TIME-SERIES FORECASTING IN RETAIL INDUSTRY USING BI-DIRECTIONAL, STACKED AND VANILLA LSTM

A PROJECT REPORT

Submitted by

Harshini Srinivasan (RA1911003040107)

Lekhashree V (RA1911003040139)

Under the guidance of

Dr S. Manohar

(Assistant Professor, Department of CSE)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering

Vadapalani Campus, Chennai

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled **"Time-Series Forecasting in Retail Industry using Bi-directional, Stacked and Vanilla LSTM"** is the bonafide work of **"Harshini Srinivasan [Reg No: RA1911003040107]"** and **"Lekhashree [Reg No: RA1911003040139]"**, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROJECT GUIDE

Dr. S. Manohar,
B.E., M.E., Ph.D.,
Assistant Professor,
Dept of Computer Science & Engg,
SRMIST, Vadapalani Campus.

HEAD OF THE DEPARTMENT

Dr. S Prasanna Devi,
B.E., M.E., Ph.D., PGDHRM,PDF(IISc)
Professor and Head,
Dept of Computer Science & Engg,
SRMIST, Vadapalani Campus.

INTERNAL EXAMINER

EXTERNAL EXAMINER

Department of Computer Science and Engineering

SRM Institute of Science and Technology

Own Work Declaration Form

This sheet must be filled in and signed with dated along with student registration number, work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course :

Student Name :

Registration Number :

Title of Work :

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENTS

We express our humble gratitude to our Honorable Chancellor **Dr. T. R. Paarivendhar**, Pro Chancellor (Administration), **Dr. Ravi Pachamoothoo**, Pro Chancellor (Academic) **Dr. P. Sathyanarayanan** for the facilities extended for the completion of the project work.

We would record our sincere gratitude to our Vice Chancellor, **Dr. C. Muthamizhchelvan** and Registrar, **Dr. S. Ponnusamy** for their support to complete our project work by giving us the best of academic excellence support system in place. We extend our sincere thanks to our Dean, **Dr. C V Jayakumar** and Vice Principal – Academics, **Dr. C. Gomathy** and Vice Principal - Examination - **Dr. S. Karthikeyan** for their invaluable support.

We wish to thank **Dr. S Prasanna Devi**, Professor & Head, Department of CSE, SRM Institute of Science and Technology, Vadapalani Campus for her valuable suggestions and encouragement throughout the period of the project work and the course.

We are extremely grateful to our Project Coordinator, **Mrs. D. Punitha**, Assistant Professor, Department of CSE, SRM Institute of Science and Technology, Vadapalani Campus, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. S. Manohar**, Assistant Professor, Department of CSE, SRM Institute of Science and Technology, Vadapalani Campus, for providing us an opportunity to pursue our project under his mentorship. He provided us the freedom and support to explore the research topics of our interest.

We sincerely thank our management, staff and students of the Department of CSE, SRM Institute of Science and Technology, Vadapalani Campus who have directly or indirectly helped our project. Finally, we would like to thank our parents, our family members and our friends for their unconditional love, constant support and encouragement.

Harshini Srinivasan

Lekhashree V

ABSTRACT

Recently, interest in deep learning research and its applicability to practical issues has grown significantly. Developing a time-series analysis model to comprehend sales and profits/losses, as well as forecast future values, is crucial for businesses and companies, whether they operate online or offline. The objective of this project is to construct a time series analysis model that can comprehend sales and profits/losses while forecasting future values. To achieve an effective analysis, we have chosen LSTM deep learning architectures, including Stacked LSTM, Vanilla LSTM, and Bi-Directional LSTM. LSTM can accommodate time intervals of different sizes without running into the problem of vanishing gradients, in contrast to traditional recurrent neural networks. Additionally, they overcome the limitation of the stationarity assumption that is present in models like ARIMA, making them a more flexible and powerful tool for time-series analysis. The three distinct LSTM models are used to train the dataset and are compared with each other with respect to their accuracy measures. The conclusion of the thesis suggests that utilizing the Stacked LSTM deep learning architecture can greatly enhance the accuracy of sales prediction using financial data. Also, the thesis includes the forecast for the next 12 months. The implications of this thesis are significant for businesses and companies, as accurate sales prediction can help in making informed decisions related to production, inventory management, and marketing strategies. Furthermore, the findings of the thesis can also contribute to the realm of deep learning research, particularly concerning time-series analysis.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ARR - Artificial Neural Network

ADF - Augmented Dickey-Fuller

AR - Autoregressive

ARMA - Autoregressive Moving Average

ARIMA - Autoregressive Integrated Moving Average

BILSTM - Bi Directional Long Short-Term Memory

CNN - Convolutional Neural Network

DBN - Deep Belief Network

DNN - Deep neural network

ETS – Exponential Smoothing

GRU - Gated Recurrent Unit

KNN – k-Nearest Neighbour

LSTM - Long Short-Term Memory

MA – Moving Average

MAPE - Mean Absolute Percentage Error

MSE - Mean Squared Error

RMSE - Root Mean Squared Error

RNN - Recurrent Neural Network

SVM – Support Vector Machine

CHAPTER 1

INTRODUCTION

1.1. OVERVIEW

Time series analysis has a broad range of applications across various fields, including but not limited to electrical signal analysis, language processing and speech recognition, traffic analysis, weather forecasting, unemployment rate analysis, inflation dynamics analysis, and others. At its core, a time series refers to any sequence of regular interval measurements or observations in chronological order over a specific time frame. The analysis of time series data usually involves statistical techniques to characterize and model data patterns, such as Autoregressive (AR), Autoregressive Moving Average (ARMA), and Autoregressive Integrated Moving Average (ARIMA) models.

Financial markets provide a good example of a special type of time series, where various activities are typically represented and analysed. However, due to the noisy and complex nature of such time series, traditional methods relying solely on linear regression and parameter estimation may not accurately capture important patterns. Most financial time series exhibit nonlinear patterns, making it challenging to predict sales behaviour without more robust and highly nonlinear modelling techniques.

Deep learning, which leverages highly complex training data to perform prediction and classification operations, has emerged as a promising solution. The remarkable advancements in deep learning have led many scientific fields to explore its high-accuracy performance in building efficient solutions to diverse problems. Deep neural networks (DNNs) have demonstrated superior performance in other areas such as signal processing, speech recognition, and image classification.

Therefore, investigating LSTM's techniques for financial time series prediction is a recommended approach. Financial time series are well-suited for deep learning techniques, and numerous studies have applied different deep learning methods to time series prediction, including LSTM. These techniques can remember preceding data

inputs while using current data to learn network weights, improving the accuracy of predictions.

This project aims to develop a sales forecasting application for the retail industry using Long Short-Term Memory time series models. Sales forecasting is a crucial aspect of any business, particularly in the retail industry, where the ability to anticipate future demand can be the difference between success and failure. Utilising the Deep learning approaches such as LSTM models can improve the accuracy of sales forecasts and enable businesses to make informed decisions about resource allocation, managing cash flow, and predicting short-term and long-term performance.

The proposed project will compare the performance of different LSTM models such as Bidirectional LSTM, Vanilla LSTM and Stacked LSTM, and evaluate their ability to accurately forecast sales for the retail industry. By analysing historical sales data and applying deep learning algorithms, this project aims to develop a comprehensive sales forecasting model that can adapt to changing market trends and consumer behaviour. The project's primary objective is to provide decision-makers in the retail industry with a reliable and accurate sales forecasting tool that can support strategic planning and improve overall business performance.

The proposed project's development of a sales forecasting application using LSTM time series models will provide valuable insights for the retail industry. By utilizing deep learning algorithms to analyse sales data, businesses can make informed decisions about resource allocation and cash flow management, which can help improve overall performance and ensure long-term success.

1.2. DOMAIN OVERVIEW

1.2.1. Deep Learning

Deep learning is a type of machine learning that relies heavily on artificial neural networks, which are designed to imitate the human brain. Unlike traditional programming, deep learning does not require explicit programming for every step. While the concept of deep learning has been around for some time, it has gained significant attention in recent years due to the increase in processing power and data availability.

Deep neural networks, which are based on deep learning models and are motivated by the structure and operation of the human brain. They are capable of semi-supervised or unsupervised learning from massive volumes of data. For tasks like speech recognition, image recognition, and natural language processing, they are particularly effective because of this.

Feedforward neural networks, convolutional neural networks, and recurrent neural networks are the three most popular deep learning architectures. The simplest sort of ANN, having a linear information flow across the network, are feedforward neural networks. These networks have been extensively employed in tasks like speech recognition, image classification, and natural language processing.

For the purpose of recognizing images and videos, convolutional neural networks (CNNs) are a particular kind of feedforward neural network. They are perfect for applications like image classification, object identification, and image segmentation because they can automatically learn features from images.

Recurrent neural networks (RNNs) are a subclass of neural networks that are capable of processing sequential input, including time series and natural language. RNNs are excellent for applications like speech recognition, natural language processing, and language translation because they can maintain an internal state that captures data from prior inputs.

Large quantities of labelled data and extensive computer resources are needed for deep learning model training. Deep learning has been able to attain state-of-the-art performance in a variety of applications, including image and audio recognition, natural language processing, and more.

1.2.2. Types of Deep Learning

There are several types of deep learning architectures, including:

Deep Neural Network (DNN): This architecture can model and handle complicated, nonlinear connections since it has several hidden levels between the input and output layers.

Deep Belief Network (DBN) is a multi-layer belief network and a subclass of DNN. The Contrastive Divergence algorithm is used to learn a layer of features from visible units, previously trained features are treated as visible units when they are activated, and then the entire DBN is trained once the learning for the final hidden layer is complete.

Recurrent Neural Network (RNN) Its architecture is comparable to the extensive feedback network of interconnected neurons in the human brain and supports both parallel and sequential computing. RNNs are more accurate at their work because they are able to recall crucial details about the input they receive. RNNs perform the same task for every element of a sequence.

1.2.3. Working

The working process of Deep Learning involves several steps. First, it is a feasible solution. Second, Data that is pertinent to the real issue should be found and produced accordingly. Third, the right Deep Learning algorithm needs to be chosen. Fourth, the dataset should be trained using the algorithm. Finally, the dataset should be examined to gauge how well the Deep Learning model works.

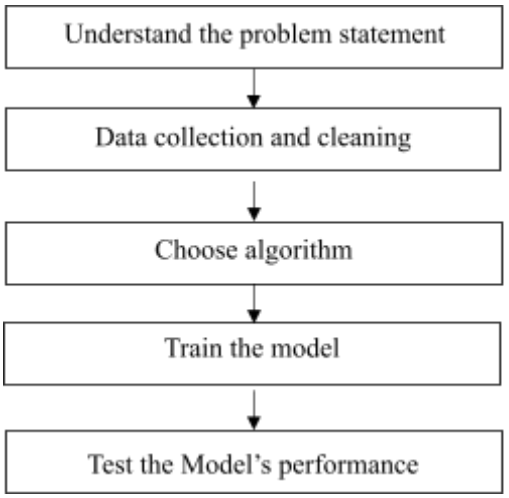


Figure 1.1: Deep Learning Architecture

1.2. OBJECTIVE

The project report will provide insights and recommendations on the application of LSTM networks for time-series forecasting in the retail industry, highlighting the

advantages of this approach over traditional methods such as ARIMA and RNN. The report will also discuss the limitations and potential future developments of the LSTM model for forecasting in the retail industry.

The time-series data from the retail business will be analyzed using the LSTM network to identify patterns and long-term dependencies, including seasonality, trends, and other intricate correlations between variables. The model will be trained on historical data to learn the underlying patterns and then used to forecast future sales and demand for different products and services.

The project report will include the following key components:

Data Preparation: The data will be collected and pre-processed to ensure that it is in the appropriate format for the LSTM model. This will involve cleaning, transforming, and normalizing the data, as well as identifying any missing or anomalous values.

Model Development: The LSTM model will be developed using the Keras library in Python. To make sure the model is accurately reflecting the underlying patterns and relationships in the data, it will be trained on historical data and validated using a holdout set.

Model Evaluation: Several measures, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error, will be used to assess the performance of the model (MAPE). The results will be compared to those of the ARIMA and RNN models to demonstrate the LSTM model's superiority.

Forecasting: The LSTM model will be used to forecast future sales and demand for different products and services in the retail industry. The forecasts will be compared to actual results to determine the model's accuracy and effectiveness in real-world scenarios.

1.3. PROJECT GOALS

1. Overall goals of the project include:
2. Improving inventory management
3. Enhancing operational efficiency
4. Increasing sales and revenue

5. Improving customer satisfaction
6. Key goals include:
7. Understand business problem
8. Hypothesis Generation
9. Collect data
10. Pre-process the dataset
11. Time- Serie decomposition
12. Model training & testing
13. Evaluation
14. Forecasting

1.4. SCOPE OF THE PROJECT

The scope of the project is to build a user – centric approach that relies solely on sales data to forecast future sales using the LSTM neural network.

CHAPTER 2

LITERATURE REVIEW

A literature review critically evaluates the current methodologies and knowledge perceived from various acclaimed sources in a particular period. It offers relevant information and discussions in multiple subject areas, providing valuable insights for future research work. The review also serves as a summary of the overall work and proposes research ideas.

The literature review is a combination of synthesis and summary, with information reorganized and recalled to provide critical insights into the subject matter. It interprets and analyses information from previous works, considering debates and progress made in the field. The review provides readers with an evaluation of relevant sources and discussions related to the research work.

Overall, a literature review is an essential component of any research work, offering insights into the subject matter and helping researchers identify areas that require further exploration.

There has been a significant increase in the application of LSTM deep learning architecture to time series forecasting, with the aim of improving accuracy, transparency, and efficiency. Because LSTM can handle time steps of any size and doesn't suffer from the vanishing gradient problem, it has been suggested in a number of studies that it be used for time series forecasting. These findings have important implications for improving the accuracy and reliability of time series forecasting, particularly in domains such as finance, economics, and meteorology.

2.1. REVIEW OF LITERATURE SURVEY

Gopalakrishnan.T et al. [1] implemented the linear regression using cost function and gradient descent. They obtained real time sales dataset from 2011-2013 to predict sales for 2014. The study compares actual values with the predicted sales values to calculate the accuracy rate and to validate the prediction. Rishi Raj Sharma et al. [2] discuss the implementation of ARIMA model along with EVDHM (Eigen Value Decomposition Hankel Matrix) for non-stationary time series which is defined by the PhilipsPerron Test (PPT). Generic Algorithm (GA) has been used to optimize the ARIMA parameters with minimizing AIC (Akaike Information Criterion) values.

Based on the historical dataset, Mehat Vijh et al. [3] developed ANN (Artificial Neural Network) and Random Forest to forecast the next day stock's closing price. Comparative investigation using RMSE, MAPE, and MBE shows that ANN provides superior prediction. Regression techniques like Linear Regression and Polynomial Regression were used by Saud Shaikh et al. [4] to analyze and forecast the COVID-19 outbreak in India. Polynomial Regression outperforms other models, according to analysis of the models using R squared score and error values.

Using a variety of machine learning models, including the Decision Tree (DT), Generalized Linear Model (GLM), Gradient Boost Tree, Sanjay. N. Gunjal et al. [5] performed Big-Mart sales prediction (GBT). Using error metrics like RMSE, MSE, and MAE to compare the models, it is found that GBT exhibits good accuracy. In order to estimate the sales based on the Big-Mart dataset, Varshini S. and Dr. D. Preethi [6] analyzed machine learning models such as XGBoost Regressor, Random Forest Regressor, ANN, and SVR (Support Vector Regression). According to the evaluation criteria of RMSE, R2 score, and MAPE, Random Forest performs better.

To analyze and forecast the Big-Mart Sales, Nayana R et al. [7] used the following ML models: Linear Regression, Ridge Regression, Polynomial Regression, and XGBoost Regression. XGBoost and Ridge Regression provide higher predictions based on accuracy rate. Meng-Chen Hsieh et al. [8] analyzed the impact of the supplier sharing their knowledge with the retailer on improving their own inventory-related expenses and forecasting, and how it influences the supplier's demand. The researchers discovered that when the retailer uses a suboptimal exponential smoothing (SES) forecast, the

supplier can recover the retailer's actual shocks and that with a thorough record of the retailer's orders, the supplier can determine the real ARMA model that creates the retailer's demand pattern.

