

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue logo with a ball and a black background

Description automatically generated

**MÔN HỌC: BIG DATA ANALYST**

**BÁO CÁO CUỐI KỲ**

**TÊN ĐỀ TÀI:**

**PHÂN TÍCH VÀ TRỰC QUAN HÓA DỮ LIỆU TRÊN POWER BI**

**GVHD:** *THS. LÊ THỊ MINH CHÂU*

**Lớp HP:** Big Data Analysis\_ Nhom 02CLC

**Nhóm thực hiện:** *Nhóm 4*

**Học kỳ:** *1*

**Năm học:** *2024 - 2025*

*Thành phố Hồ Chí Minh, tháng 12, năm 2024*

**BẢNG PHÂN CÔNG CÔNG VIỆC**

**HỌC KỲ I NĂM HỌC 2024-2025**

**Nhóm 4**

***Đề tài: PHÂN TÍCH VÀ TRỰC QUAN HÓA DỮ LIỆU TRÊN POWER BI***

|  |  |  |
| --- | --- | --- |
| **MSSV** | **Họ và tên** | **Phân chia công việc** |
| 21110162 | Trần Nguyễn Trí Đạt | * Tìm kiếm nội dung Power BI * Tiền xử lí dữ liệu * Bài tập từ câu 30 đến 34 |
| 21110169 | Tào Việt Đức | * Tìm hiểu nội dung apache Spark * Phân tích mức độ tai nạn theo vị trí địa lí * Bài tập từ câu 41 đến 45 |
| 21110190 | Phạm Tấn Huy | * Mô tả tổng quan dữ liệu * Phân tích mực độ tai nạn theo thời gian * Bài tập từ câu 35 đến 40 |
| 21110841 | Lê Nguyễn Trí Nhân | * Giới thiệu đề tài * Phân tích mức độ tai nạn theo cơ sở hạ tầng * Phân tích mức độ tai nạn theo thời gian * Bài tập câu 46 đến câu 50 |

**NHẬN XÉT CỦA GIÁO VIÊN**

**ĐIỂM GV KÝ TÊN**

**MỤC LỤC**

[**CHƯƠNG 1: GIỚI THIỆU** 1](#_Toc185468165)

[**1.1** **Giới thiệu dự án** 1](#_Toc185468166)

[**1.2 Mục tiêu** 1](#_Toc185468167)

[**1.3 Nhiệm vụ** 2](#_Toc185468168)

[**CHƯƠNG 2: CÔNG NGHỆ SỬ DỤNG** 3](#_Toc185468169)

[**2.1 Giới thiệu về Apache Spark** 3](#_Toc185468170)

[**2.1.1 Apache Spark là gì** 3](#_Toc185468171)

[**2.1.2 Lịch sử phát triển của Apache Spark** 3](#_Toc185468172)

[**2.1.3 Apache Spark hoạt động như thế nào?** 4](#_Toc185468173)

[**2.1.4 Sự khác biệt chính giữa Apache Spark và Apache Hadoop** 4](#_Toc185468174)

[**2.1.5 Lợi ích của Apache Spark** 5](#_Toc185468175)

[**2.1.6 Ưu điểm của Apache Spark** 6](#_Toc185468176)

[**2.1.7 Nhược điểm của Apache Spark** 7](#_Toc185468177)

[**2.1.8 Ứng dụng mạnh mẽ của Apache Spark** 8](#_Toc185468178)

[**2.2 Giới thiệu về Power BI** 9](#_Toc185468179)

[**2.2.1 Power BI là gì** 9](#_Toc185468180)

[**2.2.2 Lịch sử phát triển** 9](#_Toc185468181)

[**2.2.3 Cách thức hoạt động** 10](#_Toc185468182)

[**2.2.4 Ưu nhược điểm** 11](#_Toc185468183)

[**2.2.5 Lợi ích của Power BI** 12](#_Toc185468184)

[**CHƯƠNG 3: TRIỂN KHAI ĐỀ TÀI** 14](#_Toc185468185)

[**3.1** **Tổng quan tập dữ liệu** 14](#_Toc185468186)

[**3.2 Mô tả tập dữ liệu** 14](#_Toc185468187)

[**3.3 Các bước thực hiện đề tài** 16](#_Toc185468188)

[**3.3.1 Tiền xử lí dữ liệu** 16](#_Toc185468189)

[**3.3.2 Phân tích thành các file cvs để trực quan hóa đữ liệu** 20](#_Toc185468190)

[**3.3.2.1 Phân tích mức độ tai nạn theo thời tiết** 20](#_Toc185468191)

[**3.3.2.2 Phân tích mức độ tai nạn theo cơ sở hạ tầng** 22](#_Toc185468192)

[**3.3.2.3 Phân tích mức độ tai nạn theo địa lý** 22](#_Toc185468193)

[**3.3.2.4 Phân tích mức độ tai nạn theo thời gian** 23](#_Toc185468194)

[**3.3.3 Trực quan hóa dữ liệu bằng Power BI** 25](#_Toc185468195)

[**3.3.3.1 Trực quan tổng quát** 25](#_Toc185468196)

[**3.3.3.2 Trực quan hóa theo ảnh hưởng thời tiết** 26](#_Toc185468197)

[**3.3.3.3 Trực quan hóa theo ảnh hưởng cơ sở hạ tầng** 27](#_Toc185468198)

[**3.3.3.4 Trực quan hóa theo ảnh hưởng thời gian** 28](#_Toc185468199)

[**3.3.3.5 Trực quan hóa theo ảnh hưởng vị trí địa lí** 29](#_Toc185468200)

[**CHƯƠNG 4: BÀI TẬP CƠ BẢN VỀ APACHE SPARK** 36](#_Toc185468201)

[**CHƯƠNG 5: KẾT LUẬN** 111](#_Toc185468202)

[**5.1 Kết quả đạt được** 111](#_Toc185468203)

[**5.2 Ưu điểm** 111](#_Toc185468204)

[**5.3 Hạn chế** 111](#_Toc185468205)

[**5.4 Hướng phát triển** 111](#_Toc185468206)

[**TÀI LIỆU THAM KHẢO** 112](#_Toc185468207)

# **CHƯƠNG 1: GIỚI THIỆU**

* 1. **Giới thiệu dự án**

Trong thời kỳ phát triển hiện nay, các tổ chức và cơ quan quản lý giao thông cần thu thập và xử lý dữ liệu từ nhiều nguồn, thiết bị, khu vực và vùng miền khác nhau. Việc xử lý và phân tích dữ liệu là vô cùng cần thiết để có thể giảm thiểu tình trạng tai nạn giao thông, đồng thời hỗ trợ công tác bảo trì và nâng cấp cơ sở hạ tầng giao thông như đường xá, bờ đê một cách phù hợp theo từng mùa và đặc điểm riêng của từng khu vực.

Dự án phân tích tai nạn giao thông ở Mỹ và trực quan hóa bằng power BI được xây dựng với mục đích phân tích các yếu tố liên quan đến tai nạn như: địa điểm, thời gian, điều kiện thời tiết và các mức độ nghiêm trọng xảy ra hằng ngày. Thông qua việc đánh giá và xử lý dữ liệu từ thực tế, dự án sẽ cung cấp cái nhìn sâu sắc về các nguyên nhân và điều kiện làm gia tăng tỷ lệ tai nạn, từ đó đề xuất các giải pháp nhằm hạn chế rủi ro.

Bối cảnh hiện nay cho thấy tình trạng biến đổi khí hậu, thời tiết cực đoan và yếu tố chủ quan từ con người như thiếu tập trung, không tuân thủ quy định an toàn giao thông,... là những nguyên nhân chủ yếu dẫn đến việc tai nạn xảy ra ngày càng nhiều. Vì vậy, dự án hướng đến việc ứng dụng các phương pháp phân tích và trực quan hóa dữ liệu để xác định các khu vực có nguy cơ cao, xu hướng tai nạn theo mùa vụ, cũng như các điều kiện môi trường ảnh hưởng đến an toàn giao thông.

Kết quả của dự án sẽ là cơ sở dữ liệu quan trọng hỗ trợ các cơ quan quản lý trong việc:

* Nâng cao hiệu quả quản lý giao thông và cải thiện an toàn đường bộ.
* Lập kế hoạch bảo trì, nâng cấp hạ tầng giao thông dựa trên các yếu tố thời tiết và đặc điểm khu vực.
* Tuyên truyền và giáo dục ý thức tham gia giao thông cho người dân nhằm giảm thiểu các nguyên nhân chủ quan dẫn đến tai nạn.

## **1.2 Mục tiêu**

Tìm hiểu và học hỏi về Apache Spark.

Mục tiêu của dự án là chuẩn hóa, phân tích dữ liệu và trực quan hóa dữ liệu. Chúng ta sẽ thực hiện bằng công cụ Jupyter Lab, Power BI. Đầu tiên ta sẽ chuẩn hóa dữ liệu và phân tích dữ liệu theo thời gian, thời tiết, địa điểm, cơ sở hạ tầng , sau đó sẽ sử dụng Power BI để trực quan hóa dữ liệu bằng các biểu đồ.

## **1.3 Nhiệm vụ**

* Quản lý và tiền xử lý dữ liệu: Thu thập, làm sạch, và tiền xử lý dữ liệu.
* Phân tích tai nạn: Phân tích các yếu tố ảnh hưởng đến tai nạn giao thông.
* Trực quan hóa dữ liệu: Tạo các báo cáo và giao diện web để trình bày kết quả phân tích.
* Triển khai hệ thống: Đảm bảo hệ thống xử lý và phân tích dữ liệu hoạt động ổn định.

# **CHƯƠNG 2: CÔNG NGHỆ SỬ DỤNG**

## **2.1 Giới thiệu về Apache Spark**

### **2.1.1 Apache Spark là gì**

Apache Spark là một hệ thống xử lý phân tán nguồn mở được thiết kế để xử lý các khối lượng dữ liệu lớn với hiệu suất cao. Nó được tối ưu hóa để xử lý dữ liệu nhanh chóng bằng cách sử dụng bộ nhớ đệm và thực thi các truy vấn tối ưu hóa, giúp xử lý dữ liệu có kích thước lớn một cách hiệu quả. Spark hỗ trợ các API phát triển bằng các ngôn ngữ lập trình như **Java**, **Scala**, **Python**, và **R**, cho phép tái sử dụng mã nguồn cho nhiều loại khối lượng công việc khác nhau, từ xử lý dữ liệu theo lô, truy vấn dữ liệu tương tác, phân tích thời gian thực, đến máy học và xử lý đồ thị.

Spark có thể được sử dụng trong nhiều lĩnh vực và đã được các công ty lớn như **FINRA**, **Yelp**, **Zillow**, **DataXu**, **Urban Institute**, và **CrowdStrike** ứng dụng.

### **2.1.2 Lịch sử phát triển của Apache Spark**

Apache Spark bắt đầu từ năm 2009 như một dự án nghiên cứu tại AMPLab của **UC Berkeley**, với sự tham gia của các sinh viên, giảng viên, và các nhà nghiên cứu. Mục tiêu của Spark là phát triển một khung công tác (framework) tối ưu hóa cho các ứng dụng xử lý dữ liệu lớn, đặc biệt là các tác vụ cần xử lý lặp lại nhanh như **máy học** và **phân tích dữ liệu tương tác**, đồng thời vẫn giữ được khả năng chịu lỗi và khả năng mở rộng của **Hadoop MapReduce**.

* Báo cáo nghiên cứu đầu tiên mang tên "Spark: Cluster Computing with Working Sets" đã được công bố vào tháng 6 năm 2010.
* Spark trở thành dự án nguồn mở dưới giấy phép BSD.
* Vào tháng 6 năm 2013, Spark trở thành dự án ươm tạo của Quỹ Phần mềm Apache (ASF).
* Vào tháng 2 năm 2014, Spark được nâng cấp thành Dự án cấp cao nhất của Apache.
* Spark có thể hoạt động độc lập hoặc chạy trên **Apache Mesos** hoặc **Hadoop**.

### **2.1.3 Apache Spark hoạt động như thế nào?**

**So với Hadoop MapReduce:**

**Hadoop MapReduce** là mô hình lập trình sử dụng thuật toán phân tán song song để xử lý các tập dữ liệu lớn. Tuy nhiên, nó có một số hạn chế lớn:

* **Độ trễ I/O đĩa**: Mỗi bước trong quy trình MapReduce yêu cầu đọc và ghi dữ liệu vào HDFS (Hệ thống Tệp Phân tán Hadoop), gây ra độ trễ cao do thao tác với đĩa.
* **Quy trình tuần tự**: Để hoàn thành một công việc, MapReduce yêu cầu thực hiện nhiều bước tuần tự, điều này làm giảm tốc độ xử lý.

**Apache Spark** giải quyết các vấn đề này bằng cách:

* **Xử lý trong bộ nhớ**: Dữ liệu được nạp vào bộ nhớ, và các thao tác được thực hiện trực tiếp trong bộ nhớ thay vì phải đọc và ghi vào đĩa.
* **Giảm số bước trong công việc**: Spark giảm số bước cần thiết để xử lý một công việc, từ đó giảm độ trễ tổng thể.
* **Sử dụng lại dữ liệu**: Spark có khả năng tái sử dụng dữ liệu trong bộ nhớ thông qua các **DataFrames** và **RDDs (Resilient Distributed Datasets)**, giúp tăng tốc các thuật toán máy học và phân tích lặp lại.

### **2.1.4 Sự khác biệt chính giữa Apache Spark và Apache Hadoop**

Mặc dù Apache Spark và Hadoop có thể cùng được sử dụng trong một hệ thống phân tán, chúng có các chức năng khác nhau:

* **Hadoop** là một hệ sinh thái gồm nhiều công cụ, với **HDFS** là hệ thống lưu trữ phân tán và **YARN** quản lý tài nguyên tính toán. **MapReduce** là công cụ thực thi chính của Hadoop.
* **Apache Spark** là một hệ thống xử lý dữ liệu nhanh, được tối ưu hóa cho các ứng dụng như **truy vấn tương tác**, **máy học**, **phân tích thời gian thực**, và **xử lý đồ thị**. Spark không có hệ thống lưu trữ riêng, nhưng có thể chạy trên **HDFS**, **Cassandra**, **Amazon S3**, hoặc các hệ thống kho dữ liệu khác.

Khi chạy trên Hadoop, Spark sử dụng **YARN** để chia sẻ tài nguyên và tập dữ liệu giữa các ứng dụng khác nhau, giúp Spark và các công cụ Hadoop khác có thể phối hợp trong cùng một hệ thống.

### **2.1.5 Lợi ích của Apache Spark**

Apache Spark mang lại nhiều lợi ích vượt trội, khiến nó trở thành một công cụ quan trọng trong hệ sinh thái dữ liệu lớn:

* **Nhanh chóng**: Spark sử dụng bộ nhớ đệm trong bộ nhớ (in-memory caching) và thực thi các truy vấn tối ưu hóa, giúp xử lý dữ liệu nhanh hơn nhiều so với Hadoop MapReduce, đặc biệt khi làm việc với các khối lượng công việc phức tạp.
* **Thân thiện với nhà phát triển**: Spark hỗ trợ nhiều ngôn ngữ lập trình như **Java**, **Scala**, **Python**, và **R**, giúp các nhà phát triển dễ dàng xây dựng và triển khai ứng dụng. API của Spark trừu tượng hóa sự phức tạp của xử lý phân tán và giảm thiểu mã nguồn cần viết.
* **Khả năng xử lý nhiều khối lượng công việc**: Spark có thể xử lý nhiều loại khối lượng công việc khác nhau, bao gồm:
  + **Truy vấn tương tác**: Thực hiện truy vấn dữ liệu trực tiếp và nhanh chóng.
  + **Phân tích thời gian thực**: Xử lý các luồng dữ liệu và phân tích thời gian thực.
  + **Máy học**: Được sử dụng để huấn luyện và triển khai các mô hình học máy.
  + **Xử lý đồ thị**: Thực hiện các phân tích trên đồ thị dữ liệu (ví dụ: phân tích mạng xã hội, các thuật toán đồ thị).

Những đặc điểm này giúp Spark trở thành công cụ phù hợp với các ứng dụng đòi hỏi tốc độ và khả năng xử lý dữ liệu lớn một cách linh hoạt và hiệu quả.

### **2.1.6 Ưu điểm của Apache Spark**

1. **Tốc độ xử lý cao**:
   * **Xử lý trong bộ nhớ (in-memory processing)**: Spark sử dụng bộ nhớ RAM để lưu trữ dữ liệu thay vì phải liên tục đọc và ghi vào đĩa như Hadoop MapReduce. Điều này giúp giảm thiểu độ trễ và tăng tốc đáng kể các tác vụ xử lý dữ liệu.
   * **Tối ưu hóa truy vấn**: Spark có khả năng tối ưu hóa các truy vấn và giảm số lần ghi và đọc vào đĩa, giúp thực hiện các tác vụ nhanh hơn nhiều lần so với Hadoop MapReduce.
2. **Dễ dàng lập trình và phát triển**:
   * **Hỗ trợ nhiều ngôn ngữ lập trình**: Spark cung cấp API cho **Java**, **Scala**, **Python**, và **R**, giúp các nhà phát triển có thể chọn ngôn ngữ lập trình phù hợp với họ.
   * **API dễ sử dụng**: Spark cung cấp các API cấp cao, giúp nhà phát triển dễ dàng xử lý dữ liệu phân tán mà không cần phải lo lắng về việc quản lý phân phối công việc hoặc xử lý lỗi.
3. **Khả năng xử lý nhiều loại dữ liệu và khối lượng công việc**:
   * **Xử lý lô và thời gian thực**: Spark có thể xử lý dữ liệu theo lô (batch processing) và dữ liệu thời gian thực (streaming) trong cùng một hệ thống.
   * **Máy học và xử lý đồ thị**: Spark có thư viện **MLlib** để hỗ trợ máy học và **GraphX** để xử lý đồ thị, giúp giải quyết các bài toán phân tích dữ liệu phức tạp.
4. **Khả năng mở rộng cao**:
   * Spark có thể mở rộng để chạy trên hàng nghìn nút trong một cụm, cho phép xử lý khối lượng dữ liệu lớn một cách hiệu quả.
5. **Tích hợp với các công nghệ khác**:
   * Spark có thể tích hợp với nhiều hệ thống lưu trữ khác nhau như **HDFS**, **Cassandra**, **Amazon S3**, **HBase**, và các cơ sở dữ liệu khác, giúp dễ dàng lưu trữ và truy xuất dữ liệu.

