# Report on learning practice # 4

# Stationarity of the processes

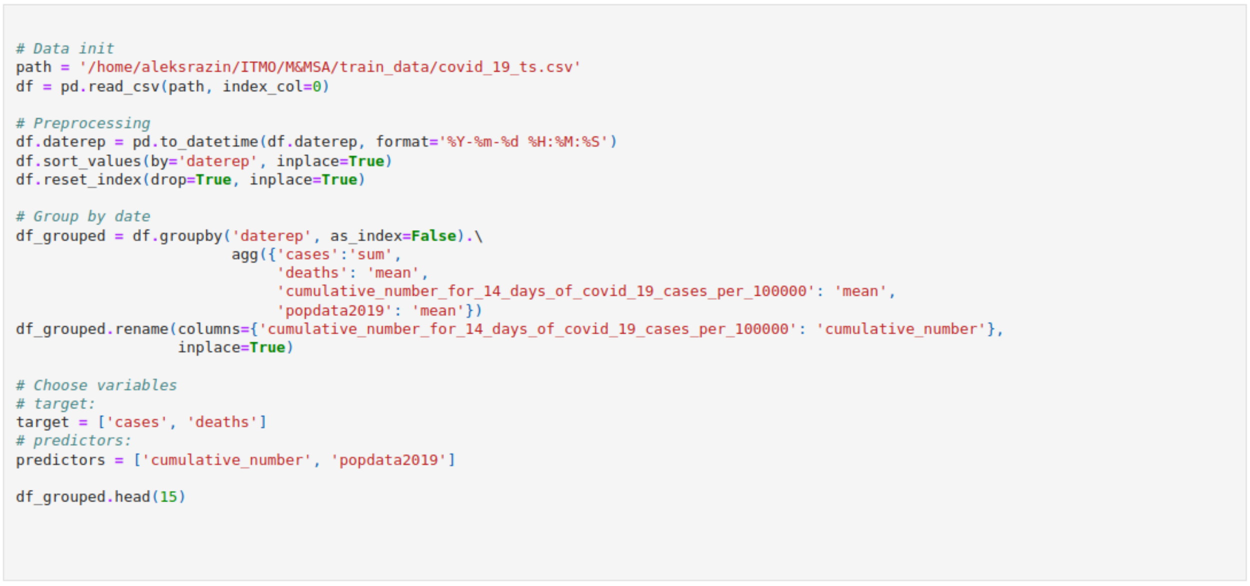
Performed by
*Alexander Razin*
*Mikhail Lovtskiy*
*Mark Evgrafov*
*Julia Pimkina*
*Ac. group J4132c*

*Saint-Petersburg*
*2022*

## Table of contents:

### *1. Substantiation of chosen sampling.*

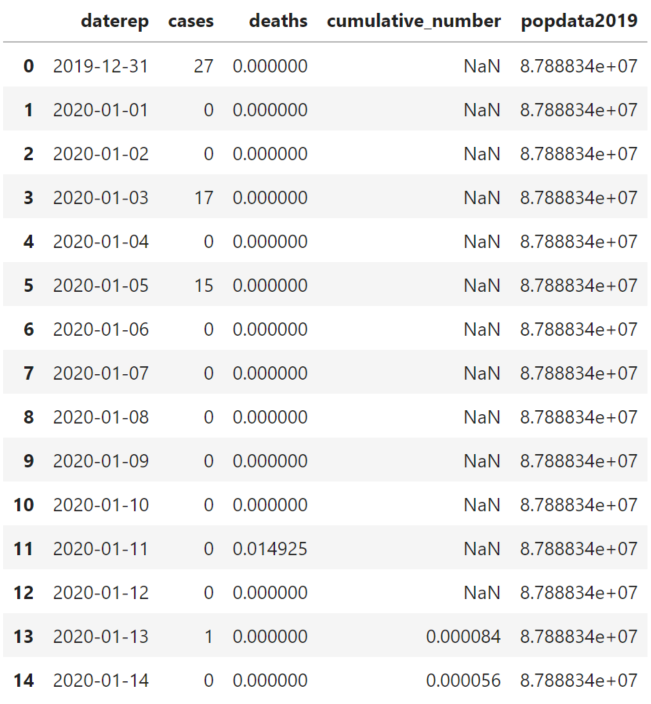
This Lab we used to have timestamps in our dataset. Dataset we used in the previous works doesn't have them. But Covid19 challenge has another data with Time Series (https://www.kaggle.com/datasets/roche-data-science-coalition/uncover). The original dataset contains 2.5 Mb of covid19 information about deaths and cases for 2020.



```
# Data init
path = '/home/aleksrazin/ITMO/M&MSA/train_data/covid_19_ts.csv'
df = pd.read_csv(path, index_col=0)

# Preprocessing
df.daterep = pd.to_datetime(df.daterep, format='%Y-%m-%d %H:%M:%S')
df.sort_values(by='daterep', inplace=True)
df.reset_index(drop=True, inplace=True)

# Group by date
df_grouped = df.groupby('daterep', as_index=False).\
                        agg({'cases':'sum',
                             'deaths': 'mean',
                             'cumulative_number_for_14_days_of_covid_19_cases_per_100000': 'mean',
                             'popdata2019': 'mean'})
df_grouped.rename(columns={'cumulative_number_for_14_days_of_covid_19_cases_per_100000': 'cumulative_number'},
                  inplace=True)

# Choose variables
# target:
target = ['cases', 'deaths']
# predictors:
predictors = ['cumulative_number', 'popdata2019']

df_grouped.head(15)
```

*Fig.1. Timestamps preprocessing.*

As You can see in the code timestamps are represented as string, so our script also converts them into default Python 3 timestamps.



| | daterep | cases | deaths | cumulative_number | popdata2019 |
|---|---|---|---|---|---|
| **0** | 2019-12-31 | 27 | 0.000000 | NaN | 8.788834e+07 |
| **1** | 2020-01-01 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **2** | 2020-01-02 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **3** | 2020-01-03 | 17 | 0.000000 | NaN | 8.788834e+07 |
| **4** | 2020-01-04 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **5** | 2020-01-05 | 15 | 0.000000 | NaN | 8.788834e+07 |
| **6** | 2020-01-06 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **7** | 2020-01-07 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **8** | 2020-01-08 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **9** | 2020-01-09 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **10** | 2020-01-10 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **11** | 2020-01-11 | 0 | 0.014925 | NaN | 8.788834e+07 |
| **12** | 2020-01-12 | 0 | 0.000000 | NaN | 8.788834e+07 |
| **13** | 2020-01-13 | 1 | 0.000000 | 0.000084 | 8.788834e+07 |
| **14** | 2020-01-14 | 0 | 0.000000 | 0.000056 | 8.788834e+07 |

*Fig.2. Working dataset for lab 4 visualization.*

For this task were chosen these target variables: *cases, deaths* and following predictors: *cumulative_number, popdata2019. daterep* variable contains timestamps for our data. Time series for all chosen variables aggregated by each day with mean and sum values are shown on the figure below.

