

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**  
**KHOA HỆ THỐNG THÔNG TIN**



**MÔN HỌC: DỮ LIỆU LỚN**  
**ĐỒ ÁN CUỐI KỲ**

**DỰ ĐOÁN CÁC BỆNH NHÂN CÓ NGUY  
CƠ BỊ TAI BIẾN MẠCH MÁU NÃO**

**GVHD:** ThS. Nguyễn Hồ Duy Tri

**LỚP:** IS405.P11.HTCL

**SINH VIÊN THỰC HIỆN:**

Châu Thanh Bình	21521871
Nguyễn Thị Thảo Ngân	21521176
Ngô Anh Tuấn	21521629
Phi Quang Thành	21521449

TP Hồ Chí Minh, tháng 12 năm 2024

## LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành và sự tri ân sâu sắc đối với các Thầy Cô của **Trường Đại học Công nghệ Thông tin – Đại học Quốc Gia TP.HCM**, đặc biệt là quý **Thầy Cô khoa Hệ Thống Thông Tin** của Trường đã giúp cho chúng em trang bị các kiến thức cơ bản, các kỹ năng thực tế và tạo điều kiện để chúng em có thể hoàn thành đồ án môn học của mình.

Đặc biệt nhóm xin gửi lời cảm ơn chân thành tới **Thầy Nguyễn Hồ Duy Tri** – Giảng viên lý thuyết môn **Dữ liệu lớn** đã tận tình giúp đỡ, trực tiếp chỉ bảo, hướng dẫn nhóm trong suốt quá trình làm đồ án môn học. Nhờ đó, nhóm em đã tiếp thu được nhiều kiến thức bổ ích trong việc vận dụng cũng như kỹ năng làm đồ án. Một lần nữa, nhóm xin chân thành cảm ơn Thầy.

Dựa trên những kiến thức được Thầy cung cấp trên trường kết hợp với việc tự tìm hiểu những công cụ và kiến thức mới, nhóm đã cố gắng thực hiện đồ án một cách tốt nhất. Trong thời gian một học kỳ thực hiện đề tài, nhóm đã vận dụng những kiến thức nền tảng đã tích lũy đồng thời kết hợp với việc học hỏi và nghiên cứu những kiến thức mới. Từ đó, em vận dụng tối đa những gì đã thu thập được để hoàn thành một báo cáo đồ án tốt nhất.

Tuy nhiên, trong quá trình thực hiện, nhóm không tránh khỏi những thiếu sót. Chính vì vậy, em rất mong nhận được những sự góp ý từ phía Thầy nhằm hoàn thiện những kiến thức mà em đã học tập và là hành trang để em thực hiện tiếp các đề tài khác trong tương lai.

## NHẬN XÉT CỦA GIẢNG VIÊN

[illegible]

# MỤC LỤC

LỜI CẢM ƠN.....	2
NHẬN XÉT CỦA GIẢNG VIÊN .....	3
MỤC LỤC .....	4
DANH MỤC HÌNH ẢNH .....	5
DANH MỤC BẢNG .....	6
CHƯƠNG 1: MÔ TẢ BÀI TOÁN.....	7
1.1 Lý do lựa chọn bài toán .....	7
1.2 Mô tả dữ liệu .....	7
1.3 Thuộc tính dữ liệu .....	8
1.4 Mô tả bài toán .....	14
CHƯƠNG 2: CÁC PHƯƠNG PHÁP, KỸ THUẬT, THUẬT TOÁN LỰA CHỌN.....	15
2.1 Kỹ thuật tiền xử lý dữ liệu .....	15
2.1.1 StringIndexer .....	15
2.1.2 MinMaxScaler .....	16
2.2 Thuật toán khai thác dữ liệu (K-Nearest Neighbors).....	16
CHƯƠNG 3: TIỀN XỬ LÝ DỮ LIỆU .....	18
3.1 Import dữ liệu .....	18
3.2 Đọc và xem dữ liệu .....	19
3.3 Kiểm tra và xử lý các dữ liệu trống (NULL).....	20
3.4 Kiểm tra giá trị không hợp lệ .....	20
3.5 Chuyển đổi kiểu dữ liệu object sang integer .....	22
3.6 Chuẩn hóa dữ liệu .....	25
3.7 Kiểm tra phân phối các lớp trong cột stroke .....	27
3.8 Xuất file sau khi tiền xử lý dữ liệu .....	27
CHƯƠNG 4: TRIỂN KHAI THUẬT TOÁN .....	28
4.1 Các bước thực hiện thuật toán KNN .....	28
4.2 Chạy thuật toán .....	30
CHƯƠNG 5: KẾT QUẢ ĐẠT ĐƯỢC.....	33
5.1 Ý nghĩa các độ đo và lý do lựa chọn.....	33
5.2 Phát biểu kết quả.....	36
5.3 So sánh, đánh giá .....	39
CHƯƠNG 6: KẾT LUẬN.....	40
6.1 Ưu điểm .....	40

6.2 Hạn chế.....	41
6.3 Hướng phát triển .....	41
<b>PHÂN CÔNG CÔNG VIỆC .....</b>	<b>42</b>
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>43</b>

## DANH MỤC HÌNH ẢNH

Hình 1. Trang Database.....	7
Hình 2. Biểu đồ phân loại độ tuổi (age) .....	9
Hình 3. Biểu đồ phân loại giới tính (gender).....	10
Hình 4. Biểu đồ phân loại tình trạng hôn nhân.....	11
Hình 5. Biểu đồ mức glucose trung bình trong máu .....	12
Hình 6. Biểu đồ chỉ số BMI.....	12
Hình 7. Biểu đồ phân loại nghề nghiệp .....	13
Hình 8. Biểu đồ phân loại thói quen hút thuốc.....	14
Hình 9. Min-Max Scaling.....	16
Hình 10. Phân loại (Classification) .....	17
Hình 11. Import thư viện cần thiết .....	19
Hình 12. Tạo SparkSession .....	19
Hình 13. Đọc file CSV .....	19
Hình 14. Kiểm tra dữ liệu.....	20
Hình 15. Đếm số lượng giá trị NULL .....	20
Hình 16. Giá trị NULL .....	20
Hình 17. Giá trị không hợp lệ trong từng cột (1).....	21
Hình 18. Giá trị không hợp lệ trong từng cột (2).....	21
Hình 19. Thay thế các giá trị N/A .....	22
Hình 20. Thay thế các giá trị Unknown .....	22
Hình 21. Chuyển thuộc tính object sang integer .....	23
Hình 22. Chuyển đổi các thuộc tính cần thiết.....	24
Hình 23. Danh sách các thuộc tính cần thiết .....	24
Hình 24. Tạo bảng label sau khi áp dụng StringIndexer .....	24
Hình 25. Danh sách label sau chuyển đổi (1).....	25
Hình 26. Danh sách label sau chuyển đổi (2).....	25
Hình 27. Chuẩn hóa dữ liệu.....	26
Hình 28. Kết quả sau khi chuẩn hóa.....	27
Hình 29. Kiểm tra phân phối cột stroke .....	27
Hình 30. Xuất file CSV .....	27

Hình 31. Flowchart của KNN.....	28
Hình 32. Khai báo thư viện .....	30
Hình 33. Chia, cân bằng dữ liệu .....	30
Hình 34. Hàm tính Euclidean .....	31
Hình 35. Thực hiện KNN .....	31
Hình 36. Hàm Evaluate Predictions .....	32
Hình 37. Tìm giá trị k.....	32
Hình 38. Các chỉ số đánh giá hiệu quả mô hình.....	33
Hình 39. Confusion Matrix (CM).....	34
Hình 40. Giá trị đo khi k từ 1 đến 20.....	36
Hình 41. Biểu đồ F1-Score.....	36
Hình 42. Biểu đồ Accuracy .....	37
Hình 43. Biểu đồ so sánh 2 giá trị .....	37
Hình 44. Biểu đồ Confusion Matrix .....	38
Hình 45. Các độ đo.....	39

## DANH MỤC BẢNG

Bảng 1. Bảng mô tả các thuộc tính của dataset .....	8
Bảng 2. Bảng một số thông kê các thuộc tính của database .....	8
Bảng 3. Phân công công việc .....	42

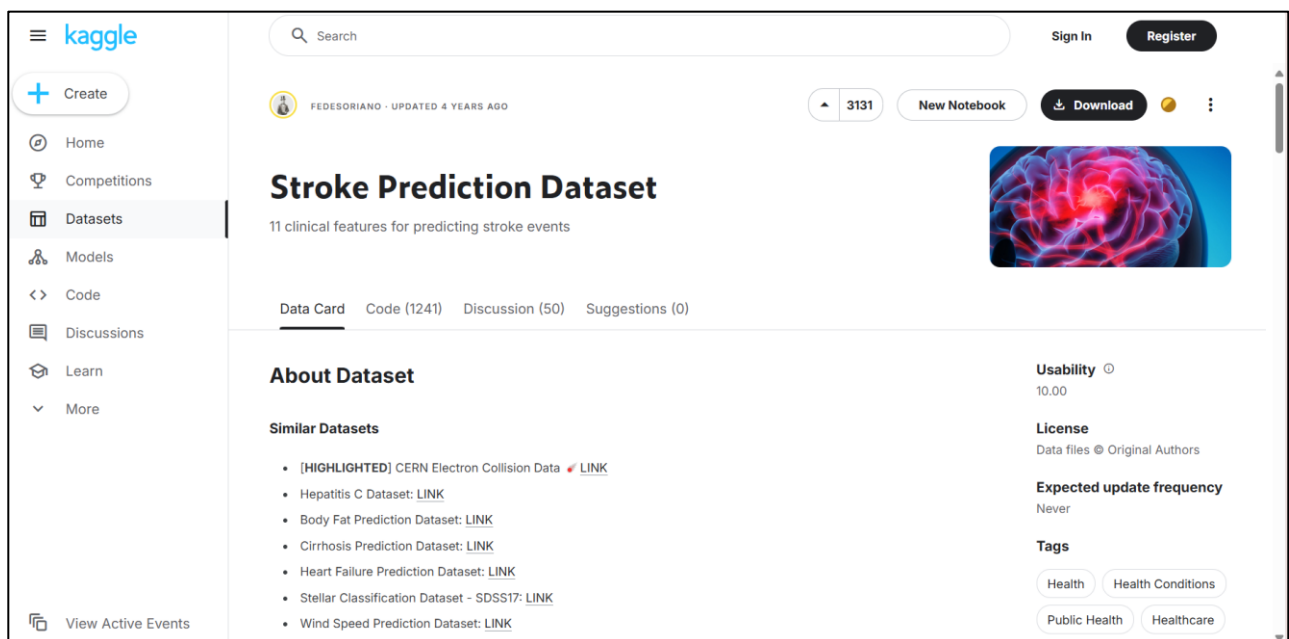
# CHƯƠNG 1: MÔ TẢ BÀI TOÁN

## 1.1 Lý do lựa chọn bài toán

Theo thống kê của Tổ chức Y tế Thế giới (WHO) năm 2021, tai biến mạch máu não đứng thứ 2 về nguyên nhân gây tử vong trên toàn cầu, chiếm 11% tổng số ca tử vong. Điều này không chỉ ảnh hưởng nghiêm trọng đến chất lượng cuộc sống của người bệnh mà còn đặt gánh nặng lớn lên hệ thống y tế và kinh tế xã hội toàn cầu nói chung và Việt Nam nói riêng.[1], [2]

Với mong muốn góp phần giải quyết vấn đề này, nhóm đã lựa chọn nghiên cứu và thực hiện dự đoán các bệnh nhân có nguy cơ bị tai biến mạch máu não. Từ đó các đơn vị có chuyên môn có thể hiểu rõ hơn về số liệu và đưa ra các biện pháp phòng chống hợp lý.

