Master

Master's in Actuarial Science

Master's Final Work

Dissertation

MODELLING HOSPITAL ADMISSION RATES IN SÃO PAULO, BRAZIL

LEE-CARTER MODEL *VS.* NEURAL NETWORKS

Rodolfo Monfilier Peres

October – 2022

Master

Master's in Actuarial Science

Master's Final Work

Dissertation

MODELLING HOSPITAL ADMISSION RATES IN SÃO PAULO, BRAZIL

LEE-CARTER MODEL *VS.* NEURAL NETWORKS

Rodolfo Monfilier Peres

Supervisor

Onofre Alves Simões

October - 2022

Abstract

In Brazil, hospital admissions represent almost 50% of the total claims cost of health insurance companies while they only represent 1% of the total medical procedures. Therefore, modeling hospital admissions is extremely useful for health insurers to assess their claim costs over time and actuaries should be capable to include that information in their analyses, in order to preserve the financial sustainability of the companies.

This dissertation analyses the use of the Lee-Carter model for predicting the general level of hospital admissions in the state of São Paulo, Brazil, using the traditional ARIMA model and contrasting it with the LSTM neural network. Publicly available data between the years 2008 and 2019, divided by gender, were used. The function *auto.arima* from the R package *forecast* was used to find the best ARIMA model for the data while the LSTM neural network model was searched in a combination of 20 models, varying the learning rate and decay factor. The results showed that the LSTM model and the ARIMA have similar RMSE and MAE performance.

Keywords: Lee-Carter; Neural Network; LSTM; Hospitalizations; Time Series

Resumo

No Brasil, hospitalizações representam quase 50% dos custos totais de sinistros em operadoras de planos de saúde enquanto representam apenas 1% dos procedimentos médicos. Portanto, estimar hospitalizações é extremamente útil para que operadoras de planos de saúde possam estimar seus custos ao longo do tempo e atuários devem ser capazes de incluir essas informações em suas análises para garantir a sustentabilidade financeira das companhias.

Essa dissertação analisa o uso do modelo de Lee-Carter para prever o nível geral de hospitalizações no estado de São Paulo, Brasil, utilizando o modelo ARIMA tradicional e comparando-o com a rede neuronal LSTM. Dados públicos entre os anos de 2008 e 2019, divididos por sexo, foram utilizados. A função *auto.arima* do pacote R *forecast* foi utilizada para encontrar o melhor modelo ARIMA enquanto que a rede neuronal LSTM foi selecionada entre a combinação de 20 modelos, variando a *learning rate* e o *decay factor*. Os resultados mostraram que o modelo LSTM e o modelo ARIMA possuem RMSE e MAE similares.

Palavras-chave: Lee-Carter Model; Redes Neuronais; LSTM; Internações; Séries Temporais

Contents

List of Figures

List of Tables

Chapter 1 - Introduction

1.1 - Overview and motivation

The main purpose of this dissertation is to add a contribution to the set of possible applications of neural network techniques in the actuarial science field. More specifically, the aim is to model the general level of hospital admission rates using recent neural network developments and to contrast results with the results supplied by the Lee-Carter model (Lee & Carter, 1992). By doing so, we intend to come to an understanding about neural network models being useful to model hospital admissions and, if they are, whether they show considerable improvement over the Lee-Carter model.

The Lee-Carter model is vastly used for mortality modeling, with few applications outside this area (Frees, 2006). The first application for health insurance was proposed by (Lee & Miller, 2002) to forecast Medicare expenditures in the period 2020-2075, in the United States. Although mostly used in the mortality field, it was shown by (Rodrigues, Andrade, Queiroz & Machado, 2013) that it is also suitable to forecast admission rates in hospitals. According to the authors, its major advantage over the traditional forecast methods in health insurance is the fact that it is possible to construct confidence intervals in this approach.

More recently, researchers have made numerous developments in the use of neural networks for mortality modeling, which have shown great improvement over the Lee-Carter model, see (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019; Hainaut, 2018; Deprez, Shevchenko & Wüthrich, 2017; Richman & Wüthrich, 2018).

Considering the elements described above, this dissertation intends to model the general level of hospital admissions rates in São Paulo, Brazil, as done by (Rodrigues, Andrade, Queiroz & Machado, 2013), using publicly available data. The idea is to model the admission rates again, in light of the recent studies that combine the Lee-Carter model with neural

networks and to contrast the models. The purpose is to determine if these new models with neural networks can provide improvements over what was done before.

In Brazil, hospital admissions represent almost 50% of the total claim cost of health insurance companies while they only represent 1% of the total medical procedures (Cechin & Lara, 2020). Given that, modeling hospital admissions is extremely useful for health insurers to assess their claim costs over time. Adverse events can seriously strain their liabilities, and actuaries should be well prepared to assess that to keep the financial sustainability of the companies.

This dissertation aims to contribute to the development of machine learning techniques in the actuarial field while also adding a contribution to forecasting models in healthcare. As will be seen in the literature review section, the use of neural networks combined with the Lee-Carter model is a new subject in the actuarial field. The first papers on this topic only appeared in 2018, see (Hainaut, 2018). Even though some contributions have been made since then, the lack of literature on the subject imposes difficulties for new studies but, at the same time, it is motivating.

1.2 - Literature Review

This section starts by briefly reviewing the Lee-Carter model. Then it presents a short survey of healthcare forecasting. Finally, it reviews the combined use of neural networks and the Lee-Carter model.

1.2.1 - Some Applications of the Lee-Carter Model

In 1992, (Lee & Carter, 1992) proposed a new model for estimating mortality rates. It became widely spread and the leading statistical model for mortality forecasting (Deaton & Paxson, 2004) and it started to be used as a benchmark model for population mortality (Hollmann, Mulder & Kallan, 2000). Still today, several developed countries such as Denmark, Sweden,

Canada, and Italy use the Lee-Carter model to forecast mortality (Kjærgaard & Bergeron-Boucher, 2022).

Lee and Carter seek to summarize an age-period surface of log-mortality rates in terms of an average age profile of mortality, mortality changes over time, and how much each age group changes when mortality changes.

Despite the simplicity of the model, it resulted in good outcomes in fitting mortality rates for several countries (Steeghs, 2020). It was used by (Wilmoth, 1996) in Japan, by (Tuljapurkar, Li & Boe, 2000) for estimating mortality in the G7 countries, and by several other authors that applied the Lee-Carter for many different populations, see for instance (Kjærgaard & Bergeron-Boucher, 2022) who forecasted mortality for age 65 or above for four European countries and (Rabbi & Mazzuco, 2020) who applied the model with smoothed mortality rates for 20 low-mortality countries.

However, its popularity and simplicity did not prevent the model from being criticized, mostly because it assumes that all information about future mortality is contained in the past observed data, not including important covariates such as tobacco use, alcohol consumption or comorbidity (Girosi & King, 2008). Moreover, exogenous shocks such as new medical technologies, economic crises, pandemics, etc. are ignored (Gutterman & Vanderhoof, 1998).

1.2.2 - Healthcare forecasting

In recent years, most healthcare systems are going through reforms, attempting to control the raising costs of healthcare. Generally, most reforms focus on strengthening primary care, adopting mechanisms for supply-induced demand, new forms of care (i.e. home care and long-

term care), and promoting changes to achieve a better lifestyle (Paris, Devaux & Wei, 2010; Menec, Lix, Nowick & Ekuma, 2007; Rodrigues, Andrade, Queiroz & Machado, 2013).

With these reforms in mind, several studies have focused on forecasting health service expenditures and frequency of utilization. Traditional methods of forecasting healthcare expenditures use a fixed utilization rate by age to estimate the pure demographic effect on health costs (Tate, MacWilliam & Finlayson, 2005; Lindberg & McCarthy, 2021). The pure demographic effect assumes that healthcare costs stay the same and are impacted only by the size and age structure of the population (Lindberg & McCarthy, 2021). Because of this assumption, traditional methods assume that costs are only impacted by the demographic changes of the population. The main caveat is that they can only be used in short-period analyses since changes in utilization patterns are not incorporated (Lindberg & McCarthy, 2021).

Other (non-traditional) methods have tried to forecast healthcare expenditure based on time series analyses (Tate, MacWilliam & Finlayson, 2005). Other studies have relied on the use of panel data to estimate healthcare utilization. These studies include other covariates such as income per capita and educational level (Xu, Saksena & Holly, 2011; European Commission, 2013).

In the first study using the Lee-Carter model to forecast health expenditures (Lee & Miller, 2002), the authors applied the model by setting a fixed age pattern of expenditures. Following (Lee & Miller, 2002), (Rodrigues, Andrade, Queiroz & Machado, 2013) were successful in predicting healthcare admission rates by applying the Lee-Carter model in Brazil.

1.2.3 - Neural networks & Lee-Carter

Research contributions, such as the ones mentioned below, that combine the use of the Lee-Carter model and neural networks in the demographic field of study have been growing recently. In the work of (Deprez, Shevchenko & Wüthrich, 2017), the authors used machine

learning algorithms to assess the goodness of fit of standard mortality models. They analyze how a standard mortality model could be improved based on feature components of an individual, such as age. This work was further extended by (Levantesi & Pizzorusso, 2019), who used machine learning algorithms to calibrate a parameter that was applied to mortality rates fitted by standard mortality models.

Although both papers applied machine learning techniques in the field of mortality modeling, none of them have specifically used neural networks. The use of neural networks to predict mortality rates started with (Hainaut, 2018), by proposing a neural network that detects the non-linearities in the structure of the log forces of mortality. In the same year, (Richman & Wüthrich, 2018) proposed a Lee-Carter approach for multiple populations, where the parameters were estimated by neural networks. In the work of (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019), the authors use the Long Short-Term Memory (LSTM) neural network, which will be detailed in the next chapter, to improve the accuracy of predictions of the general level of mortality given by the Lee-Carter model.

In the paper of (Nigri, Levantesi & Marino, 2021), the authors consider an LSTM model to predict mortality and lifespan in five developed countries. Comparing the results with standard models, they conclude that their predictions provide a more accurate portrait. As stated by the authors, an LSTM model was chosen because: "*This type of neural network leads to predicting future values of longevity indexes while maintaining the significant influence of the past trend, but at the same time adequately reproducing the recent trend into forecasting.*" (Nigri, Levantesi & Marino, 2021, p. 1).

In another recent study, (Perla, Richman, Scognamiglio & Wüthrich, 2021) tested several neural networks to simultaneously predict mortality in all countries of the Human Mortality Database from 1950 onwards, showing that great accuracy can be achieved in a large-scale prediction.

1.3 - Organization

This dissertation is divided into five chapters. Chapter 1 presents the organization of this work, an overview, and motivation for the study, and the literature review of the subject. Chapter 2 defines and introduces the approaches used in the research, namely the Lee-Carter model and the neural network framework. Chapter 3 discusses the data used. In Chapter 4, the models are built and compared. Finally, Chapter 5 presents the conclusions and discusses further recommendations for future works on the topic.

Chapter 2 - Concepts and Models

2.1 - The Lee-Carter model

The Lee-Carter model (Lee & Carter, 1992) was developed for mortality forecasting and states that:

$$\log(u_{x,t}) = a_x + b_x k_t + e_{x,t}, \qquad (1)$$

where $u_{x,t}$ is the death rate for age *x* in year *t*, a_x is the average log of mortality at age x, b_x is the rate of change of the log mortality with time at age x, k_t is the general level of mortality for calendar year t, and $e_{x,t}$ is the residual term at age x and time t, with mean 0 and variance σ_e^2.

It is a two-fold model. First, parameters a_x, b_x and k_t need to be estimated. In the second stage, the fitted k_t values are modeled as an ARIMA(*p, q, d*) process. As explained by (Lee & Carter, 1992), the model cannot be fitted by ordinary regression methods because there are no given regressors; on the right side of the equation, there are only parameters to be estimated and the unknown index k_t.

To solve this problem, the authors applied the singular value decomposition (SVD) to the matrix of log mortality rates, normalized by subtracting the average log mortality at each age from each row of the matrix. The model is not identifiable, which means that parameters are not uniquely estimable. To avoid this problem, (Lee & Carter, 1992) imposed location and scale constraints on b and k, as follows:

$$\sum b_x^2 = 1, \qquad (2)$$

$$\sum k_t = 0. \qquad (3)$$

Besides SVD, there are other ways to estimate the parameters, namely the Weighted Least Squares (WLS) and the Maximum Likelihood Estimation (MLE), as done by (Wilmoth,

1993). Even though other methods to estimate the parameters could be applied, in this dissertation it was chosen to stick with the original SVD approach, see (Lee & Carter, 1992). The performance of the three methods was compared by (Koissi, Shapiro & Högnäs, 2005) and the authors concluded that the values estimated for a_x and b_x are almost identical regardless of the method used, supporting the decision to use the original SVD in this work.

In this dissertation, the Lee-Carter model is used to model the general level of hospital admission rates. In this new context, the variables of the model assume a new interpretation. By referring back to equation (1), in this new context the variable $u_{x,t}$ is now the hospitalization rate for age x in year t, a_x is the average log of hospitalization at age x, b_x is the rate of change of the log hospitalization with time at age x, k_t is the general level of hospitalization for calendar year t and $e_{x,t}$ is the residual term at age x and time t, with mean 0 and variance σ_e^2.

2.2 - Neural Network

2.2.1 - Overview

This section intends to give a brief explanation of a general neural network and present the Recurrent Neural Networks (RNNs) and the Long Short-Term Memory (LSTM) neural networks. Being the latter the one used in this work.

Neural networks are mathematical models based on the biological neural network structures of the brain (Minsky & Seymour, 2017; McCulloch & Pitts, 1943; Wiener, 1948). Similar to the brain, neural networks are composed of neurons and synaptic connections linking them. The general neural network model is formed by an input layer, one or several hidden layers, and an output layer, and each of them is composed of several neurons (Nigri, Levantesi,

Marino, Scognamiglio & Perla, 2019). Figure 1 below shows the structure of a classical neural network.

