Strategy for Serving the Population in the Fight against Covid-19

Scenario Simulator with Artificial Intelligence and System Dynamics

Guttenberg Ferreira Passos

This article aims to contribute to a population service strategy in the fight against Covid-19, through a scenario simulator with Artificial Intelligence and System Dynamics.

In February 2021, relevant changes were announced in Brazil in the fight against Covid-19, especially regarding vaccination. This article addresses a series of predictions to assist in the strategy of serving the population in the fight against this disease. It is a good opportunity to better understand the "invisible" factors that affect the people's contamination.

To help model these "invisible" factors, social distancing data was obtained, this data was passed as input to an Artificial Intelligence system and predictions were performed in Deep Learning (LSTM). The prediction of the social distancing variable was exported to calibrate the model in System Dynamics, with the Stella software, and the simulations were made with a Covid-19 model. As future work, the result of these simulations could be plotted on a map of a cell phone through geoprocessing.

All models were adapted from the Predictive Time Series Modeling course classes that can be found on the Data Science Academy Community timeline on the portal: www.datascienceacademy.com.br.

Time Series

A time series is a sequential set of data points, typically measured at successive times. It is mathematically defined as a set of vectors x(t), t = 0,1,2, ... where t represents elapsed time. The variable x(t) is treated as a random variable.

A random variable is a quantitative variable whose result (value) depends on random factors.

Measurements taken during an event in a time series are arranged in a proper chronological order. A time series containing records of a single variable is called univariate and more than one variable is called multivariate.

Time series add an explicit order dependency between observations: a time dimension. This additional dimension is a constraint and structure that provides a source of additional information. And very, very valuable.

Time series analysis involves developing models that best capture or describe an observed time series to understand the causes. This field of study looks for the "why" behind a time series dataset.

The quality of a descriptive model is determined by how well it describes all available data and the interpretation it provides to better inform the problem domain.

The main objective of time series analysis is to develop mathematical models that provide plausible descriptions from sample data.

Making predictions about the future is called extrapolation in the classical statistical treatment of time series data. More modern fields focus on the topic and refer to it as time series forecasting. Forecasting involves fitting models to historical data and using it to predict future observations.

An important distinction in forecasting is that the future is completely unavailable and should only be estimated from what has already happened. The purpose of time series analysis is usually twofold: to understand or model the stochastic mechanisms that give rise to an observed series and to predict the future values of a series based on the history of that series. This is what we call Predictive Modeling.

Nonlinear predictive modeling with time series.

- Statistical Methods, Artificial Intelligence, Machine Learning and Deep Learning are used for non-linear predictive models, such as the following algorithms:
  - Naive;
  - Exponential Smoothing;
  - ARMA - Autoregressive Moving Average;
  - ARIMA - Autoregressive Integrated Moving Average;
  - SARIMAX – Seasonal Autoregressive Integrated Moving Average;
  - DeepAR – GluonTS - Probabilistic Time Series Modeling;
  - MLP - Multilayer Perceptron;
  - LSTM - Long short-term memory.

Naive Algorithm

It is the simplest predictive model you can create. It is based on an estimation technique in which actual data from the last period is used as a forecast for that period, without adjusting it or trying to establish causal factors. It is only used for comparison with forecasts generated by the best (sophisticated) techniques.

Naive means simple, so there is no advanced technique here it is just used as a starting point, a reference for the other models. Any more advanced model must present superior results to the Naive Algorithm.

Exponential Smoothing Algorithm

Smoothing on time series is a set of methods for smoothing time series by eliminating "jumps". There are several ways to do this. Perhaps the easiest is to calculate the Simple Moving Average.

Smoothing is basically a technique used to see the long-term trend in the data, lessening the effects of the periodic/seasonal components of the data. Basically, smoothing is used when you want to remove fluctuations in the data and focus only on preserving long-term trends.

The purpose of smoothing is to remove noise and better expose the process signal. Moving averages are a simple and common type of smoothing used in time series analysis and forecasting. Calculating a moving average involves creating a new series where the values are composed of the average of raw observations in the original time series.

Moving Average Smoothing is an effective technique in time series forecasting as well, meaning it can be used for data preparation, feature engineering, and even directly to make predictions. A moving average requires you to specify a window size called the window width. This sets the number of raw observations used to calculate the moving average value.

The "moving" part in the moving average refers to the fact that the window defined by the width of the window is slid along the time series to calculate the average values in the new series.

ARMA Algorithm

Auto Regressive Moving Average - ARMA is an autoregressive moving average model. It is simply the fusion between the AR (p) and MA (q) models. The AR(p) model attempts to explain the timing and average reversal effects frequently observed in markets (market participant effects). The MA(q) model attempts to capture the observed shock effects in terms of white noise. These shock effects can be considered unexpected events that affect the observation process, p, such as sudden gains, wars, attacks, etc.

The ARMA model tries to capture these two aspects when modeling time series, it does not take into account volatility clustering, an empirical phenomenon essential to many financial time series.

O Modelo ARMA(1,1) é representado como: x(t) = ax(t-1) + be(t-1) + e(t), onde e(t) é o ruído branco com E [e(t)] = 0.

The ARMA(1,1) Model is represented as: x(t) = ax(t-1) + be(t-1) + e(t), where e(t) is white noise with

E [e(t)] = 0.

An ARMA model generally requires fewer parameters than an AR(p) model or an individual MA(q) model.

ARIMA Algorithm

Autoregressive Integrated Moving Average - ARIMA is an autoregressive integrated moving average model, a generalization of an ARMA model.

Both models are fitted to time series data to better understand the data or to predict future points in the series (forecast). ARIMA models are applied in some cases where the data show evidence of non-stationarity, where an initial differentiation step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate non-stationarity.

The AR part of ARIMA indicates that the evolving variable of interest is regressed with its own lagged (ie, prior) values. The I (for "integrated") indicates that the data values have been replaced by the difference between their values and the previous values (and this differentiating process may have been performed more than once). The goal of each of these features is to make the model fit the data as well as possible. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past.

Non-seasonal ARIMA models are usually designated ARIMA(p, d, q), where the parameters p, d and q are non-negative integers, p is the order (number of time intervals) of the autoregressive model, d is the degree of differentiation (the number of times the data had past values subtracted) and q is the order of the moving average model.

SARIMAX Algorithm

Seasonal Autoregressive Integrated Moving Average - SARIMAX is a seasonal autoregressive integrated moving average model with the parameters (p, d, q) (P, D, Q) m, where m refers to the number of periods in each seasonality and the capital letters P, D, Q refer to the autoregressive, differentiated, and moving average terms of the seasonal part of the ARIMA model.

DeepAR - GluonTS Algorithm

DeepAR was released by Amazon, being integrated with SageMaker. Your goal is to predict the next step at each step (horizon = 1). This means that the network must receive on input the previous observation (at delay = 1) z_t-1, along with a set of optional covariates x_i. The information is propagated to the hidden layer and up to the likelihood function (which is a scoring function used at the level of a loss function). The probability function can be either Gaussian or negative binomial. The error is calculated using the current probability parameterization. This is easily represented by mu and sigma in the case of a Gaussian probability. This means that while running the backpropagation we are adjusting the network parameters (w weights) which change the parameterization of all the examples.

DeepAR can be implemented with Gluon Time Series - GluonTS, a library for time series modeling based on Deep Learning, simplifies the development and experimentation of time series models for common tasks such as prediction or anomaly detection. It provides all the necessary components and tools Data Scientists need to quickly create new models, efficiently run and analyze experiments, and assess model accuracy.

MLP Algorithm

A standard Multilayer Perceptron network - MLP is a recurrent neural network - RNN designed for sequential problems, it can be thought of as adding loops to the architecture. For example, in a given layer, each neuron can pass its signal forward (feed-forward) and also to the side.

A Recurrent Neural Network is basically a neural network that can be used when your data is treated as a sequence, where the particular order of the data points is important and this sequence can be of arbitrary length.

The clearest example is perhaps a time series of numbers, where the task is to predict the next value according to previous values. The input to the RNN at each time step is the current value, as well as a state vector representing what the network "saw" in time - previous steps. This state-vector is the encoded memory of the RNN, initially set to zero.

LSTM Algorithm

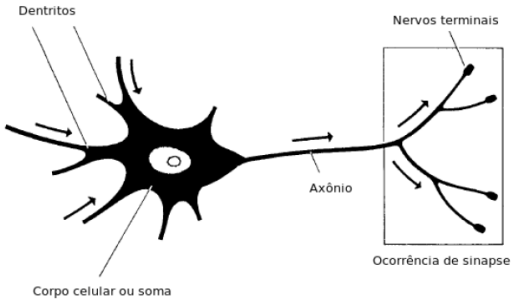
The Long Short Term Memory - LSTM network is a recurrent neural network trained using Backpropagation Through Time and overcomes the problem of gradient dissipation. As such, the model can be used to create large recurrent networks which, in turn, can be used to solve difficult sequence problems in machine learning and obtain state-of-the-art results.

