Дипломная работа по теме: «**Анализ и сравнение различных способов обработки и хранения больших данных: Pandas, Dask и Apache Spark**»

Автор: Орлов Борис Николаевич

Оглавление

1. Введение

1.1. Обоснование выбора темы

«Кто владеет информацией, то владеет миром».

В нашем современном мире информация становиться одним из главного ресурса, при помощи которого различные структуры стремятся достичь своих целей.

Государство – для контроля за своими гражданами. Оценка настроений общества, его реакция на те или иные изменения во внутренней политике. Прогноз поведения общества при этом. Контроль за исполнением законодательства. Для определения направлений государственной политики в образовании, медицине и т.п.

Корпорации – для достижения максимальной прибыли. Здесь огромный спектр требуемой информации. Что будет потреблять человечество, его востребовательность в материальном и духовном.

Структуры корпораций – производства, торговые сети, строительные компании и т.п. Анализ работы оборудования и технологии, объёмы продаж в зависимости от различных факторов, потребность в типе и качестве жилья. Реакция людей на рекламные акции.

Для достижения вышеперечисленных целей необходимо получить, обработать и правильно интерпретировать информацию, что позволит определить направление действий той или иной структуры.

Востребованность в обработке поступающей информации, при увеличении количества анализируемой информации будет возрастать. Что и обусловило выбор данной темы работы

1.2. Определение цели и задач исследования

1.2.1. Задача данной работы состоит в выполнении обработки большого объёма данных (Big Data) при помощи фреймворков (**framework**),

библиотек: **Pandas, Dask и Apache Spark,** с целью определения их сильных сторон, эффективность применения для различных случаев.

1.2.2. Выполнить практическое тестирование фреймворков и библиотек, применив их возможности при обработке данных**.**

2. Основные понятия, определения и инструменты

Большие данные (в английском языке «Big Data») — это термин, который используется для описания колоссальных объемов данных, которые невозможно эффективно обработать с использованием традиционных методов. То есть с ними не справится ни обычный человек, ни простой пользовательский компьютер. Для обработки больших данных применяют специальные технологии и программное обеспечение. При этом огромные объемы информации можно использовать для решения задач, требующих высокой точности прогнозов, поиска обоснований для тех или иных решений, персонализации сервисов и так далее.

Для характеристики **Big Data** традиционно используются три основных аспекта, которые называются «тройкой больших данных» или «3V» (в английском языке все три термина начинаются с латинской буквы V):

«Объем» (Volume):

Большие данные означают огромные объемы информации. Это включает в себя терабайты, петабайты и даже эксабайты данных.

«Разнообразие» (Variety):

Big Data может иметь различные форматы, включая текст, изображения, видео, аудио и структурированные данные, такие как таблицы и базы данных. Разнообразие информации также включает в себя данные в реальном времени и данные с географическими координатами.

«Скорость» (Velocity):

Скорость обработки и анализа данных в реальном времени является ключевым аспектом **Big Data**. Информация может поступать со скоростью нескольких тысяч транзакций в секунду.

Кроме того, со временем к тройке больших данных стали добавлять еще два признака:

«Истинность» (Veracity)

Это качество данных, включая точность, надежность и актуальность.

«Ценность» (Value)

Это способность извлекать ценную информацию и знания из **больших данных** и использовать их в бизнесе или исследованиях.

Обработка **больших данных** включает в себя использование специализированных технологий и инструментов.

Фреймворк (англ. **framework** — «каркас, структура») — готовый набор инструментов, который помогает разработчику быстро создать продукт.

Библиотека *(англ.* **library***)* — это набор готовых функций, классов и объектов для решения каких-то задач в программировании.

Apache Spark - это высокопроизводительный фреймворк для параллельной обработки данных, который предоставляет API на Java, Scala, **Python** и R. Spark поддерживает обработку данных в реальном времени и в памяти.

Особенности фреймворка:

- обработка в памяти (in-memory processing);
- resilient distributed datasets (RDD)

Свойства RDD:

- неизменяемость и секционирование;
- применение общих операций, которые способны манипулировать всеми данными одновременно;
- отказоустойчивость;
- ленивые вычисления;
- сохраняемость.
- параллельная обработка и комбинирование операций.

Dask — это библиотека для параллельных и распределённых вычислений в Python, предназначенная для работы с большими

объёмами данных. Она разработана с учётом того, чтобы предоставить инструменты для высокоуровневого управления вычислениями, которые могут быть выполнены параллельно или распределённо на нескольких вычислительных узлах. **Основная цель Dask — 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.