## 1.2 Mô tả dữ liệu



Hình 1. Trang Database

- Dữ liệu các ca tử vong vì tai biến mạch máu não được Public trên Kaggle năm 2020.
- Các thông tin trong dataset gồm: Giới tính, độ tuổi, huyết áp cao, bệnh lý về tim,...
- Dữ liệu gồm 12 cột và 5110 dòng.
- Link dữ liệu: [Stroke Prediction Dataset](#)

## 1.3 Thuộc tính dữ liệu

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	id	int	Mã bệnh nhân
2	gender	string	Giới tính bệnh nhân
3	age	float	Số tuổi bệnh nhân
4	hypertension	int	Bệnh nhân bị cao huyết áp hay không?
5	heart_disease	int	Bệnh nhân đã từng đó có bệnh lý về tim hay không?
6	ever_married	boolean	Bệnh nhân đã có vợ/chồng chưa?
7	work_type	string	Nghề nghiệp của bệnh nhân
8	residence_type	string	Khu vực sống của bệnh nhân
9	avg_glucose_level	float	Định lượng glucose trong máu của bệnh nhân
10	bmi	float	Chỉ số BMI của bệnh nhân
11	smoking_status	string	Tiểu sử hút thuốc của bệnh nhân
12	stroke	int	Liệu bệnh nhân đó có bị tai biến hay không

Bảng 1. Bảng mô tả các thuộc tính của dataset

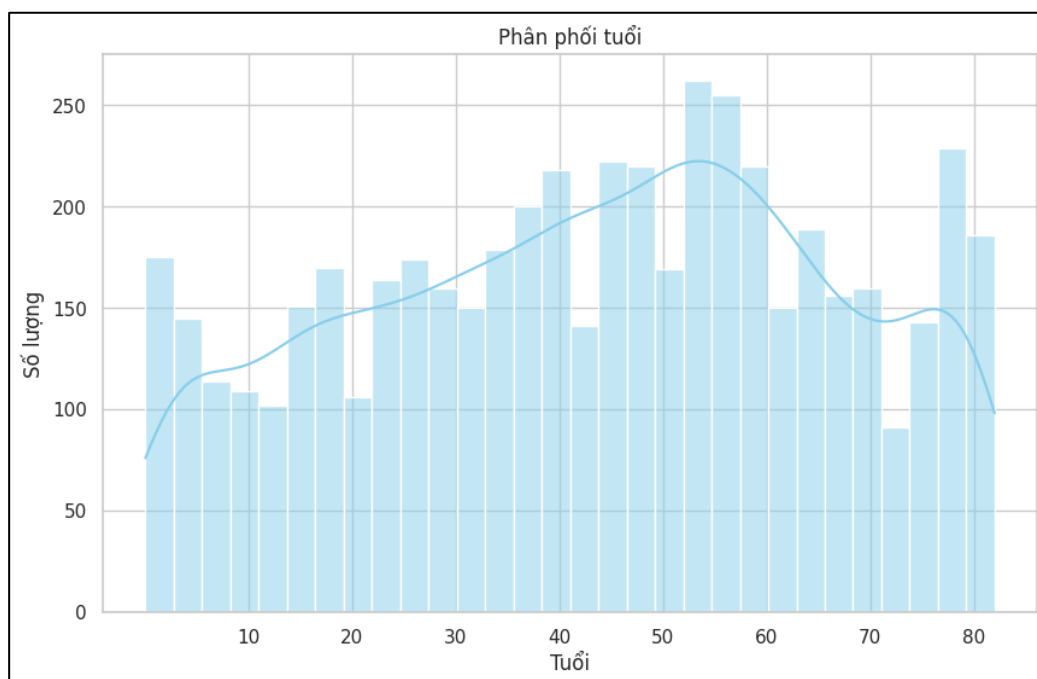
### Một số thống kê về tập dữ liệu:

STT	Tên thuộc tính	Kiểu dữ liệu	Count	Mean	Std	Max	Min
1	id	int	5110	NULL	NULL	NULL	NULL
2	gender	string	5110	NULL	NULL	NULL	NULL
3	age	float	5110	43.227	22.613	82	0
4	hypertension	int	5110	NULL	NULL	NULL	NULL
5	heart_disease	int	5110	NULL	NULL	NULL	NULL
6	ever_married	boolean	5110	NULL	NULL	NULL	NULL
7	work_type	string	5110	NULL	NULL	NULL	NULL
8	residence_type	string	5110	NULL	NULL	NULL	NULL
9	avg_glucose_level	float	5110	106.148	45.284	271.74	55.12
10	bmi	float	5110	27.757	9.529	97.6	0
11	smoking_status	string	5110	NULL	NULL	NULL	NULL
12	stroke	int	5110	NULL	NULL	NULL	NULL

Bảng 2. Bảng một số thông kê các thuộc tính của database



## Khám phá thuộc tính tuổi (age)



Hình 2. Biểu đồ phân loại độ tuổi (age)

Biểu đồ phân loại độ tuổi (age): Biểu đồ cột thể hiện tổng số lượng người bị tai biến theo từng độ tuổi khác nhau.

**Nhận xét:** Số lượng người mắc bệnh dao động ở các độ tuổi. Riêng độ tuổi từ 50 đến 60 có số lượng người tử vong do bệnh cao nhất. Một phần nhỏ người trẻ mắc bệnh rơi vào độ tuổi từ 5 đến 20.

## Khám phá thuộc tính giới tính (gender)

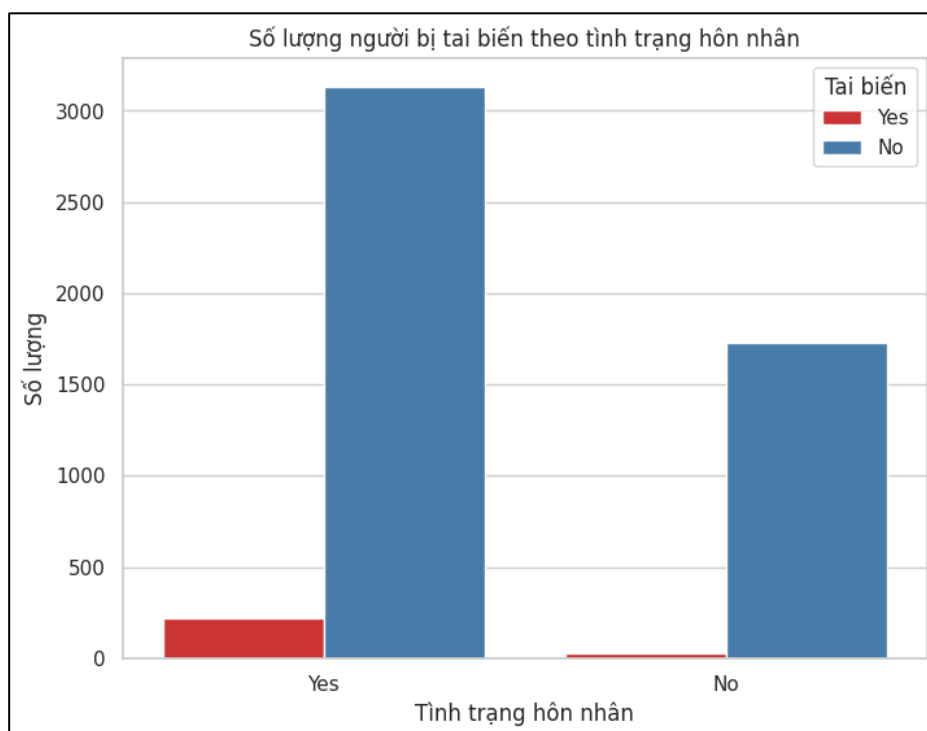


Hình 3. Biểu đồ phân loại giới tính (gender)

Biểu đồ phân loại giới tính (gender): Biểu đồ cột thể hiện số lượng người mắc bệnh theo từng giới tính khác nhau.

**Nhận xét:** Số lượng phụ nữ trong dataset nhiều hơn đàn ông và số lượng mắc bệnh cũng có phần nhỉnh so với phần còn lại. Người thuộc giới tính khác chiếm cực nhỏ trong biểu đồ, hầu như là không có.

### Khám phá thuộc tính đã kết hôn (ever\_married)

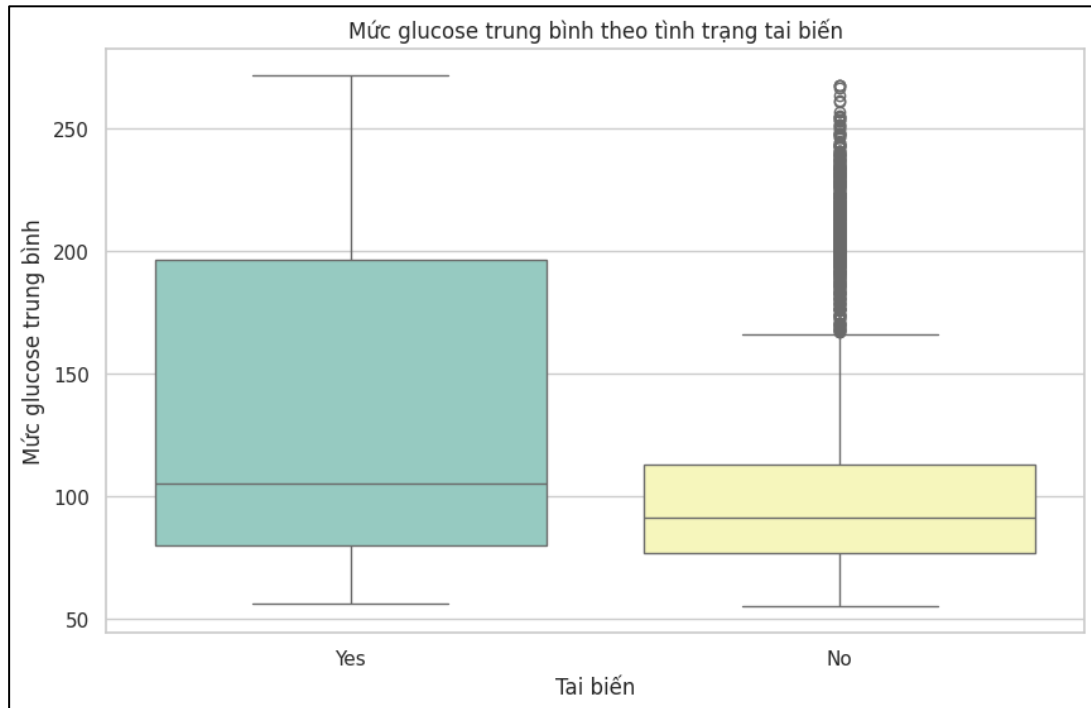


Hình 4. Biểu đồ phân loại tình trạng hôn nhân

Biểu đồ thể hiện mối quan hệ giữa tình trạng hôn nhân (Yes: Đã kết hôn, No: Chưa kết hôn) và số lượng người bị tai biến mạch máu não (Yes: Bị tai biến, No: Không bị tai biến).

**Nhận xét:** Số người không bị tai biến (No) áp đảo trong cả hai nhóm tình trạng hôn nhân, đặc biệt nhóm đã kết hôn có số lượng cao vượt trội. Nhóm đã kết hôn (Yes) chiếm đa số trong tổng số người được khảo sát, nhưng số người bị tai biến trong nhóm này vẫn thấp.

**Khám phá thuộc tính mức Glucoso trung bình trong máu (avg\_glucose\_level)**

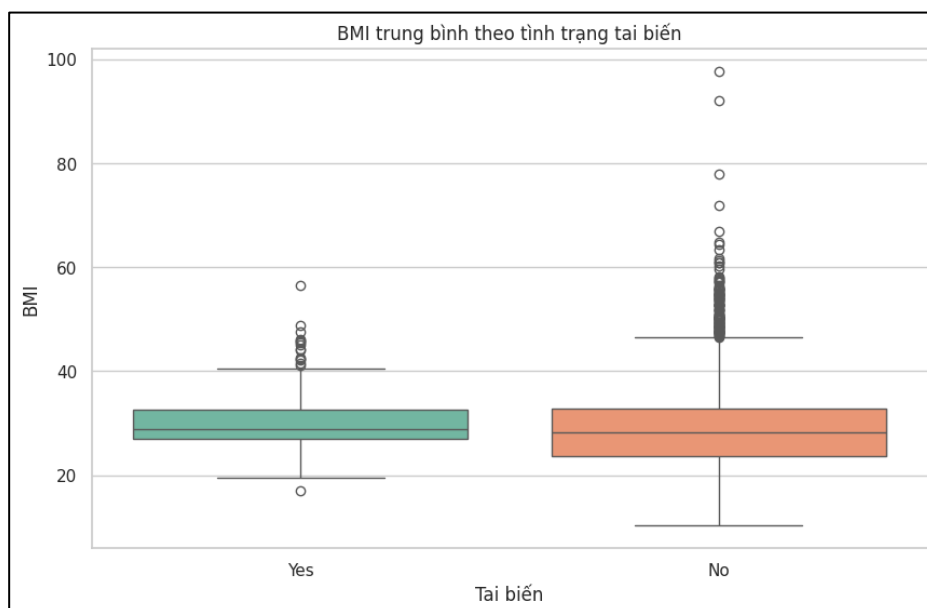


Hình 5. Biểu đồ mức glucose trung bình trong máu

Biểu đồ thể hiện mức glucose trong máu trung bình theo tình trạng tai biến (Yes: Bị tai biến, No: Không bị tai biến)

**Nhận xét:** Mức glucose trong máu trung bình của nhóm có tai biến (Yes) cao hơn đáng kể so với nhóm không có tai biến (No). Bên cạnh đó, trong nhóm người bị tai biến (Yes), mức đường huyết dao động rất lớn, từ rất thấp đến rất cao.