Instead of neurons, LSTM networks have memory blocks connected through layers. A block has components that make it smarter than a classical neuron and a memory for recent sequences. A block contains ports that manage the block's state and output. A block operates on an input sequence and each gate within a block uses the sigmoid activation units to control whether they are triggered or not, conditioning the change of state and the addition of information that flows through the block.

Biological Neuron

The Deep Learning Book Brazil is a Data Science Academy initiative with the objective of helping to spread Deep Learning, one of the most revolutionary technologies of our time used in the construction of Artificial Intelligence applications.

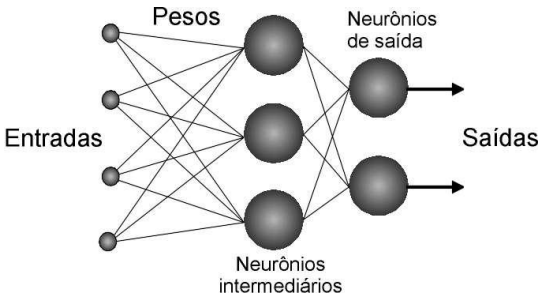
According to the Deep Learning Book, the biological neuron is a cell, which can be divided into three sections: the cell body, the dendrites and the axon, each with specific but complementary functions.



Source: Data Science Academy- DSA

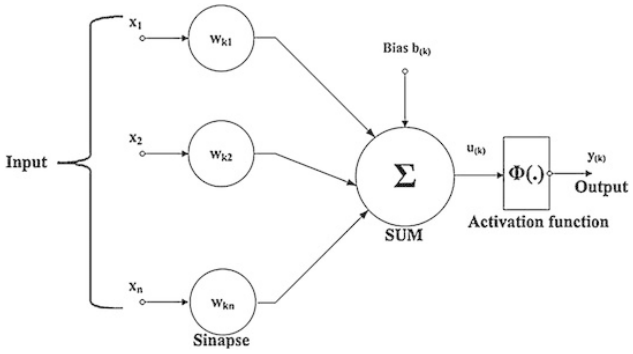
Artificial Neuron

An artificial neuron represents the basis of an Artificial Neural Network (ANN), a model of neuroinformatics oriented to biological neural networks.



Source: Data Science Academy - DSA

The knowledge of an ANN is encoded in the structure of the network, where the connections (synapses) between the units (neurons) that compose it are highlighted.



Source: Data Science Academy - DSA

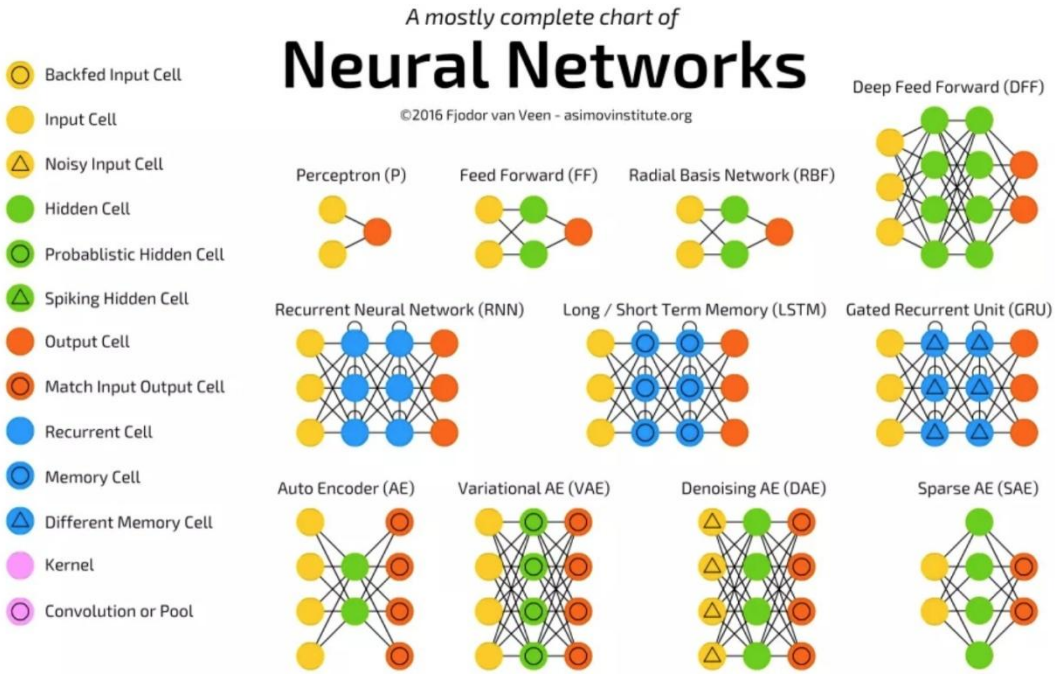
In machine learning, Perceptron is a supervised learning algorithm of binary classifiers. A binary classifier is a function that can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, that is, a classification algorithm that makes its predictions based on a linear predictor function by combining a set of weights with the vector of features.

Redes Neurais

Neural Networks

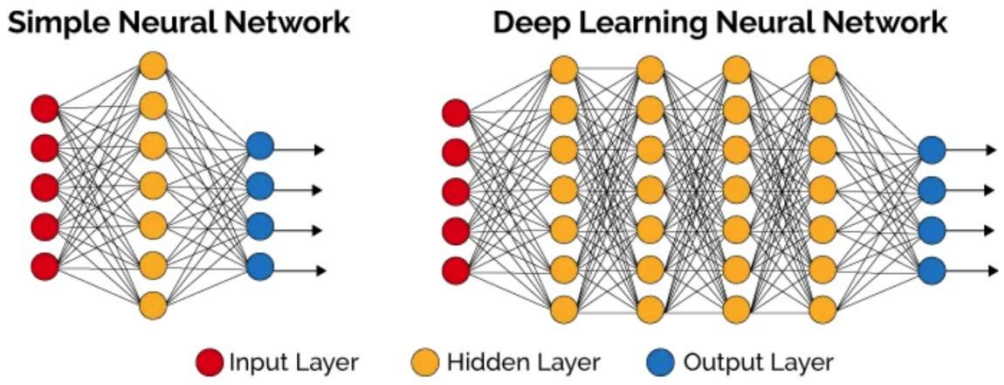
Neural networks are computing systems with interconnected nodes that function like neurons in the human brain. Using algorithms, they can recognize hidden patterns and correlations in raw data, group and classify them, and over time continually learn and improve.

Asimov Institute https://www.asimovinstitute.org/neural-network-zoo/ has published a cheat sheet containing various neural network architectures, we will concentrate on the architectures below focusing on Perceptron(P), Feed Forward (FF), Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM):



Source: THE ASIMOV INSTITUTE

Deep learning is one of the foundations of Artificial Intelligence, a type of Machine Learning that trains computers to perform tasks like humans, which includes speech recognition, image identification and predictions, learning over time. We can say that it is a Neural Network with several hidden layers:



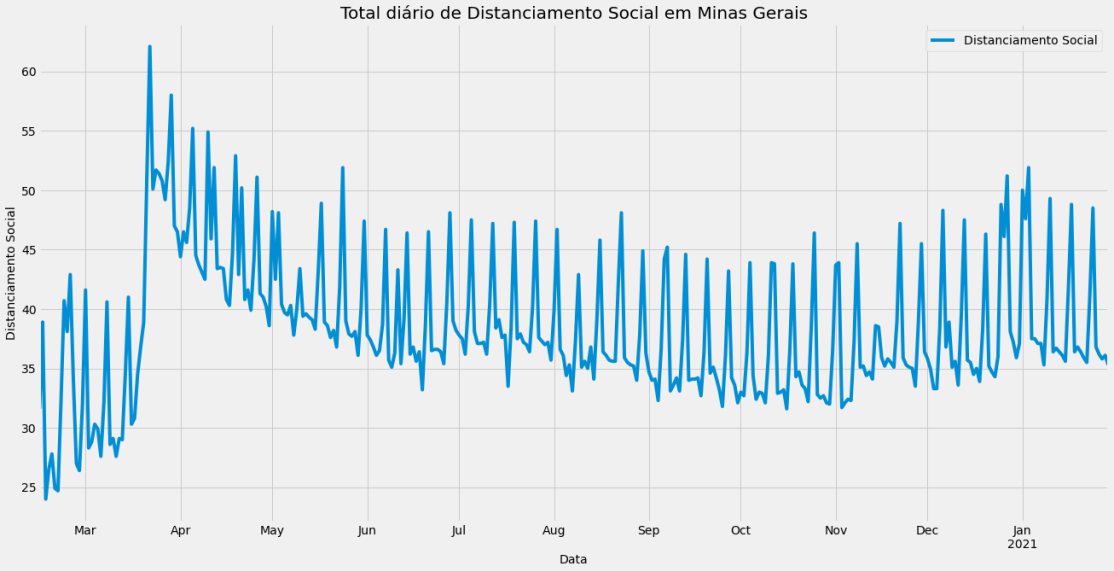
Base Model

Business Problem Definition

Forecast of social distancing in Minas Gerais, using Artificial Intelligence, in order to calibrate a System Dynamics model. It contributes to a population service strategy in the fight against Covid-19, through a scenario simulator.