Pandas — это библиотека в **Python**, которая предназначена для анализа уже структурированных данных.

Она содержит набор заранее подготовленных методов и функций, которые позволяют не изобретать велосипед каждый раз, а использовать уже созданные и протестированные алгоритмы.

С помощью **Pandas** можно:

- группировать данные по определённым признакам;
- создавать сводные таблицы из нескольких;
- очищать данные от дубликатов и невалидных строк или столбцов;
- выводить определённые значения по фильтрам или уникальности;
- использовать агрегирующие функции, например, подсчёт значений, сумму элементов, среднее значение;
- визуализировать собранные данные.

3. Методика выполнения работы

Для выполнения работы предполагается использовать базы данных, полученные из открытых источников. Будут применены базы данных одного вида, но с разным количеством данных. Будет использовано 5 (пять) .csv-фалов, объёмом: 5 кВ, 181 кВ, 18 МВ, 53МВ, 197 МВ соответственно. К сожалению, .csv-файлы большего размера в свободном доступе найти не смог.

В процессе выполнения работы будут применены методы библиотек **Pandas, Dask** и фреймворка **Apache Spark** и определена скорость выполнения расчётов. Для выполнения расчётов каждой из библиотек будет создан отдельный модуль. Также будет создан отдельный модуль для определения скорости выполнения вычислений. Для визуализации полученных данных будет построен график зависимости скорости выполнения расчёта от объёма базы данных (.csv-файла). Для этой цели будет установлена и применена библиотека **matplotlib**.

Работа выполняется на ноутбуке со следующими характеристиками:
Объём оперативной памяти – 8 ГБ, процессор – Intel Pentium Silver N5000 1.10 GHz.

Среда разработки: Jet Brains PyCharm

4. Выполнение работы

4.1. Создал проект diplomWork

4.2. В проект установил библиотеки **pandas, dask,** framework **Spark**, **matplotlib**.

4.3. На своей странице в GitHub создал репозиторий и привязал к нему созданный проект. В проекте создал модули, в которых будут написана функции для выполнения работы. Скриншот проекта представлен на Рисунке 1.

4.4. В корневой каталог проекта перенёс ранее найденные 5 (пять) .csv-файлов.