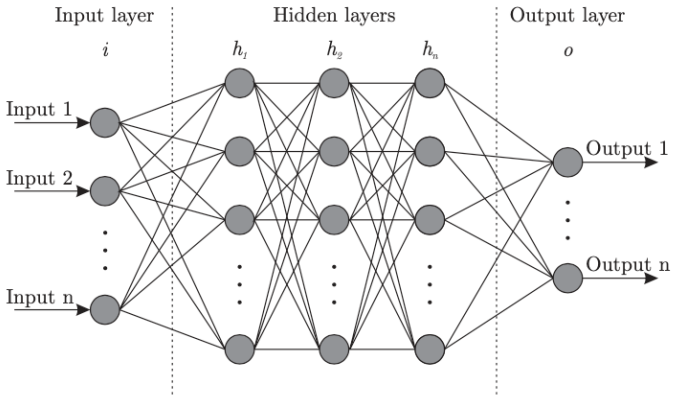


Figure 1 - General structure of a neural network

Source: (Bre, Gimenez & Fachinotti, 2017)

This structure is called feedforward neural network since the information passes through it in only one direction. Each circle represents a neuron while each line represents the synapse connections between neurons. Each neuron is connected to those neurons in the neighboring layers via adaptative weights (Bre, Gimenez & Fachinotti, 2017). Synapses take the output of a given neuron and multiply it by a given weight. Neurons add the outputs from all synapses and apply an activation function (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019).

2.2.2 - The artificial neuron

In this section, we give a brief explanation of the artificial neuron, to further understand the neural network model. A neural network can be seen as a series of artificial neurons arranged together (Bre, Gimenez & Fachinotti, 2017). An artificial neuron is an information-processing unit that receives inputs and, by the weighted sum of these inputs, returns an output (Haykin, 2008).

Figure 2 shows what happens inside a generic neuron j in a given hidden layer k. The values a_{nj}^{k-1} represent the input from i^{th} neuron of the previous layer $k-1$, w_{nj}^{k} is the weight connecting to the previous i^{th} neuron output, b_j^k is the bias applied to neuron j in layer k, z_j^k is the output of the weighted sum in neuron j at layer k, $g(z_j^k)$ is any activation function applied to z_j^k and a_j^k is the output of the neuron j in layer k.

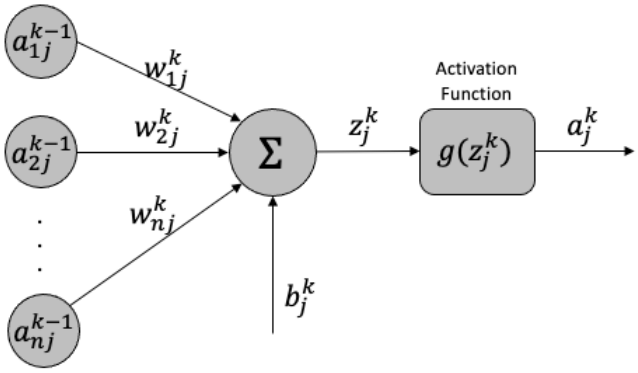


Figure 2 - The artificial neuron

Source: Adapted from (Bre, Gimenez & Fachinotti, 2017)

In analytical terms, Equation (4) and Equation (5) translate the process explained above.

$$z_j^k = \sum_{i=1}^{n} (w_{ij}^k \cdot a_{ij}^{k-1} + b_j^k), \qquad (4)$$

where w_{ij}^k is the weight of the connection between the i^{th} neuron of the previous layer and neuron j of layer k, a_{ij}^{k-1} is the output from the i^{th} neuron from the previous layer $k-1$ being applied to neuron j and b_j^k is the bias applied to neuron j in layer k.

The output z_j^k then passes through the activation function as defined below:

$$a_j^k = g(z_j^k), \quad (5)$$

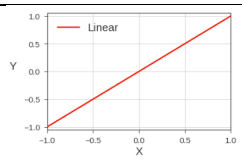
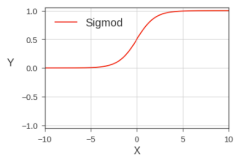
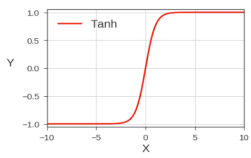
where a_j^k is the output of the neuron j in layer k, $g(\cdot)$ is any activation function and z_j^k is the weighted input to the activation function.

The weighted sum then passes through an activation function, a process that will be explained in the next section. There is also an externally applied bias b_j^k. It has the effect of increasing or decreasing the net input of the activation function, depending on if it is positive or negative (Haykin, 2008).

2.2.3 - The activation function

In the previous section, it was stated that the output of the artificial neuron passes through an activation function to limit the amplitude of the neuron's output. Without an activation function, the output of the neuron would be only a linear operation. It would consist of a dot product between the weights and the input of the neuron. This result is then added to the bias. To add non-linearity to the neural network model, an activation function is needed (Chollet, 2018). The reason why it is necessary to add non-linearity to the model is that, otherwise, it would only be able to learn linear transformations from the input data. So, for the model to be able to learn non-linear representations of the data, the activation function must be included (Chollet, 2018).

There are several activation functions in the neural network literature. Some examples of the ones commonly used are shown in Table 1:

Name	Definition	Plot
Linear	$f(x_i) = x_i$	
Logistic Sigmoid	$f(x_i) = \sigma(x_i)$	
Hyperbolic Tangent	$f(x_i) = tanh(x_i)$	

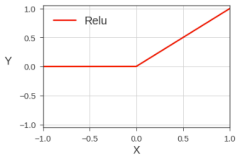
Rectified Linear Unit (ReLU)	$f(x_i) = \max\{0, x_i\}$	

Table 1 - Commonly used activation functions

Source: Adapted from (Neves, 2018)

2.2.4 - Cost function

In a neural network, to calculate the errors between the predicted values and the actual values, an error function needs to be defined. It is generally called cost function in the neural network literature for instance in (Krogh, Hertz & Thorbergsson, 1990). Several cost functions can be used, depending on the problem to solve.

Consider, for example, the cost function J, defined as the Mean Squared Error (MSE)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2, \qquad (6)$$

where θ represents the set of weights w and biases b of the neural network, m is the total number of training examples, $h_\theta(x^{(i)})$ is the prediction made for i^{th} training example, using the set of parameters θ, $x^{(i)}$ is the i^{th} training example, $y^{(i)}$ is the true value for the i^{th} training example.

The cost function needs to fulfill two conditions (Nielsen, 2015):

1. It must be written as a function of the outputs of the neural network. This always needs to hold because we want to calculate the difference between the output of the neural network and the actual values.
2. It must be written as an average, for m individual training examples. It means that the cost function is calculated as an average of m individual differences between the output and the actual value.

Since the cost function is a function of the differences between the predicted and the actual values, it needs to be minimized. The weights and biases in the network need to be adjusted in such a way that the value of the cost function is as small as possible.

2.2.5 - Gradient Descent

The idea of defining a cost function to evaluate the predictions made by the neural network was introduced in the previous section. This function needs to be minimized so that the errors are minimized too. Frequently, optimization problems are solved by using techniques of differential calculus to find the minimum or maximum of an appropriate function. In the case of neural networks, such an approach cannot be used since the network can take several weights and biases and the use of traditional differential calculus would be not effective in finding the minimum or the maximum of a function (Nielsen, 2015).

To overcome this problem, gradient descent is used. It was first proposed by (Cauchy, 1847), motivated by astronomical calculations (Lemarechal, 2012). It was also presented by (Hadamard, 1908) and further developed by (Curry, 1944) in the research of minimizing non-linear problems.

Gradient descent is an algorithm that iteratively minimizes a cost function by moving in the direction of the negative gradient of that function. The algorithm relies on the following equality:

$$\theta := \theta - \eta \cdot \nabla_{\theta} J(\theta), \qquad (7)$$

where θ represents the set of weights w and biases b of the neural network, η is the learning rate (explained next), ∇_{θ} is the gradient and $J(\theta)$ is the cost function.

2.2.6 - Learning Rate

Learning rate is a parameter that determines the step size at each iteration while moving towards a minimum of a cost function (Murphy, 2012, p. 247). The choice of the learning rate

η is important since a small value will lead to many iterations, making the algorithm slow, while a large value could not allow convergence to the minimum of the function (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019). Figure 3 illustrates how the learning rate works.

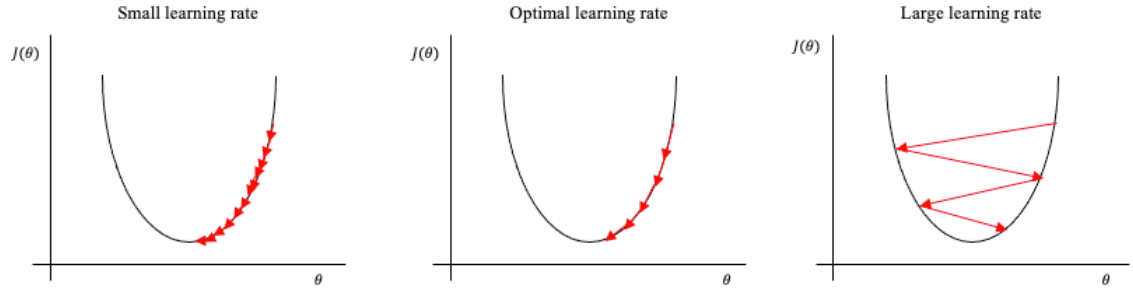


Figure 3 - Comparison between learning rates

Source: Author

When the learning rate is too small, the algorithm needs several iterations to adjust the values of the weights and find the minimum of the cost function. It is time-consuming and can be inefficient in real-life applications. If the learning rate is too high, the algorithm will rebound and never reach the minimum. The optimal learning rate should be one that allows the algorithm to converge within a desired time. There is no right or wrong value for the learning rate and one should try several values to find the one that best fits a given problem (Bengio, 2012).

2.2.7 - Backpropagation algorithm

In a neural network, weights and biases are randomly assigned and they need to be adjusted by several iterations until their optimal values are found. There are several ways in which they can be randomly initialized but, in general, random values are taken from a probability distribution such as Normal or Uniform, see (Narkhede, Bartakke & Sutaone, 2022) for an in-depth review on the topic.

The weights and biases are adjusted by backpropagating the errors through the neural network, in order to minimize the errors (Rumelhart, Hinton & Williams, 1986). The backpropagation algorithm is used to iteratively update the weights and biases of the network. It uses gradient descent to calculate the gradient of the cost function. The negative of the gradient points in the direction that minimizes the cost function.

Ultimately, the backpropagation algorithm iteratively computes the partial derivatives of the cost function in relation to the weights and biases of the network. Then, it updates the values of the weights and biases in the negative direction of the gradient. It computes again the cost function and evaluates if it has reached a given desired value. Figure 4 below summarizes how backpropagation pseudocode works.

Algorithm 1: Backpropagation Pseudocode

Input: Learning rate η; Randomly assigned θ

```
Compute cost function J(θ)
while J(θ) < Stop Criterion do
    Compute gradient ∇_θ J(θ);
    Update variable θ := θ − η∇_θ J(θ);
    Compute J(θ)
end
return θ
```

Figure 4 - Backpropagation Pseudocode

Source: Author

The algorithm receives a given learning rate η and a randomly assigned set of weights and biases θ, as input. It computes the cost function $J(\theta)$, the gradient ∇_θ and updates θ while the cost function is not smaller than a stop criterion. This criterion normally relies on human decisions and can be set in many different ways, such as defining a given threshold or stopping the algorithm after a given number of iterations (Lalis, Gerardo & Byun, 2014).

2.2.7.1 – Partial derivatives in relation to the weights

In this section, we explain how the partial derivative of the cost function in relation to the weights in the network is computed. First, let us define the cost function J. The derivative of the cost function in relation to the weight is $\frac{\partial J}{\partial w_{ij}^k}$. This can be computed by the chain rule as:

$$\frac{\partial J}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial z_j^k}\frac{\partial z_j^k}{\partial w_{ij}^k}, \quad (8)$$

The first factor on the right-hand side of equation (8) computes the derivative of the cost function J in relation to the output of neuron j, in layer k. This means that we are calculating how the input of the neuron influences the cost function. If the neuron is in the output layer it is straightforward to compute it. In this case, the neuron's output a_j^k is simply the prediction $\hat{y}(x)$ made by the network and we only need to calculate how the prediction influences the cost function. In this case, the derivative is:

$$\frac{\partial J}{\partial a_j^k} = \frac{\partial J}{\partial \hat{y}(x)}, \quad (9)$$

When neuron j is in any arbitrary hidden layer k of the network it is less obvious how to calculate the derivative. In this case, the neuron's output a_j^k influences the cost function through multiple paths. The output a_j^k is connected to several other neurons in layer $k+1$ and we need to take this into account, summing all these paths. Equation (10) shows it:

$$\frac{\partial J}{\partial a_j^k} = \sum_{j=0}^{n_{k+1}-1}\left(w_{ij}^{k+1}\cdot g'\left(z_j^{k+1}\right)\frac{\partial J}{\partial a_j^{k+1}}\right). \quad (10)$$

The second factor on the right-hand side of equation (8) calculates the derivative of the weighted input of the activation function in relation to the weights. Equation (11) calculates it:

$$\frac{\partial z_j^k}{\partial w_{ij}^k} = \frac{\partial\left(\sum_{i=1}^{n}(w_{ij}^k\cdot a_i^{k-1}+b_j^k)\right)}{\partial w_{ij}^k} = a_i^{k-1}, \quad (11)$$

This result means that the effect of a small change in the weight in neuron j, in layer k, depends on how strong the previous neuron's output a_i^{k-1} was.