### **2.1.7 Nhược điểm của Apache Spark**

1. **Sử dụng bộ nhớ nhiều**:
   * **Tiêu thụ bộ nhớ**: Do Spark sử dụng bộ nhớ để lưu trữ dữ liệu trong suốt quá trình xử lý, nếu cụm máy chủ có hạn chế về bộ nhớ, Spark có thể gặp phải vấn đề với việc xử lý dữ liệu lớn hoặc với các tác vụ yêu cầu bộ nhớ cao.
   * **Quản lý bộ nhớ phức tạp**: Việc tối ưu hóa và quản lý bộ nhớ khi chạy Spark trên một cụm lớn có thể phức tạp, đặc biệt trong môi trường có dữ liệu rất lớn.
2. **Khó triển khai và duy trì**:
   * **Quản lý cluster**: Quản lý và triển khai Spark trên một cụm phân tán đòi hỏi các kiến thức kỹ thuật sâu về hệ thống phân tán và cấu hình cụm. Việc duy trì và giám sát cụm cũng có thể gặp khó khăn đối với các tổ chức nhỏ hoặc không có đội ngũ kỹ thuật mạnh.
3. **Hỗ trợ tính toán phân tán không hoàn hảo**:
   * **Không hoàn hảo khi chạy với các hệ thống tệp phân tán**: Mặc dù Spark có thể chạy trên nhiều hệ thống tệp phân tán (như HDFS), nhưng không phải lúc nào nó cũng tối ưu khi làm việc với các hệ thống này. Một số tác vụ vẫn có thể gặp độ trễ cao nếu không được cấu hình đúng.
4. **Phức tạp trong việc xử lý các tác vụ trạng thái**:
   * **Tác vụ trạng thái trong Spark Streaming**: Spark Streaming cung cấp khả năng xử lý dữ liệu theo thời gian thực, nhưng việc duy trì trạng thái và đảm bảo tính chính xác của dữ liệu trong các ứng dụng streaming có thể trở nên phức tạp và đòi hỏi kỹ thuật cao.

### **2.1.8 Ứng dụng mạnh mẽ của Apache Spark**

Apache Spark được ứng dụng mạnh mẽ trong nhiều lĩnh vực và khối ngành, đặc biệt là khi xử lý và phân tích các khối lượng dữ liệu lớn. Một số ứng dụng phổ biến của Spark bao gồm:

1. **Xử lý và phân tích dữ liệu lớn (Big Data Analytics)**:
   * Spark được sử dụng rộng rãi để xử lý và phân tích các tập dữ liệu rất lớn. Các ứng dụng phân tích dữ liệu trong tài chính, y tế, và thương mại điện tử đều tận dụng Apache Spark để phân tích các dữ liệu lớn và phức tạp.
2. **Máy học (Machine Learning)**:
   * **MLlib** là thư viện máy học của Spark, giúp triển khai các thuật toán máy học trên dữ liệu phân tán. Spark được sử dụng để xây dựng các mô hình học máy như phân loại, hồi quy, phân cụm và giảm chiều.
3. **Phân tích dữ liệu theo thời gian thực (Real-time Data Analytics)**:
   * Spark Streaming là một công cụ mạnh mẽ để xử lý dữ liệu theo thời gian thực. Nó giúp các tổ chức phân tích các luồng dữ liệu trong thời gian thực và đưa ra các quyết định nhanh chóng (ví dụ: trong các ứng dụng phân tích log, giám sát mạng, hay phân tích sự kiện trực tuyến).
4. **Xử lý dữ liệu đồ thị (Graph Processing)**:
   * **GraphX** là một thư viện trong Spark giúp xử lý dữ liệu đồ thị. Nó được sử dụng trong các ứng dụng như phân tích mạng xã hội, tìm kiếm bạn bè, hoặc phân tích mạng lưới giao thông.
5. **Xử lý dữ liệu từ các nguồn khác nhau**:
   * Spark có thể dễ dàng tích hợp với nhiều hệ thống lưu trữ dữ liệu như **HDFS**, **Cassandra**, **Amazon S3**, **HBase**, và nhiều hệ thống lưu trữ khác, giúp dễ dàng khai thác và xử lý dữ liệu từ nhiều nguồn.
6. **Quản lý và phân tích log**:
   * Spark được sử dụng để phân tích các file log lớn trong các ứng dụng giám sát hệ thống và phân tích dữ liệu mạng. Các công ty như **Yelp**, **Netflix**, và **Uber** sử dụng Spark để phân tích và xử lý các tập log khổng lồ của họ.
7. **Tạo và phân phối các báo cáo tương tác**:
   * Spark có thể được kết hợp với các công cụ phân tích dữ liệu như **Tableau** và **PowerBI** để tạo ra các báo cáo phân tích dữ liệu động và tương tác, hỗ trợ việc ra quyết định trong thời gian thực.

## **2.2 Giới thiệu về Power BI**

### **2.2.1 Power BI là gì**

Power BI là một bộ công cụ phần mềm do Microsoft phát triển, dùng để phân tích và trực quan hóa dữ liệu (Business Intelligence). Công cụ này giúp người dùng kết nối với nhiều nguồn dữ liệu khác nhau, chuyển đổi chúng thành các báo cáo và biểu đồ trực quan để phân tích và chia sẻ thông tin dễ dàng. Power BI đặc biệt phổ biến trong việc hỗ trợ ra quyết định nhanh chóng, chính xác dựa trên dữ liệu thực tế.

Power BI được thiết kế để dễ dàng sử dụng, từ người dùng không chuyên cho đến các chuyên gia dữ liệu, giúp họ tạo báo cáo phân tích và chia sẻ dữ liệu hiệu quả. Phần mềm này bao gồm ba thành phần chính: Power BI Desktop, Power BI Service, và Power BI Mobile.

### **2.2.2 Lịch sử phát triển**

Power BI được Microsoft ra mắt lần đầu vào năm 2013, trong nỗ lực cung cấp một công cụ phân tích dữ liệu trực quan và dễ sử dụng cho các doanh nghiệp. Trước đó, Microsoft đã phát triển một số công cụ BI khác, nhưng Power BI đã mang lại sự đổi mới lớn trong cách thức người dùng có thể tiếp cận và phân tích dữ liệu mà không cần phải có kỹ năng lập trình phức tạp.

Trải qua nhiều năm phát triển, Power BI hiện nay đã trở thành công cụ BI hàng đầu và đóng góp lớn vào ngành phân tích dữ liệu toàn cầu. Power BI đã góp phần tạo ra một vị trí công việc mới – Data Analyst (Chuyên viên phân tích dữ liệu), với nhu cầu ngày càng tăng trong việc ứng dụng các công cụ BI để hỗ trợ ra quyết định trong các tổ chức.

### **2.2.3 Cách thức hoạt động**

Power BI hoạt động thông qua ba thành phần chính, mỗi thành phần phục vụ một mục đích khác nhau trong quá trình phân tích dữ liệu:

Power BI Desktop

* Đây là ứng dụng miễn phí cài đặt trên máy tính, dùng để phát triển và tạo các báo cáo, dashboard. Power BI Desktop cho phép người dùng:
  + Kết nối đến các nguồn dữ liệu khác nhau (Excel, cơ sở dữ liệu SQL, API, v.v.).
  + Làm sạch và biến đổi dữ liệu thông qua Power Query Editor.
  + Tạo mô hình dữ liệu (data model) phù hợp với yêu cầu phân tích.
  + Xây dựng báo cáo, dashboard với các biểu đồ và bảng tương tác.

Power BI Service

* Đây là phiên bản web của Power BI, phục vụ cho việc chia sẻ, quản lý và phân phối các báo cáo đã tạo từ Power BI Desktop. Power BI Service cung cấp một số tính năng như:
  + Chia sẻ các báo cáo và dashboard qua "Workspace" (nhóm chia sẻ).
  + Cập nhật tự động dữ liệu theo định kỳ.
  + Thiết lập các báo cáo email và thông báo (tính năng Subscribe).
  + Tương tác với báo cáo qua các tính năng Comment, Chat trong Teams, v.v.
  + Phân quyền người dùng để kiểm soát quyền truy cập vào các báo cáo.

Power BI Mobile

* Là ứng dụng trên điện thoại di động, cho phép người dùng xem và tương tác với các báo cáo Power BI Service. Power BI Mobile thích hợp với những người dùng cần truy cập nhanh các báo cáo khi di chuyển, nhưng cần thiết kế giao diện phù hợp với màn hình nhỏ của điện thoại.

### **2.2.4 Ưu nhược điểm**

Ưu điểm:

* Dễ sử dụng: Power BI có giao diện người dùng đơn giản, dễ học và sử dụng. Ngay cả những người không chuyên về phân tích dữ liệu cũng có thể nhanh chóng nắm bắt cách tạo báo cáo và dashboard.
* Khả năng kết nối dữ liệu mạnh mẽ: Power BI hỗ trợ kết nối với hàng loạt nguồn dữ liệu, từ các file đơn giản như Excel đến các hệ thống phức tạp như ERP, CRM, cơ sở dữ liệu SQL và các dịch vụ đám mây.
* Tính trực quan: Power BI cung cấp nhiều lựa chọn biểu đồ và báo cáo trực quan, giúp người dùng dễ dàng hiểu được dữ liệu và đưa ra quyết định.
* Hỗ trợ mạnh mẽ từ Microsoft: Microsoft cung cấp các tài liệu hỗ trợ chi tiết và có cộng đồng người dùng rộng lớn, giúp giải quyết các vấn đề trong quá trình sử dụng.
* Tích hợp với các công cụ Microsoft: Power BI tích hợp tốt với các công cụ khác của Microsoft như Excel, Azure, và Office 365.

Nhược điểm:

* Giới hạn trong phiên bản miễn phí: Power BI có phiên bản miễn phí nhưng không có nhiều tính năng nâng cao. Để sử dụng các tính năng như chia sẻ báo cáo hoặc cộng tác trong nhóm, người dùng phải nâng cấp lên các gói trả phí.
* Yêu cầu Internet ổn định: Power BI Service cần kết nối Internet để chia sẻ và truy cập báo cáo. Điều này có thể gặp khó khăn nếu không có kết nối ổn định.
* Khả năng xử lý dữ liệu lớn: Với một số lượng dữ liệu rất lớn, Power BI có thể gặp phải một số vấn đề về hiệu suất, đặc biệt là khi không được tối ưu hóa đúng cách.

### **2.2.5 Lợi ích của Power BI**

* Tiết kiệm thời gian: Power BI giúp tự động hóa nhiều quy trình báo cáo và phân tích, giúp tiết kiệm thời gian và giảm bớt công việc thủ công.
* Ra quyết định dựa trên dữ liệu: Với các báo cáo trực quan, Power BI giúp người dùng dễ dàng hiểu được các xu hướng và yếu tố quan trọng trong dữ liệu để đưa ra các quyết định kinh doanh chính xác.
* Tăng cường hợp tác: Power BI cho phép chia sẻ báo cáo và mô hình dữ liệu với các bên liên quan, tạo điều kiện cho sự hợp tác và trao đổi thông tin hiệu quả trong tổ chức.
* Phân quyền truy cập: Người quản lý có thể phân quyền cho các nhóm khác nhau, đảm bảo chỉ những người có quyền mới có thể truy cập và chỉnh sửa các báo cáo quan trọng.
* Dễ dàng tùy chỉnh và mở rộng: Power BI cho phép người dùng tùy chỉnh báo cáo theo nhu cầu cụ thể của họ và mở rộng tính năng khi cần thiết thông qua các gói trả phí và tích hợp thêm các dịch vụ bên ngoài.

- Là phần mềm phân tích dữ liệu thông dụng nhất hiện nay bởi tính dễ dùng cùng các tính năng mạnh mẽ vô cùng hiệu quả trong việc xử lý và phân tích dữ liệu  
  
- Được sử dụng rộng rãi trong đa ngành nghề, đa lĩnh vực: từ vị trí kinh doanh, marketing, phát triển sản phẩm cho tới các ngành nghề như tài chính, giáo dục, vận tải, logistic,...

# **CHƯƠNG 3: TRIỂN KHAI ĐỀ TÀI**

* 1. **Tổng quan tập dữ liệu**

Đây là bộ dữ liệu về tai nạn xe hơi trên toàn nước Mỹ, bao gồm 49 tiểu bang. Dữ liệu tai nạn được thu thập từ tháng 2 năm 2016 đến tháng 3 năm 2023 thông qua nhiều API cung cấp dữ liệu sự cố giao thông theo thời gian thực. Các API này phát dữ liệu giao thông được thu thập bởi nhiều tổ chức khác nhau, bao gồm các Sở Giao thông Vận tải liên bang và tiểu bang, các cơ quan thực thi pháp luật, camera giao thông và các cảm biến giao thông trong mạng lưới đường bộ. Hiện tại, bộ dữ liệu chứa khoảng 7,7 triệu bản ghi tai nạn.

Dữ liệu có cấu trúc: US\_Accidents.csv

Nguồn: <https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents/data>

* 1. **Mô tả tập dữ liệu**

Tập dữ liệu “US\_Accidents.csv” bao gồm các thông tin tại nạn giao thông ở Mỹ từ năm 2016 đến năm 2023. Dữ liệu gốc gồm 1 file csv(gồm 500001 dòng, 46 cột), gồm các cột:

|  |  |  |
| --- | --- | --- |
| **STT** | **Cột** | **Mô tả** |
| 1 | ID | Mã định danh duy nhất cho mỗi bản ghi tai nạn. |
| 2 | Source | Nguồn dữ liệu thu thập (ví dụ: API hoặc cảm biến). |
| 3 | Severity | Mức độ nghiêm trọng của tai nạn (thang đo mức độ). |
| 4 | Start\_Time | Thời gian bắt đầu của sự cố. |
| 5 | End\_Time | Thời gian kết thúc của sự cố. |
| 6 | Start\_Lat | Vĩ độ tại vị trí bắt đầu của sự cố. |
| 7 | Start\_Lng | Kinh độ tại vị trí bắt đầu của sự cố. |
| 8 | End\_Lat | Vĩ độ tại vị trí kết thúc của sự cố. |
| 9 | End\_Lng | Kinh độ tại vị trí kết thúc của sự cố. |
| 10 | Distance(mi) | Khoảng cách của sự cố tính bằng dặm. |
| 11 | Description | Mô tả chi tiết về sự cố giao thông. |
| 12 | Street | Tên đường xảy ra sự cố. |
| 13 | City | Thành phố nơi sự cố xảy ra. |
| 14 | County | Quận nơi sự cố xảy ra. |
| 15 | State | Tiểu bang nơi sự cố xảy ra. |
| 16 | Zipcode | Mã bưu chính nơi sự cố xảy ra. |
| 17 | Country | Quốc gia (ví dụ: USA). |
| 18 | Timezone | Múi giờ của vị trí xảy ra sự cố. |
| 19 | Airport\_Code | Mã sân bay gần vị trí sự cố nhất. |
| 20 | Weather\_Timestamp | Thời gian ghi nhận điều kiện thời tiết liên quan. |
| 21 | Temperature(F) | Nhiệt độ tại thời điểm xảy ra sự cố (độ F). |
| 22 | Wind\_Chill(F) | Độ lạnh do gió tại thời điểm xảy ra sự cố (độ F). |
| 23 | Humidity(%) | Độ ẩm tính theo phần trăm tại thời điểm xảy ra sự cố. |
| 24 | Pressure(in) | Áp suất không khí (inch). |
| 25 | Visibility(mi) | Tầm nhìn xa tại thời điểm xảy ra sự cố (dặm). |
| 26 | Wind\_Direction | Hướng gió tại thời điểm xảy ra sự cố. |
| 27 | Wind\_Speed(mph) | Tốc độ gió (dặm/giờ). |
| 28 | Precipitation(in) | Lượng mưa tại thời điểm xảy ra sự cố (inch). |
| 29 | Weather\_Condition | Điều kiện thời tiết (mưa, tuyết, sương mù,...). |
| 30 | Amenity | Có tiện ích gần đó hay không (True/False). |
| 31 | Bump | Có gờ giảm tốc gần đó hay không (True/False). |
| 32 | Crossing | Có vạch qua đường gần đó hay không (True/False). |
| 33 | Give\_Way | Có biển nhường đường gần đó hay không (True/False). |
| 34 | Junction | Có ngã giao gần đó hay không (True/False). |
| 35 | No\_Exit | Có khu vực không lối thoát gần đó hay không (True/False). |
| 36 | Railway | Có đường ray gần đó hay không (True/False). |
| 37 | Roundabout | Có vòng xuyến gần đó hay không (True/False). |
| 38 | Station | Có trạm gần đó hay không (True/False). |
| 39 | Stop | Có biển dừng gần đó hay không (True/False). |
| 40 | Traffic\_Calming | Có thiết bị giảm tốc gần đó hay không (True/False). |
| 41 | Traffic\_Signal | Có đèn giao thông gần đó hay không (True/False). |
| 42 | Turning\_Loop | Có vòng quay đầu gần đó hay không (True/False). |
| 43 | Sunrise\_Sunset | Thời điểm mặt trời (Day/Night). |
| 44 | Civil\_Twilight | Thời gian chạng vạng dân dụng (Day/Night). |
| 45 | Nautical\_Twilight | Thời gian chạng vạng hàng hải (Day/Night). |
| 46 | Astronomical\_Twilight | Thời gian chạng vạng thiên văn (Day/Night). |

* 1. **Các bước thực hiện đề tài**
* Tiền xử lí dữ liệu
* Phân tích thành các file cvs nhỏ để trực quan hóa đữ liệu
* Trực quan hóa dữ liệu bằng Power BI

**3.3.1 Tiền xử lí dữ liệu**

|  |
| --- |
| 1. from pyspark.sql import SparkSession 2. from pyspark.sql.functions import \* 3. # Tạo SparkSession 4. spark = SparkSession.builder \ 5. .appName("USTrafficAccidents") \ 6. .getOrCreate() |

Import thư viện: Import các lớp và hàm cần thiết từ PySpark.