### Khám phá thuộc tính BMI (bmi)

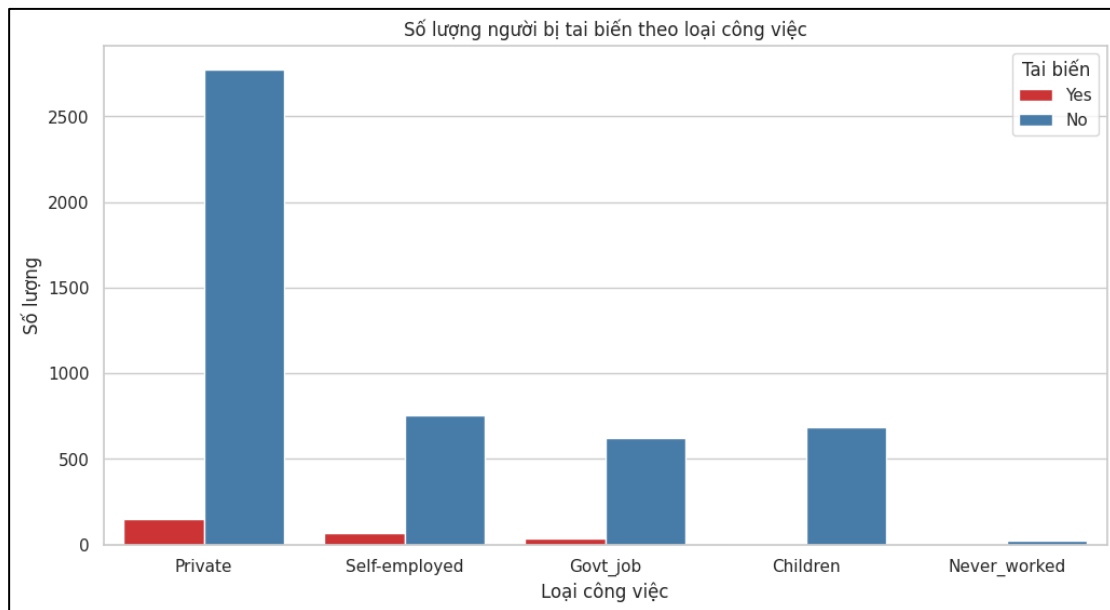


Hình 6. Biểu đồ chỉ số BMI

Biểu đồ thể hiện chỉ số BMI trung bình theo tình trạng tai biến (Yes: Bị tai biến, No: Không bị tai biến)

**Nhận xét:** BMI trung bình của nhóm bị tai biến (Yes) cao hơn so với nhóm không bị tai biến (No). Bên cạnh đó, trong nhóm người không bị tai biến (No), mức độ dao động của chỉ số BMI cao.

### Khám phá thuộc tính nghề nghiệp (work\_type)

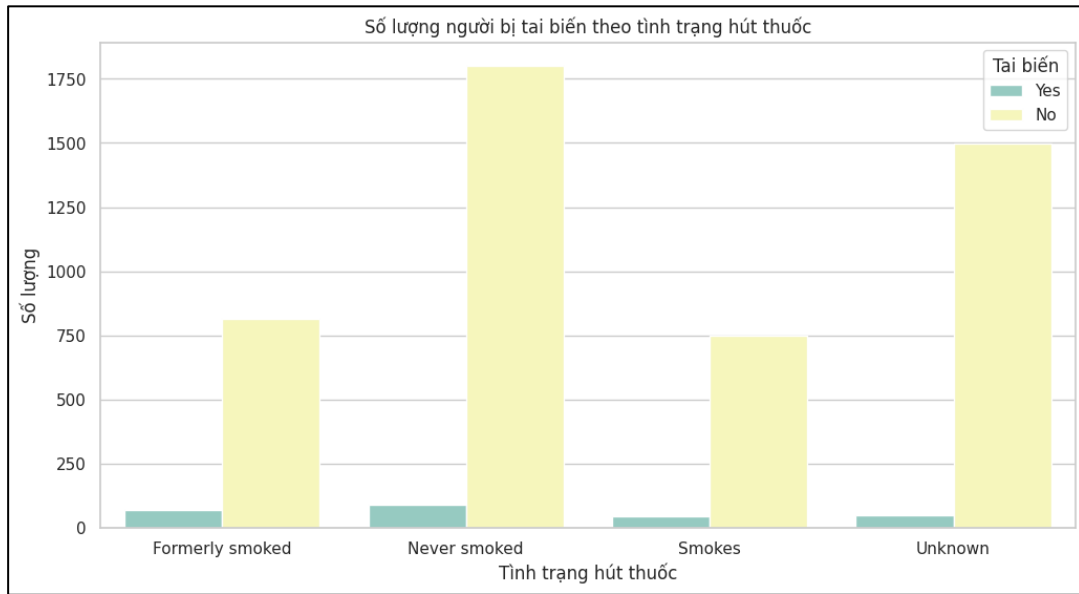


Hình 7. Biểu đồ phân loại nghề nghiệp

Biểu đồ cột thể hiện số lượng người bị tai biến theo loại công việc (Private: Tự nhân, Self-employed: Tự do, Govt\_job: Nhà nước, Children: Trẻ em, Never\_worked: Chưa đi làm).

**Nhận xét:** Nhóm người làm việc trong khu vực tư nhân (Private) có số lượng người bị tai biến cao nhất. Bên cạnh đó, số lượng người bị tai biến ở các nhóm công việc khác tương đối thấp. Tuy nhiên, số lượng tổng thể người không bị tai biến (No) vẫn cao rõ rệt so với phần còn lại.

### Khám phá thuộc tính nghề nghiệp (work\_type)



Hình 8. Biểu đồ phân loại thói quen hút thuốc

Biểu đồ cột thể hiện mối liên hệ giữa thói quen hút thuốc và nguy cơ bị tai biến mạch máu não (Formerly smoked: Đã từng hút thuốc, Never smoked: Chưa từng hút thuốc, Smokes: Đang hút thuốc, Unknown: Chưa rõ).

**Nhận xét:** Nhóm đã từng hút thuốc (Formerly smoked) và đang hút thuốc (Smokes) có số lượng người bị tai biến cao hơn so với nhóm chưa từng hút thuốc (Never smoked). Nhóm chưa rõ tình trạng hút thuốc (Unknown) cũng có tỷ lệ bị tai biến cao hơn nhóm chưa bao giờ hút thuốc. Tuy nhiên, số lượng tổng thể người không bị tai biến (No) vẫn cao rõ rệt so với phần còn lại.

## 1.4 Mô tả bài toán

Trong báo cáo này, nhóm thực hiện phương pháp phân loại bằng thuật toán KNN kết hợp với xử lý dữ liệu bằng Apache Spark để dự đoán nguy cơ đột quỵ của bệnh nhân. Dựa trên sự phân tích các đặc trưng như tuổi, bệnh nền, chỉ số BMI, mức đường huyết, và thói quen hút thuốc, để tìm ra các yếu tố ảnh hưởng đến nguy cơ đột quỵ. Kết quả của mô hình sẽ giúp hỗ trợ các chuyên gia y tế trong việc phát hiện sớm các trường hợp có nguy cơ cao và đề xuất các biện pháp can thiệp phù hợp nhằm giảm thiểu rủi ro cho bệnh nhân.

# CHƯƠNG 2: CÁC PHƯƠNG PHÁP, KỸ THUẬT, THUẬT TOÁN LỰA CHỌN

## 2.1 Kỹ thuật tiền xử lý dữ liệu

### 2.1.1 StringIndexer

**StringIndexer** là một kỹ thuật dùng để chuyển đổi dữ liệu dạng chuỗi thành các giá trị số. Việc chuyển đổi này là cần thiết vì hầu hết các thuật toán học máy không thể làm việc trực tiếp với dữ liệu chuỗi, mà chỉ xử lý được dữ liệu số.[3]

**StringIndexer** là sự lựa chọn ưu tiên trong nhiều trường hợp vì nó đơn giản, hiệu quả, và không gây ra vấn đề về chiều không gian hoặc mất thông tin phân loại. Đặc biệt đối với thuật toán KNN, nơi cần tính toán khoảng cách giữa các điểm dữ liệu, việc sử dụng **StringIndexer** giúp giảm bớt các vấn đề về tính toán và tối ưu hóa hiệu suất của mô hình.

**StringIndexer** gán một chỉ mục duy nhất cho từng giá trị chuỗi riêng biệt trong cột dữ liệu đầu vào, sau đó ánh xạ nó tới cột đầu ra mới dưới dạng các giá trị số nguyên.[4]

**StringIndexer** xử lý các giá trị chuỗi của cột đầu vào dựa trên tần suất của chúng trong tập dữ liệu. Cụ thể, giá trị xuất hiện nhiều nhất sẽ được gán chỉ số 0, giá trị xuất hiện nhiều thứ hai sẽ có chỉ số 1, và tiếp tục như vậy. Nếu có hai hoặc nhiều giá trị có tần suất giống nhau, chỉ số sẽ được gán theo thứ tự bảng chữ cái. Ngoài ra, bạn cũng có thể điều chỉnh thứ tự này thông qua tham số 'stringOrderType' để gán chỉ số theo một thứ tự tùy chỉnh.

**Ví dụ:** Nếu ta có một cột Quốc gia như ["Canada", "Trung Quốc", "Nga", "Canada"], **StringIndexer** sẽ chuyển đổi cột này thành các chỉ số như [0, 1, 2, 0], giả sử "Canada" là quốc gia xuất hiện nhiều nhất.

**StringIndexer** thường được sử dụng như một bước trong pipeline học máy, giúp chuyển đổi dữ liệu chuỗi thành dạng số để mô hình có thể học từ đó. Nó có thể kết hợp với các bước tiền xử lý khác, như chuẩn hóa dữ liệu hay mã hóa đặc trưng, để tạo thành một pipeline dữ liệu hoàn chỉnh. Kỹ thuật này rất hữu ích trong các pipeline yêu cầu xử lý đồng thời nhiều loại dữ liệu khác nhau, bao gồm cả dữ liệu số và danh mục (chuỗi).

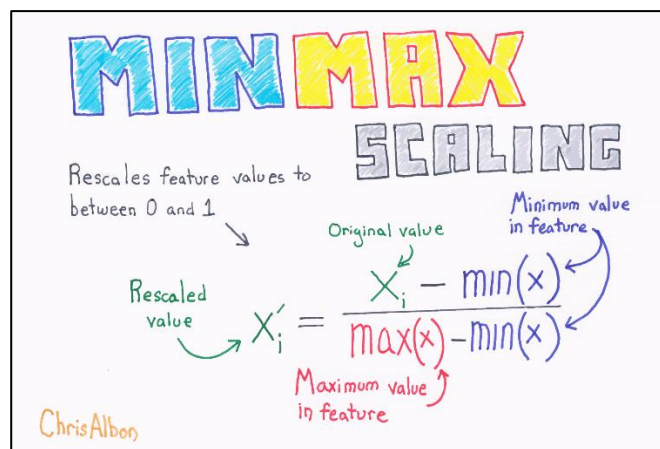
### 2.1.2 MinMaxScaler

**MinMaxScaler** là một kỹ thuật chuẩn hóa dữ liệu được sử dụng để chuẩn hóa các đặc trưng. Nó chuyển đổi các đặc trưng để nằm trong khoảng nhất định, thường là từ 0 đến 1.

Kỹ thuật này chủ yếu được sử dụng để làm cho dữ liệu có thể so sánh được và cải thiện hiệu suất của các thuật toán máy học nhạy cảm với quy mô của dữ liệu (ví dụ: mạng nơ-ron, K-Nearest Neighbors).

