Department of Computer Engineering

Academic Term: First Term 2023-24

Class: T.E /Computer Sem – V / Software Engineering

Practical No:	**1**
Title:	**Software Requirement Specification**
Date of Performance:	25.08.2023
Roll No:	9649
Team Members:	Ishita Yadav, Kashmira Sukhtankar, Aaron Rodrigues

Rubrics for Evaluation:

Sr. No	**Performance Indicator**	**Excellent**	**Good**	**Below Average**	**Total Score**
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Theory Understanding(02)	02(Correct)	NA	01 (Tried)	
3	Content Quality (03)	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Questions (04)	04(done well)	3 (Partially Correct)	2(submitted)	

Signature of the Teacher:

Department of Computer Engineering

Academic Term: First Term 2022-23

Class: T.E /Computer Sem – V / Software Engineering

Signature of the Teacher:

1. Abstract:

This report presents a stock market predictor model leveraging machine and deep learning on Google Colab. The model aims to forecast stock prices based on historical data and market indicators. Its purpose is to assist investors in making informed decisions. The scope is limited to predicting stock prices for specific companies on a particular exchange. References include relevant research papers and online resources. Developer requirements involve using suitable tools and libraries. User characteristics are investors, traders, and financial analysts. General constraints and assumptions are considered to impact model accuracy. Specific requirements include input data, functional aspects, external interfaces, performance, and design constraints with acceptance criteria.

2. Introduction:

The purpose of this report is to introduce a stock market predictor model developed using machine learning and deep learning techniques on Google Colab. The model aims to forecast stock prices by analysing historical data and market indicators. Its scope is to predict stock prices for a specific set of companies listed on a particular stock exchange. Definitions and references provide clarity and credibility. Developers require specific tools and libraries. Users are investors, traders, and analysts. General constraints and assumptions shape the model's performance. Specific requirements include input data, functional aspects, external interfaces, and performance constraints, along with design constraints and acceptance criteria.

2.1 Purpose:

The purpose of this project is to build an accurate and reliable stock market predictor model that can assist investors in making informed decisions.

2.2 Scope:

The model will focus on predicting stock prices for a specific set of companies listed on a particular stock exchange.

2.3 Definition:

Stock Market Predictor Model: A software system that uses historical stock data and market indicators to forecast future stock prices.

2.4 References:

I Parmar et al., "Stock Market Prediction Using Machine Learning," 2018 First International Conference on Secure Cyber Computing and Communication (ICSCC), Jalandhar, India, 2018, pp. 574-576, doi: 10.1109/ICSCCC.2018.8703332.

Kranthi Sai Reddy Vanukuru, “Stock Market Prediction Using Machine Learning”, Sreenidhi Institute of Science & Technology. Hyderabad, 2018

K. Hiba Sadia, Aditya Sharma, Adarrsh Paul, SarmisthaPadhi, Saurav Sanyal, “Stock Market Prediction Using Machine Learning Algorithms”, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8 Issue-4, April 2019

Heaton, J. B., Polson, N. G., & Witte, J. H. (2017). Deep learning for finance: deep portfolios. Applied Stochastic Models in Business and Industry, 33(1), 3-12.

2.5 Developers Requirements:

Specifications for a Stock Market Prediction Model without a User Interface:

1. Data Collection:
   - Obtain historical stock price data from financial APIs like Yahoo Finance or Alpha Vantage using Python libraries like `yfinance` or `alpha_vantage`.

- Gather financial indicators (e.g., P/E ratio, EPS) from financial databases or APIs.
- Retrieve market sentiment data from sentiment analysis tools or news APIs.

2. Data Preprocessing:
- Clean the data to handle missing values and outliers using libraries like `pandas`.
- Normalise or scale the data to bring all features to a similar scale.

3. Feature Engineering:
- Extract relevant features from the data, such as moving averages, price-to-earnings ratios, and technical indicators using libraries like `ta` (Technical Analysis Library) or custom functions.

4. Model Selection:
- Choose appropriate algorithms for prediction, such as linear regression, decision trees, or deep learning models like LSTM (Long Short-Term Memory) using libraries like `scikit-learn` or `tensorflow`.

5. Model Training:
- Split the data into training and validation sets using libraries like `scikit-learn`.
- Train the selected model using historical data.

6. Evaluation Metrics:
- Use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) from `scikit-learn` to evaluate model performance.

3. General Description:
The stock market predictor model functions as a tool to forecast stock prices based on historical data, financial indicators, and market sentiment. It targets investors, traders, and financial analysts who seek data-driven decisions. The model operates under certain constraints and assumptions, impacting its accuracy and performance. Inputs consist of historical stock prices and financial indicators, while outputs include predicted stock prices. Specific functionalities encompass data preprocessing, feature selection, model training, and prediction generation. External interfaces and data sources integrate with the model, and performance constraints are set to ensure its efficiency. Hardware and software constraints are outlined, and acceptance criteria measure its effectiveness.

3.1 Product Function Overview:
The stock market predictor model will analyse historical stock data, financial indicators, and market sentiment to generate predictions for stock prices.

3.2 User Characteristics:
The model is designed for investors, traders, and financial analysts who want to make data-driven decisions in the stock market.

3.3 General Constraints:
- Data Quality
- Overfitting
- Computational Resources
- Interpretability
- Regulatory Compliance

3.4 General Assumptions:

- Assumption of Stationarity: The model assumes that the statistical properties of the data, such as mean and variance, remain constant over time.

