**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

|  |  |
| --- | --- |
| **A logo for a university  Description automatically generated** |  |

|  |  |  |
| --- | --- | --- |
| **HUỲNH HOÀNG PHÚC** | **21036541** | **DHKTPM17A** |
| **PHẠM ĐĂNG KHÔI** | **21042951** | **DHKTPM17B** |

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**LỚP: NHẬP MÔN DỮ LIỆU LỚN**

**MAINTAINING YOUR DELTA LAKE**

*TP. Hồ Chí Minh, tháng 12 năm 2024*

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

|  |  |
| --- | --- |
| **A logo for a university  Description automatically generated** |  |

|  |  |  |
| --- | --- | --- |
| **HUỲNH HOÀNG PHÚC** | **21036541** | **DHKTPM17A** |
| **PHẠM ĐĂNG KHÔI** | **21042951** | **DHKTPM17B** |

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**LỚP: NHẬP MÔN DỮ LIỆU LỚN**

**MAINTAINING YOUR DELTA LAKE**

*TP. Hồ Chí Minh, tháng 12 năm 2024*

**LỜI CẢM ƠN**

Chúng em xin chân thành cảm ơn TS. Nguyễn Chí Kiên đã luôn tận tâm hướng dẫn, chia sẻ kiến thức và cung cấp sự hỗ trợ quý báu trong suốt quá trình thực hiện đồ án này. Những lời khuyên, ý tưởng và kiến thức mà thầy truyền đạt đã giúp chúng em vượt qua những khó khăn và hoàn thành công việc một cách hiệu quả hơn.

Bên cạnh đó, chúng em cũng muốn gửi lời cảm ơn đến tất cả những người đã đóng góp ý kiến, sáng kiến và hỗ trợ trong dự án này. Sự hợp tác, tinh thần đoàn kết và cống hiến của mọi người đã làm cho dự án trở nên thành công và đáng tự hào.

Dự án này không chỉ mang lại cho chúng em những kiến thức và kỹ năng quan trọng, mà còn là một trải nghiệm quý giá. Chúng em hy vọng rằng kết quả của nó sẽ có giá trị và hữu ích đối với tất cả mọi người, giống như nó đã mang lại cho chúng em.

Một lần nữa, chúng em xin gửi lời cảm ơn chân thành đến tất cả những ai đã giúp đỡ chúng em trong quá trình thực hiện dự án này.

Trân trọng!

**PHẦN ĐÁNH GIÁ CỦA GIẢNG VIÊN**

Tp. Hồ Chí Minh,tháng 12 năm 2024

(kí và ghi họ tên)

**MỤC LỤC**

[**NỘI DUNG** 1](#_Toc184293257)

[1. Giới thiệu chung về Delta Lake 1](#_Toc184293258)

[1.1. Khái niệm Delta Lake 1](#_Toc184293259)

[1.2. Lịch sử phát triển của Delta Lake 1](#_Toc184293260)

[1.3. Lợi ích của việc sử dụng Delta Lake so với các hệ thống lưu trữ dữ liệu khác. 2](#_Toc184293261)

[2. Sử dụng các thuộc tính bảng trong Delta Lake 3](#_Toc184293262)

[2.1. Các thuộc tính của bảng Delta Lake 3](#_Toc184293263)

[2.2. Tạo bảng trống với thuộc tính 4](#_Toc184293264)

[2.3. Thêm, thay đổi hoặc bổ sung thuộc tính bảng 8](#_Toc184293265)

[2.4. Xóa bỏ thuộc tính bảng 10](#_Toc184293266)

[2.5. Các thuộc tính mặc định của bảng trong Delta Lake 11](#_Toc184293267)

[3. Tối ưu hóa bảng Delta Lake 12](#_Toc184293268)

[3.1. Vấn đề của bảng lớn và các tệp nhỏ 12](#_Toc184293269)

[3.2. Sử dụng OPTIMIZE để khắc phục vấn đề tệp nhỏ 15](#_Toc184293270)

[3.3. Áp dụng Z-Order để tối ưu hóa truy vấn 16](#_Toc184293273)

[4. Điều chỉnh và quản lý bảng 17](#_Toc184293274)

[4.1. Phân vùng bảng 17](#_Toc184293275)

[4.2. Xác định phân vùng khi tạo bảng 18](#_Toc184293276)

[4.3. Chuyển đổi từ bảng không phân vùng sang bảng phân vùng 19](#_Toc184293277)

[5. Sửa chữa, khôi phục và thay thế dữ liệu bảng 20](#_Toc184293278)

[5.1. Khôi phục và thay thế bảng 20](#_Toc184293279)

[5.2. Xóa dữ liệu và loại bỏ phân vùng 21](#_Toc184293280)

[5.3. Vòng đời của bảng Delta Lake 22](#_Toc184293281)

[5.4. Khôi phục bảng 22](#_Toc184293282)

[5.5. Làm sạch dữ liệu 22](#_Toc184293283)

[5.5.1. Lệnh vacuum 23](#_Toc184293284)

[5.5.2. Xóa hoàn toàn dấu vết của bảng 23](#_Toc184293286)

[6. Tích hợp Delta Lake với các công cụ khác 24](#_Toc184293287)

[6.1. Tích hợp Delta Lake với Apache Spark 24](#_Toc184293288)

[6.2. Tích hợp Delta Lake với công cụ BI 25](#_Toc184293289)

[6.3. Tích hợp Delta Lake với Machine Learning 26](#_Toc184293290)

[7. Delta Lake với Machine Learning 27](#_Toc184293291)

[7.1. Xây dựng và huấn luyện mô hình hồi quy tuyến tính (Linear Regression) trên dữ liệu Delta Lake bằng Apache Spark 27](#_Toc184293292)

[7.2. Đánh giá Mô hình Hồi quy Tuyến tính bằng RMSE và R-squared 29](#_Toc184293293)

[7.3. Tuning mô hình hồi quy tuyến tính với Cross-Validation và tìm tham số tối ưu 30](#_Toc184293294)

[7.4. Đánh giá mô hình hồi quy tuyến tính trên dữ liệu Delta Lake với biểu đồ phân tán và sai số 31](#_Toc184293295)

[7.5. Phân tích mô hình hồi quy tuyến tính với dữ liệu Delta Lake: So sánh giá trị thực tế và dự đoán 33](#_Toc184293296)

[7.6. Trực quan hóa quá trình xử lý dữ liệu trong Delta Lake (pipeline steps) 37](#_Toc184293297)

[7.7. Hiệu suất mô hình qua các phiên bản dữ liệu Delta Lake 39](#_Toc184293298)

[8. Kết luận 41](#_Toc184293299)

[8.1. Ứng dụng thực tế của Delta Lake trong các ngành 41](#_Toc184293300)

[8.2. Tầm quan trọng của việc duy trì và tối ưu hóa Delta Lake 42](#_Toc184293301)

[8.3. Hướng phát triển của Delta Lake trong tương lai 42](#_Toc184293302)

[TÀI LIỆU THAM KHẢO 44](#_Toc184293303)

# **NỘI DUNG**

1. Giới thiệu chung về Delta Lake

1.1. Khái niệm Delta Lake

Delta Lake là một định dạng bảng nguồn mở cung cấp tính nhất quán trong giao dịch và tăng quy mô cho các tập dữ liệu bằng cách tạo ra một định nghĩa nhất quán về các tập dữ liệu và bao gồm các thay đổi tiến hóa lược đồ và đột biến dữ liệu. Với Delta Lake, các bản cập nhật cho bộ dữ liệu được xem một cách nhất quán trên mọi ứng dụng sử dụng bộ dữ liệu và người dùng sẽ không nhìn thấy chế độ xem dữ liệu không nhất quán trong quá trình chuyển đổi. Điều này tạo ra một cái nhìn nhất quán và đáng tin cậy về các tập dữ liệu trong hồ dữ liệu khi chúng được cập nhật và phát triển. Tính nhất quán của dữ liệu được bật bằng cách tạo một loạt tệp kê khai xác định lược đồ và dữ liệu cho một thời điểm nhất định cũng như nhật ký giao dịch xác định bản ghi theo thứ tự của mọi giao dịch trên tập dữ liệu. Bằng cách đọc nhật ký giao dịch và tệp kê khai, các ứng dụng được đảm bảo nhìn thấy chế độ xem dữ liệu nhất quán tại bất kỳ thời điểm nào và người dùng có thể đảm bảo các thay đổi trung gian không hiển thị cho đến khi thao tác ghi hoàn tất.

1.2. Lịch sử phát triển của Delta Lake

Lịch sử phát triển của Delta Lake bắt đầu từ năm 2017 khi Databricks, công ty sáng lập Apache Spark, nhận thấy các thách thức trong việc quản lý và xử lý dữ liệu trong các hệ thống data lakes. Mặc dù data lakes cho phép lưu trữ lượng lớn dữ liệu không cấu trúc, nhưng lại gặp phải các vấn đề về tính toàn vẹn, khả năng truy vấn và xử lý dữ liệu hiệu quả.

Delta Lake được ra đời để giải quyết những vấn đề này. Vào tháng 4 năm 2019, Delta Lake được công bố là một dự án mã nguồn mở, được xây dựng trên nền tảng Apache Spark và sử dụng Apache Parquet làm định dạng lưu trữ dữ liệu. Mục tiêu của Delta Lake là cung cấp một lớp quản lý dữ liệu có thể đảm bảo tính toàn vẹn và hỗ trợ các giao dịch ACID (Atomicity, Consistency, Isolation, Durability), điều mà các hệ thống lưu trữ dữ liệu truyền thống không làm được trong môi trường dữ liệu phân tán.

Với các tính năng nổi bật như Time Travel, ACID Transactions, và Schema Evolution, Delta Lake giúp cải thiện khả năng xử lý dữ liệu lớn trong các môi trường phân tán, đồng thời đảm bảo tính chính xác và linh hoạt trong việc lưu trữ và truy vấn dữ liệu. Sau khi trở thành dự án mã nguồn mở, Delta Lake nhanh chóng được cộng đồng công nghệ đón nhận và ngày càng trở nên phổ biến trong các ứng dụng dữ liệu lớn, đặc biệt là trong các môi trường sử dụng Apache Spark.

Tính đến nay, Delta Lake đã trở thành một trong những công cụ quan trọng trong việc xây dựng và quản lý data lakes hiện đại, đồng thời là một phần không thể thiếu trong chiến lược quản lý và xử lý dữ liệu của nhiều tổ chức và doanh nghiệp.

1.3. Lợi ích của việc sử dụng Delta Lake so với các hệ thống lưu trữ dữ liệu khác.

Việc sử dụng Delta Lake mang lại nhiều lợi ích vượt trội so với các hệ thống lưu trữ dữ liệu truyền thống và các giải pháp data lake khác. Một số lợi ích chính bao gồm:

* ACID Transactions: Delta Lake hỗ trợ các giao dịch ACID (Atomicity, Consistency, Isolation, Durability), đảm bảo tính toàn vẹn và nhất quán của dữ liệu khi thực hiện các thao tác như ghi, đọc và cập nhật. Điều này giải quyết được vấn đề của các data lake truyền thống, nơi không có cơ chế bảo vệ dữ liệu khỏi sự mất mát hay xung đột khi có nhiều người dùng truy cập đồng thời.
* Time Travel: Delta Lake cho phép truy vấn dữ liệu ở các phiên bản trước thông qua tính năng Time Travel. Điều này cực kỳ hữu ích trong các tình huống cần khôi phục dữ liệu bị lỗi hoặc tìm kiếm lịch sử thay đổi dữ liệu mà không làm gián đoạn quá trình hoạt động hiện tại.
* Schema Enforcement & Evolution: Delta Lake giúp kiểm soát và phát triển lược đồ dữ liệu một cách linh hoạt. Schema Enforcement đảm bảo rằng dữ liệu tuân thủ đúng lược đồ đã định nghĩa, trong khi Schema Evolution cho phép thay đổi cấu trúc dữ liệu mà không làm gián đoạn hệ thống.
* Tính linh hoạt và khả năng mở rộng: Delta Lake có thể dễ dàng mở rộng để xử lý hàng petabyte dữ liệu và tích hợp với các công cụ phân tích dữ liệu như Apache Spark và Databricks, giúp xử lý dữ liệu lớn với hiệu suất cao.
* Hiệu quả truy vấn và tối ưu hóa: Delta Lake hỗ trợ các phương pháp tối ưu hóa dữ liệu như OPTIMIZE và Z-Ordering, giúp cải thiện tốc độ truy vấn và giảm chi phí đọc dữ liệu, đặc biệt là trong các môi trường dữ liệu lớn.
* Tính bảo mật và quản lý quyền truy cập: Delta Lake tích hợp tốt với các công cụ quản lý quyền truy cập và bảo mật, đảm bảo rằng dữ liệu chỉ có thể được truy cập bởi những người dùng hoặc ứng dụng có quyền, giúp tăng cường bảo mật dữ liệu.
* Tiết kiệm chi phí và bảo trì: Delta Lake giảm thiểu chi phí lưu trữ và bảo trì bằng cách tự động gộp các tệp nhỏ và loại bỏ dữ liệu không còn sử dụng thông qua tính năng VACUUM, đồng thời giúp duy trì hiệu suất hệ thống trong khi giảm thiểu việc sử dụng tài nguyên.

