Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**Томский государственный университет
систем управления и радиоэлектроники (ТУСУР)**

А. Я. Суханов

РАЗРАБОТКА ВЕБ-СЕРВИСОВ ДЛЯ НАУЧНЫХ И ПРИКЛАДНЫХ ЗАДАЧ

**Методические указания
по выполнению лабораторных работ и самостоятельной работе
для студентов всех направлений**

Томск
2023

УДК 004.438:004.771:004.55(075.8)

ББК 32.973.4
С910

Рецензент:

Исакова А.И., доцент кафедры Автоматизированных систем управления ТУСУР, канд. техн. наук

Суханов, Александр Яковлевич

С910 Разработка веб-сервисов для научных и прикладных задач : методические указания по выполнению лабораторных работ и самостоятельной работе для студентов всех направлений / А. Я. Суханов. – Томск: ТУСУР, 2023. – 113 с.

Настоящее учебно-методическое пособие по выполнению лабораторных работ и самостоятельной работе студентов составлено с учетом требований федерального государственного образовательного стандарта высшего образования (ФГОС ВО). Учебное пособие содержит задания на выполнение лабораторных работ, теоретические материалы, примеры программных реализаций на языке Python, а так же все необходимые указания для выполнения лабораторных и изучения предложенных технологий студентами всех направлений специальностей и бакалавриата.

Одобрено на заседании кафедры АСУ протокол № 11 от 23 ноября 2023 года.

УДК 004.438:004.771:004.55(075.8)

ББК 32.973.4

Оглавление

Введение

Данные методические указания предназначены для выполнения лабораторных работ по дисциплине «Разработка веб-сервисов для научных и прикладных задач» и разработаны с учетом требований ФГОС ВО для всех направлений подготовки.

Цель лабораторных работ: получение навыков разработки веб-приложений и веб-сервисов с использованием одного из фреймворков, использования 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:

$$V = (N) \bmod 20 + 1,$$

где V – искомый номер варианта,

N – номер в списке.

1 Лабораторная работа 1. Базовые возможности python и numpy

В данной лабораторной работе рассматриваются примеры работы с массивами numpy, графиками matplotlib, изображениями Image Pillow, изображениями в виде массов numpy. Вначале идет необходимая для выполнения лабораторной теоретическая информация и примеры. В конце главы идет задание на лабораторную работу.

1.1 Кратко о Python.

Python является интерпретируемым языком, в силу этого многое можно выполнять в командной строке. Достоинством Python является достаточно простое написание кода, код выглядит лаконичным. Проще реализовать программу или проект на python, получить первые результаты, потому язык изначально распространился в научной среде.

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

Основным, наверное, недостатком python является то, что при написании собственного кода для выполнения простейших вычислительных задач, не используя возможностей массивов numpy и других библиотек, этот код 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 для python pypy. Для решения проблемы блокировки и возможности исполнения потоков на разных ядрах разрабатывается pypy-stm.

1.2 Начало работы.

Просмотреть описание, представленное ниже, прочитать основные моменты.

Предварительно кратко изучить **PEP 8 - 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 PEP-8, например, PyCharm.

Можно использовать и более простые IDE: Spyder, Idle.

Старайтесь при написании кода соблюдать соглашения PEP-8, кратко основные правила можно привести следующим образом.

Использовать четыре пробела для отступа (для сдачи лабораторных можно и табуляцию, но не рекомендуется). Длинные строки опускать под знаки (, {, [,:

```
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
)
```

Для имен классов использовать 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 - lower_case_with_underscores (слова из маленьких букв с подчеркиваниями).

Имена функций и переменных должны состоять из маленьких букв, а слова разделяться символами подчеркивания — это необходимо, чтобы увеличить читабельность.

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

1.3 Установка Python и настройка environment

Установка в операционной системе Windows или Mac OS реализуются самостоятельно, используя помощь поисковиков. Следуюет иметь в виду, что большая часть проблем с установкой должны решаться самостоятельно. Дальнейшие процедуры установки и команды приводятся для системы Linux определенной версии, всего лишь как пример. Так же приводится пример установки виртуальной машины Linux на VirtualBox, на случай работы этой системы как гостевой в системе Windows. Если у Вас итак установлен Linux, то вам понадобится лишь установить pycharm. Также можно попытаться установить виртуальную машину, сам Linux или устанавливать Python в Windows. Как установить VirtualBox можно найти на различных сайтах, дадим лишь краткое указание.

Прежде чем устанавливать VirtualBox в вашу Windows систему, зайдите в BIOS компьютера или ноутбука и в разделе настройки CPU включите поддержку виртуализации. На странице загрузки официального сайта 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 дистрибутив десктоп версию операционной системы ubuntu. В частности, скачать образ iso операционной системы версии 20.04 можно на сайте https://releases.ubuntu.com/20.04/. Создадим теперь виртуальную машину на базе скачанного образа. Выберем в VirtualBox машина->создать.

Выберем папку для сохранения виртуальной машины и тип операционной системы. Создать новый виртуальный жесткий диск. Можно выбрать гораздо меньше оперативной памяти, например, два, четыре гигабайта. (рисунок 1.1)

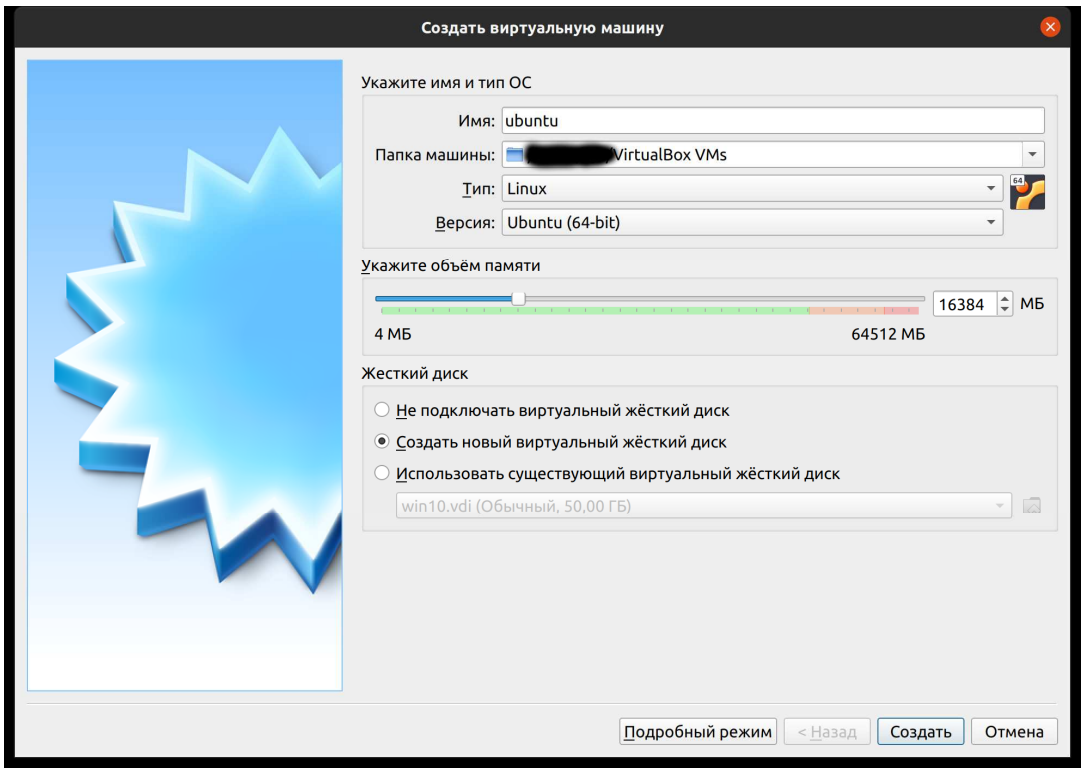

Рисунок 1.1 – Установка размера оперативной памяти для ОС и создание виртуального жесткого диска

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

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

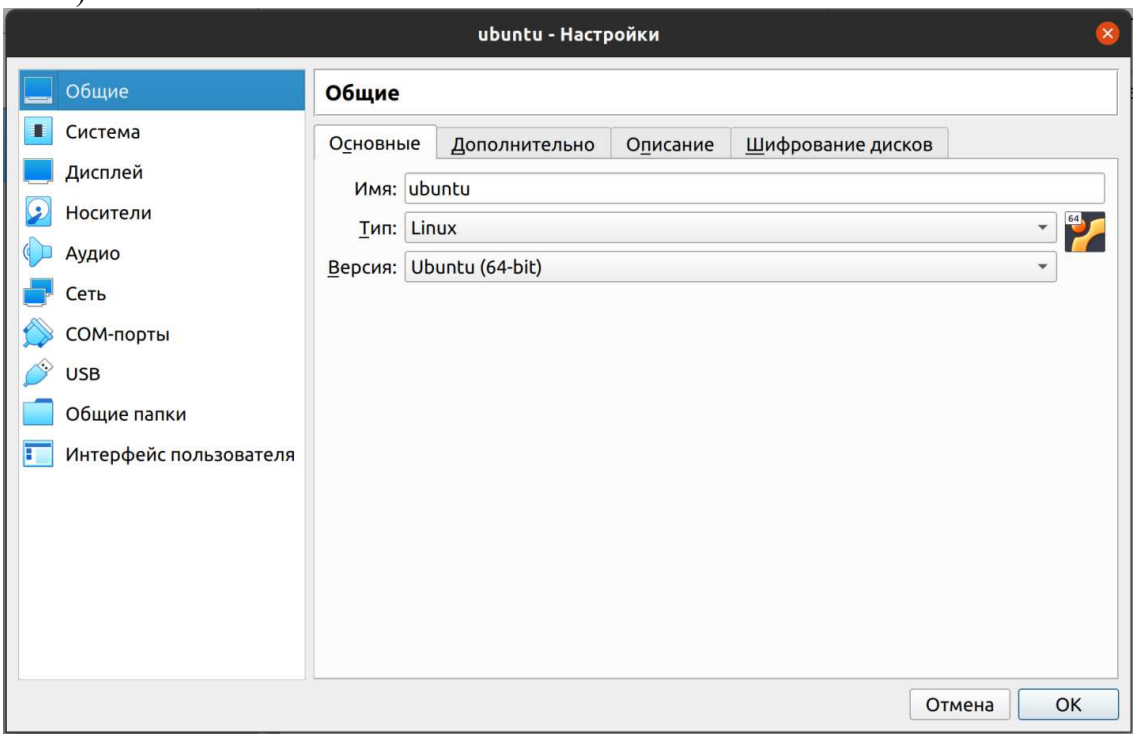

Рисунок 1.2 – Настройки виртуальной машины

Во вкладке общие папки можно выбрать разделяемые с хостовой операционной системой каталоги, нажав на кнопке зеленый плюс (рисунок 1.3).

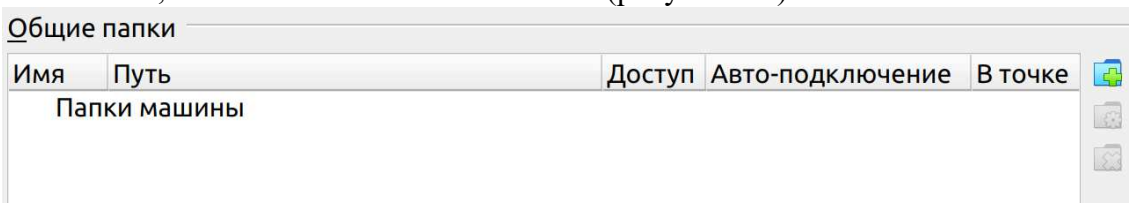

Рисунок 1.3 – Добавление разделяемых папок

Во вкладке носители можно добавить дисковый носитель, в частности наш скачанный iso образ, для загрузки и установки операционной системы, нажимаем на плюсик на фоне круглого диска (рисунок 1.4).

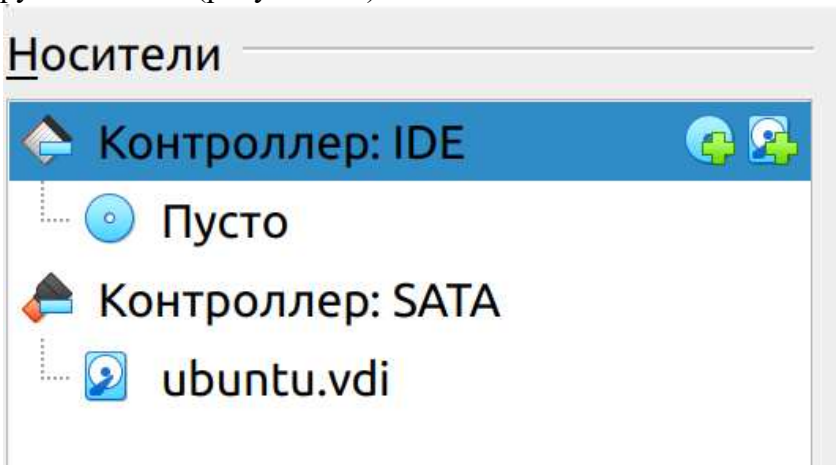

Рисунок 1.4 - Добавление устройства диска или другого носителя информации

Затем жмем добавить (рисунок 1.5) и добавляем скачанный нами iso образ.

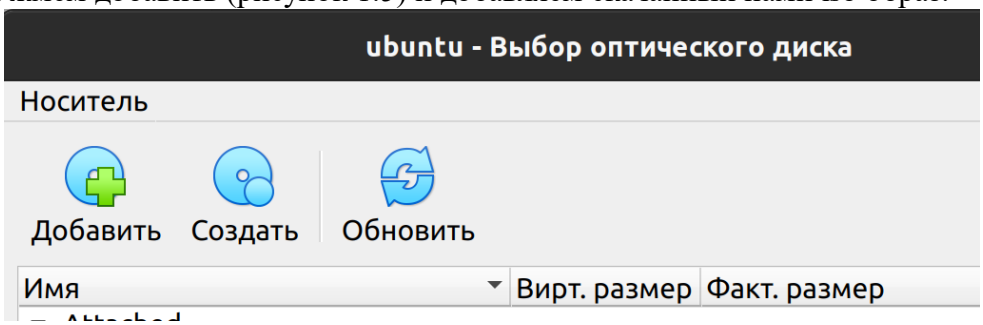

Рисунок 1.5 – Добавлении iso образа

После чего в окне Virtualbox выбрав нашу «виртуальную машину» нажимаем «запустить».

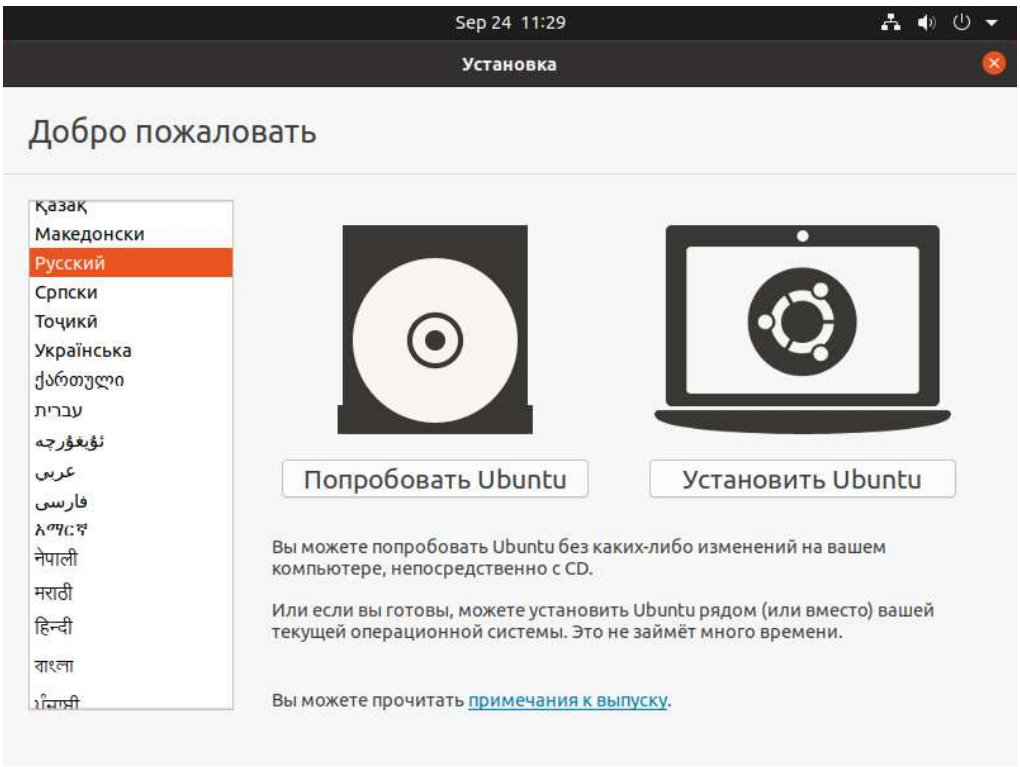

Рисунок 1.6 – Окно установки Ubuntu

Далее появится окно установки операционной системы ubuntu (рисунок 1.6), следуйте указаниям установщика. Выберете стандартную установку, задайте имя пароль пользователя в процессе установки.

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. В терминале Ubuntu Запустите команду:

python3 -V

чтобы посмотреть версию python. Если установлена версия python 3.8, то все нормально, можно продолжить. Если python не установлен, его нужно установить. Причем версию 3.8.

Если все-таки оказалось, что python 3.8 не установлен или вы устанавливаете его не в 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 python3.8.

sudo apt install python3.8

Для управления пакетами установим pip (pip3), являющейся системой 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 чего пройдет установка pip.

1.4 Настройка виртуальной среды

Виртуальные среды позволяют иметь изолированное пространство на компьютере 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. Виртуальные среды Python обычно создаются с использованием инструментов, таких как venv (встроенный в Python 3), virtualenv или conda. Также есть инструменты управления, такие как Pipenv объединяющие функциональность инструментов pip (установка пакетов) и virtualenv (создание виртуальных сред) в одном инструменте.

Расмотрим пример работы с venv.

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 этот каталог.

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 следующую 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 первый свой проект и запустить программу на python. Саму программу в принципе можно делать, используя любой текстовый редактор, очевидно, это не совсем удобно. Но простой первый проект давайте сделаем, 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 об ошибке, что модуль numpy неизвестен, его то как раз мы и 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, потому установим IDE. К сожалению, в виртуальной машине такое IDE может работать медленно, но вы можете так же продолжить работать в текстовом редакторе, либо использовать более легковесные IDE, а не предлагаемый нами далее pycharm. Можете установить свободно распространяемую версию pycharm в установщике ubuntu для обучения и изучения, либо профессиональную версию.

Выбрать программы ubuntu и далее Ubuntu Software (рисунок 1.7).

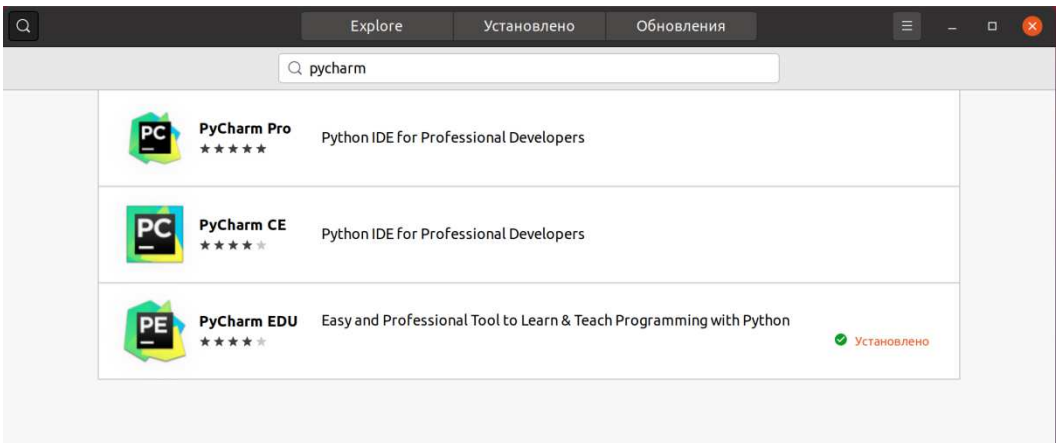

Рисунок 1.7 – Выбор в Ubuntu Software для установки IDE python PyCharm

Запустим Pycharm (рисунок 1.8) и зададим конфигурацию проекта.

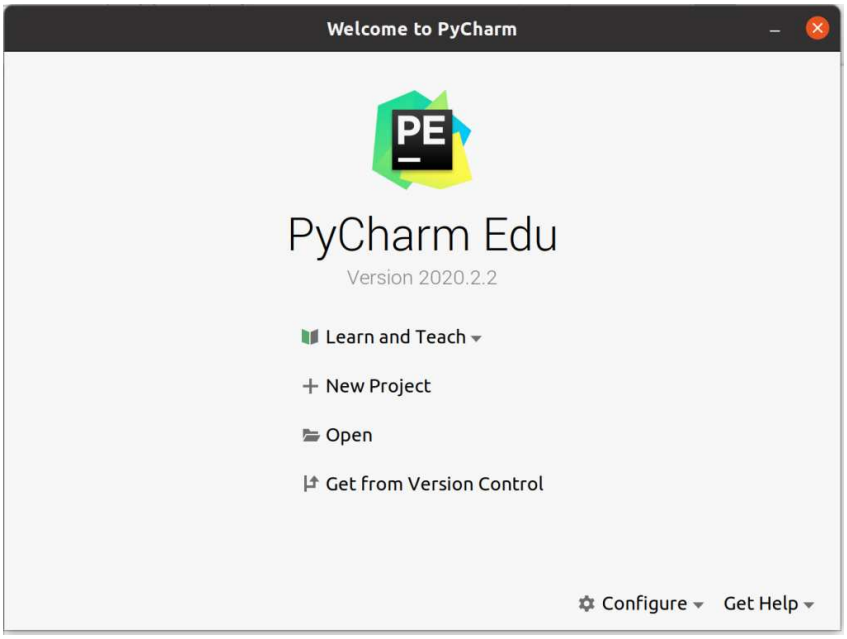

Рисунок 1.8 – Конфигурация, создания и открытие проекта

Можно, выбрав значок шестеренки Configure, установить еще одну environment.
Configure->Settings->PythonIntepreter->Add
Либо выбрать уже существующую среду, созданную нами (рисунок 1.9-1.10).

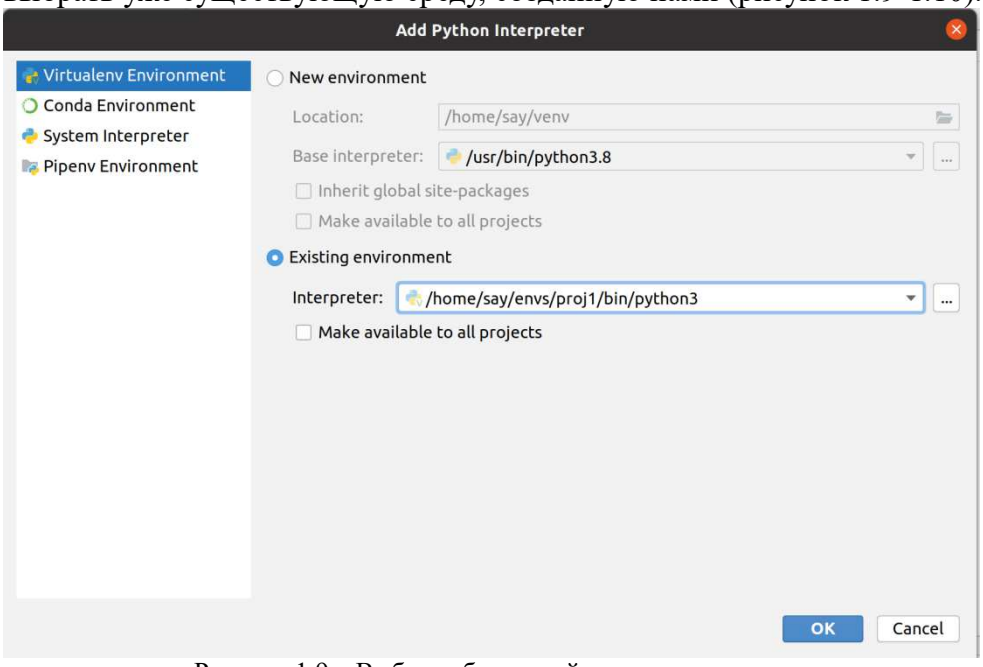

Рисунок 1.9 – Выбор собственной среды окружения

Далее создадим свой проект с указанной средой окружения.