Random Forest Regression, Support Vector Machine and Artificial Neural Networks were utilized by Xianghui Yuan et al. [9] to evaluate the profitability of multiple integrated stock selection models that employ different feature selection techniques and algorithms for predicting stock price trends. Findings indicate that applying Random Forest yields greater performance. Demand forecasting was carried out by Hossein Abbasimehra et al. [10] using various ML models, including ETS, ARIMA, ANN, SVM, KNN, basic RNN, and single layer LSTM. According to the calculated RMSE and SMAPE values, the LSTM outperforms the other models.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1. PROBLEM STATEMENT

Retail businesses generate large amounts of time series data on a daily, weekly, and monthly basis, including sales, customer traffic, and inventory levels. Accurate forecasting of these time series is critical for effective decision-making, such as inventory management, staffing, and marketing campaigns. However, traditional forecasting methods may not capture the complexity of the data, such as seasonality, trends, and non-linear relationships.

The aim of this work is to build and evaluate an LSTM model for deep learning that can be used for time series forecasting in the retail sector. The model should be able to handle complex time series data and provide accurate and reliable forecasts for key performance indicators such as sales, customer traffic, and inventory levels. The project aims to assess the feasibility and effectiveness of LSTM models for time series forecasting in the retail industry and provide insights into potential benefits and limitations of such models.

3.2.EXISTING SYSTEM

ARIMA and Recurrent Neural Networks approach (RNN) are two widely employed approaches for time-series forecasting.

3.2.1. Auto Regressive Integrated Moving Average (ARIMA)

The ARIMA model is a statistical technique employed for time-series analysis and forecasting. The ARIMA model is designed to capture the complex patterns of time-series data, including trends, seasonality, and irregularities.

Three elements make up ARIMA models: moving average (MA), integration (I), and autoregression (AR) (MA). The current observation and the prior observations are captured by the AR component, whereas the current observation and the historical error terms are captured by the MA component. The integration component adjusts the series

to make it stationary, which means that its statistical properties do not change over time.

The ARIMA is suitable for handling stationary time-series data. Therefore, the first step is to determine whether the dataset is stationary through either visual inspection or statistical tests. For visual inspection, the local mean of the dataset graph is compared with the global mean. On the other hand, the statistical test (i.e.) Augmented Dickey-Fuller (ADF) is used to determine whether the null hypothesis value is less than the threshold, which is usually 0.05. If it is less, the data is considered stationary. However, if it is greater, the time series data is non-stationary. In this scenario, various methods such as differencing or smoothing can be employed to transform the non-stationary data to a stationary form.

Once the time-series data is identified as stationary, the next step is to draw the autocorrelation function and partial autocorrelation function graphs to determine the values of AR-p, MA-q, and I-d. Then, an AR model, an MA model, and differencing are combined to construct the ARIMA model.

In summary, to use the ARIMA model, one must first determine whether the time-series data is stationary through visual inspection or statistical tests. If the data is non-stationary, it can be transformed to a stationary form using various methods. After that, the values of AR-p, MA-q, and I-d are determined, and an AR model, an MA model, and differencing are combined to construct the ARIMA model.

The ARIMA model's primary advantage is its ability to handle complex time-series patterns and forecast future values accurately. The ARIMA model may not be able to capture nonlinear relationships between variables, and it is limited in its ability to forecast longterm trends.

3.2.2. Recurrent Neural Network

Recurrent neural networks are a particular kind of neural network created with sequential data, such as time-series data. The RNN model has an internal memory that allows it to store information about past events and use that information to predict future events.

Unlike traditional neural networks, which process each input independently, RNNs can capture the temporal dependencies between inputs and producing outputs based on the entire input sequence.

The key characteristic of an RNN is its ability to maintain a hidden state that carries information across time steps. The RNN takes a vector as an input, updates its hidden state based on the input being processed at that time step and the prior hidden state, and then generates a vector output. The output vector can be fed back as input to the RNN at the next time step, allowing it to incorporate information from previous time steps.

Backpropagation through time (BPTT), a variation of the backpropagation algorithm used to train feedforward neural networks, is frequently used to train RNNs. In BPTT, the weights of the RNN are updated using gradient descent by computing the gradients of the loss function with respect to the weights at each time step.

Overfitting may result from RNNs' shortcomings in managing erratic and noisy data. The basic recurrent unit's inability to account for long-term dependencies is one of its main drawbacks. This is because gradients are propagated back through recurrent connections, they frequently either explode or disappear. This phenomenon, known as the vanishing gradient problem, may make it difficult for the RNN to learn dependencies that are long-term.

Many variations of the basic recurrent unit, such the LSTM unit and the Unit of Gated Recurrent, have been proposed to address this issue (GRU). These units have been demonstrated to be efficient at capturing long-term dependencies in sequential data. They are designed to selectively forget or retain information in the concealed state.

RNN's short term memory and stationarity problems can be solved by long short-term memory networks (LSTM).

3.3. DISADVANTAGES OF EXISTING SYSTEM

ARIMA & RNN are two commonly used time series models in the field of forecasting. LSTM is a specific form of RNN that is known for its ability to capture long-term dependencies in time series data.

While ARIMA and RNN can be effective in certain contexts, they also have some disadvantages compared to LSTM:

The mean, variance, and autocorrelation structure of the data are assumed to be stationary by ARIMA, which also assumes that the data has a linear trend. However, many real-world time series exhibit nonlinear trends and non-stationarity, which can lead to poor performance of ARIMA models.

Even though RNNs are capable of capturing non-linear relationships in time series data, the challenge of vanishing gradients, which arises when the gradients used to update the network's weights get very small, makes it difficult for the network to learn dependencies that are long-term. By including a gating mechanism that enables the network to only keep or discard certain information at each time step, LSTM was created to solve this issue.

RNNs also have the drawback of being computationally expensive to train, particularly when working with lengthy time series data. LSTM can be faster to train than traditional RNNs because of its gating mechanism, which allows the network to selectively update and forget information.

In general, ARIMA and RNNs require more expertise and manual tuning compared to LSTM, which has fewer hyperparameters to adjust and can be easier to implement.

In summary, while ARIMA and RNNs can be useful for time series forecasting, In terms of its ability to capture long-term dependencies and its effectiveness in training, LSTM offers several advantages over these models.

3.4. PROPOSED SYSTEM

3.4.1. Long Short - Term Memory (LSTM)

Long Short-Term Memory is the name of a kind of recurrent neural network called LSTM (RNN) is commonly used in natural language processing (NLP), speech recognition, and other sequential data analysis tasks. The VGP, which can arise in typical RNNs when the network is trained on lengthy data sequences, is addressed by LSTM networks.

A memory cell is the fundamental unit of an LSTM network and oversees maintaining and updating the internal state of the network as it processes sequential data. Many gates that regulate the flow of information into and out of the memory cell are part of the memory cell.

Forget Gate: This gate determines which data from the cell state should be discarded. It accepts the current input and the previous concealed state as inputs and outputs a number between 0 and 1 for each cell state element. A value of 0 indicates that the information should be deleted, but a value of 1 indicates that it should be retained.

Input gate: Selects which new data should be added to the cell state. It uses the current input and the previous concealed state as inputs and returns a number between 0 and 1 for each cell state element. Number 0 indicates that the new information should be discounted, while a value of 1 indicates that it should be considered.

Output Gate: This gate decides which information to output from the current hidden state. It takes as input the previous hidden state and the current input, as well as the current cell state, and outputs a number between 0 and 1 for each element in the hidden state. A value of 0 means that the information should be ignored, while a value of 1 means that the information should be output.

Cell State: This is the internal state of the memory cell, which is updated by the forget and input gates. It represents the long-term memory of the network.

The current input and the previous hidden state are fed into the LSTM network at the beginning of each time step, and it outputs the current hidden state and cell state. The hidden state is used as input to the next time step, and the cell state is updated by the forget and input gates. This process continues until the entire sequence has been processed.

The advantage of using an LSTM network over a traditional RNN is that it can handle long-term dependencies more effectively. This is because the cell state acts as a kind of memory that can store information from previous time steps, and the forget and input gates allow the network to selectively update and discard this information. As a result, LSTM networks are particularly useful for tasks that involve analysing sequences of text, speech, or other types of sequential data.

Three different LSTM time-series forecasting models have been applied in this study. The first one is called Vanilla LSTM and it has one output layer and one concealed layer of LSTM. The second LSTM form is known as Stacked LSTM and is made up of several LSTM stacked on top of one another. By enclosing the first hidden layer in a layer termed Bidirectional, the third LSTM model, also known as Bi-Directional LSTM, learns the input sequence both forward and backward.

3.4.2. LSTM Architecture

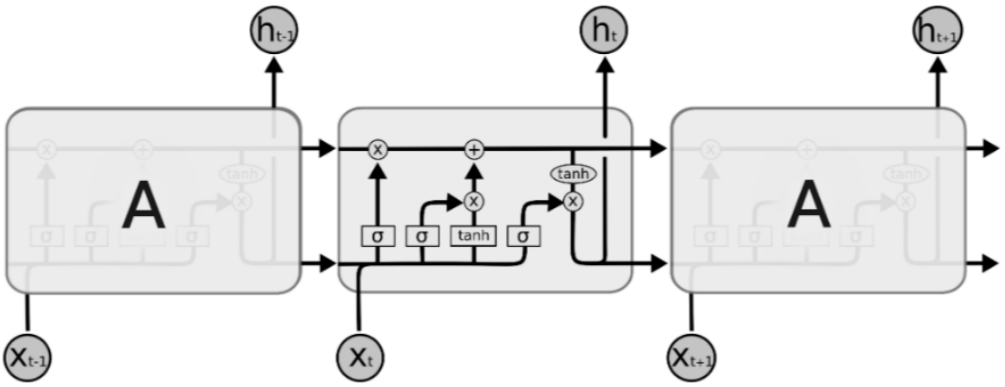


Figure 3.1: LSTM Architecture

LSTM, is a kind of neural network that deals with the problem of disappearing gradients in backpropagation. This is accomplished by controlling the memory storing process via a gating mechanism. Using gates that may be opened or closed, data can be stored, retrieved, or erased in an LSTM network. These gates utilize element-wise multiplication with sigmoid functions that output values between 0 and 1, allowing memory to be stored in an analog format. Because analog computation is differentiable, it is well-suited for use in backpropagation algorithms.

Activation functions:

TanH

TanH is a type of non-linear activation function that plays an important role in neural networks. It helps to regulate the flow of values through the network, ensuring that they remain within a range of -1 to 1. This is crucial for preventing information loss or

fading over time. In situations where certain values become very large, it can be helpful to use an activation function whose second derivative is able to persist for longer periods of time. The TanH function can achieve this, helping to prevent values from becoming insignificant.

Sigmoid

Sigmoid is a type of non-linear activation function that is commonly used in neural networks. It is frequently employed in gating devices, where it aids in regulating the flow of information by requiring values to lie between 0 and 1. This contrasts with the TanH function, which constrains values to the range of -1 to 1. The sigmoid function plays a critical role in allowing the network to update or forget data. If the result of a multiplication is 0, the information is forgotten, whereas a result of 1 indicates that the information should be retained. This mechanism helps the network to learn which data is important to remember and which can be safely disregarded.

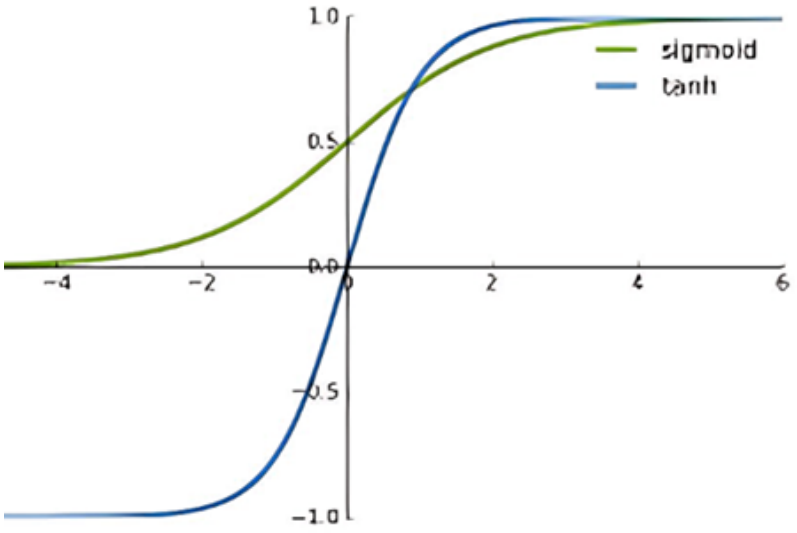


Figure 3.2: Activation functions

Rectified Linear Unit (ReLU)

ReLU, is a popular activation function in artificial neural networks. The function is simple and outputs the input if it is positive or 0 if it is negative.

The below is a mathematical expression of the ReLU function:

Because it is computationally effective and helps to solve the VGP that can arise in deep neural networks, the ReLU function is chosen over the other activation functions like sigmoid or tanh. When gradients propagate through deep networks, the vanishing gradient problem arises, making it challenging to update the weights in previous layers of the network.

ReLU offers a non-linear transformation that enables the network to learn more intricate representations of the input data, which can assist to avoid this issue. ReLU is very simple to build, which makes it a preferred option in many neural network topologies.

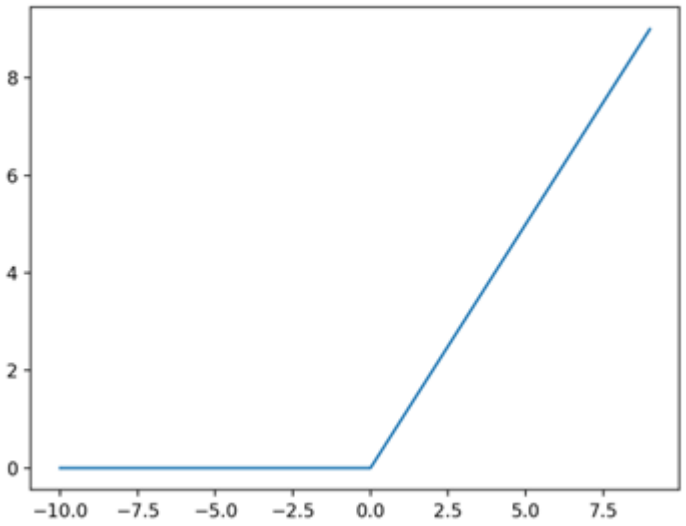


Figure 3.3: ReLU activation function

Forget Gate

The forget gate in an LSTM-based neural network is essential in deciding which input needs to be paid attention to and which may be safely disregarded. Both the current input X(t) and the prior hidden state h(t-1) are used as inputs to the forget gate, which is subsequently processed through a sigmoid function. The sigmoid function returns values between 0 and 1, which represent the relevance of the preceding result. A value that is closer to 1 denotes that the data should be kept since it is important. The cell then multiplies the points one by one using the value of f(t) that results. This enables the network to learn from and interpret information by allowing it to choose to retain or

discard information from earlier time steps more effectively.

$F_t = \sigma (W_f . [h_{t-1.} x_t] + b_t)$

t = timestep

f_t = forget gate at t

x_t = input

h_{t-1} = previous hidden state

w_f = weighted matrix between forget gate and input gate

b_t = connection bias at t

InputGate

An LSTM network's input gate is essential for updating the cell state. It carries out a number of operations on the current state X(t) and the prior hidden state h to accomplish this (t-1). These inputs are initially placed via a sigmoid function, which changes the values to range between 0 (signifying importance) and 1 (indicating importance) (indicating lack of importance). The identical inputs are then run via a TanH function. By generating a vector (C(t)) with values ranging from -1 to 1, this function aids in the regulation of the values. The point-by-point multiplication of the output values produced by these activation functions then enables the cell state to be modified depending on the significance of the input data. This method enables the network to preserve the most pertinent existing information while selectively incorporating new information.

