Predicting Covid-19 pandemic behavior to prevent deaths increase

Students:

Ion Dodon

Dana Speianu

Mirela Verebceanu

Ștefan Tîmbur

Mentor:

Bețliu Corina

Chișinău, 2022

ABSTRACT

This project aims to have in the end a machine learning model that will be able to predict the number of deaths caused by COVID-19 and to analyze how specific factors like *vaccination* affect the increase/decrease in the number of deaths.

In this project-work the data set was taken from another project that gathers data from different sources. The data set was detailed analyzed to see how it can be used for making the models forecast better. The necessary variables were selected, the countries that would make the model biased were excluded and the missing values were replaced with approximations. All the variables were plotted to see their behavior per each country and generally to see what's the pandemic behavior as time passes.

Several types of machine learning models were selected to be tested for making the forecasting. Some of the models are not intended for time series data as it is in this case, but the data set was prepared to be able to be used it with those models.

Introduction

Today, technologies are making remarkable progress and there is an increasing technologicalization of several processes. There are technical solutions for any problems, but these can solve it partially. For our semester project the problem we had to solve is the increasing number of new deaths caused by Covid-19 pandemic. As the scope was to use the Machine Learning we decided to make a model that will predict the number of new deaths in the future. By having these results we can prove and inform people how important is to respect the measures anti pandemic. Also, forecasting the new deaths caused by pandemic for the future, it would have a significant importance as it has a power to change the things and prevent the number of deaths. This can be achieved by reporting these results to the government in order to realize what measures against pandemic they must implement. The following are the main objectives of the project:

- Find and train a model with the best accuracy for a better forecasting.
- Warn people about the need to follow the rules against the pandemic according to the forecasting results.
- Cooperation with the authorities to adjust the stringency according to the forecasting results.
- Decreasing number of deaths caused by pandemic.

The purpose of this project is to analyze the Covid-19 pandemic behavior in different countries around the world and see how it performs according to the characteristics of a state or the regulation set up by the government from a state. The purpose is to detect any specific behavior which known would help to prevent the increase in the number of deaths caused by Covid-19.

During this project were tested several models: VARMAX, Epsilon-Support Vector Regression(SVR), Extreme Gradient Boosting(XGBoost), LightGBM Regressor, Multiple Linear Regression. With VAR-

MAX was obtained the worst result with an R2 score of 0.04%. With SVR better results were obtained even with the default hyper parameters, with a 99% for most of the countries and this is most probably over-fitting. The next steps are to identify what are the causes of over-fitting. With Random Forest Regression, XGBoost and LGBM Regressor we obtained best accuracies between 60%-70%.

1. Data Description

The data was collected from a Github repository open to the general public. The needed information was selected from a CSV file, imported to be analyzed and processed. After an analysis, the file contains data from 65 variables, from the beginning of the pandemic to the first day of November 2021. From the total number of variables only the most relevant were selected, the other ones were not selected for several reasons.

First, many variables contained redundant information that was going to make the model worse. Secondly the other variables contained contained irrelevant data. And the third cause the remaining variables contained missing data. The remaining variables to be used are: total_cases, new_cases, new_deaths, icu_patients, stringency_index, new_tests, positive_rate, people_vaccinated, total_boosters, new_vaccinations, continent, location, date, population, population_density, cardiovasc_death_rate, diabetes_prevalence, human_development_index.

The list below represents the meaning of each variable:

- total_cases - number of total COVID cases.
- new_cases - new confirmed cases of COVID-19.
- new_deaths - new deaths attributed to COVID-19.
- icu_patients - number of COVID-19 patients in intensive care units (ICUs) on a given day.
- stringency_index - government Response Stringency Index: composite measure based on 9 response indicators including school closures, workplace closures, and travel bans, rescaled to a value from 0 to 100 (100 = strictest response).
- new_tests - new tests for COVID-19 (only calculated for consecutive days).
- positive_rate - the share of COVID-19 tests that are positive, given as a rolling 7-day average .

- people_vaccinated - total number of people who received at least one vaccine dose.
- total_boosters - total number of COVID-19 vaccination booster doses administered (doses administered beyond the number prescribed by the vaccination protocol).
- new_vaccinations - new COVID-19 vaccination doses administered (only calculated for consecutive days).
- continent - continent of the geographical location.
- location - geographical location.
- date - date of observation.
- population - population (latest available values).
- population_density - number of people divided by land area, measured in square kilometers, most recent year available.
- cardiovasc_death_rate - death rate from cardiovascular disease in 2017 (annual number of deaths per 100,000 people).
- diabetes_prevalence - diabetes prevalence (% of population aged 20 to 79) in 2017.
- human_development_index - a composite index measuring average achievement in three basic dimensions of human development—a long and healthy life, knowledge and a decent standard of living.

2. Exploratory Data Analysis and Data Cleaning

The dataset is a set of time series, for the same period of time, for different countries around the world. Unfortunately not for all countries there is enough data in our variables. For example, for most of the countries from Asia, there is no data about icu_patients, new_tests, positive_rate, new_vaccinations, total_boosters, stringency_index. The following listing shows two examples of countries from Asia that do not have this data.

Percentage of missing values:

```
...
Afghanistan new_cases 0.0%
Afghanistan new_deaths 4.538087520259319%
Afghanistan icu_patients 100.0%
Bangladesh new_tests 5.582922824302135%
Bangladesh positive_rate 6.568144499178982%
Bangladesh people_vaccinated 70.77175697865353%
...
```

There are few countries from Asia that have the data for the variables mentioned above. The same problem is for the countries from South America, Oceania, Australia, Mexico, Canada, most of the islands, and Africa. It is not possible to replace that much data because if we give approximations, these approximations will not be true and will make the models worse. This is why the mentioned countries and continents will be removed. What remains is some countries from Europe and the United States.

In Europe, there are many countries that are small, called micro countries. These micro countries

have a population of fewer than 500000 people and are small countries in the area. The problem with the micro countries is that they may make the models biased. In a big country, if the stringency_index is high the number of cases will decrease slowly - the effect of stringency_index on the number of cases will be slow. Whereas in a micro country if the stringency_index is high, the number of cases will decrease fast. In micro countries, the behavior of the pandemic is different because of the fact they have a smaller population, and in Europe especially, the population density is high in the micro countries such as Monaco.

To not make the model biased the micro countries from Europe were removed. The following are the 11 removed micro countries:

```
[
'Andorra',
'Faeroe Islands',
'Gibraltar',
'Guernsey',
'Iceland',
'Isle of Man',
'Jersey',
'Liechtenstein',
'Monaco',
'San Marino',
'Vatican'
]
```

After plotting each variable for each country in time, the following things were observed:

- there are negative values
- there are values that increase/decrease drastically unexpectedly
- there are small section (periods of time) where the values are missing

In the following plots are shown these problems. The plots are made per continent and for each continent a separate plot for each variable. The colored lines represent the countries. It can be noticed that for all countries the pandemic has the same behavior, for example, when in the United Kingdom the number of cases increases, in other countries the number of cases also increases. One thing that explains this is that the COVID-19 pandemic also depends on the factors that are common to all countries such as the season.

The following plot represents the icu_patients(Figure 1).

