Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Факультет дистанционного обучения (ФДО)

А. Я. Суханов

РАЗРАБОТКА ВЕБ-СЕРВИСОВ ДЛЯ НАУЧНЫХ И ПРИКЛАДНЫХ ЗАДАЧ

**Методические указания
по выполнению лабораторных работ
для студентов, обучающихся с применением
дистанционных образовательных технологий**

Томск 2021

Корректор: А. Н. Миронова

Суханов А. Я.

Разработка веб-сервисов для научных и прикладных задач : методические указания по выполнению лабораторных работ для студентов, обучающихся с применением дистанционных образовательных технологий / А. Я. Суханов. – Томск : ФДО, ТУСУР, 2021. – 119 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данные методические указания разработаны с учетом требований ФГОС ВО для направления подготовки 09.03.01 «Информатика и вычислительная техника», они предназначены для выполнения лабораторных работ по дисциплине «Разработка веб-сервисов для научных и прикладных задач».

Цель лабораторных работ: получение навыков разработки веб-приложений и веб-сервисов с использованием одного из фреймворков и типовых научных библиотек для решения простейших задач по обработке данных.

Лабораторные работы выполняются в соответствии с порядком, описанным в методических указаниях.

Выбор варианта лабораторной работы осуществляется по общим правилам с использованием следующей формулы:

$$V = (N \times K) \text{ div } 100,$$

где V – искомый номер варианта,

N – общее количество вариантов,

K – код варианта,

div – целочисленное деление.

При $V = 0$ выбирается максимальный вариант.

1 ОСНОВНЫЕ ВОЗМОЖНОСТИ PYTHON

1.1 Кратко о языке Python

Python является интерпретируемым языком, в силу этого многое можно выполнять в командной строке. Достоинством Python является достаточно простое написание кода, при этом код выглядит лаконичным. Программу или проект проще всего реализовать на Python, можно быстро получить первые результаты, потому язык изначально распространился в научной среде.

Для выделения исполнимых блоков используются отступы, обычно четыре пробела, нет необходимости в завершающих или выделяющих программный блок символах: {} или begin end. На Python реализовано множество пакетов обработки данных, библиотек машинного обучения, нейронных сетей. Python является третьим по популярности языком.

Наверное, основным недостатком Python является то, что код для выполнения простейших вычислительных задач, написанный без использования возможностей массивов numpy и других библиотек, будет уступать по скорости исполнения компилируемым языкам и JIT-компилируемым. Кроме того, интерпретатор Python не обеспечивает распараллеливание потоков по ядрам процессора. Для ускорения исполнения вы можете пользоваться версией JIT-компилятора PyPy для Python. Для решения проблемы блокировки и возможности исполнения потоков на разных ядрах разрабатывается pypy-stm.

1.2 Начало работы

Перед началом работы с языком Python следует кратко изучить **PEP8 – руководство по написанию кода на Python** (https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id2).

Некоторые IDE обеспечивают автоматическую проверку на PEP8. Рекомендуется использовать, например, PyCharm, более простые IDE – Spyder, Idle.

Старайтесь при написании кода соблюдать нижеприведенные соглашения.

Используйте четыре пробела для отступа (при выполнении лабораторных работ можно использовать и табуляцию, но этот способ не рекомендуется). Длинные строки следует опускать ниже знаков: (, {, [, а следующие начинать после этих знаков сверху:

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

либо

```
foo = long_function_name(
    var_one, var_two,
    var_three, var_four
)
```

Имена классов используют соглашение CapitalizedWords (слова с заглавными буквами, или CapWords, или CamelCase).

Замечание: когда вы используете аббревиатуры в таком стиле, пишите все буквы аббревиатуры заглавными – HTTPServerError.

Имена функций и переменных должны состоять из строчных букв, а слова разделяться символами подчеркивания: lower_case_with_underscores. Это необходимо для удобочитаемости.

Константы следует представлять в формате UPPERCASE_WITH_UNDERSCORES (слова из заглавных букв с подчеркиваниями).

1.3 Установка Python и настройка environment

Процедуры установки потребуют наличия операционной системы Linux либо установки виртуальной машины Linux на VirtualBox. Способы установки VirtualBox можно найти на различных сайтах, дадим лишь краткое указание.

Прежде чем устанавливать VirtualBox в вашу систему Windows, зайдите в BIOS компьютера или ноутбука и в разделе «Настройки CPU» включите поддержку виртуализации. На странице загрузки официального сайта https://www.virtualbox.org/ выберите последнюю версию Oracle VM VirtualBox. Например, VirtualBox для Windows и других платформ можно скачать по ссылке:

https://www.oracle.com/virtualization/technologies/vm/downloads/virtualbox-downloads.html. Дальше необходимо запустить установку и следовать указаниям.

Скачайте образ диска с операционной системой Linux. Предлагаем скачать десктоп-версию операционной системы Ubuntu. В частности, скачать ISO-образ операционной системы версии 20.04 можно на сайте: https://releases.ubuntu.com/20.04/.

На базе скачанного образа создадим виртуальную машину. Выполним в VirtualBox «машина → создать».

Выберем папку для сохранения виртуальной машины и тип операционной системы. затем – «Создать новый виртуальный жесткий диск». Можно выбрать гораздо меньше оперативной памяти, например 2 или 4 Гб (рис. 1.1).

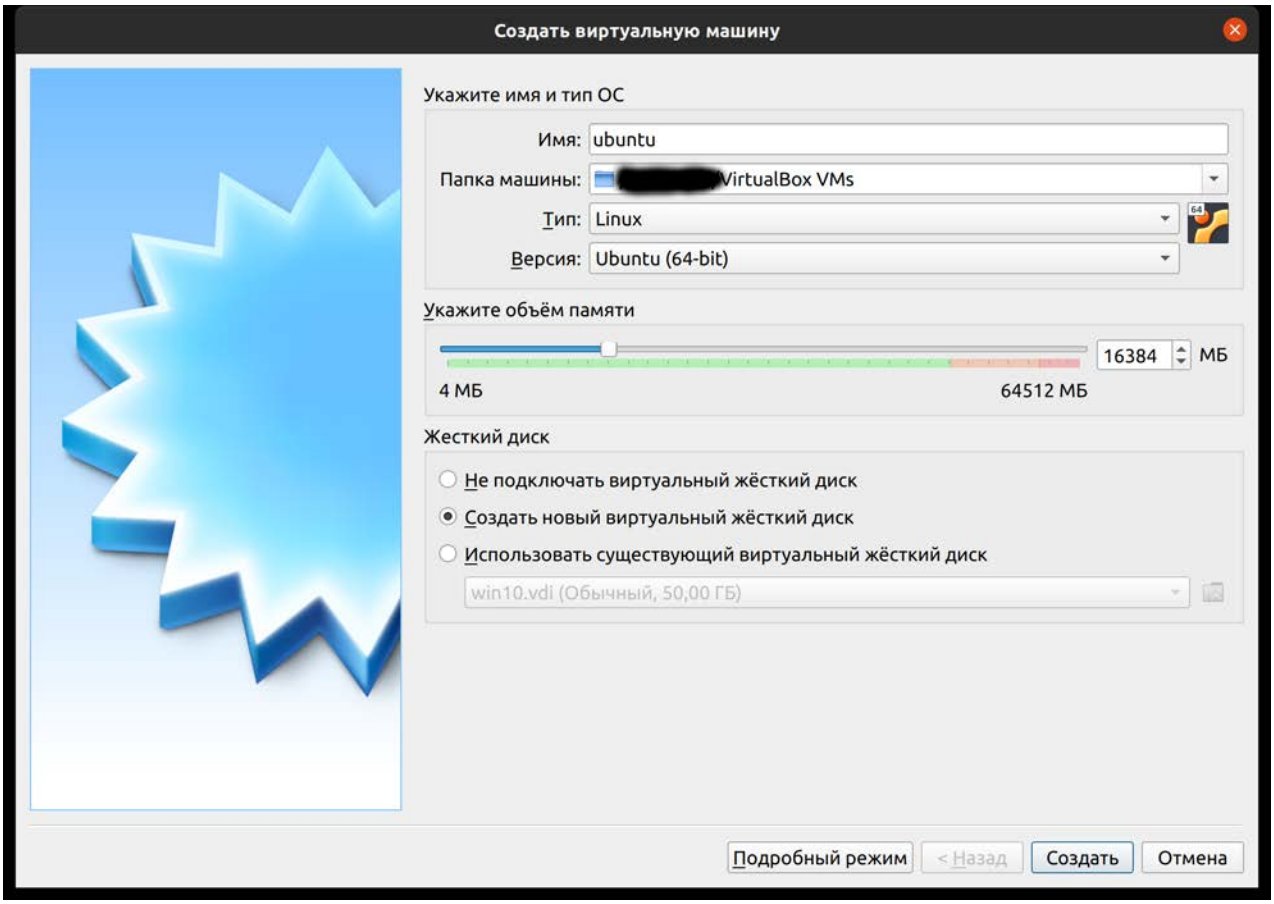

Рис. 1.1 – Установка размера оперативной памяти для ОС и создание виртуального жесткого диска

Далее появится окно создания виртуального диска. Можно выбрать VDI-диск, это родной для системы VirtualBox тип виртуальных дисков, указать размер, например, 20 Гб, динамически расширяющийся.

В окошке созданных машин можно установить различные настройки (рис. 1.2).

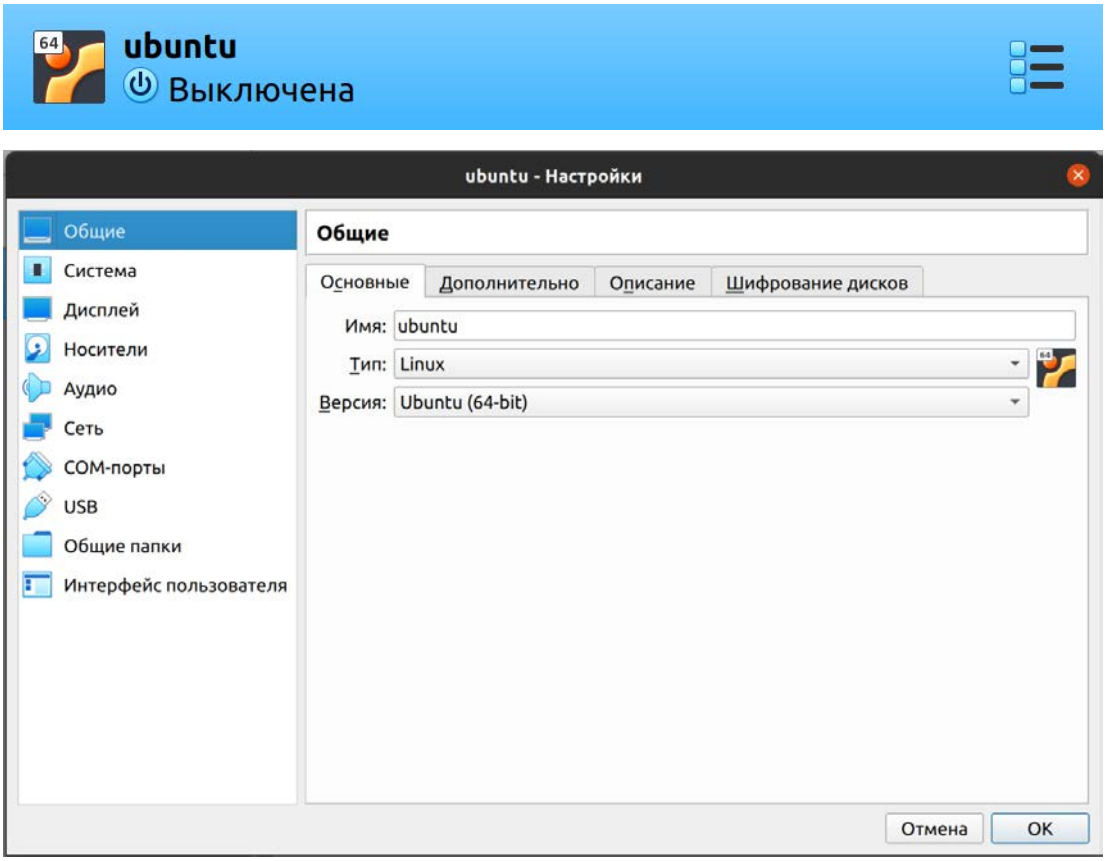

Рис. 1.2 – Настройки виртуальной машины

Во вкладке «Общие папки» можно выбрать разделяемые с хостовой операционной системой каталоги. Для этого следует нажать на папку с зеленым плюсом (рис. 1.3).

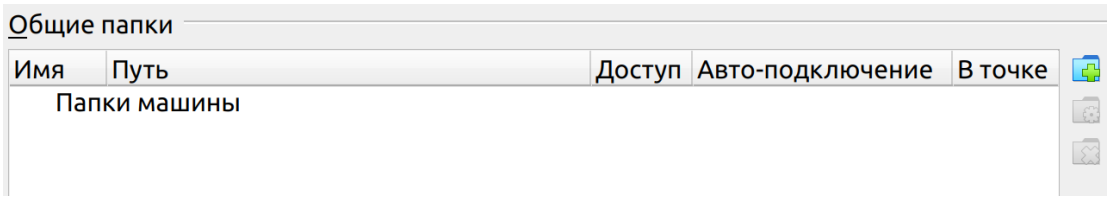

Рис. 1.3 – Добавление разделяемых папок

Во вкладке «Носители» можно добавить дисковый носитель, в частности наш скачанный ISO-образ для загрузки и установки операционной системы. Для этого нажимаем на плюсик на фоне круглого диска (рис. 1.4).

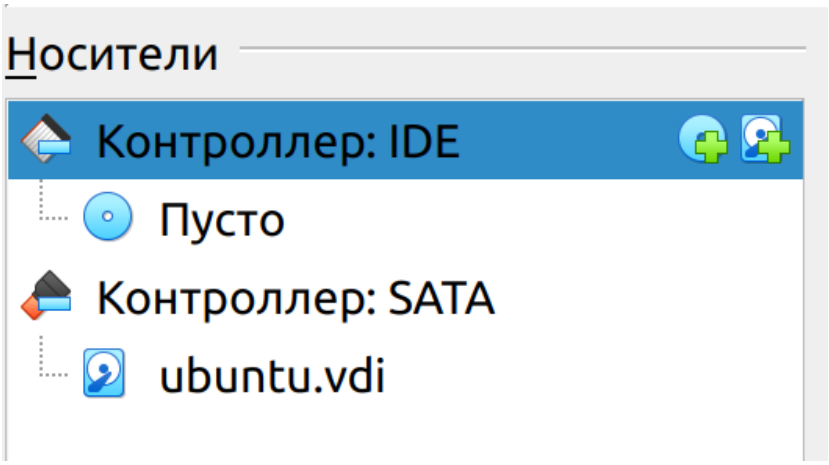

Рис. 1.4 – Добавление устройства диска
или другого носителя информации

Затем жмем «Добавить» (рис. 1.5) и выбираем скачанный нами ISO-образ.

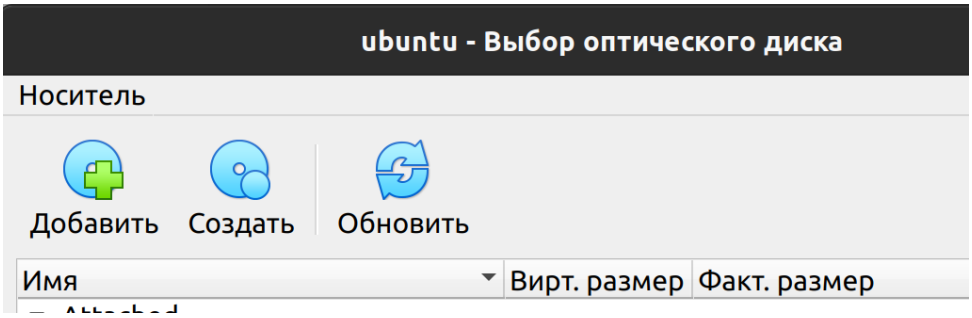

Рис. 1.5 – Добавлении ISO-образа

После этого в окне VirtualBox выбираем нашу виртуальную машину и нажимаем «Запустить».

Далее появится окно установки операционной системы Ubuntu (рис. 1.6). Следуйте указаниям установщика. Выберете стандартную установку, задайте имя и пароль пользователя.

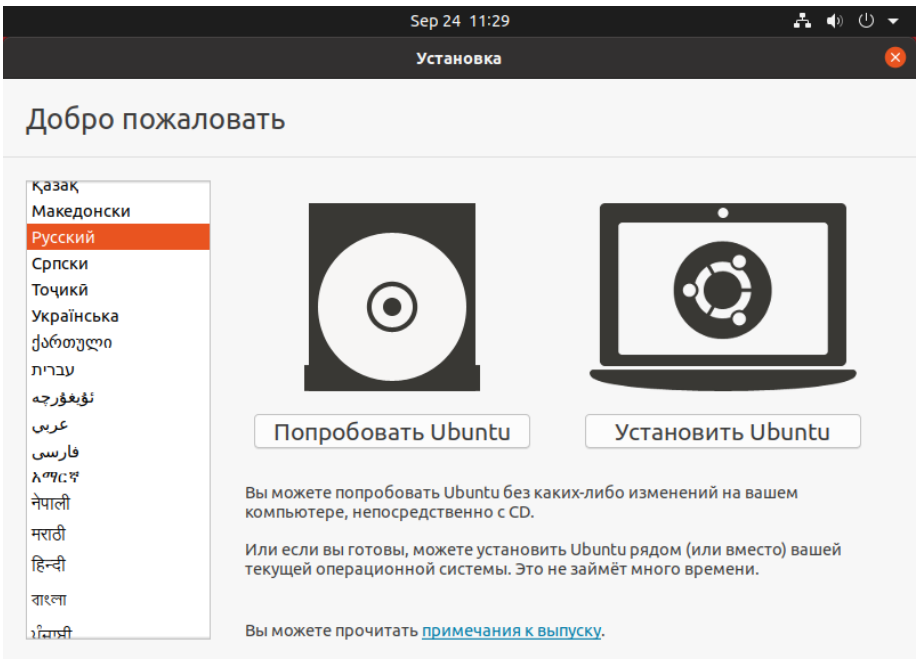

Рис. 1.6 – Окно установки Ubuntu

После установки в меню устройства можно выбрать «Подключить образ диска дополнений гостевой ОС».

Ubuntu 20.04 и другие версии Debian Linux поставляются с предустановленным Python 3. Чтобы посмотреть версию Python, в терминале Ubuntu запустите команду:

python3 -V

Если установлена версия Python 3.8, то все нормально, можно продолжить. Если Python не установлен, его нужно установить, причем именно версию 3.8.

Если все-таки оказалось, что Python 3.8 не установлен или вы устанавливаете его не на виртуальной машине Ubuntu 20.04, то необходимо проделать следующие шаги:

- Обновить список пакетов и установить необходимые библиотеки:

sudo apt update

sudo apt install software-properties-common

- Добавить репозиторий deadsnakes PPA в список источников вашей системы:

sudo add-apt-repository ppa:deadsnakes/ppa

- При появлении запроса нажать Enter.
- Установить Python 3.8.

sudo apt install python3.8

- Для управления пакетами установим pip (pip3) – систему управления пакетами Python.

Примеры:

pip${version} install some-package-name

pip install numpy.

pip3.8 install numpy.

Установка для конкретной версии Python и установка numpy.

Выполним команду:

sudo apt install -y python3-pip

После этого пройдет установка pip.

1.4 Настройка виртуальной среды

Виртуальные среды позволяют иметь на компьютере изолированное пространство для проектов Python, гарантируя, что каждый из ваших проектов может иметь свой собственный набор зависимостей, который не нарушит ни один из других проектов.

Настройка среды программирования дает больший контроль над проектами Python и над тем, как обрабатываются разные версии пакетов. Это особенно важно при работе со сторонними пакетами.

Можно настроить любое количество сред программирования Python. Каждая среда – это отдельный каталог на вашем компьютере, в котором есть несколько сценариев, позволяющих выполняться приложению в этой отдельной среде.

Сначала нужно установить модуль *venv*, являющийся частью стандартной библиотеки Python 3, чтобы можно было создавать виртуальные среды. Установим venv, набрав:

sudo apt-get install -y python3-venv

После установки настроим отдельную среду для нашего приложения.

Сначала создадим каталог, где мы будем размещать среды и зайдем в него.

```
mkdir envs
cd envs
```

Далее введем команду для создания виртуальной среды нашего приложения, назовем среду proj1.

```
python3 -m venv proj1
```

Перейдем в каталог и просмотрим его.

```
:~/envs$ cd proj1
:~/envs/proj1$ ls
bin  include  lib  lib64  pyvenv.cfg  share
```

Совместная работа этих файлов обеспечивает изоляцию проектов, так что системные файлы и файлы проекта не смешиваются. Будет очень полезно использовать контроль версий и обеспечить каждому из проектов доступ к конкретным пакетам, которые ему необходимы. Python Wheels – это формат готовых пакетов для Python, который помогает ускорить разработку программного обеспечения за счет сокращения количества операций компиляции. Он находится в каталоге share в Ubuntu 20.04.

Чтобы использовать эту среду, ее нужно активировать. Для этого необходимо ввести команду, вызывающую скрипт activate (перед этим необходимо вернуться или зайти в папку со средами).

```
:~$ cd ..
:~/envs$ source proj1/bin/activate
```

Для выхода из среды окружения proj1 введите команду deactivate.

```
(proj1) :~/envs$ deactivate
```

Далее можно опять активировать среду, создать свой первый проект и запустить программу на Python. Саму программу в принципе можно делать, используя любой текстовый редактор. Очевидно, это не совсем удобно, но можно сделать первый проект, используя простой редактор. Запустим терминал, если вы его еще не запустили, и создадим какой-нибудь каталог для наших проектов, например, progs.

```
$mkdir progs
$cd progs
```

Активируем среду.

```
$source ../envs/proj1/bin/activate
```

Создадим проект hello, находясь в окружении.

```
$mkdir hello
$cd hello
$nano hello.py
```

Внутри файла запишем строчку:

```
print("Hello world")
```

Сохраним файл (Ctrl+O) и запустим его из командной строки интерпретатором Python.

```
$python hello.py.
```

Добавим в наш файл следующие строчки:

```
import numpy as np
print("Hello world")
x = np.array([1,2,3,4])
print(x)
```

И запустим наш файл, при этом появится сообщение о том, что модуль numpy неизвестен. Его мы установим в нашем окружении с использованием pip.

```
$pip3 install numpy
```

или

```
$pip install numpy
```

И вызов дает нам ожидаемый результат.

```
$python hello.py
Hello world
[1 2 3 4]
```

Конечно, писать код в подобном редакторе неудобно, поэтому установим IDE. К сожалению, в виртуальной машине IDE может работать медленно, но вы можете продолжить работать в текстовом редакторе либо использовать более легковесные IDE, а не предлагаемую нами далее среду PyCharm. Можете установить свободно распространяемую версию PyCharm в установщике Ubuntu для обучения и изучения либо профессиональную версию.

Зайдем в центр приложений Ubuntu Software и выберем PyCharm EDU (рис. 1.7).

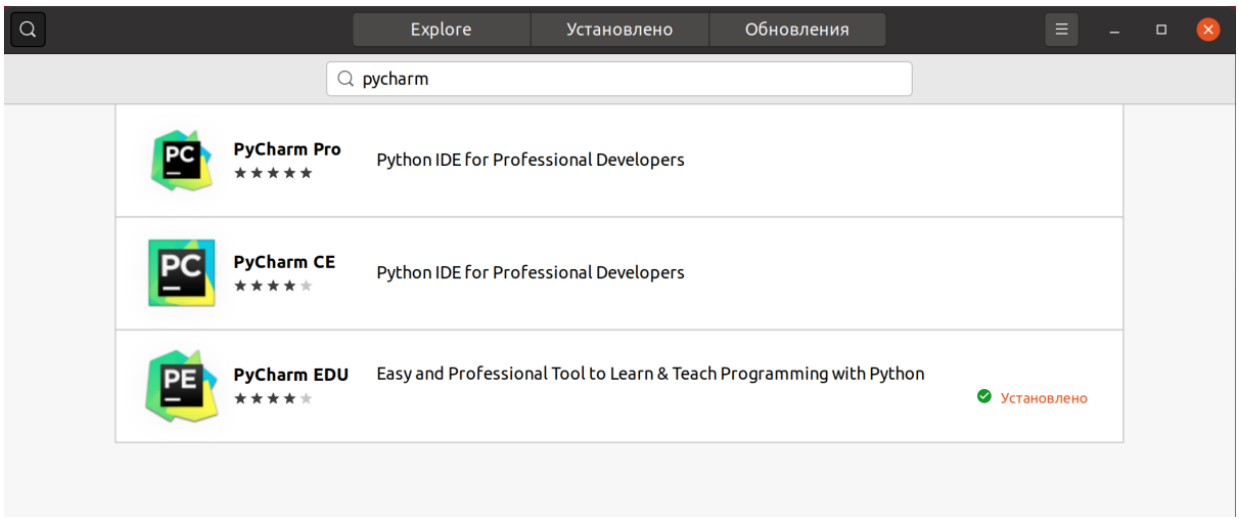

Рис. 1.7 – Выбор IDE PyCharm в Ubuntu Software

Запустим PyCharm (рис. 1.8) и зададим конфигурацию проекта.

