Научно-практическая конференция

«Марс-ИТ»

Направление

«Интеллектуальные технологии»

Web-ресурс обработки, визуализации и предсказания статистики заболеваемости и смертности от covid-19.

Выполнили:

Борисов Игнат Николаевич

Бектеев Георгий Александрович

ОГАОУ Многопрофильный лицей 20

Научный руководитель:

Шабалин Александр Станиславович

Ульяновск 2022

Содержание:

Введение:

В настоящее время весь мир терроризирует коронавирус, из-за которого уже заболели сотни миллионов человек, из которых миллионы - это летальные исходы. Какое-то время назад многие страны мира боролись с этим заболеванием на пределе возможностей: постоянно забитые больницы, измученные врачи, работающие по две смены, и, конечно же, постоянная разработка вакцины против covid-19, а в данный момент головной болью для всех является новый штамм - омикрон. Благодаря этому генерируется и собирается большое количество данных об этом вирусном заболевании, которыми можно пользоваться для выявления ранее неизвестных зависимостей, предсказания заболеваемости в разных странах и много другого. Одной из задач которую мы ставим перед собой это удобное и наглядное сравнение заболеваемости и смертности в разных странах мира.

Как было уже сказано выше, данные, которые мы используем в работе постоянно актуализируются и обновляются из открытых источников, а именно из популярного ресурса Our World in Data[1]. Что позволит нам пользоваться самой свежей информацией и пробовать различные модели, по которым мы будем строить прогноз заболеваемости и смертности на несколько дней вперёд.

Исходя из всего этого, **гипотеза**, которую мы хотим проверить такова: «Имея на руках данные, которые дают нам информацию про заболеваемость и смертность граждан, почти во всех странах мира, можно вычислять коэффициент корреляции между ними и использовать его для поиска стран с самым схожим отнормированным ростом и спадом

заболеваемости, что далее можно использовать для обучения модели регрессии и предсказывать заболеваемость на 2-3 дня вперёд».

Проблематика:

Правильное прогнозирование - это очень важная часть моделирования нынешней ситуации в любой стране. Вовремя спрогнозировать появление новых штаммов вируса или новой волны может существенно облегчить работу первичного звена помощи и значительно уменьшить смертность.

Здоровье - это самый важный человеческий ресурс, который очень дорого и трудозатратно восстанавливать. Никто бы не хотел потратить его впустую, для этого все мы должны понимать уровень опасности болезни, которая нас постигла, а также все мы должны уметь ей противостоять. Мы в своей работе построим модель которая сможет прогнозировать заболеваемость и смертность на несколько дней вперед.

Сравнение с аналогами:

Работа по визуализации данных о covid-19 ведётся у многих гигантов индустрии, например, у Яндекс или Google, но у них отсутствует определение похожих стран по статистике заболеваемости, также у нас есть предсказания, чего тоже нет у большинства аналогов.

Мы не первые, кто пытается предсказать заболеваемость на какое-то количество дней вперёд. В сообществе Habr[2] есть много публикаций, которые раскрывают тему предсказания, но мы имеем интерактивный сайт для удобного просмотра данных всем желающим.

Рисунок 1. Пример визуализации данных у Яндекса

Цель:

1. Используя методы машинного обучения, разработать модель предсказания заболеваемости Covid - 19
2. Разработать web-ресурс для визуализации данных заболеваемости и смертности Covid – 19.

Задачи:

1. Визуализация смертности и заболеваемости в разных странах
2. Поиск и сравнение стран с близкой ситуацией заболеваемости
3. Прогноз количества заболевших на несколько дней вперёд

Методы и средства программирования:

Методы и модели, представленные в работе запрограммированы с использованием языка Python. Основные библиотеки применяемые в работе:

Dash - это библиотека Python, которая позволяет визуализировать данные, а также выводить их в отдельном окне, с которым можно взаимодействовать.

Plotly - это библиотека Python, позволяющая визуализировать табличные данные для удобоваримости.

Numpy - это библиотека Python, которую мы использовали для более удобного хранения и сортировки данных.

Pandas - это библиотека Python для упорядочивания и редактирования исходных данных.

Sklearn - это набор библиотек, поддерживающий множество методов, в том числе и регрессию.

Основные определения:

Линейная регрессия - это зависимость, устанавливающая соответствие между случайными переменными, то есть математическое выражение, отражающее связь между зависимой переменной y и независимыми переменными x. $y = f(x, b) + \varepsilon$, $E(\varepsilon)$, где b - параметры модели, ε - случайная ошибка модели

Многомерная регрессия - многомерной регрессия называется в том случае, если в модели более одной независимой переменной

Полином - это то же самое, что и многочлен, либо же какая-то сумма одночленов.

Коэффициент корреляция - это линейная зависимость между двумя объектами, определяющаяся диапазоном от -1 до 1.

Нормирование - это усреднение всех данных, с целью более удобного сравнения.

Сбор и анализ данных:

Начали свой поиск мы с небольшого файла, в котором находились данные о заболеваемости и смертности на 2020-ый год. Для начала этого было достаточно, но лишь для начала. Со временем, мы поняли, что актуальность этих данных очень низкая, так что, перешли к следующему файлу, в котором уже были данные по сей день. Этот файл очень помог нам в работе с данными, так как мы стали лучше представлять то, о чём идёт речь и данные стали более осязаемыми, даже новости пошли в бой.

Преобразования данных:

Для преобразования всех наших данных мы использовали Python и конкретно его библиотеку - Pandas. Так как изначально у нас был файл на 160699 строчек и 68 столбиков (**Рисунок 2**) со статистикой разных стран, с которым было абсолютно неудобно работать, мы стали его преобразовывать в две разные таблицы - это таблица смертности (**Рисунок 3**) и таблица заболеваемости (**Рисунок 4**). Которые у нас вышли уже на 768 строчек, которые описывали каждый день и 129 колонок, которые описывали страны, а в каждой ячейке у нас были смерти за конкретный день в конкретной стране, например, так выглядела часть таблицы заболеваемости:

День от начала отсчета	Russia	World
401	16379.0	381773.0
402	15808.0	405792.0

Таблица 1. Таблица заболеваемости

	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	...	male_smol
0	AFG	Asia	Afghanistan	2020-02-24	5.0	5.0	NaN	NaN	NaN	NaN	...	I
1	AFG	Asia	Afghanistan	2020-02-25	5.0	0.0	NaN	NaN	NaN	NaN	...	I
2	AFG	Asia	Afghanistan	2020-02-26	5.0	0.0	NaN	NaN	NaN	NaN	...	I
3	AFG	Asia	Afghanistan	2020-02-27	5.0	0.0	NaN	NaN	NaN	NaN	...	I
4	AFG	Asia	Afghanistan	2020-02-28	5.0	0.0	NaN	NaN	NaN	NaN	...	I
...	...	...	...	...	...	...	...	...	...	...	...	
160694	ZWE	Africa	Zimbabwe	2022-02-05	230402.0	232.0	141.000	5362.0	5.0	4.143	...	
160695	ZWE	Africa	Zimbabwe	2022-02-06	230402.0	0.0	134.571	5362.0	0.0	3.571	...	
160696	ZWE	Africa	Zimbabwe	2022-02-07	230402.0	0.0	105.143	5362.0	0.0	3.429	...	
160697	ZWE	Africa	Zimbabwe	2022-02-08	230603.0	201.0	107.429	5366.0	4.0	2.286	...	
160698	ZWE	Africa	Zimbabwe	2022-02-09	230740.0	137.0	104.000	5367.0	1.0	2.143	...	

160699 rows × 68 columns

Рисунок 2. Таблица со всеми данными, которые нам требуются по covid-19.

	Afghanistan	Africa	Albania	Algeria	Angola	Argentina	Armenia	Asia	Australia	Austria	...	United Arab Emirates	United Kingdom	United States	Upper middle income	Uruguay	Uzbekistan
1	0.0	0	0.0	0.0	0.0	NaN	0.0	0	0.0	0.0	...	0.0	0.0	0.0	0	0.0	0.0
2	0.0	0	0.0	0.0	0.0	NaN	0.0	0	0.0	0.0	...	0.0	0.0	0.0	0	0.0	0.0
3	0.0	0	0.0	0.0	0.0	NaN	0.0	0	0.0	0.0	...	0.0	0.0	0.0	0	0.0	0.0
4	0.0	0	0.0	0.0	0.0	NaN	0.0	0	0.0	0.0	...	0.0	0.0	0.0	0	0.0	0.0
5	0.0	0	0.0	0.0	0.0	NaN	0.0	0	0.0	0.0	...	0.0	0.0	0.0	0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
765	5.0	509	0.0	9.0	0.0	145.0	6.0	1754	46.0	22.0	...	4.0	259.0	875.0	3822	19.0	2.0
766	7.0	221	9.0	6.0	0.0	100.0	5.0	2029	48.0	18.0	...	2.0	75.0	369.0	2391	12.0	3.0
767	10.0	447	7.0	8.0	0.0	259.0	11.0	2343	54.0	14.0	...	1.0	45.0	2908.0	3046	19.0	2.0
768	12.0	577	6.0	13.0	0.0	284.0	5.0	2573	71.0	25.0	...	4.0	316.0	2902.0	4178	20.0	2.0
769	11.0	557	6.0	11.0	0.0	217.0	6.0	2409	49.0	22.0	...	4.0	276.0	3435.0	4667	20.0	4.0

768 rows × 129 columns

Рисунок 3. Преобразованная таблица смертности

	Afghanistan	Africa	Albania	Algeria	Angola	Argentina	Armenia	Asia	Australia	Austria	...	United Arab Emirates	United Kingdom	United States	Upper middle income	Uruguay	Uzbekis
1	0	0	0.0	0	0	NaN	0	0	0	0	...	0	0.0	0.0	0	0	
2	0	0	0.0	0	0	NaN	0	0	0	0	...	0	0.0	0.0	0	0	
3	0	0	0.0	0	0	NaN	0	0	0	0	...	0	0.0	0.0	0	0	
4	0	0	0.0	0	0	NaN	0	0	0	0	...	0	0.0	0.0	0	0	
5	0	0	0.0	0	0	NaN	0	0	0	0	...	0	0.0	0.0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
765	353	20558	0.0	792	11	21836.0	4032	577587	22985	32258	...	1991	60114.0	104104.0	693376	9074	
766	480	14863	1452.0	378	13	12664.0	2467	583347	25247	29324	...	2015	53326.0	80051.0	529187	6094	
767	733	13575	251.0	502	45	25406.0	1631	563095	27373	27299	...	1704	64322.0	336568.0	528556	6968	
768	815	15133	841.0	610	15	32790.0	2486	633995	31510	27087	...	1615	66634.0	221262.0	721705	8444	
769	811	19268	700.0	585	29	27252.0	2811	590118	11483	38309	...	1538	67677.0	187299.0	745423	8100	

768 rows × 129 columns

Рисунок 4. Преобразованная таблица заболеваемости

Кроме того, эти две таблицы у нас были отсортированы по дням, что нам очень помогло в будущем. Также, несколько стран, которые не имели в себе данных, а если и имели, то были заполнены нулями, были пропущены, так как просто нагружали наш код лишними операциями. Все эти действия нам позволили более продуктивно работать, так что, определённо, это стоило потраченного времени.