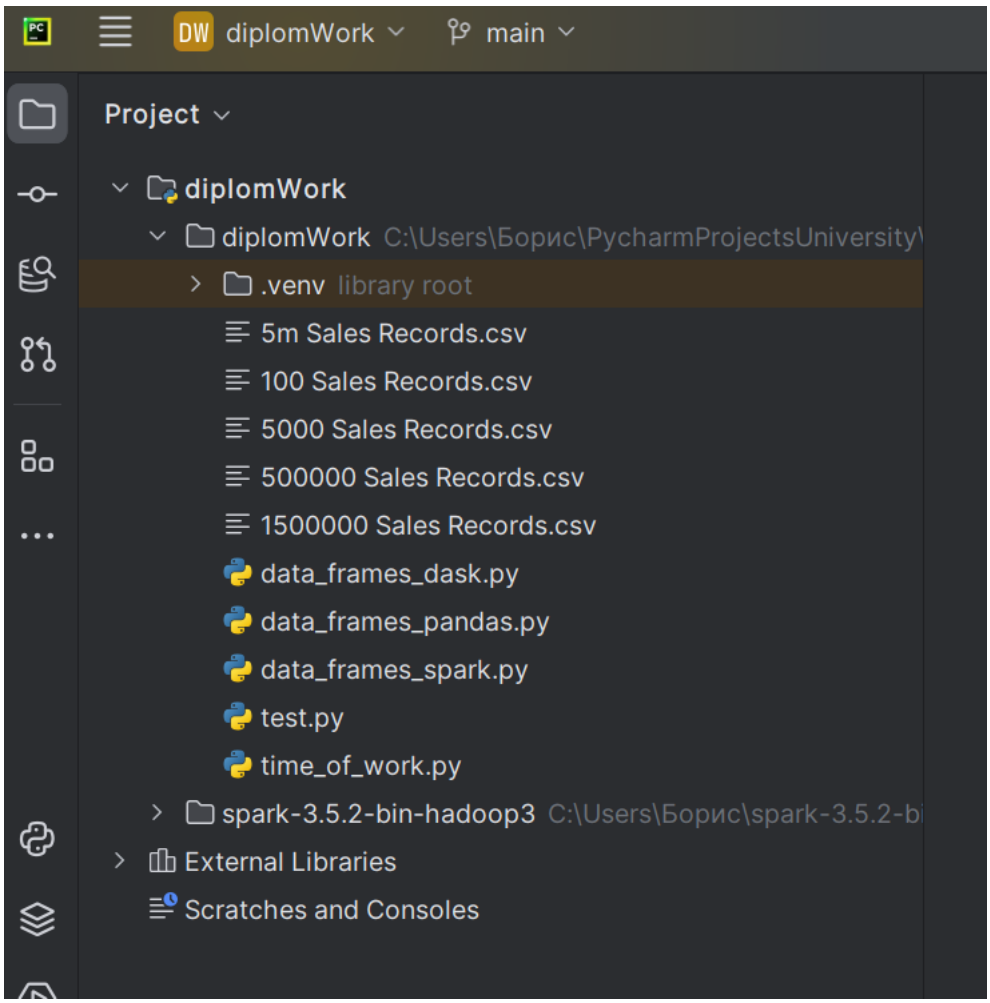

Рисунок 1. Скриншот проекта.

4.5. В созданных модулях (**data_frame_dask**, **data_frame_pandas**, **data_frame_spark**) написал функции, выполняющие идентичные расчёты для каждой из библиотек. Пример функций для работы с библиотекой **Dask** представлен на Рисунке 2.

data_frames_dask.py

```python
import dask.dataframe as dd
import time


# Создаю объект Data Frame, удаляю дубликаты из таблицы и пустые ячейки заменяю на 0
5 usages  Krokus
def create_df():
    df = dd.read_csv( path: '5000 Sales Records.csv', encoding='UTF-8')
    df = df.drop_duplicates()
    df = df.fillna(0)
    return df


# Работа с объектом Data Frame:

# 1. Фильтрация данных по столбцу
 Krokus
def filter_column():
    filter_df = create_df()
    filter_df = filter_df[filter_df['Region'] == 'Europe']
    return filter_df


# 2. Выполнение расчётов
```

data_frames_dask.py

```python


# 2. Выполнение расчётов
 Krokus
def average_value():
    df_value = create_df()
    value = df_value['Unit Cost'].mean()  # Вычисление среднего значения по столбцу,
    # но вычисления не выполняются. Создаётся граф задач, который является последовательны
    # вычислительных шагов.
    result = value.compute()  # Выполняется вычисление. Dask реализует их параллельно или
    # распределёно, в зависимости от конфигурации
    return result


# Группировка данных по одному столбцу('Region') и вычисление суммы для сформированных
# групп по другому столбцу ('Total Profit')
 Krokus
def sum_for_group():
    df = create_df()
    sum_group = df.groupby('Region')['Total Profit'].sum()
    return sum_group


# Сортировка по столбцу в порядке убывания
```

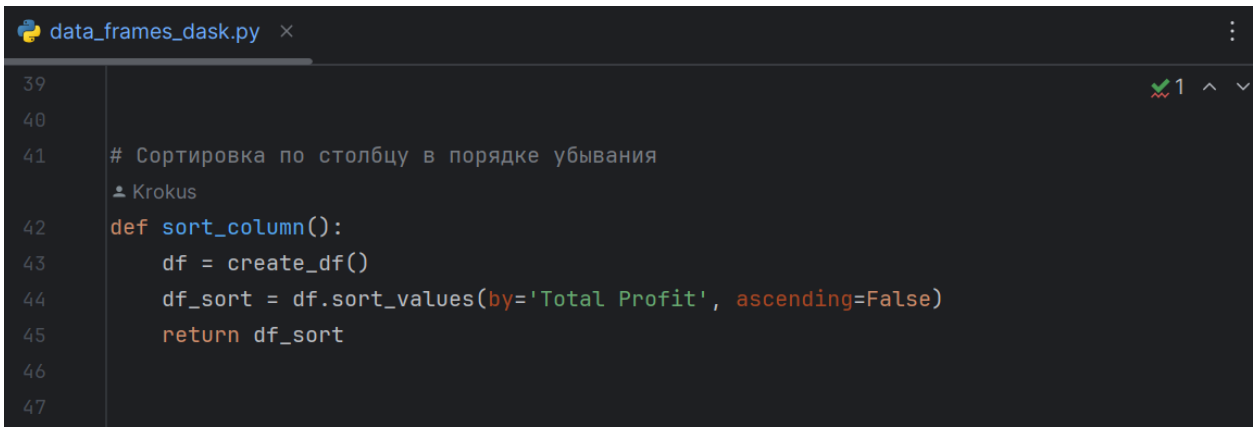

Рисунок 2. Применение методов библиотеки **Dask** для работы с .csv-файлом.