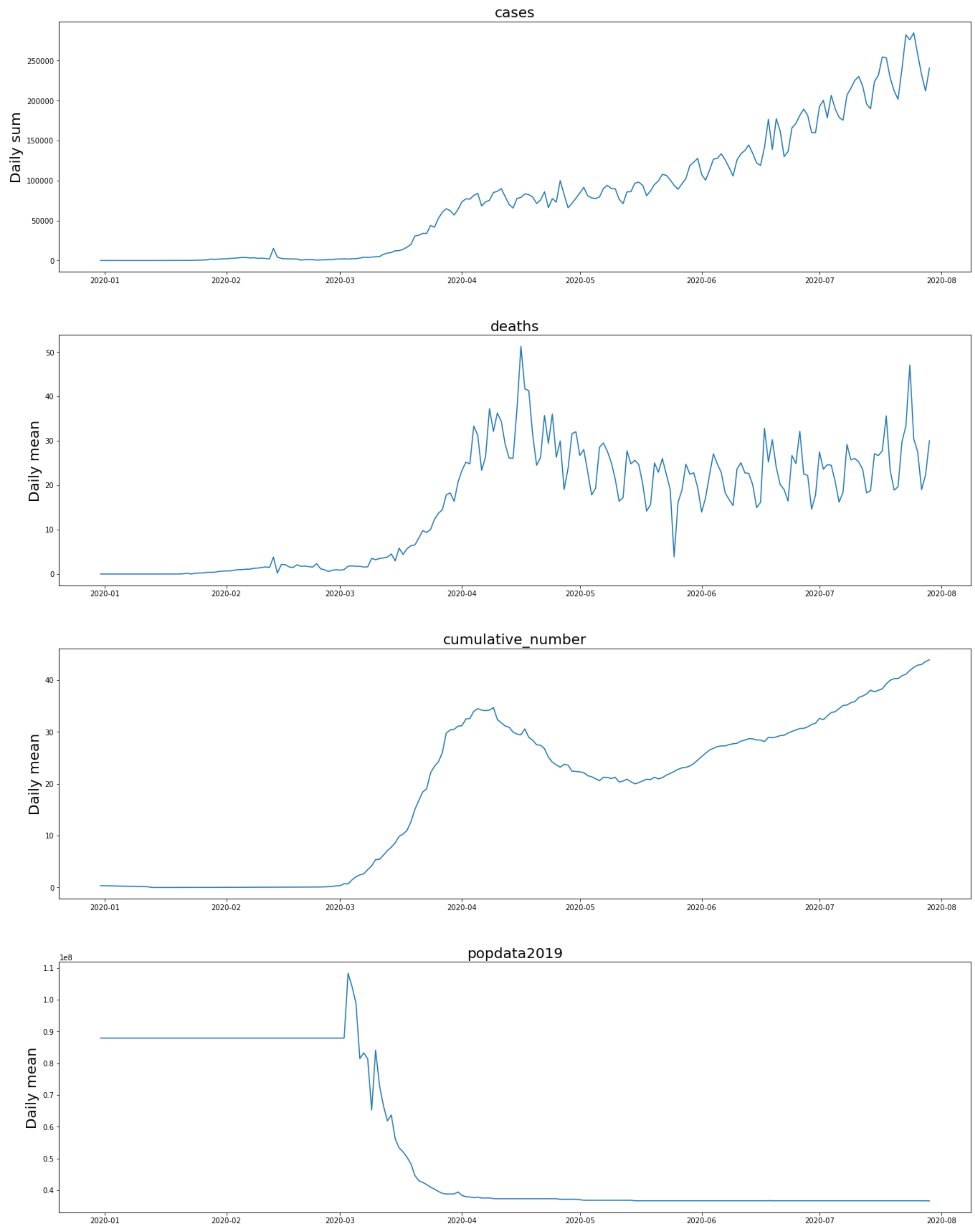


*Fig.3. Initial time series for all variables.*

## *2. Stationary analysis.*

As one can easily see, gained time series represent non-stationary processes. To make them more stationary, the trends were removed. The gained time series are presented on figure 4 before and after removing trend. All this time series passed the Augmented Dickey-Fuller Test and can be considered stationary. The Results of Dickey-Fuller Test are presented for each variable below.

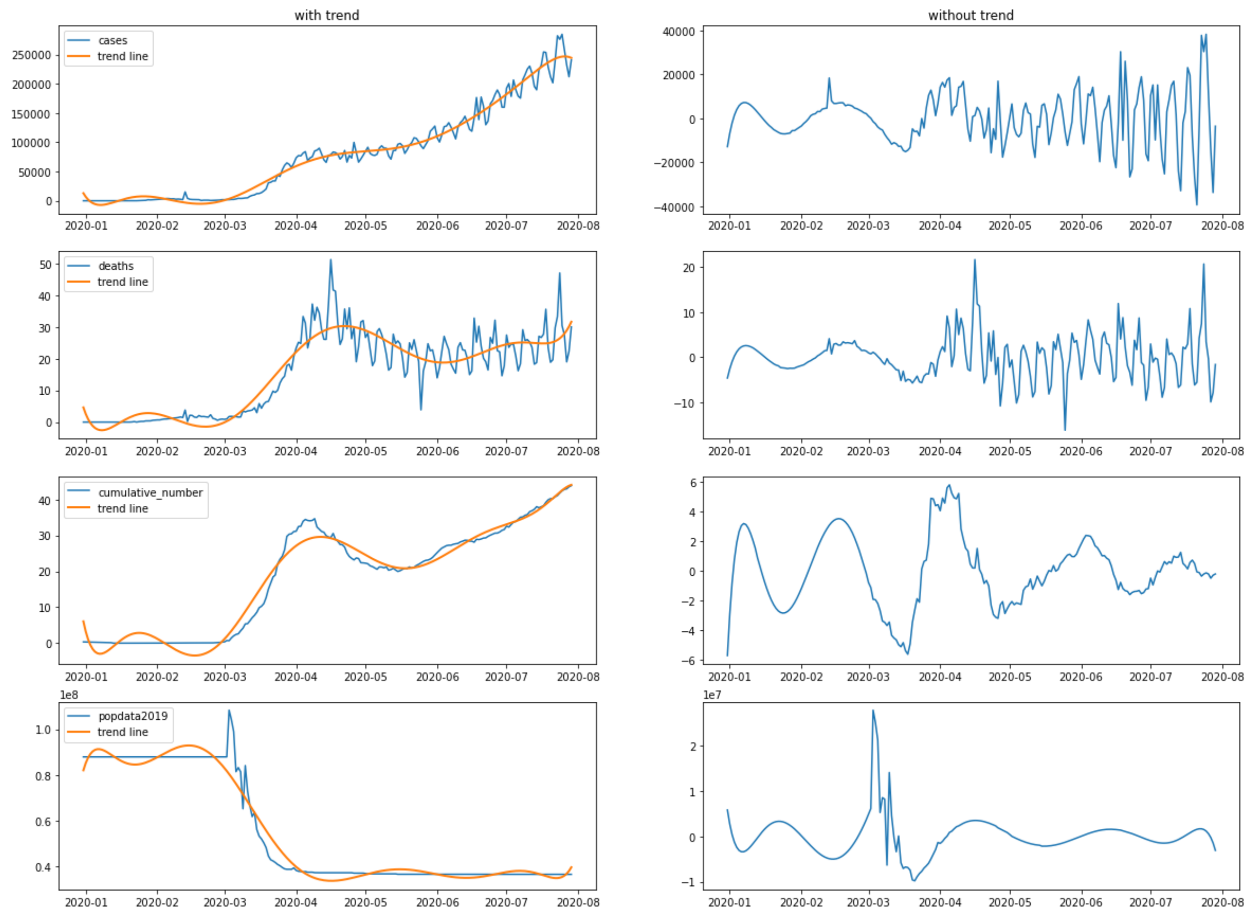
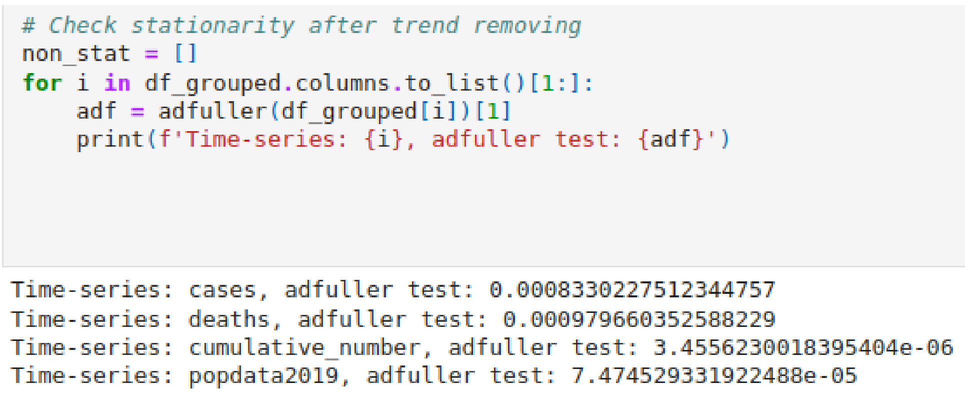