Визуализация данных:

Для визуализации данных изначально мы использовали библиотеку Pandas, но она не позволила бы нам вывести все наши графики на отдельную интернет-страничку, чтобы мы могли более наглядно показать все данные, которые были получены и обработаны, из-за чего, в итоге мы перешли к набору библиотек Dash и Plotly, что уже позволило задуманное нами воплотить в реальность. А выглядело это вот так:

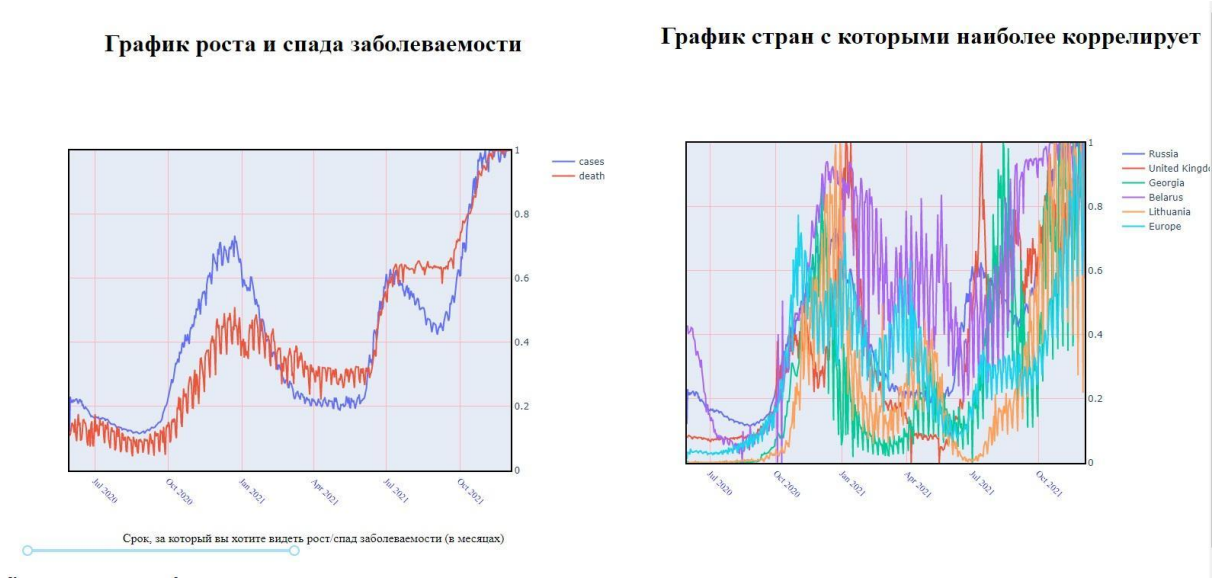

Рисунок 5. Пример графиков с нашего сайта.

И тут, мы начали сталкиваться с одной проблемой за другой. Так как мы хотели сделать сайт, он должен быть хоть немного, но интерактивным.

Для этого мы сделали возможность выбора стран в специальной панели. И выбор срока, за который можно видеть количество заболевших. Для работы этого всего нужна функция обратного вызова, которая и была основным источником нашей головной боли.

Нормирование:

Со временем нам пришлось пронормировать данные, чтобы мы могли сравнить графики заболеваемости разных стран и графики заболеваемости одной страны с графиком смертности всё той же страны, а так как без нормирования это было невозможно, мы пошли искать варианты, которые нам бы помогли с этим. Так как мы поняли, что вариантов было куча, мы попробовали не сильно сложный вариант - это было линейное нормирование (**Рисунок 6**), оно нам показалось подходящим, но мы решили попробовать ещё один вариант - логорифмизация данных (**Рисунок 7**), этот вид нормирования нам уже показался менее подходящим, так как не позволял визуально данные, а этого мы и добивались, так что, остановились мы на первом варианте.

В результате применения линейного нормирования все наши данные находились в диапазоне от 0 до 1, что позволило их сравнивать на графиках. После применения данные в нашей таблице изменились (**Рисунок 8**) и стали выглядеть так:

Таблица 2. Таблица заболеваемости с нормированными данными.

День от начало отсчета	Russia	World
401	0.407336483461825	0.420321065211623
402	0.393136035811987	0.446765291663776

Линейное нормирование:

$$\frac{X - Xmin}{Xmax - Xmin}, \quad (1)$$

где **X** - это данные о заболеваемости и смертность в какой-то день.

Логорифмизация:

$$log_{10}(x), \quad (2)$$

где **X** - это данные о заболеваемости и смертность в какой-то день.

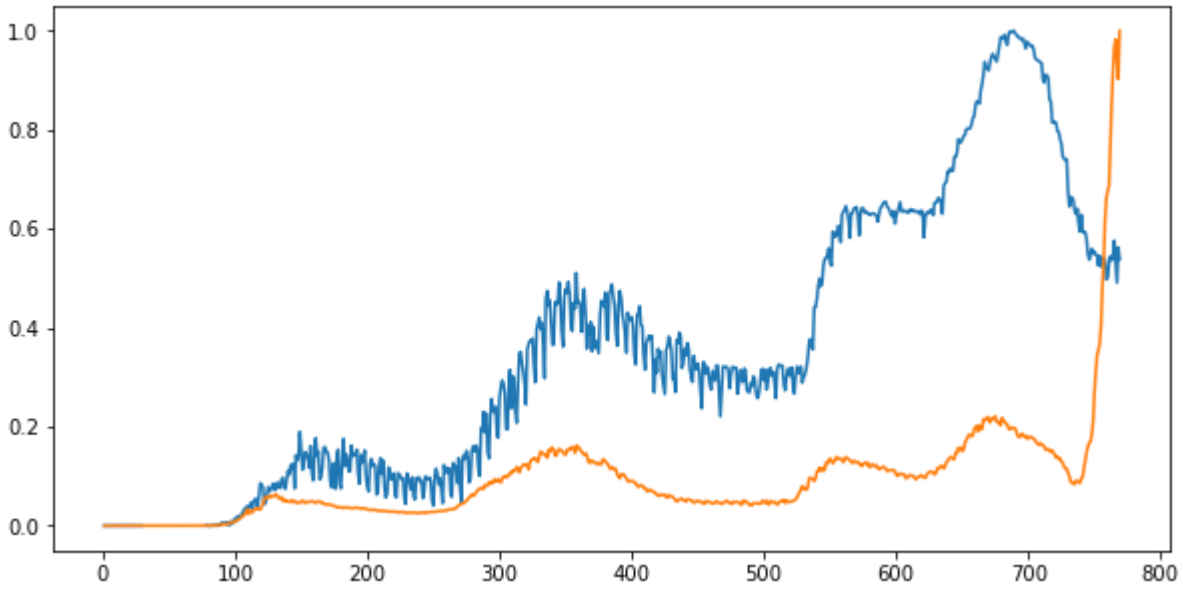

Рисунок 6. График заболеваемости и смертности с нормированными данными линейным методом.

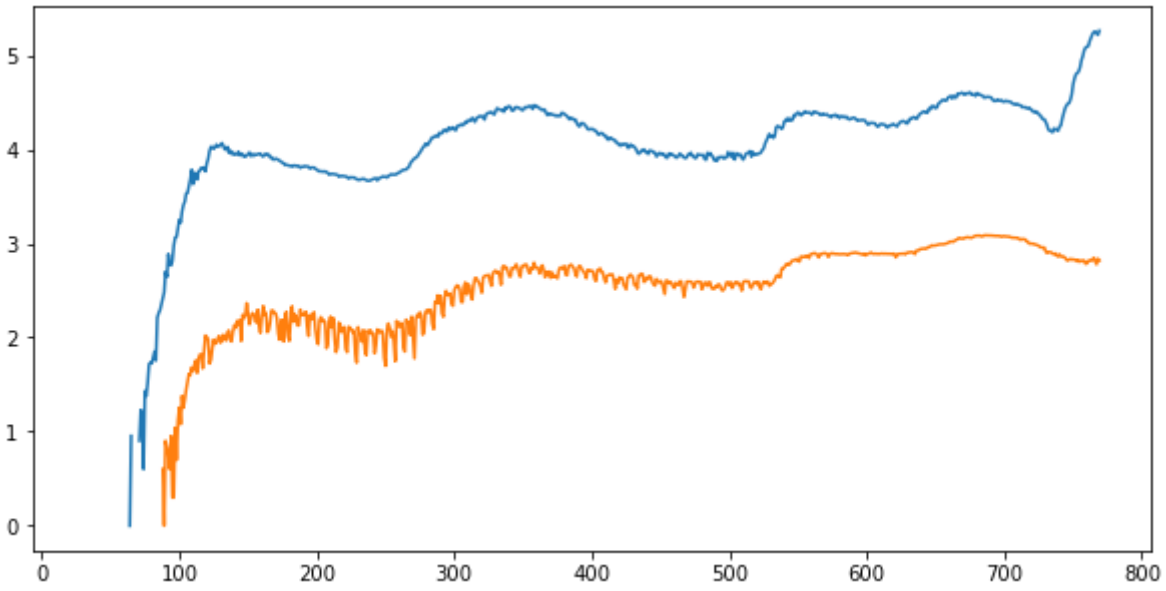

Рисунок 7. График заболеваемости и смертности с нормированными данными методом логорифмизации.