$i_t = \sigma (W_i . [h_{t-1.} x_t] + b_t)$

$C_t = TanH (Wc . [h_{t-1.} x_t] + b_C)$

t = timestep

i_t = input gate at t

x_t = input

h_{t-1} = previous hidden state

W_i = weight matrix of sigmoid operator between input gate and output

b_t = bias vector at t w.r.t W_c

C_t = value generated by TanH

W_c = weight matrix of TanH operator between cell state information and network output

Cell State

The next step is to identify and store the updated cell state when the network has received the necessary data from the input gate and forget gate. The forget vector f(t), which controls which values should be maintained or dropped from the cell state, is multiplied by the prior cell state C(t-1) to accomplish this. These data will be removed if the outcome of this multiplication is 0.

The network then executes point-by-point addition on the output value of the input vector i(t), changing the cell state and providing a fresh value for C. (t). This procedure enables the network to maintain the most crucial data from earlier time steps while also selectively incorporating new information into the cell state.

$C_t = f_i * C_{t-1} + i_t] + C_t)$

t = timestepC_t = cell state information
f_t = forget gate at t
i_t = input gate at t
C_{t-1} = previous timestep
~
C_t = value generated by TanH

Output Gate

The value of the following hidden state, which holds data from earlier inputs, is determined by the output gate. The first step in this procedure is to run the values of the previous concealed state and the current state through a sigmoid function, which produces values between 0 and 1.

Next, a TanH function is used on the newly created cell state from the previous stage to produce a vector of values between -1 and 1. The information that the hidden state should contain is subsequently decided by multiplying these two vectors point by point. The new cell state and the new hidden state are both carried over to the following time step, and the hidden state is then employed for prediction. In conclusion, the input gate chooses what relevant information can be added from the current phase, the output gate finalizes the next hidden state, which is utilized for prediction, and the forget gate

chooses which relevant information from previous steps should be maintained.

$O_t = \sigma (W_o [h_{t-1.} x_t] + b_o)$

$h_t = O_t * TanH(C_{t)}$

t = timestep

O_t = output gate at t

w_o = weight matrix of output gate

b_t = bias vector w.r.t W_o

h_t = LSTM output

3.4.3. Vanilla LSTM

Vanilla LSTM is a type of RNN that can be used for time-series forecasting in the retail industry.

The Vanilla LSTM architecture includes the following three main components: the input layer, the LSTM layer, and the output layer. Here is a brief description of each component:

Input layer: The input layer receives the input data, which is typically organized as a sequence of vectors. Each vector represents one time step in the sequence, and contains one or more features (e.g., snacks sales data for a month). The input layer feeds the data to the LSTM layer.

To use vanilla LSTM for retail sales forecasting, historical sales data is organized into a time-series format and split into training and testing sets. The data is then fed into the vanilla LSTM model, which is trained using the training data to minimize the prediction error. The trained model can then be used to make predictions on the testing data and evaluated using metrics such as mean squared error (MSE) or mean absolute percentage error (MAPE).

LSTM layer: The LSTM layer is the core component of the Vanilla LSTM architecture. It consists of a series of LSTM cells that process the input data one time step at a time. Each LSTM cell has three gates (input, forget, and output) that regulate the flow of information in and out of the cell. The gates allow the LSTM layer to

selectively remember or forget previous inputs based on their relevance to the current prediction task.

Output layer: The output layer receives the output from the LSTM layer and generates the final prediction. The output layer can be a simple linear layer that maps the LSTM layer output to a scalar value (e.g., the predicted sales for the next time step), or it can be a more complex layer that incorporates additional features or non-linearities.

The Vanilla LSTM architecture can be extended with additional components such as dropout regularization, batch normalization, and attention mechanisms, depending on the specific application and performance requirements.

To optimize the performance of vanilla LSTM for retail sales forecasting, it is often beneficial to explore various hyperparameters such as the number of LSTM layers, the number of neurons in each layer, the learning rate, and the batch size. Additionally, feature engineering techniques such as lagged variables, trend analysis, and seasonality analysis can be used to improve the quality of the input features.

Vanilla LSTM has the benefit of being able to efficiently capture temporal dependencies in sequential data, which makes it suitable for time-series forecasting jobs like forecasting retail sales. However, variables like the volume of the training data, the complexity of the model architecture, and the caliber of the input features might have an impact on how well it performs.

In summary, vanilla LSTM is a powerful technique for retail sales forecasting that can effectively capture temporal dependencies in sequential data. Its performance can be optimized through careful selection of hyperparameters and feature engineering techniques. By incorporating these methods into a project report on retail sales forecasting, it is possible to provide insights and recommendations that can inform business strategy and decision-making.

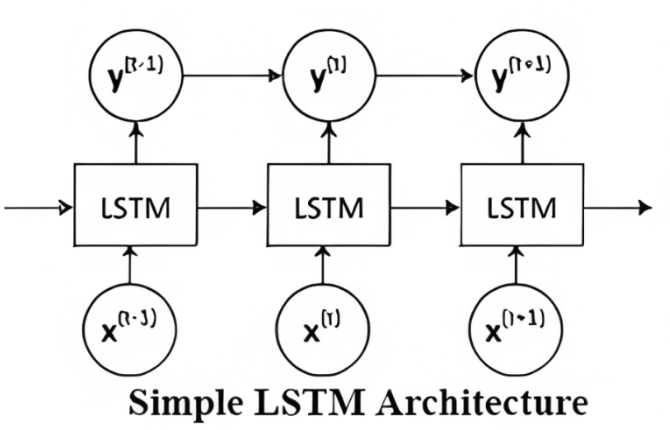


Figure 3.4: Vanilla LSTM Architecture

3.4.4. Stacked LSTM

Sequential data analysis frequently makes use of the Stacked LSTM model, a sort of deep learning architecture. It is made up of several LSTM layers layered on top of one another, with each layer processing the output of the one below it. However, in most circumstances, only two LSTM layers are stacked because more could lead to overfitting.

Based on existing data, the Stacked LSTM model can accurately forecast future values. A few LSTM layers are piled on top of one another to form its architecture. Each LSTM layer has a collection of memory cells that may selectively update or forget information based on the input data and retain information for lengthy periods of time.

The input to the Stacked LSTM model is a time-series sequence of data points, such as daily sales data for a retail store. The first LSTM layer in the model receives the input sequence and processes it to extract relevant features. The output of the first layer is then fed as input to the second layer, which further processes the information and extracts more complex features. This process is repeated for each subsequent layer, with each layer adding more levels of abstraction and complexity to the representation of the input sequence.

The output of the last LSTM layer is then fed into a fully connected layer, which maps the LSTM output to the desired output format. For example, if the goal is to predict future sales for the next 12 months, the output layer may contain 12 nodes, each representing the predicted sales for a specific month.

During training, the Stacked LSTM model learns the optimal weights and biases for each layer in the network through backpropagation. Backpropagation is the process of updating the model parameters in a way that minimizes the loss by computing the gradient of the loss function with respect to the model parameters.

It has been demonstrated that the Stacked LSTM model performs better than more established time-series models like ARIMA and other deep learning architectures like Vanilla LSTM and Bi-Directional LSTM. This is due to the Stacked LSTM model's improved ability to capture complex patterns and long-term dependencies in the input data.

The main advantage of the Stacked LSTM model is its ability to learn and represent complex patterns in the input data. By stacking multiple LSTM layers, the model can learn multiple levels of abstraction and capture more intricate dependencies between the input features and has made it a popular choice for researchers and practitioners working in fields such as finance, economics, and marketing, particularly effective for tasks such as speech recognition, natural language processing, and time-series analysis.

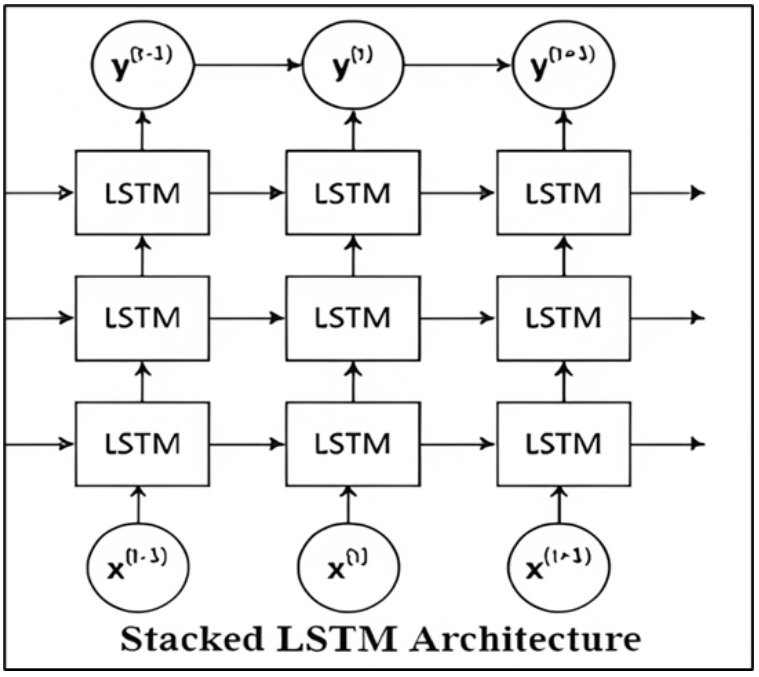


Figure 3.5: Stacked LSTM Architecture

3.4.5. Bidirectional LSTM

Recurrent neural network architecture known as bi-directional LSTM (BLSTM) enables the network to analyze input sequences in both forward and backward orientations. The architecture of BLSTM is based on the traditional LSTM model, with the addition of an extra layer that processes the input sequence in the reverse direction.

The architecture of BLSTM consists of two LSTM layers, one that processes the input sequence in the forward direction and the other that processes it in the reverse direction. Each LSTM layer has a set of hidden states that encode the context of the input sequence at each time step. The output of each LSTM layer is the hidden state at the last time step, which is fed into a fully connected layer that produces the final output.

Forward Layer: In the forward LSTM layer, the input sequence is processed from the first-time step to the last time step. At each time step, the input is fed into the LSTM cell, which updates its hidden state and produces an output. The output of the forward LSTM layer at the last time step is fed into the fully connected layer.

Backward Layer: In the backward LSTM layer, the input sequence is processed from the last time step to the first-time step. At each time step, the input is fed into the LSTM cell, which updates its hidden state and produces an output. The output of the backward LSTM layer at the first-time step is fed into the fully connected layer.

Finally, the outputs of the forward and backward LSTM layers are concatenated to produce the final output sequence. This allows the model to acquire past and future contextual information within the sequences of input, which is especially useful in NLP tasks where the meaning of a word depends on the context in which it appears.

In summary, BLSTM is a powerful model that can capture complex patterns in sequential data by processing the input sequence in forward as well as the backward directions. The advantage of BLSTM over unidirectional LSTM is that it can better capture long-term dependencies and capture more complex patterns in the input sequence. However, BLSTM requires more computational resources and can be slower to train compared to unidirectional LSTM.

Machine translation, sentiment analysis, handwriting recognition, speech recognition, and other applications have all been successfully implemented using BLSTM.

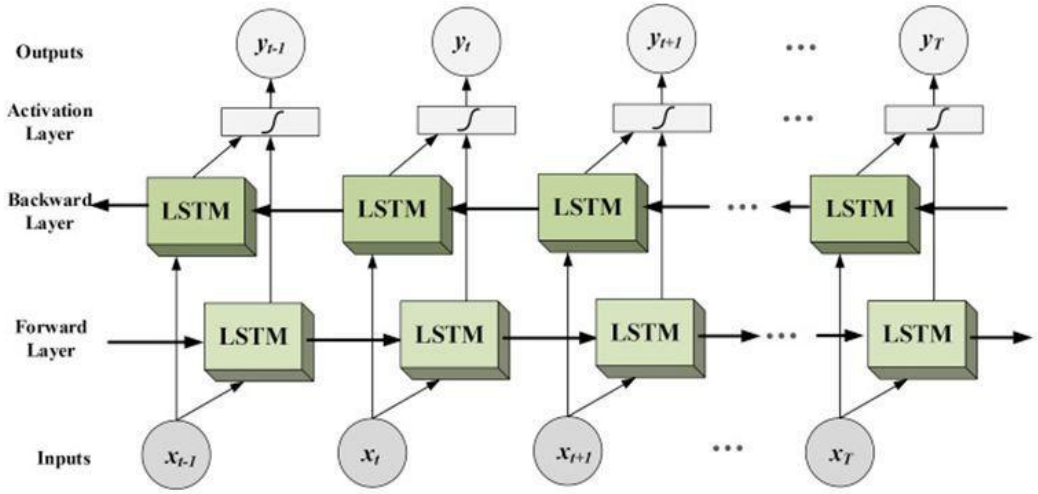


Figure 3.6: Bi-directional LSTM Architecture

3.4.6. Advantages

Since they can manage long-term dependencies in sequential data, Long Short-Term Memory (LSTM) networks, a subset of Recurrent Neural Networks (RNN), have grown in popularity in recent years. Here are some of the advantages of LSTM networks:

1. Can handle non-stationary data:

LSTM can handle non-stationary data by learning and identifying patterns in the data over time using specialized memory cells with gates that control the flow of information. This allows LSTM to identify the dependencies that persist for a long time in the data, making it a powerful tool for time series forecasting and other applications.

2. Ability to retain long-term dependencies:

The issue of vanishing gradients, which can happen in typical RNNs and make it challenging for the model to retain information over time, is addressed by LSTM networks. In order to preserve long-term dependencies in the data, LSTMs incorporate

a memory cell with the ability to selectively keep or forget information from prior time steps.

3. Flexibility in handling sequence lengths:

LSTM networks can handle input sequences of varying lengths, which makes them well-suited for a wide range of applications. This flexibility is achieved through the use of gates that can selectively update the memory cell and output state based on the input sequence, which allows the model to adapt to sequences of different lengths.

4. Robustness to noisy data:

LSTMs are robust to noisy data and can filter out irrelevant information in the input sequence. This is achieved using gates that can selectively update the memory cell and output state based on the input sequence, which allows the model to filter out noise and retain relevant information.

5. Ability to capture complex patterns:

LSTMs can capture complex patterns in sequential data, such as seasonal patterns or trends with non-linear behaviour. This is achieved using gates that can selectively update the memory cell and output state based on the input sequence, which allows the model to capture the underlying patterns in the data.

6. Easy to implement:

LSTMs are easy to implement and can be trained using standard optimization techniques such as backpropagation through time. This makes them accessible to a wide range of users, including those without a background in deep learning.

7. Automatic feature extraction:

LSTMs are capable of automatically extracting relevant features from the input sequence, which is especially helpful in time series forecasting on stationary datasets where the underlying patterns in the data may not be well understood. This can save time and effort in feature engineering and pre-processing steps.

8. Accurate and reliable forecasts:

LSTMs have been shown to produce accurate and reliable forecasts in numerous applications of time series forecasting. This is because to their potential to capture complex patterns in the data, handle long-term dependencies, and filter out noise.

9. Suitable for a wide range of applications:

Many different applications, such as speech recognition, machine translation, and time series forecasting, have effectively used LSTMs. This versatility is due to their ability to handle sequential data of varying lengths and their robustness to noisy data.

Overall, the advantages of LSTM networks make them a popular choice for a wide range of applications that involve sequential data. They are well-suited for handling long-term dependencies in the data, filtering out noise, and capturing complex patterns, which makes them a powerful tool for machine learning practitioners.

3.5. SYSTEM REQUIREMENT SPECIFICATION

3.5.1. Hardware System Configuration

- Processor – I3, i5,i7,AMD
- RAM -- 8 Gb
- Hard Disk -- 500 GB

3.5.2. Software System Configuration

- Operating System -- Windows 7/8/10
- Language – Python

3.6. SOFTWARE DESCRIPTION

3.6.1. Google Colab

Google Colab is a cloud-based development environment that provides users with access to a free, Jupyter notebook-based platform for building and running machine

learning models. It is a part of Google's suite of cloud-based tools, which also includes Google Drive, Google Docs, and Google Sheets.

Google Colab allows users to write, run, and share Python code in a collaborative environment. It includes a variety of tools and libraries that make it easy to build and test machine learning models, including TensorFlow, PyTorch, and Keras.

One of the key features of Google Colab is its ability to provide free access to GPUs and TPUs (Tensor Processing Units) for training machine learning models. This allows users to take advantage of high-performance computing resources without having to invest in expensive hardware.

Google Colab also includes a variety of other features that make it a powerful tool for data science and machine learning, including support for markdown cells, code cells, and text cells, as well as the ability to import and export data from Google Drive.