Finally, the last factor of equation (8) calculates the derivative of the output of neuron j, in layer k, with respect to the weighted input z_j^k is simply the derivative of the activation function $g(\cdot)$ shown below:

$$\frac{\partial a_j^k}{\partial z_j^k} = \frac{\partial g(z_j^k)}{\partial z_j^k} = g'(z_j^k), \qquad (12)$$

Inserting equations (9), (10), (11) and (12) in equation (8) we obtain:

$$\frac{\partial J}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial z_j^k}\frac{\partial z_j^k}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial z_j^k}\, a_i^{k-1} = \delta_j \cdot g'(z_j^k) \cdot a_i^{k-1}, \qquad (13)$$

with

$$\delta_j = \frac{\partial J}{\partial a_j^k} = \begin{cases} \dfrac{\partial J}{\partial \hat{y}(x)}, & if\ j\ is\ an\ output\ neuron \\ \displaystyle\sum_{j=0}^{n_{k+1}-1} \left(w_{ij}^{k+1} g'(z_j^{k+1}) \frac{\partial J}{\partial a_j^{k+1}} \right), & if\ j\ is\ a\ hidden\ neuron \end{cases} \qquad (14)$$

2.2.7.2 – Partial derivatives in relation to the biases

Computing the partial derivatives of the cost function in relation to the biases is almost identical to what was done for the weights. We have:

$$\frac{\partial J}{\partial b_{ij}^k}, \qquad (15)$$

which can be computed by the chain rule as:

$$\frac{\partial J}{\partial b_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial b_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial z_j^k}\frac{\partial z_j^k}{\partial b_{ij}^k}, \qquad (16)$$

Comparing equation (16) with equation (8), we see that the only difference is in the last term of the right-hand sides. It now depends on the bias instead of the weight. The calculation is as follows:

$$\frac{\partial z_j^k}{\partial b_{ij}^k} = \frac{\partial(\sum_{i=1}^{n}(w_{ij}^k \cdot a_i^{k-1} + b_j^k))}{\partial b_{ij}^k} = 1, \qquad (17)$$

So, similarly to equation (13), now we have:

$$\frac{\partial J}{\partial b_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial z_j^k}\frac{\partial z_j^k}{\partial b_{ij}^k} = \frac{\partial J}{\partial a_j^k}\frac{\partial a_j^k}{\partial z_j^k}1 = \delta_j \cdot g'\left(z_j^k\right), \qquad (18)$$

2.2.8 - Recurrent Neural Networks

Classical neural networks represented in Figure 1 are also called feedforward neural networks because information moves from the input to the output layer in a single direction. In the recurrent neural networks (RNN) structure, the information moves cyclically in the network using additional synapses. They are a special case of neural networks where the objective is to predict future steps in a sequence of observations (Namini & Namin, 2018). This means that the output of an RNN is based on previous elements of a given sequence, while the output of a standard feedforward neural network depends only on the current input (Lindholm & Palmborg, 2022). Figure 5 below shows a typical RNN.

To predict future steps in a sequence of observations, earlier stages of data need to be "remembered" and the hidden layers of RNNs act as memory storage for keeping information captured in earlier stages (Namini & Namin, 2018).

Although they are a powerful structure, the major drawback of RNNs is that they only remember a few steps in the sequence of data and therefore are not appropriate to work with long sequences (Abiodun et al., 2019; Namini & Namin, 2018). Another issue results from the fact RNNs are recurrent, making the same function to be composed with itself many times, and this leads to the vanishing gradient problem (Lindholm & Palmborg, 2022). When the backpropagation algorithm advances backward from the output layer to the input layer, the gradients often get smaller and smaller, which eventually leaves the weights of the initial layers

nearly unchanged. Because of that, the algorithm never converges to the optimal value. This is the vanishing gradient problem.

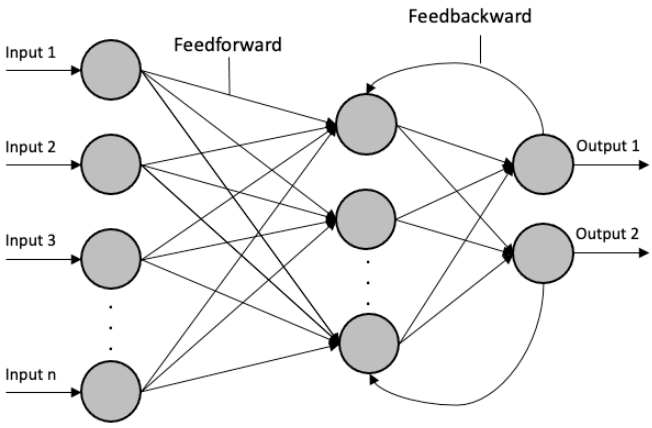


Figure 5 - Typical structure of an RNN.

Source: Adapted from Abiodun et al. (2019)

2.2.9 - Long Short-Term Memory

As RNNs are not suitable to model long-term dependencies, they are not a natural choice for time series modeling. To overcome the problem presented by RNNs, the LSTM model was developed as an improvement by (Hochreiter & Schmidhuber, 1997) and further extended by (Gers, Schmidhuber & Cummins, 2000). LSTMs are RNNs whose architecture is built in such a way that it allows for considering the relationships between the data of the sequence, even if it is long, and eliminates the vanishing gradient problem. Hence, LSTMs acquire both long and short-term memory (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019).

Figure 6 shows the basic structure of an LSTM neural network. The network receives an input, which is processed by a neuron in the input layer. After that, its output passes through an LSTM cell and part of the output of this cell goes to the other neurons in the network and another part goes to another LSTM cell.

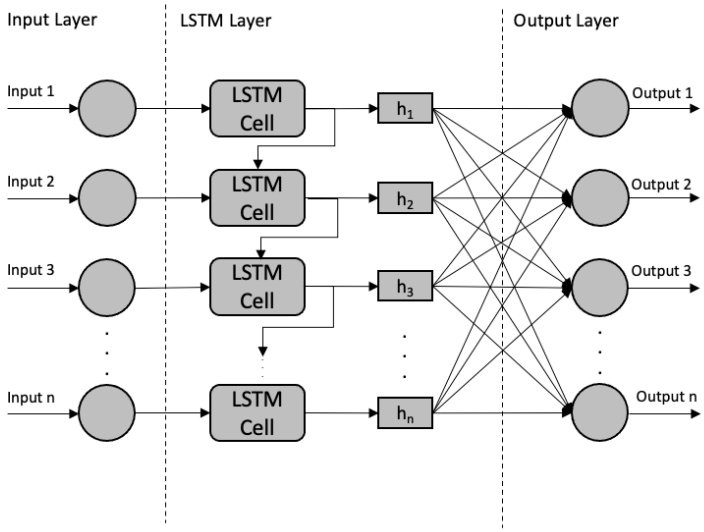


Figure 6 - Neural network with LSTM cell

Source: Author

Figure 7 illustrates what is inside an LSTM cell.

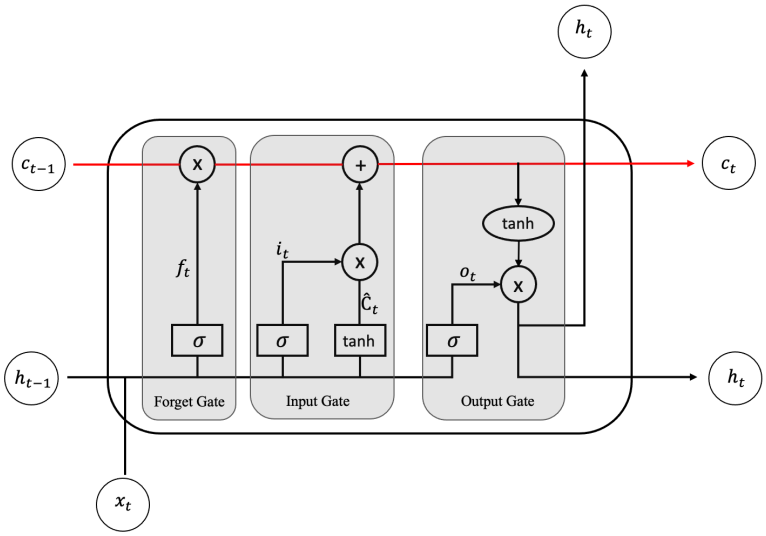


Figure 7 - LSTM Cell

Source: Adapted from (Choi & Lee, 2020)

In Figure 7, the red line is the cell state vector, which represents the long-term memory of the LSTM model (Choi & Lee, 2020). In the cell state vector, the information C_{t-1} enters the LSTM cell at learning step $t-1$ and leaves it as C_t at learning step t. The three structures called gates (forget gate, input gate, and output gate) determine how much information should be carried out to the next step (Mirzaei, Kang & Chu, 2022). In Figure 7, the + symbols

represent the mathematical addition, × symbols represent the element-wise multiplication and h_t is the short-term duration memory of the LSTM cell at time step t.

To understand the LSTM cell, we need to understand what each of the three gates does. The forget gate is responsible for information filtering (Škrlj, Kralj, Pollak & Lavrač, 2019). The forget activation f_t takes the output of the previous hidden state and the input of data. Then, it outputs a value between 0 and 1, where 0 means "forgets everything" and 1 means "remembers everything". Finally, the output of f_t is multiplied by C_{t-1}, the long-term memory of the LSTM cell, determining to what extent to forget from the previous data.

In the input gate, the model decides which new information is going to be added to the cell state vector. For doing this, it multiplies i_t by $\hat{C}_t$ (called the candidate state) and adds the result to the cell state. In equations (19) and (22) below we see that i_t is the output of a sigmoid function, bounding between 0 and 1, and that $\hat{C}_t$ is the output of a tangent hyperbolic function, which bounds between -1 and 1. In practical terms, what happens in the input gate is that $\hat{C}_t$ decides which new candidate values should be added to the cell state C_t while i_t controls to what extent they should be added.

In the output gate, the LSTM cell controls the new values of the hidden state (the short-term memory of the model). These values are based on the cell state but are filtered for relevant information. The cell state is run through the hyperbolic tangent function to regularize values between -1 and 1. Then this output is multiplied by o_t to decide which part of the information should be taken. Equations (19)-(24) summarize how each part of the LSTM cell is calculated:

$$f_t = \sigma\big(W_f[x_t, h_{t-1}] + b_f\big), \quad (19)$$

$$i_t = \sigma(W_i[x_t, h_{t-1}] + b_i), \quad (20)$$

$$o_t = \sigma(W_o[x_t, h_{t-1}] + b_o), \quad (21)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t, \quad (22)$$

$$\hat{C}_t = tanh(W_c[x_t, h_{t-1}] + b_c), \quad (23)$$

$$h_t = o_t \odot tanh(C_t), \qquad (24)$$

where f_t is the forget activation, σ is the logistic sigmoid function, W is the matrix of weights, x_t is the input data, h_{t-1} is the short-term memory at time step $t-1$, b is the bias, i_t is the input activation, o_t is the output activation, C_t is the cell state, C_{t-1} is the cell state at time step $t-1$, $\hat{C}_t$ is the candidate values to update the cell state C_t, $tanh$ is the hyperbolic tangent function, t is learning time step and $\odot$ is the element-wise multiplication. The subscripts f, i and o represent the forget, input and output gate, respectively.

Chapter 3 - Data

Hospitalization data from the state of São Paulo was gathered from *Serviço de Informações Hospitalares do Sistema Único de Saúde* (Hospital Information System of the Unified Health System - SIH/SUS) and the data from the population resident in the state of São Paulo was gathered from *Pesquisa Nacional por Amostragem de Domicílios do Instituto Brasileiro de Geografia e Estatística* (National Sample Household Survey of the Brazilian Institute of Geography and Statistics - PNAD/IBGE), both from Jan/2008 until Nov/2019. The hospitalization data encompasses all private and public hospital admissions.

The hospitalization data is monthly available while the data from the population resident in the state of São Paulo is only available at the end of the year. This annual data was transformed into monthly data, assuming that each observed annual increase/decrease in the resident population occurred uniformly during the year, a very common method described in (United Nations, 1952).

The choice of gathering data from 2008 onwards was made because this is the first year when the dates of hospital admissions started to be registered. Also, the date of Nov/2019 was chosen as the final year because of the COVID-19 pandemic in Brazil. The first COVID infection was reported in Feb/2020 by Brazilian authorities but even though we already can see great distortions in data from Dez/19 onwards.

The data provided by SIH/SUS is disposed in 18 age groups, being them: < 1 year old, 1-4 years old, …, 75-79 years old, 80+ years old. On the other hand, the data provided by PNAD/IBGE is disposed in 15 age groups: 0-4 years old, …, 70+ years old. To overcome this mismatch, the data from SIH/SUS was standardized to the same standard as the PNAD/IBGE data. Therefore, the data is disposed in 15 five-year age groups, starting from 0-4 years to 65-69 years, and the last group 70+ years.

Figure 8 illustrates how the age group data from SIH/SUS was put in the same standard as the data of PNAD/IBGE. It was simply done by summing the SIH/SUS data into the PNAD/IBGE age group standard.

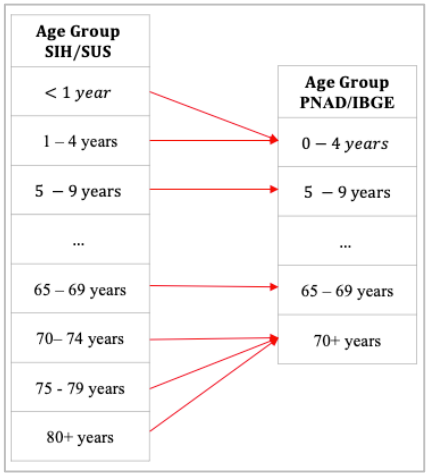


Figure 8 - SIH/SUS and PNAD/IBGE data standardization

Source: Author

Figure 9 and Figure 10 show boxplots for each age group of the hospital admission dataset, from 2008 until 2019. The number of hospital admissions is similar for both genders, except between age groups 10-14 to 40-44 years old. By further analyzing the data, we see that this is driven by *pregnancy, childbirth, and puerperium*, accounting for an overall of 60.8% of hospital admissions within these age groups for females.