Nhờ vào những lợi ích trên, Delta Lake không chỉ giúp cải thiện hiệu suất và độ tin cậy trong việc xử lý dữ liệu lớn mà còn tạo ra một nền tảng mạnh mẽ cho các tổ chức và doanh nghiệp trong việc xây dựng các hệ thống dữ liệu phân tán và phân tích dữ liệu quy mô lớn.

2. Sử dụng các thuộc tính bảng trong Delta Lake

2.1. Các thuộc tính của bảng Delta Lake

Trong Delta Lake, các **thuộc tính bảng** đóng vai trò quan trọng trong việc cấu hình và tối ưu hóa các bảng dữ liệu. Những thuộc tính này giúp điều chỉnh các khía cạnh hoạt động của bảng, như hiệu suất, cách lưu trữ dữ liệu và cách quản lý lịch sử bảng. Dưới đây là một số thuộc tính bảng quan trọng trong Delta Lake:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Data Type** | **Use With** | **Default** |
| **delta.logRetentionDuration** | **CalendarInterval** | **Cleaning** | **interval 30 days** |
| **delta.deletedFileRetentionDuration** | **CalendarInterval** | **Cleaning** | **interval 1 week** |
| **delta.setTransactionRetentionDuration** | **CalendarInterval** | **Cleaning, Repairing** | **(none)** |
| **delta.targetFileSize\*** | **String** | **Tuning** | **(none)** |
| **delta.tuneFileSizesForRewrites\*** | **Boolean** | **Tuning** | **(none)** |
| **delta.autoOptimize.optimizeWrite\*** | **Boolean** | **Tuning** | **(none)** |
| **delta.autoOptimize.autoCompact\*** | **Boolean** | **Tuning** | **(none)** |
| **delta.dataSkippingNumIndexedCols** | **Int** | **Tuning** | **32** |
| **delta.checkpoint.writeStatsAsStruct** | **Boolean** | **Tuning** | **(none)** |
| **delta.checkpoint.writeStatsAsJson** | **Boolean** | **Tuning** | **true** |

2.2. Tạo bảng trống với thuộc tính

Trong ví dụ này, chúng ta sẽ tạo một bảng Delta Lake trống, sử dụng cú pháp SQL để định nghĩa bảng với một cột ngày và áp dụng một thuộc tính bảng mặc định, cụ thể là delta.logRetentionDuration. Thuộc tính này sẽ giúp quản lý thời gian lưu trữ các log giao dịch, điều này sẽ được giải thích thêm trong các phần sau của chương.

Dưới đây là cú pháp SQL để tạo bảng trống với thuộc tính mặc định:



Bảng này hiện tại chỉ có một cột (date) và không chứa dữ liệu. Khi bạn tạo bảng này, Delta Lake sẽ tạo một thư mục \_delta\_log trong thư mục của bảng, nơi lưu trữ các log giao dịch ban đầu và siêu dữ liệu của bảng trống. Nếu muốn kiểm tra bảng sau khi tạo, bạn có thể sử dụng lệnh sau để hiển thị các tệp sao lưu của bảng:

****

Lệnh này sẽ trả về một danh sách trống vì bảng hiện tại chưa có dữ liệu. Tuy nhiên, bạn có thể thêm dữ liệu vào bảng này trong các bước tiếp theo.

Sau khi nhập dữ liệu vào bảng thì sẽ có kết quả:



**Tạo bảng và hợp nhất dữ liệu COVID-19 vào bảng Delta**

Trong ví dụ này, chúng ta sẽ đọc dữ liệu từ bộ dữ liệu COVID-19 (định dạng Parquet) và hợp nhất vào bảng Delta Lake default.covid\_nyt đã tạo trước đó. Bộ dữ liệu này có một cột date được lưu trữ dưới dạng STRING. Chúng ta sẽ chuyển cột này sang kiểu DATE để phù hợp với kiểu dữ liệu của bảng Delta.

**Bước 1: Đọc và chuyển đổi dữ liệu**



**Lỗi gặp phải: AnalysisException: Table default.covid\_nyt already exists**

Khi thực hiện thao tác ghi dữ liệu vào bảng, bạn sẽ gặp lỗi AnalysisException thông báo rằng bảng default.covid\_nyt đã tồn tại. Điều này xảy ra do mặc định trong Spark, nếu bảng đã tồn tại thì sẽ phát sinh lỗi thay vì ghi đè lên bảng.

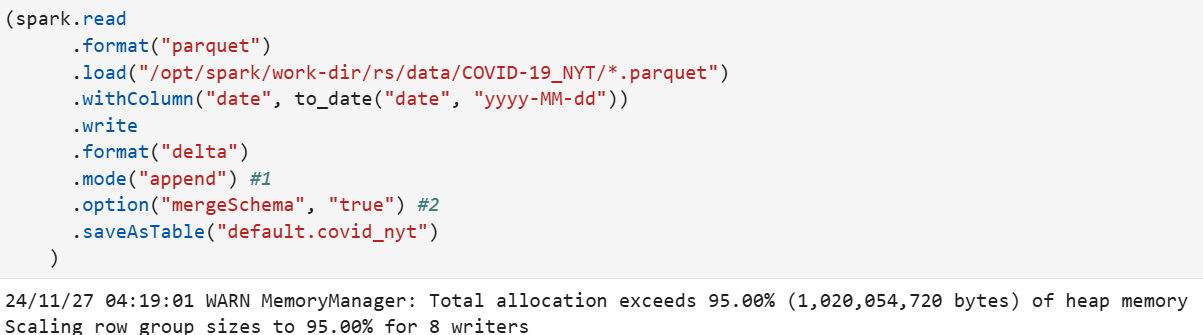
**Giải pháp:** Chúng ta cần thay đổi chế độ ghi của thao tác thành append để thêm dữ liệu vào bảng hiện có thay vì ghi đè.

**Bước 2: Sử dụng chế độ append để thêm dữ liệu**



**Lỗi gặp phải tiếp theo: AnalysisException: A schema mismatch detected…**

Sau khi khắc phục lỗi về bảng đã tồn tại, chúng ta sẽ gặp phải một lỗi khác liên quan đến **không khớp lược đồ**. Lỗi này xảy ra vì bảng default.covid\_nyt đã được tạo với một cột duy nhất (cột date), trong khi dữ liệu Parquet mà chúng ta đang ghi vào có sáu cột khác nhau. Delta Lake áp dụng **schema enforcement** để bảo vệ bảng khỏi việc thay đổi không mong muốn khi có sự khác biệt giữa lược đồ của bảng và lược đồ của dữ liệu đầu vào.

****

Sau khi thực hiện thao tác ghi, bạn có thể kiểm tra **metadata** của bảng để xác nhận các thay đổi đã được thực hiện thành công. Dùng câu lệnh DESCRIBE để xem chi tiết bảng:

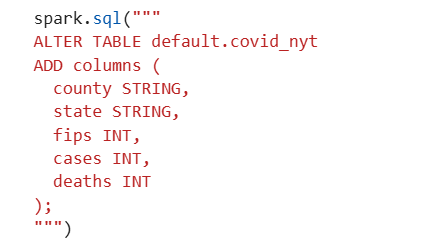


**Alternatives to Automatic Schema Evolution**

Trong Delta Lake, bạn có thể sử dụng Schema Evolution tự động với tùy chọn .option("mergeSchema", "true") để tự động thay đổi lược đồ của bảng khi dữ liệu đầu vào có sự khác biệt. Tuy nhiên, điều này có thể dẫn đến việc thay đổi lược đồ không mong muốn nếu các cột không xác định được thêm vào từ nguồn dữ liệu bên ngoài.

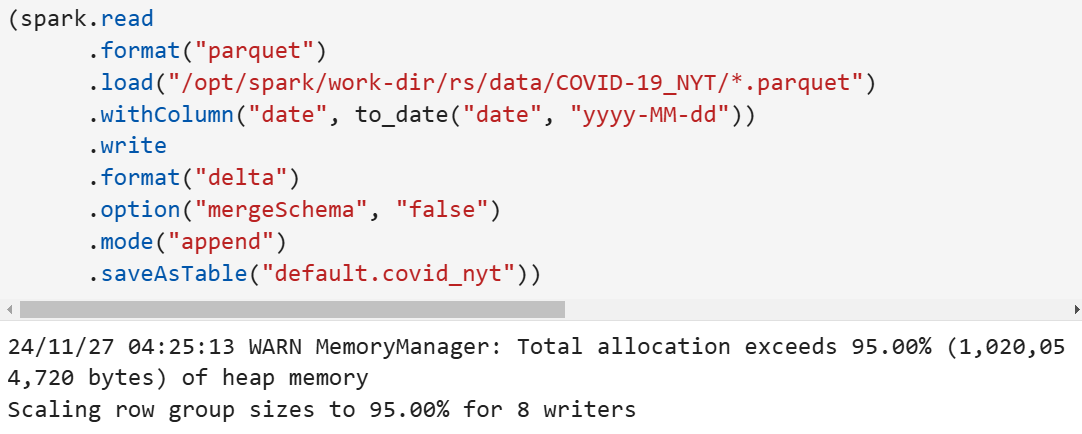
**Giải pháp 1: Thêm cột một cách rõ ràng**

Thay vì sử dụng mergeSchema, bạn có thể thêm các cột thiếu vào bảng bằng câu lệnh ALTER TABLE để kiểm soát rõ ràng các thay đổi:



**Giải pháp 2: Tắt Schema Evolution tự động**

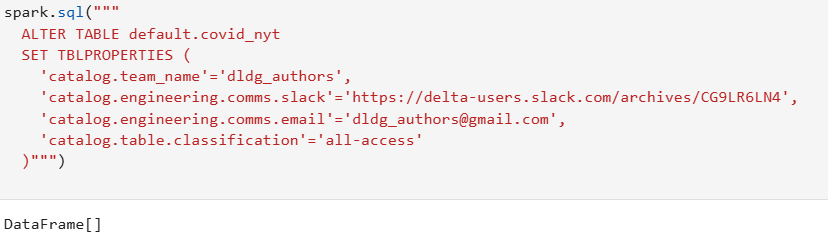
Nếu bạn muốn tắt việc thay đổi lược đồ tự động, bạn có thể đặt mergeSchema thành false khi ghi dữ liệu:



2.3. Thêm, thay đổi hoặc bổ sung thuộc tính bảng

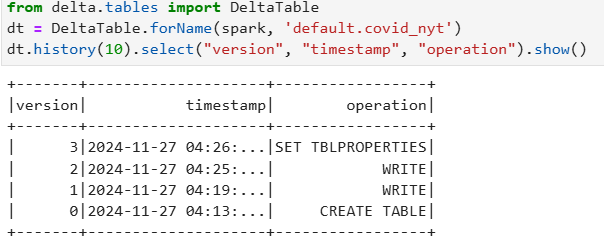
Quá trình thêm hoặc thay đổi thuộc tính bảng trong Delta Lake rất đơn giản. Nếu thuộc tính đã tồn tại, các thay đổi sẽ ghi đè lên thuộc tính hiện có mà không có cảnh báo. Nếu thuộc tính mới được thêm vào, nó sẽ được bổ sung vào tập hợp các thuộc tính của bảng.

Giả sử bạn muốn thêm hai thuộc tính mới vào bảng default.covid\_nyc, một thuộc tính lưu tên nhóm và một thuộc tính chứa thông tin Slack của nhóm. Để thực hiện điều này, bạn có thể sử dụng câu lệnh **ALTER TABLE** sau:

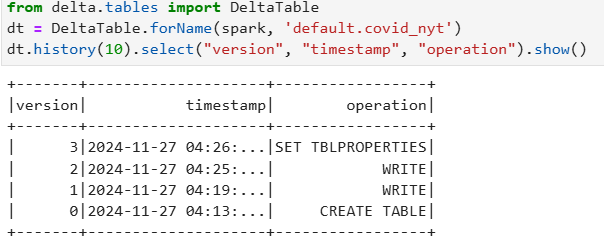


**Xem lịch sử thay đổi thuộc tính bảng**

Mỗi khi bạn thay đổi thuộc tính bảng, các thay đổi này sẽ được ghi lại trong lịch sử bảng. Để xem lịch sử thay đổi, bạn có thể sử dụng phương thức history() từ DeltaTable:

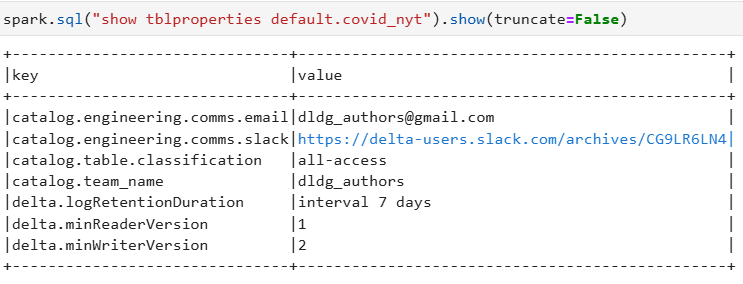


Lệnh trên sẽ hiển thị lịch sử thay đổi bảng, bao gồm phiên bản, thời gian thay đổi và loại thao tác thực hiện:

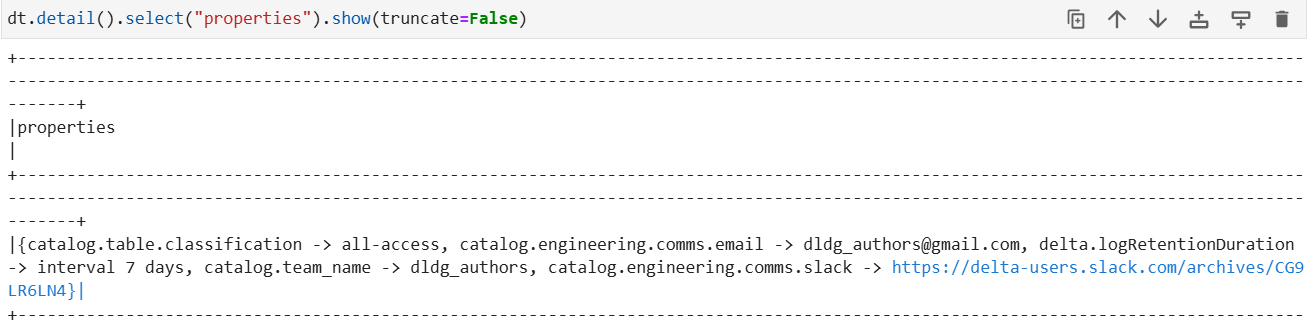


**Xem các thuộc tính hiện tại của bảng**

Để xem hoặc xác nhận các thuộc tính đã thay đổi, bạn có thể sử dụng lệnh SHOW TBLPROPERTIES trên bảng Delta:



Hoặc bạn có thể gọi phương thức **detail()** trên đối tượng DeltaTable để hiển thị thông tin chi tiết về bảng:



2.4. Xóa bỏ thuộc tính bảng

Khi làm việc với Delta Lake, bạn có thể gặp phải tình huống cần **xóa bỏ các thuộc tính bảng** không mong muốn hoặc sai chính tả. Để thực hiện điều này, bạn có thể sử dụng câu lệnh **ALTER TABLE ... UNSET TBLPROPERTIES**.

