**Вводный курс по Big Data. Обучение в записи**

**Урок 2. Семинар: Введение в Большие Данные**

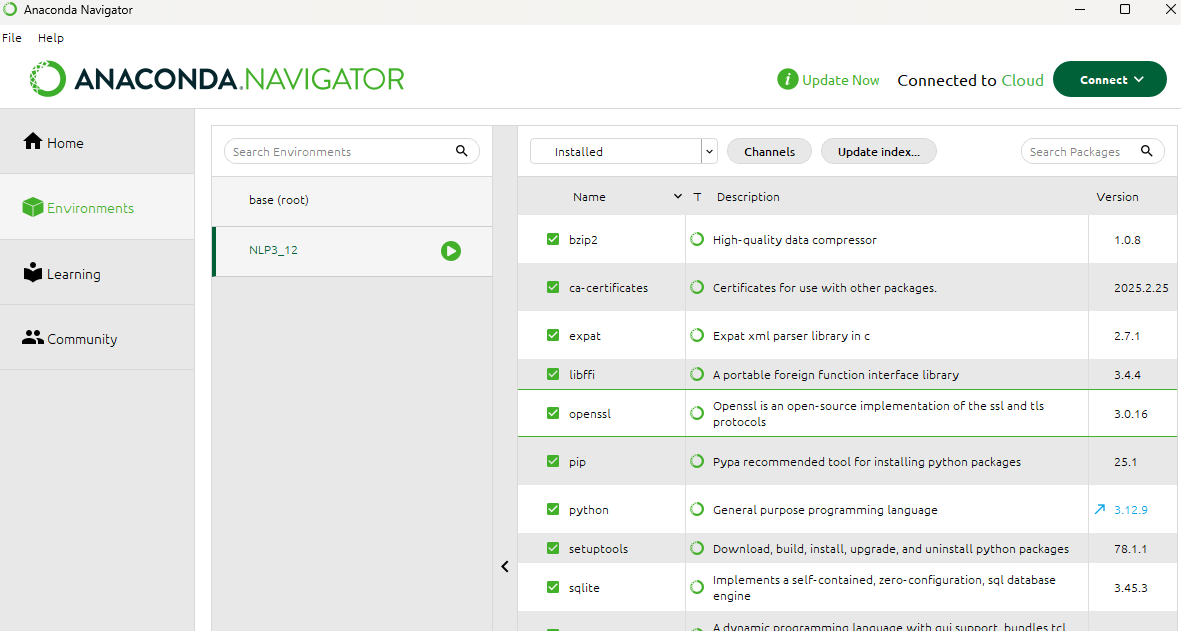
Оглавление

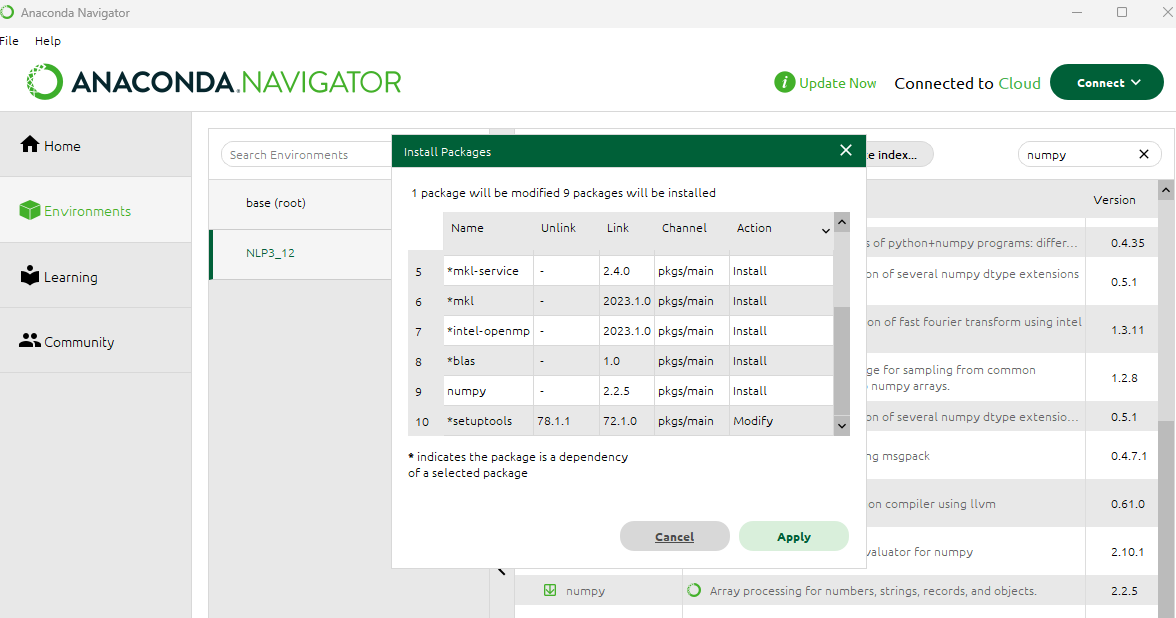
[Семинар. Установка и настройка окружения Anaconda 2](#_Toc200289876)

