**Как скачать Jupyter Notebook: 1 вариант (1 часть)**

Для новичка проще всего начать работу с Jupyter Notebooks, установив дистрибутив [Anaconda](https://anaconda.org/). Anaconda является наиболее широко используемым дистрибутивом Python для работы с данными и поставляется с предустановленными наиболее популярными библиотеками и инструментами. Некоторые из крупнейших библиотек Python, включенных в Anaconda, включают [NumPy](http://www.numpy.org/), [pandas](https://pandas.pydata.org/) и [Matplotlib](https://matplotlib.org/), хотя вы можете взглянуть на полный список с более чем [1000+](https://docs.anaconda.com/anaconda/packages/pkg-docs) пакетов. Это позволит вам приступить к работе, без хлопот управления бесчисленными установками или беспокойства о зависимостях и проблемах установки, связанных с ОС.

Чтобы установить Анаконду, просто:

1. [Загрузите](https://www.anaconda.com/download/) последнюю версию Anaconda для Python 3
2. Установите Anaconda, следуя инструкциям на странице загрузки и/или в исполняемом файле.

Если вы более продвинутый пользователь с уже установленным Python и предпочитаете управлять своими пакетами вручную, вы можете просто использовать pip для установки Jupyter Notebooks: pip3 install jupyter

**Импорт данных:**

import pandas as pd

import numpy as np

from pprint import pprint

import glob

import codecs

import json

import seaborn as sns

import matplotlib.pyplot as plt

pd.set\_option('display.max\_columns', None)

## **Экспорт блокнота**

Простейший способ — сохранить блокнот в формате IPython Notebook (.ipynb), но так как их используют не все, есть и другие варианты:

* Преобразовать блокнот в html-файл;
* Опубликовать его в [gists](https://gist.github.com/), который умеет обрабатывать файлы этого формата ([см. пример](https://gist.github.com/arogozhnikov/930f5663a63014e773d9));
* Сохранить ваш блокнот, например, в dropbox, а затем открыть ссылку в [nbviewer](http://nbviewer.jupyter.org/);
* Блокноты умеет открывать github (есть некоторые ограничения, но в большинстве случаев работает), что очень полезно, так как позволяет хранить историю исследования (если исследование доступно широкой публике).

**Построение графиков**  
Есть несколько вариантов построения графиков:

* matplotlib (фактически, стардарт), включается командой %matplotlib inline;
* %matplotlib notebook — интерактивный режим, но работает очень медленно, так как обработка графика происходит на стороне сервера;
* mpld3 — альтернативная библиотека визуализации (использующая D3) для matplotlib. Она довольно хороша, хотя и неполна.
* bokeh лучше подходит для построения интерактивных графиков;
* plot.ly строит красивые графики, но это будет стоить денег.

## **Magic-команды**

Магические команды (magics) превращают обычный python в *магический python*. Magic-команды — это ключ к могуществу IPython'а.

## **%env** Можно управлять переменными среды для вашего блокнота без перезапуска Jupyter-сервера. Некоторые библиотеки (такие, как theano) используют переменные среды, чтобы контролировать поведение, и %env — самый удобный способ.

#%env - without arguments lists environmental variables

%env OMP\_NUM\_THREADS=4

env: OMP\_NUM\_THREADS=4

## **Выполнение shell-команд** В Notebook можно вызвать любую shell-команду. Это особенно удобно для управления виртуальной средой.

!pip install numpy

!pip list | grep Theano

Requirement already satisfied (use --upgrade to upgrade): numpy **in** /Users/axelr/.venvs/rep/lib/python2.7/site-packages

Theano (0.8.2)

## **Подавление вывода последней строки** Иногда вывод не нужен, и в этом случае можно или использовать команду pass с новой строки, или поставить точку запятой в конце строки:

%matplotlib inline

**from** matplotlib **import** pyplot **as** plt

**import** numpy

# if you don't put semicolon at the end, you'll have output of function printed

plt.hist(numpy.linspace(0, 1, 1000)\*\*1.5);

## **Просмотр исходников функций/классов/чего угодно с помощью вопросительного знака**

**from** sklearn.cross\_validation **import** train\_test\_split

# show the sources of train\_test\_split function in the pop-up window

train\_test\_split??

# you can use ? to get details about magics, for instance:

%pycat?

вызовет следующее всплывающее окно:

Show a syntax-highlighted file through a pager.

This magic is similar to the cat utility, but it will assume the file

to be Python source and will show it with syntax highlighting.

This magic command can either take a local filename, an url,

an history range (see %history) or a macro as argument ::

%pycat myscript.py

%pycat 7-27

%pycat myMacro

%pycat http://www.example.com/myscript.py

## **Используйте %run для выполнения кода на Python** %run может выполнить код на языке Python из файлов с расширением .py — это поведение хорошо задокументировано. Но эта команда может выполнять и другие блокноты из Jupyter! Иногда это очень полезно. Обратите внимание, что %run — это не то же, что импорт python-модуля.

# this will execute all the code cells from different notebooks

%run ./2015-09-29-NumpyTipsAndTricks1.ipynb

[49 34 49 41 59 45 30 33 34 57]

[172 177 209 197 171 176 209 208 166 151]

[30 33 34 34 41 45 49 49 57 59]

[209 208 177 166 197 176 172 209 151 171]

[1 0 4 8 6 5 2 9 7 3]

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['b' 'a' 'e' 'i' 'g' 'f' 'c' 'j' 'h' 'd']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

[1 0 6 9 2 5 4 8 3 7]

[1 0 6 9 2 5 4 8 3 7]

[ 0.93551212 0.75079687 0.87495146 0.3344709 0.99628591 0.34355057

0.90019059 0.88272132 0.67272068 0.24679158]

[8 4 5 1 9 2 7 6 3 0]

[-5 -4 -3 -2 -1 0 1 2 3 4]

[0 0 0 0 0 0 1 2 3 4]

['eh' 'cl' 'ah' ..., 'ab' 'bm' 'ab']

['ab' 'ac' 'ad' 'ae' 'af' 'ag' 'ah' 'ai' 'aj' 'ak' 'al' 'am' 'an' 'bc' 'bd'

'be' 'bf' 'bg' 'bh' 'bi' 'bj' 'bk' 'bl' 'bm' 'bn' 'cd' 'ce' 'cf' 'cg' 'ch'

'ci' 'cj' 'ck' 'cl' 'cm' 'cn' 'de' 'df' 'dg' 'dh' 'di' 'dj' 'dk' 'dl' 'dm'