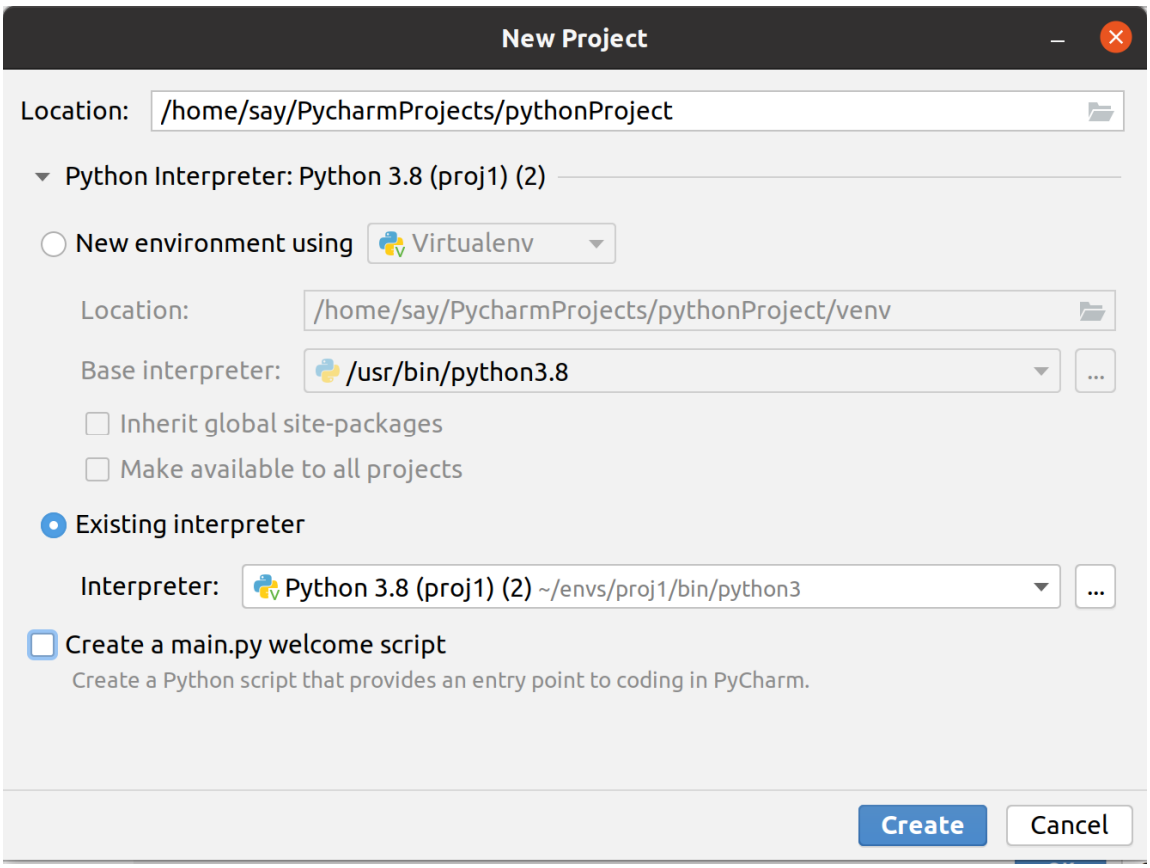

Рисунок 1.10 – Создание проекта в указанной среде окружения

Можно выбрать другое название проекта, и любую среду.

После создания проекта вы можете выбрать внизу вкладку terminal и в среде pycharm отобразится командная строка с соответствующей средой proj1.

Выбрав File->Settings можно посмотреть вкладку Project: PythonInterpreter 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

В консоли попробуйте запустить ниже представленные команды. Можно воспользоваться констольным режимом работы python. Либо запускать эти команды, записанные в файле с раширением py, например, в Pycharm. В pycharm так же есть консоль, для запуска команд интерактивно и терминал, который в том числе можно использовать для установки пакетов с помощью команды pip install пакет. В PyCharm в правом нижнем углу можно выбрать интерпретатор и его settings, и установить библиотеки в появившемся окне.

1.5.1 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, догадались, все переменные являются ссылками на объект. Если объект не используется, то есть переменные не ссылаются на этот объект, то объект может быть удален из памяти, обычно в подобных технологиях сборщиком мусора. Например, CLR или JVM (являющиеся jit-компиляторами, исполняющими байт-код) делают то же самое.

1.5.2 Простые примеры правил написания кода

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

В Python при именовании различаются заглавные и прописные символы.

В названиях переменных не использовать одиночные переменные O, l.

Название переменных 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 сзади используется для отличия с ключевыми словами 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)
```

1.5.3 Простые операции и форматированный вывод и ввод

Далее приводятся некоторые стандартные операции Python и возможности форматированного вывода на экран. Более подробное описание операторов Python можно найти в основном учебном пособии.

Реализуйте примеры в консоли Python. Консоль python доступна и в pycharm, нижняя вкладка Python Console.

```
#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
```

Для ввода строк можно использовать команду input. Как и все в python введенное значение будет представлять собой объект строкого типа, после операции «=», в приведенным ниже коде будет произведено связывание с переменной str. Кстати говоря, str является зарезервированным словом в Python, обозначающим объект класса строки. Например, str(2) преобразует число в строку, потому после такого присвоения вы потеряете стандартную переменную str и теперь она будет связана с созданным вами объектом.

```
#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
```

Выше приведен пример преобразования строки в число целого типа int.

Форматированный вывод позволяет вывести на экран объекты в удобно читаемом формате.

Здесь при выводе сначала указывается формат выводимых объектов. Описание этих объектов осуществляется в объекте строкового типа разделяя их через %, указывая количество знаков и тип объекта. Затем после значка % указывается кортеж в скобочках, где указываются сами выводимые объекты. В примере представлен форматный вывод трех объектов целого, строкового и целого типов, при этом указывается сколько знаков отводится под вывод данного объекта.

```
#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 позволяет также отформатировать вывод и создать форматированную строку, указывая формат объектов в фигурных скобочках.

```
>>> 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
```

Специальные знаки <,>, ^ позволяют выровнять текст внутри отводимого под символы поля, справа, слева и по центру.

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

Напомним, что операция * для строки продублирует символы строки указанное число раз.

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

F - строки предоставляют более удобное средство форматированного вывода, вплоть до использования внутри строки целых выражений.

```
# 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)]))
```

Пример использования lambda выражения.

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

Далее приведем пример использования форматированного вывода с использованием f строк.

print(f"index {index} {alphav.numpy():.{2}e} maxrew {smax_rewards} lossq {lossq:.{2}e} lossp {lossp:.{2}e} alph {alpha.numpy():.{2}e} border {border.numpy():.{2}e} acts {policies[self.buf_index-1:self.buf_index]} rew {cur_reward100:.{3}f} ")

Здесь выводятся числа с заданной точностью, в фигурных скобках стоит переменная, потом двоеточие затем точка указывающее на число с плавающей запятой, далее в фигурных скобаках указывается число отводимых знаков под цифры. Буква e указывает на экспоненциальный формат, f на формат с дробной частью числа. Как видим здесь выводятся даже целые массивы, если в соответствующих объектах есть функция преобразования в строку вывода.

Чтобы массивы numpy выводились с заданной точностью перед выполнением программы можно задать

numpy.set_printoptions(precision=4).

Precision указывает на количество цифр числа после запятой.

1.5.4 Работа с графиками

Реализуйте программу для работы с графиками. Для этого воспользуемся библиотекой matplotlib и модулем pyplot.

Для отображения графиков при запуске проекта в pycharm произведите установку пакетов указанных ниже (Linux Ubuntu), может и не понадобится, смотря какая оболочка, можно так же графики сохранять на диске, а не показывать в виде сплывающего на экране окна.

#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

В дальнейшем для работы с графиками нужно установить библиотеки numpy и matplotlib. После создания проекта можно в окне терминала (вкладка внизу, выбрать мышкой, Terminal) и в командной строке установить нужные библиотеки (рисунок 1.11).

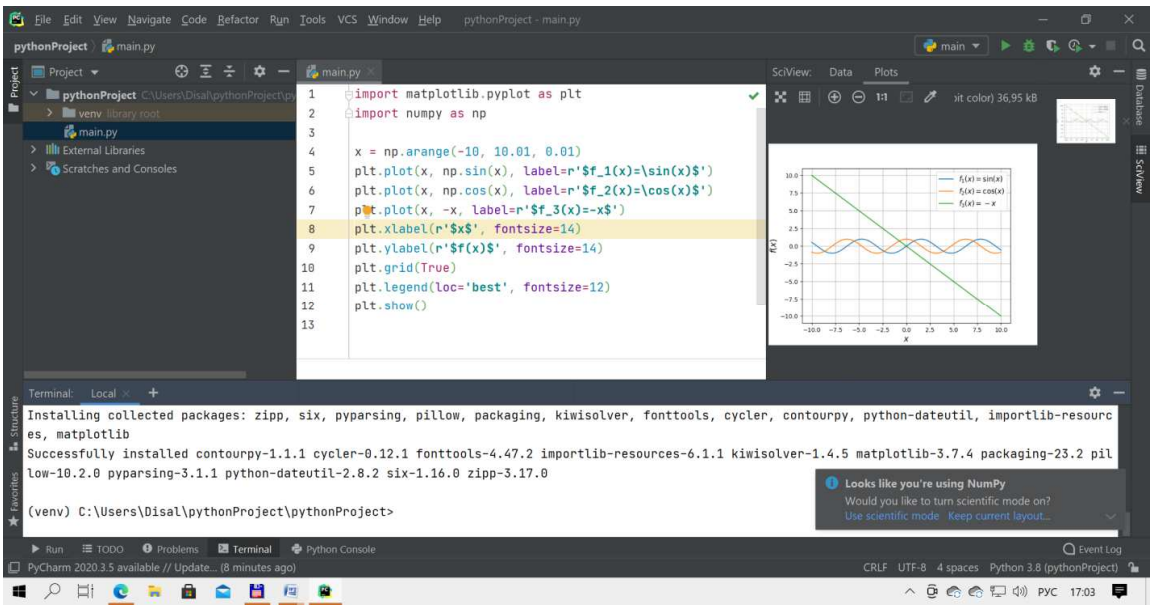

Рисунок 1.11 – Вид окна PyCharm с программой рисования графиков

>pip install numpy
>pip install matplotlib
В Linux и Windows это будет выглядеть почти идентично.

```
import matplotlib.pyplot as plt
import numpy as np
# задаем сетку по оси абсцисс с шагом 0.01
x = np.arange(-10, 10.01, 0.01)
# рисуем вместе графики синуса и косинуса, используем подписи с нижним индексом
plt.plot(x, np.sin(x), label=r'$f_1(x)=\sin(x)$')
plt.plot(x, np.cos(x), label=r'$f_2(x)=\cos(x)$')
plt.plot(x, -x, label=r'$f_3(x)=-x$')
plt.xlabel(r'$x$', fontsize=14)
plt.ylabel(r'$f(x)$', fontsize=14)
plt.grid(True)
plt.legend(loc='best', fontsize=12)
plt.show()
```

Запуск данного кода отобразит соответствующие графики, можно также сохранит график.

Разберем построчно данный код, сначала массив x заполняется значениями от -10 до 10 с шагом 0.01. Затем рисуется график синуса, косинуса и прямой. При этом указывается название которое будет у графика в легенде.

Далее приведен код, где ставятся подписи оси x и y. Здесь используется так называемая r строка, которая позволяет использовать спец символы внутри строки, в том числе можно отображать буквы греческого алфавита, степени и так далее. Пусть у нас будет не x а гамма и она будет измеряться в километрах в минус первой.

```
x = np.arange(-10, 10.01, 0.01)
plt.plot(x, np.sin(x), label=r'$f_1(\gamma)=\sin(\gamma)$')
plt.xlabel(r'$\gamma, km^{-1}$', fontsize=14)
plt.ylabel(r'$f(\gamma)$', fontsize=14)
plt.legend(loc='best', fontsize=12)
```

```
plt.show()
```

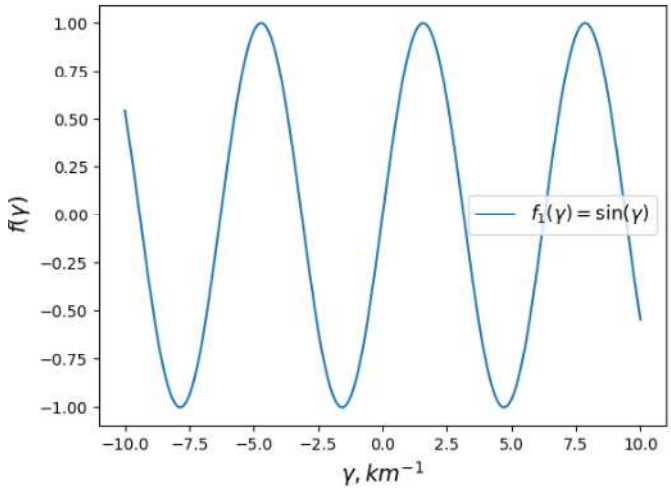

Рисунок 1.12 – График отображающий функцию с подписями осей с греческим алфавитом

Пример отображения гистограммы нормального распределения.

```
import matplotlib.pyplot as plt
# Генерация случайных чисел с нормальным распределением
mu = 0  # среднее значение
sigma = 1  # стандартное отклонение
data = np.random.normal(mu, sigma, 1000)
# Построение гистограммы
plt.hist(data, bins=30, density=True, alpha=0.7)
# Настройка осей и заголовка
plt.xlabel('Значение')
plt.ylabel('Частота')
plt.title('Гистограмма нормального распределения')
# Отображение гистограммы
plt.show()
```

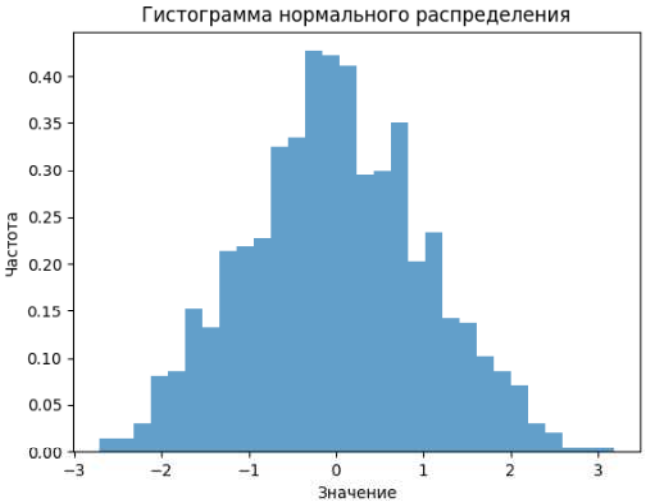

Рисунок 1.13 – Гистограмма нормального распределения

Приведем код, который позволяет рисовать графики с сеткой и уровнями ошибок.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
# Генерация случайных данных и зашумленной функции
num_data_points = 1000
x = np.linspace(0, 10, num_data_points)
errors = np.random.uniform(0.1, 0.3, num_data_points)
y = np.sin(x) + errors
# Определение оптимального шага прореживания для ошибок в зависимости от количества данных
step = max(1, num_data_points // 20)  # Пример: каждый 20-й элемент
# Выбор подмножества данных для редкого отображения уровня ошибок
error_levels = errors[::step]
x_subsampled = x[::step]
y_subsampled = y[::step]
# Построение графика данных с уровнями ошибок и оптимальным шагом прореживания
plt.errorbar(x_subsampled, y_subsampled, yerr=error_levels, fmt='o', capsize=3, color='blue', ecolor='red', elinewidth=1)
plt.plot(x, y, linestyle='-', marker='o', markersize=0, color='green', label = "данные")
plt.ylabel('Y')
plt.title('График данных с уровнями ошибок (оптимальный шаг прореживания)')
plt.legend(fontsize=16)
# включаем дополнительные отметки на осях
plt.minorticks_on()
plt.xlabel(r'$x$', fontsize=16)
# отображение только в заданном диапазоне координат
plt.xlim([0., 10.])
plt.ylim([-1., 2.])
# включаем основную сетку
plt.grid(which='major')
# включаем дополнительную сетку
plt.grid(which='minor', linestyle=':')
plt.tight_layout()
plt.show()
```

Далее приведем код, который позволяет отображать несколько графиков с несколькими осями по оси ординат, допустим, мы хотим отобразить по высоте давление, температуру и концентрацию газа, тогда нам может понадобится такой тип графика.

```
import matplotlib.pyplot as plt
import numpy as np
# Создание данных для графиков
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.tan(x)
# Построение графика с несколькими осями Y
fig, ax1 = plt.subplots()
fig.subplots_adjust(right=0.75)
color = 'tab:blue'
ax1.set_xlabel('X')
ax1.set_ylabel('Синус', color=color)
ax1.plot(x, y1, color=color)
ax1.tick_params(axis='y', labelcolor=color)
```

```
# Создание второй оси Y
ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Косинус', color=color)
ax2.plot(x, y2, color=color)
ax2.tick_params(axis='y', labelcolor=color)
# Создание третьей оси Y
ax3 = ax1.twinx()
color = 'tab:green'
ax3.spines['right'].set_position(('outward', 60))  # Сдвиг оси Y для третьего графика
ax3.set_ylabel('Тангенс', color=color)
ax3.plot(x, y3, color=color)
ax3.tick_params(axis='y', labelcolor=color)
plt.show()
```

1.5.5 Работа с массивами numpy

Массивы numpy предоставляют широкие возможности по обработке данных, представляющих собой числовые таблицы, хотя возможно обрабатывать и таблицы строк. Иногда данные многомерные массивы называют тензорами, условимся их так называть. В частности, библиотека tensorflow работает с подобными же тензорами, предоставляя похожие операции агрегирования многомерных данных.

Научимся на примерах выполнять стадартные математические операции, получение скалярного произведения, перемножение матриц и векторов, выборку по индексам, изучим использование срезов, в конце параграфа будут приведены примеры кода, реализующие вычисления интегралов численным методом трапеций и прямоугольников.

Приведем примеры создания массивов Numpy.

```
import numpy as np
#Создаем матрицу 3 на 3 из заданных числовых значений.
x = np.array([[1,2,3],[1,1,1],[-1,0,1]])
#Создаем матрицу 2 на 5 состоящую их нулей.
y = np.zeros(shape=(2,5))
#Создаем вектор размерностью 5 из едициц.
z = np.ones(shape = (5,))
#Создаем вектор с возрастающими значениями от 0 до 10 из 101 элемента, двумя способами.
a = np.linspace(start=0,stop=10.0,num = 101)
b = np.array([i*0.1 for i in range(0,101)])
print(f"x= {x}")
print(f"y= {y}")
print(f"z= {z}")
print(f"a= {a}")
print(f"b= {b}")
```

Умножим матрицу на вектор.

```
x = np.array([[1,2,3],[1,1,1],[-1,0,1]])
y = np.array([1,1,1])
res = np.dot(x,y)
print(res)
```

Умножим матрицу на матрицу.

```
x = np.array([[1,2,3],[1,1,1],[-1,0,1]])
y = np.zeros((3,3))
y[[0,1,2],[0,1,2]] = 1
res = np.dot(x,y)
```

Здесь мы создали единичную матрицу, создав матрицу из нулей, затем используя индексы, присвоили значениям на диаогнали значение 1.

То же самое можно сдалать так:

```
index = [i for i in range(3)]
y[index,index] = 1
```

Либо

```
y = np.eye(3)
```

Использование срезов. Для ознакомления со срезами можно так же обраиться к основному учебному пособию. Но в целом срез позволяет получить часть исходного тензора, по индексам от начального значения до конечного с каким-то шагом, например:

x[start1:stop1:step1,start2:stop2:step2…]

Кроме того можно использовать отрицательные значения индексов, которые интерпретируются как отступ от последнего значения, или длина массива минус значение индекса. При этом при переборе среза последнее значение индекса является не включительным, так же как и для range.

```
x = np.array([[1,2,3],[1,1,1],[-1,0,1]])
print(x[0:-1,0:-1])
```

Получим верхнюю левую часть матрицы 2 на 2.

Сделаем замену первой половины массива на вторую с помощью срезов, если число элементов нечетное, то центральный остается на месте.

```
COUNT = 11
x = np.array([i for i in range(COUNT)])
y = x[0:len(x)//2].copy()
x[0:len(x)//2] = x[len(x)//2+len(x)%2:]
x[len(x)//2+len(x)%2:] = y
print(x)
```

Здесь мы видим, как удобно использовать срезы, у нас нет циклов. Такая операция может быть оптимизирована в виде параллельных вычислений. Так же вначале использовался метод copy() создающий копию массива, например, для списков такого делать не надо, но для массивов numpy взятие среза не создает новый массив, а будет фактически на часть исходного массива. Можете проверить алгоритм, убрав метод copy().

Дополнительные удобные операции Numpy.

Умножение всех строк матрицы на элементы вектора, каждая строка умножается на свой элемент.

Здесь мы транспонировали матрицу, умножили ее на элементы вектора, и затем обратно транспонировали, чтобы получить исходную. Здесь T это транспонирование. Для транспонирования многомерного тензора так же можно использовать transpose.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([2,3])
z = (x.T*y).T
print(z)
```

Здесь мы увеличили размерность вектора y до 2 на 1, внутри каждого элемента строки получился отдельный элемент вектор размерностью 1, каждый такой элемент умножается на строку матрицы.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([2,3])[:, None]
z = (x*y)
print(z)
```

Если расширить вектор [None, :] получим внутри элемента строки вектор размерности 2.

Так же для расширения вектора существует операция expand_dims, можно указать ось по которой реализуется расширение.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([2,3])
y = np.expand_dims(y, axis=1)
z = (x*y)
```

То же самое можно сделать, используя изменение размерности массива с помощью reshape.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([2,3])
y = y.reshape((2,1))
z = (x*y)
print(z)
```

Умножение всех столбцов матрицы на элементы вектора.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([2,3, 4, 5])
y = y.reshape((1,4))
z = (x*y)
print(z)

# то же самое другим способом
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([2,3, 4, 5])[None, :]
z = (x*y)
print(z)
```

Очевидно, можно попробовать умножать поэлеметно строки матрицы и строки другой матрицы, или столбцы на столбцы.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
y = np.array([[2,3, 4, 5], [1,1,1,1]])
z = (x*y)
print(z)
```

Кроме приведенных в примере можно выполнять и другие арифметические операции, кроме того, массивы numpy содержат операции агрегирования данных, например, сумма (sum), среднее (mean), среднеквадратическое отлонение (std), дисперсия, минимум (min), максимум (max), индекс максимального и минимального в массиве (argmin, argmax), их можно определять по какой-то определенной оси.

Пример расчета сумм по столбцам, по строкам и по всему массиву.

```
x = np.array([[0,1,2,3], [1,2,3,4]])
a=x.sum(axis=0)
b=x.sum(axis=1)
```

```
c=x.sum()
print(f"a {a} b {b} c {c}")
```

Приведем пример, когда максимальным значениям в каждом столбце мы присвоим -1.

```
x = np.array([[4,1,6,3], [1,2,3,4], [5,2,4,1]])
a = x.argmax(axis=0)
x[a,[0,1,2,3]]  = -1
print(x)
```

Стоит заметить, что нельзя использовать срез в качестве индексов столбцов для данной задачи, так как срез будет интерпретироваться как перебор всех столбцов для всех значений индексов указанных в первом измерении, потому будет произведено присвоение всем элементам массива. Здесь же будут перебраны соответствущие парные индексы.

Аналогично можно получить значения соответствующие максимальным в столбцах

```
x = np.array([[4,1,6,3], [1,2,3,4], [5,2,4,1]])
a = x.argmax(axis=0)
z = x[a,[0,1,2,3]]
print(z)
```

Так же в качестве работу с массивами можно реализовать с использованием масок, маска должна иметь размерность самого массива, при этом там где маска имеет истинное значение то операция над элементом выполняется, где маска имеет ложное значение элемент остается неизменным, например, при выполнении операции присвоения.

