Machine Learning
Project 2019/2020

SAPIENZA
UNIVERSITÀ DI ROMA

COVID-19 Time Series Forecasting

Luca Podo
Dario Aragona

Professor Paola Velardi

Index

Introduction

In this project we want to propose the study of a particular type of data, *time series*. Our goal is to predict data about the spread of the COVID-19 in Italy, and compare the predictions with the actual numbers. We used data available about the COVID-19 and also other informations that we thought to be relevant with respect to the spread.
Different solutions are developed using three main models specific for time series.
Solutions are different about data sources, models and type of approach.
The results from the different models are then compared to see which one could predict better among the others.

1. Time series

Time series data are series indexed by a temporal order. A simple example can be the measurement of the temperatures during a certain period of time, for example a day or a month. A complete definition is:

An ordered sequence of values of a variable at equally spaced time intervals[1]

These series can be represented by a line chart where on the y-axis we can find the value of the entity that we are measuring that can be discrete or continuous, and on the x-axis we can find the temporal interval, which is discrete. In this way we can see how the value changes during a temporal period. The data can be visualized like in the following image[2].

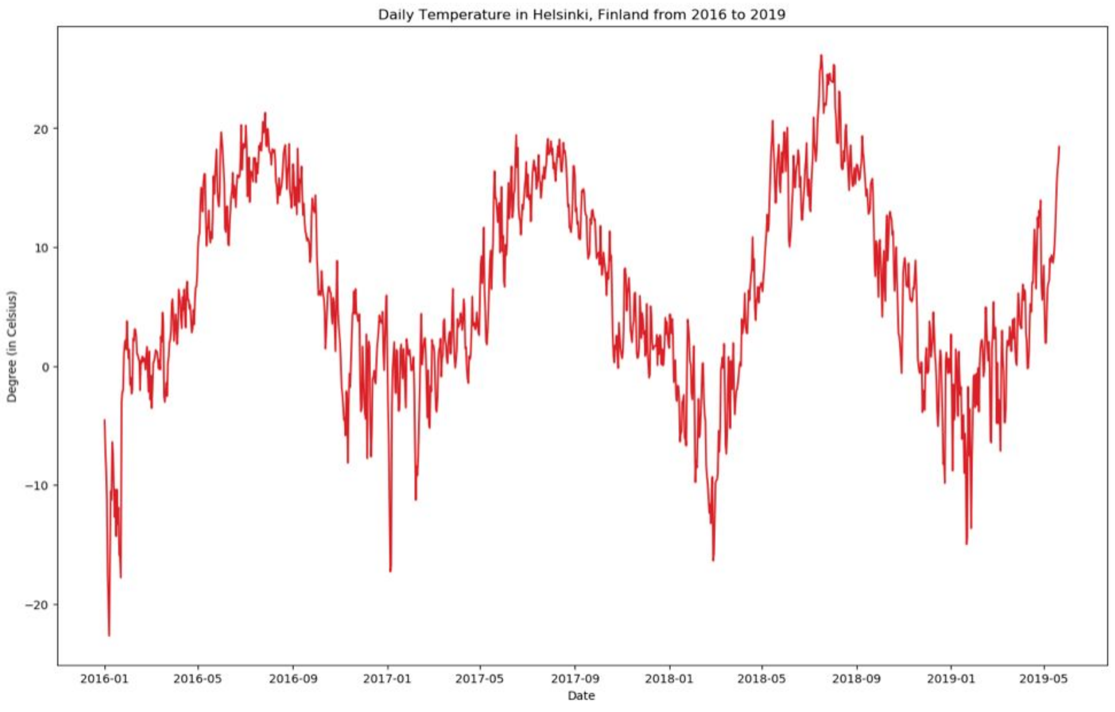


Fig.1 Example of Temperature time serie

A main application on this kind of data is to try to understand the model that drive the data to forecast a specific value in a specific date or time period. The following image show the an example of forecasting based on the previous history of the data.

[1] https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc41.htm
[2] https://medium.com/@llmkhoa511/time-series-analysis-and-weather-forecast-in-python-e80b664c7f71

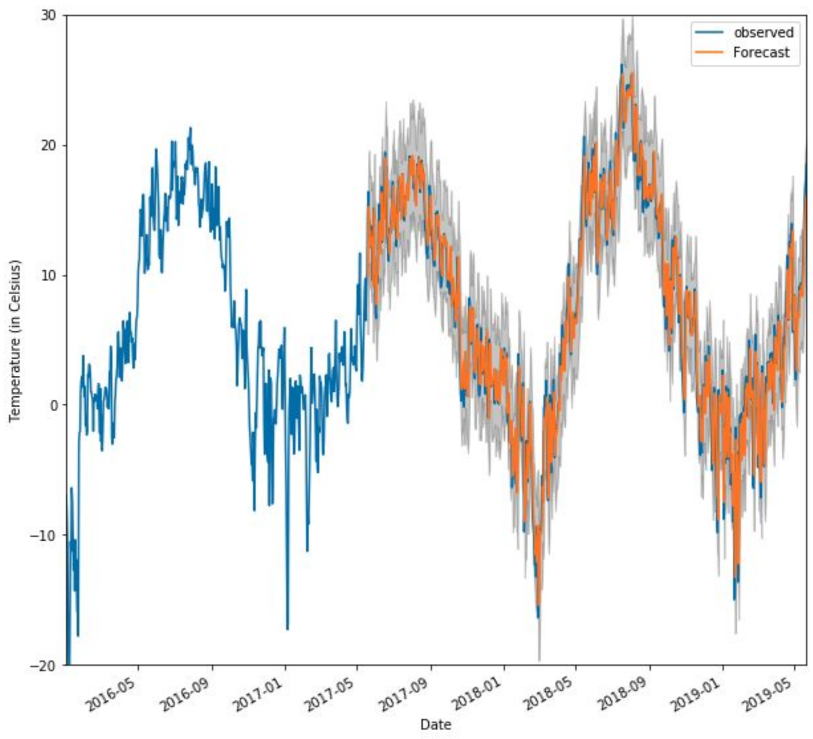


Fig.2 Example of forecast based on data history

When we analyse a time series we need to take care also of three main aspects:

- ***stationary***: the statistical properties of a time series do not change over time, this means it have constant mean and variance during time. This aspect is very important because non-stationary data, as a rule, are unpredictable and cannot be modeled or forecasted. A prediction on a non stationary series could produce a relationship between two variables, but in reality they are not related together.
- ***autocorrelation***: when a time series is linearly related to a lagged version of itself. An example could be that observed data in a specific time and the next 24h are related togher

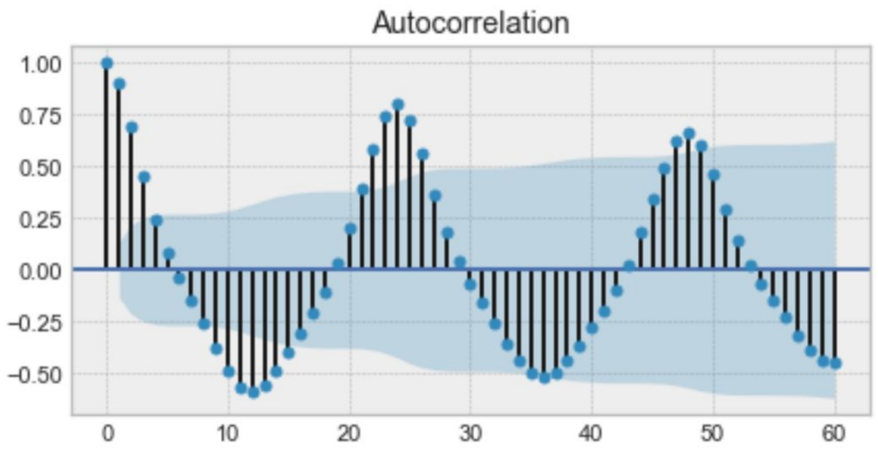


Fig. 3 Example of autocorrelation

On the above image is showed how the first data is related to the data in the next 24h. When we work with time series is important to be sure to not have autocorrelation between data, for example because some models like LSTM[3] basic assumption is that there is no autocorrelation to avoid overfitting

- ***seasonality***: refers to periodic fluctuations. This information can also be deducted from the autocorrelation. An example is the number of products sold by Amazon during the holidays. We can observe how during the christmas holiday the products sold increase, and this happen every year. Seasonal adjustment is the process of estimating and then removing from a time series influences that are systematic and calendar related. Observed data needs to be seasonally adjusted as seasonal effects can conceal both the true underlying movement in the series, as well as certain non-seasonal characteristics which may be of interest to analysts.

A time series can be exploded in four main components:

1. **Trend**: The trend shows the general tendency of the data to increase or decrease during a long period of time. It can be linear or nonlinear. It is defined as long term movement in a time serie.
2. **Seasonality**: The repeating short-term cycle in the series
3. **Noise**: The random variations in the series. These fluctuations are unforeseen, uncontrollable, unpredictable, and are erratic

A mathematical model can be

$$y_t = f(t)$$

where t is the instat time that we are observing and y the value at that time. Considering the three elements defined there are two models

- Additive: $y_t = T_t + S_t + R_t$. where T is the trend, S is the seasonality, and R the random contribution of noise
- Multiplicative: $y_t = T_t * S_t * R_t$

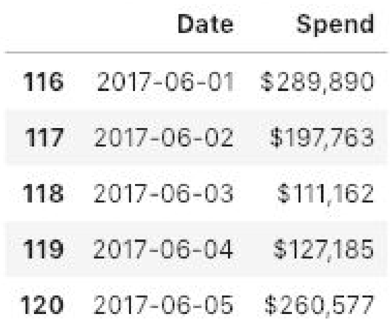
Another distinction in time series could be in terms of number of variables that play a role in the model that we want to train. For this reason, we can distinguish time series as:

- univariate
- multivariate

[3] https://medium.com/@dganais/autocorrelation-in-time-series-c870e87e8a65

1.1 Univariate time series

The term "univariate time series" refers to a time series that consists of single (scalar) observations recorded sequentially over time increments.
In other words, is a series with a single time-dependent variable. This time serie has just two columns, one for the time series variables and the other one for the temporal index. An example is shown below[4].



	Date	Spend
116	2017-06-01	$289,890
117	2017-06-02	$197,763
118	2017-06-03	$111,162
119	2017-06-04	$127,185
120	2017-06-05	$260,577

Fig. 4 Example of univariate time series

In the previous example, the time serie is about advertising spend during a certain period of time. The model try to use the previous history of this only column to try to forecast new values.

1.2 Multivariate time series

A multivariate time series has more than one time-dependent variables. Each variable depends both on its past values and on some dependencies with other variables: these dependencies are used for forecasting future values. The following table show an example of dataset about air pollution forecasting, with multiple columns for different values measured[5].

date	*pollution*	*dew*	*temp*	*press*	*wnd_dir*	*wnd_spd*	*snow*	*rain*
2010-01-02 00:00:00	129.0	-16	-4.0	1020.0	SE	1.79	0	0
2010-01-02 01:00:00	148.0	-15	-4.0	1020.0	SE	2.68	0	0
2010-01-02 02:00:00	159.0	-11	-5.0	1021.0	SE	3.57	0	0
2010-01-02 03:00:00	181.0	-7	-5.0	1022.0	SE	5.36	1	0
2010-01-02 04:00:00	138.0	-7	-5.0	1022.0	SE	6.25	2	0

In this case the value of the variable that we want to forecast is based not only on his past values but also on the past values of the other variables. We could predict the next value of pollution try to looking for a pattern in the history data of all the variables in the table.

[4] https://towardsdatascience.com/a-quick-start-of-time-series-forecasting-with-a-practical-example-using-fb-prophet-31c4447a2274
[5] https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

2. Dataset

Data is built putting together informations about Coronavirus spread, people mobility and demographic data: we look for a correlation between these informations in order to get predictions about virus.

2.1 COVID-19 dataset

Data about COVID-19 is collected by “*Sito del Dipartimento della Protezione Civile - Emergenza Coronavirus*” and is uploaded into its github repository[6].
It represents time series, each analyzing a different trend in a national or regional context on daily bases.
Our focus is on the daily prediction of the amount of current positive cases in that day.

We used both national and regional data, but their data format is the same.

2.1.1 National data format

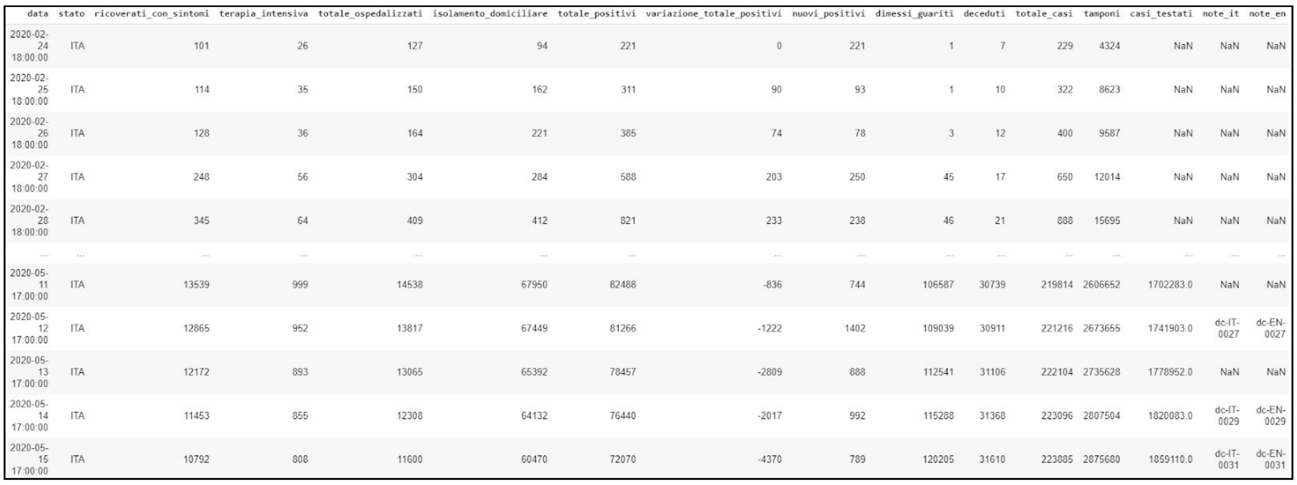
This dataset contains 16 columns, each described in the following table:

Column Name	**Description**	**Format**	**Example**
data	Date of notification	YYYY-MM-DD HH:MM:SS (ISO 8601) Ora italiana	2020-03-05 12:15:45
stato	Country of reference	XYZ (ISO 3166-1 alpha-3)	ITA
ricoverati_con_sintomi	Hospitalised patients with symptoms	Number	3
terapia_intensiva	Intensive Care	Number	3
totale_ospedalizzati	Total hospitalised patients	Number	3
isolamento_domiciliare	Home confinement	Number	3
totale_positivi	Total amount of current positive cases (Hospitalised patients + Home confinement)	Number	3
variazione_totale_positivi	News amount of current positive cases (totale_positivi current day - totale_positivi previous day)	Number	3
nuovi_positivi	News amount of current positive cases (totale_casi current day - totale_casi previous day)	Number	3
dimessi_guariti	Recovered	Number	3
deceduti	Death	Number	3
totale_casi	Total amount of positive cases	Number	3
tamponi	Tests performed	Number	3
casi_testati	Total number of people tested	Number	3

[6] https://github.com/pcm-dpc/COVID-19

note_it	Notes in italian language (separated by ;)	Text	pd-IT-000
note_en	Notes in english language (separated by ;)	Text	pd-EN-000

In the national case, our dataset contains n rows, where n is the number of days since the first data acquisition, that is on 24-05-2020. Some rows are shown in this image:



data	stato	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isolamento_domiciliare	totale_positivi	variazione_totale_positivi	nuovi_positivi	dimessi_guariti	deceduti	totale_casi	tamponi	casi_testati	note_it	note_en
2020-02-24 18:00:00	ITA	101	26	127	94	221	0	221	1	7	229	4324	NaN	NaN	NaN
2020-02-25 18:00:00	ITA	114	35	150	162	311	90	93	1	10	322	8623	NaN	NaN	NaN
2020-02-26 18:00:00	ITA	128	36	164	221	385	74	78	3	12	400	9587	NaN	NaN	NaN
2020-02-27 18:00:00	ITA	248	56	304	284	588	203	250	45	17	650	12014	NaN	NaN	NaN
2020-02-28 18:00:00	ITA	345	64	409	412	821	233	238	46	21	888	15695	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-05-11 17:00:00	ITA	13539	999	14538	67950	82488	-836	744	106587	30739	219814	2606652	1702283.0	NaN	NaN
2020-05-12 17:00:00	ITA	12865	952	13817	67449	81266	-1222	1402	109039	30911	221216	2673655	1741903.0	dc-IT-0027	dc-EN-0027
2020-05-13 17:00:00	ITA	12172	893	13065	65392	78457	-2809	888	112541	31106	222104	2735628	1778952.0	NaN	NaN
2020-05-14 17:00:00	ITA	11453	855	12308	64132	76440	-2017	992	115288	31368	223096	2807504	1820083.0	dc-IT-0029	dc-EN-0029
2020-05-15 17:00:00	ITA	10792	808	11600	60470	72070	-4370	789	120205	31610	223885	2875680	1859110.0	dc-IT-0031	dc-EN-0031

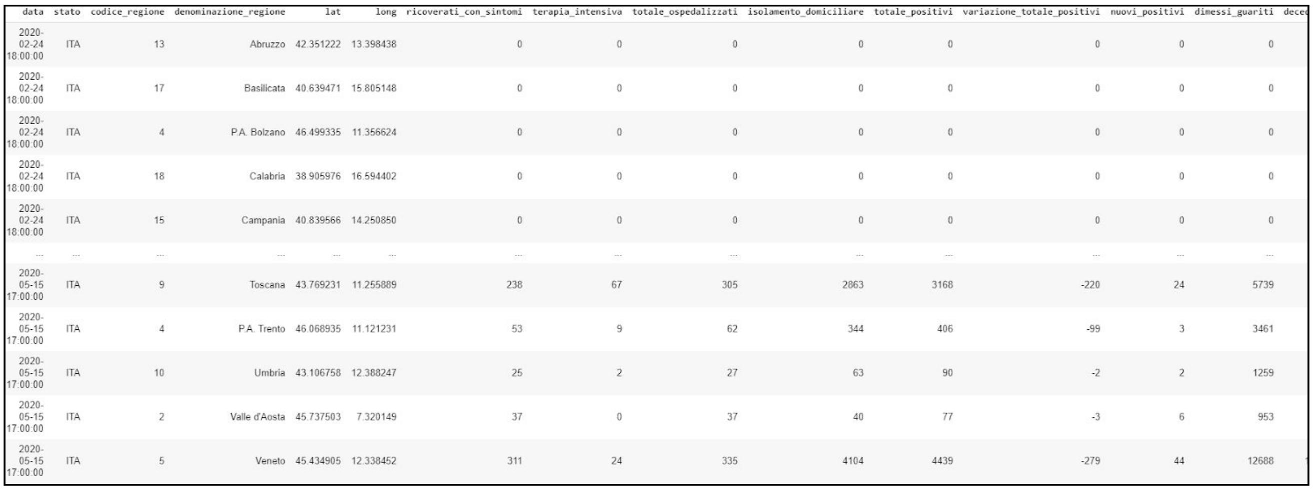
2.1.2 Regional data format

This dataset contains 20 columns, the only addictions with respect to the national one are region identifiers:

Column Name	Description	Format	Example
data	Date of notification	YYYY-MM-DDTHH:MM:SS (ISO 8601) Ora italiana	2020-03-05 12:15:45
stato	Country of reference	XYZ (ISO 3166-1 alpha-3)	ITA
codice_regione	Code of the Region (ISTAT 2019)	Number	13
denominazione_regione	Name of the Region	Text	Abruzzo
lat	Latitude	WGS84	109861.37.00
long	Longitude	WGS84	1174012.31.00
ricoverati_con_sintomi	Hospitalised patients with symptoms	Number	3
terapia_intensiva	Intensive Care	Number	3
totale_ospedalizzati	Total hospitalised patients	Number	3
isolamento_domiciliare	Home confinement	Number	3
totale_positivi	Total amount of current positive cases (Hospitalised patients + Home confinement)	Number	3

variazione_totale_positivi	News amount of current positive cases (totale_positivi current day - totale_positivi previous day)	Number	3
nuovi_positivi	News amount of current positive cases (totale_casi current day - totale_casi previous day)	Number	3
dimessi_guariti	Recovered	Number	3
deceduti	Death	Number	3
totale_casi	Total amount of positive cases	Number	3
tamponi	Tests performed	Number	3
casi_testati	Total number of people tested	Number	3
note_it	Notes in italian language (separated by ;)	Text	pd-IT-000
note_en	Notes in english language (separated by ;)	Text	pd-EN-000

In this case, the dataset contains $n * R$ rows, where n is the number of days since the first data acquisition, that is on 24-02-2020, and $R = 20$ is the number of regions. This means that in this situation it has more rows for each day due to each single region informations.



data	stato	codice_regione	denominazione_regione	lat	long	ricoverati_con_sintomi	terapia_intensiva	totale_ospedalizzati	isolamento_domiciliare	totale_positivi	variazione_totale_positivi	nuovi_positivi	dimessi_guariti
2020-02-24 18:00:00	ITA	13	Abruzzo	42.351222	13.398438	0	0	0	0	0	0	0	0
2020-02-24 18:00:00	ITA	17	Basilicata	40.639471	15.805148	0	0	0	0	0	0	0	0
2020-02-24 18:00:00	ITA	4	P.A. Bolzano	46.499335	11.356624	0	0	0	0	0	0	0	0
2020-02-24 18:00:00	ITA	18	Calabria	38.905976	16.594402	0	0	0	0	0	0	0	0
2020-02-24 18:00:00	ITA	15	Campania	40.839566	14.250850	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-05-15 17:00:00	ITA	9	Toscana	43.769231	11.255889	238	67	305	2863	3168	-220	24	5739
2020-05-15 17:00:00	ITA	4	P.A. Trento	46.068935	11.121231	53	9	62	344	406	-99	3	3461
2020-05-15 17:00:00	ITA	10	Umbria	43.106758	12.388247	25	2	27	63	90	-2	2	1259
2020-05-15 17:00:00	ITA	2	Valle d'Aosta	45.737503	7.320149	37	0	37	40	77	-3	6	953
2020-05-15 17:00:00	ITA	5	Veneto	45.434905	12.338452	311	24	335	4104	4439	-279	44	12688

2.2 Mobility dataset

Google[7] and Apple[8] make mobility data available to aid COVID-19 analysis. We thought this kind of data could be related to the virus spread, so we decided to add these informations in our analysis. Although the two different datasets have different formats, they are both time series expressing changes in users mobility during these months. In particular, they reflect the effects of lockdown in different ways.

[7] https://www.google.com/covid19/mobility/
[8] https://www.apple.com/covid19/mobility

2.2.1 Google Mobility data

Each Country Mobility Report dataset is presented by location and highlights the percent change in visits to places like grocery stores and parks within a geographic area. In our case, we only used dataset regarding Italy and its regions.

These datasets show how visits and length of stay at different places change compared to a baseline. Google calculated these changes using the same kind of aggregated and anonymized data used to show popular times for places in Google Maps.
Changes for each day since February 24th are compared to a baseline value for that day of the week, and places are:

- **Grocery & pharmacy**: Mobility trends for places like grocery markets, food warehouses, farmers markets, specialty food shops, drug stores, and pharmacies.

- **Parks**: Mobility trends for places like local parks, national parks, public beaches, marinas, dog parks, plazas, and public gardens.

- **Transit stations**: Mobility trends for places like public transport hubs such as subway, bus, and train stations.

- **Retail & recreation**: Mobility trends for places like restaurants, cafes, shopping centers, theme parks, museums, libraries, and movie theaters.

- **Residential**: Mobility trends for places of residence.

- **Workplaces**: Mobility trends for places of work.

2.2.2 Apple Mobility data

This dataset represents percent change of the number of driving directions requests for each country, subregion or city with respect to January 13th informations. Also in this case we selected only the Italian data subset.

Mobility data are encoded in 3 kind of transportation types:

- Driving
- Walking
- Transit

2.3 Demographic dataset

This dataset contains scalar information about population in each Italian region. It is built using data from ISTAT[9] and Ministero della Salute[10] websites.
This dataset contains 20 instances, one for each region, and 4 columns:

- Name of region
- Region population
- Population density
- Number of hospital beds

We assumed these kind of information to be relevant in our case study.

[9] http://dati.istat.it/
[10] http://www.dati.salute.gov.it/dati/homeDataset.jsp

3. Models

In this section we will say which model we decided to use and why we chosen it.
To reach our goal we focused our study on three models:

- VAR, Vector autoregression
- LSTM, Long Short Term Memory networks
- ConvLSTM

The presented models fit well on the kind of context we want to study. All the models allow us to make a forecasting of the variable that we want to observe on a specific future period of time based on the previous history of all the variables. There are differents other models to make time series forecasting, but non all support multivariate forecasting, an example can be Prophet developed by Facebook, that allow to make predictions only on univariate case. This is another reason why we decided to work with this kind of models.

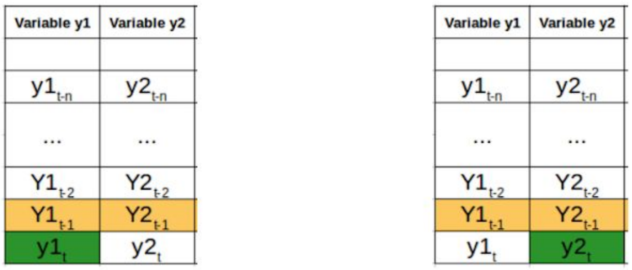
3.1 VAR

VAR, Vector AutoRegression, is a generalization of the AR (autoregression) model, that allow to work with multiple parallel time series, in our case a multivariate time series. The predictors are nothing but the lags (time delayed value) of the series.
It can be used when each time series influence the other, to try to look for the relationship that connect the variable togher. In our specific case, we start from the basic assumption that mobility data, number of new cases of positives and number of swabs are related together and they influence each other.
To use this model we need:

1. At least two time series
2. The time series should influence each other.

The relationship that occur is bi-direction this means that is not that only a variable influence another, but that they influence each other.

In VAR each variable is a linear function of his values and the past value of the other variables. In other words we can think variable with their history in past like follow.



Variable y1	Variable y2
$y1_{t-n}$	$y2_{t-n}$
...	...
$Y1_{t-2}$	$Y2_{t-2}$
$Y1_{t-1}$	$Y2_{t-1}$
$y1_t$	$y2_t$

Variable y1	Variable y2
$y1_{t-n}$	$y2_{t-n}$
...	...
$Y1_{t-2}$	$Y2_{t-2}$
$Y1_{t-1}$	$Y2_{t-1}$
$y1_t$	$y2_t$

Fig. Tabular representation of two variable in VAR[11]

[11] https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/

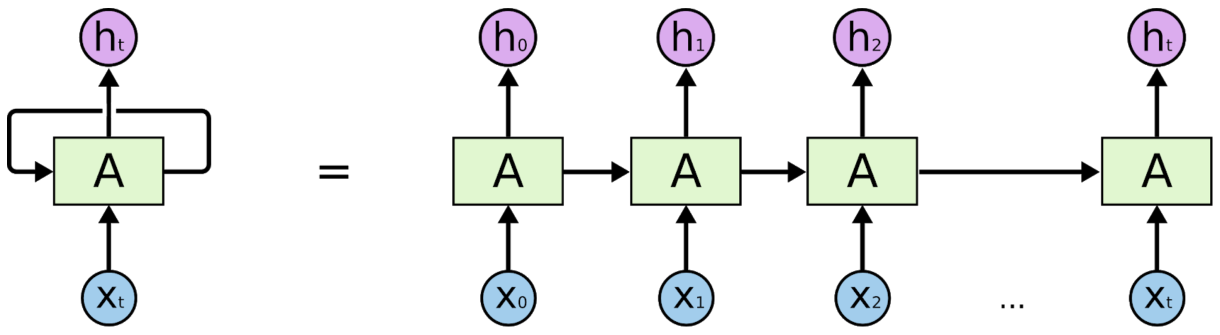
In the previous example we can notice how in the first table, the variable y1 at time t is calculated as the linear function of itself at time t-1 and the variable y2 at time t-1. The same is for the variable y2. More in general we could thing to multiple columns, where for example another variable y3 can be calculated as linear function of itself at time t-1, and also y2 and y1 at time t-1.
When we express a variable w.r.t the other we have also to consider the constant terms for each variables, the error associated at time t-1 and the coefficient related to each state of the variables at time t-1. This values comes from the AR model, where instead of considering the single serie, we consider multiple series.
An important aspect of VAR is that it is able to figure out the relationship that occurs between the different variables.
When we work with this kind of model is important to check if each time series is stationary and if not it need to be made stationary. We will see how we have ensured this propriety for VAR in the next chapter.

3.2 LSTM

LSTM Neural Networks[12], which stand for Long Short-Term Memory, are a particular type of Recurrent Neural Networks (RNN).

Generally, RNNs' connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior, using their internal state (memory) to process variable length sequences of inputs.

They are networks with loops in them, allowing information to persist.

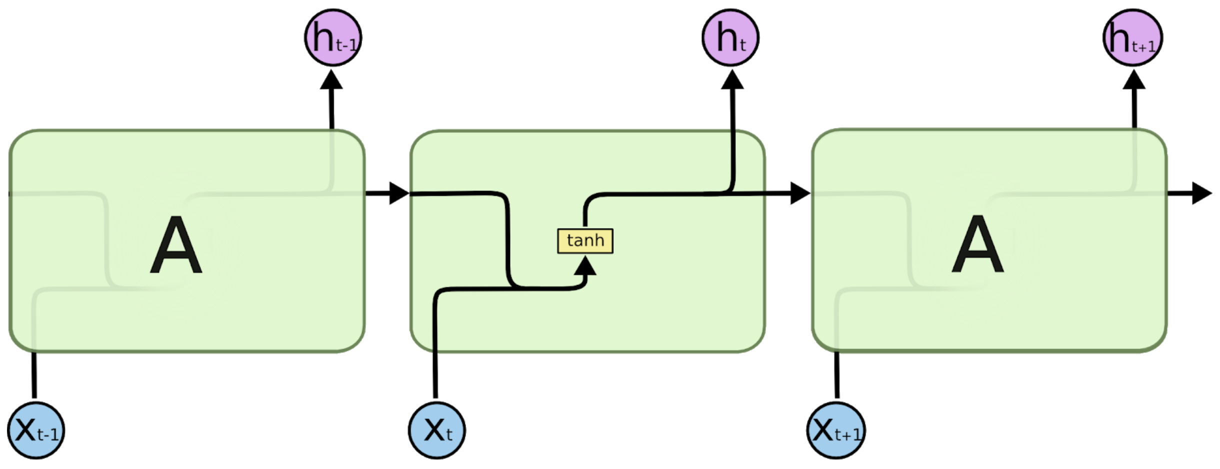


In the above diagram, a chunk of neural network, A, looks at some input x_i and outputs a value h_i. A loop allows information to be passed from one step of the network to the next.

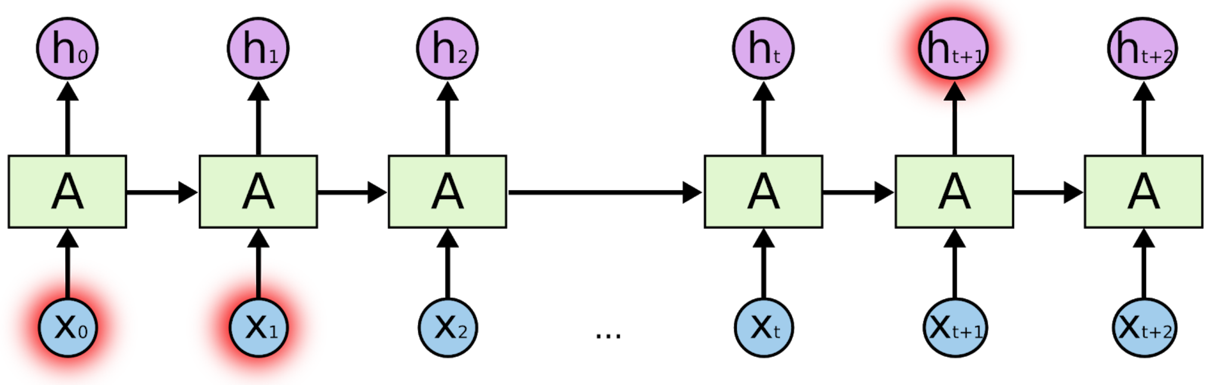
RRN can be thought of as multiple copies of the same network, each passing a message to a successor.

In standard RNNs, this repeating module have a very simple structure, such as a single tanh layer.

[12] https://colah.github.io/posts/2015-08-Understanding-LSTMs/



The problem is that it's possible that gap between the relevant information and the point where it is needed becomes very large.



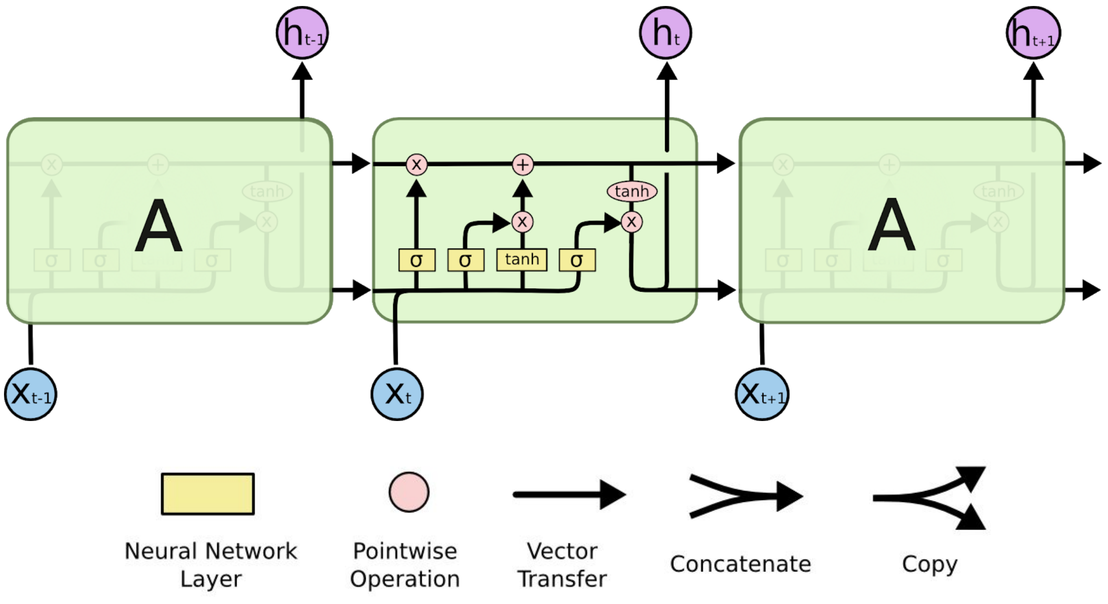
Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.
This problem is known as "short-term memory", and is caused by the infamous vanishing gradient problem: as the RNN processes more steps, it has troubles retaining information from previous steps. Short-Term memory and the vanishing gradient is due to the nature of back-propagation.

The gradient is the value used to adjust the networks internal weights, allowing the network to learn: the bigger the gradient, the bigger the adjustments and vice versa.
Here is where the problem lies. When doing back propagation, each node in a layer calculates it's gradient with respect to the effects of the gradients in the layer before it: if the adjustments to the layers before it is small, then adjustments to the current layer will be even smaller.

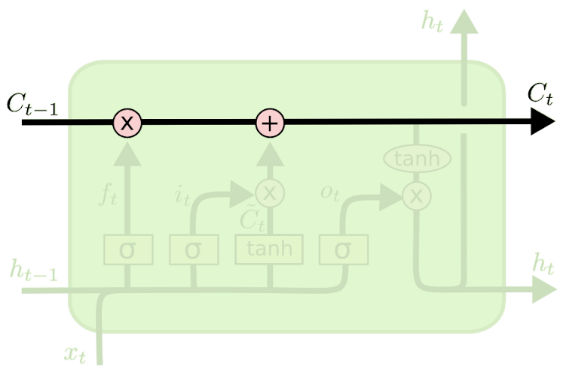
That causes gradients to exponentially shrink as it back propagates down. The earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients. And that's the vanishing gradient problem.

You can think of each time step in a recurrent neural network as a layer. To train a recurrent neural network, you use an application of back-propagation called back-propagation through time. The gradient values will exponentially shrink as it propagates through each time step.