4.6. Для определения времени выполнения операций, в модуле **time_of_work**, написал функцию, рассчитывающую время суммирования чисел в формате **float**, по заданному столбцу (название столбца – «**Total Cost**». Скриншот функции на Рисунке 3.

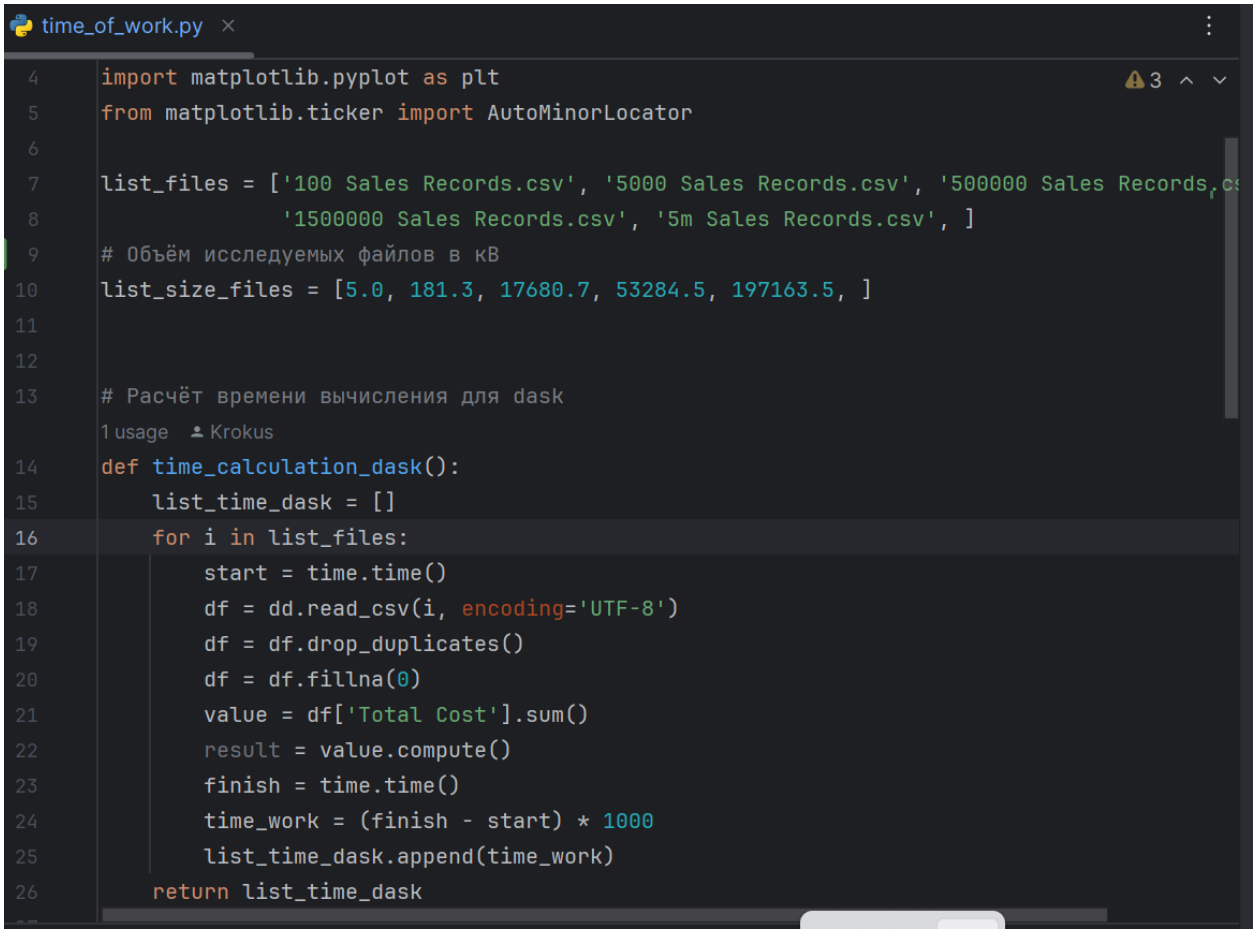

Рисунок 3. Функция, определяющая время выполнения расчёта.

