HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

COVID-19 INFECTION PREDICTION METHODS: A COMPREHENSIVE ANALYSIS

Course: Applied Statistics and Experimental Design

Supervisor: Assoc. Prof. Nguyen Linh Giang

Authors:	Student ID:
Nguyen Duc An	20225432
Nguyen Minh Duong	20225439
Nguyen Minh Duc	20225437
Truong Tuan Vinh	20225464

Hanoi, June 2024

ABSTRACT

Coronavirus disease 2019 (COVID-19), a contagious disease caused by the coronavirus SARS-CoV-2, was first identified in Wuhan, China (December 2019). Within several years since its emergence, the whole world has struggled in flattening the curve and rolling out the suitable preventive measures. This project aims to evaluate the efficacy of various COVID-19 prediction methods, including models already widely in use and ones that are still open for experiment. We then compare how well these methods work according to various criteria and attempt give a theoretical explanation of the results. Our findings indicate that ...

Mục lục

1 Introduction

1.1 The need for correct predictions

The COVID-19 has had catastrophic impacts on the global economy, public health, and more. A report published in JAMA [1] estimated the economic losses due to COVID-19 to be $ 16 Trillion in the USA alone. Another report by Front Public Health [2] claimed that the pandemic has disrupted the supply chain, heavily restricting the money flow and affecting the global economy's health in the long run. To minimise the adverse impacts of the pandemic, policy makers need to find the right balance when taking preventive measures. Having too strict policies in place would unnecessarily hurt the economy and welfare of the people, while the opposite could quickly lead to the public health care system being overloaded.

A key factor in deciding which measures to take is the current trends of pandemic growth and its prediction in the near future. Billions of US dollars would have been saved had more precise methods of prediction been available. For this reason, we need to explore our alternatives for predicting the infection count to carry out cost-effective interventions.

1.2 Overview of models in use

Sophisticated mathematical models should be able to reflect future disease growth patterns and address the effects of different interventions (such as contact tracing, compulsory hand-washing and facial masks). Throughout the history of epidemiology, various types of models have been developed, such as Sub-exponential growth and Agent-based Models (ABMs). The following section discusses a class of models called Compartmental models.

1.3 How the Compartmental models work

First emerged in the 1920s, the compartmental modelling technique divides the general population into several exhausting, non-overlapping labels (or 'compartments') - for example **S**, **I**, and **R** (**S**usceptible, **I**nfectious, and **R**ecovered). These models are then usually run with ordinary differential equations to describe the dynamic transition between compartments in each time step. Stochastic frameworks can also be used to account for the random factors, making the outputs more realistic but more complicated to analyze.

One of the simplest compartmental models is the **SIR model** which sets the basis for other more complex derivatives. In this model, **s**usceptible individuals may transition into the **i**nfected compartment, and the infected may fall into the **r**emoved category by either having died or having recovered and gained immunity. This model assumes that recovered people have obtained lasting immunity, and the birth rate is equal to the natural death rate so that the total population is relatively stable over time.

Depending on the specifics of a disease, different derivative models might be used. For

instance, the **SIS** (Susceptible - Infectious - Susceptible) model is used for infections whose immunity are not long-lasting, such as the common cold and influenza; the **SEIR** (Susceptible - Exposed - Infectious - Recovered) model accounts for the latency period during which an infected person has not become infectious, etc. In the scope of this project, to compare a SIR-based method against other approaches, we have selected the **SIRV** (Susceptible - Infectious - Recovered - Vaccinated) model to consider vaccination among the susceptible. This model will be discussed in more details in Section 2 of this report.

1.4 Problem Statement

The COVID-19 pandemic has presented unprecedented challenges globally, necessitating accurate and timely predictions of infection rates to inform public health responses and policy decisions. Various prediction models have been developed to forecast the number of COVID-19 cases, utilizing diverse datasets that include daily reported cases, deaths, vaccination details, and changes in government policies. However, these models vary significantly in their methodologies, performance, and the ability to incorporate and interpret the influence of external factors such as policy changes and vaccination rates.