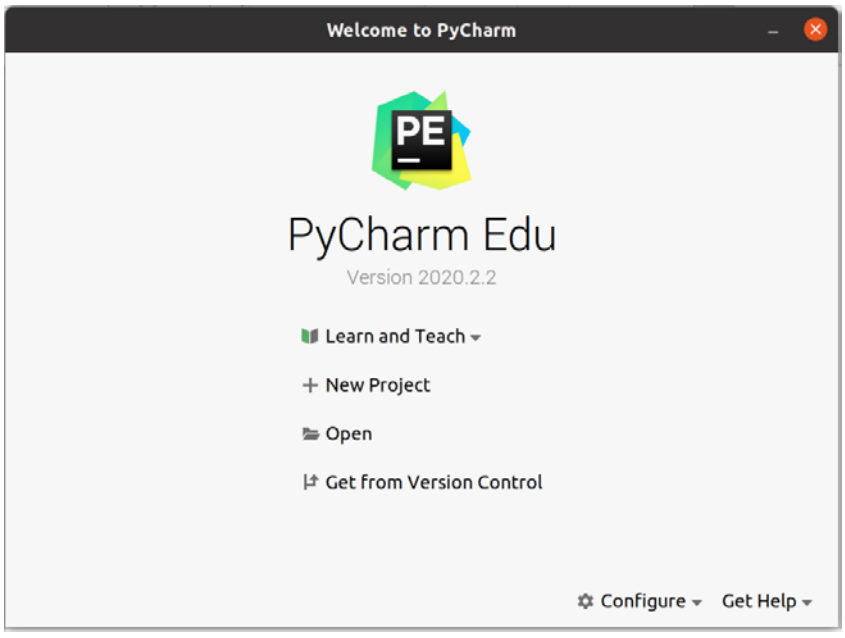

Рис. 1.8 – Конфигурация, создание и открытие проекта

В меню Configure можно установить еще одну среду.

Configure → Settings → PythonIntepreter → Add

Можно выбрать и уже существующую среду, созданную нами (рис. 1.9, 1.10).

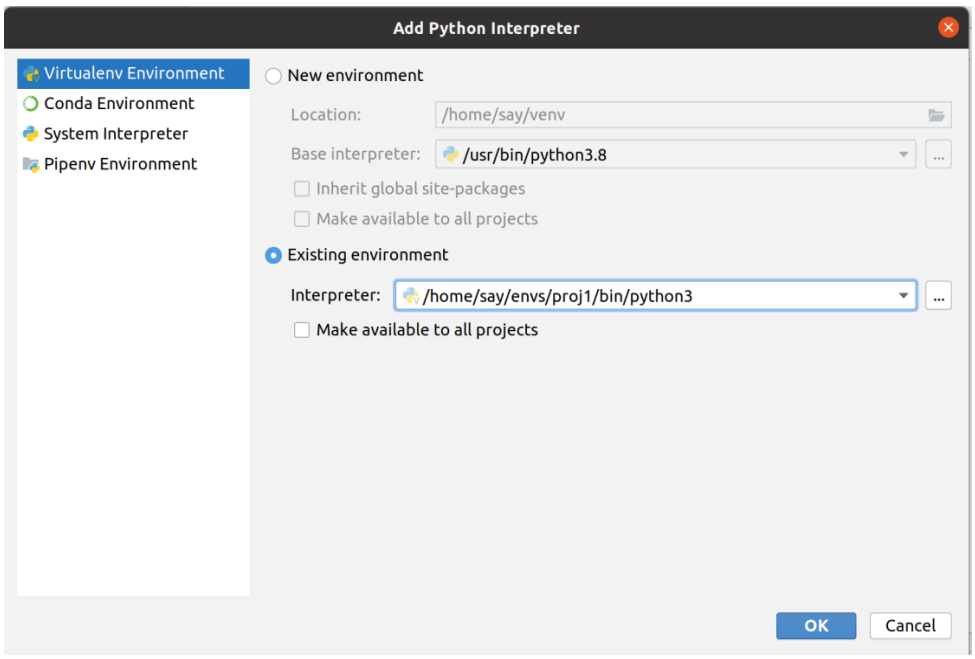

Рис. 1.9 – Выбор собственной среды окружения

Далее создадим свой проект с указанной средой окружения.

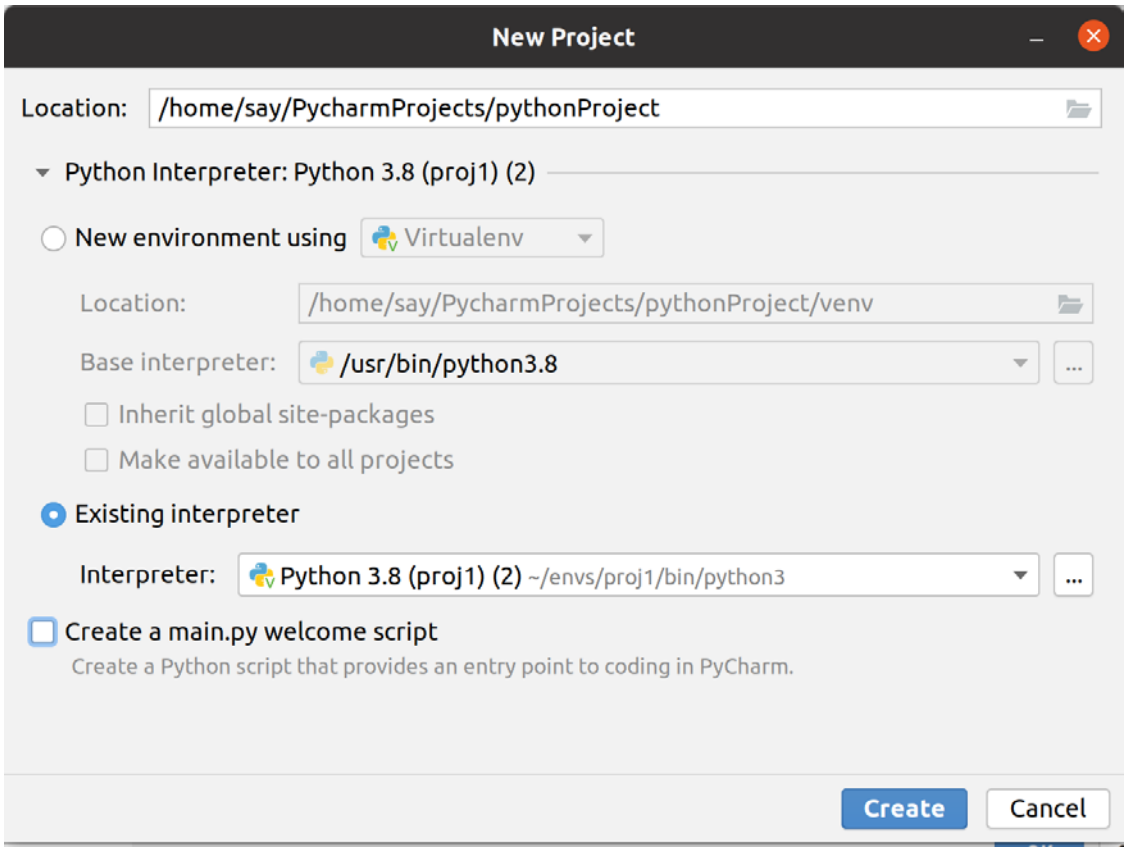

Рис. 1.10 – Создание проекта в указанной среде окружения

Вы можете выбрать другое название проекта и любую среду.

После создания проекта при выборе вкладки terminal в среде PyCharm отобразится командная строка с соответствующей средой proj1.

Выбрав File → Settings, можно посмотреть вкладку Project: PythonInterpreter, здесь мы увидим установленные библиотеки, в том числе numpy и pip.

Выбрав File → New… и создав файл hello, опять запишем код print("Hello world") и запустим программу с помощью зеленого треугольника.

Выберем в окне IDE PyCharm терминал и в нашем окружении установим библиотеки, которые нам понадобятся для разработки веб-приложения.

1.5 Первые шаги в Python

Прежде чем приступать к лабораторной работе, связанной с веб-приложением, предлагается выполнить приведенные ниже задания. Подготовительная работа поможет при выполнении заданий по вариантам при создании веб-приложения.

В консоли попробуйте запустить представленные команды.

Информация о базовых операциях и типах.

```
#Типы данных, int, float, bool, str
>>>type(int)
<class 'type'>
>>>type(2+2)
<class 'int'>
>>> str1 = "Hello"
>>> print(str1)
Hello
>>> type(str1)
<class 'str'>
>>> d = (1+2j)
>>> type(d)
<class 'complex'>
>>> complex(2.0)
(2+0j)
```

В Python, как вы уже, может быть, догадались, все переменные являются ссылками на объект. Если объект не используется, т. е. переменные не ссылаются на этот объект, то объект может быть удален из памяти, обычно сборщиком мусора. Например, CLR или JVM (являющиеся JIT-компиляторами, исполняющими байт-код) выполняют эту работу.

При написании кода на Python для начала нужно соблюдать следующие простые правила.

В Python при именовании различаются заглавные и прописные символы.

В названиях переменных не использовать одиночные переменные 0, l.

Название переменных следует начинать знаком _ (например, _x_value), только если они используются как внутренние переменные модуля. Например, при подключении from mod1 import * не будет импортирована переменная _x_value, если, конечно, не использовать в модуле mod1 __all__ (позволяющий описать публичные объекты данного модуля).

Создайте модуль (Add file). Назовите его mod1.py.

Запишите туда код.

```
_x_value = 10
y_value = 10
__all__=['_x_value','y_value']
```

В файл основной программы main.py поместите такой код.

```
import numpy
from mod1 import *
x = numpy.array([i for i in range(20)])
print(x)
print(y_value)
print(_x_value)
```

Попробуйте закомментировать строку с __all__ и посмотреть, что получится.

Подчеркивание после имени переменной используется для отличия от ключевых слов Python. Например: class_, int_, len_, for_.

Попробуйте импортировать модуль mod1 без __all__.

```
import mod1
print(mod1._x_value)
```

Попробуйте импортировать модуль mod1 таким способом:

```
import mod1 as md
print(md._x_value)
```

Реализуйте примеры в консоли Python.

```
#Операции
>>> flag = True
>>> flag
True
>>> flag+True
2
>>> flag and False
False
>>> flag+False
1
# Возведение в степень
>>> x = 2
>>> x**10
1024
#Деление нацело
>>> x = 15//2
>>> x
7
>>> x = 15/2
>>> x
7.5
#Остаток от деления
>>> x = 15%2
>>> x
1
#Ввод строк
>>> str=input("Input any value: ")
Input any value: 45
>>> val=int(input("Input any value: "))
Input any value: 45
```

```
>>> str
'45'
>>> val
45

#Форматный вывод (Оператор % для форматирования) строка с форматом % значения
>>>print("%03d:%10s %04d \n" % (flag,str,val))
001:       45 0045
>>> print("%s:%15s %04d" % (flag,str,val))
True:            45 0045
>>> c=41
>>> f=56.12
# 15 или 20 в примере – сколько всего отводится под вывод вещественного числа
>>>print("%c %15.3f %020.5e" % (c,f,f))
)          56.120 0000000005.61200e+01

>>> str = "Hello world"
>>> len(str)
11

Метод формат
>>> s = "{}".format(str)
>>> s
'Hello world'

>>> s = "{0} {1} {2}".format(str,14.24,True)
>>> s
'Hello world 14.24 True'
>>> print("{1:10.2f} {2} {0}".format(str,14.24,True))
14.24 True Hello world



print("{0:^15}\n".format("*"),"{0:^14}\n".format("***"),"{0:^14}".format("*****"))
      *
     ***
   *****
>>> print("{0:^15}\n{1:^15}\n{2:^15}".format("*","***","*****"))
      *
```

```
   ***
  *****
>>> print("{0:^15}\n{1:^15}\n{2:^15}".format("*","*"*3,"*"*5))
     *
    ***
   *****

# f- строки Python 3.6+
```

join – склеивание списка в строку.

split разбиение строки на элементы списка.

```
>>> print("".join([f"{'*'*(i*2+1):^15}\n" for i in range(5)]))
     *
    ***
   *****
  *******
 *********
>>>x = 9
>>> print("".join([f"{'*'*(i*2+1):^{x*2-1}}\n" for i in range(x)]))

>>> treg = lambda x,y: "" if x==-1 else treg(x-1,y)+f"{'*'*(x*2+1):^{y}}\n"
print(treg(7,7*2+1))
```

Реализуйте программу для работы с графиками.

Для отображения графиков при запуске проекта в PyCharm произведите установку (если ваша операционная система Linux):

```
#sudo apt install libgirepository1.0-dev gcc libcairo2-dev pkg-config python3-dev gir1.2-gtk-3.0
sudo apt-get install pkg-config libcairo2-dev libgirepository1.0-dev
pip3 install pycairo
pip3 install PyGObject
```

Изучите внимательно код и что он делает.

```
import numpy as np
import matplotlib.pyplot as plt
from math import *
print("Hello world")
# константа обозначается заглавной буквой
N = 100
```

```
x = np.array([i for i in range(N)]) # массив из N элементов
x = x*pi/(N-1) # вектор умножаем на скаляр
# создаем сетку от 0 до pi включительно с количеством элементов N
x1 = np.linspace(0,pi,N) # используем для того же функцию
# В узлах сетки рассчитываем значения функций sin и cos
y = np.sin(x)
y1 = np.cos(x1)
ax = plt.gca() # ссылка на текущий объект axes
plt.plot(x,y) # создаем обычный график sin
plt.plot(x1,y1) # создаем обычный график cos
ax.set_xlabel("angle, rad")
ax.set_ylabel("sin, cos function")
plt.show()
x2 = x1 - x1.max()/2
y2 = x2*x2
plt.plot(x2,y2)
plt.show()
y=y1

x_rol = np.zeros(N)
# копируем срез со сдвигом
# данная операция нужна, если сетка неравномерная
x_rol[0:N-1] = x[1:N]
xdif = x_rol-x
xdif[-1]=0.0 #?? как еще можно решить эту проблему
integral_rectangle = (y*xdif).sum()
y_rol = np.zeros(N)
# для метода трапеций копируем в массив со сдвигом
# в данном случае это необязательно
# ведь можно реализовать сумму от 1 до N-1
# и добавить первый и последний элемент, так как
# промежуточные суммы все равно совпадают
# попробуйте это показать математически
y_rol[0:N-1] = y[1:N]
y[-1] = 0.0 #?? как еще можно решить эту проблему
y_sum = ((y+y_rol)*0.5*xdif).sum()
integral_trapeze = y_sum
print(f"integral by rectangle {integral_rectangle}")
print(f"integral by trapeze {integral_trapeze}")
```

```
# пример работы с модулем time для учета времени работы
import time
t = time.time()
y = []
N = 10000
z = []
# добавление значений функции в список
for i in range(N):
  y = y+[sin(i*pi*2/N)]
  z.append(y[i])
print(y)
print(z)
dt = time.time()-t
print(f'Time {dt}')
import numpy as np
import matplotlib.pyplot as plt
from math import *
print("Hello world")
# константа обозначается заглавной буквой
N = 100
x = np.array([i for i in range(N)]) # массив из N элементов
x = x*pi/(N-1) # вектор умножаем на скаляр
# создаем сетку от 0 до pi включительно с количеством элементов N
x1 = np.linspace(0,pi,N) # используем для того же функцию
# В узлах сетки рассчитываем значения функций sin и cos
y = np.sin(x)
y1 = np.cos(x1)
ax = plt.gca() # ссылка на текущий объект axes
plt.plot(x,y) # создаем обычный график sin
plt.plot(x1,y1) # создаем обычный график cos
ax.set_xlabel("angle, rad")
ax.set_ylabel("sin, cos function")
plt.show()
x2 = x1 - x1.max()/2
y2 = x2*x2
plt.plot(x2,y2)
plt.show()
y=y1
x_rol = np.zeros(N)
```

```
# копируем срез со сдвигом
# данная операция нужна, если сетка неравномерная
x_rol[0:N-1] = x[1:N]
xdif = x_rol-x
xdif[-1]=0.0 #?? как еще можно решить эту проблему
integral_rectangle = (y*xdif).sum()
y_rol = np.zeros(N)
# для метода трапеций копируем в массив со сдвигом
# в данном случае это необязательно
# ведь можно реализовать сумму от 1 до N-1
# и добавить первый и последний элемент, так как
# промежуточные суммы все равно совпадают
# попробуйте это показать математически
y_rol[0:N-1] = y[1:N]
y[-1] = 0.0 # ?? как еще можно решить эту проблему
y_sum = ((y+y_rol)*0.5*xdif).sum()
integral_trapeze = y_sum
print(f"integral by rectangle {integral_rectangle}")
print(f"integral by trapeze {integral_trapeze}")
# пример работы с модулем time для учета времени работы
import time
t = time.time()
y =[]
N = 10000
z = []
# добавление значений функции в список
for i in range(N):
  y = y+[sin(i*pi*2/N)]
  z.append(y[i])
print(y)
print(z)
dt = time.time()-t
print(f'Time {dt}')
```

Лучше оформить расчет интегралов в виде функций. Представим код реализующий расчет:

```
import time
import numpy
import math
```

```
# интеграл с использованием массивов numpy
def integral_np(x,f):
  return ((f[0:-1]+f[1:])*(x[1:]-x[0:-1])*0.5).sum()

# интеграл стандартным способом
def integral(x,f):
  s = 0.0
  for i in range(min(len(x),len(f))-1):
    s = s+(f[i+1]+f[i])*(x[i+1]-x[i])*0.5
  return s


N = 10000
# задаем функцию по которой ведется расчет
x = numpy.array([i*math.pi/N for i in range(N)])
f = numpy.sin(x)

# выводим значение интеграла и время расчета
t = time.time()
print(integral_np(x,f))
dt = time.time() - t
print(f"Время работы равно: {dt} ")

# выводим значение интеграла и время расчета
# расчет интеграла обычным способом
t = time.time()
print(integral(x,f))
dt = time.time() - t

print(f"Время работы равно: {dt} ")
```

1.6 Использование Python для работы с изображениями

Целью данного параграфа является знакомство с Google Colab и изучение основных возможностей numpy, простейших функций обработки изображений и использования графиков.

Изучение данного материала поможет вам освоить в дальнейшем современные библиотеки машинного обучения, библиотеки для работы с нейронными сетями на Python, а также реализовать веб-приложение по заданию.

Обычный текст, набранный черным шрифтом, следует выполнять последовательно, изучая основные возможности инструментария.

Текст, выделенный бирюзовым цветом, – для самостоятельного освоения и принятия к сведению.

Текст, выделенный оранжевым цветом, – для самостоятельного исполнения и реализации.

Текст, отмеченный звездочкой, – для тех, кому хочется выполнить дополнительное задание (опционально).

Зайдите на сайт https://colab.research.google.com/ (перед этим зарегистрируйтесь, если необходимо, в Google). Затем в меню переименуйте текущий NoteBook.

File → Rename

Назовите свой проект. Введите код в соответствующую ячейку для кода (рис. 1.11).

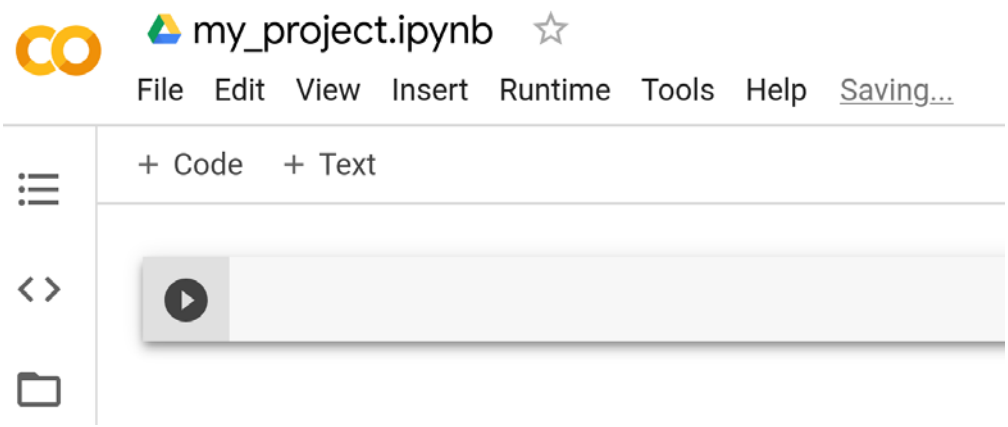

Рис. 1.11 – Ячейка Cell в Colab

В Runtime можно использовать Change Runtime type для смены вычислительного устройства на GPU или TPU. Выберите Python2 или Python3.

Нажав + Code, можно создать новую ячейку (Cell), запустить код или команды, в том числе запустить сам Python.

Можно установить непосредственно нужный пакет. Не забывайте перед командой ставить восклицательный знак (рис. 1.12).

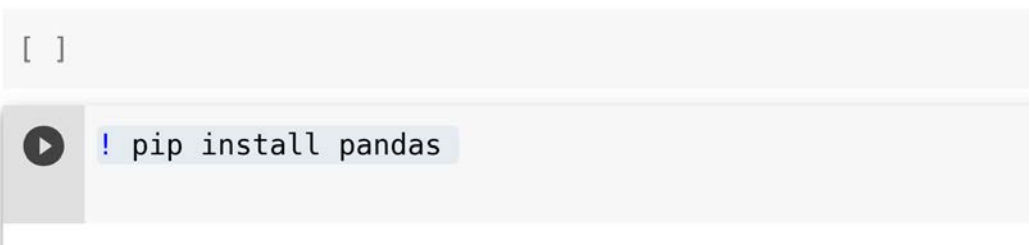

Рис. 1.12 – Запуск команды из командной строки для установки пакета

Можно клонировать какой-либо проект с github.

! git clone https://github.com/ ……

Для подключения диска Google можно воспользоваться командами, приведенными на рисунке 1.13.

Рис. 1.13 – Подключение Google Disk

Другой способ – нажмите иконку «папка» и затем Mount Drive (рис. 1.14).

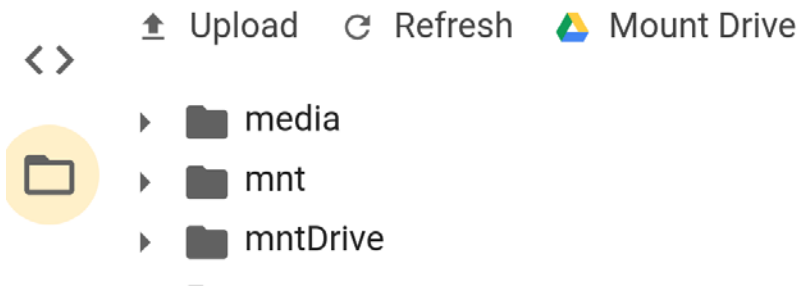

Рис. 1.14 – Подключение Google Disk

После запуска кода появится сообщение о необходимости ввода кода авторизации, для этого перейдите по указанной ссылке и разрешите сторонним компонентам получать доступ к вашему диску (рис. 1.15).

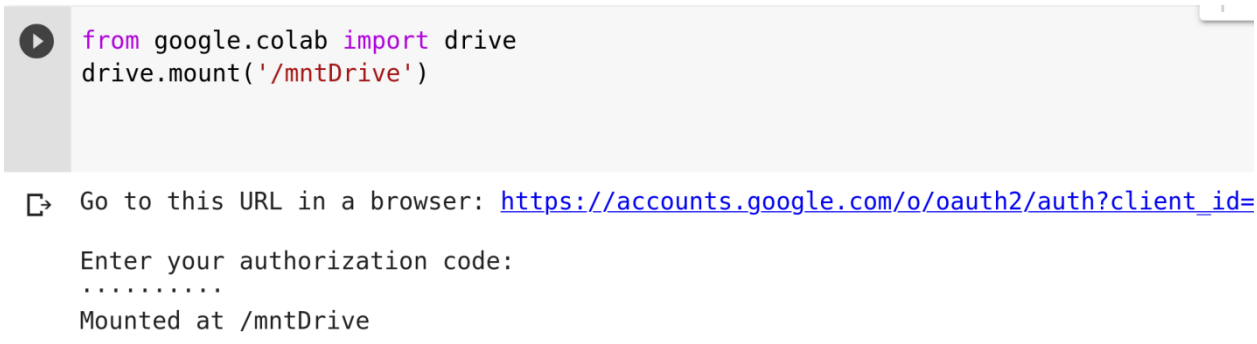

Рис. 1.15 – Результат подключения диска

На рисунке 1.16 приведен пример формы, из которой необходимо копировать код авторизации.

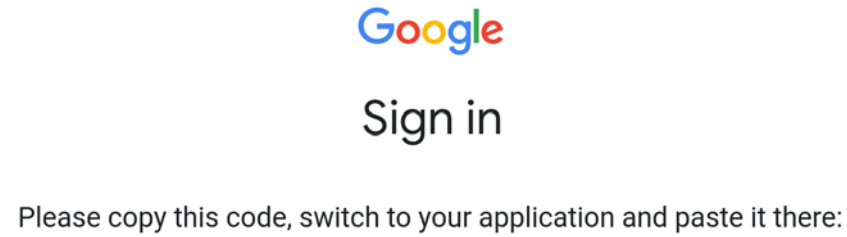

Рис. 1.16 – Форма для копирования ключа авторизации

Ниже должен быть указан код авторизации, который нужно скопировать в поле ввода.

Теперь можно обращаться к файлам на Google-диске.

Зайдем на свой Google-диск: https://drive.google.com, или из google.com.

Создадим директорию Python (название может быть другое). Можем теперь записать в нее нужные нам файлы (с помощью upload) или даже целую директорию (upload directory) (рис. 1.17).

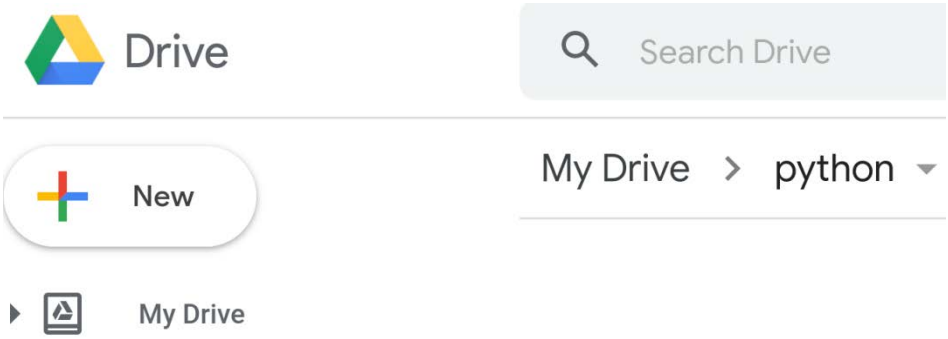

Рис. 1.17 – Диск с созданным каталогом

Можно вернуться на Colab, выбрать Download py и сохранить все Cells в файле на вашем рабочем компьютере.