	Afghanistan	Africa	Albania	Algeria	Angola	Argentina	Armenia	Asia	Australia	Austria	...	United Kingdom	United States	Upper middle income	Uruguay	Uzbek
1	0.000616	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.065594	0.000000	0.000000	0.002873	0.0
2	0.000616	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.065594	0.000000	0.000000	0.002873	0.0
3	0.000616	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.065594	0.000000	0.000000	0.002873	0.0
4	0.000616	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.065594	0.000000	0.000000	0.002873	0.0
5	0.000616	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.065594	0.000000	0.000000	0.002873	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
703	0.351852	0.609836	0.908397	0.939086	0.500000	0.547103	1.000000	0.825656	0.732551	1.000000	...	0.803424	0.316266	0.945693	0.930000	0.9
704	0.259259	0.435096	0.834606	0.873096	0.357143	0.418906	0.797561	1.000000	0.753666	0.916829	...	0.837958	0.177356	0.862205	0.346667	1.0
705	1.000000	0.327622	0.437659	0.979695	0.000000	0.801878	0.231707	0.826499	0.841056	0.816705	...	1.000000	1.000000	0.772657	0.593333	1.0
706	0.264706	0.595581	1.000000	1.000000	1.000000	1.000000	0.585366	0.928326	1.000000	0.747484	...	0.891575	0.717877	0.921653	0.790000	0.7
707	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.0

Рисунок 8. Таблица заболеваемости после применения линейного нормирования.

Коэффициент Корреляции:

Корреляция нужна для того, чтобы находить линейную зависимость между разными данными.

Так как мы решили спрогнозировать количество заболевших на несколько дней вперед, то наша идея заключалась в том, чтобы находить страны с наиболее похожим развитием событий. Как находить похожих?

Самый простой способ посмотреть есть ли между ними линейная зависимость. А для того, чтобы найти процентальные значения для каждой страны, нам бы потребовалось заново рассчитывать количество заболевших и сравнивать с количеством населения в стране, а этих данных у нас не было. Так что, было принято решение взять нормированные данные, которые отлично подходили, так как были в одном и том же диапазоне и сравнимы друг с другом.

Формула коэффициента корреляции: $r_{xy} = \frac{\Sigma(x_i - \underline{x}) \times (y_i - \underline{y})}{\sqrt{\Sigma(x_i - \underline{x})^2 \times \Sigma(y_i - \underline{y})^2}}$, где x_i - значение переменной x; y_i - значение переменной y; $\underline{x}$ - среднее арифметическое для переменной x; $\underline{y}$ - среднее арифметическое для переменной y.

Мы попробовали воспользоваться коэффициентом корреляции, предлагаемой нам библиотекой Numpy, но это было безуспешно, так как наши данные не подходили под требования Numpy, чтобы с ними работать, но также прокоррелировать данные могла и библиотека Pandas, причём успешно, так как данные хранились и преобразовывались именно при помощи этой библиотеки.

После проделанной работы у нас была таблица со значениями коррелируемых стран для каждой страны (**Рисунок 9**).

	Afghanistan	Africa	Albania	Algeria	Angola	Argentina	Armenia	Asia	Australia	Austria	...	United Arab Emirates	United Kingdom	United States	U m in
Afghanistan	1.000000	0.292700	-0.197672	0.232758	-0.018412	NaN	-0.069305	0.154058	-0.045869	-0.027776	...	0.214758	-0.054590	-0.113786	0.20
Africa	0.292700	1.000000	0.288296	0.591199	0.406593	NaN	0.016797	0.414463	0.383298	0.199602	...	0.463017	0.656802	0.525381	0.54
Albania	-0.197672	0.288296	1.000000	0.298159	0.199427	NaN	0.307765	0.409428	0.512258	0.536513	...	0.449044	0.480774	0.624450	0.58
Algeria	0.232758	0.591199	0.298159	1.000000	0.133341	NaN	0.256575	0.548941	0.331686	0.469748	...	0.303516	0.342501	0.437461	0.63
Angola	-0.018412	0.406593	0.199427	0.133341	1.000000	NaN	0.032566	0.254611	0.373619	0.104573	...	0.201878	0.569862	0.438423	0.24
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Uzbekistan	0.150998	0.483662	0.258842	0.671357	0.151012	NaN	0.269061	0.662351	0.323680	0.408301	...	0.093811	0.267009	0.319529	0.61
Venezuela	0.218098	0.293873	0.262894	0.333574	0.167093	NaN	0.280972	0.607369	0.183892	0.365346	...	0.256125	0.184782	0.171126	0.56
World	0.033326	0.534554	0.625712	0.573382	0.386376	NaN	0.386243	0.804956	0.805676	0.786313	...	0.483404	0.737204	0.854536	0.91
Zambia	0.418152	0.672904	0.161784	0.193907	0.511515	NaN	-0.202225	0.148956	0.403221	0.053392	...	0.522249	0.586121	0.388500	0.31
Zimbabwe	0.146760	0.581724	-0.019986	0.225531	0.182457	NaN	-0.128359	0.057060	0.099254	0.006171	...	0.072333	0.402839	0.173916	0.12

129 rows × 129 columns

Рисунок 9. Таблиц с коррелированными данными о смертности.

Линейная регрессия и первые прогнозы

Последний вопрос, который так и не остался раскрытым - это вопрос, связанный с предсказанием? Возможно ли это, а если и возможно, то насколько точно?

Так как все данные были подготовлены, мы приступили к созданию полиномиальной регрессии с помощью набора библиотек sklearn, которые в себе поддерживали множество методов, в том числе и регрессию. Для начала, нам надо написать небольшой кусок кода, который будет обучаться по имеющимся данным и строить свой полином, максимально приближенный к ним.

Далее, нам требуется выбрать подходящую степень полинома, так как есть более удачные версии полинома, которые проходят максимально близко к нашему графику, а есть и менее удачные. Выбирать мы будем с помощью RMSE или же среднеквадратичной ошибке, которая позволит нам найти наименьшее отклонение, например, полином второй степени (**Рисунок 10**) имеет RMSE равный 0.08, такое маленькое отклонение связано с тем, что входные данные нормированы между 0 и 1, так что, 0.08 мы можем рассматривать, как 8%. После этого мы проверили все степени вплоть до 15-ой и выяснили, что самое маленькое отклонение имеет

полином 12-ой степени (**Рисунок 11**), а именно 3.7%, но взять мы решили полином 10-ой степени (**Рисунок 12**) с отклонением 4% и предсказывать по нему, так как, при выборе 12 степени возникает переобучение модели (это когда кривая хорошо описывает имеющиеся данные, но не может сделать предсказание), поэтому мы ищем лучшую степень до 10, это прекрасно видно на графиках ниже.

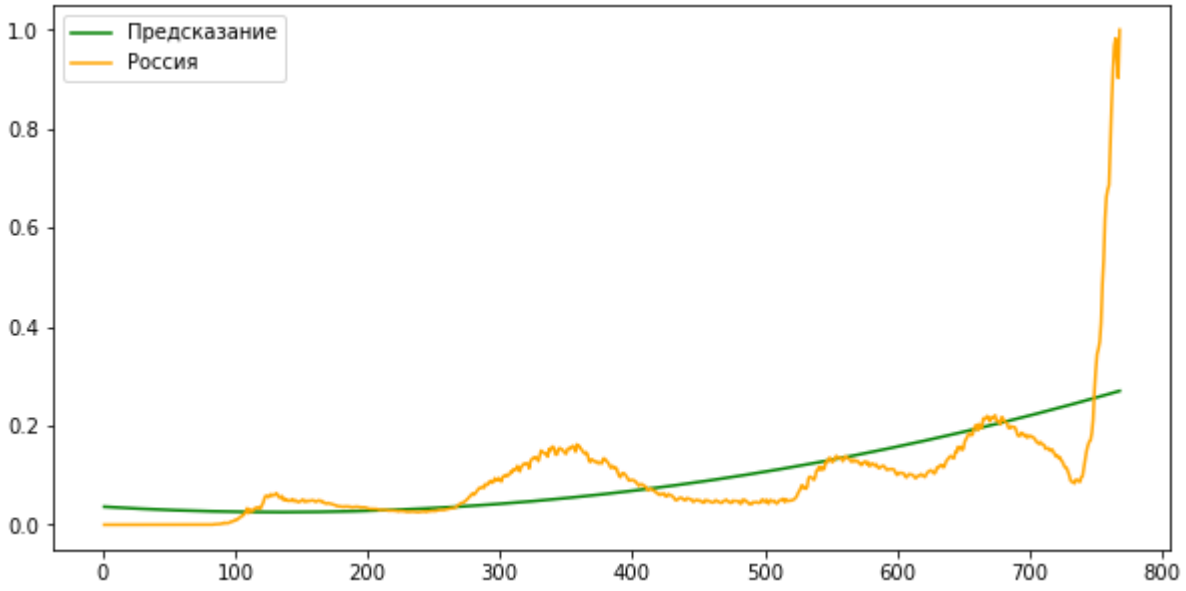

Рисунок 10. Пример полинома второй степени

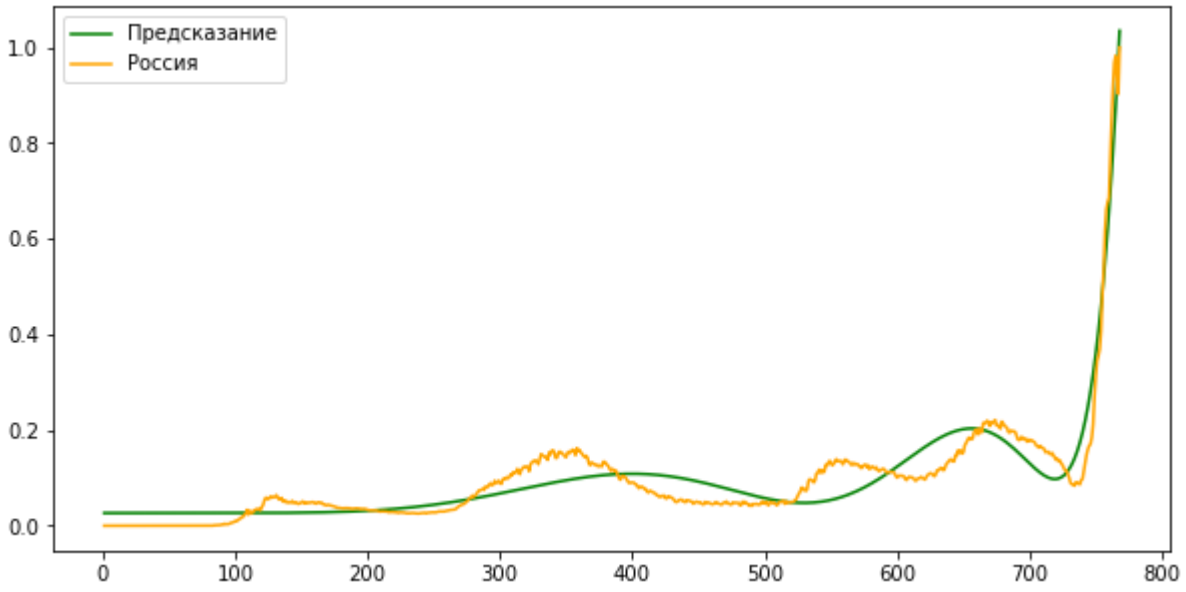

Рисунок 11. График полинома 12-ой степени.

