Исследование Polars и Pandas: выбор лучшей библиотеки для анализа и обработки данных

Воронкин Р.А., Криворот В.Г.

Постановка задачи: в современном мире анализ данных стал ключевым элементом в принятии стратегических решений и достижении конкурентных преимуществ в различных областях. Для успешного анализа данных необходимо использование эффективных инструментов, способных обрабатывать и анализировать большие объемы информации. В этой связи, библиотеки Polars и Pandas становятся важными инструментами для работы с данными в языке программирования Python.

Цель работы: сравнение библиотек Polars и Pandas в контексте их функциональности, производительности и практической применимости. Для достижения этой цели были поставлены следующие задачи:

1. Изучить основные возможности и синтаксис библиотек Polars и Pandas.
2. Сравнить производительность двух библиотек на различных операциях и в зависимости от объема данных.
3. Продемонстрировать примеры использования Polars и Pandas для решения типичных задач анализа данных.
4. Выявить преимущества и недостатки каждой библиотеки и дать рекомендации по выбору на основе конкретных потребностей.

Используемые методы: анализ документации, выполнение кода и анализ результатов. Результаты работы представлены в виде сравнительного анализа функциональности, производительности и применимости Polars и Pandas на различных задачах анализа данных.

Практическая значимость: помощь исследователям данных и специалистам в выборе наиболее подходящей библиотеки для своих задач анализа данных. Корректный выбор инструментов может значительно повысить эффективность работы и ускорить процесс анализа данных.

Результаты: данная работа позволит исследователям данных и специалистам сделать информированный выбор между библиотеками Polars и Pandas в зависимости от своих конкретных потребностей. Анализ функциональности и производительности данных библиотек, а также примеры использования помогут определить, какая библиотека лучше подходит для различных задач анализа данных. Кроме того, предоставленные рекомендации помогут оптимизировать процесс работы с данными и повысить эффективность анализа.

Ключевые слова: библиотеки Polars, Pandas, сравнение, анализ данных, производительность, функциональность, синтаксис, структуры данных, выбор инструментов.

Установка и импорт

Обе библиотеки Polars и Pandas являются открытым исходным кодом и поддерживаются активными сообществами разработчиков. В этом разделе будет рассмотрен процесс установки и импорта обеих библиотек, а также отметим некоторые различия между ними.

Установка Polars

Для установки библиотеки Polars необходимо выполнить следующие шаги:

1. Установите Python на вашу систему, если у вас его еще нет. Polars поддерживает Python 3.7 и выше.

2. Откройте терминал или командную строку и выполните следующую команду для установки Polars с помощью пакетного менеджера pip:

```
pip install polars
```

Установка Polars может потребовать установки дополнительных зависимостей, таких как Rust или CMake. В таком случае, следуйте инструкциям, предоставляемым при установке. 3. После завершения

установки вы можете импортировать Polars в свой проект Python с помощью следующей строки кода:

```
import polars as pl
```

Установка Pandas

Установка библиотеки Pandas аналогична установке Polars. Вот несколько шагов, которые нужно выполнить:

1. Убедитесь, что Python установлен на вашей системе.
2. Откройте терминал или командную строку и выполните следующую команду для установки Pandas:

```
pip install pandas
```

3. После завершения установки вы можете импортировать Pandas в свой проект Python:

```
import pandas as pd
```

Различия в процессе установки и импорта

Хотя процесс установки и импорта Polars и Pandas в основном схож, есть некоторые различия:

1. Зависимости: Polars может потребовать установки дополнительных зависимостей, таких как Rust или CMake, в то время как Pandas не требует этого.
2. Импорт: При импорте Polars используется import polars as pl, а при импорте Pandas используется import pandas as pd.

Структура данных

В этом разделе рассматриваются основные структуры данных, используемые в библиотеках Polars и Pandas, а также описываются различия в их представлении и манипуляции данными.

DataFrame

DataFrame является одной из основных структур данных в обеих библиотеках. Он представляет собой двумерную таблицу, состоящую из рядов и столбцов. DataFrame в Polars и Pandas предоставляют мощные средства для работы с данными, включая индексацию, фильтрацию, сортировку, группировку и объединение данных.