'dn' 'ef' 'eg' 'eh' 'ei' 'ej' 'ek' 'el' 'em' 'en' 'fg' 'fh' 'fi' 'fj' 'fk'

'fl' 'fm' 'fn' 'gh' 'gi' 'gj' 'gk' 'gl' 'gm' 'gn' 'hi' 'hj' 'hk' 'hl' 'hm'

'hn' 'ij' 'ik' 'il' 'im' 'in' 'jk' 'jl' 'jm' 'jn' 'kl' 'km' 'kn' 'lm' 'ln'

'mn']

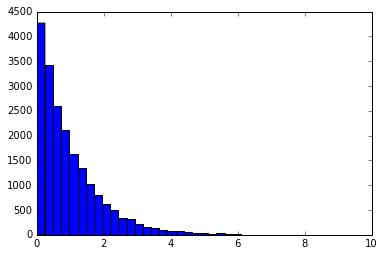
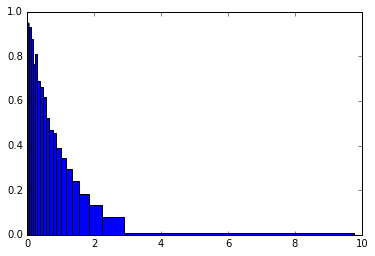
[48 33 6 ..., 0 23 0]

['eh' 'cl' 'ah' ..., 'ab' 'bm' 'ab']

['eh' 'cl' 'ah' ..., 'ab' 'bm' 'ab']

['bf' 'cl' 'dn' ..., 'dm' 'cn' 'dj']

['bf' 'cl' 'dn' ..., 'dm' 'cn' 'dj']

[ 2.29711325 1.82679746 2.65173344 ..., 2.15286813 2.308737 2.15286813]

1000 loops, best of 3: 1.09 ms per loop

The slowest run took 8.44 times longer than the fastest. This could mean that an intermediate result is being cached.

10000 loops, best of 3: 21.5 µs per loop

0.416

0.416

## **%load** Загрузит код напрямую в ячейку. Можно выбрать файл локально или из сети. Если раскомментировать и выполнить код ниже, содержание ячейки заменится на содержание файла.

# %load http://matplotlib.org/mpl\_examples/pylab\_examples/contour\_demo.py

## **%store — ленивая передача данных между блокнотами**

data = 'this is the string I want to pass to different notebook'

%store data

**del** data # deleted variable

Stored 'data' (str)

# in second notebook I will use:

%store -r data

**print** data

this is the string I want to pass to different notebook

## **%who для анализа переменных глобального пространства имен**

# pring names of string variables

%who str

data

## **Тайминг** Если вы хотите замерить время выполнения программы или найти узкое место в коде, на помощь придет IPython.

%%time

**import** time

time.sleep(2) # sleep for two seconds

CPU times: user 1.23 ms, sys: 4.82 ms, total: 6.05 ms

Wall time: 2 s

# measure small code snippets with timeit !

**import** numpy

%timeit numpy.random.normal(size=100)

The slowest run took 13.85 times longer than the fastest. This could mean that an intermediate result is being cached.

100000 loops, best of 3: 6.35 µs per loop

%%writefile pythoncode.py

**import** numpy

**def** **append\_if\_not\_exists**(arr, x):

**if** x **not** **in** arr:

arr.append(x)

**def** **some\_useless\_slow\_function**():

arr = list()

**for** i **in** range(10000):

x = numpy.random.randint(0, 10000)

append\_if\_not\_exists(arr, x)

Overwriting pythoncode.py

# shows highlighted source of the newly-created file

%pycat pythoncode.py

**from** pythoncode **import** some\_useless\_slow\_function, append\_if\_not\_exists

## **Профилирование: %prun, %lprun, %mprun**

# shows how much time program spent in each function

%prun some\_useless\_slow\_function()

Пример вывода:

26338 **function** calls **in** 0.713 seconds

Ordered by: internal time

ncalls tottime percall cumtime percall filename:lineno(**function**)

10000 0.684 0.000 0.685 0.000 pythoncode.py:3(append\_if\_not\_exists)

10000 0.014 0.000 0.014 0.000 {method 'randint' of 'mtrand.RandomState' objects}

1 0.011 0.011 0.713 0.713 pythoncode.py:7(some\_useless\_slow\_function)

1 0.003 0.003 0.003 0.003 {range}

6334 0.001 0.000 0.001 0.000 {method 'append' of 'list' objects}

1 0.000 0.000 0.713 0.713 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

%load\_ext memory\_profiler

# tracking memory consumption (show in the pop-up)

%mprun -f append\_if\_not\_exists some\_useless\_slow\_function()

Пример вывода:

Line # Mem usage Increment Line Contents

================================================

3 20.6 MiB 0.0 MiB def append\_if\_not\_exists(arr, x):

4 20.6 MiB 0.0 MiB **if** x not **in** arr:

5 20.6 MiB 0.0 MiB arr.append(x)

**%lprun** позволяет профилировать с точностью до строк кода, но, кажется, в последнем релизе Python он не работает, так что в этот раз обойдемся без магии:

**import** line\_profiler

lp = line\_profiler.LineProfiler()

lp.add\_function(some\_useless\_slow\_function)

lp.runctx('some\_useless\_slow\_function()', locals=locals(), globals=globals())

lp.print\_stats()

Timer unit: 1e-06 s

Total time: 1.27826 s

File: pythoncode.py

Function: some\_useless\_slow\_function at line 7

Line # Hits Time Per Hit % Time Line Contents

==============================================================

7 def some\_useless\_slow\_function():

8 1 5 5.0 0.0 arr = list()

9 10001 17838 1.8 1.4 **for** i **in** range(10000):

10 10000 38254 3.8 3.0 x = numpy.random.randint(0, 10000)

11 10000 1222162 122.2 95.6 append\_if\_not\_exists(arr, x)

## **Дебаг с помощью %debug**

У Jupyter есть собственный интерфейс для [ipdb](https://docs.python.org/2/library/pdb.html), что позволяет зайти внутрь функции и посмотреть, что в ней происходит.  
  
Это не PyCharm — потребуется время, чтобы освоить, но при необходимости дебага на сервере это может быть единственным вариантом (кроме pdb через терминал).

#%%debug filename:line\_number\_for\_breakpoint

# Here some code that fails. This will activate interactive context for debugging

Немного более простой способ — команда %pdb, которая активирует дебаггер, когда выбрасывается исключение:

# %pdb

# def pick\_and\_take():

# picked = numpy.random.randint(0, 1000)

# raise NotImplementedError()

# pick\_and\_take()

## **Запись формул в LateX**

Маркдаун ячейки могут отрисовывать формулы LateX с помощью MathJax.  
  