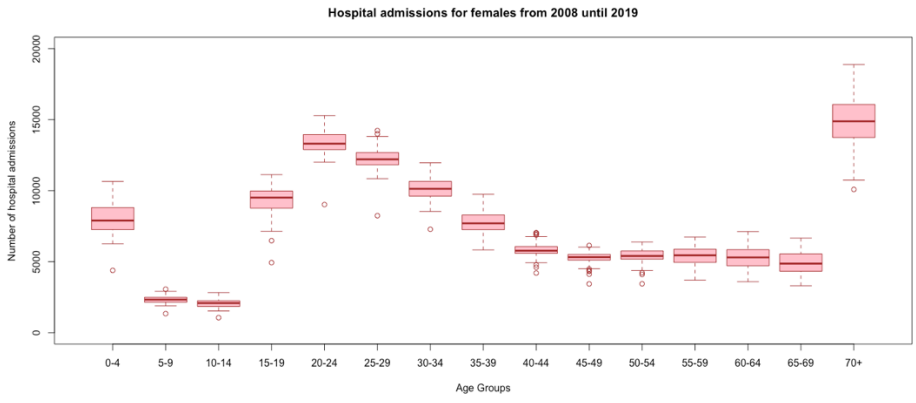


Figure 9 - Hospital admissions of females by age group

Source: Author, based on SIH/SUS data

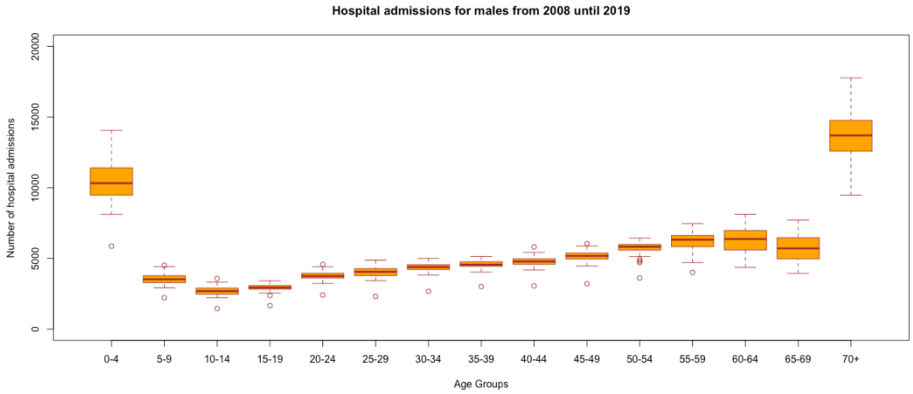


Figure 10 - Hospital admissions of males by age group

Source: Author, based on SIH/SUS data

Table 2 shows in detail the percentage of female hospital admissions that are related to *pregnancy, childbirth, and puerperium*. For the other age groups, the percentage is roughly zero and, therefore, they are not shown in the table.

Age Group	Percentage
10 – 14 years	14.9%
15 – 19 years	74.4%
20 – 24 years	78.3%
25 – 29 years	70.8%
30 – 34 years	59.8%
35 – 39 years	43.6%
40 – 44 years	18.6%
Overall	**60.8%**

Table 2 - Hospital admissions related to pregnancy, childbirth, and puerperium

Source: Author, based on SIH/SUS data

Figure 11 shows the female hospitalizations from Jan/2008 until Nov/2019, by age group. We see how the number of hospitalizations is distributed, being the age group 70+ years the highest one in the absolute number of hospital admissions.

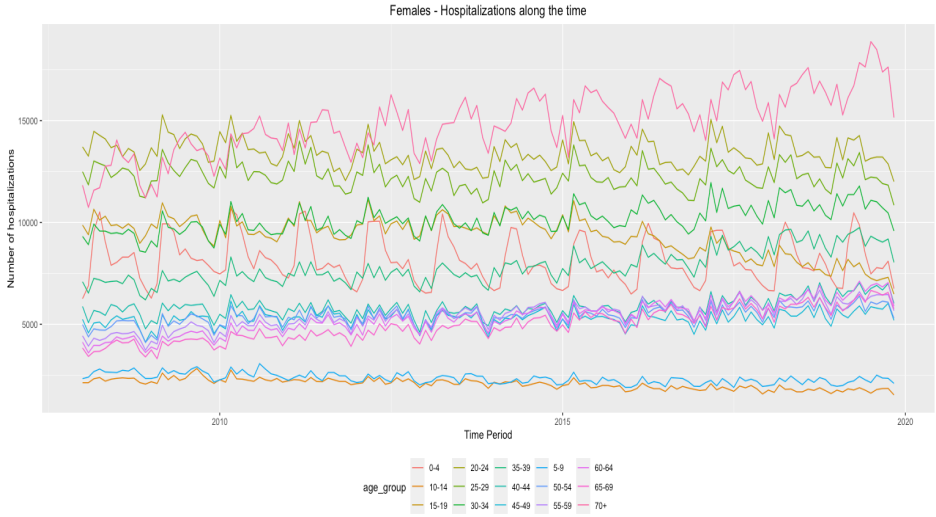


Figure 11 - Female hospital admissions from Jan/2008 until Nov/2019

Source: Author, based on SIH/SUS data

Similarly to Figure 11, Figure 12 shows the number of hospitalizations for males, by each age group. We can see that the age groups 0-4 years and 70+ years are the ones with the highest number of hospitalizations. In the other age groups, differently to what happens to females, the numbers of hospital admissions, by age group, are more similar.

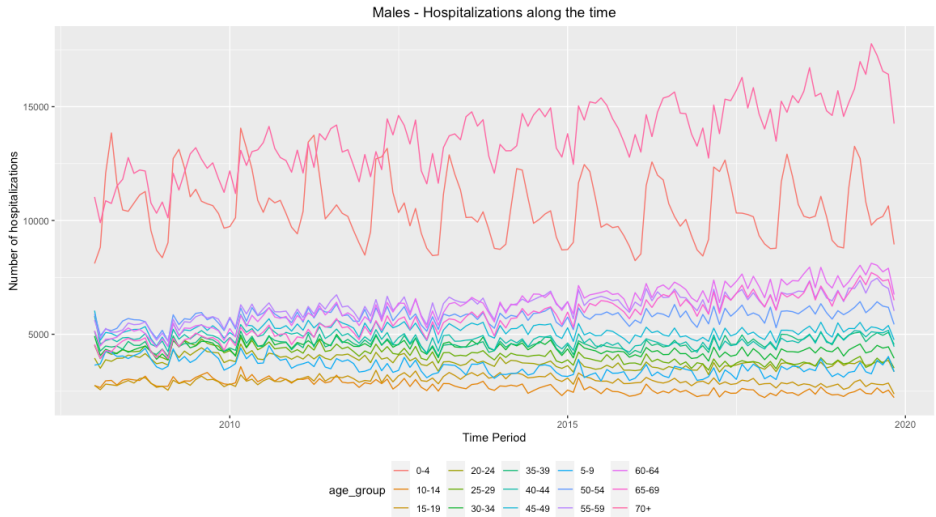


Figure 12 - Male hospital admissions from Jan/2008 until Nov/2019

Source: Author, based on SIH/SUS data

Figure 13 and Figure 14 show the residents' data, by age group, for each gender. As shown by the charts, the distribution is similar for both genders. In Table 3, we see that in general females are more numerous than males, especially in older age groups. This is related to the fact that females have higher life expectancy than males.

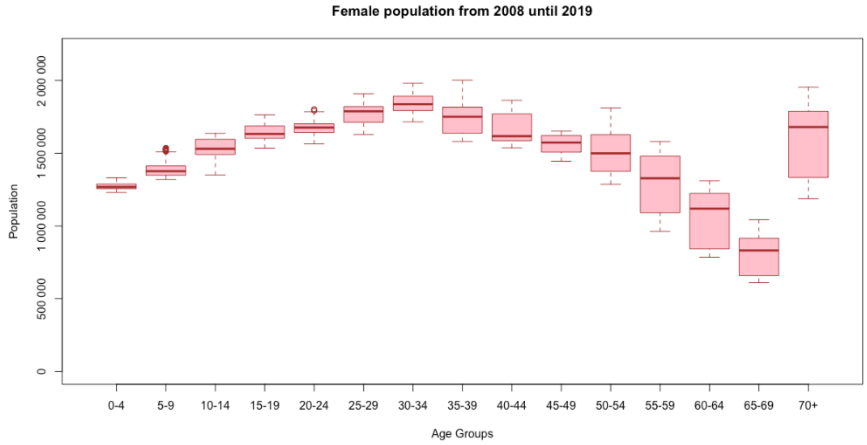


Figure 13 - Female population by age group

Source: Author, based on PNAD/IBGE data

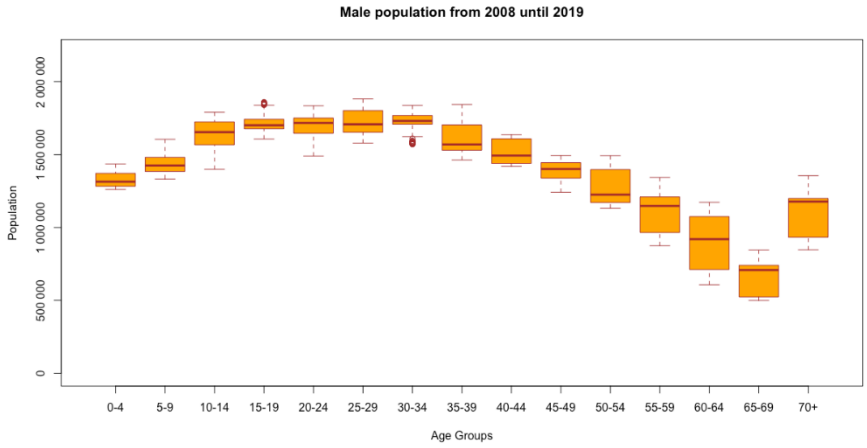


Figure 14 - Male population by age group

Source: Author, based on PNAD/IBGE data

Age Groups	Females	Males
0-4 years	49.0%	51.0%
5-9 years	49.2%	50.8%
10-14 years	48.8%	51.2%
15-19 years	49.3%	50.7%
20-24 years	49.9%	50.1%
25-29 years	51.2%	48.8%
30-34 years	51.7%	48.3%
35-39 years	52.0%	48.0%
40-44 years	52.3%	47.7%
45-49 years	53.0%	47.0%
50-54 years	53.3%	46.7%
55-59 years	53.4%	46.6%
60-64 years	54.1%	45.9%
65-69 years	55.3%	44.7%
70+ years	59.2%	40.8%
Total	**51.6%**	**48.4%**

Table 3 - Population distribution by gender

Source: Author, based on PNAD/IBGE data

Chapter 4 - Estimation and Prediction

In this chapter, we will obtain the parameters a_x and b_x as in (Lee & Carter, 1992) and we will estimate the k_t trend by an $ARIMA(p,d,q)$ model, divided by gender. We follow (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019) and estimate the $ARIMA(p,d,q)$ by using the *auto.arima* function from the R package *forecast* see (Hyndman & Khandakar, 2008; Hyndman et al., 2022).

This package applies the Hyndman-Khandakar algorithm (Hyndman & Khandakar, 2008), to automatically select the best $ARIMA(p,d,q)$ model for a given time series. This algorithm works in two steps. In the first one, it chooses the best differencing order d by checking the stationarity of the time series using a unit root test. In the second step, the algorithm selects the best values of auto-regressive and moving average orders, p and q, using an information criterion (AIC or BIC).

The result obtained by the $ARIMA(p,d,q)$ model is compared with the forecasts made by the LSTM neural network. The estimations intend to predict the last 12 months of the fitted k_t.

4.1 - Estimation of the Lee-Carter model

We obtain the parameters of the Lee-Carter model as explained in Chapter 2. The process was developed for both genders, obtaining k_t for males and females from 2008 to 2019. Figure 15 shows k_t for both genders.

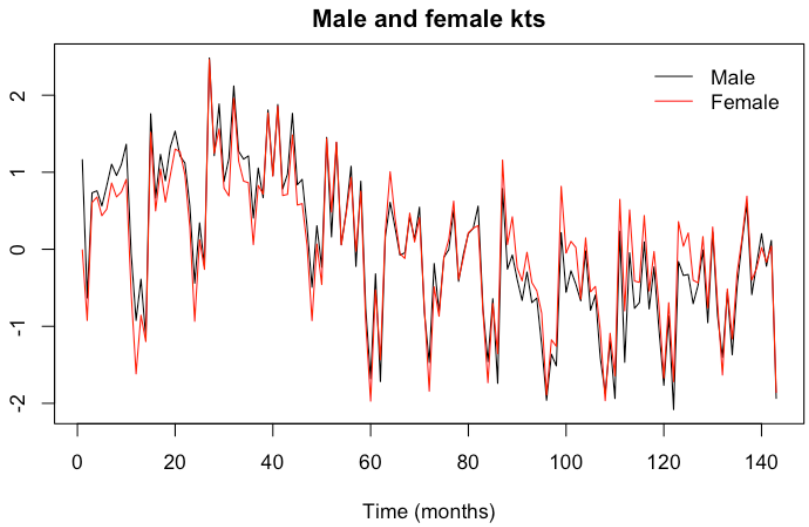


Figure 15 - Male and female fitted kts

Source: Author

Table 4 shows the fitted a_x and b_x for each age group and gender

Age Group	Female		Male	
	a_x	b_x	a_x	b_x
0-4	-5.065297	0.10449056	-4.845265	0.08736755
5-9	-6.391685	0.07711722	-6.002098	0.06256431
10-14	-6.613553	0.07728216	-6.405589	0.05413556
15-19	-5.176530	0.06819218	-6.368512	0.04540376
20-24	-4.826560	0.02003936	-6.106506	0.03171212
25-29	-4.974457	0.02094504	-6.049290	0.04420463
30-34	-5.197766	0.02922902	-5.972261	0.06880108
35-39	-5.416896	0.04026236	-5.862534	0.07340256
40-44	-5.656251	0.06463627	-5.762601	0.08326992
45-49	-5.690644	0.07470302	-5.599314	0.06810099
50-54	-5.621203	0.08955646	5.397862	0.07876780
55-59	-5.479867	0.08348134	-5.181277	0.06963178
60-64	-5.301539	0.08582828	-4.965034	0.08479263
65-69	-5.091800	0.07347616	-4.740753	0.06855017
70+	-4.668532	0.09076058	-4.391226	0.07929514

Table 4 - Fitted a_x and b_x for females and males, by age group

Source: Author

4.2 - Time series model

Since the k_t presents monthly seasonality, a SARIMA model was used for the prediction. The *auto.arima* package in R deals with it easily. The series was split into two, one for training the model with the first 131 data points of the series and another one for testing the model with the

final 12 months of data. In the prediction, the next 12 months of k_t were predicted. The *auto.arima* package chose the same $ARIMA(p, d, q)$ for both genders. This is not a surprise since both genders have similar trends, as it was shown in Figure 15. Below, there are the plots for both genders with a 95% prediction interval. The results are summarized in Table 7, in section 4.4. Further results of the model are shown in Appendix I.