Giả sử bạn đã vô tình gõ sai một thuộc tính, ví dụ như **delta.loRgetentionDuratio** thay vì **delta.logRetentionDuration**. Dù sai sót này không gây ảnh hưởng nghiêm trọng, nhưng không có lý do gì để giữ thuộc tính sai này trong bảng.



Sau khi thực hiện lệnh trên, thuộc tính **delta.logRetentionDuration** sẽ không còn tồn tại trong bảng, giúp bảng của bạn trở nên sạch sẽ và không bị chiếm dụng bởi các thuộc tính không mong muốn.

2.5. Các thuộc tính mặc định của bảng trong Delta Lake

Bạn có thể thiết lập **thuộc tính bảng mặc định** cho Delta Lake bằng cách sử dụng cấu hình Spark, cụ thể là:

spark.databricks.delta.properties.defaults.<conf>

Mặc dù điều này chỉ hiệu quả với khối lượng công việc Spark, bạn có thể hình dung ra nhiều tình huống mà khả năng tự động đưa thuộc tính vào đường ống của bạn có thể hữu ích:

spark.databricks.delta.properties.defaults.logRetentionDuration=interval 2 weeks

spark.databricks.delta.properties.defaults.deletedFileRetentionDuration=interval 28 days

Việc thiết lập các thuộc tính mặc định có thể hữu ích trong các kịch bản tự động hóa hoặc pipeline dữ liệu, giúp đảm bảo rằng các bảng mới tạo ra có cùng các thuộc tính cấu hình mà không cần phải chỉ định lại mỗi lần.

**Thuộc tính bảng** có thể lưu trữ nhiều loại metadata hữu ích, chẳng hạn như:

* Tên nhóm chịu trách nhiệm về bảng (catalog.team\_name).
* Kênh Slack hoặc email của nhóm kỹ thuật (catalog.engineering.comms.slack hoặc catalog.engineering.comms.email).
* Phân loại bảng (như bảng nhạy cảm hoặc thông tin cá nhân - PII) với catalog.table.classification.

Metadata bảng giúp **khám phá dữ liệu** dễ dàng hơn và cung cấp thông tin về chủ sở hữu và những người chịu trách nhiệm cho dữ liệu, đồng thời hỗ trợ việc **quản lý dữ liệu** và **quyền truy cập dựa trên vai trò**.

Bảng sau mô tả một số thuộc tính bảng có thể sử dụng để nâng cao khả năng quản lý dữ liệu trong Delta Lake:

|  |  |
| --- | --- |
| **Thuộc tính** | **Mô tả** |
| catalog.team\_name | Cung cấp tên nhóm và trả lời câu hỏi "Ai chịu trách nhiệm về bảng?" |
| catalog.engineering.comms.slack | Cung cấp kênh Slack cho nhóm kỹ thuật. |
| catalog.engineering.comms.email | Cung cấp địa chỉ email của nhóm kỹ thuật. |
| catalog.table.classification | Xác định loại bảng (ví dụ: PII, sensitive-pii, general). |

3. Tối ưu hóa bảng Delta Lake

3.1. Vấn đề của bảng lớn và các tệp nhỏ

Khi chúng ta nói về vấn đề **các tệp nhỏ**, chúng ta đang đề cập đến một vấn đề không chỉ riêng Delta Lake mà là một vấn đề phổ biến trong các hệ thống lưu trữ dữ liệu phân tán, đặc biệt liên quan đến **I/O mạng** và chi phí mở các tệp khi bảng chứa quá nhiều tệp nhỏ. Các **tệp nhỏ** được định nghĩa là các tệp có kích thước dưới **64 KB**.

**Làm thế nào quá nhiều tệp nhỏ lại gây hại?**

Quá nhiều tệp nhỏ có thể gây ra vấn đề theo nhiều cách khác nhau. Tuy nhiên, điểm chung của tất cả các vấn đề này là chúng dần dần ảnh hưởng đến hiệu suất của hệ thống và yêu cầu phải thay đổi cấu trúc các tệp vật lý trong bảng. Nếu không nhận ra sự suy giảm hiệu suất khi các bảng bắt đầu trở nên nặng nề do có quá nhiều tệp nhỏ, chúng ta có thể sẽ phải tăng cường tính toán phân tán để mở và thực thi truy vấn một cách hiệu quả.

**Một chiến lược** để đảm bảo bảng của chúng ta luôn trong tình trạng tốt là sử dụng **giám sát bảng cấp độ**. Giám sát này có thể được thực hiện thông qua các chiến lược theo dõi metadata, chẳng hạn như theo dõi số lượng tệp trong ảnh chụp bảng hiện tại hoặc theo dõi bao nhiêu phiên bản của bảng còn lại trên đĩa. Việc giám sát giúp phát hiện các vấn đề một cách nhanh chóng và là công cụ hữu ích trong chiến lược bảo trì.

**Chi phí thực tế khi có quá nhiều tệp nhỏ**

Việc có quá nhiều tệp nhỏ trong bảng gây ảnh hưởng lớn đến hiệu suất và chi phí I/O. Mỗi tệp nhỏ sẽ yêu cầu một bước tính toán để mở tệp đó, và số bước này sẽ tăng lên theo thời gian, dẫn đến chi phí cao hơn khi làm việc với các tệp này.

Lấy ví dụ, giả sử bảng có một triệu tệp, mỗi tệp có kích thước 1 GB, và kích thước khối là 64 MB. Khi đó, để đọc toàn bộ bảng, hệ thống sẽ cần phân phối khoảng 15,65 triệu nhiệm vụ để đọc tất cả các tệp.

Mục tiêu là tối ưu hóa kích thước tệp vật lý trong bảng để giảm thiểu I/O hệ thống và I/O mạng. Khi gặp phải các tệp không tối ưu (vấn đề tệp nhỏ), hiệu suất của bảng sẽ giảm đi rất nhiều.

**Ví dụ về vấn đề các tệp nhỏ**: Giả sử bảng có kích thước ~1 TB, nhưng các tệp trong bảng lại nhỏ (5 KB mỗi tệp). Điều này có thể dẫn đến việc bạn có khoảng **200 triệu tệp** cần mở trước khi tải toàn bộ bảng vào bộ nhớ. Trong hầu hết các trường hợp, bảng sẽ không thể mở được.

**Tạo vấn đề tệp nhỏ**

Bây giờ, chúng ta sẽ tạo ra một vấn đề các tệp nhỏ thực tế và sau đó tối ưu hóa bảng để cải thiện hiệu suất.

**Bảng covid\_nyt** có hơn một triệu bản ghi và tổng kích thước của bảng là dưới 7 MB, được phân thành 8 phân vùng.

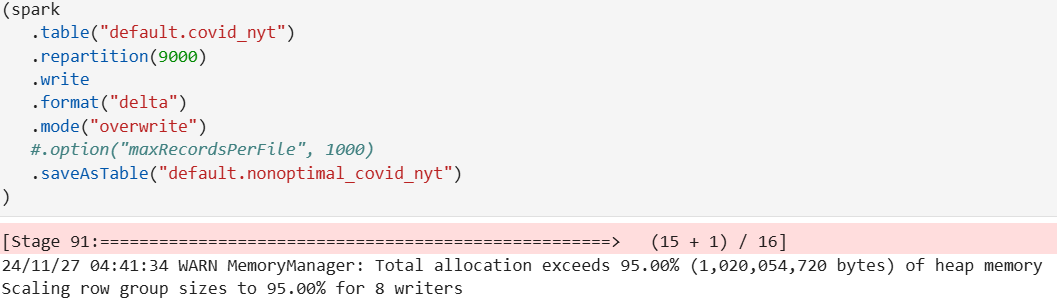


**Tạo bảng không tối ưu và gây ra vấn đề tệp nhỏ**

Giả sử bạn muốn tạo một bảng mới có tên default.nonoptimal\_covid\_nyt và cố ý tạo ra một bảng không tối ưu để mô phỏng vấn đề các tệp nhỏ. Để làm điều này, bạn có thể sử dụng các câu lệnh sau để tạo bảng và thêm các thuộc tính vào bảng:

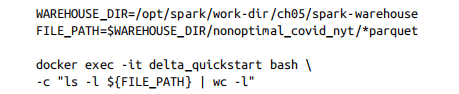


Nếu chúng ta thay đổi số lượng phân vùng từ 8 phân vùng thành 9.000 phân vùng, bảng sẽ bị chia thành **9.000 tệp nhỏ**, mỗi tệp có kích thước khoảng **5 KB**:



**Kiểm tra các tệp vật lý**

Bạn có thể kiểm tra số lượng tệp vật lý trong bảng bằng cách sử dụng lệnh sau:



Kết quả sẽ cho thấy rằng bảng hiện tại có **9.000 tệp nhỏ**.

3.2. Sử dụng OPTIMIZE để khắc phục vấn đề tệp nhỏ

**OPTIMIZE** là lệnh trong Delta Lake giúp giải quyết vấn đề các tệp nhỏ bằng cách **coalesce** nhiều tệp nhỏ thành những tệp lớn hơn. Quá trình này được gọi là **bin-packing**, nơi nhiều tệp nhỏ được gom lại thành những tệp lớn hơn để tối ưu hóa I/O và hiệu suất truy vấn.

**Bin-Packing trong OPTIMIZE**

**Bin-packing** là một kỹ thuật trong đó nhiều tệp nhỏ được gộp lại thành một tệp lớn hơn. Mỗi "bin" được định nghĩa là một tệp với kích thước tối đa (mặc định là 1 GB trong Delta Lake Spark, và 250 MB đối với Delta Rust).

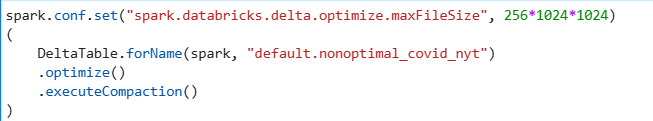
**Các cấu hình Tuning cho OPTIMIZE**

Delta Lake cung cấp các cấu hình để tùy chỉnh cách lệnh **OPTIMIZE** hoạt động, bao gồm:

* **spark.databricks.delta.optimize.minFileSize** (Spark Only):  
  Đây là ngưỡng kích thước tệp nhỏ hơn để nhóm các tệp trước khi viết lại thành một tệp lớn hơn. **Ví dụ**: Nếu một tệp nhỏ hơn ngưỡng này, nó sẽ được gom lại với các tệp nhỏ khác và ghi lại thành một tệp lớn hơn.
* **spark.databricks.delta.optimize.maxFileSize** (Spark Only):  
  Dùng để chỉ định kích thước tệp mục tiêu khi thực hiện lệnh **OPTIMIZE**. **Ví dụ**: Nếu bạn muốn tệp lớn nhất sau khi tối ưu hóa có kích thước khoảng 500 MB, bạn có thể cấu hình thông số này.
* **spark.databricks.delta.optimize.repartition.enabled** (Spark Only):  
  Thay đổi hành vi của lệnh **OPTIMIZE**, sẽ sử dụng **repartition(1)** thay vì **coalesce(1)** khi giảm số lượng tệp.
* **delta.targetFileSize** (delta-rs):  
  Thuộc tính bảng trong Delta Rust, dùng để chỉ định kích thước tệp mục tiêu cho các tệp mới được tối ưu hóa. **Ví dụ**: delta.targetFileSize=250mb

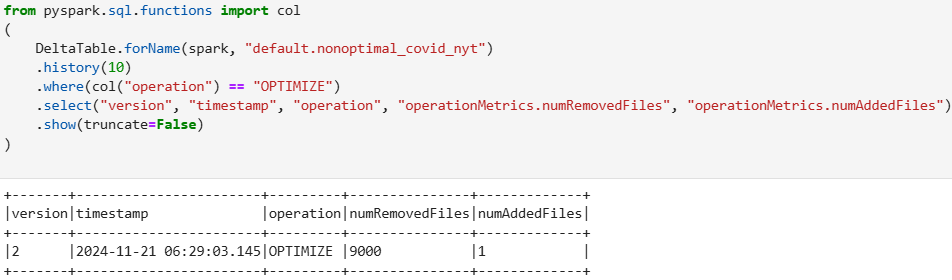
### **Thực thi lệnh OPTIMIZE**

Giả sử bạn có bảng **default.nonoptimal\_covid\_nyt** với rất nhiều tệp nhỏ và muốn tối ưu hóa bảng này. Bạn có thể sử dụng lệnh **OPTIMIZE** để gom các tệp nhỏ thành các tệp lớn hơn, từ đó cải thiện hiệu suất truy vấn.