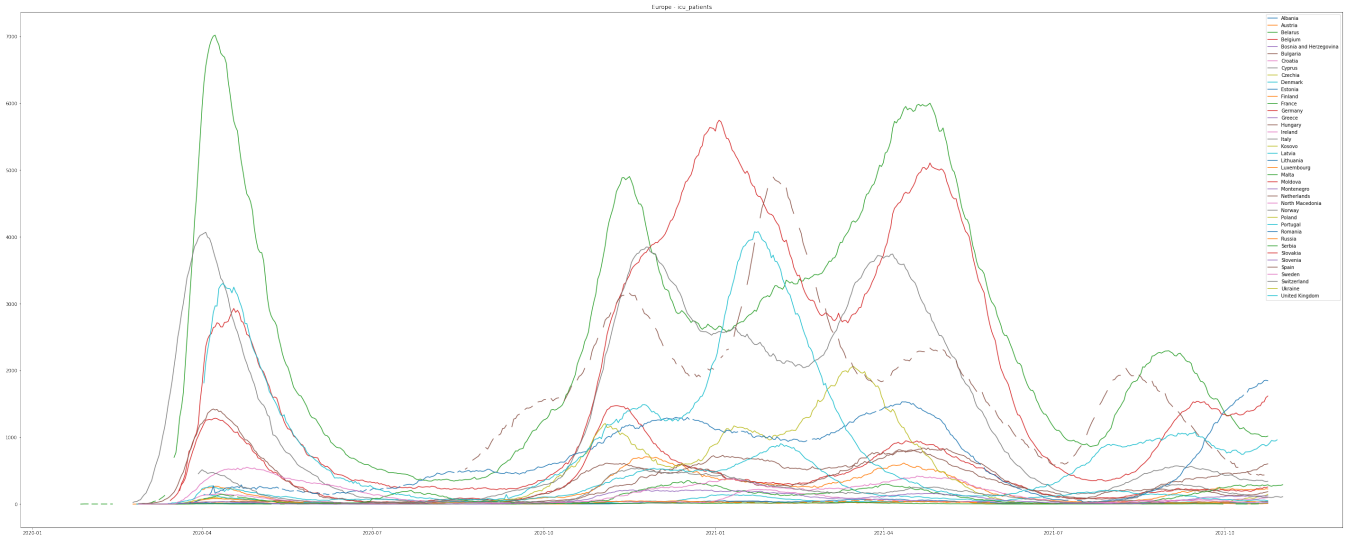


Figure 1. Plot of the icu_patients

We can see there are some countries that have periods with missing data for this variable. To replace these missing values, on the position where a value is missing, the last non-missing values were

added to the nest non-missing values and this sum was divided by two. The resulting value is to replace the current missing value. There are some variables such as new_vaccinations, new_tests that can not have data from the beginning of the pandemic. In the case of vaccination, people were able to vaccinate after at least one year of pandemic. For this kind of variable, the missing values until the first non-missing were replaced with 0. For example, there was 0 vaccination until the day when the first vaccination took place. The following diagram shows the icu_patients variable after replacing the missing values as correcting the anomalies.

There are also anomalies, which means - unexpectedly increasing/decreasing values. Anomalies can be clearly seen in the following plot that represents a positive rate. -

In Annex in the section *icu_patients&positive_rate* are represented the plots for these two variables.

The *replace_missing_values* listing from Annex represents the code that helps to replace the missing values.

To detect anomalies the quantile function from pandas was used. for each country separately and for each variable, the time series was split into smaller chunks. For each chunk, with the help of quantile function was calculated the 99% quantile, and this will return the values that are like an outlier for the respective chunk. Next, this outlier should be replaced with the previous values which should be an outlier. After making these changes the positive_rate looks like in the following plot. This plot also included the replacement of missing values. After correcting the anomalies, the OY axis has a lower range thus in this graph we can see closer the behavior of positive_rate in different countries from Europe and the United States.

In the *correct_anomalies* listing from Annex is the code that is used to correct the anomalies. In this code, one can see that the time series was split into smaller chunks to be able to find the unexpectedly increasing/decreasing values.

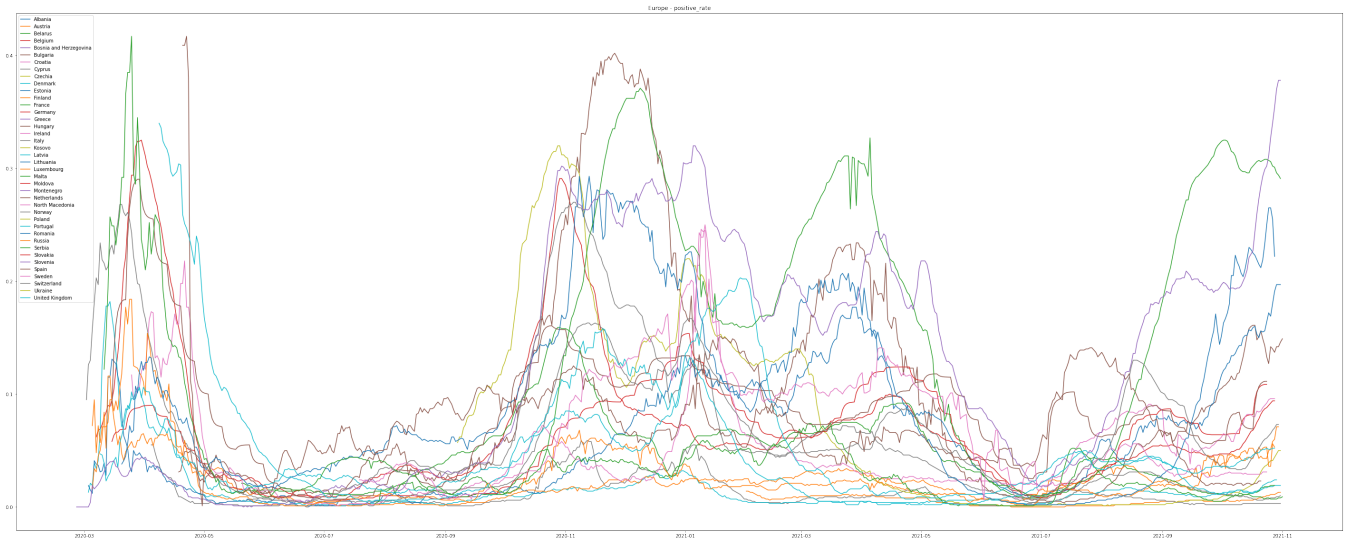


Figure 2. Processed positive_rate variable.

One of the variables that are not relevant to have data at the beginning of the pandemic is people_vaccinated.

In the *people_vaccinated* section from the annex there are two plots that show how the data for people/vaccinated looks like before processing this variable and after. It is observable that in the second plot with processed data, at the bottom where countries have a more similar number of people vaccinated, there are fewer lines. This is because the first plot includes all countries from Europe, including micro-countries.

For the variables as such, there is a *replace_first_missing_values* section in the Annex with the code that was used to replace the first missing values with 0 until the first non-missing values is met.

As we can see, for the variables that do not change their values within a country (constants per country), the missing values were replaced with the respective constant for the respective country. An example of such a variable is population_density. This variable doe not change much in 2-3 years as the pandemic takes, so it can be treated as a constant. These constant variables were also included because they might help the models to understand the pandemic behavior in relation to some characteristics

of a country.

Besides micro countries, there are also some countries from Europe that were excluded because they do not have data for the icu_patients variable. The following list shows the icu_patient data coverage percentage for some countries

- ...
- Albania, icu_patients = 0.0 %
- Austria, icu_patients = 98.7012987012987 %
- Belarus, icu_patients = 0.0 %
- Belgium, icu_patients = 92.4646781789639 %
- Croatia, icu_patients = 0.0 %
- Cyprus, icu_patients = 98.50993377483444 %
- Greece, icu_patients = 0.0 %
- Hungary, icu_patients = 0.0 %
- Ireland, icu_patients = 94.28104575163398 %
- Italy, icu_patients = 95.00780031201248 %
- Kosovo, icu_patients = 0.0 %
- Latvia, icu_patients = 0.0 %
- Lithuania, icu_patients = 0.0 %
- ...

The qualities of a stationary time series are independent of the time at which it is viewed. Time series containing trends or seasonality, on the other hand, are not stationary since the trend and

seasonality will alter the value of the time series at different times. A white noise series, on the other hand, is stationary – it should seem the same no matter when you look at it. A time series with cyclic behavior (but no trend or seasonality) is called stationary in some instances. It's because the cycles do not have a defined length, therefore it can't be predicted where the peaks and troughs will be until the series will be seen. A stationary time series will, in general, still have no predictable patterns over time. On-time plots, the series will appear to be generally horizontal (with some cyclic behavior), with constant variance. So, we need to make our data stationary as it is important because when modeling, there are assumptions that the summary statistics of observations are consistent and in terms of time series there is an expectation that the data is stationary. In order to make the data stationary must be removed the trend and seasonality. Using the Augmented Dickey-Fuller(ADF) test it can be found if the data is stationary or not. The statsmodels package provides a reliable implementation of the ADF test via the adfuller() function in statsmodels.tsa.stattools. It returns the following outputs: the p-value, the value of the test statistic, number of lags considered for the test, the critical value cutoffs. It was done by taking the logarithm of the series to stabilize the variance and also executing data differentiation. The test was run separately for each country and for each variable.