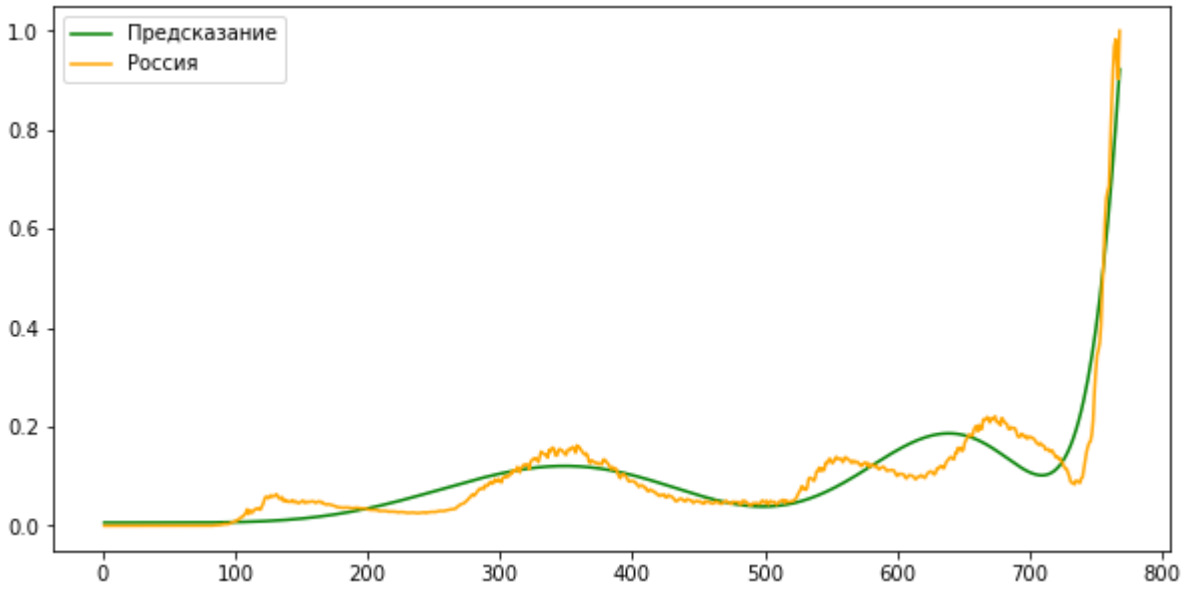

Рисунок 12. График полинома 10-ой степени.

Так как времени оставалось совсем мало, полностью нами задуманное реализовать не получилось, а именно, мы не использовали коррелирующие страны, чтобы уменьшить RMSE и сделать данные более точными.

Анализ временных рядов:

Новый штамм коронавируса принёс в стационарность графика заболеваемости большой дисбаланс, что привело к тому, что обычной регрессии было уже недостаточно для того, чтобы прогнозировать количество заболевших на следующий день, из-за чего было принято решение прогнозировать с помощью тенденции скользящего среднего.

Для начала, мы создали ещё одну таблицу (**Рисунок 22**), в которой считали разницу в приросте заболеваемости между соседними днями. Дальше, используя уже эти данные, была предпринята попытка спрогнозировать разницу, пока что, с помощью регрессии (**Рисунок 23**).

Удовлетворительных результатов это не дало, хоть и было уже лучше, чем прогнозировать количество заболевших.

Так как регрессия удовлетворительных результатов не дала, мы пошли дальше и использовали тенденцию временных рядов для прогноза (**Рисунок 24**), что было уже значительно лучше, чем регрессия. Дальше, чтобы улучшить результат, мы использовали модификацию простой скользящей средней - взвешенная средняя.

Формула тенденции скользящего среднего: $\widehat{y_t} = \frac{1}{k}\sum_{n=0}^{k-1} y_t - n$, где n - это число, от которого зависит, по скольки последним данным мы будем предсказывать. И мы выбрали 7 последних дней, так как средний латентный период - 4 дня, а инкубационный - 3 дня.

Формула взвешенного среднего: $\widehat{y_t} = \frac{1}{k}\sum_{n=0}^{k-1} \omega_n y_t + 1 - n$

	Afghanistan	Africa	Albania	Algeria	Angola	Argentina	Armenia	Asia	Australia	Austria	...	United Arab Emirates	United Kingdom	United States	Upper middle income
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
798	-0.055813	-0.177420	-0.002506	-0.001190	0.000596	-0.002703	-0.020283	0.085274	0.032202	0.340994	...	0.015433	0.027229	0.005771	-0.049463
799	0.060438	0.195826	-0.001670	-0.001190	-0.005164	0.000293	0.016180	-0.115761	-0.026508	0.033116	...	-0.005144	0.073068	0.007759	0.020228
800	-0.088498	-0.192224	0.001253	0.001983	0.000000	-0.003732	0.001367	0.036943	0.019330	-0.002205	...	0.002908	-0.052638	0.002744	-0.034394
801	-0.004934	0.089107	-0.000835	-0.002380	0.010725	-0.018798	-0.003874	0.191855	-0.009026	-0.098276	...	-0.006486	-0.332245	-0.030136	-0.045220
802	0.029911	-0.137138	-0.005638	0.001190	-0.007150	-0.012513	-0.013674	-0.264270	-0.040252	-0.129572	...	-0.007828	0.000000	-0.003420	-0.018093

800 rows × 129 columns

Рисунок 22. Таблица разницы между приростом заболевших между соседними днями.

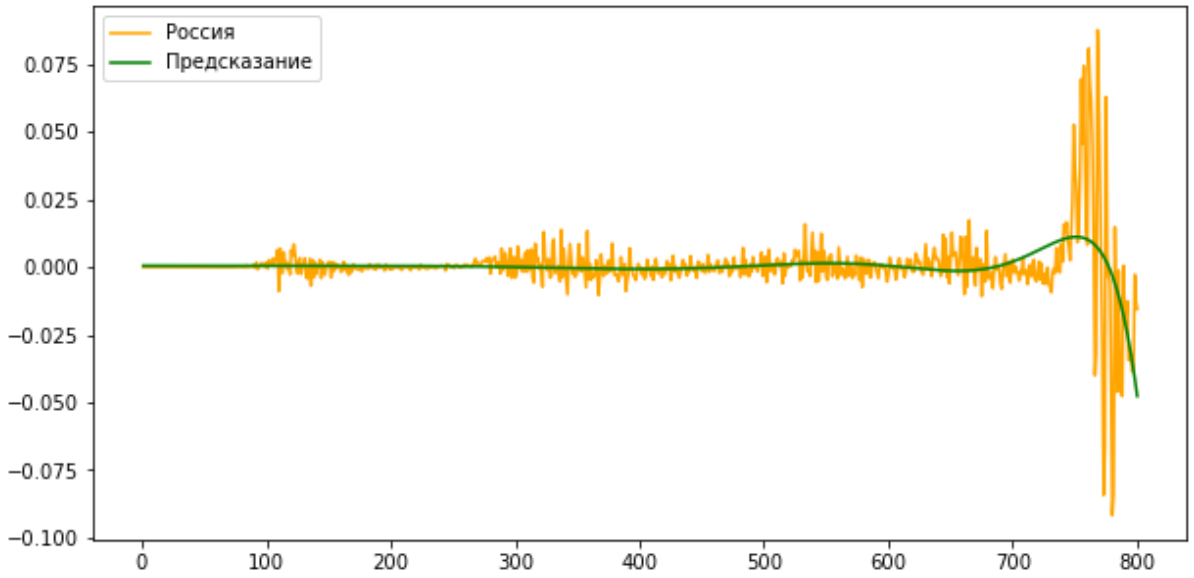

Рисунок 23. График полиномиальной регрессии 13-ой степени.

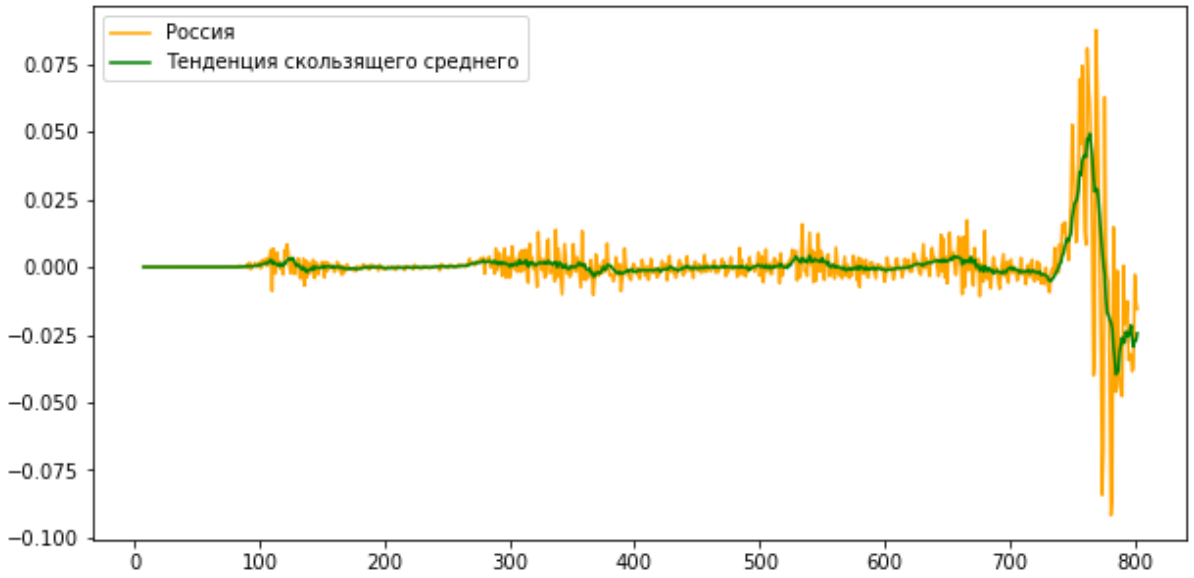

Рисунок 24. График тенденции скользящего среднего.

Статистистические данные о вакцинации:

Так как у нас уже была вся информация, мы решили подвести некоторые интересные статистические данные.

Мы решили узнать, в какой момент в России вакцинировалось 10%, в какой 30% и в какой 40% человек (**Рисунок 13**). Узнать мы это смогли,

благодаря тому же открытому ресурсу Our World In Data[1], благодаря которому мы узнали и о заболеваемости.