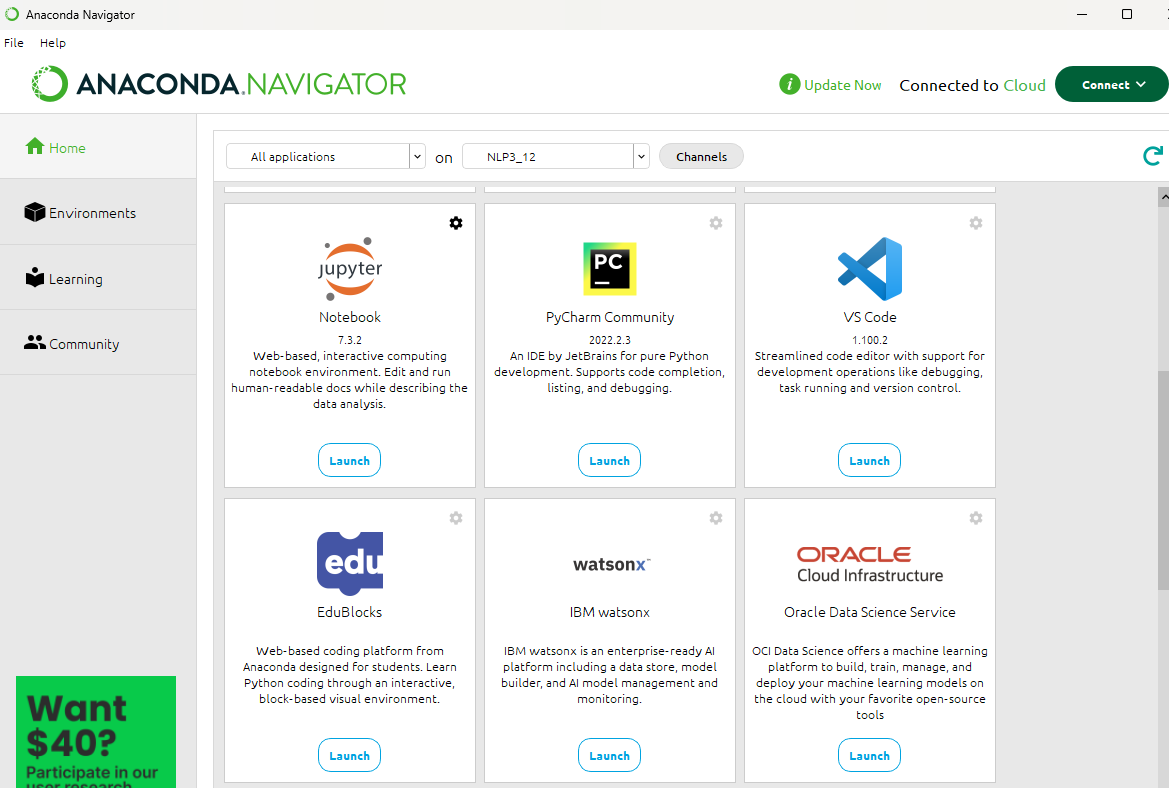
[Семинар. Выполнение практических задач 4](#_Toc200289877)