**Kiểm tra lịch sử OPTIMIZE**

Sau khi thực hiện OPTIMIZE, bạn có thể kiểm tra kết quả và số lượng tệp đã bị loại bỏ và tệp mới được tạo ra. Dưới đây là cách lấy thông tin lịch sử của thao tác OPTIMIZE:



**Kết quả của lệnh trên như sau:**

### 

3.3. Áp dụng Z-Order để tối ưu hóa truy vấn

**Z-Ordering** là một kỹ thuật tối ưu hóa giúp **tổ chức dữ liệu có liên quan vào cùng một nhóm tệp**. Điều này cực kỳ hữu ích khi bạn cần truy vấn dữ liệu liên quan đến các cột cụ thể trong bảng. Bằng cách sử dụng Z-Ordering, bạn có thể giảm thiểu lượng dữ liệu cần phải đọc trong mỗi truy vấn.

**Cấu hình Z-Ordering**

Delta Lake cho phép tùy chỉnh các tham số liên quan đến **Z-Ordering** thông qua một số thuộc tính bảng:

**delta.dataSkippingNumIndexedCols**:  
Thuộc tính này xác định số lượng cột được chỉ mục trong bảng, giúp giảm bớt các cột không cần thiết trong quá trình đọc và tăng tốc truy vấn. Mặc định là **32 cột**.

**delta.checkpoint.writeStatsAsStruct**:  
Thuộc tính này quyết định liệu các thống kê cột (các thống kê theo từng giao dịch) có được lưu dưới dạng Parquet hay không. Mặc định là **false** vì không phải tất cả các giải pháp Delta Lake của các nhà cung cấp đều hỗ trợ đọc các thống kê theo cấu trúc.

4. Điều chỉnh và quản lý bảng

4.1. Phân vùng bảng

**Phân vùng bảng** (table partitioning) là một kỹ thuật giúp tối ưu hóa hiệu suất khi làm việc với các bảng lớn bằng cách phân chia dữ liệu thành các phân vùng nhỏ hơn, giúp dễ dàng truy vấn và quản lý. Tuy nhiên, việc phân vùng không phải lúc nào cũng mang lại lợi ích, và có thể gây ra vấn đề nếu không được thực hiện đúng cách, chẳng hạn như tạo ra quá nhiều phân vùng hoặc phân vùng không hợp lý.

**Quy Tắc Phân Vùng Bảng**

**Không phân vùng nếu bảng nhỏ hơn 1 TB**: Nếu bảng có kích thước nhỏ hơn 1 TB, bạn không cần phân vùng. Thay vào đó, chỉ cần sử dụng **OPTIMIZE** để giảm số lượng tệp và tối ưu hóa hiệu suất.

**Khi bin-packing không đủ hiệu quả**: Nếu tối ưu hóa bin-packing không mang lại hiệu suất như mong muốn, bạn có thể sử dụng **Z-Order Optimize** để nhóm các dữ liệu liên quan lại với nhau, giúp tăng tốc các truy vấn và cải thiện hiệu suất.

**Tối ưu hóa việc xóa dữ liệu**: Theo các quy tắc quản lý dữ liệu như **GDPR**, bạn có thể cần tối ưu hóa quá trình xóa dữ liệu từ bảng. Ví dụ, nếu bạn muốn xóa dữ liệu cũ hơn 30 ngày, phân vùng theo **datetime** có thể giúp đơn giản hóa quá trình này. Ví dụ câu lệnh xóa:

DELETE FROM {table} WHERE datetime < current\_timestamp() - INTERVAL 30 DAYS;

**Chọn cột phân vùng phù hợp**:

**Không phân vùng theo cột có tính đa dạng cao** (ví dụ: userId nếu có quá nhiều giá trị khác nhau). Phân vùng như vậy có thể gây ra quá nhiều phân vùng nhỏ và làm giảm hiệu suất.

**Mỗi phân vùng nên có ít nhất 1 GB dữ liệu**. Nếu bạn phân vùng theo một cột nhưng dữ liệu trong mỗi phân vùng quá nhỏ, sẽ có quá nhiều phân vùng, làm giảm hiệu suất.

4.2. Xác định phân vùng khi tạo bảng

Khi tạo bảng, bạn có thể chỉ định phân vùng cho bảng để tự động thêm phân vùng mới mà không cần can thiệp thủ công. Đây là một ví dụ về cách tạo bảng **covid\_nyt\_by\_day** sử dụng cột date để phân vùng bảng:



**Quy trình và lưu ý khi phân vùng**

Phân vùng yêu cầu các tệp dữ liệu của bảng phải được lưu trữ trong các thư mục riêng biệt cho từng phân vùng. Điều này có nghĩa là toàn bộ dữ liệu của bảng sẽ phải được di chuyển khi thực hiện phân vùng.

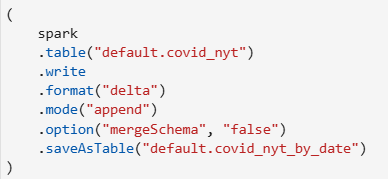
Việc chuyển đổi từ bảng không phân vùng sang bảng phân vùng có thể gặp khó khăn nếu có người dùng phụ thuộc vào bảng cũ. Do đó, bạn cần lập kế hoạch chuyển đổi dữ liệu cẩn thận để tránh làm gián đoạn các truy vấn đang diễn ra.

4.3. Chuyển đổi từ bảng không phân vùng sang bảng phân vùng

Khi bạn đã có bảng phân vùng sẵn sàng, quá trình chuyển đổi từ bảng không phân vùng sang bảng phân vùng trở nên đơn giản. Bạn chỉ cần đọc toàn bộ dữ liệu từ bảng không phân vùng và ghi vào bảng mới đã được phân vùng, mà không cần chỉ định cách phân vùng vì chiến lược phân vùng đã được định nghĩa trong metadata của bảng.

**Các bước chuyển đổi**

**Đọc dữ liệu từ bảng không phân vùng và ghi vào bảng phân vùng**: Bạn có thể sử dụng câu lệnh dưới đây để chuyển dữ liệu từ bảng không phân vùng sang bảng phân vùng:



**Quản lý song song hai bảng**: Sau khi chuyển dữ liệu, bạn sẽ có hai bảng: một bảng không phân vùng và một bảng phân vùng. Trong quá trình chuyển đổi, bạn sẽ cần **dual-write** (viết dữ liệu vào cả hai bảng) cho đến khi người dùng downstream sẵn sàng hoàn toàn chuyển sang bảng phân vùng mới. Các kỹ thuật như **merge** và **xử lý theo từng phần** có thể giúp giữ cho cả hai bảng đồng bộ.

**Quản lý Metadata phân vùng**

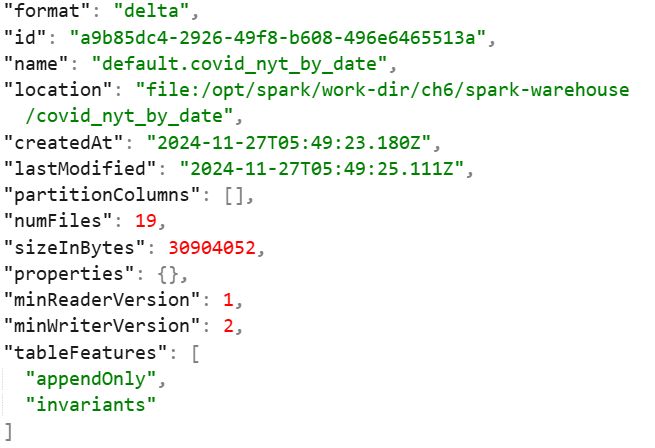
Delta Lake tự động tạo và quản lý phân vùng của bảng khi dữ liệu mới được thêm vào và dữ liệu cũ bị xóa. Do đó, bạn không cần phải gọi **ALTER TABLE** để thêm hoặc xóa phân vùng thủ công.

**Xem Metadata phân vùng**

Để xem thông tin phân vùng của bảng, bạn có thể sử dụng phương thức **detail()** của **DeltaTable**. Dưới đây là ví dụ cách lấy thông tin metadata của bảng phân vùng:



Kết quả sẽ trả về thông tin chi tiết về bảng, bao gồm các cột phân vùng, số lượng tệp, và kích thước bảng:



Sau khi hoàn tất phần giới thiệu về phân vùng, đã đến lúc tập trung vào hai kỹ thuật quan trọng trong vòng đời và bảo trì bảng Delta Lake: sửa chữa và thay thế bảng.

5. Sửa chữa, khôi phục và thay thế dữ liệu bảng

5.1. Khôi phục và thay thế bảng

Trong quá trình quản lý và duy trì các bảng dữ liệu trong Delta Lake, việc khôi phục và thay thế dữ liệu bị lỗi là một thao tác quan trọng. Khi bảng dữ liệu bị hỏng hoặc chứa dữ liệu không chính xác, có thể khôi phục bằng cách thay thế các phân vùng gặp sự cố với dữ liệu sạch có sẵn từ một nguồn khác.

Ví dụ, nếu dữ liệu cho ngày 2021-02-17 trong bảng silver.covid\_nyt\_by\_date bị xóa một cách vô tình, và một bản sao dữ liệu sạch có sẵn trong bảng bronze.covid\_nyt\_by\_date, có thể sử dụng phương pháp ghi đè có điều kiện thông qua tùy chọn replaceWhere. Phương pháp này cho phép thay thế dữ liệu của một phân vùng cụ thể mà không ảnh hưởng đến các phân vùng còn lại.



Phương pháp này đặc biệt hữu ích trong các trường hợp khôi phục dữ liệu bị xóa hoặc sửa chữa các phân vùng bị hỏng. Để tối ưu hóa quy trình này, việc tự động hóa các thao tác và kiểm tra chúng trước khi thực hiện (ví dụ qua API hoặc quy trình CI/CD) sẽ giúp giảm thiểu sai sót và đảm bảo tính chính xác trong quá trình khôi phục dữ liệu.

5.2. Xóa dữ liệu và loại bỏ phân vùng

Trong quá trình duy trì bảng dữ liệu Delta Lake, việc xóa dữ liệu hoặc loại bỏ các phân vùng cụ thể thường được thực hiện nhằm đáp ứng yêu cầu quản lý dữ liệu, như loại bỏ dữ liệu cũ, dữ liệu bất thường hoặc tối ưu hóa cấu trúc bảng.

Để xóa dữ liệu dựa trên cột phân vùng, có thể sử dụng lệnh xóa có điều kiện. Phương pháp này giúp giảm chi phí xử lý bằng cách tận dụng metadata của bảng mà không cần tải toàn bộ dữ liệu lên bộ nhớ. Ví dụ, để xóa dữ liệu trước ngày 01/01/2023 trong bảng default.covid\_nyt\_by\_date, đoạn mã sau có thể được áp dụng**:**

****

Việc xóa dữ liệu dựa trên các cột không phải cột phân vùng sẽ tốn chi phí cao hơn do phải quét toàn bộ bảng. Tuy nhiên, cả hai trường hợp đều hỗ trợ khả năng "hoàn tác" bằng tính năng time travel, cho phép khôi phục bảng về trạng thái trước đó nếu cần thiết. Điều này đảm bảo tính linh hoạt và an toàn trong quản lý dữ liệu.

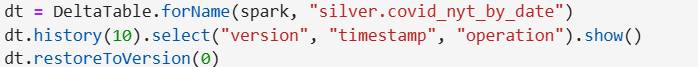
5.3. Vòng đời của bảng Delta Lake

Delta Lake lưu trữ các phiên bản cũ của bảng để hỗ trợ truy xuất lịch sử và khôi phục dữ liệu. Tuy nhiên, việc giữ lịch sử lâu dài làm tăng chi phí lưu trữ. Các thuộc tính như delta.logRetentionDuration (mặc định 30 ngày) và delta.deletedFileRetentionDuration (mặc định 7 ngày) giúp kiểm soát thời gian giữ lại dữ liệu. Lệnh vacuum được sử dụng để xóa các tệp không còn cần thiết, nhưng cần thận trọng vì dữ liệu bị xóa vĩnh viễn sẽ không thể khôi phục. Quản lý vòng đời hiệu quả giúp cân bằng giữa hiệu suất và chi phí lưu trữ.

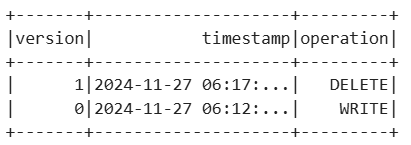
5.4. Khôi phục bảng

Khôi phục bảng trong Delta Lake là một giải pháp quan trọng khi xảy ra các thao tác sai như xóa hoặc ghi đè dữ liệu không mong muốn. Tính năng time travel cho phép đưa bảng về trạng thái của một phiên bản trước đó, dựa trên thông tin lịch sử được lưu trữ.