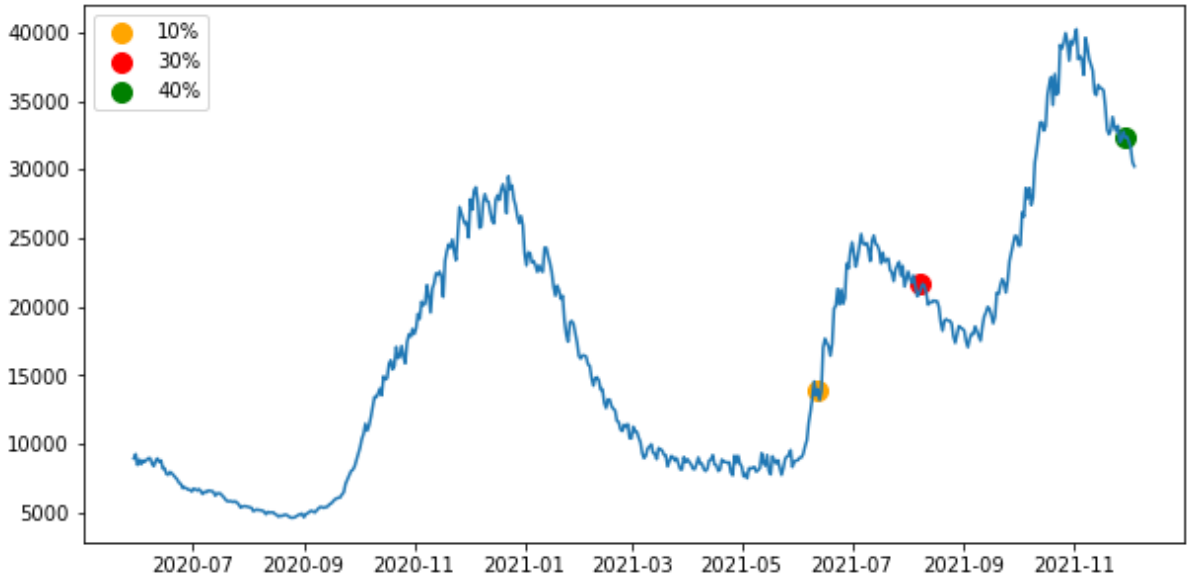

Рисунок 13. График заболеваемости России с точками вакцинации 10-ти, 30-ти и 40-а процентов населения.

А также, мы можем узнать сезонность и частоту заболеваемости по дням недели и месяцам, что мы и сделали. И вот, что мы получили (приведённые данные актуальны на момент 10-го февраля 2022-го года):

```
Заболевших зимой 2020: 277008.0
Заболевших зимой 2021: 46192816.0
Заболевших весной 2020: 8625603.0
Заболевших весной 2021: 54973659.0
Заболевших летом 2020: 22573138.0
Заболевших летом 2021: 50179346.0
Заболевших осенью 2020: 43817100.0
Заболевших осенью 2021: 38154342.0
Всего заболевших: 267829762.0
```

```
Заболевших в понедельник: 53900404.0
Заболевших во вторник: 61287551.0
Заболевших в среду: 65375155.0
Заболевших в четверг: 62900042.0
Заболевших в пятницу: 63325693.0
Заболевших в субботу: 51021337.0
Заболевших в воскресенье: 44048155.0
Всего заболевших: 403258184.0
```

Рисунок 14. Сезонная статистика. Рисунок 15. Статистика по дням

Создание web-ресурса:

Для размещения всех данных в интернете, мы использовали библиотеку Dash, которая позволяет выводить графики, созданные с помощью Plotly на страничку сайта и редактировать её.

Созданные ранее графики, мы выводили на сайт, который изначально находился на локальном компьютере, для того, чтобы было удобней выявлять и исправлять ошибки. Таким образом мы создали первую версию странички (**Рисунок 16**), пока что только с одним графиком. Со временем мы добавили на неё выбор из того, что хочет видеть пользователь - заболеваемость или смертность, от чего со временем отказались, выведя пронормированную заболеваемость и смертность отдельной страны на один график, также отредактировав внешний вид графика, сделав его приятней глазу и понятней для чтения (**Рисунок 17**).

Следующим нашим шагом было добавления на сайт строки ввода с подсказкой из выпадающего списка (**Рисунок 18**) для того, чтобы можно было выбрать любую страну и посмотреть её график заболеваемости и смертности.

Дальше, так как мы хотели найти несколько схожих стран со страной, которую мы выбрали, для будущего обучения регрессии, мы решили вывести кривые заболеваемости стран, с самым высоким коэффициентом корреляции на одном графике для их визуального сравнения (**Рисунок 19**). Рядом с ним расположилась тепловая карта этих стран (**Рисунок 20**), которая наглядно показывает коэффициент корреляции.

Дальше мы вывели все собранные статистические данные на сайт (**Рисунок 21**).

После того, как всё это исправно работало на локальном компьютере, мы стали искать хостинг для размещения нашего сайта в интернете. Им стал Heroku по очень простой причине - он бесплатный и имеет большой функционал.

Для размещения сайта на Heroku требовались некоторые приготовления.

Во-первых, данные изначально хранились на компьютере, где запускался сайт, из-за чего у проекта, который мы загружали на хостинг был вес больше, чем позволял Heroku. Чтобы этого избежать, мы загрузили данные на GitHub[3] и считывали их оттуда. Это замедляет изначальный старт программы, но сильно экономит место.

Во-вторых, был необходим текстовый документ, в котором было бы написано всё, что требуется для запуска нашей программы. Затем, в специальном файле нужно было указать, какую программу следует запускать и то, что она является web-приложением.

После проделанных действий сайт можно было запускать на хостинге. Но мы столкнулись с проблемой - Heroku не мог сам пронормировать все данные из-за вычислительной сложности данного процесса. Так что, нам пришлось создать отдельные файлы, в которых хранились уже заранее пронормированные данные.

После этого, сайт всё равно не включился. По той же причине, что и в прошлый раз, но теперь из-за подсчёта статистики. Поэтому, её тоже пришлось сохранить отдельно.

После всего этого, сайт заработал. Вы можете посетить его по этой ссылке: https://vast-wave-43499.herokuapp.com/, но из-за того, что хостинг бесплатный - первый запуск может быть долгим, это связано с тем, что в случае, если на сайт никто долго не заходил, то Heroku останавливает выполнение нашей программы.

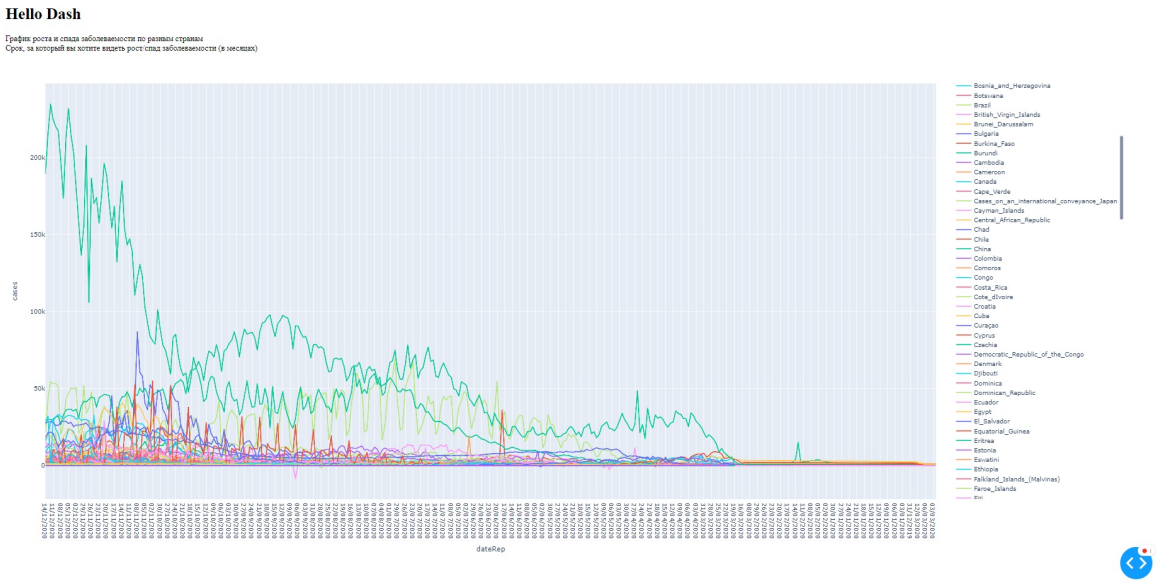

Рисунок 16. Первая версия сайта.

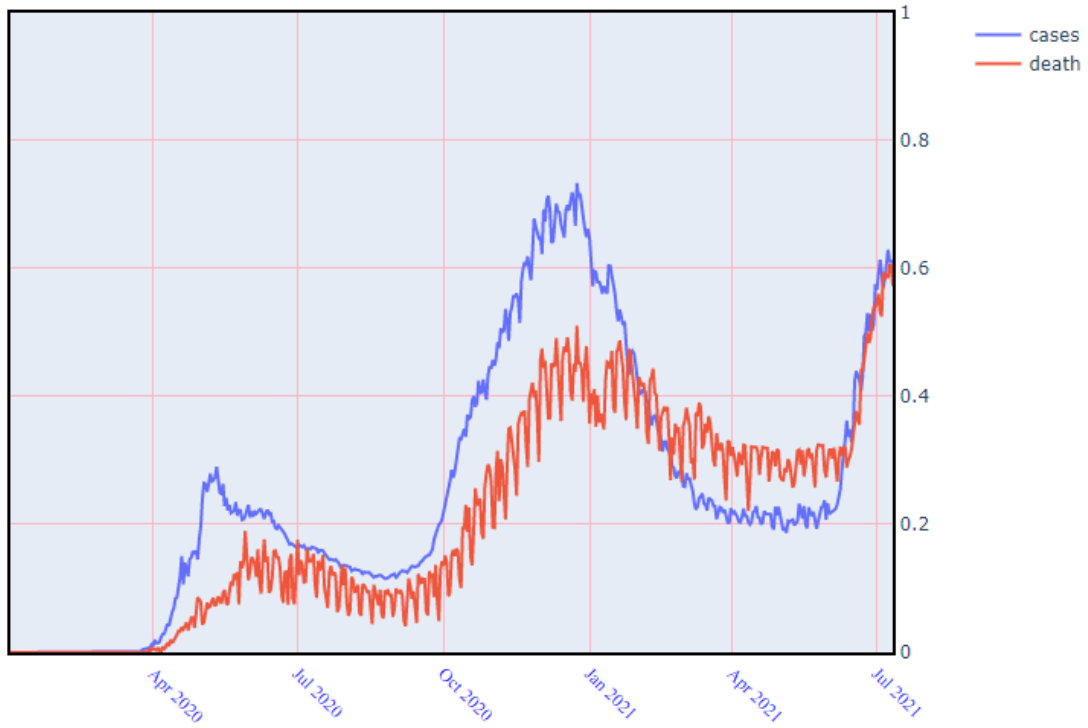

Рисунок 17. График России с пронормированными данными и выбором между заболеваемостью и смертностью.