LSTMs, a special kind of RNN, solve this problem thanks to their ability of learning long-term dependencies.
All recurrent neural networks have the form of a chain of repeating modules of neural network. The only difference between classic RNNs is the structure of the repeating module: instead of having a single neural network layer, there are four, interacting in a very special way.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers.
Lines merging denote vector concatenation, while a line forking denote its content being copied and the copies going to different locations.



The core behind LSTM is the **cell state**, the horizontal line running through the top of the diagram. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

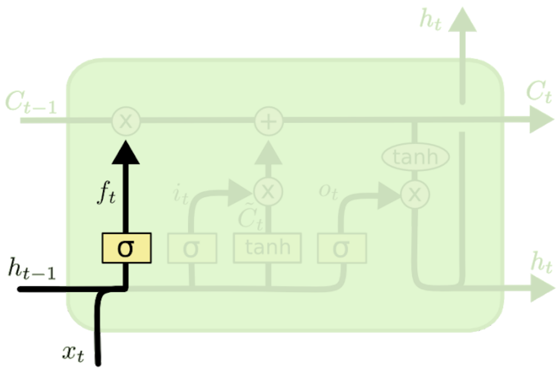
The LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation; sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "do not transmit", while a value of one means "transmit everything!".

Instead, to overcome the vanishing gradient problem, we need a function whose second derivative can sustain for a long range before going to zero. *tanh* is a suitable function with the above property.

An LSTM has three of these gates, to protect and control the cell state (in the following equations, W_k and b_k are weights and biases for layer k):

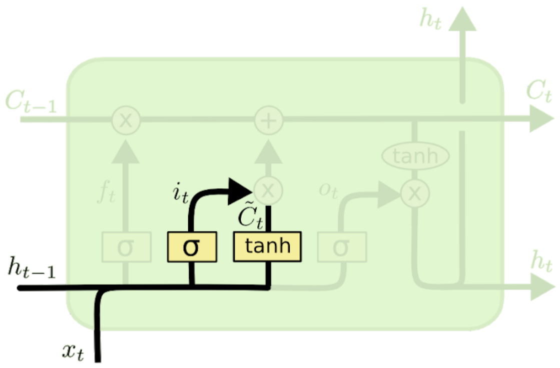
1. **Forget Gate**: this gate decides what information should be thrown away or kept. Information from the previous hidden state h_{t-1} and information from the current input x_t is passed through the sigmoid function.
   f_t values (one for each number in cell state C_{t-1}) come out between 0 and 1: the closer to 0 means to forget, and the closer to 1 means to keep.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

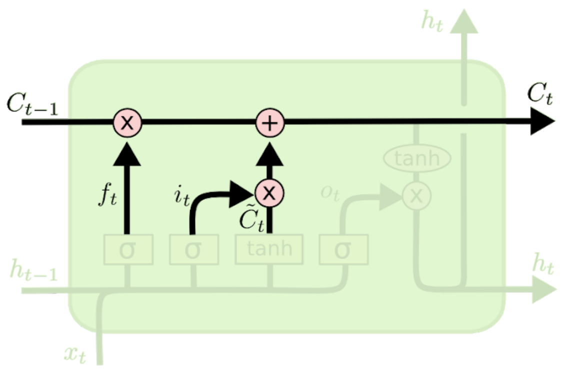
   The next step is to decide what new information we're going to store in the cell state C_t.

2. **Input Gate**: to update the cell state, it requires two steps. First, we pass the previous hidden state h_{t-1} and current input x_t into a sigmoid layer. The result i_t decides which values will be updated by transforming the values to be between 0 and 1 (0 means not important, and 1 means important).
   Next, we also pass the hidden state h_{t-1} and current input x_t into the tanh layer to squish values between -1 and 1, creating a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
   Then we multiply the sigmoid output with the tanh output ($i_t * \tilde{C}_t$): the first will decide which information is important to keep from the second.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

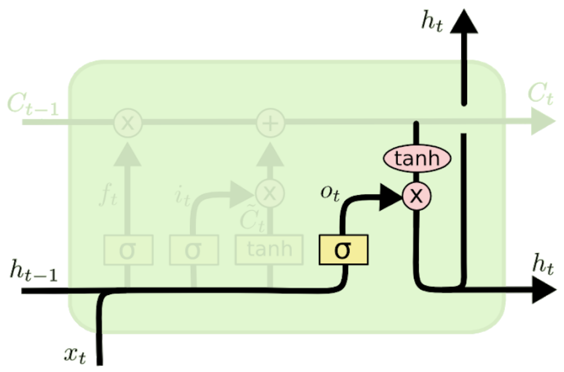
   Now we should have enough information to calculate the new cell state C_t.
   First, the old cell state C_{t-1} gets pointwise multiplied by the forget vector f_t , forgetting the things we decided to forget earlier ($f_t * C_{t-1}$).
   Then we take the output from the input gate $i_t * \tilde{C}_t$ and do a pointwise addition ($f_t * C_{t-1} + i_t * \tilde{C}_t$) which updates the cell state to new values that the neural network finds relevant, giving us our new cell state C_t.



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

Finally, we need to decide what we're going to output.

3. **Output Gate**: the output of the cell will be based on our cell state C_t, but will be a filtered version. First, we pass the previous hidden state h_{t-1} and current input x_t into a sigmoid layer which decides what parts of the cell state we're going to output (o_t). Then, we put the cell state C_t through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate o_t, so that we only output the parts we decided to. So $h_t = o_t * tanh(C_t)$.



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

3.3 ConvLSTM

This model represents a combination of LSTM model and the CNN. The main difference between this model and an LSTM model is that even if the layers are recurrent layers, internal matrix multiplications are done using convolution, directly as part of reading input into the LSTM units themselves.

Like for LSTMs, ConvLSTM supports multiple parallel input sequences for multivariate inputs.
In the model structure the CNN is as an encoder to learn features from sub-sequences of input data which are provided as time steps to an LSTM.
This is how the cell of a layer is made:

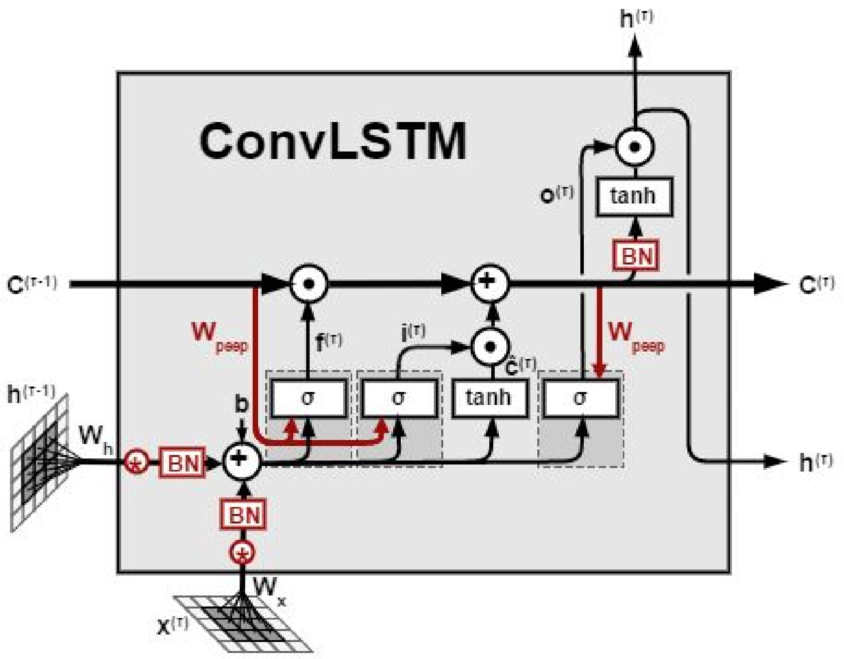


Fig. Cell of ConvLSTM[13]

[13] https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

4. Data preprocessing

In this section we are going to discuss all the data pre-processing and manipulation that we have to do to work with this kind of time series.
Mainly we will discuss three different manipulation for the three different model that we have developed. This is because, for example the VAR model requires some analysis that the other model doesn't require.
For this reason this chapter will be divided in:

- General data preprocessing
- VAR data preprocessing
- LSTM and ConvLSTM data preprocessing
- LSTM with demographic data preprocessing

To develop our solution we have decided to use Google Colab, which has helped us to work sharing easily all code and comments on the solution proposed.

4.1 General preprocessing

4.1.1 Merge

The general structure of dataframe that we used to make forecasting was built merging together the datasets previously described.
The first operation we did was a merge between datasets based on date, as reported in this code:

```
national_data = pd.merge(national_covid, national_mobility_google, on='data')
national_data = pd.merge(national_data, national_mobility_apple, on='data')
```

where ***national_covid*** is the dataset about COVID-19 daily and total cases for recovered, swabs, deaths and positives; ***national_mobility_google*** is the dataset with mobility data provided by Google and ***national_mobility_apple*** is the dataset with mobility provided by Apple.

4.1.2 Features selection and null values drop

From this dataframe we dropped some columns and saved just the one that we wanted to analyse, that are:

```
['nuovi_positivi', 'tamponi', 'grocery_and_pharmacy',
'retail_and_recreation', 'parks', 'transit_stations', 'workplaces',
'residential', 'driving', 'transit', 'walking']
```

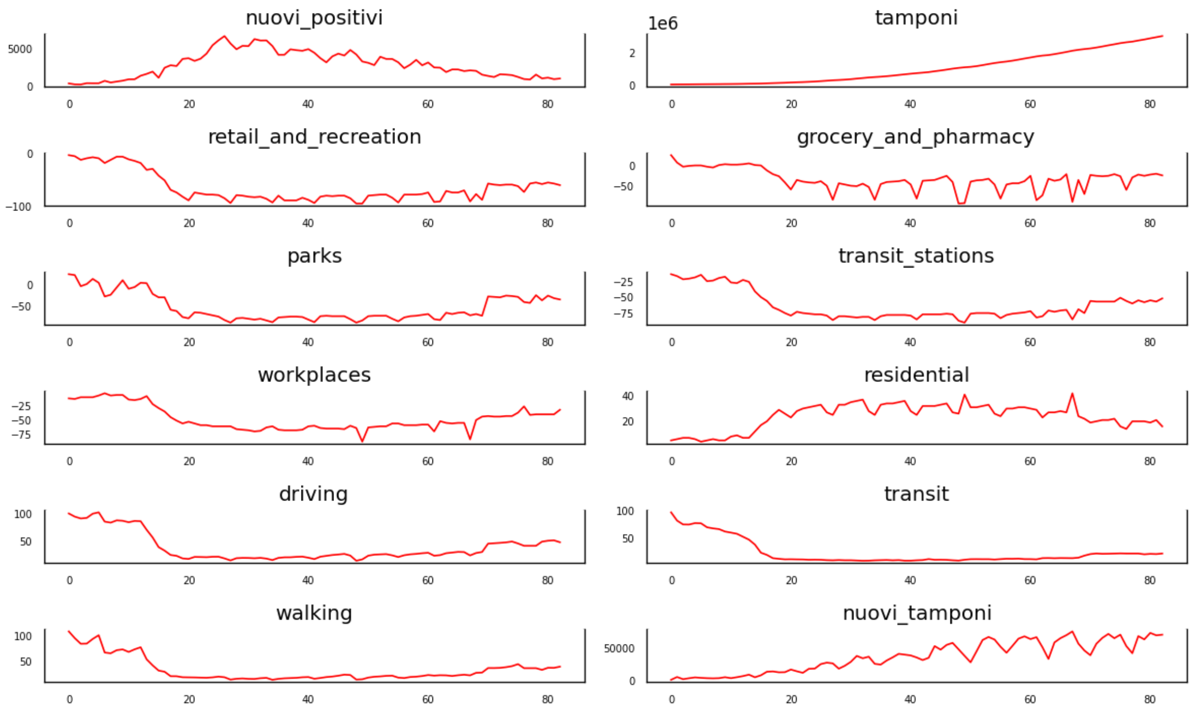
Then we created a new column, *'nuovi_tamponi'*, that represents the daily number of swabs. This is important because the daily number of swabs strongly affects the daily number of positive cases detected. This column was created thanks to the ***diff()*** function:

```
national_data['nuovi_tamponi'] = national_data['tamponi'].diff()
```

After this, we handled null-values calling ***fillna()*** method and using as parameter ***method='ffill'***: forward-fill propagates the last observed non-null value forward until another non-null value is encountered**.**

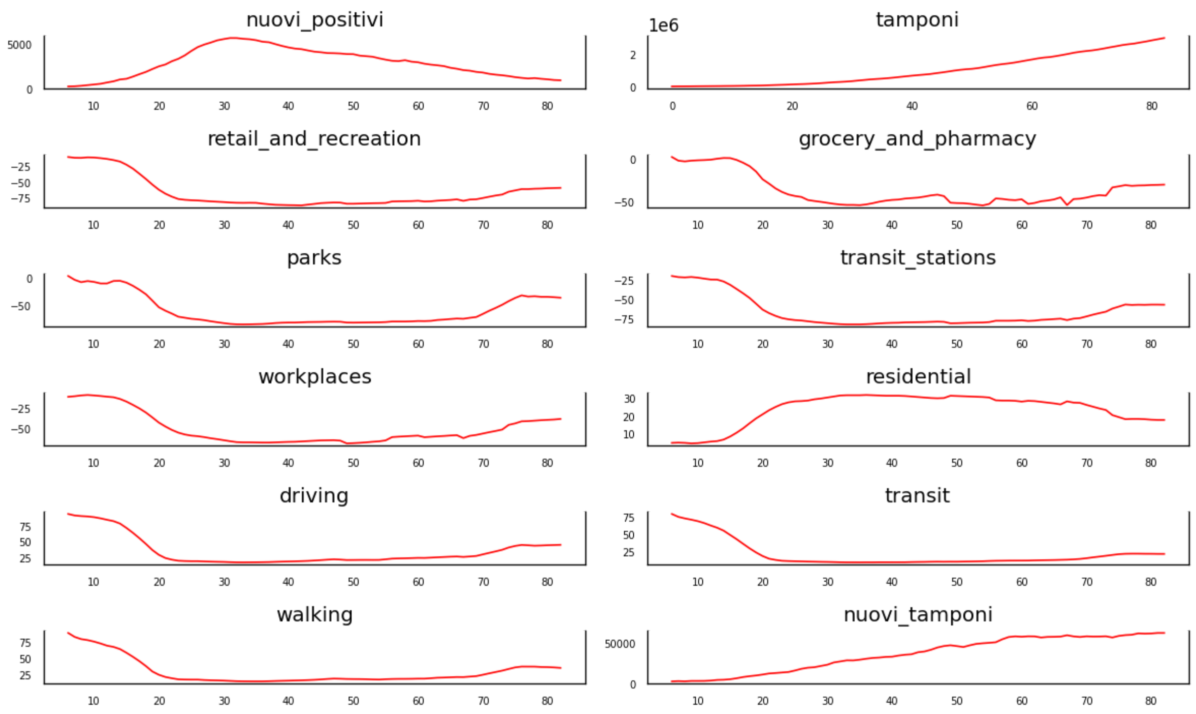
```
national_data = national_data.fillna(method = 'ffill')
```

After these operations, our time series are:



4.1.3 Bias

We realized that our time series were affected by a bias found every 7 days, probably due to the weekend data, when people stay at home and movements are less than during the week.
To get rid of this bias, we used the method of moving average. This is used to analyze time-series by calculating averages of different subsets of the complete dataset. This allow us to get smoothed data and reduce the anomalies.
Smoothing was computed for each time series taking the mean over a window using the ***rolling()*** function, with ***window*** parameter equals to **7 days**.
After this operation, our time series are:



In our work, we compared predictions for both smoothed and non-smoothed time series.

4.1.4 Normalization

Another important step was to normalize data. This is a necessary operation because when we have to work with multiple series and each one has its own range of values is necessary to normalize in a common way so that all the values are in a generic range, useful for the model to try to figure out the pattern that create the relationship between all the variables. We used a **MaxMinScaler** in [0,1] to scale the values as follows:

```
values = national_data.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
```

This scaler takes each time series and scale all the values between 0 and 1 based on the the min and max of the series itself.

4.2 VAR data pre-processing

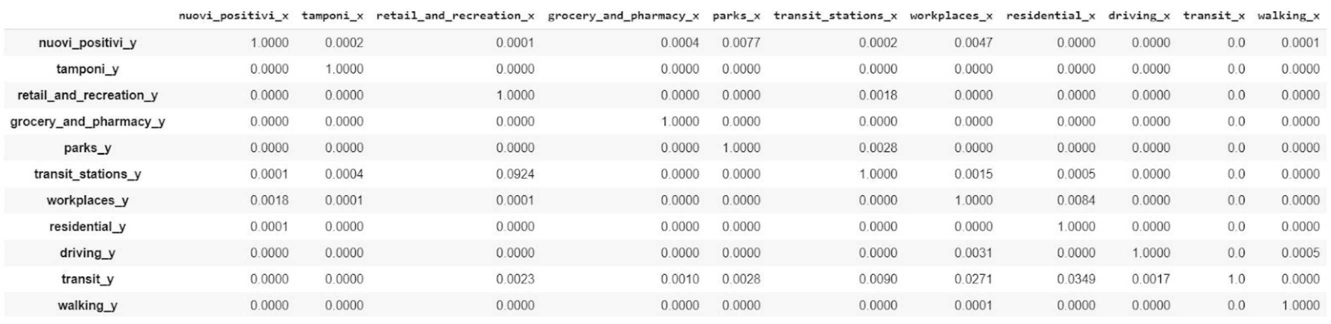
As we said before, the VAR model tries to figure out which is the pattern between all the variables that we are considering; but to make VAR works properly, we need all series to be stationary.
The first data analysis we have to do is to check if the main basis of **Vector AutoRegression** is satisfied: checking if there is a relationship between the different time series.

To do that we can use the **Granger's Causality Test**: this test verifies the null hypothesis that the past values of time series do not cause the other series. The result of this test is the p-value. If it is less than 0.05 we can reject the null hypothesis for which a series does not cause on other.