Overall, Google Colab is a powerful, free tool for building and testing machine learning models in the cloud. Its collaborative environment, access to high-performance computing resources, and built-in libraries and tools make it an ideal platform for data scientists and machine learning practitioners.

3.6.2. Jupyter Notebook

Jupyter Notebook is a freely available web-based tool that enables users to generate and exchange documents that incorporate live code, equations, data visualizations, and written explanations. Its name is derived from the fact that it supports the three fundamental programming languages: Julia, Python, and R.

Jupyter Notebook is built on a client-server architecture and runs in a web browser. It allows users to write and execute code in a variety of programming languages, including Python, R, Julia, and others. The code is organized into cells, which can be executed independently or together. This makes it easy to experiment with code and explore data in an interactive environment.

One of the key features of Jupyter Notebook is its support for rich media content, including images, audio, and video. It also supports a wide range of data visualization libraries, making it an ideal tool for data exploration and analysis.

Another important feature of Jupyter Notebook is its support for Markdown cells, which allows users to add narrative text and formatting to their documents. This makes it easy to create documents that combine code, visualizations, and explanatory text, making it an ideal platform for sharing research and analysis.

Overall, Jupyter Notebook is a robust and flexible tool for data exploration, analysis, and sharing. Its support for multiple programming languages, rich media content, and narrative text makes it an ideal tool for data scientists, researchers, and educators.

3.7. SYSTEM ARCHITECTURE

System architecture refers to the overall design and structure of a complex system or software application, including its components, their relationships, and the principles and guidelines governing their interaction. It is the blueprint or plan that guides the development and maintenance of a system throughout its entire lifecycle.

The system architecture defines the various subsystems, modules, and system components, and their interactions with each other and with external systems or users. It also specifies the hardware and software platforms, data structures, algorithms, protocols, and interfaces required to support the system's functionality and performance.

The system’s architecture has a significant impact on its scalability, security, maintainability, and dependability. A well-designed architecture ensures that the system can be easily maintained, modified, and expanded as needed, and that it can operate efficiently under varying workloads and conditions.

This project consists of eight modules.

1. Understanding the Business Problem
2. Data Collection & Pre-processing
3. Exploratory Data Analysis
4. Train & Test Split
5. Model Training

6. Validation Testing

7. Performance Evaluation

8. Sales Forecasting

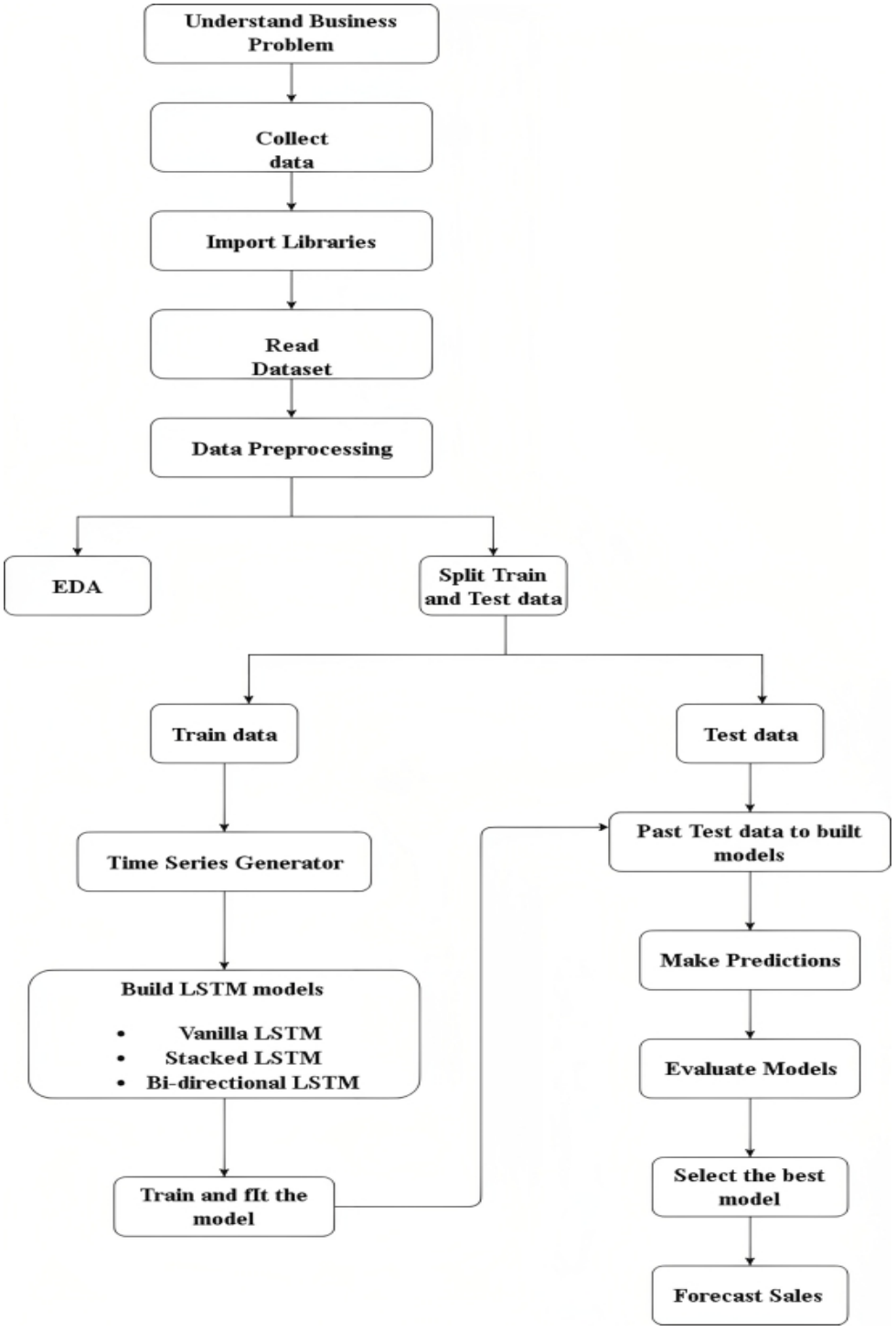


Figure 3.7: System Architecture

3.8. DESIGN

3.8.1. Activity Diagram

UML is the abbreviation for Unified Modeling Language. A form of UML diagram called a "activity diagram" is employed in software engineering and business process modelling to show the flow of operations carried out by a system, a procedure, or an organization.

Activity diagrams are frequently used to represent intricate business processes, software algorithms, and workflows. They can also be used to record how several actors or objects interact with one another in a system. Activities, decisions, flows, and forks are just a few of the parts of a process that are represented in activity diagrams using a set of standard symbols. The flows describe the order in which these activities occur, whereas the activities represent the specific stages or actions that are conducted. Choices are used to depict instances in the process when the flow can take many pathways depending on specific circumstances or inputs.

Activity diagrams, in general, offer a visual representation of a procedure or system, making it simpler to comprehend and explain how it operates.

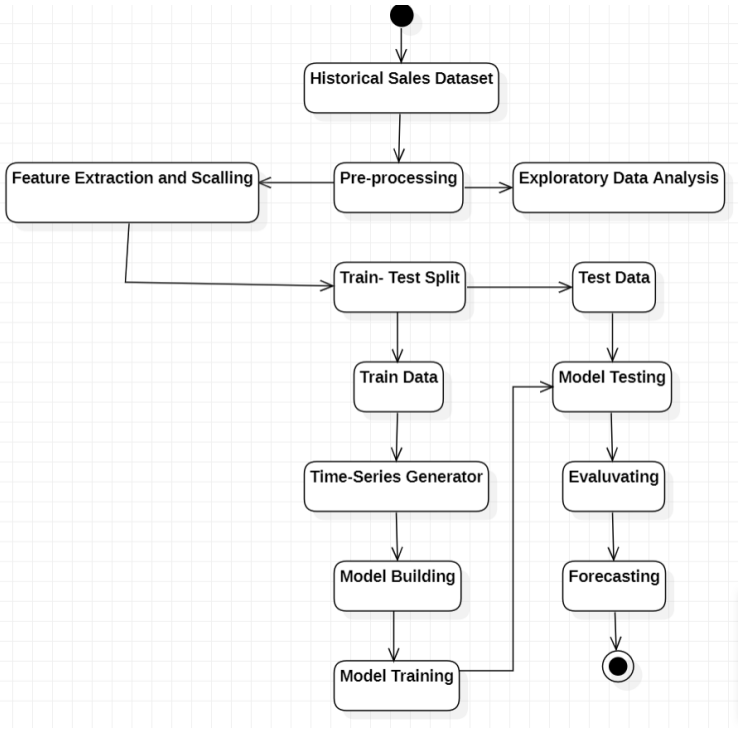


Figure 3.8: Activity Diagram

3.8.2. Use Case Diagram

The functional needs of a system or application are visualized using use case diagrams, which are a subset of UML diagrams. These diagrams are viewed from the viewpoint of the system or application's users or actors. It offers a broad overview of the actors who interact with the system and its capabilities.

In a use case diagram, actors are depicted as stick figures and use cases as ovals or rectangles. The use cases indicate actions or processes that the system or application must carry out in order to satisfy the demands of its users. The users, roles, or external systems that interact with the system to carry out these tasks or functions are represented by the actors.

Use case diagrams are frequently employed to explain to stakeholders, including business analysts, developers, and users, the specifications and functionality of a system or application. They can assist in identifying the essential components and specifications of the system, as well as any potential problems or areas in need of improvement.

Use case diagrams can also be used to define and identify system boundaries as well as to show the connections between actors and use cases. To make sure the system satisfies the needs of its users and stakeholders, they can be employed at any stage of the development lifecycle.

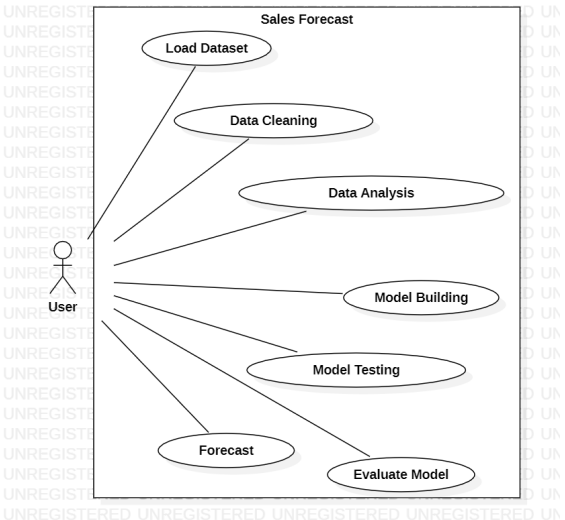


Figure 3.9: Use case Diagram

3.8.3. Sequence Diagram

A sequence diagram shows the relationships between items or components as horizontal arrows called messages and the interactions between them as vertical lines called lifelines. Each message is associated with a task or action carried out by one item or component that results in a reaction from a different object or component.

The order in which they appear on the diagram, with time flowing from top to bottom, represents the sequence of messages and activities. Together with any limitations or restrictions that apply, the graphic can also show the duration of each message or activity.

In software development, sequence diagrams are frequently used to describe the interactions between various system or application components and to pinpoint potential problems or areas for improvement. They can also be used to inform stakeholders and other team members of a system's design and functionality.

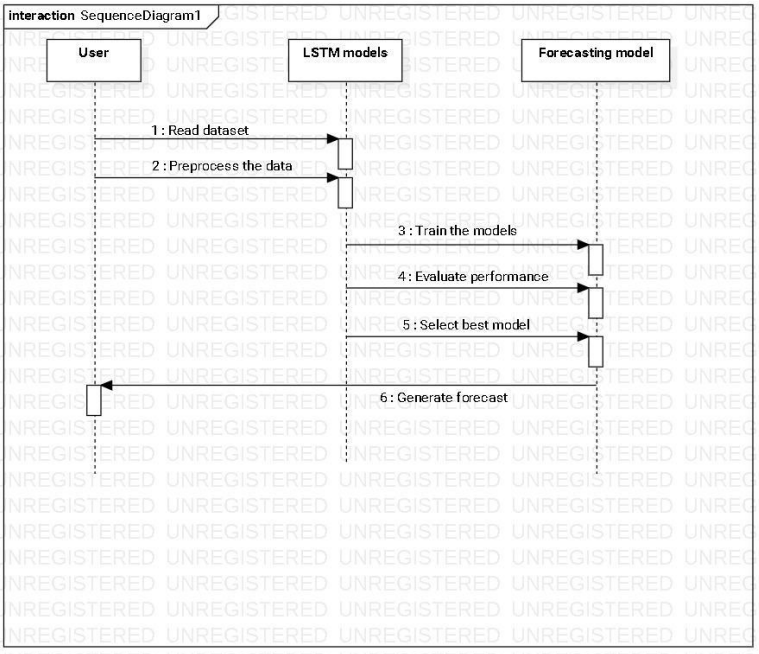


Figure 3.10: Sequence Diagram

CHAPTER 4

METHODOLOGY

4.1. DATA COLLECTION

Once the business issue has been recognised, the next phase is to gather the data that will be employed for time series forecasting. This involves gathering historical sales data, customer behaviour data, market trends, and any other relevant data sources that can provide insights into the problem at hand. Collecting and analyzing high-quality data is essential for building an accurate and effective forecasting model that can help retailers make informed decisions and stay ahead of the competition.

Data collection is a critical step in building an LSTM model for time series forecasting in the retail industry.

The following are some key considerations for **Data Collection**

Identify the relevant data sources: Retail businesses generate a wide range of data, including sales data, customer traffic data, inventory levels, and promotional activity. It is important to identify the specific data sources that are relevant for the forecasting task at hand.

Collect historical data:

Historical data is necessary for training and validating the LSTM model. Collect as much historical data as possible, ideally covering several years or more, depending on the specific forecasting task.

Collect additional data:

In addition to time series data, other relevant data may include product features, marketing campaigns, and seasonal events. Collecting this data can help improve the accuracy of the LSTM model.

Ensure data quality:

Ensure that the collected data is accurate, complete, and free from errors. It is important to address any data quality issues before proceeding with data pre-processing and model development.

Consider data privacy and security:

Retail data may contain sensitive information, such as customer data and financial data. Ensure that appropriate measures are taken to protect the privacy and security of the data.

Overall, data collection for time series forecasting using LSTM in the retail industry involves identifying relevant data sources, collecting historical and additional data, ensuring data quality, and considering data privacy and security. The accuracy as well as the effectiveness of the LSTM model for forecasting can be significantly impacted by the quality and amount of the acquired data. In this project, Super Mart Grocery sales - Retail Analytics Dataset from Kaggle has been considered.

This is a fictional dataset created for helping the data analysts to practice exploratory data analysis and data visualization. The dataset has data on orders placed by customers on a grocery delivery application.

The dataset is designed with an assumption that the orders are placed by customers living in the state of Tamil Nadu, India.

The dataset utilised for this study comprises about 10,000 observations and 11 attributes and depicts Super Mart Sales from 2019 through the end of 2022. This dataset contains sales data from seven different types of categories, 23 subcategories, 24 cities, and five different regions. Sales of snacks are the variable of interest in this experiment since they exhibit seasonal trends. The sales and order date of each data item are the most crucial characteristics for doing univariate forecasting with this publicly available dataset. Order ID, Customer Name, Sub Category, City, Order Date, Region, Sales, Discount, Profit & State are a few of the other features.

4.1.1. DATASET PREPARATION:

There are over 9994 rows of data in the dataset. Further there are 7 types in categories, 23 sub-categories, 24 cities & 5 regions

CATEGORIES

- Bakery
- Beverages
- Eggs, Meat & Fish
- Food Grains
- Fruits & Veggies
- Oil and Masala
- Snacks

SUB-CATEGORIES

- Organic Staples
- Fish
- Spices
- Organic Fruits
- Masalas
- Fresh Fruits
- Noodles
- Biscuits
- Breads & Buns
- Edible Oil & Ghee
- Rice
- Organic Vegetables
- Mutton
- Soft Drinks
- Atta & Flour
- Health Drinks
- Fresh Vegetables
- Cakes
- Eggs
- Chocolates
- Chicken
- Dals & Pulses
- Cookies

CITIES

- Chennai
- Perambalur
- Virudhunagar
- Pudukottai
- Ooty
- Coimbatore
- Namakkal
- Salem
- Nagercoil
- Krishnagiri
- Theni
- Tirunelveli
- Trichy
- Kanyakumari
- Cumbum
- Tenkasi
- Bodi
- Ramanadhapuram
- Dharmapuri
- Madurai
- Karur
- Viluppuram
- Vellore
- Dindigul

REGION

- West
- South
- East
- North
- Central

4.2. DATA PREPROCESSING

4.2.1. Data Cleaning

Retail data can have many anomalies such as missing values, outliers, and noise. Data cleaning involves identifying and removing these anomalies to prevent them from interfering with the accuracy of the LSTM model.

