UNIVERSIDAD
COMPLUTENSE
MADRID

MODELLING WEEK 2021. REPORT: A DATA-DRIVEN APPROACH TO EVALUATE THE IMPACT OF RESTRICTIVE MEASURES ON THE INCIDENCE OF COVID-19

June, 2021

Faculty of Mathematical Sciences

Master's degree in Mathematical Enginery

José Luis Barba Friginal

Inés Hernández Sánchez

Laura Ibáñez Gómez

Pablo Pérez Martín

Rodrigo Rehbein González

María Rodríguez Yaque

Abstract

A range of public health measures have been implemented to suppress local transmission of coronavirus disease 2019 (COVID-19) in all provinces of Spain. We examined the effect of these interventions and behavioural changes of the public on the incidence of COVID-19, as well as on influenza virus infections, which might share some aspects of transmission dynamics with COVID-19.

Our study shows that non-pharmaceutical interventions (including border restrictions, quarantine and isolation, distancing, and changes in population behaviour) were associated with reduced transmission of COVID-19 in provinces of Spain.

Keywords: Non Pharmaceutical Interventions (NPI), Covid incidence, Intensive Care Unit (ICU)

Contents

1 Introduction

During the past year several measures have been enforced by regional and central governments in order to mitigate the spread of SARS-Cov2 virus. These measures, known as Non-Pharmaceutical Interventions (NPIs) include limitations on social gatherings, closing or restricting commercial activities, and bounds on travel, among others. Despite the great effort invested in trying to understand the impact and efficiency of NPIs, there is still no clear consensus among the experts as to which are the most effective NPIs. The enforcement of NPIs has a considerable political cost and many collateral effects on society. Once these effects are reasonably understood, an optimal strategy can be reached for this objective optimization problem: find the combination of NPIs that reduce the incidence to reasonable values.

The main points of the project are the analysis of the available data, its transformation and then the construction of different data-driven models for the incidence using simpler approaches (as linear regression) as well as deep learning ones. We will then show some conclusions and future lines of work.

The main objectives of the report are:

- The study of influence of restrictive measures on the incidence of coronavirus in the population. To extract knowledge from this study we have applied some classical techniques as Linear Regression, Lasso Regression, Interrupted time series + Lasso, Decision Tree Regressor and Random Forest Regressor. Other methods subscribed to Artificial Intelligence (AI) for time series forecasting such as neural networks are also developed.
- Show the impact of the application of NPIs on the provinces of Spain.

The proposed problem involves merging data from different sources and creating descriptive analytics and statistical models to tackle relevant questions related to the COVID-19 pandemics in Spain.

2 Data available

In this section we describe the datasets and variables that we have used in our report. We have these datasets:

- Demographic data, in which there is the population structured by age, sex and geographic division. This geographic division is the set of provinces in Spain. There are 2183 observations.
- Epidemiological data, in which we can find the number of cases, ICU patients, deaths and other variables, structured by age, sex and geographic division. There are 811200 observations.
- Non-Pharmaceutical Interventions data, that are restrictive measures as a function of time enforced by each regional administration, structured by territorial units and expressed as a stringency index. There are 19444 observations.
- Vaccination data, in which we find the number of vaccinated individuals, structured by vaccine type and geographic division. There are 2052 observations.
- Mobility data, that is an aggregated data from mobile phones provided by either Google or the Spanish Ministry of Transport, structured by province an expressed as a relative variation with respect to a reference period. There are 24427 observations.

In order to familiarize with the different datasets we analyzed each of its variables. This will allow us to detect if any changes are needed for the information to be comparable (and cohesive) between datasets and generate a final dataset with all the data needed for our models. In this regard we had the following variables:

- In the demographic dataset, we have the variables sex, age group (in intervals of 5 years), population, province code (from 1 to 52) and name, and the autonomous community (from 1 to 19) code and name.

	sexo	edad	poblacion	cod_prov	provincia	comunidad_autonoma	cod_prov.1	cod_ccaa
0	Hombres	0-4	19050	4	Almería	Andalucía	4	1
1	Hombres	5-9	21133	4	Almería	Andalucía	4	1
2	Hombres	10-14	22107	4	Almería	Andalucía	4	1
3	Hombres	15-19	20355	4	Almería	Andalucía	4	1
4	Hombres	20-24	21794	4	Almería	Andalucía	4	1

- In the epidemiological dataset, we have the variables province code, sex (H for men and M for women), age group (in intervals of 10 years), number of cases, hospitalizations, ICU patients and deaths, and rho, that is the index of propagation of the virus.

	cod_prov	sexo	grupo_edad	fecha	num_casos	num_hosp	num_uci	num_def	rho
0	1	H	0-9	2020-01-06	0	0	0	0	NaN
1	1	H	10-19	2020-01-06	0	0	0	0	NaN
2	1	H	20-29	2020-01-06	0	0	0	0	NaN
3	1	H	30-39	2020-01-06	0	0	0	0	NaN
4	1	H	40-49	2020-01-06	0	0	0	0	NaN

- In the NPIs dataset, we have the date, province and autonomous community code and name, the territorial unit and the NPI index of culture, sport, restaurants and mobility activities. Territorial unit is only useful to distinguish the different islands that constitute the Illes Balears and Canarias, but is a variable to be omitted for coherence with other datasets. The NPIs follow, in theory, a 0 to 1 scale that shows restraint measures in a field. For example, outdoor restaurant activity measures can change between none (NPI = 0) and to be forbidden (NPI = 1), with some others such as 30% occupation limit being a value between 0 and 1.

	fecha	ccaa	cod_ccaa	cod_prov	provincia	unidad_territorial	deporte_exterior	deporte_interior	cultura	restauracion_exterior	restauracion_interior	movilidad
0	2020-06-08	Rioja, La	17.0	26.0	La Rioja	NaN	0.0	0.0	0.0	0.0	0.136	0.0
1	2020-06-09	Rioja, La	17.0	26.0	La Rioja	NaN	0.0	0.0	0.0	0.0	0.136	0.0
2	2020-06-10	Rioja, La	17.0	26.0	La Rioja	NaN	0.0	0.0	0.0	0.0	0.136	0.0
3	2020-06-11	Rioja, La	17.0	26.0	La Rioja	NaN	0.0	0.0	0.0	0.0	0.136	0.0
4	2020-06-12	Rioja, La	17.0	26.0	La Rioja	NaN	0.0	0.0	0.0	0.0	0.136	0.0

- In the vaccination dataset, we have the date, autonomous community code and name, number of doses of Pfizer, Moderna Astrazeneca and Janssen, number of delivered and administrated doses, number of people with one dose at least, and number of people completely vaccinated, and the quotient by dividing these two variables.

	informe	ccaa_name	dosis_pfizer	dosis_moderna	dosis_astrazeneca	dosis_janssen	dosis_entregadas	dosis_administradas
0	2021-01-04	Andalucía	140295	NaN	NaN	NaN	140295	25809
1	2021-01-05	Andalucía	140295	NaN	NaN	NaN	140295	40263
2	2021-01-07	Andalucía	140295	NaN	NaN	NaN	140295	53934
3	2021-01-08	Andalucía	140295	NaN	NaN	NaN	140295	69445
4	2021-01-11	Andalucía	140295	NaN	NaN	NaN	140295	81387

	personas_con_al_menos_una_dosis	personas_con_pauta_completa	fecha	cod_ccaa	administradas_sobre_entregadas
0	NaN	NaN	2021-01-03	1	0.183962
1	NaN	NaN	2021-01-04	1	0.286988
2	NaN	NaN	2021-01-06	1	0.384433
3	NaN	NaN	2021-01-07	1	0.494993
4	NaN	NaN	2021-01-09	1	0.580113

- In the mobility dataset, we have the province and autonomous community code and name, the date and percent changes in retail and recreation, grocery and pharmacies, parks, transit stations, workplaces and residential.

	sub_region_1	sub_region_2	date	retail_and_recreation_percent_change_from_baseline	grocery_and_pharmacy_percent_change_from_baseline	parks_percent_change_from_baseline
0	Galicia	A Coruña/La Coruña	2020-02-15	-4.0	-2.0	-15.0
1	Galicia	A Coruña/La Coruña	2020-02-16	-13.0	-19.0	-27.0
2	Galicia	A Coruña/La Coruña	2020-02-17	-2.0	4.0	28.0
3	Galicia	A Coruña/La Coruña	2020-02-18	1.0	0.0	21.0
4	Galicia	A Coruña/La Coruña	2020-02-19	-1.0	0.0	22.0

transit_stations_percent_change_from_baseline	workplaces_percent_change_from_baseline	residential_percent_change_from_baseline	cod_ccaa	cod_prov
-4.0	-1.0	0.0	12	15
4.0	-4.0	1.0	12	15
11.0	5.0	-1.0	12	15
2.0	4.0	-1.0	12	15
7.0	4.0	-1.0	12	15

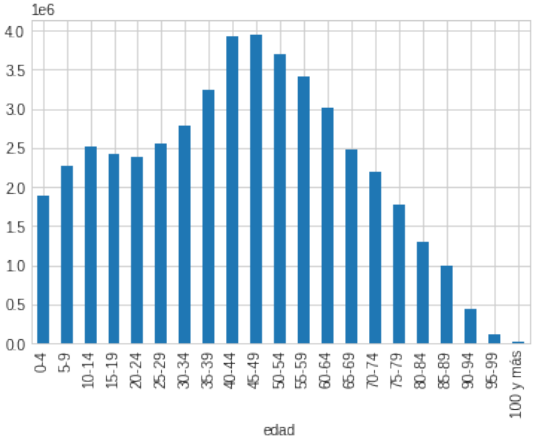
3 Data transformation

There is some degree of preprocessing that is a fundamental prerequisite for all datasets, such as removing duplicated entries, selecting the time period we are going to work with, removing observations related to Ceuta and Melilla, since they were not present in every dataset, and replace the NaN values, the ones where there is no data (the values we will replace them with, will vary from variable to variable and from dataset to dataset).

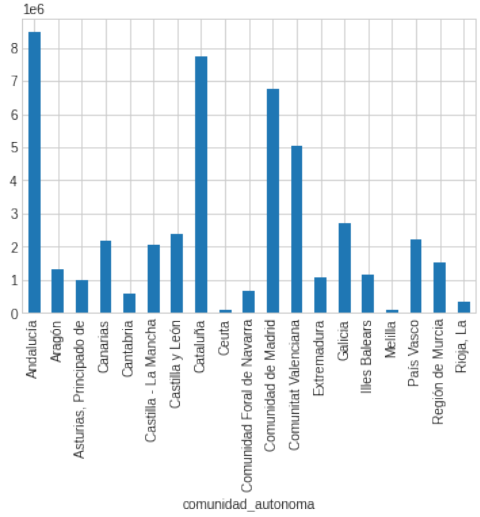
Since all data sets include different variables and come from different sources we need to take care of certain particularities, which requires processing of their own.

- Population data did not have any dates attached to them, so we assumed that the population was constant throughout duration of the experiment.
- In the NPIs dataset, the values of the NPIs should be between 0 and 1, but some of them did not so we needed to rescale them. Territorial units information was discarded to gain coherence with the other datasets.
- Incidence hadn't been calculated so we calculated it with the number of cases available in the epidemiological table. In this calculus we assumed that population was constant during the pandemic. Although this is not true, daily population numbers do not exists, so we had to run with this approximation.
- The vaccination dataset includes data grouped by autonomous communities, and all the other data is given by province. We studied the effects by provinces so we assumed that the percentage of people vaccinated in every province was equal to the percentage of the whole autonomous communities.
- Also dates related to vaccination are collected in a report that is gathered on working days, and there are some days without any vaccinations or they were not included in the report. We needed to add data to the remaining days so we assumed the vaccination count remained constant for the days we had no information and it was zero in dates before the first date of the database.
- There were a couple of human errors in dates as well. Since they were only a couple we could edit them manually.

During this process we also generated some plots of the relevant variables in order to have a visual interpretation of them. For example, we studied the population distribution by age and Autonomous Community (figure 1), the NPI's time evolution for the province of Soria (we show the rsults for restaurant activities in figure 2) and the variation of mobility and retail and recreation data with respect to before-pandemic data for the province of Álava (figure 3).

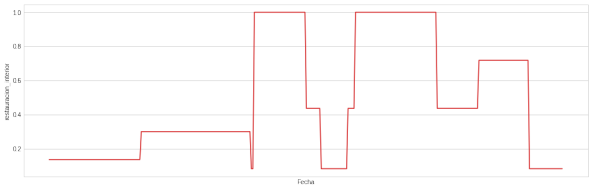


(a) Population by age

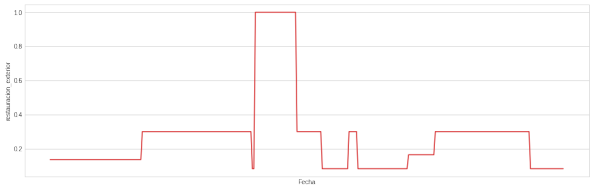


(b) Population by AACC

Figure 1: Population in Spain

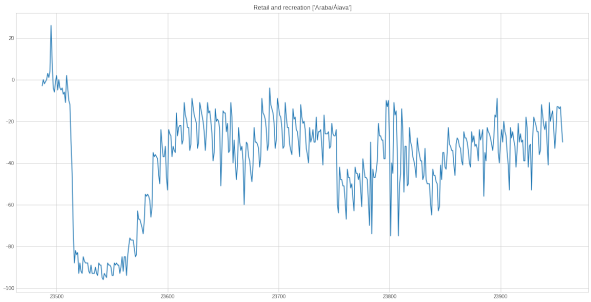


(a) Indoor restaurant activities restrictions

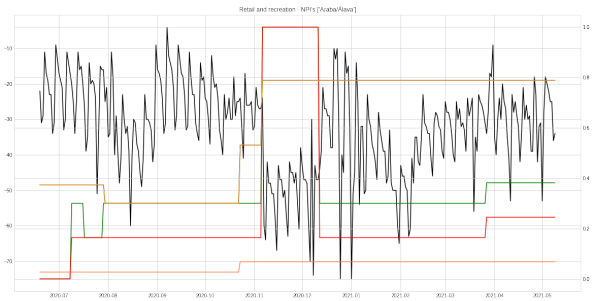


(b) Outdoor restaurant activities restrictions

Figure 2: NPI's vs time for the province of Soria



(a) Mobility data variation to standard



(b) Retail and recreation variation and NPI's

Figure 3: Movility and Retail and recreation for Álava

The figures show, at first sight, that Spain has an aged population no equally distributed by Autonomous Community (AACC). We also see that the restrictions in restaurant business were different for outdoor and indoor activity are different, motivated by the fact that the virus is more easily spreadable in closed environments. Also, the graph on the retail and recreation for Alava shows how the measures of the second wave of the pandemic affected to the data.

Then we can proceed and join all the data on a single dataframe to work with it. We will need to group the variables and perform the aggregations needed (sums when talking about number of cases for example).