Công thức để chuẩn hóa một đặc trưng  $X$  trong khoảng  $[0, 1]$  là:

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$



Hình 9. Min-Max Scaling

Trong đó:

- $X$  là giá trị gốc
- $X_{\min}$  là giá trị nhỏ nhất của đặc trưng
- $X_{\max}$  là giá trị lớn nhất của đặc trưng

## 2.2 Thuật toán khai thác dữ liệu (K-Nearest Neighbors)

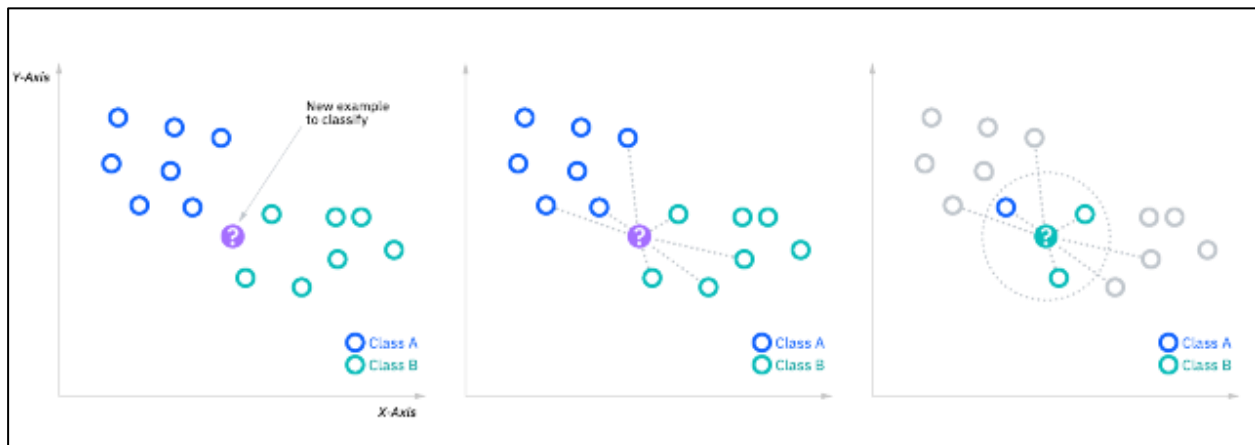
**Thuật toán K-Nearest Neighbors (K-NN)**, là một mô hình học máy (Machine Learning) có giám sát và không tham số (non-parametric supervised learning). Thuật toán sử dụng tính gần gũi giữa các điểm dữ liệu để phân loại hoặc dự đoán nhóm mà một điểm dữ liệu thuộc về. KNN có thể được sử dụng cho cả bài toán hồi quy (Regression) và phân loại



(Classification), nhưng nó chủ yếu được sử dụng trong phân loại với giả định rằng các điểm tương tự thường gần nhau.

### Trong bài toán phân loại:

- Một nhãn lớp sẽ được gán cho điểm dữ liệu dựa trên biểu quyết đa số (**Majority vote**). Nghĩa là, nhãn lớp xuất hiện nhiều nhất trong số các điểm gần nhất sẽ được chọn.
- Mặc dù thuật ngữ “biểu quyết đa số” thường được sử dụng, nhưng về mặt kỹ thuật, cách thức này là “biểu quyết đa nguyên” (**Plurality voting**). Cụ thể:
  - Biểu quyết đa số yêu cầu hơn 50% số phiếu, thích hợp khi chỉ có 2 lớp.
  - Biểu quyết đa nguyên: Khi có nhiều lớp, một lớp có thể được chọn chỉ cần hơn 25% số phiếu nếu có 4 lớp.



Hình 10. Phân loại (Classification)

**Mục tiêu của thuật toán K-NN** là xác định láng giềng gần nhất của một điểm cần phân loại (điểm truy vấn - query point), để gán nhãn lớp cho điểm này.

### Xác định cách đo khoảng cách (distance metric)

- Để tìm điểm gần nhất, khoảng cách giữa điểm truy vấn và các điểm dữ liệu khác cần được tính.
- Khoảng cách này hình thành nên **ranh giới quyết định (decision boundaries)** – phân chia không gian điểm thành các vùng ứng với từng lớp.

### Các công thức đo khoảng cách phổ biến

- **Khoảng cách Euclid (Euclidean distance):** Đây là công thức phổ biến nhất, dùng để đo khoảng cách đường thẳng giữa hai điểm. Công thức:

$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

- **Khoảng cách Manhattan (Manhattan distance):** Được gọi là khoảng cách taxicab hoặc city block distance, vì thường được hình dung trên lưới đường phố. Công thức:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

- **Khoảng cách Minkowski (Minkowski distance):** Đây là một dạng tổng quát của khoảng cách Euclid và Manhattan, với tham số  $p$  điều chỉnh loại khoảng cách. Công thức:

$$d(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- Khi  $p = 2$ , công thức trở thành khoảng cách Euclid.
- Khi  $p = 1$ , công thức trở thành khoảng cách Manhattan.
- **Khoảng cách Hamming (Hamming distance):** Thường được sử dụng với các vector Boolean hoặc chuỗi, đo số lượng vị trí khác nhau giữa hai vector. Công thức:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

## CHƯƠNG 3: TIỀN XỬ LÝ DỮ LIỆU

### 3.1 Import dữ liệu

Khi làm việc với PySpark, việc import các thư viện cần thiết là bước quan trọng đầu tiên trong bất kỳ script hoặc notebook nào. Các thư viện này cung cấp các chức năng và lớp cần thiết để xử lý dữ liệu, xây dựng mô hình, và thực hiện nhiều tác vụ khác trong môi trường PySpark. Thao tác này được thực hiện bằng cú pháp sau:

```
# Import các thư viện cần thiết
from pyspark import SparkContext
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import col, isnan, when, count, mean, round
from pyspark.ml.feature import MinMaxScaler, StringIndexer
from pyspark.ml import Pipeline
from pyspark.sql import functions as F
```

Hình 11. Import thư viện cần thiết

## 3.2 Đọc và xem dữ liệu

Trong môi trường PySpark, khởi tạo *SparkSession* là bước đầu tiên và quan trọng nhất trong việc thiết lập một phiên làm việc với Spark. Điều này được thực hiện thông qua lớp *SparkSession* từ module *pyspark.sql*

```
# Tạo SparkSession
spark = SparkSession.builder.appName("StrokePrediction").getOrCreate()
```

Hình 12. Tạo SparkSession

Sử dụng lệnh *spark.read.csv* để đọc file dữ liệu csv vào *DataFrame* và sử dụng hàm *show()* để xem tập dữ liệu và hàm *printSchema()* để kiểm tra cấu trúc dữ liệu

```
# Đọc file CSV
file_path = "healthcare-dataset-stroke-data.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)
```

Hình 13. Đọc file CSV

```
# Kiểm tra schema (cấu trúc dữ liệu)
df.printSchema()
# Hiển thị vài dòng dữ liệu đầu tiên
df.show(5)
```

```
root
|-- id: integer (nullable = true)
|-- gender: string (nullable = true)
|-- age: double (nullable = true)
|-- hypertension: integer (nullable = true)
|-- heart_disease: integer (nullable = true)
|-- ever_married: string (nullable = true)
|-- work_type: string (nullable = true)
|-- Residence_type: string (nullable = true)
|-- avg_glucose_level: double (nullable = true)
|-- bmi: string (nullable = true)
|-- smoking_status: string (nullable = true)
|-- stroke: integer (nullable = true)
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	N/A	never smoked	1
31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24	never smoked	1

only showing top 5 rows

Hình 14. Kiểm tra dữ liệu

### 3.3 Kiểm tra và xử lý các dữ liệu trống (NULL)

Duyệt qua từng thuộc tính (cột) trong tập dữ liệu bằng vòng lặp *for c in df.columns*.

Đếm số lượng giá trị NULL cho mỗi cột bằng cách sử dụng hàm *count()* với các giá trị được kiểm tra bằng cách sử dụng hàm *isNull()* và sử dụng hàm *isnan()*.

```
df.select([count(when(col(c).isNull() | isnan(c), c)).alias(c) for c in df.columns]).show()
```

Hình 15. Đếm số lượng giá trị NULL

```
[ ] # Đếm số giá trị null trong mỗi cột
df.select([count(when(col(c).isNull() | isnan(c), c)).alias(c) for c in df.columns]).show()
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	0	0	0	0	0	0	0	0	0	0

Hình 16. Giá trị NULL

Từ kết quả trên ta thấy không có cột nào bị thiếu dữ liệu.

### 3.4 Kiểm tra giá trị không hợp lệ

Ta sẽ tiến hành xác định các giá trị không phải số và các giá trị không hợp lệ khác cho từng cột trừ thuộc tính “id” bằng cách duyệt qua từng thuộc tính (cột) trong tập dữ liệu bằng vòng lặp *for column in df.columns* với điều kiện *column != “id”*

```

# Xác định các giá trị không hợp lệ cho từng cột (trừ cột "id")
for column in df.columns:
    if column != "id":
        print(f"Invalid values in column '{column}':")
        # Giả sử "invalid" có nghĩa là các giá trị không phải số cho cột số và các giá trị không hợp lệ khác
        # Tùy chỉnh điều kiện kiểm tra giá trị không hợp lệ theo yêu cầu.
        invalid_values = df.filter(~col(column).rlike("^[0-9]*\.[0-9]+$")) # Kiểm tra các giá trị không phải số
        invalid_values.select(column).distinct().show()

```

Invalid values in column 'gender':

```

+-----+
|gender|
+-----+
|Female|
| Other|
|  Male|
+-----+

```

Invalid values in column 'age':

```

+----+
|age|
+----+
+----+

```

Hình 17. Giá trị không hợp lệ trong từng cột (1)

```

Invalid values in column 'avg_glucose_level':
+-----+
|avg_glucose_level|
+-----+

Invalid values in column 'bmi':
+---+
|bmi|
+---+
|N/A|
+---+

Invalid values in column 'smoking_status':
+-----+
| smoking_status|
+-----+
|      smokes|
|      Unknown|
|   never smoked|
|formerly smoked|
+-----+

Invalid values in column 'stroke':
+-----+
|stroke|
+-----+
+-----+

```

Hình 18. Giá trị không hợp lệ trong từng cột (2)

Từ kết quả trên ta thấy thuộc tính *bmi* có giá trị “N/A” và thuộc tính *smoking\_status* có giá trị “Unknown”.

Thuộc tính *bmi* có kiểu dữ liệu số nên ta sẽ xử lý giá trị “N/A” bằng cách thay giá trị này bằng giá trị trung bình và làm tròn đến 1 chữ số thập phân.

```

# Bỏ qua các giá trị "N/A" để tính mean
mean_bmi = df.filter(col("bmi") != "N/A").select(mean(col("bmi").cast("float")), alias("mean_bmi")).collect()[0]["mean_bmi"]
# Thay thế "N/A" bằng giá trị mean và làm tròn đến 1 chữ số thập phân
df = df.withColumn(
    "bmi",
    round(
        when(col("bmi") == "N/A", mean_bmi).otherwise(col("bmi").cast("float")), 1
    )
)
# Hiển thị dữ liệu sau khi thay thế
df.show(5)

```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.9	never smoked	1
31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

only showing top 5 rows

Hình 19. Thay thế các giá trị N/A

Đối với thuộc tính *smoking\_status* là dữ liệu phân loại, ta sẽ xử lý giá trị “Unknown” bằng cách thay giá trị này bằng giá trị thường gặp nhất.

```

# Tính toán giá trị xuất hiện nhiều nhất trong cột 'smoking_status'
most_common_value = df.groupBy("smoking_status").count().orderBy(F.desc("count")).first()[0]

# Thay thế giá trị 'unknown' bằng giá trị được lập lại nhiều nhất
df = df.withColumn("smoking_status", F.when(F.col("smoking_status") == "Unknown", most_common_value)
    .otherwise(F.col("smoking_status")))

# Hiển thị 5 hàng đầu tiên của DataFrame sau khi thay thế
df.show(5)

```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	28.9	never smoked	1
31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

only showing top 5 rows

Hình 20. Thay thế các giá trị Unknown

### 3.5 Chuyển đổi kiểu dữ liệu object sang integer

Ta có các thuộc tính như *gender*, *ever\_married*, *Residence\_type*, *smoking\_status*, *work\_type* có kiểu object nên ta sẽ tiến hành chuyển thuộc tính này sang kiểu integer bằng kỹ thuật **StringIndexer**.

```
root
|-- id: integer (nullable = true)
|-- gender: string (nullable = true)
|-- age: double (nullable = true)
|-- hypertension: integer (nullable = true)
|-- heart_disease: integer (nullable = true)
|-- ever_married: string (nullable = true)
|-- work_type: string (nullable = true)
|-- Residence_type: string (nullable = true)
|-- avg_glucose_level: double (nullable = true)
|-- bmi: string (nullable = true)
|-- smoking_status: string (nullable = true)
|-- stroke: integer (nullable = true)
```

Hình 21. Chuyển thuộc tính object sang integer

Ở đây ta có 5 thuộc tính cần phải chuyển đổi, ta cần tiến hành chuyển chúng về dưới dạng số nguyên bằng cách sử dụng **StringIndexer** với các bước sau:

- Đầu vào input là các thuộc tính có kiểu dữ liệu object
- Đầu ra sẽ tạo các thuộc tính có tên là *tên thuộc tính* + *\_indexed*
- Huấn luyện mô hình trên đối tượng StringIndexer sau đó áp dụng biến đổi này lên dataframe *df*. Kết quả là *df* sẽ tạo các cột mới chứa giá trị số nguyên tương ứng với mỗi chuỗi trong từng cột.
- Sau đó ta sẽ tiến hành cập nhật các cột làm đầu vào bằng giá trị số nguyên của các cột đầu ra (lúc này, giá trị các cột đầu ra có kiểu double) bằng phương thức *cast('int')* và xóa đi các cột đầu ra.
- Cuối cùng, ta sẽ tiến hành hiển thị danh sách các nhãn và chỉ số để có thể biết được là nhãn nào đã được chuyển thành giá trị 0, nhãn nào đã được chuyển thành giá trị 1, ...

```
#Các cột cần chuyển đổi
columns = ["gender", "ever_married", "Residence_type", "smoking_status", "work_type"]
#Áp dụng StringIndexer và ghi đè lên cột cũ
indexers = [
    StringIndexer(inputCol=column, outputCol=column + "_index").fit(df)
    for column in columns
]

# Sử dụng Pipeline để áp dụng
pipeline = Pipeline(stages=indexers)
df = pipeline.fit(df).transform(df)

# Ghi đè các cột cũ bằng các cột mới và ép kiểu chuyển đổi thành integer
for column in columns:
    df = df.withColumn(column, col(column + "_index").cast("int")).drop(column + "_index")
# Hiển thị 5 hàng đầu tiên của DataFrame
df.show(5)
```

Hình 22. Chuyển đổi các thuộc tính cần thiết

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	1	67.0	0	1	0	0	0	228.69	36.6	1	1
51676	0	61.0	0	0	0	1	1	202.21	28.9	0	1
31112	1	80.0	0	1	0	0	1	105.92	32.5	0	1
60182	0	49.0	0	0	0	0	0	171.23	34.4	2	1
1665	0	79.0	1	0	0	1	1	174.12	24.0	0	1

only showing top 5 rows

Hình 23. Danh sách các thuộc tính cần thiết

```
# Tạo bảng hiển thị nhãn và giá trị chuyển đổi cho từng cột sau khi áp dụng StringIndexer
for indexer, column in zip(indexers, columns):
    # Lấy danh sách nhãn từ StringIndexerModel
    labels = indexer.labels

    # Tạo DataFrame với cột Label và Index
    mapping_df = spark.createDataFrame([(label, idx) for idx, label in enumerate(labels)], ["Label", "Index"])

    # Hiển thị bảng cho cột hiện tại
    print(f"Mapping for column '{column}':")
    mapping_df.show()
```

Hình 24. Tạo bảng label sau khi áp dụng StringIndexer



```

Mapping for column 'gender':
+-----+-----+
| Label|Index|
+-----+-----+
| Female|    0|
|  Male|    1|
|  Other|    2|
+-----+-----+

Mapping for column 'ever_married':
+-----+-----+
|Label|Index|
+-----+-----+
|  Yes|    0|
|  No|    1|
+-----+-----+

Mapping for column 'Residence_type':
+-----+-----+
|Label|Index|
+-----+-----+
|Urban|    0|
|Rural|    1|
+-----+-----+

```

Hình 25. Danh sách label sau chuyển đổi (1)

```

Mapping for column 'smoking_status':
+-----+-----+
|          Label|Index|
+-----+-----+
| never smoked|    0|
|formerly smoked|    1|
|          smokes|    2|
+-----+-----+

Mapping for column 'work_type':
+-----+-----+
|          Label|Index|
+-----+-----+
|      Private|    0|
|Self-employed|    1|
|   children|    2|
|   Govt_job|    3|
| Never_worked|    4|
+-----+-----+

```

Hình 26. Danh sách label sau chuyển đổi (2)

Từ hiển thị trên, ta có thể thấy được các nhãn được mã hóa thành những giá trị nào và cũng biết được tần suất xuất hiện của từng nhãn.

### 3.6 Chuẩn hóa dữ liệu

Chuẩn hóa dữ liệu là quá trình quan trọng trong khoa học dữ liệu và học máy, nhằm đưa dữ liệu về một dạng tiêu chuẩn. Điều này giúp cải thiện hiệu suất và độ chính xác của

các mô hình học máy bằng cách loại bỏ sự thiên vị đối với các thuộc tính có phạm vi giá trị lớn hơn.

Chuẩn hóa dữ liệu làm dữ liệu bị thay đổi về mặt ý nghĩa, nhưng không thay đổi tính chất của dữ liệu. Ở đây, nhóm sử dụng Min-Max Scaling để chuẩn hóa dữ liệu về các giá trị từ 0 đến 1.

Để chuẩn hóa dữ liệu, ta cần thực hiện theo các bước sau:

- Tạo danh sách chứa các cột cần chuẩn hóa
- Duyệt từng cột trong các cột cần chuẩn hóa bằng vòng lặp *for*, sau đó tìm ra các giá trị lớn nhất và nhỏ nhất của từng cột bằng phương thức *agg*.
- Sau đó, áp dụng **Min-Max Scaling** để chuẩn hóa dữ liệu của các cột trên theo công thức:

$$X_{\text{new}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

- Sử dụng *df.show()* để hiển thị kết quả sau khi chuẩn hóa dữ liệu.

```
# Chọn các cột số cần chuẩn hóa
numeric_columns = [col for col, dtype in df.dtypes if dtype in ['int', 'double'] and col != 'id']

# Thực hiện Min-Max Scaling cho mỗi cột
for col_name in numeric_columns:
    # Tính min và max cho cột
    min_col = df.agg({col_name: "min"}).collect()[0][0]
    max_col = df.agg({col_name: "max"}).collect()[0][0]

    # Áp dụng Min-Max Scaling
    df = df.withColumn(col_name, (col(col_name) - min_col) / (max_col - min_col))

# Hiển thị dữ liệu sau khi chuẩn hóa
df.show(5)
```

Hình 27. Chuẩn hóa dữ liệu

```
# Chọn các cột số cần chuẩn hóa
numeric_columns = [col for col, dtype in df.dtypes if dtype in ['int', 'double'] and col != 'id']

# Thực hiện Min-Max Scaling cho mỗi cột
for col_name in numeric_columns:
    # Tính min và max cho cột
    min_col = df.agg({col_name: "min"}).collect()[0][0]
    max_col = df.agg({col_name: "max"}).collect()[0][0]

    # Áp dụng Min-Max Scaling
    df = df.withColumn(col_name, (col(col_name) - min_col) / (max_col - min_col))

# Hiển thị dữ liệu sau khi chuẩn hóa
df.show(5)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	0.5	0.81689453125	0.0	1.0	0.0	0.0	0.0	0.8012648878219923	0.3012600229095075	0.5	1.0	
51676	0.0	0.74365234375	0.0	0.0	0.0	0.25	1.0	0.6790231742221402	0.21305841924398625	0.0	1.0	
31112	0.5	0.9755859375	0.0	1.0	0.0	0.0	1.0	0.23451204874896134	0.2542955326460481	0.0	1.0	
60182	0.0	0.59716796875	0.0	0.0	0.0	0.0	0.0	0.5360077555165728	0.2760595647193585	1.0	1.0	
1665	0.0	0.96337890625	1.0	0.0	0.0	0.25	1.0	0.5493490905733542	0.15693012600229095	0.0	1.0	

only showing top 5 rows

Hình 28. Kết quả sau khi chuẩn hóa

### 3.7 Kiểm tra phân phối các lớp trong cột stroke

Việc kiểm tra phân phối trong cột phân loại là một bước quan trọng trong quá trình chuẩn bị dữ liệu, giúp ta hiểu rõ hơn về cấu trúc dữ liệu và đưa ra quyết định hợp lý về cách xử lý hoặc mã hóa các giá trị này.

Để kiểm tra phân phối trong một cột stroke, ta sử dụng hàm `groupBy()` và `count()` để tính toán số lượng các giá trị duy nhất trong cột đó.

```
[ ] # Kiểm tra phân phối các lớp trong cột 'stroke'
df.groupBy("stroke").count().show()
```

stroke	count
0.0	4861
1.0	249

Hình 29. Kiểm tra phân phối cột stroke

### 3.8 Xuất file sau khi tiền xử lý dữ liệu

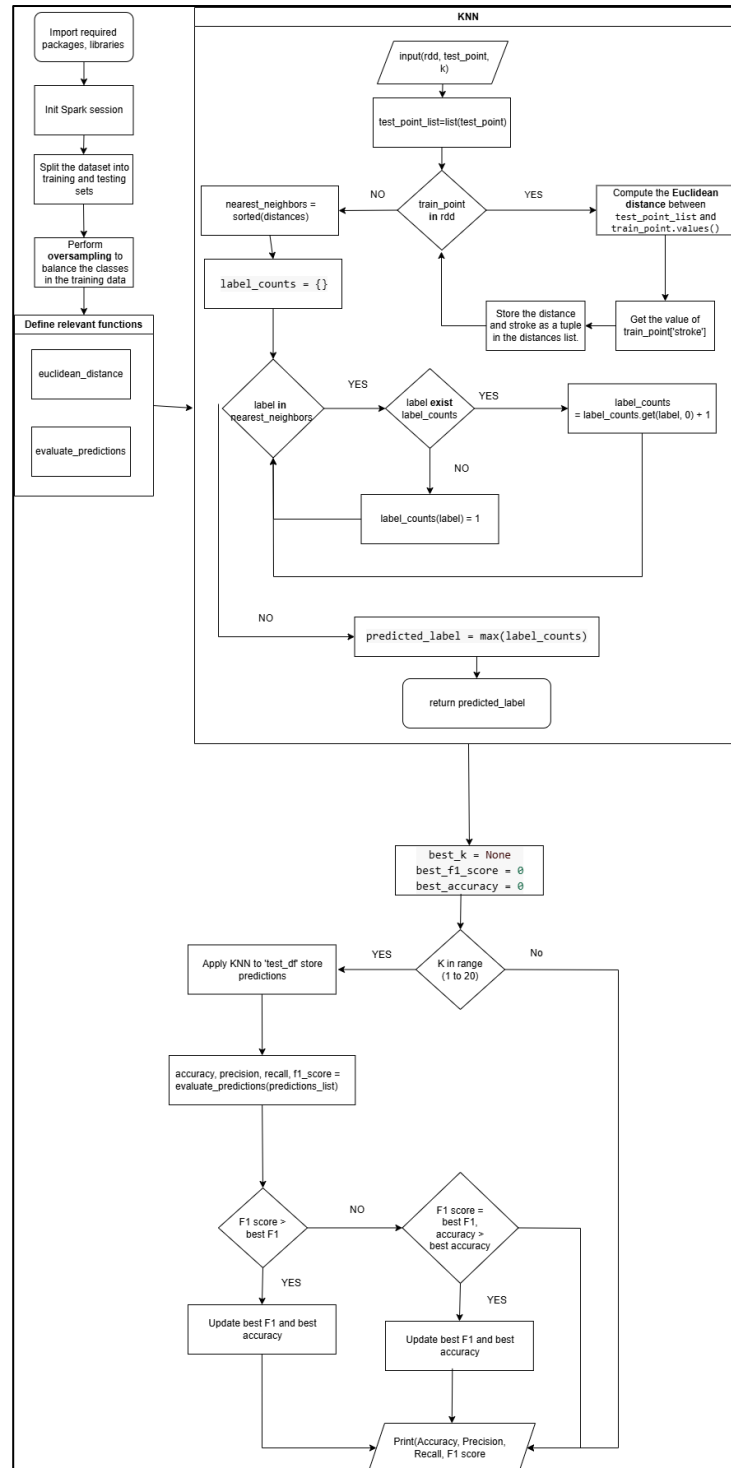
Sau khi đã hoàn tất việc tiền xử lý dữ liệu, ta xuất dữ liệu ra thành 1 file mới để thuận tiện cho việc tiến hành train model bằng hàm `write.csv()`

