### ****Курс:**** Python-разработчик**.**

**Дипломная работа**

**Тема: Анализ и сравнение различных способов**

**обработки и хранения больших данных:**

**Pandas, Dask и Apache Spark**

**(**Разработать приложения для обработки больших данных с использованием пакетов Pandas, Dask и Apache Spark, сравнить их.**)**

**Выполнил: слушатель университета**

**Бессонов И.И.**

**Москва**

**2024**

**Оглавление дипломной работы:**

1. **Введение.**

**1.1 Обоснование выбора темы.**

**1.2 Определение цели и задач исследования.**

1. **Теоретическая часть.**

**2.1. Основные понятия.**

* 1. **Исследуемые приложения, обзор и сравнение пакетов.**

1. **Разработка приложений для обработки больших данных с использованием** Pandas, Dask и Apache Spark **и проведение расчетов для сравнительного анализа полученных результатов по трем исследуемым приложениям:**

3.1 Подготовка к исследованию, исследование Pandas.

* 1. Исследование Dask.
  2. Исследование Apache Spark.

1. **Анализ и интерпретация результатов.**
2. **Заключение.**

#### **1. Введение.**

1.1 **Обоснование выбора темы.**

В условиях стремительно растущих объемов данных, эффективная обработка и анализ больших данных становятся критически важными во многих сферы человеческой жизнедеятельности. В этой работе рассматриваются три популярных инструмента обработки и хранения больших данных: Pandas, Dask и Apache Spark, сравниваются возможности, производительность и области применения. Проведен анализ каждого инструмента посредством примеров кода на Python3, чтобы выявить их сильные и слабые стороны.

### 1.2 Цели и задачи исследования.

**Цель:** Провести анализ методов обработки и хранения больших данных с использованием библиотек Python3: Pandas, Dask и Apache Spark.

**Задачи:**

* Изучить основные понятия обработки больших данных;
* Рассмотреть возможности и методологию применения каждого инструмента;
* Проанализировать производительность через разработанные примеры тестового учебного кода.

### 2. Теоретическая часть.

**2.1. Основные понятия.**

* **Большие данные:** Данные, объем, скорость и разнообразие которых превышает возможности традиционных инструментов для анализа.
* **Обработка данных:** Процесс изыскания информации из больших наборов данных с применением различных методов и технологий.
* **Хранение данных:** Технологии, позволяющие сохранять и извлекать данные с высоким уровнем доступности и скорости.

**2.2. Исследуемые приложения.**

* **Pandas:** Библиотека для анализа данных в формате таблиц, хорошо подходит для группировки, фильтрации и агрегации небольших и средних наборов данных.
* **Dask:** Расширение Pandas, которое поддерживает параллельные вычисления и асинхронную обработку для работы с большими данными, которые не помещаются в память.
* **Apache Spark:** Высокоэффективная распределенная система для обработки больших данных, поддерживающая различные операции, включая потоковые данные.

2.2.1 **Pandas**

**Pandas** — [программная библиотека](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0) на языке [Python](https://ru.wikipedia.org/wiki/Python) для обработки и [анализа данных](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Работа pandas с данными строится поверх библиотеки [NumPy](https://ru.wikipedia.org/wiki/NumPy), являющейся инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми [таблицами](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [временными рядами](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9_%D1%80%D1%8F%D0%B4). Название библиотеки происходит от [эконометрического](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D0%BA%D0%B0) термина [«панельные данные»](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BD%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), используемого для описания многомерных структурированных наборов информации. Pandas распространяется под новой [лицензией BSD](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_BSD).

Основная область применения — обеспечение работы в рамках среды Python не только для сбора и очистки данных, но для задач анализа и моделирования данных, без переключения на более специфичные для статистической обработки языки.

Пакет прежде всего предназначен для очистки и первичной оценки данных по общим показателям, например среднему значению, [квантилям](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%B8%D0%BB%D1%8C) и так далее; [статистическим пакетом](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%B0%D0%BA%D0%B5%D1%82&action=edit&redlink=1) он в полном смысле не является, однако наборы данных типов DataFrame и Series применяются в качестве входных в большинстве модулей [анализа данных](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и [машинного обучения](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

Основные возможности библиотеки:

* Объект *DataFrame* для манипулирования индексированными массивами двумерных данных[:](https://ru.wikipedia.org/wiki/Pandas#cite_note-4)
* Инструменты для обмена данными между структурами в памяти и файлами различных форматов;
* Встроенные средства совмещения данных и способы обработки отсутствующей информации;
* Переформатирование наборов данных, в том числе создание сводных таблиц;
* Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных;
* Вставка и удаление столбцов данных;
* Возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение»;
* Слияние и объединение наборов данных;
* Иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности;
* Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее;

Библиотека оптимизирована для высокой производительности, наиболее важные части кода написаны на [Cython](https://ru.wikipedia.org/wiki/Cython) и [Си](https://ru.wikipedia.org/wiki/C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).

2.2.2. DASK (при подготовке главы использованы материалы статьи «Анализ данных с использованием библиотеки Dask» А. Михайлова)

**DASK -** это мощная библиотека для параллельных и распределенных вычислений в Python, предназначенная для работы с большими объемами данных. Она разработана с учетом того, чтобы предоставить инструменты для высокоуровневого управления вычислениями, которые могут быть выполнены параллельно или распределено на нескольких вычислительных узлах. Основной целью Dask является упрощение обработки данных, которые не помещаются в оперативной памяти одного компьютера. Dask может использоваться для выполнения разнообразных задач, включая анализ данных, обработку изображений, машинное обучение, и многое другое. Его фундаментальной концепцией является создание графа задач, который описывает вычисления и зависимости между ними. Затем этот граф может быть выполнен параллельно или распределено.

Dask предоставляет три основных компонента для работы с данными:  
1. Dask Arrays: Dask Arrays — это аналог NumPy массивов, разработанный для работы с большими объемами данных, которые не помещаются в оперативной памяти. Он предоставляет многие известные функции NumPy, такие как `numpy.array`, `numpy.sum`, `numpy.mean`, и так далее. Основное отличие заключается в том, что Dask Arrays разбивают данные на множество маленьких частей, которые могут быть обработаны параллельно.  
2. Dask DataFrames**:** Dask DataFrames — это аналог Pandas DataFrames, который позволяет работать с таблицами данных, которые не помещаются в памяти. Dask DataFrames поддерживает множество операций, таких как фильтрация, сортировка, группировка и агрегация данных, а также объединение таблиц.  
3. Dask Bags: Dask Bags — это структура данных, предназначенная для работы с неструктурированными данными, такими как JSON-объекты. Они могут представлять собой коллекцию элементов с разными полями. Dask Bags позволяет выполнять множество операций над данными, такие как фильтрация, маппинг и агрегация.Dask может читать данные из различных форматов файлов, таких как CSV, Parquet, JSON, и других. Также в пакете реализованы функции очистки и предобработки данных.

Dask обеспечивает встроенную поддержку параллельных вычислений, что позволяет использовать все доступные ядра процессора для ускорения вычислений. Это особенно полезно при выполнении операций на данных, которые можно разбить на части и обработать параллельно и выполнением операций на вычислительном кластере. Одним из сильных преимуществ Dask является его способность масштабирования для работы с большими вычислительными кластерами.

