**VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY**

**UNIVERSITY OF INFORMATION TECHNOLOGY**

**INFORMATION SYSTEM FACULTY**

Ảnh có chứa biểu tượng, Đồ họa, hình mẫu, Phông chữ

Mô tả được tạo tự động

**FINAL REPORT**

**BUSINESS ANALYSIS**

**Lecturer:** [Nguyen Dinh Thuan](https://courses.uit.edu.vn/user/view.php?id=490&course=1)

**Instructor:** Nguyen Minh Nhut

Nguyen Thi Viet Huong

**Student performance:** Team 15

Nguyen Cam Tu 20520837

Nguyen Thi Thu Thuy 20520797

Nguyen Thanh Son 20521847

**HO CHI MINH CITY, JUNE 2023**

# TEACHER’S COMMENTS

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

# TABLE OF CONTENTS

[TEACHER’S COMMENTS 2](#_Toc138234000)

[TABLE OF CONTENTS 3](#_Toc138234001)

[TABLE OF FIGURES 5](#_Toc138234002)

[TABLE OF TABLES 6](#_Toc138234003)

[I. INTRODUCTION 7](#_Toc138234004)

[II. RELATED WORK 8](#_Toc138234005)

[III. MODELING 10](#_Toc138234006)

[3.1. DATA PREPARATION 10](#_Toc138234007)

[3.2. RNN 10](#_Toc138234008)

[3.3. LSTM 11](#_Toc138234009)

[3.4. DNN 11](#_Toc138234010)

[3.5. ARIMA 12](#_Toc138234011)

[3.6. ARIMAX 12](#_Toc138234012)

[3.7. KNN 13](#_Toc138234013)

[3.8. LR 13](#_Toc138234014)

[3.9. TIME SERIES CLUSTERING 13](#_Toc138234015)

[3.10. TIME SERIES ANOMALY DETECTION 13](#_Toc138234016)

[IV. METHODOLOGY 14](#_Toc138234017)

[4.1. SINGLE MODEL 14](#_Toc138234018)

[4.1.1 RNN 14](#_Toc138234019)

[4.1.2 LSTM 14](#_Toc138234020)

[4.1.3 DNN 15](#_Toc138234021)

[4.1.4 ARIMA 17](#_Toc138234022)

[4.1.5 ARIMAX 18](#_Toc138234023)

[4.1.6 LR 19](#_Toc138234024)

[4.2. HYBRID MODEL 20](#_Toc138234025)

[4.2.1. Hybrid model based on TIME SERIES ANOMALY DETECTION with ARIMA 20](#_Toc138234026)

[4.2.2. Hybrid model based on TIME SERIES CLUSTERING with RNN, LSTM, DNN 21](#_Toc138234027)

[4.2.3. Hybrid model based on KNN with 50](#_Toc138234028)

[V. RESULT 51](#_Toc138234029)

[5.1. Materials 51](#_Toc138234030)

[5.2. Tools used 51](#_Toc138234031)

[5.3. Evaluation 51](#_Toc138234032)

[5.3.1. MAE (Mean Absolute Error) 51](#_Toc138234033)

[5.3.2. RMSE (Root Mean Square Error) 52](#_Toc138234034)

[5.3.3. MAPE (Mean Absolute Percentage Error) 53](#_Toc138234035)

[5.4. Result 54](#_Toc138234036)

[5.4.1 SINGLE MODEL 54](#_Toc138234037)

[5.4.2 HYBRID MODEL 58](#_Toc138234038)

[5.5. Predict the price of cryptocurrencies 61](#_Toc138234039)

[5.5.1 SINGLE MODEL 61](#_Toc138234040)

[5.5.2 HYBRID MODEL 63](#_Toc138234041)

[VI. CONCLUSION 65](#_Toc138234042)

[REFERENCES 66](#_Toc138234043)

# TABLE OF FIGURES

[FIGURE 1. The architecture of LSTM 14](https://uithcm-my.sharepoint.com/personal/20520797_ms_uit_edu_vn/Documents/TEAM15_FINAL_REPORT.docx#_Toc138234044)

[FIGURE 2. Deep feedforward neural network. 16](#_Toc138234045)

[FIGURE 3. Flow of information in an artificial neuron. 17](#_Toc138234046)

[FIGURE 4. Scatter chart 19](#_Toc138234047)

[FIGURE 5. An example isolation tree 20](#_Toc138234048)

[FIGURE 6. Formula for calculating MAE 52](#_Toc138234049)

[FIGURE 7. Graphical description of the MAE 52](#_Toc138234050)

[FIGURE 8. Formula for calculating RMSE 53](#_Toc138234051)

[FIGURE 9. Graphical description of the RMSE 53](#_Toc138234052)

[FIGURE 10. Formula for calculating MAPE 54](#_Toc138234053)

[FIGURE 11. Graphical description of the MAPE 54](#_Toc138234054)

# TABLE OF TABLES

[Table 1. Data description table 10](#_Toc138234103)

[TABLE 2. Descriptive statistics table 51](#_Toc138234104)

# INTRODUCTION

# RELATED WORK

Yongqiong Zhu [1] used RNNs to predict stocks price and admit that RNN is very suitable for predicting stocks. Gabor Petnehazi [2] used RNN Network and LSTM Network for time series forcasting. The objective of this study was to investigate and elucidate various facets of employing RNNs for the purpose of time series forecasting. This idea is significant as it allows us to build neural networks that can process time-dependent values. Benjamin Lindemann, Timo Müller, Hannes Vietz,Nasser Jazdi, Michael Weyrich [3] did a survey on long short-term memory networks for time series prediction . They give an overview of LSTM architectures that are developed to predict nonlinear time series behavior.

Aysun Bozantaa, Sean Berrya, Mucahit Cevika, Beste Bulutb, Deniz Yigitb, Fahrettin F. Gonenb, and Ayse Basara [4] investigated alternative clustering methodologies to group the products based on the price patterns and sales volumes. They found that our proposed clustering approach and image clustering both perform well for finding the products with similar price and sales patterns within large datasets. Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah [5] used the K-Means clustering algorithm to partition a set of unlabelled objects into k clusters, ensuring that each cluster contains at least one object. Ali Javed, Byung Suk Lee, and Donna M Rizzo [6] conducted a benchmark time series clustering study using the UCR archive datasets. They combined k-means and density-based clustering algorithms with Euclidean and DTW distances. Francisco Martinez Alvarez, Alicia Troncoso, Jose C. Riquelme, and Jesus S. Aguilar Ruiz [7] used K-means clustering to group and label time series data on electricity demand. Before performing time series forecasting, they employed K-means clustering to group and label the time series data. One of the measures they used to evaluate the performance of the clustering algorithm is by using the silhouette index. Afterwards, they proceeded to predict electricity prices of Spanish, Australian, and New York markets using various forecasting algorithms such as ARIMA, KNN, SVM….

Yuhan Li[9] conducted research on predicting stock prices using multiple regression models. The paper focuses on applying the multiple regression method to identify and measure the relationship between the dependent variable (stock price) and multiple independent variables (financial indicators, market data, and other relevant factors).

Mojtaba Nabipour, Pooyan Nayyeri, Hamed Jabani, and Amir Mosavi[10] focus on the application of deep learning in stock market prediction. They explain how deep learning is used to forecast stock prices and market fluctuations. The paper provides an overview of popular deep learning architectures and discusses the advantages and limitations of deep learning. It also addresses potential directions for future development in this field.

Cyril Bachelard, Apostolos Chalkis, Vissarion Fisikopoulos, and Elias Tsigaridas from the Faculty of Business and Economics (HEC) at the University of Lausanne, Switzerland, and the National & Kapodistrian University of Athens, Greece, along with GeomScale.org, Inria Paris, and IMJ-PRG at Sorbonne Universite and Paris Universite[11] focus on applying randomized geometric tools for anomaly detection in the stock market. These tools are used to analyze market data and identify unusual or outlier patterns. The paper introduces specific methods and algorithms, emphasizing the significant differences and deviations from conventional models. It also examines the effectiveness and advantages of randomized geometric methods and discusses potential directions for future development in this field.

Matthew Chen, Neha Narwal and Mila Schultz [[16]](https://www.zotero.org/google-docs/?2UY4BR) acknowledged the best performance was achieved by the Automated Regression Integrated Moving Average (ARIMA) model, attributed to its features and relevance with time series data with accuracy 61,17%.

Richard M. N. Y. Sarpong-Streetor, Rajalingam Sokkalingam , Mahmod Othman , Hanita Daud , and Derrick Asamoah Owusu [[17]](https://www.zotero.org/google-docs/?2FUpHS) has examined the ability of ARIMAX to model the fuel price of Ron97 using times series data of Ron97 and another exogenous time series, the crude oil price, in this case OPEC in Malaysia. It is possible to do the modelling and forecasting accurately using the ARIMAX. The findings demonstrate that incorporating external variables improves the accuracy of the ARIMAX model compared to ARIMA.

Saad Ali Alahmari [18] compared and assesses tow machine-learning methods (DTR and KNN) for predicting cryptocurrency using the datasets of three-pricemajor cryptocurrencies: Bitcoin, XRP and Ethereum. The dependent variable is time series continuous and predictor variables are all continuous. Many studies have been performed in the past experimental, however, very few compared more than one model such as DRT and KNN with several cryptocurrencies. It is remarkable that the Decision Tree Regressor (DTR) outperforms the K-Nearest Neighbor (KNN) in terms of MAE, MSE, RMSE.

# MODELING

## DATA PREPARATION

We use 3 dataset files TRX-USDT, DOGE-USD and XLM-USD

Link dataset: [Crypto Real Time Prices & Latest News - Yahoo Finance](https://finance.yahoo.com/crypto/)

Start date: 01/12/2017

End date: 10/06/2023

|  |  |
| --- | --- |
| **Attribute** | **Describe** |
| Date | This timestamp is converted to NYE Standard Time |
| Open | The opening price of the time period |
| High | The highest price of the time period |
| Low | The lowest price of the time period |
| Close | The closing price of the time period |
| Adj Close | The closing price after adjustment for all current dividend splits and mixes |
| Volume | The volume in the transacted Ccy.le.For BTC/USDT, this is in BTC amount |

Table 1. Data description table

## RNN

The origin of RNN (Recurrent Neural Network) can be traced back to the research efforts of scientists in the field of artificial intelligence and machine learning. In the neural network revolution, Paul Werbos made a significant contribution to the development of RNN through his research work in 1974. In his doctoral thesis, Werbos introduced the concept of recurrent neural networks and the backpropagation algorithm for training such networks.

However, the practical interest and development of RNN gained momentum after the publication of a research paper by Ronald J. Williams and David Zipser in 1989. In their paper titled "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," Williams and Zipser introduced the concept of reinforcement learning for RNN, allowing the neural network to maintain hidden states and learn through continuous interaction with the environment.

The origin and development of RNN reflect the research objectives of scientists in simulating the ability to process sequential data with temporal relationships, similar to human cognition. RNN was designed to address challenges associated with sequential data such as natural language, time series, and sequential data.

With the ability to maintain hidden states and utilize historical information, RNN enables models to learn the structure and complex relationships within sequential data, thereby making predictions and generating output based on historical context. RNN algorithms and architectures, such as LSTM and GRU, continue to evolve and improve the performance of the models.

In conclusion, RNN emerged as a result of the research interests and efforts of scientists in studying and developing models capable of processing sequential data with temporal relationships. It originated from Werbos' work and was further improved by Williams and Zipser, receiving continued attention and development from the research community in the field of artificial intelligence and deep learning.

## LSTM

LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997. LSTM was developed to address the "vanishing gradient" problem of traditional RNNs, where the gradient approaches zero during backpropagation. This makes it difficult to retain and store information over long sequences.

LSTM solves this issue by using gated mechanisms to regulate the process of information storage and retrieval within the network. Specifically, LSTM employs three main gates: the forget gate, the input gate, and the output gate. These gates enable LSTM to control the flow of information through time steps.

In 1997, Hochreiter and Schmidhuber published the paper "Long Short-Term Memory", which presented the LSTM architecture and its functioning. This paper opened up a new research direction in the field of recurrent neural networks and became a significant foundation for the development of LSTM in practical applications.

## DNN

DNN (Deep Neural Network) originates from the theoretical foundation of reinforcement learning and multilayer neural networks in research from the 1940s and 1950s. However, the true development of DNN has occurred in recent years, thanks to advancements in computing technology and parallel computing capabilities, such as GPU clusters, as well as the availability of large and powerful datasets for model training.

DNN operates by utilizing a deep neural network with multiple hidden layers to learn and extract information from input data. Each hidden layer in DNN consists of a number of interconnected neurons with associated weights. These weights are adjusted during the training process so that the model can learn and represent complex patterns and features of the data.

DNN has achieved impressive results in various fields such as image recognition, speech recognition, natural language processing, and classification and prediction tasks. Due to its powerful performance and the ability to automate the learning process, DNN has become an important tool in machine learning and finds wide applications in many different domains.

## ARIMA

Autoregressive integrated moving average (ARIMA) models are models which can be fitted to a single time series and used to make predictions of future observations. They owe their popularity primarily to the work of Box and Jenkins (1970), who defined the class of ARIMA and seasonal ARIMA models and provided a methodology for selecting a suitable model from that class.

An ARIMA model is composed of 3 constituent units which are :

* **AR**: ***Autoregression***. This part explores any dependent relationship between an observation and some number of lagged variables.
* **I**: ***Integrated***. This part aims to make time-series stationery by subtracting or differencing an observation from observation at the previous time step of the same time series.
* **MA**: ***Moving Average*.** This part explores the relationship between an observation and a residual error by application of moving average to lagged observations, with any given time window.

## ARIMAX

ARIMAX stands for AutoRegressive Integrated Moving Average with eXogenous variables.

The name ARIMAX is as extension of the ARIMA respectively. The X added to the end stands for “exogenous”. In other words, it suggests adding a separate different outside variable to help measure our endogenous variable.

An exogenous variable is a variable that is not affected by other variables in a model but can affect the dependent variable. Some examples of exogenous variables include income tax rates, weather, the presence of pests, and interest rates.

These variables are considered exogenous because they are not influenced by other variables in the model. For example, the amount of rainfall cannot be affected by the amount of fertilizer used or the type of soil used in a crop yield model.

## KNN

The k-nearest neighbors (KNN) algorithm was first introduced in 1951 by Fix and Hodges for non-parametric discriminant analysis. Since then, it has been widely used in pattern recognition and machine learning applications.

## LR

Linear regression algorithm has long existed in the field of statistics and regression. It is one of the simplest and fundamental algorithms in machine learning and has been developed since the early 19th century. The linear regression algorithm works by finding the best line (or hyperplane) to model the linear relationship between input variables and output values. It uses the method of least squares, where the parameters are adjusted to minimize the error between the predicted values and the actual values. Linear regression is used for several reasons. First, it is a simple and understandable algorithm that does not require complex knowledge. Second, the computation of linear regression is efficient and fast, making it suitable for handling large datasets. Third, it provides the ability to make predictions based on linear relationships, which can sometimes yield good approximations and be easily interpretable. However, linear regression also has limitations and cannot model complex nonlinear relationships. In such cases, other machine learning methods may be preferred.

## TIME SERIES CLUSTERING

The origin of time series clustering can be traced back to various fields and sources. Here are some important origins of time series clustering:

* Techniques in time series clustering: The techniques for clustering time series have been studied and developed in the fields of statistics and data mining. Methods such as K-means clustering, hierarchical clustering, density-based clustering, and model-based clustering have been applied for grouping time series.
* Macrostatistics: Time series clustering is also related to the field of macrostatistics and economics. Clustering time series in this domain helps analyze and understand the variations and correlations among economic variables over time.
* Artificial intelligence and data mining: In recent years, the advancements in artificial intelligence and data mining have contributed to the development of time series clustering. Methods such as neural network-based clustering, unsupervised learning techniques, and specialized data mining techniques for time series have been applied for clustering and analyzing time series data.
* Applications: The need for clustering time series has emerged from various application domains such as finance, healthcare, information technology, power systems, and many others. These applications have contributed to the expansion and development of time series clustering.

In summary, time series clustering has origins in multiple fields such as statistics, macrostatistics, artificial intelligence, data mining, and real-world applications. The combination of these origins has led to the development of modern methods and techniques for clustering time series.

## TIME SERIES ANOMALY DETECTION

The Time series anomaly detection algorithm has existed for a long time in the field of data analysis and forecasting. However, with the advancement of technology and the need to detect anomalous events in time series data, this algorithm has become increasingly popular and studied more deeply. The Time series anomaly detection algorithm works by identifying data points in a time series that do not follow the usual pattern or behavior. To do this, the algorithm uses methods such as statistical analysis, machine learning, and neural networks. Models are built based on historical data to estimate patterns and model the normal variations in the time series. When new data is available, the algorithm compares it to the learned model to determine if the data is anomalous or not. The Time series anomaly detection algorithm plays an important role in detecting anomalies in time series data, improving operational processes, safeguarding data, and conducting data analysis research.

# METHODOLOGY

## SINGLE MODEL

### RNN

RNN is a deep learning network structure. It is designed to handle sequence data. The key feature of an RNN is its ability to consider the sequential nature of the input data by introducing loops within the network. This loop allows information to be passed from one step to the next, which enables the network to capture dependencies and patterns in the data over time.

In time series analysis, RNNs can be trained to predict future values in a time series based on past observations. By using the historical data, the RNN can learn patterns and trends in the time series, allowing it to make predictions.