Before making the data stationary the tests should the following results

```
...


Germany

new_cases

ADF Statistic: -1.791618

p-value: 0.384631
```

```
icu_patients

ADF Statistic: -2.367005

p-value: 0.151266


...


...


Lithuania

new_cases

ADF Statistic: -2.383213

p-value: 0.146531




new_deaths

ADF Statistic: -1.536004

p-value: 0.515632


people_vaccinated

ADF Statistic: -0.865159

p-value: 0.799229
```

. . .

Most of the variable are not stationary because the p values calculated by augmented Dickey-Fuller tests is greater than 0.05. In order to make the data stationary the $np.log()$ was applied on each variable. After applying the log function there were still some variables that were not stationary. After this, the $diff()$ was applied one and after analyzing with Augmented Dickey-Fuller there were still some variables that were not stationary. After the second differentiation, the test returned a p ¡ 0.05 for all variables and countries. The following plots show the new_deaths variable non-stationary and after making it stationary.

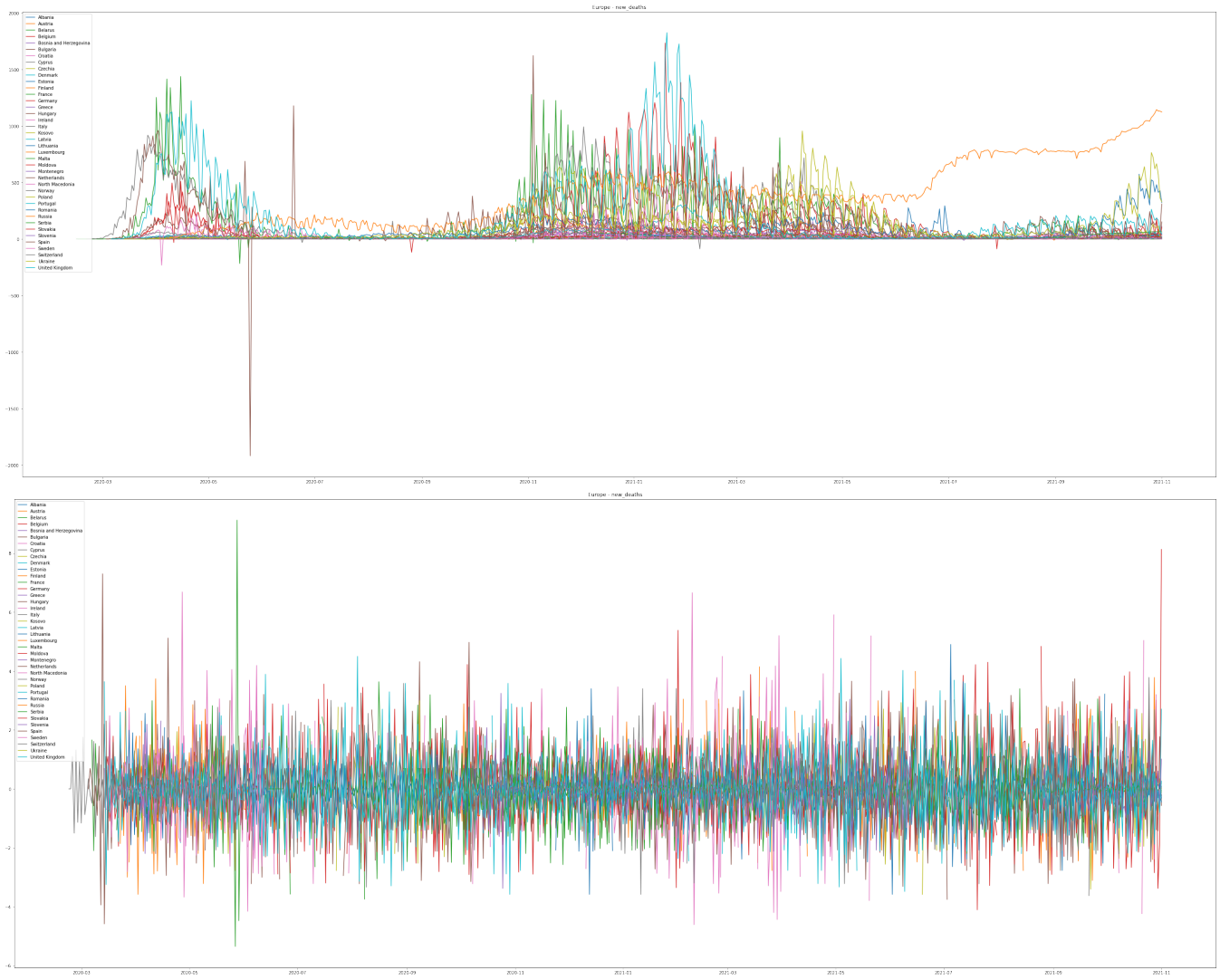


Figure 3. Non-stationary and stationary data.

3. Hypotheses and Selection of Predictors

When building a machine learning model, we need to understand which variables are relevant for the model purpose and how these predictors are going to affect the model. Redundant variables might reduce the model accuracy, furthermore, it increases the overall complexity of the model. Selection of predictors represents an indispensable action when building a machine learning model.

The main goal in this section is to find the best set of features that allows to build the model. There are more feature selection techniques in Machine Learning. In order to select the best predictors set for building a Machine Learning model that predicts the number of deaths caused by Covid-19, the correlation coefficient techniques was used.

Correlation is a measure of the linear relationship of 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that the good variables are highly correlated with the target. Furthermore, variables should be correlated with the target but should be uncorrelated among themselves.

If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information. We will use the Pearson Correlation here. We need to set an absolute value, say 0.5 as the threshold for selecting the variables. If we find that the predictor variables are correlated among themselves, we can drop the variable which has a lower correlation coefficient value with the target variable. We can also compute multiple correlation coefficients to check whether more than two variables are correlated to each other. This phenomenon is known as multicollinearity.(1)

In Figure 4 is displayed the heatmap with correlation between all initial variables.

We observe a lot of variables which duplicates themselves. Therefore we have the following set of

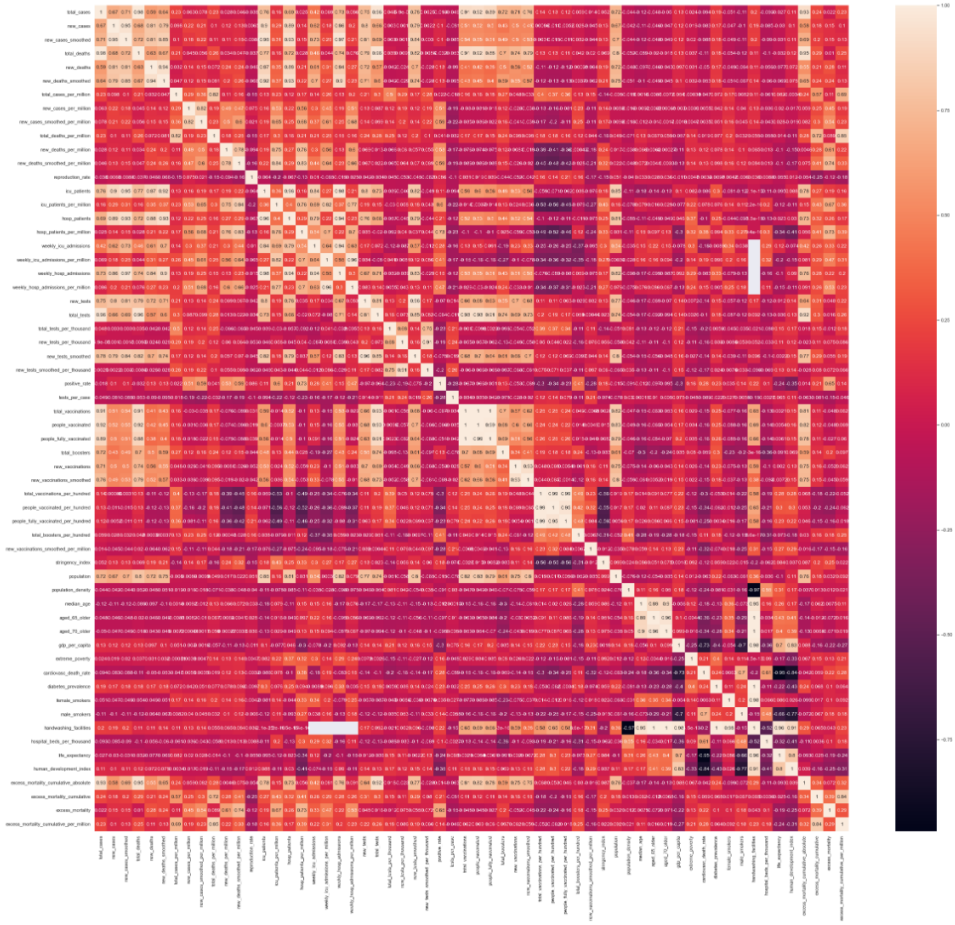


Figure 4. Heatmap

duplicates:

– The first one is related to the number of identified confirmed cases of Covid-19: total cases, new cases, new cases smoothed, total cases per million, new cases per million, new cases smoothed per million. Looking at the correlation for these variables we see that new cases and new cases smoothed have the same correlation with new deaths (what we want to predict), and it is bigger that the selected threshold, correlation is 0.81. The total cases has smaller correlation 0.59, total cases per million, new cases per million, new cases smoothed per million have even less than the threshold that was set. Therefore, from this set, further as predictor will be used new cases.

