Дипломная работа

***Анализ и сравнение различных способов обработки и хранения больших данных: Pandas, Dask и Apache Spark***

Автор: Дмитриева С.Л.

2025

**Содержание**

1. Введение

- Актуальность темы

- Цели и задачи исследования

- Структура работы

2. Обзор технологий обработки больших данных

- Понятие "большие данные"

- Основные характеристики больших данных

3. Pandas

- Описание библиотеки

- Преимущества и недостатки

- Примеры использования

4. Dask

- Описание библиотеки

- Преимущества и недостатки

- Примеры использования

5. Apache Spark

- Описание платформы

- Преимущества и недостатки

- Примеры использования

6. Сравнительный анализ Pandas, Dask и Apache Spark

- Архитектура и система вычислений

- Подходы к обработке данных

- Масштабируемость и производительность

- Примеры сценариев использования

7. Разработка приложения для обработки БД с использованием Pandas, Dask и Apache Spark

- Описание выбора объектов для разработки приложений

- Планирование алгоритма приложений и его реализация

- Сравнительный анализ результатов работы приложений

- Структура проекта

8. Заключение

- Выводы о каждой технологии

- Рекомендации по выбору инструмента в зависимости от задач

***1.Введение***

В современном мире объемы данных стремительно растут, и организации сталкиваются с необходимостью эффективной обработки и хранения этих данных. Большие данные характеризуются объемом, большой скоростью обработки и разнообразием, что требует применения специализированных технологий для их анализа. В данной работе будет проведен анализ и сравнение трех популярных инструментов обработки больших данных: Pandas, Dask и Apache Spark.

***Актуальность темы***

С учетом постоянного увеличения объемов данных, выбор оптимальных технологий для их обработки и хранения становится критически важным. Понимание различных подходов и особенностей данных инструментов поможет аналитикам и разработчикам принимать более обоснованные решения.

В современном мире данные уже стали одним из самых ценных ресурсов. Они помогают принимать обоснованные решения, прогнозировать будущее и оптимизировать процессы. Но для этого необходимо не только хранить большие объемы данных, но и уметь их эффективно обрабатывать.

Хранение больших данных имеет множество преимуществ:

1. Анализ и прогнозирование: используя большое количество данных можно выявлять скрытые закономерности и тренды. Это полезно в различных сферах, от маркетинга до медицины.
2. Оптимизация процессов: данные помогают понять, как улучшить процессы и сделать их более эффективными.
3. Персонализация: компании могут лучше понимать своих клиентов и предлагать им индивидуализированные продукты и услуги.
4. Научные исследования: в науке большие данные позволяют проводить более точные и масштабные исследования.

Приведу примеры использования больших данных в маркетинге

Большие данные играют важную роль в маркетинге, позволяя компаниям лучше понимать своих клиентов и разрабатывать более эффективные стратегии. Вот несколько ключевых областей, где используются большие данные в маркетинге:

1. ***Персонализация рекламных кампаний:***

Используя большие данные, маркетологи могут собирать и анализировать информацию о поведении пользователей на сайтах, их предпочтениях и взаимодействиях с брендом.

Системы аналитики собирают данные из различных источников (например, социальных сетей, файлов журналов веб-серверов, данных транзакций). С помощью алгоритмов машинного обучения эти данные анализируются, и на их основе создаются персонализированные рекламные предложения для каждого пользователя. Например, если пользователь часто ищет спортивные товары, ему будут показываться релевантные объявления о спортивной экипировке.

1. ***Предсказание поведения клиентов:***

С помощью больших данных можно предсказать будущее поведение клиентов, включая вероятность покупки, отклик на рекламу и склонность к уходу.

Используются модели предсказательной аналитики, такие как логистическая регрессия или случайные леса, которые обучаются на исторических данных о покупках и взаимодействиях клиентов. Например, зная, что клиент часто покупает товары определённой категории, система может предсказать, когда он сделает следующую покупку, и предложить ему соответствующие продукты.

1. ***Анализ настроений:***

Большие данные позволяют анализировать настроения потребителей по отношению к бренду или продукту, используя данные из социальных сетей, обзоров и комментариев.

Алгоритмы обработки естественного языка (NLP) анализируют текстовые данные из таких источников, как ВКонтакте, Одноклассники, отзывы на маркетплейсах и т.д. Эти алгоритмы способны определить тональность сообщений (позитивная, негативная, нейтральная) и выделить ключевые темы, что помогает маркетологам понимать общественное мнение и корректировать стратегии.

1. ***Оптимизация ценообразования:***

Использование больших данных для динамического ценообразования позволяет компаниям устанавливать цены на продукты в реальном времени в зависимости от спроса, конкуренции, сезонности и других факторов.

Алгоритмы машинного обучения анализируют исторические данные о продажах, текущие рыночные условия и поведение конкурентов. На основе этих данных система может рекомендовать оптимальные цены для максимизации прибыли. Например, алгоритм может предложить снизить цену на продукт в периоды низкого спроса для стимулирования продаж.

1. ***Улучшение таргетинга:***

С помощью больших данных можно точно определять целевую аудиторию для маркетинговых кампаний, что повышает их эффективность.

Анализируются данные о демографии, географии, интересах и поведении пользователей. Используются кластеры (группировки) данных для определения наиболее перспективных сегментов аудитории. Например, алгоритмы могут выявить группу пользователей, которые с высокой вероятностью откликнутся на предложение скидки, и нацелить рекламу именно на них.

***Цели и задачи исследования***

Целью исследования является анализ и сравнение методов обработки и хранения больших данных с использованием Pandas, Dask и Apache Spark. Для достижения данной цели требуется решить следующие задачи:

* Изучить принципы работы каждой из технологий.
* Сравнить производительность и масштабируемость различных подходов.
* Определить преимущества и недостатки каждой технологии.

Python предлагает множество инструментов для обработки больших данных, каждый из которых имеет свои особенности и преимущества. Pandas является отличным выбором для работы с небольшими и средними наборами данных, благодаря своей простоте и функциональности. Dask и PySpark позволяют масштабировать обработку данных и эффективно использовать ресурсы системы, что делает их идеальными для работы с большими данными. Выбор инструмента зависит от конкретных задач и объема данных, с которыми вы работаете. Очевидно, что правильный выбор поможет ускорить решение задач по обработке больших данных, сократит требующиеся финансовые и технические ресурсы, в конечном итоге приведет к росту эффективности программных разработок в любой сфере экономики.