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from google.colab import drive  drive.mount('/content/drive')  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, SimpleRNN  Step 1: import library |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df1 = df.reset\_index()['Close']  Step 2: Read data |
| scaler = MinMaxScaler(feature\_range=(0,1))  df1 = scaler.fit\_transform(np.array(df1).reshape(-1,1))  Step 3: Scale data |
| train\_size = int(0.7 \* len(df1))  test\_size = int(0.2 \* len(df1))  val\_size = len(df1) - train\_size - test\_size  train\_data = df1[:train\_size]  test\_data = df1[train\_size:train\_size + test\_size]  val\_data = df1[train\_size + test\_size:]  Step 4: Separate train, test, validate |
| import numpy  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset) - time\_step - 1):      a = dataset[i:(i + time\_step), 0]      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)  Step 5: Create function create\_dataset with time\_step |
| time\_step = 100  X\_train, y\_train = create\_dataset(train\_data, time\_step)  X\_val, yval = create\_dataset(val\_data, time\_step)  X\_test, ytest = create\_dataset(test\_data, time\_step)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_val = X\_val.reshape(X\_val.shape[0], X\_val.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  Step 8: Apply SimpleRNN |
| train\_predict = model.predict(X\_train)  y\_pred = model.predict(X\_test)  y\_pred\_val = model.predict(X\_val)  Step 9: Predict train, test, validate |
| train\_predict = scaler.inverse\_transform(train\_predict)  y\_pred = scaler.inverse\_transform(y\_pred)  y\_pred\_val = scaler.inverse\_transform(y\_pred\_val)  Step 10: Normalise data |
| print('---------VALIDATE----------')  valid\_mape = np.mean(np.abs( (yval - y\_pred\_val) / yval )) \* 100  print('MAPE:', valid\_mape)  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST-------------')  test\_mape = np.mean(np.abs((ytest - y\_pred) / ytest)) \* 100  print('MAPE:', test\_mape)  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------------')  Step 11: Evaluate RNN model with MAPE, MAE, RMSE |
| train\_data\_index = pd.RangeIndex(start=0, stop=train\_size, step=1)  plt.plot(scaler.inverse\_transform(train\_data))  test\_data\_index = pd.RangeIndex(start=train\_size, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,scaler.inverse\_transform(test\_data))  test\_data\_index = pd.RangeIndex(start=train\_size+101, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,(y\_pred))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,scaler.inverse\_transform(val\_data))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size+101, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,y\_pred\_val)  plt.legend(['Train','Test','Predict','Validate','ValidatePred'])  plt.show()  Step 12: Plot data |

*Result of model RNN 7-2-1*

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from google.colab import drive  drive.mount('/content/drive')  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, SimpleRNN  Step 1: import library |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df1 = df.reset\_index()['Close']  Step 2: Read data |
| scaler = MinMaxScaler(feature\_range=(0,1))  df1 = scaler.fit\_transform(np.array(df1).reshape(-1,1))  Step 3: Scale data |
| train\_size = int(0.5 \* len(df1))  test\_size = int(0.3 \* len(df1))  val\_size = len(df1) - train\_size - test\_size  train\_data = df1[:train\_size]  test\_data = df1[train\_size:train\_size + test\_size]  val\_data = df1[train\_size + test\_size:]  Step 4: Separate train, test, validate |
| import numpy  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset) - time\_step - 1):      a = dataset[i:(i + time\_step), 0]      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)  Step 5: Create function create\_dataset with time\_step |
| time\_step = 100  X\_train, y\_train = create\_dataset(train\_data, time\_step)  X\_val, yval = create\_dataset(val\_data, time\_step)  X\_test, ytest = create\_dataset(test\_data, time\_step)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_val = X\_val.reshape(X\_val.shape[0], X\_val.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  Step 8: Apply SimpleRNN |
| train\_predict = model.predict(X\_train)  y\_pred = model.predict(X\_test)  y\_pred\_val = model.predict(X\_val)  Step 9: Predict train, test, validate |
| train\_predict = scaler.inverse\_transform(train\_predict)  y\_pred = scaler.inverse\_transform(y\_pred)  y\_pred\_val = scaler.inverse\_transform(y\_pred\_val)  Step 10: Normalise data |
| print('---------VALIDATE----------')  valid\_mape = np.mean(np.abs( (yval - y\_pred\_val) / yval )) \* 100  print('MAPE:', valid\_mape)  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST-------------')  test\_mape = np.mean(np.abs((ytest - y\_pred) / ytest)) \* 100  print('MAPE:', test\_mape)  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------------')  Step 11: Evaluate RNN model with MAPE, MAE, RMSE |
| train\_data\_index = pd.RangeIndex(start=0, stop=train\_size, step=1)  plt.plot(scaler.inverse\_transform(train\_data))  test\_data\_index = pd.RangeIndex(start=train\_size, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,scaler.inverse\_transform(test\_data))  test\_data\_index = pd.RangeIndex(start=train\_size+101, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,(y\_pred))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,scaler.inverse\_transform(val\_data))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size+101, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,y\_pred\_val)  plt.legend(['Train','Test','Predict','Validate','ValidatePred'])  plt.show()  Step 12: Plot data |

*Result of model RNN 7-2-1*

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from google.colab import drive  drive.mount('/content/drive')  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, SimpleRNN  Step 1: import library |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df1 = df.reset\_index()['Close']  Step 2: Read data |
| scaler = MinMaxScaler(feature\_range=(0,1))  df1 = scaler.fit\_transform(np.array(df1).reshape(-1,1))  Step 3: Scale data |
| train\_size = int(0.6 \* len(df1))  test\_size = int(0.2 \* len(df1))  val\_size = len(df1) - train\_size - test\_size  train\_data = df1[:train\_size]  test\_data = df1[train\_size:train\_size + test\_size]  val\_data = df1[train\_size + test\_size:]  Step 4: Separate train, test, validate |
| import numpy  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset) - time\_step - 1):      a = dataset[i:(i + time\_step), 0]      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)  Step 5: Create function create\_dataset with time\_step |
| time\_step = 100  X\_train, y\_train = create\_dataset(train\_data, time\_step)  X\_val, yval = create\_dataset(val\_data, time\_step)  X\_test, ytest = create\_dataset(test\_data, time\_step)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_val = X\_val.reshape(X\_val.shape[0], X\_val.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  Step 8: Apply SimpleRNN |
| train\_predict = model.predict(X\_train)  y\_pred = model.predict(X\_test)  y\_pred\_val = model.predict(X\_val)  Step 9: Predict train, test, validate |
| train\_predict = scaler.inverse\_transform(train\_predict)  y\_pred = scaler.inverse\_transform(y\_pred)  y\_pred\_val = scaler.inverse\_transform(y\_pred\_val)  Step 10: Normalise data |
| print('---------VALIDATE----------')  valid\_mape = np.mean(np.abs( (yval - y\_pred\_val) / yval )) \* 100  print('MAPE:', valid\_mape)  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST-------------')  test\_mape = np.mean(np.abs((ytest - y\_pred) / ytest)) \* 100  print('MAPE:', test\_mape)  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------------')  Step 11: Evaluate RNN model with MAPE, MAE, RMSE |
| train\_data\_index = pd.RangeIndex(start=0, stop=train\_size, step=1)  plt.plot(scaler.inverse\_transform(train\_data))  test\_data\_index = pd.RangeIndex(start=train\_size, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,scaler.inverse\_transform(test\_data))  test\_data\_index = pd.RangeIndex(start=train\_size+101, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,(y\_pred))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,scaler.inverse\_transform(val\_data))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size+101, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,y\_pred\_val)  plt.legend(['Train','Test','Predict','Validate','ValidatePred'])  plt.show()  Step 12: Plot data |

*Result of model RNN 6-2-2*

*Visualise data:*

|  |
| --- |
| *Result of model RNN on 7-2-1*    *Result of model RNN on 5-3-2*    *Result of model RNN on 6-2-2*  *TRX* |
| *Result of model RNN on 7-2-1*    *Result of model RNN on 5-3-2*    *Result of model RNN on 6-2-2*  *DOGE* |
| *Result of model RNN on 7-2-1*    *Result of model RNN on 5-3-2*    *Result of model RNN on 6-2-2*  *XLM* |

### LSTM

A diagram of a tank

Description automatically generated with low confidenceLong Short-Term Memory (LSTM) is based on the idea of gates. The main element of LSTM is the cell state (Ct) that can “remember” long-term dependencies, and the gates decide how the information flow.

FIGURE 1. The architecture of LSTM (source: https://colah.github.io/posts/2015-08-Understanding-LSTMs)

The input gate (1) decides how much newly received information (combined vector of ht−1 and xt) is added to the current cell state (4). Additionally, this gate applies appropriate transformations to the newly received information.

The forget gate (2) decides which part of the cell state from the previous period (Ct−1) will be forwarded. This gate receives a combined vector consisting of the hidden state (5) from the previous period (ht−1) and the input from the current period (xt). Forget gate returns a value from 0 to 1, where 0 means complete “forgetting” and 1 means all the information from the previous cell state is transferred.

Finally, the output flow is controlled by the output gate (3). It first decides what parts of the cell state are output and then transforms them accordingly using the *tanh* function.

|  |  |
| --- | --- |
| *it = sigmoid (Wi xt + Ui ht-1 + bi )* | (1) |
| *ft = sigmoid (Wf xt + Ui ht-1 + bf )* | (2) |
| *ot = sigmoid (Wo xt + Ui ht-1 + bo )* | (3) |
| *ct = ft ⋅ ct-1 + it ⋅ tanh(Wc xt + Uc ht-1 + bc )* | (4) |
| *ht = ot ⋅ tanh( ct )* | (5) |

LSTM neural network is a powerful tool in the field of machine learning. It can extract features, dimensions, and improve data classification. It is a machine language with the ability to learn internal representation and solve complex combinatorial problems.

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from google.colab import drive  drive.mount('/content/drive')  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, SimpleRNN  Step 1: import library |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df1 = df.reset\_index()['Close']  Step 2: Read data |
| scaler = MinMaxScaler(feature\_range=(0,1))  df1 = scaler.fit\_transform(np.array(df1).reshape(-1,1))  Step 3: Scale data |
| train\_size = int(0.7 \* len(df1))  test\_size = int(0.2 \* len(df1))  val\_size = len(df1) - train\_size - test\_size  train\_data = df1[:train\_size]  test\_data = df1[train\_size:train\_size + test\_size]  val\_data = df1[train\_size + test\_size:]  Step 4: Separate train, test, validate |
| import numpy  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset) - time\_step - 1):      a = dataset[i:(i + time\_step), 0]      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)  Step 5: Create function create\_dataset with time\_step |
| time\_step = 100  X\_train, y\_train = create\_dataset(train\_data, time\_step)  X\_val, yval = create\_dataset(val\_data, time\_step)  X\_test, ytest = create\_dataset(test\_data, time\_step)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_val = X\_val.reshape(X\_val.shape[0], X\_val.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  Step 8: Apply SimpleRNN |
| train\_predict = model.predict(X\_train)  y\_pred = model.predict(X\_test)  y\_pred\_val = model.predict(X\_val)  Step 9: Predict train, test, validate |
| train\_predict = scaler.inverse\_transform(train\_predict)  y\_pred = scaler.inverse\_transform(y\_pred)  y\_pred\_val = scaler.inverse\_transform(y\_pred\_val)  Step 10: Normalise data |
| print('---------VALIDATE----------')  valid\_mape = np.mean(np.abs( (yval - y\_pred\_val) / yval )) \* 100  print('MAPE:', valid\_mape)  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST-------------')  test\_mape = np.mean(np.abs((ytest - y\_pred) / ytest)) \* 100  print('MAPE:', test\_mape)  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------------')  Step 11: Evaluate RNN model with MAPE, MAE, RMSE |
| train\_data\_index = pd.RangeIndex(start=0, stop=train\_size, step=1)  plt.plot(scaler.inverse\_transform(train\_data))  test\_data\_index = pd.RangeIndex(start=train\_size, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,scaler.inverse\_transform(test\_data))  test\_data\_index = pd.RangeIndex(start=train\_size+101, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,(y\_pred))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,scaler.inverse\_transform(val\_data))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size+101, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,y\_pred\_val)  plt.legend(['Train','Test','Predict','Validate','ValidatePred'])  plt.show()  Step 12: Plot data |

*Result of model RNN 7-2-1*

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from google.colab import drive  drive.mount('/content/drive')  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, SimpleRNN  Step 1: import library |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df1 = df.reset\_index()['Close']  Step 2: Read data |
| scaler = MinMaxScaler(feature\_range=(0,1))  df1 = scaler.fit\_transform(np.array(df1).reshape(-1,1))  Step 3: Scale data |
| train\_size = int(0.5 \* len(df1))  test\_size = int(0.3 \* len(df1))  val\_size = len(df1) - train\_size - test\_size  train\_data = df1[:train\_size]  test\_data = df1[train\_size:train\_size + test\_size]  val\_data = df1[train\_size + test\_size:]  Step 4: Separate train, test, validate |
| import numpy  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset) - time\_step - 1):      a = dataset[i:(i + time\_step), 0]      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)  Step 5: Create function create\_dataset with time\_step |
| time\_step = 100  X\_train, y\_train = create\_dataset(train\_data, time\_step)  X\_val, yval = create\_dataset(val\_data, time\_step)  X\_test, ytest = create\_dataset(test\_data, time\_step)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_val = X\_val.reshape(X\_val.shape[0], X\_val.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  Step 8: Apply SimpleRNN |
| train\_predict = model.predict(X\_train)  y\_pred = model.predict(X\_test)  y\_pred\_val = model.predict(X\_val)  Step 9: Predict train, test, validate |
| train\_predict = scaler.inverse\_transform(train\_predict)  y\_pred = scaler.inverse\_transform(y\_pred)  y\_pred\_val = scaler.inverse\_transform(y\_pred\_val)  Step 10: Normalise data |
| print('---------VALIDATE----------')  valid\_mape = np.mean(np.abs( (yval - y\_pred\_val) / yval )) \* 100  print('MAPE:', valid\_mape)  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST-------------')  test\_mape = np.mean(np.abs((ytest - y\_pred) / ytest)) \* 100  print('MAPE:', test\_mape)  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------------')  Step 11: Evaluate RNN model with MAPE, MAE, RMSE |
| train\_data\_index = pd.RangeIndex(start=0, stop=train\_size, step=1)  plt.plot(scaler.inverse\_transform(train\_data))  test\_data\_index = pd.RangeIndex(start=train\_size, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,scaler.inverse\_transform(test\_data))  test\_data\_index = pd.RangeIndex(start=train\_size+101, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,(y\_pred))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,scaler.inverse\_transform(val\_data))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size+101, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,y\_pred\_val)  plt.legend(['Train','Test','Predict','Validate','ValidatePred'])  plt.show()  Step 12: Plot data |

*Result of model RNN 7-2-1*

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from google.colab import drive  drive.mount('/content/drive')  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, SimpleRNN  Step 1: import library |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df1 = df.reset\_index()['Close']  Step 2: Read data |
| scaler = MinMaxScaler(feature\_range=(0,1))  df1 = scaler.fit\_transform(np.array(df1).reshape(-1,1))  Step 3: Scale data |
| train\_size = int(0.6 \* len(df1))  test\_size = int(0.2 \* len(df1))  val\_size = len(df1) - train\_size - test\_size  train\_data = df1[:train\_size]  test\_data = df1[train\_size:train\_size + test\_size]  val\_data = df1[train\_size + test\_size:]  Step 4: Separate train, test, validate |
| import numpy  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset) - time\_step - 1):      a = dataset[i:(i + time\_step), 0]      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)  Step 5: Create function create\_dataset with time\_step |
| time\_step = 100  X\_train, y\_train = create\_dataset(train\_data, time\_step)  X\_val, yval = create\_dataset(val\_data, time\_step)  X\_test, ytest = create\_dataset(test\_data, time\_step)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_val = X\_val.reshape(X\_val.shape[0], X\_val.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  Step 8: Apply SimpleRNN |
| train\_predict = model.predict(X\_train)  y\_pred = model.predict(X\_test)  y\_pred\_val = model.predict(X\_val)  Step 9: Predict train, test, validate |
| train\_predict = scaler.inverse\_transform(train\_predict)  y\_pred = scaler.inverse\_transform(y\_pred)  y\_pred\_val = scaler.inverse\_transform(y\_pred\_val)  Step 10: Normalise data |
| print('---------VALIDATE----------')  valid\_mape = np.mean(np.abs( (yval - y\_pred\_val) / yval )) \* 100  print('MAPE:', valid\_mape)  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST-------------')  test\_mape = np.mean(np.abs((ytest - y\_pred) / ytest)) \* 100  print('MAPE:', test\_mape)  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------------')  Step 11: Evaluate RNN model with MAPE, MAE, RMSE |
| train\_data\_index = pd.RangeIndex(start=0, stop=train\_size, step=1)  plt.plot(scaler.inverse\_transform(train\_data))  test\_data\_index = pd.RangeIndex(start=train\_size, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,scaler.inverse\_transform(test\_data))  test\_data\_index = pd.RangeIndex(start=train\_size+101, stop=train\_size+test\_size, step=1)  plt.plot(test\_data\_index,(y\_pred))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,scaler.inverse\_transform(val\_data))  val\_data\_index = pd.RangeIndex(start=train\_size+test\_size+101, stop=train\_size+test\_size+val\_size, step=1)  plt.plot(val\_data\_index,y\_pred\_val)  plt.legend(['Train','Test','Predict','Validate','ValidatePred'])  plt.show()  Step 12: Plot data |

*Result of model RNN 6-2-2*

*Visualise data:*

|  |
| --- |
| *Result of model LSTM on 7-2-1*    *Result of model LSTM on 5-3-2*    *Result of model LSTM on 6-2-2*  *TRX* |
| *Result of model LSTM on 7-2-1*    *Result of model LSTM on 5-3-2*    *Result of model LSTM on 6-2-2*  *DOGE* |
| *Result of model LSTM on 7-2-1*    *Result of model LSTM on 5-3-2*    *Result of model LSTM on 6-2-2*  *XLM* |

### DNN

The Deep Neural Network (DNN) model is a multilayer machine learning model used in artificial intelligence. It is based on the structure of artificial neural networks, with multiple hidden layers between the input layer and the output layer. DNN is renowned as a deep neural network because it has the ability to learn and understand complex features from input data.

A picture containing diagram, screenshot, line, circle

Description automatically generated

FIGURE 2. Deep feedforward neural network. (Source: [deep\_nn.png (432×313) (uc-r.github.io)](https://uc-r.github.io/public/images/analytics/deep_learning/deep_nn.png))

The DNN model consists of three main components: the input layer, hidden layers, and output layer. The input layer receives the input data and passes it through the neural network. The hidden layers are where the learning and feature extraction processes take place. The output layer returns the prediction results. The number of neurons in each layer depends on the specific problem that the model is addressing.

The training process of the DNN model involves two main stages: forward propagation and backpropagation. In forward propagation, the input data is passed from the input layer through the hidden layers, using the ReLU activation function, to the output layer to compute the prediction results. Backpropagation utilizes the loss function to measure the error and adjust the weights in the neural network.

Activation function - ReLU:

**A diagram of a function

Description automatically generated with low confidence**

FIGURE 3. Flow of information in an artificial neuron.

Cre: [Feedforward Deep Learning Models · UC Business Analytics R Programming Guide (uc-r.github.io)](http://uc-r.github.io/feedforward_DNN)

**A black text on a white background

Description automatically generated with medium confidence**

Cre: [Feedforward Deep Learning Models · UC Business Analytics R Programming Guide (uc-r.github.io)](http://uc-r.github.io/feedforward_DNN)

Meaning: If the input value x is greater than 0, the output value will be equal to x. Conversely, if the input value x is less than or equal to 0, the output value will be 0.

