Politecnico di Bari

BIG DATA COURSE

PROJECT:
PREDICTION MODEL FOR COVID-19 NEW DEATHS

STUDENT: KLAJDI ISTERI
DATE: 19/04/2021

Summary

Introduction

Overview

The main purpose of this project is the analysis of the "Our World in Data" Covid-19 Dataset and the creation of a ML model for the prediction of new deaths related to the coronavirus.

We will start by setting up the development environment then we will focus on the exploration and visualization of the data contained in the dataset to understand and develop the ML model. Finally, we will test the model on data before vaccinations and on data after vaccinations.

The analysis is generalized for all the locations contained in the dataset but in this report, we will focus mainly on Italy.

Enviroment Setup

Toolkit

For this project we will work on Windows Environment, so before starting we need to install some applications and frameworks:

1. Anaconda: is a free, easy-to-install package manager, environment manager, and Python distribution with open source packages [1]. Anaconda is platform-agnostic, so you can use it whether you are on Windows, macOS, or Linux.
   Anaconda is composed of two components:
   - Anaconda Navigator: a desktop graphical user interface (GUI).
   - Anaconda Prompt: if you prefer to use a command line style (or terminal on Linux or macOS).

   As we can see Anaconda includes the Jupyter framework, in particular we will use the Jupyter Notebook [2] that is an open-source web application that allows us to create and share documents that contain live code, equations, visualizations and narrative text.

   Notebook is also used for cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and it also integrates leverage Big Data tools, such as Apache Spark.

2. Apache Spark: Apache Spark™ is a unified analytics engine for large-scale data processing. Spark is a cluster computing platform designed to be fast.
   It extends MapReduce model to efficiently support more types of computations, including interactive queries.
   It is also designed to be highly accessible thanks to simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. Spark can work with its standalone cluster and it can also run in Hadoop clusters and access any Hadoop data source.
   In this project we used Spark version 3.1.1 in local mode, but it can be also used in cluster environments.

Jupyter Notebook

As we said before we will use the notebook for our work, but before start working, we need to set some libraries for the python code we will develop.

In [1]:
```
# Import spark libraries
import findspark
findspark.init()
findspark.find()
import pyspark
from pyspark import SparkContext, SparkConf

# import pyspark.sql libraries
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.functions import row_number, monotonically_increasing_id
from pyspark.sql import Window

# Import pyspark.ml libraries
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import GeneralizedLinearRegression
from pyspark.ml.feature import StandardScaler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Import pandas and plotly libraries
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px

# Initiate Spark Context
conf = pyspark.SparkConf().setAppName('SparkApp').setMaster('local')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
spark=SparkSession.builder.getOrCreate()
```

In [2]:
```
# Get infos about spark session
spark
```

SparkSession - in-memory
SparkContext

Spark UI
Version
`v3.1.1`
Master
`local`
AppName
`SparkApp`

We imported:

1. FindSpark: as we are working on windows using a notebook, we need to make the notebook see spark, PySpark is not on system path by default, but that does not mean it can't be used as a regular library. We can address this by adding pyspark to system path at runtime using a windows environment variable. [3]

2. PySpark: we also imported SQL and ML libraries provided by spark for managing the dataframe's data[4].

3. Data Science Libraries: for managing and visualizing data we used some famous libraries like Pandas, Seaborn, NumPy and Plotly to create interactive plots.[5],[6],[7],[8]

As we can see a Spark session was initiated in memory so we can now work with spark on notebook.

Note that spark is currently working in local mode into notebook environment, but all the code can also run on Spark clusters and Hadoop clusters. Everything is set, so we are ready to start the data analysis with this powerful engine on Jupyter Notebook.

Data Exploration and Visualization

To better understand the problem and the creation of the model we will explore data to find relations between columns and we will analyze some statistical data of the dataset using some visualization techniques for a clear and easy view.

Exploration

For the acquiring data part, we will import OS libraries in python, in this way we can read data from windows paths, we will also use the spark read function to import the .csv as Dataframe.

Dataset updated at: 02/05/2021

InferSchema is used to keep track of column types.

```
In [3]: # Import python Operative System Libraries for interaction with OS
        import os
        from os.path import isfile, join

        # Define input data location path
        loc = os.path.abspath("")
        data_loc = f"{loc}/data/"
```

```
In [4]: # Read file csv that contains header, also get the type information from fields
        df_init = spark.read.csv(f"{data_loc}owid_covid.csv",inferSchema=True, header=True)
```

The "Our World in Data" Covid-19 Dataframe is composed of the following columns:

iso_code	new_deaths_smoothed_per_million	new_tests_smoothed_per_thousand	population_density
continent	reproduction_rate	positive_rate	median_age
location	icu_patients	tests_per_case	aged_65_older
date	icu_patients_per_million	tests_units	aged_70_older
total_cases	hosp_patients	total_vaccinations	gdp_per_capita
new_cases	hosp_patients_per_million	people_vaccinated	extreme_poverty
new_cases_smoothed	weekly_icu_admissions	people_fully_vaccinated	cardiovasc_death_rate
total_deaths	weekly_icu_admissions_per_million	new_vaccinations	diabetes_prevalence
new_deaths	weekly_hosp_admissions	new_vaccinations_smoothed	female_smokers
new_deaths_smoothed	weekly_hosp_admissions_per_million	total_vaccinations_per_hundred	male_smokers
total_cases_per_million	total_tests	people_vaccinated_per_hundred	handwashing_facilities
new_cases_per_million	new_tests	people_fully_vaccinated_per_hundred	hospital_beds_per_thousand
new_cases_smoothed_per_million	total_tests_per_thousand	new_vaccinations_smoothed_per_million	life_expectancy
total_deaths_per_million	new_tests_per_thousand	stringency_index	human_development_index
new_deaths_per_million	new_tests_smoothed	population	

Specifically, the field types are described below:

```
In [5]: # Print schema and verify type of fields
        df_init.printSchema()
```

```
root
 |-- iso_code: string (nullable = true)
 |-- continent: string (nullable = true)
 |-- location: string (nullable = true)
 |-- date: string (nullable = true)
 |-- total_cases: double (nullable = true)
 |-- new_cases: double (nullable = true)
 |-- new_cases_smoothed: double (nullable = true)
 |-- total_deaths: double (nullable = true)
 |-- new_deaths: double (nullable = true)
 |-- new_deaths_smoothed: double (nullable = true)
 |-- total_cases_per_million: double (nullable = true)
 |-- new_cases_per_million: double (nullable = true)
 |-- new_cases_smoothed_per_million: double (nullable = true)
 |-- total_deaths_per_million: double (nullable = true)
 |-- new_deaths_per_million: double (nullable = true)
 |-- new_deaths_smoothed_per_million: double (nullable = true)
 |-- reproduction_rate: double (nullable = true)
 |-- icu_patients: double (nullable = true)
 |-- icu_patients_per_million: double (nullable = true)
 |-- hosp_patients: double (nullable = true)
 |-- hosp_patients_per_million: double (nullable = true)
 |-- weekly_icu_admissions: double (nullable = true)
 |-- weekly_icu_admissions_per_million: double (nullable = true)
 |-- weekly_hosp_admissions: double (nullable = true)
 |-- weekly_hosp_admissions_per_million: double (nullable = true)
 |-- new_tests: double (nullable = true)
 |-- total_tests: double (nullable = true)
 |-- total_tests_per_thousand: double (nullable = true)
 |-- new_tests_per_thousand: double (nullable = true)
 |-- new_tests_smoothed: double (nullable = true)
 |-- new_tests_smoothed_per_thousand: double (nullable = true)
 |-- positive_rate: double (nullable = true)
 |-- tests_per_case: double (nullable = true)
 |-- tests_units: string (nullable = true)
 |-- total_vaccinations: double (nullable = true)
 |-- people_vaccinated: double (nullable = true)
 |-- people_fully_vaccinated: double (nullable = true)
 |-- new_vaccinations: double (nullable = true)
 |-- new_vaccinations_smoothed: double (nullable = true)
 |-- total_vaccinations_per_hundred: double (nullable = true)
 |-- people_vaccinated_per_hundred: double (nullable = true)
 |-- people_fully_vaccinated_per_hundred: double (nullable = true)
 |-- new_vaccinations_smoothed_per_million: double (nullable = true)
 |-- stringency_index: double (nullable = true)
 |-- population: double (nullable = true)
 |-- population_density: double (nullable = true)
 |-- median_age: double (nullable = true)
 |-- aged_65_older: double (nullable = true)
 |-- aged_70_older: double (nullable = true)
 |-- gdp_per_capita: double (nullable = true)
 |-- extreme_poverty: double (nullable = true)
 |-- cardiovasc_death_rate: double (nullable = true)
 |-- diabetes_prevalence: double (nullable = true)
 |-- female_smokers: double (nullable = true)
 |-- male_smokers: double (nullable = true)
 |-- handwashing_facilities: double (nullable = true)
 |-- hospital_beds_per_thousand: double (nullable = true)
 |-- life_expectancy: double (nullable = true)
 |-- human_development_index: double (nullable = true)
```

These information are important cause we now know the type of data and we also know if the dataframe have some missing fields.