Однако есть несколько различий в представлении DataFrame в Polars и Pandas. В Polars DataFrame является неизменяемой структурой данных, что означает, что после создания DataFrame его содержимое нельзя изменить. Вместо этого, большинство операций над DataFrame в Polars возвращают новый DataFrame с примененными изменениями. В Pandas же DataFrame является изменяемой структурой данных, и его содержимое может быть изменено непосредственно.

Series

Series представляет собой одномерный массив данных, который используется для представления столбцов в DataFrame. Как и DataFrame, Series также имеет некоторые различия в представлении и манипуляции в Polars и Pandas.

В Polars Series является неизменяемой структурой данных, аналогично DataFrame. Операции над Series в Polars создают новые Series с примененными изменениями. В Pandas же Series является изменяемой структурой данных, и его содержимое может быть изменено напрямую.

Различия в манипуляции данными

Помимо различий в представлении данных, Polars и Pandas также имеют некоторые различия в синтаксисе и подходах к манипуляции данными.

Например, в Polars операции фильтрации, сортировки и группировки могут быть выполнены с использованием метода chaining (цепочки методов), что делает код более компактным и читаемым:

```
df.filter(pl.col("age") >
30).sort("name").groupby("gender").agg({"income":
"mean"})
```

В Pandas же эти операции выполняются с использованием отдельных функций:

```
df[df["age"] >
30].sort_values("name").groupby("gender").agg({"inc
ome": "mean"})
```

Присвоение значения по условию

В Polars, для присвоения значения столбцу по определенному условию, используется метод with_column, который возвращает новый DataFrame с примененными изменениями:

```
df = df.with_column(pl.when(pl.col("age") >
30).then("Old").otherwise("Young"), "AgeGroup")
```

В Pandas же это можно сделать с использованием прямого присвоения значения столбцу:

```
df.loc[df["age"] > 30, "AgeGroup"] = "Old"
df.loc[df["age"] <= 30, "AgeGroup"] = "Young"
```

Объединение данных по столбцам

В Polars, для объединения данных по столбцам, можно использовать метод hstack, который объединяет два DataFrame горизонтально:

```
df1 = pl.DataFrame({
    "A": [1, 2, 3],
    "B": ["a", "b", "c"]
})

df2 = pl.DataFrame({
```

```
    "C": [4, 5, 6],
    "D": ["d", "e", "f"]
})

result = df1.hstack(df2)
```

В Pandas, для объединения данных по столбцам, можно использовать функцию concat, указав ось axis=1:

```
df1 = pd.DataFrame({
    "A": [1, 2, 3],
    "B": ["a", "b", "c"]
})

df2 = pd.DataFrame({
    "C": [4, 5, 6],
    "D": ["d", "e", "f"]
})

result = pd.concat([df1, df2], axis=1)
```

Синтаксис и операции

В этом разделе сравнивается синтаксис и подходы к выполнению распространенных операций в библиотеках Polars и Pandas. Рассмотрим операции фильтрации, сортировки, группировки и объединения данных.

Фильтрация

Polars:

```
import polars as pl

df = pl.DataFrame({
    'A': [1, 2, 3, 4, 5],
```

```
    'B': ['a', 'b', 'c', 'd', 'e']
})

filtered = df.filter(pl.col('A') > 3)
```

Pandas:

```
import pandas as pd

df = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': ['a', 'b', 'c', 'd', 'e']
})

filtered = df[df['A'] > 3]
```

Обе библиотеки предоставляют возможность фильтрации данных на основе условий. В Polars используется метод filter() и передается условие в виде выражения, в то время как в Pandas используются квадратные скобки и передется условие в виде логического выражения.

Сортировка

Polars:

```
import polars as pl

df = pl.DataFrame({
    'A': [2, 1, 3, 5, 4],
    'B': ['b', 'a', 'c', 'e', 'd']
})

sorted_df = df.sort('A')
```

Pandas:

```
import pandas as pd

df = pd.DataFrame({
    'A': [2, 1, 3, 5, 4],
    'B': ['b', 'a', 'c', 'e', 'd']
})

sorted_df = df.sort_values('A')
```

И в Polars, и в Pandas можно отсортировать данные по одному или нескольким столбцам. В Polars используется метод sort() и передаем имя столбца, по которому необходимо выполнить сортировку, в то время как в Pandas используется метод sort_values() и передаем имя столбца или список столбцов.