На нашей рабочей машине создадим в любом удобном редакторе файл со скриптом на Python, например, с названием file1.py. Внимательно вводите

пути, чтобы не ошибиться, в конце поставьте слеш, если вдруг происходит ошибка.

```
import math
import numpy
x = numpy.array([i for i in range(100)])
print(x)
```

Запишем этот файл со скриптом на Google-диск.

Затем вернемся в colab cell и запустим наш скрипт (рис. 1.18).

```
! python3 /mntDrive/My\ Drive/python/file1.py

[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
```

Рис. 1.18 – Результат работы скрипта

Попробуем импортировать этот скрипт как модуль (рис. 1.19).

```
import sys
sys.path.append('/mntDrive/My Drive/python')
import file1

[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
```

Рис. 1.19 – Результат работы скрипта

Как видите, при подключении модуля его код исполнился. Вы можете написать свой скрипт или реализовать свой проект, затем сохранить его на Google-диск и запустить для исполнения на colab.

Вернитесь на Google-диск и в папке Python создайте папку dataf, скопируйте в эту папку какие-либо изображения в формате jpg, порядка 5 файлов

(используйте файлы, содержащие 3 цветовые гаммы, если использовать 4 или меньше, то необходимо будет изменить код самостоятельно).

Создайте еще одну ячейку + Code и в ней проводите эксперименты. Запишите туда данный код, внимательно изучите его, при необходимости проведите рефакторинг.

```
      import sys
      import os
      import numpy as np
      from PIL import Image
      import matplotlib.pyplot as plt
      sys.path.append('/mntDrive/My Drive/python')
      def read_image_files(files_max_count,dir_name):
        files = os.listdir(dir_name)
        if(files_max_count>len(files)): # определяем количество файлов,
не больше max
          files_count = len(files)
        else:
          files_count = files_max_count
        image_box = [[]]*files_count
        for file_i in range(files_count): # сохраняем изображения в список
          image_box[file_i] = Image.open(dir_name+'/'+files[file_i]) #  / ??
        return files_count, image_box

      files_count , image_box = read_image_files(10,'/mntDrive/My
Drive/python/dataf')
      print(files_count)
      fig = plt.figure(figsize=(15,15))
      plot_countx = int(files_count**0.5)+1
      plot_county = int(files_count**0.5)+1
      print(image_box)
      viewer = [[]]*files_count # массив саб графиков
      for i in range(files_count):
        viewer[i] = fig.add_subplot(plot_countx,plot_county,i+1)
        viewer[i].imshow(np.array(image_box[i])) # делаем график изображения
      fig.show()
```

Если все сделано правильно, должны быть отображены файлы с изображениями. Попробуйте изменить программу, создав более короткий или удобочитаемый код.

Попробуем изменить немного программу и преобразовать изображение, изменив его размер.

Какими другими более удачными названиями можно обозначить некоторые переменные?

```
import random
height = 224
width = 224
# копирование изображений в массив numpy и изменение их размеров
# попробуйте написать код, который сохраняет соотношение сторон
images_resized = [[]]*files_count
for i in range(files_count):
  images_resized[i] = np.array(image_box[i].resize((height,width)))/255.0
images_resized = np.array(images_resized)
# пример кода вместо кода выше,
# закомментировать код нежелательно, или оставлять
# закомментированный код, лучше сразу удалять
# np.array([np.array(image_box[i].resize((height,width))) for i in
# range(files_count)])
al = 0.3
image_ = images_resized[0]*al+images_resized[1]*(1-al)
fig_ = plt.figure(figsize=(7,7))
viewer_ = fig_.add_subplot(1,1,1)
viewer_.imshow(image_)
fig_.show()
```

Изучим некоторые возможности работы с массивами numpy.

Суть данного примера заключается в том, что исходное изображение представляет собой трехмерную матрицу, в которой сначала идет измерение по высоте, потом по ширине, далее – измерение, хранящее цветовые составляющие r, g, b. Фактически матрица хранит списки из трех элементов. С помощью операции суммирования и деления на 3 по третьей оси мы получаем гамму в серых цветах и фактически матрицу 224 на 224. Но такая матрица на графике

будет отображена как тепловой график, где малые значения будут близки к холодным синим цветам, а большие – к ярким красным. Но мы хотим отобразить серые цвета, тогда мы должны растиражировать нашу составляющую интенсивности три раза, для этого мы с помощью expand_dims расширяем опять массив до размерности 224*224*1, а затем с помощью repeat дублируем три раза каждую составляющую в матрице, таким образом получая тензор 224*224*3.

```
# суммируем цветовые составляющие трехмерного "тензора" 224*224*3
# суммирование по третьей оси, индекс 2
# результат матрица 224*224, где все суммы r, g, b поделены на 3
image_ = images_resized[0].sum(axis=2)/3.0
# расширяем измерения массива от 2 до 3
image_1 = np.expand_dims(image_,axis=2)
# повторяем матрицы 224 на 224 три раза, получая 224*224*3
image_2 = np.repeat(image_1,repeats=3,axis=2)
print(image_1.shape)
# задаем пространство для размещения графиков
fig_ = plt.figure(figsize=(7,7))
# отображаем первый график слева
viewer_1 = fig_.add_subplot(1,2,1
# отображаем второй график справа
viewer_2 = fig_.add_subplot(1,2,2)
viewer_1.imshow(image_)
viewer_2.imshow(image_2)
fig_.show()
```

Первый аргумент в методе figure задает пространственный размер по вертикали и горизонтали. В методе subplot первый аргумент задает количество клеток по вертикали, второй – количество по горизонтали, третий аргумент задает номер графика, который будет отображен в соответствующей клетке.

В numpy, как и со списками, можно использовать срезы, в этом случае различаются представления view и полные копии copy, простое присвоение переменной не создает никакой копии, просто переменная ссылается на тот же объект.

Поэкспериментируем.

```
# создаем представление массива, верхняя половина
image_ = images_resized[0,0:112,0:224]
# создаем полную копию
```

```
image_copy = images_resized[0].copy()
# создаем представление массива, нижняя половина рисунка
image_1 = images_resized[0,112:224,0:224]
# должен измениться и рисунок images_resized[0]
# поизменяйте третий срез
image_1[56:112,:,0:3] = 1.0
print(image_.shape)
fig_ = plt.figure(figsize=(7,7))
# здесь задаем четыре графика в клетках 2 на 2
viewer_1 = fig_.add_subplot(2,2,1)
viewer_2 = fig_.add_subplot(2,2,2)
viewer_3 = fig_.add_subplot(2,2,3)
viewer_4 = fig_.add_subplot(2,2,4)
# в созданных клетках отображаем графики
viewer_1.imshow(image_)
viewer_2.imshow(image_1)
viewer_3.imshow(images_resized[0])
viewer_4.imshow(image_copy)
fig_.show()
```

Примеры для ознакомления, которые необходимо создать.

```
image_1[56:112,:,0:3] = np.random.rand(56,224,3)
# меняем местами первую и вторую ось (транспонируем), поворачиваем картин-
ку
viewer_4.imshow(image_copy.transpose(1,0,2))
# задаем один оттенок белого
image_1[56:112,:] = np.expand_dims(np.random.rand(56,224),axis=2)
# эквивалентно вызову np.expand два раза
image_1[56:112,:] = np.random.rand(56)[:,np.newaxis,np.newaxis]
# сделайте золотым цветом
image_1[56:112,:] = np.random.rand(3)[np.newaxis,np.newaxis,:]
```

Соответственно, элементы массива с меньшим числом элементов копируются в массив с большим количеством элементов, дублируясь, но количество измерений, как и размерность, должно совпадать по соответствующей оси. Предполагается, что одиночные элементы дублируются.

Например:

image_1[56:112,:] = np.random.rand(224,3)[np.newaxis,:]

Дополнительная информация о слайсах:

Запомните, копирование x[0:-1] = x[0:-1] приведет к клонированию всего массива без последнего элемента. Для получения полного клона нужно указать, например, len(x), x[0:len(x)] = x[0:len(x)] либо x[0:]= y[0:], при этом x[-1] даст последний элемент массива, т. к. слайсы не захватывают последний указанный элемент диапазона.

Закрасьте нижнюю левую часть изображения золотым цветом.

Заполнить случайными значениями другие части рисунка, например центральную часть.

Сделайте цвет рисунка в оттенках красного. Вместо констант используйте переменные.

Сделайте заливку верхней правой части рисунка градиентной.

Реализовать функции, которые это делают.

*Необходимо изменить размер исходного изображения на произвольный, сохранив при этом исходное соотношение сторон. Пустые части изображения заполнить черным либо цветом точки, наиболее близкой к краю расширения.

Наложение на изображение шума, распределенного равномерно, и последующее сглаживание результата гауссовым фильтром.

```
import scipy.ndimage.filters as filt
# создаем копию изображения
image_copy1 = image_copy.copy()
# зашумляем изображение
image_rand = image_copy + (np.random.rand(*image_copy.shape)-0.5)*0.3
# нормируем зашумленное изображения в пределах от 0 до 1
image_rand  = (image_rand-image_rand.min()) / (image_rand.max()-
image_rand.min())
# транспонируем, выводя измерение цветовых карт на первое место
# 3*224*224
image_rand = image_rand.transpose((2,0,1))
# фильтруем гауссовым фильтром каждую карту
image_filt = np.array([filt.gaussian_filter(image_rand[i],1) for i in
range(3)])
# возвращаем исходный формат 224*224*3
image_filt = image_filt.transpose((1,2,0))
image_rand = image_rand.transpose((1,2,0))
```

```
viewer_2.imshow(image_copy)
viewer_3.imshow(image_rand)
viewer_4.imshow(image_filt)
fig_.show()
```

Можно использовать медианный фильтр. Фильтрация будет изучаться на системах цифровой обработки сигналов. Одна из простейших идей фильтров – это скользящее среднее и медианная фильтрация. Скользящее среднее предполагает получение среднего в некотором небольшом окне относительно точки, медианная – взятие центрального элемента среди упорядоченных в окне. Более сложные методы основаны на преобразовании Фурье, вейвлет-преобразовании и т. д.

```
filt.median_filter(image_rand[i],7)
```

Можно сохранить numpy массив на диск, в том числе на Google-диск.

```
np.save('/mntDrive/My Drive/python/img',image_filt)
```

Или загрузить:

```
arr = np.load('/mntDrive/My Drive/python/img.npy')
```

Кроме того, можно загружать или сохранять несколько массивов.

```
>>> np.savez('example_2', a, b, c)
>>> np.savez('example_2', a=a, b=b, c=c)
>>>
>>> ex_2 = np.load('example_2.npz')
```

Оцените ошибку фильтрации, используя исходное и зашумленное (сглаженное) изображение, обычный подход с помощью циклов и подход с применением массивов numpy. Наряду с равномерным шумом попробуйте гауссов шум randn.

Сохраните и посчитайте результаты, отобразите их в виде графиков или в текстовом виде, используя форматный вывод или f-строки.

*Попробуйте написать алгоритм, который реализует подбор фильтра по одному из его параметров, минимизирующему ошибку сглаживания для данной группы картинок (например, для изображения растений). Можно использовать случайный поиск, метод дифференциальной эволюции или генетический алгоритм.

Изменение размера изображения с помощью интерполяции.

```
import scipy.ndimage.interpolation as interp
fig_ = plt.figure(figsize=(7,7))
viewer_1 = fig_.add_subplot(2,2,1)
viewer_2 = fig_.add_subplot(2,2,2)
viewer_3 = fig_.add_subplot(2,2,3)
viewer_4 = fig_.add_subplot(2,2,4)
# меняем масштаб изображения
image_interp = interp.zoom(image_copy,(2,2,1))
# вращаем изображение на 45 градусов, вращаем оси 0,1
image_rot = interp.rotate(input=image_copy, angle=45, axes=(0,1),
reshape = False)
# с изменением исходного размера массива
image_rot_r = interp.rotate(input=image_copy, angle=45, axes=(0,1),
reshape = True)
# отображаем форму массива
print(image_interp.shape)
print(image_rot.shape)
print(image_rot_r.shape)
viewer_1.imshow(image_copy)
viewer_2.imshow(image_interp)
viewer_3.imshow(image_rot)
viewer_4.imshow(image_rot_r)
fig_.show()
```

Результат представлен на рисунке 1.20.

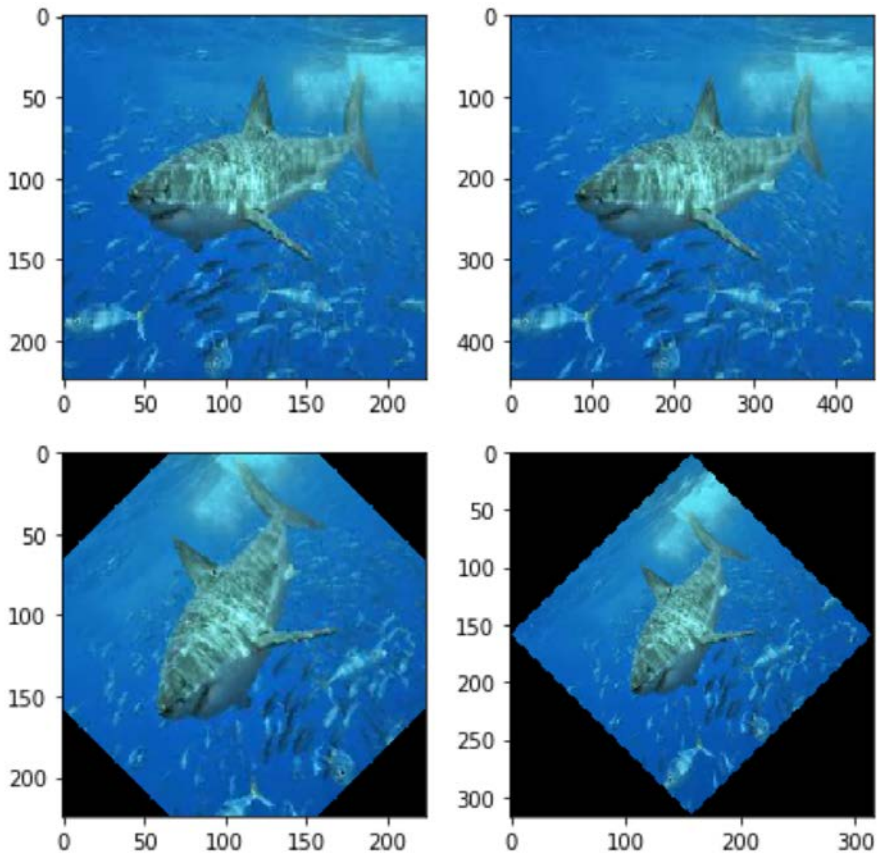

Рис. 1.20 – Результат работы скрипта поворота

В модуле interpolation также есть функции трансформации изображения с помощью матрицы поворота, масштабирования и сдвига. Есть возможность выбора порядка сплайновой интерполяции.

Наложите повернутую уменьшенную картинку на другую в виде водяного знака.

1.7 Возможности Python для обработки данных

В данном параграфе рассмотрены возможности доступа к сервисам, предоставляющим открытые данные; продемонстрированы способы обработки и визуализации данных на Python, некоторые возможности библиотек panda, matplotlib, seaborn; описан доступ к API сервиса data.gov.ru.

1.7.1 Примеры ресурсов с открытыми данными

В настоящее время проекты, связанные с публикацией различных открытых данных, очень популярны, есть множество ресурсов, представляющих возможности для поиска, публикации и анализа.

1. Проект data.world. Открытое сообщество data.world предназначено для тех, кто увлечен анализом данных. В коллекции доступно более 450 наборов данных практически для любых целей. Большинство из них требуют выполнения очистки, а очистка данных является важным этапом любого проекта по науке о данных. Наборы данных охватывают такие темы, как финансы, преступность, экономика, образование, перепись населения, образование, окружающую среда, энергетика, спорт, НАСА и многие другие.

2. Проект Kaggle. Одна из самых крупных Data Science платформ, с большим набором данных для анализа, проведения соревнований по обработке, с примерами проектов. В основном присутствуют чистые данные, в особенности те, которые выложены для проведения соревнований.

3. Проект Socrata OpenData. Наиболее мощная поисковая система, в которой есть возможность использования SQL-запросов. Включает данные по темам, связанным с финансами, инфраструктурой, транспортом, окружающей средой, экономикой и общественной безопасностью. Все наборы данных категоризированы с помощью алгоритмов машинного обучения. Помимо этого Discovery API от Socrata OpenData предоставляет способ получения доступа ко всем общедоступным данным с платформы. Еще одна отличительная особенность для разработчиков заключается в том, что вызовы API возвращают вложенные объекты JSON, которые легко понять и проанализировать.

4. Открытые государственные наборы данных: data.gov, data.gov.ru, data.gov.uk и т. д.

1.7.2 Обработка Kaggle датасетов

Kaggle предоставляет данные для анализа и NoteBook для написания кода на Python. Кроме того, проект проводит соревнования по обработке данных. Выбрав необходимые данные на сайте, можно получить сразу готовый код, в котором присутствует считывание csv-файла. При этом даже не нужно регистрироваться (хотя лучше это сделать, чтобы не потерять нужные вам результаты). Можно также добавлять markdown-текст (например, данный документ написан с использованием markdown).

Попробуйте проделать по шагам действия, указанные ниже. Если Вас смутила тематика датасета, можете попробовать найти свой вариант и выполнять действия по аналогии, но результат не гарантируется.

Переходим по ссылке Kaggle datasets (https://www.kaggle.com/datasets).

Нажимаем New Notebook.

Add Data (ищем temperature, рис. 1.21).

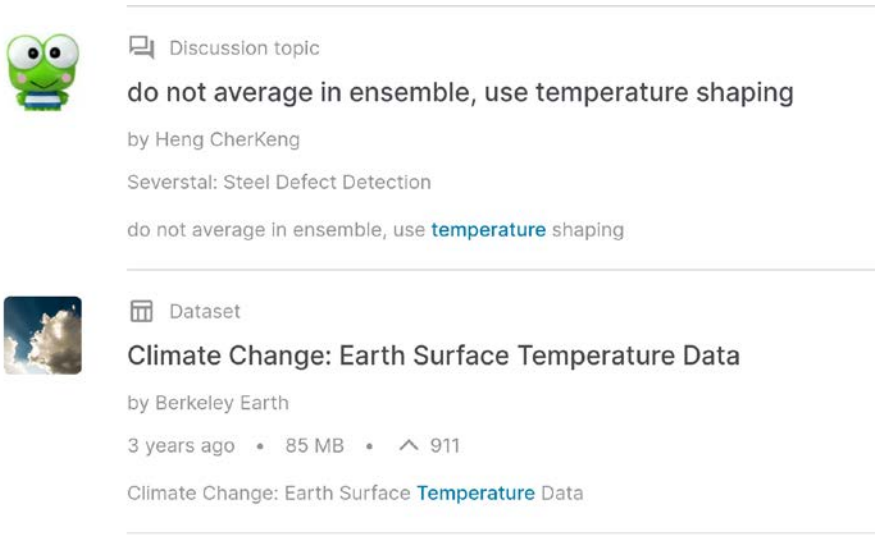

Рис. 1.21 – Выбор датасета по температуре

Выбираем "Climate Change: Earth Surface Temperature Data Exploring global temperatures since 1750".

Соглашаемся на добавление данных

В результате получаем две ячейки с кодом и можем сразу приступать к анализу данных.

Листинг

```
# This Python 3 environment comes with many helpful analytics
# libraries installed
# It is defined by the kaggle/python docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in
import numpy as np # linear algebra
# data processing, CSV file I/O (e.g. pd.read_csv)
import pandas as pd
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# Any results you write to the current directory are saved as output.
import pandas as pd
GlobalLandTemperaturesByCity = pd.read_csv("../input/climate-change-earth-surface-temperature-data/GlobalLandTemperaturesByCity.csv")
GlobalLandTemperaturesByCountry = pd.read_csv("../input/climate-change-earth-surface-temperature-data/GlobalLandTemperaturesByCountry.csv")
GlobalLandTemperaturesByMajorCity = pd.read_csv("../input/climate-change-earth-surface-temperature-data/GlobalLandTemperaturesByMajorCity.csv")
GlobalLandTemperaturesByState = pd.read_csv("../input/climate-change-earth-surface-temperature-data/GlobalLandTemperaturesByState.csv")
GlobalTemperatures = pd.read_csv("../input/climate-change-earth-surface-temperature-data/GlobalTemperatures.csv")
```

Печать таблицы на экран покажет нам столбцы (поля), которые содержит таблица, первые и последние строки таблицы.

```
print(GlobalLandTemperaturesByCountry)
print(GlobalLandTemperaturesByCity)
```

Здесь указаны данные о температуре по городам и странам начиная с 1700-х гг. Для таблицы с городами также указаны широта и долгота. Кое-где есть пропуски данных Nan.

Попробуем провести какой-либо анализ указанных данных.

```
df = GlobalLandTemperaturesByCity
print(df.describe())
```

Функция describe описывает для каждого поля DataFrame различные характеристики, включая среднее, среднеквадратическое отклонение, максимальное, минимальное значение, квантили. Запустите указанный код, который выведет на экран всю информацию о датасете, по данной информации уже можно сделать различные выводы, например о том, что разброс температур от –42 до +39 ºС. Этот факт вызывает некоторое сомнения, ведь в некоторых регионах бывают и куда меньшие и большие температуры. Максимальная ошибка для максимальных температур составляет до 15 °С, возможно, для данных, указанных для XVIII и XIX вв. Но это нужно выяснить.

Посмотрим, какие страны присутствуют в списке, получив только уникальные названия стран, и поместим их в список.

```
print(pd.unique(df['Country']).tolist())
print(f"Number of countries = {len(pd.unique(df['Country']).tolist())}")
```

В списке присутствует 159 различных стран.

Вероятно, здесь присутствуют не все страны. Попробуем уточнить данные по России. Заодно определим, какие столбцы данных присутствуют в таблице.

```
df = GlobalLandTemperaturesByCity
print(df.columns.tolist())
# получаем данные по России, | - или
with_russia = df[ (df['Country'] == 'Russia') |
(df['Country'] == 'USSR') |
(df['Country'] == 'Russian Federation')]
print(pd.unique(with_russia['Country']).tolist())
print(with_russia)
```

Столбцы, присутствующие в таблице:

```
['dt', 'AverageTemperature', 'AverageTemperatureUncertainty',
```

```
'City', 'Country', 'Latitude', 'Longitude']
```

Очевидно, что мы могли и ошибиться немного в названии, вдруг в базе названия стран указаны со строчной буквы.

```
with_dif = df[df['Country'].str.contains(r'(ru|sov|us)',case=False)]
print(pd.unique(with_dif['Country']).tolist())
```

При запуске придется подождать, ведь данных очень много.

Пытаемся получить все страны в именах, в которых присутствуют указанные символы, здесь мы использовали регулярное выражение.

['Russia', 'Australia', 'Peru', 'Belarus', 'Burundi', 'Austria', 'Cyprus', 'Uruguay', 'Mauritius']

Скорее всего, в базе не отражены изменения названий стран в разные периоды истории, а указано текущее положение дел.

Среди стран нет США (USA), возможно, указанное государство представлено под другим названием, проверим нашу догадку.

```
with_dif = df[df['Country'].str.contains(r'(united)',case=False)]
print(pd.unique(with_dif['Country']).tolist())
```

Догадка оказалась верной ['United Kingdom', 'United States', 'United Arab Emirates'].

Построим графики, на которых будет отображена средняя температура для каких-либо нескольких стран по годам. Очевидно, нам для этого потребуется группировка и агрегированная функция mean. Так как в данных присутствует дата проведения измерения, потребуется группировать и по году в поле «Дата», для этого есть различные варианты, мы можем даже создать новые столбы, в которые занесем отдельно день, месяц и год, но это необязательно.

```
df = GlobalLandTemperaturesByCity
# преобразуем строку в тип данных дата-время
df['dt'] = df['dt'].astype('datetime64[ns]')
# группируем по стране и по году в дате
df_by_dt = df.groupby([df['Country'],df['dt'].dt.year]).mean()
# убираем индексы, Country и dt становятся column
df_by_dt = df_by_dt.reset_index()
print(df_by_dt)
```

Конечно, это может быть неоптимально, логичнее было бы сначала выбрать страны, чтобы потом проводить групповые операции. Можете сделать это самостоятельно. Кроме того, можно использовать параметр groupby as_index = False (но аккуратно).

Выберем несколько стран и попробуем отобразить данные о них по годам. Очевидно, что в качестве среднего значения по стране за год будет выступать температура.

```
df = df_by_dt
# запишем более коротко условие по выбору стран
df_by_country = df[df['Country'].isin(['Russia', 'Italy', 'Canada'])]
print(df_by_country.columns.tolist())
print(df_by_country)
pd.unique(df_by_country['Country']).tolist()
```

Теперь преобразуем таблицу так, чтобы данные по странам оказались в отдельных столбцах для каждой страны.

```
# преобразуем таблицу так, чтобы столбцы опять стали индексами
# либо уберем reset_index() в коде выше
df_by_country = df_by_country.set_index(['Country','dt'])
dfun = df_by_country.unstack('Country')
# печатаем полученные столбцы
print(list(dfun))
```

Получим следующий результат:

```
[('AverageTemperature', 'Canada'), ('AverageTemperature', 'Italy'),
('AverageTemperature', 'Russia'), ('AverageTemperatureUncertainty', 'Canada'),
 ('AverageTemperatureUncertainty', 'Italy'),
('AverageTemperatureUncertainty', 'Russia')]
```

Теперь названия столбцов представляют собой кортеж. Нарисуем графики.

```
# выбираем только те столбцы, где есть столбец AverageTemperature
newcols = [item for item in dfun if item[0]=='AverageTemperature']
# строим новую таблицу из столбцов
dpop = dfun[newcols]
# выводим наш график
dpop.plot()
# выбираем только те столбцы, где есть столбец AverageTemperatureUncertainty
```

```
newcols = [item for item in dfun if item[0]=='AverageTemperatureUncertainty']
# строим новую таблицу из столбцов
dpop = dfun[newcols]
# выводим наш график
dpop.plot()
```

Пример графика представлен на рисунке 1.22.