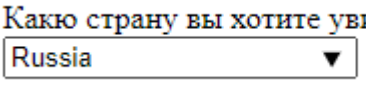

Рисунок 18. Строка ввода.

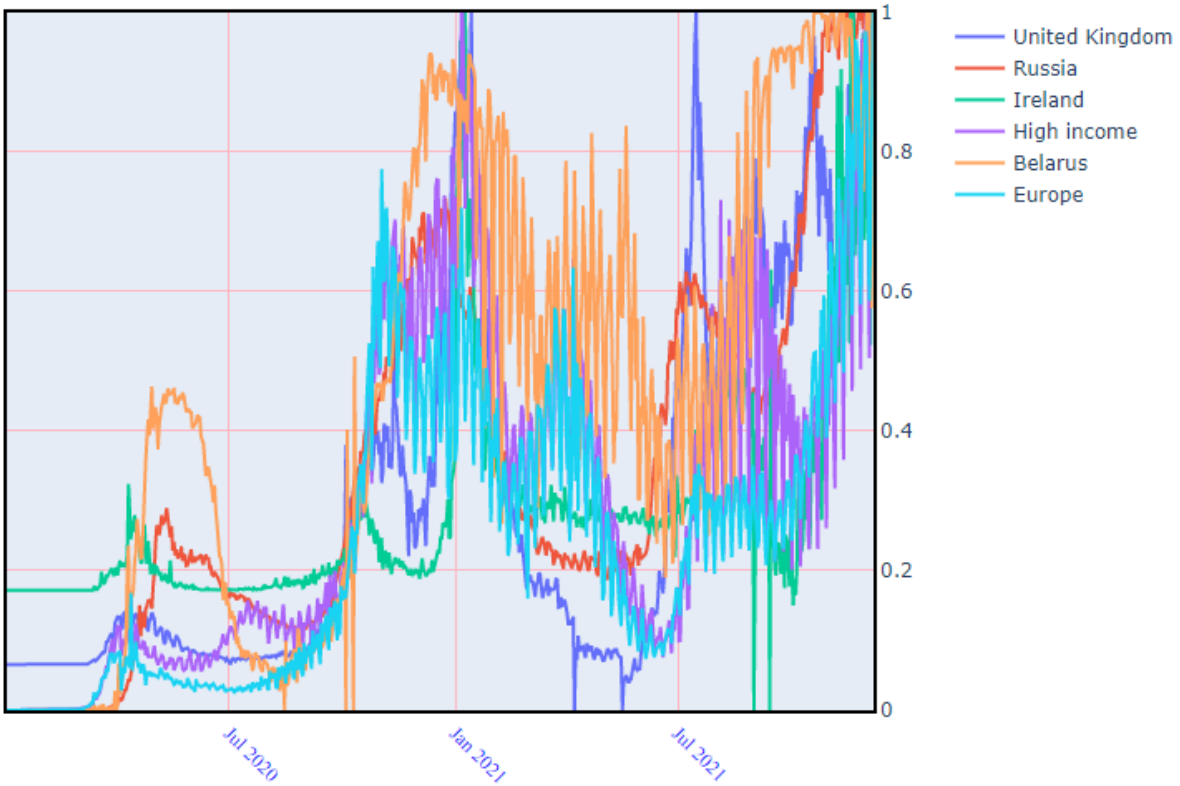

Рисунок 19. Графики стран с высоким коэффициентом корреляции, относительно России.

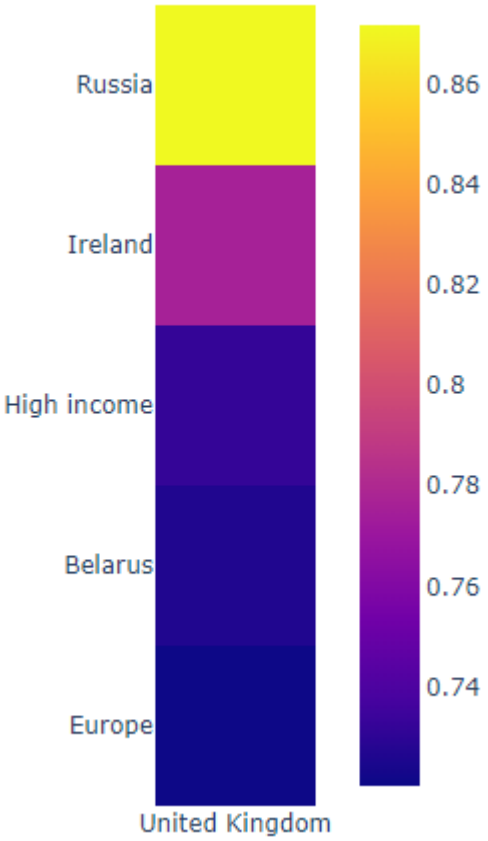

Рисунок 20. Тепловая карта пяти стран с самым большим коэффициентом корреляции, относительно Англии.

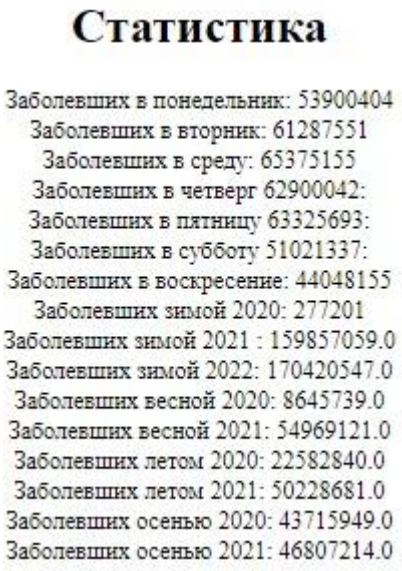

Рисунок 21. Статистические данные.

Итоги:

Итак, подводя итоги нашей работы, можно сказать следующее: прежде всего, гипотеза наша подтвердилась, но предсказать заболеваемость мы смогли не так точно, как хотелось бы, ведь формально, количество заражённых - это случайная величина, зависящая от множества факторов, даже от сезона и дня недели, также, мы можем утверждать, что заболеваемость - это случайный процесс, ведь он распределён во времени и зависит от случайных величин, а прогнозировать случайные процессы с высокой точностью можно только тогда, когда они стационарны. Как всем нам известно, последний штамм коронавируса - омикрон, внёс большие корректировки в стационарность заболеваемости и привёл к огромному отклонению от её среднего значения.

Но кроме этого, был создан web-ресурс для визуализации данных и наглядной оценки, а также, была найдена корреляцию стран между собой и подведена небольшая статистика.

Преимущества нашей работы:

1. Наша работа понятно всем, даже тем людям, которые ни разу не работали с Python или предсказаниями.
2. Наша работа проста в освоении и легко читаема для изучения.
3. У нас есть отдельный сайт, который подводит итоги нашей работы и одновременно позволяет просто предсказать ситуацию с заболеваемостью.

Перспективы:

Мы и в будущем будем продолжать работу над нашим проектом, так как мы всё ещё можем уменьшить RMSE и сделать более точное предсказание с помощью временных рядов. Кроме этого, мы можем точно

утверждать, что наш проект помог нам развиться и дал много информации в сфере работы с данными и их обработке.

Список литературы:

1. https://ourworldindata.org/
2. https://habr.com/ru/all/
3. https://github.com/ZelshaR/Project
4. https://yandex.ru/covid19/stat
5. https://news.google.com/covid19/map?hl=ru&gl=RU&ceid=RU%3Aru&mid=%2Fm%2F06bnz&state=1
6. https://dash.plotly.com/
7. https://pandas.pydata.org/docs/
8. https://neurohive.io/ru/osnovy-data-science/linejnaja-regressija/
9. https://www.youtube.com/