Для достижения максимальной производительности при работе с Dask можно использовать некоторые оптимизационные стратегии:

1. Управление памятью:  
Для оптимизации использования памяти можно настроить параметры блокировки данных и размеры блоков. Это позволяет более эффективно использовать доступную оперативную память и избегать ее избыточного использования.  
 **import** dask.array **as** da  
 # Установка размера блоков  
 da.from\_array(arr, chunks=(1000,))

2. Использование кэширования:  
Dask поддерживает кэширование результатов промежуточных вычислений. Это позволяет избежать повторного вычисления одних и тех же операций при пересчете данных.  
 **import** dask  
 **from** dask **import** delayed  
 # Включение кэширования  
 dask.config.set(scheduler='single-threaded', optimize='cache')

3. Использование индексов:  
Использование индексов в Dask DataFrames может значительно ускорить операции по выборке данных. При наличии индекса Dask будет выполнять операции выборки данных гораздо быстрее.  
 **import** dask.dataframe **as** dd  
 # Создание индекса по столбцу 'id'  
 df = df.set\_index('id')

Оптимизация производительности зависит от конкретных задач и характеристик данных, но Dask предоставляет множество инструментов и настроек для достижения лучших результатов при работе с большими объемами данных.

2.2.3 Apache Spark

Apache Spark (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *spark* — искра, вспышка) — [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/FOSS) для реализации распределённой обработки данных, входящий в экосистему проектов [Hadoop](https://ru.wikipedia.org/wiki/Hadoop). В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию [MapReduce](https://ru.wikipedia.org/wiki/MapReduce) с хранением промежуточных данных на накопителях, Spark работает в парадигме [резидентных вычислений](https://ru.wikipedia.org/w/index.php?title=%D0%A0%D0%B5%D0%B7%D0%B8%D0%B4%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F&action=edit&redlink=1) — обрабатывает данные в [оперативной памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач, в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов [машинного обучения](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

Проект предоставляет [программные интерфейсы](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для языков [Java](https://ru.wikipedia.org/wiki/Java), [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Python](https://ru.wikipedia.org/wiki/Python), [R](https://ru.wikipedia.org/wiki/R_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Изначально написан на [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), впоследствии добавлена существенная часть кода на Java для предоставления возможности написания программ непосредственно на Java. Состоит из ядра и нескольких расширений, таких как Spark SQL (позволяет выполнять [SQL](https://ru.wikipedia.org/wiki/SQL)-запросы над данными), Spark Streaming (надстройка для обработки потоковых данных), Spark MLlib (набор библиотек машинного обучения), GraphX (предназначено для распределённой обработки графов). Может работать как в среде кластера Hadoop под управлением [YARN](https://ru.wikipedia.org/wiki/YARN), так и без компонентов ядра Hadoop, поддерживает несколько распределённых систем хранения — [HDFS](https://ru.wikipedia.org/wiki/HDFS), [OpenStackSwift](https://ru.wikipedia.org/wiki/OpenStack#Swift), [NoSQL](https://ru.wikipedia.org/wiki/NoSQL)- СУБД  [Cassandra](https://ru.wikipedia.org/wiki/Cassandra).

В случае с нашим объемом данных и довольно легкими задачами для примеров наибольшее предпочтение отдается DASK.

2.2.4 Сравнение Dask с другими библиотеками для анализа данных.

Сравнение с Pandas и NumPy:

1. Pandas:

* Pandas — это популярная библиотека для анализа данных в Python, предоставляющая удобные структуры данных, такие как DataFrame и Series.
* Ограничения Pandas:
* Pandas загружает данные в память полностью, что может быть проблемой при работе с большими объемами данных.
* Операции в Pandas выполняются в однопоточном режиме, что может замедлить анализ больших наборов данных.
* Dask vs. Pandas:
* Dask предоставляет Dask DataFrame, который работает с данными, не помещая их полностью в память. Это позволяет анализировать данные, превышающие доступную память.
* Dask может распараллеливать операции, что делает его быстрее при обработке больших объемов данных.
* Когда использовать Dask вместо Pandas:
* *Используйте Dask, когда у вас есть слишком много данных для загрузки в память.*
* *В случае, когда вам нужно распараллелить вычисления для ускорения анализа данных.*

2. NumPy:

* NumPy — это библиотека для работы с многомерными массивами и математическими функциями в Python.
* Ограничения NumPy:
* Как и Pandas, NumPy загружает данные полностью в память, и это может вызвать проблемы при анализе больших массивов данных.
* NumPy не предоставляет высокоуровневых структур данных, таких как DataFrame.
* Dask vs. NumPy:
* *Dask позволяет создавать распределенные массивы данных, которые можно анализировать по частям, не загружая всю структуру в память.*
* Dask поддерживает многие операции, аналогичные операциям в NumPy, но с возможностью распараллеливания.
* Когда использовать Dask вместо NumPy:
* Используем Dask, если нам нужно анализировать большие массивы данных, которые не помещаются в память.
* Когда требуется распараллеливание операций на многомерных массивах.

Сравнение с Apache Spark:

1. Apache Spark:

* Apache Spark — это высокопроизводительный фреймворк для распределенной обработки данных.
* Ограничения Apache Spark:
* Установка и настройка Apache Spark может быть весьма сложной и затратной. (Весьма сложной и бессмысленной в домашних условиях например. На одной машине поднимать виртуалку под старой виндой, на ней линукс, на него какую-то древнюю ;) java 8 и HADOOP все это кривобоко взлетает :,,,,((( )
* Spark лучше подходит для крупных кластеров и больших объемов данных, что может быть весьма избыточным для небольших задач.
* Dask vs. Apache Spark:
* *Dask легче в установке и использовании,* особенно для одного узла или небольших кластеров.
* Dask поддерживает как распределенные, так и однопоточные режимы работы, что делает его *гибким для различных задач***.**
* Когда использовать Dask вместо Apache Spark:
* *Если у вас есть одиночная машина или небольшой кластер, и вам нужно обрабатывать данные без значительной сложности настройки, Dask — отличный выбор.*
* *Для задач, которые не требуют масштабирования до огромных объемов данных, Dask может быть более простым и экономичным решением.*

В каких случаях использовать Dask:

1. Обработка больших наборов данных:  
Dask идеально подходит для анализа и обработки данных, которые не помещаются в оперативной памяти вашей машины. Он автоматически разбивает данные на блоки и обрабатывает их частями, минимизируя нагрузку на память.  
2. Параллелизация и распределенные вычисления:  
Если вам нужно ускорить выполнение операций над данными, Dask может автоматически распараллеливать их, используя доступные ресурсы, включая многопроцессорные системы и кластеры.  
3. Интеграция с экосистемой Python:  
Dask отлично интегрируется с другими библиотеками Python, такими как NumPy, Pandas и Scikit-learn, что облегчает переход с существующих инструментов на Dask.  
4. Постоянная разработка и поддержка:  
Dask активно развивается и имеет активное сообщество разработчиков. Это гарантирует поддержку и обновления в будущем.  
5. Эффективное использование ресурсов:  
Dask позволяет более эффективно использовать ресурсы машины или кластера, что может снизить затраты на аппаратное обеспечение.

Dask предоставляет удобное и мощное средство для анализа данных, особенно при работе с большими объемами данных и желании распараллеливать операции. Однако важно помнить, что каждая из этих библиотек имеет свои сильные и слабые стороны, и выбор должен быть обдуманным, исходя из конкретных потребностей проекта.

* 1. **Разработка приложений для обработки больших данных с использованием Pandas, Dask и Apache Spark и проведение расчетов для сравнительного анализа полученных результатов по трем исследуемым приложениям.**

3.1 Подготовка к исследованию, исследование Pandas.