Also, we add lags and gaps to our final modelling data. We cannot expect measures to have a meaningful effect right of the bat, since the virus takes time to develop. Thus, we assumed measures started working two weeks after being implanted and we lagged the incidence variable several times to study it's evolution through time.

Finally, we need to perform several tests in order to check that everything else is in order: any remaining NaN that appeared during the merging process, any variables having greater (or smaller) values than expected, and so on.

4 Simpler Approach

There are several algorithms that allow us to analyze the impact of the NPIs in the (evolution of the) incidence. However, since this is a problem of massive importance and a matter of interest for the general public, the results shall be crystal clear. Any measures taken in response to the results obtained by our models needs to be justifiable to everyone unfamiliar with advanced models of data analysis. While neural networks are more powerful, allowing us to introduce more variables and get more precise results, they lack explainability. Therefore, our initial approach to this project was what we called "simpler models", models much less versatile yet more straightforward to interpret.

These "simpler models" are also a good starting point for the project since they are easier to implement and less computationally expensive to train, so they allow us to extract early conclusions and check whether there are any serious discordance in our data. However, we will have to be aware that they tend to overfit, that is, to adjust very well to the training data and give a very poor prediction of the result.

We implemented the following models, using Python's Keras library:

- Linear Regression, where we assume that all the data is in a hyperplane of the same dimension as the variables used. It requires a selection of variables in order to lose the cross correlated ones.
- Lasso Regeression, an evolved version of the previous one, where we do not choose the variables. Instead, all the variables are in the model, but have weights that influence in the quality of the model, and will turn to zero if the variable is not needed, making the selection of variables for us.
- Lasso Regression with added variables, where we also consider the effect of lagged variables.
- Decision Tree Regressor, that creates a logical categorization of the result variable using the remaining variables in order to give an estimation.
- Random Forest, which consists in a combination of different decision trees.

Due to the linear regression needing a selection of variables, such process was also implemented by means of the cross correlation function of the matplotlib.pyplot library.

The variables we will use as input for all these models will be the NPIs described previously as long as lagged (i.e. values from former dates) incidence values, in order to calculate later incidence levels.

Another parameter to take into account when evaluating models is the gap and the lags we introduce to the data. Since incubation period is estimated in two weeks we can start by adding a gap of 14 days when taking into account NPIs, and then choose the number of lags to add to incidence. Especially for these models we don't expect to be able to predict the incidence value, but rather how is it going to vary, so we will add values of previous days. In all cases, data will be split between train data and test data, and results will be obtained from each province independently.

4.1 Linear Regression

Linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). We obviously don't expect this relation to be lineal, so results obtained from this model will not be held in high esteem.

To implement the linear regression model we must take into account the crossed correlations between each pair of variables and for each province. So we must introduce non correlated variables in linear regression model. To that end, we created a function that computes cross correlations for the variables of the problem and chooses the m ones less correlated in order to make a m-variables model. Then, using the SKLearn library, we created linear models for each province, computed the predicted data and generated plots of the incident respect to time and the coefficients of the model. The plots obtained for the province of Zaragoza are the figures 4 and 5.

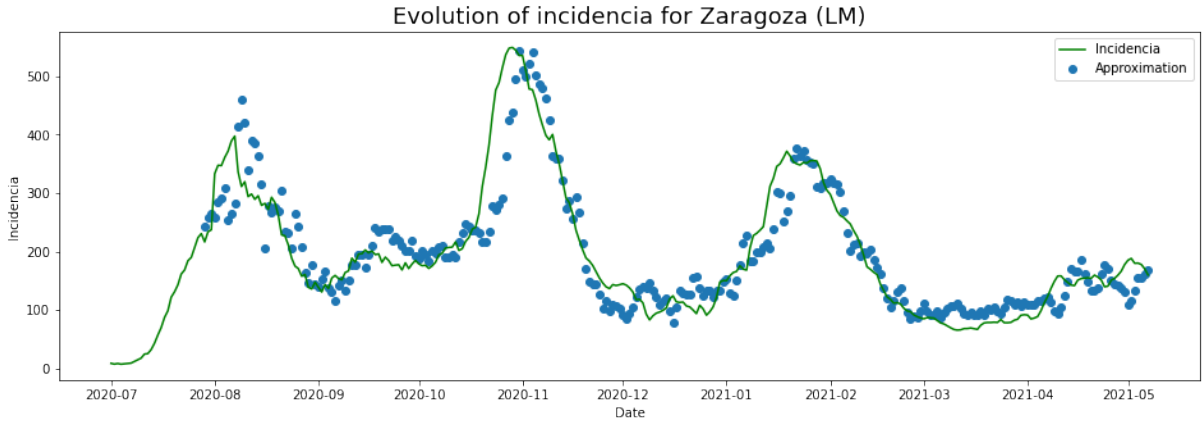


Figure 4: Real vs predicted data for Zaragoza's Linear Model

Metrics to evaluate this model will be the score (the R^2 coefficient of determination) and the MSE (mean squared error), that was calculated for each province.

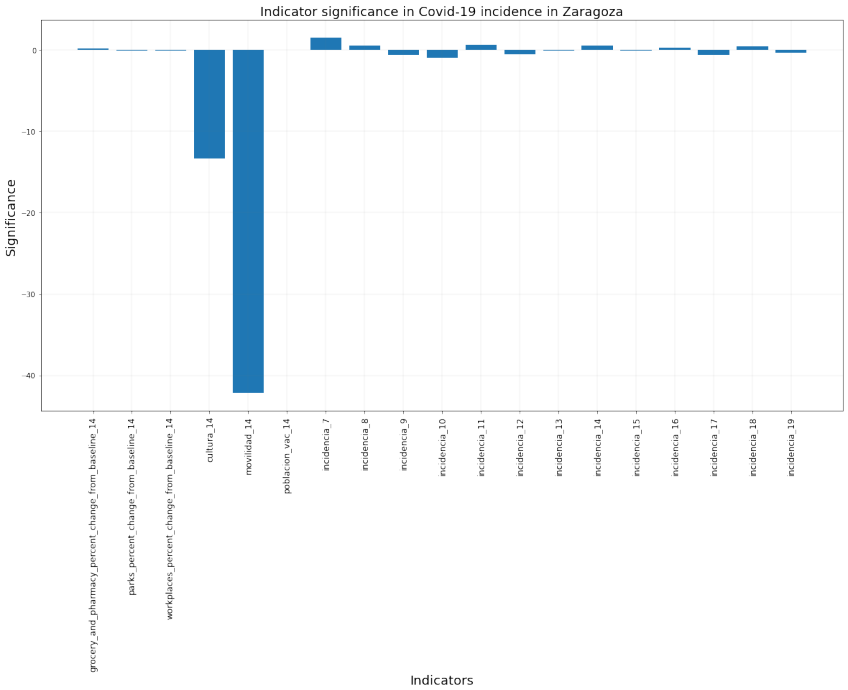


Figure 5: Coefficients of model for Zaragoza's Linear Model

In the case of Zaragoza, we get:

```
'MSE': 3718.328869013733,
'Score': 0.7259183806331236,
'cod_prov': 50}
```

what shows that our model is not great, but provides a feasible prediction of the incidence for the given data.

4.2 Lasso Regression

To reduce the variance of the linear regression model we can use a learning algorithm that includes a shrinkage penalty (also called regularization) which is Lasso Model. Lasso regularization penalizes the sum of the absolute value of the regression coefficients. This penalty is known as L1 and has the effect of forcing the coefficients of the predictors to tend to zero. Because a predictor with zero regression coefficient does not influence the model, lasso manages to exclude the least relevant predictors.

We will implement this Lasso method with $\alpha = 0.25$ and the metrics to evaluate this model will be the score and the MSE. We will generate the same plots than before and show its results for Zaragoza (figures 6 and 7).

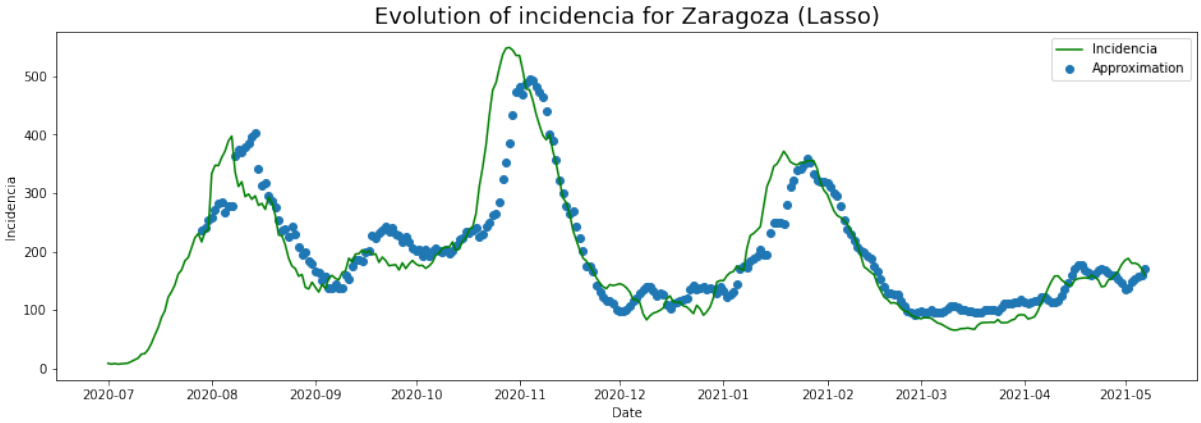


Figure 6: Real vs predicted data for Zaragoza's Lasso Model

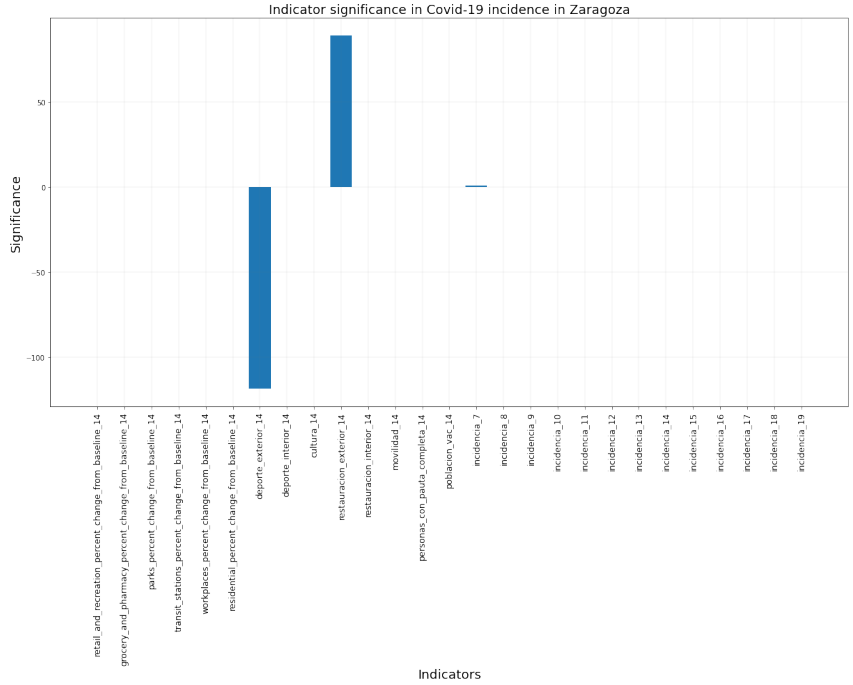


Figure 7: Coefficients of model for Zaragoza's Lasso Model

This model proves to be more capable than the first one for each of the provinces, with Zaragoza reporting a lower MSE and higher R^2 score.

```
'Lasso Regression: R^2 score on test set': 72.96177477919544,
'Lasso Regression: R^2 score on training set': 72.96177477919544,
'MSE': 3668.1413966267255,
'cod_prov': 50,
```

also, we can see that the Lasso Model selects the variables by itself, obtaining that only the restrictive measures over the outdoor sport and indoor restaurant activity, as well as the incidence seven-days-prior are relevant to the study.

4.3 Lasso Regression with added variables

In order to add more information to the model we can also add variables that represent time passage, not only to discern evolution of incidence but furthermore tendencies as time passes on. Instead of using concrete time series models models this features can be implemented through Interrupted Time Series (ITS)[BCG17]. On their paper James Lopez Bernal suggests adding a new monotone increasing variable, 'T', the numbers of days since the beginning of the experiment (in our case, July 1st of 2020, since former data is unreliable). Then, use T and T*X (X being the other predictors) as variables. Given the high number of variables we are working with we will use this T variable.

To consider the plausible effects of seasonality in our data we can add a variable that evaluates the position through the year of our data and transforms it to be a periodic function. Every other parameter of the model will be the same and metrics to evaluate this model will be the score and the MSE, just like before.

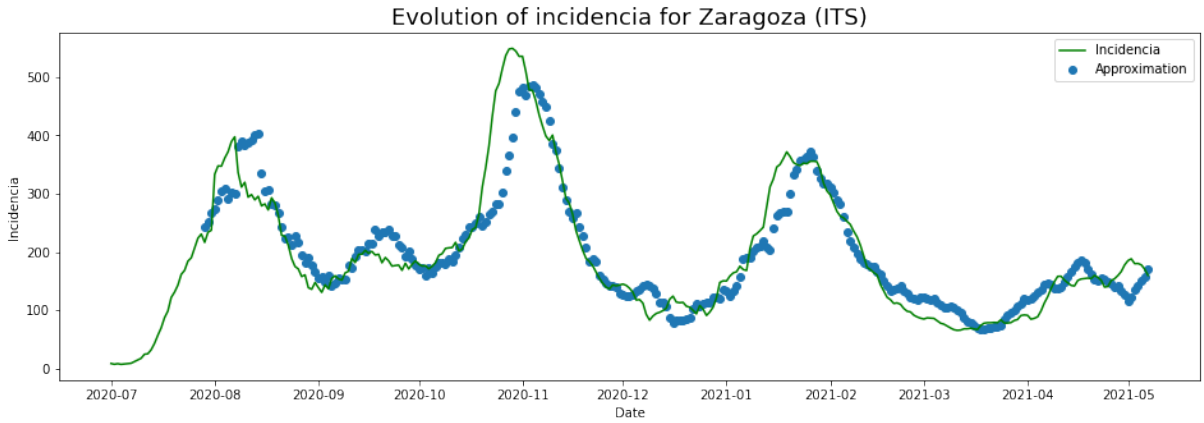


Figure 8: Real vs predicted data for Zaragoza's Interrupted Time Series Model

The results improve once again, with a even lower MSE and a better R^2 score in the case of Zaragoza.

```
'Interrumped Regression: R^2 score on test set': 77.47094938847134,
'Interrumped Regression: R^2 score on training set': 77.47094938847134,
'MSE': 3056.4041278589457,
'Score': 0.7747094938847133
```

In this case the variables of interest of the model are the restrictive measures over the outdoor sport, the change on the transit stations respect to the data before the pandemic, the seasonal variables introduced to the model and the incidence 20 days prior.