4.3. DATA ANALYSIS

4.3.1. Statistical Analysis

To summarise and characterise the data, descriptive statistics are employed. The data are described using measures of principal tendency like mean, median, and mode as well as metrics of variability like range, standard deviation, and variance

4.3.2. Exploratory Data Analysis

In the retail sector, exploratory data analysis (EDA) is a crucial stage in time series forecasting. EDA assist in spotting trends, cycles, and

irregularities in the data that may be exploited to increase the forecasting

model's precision.

4.4. TIME-SERIES DECOMPOSITION

EDA involves identifying trends and seasonality in the data. Trends are longterm patterns that show the overall direction of the data, while seasonality refers to recurring patterns that occur over a fixed period. This step is crucial in selecting the appropriate forecasting model.

4.5. DATA NORMALIZATION

LSTM models perform better when the data is normalized. To ensure that the data are of same magnitude, normalisation involves scaling the data between zero and one. This step is achieved using techniques such as min-max scaling.

4.6. TRAIN & TEST SPLIT

As a final preprocessing step, it is necessary to divide the data into separate training and testing datasets. The training data is utilized to train the LSTM model, while the testing data is employed to evaluate its performance. For training and testing, the normal split is 70:30 or 80:20, respectively. The split between the data used for training and for testing in this project is in 75:25 ratio.

By taking these pre-processing steps, retailers can improve the performance of their LSTM models, resulting in more accurate time series forecasts.

4.7. MODEL TRAINING

Recurrent Neural Networks (RNN) are commonly used for time-series forecasting as they can remember information from past events. However, the RNN may face the vanishing gradient problem, where the output diminishes or explodes after multiple hidden layers due to multiplication. To solve this problem, the Long Short-Term Memory (LSTM) network was developed. LSTM uses back-propagation over time to overcome the vanishing gradient problem and enhance short-term memory.

This project investigates the effectiveness of three different LSTM models for time series forecasting. The first model is the Vanilla LSTM, which has one hidden layer of LSTM and one output layer. The second model is the Stacked LSTM, consisting of multiple LSTM layers stacked on top of each other. The third model is the Bidirectional LSTM, which learns the input sequence both forward and backward by incorporating a Bidirectional layer in the first hidden layer.

Training LSTM models is crucial for time-series forecasting. The aim is to optimize the model's parameters to minimize prediction error. In this project, the backpropagation algorithm with the mean squared error (MSE) loss function was used to train the three different LSTM models. The Keras library was used for model training as it allows for easy implementation of various deep learning models. Each model was trained for 200 epochs, and the best model was selected based on its performance on the testing set.

4.8. MODEL TESTING

Model testing is an essential step in a time series forecasting project as it helps to evaluate the performance and accuracy of the model before deploying it for predictions.

The training set was used to optimize the model parameters, while the testing set was used to evaluate the model's performance.

4.9. MODEL EVALUATION

The performance of the models was evaluated using various metrics, such as mean absolute percentage error (MAPE), root mean squared error (RMSE), and mean squared error (MSE).

i. **Mean Squared Error (MSE)** is a commonly used statistic to assess a forecasting model's accuracy. The average of the squared discrepancies between the predicted and actual values is what is measured. The procedure entails multiplying the total number of observations in the dataset by the square of the difference between each actual and anticipated value before adding them altogether. The following is the MSE formula:

MSE= (1/n) * ∑ (actual - predicted) ^2) (4.1)

(Where n is the dataset's total number of observations).

MSE is a useful metric as it penalizes large errors more than small errors, and it is easy to interpret since it is expressed in the same units as the squared error.

ii. **MAPE** stands for **Mean Absolute Percentage Error**, and is a commonly used statistic to assess the precision of a forecasting model. The average percentage difference between the anticipated and actual numbers is what is measured.

In order to calculate MAPE, one must first calculate the absolute difference between the anticipated and actual values, divide that result by the actual value, and then average those results over all forecasts. The formula for MAPE can be expressed as follows:

MAPE = (1/n) * ∑ (abs (actual - predicted) / actual) * 100 (4.2)

(Where n is the number of observations in the dataset).

MAPE is a useful metric as it provides a percentage value, which makes it easier to interpret the accuracy of the model compared to other metrics such as Mean Squared Error (MSE) or Mean Absolute Error (MAE).

iii. **RMSE**, which stands for **Root Mean Squared Error** is a statistic frequently used to assess the precision of a prediction model. It is used for estimating the discrepancy between a continuous variable's actual and expected values. The RMSE is calculated as the square root of the average of the squared differences between the anticipated values and the actual values. The prediction error is bigger with higher values. It gives an indication of the typical gap between the values that were projected and the actual values.

The following formula is used to determine RMSE:

RMSE = sqrt(mean((predicted - actual)2)) (4.3)

The evaluation metrics in the table show that the Stacked LSTM model has the lowest values for all three metrics, indicating better performance compared to the Vanilla LSTM and Bidirectional LSTM models. The Stacked LSTM has an MSE of 9183.94, an RMSE of 95.83, and a MAPE of 6.01, which are lower than the corresponding metrics for the other two models.

The least accurate performance is displayed by the Vanilla LSTM model, which has the greatest MSE and RMSE values among the three models. In comparison to the other two models, the Bidirectional LSTM model has the greatest MAPE value, indicating a higher percentage of prediction error.

Overall, it appears that the Stacked LSTM model outperforms all other models for the given dataset and evaluation measures. A comprehensive examination and comparison of many models is usually advised before drawing any conclusions because differing hyperparameter tuning and other evaluation criteria may produce different outcomes.

4.10. FORECASTING

After determining that Stacked LSTM is the best model, the project uses it to forecast future sales for the following 12 months while keeping in mind that sales forecasting is uncertain and necessitates continual evaluation and improvement.

CHAPTER 5

CODE IMPLEMENTATION

Data Pre-processing:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
from sklearn.preprocessing import MinMaxScaler
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers import Bidirectional
from keras.layers import LSTM
import warnings
from sklearn.metrics import mean_squared_error
from pandas import datetime
from pandas import DataFrame
from pandas import concat
from pandas import Series
!pip install -U kaleido
```

```
<ipython-input-1-153d3b894166>:16: FutureWarning: The pandas.datetime class is deprecated and will b
  from pandas import datetime
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting kaleido
  Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl (79.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.9/79.9 MB 8.8 MB/s eta 0:00:00
Installing collected packages: kaleido
Successfully installed kaleido-0.2.1
```

```
df= pd.read_csv('/content/Updated Dataset.csv')
df
```

	Order ID	Customer Name	Category	Sub Category	City	Order Date	Region	Sales	Discount	Profit	State
0	OD3324	Jackson	Bakery	Breads & Buns	Salem	03-01-2019	Central	864	0.26	380.16	Tamil Nadu
1	OD740	Haseena	Bakery	Cakes	Nagercoil	04-01-2019	Central	2033	0.15	325.28	Tamil Nadu
2	OD741	Vinne	Beverages	Health Drinks	Trichy	04-01-2019	Central	698	0.32	223.36	Tamil Nadu
3	OD742	Sudha	Snacks	Chocolates	Dharmapuri	04-01-2019	Central	759	0.27	174.57	Tamil Nadu
4	OD3873	Verma	Eggs, Meat & Fish	Fish	Tirunelveli	05-01-2019	East	878	0.35	149.26	Tamil Nadu

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Order ID       9994 non-null   object 
 1   Customer Name  9994 non-null   object 
 2   Category       9994 non-null   object 
 3   Sub Category   9994 non-null   object 
 4   City           9994 non-null   object 
 5   Order Date     9994 non-null   object 
 6   Region         9994 non-null   object 
 7   Sales          9994 non-null   int64  
 8   Discount       9994 non-null   float64
 9   Profit         9994 non-null   float64
 10  State          9994 non-null   object 
dtypes: float64(2), int64(1), object(8)
memory usage: 859.0+ KB
```

```
[ ]  df['order_date'] = pd.to_datetime(df['Order Date'])
     df=df.drop('Order Date', axis=1)
```

```
     df.shape
```

```
     (9994, 11)
```

```
[ ]  df.columns
```

```
     Index(['Order ID', 'Customer Name', 'Category', 'Sub Category', 'City',
            'Region', 'Sales', 'Discount', 'Profit', 'State', 'order_date'],
           dtype='object')
```

```
[ ]  df.rename(columns=lambda x: x.strip().lower().replace(" ", "_"), inplace=True)
     df.columns
```

```
     Index(['order_id', 'customer_name', 'category', 'sub_category', 'city',
            'region', 'sales', 'discount', 'profit', 'state', 'order_date'],
           dtype='object')
```

```
df.describe()
```

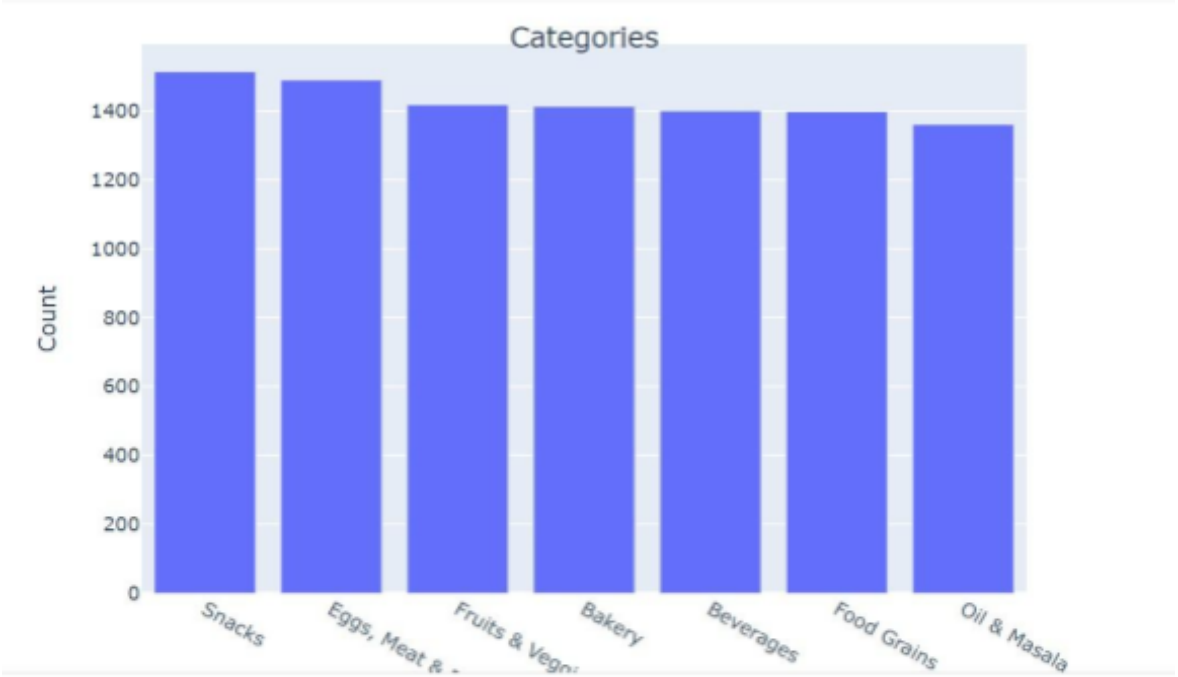
	sales	discount	profit
count	9994.000000	9994.000000	9994.000000
mean	1496.603362	0.226817	374.937082
std	577.565552	0.074636	239.932881
min	500.000000	0.100000	25.250000
25%	1000.000000	0.160000	180.022500
50%	1498.000000	0.230000	320.780000
75%	1994.750000	0.290000	525.627500
max	2500.000000	0.350000	1120.950000

```
for c in df.columns:
    if len(set(df[c]))<25:
        print(c,set(df[c]))
```

```
category {'Bakery', 'Snacks', 'Beverages', 'Oil & Masala', 'Eggs, Meat & Fish', 'Fruits & Veggies', 'Food Grains'}
sub_category {'Health Drinks', 'Mutton', 'Dals & Pulses', 'Atta & Flour', 'Spices', 'Masalas', 'Organic Fruits', 'Fre
city {'Chennai', 'Dindigul', 'Cumbum', 'Salem', 'Tenkasi', 'Trichy', 'Perambalur', 'Bodi', 'Viluppuram', 'Vellore', '
region {'East', 'West', 'North', 'Central', 'South'}
state {'Tamil Nadu'}
```

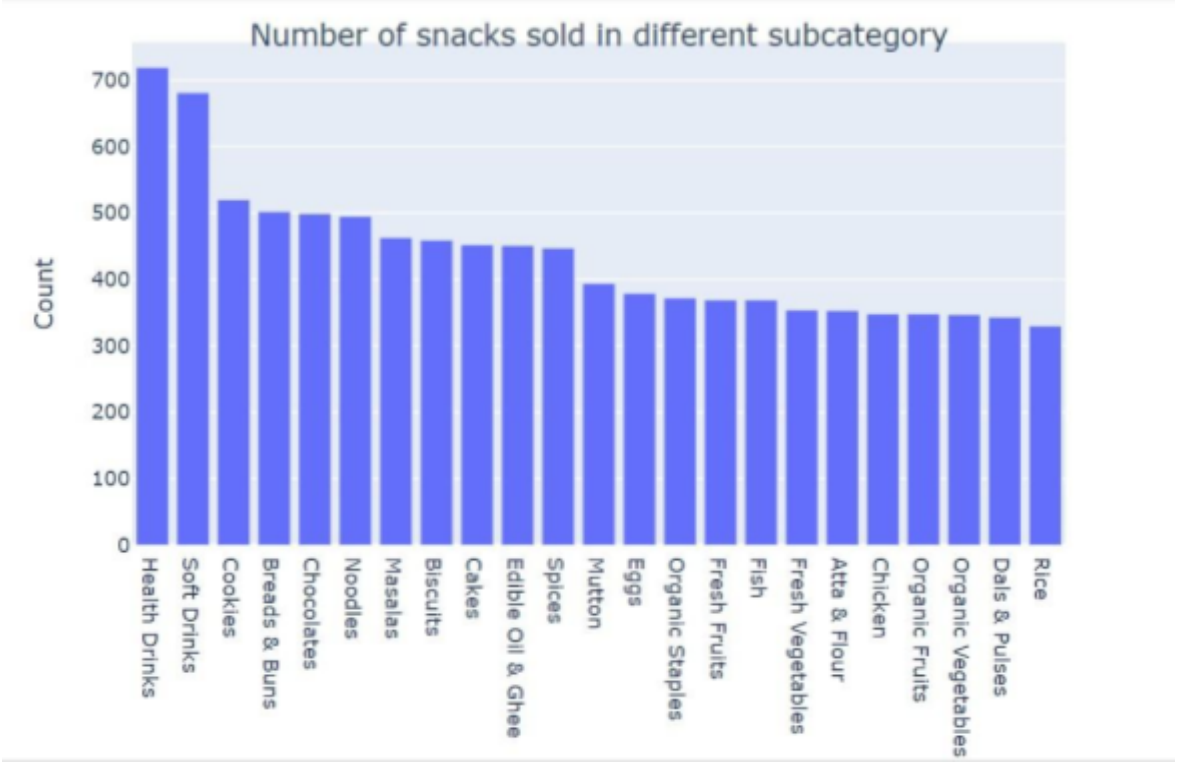
```
ax = df['category'].value_counts()
fig = px.bar(ax, x=ax.index, y=ax.values, height=500)
fig.update_layout( title={
        'text': "Categories",
        'y':0.9,'x':0.5,'xanchor': 'center','yanchor': 'top'},
    xaxis_title="Category",
    yaxis_title="Count")
print(ax)
fig.show(renderer='svg')
```

```
Snacks               1514
Eggs, Meat & Fish    1490
Fruits & Veggies     1418
Bakery               1413
Beverages            1400
Food Grains          1398
Oil & Masala         1361
Name: category, dtype: int64
```




```
bx = df['sub_category'].value_counts()
fig = px.bar(bx, x=bx.index, y=bx.values, height=500)
fig.update_layout( title={
        'text': "Number of snacks sold in different subcategory",
        'y':0.9,'x':0.5,'xanchor': 'center','yanchor': 'top'},
    xaxis_title="Sub Category",
    yaxis_title="Count")
print(bx)
fig.show(renderer='svg')
```