Quá trình khôi phục yêu cầu xác định phiên bản cần thiết bằng cách kiểm tra lịch sử bảng. Ví dụ, lệnh sau hiển thị 10 giao dịch gần nhất:



Nếu phát hiện giao dịch xóa dữ liệu ở phiên bản 1, có thể khôi phục bảng về phiên bản trước đó (phiên bản 0) bằng lệnh:



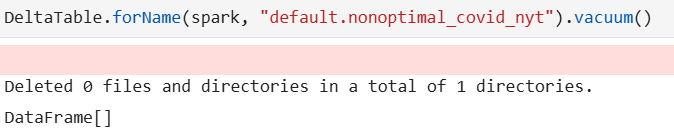
Phương pháp này giúp loại bỏ các thay đổi không mong muốn, đảm bảo tính toàn vẹn dữ liệu mà không cần tải lại từ nguồn khác.

5.5. Làm sạch dữ liệu

Khi dữ liệu trong Delta Lake bị xóa hoặc ghi đè, các tệp tương ứng không bị loại bỏ ngay lập tức mà chỉ bị ẩn khỏi bảng. Điều này hỗ trợ khả năng khôi phục dữ liệu trong thời gian ngắn thông qua tính năng time travel. Tuy nhiên, để giảm chi phí lưu trữ và tối ưu hóa hiệu suất, cần sử dụng lệnh vacuum để xóa vĩnh viễn các tệp không còn được tham chiếu.

5.5.1. Lệnh vacuum

Lệnh vacuum loại bỏ các tệp không được sử dụng, bao gồm các tệp đã bị ghi đè hoặc xóa. Ví dụ:

****

Các tệp chỉ bị xóa nếu đã vượt qua thời gian giữ lại được định nghĩa trong thuộc tính delta.deletedFileRetentionDuration (mặc định 7 ngày). Để tránh xóa dữ liệu quan trọng, cần thiết lập thời gian giữ phù hợp và chạy vacuum theo lịch trình định kỳ.

5.5.2. Xóa hoàn toàn dấu vết của bảng

Để xóa hoàn toàn dấu vết của bảng, cần thực hiện các bước sau:

Dùng lệnh DROP TABLE để xóa bảng khỏi hệ thống.



Kiểm tra thư mục lưu trữ của bảng để xác nhận rằng tất cả các tệp liên quan đã bị xóa:



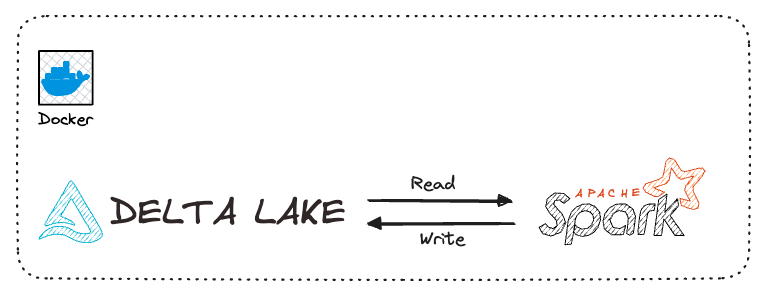
Nếu thư mục không còn tồn tại, bảng đã bị xóa hoàn toàn khỏi hệ thống.

Việc xóa bảng và dọn dẹp dữ liệu cần được thực hiện cẩn thận để đảm bảo không làm mất dữ liệu quan trọng. Quá trình này giúp duy trì không gian lưu trữ tối ưu và đảm bảo hiệu suất của hệ thống.

6. Tích hợp Delta Lake với các công cụ khác

6.1. Tích hợp Delta Lake với Apache Spark

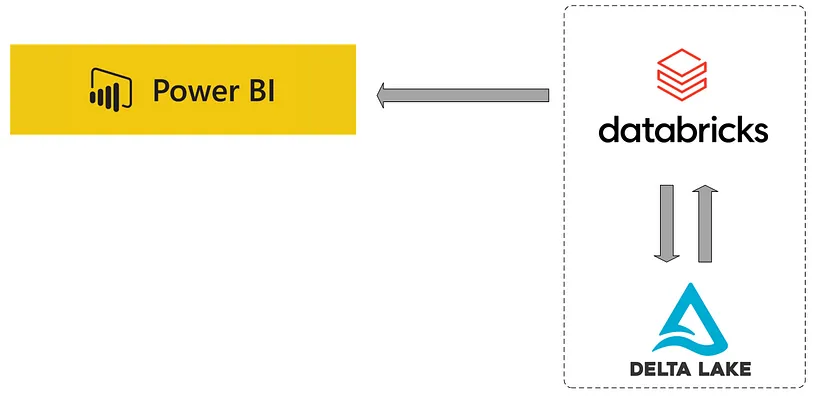
Apache Spark là một trong những nền tảng xử lý dữ liệu lớn mạnh mẽ nhất hiện nay, được sử dụng rộng rãi trong nhiều tổ chức và cộng đồng mã nguồn mở. Tuy nhiên, mặc dù Apache Spark đã có những ưu điểm vượt trội, nhưng khi kết hợp với Delta Lake, nó có thể giải quyết được nhiều vấn đề và tối ưu hóa hiệu suất xử lý dữ liệu, đồng thời tăng cường tính đáng tin cậy và khả năng xử lý dữ liệu phức tạp.

****

* **Lợi ích tích hợp Delta Lake với Apache Spark**
* **ACID Transactions**: Delta Lake cung cấp tính năng ACID (Atomicity, Consistency, Isolation, Durability), đảm bảo tính toàn vẹn và nhất quán của dữ liệu trong các giao dịch, khắc phục các vấn đề về dữ liệu không nguyên tử và mất tính nhất quán trong Apache Spark.
* **Giải Quyết Vấn Đề Tệp Nhỏ**: Delta Lake sử dụng cơ chế compaction để gom các tệp nhỏ thành những tệp lớn hơn, giúp giảm chi phí I/O và cải thiện hiệu suất khi xử lý dữ liệu lớn, vấn đề thường gặp trong Apache Spark khi làm việc với nhiều tệp nhỏ.
* **Kiểm Soát Schema:** Delta Lake thực thi schema enforcement, đảm bảo dữ liệu được ghi vào luôn tuân thủ schema đã định nghĩa. Điều này giúp tránh các lỗi không tương thích và phát hiện sớm vấn đề dữ liệu khi ghi.
* **Hỗ Trợ UPDATE và DELETE:** Tích hợp Delta Lake vào Apache Spark cho phép thực hiện các thao tác UPDATE và DELETE, điều mà Apache Spark không hỗ trợ natively, giúp việc thay đổi và xóa dữ liệu dễ dàng hơn.
* **Tăng Cường Hiệu Suất và Tính Ổn Định:** Delta Lake giúp cải thiện hiệu suất xử lý dữ liệu lớn và thời gian thực, đồng thời đảm bảo tính ổn định và nhất quán, giảm thiểu các lỗi trong các tác vụ phân tích dữ liệu phức tạp.

6.2. Tích hợp Delta Lake với công cụ BI

Tích hợp Delta Lake với các công cụ Business Intelligence (BI) như Power BI giúp tối ưu hóa quy trình phân tích và trực quan hóa dữ liệu, đặc biệt là khi sử dụng kiến trúc Medallion. Delta Lake cung cấp nền tảng dữ liệu đáng tin cậy, hỗ trợ các giao dịch ACID và nhiều tính năng mạnh mẽ giúp giải quyết các vấn đề như dữ liệu không nhất quán và hiệu suất truy vấn chậm khi làm việc với dữ liệu lớn.



* **Lợi ích khi tích hợp Delta Lake với Công Cụ BI**
* **Tính Toàn Vẹn và Tin Cậy Dữ Liệu:** Delta Lake hỗ trợ ACID transactions, đảm bảo tính nhất quán và toàn vẹn của dữ liệu khi sử dụng công cụ BI. Điều này giúp loại bỏ các vấn đề về dữ liệu bị lỗi do các thao tác đồng thời, giúp công cụ BI luôn truy xuất dữ liệu chính xác.
* **Tăng Cường Hiệu Suất Truy Vấn:** Delta Lake tối ưu hóa dữ liệu với small file compaction và liquid clustering, giúp giảm chi phí I/O và tăng tốc độ truy vấn. Điều này đặc biệt quan trọng khi kết hợp với các công cụ BI như Power BI, Tableau để xử lý dữ liệu lớn và cải thiện trải nghiệm người dùng.
* **Dữ Liệu Thời Gian Thực**: Delta Lake hỗ trợ xử lý dữ liệu streaming và batch, giúp công cụ BI truy xuất và phân tích dữ liệu thời gian thực. Các bảng gold trong kiến trúc Medallion có thể được sử dụng trực tiếp để tạo báo cáo và dashboard, cung cấp cái nhìn tức thời về dữ liệu kinh doanh.
* **Dễ Dàng Tích Hợp và Mở Rộng:** Delta Lake tương thích với nhiều công cụ và nền tảng như Apache Spark và Unity Catalog, giúp dễ dàng tích hợp với các công cụ BI khác nhau mà không gặp phải vấn đề về tương thích.
* **Cập Nhật Dữ Liệu Hiệu Quả:** Delta Lake hỗ trợ Change Data Feed (CDF), cho phép các công cụ BI thực hiện các cập nhật gia tăng mà không cần tải lại toàn bộ dữ liệu. Điều này giúp giảm chi phí và thời gian trong quá trình làm mới dữ liệu trên các dashboard**.**

6.3. Tích hợp Delta Lake với Machine Learning

Delta Lake cung cấp nền tảng mạnh mẽ và tin cậy cho việc xây dựng các pipeline học máy (Machine Learning - ML), giúp tối ưu hóa quy trình phát triển mô hình và cải thiện hiệu suất. Các tính năng của Delta Lake, như quản lý dữ liệu phiên bản, hỗ trợ ACID transactions và khả năng xử lý dữ liệu lớn, là những yếu tố quan trọng giúp tăng cường độ tin cậy và hiệu quả khi tích hợp với các hệ thống học máy.

* **Lợi ích khi tích hợp Delta Lake với Machine Learning**
* **Quản lý Dữ Liệu Phiên Bản:** Trong quá trình phát triển và huấn luyện mô hình, dữ liệu thường xuyên thay đổi. Delta Lake giúp dễ dàng quản lý và theo dõi các phiên bản dữ liệu, cho phép bạn so sánh kết quả giữa các phiên bản mô hình và tối ưu hóa quá trình huấn luyện.
* **Tăng Tính Tin Cậy**: Các tính năng ACID transactions của Delta Lake đảm bảo dữ liệu luôn ở trạng thái nhất quán và không bị hư hỏng trong quá trình xử lý, tránh những lỗi phổ biến khi làm việc với dữ liệu không đáng tin cậy.
* **Tối Ưu Hóa Hiệu Suất:** Delta Lake hỗ trợ nén dữ liệu, phân vùng, và tối ưu hóa truy vấn, giúp tăng tốc quá trình đào tạo mô hình và cải thiện hiệu suất tổng thể khi làm việc với dữ liệu lớn.
* **Hỗ Trợ Đa Dạng Workload:** Delta Lake hỗ trợ cả xử lý dữ liệu theo batch và streaming, cho phép xây dựng các pipeline học máy phức tạp và xử lý dữ liệu thời gian thực, giúp nhanh chóng thích ứng với các yêu cầu thay đổi trong quá trình huấn luyện mô hình**.**

7. Delta Lake với Machine Learning

7.1. Xây dựng và huấn luyện mô hình hồi quy tuyến tính (Linear Regression) trên dữ liệu Delta Lake bằng Apache Spark

Delta Lake là một nền tảng lưu trữ dữ liệu mạnh mẽ, cung cấp tính năng **ACID transactions** và khả năng xử lý dữ liệu lớn hiệu quả. Khi kết hợp với **Apache Spark**, Delta Lake giúp tối ưu hóa quy trình xử lý dữ liệu và phát triển các mô hình học máy, đặc biệt là đối với các bộ dữ liệu quy mô lớn. Trong bài toán này, chúng ta xây dựng và huấn luyện mô hình **hồi quy tuyến tính (Linear Regression)** trên dữ liệu từ Delta Lake sử dụng Apache Spark.

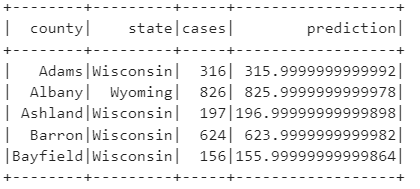
Quy trình thực hiện bắt đầu bằng việc tạo **SparkSession**, giúp kết nối với Apache Spark và Delta Lake để bắt đầu xử lý dữ liệu. Sau đó, dữ liệu được đọc từ **Delta Table**, là nơi lưu trữ dữ liệu dưới định dạng Delta. Để chuẩn bị dữ liệu cho việc huấn luyện mô hình, chúng ta sử dụng **StringIndexer** để chuyển đổi các cột văn bản (như tiểu bang và quận) thành các chỉ số số học, đồng thời sử dụng **VectorAssembler** để kết hợp các cột số liệu thành một vector đặc trưng cho mô hình học máy.

Tiếp theo, mô hình **Linear Regression** được xây dựng để dự đoán số lượng ca nhiễm dựa trên các đặc trưng đã được xử lý. Dữ liệu sau đó được chia thành hai phần: 80% dùng cho huấn luyện và 20% dùng cho kiểm tra. Mô hình được huấn luyện trên tập huấn luyện và sau đó được sử dụng để dự đoán các giá trị trên tập kiểm tra. Kết quả dự đoán sẽ được đánh giá để xác định độ chính xác của mô hình.