Tạo SparkSession:

* Dùng SparkSession.builder để khởi tạo phiên làm việc với Spark.
* Đặt tên ứng dụng là "USTrafficAccidents".
* Tạo một phiên bản SparkSession bằng .getOrCreate()

|  |
| --- |
| file\_path = "dataset/US\_Accidents.csv"  df = spark.read.csv(file\_path, header=True, inferSchema=True) |

Đường dẫn tệp: file\_path trỏ đến tệp CSV có tên US\_Accidents.csv nằm trong thư mục dataset.

Đọc dữ liệu:

* spark.read.csv() dùng để đọc tệp CSV.
* header=True: Dòng đầu tiên trong tệp CSV được coi là tên cột.
* inferSchema=True: Tự động xác định kiểu dữ liệu cho các cột.

Kết quả: Dữ liệu được lưu vào DataFrame có tên df.

|  |
| --- |
| null\_counts = df.select([sum(when(col(c).isNull(), 1).otherwise(0)).alias(c) for c in df.columns])  # Hiển thị kết quả  null\_counts.show() |

Kiểm tra giá trị null:

* Với từng cột trong DataFrame df, hàm when(col(c).isNull(), 1).otherwise(0) sẽ trả về 1 nếu giá trị là null, ngược lại trả về 0.
* sum() sẽ tính tổng số lượng giá trị null của từng cột.

Tạo DataFrame null\_counts:

* Sử dụng select() để tạo một cột mới cho mỗi cột trong df với số lượng giá trị null được tính.
* alias(c) đặt tên cột kết quả trùng với tên cột gốc.

Hiển thị kết quả:

* null\_counts.show() hiển thị số lượng giá trị null của từng cột trong DataFrame.

🡺Sau khi kiểm tra các cột null, sẽ loại bỏ những cột có số lượng quá lớn NULL, những cột có số lượng NULL không qua lớn sẽ thay thế bằng các giá trị phù hợp.

|  |
| --- |
| def fill\_date\_for\_missing(df):      # Lấy ngày tháng năm từ Weather\_Timestamp      df = df.withColumn("Start\_Date", date\_format("Weather\_Timestamp", "M/d/yyyy"))      df = df.withColumn("End\_Date", date\_format("Weather\_Timestamp", "M/d/yyyy"))      # Thêm ngày tháng năm vào Start\_Time và End\_Time nếu thiếu      df = df.withColumn(          "Start\_Time",          when(              col("Start\_Time").rlike("^[0-9]{1,2}:[0-9]{2}(\.[0-9]+)?$"),              concat(col("Start\_Date"), lit(" 00:"), regexp\_replace(col("Start\_Time"), r":[0-9]{2}(\.[0-9]+)?$", ""))          ).otherwise(col("Start\_Time"))      )      df = df.withColumn(          "End\_Time",          when(              col("End\_Time").rlike("^[0-9]{1,2}:[0-9]{2}(\.[0-9]+)?$"),              concat(col("End\_Date"), lit(" 00:"), regexp\_replace(col("End\_Time"), r":[0-9]{2}(\.[0-9]+)?$", ""))          ).otherwise(col("End\_Time"))      )      # Chuyển đổi cột Start\_Time\_Filled và End\_Time\_Filled thành kiểu timestamp      df = df.withColumn("Start\_Time", to\_timestamp("Start\_Time", "M/d/yyyy H:mm"))      df = df.withColumn("End\_Time", to\_timestamp("End\_Time", "M/d/yyyy H:mm"))        return df  # Áp dụng hàm vào DataFrame  df = fill\_date\_for\_missing(df)  # Bỏ 2 cột Start\_Date và End\_Date  df = df.drop("Start\_Date", "End\_Date") |

**Hàm fill\_date\_for\_missing** xử lý các giá trị thiếu hoặc không đầy đủ trong cột Start\_Time và End\_Time bằng cách:

* Tạo cột tạm thời Start\_Date và End\_Date: Lấy ngày từ cột Weather\_Timestamp và định dạng theo kiểu M/d/yyyy.
* Điền ngày vào Start\_Time và End\_Time nếu chỉ có thời gian:
  + Nếu Start\_Time hoặc End\_Time chỉ chứa thời gian (dạng HH:mm), hàm sẽ ghép ngày từ Start\_Date/End\_Date và gán thời gian đó vào cột.
  + Dùng các hàm rlike, concat, và regexp\_replace để kiểm tra và xử lý chuỗi thời gian.
* Chuyển đổi kiểu dữ liệu: Chuyển đổi Start\_Time và End\_Time sang kiểu timestamp (M/d/yyyy H:mm) để đảm bảo tính nhất quán và dễ thao tác.
* Xóa các cột tạm thời: Cột Start\_Date và End\_Date được xóa sau khi xử lý xong để giữ DataFrame gọn gàng.

|  |
| --- |
| df = df.fillna({      "Description": "No Description",      "Street": "Unknown",      "City": "Unknown",      "Weather\_Condition": "Unknown",      "Wind\_Direction": "Unknown"  })  # Điền giá trị cho các cột boolean  boolean\_columns = [      "Amenity", "Bump", "Crossing", "Give\_Way", "Junction", "No\_Exit",      "Railway", "Roundabout", "Station", "Stop", "Traffic\_Calming",      "Traffic\_Signal", "Turning\_Loop"  ]  for col\_name in boolean\_columns:      df = df.withColumn(col\_name, when(col(col\_name).isNull(), lit(False)).otherwise(col(col\_name)))  # Điền giá trị cho các cột số liệu với giá trị trung bình  numeric\_columns = [      "Severity", "Distance(mi)", "Temperature(F)", "Wind\_Chill(F)",      "Humidity(%)", "Pressure(in)", "Visibility(mi)", "Wind\_Speed(mph)",      "Precipitation(in)"  ]  for col\_name in numeric\_columns:      mean\_value = df.select(mean(col(col\_name))).collect()[0][0]      df = df.withColumn(col\_name, when(col(col\_name).isNull(), lit(mean\_value)).otherwise(col(col\_name))) |

Các cột chuỗi sẽ không còn null, thay thế bằng giá trị mặc định.

Các cột Boolean sẽ thay thế null bằng False.

Các cột số liệu sẽ thay thế null bằng giá trị trung bình của cột đó.

|  |
| --- |
| Điền giá trị dựa trên cột Start\_Time  df = df.withColumn(      "Sunrise\_Sunset",      when(hour("Start\_Time").between(6, 18), "Day").otherwise("Night")  )  df = df.withColumn(      "Civil\_Twilight",      when(hour("Start\_Time").between(5, 19), "Day").otherwise("Night")  )  df = df.withColumn(      "Nautical\_Twilight",      when(hour("Start\_Time").between(4, 20), "Day").otherwise("Night")  )  df = df.withColumn(      "Astronomical\_Twilight",      when(hour("Start\_Time").between(3, 21), "Day").otherwise("Night")  )  null\_counts = df.select([sum(when(col(c).isNull(), 1).otherwise(0)).alias(c) for c in df.columns]) |

Các cột mới Sunrise\_Sunset, Civil\_Twilight, Nautical\_Twilight, và Astronomical\_Twilight sẽ có giá trị "Day" hoặc "Night" dựa trên giờ trong Start\_Time.

Đếm các giá trị null trong DataFrame và hiển thị kết quả với mỗi cột.

|  |
| --- |
| df = df.withColumn("Start\_Time", date\_format("Start\_Time", "yyyy-MM-dd HH:mm:ss"))  df = df.withColumn("End\_Time", date\_format("End\_Time", "yyyy-MM-dd HH:mm:ss"))  df = df.withColumn("Weather\_Timestamp", date\_format("Weather\_Timestamp", "yyyy-MM-dd HH:mm:ss")) |

Các cột Start\_Time, End\_Time, và Weather\_Timestamp sẽ được chuyển sang định dạng timestamp với năm, tháng, ngày, giờ, phút và giây. Điều này giúp dễ dàng thực hiện các phép toán và thao tác với thời gian trong DataFrame.

|  |
| --- |
| output\_path = "dataset/CleanData"  # Xuất DataFrame ra file CSV  df.coalesce(1).write.csv(output\_path, header=True, mode="overwrite")  spark.stop() |

DataFrame df sẽ được lưu vào thư mục dataset/CleanData dưới dạng tệp CSV duy nhất.

Sau khi xuất dữ liệu, phiên làm việc Spark sẽ được dừng lại.

### **3.3.2 Phân tích thành các file cvs để trực quan hóa đữ liệu**

#### **3.3.2.1 Phân tích mức độ tai nạn theo thời tiết**

|  |
| --- |
| from pyspark.sql import SparkSession  from pyspark.sql.functions import \*  # Khởi tạo Spark session  spark = SparkSession.builder.appName('Weather\_Analysis').getOrCreate()  # Đọc dữ liệu vào DataFrame  df = spark.read.csv("dataset/CleanData/cleandata.csv", header=True, inferSchema=True)  # Chuyển đổi Start\_Time thành kiểu dữ liệu timestamp  df = df.withColumn("Start\_Time", to\_timestamp("Start\_Time", "M/d/yyyy H:mm")) |

DataFrame df sẽ có cột Start\_Time dưới dạng timestamp, giúp dễ dàng thực hiện các phép toán và xử lý dữ liệu liên quan đến thời gian.

|  |
| --- |
| df = df.withColumn(      "Severity",      when(col("Severity") == 1, "Low")      .when(col("Severity") == 2, "Moderate")      .when(col("Severity") == 3, "High")      .when(col("Severity") == 4, "Severe")  )  df.printSchema()  df.count() |

Dùng hàm when() để thay thế các giá trị số trong cột Severity bằng các mức độ mô tả:

* 1 được thay bằng "Low",
* 2 được thay bằng "Moderate",
* 3 được thay bằng "High",
* 4 được thay bằng "Severe".

Các giá trị khác trong cột Severity không thay đổi và giữ nguyên.

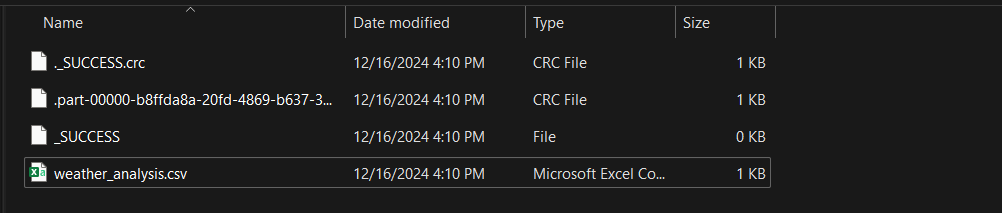
|  |
| --- |
| weather\_analysis = df.groupBy("Severity").agg(      avg("Temperature(F)").alias("Avg\_Temperature"),      avg("Visibility(mi)").alias("Avg\_Visibility"),      avg("Wind\_Speed(mph)").alias("Avg\_WindSpeed")  )  weather\_analysis.show() |

DataFrame weather\_analysis sẽ chứa các mức độ nghiêm trọng (Severity) cùng với các giá trị trung bình của:

* Temperature(F) (Nhiệt độ),
* Visibility(mi) (Khoảng cách quan sát),
* Wind\_Speed(mph) (Tốc độ gió).

Kết quả sẽ được hiển thị dưới dạng bảng với các mức độ nghiêm trọng và giá trị trung bình của các chỉ số thời tiết.

Cuối cùng xuất kết quả ra weather\_analysis và kết thúc phiên làm việc spark



#### **3.3.2.2 Phân tích mức độ tai nạn theo cơ sở hạ tầng**

Tương tự Phân tích mức độ tai nạn theo thời tiết:

* DataFrame df sẽ có cột Start\_Time dưới dạng timestamp, giúp dễ dàng thực hiện các phép toán và xử lý dữ liệu liên quan đến thời gian.
* Dùng hàm when() để thay thế các giá trị số trong cột Severity bằng các mức độ mô tả

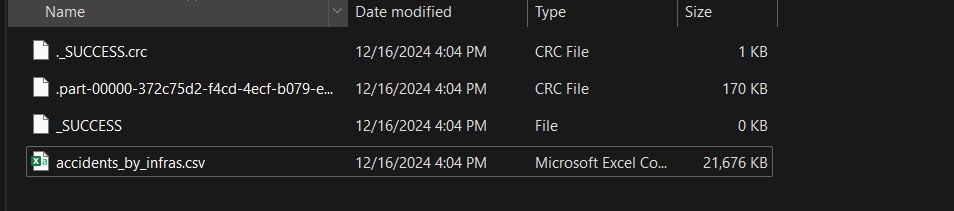
|  |
| --- |
| boolean\_columns = ["Amenity", "Bump", "Crossing", "Give\_Way", "Junction", "No\_Exit", "Railway",                     "Roundabout", "Station", "Stop", "Traffic\_Calming", "Traffic\_Signal", "Turning\_Loop"]  for col\_name in boolean\_columns:      df = df.withColumn(col\_name, when(col(col\_name) == True, 1).otherwise(0))  # Xem dữ liệu đã xử lý  accidents\_by\_infras = df.select("ID", "Severity", \*boolean\_columns)  accidents\_by\_infras.show() |

DataFrame accidents\_by\_infras sẽ chứa các cột:

* ID: Mã định danh tai nạn.
* Severity: Mức độ nghiêm trọng của tai nạn.
* Các cột boolean (như Amenity, Bump, Crossing, v.v.) sẽ được chuyển thành các giá trị số 0 hoặc 1.

Kết quả sẽ được hiển thị dưới dạng bảng, giúp bạn phân tích tai nạn dựa trên cơ sở hạ tầng.

Cuối cùng xuất kết quả ra accidents\_by\_infras và kết thúc phiên làm việc spark



#### **3.3.2.3 Phân tích mức độ tai nạn theo địa lý**

Tương tự Phân tích mức độ tai nạn theo thời tiết:

* DataFrame df sẽ có cột Start\_Time dưới dạng timestamp, giúp dễ dàng thực hiện các phép toán và xử lý dữ liệu liên quan đến thời gian.
* Dùng hàm when() để thay thế các giá trị số trong cột Severity bằng các mức độ mô tả

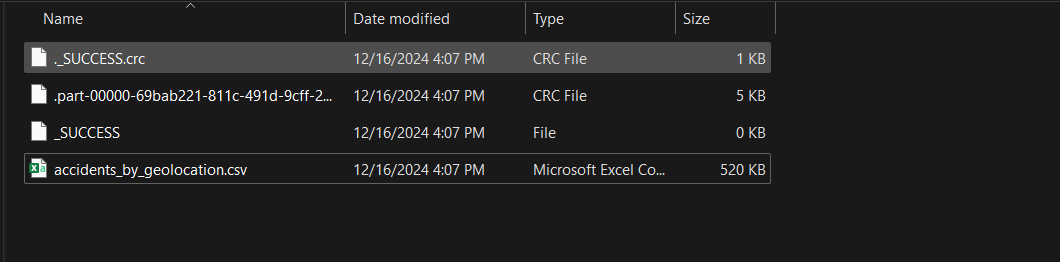
|  |
| --- |
| accidents\_by\_geolocation = df.groupBy("State", "City", "Severity").count()  accidents\_by\_geolocation.show() |

DataFrame accidents\_by\_geolocation sẽ chứa các cột:

* State: Tên bang.
* City: Tên thành phố.
* Severity: Mức độ nghiêm trọng của tai nạn.
* count: Số lượng tai nạn trong mỗi nhóm (theo bang, thành phố và mức độ nghiêm trọng).

Kết quả sẽ cho phép bạn phân tích số lượng tai nạn theo vị trí và mức độ nghiêm trọng trong các thành phố và bang.