Приложение.

```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import seaborn as sns
import dash
import datetime as dt
from matplotlib.colors import ListedColormap

from dash import dcc
from dash import html
from itertools import chain
from collections import Counter
from dash.dependencies import Input, Output
from sklearn.pipeline import make_pipeline

import plotly.graph_objects as go
from plotly.subplots import make_subplots

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean_squared_error


df = pd.read_csv('https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/owid-covid-data.csv')

dfc = pd.read_csv('https://raw.githubusercontent.com/ZelshaR/Project/main/NoteBooks/CSVs/newdfc.csv')
dfc.index = dfc['Unnamed: 0'].values
del dfc['Unnamed: 0']
dfc['nu'] = 0
dfc['nu'] = dfc.index

dfd = pd.read_csv('https://raw.githubusercontent.com/ZelshaR/Project/main/NoteBooks/CSVs/newdfd.csv')
dfd.index = dfd['Unnamed: 0'].values
del dfd['Unnamed: 0']
dfd['nu'] = 0
dfd['nu'] = dfd.index

dfdsr = pd.read_csv('https://raw.githubusercontent.com/ZelshaR/Project/main/NoteBooks/CSVs/newdfdsr.csv')
dfdsr.index = dfdsr['Unnamed: 0'].values
del dfdsr['Unnamed: 0']


dfcsr = pd.read_csv('https://raw.githubusercontent.com/ZelshaR/Project/main/NoteBooks/CSVs/newdfcsr.csv')
dfcsr.index = dfcsr['Unnamed: 0'].values
del dfcsr['Unnamed: 0']


allcountli = list(dfc.columns)
allcountli.pop(0)
allcount = allcountli
for i in range(len(allcount)):
    allcount[i] = html.Option(value=allcount[i])

the_number_of_cases = dcc.RangeSlider(
    id='value-cases',
    min=1,
    max=len(dfc),
    value=[1, len(dfc)]
)

#это кореляция,её вычисление
lisc = list(dfc.columns)
inp='Afghanistan' #Afghanistan
```

```
inpcol=lisc.index(inp)
c = dfc.corr()
cd=c.iloc[:,[inpcol]]
cd=cd.sort_values(by=inp,ascending=False)
cd=cd[1:6]
sns.heatmap(cd,annot=True) # vmin=0,vmax=1
# plt.show()

# a = 0
# for i in range(5, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin1=53900404
# print('Заболевших в понедельник:', a)
# a = 0
# for i in range(6, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin2= 61287551
# print('Заболевших во вторник:', a)
# a = 0
# for i in range(0, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin3= 65375155
# print('Заболевших в среду:', a)
# a = 0
# for i in range(1, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin4=62900042
# print('Заболевших в четверг:', a)
# a = 0
# for i in range(2, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin5=63325693
# print('Заболевших в пятницу:', a)
# a = 0
# for i in range(3, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin6=51021337
# print('Заболевших в субботу:', a)
# a = 0
# for i in range(4, 687, 7):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casin7=44048155
# print('Заболевших в воскресенье:', a)
# print('Всего заболевших:', df[df['location'] == 'World']['total_cases'].iloc[686])
casinall= 403258184.0
# a = 0
# for i in range(0, 59):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinwin20=277201
# print('Заболевших зимой 2020:', a)
# a = 0
# for i in range(334, 424):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinwin21i=46278237.0
casinwin21d=113578822.0
casinwin22d=170420547.0
# print('Заболевших зимой 2021:', a)
# a = 0
# for i in range(60, 152):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinspr20=8645739.0
# print('Заболевших весной 2020:', a)
# a = 0
# for i in range(425, 514):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinspr21=54969121.0
# print('Заболевших весной 2021:', a)
# a = 0
# for i in range(153, 245):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinsum20=22582840.0
```

```
# print('Заболевших летом 2020:', a)
# a = 0
# for i in range(515, 607):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinsum21=50228681.0
# print('Заболевших летом 2021:', a)
# a = 0
# for i in range(246, 333):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinaut20=43715949.0
# print('Заболевших осенью 2020:', a)
# a = 0
# for i in range(607, 686):
#     a = a + df[df['location'] == 'World']['new_cases'].iloc[i]
casinaut21=46807214.0
# print('Заболевших осенью 2021:', a)
# print('Всего заболевших:', df[df['location'] == 'World']['total_cases'].iloc[686])


# corlist = dfc.columns  # 'это нармирование'
# dfcsr = dfc.copy(deep=True)
# dfdsr = dfd.copy(deep=True)
# # for i in corlist:
# #    for j in range(554):
# #       dfcsr[i].iloc[j] = (dfcsr[i].iloc[j]-min(dfcsr[i]))/(max(dfcsr[i])-min(dfcsr[i]))
# # for i in corlist:
# #    for j in range(554):
# #      dfdsr[i].iloc[j] = (dfdsr[i].iloc[j]-min(dfdsr[i]))/(max(dfdsr[i])-min(dfdsr[i]))
# dfcsr = pd.read_csv("https://raw.githubusercontent.com/ZelshaR/Project/main/NoteBooks/CSVs/dfcsr.csv")
# dfdsr = pd.read_csv("https://raw.githubusercontent.com/ZelshaR/Project/main/NoteBooks/CSVs/dfdsr.csv")
# del dfcsr['Unnamed: 0']
# del dfdsr['Unnamed: 0']
print(len(dfc))
dfc.index = np.unique(df['date'])[1:len(dfc)+1]
dfd.index = np.unique(df['date'])[1:len(dfd)+1]

# 'это превый грфик'
trace1 = go.Scatter(
    x=dfc.index,
    y=dfcsr['Afghanistan'],
    name='cases',
    yaxis='y2'
)
trace2 = go.Scatter(
    x=dfd.index,
    y=dfdsr['Afghanistan'],
    name='death',
    yaxis='y2'
)
fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(trace1)
fig.add_trace(trace2, secondary_y=True)
fig['layout'].update(height=600, width=800, xaxis=dict(
    tickangle=45
))
fig.update_yaxes(range=[0, 1])
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
fig.update_xaxes(tickangle=45, tickfont=dict(family='Rockwell', color='blue', size=12))
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
# fig.show()

# 'это второй грфик'
trace3 = go.Scatter(
    x=dfc.index,
    y=dfcsr['Afghanistan'],
    name='Russia',
    yaxis='y2'
)
trace4 = go.Scatter(
    x=dfc.index,
    y=dfcsr[cd.iloc[0].name],
```

```
    name=cd.iloc[0].name,
    yaxis='y2'
)
trace5 = go.Scatter(
    x=dfc.index,
    y=dfcsr[cd.iloc[1].name],
    name=cd.iloc[1].name,
    yaxis='y2'
)
trace6 = go.Scatter(
    x=dfc.index,
    y=dfcsr[cd.iloc[2].name],
    name=cd.iloc[2].name,
    yaxis='y2'
)
trace7 = go.Scatter(
    x=dfc.index,
    y=dfcsr[cd.iloc[3].name],
    name=cd.iloc[3].name,
    yaxis='y2'
)
trace8 = go.Scatter(
    x=dfc.index,
    y=dfcsr[cd.iloc[4].name],
    name=cd.iloc[4].name,
    yaxis='y2'
)
fig3 = make_subplots(specs=[[{"secondary_y": True}]])
fig3.add_trace(trace3)
fig3.add_trace(trace4)
fig3.add_trace(trace5)
fig3.add_trace(trace6)
fig3.add_trace(trace7)
fig3.add_trace(trace8)
fig3['layout'].update(height=600, width=800, xaxis=dict(
    tickangle=45
))
fig3.update_yaxes(range=[0, 1])
fig3.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
fig3.update_xaxes(tickangle=45, tickfont=dict(family='Rockwell', color='blue', size=12))
fig3.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
# fig3.show()

# это штука делает кореляцию не уродливой
fig2 = px.imshow(cd)
fig2.update_layout(width=500, height=600, margin=dict(l=200, r=200, b=100, t=100))

#дальше смерть
yaxis = dfcsr['Russia'].values
xaxis = [i for i in range(1, len(dfc) + 1)]

xaxis1 = np.reshape(xaxis, (-1, 1))
lr.fit(xaxis1, yaxis)

rmses = []
# enumerate = [i for i in range(9)]
enumerate = [10]

for i in enumerate:
    p = PolynomialFeatures(i, include_bias=True)

    xaxis_poly = p.fit_transform(xaxis1)

    lr.fit(xaxis_poly, yaxis)

    xaxis_ = np.linspace(xaxis1.min(0), xaxis1.max(0), len(dfc)).reshape(len(dfc), 1)
    y_pred_lr = lr.predict(p.transform(xaxis_))
    poly_mse = mean_squared_error(yaxis, y_pred_lr)
    poly_rmse = np.sqrt(poly_mse)
    rmses.append(poly_rmse)
    modelo = make_pipeline(p, lr)
```

```
    # сверху не трогать
    # а снизу можно

    # plt.figure(figsize=(10, 5))
    # plt.plot(xaxis_, y_pred_lr, label='Предсказание', c='green')
    # plt.plot(xaxis, yaxis, label='Действительная кривая', c='orange')

yaxis = dfcsr['Russia'].values
xaxis = [i for i in range(1, len(dfc) + 1)]

xaxis1 = np.reshape(xaxis, (-1, 1))
lr.fit(xaxis1, yaxis)

enumerate = [rmses.index(min(rmses))]
p = PolynomialFeatures(i, include_bias=True)

xaxis_poly = p.fit_transform(xaxis1)

lr.fit(xaxis_poly, yaxis)

xaxis_ = np.linspace(xaxis1.min(0), xaxis1.max(0), len(dfc)).reshape(len(dfc), 1)
y_pred_lr = lr.predict(p.transform(xaxis_))
poly_mse = mean_squared_error(yaxis, y_pred_lr)
poly_rmse = np.sqrt(poly_mse)
rmses.append(poly_rmse)

trace1 = go.Scatter(
    x=[i for i in range(1, len(dfc) + 1)],
    y=list(y_pred_lr),
    name='predict',
    yaxis='y2'
)
trace2 = go.Scatter(
    x=xaxis,
    y=list(yaxis),
    name='true',
    yaxis='y2'
)
fig4 = make_subplots(specs=[[{"secondary_y": True}]])
fig4.add_trace(trace1)
fig4.add_trace(trace2, secondary_y=True)
fig4['layout'].update(height=600, width=800, xaxis=dict(
    tickangle=45
))
fig4.update_yaxes(range=[0, 1])
fig4.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
fig4.update_xaxes(tickangle=45, tickfont=dict(family='Rockwell', color='blue', size=12))
fig4.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')

g = np.reshape(dfcsr.index, (-1, 1))
x = 769
a = 0
for i in range (1,11):
    a = ((x ** (i)) * modelo.steps[1][1].coef_[i]) + a
a = a + modelo.steps[1][1].intercept_
a = a * (max(dfc['Russia']) - min(dfc['Russia'])) + min(dfc['Russia'])

app = dash.Dash(__name__)

server = app.server

app.layout = html.Div([
    html.Div(children=[
        html.H1(children='График роста и спада заболеваемости', style={
            'textAlign': 'center',
        }),

        html.Div(children='Срок, за который вы хотите видеть рост/спад заболеваемости (в месяцах)', style={
            'textAlign': 'center'}),
        html.Div(the_number_of_cases,
                 style={'width': '400px',
```

```
                'margine-bottom': '40px'}),
    html.Div('Какю страну вы хотите увидеть?'),
    dcc.Input(
        id='txtinput',
        type='text',
        pattern=r"^[A-Za-z].*",
        list='browser',
        autoFocus=True,
        value='Russia'
    ),
    dcc.Graph(id='num-of-cas',
              figure=fig),
    html.H1(children='Предсказание', style={
        'textAlign': 'center',
    }),
    dcc.Graph(id='figure4',
              figure=fig4),
]),

# Первый столб
# ------------------------------------------------------------------------------------------------------

html.Div(children=[