We need to do this test for all the series to see the relationship between them all.
The code below allowed us to check this property[14]:

```
def grangers_causation_matrix(data, variables, test='ssr_chi2test', verbose=False):
    df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)
    for c in df.columns:
        for r in df.index:
            test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
            min_p_value = np.min(p_values)
            df.loc[r, c] = min_p_value
    df.columns = [var + '_x' for var in variables]
    df.index = [var + '_y' for var in variables]
    return df
```

The result that we got is:



	nuovi_positivi_x	tamponi_x	retail_and_recreation_x	grocery_and_pharmacy_x	parks_x	transit_stations_x	workplaces_x	residential_x	driving_x	transit_x	walking_x
nuovi_positivi_y	1.0000	0.0002	0.0001	0.0004	0.0077	0.0002	0.0047	0.0000	0.0000	0.0	0.0001
tamponi_y	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0	0.0000
retail_and_recreation_y	0.0000	0.0000	1.0000	0.0000	0.0000	0.0018	0.0000	0.0000	0.0000	0.0	0.0000
grocery_and_pharmacy_y	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0	0.0000
parks_y	0.0000	0.0000	0.0000	0.0000	1.0000	0.0028	0.0000	0.0000	0.0000	0.0	0.0000
transit_stations_y	0.0001	0.0004	0.0924	0.0000	0.0000	1.0000	0.0015	0.0005	0.0000	0.0	0.0000
workplaces_y	0.0018	0.0001	0.0001	0.0000	0.0000	0.0000	1.0000	0.0084	0.0000	0.0	0.0000
residential_y	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0	0.0000
driving_y	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0031	0.0000	1.0000	0.0	0.0005
transit_y	0.0000	0.0000	0.0023	0.0010	0.0028	0.0090	0.0271	0.0349	0.0017	1.0	0.0000
walking_y	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0000	0.0	1.0000

As we can see each time series is compared with all the other, like in the above tabular notation. We can notice that p-value of *transit_station_y* and *retail_and_recreation_x*, is higher than 0.05: this means that for this two series there is no correlation. But for us this doesn't create any problem because we are sure that all the time series influence the 'nuovi_positivi', as we can see from the table where all the p-values are less than 0.05.

Now the last check to do is about stationarity, an important aspect for time series forecasting. If there a series is not stationary we cannot make any future assumption.
To check the stationarity we used **ADF Test**: this test was suggested better than the others on all the references that we found, so we decided to use it too.

[14] https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

This operation tests that the null hypothesis that a unit root is present, where unit root is a feature of some stochastic process that can cause problems in statistical inference[15].
The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary. In Python this test can be done by importing ***adfuller*** from ***statsmodel package***.

```
def adfuller_test(series, signif=0.05, name='', verbose=False):
    r = adfuller(series, autolag='AIC')
    output = {'test_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
    p_value = output['pvalue']

    # Print Summary
    print(f'Result ADF Test on "{name}"', "\n")

    if p_value <= signif:
        print(f" => Series is Stationary.")
    else:
        print(f" => Series is Non-Stationary.")
```

Here we are performing the ADF test and getting back the p-value: if this value is less than the significance standard of 0.05, we reject the null-hypothesis otherwise we accept it, meaning that the series is non stationary.

It turns out that some series are not stationary, and to make them so we use the ***differencing method***: differencing can help stabilize the mean of the time series by removing changes in the level of a time series, and so eliminating (or reducing) trend and seasonality[16].
Differencing is performed by subtracting the previous observation from the current observation.

difference(t) = observation(t) - observation(t-1)

We could apply this operation thanks to ***diff()*** function.

```
df_differenced = df_train.diff().dropna()
```

After two differencing operations we got not all the time series stationary so, we decided to apply the *MinMaxScaler*.

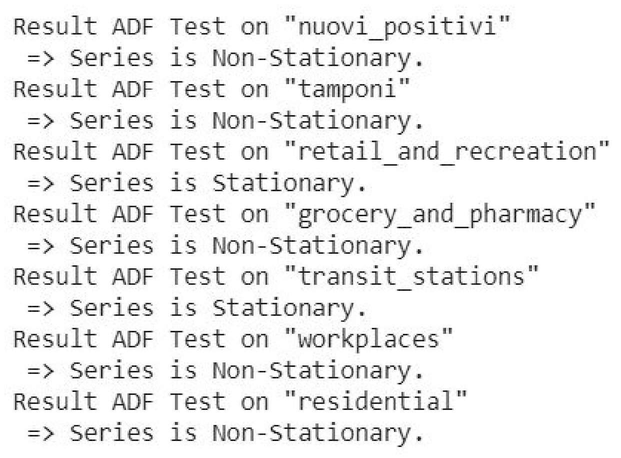
```
test_frec = 0.25
n_test = round((len(dataset_var)) * test_frec)
df_train, df_test = dataset_var[0:-n_test], dataset_var[-n_test:]


scaled_var = scaler_var.fit_transform(df_train.values)
df_train = pd.DataFrame(data=scaled_var,    # values
              index=df_train.index,    # 1st column as index
              columns=df_train.columns)
df_train
```

We applied the *MinMaxScaler* only on the training set because predictions returned by the model are converted back to original range and compared with the test set, not scaled.

[15] https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test
[16] https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

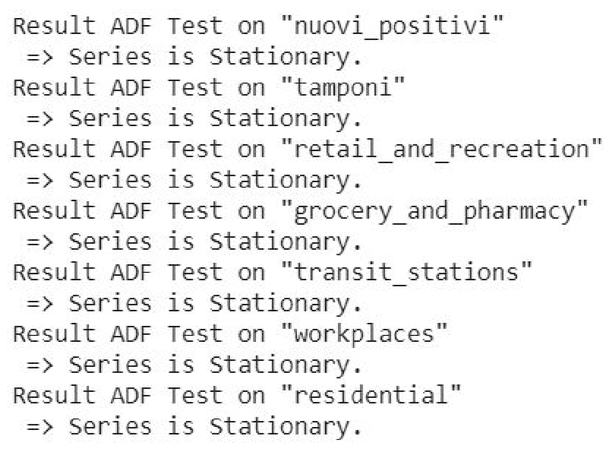
In this case, repeating the same operations described before provides all series to be stationary except for *'parks*', *'driving'*, *'transit'* and *'walking'* features, still non-stationary: we decided to drop them.

This is the result of stationarity test at start and at the end:



```
Result ADF Test on "nuovi_positivi"
 => Series is Non-Stationary.
Result ADF Test on "tamponi"
 => Series is Non-Stationary.
Result ADF Test on "retail_and_recreation"
 => Series is Stationary.
Result ADF Test on "grocery_and_pharmacy"
 => Series is Non-Stationary.
Result ADF Test on "transit_stations"
 => Series is Stationary.
Result ADF Test on "workplaces"
 => Series is Non-Stationary.
Result ADF Test on "residential"
 => Series is Non-Stationary.
```

Stationarity before differencing



```
Result ADF Test on "nuovi_positivi"
 => Series is Stationary.
Result ADF Test on "tamponi"
 => Series is Stationary.
Result ADF Test on "retail_and_recreation"
 => Series is Stationary.
Result ADF Test on "grocery_and_pharmacy"
 => Series is Stationary.
Result ADF Test on "transit_stations"
 => Series is Stationary.
Result ADF Test on "workplaces"
 => Series is Stationary.
Result ADF Test on "residential"
 => Series is Stationary.
```

Stationarity after 2 differentiations

4.3 LSTM and ConvLSTM data preprocessing

The first step is to prepare our dataset for the LSTM; in this case our dataset is built on a national-basis.
This involves framing the dataset as a supervised learning problem, modeled as predicting number of infected people in future days given measurements and mobility data at previous days.

The dataset is transformed in order to have ***N*** input variables (input series) and ***one*** output variable (number of infections). The input variables are represented by mobility data and infected people in the previous ***k*** days, and the output variable represents the number of infected people in the following day.

	var1(t-1)	**var2(t-1)**	**var3(t-1)**	**var4(t-1)**	**var5(t-1)**	**var6(t-1)**	**var7(t-1)**	**var8(t-1)**	**var1(t)**
0	0.12977	0.35294	0.24590	0.52727	0.66666	0.00229	0.0	0.0	0.14889
1	0.14889	0.36764	0.24590	0.52727	0.66666	0.00381	0.0	0.0	0.15996
2	0.15996	0.42647	0.22950	0.54545	0.66666	0.00533	0.0	0.0	0.18209
3	0.18209	0.48529	0.22950	0.56363	0.66666	0.00839	0.03703	0.0	0.13883
4	0.13883	0.48529	0.22950	0.56363	0.66666	0.00991	0.07407	0.0	0.10965

In this example we want to predict the future *(t)* value of *var1* given its previous *(t-1)* values and other variables *(var2, ... , var8)*.
This means we have 8 input features *(var1(t-1)*, …, *var8(t-1)*) and one output feature (*var1(t)*). In this case the number of previous days (temporal window *k*) to analyze is just one *(t-1)*.

Each row of this new dataset is a **sample**, composed by one **timestep** in the past and 8 **features**, giving one value as output for the next timestep.
As you can notice, the *var1(t)* value for the first row corresponds to *var1(t-1)* value of the following row: samples follow a sequential ordering.

At this point, we must split the prepared dataset into train and test sets, and then split again the train and test sets into input (*var1(t-1), …, var8(t-1)*) and output variables (*var1(t)*).

Train input:

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var8(t-1)
0	0.12977	0.35294	0.24590	0.52727	0.66666	0.00229	0.0	0.0
1	0.14889	0.36764	0.24590	0.52727	0.66666	0.00381	0.0	0.0
2	0.15996	0.42647	0.22950	0.54545	0.66666	0.00533	0.0	0.0

Train output:

	var1(t)
0	0.14889
1	0.15996
2	0.18209

Test input:

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var8(t-1)
3	0.18209	0.48529	0.22950	0.56363	0.66666	0.00839	0.03703	0.0
4	0.13883	0.48529	0.22950	0.56363	0.66666	0.00991	0.07407	0.0

Test output (groundtruth):

	var1(t)
3	0.13883
4	0.10965

Finally, the inputs are reshaped into the 3D format expected by LSTM, namely **[samples, timesteps, features]**.

For ConvLSTM, input shape must be 5D:
[samples, subsequences, channels, timesteps, features]

In this case, we used the complete dataframe:

```
['nuovi_positivi', 'nuovi_tamponi', 'grocery_and_pharmacy',
'retail_and_recreation', 'parks', 'transit_stations', 'workplaces',
'residential', 'driving', 'transit', 'walking']
```

In both LSTM and ConvLSTM we compared results from the smoothed time series (without the 7-days bias) and the raw ones (no smooth).

4.4 LSTM data with demographic info preprocessing

In this solution the time series dataset is built on a region-based approach, so that we have time series specific for each Italian region.
Time series is processed in the same way before, but now we have demographic data in addition: this data encodes population, density of population and number of hospital beds for each specific region.

To represent these information, we added a column to our previous transformed dataset containing the list of demographic features of the region to which time series data corresponds to.

	var1(t-1)	**var2(t-1)**	**var3(t-1)**	**var4(t-1)**	**var5(t-1)**	**...**	**demo_info**	**var1(t)**
0	0.12977	0.35294	0.24590	0.52727	0.66666	...	[pop, dens, hosp]	0.14889
1	0.14889	0.36764	0.24590	0.52727	0.66666	...	[pop, dens, hosp]	0.15996
2	0.15996	0.42647	0.22950	0.54545	0.66666	...	[pop, dens, hosp]	0.18209
3	0.18209	0.48529	0.22950	0.56363	0.66666	...	[pop, dens, hosp]	0.13883
4	0.13883	0.48529	0.22950	0.56363	0.66666	...	[pop, dens, hosp]	0.10965

These scalar informations are passed through the network from a different input layer with respect to time series, then are analyzed and concatenated together in order to predict the future target value.

In this scenario, our dataframe is:

```
['nuovi_positivi', 'nuovi_tamponi', 'retail_and_recreation',
'grocery_and_pharmacy', 'parks', 'transit_stations', 'workplaces',
'residential', 'abitanti', 'densità', 'posti_letto']
```

Also in this case we used both non smoothed and smoothed time series.

5. Train and predict

5.1 VAR

After we processed and analysed the data in the previous step, now we can fit the model. The VAR model can be imported from the statsmodels in python:

```
from statsmodels.tsa.api import VAR
```

After that, we need to pass the data to the VAR model:

```
model = VAR(df_differenced)
```

The next step is the lag order selection and to fit the model.
Lag selection methods allows us to find the best lag order to minimize the (out-of-sample) forecast error, where for lag we mean the number of past timesteps of the series the model must consider during the fitting process. In our case we find out that the best value was 2.

In fit operation we can set based on which criterion it has to select the best value of the lag. In this case we fixed the maxlangs equals to four, and the criterion used is the **Akaike information criterion** (AIC)[17]. "*Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models*". It is used in model selection.

```
fitted = model.fit(maxlags=4, ic='aic')
```

This function returns as selected lag value 2.

At this point we can forecast. We need to specify the value from which it has to start to predict and the number of step in future to forecast.

```
forecast_input = df_differenced.values[-lag_order:]

groundtruth = dataset_var

fc = fitted.forecast(y=forecast_input, steps=n_test)

df_forecast = pd.DataFrame(fc, index=groundtruth.index[-n_test:], columns=groundtruth.columns + '_2d')
```

Now that we have our prediction, we must convert them to original format of our data, before the application of differentiation and scaling operations.
To invert the differentiating operation we need to consider how many times we have applied it. In our case is two and using this method brings correctly-scaled data back.

[17] https://en.wikipedia.org/wiki/Akaike_information_criterion

```
def invert_transformation(df_train, df_forecast, second_diff=False):
    df_fc = df_forecast.copy()
    columns = df_train.columns
    for col in columns:
        if second_diff:
            df_fc[str(col)+'_1d'] = (df_train[col].iloc[-1]-df_train[col].iloc[-2]) + df_fc[str(col)+'_2d'].cumsum()
        df_fc[str(col)+'_forecast'] = df_train[col].iloc[-1] + df_fc[str(col)+'_1d'].cumsum()
    return df_fc

df_results = invert_transformation(df_train, df_forecast, second_diff=True)
```

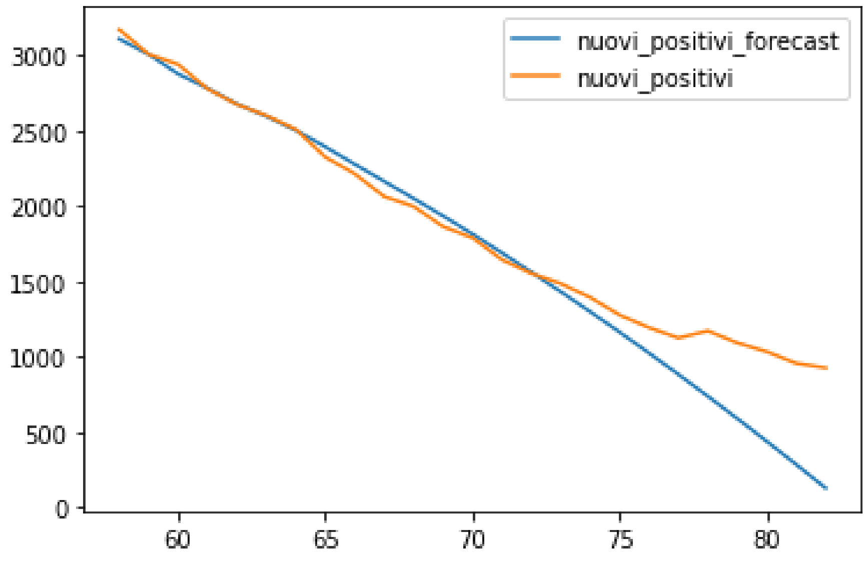
This method renames all the time series with a suffix equals to "*_forecast*": this is useful in the plotting operation and evaluation phase.

To invert the scaling, we can use the inverse operation available on the *MinMaxScaler* as follow:

```
columns_forecast = [sub + '_forecast' for sub in df_train.columns]

forecasted_df = df_results[columns_forecast]

df_results = scaler_var.inverse_transform(forecasted_df)

final_forecast =pd.DataFrame(data=df_results,
                    index=forecasted_df.index,
                    columns=forecasted_df.columns)
```

Here we are redefining a new dataframe with the structure of the original one and the value of the new data inverted.

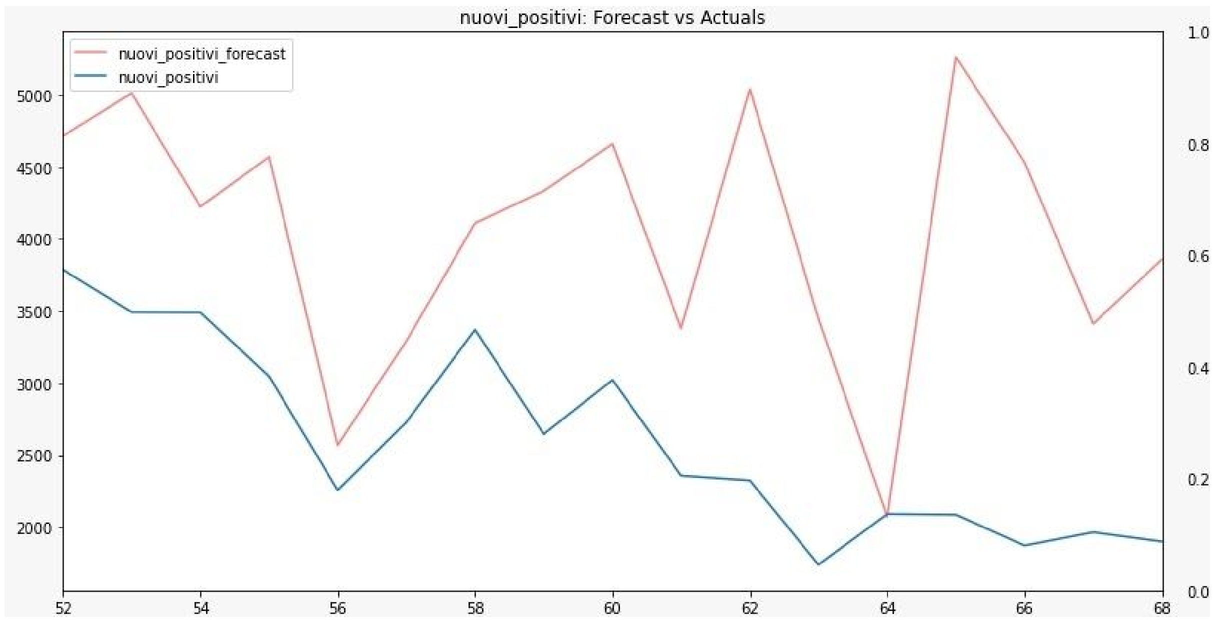
The last operation is just to plot data:



```
# calculate RMSE
rmse = sqrt(mean_squared_error(df_test['nuovi_positivi'], final_forecast['nuovi_positivi_forecast']))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 284.491
```

The curve of the *nuovi_positivi* is smoother because of the operation on moving average that we have done in preprocessing.
Differently, if we had not performed that smoothing, the result was pretty worst.