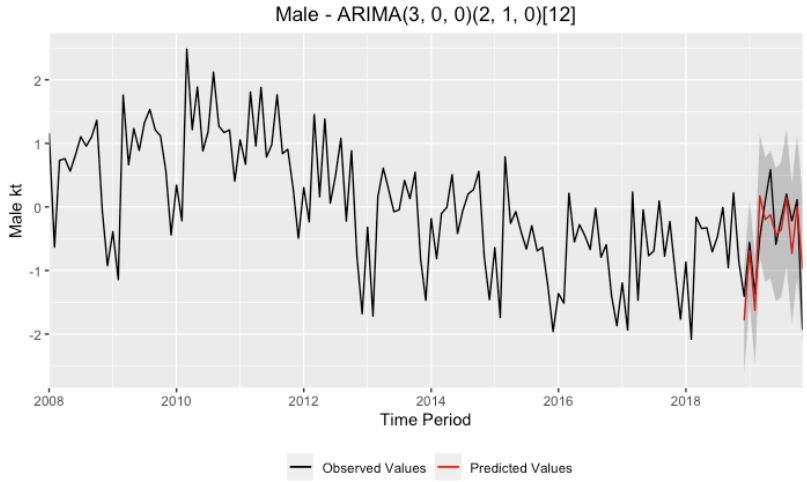


Figure 16 - Male kt prediction for the next 12 months

Source: Author

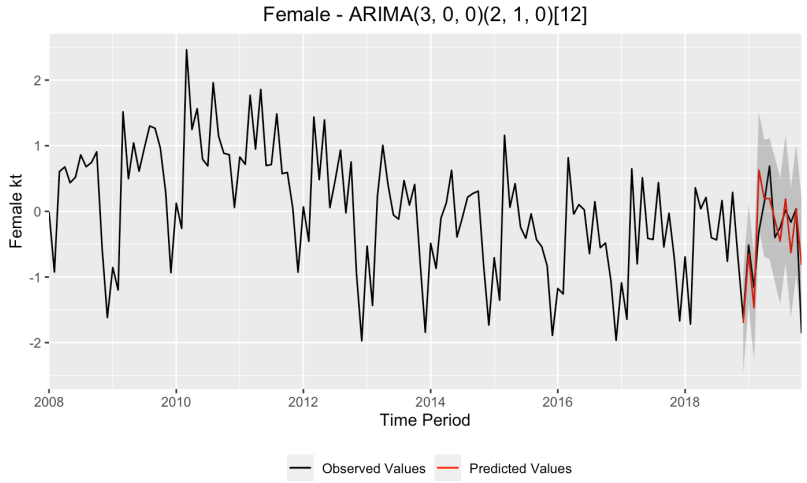


Figure 17 - Female kt prediction for the next 12 months

Source: Author

The results obtained are in line with the results of (Rodrigues, Andrade, Queiroz & Machado, 2013) although not directly comparable. In their work, they used annual data from the state of Minas Gerais, Brazil while in this dissertation we used monthly data from the state of São Paulo, Brazil. The choice for using monthly data is to have more data points that can

improve the estimations while the choice of the state of São Paulo is because it is the largest state in Brazil, accounting for 22% of the Brazilian population, more than double the state of Minas Gerais, which accounts for 10% of the population (IBGE, 2021).

4.3 - LSTM model

In this chapter, we describe in detail how the LSTM neural network model was built. The model was implemented in R, using the packages *Keras* (Chollet, 2015) and *TensorFlow* (Abadi et al., 2015). A major aspect is that the neural network demands great data preparation before the model can be fit.

4.3.1 - Data scaling

As done in section 4.2, the data was split into training and test. For the neural network, it also needs to be scaled before fitting the model. The backpropagation algorithm, presented in section 2.2.7, converges faster when the data provided as input has its mean close to zero (LeCun, Bottou, Orr & Muller, 2012). For standardizing the data, the mean and standard deviation of the training dataset is calculated, and their values can be seen in Table 5.

Statistic	**Female k_t**	**Male k_t**
Mean	0.09714466	0.1279912
Standard Deviation	0.91980280	0.9871962

Table 5 - Mean and standard deviation of females and males from the training dataset

Source: Author

With these figures, the training and test datasets were then standardized by applying the usual formula, see Equation (25).

$$z = \frac{x - \bar{x}}{\hat{\sigma}}, \quad (25)$$

where z is the standardized data, x is the original data, $\bar{x}$ is the mean and σ is the standard deviation.

It is important to note that the test dataset also needs to be standardized with the same parameters used to standardize the training dataset. In this way, both the test and training data will be on the same scale.

4.3.2 - Sliding window and differencing

After being standardized, the data keeps the same dimensions as the original data (a single vector), but it needs to be put in a sliding window format for the neural network (Brownlee, 2018). To do so, the data is split into two different datasets, one called input and the other one called output. Figure 18 shows how the sliding window is applied.

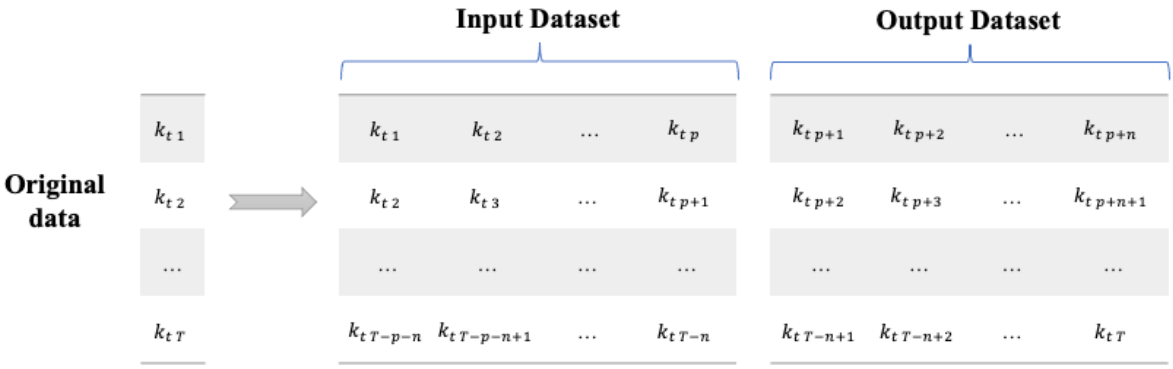


Figure 18 - LSTM sliding window

Source: Adapted from (Neves, 2018)

T represents the length of the time series, p represents the length of the training sequence and n represents the length of the output sequence.

In the sliding window, the input and output data have the same length, being the output equal to the input shifted one-time step (Neves, 2018). The values of ($k_{t\,1}$, …, $k_{t\,12}$) are used to predict the values of ($k_{t\,13}$, …, $k_{t\,24}$), the values of ($k_{t\,2}$, …, $k_{t\,13}$) are used to predict the values of ($k_{t\,14}$, …, $k_{t\,25}$), and so on. This is how sliding works in the training step of the model. After this step, the neural network has learned the input-output relationship of the data and it should be able to predict future values (Nigri, Levantesi, Marino, Scognamiglio & Perla, 2019).

In this work we want to predict the next 12 months ($n = 12$). Therefore, we build the datasets in such a way that 12 data points are predicted by the past 12 data points. It is important to note that differencing was not applied to the data used for the neural network model. LSTMs do not require previous information of the time series structure and are less subject to time series stationarity (Silva, Steen & Darley, 2019). Research suggests that these models are more flexible to work with non-stationarity data (Silva, Steen & Darley, 2019).

4.3.3 - Tridimensional form

Another characteristic of the LSTM neural network is that it needs to be fed with tridimensional data in the form of *[samples, time steps, features]* (Brownlee, 2018). Where:

- *samples* specify the number of observations in the dataset
- *time steps* specify the number of time steps we want the neural network to look back in time
- *features* specify the number of predictors of the series

The training dataset has the dimensions [96, 12, 1]. The first dimension is 96 because of the number of observations in the training dataset (as shown in Figure 18), the second dimension is 12 because it is the number of time steps used as input and the last dimension is 1 because it is a univariate time series, so it has only one predictor.

The test dataset has the dimensions of *[1, 12, 1]*. The first dimension is 1 because the test dataset has only one line of observations, composed of 12-time steps. The third dimension is again 1 because it is a univariate time series.

4.3.4 - LSTM general architecture

The past three sections presented how the data needs to be preprocessed before being used by the neural network. This section explains how the model structure was chosen and how it was fitted. The first task is to determine the general structure of the neural network (i.e. the number of neurons and layers). Unfortunately to choose the number of neurons and layers is not easy.

Neural network architecture typically relies on human knowledge and trial and error (Dong, Kedziora, Musial & Gabrys, 2021). Neural architecture search (NAS) has been proposed to automatically search for the best architecture for neural networks, but currently, algorithms suffer from computational cost (Jin, Song & Hu, 2019). For further readings into the NAS field see (Dong, Kedziora, Musial & Gabrys, 2021; Hu, Chu, Pei, Liu & Bian, 2021).

Given that it is not trivial to find the appropriate architecture for a neural network model, this work proposed to have a simple neural network model, without sophistication, that could be contrasted with the ARIMA model. The final architecture used in this work can be seen below in Figure 19.

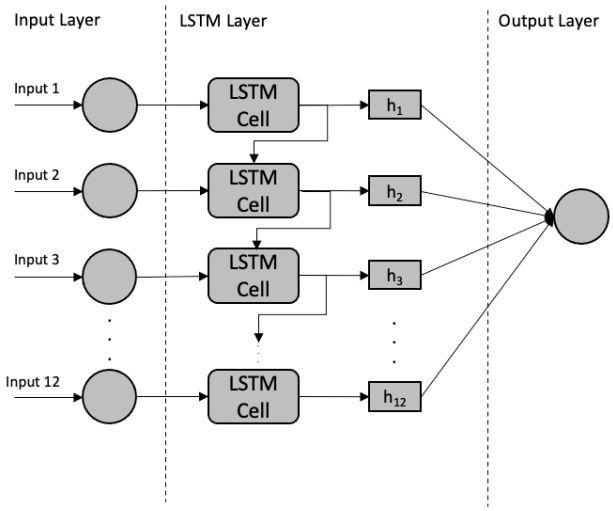


Figure 19 - LSTM architecture used in this work

Source: Author

The input layer has 12 neurons because each neuron represents one data point of the input sequence shown in Figure 18. Regarding the hidden layer, problems that require two or more hidden layers are not commonly seen (Heaton, 2005, p. 128), since training can be too difficult due to the increase in the number of parameters, overall complexity, and time to execute the model (Uzair & Jamil, 2020). Considering the complexities that adding hidden layers imposes and since in this work we are dealing with a univariate time series (only one predictor), we chose to have only one hidden layer in the neural network. The output layer is composed of a single neuron that outputs a sequence of 12 predictions. Even though the used

architecture seems to be simple, it has 685 adjustable parameters. Details of the model are in Appendix II.

4.3.5 - Learning Rate

Recall from section 2.2.6 that the learning rate is a parameter that should be empirically chosen. It is one of the most important parameters to adjust in neural network architecture (Bengio, 2012). A default value of 0.01 typically works for default neural networks but other values should be tested as well (Bengio, 2012). Values tested are usually small e.g., a learning rate within the set $\{0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$ (Goodfellow, Bengio & Courville, 2016, p. 436).

Despite only choosing a fixed value for the learning rate, some authors such as (Wang, You, Long & Jordan, 2019) have shown that using a learning rate decay can provide benefits for training neural networks. In this approach, the learning rate starts with a given value that decays by a factor during the iterations.

Considering that, in this work we have tested four learning rates and five decay values, comprising a total of 20 model combinations, per gender, to find the combination that minimizes the cost function. All models have the same structure, only varying the learning rate and decay factor. Table 6 shows the LSTM models combination made.