Data set

We used datasets that show social distancing in Minas Gerais. The data have records from 02/15/2020 to 01/29/2021.



Exploratory Data Analysis

Time Series Decomposition

Statistical models can be used to perform a decomposition of this time series. Time series decomposition is a statistical task that deconstructs a time series into several components, each representing one of the pattern categories. With statistics models, we will be able to see the trend, seasonal and residual components of our data. A classic decomposition of a time series can be performed, considering the series as an additive or multiplicative combination of the base level, trend, seasonal and residual index.

Below is the decomposition of the time series:

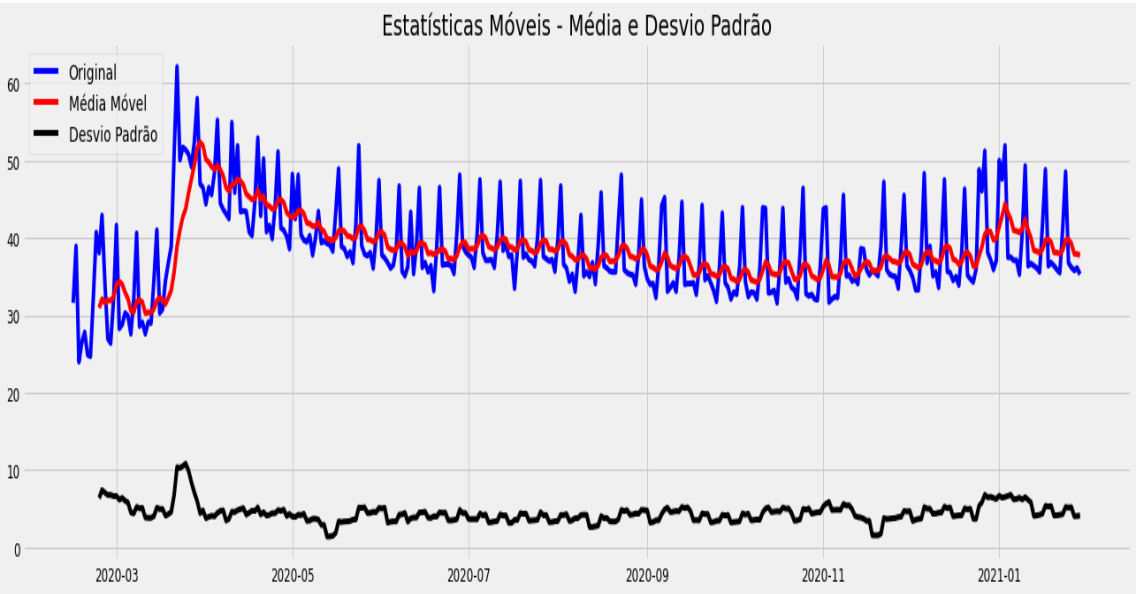
The graph above shows 3 components of the series: Seasonality, Trend and Residuals.

- Seasonality - the phenomenon is repeated at fixed periods;
- Trend - over time, the series follows an upward trend;
- Residuals are the difference between the value of the time series used and the value modeled for the same instant.

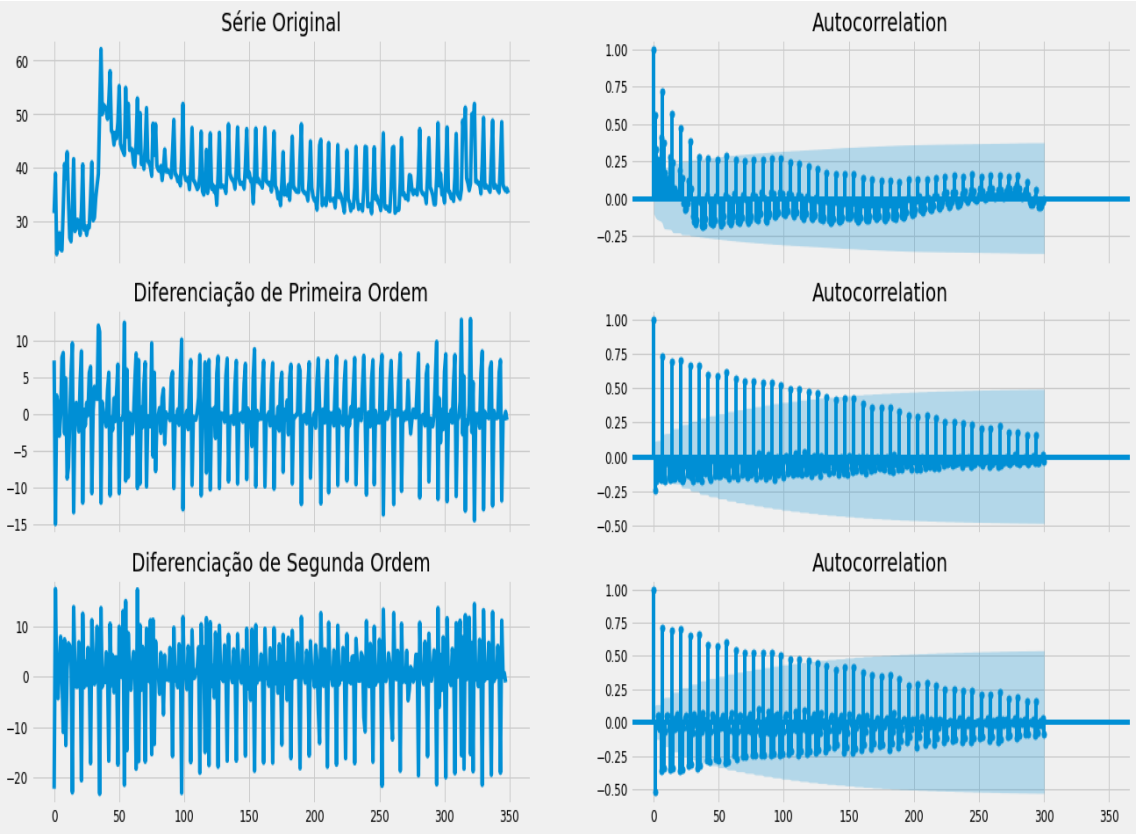
Series stationarity.

Stationarity is an important concept in time series modeling and is characterized by a variable that behaves randomly over time around a constant mean. A time series is considered stationary if its statistical properties, such as mean and variance, remain constant over time. Intuitively, we can say that if a time series has a specific behavior over time, there is a very high probability that it will follow the same behavior in the future.

Basically, time series that have a trend and/or seasonality are not stationary and it is necessary to use appropriate techniques for such a situation.



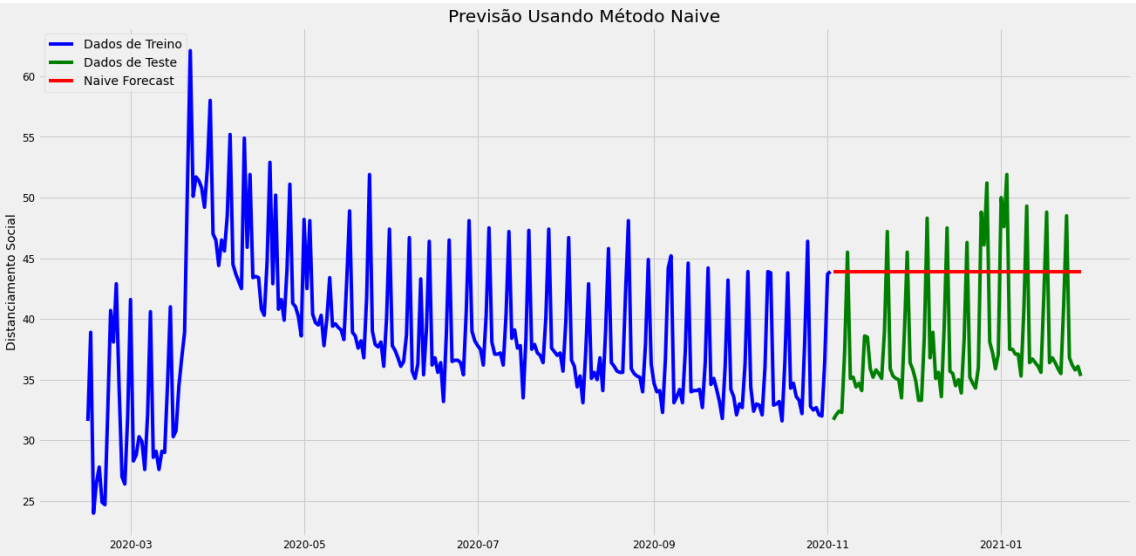
The ACF graph allows the evaluation of the minimum differentiation necessary to obtain a stationary series (Parameter d for the ARIMA Model):



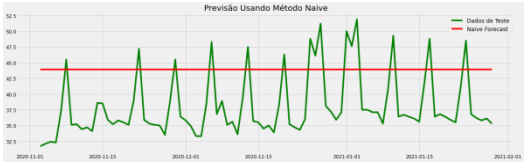
The data were separated into two sets for the execution of the models:

- 262 training records and;
- 88 validation records.