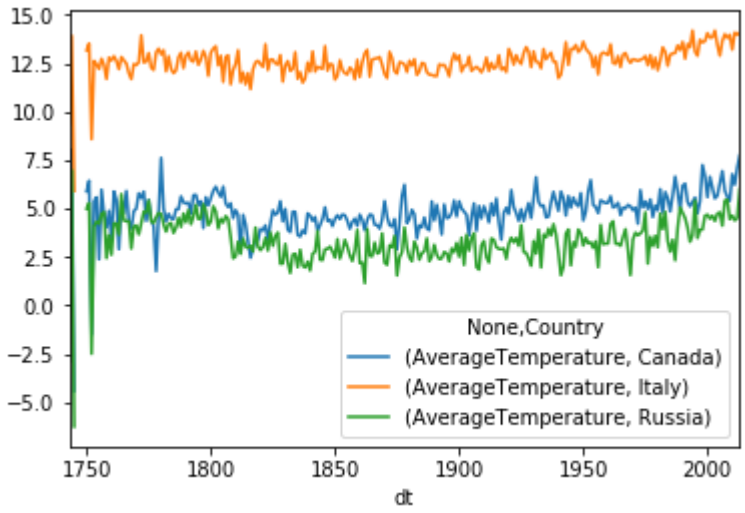

Рис. 1.22 – График со средней температурой

По графику можно сделать вывод, что после 1950 г. средняя температура во всех странах растет. Первые столетия измерения температур показывают некоторую «близость» температур в Канаде и России, что, возможно, связано с отсутствием измерений в холодных областях.

Пример графика с погрешностями измерения температуры за все годы представлен на рисунке 1.23.

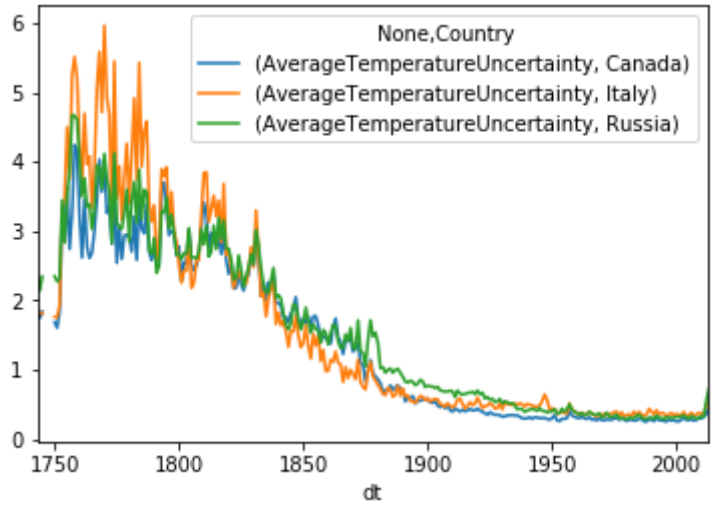

Рис. 1.23 – График погрешности в измерениях температуры

Как и ожидалось, точность указанной температуры со временем выросла и даже составляет менее 1 градуса.

Действительно, если запустить код, который размещен ниже, то обнаружим, что измерения для Томска присутствуют только с 1825 г. При группировке для столбца City мы получим значения False и True, в столбце False указаны сгруппированные данные по тем полям, где условие по Томску не сработало, поэтому эти данные можно отобразить отдельно или отбросить (рис. 1.24).

```
df_by_dt = df.groupby([df['City']== 'Tomsk',df['dt'].dt.year]).mean()
df_by_dt = df_by_dt.reset_index()
print(df_by_dt)
df_by_dt[df_by_dt["City"]==True].plot(x= 'dt', y = 'AverageTemperature')
df_by_dt[df_by_dt["City"]==False].plot(x= 'dt', y = 'AverageTemperature')
dfTomsk = df[df['City'] == 'Tomsk']
print(dfTomsk)
```

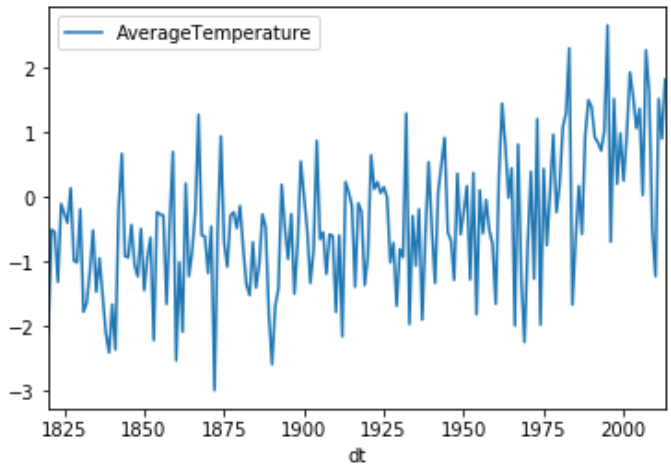

Рис. 1.24 – Пример графика средней температуры по городу Томску

Очевидно, что логичнее было бы сделать то же самое следующим образом:

```
df_by_dt = df[df['City']== 'Tomsk'].groupby(df['dt'].dt.year).mean()
df_by_dt = df_by_dt.reset_index()
print(df_by_dt)
df_by_dt.plot(x= 'dt', y = 'AverageTemperature')
```

На графике после 1950 г. виден некоторый тренд роста температуры.

Отобразим таким же образом и наши предыдущие графики для разных стран, используя построение графиков seaborn и выбор по условию.

```
df = GlobalLandTemperaturesByCity
df['dt'] = df['dt'].astype('datetime64[ns]')
df_by_dt = df.groupby([df['Country'],df['dt'].dt.year]).mean()
df_by_dt = df_by_dt.reset_index()
df = df_by_dt
df_by_country = df[df['Country'].isin(['Russia', 'Italy', 'Canada'])]
print(df_by_country.columns.tolist())
print(df_by_country)
pd.unique(df_by_country['Country']).tolist()
# преобразуем таблицу так, чтобы столбцы опять стали индексами
# либо уберем reset_index() в коде выше
df_by_country = df_by_country.set_index(['Country'])
import seaborn
# Нарисует среднее среди всех
# с максимальными и минимальными отклонениями
```

```
seaborn.lineplot(x='dt',y='AverageTemperature', data= df_by_country)
df_by_country = df_by_country.reset_index()
import matplotlib.pyplot as plt
# создаем два подграфика, расположенных по вертикали.
# axv – осевая система для отображения одного графика
# здесь две осевые системы, axv[0] и axv[1] для
# двух подграфиков
fig, axv = plt.subplots(nrows = 2,ncols = 1,figsize=(6, 7))
countries = pd.unique(df_by_country['Country']).tolist()
for country in countries:
    # выбираем страну для отображения графиков на одной системе axv[0]
    df_by_country[df_by_country['Country']==country].plot(
    x = 'dt', y = 'AverageTemperature',
    ax = axv[0], label = country)
    # выбираем страну для отображения графиков на другой системе axv[1]
    df_by_country[df_by_country['Country']==country].plot(
    x = 'dt', y = 'AverageTemperatureUncertainty',
    ax = axv[1], label = country)
fig.show()
```

Получим примерно то же самое, что и в прошлый раз, только теперь графики имеют подписи для стран и совмещены на одном поле (рис. 1.25, 1.26).

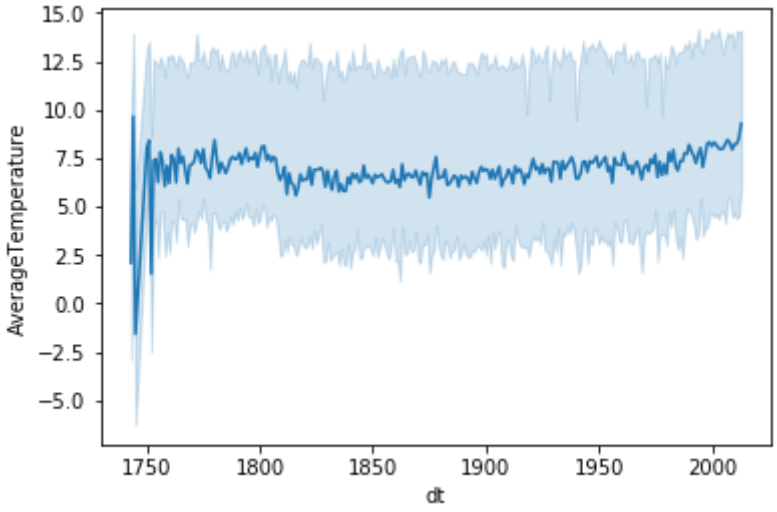

Рис. 1.25 – Средняя температура на графике seaborn

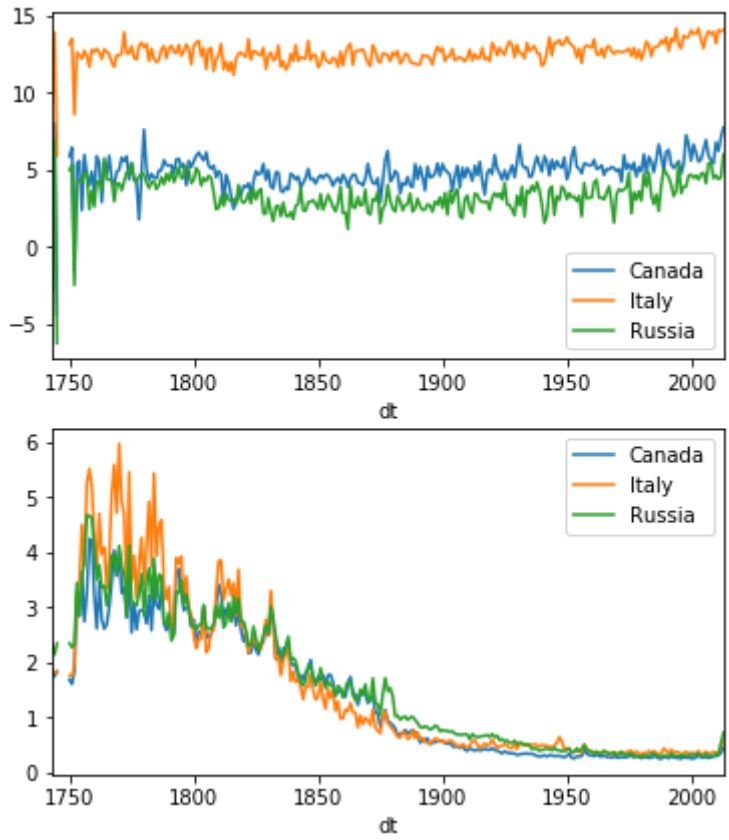

Рис. 1.26 – Построение двух графиков вместе

Полезный график pairplot в seaborn позволяет посмотреть соотношение данных попарно по отношению друг к другу (рис. 1.27).

```
df = GlobalLandTemperaturesByCity
df['dt'] = df['dt'].astype('datetime64[ns]')
df = df[df['City']=='Tomsk']
df_by_dt = df.groupby([df['dt'].dt.year]).mean()
df = df_by_dt.reset_index()
import seaborn
seaborn.pairplot(df)
```

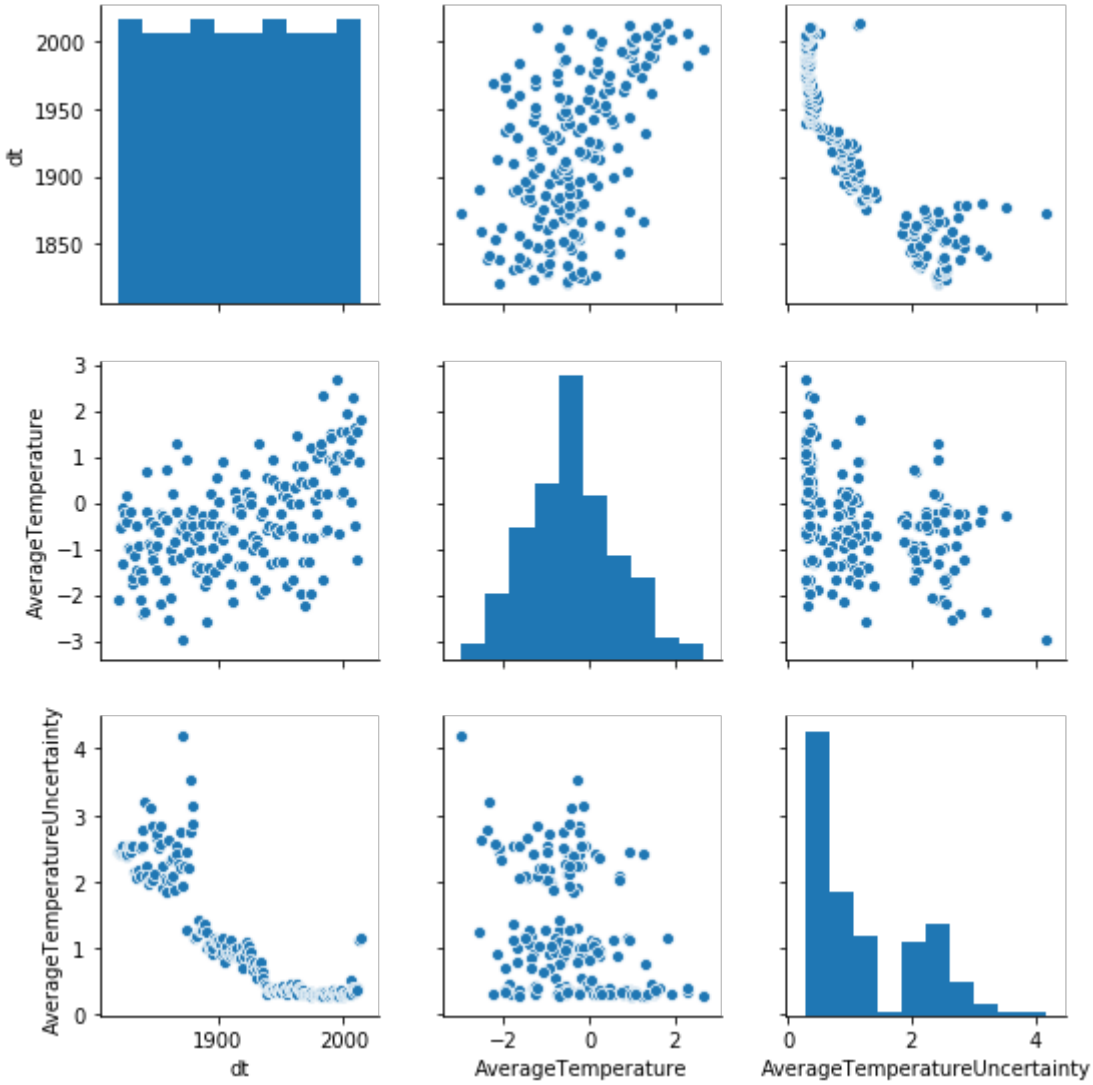

Рис. 1.27 – График pairplot

Например, очевидно, что температура в Томске в среднем с годами выросла, неопределенность в определении температуры с годами снизилась. А вот средняя температура в Мехико выросла еще больше, и тренд куда более четкий (рис. 1.28).

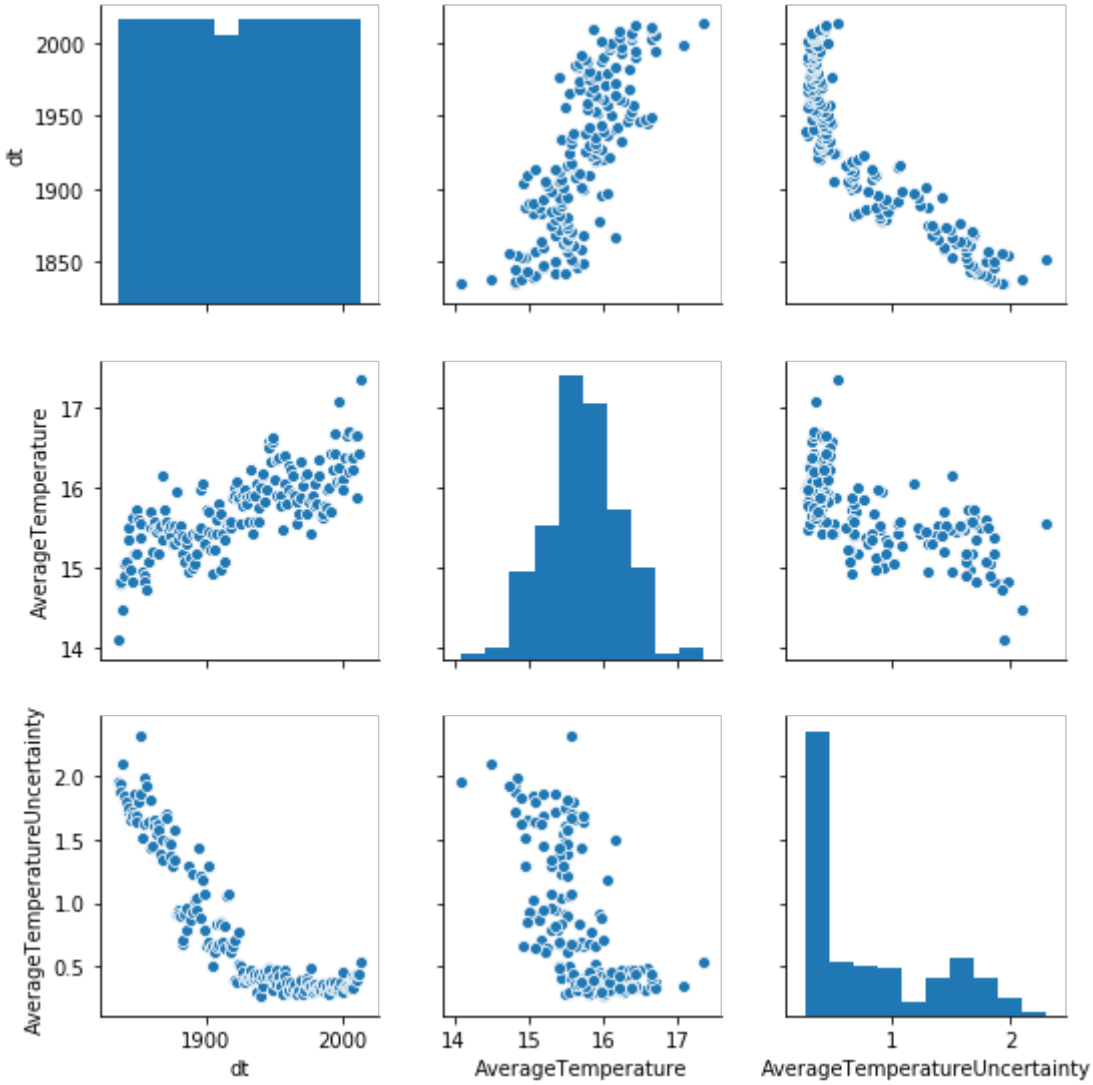

Рис. 1.28 – График pairplot для города Мехико

Еще один пример обработки данных на Kaggle datasets (рис. 1.29).

Рис. 1.29 – Выбор датасета

Нажимаем New Notebook.

Add Data (ищем suicide)

Выбираем Suicide rates Overview 1985 to 2016.

Соглашаемся на добавление данных

В результате получаем две ячейки с кодом и можем сразу приступать к анализу данных.

Листинг

```
# This Python 3 environment comes with many helpful analytics
# libraries installed
# It is defined by the kaggle/python docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in
import numpy as np # linear algebra
# data processing, CSV file I/O (e.g. pd.read_csv)
import pandas as pd
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# Any results you write to the current directory are saved as output.
import pandas as pd
master = pd.read_csv(
"../input/suicide-rates-overview-1985-to-2016/master.csv")
```

Печать таблицы на экран покажет нам столбцы (поля), которые содержит таблица, первые и последние строки таблицы.

```
print(master)
```

В таблице указаны пол и возраст людей, годы, страна, население, количество случаев на 100 тыс. населения, GDP (Gross domestic product – внутренний валовый продукт), HDI (Human Development Index – Индекс развития человеческого капитала, разработан ООН для оценки качества жизни людей). Кроме того, где-то есть пропуски данных Nan.

Попробуем провести какой-либо анализ указанных данных.

```
print(master.describe())
```

Для каждого поля DataFrame описаны различные характеристики, включая среднее, среднеквадратическое отклонение, максимальное, минимальное значение, квантили. Посмотрев на характеристики поля «Количество случаев на 100 тыс. населения», можно сделать простые выводы о среднем количестве случаев, о максимальном и минимальном. И сразу возникает простой вопрос, каково население страны, если максимальное – 43 млн. Посмотрим, какие страны присутствуют в списке, получив только уникальные названия стран, и поместим их в новый список.

```
print(pd.unique(master['country']).tolist())
```

Хотя здесь явно присутствуют не все страны, сразу возникает сомнение, так как, например, население Японии в 2019 г. составляло порядка 125 млн. Попробуем уточнить данные, которые имеются по Японии.

```
df = master
print(df[df['country'] == 'Japan'])
```

И действительно, данные содержат отдельные строки по полу и возрастам. Таким образом, при вызове describe мы получили максимальное количество людей какого-то возраста в какой-то стране, какого-то пола в каком-то году.

Попробуем получить данные по населению стран. Для этого нам понадобится осуществить группировку данных по какому-либо полю (по стране и году) и выполнить агрегированную функцию, в данном случае суммы. Таким образом, мы получим информацию по случаям в данной стране за данный год, по всем группам населения независимо от пола и возраста. Следует учесть, что тогда поле, которое считалось, как отношение случаев на 100 тыс. населения, будет уже неверным, так как оно соотносилось с количеством людей в определенной группе.

```
# получить данные с отсутствием повторения по группированному полю
# Группировка по странам и годам, суммирование
# поле как индекс
dfcy = df.groupby(['country','year'],as_index=True).sum()
print(dfcy)
print(dfcy.describe())
# получить данные с повторениями по группированному полю
# в стиле SQL
dfcy = df.groupby(['country','year'],as_index=False).sum()
print(dfcy)
print(dfcy.describe())
# получить данные по Японии за все годы
print(dfcy[dfcy['country']=='Japan'])
```

Попробуем построить графики так, чтобы на них отображалась информация по годам о количестве населения в каких-либо странах.

```
dfcy = df.groupby(['country','year'],as_index=False).sum()
# выбираем страны из сгруппированной таблицы
dfj = dfcy[dfcy['country'].isin(['Japan','Uzbekistan', 'Russian Federation'])]
# устанавливаем в качестве индекса (первичного ключа) страну и год
# чтобы можно было преобразовать таблицу со столбцами
# по Узбекистану и Японии отдельно
dfj = dfj.set_index(['country','year'])
# печатаем полученную таблицу
```

```
print(dfj)
# преобразуем таблицу так, чтобы были отдельные столбцы по странам
dfun = dfj.unstack('country')
# печатаем полученные столбцы
print(list(dfun))
```

Вот такие получились у нас имена столбцов. Как видите, они являются словарями.

```
[('suicides_no', 'Japan'), ('suicides_no', 'Uzbekistan'),
('population', 'Japan'), ('population', 'Uzbekistan'),
('suicides/100k pop', 'Japan'), ('suicides/100k pop', 'Uzbekistan'),
('HDI for year', 'Japan'), ('HDI for year', 'Uzbekistan'),
('gdp_per_capita ($)', 'Japan'), ('gdp_per_capita ($)', 'Uzbekistan')]
```

Если изучить таблицу, то обнаружим, что в ней отсутствуют некоторые данные, они обозначены как Nan. Если вы уже изучали базы данных, вам это будет понятно.

Теперь попробуем отобразить полученную таблицу на графике. Очевидно, если ее отобразить как dfun.plot(), на графике будут присутствовать как данные по населению, так и различные данные по случаям и внутреннему валовому продукту двух стран. Нам нужны данные только по населению.

```
# получаем список имен столбцов таблицы
newdf = list(dfun)
# выбираем только те столбцы, где есть столбец население
newcols = [item for item in newdf if item[0]=='population']
# строим новую таблицу из столбцов с населением Узбекистана
# и Японии
dpop = dfun[newcols]
# выводим наш график
dpop.plot()
```

Получаем два графика (рис. 1.30).

