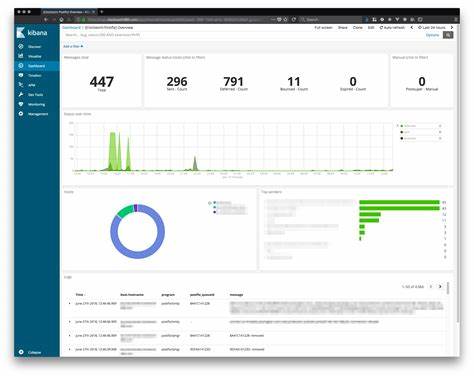
A computer screen shot of a computer screen

Description automatically generated

**Figure 4.2: Execution Window**

Kibana interface:

Kibana is used to have a graphical overview of elastic search.



**Figure 4.3: Kibana Interface**

Elastic Search:

Running prompt of elastic search.

A screenshot of a computer

Description automatically generated

**Figure 4.4: Elastic Search**

Kibana secondary:

Final predictive sentiment analysis.

A screenshot of a computer

Description automatically generated

**Figure 4.5: Kibana Secondary**

**CONCLUSION AND FUTURE ENHANCEMENTS**

In conclusion, this project presents a comprehensive and sophisticated solution for real-time Twitter sentiment analysis and stock price tracking. By seamlessly integrating data collection, sentiment analysis, and stock market data retrieval, it offers a powerful tool for gaining insights into the dynamics of financial markets. The system's ability to process Twitter data in real time, perform sentiment analysis on both tweet content and linked content, and concurrently track stock prices and store data in Elasticsearch underscores its versatility and relevance for traders, analysts, and researchers.

However, there is room for improvement in terms of user-friendliness, advanced analytics, and error handling. The addition of a user-friendly interface, advanced analytical capabilities, and enhanced error handling mechanisms would further enhance the project's utility and accessibility.

In summary, this project represents a valuable resource for those seeking to harness the power of social media sentiment analysis alongside stock market data. With its potential for refinement and expansion, it stands as a promising tool for understanding sentiment-driven influences on stock prices and making informed financial decisions in a dynamic market environment.

**Technical Outcomes:**

* **Algorithm Implementation:** Developed machine learning models using Random Forest, Naive Bayes, Decision Trees, and Gradient Boosting algorithms to predict chronic diseases. Processed the disease prediction dataset, including data cleaning, feature engineering, and model training.
* **Model Evaluation and Comparison:** Evaluated the performance of each algorithm using metrics like accuracy, precision, recall, and F1-score. Employed cross-validation techniques to ensure the reliability of model performance. Compared and analyzed the results to identify the most effective algorithm for chronic disease prediction.
* **Feature Engineering and Selection:** Utilized feature engineering techniques to extract relevant information from the dataset and improve model performance.
* **Integration and Deployment:** Prepared trained models for integration into a larger system by serializing them and developing APIs for inference. Overcame challenges related to integration to ensure smooth deployment. Worked towards the seamless integration of predictive models into a web-based platform for real-time disease predictions.

**Non Technical Outcomes:**

* **Communication Skills:** Improved your ability to communicate technical concepts to non-technical stakeholders. This may include presenting your findings to project managers, discussing project progress with team members, or explaining machine learning concepts to colleagues from different departments.
* **Collaboration Experience:** Gained experience working in a collaborative team environment, both with fellow interns and full-time employees. Learned how to effectively collaborate on projects, share ideas, and contribute to team discussions and decision-making processes.
* **Time Management:** Developed strong time management skills by balancing multiple tasks and deadlines during the internship. Learned to prioritize tasks, allocate time efficiently for coding, data analysis, and project documentation, and adapt to changing project requirements.

**Future Enhancements:**

* + - Dedicated Graphical User Interface (GUI)
    - Advanced Analytics
    - Sentiment Visualization
    - Customizable Alerts
    - Historical Data Analysis
    - Integration with Trading Platforms
    - Natural Language Processing Enhancements
    - Machine Learning for Sentiment Improvement
    - Community and Collaboration

**REFERENCES**

1. https://[www.javapoint.com](http://www.javapoint.com/)
2. https://[www.javatutorials.com](http://www.javatutorials.com/)
3. https://[www.geeksforgeeks.org](http://www.geeksforgeeks.org/)
4. <https://www.github.com/shirosaidev/stocksight>
5. https:/[/www](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC10403218/).[ncbi.nlm.nih.gov/pmc/articles/PMC10403218/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC10403218/)
6. <https://arxiv.org/ftp/arxiv/papers/2204/2204.05783.pdf>
7. https:/[/www](http://www.researchgate.net/publication/317214679_Sentiment_Analysis_for_Ef).[researchgate.net/publication/317214679\_Sentiment\_Analysis\_for\_Ef](http://www.researchgate.net/publication/317214679_Sentiment_Analysis_for_Ef) fective\_Stock\_Market\_Prediction
8. https://ieeexplore.ieee.org/document/9716326