For a better understanding of data, we will analyze some statistic information such as min, max, count, mean and standard deviation to understand the types of data distribution we have.

In [6]:
```
# Print statistical informations about the dataframe
df_init.describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
iso_code	85580	None	None	ABW	ZWE
continent	81451	None	None	Africa	South America
location	85580	None	None	Afghanistan	Zimbabwe
date	85580	None	None	2020-01-01	2021-05-03
total_cases	83470	832778.9957349946	5757477.472047493	1.0	1.52870507E8
new_cases	83468	5835.210092490535	36508.86533731523	-74347.0	905992.0
new_cases_smoothed	82467	5814.830038548904	35814.991549206636	-6223.0	826374.286
total_deaths	73790	23163.885973709173	137024.56622028106	1.0	3202523.0
new_deaths	73948	139.27223183858928	760.1312758706464	-1918.0	17906.0
new_deaths_smoothed	82467	123.41811175379846	695.9858349571732	-232.143	14435.143
total_cases_per_million	83019	10162.936744359886	19463.958179571233	0.001	171901.896
new_cases_per_million	83017	74.39997649878887	175.60479416028457	-2153.437	8652.658
new_cases_smoothed_per_million	82021	74.5016326550511	149.09617539623656	-276.825	2648.773
total_deaths_per_million	73352	226.73812504090714	397.8053083164839	0.001	2877.95
new_deaths_per_million	73510	1.5092371786151573	3.977495993528583	-76.445	218.329
new_deaths_smoothed_per_million	82021	1.338711878665185	2.9391449349779664	-10.921	63.14
reproduction_rate	69306	1.0182404120855324	0.35633672563155433	-0.01	5.77
icu_patients	8691	1087.9978138303993	3033.991358320877	0.0	29990.0
icu_patients_per_million	8691	26.403532965136367	27.860613702241253	0.0	192.642
hosp_patients	10821	4832.752795490251	12433.561423445204	0.0	129637.0
hosp_patients_per_million	10821	173.69404269475964	216.330380008436	0.0	1532.573
weekly_icu_admissions	790	280.36216708860724	588.1153416060671	0.0	4037.019
weekly_icu_admissions_per_million	790	21.10003544303796	37.10012126341996	0.0	279.13
weekly_hosp_admissions	1298	3994.2353412943	11634.79089101694	0.0	116232.0
weekly_hosp_admissions_per_million	1298	115.32590986132524	230.24720305086265	0.0	2656.911
new_tests	38945	44019.99345230453	228082.89074859317	-239172.0	3.2022805E7
total_tests	38652	5963484.145788058	2.702657004024681E7	0.0	4.13502739E8
total_tests_per_thousand	38652	227.4760523388175	495.3537686855608	0.0	6233.953
new_tests_per_thousand	38945	1.9372400564899257	15.289038589037073	-23.01	2827.217
new_tests_smoothed	44625	41416.30742857143	148450.0041091272	0.0	4594014.0
new_tests_smoothed_per_thousand	44625	1.7755527619047682	4.810456516565619	0.0	405.595
positive_rate	42904	0.08897072534029671	0.09759395562535529	0.0	0.742
tests_per_case	42311	159.4553425823062	864.9287640966207	1.3	44258.7
tests_units	46079	None	None	people tested	units unclear
total_vaccinations	9551	1.4957485974452937E7	6.869741373731461E7	0.0	1.162067962E9
people_vaccinated	8910	9225179.1661055	3.902779279540956E7	0.0	6.04516098E8
people_fully_vaccinated	6576	4772619.979622871	1.886845361435513 6E7	1.0	2.75959041E8
new_vaccinations	8110	424912.7676942047	1698118.626609131	0.0	2.4728855E7
new_vaccinations_smoothed	15322	226123.2664143062	1159589.1827649393	0.0	2.0323434E7
total_vaccinations_per_hundred	9551	13.58571144382781	21.893459977692924	0.0	211.08
people_vaccinated_per_hundred	8910	9.51923007856343	13.923543885901049	0.0	111.32
people_fully_vaccinated_per_hundred	6576	5.181970802919693	9.721152822921756	0.0	99.76
new_vaccinations_smoothed_per_million	15322	2784.0497324109124	4620.116653779478	0.0	118759.0
stringency_index	72673	58.71260275480682	21.648871289508666	0.0	100.0
population	85029	1.283663044966776E8	6.902714046105045E8	809.0	7.794798729E9
population_density	79659	349.79385832101934	1703.2164207822327	0.137	20546.766
median_age	77078	30.521583331174902	9.115053092093216	15.1	48.2
aged_65_older	76198	8.772912753614143	6.223711873362256	1.144	27.049
aged_70_older	76646	5.556767946141541	4.248685782933569	0.526	18.493
gdp_per_capita	77418	19139.031322351304	19826.54402373803	661.24	116935.6
extreme_poverty	52700	13.35125616698124	19.943899746923773	0.1	77.6
cardiovasc_death_rate	78005	257.8088030126566	118.77751200084232	79.37	724.417
diabetes_prevalence	79158	7.821495237373721	3.9780571783350993	0.99	30.53
female_smokers	61115	10.519806561401039	10.402752297957854	0.1	44.0
male_smokers	60214	32.657161872646405	13.475414080788601	7.7	78.1
handwashing_facilities	39197	50.91309169068946	31.763061733684147	1.188	98.999
hospital_beds_per_thousand	71182	3.0294416846951417	2.4634261515052818	0.1	13.8
life_expectancy	81224	73.16502240718631	7.549745661661656	53.28	86.75
human_development_index	77890	0.7271036590064779	0.15005860466163015	0.394	0.957

Visualization

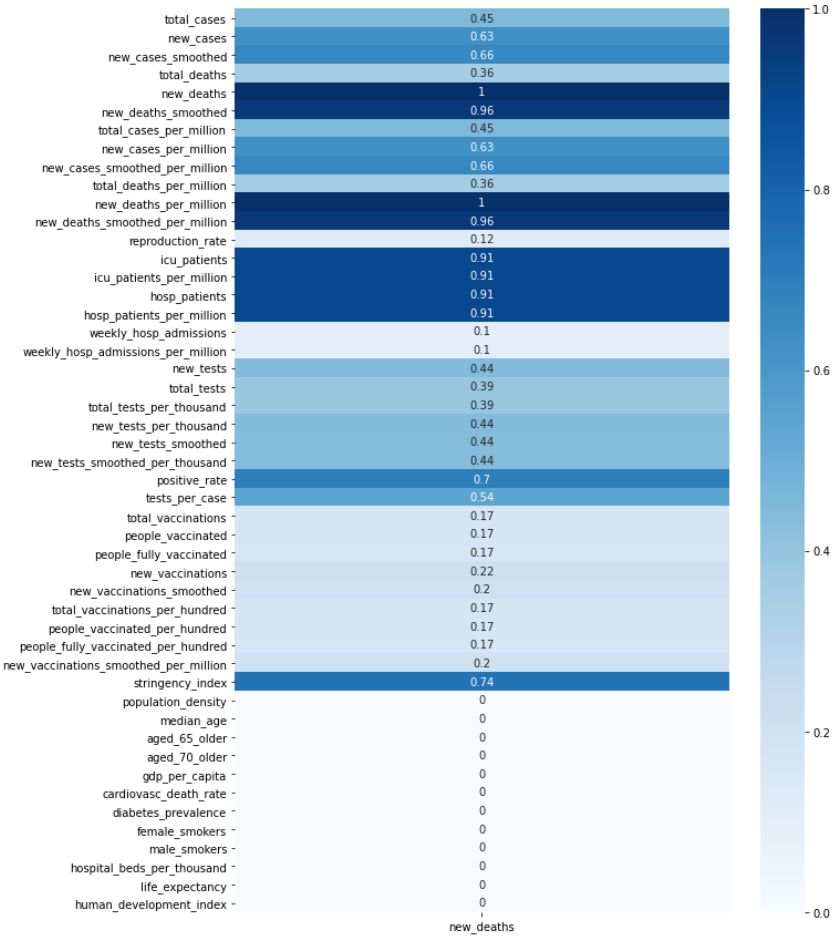
In the table above we have seen information about all the locations contained in the dataframe, but in this project we will focus on the data related to Italy.

The goal of this project is the prediction of new people deaths related to the Covid-19 Coronavirus so the target value to predict is *new_deaths*.

To understand which feature to include in the model for the prediction, we will analyze the correlation between *new_deaths* and all the other columns using a correlation matrix with just one column associated to *new_deaths*.

(Before doing this, we made a dataset data preparation, precisely we dropped all the columns without precious information and filled all the empty fields with zeros because we had all numerical values in dataset, we assumed that an empty field means a value of zero).