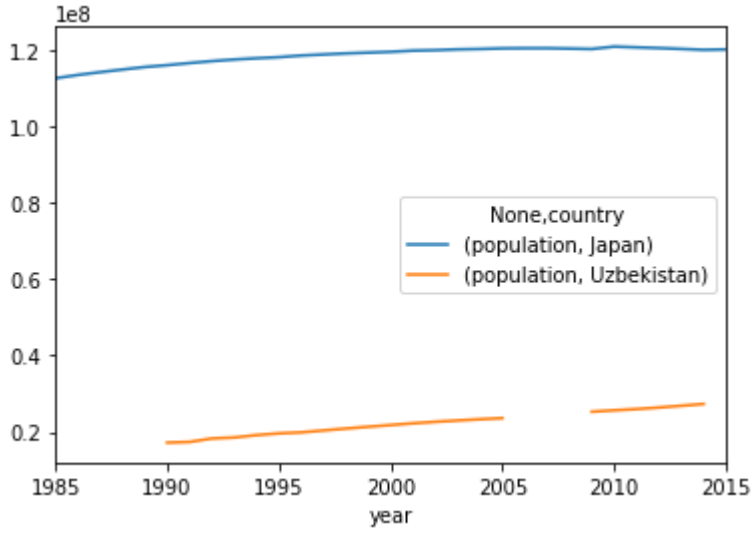

Рис. 1.30 – График случаев по годам для двух стран

Попробуем сделать более сложный анализ, например, коррелируют ли происходящие случаи с каким-либо индексом человеческого развития или внутренним валовым продуктом (приложение А, рис. А.1–А.3).

```
# выводим наш график
%matplotlib inline
import matplotlib.pyplot as plt
# как будут расположены графики на панно, в ячейках 2*2
# размер графиков
fig, axv = plt.subplots(nrows = 2,ncols = 2,figsize=(10, 10))
# пространство между графиками
plt.subplots_adjust(hspace = 0.5)
plt.subplots_adjust(wspace = 0.5)
# способ определения соотношения сторон
axv[0][0].set_aspect(aspect='auto')
axv[1][0].set_aspect(aspect='equal')
axv[1][1].set_aspect(aspect='equal')
# график по населению
dpop.plot(ax=axv[0][0])
import seaborn as sns
print(list(df))
corr = df.corr()
corr1 = dfcy.corr()
# цветовая карта корреляций
sns.heatmap(corr,ax = axv[1][0])
```

```
sns.heatmap(corr1,ax = axv[1][1])
# пары соотношений данных по полу и возрастам
sns.pairplot(data=df,hue='sex')
sns.pairplot(data=df,hue='age')
```

По графикам, представленным на рисунках можно сделать вывод, что у мужчин случаев самоубийств на 100 тыс. населения больше, чем у женщин. Рост ВВП уменьшает количество случаев, но при этом средний уровень развития человеческого капитала приводит к печальным последствиям чаще, а вот низкий или высокий, наоборот. Интересно было бы сравнить с уровнем убийств. Количество суицида среди населения старшего возраста выше, чем у молодых. Начиная с 1980-х гг. население старше 35 лет растет, а с 2005 г. начинает снижаться, оно составляет большую часть населения представленных стран.

Попробуем проанализировать страны по количеству случаев на 100 тыс. населения. Как было сказано выше, использовать непосредственно поле, присутствующее в таблице, уже нельзя, мы его агрегировали.

Таким образом, применим два подхода, с использованием seaborn и «в лоб», будем выводить упорядоченные усредненные данные по десяткам лет. Используем поле suicides_no, указывающее на количество случаев за год в данной стране, начиная с 1986 до 2016 г. (приложение А, рис. А.4).

```
%matplotlib inline
import matplotlib.pyplot as plt
# как будут расположены графики на панно, в ячейках 2*2
# размер графиков
fig, axv = plt.subplots(nrows = 3,ncols = 1,figsize=(20, 20))
# пространство между графиками
plt.subplots_adjust(hspace = 0.7)
plt.subplots_adjust(wspace = 0.5)
# способ определения соотношения сторон
axv[0].set_aspect(aspect='auto')
axv[1].set_aspect(aspect='auto')
axv[2].set_aspect(aspect='auto')
# график по населению
dpop.plot(ax=axv[0])
```

```
import seaborn as sns
print(list(df))
# определяем несколько DataFrame с усреднением по годам
# в десять лет
dfcl=[]
dfcl.append(dfcy[(dfcy['year']>=1980) & (dfcy['year']<1990)])
dfcl.append(dfcy[(dfcy['year']>=1990) & (dfcy['year']<2000)])
dfcl.append(dfcy[(dfcy['year']>=2000) & (dfcy['year']<2010)])
dfcl.append(dfcy[(dfcy['year']>=2010) & (dfcy['year']<2020)])
dfall = []
# цикл, в котором создаем группированные по странам выборки
for i in range(4):
    dfcymean = dfcl[i].groupby('country',as_index=False).mean()
    # делим количество случаев на население
    dfcymean['suon1'] = \
    dfcymean['suicides_no'].divide(dfcymean['population'],
                                   fill_value=0.0)
    # умножаем на 100000, чтобы получить количество
    # случаев на 100 тыс.
    dfcymean['suon1'] = dfcymean['suon1'] *1e+5
    # добавляем в список фреймов
    dfall.append(dfcymean)
# объединяем фреймы между собой
res = pd.concat(dfall,keys=['if1', 'if2', 'if3', 'if4'])
# поворачиваем таблицу, чтобы условия были в столбцах
resv = res.unstack(level=0)
# сортируем по последнему десятку лет и по количеству
# случаев за год
resv = resv.sort_values(by=[('suon1','if4')], ascending=False)
# создаем скаттер графики на которых будут
# отображены случаи по убыванию по странам
ax1 = sns.scatterplot(x = ('country','if4'), y = ('suon1','if4'),
                      data=resv,   ax = axv[1],label="2010-2020")
sns.scatterplot(x = ('country','if3'), y = ('suon1','if3'),
                data=resv,   ax = axv[1],label="2000-2010")
sns.scatterplot(x = ('country','if2'), y = ('suon1','if2'),
                data=resv,   ax = axv[1],label="1990-2000")
sns.scatterplot(x = ('country','if1'), y = ('suon1','if1'),
```

```
            data=resv,   ax = axv[1],label="1980-1990")
# вращаем подписи к графику
for item in ax1.get_xticklabels():
    item.set_rotation(90)
# попробуем сделать примерно то же самое парой строчек кода
# сортируем по количеству случаев DataFrame
# с количеством случаев на 100 тыс. населения
res = res.sort_values(by = 'suon1', ascending=False)
# строим график с отклонениями от среднего
ax2 = sns.lineplot(x='country',y='suon1', data = res,
                   sort = False,
                   ax = axv[2])
for item in ax2.get_xticklabels():
    item.set_rotation(90)
```

Например, по указанным на рисунке А.4 графикам можно сделать вывод, что в России ситуация в 1990–2000-е гг. ухудшилась, как и во многих других странах бывшего СССР, а с 2010 г. начала улучшаться.

1.7.3 Изучение возможностей доступа к API сервиса data.gov.ru

Обычно подобные data.gov.ru сервисы предоставляют API доступ и передачу данных приложению клиенту в форматах json, xml, csv, по протоколу HTTP.

Получить информацию по предоставляемым возможностям можно, например, здесь: Руководство по порталу data.gov.ru.

Зарегистрируйтесь на сайте https://data.gov.ru/. Мировым аналогом является, например, https://data.world/.

На странице https://data.gov.ru/get-api-key получите ключ API и скопируйте его в приложение Colab.

Строка запроса для получения данных с учетом полученного токена:

https://data.gov.ru/api/json/dataset/?access_token=YOUR_TOKEN_REPLACE&search=2019.

Для получения доступа к API воспользуемся компонентом http.client, позволяющим выполнять HTTP-запросы и получать ответы.

Используем основной тип запроса для получения данных с сервера GET, далее указываем URL и протокол с версией. Затем идут заголовки запроса.

С помощью GET можно и передавать данные на сервер, после URL ставится знак ? затем через амперсанд (&) идут поля атрибут = значение.

В ответ возвращается код ответа, 1xx, 2xx, 3xx, 4xx, 5xx, потом заголовки и тело ответа. Коды ответов: 1 – информационные, 2 – успешные, 3 – перенаправление, 4 – ошибки клиента, 5 – ошибки сервера.

Формат заголовков, имя заголовка: значение.

Например:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
 Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
<html>
<body>
<h1>Hello, World!</h1>
</body>
 </html>
```

Наш код.

```
import http.client as cl
import ssl
import json
conn = cl.HTTPSConnection('data.gov.ru', timeout=5,
                          context=ssl._create_unverified_context())
conn.request("GET", \
"https://data.gov.ru/api/json/dataset/?access_token=YOUR_TOKEN&search=2019")
resp = conn.getresponse()
print(resp.status, resp.reason)
print(resp.getheaders())
```

Теперь можно попытаться проанализировать различные датасеты.

```
data = resp.read().decode()
# Можно считать и пропарсить возвращенные данные в объект json.
```

```
#js_data = json.load(resp)
# Но мы скачаем список датасетов в объект panda DataFrame.
dt = pandas.read_json(data,typ='frame')
print(dt)
# Выводим первые 1024 байта скачанного датасета.
print(data[0:1024])
```

Пример полученных датасетов в формате json.


```
[
  {
    "identifier": "7017069388-obincstmsopprone",
    "title": "Объем инвестиций в основной капитал за счет всех источников финансирования (без субъектов малого предпринимательства и объемов инвестиций, не наблюдаемых прямыми статистическими методами) – всего – в сопоставимых ценах (ПРОГНОЗ 1 вариант)",
    "organization": "7017069388",
    "organization_name": "Администрация Томской области",
    "topic": "Government"
  },
  {
    "identifier": "7017069388-rgortsopprtwo",
    "title": "Раздел G: Оптовая и розничная торговля; ремонт автотранспортных средств, мотоциклов, бытовых изделий и предметов личного пользования – в сопоставимых ценах (ПРОГНОЗ 2 вариант)",
    "organization": "7017069388",
    "organization_name": "Администрация Томской области",
    "topic": "Government"
  },
```


Данных довольно много, поэтому сохраним их, а потом будем читать с диска. Используем формат HDF (Hierarchical Data Format – иерархический формат данных). Это формат файлов, созданный для хранения большого объема цифровой информации. Первоначально был разработан Национальным центром суперкомпьютерных приложений, сейчас поддерживается некоммерческой организацией HDF Group.

HDF5 – современная версия формата. Содержит иерархию из двух основных типов объектов.

Пример структуры HDF Datasets – наборы данных, многомерные массивы объектов одного типа Groups (группы), являются контейнерами для наборов данных и других групп.

Содержимое файлов HDF5 организовано подобно иерархической файловой системе, и для доступа к данным применяются пути, сходные с POSIX-синтаксисом, например, /path/to/resource. Метаданные хранятся в виде набора именованных атрибутов объектов.

```
path_file = "mntDrive/My Drive/python/dataf/dataset.hdf5"
dt.to_hdf(path_file, key='datasets_2019', mode='w', format='table')
```

Формат table в отличие от fixed позволяет работать с различными функциями поиска, использовать смешанные типы, при этом он более медленный. Для нашего примера с разными типами fixed работать не будет, поэтому мы использовали table. Можно, кроме того, добавлять в один и тот же файл разные датафреймы (DataFrame).

```
# Dt_frame.to_hdf(file_name, key='key_name', mode='a',append= True)
```

В новом cell считаем наш DataFrame в формате hdf.

```
import pandas
fname = 'mntDrive/My Drive/python/dataf/dataset.hdf5'
dt = pandas.read_hdf(fname, key = 'datasets_2019')
print(dt)
```

Отфильтруем получены данные, найдем информационные источники по студентам.

```
dt_filt  = dt[dt['title'].str.contains(r'студент')]
print(dt_filt['title'])
print(dt_filt[['title','identifier']])
```

Например:

4057 Принято студентов в государственные и муниципа...

7708234640-fourafouratwoasixaeight

Структура запроса данных требует указывать версию датасета, потому получим версию:

```
conn = cl.HTTPSConnection('data.gov.ru', timeout=5,
                          context=ssl._create_unverified_context())
conn.request("GET",
```

```
"https://data.gov.ru/api/json/dataset/7708234640-
fourafouratwoasixaeight/version?access_token=ваш токен")
resp = conn.getresponse()
data = resp.read().decode()
print(data[0:2024])
```

Версия

```
[ {   "created": "20180215T172910"  }]
```

Как указано в руководстве API, можно узнать структуру набора данных. Структура запроса:

«/api/<format>/dataset/<dataset>/version/<version>/structure»

Пример запроса:

/api/json/dataset/7710474375-perechenpodved/version/20131219T130000/structure

К сожалению, ресурс оказался довольно-таки сырым, поэтому получаем пустой ответ. Возможно, нам запрещено получение этих данных.

Разберите код, приведенный ниже. Попытайтесь структурировать его, выделив общие функции.

```
import http.client as cl
import ssl
import json
import pandas
import pandas
fname = 'mntDrive/My Drive/python/dataf/dataset.hdf5'
dt = pandas.read_hdf(fname, key = 'datasets_2019')
#print(dt)
#dt_filt  = dt[dt['title'].str.contains(r'студент')]
#print(dt_filt[['title','identifier']])
conn = cl.HTTPSConnection('data.gov.ru', timeout=5,
                          context=ssl._create_unverified_context())
for i in range(20,100):
  print("identifier ",dt['identifier'].iloc[i])
  print("title ",dt['title'].iloc[i])
  # задаем запрос на получение идентификатора версии, который будем использовать
  # при получении структуры и контента датасета
  request = "https://data.gov.ru/api/json/dataset/"
  request = request + dt['identifier'].iloc[i]+\
```

```
                "/version/?access_token=your_token"
conn.request("GET", request)
resp = conn.getresponse()
# берем json содержимой ответа
js_data = json.load(resp)
print("version ",js_data[0]['created'])
ident_v = js_data[0]['created']
# задаем строку запроса для получения структуры датасета
request = "https://data.gov.ru/api/json/dataset/"
request = request + dt['identifier'].iloc[i]+"/version/"
request = request+ ident_v+"/structure?access_token=your_token"
conn.request("GET", request)
resp = conn.getresponse()
js_data = json.load(resp)
# если при получении структуры данные не пустые, выводим данные о структуре
if js_data:
  print(js_data)
  # делаем запрос на получение данных в формате xml
  request = "https://data.gov.ru/api/xml/dataset/"
  request = request + dt['identifier'].iloc[i]+"/version/"
  request = request+ ident_v+"/content?access_token=your_token"
  conn.request("GET", request)
  resp = conn.getresponse()
  data_content = resp.read().decode()
  print(data_content[0:24])
  # попробуем обработать наши данные и парсить xml строку
  import pandas as pd
  import xml.etree.ElementTree as et
  import xml.dom.minidom
  # используем стандартный парсер Python, есть и сторонние
  # Два основных парсера DOM и sax. Почитайте, что это.
  parser = et.XMLParser(encoding="utf-8")
  xroot = et.fromstring(data_content, parser=parser)
  # ищем в xml теги, используя структуру, предлагаемую API
  print(xroot.tag)
  for rows in xroot.findall('rows'):
    print("rows")
    for row in rows.findall('row'):
```

```
        print("row")
        for value in row.findall('value'):
          print(value.text)
    conn.close()
```

Как видите, приложение крашится при первом же запуске. Тем не менее, оно выводит некоторую полезную информацию. Во-первых, многие датасеты отсутствуют, формат некоторых датасетов оставляет желать лучшего. Тем не менее, некоторые даже содержат XML-данные и некоторые из них даже читаются. Таким образом, здесь полезной оказалась бы обработка исключений.

Для ознакомления с исключениями на Python обратитесь к ресурсу: https://pythonworld.ru/tipy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html .

Попробуйте переписать вышестоящий код с использованием функций, декомпозируйте задачу, перепишите данные из xml в hdf формат. Учтите исключения, отсутствующие данные отбрасывайте.

2 ЛАБОРАТОРНАЯ РАБОТА № 1 «РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ НА PYTHON»

Целью данной лабораторной работы является изучение возможностей языка Python по обработке данных, создание веб-приложения с использованием фреймворка Flask, системы контроля версий GIT и системы непрерывной интеграции Travis. Перед выполнением задания лабораторной работы необходимо выполнить задания из первой главы настоящих методических указаний.

2.1 Непрерывная интеграция для GitHub

Непрерывная интеграция (англ. Continuous Integration, CI) – способ разработки программного обеспечения, который заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счёт наиболее раннего обнаружения и устранения ошибок и противоречий, но основным преимуществом является сокращение стоимости исправления дефекта благодаря раннему его выявлению.

Непрерывная интеграция впервые концептуализирована и предложена Гради Бучем в 1991 г. Является одним из основных элементов практики экстремального программирования.

Для применения практики необходимо выполнение ряда базовых требований к проекту разработки. В частности, исходный код и всё, что необходимо для сборки и тестирования проекта, должно храниться в репозитории системы управления версиями, а операции копирования из репозитория, сборки и тестирования всего проекта должны быть автоматизированы и легко вызываться из внешних программ.

Для организации процесса непрерывной интеграции на выделенном сервере запускается служба, в задачи которой входят:

- получение исходного кода из репозитория;
- сборка проекта;
- выполнение тестов;
- развёртывание готового проекта;
- отправка отчетов.

2.2 Примеры веб-сервисов для непрерывной интеграции

Существует ряд веб-сервисов, которые позволяют реализовать процесс непрерывной интеграции. Для операционной системы Windows это AppVeyor https://www.appveyor.com/, для Mac OS и Linux – Travis CI https://travis-ci.org/.

Мы будем работать под Linux с использованием Travis CI. Он имеет ряд особенностей, которые делают его хорошим выбором для начала работы с конвейерами сборки:

- быстро интегрируется с любым общедоступным GitHub-репозиторием;
- поддерживает все основные языки программирования;
- развертывается на нескольких разных облачных платформах;
- предлагает множество инструментов для обмена сообщениями и оповещения.

На высоком уровне он работает путем мониторинга github-репозитория на предмет новых коммитов (commit).

Когда создается новый коммит, Travis CI выполняет шаги конвейера сборки, как определено в файле конфигурации. Если какой-либо шаг не удался, конвейер завершается, и об этом создается уведомление.

Из коробки Travis CI требует незначительных настроек конфигурации. Единственная необходимая конфигурация – это указание языка программирования.

Всегда можно предоставить больше настроек конфигурации для адаптации нашего конвейера, если это необходимо. Например, мы можем ограничить ветви, для которых запускаются сборки, добавить дополнительные шаги в конвейер и многое другое.

Travis умеет работать как из полновесной виртуальной машины, так и из Docker-контейнера. В теории, это позволяет сократить время между git push и началом сборки приблизительно на одну минуту. К сожалению, на практике за это ускорение придётся заплатить потерей возможности делать sudo, а это, в свою очередь, ведёт к ограничениям при установке нужных зависимостей.

Как именно собирать, тестировать и развёртывать проект, описывается в специальном конфигурационном файле на языке YAML. Этот файл должен лежать в корне репозитория и иметь имя .travis.yml или appveyor.yml (допускается .appveyor.yml) – для Travis CI и AppVeyor соответственно.

2.3 Что такое YAML

YAML – это язык с синтаксическим структурированием с помощью отступов (как и, например, Python), но при этом не разрешается использование табуляции.

После того как YAML-файлы добавлены в репозиторий, нужно включить непрерывную интеграцию для заданного проекта на сайтах Travis и AppVeyor. Нужно зайти на https://travis-ci.org, авторизовавшись через свой github-аккаунт. Затем соглашаемся с доступом, который запрашивает Travis CI (ему нужно будет получать уведомления о новых коммитах), синхронизуем список своих проектов, выбираем нужный и щёлкаем на включатель. Можно повторить аналогичный процесс на сайте https://ci.appveyor.com, если вы все-таки решили использовать Windows.

Начиная с этого момента каждый git push в ваш репозиторий будет запускать процесс непрерывной интеграции: сервисы Travis поднимут виртуальную машину, настроят среду, установят зависимости, скачают ваш проект, соберут

и протестируют его, а также, при желании, выложат инсталляторы, архивы с исходниками и документацию – всё согласно спецификации в YAML-файлах.

В создании YAML-файлов и заключается основная работа.

Когда вы зайдете на сайт travis-ci, вас попросят о входе через GitHub или Bitbucket. Выбираем GitHub. Возможно, произойдет переключение на travis-ci.com, если вы заходите использовать приватные репозитории GitHub, но можно указать org и работать с org.

2.4 Создание проекта веб-приложения на Flask

Создадим на сайте GitHub новый проект. Можете заполнить файл readme.md. Если будете использовать git client на своей машине, то лучше не делать этого сразу. Дальнейшие указания будут связаны с непосредственной работой через сайт GitHub. Создадим каталог, в котором будет храниться наш проект, flaskapp. Для этого выбираем Create New File (рис. 2.1).

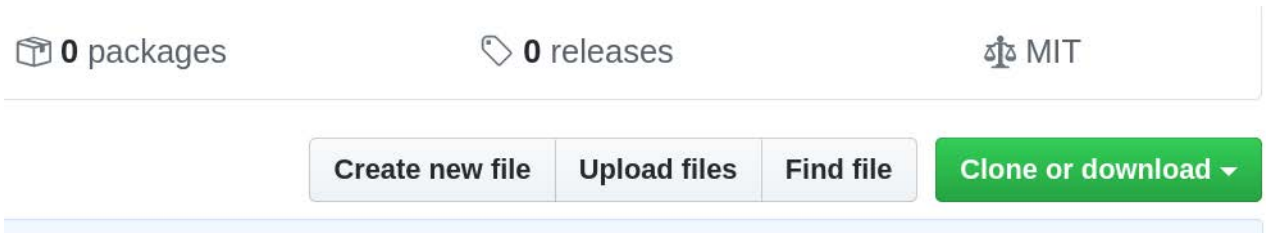

Рис. 2.1 – Создание файла в проекте

Поставим слеш после имени каталога flaskapp и имя файла .gitkeep (рис. 2.2). В указанном файле можем написать, что угодно, например, для чего данный каталог. Нажимаем кнопку commit снизу.

Рис. 2.2 – Создание папки проекта и неиспользуемого файла

Теперь создадим файлы, в которых будут указаны библиотеки и инициирующие данные для реализации нашего веб-сервера. Перед разработкой веб-сервера желательно разработать модель будущего сайта, провести анализ требований пользователей, оценить возможную нагрузку на ваш сайт. Но мы пишем достаточно простое приложение, и поэтому пока изучим возможности CI и создание простого веб-сервиса. В нашем каталоге на сайте создадим файл some_app.py, в котором содержится следующий код:

```
print("Hello world")
```

Можно создать пустой файл и туда скопировать код, можно скопировать созданный файл с локального диска, затем реализовать commit.

После того как вы привязали свой проект на GitHub с непрерывной интеграцией travis-ci, можно создать файл .travis.yml в проекте GitHub и после commit будет запущен build (рис. 2.3).

```
language: python
install:
  - pip3 install flask
script:
  - python3 ./flaskapp/some_app.py
```

В данном случае наш проект пуст, и инсталляция Flask тут, по сути, тоже не нужна.

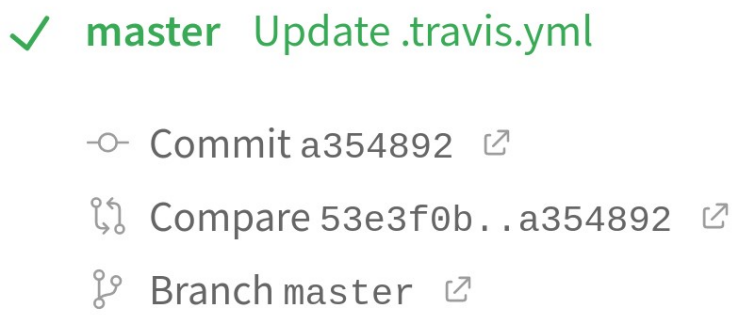

Рис. 2.3 – Пример подтверждения commit

Здесь мы просто запускаем скрипт, который напечатает "Hello world".

2.5 Продолжение простейшего эксперимента с проектом Flask

Продолжим дальше наши эксперименты уже непосредственно с простейшим сайтом с использованием фреймворка Flask. С точки зрения непрерывной интеграции необходимо создать работающий веб-сайт, протестировать его и разместить на каком-либо хостинге. Система непрерывной интеграции позволяет это сделать, в частности, на Heroku. Пока попробуем просто запустить сайт в фоновом режиме. Для этого нам понадобится создать небольшой скрипт под Linux и изменить файл YAML. Удобнее всего это делать, используя git-клиент, но, если у вас его нет, можно воспользоваться непосредственно интерфейсом сайта GitHub. Если создавать файлы из меню, это всякий раз будет вызывать процесс build на travis-ci при нажатии commit, очевидно, некоторые из этих build не сработают вовсе. Потому лучше создать у себя на диске отдельную директорию и потом сделать upload содержимого директории на GitHub через интерфейс и подтвердить commit.

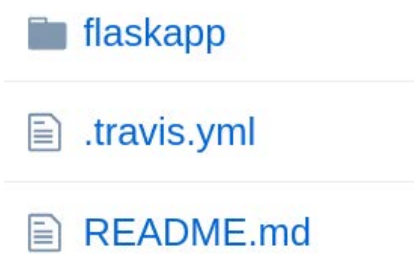

Рис. 2.4 – Папка проекта

Итак, какие файлы нам понадобятся в каталоге flaskapp. Файл, содержащий приложение Flask, назовем some_app.py. Можно создавать свой проект непосредственно в среде PyCharm, а затем скопировать весь проект на сайт GitHub.

```
print("Hello world")
from flask import Flask
app = Flask(__name__)

#декоратор для вывода страницы по умолчанию
@app.route("/")
def hello():
    return " <html><head></head> <body> Hello World! </body></html>"

if __name__ == "__main__":
    app.run(host='127.0.0.1',port=5000)
```

2.5.1 Что такое WSGI

WSGI (Web Server Gateway Interface) – это стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером, например Apache. WSGI является потомком CGI (Common Gateway Interface). Когда Веб начал развиваться, CGI разрастался из-за поддержки огромного количества языков и отсутствия других решений. Однако такое решение было медленным и ограниченным. WSGI был разработан как интерфейс для маршрутизации запросов от веб-серверов (Apache, Nginx и т. д.) на веб-приложения.

В простейшем случае WSGI состоит из двух основных компонентов:

- веб-сервер (Nginx, Apache и т. д.);
- веб-приложение, написанное на языке Python.

Веб-сервер исполняет код и отправляет связанную с http-запросом информацию и callback-функцию в веб-приложение. Затем запрос на стороне приложения обрабатывается и высылается ответ на веб-сервер. Периодически между веб-сервером и веб-приложением существуют одна или несколько промежуточных прослоек. Такие прослойки позволяют осуществить, например, балансировку между несколькими веб-приложениями или предпроцессинг (предобработку) отдаваемого контента.

Файл wsgi.py.