Маркдаун — важная часть блокнотов, так что не забывайте использовать его выразительные возможности!

## **Использование разных языков внутри одного блокнота**

Если вы соскучились по другим языкам программирования, можете использовать их в Jupyter Notebook:

* %%python2
* %%python3
* %%ruby
* %%perl
* %%bash
* %%R,

но, разумеется, среда должна быть настроена соответствующим образом.

%%ruby

puts 'Hi, this is ruby.'

Hi, **this** **is** ruby.

%%bash

echo 'Hi, this is bash.'

Hi, **this** **is** bash.

## **Анализ Big Data**

Существует несколько решений, чтобы запрашивать/обрабатывать большие объемы данных:

* [ipyparallel (бывший ipython cluster)](https://github.com/ipython/ipyparallel) — хороший инструмент для простых операций map-reduce на Python. Мы используем его в [rep](https://github.com/yandex/rep) для обучения большого количества моделей машинного обучения параллельно.
* [pyspark](http://www.cloudera.com/documentation/enterprise/5-5-x/topics/spark_ipython.html)
* spark-sql magic [%%sql](https://github.com/jupyter-incubator/sparkmagic)

## **Ваши коллеги могут экспериментировать с вашим кодом, ничего не устанавливая**

Такие сервисы, как [mybinder](http://mybinder.org/), предоставляют доступ к Jupiter Notebook со всеми установленными библиотеками, так что пользователь может с полчаса поиграться с вашим кодом, имея под рукой только браузер.  
  
Вы также можете установить вашу собственную системы с помощью [jupyterhub](https://github.com/jupyterhub/jupyterhub), что очень удобно, если вы проводите мини-курс или мастер-класс и вам некогда думать о машинах для студентов.

## **Написание функций на других языках**

Иногда скорости NumPy бывает недостаточно, и мне необходимо написать немного быстрого кода. В принципе, можно собрать нужные функции в динамические библиотеки, а затем написать обертку на Python…  
  
Но гораздо лучше, когда скучная часть работы сделана за нас, правда?  
  
Ведь можно написать нужные функции на Cython или Fortran и использовать их напрямую из кода на Python.  
  
Для начала нужно установить модули

!pip install cython fortran-magic

%load\_ext Cython

%%cython

def **myltiply\_by\_2**(**float** x):

**return** 2.0 \* x

myltiply\_by\_2(23.)

46.0

Лично я предпочитаю Fortran, на котором, я считаю, удобно писать функции для обработки большого объема численных данных. Подробнее о его использовании можно почитать [здесь](http://arogozhnikov.github.io/2015/11/29/using-fortran-from-python.html).

%load\_ext fortranmagic

/Users/axelr/.venvs/rep/lib/python2.7/site-packages/IPython/utils/path.py:265: UserWarning: get\_ipython\_cache\_dir has moved to the IPython.paths module

warn("get\_ipython\_cache\_dir has moved to the IPython.paths module")

%%fortran

subroutine compute\_fortran(x, y, z)

real, intent(in) :: x(:), y(:)

real, intent(out) :: z(size(x, 1))

z = sin(x + y)

**end** subroutine compute\_fortran

compute\_fortran([1, 2, 3], [4, 5, 6])

array([-0.95892429, 0.65698659, 0.41211849], dtype=float32)

Должен заметить, что есть и другие способы ускорить ваш код на Python. Примеры можно найти в [моем блокноте](http://arogozhnikov.github.io/2015/09/08/SpeedBenchmarks.html).

## **Множественный курсор**

С недавнего времени Jupyter поддерживает множественный курсор, такой, как в Sublime или IntelliJ!  
  
  
Источник: [swanintelligence.com/multi-cursor-in-jupyter.html](http://swanintelligence.com/multi-cursor-in-jupyter.html)

## [**Расширения Jupyter-contrib**](https://github.com/ipython-contrib/jupyter_contrib_nbextensions)

устанавливаются с помощью

!pip install https://github.com/ipython-contrib/jupyter\_contrib\_nbextensions/tarball/master

!pip install jupyter\_nbextensions\_configurator

!jupyter contrib nbextension install --user

!jupyter nbextensions\_configurator enable --user

Это целое семейство различных расширений, включая, например, jupyter spell-checker и code-formatter, которых по умолчанию в Jupyter нет.

## [**RISE**](https://github.com/damianavila/RISE)**: презентации в Notebook**

Расширение, написанное Damian Avila, позволяет демонстрировать блокноты как презентации. Пример такой презентации: [bollwyvl.github.io/live\_reveal/#/7](http://bollwyvl.github.io/live_reveal/#/7)  
  
[Это может пригодиться, если вы обучаете использованию какой-либо библиотеки.](#графики)

ВАРИАНТ 2 ↓

# Отчёт

2.1 Разбиение набора данных - набор данныхз разбит на обучаюшую и тестовую выборки

2.3 Классификация - выбраны 3 алгоритма классификации и метрики для их тестирования

2.4 Обучение - произведена классификация по тяжести ДТП

2.5 Feature Engineering - произведено обечение ещё раз на преобразованных данных

# Сохранение данных

df.to\_csv('result\_data.csv', encoding='utf-8-sig', index=False)

# 2.4 Обучение

#Импорт моделей

from sklearn.metrics import classification\_report

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import GaussianNB

#Обучение

neigh = KNeighborsClassifier(n\_neighbors=3)

neigh.fit(X\_train, y\_train)

preds=neigh.predict(X\_test)

print(classification\_report(preds, y\_test))

#Обучение

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

rfc\_preds=rfc.predict(X\_test)

print(classification\_report(rfc\_preds, y\_test))

#Обучение

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

gnb\_preds=gnb.predict(X\_test)

print(classification\_report(gnb\_preds, y\_test))

Вывод

Наиболее оптимальной моделью будет RandomForestClassifier c accuracy f1-score = 0.68 и macro avg f1-score = 0.71, поскольку по сравнению с другими он показал наилучший результат.