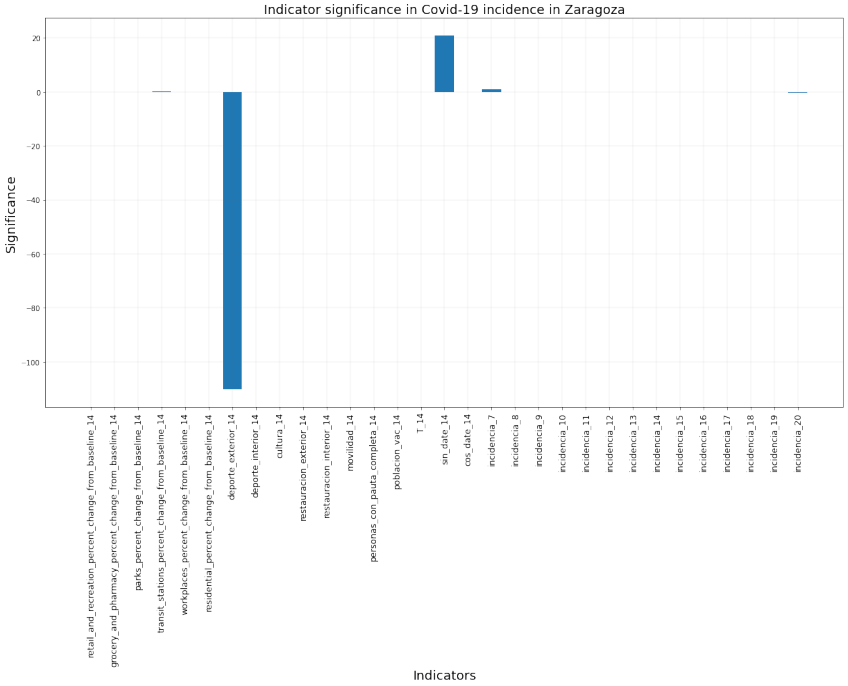


Figure 9: Coefficients of model for Zaragoza's Interrupted Time Series Model

4.4 Decision Tree Regressor

Next pair of models will be based on Decision Trees. These models uses trees of decision splitting by the values each variable will take and what will be its consequences. This model is more precise and while their intelligibility remains in a better place than with neural networks influence of variables is harder to measure since the results obtained from these models are weights rather than variables, measuring significance but not indicating if in a positive or a negative manner.

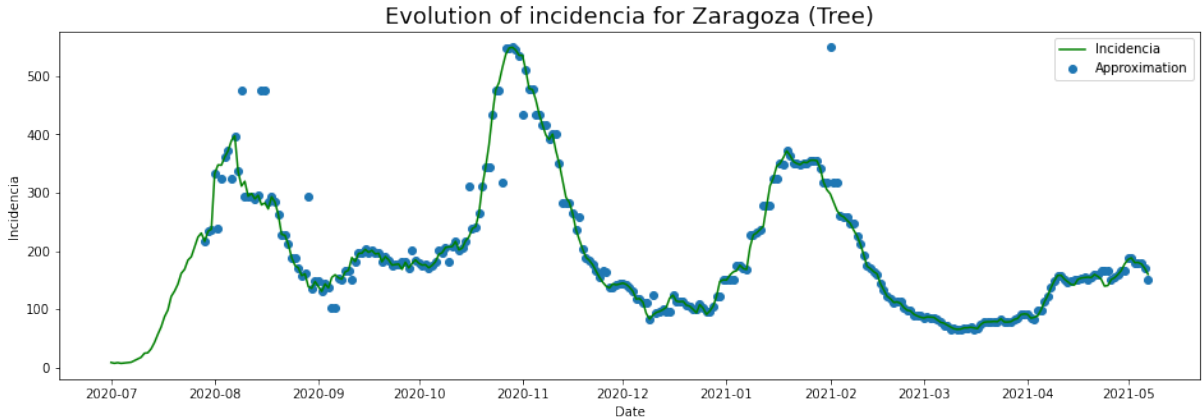


Figure 10: Real vs predicted data for Zaragoza's Decision Tree Regressor

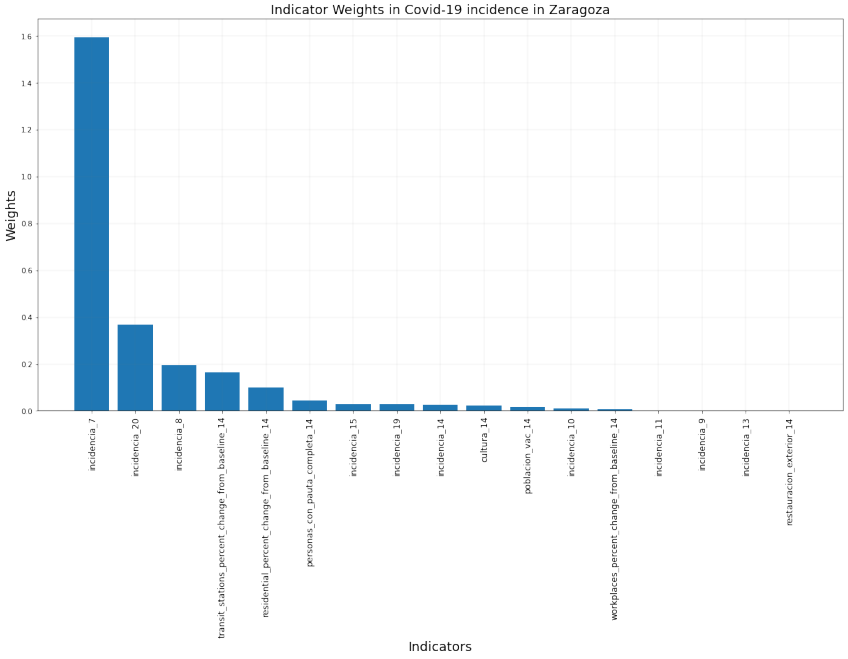


Figure 11: Weights of model for Zaragoza's Decision Tree Regressor

As a metric on the goodness of the model we compute the MSE by definition, computing the diferente between the predictions and the real values of incidence, obtaining that, in general, the decion tree is a way better model than any of the linear regressions we hace implemented. For example in the case of Zaragoza, we have a MSE of

```
'MSE': 2488.453622800451
```

which is 500 units lower than the case of the Interrupted Time Series. Also, we can compute the simulation (figure 10 and the weighting factors for each variable, being the incidence data the most important ones for Zaragoza.

4.5 Random Forest

Random Forest Regressor is a model that uses a bootstrapped subset of observations and at each node the decision rule considers only a subset of features. A prediction from the Random Forest Regressor is an average of the predictions produced by the trees in the forest. We generate a study similar to the previous ones, obtaining the MSE of the Random Forest and the simulation and weights for each province. It is observed that the MSE of the random forest is less than that the one of the regression tree.

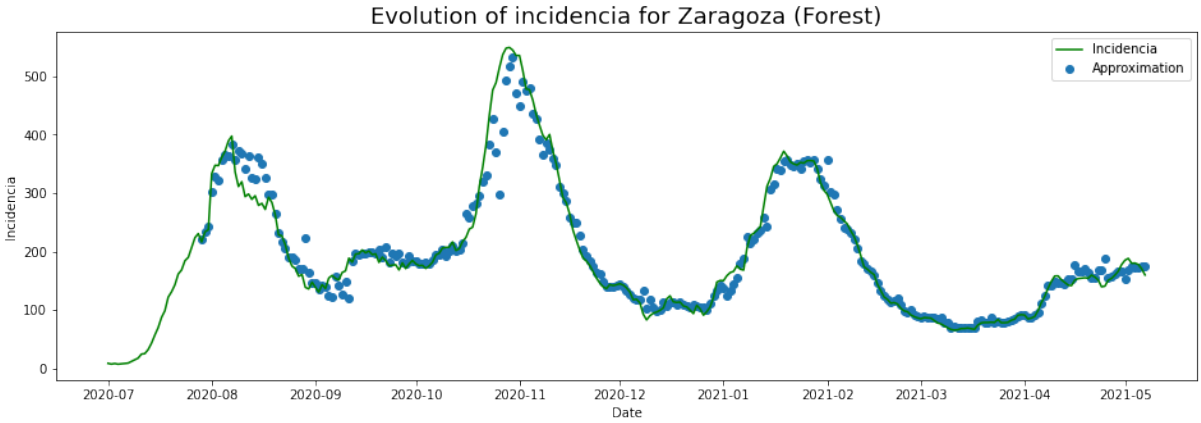


Figure 12: Real vs predicted data for Zaragoza's Random Forest

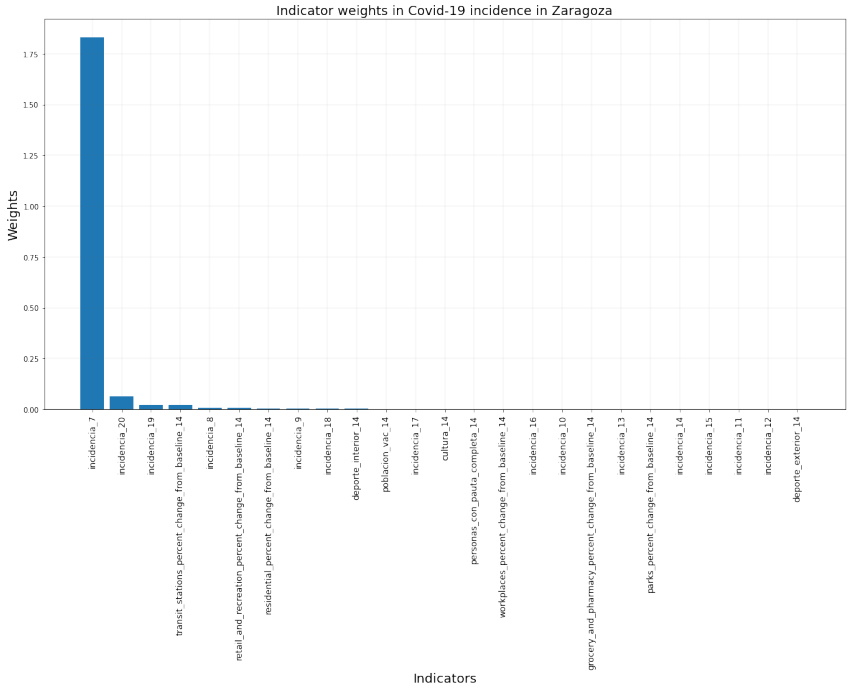


Figure 13: Weights of model for Zaragoza's Random Forest

In the case of Zaragoza the lowest MSE of all the models, what makes the Random Forest the best simpler model,

```
'MSE': 1892.2746204990972
```

being the most important variables, once again, the incidence ones.

It is worth mentioning that NPIs doesnt show to be very important in these models and, usually, only one or two of them appear as a selected variable for the model. The latter fact can be explained by the deep correlation between those variables due to the fact that the NPIs were implemented all together and straightened all at once in each wave, so their behavior is similar.

5 Deep Learning Approach

The principal objective of this section was evaluating the impact of NPI's in the incidence. In this context, we first predict the incidence with LSTM Neural Network and then, we implement an explanation method based on Shapley Value to interpret the prediction.

LSTM Neural Network is a Recurrent Neural Network (RNN) able to learn about data with long- and short-range temporal dependencies. These models are used for modeling sequential type data. Some applications are time series forecasting and text analysis. In making a prediction for the current observation, LSTM can condition the prediction on past, current and in some cases future information.

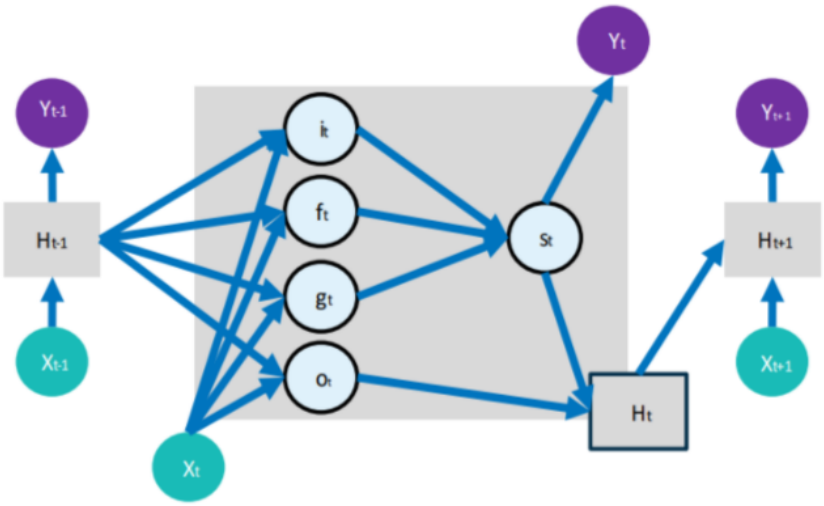


Figure 14: Neural network scheme.

LSTM models are black box, so it is necessary implement other methods to interpret the prediction. To that end, Shapley Values was applied to explain predictions generated by the LSTM model. We calculated the Shapley Values for each feature of every observation to get a global interpretation.

The following plot (figre 15) shows the Shapley Value on the x-axis. Here, all the values on the left represent the observations that shift the predicted value in the negative direction while the points on the right contribute to shifting the prediction in a positive direction.

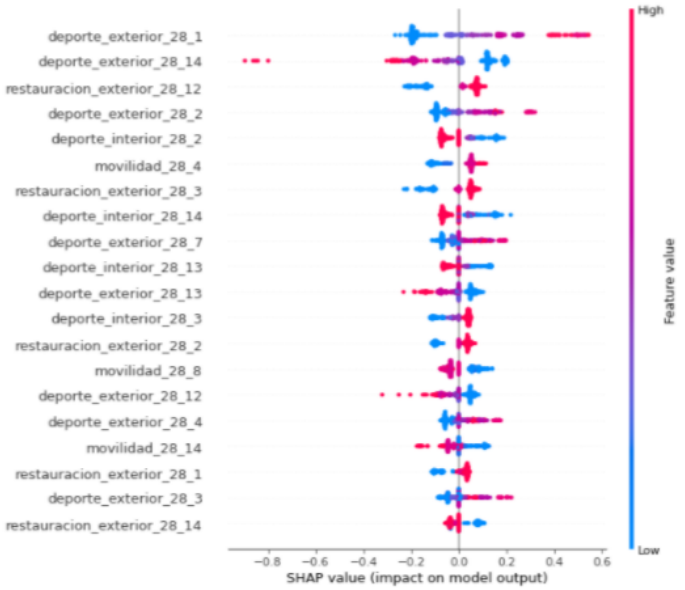


Figure 15: Shapley Values for the different variables of the model

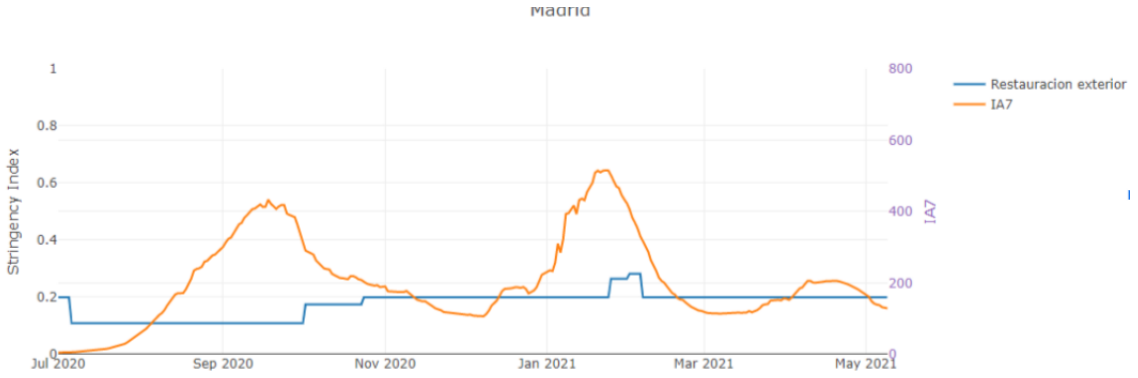


Figure 16: Madrid data

Now, we focus our Shapley Value in Madrid province, whose data can be seen in figure 16. The outdoor restaurant activities restriction affects to shifting the prediction in a positive direction and lag 14 of this feature, affect in a negative direction. At first, this doesn't make sense. However, the model is learning that restriction is removed when government acknowledges incidence decrease.