***2. Обзор технологий обработки больших данных***

***Понятие "большие данные"***

Большие данные представляют собой массивы информации, которые слишком велики или сложны для обработки с помощью традиционных методов и инструментов. Они имеют три основные характеристики:

* Объем: количество данных.
* Скорость: скорость их создания и обработки.
* Разнообразие: различные форматы и источники данных.

***Большие данные*** в первую очередь относятся к набору данных, которые слишком велики или сложны, чтобы с ними можно было справиться с помощью традиционной программной обработки данных. Данные с большим количеством записей (строк) обеспечивают большую статистическую достоверность, в то время как данные с более высокой сложностью (большее количество атрибутов или столбцов) могут привести к более высокому уровню ложных утверждений.

Задачи анализа больших данных включают в себя сбор, хранение и анализ данных, а также поиск, совместное использование данных, их передачу и визуализацию, запрос информации, обновление данных, конфиденциальность информации и источник данных. Изначально большие данные ассоциировались с тремя ключевыми понятиями: *объем*, *разнообразие* и *скорость*. Анализ больших данных сопряжен с трудностями при отборе проб и, таким образом, ранее допускал только наблюдения и выборку. Таким образом, четвертое понятие, *достоверность,* относится к качеству или достоверности данных. Без достаточных инвестиций в экспертные знания для обеспечения достоверности больших данных объем и разнообразие данных могут привести к затратам и рискам, которые превышают возможности организации создавать и извлекать пользу из больших данных*.*

Текущее использование термина *большие данные*, как правило, относится к использованию  прогнозной аналитики, анализа поведения пользователей или некоторых других передовых методов анализа данных, которые извлекают ценную информацию из больших данных, и редко к определенному размеру набора данных. Анализ наборов данных может выявить новые взаимосвязи для выявления тенденций в бизнесе, предотвращения заболеваний, борьбы с преступностью и так далее. Ученые, руководители предприятий, работники здравоохранения, рекламисты и  правительственные структуры  регулярно сталкиваются с трудностями при работе с большими наборами данных в таких областях, как поиск информации в интернете, финансовые технологии, аналитика здравоохранения, географические информационные системы, городская информатика  и бизнес-информатика. Ученые сталкиваются с ограничениями в работе с электронными науками (E-Science), включая  метеорологию, геномику, сложное физическое моделирование, биологию и исследования окружающей среды.

Размер и количество доступных наборов данных быстро растут по мере сбора данных такими устройствами, как  мобильные устройства, многочисленные устройства, считывающие информацию, воздушное оборудование (дистанционное зондирование), журналы программного обеспечения, камеры, микрофоны, считыватели  радиочастотной идентификации (RFID) и беспроводные сенсорные сети.   Размеры хранимой информации примерно удваиваются каждые 40 месяцев с 1980-х годов.   На основе прогноза отчета IDC (International Data Group), к 2025 году объем данных составит 163 зеттабайта. В отчете  Statista (Statista  - немецкая онлайн-платформа, специализирующаяся на  сборе и визуализации информации) прогнозируется, что мировой рынок больших данных вырастет до 103 миллиардов долларов к 2027 году.

В 2011 году  McKinsey & Company (McKinsey или McK - американская транснациональная  консалтинговая фирма по стратегии и управлению) сообщила, что если бы здравоохранение США творчески и эффективно использовало большие данные для повышения эффективности и качества, сектор мог бы ежегодно приносить прибыль более чем в 300 миллиардов долларов. В развитых странах Европы государственные администраторы могли бы сэкономить более 100 миллиардов евро (149 миллиардов долларов) только на повышении операционной эффективности за счет использования больших данных. А пользователи сервисов, основанных на данных о личном местоположении, могли бы получить прибыль в размере 600 миллиардов долларов. Одним из вопросов для крупных предприятий является определение того, кто должен владеть инициативами в области больших данных, которые затрагивают всю организацию.

Системы управления реляционными базами данных и пакеты настольного статистического программного обеспечения, используемые для визуализации данных, часто испытывают трудности с обработкой и анализом больших данных. Для обработки и анализа больших данных может потребоваться массово параллельное программное обеспечение, работающее на десятках, сотнях или даже тысячах серверов. То, что квалифицируется как *"большие данные",* варьируется в зависимости от возможностей тех, кто их анализирует, и их инструментов. Кроме того, расширение возможностей делает большие данные движущейся целью. Для некоторых организаций, впервые столкнувшихся с сотнями  гигабайт данных, может возникнуть необходимость пересмотреть варианты управления данными. Для других могут потребоваться десятки или сотни терабайт, прежде чем размер данных станет важным фактором.

Термин "*большие данные"* используется с 1990-х годов, и некоторые отдают должное Джону Мэши, американскому специалисту по информатике, за популяризацию этого термина. *Большие данные* обычно включают наборы данных, размеры которых превышают возможности обычно используемых программных средств сбора, контроля, управления и обработки данных за приемлемый промежуток времени. Философия больших данных охватывает неструктурированные, полуструктурированные и структурированные данные; однако основное внимание уделяется неструктурированным данным. "Размер" больших данных постоянно меняется; по состоянию на 2012 год объем данных варьируется от нескольких десятков терабайт до многих  зеттабайт. Для больших данных требуется набор методов и технологий с новыми формами интеграции для получения информации из  наборов данных, которые разнообразны, сложны и имеют массовый масштаб.

***3. Pandas***

***Описание библиотеки***

***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. Эта библиотека предоставляет специальные структуры данных, такие как Series и DataFrame 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 (собранный в разные моменты времени статистический материал о значении каких-либо параметров, в простейшем случае одного, исследуемого процесса).

Название библиотеки происходит от эконометрического термина «панельные данные», используемого для описания многомерных структурированных наборов информации.

Основная область применения Pandas — 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 статобработки языки (такие, как R и [Octave](https://ru.wikipedia.org/wiki/GNU_Octave)). Также активно ведётся работа по реализации категориальных типов данных.

Pandas прежде всего предназначен для очистки и первичной оценки данных по общим показателям, например, среднему значению, квантилям (в математической статистике – значение, которое заданная случайная величина не превышает с фиксированной вероятностью) и так далее; статистическим пакетом (программным продуктом, предназначенным для статистической обработки) он в полном смысле не является, однако наборы данных типов DataFrame и Series применяются в качестве входных данных в большинстве модулей анализа данных и машинного обучения. Например, SciPy - библиотеки для языка программирования Python с открытым исходным кодом, предназначенным для выполнения научных и инженерных расчётов; Scikit-Learn - популярной Python-библиотеки машинного обучения, предназначенной для взаимодействия с числовыми и научными библиотеками Python: NumPy и SciPy.