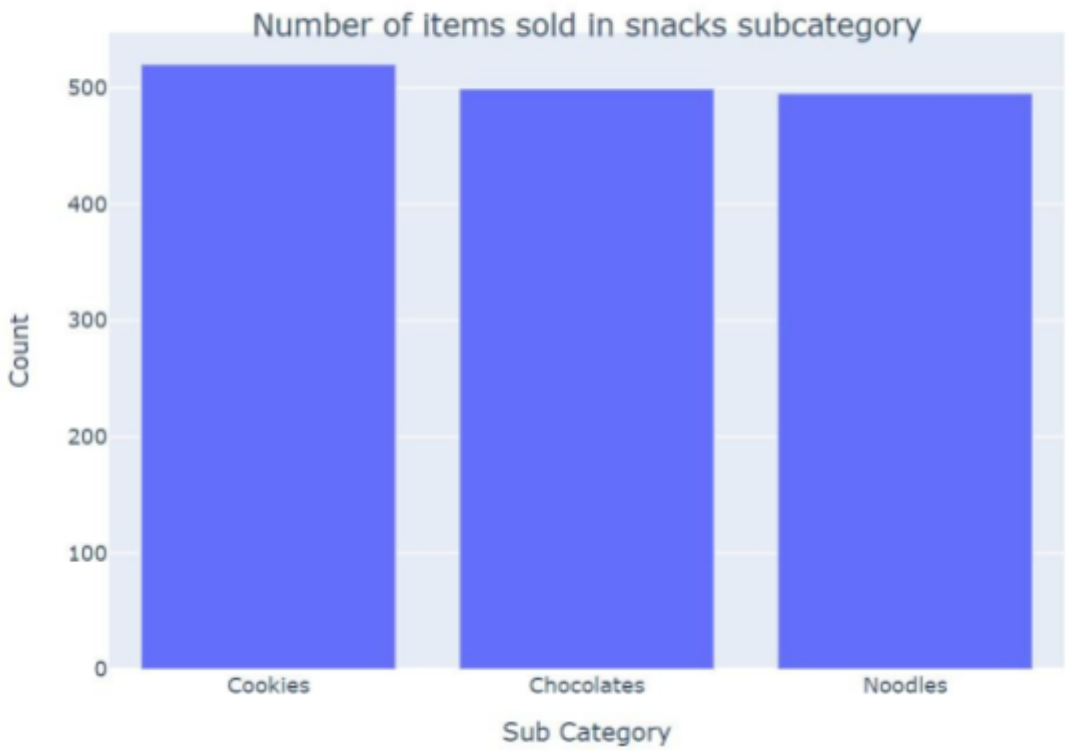
Health Drinks	719
Soft Drinks	681
Cookies	520
Breads & Buns	502
Chocolates	499
Noodles	495
Masalas	463
Biscuits	459
Cakes	452
Edible Oil & Ghee	451
Spices	447
Mutton	394
Eggs	379
Organic Staples	372
Fresh Fruits	369
Fish	369
Fresh Vegetables	354
Atta & Flour	353
Chicken	348
Organic Fruits	348
Organic Vegetables	347
Dals & Pulses	343
Rice	330



```
main_snacks=df.loc[df['category'] == 'Snacks']
```

```
sub = main_snacks['sub_category'].value_counts()
fig = px.bar(sub, x=sub.index, y=sub.values, height=500)
fig.update_layout( title={
        'text': "Number of items sold in snacks subcategory",
        'y':0.9,'x':0.5,'xanchor': 'center','yanchor': 'top'},
    xaxis_title="Sub Category",
    yaxis_title="Count")
print(sub)
fig.show(renderer='svg')
```

```
Cookies       520
Chocolates    499
Noodles       495
```



```
snacks = main_snacks.groupby('order_date').agg({'sales': 'sum'}).reset_index()
```

```
snacks = main_snacks.set_index('order_date')
```

```
snacks.index
```

```
DatetimeIndex(['2019-04-01', '2019-06-01', '2019-06-01', '2019-06-01',
               '2019-01-13', '2019-01-19', '2019-01-26', '2019-01-26',
               '2019-01-28', '2019-02-02',
               ...
               '2022-12-24', '2022-12-25', '2022-12-25', '2022-12-25',
               '2022-12-26', '2022-12-28', '2022-12-28', '2022-12-30',
               '2022-12-30', '2022-12-30'],
              dtype='datetime64[ns]', name='order_date', length=1514, freq=None)
```

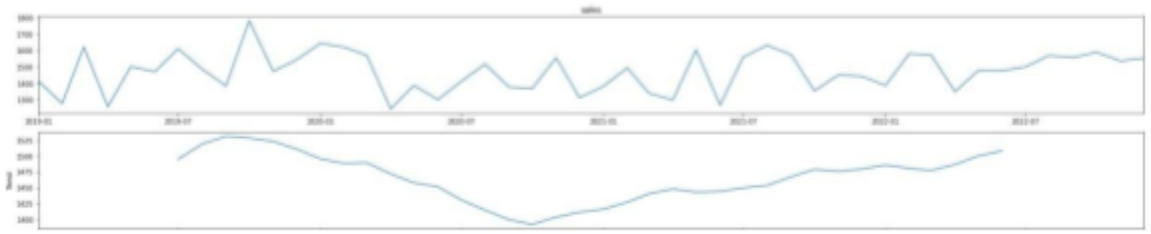
```
snacks
```

order_date	order_id	customer_name	category	sub_category	city	region	sales	discount	profit	state
2019-04-01	OD742	Sudha	Snacks	Chocolates	Dharmapuri	Central	759	0.27	174.57	Tamil Nadu
2019-06-01	OD8635	Hussain	Snacks	Cookies	Pudukottai	South	1504	0.23	616.64	Tamil Nadu
2019-06-01	OD9387	Akash	Snacks	Noodles	Chennai	South	1495	0.19	194.35	Tamil Nadu
2019-06-01	OD9418	Vince	Snacks	Noodles	Karur	South	2157	0.18	258.84	Tamil Nadu
2019-01-13	OD767	Arutra	Snacks	Noodles	Nagercoil	South	1903	0.11	666.05	Tamil Nadu
...	...	...	...	...	...	...	...	...	...	...
2022-12-28	OD271	Esther	Snacks	Noodles	Theni	West	764	0.11	160.44	Tamil Nadu
2022-12-28	OD9124	Peer	Snacks	Chocolates	Trichy	Central	1250	0.13	525.00	Tamil Nadu
2022-12-30	OD907	Veena	Snacks	Chocolates	Tirunelveli	East	1240	0.34	458.80	Tamil Nadu
2022-12-30	OD908	Arvind	Snacks	Cookies	Ooty	East	1854	0.20	685.98	Tamil Nadu

```
avg_snacks_sales = snacks['sales'].resample('MS').mean()
avg_snacks_sales
```

```
order_date
2019-01-01    1414.315789
2019-02-01    1276.437500
2019-03-01    1621.500000
2019-04-01    1257.238095
2019-05-01    1502.136364
2019-06-01    1470.480000
2019-07-01    1610.708333
2019-08-01    1486.666667
2019-09-01    1383.888889
2019-10-01    1781.107143
2019-11-01    1473.190476
2019-12-01    1542.891892
2020-01-01    1643.357143
2020-02-01    1620.142857
2020-03-01    1570.666667
2020-04-01    1243.913043
2020-05-01    1386.789474
2020-06-01    1300.451613
2020-07-01    1410.466667
2020-08-01    1517.130435
2020-09-01    1376.900000
```

```
rcParams['figure.figsize'] = 24, 10
decomposed_model = sm.tsa.seasonal_decompose(avg_snacks_sales, model='additive')
fig = decomposed_model.plot()
plt.show()
```



```
avg_snacks_sales.plot(figsize=(15, 5),color='red',label = 'snacks')
plt.xlabel("Order Date")
plt.ylabel('Sales')
plt.title("Monthly snacks sales")
plt.legend()
plt.show()
```

```python
def performance(y_true, y_pred):
    squared_errors = (y_pred - y_true) ** 2
    mse = squared_errors.mean()
    rmse = np.sqrt(mse)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100

    results = {
        'MSE': round(mse, 2),
        'RMSE': round(rmse, 2),
        'MAPE': round(mape, 2)
    }

    return results
```

Train & Test Split

```python
[ ] train, test = np.array(avg_snacks_sales[:-12]), np.array(avg_snacks_sales[-12:])
    train= train.reshape(-1,1)
    test= test.reshape(-1,1)
```

```python
[ ] scaler = MinMaxScaler()
    scaler.fit(train)
    train_scaled = scaler.transform(train)
    test_scaled = scaler.transform(test)
```

```python
[ ] num_input = 12
    num_features = 1
    generator = TimeseriesGenerator(train_scaled, train_scaled, length=num_input, batch_size=20)
```

Training & Testing Vanilla LSTM Model

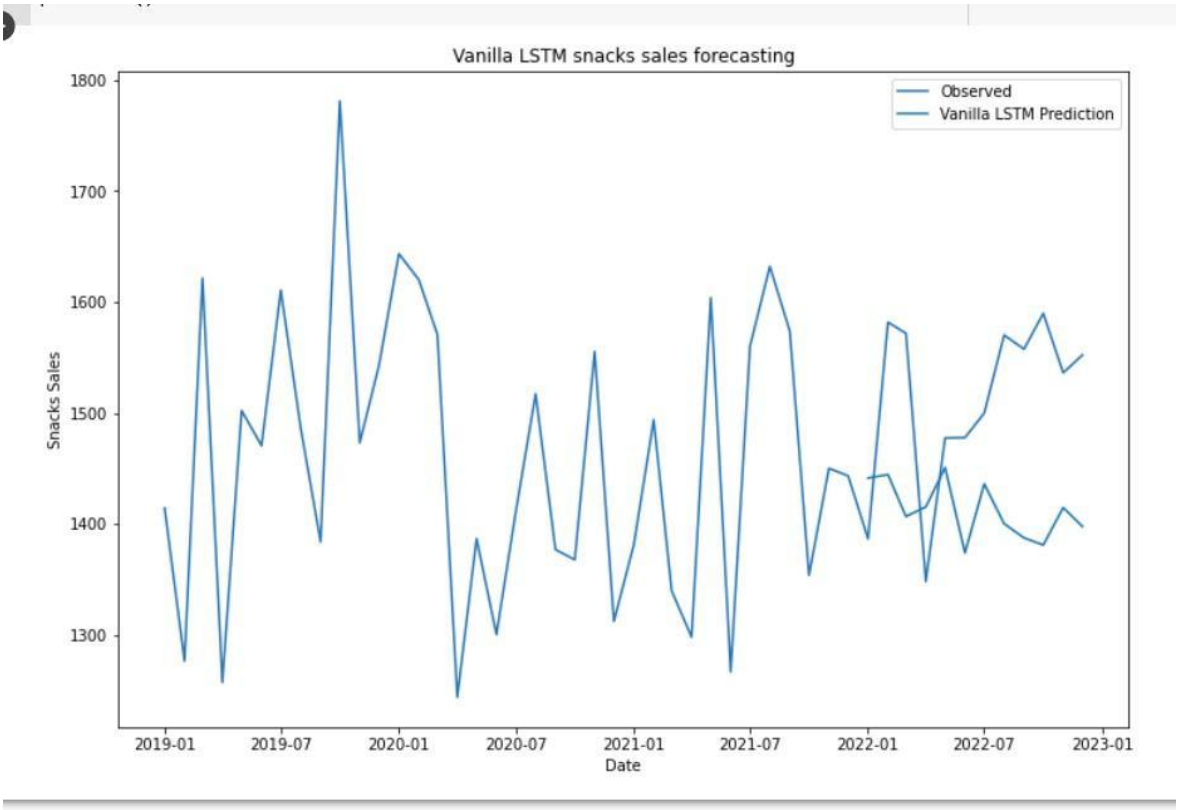
```
n=20
arr1= np.zeros((12,n))
for i in range(n):
    vanilla_model = Sequential()
    vanilla_model.add(LSTM(50, activation='relu', input_shape=(12, 1)))
    vanilla_model.add(Dense(100, activation='relu'))
    vanilla_model.add(Dense(100, activation='relu'))
    vanilla_model.add(Dense(1))
    vanilla_model.compile(optimizer='adam', loss='mse')
    vanilla_model.fit_generator(generator,epochs=200)

    pred_list_v = []

    batch = train_scaled[-num_input:].reshape((1, num_input, num_features))

    for j in range(num_input):
        pred_list_v.append(vanilla_model.predict(batch)[0])
        batch = np.append(batch[:,1:,:],[[pred_list_v[j]]],axis=1)

    df_predicted_vanilla = pd.DataFrame(scaler.inverse_transform(pred_list_v),
                            index=avg_snacks_sales[-num_input:].index, columns=['Prediction'])


    arr1[:,i]=df_predicted_vanilla['Prediction']
print(arr1)
```

```
Epoch 1/200
<ipython-input-25-b9d1f6ed1447>:10: UserWarning:

`Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.

2/2 [==============================] - 4s 30ms/step - loss: 0.1843
Epoch 2/200
```

```
Epoch 21/200
2/2 [==============================] - 0s 20ms/step - loss: 0.0540
Epoch 22/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0537
Epoch 23/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0534
Epoch 24/200
2/2 [==============================] - 0s 28ms/step - loss: 0.0534
Epoch 25/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0536
Epoch 26/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0537
Epoch 27/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0537
```

```
[ ]  final_v= np.zeros((arr1.shape[0],1))
     for i in range(arr1.shape[0]):

         final_v[i]=np.mean(arr1[i,:])
     final_v=final_v.reshape((12,))
```

```
rcParams['figure.figsize'] = 12, 8
plt.plot(avg_snacks_sales.index,avg_snacks_sales,label="Observed",color='#2574BF')
plt.plot(avg_snacks_sales[36:].index,final_v,label="Vanilla LSTM Prediction")
plt.title('Vanilla LSTM snacks sales forecasting')
plt.xlabel('Date')
plt.ylabel('Snacks Sales')
plt.legend()
plt.show()
```




```
vanilla_lstm= performance(avg_snacks_sales[-12:],final_v)
vanilla_lstm
```

```
{'MSE': 17428.6, 'RMSE': 132.02, 'MAPE': 7.83}
```