4.7. Для визуализации полученных данных, в функции **time_of_work** написал код для создания графика зависимости времени расчёта от объёма файла. Код представлен на Рисунке 4.

```
# Построение графика зависимости скорости вычисления от объёма файла
x = list_size_files
y1 = time_calculation_pandas()
y2 = time_calculation_dask()
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_title( label: "Графики зависимостей: y1 - pandas, y2 - dask", fontsize=16)
ax.set_xlabel("Объём файла, кВ", fontsize=14)
ax.set_ylabel("Время вычисления, ms", fontsize=14)
ax.grid(which="major", linewidth=1.2)
ax.grid(which="minor", linestyle="--", color="gray", linewidth=0.5)
ax.plot( *args: x, y1, c="red", label="pandas")
ax.plot( *args: x, y2, label="dask")
ax.legend()
ax.xaxis.set_minor_locator(AutoMinorLocator())
ax.yaxis.set_minor_locator(AutoMinorLocator())
ax.tick_params(which='major', length=10, width=2)
ax.tick_params(which='minor', length=5, width=1)
plt.show()
```

Рисунок 4. Код для графического изображения полученных результатов.

4.8. В результате выполнения модуля **time_of_work**, был получен график зависимости, Рисунок 5.

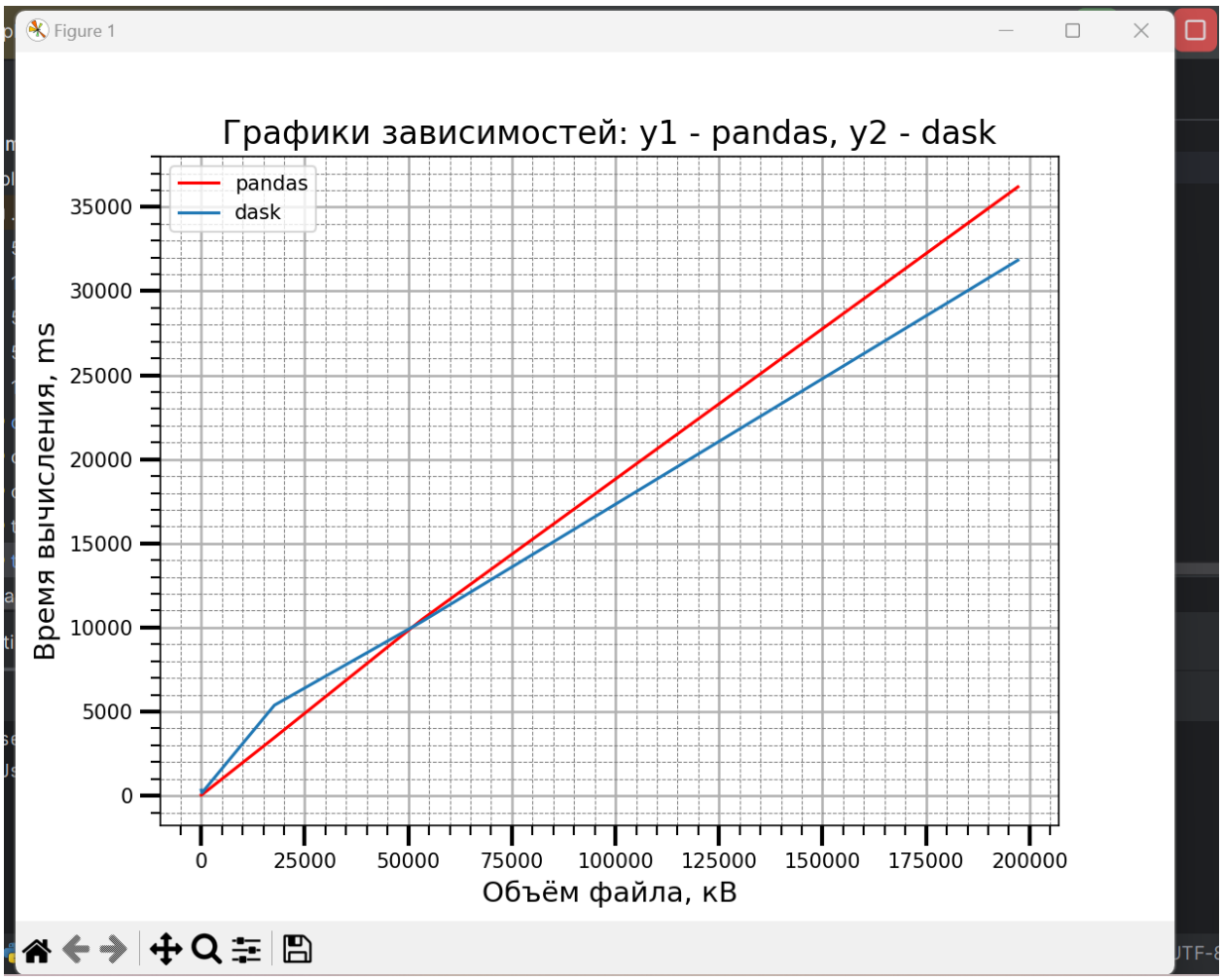

Рисунок 5. График зависимости времени выполнения расчёта от объёма обрабатываемого файла.