We can conclude that there exists a bidirectional causal relationship between NPI's and COVID-19 incidence.

6 Conclusions and future lines of work

The aim of this project is the study of influence of restrictive measures on the incidence of coronavirus in the population. After a week working on it we have obtained the following conclusions:

- This is an overly complicated problem, even more than we anticipated, so conclusions drawn from out models are limited, given the quantity and the quality of the data.
- The data are a real life data, coming from different sources allows us merge various types of data substantially harder and time consuming.
- After applying different measures reports we have obtain significantly unalike results around most provinces of Spain.
- In the AI model, (simulating) removing the most important NPIs makes incidence drop. Model learns that restrictions are removed when the government acknowledges incidence decreases.

In order to try and obtain more precise results we could try to:

- Add more variables and layers to the neural network.
- Rearrange how we use certain variables, changing the lags/gaps we introduce to them.
- Try different neural network structures whose characteristics are tougher to explain clearly and transparently.

A Code

A.1 Data preprocessing

```
# -*- coding: utf-8 -*-
"""00_Preprocessing.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1GvcwkZ72xP_coMyWIyaYy1fhBgPB4lJy

#### CARGA DE LIBRERIAS
"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.dates as mdates
from sklearn.preprocessing import MinMaxScaler
# %matplotlib inline
plt.style.use('seaborn-whitegrid')

# !pip install dfply
# from dfply import *

"""##### CONEXIÓN CON GOOGLE DRIVE"""

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

"""# POBLACIÓN"""

path = "/content/drive/MyDrive/Colab Notebooks/MODELLING WEEK/Conjunto de datos"

df_pob = pd.read_csv(path+"/poblacion.csv",thousands=r'.')

df_pob

"""La clave primaria del conjunto de datos es $(sexo, edad, cod\_prov)$, donde las claves toman valores:
* $sexo \in \{Hombres, Mujeres\}$.
```

```
41 * $edad \in \{0-4,5-9,...,95-99,100 \hspace{0.1cm} y \hspace{0.1cm} más\}$.
42 * $cod\_prov \in \{1,...,52\}$.
43
44 Resultando en un total de $2*21*52=2184$ observaciones.
45
46 **Datos ausentes.**
47
48 Vemos si hay datos ausentes en las variables de importancia del dataset,
       que son las de la clave primaria y la población. Si hay registros
       ausentes en los otros campos, se puede completar a partir del código de
       la provincia, ya que $provincia, comunidad\_autonoma, cod\_prov.1, cod\
       _cca$ dependen funcionalmente de $cod\_provincia$.
49 """
50
51 df_pob[df_pob.sexo.isna() | df_pob.edad.isna() | df_pob.poblacion.isna() |
       df_pob.cod_prov.isna()]
52
53 df_pob[df_pob.provincia.isna() | df_pob.comunidad_autonoma.isna() | df_pob[
       "cod_prov.1"].isna() | df_pob.cod_ccaa.isna()]
54
55 """Igualmente, no hay valores ausentes en las variables dependientes del có
       digo de provincia.
56
57 Aseguremos que hay el número de códigos de provincia coincide con el número
        de provincias escritas, e igual para las comunidades.
58 """
59
60 len(df_pob.groupby("provincia"))==len(df_pob.groupby("cod_prov")) and len(
       df_pob.groupby("cod_ccaa"))==len(df_pob.groupby("comunidad_autonoma"))
61
62 """Aseguremos también que el código de provincia está duplicado y, de serlo
       , eliminamos la variable duplicada."""
63
64 b = sum(df_pob.cod_prov == df_pob["cod_prov.1"])==2184
65 if b:
66   df_pob = df_pob.drop(['cod_prov.1'],axis=1)
67 df_pob.head()
68
69 """Tomemos sólo las variables de interés y agrupemos las variables de la
       clave primaria en las primeras columnas para realizar la siguiente parte
        del preproceso."""
70
71 pob = df_pob[["sexo","edad","cod_prov","poblacion"]]
72 pob
73
74 """**Datos duplicados.**
75
76 Veamos ahora si hay observaciones duplicadas.
```

```
77
78 Claramente, si hay dos observaciones idénticas, los valores de clave
       primaria coincidirán. Así, si no hay observaciones con idéntica clave
       primaria, no hay registros duplicados.
79 """
80
81 pob[["sexo","edad","cod_prov"]].duplicated().sum()
82
83 """No hay observaciones duplicadas.
84
85 **Número de clases de las claves.**
86
87 Cerciorémosnos de que el número de registros por sexo es 2184/2 para cada
       uno, por edad es 2184/21 por cada tramo de edad y por código de
       provincia es 2184/52 por provincia.
88 """
89
90 len(pob[pob.sexo == "Mujeres"]) == len(pob)/2
91
92 """Por complementario, hay también el mismo número de registros para
       hombres."""
93
94 valores = list(pob.edad.values)
95 clases = valores[:21]
96 b = True
97 for i in range(21):
98   b = b and (valores.count(clases[i])==len(pob)/21)
99 b
100
101 """Hay el mismo número de registros por tramo de edad."""
102
103 valores = list(pob.cod_prov.values)
104 clases = valores[:52]
105 b = True
106 for i in range(52):
107   b = b and (valores.count(clases[i])==len(pob)/52)
108 b
109
110 """Hay el mismo número de observaciones por cada provincia.
111
112 Así, tenemos consistencia en los datos.
113
114 Hagamos algunos gráficos.
115
116 **Población total por sexos.**
117 """
118
119 df_pob.groupby("sexo").poblacion.agg(sum).plot(kind = 'bar')
```

```
120
121 """**Población total por tramos de edad.**"""
122
123 df_pob.groupby("edad",sort=False).poblacion.agg(sum).plot(kind = 'bar')
124
125 """**Población total por provincias.**"""
126
127 df_pob.groupby("provincia").poblacion.agg(sum).plot(kind = 'bar')
128
129 """**Población total por comunidades autónomas.**"""
130
131 df_pob.groupby("comunidad_autonoma").poblacion.agg(sum).plot(kind = 'bar')
132
133 """##Agregacion de la tabla de poblacion a nivel provincia
134
135 """
136
137 df_pob_agr=pd.DataFrame(df_pob.groupby(['cod_prov','cod_ccaa']).poblacion.
      agg(sum))
138 df_pob_agr.head()
139
140 len(df_pob_agr.groupby("cod_prov"))
141
142 """# VACUNAS """
143
144 df_vac = pd.read_csv(path+'/vacunas.csv')
145 df_vac = df_vac.rename(columns={'ultima_vacuna_registrada':'fecha'})
146 df_vac["fecha"] = pd.to_datetime(
147     df_vac["fecha"], format="%Y/%m/%d"
148 )
149 df_vac["informe"] = pd.to_datetime(
150     df_vac["informe"], format="%Y/%m/%d"
151 )
152
153 df_vac['mes'] = df_vac.informe.apply(lambda x: x.month)
154 df_vac.sort_values(by=['cod_ccaa','fecha'])
155 df_vac.head(10)
156
157 len(df_vac.groupby("cod_ccaa"))
158
159 df_vac.describe()
160
161 df_count = df_vac.groupby(['ccaa_name','cod_ccaa']).size().reset_index().
      rename(columns={0:'count'}).sort_values(by=['cod_ccaa'])
162 code_list = list(zip(df_count['cod_ccaa'],df_count['ccaa_name']))
163 code_list
164
165 def ccaa_to_cod(x):
```

```
166     idx = [y[0] for y in code_list].index(x)
167     return code_list[idx][1]
168
169 cols = list(df_vac)
170 cols.insert(0, cols.pop(cols.index('fecha')))
171 df_vac = df_vac[cols]
172
173 df_vac.isnull().sum()
174
175 """Como los NaN's de las dosis son porque no se repartieron ninguna de esas
        farmacéuticas podemos poner 0's."""
176
177 df_vac[['dosis_pfizer', 'dosis_moderna', 'dosis_astrazeneca', '
    dosis_janssen']] = df_vac[['dosis_pfizer', 'dosis_moderna', '
    dosis_astrazeneca', 'dosis_janssen']].where(df_vac[['dosis_pfizer', '
    dosis_moderna', 'dosis_astrazeneca', 'dosis_janssen']].isnull(),df_vac[[
    'dosis_pfizer', 'dosis_moderna', 'dosis_astrazeneca', 'dosis_janssen']])
    .fillna(0).astype(int)
178 # Solo en una nos faltan datos de vacunas
179 df_vac[df_vac.dosis_pfizer + df_vac.dosis_moderna + df_vac.
    dosis_astrazeneca + df_vac.dosis_janssen != df_vac.dosis_entregadas]
180
181 # Cantidad de informes
182 len(df_vac.informe.unique())
183
184 """Hay 108 informes y 108*19 = 2052 registros. (Decidir si usamos la fecha
    del informe o la de última vacuna.)
185
186 Gráficas de dosis administradas por meses
187 """
188
189 # l = df_vac.groupby(['cod_ccaa','mes']).dosis_administradas.max()
190 # ccaa = [i for i in range(1,20)]
191 # for i in range(1,7):
192 #     entr = [l[j][i] for j in ccaa]
193 #     plt.bar(ccaa, entr)
194 #     plt.xticks(ccaa,[y for (x,y) in code_list], rotation = 90)
195 #     plt.title('Dosis administradas a principios del mes '+str(i))
196 #     plt.show()
197
198 """Gráficas de dosis administradas por CC.AA."""
199
200 # meses = [i for i in range(1,7)]
201 # for i in range(1,20):
202 #     entr = [l[i][j] for j in meses]
203 #     plt.bar(meses, entr)
204 #     plt.title('Dosis administradas en '+str([y for (x,y) in code_list if
    x==i][0]))
```

```
#       plt.show()

df_vac = df_vac.drop_duplicates(subset=['fecha', 'cod_ccaa'])
df_vac.reset_index()

df_vac[df_vac.fecha>pd.Timestamp('2021-06-06 00:00:00')]

df_vac.at[431, 'fecha'] = pd.Timestamp('2021-01-07 00:00:00')
df_vac.at[539, 'fecha'] = pd.Timestamp('2021-01-13 00:00:00')

data = []
for i in range(1,20):
    data.insert(0, {'cod_ccaa': i, 'fecha': pd.to_datetime('06-01-2020'), '
        personas_con_pauta_completa': 0.0})

df_vac = pd.concat([pd.DataFrame(data), df_vac], ignore_index=True)

df_vac = (df_vac[['fecha', 'cod_ccaa', 'personas_con_pauta_completa']].
      set_index('fecha').
      groupby('cod_ccaa').
      resample('1D').
      sum().
      drop('cod_ccaa', axis=1).
      reset_index(['fecha', 'cod_ccaa']))
df_vac = df_vac[df_vac.fecha < '2021-05-09']

df_vac['personas_con_pauta_completa'] = df_vac['personas_con_pauta_completa
    '].replace({0:np.nan})
df_vac['personas_con_pauta_completa'] = df_vac.groupby('cod_ccaa')['
    personas_con_pauta_completa'].transform(lambda x: x.fillna(method='ffill
    '))
df_vac['personas_con_pauta_completa'] = df_vac['personas_con_pauta_completa
    '].fillna(0)

df_vac.head()

len(df_vac.groupby("cod_ccaa"))

"""## Normalizar Vacunas"""

df_pob_agr.reset_index(inplace=True)

df_aux=df_pob_agr.merge(df_vac, left_on=['cod_ccaa'], right_on=['cod_ccaa'
    ],how='left')

com = df_aux[df_aux.fecha<pd.to_datetime('2020-07-02')].groupby('cod_ccaa')
    .poblacion.sum()
com = pd.DataFrame(com)
```

```
246
247 com.head()
248
249 com.reset_index(inplace=True)
250 com = com.rename(columns={'poblacion':'poblacion_ccaa'})
251 com.head()
252
253 df_aux2=df_aux.merge(com, left_on=['cod_ccaa'], right_on=['cod_ccaa'],how='
    left')
254
255 df_aux2['porcentaje_vac'] = (df_aux2['personas_con_pauta_completa']/df_aux2
    ['poblacion_ccaa'])
256
257 df_aux2['poblacion_vac'] = (df_aux2['personas_con_pauta_completa']/df_aux2[
    'poblacion_ccaa'])*df_aux2['poblacion']
258
259 df_aux2.tail()
260
261 df_aux2 = df_aux2[df_aux2['cod_ccaa']!=19]
262 df_aux2 = df_aux2[df_aux2['cod_ccaa']!=18]
263
264 df_aux2 = df_aux2.drop(['porcentaje_vac'],axis=1)
265
266 len(df_aux2.groupby("cod_prov"))
267
268 """# NPI STRINGENCY"""
269
270 # Cargamos los datos a estudiar
271 df_npi = pd.read_csv(path+"/npi_stringency.csv")
272 # Posibles valores de cada variable
273 for col in df_npi.columns:
274     print('\nPosibles valores de la variable',col,':')
275     print(df_npi[col].unique())
276     print('Total de valores posibles: ',len(df_npi[col].unique()))
277
278 (df_npi['restauracion_interior']>1).sum()
279 (df_npi['restauracion_exterior']>1).sum()
280
281 # Eliminamos observaciones duplicadas
282 df_npi = df_npi.drop_duplicates()
283
284 # Todas las variables de restricciones son numéricas
285 # Eliminamos todas las observaciones que no proporcionan información sobre
    la provincia
286 df_npi = df_npi.dropna(how='any',subset = ['cod_prov'])
287 # Contamos los missings en el resultante
288 print('\nConteo de missings: ')
289 print(df_npi.isnull().sum())
```

```
290
291 df_npi = df_npi[df_npi['unidad_territorial'].isnull()]
292 df_npi = df_npi.drop('unidad_territorial', axis=1)
293 df_npi
294
295 """En estas condiciones la clave primaria es: $( fecha , cod\_prov)$."""
296
297
298
299 """Extraemos el dataset por fecha."""
300
301 # df_npi = df_npi[df_npi['fecha']>='2020-07-01']
302 # df_npi = df_npi[df_npi['fecha']<='2021-05-08']
303 # df_npi
304
305 """Gráficas de restricción por provincia:"""
306
307 def RestrictionEvolution(province):
308     df_aux = df_npi[df_npi['provincia']==province]
309     for var in df_aux.columns.tolist()[-6:-1]:
310         locator = mdates.MonthLocator()
311         plt.figure(figsize=(16,5))
312         plt.plot(df_aux.fecha, df_aux[var], color='tab:red')
313         plt.gca().set(title='', xlabel='Fecha', ylabel=var)
314         X = plt.gca().xaxis
315         X.set_major_locator(locator)
316         plt.show()
317
318 RestrictionEvolution('Soria')
319
320 len(df_npi.groupby("cod_prov"))
321
322 """# MOVILIDAD"""
323
324 #definimos la ruta donde hemos colocado los archivos
325 # path= '/content/drive/MyDrive/MODELLING WEEK'
326 #pathLaura='/content/drive/MyDrive/COVID19'
327
328 df_mov= pd.read_csv(path + '/movilidad.csv')
329 df_mov0= pd.read_csv(path + '/movilidad.csv')
330
331 df_mov.head()
332
333 """**Missing**"""
334
335 # Contamos el número de datos missings que contiene el conjunto de datos
      inicial
336 num_missings = df_mov.isnull().sum().sort_values(ascending=False)
```

```
337 print("El número de valores missing es de :\n{}".format(num_missings))
338
339 """**Duplicados**"""
340
341 # Número de registros del dataset original
342 num_filas = df_mov.shape[0]
343 # Comprobamos si hay duplicados sobre el registro completo
344 df_mov = df_mov.drop_duplicates()
345 # Número de registros del dataset sin duplicados
346 num_filas_sin_dup= df_mov.shape[0]
347 print("En este dataset hay {} registros repetidos en total".format(
        num_filas - num_filas_sin_dup))
348
349 """**Interpolamos los valores nulos**"""
350
351 # Select variables with nan
352 variables_with_nan = ['retail_and_recreation_percent_change_from_baseline',
353                       'grocery_and_pharmacy_percent_change_from_baseline',
354                       'parks_percent_change_from_baseline',