```
In [7]: #Plotting correlation of target column new_deaths with the other columns of dataframe
df_init_c = df_init.na.fill(0)
location_of_analysis = "Italy"
columns_to_drop = ['weekly_icu_admissions', 'weekly_icu_admissions_per_million', 'population', 'extreme_poverty', 'handwashing_facilities']
df_init_c = df_init_c.drop(*columns_to_drop)
df_init_c = df_init_c.select("*").where(df_init.location == location_of_analysis).orderBy("date")
df_init_c = df_init_c.toPandas()
corrMatrix = df_init_c.corr().abs().round(4)

corrMatrix = corrMatrix[['new_deaths']]

plt.subplots(figsize=(10,15))
sns.heatmap(corrMatrix, annot=True, cmap="Blues")
```



Has we can see in the above heatmap, *new_deaths* has a strong correlation between these tree fields:

- new_deaths & new_cases → ρ = 0.633
- new_deaths & icu_patients → ρ = 0.912
- new_deaths & hosp_patients → ρ = 0.913

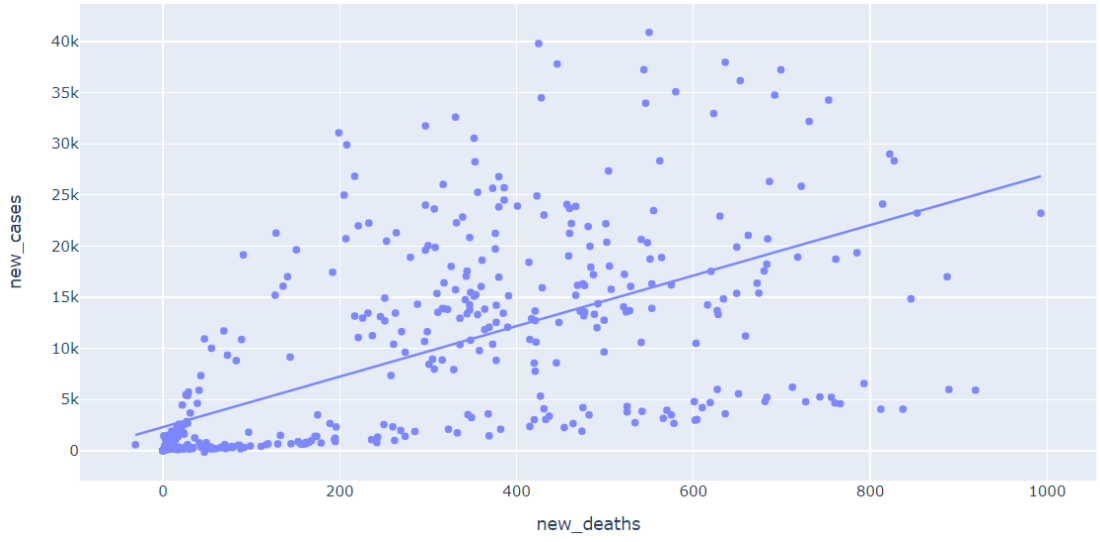
We remember that:

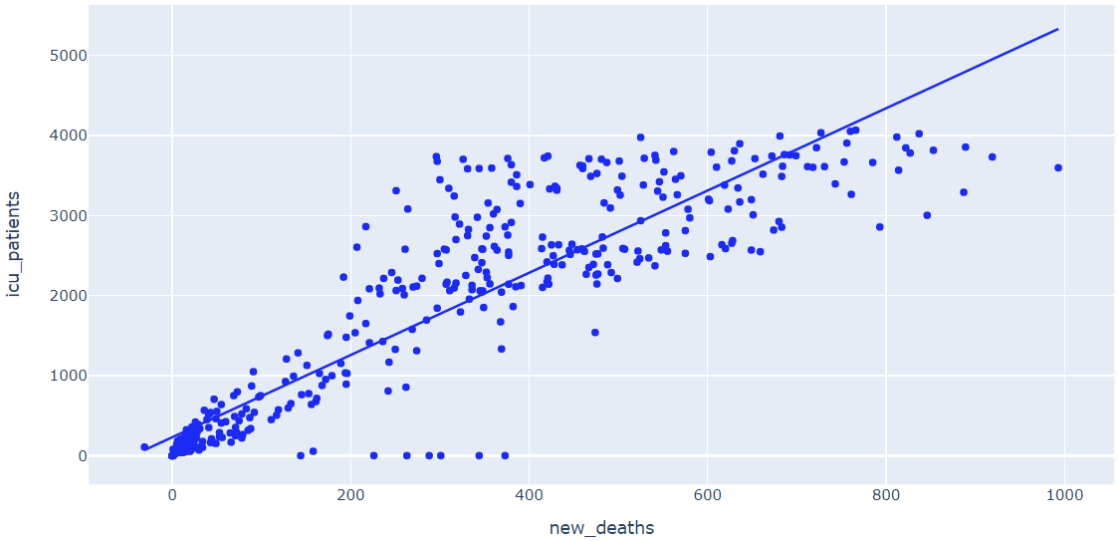
- ρ = 1 ➔ Strong Direct Correlation
- ρ = 0 ➔ No Correlation
- ρ = -1 ➔ Strong Inverse Correlation

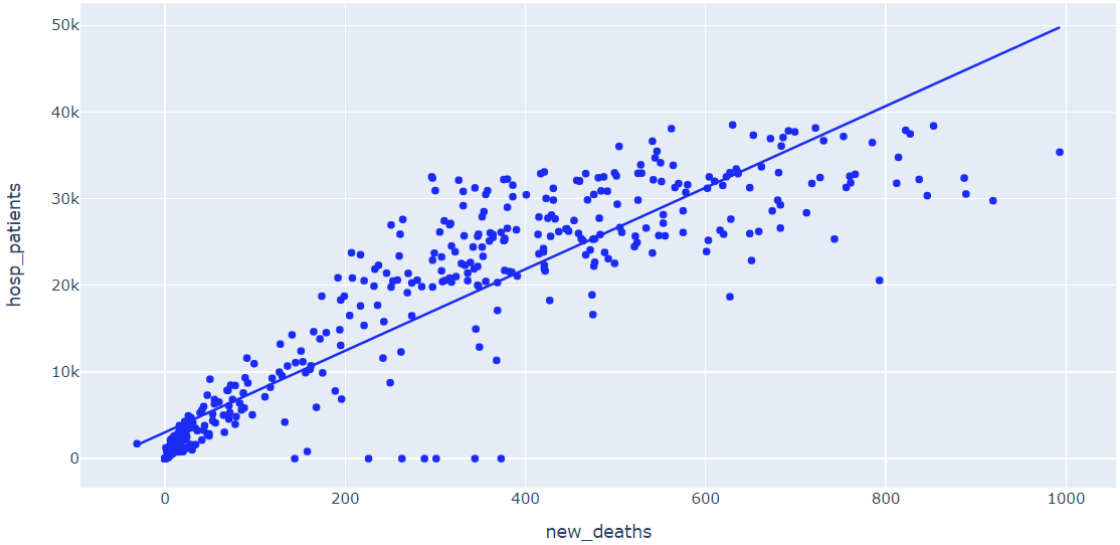
ρ = correlation
Where: -1 < ρ < 1

For a better understanding we will use a scatterplot visualization:

In [8]:
```
# Scatterplot for graphic data correlation visualization
fig1 = px.scatter(df_init_c, x="new_deaths", y="new_cases", color_discrete_sequence=['#7d87ff'], trendline="ols")
fig2 = px.scatter(df_init_c, x="new_deaths", y="icu_patients", color_discrete_sequence=['#1b2bf4'], trendline="ols")
fig3 = px.scatter(df_init_c, x="new_deaths", y="hosp_patients", color_discrete_sequence=['#1b2bf4'], trendline="ols")
fig1.show()
fig2.show()
fig3.show()
```







In the following scatterplots we can see a trendline calculated with the OLS method, the trendline's slope show us the grade of correlation between the two variables, due to these grades we will choose these variables for the prediction.

Machine Learning Model

Here we will use the Apache Spark MLlib library to create and train a Generalized Linear Regression Model for the prediction of *new_deaths,* based on the before selected features "*new_cases*", " *hosp_patients*", "*icu_patients*".

We will split the dataframe in two parts:

1. Dataframe with the data before Vaccinations;
2. Dataframe with data after Vaccinations.

The objective is to split the first dataframe "before vaccinations" in two parts where 70% will be used to train the model and 30% to test the model.

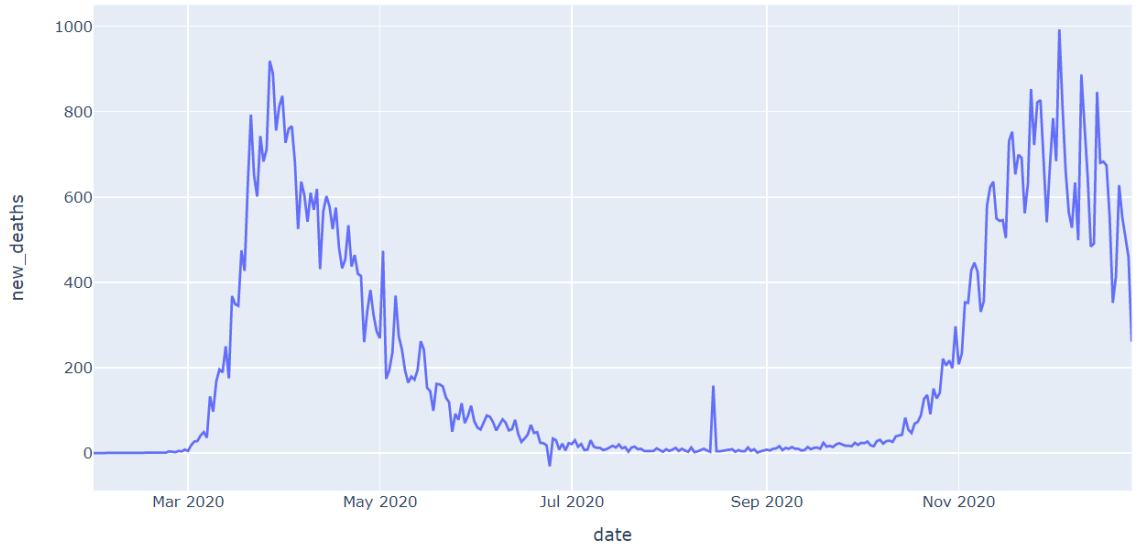
After that we will apply the ML model to the second dataframe "after vaccinations" to also see if the model can give quality predictions to it, we will see that we will have a strange result, but we will discuss about it later.

Dataframe before Vaccinations

As we sed before we are going to create the first dataframe before vaccinations for training and testing the model, we will select the label column to predict *"new_deaths"* and the feature columns and we will also add an index column to keep track of dates (during the 70/30 split the algorithm takes random rows, so index will be useful to keep track of rows after the split).

In [9]:
```
# df contains only rows before vaccinations
df_init = df_init.na.fill(0)
df = df_init.filter(df_init.total_vaccinations == 0)
df = df.select(to_date(col("date"),"yyyy-MM-dd").alias("date"),"new_cases","hosp_patients","icu_patients","new_deaths").where(df.location == location_of_analysis).orderBy("date")