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, LSTM  Step 1: import library |
| data = pd.read\_csv('/content/DOGE-USD.csv')  Step 2: Read data |
| timestamps = pd.to\_datetime(data['Date'])  prices = data['Close']  scaler = MinMaxScaler()  scaled\_prices = scaler.fit\_transform(prices.values.reshape(-1, 1))  Step 3: Scale data |
| train\_size = int(len(scaled\_prices) \* 0.7)  test\_size = int(len(scaled\_prices) \* 0.2)  validate\_size = len(scaled\_prices) - train\_size - test\_size  train\_data = scaled\_prices[:train\_size]  test\_data = scaled\_prices[train\_size:train\_size+test\_size]  validate\_data = scaled\_prices[train\_size+test\_size:]  Step 4: Separate train, test, validate |
| def create\_time\_series(data, time\_steps):      X, y = [], []      for i in range(len(data) - time\_steps):          X.append(data[i:i+time\_steps])          y.append(data[i+time\_steps])      return np.array(X), np.array(y)  Step 5: Create function create\_time\_series with time\_step |
| time\_steps = 100  X\_train, y\_train = create\_time\_series(train\_data, time\_steps)  X\_test, y\_test = create\_time\_series(test\_data, time\_steps)  X\_validate, y\_validate = create\_time\_series(validate\_data, time\_steps)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_validate = X\_validate.reshape(X\_validate.shape[0], X\_validate.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(Dense(50, activation='relu', input\_shape=(time\_steps,)))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mean\_squared\_error')  model.fit(X\_train, y\_train, epochs=100, batch\_size=64)  loss = model.evaluate(X\_test, y\_test)  print('Loss trên tập kiểm tra:', loss)  Step 8: Apply DNN |
| y\_test\_pred = model.predict(X\_test)  y\_test\_pred = scaler.inverse\_transform(y\_test\_pred)  y\_validate\_pred = model.predict(X\_validate)  y\_validate\_pred = scaler.inverse\_transform(y\_validate\_pred)train = prices[:train\_size]  test = prices[train\_size:train\_size+test\_size]  validate = prices[train\_size+test\_size:]  Step 9: Predict train, test, validate |
| from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error  y\_test\_predicted = model.predict(X\_test)  y\_validate\_predicted = model.predict(X\_validate)  y\_test\_predicted = scaler.inverse\_transform(y\_test\_predicted)  y\_validate\_predicted = scaler.inverse\_transform(y\_validate\_predicted)  y\_test = scaler.inverse\_transform(y\_test)  y\_validate = scaler.inverse\_transform(y\_validate)  mae\_test = mean\_absolute\_error(y\_test, y\_test\_predicted)  mape\_test = mean\_absolute\_percentage\_error(y\_test, y\_test\_predicted)  rmse\_test = mean\_squared\_error(y\_test, y\_test\_predicted, squared=False)  mae\_validate = mean\_absolute\_error(y\_validate, y\_validate\_predicted)  mape\_validate = mean\_absolute\_percentage\_error(y\_validate, y\_validate\_predicted)  rmse\_validate = mean\_squared\_error(y\_validate, y\_validate\_predicted, squared=False)  Step 10: Evaluate DNN model with MAPE, MAE, RMSE |
| plt.figure(figsize=(10, 6))  plt.plot(train, label='Train')  plt.plot(range(train\_size, train\_size+len(test)), test, label='Test')  plt.plot(range(train\_size+test\_size, train\_size+test\_size+len(validate)), validate, label='Validate')  plt.plot(range(train\_size, train\_size+len(y\_test\_pred)), y\_test\_pred, 'g', label='Test Predicted')  plt.plot(range(train\_size+test\_size, train\_size+test\_size+len(y\_validate\_pred)), y\_validate\_pred, 'b', label='Validate Predicted')  plt.plot(range(len(prices)-7, len(prices)), predicted\_prices\_7days, 'r', label='Predicted')  plt.xlabel('Ngày')  plt.ylabel('Giá cổ phiếu')  plt.title('Biểu đồ train, test, validate và dự đoán 7 ngày tiếp theo')  plt.legend()  plt.show()  Step 11: Plot data |

*Result of model DNN 7-2-1*

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, LSTM  Step 1: import library |
| data = pd.read\_csv('/content/DOGE-USD.csv')  Step 2: Read data |
| timestamps = pd.to\_datetime(data['Date'])  prices = data['Close']  scaler = MinMaxScaler()  scaled\_prices = scaler.fit\_transform(prices.values.reshape(-1, 1))  Step 3: Scale data |
| train\_size = int(len(scaled\_prices) \* 0.5)  test\_size = int(len(scaled\_prices) \* 0.3)  validate\_size = len(scaled\_prices) - train\_size - test\_size  train\_data = scaled\_prices[:train\_size]  test\_data = scaled\_prices[train\_size:train\_size+test\_size]  validate\_data = scaled\_prices[train\_size+test\_size:]  Step 4: Separate train, test, validate |
| def create\_time\_series(data, time\_steps):      X, y = [], []      for i in range(len(data) - time\_steps):          X.append(data[i:i+time\_steps])          y.append(data[i+time\_steps])      return np.array(X), np.array(y)  Step 5: Create function create\_time\_series with time\_step |
| time\_steps = 100  X\_train, y\_train = create\_time\_series(train\_data, time\_steps)  X\_test, y\_test = create\_time\_series(test\_data, time\_steps)  X\_validate, y\_validate = create\_time\_series(validate\_data, time\_steps)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_validate = X\_validate.reshape(X\_validate.shape[0], X\_validate.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(Dense(50, activation='relu', input\_shape=(time\_steps,)))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mean\_squared\_error')  model.fit(X\_train, y\_train, epochs=100, batch\_size=64)  loss = model.evaluate(X\_test, y\_test)  print('Loss trên tập kiểm tra:', loss)  Step 8: Apply DNN |
| y\_test\_pred = model.predict(X\_test)  y\_test\_pred = scaler.inverse\_transform(y\_test\_pred)  y\_validate\_pred = model.predict(X\_validate)  y\_validate\_pred = scaler.inverse\_transform(y\_validate\_pred)train = prices[:train\_size]  test = prices[train\_size:train\_size+test\_size]  validate = prices[train\_size+test\_size:]  Step 9: Predict train, test, validate |
| from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error  y\_test\_predicted = model.predict(X\_test)  y\_validate\_predicted = model.predict(X\_validate)  y\_test\_predicted = scaler.inverse\_transform(y\_test\_predicted)  y\_validate\_predicted = scaler.inverse\_transform(y\_validate\_predicted)  y\_test = scaler.inverse\_transform(y\_test)  y\_validate = scaler.inverse\_transform(y\_validate)  mae\_test = mean\_absolute\_error(y\_test, y\_test\_predicted)  mape\_test = mean\_absolute\_percentage\_error(y\_test, y\_test\_predicted)  rmse\_test = mean\_squared\_error(y\_test, y\_test\_predicted, squared=False)  mae\_validate = mean\_absolute\_error(y\_validate, y\_validate\_predicted)  mape\_validate = mean\_absolute\_percentage\_error(y\_validate, y\_validate\_predicted)  rmse\_validate = mean\_squared\_error(y\_validate, y\_validate\_predicted, squared=False)  Step 10: Evaluate DNN model with MAPE, MAE, RMSE |
| plt.figure(figsize=(10, 6))  plt.plot(train, label='Train')  plt.plot(range(train\_size, train\_size+len(test)), test, label='Test')  plt.plot(range(train\_size+test\_size, train\_size+test\_size+len(validate)), validate, label='Validate')  plt.plot(range(train\_size, train\_size+len(y\_test\_pred)), y\_test\_pred, 'g', label='Test Predicted')  plt.plot(range(train\_size+test\_size, train\_size+test\_size+len(y\_validate\_pred)), y\_validate\_pred, 'b', label='Validate Predicted')  plt.plot(range(len(prices)-7, len(prices)), predicted\_prices\_7days, 'r', label='Predicted')  plt.xlabel('Ngày')  plt.ylabel('Giá cổ phiếu')  plt.title('Biểu đồ train, test, validate và dự đoán 7 ngày tiếp theo')  plt.legend()  plt.show()  Step 11: Plot data |

*Result of model DNN 5-3-2*

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from keras.models import Sequential  from keras.layers import Dense, LSTM  Step 1: import library |
| data = pd.read\_csv('/content/DOGE-USD.csv')  Step 2: Read data |
| timestamps = pd.to\_datetime(data['Date'])  prices = data['Close']  scaler = MinMaxScaler()  scaled\_prices = scaler.fit\_transform(prices.values.reshape(-1, 1))  Step 3: Scale data |
| train\_size = int(len(scaled\_prices) \* 0.6)  test\_size = int(len(scaled\_prices) \* 0.2)  validate\_size = len(scaled\_prices) - train\_size - test\_size  train\_data = scaled\_prices[:train\_size]  test\_data = scaled\_prices[train\_size:train\_size+test\_size]  validate\_data = scaled\_prices[train\_size+test\_size:]  Step 4: Separate train, test, validate |
| def create\_time\_series(data, time\_steps):      X, y = [], []      for i in range(len(data) - time\_steps):          X.append(data[i:i+time\_steps])          y.append(data[i+time\_steps])      return np.array(X), np.array(y)  Step 5: Create function create\_time\_series with time\_step |
| time\_steps = 100  X\_train, y\_train = create\_time\_series(train\_data, time\_steps)  X\_test, y\_test = create\_time\_series(test\_data, time\_steps)  X\_validate, y\_validate = create\_time\_series(validate\_data, time\_steps)  Step 6: Create dataset with time\_step = 100 |
| X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  X\_validate = X\_validate.reshape(X\_validate.shape[0], X\_validate.shape[1], 1)  Step 7: Reshape into shape (none,none,none) for LSTM |
| model = Sequential()  model.add(Dense(50, activation='relu', input\_shape=(time\_steps,)))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mean\_squared\_error')  model.fit(X\_train, y\_train, epochs=100, batch\_size=64)  loss = model.evaluate(X\_test, y\_test)  print('Loss trên tập kiểm tra:', loss)  Step 8: Apply DNN |
| y\_test\_pred = model.predict(X\_test)  y\_test\_pred = scaler.inverse\_transform(y\_test\_pred)  y\_validate\_pred = model.predict(X\_validate)  y\_validate\_pred = scaler.inverse\_transform(y\_validate\_pred)train = prices[:train\_size]  test = prices[train\_size:train\_size+test\_size]  validate = prices[train\_size+test\_size:]  Step 9: Predict train, test, validate |
| from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error  y\_test\_predicted = model.predict(X\_test)  y\_validate\_predicted = model.predict(X\_validate)  y\_test\_predicted = scaler.inverse\_transform(y\_test\_predicted)  y\_validate\_predicted = scaler.inverse\_transform(y\_validate\_predicted)  y\_test = scaler.inverse\_transform(y\_test)  y\_validate = scaler.inverse\_transform(y\_validate)  mae\_test = mean\_absolute\_error(y\_test, y\_test\_predicted)  mape\_test = mean\_absolute\_percentage\_error(y\_test, y\_test\_predicted)  rmse\_test = mean\_squared\_error(y\_test, y\_test\_predicted, squared=False)  mae\_validate = mean\_absolute\_error(y\_validate, y\_validate\_predicted)  mape\_validate = mean\_absolute\_percentage\_error(y\_validate, y\_validate\_predicted)  rmse\_validate = mean\_squared\_error(y\_validate, y\_validate\_predicted, squared=False)  Step 10: Evaluate DNN model with MAPE, MAE, RMSE |
| plt.figure(figsize=(10, 6))  plt.plot(train, label='Train')  plt.plot(range(train\_size, train\_size+len(test)), test, label='Test')  plt.plot(range(train\_size+test\_size, train\_size+test\_size+len(validate)), validate, label='Validate')  plt.plot(range(train\_size, train\_size+len(y\_test\_pred)), y\_test\_pred, 'g', label='Test Predicted')  plt.plot(range(train\_size+test\_size, train\_size+test\_size+len(y\_validate\_pred)), y\_validate\_pred, 'b', label='Validate Predicted')  plt.plot(range(len(prices)-7, len(prices)), predicted\_prices\_7days, 'r', label='Predicted')  plt.xlabel('Ngày')  plt.ylabel('Giá cổ phiếu')  plt.title('Biểu đồ train, test, validate và dự đoán 7 ngày tiếp theo')  plt.legend()  plt.show()  Step 11: Plot data |

*Result of model DNN 6-2-2*

*Visualize data:*

|  |
| --- |
| *Result of model DNN on 7-2-1*    *Result of model DNN on 5-3-2*    *Result of model DNN on 6-2-2*  *TRX* |
| *Result of model DNN on 7-2-1*    *Result of model DNN on 5-3-2*    *Result of model DNN on 6-2-2*  *DOGE* |
| *Result of model DNN on 7-2-1*    *Result of model DNN on 5-3-2*    *Result of model DNN on 6-2-2*  *XLM* |

### ARIMA

ARIMA is a typical autoregression model which also includes the application of moving averages to increase the accuracy. This model is said to work best with a non-seasonal or stationary dataset. A stationary data must have no trend, constant amplitude variations around its mean, and consistent ups-and-lows, which means that statistically speaking, its short-term random time patterns remain the same. The latter requires that its power spectrum, or more precisely, its autocorrelations—correlations with its own prior departures from the mean—remain constant across time.

An "ARIMA(p,d,q)" model is a nonseasonal ARIMA model, where:

1) p is the number of autoregressive terms.

2) d is the number of nonseasonal variations required for stationarity.

3) q is the number of lags forecast errors.

A pure **Auto Regressive (AR) model** is one where Yt depends only on its own lags. That is, Yt is a function of the ‘lags of Yt’.

Ảnh có chứa Phông chữ, chữ viết tay, văn bản, thư pháp

Mô tả được tạo tự động

where,

Yt is the value of the time series at time t

Yt-1 , Y t-2, …, Y t-p are the lag of the series,

α is constants.

ꞵ1, ꞵ2, …ꞵp are autoregressive coefficients, representing the dependence of the current value on past values

ℇt is the random noise

A pure **Moving Average (MA) model** is one where Yt depends only on the lagged forecast errors.

Ảnh có chứa Phông chữ, chữ viết tay, thư pháp, thuật in máy

Mô tả được tạo tự động

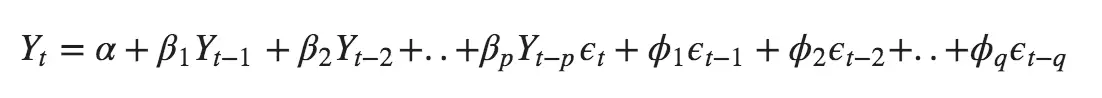
where

Y t is the value of the time series at time t

α is a constant

ϕ1 ,ϕ2, …, ϕp are moving average coefficients, representing the dependence on past random noise.

ℇ t-1, ℇ t-2, …,ℇ t-p are the random noise values in the past.



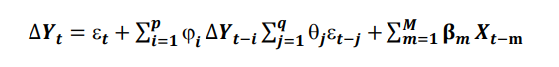
**Visualization**

|  |  |
| --- | --- |
| TRX-USD | Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, biểu đồ  Mô tả được tạo tự động  Ratio 7:2:1 |
| *Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, biểu đồ  Mô tả được tạo tự động*  Ratio 5:3:2 |
| Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, biểu đồ  Mô tả được tạo tự động  Ratio 6:2:2 |
| DOGE-USD | Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, biểu đồ  Mô tả được tạo tự động  Ratio 7:2:1 |
| Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, biểu đồ  Mô tả được tạo tự động  Ratio 5:3:2 |
| Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, hàng  Mô tả được tạo tự động  Ratio 6:2:2 |
| XLM-USD | Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, hàng  Mô tả được tạo tự động  Ratio 7:2:1 |
|  | Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, hàng  Mô tả được tạo tự động  Ratio 5:3:2 |
|  | Ảnh có chứa văn bản, ảnh chụp màn hình, Sơ đồ, biểu đồ  Mô tả được tạo tự động  Ratio 6:2:2 |

### ARIMAX

ARIMAX is the machine learning model used for time series analysis. It takes data and make observations. On the basis of previous observations it takes mean average and differentiate between consecutive two time-stamps in order to make time series stationary.

The mathematical expression for the ARIMAX model is given below:



where, 1,…,p and 1,…,q are the parameters; t, t-1 are white noise error and β1,…,βm are the parameters of independent variables input Xt and t is the time

**Visualization**

|  |  |
| --- | --- |
| **TRX-USD** | **Ratio 7:2:1** |
|  | **Ratio 5:3:2** |
|  | **Ratio 6:2:2** |
| **DOGE-USD** | **Ratio 7:2:1** |
|  | **Ratio 5:3:2** |
|  | **Ratio 6:2:2** |
| **XLM-USD** | **Ratio 7:2:1** |
|  | **Ratio 5:3:2** |
|  | **Ratio 6:2:1** |

### KNN

The KNN is straightforward to implement machine learning classification and regression problems. KNN is based on the similarity function and finds the K most similar instances in the training dataset for a new data instance. A mean or median target variable is taken as a prediction from the K neighbors.

The K-NN regression works as follows:

1. Initialize K number of neighbors.

2. Compute K of the nearest neighbors.

3. Calculate the distance between the test samples and the training samples.

4. Sort the training data based on distances.

5. Find the selected K entries' labels and assign them as prediction values.

**Visualization**

|  |  |
| --- | --- |
| **TRX-USD** | **Ratio 7:2:1** |
|  | **Ratio 5:3:2** |
|  | **Ratio 6:2:2** |
| **DOGE-USD** | **Ratio 7:2:1** |
|  |  |
|  | **Ratio 6:2:2** |
| **XLM-USD** | **Ratio 7:2:1** |
|  | **Ratio 5:3:2** |
|  | **Ratio 6:2:1** |

### LR

Linear regression, a method of machine learning, the model is based on a pair of two variables, the independent variable (x) and the dependent variable (y). In the context of multiple linear regression, there may be many independent variables. A simple linear regression has only one independent variable x. In the conditions given by the current model, ... the data set, there is only one independent variable, date. The first date that rises to the length of the date vector is represented by the integer . The length of this vector must be an integer and the date variable will change according to the time and, at the same time, the price of the stock, which is the dependent variable, will also change.