```
x = np.array([[4,1,6,3], [1,2,3,4], [5,2,4,1]])
mask = (x > 2) & (x < 4)
x[mask] = 10
print(x)
```

Здесь всем элементам матрицы, где его значение больше 2 и меньше 4 будет присовено значение 10.

В основу расчета интеграла методом трапеций или прямоугольников положена достаточно простая идея того, что исходную функцию можно представить разбитой на достаточно малые интервалы, в которых можно интерполировать функцию прямой или константой, что даст возможность посчитать значение интеграла как сумму площадей фигур, которые образуются на каждом интервале – трапеция или прямоугольник. Далее приводятся функции реализующие расчет методом трапеций, используя массивы numpy и без их использования:

```
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} ")
```

Очевидно используя массивы Numpy, вычисления будут быстрее, так как операции могут выполняться параллельно и при этом не тратится время на работу интерпретатора.

1.6 Использование Python на Google Colab

Целью данной части работы является знакомство с Google Colab и изучение основных возможностей numpy, простейших функций обработки изображений и использования графиков. Эту же работу можно выполнять используя и Pycharm.

Изучение данного материала позволит вам легче освоить в дальнейшем современные библиотеки машинного обучения, библиотеки для работы с нейронными сетями на Python.

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

File - > Rename

Назовите свой проект. Вводите код в соответствующую ячейку для кода (рисунок 1.14).

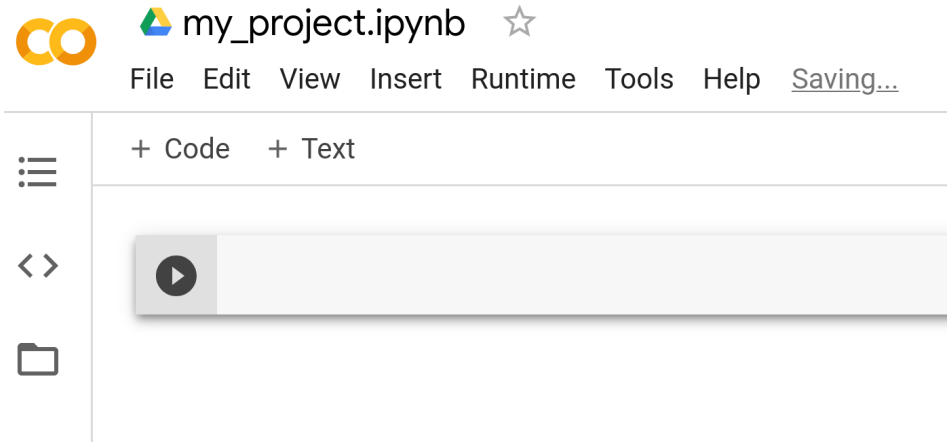

Рисунок 1.14 - Ячейка Cell в Colab

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

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

Можно непосредственно установить нужный пакет. Не забывайте перед командой ставить восклицательный знак (рисунок 1.15).

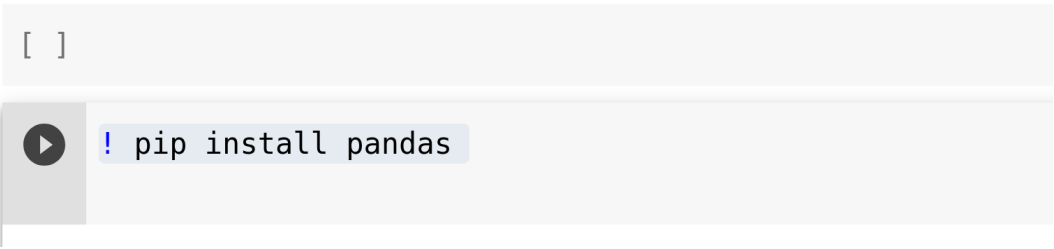

Рисунок 1.15 – Запуск команды из командной строки для установки пакета

Можно клонировать какой-либо проект с github.

! git clone https://github.com/ ……

Для подключения диска Google можно воспользоваться такими командами, которые приведены на рисунке 1.16.

Рисунок 1.16 – Подключение Google Disk

Либо нажав знак «папочка», и затем «Mount Drive» (Рисунок 1.17).

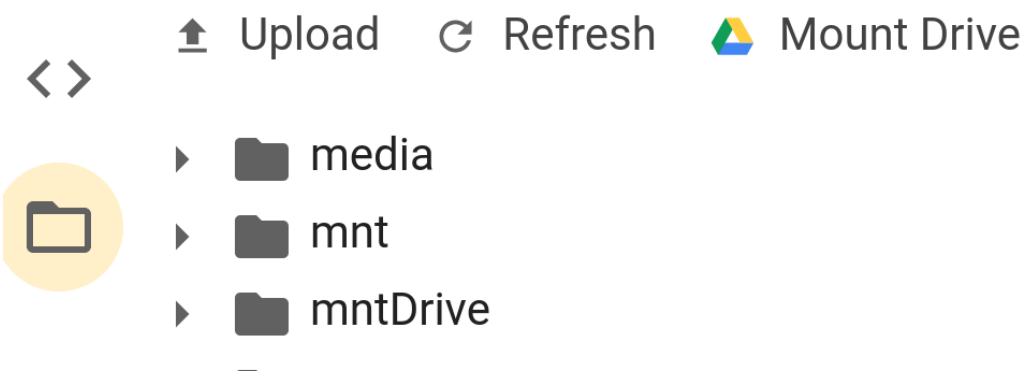

Рисунок 1.17 – Подключение Google Disk

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 (рисунок 1.18).

```
from google.colab import drive
drive.mount('/mntDrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /mntDrive
```

Рисунок 1.18 - Результат подключения диска

Далее приводится пример формы откуда необходимо копировать код авторизации (рисунок 1.19).

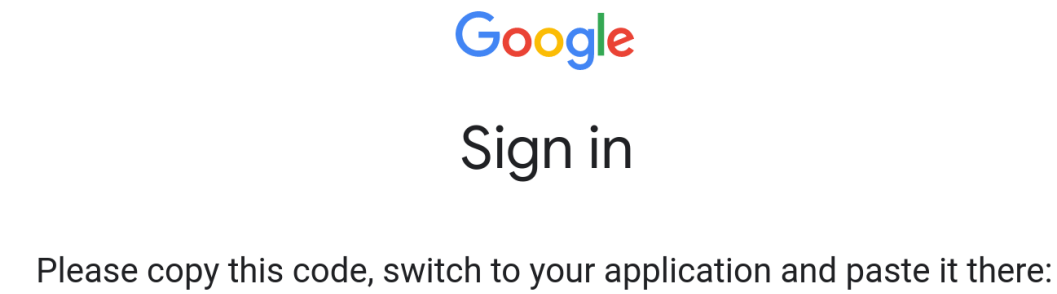

Рисунок 1.19 – Форма откуда копируется ключ авторизации

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

Теперь можно обращаться к файлам на Google диск.

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

Создадим директорию python, либо назовите как вам приятнее. Можем теперь записать туда нужный или нужные нам файлы с помощью upload, можно записать целую директорию upload directory (рисунок 1.20).

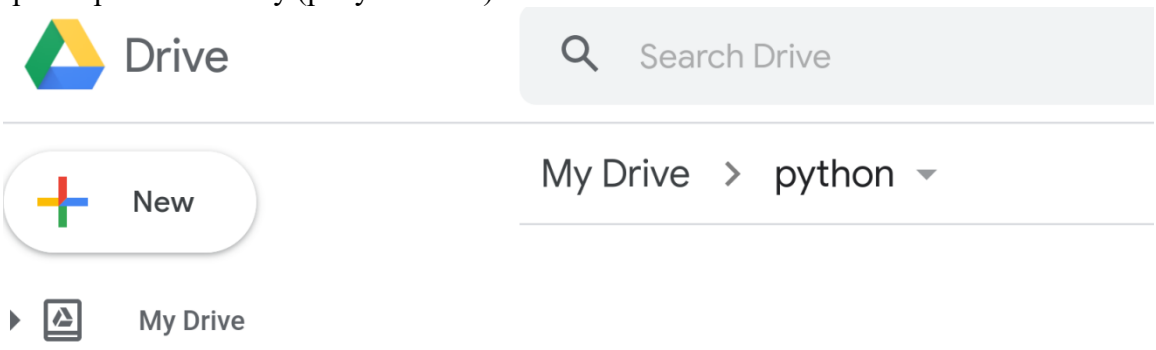

Рисунок 1.20 – Диск с созданным каталогом

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 слэш, если вдруг происходит ошибка.

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

print(x)

Перейдем на Google диск и запишем этот файл со скриптом на Google диск. Можно сохранить во временное хранилище на Colab папка Content, только нужно правильно прописать к ней пути.

Затем вернемся в colab cell и запустим наш скрипт (рисунок 1.21).

```
! 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]
```

Рисунок 1.21 – Результат работы скрипта

Попробуем импортировать этот скрипт как модуль (рисунок 1.22).

```
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]
```

Рисунок 1.22 – Результат работы скрипта

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

1.7 Работа с изображениями на Python

Для работы с изображениями на python можно воспользоваться библиотекой Pillow и массивами numpy. При этом изображения в виде массива могут рассматриваться как многомерный тензор, первые два измерения которого определяют высоту и ширину изображения, а третье измерение отвечает за цветовые карты r, g, b. Таким образом, размер такого тензора будет Высота * Ширина * 3, либо Высота * Ширина * 4, если используется дополнительный альфа канал. Над таким тензором допустимы все операции как над массивами Numpy, потому можно выполнять различные преобразования изображений.

Установка Pillow.

pip install Pillow.

Импортирование нужных нам пакетов из библиотеки.

```
from PIL import Image
```

Пример. Загрузка изображения с Google диска. Там должна быть папка python, dataf и файл 1.jpg.

```
image = Image.open("/mntDrive/My Drive/python/dataf/1.jpg")
```

Пример. Загрузка с локального диска Colab.

```
image = Image.open("/content/1.jpg")
```

Пример. Загрузка с локального диска из папки в текущем проекте, например в PyCharm.

```
image = Image.open("./dataf/1.jpg")
```

Будет создан объект Image, с которым можно работать как с изображением, используя например объект Draw той же библиотеки.

Пример. Преобразование изображения в массив numpy. Не забудьте подключить модуль numpy.

```
imx = numpy.array(image)
```

Пример. Отобразить изображение на графике.

```
import matplotlib.pyplot as plt
plt.imshow(imx)
plt.show()
```

Пример. Поменять разрешение изображения.

Используя метод объекта Image – resize:

```
imnew = image.resize(size = (20,20))
```

Используя массив Numpy берем срез уменьшающий изображение в три раза по обеим осям:

```
imx = imx[0:imx.shape[0]:3,0:imx.shape[1]:3,:]
```

Здесь мы берем срез с шагом три исходного тензора по осям высоты и ширины.

Пример. Смешивание двух изображений. Изображения считаны и преобразованы в Numpy массивы. Помещаем одно изображение в правый верхний угол, предварительно взяв минимальные размерности из двух изображений.

```
miny = min(imx1.shape[0],imx2.shape[0])
minx = min(imx1.shape[1],imx2.shape[1])
al = 0.1
imx = imx1.copy()
imx[0:miny,-minx:] = imx1[0:miny,-minx:]*al + imx2[0:miny,-minx:]*(1-al)
```

Пример. Наложение одного изображения на другое через маску. Маска представляет собой False где фон белый, а True где изображение, при умножении преобразуется в 1 и 0. Маска будет представлять собой матрицу той же размерности что и используемая для условия.

```
imx = imx1.copy()
mask = imx1[0:miny,-minx:] < 250
imx[0:miny,-minx:] = imx1[0:miny,-minx:]*(mask) + imx2[0:miny,-minx:]*(1-mask)
```

Пример. Наложение на изображение шума распределенного равномерно и последующее сглаживание результата Гауссовым фильтром.

```
import scipy.ndimage.filters as filt
# 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))
```

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 и т.д.

Пример. Изменение размера изображения с помощью интерполяции.

```
import scipy.ndimage.interpolation as interp
# 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)
```

Можно вывести исходные и измененные картинки на графиках и получится такой результат (рисунок 1.23).

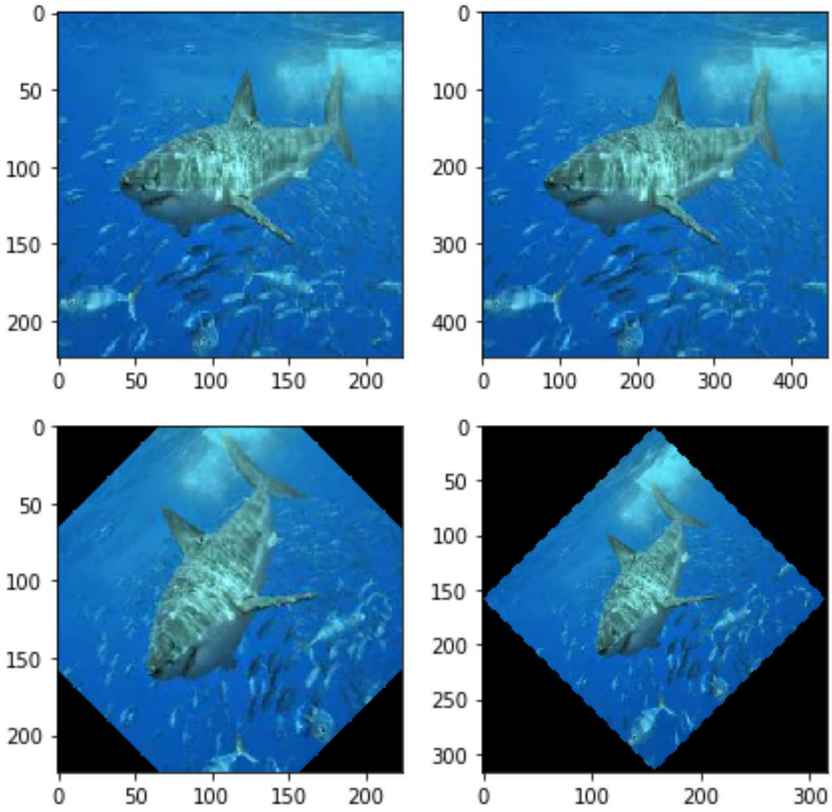

Рисунок 1.23 – Результат работы скрипта поворота

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

1.8 Задание

Ознакомиться с параграфами 1.1-1.7 методического пособия.

Требования к отчету. Отчет должен быть выполнен с использованием любого текстового процессора и преобразован в pdf документ и содержать Титульный лист, введение в котором указывается цель и задание. Затем ход выполнения работы с представлением основных результатов в виде графиков, изображений, примеров кода и их описания. Далее следуют Выводы и Приложение с листингом кода или программ.

Первая часть задания. Расчитать интегралы методом трапеций и прямоугольников с использованием срезов и без них (используя циклы) для функции заданной в вашем варианте. Реализовать отдельно функции с использованием срезов и с использованием циклов. Значения интегралов вывести, используя заданную точность, используя f строки и форматированный вывод в экспоненциальном формате и числом с дробной частью.

Рассчитать время работы каждой из функций. По возможности реализовать декоратор, позволяющий рассчитать время. Для изучения возможностей python можно обратиться к соответствующим сайтам или учебному пособию, глава 4.

Функции отобразить на графиках.

Для расчета интегралов задан интервал [a,b], количество точек n, и функция по вариантам.

Вторая часть задания. Провести обработку изображений.

а) Повернуть изображение на угол {angle} градусов. Отобразить на картинке крест цвета color {color = red, green, blue, same} (same — не менять цветовую составляющую). Крест рисовать используя срезы в виде двуз полос. Крест сдвинуть на {shift} пикселей по горизонтали от центра изображения.

б) Используя функцию transpose поменять местами цветовые карты изображений в порядке указанным в shuffle = {r,g,b}.

в) Используя transpose повернуть изображение на 90 градусов.

г) Используя операции фильтрации numpy обнулить в соответствии с указанным номером цветовой карты {color_filter = {r,g,b}} пикселы имеющие значение больше величины {alpha<1}, Значение <1 если интенсивность пикселей нормирована на 255, если нет, то альфа умножайте на 255.

д) Используя операции фильтрации и маску наложить водяной знак в виде другой картинки и наложить другую картинку используя маску. Маску получить по условию {color>betta}

е) Изображение «зашумить» на уровень {level}, выбрать самостоятельно два уровня зашумления и представить результаты сравнив исходное изображение и сглаженное (среднюю ошибку в процентах, для расчета использовать только срезы и агрегированные функции). Отобразить разницу сглаженного и исходного изображения.

Все задания с матрицами делать без использования циклов.

Третья часть задания. Выполнить третье задание, связанное с массивами numpy в вашем варианте.

Задания по вариантам. Указаны переменные которые надо задать в общем варианте.

1)

Интеграл.

[a,b] = [-1,1], n = 100, n = 1000, f(x) = |x-2|.

Изображения.

{angle = 20} {color =200, 0, same} {shift = 5} {shuffle = {g,r,b}} {color_filter = {r,g}}

{alpha = 0.2} {betta = 0.1}

Используя массивы numpy написать программу умножения половины строк матрицы на вектор длина которого равна половине строк матрицы.

2)

Интеграл.

[a,b] = [-3,1], n = 100, n = 1000, f(x) = |sin(x)-2|.

Изображения.

{angle = 40} {color =200, same, same} {shift = 0} {shuffle = {r,b,g}} {color_filter = {g}}

{alpha = 0.3} {betta = 0.2}

Используя массивы numpy написать программу умножения половины столбцов матрицы на вектор длина которого равна половине столбцов матрицы.

3)

Интеграл.

[a,b] = [-3,1], n = 100, n = 10000, n= 2000, f(x) = cos(x).

Изображения.

{angle = 10} {color =200, 100, same} {shift = -3} {shuffle? = {g,r,g}} {color_filter = {b}}

{alpha = 0.5} {betta = 0.5}

Используя массивы numpy написать программу умножения максимальных элементов в столбцах матрицы на первую строку матрицы.

4)

Интеграл.

[a,b] = [0,pi], n = 10, n = 10000, n= 2000, f(x) = -cos(x).

Изображения.

{angle = 40} {color =200, same, same} {shift = 0} {shuffle? = {r,r,g}} {color_filter = {b}}

{alpha = 0.5} {betta = 0.5}

Используя массивы numpy написать программу умножения индексов максимальных элементов в столбцах матрицы на первую строку матрицы.

5)

Интеграл.

[a,b] = [1,pi], n = 10, n = 10000, n= 2000, f(x) = ln(x).

Изображения.

{angle = 40} {color =same, 100, same} {shift = 10} {shuffle? = {b,r,g}} {color_filter = {r}}

{alpha = 0.5} {betta = 0.2}

Используя массивы numpy написать программу разницы первой и второй половины матрицы по строкам.

6)

Интеграл.

[a,b] = [0,1], n = 10, n = 100, n= 2000, f(x) = ln(x+1).

Изображения.

{angle = 10} {color =same, 100, same} {shift = 0} {shuffle? = {g,r,g}} {color_filter = {b,r}}

{alpha = 0.3} {betta = 0.2}

Используя массивы numpy написать программу разницы первой и второй половины матрицы по столбцам.

7)

Интеграл.

[a,b] = [0,1], n = 10, n = 100, n= 2000, f(x) = ln(x+1)+cos(x).

Изображения.

{angle = 30} {color =same, 100, same} {shift = 0} {shuffle? = {b,r,g}} {color_filter = {g,r}}

{alpha = 0.5} {betta = 0.2}

Используя массивы numpy написать программу, которая получает новую матрицу путем деления строк матрицы на диагональные элементы.

8)

Интеграл.

[a,b] = [0,1], n = 10, n = 100, n= 2000, f(x) = cos(x)*sin(x).

Изображения.

{angle = 20} {color =same, 100, 200} {shift = 4} {shuffle? = {b,g,g}} {color_filter = {r,b}}

{alpha = 0.3} {betta = 0.2}

Используя массивы numpy написать программу, которая получает новую матрицу путем деления строк матрицы на элементы побочной диагонали.

9)

Интеграл.

[a,b] = [0,1], n = 10, n = 100, n= 2000, f(x) = sin(x)-exp(x).

Изображения.

{angle = 10} {color =same, 200, same} {shift = 3} {shuffle = {r,r,g}} {color_filter = {r,g}}

{alpha = 0.5} {betta = 0.7}

Используя массивы numpy написать программу, которая получает на основе вектора нормированный вектор равный 1 по длине, на основе полученного вектора рассчитать энтропию используя элементы вектора как вероятности.

10)

Интеграл.

[a,b] = [-1,1], n = 10, n = 500, n= 2000, f(x) = sin(x)-exp(x).

Изображения.

{angle = 40} {color =1, 200, same} {shift = 3} {shuffle? = {b,r,b}} {color_filter = {g}}

{alpha = 0.2} {betta = 0.2}

Используя массивы numpy написать программу, которая умножает матрицу на себя транспонированную и умножает верхнюю левую часть матрицы на нижнюю правую, использовать квадрат части матрицы, а не треугольную часть.

2 Лабораторная работа 2. Возможности python для обработки данных

В данной части рассматриваются возможности доступа к сервисам, предоставляющим открытые данные. Изучение возможностей обработки и визуализации данных на python, некоторых возможностей библиотеки panda, matplotlib, seaborn. Изучение доступа к API сервиса data.gov.ru, data.world. В начале идет теоретическая часть и примеры обработки и в конце задание на лабораторную работу.

2.1 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.

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, образование, окружающую среда, энергетика, спорт, НАСА и многие другие.

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 что выложены для проведения соревнований.

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.

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

2.2 Обработка Kaggle датасетов с помощью Pandas

Kaggle предоставляет данные для анализа и NoteBook для написания кода на python. Кроме того, здесь проводятся соревнования по обработке данных. Выбрав необходимые данные на сайте можно получить сразу готовый код, в котором присутствует считывание csv файла. При этом даже не надо регистрироваться (хотя, лучше это сделать, чтобы не потерять нужные вам результаты). Можно также добавлять markdown текст.

Так же при выполнении задания вы можете скачать файл csv локально и обрабатывать его у себя.

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

Нажимаем New Notebook.

Add Data (ищем temperature, рисунок 2.1)

Discussion topic

do not average in ensemble, use temperature shaping

by Heng CherKeng

Severstal: Steel Defect Detection

do not average in ensemble, use temperature shaping

Dataset

Climate Change: Earth Surface Temperature Data

by Berkeley Earth

3 years ago • 85 MB • 911

Climate Change: Earth Surface Temperature Data

Рисунок 2.1 – 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 со следующим кодом и можем сразу приступать к анализу данных.

Листинг 1. Пример считывания датасетов из файлов в объект Pandas dataframe.

```
# 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")
```

Пример считывания файла библиотекой pandas. Для установки библиотеки локально можно восользоваться командой

```
pip install pandas
import pandas as pd
GlobalLandTemperaturesByCity = pd.read_csv("../input/climate-change-earth-surface-temperature-data/GlobalLandTemperaturesByCity.csv")
```

Будет создан объект Dataframe, который рассматривается как таблица, к каждому столбцу можно обращаться по его имени как к словарю.

Для более подробной работы с данным объектом можно обратиться по ссылке https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

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, странам начиная с 1700 годов, для таблицы с городами, указана так же широта и долгота.

Попробуем провести какой-либо анализ указанных данных.

2.2.1 Вывод общей информации describe и сведений о столбцах

Для получения общей информации о данных можно использовать следующий код:

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

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 датсете. По данной информации уже можно сделать различные выводы, например, присутствующие здесь максимальные температуры 39°C, и минимальные -42°C. Хотя это и вызывает некоторое сомнения, ведь в некоторых регионах бывают и куда меньшие и большие температуры. Максимальная ошибка для максимальных температур до 15°C, возможно, для данных указанных в 18, 19 веках. Но это нужно выяснить.

2.2.2 Просмотр только уникальных значений полей, без повторов

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.

2.2.3 Выборка и фильтрация данных по значению полей

Вероятно, здесь присутствуют не все страны. Попробуем уточнить данные по России. Заодно определим, какие стобцы 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 где-то и маленькими буквами.

```
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, которых присутствуют указанные символы, здесь мы использовали регулярное выражение.

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

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

Среди стран нет США (USA), возможно, указанная страна там 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'].

Для выборки и фильтрации по строкам в DataFrame с использованием библиотеки Pandas можно воспользоваться методом `loc` или условным индексированием. Вот несколько примеров:

Использование метода `loc` для фильтрации строк по условию:

```
# Фильтрация строк, где значение в столбце 'column1' больше 10
filtered_df = df.loc[df['column1'] > 10]
```

Использование условного индексирования для выборки строк:

```
# Выборка строк, где значение в столбце 'column2' равно 'value'
selected_rows = df[df['column2'] == 'value']
```

Комбинирование условий для фильтрации:

```
# Фильтрация строк, где значение в столбце 'column1' больше 5 и значение в столбце 'column2' равно 'value'
filtered_df = df[(df['column1'] > 5) & (df['column2'] == 'value')]
```

Эти методы позволяют выбирать и фильтровать строки в DataFrame на основе определенных условий. При этом `loc` используется для более сложных фильтраций, а условное индексирование - для более простых случаев.

2.2.3 Агрегирование и группировка данных

Построим графики, в 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, то 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 не очень оптимально, 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 затем отобразить данные об этих странах 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()
```

Нужно помнить, что группировка осуществляется по строковым значениям, а агрегированная операция среднее, дисперсия, сумма по числовым. Потому если в таблице остаются столбцы с нечисловыми данными при агрегировании может быть ошибка. Нужно взять выборку из столбцов с числовыми данными и те, по которым реализуется групповая операция. Либо реализовать преобразование числовых данных в строковые для группировки по ним.

2.2.4 Преобразование значений полей в столбцы Dataframe (stack)

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()
```

График средней температуры (рисунок 2.2). Если вы делаете локально обработку, то не забывайте вызвать plt.show(), чтобы увидеть график.