355                       'transit_stations_percent_change_from_baseline',
356                       'workplaces_percent_change_from_baseline']
357 # Sort by province and date
358 df_mov = df_mov.sort_values(by = ['cod_prov','date'])
359 # Interpolate
360 for var in variables_with_nan:
361   for codprov in df_mov['cod_prov'].unique():
362         mask = df_mov['cod_prov']==codprov
363         df_mov.loc[mask,var] = df_mov.loc[mask,var].interpolate()
364   # Fill nan with 0
365   #df_mov = df_mov.fillna(0)
366 print(df_mov[variables_with_nan].isna().sum())
367
368 for i in range(1,52):
369     df_prov=df_mov[df_mov['cod_prov']==i]
370     plt.figure(figsize=(20,10))
371     nombre_provincia=df_prov['sub_region_2'].unique().tolist()
372     plt.title("Retail and recreation {}". format(nombre_provincia))
373     df_prov['retail_and_recreation_percent_change_from_baseline'].plot()
374     plt.show()
375
376 """**Establecer indice de fecha** (en el df_mov0)"""
377
378 df_mov0.date = pd.to_datetime(df_mov.date)
379 df_mov0.set_index('date', inplace = True)
380
381 df_mov0.head()
382
383 """**Campos del dataset**"""
```

```
384
385 df_mov.columns
386
387 name_vars_object = [name for name,tipo in df_mov.dtypes.iteritems() if '
    object' in str(tipo)]
388 name_vars_float = [name for name, tipo in df_mov.dtypes.iteritems() if '
    float' in str(tipo)]
389 name_vars_int = [name for name, tipo in df_mov.dtypes.iteritems() if 'int'
    in str(tipo)]
390
391 """**Análisis movilidad por provincias**
392
393 No se dispone de los datos a nivel comunidad autónoma. Por lo tanto, se
    realizará un análisis de los mismos a nivel provincial.
394
395 - Retail and recreation
396 """
397
398 for i in range(1,52):
399   df_prov=df_mov0[df_mov0.cod_prov==i]
400   plt.figure(figsize=(20,10))
401   nombre_provincia=df_prov['sub_region_2'].unique().tolist()
402   plt.title("Retail and recreation {}". format(nombre_provincia))
403   df_prov['retail_and_recreation_percent_change_from_baseline'].plot()
404   plt.show()
405
406 """- Grocery and pharmacy"""
407
408 for i in range(1,52):
409   df_prov=df_mov0[df_mov0.cod_prov==i]
410   plt.figure(figsize=(20,10))
411   nombre_provincia=df_prov['sub_region_2'].unique().tolist()
412   plt.title("Parks {}". format(nombre_provincia))
413   df_prov['parks_percent_change_from_baseline'].plot()
414   plt.show()
415
416 """- Transit stations"""
417
418 for i in range(1,52):
419   df_prov=df_mov0[df_mov0.cod_prov==i]
420   plt.figure(figsize=(20,10))
421   nombre_provincia=df_prov['sub_region_2'].unique().tolist()
422   plt.title("Transit Stations {}". format(nombre_provincia))
423   df_prov['transit_stations_percent_change_from_baseline'].plot()
424   plt.show()
425
426 """- Workplaces"""
427
```

```
for i in range(1,52):
  df_prov=df_mov0[df_mov0.cod_prov==i]
  plt.figure(figsize=(20,10))
  nombre_provincia=df_prov['sub_region_2'].unique().tolist()
  plt.title("Workplaces {}". format(nombre_provincia))
  df_prov['workplaces_percent_change_from_baseline'].plot()
  plt.show()

"""- Residential"""

for i in range(1,52):
  df_prov=df_mov0[df_mov0.cod_prov==i]
  plt.figure(figsize=(20,10))
  nombre_provincia=df_prov['sub_region_2'].unique().tolist()
  plt.title("Residential {}". format(nombre_provincia))
  df_prov['residential_percent_change_from_baseline'].plot()
  plt.show()

len(df_mov.groupby("cod_prov"))

"""# Análisis relación NPIs - Movilidad"""

# Lectura datos NPI's
df_npi_mov=pd.read_csv(path + '/npi_stringency.csv')
df_npi_mov.head()

# Cruce
df_mov_npi=df_mov.merge(df_npi_mov, left_on=['date','cod_prov'], right_on=[
    'fecha','cod_prov'])
df_mov_npi.date = pd.to_datetime(df_mov_npi.date)
df_mov_npi.set_index('date', inplace = True)
df_mov_npi

"""### Retail and recreation

El objetivo de este apartado es visualizar la relación entre las
    restricciones NPI y la movilidad. En el eje de la izquierda se
    representa el indice de movilidad en porcentaje respecto a la base y en
    el eje de la derecha los diferentes restricciones que toman valor entre
    0 y 1, siendo 1 lo más restrictivo
"""

for i in range(1,51):
  df_prov=df_mov_npi[df_mov_npi.cod_prov==i]
  nombre_provincia=df_prov['sub_region_2'].unique().tolist()
  fig, ax = plt.subplots(figsize=(20,10))
  # Plot linear sequence, and set tick labels to the same color
  ax.plot( df_prov['retail_and_recreation_percent_change_from_baseline'],
```

```
        color='black')
    ax.tick_params(axis='y', labelcolor='black')
    # Generate a new Axes instance, on the twin-X axes (same position)
    ax2 = ax.twinx()
    # Plot exponential sequence, set scale to logarithmic and change tick
        color
    ax2.plot(df_prov['restauracion_interior'], color='green')
    ax2.plot(df_prov['restauracion_exterior'], color='red')
    ax2.plot(df_prov['deporte_exterior'], color='blue')
    ax2.plot(df_prov['deporte_interior'], color='orange')
    ax2.plot(df_prov['cultura'], color='coral')
    ax2.tick_params(axis='y', labelcolor='black')
    plt.title("Retail and recreation - NPI's {}". format(nombre_provincia))
    plt.show()

"""# INCIDENCIA"""

df_inc= pd.read_csv(path + '/incidencia.csv')

len(df_inc.groupby("cod_prov"))

df_inc

df_inc.columns.values

df_inc.dtypes

df_inc.head(10)

df_inc.tail(10)

df_inc.info()

df_inc.describe()

df_inc.isnull().sum()

df_inc.duplicated()

df_inc=df_inc.drop_duplicates()

df_inc=df_inc.set_index("fecha")

df_inc.isnull().sum()

corr = df_inc.corr()

f, ax = plt.subplots(figsize=(10,12))
```

```
517
518 cmap = sns.diverging_palette(220, 10, as_cmap=True)
519
520 _ = sns.heatmap(corr, cmap="YlGn", square=True, ax=ax, annot=True,
        linewidth=0.1)
521
522 plt.title("Pearson correlation of Features", y=1.05, size=15)
523
524 len(df_inc.groupby("cod_prov"))
525
526
527
528 """## Agregacion de Incidencias a nivel provincia"""
529
530 df_inc.reset_index(inplace=True)
531 df_inc.head()
532
533 df_inc = df_inc.drop(['sexo','grupo_edad','rho'],axis=1).groupby(["cod_prov
        ","fecha"]).agg(sum)
534 df_inc
535
536 df_inc.reset_index(inplace=True)
537 df_inc_agr = df_inc
538 df_inc_agr.head()
539
540
541
542 """# TURISMO (POR SI SE USA)
543
544 de: https://www.ine.es/jaxiT3/Tabla.htm?t=10823
545 """
546
547 df_tur = pd.read_csv(path+'/turismo.csv',sep = ';', engine='python',
        thousands = ',')
548 df_tur = df_tur[df_tur['Autonomous Communities'] != 'Other Autonomous
        Communities']
549 df_tur = df_tur[df_tur['Autonomous Communities'] != 'Total']
550 df_tur['mes'] = df_tur.Period.apply(lambda x: int(x[-1]))
551 df_tur['cod_ccaa'] = df_tur['Autonomous Communities'].apply(lambda x: int(x
        .split()[0]))
552 df_tur["total"] = pd.to_numeric(df_tur["Total"])
553
554 df_tur = df_tur[['mes','cod_ccaa','total']]
555 t = df_tur.groupby(['cod_ccaa','mes']).total.max()
556 t
557
558 ccaa = sorted(df_tur.cod_ccaa.unique())
559 for i in range(1,5):
```

```
560     entr = [t[j][i] for j in ccaa]
561     plt.bar(ccaa, entr)
562     plt.xticks(ccaa,[y for (x,y) in code_list if x in ccaa], rotation = 90)
563     plt.title('Turistas durante el mes '+str(i))
564     plt.show()
565
566 meses = [i for i in range(1,5)]
567 for i in ccaa:
568     entr = [t[i][j] for j in meses]
569     plt.bar(meses, entr)
570     plt.xticks(meses, meses)
571     plt.title('Turistas '+str([y for (x,y) in code_list if x==i][0]))
572     plt.show()
573
574 """# TABLÓN MODELIZACIÓN"""
575
576 # Cruce
577 df=df_mov.merge(df_npi, left_on=['date','cod_prov'], right_on=['fecha','
        cod_prov'])
578
579 df=df.merge(df_inc_agr, left_on=['date','cod_prov'], right_on=['fecha','
        cod_prov'])
580
581 df['date'] = pd.to_datetime(
582     df["date"], format="%Y/%m/%d"
583 )
584
585 df = df.merge(df_aux2, left_on=['date','cod_prov'], right_on=['fecha','
        cod_prov'])
586
587 df.head(20)
588
589 """**Eliminar variables del tablón final**"""
590
591 df_final=df.drop(["sub_region_1","sub_region_2","fecha_x","fecha_y",'
        cod_ccaa_x',"cod_ccaa_y",'date'],axis=1)
592
593 df.head()
594
595 df_final.shape
596
597 df_final.head(20)
598
599 """No hay datos para Ceuta y Melilla"""
600
601 df_final[df_final["cod_prov"]==52]
602
603 df_final.head()
```

```
604
605 incidencia = ((df_final.
606       set_index('fecha').
607       groupby('cod_prov')['num_casos'].rolling(7).sum().reset_index(['fecha
          ', 'cod_prov'])).num_casos / df_final.poblacion) * 100000
608
609 df_final['incidencia'] = incidencia
610
611 df_final_2 = df_final[df_final.fecha > pd.to_datetime('2020-06-30', format=
    "%Y/%m/%d")]
612 df_final_2 = df_final_2[df_final_2.fecha < pd.to_datetime('2021-05-08',
    format="%Y/%m/%d")]
613
614 df_final_2.reset_index(drop=True, inplace=True)
615
616 df_final_2.isna().sum()
617
618 #dff = (df_final.
619 #       set_index('fecha').
620 #       groupby('cod_prov')['deporte_exterior','deporte_interior','cultura
    ','restauracion_exterior','restauracion_interior','movilidad','
    personas_con_pauta_completa','incidencia'].shift(14).fillna(0)).
    reset_index()
621
622 #df_final[['fecha','deporte_exterior','deporte_interior','cultura','
    restauracion_exterior','restauracion_interior','movilidad','
    personas_con_pauta_completa','incidencia'] ]= dff[['fecha','
    deporte_exterior','deporte_interior','cultura','restauracion_exterior','
    restauracion_interior','movilidad','personas_con_pauta_completa','
    incidencia']]
623
624 df_final_2
625
626 scaler = MinMaxScaler()
627 lista_sc=['deporte_exterior','deporte_interior','cultura','
    restauracion_exterior','restauracion_interior','movilidad']
628 df_final_2[lista_sc] = scaler.fit_transform(df_final_2[lista_sc])
629
630 df_final_2['cultura'].max()
631
632 def compute_incidence_normed(
633     series_casos: pd.Series, days: int = 7, num_lag: int = 4
634 ) -> pd.Series:
635     """
636     Parameters
637     ----------
638     series_casos : pandas.Series
639         COVID incidence per date
```

```
640     days : int, optional
641         Size of the cumulative sum, by default 7
642     num_lag : int, optional
643         Number of lags to use, by default 4
644     Returns
645     -------
646     pandas.Series
647     """
648     series_casos_peso = compute_incidence_weighted(series_casos, days=days)
649     list_series_peso = [
650         series_casos_peso.shift(lag * days) for lag in range(num_lag + 1)
651     ]
652     series_mean = pd.concat(list_series_peso, axis=1).mean(axis=1)
653     series_casos_norm = np.divide(series_casos, series_mean)
654     return series_casos_norm
655
656 def compute_rho(
657     series_casos: pd.Series, days: int = 7, lag_peso: int = 4, lag_norm:
        int = 6
658 ) -> pd.Series:
659     """
660     Parameters
661     ----------
662     series_casos : pandas.Series
663         COVID incidence per date
664     days : int, optional
665         Size of the cumulative sum, by default 7
666     lag_peso : int, optional
667         Number of lags to use when computing weighted incidence, by default
            4
668     lag_norm : int, optional
669         Number of lags to use when computing movavg normed incidence, by
            default 7
670     Returns
671     -------
672     pandas.Series
673         Rho
674     """
675     # Compute the moving average of normed incidence
676     series_casos_norm = compute_incidence_normed(
677         series_casos, days=days, num_lag=lag_peso
678     )
679     series_norm_movavg = moving_average(series_casos_norm, lag_norm)
680     # Compute rho
681     list_numerator = [series_norm_movavg.shift(lag) for lag in range(3)]
682     numerator = pd.concat(list_numerator, axis=1).mean(axis=1)
683     list_denominator = [series_norm_movavg.shift(lag) for lag in range(5,
        8)]
```

```
684     denominator = pd.concat(list_denominator, axis=1).mean(axis=1)
685     rho = np.divide(numerator, denominator)
686     rho.name = "rho"
687     return rho
688
689 def compute_incidence_weighted(series_casos: pd.Series, days: int = 7) ->
        pd.Series:
690     """Computes weighted incidence: incidence per day divided by the
            average incidence
691     during the last days
692     Parameters
693     ----------
694     series_casos : pandas.Series
695         COVID incidence per date
696     days : int, optional
697         Size of the cumulative sum, by default 7
698     Returns
699     -------
700     pandas.Series
701         COVID incidence per date, normalized by the average incidence of
                the last days
702     """
703     acum = moving_average(series_casos, days)
704     series_casos_peso = np.divide(series_casos, acum)
705     return series_casos_peso
706
707 def cumulative_incidence(x: pd.Series, w: int) -> pd.Series:
708     """Computes the cumulative incidence of a series
709     Parameters
710     ----------
711     x : pandas.Series
712     w : int
713         Size of the acumulation
714     Returns
715     -------
716     pandas.Series
717     """
718     idx = x.index
719     x_cum = np.convolve(x, np.ones(w), "valid")
720     x_cum = pd.Series(x_cum, index=idx[(w - 1) :])
721     return x_cum
722 def moving_average(x: pd.Series, w: int) -> pd.Series:
723     """Computes the moving average of a series
724     Parameters
725     ----------
726     x : pandas.Series
727     w : int
728         Size of the moving average
```

```
729     Returns
730     -------
731     pandas.Series
732     """
733     x_movavg = cumulative_incidence(x, w) / w
734     return x_movavg
735
736 def clean_inf(df, col = "rho"):
737     # we fill the infinities in rho for 0
738     ser = df[col]
739     mask_inf = ser == np.inf
740     df.loc[mask_inf, col] = 0
741     assert (df[col] == np.inf).sum() == 0
742     return df
743
744 #df_final_2['rho'] = compute_rho(df_final_2.incidencia)
745 df_final_2 = clean_inf(df_final_2, col="rho")
746
747 df_final_2
748
749 df_final_2.to_parquet("/content/drive/MyDrive/Colab Notebooks/MODELLING
    WEEK/tablon.parquet")
```