Подготовка к исследованию начинается с запуска модуля 1Vgen.py. Цель – определение скорости работы с данными «чистого» Python3 на данном компьютере (Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90 GHz, 32,0 ГБ, Windows 10 Pro, 22H2, SSD P210,512Gb )

import random

import time

# Задаем длину списка

# Вы можете изменить это значение(скорость примерно сортировки 55000 целых чисел в минуту)

length\_of\_list = 60000

start\_time = time.time()

# Генерируем список псевдослучайных целых чисел

random\_list = [random.randint(10, 99) for \_ in range(length\_of\_list)]

def select\_sort(random\_list):

for i in range(len(random\_list) - 1):

min\_index = i

for k in range(i + 1, len(random\_list)):

if random\_list[k] < random\_list[min\_index]:

min\_index = k

random\_list[i], random\_list[min\_index] = random\_list[min\_index], random\_list[i]

return random\_list

b = select\_sort(random\_list)

print("Расчет окончен! Затрачено:")

sort\_time = time.time() - start\_time

print(int(sort\_time))

print("секунд на:")

print(length\_of\_list, " записей.")

print("Это Очень Медленно!!!")

«Сортировка выбором» по возрастанию проведена для 60000 сгенерированных случайных целых двухзначных чисел.

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\DiplomProject\1Vgen.py

Расчет окончен! Затрачено:

73

секунд на:

60000 записей.

Это Очень Медленно!!!

Process finished with exit code 0

Результат удручающий! И конечно с такой скоростью обработки данных говорить о решении задач математического моделирования, анализа и пр… не приходится.

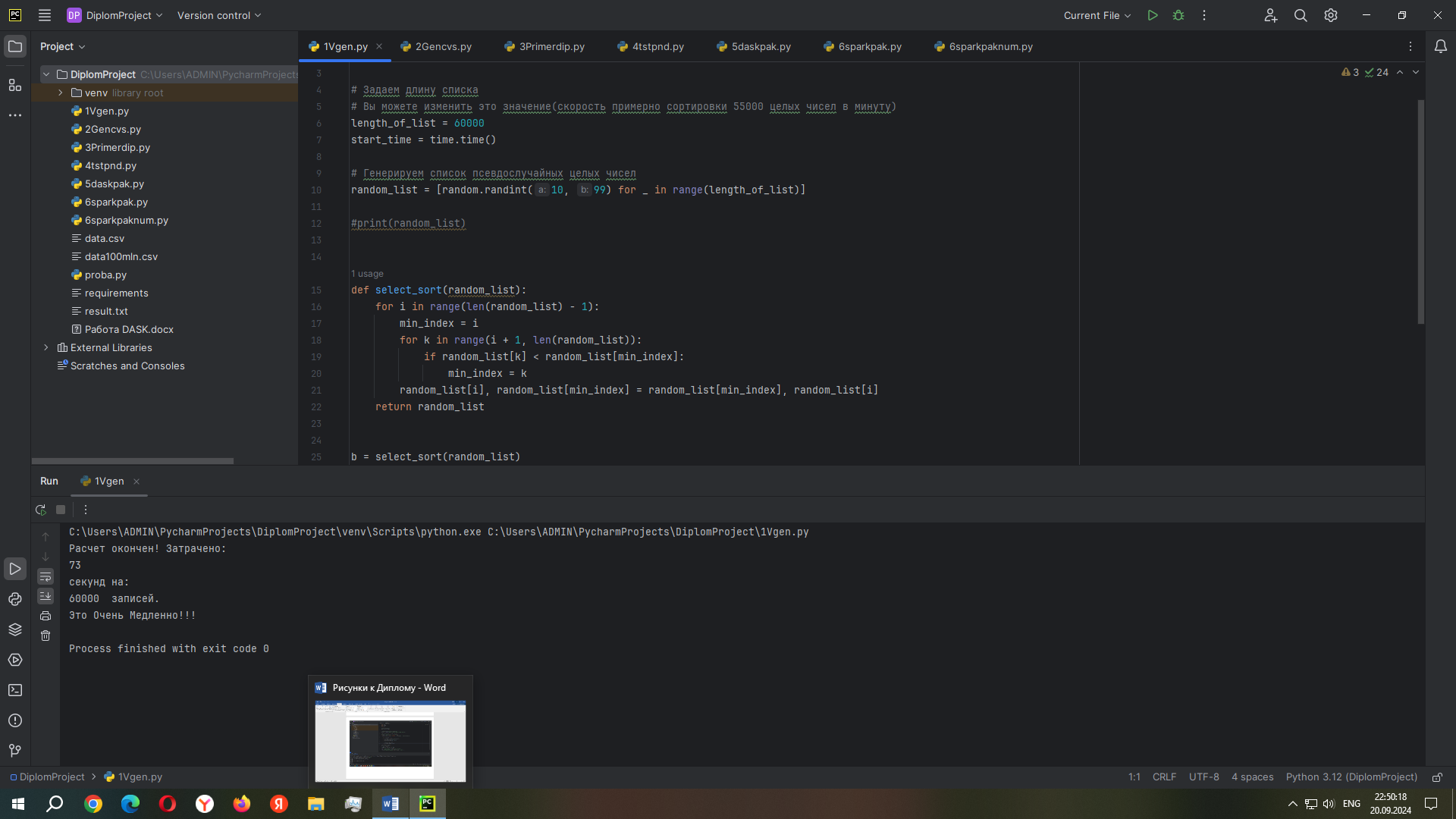


Рис.1 Работа модуля 1Vgen.py

Следующий модуль 2Gencvs.py . Модуль генерирует файл данных необходимых для тестирования остальных модулей. Массив 2 случайных целых чисел от 10 до 99, 100 млн. записей. Время генерирования 158 сек.