*Fig.4. Gained stationary process.*



```
# Check stationarity after trend removing
non_stat = []
for i in df_grouped.columns.to_list()[1:]:
    adf = adfuller(df_grouped[i])[1]
    print(f'Time-series: {i}, adfuller test: {adf}')
```

```
Time-series: cases, adfuller test: 0.0008330227512344757
Time-series: deaths, adfuller test: 0.000979660352588229
Time-series: cumulative_number, adfuller test: 3.4556230018395404e-06
Time-series: popdata2019, adfuller test: 7.474529331922488e-05
```

*Fig.5. Check stationary after removing trends .*

## *3. Covariance or correlation function analysis.*

Figure 5 represents Auto-covariance function for target variables. On figure 6 Auto-covariance function gained using a window of 30 items is shown. These functions look like their processes are stationary.

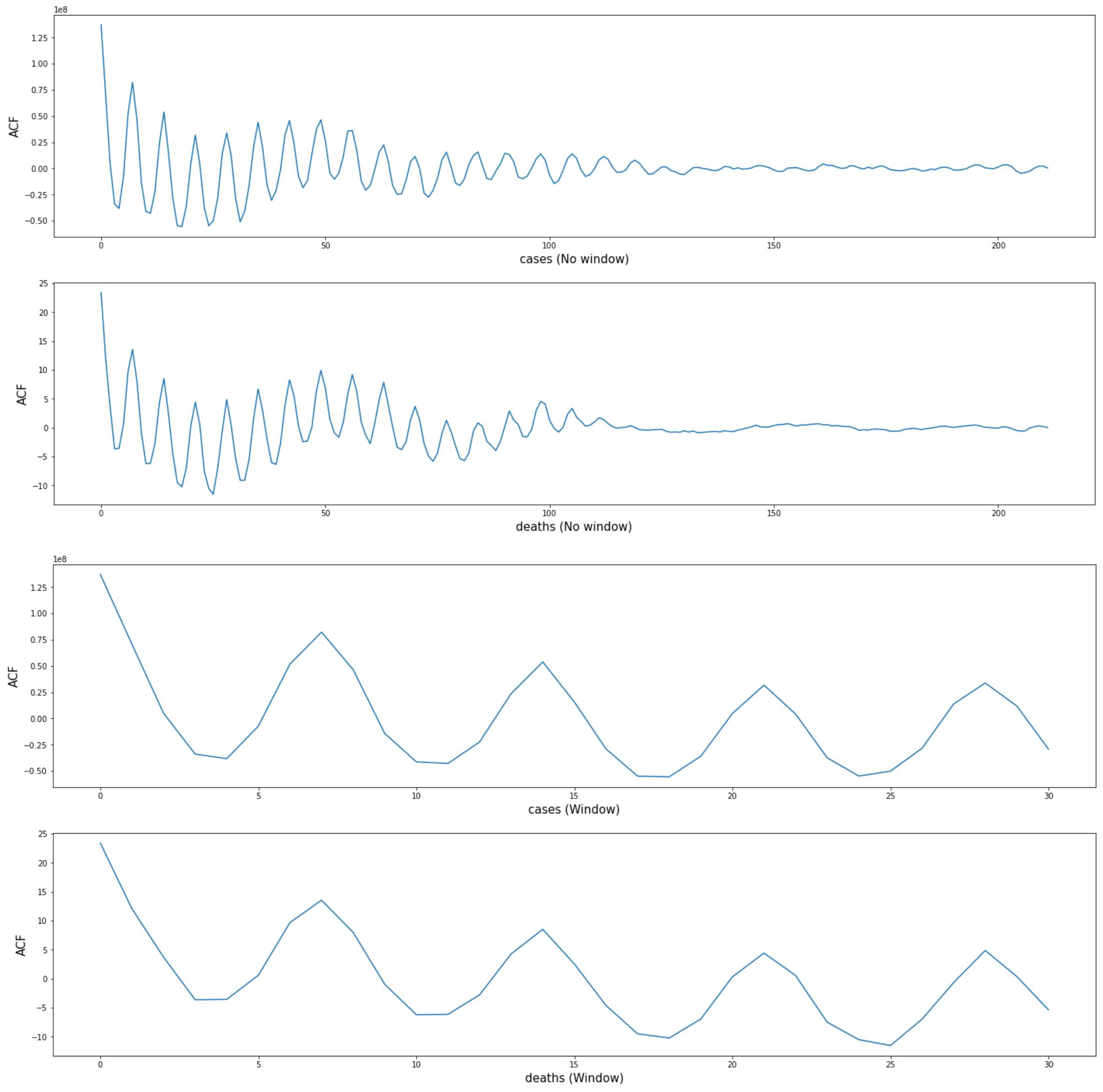


*Figure 6. Autocovariance with and without window.*

Figure 7 represents cross-covariance (mutual correlation) function between each target and predictor variable.