Ảnh có chứa biểu đồ

Mô tả được tạo tự động

FIGURE . Scatter chart

Cre: [Ước lượng hồi quy tuyến tính bằng OLS (xulydinhluong.com)](https://xulydinhluong.com/uoc-luong-hoi-quy-tuyen-tinh-bang-ols/)

* Simple Linear Regression Equation: *Y = β0 + β1X +* ℇ
* Multiple Linear Regression Equation: *Y = β0 + β1X1 + β2X2 + … + βk Xk +* ℇ [12]

Where:

Y is the dependent variable

X is the independent variable

X1, X2, ..., Xn are the independent variables

β0 is the y-intercept (the value of Y when all X variables are 0)

β1, β2, ..., βn are the slopes (the change in Y for a unit change in each respective X variable)

ε is the error term or residual (the difference between the predicted and actual values of Y)

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LinearRegression  from keras.models import Sequential  Step 1: import library |
| df = pd.read\_csv('/content/DOGE-USD.csv')  df = df[['Close']]  df = df.dropna()  df = df.reset\_index(drop=True)  Step 2: Read data |
| train\_size = int(0.7 \* len(df))  test\_size = int(0.2 \* len(df))  val\_size = len(df) - train\_size - test\_size  train\_data = df[:train\_size]  test\_data = df[train\_size:train\_size+test\_size]  val\_data = df[train\_size+test\_size:]  Step 3: Split the data |
| x\_train = np.array(train\_data.index).reshape(-1, 1)  y\_train = np.array(train\_data['Close'])  model = LinearRegression()  model.fit(x\_train, y\_train)  Step 4: Training |
| x\_test = np.array(test\_data.index).reshape(-1, 1)  y\_test = np.array(test\_data['Close'])  y\_pred = model.predict(x\_test)  x\_val= np.array(val\_data.index).reshape(-1, 1)  y\_val = np.array(val\_data['Close'])  y\_pred\_val =  model.predict(x\_val)  Step 5: Testing Validation |
| from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  def calculate\_mape(y\_true, y\_pred):      return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  mae\_test = mean\_absolute\_error(y\_test, y\_pred)  rmse\_test = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  mape\_test = calculate\_mape(y\_test, y\_pred)  mae\_val = mean\_absolute\_error(y\_val, y\_pred\_val)  rmse\_val = np.sqrt(mean\_squared\_error(y\_val, y\_pred\_val))  mape\_val = calculate\_mape(y\_val, y\_pred\_val)  print('---------VALIDATE----------')  print("MAPE:", round(mape\_val, 3))  print("MAE:", round(mae\_val, 3))  print("RMSE:", round(rmse\_val, 3))  print('--------------------------')  print('---------TEST----------')  print("MAPE:", round(mape\_test, 3))  print("MAE:", round(mae\_test, 3))  print("RMSE:", round(rmse\_test, 3))  print('-------------------')  Step 6: Evaluate LN model with MAPE, MAE, RMSE |
| plt.plot(train\_data.index, train\_data['Close'])  plt.plot(test\_data.index, test\_data['Close'])  plt.plot(test\_data.index, y\_pred)  plt.plot(val\_data.index, val\_data['Close'])  plt.plot(val\_data.index, y\_pred\_val)  plt.legend(['Train', 'Test', 'Predictions', 'Validate', 'Validate Predictions'])  plt.show()  Step 7: Plot data |

*Result of model LN 7-2-1*

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LinearRegression  from keras.models import Sequential  Step 1: import library |
| df = pd.read\_csv('/content/DOGE-USD.csv')  df = df[['Close']]  df = df.dropna()  df = df.reset\_index(drop=True)  Step 2: Read data |
| train\_size = int(0.5 \* len(df))  test\_size = int(0.3 \* len(df))  val\_size = len(df) - train\_size - test\_size  train\_data = df[:train\_size]  test\_data = df[train\_size:train\_size+test\_size]  val\_data = df[train\_size+test\_size:]  Step 3: Split the data |
| x\_train = np.array(train\_data.index).reshape(-1, 1)  y\_train = np.array(train\_data['Close'])  model = LinearRegression()  model.fit(x\_train, y\_train)  Step 4: Training |
| x\_test = np.array(test\_data.index).reshape(-1, 1)  y\_test = np.array(test\_data['Close'])  y\_pred = model.predict(x\_test)  x\_val= np.array(val\_data.index).reshape(-1, 1)  y\_val = np.array(val\_data['Close'])  y\_pred\_val =  model.predict(x\_val)  Step 5: Testing Validation |
| from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  def calculate\_mape(y\_true, y\_pred):      return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  mae\_test = mean\_absolute\_error(y\_test, y\_pred)  rmse\_test = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  mape\_test = calculate\_mape(y\_test, y\_pred)  mae\_val = mean\_absolute\_error(y\_val, y\_pred\_val)  rmse\_val = np.sqrt(mean\_squared\_error(y\_val, y\_pred\_val))  mape\_val = calculate\_mape(y\_val, y\_pred\_val)  print('---------VALIDATE----------')  print("MAPE:", round(mape\_val, 3))  print("MAE:", round(mae\_val, 3))  print("RMSE:", round(rmse\_val, 3))  print('--------------------------')  print('---------TEST----------')  print("MAPE:", round(mape\_test, 3))  print("MAE:", round(mae\_test, 3))  print("RMSE:", round(rmse\_test, 3))  print('-------------------')  Step 6: Evaluate LN model with MAPE, MAE, RMSE |
| plt.plot(train\_data.index, train\_data['Close'])  plt.plot(test\_data.index, test\_data['Close'])  plt.plot(test\_data.index, y\_pred)  plt.plot(val\_data.index, val\_data['Close'])  plt.plot(val\_data.index, y\_pred\_val)  plt.legend(['Train', 'Test', 'Predictions', 'Validate', 'Validate Predictions'])  plt.show()  Step 7: Plot data |

*Result of model LN 5-3-2*

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LinearRegression  from keras.models import Sequential  Step 1: import library |
| df = pd.read\_csv('/content/DOGE-USD.csv')  df = df[['Close']]  df = df.dropna()  df = df.reset\_index(drop=True)  Step 2: Read data |
| train\_size = int(0.6 \* len(df))  test\_size = int(0.2 \* len(df))  val\_size = len(df) - train\_size - test\_size  train\_data = df[:train\_size]  test\_data = df[train\_size:train\_size+test\_size]  val\_data = df[train\_size+test\_size:]  Step 3: Split the data |
| x\_train = np.array(train\_data.index).reshape(-1, 1)  y\_train = np.array(train\_data['Close'])  model = LinearRegression()  model.fit(x\_train, y\_train)  Step 4: Training |
| x\_test = np.array(test\_data.index).reshape(-1, 1)  y\_test = np.array(test\_data['Close'])  y\_pred = model.predict(x\_test)  x\_val= np.array(val\_data.index).reshape(-1, 1)  y\_val = np.array(val\_data['Close'])  y\_pred\_val =  model.predict(x\_val)  Step 5: Testing Validation |
| from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  def calculate\_mape(y\_true, y\_pred):      return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  mae\_test = mean\_absolute\_error(y\_test, y\_pred)  rmse\_test = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  mape\_test = calculate\_mape(y\_test, y\_pred)  mae\_val = mean\_absolute\_error(y\_val, y\_pred\_val)  rmse\_val = np.sqrt(mean\_squared\_error(y\_val, y\_pred\_val))  mape\_val = calculate\_mape(y\_val, y\_pred\_val)  print('---------VALIDATE----------')  print("MAPE:", round(mape\_val, 3))  print("MAE:", round(mae\_val, 3))  print("RMSE:", round(rmse\_val, 3))  print('--------------------------')  print('---------TEST----------')  print("MAPE:", round(mape\_test, 3))  print("MAE:", round(mae\_test, 3))  print("RMSE:", round(rmse\_test, 3))  print('-------------------')  Step 6: Evaluate LN model with MAPE, MAE, RMSE |
| plt.plot(train\_data.index, train\_data['Close'])  plt.plot(test\_data.index, test\_data['Close'])  plt.plot(test\_data.index, y\_pred)  plt.plot(val\_data.index, val\_data['Close'])  plt.plot(val\_data.index, y\_pred\_val)  plt.legend(['Train', 'Test', 'Predictions', 'Validate', 'Validate Predictions'])  plt.show()  Step 7: Plot data |

*Result of model LN 6-2-2*

*Visualise data:*

|  |
| --- |
| *Result of model LN on 7-2-1*    *Result of model LN on 5-3-2*    *Result of model LN on 6-2-2*  *TRX* |
| *Result of model LN on 7-2-1*    *Result of model LN on 5-3-2*    *Result of model LN on 6-2-2*  *DOGE* |
| *Result of model LN on 7-2-1*    *Result of model LN on 5-3-2*    *Result of model LN on 6-2-2*  *XLM* |

## HYBRID MODEL

### Hybrid model based on TIME SERIES ANOMALY DETECTION with ARIMA

e Detecting outliers, or anomaly detection, is a method to identify patterns in a dataset that do not match the expected patterns and significantly differ from them. There are various methods such as isolation-based methods, model-based methods, density-based methods, and distance-based methods. Among them, Isolation Forest (IF) is an efficient technique that uses a machine learning algorithm based on a binary tree structure with random sampling, providing a set of trees from training and testing datasets.

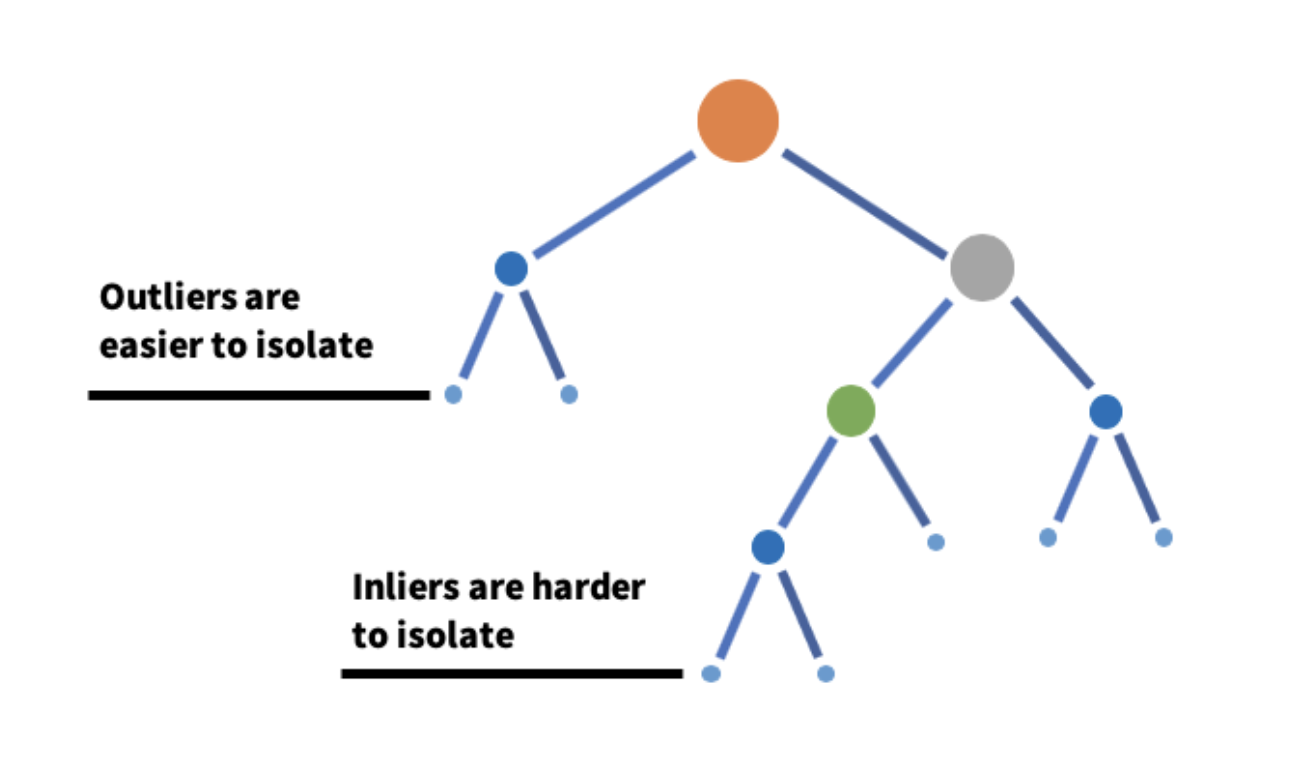


FIGURE . An example isolation tree

Cre: [Detecting and preventing abuse on LinkedIn using isolation forests | LinkedIn Engineering](https://engineering.linkedin.com/blog/2019/isolation-forest)

Isolation Forest (IF) is a method for anomaly detection in data. It relies on constructing binary trees from the training data and using them to measure the outlierness of data samples. The trees are built by randomly selecting a subset of the data and creating partitions based on attribute values. Anomalous samples often have shorter paths in the trees compared to normal samples. The outlierness score of a sample is computed based on the average path length through the trees. The formula S(x,n) is used to calculate the outlier score of sample x:

Where:

E(h(x)) is the average path length

c(n) is the expected average path length of random paths.

IF can quickly and effectively detect data points that deviate from normal patterns.

Subsequently, we proceeded to predict the closing prices using various advanced algorithms such as RNN, DNN, LSTM. These algorithms are commonly employed in analyzing time series data.

We trained these models using clustered time series data, enabling them to learn and remember patterns from past observations. Our goal was to uncover trends and potential patterns in closing price data. This approach aimed to provide valuable insights to support investment decision-making and effective risk management in the cryptocurrency domain.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.ensemble import IsolationForest  import matplotlib.pyplot as plt  from sklearn.linear\_model import LinearRegression  from pmdarima.arima import auto\_arima  Step 1: import library |
| data = pd.read\_csv('/content/DOGE-USD.csv')  data['Date'] = pd.to\_datetime(data['Date'])  data = Đọc dữ liệu từ file CSV  Step 2: Read data |
| train\_size = int(0.7 \* len(data))  test\_size = int(0.2 \* len(data))  validate\_size = len(data) - train\_size - test\_size  train\_data = data[:train\_size]  test\_data = data[train\_size:train\_size+test\_size]  validate\_data = data[train\_size+test\_size:]  Step 3: Split data |
| train\_dates = train\_data['Date'].values  train\_prices = train\_data['Close'].values  test\_dates = test\_data['Date'].values  test\_prices = test\_data['Close'].values  validate\_dates = validate\_data['Date'].values  validate\_prices = validate\_data['Close'].values  Step 4: train, test, validate to chart |
| model = IsolationForest(contamination=0.05)  model.fit(train\_prices.reshape(-1, 1))  Step 5: Isolation Forest |
| train\_scores = model.decision\_function(train\_prices.reshape(-1, 1))  test\_scores = model.decision\_function(test\_prices.reshape(-1, 1))  validate\_scores = model.decision\_function(validate\_prices.reshape(-1, 1))  Step 6: Train test validate predicted |
| train\_anomalies = np.where(model.predict(train\_prices.reshape(-1, 1)) == -1)  test\_anomalies = np.where(model.predict(test\_prices.reshape(-1, 1)) == -1)  validate\_anomalies = np.where(model.predict(validate\_prices.reshape(-1, 1)) == -1)  Step 7: anomaly detection |
| plt.figure(figsize=(10, 6))  plt.plot(train\_dates, train\_prices, label='Train')  plt.plot(test\_dates, test\_prices, label='Test')  plt.plot(validate\_dates, validate\_prices, label='Validate')  plt.scatter(train\_dates[train\_anomalies], train\_prices[train\_anomalies], color='red', label='Train Anomalies')  plt.scatter(test\_dates[test\_anomalies], test\_prices[test\_anomalies], color='red', label='Test Anomalies')  plt.scatter(validate\_dates[validate\_anomalies], validate\_prices[validate\_anomalies], color='red', label='Validate Anomalies')  plt.xlabel('Date')  plt.ylabel('Closing Price')  plt.title('Time Series Anomaly Detection')  plt.legend()  plt.show()    Step 8: Chart anomaly detection |
| plt.figure(figsize=(10, 6))  plt.plot(train\_dates, train\_prices, label='Train')  plt.plot(test\_dates, test\_prices, label='Test')  plt.plot(validate\_dates, validate\_prices, label='Validate')  plt.plot(test\_dates, test\_predicted\_prices\_arima, label='Test Predicted Prices (ARIMA)')  plt.plot(validate\_dates, validate\_predicted\_prices\_arima, label='Validate Predicted Prices (ARIMA)')  plt.xlabel('Date')  plt.ylabel('Closing Price')  plt.title('Train, Test, Validate Data')  plt.legend()  plt.show()  Step 9: Plot data |
| def calculate\_metrics(actual, predicted):      mape = np.mean(np.abs((actual - predicted) / actual)) \* 100      mae = np.mean(np.abs(actual - predicted))      mse = np.mean((actual - predicted)\*\*2)      rmse = np.sqrt(mse)      return mae, mse, rmse  validate\_mape, validate\_mae, validate\_rmse = calculate\_metrics(validate\_prices, validate\_scores)  test\_mape, test\_mae, test\_rmse = calculate\_metrics(test\_prices, test\_scores)  print('---------VALIDATE----------')  print("MAPE:", round(validate\_mape, 3))  print("MAE:", round(validate\_mae, 3))  print("RMSE:", round(validate\_rmse, 3))  print('--------------------------')  print('---------TEST----------')  print("MAPE:", round(test\_mape, 3))  print("MAE:", round(test\_mae, 3))  print("RMSE:", round(test\_rmse, 3))  print('-------------------')  Step 10: Evaluate model with MAPE, MAE, RMSE |