Группировка

Polars:

```
import polars as pl

df = pl.DataFrame({
    'A': [1, 2, 1, 2, 1],
    'B': ['a', 'b', 'a', 'b', 'a'],
    'C': [10, 20, 30, 40, 50]
})

grouped = df.groupby('A').agg({'C': 'sum'})
```

Pandas:

```
import pandas as pd
```

```
df = pd.DataFrame({
    'A': [1, 2, 1, 2, 1],
    'B': ['a', 'b', 'a', 'b', 'a'],
    'C': [10, 20, 30, 40, 50]
})

grouped = df.groupby('A')['C'].sum()
```

Обе библиотеки предоставляют функциональность для группировки данных по значениям столбцов и агрегации результатов. В Polars используется метод groupby() и указываем столбец для группировки, а затем применяем агрегацию с помощью метода agg(). В Pandas также используется метод groupby(), а затем применяем агрегацию, указывая столбец и операцию.

Объединение

Polars:

```
import polars as pl

df1 = pl.DataFrame({
    'A': [1, 2, 3],
    'B': ['a', 'b', 'c']
})

df2 = pl.DataFrame({
    'A': [4, 5, 6],
    'B': ['d', 'e', 'f']
})

merged = df1.join(df2, on='A', how='inner')
```

Pandas:

```
import pandas as pd

df1 = pd.DataFrame({
    'A': [1, 2, 3],
    'B': ['a', 'b', 'c']
})

df2 = pd.DataFrame({
    'A': [4, 5, 6],
    'B': ['d', 'e', 'f']
})

merged = df1.merge(df2, on='A', how='inner')
```

Как и ожидалось, обе библиотеки позволяют объединять данные на основе ключевых столбцов. В Polars используется метод join() и указываем, по какому столбцу необходимо объединить данные, а также указываем тип объединения с помощью параметра how. В Pandas используем метод merge() и указываем столбец для объединения и тип объединения.

Производительность

В этом разделе сравним производительность библиотек Polars и Pandas на различных операциях и объемах данных. Также рассмотрим особенности и оптимизации, присутствующие в каждой из этих библиотек.

Обе библиотеки предоставляют мощные инструменты для работы с данными, однако их подходы к обработке и оптимизации данных имеют некоторые отличия, которые могут повлиять на производительность.

Особенности и оптимизации Polars

Polars предназначен для работы с большими объемами данных и быстрой обработки. Он использует многопоточность и векторизацию для эффективной обработки данных.

Некоторые из ключевых особенностей и оптимизаций Polars включают:

1. **Lazy вычисления**: Polars выполняет ленивые вычисления, что означает, что он откладывает фактическое выполнение операций до тех пор, пока это необходимо. Это позволяет Polars строить оптимальные планы выполнения операций и избегать ненужных вычислений.

2. **Векторизация**: Polars использует векторизованные операции, что позволяет выполнять операции над целыми массивами данных, а не над отдельными элементами. Это значительно повышает производительность при обработке больших объемов данных.

3. **Многопоточность**: Polars поддерживает выполнение операций в нескольких потоках, что позволяет параллельно обрабатывать данные и ускоряет вычисления.

Особенности и оптимизации Pandas

Pandas, с другой стороны, обладает широкой функциональностью и гибкостью при работе с данными. Он предоставляет широкий спектр операций и функций для манипуляции и анализа данных.

Некоторые из ключевых особенностей и оптимизаций Pandas включают:

1. **Индексы**: Pandas использует индексы для эффективного доступа к данным и быстрого выполнения операций, таких как фильтрация, сортировка и группировка. Использование индексов может значительно повысить производительность Pandas.

2. **Кэширование**: Pandas может кэшировать результаты некоторых операций, что позволяет избежать повторных вычислений при последующих обращениях к данным. Это особенно полезно при итеративных вычислениях или при повторном использовании промежуточных результатов.