Cuối cùng xuất kết quả ra accidents\_by\_geolocation và kết thúc phiên làm việc spark



#### **3.3.2.4 Phân tích mức độ tai nạn theo thời gian**

Tương tự Phân tích mức độ tai nạn theo thời tiết:

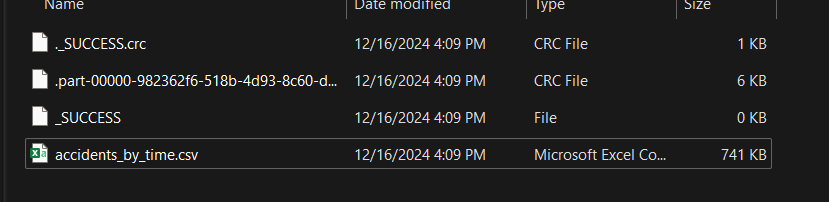
* DataFrame df sẽ có cột Start\_Time dưới dạng timestamp, giúp dễ dàng thực hiện các phép toán và xử lý dữ liệu liên quan đến thời gian.
* Dùng hàm when() để thay thế các giá trị số trong cột Severity bằng các mức độ mô tả

|  |
| --- |
| df = df.withColumn("Hour", hour("Start\_Time"))  df = df.withColumn("Day", dayofweek("Start\_Time"))  df = df.withColumn("Month", month("Start\_Time"))  df = df.withColumn("Year", year("Start\_Time"))  accidents\_by\_time = df.groupBy("Year", "Month", "Day", "Hour", "Severity").count()  accidents\_by\_time.show() |

DataFrame accidents\_by\_time sẽ chứa các cột:

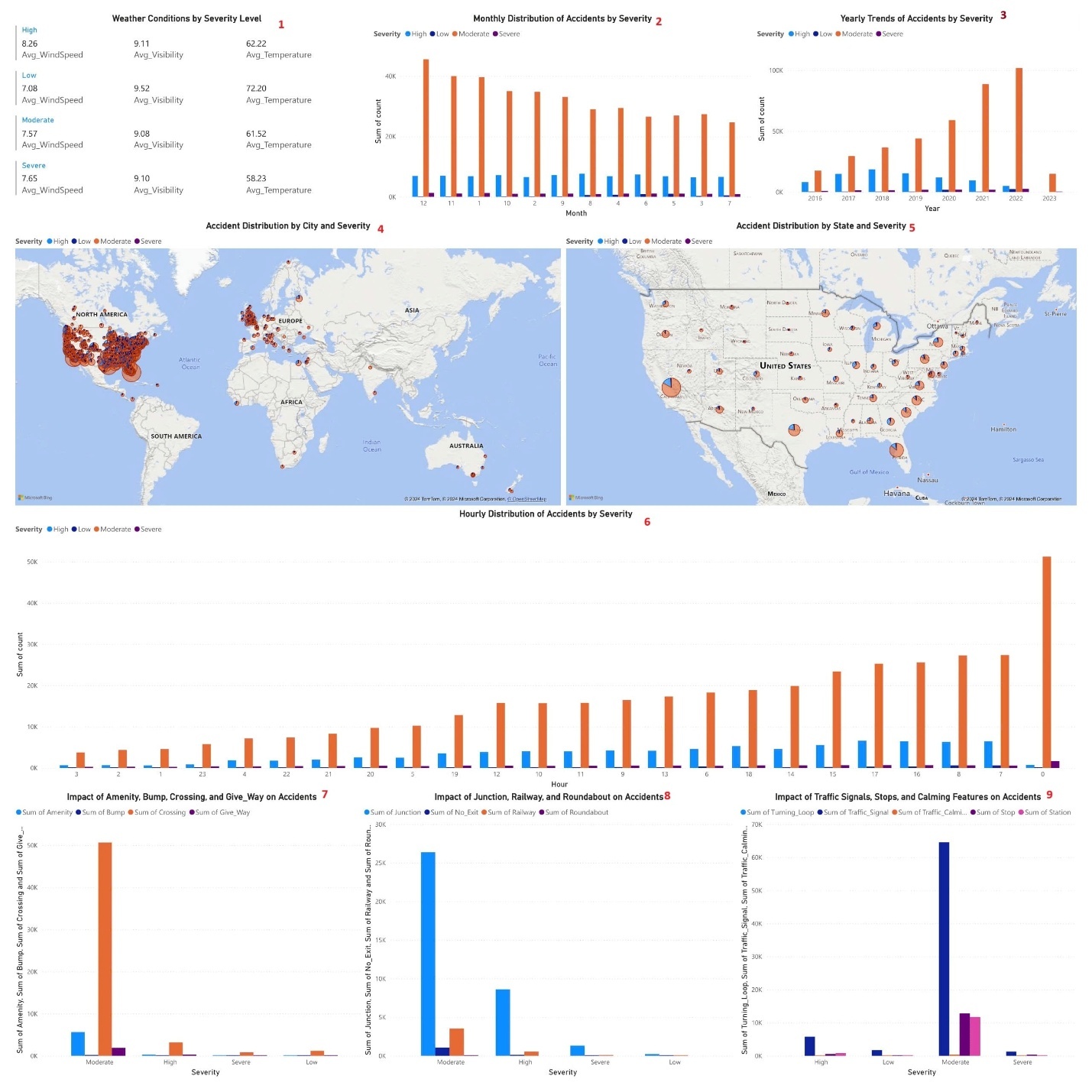
* Year: Năm.
* Month: Tháng.
* Day: Ngày trong tuần (1-7).
* Hour: Giờ trong ngày (0-23).
* Severity: Mức độ nghiêm trọng của tai nạn.
* count: Số lượng tai nạn trong mỗi nhóm theo thời gian và mức độ nghiêm trọng.

Kết quả sẽ giúp bạn phân tích số lượng tai nạn theo thời gian trong ngày, tuần, tháng và năm, cũng như mức độ nghiêm trọng của các tai nạn trong các khoảng thời gian đó.

Cuối cùng xuất kết quả ra accidents\_by\_time và kết thúc phiên làm việc spark 

### **3.3.3 Trực quan hóa dữ liệu bằng Power BI**

#### **3.3.3.1 Trực quan tổng quát**



|  |  |  |
| --- | --- | --- |
| **STT** | **Thông tin** | **Mô tả** |
| 1 | Weather Conditions by Severity Level | Phân loại điều kiện thời tiết theo mức độ nghiêm trọng và phân tích mối quan hệ với tai nạn. |
| 2 | Monthly Distribution of Accidents by Severity | Số lượng tai nạn xảy ra mỗi tháng, phân loại theo mức độ nghiêm trọng. |
| 3 | Yearly Trends of Accidents by Severity | Xu hướng số lượng tai nạn theo năm, phân tích mức độ nghiêm trọng. |
| 4 | Accident Distribution by City and Severity | Phân bố tai nạn theo từng thành phố, kết hợp với mức độ nghiêm trọng. |
| 5 | Accident Distribution by State and Severity | Phân bố tai nạn theo từng bang/khu vực, kết hợp với mức độ nghiêm trọng. |
| 6 | Hourly Distribution of Accidents by Severity | Số lượng tai nạn xảy ra theo từng khung giờ trong ngày, phân loại theo mức độ nghiêm trọng. |
| 7 | Impact of Amenity, Bump, Crossing, and Give Way | Phân tích tác động của các yếu tố tiện ích, gờ giảm tốc, lối băng qua đường, và biển nhường đường đến tai nạn. |
| 8 | Impact of Junction, Railway, and Roundabout | Ảnh hưởng của giao lộ, đường sắt, và vòng xuyến đến số lượng và mức độ nghiêm trọng của tai nạn. |
| 9 | Impact of Traffic Signals, Stops, and Calming Features | Nghiên cứu tác động của đèn giao thông, biển dừng, và các biện pháp làm chậm tốc độ đối với tai nạn. |

#### **3.3.3.2 Trực quan hóa theo ảnh hưởng thời tiết**

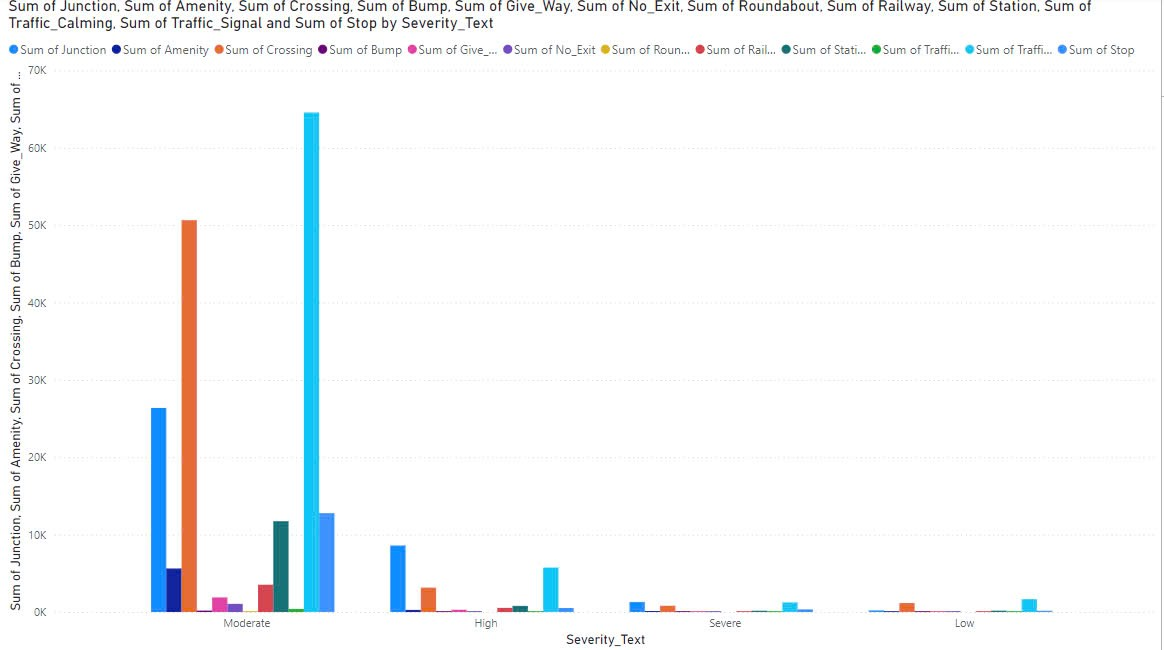
****

* Nhiệt độ trung bình (Avg\_Temp):
  + Mức độ tai nạn Low (ít nghiêm trọng) có nhiệt độ trung bình cao nhất là 72.2°F, cho thấy tai nạn ít nghiêm trọng thường xảy ra trong điều kiện thời tiết ấm áp hơn.
  + Ngược lại, mức độ Severe (nghiêm trọng nhất) có nhiệt độ trung bình thấp nhất là 58.2°F, chứng tỏ thời tiết lạnh hơn có thể liên quan đến mức độ nghiêm trọng cao hơn.
  + Moderate và High nằm trong khoảng trung bình.
* Tầm nhìn trung bình (Avg\_Visibil):
  + Tầm nhìn trung bình giữa các mức độ khá đồng đều (~9.1 đến 9.5).
  + Tuy nhiên, mức độ Low có tầm nhìn tốt nhất là 9.52, còn Moderate và Severe có giá trị thấp hơn một chút.
* Tốc độ gió trung bình (Avg\_WindSpeed):
  + Tốc độ gió trung bình cao nhất là 8.26 ở mức độ High, cho thấy gió mạnh hơn có thể liên quan đến tai nạn nghiêm trọng.
  + Low có tốc độ gió trung bình thấp nhất là 7.08, cho thấy điều kiện ít gió hơn thường đi kèm với mức độ nhẹ hơn.
  + Mức độ Severe và Moderate nằm ở khoảng giữa với tốc độ gió ~7.6.

Kết luận:

* Tai nạn nghiêm trọng (Severe) có thể liên quan đến nhiệt độ thấp hơn và tốc độ gió trung bình cao hơn.
* Tai nạn ít nghiêm trọng (Low) xảy ra khi nhiệt độ cao và tầm nhìn tốt.
* Các yếu tố thời tiết như nhiệt độ và tốc độ gió có thể đóng vai trò quan trọng trong việc xác định mức độ nghiêm trọng của tai nạn.

#### **3.3.3.3 Trực quan hóa theo ảnh hưởng cơ sở hạ tầng**

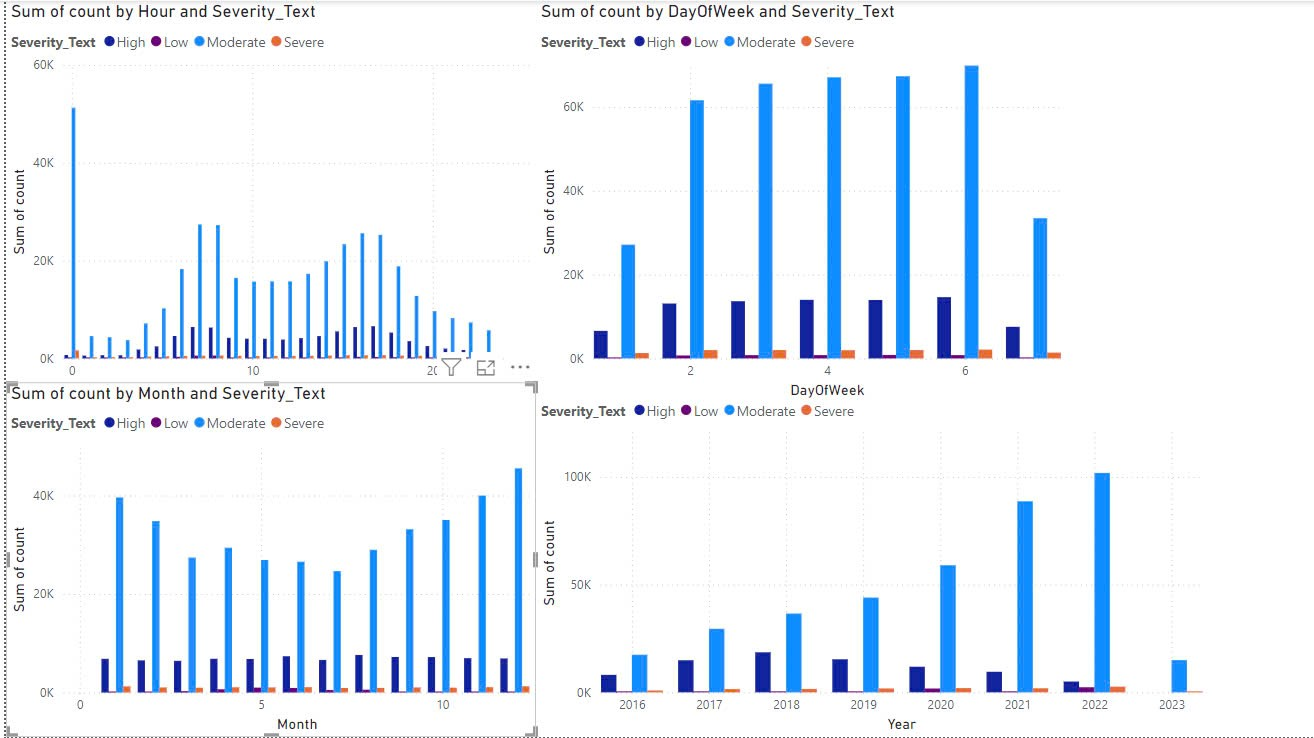
****

Biểu đồ thể hiện sự ảnh hưởng của các cơ sở hạ tầng đối với mức độ nguy hiểm của tai nạn giao thông

Khi có các cơ sở hạ tầng thì độ nghiêm trọng của tai nạn giao thông chỉ xảy ra ở mức bình thường. Có nghĩa là ở những nơi có đầy đủ các tiện ích của cơ sở hạ tầng thì độ nghiêm trọng của tai nạn giao thông chỉ ở mức bình thường, không gây ảnh hưởng nhiều đến giao thông đường bộ.

Kết luận: Nên xây dựng các cơ sở hạ tầng cần thiết cho việc đi lại để giảm mật độ tai nạn giao thông.

#### **3.3.3.4 Trực quan hóa theo ảnh hưởng thời gian**



Biểu đồ thể hệ từng mức độ nghiêm trọng của tai nạn giao thông ở từng khung giờ. Các Khung giờ vào lúc 0 giờ, 7 giờ, 8 giờ, 16 giờ, 18 giờ thì mức độ nghiêm trọng của tai nạn giao thông nằm ở mức bình thường nhưng mật độ xảy ra tai nạn giao thông lại quá cao. Bởi vì đó có thể là khung giờ cao điểm nên mật độ tai nạn xảy ra ở các khung giờ đó tương đối cao.

Biểu đồ thể hệ từng mức độ nghiêm trọng của tai nạn giao thông ở từng ngày trong tuần. Các ngày thứ ba đến thứ thứ bảy thì mật độ xảy ra tai nạn là rất cao. Bởi vì các ngày này người dân phải đi làm nên lưu lương phương tiện đi lại rất nhiều, nên việc xảy ra tai nạn là rất cao.

Biểu đồ thể hệ từng mức độ nghiêm trọng của tai nạn giao thông ở từng tháng. Các tháng xảy ra tai nạn nhiều là tháng 1, tháng 2, tháng 11 và tháng 12. Bởi vì các tháng này người dân tấp nập cho việc nghĩ cuối năm và chuẩn bị công việc cho đầu năm, đặc biệt có 2 ngày lễ lớn quan trong là ngày giáng sinh(ngày 25 tháng 12), ngày đầu năm (ngày 1 tháng 1),…, nên việc đi lại là rất nhiều. Các tháng ít xảy ra tai nạn là tháng 6, tháng 7. Bởi vì các tháng này là học sinh và sinh viên được nghỉ hè nên mật độ tai nạn thấp.

Kết luận: Việc đi lại tuy rất quan trong nhưng cẩn thận khi ra đường, hạn chế ra đường giờ cao điểm, sử dụng phương tiện công cộng.

#### **3.3.3.5 Trực quan hóa theo ảnh hưởng vị trí địa lí**

**Dashboard mức độ tai nạn ở mức thấp (Low)**A screenshot of a map

Description automatically generated

**Biểu đồ cột (Sum of Count):**

* Tổng số vụ tai nạn mức độ "Low" rất lớn, đạt khoảng 0.5M (500,000 vụ).
* Điều này cho thấy mức độ "Low" chiếm tỷ trọng đáng kể trong tổng thể các mức độ tai nạn.

**Phân bổ theo State (Bản đồ nước Mỹ):**

* Tai nạn mức độ "Low" trải đều trên khắp các bang của Hoa Kỳ.
* **Các cụm điểm tập trung cao:**
  + Một số bang như **California, Texas, Florida** và khu vực **Đông Bắc** có mật độ điểm cao hơn.
  + Đây có thể là do dân số tập trung đông hoặc tần suất giao thông cao ở các khu vực này.
* **Khu vực miền Trung và Tây Bắc Mỹ** có ít điểm hơn, phản ánh số lượng tai nạn mức độ "Low" thấp hơn.

**Phân bổ theo City (Bản đồ thế giới):**

* Phần lớn các vụ tai nạn ở mức "Low" tập trung tại:
  + **Bắc Mỹ**: Đặc biệt dày đặc tại Hoa Kỳ.
  + **Châu Âu**: Có một số cụm điểm, chủ yếu ở các nước phát triển có hạ tầng giao thông tốt như Đức, Pháp, và Anh.
  + **Úc** và **một phần Đông Nam Á** cũng có sự xuất hiện nhưng ít hơn.
* Châu Phi và Nam Mỹ chỉ có vài điểm lẻ tẻ, cho thấy dữ liệu tai nạn ở mức "Low" ở đây ít hơn.

**Phân tích xu hướng theo mức độ "Low"**

* **Số lượng lớn tai nạn mức "Low"**: Điều này phản ánh hầu hết các sự cố giao thông có thể không nghiêm trọng, chủ yếu là các va chạm nhỏ hoặc tình huống an toàn.
* **Dữ liệu theo thành phố và bang**: Tập trung nhiều ở các thành phố lớn hoặc khu vực có mật độ phương tiện dày đặc.

**Dashboard mức độ tai nạn ở mức vừa (Moderate)**

A screenshot of a computer

Description automatically generated

**Biểu đồ cột (Sum of Count):**

* Tổng số vụ tai nạn ở mức "Moderate" đạt khoảng **0.5 triệu vụ**.
* Mức này tương tự hoặc thậm chí nhỉnh hơn so với mức "Low", cho thấy số lượng tai nạn ở mức trung bình vẫn rất đáng kể.