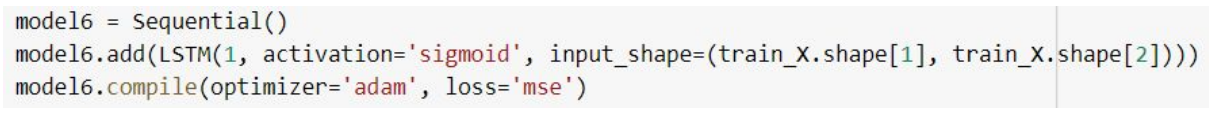
5.2 LSTM

We developed different kind of network architecture to check which returns the best predictions.

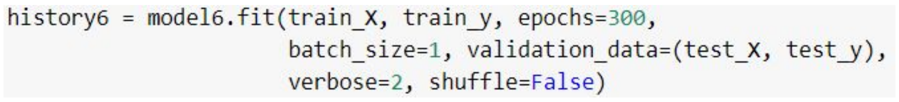
As said before, our first scenario consists in the forecast of national-based time series, instead the second is focused on the regional time series.

We used 2 different models: a simple one (**model6**) and one more complex (**model2**)

- **model6**

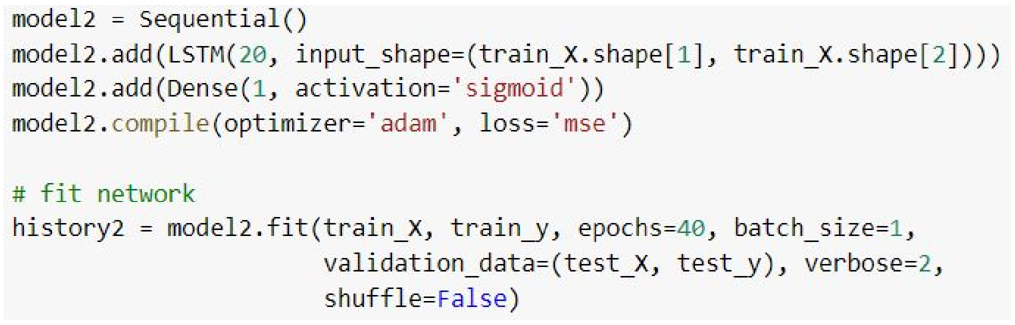


```
model6 = Sequential()
model6.add(LSTM(1, activation='sigmoid', input_shape=(train_X.shape[1], train_X.shape[2])))
model6.compile(optimizer='adam', loss='mse')
```



```
history6 = model6.fit(train_X, train_y, epochs=300,
                      batch_size=1, validation_data=(test_X, test_y),
                      verbose=2, shuffle=False)
```

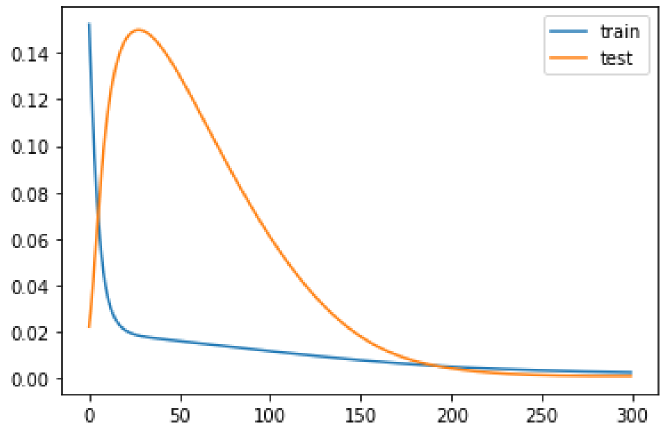
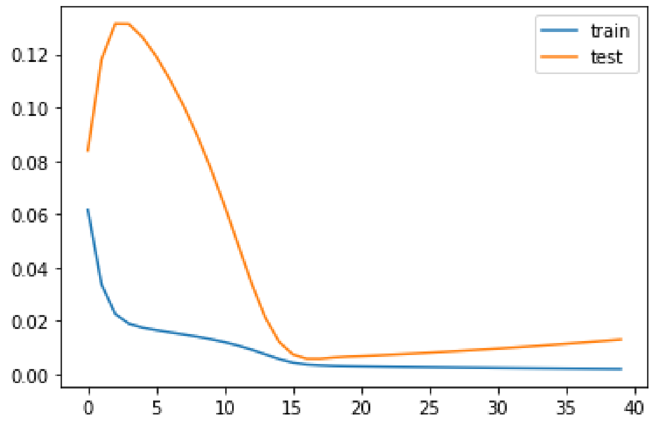
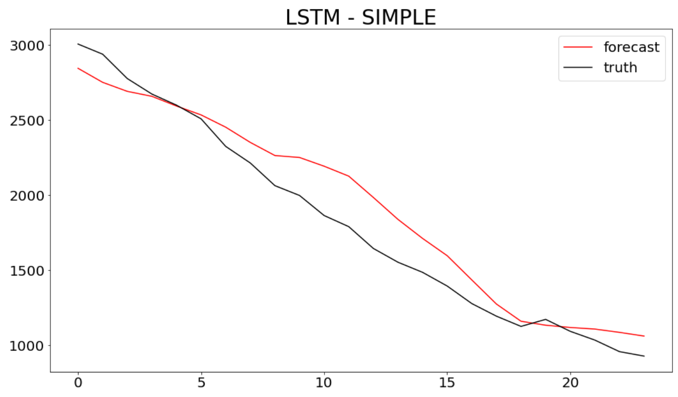
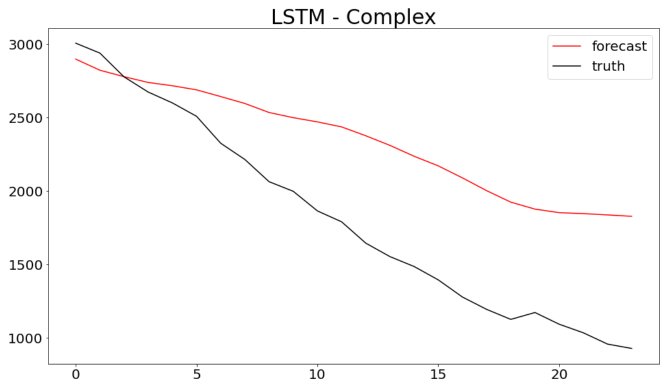
- **model2**

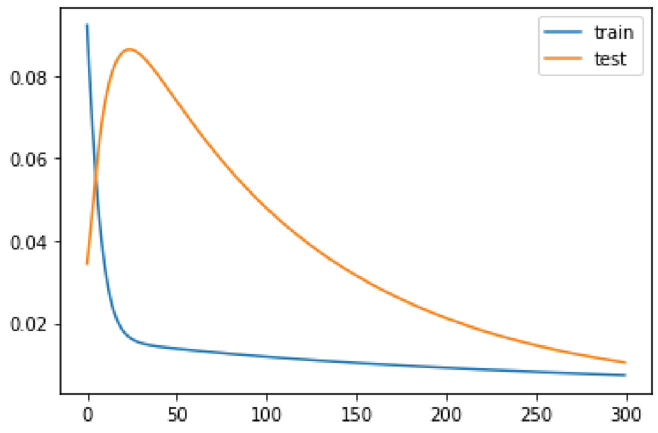
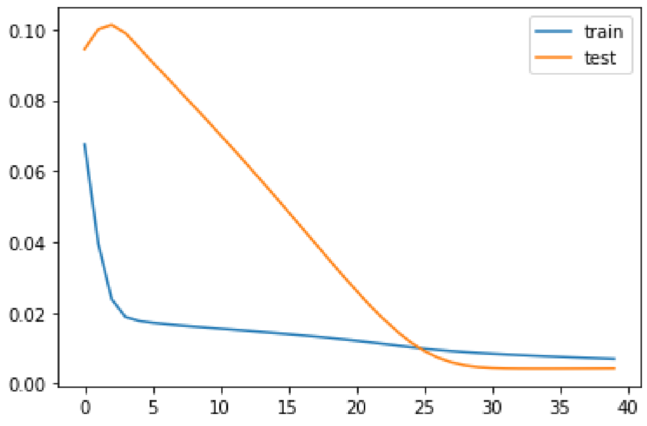
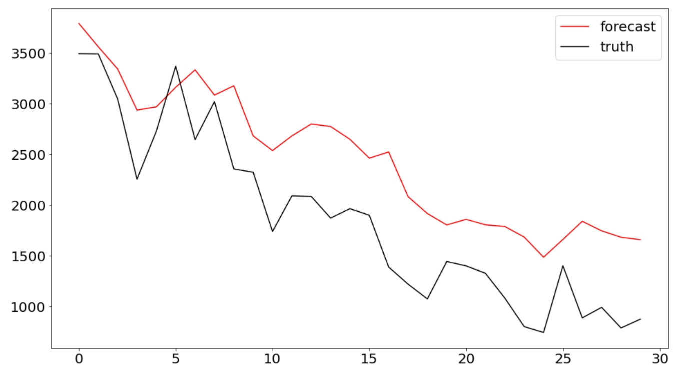
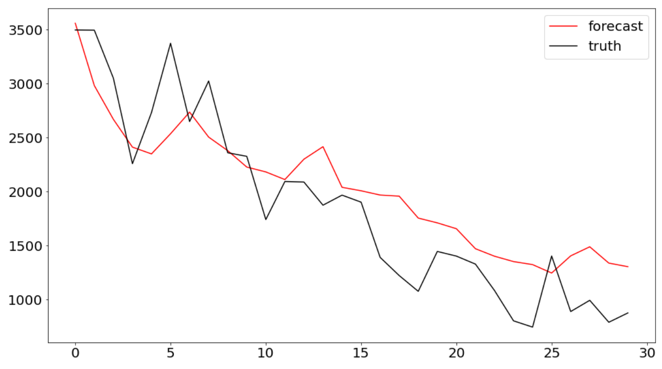


```
model2 = Sequential()
model2.add(LSTM(20, input_shape=(train_X.shape[1], train_X.shape[2])))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(optimizer='adam', loss='mse')

# fit network
history2 = model2.fit(train_X, train_y, epochs=40, batch_size=1,
                      validation_data=(test_X, test_y), verbose=2,
                      shuffle=False)
```

We also split our National and Regional analysis based on the smoothness of our time series: **smoothed** and **non-smoothed**.

5.2.1 LSTM - National predictions

SMOOTHED TIME SERIES	
MODEL6 (simpler)	**MODEL2** (more complex)
Loss over epochs 	Loss over epochs 
Test RMSE: 181.655	Test RMSE: 615.956
Forecast: 	Forecast: 

NON-SMOOTHED TIME SERIES	
MODEL6 (simpler)	**MODEL2** (more complex)
Loss over epochs 	Loss over epochs 
Test RMSE: 663.861	Test RMSE: 422.578
Forecast: 	Forecast: 

5.2.2 LSTM - Regional prediction

In this case we tried to train a model on more data with respect to the National scenario: this because we have more peculiar data, one dataset for each region.
The idea is to develop a model that takes also in account demographic informations about one region and use them to forecast more precisely.

Also in this case we used 2 different models: **model6** (simpler) and **model1** (more complex).
Both models have 2 input branches, one for time series and one for scalar data (demographic info). Then we added a Concatenation layer to put together both scalar and temporal data in order to make predictions.

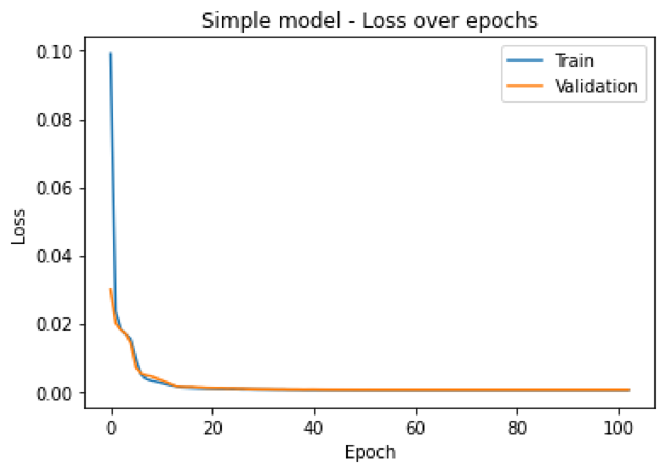
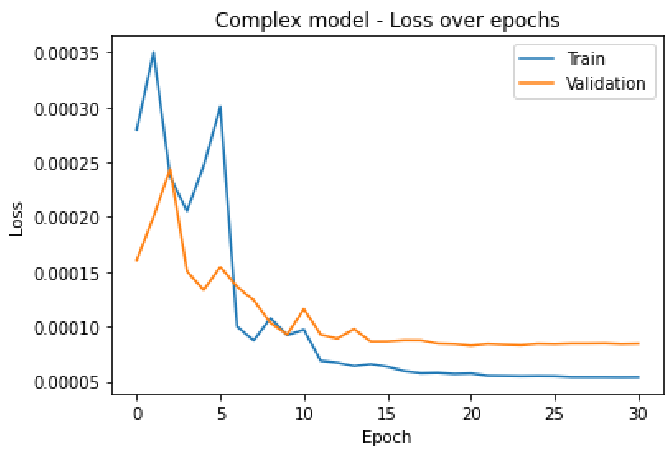
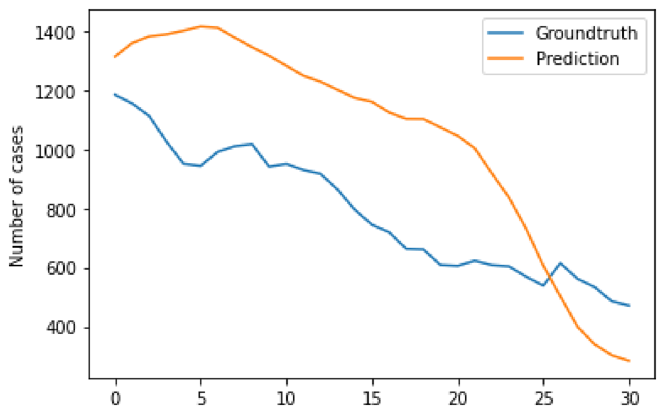
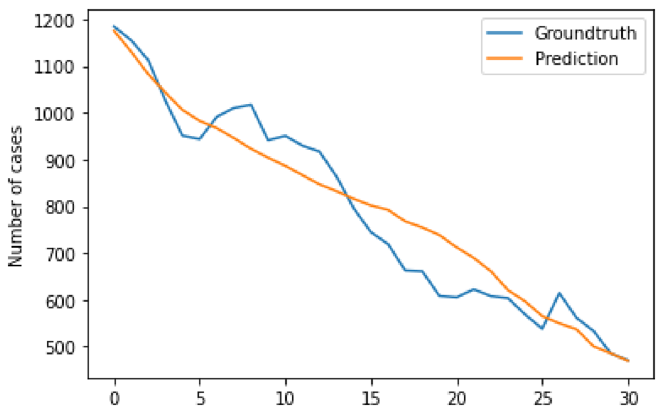
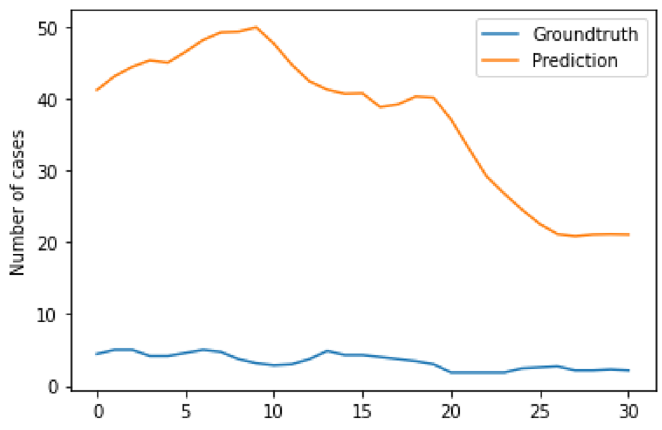
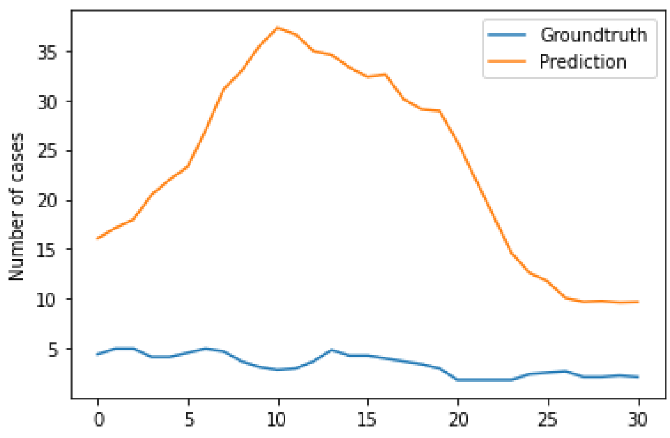
- Simple model (**model6**):

```
#time series input branch
temporal_input_layer = layers.Input(shape=(sequence_length,8))
main_rnn_layer = layers.LSTM(1)(temporal_input_layer)

#demographic input branch
demographic_input_layer = layers.Input(shape=(3,))

#concatenation
merge_c = layers.Concatenate(axis=-1)([main_rnn_layer,demographic_input_layer])

#output
nuovi_positivi = layers.Dense(1, activation='sigmoid', name="nuovi_positivi")(merge_c)

model6 = Model([temporal_input_layer,demographic_input_layer], [nuovi_positivi])
```

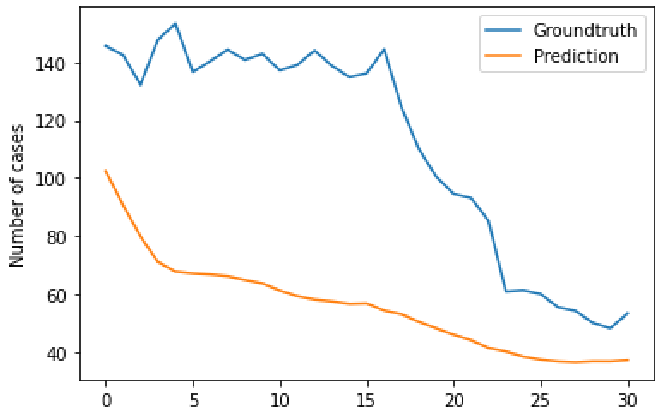
```
history6 = model6.fit([X_temporal_train,X_demographic_train], [Y_cases_train],
          epochs = 250,
          batch_size = 1,
          validation_data=([X_temporal_test,X_demographic_test],  [Y_cases_test]))
```