# Define and add an ordered index column in dataframe
df = df.withColumn(
    "index",
    row_number().over(Window.orderBy(monotonically_increasing_id()))-1
)

# Show dataframe and schema
df.show()
df.printSchema()

# Convert dataframe to pandas
pandasDF = df.toPandas()
print("Total Rows 1° dataframe -before vaccinations-: %.3f" % len(pandasDF))

# This date array will be used for the succesive mapping with the index array (before vaccinations)
date_column = pandasDF.loc[:,'date']
x_date = date_column.values

# Plot dataframe
fig = px.line(pandasDF, x="date", y="new_deaths")
fig.show()
```

```
+----------+---------+-------------+------------+----------+-----+
|      date|new_cases|hosp_patients|icu_patients|new_deaths|index|
+----------+---------+-------------+------------+----------+-----+
|2020-01-31|      2.0|          0.0|         0.0|       0.0|    0|
|2020-02-01|      0.0|          0.0|         0.0|       0.0|    1|
|2020-02-02|      0.0|          0.0|         0.0|       0.0|    2|
|2020-02-03|      0.0|          0.0|         0.0|       0.0|    3|
|2020-02-04|      0.0|          0.0|         0.0|       0.0|    4|
|2020-02-05|      0.0|          0.0|         0.0|       0.0|    5|
|2020-02-06|      0.0|          0.0|         0.0|       0.0|    6|
|2020-02-07|      1.0|          0.0|         0.0|       0.0|    7|
|2020-02-08|      0.0|          0.0|         0.0|       0.0|    8|
|2020-02-09|      0.0|          0.0|         0.0|       0.0|    9|
|2020-02-10|      0.0|          0.0|         0.0|       0.0|   10|
|2020-02-11|      0.0|          0.0|         0.0|       0.0|   11|
|2020-02-12|      0.0|          0.0|         0.0|       0.0|   12|
|2020-02-13|      0.0|          0.0|         0.0|       0.0|   13|
|2020-02-14|      0.0|          0.0|         0.0|       0.0|   14|
|2020-02-15|      0.0|          0.0|         0.0|       0.0|   15|
|2020-02-16|      0.0|          0.0|         0.0|       0.0|   16|
|2020-02-17|      0.0|          0.0|         0.0|       0.0|   17|
|2020-02-18|      0.0|          0.0|         0.0|       0.0|   18|
|2020-02-19|      0.0|          0.0|         0.0|       0.0|   19|
+----------+---------+-------------+------------+----------+-----+
only showing top 20 rows

root
 |-- date: date (nullable = true)
 |-- new_cases: double (nullable = false)
 |-- hosp_patients: double (nullable = false)
 |-- icu_patients: double (nullable = false)
 |-- new_deaths: double (nullable = false)
 |-- index: integer (nullable = true)

Total Rows 1° dataframe -before vaccinations-: 331.000
```



*For interactive graph go to the Notebook's HTML file.

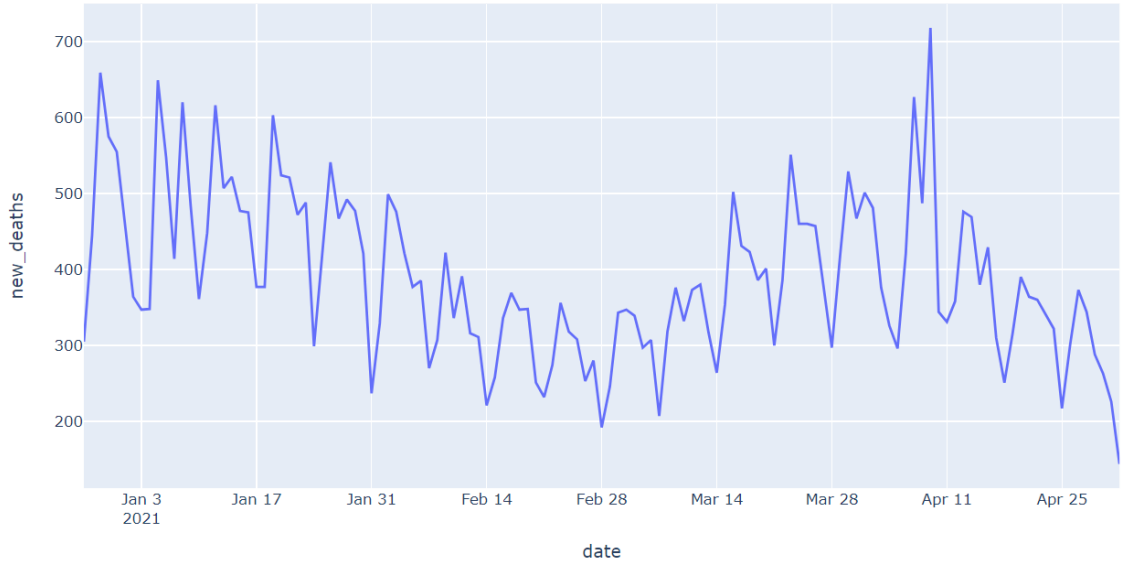
Dataframe after Vaccinations

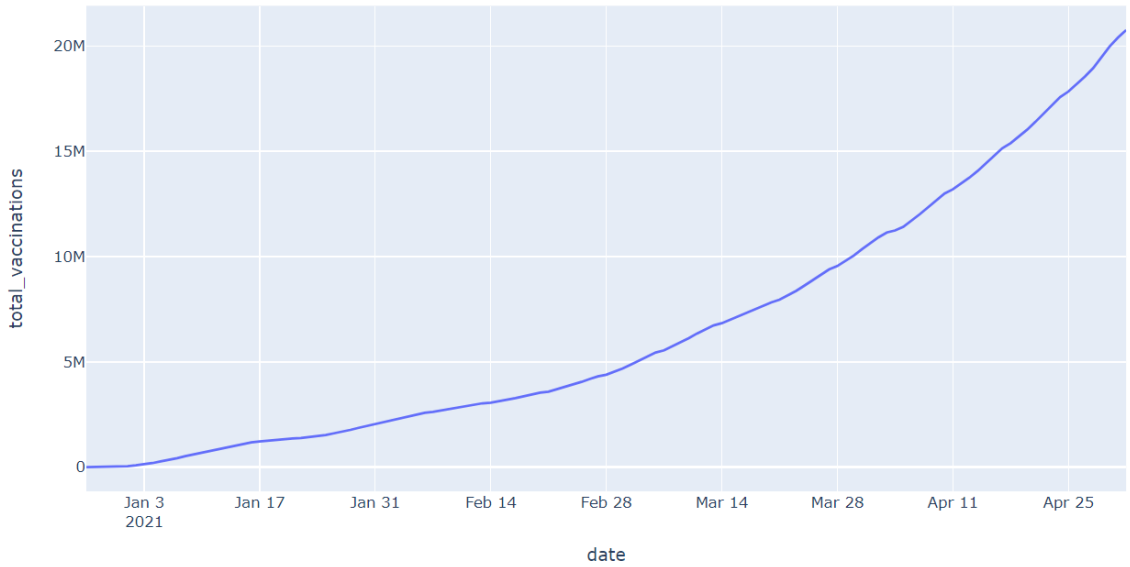
Here we are going to create the second dataframe after vaccinations for testing the model, we will select the label column to predict *"new_deaths"*, the feature columns and here we will also add an index column to keep track of dates.

Furthermore, we will plot vaccinations data to see that we have an exponential function vaccination.

In [10]:
```
# df2 contains all row after vaccination
df2 = df_init.filter(df_init.total_vaccinations > 0)
df2 = df2.select(to_date(col("date"),"yyyy-MM-dd").alias("date"),"new_cases","hosp_patients","icu_patients","new_deaths","total_vaccinations").where(df2.location == location_of_analysis).orderBy("date")