5. Оценка полученных результатов.

Каждая из используемых библиотек выполняет требуемые вычисления при работе с .csv-файлами в среде PyCharm на персональном ноутбуке.

Ввиду отсутствия базы данных большого объёма (от 2 ГБайт и более) не удалось на практике удостовериться, что библиотека **pandas** неспособна выполнять вычисление, когда объём файла превышает 25% от оперативной памяти компьютера, на котором выполняются вычисления (будет описано ниже в процессе обзора литературы по применению и сравнению библиотек).

Анализируя результаты скорости выполнения однотипных расчётов для библиотек **pandas** и **dask** на примере .csv-файлов различного объёма (от 5 кБайт до 200 Мбайт), подтверждаются утверждения, описанные в литературе:

для данных небольшого объёма (до 70 Мбайт) правильно будет использовать библиотеку **pandas**. Для данных большего объёма – библиотеки **dask** и **spark**.

На графике, Рисунок 5, наглядно видно, что при выполнении расчёта, начиная с объёма файла 60 Мбайт, время выполнения расчёта в библиотеке **pandas** увеличивается и разность во времени выполнения растёт по мере роста объёма обрабатываемой информации.

6. Сравнение библиотек **pandas**, **dask** и **spark**.

6.1. Библиотека **Pandas**.

В библиотеке **Pandas** определены два класса объектов для работы с данными:

Series - одномерный массив, который может хранить значения любого типа данных;

Data Frame двумерный массив (таблица), в котором столбцами являются объекты класса **Series**.

Для работы с этой библиотекой, её необходимо установить (***pip install pandas)*** и импортировать: ***import pandas as pd***.

Объект класса **Series** по сути является частным случаем объекта **DataFrame**. Создать объект **Data Frame** в **Python** проще из словаря (*имя объекта = pd.DataFrame (имя словаря)).* Базы данных как правило хранят в табличных файлах различных форматов. **Pandas** поддерживает операции чтения и записи для CSV, Excel 2007+, SQL, HTML, JSON и др. Создание объекта **DataFrame** из файлов различных форматов, используются следующие функции:

- *read_csv(file)*
- *read_excel(file)*
- *read_json(file)*

где file – строка, в которой прописан путь к дата сет.

После создания объекта **Data Frame** из дата сет, можно приступать к работе с полученной базой. Библиотека **Pandas** располагает большим набором методов для извлечения, группировки информации из дата сет, с целью

последующего анализа данных, выполнения математических операций. Позволяет исключать дублирование данных, удалять ячейки с отсутствующими данными. На основании полученных данных с помощью библиотеки **Pandas** возможно построение графиков зависимостей, отображение результатов на гистограммах распределения.

6.2. Библиотека **Dask**

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 распределенно 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 распределенно.

Установка библиотеки: ***pip install dask***.

Dask предоставляет три основных компонента для работы с данными:

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.

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.

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.

Чтение данных из .csv-файла аналогично тому, как это выполняется в библиотеке **Pandas**: (предварительно импортируем ***import dask.dataframe as dd) df = dd.read_csv('путь к файлу. csv')***

Dask поддерживает чтение данных из различных источников, включая базы данных, Hadoop Distributed File System (HDFS), и даже HTTP-сервисы. Это особенно полезно, если ваши данные хранятся в удаленных источниках.

Аналогично **Pandas**, **Dask** выполняет удаление дубликатов в базе, обработку отсутствующих значений. Изменение типа данных (например, ***int*** на ***floa***t). Фильтрация данных, их группировка. Сортировка данных.

Одним из ключевых преимуществ **Dask** является его способность работать с большими объемами данных, которые не помещаются в оперативной памяти одного компьютера. Это достигается за счет разделения данных на блоки и выполнения вычислений параллельно.

Параллельные и распределенные вычисления являются ключевыми аспектами анализа данных при работе с большими объемами информации. **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.

Dask также поддерживает параллельные вычисления с **Dask** DataFrames. При выполнении операций на **Dask** DataFrames, **Dask** разбивает данные на блоки по столбцам и строкам и выполняет операции параллельно. Важно отметить, что **Dask** обеспечивает прозрачное выполнение параллельных вычислений без необходимости явно управлять потоками или процессами.