A.2 Simpler models

```
1  # -*- coding: utf-8 -*-
2  """01_Simple_Models.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1y-QfI6bozioF37gWxzRkb-
           D1Nf3XpyCt
8  """
9
10 from google.colab import drive
11 drive.mount('/content/drive', force_remount=True)
12
13 """# Importing the packages"""
14
15 # Import packages
16 import pandas as pd
17 import matplotlib.pyplot as plt
18 import numpy as np
19 import plotly
20 import plotly.graph_objects as go
21 import plotly.offline as offline
```

```
from plotly.graph_objs import *
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
    iplot
import json
import plotly.express as px
import matplotlib.dates as mdates
import datetime as dt
from scipy import signal
import time
from sklearn.preprocessing import StandardScaler
import operator
import copy
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.preprocessing import StandardScaler
from sklearn.tree import export_graphviz
from graphviz import Source
import os
import statsmodels.api as sm
from scipy import stats
from sklearn.linear_model import Lasso
from sklearn.model_selection import RepeatedKFold, GridSearchCV
from sklearn.linear_model import RidgeCV
from sklearn.externals.six import StringIO
import pydot
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance

"""<break>

# Loading the data
"""

path = "/content/drive/MyDrive/Colab Notebooks/MODELLING WEEK"

df = pd.read_parquet(path + '/tablon.parquet')
ser_day = df["fecha"].dt.dayofyear / (2*np.pi)
df["sin_date"],df["cos_date"] = np.sin(ser_day), np.cos(ser_day)
df

"""<break>

# Crossed correlations

We are interested in knowing the crossed correlations between each pair of
    variables (different from a territory-related identificator) and for
```

```
    each province. We start defining the variables of study.
"""

provs = df.provincia.unique()
cods = [i for i in range(1,51)]
dict_cod_provs = dict(zip(cods,provs))

dict_cod_provs

col_corr = list(df.columns)
remove = ['cod_prov','ccaa','provincia','poblacion','fecha','poblacion_ccaa
    ','cod_ccaa','num_casos','num_hosp','num_uci','num_def','incidencia','
    sin_date','cos_date','rho'] #+'rho'
col_corr = [i for i in col_corr if i not in remove]

"""Since the crossed correlation function is not defined in a suitable way
    for our propourses we are going to create a function *xcorr* which
    computes the cross correlation and returns its possible values for each
    lag. Then, another function is considered, capable of appling *xcorr* to
     the different (province,var1,var2)-uples. If the user wants, the
    different graphs of crossed correlation can also be displayed."""

def xcorr(x,y,lags=14):
    fig = plt.xcorr(x, y, normed = True, maxlags = lags)
    corr = fig[1]
    lags = fig[0]
    return lags, corr, fig

df_aux = df[df.cod_prov==1]
x = df_aux[col_corr[0]]
y = df_aux[col_corr[1]]
lags, corr, fig = xcorr(x,y)
len(df_aux)
corr

def xcorrprov(province,display,lags=14):
    df_aux = df[df.cod_prov==province]
    matrizcorr = np.zeros((len(col_corr),len(col_corr),2*lags+1))
    for i in range(len(col_corr)):
        for j in range(i+1,len(col_corr)):
            dfcorr = df_aux[[col_corr[i],col_corr[j]]].astype(float)
            lags, corr, fig = xcorr(dfcorr[col_corr[i]], dfcorr[col_corr[j
                ]])
            matrizcorr[i,j,:], matrizcorr[j,i,:] = corr, corr
            if display == 1:
              plt.title('Correlación cruzada entre '+ col_corr[i] + ' y ' +
                  col_corr[j] + ' para la provincia ' + str(province))
              plt.show(fig)
```

```
106             else:
107               plt.close()
108     return matrizcorr
109
110 matrizsol = xcorrprov(31,0)
111 matrizsol
112
113 """Matrix solution has the maximum value of crossed correlation between
    variables $i$ and $j$ in the position $M_{ij}$. The simmetrical
    corresponds to the lag value where the maximum is met. We now generate a
     function that, given a list of provinces, generates its matrix of
    crossed correlations and removes (one of the) variables if $M_{ij}>0.7$.
     """
114
115 def varselect(provincia, lag=14, maxlags=14):
116       varsel = copy.deepcopy(col_corr)
117       variablesrem = []
118       matrizcorr = xcorrprov(provincia,0,maxlags)
119       matrizlag = matrizcorr[:,:,lag+14]
120       size = len(matrizlag)
121       fin = False
122       while not fin:
123         unique, counts = np.unique(np.where(matrizlag>0.7)[0],
              return_counts=True)
124         if unique.size != 0:
125           dic = dict(zip(unique, counts))
126           (idx, maxi) = max(dic.items(), key=operator.itemgetter(1))
127           idx = int(idx)
128           matrizlag[idx,:] = np.repeat(-100, size)
129           matrizlag[:,idx] = np.repeat(-100, size)
130           variablesrem += [col_corr[idx]]
131         else:
132           fin = True
133       return [i for i in varsel if i not in variablesrem]
134
135 varselect(1)
136
137 def create_lags(ser, gap = 14, lag = 1):
138   X = []
139   for i in range(-gap, -gap-lag, -1):
140     ser_ = ser.shift(-i)
141     ser_.name = f'{ser.name}_{-i}'
142     X.append(ser_)
143   dff = pd.concat(X, axis=1)
144   return dff
145
146 """<break>
147
```

```
148 # Linear Regression
149 """
150
151 def generateXy(provincia, vars, x_gap = 14, y_gap = 7, x_lag = 1, y_lag =
        14, objetivo = 'incidencia'):
152   df_temp = df[df.cod_prov==provincia].reset_index(drop=True)
153   df_temp['T'] = df_temp.index
154   df_temp.set_index('fecha',drop=True, inplace = True)
155   y = df_temp[objetivo]
156   x = pd.DataFrame({})
157   for var in vars:
158     x_ = create_lags(df_temp[var], x_gap, x_lag)
159     x = pd.concat([x, x_], axis=1)
160   #x_ = create_lags(df_temp['rho'], y_gap, y_lag)
161   x = pd.concat([x, x_], axis=1)
162   y_ = create_lags(y, y_gap, y_lag)
163   X =  pd.concat([x, y_], axis=1)
164   i_min = max(x_gap,y_gap) + max(x_lag, y_lag)
165   X = X.iloc[i_min:]
166   y = y.iloc[i_min:]
167   return X, y
168
169 def Xy_test_train(X,y,testsize,random):
170     X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=testsize
          ,random_state=random)
171     return X_train,X_test,y_train,y_test
172
173 def LinearModel(provincia, x_gap = 14,  y_gap = 7, x_lag = 1, y_lag = 14,
        objetivo = 'incidencia',testsize = 0.3,random = 1):
174     vars = varselect(provincia)
175     X,y = generateXy(provincia, vars, x_gap, y_gap, x_lag, y_lag, objetivo)
176     X_train,X_test,y_train,y_test = Xy_test_train(X,y,testsize,random)
177     model1 = LinearRegression(normalize = True, n_jobs = -1)
178     reg1 = model1.fit(X_train,y_train)
179     #model2 = sm.OLS(y_train,X_train)
180     #reg2 = model2.fit()
181     return X_train, X_test, y_train, y_test, reg1, #reg2
182
183 def LinearModelProvinces(provinces, x_gap = 14,  y_gap = 7, x_lag = 1,
        y_lag = 14, objetivo = 'incidencia',testsize = 0.3,random = 1,display =
        1):
184     results_lm = []
185     for i in provinces:
186         X_train, X_test, y_train, y_test, reg = LinearModel(i, x_gap=14,
              y_gap=7, x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize
              = 0.3,random = 1)
187         dic_coefficients = dict(zip(X_train.columns,reg.coef_))
188         dic_coefficients['intercept'] = reg.intercept_
```

```
189             dic_coefficients['MSE'] = np.average((reg.predict(X_test)-y_test)
                    **2)
190             dic_coefficients['Score'] = reg.score(X_test,y_test)
191             dic_coefficients['cod_prov'] = i
192             results_lm.append(dic_coefficients)
193             if display == 1:
194               X = pd.concat([X_train,X_test])
195                plt.figure(figsize=(16,5))
196                plt.plot(df[df['cod_prov']==50].fecha,df[df['cod_prov']==50][
                      objetivo],c='g',label=objetivo.capitalize() )
197                plt.scatter(X.index,reg.predict(X),label='Approximation')
198                plt.xlabel('Date')
199                plt.ylabel(objetivo.capitalize())
200                plt.title('Evolution of ' + objetivo + ' for '+ dict_cod_provs[i]
                      + ' (LM)', fontsize=18)
201                plt.legend(loc='upper right')
202                plt.show()
203         return results_lm
204
205 results_lm = LinearModelProvinces(provinces = range(1,51), x_gap = 14,
        y_gap = 7, x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize =
        0.3,random = 1,display = 1)
206
207 i = 49
208 D = results_lm[49]
209 plt.figure(figsize=(20, 10))
210 plt.bar(range(len(D)-5), list(D.values())[:-5], align='center')
211 plt.grid(color='grey', linestyle='-', linewidth=0.1)
212 plt.xticks(range(len(D)-5), list(D.keys())[:-5], rotation='vertical',
        fontsize=12)
213 plt.xlabel('Indicators', fontsize=18)
214 plt.ylabel('Significance', fontsize=18)
215 plt.title('Indicator significance in Covid-19 incidence in '+
        dict_cod_provs[i+1], fontsize=18)
216
217 results_lm
218
219 """<break>
220
221 # Lasso Regression
222 """
223
224 def LassoModel(provincia, x_gap=14, y_gap=7, x_lag = 1, y_lag = 14,
        objetivo = 'incidencia',testsize = 0.3,random = 1):
225     X,y = generateXy(provincia, col_corr, x_gap, y_gap, x_lag, y_lag,
            objetivo)
226     X_train,X_test,y_train,y_test = Xy_test_train(X,y,testsize,random)
227     ## Lasso inicial
```

```
228     reg = Lasso(alpha=0.25,normalize=True,max_iter=10**5)
229     reg.fit(X_train, y_train)
230     ## Intento mejora 1
231     # model = Lasso(normalize=True,max_iter=10**5)
232     # cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
233     # grid = dict()
234     # grid['alpha'] = np.arange(0, 1, 0.01)
235     # search = GridSearchCV(model, grid, scoring='neg_mean_absolute_error',
            cv=cv, n_jobs=-1)
236     # results = search.fit(X, y)
237     # print('MAE: %.3f' % results.best_score_)
238     # print('Config: %s' % results.best_params_)
239     ## Intento mejora 2
240     # modelo = RidgeCV(
241     #          alphas          = np.logspace(-10, 2, 200),
242     #          fit_intercept   = True,
243     #          normalize       = True,
244     #          store_cv_values = True
245     #       )
246     # reg = modelo.fit(X = X_train, y = y_train)
247     return X_train, X_test, y_train, y_test, reg
248
249 def LassoModelProvinces(provinces = range(1,51), x_gap = 14,  y_gap = 7,
        x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize = 0.3,random =
        1,display = 1):
250     results_lasso = []
251     for i in provinces:
252         X_train, X_test, y_train, y_test, reg = LassoModel(i, x_gap=14,
              y_gap=7, x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize
               = 0.3,random = 1)
253         dic_coefficients = dict(zip(X_train.columns,reg.coef_))
254         dic_coefficients['Lasso Regression: R^2 score on training set'] =
              reg.score(X_test, y_test)*100
255         dic_coefficients['Lasso Regression: R^2 score on test set'] = reg.
              score(X_test, y_test)*100
256         dic_coefficients['MSE'] = np.average((reg.predict(X_test)-y_test)
              **2)
257         dic_coefficients['cod_prov'] = i
258         results_lasso.append(dic_coefficients)
259         if display == 1:
260           X = pd.concat([X_train,X_test])
261           plt.figure(figsize=(16,5))
262           plt.plot(df[df['cod_prov']==50].fecha,df[df['cod_prov']==50][
                objetivo],c='g',label=objetivo.capitalize() )
263           plt.scatter(X.index,reg.predict(X),label='Approximation')
264           plt.xlabel('Date')
265           plt.ylabel(objetivo.capitalize())
266           plt.title('Evolution of ' + objetivo + ' for '+ dict_cod_provs[i]
```

```
                + ' (Lasso)', fontsize=18)
267             plt.legend(loc='upper right')
268             plt.show()
269         return results_lasso
270
271 results_lasso = LassoModelProvinces(provinces = range(1,51), x_gap = 14,
        y_gap = 7, x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize =
        0.3,random = 1,display = 1)
272
273 i = 49
274 D = results_lasso[49]
275 plt.figure(figsize=(20, 10))
276 plt.bar(range(len(D)-5), list(D.values())[:-5], align='center')
277 plt.grid(color='grey', linestyle='-', linewidth=0.1)
278 plt.xticks(range(len(D)-5), list(D.keys())[:-5], rotation='vertical',
        fontsize=12)
279 plt.xlabel('Indicators', fontsize=18)
280 plt.ylabel('Significance', fontsize=18)
281 plt.title('Indicator significance in Covid-19 incidence in '+
        dict_cod_provs[i+1], fontsize=18)
282
283 results_lasso
284
285 """<break>
286
287 # Interrupted time series + Lasso
288 """
289
290 col_interrupted = col_corr + ['T','sin_date','cos_date']
291
292 def InterruptedModel(provincia,x_gap=14, y_gap=7, x_lag = 1, y_lag = 14,
        objetivo = 'incidencia',testsize = 0.3,random = 1):
293     X,y = generateXy(provincia, col_interrupted, x_gap, y_gap, x_lag, y_lag
            , objetivo)
294     X_train,X_test,y_train,y_test = Xy_test_train(X,y,testsize,random)
295     reg = Lasso(alpha=0.25,normalize=True,max_iter=10**5)
296     reg.fit(X_train, y_train)
297     return X_train, X_test, y_train, y_test, reg
298
299 def InterruptedModelProvinces(provinces = range(1,51), x_gap = 14,  y_gap =
        7, x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize = 0.3,random
        = 1,display = 1):
300     results_interrumped = []
301     for i in provinces:
302         X_train, X_test, y_train, y_test, reg = InterruptedModel(i, x_gap,
                y_gap, x_lag, y_lag, objetivo,testsize,random)
303         dic_coefficients = dict(zip(X_train.columns,reg.coef_))
304         dic_coefficients['Interrumped Regression: R^2 score on training set
```

```
            '] = reg.score(X_test, y_test)*100
305         dic_coefficients['Interrumped Regression: R^2 score on test set'] =
                reg.score(X_test, y_test)*100
306         dic_coefficients['MSE'] = np.average((reg.predict(X_test)-y_test)
                **2)
307         dic_coefficients['Score'] = reg.score(X_test,y_test)
308         dic_coefficients['cod_prov'] = i
309         results_interrumped.append(dic_coefficients)
310         if display == 1:
311           X = pd.concat([X_train,X_test])
312           plt.figure(figsize=(16,5))
313           plt.plot(df[df['cod_prov']==50].fecha,df[df['cod_prov']==50][
                objetivo],c='g',label=objetivo.capitalize() )
314           plt.scatter(X.index,reg.predict(X),label='Approximation')
315           plt.xlabel('Date')
316           plt.ylabel(objetivo.capitalize())
317           plt.title('Evolution of ' + objetivo + ' for '+ dict_cod_provs[i]
                + ' (ITS)', fontsize=18)
318           plt.legend(loc='upper right')
319           plt.show()
320     return results_interrumped
321
322 results_interrumped = InterruptedModelProvinces(provinces = range(1,51),
    x_gap = 14,  y_gap = 7, x_lag = 1, y_lag = 14, objetivo = 'incidencia',
    testsize = 0.3,random = 1,display = 1)
323
324 results_interrumped
325
326 i = 49
327 D = results_interrumped[49]
328 plt.figure(figsize=(20, 10))
329 plt.bar(range(len(D)-5), list(D.values())[:-5], align='center')
330 plt.grid(color='grey', linestyle='-', linewidth=0.1)
331 plt.xticks(range(len(D)-5), list(D.keys())[:-5], rotation='vertical',
    fontsize=12)
332 plt.xlabel('Indicators', fontsize=18)
333 plt.ylabel('Significance', fontsize=18)
334 plt.title('Indicator significance in Covid-19 incidence in '+
    dict_cod_provs[i+1], fontsize=18)
335
336 """<break>
337
338 # Decision Tree Regressor
339 """
340
341 def FeatureImportances(model, X_val, y_val, n_repeats=30, random_state=0):
342     r = permutation_importance(model, X_val, y_val, n_repeats=30,
            random_state=0)
```

```
343     variables = []
344     weigths = []
345     uncertainty = []
346     for i in r.importances_mean.argsort()[::-1]:
347         if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
348           variables.append(X_val.columns[i])
349           weigths.append(r.importances_mean[i])
350           uncertainty.append(r.importances_std[i])
351     return variables, weigths, uncertainty
352 
353 def DecisionTreeReg(X_train,y_train,X_test,y_test,vars,testsize):
354     DecisionTree = tree.DecisionTreeRegressor(random_state=0)
355     DecisionTree.fit(X_train,y_train)
356     y_pred = DecisionTree.predict(X_test)
357     return y_pred, DecisionTree
358 
359 def TreeModelProvince(provincia, testsize, random, x_gap=14, y_gap=7, x_lag
        = 1, y_lag = 14, objetivo = 'incidencia'):
360     X,y = generateXy(provincia, col_corr, x_gap, y_gap, x_lag, y_lag,
            objetivo)
361     X_train,X_test,y_train,y_test = Xy_test_train(X,y,testsize,random)
362     y_pred, DecisionTree = DecisionTreeReg(X_train,y_train,X_test,y_test,
            objetivo,testsize)
363     tree.export_graphviz(DecisionTree, out_file='tree.dot', feature_names=
            X_train.columns)
364     return y_pred, y_test, DecisionTree, X_train, X_test, Source.from_file(
            'tree.dot')
365 
366 def DecisionTreeProvinces(provinces = range(1,51), x_gap = 14,  y_gap = 7,
      x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize = 0.3,random =
      1,display = 1):
367     results_tree = []
368     uncertanty_tree = []
369     for i in provinces:
370         y_pred, y_test, DecisionTree, X_train, X_test, drawing =
              TreeModelProvince(i,testsize,random, x_gap, y_gap, x_lag, y_lag,
              objetivo)
371         namefile = 'DecisionTree_prov_'+str(i)
372         drawing.render(filename=namefile)
373         #weigths = DecisionTree.feature_importances_
374         variables, weigths, uncertainty = FeatureImportances(DecisionTree,
              X_test, y_pred, n_repeats=30, random_state=0)
375         #dic_coefficients = dict(zip(X_train.columns,weigths))
376         dic_coefficients = dict(zip(variables,weigths))
377         dic_uncertainty = dict(zip(variables,uncertainty))
378         dic_coefficients['MSE'] = np.average((y_pred-y_test)**2)
379         dic_coefficients['cod_prov'] = i
380         results_tree.append(dic_coefficients)
```

```
381             uncertanty_tree.append(dic_uncertainty)
382             if display == 1:
383               X = pd.concat([X_train,X_test])
384               plt.figure(figsize=(16,5))
385               plt.plot(df[df['cod_prov']==50].fecha,df[df['cod_prov']==50][
                      objetivo],c='g',label=objetivo.capitalize() )
386               plt.scatter(X.index,DecisionTree.predict(X),label='Approximation'
                      )
387               plt.xlabel('Date')
388               plt.ylabel(objetivo.capitalize())
389               plt.title('Evolution of ' + objetivo + ' for '+ dict_cod_provs[i]
                      + ' (Tree)', fontsize=18)
390               plt.legend(loc='upper right')
391               plt.show()
392         return results_tree, uncertanty_tree
393
394 results_tree, uncertanty_tree = DecisionTreeProvinces(provinces = range
        (1,51), x_gap = 14,  y_gap = 7, x_lag = 1, y_lag = 14, objetivo = '
        incidencia',testsize = 0.3,random = 1,display = 1)
395
396 results_tree
397
398 uncertanty_tree
399
400 i = 49
401 D = results_tree[49]
402 plt.figure(figsize=(20, 10))
403 plt.bar(range(len(D)-5), list(D.values())[:-5], align='center')
404 plt.grid(color='grey', linestyle='-', linewidth=0.1)
405 plt.xticks(range(len(D)-5), list(D.keys())[:-5], rotation='vertical',
        fontsize=12)
406 plt.xlabel('Indicators', fontsize=18)
407 plt.ylabel('Weights', fontsize=18)
408 plt.title('Indicator Weights in Covid-19 incidence in '+ dict_cod_provs[i
        +1], fontsize=18)
409
410 """<break>
411
412 # Random Forest Regressor
413 """
414
415 def RandomForestReg(X_train,y_train,X_test,y_test,vars,testsize):
416     RandomForest = RandomForestRegressor(random_state=0, n_jobs=-1)
417     RandomForest.fit(X_train,y_train)
418     y_pred = RandomForest.predict(X_test)
419     return y_pred, RandomForest
420
421 def RandomForestProvince(provincia, testsize, random, x_gap=14, y_gap=7,
```

```
    x_lag = 1, y_lag = 14, objetivo = 'incidencia'):
422     X,y = generateXy(provincia, col_corr, x_gap, y_gap, x_lag, y_lag,
            objetivo)
423     X_train,X_test,y_train,y_test = Xy_test_train(X,y,testsize,random)
424     y_pred, RandomForest = RandomForestReg(X_train,y_train,X_test,y_test,
            objetivo,testsize)
425     return y_pred, y_test, RandomForest, X_train, X_test
426
427 def RandomForestProvinces(provinces = range(1,51), x_gap = 14,  y_gap = 7,
    x_lag = 1, y_lag = 14, objetivo = 'incidencia',testsize = 0.3,random =
    1,display = 1):
428     results_forest = []
429     uncertanty_forest = []
430     for i in provinces:
431         y_pred, y_test, RandomForest, X_train, X_test =
                RandomForestProvince(i,0.3,1, x_gap=14, y_gap=7, x_lag = 1,
                y_lag = 14 ,objetivo = 'incidencia')
432         #weigths = RandomForest.feature_importances_
433         variables, weigths, uncertainty = FeatureImportances(RandomForest,
                X_test, y_pred, n_repeats=30, random_state=0)
434         #dic_coefficients = dict(zip(X_train.columns,weigths))
435         dic_coefficients = dict(zip(variables,weigths))
436         dic_uncertainty = dict(zip(variables,uncertainty))
437         dic_coefficients['MSE'] = np.average((y_pred-y_test)**2)
438         dic_coefficients['cod_prov'] = i
439         results_forest.append(dic_coefficients)
440         uncertanty_forest.append(dic_uncertainty)
441         if display == 1:
442           X = pd.concat([X_train,X_test])
443           plt.figure(figsize=(16,5))
444           plt.plot(df[df['cod_prov']==50].fecha,df[df['cod_prov']==50][
                objetivo],c='g',label=objetivo.capitalize() )
445           plt.scatter(X.index,RandomForest.predict(X),label='Approximation'
                )
446           plt.xlabel('Date')
447           plt.ylabel(objetivo.capitalize())
448           plt.title('Evolution of ' + objetivo + ' for '+ dict_cod_provs[i]
                + ' (Forest)', fontsize=18)
449           plt.legend(loc='upper right')
450           plt.show()
451     return results_forest, uncertanty_forest
452
453 results_forest, uncertanty_forest = RandomForestProvinces(provinces = range
    (1,51), x_gap = 14,  y_gap = 7, x_lag = 1, y_lag = 14, objetivo = '
    incidencia',testsize = 0.3,random = 1,display = 1)
454
455 results_forest
456
```

```
457 uncertanty_forest
458
459 i = 49
460 D = results_forest[49]
461 plt.figure(figsize=(20, 10))
462 plt.bar(range(len(D)-5), list(D.values())[:-5], align='center')
463 plt.grid(color='grey', linestyle='-', linewidth=0.1)
464 plt.xticks(range(len(D)-5), list(D.keys())[:-5], rotation='vertical',
        fontsize=12)
465 plt.xlabel('Indicators', fontsize=18)
466 plt.ylabel('Weights', fontsize=18)
467 plt.title('Indicator weights in Covid-19 incidence in '+ dict_cod_provs[i
        +1], fontsize=18)
468
469 """<break>
470
471 # Graphs of results
472 """
473
474 keys_ = results_lasso[0].keys()
475 keys_ = list(keys_)
476 df_values = pd.DataFrame(results_lasso)
477 df_values.describe()
478
479 df_values = pd.DataFrame(results_lasso)
480 df_values
```