## *Основные возможности библиотеки:*

* Объект *DataFrame* для манипулирования индексированными массивами двумерных данных
* 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» (англ. *split-apply-combine*).
* 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 (язык программирования, упрощающий написание модулей на С/C++ для Python.) и Си.

Разработка пакета была начата в 2008 году Уэсом Маккини (англ. *Wes McKinney),* опубликовавшим исходный код библиотеки под свободной лицензией. В 2012 году к проекту присоединился Чан Шэ, ставший вторым главным разработчиком библиотеки. Примерно в то же время библиотека набрала популярность в среде Python-разработчиков, и к проекту присоединилось множество новых участников.

***Преимущества и недостатки***

***Преимущества****:*

* Простота использования и удобный API.
* Высокая скорость благодаря оптимизации кода.
* Интуитивно понятный интерфейс.
* Широкие возможности для обработки и анализа данных.
* Отличная поддержка табличных данных.
* Расширенные возможности за счет интеграции с другими библиотеками на Python, в частности, с NumPy, Matplotlib и Scikit-learn.
* Сильное мировое сообщество программистов, силами которого продукт постоянно совершенствуется.

***Недостатки:***

* ***Потребление памяти***. Pandas может потреблять большое количество оперативной памяти при работе с большими объёмами данных. Это может ограничивать возможность обработки очень больших наборов данных на компьютерах с ограниченными ресурсами.
* ***Скорость выполнения***. В сравнении с низкоуровневыми библиотеками, такими как NumPy, операции с данными в Pandas могут выполняться медленнее, особенно при работе с большими наборами данных.
* ***Неэффективное использование циклов***. Pandas лучше работает с векторизованными операциями, и его производительность снижается при использовании явных циклов.
* ***Проблемы слияния****.* При слиянии больших наборов данных Pandas может потребовать больших объёмов памяти и времени для выполнения операций объединения. Особенно это актуально для сложных слияний с большим числом повторяющихся значений.
* ***Некоторые сложные операции***. В редких случаях сложные операции с данными могут быть сложно выполнимы или требовать большого количества кода. В таких ситуациях может потребоваться более прямой подход с использованием других библиотек или специфических алгоритмов.
* ***Отсутствие поддержки больших кластеров***. В отличие от некоторых других библиотек для анализа данных, таких как Apache Spark, Pandas не предоставляет поддержку для распределённых вычислений на больших кластерах.
* ***Плохая структура документации****.* Документация Pandas обширна, но некоторые пользователи считают её сложной для навигации и недостаточной по практическим примерам для конкретных задач.

***Примеры использования***

Pandas широко используется для анализа данных в научных исследованиях, разработке финансовых спецификаций и в области образования, предоставляя возможность легко очищать и преобразовывать данные.

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

* **Series**— одномерный массив, который может хранить значения любого типа данных;
* **DataFrame** — двумерный массив (таблица), в котором столбцами являются объекты класса Series.

На примере DataFrame рассмотрим наиболее часто применимые операции по обработке данных.

## *1.* *Удаление столбцов*

Функция **drop** используется для удаления столбцов и строк. Мы передаем в нее метки строк или столбцов, которые нужно удалить.

**df.drop (['RowNumber', 'CustomerId', 'Surname', 'CreditScore'], axis=1, inplace=True)**

**'RowNumber', 'CustomerId', 'Surname', 'CreditScore' –** метки удаляемых столбцов или строк.

Параметр **axis** («ось») определяет, что именно мы удаляем: строки или столбцы. Для строк значение **axis** — 0, а для столбцов — 1.

Параметр **inplace** определяет, нужно ли вносить изменения в существующий датафрейм или нужно вернуть его копию. По умолчанию значение **inplace** — **False** (возвращается копия). У нас **inplace**=**True** для внесения изменений на месте. Мы удаляем 4 столбца, поэтому число столбцов уменьшается до 10 (было 14).

**df.shape**

**# (10000,10) # shape** – выводит на дисплей размеры датафрейма.

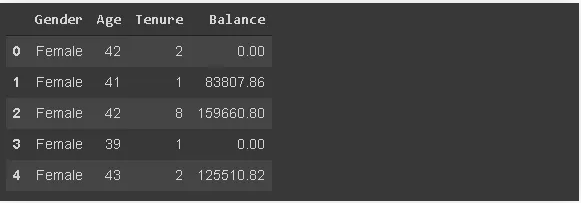
## *2. Выборка определенных столбцов при чтении*

Мы можем прочитать не целый CSV-файл, а только определенные столбцы. Для этого нужно передать список нужных столбцов в параметр **usecols** при считывании файла. При этом, необходимые столбцы лучше выбрать заранее, а не удалять ненужные после считывания всего файла.

**df\_spec = pd.read\_csv("/content/churn.csv", usecols=['Gender', 'Age', 'Tenure', 'Balance'])**

**df\_spec.head()**

**# content/churn.csv –** считываемый файл **# head()-** выводит на экран первые 5 строк



## *3. Чтение части датафрейма*

Функция **read\_csv** позволяет считывать часть csv-файла, определяя ее в строках. Применять эту функцию можно двумя способами.

Первый вариант – указать нужное количество строк в параметре **nrows**.

**df\_partial = pd.read\_csv ("/content/churn.csv", nrows=5000)**

**df\_partial.shape**

**# (5000,14)**

Используя **nrows**, мы создаем датафрейм, содержащий первые 5000 строк csv-файла.

Второй вариант — использовать параметр **skiprows**. Если мы укажем skiprows=5000, при считывании файла первые 5000 строк будут пропущены.

## *4. Случайная выборка*

После создания датафрейма вы можете захотеть просмотреть небольшую выборку из него. Для определения размера выборки можно использовать следующие параметры:

**n** — число строк в выборке

**frac**— доля от всего датафрейма

**df\_sample = df.sample(n=1000)**

**df\_sample.shape**

**# (1000, 10)**

**df\_sample2 = df.sample(frac=0.1)**

**df\_sample2.shape**

**# (1000,10)**

## *5. Проверка пропущенных значений*

Функция **isna** определяет пропущенные значения в датафрейме. Используя ее с функцией ***sum***, можно увидеть количество недостающих значений по каждому столбцу.

**df.isna().sum()**



Здесь нет пропущенных значений.