This project aims to conduct a comprehensive analysis of different COVID-19 infection prediction methods. The primary objectives are to evaluate the performance of various models in predicting the number of future cases and to provide empirical explanations of how external factors, such as government policies and real-life events, impact the predictions. By systematically comparing the prediction accuracy and interpretability of multiple models, this study seeks to identify the most effective approaches for forecasting COVID-19 infection trends and understanding the underlying drivers of these trends.

Through this comprehensive analysis, we aim to offer valuable insights into the strengths and limitations of existing prediction methods, ultimately contributing to the development of more robust and reliable tools for managing the ongoing and future public health crises.

2 Methodology

2.1 Data Collection and Exploration

2.1.1 Data Scraping

The data for this analysis was sourced from two websites:

1. **https://covid19.ncsc.gov.vn/dulieu**: To examine the impact of specific policies on disease outcomes, we focused on collecting daily case data from Hanoi city.

2. **https://www.google.com/covid19/mobility** We also gathered information on the policies implemented over a three-year period, from 2020 to 2022, to understand how these measures evolved in response to the COVID-19 pandemic. The reports documented movement trends over time by geography and across various categories of places such as retail and recreation, groceries and pharmacies, parks, transit stations, workplaces, and residential areas.

2.1.2 Exploratory Data Analysis (EDA)

Initially, we performed an Exploratory Data Analysis to gain preliminary insights into our dataset. Our dataset spans a period of three years, covering 1009 days from February 2, 2020, to November 19, 2022. Except for the 'Date' field, which is of 'Datetime' type, all features in our dataset are numerical.

image here

Due to the nature of web scraping from different sources, our dataset inevitably contained a significant amount of missing values. As it is challenging to impute future data, we opted to remove all records that had at least one NaN value. After this cleaning process, we were left with 581 records, ranging from March 18, 2020, to October 19, 2021.

image here

To further analyze our data, we utilized df.describe() to obtain statistical summaries and visualized the distributions. The analysis revealed that all policies had a similar range and distribution.

image of data statistics + visualization here

From the visualizations, it was evident that policy stringency tended to decrease as the number of daily cases increased. However, there was a lag, with the peak of the disease occurring slightly after the peak in policy stringency. This lag can be attributed to two main factors: the need for stricter policies during the peak of the disease and the likelihood that daily case numbers are influenced by policy measures. We tested this hypothesis and obtained the following results:

image of Duc's hypothesis here

Finally, to verify the stationarity of the data, we plotted the ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) charts to (evaluate the correlation between observations in a time series and identify the lagged relationships). Specifically:

- ACF: (The ACF helps in understanding the overall correlation structure of the time series by showing the correlation between the series and its lagged values at different lags).
- PACF: (The PACF is used to identify the extent of the correlation at each lag that is not accounted for by previous lags, helping to determine the order of the autoregressive model).

ACF and PACF here

2.2 Data Processing

After exploring our dataset, we need to preprocess it before feeding it to our model. This step is particularly crucial when working with time series data, as a clean dataset can significantly enhance model performance.

2.2.1 Data Partitioning

We will divide the data into two segments: a training segment and a testing segment. For the training segment, we will use 80% of the data for the training set and 20% for the validation set. We will then test the model's performance by predicting the disease spread for the next 14 days.

2.2.2 Feature Normalization

After oversampling the data, we observed that each feature now has different ranges. This inconsistency can hinder our training process by slowing down convergence and potentially causing overfitting. To ensure that each feature contributes equally to the decision boundary, we need to scale all features uniformly. We will normalize the data to a **standard distribution** with a **mean of 0** and a **standard deviation of 1.**

2.2.3 Feature Engineering

Feature Engineering: This involves selecting, manipulating, and transforming raw data into features suitable for supervised learning.

- **Date-Related Features:** Since our data is a time series, it is essential to include temporal information for each sample, especially for seasonal or cyclical data. We will extract the day, month, and year from the 'Date' column and create corresponding new columns.
- **Lag Features:** We will use the target column to create lagged versions of the data, with lags of 1, 2, and 3 days. This involves shifting the target column to create new columns that represent the previous days' values. Any resulting NaN values will be removed. (Lag features help capture temporal dependencies and trends in the data, providing the model with relevant past information.)