Одним из сильных преимуществ **Dask** является его способность масштабирования для работы с большими вычислительными кластерами. Для начала работы с распределенными вычислениями с **Dask**, необходимо создать вычислительный кластер. Это может быть локальный кластер, который использует все доступные ядра на вашей машине, или удаленный кластер на нескольких машинах. Важно иметь в виду, что для удаленного кластера требуется настройка доступной инфраструктуры, такой как **Kubernetes** или **Apache Mesos**.

Dask можно использовать для распределенных вычислений при обучении моделей машинного обучения, особенно когда есть большие наборы данных.

Dask также предоставляет инструменты для визуализации данных, что позволяет легко создавать графики и диаграммы для анализа данных.

6.3. Фреймворк **Apache Spark**

Apache Spark — это фреймворк для обработки и анализа больших объёмов информации, входящий в инфраструктуру Hadoop. Он позволяет быстро выполнять операции с данными в вычислительных кластерах и поддерживает такие языки программирования, как Scala, Java, **Python**, R и SQL. **Spark** ускоряет обработку больших данных, распределяя вычисления между сотнями и тысячами машин, объединённых в кластеры.

Spark можно запустить в локальном режиме, без кластера. **Spark** работает на JVM. Поэтому для запуска заданий и разработки приложений на компьютере должен быть установлен JDK, путь к *java* должен находиться в переменной PATH, и должна быть установлена переменная JAVA_HOME.

Apache Spark состоит из нескольких модулей:

- **Spark Core;**
- **Spark SQL;**
- **Spark Streaming;**
- **MLlib;**
- **GraphX.**

Spark Core. Core-модуль обеспечивает основные функциональные возможности фреймворка: управление задачами, распределение данных, планирование и выполнение операций в кластере. Именно Spark Core позволяет работать с данными в оперативной памяти: он предоставляет для этого специальное API и обеспечивает высокую скорость обработки.

Spark SQL. Модуль для работы со структурированными данными, который поддерживает SQL-подобный язык запросов. Он позволяет использовать типичные для реляционных баз данных операции на распределённых данных.

Spark Streaming. Модуль Streaming содержит методы для обработки данных в реальном времени. Он отбирает отдельные блоки из общего потока данных, переводит их в форму мультимножества и передаёт в другие модули Spark.

GraphX. Библиотека содержит объекты и методы, которые упрощают обход и анализ графовых структур. Кроме того, в GraphX есть готовый набор алгоритмов для решения типовых задач графовой аналитики: ранжирования страниц, **SVD++**, подсчёта треугольников, обнаружения сообществ и так далее.

MLlib. Эта библиотека для машинного обучения включает в себя различные алгоритмы классификации, регрессии, кластеризации данных и так далее.

Для работы с таблицами баз данных используем модуль **Spark SQL.**

Выполняем установку: *pip install pyspark*

Приложения PySpark запускаются с инициализации, ***SparkSession*** которая является точкой входа в PySpark:

from pyspark.sql import SparkSession

Создаём сеанс в переменной ***spark***

spark* = *SparkSession*.*builder*.*getOrCreate()

Далее всё аналогично, как в библиотеках **Pandas** и **Dask**. Можно создавать **DataFrame** из словаря, можно работать с базами данных, размещённых в таблицах. Для примера берём .csv-файл.

Создаём объект: ***df = spark.read.csv('путь к файлу')***

Работаем с объектом применяя необходимые методы.

7. Сравнение библиотек **Pandas**, **Dask** и **Spark**

7.1. Pandas и Dask

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 - **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.

В работе Поручикова Михаила и Цепкова Ярослава, г. Самара (*https://newtechaudit.ru/ot-pandas-k-dask/*) выполнено сравнение скорости выполнения вычисления в библиотеках **Dask** и **Pandas** от размера Data Set. График зависимости предоставлен на Рисунке 6. Из графика видно, что до определённого момента, скорость выполнения вычислений для **Dask** и **Pandas** практически одинакова. Но начиная с объёма обрабатываемой информации 1,5 ГБ (что составляет 18% от объёма оперативной памяти компьютера), скорость обработки в библиотеке **Pandas** существенно снижается и при достижении 50% от оперативной памяти, перестаёт работать. Что коррелируется с результатом полученном в моей работе.