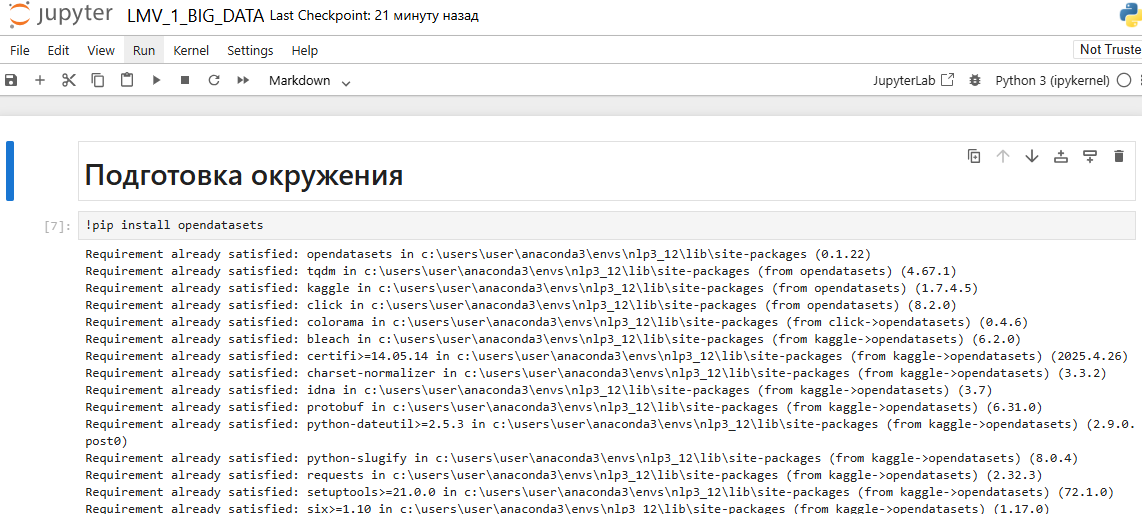
[Домашняя работа 7](#_Toc200289878)

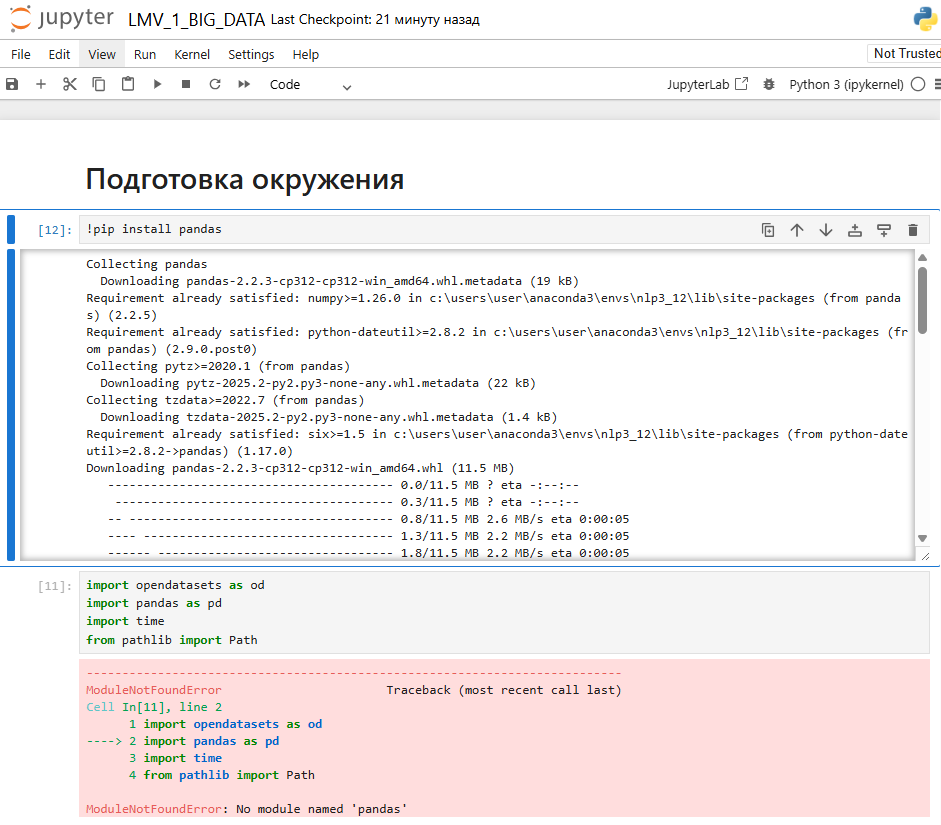
### Семинар. Установка и настройка окружения Anaconda