3. **Оптимизированные операции**: Pandas предоставляет оптимизированные операции для выполнения распространенных задач, таких как фильтрация, сортировка, группировка и объединение. Эти операции реализованы на низком уровне для обеспечения максимальной производительности.

Важно отметить, что производительность Polars и Pandas может зависеть от конкретной операции, типа данных и объема данных. Рекомендуется проводить собственные тесты производительности для конкретных сценариев использования и требований.

Примеры использования

В данном разделе рассмотрим более сложные примеры использования библиотек Polars и Pandas для решения типичных задач анализа данных. Для каждого примера представлен код и результаты работы сравниваемых библиотек.

Пример 1: Объединение данных

Предположим, у нас есть два набора данных: sales_data с информацией о продажах товаров и customer_data с информацией о клиентах. Наша задача - объединить эти данные по общему столбцу customer_id (рис. 1.1 – 1.2).

```python
import polars as pl

# Создаем DataFrame с данными о продажах
sales_data = pl.DataFrame({
    'customer_id': [1, 2, 3, 4],
    'product': ['A', 'B', 'C', 'D'],
    'quantity': [10, 20, 15, 25]
})

# Создаем DataFrame с данными о клиентах
customer_data = pl.DataFrame({
    'customer_id': [1, 2, 3, 4],
    'name': ['John', 'Jane', 'Alice', 'Bob'],
    'age': [30, 25, 40, 35]
})

# Объединяем данные по столбцу customer_id
merged = sales_data.join(customer_data, on='customer_id')

# Выводим результат
print(merged)
```

```
shape: (4, 5)
```

customer_id	product	quantity	name	age
---	---	---	---	---
i64	str	i64	str	i64
1	A	10	John	30
2	B	20	Jane	25
3	C	15	Alice	40
4	D	25	Bob	35

Рисунок 1.1 – пример объединения данных с библиотекой Polars

```python
import pandas as pd

# Создаем DataFrame с данными о продажах
sales_data = pd.DataFrame({
    'customer_id': [1, 2, 3, 4],
    'product': ['A', 'B', 'C', 'D'],
    'quantity': [10, 20, 15, 25]
})

# Создаем DataFrame с данными о клиентах
customer_data = pd.DataFrame({
    'customer_id': [1, 2, 3, 4],
    'name': ['John', 'Jane', 'Alice', 'Bob'],
    'age': [30, 25, 40, 35]
})

# Объединяем данные по столбцу customer_id
merged = sales_data.merge(customer_data, on='customer_id')

# Выводим результат
print(merged)
```

```
   customer_id product  quantity   name  age
0            1       A        10   John   30
1            2       B        20   Jane   25
2            3       C        15  Alice   40
3            4       D        25    Bob   35
```

Рисунок 1.2 – пример объединения данных с библиотекой Pandas

Пример 2: Агрегация данных

Предположим, у нас есть набор данных о продажах разных товаров, и мы хотим посчитать общую сумму продаж и среднюю цену для каждого продукта (рис. 1.3 - 1.4).

```python
import polars as pl

# Создаем DataFrame с данными о продажах
df = pl.DataFrame({
    'product': ['A', 'B', 'A', 'B'],
    'sales': [100, 200, 150, 250],
    'price': [10, 20, 15, 25]
})

# Группируем данные по продукту и выполняем агрегацию
aggregated = df.groupby('product').agg({'sales': 'sum', 'price':

# Выводим результат
print(aggregated)
```

```
shape: (2, 3)
┌─────────┬────────────┬────────────┐
│ product ┆ sales      ┆ price      │
│ ---     ┆ ---        ┆ ---        │
│ str     ┆ list[i64]  ┆ list[i64]  │
╞═════════╪════════════╪════════════╡
│ B       ┆ [200, 250] ┆ [20, 25]   │
│ A       ┆ [100, 150] ┆ [10, 15]   │
└─────────┴────────────┴────────────┘
```

