# Forecasting Temperature Feature

## Abstract

After using EDA techniques to know the data, it is time to step forward and find the best possible model to forecast the future with current data. There are several techniques exist to deal with time-series datasets.

In this project, I will use two of the most popular for it,

- Autoregressive Moving Average,
- Recurrent Neural Network

Each of these two technologies has different topologies to deal with series data. In the first step, I will briefly define each of these technologies and look at their pros and cons.

## Autoregressive Moving Average (ARMA)

The technique uses the relation between the lags in the data series and finds the hyperparameters to tune the model based on the ACF (Autocorrelation function) and PACF (Partial-Autocorrelation function). This methodology has at least two hyperparameters, p and q. The value of the p represents autoregressive and gain from the number of considering spikes from PACF which shows the direct correlation of the lag with the first value of the time series and the value of the q represents the moving average and gain from considering spikes from ACF which shows the degree of differencing or integration of the lag with the first value of the time series.

The advanced models in this methodology are ARIMA and SARIMA which added the concept of integrated and sessional to the basic model. The SARIMA model needs to feed with 7 hyperparameters,

- p, represents the number of lagged values or autoregressive terms,
- d, represents the degree of differencing or integration,
- q, represents the number of moving average terms or error terms,
- P, D, and Q have the same meaning as lowercase but they apply to the seasonal component of the data,
- S, represents the length of the seasonal cycle.

In ARMA methodology we have a favorite best practice to find p and q values which is known as the Box-Jenkins Method

| ACF Shape | Indicated Model |
|---|---|
| **Exponential, decaying to zero** | Autoregressive model. Use the partial autocorrelation plot to identify the order of the autoregressive model. |
| **Alternating positive and negative, decaying to zero** | Autoregressive model. Use the partial autocorrelation plot to help identify the order. |
| **One or more spikes, the rest are essentially zero** | Moving average model, order identified by where the plot becomes zero. |
| **Decay, starting after a few lags** | Mixed autoregressive and moving average (ARMA) model |
| **All zero or close to zero** | Data are essentially random. |
| **High values at fixed intervals** | Include seasonal autoregressive terms. |
| **No decay to zero** | The series is not stationary. |

The most important advantage of using this methodology is, using the less number hyperparameters which leads to increasing the speed of the learning model (ARMA has 2, ARIMA 3, and SARIMA 7 hyperparameters). Another advantage is the capability to forecast the future. But it has different disadvantages as well,

- The data should be stationary,
- The SARIMA model cannot be used for series without cycle and tunning of hyperparameters should be done manually,
- In the SARIMA model the value of the ‘s’ cannot be large, it takes too many resources and increases the time of training.

In the series with sessional cycles such as temperature, we cannot use a daily basis for the SARIMA model (the value of s is equal to 365). In this kind of situation, we use some techniques to merge the series values such as mean, min, mode, etc. and another point that should be considered is, the selecting period of merging, the values should provide meaningful for the sessional cycle in this project weekly or monthly for the temperature data.

## Recurrent Neural Network

This methodology of deep learning design for the processing languages of the separated words is not useful for understanding of whole sentence meaning, but using the same methodology for time series data is useful because same as the sentences there is a relation between the data in the time series.

Two of the most favorite models are SimpleRNN and LSTM each of them has its advantages and disadvantages, but in general, this methodology works for the daily prediction and the near future and if the data is not stationary it works (if data is stationary the training time will be decreased). The SimpleRNN model has fewer hyperparameters than LSTM but it is not good for long-length sample input to the model, On the other hand, the LSTM is designed for remembering the beginning values of sample length to increase their affection in the result.

To feed the time series data to RNN models we need to do some preparation and create our samples with the desired length and we need to use the randomly and test different lengths to find the best one for our data (the best practice of selecting length for temperature is between 15 to 30).

One of the most important advantages of using RNN methodology works with different features to forecast one specific feature in a time series which separates it from the traditional methodology that supports just one feature as input. Based on this feature I will use three different strategies to forecast the temperature,

- Use the temperature feature
- Add manual features that are different between input from 1 day to 7 days
- Add manual features that shift between input from 1 day to 7 days

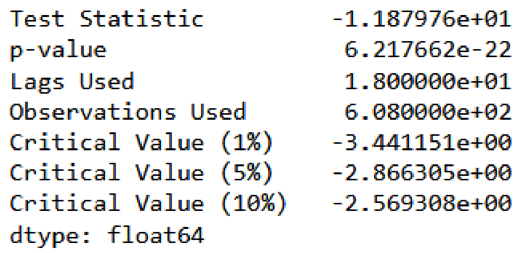
This practice will help me narrow the strategy for modeling other features (specifically emission ones).