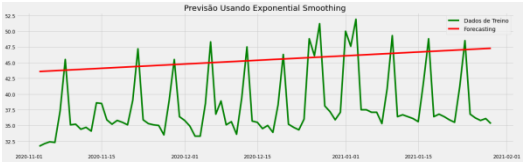
Model 11 - Naive Method Forecasts



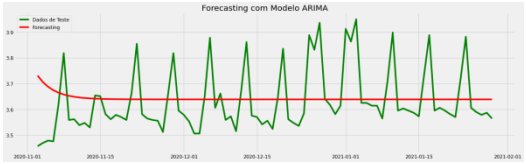
11 – Naive Method



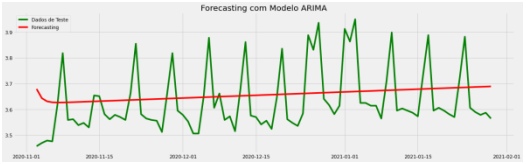
12 – Exponential Smoothing



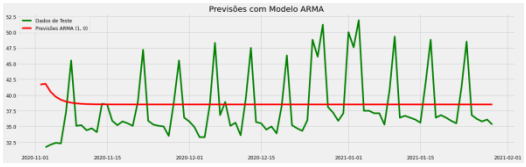
13 - Forecasting–ARIMALOG (1, 0, 1)



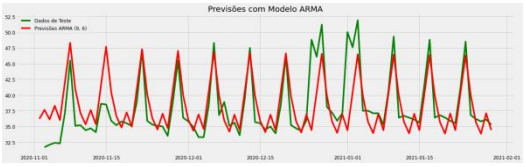
14 - Forecasting– ARIMALOG (1, 1, 1)



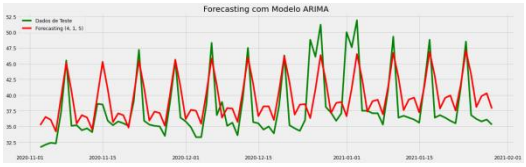
15 –ARMA (1, 0)



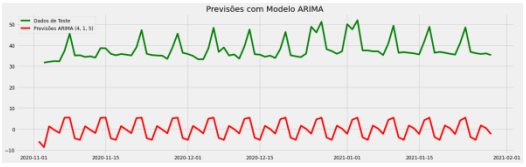
15 – ARMA (9, 6)



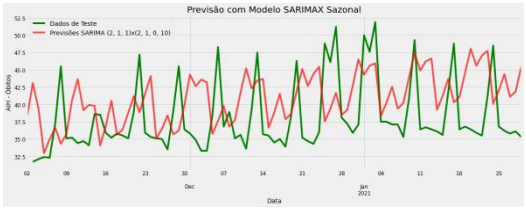
16 – Forecasting - ARIMA (4, 1, 5)



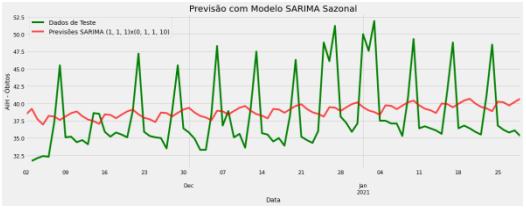
16 - Predict-ARIMA (4, 1, 5)



17 – SARIMAX(2,1,1)(2,1,0,10)



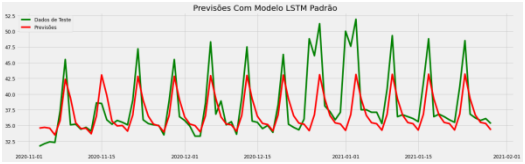
18 – SARIMAX(1,1,1)(0,1,1,10)



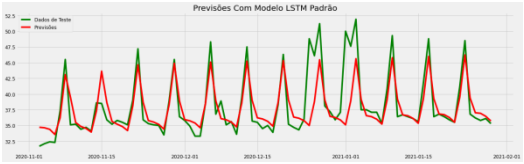
Models using:

Artificial Intelligence - AI

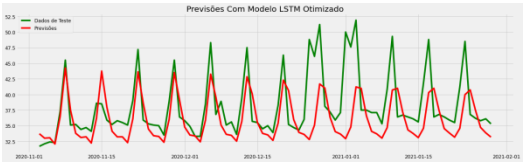
22 – LSTM - IA (3 repetitions)



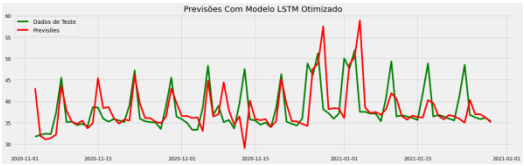
22 – LSTM - IA (5 repetitions)