*Result of model 7-2-1*

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.ensemble import IsolationForest  import matplotlib.pyplot as plt  from sklearn.linear\_model import LinearRegression  from pmdarima.arima import auto\_arima  Step 1: import library |
| data = pd.read\_csv('/content/DOGE-USD.csv')  data['Date'] = pd.to\_datetime(data['Date'])  data = Đọc dữ liệu từ file CSV  Step 2: Read data |
| train\_size = int(0.5 \* len(data))  test\_size = int(0.3 \* len(data))  validate\_size = len(data) - train\_size - test\_size  train\_data = data[:train\_size]  test\_data = data[train\_size:train\_size+test\_size]  validate\_data = data[train\_size+test\_size:]  Step 3: Split data |
| train\_dates = train\_data['Date'].values  train\_prices = train\_data['Close'].values  test\_dates = test\_data['Date'].values  test\_prices = test\_data['Close'].values  validate\_dates = validate\_data['Date'].values  validate\_prices = validate\_data['Close'].values  Step 4: train, test, validate to chart |
| model = IsolationForest(contamination=0.05)  model.fit(train\_prices.reshape(-1, 1))  Step 5: Isolation Forest |
| train\_scores = model.decision\_function(train\_prices.reshape(-1, 1))  test\_scores = model.decision\_function(test\_prices.reshape(-1, 1))  validate\_scores = model.decision\_function(validate\_prices.reshape(-1, 1))  Step 6: Train test validate predicted |
| train\_anomalies = np.where(model.predict(train\_prices.reshape(-1, 1)) == -1)  test\_anomalies = np.where(model.predict(test\_prices.reshape(-1, 1)) == -1)  validate\_anomalies = np.where(model.predict(validate\_prices.reshape(-1, 1)) == -1)  Step 7: anomaly detection |
| plt.figure(figsize=(10, 6))  plt.plot(train\_dates, train\_prices, label='Train')  plt.plot(test\_dates, test\_prices, label='Test')  plt.plot(validate\_dates, validate\_prices, label='Validate')  plt.scatter(train\_dates[train\_anomalies], train\_prices[train\_anomalies], color='red', label='Train Anomalies')  plt.scatter(test\_dates[test\_anomalies], test\_prices[test\_anomalies], color='red', label='Test Anomalies')  plt.scatter(validate\_dates[validate\_anomalies], validate\_prices[validate\_anomalies], color='red', label='Validate Anomalies')  plt.xlabel('Date')  plt.ylabel('Closing Price')  plt.title('Time Series Anomaly Detection')  plt.legend()  plt.show()    Step 8: Chart anomaly detection |
| plt.figure(figsize=(10, 6))  plt.plot(train\_dates, train\_prices, label='Train')  plt.plot(test\_dates, test\_prices, label='Test')  plt.plot(validate\_dates, validate\_prices, label='Validate')  plt.plot(test\_dates, test\_predicted\_prices\_arima, label='Test Predicted Prices (ARIMA)')  plt.plot(validate\_dates, validate\_predicted\_prices\_arima, label='Validate Predicted Prices (ARIMA)')  plt.xlabel('Date')  plt.ylabel('Closing Price')  plt.title('Train, Test, Validate Data')  plt.legend()  plt.show()  Step 9: Plot data |
| def calculate\_metrics(actual, predicted):      mape = np.mean(np.abs((actual - predicted) / actual)) \* 100      mae = np.mean(np.abs(actual - predicted))      mse = np.mean((actual - predicted)\*\*2)      rmse = np.sqrt(mse)      return mae, mse, rmse  validate\_mape, validate\_mae, validate\_rmse = calculate\_metrics(validate\_prices, validate\_scores)  test\_mape, test\_mae, test\_rmse = calculate\_metrics(test\_prices, test\_scores)  print('---------VALIDATE----------')  print("MAPE:", round(validate\_mape, 3))  print("MAE:", round(validate\_mae, 3))  print("RMSE:", round(validate\_rmse, 3))  print('--------------------------')  print('---------TEST----------')  print("MAPE:", round(test\_mape, 3))  print("MAE:", round(test\_mae, 3))  print("RMSE:", round(test\_rmse, 3))  print('-------------------')  Step 10: Evaluate model with MAPE, MAE, RMSE |

*Result of model 5-3-2*

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.ensemble import IsolationForest  import matplotlib.pyplot as plt  from sklearn.linear\_model import LinearRegression  from pmdarima.arima import auto\_arima  Step 1: import library |
| data = pd.read\_csv('/content/DOGE-USD.csv')  data['Date'] = pd.to\_datetime(data['Date'])  data = Đọc dữ liệu từ file CSV  Step 2: Read data |
| train\_size = int(0.6 \* len(data))  test\_size = int(0.2 \* len(data))  validate\_size = len(data) - train\_size - test\_size  train\_data = data[:train\_size]  test\_data = data[train\_size:train\_size+test\_size]  validate\_data = data[train\_size+test\_size:]  Step 3: Split data |
| train\_dates = train\_data['Date'].values  train\_prices = train\_data['Close'].values  test\_dates = test\_data['Date'].values  test\_prices = test\_data['Close'].values  validate\_dates = validate\_data['Date'].values  validate\_prices = validate\_data['Close'].values  Step 4: train, test, validate to chart |
| model = IsolationForest(contamination=0.05)  model.fit(train\_prices.reshape(-1, 1))  Step 5: Isolation Forest |
| train\_scores = model.decision\_function(train\_prices.reshape(-1, 1))  test\_scores = model.decision\_function(test\_prices.reshape(-1, 1))  validate\_scores = model.decision\_function(validate\_prices.reshape(-1, 1))  Step 6: Train test validate predicted |
| train\_anomalies = np.where(model.predict(train\_prices.reshape(-1, 1)) == -1)  test\_anomalies = np.where(model.predict(test\_prices.reshape(-1, 1)) == -1)  validate\_anomalies = np.where(model.predict(validate\_prices.reshape(-1, 1)) == -1)  Step 7: anomaly detection |
| plt.figure(figsize=(10, 6))  plt.plot(train\_dates, train\_prices, label='Train')  plt.plot(test\_dates, test\_prices, label='Test')  plt.plot(validate\_dates, validate\_prices, label='Validate')  plt.scatter(train\_dates[train\_anomalies], train\_prices[train\_anomalies], color='red', label='Train Anomalies')  plt.scatter(test\_dates[test\_anomalies], test\_prices[test\_anomalies], color='red', label='Test Anomalies')  plt.scatter(validate\_dates[validate\_anomalies], validate\_prices[validate\_anomalies], color='red', label='Validate Anomalies')  plt.xlabel('Date')  plt.ylabel('Closing Price')  plt.title('Time Series Anomaly Detection')  plt.legend()  plt.show()    Step 8: Chart anomaly detection |
| plt.figure(figsize=(10, 6))  plt.plot(train\_dates, train\_prices, label='Train')  plt.plot(test\_dates, test\_prices, label='Test')  plt.plot(validate\_dates, validate\_prices, label='Validate')  plt.plot(test\_dates, test\_predicted\_prices\_arima, label='Test Predicted Prices (ARIMA)')  plt.plot(validate\_dates, validate\_predicted\_prices\_arima, label='Validate Predicted Prices (ARIMA)')  plt.xlabel('Date')  plt.ylabel('Closing Price')  plt.title('Train, Test, Validate Data')  plt.legend()  plt.show()  Step 9: Plot data |
| def calculate\_metrics(actual, predicted):      mape = np.mean(np.abs((actual - predicted) / actual)) \* 100      mae = np.mean(np.abs(actual - predicted))      mse = np.mean((actual - predicted)\*\*2)      rmse = np.sqrt(mse)      return mae, mse, rmse  validate\_mape, validate\_mae, validate\_rmse = calculate\_metrics(validate\_prices, validate\_scores)  test\_mape, test\_mae, test\_rmse = calculate\_metrics(test\_prices, test\_scores)  print('---------VALIDATE----------')  print("MAPE:", round(validate\_mape, 3))  print("MAE:", round(validate\_mae, 3))  print("RMSE:", round(validate\_rmse, 3))  print('--------------------------')  print('---------TEST----------')  print("MAPE:", round(test\_mape, 3))  print("MAE:", round(test\_mae, 3))  print("RMSE:", round(test\_rmse, 3))  print('-------------------')  Step 10: Evaluate model with MAPE, MAE, RMSE |

*Result of model 6-2-2*

|  |
| --- |
| *Result of model on 7-2-1*    *Result of model on 5-3-2*    *Result of model on 6-2-2*  *TRX* |
| *Result of model on 7-2-1*    *Result of model on 5-3-2*    *Result of model on 6-2-2*  *DOGE* |
| *Result of model on 7-2-1*    *Result of model on 5-3-2*    *Result of model on 6-2-2*  *XLM* |

### Hybrid model based on TIME SERIES CLUSTERING with RNN, LSTM, DNN

Clustering time series can help extract the closing prices of cryptocurrencies that change within a specific time. By applying this method, we can identify trends and price charts of cryptocurrencies in the past and use them to predict potential fluctuations in the future.

Through clustering, we can recognize clusters of data with increasing price trends, decreasing price trends, and clusters with stable prices. This enables us to determine significant price changes in the past and predict potential future changes in cryptocurrencies.

In this study, we experimented with the standard k-means clustering method using two popular distance metrics: Euclidean distance and Dynamic Time Warping (DTW) distance. These distance measures were used to assess the similarity between time series data, and the k-means algorithm was applied to cluster the corresponding data.

To evaluate the performance of the generated clusters, we employed the silhouette score. The silhouette score measures the cohesion within clusters and the separation between clusters, providing an overall assessment of clustering quality. Higher silhouette scores indicate well-defined and well-separated clusters, while lower scores suggest overlapping or poorly separated clusters.

By using k-means clustering with different distance metrics and evaluating the results using the silhouette score, our aim was to determine the most suitable clustering approach for analysing time series data in the context of cryptocurrency closing prices.

Subsequently, we proceeded to predict the closing prices of Bitcoin using various advanced algorithms such as RNN, DNN, LSTM. These algorithms are commonly employed in analyzing time series data.

We trained these models using clustered time series data, enabling them to learn and remember patterns from past observations. Our goal was to uncover trends and potential patterns in Bitcoin's closing price data. This approach aimed to provide valuable insights to support investment decision-making and effective risk management in the cryptocurrency domain.

* **TRX-USD**

|  |
| --- |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/TRX-USD.csv',usecols=['Date','Close'])  df = df['Close']  df1 = df.values.reshape(-1, 1)  import numpy  # convert an array of values into a dataset matrix  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset)-time\_step-1):      a = dataset[i:(i+time\_step), 0]   ###i=0, X=0,1,2,3-----99   Y=100      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)    time\_step = 100  X, y = create\_dataset(df1, time\_step)  X = X.reshape(X.shape[0],X.shape[1] , 1)  *Step 1: Create dataset with time\_step = 100.* |
| *We have 1916 samples, each sample has a length of 100 and 1 feature* |
| seed = 0  np.random.seed(seed)  X = TimeSeriesScalerMeanVariance().fit\_transform(X)  sz = X.shape[1]  *Step 2: Using TimeSeriesScalerMeanVariance() to scaled time series data* |
| for yi in range(12):      plt.subplot(4, 3, yi + 1)      plt.plot(X[yi].ravel(), "k-", alpha=.2)    *Step 3: Look shape of 12 data* |
| # Using sum of squared distances to decide cluster number  Sum\_of\_squared\_distances = []  K = range(1,8)  for k in K:      km = TimeSeriesKMeans(n\_clusters=k,                            n\_init=2,                            metric="dtw",                            verbose=False,                            max\_iter\_barycenter=10,                            random\_state=0)      km = km.fit(X)      Sum\_of\_squared\_distances.append(km.inertia\_)  plt.plot(K, Sum\_of\_squared\_distances, 'bx-')  plt.xlabel('k')  plt.ylabel('Sum\_of\_squared\_distances')  plt.title('Elbow Method For Optimal k')  plt.show()  *Step 4: Using sum of squared distances to decide cluster number (k)*    *Look at this plot, we choose k = 3* |
| # Euclidean k-means  print("-----Euclidean k-means-----")  n\_clusters=3  km = TimeSeriesKMeans(n\_clusters=n\_clusters, verbose=False, random\_state=seed)  y\_km = km.fit\_predict(X)  print("Euclidean silhoutte: {:.2f}".format(silhouette\_score(X, y\_km, metric="euclidean")))  plt.figure()  for yi in range(n\_clusters):      plt.subplot(3, n\_clusters, yi + 1)      for xx in X[y\_km == yi]:          plt.plot(xx.ravel(), "k-", alpha=.2)      plt.plot(km.cluster\_centers\_[yi].ravel(), "r-")      plt.xlim(0, sz)      plt.ylim(-4, 4)      plt.text(0.55, 0.85,'Cluster %d' % (yi + 1),               transform=plt.gca().transAxes)      if yi == 1:          plt.title("Euclidean $k$-means")    *Step 4: Kmeans using euclidean distance* |
| # DBA-k-means  print("DBA k-means")  dba\_km = TimeSeriesKMeans(n\_clusters=n\_clusters,                            n\_init=2,                            metric="dtw",                            verbose=False,                            max\_iter\_barycenter=10,                            random\_state=seed)  y\_dba\_km = dba\_km.fit\_predict(X)  print("DBA silhoutte: {:.2f}".format(silhouette\_score(X, y\_dba\_km, metric="dtw")))  for yi in range(n\_clusters):      plt.subplot(2, n\_clusters, yi+1)      for xx in X[y\_dba\_km == yi]:          plt.plot(xx.ravel(), "k-", alpha=.2)      plt.plot(dba\_km.cluster\_centers\_[yi].ravel(), "r-")      plt.xlim(0, sz)      plt.ylim(-4, 4)      plt.text(0.55, 0.85,'Cluster %d' % (yi + 1),               transform=plt.gca().transAxes)      if yi == 1:          plt.title("DBA $k$-means")  plt.tight\_layout()  plt.show()  *Step 5: Kmeans using dynamic time warping (dtw) distance* |
| *The silhoutte of DBA K-Means is higher so we choose dba\_km to preidct cluster\_labels.*  cluster\_labels = dba\_km.labels\_  cluster\_0 = X[cluster\_labels == 0]  cluster\_1 = X[cluster\_labels == 1]  cluster\_2 = X[cluster\_labels == 2]  *Step 6: We separate cluster\_labels with 3 dataset: cluster\_0, cluster\_1, cluster\_2* |

**Now we apply RNN, LSTM, DNN for each dataset:**

|  |
| --- |
| *RNN:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  *Step 2: Apply RNN model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate RNN model* |
| *LSTM:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model=Sequential()  model.add(LSTM(50,return\_sequences=True,input\_shape=(time\_step,1)))  model.add(LSTM(50,return\_sequences=True))  model.add(LSTM(50))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  *Step 2: Apply LSTM model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate LSTM model* |
| *DNN:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model = Sequential()  model.add(Dense(50, activation='relu', input\_shape=(time\_step,)))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mean\_squared\_error')  model.fit(X\_train, y\_train, epochs=100, batch\_size=64)  *Step 2: Apply LSTM model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate LSTM model* |

*Do the same with cluster\_1 and cluster\_2.*

* **DOGE-USD**

|  |
| --- |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/DOGE-USD.csv',usecols=['Date','Close'])  df = df['Close']  df1 = df.values.reshape(-1, 1)  import numpy  # convert an array of values into a dataset matrix  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset)-time\_step-1):      a = dataset[i:(i+time\_step), 0]   ###i=0, X=0,1,2,3-----99   Y=100      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)    time\_step = 100  X, y = create\_dataset(df1, time\_step)  X = X.reshape(X.shape[0],X.shape[1] , 1)  *Step 1: Create dataset with time\_step = 100.* |
| *We have 1916 samples, each sample has a length of 100 and 1 feature* |
| seed = 0  np.random.seed(seed)  X = TimeSeriesScalerMeanVariance().fit\_transform(X)  sz = X.shape[1]  *Step 2: Using TimeSeriesScalerMeanVariance() to scaled time series data* |
| for yi in range(12):      plt.subplot(4, 3, yi + 1)      plt.plot(X[yi].ravel(), "k-", alpha=.2)    *Step 3: Look shape of 12 data* |
| # Using sum of squared distances to decide cluster number  Sum\_of\_squared\_distances = []  K = range(1,8)  for k in K:      km = TimeSeriesKMeans(n\_clusters=k,                            n\_init=2,                            metric="dtw",                            verbose=False,                            max\_iter\_barycenter=10,                            random\_state=0)      km = km.fit(X)      Sum\_of\_squared\_distances.append(km.inertia\_)  plt.plot(K, Sum\_of\_squared\_distances, 'bx-')  plt.xlabel('k')  plt.ylabel('Sum\_of\_squared\_distances')  plt.title('Elbow Method For Optimal k')  plt.show()  *Step 2: Using sum of squared distances to decide cluster number (k)*    *Look at this plot, we choose k = 3* |
| # Euclidean k-means  print("-----Euclidean k-means-----")  n\_clusters=3  km = TimeSeriesKMeans(n\_clusters=n\_clusters, verbose=False, random\_state=seed)  y\_km = km.fit\_predict(X)  print("Euclidean silhoutte: {:.2f}".format(silhouette\_score(X, y\_km, metric="euclidean")))  plt.figure()  for yi in range(n\_clusters):      plt.subplot(3, n\_clusters, yi + 1)      for xx in X[y\_km == yi]:          plt.plot(xx.ravel(), "k-", alpha=.2)      plt.plot(km.cluster\_centers\_[yi].ravel(), "r-")      plt.xlim(0, sz)      plt.ylim(-4, 4)      plt.text(0.55, 0.85,'Cluster %d' % (yi + 1),               transform=plt.gca().transAxes)      if yi == 1:          plt.title("Euclidean $k$-means")  *Step 4: Kmeans using euclidean distance* |
| # DBA-k-means  print("DBA k-means")  dba\_km = TimeSeriesKMeans(n\_clusters=n\_clusters,                            n\_init=2,                            metric="dtw",                            verbose=False,                            max\_iter\_barycenter=10,                            random\_state=seed)  y\_dba\_km = dba\_km.fit\_predict(X)  print("DBA silhoutte: {:.2f}".format(silhouette\_score(X, y\_dba\_km, metric="dtw")))  for yi in range(n\_clusters):      plt.subplot(2, n\_clusters, yi+1)      for xx in X[y\_dba\_km == yi]:          plt.plot(xx.ravel(), "k-", alpha=.2)      plt.plot(dba\_km.cluster\_centers\_[yi].ravel(), "r-")      plt.xlim(0, sz)      plt.ylim(-4, 4)      plt.text(0.55, 0.85,'Cluster %d' % (yi + 1),               transform=plt.gca().transAxes)      if yi == 1:          plt.title("DBA $k$-means")  plt.tight\_layout()  plt.show()  *Step 5: Kmeans using dynamic time warping (dtw) distance* |
| cluster\_labels = dba\_km.labels\_  cluster\_0 = X[cluster\_labels == 0]  cluster\_1 = X[cluster\_labels == 1]  cluster\_2 = X[cluster\_labels == 2]  *Step 6: We separate cluster\_labels with 3 dataset: cluster\_0, cluster\_1, cluster\_2* |