# Define and add an ordered index column in dataframe
df2 = df2.withColumn(
    "index",
    row_number().over(Window.orderBy(monotonically_increasing_id()))-1
)

# Show dataframe and schema
df2.show()
df2.printSchema()

# Convert dataframe to pandas
pandasDF2 = df2.toPandas()
print("Total Rows 2° dataframe -after vaccinationsi- : %.3f" % len(pandasDF2))

# This date array will be used for the succesive mapping with the index array (after vaccinations)
date_column2 = pandasDF2.loc[:,'date']
x_date2 = date_column2.values

# Plot dataframe new deaths
fig2 = px.line(pandasDF2, x="date", y="new_deaths")
fig2.show()

# Plot dataframe vaccinations
fig3 = px.line(pandasDF2, x="date", y="total_vaccinations")
fig3.show()
```

```
+----------+---------+-------------+------------+----------+------------------+-----+
|      date|new_cases|hosp_patients|icu_patients|new_deaths|total_vaccinations|index|
+----------+---------+-------------+------------+----------+------------------+-----+
|2020-12-27|   8937.0|      26151.0|      2580.0|     305.0|            7175.0|    0|
|2020-12-28|   8581.0|      26497.0|      2565.0|     445.0|            8600.0|    1|
|2020-12-29|  11210.0|      26211.0|      2549.0|     659.0|            9609.0|    2|
|2020-12-30|  16202.0|      26094.0|      2528.0|     575.0|           14337.0|    3|
|2020-12-31|  23477.0|      25706.0|      2555.0|     555.0|           39818.0|    4|
|2021-01-01|  22210.0|      25375.0|      2553.0|     462.0|           50877.0|    5|
|2021-01-02|  11825.0|      25517.0|      2569.0|     364.0|           89400.0|    6|
|2021-01-03|  14245.0|      25658.0|      2583.0|     347.0|          124565.0|    7|
|2021-01-04|  10798.0|      25896.0|      2579.0|     348.0|          193333.0|    8|
|2021-01-05|  15375.0|      25964.0|      2569.0|     649.0|          273130.0|    9|
|2021-01-06|  20326.0|      25745.0|      2571.0|     548.0|          338319.0|   10|
|2021-01-07|  18416.0|      25878.0|      2587.0|     414.0|          430571.0|   11|
|2021-01-08|  17529.0|      25900.0|      2587.0|     620.0|          526196.0|   12|
|2021-01-09|  19976.0|      25853.0|      2593.0|     483.0|          613188.0|   13|
|2021-01-10|  18625.0|      26042.0|      2615.0|     361.0|          673555.0|   14|
|2021-01-11|  12530.0|      26245.0|      2642.0|     448.0|          754532.0|   15|
|2021-01-12|  14242.0|      26348.0|      2636.0|     616.0|          836594.0|   16|
|2021-01-13|  15773.0|      26104.0|      2579.0|     507.0|          931146.0|   17|
|2021-01-14|  17243.0|      25667.0|      2557.0|     522.0|         1024900.0|   18|
|2021-01-15|  16144.0|      25363.0|      2522.0|     477.0|         1114574.0|   19|
+----------+---------+-------------+------------+----------+------------------+-----+
only showing top 20 rows

root
 |-- date: date (nullable = true)
 |-- new_cases: double (nullable = false)
 |-- hosp_patients: double (nullable = false)
 |-- icu_patients: double (nullable = false)
 |-- new_deaths: double (nullable = false)
 |-- total_vaccinations: double (nullable = false)
 |-- index: integer (nullable = true)

Total Rows 2° dataframe -after vaccinationsi- : 127.000
```





*For interactive graph go to the Notebook's HTML file.

Features for Predictions

In this part we will select the features and the two labels to predict:

- Features for prediction = [new_cases, icu_patients, hosp_patients, index]
- First Label to predict = new_deaths of first dataframe
- Second Label to predict = new_deaths of second dataframe

In [11]:
```
# Select features to include in the ML model
features = ["new_cases","icu_patients","hosp_patients","index"]

# Select input data for the generalized linear regression (training+test)
lr_data = df.select(col("new_deaths").alias("label"), *features)

# Select input data for the generalized linear regression (test)
test2 = df2.select(col("new_deaths").alias("label"), *features)

# Print schema for input data (training+test) before vaccination
lr_data.printSchema()
lr_data.show()

# Print schema for input data (test) after vaccination
test2.printSchema()
test2.show()
```

```
root
 |-- label: double (nullable = false)
 |-- new_cases: double (nullable = false)
 |-- icu_patients: double (nullable = false)
 |-- hosp_patients: double (nullable = false)
 |-- index: integer (nullable = true)

+-----+---------+------------+-------------+-----+
|label|new_cases|icu_patients|hosp_patients|index|
+-----+---------+------------+-------------+-----+
|  0.0|      2.0|         0.0|          0.0|    0|
|  0.0|      0.0|         0.0|          0.0|    1|
|  0.0|      0.0|         0.0|          0.0|    2|
|  0.0|      0.0|         0.0|          0.0|    3|
|  0.0|      0.0|         0.0|          0.0|    4|
|  0.0|      0.0|         0.0|          0.0|    5|
|  0.0|      0.0|         0.0|          0.0|    6|
|  0.0|      1.0|         0.0|          0.0|    7|
|  0.0|      0.0|         0.0|          0.0|    8|
|  0.0|      0.0|         0.0|          0.0|    9|
|  0.0|      0.0|         0.0|          0.0|   10|
|  0.0|      0.0|         0.0|          0.0|   11|
|  0.0|      0.0|         0.0|          0.0|   12|
|  0.0|      0.0|         0.0|          0.0|   13|
|  0.0|      0.0|         0.0|          0.0|   14|
|  0.0|      0.0|         0.0|          0.0|   15|
|  0.0|      0.0|         0.0|          0.0|   16|
|  0.0|      0.0|         0.0|          0.0|   17|
|  0.0|      0.0|         0.0|          0.0|   18|
|  0.0|      0.0|         0.0|          0.0|   19|
+-----+---------+------------+-------------+-----+
only showing top 20 rows
```