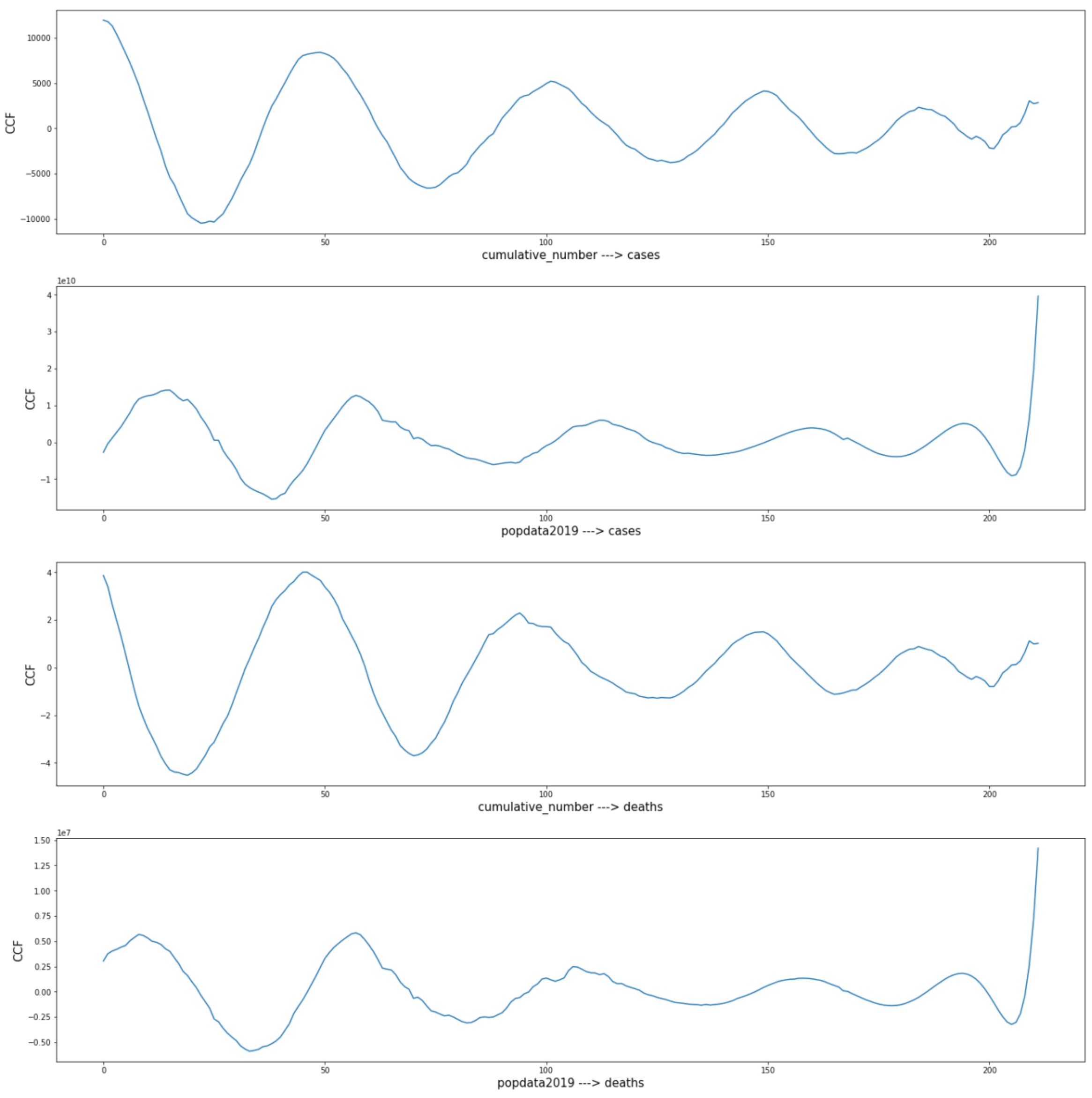


*Fig.7. Cross-covariance functions.*

Cross correlation matrix for variables is presented on figure 8. As one can see, target and predictor variables are highly correlated, especially *cases* are correlated with *comulative_number*.

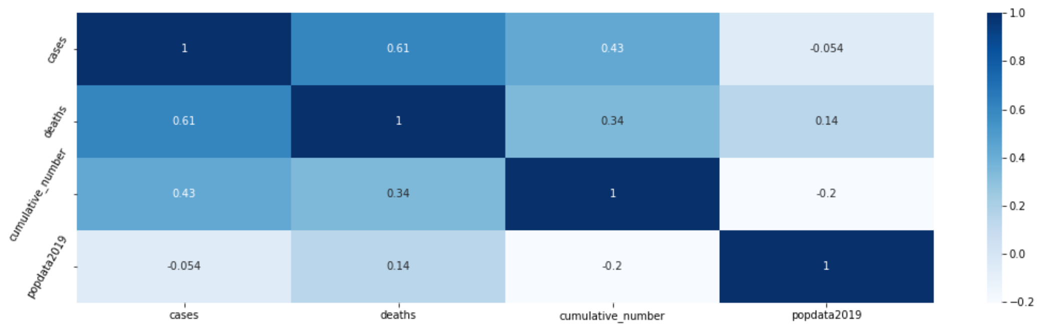


*Fig.8. Cross correlation matrix.*

## *4. Noise filtration.*

For filtering high frequencies Moving average filter (fig.9), Gaussian filter (fig.10) from FEDOT framework were used, and also Kalman filter was implemented (fig.11).