2.5 Feature Engineering

Преобразуем набор данных путём генерации новых данных с целью повышения точности классификатора и использование StandardScaler

#Генерация данных

result = pd.merge(df, df.groupby(['features.properties.vehicles.brand']).size().sort\_values().to\_frame(), on='features.properties.vehicles.brand')

result.rename(columns={0: 'brand\_count'}, inplace=True)

df = result

from sklearn.preprocessing import StandardScaler

#Преобразщование с помощью StandardScaler

scaler = StandardScaler()

X=df[['features.properties.dead\_count', 'features.properties.injured\_count', 'features.properties.participants\_count', 'brand\_count']]

y=df['features.properties.severity']

#Получение выборок

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42, stratify=y)

#Обучение

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

rfc\_preds=rfc.predict(X\_test)

print(classification\_report(rfc\_preds, y\_test))

Выводы по Feature Engineering

Из резёльтатов выше, преобразование данных для Feature Engineering не привёло к улучшению модели

# 2.3 Классификация

Рассмотрим три модели классификации

KNeighborsClassifier

Классификация на основе соседей - это тип обучения на основе экземпляров или необобщающего обучения: он не пытается построить общую внутреннюю модель, а просто сохраняет экземпляры обучающих данных. Классификация вычисляется простым большинством голосов ближайших соседей каждой точки: точке запроса назначается класс данных, который имеет наибольшее количество представителей среди ближайших соседей точки.

RandomForestClassifier

Случайный лес — это метаоценка, которая соответствует ряду классификаторов дерева решений для различных подвыборок набора данных и использует усреднение для повышения точности прогнозирования и контроля переобучения. Размер подвыборки управляется параметром max\_samples, если bootstrap=True (по умолчанию), в противном случае для построения каждого дерева используется весь набор данных

GaussianNB

Наи́вный ба́йесовский классифика́тор — простой вероятностный классификатор, основанный на применении теоремы Байеса со строгими (наивными) предположениями о независимости. В зависимости от точной природы вероятностной модели, наивные байесовские классификаторы могут обучаться очень эффективно

Матрикики

Рассмотрим две метрикики для оценивания модели классификации

accuracy f1-score

Это гармоническое среднее значений точности и полноты. Возьмём её, потому что она дает лучшую оценку неправильно классифицированных случаев

macro avg f1-score

macro avg f1-score пожалуй, самый простой из многочисленных методов усреднения. Макроусредненная оценка F1 (или макрооценка F1) вычисляется путем взятия среднего арифметического (также известного как невзвешенное среднее) всех оценок F1 для каждого класса. Этот метод будет взят, поскольку он обрабатывает все классы одинаково, независимо от их значений поддержки

# 2.1 Разбиение набора данных

import pandas as pd

pd.set\_option('display.max\_columns', None)

# Загрузка данных

df=pd.read\_csv('result\_data.csv')

df.head()

Разобъём набор данных таким образом, как это рекомендовано согласно документации Sklearn. А именно 30 на 70. Как представленно в описании, такая выборка является оптимальной, поскольку абсолютное большинство данных должно находится при обучении модели, чтобы получить наиболее оптимизированную модель со стороны её точности

**Стратификация**

При разделении стратифицируем данные, чтобы получить одинаковую в процентом соотношении выборку, чтобы не было перевеса на какой-то один класс и такая ситуация не повлияла на некорректное обучение модели

X=df[['features.properties.dead\_count', 'features.properties.injured\_count', 'features.properties.participants\_count']]

y=df['features.properties.severity']

#Получение выборок

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42, stratify=y)

# Отчёт

1.1 Парсинг данных - Данные загруженны из папки data

1.2 Предобработка данных и выделение значимых атрибутов - Данные предобработаны и выделенны наиболее значимые атрибуты

1.3 Описание структуры набора данных - для каждого атибута представленно описание

1.4 Формирование дополнительных атрибутов - дополнительный индекс на основе данных сформированн

# 1.4 Формирование дополнительных атрибутов

Формиование индекса будет производиться на основе количества ДТП, частоты и тяжести.

#Формирование первичных полей

df['Hazard\_level'] = None

count\_places\_max = df['count'].max()

injured\_max = df['features.properties.injured\_count'].max()

dead\_max = df['features.properties.dead\_count'].max()

#Вычисление индекса

for i in range(len(df)):

    if df['features.properties.dead\_count'][i] > 0:

        df['Hazard\_level'][i] = (df['features.properties.injured\_count'][i]+df['count'][i])/((injured\_max+count\_places\_max)/2)/4

    else:

        df['Hazard\_level'][i] = (df['features.properties.dead\_count'][i]\*100/dead\_max)/100/2+0.5

df.head()

df.to\_csv('result\_data.csv', encoding='utf-8-sig', index=False)

# 1.3 Описание структуры набора данных

"id": 384094, # идентификатор

"tags": ["Дорожно-транспортные происшествия"], # показатели с официального сайта ГИБДД

"light": "Светлое время суток", # время суток

"point": {"lat": 50.6039, "long": 36.5578}, # координаты

"nearby": [ "Нерегулируемый перекрёсток неравнозначных улиц (дорог)", "Жилые дома индивидуальной застройки"], # координаты

"region": "Белгород", # город/район

"address": "г Белгород, ул Сумская, 30", # адрес

"weather": ["Ясно"], # погода

"category": "Столкновение", # тип ДТП

"datetime": "2017-08-05 13:06:00", # дата и время

"severity": "Легкий", # тяжесть ДТП/вред здоровью

"vehicles": [ # участники – транспортные средства

"year": 2010, # год производства транспортного средства

"brand": "ВАЗ", # марка транспортного средства

"color": "Иные цвета", # цвет транспортного средства

"model": "Priora", # модель транспортного средства

"category": "С-класс (малый средний, компактный) до 4,3 м", # категория транспортного средства

"participants": [ # участники внутри транспортных средств

"role": "Водитель", # роль участника

"gender": "Женский", # пол участника

"violations": [], # нарушения правил участником

"health\_status": "Раненый, находящийся...", # состояние здоровья участника

"years\_of\_driving\_experience": 11 # стаж вождения участника (только у водителей)

"dead\_count": 0, # кол-во погибших в ДТП

"participants": [], # участники без транспортных средств (описание, как у участников внутри транспортных средств)

"injured\_count": 2, # кол-во раненых в ДТП

"parent\_region": "Белгородская область", # регион

"road\_conditions": ["Сухое"], # состояние дорожного покрытия

"participants\_count": 3, # кол-во участников ДТП

"participant\_categories": ["Все участники", "Дети"] # категории участников

df.isna().sum()

# 1.2 Предобработка данных и выделение значимых атрибутов

Все данных выгруженные и json представленны. Рассмотрим некоторые статистику и размерность

#Заполнение недостаюших данных

df=df.fillna(0)

df.shap

df.info()

Здесь укузано количество пустых значений после предобработки

df.isna().sum()