## *6. Добавление пропущенных значений при помощи* loc *и* iloc

Методы **loc** и **iloc** служат для выборки строк и столбцов по индексу или меткам.

* [**loc**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html) — выборка по меткам
* [**iloc**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html) — выборка по индексу

Давайте вначале создадим переменную, содержащую 20 случайных индексов.

**missing\_index = np.random.randint(10000, size=20)**

Эти индексы мы используем, чтобы заменить некоторые значения на **np.nan** (пропущенное значение). Мы применяем функцию **loc** и указываем метки столбцов — ['Balance', 'Geography'].

**df.loc[missing\_index, ['Balance','Geography']] = np.nan**

Теперь в столбцах Balance и Geography 20 пропущенных значений.

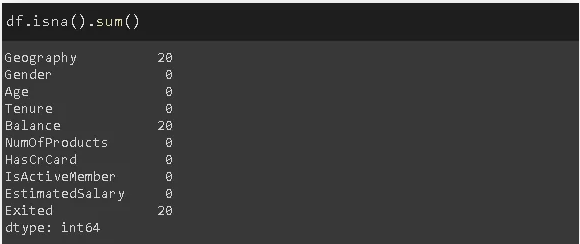
Теперь применим **iloc** и индексы столбцов вместо меток.

**df.iloc[missing\_index, -1] = np.nan**

-1 — это индекс последнего столбца (**Exited**).

Хотя мы использовали разные представления столбцов для **loc** и **iloc,** значения строк не изменились. Причина в том, что в качестве меток строк мы используем числовой индекс. То есть, для строк метки и индексы одинаковы.

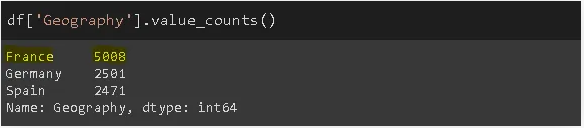
Количество пропущенных значений поменялось:



## *7. Заполнение отсутствующих значений*

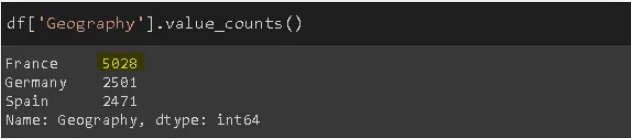
При помощи функции [**fillna**](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html) можно заполнить недостающие значения. У этой функции много опций. Мы можем использовать конкретное значение, агрегатную функцию (например, среднее значение), предыдущее или следующее значение.

В этом примере мы выбрали в качестве заполнителя для пропущенных значений в столбце Geography наиболее часто встречающееся значение.



**mode = df['Geography'].value\_counts().index[0]**

**df['Geography'].fillna(value=mode, inplace=True)**



А для столбца Balance используем среднее значение:

**avg = df['Balance'].mean()**

**df['Balance'].fillna(value=avg, inplace=True)**

## *8. Удаление пропущенных значений*

Пропущенные значения можно не только заполнять, но и просто удалять. У нас есть такие пропуски значений в столбце **Exited**. При помощи следующего кода можно удалить строки, содержащие любые пропущенные значения.

**df.dropna(axis=0, how='any', inplace=True)**

Если задать параметр **axis**=1, то удалены будут не строки, а столбцы с отсутствующими значениями. Можно также задать параметр **thresh**, чтобы указать, какое минимальное число непропущенных значений должно быть в столбце или строке. Например, **thresh**=5 означает, что строка должна иметь как минимум 5 значений. Строки с меньшим количеством значений будут удалены.

Теперь в датафрейме нет пропущенных значений.

**df.isna().sum().sum()**

**# 0**

## *9. Выборка строк на основе условий*

В некоторых случаях нам нужно сосредоточиться на данных, отвечающих каким-то условиям. Например, можно выбрать отпавших клиентов, которые живут во Франции.

**france\_churn = df[(df.Geography == 'France') & (df.Exited == 1)]**

**france\_churn.Geography.value\_counts()**

**# France 808**

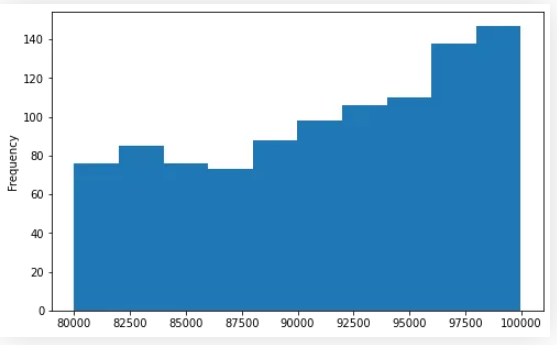
## *10. Описание условий при помощи* query

Функция **query** позволяет задавать условия более гибко. Например, при помощи строк:

**df2 = df.query('80000 < Balance < 100000')**

Давайте проверим результат, нарисовав диаграмму столбца Balance.

**df2['Balance'].plot(kind='hist', figsize=(8,5))**



Pandas предлагает куда больше возможностей, здесь приведены простые примеры использования Pandas в Python.

***4. Dask***

***Описание библиотеки***

Dask - это библиотека для параллельных вычислений на Python, которая позволяет обрабатывать большие данные, распределяя задачи по нескольким процессам и машинам. Dask был создан Мэтью Роклином в декабре 2014 года.

Dask предоставляет аналогичные структуры данных (DataFrame и Array), как Pandas, но с возможностью работы с данными, превышающими объем оперативной памяти. Dask предоставляет знакомый пользовательский интерфейс, а также API  для взаимодействия с библиотеками экосистемы PyData, включая Pandas, Scikit-learn и NumPy.

Dask лучше всего подходит для работы с научными вычислениями, данными размером в несколько гигабайт до терабайтов, а также для обработки данных в реальном времени.

Он идеально подходит для обработки данных, где расчетные графы зависят от предыдущих результатов, таких как сложные вычисления в рамках машинного обучения или численного моделирования.

***Основные компоненты Dask****:*

* ***Dask Arrays****.* Это аналог NumPy массивов, разработанный для работы с большими объёмами данных, которые не помещаются в оперативной памяти.
* ***Dask DataFrames****.* Это аналог Pandas DataFrames, который позволяет работать с таблицами данных, которые не помещаются в памяти.
* ***Dask Bags****.* Это структура данных, предназначенная для работы с неструктурированными данными, такими как JSON-объекты.