Model	Learning rate	Decay Factor
Model 1	0.001	0
Model 2	0.001	0.0000001
Model 3	0.001	0.00001
Model 4	0.001	0.001
Model 5	0.001	0.1
Model 6	0.002	0
Model 7	0.002	0.0000001
Model 8	0.002	0.00001
Model 9	0.002	0.001
Model 10	0.002	0.1
Model 11	0.01	0
Model 12	0.01	0.0000001
Model 13	0.01	0.00001
Model 14	0.01	0.001
Model 15	0.01	0.1

Model 16	0.02	0
Model 17	0.02	0.0000001
Model 18	0.02	0.00001
Model 19	0.02	0.001
Model 20	0.02	0.1

Table 6 - LSTM models

Source: Author

4.3.6 - Epochs

In the fitting step of the neural network, the model is iterated several times to adjust its parameters, in order to reduce the cost function. These number of iterations are called *epochs*. A training epoch refers to the number of passes of the entire training dataset through the neural network algorithm (Hastie, Tibshirani & Friedman, 2017, p. 397).

The number of epochs can be determined using a graphic approach, by plotting the cost function in relation to the number of epochs. Figure 20 shows the Mean Squared Error (MSE) cost function in relation to the number of epochs. By further analyzing the chart, we see that the error starts to rebound around the 25th epoch. This means that the gradient descent has already reached the minimum of the cost function and, since the iterations did not stop, it continues iterating around the minimum and rebounding. Also, we see that between the 20th and 25th epoch the error is still basically the same, it did not considerably decrease. Given that, the number of 20 epochs was chosen to be used in this work.

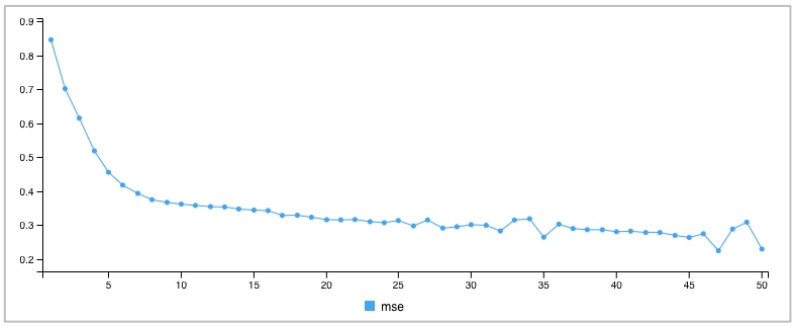


Figure 20 - Cost function in relation to the number of epochs

Source: Author

As explained in Chapter 2, the parameters of a neural network are randomly assigned. The backpropagation algorithm takes random weights and biases to minimize the cost function. Since it receives random parameters as input, each time the model is run it outputs a different result. Because of this stochastic behavior, to evaluate the model's performance it is necessary to run it several times and calculate the average of the error metrics (Brownlee, 2017). The minimum number of 30 times is recommended by (Brownlee, 2017), limited only by computational resources and time expend running the model.

Section 2.3.5 described that a total of 20 model combinations were made, by varying the learning rate and decay rate. It is worth noting that 30 random seeds were generated. That means that each round of each model has been initialized with the same weights and biases so any difference in performance can only be attributed to differences in the learning rate and decay. The LSTM models with the smallest average errors are shown in Table 7, in the next section. In Appendix II, the statistics are shown in detail.

4.4 - Comparison between models

To evaluate the forecast performance made by the standard ARIMA and the LSTM models, the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE) error metrics were calculated. The models presented in the 4.2 and 4.3 sections were used to predict the k_t from Dec/18 to Nov/19 and their results were then contrasted with the fitted k_t to calculate the error metrics.

Table 7 summarizes the performance of the ARIMA model, LSTM Model 3, and LSTM Model 20. The lowest errors are highlighted in **bold**. Model 3 is the one that presented the lowest errors for males and Model 20 is the one with the lowest errors for females.

Model	Female		Male	
	RMSE	**MAE**	**RMSE**	**MAE**
k_t ARIMA	0.4714	**0.3452**	**0.4550**	**0.3655**
k_t LSTM - Model 3	0.4974	0.4050	0.5635	0.4760
k_t LSTM - Model 20	**0.4633**	0.3582	0.5993	0.5059

Table 7 - Comparing forecast performance between ARIMA and LSTM

Source: Author

The error presented in Model 3, for males, is higher than the error presented by the ARIMA model. For the females, in Model 20, only the RMSE is lower than the RMSE of the ARIMA model. Overall, both RMSE and MAE showed no great differences for females.

With these error metrics calculated, we also performed the prediction of the next 12 months of k_t, between Dec/2019 and Nov/20, for both genders, predicting how the k_t would behave in the absence of the Covid-19 pandemic. The results are shown in Figure 21 and Figure 22. We can see that the ARIMA and the LSTM models have a similar pattern, which is not a surprise since the errors shown in Table 7 are similar.

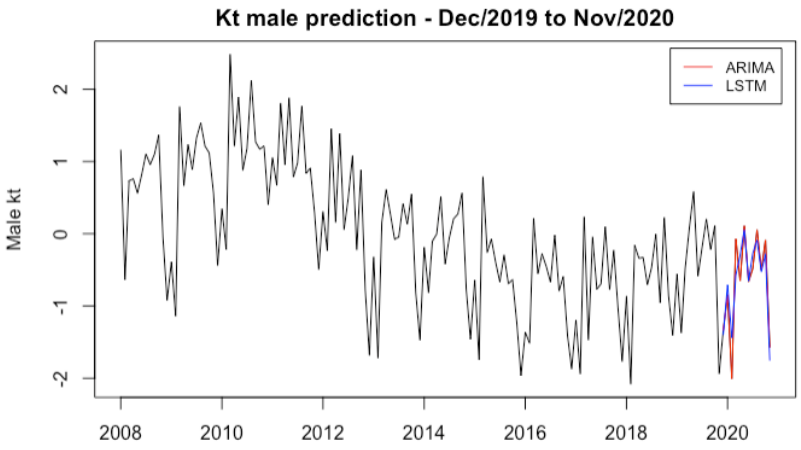


Figure 21 - Male kt prediction from Dec/19 to Nov/20

Source: Author

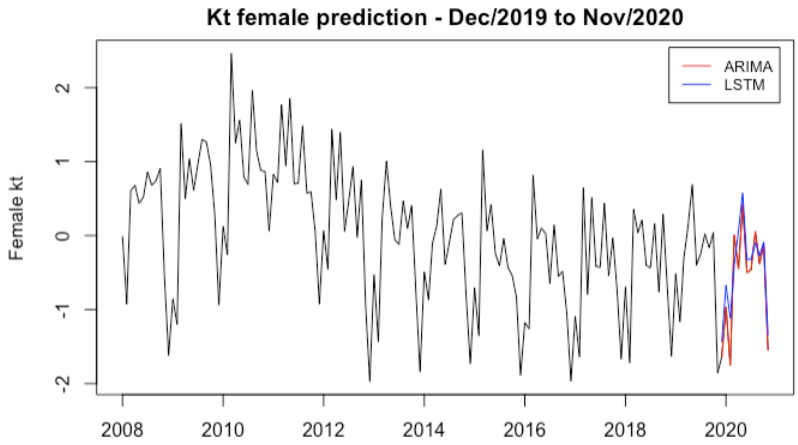


Figure 22 - Female kt prediction from Dec/19 to Nov/20

Source: Author

Chapter 5 - Conclusions

The main purpose of this work is to add a contribution to the set of possible applications of machine learning techniques in the actuarial science field. This dissertation focused on how to model the general level of hospital admission rates using the Long Short-Term Memory (LSTM) neural network and to contrast its results with the results obtained by the Lee-Carter model (Lee & Carter 1992), as done in the work of (Rodrigues, Andrade, Queiroz & Machado, 2013).

This study used publicly available datasets about hospital admissions and population from the state of São Paulo, Brazil. The *auto.arima* function from the R package *forecast* was used to search for the best *ARIMA(p, d, q)* and this result was compared with the result provided by a neural network model. A simple neural network architecture was implemented, and 20 LSTM models were built to search for the best combination of learning rates and decay factors, by each gender.

Each of these 20 models was run 30 times to average its RMSE and MAE results. This was performed on a personal computer, and it took around three hours, per gender, to run. In contrast, the *auto.arima* function took only a few seconds to find the best set of ARIMA parameters. Despite the much greater time spent in preparing the data and effectively running the neural network model, the results for females provided by the neural network and by the ARIMA model were similar. On the other hand, for males, the ARIMA model performed better than the LSTM, showing lower RMSE and MAE.

A 12-month k_t prediction between Dec/2019 and Nov/2019 was presented in Figure 21 and Figure 22, showing similarities between the predictions made by the ARIMA and LSTM models. These results intended to predict how the k_t of each gender would have behaved if the Covid-19 pandemic did not break through.

In practical terms, by modeling hospital admission rates actuaries can better assess the technical provisions of health insurance companies. Considering that hospital admissions account for approximately 50% of claim costs in the Brazilian health insurance market (Cechin & Lara, 2020), even small variations in hospital admission rates can put a serious strain on the liabilities of insurers. Additionally, in Brazil, there are the so-called *verticalized* health companies, which are health insurers that own hospitals and clinics in an attempt to control costs and frequency of use. For this kind of companies, modeling hospital admissions is even more important, since they can not only better assess their technical provisions but also better plan human and medical resources considering the level of predicted hospital admissions.

Even though a small data sample was used in this work, the performance of the LSTM and ARIMA models was similar. It is well known that neural networks models demand huge amounts of data to be fitted but his similar performance suggests that future works could rely on two approaches to test if the LSTM model can be significantly better than the ARIMA:

1. Work with longer data sequences. For example, health insurance companies always dispose of daily data. Comparing the ARIMA and LSTM on a daily sequence of data instead of a monthly one could show a significant difference in model performance.
2. Fine-tune the LSTM model. The structure of the LSTM model could be adjusted until finding a structure that best described the problem.

Considering point 2 introduced above, future work could explore the automatic search of parameters to adjust the neural network, but researchers and practitioners should be aware that it can be computationally costly. The work of (Jin, Song & Hu, 2019) proposed the package *AutoKeras* for this search, implemented in Python, which they claim to be efficient when compared to existing auto-search algorithms. Since this dissertation was done in R, this package could not be tested.

The use of regularization techniques in neural networks (i.e. L1 and L2 regularization and dropout) should also be explored, see (Hastie, Tibshirani & Friedman, 2017; Chollet, 2018). This gives a possibility to improve results and to study how the use of different regularization techniques impacts the performance of a neural network.

This research has shown that, for the given problem and available data, the LSTM and ARIMA models performed similarly in predicting the the general level of hospital admissions k_t. It is known that neural networks depend on huge amounts of data but, even though the data used was small, the neural network was able to have a similar result to the ARIMA model. This indicates that the LSTM model could perform better than the ARIMA model with longer data sequences and/or fine-tuning the LSTM model with a different structure and set of parameters. Modeling hospital admissions is useful for health actuaries to better estimate technical reserves and liabilities of insurers and also for verticalized companies to better plan resources accordingly to the predicted admission rates.

References

Abadi, M. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org.

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., . . . Kiru, M. U. (2019). Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition. *IEEE Access, 7*, 158820-158846. https://doi.org/10.1109/ACCESS.2019.2945545

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. In *Neural Networks: Tricks of the Trade* (2nd edition, pp. 437-478). Berlin, Heidelberg: Springer.

Bre, F., Gimenez, J. & Fachinotti, V. (2017). Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *Energy and Buildings, 158,* 1429-1441. https://doi.org/10.1016/j.enbuild.2017.11.045

Brownlee, J. (2017). *Long Short-Term Memory Networks With Python: Develop Sequence Prediction Models With Deep Learning.* Machine Learning Mastery.

Brownlee, J. (2018). *Deep Learning for Time Series Forecasting - Predict the Future with MLPs, CNNs, and LSTMs in Python.* Machine Learning Mastery.

Cauchy, A. (1847). Méthode Général pour la résolution des systèmes d'équations simultées. *Comptes Rendus Hebd. Séances Acad. Sci., 25*(2), 536-538.

Cechin, J. & Lara, N. (2020). *Análise Especial do Mapa Assistencial da Saúde Suplementar no Brasil Entre 2014 e 2019.* São Paulo: Instituto de Estudos da Saúde Suplementar. https://www.iess.org.br/biblioteca/tds-e-estudos/estudos-especiais-do-iess/analise-especial-do-mapa-assistencial-da-saude

Choi, J. & Lee, S. (2020). Consistency Index-Based Sensor Fault Detection System for Nuclear Power Plant Emergency Situations Using an LSTM Network. *Sensors, 20*(6), 1651. https://doi.org/10.3390/s20061651

Chollet, F. (2015). Keras. https://keras.io.

Chollet, F. (2018). *Deep Learning with Python.* United States of America: Manning Publications Co.

Curry, H. B. (1944). The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics, 2*(3), 258-261. https://doi.org/10.1090/qam/10667

Deaton, A. & Paxson, C. (2004). *Mortality, Income and Income Inequality Over Time in Britain and the United States.* In *Perspectives on the Economics of Aging (pp. 247-286).* National Bureau of Economic Research, Inc.

Deprez, P., Shevchenko, P. V. & Wüthrich, M. V. (2017). Machine Learning Techniques for Mortality Modeling. *European Actuarial Journal, 7*, 337-352. https://doi.org/10.1007/s13385-017-0152-4

Dong, X., Kedziora, D. J., Musial, K. & Gabrys, B. (2021). Automated Deep Learning: Neural Architecture Search Is Not the End. *arXiv.* https://doi.org/10.48550/arxiv.2112.09245

European Commission, Directorate-General for Economic and Financial Affairs, Schwierz, C., Medeiros, J. (2013). *Estimating the drivers and projecting long-term public health*

expenditure in the European Union: Baumol's «cost disease» revisited, European Commission. https://data.europa.eu/doi/10.2765/54565

Frees, E. (2006). Forecasting labor force participation rates. *Journal of Official Statistics, 22*(3), 453-485.

Gers, F., Schmidhuber, J. & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation, 12*(10), 2451-2471. https://doi.org/10.1162/089976600300015015

Girosi, F. & King, G. (2008). *Demographic Forecasting.* Princeton University Press.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning.* MIT Press.