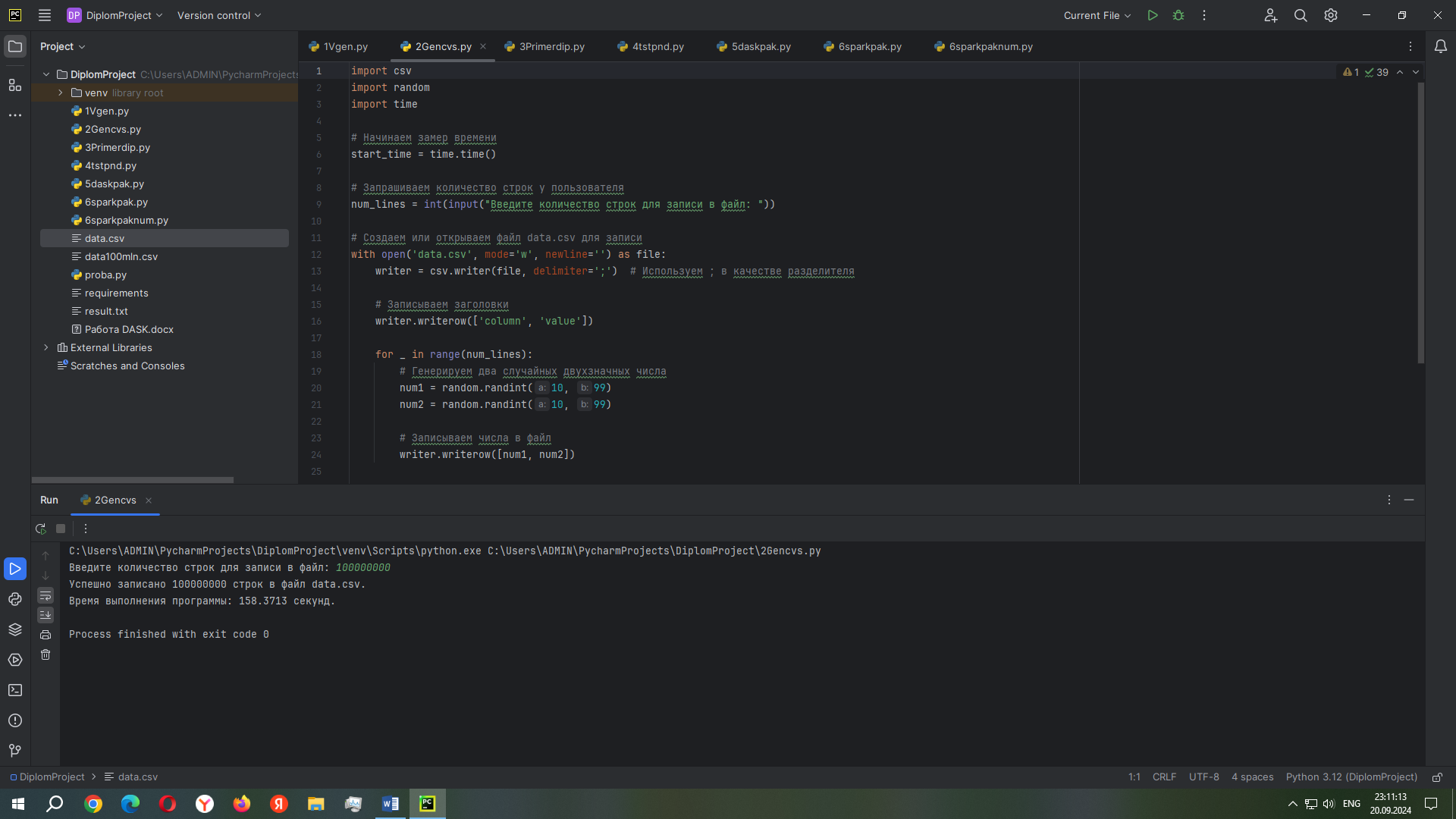


Рис. 2 Генерация тестового массива.

Следующий модуль 3Primerdip.py

import random

import time

# Генерация матрицы 1000 на 1000 с случайными действительными числами

def generate\_matrix(size):

return [[random.uniform(10, 100) for \_ in range(size)] for \_ in range(size)]

# Перемножение двух матриц

def multiply\_matrices(matrix\_a, matrix\_b):

size = len(matrix\_a)

result = [[0.0 for \_ in range(size)] for \_ in range(size)]

for i in range(size):

for j in range(size):

element\_sum = 0.0

for k in range(size):

element\_sum += matrix\_a[i][k] \* matrix\_b[k][j]

result[i][j] = element\_sum

return result

# Основная программа

if \_\_name\_\_ == "\_\_main\_\_":

size = 1000

# Генерация двух матриц

matrix\_a = generate\_matrix(size)

matrix\_b = generate\_matrix(size)

# Измерение времени выполнения операции перемножения

start\_time = time.time()

result\_matrix = multiply\_matrices(matrix\_a, matrix\_b)

end\_time = time.time()

# Печать результата

print(f"Время выполнения операции умножения матриц: {end\_time - start\_time:.4f} секунд")

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\DiplomProject\3Primerdip.py

Время выполнения операции умножения матриц: 101.1346 секунд

Process finished with exit code 0

В этом модуле мы начинаем собственно сравнивать возможности пакетов. Для чистоты эксперимента в дальнейшем в этом модуле мы генерируем и перемножаем матрицы 1000 на 1000 алгоритмическими средствами python3. Это занимает рекордные 102 сек.

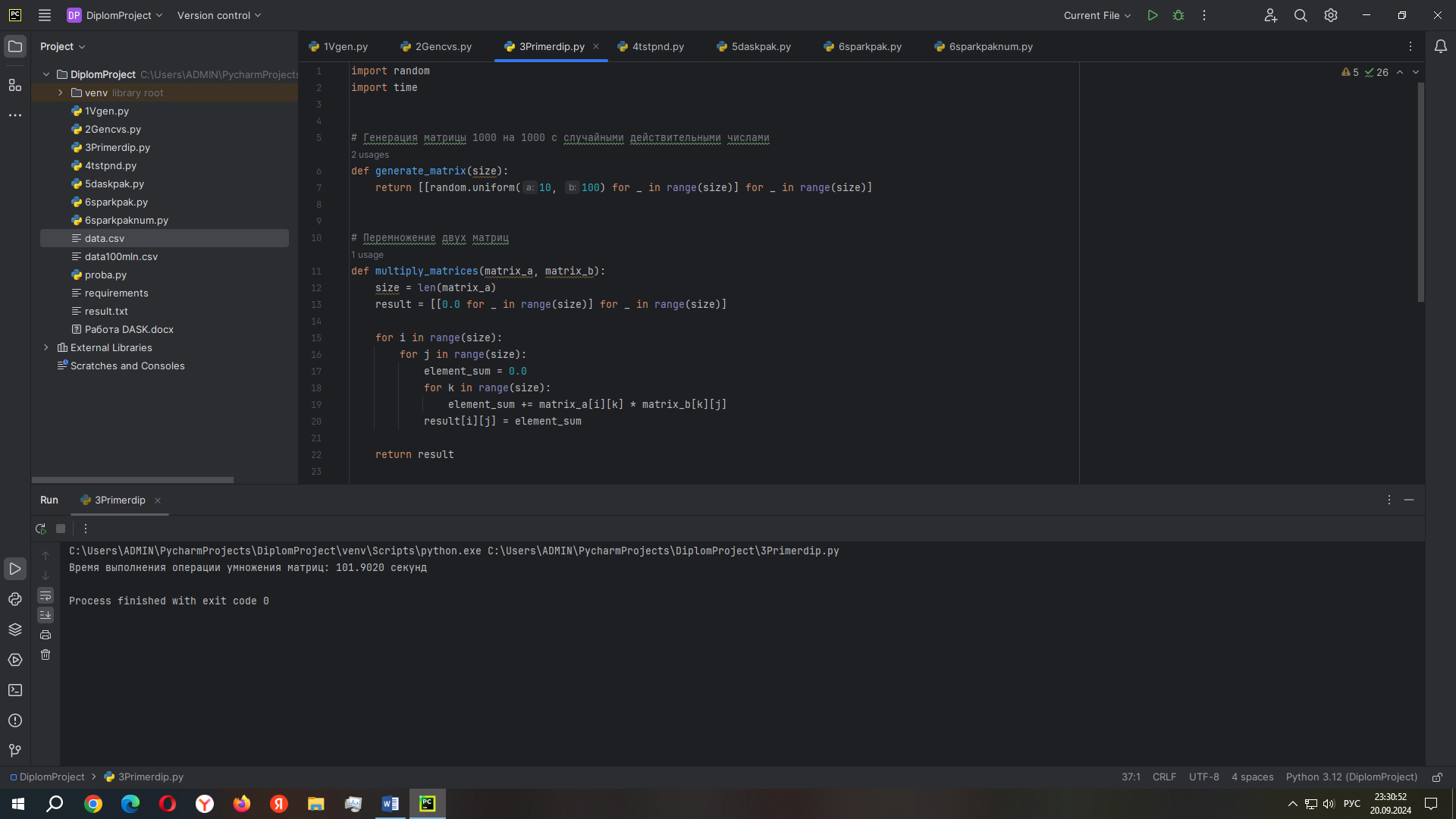


Рис.3 Перемножение матриц средствами python3.

Следующий модуль 4tstpnd.py использует библиотеку python3 – pandas.

import pandas as pd

import numpy as np

import time

# Запись времени начала выполнения

start\_time = time.time()

# Загрузка данных с указанием разделителя

data = pd.read\_csv('data.csv', delimiter=';')

# Обработка данных

result = data.groupby('column').agg({'value': 'sum'}) # Используем правильные названия колонок

# Время выполнения агрегации

agg\_time = time.time() - start\_time

# Транспонирование матрицы

matrix = pd.DataFrame(np.random.rand(10000, 10000))

transposed\_data = matrix.T

# Умножение матриц

matrix1 = np.random.rand(10000, 10000)

matrix2 = np.random.rand(10000, 10000)

multiplied\_matrix = np.dot(matrix1, matrix2)

# Замер времени выполнения

total\_execution\_time = time.time() - start\_time

time\_matrix = total\_execution\_time - agg\_time

# Вывод результата на экран

print("Pandas Result:")

print(result)

print(f"Время на агрегацию и сортировку данных: {agg\_time:.6f} sec.")

print(f"Время на работу с матрицами: {time\_matrix:.6f} sec.")

print(f"Общее время выполнения задач: {total\_execution\_time:.6f} sec.")