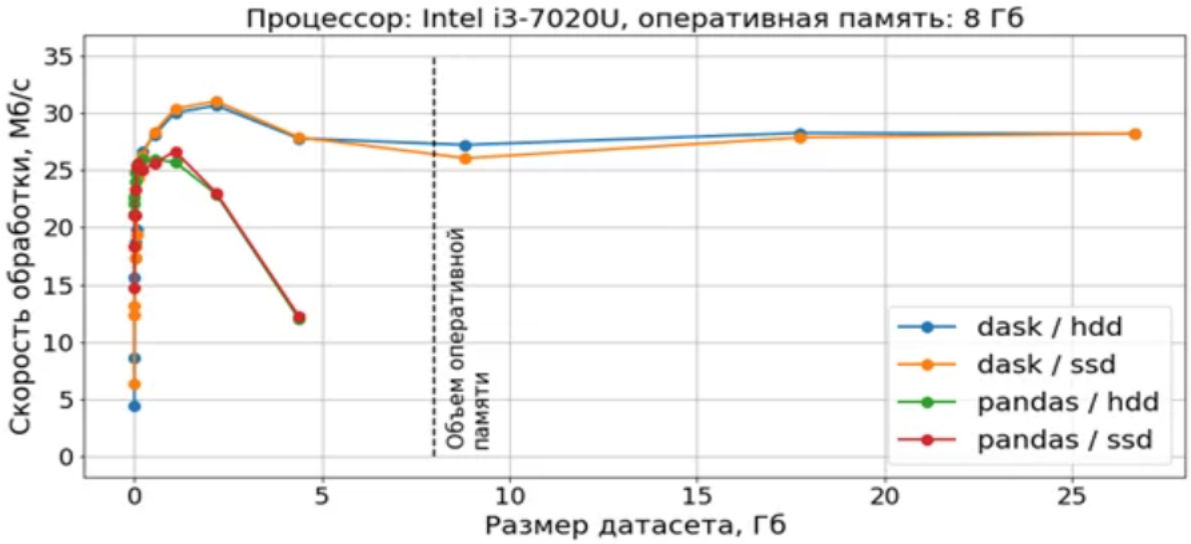

Рисунок 6. Зависимость скорости обработки от размера файла

В библиотеке **Dask** возможно работать со всеми однотипными файлами сразу: ***df = dd. read_csv('*.csv')***. В библиотеке **Pandas** такой возможности нет. Объект создаём, читая только один файл.

7.2. Dask и Spark

Apache Spark: Apache Spark — это высокопроизводительный фреймворк для распределенной обработки данных.

Ограничения **Apache Spark**: Установка и настройка **Apache Spark** может быть сложной. **Spark** лучше подходит для крупных кластеров и больших объемов данных, что может быть избыточным для небольших задач. **Dask** сравнение с **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**:

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. 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. 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**. Постоянная разработка и поддержка: **Dask** активно развивается и имеет активное сообщество разработчиков. Это гарантирует поддержку и обновления в будущем. Эффективное использование ресурсов: **Dask** позволяет более эффективно использовать ресурсы машины или кластера, что может снизить затраты на аппаратное обеспечение. Внутренняя модель Dask имеет более низкий уровень абстракции, поэтому в ней отсутствуют высокоуровневые оптимизации, но она способна реализовывать более сложные алгоритмы и создавать более сложные индивидуальные системы. Dask основан на общем планировании задач.

В каких случаях использовать **Spark**:

Apache Spark способен справляться с гораздо большими нагрузками, чем **Dask**. В частности, **Spark** отлично справляется терабайтными объемами и даже больше. Кроме того, **Spark** позволяет манипулировать данными с помощью SQL-запросов, а **Dask** – нет, что является недостатком феймворка с точки зрения аналитика данных. Наконец, **Spark** является более зрелым проектом и частью Big Data экосистемы **Apache**, включая тесную интеграцию с NoSQL-хранилищами и инструментами стека SQL-on-Hadoop типа Hive и Iceberg. **Spark** масштабируется до множества кластеров на тысячи узлов, а для **Dask** 1000 узлов – это предел.

8. Выводы

Для работы с базами данных, применяя язык **Pathon** и используя один компьютер, лучший вариант – это библиотеки **Pandas** и **Dask**. Выбирая между этими библиотеками, применяем **Pandas**, если объём обрабатываемых данных менее 20% от оперативной памяти устройства. В противном случае используем библиотеку **Dask**.

Оценку практических возможностей фреймворка **Spark** на отдельном персональном компьютере, используя язык программирования **Pathon**, дать очень сложно. Скорее всего работа со **Spark** – прерогатива обработки огромных потоков информации (терабайты и больше), поступающих постоянно и обрабатываемых на многих устройствах одновременно.