- Second set of duplicated data are the number of deaths caused by Covid-19. In this set we have: new deaths, new deaths smoothed, total deaths per million, new deaths per million,new deaths smoothed per million. What our model should predict is new deaths, so will keep this variable further.
- Another set of duplication might be considered the ones related to number of patients. Thus, here we have: ICU patients, ICU patients per million, hosp patients, hosp patients per million, weekly ICU admissions, weekly ICU admissions per million, weekly hosp admissions, weekly hosp admissions per million. Higher correlations from this set are for ICU patients and hospital patients. Because there are more countries which do not have data for hospital patients, will keep ICU patients as predictor, with 0.87 correlation.
- One more set of duplicated data implies the number of tests provided. In this set are included: total tests, new tests, total tests per thousand, new tests per thousand, new tests smoothed, new tests smoothed per thousand, positive rate, tests per case, tests units. From this list will keep new tests, with higher correlation then others, 0.72. Also, decided to give a try for positive rate.
- Vaccines a supposed to help humans in battle with the virus, therefore new vaccinations, and total booster are used as predictors.

The fields that are used as predictors are:

```
[
    'continent',
    'location',
    'date'
    'new_cases',
```

```
    'new_deaths',
    'icu_patients',
    'new_tests',
    'people_vaccinated',
    'new_vaccinations',
    'total_boosters',
]
```

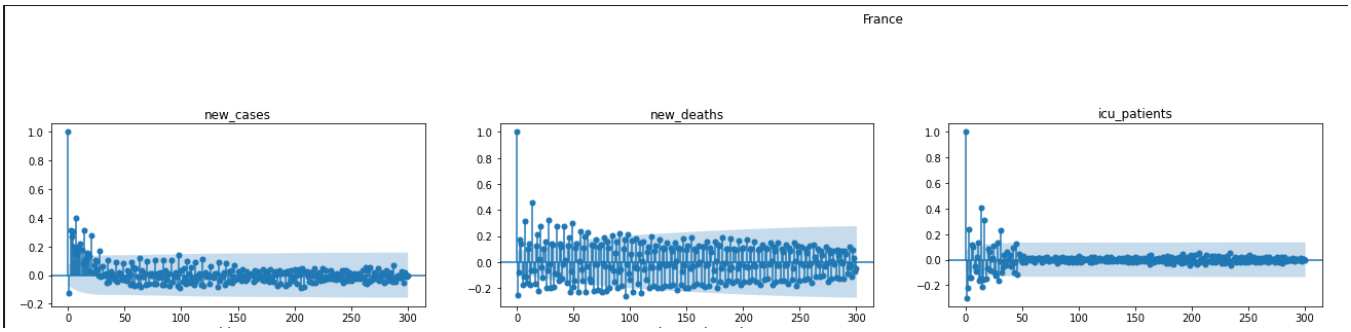
4. Modeling and Model Selection

It this chapter are shown the results given by different machine learning models of different categories such as classification models, regression models, and models for forecasting.

4.1. VARMAX

VARMAX was selected to be tested with this dataset because it is suitable for panel data as the dataset described in this paper is. The three most important parameters that VARMAX takes are: endog, exog and order. endog is a list that tell the models which are the endogenous variable s and exog is a list the tells which are the exogenous variables. *order* is a tuple of two elements. The first element is p and the second is q. The exogeneous variable is all the variables except the one that we want to predict (the endogenous variable). The endogenous variables will be the lags of the variable that we want to predict - new_deaths.

First, it is needed to know how many lags have to be used - p. Also, it should be investigated the number of lagged errors - q. There are at least two ways to find out what values should these parameters take. One was is to use ACF ((complete) auto-correlation function) and PACF ((partial) auto-correlation function). The following are the plots that show the ACF plot for three variables from France as an example.



A similar result is shown for the other variables. It is difficult and error-prone to choose the number of p and q from these plots because there are too many points that have to be counted. A better approach would be to use AutoARIMA in order to find out what should the p and q be equal to. AutoARIMA is a model that can by itself find out the p and q values. The AutoARIMA type of model was trained separately for each country and for each variable and the following results were obtained

```
...


Using auto_arima for Albania - new_cases

ARIMA(1,0,2)(0,0,2)[4] - p=1 & q=2


Using auto_arima for Albania - new_cases

ARIMA(3,0,0)(2,0,0)[4] - p=3 & q=0


...


Using auto_arima for Finland - new_cases

ARIMA(3,0,1)(0,0,0)[4] - p=3 & q=1


Using auto_arima for Finland - new_tests

ARIMA(1,0,0)(1,0,0)[4] - p=3 & q=0


...
```

The p and q values vary from country to country and from variable to variable. For the majority of the countries q=0 while p takes values from the range from 1 to 1. The VARMAX model will be tested with p=1, p=2, and then p=3. The best VARMAX model will be taken.

The following listing shows the line of code that sets up the model configuration

```
exogeneous_variables = [
    'population',
    'population_density',
    'diabetes_prevalence',
    'human_development_index',
    'new_tests',
    'stringency_index',
    'icu_patients',
    'cardiovasc_death_rate',
    'people_vaccinated',
    'new_vaccinations',
    'total_boosters',
    'positive_rate',
    'new_cases' ]


endogeneous_variables = [
    'new_deaths_0',
    'new_deaths_1',
```

```
    'new_deaths_2'
]

mod = sm.tsa.VARMAX(np.asarray(varmax_train_dataset[endogeneous_variables]),
            np.asarray(varmax_train_dataset[exogeneous_variables]), order=(3, 0))
```

The new_deaths_0, new_deaths_1 and new_deaths_2 are the lagged values for the dependent variable and they were calculated as follows

```
varmax_dataset['new_deaths_0'] = varmax_dataset['new_deaths'].shift(1)
varmax_dataset['new_deaths_1'] = varmax_dataset['new_deaths'].shift(2)
varmax_dataset['new_deaths_2'] = varmax_dataset['new_deaths'].shift(3)
```

To work with the model, the dataset was split into training dataset and test dataset

```
# split into train and test
training_date_limit = date(2021, 10, 1)

varmax_dataset.index = pd.to_datetime(varmax_dataset.date)
varmax_dataset.index.freq = varmax_dataset.index.inferred_freq

varmax_train_dataset=varmax_dataset[varmax_dataset['date'].dt.date<training_date_limit]
varmax_test_dataset=varmax_dataset[varmax_dataset['date'].dt.date>=training_date_limit]
```

After making the prediction with the test data for the United States the following results were obtained. The first column is the real data and the second list is the forecasted data.

```
...
-0.3437508314488875  0.28445733706109705
-0.8024151914418418 -0.14589590388008872
0.6440001128281265  -0.05591445134813724
1.8950502628532764   0.16089066913626682
...
```

The r2 score obtained is 0.04%.

4.2. SVR

The same dataset with stationary data was used to test the SVR model, as for VARMAX. As AutoARIMA suggested, we added 1 lag for the independent variables and 1 for the dependent variable.

The code from the *SVRlags* section from the Annex shows how the lagged variables were added.

The dummy variables help the model distinguish the pandemic behavior between the countries. Dummy variables are useful because they enable us to use a single regression equation to represent multiple groups. This means that we don't need to write out separate equation models for each subgroup. The dummy variables act like 'switches' that turn various parameters on and off in an equation.

In the dataset was added a new dummy variable for each country and this variable will take the value of 1 when the location variable will be equal to the respective country, otherwise 0.