```
root
 |-- label: double (nullable = false)
 |-- new_cases: double (nullable = false)
 |-- icu_patients: double (nullable = false)
 |-- hosp_patients: double (nullable = false)
 |-- index: integer (nullable = true)
```

label	new_cases	icu_patients	hosp_patients	index
305.0	8937.0	2580.0	26151.0	0
445.0	8581.0	2565.0	26497.0	1
659.0	11210.0	2549.0	26211.0	2
575.0	16202.0	2528.0	26094.0	3
555.0	23477.0	2555.0	25706.0	4
462.0	22210.0	2553.0	25375.0	5
364.0	11825.0	2569.0	25517.0	6
347.0	14245.0	2583.0	25658.0	7
348.0	10798.0	2579.0	25896.0	8
649.0	15375.0	2569.0	25964.0	9
548.0	20326.0	2571.0	25745.0	10
414.0	18416.0	2587.0	25878.0	11
620.0	17529.0	2587.0	25900.0	12
483.0	19976.0	2593.0	25853.0	13
361.0	18625.0	2615.0	26042.0	14
448.0	12530.0	2642.0	26245.0	15
616.0	14242.0	2636.0	26348.0	16
507.0	15773.0	2579.0	26104.0	17
522.0	17243.0	2557.0	25667.0	18
477.0	16144.0	2522.0	25363.0	19

only showing top 20 rows

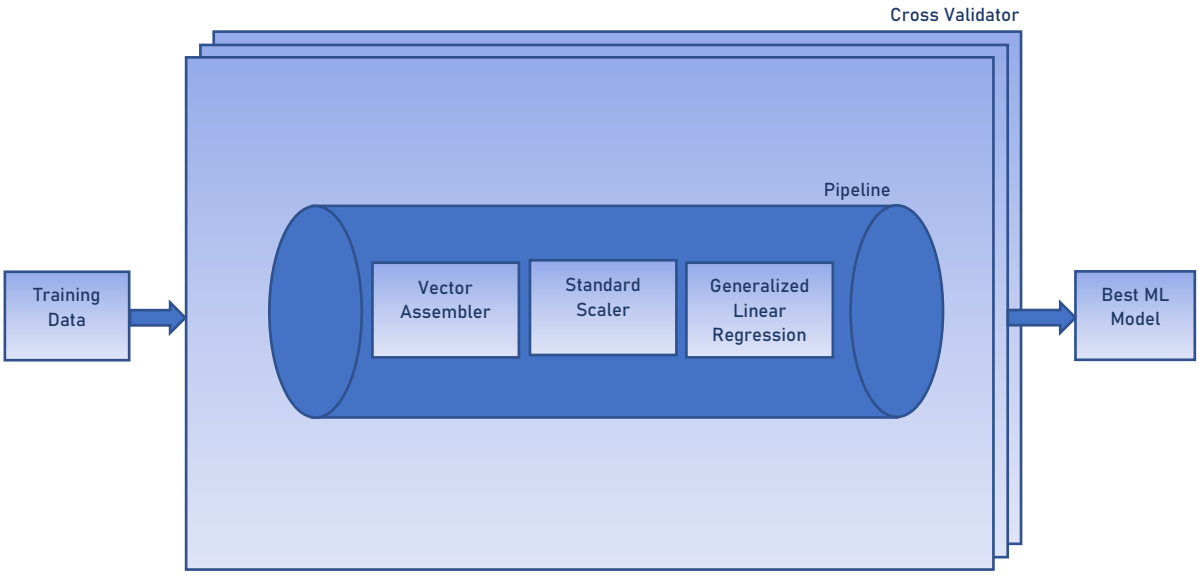
Apache Spark ML Pipeline

Now we have everything set for the creation of the ML model, to do so we will take advantage of the Apache Spark pipeline.

Basically, a pipeline is a set of ML stages composed of transformers and estimators, in our case:

- Vector Assembler: used to create a feature vector;
- Standard Scaler: a transformer that normalizes each feature of the vector assembler to have unit standard deviation, used to have a lower dispersion distribution;
- Generalized Linear Regression: an estimator which takes features and creates a model which is a transformer used to predict future values of a label.

Then we have a Cross-Validator, which is basically a hyperparameter tuner, its main function is to automatically split the dataframe into a set of folds which are used for training/test of the model, for each fold the validator analyzes results of prediction, then compares all the results and choses the best parameters to assign to the model, parameters are chosen inside a Param Grid, then the validator gives us the best model found.



This procedure will be done for the creation and testing of the model with data before vaccinations, so we need to create the train/test dataframes:

```
In [12]: # Divide data in 70% for training and 30% for testing (before vaccinations)
         (training, test) = lr_data.randomSplit([.7, .3])
```

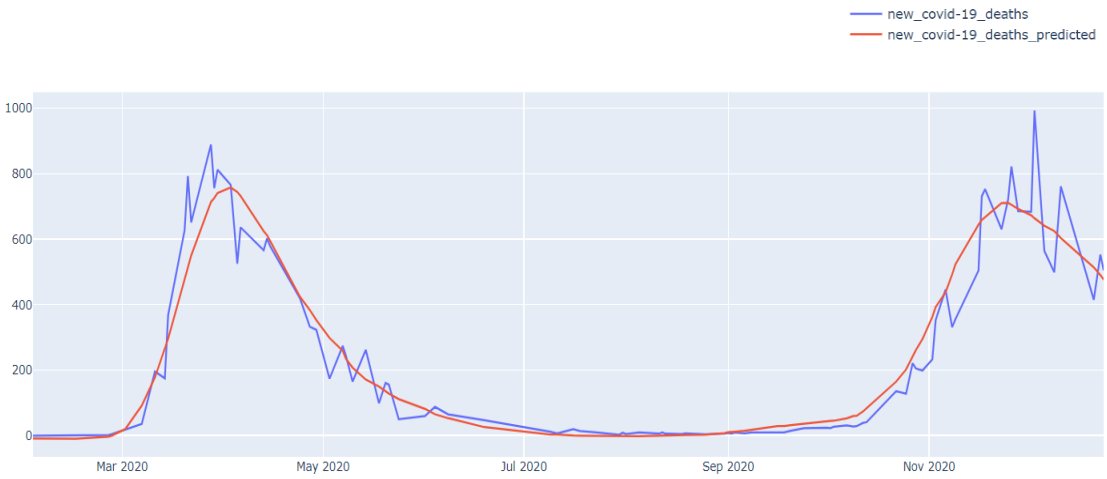
Now we have the data, and we must create the pipeline stages, in particular in the feature vector stage using the vector assembler we also need to scale all value because single features have different scale and distributions so we create the standard scaler stage where we set all features distributions to have a unit standard deviation. Finally, we create the Generalized Linear Regression stage where we apply the machine learning algorithm, we create the parameter grid and we put all stages in the pipeline waiting for the cross validator to find the best model.

In [13]:
```
# Create a array with features (unscaled)
vectorAssembler = VectorAssembler(inputCols=features, outputCol="unscaled_features")

# Scale the features array by normalizing each feature to have unit standard deviation.
standardScaler = StandardScaler(inputCol="unscaled_features", outputCol="features")

# Define GLR with params
lr = GeneralizedLinearRegression()

# Define stages for pipeline
stages = [vectorAssembler, standardScaler, lr]

# Define the pipeline with stages
pipeline = Pipeline(stages=stages)

# We use a ParamGridBuilder to construct a grid of parameters to search over.
param_grid = ParamGridBuilder().addGrid(lr.regParam, [1, 0.001]).build()

# We now treat the Pipeline as an Estimator, wrapping it in a CrossValidator instance.
# This will allow us to jointly choose parameters for all Pipeline stages.
# A CrossValidator requires an Estimator, a set of Estimator ParamMaps, and an Evaluator.
# Note that the evaluator here is a RegressionEvaluator and its default metric is rmse
# where rmse stands for Root Mean Square Error
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=param_grid, evaluator=RegressionEvaluator(), numFolds=10)
```

When we run the cross validation, we lose the data order in results, this is the reason why we put an index column in the dataframe, basically the idea is to create a map between indexes and dates to plot ordered data by date for looking at the results in the graph, to do that we used a list where we pushed all the dates linked to the indexes.

In [14]:
```
# NOTE: ALL THIS CODE IS FOR DATA BEFORE VACCINATIONS (TRAINING+TEST)
# Run cross-validation, and choose the best set of parameters for the model
model = cv.fit(training)

# Model creates prediction on test data by using the best model found with cross-validation
prediction = model.transform(test)
prediction = prediction.orderBy("index")
prediction.show()

# Shows number of predicted rows
pandasDF = prediction.toPandas()
print("Total Rows -before vaccinations- : %.3f" % len(pandasDF))

# Take index column values and put in x_array for the next mapping with date array to create an accurate plot
index_column = pandasDF.loc[:,'index']
x_array = index_column.values

# Define list for the date and indexes(indexes of rows selected by GLR algorithm) mapping.
list_x=[]
for i in range(len(x_array)):         # for i from start to lenght of indexes array created by GLR
    list_x.append(x_date[x_array[i]]) # push date in list based on indexes chosen by GLR for correct mapping of date and index
                                      # see the correct mapping in the plot, in the x axis

# Put columns label and prediction in y1_array and y2_array arrays for plotting
label_column = pandasDF.loc[:,'label']
y1_array = label_column.values
index_column = pandasDF.loc[:,'prediction']
y2_array = index_column.values

# Plot original data and predicted data
fig = go.Figure()
fig.add_trace(go.Scatter(x=list_x, y=y1_array,
                    mode='lines',
                    name='new_covid-19_deaths'))
fig.add_trace(go.Scatter(x=list_x, y=y2_array,
                    mode='lines',
                    name='new_covid-19_deaths_predicted'))
fig.show()
```