Рисунок 1.3 – пример агрегации данных с библиотекой Polars

```python
import pandas as pd

# Создаем DataFrame с данными о продажах
df = pd.DataFrame({
    'product': ['A', 'B', 'A', 'B'],
    'sales': [100, 200, 150, 250],
    'price': [10, 20, 15, 25]
})

# Группируем данные по продукту и выполняем агрегацию
aggregated = df.groupby('product').agg({'sales': 'sum', 'price':

# Выводим результат
print(aggregated)
```

```
         sales  price
product
A          250   12.5
B          450   22.5
```

Рисунок 1.4 – пример агрегации данных с библиотекой Pandas

Выводы

В ходе нашего исследования рассмотрели и сравнили библиотеки Polars и Pandas для работы с данными в Python. Обе библиотеки

предоставляют мощные инструменты для анализа и манипуляции данными, однако они имеют ряд различий, которые следует учитывать при выборе подходящей библиотеки в зависимости от конкретных потребностей проекта.

Различия между Polars и Pandas

1. **Производительность**: Polars обладает высокой производительностью благодаря использованию Rust для ядра библиотеки. Это позволяет ей эффективно обрабатывать большие объемы данных и выполнять операции с высокой скоростью. Pandas, в свою очередь, написан на Python, что может сказываться на производительности при работе с большими наборами данных.

2. **Поддержка типов данных**: Polars имеет строгую типизацию данных, что позволяет избежать ошибок типов при выполнении операций. Pandas, в свою очередь, предоставляет более гибкую работу с типами данных, позволяя использовать различные комбинации типов.

3. **Синтаксис и API**: Синтаксис и API в Polars и Pandas имеют некоторые различия. Polars предлагает удобные функции для выполнения операций над данными и имеет более компактный и интуитивно понятный синтаксис, который позволяет более легко читать и писать код. Pandas, с другой стороны, обладает более обширным набором функций и более привычным для многих пользователей синтаксисом.

Преимущества и недостатки

Polars:

- Преимущества:
  - Высокая производительность на больших объемах данных.
  - Строгая типизация данных, что позволяет избежать ошибок типов.
  - Удобный и интуитивно понятный синтаксис.
- Недостатки:
  - Не такой широкий набор функций и возможностей, как у Pandas.
  - Отсутствие полной совместимости со всеми функциями Pandas.

Pandas:

- Преимущества:
  - Большой набор функций и возможностей для работы с данными.
  - Широкая поддержка и активное сообщество пользователей.
  - Более привычный синтаксис для пользователей, знакомых с языком Python.
- Недостатки:
  - Производительность может быть ниже на больших объемах данных.
  - Большой размер памяти, требуемый для обработки больших наборов данных.

Рекомендации по выбору

При выборе между Polars и Pandas следует учитывать следующие факторы:

1. Если вам требуется работа с большими объемами данных и высокая производительность является приоритетом, то Polars может быть лучшим выбором.
2. Если вам важна гибкость работы с типами данных и большой набор функций для анализа данных, а производительность не является критическим фактором, то Pandas может быть предпочтительнее.
3. Если у вас уже есть опыт работы с Pandas и вы чувствуете себя комфортно с его синтаксисом и функциональностью, то вам может быть проще остаться на нем и не переходить на Polars, если нет четкой необходимости.

Итак, выбор между Polars и Pandas зависит от ваших конкретных потребностей, приоритетов и ожидаемой производительности. Обе библиотеки предоставляют мощные инструменты для работы с данными, и правильный выбор поможет вам эффективно проводить анализ данных и извлекать ценную информацию.

Список используемой литературы:

1. Официальная документация Polars: https://docs.polars.rs/
2. Репозиторий Polars на GitHub: https://github.com/pola-rs/polars
3. Официальная документация Pandas: https://pandas.pydata.org/docs/
4. Репозиторий Pandas на GitHub: https://github.com/pandas-dev/pandas
5. Статья "Polars: A Blazingly Fast DataFrame Library for Python and Rust" на Towards Data Science: https://towardsdatascience.com/polars-a-blazingly-fast-dataframe-library-for-python-and-rust-b38cc0ac8ee9
6. Статья "Polars: A Blazing Fast DataFrame Library for Python" на Real Python: https://realpython.com/polacrs-python-dataframe/
7. Официальный блог Pandas: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html#user-guide