Gutterman, S. & Vanderhoof, I. T. (1998). Forecasting changes in mortality: a search for a law of causes and effects. *North American Actuarial Journal, 2*(4), 135-138.

Hadamard, J. (1908). Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées. *Memoires presentés par divers savants a l'Académie des Sciences de l'Institut National de France, 33*(4).

Hainaut, D. (2018). A Neural-Network Analyzer for Mortality Forecast. *ASTIN Bulletin, 48*(2), 481-508. https://doi.org/10.1017/asb.2017.45

Hastie, T., Tibshirani, R. & Friedman, J. (2017). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd Edition). Springer.

Haykin, S. S. (2008). *Neural Networks and Learning Machines* (3rd Edition). Person.

Heaton, J. (2005). *Introduction to Neural Networks with Java* (1st Edition). Heaton Research.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hollmann, F. W., Mulder, T. J. & Kallan, J. E. (2000). *Methodology and Assumptions for the Population Projections of the United States: 1999 to 2100.* (Working Paper No. POP-WP038), U.S. Bureau of Census, Population Division.

Hu, X., Chu, L., Pei, J., Liu, W. & Bian, J. (2021). Model Complexity of Deep Learning: A Survey. *Knowledge and Information Systems, 63*, 2585-2619. https://doi.org/10.1007/s10115-021-01605-0

Hyndman, R., Athanasopoulos, G., Bergmeir, C., Caceres, G., Chhay, L., O'Hara-Wild M., Petropoulos F., Razbash S., Wang E. & Yasmeen F. (2022). forecast: Forecasting Functions for Time Series and Linear Models. R package version 8.16, URL: https://pkg.robjhyndman.com/forecast/

Hyndman, R. J. & Khandakar, Y. (2008). Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software, 26*(3), 1–22. https://doi.org/10.18637/jss.v027.i03

IBGE. Instituto Brasileiro de Geografia e Estatística (2021). Estimativas da População Residente no Brasil e Unidades da Federação com data de Referência em 1 de Julho de 2021. Retrieved from https://biblioteca.ibge.gov.br/visualizacao/livros/liv101849.pdf

Jin, H., Song, Q. & Hu, X. (2019). Auto-Keras: An Efficient Neural Architecture Search System. *arXiv*. https://doi.org/10.48550/arXiv.1806.10282

Kjærgaard, S. & Bergeron-Boucher, M.-P. (2022). Mortality forecasting at age 65 and above: an age-specific evaluation of the Lee-Carter model. *Scandinavian Actuarial Journal*, *1*, 64-79. https://doi.org/10.1080/03461238.2021.1928542

Koissi, M. C., Shapiro, A. & Högnäs, G. (2005). Fitting and forecasting mortality rates for nordic countries using the Lee-Carter method. *Actuarial Research Clearing House*, 1-21.

Krogh, A., Hertz, J. A. & Thorbergsson, G. I. (1990). A cost function for internal representations. In *Advances in Neural Information Processing Systems 2* (pp. 733-740). Morgan Kaufmann Publishers Inc.

Lalis, J. T., Gerardo, B. D. & Byun, Y. (2014). An Adaptive Stopping Criterion for Backpropagation Learning in Feedforward Neural Network. *International Journal of Multimedia and Ubiquitous Engineering, 9*(8), 149-156. https://doi.org/10.14257/ijmue.2014.9.8.13

LeCun, Y., Bottou, L., Orr, G. & Muller, K.-R. (2012). Efficient BackProp. In *Neural Networks: tricks of the trade* (pp. 9-48). Springer.

Lee, R. & Carter, L. (1992). Modeling and Forecasting US Mortality. *Journal of the American Statistical Association, 87*, 659–671. https://doi.org/10.2307/2290201

Lee, R. & Miller, T. (2002). An Approach to Forecasting Health Expenditures with Application to the U.S. Medicare System. *Health Services Research (Ann Arbor), 37*(5), 1365-1386. https://doi.org/10.1111/1475-6773.01112

Lemarechal, C. (2012). Cauchy and the gradient method. *Documenta Mathematica, Extra Volume ISMP*, 251-254.

Levantesi, S. & Pizzorusso, V. (2019). Application of machine learning to mortality modeling and forecasting. *Risks, 7*(1), 26. https://doi.org/10.3390/risks7010026

Lindberg, C. & McCarthy, T. (2021). Impact of Demographic Change on Health Expenditure 2022-2025. Report prepared by the Irish Government Economic and Evaluation Service (IGEES) staff in the Department of Health.

Lindholm, M. & Palmborg, L. (2022). Efficient Use Of Data For LSTM Mortality Forecasting. *European Actuarial Journal*. https://doi.org/10.1007/s13385-022-00307-3

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115–133. https://doi.org/10.1007/BF02478259

Menec, V., Lix, L., Nowick, S. & Ekuma, O. (2007). Health care use at the end of life among older adults: Does it vary by age? *Journal of Gerontology, 62*(4), 400-407. https://doi.org/10.1093/gerona/62.4.400

Minsky, M. & Seymour, P. (2017). *Perceptrons, Reissue of the 1988 Expanded Edition with a new Foreword by Léon Bottou.* Cambridge: Massachusetts Institute of Technology Press.

Mirzaei, S., Kang, J.-L. & Chu, K.-Y. (2022). A comparative study on long short-term memory and gated recurrent unit neural networks in fault diagnosis for chemical processes using visualization. *Journal of the Taiwan Institute of Chemical Engineers, 130*. https://doi.org/10.1016/j.jtice.2021.08.016

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* Cambridge: MIT Press.

Namini, S. & Namin, A. (2018). Forecasting Economic and Financial Time Series: ARIMA vs. LSTM. *ArXiv.* https://doi.org/10.48550/arXiv.1803.06386

Narkhede, M. V., Bartakke, P. P. & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial Intelligence Review, 55*, 291-322. https://doi.org/10.1007/s10462-021-10033-z

Neves, R. (2018). *An Overview of Deep Learning Strategies for Time Series Prediction* (Master's thesis, Instituto Superior Técnico, Lisbon, Portugal). Retrieved from https://fenix.tecnico.ulisboa.pt/downloadFile/1126295043835783/

Nielsen, M. A. (2015). *Neural Networks and Deep Learning.* Determination Press.

Nigri, A., Levantesi, S. & Marino, M. (2021). Life expectancy and lifespan disparity forecasting: a long short-term memory approach. *Scandinavian Actuarial Journal 2021, 2*, 110-133. https://doi.org/10.1080/03461238.2020.1814855

Nigri, A., Levantesi, S., Marino, M., Scognamiglio, S., & Perla, F. (2019). A Deep Learning Integrated Lee–Carter Model. *Risks, 7*(1), 33. https://doi.org/10.3390/risks7010033

Paris, V., Devaux, M. & Wei, L. (2010). Health systems institutional characteristics: A survey of 29 OECD countries. *OECD Working Papers, No. 50, OECD Publishing, Paris*. https://doi.org/10.1787/5kmfxfq9qbnr-en

Perla, F., Richman, R., Scognamiglio, S. & Wüthrich, M. (2021). Time-series forecasting of mortality rates using deep learning. *Scandinavian Actuarial Journal 2021*, *7*, 572-598. https://doi.org/10.1080/03461238.2020.1867232

Rabbi, A. & Mazzuco, S. (2020). Mortality Forecasting with the Lee-Carter Method: Adjusting for Smoothing and Lifespan Disparity. *European Journal of Population, 37*(1), 97-120. https://doi.org/10.1007/s10680-020-09559-9

Richman, R. & Wüthrich, M. V. (2018). A Neural Network Extension of the Lee-Carter Model to Multiple Populations. http://dx.doi.org/10.2139/ssrn.3270877

Rodrigues, C. G., Andrade, M. V., Queiroz, B. L. & Machado, C. J. (2013). *The Applicability of the Lee-Carter Method to Forecast Health Services Use in Brazil.* In: Hoque, N., McGehee, M., Bradshaw, B. (eds) Applied Demography and Public Health. Applied Demography Series, vol 3. Springer, Dordrecht. https://doi.org/10.1007/978-94-007-6140-7_21

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature 323*, 533-536. https://doi.org/10.1038/323533a0

Silva, R., Steen, E. & Darley, O. (2019). *Time Series Forecasting with Deep Learning Models.*

Škrlj, B., Kralj, J., Pollak, S. & Lavrač, N. (2019). Towards Robust Text Classification with Semantics-Aware Recurrent Neural Architecture. *Machine Learning and Knowledge Extraction, 1*(2), 575-589. https://doi.org/10.3390/make1020034

Steeghs, K. (2020). Parameter Uncertainty in the Lee-Carter Model. *Network for Studies on Pensions, Aging, and Retirement* (Master's thesis, Maastricht University, Amsterdam, The Netherlands). Retrieved from https://www.netspar.nl/assets/uploads/P20200814_MSc014_Steeghs.pdf

Szandala, T. (2021). Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. *arXix*. https://doi.org/10.48550/arXiv.2010.09458

Tate, R. B., MacWilliam, L. R. & Finlayson, G. (2005). A Methodology for Evaluating Hospital Bed Need in Manitoba in 2020. *Canadian Journal on Aging/La Revue canadienne du vieillissement, 24*(5), 141-151. https://doi.org/10.1353/cja.2005.0056

Tuljapurkar, S., Li, N. & Boe, C. (2000). A universal pattern of mortality decline in the G7 countries. *Nature 405*, 789-792. https://doi.org/10.1038/35015561

United Nations. (1952). *Methods of Estimating Total Population for Current Dates.*

Uzair, M. & Jamil, N. (2020). Effects of Hidden Layers on the Efficiency of Neural networks. *2020 IEEE 23rd International Multitopic Conference (INMIC)*, 1-6. https://doi.org/10.1109/INMIC50486.2020.9318195

Wang, J., You, K., Long, M. & Jordan, M. I. (2019). How Does Learning Rate Decay Help Modern Neural Networks? *arXiv*. https://doi.org/10.48550/arXiv.1908.01878

Wiener, N. (1948). *Cybernetics: Or Control and Communication in the Animal and the Machine,* (2nd ed.). Cambridge: Massachusetts Institute of Technology Press.

Wilmoth. (1993). *Computational methods for fitting and extrapolating the Lee-Carter model of mortality change.* Technical Report. University of California, Berkeley.

Wilmoth. (1996). Mortality projections for Japan. In: *Health and mortality among elderly populations* (pp. 266-288). Clarendon Press.

Xu, K., Saksena, P. & Holly, A. (2011). The determinants of health expenditure. A country-level panel data analysis. *(WHO Working Paper R4D). Geneva: World Health Organization*.

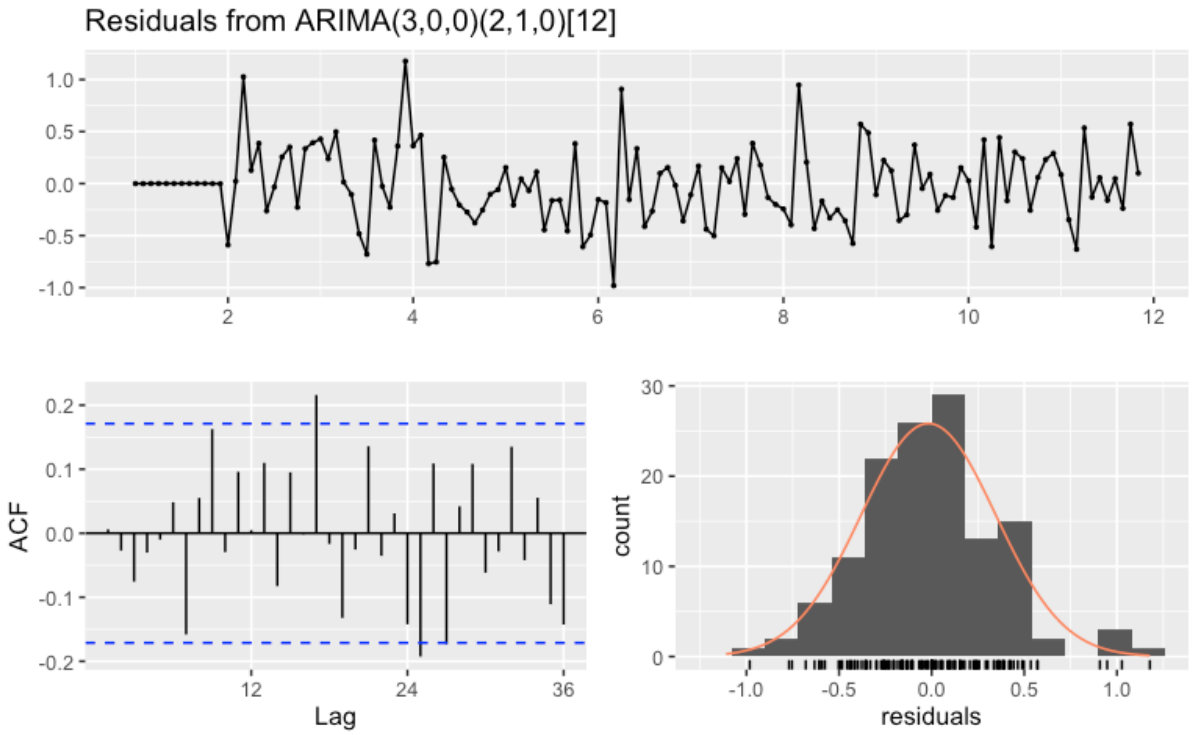
Appendix I - Time Series Outputs

Residuals for Females - ARIMA(3, 0, 0)(2, 1, 0)[12]

```
        Ljung-Box test

data:  Residuals from ARIMA(3,0,0)(2,1,0)[12]
Q* = 31.292, df = 19, p-value = 0.03749

Model df: 5.   Total lags used: 24
```