## Modeling

Till now we just gained some knowledge of these two methodologies to deal with time series datasets, I will work with both of them and compare next week's forecast result based on the real value. Here I just share the best possible models I applied to the data.

### SARIMA

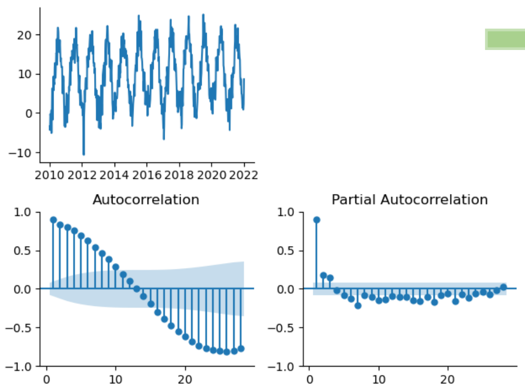
To gain the goal of forecasting next week's temperature value I need to resample data for the SARIMA methodology weekly and I use the mean value here. As discussed earlier essential situation to work with the SARIMA model is 'data should be stationary', I use the statics test as a result the output series of weekly resampling of temperature data in my dataset is stationary.



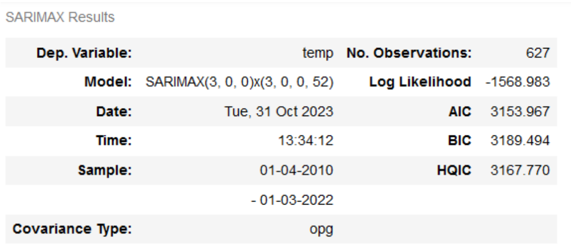
```
Test Statistic          -1.187976e+01
p-value                  6.217662e-22
Lags Used                1.800000e+01
Observations Used        6.080000e+02
Critical Value (1%)     -3.441151e+00
Critical Value (5%)     -2.866305e+00
Critical Value (10%)    -2.569308e+00
dtype: float64
```

**Note**: if the data you are working on is not stationary you first need to make it stationary, one of the simple methods you can use is diff() which gives the difference between the values of two values in a series.

After I was sure about the stationary situation of the series, I checked the ACF and PACF graphs for my first guest,

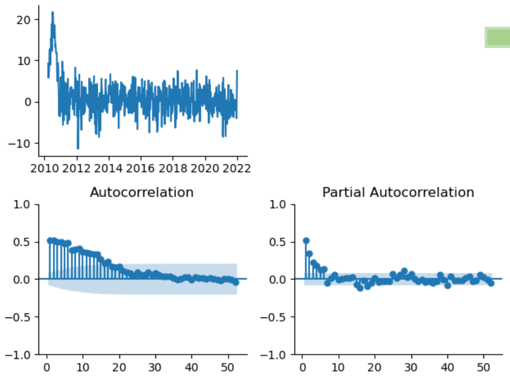


SARIMAX(3,0,0)(3,0,0,52)

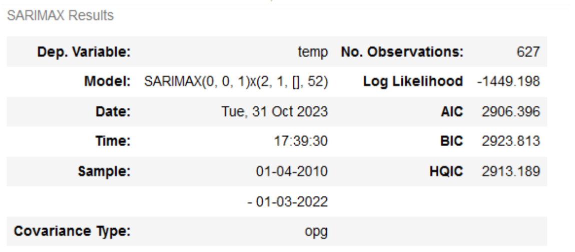

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | temp | **No. Observations:** | 627 |
| **Model:** | SARIMAX(3, 0, 0)x(3, 0, 0, 52) | **Log Likelihood** | -1568.983 |
| **Date:** | Tue, 31 Oct 2023 | **AIC** | 3153.967 |
| **Time:** | 13:34:12 | **BIC** | 3189.494 |
| **Sample:** | 01-04-2010 | **HQIC** | 3167.770 |
| | - 01-03-2022 | | |
| **Covariance Type:** | opg | | |

I trained the SARIMA model with different values (manually & grid search), at the end I decided to follow the below hyperparameters for forecasting,