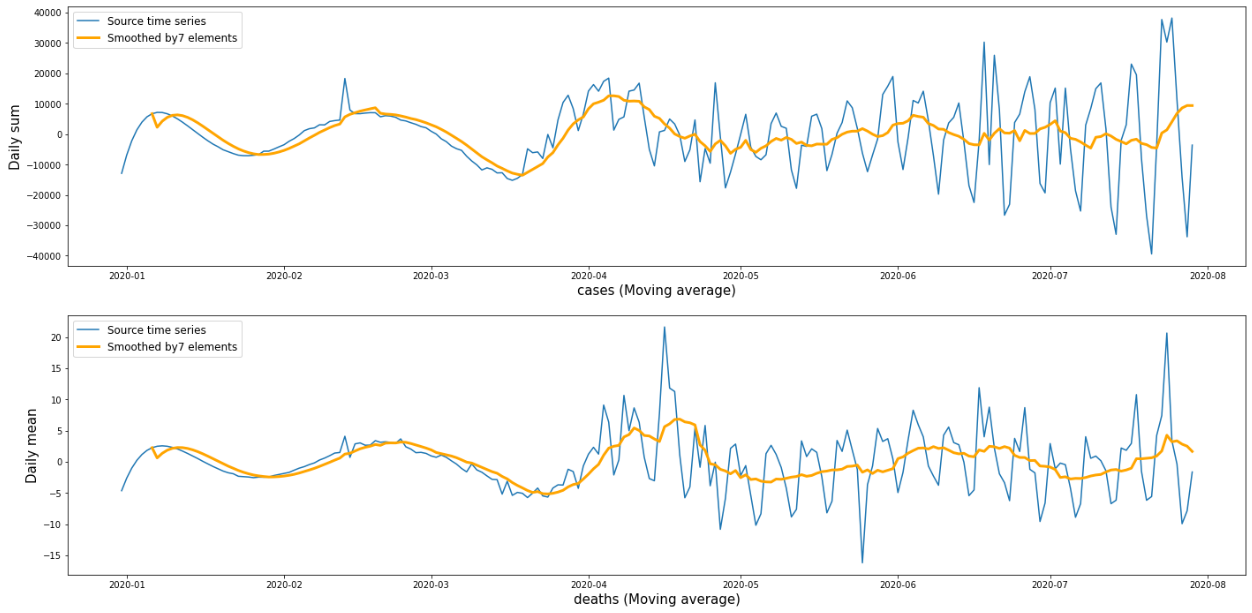


*Fig.9. Using Moving average filter.*

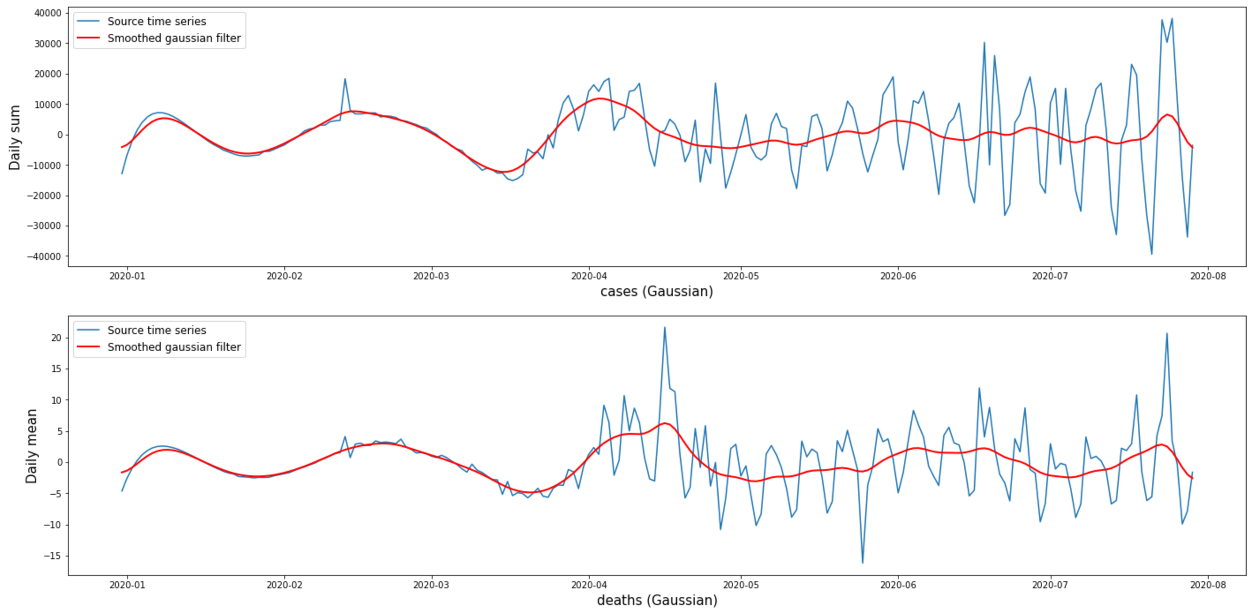


*Fig.10. Using a Gaussian filter.*

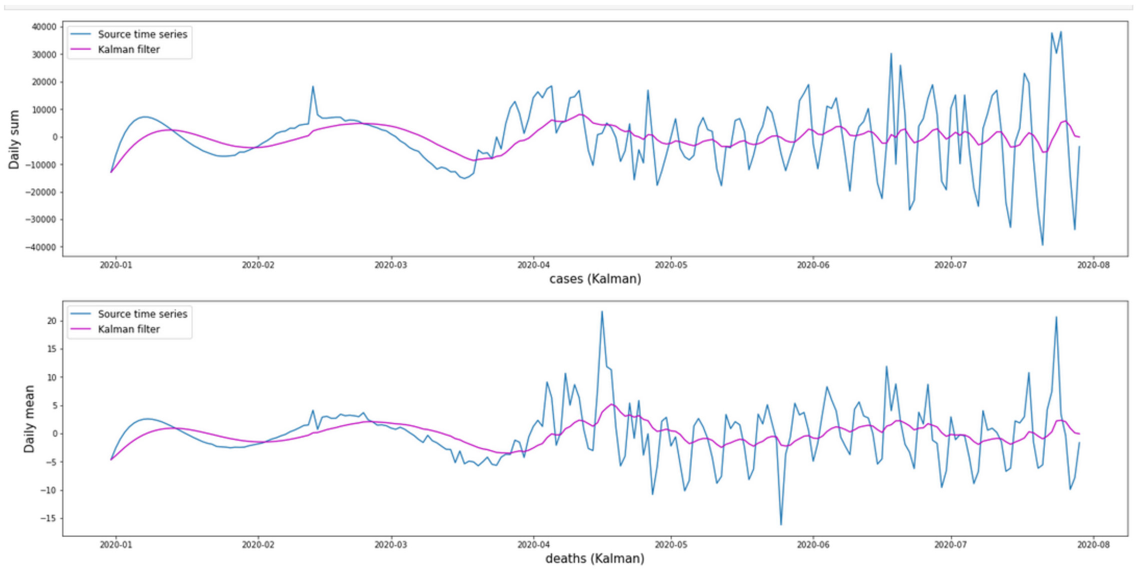


*Fig.11. Using a Kalman filter.*

## 5. *Estimation of spectral density function.*

Spectral density was estimated by Welch's method from the scipy.signal module. The results for both non-filtered and filtered time series are presented on figure 12. There are two plots for each target variable. Three different filters are shown using different colors: orange for Moving average, red for Gaussian filter and pink for Kalman filter.

As one can see all filters removed high frequencies from the spectral density function.

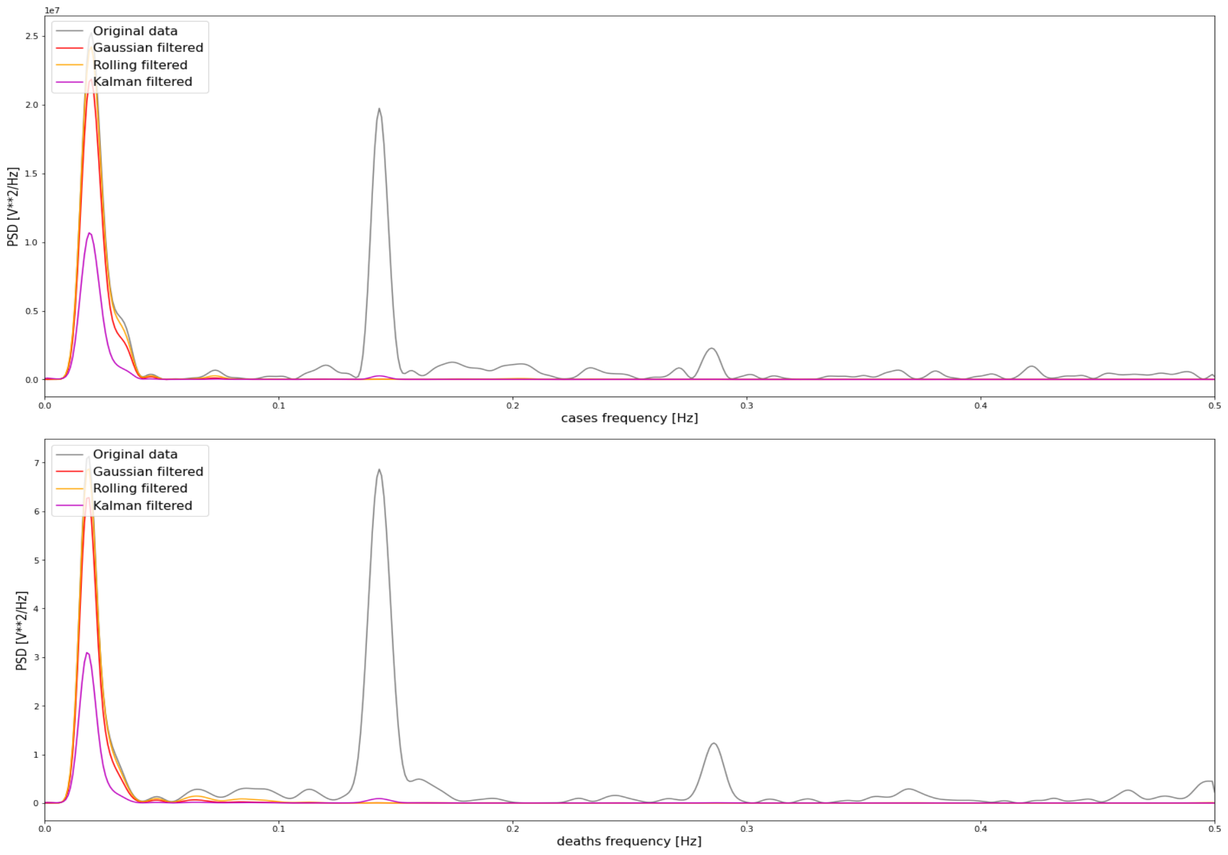


*Fig.12. Spectral density functions.*

## *6. Auto-regression model.*

For building Auto-regression model AR Pipeline from FEDOT framework was used. The generated auto-regression model was used to forecast the future values of time series. All in all, four AR models were built – for both filtered and non-filtered time series of both target variables. The results of forecasting for non-filtered time series are shown on figure 13, for filtered data – on figure 14.

As one can see the generated forecasts are not accurate.