**Phân bổ theo State (Bản đồ nước Mỹ):**

* **Độ phủ rộng**:
  + Số lượng các vụ tai nạn mức "Moderate" xuất hiện trên hầu hết các bang của Hoa Kỳ.
* **Các cụm lớn**:
  + **Bờ Đông** (các bang như New York, Pennsylvania, và khu vực New England) có mật độ điểm cao.
  + **Bờ Tây** (California, Arizona) cũng có lượng lớn các vụ tai nạn.
  + **Texas và Florida** tiếp tục là các điểm nóng.

**Phân bổ theo City (Bản đồ Thế giới):**

* **Các khu vực chính**:
  + **Bắc Mỹ** vẫn là khu vực chiếm ưu thế với số lượng lớn các vụ tai nạn mức "Moderate".
  + **Châu Âu** có mật độ cao, đặc biệt ở **Anh, Đức, và Pháp**.
  + **Úc** cũng nổi bật với nhiều điểm tập trung.
* **Khu vực ít tai nạn hơn**:
  + Châu Phi và Nam Mỹ chỉ xuất hiện lẻ tẻ vài điểm dữ liệu.

**Phân tích xu hướng mức "Moderate"**

* **Tăng từ mức Low**: Với số lượng 0.5 triệu, mức "Moderate" phản ánh tình trạng giao thông có phần nghiêm trọng hơn mức "Low".
* **Độ phủ toàn cầu**: Mức "Moderate" không chỉ tập trung ở Bắc Mỹ mà còn lan rộng sang châu Âu và Úc.

**Dashboard mức độ tai nạn ở mức cao (High)**

A screenshot of a computer screen

Description automatically generated

**Biểu đồ cột (Sum of Count):**

* Mức độ nghiêm trọng "High" chiếm **phần nhỏ hơn** so với các mức khác, dưới **0.1M vụ**.
* Điều này chứng tỏ các tai nạn ở mức nghiêm trọng "High" ít hơn nhiều so với các mức "Moderate" hoặc "Low".

**Phân bổ theo State (Bản đồ nước Mỹ):**

* **Mức độ nghiêm trọng "High" trải đều trên khắp nước Mỹ.**
  + Một số bang như **California, Texas, Florida** tiếp tục là các điểm nóng với nhiều vụ nghiêm trọng.
  + **Bờ Đông** cũng có mật độ điểm cao, tập trung ở các bang như **New York, Pennsylvania**.
  + Các khu vực **miền Trung và miền Tây nước Mỹ** cũng ghi nhận rải rác nhiều vụ nghiêm trọng.

**Phân bổ theo City (Bản đồ Thế giới):**

* **Các điểm nổi bật**:
  + **Bắc Mỹ** chiếm phần lớn số lượng các vụ nghiêm trọng "High".
  + **Châu Âu** cũng xuất hiện nhiều điểm tập trung, đặc biệt tại các quốc gia **Anh, Đức, Pháp**.
  + **Úc** tiếp tục nổi bật như một khu vực có nhiều vụ tai nạn nghiêm trọng.
* **Ít điểm tại các khu vực khác**:
  + Châu Phi và Nam Mỹ chỉ xuất hiện một số ít vụ tai nạn ở mức nghiêm trọng.

**Xu hướng mức "High"**

* **Số lượng thấp hơn nhưng tác động lớn**: Mức "High" tuy chiếm tỷ lệ nhỏ nhưng hậu quả có thể nghiêm trọng hơn nhiều so với các mức độ còn lại.
* **Tập trung ở các khu vực đô thị lớn**:
  + Các thành phố lớn như **Los Angeles, New York, London, Sydney** thường có tỷ lệ tai nạn mức "High" cao do mật độ giao thông lớn.

**Dashboard mức độ tai nạn ở mức nghiêm trọng (Severity)**

A screenshot of a computer screen

Description automatically generated

**Tổng quan dữ liệu:**

* Biểu đồ cột **"Sum of count"** ở phía trên bên trái cho thấy tổng số tai nạn ở mức "Low" rất cao, gần nửa triệu vụ.
* Điều này có thể chỉ ra rằng hầu hết các sự cố ghi nhận được có mức độ nghiêm trọng thấp.

Bản đồ ở phía trên bên phải thể hiện **phân bổ tai nạn trên nước Mỹ**:

* Tai nạn xảy ra trải dài trên hầu hết các tiểu bang.
* Có vẻ như các khu vực tập trung nhiều vụ tai nạn là các bang đông dân hoặc có hạ tầng giao thông lớn như **Texas, California và các bang ở bờ Đông Bắc (New York, Massachusetts, Pennsylvania)**.
* Các bang miền trung và tây bắc có mật độ tai nạn thấp hơn.