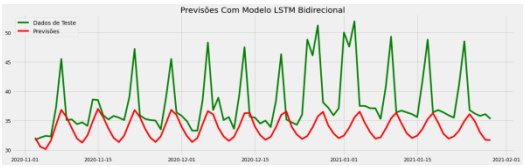
23 – Optimized LSTM - IA



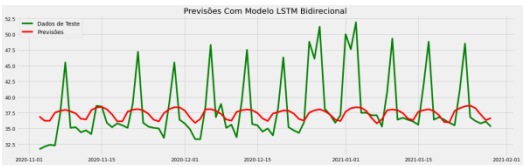
24 – Stacked LSTM - IA



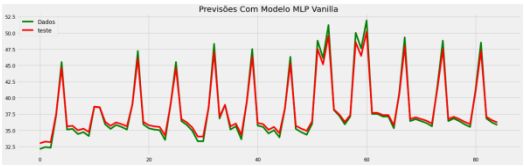
25 – LSTM Bidirectional - IA



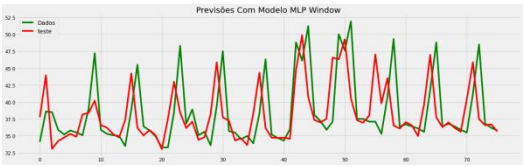
25 – LSTM - IA - Grid Search



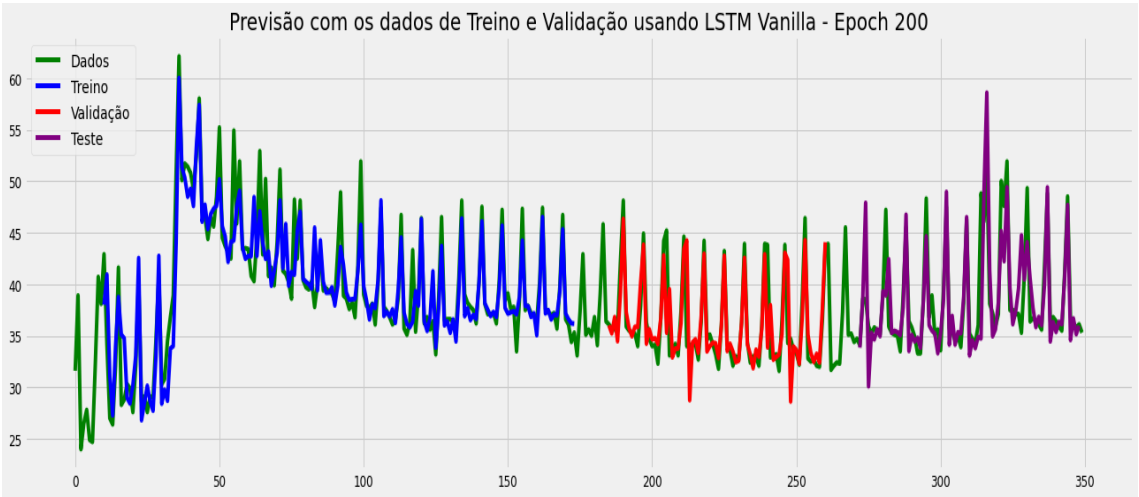
RNN01 – MLP Vanilla - IA



RNN01 – MLP Window - IA

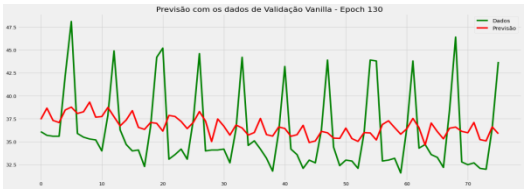


Forecasting using Artificial Intelligence - AI - Deep Learning



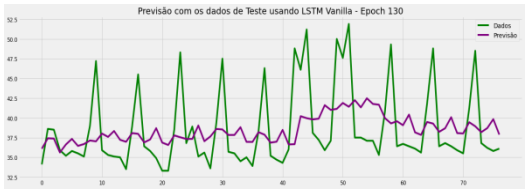
RNN02 – LSTM Vanilla IA – Validation

Epoch = 130 - avoid Overfitting



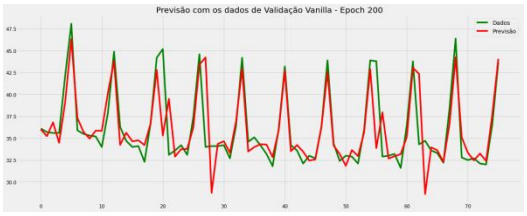
RNN02 – LSTM Vanilla IA – Test

Epoch = 130 - avoid Overfitting



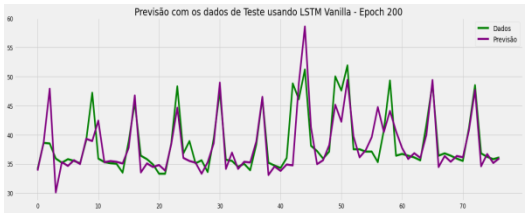
RNN02 – LSTM Vanilla IA – Validation

Epoch = 200



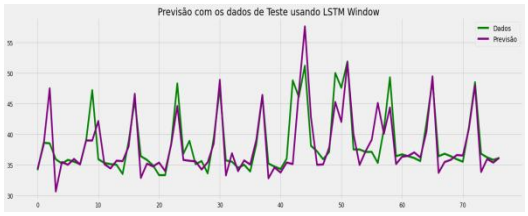
RNN02 – LSTM Vanilla IA – Test

Epoch = 200



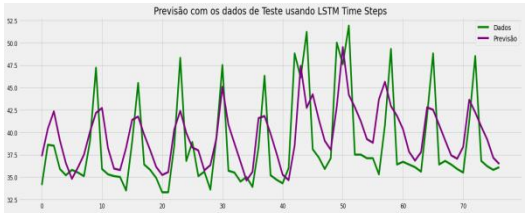
RNN02 – LSTM Window IA– Validation

RNN02 – LSTM Window IA – Test

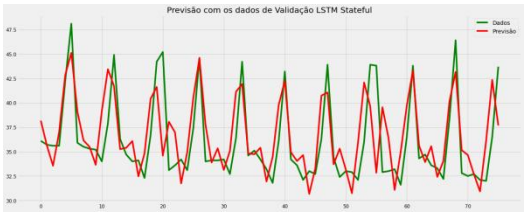


RNN02–LSTM Time Steps IA Validation

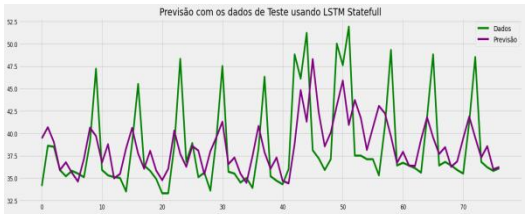
RNN02 – LSTM TimeSteps IA – Test



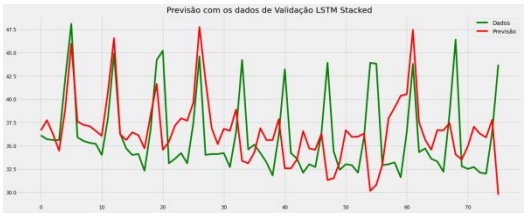
RNN02 – LSTM Stateful IA – Validation



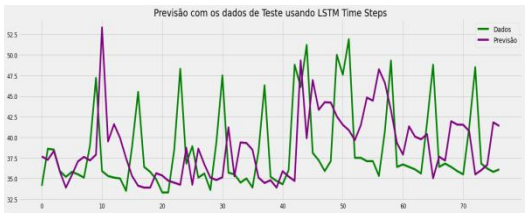
RNN02 – LSTM Stateful IA – Test



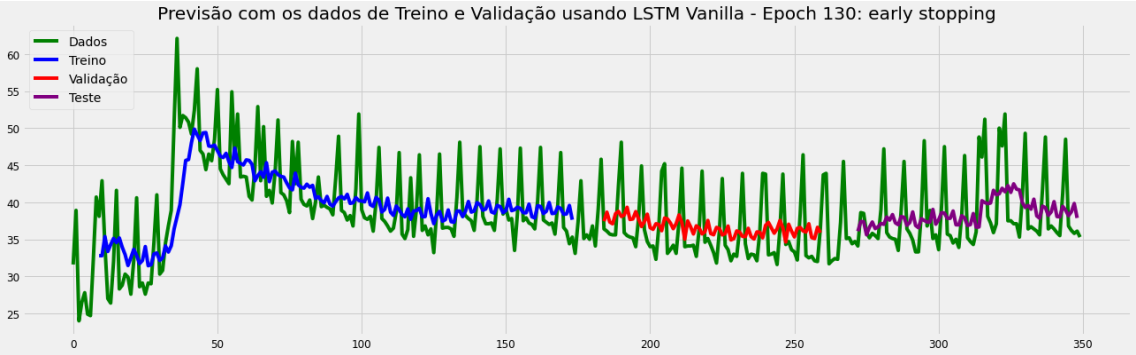
RNN02 – LSTM Stacked IA – Validation



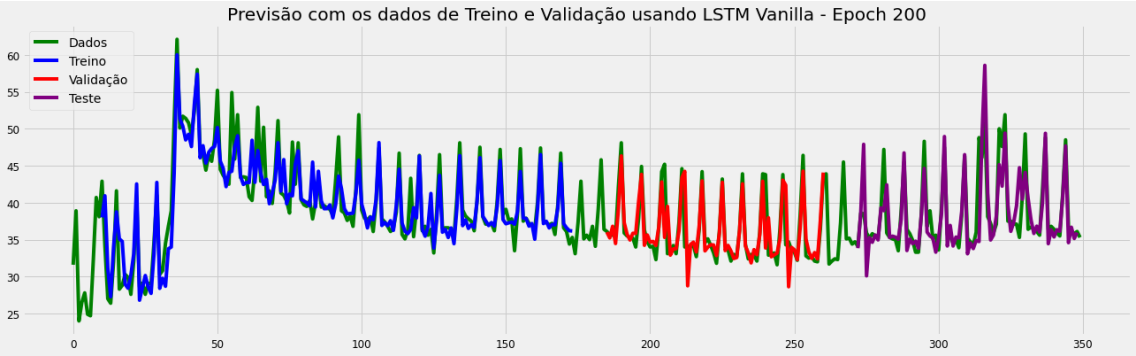
RNN02 – LSTM Stacked IA – Test



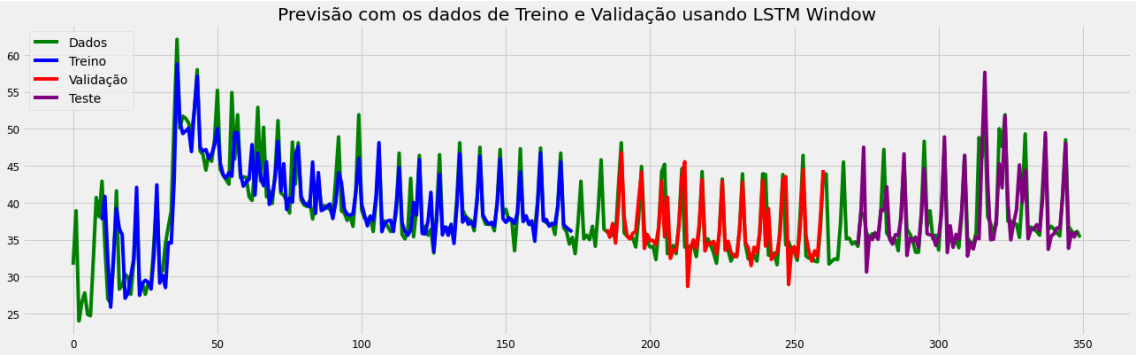
Model RNN02 – LSTM Vanilla IA – Epoch = 130 – avoid Overfitting



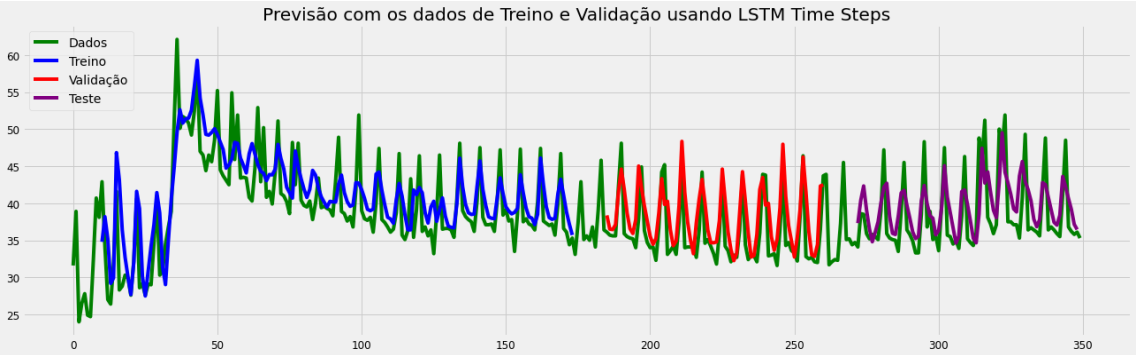
Model RNN02 – LSTM Vanilla IA – Epoch = 200