- More complex model (**model1**):

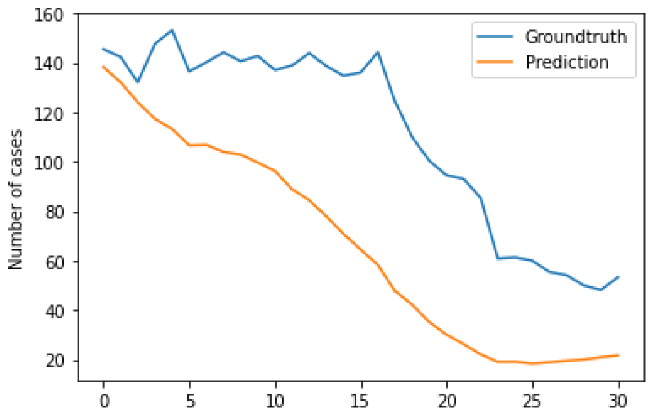
```
#time series input branch
temporal_input_layer = layers.Input(shape=(sequence_length,8))
main_rnn_layer = layers.LSTM(100, return_sequences=True)(temporal_input_layer)

#demographic input branch
demographic_input_layer = layers.Input(shape=(3,))
demographic_dense = layers.Dense(100)(demographic_input_layer)
demographic_dropout = layers.Dropout(1)(demographic_dense)

#cases output branch
rnn_c = layers.LSTM(100)(main_rnn_layer)

#concatenation
merge_c = layers.Concatenate(axis=-1)([rnn_c,demographic_dropout])

#output
dense_c = layers.Dense(100)(merge_c)
dropout_c = layers.Dropout(1)(dense_c)
nuovi_positivi = layers.Dense(1, activation= 'sigmoid', name="nuovi_positivi")(dropout_c)

model1 = Model([temporal_input_layer,demographic_input_layer], [nuovi_positivi])
```

SMOOTHED TIME SERIES	
MODEL6 (simpler)	**MODEL1** (more complex)
Loss over epochs 	Loss over epochs 
Test RMSE (mean over 20 regions): 67.574	Test RMSE (mean over 20 regions): 26.937
Forecast Lombardia: 	Forecast Lombardia: 
Forecast Umbria: 	Forecast Umbria: 

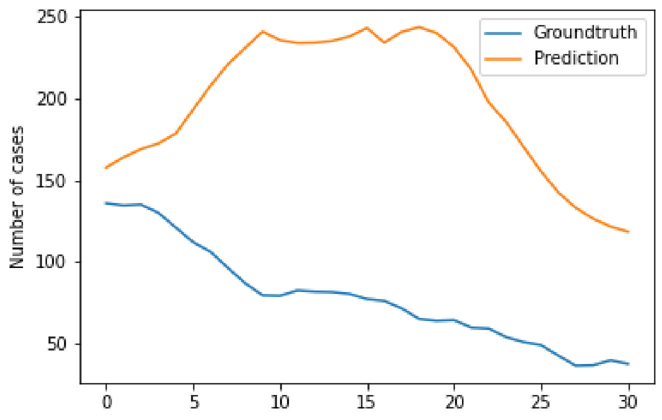
Forecast Liguria:



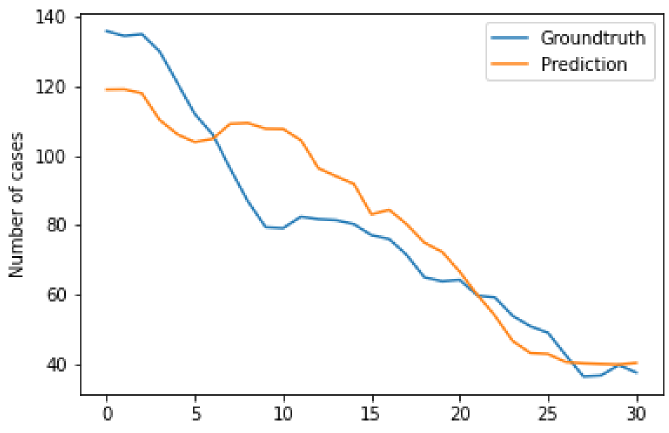
Forecast Liguria:



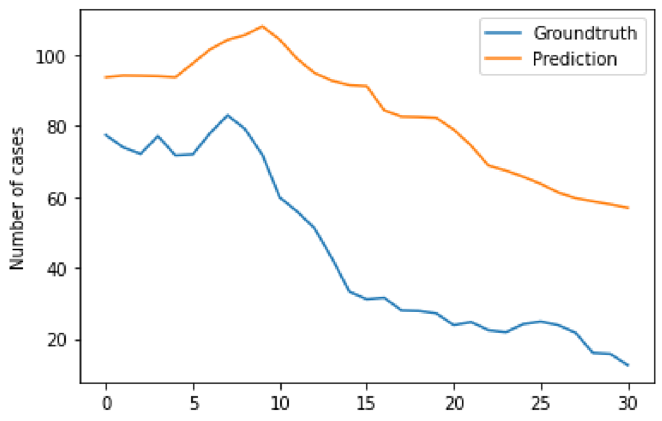
Forecast Lazio:



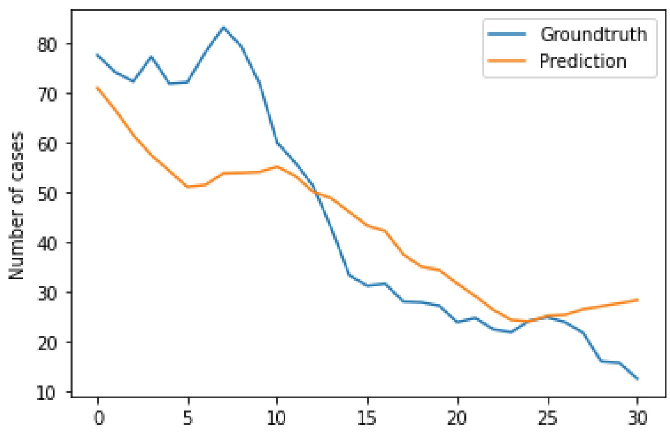
Forecast Lazio:



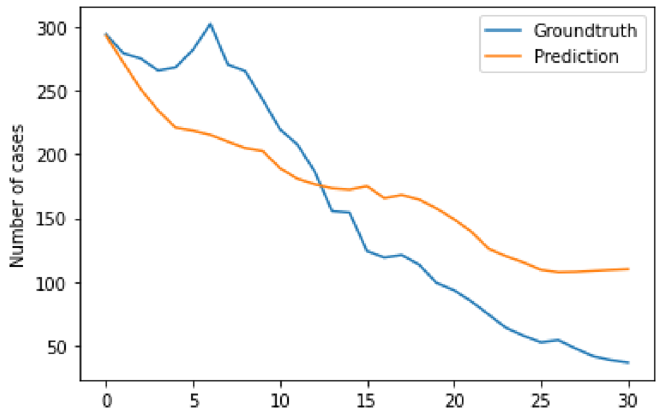
Forecast Puglia:



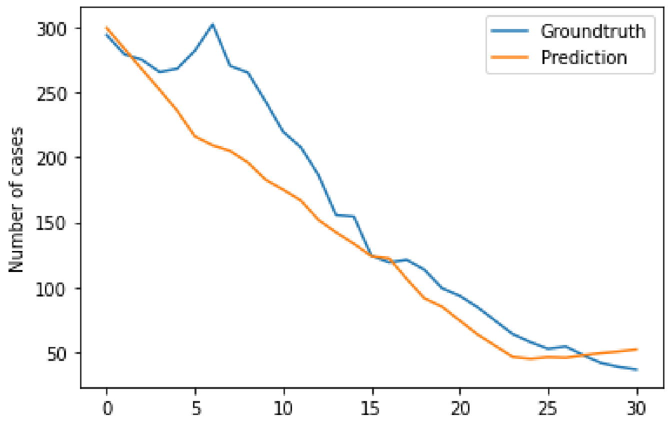
Forecast Puglia:



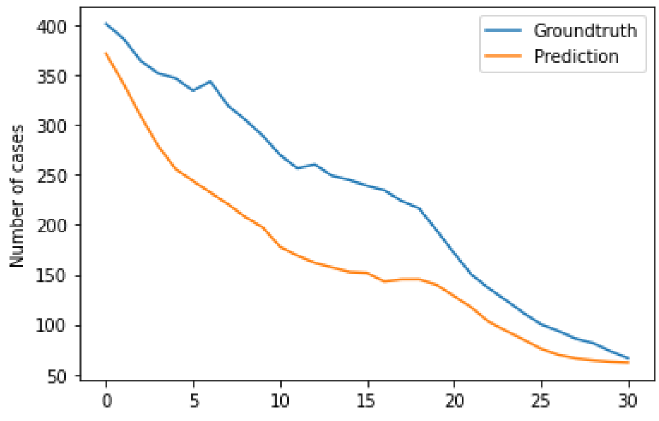
Forecast Veneto:



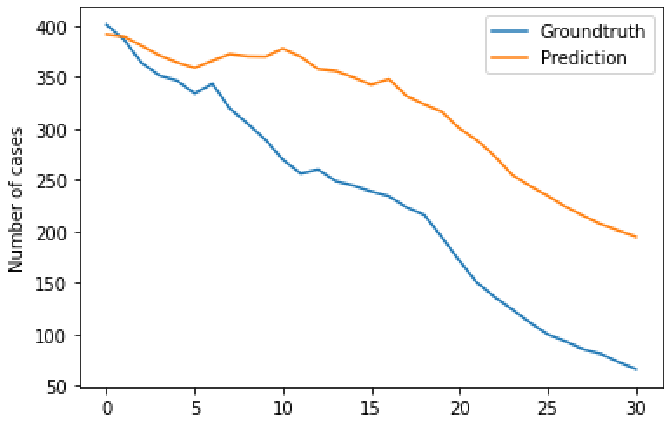
Forecast Veneto:



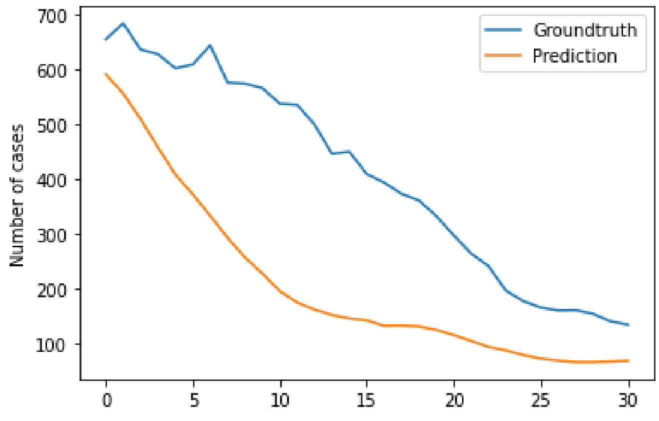
Forecast Emilia-Romagna:



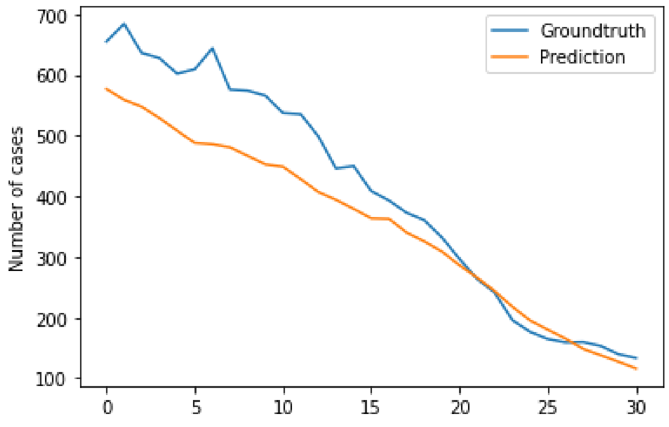
Forecast Emilia-Romagna:

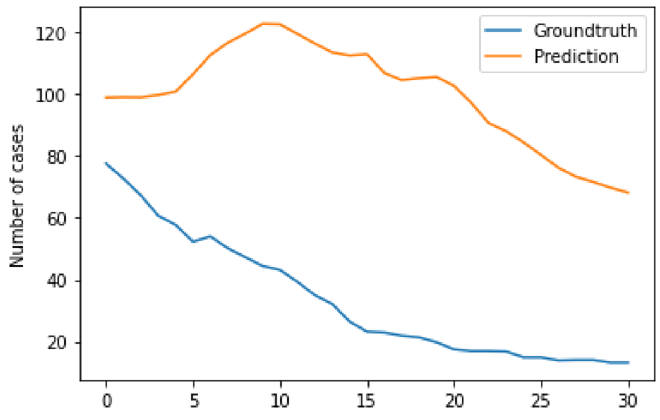
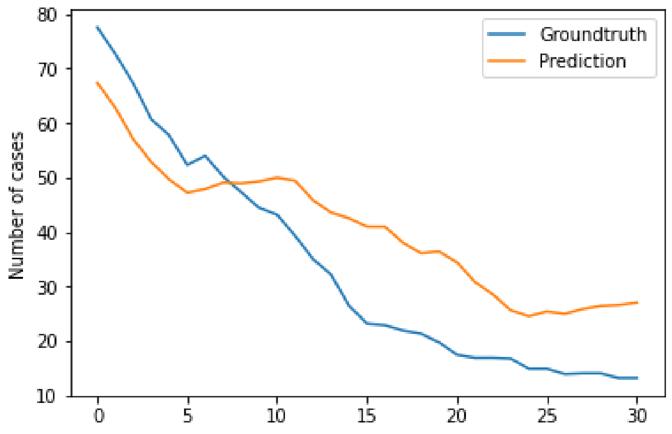
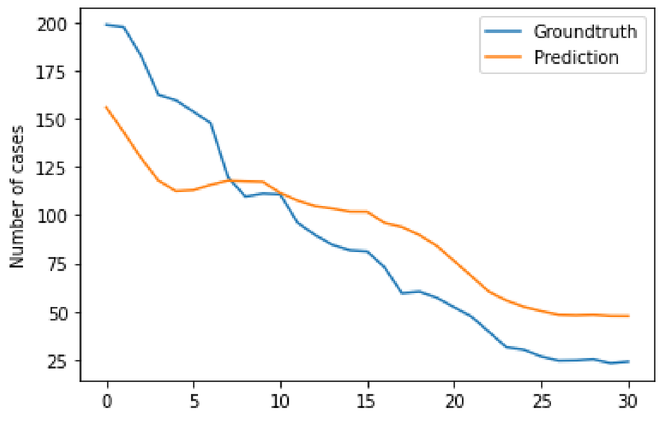
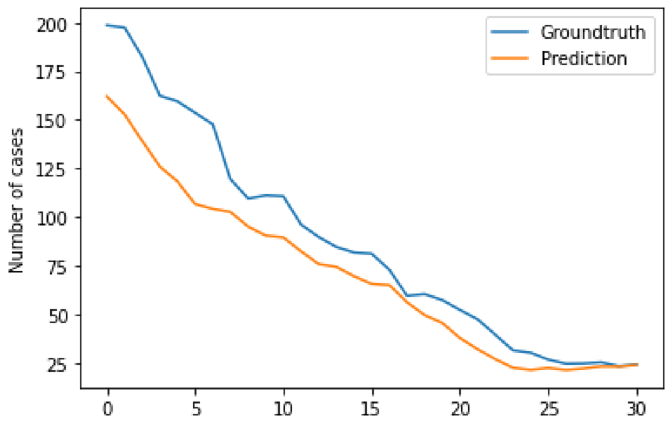
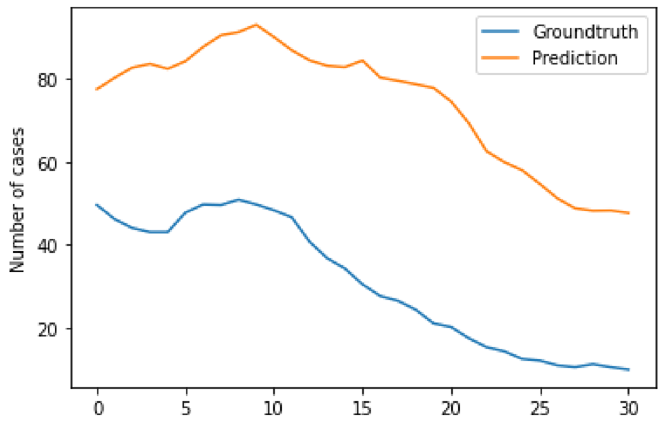
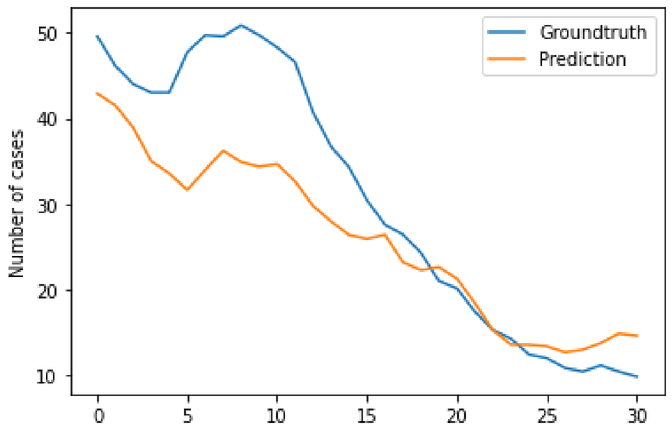