- **Expanding Rolling Window:** We will create a new column where the value at row N is the average of the target column values from the previous N-1 rows. (This method helps smooth the data and capture the overall trend, improving the model's ability to make predictions.)

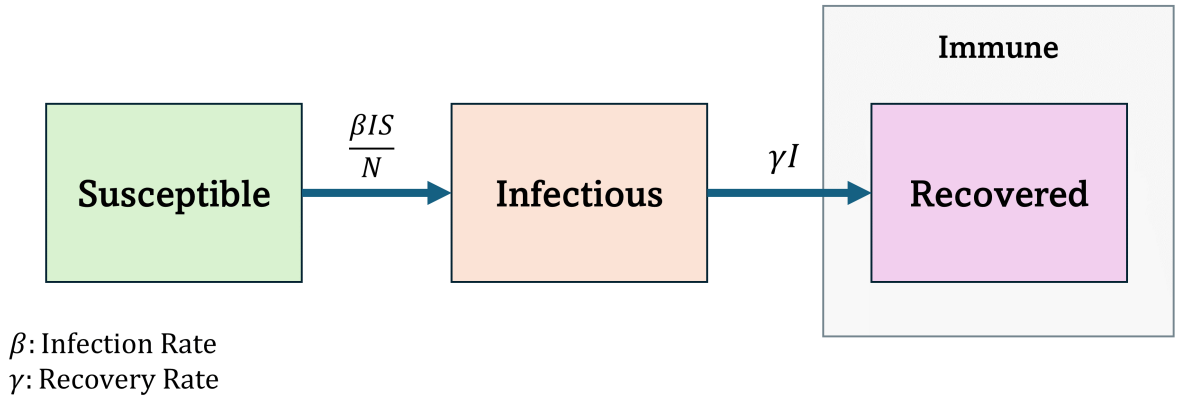
2.3 Models

2.3.1 Traditional Models

In this section, we will explore the mathematical basis of the **SIR** model and how it was adapted to **SIRV** to account for vaccination programs.
The mathematical notations used in this section are as follows.

- *S, I, R, V* or *S(t), I(t), R(t), V(t)* denote the number of susceptible, infected, removed (either by death or recovery), and vaccinated people at time t respectively.
- N denotes the total population, including those died of the disease in question. In this problem, it is assumed that N remains stable throughout.
- β, the *infection rate*, is the average number of contacts per person per time.
- γ, the *recovery rate*, represents how fast people recover from their infectious state.
- $R_t = \frac{\beta_t}{\gamma_t}$ is the *real-time reproduction number*. Its variants include the basic reproduction number $R_0 = \frac{\beta_0}{\gamma_0}$ and the real-time effective reproduction number $R_e = \frac{\beta_t S}{\gamma_t N}$.

a. The SIR Model



Hình 1: Overall structure of the SIR Model

Originally created in 1927 by W. O. Kermack and A. G. McKendrick, this model concerns a fixed population with only three compartments. The flow of the model is as follows:

$$S \to I \to R$$

The transition between these compartments are governed by the following equations:

$$\frac{dS}{dt} = -\beta SI$$

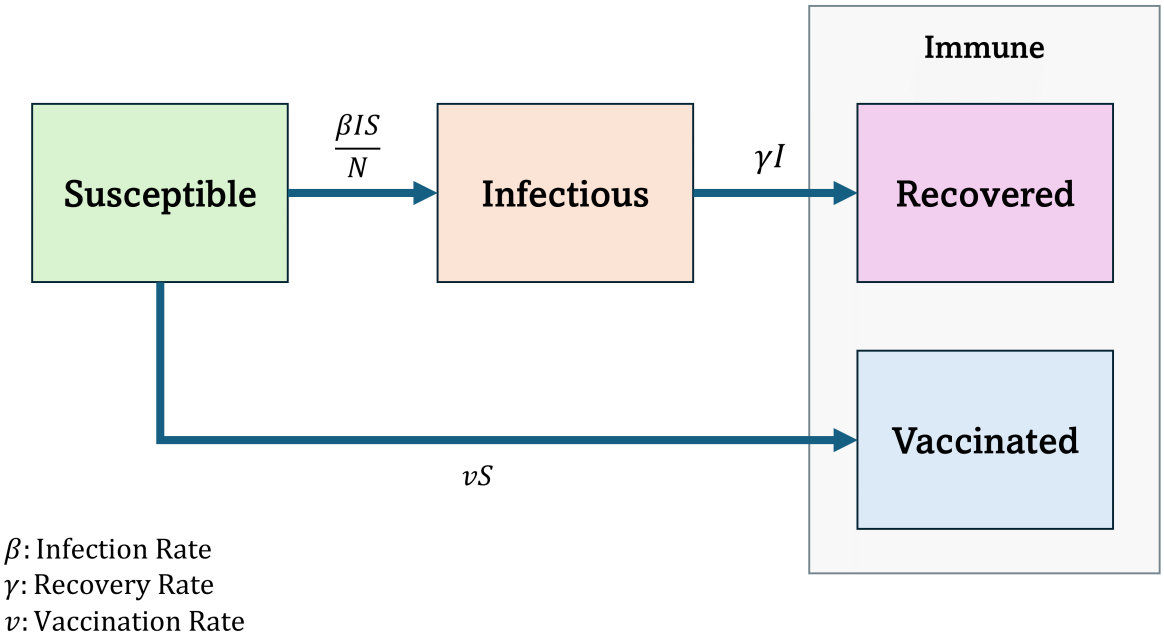
$$\frac{dI}{dt} = \beta SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

Here, βSI represents the number of people getting infected at each time step, and γI represents the number of people either dying or having recovered at each time step. Furthermore, under the assumption that the rates of infection and recovery are much higher than the time scale of birth and death, the population can be considered to be stationary, leading to $N = S(t) + I(t) + R(t)$ at all time point.

Under these conditions, it is possible to study the spread of a disease by fitting the parameters γ and β under specific conditions (i.e. with a certain preventive measure or a combination thereof).

b. The SIRV Model



Hình 2: Overall structure of the SIRV Model

The SIRV Model introduces an additional vaccination rate v. It assumes that vaccination is done on susceptible people and provides definite immunity for those in the vaccinated compartment.

The transition between compartments in the SIRV model is given by the following equations:

$$\frac{dS}{dt} = -\frac{\beta IS}{N} - vS$$

$$\frac{dI}{dt} = \frac{\beta IS}{N} - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

$$\frac{dV}{dt} = vS$$

As Vietnam has a high vaccination rate (

2.3.2 Deep Learning Models

a. RNN Model

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data, making them highly suitable for tasks involving time series analysis, natural language processing, and other applications where the order of inputs is significant.

Unlike traditional feedforward neural networks, RNNs have connections that loop back on themselves, allowing them to maintain a memory of previous inputs through their hidden states. This capability enables RNNs to capture temporal dependencies and patterns within the data.

- **Input Layer**: Processes the sequential data, feeding one element of the sequence into the network at each time step.
- **Hidden Layer**: Each hidden unit receives input from both the current element of the sequence and the hidden state from the previous time step. The recurrent connections allow the network to retain information over time, effectively creating a dynamic memory that captures temporal patterns.
- **Output Layer**: Produces an output for each time step based on the current hidden state. The nature of the output depends on the specific task, such as classification, regression, or sequence generation.

At each time step t, the RNN updates its hidden state h_t and generates an output y_t using the following equations:

$$h_t = f(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = g(W_y h_t + b_y)$$

Where:

- x_t is the input at time step t
- h_t is the hidden state at time step t
- h_{t-1} is the hidden state from the previous time step
- W_h and U_h are the weight matrices for the input and recurrent connections, respectively

This formulation captures the essence of how RNNs process sequential data, enabling the model to learn and remember temporal patterns by updating its hidden states at each step based on both the current input and the accumulated information from previous inputs.

b. LSTM Model

LSTMs, or Long Short-Term Memory networks, are a specific type of RNN architecture designed to address a major limitation of standard RNNs: the vanishing gradient problem.