**Предобработка перечеслений в наборе данных**

**#Функция explode разворачивает списки в строки**

df = df.explode('violations\_x')

df = df.explode('features.properties.tags')

df = df.explode('features.properties.nearby')

df = df.explode('features.properties.weather')

df = df.explode('features.properties.road\_conditions')

df = df.explode('features.properties.participant\_categories')

df

**#Удаление дубликатов**

df=df.drop\_duplicates(subset=['features.properties.id'])

df=df.fillna(0)

df=df[df['features.properties.address']!=0]

df.reset\_index(drop=True, inplace=True)

**#Вычисление частоты и количества ДТП**

result = pd.merge(df, df.groupby(['features.properties.address']).size().sort\_values(ascending=False).to\_frame(), on="features.properties.address")

result.rename(columns={0: 'count'},inplace=True)

df = result

**Определение наиболее важных атрибутов**

Чтобы найти наиболее значимые атрибуты, построим корреляцию Пирсона на тепловой карте

**#Фомирование корреляции Пирсона**

corr=df.drop(['features.properties.id'], axis=1).corr()

plt.figure(figsize=(16, 16))

heatmap = sns.heatmap(corr, vmin=-1, vmax=1, annot=True)

heatmap.set\_title('Correlation Heatmap', fontdict={'fontsize':20}, pad=20)

Выше видно наиболее значимые атрибуты, а это: `features.properties.dead\_count`, `features.properties.injured\_count`, `features.properties.participants\_count`

# 1.1 Парсинг данных -----------------------------------------------------------------------------

**#Чтение каждого файла**

read\_files = glob.glob("data/\*.geojson")

output\_list = []

for f in read\_files:

    with open(f, "rb") as infile:

        output\_list.append(json.load(infile))

with open("merged\_file.json", "w") as outfile:

    json.dump(output\_list, outfile)

**#Объединение файлов**

with codecs.open('merged\_file.json', 'r', 'utf-8') as json\_file:

    data = json.load(json\_file)

df = pd.json\_normalize(data, errors='ignore')

df

**#Нормализация данных**

df = pd.json\_normalize(data, record\_path=['features', 'properties', 'vehicles', 'participants'], meta = [

    ['features', 'properties','id'],

    ['features', 'properties', 'tags'],

    ['features', 'properties', 'light'],

    ['features', 'properties', 'point'], #2 cols

    ['features', 'properties', 'nearby'],

    ['features', 'properties', 'region'],

    ['features', 'properties', 'address'],

    ['features', 'properties', 'weather'],

    ['features', 'properties', 'category'],

    ['features', 'properties', 'datetime'],

    ['features', 'properties', 'severity'],

    ['features', 'properties', 'vehicles', 'year'],

    ['features', 'properties', 'vehicles', 'brand'],

    ['features', 'properties', 'vehicles', 'color'],

    ['features', 'properties', 'vehicles', 'model'],

    ['features', 'properties', 'vehicles', 'category'],

    ['features', 'properties','dead\_count'],

    ['features', 'properties','participants'],

    ['features', 'properties','injured\_count'],

    ['features', 'properties','parent\_region'],

    ['features', 'properties','road\_conditions'],

    ['features', 'properties','participants\_count'],

    ['features', 'properties','participant\_categories'],

], errors='ignore')

df = pd.concat([df.drop('features.properties.point', axis=1), pd.DataFrame(df['features.properties.point'].tolist())], axis=1)

df

df1 =  (df.set\_index('features.properties.id')['features.properties.participants']

       .apply(pd.Series).stack()

         .apply(pd.Series).reset\_index().drop('level\_1',1))

df = df.merge(df1, how='left', on='features.properties.id')

df=df.drop('features.properties.participants', axis=1)

Рассмотрим количество пустых значений

df.isna().sum()

ВАРИАНТ 1

# Отчёт

2.1 Разбиение набора данных - набор данныхз разбит на обучаюшую и тестовую выборки

2.3 Классификация - выбраны 3 алгоритма классификации

2.4 Обучение - произведена классификация по уровню опасности

**# Сохранение данных**

df.to\_csv('result\_data.csv', encoding='utf-8-sig', index=False)

# 2.4 Обучение

#Импорт моделей

from sklearn.metrics import classification\_report

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import GaussianNB

#Обучение

neigh = KNeighborsClassifier(n\_neighbors=3)

neigh.fit(X\_train, y\_train)

preds=neigh.predict(X\_test)

print(classification\_report(preds, y\_test))

#Обучение

rfc = RandomForestClassifier()

rfc.fit(X\_train, y\_train)

rfc\_preds=rfc.predict(X\_test)

print(classification\_report(rfc\_preds, y\_test))

#Обучение

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

gnb\_preds=gnb.predict(X\_test)

print(classification\_report(gnb\_preds, y\_test))

**Вывод**

Наиболее оптимальной моделью будет KNeighborsClassifier c accuracy f1-score = 0.78 и macro avg f1-score = 0.74, поскольку по сравнению с другими он показал наилучший результат. RandomForestClassifier не будет взят, поскольку у него явное переобучение

# 2.3 Классификация

Рассмотрим три модели классификации

KNeighborsClassifier

Классификация на основе соседей - это тип обучения на основе экземпляров или необобщающего обучения: он не пытается построить общую внутреннюю модель, а просто сохраняет экземпляры обучающих данных. Классификация вычисляется простым большинством голосов ближайших соседей каждой точки: точке запроса назначается класс данных, который имеет наибольшее количество представителей среди ближайших соседей точки.

RandomForestClassifier

Случайный лес — это метаоценка, которая соответствует ряду классификаторов дерева решений для различных подвыборок набора данных и использует усреднение для повышения точности прогнозирования и контроля переобучения. Размер подвыборки управляется параметром max\_samples, если bootstrap=True (по умолчанию), в противном случае для построения каждого дерева используется весь набор данных

GaussianNB

Наи́вный ба́йесовский классифика́тор — простой вероятностный классификатор, основанный на применении теоремы Байеса со строгими (наивными) предположениями о независимости. В зависимости от точной природы вероятностной модели, наивные байесовские классификаторы могут обучаться очень эффективно

Матрикики

Рассмотрим две метрикики для оценивания модели классификации

accuracy f1-score

Это гармоническое среднее значений точности и полноты. Возьмём её, потому что она дает лучшую оценку неправильно классифицированных случаев

macro avg f1-score

macro avg f1-score пожалуй, самый простой из многочисленных методов усреднения. Макроусредненная оценка F1 (или макрооценка F1) вычисляется путем взятия среднего арифметического (также известного как невзвешенное среднее) всех оценок F1 для каждого класса. Этот метод будет взят, поскольку он обрабатывает все классы одинаково, независимо от их значений поддержки