The code from *SVRDummyVariables* section from Annex shows how the dummy variables were created.

The SVR model was tests with different values for the hyperparamaters and the default ones gave a good result. In the *SVRmodel* section from the Annex is presented the code that creates the SVR

model creation and the plots that compare the predicted values with the real ones. The models return 99% for most of the countries. This is most probably the case of over-fitting. To make this models better it should be analysed what the cause of over-fitting.

Even if we include only one lag for the dependent variable and for training we use for X only this lag, we already get a high accuracy. There are only just some countries that have an accuracy of approximately 70% in this case. We anyway tested the model with the lagged values for a few independent variables (icu_patients, new_tests , new_vaccinations) plus the lagged values for y and remove the original independent variables from X for training. In this case we got a high accuracy of 99% for most of the countries and for some countries an accuracy between 85% and 98%.

The result for SVR can also be seen on the following link
https://github.com/iondodon/covid19/blob/main/svr.ipynb

4.3. Multiple Linear Regression

The algorithm we choose here is known as Regression — this is a technique used to model the relationship between 2 variables and understanding how they contribute to a particular outcome together. Multiple Linear Regression (MLR) is similar to Simple Linear Regression but instead of using 1 variable to predict the outcome of another variable, MLR uses 2 or more variables to do so.

As for previous models, is used the same set of data. Before implementing and training multiple linear regression some operations were provided.

- Fixing date column datatype;
- Make the data set stationary;
- Keep only the countries that have enough variation;

- Add 3(according to AutoARIMA) lags for predictors and prediction variables, and remove them, leaving just the lags;
- Remove NaN values created by shifts;
- Add dummy variables;
- Split into train and test data.

Therefore, have the following predictors:

```
['new_cases0',
 'new_cases1',
 'new_cases2',
 'new_deaths0',
 'new_deaths1',
 'new_deaths2',
 'icu_patients0',
 'icu_patients1',
 'icu_patients2',
 'new_tests0',
 'new_tests1',
 'new_tests2',
 'people_vaccinated0',
 'people_vaccinated1',
 'people_vaccinated2',
 'new_vaccinations0',
```

```
'new_vaccinations1',
'new_vaccinations2',
'total_boosters0',
'total_boosters1',
'total_boosters2']

   X = lrm_test_dataset[numerical_variables_with_lags]
   X = X.drop(columns=['new_deaths'])
   y = lrm_test_dataset['new_deaths']
   .
   .
   .
   lrm_model = LinearRegression()
   lrm_model.fit(X, y)
   prediction = lrm_model.predict(X_test)
```

Multiple Linear Regression Score: 0.5270066972745802. R2 result tells how well the model fitted to the data. Let's look on a country example. In Figure 8 is displayed the graph for Multiple Linear Regression model for Austria, with R2 score equals to 0.3897397997845007. Looking through the results, could conclude that multiple linear regression model is not the best approach, model to predict the number of deaths caused by Covid-19, according to the used data frame.

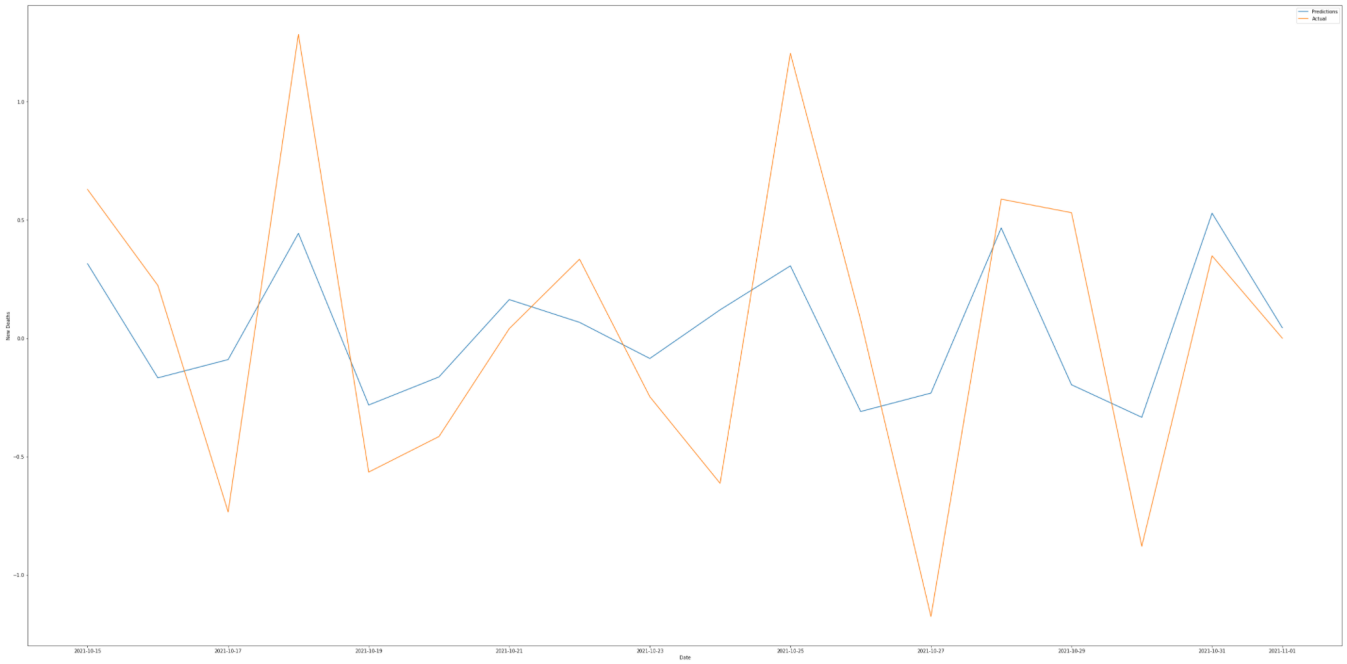


Figure 5. Multiple Linear Regression for Austria

4.4. Random Forest Regression

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. The ensemble learning method combines predictions from several machine learning algorithms to get a more accurate forecast than a single model. Let's go over the steps to acquire a better knowledge of the Random Forest algorithm:

- Pick k data points at random from the training set.
- Create a decision tree based on the k data points.
- Repeat steps 1 and 2 for the number N of trees you want to build.
- Make each of your N-tree trees forecast the value of y for the data point in question for a new data point, and then assign the new data point to the average of all of the predicted y values.

After importing the libraries, importing the data set, addressing null values, and dropping any neces-

sary columns, we determined the dependent(y) and non-dependent(X) variables. The non-dependable variables(predictors) for our model are: date, lags of predictors ('new_cases_0', 'stringency_index_0', 'icu_patients_0', 'people_vaccinated_0', 'positive_rate_0', 'new_vaccinations_0','new_tests_0', 'total_boosters_0'), and dummies for all the used countries('Austria', 'Belgium', 'Bulgaria', 'Cyprus', Czech', 'Denmark', 'Estonia', 'Finland', 'France', 'Germany', 'Ireland', 'Italy', 'Luxembourg', 'Malta', 'Netherlands', 'Portugal', 'Romania', 'Serbia', 'Slovenia', 'Spain', 'Sweden', 'Switzerland', 'United Kingdom', 'United States') and two lags for the predicted variable(new_deaths_lag1, new_deaths_lag2). The dependent variable is new_deaths.

Below is shown the results of prediction for the Bulgaria country for a period of one month (October 2021)(See Figure 3). The accuracy of the predictions is 59.68%.

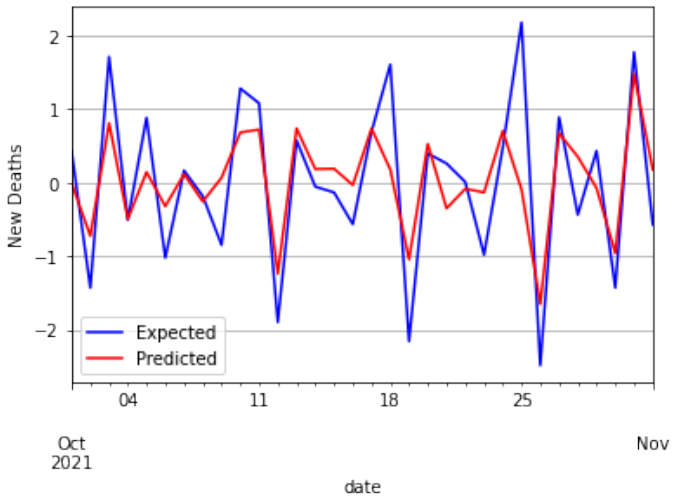


Figure 6. Random Forest Regression for Bulgaria

Also we obtained for Estonia - 51.55%, Denmark - 24.88%, Czechia - 33.62%, Cyprus - 62.28%, and bad results for Belgium, Finland that have accuracies below 0. The results for other countries you can check here: `https://github.com/iondodon/covid19/blob/main/RandomForestReg_%26_XGBoost.ipynb`.