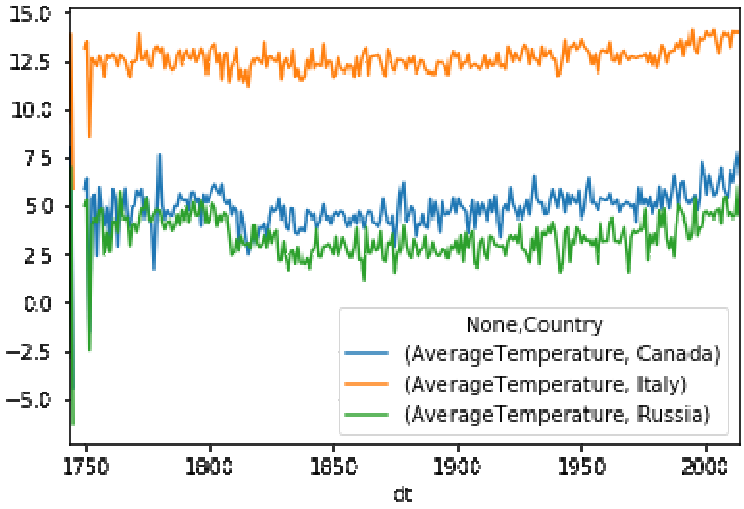

Рисунок 2.2 – Пример графика со средней температурой

По графику можно сделать вывод, что после 1950 года 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.

График погрешности в измерениях температуры (рисунок 2.3).

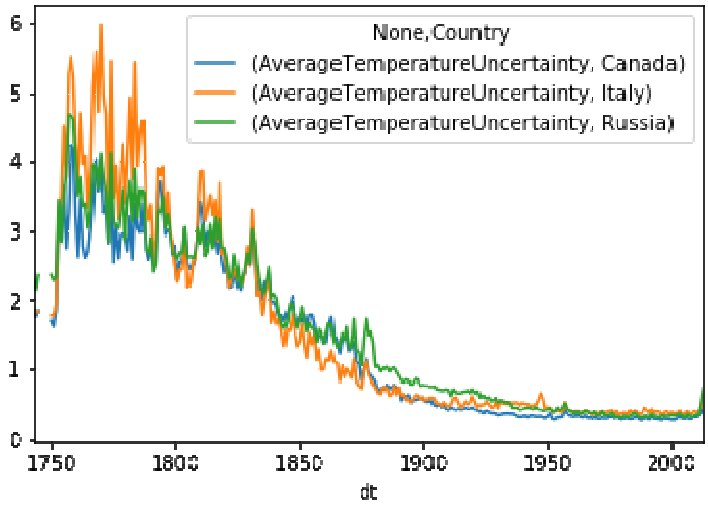

Рисунок 2.3 - Пример графика с погрешностями измерения температуры за все годы

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

Действительно, если запустить код, который размещен ниже, то получим, что измерения для Томска присутствуют только с 1825 года. При этом, при группировке мы получаем значение столбца City, 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 (рисунок 2.4).

```
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)
```

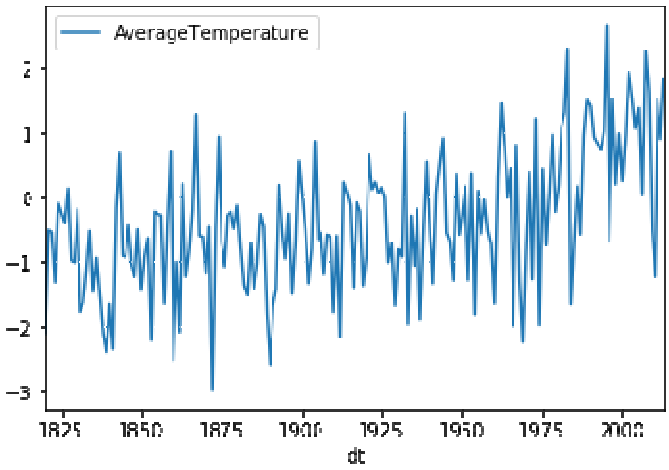

Рисунок 2.4 – Пример графика средней температуры по городу Томску

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 так же после 1950 года виден некоторый тренд роста температуры.

2.2.5 Отображение данных, используя seaborn и matplotlib

Вы так же можете использовать материалы и примеры главы 1 для отображения графиков. Информацию по seaborn можно найти здесь https://seaborn.pydata.org/.

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 расположенные по вертикали.
# axv - осевая система для отображения одного графика
# здесь две осевых 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 вы прошлый раз (рисунок 2.5, рисунок 2.6).

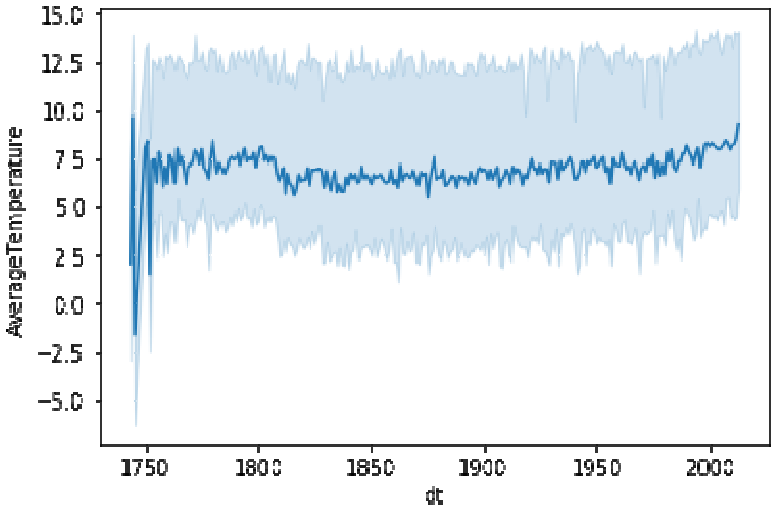

Рисунок 2.5 – Средняя температура на графике seaborn

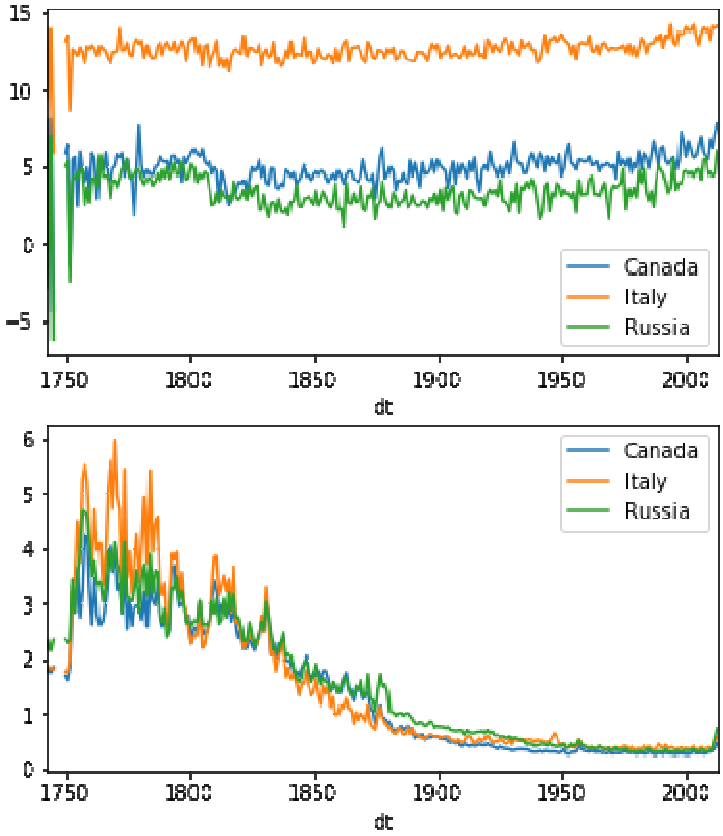

Рисунок 2.6 –Построение двух графиков вместе

Только теперь графики имеют подписи для стран и совмещены на одном поле.

2.2.6 Использование pairplot для отображения данных таблиц

Полезный график pairplot в seaborn, позволяет посмотреть соотношение данных попарно по отношению друг к другу (рисунок 2.7).

```
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)
```

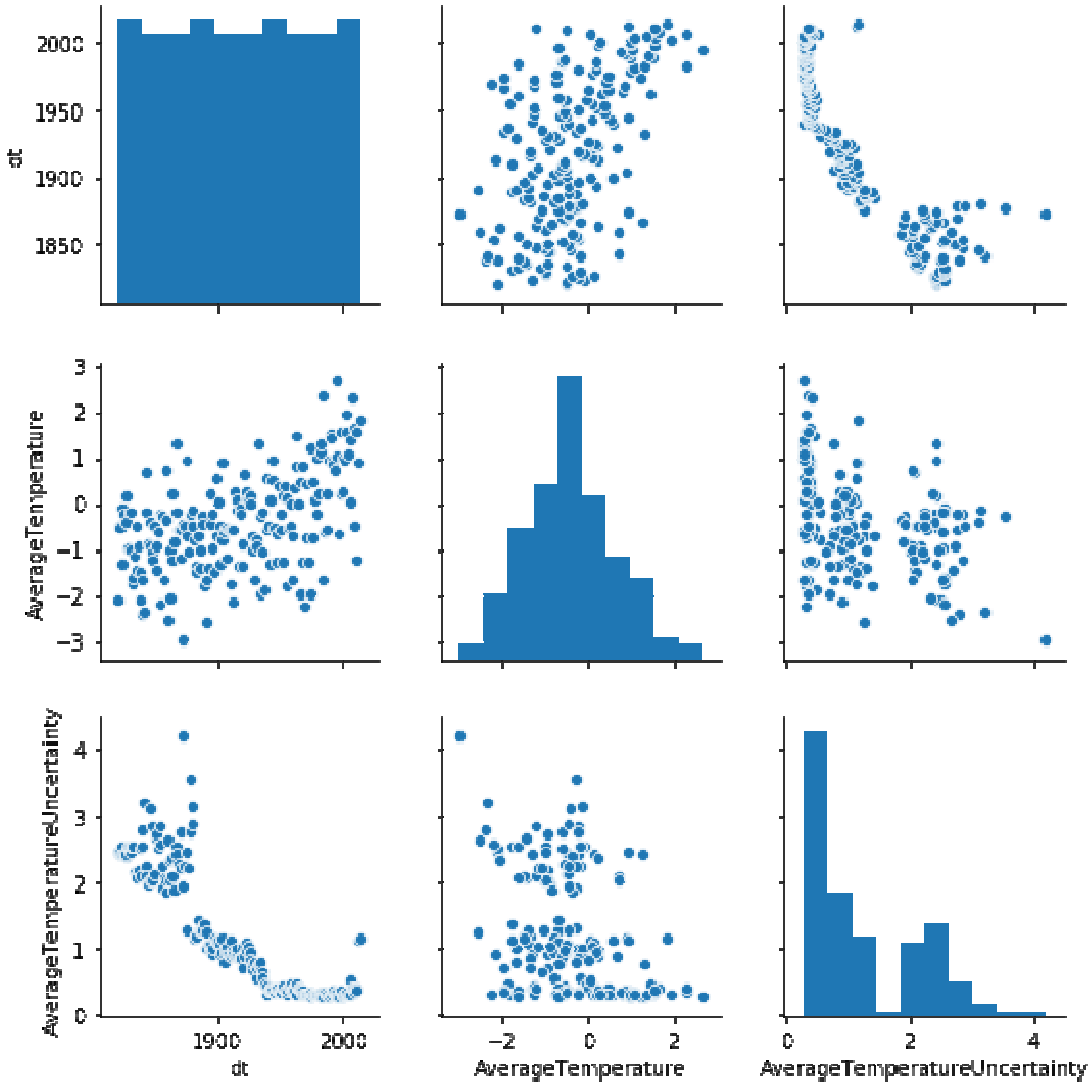

Рисунок 2.7 – График pairplot

Например, очевидно, что температура в Томске в среднем с годами выросла, неопределенность в определении температуры снизилась с годами. А вот средняя температура в Мехико выросла еще больше и тренд куда более четкий (рисунок 2.8).

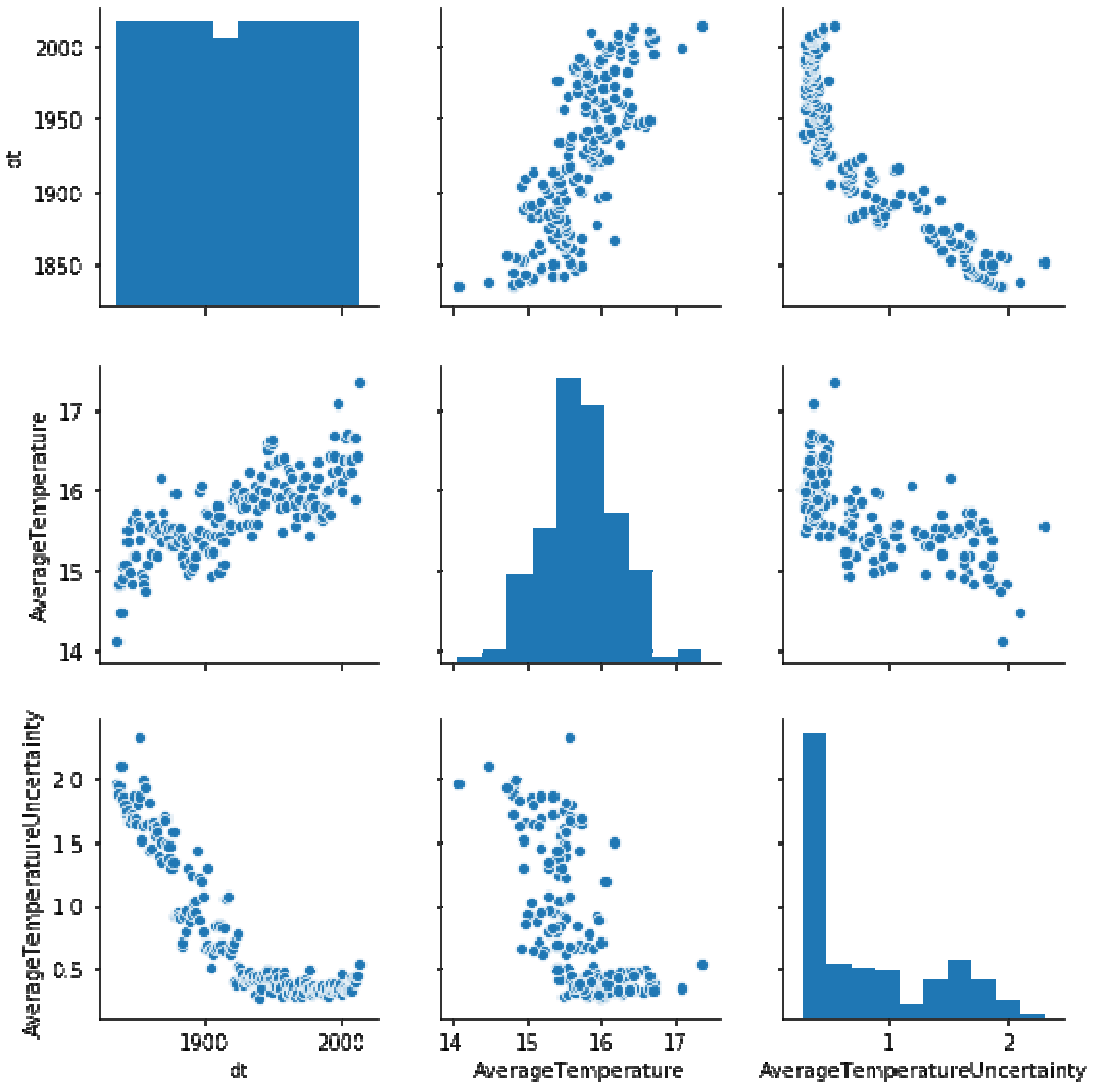

Рисунок 2.8 – График pairplot для города Мехико

2.2.7 Получение части датафрейма по столбцам

Использование iloc: Метод iloc позволяет выбирать столбцы по их числовому индексу. Например, чтобы получить только часть столбцов, можно использовать iloc с указанием нужных индексов столбцов. Например, df.iloc[:, 2:5] выберет столбцы с индексами от 2 до 4.

Использование loc: Метод loc позволяет выбирать столбцы по их именам. Для получения только части столбцов по их именам, можно использовать loc. Например, df.loc[:, 'column1':'column3'] выберет столбцы с именами от 'column1' до 'column3'.

Использование названий столбцов: Можно также выбирать столбцы по их именам напрямую. Например, df[['column1', 'column2', 'column3']] позволяет выбрать только указанные столбцы.

Таким образом, для получения только части столбцов в DataFrame в Pandas можно использовать методы iloc, loc или прямое указание названий столбцов в квадратных скобках.

2.3 Пример обработки еще одного датасета

Здесь можно найти примеры обработки данных, которые есть в задании. Также приводится пример поворота подписей осей, вывод графика с диапазонами вариаций и средним.

Еще один пример обработки данных на Kaggle datasets (рисунок 2.9).

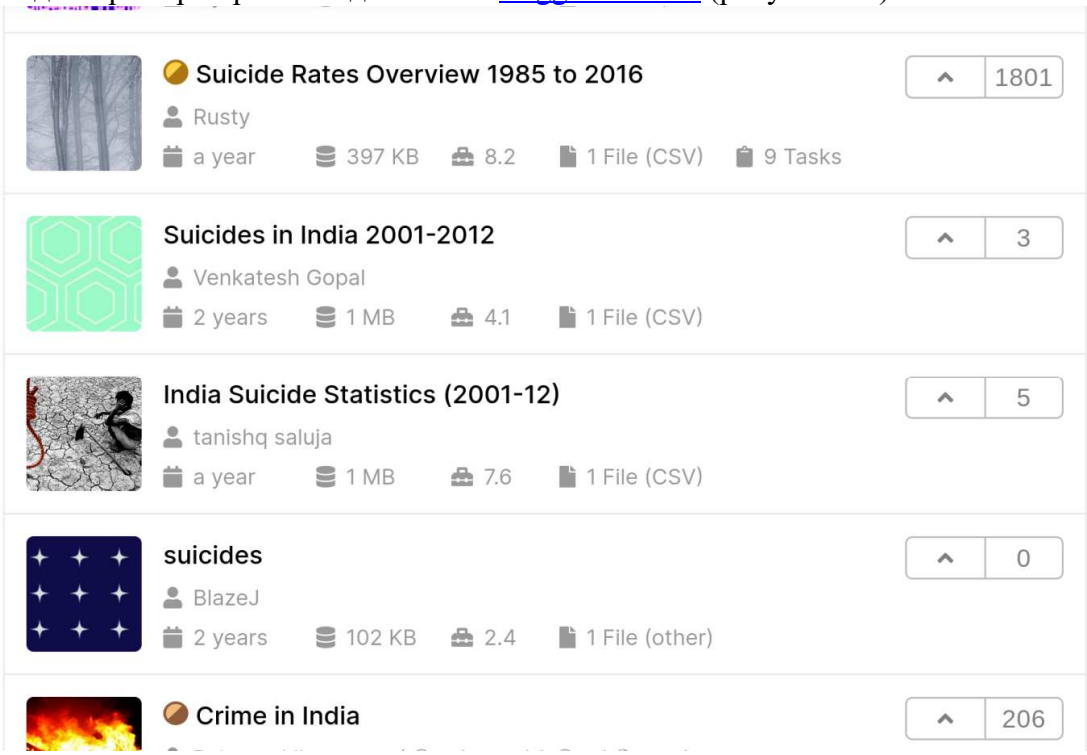

Рисунок 2.9 – Выбор датасета

Нажимаем New Notebook.
Add Data (ищем suicide)
Выбираем "Suicide rates Overview 1985 to 2016".
Соглашаемся на добавление данных
В результате получаем две ячейки со следующим кодом и можем сразу приступать к анализу данных.

Листинг 1.

```
# 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)
```

Там указан пол людей, возраст, 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).

Попробуем провести какой-либо анализ указанных данных. Кроме того, где-то есть пропуски данных Nan.

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

Описывает для каждого поля DataFrame различные характеристики, включая среднее, среднеквадратическое отклонение, максимальное, минимальное значение, квантили. Уже посмотрев на характеристики поля количество случаев на 100 тыс. населения можно сделать какие-то простые выводы, о среднем количестве случаев, о максимальном и минимальном. И сразу возникает простой вопрос, что это за население стран, если максимальное 43 млн. Посмотрим какие страны присутствуют в списке, получив только уникальные названия стран, и помести их в список.

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

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

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

И действительно данные же 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.

2.3.1 Группировка по данным

Попробуем получить данные по населению стран. Для этого нам понадобиться сделать группировку данных по какому-либо полю (по стране и году) и выполнение агрегированной функции, 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 к количеству 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'])
```

2.3.2 Преобразование части данных в столбцы unstack

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

2.3.3 Фильтрация по условию

Теперь попробуем отобразить полученную таблицу на графике. Очевидно, если 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 такие два графика (рисунок 2.10).

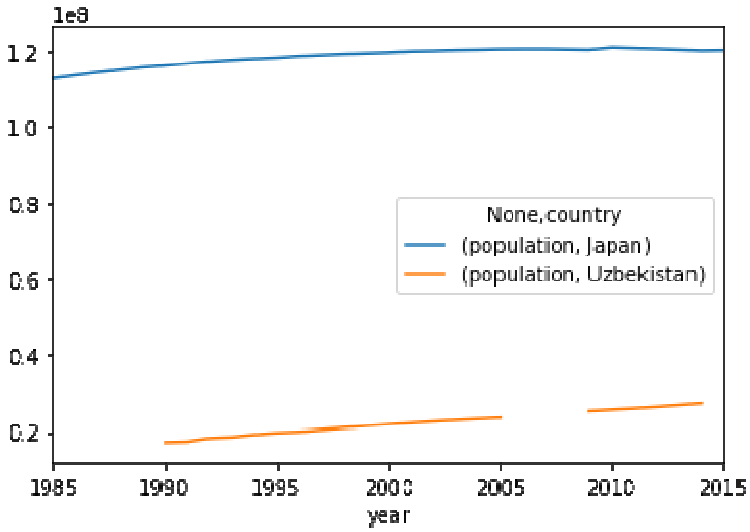

Рисунок 2.10 – График случаев по годам для двух стран

2.3.4 Проверка корреляции в данных, график тепловой карты и pairplot

Попробуем сделать какой-то 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 население старше 35 лет растет, потом с 2005 начинает снижаться, оно составляет большую часть населения представленных стран.

2.3.5 Консолидация данных по диапазонам значений

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. Будем использовать поле suicides_no, указывающим на количество случаев за год в данной стране, начиная с 1986 до 2016 года (Приложение А (рис.А.4)).

```
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 годы ухудшилась, как и во многих других странах бывшего СССР, с 2010 начала исправляться.

2.4 Задание

Найти и выбрать на kaggle датасет - файл с раширением csv, сделать обработку данных и отображение графиков по аналогии с пунктом 2.2, 2.3 методического пособия. Датасет должен содержать по крайней мере три, четыре числовых столбца и три четыре строковых столбца или с датами.

Отобразить все названия столбцов, указать, что они означают в этом датасете. Вывести общую информацию по датасету, используя describe. Нарисовать pair plot график по данным, или по части данных. Нарисовать scatter график сравнения стобчатых числовых данных. Сделать консолидацию, группировку и агрегацию по данным (минимум, максимум, среднее). Например, среднее по каким-то наименованиям какого-то из столбцов. Допустим, в столбце хранятся названия фирм, и доходы за разные годы, можно, например, вывести средний доход по каждой фирме. Некоторые данные хранимые в столбцах преобразовать в названия столбцов и вывести на графиках связанные с ними числовые величины. Осуществить фильтрацию данных. Осуществить вывод диапазонов изменения данных.

Получить сравнение по нескольким атрибутам на графиках, например, за годы или месяцы. Отобразить круговые диаграммы, bar диаграммы, цветовые карты, линейные графики. Придумать, где какие можно использовать. Использовать sns. Будет оцениваться вариативность применяемых методов анализа, разнообразие отображаемых графиков и их оформление.

Варианты датасетов для скачивания.

1) Экономические данные по странам

2) Результаты выборов по странам

3) Данные по странам, количество населения, возрастные группы, внутренний валовый продукт, уровень довольства.