```
+-----+---------+------------+-------------+-----+--------------------+--------------------+------------------+
|label|new_cases|icu_patients|hosp_patients|index|   unscaled_features|            features|        prediction|
+-----+---------+------------+-------------+-----+--------------------+--------------------+------------------+
|  0.0|      0.0|         0.0|          0.0|    3|       (4,[3],[3.0])|(4,[3],[0.0315751...|-9.104579437450319|
|  0.0|      0.0|         0.0|          0.0|    6|       (4,[3],[6.0])|(4,[3],[0.0631503...|  -9.1319248065238|
|  0.0|      0.0|         0.0|          0.0|   12|      (4,[3],[12.0])|(4,[3],[0.1263006...|-9.186615544670763|
|  0.0|      0.0|         0.0|          0.0|   13|      (4,[3],[13.0])|(4,[3],[0.1368256...|-9.195730667695257|
|  0.0|      0.0|         0.0|          0.0|   14|      (4,[3],[14.0])|(4,[3],[0.1473507...| -9.20484579071975|
|  0.0|      0.0|         0.0|          0.0|   16|      (4,[3],[16.0])|(4,[3],[0.1684008...|-9.223076036768738|
|  2.0|    131.0|        36.0|        164.0|   26|[131.0,36.0,164.0...|[0.01346131007941...|-3.114129154261537|
|  5.0|    202.0|        56.0|        304.0|   27|[202.0,56.0,304.0...|[0.02075713462627...|  0.50461799780002|
| 18.0|    342.0|       166.0|        908.0|   31|[342.0,166.0,908....|[0.03514326753557...|20.152549040516014|
| 36.0|   1247.0|       567.0|       3218.0|   36|[1247.0,567.0,321...|[0.12813934098496...| 91.72153314313375|
|196.0|   2313.0|      1028.0|       6866.0|   40|[2313.0,1028.0,68...|[0.23767946728004...|177.65195587290464|
|175.0|   3497.0|      1518.0|       9890.0|   43|[3497.0,1518.0,98...|[0.35934504845581...|265.77085230441935|
|368.0|   3590.0|      1672.0|      11335.0|   44|[3590.0,1672.0,11...|[0.36890155103128...| 295.6799787889233|
|627.0|   5986.0|      2655.0|      18675.0|   49|[5986.0,2655.0,18...|[0.61510993996469...| 477.1504384412424|
|793.0|   6557.0|      2857.0|      20565.0|   50|[6557.0,2857.0,20...|[0.67378481061618...| 515.7456205332933|
|651.0|   5560.0|      3009.0|      22855.0|   51|[5560.0,3009.0,22...|[0.57133499268354...| 549.9639294414459|
|889.0|   5974.0|      3856.0|      30532.0|   57|[5974.0,3856.0,30...|[0.61387684285818...| 713.5930381028747|
|756.0|   5217.0|      3906.0|      31292.0|   58|[5217.0,3906.0,31...|[0.53608896705576...| 725.4594001811868|
|812.0|   4050.0|      3981.0|      31776.0|   59|[4050.0,3981.0,31...|[0.41617027344754...| 740.8794296036061|
|766.0|   4585.0|      4068.0|      32809.0|   63|[4585.0,4068.0,32...|[0.47114585277950...|  757.881319629163|
+-----+---------+------------+-------------+-----+--------------------+--------------------+------------------+
only showing top 20 rows

Total Rows -before vaccinations- : 104.000
```




```python
# Call Evaulator for evaluating the model
eval = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")

# Root Mean Square Error
rmse = eval.evaluate(prediction)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval.evaluate(prediction, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval.evaluate(prediction, {eval.metricName: "mae"})
print("MAE: %.3f" % mae)

# R2 - coefficient of determination (R2-->1 perfect prediction, R2=1 so perfect that there is an error in the model)
R2 = eval.evaluate(prediction, {eval.metricName: "r2"})
print("R2: %.3f" %R2)
```

```
RMSE: 75.117
MSE: 5642.502
MAE: 46.026
R2: 0.929
```

As we can see we have a coefficient of determination (R2) really close to 1, that means we have a very good model for predictions.

Conclusion

In the above part we created and tested the prediction model on data, before vaccinations had occurred, here we will test the model on the dataframe with data after vaccinations to analyze what is the situation and we will also see the execution time of the ML model training and test in local mode and in an Apache Standalone Cluster.

Prediction after Vaccinations

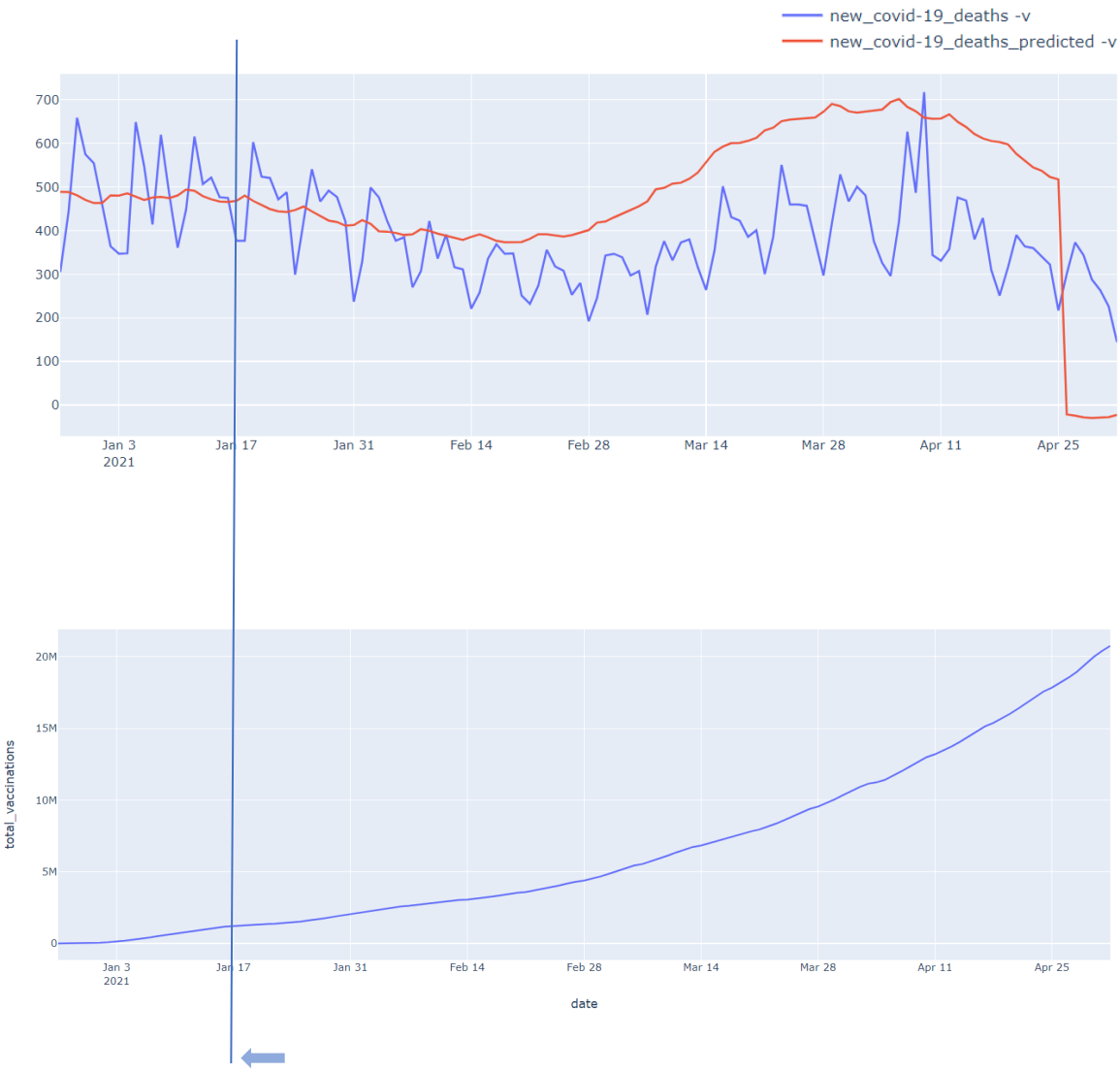
To understand the results, we need to plot them as we did with data before vaccinations:

In [16]:
```
# NOTE: ALL THIS CODE IS FOR DATA AFTER VACCINATIONS (TESTING MODEL)
# Applying model to data after vaccinations for testing and prediction
prediction2 = model.transform(test2)
prediction2 = prediction2.orderBy("index")
prediction2.show()

# Shows number of predicted rows
pandasDF2 = prediction2.toPandas()
print("Total Rows -after vaccinations- : %.3f" % len(pandasDF2))

# Take index column values and put in x_array2 for the next mapping with date array to create an accurate plot
index_column2 = pandasDF2.loc[:,'index']
x_array2 = index_column2.values

# Define list for the date and indexes(indexes of rows selected by GLR algorithm) mapping.
list_x2=[]
for j in range(len(x_array2)):           # for i from start to lenght of indexes array created by GLR
    list_x2.append(x_date2[x_array2[j]]) # push date in list based on indexes chosen by GLR for correct mapping of date and index
                                         # see the correct mapping in the plot, in the x axis

# Put columns label and prediction in y1_array2 and y2_array2 arrays for plotting
label_column2 = pandasDF2.loc[:,'label']
y1_array2 = label_column2.values
index_column2 = pandasDF2.loc[:,'prediction']
y2_array2 = index_column2.values

# Plot original data and predicted data
fig2 = go.Figure()
fig2.add_trace(go.Scatter(x=list_x2, y=y1_array2,
                    mode='lines',
                    name='new_covid-19_deaths -v'))
fig2.add_trace(go.Scatter(x=list_x2, y=y2_array2,
                    mode='lines',
                    name='new_covid-19_deaths_predicted -v'))
fig2.show()
```