**Now we apply RNN, LSTM, DNN for each dataset:**

|  |
| --- |
| *RNN:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  *Step 2: Apply RNN model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate RNN model* |
| *LSTM:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model=Sequential()  model.add(LSTM(50,return\_sequences=True,input\_shape=(time\_step,1)))  model.add(LSTM(50,return\_sequences=True))  model.add(LSTM(50))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  *Step 2: Apply LSTM model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate LSTM model* |
| *DNN:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model = Sequential()  model.add(Dense(50, activation='relu', input\_shape=(time\_step,)))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mean\_squared\_error')  model.fit(X\_train, y\_train, epochs=100, batch\_size=64)  *Step 2: Apply LSTM model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate LSTM model* |
|  |

* **XLM-USD**

|  |
| --- |
| df = pd.read\_csv('/content/drive/MyDrive/BA/Dataset/XLM-USD.csv',usecols=['Date','Close'])  df = df['Close']  df1 = df.values.reshape(-1, 1)  import numpy  # convert an array of values into a dataset matrix  def create\_dataset(dataset, time\_step=1):    dataX, dataY = [], []    for i in range(len(dataset)-time\_step-1):      a = dataset[i:(i+time\_step), 0]   ###i=0, X=0,1,2,3-----99   Y=100      dataX.append(a)      dataY.append(dataset[i + time\_step, 0])    return numpy.array(dataX), numpy.array(dataY)    time\_step = 100  X, y = create\_dataset(df1, time\_step)  X = X.reshape(X.shape[0],X.shape[1] , 1)  *Step 1: Create dataset with time\_step = 100.* |
| *A picture containing text, font, white, typography  Description automatically generated*  *We have 1916 samples, each sample has a length of 100 and 1 feature* |
| seed = 0  np.random.seed(seed)  X = TimeSeriesScalerMeanVariance().fit\_transform(X)  sz = X.shape[1]  *Step 2: Using TimeSeriesScalerMeanVariance() to scaled time series data* |
| for yi in range(12):      plt.subplot(4, 3, yi + 1)      plt.plot(X[yi].ravel(), "k-", alpha=.2)  *A picture containing line, diagram, text, pattern  Description automatically generated*  *Step 3: Look shape of 12 data* |
| # Using sum of squared distances to decide cluster number  Sum\_of\_squared\_distances = []  K = range(1,8)  for k in K:      km = TimeSeriesKMeans(n\_clusters=k,                            n\_init=2,                            metric="dtw",                            verbose=False,                            max\_iter\_barycenter=10,                            random\_state=0)      km = km.fit(X)      Sum\_of\_squared\_distances.append(km.inertia\_)  plt.plot(K, Sum\_of\_squared\_distances, 'bx-')  plt.xlabel('k')  plt.ylabel('Sum\_of\_squared\_distances')  plt.title('Elbow Method For Optimal k')  plt.show()  *Step 2: Using sum of squared distances to decide cluster number (k)*  *A picture containing line, plot, diagram, screenshot  Description automatically generated*  *Look at this plot, we choose k = 3* |
| # Euclidean k-means  print("-----Euclidean k-means-----")  n\_clusters=3  km = TimeSeriesKMeans(n\_clusters=n\_clusters, verbose=False, random\_state=seed)  y\_km = km.fit\_predict(X)  print("Euclidean silhoutte: {:.2f}".format(silhouette\_score(X, y\_km, metric="euclidean")))  plt.figure()  for yi in range(n\_clusters):      plt.subplot(3, n\_clusters, yi + 1)      for xx in X[y\_km == yi]:          plt.plot(xx.ravel(), "k-", alpha=.2)      plt.plot(km.cluster\_centers\_[yi].ravel(), "r-")      plt.xlim(0, sz)      plt.ylim(-4, 4)      plt.text(0.55, 0.85,'Cluster %d' % (yi + 1),               transform=plt.gca().transAxes)      if yi == 1:          plt.title("Euclidean $k$-means")  *Step 4: Kmeans using euclidean distance* |
| # DBA-k-means  print("DBA k-means")  dba\_km = TimeSeriesKMeans(n\_clusters=n\_clusters,                            n\_init=2,                            metric="dtw",                            verbose=False,                            max\_iter\_barycenter=10,                            random\_state=seed)  y\_dba\_km = dba\_km.fit\_predict(X)  print("DBA silhoutte: {:.2f}".format(silhouette\_score(X, y\_dba\_km, metric="dtw")))  for yi in range(n\_clusters):      plt.subplot(2, n\_clusters, yi+1)      for xx in X[y\_dba\_km == yi]:          plt.plot(xx.ravel(), "k-", alpha=.2)      plt.plot(dba\_km.cluster\_centers\_[yi].ravel(), "r-")      plt.xlim(0, sz)      plt.ylim(-4, 4)      plt.text(0.55, 0.85,'Cluster %d' % (yi + 1),               transform=plt.gca().transAxes)      if yi == 1:          plt.title("DBA $k$-means")  plt.tight\_layout()  plt.show()  *Step 5: Kmeans using dynamic time warping (dtw) distance* |
| cluster\_labels = dba\_km.labels\_  cluster\_0 = X[cluster\_labels == 0]  cluster\_1 = X[cluster\_labels == 1]  cluster\_2 = X[cluster\_labels == 2]  *Step 6: We separate cluster\_labels with 3 dataset: cluster\_0, cluster\_1, cluster\_2* |

**Now we apply RNN, LSTM, DNN for each dataset:**

|  |
| --- |
| *RNN:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model = Sequential()  model.add(SimpleRNN(50, input\_shape=(time\_step, 1)))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  *Step 2: Apply RNN model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate RNN model* |
| *LSTM:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model=Sequential()  model.add(LSTM(50,return\_sequences=True,input\_shape=(time\_step,1)))  model.add(LSTM(50,return\_sequences=True))  model.add(LSTM(50))  model.add(Dense(1))  model.compile(loss='mean\_squared\_error',optimizer='adam')  model.fit(X\_train,y\_train,validation\_data=(X\_test,ytest),epochs=100,batch\_size=64,verbose=1)  *Step 2: Apply LSTM model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate LSTM model* |
| *DNN:*  # Number of samples in cluster\_0  num\_samples = cluster\_0.shape[0]  # Create an array of random indices for the samples  indices = np.random.permutation(num\_samples)  # Calculate the number of samples for each set  num\_train = int(0.7 \* num\_samples)  num\_test = int(0.2 \* num\_samples)  # Split the data into train, test, validate sets  train\_indices = indices[:num\_train]  test\_indices = indices[num\_train:num\_train+num\_test]  val\_indices = indices[num\_train+num\_test:]  # Create sublists to store the samples and corresponding labels  X\_train = []  y\_train = []  X\_test = []  ytest = []  X\_val = []  yval = []  # Create X\_train and y\_train  for i in train\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_train.append(sample)      y\_train.append(label)  # Create X\_test and ytest  for i in test\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_test.append(sample)      ytest.append(label)  # Create X\_val and yval  for i in val\_indices:      sample = cluster\_1[i]      label = cluster\_1[i][-1]  # Take the last value in each data sequence      X\_val.append(sample)      yval.append(label)  # Convert lists to numpy arrays  X\_train = np.array(X\_train)  y\_train = np.array(y\_train).reshape(-1)  X\_test = np.array(X\_test)  ytest = np.array(ytest).reshape(-1)  X\_val = np.array(X\_val)  yval = np.array(yval).reshape(-1)  *Step 1: create X\_train, y\_train, X\_test, ytest, X\_val, yval with cluster\_0* |
| model = Sequential()  model.add(Dense(50, activation='relu', input\_shape=(time\_step,)))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mean\_squared\_error')  model.fit(X\_train, y\_train, epochs=100, batch\_size=64)  *Step 2: Apply LSTM model to predict* |
| train\_predict=model.predict(X\_train)  y\_pred=model.predict(X\_test)  y\_pred\_val=model.predict(X\_val)  *Step 3: Predict X\_train, X\_test, X\_val* |
| print('---------VALIDATE----------')  valid\_mae = np.mean(np.abs(y\_pred\_val - yval))  print('MAE:', valid\_mae)  valid\_rmse = np.sqrt(np.mean((y\_pred\_val - yval)\*\*2))  print('RMSE:', valid\_rmse)  print('--------------------------')  print('---------TEST----------')  test\_mae = np.mean(np.abs(y\_pred - ytest))  print('MAE:', test\_mae)  test\_rmse = np.sqrt(np.mean((y\_pred - ytest)\*\*2))  print('RMSE:', test\_rmse)  print('-------------------')  *Step 4: Evaluate LSTM model* |

# RESULT

## Materials

|  |  |  |  |
| --- | --- | --- | --- |
| Meansure | TRX-USD | DOGE-USD | XLM-USD |
| Mean | 0.047120 | 0.062898 | 0.184427 |
| Std | 0.029368 | 0.094856 | 0.133435 |
| Min | 0.002073 | 0.001537 | 0.033441 |
| Max | 0.220555 | 0.684777 | 0.896227 |
| Histogram |  |  |  |
| Box plot |  |  |  |

TABLE 2. Descriptive statistics table

## Evaluation

### MAE (Mean Absolute Error)

MAE measures the average magnitude of the errors made by a model in predicting the target variable. It is calculated by taking the average of the absolute differences between the predicted values and the actual values. MAE is useful because it gives equal weight to all errors without considering their direction (positive or negative). The formula for MAE is as follows:

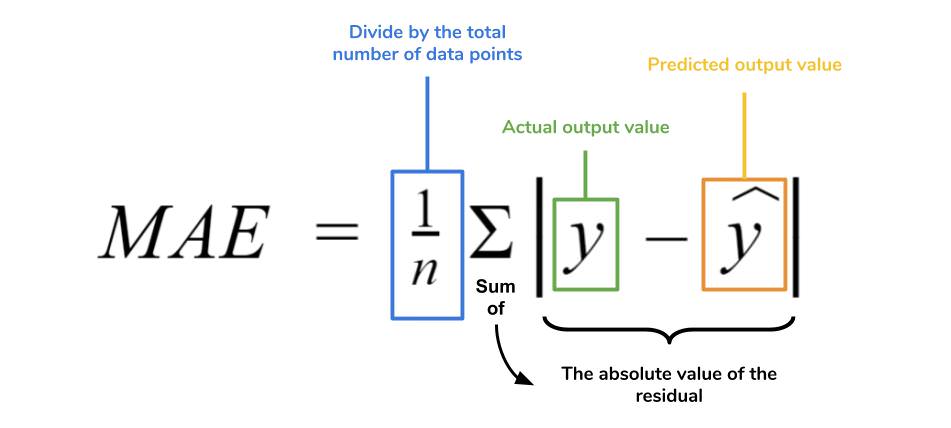


FIGURE . Formula for calculating MAE

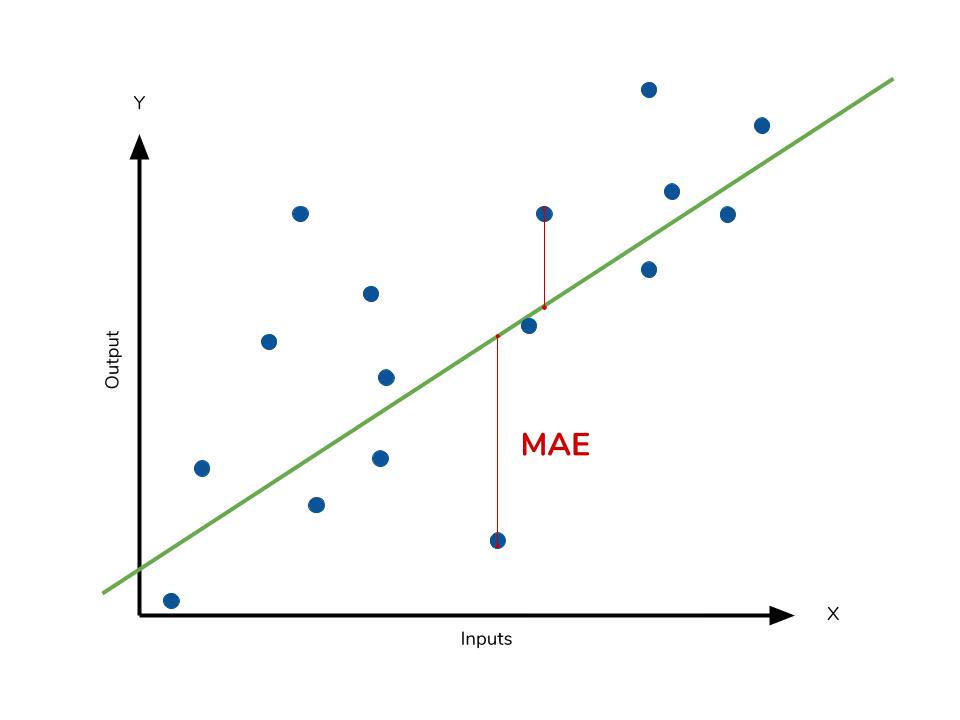


FIGURE . Graphical description of the MAE

The lower the MAE, the better the model's performance, with a value of 0 indicating a perfect prediction.

### RMSE (Root Mean Square Error)

RMSE is another commonly used metric that measures the square root of the average of the squared differences between the predicted values and the actual values. RMSE penalizes larger errors more heavily than MAE since it squares the errors before taking the average. The formula for RMSE is as follows:

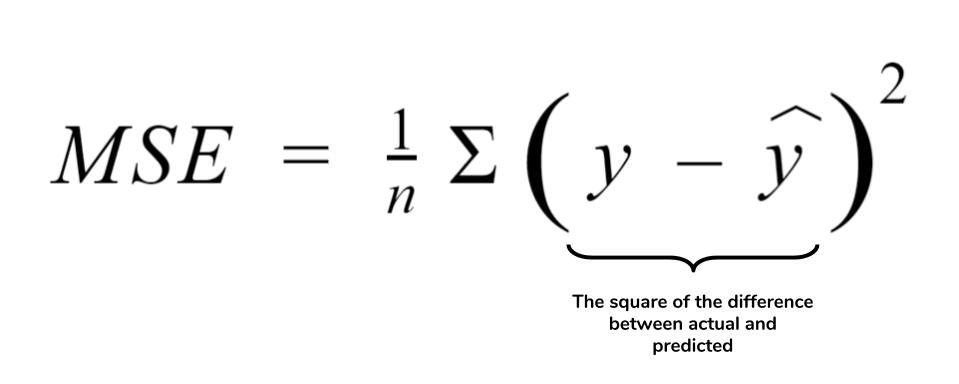


FIGURE . Formula for calculating RMSE

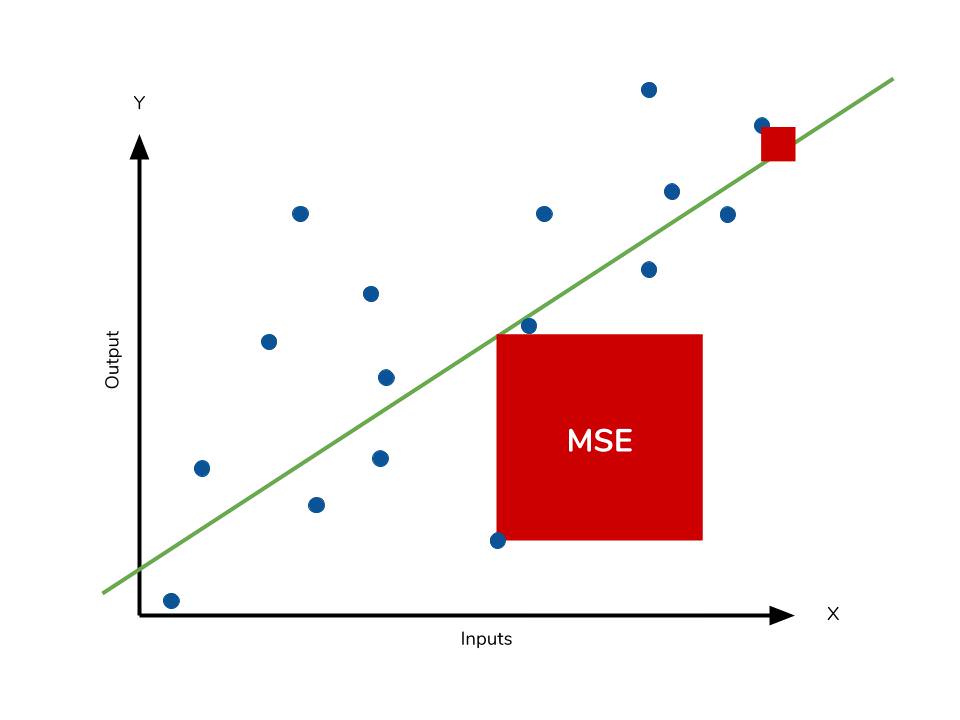


FIGURE . Graphical description of the RMSE

Like MAE, lower values of RMSE indicate better model performance, with a value of 0 indicating a perfect prediction.