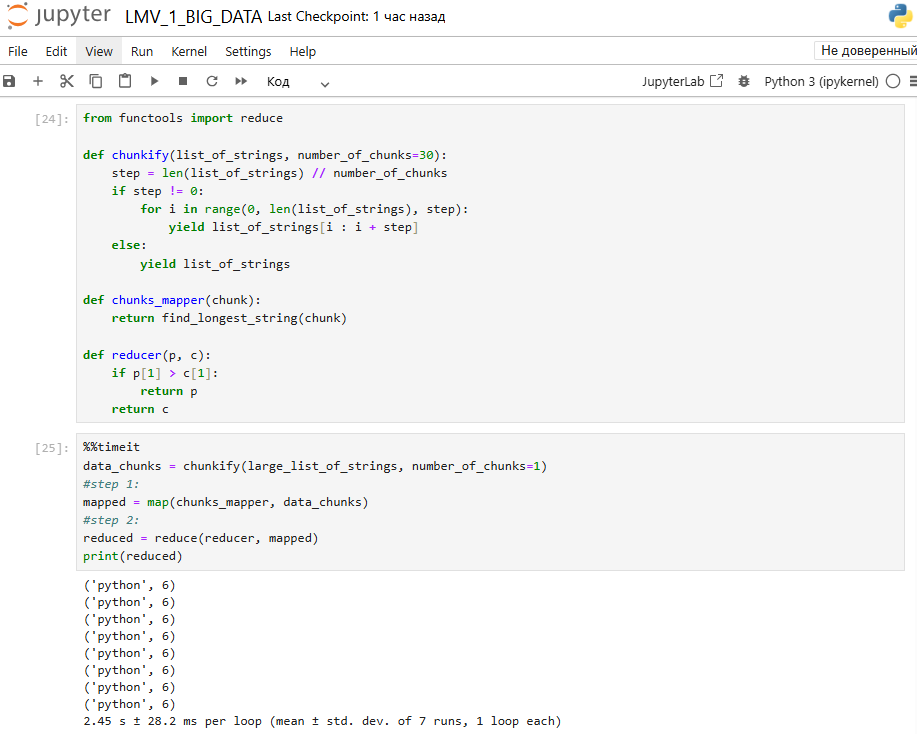




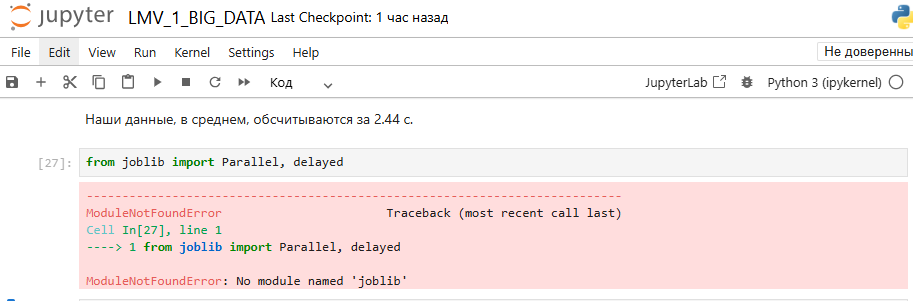
Установка Pandas и загрузка ключа (токена) Kaggle …

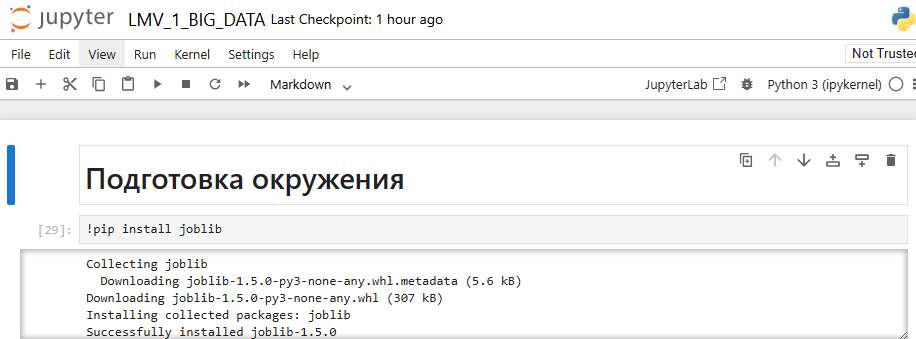
### Семинар. Выполнение практических задач