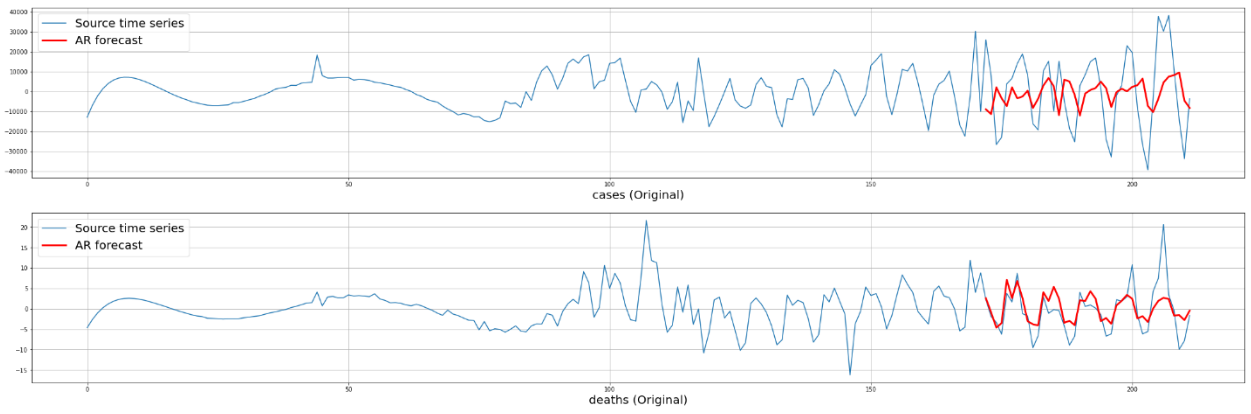


*Fig.13. AR model forecasting non-filtered time series.*

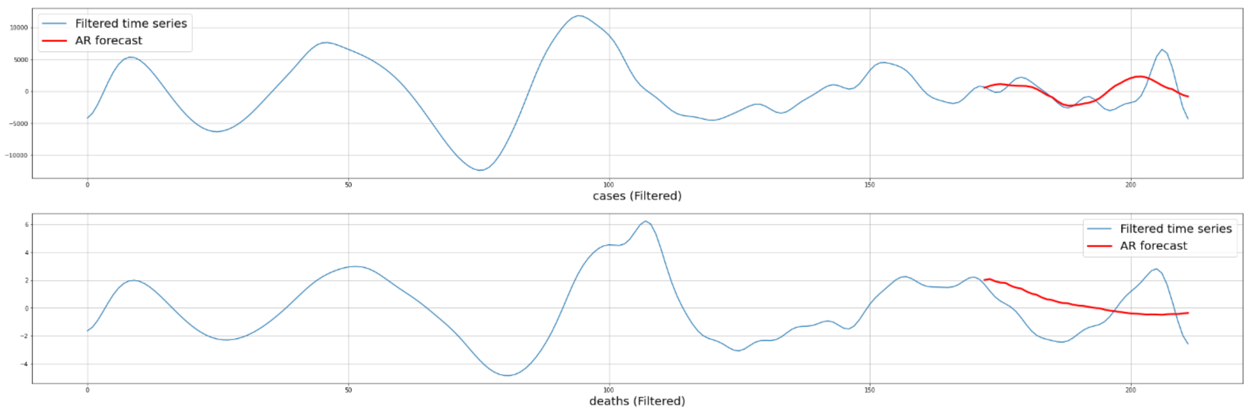


*Fig.14. AR model forecasting filtered time series.*

For building AutoRegressive Integrated Moving Average with eXogenous regressors model (ARIMA) function ARIMAX from *statsmodels* module was used. The parameters for these models can be seen in Jupyter notebook for this laboratory work The generated ARIMA model was used to forecast the future values of time series. All in all, two ARIMA models were built – for non-filtered time series of both target variables. The results of forecasting for non-filtered time series are shown on figure 15 and figure 16. And the same situation with the SARIMAX model which is good for seasonal time series (fig. 17, fig. 18).

As one can see the generated forecasts are rather better compared with AR model results.