- Assumption of Independence: The model assumes that data points are independent and not influenced by previous observations.

- Assumption of Linearity: Linear models assume that the relationship between variables is linear and can be represented by a straight line.

- Assumption of Normality: Some models assume that the data follows a normal distribution for accurate predictions.

- Assumption of Homoscedasticity: The model assumes that the variance of the errors is constant across all levels of the independent variables.

4.Specific Requirements:

The stock market predictor model relies on inputs like historical stock data and market indicators to produce accurate stock price predictions as outputs. Functional requirements include data preprocessing, feature selection, model training, and generating predictions. External interface requirements involve integrating APIs and data sources. Performance constraints dictate the model's accuracy, processing time, and memory usage. Design constraints encompass software and hardware limitations. Acceptance criteria are established to evaluate the model's performance and overall efficacy in predicting stock prices for investors and traders. The report highlights the development and implementation of the model using machine and deep learning techniques on Google Colab.

4.1 Inputs and Outputs:

Inputs:
1. Historical Stock Price Data: Time-series data containing past stock prices for the target company or stock index.
2. Financial Indicators: Relevant financial metrics like earnings, revenue, P/E ratio, and other fundamental indicators.
3. Market Sentiment Data: Sentiment scores derived from news articles, social media, or market sentiment analysis.

Outputs:
1. Predicted Stock Prices: The model generates forecasts for future stock prices based on the input data and the chosen prediction method.
2. Confidence or Uncertainty Measures: Some models provide measures of confidence or uncertainty for their predictions to indicate the reliability of the forecasts.
3. Error Metrics: Evaluation metrics such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) may be outputted to measure the model's performance.
4. Trend Indicators: Binary indicators or classifications for whether the stock price is expected to rise or fall in the future.

4.2 Functional Requirements:

- Data preprocessing: Handle missing values, scale features, and engineer relevant features.
- Model selection: Choose appropriate algorithms and optimize hyperparameters.
- Real-time prediction: Integrate APIs for fetching real-time data and handle streaming data efficiently.
- Interpretability: Provide feature importance and explain model predictions.
- Performance metrics: Assess accuracy using evaluation metrics and ensure real-time response time.

4.3 External Interface Requirements:
- Financial APIs for real-time stock prices and market sentiment data.
- Data Sources for historical stock prices and financial indicators.
- News APIs for sentiment scores and market-related news.
- Real-time Data handling for streaming data.
- Output Formats for exporting prediction results.

4.4 Performance Constraints:
- Accuracy Target: Achieve a specific accuracy goal, e.g., 80%.
- Real-time Response: Generate predictions within 5 seconds per query.
- Efficient Resource Usage: Optimise computational resources to avoid excessive consumption.
- Scalability: Design the model to handle large datasets and potential growth.
- Stability: Ensure consistent performance over time and adaptability to market changes.

4.5 Design Constraints:

4.5.1 Software Constraints:
- Python Compatibility: The model should be compatible with the required Python version and dependencies.
- Library Dependencies: The model may have specific library requirements, such as TensorFlow, PyTorch, or scikit-learn.
- Platform Compatibility: Ensure the model is compatible with the target platform, whether it's Google Colab, a local server, or cloud infrastructure.
- Resource Constraints: Consider memory and processing limitations, especially when dealing with large datasets or complex models.
- Deployment Environment: The model should be deployable in the desired environment, considering factors like containerization or server configurations.

4.5.2 Hardware Constraints:
- Computing Power: The model's complexity may require high computational resources, such as GPUs or multiple cores, for efficient training and prediction.
- Memory: The size of the dataset and model parameters may demand sufficient memory capacity to avoid memory overflow during processing.
- Storage: Large datasets and model checkpoints may require adequate storage space for data retention and model saving.
- Network Bandwidth: If the model fetches real-time data from external APIs, a stable and sufficient network bandwidth is needed to avoid delays or interruptions.
- Latency: The hardware should support low-latency processing to ensure real-time predictions without significant delays.

4.5.3 Acceptance Criteria:
- Prediction Accuracy: The model should achieve a minimum accuracy of 75% on the validation dataset.
- Real-time Response: The model should generate predictions within 100 milliseconds for each data point during real-time inference.
- Backtesting Performance: The model's backtesting results should demonstrate consistent performance on historical data for at least six months.

- Documentation: The model documentation should be comprehensive, providing clear explanations of the model's architecture, data sources, and methodologies used.
- Maintenance: The model should be easy to maintain and update, allowing for seamless integration of new data and model improvements.

POSTLAB:

a) Importance of a well-defined SRS:

A well-defined Software Requirement Specification (SRS) is crucial as it outlines the project scope, prevents miscommunication, guides development, and mitigates risks. It serves as a basis for validation, estimation, and customer satisfaction.

b) Analysis and Improvement of SRS:

To enhance the SRS document, review it for ambiguities, ensure clear requirements, coherence, and consistency. Propose improvements with clearer descriptions and remove contradictions.

c) Comparison of Requirement Elicitation Techniques:

Interviews provide in-depth information, surveys reach many stakeholders quickly but lack depth, and use case modelling identifies interactions. Combining techniques is effective, depending on project size, complexity, and stakeholders' availability, improving understanding of user needs.