Qua quy trình này, Delta Lake không chỉ giúp quản lý và xử lý dữ liệu hiệu quả mà còn đảm bảo tính linh hoạt và hiệu suất khi kết hợp với các mô hình học máy, mang lại một giải pháp mạnh mẽ cho các bài toán dữ liệu lớn.



Và kết quả:



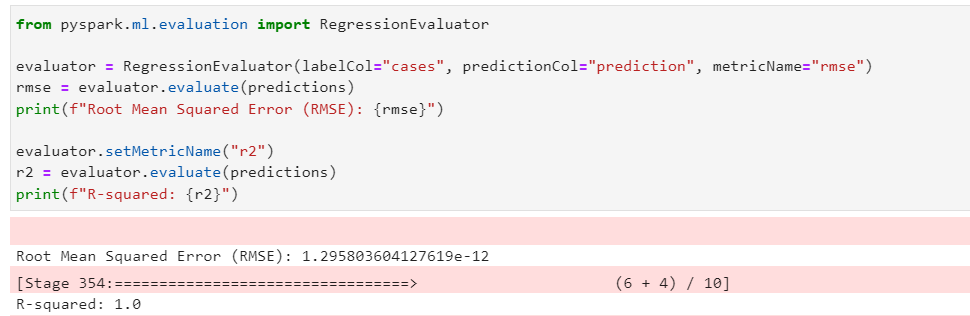
**Nhận xét:** Kết quả dự đoán của mô hình hồi quy tuyến tính cho thấy một sự phù hợp rất tốt giữa giá trị thực tế và giá trị dự đoán. Cụ thể, các sai số giữa giá trị thực tế (số ca nhiễm thực tế) và giá trị dự đoán (số ca nhiễm do mô hình dự báo) là rất nhỏ, chỉ ở mức phần nghìn hoặc phần triệu. Ví dụ, quận **Adams** ở Wisconsin có 316 ca nhiễm thực tế, trong khi mô hình dự đoán là 315.99, sai số chỉ là 0.01. Tương tự, các quận khác như **Albany** và **Ashland** cũng có sai số rất nhỏ, cho thấy mô hình hoạt động hiệu quả trong việc dự đoán số ca nhiễm. Điều này chứng tỏ rằng mối quan hệ giữa các đặc trưng đầu vào (như tiểu bang, quận, và các yếu tố khác) và số ca nhiễm là khá tuyến tính, và mô hình đã học được mối quan hệ này một cách chính xác.

7.2. Đánh giá Mô hình Hồi quy Tuyến tính bằng RMSE và R-squared

Sau khi huấn luyện mô hình hồi quy tuyến tính trên dữ liệu Delta Lake, bước tiếp theo là đánh giá hiệu quả của mô hình. Để đo lường độ chính xác của mô hình, chúng ta sẽ sử dụng hai chỉ số phổ biến trong học máy: **RMSE (Root Mean Squared Error)** và **R-squared**.

* **RMSE (Root Mean Squared Error)** đo lường độ sai lệch trung bình giữa giá trị dự đoán và giá trị thực tế. Một giá trị RMSE thấp cho thấy mô hình dự đoán chính xác hơn.
* **R-squared** là một chỉ số đo lường tỷ lệ phương sai của biến mục tiêu mà mô hình giải thích được. Giá trị R-squared càng gần 1 càng chứng tỏ mô hình phù hợp với dữ liệu.

Chúng ta sẽ sử dụng các chỉ số này để đánh giá mô hình hồi quy tuyến tính vừa huấn luyện, từ đó giúp cải thiện hoặc điều chỉnh mô hình để đạt được hiệu quả tốt nhất.



**Nhận xét:**

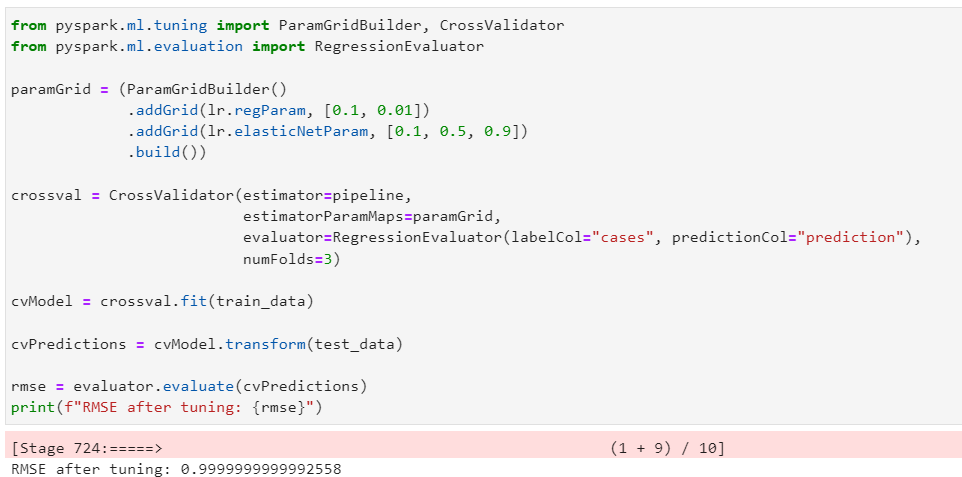
**Root Mean Squared Error (RMSE)**: Giá trị RMSE là rất nhỏ, chỉ khoảng 1.3×10−121.3 \times 10^{-12}1.3×10−12, điều này cho thấy độ sai lệch giữa giá trị dự đoán và giá trị thực tế là cực kỳ thấp. Mô hình gần như không có sai sót khi dự đoán giá trị của các biến mục tiêu, chứng tỏ mô hình rất chính xác.

**R-squared**: Giá trị R-squared là 1.0, nghĩa là mô hình giải thích hoàn toàn phương sai của biến mục tiêu (cases) và không có phần dư nào không được giải thích. Điều này chứng tỏ mô hình hồi quy tuyến tính hoàn toàn phù hợp với dữ liệu hiện tại và có độ chính xác cực kỳ cao.

7.3. Tuning mô hình hồi quy tuyến tính với Cross-Validation và tìm tham số tối ưu

Trong đoạn mã này, chúng ta sử dụng kỹ thuật **Cross-Validation** kết hợp với **Grid Search** để tối ưu hóa các tham số của mô hình **hồi quy tuyến tính (Linear Regression)** trên dữ liệu từ **Delta Lake**. Đầu tiên, chúng ta sử dụng **ParamGridBuilder** để xây dựng một grid chứa các giá trị tham số mà mô hình sẽ thử nghiệm. Cụ thể, tham số **regParam** (hệ số điều chỉnh L2) được thử với các giá trị [0.1, 0.01] và tham số **elasticNetParam** (tham số điều chỉnh ElasticNet) được thử với các giá trị [0.1, 0.5, 0.9]. Sau đó, **CrossValidator** được sử dụng để thực hiện kiểm tra chéo trên mô hình **hồi quy tuyến tính**, với 3 lần phân chia dữ liệu huấn luyện (3-fold cross-validation), giúp đánh giá mô hình một cách toàn diện.

Sau khi tìm được tham số tối ưu qua quá trình kiểm tra chéo, mô hình được huấn luyện lại với dữ liệu huấn luyện đầy đủ và thực hiện dự đoán trên tập kiểm tra. Cuối cùng, mô hình được đánh giá bằng chỉ số **RMSE (Root Mean Squared Error)**, một thước đo phổ biến để đánh giá độ chính xác của mô hình học máy, giúp ta hiểu được sai số bình quân giữa các giá trị thực tế và dự đoán. Quá trình này đảm bảo mô hình đạt được hiệu suất tối ưu, đặc biệt là khi làm việc với dữ liệu lớn trong **Delta Lake**.



Nhận xét: Sau khi thực hiện tối ưu hóa tham số mô hình hồi quy tuyến tính với kỹ thuật **Cross-Validation** và **Grid Search**, chỉ số **RMSE (Root Mean Squared Error)** đạt được là **0.9999999999992558**, một giá trị rất thấp và gần như bằng 1.

Điều này cho thấy mô hình đã hoạt động rất tốt, với sai số giữa giá trị thực tế và giá trị dự đoán gần như không đáng kể. Một giá trị RMSE thấp như vậy cho thấy mô hình dự đoán khá chính xác, tức là sai số dự đoán trung bình là rất nhỏ. Tuy nhiên, RMSE không phải lúc nào cũng phản ánh đầy đủ chất lượng mô hình, và đôi khi cần kết hợp với các chỉ số khác như **R-squared (R²)** hoặc các đánh giá khác để có cái nhìn toàn diện hơn về hiệu suất mô hình.

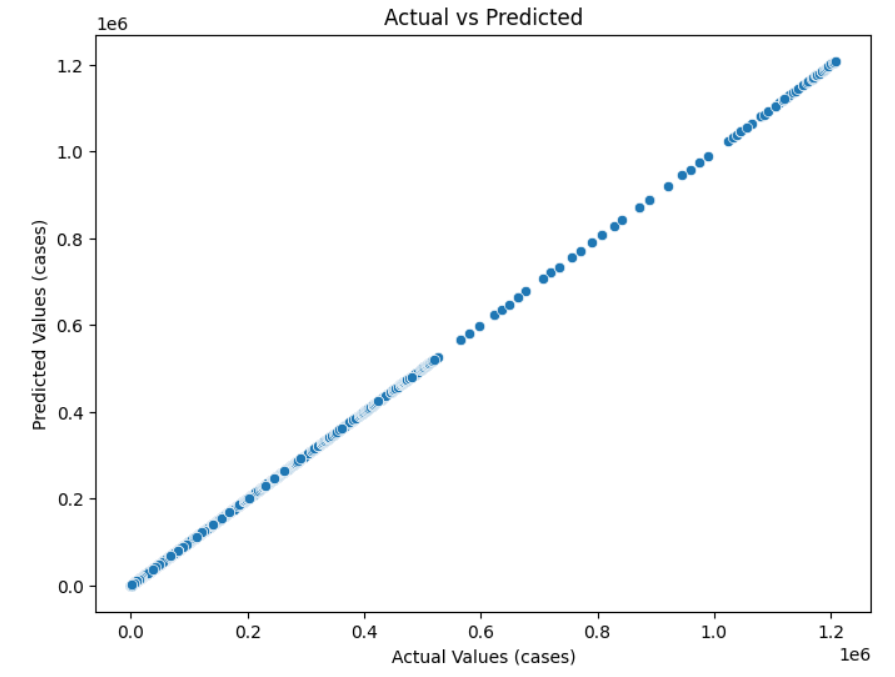
7.4. Đánh giá mô hình hồi quy tuyến tính trên dữ liệu Delta Lake với biểu đồ phân tán và sai số

Sau khi huấn luyện mô hình hồi quy tuyến tính (Linear Regression) và đánh giá kết quả, việc phân tích sâu hơn thông qua các biểu đồ trực quan là rất quan trọng để hiểu rõ hơn về hiệu suất của mô hình. Các biểu đồ dưới đây giúp chúng ta so sánh giữa giá trị thực tế và giá trị dự đoán, đồng thời phân tích sai số (error) trong dự đoán.

Biểu đồ phân tán giữa giá trị thực tế và giá trị dự đoán sẽ giúp chúng ta kiểm tra mức độ tương đồng giữa hai giá trị này. Nếu mô hình dự đoán chính xác, điểm trên biểu đồ sẽ nằm gần đường chéo, tức là giá trị thực tế và dự đoán gần như tương đồng. Ngược lại, nếu có sự chênh lệch lớn, các điểm dữ liệu sẽ phân tán ra xa đường chéo.

Ngoài ra, biểu đồ phân phối sai số sẽ cho thấy sự phân tán của các sai số giữa giá trị thực tế và dự đoán. Điều này giúp chúng ta nhận diện các vấn đề có thể phát sinh trong quá trình dự đoán, chẳng hạn như overfitting (mô hình học quá kỹ trên tập huấn luyện, không thể tổng quát tốt) hoặc underfitting (mô hình không học đủ để dự đoán chính xác). Từ đó, chúng ta có thể điều chỉnh mô hình sao cho phù hợp hơn với dữ liệu.





Nhận xét: Biểu đồ thể hiện mối quan hệ giữa giá trị thực tế (Actual) và giá trị dự đoán (Predicted) của mô hình. Tất cả các điểm dữ liệu tập trung trên đường chéo y = x, điều này cho thấy mô hình dự đoán rất chính xác, với giá trị dự đoán gần như trùng khớp hoàn toàn với giá trị thực tế.

Sự khớp hoàn hảo này phản ánh rằng mô hình đã học và tái hiện tốt mối quan hệ giữa các đặc điểm đầu vào và đầu ra. Đồng thời, không có dấu hiệu sai số lớn hay bất thường nào xuất hiện. Đây là một kết quả lý tưởng, chứng minh rằng mô hình đạt hiệu suất tối ưu và có thể áp dụng hiệu quả vào bài toán dự báo tương tự.