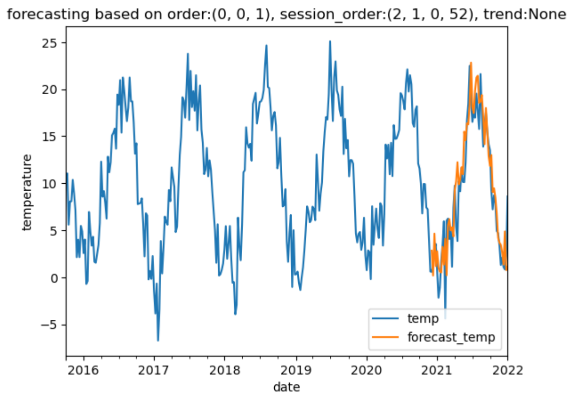


SARIMAX(0,0,1)(2,1,0,52)

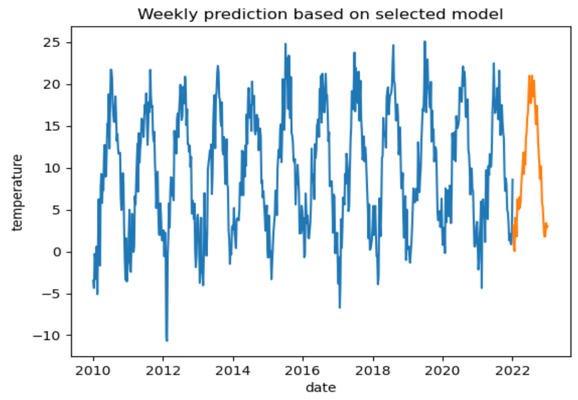

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | temp | **No. Observations:** | 627 |
| **Model:** | SARIMAX(0, 0, 1)x(2, 1, [], 52) | **Log Likelihood** | -1449.198 |
| **Date:** | Tue, 31 Oct 2023 | **AIC** | 2906.396 |
| **Time:** | 17:39:30 | **BIC** | 2923.813 |
| **Sample:** | 01-04-2010 | **HQIC** | 2913.189 |
| | - 01-03-2022 | | |
| **Covariance Type:** | opg | | |

*Prediction on test space*



*Forecasting the next year*

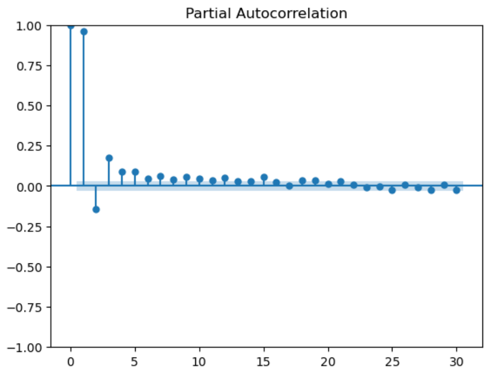


The predicted value for next week's temperature is "3.2928"

## RNN and LSTM

We saw this methodology is not dependent on the situation of the series and can work with data frame as input as well, but to feed the model it is needed to reshape data to be acceptable for the model. there are some best practices for using deep learning models and I will discuss a few of them in this project.

The first important point to be considered is the length of input sentences (in the time series the samples) to be enough to find the target and not too long that has not necessary information to mislead the model. I used the PACF graph to find the reasonable size of needed samples and got the number of spikes before it reached less than 0.05. I simply follow the basic definition of PACF which is the separate relative between the spike sample and with first sample.



it is 15 for my data.

In all deep learning models selecting the loss function is important and here I chose the mean square error (as we faced the regression model). The other method that helped me is using the call-back method

- to save the optimal parameters to reuse in the same topology (optimizer and loss function can change and continue learning),
- to check the live loss plot on each epoch and early stopping to avoid time-consuming.