# 2.1 Разбиение набора данных

import pandas as pd

import numpy as np

df=pd.read\_csv('result\_data.csv')

Разобъём набор данных таким образом, как это рекомендовано согласно документации Sklearn. А именно 30 на 70. Как представленно в описании, такая выборка является оптимальной, поскольку абсолютное большинство данных должно находится при обучении модели, чтобы получить наиболее оптимизированную модель со стороны её точности

**Стратификация**

При разделении стратифицируем данные, чтобы получить одинаковую в процентом соотношении выборку, чтобы не было перевеса на какой-то один класс и такая ситуация не повлияла на некорректное обучение модели

df

df.replace([np.inf, -np.inf], np.nan, inplace=True)

df=df[df['Rt']<5].reset\_index(drop=True)

Определение переменной опасности

df1=df[df['Rt']<=0.7]

df1['Danger']

df2=df[(df['Rt']>0.7) & (df['Rt']<=0.95)]

df2['Danger']=1

df3=df[df['Rt']>0.95]

df3['Danger']=2

df=pd.concat([df1, df2, df3]).reset\_index(drop=True)

X=df[['new\_cases', 'new\_deaths', 'Rt']]

y=df['Danger']

#Получение выборок

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42, stratify=y)

**Импорт необходимых библиотек:**

## Отчёт Импорт необходимых библиотек import pandas as pd import seaborn as sns import matplotlib.pyplot as plt

* 1.1 Парсинг данных - Набор данных загружен из репозитория, а также произведенно дополнение некоторыми атрибутами
* 1.2 Предобработка данных и выделение значимых атрибутов - Набор данных обработан от пустых значений, а также определенны наиболее "важные" атрибуты по корреляции Пирсона
* 1.3 Описание структуры набора данных - для каждого атрибута представленно описание и плотность распределения данных
* 1.4 Формирование дополнительных атрибутов - Сформированн дополнительный атрибут Rt, который определяет коэффициента распространения инфекции за последние 8 дней

# Сохранение данных

df.to\_csv('result\_data.csv', encoding='utf-8-sig', index=False)

# 1.4. Формирование дополнительных атрибутов

Сформируем новый атрибут согласно формуле приведённой в задании:

*Rt = число зарегистрированных заболеваний за последние 4 дня / число зарегистрированных заболеваний за предыдущие 4 дня*

**#Алгоритм вычисления нового атрибута Rt**

df['Rt']=None

data=pd.DataFrame()

for country in df['location'].value\_counts().keys():

r=df[df['location']==country].copy()

da=pd.DataFrame()

for i in range(0, len(r), 8):

tida=pd.DataFrame()

su=r['new\_cases'].tail(8).tail(4).sum()/r['new\_cases'].tail(8).head(4).sum()

tida=r.tail(8)

tida['Rt']=su

r.drop(r.tail(8).index,inplace=True)

da=da.append(tida)

data=data.append(da)

data=data.fillna(0)

data.reset\_index(drop=True, inplace=True)

df=data

df.head()

Результат алгоритма выше демонстрирует формирование дополнительного атрибута Rt - коэффициент распространения инфекции

### Анализ возможности определения изменения эпидемиологической

Проанализируем полученные данные для нескольких стран

d=pd.DataFrame({'Russia': [list(df[df['location']=='Russia']['Rt'])[0]],

                'Mexico':[list(df[df['location']=='Mexico']['Rt'])[0]],

                'France': [list(df[df['location']=='France']['Rt'])[0]],

                'Taiwan':[list(df[df['location']=='Taiwan']['Rt'])[0]],

                'United States':[list(df[df['location']=='United States']['Rt'])[0]],

                'Japan':[list(df[df['location']=='Japan']['Rt'])[0]],

                'Canada':[list(df[df['location']=='Canada']['Rt'])[0]],

                'Singapore':[list(df[df['location']=='Singapore']['Rt'])[0]],}).T

**Выведем актуальные Rt - значения для каждой страны**

plt.rcParams.update({'font.size': 15,})

plt.figure(figsize=(15, 8))

plots = sns.barplot(x=d.index, y=d[0], data=df)

for bar in plots.patches:

plots.annotate(format(bar.get\_height(), '.2f'),

(bar.get\_x() + bar.get\_width() / 2,

bar.get\_height()), ha='center', va='center',

size=15, xytext=(0, 8),

textcoords='offset points')

plt.title('Анализ эпидемиологической обстановки')

plt.ylabel('Rt - значение')

plt.xlabel('Страна')

plt.show()

Из полученных результатов можно сказать, что максимальное Rt - значение предложенных стран - это Сингапур(2.27). Минимальное - Мексика (0.56). Россия на текущий день имеет Rt значение 0.94

# 1.3. Описание структуры набора данных

**Описание каждого атрибута**

df.info()

total\_cases - Всего подтвержденных случаев COVID-19.

new\_cases - Новые подтвержденные случаи заболевания COVID-19. Подсчеты могут включать вероятные случаи, о которых сообщается

new\_cases\_smoothed - Новые подтвержденные случаи COVID-19 (7-дневный период сглажен). Подсчеты могут включать вероятные случаи, о которых сообщается.

total\_cases\_per\_million - Общее количество подтвержденных случаев COVID-19 на 1 000 000 человек. Подсчеты могут включать вероятные случаи, о которых сообщается.

new\_cases\_per\_million - Новые подтвержденные случаи COVID-19 на 1 000 000 человек. Подсчеты могут включать вероятные случаи, о которых сообщается.

new\_cases\_smoothed\_per\_million - Новые подтвержденные случаи COVID-19 (сглаженные за 7 дней) на 1 000 000 человек. Подсчеты могут включать вероятные случаи, о которых сообщается.

total\_deaths - Общее число смертей, связанных с COVID-19. Сообщалось, что подсчеты могут включать вероятные смертельные случаи.

new\_deaths - Новые случаи смерти, связанные с COVID-19. Сообщалось, что подсчеты могут включать вероятные смертельные случаи. В редких случаях, когда наш источник сообщает об отрицательном ежедневном изменении из-за корректировки данных, мы устанавливаем для этой метрики значение NAME.

new\_deaths\_smoothed - Новые случаи смерти, связанные с COVID-19 (7-дневный период сглажен). Сообщалось, что подсчеты могут включать вероятные смертельные случаи.

total\_deaths\_per\_million - Общее число смертей, связанных с COVID-19, на 1 000 000 человек. Сообщалось, что подсчеты могут включать вероятные смертельные случаи.