```
#Xuất file bằng lệnh write.csv
output_path= "processed_stroke_dataset.csv"
df.write.csv(output_path, mode="overwrite", header=True)
```

Hình 30. Xuất file CSV

# CHƯƠNG 4: TRIỂN KHAI THUẬT TOÁN

## 4.1 Các bước thực hiện thuật toán KNN



Hình 31. Flowchart của KNN

Lưu đồ thuật toán KNN:

- Tạo hàm KNN với tham số `rdd`, `test_point`, `k`

- Chuyển đổi *test\_point* thành list (*test\_point\_list*).
- Chạy vòng lặp:
  - Duyệt qua từng *train\_point* trong *rdd*.
  - Nếu tồn tại *train\_point* trong *rdd* thì tính toán khoảng cách bằng hàm *euclidean\_distance* giữa *test\_point* và *train\_point.values()*.
  - Sau đó lấy giá trị *train\_point["stroke"]* tương ứng lưu trữ khoảng cách và hành trình dưới dạng một bộ dữ liệu trong danh sách khoảng cách.
- Khi vòng lặp kết thúc thì thực hiện sắp xếp các mảng các distance từ thấp đến cao và gán cho *nearest\_neighbors*.
- Khởi tạo mảng *label\_counts* để lưu trữ các nhãn và đếm số lượng của nhãn.
- Chạy vòng lặp:
  - Duyệt qua từng label trong *nearest\_neighbors*.
  - Kiểm tra nếu label có trong *label\_counts* hay không, nếu có thì tăng giá trị của thuộc tính ghi nhận số lượng trong *label\_counts* tăng lên 1. Còn nếu không có thì thêm label vào *label\_counts* và khởi tạo giá trị là 1.
- Khi vòng lặp kết thúc lấy nhãn xuất hiện nhiều nhất trong *label\_counts* và gán cho biến *predicted\_label* và return *predicted\_label* cho hàm *knn*
- Khởi tạo biến *best\_k=None*, *best\_f1\_score=0*, *best\_accuracy=0*
- Chạy vòng lặp:
  - Duyệt qua từng k từ 1 đến 20
  - Ta *test\_df* chuyển đổi từ Dataframe qua RDD sau đó áp dụng thuật toán KNN để lấy ra nhãn nhiều nhất từ hàm *knn* và gán cho *predictions*.
  - Chuyển đổi *predictions* thành *predictions\_list*
  - Kiểm tra *F1 score* có lớn hơn *best F1* không, nếu đúng thì sẽ cập nhật lại *best F1* và *best accuracy*
  - Nếu điều kiện trên không được sẽ kiểm tra điều kiện còn lại: Nếu *F1 score* bằng *best F1*, sẽ so sánh *accuracy* lớn hơn *best accuracy*
  - In ra Accuracy, Precision, Recall, F1 score

## 4.2 Chạy thuật toán

### Thuật toán KNN:

- Khai báo các thư viện cần thiết

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, mean, stddev
3 import numpy as np
4 import random
5 from pyspark.sql import functions as F
6 import os
7 from math import sqrt
8 from google.colab import drive
```

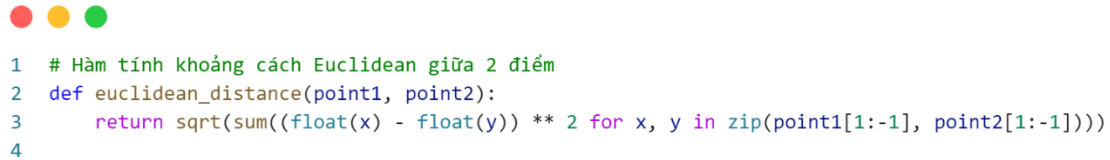
Hình 32. Khai báo thư viện

- Đoạn mã thực hiện việc **chia dữ liệu** thành hai tập huấn luyện và kiểm tra, **cân bằng lại dữ liệu** trong tập huấn luyện bằng phương pháp **oversampling**, và cuối cùng in ra các kết quả phân phối dữ liệu và số lượng mẫu.

```
1 # Chia dữ liệu thành tập huấn luyện và kiểm tra (80% huấn luyện, 20% kiểm tra)
2 train_df, test_df = df.randomSplit([0.8, 0.2], seed=0)
3
4 # Oversampling lớp thiểu số
5 minority_class = train_df.filter(col("stroke") == 1.0)
6 majority_class = train_df.filter(col("stroke") == 0.0)
7
8 # Sao chép các mẫu từ lớp thiểu số
9 oversampling_fraction = 3.0
10 oversampled_minority = minority_class.sample(withReplacement=True, fraction=oversampling_fraction)
11 balanced_train_df = majority_class.union(oversampled_minority)
12
13 # Kiểm tra lại phân phối
14 balanced_train_df.groupBy("stroke").count().show()
15
16 train_df.groupBy('stroke').count().show()
17 test_df.groupBy('stroke').count().show()
18
19 # Đếm số lượng mẫu trong tập huấn luyện và kiểm tra
20 train_count = train_df.count()
21 test_count = test_df.count()
22
23 # In kết quả số lượng mẫu
24 print(f"Số lượng dữ liệu huấn luyện: {train_count}")
25 print(f"Số lượng dữ liệu kiểm tra: {test_count}")
```

Hình 33. Chia, cân bằng dữ liệu

- Hàm **euclidean\_distance** tính khoảng cách Euclidean giữa hai điểm trong không gian. Đây là một công thức toán học dùng để đo lường độ dài (khoảng cách) giữa hai điểm trong không gian nhiều chiều.



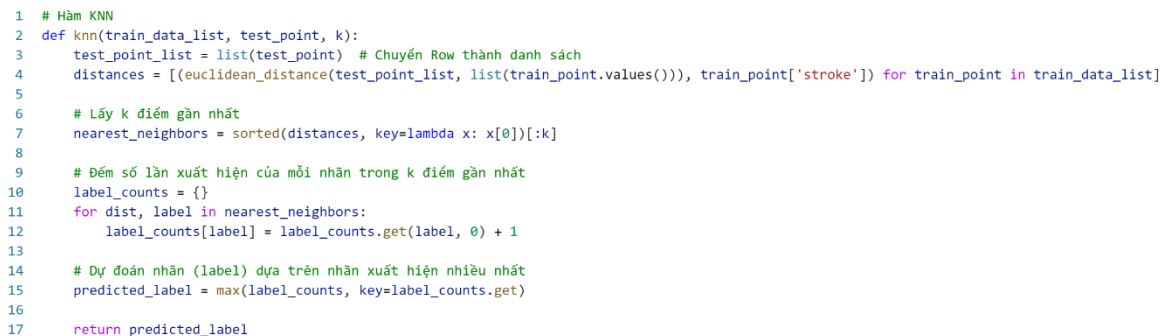
```

1 # Hàm tính khoảng cách Euclidean giữa 2 điểm
2 def euclidean_distance(point1, point2):
3     return sqrt(sum((float(x) - float(y)) ** 2 for x, y in zip(point1[1:-1], point2[1:-1])))
4

```

Hình 34. Hàm tính Euclidean

- Hàm KNN thực hiện thuật toán K-Nearest Neighbors (K-NN) để dự đoán nhãn (**label**) của một điểm dữ liệu kiểm tra (**test\_point**) dựa trên tập dữ liệu huấn luyện (**train\_data\_list**) và giá trị K.



```

1 # Hàm KNN
2 def knn(train_data_list, test_point, k):
3     test_point_list = list(test_point) # Chuyển Row thành danh sách
4     distances = [(euclidean_distance(test_point_list, list(train_point.values()))), train_point['stroke']] for train_point in train_data_list
5
6     # Lấy k điểm gần nhất
7     nearest_neighbors = sorted(distances, key=lambda x: x[0])[:k]
8
9     # Đếm số lần xuất hiện của mỗi nhãn trong k điểm gần nhất
10    label_counts = {}
11    for dist, label in nearest_neighbors:
12        label_counts[label] = label_counts.get(label, 0) + 1
13
14    # Dự đoán nhãn (label) dựa trên nhãn xuất hiện nhiều nhất
15    predicted_label = max(label_counts, key=label_counts.get)
16
17    return predicted_label

```

Hình 35. Thực hiện KNN

- Hàm **evaluate\_predictions** dùng để tính các chỉ số đánh giá hiệu quả của một mô hình phân loại nhị phân dựa trên danh sách các dự đoán.

```

1 def evaluate_predictions(predictions_list):
2     true_positives = sum(1 for actual, predicted in predictions_list if actual == 1.0 and predicted == 1.0)
3     true_negatives = sum(1 for actual, predicted in predictions_list if actual == 0.0 and predicted == 0.0)
4     false_positives = sum(1 for actual, predicted in predictions_list if actual == 0.0 and predicted == 1.0)
5     false_negatives = sum(1 for actual, predicted in predictions_list if actual == 1.0 and predicted == 0.0)
6
7     accuracy = (true_positives + true_negatives) / len(predictions_list)
8     precision = true_positives / (true_positives + false_positives) if (true_positives + false_positives) > 0 else 0
9     recall = true_positives / (true_positives + false_negatives) if (true_positives + false_negatives) > 0 else 0
10    f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
11
12    return accuracy, precision, recall, f1_score

```

Hình 36. Hàm Evaluate Predictions

- Đoạn mã thực hiện việc tìm giá trị  $k$  tốt nhất cho thuật toán K-Nearest Neighbors (K-NN) bằng cách đánh giá hiệu quả trên tập kiểm tra.

```

1 best_k = None
2 best_f1_score = 0
3 best_accuracy = 0
4 train_data_list = balanced_train_df.rdd.map(lambda row: row.asDict()).cache().collect()
5 train_data_broadcast = spark.sparkContext.broadcast(train_data_list)
6 accuracies = []
7 f1_scores = []
8 k_values = range(1, 7)
9
10 for k in range(1, 21):
11     predictions = test_df.rdd.map(lambda test_point: (test_point[-1], knn(train_data_list, test_point, k)))
12     predictions_list = predictions.collect()
13     accuracy, precision, recall, f1_score = evaluate_predictions(predictions_list)
14
15     accuracies.append(accuracy)
16     f1_scores.append(f1_score)
17
18     if f1_score > best_f1_score or (f1_score == best_f1_score and accuracy > best_accuracy):
19         best_accuracy = accuracy
20         best_f1_score = f1_score
21         best_k = k
22
23     print(f"K = {k}, Accuracy = {accuracy:.2f}, Precision = {precision:.2f}, Recall = {recall:.2f}, F1 Score = {f1_score:.2f}")

```

Hình 37. Tìm giá trị  $k$ 

- Đoạn mã tính toán các chỉ số đánh giá hiệu quả của một mô hình phân loại nhị phân bằng cách xác định các trường hợp đúng/sai của dự đoán và sau đó sử dụng chúng để tính các chỉ số như **accuracy**, **precision**, **recall**, và **F1 score**.



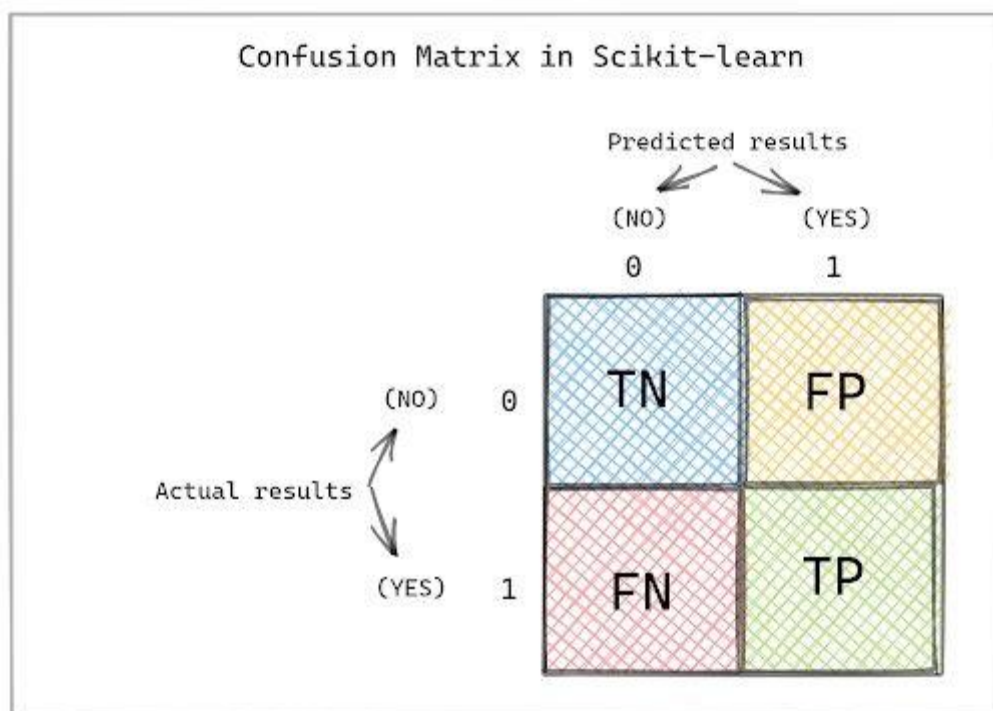
```
1 # Tính toán true positives, true negatives, false positives, false negatives
2 true_positives = 0
3 true_negatives = 0
4 false_positives = 0
5 false_negatives = 0
6
7 # Tính toán các giá trị trên driver
8 for prediction in predictions_list:
9     actual = prediction[0] # Thực tế (label)
10    predicted = prediction[1] # Dự đoán
11
12    if actual == 1.0 and predicted == 1.0:
13        true_positives += 1
14    elif actual == 0.0 and predicted == 0.0:
15        true_negatives += 1
16    elif actual == 0.0 and predicted == 1.0:
17        false_positives += 1
18    elif actual == 1.0 and predicted == 0.0:
19        false_negatives += 1
20 print(f"True Positives: {true_positives}")
21 print(f"True Negatives: {true_negatives}")
22 print(f"False Positives: {false_positives}")
23 print(f"False Negatives: {false_negatives}")
24
25 # Tính các chỉ số đánh giá
26 accuracy = (true_positives + true_negatives) / len(predictions_list)
27 precision = true_positives / (true_positives + false_positives) if (true_positives + false_positives) > 0 else 0
28 recall = true_positives / (true_positives + false_negatives) if (true_positives + false_negatives) > 0 else 0
29 f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
30
31 # In các chỉ số đánh giá
32 print(f"Accuracy: {accuracy:.2f}")
33 print(f"Precision: {precision:.2f}")
34 print(f"Recall: {recall:.2f}")
35 print(f"F1 Score: {f1_score:.2f}")
```

Hình 38. Các chỉ số đánh giá hiệu quả mô hình

## CHƯƠNG 5: KẾT QUẢ ĐẠT ĐƯỢC

### 5.1 Ý nghĩa các độ đo và lý do lựa chọn

**Confusion Matrix (CM):**



Hình 39. Confusion Matrix (CM)

**Confusion Matrix**[5] là công cụ trực quan và hữu ích để tính toán các độ đo như **Accuracy**, **Precision**, **Recall**, và **F1 Score**. Các giá trị TP, TN, FP, FN được định nghĩa như sau:

- True Positives (TP): Số mẫu thuộc lớp dương tính được dự đoán đúng.
- True Negatives (TN): Số mẫu thuộc lớp âm tính được dự đoán đúng.
- False Positives (FP): Số mẫu thuộc lớp âm tính bị dự đoán nhầm thành lớp dương tính.
- False Negatives (FN): Số mẫu thuộc lớp dương tính bị dự đoán nhầm thành lớp âm tính.