LSTMs address this issue by introducing a concept called a memory cell. This cell, along with forget gates, input gates, and output gates, allows LSTMs to selectively remember and utilize information over long sequences.

- **Memory Cell**: Unlike the hidden state in a standard RNN, the LSTM's memory cell can store information for extended periods.
- **Gates**: LSTM networks use three gates to control the flow of information within the memory cell:
  - **Forget Gate**: Decides what information to forget from the cell's current state.
  - **Input Gate**: Determines what new information to store in the cell.
  - **Output Gate**: Controls what information from the cell's current state to output.

By regulating information flow through these gates, LSTMs can effectively learn and utilize long-term dependencies within sequences.

At each time step t, the LSTM updates its cell state C_t and hidden state h_t using the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

Where:

- x_t: Input at time step t
- h_t: Hidden state at time step t
- h_{t-1}: Previous hidden state
- C_t: Cell state at time step t
- C_{t-1}: Previous cell state
- W, U, b: Weight matrices and biases

LSTM gates (forget f_t, input i_t, and output o_t) control the flow of information, allowing the network to maintain and update the cell state over long sequences, capturing long-term dependencies. This architecture allows LSTMs to maintain and update a cell state over long sequences, effectively capturing long-term dependencies in the data.

c. GRU Model

Gated Recurrent Units (GRUs) are a type of RNN designed to address the vanishing gradient problem commonly encountered in traditional RNNs. GRUs simplify the architecture by combining the hidden state and the cell state into a single state and using two gating mechanisms to control the flow of information. This enables GRUs to capture temporal dependencies more effectively, making them suitable for tasks such as time series prediction and sequence modeling.

GRUs (Gated Recurrent Units) address the vanishing gradient problem through a simplified approach compared to LSTMs. They introduce two new gating mechanisms: **Reset Gate** and **Update Gate**. By regulating the flow of information through these gates, GRUs can effectively learn and utilize long-term dependencies within sequences.

A GRU unit consists of the following components:

- **Update Gate**: Determines how much of the previous hidden state needs to be passed along to the current hidden state.
- **Reset Gate**: Decides how much of the previous hidden state should be forgotten.
- **Candidate Hidden State**: Represents the new hidden state value, which is computed using the current input and the reset-modified previous hidden state.
- **Hidden State**: The final hidden state at time step t, which combines the previous hidden state and the candidate hidden state based on the update gate.

At each time step t, the GRU updates its hidden state h_t using the following equations:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Where:

- x_t: Input at time step t
- h_t: Hidden state at time step t
- h_{t-1}: Previous hidden state
- z_t: Update gate
- r_t: Reset gate
- $\tilde{h}_t$: Candidate hidden state

GRU gates (update z_t and reset r_t) control the flow of information, allowing the network to efficiently capture temporal dependencies.

d. Stacked LSTM

Stacked Long Short-Term Memory networks are an advanced variant of LSTM designed to capture more complex temporal patterns and dependencies by utilizing multiple layers of LSTM cells. By stacking LSTM layers, the network can learn hierarchical representations of the sequential data, making it suitable for more challenging tasks involving long-term dependencies and complex sequences.

The architecture of a Stacked LSTM consists of multiple layers of LSTM cells, where the output of one layer serves as the input to the next. This layered structure allows the model to capture a wider range of temporal dependencies and enhances its ability to model complex sequences. The primary components of a Stacked LSTM include:

- **Input Layer**: Receives the sequential data and feeds it into the first LSTM layer.
- **Stacked LSTM Layers**: Multiple layers of LSTM cells, each processing the sequence and passing its output to the next layer. Each LSTM layer maintains its own cell state and hidden state, which are updated at each time step based on the input and previous states.
- **Output Layer**: Generates the final prediction based on the output of the last LSTM layer.