4) Уровень производства различных ресурсов или товаров по странам мира за различные годы

5) Национальности по странам

6) Уровень рождаемости и смертности по странам или регионам

7) Производство продуктов различного вида питания по странам

8) Данные по добываемым промысловым видам рыб, если нет, то по количеству производимого мяса разного вида

9) Уровень потребления и выращивания злаковых

10) Цены на жилье, аренду жилья, покупку квартир

Если датасетов по варианту не нашли, можно взять подобный или предложить свой вариант.

3 Лабораторная работа № 3 «Разработка веб-приложения на python»

Целью данной лабораторной работы является изучение возможностей языка Python, возможностей по обработке данных, создание веб-приложения с использованием фрейморков Flask или FastAPI, системы контроля версий GIT и систем непрерывной интеграции Travis или GitAction, а так же деплой на какую-либо paas платформу, например, Heroku или Render. Перед выполнением задания лабораторной работы необходимо ознакомиться с теоретическим материалом. Можно предварительно реализовать локальное приложение и запускать локально в браузере и проверять его без коммитов и деплоя, рассматривая только код самого приложения и папки нужные для его работы. Затем реализовать тестирование, коммит и деплой. Для знакомства с описание стандартной архитектурой монолитного веб-приложения можно обратиться к учебному пособию.

3.1 Непрерывная интеграция (CI) для GitHub

Непрерывная интеграция (CI, англ. Continuous Integration) — 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, за счёт раннего его выявления.

Впервые концептуализирована и предложена Гради Бучем в 1991 году. 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.

3.2 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 это Travis CI https://travis-ci.org/, GitHub Actions, Circle CI. Рассмотрим для начала использование Travis CI. К сожалению, часто такие проекты требуют оплаты, потому рассматривайте данную информацию как ознакомительную, если у вас есть возможнось использать подобные системы реализуйте в них, либо рассмотрите дальнейшие указания, которые будут касаться GitHub actions (https://docs.github.com/ru/actions).

Travis CI 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

Развертывание 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, он 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.

3.3 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 будет включить непрерывную интеграцию для заданного проекта на сайтах Travis и AppVeyor. Нужно зайти на https://travis-ci.org под своим GitHub аккаунтом, 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.

3.4 Создание проекта веб-приложения на Flask

Проект можно создать, используя pycharm. Если вы используете данное IDE, можете пропустить дальнейшие указания создания проекта на github.

Создадим на сайте github новый проект. Можем заполнить файл readme.md, либо если будете использовать git client на своей машине, то лучше не делать этого сразу. Дальнейшие указания будут связаны с непосредственной работой через github сайт.

Создадим каталог, в котором будет храниться наш проект, flaskapp. Для этого выбираем Create New File (рисунок 3.1).

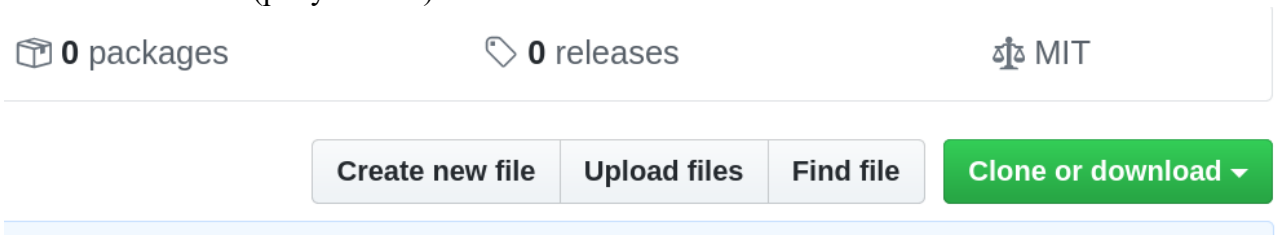

Рисунок 3.1 – Создание файла в проекте

Поставим слэш после имени каталога flaskapp и имя файла .gitkeep (рисунок 3.2). В указанном файле можем написать, что угодно, например, для чего данный каталог. Нажимаем кнопку commit снизу. Также вы можете использовать локальный файл .gitignore для указания папок и файлов, которые не должны отслеживаться системой Git и игнорироваться при commit, например, файлы с паролями и т.д.

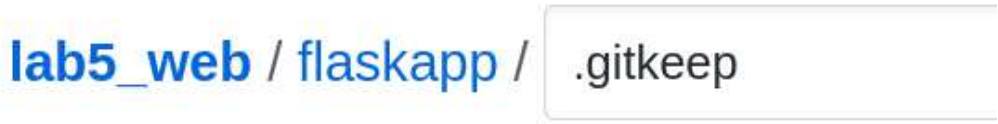

Рисунок 3.2 - Создание папки проекта и неиспользуемого файла

Теперь создадим файлы, в которых будут начальные файлы с нашим веб-сервером. Перед разработкой веб-сервера желательно разработать модель будущего сайта, провести анализ требований пользователей, оценить возможную нагрузку на ваш сайт. Но, в нашем случае мы пишем достаточно простое приложение и потому пока изучим возможности CI и создание простого веб-сервиса. В нашем каталоге на сайте создадим файл some_app.py, в котором содержится следующий код. Можно непосредственно создать пустой файл и туда скопировать код, можно скопировать созданный файл с локального диска, затем реализовать commit.

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

После того как вы привязали свой проект на github с непрерывной интеграцией travis-ci, можно создать .travis.yml файл в проекте github и после commit будет запущен build (рисунок 3.3).

```
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.

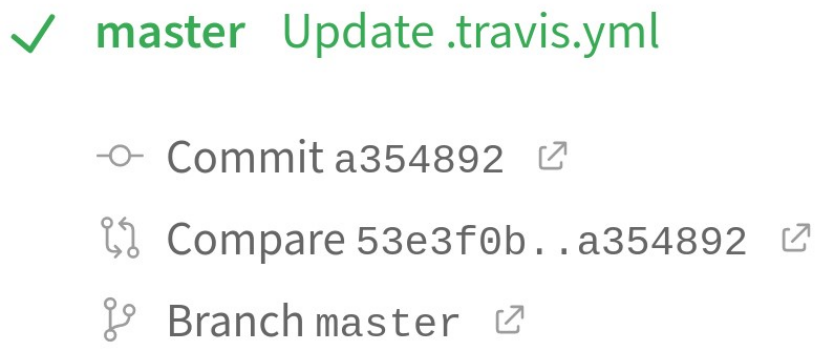

Рисунок 3.3 – Пример подтверждения Commit

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

3.5 Продолжение простейшего эксперимента с проектом Flask

Продолжим дальше наши эксперименты уже непосредственно с простейшим сайтом с использованием фреймворка Flask. Очевидно, 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, существует много и других систем подобного рода, так называемых PaaS (Google APP Engine, Amazon web services, Microsoft Azure и т.д.). Но 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, при это 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.

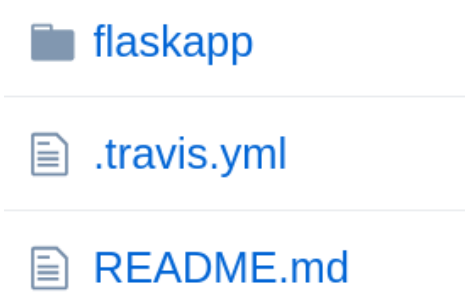

Рисунок 3.4 – Папка проекта

Итак, какие файлы нам понадобятся в каталоге flaskapp. Файл содержащий приложение Flask. Назовем его 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)
```

Здесь представлено простейшее приложение, которое имеет конечную точку входа, в виде url и выдает на запрос пользователя страницу html. Если вы прочитали учебное пособие или знакомы с протоколом HTTP, то понимаете, что со стороны пользователя (браузера) идет запрос GET, на этот запрос веб сервер приложения возвращает ответ, HTTP ответ имеет свой формат в виде кода, заголовков и тела ответа. Здесь телом ответа будет HTML. Но в подобных фреймворках обычно добавляется возможность возвращать обрабатываемый веб-сервером шаблон, чтобы обеспечить динамическое изменение страницы в зависимости от команд пользователя. Такие фрейворки предлагают хранить шаблоны в отдельных директориях, как и так называемый статический контент. Вы можете сами задать структуру папок своего проекта при разработке приложения, обычно предлагаются стандартные решения, позволяющие обеспечить будущую масштабируемость проекта, его тестирование и так далее. Иногда такие фреймворки дают возможность на лету добавлять конечные точки доступа к новой функциональности приложения, за счет разделения проекта на части, где новая точка может быть добавлена динамически. Так же можно группировать конечные точки в маршруты, добавлять маршруты в основное приложение с указанием префикса url, что улучшает читаемость кода и его сопровождение.

Можно привести следующий типичный набор папок для приложения с использованием фреймворка Flask:

/static: В этой папке обычно хранятся статические файлы, такие как CSS, JavaScript и изображения.

/templates: Здесь располагаются шаблоны HTML-страниц, которые используются для отображения контента.

/routes: В этой папке обычно содержатся файлы, отвечающие за маршрутизацию и обработку запросов.

/config: В папке config могут находиться файлы с настройками приложения, например, файлы с переменными окружения или конфигурационные файлы.

/data: Папка, где могут храниться данные, такие как файлы базы данных или другие данные, используемые в приложении.

/logs: Здесь могут храниться логи приложения.

/tests: Папка, где располагаются тесты для приложения.

3.5.1 Что такое WSGI

Файл wsgi.py. WSGI (Web Server Gateway Interface) - это стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером, например Apache. WSGI (Web-Server Gateway Interface) является потомком CGI (Common Gateway Interface). Когда веб начал развиваться, CGI разрастался из-за поддержки огромного количества языков и из-за 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-приложениеми или предпроцессинг(предобработку) отдаваемого контента.

Файл wsgi.py.

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

3.5.2 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.

3.5.3 Запуск проекта с использование 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 чего 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 общем-то, в данном случае произойдет успешный build commit-a.

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.

3.5.4 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 обосновано 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 commit будет подтвержден, несмотря на, казалось бы, отсутствие каких-то библиотек и ошибок сборки проекта. Но, 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.

3.6 Краткое знакомство с шаблонами Flask

Для начала изучим возможности Flask для нашего будущего проекта. Сначала 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 так же 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, так как в python функция является так же 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 в нее поместить файл simple.html. Затем обновить файл some_app.py. Затем файл 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 commit. 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.

3.7 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.

3.7.1 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 наше api : some_app.py, в функции обработки запроса будет рендериться форма, которая так же добавлена как класс в файл some_app.py. В шаблоне template/net.html, который у нас рендерится мы добавим вызов обработки формы, которая передается 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 и для сабмит 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)
```

Так же 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 тэгов 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 %}
```

3.8 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)
```

3.9 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'] = '_____'
```

3.10 Добавление возможности классификации изображения

Расширим функционал 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, дабы у вас не возникала ошибка на стороне сервера 500.

3.10.1 Итоговый client.py запрашивающий реализованный сервис

Создадим файл клиента запрашивающего данные с сервера:

```
# импортируем нужные модули
```

```
import os
from io import BytesIO
import base64
img_data = None
# создаем путь к файлу (для кроссплатформенности, 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 commit 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. Либо вместо web_site распознанный класс, переданный вами в файле скрипта client.py.

```
[('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.
```

Вообще говоря, тут мы используем travis-ci как удаленное средство запуска проекта, предполагается обычно, 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 бы и не происходило.

3.11 Возвращающение разных документов в зависимости от шаблона.

Добавим дополнительный функционал 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)
```

А 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 commit не будет фиксироваться на github.

Загрузим новый проект на github.

Давайте проверим. Изменим YAML файл, добавив туда

```
 - pip3 install lxml
```

Но 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 commit не подтвердится.

3.12 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 (рисунок 3.5)

Рисунок 3.5 – Пример генерации ключа на heroku

При этом чтобы реализовать deploy на heroku желательно получить encrypt ключа с помощью консоли travis-ci, дабы никто не увидел вашего секретного ключа heroku. Если вам нестрашны 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 или 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.

И 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 deploy.

Выберете deploy метод Github (рисунок 3.6).

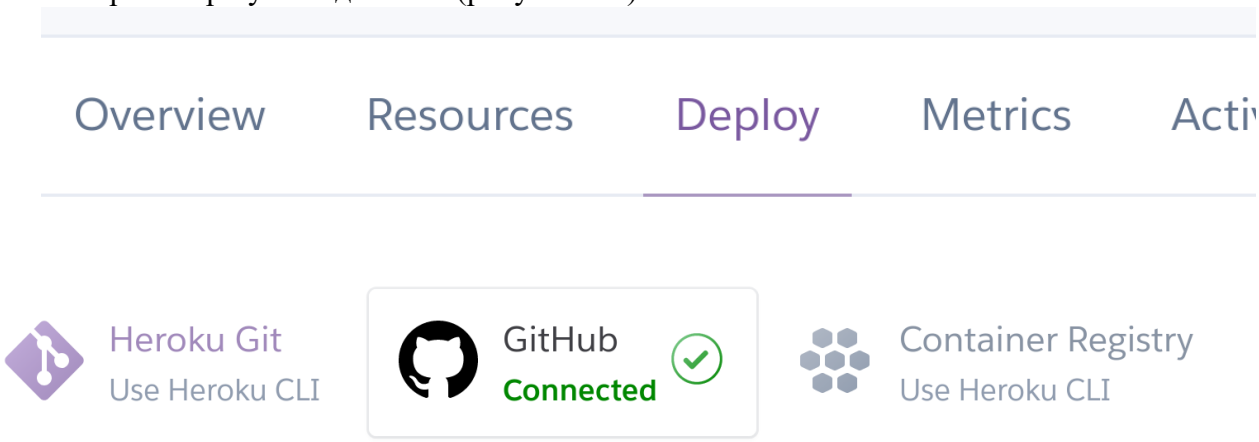

Рисунок 3.6 – Пример выпора деплоя

После чего 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 commit 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.

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

И воспользоваться командой.

$ heroku run bash --app YOUR_HEROKU_APP

Для просмотра логов можно воспользоваться коммандой.

$ heroku logs --tail --app YOUR_HEROKU_APP

3.13 Разработка придожения используя Fastapi

Данный параграф содержит основные этапы работы на примере GitHub Actions, render, Pipenv и FastAPI и примеры кода для работы с Captcha. Описывает процесс тестирования и деплоя. Вначале идет пример работы с Pipenv, но можно использовать venv.

Все указания приводятся для версии Python 3.8.10. Устанавливаем Python версии 3.8 и PyCharm. Если пользуетесь другими версиями Python, то постарайтесь самостоятельно исправлять проблемы, связанные с несоответствием версий библиотек и установкой нужного программного обеспечения. Развитие данного умения также является основополагающим моментом в обучении и приобретении необходимых навыков. Не возбраняется пользоваться удаленными ресурсами и поисковиками для устранения проблем и возможных ошибок.

3.13.1 Пример использования в качестве системы контроля версий GitFlic

Здесь кратко будет рассмотрен, для тех, кому интересно, Российский аналог систем типа GitHub. К сожалению, таких возможностей как GitHub он не предоставляет, тем не менее, по заявлениям разработчиков в нем присутствует и система CI (правда в платном варианте или если бесплатно, то ее можно развернуть на локальной машине пользователя). Далее мы будем рассматривать систему GitHub, но приведем здесь начальный пример работы с GitFlic. Можно начать выполнение лабораторной работы с этого момента, так как здесь будут приведены примеры взаимодействия PyCharm и системы контроля версий Git.

Регистрируемся на gitflic.ru. Вводим пароль и свою почту, туда придет письмо для подтверждения аккаунта.

Создаем проект, выбрав язык программирования, и добавляем название проекта выше в поле ввода (рис. 3.7).

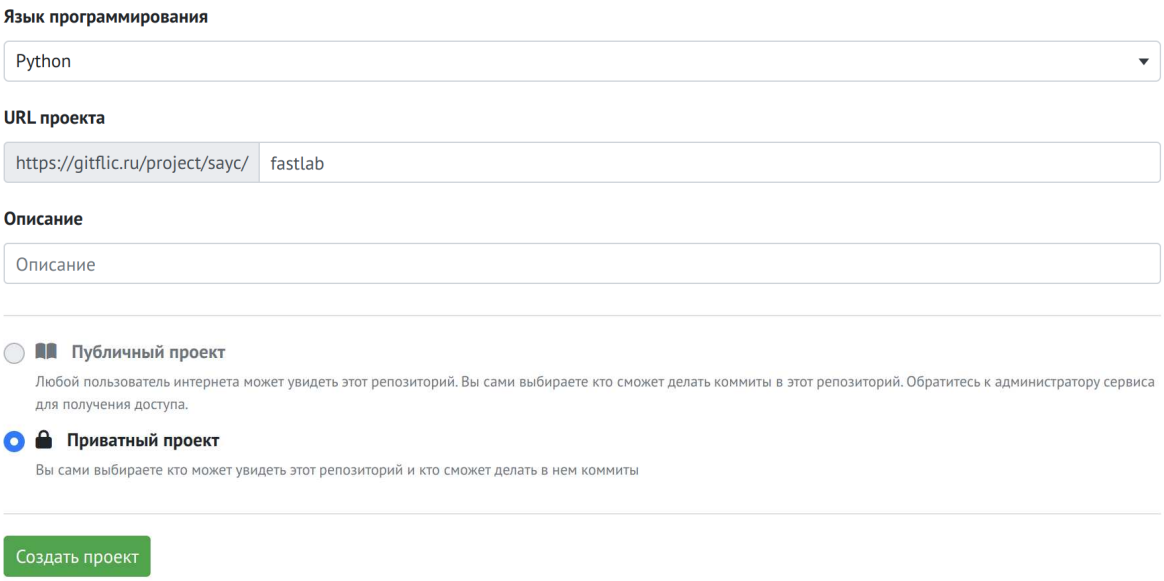

Рисунок 3.7 – Пример создания проекта на GitFlic

Установите Git.
sudo apt install git
Проверьте, что он установился запустив команду.
git --version

3.13.2 Управление окружением Pipenv и краткие сведения о ней.

В данном параграфе приводятся примеры работы с системой окружения Pipenv (в основных методических указаниях приводилась virtualenv). На ваше усмотрение вы можете использовать либо ту, либо другую.

Краткая информация по работе с Pipenv.

В совместной разработке на Python лучше использовать Pipenv, который позволяет управлять окружениями enviroments, устанавливать и обновлять пакеты (pip), используя файл Pipfile.lock, устанавливать детерминированный набор пакетов. Например, Pipenv удобно применять если разные пакеты используют зависимости от пакетов разных версий.

Используются два файла: Pipfile, являющийся фактически заменой requirements.txt, и файл Pipfile.lock, в котором находятся все зависимости, отслеживающиеся автоматически (дерево зависимостей).

Для создания или инициализации среды можно использовать команду:

pipenv shell,

если среды не существует она будет создана.

В текущей папке будет создан файл Pipfile, а среда будет создана в папке по умолчанию (например, /home/user/.local/share/virtualenvs/….), если запуск произошел из командной строки, а не из environment, если же запуск произошел из environment, то будет инициализирована среда соответствующая текущей папке.

Перед запуском нужно перейти в папку вашего проекта. Если же используется PyCharm, то при выборе папки проекта установка среды и создание файла Pipfile реализуется автоматически в ней, среда размещается в папке по умолчанию для данной операционной системы. Удалить среду можно с помощью команды pipenv —rm, запустив ее в папке, где находится файл Pipefile или откуда вызывалась команда pipenv shell.

Генерация Pipefile.lock будет выполнена командой:
pipenv lock.
Установка всех пакетов в окружении из Pipefile.lock:

pipenv sync

Удаление всех пакетов установленных для данного environment, которых нет в Pipefile.lock:
pipenv clean

Для установки пакетов используется команда:
pipenv install имя_пакета

Вызов команды:
pipenv install
произведет установку всех пакетов указанных в Pipfile и приведет к обновлению Pipfile.lock.

3.13.3 Создание проекта в PyCharm с использованием Pipenv

Создадим проект в PyCharm.
В PyCharm Pipenv можно выбрать следующим образом (рис. 3.8):

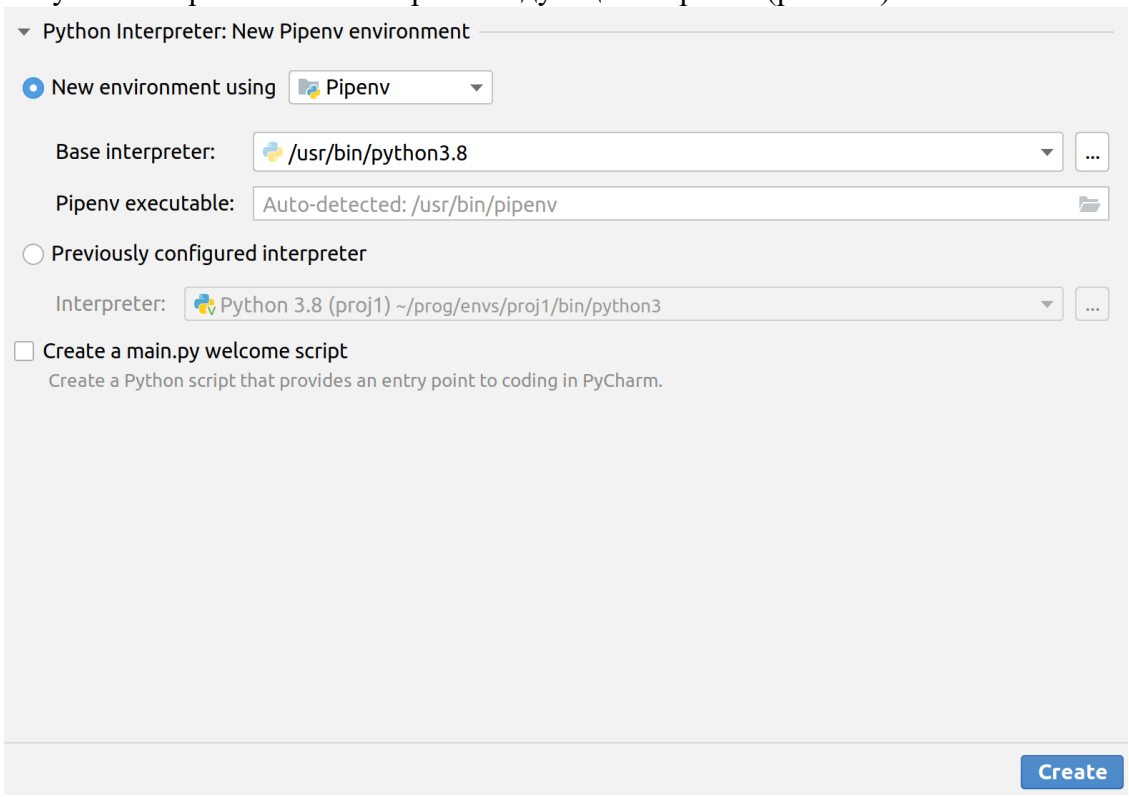

Рисунок 3.8 — Выбор Pipenv

3.13.4 Подключение git в PyCharm и push на Gitflic

Далее подключим Git. Выберем вкладу VCS - > Enable version control integration.
Создадим файл .gitignore, при добавлении в Git нажимаем Cancel, чтобы не добавлять его в отслеживание системой контроля версий, в этот файл указываем строчки (иногда наоборот лучше добавить pipfile.lock в отслеживаемые, чтобы потом не запускать команды создания данного файла):
.idea/.
Pipfile.lock

Правой кнопкой мыши щелкаем на Pipfile, выбираем GIT и add, тем самым добавляя данный файл в отслеживаемые, в результате он должен стать зеленого цвета.

Далее можно сделать commit во вкладке GIT.

Commit сохранит отслеживаемые файлы в системе контроля версий, push позволяет их поместить на удаленный сервер, pull – взять на ваш локальный компьютер с сервера.

В папке нашего проекта запускаем через терминал команды, указанные в проекте GitFlic.

git config --global user.name "*****"

git config --global user.email "***@some_your_mail.org"

Пример окна с коммитом в PyCharm приведен на рисунке 3.9.

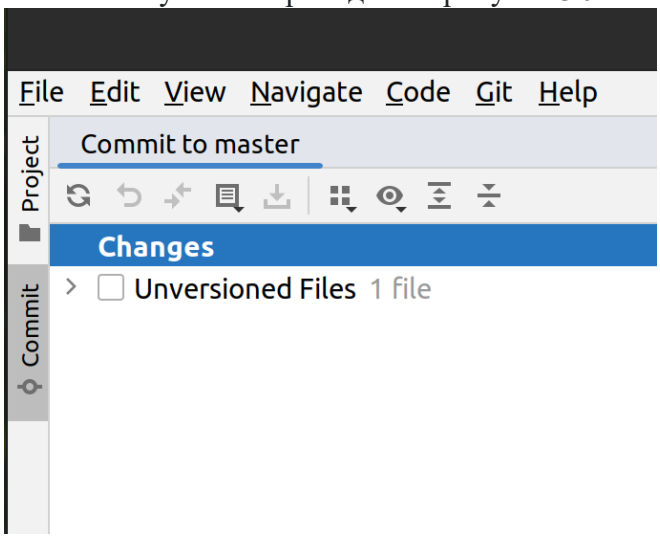

Рисунок 3.9 – Пример commit в PyCharm

Можно переключаться между Commit и Project.