### Độ chính xác (Accuracy)

- Công thức:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Ý nghĩa:** Accuracy đo lường tỷ lệ các dự đoán đúng (gồm cả dương tính và âm tính) trên tổng số các dự đoán. Nó giúp xác định mức độ chính xác chung của mô hình.

- **Lý do chọn:** Accuracy dễ hiểu và phù hợp khi dữ liệu cân bằng giữa các lớp. Tuy nhiên, trong dữ liệu không cân bằng, Accuracy có thể gây hiểu nhầm nếu mô hình chỉ tập trung vào lớp chiếm đa số.

### Độ chính xác theo lớp dương (Precision)[6]

- **Công thức:**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Ý nghĩa:** Precision cho biết trong số các dự đoán dương tính, bao nhiêu là chính xác.
- **Lý do chọn:** Precision hữu ích trong các bài toán mà chi phí của các dự đoán sai dương tính (False Positives) cao, chẳng hạn trong phát hiện gian lận hoặc chẩn đoán bệnh không tồn tại.

### Độ phủ (Recall)

- **Công thức:**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **Ý nghĩa:** Recall đo lường khả năng phát hiện các trường hợp dương tính thực sự trong tổng số dương tính thực sự.
- **Lý do chọn:** Recall được ưu tiên trong các bài toán mà việc bỏ sót các trường hợp dương tính (False Negatives) là nghiêm trọng, như trong chẩn đoán bệnh ung thư hoặc phát hiện bất thường.

### F1 Score[7]

- **Công thức:**

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Ý nghĩa:** F1 Score là trung bình điều hòa giữa Precision và Recall, nhấn mạnh vào việc cân bằng hai chỉ số này. Giá trị F1 cao cho thấy mô hình đạt được hiệu suất tốt ở cả Precision và Recall.

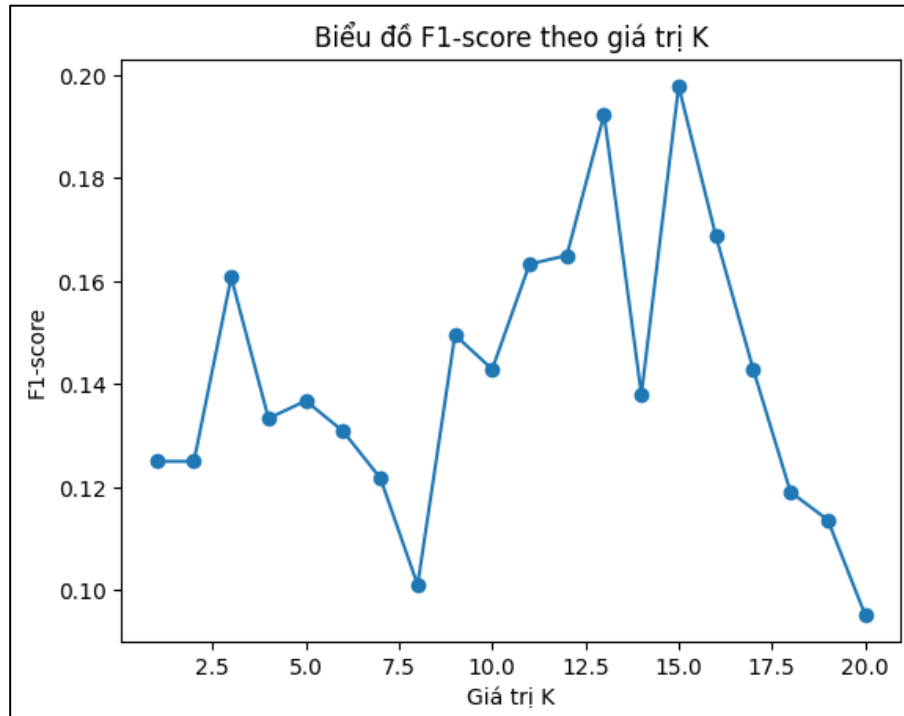
- **Lý do chọn:** F1 phù hợp khi bài toán yêu cầu cân bằng giữa Precision và Recall, đặc biệt khi các lớp không cân bằng.

## 5.2 Phát biểu kết quả

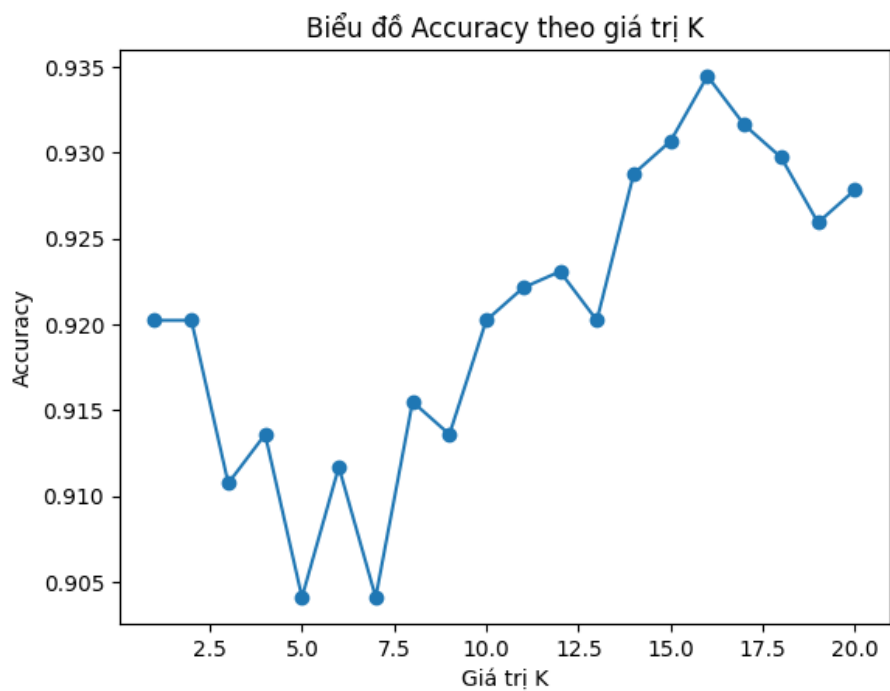
Các giá trị độ đo sau khi test giá trị K từ 1 đến 20:

K = 1, Accuracy = 0.92, Precision = 0.13, Recall = 0.11, F1 Score = 0.12
K = 2, Accuracy = 0.92, Precision = 0.13, Recall = 0.11, F1 Score = 0.12
K = 3, Accuracy = 0.91, Precision = 0.18, Recall = 0.18, F1 Score = 0.18
K = 4, Accuracy = 0.92, Precision = 0.19, Recall = 0.18, F1 Score = 0.19
K = 5, Accuracy = 0.92, Precision = 0.20, Recall = 0.20, F1 Score = 0.20
K = 6, Accuracy = 0.92, Precision = 0.22, Recall = 0.20, F1 Score = 0.21
K = 7, Accuracy = 0.92, Precision = 0.21, Recall = 0.22, F1 Score = 0.21
K = 8, Accuracy = 0.93, Precision = 0.23, Recall = 0.18, F1 Score = 0.20
K = 9, Accuracy = 0.92, Precision = 0.22, Recall = 0.20, F1 Score = 0.21
K = 10, Accuracy = 0.93, Precision = 0.24, Recall = 0.16, F1 Score = 0.20
K = 11, Accuracy = 0.93, Precision = 0.22, Recall = 0.16, F1 Score = 0.19
K = 12, Accuracy = 0.93, Precision = 0.24, Recall = 0.16, F1 Score = 0.19
K = 13, Accuracy = 0.92, Precision = 0.16, Recall = 0.13, F1 Score = 0.14
K = 14, Accuracy = 0.92, Precision = 0.07, Recall = 0.04, F1 Score = 0.05
K = 15, Accuracy = 0.93, Precision = 0.13, Recall = 0.07, F1 Score = 0.09
K = 16, Accuracy = 0.93, Precision = 0.14, Recall = 0.07, F1 Score = 0.10
K = 17, Accuracy = 0.93, Precision = 0.16, Recall = 0.07, F1 Score = 0.10
K = 18, Accuracy = 0.93, Precision = 0.13, Recall = 0.05, F1 Score = 0.08
K = 19, Accuracy = 0.93, Precision = 0.17, Recall = 0.09, F1 Score = 0.12
K = 20, Accuracy = 0.93, Precision = 0.19, Recall = 0.09, F1 Score = 0.12