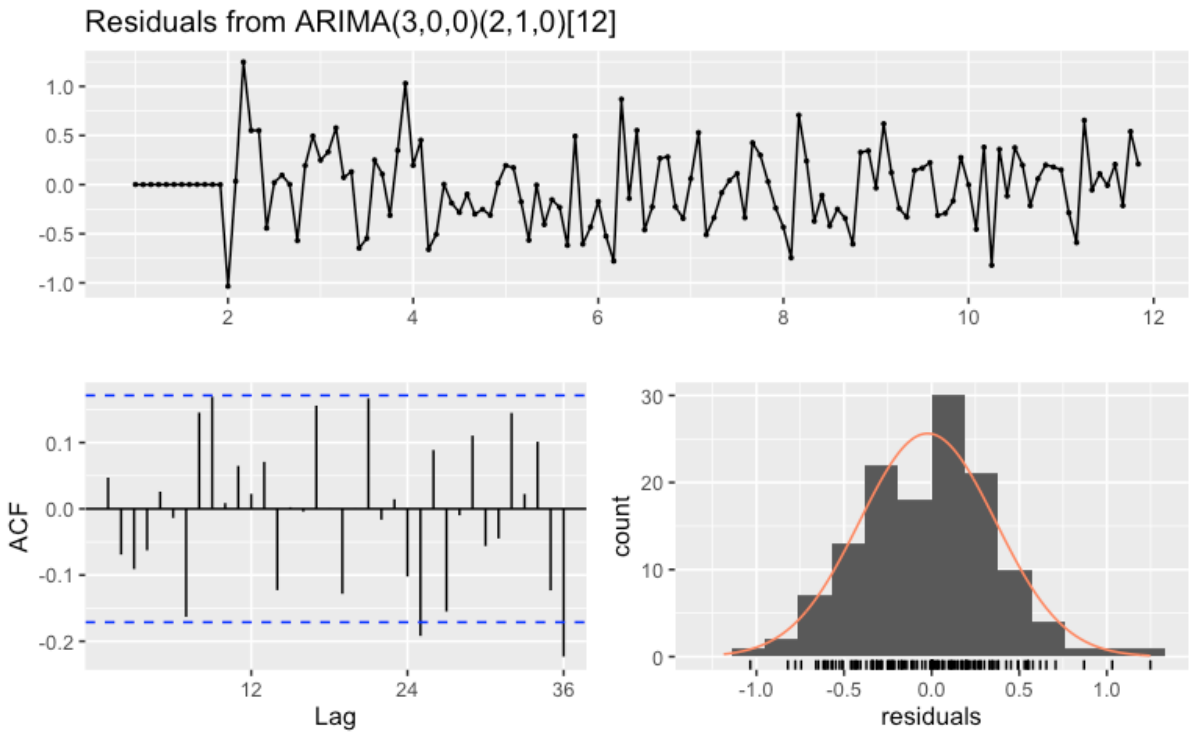


Residuals for Males - ARIMA(3, 0, 0)(2, 1, 0)[12]

```
        Ljung-Box test

data:  Residuals from ARIMA(3,0,0)(2,1,0)[12]
Q* = 29.717, df = 19, p-value = 0.05551

Model df: 5.   Total lags used: 24
```


Residuals from ARIMA(3,0,0)(2,1,0)[12]
1.0
0.5
0.0
-0.5
-1.0
2
4
6
8
10
12
ACF
0.1
0.0
-0.1
-0.2
12
24
36
Lag
count
30
20
10
0
-1.0
-0.5
0.0
0.5
1.0
residuals

Appendix II - LSTM Outputs

Model summary - Females and Males

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM (LSTM)                 (1, 12, 12)               672
 Output (TimeDistributed)    (1, 12, 1)                13
=================================================================
Total params: 685
Trainable params: 685
Non-trainable params: 0
_________________________________________________________________
```

Female LSTM - RMSE metrics

Model	Learning Rate	decay	avg_rmse	sd_rmse	min_rmse	q25_rmse	median_rmse	q75_rmse	max_rmse
Model 1	0,001	0	0,4978	0,0107	0,4763	0,489675	0,49785	0,506725	0,519
Model 2	0,001	1,00E-07	0,498	0,0109	0,4762	0,48985	0,4977	0,50845	0,5184
Model 3	0,001	1,00E-05	0,4974	0,0105	0,4769	0,4903	0,4974	0,506775	0,5165
Model 4	0,001	0,001	0,4912	0,0079	0,4789	0,4859	0,4898	0,497125	0,5141
Model 5	0,001	0,1	0,7828	0,0661	0,6774	0,732475	0,76195	0,845375	0,8936
Model 6	0,002	0	0,504	0,0141	0,476	0,495475	0,50475	0,511175	0,5406
Model 7	0,002	1,00E-07	0,504	0,0147	0,4764	0,495025	0,5035	0,511525	0,5406
Model 8	0,002	1,00E-05	0,5049	0,0126	0,482	0,496625	0,5045	0,511275	0,5348
Model 9	0,002	0,001	0,499	0,0114	0,4759	0,491975	0,49765	0,50725	0,5208
Model 10	0,002	0,1	0,7241	0,0515	0,6355	0,68995	0,7158	0,770375	0,8149
Model 11	0,01	0	0,4938	0,0326	0,4286	0,469725	0,4881	0,5106	0,5883
Model 12	0,01	1,00E-07	0,4889	0,0303	0,4443	0,468775	0,48315	0,50165	0,5615
Model 13	0,01	1,00E-05	0,4931	0,0306	0,4296	0,478625	0,4861	0,5148	0,5459
Model 14	0,01	0,001	0,5032	0,0313	0,4365	0,48515	0,50505	0,5178	0,5938
Model 15	0,01	0,1	0,4881	0,0238	0,4534	0,466725	0,49025	0,50035	0,5343
Model 16	0,02	0	0,5062	0,033	0,4468	0,4806	0,5091	0,523025	0,5698
Model 17	0,02	1,00E-07	0,5014	0,0291	0,4549	0,4772	0,49835	0,520075	0,5841
Model 18	0,02	1,00E-05	0,5058	0,0252	0,455	0,488175	0,50685	0,51765	0,5525
Model 19	0,02	0,001	0,4805	0,0268	0,4381	0,4651	0,4836	0,49875	0,5461
Model 20	0,02	0,1	0,4633	0,0076	0,4483	0,459825	0,46365	0,4666	0,4778

Female LSTM - MAE metrics

Model	Learning Rate	decay	avg_mae	sd_mae	min_mae	q25_mae	median_mae	q75_mae	max_mae
Model 1	0,001	0	0,4049	0,0165	0,379	0,387175	0,40655	0,41555	0,4355
Model 2	0,001	1,00E-07	0,4054	0,0163	0,3785	0,39345	0,40575	0,416225	0,4354
Model 3	0,001	1,00E-05	0,405	0,0156	0,3803	0,39005	0,407	0,4157	0,4348
Model 4	0,001	0,001	0,3967	0,0118	0,3735	0,38685	0,39675	0,40565	0,4229
Model 5	0,001	0,1	0,5407	0,0596	0,4425	0,492925	0,52625	0,593	0,6406
Model 6	0,002	0	0,4097	0,0151	0,3797	0,400525	0,4079	0,42205	0,4437
Model 7	0,002	1,00E-07	0,4093	0,0174	0,3652	0,397675	0,4109	0,420075	0,4437
Model 8	0,002	1,00E-05	0,4111	0,0141	0,3802	0,401325	0,4123	0,42065	0,4378
Model 9	0,002	0,001	0,401	0,0141	0,3765	0,39175	0,39845	0,413975	0,4239
Model 10	0,002	0,1	0,4942	0,0423	0,4165	0,4672	0,48575	0,52735	0,5705
Model 11	0,01	0	0,3824	0,0361	0,3046	0,3596	0,37575	0,40485	0,4594
Model 12	0,01	1,00E-07	0,3828	0,0382	0,3339	0,3549	0,3745	0,404425	0,4829
Model 13	0,01	1,00E-05	0,3836	0,0318	0,3394	0,365225	0,3741	0,401525	0,4775
Model 14	0,01	0,001	0,3967	0,0299	0,3346	0,374575	0,3931	0,409525	0,4696
Model 15	0,01	0,1	0,3627	0,0285	0,3139	0,342	0,3648	0,3766	0,4262
Model 16	0,02	0	0,3937	0,0359	0,3114	0,370075	0,39035	0,41935	0,459
Model 17	0,02	1,00E-07	0,3915	0,0333	0,3269	0,36835	0,39445	0,412275	0,4718
Model 18	0,02	1,00E-05	0,3968	0,033	0,3304	0,3762	0,39895	0,4135	0,4733
Model 19	0,02	0,001	0,3813	0,0351	0,3046	0,3698	0,3828	0,409475	0,4643
Model 20	0,02	0,1	0,3582	0,0102	0,3351	0,350725	0,3576	0,365175	0,3785

Male LSTM - RMSE metrics

Model	Learning Rate	decay	avg_rmse	sd_rmse	min_rmse	q25_rmse	median_rmse	q75_rmse	max_rmse
Model 1	0,001	0	0,5643	0,0162	0,5285	0,54935	0,5682	0,577725	0,5873
Model 2	0,001	1,00E-07	0,5641	0,0155	0,5357	0,5493	0,56655	0,57765	0,5867
Model 3	0,001	1,00E-05	0,5635	0,0165	0,5305	0,549325	0,56285	0,577975	0,5908
Model 4	0,001	0,001	0,5878	0,024	0,5376	0,571475	0,58625	0,609325	0,6273
Model 5	0,001	0,1	0,7715	0,068	0,6747	0,717675	0,74615	0,826325	0,9021
Model 6	0,002	0	0,5876	0,026	0,5416	0,570275	0,58885	0,599975	0,6482
Model 7	0,002	1,00E-07	0,5895	0,0265	0,5433	0,5698	0,58685	0,60265	0,6483
Model 8	0,002	1,00E-05	0,5926	0,0324	0,5387	0,568475	0,58875	0,610125	0,6713
Model 9	0,002	0,001	0,5719	0,0187	0,5398	0,5607	0,573	0,580925	0,6123
Model 10	0,002	0,1	0,7311	0,0268	0,6868	0,70915	0,7346	0,74895	0,7969
Model 11	0,01	0	0,5897	0,0368	0,532	0,5673	0,5816	0,603	0,6823
Model 12	0,01	1,00E-07	0,5899	0,042	0,517	0,565425	0,58575	0,60205	0,6987
Model 13	0,01	1,00E-05	0,5877	0,043	0,4873	0,5575	0,58365	0,623475	0,6689
Model 14	0,01	0,001	0,6075	0,0302	0,5542	0,58465	0,6065	0,627275	0,6945
Model 15	0,01	0,1	0,7165	0,0339	0,6301	0,6879	0,7207	0,743525	0,7759
Model 16	0,02	0	0,5987	0,0774	0,4884	0,550875	0,59505	0,626975	0,8794
Model 17	0,02	1,00E-07	0,616	0,081	0,4951	0,5749	0,60305	0,632825	0,9174
Model 18	0,02	1,00E-05	0,6056	0,0566	0,4518	0,57455	0,6051	0,63895	0,7048
Model 19	0,02	0,001	0,6125	0,0502	0,5284	0,579825	0,60875	0,641975	0,7262
Model 20	0,02	0,1	0,5993	0,0182	0,5702	0,58855	0,599	0,60615	0,6576

Male LSTM - MAE metrics

Model	Learning Rate	decay	avg_mae	sd_mae	min_mae	q25_mae	median_mae	q75_mae	max_mae
Model 1	0,001	0	0,4768	0,0194	0,4399	0,4604	0,4818	0,492625	0,5051
Model 2	0,001	1,00E-07	0,4769	0,0183	0,4399	0,46375	0,48095	0,489975	0,505
Model 3	0,001	1,00E-05	0,476	0,0191	0,4449	0,4606	0,4789	0,4913	0,5112
Model 4	0,001	0,001	0,4976	0,0275	0,4396	0,479825	0,4917	0,520975	0,5406
Model 5	0,001	0,1	0,5836	0,0461	0,5049	0,5521	0,5748	0,6186	0,6716
Model 6	0,002	0	0,5014	0,0256	0,4504	0,485425	0,50105	0,517725	0,5574
Model 7	0,002	1,00E-07	0,5028	0,0247	0,4593	0,483275	0,50045	0,520425	0,5576
Model 8	0,002	1,00E-05	0,5062	0,0312	0,4485	0,481825	0,50905	0,5276	0,5674
Model 9	0,002	0,001	0,4849	0,0222	0,4428	0,46945	0,4866	0,497275	0,5322
Model 10	0,002	0,1	0,5904	0,0327	0,5296	0,570925	0,5905	0,61305	0,6616
Model 11	0,01	0	0,4971	0,0403	0,4258	0,47655	0,4917	0,512	0,591
Model 12	0,01	1,00E-07	0,502	0,0396	0,4209	0,477675	0,50455	0,522875	0,5932
Model 13	0,01	1,00E-05	0,4963	0,0534	0,3824	0,465875	0,4992	0,5468	0,5794
Model 14	0,01	0,001	0,5158	0,0318	0,4656	0,4936	0,51355	0,537675	0,598
Model 15	0,01	0,1	0,6172	0,0357	0,54	0,58625	0,6182	0,6479	0,6775
Model 16	0,02	0	0,5044	0,0744	0,382	0,463425	0,4994	0,539575	0,791
Model 17	0,02	1,00E-07	0,5197	0,0817	0,3966	0,4806	0,51295	0,546075	0,8468
Model 18	0,02	1,00E-05	0,5106	0,0565	0,3752	0,4745	0,51595	0,550625	0,6077
Model 19	0,02	0,001	0,5236	0,0514	0,4215	0,488925	0,5193	0,5606	0,6513
Model 20	0,02	0,1	0,5059	0,0241	0,462	0,49285	0,5056	0,51935	0,579