A.3 Deep learning approach

```
1 # -*- coding: utf-8 -*-
2 """02_RNN.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1fHMX8iq7nrsjCGwhD8O7Gl-2
          xrmvEidL
8
9 # Configuración del entorno
10
11 ## TODO
12 - [ ] Predecir de forma secuencial (hay que ir cambiando la X con cada
       prediccion). Moderado
13 - [ ] Utilizar para Y y X de incidencia/rho un moving average para suavizar
       (14 días). Trivial
14
15 Importamos las librerías necesarias.
```

```
"""

!pip install shap

# Commented out IPython magic to ensure Python compatibility.
# Importamos las librerias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras import Sequential, layers, callbacks
from tensorflow.keras.layers import Dense, LSTM, Dropout,  Bidirectional
import math
from scipy import stats

#import shap


# %matplotlib inline
import math

import seaborn as sns

import shap
shap.initjs()
print(tf.__version__)

"""Fijamos la semilla para poder obtener los mismos resultados cuando ejecutemos un mismo código en diferentes momentos del tiempo.

Uno de los motivos por el cual una red neuronal puede no funcionar bien es por la semilla. Si se obtienen resultados que no son buenos habría que probar a cambiar la semilla por si la red ha alcanzado un mínimo local y no un mínimo global.
"""

# Fijamos la semilla
np.random.seed(7588)
tf.random.set_seed(7588)

"""# Lectura de los datos"""

# Conexión a Drive donde se encuentran los datos
```

```
60 from google.colab import drive
61 drive.mount('/content/drive')
62
63 """# Análisis gráfico de los datos de entrada"""
64
65 # for i in range(1,51):
66 #   df_prov=df[df.cod_prov==i]
67 #   plt.figure(figsize=(20,10))
68 #   nombre_provincia=df_prov['provincia'].unique().tolist()
69 #   plt.title("num_casos {}".format(nombre_provincia))
70 #   df_prov['num_casos'].plot()
71 #   plt.show()
72
73 # for i in range(1,51):
74 #   df_prov=df[df.cod_prov==i]
75 #   plt.figure(figsize=(20,10))
76 #   nombre_provincia=df_prov['provincia'].unique().tolist()
77 #   plt.title("num_hosp {}".format(nombre_provincia))
78 #   df_prov['num_hosp'].plot()
79 #   plt.show()
80
81 # for i in range(1,51):
82 #   df_prov=df[df.cod_prov==i]
83 #   plt.figure(figsize=(20,10))
84 #   nombre_provincia=df_prov['provincia'].unique().tolist()
85 #   plt.title("num_uci {}".format(nombre_provincia))
86 #   df_prov['num_uci'].plot()
87 #   plt.show()
88
89 # for i in range(1,51):
90 #   df_prov=df[df.cod_prov==i]
91 #   plt.figure(figsize=(20,10))
92 #   nombre_provincia=df_prov['provincia'].unique().tolist()
93 #   plt.title("num_def {}".format(nombre_provincia))
94 #   df_prov['num_def'].plot()
95 #   plt.show()
96
97 # for i in range(1,51):
98 #   df_prov=df[df.cod_prov==i]
99 #   plt.figure(figsize=(20,10))
100 #   nombre_provincia=df_prov['provincia'].unique().tolist()
101 #   plt.title("personas_con_pauta_completa {}".format(nombre_provincia))
102 #   df_prov['personas_con_pauta_completa'].plot()
103 #   plt.show()
104
105 # for i in range(1,51):
106 #   df_prov=df[df.cod_prov==i]
107 #   plt.figure(figsize=(20,10))
```

```
108 #    nombre_provincia=df_prov['provincia'].unique().tolist()
109 #    plt.title("poblacion_vac {}". format(nombre_provincia))
110 #    df_prov['poblacion_vac'].plot()
111 #    plt.show()
112
113 # for i in range(1,51):
114 #    df_prov=df[df.cod_prov==i]
115 #    plt.figure(figsize=(20,10))
116 #    nombre_provincia=df_prov['provincia'].unique().tolist()
117 #    plt.title("retail_and_recreation_percent_change_from_baseline {}".
    format(nombre_provincia))
118 #    df_prov['retail_and_recreation_percent_change_from_baseline'].plot()
119 #    plt.show()
120
121 # for i in range(1,51):
122 #    df_prov=df[df.cod_prov==i]
123 #    plt.figure(figsize=(20,10))
124 #    nombre_provincia=df_prov['provincia'].unique().tolist()
125 #    plt.title("grocery_and_pharmacy_percent_change_from_baseline {}".
    format(nombre_provincia))
126 #    df_prov['grocery_and_pharmacy_percent_change_from_baseline'].plot()
127 #    plt.show()
128
129 # for i in range(1,51):
130 #    df_prov=df[df.cod_prov==i]
131 #    plt.figure(figsize=(20,10))
132 #    nombre_provincia=df_prov['provincia'].unique().tolist()
133 #    plt.title("parks_percent_change_from_baseline {}". format(
    nombre_provincia))
134 #    df_prov['parks_percent_change_from_baseline'].plot()
135 #    plt.show()
136
137 # for i in range(1,51):
138 #    df_prov=df[df.cod_prov==i]
139 #    plt.figure(figsize=(20,10))
140 #    nombre_provincia=df_prov['provincia'].unique().tolist()
141 #    plt.title("transit_stations_percent_change_from_baseline {}". format(
    nombre_provincia))
142 #    df_prov['transit_stations_percent_change_from_baseline'].plot()
143 #    plt.show()
144
145 # for i in range(1,51):
146 #    df_prov=df[df.cod_prov==i]
147 #    plt.figure(figsize=(20,10))
148 #    nombre_provincia=df_prov['provincia'].unique().tolist()
149 #    plt.title("workplaces_percent_change_from_baseline {}". format(
    nombre_provincia))
150 #    df_prov['workplaces_percent_change_from_baseline'].plot()
```

```
151 #    plt.show()
152
153 # for i in range(1,51):
154 #    df_prov=df[df.cod_prov==i]
155 #    plt.figure(figsize=(20,10))
156 #    nombre_provincia=df_prov['provincia'].unique().tolist()
157 #    plt.title("residential_percent_change_from_baseline {}".format(
        nombre_provincia))
158 #    df_prov['residential_percent_change_from_baseline'].plot()
159 #    plt.show()
160
161 # for i in range(1,51):
162 #    df_prov=df[df.cod_prov==i]
163 #    plt.figure(figsize=(20,10))
164 #    nombre_provincia=df_prov['provincia'].unique().tolist()
165 #    plt.title("deporte_exterior {}".format(nombre_provincia))
166 #    df_prov['deporte_exterior'].plot()
167 #    plt.show()
168
169 # for i in range(1,51):
170 #    df_prov=df[df.cod_prov==i]
171 #    plt.figure(figsize=(20,10))
172 #    nombre_provincia=df_prov['provincia'].unique().tolist()
173 #    plt.title("deporte_interior {}".format(nombre_provincia))
174 #    df_prov['deporte_interior'].plot()
175 #    plt.show()
176
177 # for i in range(1,51):
178 #    df_prov=df[df.cod_prov==i]
179 #    plt.figure(figsize=(20,10))
180 #    nombre_provincia=df_prov['provincia'].unique().tolist()
181 #    plt.title("cultura {}".format(nombre_provincia))
182 #    df_prov['cultura'].plot()
183 #    plt.show()
184
185 # for i in range(1,51):
186 #    df_prov=df[df.cod_prov==i]
187 #    plt.figure(figsize=(20,10))
188 #    nombre_provincia=df_prov['provincia'].unique().tolist()
189 #    plt.title("restauracion_exterior {}".format(nombre_provincia))
190 #    df_prov['restauracion_exterior'].plot()
191 #    plt.show()
192
193 # for i in range(1,51):
194 #    df_prov=df[df.cod_prov==i]
195 #    plt.figure(figsize=(20,10))
196 #    nombre_provincia=df_prov['provincia'].unique().tolist()
197 #    plt.title("restauracion_interior {}".format(nombre_provincia))
```

```
#    df_prov['restauracion_interior'].plot()
#    plt.show()

# for i in range(1,51):
#    df_prov=df[df.cod_prov==i]
#    plt.figure(figsize=(20,10))
#    nombre_provincia=df_prov['provincia'].unique().tolist()
#    plt.title("movilidad {}". format(nombre_provincia))
#    df_prov['movilidad'].plot()
#    plt.show()

"""# Pre-processing RNN

"""

# Lectura de los datos con read_csv
# Creamos un dataframe donde el indice es la Fecha
df=pd.read_parquet(r'/content/drive/MyDrive/Colab Notebooks/MODELLING WEEK/tablon.parquet')
#df=pd.read_parquet(r'/content/drive/MyDrive/COVID19/tablon.parquet' )
df=df.fillna(0)
df=df.drop(['rho'], axis=1)
df.head()

df2 = df.pivot_table(index=['fecha'],columns=['cod_prov'],values=df.drop(['fecha','cod_prov','cod_ccaa'],axis=1))
df2.head()

df['cod_prov_text'] = df['cod_prov'].astype('str')
df2 = df.pivot_table(index=['fecha'],
                     columns=['cod_prov_text'],
                     values=df.drop(['fecha','cod_prov','cod_prov_text','cod_ccaa','ccaa','provincia'],axis=1))
df2 = df2.reset_index()
df2.columns = df2.columns.map('_'.join)
drop_col=['fecha_']
df2=df2.set_index(drop_col)

df2.head()

name_vars_object = [name for name,tipo in df2.dtypes.iteritems() if 'object' in str(tipo)]
name_vars_float = [name for name, tipo in df2.dtypes.iteritems() if 'float' in str(tipo)]
name_vars_int = [name for name, tipo in df2.dtypes.iteritems() if 'int' in str(tipo)]

name_vars_float
```

```
240
241 """### Define features
242
243
244 """
245
246 # Make a copy of the dataframe to start over from this cell as needed,
      without downloading CSV again
247 df = df2.copy()
248
249 FEATURES = [ 'cultura_28',
250   'deporte_interior_28',
251   'grocery_and_pharmacy_percent_change_from_baseline_28',
252   'movilidad_28',
253   'parks_percent_change_from_baseline_28',
254   'personas_con_pauta_completa_28',
255   'poblacion_vac_28',
256   'residential_percent_change_from_baseline_28',
257   'restauracion_exterior_28',
258   'restauracion_interior_28',
259   'retail_and_recreation_percent_change_from_baseline_28',
260   'transit_stations_percent_change_from_baseline_28',
261   'workplaces_percent_change_from_baseline_28']
262 LABEL = 'incidencia_28'
263 SELECTED_COLUMNS = FEATURES + [LABEL]
264 df=df[SELECTED_COLUMNS]
265
266 """## Describe data"""
267
268 # Extract labels
269 df_train = df
270 train_labels = df_train.pop(LABEL)
271
272 # Describe the dataset
273 train_stats = df_train.describe()
274 train_stats = train_stats.transpose()
275 train_stats
276
277 """## Normalize data"""
278
279 def norm(x):
280   return (x - train_stats['mean']) / train_stats['std']
281
282 df_train_normed = norm(df_train)
283
284 df_train_normed.head()
285
286 corr = df.corr()
```

```
287 sns.heatmap(corr)
288
289 """## Build model"""
290
291 def build_model(df):
292   model = keras.Sequential([
293     layers.Dense(16, activation=tf.nn.relu, input_shape=[len(df.keys())]),
294     layers.Dense(16, activation=tf.nn.relu),
295     layers.Dense(1)
296   ])
297
298   # TF 2.0: optimizer = tf.keras.optimizers.RMSprop()
299   optimizer = tf.keras.optimizers.RMSprop()
300   # optimizer = tf.train.RMSPropOptimizer(learning_rate=0.001)
301
302   model.compile(loss='mse',
303                 optimizer=optimizer,
304                 metrics=['mae'])
305   return model
306
307 model = build_model(df_train_normed)
308 model.summary()
309
310 """## Train model"""
311
312 # Display training progress by printing a single dot for each completed
      epoch
313 class PrintDot(keras.callbacks.Callback):
314   def on_epoch_end(self, epoch, logs):
315     if epoch % 100 == 0: print('')
316     print('.', end='')
317
318 EPOCHS = 1000
319
320 early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=50)
321
322 history = model.fit(
323   df_train_normed, train_labels,
324   epochs=EPOCHS, validation_split = 0.2, verbose=2,
325   callbacks=[early_stop, PrintDot()])
326
327 hist = pd.DataFrame(history.history)
328 hist['epoch'] = history.epoch
329
330 def plot_history(history):
331   plt.figure()
332   plt.xlabel('Epoch')
333   plt.ylabel('Mean Absolute Error')
```

```
334     plt.plot(hist['epoch'], hist['mae'],
335              label='Train Error')
336     plt.plot(hist['epoch'], hist['val_mae'],
337              label = 'Val Error')
338     plt.legend()
339
340 plot_history(history)
341
342 """## Explain model
343
344 ### KernelExplainer
345 """
346
347 # KernelExplainer is a general approach that can work with any ML framework
348 # Its inputs are the predictions and training data
349
350 # Summarize the training set to accelerate analysis
351 df_train_normed_summary = shap.kmeans(df_train_normed.values, 25)
352
353 # Instantiate an explainer with the model predictions and training data
        summary
354 explainer = shap.KernelExplainer(model.predict, df_train_normed_summary)
355
356 # Extract Shapley values from the explainer
357 shap_values = explainer.shap_values(df_train_normed.values)
358
359 # Summarize the Shapley values in a plot
360 shap.summary_plot(shap_values[0], df_train)
361
362 # Plot the SHAP values for one instance
363 shap.initjs()
364 INSTANCE_NUM = 0
365 shap.force_plot(explainer.expected_value[0], shap_values[0][INSTANCE_NUM],
        df_train.iloc[INSTANCE_NUM,:])
366
367 # Plot the SHAP values for multiple instances
368 shap.initjs()
369 NUM_ROWS = 10
370 shap.force_plot(explainer.expected_value[0], shap_values[0][0:NUM_ROWS],
        df_train.iloc[0:NUM_ROWS])
371
372 # Create a SHAP dependence plot to show the effect of a single feature
        across the whole dataset
373 shap.initjs()
374 shap.dependence_plot('grocery_and_pharmacy_percent_change_from_baseline_28'
        , shap_values[0], df_train, interaction_index='movilidad_28')
```