Разработаем наше веб-приложение, веб-сервис используя фреймворк FastAPI. Данный фрейморк позволяет разрабатывать приложения довольно быстро, в том числе асинхронные, и базируется на двух фреймворках: Pydantic и Starlette, первый отвечает за валидацию данных, второй –за работу с web. Также FastAPI поддерживает REST API, документацию Swagger и авторизацию OAuth *2.0*.

Для установки нужных библиотек можно изменить GitFlic:

```
[packages]
fastapi = "*"
pydantic = "*"
uvicorn = "*"
```

Также для поиска информации об управлении пакетами можно обратиться к инструкции по использованию Pipfile:

https://www.jetbrains.com/help/pycharm/using-pipfile.html.

Можно в терминале вызывать команду pipenv update или сделать запрос на обновление и установку зависимостей в PyCharm (рисунок 3.10)

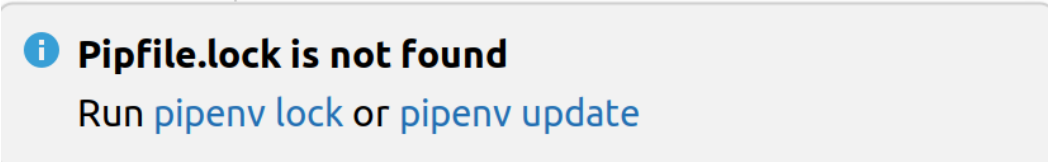

Рисунок 3.10 – Пример запроса на обновление и установку зависимостей в PyCharm

Можно записать изменения в pipenv.lock файл или записать и установить все пакеты и зависимости с помощью команды pipenv update.

Создадим в нашем проекте файл fastlab.py с кодом. Пример на FastAPI.

```
from fastapi import FastAPI
import uvicorn
app = FastAPI()
# Hello World route
@app.get("/")
def read_root():
    return {"Hello": "World"}
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Документацию можно посмотреть здесь:

http://0.0.0.0:8000/docs

Запустим в PyCharm данный файл и посмотрим как работает наше приложение, перейдя по предложенной ссылке.

Конечно, запускать uvicorn можно не из самого файла Python, а наоборот использовать uvicorn для запуска приложения.

Создадим файл README.md. Запишем туда нужную нам информацию касательно проекта. Можно откатить добавленные для отслеживания файлы с помощью GIT → Rollback.

В .gitignore можно добавить Pipefile.lock как уже было сказано, но можно и не добавлять, дальнейшие указания предполагают, что мы не отслеживаем данный файл и тогда его надо создавать при запуске тестов и при деплое.

Итоговое содержимое нашего проекта (рис. 3.11):

```
fastlab/
 .gitignore
 fastlab.py
 Pipfile
 Pipfile.lock
 README.md
```

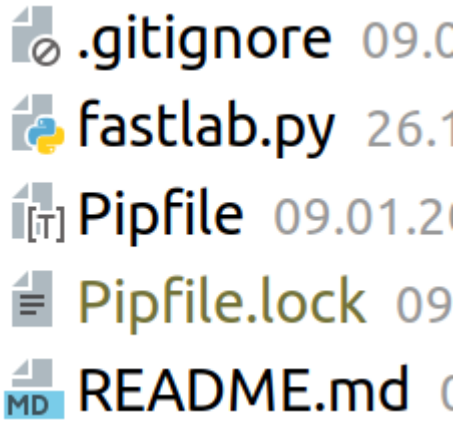

Рисунок 3.11 – Итоговое содержимое проекта

Теперь сделаем push на наш репозиторий в GitFlic. Необходимо ввести пароль и адрес указываемой командой: git remote add origin https://gitflic.ru/project/ваш логин /имя проекта.git. При создании проекта этот адрес указывается снизу для помощи в пункте «Быстрая настройка», если вы уже делали подобное раньше. «Запушить» существующий репозиторий.

Для работы с CI/CD можно использовать standalone версию. Можете рассмотреть этот вариант самостоятельно.

3.13.5 Push проекта на GitHub и тестирование на GitActions

GitHub является известной системой контроля версий, кроме того, достаточно большой набор функций там предоставляется бесплатно.

Рассмотрим вариант использования GitHub и GitActions. Зарегистрируемся на github.com и создадим новый репозиторий (рис. 3.12).

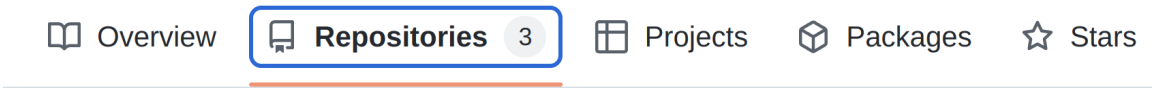

Рисунок 3.12 – Выбор или создание репозитория на GitHub

Выберем *репозиторий* и далее *создание нового репозитория* (зеленая кнопка *New*).

Затем укажем имя репозитория, которое потом будем использовать (рис. 3.13).

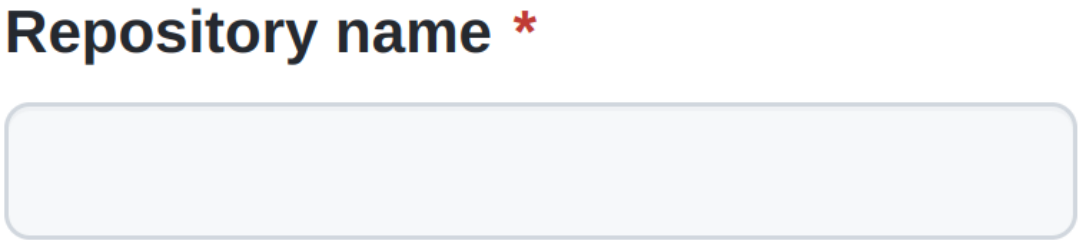

Рисунок 3.13 – Ввод имени репозитория

Бесплатно можно использовать как публичный, так и закрытый репозиторий.

После создания репозитория уже готовый проект из PyCharm «запушим» на GitHub, указав адрес, который можно получить во вкладке *Code* на GitHub. Также далее можно указать специальный токен доступа, либо напрямую логин/пароль. Чтобы получить токен можно проделать следующие шаги:

- Нажмите на свой аватар в правом верхнем углу и перейдите в меню Settings.

- Из боковой левой панели перейдите в *Developer settings* > *Personal access tokens* (в самом низу). Нажмите *Generate new token*, при необходимости введите пароль от учетной записи.
- В поле *Note* введите назначение токена.
- Выбрать token(classic) и галочки Repo, workflow, read.org, gist.

Далее пробуем во вкладке PyCharm или в командной строке запустить команду push в ветку main, указав во всплывающем окне адрес HTTPS, и далее сгенерированный токен.

В случае если push не сработает (из-за несовпадения истории коммитов, например, если в удаленный репозиторий была добавлена лицензия), может потребоваться слияние проектов. Поэтому сначала делаем pull, при этом переименовывая ветку в main или то название, которое есть в GitHub. Указываем параметр rebase во вкладке *Modify options* (рис. 3.14). Rebase – это наложение коммитов поверх другого базового коммита. Под базовым понимается тот коммит, к которому применяются коммиты выбранной ветки.

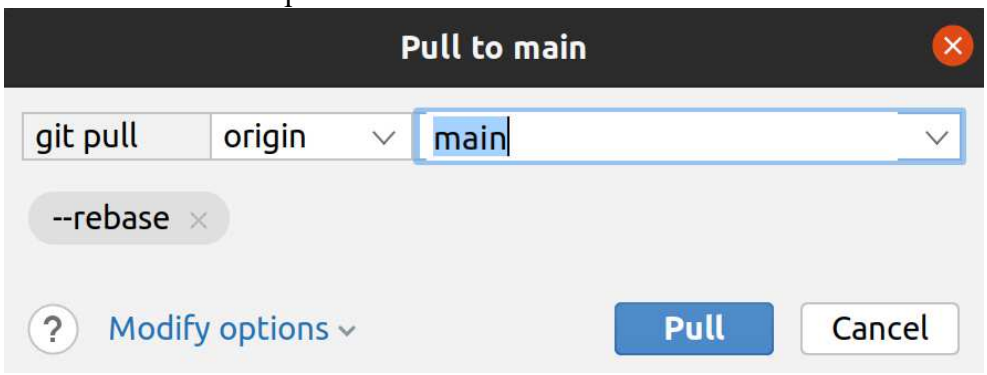

Рисунок 3.14 - Пример выполнения pull с git

Можно посмотреть разницу двух веток: удаленной и локальной (рис. 3.15). Далее снова делаем push.

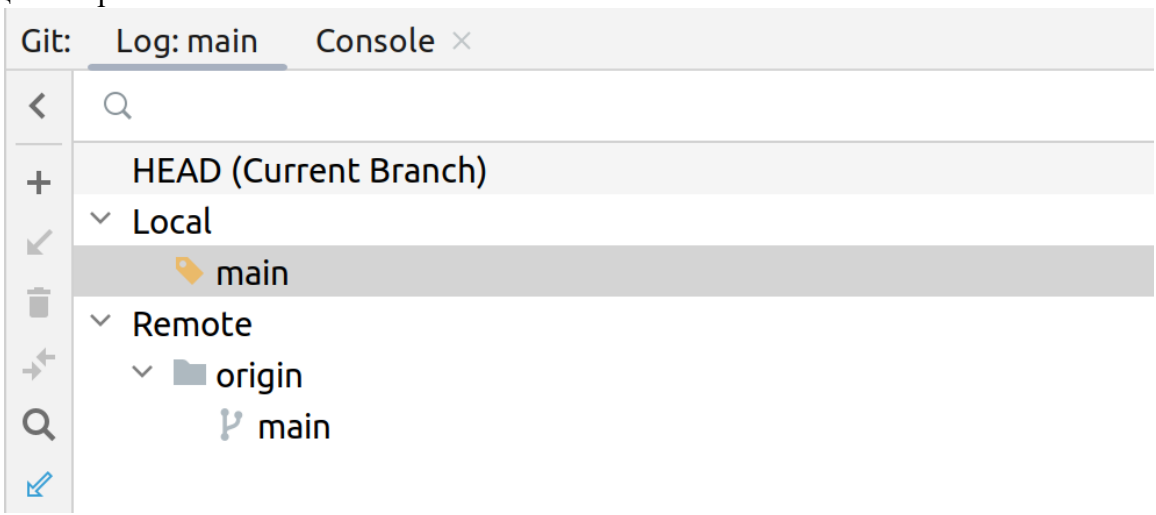

Рисунок 3.15 - Просмотр разницы двух веток в PyCharm

3.13.6 Тестирование с помощью workflows

В нашем проекте PyCharm создадим следующие папки.

.github

workflows

my.yaml

В my.yaml файл, который будет использоваться для запуска actions на GitHub на событие push, добавим следующие декларации:

```
# имя, отображаемое в интерфейсе
name: 'test my project'
on: [push] # список событий, на которые запускается действие
jobs: # список работ, которые будут производиться (каждая работа будет выводиться отдельно)
  checks: # имя работы checks
    runs-on: ubuntu-latest # на какой машине делать работы (можно задать матрицы машин, допустимы windows и mac os)
    steps: # выполняемые последовательно шаги
      - uses: actions/checkout@v3
      - run: echo "hello world"
```

Ключевое слово uses сообщает о том, что на этом шаге будет запущена версия 3 «действий»/«действия проверки». Это действие, которое проверяет ваш репозиторий на исполнителе, позволяя вам запускать сценарии или другие действия с вашим кодом (например, инструменты сборки и тестирования). Вы должны использовать «действие проверки» каждый раз, когда ваш рабочий процесс будет выполняться с кодом репозитория.

Run запускает команду, в данном случае команду echo.

Теперь попробуем протестировать наше приложение с помощью GitActions и если тесты не пройдут успешно, то commit на GitHub не сработает, дальнейшей целью будет сделать deploy на удаленную PaaS систему. Вы можете выбрать любую, которая вам доступна, естественно, указания для них будут отличаться от предложенного здесь варианта render.

Изменим наш yaml файл, добавив следующее содержимое в соответствующий блок и сделаем еще один commit и push, данные коммиты позволят установить Python и Pipenv.

```
checks:
  runs-on: ubuntu-latest
  steps:
- name: Begin
    uses: actions/checkout@v3
- name: Echo hello
    run: echo "hello world"
- name: Setup Python
    uses: actions/setup-python@v2
    with:
      python-version: "3.8"
- name: Install pipenv
    run: python -m pip install pipenv
```

Итоговый файл:

```
name: 'test my project'
on: [push]
jobs:
  checks:
    runs-on: ubuntu-latest
    steps:
      - name: Begin
```

```
    uses: actions/checkout@v3
  - name: Echo hello
    run: echo "hello world"

  - name: Setup Python
    uses: actions/setup-python@v2
    with:
      python-version: "3.8"
  - name: Install pipenv
    run: python -m pip install pipenv
```

Все изменения на GitHub можно смотреть во вкладке actions. Для этого необходимо щелкнуть мышью на соответствующем коммите и посмотреть workflow check.

Теперь нужно установить требуемое окружение и запустить наше приложение или тесты.

Содержимое Pipefile

```
[packages]
fastapi = "*"
pydantic = "*"
uvicorn = "*"
pytest = "*"
[dev-packages]
[requires]
python_version = "3.8"
```

Добавим в yaml файл установку требуемых зависимостей и запуск в среде окружения нового файла с тестом.

```
- name: Pipenvlock
  run: pipenv lock
- name: Pipenvsync
  run: |
    pipenv sync
- name: Start tests
  run: pipenv run python -m pytest my_tests.py
```

Последняя строчка запускает тестирование в заданном окружении Pipenv с учетом установленных библиотек.

Добавим файл с тестом, в котором проверяется что 2+2=4.

Файл my_tests.py.

```
import fastlab
def test1():
    assert  fastlab.sum_two_args(2,2) == 4
```

Изменим файл fastlab.py, добавив функцию, которая суммирует два аргумента.

```
import uvicorn
app = FastAPI()
```

```
def sum_two_args(x,y):
    return x+y

# Hello World route
@app.get("/")
def read_root():
    return {"Hello": "World"}
if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

В данных примерах мы добавляем одну функцию в наш файл, и затем тестируем ее в модуле тестирования, это так называемое unit тестирование.

Проведем изменения, добавим my_tests.py в отслеживаемые git, сделаем commit и push. Не забывайте указывать в commit, какие изменения вы делаете. Смотрим результаты на GitHub. Не переживайте, если сразу что-то не получилось, можно посмотреть ошибки и попытаться их исправить, не бойтесь пользоваться поисковиком.

Добавим заведомо два ошибочных теста в файл my_tests.py и один верный и посмотрим вывод, который будет подготовлен в CI/CD после push.

```
def test2():
    assert  fastlab.sum_two_args(2.0001,2) == 4
def test3():
    assert  fastlab.sum_two_args(3,2) == 5
def test4():
    assert  fastlab.sum_two_args("2",2) == 4
```

Очевидно, проверка не дала положительного результата, можно посмотреть ошибки и их исправить.

```
def test2():
    assert  fastlab.sum_two_args(2.0001,2) == 4.0001
def test3():
    assert  fastlab.sum_two_args(3,2) == 5
def test4():
    assert  fastlab.sum_two_args("2","2") == "22"
```

Снова запустим. Тесты должны пройти.

3.13.7 Запуск uvicorn и работа с веб-приложением

Дальше рассмотрим работу с нашим веб приложением.

Для запуска uvicorn используем его возможности.
Запустить веб-приложение Python:
uvicorn {{import.path:app_object}}
Запустить сервер на порту 8080 на локальном хосте (localhost):
uvicorn --host {{localhost}} --port {{8080}} {{import.path:app_object}}
Включить "горячую перезагрузку" для отслеживания изменений:
uvicorn --reload {{import.path:app_object}}

Использовать 4 рабочих процесса для обработки запросов:
uvicorn --workers {{4}} {{import.path:app_object}}
Запустить приложение по HTTPS:
uvicorn --ssl-certfile {{cert.pem}} --ssl-keyfile {{key.pem}} {{import.path:app_object}}

В нашем случае используем команду
uvicorn --host 127.0.0.1 --port 8000 fastlab:app
Запустим ее в терминале PyCharm, предварительно удалив из файла fastlab запуск uvicorn.

```
app = FastAPI()
def sum_two_args(x,y):
    return x+y
# Hello World route
@app.get("/")
def read_root():
    return {"Hello": "World"}
```

Обычно тесты подобных веб-приложений можно реализовать с помощью такого средства как selenium, но, к сожалению, выбрав его, могут возникнуть трудности при установке и при дальнейшей работе с ним, поэтому в рамках лабораторной работы лишь приведем начальный код, который поможет вам запустить работу с selenium на Python. Вы можете самостоятельно добавить в свой код любые тесты на selenium, изучив материалы в интернете.

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as ChromeService
from webdriver_manager.chrome import ChromeDriverManager
options = webdriver.ChromeOptions()
# отключает режим показа браузера
#options.add_argument('—headless')
# пример других опций, которые вы можете использовать
#options.add_argument('—disable-gpu')
#options.add_argument("—disable-popup-blocking")
# создать драйвер браузера Chromium
driver = webdriver.Chrome(options=options, service=ChromeService(ChromeDriverManager().install()))
# получить страницу
driver.get("http://127.0.0.1:8000")
print(driver.title)
driver.close()
```

3.13.8 Deploy на render.com

Далее попробуем сделать deploy нашего проекта на render.com. Для этого необходимо там зарегистрироваться любым доступным образом, так как вы уже зарегистрированы на GitHub, то можете зайти туда под аккаутном GitHub. Далее выбираем тарифный план (среди них есть и бесплатный).

Далее на вкладке dashboard выбираем, например веб-сервисы (рис. 3.16).

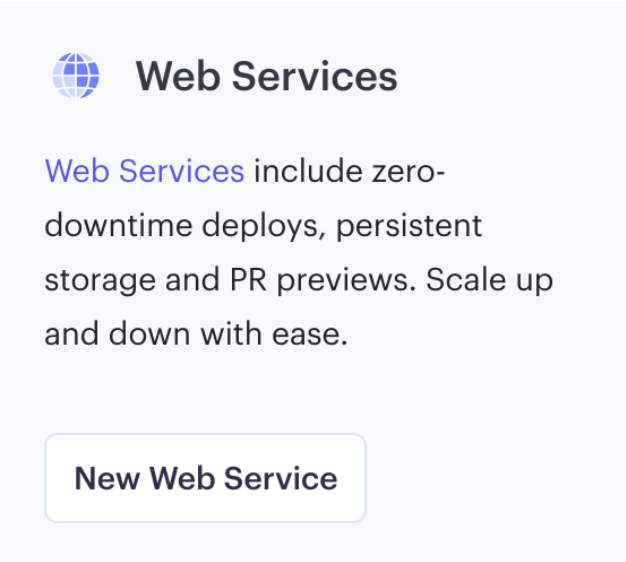

Рисунок 3.16 – Создание веб-сервиса на render.com

Настраиваем connect с GitHub справа или выбираем ваш репозиторий, если он доступен (рис. 3.17).

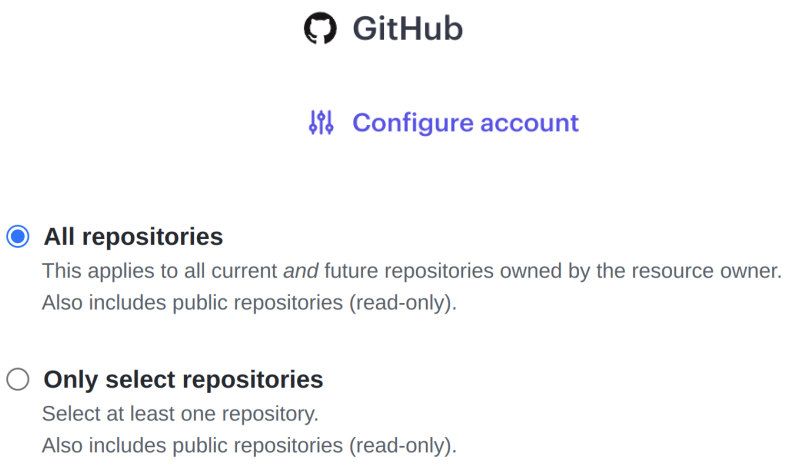

Рисунок 3.17 – Создание коннекта с репозиторием на GitHub

Можно выбрать все репозитории или какой-то конкретный, можно указать ваш для лабораторной работы.

Затем вводите пароль для входа в GitHub.

Перед тем как нажать кнопку коннект на render.com необходимо **добавить Pipfile.lock в отслеживаемые git.**

Изменим наш yaml файл на следующее содержимое:

```
name: 'test my project'
on: [push]
jobs:
  checks:
    runs-on: ubuntu-latest
    steps:
```

```
- name: Begin
      uses: actions/checkout@v3
- name: Echo hello
      run: echo "hello world"
- name: Setup Python
      uses: actions/setup-python@v2
      with:
        python-version: "3.8"
- name: Install pipenv
      run: python -m pip install pipenv
- name: Pipenvsync
      run: |
        pipenv sync
- name: Start tests
      run: pipenv run python -m pytest my_tests.py
```

Файл fastlab.py