At each time step t, the LSTM updates its cell state C_t and hidden state h_t using the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

Where:

- x_t: Input at time step t
- h_t: Hidden state at time step t
- h_{t-1}: Previous hidden state
- C_t: Cell state at time step t
- C_{t-1}: Previous cell state
- W, U, b: Weight matrices and biases

e. Convolutional LSTM

Convolutional LSTMs (ConvLSTMs) are a type of deep learning architecture specifically designed to tackle time series problems involving spatiotemporal data. This means the data unfolds not only over the temporal dimension but also across spatial dimensions, like images or grids. ConvLSTMs combine the strengths of Convolutional Neural Networks (CNNs) for spatial feature extraction and Long Short-Term Memory (LSTMs) networks for capturing long-term dependencies within sequences.

ConvLSTMs are widely preferred due to the following reasons:

- **Effective Feature Extraction**: ConvLSTMs leverage CNN layers to automatically extract relevant spatial features from each element in the sequence, making them ideal for various time series applications.
- **Long-term Dependency Learning**: The LSTM layers in a ConvLSTM excel at learning long-term dependencies within the extracted spatial features, crucial for understanding complex time series relationships.
- **Handling Complex Spatiotemporal Data**: ConvLSTMs are adept at handling complex data with both spatial and temporal components.

However, there are some cons that came across while working with ConvLSTMs: Firstly, the **increased complexity** of the model, as ConvLSTMs are more complex to design and train compared to standard LSTMs due to the combined architecture. Secondly, these models **require larger datasets** to train, which is a difficult challenge if the data is scarce. And finally, developers may face problems in the **interpretability** of ConvLSTMs, since the combined layers and internal processing within the network are sophisticated to understand.

A typical ConvLSTM architecture for time series problems consists of the following components:

- **Input Layer**: Receives the spatiotemporal data, like a sequence of video frames or sensor readings from a grid.
- **Convolutional Layers**: These layers extract spatial features from each element in the sequence.
- **ConvLSTM Layers**: Here, the core functionality lies. These layers combine CNN and LSTM capabilities. They process the sequence of spatially-aware representations generated by the CNN layers, capturing long-term dependencies within the spatial features across time steps.
- **Output Layer**: The final layer interprets the output from the ConvLSTM layers to produce the desired predictions.

At each time step t, the ConvLSTM updates its cell state C_t and hidden state H_t using the following equations:

$$f_t = \sigma(W_f * X_t + U_f * H_{t-1} + b_f)$$

$$i_t = \sigma(W_i * X_t + U_i * H_{t-1} + b_i)$$

$$\tilde{C}_t = \tanh(W_C * X_t + U_C * H_{t-1} + b_C)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$o_t = \sigma(W_o * X_t + U_o * H_{t-1} + b_o)$$

$$H_t = o_t \odot \tanh(C_t)$$

Where:

- X_t: Input at time step t
- H_t: Hidden state at time step t
- H_{t-1}: Previous hidden state
- C_t: Cell state at time step t
- C_{t-1}: Previous cell state

- W, U, b: Convolutional kernels and biases
- σ: Sigmoid activation function
- tanh: Hyperbolic tangent function

2.3.3 ARIMA model

The ARIMA (AutoRegressive Integrated Moving Average) model is a popular and widely used statistical method for time series forecasting. It combines three components: autoregression (AR), differencing (I for integration), and moving average (MA). Here's an in-depth look at each component and the overall model. Components of ARIMA comprise:

- **The autoregressive component** involves regressing the time series on its own previous values. This means the model uses the relationship between an observation and a number of lagged observations (i.e., previous time points) to predict future values. Mathematically, an autoregressive model of order p (AR(p)) can be written as:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t$$

  where Y_t is the value at time t, ϕ_i are the parameters of the model, and ε_t is white noise.