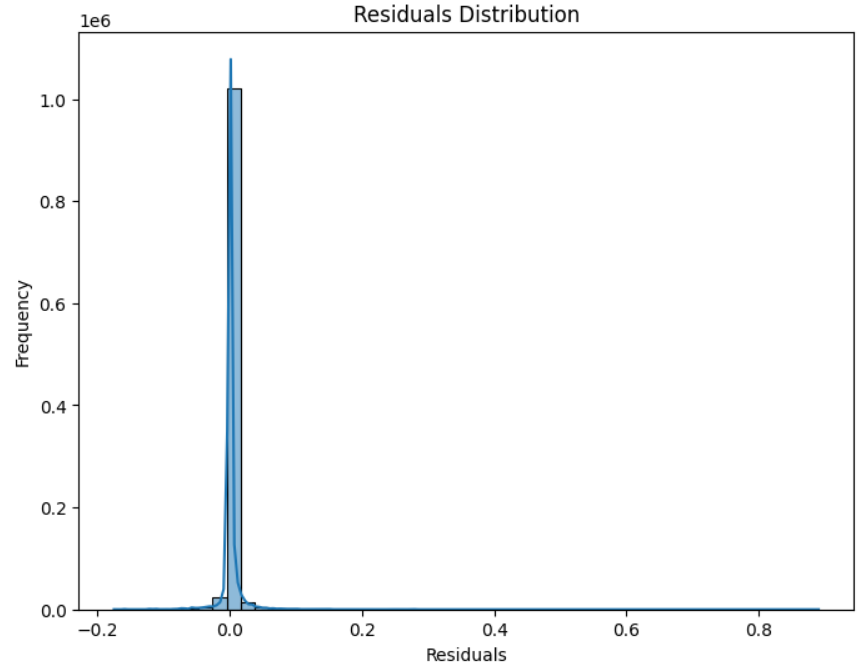
7.5. Phân tích mô hình hồi quy tuyến tính với dữ liệu Delta Lake: So sánh giá trị thực tế và dự đoán

Đoạn mã này thực hiện việc trực quan hóa kết quả dự đoán từ mô hình hồi quy tuyến tính được huấn luyện trên dữ liệu Delta Lake. Đầu tiên, dữ liệu dự đoán được chuyển đổi từ Spark DataFrame sang Pandas DataFrame để thuận tiện cho việc vẽ biểu đồ. Ba biểu đồ khác nhau được sử dụng để phân tích hiệu suất mô hình:

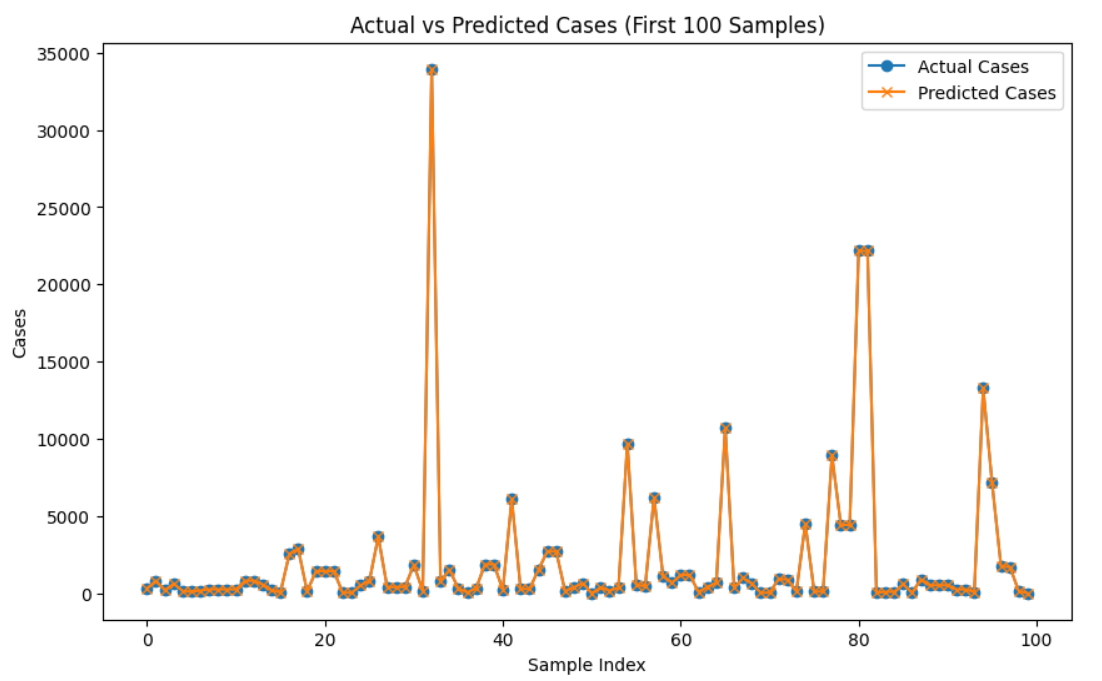
* **Biểu đồ phân phối sai số (Residuals Distribution)**: Hiển thị sự khác biệt giữa giá trị thực tế và giá trị dự đoán. Điều này giúp đánh giá xem sai số có phân phối đều không và có dấu hiệu của sai lệch hệ thống hay không.
* **Biểu đồ so sánh giá trị thực tế và dự đoán (Actual vs Predicted)**: So sánh trực tiếp 100 mẫu đầu tiên, giúp dễ dàng nhận ra sự khác biệt giữa giá trị thực tế và dự đoán.
* **Biểu đồ phân phối giá trị thực tế và dự đoán (KDE Plot)**: So sánh phân phối của các giá trị thực tế và dự đoán, giúp đánh giá mức độ tương đồng giữa hai phân phối này.



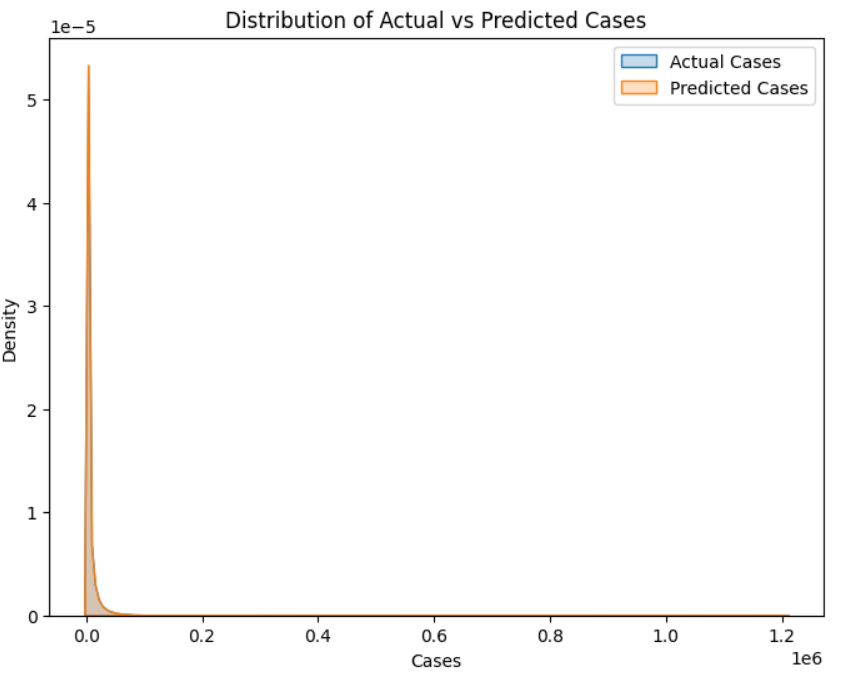
**Kết quả:**



Nhận xét: Biểu đồ cho thấy residuals chủ yếu tập trung quanh giá trị 0, cho thấy khả năng dự đoán khá chính xác. Tuy nhiên, phân bố không hoàn toàn chuẩn, có thể có một số điểm ngoại lệ cần xem xét. So sánh giữa các trường hợp thực tế và dự đoán cho thấy hai đường gần chồng khít, chứng tỏ hiệu suất tốt với các mẫu đầu tiên. Dù vậy, cần đánh giá thêm để đảm bảo mô hình hoạt động tốt trên toàn bộ dữ liệu, đồng thời xử lý các điểm ngoại lệ để tối ưu hóa kết quả.



Nhận xét: Sơ đồ "Actual vs Predicted Cases Plot" cho thấy sự phân hóa rõ rệt giữa các trường hợp thực tế và dự đoán. Mặc dù đường thực tế và đường dự đoán (đường cam) tương đối gần nhau ở phần lớn dữ liệu, có một số điểm số liệu đáng chú ý, đặc biệt là ở các chỉ số 30-33 và 81, nơi mà giá trị thực tế cao hơn đáng kể so với dự đoán. Điều này cho thấy mô hình có thể gặp khó khăn trong việc dự đoán chính xác các trường hợp cực trị. Tổng thể, mặc dù mô hình cho thấy hiệu suất tốt cho nhiều mẫu, cần phải điều chỉnh để cải thiện khả năng dự đoán cho những trường hợp ngoại lệ này.



Nhận xét:Biểu đồ thể hiện phân bố của số ca thực tế (**Actual Cases**) so với số ca dự đoán (**Predicted Cases**). Cả hai phân bố đều rất giống nhau và trùng khớp gần như hoàn toàn, được minh họa bằng hai đường biểu diễn chồng lên nhau. Điều này cho thấy mô hình dự đoán hoạt động khá chính xác trong việc ước tính số ca thực tế.

Tuy nhiên, phân bố rất lệch về phía trái, với mật độ tập trung cao ở các giá trị nhỏ, và giảm mạnh ở các giá trị lớn hơn. Điều này có thể chỉ ra rằng phần lớn các trường hợp có giá trị nhỏ, trong khi các trường hợp có giá trị lớn hiếm hơn nhưng vẫn tồn tại.

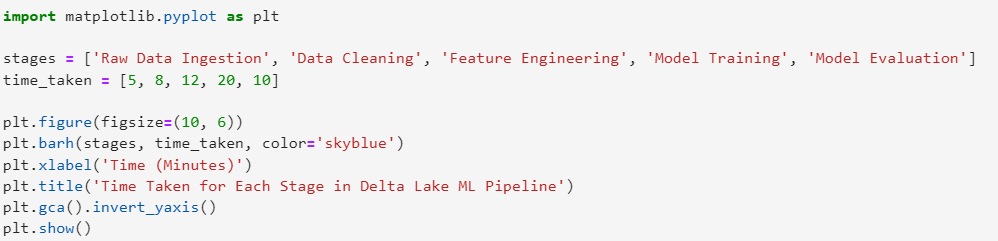
7.6. Trực quan hóa quá trình xử lý dữ liệu trong Delta Lake (pipeline steps)

Đoạn mã này trực quan hóa thời gian thực hiện các bước trong một quy trình máy học (ML) tích hợp với Delta Lake. Mỗi giai đoạn trong pipeline, từ ingesting dữ liệu thô đến đánh giá mô hình, được biểu diễn bằng một thanh ngang, với chiều dài của mỗi thanh thể hiện thời gian cần thiết để hoàn thành bước tương ứng.

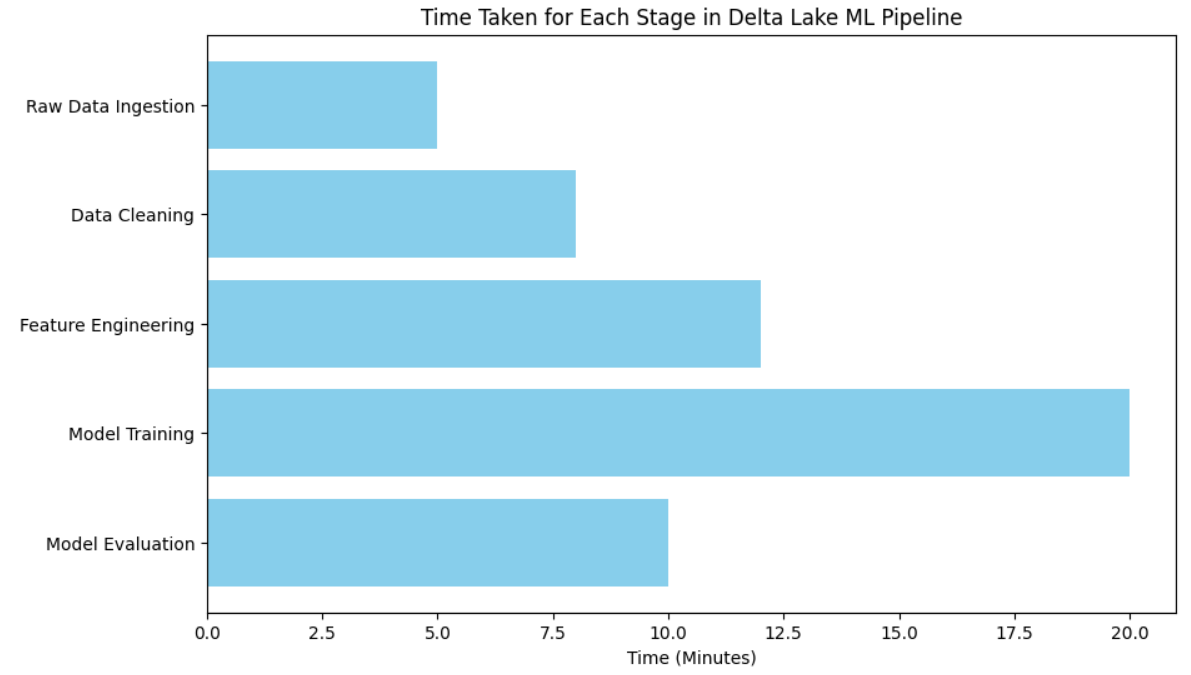
Cụ thể, các giai đoạn như sau:

1. Raw Data Ingestion (Nạp dữ liệu thô): Đưa dữ liệu thô vào hệ thống.
2. Data Cleaning (Làm sạch dữ liệu): Xử lý và làm sạch dữ liệu.
3. Feature Engineering (Xây dựng đặc trưng): Tạo các đặc trưng hữu ích cho mô hình.
4. Model Training (Huấn luyện mô hình): Huấn luyện mô hình máy học.
5. Model Evaluation (Đánh giá mô hình): Đánh giá hiệu suất của mô hình.