***Преимущества и недостатки***

***Преимущества:***

* ***Параллельные вычисления***. Dask обеспечивает параллельное выполнение задач, используя несколько ядер на одной машине или распределённых по кластеру.
* ***Масштабируемость****.* Dask может масштабироваться от одного ноутбука до большого кластера, что делает его подходящим как для небольшой, так и для крупномасштабной обработки данных.
* ***Совместимость****.* Dask легко интегрируется с популярными библиотеками Python, такими как NumPy, pandas и scikit-learn.
* ***Гибкое планирование***. Dask предоставляет различные планировщики, включая однопоточные, многопоточные, многопроцессорные и распределённые планировщики, для удовлетворения различных вычислительных потребностей.
* ***Демократизация машинного обучения***. Dask делает крупномасштабное машинное обучение более доступным, позволяя даже пользователям с ограниченными ресурсами работать с большими данными.
* ***Легкость в использовании для пользователей Pandas.***  **Многие операции, такие как фильтрация, группировка и объединение, разработаны идентично.**
* ***Поддержание различных форматов данных.*** Dask поддерживает форматы данных CSV, Parquet и HDF5, что позволяет легко интегрироваться с существующими хранилищами данных.
* ***Сообщество разработчиков активно развивает Dask***. Благодаря этому, создаются новые возможные интеграции и функционал.

***Недостатки:***

* ***Большая зависимость*** ***от конфигурации системы***, так как от неё зависит, как будут выполняться вычисления – параллельно или распределённо.
* ***В Dask разработаны аналогично лишь некоторые функции Pandas***. Некоторые задачи сложно распараллелить, например, сортировку данных и назначение индексов неотсортированным столбцам.
* ***Dask не решает абсолютно все задачи анализа и обработки данных***. Библиотеку рекомендуется использовать только для работы с наборами данных, которые не помещаются в памяти целиком.
* ***Поскольку Dask основан на Pandas, то всё, что медленно работает в Pandas, остаётся медленным и в Dask****.*
* ***Объекты Dask неизменяемы и не поддерживают inplace-операции***. Поэтому в Dask нельзя присваивать новые значения с помощью метода loc().

***Примеры использования***

Dask находит применение в задачах, требующих обработки данных, превышающих ресурсы одной машины, например, в анализе больших наборов данных в области машинного обучения (machine learning) и научных исследований.

Dask Dataframes может считывать и хранить данные во многих из тех же форматов, что и Pandas Dataframes.

Варианты использования Dask можно разделить на две категории:

* Можно оптимизировать вычисления с помощью динамического планирования задач.
* Большие наборы данных могут обрабатываться с использованием коллекций "Больших данных", таких как параллельные массивы и фреймы данных.

График задач, визуальное изображение организации заданий по обработке данных, могут быть составлены с использованием коллекций Dask.

Использование следующих наборов данных делают Dask доступным для применения:

* ***Dask.array*** - Используя интерфейс Numpy, *dask.array* делит огромный массив на меньшие, позволяя нам выполнять вычисления на массивах, размер которых превышает объем системной памяти.

*Dask Arrays* 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:

**import dask.array as da  
 # Создание Dask Array из массива NumPy  
 numpy\_array = np.arange(1000)  
 dask\_array = da.from\_array(numpy\_array, chunks=100) # Разбиваем на блоки по 100 элементов**

* ***Dask.bag*** – 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 **filter**, **map**, **group** **by** и **fold**.

**import dask.bag as db**  
 **# Создание Dask Bag из списка словарей**  
 **data = [{'name': 'Alice', 'age': 30},  
 {'name': 'Bob', 'age': 25},  
 {'name': 'Charlie', 'age': 35}]  
 bag = db.from\_sequence(data)**

* ***Dask.dataframe*** — это аналог Pandas DataFrames, который позволяет работать с таблицами данных, которые не помещаются в памяти. Dask DataFrames поддерживает множество операций, таких как фильтрация (**query)** , сортировка (**sort)**, группировка (**groupby)** и агрегация данных (**agg)** , а также объединение таблиц (**concat)**.

**import dask.dataframe as dd  
 # Создание Dask DataFrame из CSV файла  
 df = dd.read\_csv('data.csv')**

***5. Apache Spark***

***Описание платформы***

*Apache Spark* (от англ. *spark* — искра, вспышка) – это фреймворк с открытым исходным кодом для реализации распределённой обработки данных, входящий в экосистему проектов *Hadoop*. В отличие от классического обработчика из ядра *Hadoop*, реализующего двухуровневую концепцию *MapReduce* с хранением промежуточных данных на накопителях, *Spark* работает в парадигме резидентных вычислений — обрабатывает данные в оперативной памяти. Благодаря этому, использование *Spark* 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.

*Spark*  предоставляет программные интерфейсы для языков Java, Scala, Python, R. Изначально *Spark*  был написан на Scala, 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-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, так и без компонентов ядра Hadoop, поддерживает несколько распределённых систем хранения — HDFS, OpenStack Swift, NoSQL-СУБД Cassandra, Amazon S3.

Ключевой автор – румынско-канадский учёный в области информатики Матей Захария (англ. *Matei Zaharia*), начал работу над проектом в 2009 году, будучи аспирантом Университета Калифорнии в Беркли. В 2010 году проект опубликован под лицензией BSD, в 2013 году передан фонду Apache и переведён на лицензию Apache 2.0, в 2014 году принят в число проектов верхнего уровня Apache.