```
from some_app import app
if __name__ == "__main__":
    app.run()
```

2.5.2 Примеры WSGI-серверов и Gunicorn

В качестве веб-сервера будем использовать Gunicorn, хотя есть и множество других (Bjoern, uWSGI, mod_wsgi, Meinheld, CherryPy). Gunicorn – это WSGI-сервер, созданный для использования в UNIX-системах. Название – сокращенная и комбинированная версия слов «Green Unicorn». Он относительно быстрый, легко запускается и работает с широким спектром веб-фреймворков. Команда разработчиков рекомендует использовать Gunicorn в связке с Nginx, где Nginx используется в качестве прокси-сервера.

2.5.3 Запуск проекта с использованием Gunicorn

В файле YAML наряду с Flask мы установим Gunicorn, а в скрипте ниже реализуется вызов веб-сервера с нашим веб-приложением.

Файл st.sh для вызова в скрипте yaml:

```
gunicorn --bind 127.0.0.1:5000 wsgi:app & APP_PID=$!
sleep 5
echo $APP_PID
kill -TERM $APP_PID
echo process gunicorns kills
exit 0
```

Предварительно можно протестировать ваш проект на локальной машине путем запуска из командной строки вашего проекта.

```
gunicorn --bind 127.0.0.1:5000 wsgi:app
```

Затем в браузере наберите адрес 127.0.0.1:5000.

Что здесь делается (в скрипте st.sh)? Сначала мы запускаем веб-сервер в фоновом режиме, на это указывает символ & в конце команды, при этом в переменной APP_PID мы сохраняем PID последнего фонового процесса, этот номер хранится в переменной $!. Затем делаем приостановку на 5 секунд, это сделано заранее, чтобы потом можно было запустить какие-то проверочные скрипты, возможно, этого будет и недостаточно. Наше приложение выполняется в так называемых worker-ах, можно указать их количество. После этого мы останавливаем master процесс веб-сервера kill -TERM. Можно посмотреть другие ключи для команды kill. Например, -HUP, перезапуск.

Запуск сервера в фоновом режиме реализуется, чтобы освободить консоль для дальнейших команд, иначе travis-ci будет ждать бесконечно завершение процесса (порядка часа).

Файл .travis.yml, который хранится вне папки flaskapp.

```
language: python
before_install:
  - chmod +x ./flaskapp/st.sh
install:
  - pip3 install flask
```

```
  - pip3 install gunicorn
script:
  - cd flaskapp
  - ./st.sh
```

Здесь мы указываем, что будет запускаться в виртуальной машине. Как видим, указывается язык программирования. До инсталляции назначается атрибут файла скрипта – исполнимый. Устанавливается Flask, Gunicorn, в исполнимых скриптах происходит переход в папку и запуск нашего скрипта.

В данном случае произойдет успешный build коммита.

Попробуем запустить некоторый тест. Создадим файл client.py.

```
import requests
r = requests.get('http://localhost:5000/')
print(r.status_code)
print(r.text)
```

Изменим файл st.sh на следующий:

```
gunicorn --bind 127.0.0.1:5000 wsgi:app & APP_PID=$!
sleep 5
echo start client
python3 client.py
sleep 5
echo $APP_PID
kill -TERM $APP_PID
exit 0
```

Изменим наш файл .travis.yml, загрузим файлы на github и закоммитим (commit).

```
language: python
before_install:
  - chmod +x ./flaskapp/st.sh
install:
  - pip3 install flask
  - pip3 install gunicorn
  - pip3 install requests
script:
  - cd flaskapp
  - ./st.sh
```

Ниже в job log выводится консольная информация при запуске виртуальной машины с инициирующими командами из YAML-файла.

2.5.4 Ремарка о тестировании

Очевидно, build пройдет независимо от того, проработает ли наш импровизированный тест или нет, учитывая, что мы в любом случае завершаем наш исполнимый файл с успехом. Возможна только проблема с «зависанием». Так как мы вызываем скрипт, который в любом случае возвращает успешное исполнение, такое использование travis-ci и системы контроля версий не совсем обоснованно и не имеет особого смысла, кроме того, что, допустим, у вас нет возможностей проверки работы на локальной машине. То есть в этом случае практически любой коммит будет подтвержден, несмотря на, казалось бы, отсутствие каких-то библиотек и ошибок сборки проекта. Тем не менее, пока попытаемся использовать travis-ci для разворачивания нашего проекта. А потом посмотрим, как на самом деле нужно было запускать тестовую проверку.

Естественно, хотелось бы потом добавить возможности автоматизированного тестирования и деплоя нашего проекта на какой-либо сервер. Для этого есть соответствующие средства, для тестирования веб-сервера можно использовать selenium.webdriver, для тестирования приложения подходит pytest, тесты можно указать в качестве блока YAML-файла в поле script.

2.6 Краткое знакомство с шаблонами Flask

Для начала изучим возможности Flask для нашего будущего проекта. Нам понадобятся шаблоны. Импортируем в наш файл some_app.py модуль и добавим новую функцию.

```
from flask import render_template
#наша новая функция сайта
@app.route("/data_to")
def data_to():
    #создаем переменные с данными для передачи в шаблон
    some_pars = {'user':'Ivan','color':'red'}
```

```
    some_str = 'Hello my dear friends!'
    some_value = 10
    #передаем данные в шаблон и вызываем его
    return render_template('simple.html',some_str = some_str,
    some_value = some_value,some_pars=some_pars)
```

Здесь мы передаем словарь, строку и просто целое значение. Для удобства можно все передать в одном словаре.

Создадим также сам шаблон, для этого сначала создадим каталог templates в нашем каталоге приложения. И запишем туда файл simple.html.

```
<html>
  <head>
    {% if some_str %}
    <title>{{ some_str }} </title>
    {% else %}
    <title>Hm, there is no string!</title>
    {% endif %}
  </head>
  <body>
    <h1 style="color:{{ some_pars.color }};">Hello, {{ some_pars.user }}!</h1>
    <h2>some_value {{ some_value }}!</h2>
  </body>
</html>
```

В шаблон {{}} можно передавать не только переменные, но и функции, т. к. в Python функция также является объектом. В данному случае функция render_template вызывает шаблонизатор Jinja2, который является частью фреймворка Flask. Jinja2 заменяет блоки {{...}} на соответствующие им значения, переданные как аргументы шаблона. А в {% %} можно указывать специальные управляющие операторы, в данном случае if else endif. Можно также, например,

```
{% for x in mylist | reverse %}
{% endfor %}
```

и многие другие.

У себя на локальной машине запустите ваш проект через Gunicorn и проверьте 127.0.0.1:5000/data_to.

При этом мы видим, как динамически сформировалась страница, с переданным заголовком страницы, строкой с именем пользователя красного цвета.

Используя GitHub или Git client, загрузите файлы в проект. Можно сначала создать папку templates и в нее поместить файл simple.html. Затем обновить файл some_app.py, затем – client.py.

```
import requests
r = requests.get('http://localhost:5000/')
print(r.status_code)
print(r.text)
r = requests.get('http://localhost:5000/data_to')
print(r.status_code)
print(r.text)
```

Тревис должен успешно обработать наш коммит, как и было сказано выше, практически в любом случае. Если хотите, можете сразу перенести тестирование в YAML-файле в соответствующий блок либо изменить возврат ошибки в файле st.sh не на exit 0. То, как у вас это получилось, можете включить в отчет.

2.7 Изучение шаблонов, форм

Использование шаблонов не всегда удобно. В силу того что нам так или иначе придется описывать нашу страницу целиком на HTML, нам бы хотелось ускорить этот процесс, используя заготовки, для этого можно воспользоваться удобными библиотеками bootstrap и WTForms. Кроме того, добавим в наш проект нейронную сеть, которая будет классифицировать изображения.

Добавление в проект форм

В файл some_app добавим наши формы. Формы, конечно, можно описать непосредственно в шаблоне файла HTML с помощью forms input, но зачем нам это делать, если мы, допустим, не мастера дизайна, пусть все будет делаться за нас гораздо быстрее и желательно в несколько строк.

Общая суть задачи. Добавим обработку запроса GET и POST в наш API: some_app.py, в функции обработки запроса будет рендериться форма, которая

также добавлена как класс в файл some_app.py. В шаблоне template/net.html, который у нас рендерится, мы добавим вызов обработки формы, передаваемой при обработке запроса пользователя с помощью одной функции:

wtf.quick_form(form, method='post',enctype="multipart/form-data", action="net").

Данная функция сама сформирует HTML-код с формой. Так как на форме есть кнопка submit (для submit указан обработчик POST), то, естественно, будет вызван метод POST, который мы обрабатываем в методе net-файла some_app.py. Кроме того, на форме есть капча, проверяющая наличие человека во взаимодействии (как установить капчу, показано ниже, после листингов), загрузка файла с изображением, которое классифицируется нейронной сетью, прописанной в файле net.py, функции НС вызываются тоже из нашего же обработчика. Данные формы автоматически валидируются на введение и правильность.

Дальше изучаем код по комментариям.

Создадим папку во flaskapp/static, поместим туда файл с изображением image0008.png

Например, md static

Установим нужные библиотеки Python.

- pip3 install flask-bootstrap
- pip3 install flask-wtf
- pip3 install pillow
- pip3 install tensorflow==2.0.0-alpha0
- pip3 install keras

Если хотите запускать нейронную сеть с использованием gpu, можно установить tensorflow-gpu==2.0.0-alpha0. Но тогда вам придется устанавливать дополнительно cuda, cudnn, иметь соответствующую видеокарту. Потому в нашем случае достаточно tensorflow для CPU.

Добавим код в some_app.py

Можем этот код добавить непосредственно после того кода, который у нас уже есть, и реализует метод data_to. В коде присутствуют ключи для капчи, которые нужно сформировать на сайте Google.

```
# модули работы с формами и полями в формах
from flask_wtf import FlaskForm,RecaptchaField
from wtforms import StringField, SubmitField, TextAreaField
# модули валидации полей формы
from wtforms.validators import DataRequired
from flask_wtf.file import FileField, FileAllowed, FileRequired
# используем csrf токен, можете генерировать его сами
SECRET_KEY = 'secret'
app.config['SECRET_KEY'] = SECRET_KEY
# используем капчу и полученные секретные ключи с сайта Google
app.config['RECAPTCHA_USE_SSL'] = False
app.config['RECAPTCHA_PUBLIC_KEY'] = 'сюда поместить ключ из google'
app.config['RECAPTCHA_PRIVATE_KEY'] = 'сюда поместить секретный ключ из google'
app.config['RECAPTCHA_OPTIONS'] = {'theme': 'white'}
# обязательно добавить для работы со стандартными шаблонами
from flask_bootstrap import Bootstrap
bootstrap = Bootstrap(app)
# создаем форму для загрузки файла
class NetForm(FlaskForm):
    # поле для введения строки, валидируется наличием данных
    # валидатор проверяет введение данных после нажатия кнопки submit
    # и указывает пользователю ввести данные, если они не введены
    # или неверны
    openid = StringField('openid', validators = [DataRequired()])
    # поле загрузки файла
    # здесь валидатор укажет ввести правильные файлы
    upload = FileField('Load image', validators=[
        FileRequired(),
        FileAllowed(['jpg', 'png', 'jpeg'], 'Images only!')])
    # поле формы с capture
    recaptcha = RecaptchaField()
    #кнопка submit, для пользователя отображена как send
    submit = SubmitField('send')
# функция обработки запросов на адрес 127.0.0.1:5000/net
# модуль проверки и преобразование имени файла
# для устранения в имени символов типа / и т.д.
from werkzeug.utils import secure_filename
import os
```

```
# подключаем наш модуль и переименовываем
# для исключения конфликта имен
import net as neuronet
# метод обработки запроса GET и POST от клиента
@app.route("/net",methods=['GET', 'POST'])
def net():
    # создаем объект формы
    form = NetForm()
    # обнуляем переменные, передаваемые в форму
    filename=None
    neurodic = {}
    # проверяем нажатие сабмит и валидацию введенных данных
    if form.validate_on_submit():
        # файлы с изображениями читаются из каталога static
        filename = os.path.join('./static', secure_filename(form.upload.data.filename))
        fcount, fimage = neuronet.read_image_files(10,'./static')
        # передаем все изображения в каталоге на классификацию
        # можете изменить немного код и передать только загруженный файл
        decode = neuronet.getresult(fimage)
        # записываем в словарь данные классификации
        for elem in decode:
            neurodic[elem[0][1]] = elem[0][2]
        # сохраняем загруженный файл
        form.upload.data.save(filename)
    # передаем форму в шаблон, так же передаем имя файла и результат работы нейронной
    # сети, если был нажат сабмит, либо передадим falsy значения
    return render_template('net.html',form=form,image_name=filename,neurodic=neurodic)
```

Здесь используется класс формы, которая реализует размещение полей ввода строки, капчи и загрузки файла, кроме того, шаблон выводит содержимое изображения. Автоматически благодаря bootstrap реализуется отображение формы. Для работы шаблона не забудьте в some_app.py добавить строчку кода:

```
bootstrap = Bootstrap(app)
```

Также не забудьте добавить секретный токен для защиты от CSRF-атаки. Можете его генерировать как случайную строку при запуске сервера.

```
SECRET_KEY = 'secret'
app.config['SECRET_KEY'] = SECRET_KEY
```

В папке templates создадим шаблон net.html для обработки форм.

```
{% extends "bootstrap/base.html" %}
{% import "bootstrap/wtf.html" as wtf %}
<!-- задаем заголовок страницы -->
{% block title %}This is an page{% endblock %}
<!-- блок body -->
{% block content %}
{{ wtf.quick_form(form, method='post',enctype="multipart/form-data", action="net") }}
<!-- один из стандартных тегов html – заголовок второго уровня -->
<h2>Classes: </h2>
<!—проверяем, есть ли данные классификации -->
{% if neurodic %}
  <!-- запускаем цикл прохода по словарю и отображаем ключ-значение -->
  <!-- классифицированных файлов -->
  {% for key, value in neurodic.items() %}
  <h3>{{key}}: {{value}}</h3>
  {% endfor %}
{% else %}
  <h3> There is no classes </h3>
{% endif %}
<h2>Image is here: </h2>
<!-- отображаем загруженное изображение с закругленными углами -->
<!-- если оно есть (после submit) -->
{% if image_name %}
  <p>{{image_name}}
  <p><img src={{image_name}} class="img-rounded" alt="My Image" width = 224 height=224 />
{% else %}
  <p> There is no image yet </p>
{% endif %}
{% endblock %}
```

2.8 Добавление нейронной сети для классификации

Создадим в основной папке файл net.py.

```
import random
# библиотека keras для НС
import keras
# входной слой сети и модель сети
```

```
from keras.layers import Input
from keras.models import Model
# одна из предобученных сетей
from keras.applications.resnet50 import preprocess_input, decode_predictions
import os
# модуль работы с изображениями
from PIL import Image
import numpy as np
# для конфигурации gpu
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
# настраиваем работу с GPU, для CPU эта часть не нужна
config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.7
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
height = 224
width = 224
nh=224
nw=224
ncol=3
# загружаем и создаем стандартную уже обученную сеть keras
visible2 = Input(shape=(nh,nw,ncol),name = 'imginp')
resnet = keras.applications.resnet_v2.ResNet50V2(include_top=True,
weights='imagenet', input_tensor=visible2,
input_shape=None, pooling=None, classes=1000)
# чтение изображений из каталога
# учтите, если там есть файлы, не соответствующие изображениям, или каталоги
# возникнет ошибка
def read_image_files(files_max_count,dir_name):
    files = os.listdir(dir_name)
    files_count = files_max_count
    if(files_max_count>len(files)): # определяем количество файлов не больше max
        files_count = len(files)
    image_box = [[]]*files_count
    for file_i in range(files_count): # читаем изображения в список
        image_box[file_i] = Image.open(dir_name+'/'+files[file_i]) #  / ??
    return files_count, image_box
```

```
# возвращаем результаты работы нейронной сети
def getresult(image_box):
    files_count = len(image_box)
    images_resized = [[]]*files_count
    # нормализуем изображения и преобразуем в питру
    for i in range(files_count):
        images_resized[i] = np.array(image_box[i].resize((height,width)))/255.0
    images_resized = np.array(images_resized)
    # подаем на вход сети изображение в виде питру массивов
    out_net = resnet.predict(images_resized)
    # декодируем ответ сети в один распознанный класс top=1 (можно больше классов)
    decode = decode_predictions(out_net, top=1)
    return decode
# заранее вызываем работу сети, так как работа с gpu требует времени
# из-за инициализации библиотек
# возможно, лучше убрать и закомментировать эти строки
# fcount, fimage = read_image_files(1,'./static')
# decode = getresult(fimage)
```

2.9 Добавление капчи

Создаем проверку google-капчи. Для этого заходим по адресу https://www.google.com/recaptcha, затем выбираем admin console. Создаем ключи для капчи, label – localhost, выбираем капчу второй версии, добавляем два домена localhost и 127.0.0.1. Копируем ключи (Copy site key, Copy secret key) в

```
app.config['RECAPTCHA_PUBLIC_KEY'] = '_____'
app.config['RECAPTCHA_PRIVATE_KEY'] = '_____'
```

2.10 Добавление возможности классификации изображения

Расширим функциональность нашего проекта, добавив обработку запроса от клиента в json-формате. Общая идея заключается в передаче от клиента в json-запросе файла изображения, закодированного строкой base64, и затем сервер возвращает класс объекта, изображенного на картинке.

Добавим в наш some_app.py следующий код:

```
from flask import request
from flask import Response
```

```
import base64
from PIL import Image
from io import BytesIO
import json
# метод для обработки запроса от пользователя
@app.route("/apinet",methods=['GET', 'POST'])
def apinet():
    neurodic = {}
    # проверяем, что в запросе json данные
    if request.mimetype == 'application/json':
        # получаем json данные
        data = request.get_json()
        # берем содержимое по ключу, где хранится файл
        # закодированный строкой base64
        # декодируем строку в массив байт, используя кодировку utf-8
        # первые 128 байт ascii и utf-8 совпадают, потому можно
        filebytes = data['imagebin'].encode('utf-8')
        # декодируем массив байт base64 в исходный файл изображение
        cfile = base64.b64decode(filebytes)
        # чтобы считать изображение как файл из памяти, используем BytesIO
        img = Image.open(BytesIO(cfile))
        decode = neuronet.getresult([img])
        neurodic = {}
        for elem in decode:
            neurodic[elem[0][1]] = str(elem[0][2])
            print(elem)
        # пример сохранения переданного файла
        # handle = open('./static/f.png','wb')
        # handle.write(cfile)
        # handle.close()
    # преобразуем словарь в json-строку
    ret = json.dumps(neurodic)
    # готовим ответ пользователю
    resp = Response(response=ret,
                    status=200,
                    mimetype="application/json")
    # возвращаем ответ
    return resp
```

Здесь мы не проверяем ошибки в самом json-запросе. Кроме того, если это не json-запрос, вернется пустой словарь. Желательно, конечно, добавить все необходимые проверки, чтобы у вас не возникала ошибка на стороне сервера 500.

Итоговый client.py, запрашивающий сервис, который мы создали

```
# импортируем нужные модули
import os
from io import BytesIO
import base64
img_data = None
# создаем путь к файлу (для кросс-платформенности, например)
path = os.path.join('./static','image0008.png')
# читаем файл и енкодируем его в строку base64
with open(path, 'rb') as fh:
    img_data = fh.read()
    b64 = base64.b64encode(img_data)
# создаем json словарь, который
# отправляется на сервер в виде json-строки
# преобразование делает сама функция отправки запроса post
jsondata = {'imagebin':b64.decode('utf-8')}
res = requests.post('http://localhost:5000/apinet', json=jsondata)
if res.ok:
    print(res.json())
```

Можем все это теперь закоммитить на github. И подкрепить проверку на travis. Что должно быть у нас в итоге в папке flaskapp:

```
/flaskapp
    /static
        image0008.png
    /templates
        net.html
        simple.html
    some_app.py
    client.py
    net.py
    st.sh
```

Реализуем коммит и изменим содержимое YAML-файла.

```
language: python
before_install:
  - chmod +x ./flaskapp/st.sh
install:
  - pip3 install flask
  - pip3 install gunicorn
  - pip3 install requests
  - pip3 install flask-bootstrap
  - pip3 install flask-wtf
  - pip3 install pillow
  - pip3 install tensorflow==2.0.0-alpha0
  - pip3 install keras
script:
  - cd flaskapp
  - ./st.sh
```

Если все нормально, то на travis-ci в конце будут такие строчки:

```
[('n06359193', 'web_site', 0.9643341)]
{'web_site': '0.9643341'}
3620
[2020-04-30 07:39:08 +0000] [3620] [INFO] Handling signal: term
[2020-04-30 07:39:08 +0000] [3624] [INFO] Worker exiting (pid: 3624)
The command "./st.sh" exited with 0.
```

Либо вместо web_site распознанный класс, переданный вами в файле скрипта client.py.

Вообще говоря, тут мы используем travis-ci как удаленное средство запуска проекта. Обычно предполагается, что есть множество разработчиков и частое слияние рабочих копий в общий проект. Если бы мы сразу пошли путем размещения теста в соответствующем блоке или следили за правильностью срабатывания скрипта, то коммитов нерабочих проектов и не происходило бы.

2.11 Дополнительная возможность по возвращению разных документов в зависимости от шаблона

Добавим дополнительную функциональность в наш проект, в данном случае обработку xml-документа с помощью шаблона обработки xsl. В папке static добавим папку xml и туда запишем два файла: file.xml и file.xslt. Как вариант, предлагается взять различные прикладные области и возвращать данные в виде таблицы, списка, простого текста. Мы здесь рассмотрим только один шаблон как пример.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="file.xslt" ?>
<people>
 <man id= "1">
  <name>John</name>
  <age>30</age>
  <work>Driver</work>
 </man>
 <man id = "2">
  <name>Lisa</name>
  <age>20</age>
  <work>Programmist</work>
 </man>
</people>
```

Шаблон выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version = "1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
  <head>
   <title>People</title>
  </head>
  <body>
   <table border = "1">
  <tbody>
   <xsl:for-each select="people/man">
    <tr>
```

```
            <th>
          <xsl:value-of select="@id"/>
           </th>
           <th>
          <xsl:value-of select="name"/>
           </th>
           <th>
               <xsl:value-of select="age"/>
           </th>
           <th>
               <xsl:value-of select="work"/>
           </th>
         </tr>
       </xsl:for-each>
     </tbody>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

В файл some_app.py добавим наш новый API.

```
import lxml.etree as ET
@app.route("/apixml",methods=['GET', 'POST'])
def apixml():
    #парсим xml файл в dom
    dom = ET.parse("./static/xml/file.xml")
    #парсим шаблон в dom
    xslt = ET.parse("./static/xml/file.xslt")
    #получаем трансформер
    transform = ET.XSLT(xslt)
    #преобразуем xml с помощью трансформера xslt
    newhtml = transform(dom)
    #преобразуем из памяти dom в строку, возможно, понадобится указать кодировку
    strfile = ET.tostring(newhtml)
    return strfile
```

В файле client.py добавим следующие строки для тестирования нового API.

```
try:
    r = requests.get('http://localhost:5000/apixml')
    print(r.status_code)
    if(r.status_code!=200):
        exit(1)
    print(r.text)
except:
    exit(1)
```

Файл st.sh изменим так, чтобы он возвращал нам ошибку в случае, если процесс не будет выполнен.

```
gunicorn --bind 127.0.0.1:5000 wsgi:app & APP_PID=$!
sleep 25
echo start client
python3 client.py
APP_CODE=$?
sleep 5
echo $APP_PID
kill -TERM $APP_PID
echo app code $APP_CODE
exit $APP_CODE
```

Теперь у нас при выполнении данного скрипта в случае ошибок исполнения будет возвращаться код ошибки и коммит не будет фиксироваться на github.

Загрузим новый проект на github.

Давайте проверим. Изменим YAML-файл, добавив туда

```
- pip3 install lxml
```

При коммите у нас возникает ошибка, которая обусловлена тем, что мы использовали не очень хорошо спроектированную функцию в net.py, которая считывает все подряд, включая каталоги, кроме того, мы не обрабатываем никак try except на сервере. Заменим в файле net.py функцию:

```
def read_image_files(files_max_count,dir_name):
    files = [item.name for item in os.scandir(dir_name) if item.is_file()]
    files_count = files_max_count
    if(files_max_count>len(files)): # определяем количество файлов не больше max
        files_count = len(files)
```

```
    image_box = [[]]*files_count
    for file_i in range(files_count): # читаем изображения в список
        image_box[file_i] = Image.open(dir_name+'/'+files[file_i]) # / ??
    return files_count, image_box
```

Загрузим на GitHub новую версию net.py. Теперь все должно получиться, если нет, смотрим ошибки и пытаемся их исправить, пока коммит не подтвердится.

2.12 Деплой на Heroku

Можно реализовать деплой непосредственно на Heroku:

```
deploy:
  provider: heroku
  api_key:
    secure: "YOUR ENCRYPTED API KEY"
```

Правда, необходимо получить heroku auth:token, это можно сделать, например, через командную строку и команду:

heroku auth:token,

но для этого необходимо установить Command Line Interface Heroku https://devcenter.heroku.com/categories/command-line или по адресу https://dashboard.heroku.com/account (рис. 2.5).

Рис. 2.5 – Пример генерации ключа на Heroku

Чтобы реализовать деплой на Heroku, желательно получить encrypt ключа с помощью консоли travis-ci, чтобы никто не увидел вашего секретного ключа Heroku. Если вам не страшны возможности доступа со стороны к вашему аккаунту, можете добавить ключ непосредственно в открытом виде в YAML-файл на github в поле api_key. Но предварительно придется установить travis-cli.

```
$ travis encrypt YOUR_API_HEROKU_SECRET --org -r YOUR_GIT_ACCOUNT/YOUR_WEBPROJECT
```

При разработке желательно использовать стандартный buildpack для какого-то языка, клонировав его с официального репозитория. Для Python это, например:

```
$ git clone https://github.com/heroku/python-getting-started.git
```

Затем можно настроить свой проект на базе созданного, правда, тут используется django.