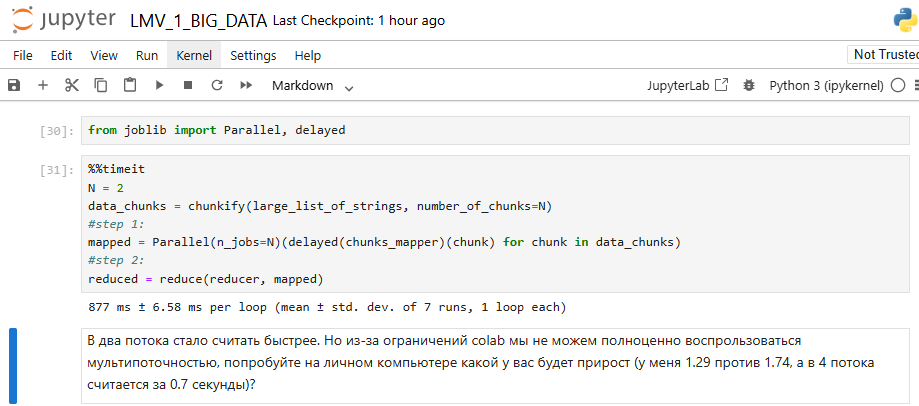


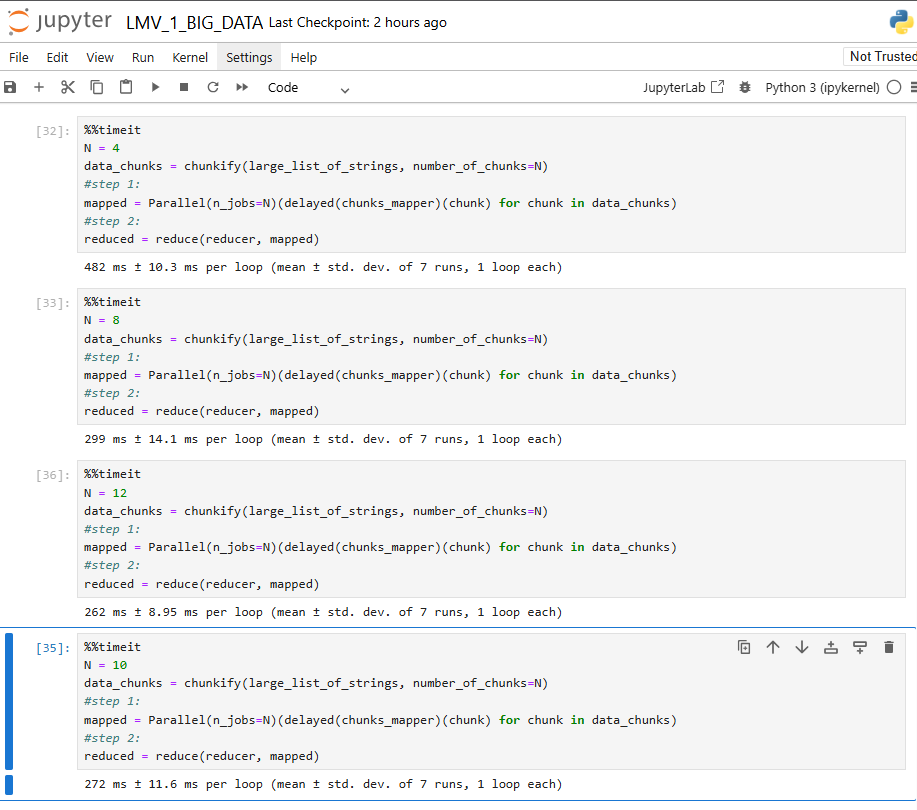












### Домашняя работа

**Ознакомьтесь с библиотеками для распараллеливания задач в Python:**

1.multiprocessing

2.joblib

3.dask

**Условия и критерии выполнения домашнего задания внутри файла**

1. **MULTIPROCESSING**

**Модуль multiprocessing в Python** позволяет запускать несколько процессов одновременно для параллельного выполнения задач. Каждый процесс имеет собственную память и независимое выполнение, в отличие от потоков, которые работают в рамках одного процесса.

Это позволяет обходить ограничение Global Interpreter Lock (GIL), что даёт возможность эффективно использовать многоядерные процессоры.

**Основные понятия**

Некоторые ключевые элементы модуля multiprocessing:

*Класс Process* — для создания и управления процессами. Позволяет запускать функции в новом процессе.

*Класс Pool* — предоставляет пул процессов для выполнения задач параллельно. Позволяет распределять задачи между процессами.

*Класс Queue* — для обмена данными между процессами через очередь.

*Классы Lock, Event, Condition, Semaphore* — для синхронизации процессов и предотвращения состязаний при доступе к общим ресурсам.

**Преимущества и недостатки**

Преимущества:

* истинный параллелизм для CPU-задач;
* изоляция процессов (сбой в одном не влияет на другие).