    html.H1(children='График стран с которыми наиболее коррелирует', style={
        'textAlign': 'center',
    }),

    dcc.Graph(id='figure3',
              figure=fig3),

    html.H1(children='Статистика', style={'textAlign': 'center', }),
    html.Div(children='Заболевших в понедельник: {}'.format(casin1), style={'textAlign': 'center', }),
    html.Div(children='Заболевших в вторник: {}'.format(casin2), style={'textAlign': 'center', }),
    html.Div(children='Заболевших в среду: {}'.format(casin3), style={'textAlign': 'center', }),
    html.Div(children='Заболевших в четверг {}:'.format(casin4), style={'textAlign': 'center', }),
    html.Div(children='Заболевших в пятницу {}:'.format(casin5), style={'textAlign': 'center', }),
    html.Div(children='Заболевших в субботу {}:'.format(casin6), style={'textAlign': 'center', }),
    html.Div(children='Заболевших в воскресение: {}'.format(casin7), style={'textAlign': 'center', }),
    html.Div(children='Заболевших зимой 2020: {}'.format(casinwin20), style={'textAlign': 'center', }),
    html.Div(children='Заболевших зимой 2021 : {}'.format(casinwin21d + casinwin21i),
             style={'textAlign': 'center', }),
    html.Div(children='Заболевших зимой 2022: {}'.format(casinwin22d), style={'textAlign': 'center', }),
    html.Div(children='Заболевших весной 2020: {}'.format(casinspr20), style={'textAlign': 'center', }),
    html.Div(children='Заболевших весной 2021: {}'.format(casinspr21), style={'textAlign': 'center', }),
    html.Div(children='Заболевших летом 2020: {}'.format(casinsum20), style={'textAlign': 'center', }),
    html.Div(children='Заболевших летом 2021: {}'.format(casinsum21), style={'textAlign': 'center', }),
    html.Div(children='Заболевших осенью 2020: {}'.format(casinaut20), style={'textAlign': 'center', }),
    html.Div(children='Заболевших осенью 2021: {}'.format(casinaut21), style={'textAlign': 'center', }),
    html.H1(children='Предсказание', style={'textAlign': 'center', }),
    html.Div(id="predict", style={'textAlign': 'center', }),
]),

# второй столб
# ------------------------------------------------------------------------------------------------
html.Div(children=[

    html.H1(children='Тепловая карта', style={
        'textAlign': 'center',
    }),

    dcc.Graph(id='cor',
              figure=fig2)

], style={'padding': 10, 'flex': 0,
          "left": "50%",

          'align-items': 'left',
          }),

html.Datalist(id='browser', children=allcount),
#     html.Div([
```

```
    #         dcc.Graph(id='num-of-cas1', figure=fig) #возможно
    #     ]),
], style={'alignitems': 'center', 'display': 'flex', 'flex-direction': 'row'})


@app.callback(
    Output(component_id='num-of-cas', component_property='figure'),
    #     Output(component_id='cor', component_property='figure'),
    Input(component_id='value-cases', component_property='value'),
    Input(component_id='txtinput', component_property='value'),
)
def update_num_of_cas(slider, con):
    w = dfcsr[slider[0]:slider[1]]
    ww = dfdsr[slider[0]:slider[1]]
    www = dfc[slider[0]:slider[1]]
    trace1 = go.Scatter(
        x=www.index,
        y=w[con],
        name='cases',
        yaxis='y2'
    )
    trace2 = go.Scatter(
        x=www.index,
        y=ww[con],
        name='death',
        yaxis='y2'
    )
    fig = make_subplots(specs=[[{"secondary_y": True}]])
    fig.add_trace(trace1)
    fig.add_trace(trace2, secondary_y=True)
    fig['layout'].update(height=600, width=800, xaxis=dict(
        tickangle=45
    ))
    fig.update_yaxes(range=[0, 1])
    fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
    fig.update_xaxes(tickangle=45, tickfont=dict(family='Rockwell', color='blue', size=12))
    fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')

    return fig


@app.callback(
    Output(component_id='cor', component_property='figure'),
    Input(component_id='txtinput', component_property='value'),
)
def update_heatmap(con):
    lisc = list(dfc.columns)
    inp = con  # Afghanistan
    inpcol = lisc.index(inp)
    dfc1=dfc.copy(deep=True)
    del dfc1['nu']
    c = dfc.corr()
    del c['nu']
    cd = c.iloc[:, [inpcol]]
    cd = cd.sort_values(by=inp, ascending=False)
    cd = cd[1:6]
    sns.heatmap(cd, annot=True)  # vmin=0,vmax=1
    fig2 = px.imshow(cd)
    fig2.update_layout(width=500, height=600, margin=dict(l=200, r=200, b=100, t=100))

    return fig2


@app.callback(
    Output(component_id='figure4', component_property='figure'),
    Input(component_id='txtinput', component_property='value'),
)
def update_predict(con):
    yaxis = dfcsr[con].values
    xaxis = [i for i in range(1, len(dfc) + 1)]
```

```
    xaxis1 = np.reshape(xaxis, (-1, 1))
    lr.fit(xaxis1, yaxis)

    rmses = []
    #     enumerate = [i for i in range(11)]
    enumerate = [10]

    for i in enumerate:
        p = PolynomialFeatures(i, include_bias=True)

        xaxis_poly = p.fit_transform(xaxis1)

        lr.fit(xaxis_poly, yaxis)

        xaxis_ = np.linspace(xaxis1.min(0), xaxis1.max(0), len(dfc)).reshape(len(dfc), 1)
        y_pred_lr = lr.predict(p.transform(xaxis_))
        poly_mse = mean_squared_error(yaxis, y_pred_lr)
        poly_rmse = np.sqrt(poly_mse)
        rmses.append(poly_rmse)
        modelo = make_pipeline(p, lr)

    # выше мы определяем лудший вариант
    # ф ниже запихиваем всё в график
    yaxis = dfcsr[con].values
    xaxis = [i for i in range(1, len(dfc) + 1)]

    xaxis1 = np.reshape(xaxis, (-1, 1))
    lr.fit(xaxis1, yaxis)

    enumerate = [rmses.index(min(rmses))]
    p = PolynomialFeatures(i, include_bias=True)

    xaxis_poly = p.fit_transform(xaxis1)

    lr.fit(xaxis_poly, yaxis)

    xaxis_ = np.linspace(xaxis1.min(0), xaxis1.max(0), len(dfc)).reshape(len(dfc), 1)
    y_pred_lr = lr.predict(p.transform(xaxis_))
    poly_mse = mean_squared_error(yaxis, y_pred_lr)
    poly_rmse = np.sqrt(poly_mse)
    rmses.append(poly_rmse)

    trace1 = go.Scatter(
        x=[i for i in range(1, len(dfc) + 1)],
        y=list(y_pred_lr),
        name='predict',
        yaxis='y2'
    )
    trace2 = go.Scatter(
        x=xaxis,
        y=list(yaxis),
        name='true',
        yaxis='y2'
    )
    fig4 = make_subplots(specs=[[{"secondary_y": True}]])
    fig4.add_trace(trace1)
    fig4.add_trace(trace2, secondary_y=True)
    fig4['layout'].update(height=600, width=800, xaxis=dict(
        tickangle=45
    ))
    fig4.update_yaxes(range=[0, 1])
    fig4.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
    fig4.update_xaxes(tickangle=45, tickfont=dict(family='Rockwell', color='blue', size=12))
    fig4.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
    return fig4


@app.callback(
    Output('predict', "children"),
    Input(component_id='txtinput', component_property='value'),
)
```

```
def update_predict(con):
    yaxis = dfcsr[con].values
    xaxis = [i for i in range(1, len(dfc) + 1)]

    xaxis1 = np.reshape(xaxis, (-1, 1))
    lr.fit(xaxis1, yaxis)

    rmses = []
    #     enumerate = [i for i in range(11)]
    enumerate = [10]

    for i in enumerate:
        p = PolynomialFeatures(i, include_bias=True)

        xaxis_poly = p.fit_transform(xaxis1)

        lr.fit(xaxis_poly, yaxis)

        xaxis_ = np.linspace(xaxis1.min(0), xaxis1.max(0), len(dfc)).reshape(len(dfc), 1)
        y_pred_lr = lr.predict(p.transform(xaxis_))
        poly_mse = mean_squared_error(yaxis, y_pred_lr)
        poly_rmse = np.sqrt(poly_mse)
        rmses.append(poly_rmse)
        modelo = make_pipeline(p, lr)

    ans = []
    for j in range(1, 4):
        g = np.reshape(dfcsr.index, (-1, 1))
        x = len(dfc) + j
        a = 0
        for i in range(1, 11):
            a = ((x ** (i)) * modelo.steps[1][1].coef_[i]) + a
        a = a + modelo.steps[1][1].intercept_
        a = a * (max(dfc[con]) - min(dfc[con])) + min(dfc[con])
        ans.append(a)
    return "11 февраля: {} ± {}, 12 февраля: {} ± {}, 13 февраля: {} ± {}".format(round(ans[0]),
                                                          round(ans[0] * round(rmses[-1], 2)),
                                                          round(ans[1]),
                                                          round(ans[1] * round(rmses[-1], 2)),
                                                          round(ans[2]),
                                                          round(ans[2] * round(rmses[-1], 2)))

@app.callback(
    Output(component_id='figure3', component_property='figure'),
    Input(component_id='txtinput', component_property='value'),
)
def update_num_of_cas(con):
    lisc = list(dfc.columns)
    inp = con  # Afghanistan
    inpcol = lisc.index(inp)
    dfc1=dfc.copy(deep=True)
    del dfc1['nu']
    c = dfc1.corr()
    cd = c.iloc[:, [inpcol]]
    cd = cd.sort_values(by=inp, ascending=False)
    cd = cd[1:6]

    trace3 = go.Scatter(
        x=dfc.index,
        y=dfcsr[con],
        name=con,
        yaxis='y2'
    )
    trace4 = go.Scatter(
        x=dfc.index,
        y=dfcsr[cd.iloc[0].name],
        name=cd.iloc[0].name,
        yaxis='y2'
    )
    trace5 = go.Scatter(
        x=dfc.index,
```

```
        y=dfcsr[cd.iloc[1].name],
        name=cd.iloc[1].name,
        yaxis='y2'
    )
    trace6 = go.Scatter(
        x=dfc.index,
        y=dfcsr[cd.iloc[2].name],
        name=cd.iloc[2].name,
        yaxis='y2'
    )
    trace7 = go.Scatter(
        x=dfc.index,
        y=dfcsr[cd.iloc[3].name],
        name=cd.iloc[3].name,
        yaxis='y2'
    )
    trace8 = go.Scatter(
        x=dfc.index,
        y=dfcsr[cd.iloc[4].name],
        name=cd.iloc[4].name,
        yaxis='y2'
    )
    fig3 = make_subplots(specs=[[{"secondary_y": True}]])
    fig3.add_trace(trace3)
    fig3.add_trace(trace4)
    fig3.add_trace(trace5)
    fig3.add_trace(trace6)
    fig3.add_trace(trace7)
    fig3.add_trace(trace8)
    fig3['layout'].update(height=600, width=800, xaxis=dict(
        tickangle=45
    ))
    fig3.update_yaxes(range=[0, 1])
    fig3.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')
    fig3.update_xaxes(tickangle=45, tickfont=dict(family='Rockwell', color='blue', size=12))
    fig3.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True, gridcolor='LightPink')

    return fig3


if __name__ == '__main__':
    app.run_server()
```