4.5. Extreme Gradient Boosting

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.(2) The predictors and predicted variables we used for XGBoost were used the same as for Random Forest Regression. For this model we have used the default parameters. Below we have the results for Bulgaria for this model that has 58.20%(See Figure 4)

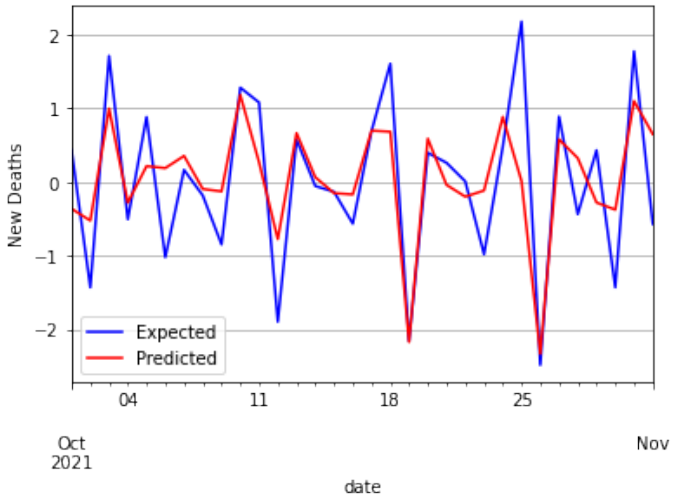


Figure 7. XGBoost for Bulgaria

Also, for other countries we have these results: Cyprus- 62.13%, Czechia - 25.97%, Denmark - 8.48%, Germany - 49.51%, United States - 57.33%. Other results for other countries you can check here: `https://github.com/iondodon/covid19/blob/main/RandomForestReg_%26_XGBoost.ipynb`.

4.6. LightGBM Regressor

Light GBM is a gradient boosting framework that uses tree based learning algorithm. For training the model we used boosting_type = 'goss', max_depth = 4 parameters. The predictors and predicted variables are used as same as for Random Forest Regression. Below can be observed the results for United States that has 62.55% accuracy score for this model(See Figure 5).

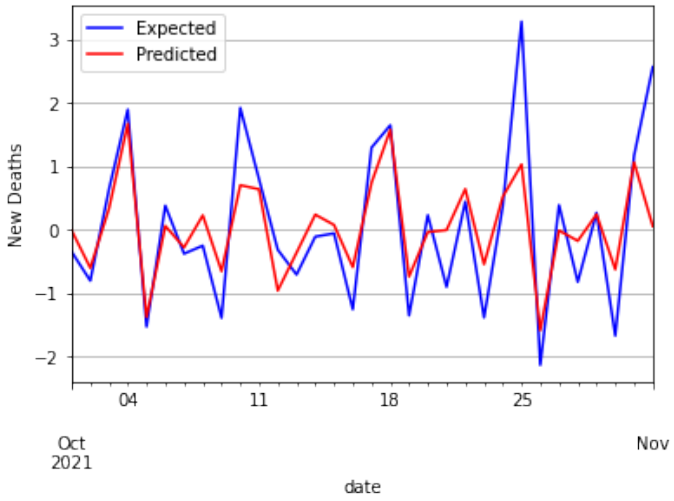


Figure 8. LGBM Regressor for United States

For other coutries we have the following results: United Kingdom - 26.25%, Switzerland - 24.85%, Sweden - 29.94%, Slovenia - 15.56%. The results for other countries are shown here: `https://github.com/iondodon/covid19/blob/main/RandomForestReg_%26_XGBoost.ipynb`.

5. Cross-Validation and Model Tuning

Some countries were removed from the dataset because their data doesn't fluctuate much. The data from these countries is stationary.

```
    countries_to_exclude = [
    ...
    'Slovenia',
    'Spain',
    'Sweden',
    'Switzerland',
    'Latvia',
    'Kosovo',
    'Ireland',
    'Hungary',
    ...
]
```

In order to see if the model was tested with a lucky dataset, we took one model that gave us a high accuracy and tested it on different training data. Since we have time-series data, we split the dataset in the following way iteratively: first was taken the data from the first 120 days. From these 120 we took the last 20% of days for testing and the remaining for training. At the second iteration, we took the first 240 days from the original dataset. From these 240 days, we took the last 20% for testing the first 80% for training. And so on.

We have noticed that for most countries, despite changing the datasets for training, the model

returns a high accuracy percentage. For some countries, the model can't predict at all and return a 0% accuracy. This is most probably because for those countries there is no data at the beginning of the pandemic.

In the *CrossValidation* section in Annex can be found the code that does the cross-validation.

To find the best parameters for the SVR model we used Grid Search. The Grid Search technique gave us the following results: 'C': 0.5, 'epsilon': 1e-05, 'gamma': 0.001, 'kernel': 'linear'. Grid Search helped us to find the best parameter to get a higher score. The parameters specified in the Grid Search are the following

```
parameters = {
    'C': [0.01, 0.1, 0.2, 0.5, 1, 10, 50, 100, 1000],
    'gamma': [0.001, 0.0001],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'epsilon': [0.0001, 0.001, 0.0001, 0.00001]
}
```

But the score that we obtained was already too high. In this case, Grid Search can't help us solve the problem of over-fitting. By trying manually some other hyperparameters such as C=2, epsilon=0.01, kernel='rbf' didn't return a better result.

Even if we use as a predictor only the first lag of new_deaths we already get scores like 99%.

On the following link can be found the result for Cross-Validation and Grid Search

https://github.com/iondodon/covid19/blob/main/svr.ipynb

6. Conclusions

Before modeling it is needed to analyze the dataset very. A good way to analyze the dataset is by plotting. After displaying the plots we noticed that there is a lot of missing data for some countries, for most of the countries from different continents except Europe there was no data available for icu_patient, and for the other variables, there was not enough data. To be sure that our model will be well trained we decided to exclude those continents and leave only Europe. We also added the United States because there was enough data for this country and in this section, the dataset was quite well populated with data.

To make sure the model will not be biased we excluded countries that have a number of population less than 500000 people because these countries have a slightly different behavior different from the majority of the countries. The dataset included negative values which are not normal for the variables that we have. We replaced the negative variables with the previous non-negative value. There were also anomalies that we also replaced with the previous values. To make the time series models work well the data was made stationary.

There are several models that were tested. The first one was VARMAX. With VARMAX we got an R2 score of 0.04%. This model was one that forecasted the worst. Another model that was tested was SVR. With SVR we had to include dummy variables for each country and create lagged variables for independent and dependent variables. With SVR we got a better score than with VARMAX, a score of 99%for most of the countries. In this case most probably it is over-fitting. In the future it should be investigated what's the cause of over-fitting. For the Random Forest Regression, XGBoost and LGBM Regressor we obtained best accuracies between 60%-70%. For these models we plan to make the tuning of the parameters and to test them for over fitting using the cross-validation.