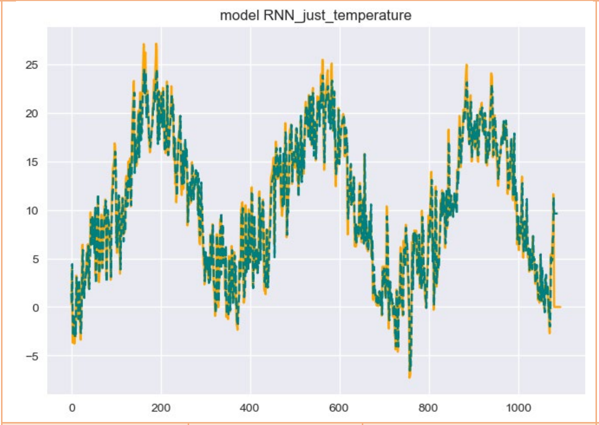
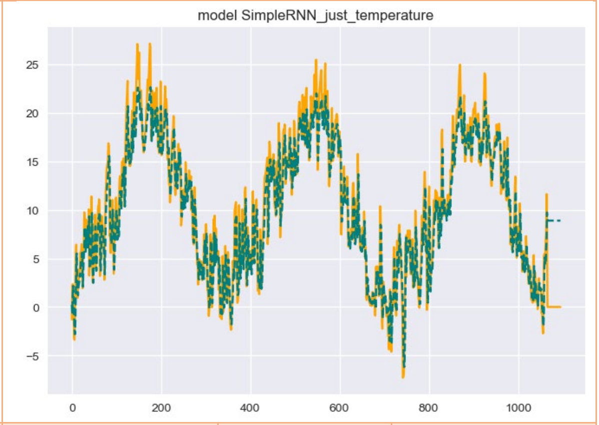
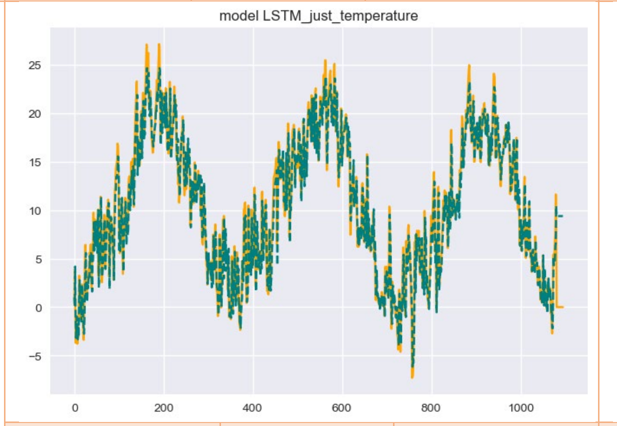
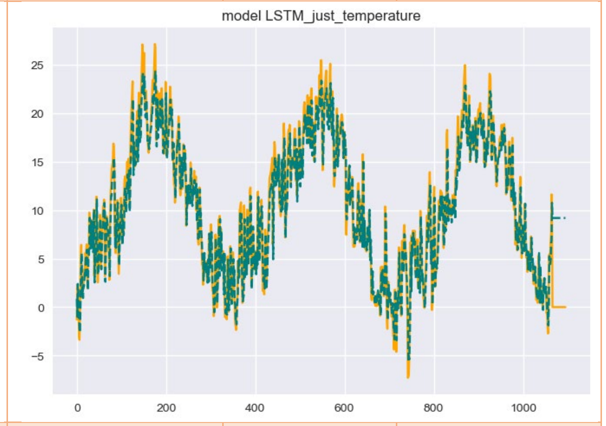
Using the activation function is essential in deep learning, but selecting the correct one can hardly affect the learning time, and using the simple in the middle layers is the best practice.

Now, after briefly looking at some points of deep learning and the RNN I will share the best model output of each situation I have worked.

**Note**: All the training has been done for two different lengths of sending samples to RNN models.

## *Use the temperature feature solitude*

In this situation, we will face the series it is too similar to work with traditional machine learning but has the advantage of forecasting daily values instead of dependency cycling of data, the idea of using series in LSTM helps us to reduce training time as well and it will be great for simple series.