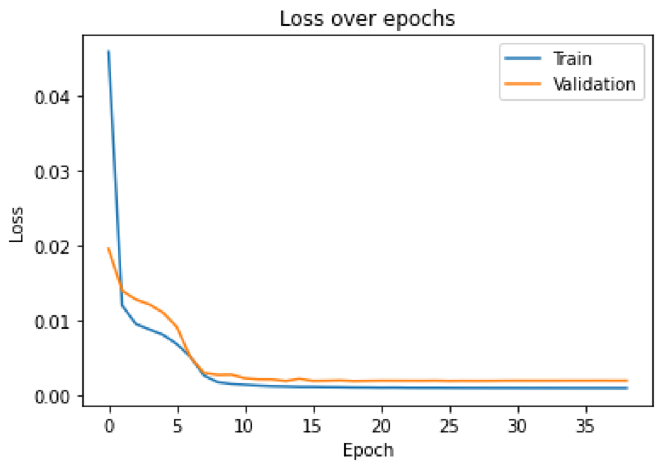
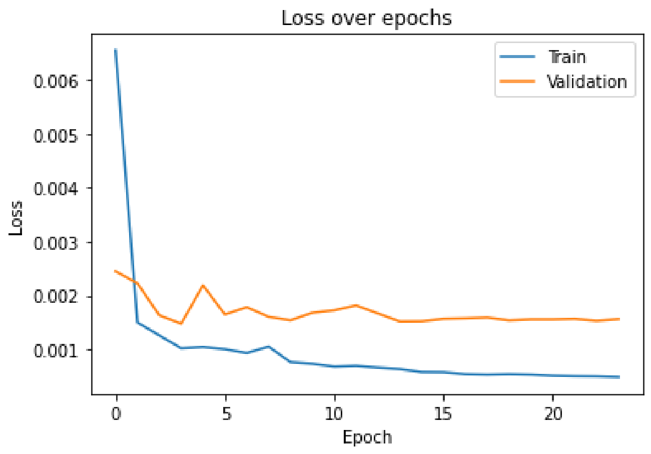
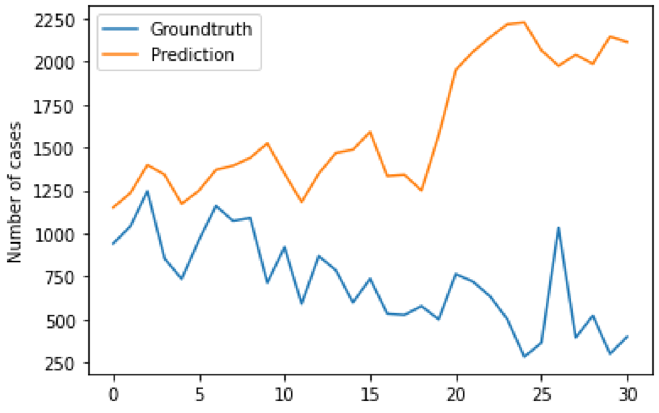
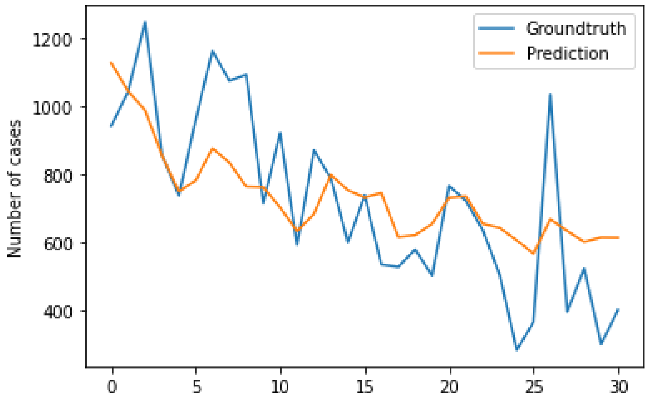
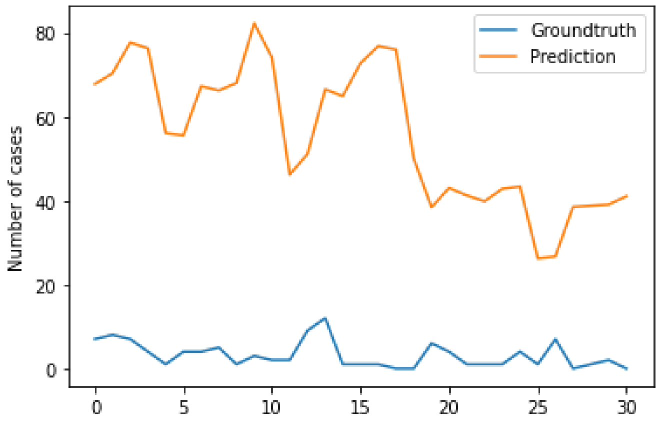
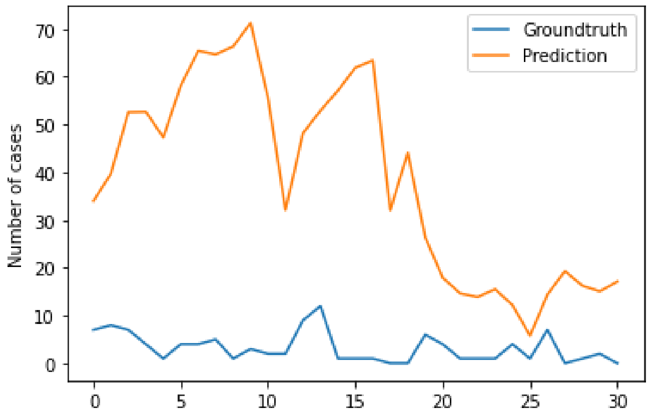
Forecast Piemonte:



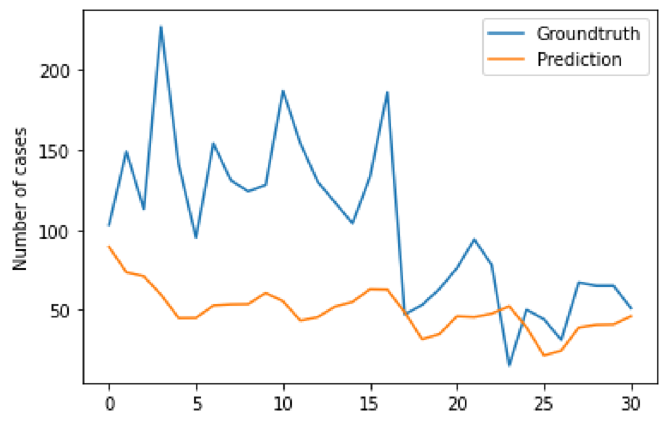
Forecast Piemonte:



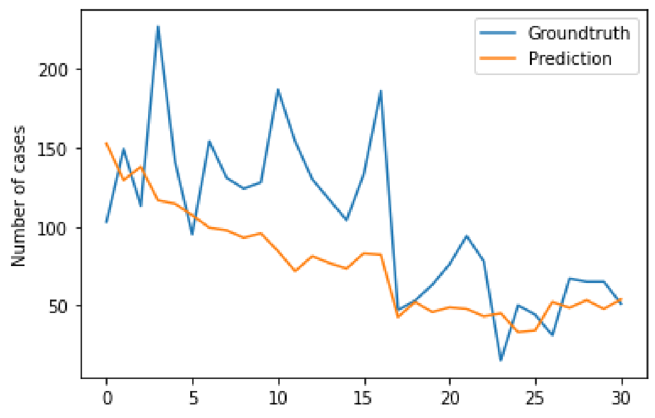
Forecast Campania: 	Forecast Campania: 
Forecast Toscana: 	Forecast Toscana: 
Forecast Sicilia: 	Forecast Sicilia: 

NON-SMOOTHED TIME SERIES	
MODEL6 (simpler)	**MODEL1** (more complex)
Loss over epochs 	Loss over epochs 
Test RMSE: 132.694	Test RMSE: 97.850
Forecast Lombardia: 	Forecast Lombardia: 
Forecast Umbria: 	Forecast Umbria: 

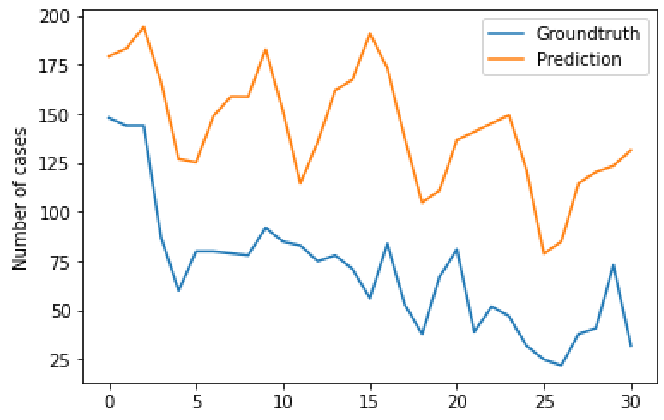
Forecast Liguria:



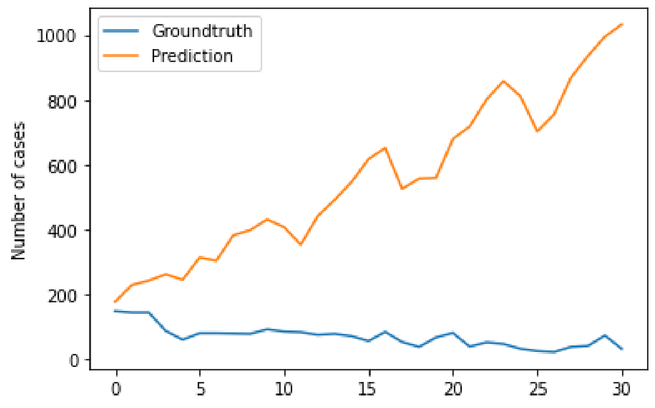
Forecast Liguria:



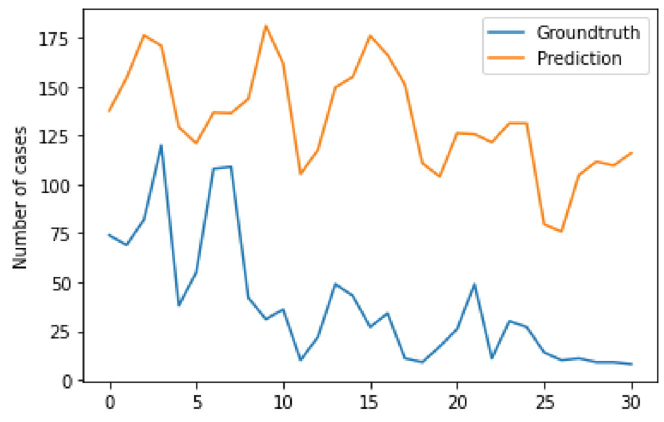
Forecast Lazio:



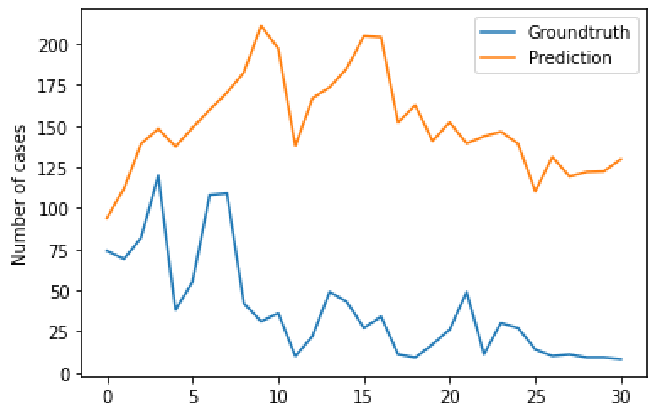
Forecast Lazio:



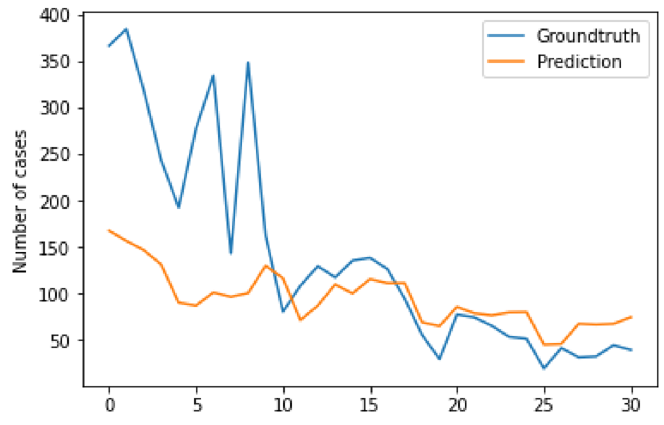
Forecast Puglia:



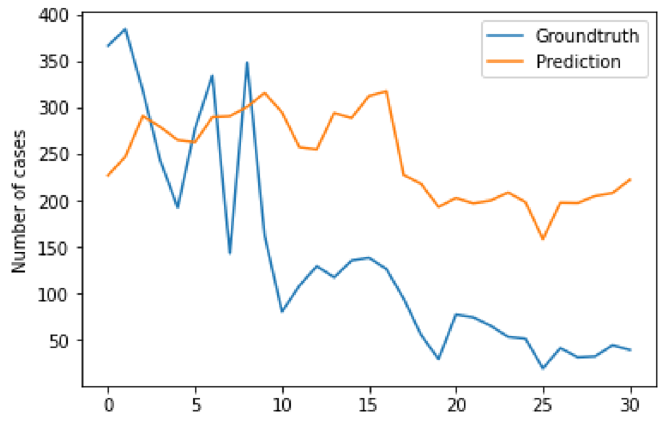
Forecast Puglia:



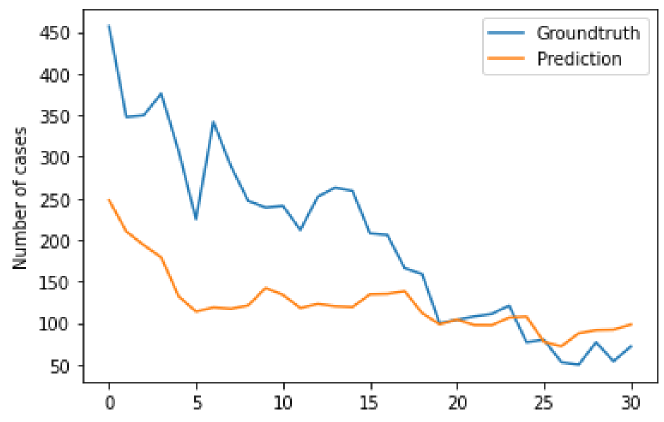
Forecast Veneto:



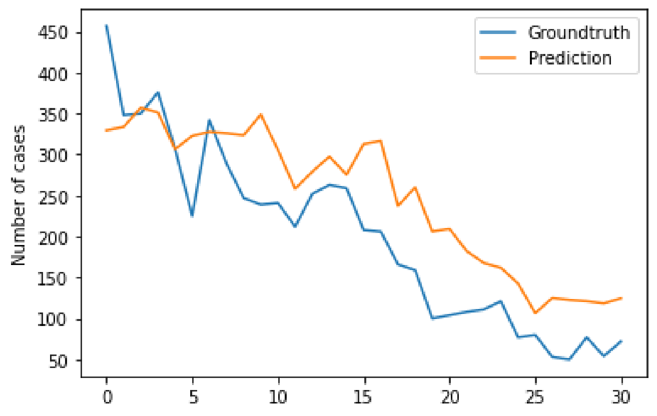
Forecast Veneto:



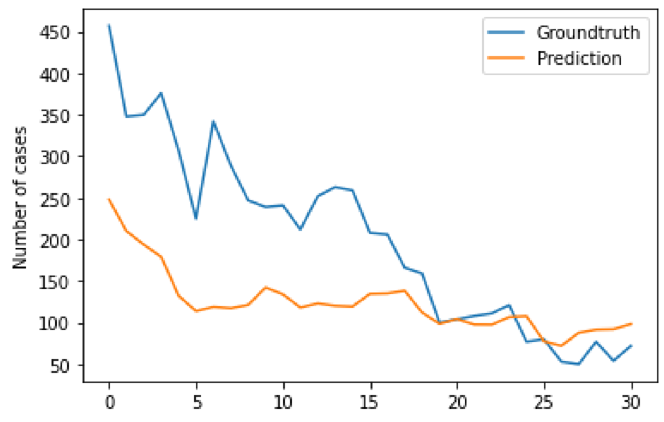
Forecast Emilia-Romagna:



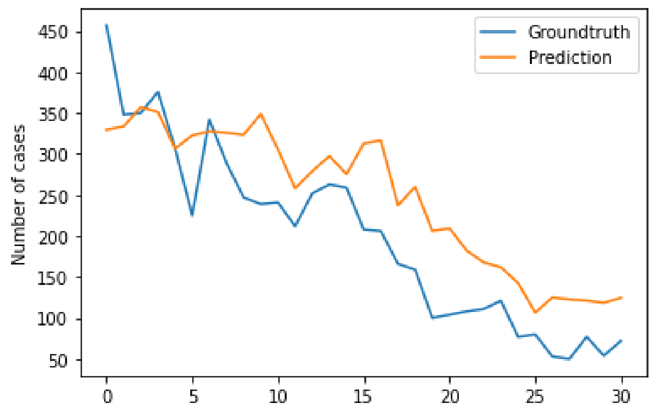
Forecast Emilia-Romagna:



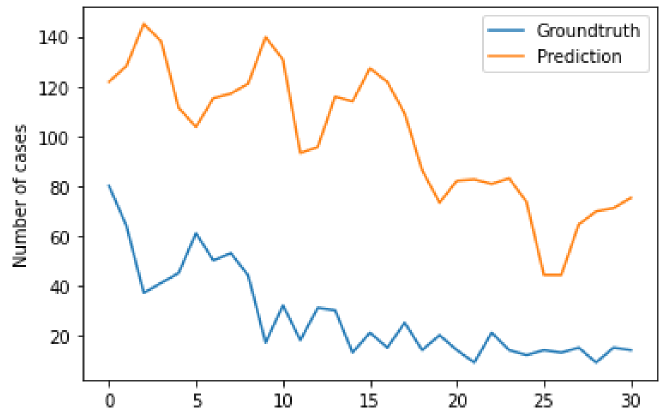
Forecast Piemonte:



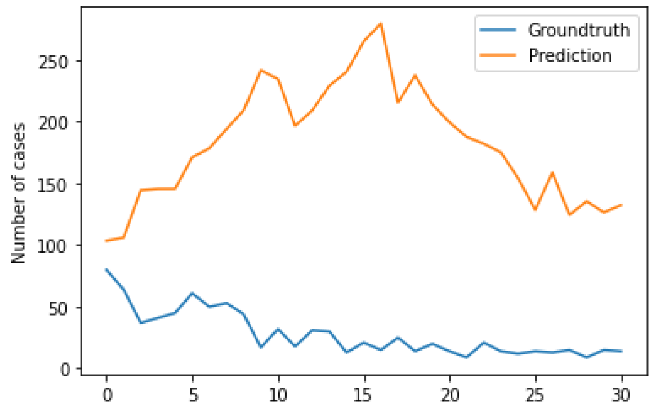
Forecast Piemonte:



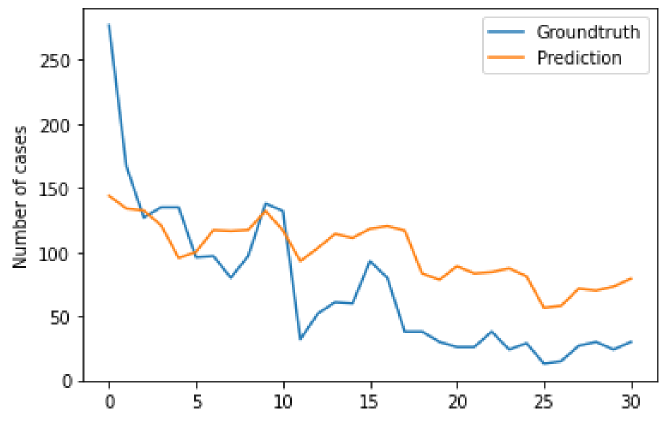
Forecast Campania:



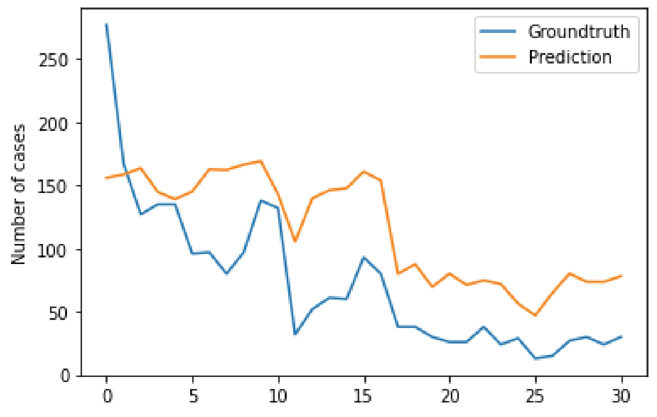
Forecast Campania:



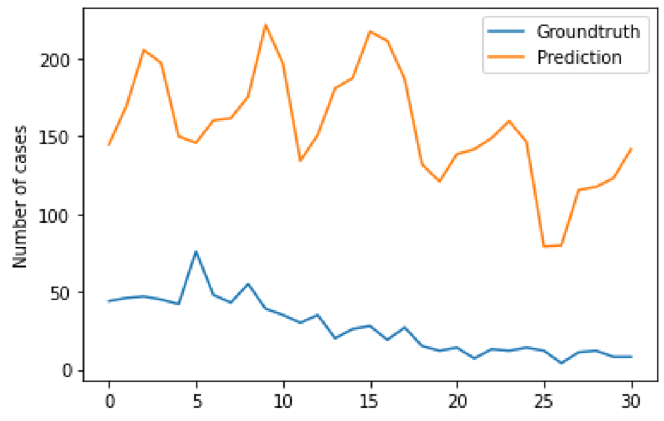
Forecast Toscana:



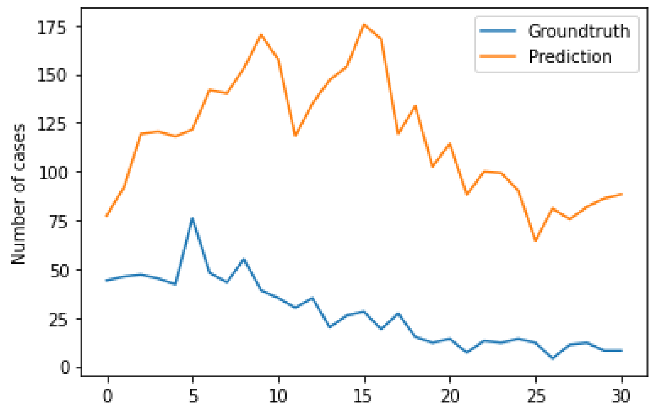
Forecast Toscana:



Forecast Sicilia:



Forecast Sicilia:

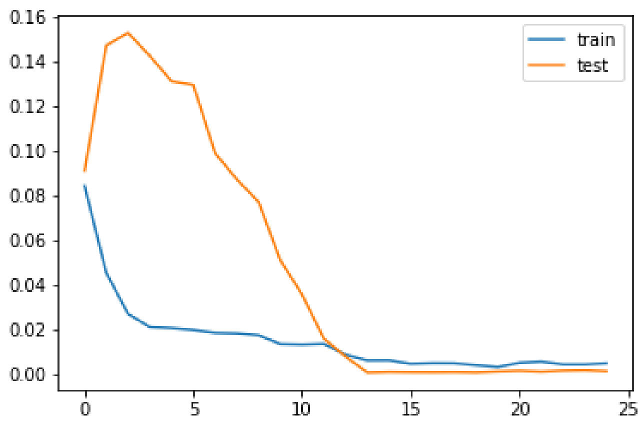
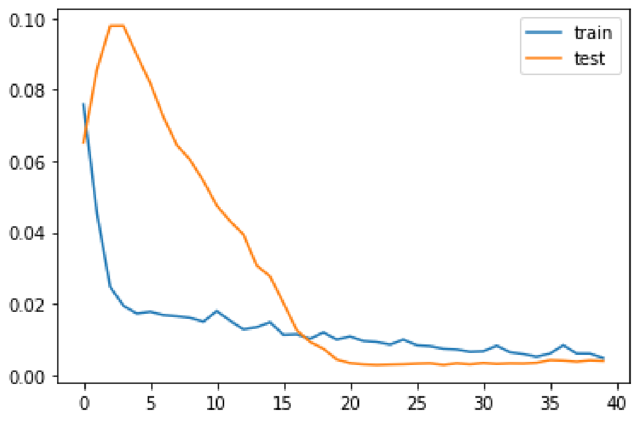
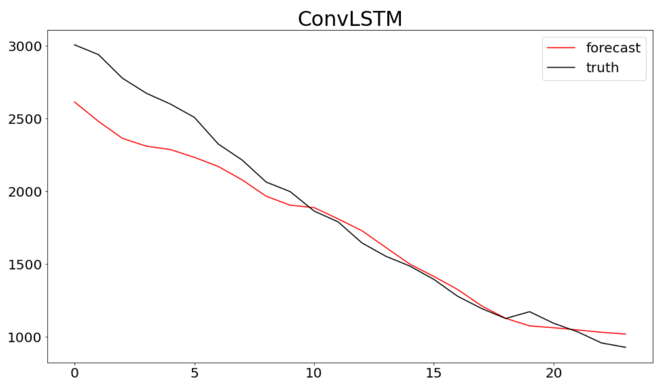
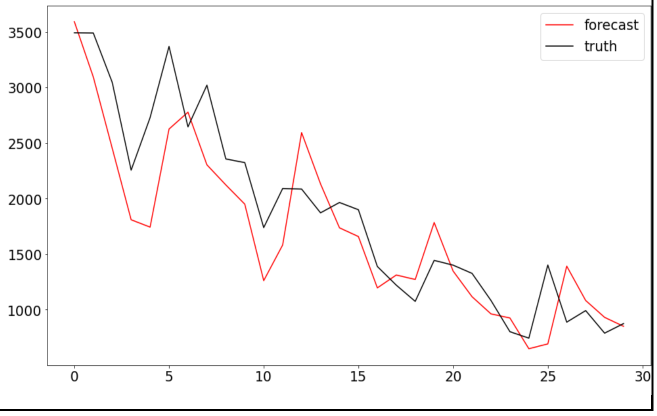


5.3 ConvLSTM

With ConvLSTM, data are pre-processed in the same way as LSTM. Also model architecture is quite similar, with the only difference of using a ConvLSTM2D layer instead of a classic LSTM one.

We used ConvLSTM in our National scenario, and after several trials with different architectures, we decided to present this one (**model6**):

```
model6 = Sequential()
model6.add(ConvLSTM2D(filters=16, kernel_size=(1,2),
                      input_shape=(n_seq, 1, n_days, n_features)))
model6.add(Flatten())
model6.add(Dense(16))
model6.add(Dropout(0.2))
model6.add(Dense(1, activation='sigmoid'))
model6.compile(optimizer='adam', loss='mse')
```

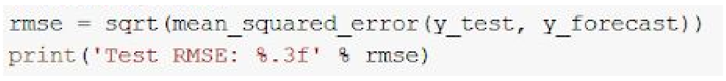
SMOOTHED TIME SERIES	**NON-SMOOTHED TIME SERIES**
Loss over epochs 	Loss over epochs 
Test RMSE: 197.863	Test RMSE: 406.724
Forecast: 	Forecast: 

6. Evaluation

Differently from others cases of study, where for evaluating results of a model we can use methods like **k-fold** to improve results generalization, in time series model this cannot be done.
This because time series must follow a temporal order and splitting or mixing data will change the meaning of the entire series.

In this case, to evaluate the model we splitted the dataset in training and test, where test data are compared to the forecasted values from the model on the same time interval, to see how much the model was able to create a pattern based on the history of the time series defined in the training set.

The evaluation method is the same for all the models, and we used the *Root Mean Squared Error* metric.



```
rmse = sqrt(mean_squared_error(y_test, y_forecast))
print('Test RMSE: %.3f' % rmse)
```

We compute the **MSE** using the *y_test* set and *y_forecast* set. The result of this operation is passed to the *sqrt* function and it returns the **RMSE**.

These are the RMSEs for each model we trained.

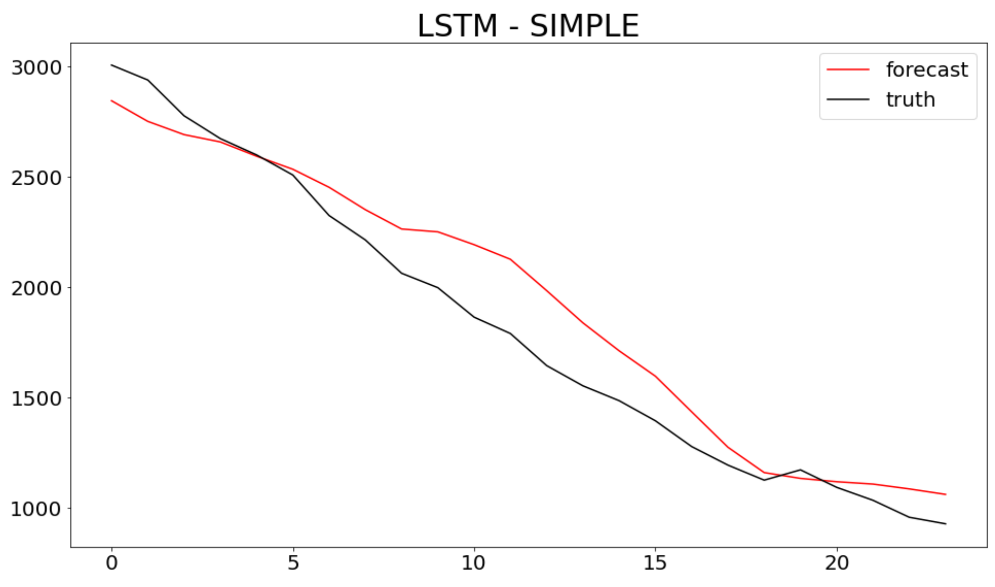
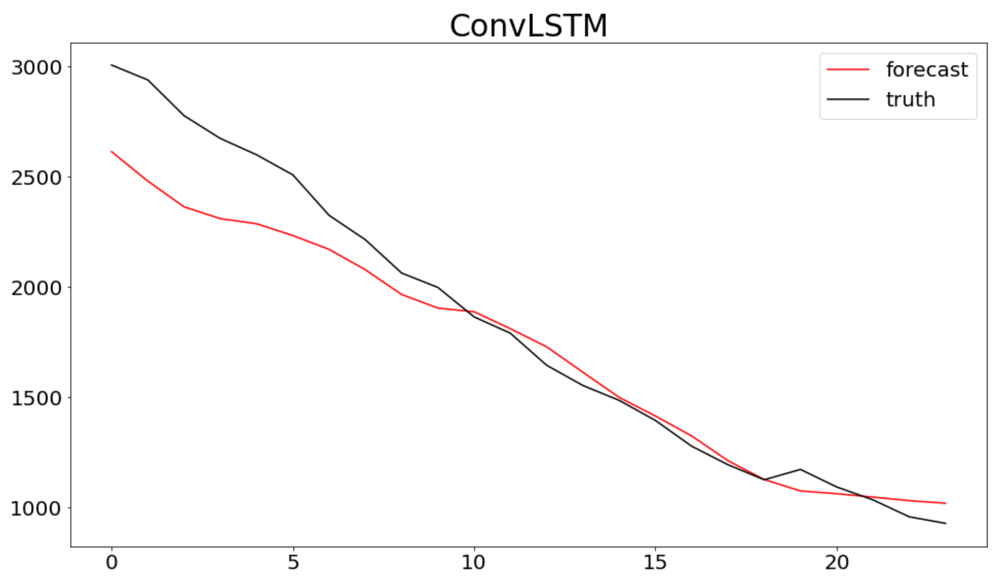
Model	Context	RMSE
VAR	National	284,491
ConvLSTM	National	197,863
LSTM	National	181,655
LSTM	Regional with demographic	26,937 (mean over 20 regions)

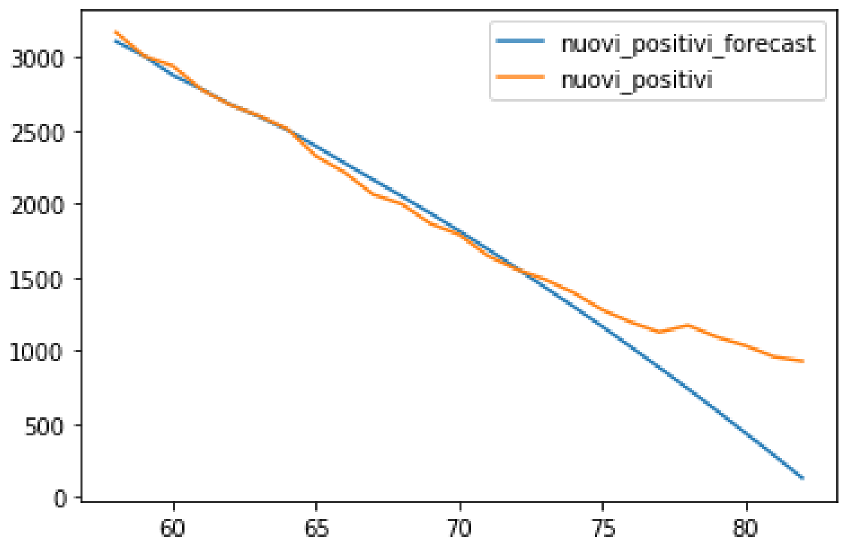
7. Conclusions

In our analysis, we decided to check whether mobility data can be used in addition to COVID-19 time series to forecast the epidemic.
We compared different algorithms and different kind of data in order to make predictions about the epidemic and evaluate which model returns the best ones.

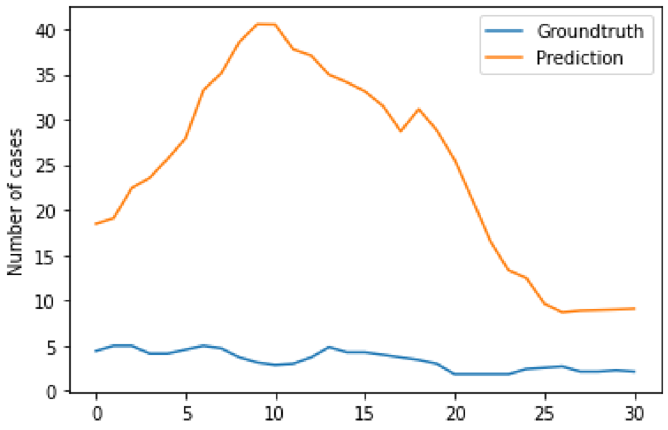
As seen, working with **smoothed** time series improves a lot the accuracy of predictions: in this way our models avoid learning a bias. Moreover, using smoothed data made simpler models perform better than complex ones.

For the **national case**, we realized that ConvLSTM and a simple LSTM model performs better than VAR.

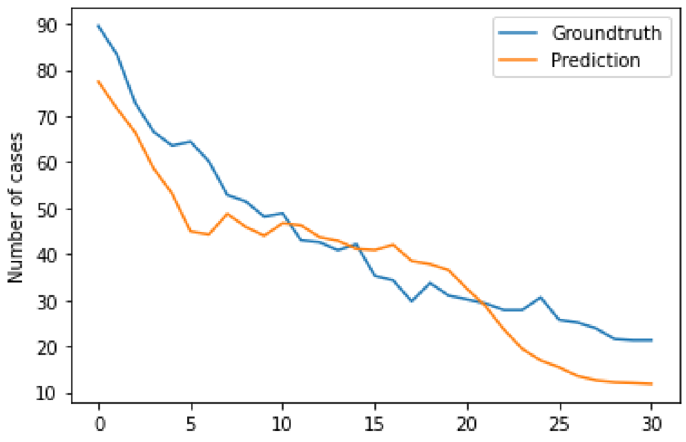
LSTM	
ConvLSTM	

VAR	

In the **regional analysis**, it’s difficult to develop a unique model that can well describe regional situations. In fact there are regions whose predictions are quite acceptable, but others for which predictions are widely wrong.
For example, when number of infected people is low, predictions do not meet groundtruth really well. Forecast for Umbria depicts this situation:



Instead, for those regions where number of infected people is higher, results are better. Forecast for Marche is an example:



For what concerns VAR and LSTM[18], one important difference is their parametric form: LSTM is made of layers applying non-linear transformations, instead VAR is based on a linear system of equations, highly parameterized.

Another difference between the two is that VAR could only perform well on stationary time series (where there is no seasonality, trend and etc.), instead for LSTM this is not a requirement.

LSTM works better if we are dealing with huge amount of data and enough training data is available, while VAR is better for smaller datasets.

VAR data processing and model specification are in general simpler than LSTM, and training time is shorter; instead LSTM involves more data preprocessing and requires much more efforts in the network architecture design, model training and hyperparameter tuning.

A common neural network's drawback is that they use black boxes. This is a big problem if we have to explain how the model works.

Usually a perfect compromise is to form an ensemble estimator combining two or more models.

[18] https://databricks.com/session/time-series-forecasting-using-recurrent-neural-network-and-vector-autoregressive-model-when-and-how