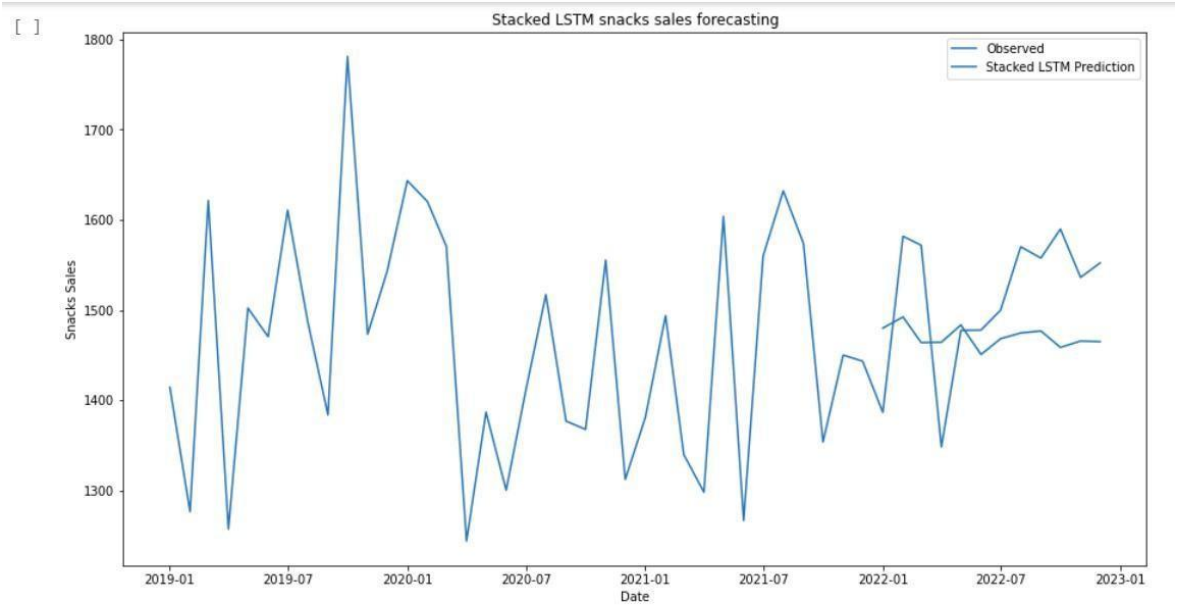
Stacked LSTM Model

```
n=20
arr2= np.zeros((12,n))
for i in range(n):
    stacked_model = Sequential()
    stacked_model.add(LSTM(50, activation='relu', return_sequences = True, input_shape=(12, 1)))
    stacked_model.add(LSTM(50, activation='relu'))
    stacked_model.add(Dense(100, activation='relu'))
    stacked_model.add(Dense(50, activation='relu'))
    stacked_model.add(Dense(1))
    stacked_model.compile(optimizer='adam', loss='mse')
    stacked_model.fit_generator(generator,epochs=200)

    pred_list_stack = []

    batch = train_scaled[-num_input:].reshape((1, num_input, num_features))

    for j in range(num_input):
        pred_list_stack.append(stacked_model.predict(batch)[0])
        batch = np.append(batch[:,1:,:],[[pred_list_stack[j]]],axis=1)

    df_predicted_stacked = pd.DataFrame(scaler.inverse_transform(pred_list_stack),
                            index=avg_snacks_sales[-num_input:].index, columns=['Prediction'])


    arr2[:,i]=df_predicted_stacked['Prediction'].values
print(arr2)
```

```
`Model.fit_generator` is deprecated and will be removed in a future version.

2/2 [==============================] - 5s 23ms/step - loss: 0.1960
Epoch 2/200
2/2 [==============================] - 0s 24ms/step - loss: 0.1769
Epoch 3/200
2/2 [==============================] - 0s 34ms/step - loss: 0.1686
Epoch 4/200
2/2 [==============================] - 0s 24ms/step - loss: 0.1546
Epoch 5/200
2/2 [==============================] - 0s 23ms/step - loss: 0.1325
Epoch 6/200
2/2 [==============================] - 0s 21ms/step - loss: 0.1197
Epoch 7/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0986
Epoch 8/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0707
Epoch 9/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0588
Epoch 10/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0604
Epoch 11/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0752
Epoch 12/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0699
Epoch 13/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0605
Epoch 14/200
2/2 [==============================] - 0s 34ms/step - loss: 0.0579
Epoch 15/200
```

```
final_stack= np.zeros((arr2.shape[0],1))
for i in range(arr2.shape[0]):

    final_stack[i]=np.mean(arr2[i,:])
final_stack=final_stack.reshape((12,))
```

```
rcParams['figure.figsize'] = 15, 8
plt.plot(avg_snacks_sales.index,avg_snacks_sales,label="Observed",color='#2574BF')
plt.plot(avg_snacks_sales[36:].index,final_stack,label="Stacked LSTM Prediction")
plt.title('Stacked LSTM snacks sales forecasting')
plt.xlabel('Date')
plt.ylabel('Snacks Sales')
plt.legend()
plt.show()
```




```
[ ]  stacked_lstm= performance(avg_snacks_sales[-12:],final_stack)
     stacked_lstm
```

```
{'MSE': 7419.42, 'RMSE': 86.14, 'MAPE': 5.16}
```

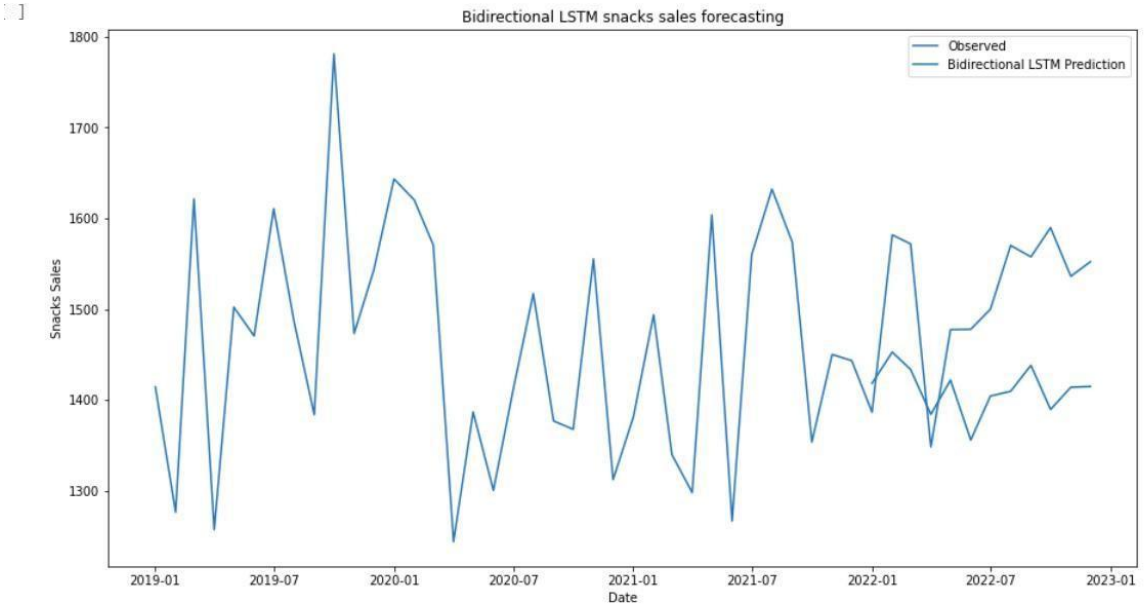
Bi-Directional LSTM Model

```
n=20
arr3= np.zeros((12,n))
for i in range(n):
  bi_model = Sequential()
  bi_model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(12, 1)))
  bi_model.add(Dense(1))
  bi_model.compile(optimizer='adam', loss='mse')
  bi_model.fit_generator(generator,epochs=200)
  pred_list_bi = []

  batch = train_scaled[-num_input:].reshape((1, num_input, num_features))

  for j in range(num_input):
    pred_list_bi.append(bi_model.predict(batch)[0])
    batch = np.append(batch[:,1:,:],[[pred_list_bi[j]]],axis=1)

  df_predicted_bi = pd.DataFrame(scaler.inverse_transform(pred_list_bi),
                          index=avg_snacks_sales[-num_input:].index, columns=['Prediction'])

  arr3[:,i]=df_predicted_bi['Prediction'].values
print(arr3)
```

```
Epoch 1/200
<ipython-input-33-86f24d5a8bbf>:8: UserWarning:

`Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.

2/2 [==============================] - 4s 17ms/step - loss: 0.2888
Epoch 2/200
2/2 [==============================] - 0s 21ms/step - loss: 0.2422
Epoch 3/200
2/2 [==============================] - 0s 20ms/step - loss: 0.2118
```

```
final_bi= np.zeros((arr3.shape[0],1))
for i in range(arr3.shape[0]):

    final_bi[i]=np.mean(arr3[i,:])
final_bi=final_bi.reshape((12,))
```

```
rcParams['figure.figsize'] = 15, 8
plt.plot(avg_snacks_sales.index,avg_snacks_sales,label="Observed",color='#2574BF')
plt.plot(avg_snacks_sales[36:].index,final_bi,label="Bidirectional LSTM Prediction")
plt.title('Bidirectional LSTM snacks sales forecasting')
plt.xlabel('Date')
plt.ylabel('Snacks Sales')
plt.legend()
plt.show()
```

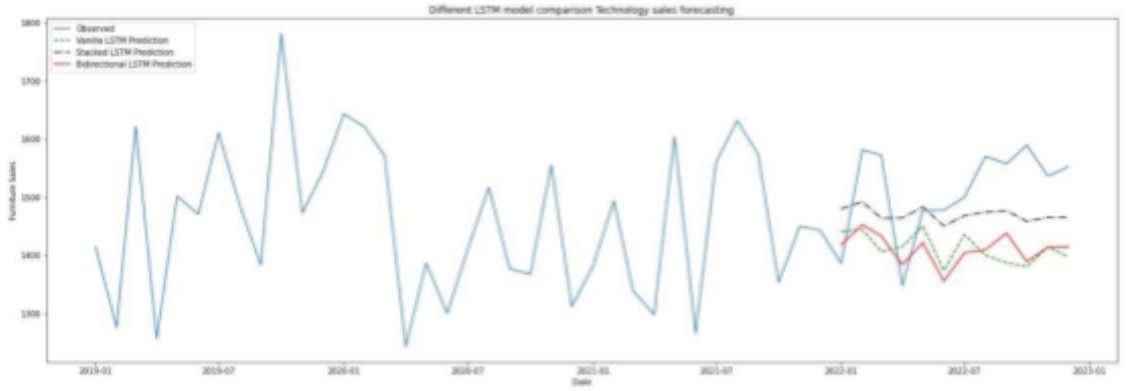



```
bi_lstm= performance(avg_snacks_sales[-12:],final_bi )
bi_lstm
```

```
{'MSE': 14924.49, 'RMSE': 122.17, 'MAPE': 7.3}
```

Model Performance & Evaluation

```
rcParams['figure.figsize'] = 24, 8
plt.plot(avg_snacks_sales.index,avg_snacks_sales,label="Observed",color='#2574BF')
plt.plot(avg_snacks_sales[36:].index,final_v,label="Vanilla LSTM Prediction",linestyle= "--", color='g')
plt.plot(avg_snacks_sales[36:].index,final_stack,label="Stacked LSTM Prediction",linestyle= "-.", color="black")
plt.plot(avg_snacks_sales[36:].index,final_bi,label="Bidirectional LSTM Prediction",marker=',',color='r')
plt.title('Different LSTM model comparison Technology sales forecasting')
plt.xlabel('Date')
plt.ylabel('Furniture Sales')
plt.legend(loc='upper left')
plt.show()
```




```
[ ]  predicted_data = {'Model':['Vanilla LSTM','Stacked LSTM','Bidirectional LSTM'],

                       'MSE':[vanilla_lstm['MSE'],stacked_lstm['MSE'],bi_lstm['MSE']],
                      'RMSE':[vanilla_lstm['RMSE'],stacked_lstm['RMSE'],bi_lstm['RMSE']],
                      'MAPE':[vanilla_lstm['MAPE'],stacked_lstm['MAPE'],bi_lstm['MAPE']]}

     predicted_table = pd.DataFrame(predicted_data)
     predicted_table
```

	Model	MSE	RMSE	MAPE
0	Vanilla LSTM	17428.60	132.02	7.83
1	Stacked LSTM	7419.42	86.14	5.16
2	Bidirectional LSTM	14924.49	122.17	7.30

```
predicted_data2 = {
        'Test Set':avg_snacks_sales[-12:],
        'Vanilla LSTM':final_v,
        'Stacked LSTM':final_stack,
        'Bidirectional LSTM':final_bi
        }

# Create DataFrame
predicted_table2 = pd.DataFrame(predicted_data2)
predicted_table2
```

order_date	Test Set	Vanilla LSTM	Stacked LSTM	Bidirectional LSTM
2022-01-01	1386.555556	1441.352201	1479.949604	1418.365808
2022-02-01	1581.787879	1444.513512	1492.364597	1452.710871
2022-03-01	1571.796296	1406.732038	1463.911766	1433.496983
2022-04-01	1348.348837	1415.357039	1464.347983	1384.155212
2022-05-01	1477.387097	1450.857348	1483.468456	1421.912755
2022-06-01	1477.827586	1373.979675	1450.724112	1355.845774
2022-07-01	1499.848485	1436.101928	1468.242141	1404.347148
2022-08-01	1570.136364	1400.233710	1474.568323	1409.541809
2022-09-01	1557.583333	1387.456291	1476.803037	1438.199088
2022-10-01	1589.738095	1381.001497	1458.617963	1389.529793
2022-11-01	1536.196970	1414.710510	1465.630745	1413.987305
2022-12-01	1552.229167	1397.736300	1464.948676	1414.849802

Forecasting the Future

```
test_scaled
```

```
array([[0.26553254],
       [0.6289623 ],
       [0.61036272],
       [0.19440979],
       [0.43461768],
       [0.43543766],
       [0.47643011],
       [0.60727272],
       [0.58390494],
       [0.64376182],
       [0.5440937 ],
       [0.57393803]])
```

```
n=1
pred_arr= np.zeros((12,n))
for i in range(n):

    test_list_s = []

    batch = test_scaled[-num_input:].reshape((1, num_input, num_features))

    for j in range(num_input):
        test_list_s.append(stacked_model.predict(batch)[0])
        batch = np.append(batch[:,1:,:],[[test_list_s[j]]],axis=1)

    df_forecasted_stacked = pd.DataFrame(scaler.inverse_transform(test_list_s),
                                index=avg_snacks_sales[-num_input:].index, columns=['Prediction'])


    pred_arr[:,i]=df_forecasted_stacked['Prediction']

print(pred_arr)
```

```
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
[[1482.61721298]
 [1445.25510314]
 [1468.75975604]
 [1454.20295035]
 [1416.90092466]
 [1425.22839865]
 [1449.43784356]
 [1489.21902477]
 [1519.96379347]
 [1530.73145553]
 [1534.50976063]
 [1531.70717854]]
```

```
[ ]  final_prediction= np.zeros((pred_arr.shape[0],1))
     for i in range(pred_arr.shape[0]):

         final_prediction[i]=np.mean(pred_arr[i,:])
     final_prediction=final_prediction.reshape((12,))
```

```
[ ]  final_prediction
```

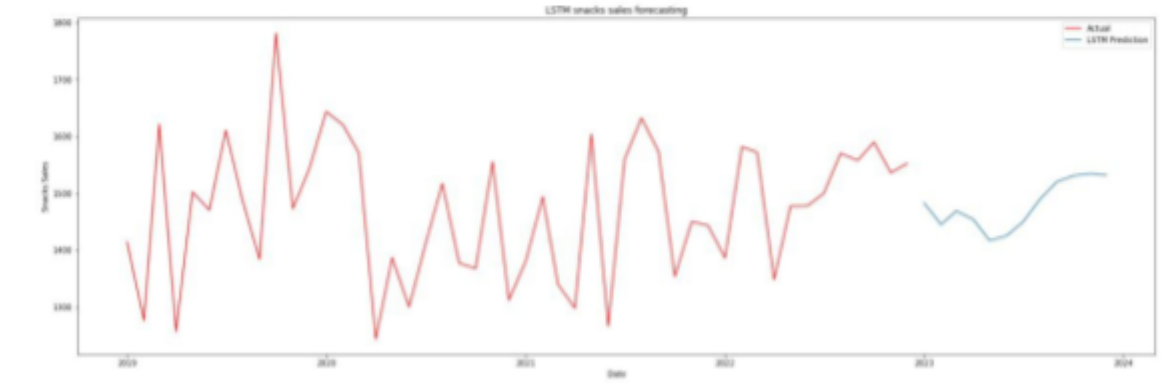
```
array([1482.61721298, 1445.25510314, 1468.75975604, 1454.20295035,
       1416.90092466, 1425.22839865, 1449.43784356, 1489.21902477,
       1519.96379347, 1530.73145553, 1534.50976063, 1531.70717854])
```

```
new_dates = pd.date_range('2019-01-01', periods=60, freq='MS')
avg_snacks_sales = avg_snacks_sales.reindex(new_dates)
avg_snacks_sales
```