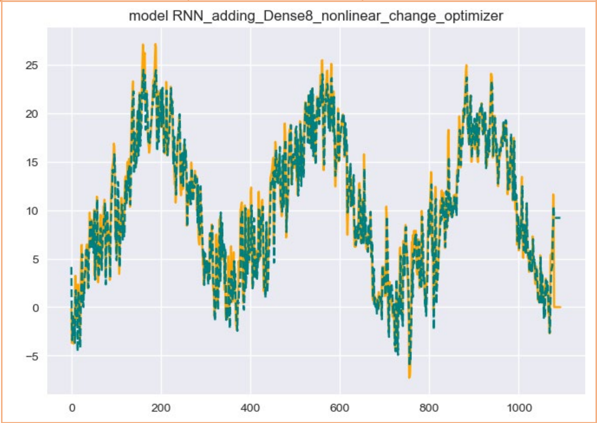
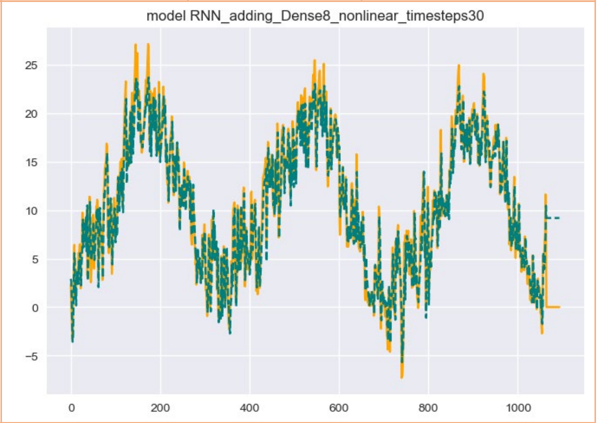
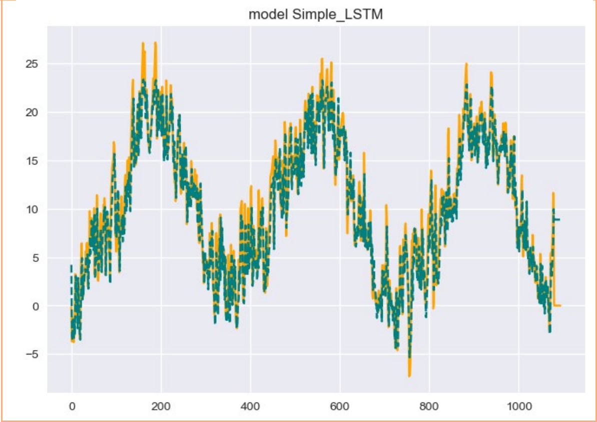
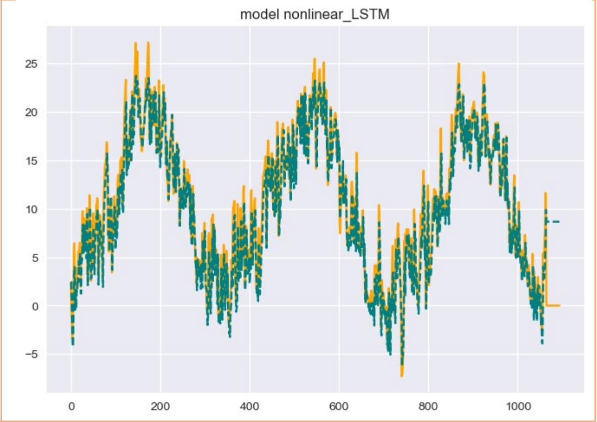
| | Length: 15 samples | | | Length: 30 samples | | |
|---|---|---|---|---|---|---|
| | Epoch | MSE | Parameters | Epoch | MSE | Parameters |
| **SimpleRNN** |  | | |  | | |
| | 56 | 97.17 | 4,289 | 40 | 91.9 | 4,289 |
| **LSTM** |  | | |  | | |
| | 54 | 98.45 | 16,961 | 49 | 93.35 | 16,961 |

According to the above graphs, when the timesteps increased to 30 the mean prediction was better than timesteps equal to 15 but at that time not good with the edges, so because there is not too much difference between MSE, the selected situation is SimpleRNN algorithm with timesteps 15.

## *Add manual features that are different between input from 1 day to 7 days*

The first idea is adding new features to help the algorithm see the data relation in each row in addition to looking at time series as a series. I used the possibility of working as a data frame in RNN topology.