- **Integrated (I) component:** The integrated component involves differencing the time series to make it stationary, which means that statistical properties like mean and variance are constant over time. This step is crucial because many time series are non-stationary. The order of differencing is denoted by d, and it refers to the number of times the data values have had past values subtracted. For example, if $d = 1$, the differencing is done as:

$$Y_t' = Y_t - Y_{t-1}$$

- **Moving Average (MA) Component:** The moving average component models the relationship between an observation and a residual error from a moving average model applied to lagged observations. Mathematically, a moving average model of order q (MA(q)) can be written as:

$$Y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

  where θ_i are the parameters of the model, and ε_t is white noise.

The ARIMA model combines these three components into a single framework. The notation ARIMA(p, d, q) indicates the order of the autoregressive, integrated, and moving average parts of the model, respectively.

The general form of the ARIMA model is:

$$\Delta^d Y_t = \phi_1 \Delta^d Y_{t-1} + \phi_2 \Delta^d Y_{t-2} + \cdots + \phi_p \Delta^d Y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

where Δ^d denotes differencing d times.

Steps to Build an ARIMA Model

- **Identification:** Determine the values of p, d, and q by analyzing autocorrelation (ACF) and partial autocorrelation (PACF) plots, and using criteria like AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion).
- **Estimation:** Estimate the parameters ϕ and θ using methods like maximum likelihood estimation.
- **Diagnostic Checking:** Check the residuals of the model to ensure they resemble white noise (i.e., no patterns or autocorrelations remaining).
- **Forecasting:** Use the fitted ARIMA model to make forecasts.

How to Select Optimal Parameters in ARIMA

Selecting the optimal parameters in an ARIMA (p, d, q) model is crucial to ensure accurate and effective forecasting. Here are the steps and methods commonly used to choose the optimal parameters:

1. Determine the value of d (Differencing)

First, we use Augmented Dickey-Fuller (ADF) Test need to ensure that your time series is stationary. You can check the stationarity of the time series using the following methods:

- **Visual Inspection**: Observe the time series plot to see if it has a trend.
- **Augmented Dickey-Fuller (ADF) Test**: Use the ADF test to check for stationarity. If the series is not stationary, you can apply differencing until the series becomes stationary.

2. Determine the values of p and q

Once the time series is stationary, you can determine the values of p and q by using ACF and PACF plots:

- **ACF (Autocorrelation Function)**: The ACF plot helps identify the lag for the MA part. The lag at which the ACF cuts the x-axis is the suggested value for q.

- **PACF (Partial Autocorrelation Function)**: The PACF plot helps identify the lag for the AR part. The lag at which the PACF cuts the x-axis is the suggested value for p.

3. Use Model Selection Criteria

After obtaining initial values for p, d, q, you can fine-tune them using model selection criteria such as:

- **Akaike Information Criterion (AIC)**: AIC balances model accuracy and complexity. The model with the lowest AIC value is considered optimal.
- **Bayesian Information Criterion (BIC)**: Similar to AIC, but BIC penalizes complex models more strongly. The model with the lowest BIC value is considered optimal.

3 Results

3.1 Evaluation Metrics

For evaluating the performance of various models used in our project, we will require the help of three indicators:

- **MAE (Mean Absolute Error)**: This metric represents the average magnitude of the difference between the predicted values in the model and the actual values in the data.

$$\textbf{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **RMSE (Root Mean Square Error)**: This metric is essentially the square root of the MAE, therefore scaling back the error to the same metrics as our data.

$$\textbf{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **R-squared**: R-squared is used to represent the proportion of variance in the dependent variable that can be explained by the independent variables inside the model. In simpler terms, it describes how well the model can fit the data.

$$R^2 = \frac{Var(y) - Var(e)}{Var(y)} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Model	MAE	RMSE	R square score	Adjusted R square score
RNN	0.67	0.85	-0.10	1.73
GRU	0.63	0.82	-0.03	1.69
LSTM	0.67	0.84	-0.11	1.74
Convolutional LSTM	1.05	1.17	-0.21	1.41
Stacked LSTM	0.53	0.79	0.51	1.21
ARIMA model	0.23	8e-5	0.20	6.18

Bång 1: Comparison of Model Performance Metrics

3.2 Hyperparameter tuning

In this study, we consistently used the Adam optimizer with a learning rate of 0.001 and a batch size of 100. We then experimented with different sets of hyperparameters and found the final results as follows:

- **RNN**: We set the number of hidden units to 300 and used the 'relu' activation function with an output unit size of 10.