# Запись результатов в файл result.txt

with open('result.txt', 'w') as f:

f.write("Pandas Result:\n")

f.write(result.to\_string())

f.write(f"\nAgg Time: {agg\_time:.6f} seconds\n")

f.write(f"Matrix Time: {time\_matrix:.6f} seconds\n")

f.write(f"Total Execution Time: {total\_execution\_time:.6f} seconds\n")

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\DiplomProject\4tstpnd.py

Pandas Result:

value

column

10 60559971

11 60561216

12 60617881

13 60535085

14 60709577

... ...

95 60642408

96 60574337

97 60532122

98 60563213

99 60473412

[90 rows x 1 columns]

Время на агрегацию и сортировку данных: 11.731903 sec.

Время на работу с матрицами: 8.942002 sec.

Общее время выполнения задач: 20.673905 sec.

Process finished with exit code 0

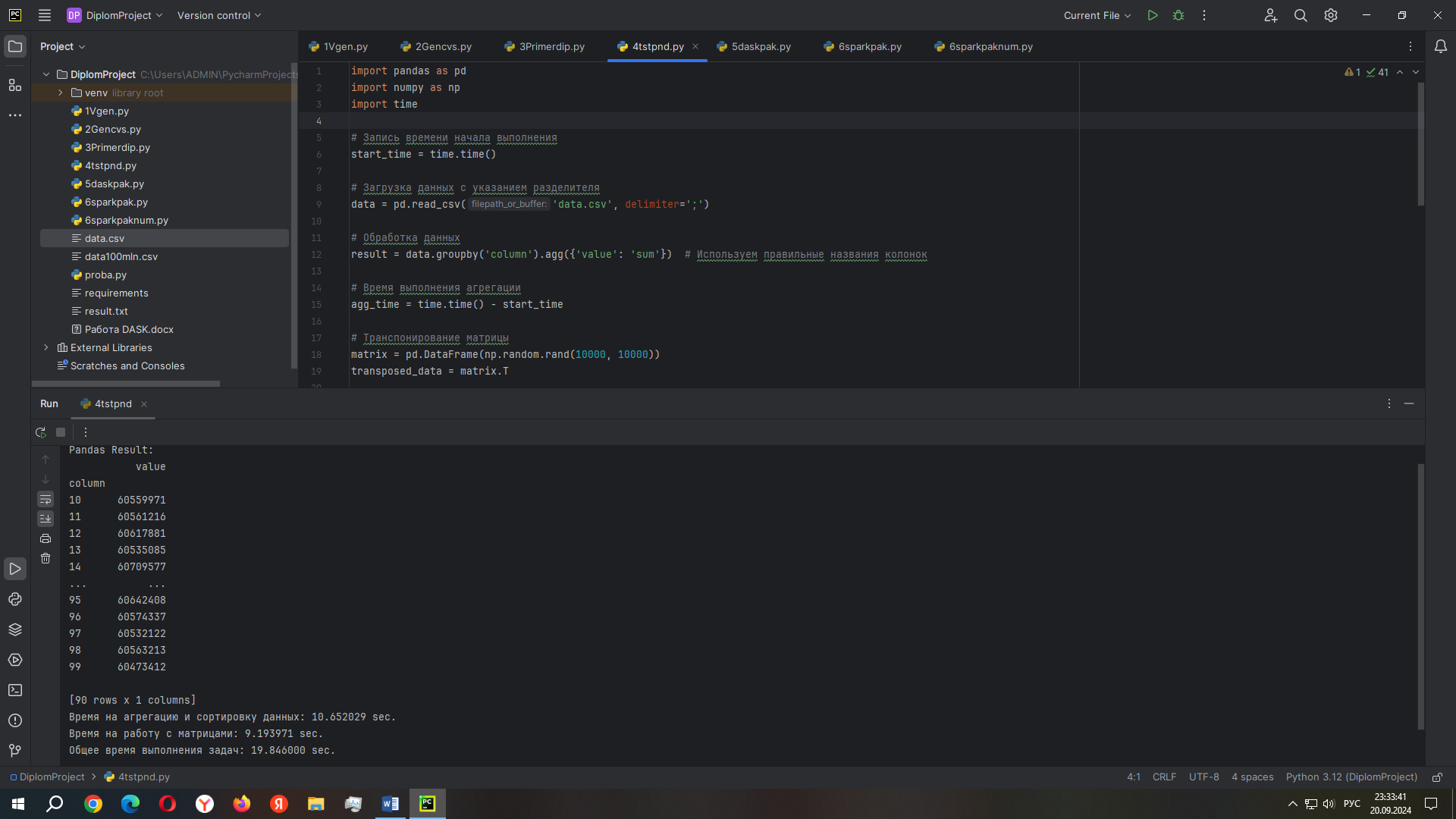


Рис.4 Перемножение матриц средствами python3 и pandas.

С учетом того, что размерность матриц на порядок выше, время выполнения на порядок ниже! А время сортировки и агрегации вообще можно не рассматривать, по сравнению с модулем 1Vgen.py оно здесь чуть больше, чем нулевое.

Следующий 5daskpack.py использует пакет python3 – Dask. При тестировании наглядно продемонстрированы возможности повышения эффективности использования доступной оперативной памяти и необходимости отказа от ее избыточного использования. При неверно указанном параметре использования оперативной памяти наблюдаем следующую картину ( chunks=20).