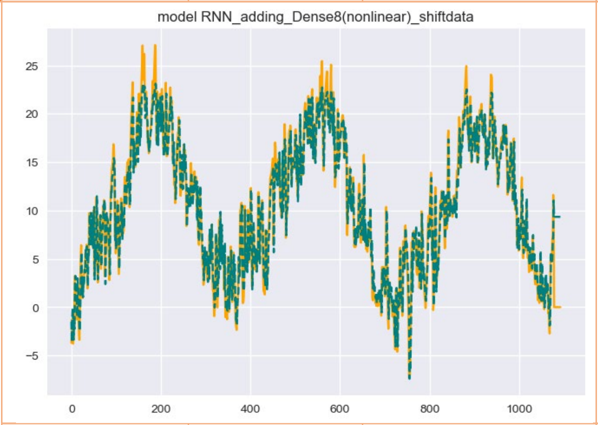
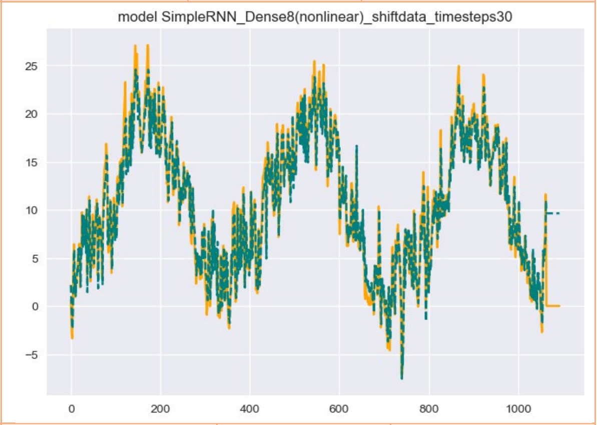
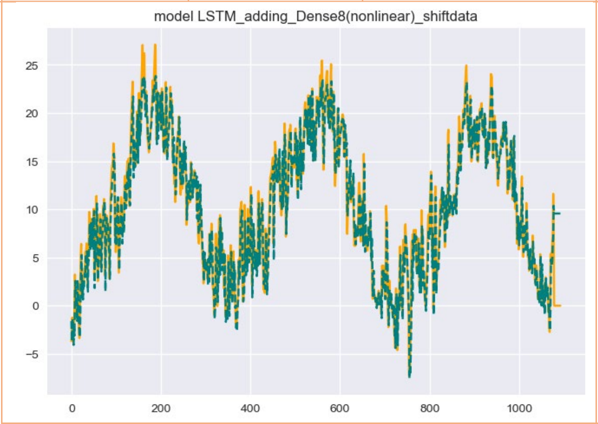
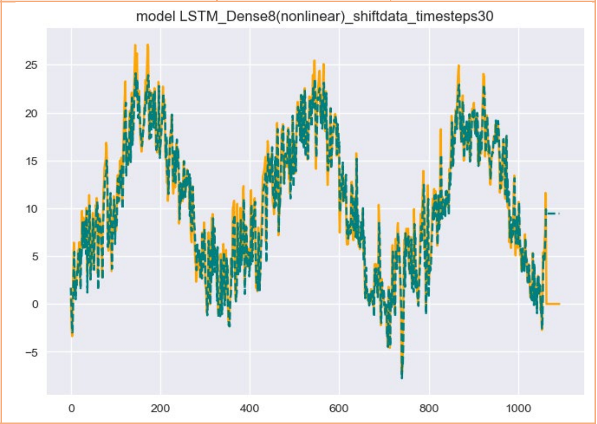
To gain work with data frames in RNN, I increased the number of features for temperature forecasting by adding a minus value between the temperature of one day to 7 days and calling them new features and I used the same separation of train/test data to be comparable with the above situation.

| | **Length: 15 samples** | | | **Length: 30 samples** | | |
|---|---|---|---|---|---|---|
| | Epoch | MSE | Parameters | Epoch | MSE | Parameters |
| **SimpleRNN** |  | | |  | | |
| | 45 | 99.72 | 1,585 | 38 | 92.45 | 1,585 |
| **LSTM** |  | | |  | | |
| | 71 | 97.06 | 5,281 | 32 | 98.18 | 19,217 |

The selected algorithm to continue for the final decision is SimpleRNN with timesteps equal to 30.

### *Add manual features that shift between input from 1 day to 7 days*

In the before experience, I had used the difference between data to create new features, here I have used the shift data for the same purpose.

| | Length: 15 samples | | | Length: 30 samples | | |
|---|---|---|---|---|---|---|
| | Epoch | MSE | Parameters | Epoch | MSE | Parameters |
| **SimpleRNN** |  | | |  | | |
| | 46 | 93.94 | 5,201 | 58 | 91.67 | 5,201 |
| **LSTM** |  | | |  | | |
| | 51 | 98.87 | 19,217 | 41 | 96.34 | 19,217 |

In this situation, LSTM algorithms worked better to catch the edges, and with timesteps equal to 15 faced the better result.

### *Conclusion*

In general, when I used the RNN all algorithms worked better to predict the lower edges than the upper one. In the selection of the best possible situation and algorithm for forecasting temperature data in my dataset with RNN, I chose the SimpleRNN with timesteps 30 when I had added the difference manually. Now, I need to predict the next week of data temperature to compare with using SARIMA algorithms.

To forecast with an RNN structure the number of outputs should be equal to the steps you want to predict, for example in my case I want to predict 7 days after the final day of information, so I need to take 7 values as output from the model. I used the selected model and changed the 7 values as output and learned with whole data (without separating of train/test). The temperature for the next week is "3.9062" with using RNN.

## Conclusion

According to the main data, the temperature on 2023-01-07 is equal to "4.0000" which shows that RNN predicts better than the SARIMA model for this data, but it does not mean the RNN algorithms work better than the SARIMA in general.