```
from fastapi import FastAPI
app = FastAPI()
def sum_two_args(x,y):
    return x+y
# Hello World route
@app.get("/")
def read_root():
    return {"Hello": "World"}
```

После этого нажимаем кнопку коннект на выбранном репозитории на render.com. Указываем серверы, где хотим разместить наш сервис, его имя и команды для запуска (рис. 3.18, 3.19).

python -m pip install pipenv && pipenv sync

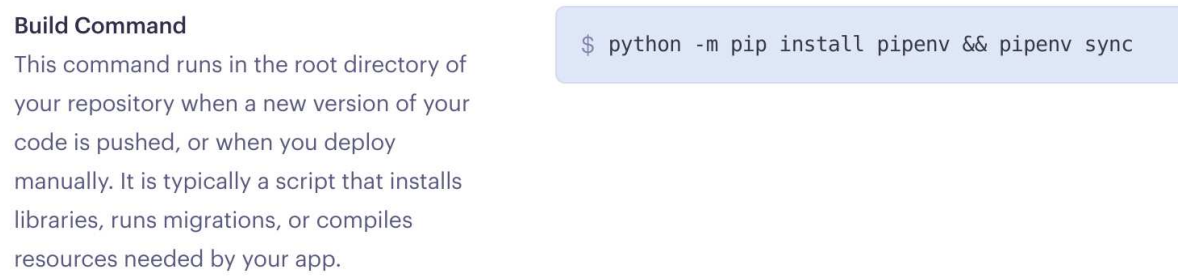

Рисунок 3.18 – Настройка build приложения перед его запуском

pipenv run python -m uvicorn fastlab:app --host 0.0.0.0

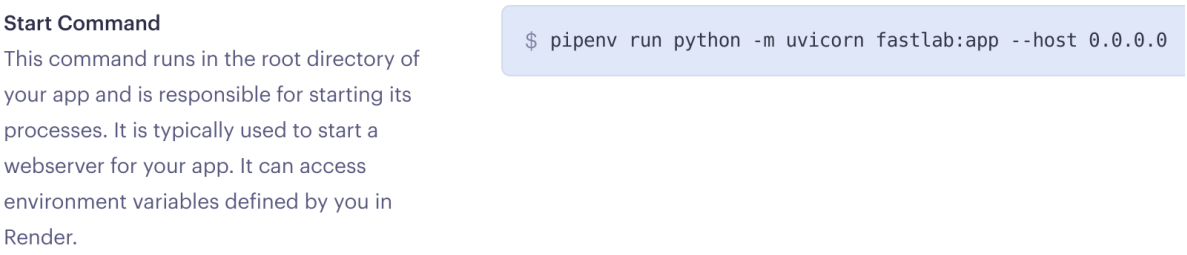

Рисунок 3.19 – Настройка запуска самого приложения

После достаточно длительного процесса наш мини веб-сервис готов (рис. 3.20).

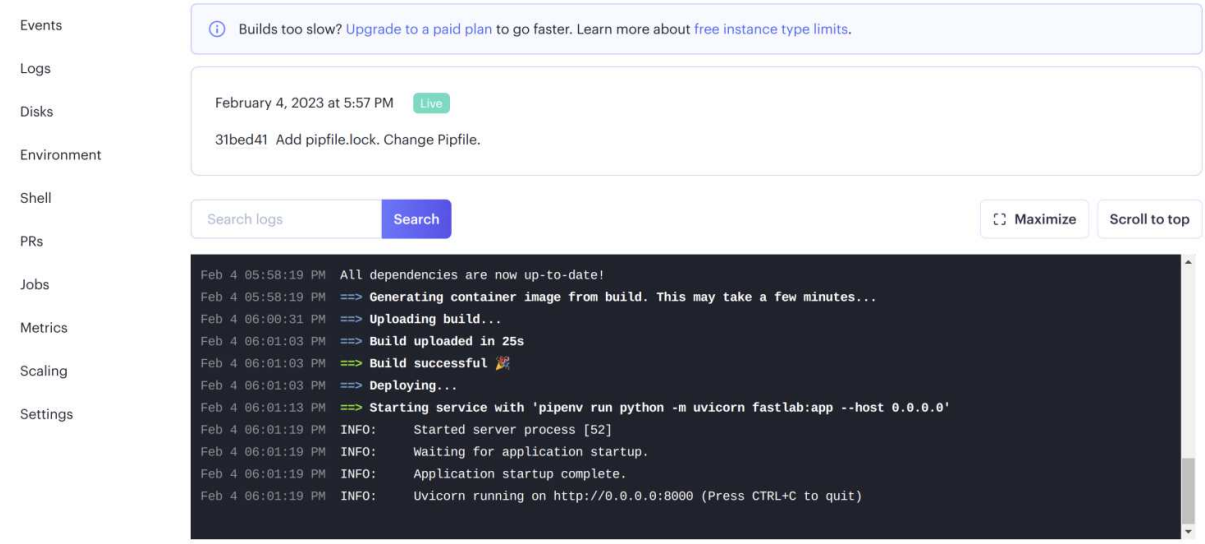

Рисунок 3.20 – Просмотр результата build и запуска приложения в окне на render.com

Теперь после каждого push на GitHub будет проходить процесс деплоя на render.com.

Указанное имя сервиса доступно по приведенной наверху ссылке https://some_name.onrender.com. some_name это имя, которое вы указали.

Также вы можете попытаться использовать другие PaaS системы, которые вам доступны, обычно они платные.

Процесс деплоя виден во вкладке events, там же можно нажать на Deploy и посмотреть весь процесс, кроме того, доступны логи выполнения вашей программы и ее вывод (рис. 3.21).

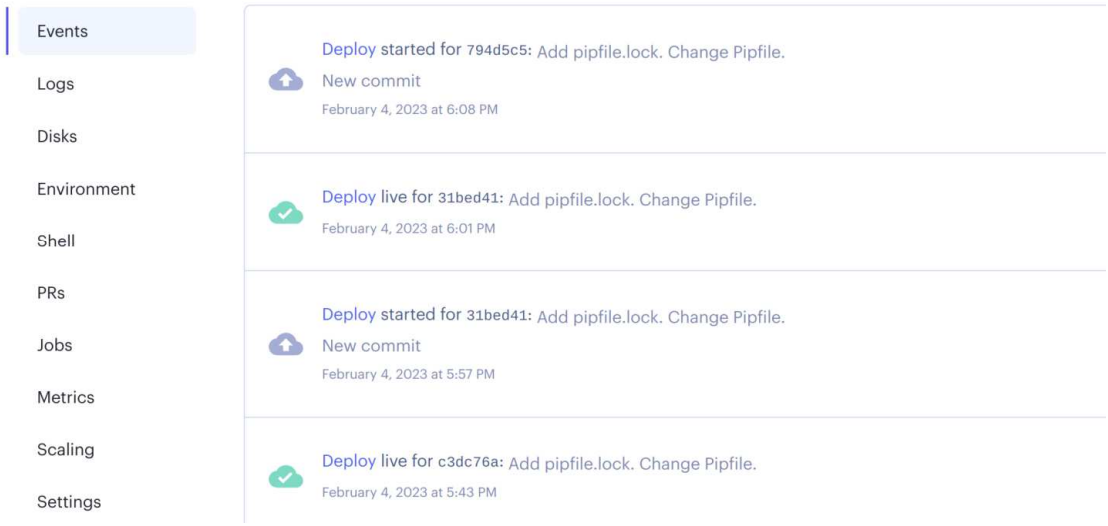

Рисунок 3.21 – Отображения событий на render.com при операциях push и деплой

Вообще, в дальнейшем можно не делать push, если об этом не сказано отдельно, если это занимает время или не хочется ждать результатов. Можно просто смотреть все добавления и изменения непосредственно на локальной машине.

Можете использовать команду для запуска вашего приложения из терминала PyCharm.

uvicorn --host 127.0.0.1 --port 8000 fastlab:app

3.13.9 Продолжение реализации проекта, создание шаблона FastAPI

Здесь мы реализуем шаблоны для вывода созданной картинки, как сохраненной в static, так и выдаваемой через url и запрос get.

Создадим два каталога static и templates, в каталог static будем класть css файлы, статические изображения и т. д., в templates – шаблоны html файлов, которые будут рендериться (обрабатываться) с помощью jinja и далее в виде html отдаваться на сторону клиента (типичная работа динамического веб-сервера, запустить скрипт, который сформирует страницу).

В каталоге templates создадим файл some.html. Автоматически PyCharm добавит туда тэги.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
   <link href="{{ url_for('static', path='/styles.css') }}" rel="stylesheet">
</head>
<body>
   <h1>Something from user : {{ something }}</h1>
</body>
</html>
```

В каталог static добавим файл styles.css, который задаст цвет заголовка h1 в html файле.

```
h1 {
```

```
    color: blue;
}
```

В Pipefile добавим строчку jinja2 = "*".

Изменим код файла fastlab.py на следующий.

```
from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
app = FastAPI()
def sum_two_args(x,y):
    return x+y
# Hello World route
@app.get("/")
def read_root():
    return {"Hello": "World"}
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")
@app.get("/some_url/{something}", response_class=HTMLResponse)
async def read_something(request: Request, something: str):
    return templates.TemplateResponse("some.html", {"request": request, "something": something})
```

Запустим из терминала:

uvicorn --host 127.0.0.1 --port 8000 fastlab:app

В строке браузера введем адрес

127.0.0.1:8000/some_url/12345_string

В результате должна появиться надпись, которая была в some.html, и значение, которое введено после some_url.

Важное замечание, если вы собираетесь далее делать деплой на сервер render, то задайте нужную версию Python в Environments. Например, 3.8.10.

Переменная PYTHON_VERSION (рис. 3.22). В поле ввода текста (Value) указываем версию, например, 3.8.10, если не задать эту переменную, то по умолчанию на render будет версия Python 3.7.

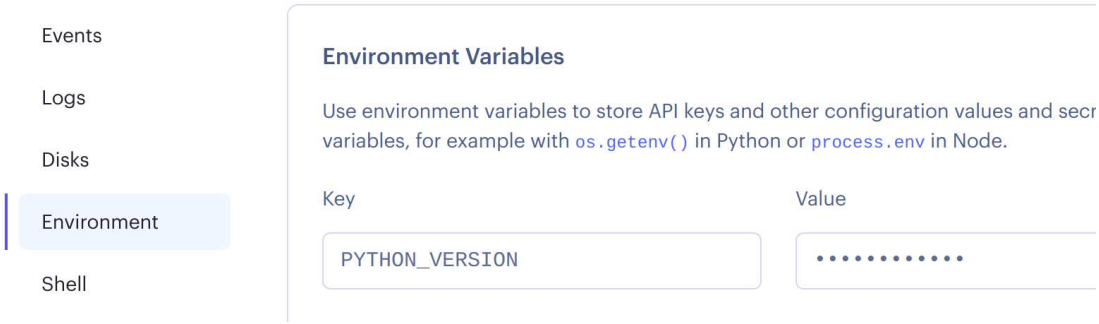

Рисунок 3.22 – Изменение версии Python на 3.8

Теперь сделаем вывод картинки.

Создадим еще один файл с шаблоном html (image.html) и поместим его в templates. Можете назвать его по-другому, если захотите.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Images</title>
   <link href="{{ url_for('static', path='/styles.css') }}" rel="stylesheet">
</head>
<body>
<h1> Images </h1>
 <table>
  <tr>
   <td> Изображение static </td>
   <td> Изображение из буфера </td>
  </tr>
  <tr>
   <td> <img src = "{{ im_st }}"/>  </td>
   <td> <img src = "{{ im_dyn }}"/> </td>
  </tr>
</table>
</body>
</html>
```

Как видите, в нем есть две переменные в фигурных скобках, то, что находится в этих скобках, воспринимается интерпретатором как выражение.

В том числе url_for есть выражение, вызывающее функцию, которая у нас появится и в файле fastlab.py.

В css файл style.css добавьте строки, чтобы у таблицы были границы:

```
table, th, td {
  border: 1px solid black;
}
```

В Pipfile добавьте строки:

```
jinja2 = "*"
numpy = "*"
Pillow = "*"
```

Далее укажем, что теперь у нас будет в файле fastlab.py.

```
import fastapi.responses
```

```
import numpy
import io
from PIL import Image
from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
app = FastAPI()
def sum_two_args(x,y):
    return x+y
# Hello World route
@app.get("/")
def read_root():
    return {"Hello": "World"}
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")
# возвращаем some.html, сгенерированный из шаблона
# передав туда одно значение something
@app.get("/some_url/{something}", response_class=HTMLResponse)
async def read_something(request: Request, something: str):
    return templates.TemplateResponse("some.html", {"request": request, "something":
something})
def create_some_image(some_difs):
    imx = 200
    imy = 200
    image = numpy.zeros((imx,imy, 3), dtype=numpy.int8)
    image[0:imy//2,0:imx//2,0] = some_difs
    image[imy//2:,imx//2:,2] = 240
    image[imy//2:,0:imx//2, 1] = 240
    return image
# возврат изображения в виде потока медиа данных по URL
@app.get("/bimage", response_class=fastapi.responses.StreamingResponse)
async def b_image(request: Request):
    # рисуем изображение, сюда можете вставить GAN, WGAN сети и т. д.
    # взять изображение из массива в Image PIL
    image = create_some_image(100)
    im = Image.fromarray(image, mode="RGB")
    # сохраняем изображение в буфере оперативной памяти
    imgio = io.BytesIO()
    im.save(imgio, 'JPEG')
    imgio.seek(0)
    # Возвращаем изображение в виде mime типа image/jpeg
    return fastapi.responses.StreamingResponse(content=imgio,
media_type="image/jpeg")
# возврат двух изображений в таблице html, одна ячейка ссылается на url bimage
# другая ячейка указывает на файл из папки static по ссылке
# при этом файл туда предварительно сохраняется после генерации из массива
@app.get("/image", response_class=HTMLResponse)
async def make_image(request: Request):
    image_n = "image.jpg"
    image_dyn = request.base_url.path+"bimage"
    image_st = request.url_for("static", path = f'/{image_n}')
```

```
        image = create_some_image(250)
        im = Image.fromarray(image, mode="RGB")
        im.save(f"./static/{image_n}")
        # передаем в шаблон две переменные к которым сохранили url-ы
        return templates.TemplateResponse("image.html", {"request": request,
"im_st":image_st, "im_dyn": image_dyn})
```

Можно теперь опять запустить наш проект из терминала.
uvicorn --host 127.0.0.1 --port 8000 fastlab:app

Далее открываем по ссылке браузер и вводим адреса
127.0.0.1:8000/image
127.0.0.1:8000/bimage

Смотрим, что получилось. Далее можно сделать commit и деплой, посмотреть на результат. Если возникают проблемы, нужно проверить совместимость библиотек и версии Python (здесь 3.8.10). Установить переменную Environment на render.com PYTHON_VERSION 3.8.10. Либо вашу версию Python.

3.13.10 Добавление работы с формами

Теперь рассмотрим работу с формами на fastapi. Сделаем простейшую возможность открытия файлов с изображениями, их изменения и вывода на странице. Здесь уже появится выполнение post запроса и передача данных формы через multipart формат.

Добавим в файл fastapi следующий код, содержащий обработку get и post запросы с формой.

```
from fastapi import Form, File, UploadFile
from typing import List
import hashlib
from PIL import ImageDraw

@app.post("/image_form", response_class=HTMLResponse)
async def make_image(request: Request,
                     name_op:str = Form(),
                     number_op:int = Form(),
                     r:int = Form(),
                     g:int = Form(),
                     b:int = Form(),
                     files: List[UploadFile] = File(description="Multiple files as UploadFile")
                     ):
    # устанавливаем готовность прорисовки файлов, можно здесь проверить, что
файлы вообще есть
    # лучше использовать исключения
    ready = False
    print(len(files))
    if(len(files)>0):
        if(len(files[0].filename)>0):
            ready = True
    images = []
    if ready:
        print([file.filename.encode('utf-8') for file in files])
        #  преобразуем имена файлов в хэш строку
```

```
            images = ["static/"+hashlib.sha256(file.filename.encode('utf-8')).hexdigest() for file in files]
            # берем содержимое файлов
            content = [await file.read() for file in files]
            # создаем объекты Image типа RGB размером 200 на 200
            p_images = [Image.open(io.BytesIO(con)).convert("RGB").resize((200,200)) for con in content]
            # сохраняем изображения в папке static
            for i in range(len(p_images)):
                draw = ImageDraw.Draw(p_images[i])
                # Рисуем красный эллипс с черной окантовкой.
                draw.ellipse((100, 100, 150, 200+number_op), fill=(r,g,b), outline=(0, 0, 0))
                p_images[i].save("./"+images[i],'JPEG')
        # возвращаем html с параметрами-ссылками на изображения, которые потом будут
        # извлечены браузером запросами get по указанным ссылкам в img src
        return templates.TemplateResponse("forms.html", {"request": request, "ready": ready, "images": images})

    @app.get("/image_form", response_class=HTMLResponse)
    async def make_image(request: Request):
        return templates.TemplateResponse("forms.html", {"request": request})
```

Добавим код шаблона с формой, в этом шаблоне происходит проверка готовности изображений и вывод их в таблицу в виде ссылок на сохраненные файлы в static. Здесь используется шаблон if и for, которые воспринимаются как условный оператор и цикл для того, чтобы в ячейках таблицы отобразить несколько выбранных изображений. В конце расположения есть кнопка submit, после нажатия которой осуществляется post запрос на сервер.

Создадим файл forms.html. В данном html используется форма с со слайдерами, которые позволяют выбирать значение из диапазона с помощью бегунка, а также элементы ввода строки и числового значения.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
 <h2>Форма ввода данных для изображений</h2>
    <form action="image_form" enctype="multipart/form-data" method="post">
        <p>
            Имя операции <br>
            <input name="name_op" value="operation" />
        </p>
        <p>
            Введите число <br>
            <input name="number_op" type="number" value = "0"  />
        </p>
        <table>
```

```
            <tr>
              <td><input type="range" id="r" name="r" value="123"
oninput="x1.value=r.value" min="0" max="255" > </td>
              <td><output name="x1" for="r"> 123 </output></td>
            </tr>
            <tr>
              <td><input type="range" id="g" name="g" value="123"
oninput="x2.value=g.value" min="0" max="255" > </td>
              <td><output name="x2" for="g"> 123 </output></td>
            </tr>
            <tr>
              <td><input type="range" id="b" name="b" value="123"
oninput="x3.value=b.value" min="0" max="255" > </td>
              <td><output name="x3" for="b"> 123 </output></td>
            </tr>
            </table>
            <input name="files" type="file" multiple>
            <p>
            <input type="submit" value="Send" />
            </p>
          </form>
        {% if ready %}
        <table>
        {% for image in images %}
        <tr>
          <td>
            <img src = "{{ image }}">
          </td>
        </tr>
         {% endfor %}
        </table>
        {% endif %}
        </body>
        </html>
```

Не забудьте добавить в Pipfile
python-multipart = "*"
или установить
pip install python-multipart

3.13.11 Пример использования Captcha

Для использования Captcha совместно с FastAPI установим сначал следующие библиотеки.

```
pip install fastapi
pip install uvicorn
pip install jinja2
pip install python-multipart
pip install pillow
pip install requests
```

Создайте Captcha на сайте Google.

https://www.google.com/recaptcha/admin/create

Выберите вторую версию. Введите имена доменов Domain, если проверяете локально, то localhost, для вашего хостинга укажите ваш адрес. Скопируйте Site key и Server key.

Создадим файл maim.py с запускаемым локально сервером.

```
from fastapi import FastAPI, Request, Form, HTTPException
from fastapi.templating import Jinja2Templates
import uvicorn
from PIL import Image
from fastapi.staticfiles import StaticFiles
import requests
app = FastAPI()
# указываем папку с шаблонами
templates = Jinja2Templates(directory="templates")
# монтируем папку static
app.mount("/static", StaticFiles(directory="static"), name="static")

# возвращаем основной обработанный шаблон index.html
@app.get("/")
def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

# обрабатываем post запрос с данными формы и Captcha
@app.post("/rotate_cross")
async def rotate_cross(request:Request, angle:int = Form(), color:str = Form(), resp:str = Form()):
    secret_key = "Your secret key"  # Замените на ваш секретный ключ
    # подготовим секретный ключ и ответ пришедший от браузера со стороны клиента
    payload = {
      "secret": secret_key,
      "response": resp
    }
    # посылаем post запрос на сайт Google для проверки прохождения Captcha
    response = requests.post("https://www.google.com/recaptcha/api/siteverify", data=payload)
    result = response.json()
    # проверяем успешность
    if result["success"]:
      # Открыть изображение которое лежит в папке static
      with Image.open(f"./static/is1.jpg") as img:
        # Rotate the image by the specified angle
        rotated_img = img.rotate(angle)
        # Save the rotated image with a new filename
        rotated_img.save(f"./static/{color}_rotated.jpg")
      return templates.TemplateResponse("result.html", {"request": request, "filename": f"{color}_rotated.jpg"})
    else:
      # в случае неуспеха проверки Captcha возвращаем 400
      raise HTTPException(status_code=400, detail="Ошибка проверки капчи")
# запускаем локально веб сервер
```

```
if __name__ == "__main__":
    uvicorn.run(app, host="localhost", port=8000)
```

Создадим папки static и templates, поместим в папку static файл styles.css, и в templates index.html и result.html.

Файл styles.css.

```
<style>
    form {
        background-color: #f4f4f4;
        padding: 20px;
        border-radius: 10px;
    }
    input {
        padding: 5px;
        margin: 5px;
        border: 1px solid #ccc;
        border-radius: 5px;
    }
    button {
        background-color: #4CAF50;
        color: white;
        padding: 10px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
</style>
```

Файл index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FastAPI Website Starter</title>
    <link rel="stylesheet" type="text/css" href="/static/styles.css">
    <script src="https://www.google.com/recaptcha/api.js" async defer></script>
     <script>
        var correctCaptcha = function(response) {

        document.getElementsByName('resp')[0].value = response;
                document.forms[0].submit();
        };
     </script>

</head>
<body>
<div class="container mt-5">
    <h1>Rotate Cross</h1>
    <form method="post" action="/rotate_cross" enctype="multipart/form-data">
        <div class="form-group">
            <label for="angle">Angle:</label>
```

```
                <input type="number" class="form-control" id="angle" name="angle"
required>
            </div>
            <div class="form-group">
                <label for="color">Color:</label>
                <input type="text" class="form-control" id="color" name="color"
required>
            </div>
            <div class="form-group">
                <input type="hidden" class="form-control" id = "resp" name="resp" value
= "0" required>
            </div>
            <div class="form-group">
                <div class="g-recaptcha" id="rcaptcha" data-sitekey="Your site key " data-
callback="correctCaptcha"> </div>
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>
    </body>
    </html>
```

Файл result.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FastAPI Website Starter</title>
</head>
<body>
<div class="container mt-5">
    <h1>Rotated Image</h1>
    <img src="{{ url_for('static', path=filename) }}" alt="Rotated Image" />
    <a href="/">Go back to home page</a>
</div>
 </body>
 </html>
```

3.14 Задание на лабораторную работу №1 «Разработка web приложения».

Необходимо реализовать веб-приложение в соответствии с заданием, используя какой-либо фреймворк и язык разработки (предлгается python). В методическом пособии предлагаются теоретические материалы для выполнения задания, используя язык python и фреймворки Flask или Fastapi, системы непрерывной интеграции Travis CI, github actions, и системы PaaS heroku или render.com. В работе должна использоваться Capcha, по возможности добавить решение задачи классификации картинок используя нейронные сети. Реализовать выгрузку проекта на github или любую другую VCS (система контроля версий) и простейшее тестирование, используя CI, при успешном тестировании должен осуществляться деплой на какую-либо PaaS систему. Входные параметры можно задавать, используя формы фреймворка, либо html формы. Задать стили CSS приложения, разместить результаты и формы в виде картинок в html таблицы или используя тэги <legend> и <fieldset>.

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). И выдывать график распределения цветов на картинке. График 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 картинки 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 соответствии с шахматным порядком, размер шахматной клетки задает пользователь в процентах от размера исходного изображения. Нарисовать график распределения цветов для исходного изображения и нового.

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 зависисости 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 исходого путем умножения изображения на периодическую функцию sin или cos с нормировкой, период изменения задает пользователь, аргумент функции определяется либо вертикальной или горизонтальной составляющей. Нарисовать график распределения цветов для нового и исходного изображения.

Вариант 19

Веб-приложение должно формировать новое изображение на основе исходого 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 исходого 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.

4 Лабораторная работа №4 «Разработка web-сервиса».

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

4.1 Создание отдельной среды окружения для проекта flasgger

Создадим отдельный проект, в котором рассмотрим кратко использование одной из библиотек позволяющей вести автоматическую документацию api с использованием swagger. Без документации ваше api бесполезно 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 в себе свои шаблоны и свои статичные файлы, это и есть основная концепция использования blueprint. Когда проект разрастается и есть 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 енвайронмент, 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 наше 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 (рисунок 4.1) или в формате json {

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

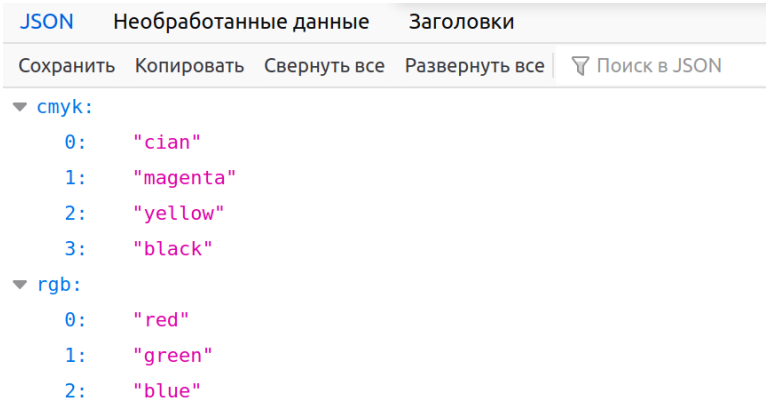

Рисунок 4.1 — Результат запроса по url к нашему api

Результат запроса apidocs отображен на рисунке 4.2.

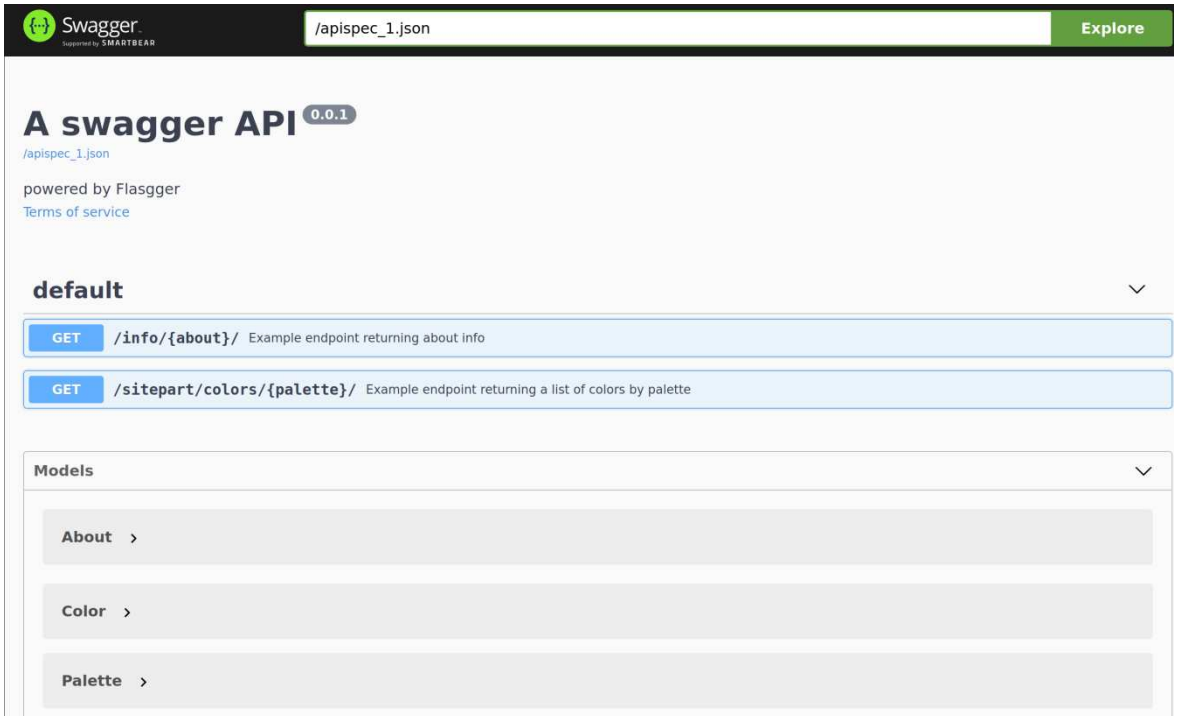

Рисунок 4.2 — Пример отображения информации об api

4.2 Библиотека 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 один енвайронмент.
(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 должно 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'.