References

[BCG17] James Lopez Bernal, Steven Cummins, and Antonio Gasparrini. "Interrupted time series regression for the evaluation of public health interventions: a tutorial". In: *International journal of epidemiology* 46.1 (2017), pp. 348–355.

[Alb18] Chris Albon. *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning.* " O'Reilly Media, Inc.", 2018.

[BP20] Vaishak Belle and Ioannis Papantonis. "Principles and practice of explainable machine learning". In: *arXiv preprint arXiv:2009.11698* (2020).

[Cat+20] Martí Català Sabaté et al. "Analysis and prediction of COVID-19 for EU-EFTA-UK and other countries". In: *Daily report; 163* (2020).

[Cow+20] Benjamin J Cowling et al. "Impact assessment of non-pharmaceutical interventions against coronavirus disease 2019 and influenza in Hong Kong: an observational study". In: *The Lancet Public Health* 5.5 (2020), e279–e288.

[Deh+20] Jonas Dehning et al. "Inferring change points in the spread of COVID-19 reveals the effectiveness of interventions". In: *Science* 369.6500 (2020).

[Fla+20] Seth Flaxman et al. "Estimating the effects of non-pharmaceutical interventions on COVID-19 in Europe". In: *Nature* 584.7820 (2020), pp. 257–261.

[Hau+20] Nils Haug et al. "Ranking the effectiveness of worldwide COVID-19 government interventions". In: *Nature human behaviour* 4.12 (2020), pp. 1303–1312.

[Kar+20] Petr Karnakov et al. "Data-driven inference of the reproduction number for COVID-19 before and after interventions for 51 European countries". In: *Swiss medical weekly* 150 (2020), w20313.

[Kuc+20] Adam J Kucharski et al. "Effectiveness of isolation, testing, contact tracing, and physical distancing on reducing transmission of SARS-CoV-2 in different settings: a mathematical modelling study". In: *The Lancet Infectious Diseases* 20.10 (2020), pp. 1151–1160.

[MCB20] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. "Interpretable Machine Learning–A Brief History, State-of-the-Art and Challenges". In: *arXiv preprint arXiv:2010.09337* (2020).

[VP20] Zoltán Vokó and János György Pitter. "The effect of social distance measures on COVID-19 epidemics in Europe: an interrupted time series analysis". In: *GeroScience* 42.4 (2020), pp. 1075–1082.