***Преимущества и недостатки***

***Преимущества:***

* ***Богатый API.*** Apache Spark предоставляет разработчику довольно обширный API, позволяя работать с разными языками программирования: Python, R, Scala и Java. Также возможна интеграция Spark с такими BigData фреймворками, *как Apache Kafka* и *Apache KUDU*, а также с СУБД *MySQL* посредством компонента *Spark SQL.*
* ***Широкие функциональные возможности за счет многокомпонентного состава в виде модулей Spark SQL, Spark Streaming, MLib***. *Spark SQL* – модуль, который служит для аналитической обработки данных с помощью SQL-запросов; *Spark Streaming* – модуль, обеспечивающий надстройку для обработки потоковых данных в режиме онлайн; *MLib* – модуль, предоставляющий набор библиотек машинного обучения.
* ***Отложенные или «ленивые» вычисления (lazy evaluation).*** «Ленивые» вычисления позволяют снизить общий объем вычислений и повысить производительность программы за счет снижений требований к памяти. Также можно проверить структуру конечного результата, не выполняя какие-либо промежуточные шаги. Еще Spark автоматически проверяет план выполнения запросов или программы на наличие ошибок.
* ***Распределенная обработка данных.*** Apache Spark предусматривает распределенную обработку данных, включая концепцию [*RDD*](https://spark-school.ru/wiki/rdd-wiki/) (resilient distributed dataset) – это распределенная структура данных, размещаемая в оперативной памяти. Spark сам раскидывает код по всем узлам кластера, разбивает его на подзадачи, создает план выполнения и отслеживает успешность выполнения.
* ***Простые вращения данных при матричных и векторных операциях.*** Вращение данных (data pivoting) считается проблемой для многих Фреймворков, работающих с большими данными, например, *Apache Kaflka* или *Flink.* Spark имеет в своем арсенале простой и интуитивно понятный способ поворота Datafame. Пользователю необходимо всего лишь выполнить операцию *groupBy* для столбцов с целевым индексом, поворачивая целевое поле для дальнейшего использования в качестве столбцов.
* ***Легкие преобразования одних структур данных в другие.*** Для поддержки языка Python сообщество Apache Spark выпустило инструмент *PySpark*, который предлагает модуль *Pyspark Shell*, связывающий Python API и контекст Spark. Поэтому разработчик Big Data приложения может применять метод *toPandas* для беспрепятственного преобразования Spark DataFrame в Pandas. *PySpark* расширяет функциональность языка Python до операций с массивными наборами данных, ускоряя цикл разработки.
* ***Динамично развивающийся open-source проект с активным профессиональным сообществом.***

***Недостатки:***

* ***Псевдопотоковая обработка в режиме micro-batch***. Spark разбивает непрерывный поток данных на серию микропакетов, из-за чего возможны временные задержки порядка секунды.
* ***Высокое потребление памяти и повышенные требования к оборудованию***. В отличие от классического MapReduce, Spark не записывает промежуточные данные на диск, а размещает их в оперативную память. Сервера, на которых развёрнут Spark, требуют большего объёма RAM, что ведёт к удорожанию кластера.
* ***Сложности при разработке распределённых приложений****.* Код распределённого приложения Apache Spark недостаточно изолирован от кода самого Фреймворка. Это может привести к сбоям приложения при обновлении версии Spark или отдельных библиотек, которые использовались при разработке.
* ***Проблема SQL-инъекций***. Для Spark характерна уязвимость, когда в запрос к базе данных внедряется вредоносный код.
* ***Отсутствие собственной системы управления файлами***. Apache Spark зависит от других платформ, например Hadoop или облачных, для системы управления файлами.

***Примеры использования***

Apache Spark применяется в крупных компаниях для обработки и анализа данных в реальном времени, например, в финансовом секторе для обнаружения мошенничества.

Эффективная загрузка и преобразование данных – это фундамент любого успешного анализа с использованием Apache Spark и DataFrames. Spark 3.0.1 предоставляет мощный инструментарий для работы с различными источниками данных и выполнения широкого спектра преобразований. Рассмотрим несколько практических примеров, используя PySpark для иллюстрации.

* ***Загрузка данных***: Spark DataFrames поддерживают множество форматов данных, включая CSV, JSON, *Parquet, ORC* и многие другие. Выбор формата зависит от специфики данных и требований к производительности. *Parquet*, например, известен своей эффективностью хранения и считывания данных, что особенно важно при работе с большими объемами информации. Вот как загрузить данные из CSV-файла:

**from pyspark.sql import SparkSession**

**spark = SparkSession.builder.appName("DataFrameExample").getOrCreate**

**# Загрузка данных из CSV-файла**

**df = spark.read.csv("path/to/your/file.csv", header=True, inferSchema=True)**

**df.show**

Здесь **header=True** указывает, что первая строка файла содержит заголовки столбцов, а **inferSchema=True** позволяет Spark автоматически определить типы данных для каждого столбца.

* ***Преобразование данных***: После загрузки данных часто необходимо выполнить различные преобразования для подготовки данных к анализу. Spark DataFrames предоставляют широкий набор функций для этого. Рассмотрим несколько примеров:
* ***Фильтрация***: Выберем строки, удовлетворяющие определенным условиям:

**filtered\_df = df.filter(df["column\_name"] > 10)**

**filtered\_df.show**

* ***Группировка и агрегация***: Группируем данные по одному или нескольким столбцам и вычисляем агрегатные функции (среднее, сумма, количество и т.д.):

**grouped\_df = df.groupBy("column\_name").agg({"another\_column": "avg"})**

**grouped\_df.show**

* ***Изменение имен столбцов***: Переименование столбцов для удобства анализа:

**renamed\_df = df.withColumnRenamed("old\_name", "new\_name")**

**renamed\_df.show**

* ***Соединение DataFrames:*** Объединение нескольких DataFrames на основе общих столбцов:

**joined\_df = df1.join(df2, "common\_column", "inner")**

**joined\_df.show**

Эти примеры демонстрируют лишь малую часть возможностей Spark DataFrames для загрузки и преобразования данных. Более сложные преобразования могут включать в себя использование *window* функций, UDFs (User Defined Functions) и другие техники. Эффективность преобразований зависит от выбора подходящих функций и оптимизации запросов.

***6. Сравнительный анализ Pandas, Dask и Apache Spark***

При сравнении Pandas, Dask и Apache Spark следует учитывать различные аспекты:

***Архитектура и система вычислений***

* ***Pandas***: работает в памяти как локальный инструмент. Он работает на одном компьютере, и все данные загружаются в оперативную память.  Это делает его пригодным для небольших и средних наборов данных, которые могут уместиться в памяти (обычно до нескольких миллионов строк). Архитектура Pandas не предназначена для работы с данными, превышающими объём оперативной памяти, поэтому при больших объёмах данных могут возникнуть проблемы с производительностью и памятью.

Однако стоит учитывать, что Pandas хранит данные в памяти непрерывным блоком, что ускоряет доступ к ним и манипуляции с ними. При этом инструменту нужно выделить достаточно памяти для хранения всего DataFrame, даже если он работает только с подмножеством данных.

* ***Dask***: поддерживает параллельные вычисления с помощью распределенной обработки. Параллельные вычисления с Dask Arrays выполняются автоматически при выполнении операций. Dask разбивает данные на блоки и распределяет вычисления по доступным ядрам. Dask самостоятельно управляет параллельными вычислениями и оптимизирует их выполнение.

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.