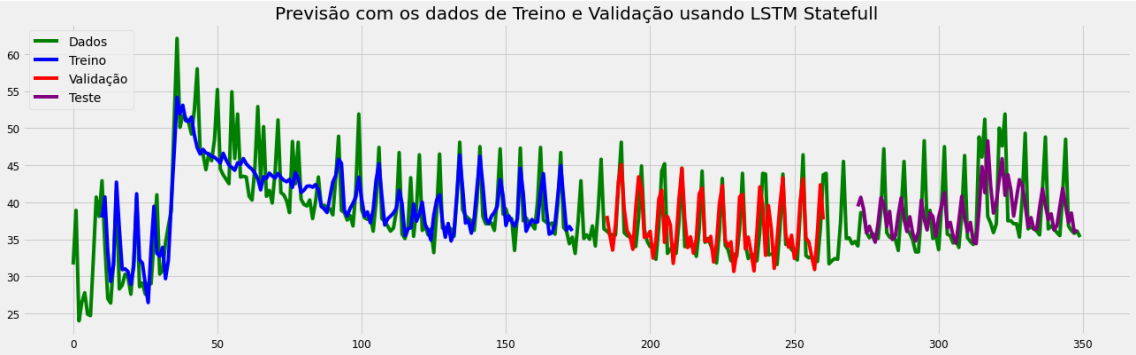
Model RNN02 – LSTM Window IA



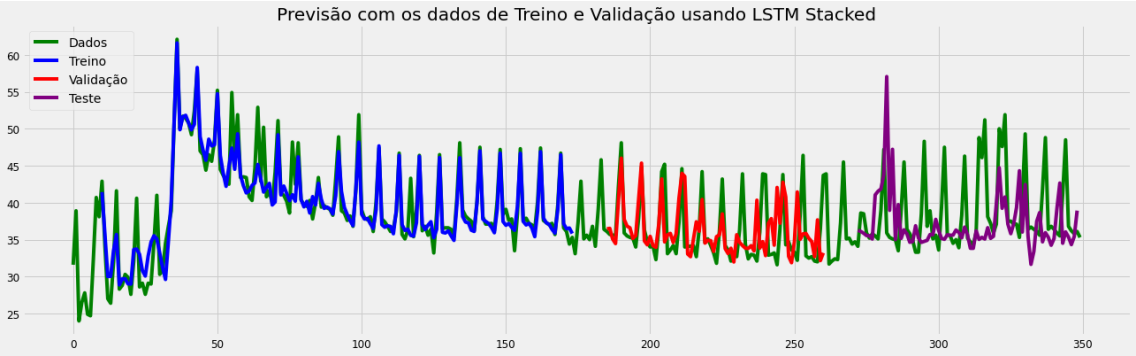
Model RNN02– LSTM Time Steps



Model RNN02– LSTM Stateful

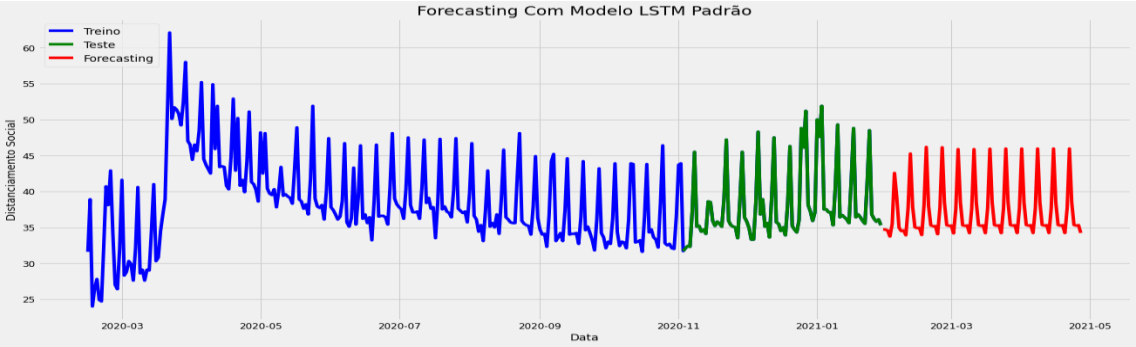


Model RNN02– LSTM Stacked

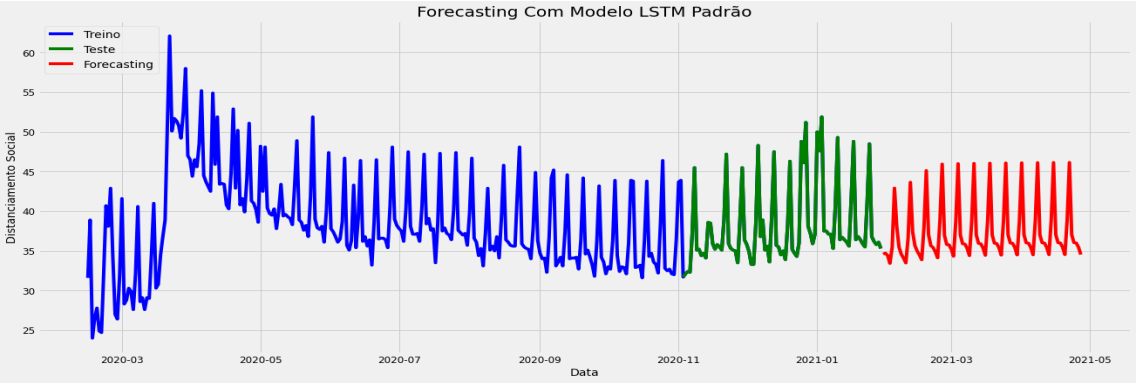


Forecasting using Artificial Intelligence - AI

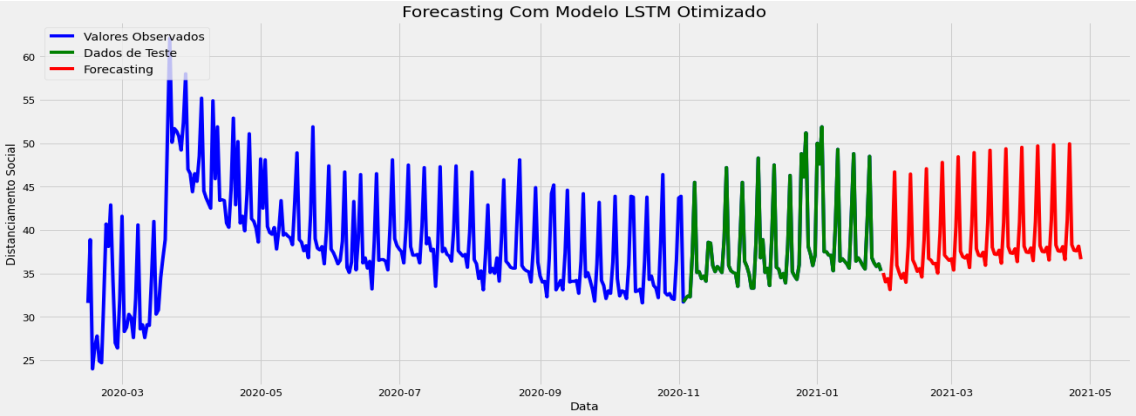
Model 22 – LSTM - IA (3 repetitions)



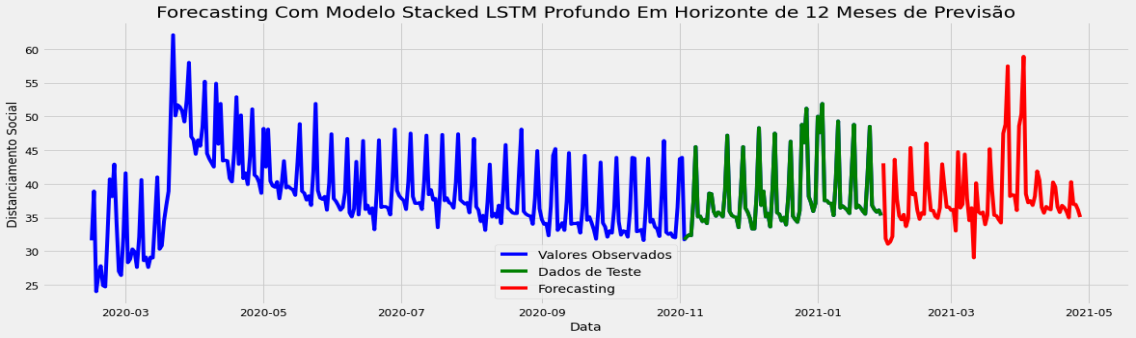
Model 22 – LSTM - IA (5 repetitions)