### MAPE (Mean Absolute Percentage Error)

MAPE measures the average percentage difference between the predicted and actual values. It is commonly used when the relative error between the predicted and actual values is more important than the absolute error. MAPE is calculated using the following formula:

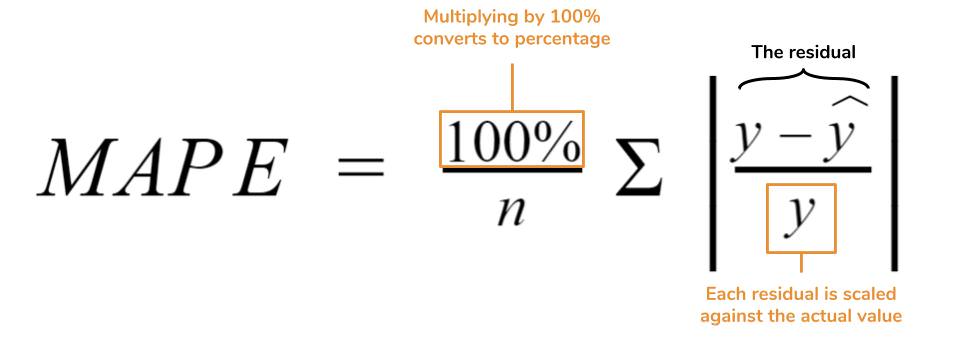


FIGURE . Formula for calculating MAPE

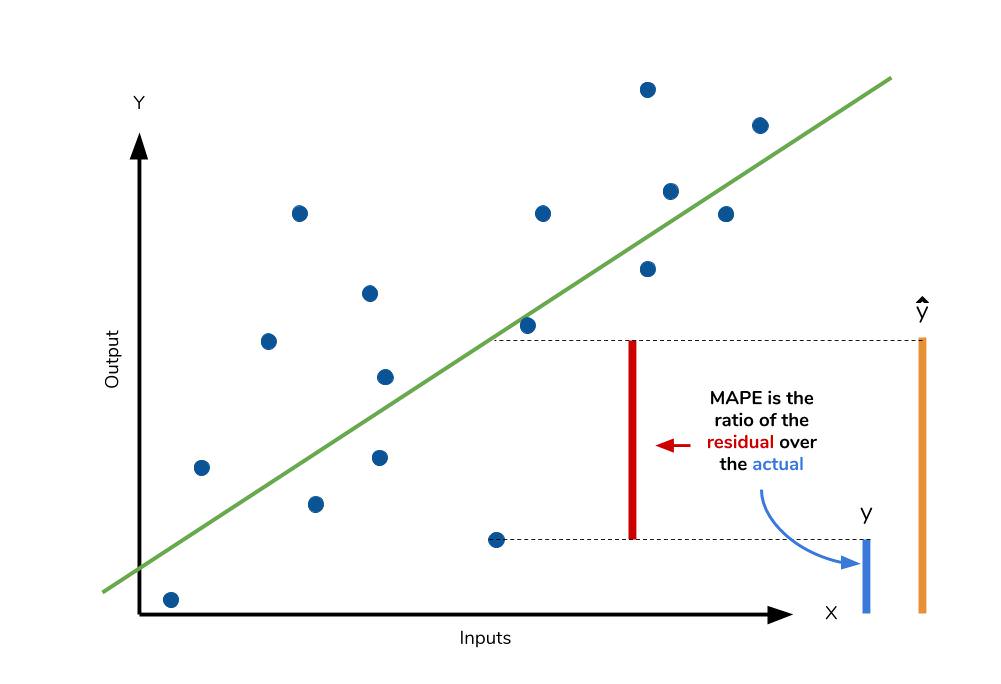


FIGURE . Graphical description of the MAPE

It is important to note that MAPE can result in undefined values or infinity if there are actual values that are zero or close to zero. Additionally, MAPE is often expressed as a percentage.

Lower values of MAPE indicate better model performance, with a value of 0 indicating a perfect prediction.

## Result

### SINGLE MODEL

* **TRX-USD Validation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate | MAPE Validate |
| RNN | **7:2:1** | 0.235 | 0.236 | 77.05 |
| **5:3:2** | 0.217 | 0.219 | 77.2 |
| **6:2:2** | 0.216 | 0.218 | 76.9 |
| LSTM | **7:2:1** | 0.236 | 0.237 | 77.32 |
| **5:3:2** | 0.216 | 0.219 | 77.11 |
|  | **6:2:2** | 0.032 | 0.039 | 26.62 |
| DNN | **7:2:1** | **0.002** | **0.002** | **2.547** |
| **5:3:2** | **0.002** | **0.003** | **3.494** |
|  | **6:2:2** | **0.002** | **0.003** | **3.868** |
| ARIMA | **7:2:1** | **0.033** | **0.034** | **54.299** |
| **5:3:2** | **0.017** | **0.02** | **25.642** |
|  | **6:2:2** | **0.011** | **0.013** | **15.528** |
| ARIMAX | **7:2:1** | 0.033 | 0.034 | 54.299 |
| **5:3:2** | 0.017 | 0.02 | 25.642 |
|  | **6:2:2** | 0.011 | 0.013 | 15.528 |
| KNN | **7:2:1** | 0.03 | 0.031 | 48.707 |
| **5:3:2** | 0.038 | 0.038 | 57.167 |
|  | **6:2:2** | 0.011 | 0.013 | 16.269 |
| LR | **7:2:1** | 0.006 | 0.007 | 8.897 |
| **5:3:2** | 0.089 | 0.089 | 136.418 |
|  | **6:2:2** | 0.058 | 0.058 | 88.746 |

* **TRX-USD Test**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test | MAPE Test |
| RNN | **7:2:1** | 0.224 | 0.226 | 76.81 |
| **5:3:2** | 0.258 | 0.288 | 73.83 |
|  | **6:2:2** | 0.271 | 0.283 | 76.19 |
| LSTM | **7:2:1** | 0.225 | 0.227 | 77.14 |
| **5:3:2** | 0.258 | 0.288 | 73.6 |
|  | **6:2:2** | 0.219 | 0.221 | 78.2 |
| DNN | **7:2:1** | **0.003** | **0.004** | **4.367** |
| **5:3:2** | **0.006** | **0.01** | **7.846** |
|  | **6:2:2** | **0.005** | **0.006** | **6.165** |
| ARIMA | **7:2:1** | **0.027** | **0.029** | **41.365** |
| **5:3:2** | **0.023** | **0.032** | **33.778** |
|  | **6:2:2** | **0.028** | **0.037** | **29.546** |
| ARIMAX | **7:2:1** | 0.027 | 0.029 | 41.365 |
| **5:3:2** | 0.023 | 0.032 | 33.778 |
|  | **6:2:2** | 0.028 | 0.037 | 29.546 |
| KNN | **7:2:1** | 0.024 | 0.026 | 36.38 |
| **5:3:2** | 0.032 | 0.04 | 44.581 |
|  | **6:2:2** | 0.024 | 0.033 | 24.08 |
| LR | **7:2:1** | 0.016 | 0.023 | 20.142 |
| **5:3:2** | 0.089 | 0.089 | 101.871 |
|  | **6:2:2** | 0.069 | 0.069 | 81.606 |

* **DOGE-USD**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate | MAPE Validate |
| RNN | **7:2:1** | **0.015** | **0.178** | **35.16** |
| **5:3:2** | **0.039** | **0.045** | **33.49** |
|  | **6:2:2** | **0.038** | **0.045** | **32.532** |
| LSTM | **7:2:1** | **0.021** | **0.024** | **18.7** |
| **5:3:2** | **0.026** | **0.033** | **21.91** |
|  | **6:2:2** | **0.032** | **0.038** | **26.62** |
| DNN | **7:2:1** | 0.041 | 0.041 | 53.307 |
| **5:3:2** | 0.005 | 0.051 | 64.735 |
|  | **6:2:2** | 0.011 | 0.015 | 13.021 |
| ARIMA | **7:2:1** | 0.153 | 0.153 | 192.462 |
| **5:3:2** | 0.075 | 0.076 | 96.333 |
|  | **6:2:2** | 0.027 | 0.03 | 32.077 |
| ARIMAX | **7:2:1** | 0.153 | 0.153 | 192.462 |
| **5:3:2** | 0.017 | 0.02 | 25.642 |
|  | **6:2:2** | 0.027 | 0.03 | 32.077 |
| KNN | **7:2:1** | 0.143 | 0.143 | 180.44 |
| **5:3:2** | 0.074 | 0.076 | 95.561 |
|  | **6:2:2** | 0.021 | 0.026 | 25.199 |
| LR | **7:2:1** | 0.133 | 0.134 | 168.428 |
| **5:3:2** | 0.078 | 0.08 | 100.996 |
|  | **6:2:2** | 0.06 | 0.062 | 76.469 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test | MAPE Test |
| RNN | **7:2:1** | **0.048** | **0.058** | **35.16** |
| **5:3:2** | **0.172** | **0.221** | **300.49** |
|  | **6:2:2** | **0.111** | **0.136** | **35.81** |
| LSTM | **7:2:1** | 0.055 | 0.068 | 38.68 |
| **5:3:2** | 0.169 | 0.218 | 339.5 |
|  | **6:2:2** | 0.108 | 0.134 | 34.43 |
| DNN | **7:2:1** | **0.046** | **0.048** | **52.762** |
| **5:3:2** | **0.115** | **0.141** | **59.124** |
|  | **6:2:2** | **0.057** | **0.067** | **28.141** |
| ARIMA | **7:2:1** | 0.114 | 0.126 | 137.832 |
| **5:3:2** | 0.15 | 0.195 | 79.616 |
|  | **6:2:2** | 0.169 | 0.196 | 71.293 |
| ARIMAX | **7:2:1** | 0.114 | 0.126 | 137.832 |
| **5:3:2** | 0.023 | 0.032 | 33.778 |
|  | **6:2:2** | 0.169 | 0.196 | 71.293 |
| KNN | **7:2:1** | 0.106 | 0.117 | 128.743 |
| **5:3:2** | 0.149 | 0.195 | 80.675 |
|  | **6:2:2** | 0.163 | 0.19 | 68.076 |
| LR | **7:2:1** | 0.077 | 0.09 | 92.26 |
| **5:3:2** | 0.152 | 0.197 | 88.538 |
|  | **6:2:2** | 0.206 | 0.229 | 92.234 |

* **XLM-USD**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate | MAPE Validate |
| RNN | **7:2:1** | **0.025** | **0.278** | **38.79** |
| **5:3:2** | **0.023** | **0.027** | **35.35** |
|  | **6:2:2** | **0.027** | **0.031** | **42.01** |
| LSTM | **7:2:1** | 0.025 | 0.027 | 38.49 |
| **5:3:2** | 0.027 | 0.032 | 42.7 |
|  | **6:2:2** | 0.026 | 0.03 | 40.24 |
| DNN | **7:2:1** | **0.004** | **0.005** | **4.726** |
| **5:3:2** | **0.003** | **0.004** | **3.689** |
|  | **6:2:2** | **0.004** | **0.006** | **4.262** |
| ARIMA | **7:2:1** | 0.255 | 0.255 | 290.214 |
|  | **5:3:2** | 0.024 | 0.031 | 21.459 |
|  | **6:2:2** | 0.011 | 0.013 | 15.528 |
| ARIMAX | **7:2:1** | 0.255 | 0.255 | 290.214 |
|  | **5:3:2** | 0.017 | 0.02 | 25.642 |
|  | **6:2:2** | 0.27 | 0.271 | 273.43 |
| KNN | **7:2:1** | 0.217 | 0.217 | 247.918 |
|  | **5:3:2** | 0.015 | 0.019 | 14.835 |
|  | **6:2:2** | 0.299 | 0.3 | 302.112 |
| LR | **7:2:1** | 0.127 | 0.128 | 145.49 |
| **5:3:2** | 0.377 | 0.378 | 378.683 |
| **6:2:2** | 0.108 | 0.109 | 107.23 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test | MAPE Test |
| RNN | **7:2:1** | **0.053** | **0.066** | **46.806** |
| **5:3:2** | **0.131** | **0.169** | **43.16** |
| **6:2:2** | **0.082** | **0.1** | **30.94** |
| LSTM | **7:2:1** | 0.055 | 0.069 | 49.04 |
| **5:3:2** | 0.136 | 0.176 | 46.39 |
| **6:2:2** | 0.081 | 0.1 | 30.88 |
| DNN | **7:2:1** | **0.009** | **0.011** | **6.021** |
| **5:3:2** | **0.047** | **0.072** | **12.788** |
| **6:2:2** | **0.021** | **0.026** | **7.813** |
| ARIMA | **7:2:1** | 0.16 | 0.178 | 120.219 |
| **5:3:2** | 0.204 | 0.244 | 62.529 |
| **6:2:2** | 0.028 | 0.037 | 29.546 |
| ARIMAX | **7:2:1** | 0.16 | 0.178 | 120.219 |
| **5:3:2** | 0.023 | 0.032 | 33.778 |
| **6:2:2** | 0.106 | 0.128 | 40.133 |
| KNN | **7:2:1** | 0.131 | 0.146 | 98.947 |
| **5:3:2** | 0.188 | 0.226 | 58.926 |
| **6:2:2** | 0.102 | 0.124 | 38.698 |
| LR | **7:2:1** | 0.079 | 0.091 | 52.16 |
| **5:3:2** | 0.391 | 0.42 | 142.836 |
| **6:2:2** | 0.266 | 0.285 | 83.139 |

### HYBRID MODEL

*Hybrid model based on*  *TIME SERIES ANOMALY DETECTION with ARIMA,*

* **TRX-USD**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate |
| Time Series Anomaly detection | **7:2:1** | 0.001 | 0.024 |
| **5:3:2** | 0.004 | 0.063 |
|  | **6:2:2** | 0.004 | 0.063 |

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test |
| Time Series Anomaly detection | **7:2:1** | 0.002 | 0.04 |
| **5:3:2** | 0.022 | 0.148 |
| **6:2:2** | 0.022 | 0.148 |

* **DOGE-USD**

|  |  |  |  |
| --- | --- | --- | --- |
| * Method | Ratio | MAE Validate | RMSE Validate |
| Time Series Anomaly detection | **7:2:1** | 0.016 | 0.128 |
| **5:3:2** | 0.109 | 0.33 |
|  | **6:2:2** | 0.037 | 0.192 |

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test |
| Time Series Anomaly detection | **7:2:1** | 0.025 | 0.158 |
| **5:3:2** | 0.169 | 0.411 |
| **6:2:2** | 0.134 | 0.366 |

* **XLM-USD**

|  |  |  |  |
| --- | --- | --- | --- |
| * Method | Ratio | MAE Validate | RMSE Validate |
| Time Series Anomaly detection | **7:2:1** | 0.005 | 0.068 |
| **5:3:2** | 0.005 | 0.068 |
|  | **6:2:2** | 0.002 | 0.048 |

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test |
| Time Series Anomaly detection  detection+Arima | **7:2:1** | 0.017 | 0.131 |
| **5:3:2** | 0.084 | 0.29 |
| **6:2:2** | 0.105 | 0.324 |

*Hybrid model based on TIME SERIES CLUSTERING with RNN, LSTM, DNN,*

* **TRX-USD**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate |
| KMeans + RNN | **7:2:1** | 0.915 | 1.173 |
| **5:3:2** | 0.933 | 1.193 |
|  | **6:2:2** | 0.993 | 1.27 |
| KMeans + LSTM | **7:2:1** | 1.032 | 1.312 |
| **5:3:2** | 0.959 | 1.237 |
|  | **6:2:2** | 0.89 | 1.167 |
| KMeans + DNN | **7:2:1** | 1.241 | 1.493 |
| **5:3:2** | 1.019 | 1.303 |
|  | **6:2:2** | 1.195 | 1.438 |

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test |
| KMeans + RNN | **7:2:1** | 1.006 | 1.293 |
| **5:3:2** | 1.009 | 1.285 |
| **6:2:2** | 0.96 | 1.224 |
| KMeans + LSTM | **7:2:1** | 0.955 | 1.232 |
| **5:3:2** | 0.987 | 1.287 |
| **6:2:2** | 0.938 | 1.203 |
| KMeans + DNN | **7:2:1** | 1.306 | 1.578 |
| **5:3:2** | 0.992 | 1.265 |
| **6:2:2** | 1.247 | 1.515 |

* **DOGE-USD**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate |
| KMeans + RNN | **7:2:1** | 1.075 | 1.232 |
| **5:3:2** | 0.97 | 1.244 |
|  | **6:2:2** | 0.912 | 1.206 |
| KMeans + LSTM | **7:2:1** | 0.957 | 1.213 |
| **5:3:2** | 0.968 | 1.271 |
|  | **6:2:2** | 0.949 | 1.236 |
| KMeans + DNN | **7:2:1** | 1.107 | 1.43 |
| **5:3:2** | 0.995 | 1.268 |
|  | **6:2:2** | 1.195 | 0.004 |

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Test | RMSE Test |
| KMeans + RNN | **7:2:1** | 0.943 | 1.232 |
| **5:3:2** | 1.025 | 1.371 |
|  | **6:2:2** | 0.928 | 1.21 |
| KMeans + LSTM | **7:2:1** | 0.914 | 1.225 |
| **5:3:2** | 0.992 | 1.29 |
|  | **6:2:2** | 0.963 | 1.294 |
| KMeans + DNN | **7:2:1** | 1.082 | 1.382 |
| **5:3:2** | 1.059 | 0.008 |
|  | **6:2:2** | 1.196 | 1.466 |

* **XLM-USD**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Ratio | MAE Validate | RMSE Validate |
| KMeans + RNN | **7:2:1** | 0.96 | 1.215 |
| **5:3:2** | 0.905 | 1.139 |
|  | **6:2:2** | 0.92 | 1.2 |
| KMeans + LSTM | **7:2:1** | 1.048 | 1.347 |
| **5:3:2** | 0.954 | 1.22 |
|  | **6:2:2** | 1.028 | 1.329 |
| KMeans + DNN | **7:2:1** | 1.201 | 1.486 |
| **5:3:2** | 1.032 | 1.327 |
|  | **6:2:2** | 1.09 | 1.361 |
| **Method** | **Ratio** | **MAE Test** | **RMSE Test** |
| KMeans + RNN | **7:2:1** | 0.91 | 1.171 |
|  | **5:3:2** | 0.96 | 1.245 |
|  | **6:2:2** | 0.924 | 1.19 |
| KMeans + LSTM | **7:2:1** | 1.024 | 1.323 |
|  | **5:3:2** | 0.939 | 1.197 |
|  | **6:2:2** | 0.898 | 1.165 |
| KMeans + DNN | **7:2:1** | 1.168 | 1.389 |
|  | **5:3:2** | 1.012 | 1.265 |
|  | **6:2:2** | 1 | 1.235 |

## Predict the price of cryptocurrencies

### TRX-USD

ARIMA

|  |
| --- |
| Ratio 7:2:1 |
| Ratio 5:3:2 |
| Ratio 6:2:2 |

DNN

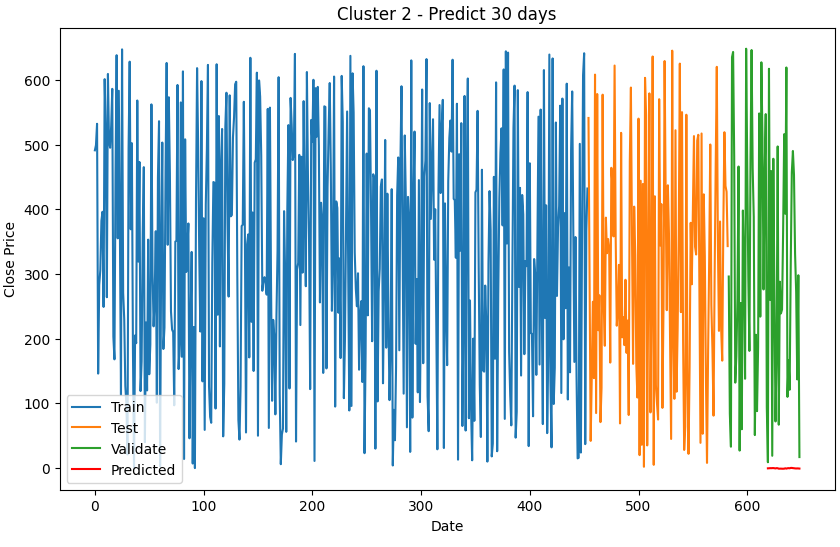
|  |
| --- |
| Ratio 7:2:1 |
| Ratio 5:3:2 |
| Ratio 6:2:2 |