* ***Apache Spark:*** построен на основе распределенной архитектуры и оптимизирован для обработки больших данных. Операции с RDD также могут быть распределены по кластеру и выполнены параллельно в пакетном режиме, что обеспечивает быструю и масштабируемую параллельную обработку. Apache Spark преобразует команды пользователя по обработке данных в Directed Acyclic Graph, или DAG. DAG является слоем планирования Apache Spark; он определяет, какие задачи выполняются на каких узлах и в какой последовательности.

***Подходы к обработке данных***

* ***Pandas***: оптимален для небольших и средних наборов данных. Библиотека Python предоставляет высокоуровневые структуры данных, которые позволяют легко работать с табличными данными. Операции в Pandas оптимизированы для работы по столбцам, что во многих случаях обеспечивает более высокую производительность, чем при операциях по строкам.

Однако когда объём данных становится слишком большим, Pandas может столкнуться с проблемами производительности. В таких случаях стоит рассмотреть другие инструменты, которые могут обрабатывать данные более эффективно, в частности, Dask.

Для наборов данных средних размеров лучше попробовать более эффективно использовать Pandas, а не переходить на другие инструменты. Например, подбирая подходящие типы данных, хранящихся в столбцах табличных структур DataFrame, можно снизить потребление памяти почти на 90%.

* ***Dask***: может обрабатывать большие данные, превышающие объем оперативной памяти. Это достигается за счёт того, что Dask распределяет вычисления по нескольким ядрам процессора или даже по нескольким машинам. Также библиотека хранит все данные на диске и обрабатывает фрагменты данных с диска, чтобы потреблять меньше памяти во время вычислений.
* ***Apache Spark***: подходит для обработки любого объема данных, начиная от малых до экстраординарных.

Однако у него есть особенности, которые позволяют ускорить обработку данных:

***Обработка в памяти*.** Spark хранит и обрабатывает данные в оперативной памяти, что обеспечивает большую скорость, чем загрузка и чтение данных с диска.

**«*Ленивые» вычисления.*** Операции над данными проводятся только перед непосредственным использованием результатов этих операций. Благодаря этому вычислительные мощности не тратятся на вычисления, которые понадобятся когда-то в будущем.

***Параллельная обработка и комбинирование операций.*** Spark распределяет данные и вычисления по нескольким узлам в кластере, выполняя разные операции обработки параллельно в режиме реального времени.

При этом у платформы есть недостаток: она может показывать более низкую производительность при работе с множеством файлов маленького размера.

***Масштабируемость и производительность***

* ***Pandas***: ограниченная масштабируемость, эффективен для небольших наборов данных.
* ***Dask***: высокая масштабируемость, подходит для средних и больших объемов данных.
* ***Apache Spark***: очень высокая производительность и масштабируемость, подходит для масштабных производственных задач.

***7.Разработка приложения для обработки БД с использованием Pandas, Dask и Apache Spark***

***Описание выбора объектов для разработки приложений***

Для реализации приложений с использованием Pandas, Dask и Apache Spark была выбраны структуры Dataframe, исходные файлы для обработки - csv-файлы. Приложение Apache Spark реализовано с использованием фреймворка Pyspark.

В процессе подготовки проекта дважды была проведена смена исходного csv-файла. Окончательно был сделан выбор файла из 500 000 строк и 9 столбцов, содержащих сведения о неких персонах.

Для подготовки исходного файла первоначально с помощью Excel была проведена следующая подготовка:

- из названий колонок были убраны все символы и пробелы, так как это обстоятельство в некоторых приложениях вызывало ошибки;

- были убраны периодически встречающиеся символы кавычек;

- был удален случайно образовавшийся в некоторых строках десятый столбец;

судя по всему, это произошло по причине того, что в этом столбце была указана профессия, и в некоторых столбцах через запятую было указано место работы.

Для удаления десятого столбца была написана простейшая программа на Python, которая позволяла построчно считывать по девять символов из исходного и записывать в новый csv-файл.

Также для того, чтобы можно было сравнить работу всех трех приложений по обработке файлов, было написано небольшое приложение на Pandas, позволяющее создавать csv-файлы определенного размера. Алгоритм подготовки этих файлов можно описать следующим образом:

- исходный файл считывался в DataFrame;

- первый csv-файл для обработки был получен многократным удвоением полученного DataFrame, вплоть до 800 Мб. Как только объем DataFrame достигал 800 Мб или превышал его, производилась запись полученного DataFrame в csv-файл. Каждый последующий csv-файл был получен посредством удвоения предыдущего. В результате было получено 7 csv-файлов размером от 110 Мб до 7027 Мб.

***Планирование алгоритма приложений и его реализация***

Предполагалось, что алгоритм всех трех приложений будет одинаков:

- последовательно считывается каждый csv-файл, проходит цикл операций по обработке полученных DataFrame;

- происходит фиксация времени начала и конца обработки;

- в процессе обработки файла на дисплей выводятся результаты каждого вида операций;

-после обработки всех файлов формируется текстовый файл, содержащий строку длительностей обработки каждого из 7 файлов;

- для получения максимально близких последовательностей видов обработки были выбраны схожие операции.

В конечном итоге были выбраны следующие виды обработки DataFrame:

- замена пропущенных значений в строках на ‘Unknown’ (этот выбор очевиден, так как все столбцы, кроме первого ID, были строковыми данными);

- вывод типов данных для столбцов;

- вывод на дисплей параметров DataFrame: количество строк (rows), и колонок (columns);

- получение и вывод на дисплей статистических данных по столбцам DataFrame.

- группировка по столбцу «Пол», и вывод сообщений о количестве соответственно мужчин и женщин;

- фильтрация сведений с разделением на три группы по дате рождения:

- родившиеся до 01.01.1950;

- родившиеся в интервале от 01.01.1950 до 01.01.2000;

- родившиеся после 01.01.2000.

Последним действием приложения было перенесение полученных текстовых файлов с данными о длительности обработки каждого csv-файла. Перенос этого файла осуществляется в определенную папку для дальнейшей визуализации результатов. После отработки всех трех приложений в папке display\_results было сформировано три текстовых файла: dask.txt, pandas.txt, pyspark.txt. В этом приложении в этой же папке находится небольшой скрипт, позволяющий извлечь информацию их полученных текстовых файлов, и вывести на экран сравнительный график, иллюстрирующий изменения времени обработки каждого csv-файла во всех приложениях.

В начале процесса подготовки приложений пришлось изменить объем оперативной памяти (ОП) ПК. Первоначально имеющийся объем ОП составлял 8Гб. В процессе предварительной подготовки выяснилось, что объем доступной ОП составлял только 25-30% от имеющейся памяти. Объем новой ОП составил 12Гб, что позволило получить в процессе обработки от 6 до 7 Гб доступной ОП.