new\_deaths\_per\_million - Новые случаи смерти, связанные с COVID-19, на 1 000 000 человек. Сообщалось, что подсчеты могут включать вероятные смертельные случаи.

new\_deaths\_smoothed\_per\_million - Новые случаи смерти, связанные с COVID-19 (сглаженные за 7 дней) на 1 000 000 человек. Сообщалось, что подсчеты могут включать вероятные смертельные случаи.

excess\_mortality - Процентная разница между зарегистрированным числом еженедельных или ежемесячных смертей в 2020-2021 годах и прогнозируемым числом смертей за тот же период на основе предыдущих лет. Для получения дополнительной информации см. <https://github.com/owid/covid-19-data/tree/master/public/data/excess_mortality>

excess\_mortality\_cumulative - Процентная разница между совокупным числом смертей с 1 января 2020 года и совокупным прогнозируемым числом смертей за тот же период, основанным на предыдущих годах. Для получения дополнительной информации см. <https://github.com/owid/covid-19-data/tree/master/public/data/excess_mortality>

excess\_mortality\_cumulative\_absolute - Совокупная разница между зарегистрированным числом смертей с 1 января 2020 года и прогнозируемым числом смертей за тот же период на основе предыдущих лет. Для получения дополнительной информации см. <https://github.com/owid/covid-19-data/tree/master/public/data/excess_mortality> избыток

excess\_mortality\_cumulative\_per\_million- Совокупная разница между зарегистрированным числом смертей с 1 января 2020 года и прогнозируемым числом смертей за тот же период на основе предыдущих лет на миллион человек. Для получения дополнительной информации см. <https://github.com/owid/covid-19-data/tree/master/public/data/excess_mortality>

icu\_patients - Количество пациентов с COVID-19 в отделениях интенсивной терапии (ОИТ) в данный день

icu\_patients\_per\_million - Количество пациентов с COVID-19 в отделениях интенсивной терапии (ОИТ) в данный день на 1 000 000 человек

hosp\_patients - Количество пациентов с COVID-19 в больнице в данный день

hosp\_patients\_per\_million - Количество пациентов с COVID-19, находящихся в больнице в данный день на 1 000 000 человек

weekly\_icu\_admissions - Число пациентов с COVID-19, впервые поступивших в отделения интенсивной терапии (ОИТ) за данную неделю

weekly\_icu\_admissions\_per\_million - Число пациентов с COVID-19, впервые поступивших в отделения интенсивной терапии (ОИТ) за данную неделю на 1 000 000 человек

weekly\_hosp\_admissions - Число пациентов с COVID-19, впервые поступивших в больницы за данную неделю

weekly\_hosp\_admissions\_per\_million - Число пациентов с COVID-19, впервые поступивших в больницы за данную неделю на 1 000 000 человек

stringency\_index - Индекс строгости реагирования правительства: сводный показатель, основанный на 9 показателях реагирования, включая закрытие школ, закрытие рабочих мест и запреты на поездки, масштабированный до значения от 0 до 100 (100 = самый строгий ответ)

reproduction\_rate - Оценка в реальном времени эффективной скорости размножения (R) COVID-19. <https://github.com/crondonm/TrackingR/tree/main/Estimates-Database>

total\_tests - Общее количество тестов на COVID-19

new\_tests - Новые тесты на COVID-19 (рассчитываются только для последовательных дней)

total\_tests\_per\_thousand - Общее количество тестов на COVID-19 на 1000 человек

new\_tests\_per\_thousand - Новые тесты на COVID-19 на 1000 человек

new\_tests\_smoothed - Новые тесты на COVID-19 (7-дневный сглаженный). Для стран, которые не сообщают данные тестирования на ежедневной основе, мы предполагаем, что тестирование менялось одинаково ежедневно в течение любых периодов, в течение которых данные не сообщались. Это дает полный ряд ежедневных показателей, которые затем усредняются в течение скользящего 7-дневного окна

new\_tests\_smoothed\_per\_thousand - Новые тесты на COVID-19 (7-дневный сглаженный) на 1000 человек

positive\_rate - Доля положительных тестов на COVID-19, приведенная в виде скользящего среднего за 7 дней (это обратная величина tests\_per\_case)

tests\_per\_case - Тесты, проведенные для каждого нового подтвержденного случая COVID-19, приведены в виде скользящего среднего значения за 7 дней (это обратная величина positive\_rate)

tests\_units - Единицы измерения, используемые местоположением для представления своих данных тестирования

total\_vaccinations - Общее количество доз вакцинации против COVID-19, введенных вакцинированным людям

people\_vaccinated - Общее число людей, получивших хотя бы одну дозу вакцины

people\_fully\_vaccinated - Общее число людей, получивших все дозы, предписанные первоначальным протоколом вакцинации

total\_boosters - Общее количество введенных бустерных доз вакцинации против COVID-19 (дозы, введенные сверх количества, предписанного протоколом вакцинации)

new\_vaccinations - Вводимые новые дозы вакцинации против COVID-19 (рассчитываются только для последовательных дней)

new\_vaccinations\_smoothed - Введены новые дозы вакцинации против COVID-19 (7-дневный период сглажен). Для стран, которые не сообщают данные о вакцинации на ежедневной основе, мы предполагаем, что вакцинация менялась одинаково ежедневно в течение любых периодов, в течение которых данные не сообщались. Это дает полный ряд ежедневных показателей, которые затем усредняются в течение скользящего 7-дневного окна

total\_vaccinations\_per\_hundred - Общее количество доз вакцинации против COVID-19, введенных на 100 человек в общей популяции

people\_vaccinated\_per\_hundred - Общее число людей, получивших по крайней мере одну дозу вакцины на 100 человек в общей численности населения

people\_fully\_vaccinated\_per\_hundred- Общее число людей, получивших все дозы, предписанные протоколом первичной вакцинации, на 100 человек в общей численности населения

total\_boosters\_per\_hundred - Общее количество бустерных доз вакцинации против COVID-19, введенных на 100 человек в общей численности населения

new\_vaccinations\_smoothed\_per\_million - Новые дозы вакцинации против COVID-19, введенные (сглаженные на 7 дней) на 1 000 000 человек от общей численности населения

new\_people\_vaccinated\_smoothed - Ежедневное количество людей, получающих свою первую дозу вакцины (сглажено на 7 дней)

new\_people\_vaccinated\_smoothed\_per\_hundred - Ежедневное число людей, получающих первую дозу вакцины (7-дневная сглаженная) на 100 человек в общей численности населения