Но мы пойдем другим путем. Зарегистрируемся на сайте Heroku и создадим приложение. В нашем приложении на GitHub сделаем изменения. Heroku определяет наличие какого-либо типа приложения и поддержку языка на основе наличия определенных специализированный файлов. Например, для Python это requirements.txt, setup.py или pipfile в корне нашего проекта вместе с YAML-файлом. Добавим requirements.txt следующего содержания:

```
gunicorn==20.0.4
Flask==1.1.2
requests==2.23.0
Flask-Bootstrap==3.3.7.1
Flask-WTF==0.14.3
Pillow==6.2.2
tensorflow==2.0.1
Keras==2.3.1
lxml==4.3.3
```

Как видите, это те библиотеки, которые нам понадобятся при работе нашего приложения. Heroku будет считывать данный файл при попытке build нашего приложения.

Кроме того, мы должны добавить Procfile, где укажем запуск воркеров через Gunicorn.

```
web: gunicorn wsgi:app -b 0.0.0.0:$PORT --chdir flaskapp
```

Порт указывается Heroku.

Добавим runtime.txt, где укажем, например, версию Python. Можете указать другую версию.

```
python-3.7.6
```

Слегка перепишем YAML-файл.

```
language: python
python:
 - "3.7.6"
before_install:
 - chmod +x ./flaskapp/st.sh
install:
 - pip install -r requirements.txt
script:
 - cd flaskapp
 - ./st.sh
```

Удалим в нашем файле net.py строки вызова нейронной сети в конце скрипта. Или закомментируем

```
# fcount, fimage = read_image_files(1,'./static')
# decode = getresult(fimage)
```

В нашем созданном приложении на Heroku выберем вкладку деплой.

Выберете деплой-метод Github (рис. 2.6).

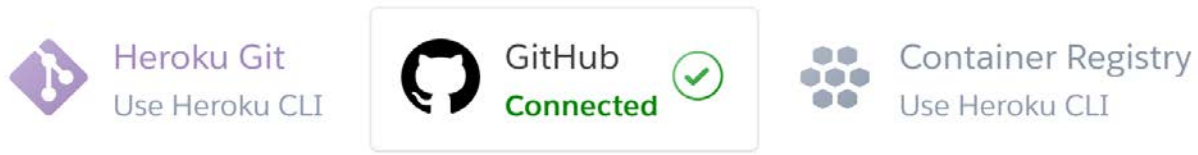

Рис. 2.6 – Пример выбора деплоя

После этого реализуйте подключение к вашему аккаунту и присоединение к вашему проекту с Flask.

Во вкладке Automatic deploys укажете галочкой поле Wait for CI to pass before deploy, чтобы деплоймент разрешался системой интеграции travis-ci.

По идее все, после того как вы сделаете коммит всех изменений проекта, он должен загрузиться в качестве вашего приложения на Heroku. Приложение получилось тяжелым и медленным, возможно, потребует дополнительного времени ожидания или перезагрузки на странице браузера.

Подумайте, как решить проблему постоянного сохранения и накопления файлов в папке static.

Как ускорить загрузку приложения в связи с постоянной загрузкой нейронной сети.

Тестирование pytest остается на самостоятельное изучение.

Если вам хочется просмотреть структуру папок на Heroku, нужно установить себе консольный клиент.

```
$ sudo snap install --classic heroku
```

или

```
$ curl https://cli-assets.heroku.com/install-ubuntu.sh | sh
$ sudo apt-get install heroku
```

Воспользуйтесь командой.

```
$ heroku run bash --app YOUR_HEROKU_APP
```

Для просмотра логов можно воспользоваться командой:

```
$ heroku logs --tail --app YOUR_HEROKU_APP
```

2.13 Задание на лабораторную работу № 1

Внимательно изучите материал методических указаний, последовательно выполняя представленные задания. После изучения и выполнения предложенных заданий реализуйте в соответствии со своим вариантом веб-приложение с использованием фреймворка Flask. Каждое приложение должно обеспечивать проверку на робота с помощью капчи или любой другой технологии. Разместите объекты отображения и ввода удобно для пользователя.

Варианты для выполнения задания

Вариант 1

Веб-приложение должно обеспечивать преобразование подаваемой на вход картинки путем устранения шума в соответствии с переданным параметром сглаживания или любого другого параметра (метод фильтрации можно выбрать самостоятельно). Должна быть возможность вывода графика распределения цветов на картинке и графика распределения шума.

Вариант 2

Веб-приложение должно изменять контраст картинки по указанному значению уровня контраста и показывать результат пользователю, выдавать графики распределения цветов исходной и полученной картинки.

Вариант 3

Веб-приложение должно изменять яркость картинки по указанному значению уровня яркости и показывать результат пользователю, выдавать графики распределения цветов исходной и полученной картинки.

Вариант 4

Веб-приложение должно смешивать две картинки по указанному значению уровня смешения от 0 до 1 и показывать результат пользователю, выдавать графики распределения цветов исходных и полученной картинки.

Вариант 5

Веб-приложение должно изменять размер картинки по указанному значению масштаба и показывать результат пользователю, выдавать графики распределения цветов исходной и полученной картинки.

Вариант 6

Веб-приложение должно поворачивать картинку по указанному значению угла поворота и показывать результат пользователю, выдавать графики распределения цветов исходной и полученной картинки.

Вариант 7

Веб-приложение должно менять цветовые карты изображения r, g, b в соответствии с заданным пользователем порядком, выдавать графики распределения цветов исходной картинки и графики среднего значения цвета по вертикали и горизонтали.

Вариант 8

Веб-приложение должно поменять местами правую и левую часть картинки либо верхнюю и нижнюю, в зависимости от желания пользователя, нарисовать график распределения цветов исходной картинки.

Вариант 9

Веб-приложение должно изменить интенсивность любой цветовой карты изображения или сразу нескольких, выдавать графики распределения цветов исходной картинки и новой картинки.

Вариант 10

Веб-приложение должно рисовать на картинке вертикальный или горизонтальный крест заданного цвета в зависимости от желания пользователя, выдавать графики распределения цветов исходной картинки и новой картинки.

Вариант 11

Веб-приложение должно склеить две картинки в одну по вертикали или горизонтали в зависимости от желания пользователя, выдавать графики распределения цветов исходных картинок и новой картинки.

Вариант 12

Веб-приложение должно зашумлять картинку с уровнем шума заданным пользователем, выдавать графики распределения цветов исходной и новой картинки.

Вариант 13

Веб-приложение должно разбить изображение на четыре части по вертикали и горизонтали и создать четыре отдельных изображения, для каждого нового и исходного изображения нарисовать графики распределения цветов.

Вариант 14

Веб-приложение должно разбить изображение на четыре части по вертикали и горизонтали и реализовать сдвиг по часовой стрелке данных частей, создав новое изображение. Нарисовать график распределения цветов для исходного изображения.

Вариант 15

Веб-приложение должно закрасить части изображения в шахматном порядке, размер шахматной клетки задает пользователь в процентах от размера исходного изображения. Нарисовать график распределения цветов для исходного и нового изображения.

Вариант 16

Веб-приложение должно добавить рамку для изображения, размер рамки задает пользователь. Нарисовать график распределения цветов для исходного изображения.

Вариант 17

Веб-приложение должно изменить изображением путем обмена местами чередующихся полос либо по вертикали, либо по горизонтали в зависимости от данных пользователя, так же пользователь может задать ширину полосы. Нарисовать график распределения цветов для исходного изображения.

Вариант 18

Веб-приложение должно формировать новое изображение на основе исходного путем умножения изображения на периодическую функцию sin или cos с нормировкой, период изменения задает пользователь, аргумент функции определяется вертикальной или горизонтальной составляющей. Нарисовать график распределения цветов для нового и исходного изображения.

Вариант 19

Веб-приложение должно формировать новое изображение на основе исходного путем умножения изображения на периодическую функцию sin или cos с нормировкой, период изменения задает пользователь, аргумент функции определяется и вертикальной, и горизонтальной составляющей. Нарисовать график распределения цветов для нового и исходного изображения.

Вариант 20

Веб-приложение должно формировать новое изображение на основе исходного путем сдвига по замкнутым прямоугольным составляющим на определенное число пикселов, которое задает пользователь. Например, сдвигается внешняя рамка, затем вторая и так до внутренней части. Учесть, что размер сдвига внутренней части зависит от размера внутренней части и не должен превышать максимального циклического сдвига по условному кругу. Нарисовать график распределения цветов для исходного изображения.

3 ЛАБОРАТОРНАЯ РАБОТА № 2 «РАЗРАБОТКА ВЕБ-СЕРВИСА»

Целью данной лабораторной работы является изучение создания веб-сервиса с использованием фреймворка Flask и его документирование с использование Swagger.

3.1 Создание отдельной среды окружения для проекта Flasgger и документирования Swagger

Создадим отдельный проект, в котором рассмотрим кратко использование одной из библиотек, позволяющих вести автоматическую документацию API с применением Swagger. Без документации ваш API бесполезен для других людей, впрочем, и для вас через какое-то время, когда вы забудете, что делали ранее, потому необходимо вести документацию, а еще проще ее вести, если это будет делаться автоматически.

Папки, которые созданы в данном проекте, не используются, и даны всего лишь для наглядности и понимания структуры проекта на Flask. В данном случае используется некоторая отдельная часть сайта, сделанная с помощью Blueprint (sitepart), которая содержит свои шаблоны и свои статичные файлы, это и есть основная концепция использования Blueprint. Когда проект разрастается и в одной папке или файле сложно хранить все элементы и части веб-приложения, то можно использовать Blueprint, который позволяет создавать отдельную часть сайта с использованием отдельного каталога, более того, можно вести независимую разработку и потом подключить кусок такого Blueprint к основному сервису просто со своим URL.

Кроме того, в примере будет приведена возможность описания сайта с использованием Swagger, благодаря библиотеке Flasgger. Создадим следующую структуру файлов и папок.

Создав среду, установим необходимые библиотеки.

```
(proj1)$ pip install flask=1.1.2
(proj1)$pip install flask-blueprint=1.3.0
(proj1)$pip install flasgger=0.9.5
```

sitepart

sitepart/static

sitepart/templates

sitepart/sitepart.py

static

templates

main.py

Содержимое main.py.

```
# Подключение библиотек для работы с Flask и Blueprint
from flask import Flask, jsonify, Blueprint
# Подключение библиотеки для создания автоматической документации API
from flasgger import Swagger
# Подключение части нашего веб-сервиса с использованием Blueprint
from sitepart.sitepart import sitepart

# Приложение Flask
app = Flask(__name__)
# Инициализация для нашего API сервиса документации Swagger
swagger = Swagger(app)
# Создаем основной Blueprint сайта
main = Blueprint("/", __name__, template_folder='templates',static_folder='static')
# объявляем декоратор для метода http get
# Информация, которая будет выдаваться по URL/info/something
# Параметр в <> при вводе URL будет передан в переменную about функции info
@main.route('/info/<about>/')
def info(about):
    """Example endpoint returning about info
    This is using docstrings for specifications.
    ---
    parameters:
      - name: about
        in: path
        type: string
        enum: ['all','version', 'author', 'year']
        required: true
```

```
      default: all
    definitions:
      About:
        type: string
    responses:
      200:
        description: A string
        schema:
          $ref: '#/definitions/About'
        examples:
          version: '1.0'
    """
    all_info = {
        'all': 'main_author 1.0 2020',
        'version': '1.0',
        'author': 'main_author',
        'year': '2020'
    }

    result = {about:all_info[about]}
    return jsonify(result)

# Регистрируем основной Blueprint и Blueprint другой части сайта
app.register_blueprint(main,url_prefix='/')
# url_prefix указывает URL в контексте которого будет доступна часть данного Blueprint
app.register_blueprint(sitepart,url_prefix='/sitepart')

# Запуск приложения Flask в режиме debug
app.run(debug=True)
```

Debug = True означает, что отладчик Flask работает. Эта функция полезна при разработке, так как при возникновении проблем она выдает детализированные сообщения об ошибке, что упрощает работу по их устранению.

Большинство функций здесь имеют комментарии, но, наверное, следует уделить внимание следующей части:

```
"""Example endpoint returning about info
   This is using docstrings for specifications.
   ---
   parameters:
     - name: about
       in: path
       type: string
       enum: ['all','version', 'author', 'year']
       required: true
       default: all
   definitions:
     About:
       type: string
   responses:
     200:
       description: A string
       schema:
         $ref: '#/definitions/About'
       examples:
         version: '1.0'
   """
```

Как видите, указанная часть, использующаяся для описания и документирования функций в Python, здесь описывает наш API. Здесь указан входной параметр (parameters), фактически часть имени в пути URL, он имеет тип string и может быть значением all, version, author или year (например, 127.0.0.1:5000/info/all). Более того, по адресу http://127.0.0.1:5000/apidocs можно получить документацию на API. Мы определяем тип About string в definitions и затем используем это определение в ответах responses. Тип ответа 200 OK, имеет описание и схему, говорящую о том, что в ответе json возвращается строка и даже приводится пример ответа на запрос version.

То же самое можно сделать, сославшись в документации к функции на файл, содержащий - - -, и далее содержимое, приводимое выше или в декораторе. На официальном сайте библиотеки приводится следующие примеры:

1.

```
from flasgger import swag_from

@app.route('/colors/<palette>/')
@swag_from('colors.yml')
def colors(palette):
    ...
```

2.

```
@app.route('/colors/<palette>/')
def colors(palette):
    """
    file: colors.yml
    """
    ...
```

Кроме того, приводятся и другие варианты использования определений Swagger, в том числе через словарь Python и др.

В файле sitepart.py в каталоге part содержится следующий код:

```
from flask import Blueprint,jsonify
# Создаем Blueprint для отдельной части веб API
sitepart = Blueprint("sitepart", __name__, template_folder='templates',static_folder='static')
# Возвращает цветовую палитру по имени палитры (rgb, cmyk ...)
@sitepart.route('/colors/<palette>/')
def colors(palette):
    """Example endpoint returning a list of colors by palette
    This is using docstrings for specifications.
    ---
    parameters:
      - name: palette
        in: path
        type: string
        enum: ['all', 'rgb', 'cmyk']
        required: true
        default: all
    definitions:
      Palette:
        type: object
        properties:
```

```
      palette_name:
        type: array
        items:
          $ref: '#/definitions/Color'
  Color:
    type: string
responses:
  200:
    description: A list of colors (may be filtered by palette)
    schema:
      $ref: '#/definitions/Palette'
    examples:
      rgb: ['red', 'green', 'blue']
"""
# содержимое цветов палитр
all_colors = {
    'cmyk': ['cian', 'magenta', 'yellow', 'black'],
    'rgb': ['red', 'green', 'blue']
}
# что возвратить, если URL all
if palette == 'all':
    result = all_colors
else:
# возврат в зависимости от имени палитры
    result = {palette: all_colors.get(palette)}
# преобразуем словарь в json строку и возвращаем ее
return jsonify(result)
```

Здесь используется код с официального репозитория Flasgger, как видим, схема использует тип Palette, являющийся объектом, который содержит массив цветов, каждый при этом цвет является строкой. Соответственно, для нашего сайта доступ к этой части API будет реализован через URL http://127.0.0.1:5000/sitepart/colors/all/ или http://127.0.0.1:5000/sitepart/colors/rgb/ и т. д. При этом вы получите, например, такой результат (рис. 3.1) или в формате json {

```
 "cmyk": [
   "cian",
   "magenta",
   "yellow",
   "black"
 ],
 "rgb": [
   "red",
   "green",
   "blue"
 ]
}
```


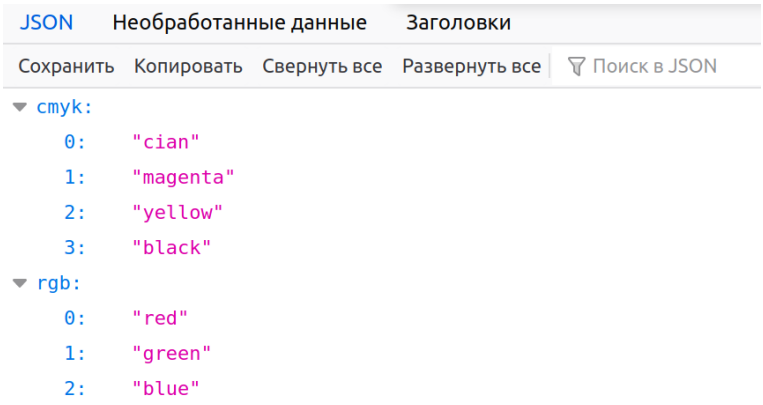

Рис. 3.1 – Результат запроса по URL к нашему API

Результат запроса apidocs отображен на рисунке 3.2.

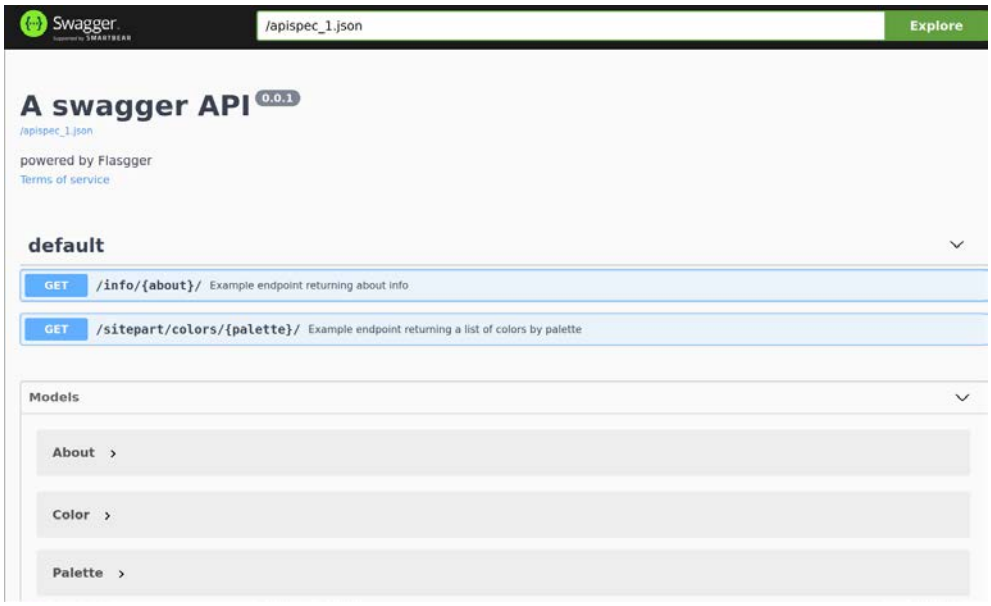

Рис. 3.2 – Пример отображения информации об API

3.2 Использование библиотеки flask_restplus для документирования веб-сервиса

Далее рассмотрим другую библиотеку, она несколько удобнее для ведения документации. Эта библиотека flask_restplus, кроме того она позволяет создавать отдельные части API, также как и Blueprint. То есть вы, например, можете создавать API с документацией, используя flask_restplus, а визуальную часть сайта реализовывать, используя Blueprint, создавая интерфейс пользователя и т. д. При этом для описания отдельных частей API используются неймспейсы.

Папки и файлы проекта.

part
part/static
part/templates
part/part.py
part/parttmpl.py
main.py

Содержимое файлов.

part.py

Более удобно в этом плане использовать модуль flask_restplus, создадим еще одну среду.

(proj2)pip install flask_restplus=0.13.0
(proj2)pip install Werkzeug==0.16.1

Структура папок здесь будет такой:

part
part/static
part/templates
part/part.py
part/parttmpl.py

main.py

```
main.py
from flask import Flask, Blueprint
from flask_restplus import Api, Resource

app = Flask(__name__)
api = Api(app = app)
# описание главного блока нашего API http://127.0.0.1:5000/main/.
name_space = api.namespace('main', description='Main APIs')

@name_space.route("/")
class MainClass(Resource):
        def get(self):
                return {"status": "Got new data"}
        def post(self):
                return {"status": "Posted new data"}
# подключение API из другого файла
from part.part import api as partns1
api.add_namespace(partns1)

from part.parttmpl import api as partns2
from part.parttmpl import templ as templ
api.add_namespace(partns2)
app.register_blueprint(templ,url_prefix='/templ')
app.run(debug=True)
```

Далее описан файл, содержащий отдельную часть API по другому URL http://127.0.0.1:5000/part/ и http://127.0.0.1:5000/part/id, под id здесь понимается любой передаваемый идентификатор в запросе URL.

part.py

```
from flask_restplus import Namespace, Resource, fields

api = Namespace('part', description='some information')
# описание возвращаемых полей
info = api.model('part', {
```

```
    'id': fields.String(required=True, description='The identifier'),
    'name': fields.String(required=True, description='The name'),
})

INFO = [
    {'id': '1111', 'name': 'Alex'},
]

@api.route('/')
class InfoList(Resource):
    @api.marshal_list_with(info)
    def get(self):
        '''List all / это описание появится в браузере на экране напротив get'''
        return INFO
# URL вида http://127.0.0.1:5000/part/1111   http://127.0.0.1:5000/part/2.
@api.route('/<id>')
@api.param('id', 'The identifier')
@api.response(404, 'id not found')
class InfoId(Resource):
    @api.doc(params={'id': 'An ID'}) # описание id в документации по адресу 127.0.0.1
    @api.marshal_with(info)
    def get(self, id):
        for idi in INFO:
            if idi['id'] == id:
                return idi
        else:
            return {'id': id, 'name': 'your name'},
        api.abort(404)
```

Здесь можно указать ограничение на id следующим образом:

@api.route('/<int:id>')

Таким образом, id должен быть целым, теперь, если вы запустите тот же проект, но с данной заменой, на id, взятую как строка с нецелым значением, мы получим возврат ошибки.

Декоратор api.doc позволяет документировать ваш API дополнительно, например, напротив параметра id будет выводиться дополнительная информация An id.

Можно для конкретного метода уточнить описания ответов. Например, если добавить responses методу get

@api.doc(params={'id': 'An ID'}, responses={404: 'ID Not Found'}), то вместо 'id not found', будет написано 'ID Not Found'.

Также мы видим применение декоратора маршалинга к запросу get, фактически это то же самое, что return marshal(db_get_todo(), model), 200, то есть применение маршалинга к данным, возвращенным из базы данных в соответствии с моделью данных. В нашем случае, например, return marshal({**'id'**: id, **'name'**: **'your name'**}, info), 200 или return marshal(idi , info), 200.

В Python термин «маршалинг» схож с термином «сериализация». Как видите, здесь объект «словарь» сериализуется в строку, передаваемую пользователю. Под маршалингом в компьютерных сетях понимается преобразование данных в формат, пригодный для передачи по сети, и затем преобразование в исходный формат. В общем случае маршалинг – это еще и запись состояния объекта так, чтобы можно было получить копию исходного объекта, путем автоматической загрузки определений класса объекта (фактически при маршалинге передаются еще и описания объекта и его код или расположение кода), в целом мы видим, что и здесь определения класса объекта загружаются автоматически. Маршалинг реализует представление объекта для получателя. При сериализации просто происходит преобразование объекта в битовую строку и обратно (в некоторых случаях сериализация – это частный случай маршалинга или способ реализации).

Здесь приводится пример шаблона http://127.0.0.1:5000/templ/.

parttmpl.py

```
from flask import Blueprint
from flask_restplus import Api
```

```
templ = Blueprint('templ', __name__, template_folder='templates',static_folder='static')
api = Api(templ)


@templ.route("/")
def index():
   return "template"
```

Опишем передачу, добавление и валидацию данных. Для этого рассмотрим простой пример работы с массивами. Здесь мы зададим некую модель массива, которая содержит размер массива и сам массив строковых значений. Разберите пример, который приведен ниже, можно вставить данный код до app.run. В данном случае можно провести валидацию данных, передаваемых в теле запроса post с помощью декоратора expect, данные должны соответствовать определениям модели. Маршалинг, как и ожидается, позволяет преобразовать данные в формат json.

```
from flask_restplus import fields
# определение модели данных массива
list_ = api.model('list', {
        'len': fields.String(required=True, description='Size of array'),
   'array': fields.List(fields.String,required=True, description='Some array'),
})

# массив, который хранится в оперативной памяти
allarray = ['1']
name_space1 = api.namespace('list', description='list APIs')
@name_space1.route("/")
class ListClass(Resource):
        @name_space1.doc("")
        @name_space1.marshal_with(list_)
        def get(self):
                """Получение всего хранимого массива"""
                return {'len': str(len(allarray)), 'array': allarray}

        @name_space1.doc("")
        # ожидаем на входе данных в соответствии с моделью list_
        @name_space1.expect(list_)
        # маршалинг данных в соответствии с list_
        @name_space1.marshal_with(list_)
        def post(self):
                """Создание массива/наше описание функции пост"""
                global allarray
                # получить переданный массив из тела запроса
                allarray = api.payload['array']
                # возвратить новый созданный массив клиенту
                return {'len': str(len(allarray)), 'array': allarray}
```

```
# модель данные с двумя параметрами строкового типа
minmax = api.model('minmax', {'min':fields.String, 'max':fields.String}, required=True, description='two values')
# url 127.0.0.1/list/mimmax
@name_space1.route("/minmax")
class MinMaxClass(Resource):
        @name_space1.doc("")
        # маршалинг данных в соответствии с моделью minmax
        @name_space1.marshal_with(minmax)
        def get(self):
                """Получение максимума и минимума массива"""
                global allarray
                return {'min': min(allarray), 'max': max(allarray)}
api.add_namespace(name_space1)
```

Можно выполнять запросы, используя Swagger. Для этого нажимаем кнопку Try it out.

Данные можно ввести в открывшемся поле (Edit value), например, **{"len":"3", "array": ["11","5","3"]}** (рис. 3.3).