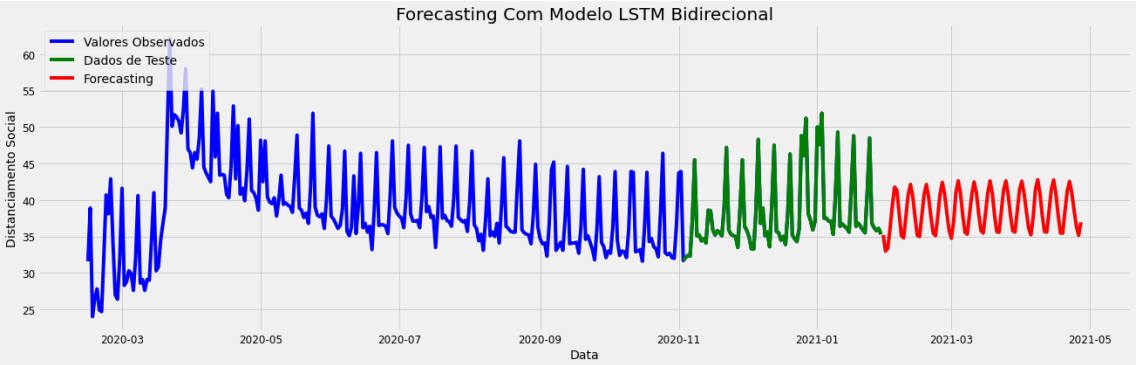
Model 23 – Optimized LSTM - IA



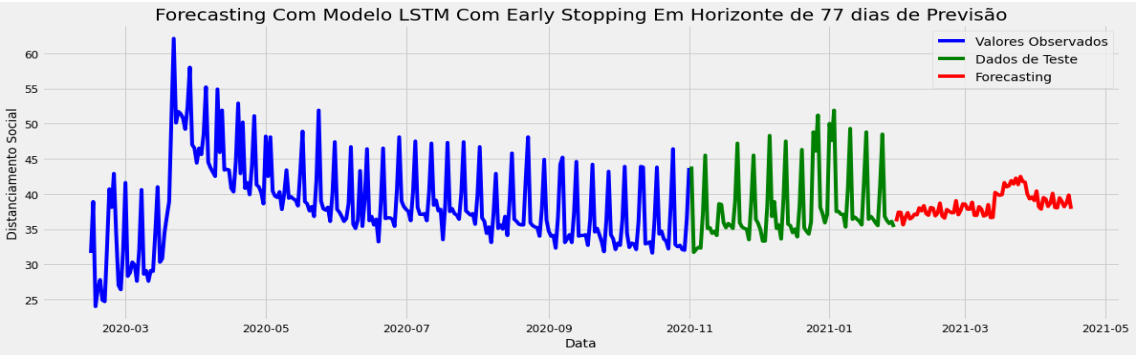
Model 24 – Stacked LSTM - IA



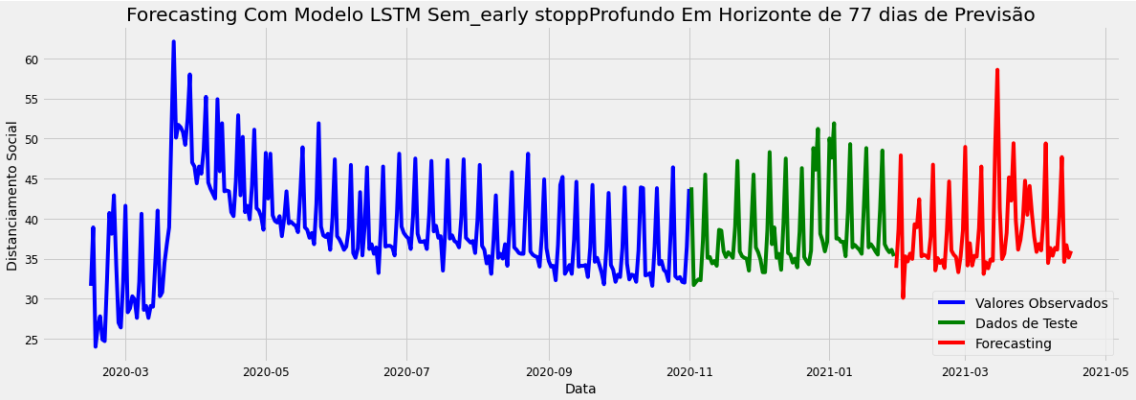
Model 25 – Bidirectional LSTM - IA



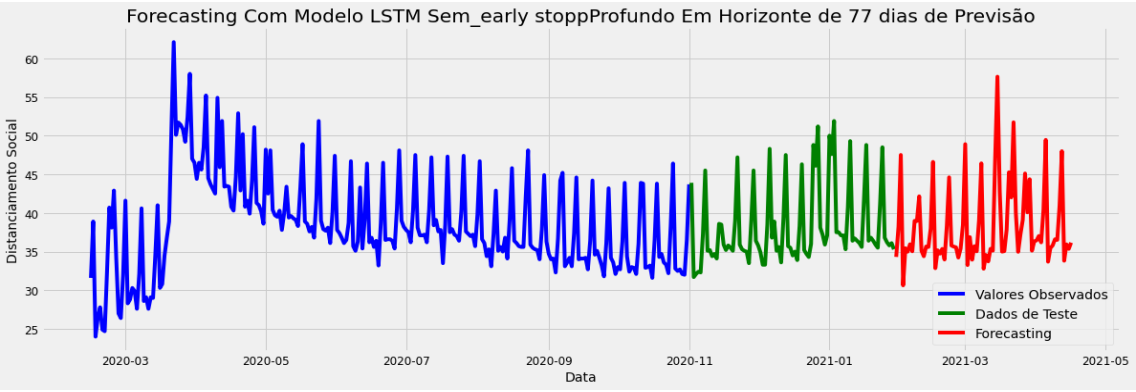
Model RNN02 – LSTM Vanilla IA – Epoch = 130 – avoid Overfitting



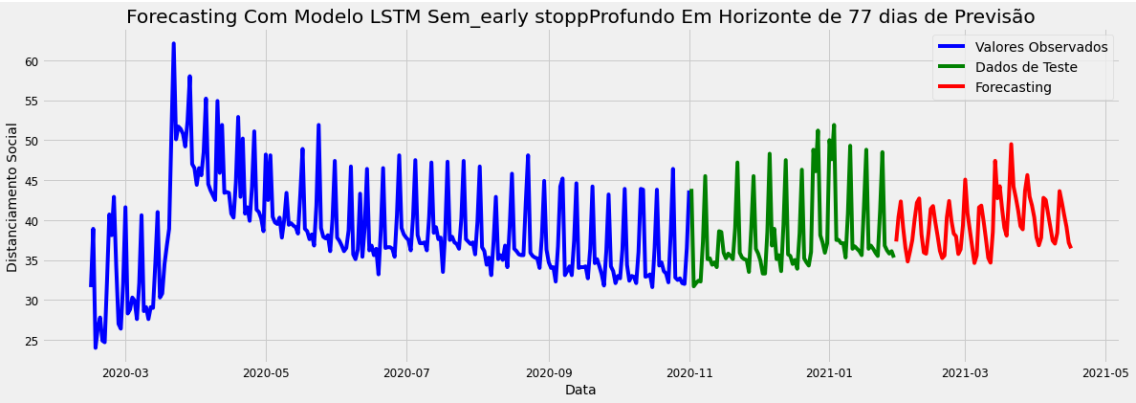
Model RNN02 – LSTM Vanilla IA – Epoch = 200