Изменение размера ОП свести к минимуму вероятность сбоев при обработке больших файлов. Кроме того, для максимального уменьшения вероятности сбоев, связанных с объемом ОП, при реализации Pandas-приложения было принято решение считывать и обрабатывать строки csv-файла чанками по 10 000 000 строк. Данное ограничение позволило считывать без разбиения на части первые 4 сравнительно небольших файла. Но последующие файлы, начиная с csv\_04.csv, считывались по частям. Исходя из этого, алгоритм обработки приложений, реализованных на Pandas, имеет отличия. Каждый считываемый файл большого размера считывался и обрабатывался частями. При этом, полученные результаты по каждой обрабатываемой части, сохранялись и объединялись друг с другом.

После обработки всех частей полученная информация так или иначе объединялась и выводилась на дисплей.

***Сравнительный анализ результатов работы приложений***

Надо отметить, что скорость обработки файлов в Pandas-приложении, особенно двух последних, неизбежно падала, но, тем не менее, результаты показали, что Pandas хорошо справляется с обработкой. Вплоть до обработки предпоследнего файла размером 3,5Гб, это приложение опережало по скорости обработки Dask-приложения, и шло практически вровень с приложением на PySpark при обработке файлов размером до 1,76Гб. Результаты обработки файлов объемом 7,03Гб дают основание подтвердить тот факт, что скорость обработки при объемах данных близких к критическим, резко падает у Pandas-приложения.

При этом скорость выполнения обработки файлов у Dask-приложения увеличилась практически линейно в течение обработки всех 7 файлов.

Практически прямая достаточно пологая линия графика изменения скорости обработки у PySpark-приложения, позволяет утверждать, что в предложенных условиях PySpark оказался несомненным лидером.

При подготовке PySpark-приложения понадобилось увеличить объем выделяемой памяти для executor и driver до 4Гб.

При расчете статистических данных с помощью оператора describe() в PySpark-приложении количество данных меньше по сравнению с остальными приложениями, но это в некоторой степени компенсируется тем обстоятельством, что при фильтрации файлов по столбцу даты рождения (DateOfBirth), каждый результат сортировался, чего не было в Pandas- и Dask-приложениях.

***Структура проекта***

- в папке **Dask\_task**:

- **data\_processing\_dask.py**(Dask-приложение)

- в папке **display\_results**:

- **dask.txt** \

**- pandas.txt** -- результаты работы приложений

- **pyspark.txt** /

- **print\_results.py** – программа вывода результатов работы всех приложений на дисплей.

- в папке **Pandas\_task**:

-**data\_prpcessing\_pandas.py** (Pandas-приложение)

- в папке **Prelimenery\_preparetion**:

- в папке **received\_files**:

- **файл csv\_00.csv; \**

**- - - - - - - - - -** полученные файлы для обработки приложениями

**- файл csv\_06.csv; /**

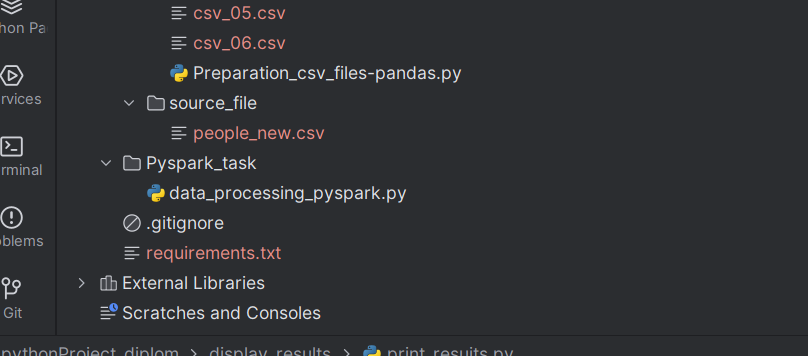
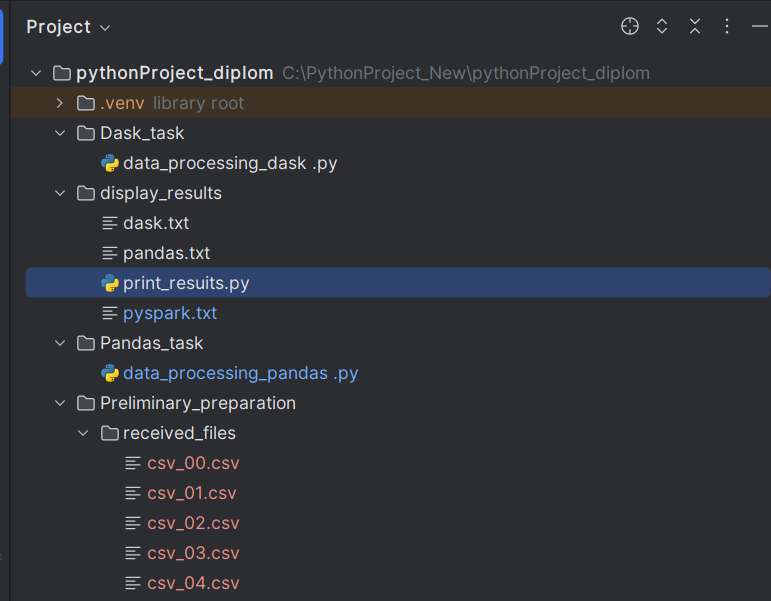
- **Preparation\_csv\_files\_pandas.py** (приложение на Pandas для формирования 7 файлов различного размера.

- в папке **source\_file**:

- **people\_new.csv** (исходный файл для формирования обрабатываемых csv-файлов)

- папка **Pyspark\_task**:

- **dask\_processing\_pyspark.py** (Pyspark-приложение)



***8. Заключение***

В данной работе были проанализированы три популярных инструмента для обработки и хранения больших данных: Pandas, Dask и Apache Spark. Каждый из них имеет свои особенности, преимущества и недостатки.

- Pandas удобен и прост в использовании для небольших данных.

- Dask подходит для среднечисленных задач и данных, превышающих память.

- Apache Spark является самым мощным инструментом для обработки больших объемов данных на распределенной архитектуре.

При выборе инструмента необходимо учитывать характеристики данных и требования к производительности. Перспективы развития технологий обработки больших данных обещают расширение возможностей существующих инструментов, а также появление новых решений, направленных на улучшение производительности и удобства работы с большими данными.