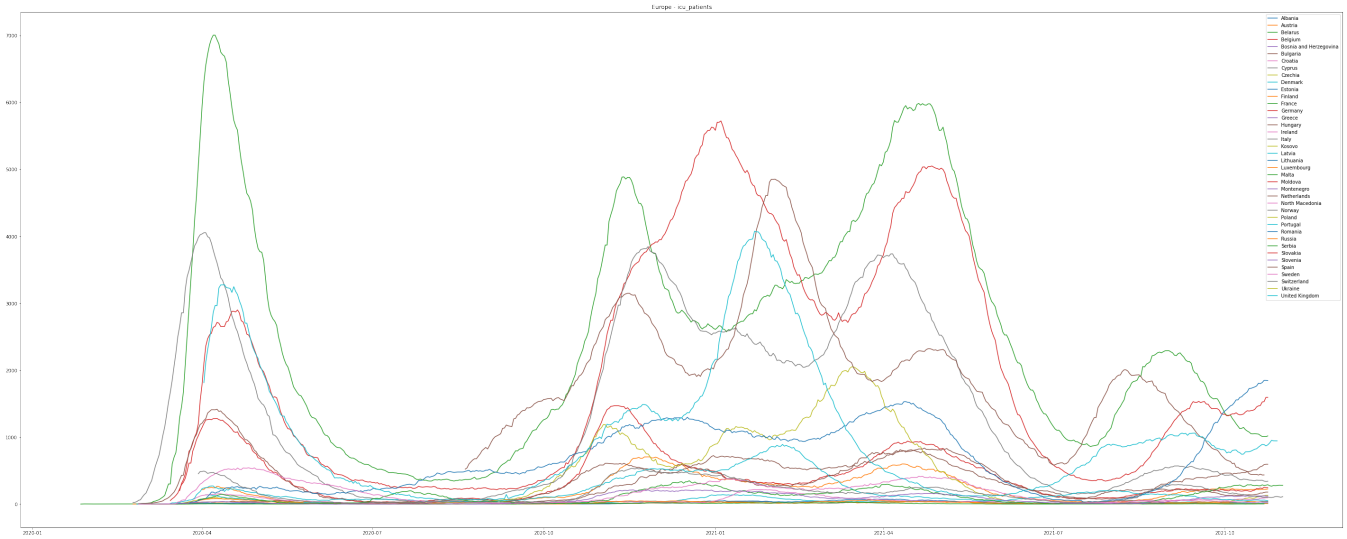
References

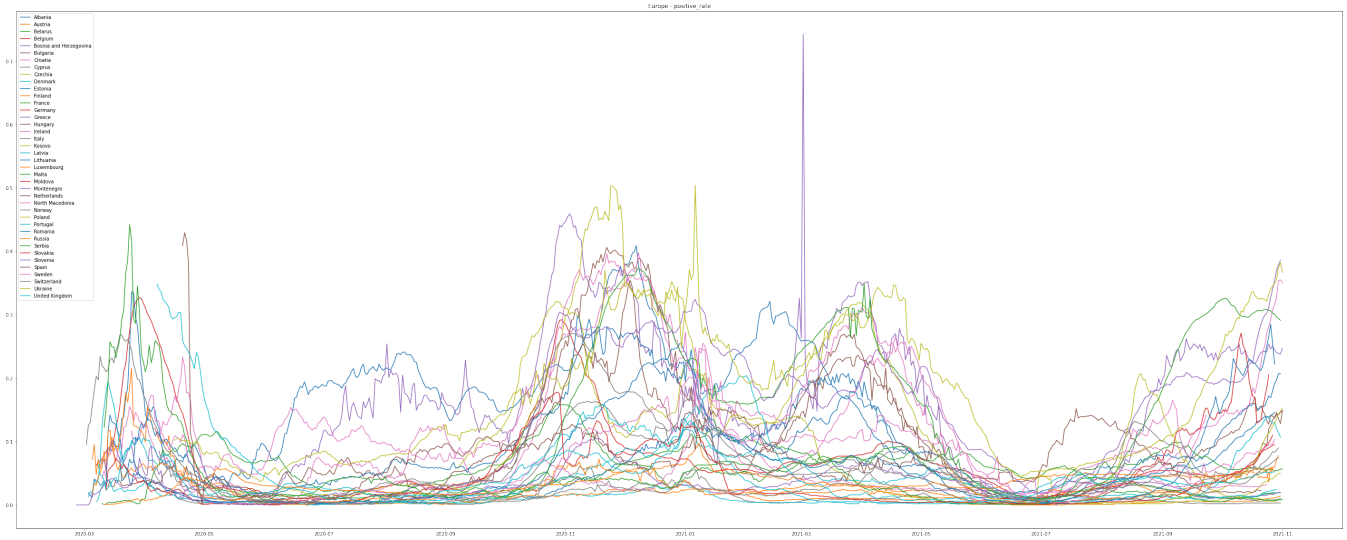
[1] Aman Gupta (2020) *Feature Selection Techniques in Machine Learning*, https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/

[2] Vishal Morde (2019) *XGBoost Algorithm: Long May She Reign!*, https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Annex

icu_patients & positive_rate





replace_missing_values

```
def replace_missing_values(dataset):
    df = dataset.copy()
    new_dataset = pd.DataFrame()
    for country in df['location'].unique():
        country_dataset = df[df['location'] == country]
        for variable in variables:
            if variable in ['location', 'continent', 'date']:
                continue
            found_non_missing = False
            for index, row in country_dataset.iterrows():
                if pd.isna(row[variable]):
                    if found_non_missing and (index - 1) in country_dataset.index
                            and (index + 1) in country_dataset.index:
                        next = next_non_missing_value(country_dataset, index, variable)
                        last = last_non_missing_value(country_dataset, index, variable)
                        if next[1] == -1 or last[1] == -1:
                            continue
                        new_val = (next[1] - last[1]) / 2
                        country_dataset.loc[index, variable] = last[1] + new_val
                else:
                    found_non_missing = True
```

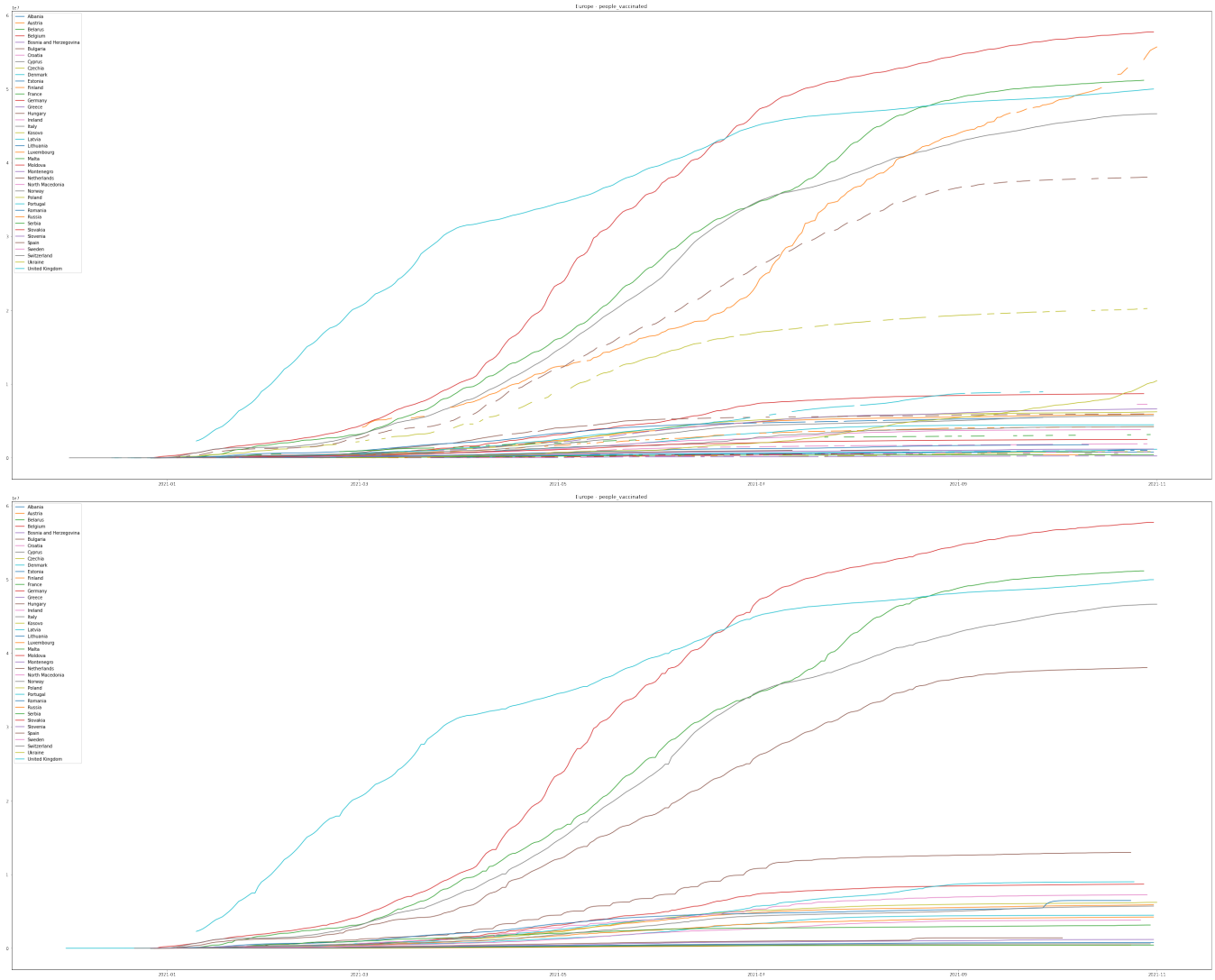
```
        new_dataset = pd.concat([new_dataset, country_dataset])
    return new_dataset
```