Недостатки:

* высокие накладные расходы на создание процессов;
* сложности с обменом данными (требуется сериализация через pickle);
* большее потребление памяти.

Примеры использования

Модуль multiprocessing подходит для работы с CPU-зависимыми задачами, такими как сложные вычисления, обработка изображений, научные расчёты. Также его можно использовать для распараллеливания большого объёма данных или ускорения итераций тяжёлых циклов.

Если основная нагрузка связана с вводом-выводом (например, сетевые запросы или работа с файлами), лучше использовать асинхронное программирование или многопоточность.

1. **JOBLIB**

Joblib — библиотека для распараллеливания задач в Python, которая предоставляет инструменты для параллельных вычислений, кэширования и эффективной обработки данных.

**Основные функции и возможности**

Распараллеливание вычислений с помощью класса Parallel. Он позволяет распределить выполнение итераций цикла или вызовы функций на несколько ядер процессора.

Кэширование результатов с помощью класса Memory. При повторном вызове функции с одинаковыми аргументами результат не вычисляется заново, а возвращается из кэша.

Сохранение и загрузка объектов Python с помощью функций joblib.dump() и joblib.load(). Поддерживаются различные методы сжатия данных, например zlib, gzip, bz2, xz.

**Примеры использования**

Распараллеливание цикла. Например, вычисление квадратного корня для каждого элемента списка чисел. Без параллельных вычислений это работает последовательно, а с помощью Joblib можно распараллелить цикл.

Параллельная загрузка файлов. Например, импорт нескольких файлов CSV или Excel. Joblib позволяет распределить задачи между несколькими ядрами или потоками, сократив общее время загрузки.

1. **DASK**

Dask — библиотека Python для параллельных вычислений и масштабируемой обработки данных. Она позволяет работать с данными, которые не помещаются в оперативную память, и исполнять задачи распределённо на многопроцессорных машинах или кластерах.

**Основные возможности**

Разбиение данных на части (чанки) для параллельной обработки.

Поддержка ленивых вычислений — операции выполняются только при необходимости.

Интеграция с другими библиотеками Python, такими как Pandas, NumPy и Scikit-learn.

Масштабируемость — работа от одного ядра до кластера.

Визуализация вычислений и мониторинг в браузере.

**Установка и подключение**

Установка можно использовать пакетный менеджер pip или conda. Например, в дистрибутиве Anaconda Dask установлен по умолчанию.

Подключение осуществляется через импорт модулей, например import dask.dataframe as dd или import dask.array as da.

**Примеры использования**

Обработка большого CSV-файла. Вместо загрузки всего файла в память Dask загружает его частями, а затем выполняет параллельные операции, такие как фильтрация и группировка.

Анализ логов веб-сервера. Например, подсчёт количества запросов к каждой странице.

Обработка финансовых данных для вычисления скользящего среднего.

**PYTHON**

import json

from pathlib import Path

from functools import reduce

# Функция mapper: извлекает оценку и вычисляет промежуточные значения

def mapper(path):

n, mean, M2 = 0, 0.0, 0.0

if path.is\_file() and path.suffix == '.json':

with open(path, 'r') as f:

info = json.load(f)

score = float(info['movieIMDbRating'])

n += 1

delta = score - mean

mean += delta / n

M2 += delta \* (score - mean)

return n, mean, M2

# Функция reducer: объединяет промежуточные результаты

def reducer(score\_data1, score\_data2):

n1, mean1, M21 = score\_data1

n2, mean2, M22 = score\_data2

n = n1 + n2

if n == 0:

return 0, 0.0, 0.0

delta = mean2 - mean1

mean = mean1 + delta \* n2 / n

M2 = M21 + M22 + delta \*\* 2 \* n1 \* n2 / n

return n, mean, M2

# Измерение времени выполнения всего кода

%%time

# Применяем mapper и объединяем результаты с помощью reducer

n, mean, M2 = reduce(reducer, map(mapper, Path('imdb-user-reviews').glob('\*\*/\*')))

# Выводим результат

if n > 0:

print(f"Среднее: {mean}, Стандартное отклонение: {(M2 / n) \*\* (1/2)}")

else:

print("Нет данных для обработки")

Ссылка на репозиторий:

<https://github.com/olgashenkel/GeekBrains-specialization-ELECTIVES/tree/main/12.%20Introductory%20course%20on%20Big%20Data>