# **CHƯƠNG 4: BÀI TẬP CƠ BẢN VỀ APACHE SPARK**

**Khởi động hdfs**

**Máy master:**

A screenshot of a computer

Description automatically generated

**Máy slave:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Kiểm tra HDFS:**

A screenshot of a computer

Description automatically generated

* + Tất cả các máy điều bình thường

|  |
| --- |
| **Bài 30**  Tạo thư mục  A black screen with white text  Description automatically generated  A black screen with white text  Description automatically generated  Copy thư mục  A screen shot of a computer  Description automatically generated  A computer screen with text and numbers  Description automatically generated  A computer screen with green and blue text  Description automatically generated  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated  Giải thích:  **SparkSession.builder.appName('Exercise-30')**: Khởi tạo một phiên làm việc Spark với tên ứng dụng là 'Exercise-30'.  **getOrCreate()**: Kiểm tra xem một phiên làm việc Spark có đang chạy không, nếu không thì tạo mới một phiên làm việc mới.  **input\_path**: Tạo đường dẫn đến tệp input30.txt trong thư mục trên HDFS.  **output\_path**: Tạo đường dẫn cho thư mục đầu ra (output) trong thư mục trên HDFS.  **spark.read.text(input\_path)**: Đọc tệp văn bản input30.txt và tạo một DataFrame có mỗi dòng văn bản là một mục trong cột.  **rdd**: Chuyển DataFrame sang RDD (Resilient Distributed Dataset) để có thể thao tác với dữ liệu theo kiểu RDD (RDD cho phép sử dụng các phép biến đổi hàm cao cấp như map, filter, v.v.).  **map(lambda x: x[0])**: Sử dụng hàm map để lấy dữ liệu từ cột đầu tiên của DataFrame (mỗi dòng văn bản) và tạo một RDD chỉ chứa các dòng văn bản.  **filter(lambda x: 'google' in x)**: Lọc các dòng trong RDD chỉ giữ lại các dòng có chứa từ 'google'. Hàm filter tạo một RDD mới chỉ chứa các phần tử thỏa mãn điều kiện.  **coalesce(1)**: Sử dụng coalesce(1) để giảm số lượng phân vùng (partition) trong RDD xuống 1. Điều này có nghĩa là Spark sẽ viết toàn bộ kết quả vào một tệp duy nhất (thay vì nhiều tệp, mặc định Spark tạo ra một tệp cho mỗi phân vùng).  **saveAsTextFile(output\_path)**: Lưu RDD đã lọc vào thư mục chỉ định (output\_path). Dữ liệu sẽ được lưu dưới dạng tệp văn bản trong thư mục này.  **spark.stop()**: Dừng phiên làm việc Spark khi công việc đã hoàn thành. Việc này giúp giải phóng tài nguyên và tránh việc phiên làm việc vẫn tiếp tục chiếm dụng tài nguyên hệ thống khi không còn cần thiết nữa.  **KẾT QUẢ**:  A screenshot of a computer program  Description automatically generated  A screenshot of a computer program  Description automatically generated  A black background with white text  Description automatically generated |
| **Bài 31**  A black background with white text  Description automatically generated  A screenshot of a computer  Description automatically generated  Giải thích:   * **SparkSession.builder.appName('Exercise-31')**: Khởi tạo một phiên làm việc Spark với tên ứng dụng là 'Exercise-31'. * **getOrCreate()**: Kiểm tra xem một phiên làm việc Spark có đang chạy không, nếu không thì tạo mới một phiên làm việc mới. * **input\_path**: Tạo đường dẫn đến tệp input31.txt trong thư mục trên HDFS. * **output\_path**: Tạo đường dẫn cho thư mục đầu ra (output) trong thư mục trên HDFS. * **spark.read.text(input\_path)**: Đọc tệp văn bản input31.txt và tạo một DataFrame có mỗi dòng văn bản là một mục trong cột. * **rdd**: Chuyển DataFrame sang RDD (Resilient Distributed Dataset) để có thể thao tác với dữ liệu theo kiểu RDD (RDD cho phép sử dụng các phép biến đổi hàm cao cấp như map, filter, v.v.). * **map(lambda x: x[0])**: Sử dụng hàm map để lấy dữ liệu từ cột đầu tiên của DataFrame (mỗi dòng văn bản) và tạo một RDD chỉ chứa các dòng văn bản. * **lines.filter(lambda x: 'google' in x):** Áp dụng bộ lọc (filter) cho RDD, chỉ giữ lại các dòng chứa từ "google". Nếu dòng có chứa từ "google", hàm lambda x: 'google' in x trả về True, và dòng đó sẽ được giữ lại. * **lines\_w\_google.map(lambda x: str.split(x, ' ')[0]):** Mỗi dòng trong RDD lines\_w\_google được xử lý để lấy địa chỉ IP (giả sử địa chỉ IP là phần tử đầu tiên trong dòng văn bản). str.split(x, ' ') chia dòng thành một danh sách các từ, và [0] lấy phần tử đầu tiên (giả định đó là địa chỉ IP). * **.distinct():** Loại bỏ các địa chỉ IP trùng lặp, chỉ giữ lại các địa chỉ IP duy nhất. * **unique\_ips.coalesce(1)**: Sử dụng phương thức coalesce(1) để giảm số lượng phần vùng RDD xuống còn một phần vùng (1 file đầu ra). Điều này giúp tất cả kết quả được ghi vào một tệp duy nhất thay vì nhiều tệp. * **.saveAsTextFile(output\_path)**: Lưu kết quả đầu ra vào thư mục output\_path đã chỉ định. * **spark.stop()**: Dừng phiên làm việc Spark khi công việc đã hoàn thành. Việc này giúp giải phóng tài nguyên và tránh việc phiên làm việc vẫn tiếp tục chiếm dụng tài nguyên hệ thống khi không còn cần thiết nữa.   **Kết quả:**  A screenshot of a computer screen  Description automatically generated  A screenshot of a computer screen  Description automatically generated  A black background with white text  Description automatically generated |
| **Bài 32**  A black background with white lines  Description automatically generated  A screenshot of a computer  Description automatically generated  Giải thích:   * **split\_data(line)**: Đây là một hàm tùy chỉnh (user-defined function) nhận một dòng dữ liệu (line) từ tệp CSV và trả về nhiệt độ (giả sử nhiệt độ là phần tử thứ ba trong dòng, tức là data[2]). * **str.split(line, ',')**: Tách dòng thành một danh sách các phần tử, sử dụng dấu phẩy làm dấu phân cách (do dữ liệu là CSV). * **SparkSession.builder.appName('Exercise-32').getOrCreate()**: Tạo một phiên làm việc Spark, với tên ứng dụng là 'Exercise-32'. Nếu đã có một SparkSession, nó sẽ được tái sử dụng; nếu không, một phiên mới sẽ được tạo. * **input\_path:** Xây dựng đường dẫn cho tệp đầu vào input32.csv từ thư mục hiện tại. * **sc = spark.sparkContext**: Lấy đối tượng SparkContext từ SparkSession. SparkContext là đối tượng giúp làm việc với dữ liệu phân tán trong Spark. * **sc.textFile(input\_path)**: Đọc tệp CSV dưới dạng RDD (Resilient Distributed Dataset), mỗi dòng trong tệp trở thành một phần tử trong RDD. * **lines.map(split\_data)**: Áp dụng hàm split\_data cho mỗi dòng trong RDD lines. Kết quả là một RDD mới, temperatures, chứa các giá trị nhiệt độ (được trích xuất từ mỗi dòng). * **temperatures.reduce(lambda x, y: x if x > y else y)**: Sử dụng phương thức reduce để tính toán giá trị lớn nhất trong RDD temperatures. Phương thức reduce áp dụng hàm lambda cho các phần tử của RDD theo từng cặp (x, y). Nếu x lớn hơn y, trả về x; ngược lại, trả về y. Kết quả cuối cùng là giá trị lớn nhất trong RDD. * **print(max\_temp)**: In kết quả nhiệt độ lớn nhất ra màn hình. * **spark.stop()**: Dừng phiên làm việc Spark và giải phóng tài nguyên. Đây là bước quan trọng để kết thúc phiên làm việc của Spark sau khi xử lý dữ liệu.   **Kết quả:**  A screenshot of a computer screen  Description automatically generated |
| **Bài 33**  A black background with white lines  Description automatically generated  A screenshot of a computer  Description automatically generated  Giải thích:   * **temperatures.top(3)**: Đây là phương thức được sử dụng để lấy **3 giá trị nhiệt độ cao nhất** từ RDD temperatures. Phương thức top(n) trả về danh sách các phần tử có giá trị cao nhất trong RDD (n phần tử), trong trường hợp này là ba giá trị lớn nhất. Phương thức này sử dụng sắp xếp giảm dần để tìm các giá trị cao nhất. * **print('\n'.join(top\_3))**: Kết quả trả về từ top(3) là một danh sách các giá trị. Hàm join sẽ nối các phần tử trong danh sách top\_3 thành một chuỗi, cách nhau bằng ký tự xuống dòng \n, sau đó in ra kết quả.   **Kết quả:**  A screenshot of a computer  Description automatically generated |
| **Bài 34**  A black background with white text  Description automatically generated  A screenshot of a computer  Description automatically generated  Giải thích:   * **lines.map(split\_data)**: Chuyển mỗi dòng trong RDD lines thành giá trị nhiệt độ bằng cách sử dụng hàm split\_data, giống như các ví dụ trước. Kết quả là một RDD chứa các nhiệt độ. * **top(1)**: Phương thức top(1) lấy giá trị cao nhất trong RDD, tức là nhiệt độ cao nhất. * **lines.filter(lambda x: split\_data(x) == max\_temp)**: Đây là phương thức để lọc các dòng dữ liệu trong RDD lines, chỉ giữ lại những dòng mà nhiệt độ (được trích xuất bằng split\_data(x)) bằng max\_temp (nhiệt độ cao nhất). * **collect()**: Phương thức collect() thu thập tất cả các phần tử của RDD về dạng danh sách trong bộ nhớ của driver. Dùng để lấy kết quả về và thực hiện in ra. * **print('\n'.join(max\_temp\_lines))**: In các dòng dữ liệu có nhiệt độ cao nhất ra màn hình   **Kết quả:**  A screenshot of a computer screen  Description automatically generated |
| **Bài 35**  **Tạo thư mục để lưu trữ dữ liệu đầu vào trong HDFS**    **Copy thư mục chứa các bài từ 35 đến 40, từ máy chính sang máy ảo**    **Mục tiêu**  **Input:** Một tập hợp các file CSV có cấu trúc chứa giá trị PM10 (hàm lượng bụi mịn trong không khí) theo từng ngày cho một tập hợp các cảm biến.  Mỗi dòng trong file có định dạng như sau:  sensorId,date,PM10\_value (μg/m3)  **Output**:  Kết quả là các ngày tương ứng với giá trị PM10 lớn nhất.  **Tệp CSV input35.csv:**   |  |  |  | | --- | --- | --- | | s1 | 1/1/2016 | 20.5 | | s2 | 1/1/2016 | 30.1 | | s1 | 1/2/2016 | 60.2 | | s2 | 1/2/2016 | 20.4 | | s1 | 1/3/2016 | 60.2 | | s2 | 1/3/2016 | 52.5 |   **Code python:**    **Thực thi**  Đưa file input tại /home/hadooptanhuy/CK/ex-35 lên hdfs tại home/ hadooptanhuy/CK/inputs    **Kết quả:**    **Giải thích:**   * **import sys:**   + Thư viện sys cung cấp các chức năng tương tác với hệ thống, như xử lý tham số dòng lệnh và truy cập các tham số cấu hình hệ thống. * **import os:**   + Thư viện os cung cấp các hàm tương tác với hệ điều hành, bao gồm các thao tác với tệp và thư mục, như tạo và kiểm tra sự tồn tại của tệp hoặc thư mục. * **from pathlib import Path:**   + Nhập Path từ thư viện pathlib, giúp dễ dàng làm việc với các đường dẫn tệp theo cách trực quan và an toàn hơn. * **from pyspark.sql import SparkSession:**   + Nhập SparkSession từ thư viện PySpark, dùng để khởi tạo ứng dụng Spark và tương tác với các dữ liệu phân tán. * **def split\_data(line)::**   + Định nghĩa một hàm có tên split\_data, nhận một dòng dữ liệu (line) và sẽ trả về nhiệt độ trong dòng đó. * **data = str.split(line, ','):**   + Tách dòng line thành một danh sách data bằng cách sử dụng dấu phẩy (,) làm dấu phân cách giữa các giá trị trong dòng. * **return data[2]:**   + Trả về phần tử thứ ba trong danh sách data, là giá trị nhiệt độ từ cột thứ ba trong tệp CSV. * **if \_\_name\_\_ == "\_\_main\_\_"::**   + Đảm bảo rằng đoạn mã dưới đây chỉ được thực thi khi chương trình được chạy trực tiếp, không phải khi được nhập khẩu như một module. * **spark = SparkSession.builder.appName('Exercise-35').getOrCreate():**   + Tạo một phiên làm việc Spark mới với tên 'Exercise-35'. Nếu phiên làm việc đã tồn tại, nó sẽ được tái sử dụng. * **absolute\_path = Path().absolute():**   + Lấy đường dẫn tuyệt đối của thư mục làm việc hiện tại bằng cách sử dụng thư viện pathlib. * **input\_path = os.path.join('hdfs:/user/hadooptanhuy/CK/inputs', 'input35.csv'):**   + Xác định đường dẫn đầy đủ đến tệp CSV trên HDFS bằng cách kết hợp thư mục và tên tệp, sử dụng hàm os.path.join. * **sc = spark.sparkContext:**   + Lấy đối tượng SparkContext từ phiên làm việc Spark (spark), đối tượng này cho phép thực hiện các phép toán với RDD (Resilient Distributed Datasets). * **lines = sc.textFile(input\_path):**   + Đọc tệp CSV từ HDFS thành một RDD chứa các dòng văn bản từ tệp. * **max\_temp = lines.map(split\_data).top(1)[0]:**   + Dùng map(split\_data) để áp dụng hàm split\_data cho mỗi dòng, lấy nhiệt độ từ mỗi dòng. top(1) trả về giá trị lớn nhất trong RDD, và [0] lấy phần tử đầu tiên (nhiệt độ cao nhất). * **max\_temp\_dates = lines.filter(lambda x: split\_data(x) == max\_temp).map(lambda x: str.split(x,',')[1]).collect():**   + Lọc các dòng có nhiệt độ bằng với giá trị max\_temp bằng cách sử dụng filter. Sau đó, lấy ngày tháng từ các dòng đó bằng map, và thu thập kết quả thành một danh sách bằng collect(). * **print('\n'.join(max\_temp\_dates)):**   + In ra các ngày có nhiệt độ cao nhất, mỗi ngày trên một dòng mới. Dùng join để nối các phần tử của danh sách max\_temp\_dates với dấu xuống dòng (\n). * **spark.stop():**   + Dừng phiên làm việc Spark để giải phóng tài nguyên đã được Spark sử dụng. |
| **Bài 36**  **Mục tiêu**  **Input:** Một tập hợp các tệp CSV (có cấu trúc) chứa giá trị hàng ngày của PM10 cho một tập hợp các cảm biến.  Mỗi dòng trong tệp có định dạng:  sensorId,date,PM10 value (μg/m3)  **Output:**  Tính giá trị trung bình của PM10 từ tất cả các đối tượng  **Tệp CSV input36.csv:**   |  |  |  | | --- | --- | --- | | s1 | 1/1/2016 | 20.5 | | s2 | 1/1/2016 | 30.1 | | s1 | 1/2/2016 | 60.2 | | s2 | 1/2/2016 | 20.4 | | s1 | 1/3/2016 | 55.5 | | s2 | 1/3/2016 | 52.5 |   **Code python**    **Thực thi**  Đưa file input tại /home/hadooptanhuy/CK/ex-36 lên hdfs tại home/ hadooptanhuy/CK/inputs    **Kết quả:**    **Giải thích:**   * **from pyspark.sql import SparkSession**: Nhập SparkSession từ thư viện PySpark, lớp này cho phép bạn khởi tạo một ứng dụng Spark và tương tác với các dữ liệu phân tán. * **def split\_data(line):**: Định nghĩa một hàm có tên split\_data, nhận vào một dòng dữ liệu (line) và trả về giá trị nhiệt độ từ dòng đó. * **data = str.split(line, ',')**: Hàm split\_data tách dòng dữ liệu line thành một danh sách data bằng cách sử dụng dấu phẩy (,) làm dấu phân cách giữa các cột. * **return float(data[2])**: Trả về giá trị nhiệt độ ở cột thứ ba (data[2]), và chuyển nó thành kiểu float để có thể tính toán. * **if \_\_name\_\_ == "\_\_main\_\_":**: Đảm bảo rằng đoạn mã dưới đây chỉ được thực thi khi chương trình được chạy trực tiếp, không phải khi được nhập khẩu như một module. * **spark = SparkSession.builder.appName('Exercise-36').getOrCreate()**: Tạo một phiên làm việc Spark mới với tên ứng dụng là 'Exercise-36'. Nếu phiên làm việc đã tồn tại, nó sẽ được tái sử dụng. * **absolute\_path = Path().absolute()**: Lấy đường dẫn tuyệt đối của thư mục làm việc hiện tại (dùng thư viện pathlib). * **input\_path = os.path.join('hdfs:/user/hadooptanhuy/CK/inputs', 'input36.csv')**: Xác định đường dẫn đầy đủ đến tệp CSV trên HDFS bằng cách kết hợp thư mục và tên tệp. * **sc = spark.sparkContext**: Lấy đối tượng SparkContext từ phiên làm việc Spark, đối tượng này dùng để thực hiện các phép toán phân tán với RDD (Resilient Distributed Datasets). * **lines = sc.textFile(input\_path)**: Đọc tệp CSV từ HDFS thành một RDD, mỗi phần tử trong RDD là một dòng văn bản từ tệp. * **mean\_temp = lines.map(split\_data).mean()**: Dùng map(split\_data) để áp dụng hàm split\_data cho mỗi dòng trong RDD, lấy giá trị nhiệt độ từ mỗi dòng. Sau đó, dùng phương thức mean() để tính giá trị trung bình của các nhiệt độ này. * **print('%2.2f' % mean\_temp)**: In ra giá trị trung bình nhiệt độ dưới dạng số thực với hai chữ số thập phân. * **spark.stop()**: Dừng phiên làm việc Spark để giải phóng tài nguyên. |
| **Bài 37**  **Mục tiêu**  **Input:** Một tập hợp các file CSV có cấu trúc chứa giá trị PM10 (hàm lượng bụi mịn trong không khí) theo từng ngày cho một tập hợp các cảm biến.  Mỗi dòng trong file có định dạng như sau:  sensorId,date,PM10\_value (μg/m3)  **Output**:   * Xác định giá trị lớn nhất của PM10 cho mỗi cảm biến (sensorId). * In kết quả ra màn hình, hiển thị giá trị tối đa của PM10 cho từng cảm biến.   **Tệp CSV input36.csv:**   |  |  |  | | --- | --- | --- | | s1 | 1/1/2016 | 20.5 | | s2 | 1/1/2016 | 30.1 | | s1 | 1/2/2016 | 60.2 | | s2 | 1/2/2016 | 20.4 | | s1 | 1/3/2016 | 55.5 | | s2 | 1/3/2016 | 52.5 |   **Code python**    **Thực thi**  Đưa file input tại /home/hadooptanhuy/CK/ex-37 lên hdfs tại home/ hadooptanhuy/CK/inputs    **Kết quả:**    **Giải thích:**   * **from pathlib import Path**: Nhập lớp Path từ thư viện pathlib, giúp dễ dàng làm việc với các đường dẫn tệp theo cách trực quan và an toàn hơn. * **from pyspark.sql import SparkSession**: Nhập SparkSession từ thư viện PySpark, cho phép bạn tạo và tương tác với một ứng dụng Spark. * **def split\_data(line):**: Định nghĩa một hàm split\_data, nhận vào một dòng dữ liệu (line) và tách nó thành các cột. * **data = str.split(line, ',')**: Hàm split\_data tách dòng line thành một danh sách data bằng cách sử dụng dấu phẩy (,) làm dấu phân cách giữa các cột. * **return [data[0], data[2]]**: Trả về một danh sách với hai phần tử: phần tử đầu tiên là sensorID (cột đầu tiên) và phần tử thứ ba là temp (nhiệt độ). * **if \_\_name\_\_ == "\_\_main\_\_":**: Đảm bảo rằng đoạn mã dưới đây chỉ được thực thi khi chương trình được chạy trực tiếp, không phải khi nhập khẩu như một module. * **spark = SparkSession.builder.appName('Exercise-37').getOrCreate()**: Tạo một phiên làm việc Spark mới với tên 'Exercise-37'. Nếu phiên làm việc đã tồn tại, nó sẽ được tái sử dụng. * **absolute\_path = Path().absolute()**: Lấy đường dẫn tuyệt đối của thư mục làm việc hiện tại (dùng thư viện pathlib). * **input\_path = os.path.join('hdfs:/user/hadooptanhuy/CK/inputs', 'input37.csv')**: Xác định đường dẫn đầy đủ đến tệp CSV trên HDFS, kết hợp thư mục và tên tệp bằng os.path.join. * **sc = spark.sparkContext**: Lấy đối tượng SparkContext từ phiên làm việc Spark. Đối tượng này cho phép thực hiện các phép toán phân tán với RDD (Resilient Distributed Datasets). * **lines = sc.textFile(input\_path)**: Đọc tệp CSV từ HDFS thành một RDD chứa các dòng văn bản từ tệp. * **sensor\_temp = lines.map(split\_data)**: Áp dụng hàm split\_data cho mỗi dòng trong RDD lines, tạo ra một RDD mới sensor\_temp, trong đó mỗi phần tử là một cặp (sensorID, temp). * **max\_temp\_by\_sensor = sensor\_temp.reduceByKey(lambda x, y: x if x > y else y).collect()**: Sử dụng reduceByKey để nhóm các cặp (sensorID, temp) theo sensorID và lấy giá trị nhiệt độ cao nhất cho mỗi cảm biến. Hàm lambda x, y: x if x > y else y so sánh hai nhiệt độ và giữ lại giá trị lớn nhất. Kết quả sẽ được thu thập bằng collect(). * **print(max\_temp\_by\_sensor)**: In ra kết quả là danh sách các cảm biến và nhiệt độ cao nhất của chúng. * **spark.stop()**: Dừng phiên làm việc Spark để giải phóng tài nguyên đã được Spark sử dụng. |
| **Bài 38**  **Mục tiêu**  **Input:** Một tập hợp các file CSV có cấu trúc chứa giá trị PM10 (hàm lượng bụi mịn trong không khí) theo từng ngày cho một tập hợp các cảm biến.  Mỗi dòng trong file có định dạng như sau:  sensorId,date,PM10\_value (μg/m3)  **Output**:  iệt kê các cảm biến (sensorId) có ít nhất **2 lần đọc** với giá trị PM10 lớn hơn ngưỡng **50 µg/m³**.  **Tệp CSV input38.csv:**   |  |  |  | | --- | --- | --- | | s1 | 1/1/2016 | 20.5 | | s2 | 1/1/2016 | 30.1 | | s1 | 1/2/2016 | 60.2 | | s2 | 1/2/2016 | 20.4 | | s1 | 1/3/2016 | 55.5 | | s2 | 1/3/2016 | 52.5 |   **Code python:**    **Thực thi**  Đưa file input tại /home/hadooptanhuy/CK/ex-38 lên hdfs tại home/ hadooptanhuy/CK/inputs    **Kết quả:**    **Giải thích:**   * **from pathlib import Path**: Nhập lớp Path từ thư viện pathlib, giúp dễ dàng làm việc với các đường dẫn tệp theo cách trực quan và an toàn hơn. * **from pyspark.sql import SparkSession**: Nhập SparkSession từ thư viện PySpark, cho phép bạn tạo và tương tác với một ứng dụng Spark. * **if \_\_name\_\_ == "\_\_main\_\_":**: Đảm bảo rằng đoạn mã dưới đây chỉ được thực thi khi chương trình được chạy trực tiếp, không phải khi nhập khẩu như một module. * **spark = SparkSession.builder.appName('Exercise-38').getOrCreate()**: Tạo một phiên làm việc Spark mới với tên 'Exercise-37'. Nếu phiên làm việc đã tồn tại, nó sẽ được tái sử dụng. * **absolute\_path = Path().absolute()**: Lấy đường dẫn tuyệt đối của thư mục làm việc hiện tại (dùng thư viện pathlib). * **input\_path = os.path.join('hdfs:/user/hadooptanhuy/CK/inputs', 'input38.csv')**: Xác định đường dẫn đầy đủ đến tệp CSV trên HDFS, kết hợp thư mục và tên tệp bằng os.path.join. * **sc = spark.sparkContext**: Lấy đối tượng SparkContext từ phiên làm việc Spark. Đối tượng này cho phép thực hiện các phép toán phân tán với RDD (Resilient Distributed Datasets). * **lines = sc.textFile(input\_path)**: Đọc tệp CSV từ HDFS thành một RDD chứa các dòng văn bản từ tệp. * **lines\_above = lines.filter(lambda x: float(str.split(x,',')[2]) > 50)** **filter**: Kiểm tra từng dòng dữ liệu. Sử dụng str.split(x, ',') để tách các cột. **float(str.split(x, ',')[2])**: Chuyển giá trị ở cột thứ 3 (nhiệt độ) thành kiểu float. Chỉ giữ lại các dòng có nhiệt độ lớn hơn 50. * **counter\_readings = lines\_above.map(split\_data\_add\_counter)** **Hàm split\_data\_add\_counter**: Nhận vào một dòng dữ liệu. Tách dòng thành các cột và lấy sensorId (cột đầu tiên). Trả về cặp (sensorId, 1) để đếm số lần xuất hiện của từng cảm biến. * **readings\_above\_threshold = counter\_readings.reduceByKey(add).filter(lambda x: x[1] > 1).collect():**   Gộp các giá trị (value) theo từng key (sensorId) bằng cách cộng các giá trị. Kết quả là tổng số lần xuất hiện của mỗi cảm biến có giá trị nhiệt độ trên 50.   * **print(readings\_above\_threshold)**: In ra kết quả. * **spark.stop()**: Dừng phiên làm việc Spark để giải phóng tài nguyên đã được Spark sử dụng. |
| **Bài 39**  **Mục tiêu**  **Input:** Một tập hợp các file CSV có cấu trúc chứa giá trị PM10 (hàm lượng bụi mịn trong không khí) theo từng ngày cho một tập hợp các cảm biến.  Mỗi dòng trong file có định dạng như sau:  sensorId,date,PM10\_value (μg/m3)  **Output**:  Mỗi dòng trong tệp kết quả chứa:   * sensorId * Danh sách các ngày (date) mà giá trị PM10 vượt ngưỡng 50 µg/m³ cho cảm biến đó.   **Tệp CSV input39.csv:**   |  |  |  | | --- | --- | --- | | s1 | 1/1/2016 | 20.5 | | s2 | 1/1/2016 | 30.1 | | s1 | 1/2/2016 | 60.2 | | s2 | 1/2/2016 | 20.4 | | s1 | 1/3/2016 | 55.5 | | s2 | 1/3/2016 | 52.5 |   **Code python**    **Thực thi**  Đưa file input tại /home/hadooptanhuy/CK/ex-39 lên hdfs tại home/ hadooptanhuy/CK/inputs    **Kết quả:**    **Giải thích:**   * **from pathlib import Path**: Nhập lớp Path từ thư viện pathlib, giúp dễ dàng làm việc với đường dẫn tệp theo cách trực quan và an toàn hơn. * **from pyspark.sql import SparkSession**: Nhập SparkSession từ thư viện PySpark, là đối tượng chính để khởi tạo và tương tác với ứng dụng Spark. * **from operator import add**: Nhập hàm add từ thư viện operator. Tuy nhiên, trong đoạn mã này, hàm add không được sử dụng, có thể do một phần của mã chưa được sử dụng hoặc lẽ ra sẽ được dùng cho một phép toán giảm dần (reduceByKey). * **def split\_data(line):**: Định nghĩa một hàm split\_data, nhận vào một dòng dữ liệu (line) và tách nó thành các cột. * **data = str.split(line, ',')**: Hàm split\_data tách dòng line thành một danh sách data bằng cách sử dụng dấu phẩy (,) làm dấu phân cách giữa các cột. * **return [data[0], data[1]]**: Trả về một cặp gồm sensorId (cột đầu tiên) và date (cột thứ hai) từ dòng dữ liệu. * **def filter\_by\_threshold(line):**: Định nghĩa một hàm filter\_by\_threshold dùng để lọc các dòng dữ liệu có nhiệt độ lớn hơn 50. * **temp = float(str.split(line, ',')[2])**: Chuyển đổi giá trị trong cột thứ ba (nhiệt độ) sang kiểu số thực (float). * **return temp > 50**: Trả về True nếu nhiệt độ lớn hơn 50, ngược lại trả về False. * **if \_\_name\_\_ == "\_\_main\_\_":**: Đảm bảo rằng đoạn mã dưới đây chỉ được thực thi khi chương trình được chạy trực tiếp, không phải khi nhập khẩu như một module. * **spark = SparkSession.builder.appName('Exercise-39').getOrCreate()**: Tạo một phiên làm việc Spark mới với tên ứng dụng là 'Exercise-39'. Nếu phiên làm việc đã tồn tại, nó sẽ được tái sử dụng. * **absolute\_path = Path().absolute()**: Lấy đường dẫn tuyệt đối của thư mục làm việc hiện tại bằng pathlib. * **input\_path = os.path.join('hdfs:/user/hadooptanhuy/CK/inputs', 'input39.csv')**: Xác định đường dẫn đầy đủ đến tệp CSV trên HDFS bằng cách kết hợp thư mục và tên tệp. * **sc = spark.sparkContext**: Lấy đối tượng SparkContext từ phiên làm việc Spark. Đối tượng này giúp thực hiện các phép toán phân tán trên RDD (Resilient Distributed Datasets). * **lines = sc.textFile(input\_path)**: Đọc tệp CSV từ HDFS thành một RDD, trong đó mỗi phần tử là một dòng văn bản từ tệp. * **lines\_above = lines.filter(filter\_by\_threshold).map(split\_data)**: Lọc các dòng có nhiệt độ lớn hơn 50 bằng cách sử dụng hàm filter\_by\_threshold, sau đó áp dụng hàm split\_data để lấy ra các cặp (sensorId, date). * **lines\_above\_by\_key = lines\_above.groupByKey().mapValues(list).collect()**: Dùng groupByKey() để nhóm các giá trị (date) theo sensorId, sau đó dùng mapValues(list) để chuyển các giá trị thành danh sách. Kết quả được thu thập bằng collect(). * **print(lines\_above\_by\_key)**: In ra kết quả là danh sách các cảm biến cùng với danh sách các ngày đo có nhiệt độ trên 50. * **spark.stop()**: Dừng phiên làm việc Spark để giải phóng tài nguyên đã được Spark sử dụng. |
| **Bài 40**  **Mục tiêu**  **Input:** Một tập hợp các file CSV có cấu trúc chứa giá trị PM10 (hàm lượng bụi mịn trong không khí) theo từng ngày cho một tập hợp các cảm biến.  Mỗi dòng trong file có định dạng như sau:  sensorId,date,PM10\_value (μg/m3)  **Output**: các cảm biến (sensorId) được sắp xếp theo số lượng các ngày "nguy cấp" mà giá trị PM10 vượt ngưỡng 50 µg/m³.  Mỗi dòng trong tệp kết quả chứa:   * Số lượng các ngày vượt ngưỡng 50 µg/m³ * sensorId tương ứng của cảm biến.   **Tệp CSV input40.csv:**   |  |  |  | | --- | --- | --- | | s1 | 1/1/2016 | 20.5 | | s2 | 1/1/2016 | 30.1 | | s1 | 1/2/2016 | 60.2 | | s2 | 1/2/2016 | 20.4 | | s1 | 1/3/2016 | 55.5 | | s2 | 1/3/2016 | 52.5 |   **Code python**      **Thực thi**  Đưa file input tại /home/hadooptanhuy/CK/ex-40 lên hdfs tại home/ hadooptanhuy/CK/inputs    **Kết quả:**    **Giải thích**   * **from operator import add**: Nhập hàm add từ thư viện operator, sẽ được sử dụng trong quá trình giảm dần để cộng dồn các giá trị. * **from pyspark.sql import SparkSession**: Nhập SparkSession từ thư viện PySpark, là đối tượng chính để khởi tạo và tương tác với Spark. * **def split\_data(line):**: Định nghĩa hàm split\_data, nhận vào một dòng dữ liệu (line) và tách nó thành các cột. * **data = str.split(line, ',')**: Tách dòng dữ liệu line thành một danh sách data bằng dấu phẩy (,) làm dấu phân cách. * **key = data[0]**: Gán giá trị của cột đầu tiên (sensorId) vào biến key. * **temperature = float(data[2])**: Chuyển giá trị trong cột thứ ba (nhiệt độ) thành kiểu số thực (float). * **return [key, 1 if temperature > 50 else 0]**: Trả về một cặp (key, value), trong đó key là sensorId và value là 1 nếu nhiệt độ lớn hơn 50, ngược lại là 0. * **def swap\_parts(line):**: Định nghĩa hàm swap\_parts, dùng để đảo ngược thứ tự của các phần tử trong cặp (key, value). * **return (line[1], line[0])**: Đảo ngược cặp (key, value), tức là chuyển line[1] (giá trị) lên trước và line[0] (khóa) lên sau. * **if \_\_name\_\_ == "\_\_main\_\_":**: Đảm bảo rằng đoạn mã dưới đây chỉ được thực thi khi chương trình được chạy trực tiếp, không phải khi nhập khẩu như một module. * **spark = SparkSession.builder.appName('Exercise-40').getOrCreate()**: Tạo một phiên làm việc Spark mới với tên ứng dụng là 'Exercise-40'. Nếu phiên làm việc đã tồn tại, nó sẽ được tái sử dụng. * **absolute\_path = Path().absolute()**: Lấy đường dẫn tuyệt đối của thư mục làm việc hiện tại bằng pathlib. * **input\_path = os.path.join('hdfs:/user/hadooptanhuy/CK/inputs', 'input40.csv')**: Xác định đường dẫn đầy đủ đến tệp CSV trên HDFS bằng cách kết hợp thư mục và tên tệp. * **sc = spark.sparkContext**: Lấy đối tượng SparkContext từ phiên làm việc Spark. Đối tượng này giúp thực hiện các phép toán phân tán trên RDD (Resilient Distributed Datasets). * **lines = sc.textFile(input\_path)**: Đọc tệp CSV từ HDFS thành một RDD, trong đó mỗi phần tử là một dòng văn bản từ tệp. * **lines\_key\_value = lines.map(split\_data)**: Áp dụng hàm split\_data để tách từng dòng thành các cặp (sensorId, 1 hoặc 0) tùy vào nhiệt độ. * **sensors\_counter = lines\_key\_value.reduceByKey(add).map(swap\_parts).sortByKey(ascending=False).collect()**: Dùng reduceByKey(add) để cộng dồn số lần xuất hiện (số lần nhiệt độ vượt quá 50) cho mỗi sensorId. Sau đó, áp dụng swap\_parts để đảo ngược cặp (sensorId, count), rồi sắp xếp các kết quả theo count giảm dần và thu thập kết quả về. * **print(sensors\_counter)**: In ra kết quả là danh sách các cảm biến và số lần mà nhiệt độ của chúng vượt quá 50, được sắp xếp giảm dần theo số lần. * **spark.stop()**: Dừng phiên làm việc Spark để giải phóng tài nguyên đã được Spark sử dụng. |
| **Bài 41**  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated   * Tạo thư mục          * Copy thư mục   A screenshot of a computer  Description automatically generated  A screenshot of a computer screen  Description automatically generated  A screenshot of a computer screen  Description automatically generated   * Ex-41   A screenshot of a computer program  Description automatically generated  A computer screen shot of a program  Description automatically generated  Giải thích:   |  | | --- | | def check\_argv\_value(vals):  if len(vals) <= 1:  print("ERR: Please provide a valid value for K", file=sys.stderr)  exit(-1)  try:  k = int(vals[1])  except ValueError:  print("ERR: {} is not a valid value for K".format(vals[1]), file=sys.stderr)  exit(-1)  return k |  * Mục đích: Đảm bảo tham số dòng lệnh đầu vào hợp lệ. * Kiểm tra nếu không có tham số hoặc tham số không phải số nguyên (int), chương trình sẽ in lỗi ra stderr và kết thúc. * Nếu hợp lệ, trả về giá trị k (số lượng cảm biến cần hiển thị).  |  | | --- | | def split\_data(line):  data = str.split(line, ',')  key = data[0] # sensorId  temperature = float(data[2]) # nhiệt độ  return [key, 1 if temperature > 50 else 0] |  * **Mục đích**: Chuyển đổi một dòng dữ liệu thành cặp (sensorId, 1 hoặc 0).   + Nếu nhiệt độ lớn hơn 50 (temperature > 50), trả về 1.   + Ngược lại, trả về 0.  |  | | --- | | def swap\_parts(line):  return (line[1], line[0]) |  * Mục đích: Đảo ngược cặp (sensorId, count) thành (count, sensorId). * Dùng để hỗ trợ sắp xếp giảm dần theo số lần vượt ngưỡng nhiệt độ.  |  | | --- | | sensors\_counter = lines\_key\_value.reduceByKey(add).map(swap\_parts).sortByKey(ascending=False).collect() |  * **Giải thích**:   + reduceByKey(add): Cộng tổng số lần vượt ngưỡng cho từng sensorId.   + map(swap\_parts): Đảo ngược thành (count, sensorId).   + sortByKey(ascending=False): Sắp xếp giảm dần theo count.   + collect(): Thu thập kết quả về driver node dưới dạng danh sách.  |  | | --- | | print(sensors\_counter[:k]) |  * **Mục đích**: Hiển thị k cảm biến có số lần vượt ngưỡng nhiệt độ cao nhất.   A screenshot of a computer screen  Description automatically generated |
| **Bài 42**  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated   |  | | --- | | questionsPairRDD = questionsInputRDD.map(lambda line: (line.split(",")[0], line.split(",")[2])) |  * **Mục đích**: Chuyển đổi mỗi dòng dữ liệu câu hỏi thành cặp (Q\_id, question\_text): * **Key**: Q\_id (mã câu hỏi). * **Value**: question\_text (nội dung câu hỏi).  |  | | --- | | answersPairRDD = answersInputRDD.map(lambda line: (line.split(",")[1], line.split(",")[3])) |   **Mục đích**: Chuyển đổi mỗi dòng dữ liệu câu trả lời thành cặp (Q\_id, answer\_text):   * **Key**: Q\_id (mã câu hỏi liên quan). * **Value**: answer\_text (nội dung câu trả lời).  |  | | --- | | for pair in joinedRDD.collect():  print(pair[0], list(pair[1])) |  * collect: Thu thập toàn bộ dữ liệu từ RDD vào driver node. * In kết quả: * pair[0]: Key bao gồm Q\_id và question\_text. * list(pair[1]): Danh sách các câu trả lời liên quan đến câu hỏi.   A screenshot of a computer  Description automatically generated |
| **Bài 43**  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated   * Giảt thích:  |  | | --- | | numberOfCriticalSituationsForStationRDD = readingsRDD.filter(lambda valuint(value.split(",")[5]) < freeSlotsThreshold) \  .map(lambda value: (value.split(",")[0], 1)) \  .groupByKey() \  .map(lambda pair: (pair[0], len(list(pair[1])))) |  * Mục tiêu: Đếm số lần mỗi trạm rơi vào tình huống nguy cấp. * Bước 1: Lọc các bản ghi mà num\_of\_free\_slots < 3. * Bước 2: Tạo cặp (stationId, 1) cho mỗi bản ghi. * Bước 3: Gom nhóm theo stationId (groupByKey()). * Bước 4: Đếm số bản ghi nguy cấp cho mỗi trạm.  |  | | --- | | numberOfReadingsForStationRDD = readingsRDD.map(lambda value: (value.split(",")[0], 1)) \  .groupByKey() \  .map(lambda pair: (pair[0], len(list(pair[1])))) |  * Mục tiêu: Đếm tổng số bản ghi được liên kết với mỗi stationId. * Tương tự như trên, nhưng không áp dụng bộ lọc.  |  | | --- | | outputJoinReadingsRDD = joinReadingsRDD.map(lambda pair: (pair[1], pair[0])) \  .filter(lambda pair: pair[0] > criticalSituationPercentageThreshold) \  .sortByKey() \  .map(lambda pair: (pair[1], pair[0]))  outputJoinReadingsRDD.saveAsTextFile("critical\_situations\_stations" |  * Ý nghĩa: Lọc các trạm có tỷ lệ nguy cấp cao hơn ngưỡng (criticalSituationPercentageThreshold = 80) và lưu kết quả.  |  | | --- | | outputJoinReadingsRDD = joinReadingsRDD.map(lambda pair: (pair[1], pair[0])) \  .filter(lambda pair: pair[0] > criticalSituationPercentageThreshold) \  .sortByKey() \  .map(lambda pair: (pair[1], pair[0]))  outputJoinReadingsRDD.saveAsTextFile("critical\_situations\_stations\_timeslots") |  * Ý nghĩa: Lọc các cặp (stationId, timeslot) có tỷ lệ nguy cấp cao hơn ngưỡng và lưu kết quả.   A screenshot of a computer screen  Description automatically generated |
| **Bài 44**  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated   |  | | --- | | def misleadingProfileFunc(userWatchedLikedGenresLists):  likedGenres = list(userWatchedLikedGenresLists[1][1]) # Thể loại yêu thích  numWatchedMovies = 0  notLiked = 0  for watchedGenre in userWatchedLikedGenresLists[1][0]: # Thể loại đã xem  numWatchedMovies += 1  if watchedGenre not in likedGenres: # Nếu thể loại đã xem không thuộc danh sách yêu thích  notLiked += 1  # Kiểm tra nếu tỷ lệ phim đã xem không thuộc thể loại yêu thích vượt quá ngưỡng `threshold`  if float(notLiked) > threshold \* float(numWatchedMovies):  return True  else:  return False |  * Hàm này: * Lấy danh sách thể loại yêu thích của người dùng. * Đếm số phim đã xem (numWatchedMovies) và số phim không thuộc thể loại yêu thích (notLiked). * So sánh tỷ lệ notLiked / numWatchedMovies với ngưỡng threshold.  |  | | --- | | misleadingUsersListsRDD = userWatchedLikedGenres.filter(misleadingProfileFunc) |  * Lọc ra các người dùng mà hồ sơ của họ gây hiểu lầm (theo định nghĩa của hàm misleadingProfileFunc).  |  | | --- | | misleadingUsersRDD = misleadingUsersListsRDD.keys() |  * Lấy danh sách userid của các hồ sơ gây hiểu lầm.  |  | | --- | | for user in misleadingUsersRDD.collect():  print(user) |  * In ra danh sách các ID người dùng với hồ sơ gây hiểu lầm.   A screenshot of a computer screen  Description automatically generated |
| **Bài 45**  A computer screen shot of a black screen  Description automatically generated  **A screenshot of a computer  Description automatically generated**  Giải thích:   |  | | --- | | moviesRDD = sc.parallelize(movies\_data).map(lambda value: (value.split(",")[0], value.split(",")[2]))  preferencesRDD = sc.parallelize(preferences\_data).map(lambda value: (value.split(",")[0], value.split(",")[1]))  watchedMoviesRDD = sc.parallelize(watched\_movies\_data).map(lambda value: (value.split(",")[1], value.split(",")[0])) |  * Chuyển đổi dữ liệu thô thành các RDD chứa cặp khóa-giá trị: * moviesRDD: (movieid, genre). * preferencesRDD: (userid, preferred\_genre). * watchedMoviesRDD: (movieid, userid).  |  | | --- | | userMovieGenreRDD = watchedMoviesRDD.join(moviesRDD) \  .map(lambda pair: (pair[1][0], pair[1][1])) \  .groupByKey() |  * watchedMoviesRDD.join(moviesRDD): Ghép nối để ánh xạ mỗi movieid đã xem tới userid và movie-genre. * Kết quả: (userid, [watched\_movie\_genres]).  |  | | --- | | def isUserMisleading(pair):  listPreferenceMovieGenres = list(pair[1][0]) # List of preferred genres  listWatchedMovieGenres = list(pair[1][1]) # List of genres of watched movies  numberOfMoviesSeen = len(listWatchedMovieGenres)  notPreferredGenresFilms = 0    for watchedMovieGenres in listWatchedMovieGenres:  for movieGenre in watchedMovieGenres:  if movieGenre not in listPreferenceMovieGenres:  notPreferredGenresFilms += 1    proportion = notPreferredGenresFilms / numberOfMoviesSeen    if proportion > threshold:  return True  else:  return False |  * **Đầu vào**: * pair: (userid, ([preferred\_genres], [watched\_genres])). * **Xử lý**: * Tạo danh sách thể loại yêu thích (listPreferenceMovieGenres) và đã xem (listWatchedMovieGenres). * Tính tổng số phim đã xem (numberOfMoviesSeen) và số phim không thuộc thể loại yêu thích (notPreferredGenresFilms). * Tính tỷ lệ phim không thuộc sở thích: proportion = notPreferredGenresFilms / numberOfMoviesSeen. * So sánh với threshold. Nếu vượt ngưỡng, trả về True.   A screenshot of a computer program  Description automatically generated  Ghi chú   * Ngưỡng (threshold): * Ngưỡng 0.5 nghĩa là nếu trên 50% phim đã xem không thuộc sở thích, người dùng được coi là gây hiểu lầm. * Có thể thay đổi giá trị threshold để điều chỉnh độ nhạy. * Hiệu suất: * Nếu dữ liệu lớn, cần sử dụng hành động Spark như take() thay vì collect() để tránh tải toàn bộ dữ liệu về driver. * Hàm isUserMisleading: * Đoạn mã này có thể tối ưu hóa hơn nữa bằng cách sử dụng các phép toán Spark thay vì chuyển đổi dữ liệu về danh sách Python (list()), để tránh mất lợi thế phân tán của Spark.   **Bài 46** |
|  |