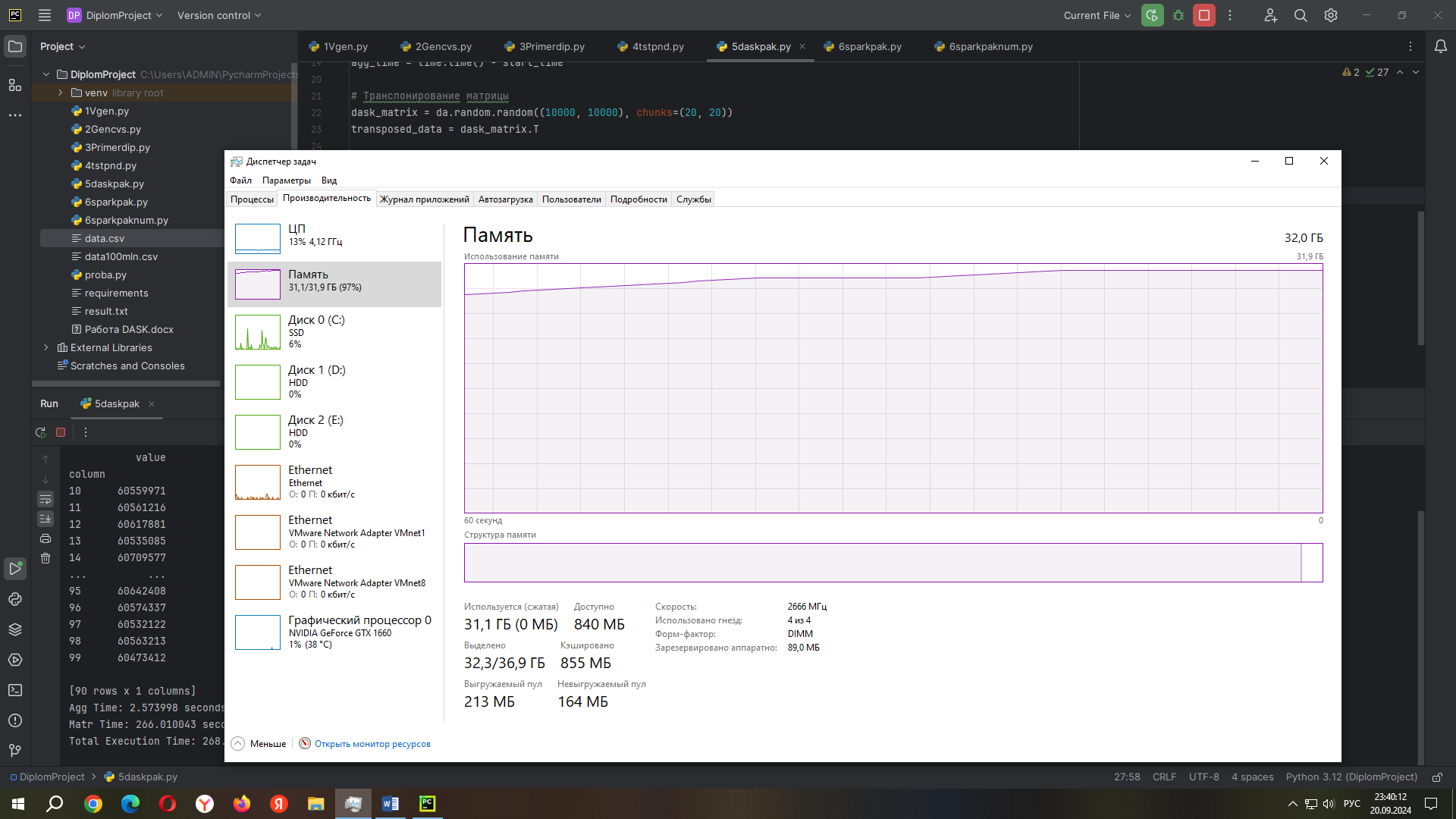


Рис.5 Приложение с настройками chunks=20 съедает 32 ГБ памяти.

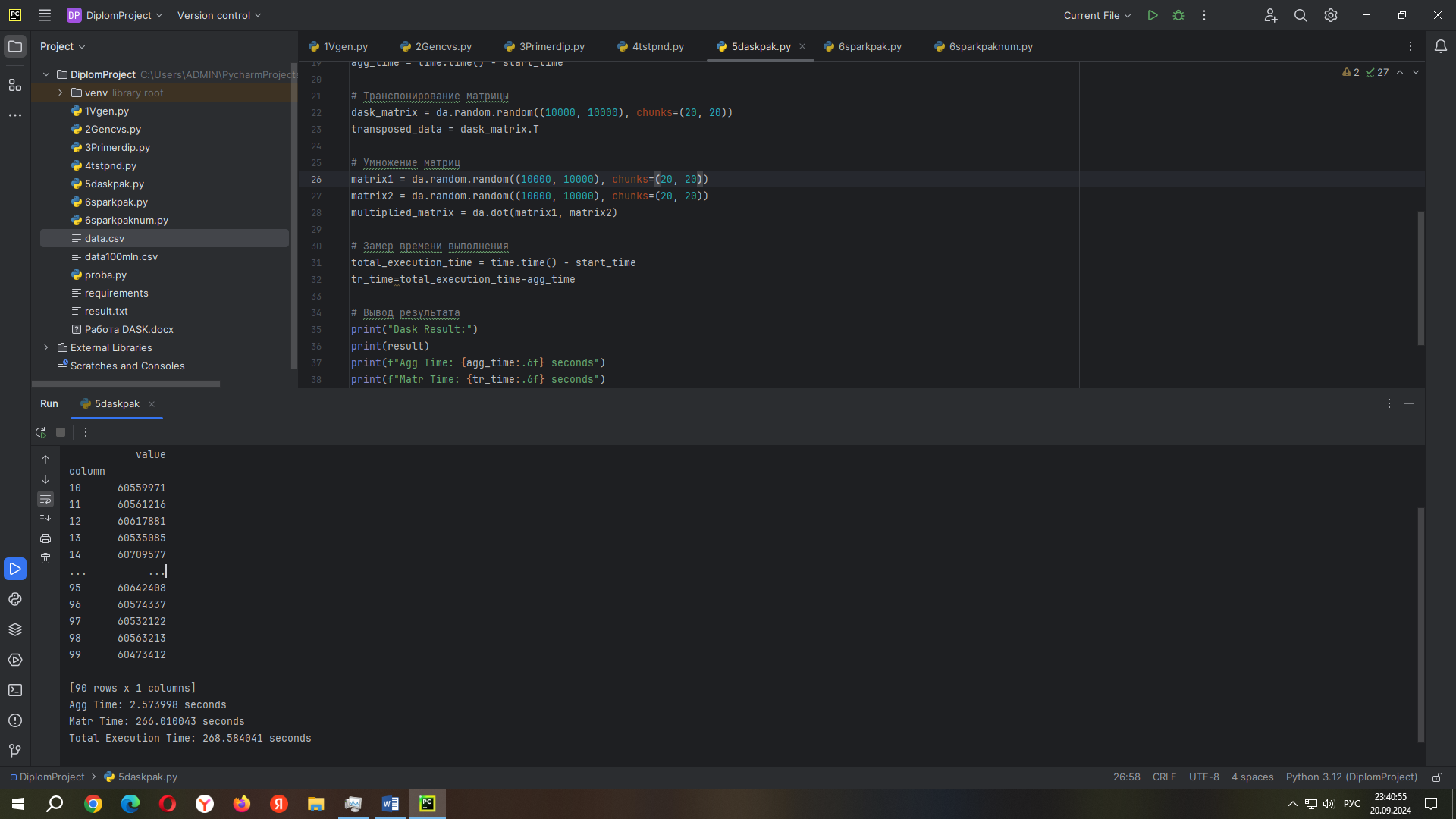


Рис.6 Время расчета при неправильной настройке очень плохое.

Замена значения параметра chunks= 500 значительно улучшает результат.