- **GRU**: We set the number of hidden units to 300 and used the 'relu' activation function with an output unit size of 8.

- **LSTM**: We set the number of hidden units to 200 and used the 'relu' activation function with an output unit size of 8.
- **Convolutional LSTM**: We used 2 convolutional layers and 4 LSTM layers with hidden units of 64, 32, 16, and 8 respectively.
- **Stacked LSTM**: We used 4 LSTM layers with hidden units of 80, 60, 40, and 20 respectively.

The results were quite surprising: although LSTM is capable of capturing long-term dependencies, its performance was worse than RNN (this can be explained by the possibility that the specific dataset might not have required complex temporal dependencies, thus leading to overfitting or underfitting in the LSTM model) and comparable to GRU. The Convolutional LSTM, despite having a complex architecture and achieving better results, showed a negative R square score (indicating that the model might be overfitting to the training data and failing to generalize well to the validation set). The best results were obtained with the Stacked LSTM (because the multiple LSTM layers can capture a wider range of temporal patterns and dependencies, leading to better performance and generalization). Please note that due to the poor performance of the SIR model, we have not included its results in this report. We concluded that the SIR model should only be used to predict the general trend of the epidemic and to assess whether there might be hospital overloads in the future. As for the ARIMA model, we limited our predictions to a short-term period (7 days), yet it yielded quite satisfactory results compared to other models (provide additional explanation).

4 Conclusion

In this project, we have undertaken a comprehensive analysis of various methods for predicting COVID-19 infection rates. Our investigation covered traditional statistical models, deep learning architectures, and also provided insights on ARIMA. Our objective was to evaluate the performance of these models in predicting infection rates and understand the impact of various features, such as government policies and real-life events, on the prediction outcomes.

Throughout the project, we have drawn some key findings as follows:

- In terms of **model performance**, the **SIR** model provided a foundational understanding of the epidemic dynamics, capturing the basic infection trends but struggling with complex temporal patterns. Turning to **ARIMA**, they performed well in capturing linear temporal dependencies and were particularly effective for short-term forecasts. However, they had limitations in handling non-linearities and long-term dependencies. **Deep learning models**, on the other hand, demonstrated superior performance in capturing intricate temporal and spatiotemporal dependencies, offering more accurate and robust predictions.
- The **additional features**, such as the number of deaths, vaccination details, and changes in government policies also showed great impact, as they significantly enhanced the prediction accuracy across all models. These features provided critical context, allowing models to better understand and predict infection trends influenced by external factors.

To sum up, the analysis highlighted the significant impact of government policies and real-life events on the spread of the pandemic. The policies in Hanoi evolved drastically, from initial responses and containment of early infections to adaptive strategies, vaccine rollouts, and the "new normal" life in the province. The data generally reflects these policies, with increases in cases when restrictions were relaxed and gradual decreases due to vaccination efforts.

However, several issues need to be addressed for further data analysis. There are inconsistencies and significant variance in the numbers within specific periods when policies remained consistent, as well as a lack of immediate response in the data when new measures were implemented. More research is needed based on data insights, and improving the quality of data collection is essential to better clean the data, reduce anomalies, and support future research efforts.

References

[1] Cutler, D. M., & Summers, L. H. (2020). The COVID-19 Pandemic and the $16 Trillion Virus. JAMA, 324(15), 1495–1496. https://doi.org/10.1001/jama.2020.19759

[2] Naseer, S., Khalid, S., Parveen, S., Abbass, K., Song, H.,& Achim, M. V. (2023). COVID-19 outbreak: Impact on global economy. Frontiers in public health, 10, 1009393. https://doi.org/10.3389/fpubh.2022.1009393