iso\_code - ISO 3166-1 alpha-3 – трехбуквенные коды стран

continent - Континент географического местоположения

location - Географическое положение

date - Дата наблюдения

population - Численность населения для получения полного списка источников

population\_density - Количество людей, разделенное на площадь земли, измеренную в квадратных километрах, самый последний год, доступный

median\_age - Средний возраст населения, прогноз ООН на 2020 год

aged\_65\_older - Доля населения в возрасте 65 лет и старше, последний доступный год

aged\_70\_older - Доля населения в возрасте 70 лет и старше в 2015 году

gdp\_per\_capita - Валовой внутренний продукт по паритету покупательной способности (постоянный международный доллар 2011 года), последний доступный год

extreme\_poverty - Доля населения, живущего в крайней нищете, последний год, доступный с 2010 года

cardiovasc\_death\_rate - Уровень смертности от сердечно-сосудистых заболеваний в 2017 году (годовое число смертей на 100 000 человек)

diabetes\_prevalence - Распространенность диабета (% населения в возрасте от 20 до 79 лет) в 2017 году

female\_smokers - Доля курящих женщин, за последний год доступных

male\_smokers - Доля курящих мужчин, за последний год доступных

handwashing\_facilities - Доля населения, имеющего базовые средства для мытья рук в помещениях, за последний год количество доступных

hospital\_beds\_per\_thousand - Больничные койки на 1000 человек, последний доступный год с 2010 года

life\_expectancy - Ожидаемая продолжительность жизни при рождении в 2019 году

human\_development\_index - Сводный индекс, измеряющий средние достижения в трех основных аспектах человеческого развития — долгая и здоровая жизнь, знания и достойный уровень жизни.

## **Пустые значения**

Пустые значения были предобработанны ранее, после предобработки их больше не осталось

pd.set\_option('display.max\_rows',None)

df.isnull().sum()

pd.set\_option('display.max\_rows',10)

## Плотность распределения данных

**Сформируем графики плотностей для каждого признака**

df.head()

**# Распределение категориальной переменной**

plt.figure(figsize=(10, 5))

sns.kdeplot(df['iso\_code'].value\_counts())

plt.title('Distribution iso\_code')

plt.xlabel('Значения')

plt.ylabel('Распределение')

plt.show()

**# Распределение категориальной переменной**

plt.figure(figsize=(10, 5))

sns.kdeplot(df['continent'].value\_counts())

plt.title('Distribution continent')

plt.xlabel('Значения')

plt.ylabel('Распределение')

plt.show()

**# Распределение категориальной переменной**

plt.figure(figsize=(10, 5))

sns.kdeplot(df['location'].value\_counts())

plt.title('Distribution location')

plt.xlabel('Значение')

plt.ylabel('Распределение')

plt.show()

**#Функция вывода распределения каждого атрибута, являющегося численным признаком**

def plot(column):

plt.figure(figsize=(10, 5))

sns.kdeplot(df[column])

plt.title('Distribution '+column)

plt.xlabel('Значения')

plt.ylabel('Распределение')

plt.show()

**#Применение функции**

for column in df[:100].select\_dtypes(exclude=['object']).columns:

plot(column)

df.head()

**# Распределение категориальной переменной**

plt.figure(figsize=(10, 5))

sns.kdeplot(df['iso\_code'].value\_counts())

plt.title('Distribution iso\_code')

plt.xlabel('Значения')

plt.ylabel('Распределение')

plt.show()

**# Распределение категориальной переменной**

plt.figure(figsize=(10, 5))

sns.kdeplot(df['continent'].value\_counts())

plt.title('Distribution continent')

plt.xlabel('Значения')

plt.ylabel('Распределение')

plt.show()

**# Распределение категориальной переменной**

plt.figure(figsize=(10, 5))

sns.kdeplot(df['location'].value\_counts())

plt.title('Distribution location')

plt.xlabel('Значение')

plt.ylabel('Распределение')

plt.show()

**#Функция вывода распределения каждого атрибута, являющегося численным признаком**

def plot(column):

plt.figure(figsize=(10, 5))

sns.kdeplot(df[column])

plt.title('Distribution '+column)

plt.xlabel('Значения')

plt.ylabel('Распределение')

plt.show()

**#Применение функции**

for column in df[:100].select\_dtypes(exclude=['object']).columns:

plot(column)

# 1.2. Предобработка данных и выделение значимых атрибутов

**Заполним все пустые значения нулями**

df=df.fillna(0)

**Определение наиболее значимых атрибутов**

**Чтобы найти наиболее значимые атрибуты, построим корреляцию Пирсона на тепловой карте**

**Фомирование корреляции Пирсона**

corr=df.corr()

plt.figure(figsize=(70, 70))

heatmap = sns.heatmap(corr,cmap = "YlGnBu", vmin=-1, vmax=1, annot=True)

heatmap.set\_title('Correlation Heatmap', fontdict={'fontsize':20}, pad=20)

**Как видим выше, достаточно много признаков имеют высокий коэффициент корреляции, но наиболее значимыми атрибутами являются: total\_casem, new\_case, new\_cases\_smoothed, total\_deaths, new\_deaths и new\_deaths\_smoothed**

# 1.1 Парсинг данных ВАРИАНТ 1 ↑

**# Загрузка данных из репозитория**

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

**# Отображение таблицы**

df.head()

**# Приведение типов**

df['date']=pd.to\_datetime(df['date'])

**# Отображение типов**

df.info()

**#Вывод пустых значений**

pd.set\_option('display.max\_rows',None)

df.isnull().sum()

pd.set\_option('display.max\_rows',10)

**#Количество строк и столбцов в датафрейме**

df.shape

Все данные при парсинге из репозитория были включены. Размерность набора данных составляет `183348 строк` и `67 столбцов`. Дополним набор данных новой информацией, чтобы в случае необходимости точность модели при обучении была больше. Также новые данные могут пригодиться для того, чтобы включить их в анализ данных, из которого можно вынести некоторые зависимости, если они присутствует.

В качестве новых данных возьмём среднее количество смертей и заражённых на регион.

**# Заполнение пустых полей**

df[['location', 'new\_cases', 'new\_deaths']]=df[['location', 'new\_cases', 'new\_deaths']].fillna(0)

**# Формирование дополнительных атрибутов**

grouped\_cases=df[['location',

'new\_cases', 'new\_deaths']].groupby(by="location").mean().rename(columns={'new\_cases':'mean\_new\_cases',

'new\_deaths':'mean\_new\_deaths'})

grouped\_cases

df=df.merge(grouped\_cases, on='location')

df.head()

**Генерация новых данных произведена**