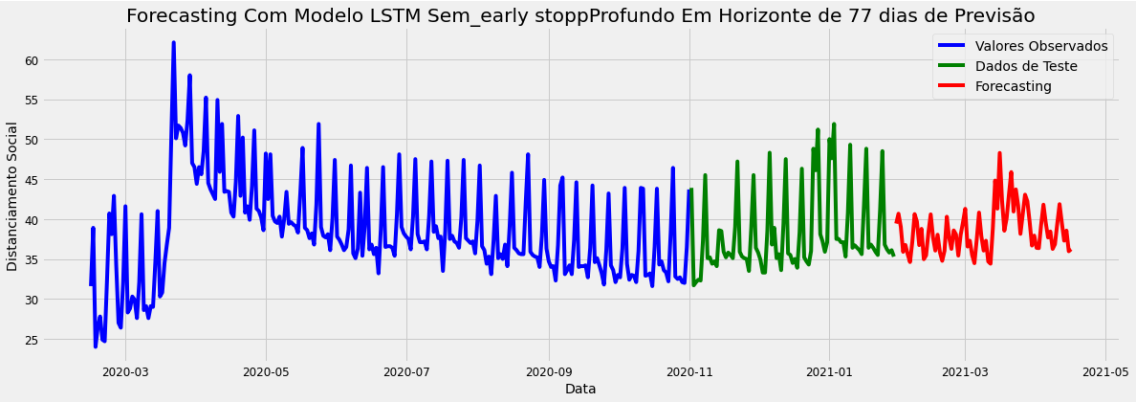
Model RNN02 – LSTM Window IA



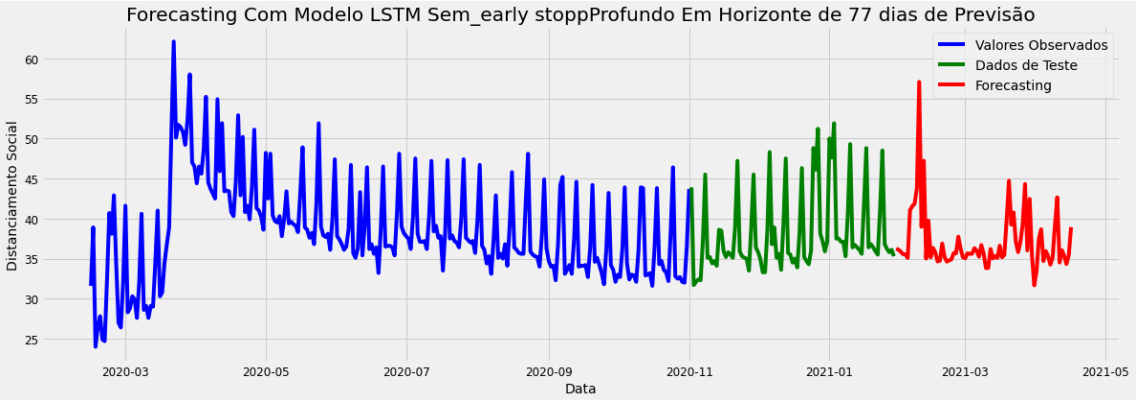
Model RNN02– LSTM Time Steps



Model RNN02– LSTM Stateful



Model RNN02– LSTM Stacked



Model Results:

Estratégia de Atendimento à População no Combate à Covid-19				RMSE	RMSE	RMSE
Nº	Arquitetura	Modelo	Setup	treino	validação	teste
1	Métodos Estatísticos	BASE 11	Método Naive			7,6327
2	Métodos Estatísticos	Forecasting 12	Exponential Smoothing v1			7,3602
3	Métodos Estatísticos	Forecasting 12	Exponential Smoothing v2			8,7714
4	Métodos Estatísticos	ARIMA 13	ARIMA LOG (1, 0, 1)			5,1253
5	Métodos Estatísticos	ARIMA 14	ARIMA LOG (1, 1, 1)			4,9166
6	Métodos Estatísticos	ARMA 15	ARMA (1, 0)			5,1045
7	Métodos Estatísticos	ARMA 15	ARMA (3, 5)			4,7948
8	Métodos Estatísticos	ARMA 15	ARMA (9, 6)			3,5966
9	Métodos Estatísticos	ARIMA 16	ARIMA (4, 1, 5) Forecast			3,7513
10	Métodos Estatísticos	SARIMAX 17	SARIMAX (2, 1, 1) (2, 1, 0, 10)			6,5842
11	Métodos Estatísticos	SARIMAX 18	SARIMAX (1, 1, 1) (0, 1, 1, 10)			5,0104
12	IA - Deep Learning	LSTM 22	LSTM (3 repetições)			5,8273
13	IA - Deep Learning	LSTM 22	LSTM (5 repetições)			5,9767
14	IA - Deep Learning	LSTM 23	LSTM Otimizado			4,6203
15	IA - Deep Learning	LSTM 24	Stacked LSTM			7,0475
16	IA - Deep Learning	LSTM 25	LSTM Bidirecional			5,0781
17	IA - Deep Learning	DeepAR 26	DeepAR			
18	Inteligência Artificial - IA	RNA - MLP 1	MLP Vanilla	5,1734		5,3914
19	Inteligência Artificial - IA	RNA - MLP 2	MLP e Método Window	3,1401		2,9119
20	IA - Deep Learning	RNN - LSTM 1	LSTM Vanilla - 130 epochs	4,7371	4,1557	4,7485
21	IA - Deep Learning	RNN - LSTM 1	LSTM Vanilla - 200 epochs	2,6281	2,7633	3,2239
22	IA - Deep Learning	RNN - LSTM 2	LSTM e Método Window	2,7596	2,9540	3,1561
23	IA - Deep Learning	RNN - LSTM 3	LSTM e Time Steps	3,9509	3,4389	3,8108
24	IA - Deep Learning	RNN - LSTM 4	LSTM e Stateful	3,4706	3,2986	4,3423
25	IA - Deep Learning	RNN - LSTM 5	LSTM e Stacked	6,3807	5,7923	5,2433

RMSE = Raiz Quadrada do Erro Quadrático Médio

Model Architectures using AI

Model 22 – LSTM - IA (5 repetitions)

Número de repetições = 5

```
modelo_lstm.add(LSTM(50, activation = 'relu', input_shape = (n_input, n_features)))
modelo_lstm.add(Dropout(0.10))
modelo_lstm.add(Dense(100, activation = 'relu'))
modelo_lstm.add(Dense(100, activation = 'relu'))
modelo_lstm.add(Dense(1))
modelo_lstm.compile(optimizer = 'adam', loss = 'mean_squared_error')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=3, verbose=1, mode='auto')
modelo_lstm.fit_generator(generator, epochs = 200)
```

Model 23 – Optimized LSTM - IA

Número de repetições = 3

```
modelo_lstm.add(LSTM(40, activation = 'tanh', return_sequences = True, input_shape = (n_input, n_features)))
modelo_lstm.add(LSTM(40, activation = 'relu'))
modelo_lstm.add(Dense(50, activation = 'relu'))
modelo_lstm.add(Dense(50, activation = 'relu'))
modelo_lstm.add(Dense(1))
adam = optimizers.Adam(lr = 0.001)
modelo_lstm.compile(optimizer = adam, loss = 'mean_squared_error')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
modelo_lstm.fit_generator(generator, epochs = 100)
```

Model 24 – Stacked LSTM - IA

```
modelo_lstm.add(LSTM(200, activation = 'relu', return_sequences = True, input_shape = (1, 1)))
modelo_lstm.add(LSTM(100, activation = 'relu', return_sequences = True))
modelo_lstm.add(LSTM(50, activation = 'relu', return_sequences = True))
modelo_lstm.add(LSTM(25, activation = 'relu'))
modelo_lstm.add(Dense(20, activation = 'relu'))
modelo_lstm.add(Dense(10, activation = 'relu'))
modelo_lstm.add(Dense(1))
modelo_lstm.compile(optimizer = 'adam', loss = 'mean_squared_error')
modelo_lstm.fit(X, y, epochs5000, verbose = 1)
```

Model Architectures using AI Deep Learning

Model RNN01 – MLP Vanilla – IA

```
model.add(Dense(8, input_dim = look_back, activation = 'relu'))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
model.fit(trainX, trainY, epochs = 200, batch_size = 2, verbose = 2)
```

Model RNN01 – MLP Vanilla – IA

```
model.add(LSTM(4, input_shape = (1, look_back)))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
model.fit(trainX, trainY, epochs = 200, batch_size = 1, verbose = 2)
```

Model RNN02 – LSTM Vanilla – IA

```
model.add(LSTM(4, input_shape = (1, look_back)))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
model.fit(trainX, trainY, epochs = 200, batch_size = 1, verbose = 2)
```

Model RNN02 – LSTM Window – IA

```
model.add(LSTM(4, input_shape = (1, look_back)))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
model.fit(trainX, trainY, epochs = 200, batch_size = 1, verbose = 2)
```

Model RNN02 – LSTM Time Steps – IA

```
model.add(LSTM(4, input_shape = (None, 1)))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
```

```
model.fit(trainX, trainY, epochs = 200, batch_size = 1, verbose = 2)
```

Model RNN02 – LSTM Stateful – IA

```
model.add(LSTM(4, batch_input_shape = (batch_size, look_back, 1), stateful = True))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
fori in range(200):
model.fit(trainX, trainY, epochs = 1, batch_size = batch_size, verbose = 2, shuffle = False)
model.reset_states()
```

Model RNN02 – LSTM Stacked – IA

```
model.add(LSTM(4, batch_input_shape = (batch_size, look_back, 1), stateful = True, return_sequences = True))
model.add(LSTM(4, batch_input_shape = (batch_size, look_back, 1), stateful = True))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
fori in range(200):
model.fit(trainX, trainY, epochs = 1, batch_size = batch_size, verbose = 2, shuffle = False)
model.reset_states()
```

Model RNN02 – LSTM Stacked – IA (número maior de camadas)

```
model = Sequential()
model.add(LSTM(100, batch_input_shape = (batch_size, look_back, 1), activation = 'relu', return_sequences = True))
model.add(LSTM(50, activation = 'relu', return_sequences = True))
model.add(LSTM(20,  activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(5, activation = 'relu'))
model.add(Dense(1))
 model.compile(loss = 'mean_squared_error', optimizer = 'adam')
for i in range(400):
   model.fit(trainX, trainY, epochs = 1, batch_size = batch_size, verbose = 1, shuffle = False)
   model.reset_states()
```

The models were based on courses from the Data Science Academy DSA and the Community timeline on the portal:
www.datascienceacademy.com.br