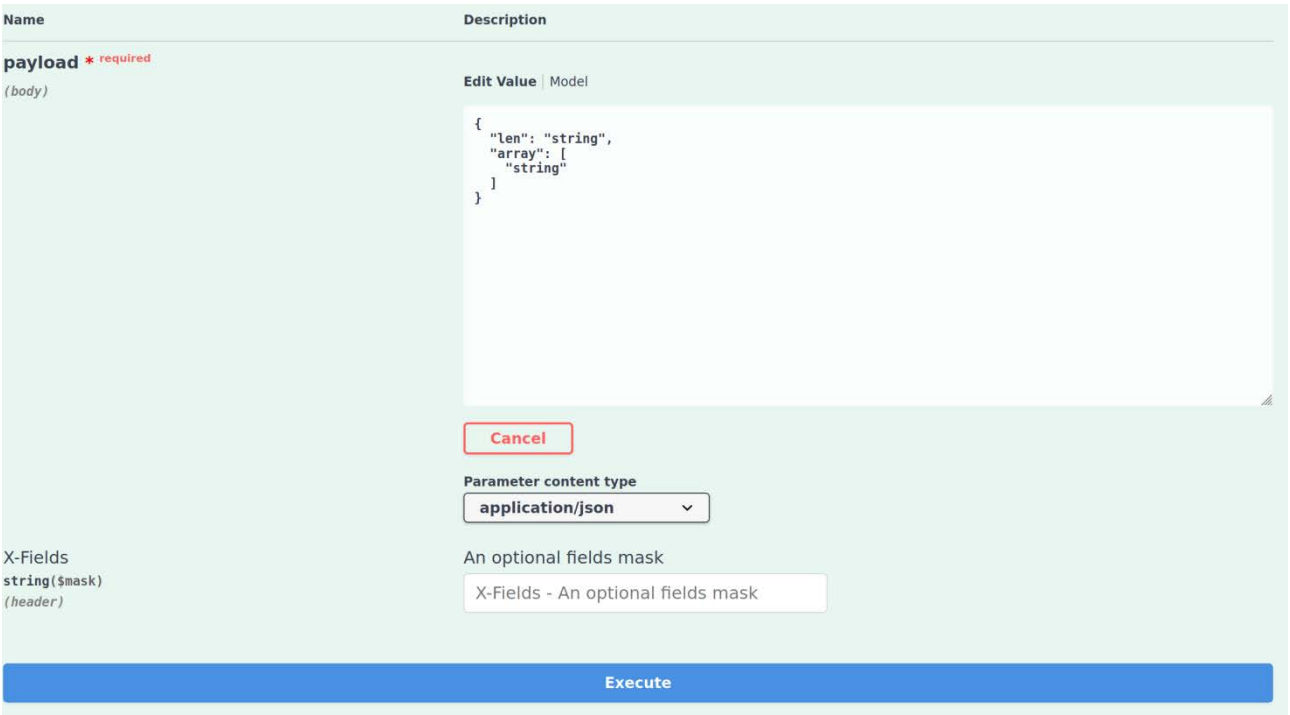

Рис. 3.3 – Ввод данных для запроса Post

Далее добавим пример, где данные передаются в запросе GET и в ответ возвращается результат. В данном примере в запросе GET передаются парамет-

ры, которые задают размер массива, минимальное и максимальное значение равномерного распределения.

```
from flask_restplus import reqparse
from random import random
reqp = reqparse.RequestParser()
# добавление аргументов, передаваемых запросом GET
# например GET http://127.0.0.1:5000/list/makerand?len=7&minval=1&maxval=12
reqp.add_argument('len', type=int, required=False)
reqp.add_argument('minval', type=float, required=False)
reqp.add_argument('maxval', type=float, required=False)


@name_space1.route("/makerand")
class MakeArrayClass(Resource):
        @name_space1.doc("")
        # маршалинг данных в соответствии с моделью minmax
        @name_space1.expect(reqp)
        @name_space1.marshal_with(list_)
        def get(self):
                """Возвращение массива случайных значений от min до max"""
                args = reqp.parse_args()
                array = [random()*(args['maxval']-args['minval'])+args['minval'] for i in
range(args['len'])]
                return {'len': args['len'], 'array': array}
```

Таким образом, мы в итоге получили функцию получения массива случайных значений. Пример достаточно простой, но он демонстрирует возможности фреймворка по валидации данных API и документирования. Итоговый набор запросов представлен на рисунке 3.4, очевидно, что можно использовать и запросы put, patch и delete.

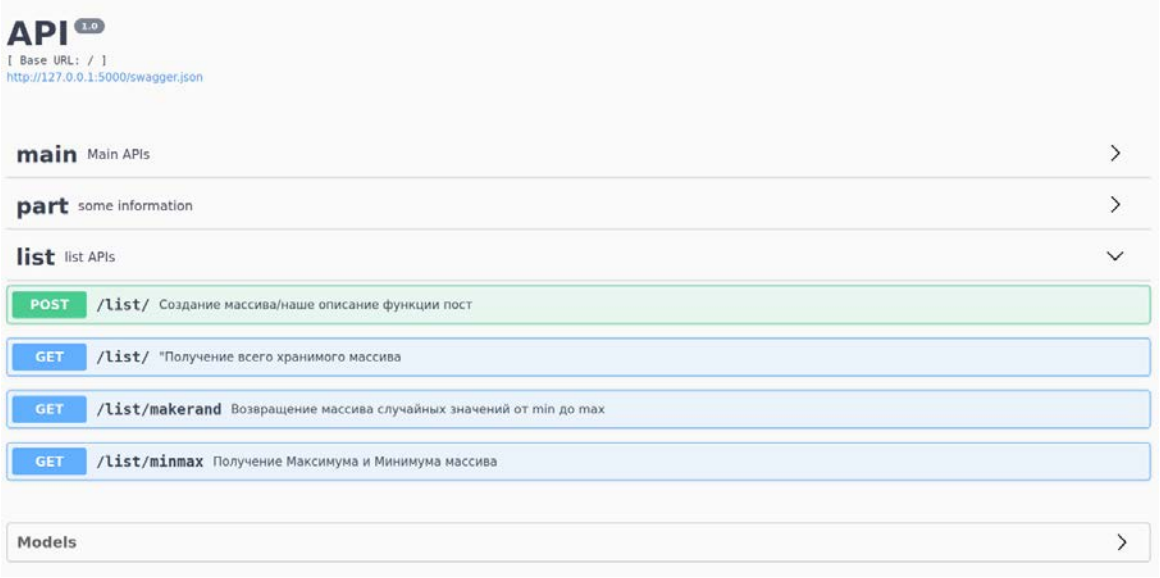

Рис. 3.4 – Пример Swagger документации API

Более подробное описание библиотеки flask_restplus можно найти по ссылке: https://flask-restplus.readthedocs.io/en/stable/.

3.3 Задание на лабораторную работу № 2

Необходимо повторить описанные в главе 3 примеры и создать свой собственный веб-сервис и API, реализующий возврат данных по заданию в соответствии с вариантом, где задана простая предметная область. Необходимо разработать простую модель данных, в соответствии с которой будут храниться данные, соответствующие предметной области. Количество полей должно быть более четырех, два-три поля могут быть строкового или другого типа, остальные – числового.

Веб-сервис должен предоставлять возможность сортировки по всем полям записей, выдавать среднее, максимальное и минимальное значение по числовым полям, добавлять, удалять записи и обновлять записи, например, по идентификатору.

Например, для предметной области «Продажа сервизов» можно выделить поля:

«Название сервиза»;

«Фирма-производитель»;

«Количество предметов»;

«Цена сервиза»;

«Материал».

Предметная область «Школьники, изучающие предметы в школе»:

«Ф.И.О.»;

«Название предмета»;

«Номер четверти»;

«Оценка»;

«Год начала учебы»;

«Возраст» и т. д.

Очевидно, можно выделить отдельную сущность и для нее назначить поля, но здесь мы не затрагиваем «Базы данных», но, если есть такое желание, то можно разработать модель данных и для хранения и запросов использовать СУБД.

Варианты для выполнения задания

№ варианта	Предметная область
1	Студенты, изучающие дисциплины в университете
2	Страны: язык и население
3	Продажа продуктов
4	Продажа напитков
5	Рынок автомобилей
6	Компьютерные комплектующие
7	Заболевшие опасным заболеванием по странам (выздоровевшие, погибшие)
8	Врачебные услуги
9	Услуги парикмахерских
10	Музыкальные произведения (длительность, жанр и т. д.)
11	Литературные произведения
12	Туристические поездки
13	Стоимость билетов на транспорт до городов
14	Киносеансы
15	Продажа квартир
16	Домашняя видеотека (продолжительность фильма, жанр и т. д.)
17	Футбольные матчи
18	Спортивные состязания
19	Выставки
20	Лотерея

4 ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ОТЧЕТА

Результатом выполнения каждой лабораторной работы является отчёт, сохраненный в файле с расширением pdf. Отчет должен содержать задание, скриншоты примеров работы веб-приложения, примеры выполнения задания, код приложения с комментариями, примеры выполняемых команд и используемых скриптов для деплоя на PaaS-системы, содержимое файлов переменных окружения и т. д., структуру каталогов и файлов веб-проекта.

Преподавателю также отправляется рецензия на предыдущий вариант работы, если она сдается повторно после исправления замечаний.

Оформление отчета должно соответствовать требованиям образовательного стандарта вуза ОС ТУСУР 01–2013 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления» (https://regulations.tusur.ru/documents/70).

Образец титульного листа представлен в приложении Б.

ЛИТЕРАТУРА

1. Суханов, А. Я. Разработка веб-сервисов для научных и прикладных задач : учеб. пособие / А. Я. Суханов. – Томск : ФДО, ТУСУР, 2021. – 246 с.

2. Официальная документация Flask [Электронный ресурс]. – URL: https://flask.palletsprojects.com/en/1.1.x/ (дата обращения: 25.03.2020).

3. Крупнейшая вики об WSGI [Электронный ресурс]. – URL: http://wsgi.org/ (дата обращения: 28.03.2020).

4. Python. – URL: https://www.python.org/ (дата обращения: 21.08.2020).

5. Бэрри, П. Изучаем программирование на Python / П. Бэрри. – М. : Эксмо, 2018. – 624 с. : ил.

6. Мэтиз, Э. Изучаем Python : программирование игр, визуализация данных, веб-приложения / Э. Мэтиз. – 2-е изд. – СПб. : Питер, 2018. – 496 с. : ил.

7. Miguel Grinberg. Проектирование RESTful API с помощью Python и Flask [Электронный ресурс]. – URL: https://habr.com/ru/post/246699/ (дата обращения: 28.01.2021).

8. Miguel Grinberg. Мега-Учебник Flask, Часть 1: «Привет, Мир!» [Электронный ресурс]. – URL: https://habr.com/ru/post/193242/ (дата обращения: 28.01.2021).

ПРИЛОЖЕНИЕ А

Примеры графиков

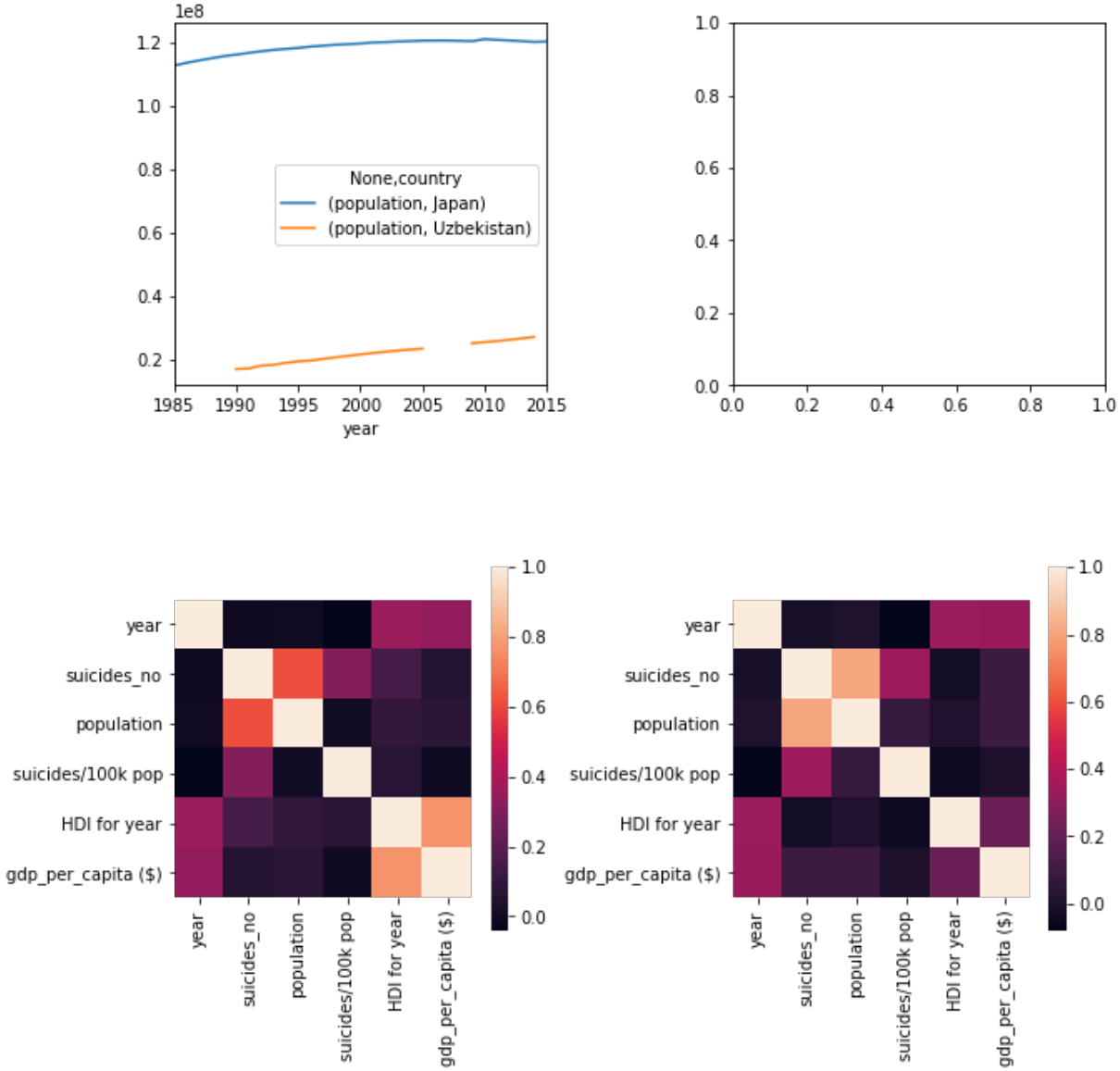

Рис. А.1 – Графики с использованием цветовой карты

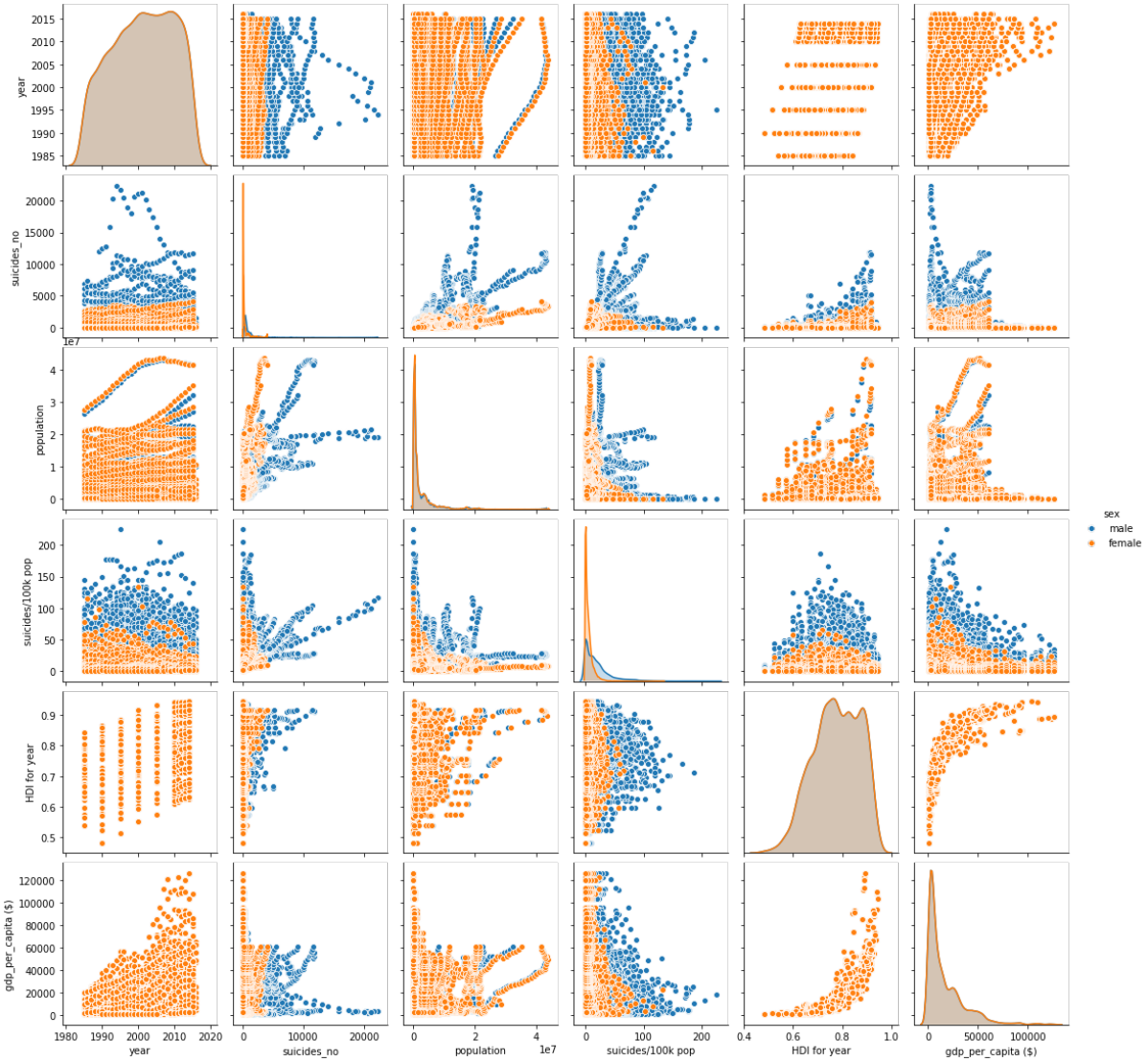

Рис. А.2 – График pairplot по полу

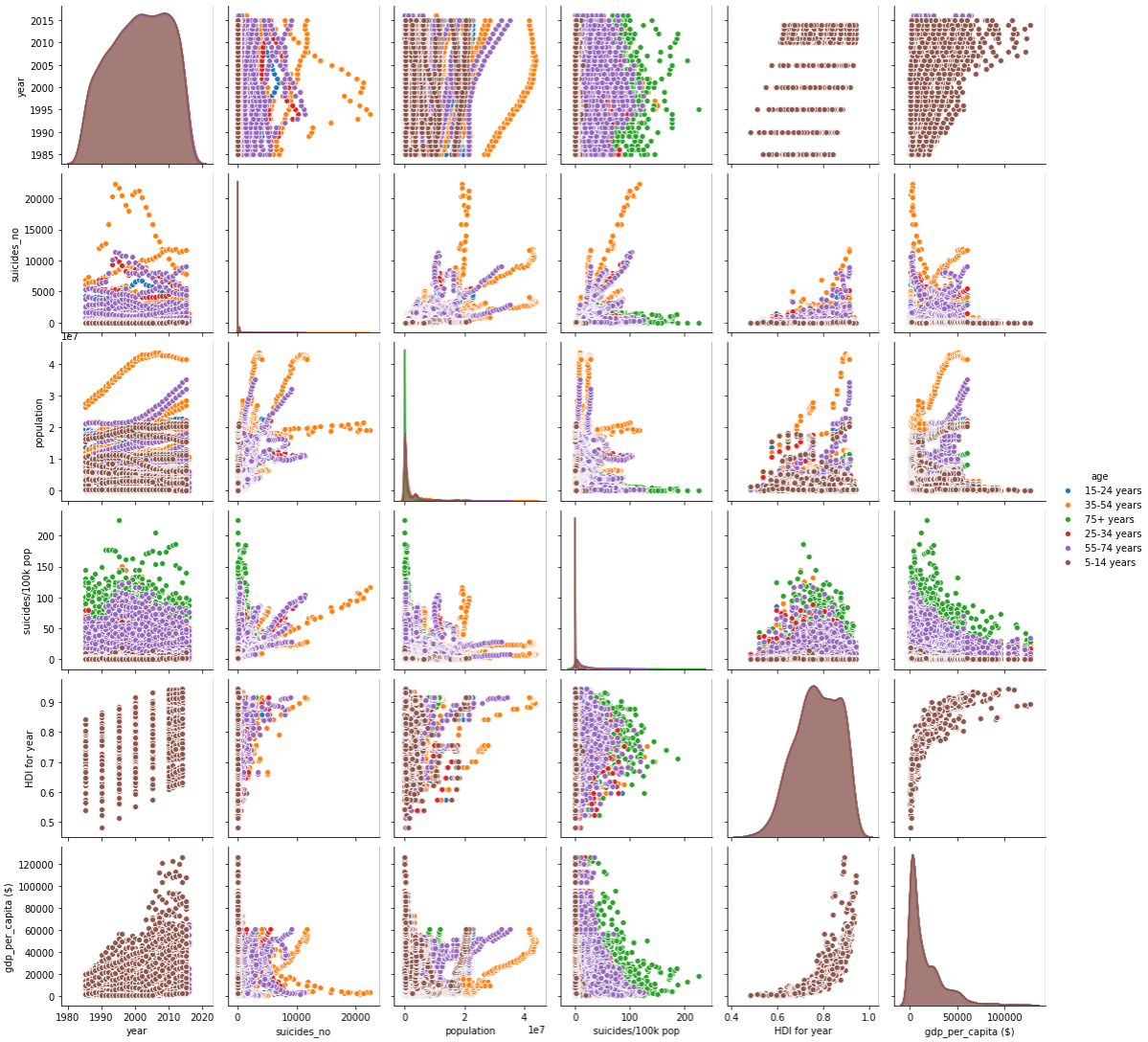

Рис. А.3 – График pairplot по возрасту

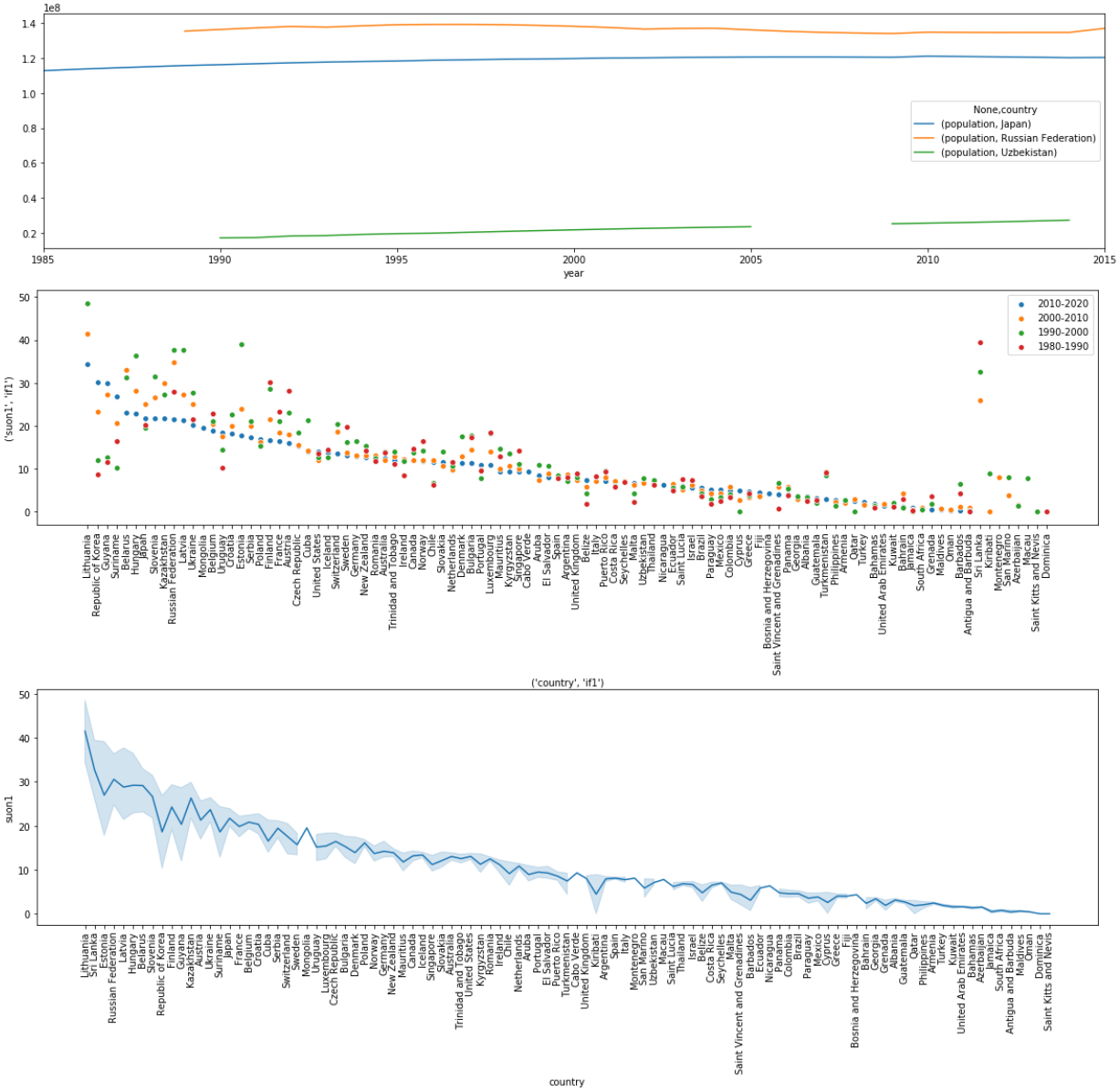

Рис. А.4 – График случаев по странам

ПРИЛОЖЕНИЕ Б

Образец титульного листа

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

НАЗВАНИЕ ЛАБОРАТОРНОЙ РАБОТЫ
Отчет по лабораторной работе по дисциплине
«Разработка веб-сервисов для научных и прикладных задач»
Вариант ___

Студент гр. _____
_____
(И. О. Фамилия)
«___»_____20__ г.
Руководитель
доцент каф. АСУ,
канд. техн. наук
_____ А. Я. Суханов
«___»_____20__ г.

Томск 20__