Так же 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 видете здесь объект словарь сериализуется в строку, передаваемую пользователю, так же в компьютерных сетях под маршалингом 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. Так же маршалинг реализует представление объекта для получателя. При сериализации просто происходит преобразование объекта в битовую строку и обратно (в некоторых случаях сераилизация 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("")
        # маршаллинг 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 (рисунок 4.3)

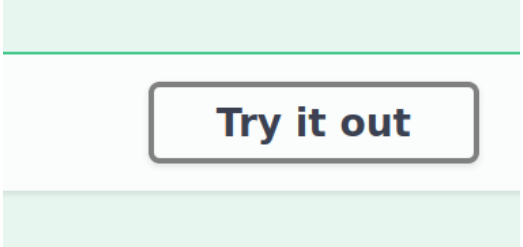

Рисунок 4.3 – Проверка нашего api

Данные можно ввести в открывшемся поле (Edit value), например, {**"len":"3", "array": ["11","5","3"]**} (рисунок 4.4).

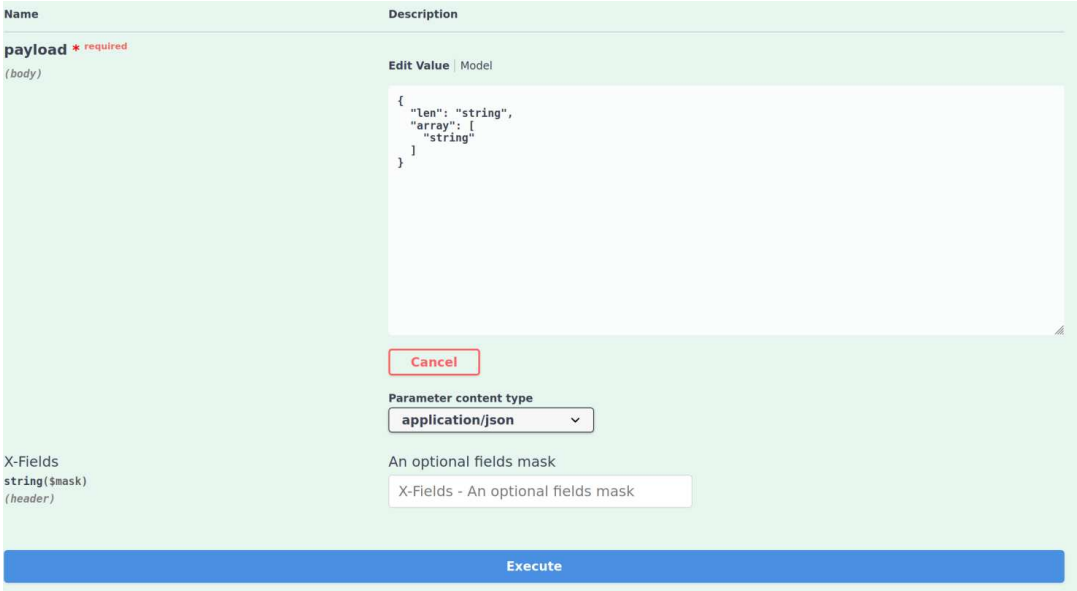

Рисунок 4.4 – Ввод данных для запроса Post

Далее добавим пример, где данные передаются в запросе get и в ответ возвращается результат. В данном примере в запросе Get передаются параметры, 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("")
        # маршаллинг 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 фактически 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 4.5, очевидно, что можно использовать и запросы put, patch и delete.

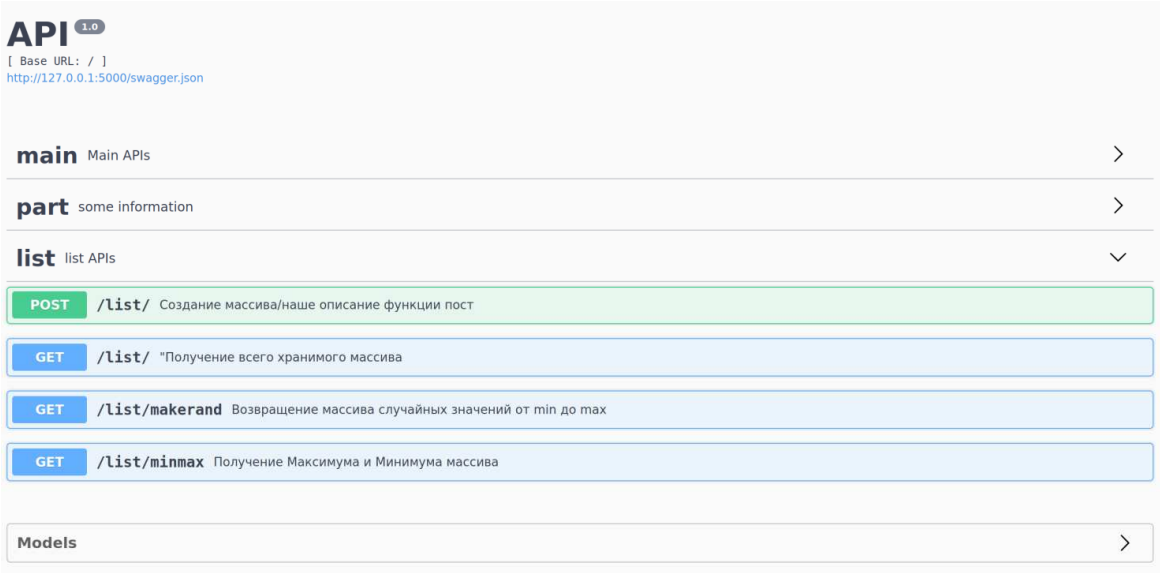

Рисунок 4.5 – Пример Swagger документации API

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

4.3 Примеры API функций и их документации на FastAPI

Для реализации REST API функций можно использовать pydantic. Здесь автоматически проводится проверка. Можно делать по аналогии с flasger и т.д.

```
from pydantic import BaseModel
class User(BaseModel):
   name: str
   age: int
```

```
@app.get('/users/{user_id}')
def get_user(user_id):
    return User(name="John Doe", age=20)
@app.put('/users/{user_id}')
def update_user(user_id, user: User):
    # поместите сюда код для обновления данных
    return user
```

Документацию можно получать по адресу url http://127.0.0.1:8000/docs.

4.4 Задание на лабораторную работу №4 «Разработка web-сервиса».

Необходимо повторить описанные в главе 3 примеры и реализовать 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 новые записи, удалять записи, 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»:
«ФИО»,
«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 с указанием предметной области.

1. Студенты, изучающие дисциплины в университете
2. Страны, язык страны, население стран
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

5 Требования к содержанию и оформлению отчета

Результатом выполнения лабораторных работ является отчёт, сохраненный в файле с расширением pdf, который высылаются на проверку преподавателю. 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 А.

Список использованных источников

1. Суханов А. Я. Разработка веб-сервисов для научных и прикладных задач : Учебное пособие / А. Я. Суханов, – Томск : ФДО, ТУСУР, 2021. – 244 с.
2. Официальная документация Flask [Электронный ресурс]. URL: https://flask.palletsprojects.com/en/1.1.x/ (дата обращения: 25.06.2023)
3. Крупнейшая вики об WSGI. [Электронный ресурс]. http://wsgi.org/ (дата обращения: 28.03.2020)
4. Python. https://www.python.org/ (дата обращения: 21.07.2023)
5. Бэрри, П. Изучаем программирование на Python / П. Бэрри. - Москва : Эксмо, 2018. - 624 с. : ил.
6. Мэтиз, Э. Изучаем Python : программирование игр, визуализация данных, веб-приложения / Э. Мэтиз. - 2-е изд. - Санкт-Петербург : Питер, 2018. - 496 с. : ил.
7. Miguel Grinberg. Проектирование RESTful API с помощью Python и Flask [Электронный ресурс]. https://habr.com/ru/post/246699/ (дата обращения: 28.07.2023)
8. Miguel Grinberg. Мега-Учебник Flask, Часть 1: «Привет, Мир!» [Электронный ресурс]. https://habr.com/ru/post/193242/ (дата обращения: 28.07.2023)

ПРИЛОЖЕНИЕ А

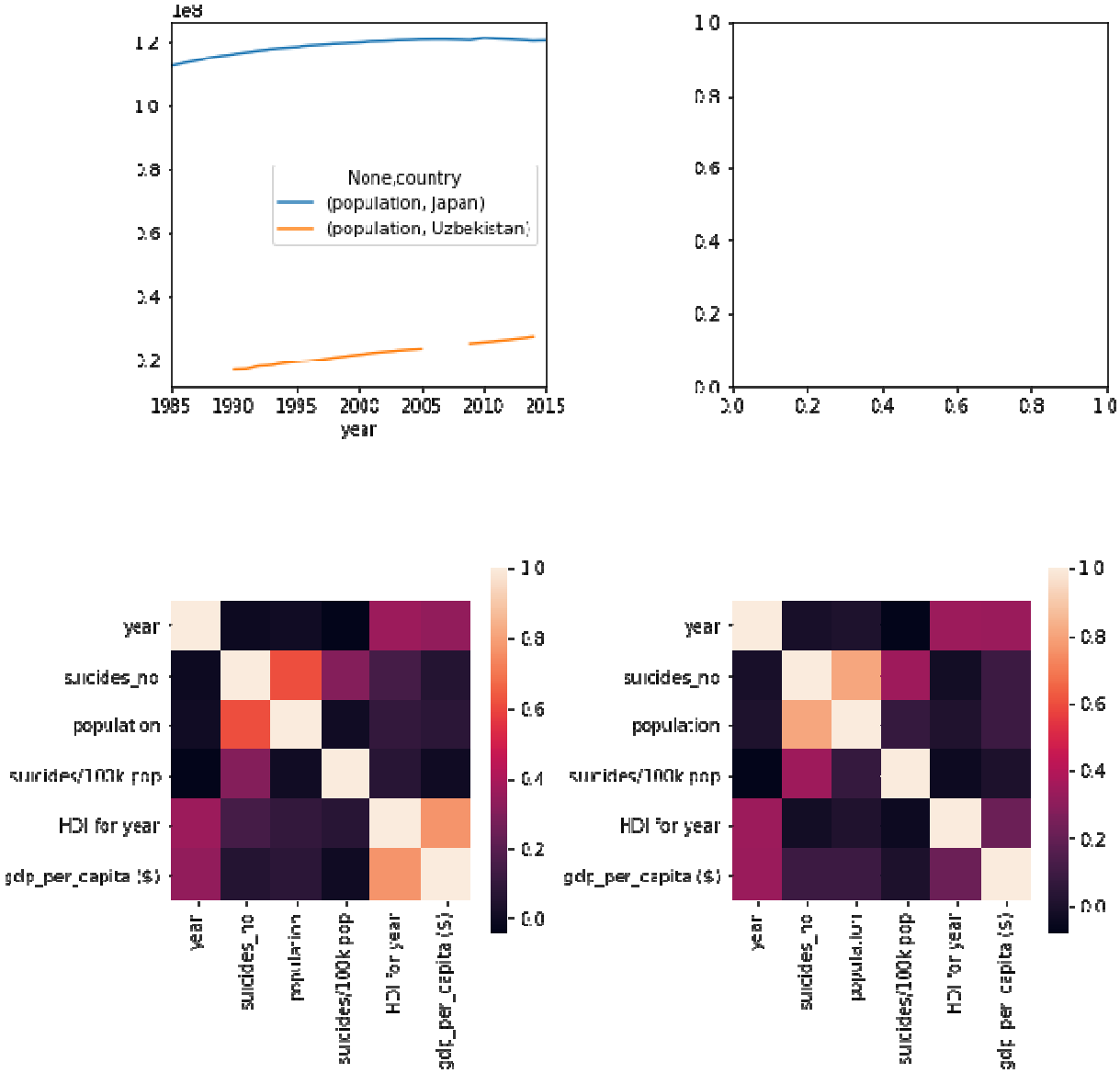

Рисунок А.1 – Графики с использованием цветовой карты

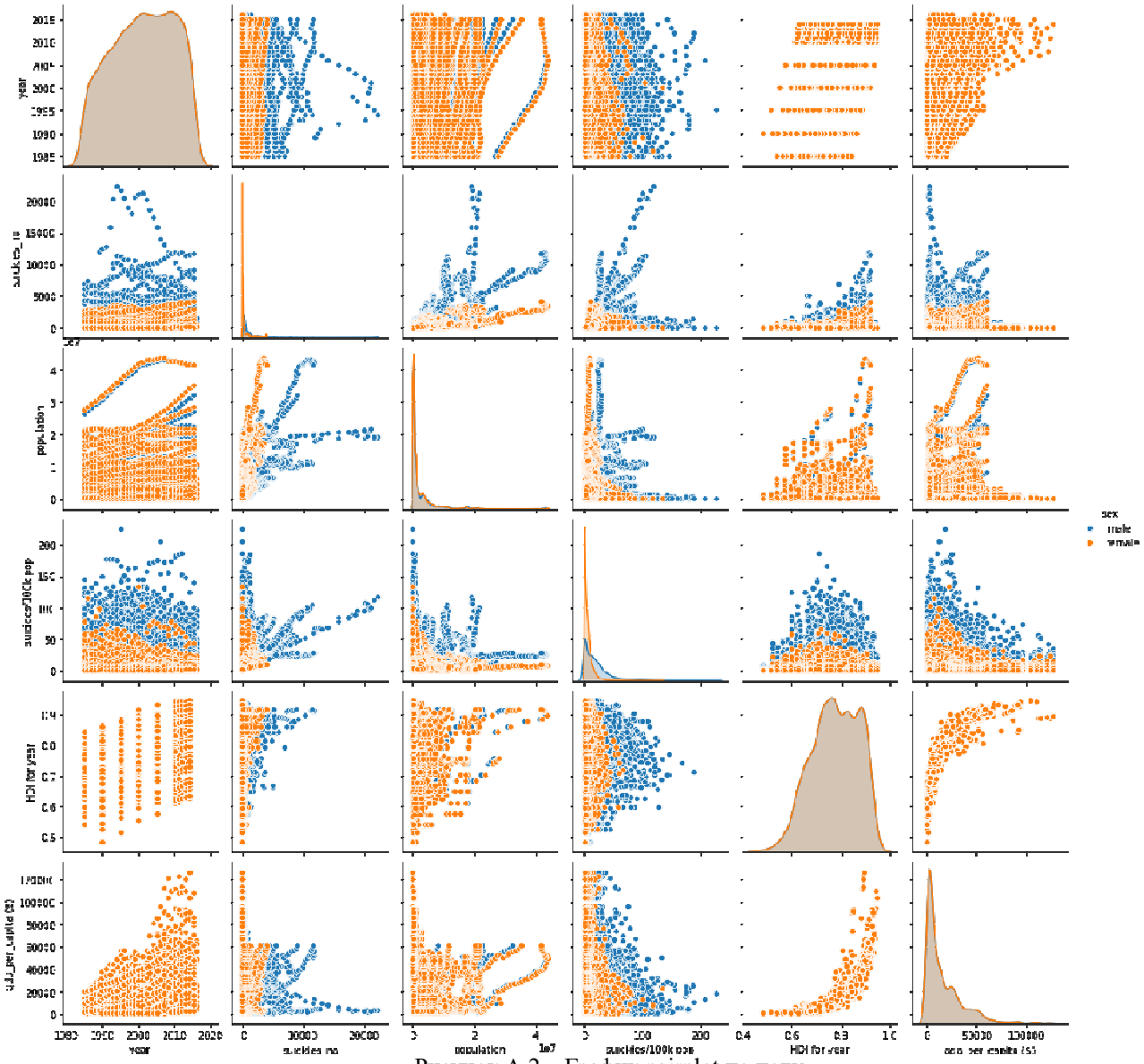

Рисунок А.2 – График pairplot по полу

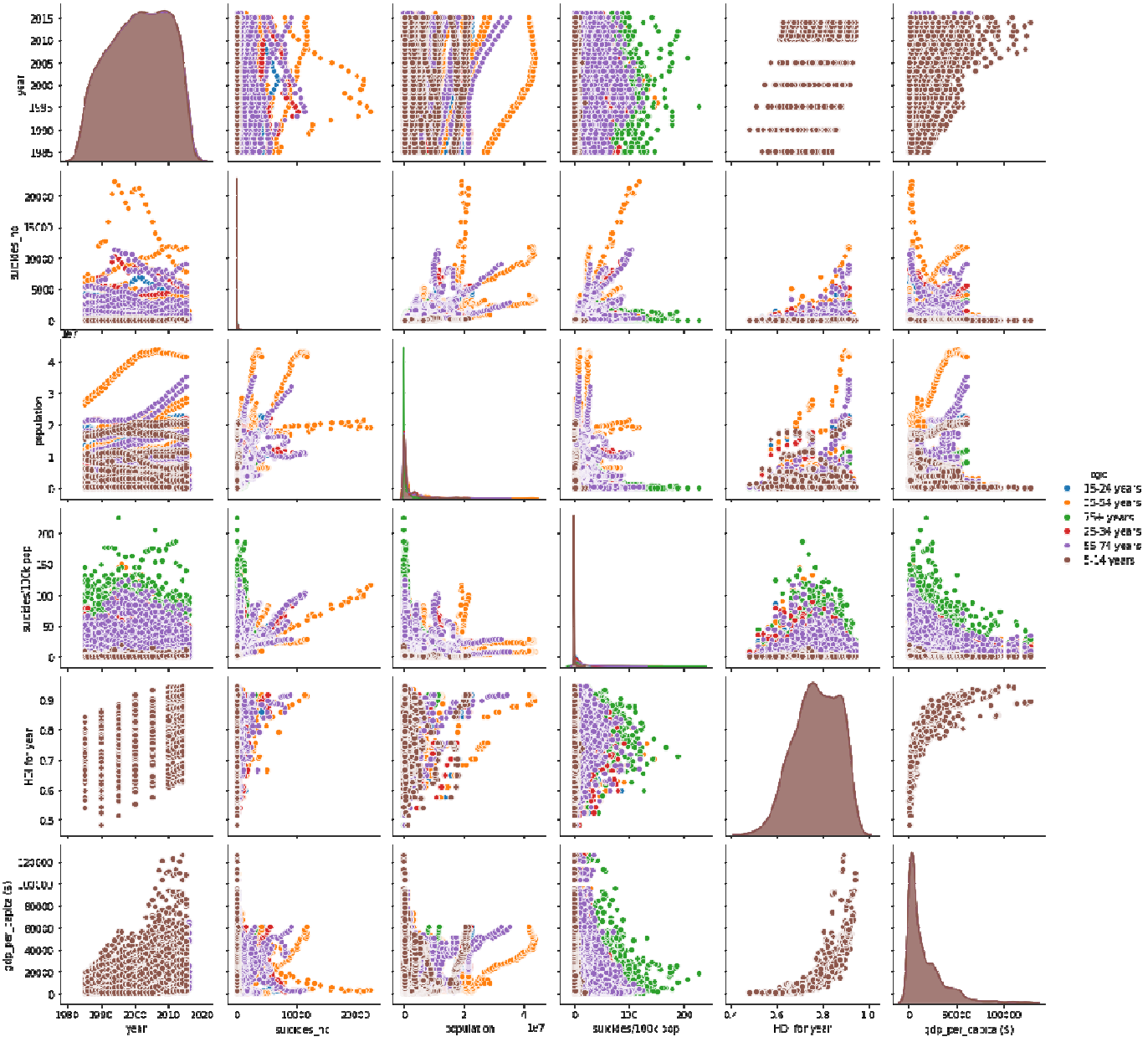

Рисунок А.3– График pairplot по возрасту

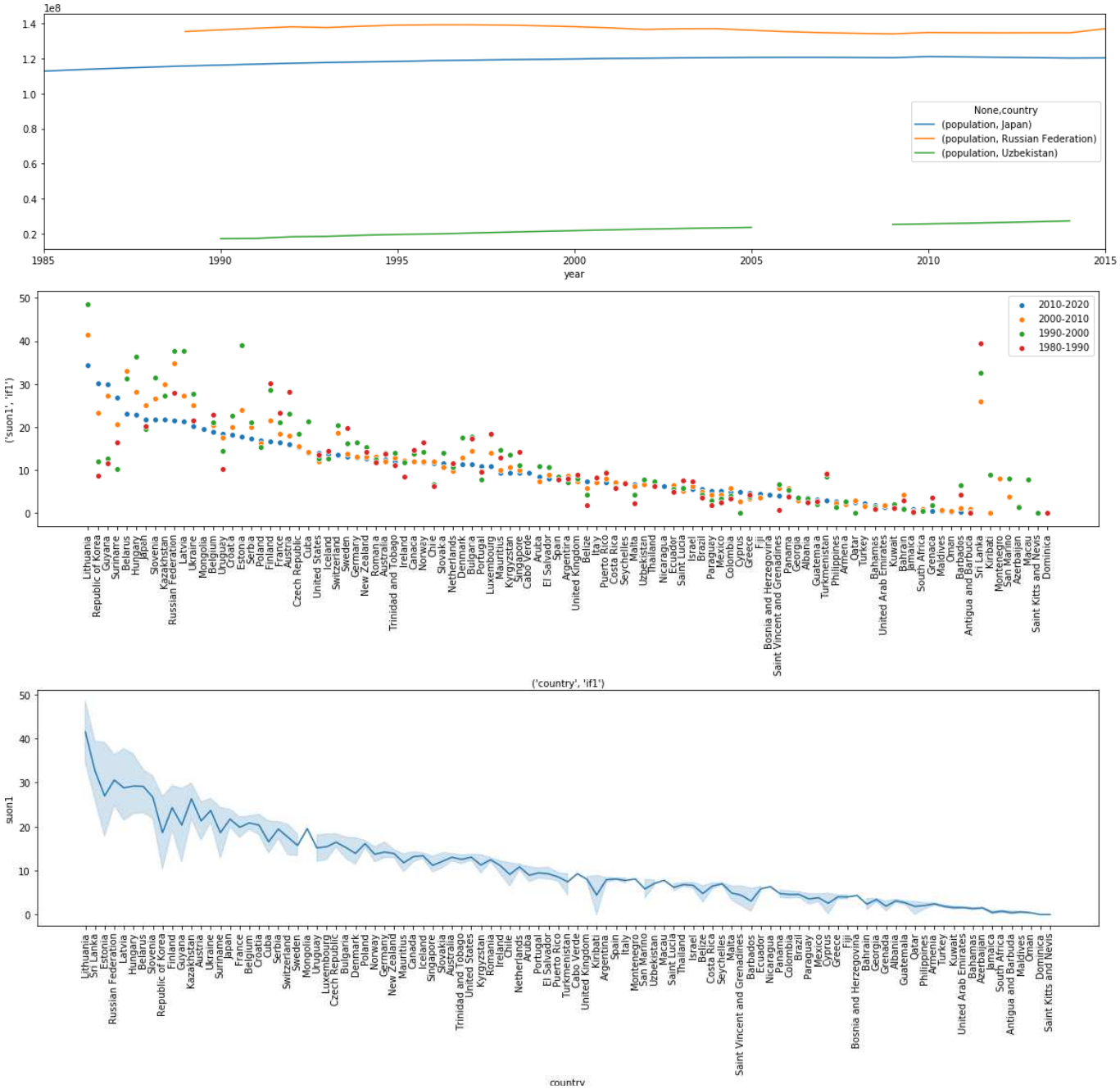

Рисунок А.4 – График случаев по странам

ПРИЛОЖЕНИЕ Б

Пример оформления титульного листа отчета

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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__