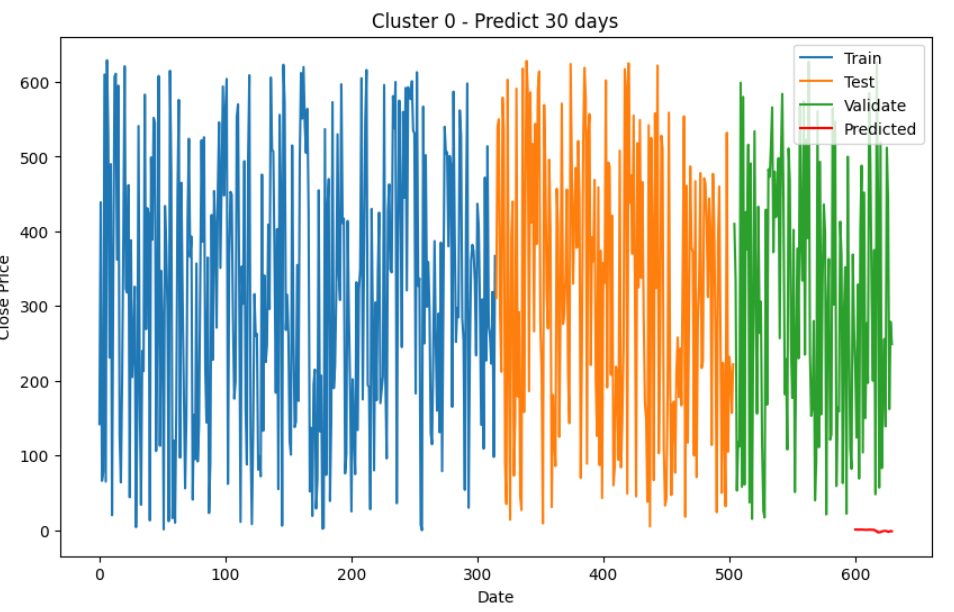
Kmeans + RNN:

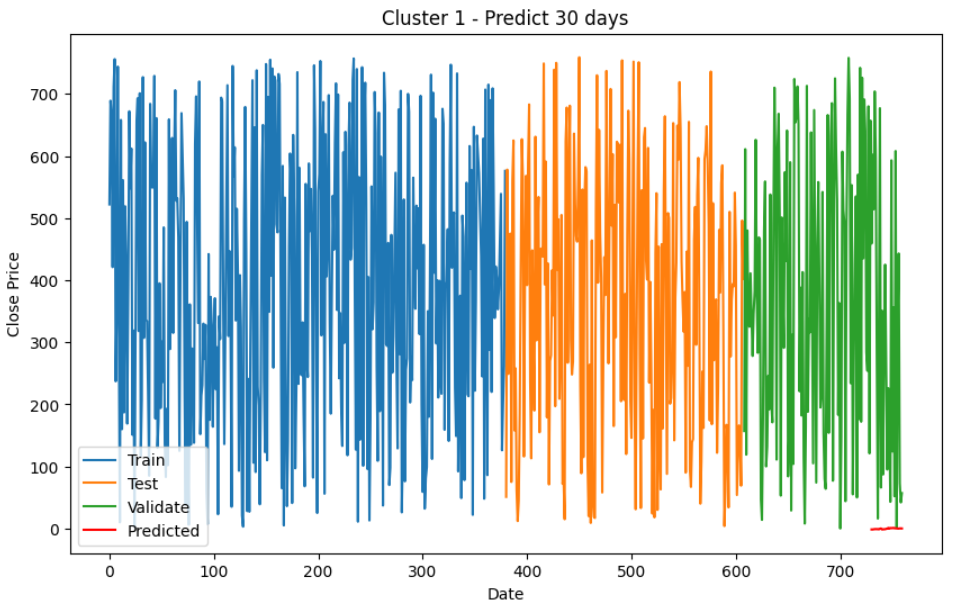
A picture containing text, screenshot, colorfulness, plot

Description automatically generatedA graph of blue and orange lines

Description automatically generated with low confidence

*Ratio 7-2-1*

**

**

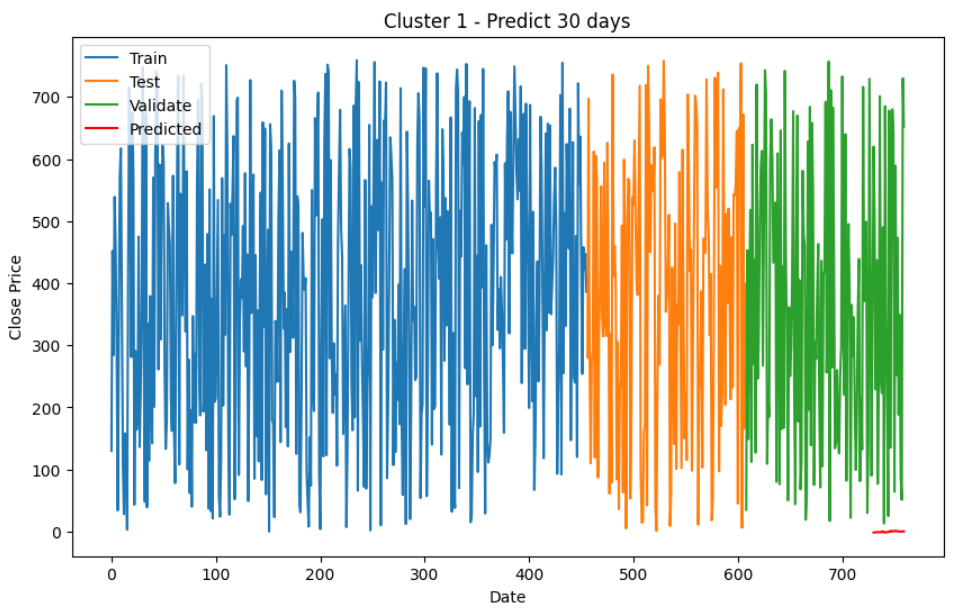
*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

*Ratio 5-3-2*

*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

**

Ratio 6-2-2

### DOGE-USD

RNN

|  |
| --- |
| Ratio 7:2:1 |
| Ratio 5:3:2 |
| Ratio 6:2:2 |

DNN

|  |
| --- |
| Ratio 7:2:1 |
| Ratio 5:3:2 |
| Ratio 6:2:2 |

Kmeans + LSTM:

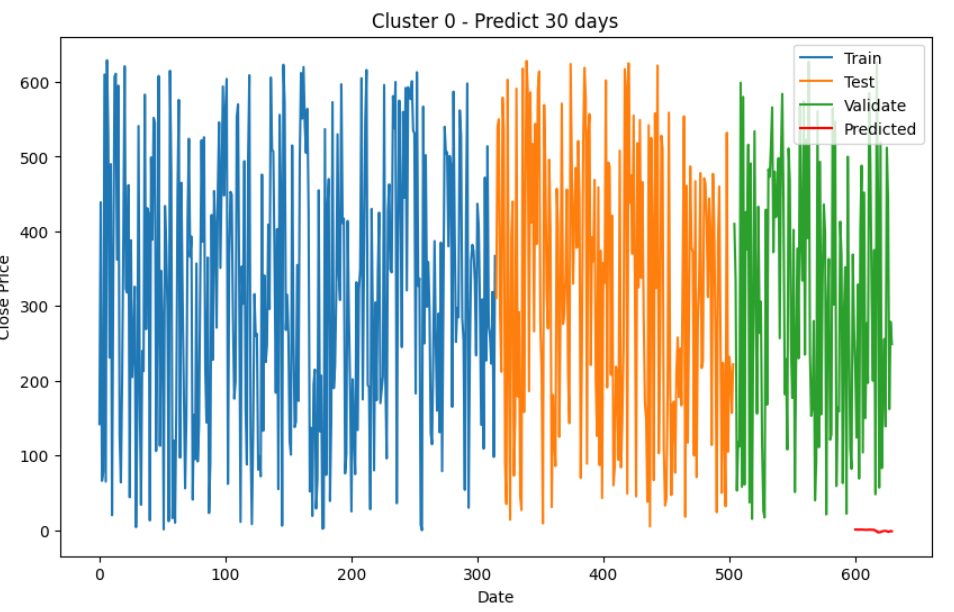
A picture containing text, screenshot, colorfulness, plot

Description automatically generatedA graph of blue and orange lines

Description automatically generated with low confidence

A graph of blue and orange lines

Description automatically generated with low confidence*Ratio 7-2-1*

**

*A picture containing screenshot, text, colorfulness, plot

Description automatically generated*

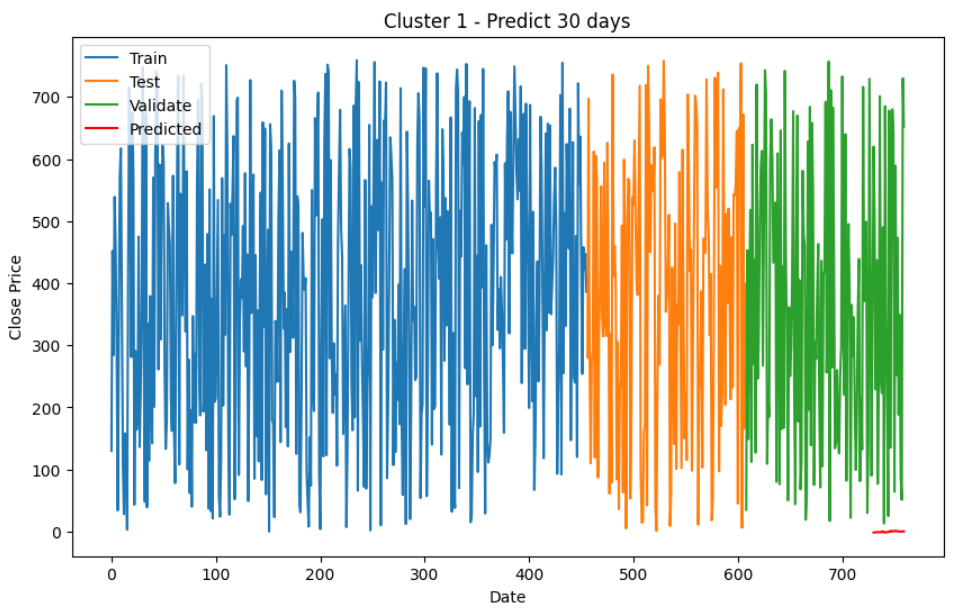
*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

*Ratio 5-3-2*

*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

**

Ratio 6-2-2

### XLM-USD

RNN

|  |
| --- |
| Ratio 7:2:1 |
| Ratio 5:3:2 |
| Ratio 6:2:2 |

DNN

|  |
| --- |
| Ratio 7:2:1 |
| Ratio 5:3:2 |
| Ratio 6:2:2 |

Kmeans + RNN:

A picture containing text, screenshot, colorfulness, plot

Description automatically generatedA graph of blue and orange lines

Description automatically generated with low confidence

A graph of blue and orange lines

Description automatically generated with low confidence*Ratio 7-2-1*

*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

*A picture containing screenshot, text, colorfulness, plot

Description automatically generated*

*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

*Ratio 5-3-2*

*A picture containing text, screenshot, colorfulness, plot

Description automatically generated*

*A picture containing screenshot, text, colorfulness, plot

Description automatically generated*

Ratio 6-2-2

# CONCLUSION

Through rigorous experiments using real-world cryptocurrency datasets, we demonstrated the effectiveness of these machine learning algorithms in predicting cryptocurrency prices. Our models outperformed traditional statistical methods and baseline machine learning approaches, providing valuable insights for investors in navigating the volatile cryptocurrency markets.

We found that the selection of input features, such as historical price data and technical indicators, played a significant role in improving the accuracy of predictions. Furthermore, the incorporation of advanced algorithms, particularly RNN and LSTM, proved crucial in capturing the temporal dependencies and nonlinear relationships inherent in cryptocurrency price data.

However, it is important to note that cryptocurrency markets are highly volatile and subject to various external factors, such as regulatory changes and market sentiment, which can impact the accuracy of predictions. Therefore, prudent risk management strategies should accompany any investment decisions based on these predictions.

The findings of this research contribute to the growing body of knowledge in cryptocurrency price prediction. By leveraging advanced machine learning techniques, our framework offers a reliable tool for making informed investment decisions in cryptocurrency markets. Furthermore, the insights gained from this study can assist investors in identifying potential investment opportunities and devising risk management strategies.

It is worth mentioning that the field of cryptocurrency price prediction is still evolving, and there is ample room for further research. Future studies can explore the integration of additional data sources, such as social media sentiment analysis and macroeconomic indicators, to enhance prediction accuracy. Additionally, the development of ensemble models and the exploration of deep learning architectures can further improve the predictive performance.

In conclusion, accurate prediction of cryptocurrency prices is a challenging yet crucial task for investors. The framework presented in this study provides a foundation for effective prediction using time series analysis and machine learning techniques. However, it is essential to exercise caution, continuously monitor market conditions, and apply sound risk management practices when utilizing these predictions in real-world investment decisions.

# REFERENCES

[1] Y. Zhu, ‘Stock price prediction using the RNN model’, *J. Phys. Conf. Ser.*, vol. 1650, no. 3, p. 032103, Oct. 2020, doi: 10.1088/1742-6596/1650/3/032103.

[2] G. Petneházi, *Recurrent Neural Networks for Time Series Forecasting*. 2018.

[3] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, ‘A survey on long short-term memory networks for time series prediction’, *Procedia CIRP*, vol. 99, pp. 650–655, Jan. 2021, doi: 10.1016/j.procir.2021.03.088.

[4] A. Bozanta *et al.*, ‘Time Series Clustering for Grouping Products Based on Price and Sales Patterns’. arXiv, Apr. 18, 2022. doi: 10.48550/arXiv.2204.08334.

[5] S. Aghabozorgi, A. Seyed Shirkhorshidi, and T. Ying Wah, ‘Time-series clustering – A decade review’, *Inf. Syst.*, vol. 53, pp. 16–38, Oct. 2015, doi: 10.1016/j.is.2015.04.007.

[6] A. Javed, B. Lee, and D. Rizzo, ‘A benchmark study on time series clustering’, *Mach. Learn. Appl.*, vol. 1, Sep. 2020, doi: 10.1016/j.mlwa.2020.100001.

[7] F. Martínez-Álvarez, A. Troncoso, J. Riquelme, and J. Aguilar-Ruiz, ‘Energy Time Series Forecasting Based on Pattern Sequence Similarity’, *Knowl. Data Eng. IEEE Trans. On*, vol. 23, pp. 1230–1243, Sep. 2011, doi: 10.1109/TKDE.2010.227.

[8] Williams, R. J. and Zipser, R. A., A learning algorithm for continually training neural networks. Neural computation, vol. 1, 1989, pp. 270–280.

[9] Yuhan Li Beijing University of Technology, Beijing 100083, China , “Stock Price Prediction based on Multiple Regression Models”,2023.

[10] Mojtaba Nabipour, Pooyan Nayyeri, Hamed Jabani, Amir Mosavi, “Deep learning for Stock Market Prediction”,2019.

[11] Cyril Bachelard, Apostolos Chalkis, Vissarion Fisikopoulos, and Elias Tsigaridas,Faculty of Business and Economics (HEC), Department of Operations, University of Lausanne, Switzerland, National & Kapodistrian University of Athens, Greece GeomScale org Inria Paris and IMJ-PRG, Sorbonne Universite and Paris Universite, “Randomized geometric tools for anomaly detection in stock markets”,2019.

[12] Xiaonan Zuo , “2nd International Conference on Financial Market and Enterprises Management Engineering (FMEME 2023)”, vol. 44, p. 691-692,

doi: 10.54691/bcpbm.v44i.4919, Jun. 2022.

[13]“k-nearest neighbors algorithm,” Wikipedia. Jun. 18, 2023. Accessed: Jun. 21, 2023. [Online].

[14]“What Is an ARIMAX Model?,” 365 Data Science, Jun. 25, 2020. (accessed Jun. 21, 2023).

[15] R. Chow, “K-Nearest Neighbors Algorithm: Classification and Regression Star,” *History of Data Science*, Aug. 27, 2021. https://www.historyofdatascience.com/k-nearest-neighbors-algorithm-classification-and-regression-star/ (accessed Jun. 21, 2023).

[16] M. Chen, N. Narwal, and M. Schultz, “Predicting Price Changes in Ethereum”.

[17] R. Sarpong-Streetor, R. Sokkalingam, O. Mahmod, H. Daud, and D. Asamoah Owusu, “ARIMAX Modelling of Ron97 Price with Crude Oil Price as an Exogenous Variable in Malaysian,” 2022, pp. 679–691. doi: 10.1007/978-981-16-4513-6\_59.

[18] S. Alahmari, “Using Nonlinear Machine Learning Algorithms to Predict the Price of Cryptocurrencies,” Feb. 2020.

[19] A. C. Harvey, “ARIMA Models,” in Time Series and Statistics, J. Eatwell, M. Milgate, and P. Newman, Eds., in The New Palgrave. London: Palgrave Macmillan UK, 1990, pp. 22–24. doi: 10.1007/978-1-349-20865-4\_2.

[20] F. Islam and M. A. Imteaz, “The effectiveness of ARIMAX model for prediction of summer rainfall in northwest Western Australia,” IOP Conf. Ser. Mater. Sci. Eng., vol. 1067, no. 1, p. 012037, Feb. 2021, doi: 10.1088/1757-899X/1067/1/012037.

[21] “A guide on regression error metrics (MSE, RMSE, MAE, MAPE, sMAPE, MPE) with Python code,” Amir Masoud Sefidian - Sefidian Academy, Aug. 18, 2022. https://sefidian.com/2022/08/18/a-guide-on-regression-error-metrics-with-python-code/ (accessed Jun. 21, 2023).

[22] “Basic Understanding of ARIMA/SARIMA vs Auto ARIMA/SARIMA using Covid-19 Prediction,” Predict the future, Jun. 28, 2020. https://techairesearch.com/basic-understanding-of-arima-sarima-vs-auto-arima-sarima-using-covid-19-prediction/ (accessed Jun. 19, 2023).