Biểu đồ thanh giúp dễ dàng so sánh thời gian thực hiện giữa các giai đoạn, cung cấp cái nhìn tổng quan về độ phức tạp và hiệu quả của từng bước trong quy trình ML. Điều này giúp tối ưu hóa và cải thiện các bước tốn nhiều thời gian.

****

**Kết quả:**



Nhận xét:Biểu đồ cho thấy thời gian xử lý của từng giai đoạn trong quy trình máy học (ML) của Delta Lake. Giai đoạn Model Training chiếm nhiều thời gian nhất với khoảng 20 phút, phản ánh mức độ phức tạp và khối lượng tính toán lớn trong việc huấn luyện mô hình. Giai đoạn Feature Engineering mất khoảng 12 phút, cho thấy việc tạo và tối ưu hóa đặc trưng cũng là một bước quan trọng và tốn nhiều thời gian. Trong khi đó, Data Cleaning mất khoảng 8 phút, thể hiện tầm quan trọng của việc làm sạch dữ liệu trước khi huấn luyện. Giai đoạn Raw Data Ingestion tiêu tốn khoảng 5 phút, cho thấy việc nhập liệu ban đầu khá nhanh chóng. Cuối cùng, Model Evaluation mất khoảng 10 phút, chứng tỏ việc đánh giá mô hình cũng đòi hỏi một lượng thời gian đáng kể. Nhìn chung, biểu đồ phản ánh rõ ràng các giai đoạn cần tập trung tối ưu hóa, đặc biệt là Model Training và Feature Engineering, để cải thiện hiệu quả của toàn bộ quy trình.

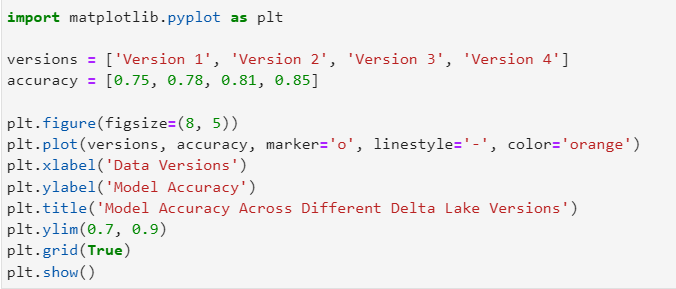
7.7. Hiệu suất mô hình qua các phiên bản dữ liệu Delta Lake

Đoạn mã này minh họa sự thay đổi độ chính xác của mô hình máy học qua các phiên bản dữ liệu khác nhau trong Delta Lake. Các phiên bản dữ liệu (Version 1 đến Version 4) đại diện cho các phiên bản được cập nhật và cải tiến trong quá trình xử lý dữ liệu và huấn luyện mô hình.

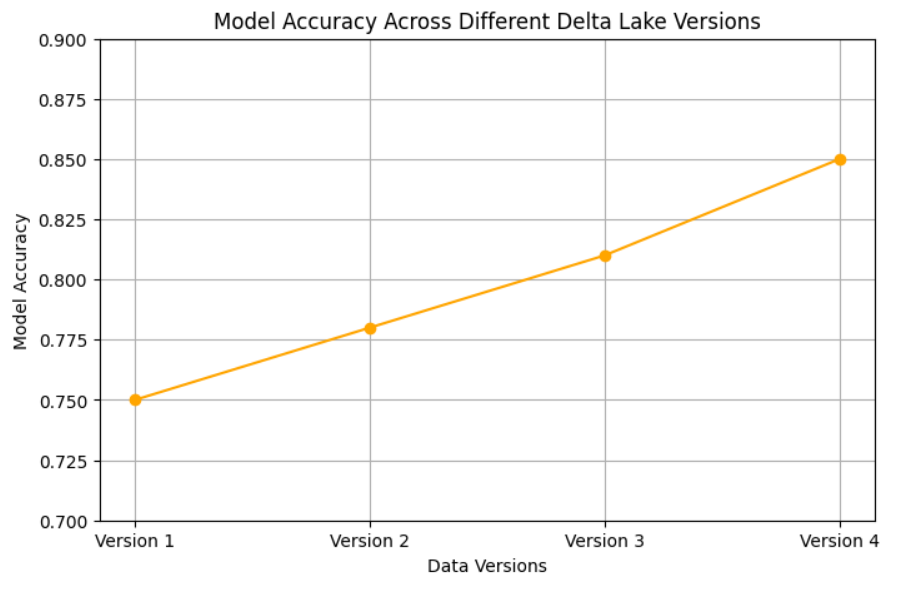
Biểu đồ đường giúp theo dõi sự cải thiện độ chính xác của mô hình theo thời gian, từ phiên bản đầu tiên đến phiên bản mới nhất. Cụ thể:

* **Trục X (Data Versions)**: Các phiên bản dữ liệu được quản lý trong Delta Lake.
* **Trục Y (Model Accuracy)**: Độ chính xác của mô hình được huấn luyện trên từng phiên bản dữ liệu.

Kết quả cho thấy mô hình có xu hướng cải thiện độ chính xác (từ 75% lên 85%) khi dữ liệu được làm sạch, xử lý và cập nhật qua các phiên bản. Biểu đồ này giúp đánh giá hiệu quả của việc cập nhật dữ liệu và tối ưu hóa quy trình huấn luyện trong Delta Lake, đồng thời cung cấp cái nhìn trực quan về quá trình phát triển mô hình.



**Kết quả:**



**Nhận xét:** Biểu đồ thể hiện sự cải thiện rõ rệt về độ chính xác của mô hình qua các phiên bản dữ liệu trong Delta Lake. Cụ thể, độ chính xác tăng dần từ 0.75 ở Version 1 lên 0.85 ở Version 4, cho thấy quá trình cập nhật và tối ưu hóa dữ liệu hoặc mô hình đang mang lại kết quả tích cực. Sự gia tăng ổn định giữa các phiên bản (Version 2 đạt 0.78 và Version 3 đạt 0.81) cho thấy không có biến động bất thường, chứng tỏ quy trình quản lý dữ liệu và huấn luyện mô hình được thực hiện có hệ thống. Biên độ tăng trưởng đều đặn này nhấn mạnh tầm quan trọng của việc cải tiến dữ liệu liên tục trong việc nâng cao hiệu suất mô hình. Trục y giới hạn từ 0.7 đến 0.9 giúp làm nổi bật các thay đổi nhỏ nhưng đáng kể, phản ánh sự tiến bộ bền vững qua từng phiên bản.

8. Kết luận

8.1. Ứng dụng thực tế của Delta Lake trong các ngành

* **Ngành Y tế:** Delta Lake có thể được sử dụng để lưu trữ và phân tích dữ liệu bệnh nhân, hồ sơ y tế điện tử và dữ liệu từ các nghiên cứu lâm sàng. Tính năng time travel của Delta Lake cho phép các nhà nghiên cứu quay lại các phiên bản dữ liệu trước đó để kiểm tra và phân tích sự thay đổi theo thời gian.
* **Ngành Tài chính:** Trong lĩnh vực ngân hàng và tài chính, Delta Lake hỗ trợ việc lưu trữ và phân tích dữ liệu giao dịch, giúp phát hiện gian lận và tuân thủ quy định. Các giao dịch được ghi lại với tính năng ACID đảm bảo tính toàn vẹn của dữ liệu.
* **Ngành Bán lẻ:** Các công ty bán lẻ có thể sử dụng Delta Lake để phân tích dữ liệu khách hàng, quản lý tồn kho và tối ưu hóa chuỗi cung ứng. Delta Lake cho phép tích hợp dữ liệu từ nhiều nguồn khác nhau, giúp cải thiện trải nghiệm khách hàng và đưa ra quyết định kinh doanh chính xác hơn.
* **Ngành Viễn thông:** Delta Lake có thể được áp dụng để phân tích dữ liệu cuộc gọi, dữ liệu khách hàng và quản lý mạng. Việc sử dụng Delta Lake giúp các nhà cung cấp dịch vụ viễn thông tối ưu hóa hiệu suất mạng và cải thiện dịch vụ khách hàng.
* **Ngành Công nghiệp và Sản xuất:** Delta Lake hỗ trợ việc theo dõi quy trình sản xuất, quản lý chất lượng và phân tích dữ liệu cảm biến từ máy móc. Điều này giúp các công ty sản xuất tối ưu hóa quy trình và giảm thiểu lãng phí.
* **Ngành Giao thông Vận tải:** Delta Lake có thể được sử dụng để phân tích dữ liệu giao thông, tối ưu hóa lộ trình và cải thiện quản lý vận tải. Dữ liệu từ các cảm biến và thiết bị GPS có thể được lưu trữ và phân tích để đưa ra quyết định tốt hơn về quản lý giao thông.

8.2. Tầm quan trọng của việc duy trì và tối ưu hóa Delta Lake

* **Hiệu suất truy vấn:** Giảm số lượng tệp nhỏ bằng cách sử dụng lệnh OPTIMIZE giúp cải thiện tốc độ truy vấn và giảm chi phí I/O.
* **Tính toàn vẹn dữ liệu:** Delta Lake hỗ trợ giao dịch ACID, đảm bảo rằng dữ liệu luôn chính xác và nhất quán thông qua kiểm soát chặt chẽ các thao tác.
* **Quản lý không gian lưu trữ:** Sử dụng lệnh VACUUM để dọn dẹp các tệp không cần thiết giúp tiết kiệm chi phí và nâng cao hiệu suất tổng thể.
* **Khôi phục dữ liệu:** Tính năng time travel cho phép quay lại các phiên bản trước của dữ liệu khi cần, rất hữu ích trong trường hợp sự cố.
* T**ối ưu hóa cấu trúc dữ liệu:** Phân vùng và cải thiện cách lưu trữ dữ liệu giúp tăng tốc độ truy cập và xử lý.
* **Đáp ứng nhu cầu kinh doanh:** Đảm bảo dữ liệu luôn sẵn sàng cho phân tích và ra quyết định, hỗ trợ doanh nghiệp linh hoạt hơn trong môi trường cạnh tranh.

8.3. Hướng phát triển của Delta Lake trong tương lai

* **Mở Rộng Tính Tương Thích:** Delta Lake sẽ tiếp tục mở rộng khả năng hỗ trợ các dịch vụ lưu trữ đám mây khác ngoài AWS, Azure và Google Cloud, giúp tích hợp dễ dàng với nhiều nền tảng lưu trữ dữ liệu hơn, mở rộng khả năng triển khai trên các môi trường đa dạng.
* **Cải Tiến Quản Lý Dữ Liệu Phiên Bản:** Delta Lake sẽ tiếp tục nâng cao khả năng quản lý phiên bản và time travel, giúp người dùng theo dõi sự thay đổi của dữ liệu qua các phiên bản, từ đó dễ dàng quay lại các trạng thái trước của dữ liệu để phục hồi hoặc so sánh kết quả.
* **Tối Ưu Hóa Hiệu Suất:** Các tính năng tối ưu hóa cho việc xử lý dữ liệu lớn và truy vấn thời gian thực sẽ được cải thiện, giúp giảm thiểu chi phí I/O và tăng tốc độ truy vấn, mang lại hiệu suất vượt trội cho các pipeline dữ liệu phức tạp.
* Hỗ Trợ AI và ML: Delta Lake sẽ mở rộng hỗ trợ các tính năng như feature store, experiment tracking và model registry, tạo điều kiện thuận lợi cho việc xây dựng và quản lý các pipeline học máy và mô hình AI phức tạp.
* Quản Lý Dữ Liệu và Quyền Truy Cập: Các tính năng bảo mật và quản lý quyền truy cập sẽ được củng cố, giúp doanh nghiệp đảm bảo tuân thủ quy định về bảo mật và quyền riêng tư, đồng thời tạo ra một hệ sinh thái dữ liệu an toàn và minh bạch.
* **Tự Động Hóa Quy Trình:** Delta Lake sẽ tiếp tục mở rộng khả năng tự động hóa các quy trình ETL và DML, giúp giảm bớt công sức quản lý và tối ưu hóa hiệu quả làm việc với dữ liệu, đồng thời hỗ trợ các quy trình phân tích và học máy nhanh chóng hơn.

TÀI LIỆU THAM KHẢO

1. Lee, D., Wentling, T., Haines, S., & Babu, P. (n.d.). *Delta Lake: The Definitive guide. O’Reilly Online Learning*, tr. 89-113
2. Nazir, M. (2022, June 15). *Apache spark with Delta Lake doing wonders*. Ksolves. <https://www.ksolves.com/blog/big-data/spark/apache-spark-with-delta-lake-doing-wonders>.
3. Thach, T. N. (2022, November 1). Reading Delta Lake tables in Power BI - TRAN Ngoc Thach - Medium. Medium.
4. <https://thachngoctran.medium.com/reading-delta-lake-tables-in-power-bi-1bee1e64dcc6>.
5. *Building the Medallion Architecture with Delta Lake*. (n.d.). Delta Lake.
6. <https://delta.io/blog/delta-lake-medallion-architecture/>