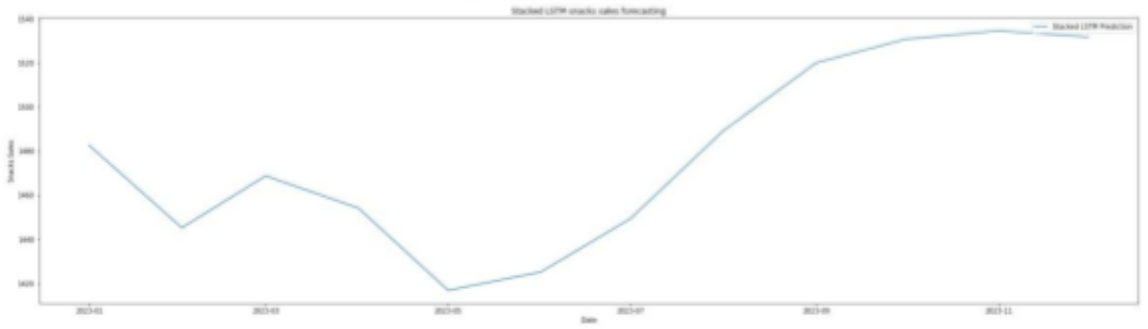
```
2019-01-01    1414.315789
2019-02-01    1276.437500
2019-03-01    1621.500000
2019-04-01    1257.238095
2019-05-01    1502.136364
2019-06-01    1470.480000
2019-07-01    1610.708333
2019-08-01    1486.666667
2019-09-01    1383.888889
2019-10-01    1781.107143
2019-11-01    1473.190476
2019-12-01    1542.891892
```

```
2022-10-01    1589.738095
2022-11-01    1536.196970
2022-12-01    1552.229167
2023-01-01            NaN
2023-02-01            NaN
2023-03-01            NaN
2023-04-01            NaN
2023-05-01            NaN
2023-06-01            NaN
2023-07-01            NaN
2023-08-01            NaN
2023-09-01            NaN
2023-10-01            NaN
```

```
[ ]  rcParams['figure.figsize'] = 25, 8
     plt.plot(avg_snacks_sales.index,avg_snacks_sales,label="Actual",color='red')
     plt.plot(avg_snacks_sales[48:].index,final_prediction,label="LSTM Prediction")
     plt.title('LSTM snacks sales forecasting')
     plt.xlabel('Date')
     plt.ylabel('Snacks Sales')
     plt.legend()
     plt.show()
```




```
rcParams['figure.figsize'] = 30, 8
plt.plot(avg_snacks_sales[48:].index,final_prediction,label="Stacked LSTM Prediction")
plt.title('Stacked LSTM snacks sales forecasting')
plt.xlabel('Date')
plt.ylabel('Snacks Sales')
plt.legend()
plt.show()
```



CHAPTER 6

RESULTS AND OBSERVATIONS

In this study, sales data were analysed using three different deep learning neural network models: Vanilla Long Short-Term Memory (LSTM), Stacked LSTM, and Bidirectional LSTM. The objective was to predict sales for a dataset from a supermarket, specifically for the category of Snacks. Calculating the average monthly sales, the dataset was split into 25% for validation testing and 75% for training. Following the definition of a time-series generator, the three LSTM models were constructed and fitted using the generator. The Stacked LSTM model demonstrated superior performance compared to the Vanilla LSTM and Bidirectional LSTM models in terms of lower MSE, RMSE and MAPE. During the following 12 months, sales were predicted using the Stacked LSTM model.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

Conclusion: While LSTM models have shown improved performance compared to ARIMA, SARIMA, and RNN models, they can be computationally demanding when dealing with large and complex datasets. So, therefore SARIMA can be used for these datasets, particularly if the data is seasonal. It is recommended to terminate training as soon as a respectable level of accuracy is attained because the number of epochs used did not consistently affect accuracy.

Future Scope: The more time that has passed since the last data point, the less accurate the future forecasts become. Future research may try alternative deep learning models, such GRU, and, depending on the data, may combine stochastic and DL models. Businesses can also create a web- or mobile-based application to aid with sales forecasting decisions.

REFERENCES

[1] Muhammad A. U, Yahaya A. S, Kamal S. M, Adam J. M, Muhammad W. I and Elsafi A, "A Hybrid Deep Stacked LSTM and GRU for Water Price Prediction," 2020 2nd International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2020, pp. 1-6, doi: 10.1109/ICCIS49240.2020.9257651.

[2] G. Cui, D. Song, S. Yu and Y. You, "Medium-term load prediction based on GALightGBM- LSTM," 2022 9th International Forum on Electrical Engineering and Automation (IFEEA), Zhuhai, China, 2022, pp. 1-5, Doi:10.1109/IFEEA57288.2022.10038071.

[3] M. Masum, H. Shahriar, H. M. Haddad and M. S. Alam, "r-LSTM: Time Series Forecasting for COVID-19 Confirmed Cases with LSTMbased Framework," 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 1374-1379, doi: 10.1109/BigData50022.2020.9378276.

[4] N. Kapali, T. Tuhin, A. Pramanik, M. S. Rahman and S. R. H. Noori, "Sentiment Analysis of Facebook and YouTube Bengali Comments Using LSTM and Bi-LSTM," 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2022, pp. 1-6, doi: 10.1109/ICCCNT54827.2022.9984395.

[5] Y. Li and Y. Lu, "LSTM-BA: DDoS Detection Approach Combining LSTM and Bayes," 2019 Seventh International Conference on Advanced Cloud and Big Data (CBD), Suzhou, China, 2019, pp. 180-185, doi: 10.1109/CBD.2019.00041.

[6] M. A. Istiake Sunny, M. M. S. Maswood and A. G. Alharbi, "Deep Learning-Based Stock Price Prediction Using LSTM and Bi-Directional LSTM Model," 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), Giza, Egypt, 2020, pp. 87-92, doi: 10.1109/NILES50944.2020.9257950.

[7] A. Joshi, P. K. Deshmukh and J. Lohokare, "Comparative analysis of Vanilla LSTM and Peephole LSTM for stock market price prediction," 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India, 2022, pp. 1-6, doi: 10.1109/IC3SIS54991.2022.9885528.

[8] N. Daoud, M. Eltahan and A. Elhennawi, "Aerosol Optical Depth Forecast over Global Dust Belt Based on LSTM, CNN-LSTM, CONV-LSTM and FFT Algorithms," IEEE EUROCON 2021 - 19th International Conference on Smart Technologies, Lviv, Ukraine, 2021, pp. 186-191, doi: 10.1109/EUROCON52738.2021.9535571.

[9] B. J. T and D. S. Misbha, "Detection of Attacks using Attention-based Conv-LSTM and Bi-LSTM in Industrial Internet of Things," 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2022, pp. 402-407, doi: 10.1109/ICACRS55517.2022.10029012.

[10] G. S. Mahara and S. Gangele, "Fake news detection: A RNN-LSTM, Bi-LSTM based deep learning approach," 2022 IEEE 1st International Conference on Data, Decision and Systems (ICDDS), Bangalore, India, 2022, pp. 01-06,

doi:10.1109/ICDDS56399.2022.10037403.

[11] Biswajit Biswas, Manas Kumar Sanyal, Tuhin Mukherjee, "AI-Based Sales Forecasting Model for Digital Marketing", International Journal of E-Business Research, vol.19, no.1, pp.1, 2023.

[12] S. Montaha, S. Azam, A. K. M. R. H. Rafid, M. Z. Hasan, A. Karim and A. Islam, "TimeDistributed-CNN-LSTM: A Hybrid Approach Combining CNN and LSTM to Classify Brain Tumor on 3D MRI Scans Performing Ablation Study," in IEEE Access, vol. 10, pp. 60039-60059, 2022, doi: 10.1109/ACCESS.2022.3179577.

[13] X. Wang and Y. Zhang, "Multi-Step-Ahead Time Series Prediction Method with Stacking LSTM Neural Network," 2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD), Chengdu, China, 2020, pp. 51-55, doi: 10.1109/ICAIBD49809.2020.9137492.

[14] S. K. Verma, A. Gupta and A. Jyoti, "Stack layer & Bidirectional Layer Long Short -Term Memory (LSTM) Time Series Model with Intermediate Variable for weather Prediction," 2021 International Conference on Computational Performance Evaluation (ComPE),Shillong, 2021, pp. 065-070, doi: 10.1109/ComPE53109.2021.9752357.

[15] A. Joshi, P. K. Deshmukh and J. Lohokare, "Comparative analysis of Vanilla LSTM and Peephole LSTM for stock market price prediction," 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India, 2022, pp. 1-6, doi: 10.1109/IC3SIS54991.2022.9885528.

APPENDIX A

CONFERENCE PRESENTATION

Our paper IRCICD_2023_25 on Time-Series forecasting in Retail Industry using Bi-LSTM, Stacked LSTM and Vanilla LSTM was presented at IRCICD 2023 conference held at SRMIST Vadapalani Campus. 200+ shortlisted teams presented their papers on various fields in the conference. Our paper got accepted with a plagiarism of just 6 %.

On presenting the paper in this international conference held at virtually campus, we received positive remarks and suggestions from the judging panel.

International Research Conference on IoT, Cloud and Data Science (IRCICD'23)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY VADAPALANI

Certificate of Participation

This is to certify that

Mr./Ms./Dr. Lekhashree V

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, VADAPALANI

has participated and presented a paper titled

Time - Series Forecasting in Retail Industry using Bidirectional, Stacked and Vanilla LSTM

in the International Research Conference on IoT, Cloud and Data Science (IRCICD'23), organised by the Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani, Chennai, Tamil Nadu, India held on 28th & 29th April 2023.

Dr. G. Paavai Anand, Organizing Secretary
Dr. P. Durgadevi, Organizing Secretary
Dr. S. Prasanna Devi, HOD- CSE and Convener
Dr. C . Gomathy, VP (Academics and Placements)
Dr. C. V. Jayakumar, Dean (CET) - VDP

International Research Conference on IoT, Cloud and Data Science (IRCICD'23)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY VADAPALANI

Certificate of Participation

This is to certify that

Mr./Ms./Dr. S. Manohar

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, VADAPALANI

has participated and presented a paper titled

Time - Series Forecasting in Retail Industry using Bidirectional, Stacked and Vanilla LSTM

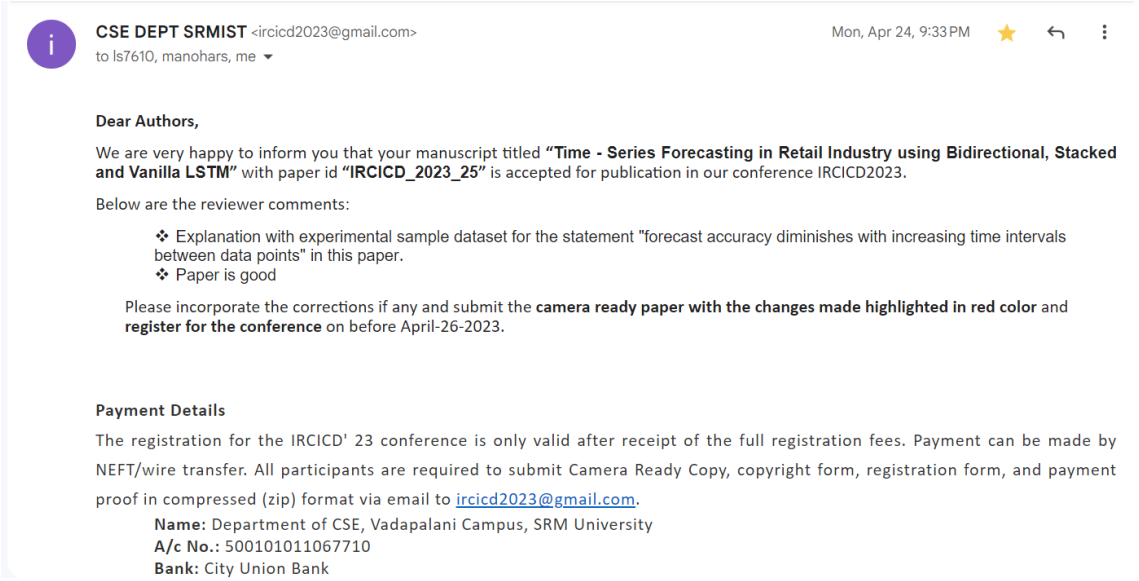
in the International Research Conference on IoT, Cloud and Data Science (IRCICD'23), organised by the Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani, Chennai, Tamil Nadu, India held on 28th & 29th April 2023.

Dr. G. Paavai Anand, Organizing Secretary
Dr. P. Durgadevi, Organizing Secretary
Dr. S. Prasanna Devi, HOD- CSE and Convener
Dr. C . Gomathy, VP (Academics and Placements)
Dr. C. V. Jayakumar, Dean (CET) - VDP

APPENDIX B

PUBLICATION DETAILS

We submitted our research paper for publication at ACM conference proceedings for a Scopus Indexed Journal.



CSE DEPT SRMIST <ircicd2023@gmail.com> Mon, Apr 24, 9:33 PM

to ls7610, manohars, me

Dear Authors,

We are very happy to inform you that your manuscript titled **"Time - Series Forecasting in Retail Industry using Bidirectional, Stacked and Vanilla LSTM"** with paper id **"IRCICD_2023_25"** is accepted for publication in our conference IRCICD2023.

Below are the reviewer comments:

- Explanation with experimental sample dataset for the statement "forecast accuracy diminishes with increasing time intervals between data points" in this paper.
- Paper is good

Please incorporate the corrections if any and submit the **camera ready paper with the changes made highlighted in red color** and **register for the conference** on before April-26-2023.

Payment Details

The registration for the IRCICD' 23 conference is only valid after receipt of the full registration fees. Payment can be made by NEFT/wire transfer. All participants are required to submit Camera Ready Copy, copyright form, registration form, and payment proof in compressed (zip) format via email to ircicd2023@gmail.com.

Name: Department of CSE, Vadapalani Campus, SRM University
A/c No.: 500101011067710
Bank: City Union Bank

D

To Department of CSE, Vadapalani Campu...

₹10,000

IRCICD_2023_25

Pay again | Split with friends

Completed

Apr 26, 2023 3:45 PM

ICICI Bank 7231

UPI transaction ID
311634690612

To
•••• 7710

From: K SRINIVASAN (ICICI Bank)
srinivasankarum@okicici

Google transaction ID
CICAgJCfl-uGPw

Powered by UPI

G Pay

APPENDIX C

PLAGIARISM REPORT

BATCH9_REPORT.pdf

ORIGINALITY REPORT

9%	5%	5%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Liverpool John Moores University Student Paper	1%
2	www.mdpi.com Internet Source	1%
3	www.researchgate.net Internet Source	1%
4	Submitted to Middle East College of Information Technology Student Paper	1%
5	Submitted to Georgetown University Student Paper	1%
6	Kaushal Kumar. "Reservoir Computing in Epidemiological Forecasting: Predicting Chicken Pox incidence", Cold Spring Harbor Laboratory, 2023 Publication	<1%
7	openxmldeveloper.org Internet Source	<1%
	Submitted to CSU, San Jose State University	

8	Student Paper	<1%
9	Submitted to University of Ulster Student Paper	<1%
10	link.springer.com Internet Source	<1%
11	Submitted to University of Monastir Student Paper	<1%
12	www.frontiersin.org Internet Source	<1%
13	Submitted to University of South Australia Student Paper	<1%
14	www.ijraset.com Internet Source	<1%
15	www.biorxiv.org Internet Source	<1%
16	Submitted to CSU, San Diego State University Student Paper	<1%
17	arxiv.org Internet Source	<1%
18	iopscience.iop.org Internet Source	<1%
19	Binh Nguyen, Yves Coelho, Teodiano Bastos, Sridhar Krishnan. "Trends in human activity	<1%

	recognition with focus on machine learning and power requirements", Machine Learning with Applications, 2021 Publication	
20	Zonglei Chen, Minbo Ma, Tianrui Li, Hongjun Wang, Chongshou Li. "Long sequence time-series forecasting with deep learning: A survey", Information Fusion, 2023 Publication	<1 %
21	Submitted to Aston University Student Paper	<1 %
22	Submitted to University of Glasgow Student Paper	<1 %
23	Submitted to University of York Student Paper	<1 %
24	Chun Yan, Jiahui Liu, Wei Liu, Xinhong Liu. "Research on public opinion sentiment classification based on attention parallel dual-channel deep learning hybrid model", Engineering Applications of Artificial Intelligence, 2022 Publication	<1 %
25	ijsrcseit.com Internet Source	<1 %
26	journalofbigdata.springeropen.com Internet Source	<1 %

27	prg.is.titech.ac.jp Internet Source	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		