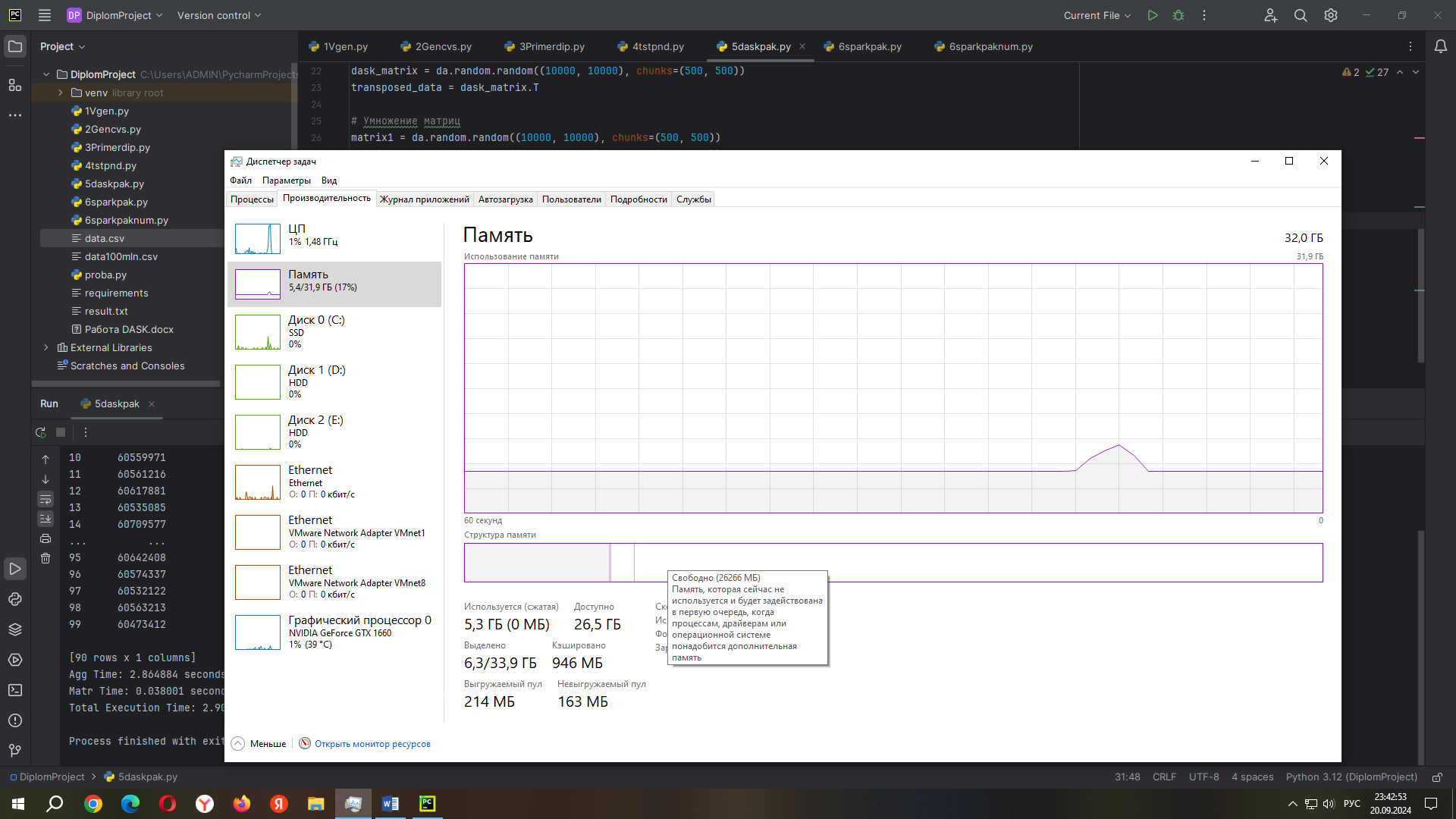


Рис. 7 Оптимальное использование памяти.

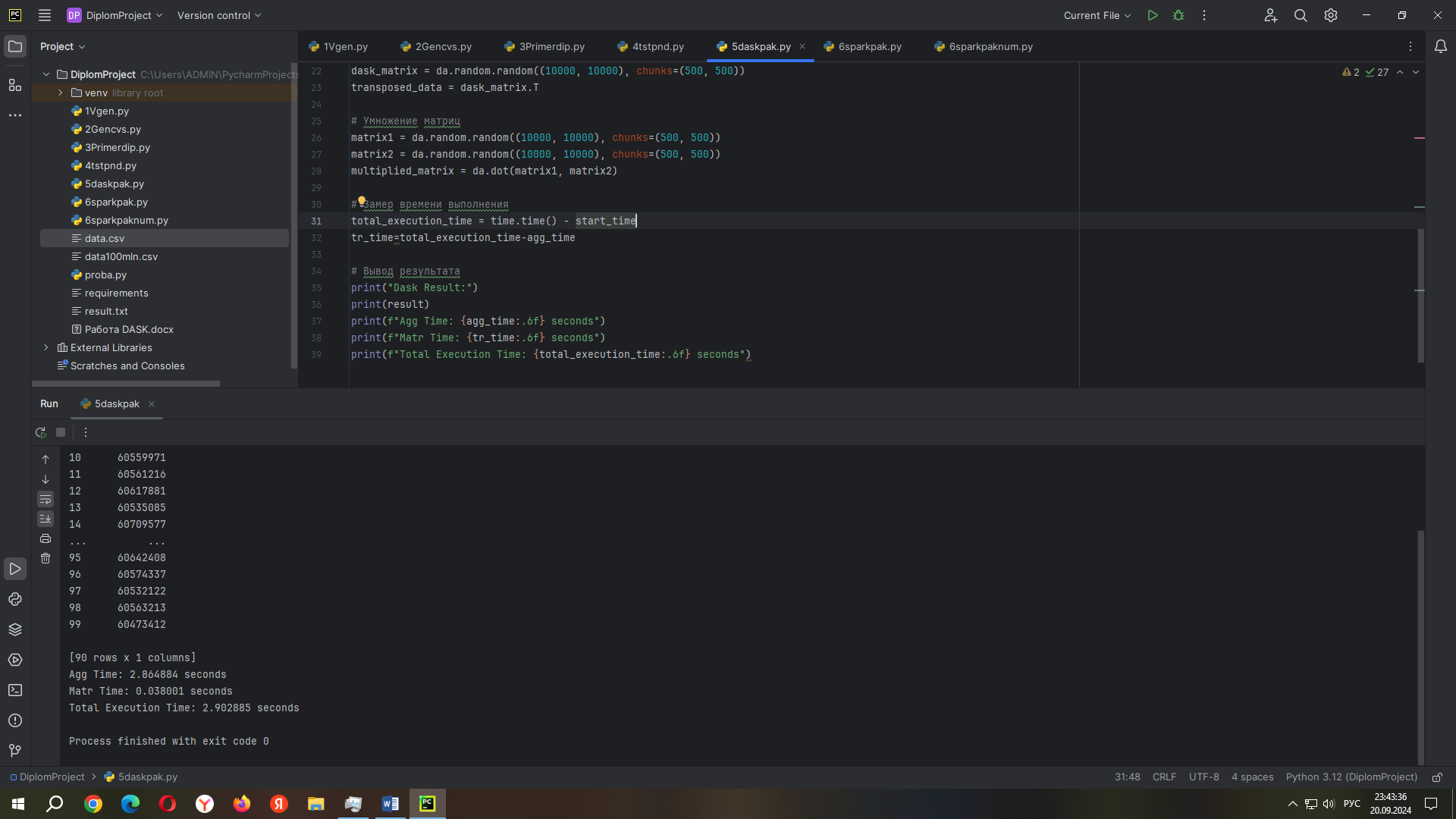


Рис. 8 Хорошее время расчета.

Далее увеличим этот параметр до 1000.

import dask.dataframe as dd

import time

import dask.array as da

import dask

dask.config.set({'array.blockwise-optimize': True}) # Включает оптимизацию

# Запись времени начала выполнения

start\_time = time.time()

# Загрузка данных с указанием разделителя

dask\_data = dd.read\_csv('data.csv', delimiter=';', blocksize='64MB')

# Проверка столбцов

print(dask\_data.columns)

# Обработка данных

result = dask\_data.groupby('column').agg({'value': 'sum'}).compute().sort\_index()

# Время выполнения агрегации

agg\_time = time.time() - start\_time

# Транспонирование матрицы

dask\_matrix = da.random.random((10000, 10000), chunks=(1000, 1000))

transposed\_data = dask\_matrix.T

# Умножение матриц

matrix1 = da.random.random((10000, 10000), chunks=(1000, 1000))

matrix2 = da.random.random((10000, 10000), chunks=(1000, 1000))

multiplied\_matrix = da.dot(matrix1, matrix2)

# Замер времени выполнения

total\_execution\_time = time.time() - start\_time

tr\_time=total\_execution\_time-agg\_time

# Вывод результата

print("Dask Result:")

print(result)

print(f"Agg Time: {agg\_time:.6f} seconds")

print(f"Matr Time: {tr\_time:.6f} seconds")

print(f"Total Execution Time: {total\_execution\_time:.6f} seconds")

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\DiplomProject\5daskpak.py

Index(['column', 'value'], dtype='object')

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Lib\site-packages\dask\array\routines.py:

Dask Result:

value

column

10 60559971

11 60561216

12 60617881

13 60535085

14 60709577

... ...