A screenshot of a computer program

Description automatically generated

Giải thích code:

 **from pyspark import SparkConf, SparkContext:**

* **SparkConf**: Được sử dụng để cấu hình các tham số Spark, ví dụ như tên ứng dụng, các tham số kết nối với cluster Spark (nếu cần).
* **SparkContext**: Đây là đối tượng chính để giao tiếp với Spark. Nó được sử dụng để tạo ra các RDD, thực thi các hành động và chuyển tải các tác vụ vào hệ thống Spark.

 **import sys:**

* Dùng để tương tác với hệ thống, đặc biệt là để truy xuất tham số dòng lệnh (tuy nhiên trong mã trên, nó chưa được sử dụng). Nếu bạn muốn sử dụng sys.argv để nhận các tham số từ dòng lệnh khi chạy chương trình, bạn có thể import thư viện này.

**Định nghĩa lớp TimeStampTemperature**

class TimeStampTemperature:

def \_\_init\_\_(self, timestamp, temperature):

self.timestamp = timestamp

self.temperature = temperature

def \_\_str\_\_(self):

return f"{self.timestamp},{self.temperature}"

* Lớp TimeStampTemperature này được sử dụng để chứa các giá trị timestamp và temperature của một bản ghi.
* Hàm \_\_str\_\_ định nghĩa cách để chuyển đối tượng thành chuỗi (String) theo định dạng timestamp,temperature.

**2. Hàm process\_window**

def process\_window(reading):

fields = reading.split(",")

timestamp = int(fields[0])

temperature = float(fields[1])

result = [

(timestamp, TimeStampTemperature(timestamp, temperature)),

(timestamp - 60, TimeStampTemperature(timestamp, temperature)),

(timestamp - 120, TimeStampTemperature(timestamp, temperature)),

]

return result

* Hàm này nhận vào một bản ghi reading (dưới dạng chuỗi), tách nó thành timestamp và temperature.
* Sau đó, nó tạo ba "cửa sổ thời gian" (windows) cho bản ghi đó:
  + Một cửa sổ tại timestamp (tương ứng với thời điểm gốc).
  + Một cửa sổ tại timestamp - 60 (sau 60 giây).
  + Một cửa sổ tại timestamp - 120 (sau 120 giây).
* Kết quả là một danh sách gồm các tuple, trong đó key là thời gian bắt đầu cửa sổ và value là một đối tượng TimeStampTemperature.

**3. Hàm filter\_windows**

def filter\_windows(window):

timestamp\_temp = {}

min\_timestamp = float('inf')

for element in window:

timestamp\_temp[element.timestamp] = element.temperature

min\_timestamp = min(min\_timestamp, element.timestamp)

if len(timestamp\_temp) == 3:

increasing = True

for ts in range(min\_timestamp + 60, min\_timestamp + 121, 60):

if timestamp\_temp[ts] <= timestamp\_temp[ts - 60]:

increasing = False

break

return increasing

return False

* Hàm này nhận một cửa sổ thời gian window (là một iterable chứa các đối tượng TimeStampTemperature).
* **Điều kiện 1:** Kiểm tra nếu cửa sổ có đúng 3 phần tử (tức là ba timestamp: t, t-60, t-120).
* **Điều kiện 2:** Kiểm tra xem nhiệt độ có tăng dần theo thời gian không, nghĩa là temperature(t) > temperature(t-60), temperature(t-60) > temperature(t-120).
* Nếu cả hai điều kiện đều đúng, cửa sổ được chọn, nếu không thì bị loại.