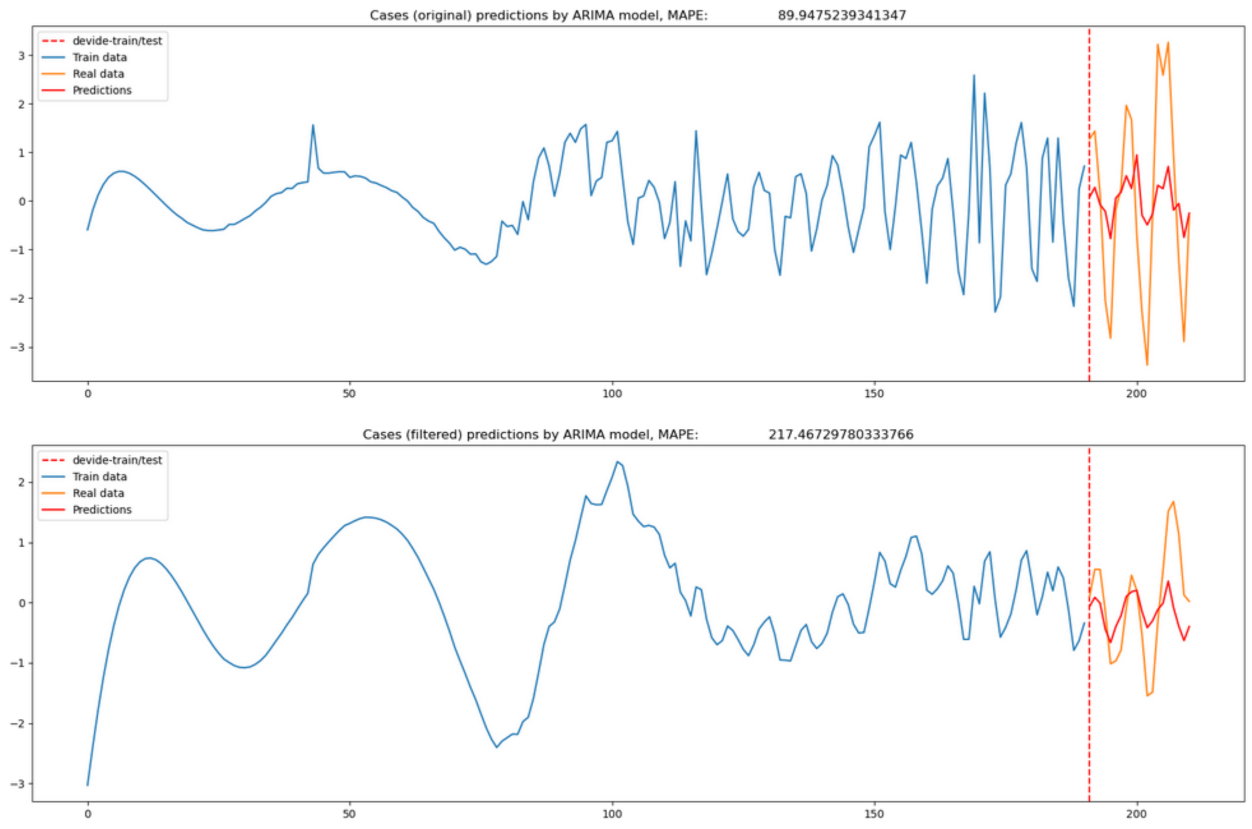


*Fig.15. ARIMA model forecasting for CASES.*

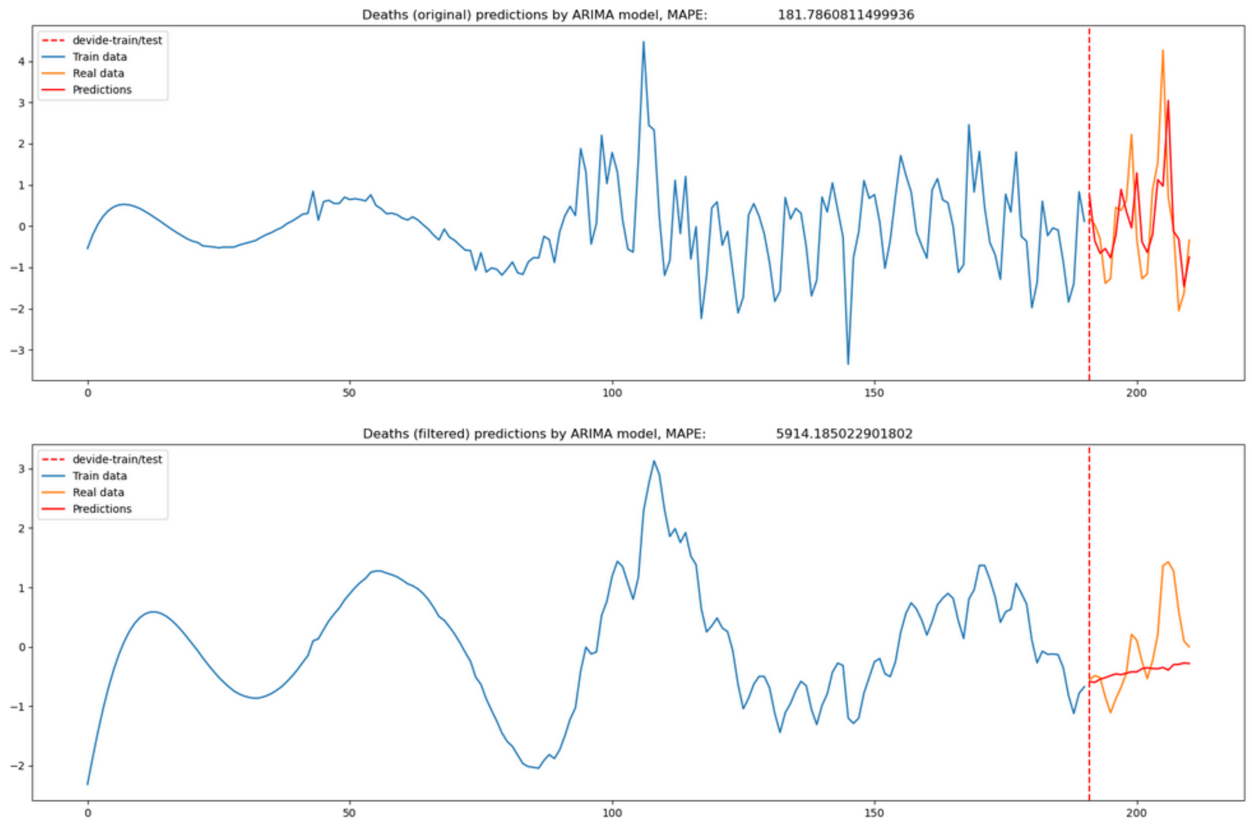


*Fig.16. ARIMA model forecasting for DEATHS.*

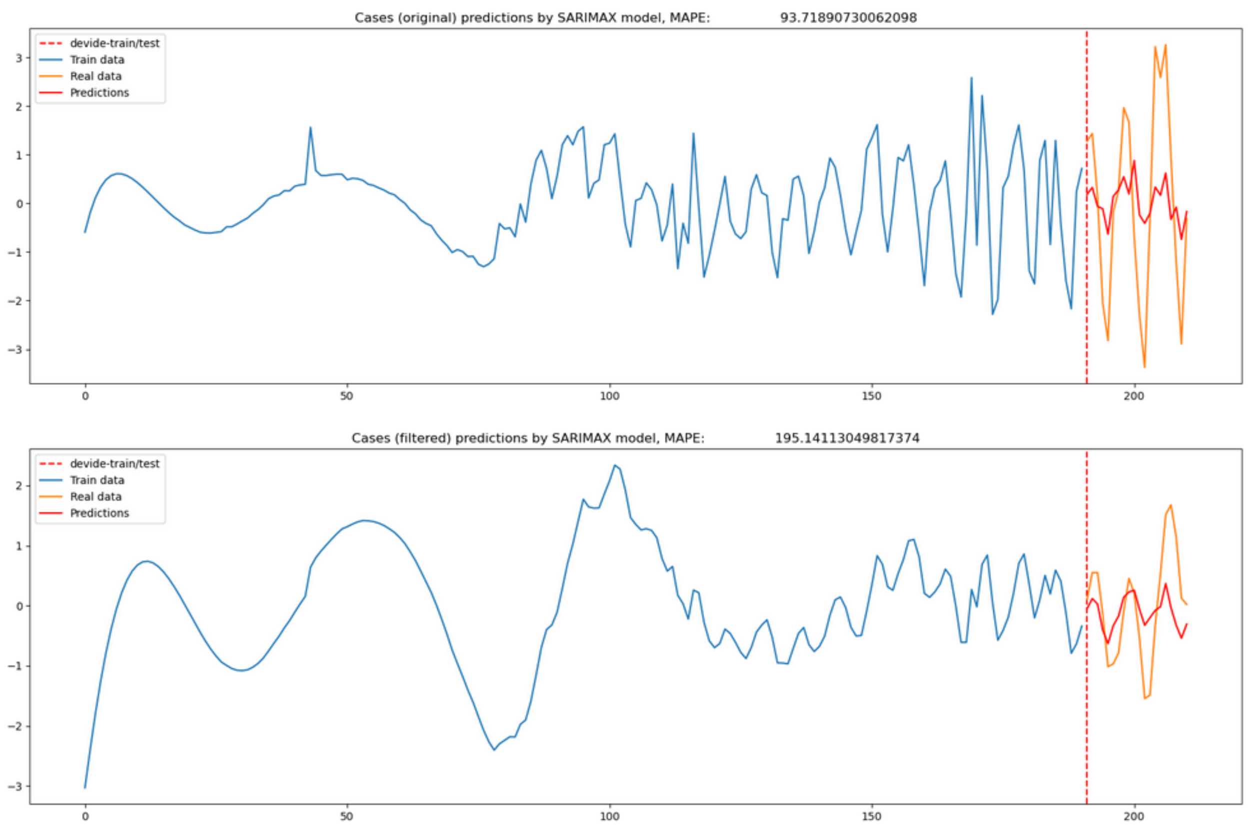


*Fig.17. SARIMAX model forecasting for Cases.*

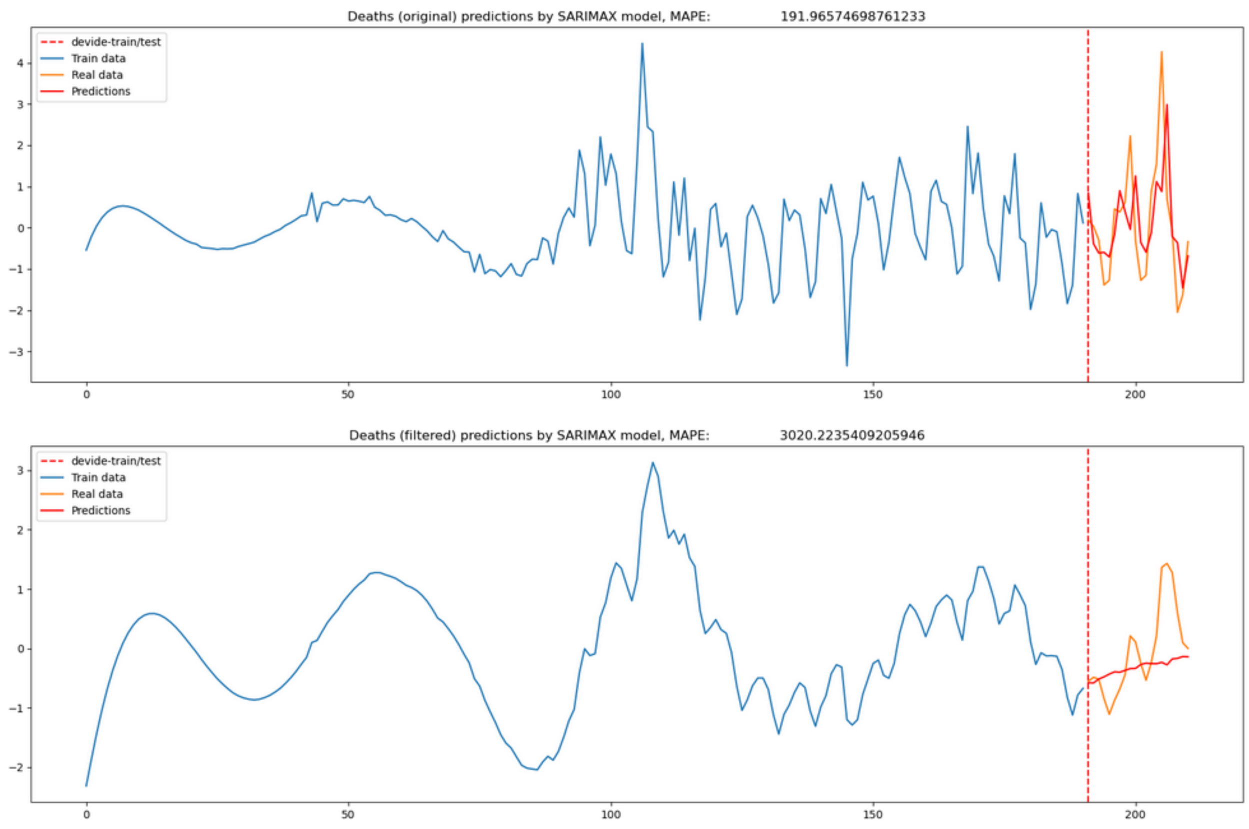
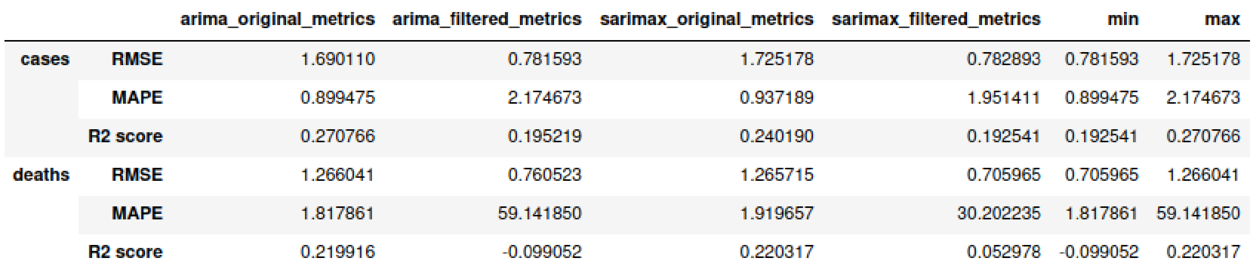


*Fig.18. SARIMAX model forecasting for Deaths.*

The result metrics of all predictions can be viewed on figure 19.



| | | arima_original_metrics | arima_filtered_metrics | sarimax_original_metrics | sarimax_filtered_metrics | min | max |
|---|---|---|---|---|---|---|---|
| **cases** | **RMSE** | 1.690110 | 0.781593 | 1.725178 | 0.782893 | 0.781593 | 1.725178 |
| | **MAPE** | 0.899475 | 2.174673 | 0.937189 | 1.951411 | 0.899475 | 2.174673 |
| | **R2 score** | 0.270766 | 0.195219 | 0.240190 | 0.192541 | 0.192541 | 0.270766 |
| **deaths** | **RMSE** | 1.266041 | 0.760523 | 1.265715 | 0.705965 | 0.705965 | 1.266041 |
| | **MAPE** | 1.817861 | 59.141850 | 1.919657 | 30.202235 | 1.817861 | 59.141850 |
| | **R2 score** | 0.219916 | -0.099052 | 0.220317 | 0.052978 | -0.099052 | 0.220317 |

*Fig.19. Results of forecasting via three metrics for both models: ARIMA and SARIMAX.*

## *6. Build model in a form of linear dynamical system, using chosen predictors.*

Build a model in a form of linear dynamical system, using chosen predictors. To analyze residual error and to define appropriate order of model.