correct_anomalies

```
def correct_anomalies(dataset):
    new_dataset = pd.DataFrame()
    for country in dataset['location'].unique():
        new_country_data = pd.DataFrame()
        country_data = dataset[dataset['location'] == country]
        dataset_chunks = split_dataframe(country_data, 25)

        for chunk in dataset_chunks:
            for variable in variables:
                if variable in ['location', 'continent', 'date']:
                    continue
                anomaly_indexes=chunk[chunk[variable]>chunk[variable].quantile(0.99)].index
                for index in anomaly_indexes:
                    if index - 1 in chunk.index:
                        chunk.at[index, variable] = chunk.at[index - 1, variable]
            new_country_data = new_country_data.append(chunk)

        new_dataset = new_dataset.append(new_country_data)
```

```
    return new_dataset
```

people_vaccinated



replace_first_missing_values

```
def replace_backwards(new_dataset, index, variable, value_to_replace_with):
    while index in new_dataset.index:
```

```
        new_dataset.at[index, variable] = value_to_replace_with
        index -= 1

# replace first missing values with first non missing value
def replace_first_missing_values(dataset):
    new_dataset = dataset.copy()

    for variable in ['population', 'population_density', 'cardiovasc_death_rate',
                     'diabetes_prevalence', 'human_development_index']:
        for index, row in dataset.iterrows():
            if not pd.isna(row[variable]):
                replace_backwards(new_dataset, index, variable, row[variable])
                break

    for variable in ['new_vaccinations', 'people_vaccinated', 'total_boosters',
                     'icu_patients', 'new_tests', 'new_cases', 'new_deaths',
                     'positive_rate', 'stringency_index']:
        for index, row in dataset.iterrows():
            if not pd.isna(row[variable]):
                replace_backwards(new_dataset, index, variable, 0)
                break

    return new_dataset
```

SVR lags

```
nb_lags_dependent = 2 # calculated with AutoARIMA

nb_lags_independent = 1 # calculated with AutoARIMA


# for each variable

for variable in numerical_variables:

    if variable == 'new_deaths':

        continue

    for nb_lag in range(0, nb_lags_independent):

        svr_dataset[variable + '_' + str(nb_lag)] = svr_dataset[variable].shift(nb_lag)


for nb_lag in range(nb_lags_dependent):

    svr_dataset['new_deaths_' + str(nb_lag)] = svr_dataset['new_deaths'].shift(nb_lag)
```

SVR dummy variables

```
# for each country

for country in svr_dataset['location'].unique():

    # set empty column with 0 values for svr_dataset[country]

    svr_dataset[country] = 0

    svr_dataset[country].loc[svr_dataset['location'] == country] = 1
```

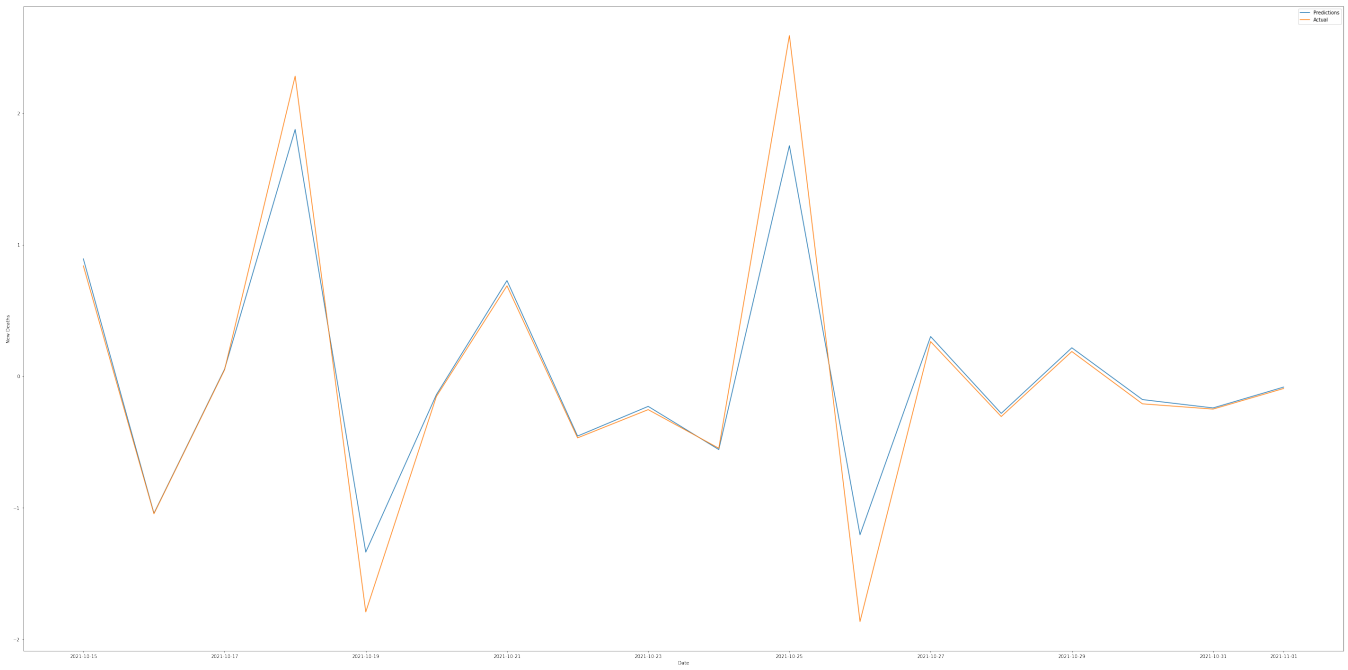
SVR model

```
def svr(svr_train_dataset):
```

```
    y = svr_train_dataset['new_deaths']

    X = svr_train_dataset[numerical_variables_with_lags]

    for numercal_variable in numerical_variables:
        X = X.drop(columns=numercal_variable)

    svr_model = SVR()
    svr_model.fit(X, y)

    return svr_model

def svr_predict(model, country, svr_test_dataset):
    y_test = svr_test_dataset['new_deaths'].loc[svr_test_dataset[country] == 1]

    # show score
    X_test = svr_test_dataset[numerical_variables_with_lags].loc[svr_test_dataset[country] ==

    for numerical_variable in numerical_variables:
        X_test = X_test.drop(columns=numerical_variable)

    predictions = model.predict(X_test)
```

```
print("C: {} - SVR Score: {}".format(model.C, model.score(X_test, y_test)))


# plot predictions vs actual

plt.figure(figsize=(50, 25))

plt.plot(svr_test_dataset['date'].loc[svr_test_dataset[country] == 1],

    predictions, label='Predictions')

plt.plot(svr_test_dataset['date'].loc[svr_test_dataset[country] == 1],

    y_test, label='Actual')

plt.xlabel("Date")

plt.ylabel("New Deaths")

plt.legend()

plt.show()


return predictions
```

SVR Model Result



Cross Validation

```
    index = min_date + pd.Timedelta(days=120)


while index <= max_date:

    print("==============================================

             ============ Data taken from {} to {}".format(min_date, index))


    # get data where svr_dataset.index < index

    df = svr_dataset[svr_dataset.index < index]

    df.index = pd.to_datetime(df.date)

    print(df.shape)
```

```
# 80% of the number of days from min_date to index
training_date_limit = min_date + pd.Timedelta(days=int(0.8 * (index - min_date).days))

svr_train_dataset = df[df.index < training_date_limit]
svr_test_dataset = df[df.index >= training_date_limit]

model = svr(svr_train_dataset)
# for each country
for country in countries:
    plot = False
    if index + pd.Timedelta(days=120) > max_date:
        plot = True
    svr_predict(model, country, svr_test_dataset, plot)

index = index + pd.Timedelta(days=120)
```