**4. Phần chính của chương trình (if \_\_name\_\_ == "\_\_main\_\_":)**

**Cấu hình Spark và đọc dữ liệu**

conf = SparkConf().setAppName("Spark Exercise #46")

sc = SparkContext(conf=conf)

readingsRDD = sc.textFile(input\_path)

* Cấu hình Spark với tên ứng dụng là "Spark Exercise #46".
* Tạo một SparkContext để bắt đầu làm việc với Spark.
* Đọc dữ liệu từ file đầu vào (input\_path) và tạo một RDD từ dữ liệu đó (readingsRDD).

**Tạo các phần tử của các cửa sổ thời gian**

windows\_elements\_RDD = readingsRDD.flatMap(process\_window)

* Dùng phương thức flatMap để áp dụng hàm process\_window cho mỗi bản ghi trong readingsRDD. Kết quả là một RDD mới, windows\_elements\_RDD, chứa các phần tử TimeStampTemperature với các cửa sổ thời gian.

**Nhóm các phần tử theo timestamp**

timestamps\_windows\_RDD = windows\_elements\_RDD.groupByKey()

* Phương thức groupByKey() nhóm các phần tử trong windows\_elements\_RDD theo timestamp. Kết quả là một RDD chứa các cặp (timestamp, Iterable[TimeStampTemperature]).

**Lọc các cửa sổ hợp lệ**

windows\_RDD = timestamps\_windows\_RDD.values()

selected\_windows\_RDD = windows\_RDD.filter(filter\_windows)

* Chỉ lấy giá trị từ các cặp (timestamp, value) trong timestamps\_windows\_RDD (bỏ qua timestamp).
* Sau đó, áp dụng hàm filter\_windows để lọc ra các cửa sổ thỏa mãn điều kiện (có 3 phần tử và nhiệt độ tăng dần).

**Chuyển các cửa sổ hợp lệ thành định dạng chuỗi**

formatted\_windows\_RDD = selected\_windows\_RDD.map(lambda window: "\n".join(map(str, window)))

* Chuyển các đối tượng TimeStampTemperature trong window thành chuỗi bằng cách dùng hàm \_\_str\_\_ đã định nghĩa.
* Sau đó, dùng map để nối các chuỗi này lại với nhau, mỗi đối tượng cách nhau bởi một dòng mới (\n).

**Lưu kết quả**

formatted\_windows\_RDD.saveAsTextFile(output\_path)

* Lưu kết quả vào thư mục đầu ra (output\_path), mỗi cửa sổ sẽ được ghi vào một file riêng biệt dưới dạng .txt.

**Dừng SparkContext**

sc.stop()

* Dừng SparkContext sau khi quá trình xử lý kết thúc để giải phóng tài nguyên.

Chạy chương trình:

Spark\_submit Final\_Bigdata/ex-46/code.py

A screenshot of a computer screen

Description automatically generated

Xem kết quả:

A screenshot of a computer

Description automatically generated

* + Đã xuất hiện file output
  + Kiểm tra kết quả

A screenshot of a computer

Description automatically generated

|  |
| --- |
| **Bài 47**    Giải thích code:  **Import thư viện**   * SparkSession: Để khởi tạo một phiên Spark. * Path: Để lấy đường dẫn tuyệt đối của thư mục hiện tại.   **Hàm perform\_analysis\_dataframe**  **Chức năng: Thực hiện phân tích bằng DataFrame API.**   1. Lọc dữ liệu:    * Chỉ lấy các dòng có gender là 'male'. 2. Chọn cột:    * Giữ lại hai cột: name và age. 3. Cập nhật tuổi:    * Tăng giá trị age lên 1. 4. Sắp xếp dữ liệu:    * Sắp xếp theo age giảm dần và name tăng dần. 5. Ghi kết quả ra file CSV:    * Tái phân phối dữ liệu về 1 file duy nhất trước khi ghi vào đường dẫn output\_path.   **Hàm perform\_analysis\_sql**  **Chức năng: Thực hiện phân tích bằng SQL queries thông qua TempView.**   1. Tạo TempView:    * Tạo view tạm thời users liên kết với DataFrame đầu vào. 2. Viết SQL query:    * Lọc các dòng có gender là 'male'.    * Tăng giá trị age lên 1.    * Sắp xếp kết quả theo:      + age giảm dần (DESC).      + name tăng dần (ASC). 3. Ghi kết quả ra file CSV:    * Tái phân phối dữ liệu thành 1 file duy nhất trước khi ghi vào đường dẫn output\_path. 4. Xóa TempView:    * Giải phóng bộ nhớ bằng cách xóa TempView users.   **Hàm main**   1. Tạo Spark Session:    * Khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-47. 2. Xác định đường dẫn:    * input\_path: Đường dẫn đến file CSV đầu vào.    * output\_df\_path và output\_sql\_path: Đường dẫn lưu kết quả của từng phương pháp. 3. Đọc dữ liệu:    * Dữ liệu đầu vào được đọc từ file CSV thành DataFrame với:      + header=True: File CSV có header.      + inferSchema=True: Tự động suy luận kiểu dữ liệu. 4. Thực hiện phân tích:    * Gọi hàm perform\_analysis\_dataframe để phân tích bằng DataFrame API.    * Gọi hàm perform\_analysis\_sql để phân tích bằng SQL queries. 5. Dừng Spark Session:    * Giải phóng tài nguyên bằng cách gọi spark.stop().   **Chạy chương trình**  **Spark-submit Final\_Bigdata/ex-47/code.py**    Kiểm tra kết quả     * + Đã có 2 folder chứa kết quả xuất hiện   + Kết quả chi tiết |
|  |
|  |

|  |
| --- |
| **Bài 48**    Giải thích code:  **import các thư viện cần thiết**  from pyspark.sql import SparkSession  from pyspark.sql.types import IntegerType   * **SparkSession**: Để khởi tạo phiên Spark. * **IntegerType**: Được sử dụng để ép kiểu dữ liệu của một cột.   **Hàm perform\_analysis\_dataframe**  def perform\_analysis\_dataframe(df, output\_path):  df\_profiles = df.groupBy('name').agg({'age':'avg', '\*':'count'}) \  .withColumnRenamed("avg(age)", "avg\_age") \  .withColumnRenamed("count(1)", "counter")  result\_df = df\_profiles.filter(df\_profiles.counter > 1).select("name", "avg\_age")  result\_df = result\_df.withColumn("avg\_age", result\_df["avg\_age"].cast(IntegerType()))  result\_df.repartition(1).write.csv(output\_path, mode='overwrite')   1. **Nhóm dữ liệu theo cột name**:    * **agg({'age':'avg', '\*':'count'})**:      + Tính **trung bình tuổi (avg)**.      + Đếm số lần xuất hiện của mỗi tên (count). 2. **Đổi tên các cột**:    * Đổi tên cột kết quả thành:      + avg(age) → **avg\_age**.      + count(1) → **counter**. 3. **Lọc kết quả**:    * Chỉ giữ lại những dòng có counter > 1 (tên xuất hiện hơn 1 lần). 4. **Ép kiểu dữ liệu**:    * Chuyển cột avg\_age thành kiểu **Integer**. 5. **Ghi kết quả vào file CSV**:    * Sử dụng **repartition(1)** để đảm bảo chỉ ghi vào **1 file duy nhất**.    * **mode='overwrite'**: Ghi đè nếu file đã tồn tại.   **Hàm perform\_analysis\_sql**  def perform\_analysis\_sql(session, df, output\_path):  df.createOrReplaceTempView("users")  df\_2 = session.sql("select name, avg(age) FROM users GROUP BY name HAVING count(\*) > 1;")  df\_2.repartition(1).write.csv(output\_sql\_path, mode='overwrite')  session.catalog.dropTempView("users")   1. **Tạo TempView**:    * Liên kết DataFrame df với một view tạm thời tên **users**. 2. **Thực thi SQL query**:   sql  Sao chép mã  SELECT name, avg(age)  FROM users  GROUP BY name  HAVING count(\*) > 1;   * + Tính trung bình tuổi (avg(age)) của các tên xuất hiện hơn 1 lần (HAVING count(\*) > 1).  1. **Ghi kết quả vào file CSV**:    * Tương tự như phương thức DataFrame API, kết quả được ghi vào **1 file duy nhất**. 2. **Xóa TempView**:    * Giải phóng bộ nhớ sau khi hoàn thành.   **Hàm main**  if \_\_name\_\_ == "\_\_main\_\_":  spark = SparkSession.builder.appName('Exercise-48').getOrCreate()  input\_path = "/Final\_Bigdata/ex-48/input.csv"  output\_df\_path = "/Final\_Bigdata/ex-48/output/result\_dataframe\_mode"  output\_sql\_path = "/Final\_Bigdata/ex-48/output/result\_sql\_mode"  df = spark.read.csv(input\_path, header=True, inferSchema=True)  perform\_analysis\_dataframe(df, output\_df\_path)  perform\_analysis\_sql(spark, df, output\_sql\_path)  spark.stop()  **Chạy chương trình :**  **Spark-submit Final\_Bigdata/ex-48/code.py** |
| Kiểm tra kết quả     * + Đã có 2 folder chứa kết quả xuất hiện   + Kết quả |
| **Bài 49**    **Import thư viện**   * **SparkSession**: Để khởi tạo phiên làm việc Spark. * **IntegerType, StringType**: Các kiểu dữ liệu Spark SQL. * **udf**: Được dùng để tạo **User Defined Function (UDF)** trong PySpark.   **UDF parse\_age**  Hàm này định nghĩa một **User-Defined Function**:   * **Mục đích**: Phân loại giá trị age thành các khoảng tuổi. * **Logic**:   "[{}-{}]".format(int(age/10)\*10, int(age/10)\*10+9)   * + Tuổi được chia thành các khoảng 10 năm một. Ví dụ:     - 23 → [20-29]     - 45 → [40-49]   **Hàm perform\_analysis\_dataframe**  **Chức năng**: Thực hiện phân tích dữ liệu bằng DataFrame API.   1. **Thêm cột mới**:    * Sử dụng UDF parse\_age để tính giá trị cho cột rangeage.    * Chỉ giữ lại các cột: name, surname, rangeage. 2. **Ghi kết quả ra file CSV**:    * Dữ liệu được tái phân vùng thành **1 file duy nhất** để ghi kết quả vào đường dẫn output\_path.    * Thêm header cho file CSV.   **Kết quả**: File CSV chứa các cột name, surname, rangeage.  **Hàm perform\_analysis\_sql**  **Chức năng**: Thực hiện phân tích bằng SQL queries thông qua TempView.   1. **Đăng ký UDF**:    * Đăng ký hàm parse\_age dưới dạng UDF SQL với tên range\_age\_func. 2. **Tạo TempView**:    * Tạo một TempView tên là users từ DataFrame đầu vào. 3. **Chạy SQL query**:    * SQL query để tính cột rangeage bằng cách gọi UDF:   sql  Sao chép mã  SELECT name, surname, range\_age\_func(age) AS rangeage  FROM users;   1. **Ghi kết quả ra file CSV**:    * Kết quả được tái phân vùng thành **1 file duy nhất** và ghi vào output\_path với header. 2. **Xóa TempView**:    * Giải phóng bộ nhớ bằng cách xóa TempView users.   **Hàm main**   1. **Tạo Spark Session**:    * Khởi tạo một Spark session với tên ứng dụng là Exercise-49. 2. **Đường dẫn**:    * input\_path: Đường dẫn file CSV đầu vào.    * output\_df\_path và output\_sql\_path: Đường dẫn lưu kết quả. 3. **Đọc dữ liệu đầu vào**:    * File CSV được đọc thành DataFrame với:      + header=True: File có header.      + inferSchema=True: Tự động suy luận kiểu dữ liệu. 4. **Thực hiện phân tích**:    * Gọi hàm perform\_analysis\_dataframe để phân tích bằng **DataFrame API**.    * Gọi hàm perform\_analysis\_sql để phân tích bằng **SQL queries**. 5. **Kết thúc chương trình**:    * Dừng Spark session để giải phóng tài nguyên. |
| Kiểm tra kết quả     * + Đã có 2 folder chứa kết quả xuất hiện   + Kết quả |
| **Bài 50** |

Giải thích code:

**Import các thư viện cần thiết**

from pyspark.sql import SparkSession

from pyspark.sql.types import IntegerType, StringType

from pyspark.sql.functions import udf

* **SparkSession**: Tạo Spark Session.
* **udf**: Định nghĩa User-Defined Function (hàm do người dùng tự viết).

**2.2 Định nghĩa UDF parse\_columns**

@udf

def parse\_columns(name, surname):

return name+""+surname

* Hàm **parse\_columns**: Nhận vào 2 cột **name** và **surname**, kết hợp chúng thành một chuỗi mới.

**Hàm perform\_analysis\_dataframe**

def perform\_analysis\_dataframe(df, output\_path):

df\_result = df.withColumn("name\_surname", parse\_columns(df.name, df.surname)) \

.select("name\_surname")

df\_result.repartition(1).write.csv(output\_path, mode="overwrite", header=True)

1. **Thêm cột mới**:
   * Sử dụng UDF parse\_columns để kết hợp name và surname thành cột mới **name\_surname**.
2. **Chọn cột cần xuất**:
   * Chỉ chọn cột name\_surname để ghi vào file kết quả.
3. **Ghi kết quả vào file CSV**:
   * **repartition(1)**: Đảm bảo kết quả chỉ ghi vào **1 file duy nhất**.
   * **header=True**: Ghi tiêu đề cột vào file CSV.
   * **mode="overwrite"**: Ghi đè nếu file đã tồn tại.

**Hàm perform\_analysis\_sql**

def perform\_analysis\_sql(session, df, output\_path):

session.udf.register("name\_surname\_func", parse\_columns)

df.createOrReplaceTempView("users")

df\_2 = session.sql("select name\_surname\_func(name, surname) as name\_surname FROM users;")

df\_2.repartition(1).write.csv(output\_sql\_path, mode='overwrite', header=True)

session.catalog.dropTempView("users")

1. **Đăng ký UDF vào Spark Session**:
   * **session.udf.register** đăng ký hàm parse\_columns với tên **name\_surname\_func**.
2. **Tạo TempView**:
   * Tạo view tạm thời tên **users** để truy vấn dữ liệu từ DataFrame.
3. **Thực thi SQL Query**:

sql

Sao chép mã

SELECT name\_surname\_func(name, surname) AS name\_surname

FROM users;

* + Sử dụng UDF name\_surname\_func để kết hợp **name** và **surname**.

1. **Ghi kết quả vào file CSV**:
   * **repartition(1)**: Ghi kết quả vào 1 file duy nhất.
   * **header=True**: Ghi tiêu đề cột vào file CSV.
2. **Xóa TempView**:
   * Dọn dẹp bộ nhớ bằng cách xóa TempView **users**.

**Hàm main**

1. **Khởi tạo Spark Session**:
   * Tạo phiên Spark với tên **Exercise-50**.
2. **Đọc dữ liệu đầu vào**:
   * **header=True**: File CSV có chứa tiêu đề.
   * **inferSchema=True**: Tự động xác định kiểu dữ liệu.
3. **Gọi hai phương thức phân tích**:
   * **perform\_analysis\_dataframe**: Sử dụng DataFrame APIs.
   * **perform\_analysis\_sql**: Sử dụng SQL queries.
4. **Dừng Spark Session**:
   * Giải phóng tài nguyên.

Thực hiện

A screenshot of a computer

Description automatically generated

Kiểm tra xem kết quả

A screenshot of a computer

Description automatically generated

* + Đã thấy 2 folder chứa kết quả
  + Kết quả

A screenshot of a computer

Description automatically generated

# **CHƯƠNG 5: KẾT LUẬN**

## **5.1 Kết quả đạt được**

* + Xây dựng pipeline bằng PySpark để xử lý dữ liệu nhanh chóng, tự động, và hiệu quả trên bộ dữ liệu lớn.
  + Phân tích chi tiết các yếu tố ảnh hưởng đến tai nạn giao thông, bao gồm thời gian, địa điểm, điều kiện thời tiết và các tham số khác.
  + Thiết kế dashboard tương tác bằng Power BI, cung cấp cái nhìn trực quan và toàn diện về tình hình tai nạn giao thông.

**5.2 Ưu điểm**

* + Sử dụng PySpark giúp xử lý dữ liệu nhanh chóng, đáp ứng nhu cầu xử lý khối lượng lớn.
  + Power BI cung cấp giao diện trực quan, dễ dàng truy vấn, đồng thời cung cấp những biểu đồ sinh động, giúp người dùng nhanh chóng hiểu được dữ liệu.
  + Đề tài tập trung vào tính ứng dụng cao trong việc hỗ trợ quyết định giao thông và đề xuất giải pháp.

## **5.3 Hạn chế**

* + PySpark yêu cầu cấu hình hạ tầng phức tạp, đòi hỏi khả năng kỹ thuật cao.
  + Chất lượng dữ liệu gốc không đồng đều, một số trường dữ liệu bị thiếu hoặc sai lệch.
  + Power BI có những giới hạn về xử lý dữ liệu khi khởi lượng quá lớn hoặc yêu cầu phân tích phức tạp.

## **5.4 Hướng phát triển**

* + Tích hợp nhiều nguồn dữ liệu hơn, bao gồm các đồng góp từ các hệ thống IoT và các ngịuồn dữ liệu thời gian thực.
  + Đồng bộ dữ liệu và phân tích nâng cao nhờ AI/ML để tự động phát hiện xu hướng và cảnh báo nguy cơ.
  + Xây dựng các đối tượng trực quan hóa mới, tăng khả năng tương tác và tính trực quan.

# **TÀI LIỆU THAM KHẢO**

Tài liệu tham khảo apache spark <https://aws.amazon.com/vi/what-is/apache-spark/>  
Tài liệu tham khảo power BI

* <https://datapot.vn/power-bi-cong-cu-hoan-hao-cho-data-analyst/>
* <https://powerbi03.gitiho.com/>

Nguồn dữ liệu: <https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents/data>

[chat gpt](file:///D:\Big_data_analys\CuoiKy\chat%20gpt)