A linear dynamical system was implemented in two forms: univariate and multivariate. For the first one, target variable ‘cases’ was used in lagged and ridge regressions pipeline (fig. 20).

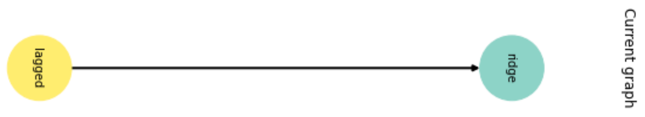


*Fig. 20. Univariate model scheme.*

For the second - ‘deaths’ in lagged and lasso regressions and cases in lagged and ridge (fig. 21).

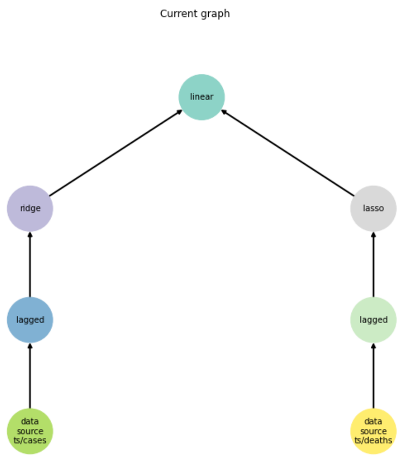


*Fig. 21. Multivariate forecasting model scheme.*

Models demonstrated quite accurate predictions for 20 values with evaluation on test datasets (fig. 22-23). MAE, MSE and MAPE errors (fig. 24-25) demonstrate that the first model has a little bit higher accuracy.

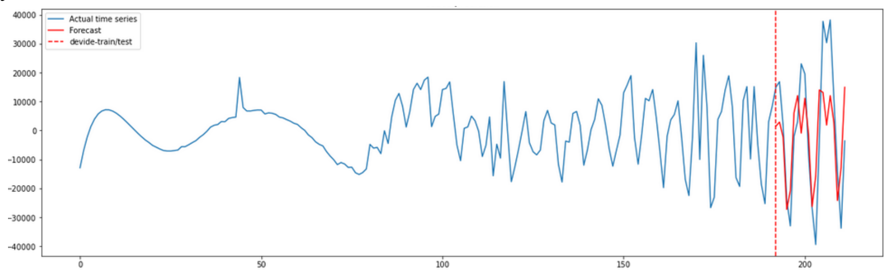


*Fig. 22. Univariate model prediction.*

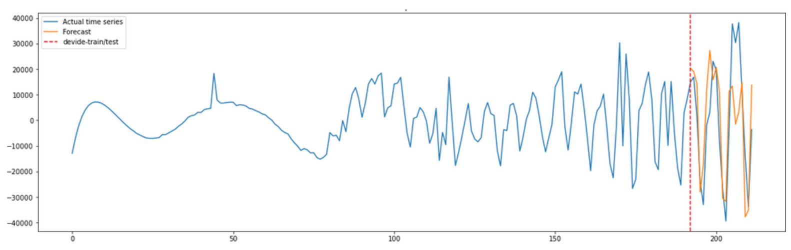


*Fig. 23. Multivariate model prediction.*

```
MAE metric value: 14112.957614431361
MAPE metric value: 1.4476150450236713
MSE metric value: 265724871.76924127
```

*Fig. 24. Univariate result metrics.*

```
MAE metric value: 13414.557887756455
MAPE metric value: 1.985873058661581
MSE metric value: 281393427.73515576
```

*Fig. 25. Multivariate result metrics.*

## Sourcecode

- The full repository with all the labs:
  https://github.com/RazinAleksandr/M-M-MSA-ITMO
- The repo with Datasets and additional used Data info:
  https://github.com/RazinAleksandr/M-M-MSA-ITMO/tree/main/Datasets
- The Lab4 ipynb file:
  https://github.com/RazinAleksandr/M-M-MSA-ITMO/tree/main/Lab_4/lab_4.ipynb

Furthermore, you can find README file with links for every lab folder on the main GitHub repository.