```
+-----+---------+------------+-------------+-----+--------------------+--------------------+------------------+
|label|new_cases|icu_patients|hosp_patients|index|   unscaled_features|            features|        prediction|
+-----+---------+------------+-------------+-----+--------------------+--------------------+------------------+
|305.0|   8937.0|      2580.0|      26151.0|    0|[8937.0,2580.0,26...|[0.91834907007425...| 489.1636274603039|
|445.0|   8581.0|      2565.0|      26497.0|    1|[8581.0,2565.0,26...|[0.88176718924775...|  488.520004660835|
|659.0|  11210.0|      2549.0|      26211.0|    2|[11210.0,2549.0,2...|[1.15191821366592...| 481.2611337111114|
|575.0|  16202.0|      2528.0|      26094.0|    3|[16202.0,2528.0,2...|[1.66488660997460...| 470.5672575061037|
|555.0|  23477.0|      2555.0|      25706.0|    4|[23477.0,2555.0,2...|[2.41245173079705...|463.42089554850156|
|462.0|  22210.0|      2553.0|      25375.0|    5|[22210.0,2553.0,2...|[2.28225722796790...|  463.613284543112|
|364.0|  11825.0|      2569.0|      25517.0|    6|[11825.0,2569.0,2...|[1.21511444037462...|481.01619481350303|
|347.0|  14245.0|      2583.0|      25658.0|    7|[14245.0,2583.0,2...|[1.46378902352106...| 480.4371779883596|
|348.0|  10798.0|      2579.0|      25896.0|    8|[10798.0,2579.0,2...|[1.10958187967570...|485.42047538608466|
|649.0|  15375.0|      2569.0|      25964.0|    9|[15375.0,2569.0,2...|[1.57990566771753...|477.74768647731327|
|548.0|  20326.0|      2571.0|      25745.0|   10|[20326.0,2571.0,2...|[2.08866098224563...|470.42175656608674|
|414.0|  18416.0|      2587.0|      25878.0|   11|[18416.0,2587.0,2...|[1.89239302612593...|476.10476024678616|
|620.0|  17529.0|      2587.0|      25900.0|   12|[17529.0,2587.0,2...|[1.80124659833630...| 477.4001842255914|
|483.0|  19976.0|      2593.0|      25853.0|   13|[19976.0,2593.0,2...|[2.05269564997239...|474.80516664017307|
|361.0|  18625.0|      2615.0|      26042.0|   14|[18625.0,2615.0,2...|[1.91386946739766...| 480.8868740950102|
|448.0|  12530.0|      2642.0|      26245.0|   15|[12530.0,2642.0,2...|[1.28755889538216...| 494.3640530399432|
|616.0|  14242.0|      2636.0|      26348.0|   16|[14242.0,2636.0,2...|[1.46348074924443...| 491.4130130923759|
|507.0|  15773.0|      2579.0|      26104.0|   17|[15773.0,2579.0,2...|[1.62080338841682...|479.24786009378516|
|522.0|  17243.0|      2557.0|      25667.0|   18|[17243.0,2557.0,2...|[1.77185778396445...| 472.0661108180667|
|477.0|  16144.0|      2522.0|      25363.0|   19|[16144.0,2522.0,2...|[1.65892664062646...|466.83404063606395|
+-----+---------+------------+-------------+-----+--------------------+--------------------+------------------+
only showing top 20 rows

Total Rows -after vaccinations- : 127.000
```

```
# Call Evaulator for evaluating the model
eval = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")

# Root Mean Square Error
rmse = eval.evaluate(prediction2)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval.evaluate(prediction2, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval.evaluate(prediction2, {eval.metricName: "mae"})
print("MAE: %.3f" % mae)

# R2 - coefficient of determination (R2-->1 perfect prediction, R2=1 so perfect that there is an error in the model)
R2 = eval.evaluate(prediction2, {eval.metricName: "r2"})
print("R2: %.3f" %R2)
```

```
RMSE: 180.718
MSE: 32658.934
MAE: 150.669
R2: -1.428
```



Above we also plotted the vaccinations graph to see an interesting trend, when the vaccinations' function starts to increase significantly, we have a loss of precision in the data predicted by the model due to the features we had chosen to build the model (we can see that analyzing the coefficient of determination) in this case we have a negative R2 value so data are fitted very poorly. In the features we choose to create the model we had not put *total_vaccinations* and this factor decreases the number of new deaths.

So, we can conclude that the model cannot process these new predictions, but we can also deduce that new vaccinations are lowering the number of new Covid-19 deaths, so we can at least predict that the pandemic will end very soon.

Execution Time

Apache Spark allows users to run its jobs in Local Mode and in Distributed Mode using its Standalone Cluster.

Here we will run the code in Local Mode and Spark Standalone Cluster Mode to see the difference in execution times.

As we are on Windows Environment, before running spark in standalone cluster on Jupyter, we need to go in the Spark home and write these commands in CMD to start Spark master:

START MASTER: spark-class org.apache.spark.deploy.master.Master

START WORKER 1: spark-class org.apache.spark.deploy.worker.Worker spark://<MASTER-IP>:7077

START WORKER 2: spark-class org.apache.spark.deploy.worker.Worker spark://<MASTER-IP>:7077

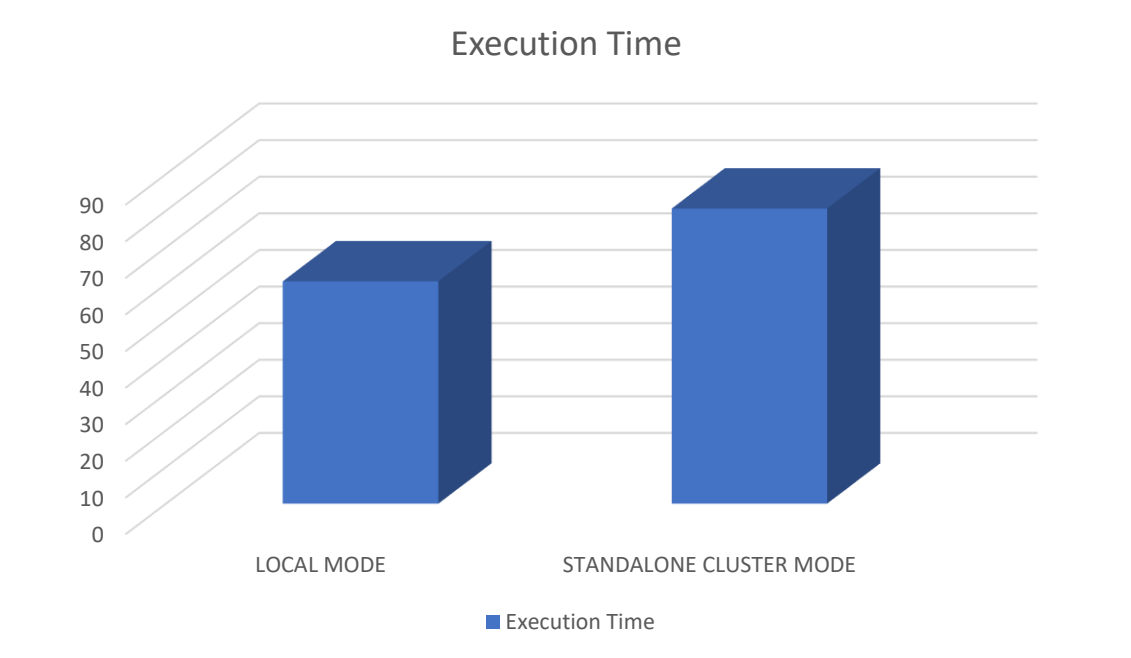
To run Spark in different modes we just set the mode in the Jupyter PySpark Configuration:

LOCAL MODE

conf = pyspark.SparkConf().setAppName('SparkApp').setMaster('local')

STANDALONE CLUSTER

conf = pyspark.SparkConf().setAppName('SparkApp').setMaster('spark://<MASTER-IP>:7077')



*Execution Time in Seconds

Bibliography

[1] «Anaconda Documentation» [Online].
Available: https://docs.anaconda.com.

[2] «Project Jupyter » [Online].
Available: https://jupyter.org/.

[3] «FindSpark Package» [Online].
Available: https://github.com/minrk/findspark.

[4] «PySpark Documentation Package» [Online].
Available: https://spark.apache.org/docs/latest/api/python/.

[5] «Pandas Documentation » [Online].
Available: https://pandas.pydata.org/docs.

[6] «Seaborn API» [Online].
Available: https://seaborn.pydata.org/api.html.

[7] «NumPy Documentation» [Online].
Available: https://numpy.org/doc/.

[8] «Matplotlib Documentation» [Online].
Available: https://matplotlib.org/stable/contents.html.

[9] «Spark Standalone Cluster Mode on Jupyter» [Online].
Available:
https://medium.com/analytics-vidhya/creating-apache-spark-standalone-cluster-with-on-windows-95e66e00a2d8.