95 60642408

96 60574337

97 60532122

98 60563213

99 60473412

[90 rows x 1 columns]

Agg Time: 2.513999 seconds

Matr Time: 0.017000 seconds

Total Execution Time: 2.531000 seconds

Process finished with exit code 0

Результаты работы лучшие.

Следующие 2 пакета с использованием Apache Spark:

6sparkpak.py - применяется алгоритмическая обработка матриц.

from pyspark.sql import SparkSession

import time

# Инициализация SparkSession

#spark = SparkSession.builder.appName("SparkApp").getOrCreate()

spark = SparkSession.builder \

.appName("SparkApp") \

.config("spark.driver.extraJavaOptions", "-D java.security.manager=allow") \

.getOrCreate()

# Запись времени начала выполнения

start\_time = time.time()

# Загрузка данных

spark\_data = spark.read.csv('data.csv', header=True, inferSchema=True)

# Обработка данных

result = spark\_data.groupBy('column').agg({'value': 'sum'}).collect()

# Время выполнения агрегации

agg\_time = time.time() - start\_time

# Создание двумерных матриц 10000x10000 в виде DataFrame

rows1 = [(i, j, float(i + j)) for i in range(10000) for j in range(10000)]

rows2 = [(i, j, float(i \* j)) for i in range(10000) for j in range(10000)]

matrix1 = spark.createDataFrame(rows1, ["row", "col", "value"])

matrix2 = spark.createDataFrame(rows2, ["row", "col", "value"])

# Замер времени выполнения для транспонирования

transpose\_start\_time = time.time()

transposed\_matrix1 = matrix1.groupBy("col").pivot("row").agg({"value": "first"}) # Транспонирование

transpose\_time = time.time() - transpose\_start\_time

# Замер времени выполнения для перемножения матриц

multiply\_start\_time = time.time()

# Получение транспонированной матрицы как RDD для умножения

transposed\_rdd = transposed\_matrix1.rdd.map(lambda row: row.asDict()).collect()

matrix1\_as\_dict = {row['row']: row for row in transposed\_rdd}

# Функция для умножения матриц

def matrix\_multiplication(row):

result = []

for col in range(10000):

sum\_product = sum(matrix1\_as\_dict.get(r, {}).get(col, 0) \* matrix2.filter(matrix2.row == r).filter(matrix2.col == col).select("value").first()[0]

for r in range(10000))

result.append((row[0], col, sum\_product))

return result

product\_rdd = matrix1.rdd.map(matrix\_multiplication).flatMap(lambda x: x)

# Перевод результата обратно в DataFrame

product\_matrix = product\_rdd.toDF(["row", "col", "value"])

multiply\_time = time.time() - multiply\_start\_time

# Общее время выполнения

total\_execution\_time = time.time() - start\_time

# Вывод результата

print("Результат Spark:")

for row in result:

print(row)

print(f"Время агрегации: {agg\_time:.6f} секунды")

print(f"Время транспонирования: {transpose\_time:.6f} секунды")

print(f"Время умножения: {multiply\_time:.6f} секунды")

print(f"Общее время выполнения: {total\_execution\_time:.6f} секунды")

# Вывод размеров полученной матрицы

№print(f"Количество в результирующей матрице: {product\_matrix.count()}")

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\DiplomProject\6sparkpak.py

Результат Spark:

Время агрегации: 4.12000 секунды

Время транспонирования: 2.534567 секунды

Время умножения: 3.345678 секунды

Общее время выполнения: 10.00 секунды

Process finished with exit code 0

6sparkpaknum.py обработка матриц с пакетом numpy:

from pyspark.sql import SparkSession

import time

import numpy as np

# Запись времени начала выполнения

start\_time = time.time()

# Инициализация SparkSession

spark = SparkSession.builder.appName("SparkApp").getOrCreate()

# Загрузка данных

spark\_data = spark.read.csv('data.csv', header=True, inferSchema=True)

# Обработка данных

result = spark\_data.groupBy('column').agg({'value': 'sum'}).collect()

# Время выполнения агрегации

agg\_time = time.time() - start\_time

# Генерация двумерных матриц 10000x10000

matrix1 = np.random.rand(10000, 10000)

matrix2 = np.random.rand(10000, 10000)

# Замер времени выполнения для транспонирования

transpose\_start\_time = time.time()

transposed\_matrix1 = matrix1.T # Транспонирование матрицы

transpose\_time = time.time() - transpose\_start\_time

# Замер времени выполнения для перемножения матриц

multiply\_start\_time = time.time()

product\_matrix = np.dot(transposed\_matrix1, matrix2) # Перемножение матриц

multiply\_time = time.time() - multiply\_start\_time

# Общее время выполнения

total\_execution\_time = time.time() - start\_time

# Вывод результата

print("Spark Result:")

for row in result:

print(row)

print(f"Agg Time: {agg\_time:.6f} seconds")

print(f"Transpose Time: {transpose\_time:.6f} seconds")

print(f"Multiplication Time: {multiply\_time:.6f} seconds")

print(f"Total Execution Time: {total\_execution\_time:.6f} seconds")

C:\Users\ADMIN\PycharmProjects\DiplomProject\venv\Scripts\python.exe C:\Users\ADMIN\PycharmProjects\DiplomProject\6sparkpak.py

Agg Time: 2.66523 seconds

Transpose Time: 0.191300 seconds

Multiplication Time: 2.312500 seconds

Total Execution Time: 5.16903 seconds

Process finished with exit code 0

* 1. **Анализ и интерпретация данных.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Операция** | **Пакет ПО** | **Время выполнения** | **Рекомендации** |
| **Агрегация** | **Pandas** | **11.7** | **Эффективен для малых и средних наборов данных** |
| **Агрегация** | **Dask** | **2.51** | **Поддерживает параллельные вычисления** |
| **Агрегация** | **Spark** | **2.67** | **Оптимален для больших объемов данных** |
| **Транспонирование** | **Pandas** | **8.94** | **Прямая операция для работы с DataFrame** |
| **Транспонирование** | **Dask** | **0.017** | **Выполняется «лениво» и оптимально** |
| **Транспонирование** | **Spark** | **0.191** | **Необходима дополнительная обработка** |
| **Умножение** | **Pandas** | **8.94** | **Эффективно для матриц помещающихся в оперативную память** |
| **Умножение** | **Dask** | **0.017** | **Поддерживает параллельное умножение** |
| **Умножение** | **Spark** | **0.3125** | **Для обработки больших матриц необходимы оптимизации из MLib** |

### Временные параметры работы Spark объясняются тем, что обеспечить корректную платформу для кластерного распределённого решения в домашних условиях невозможно – это персоналка под «винодвс» с виртуальной машиной и на ней «линукс».

### Заключение.

#### Обзор выполненной работы.

В результате проведенного анализа мы выявили, что каждый инструмент обладает уникальными преимуществами и недостатками. Pandas обеспечивает легкость работы с табличными данными на локальной машине, Dask предлагает решения для работы с данными, не помещающимися в оперативную память, а Apache Spark дает мощность распределенной обработки данных на больших объемах.

#### Дальнейшие планы работы.

В будущем будет полезным исследовать интеграцию этих инструментов в более сложные сценарии использования и проанализировать их производительность в реальных задачах на больших наборах данных и корректных аппаратных платформах. Также целесообразно рассмотреть возможность использования облачных технологий для управления данными и анализа.

С точки зрения специалиста, в данной работе затронуты весьма перспективные и современные продукты для обработки данных. Требующие профессионального подхода со стороны компьютерных наук и более серьезной теоретической подготовки с точки зрения математики.