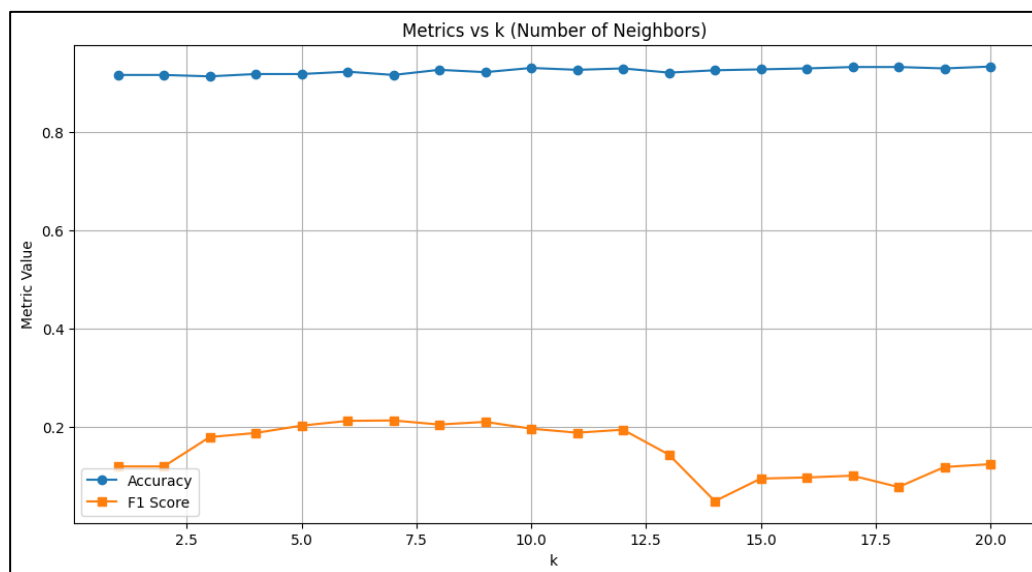
Hình 40. Giá trị đo khi k từ 1 đến 20



Hình 41. Biểu đồ F1-Score



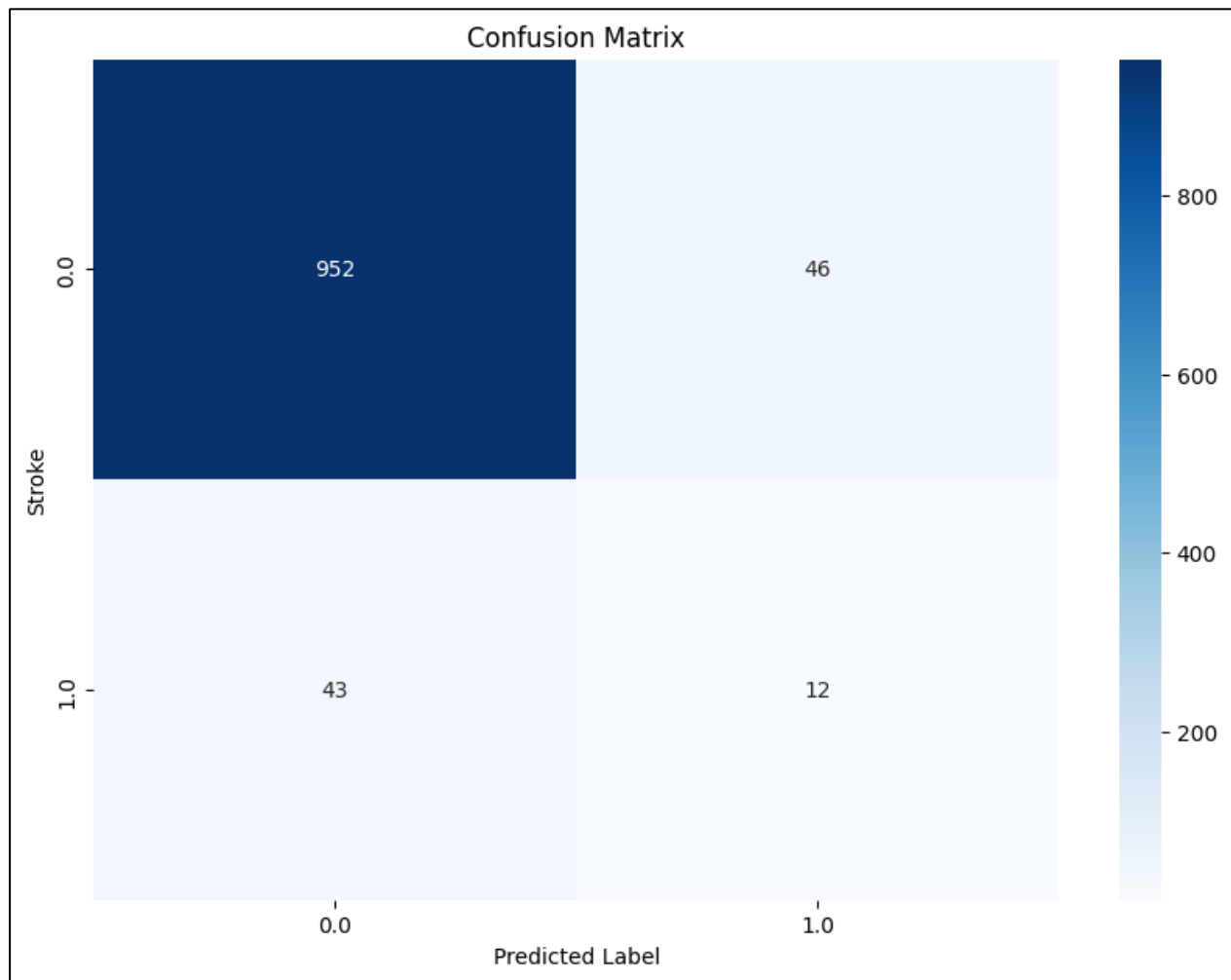
Hình 42. Biểu đồ Accuracy



Hình 43. Biểu đồ so sánh 2 giá trị

**Kết luận:** Giá trị K tốt nhất: 7, Độ chính xác tốt nhất: 0.92, F1-Score: 0.21

**Confusion Matrix:**



Hình 44. Biểu đồ Confusion Matrix

**Kết luận:** Mô hình có số lượng True Negatives (TN = 952) cao, cho thấy nhận diện chính xác các trường hợp **không đột quỵ** khá tốt. Tuy nhiên, False Positives (FP = 46) và False Negatives (FN = 43) vẫn cần chú ý. Số lượng False Positives (46) cho thấy mô hình dự đoán sai đột quỵ ở một số trường hợp **không có đột quỵ**, có thể dẫn đến các cảnh báo sai. Mặc dù True Positives (12) là khá thấp, nhưng số lượng False Negatives (43) cho thấy mô hình không phát hiện được một số trường hợp đột quỵ thực tế, điều này có thể nguy hiểm trong bối cảnh y tế, vì bệnh nhân có thể không nhận được sự chăm sóc kịp thời. Nhìn chung, mô hình có hiệu suất chưa cao trong việc phát hiện các trường hợp đột quỵ, đặc biệt là với lớp thiểu số (stroke = 1)

### Các độ đo Accuracy, Precision, Recall, F1-Score:

Label	Accuracy	Precision	Recall	F1_Score
0.0	0.9154795821462488	0.9539078156312625	0.9567839195979899	0.9553437029603612
1.0	0.9154795821462488	0.218181818181817	0.20689655172413793	0.21238938053097345

Hình 45. Các độ đo

**Kết luận:** Mô hình có độ chính xác cao (Accuracy = 91.55%), nhưng sự phân hóa giữa các lớp rất rõ rệt. Đối với lớp 0.0 (không đột quy), mô hình đạt Precision = 95.39%, Recall = 95.68%, và F1-Score = 95.53%, cho thấy nó rất tốt trong việc nhận diện các trường hợp không đột quy. Tuy nhiên, đối với lớp 1.0 (đột quy), các chỉ số như Precision = 21.82%, Recall = 20.69%, và F1-Score = 21.24% cho thấy mô hình rất yếu trong việc phát hiện đột quy. Tập dataset bị lệch nghiêm trọng về lớp đa số (không đột quy), và vì vậy không nhận diện tốt lớp thiểu số (đột quy).

## 5.3 So sánh, đánh giá

### Đánh giá về Dataset:

Bộ dữ liệu này có sự mất cân bằng rõ rệt giữa hai lớp: lớp **0.0 (không đột quy)** chiếm ưu thế, trong khi lớp **1.0 (đột quy)** là thiểu số. Điều này có thể dẫn đến việc mô hình thiên về việc dự đoán lớp đa số, gây ra các vấn đề trong việc nhận diện lớp thiểu số. Mặc dù độ chính xác chung của mô hình cao (91.55%), nhưng các chỉ số khác cho thấy mô hình chưa làm tốt trong việc phát hiện đột quy, vì lớp thiểu số có các giá trị rất thấp về precision, recall và F1-score.

### Đánh giá về Thuật toán (KNN):

Kết quả từ thuật toán KNN cho thấy một số điểm mạnh và hạn chế rõ ràng:

- **Độ chính xác (Accuracy)** của mô hình khá cao (**91.55%**), nhưng độ chính xác này chủ yếu là nhờ vào lớp đa số (0.0) do sự mất cân bằng lớp.
- Đối với lớp **0.0**, các chỉ số **precision, recall và F1-score** đều rất tốt (**> 95%**), cho thấy mô hình có khả năng phân loại tốt các trường hợp không đột quy.
- Tuy nhiên, đối với lớp **1.0**, các chỉ số này đều rất thấp, với **precision** chỉ đạt **21.82%**, **recall 20.69%**, và **F1-score 21.24%**. Điều này cho thấy mô hình gặp khó khăn trong việc phát hiện các trường hợp đột quy, dẫn đến nhiều **false**

**negatives** (bệnh nhân bị đột quỵ nhưng bị dự đoán là không có đột quỵ) và **false positives** (bệnh nhân không có đột quỵ nhưng bị dự đoán có đột quỵ).

### Kết luận:

Mô hình KNN hoạt động khá tốt với lớp đa số (không đột quỵ), nhưng lại gặp khó khăn khi xử lý lớp thiểu số (đột quỵ). Điều này chủ yếu xuất phát từ sự mất cân bằng trong bộ dữ liệu, khiến mô hình thiên về dự đoán lớp đa số, dẫn đến hiệu suất kém đối với lớp thiểu số. Mặc dù đã áp dụng một số cải tiến như oversampling và trọng số theo khoảng cách, nhưng kết quả vẫn chưa đạt được như kỳ vọng, đặc biệt ở các chỉ số đánh giá liên quan đến lớp thiểu số.

Với bộ dữ liệu hiện tại, sự mất cân bằng giữa hai lớp (đa số là không đột quỵ, thiểu số là đột quỵ) đã ảnh hưởng nghiêm trọng đến khả năng nhận diện lớp thiểu số của mô hình. Dù đã tìm được các tham số tối ưu, hiệu suất mô hình vẫn không cải thiện đáng kể. Điều này chỉ ra rằng thuật toán KNN có một số hạn chế khi xử lý bộ dữ liệu mất cân bằng và không thực sự phù hợp cho các bài toán có sự chênh lệch giữa các lớp như thế này.

## CHƯƠNG 6: KẾT LUẬN

### 6.1 Ưu điểm

Đề tài đã thành công trong việc nhận diện và giải quyết bài toán khai thác dữ liệu lớn, đặc biệt là với Apache Spark, nền tảng mạnh mẽ hỗ trợ xử lý dữ liệu phân tán.

Việc áp dụng thuật toán KNN trên nền tảng phân tán thể hiện sự kết hợp giữa một thuật toán học máy đơn giản nhưng hiệu quả với công nghệ hiện đại, mang lại tiềm năng lớn trong việc xử lý dữ liệu lớn.

Mặc dù KNN không phải thuật toán lý tưởng cho dữ liệu lớn, nhưng Spark đã giúp giảm đáng kể thời gian xử lý, đồng thời tận dụng khả năng phân tán của hệ thống.

Nhóm cũng đã thử nghiệm các kỹ thuật như oversampling và tối ưu hóa tham số  $k$  để cải thiện hiệu suất của mô hình, nhằm giảm thiểu sự thiên lệch giữa các lớp.



## 6.2 Hạn chế

Bộ dữ liệu sử dụng trong nghiên cứu có sự mất cân bằng nghiêm trọng giữa các lớp, dẫn đến mô hình có xu hướng dự đoán thiên về lớp đa số, ảnh hưởng đến độ chính xác trong việc nhận diện lớp thiểu số.

Mặc dù KNN là một thuật toán đơn giản và dễ áp dụng, nhưng trong trường hợp này, hiệu quả của nó bị hạn chế bởi dữ liệu không cân bằng và sự thiếu tập trung vào phát hiện lớp thiểu số như các trường hợp đột quy. KNN không phải là thuật toán tối ưu khi đối mặt với dữ liệu lớn, đặc biệt khi dữ liệu chứa nhiều hoặc có sự phân phối không đều, làm giảm chất lượng dự đoán.

Thuật toán KNN yêu cầu lưu trữ toàn bộ dữ liệu huấn luyện trong bộ nhớ, điều này tạo ra áp lực lớn lên tài nguyên hệ thống, khiến mô hình khó triển khai với quy mô lớn.

Dù sử dụng Apache Spark để xử lý dữ liệu nhanh chóng, nhưng vẫn chưa khai thác hết tiềm năng của nền tảng này khi áp dụng cho thuật toán KNN, đặc biệt trong việc tối ưu hóa tốc độ xử lý và giảm thiểu chi phí tính toán.

## 6.3 Hướng phát triển

Nhóm có thể chọn dataset khác có sự cân bằng giữa các lớp và làm giảm sự thiên lệch của mô hình.

Cần chọn lọc các đặc trưng quan trọng sẽ giúp cải thiện hiệu suất mô hình, đồng thời giúp giảm thiểu chi phí tính toán và tăng tốc độ xử lý.

Có thể thử nghiệm và áp dụng các thuật toán khác như K-Mean, Gradient Boosting hay các thuật toán học sâu (deep learning) sẽ giúp cải thiện độ chính xác và khả năng tổng quát của mô hình.

Khả năng triển khai các mô hình học máy theo thời gian thực trên nền tảng phân tán như Apache Spark cũng là một hướng phát triển quan trọng, giúp mô hình có thể được sử dụng trong các ứng dụng thực tiễn yêu cầu xử lý dữ liệu liên tục.

Kết quả đề tài có thể được mở rộng để phát triển các hệ thống phân tích dữ liệu lớn trong nhiều lĩnh vực khác, từ đó nâng cao tính ứng dụng và khả năng giải quyết các vấn đề thực tế của mô hình.

## PHÂN CÔNG CÔNG VIỆC

	Thanh Bình	Thảo Ngân	Quang Thành	Anh Tuấn
Lựa chọn, thống nhất đề tài	x	x	x	x
Lý do chọn đề tài	x	x	x	x
Giới thiệu dataset				x
Mô tả dữ liệu				x
Mô tả bài toán				x
Kỹ thuật tiền xử lý dữ liệu		x		
Thuật toán khai thác dữ liệu			x	
Tiền xử lý dữ liệu		x		x
Triển khai thuật toán K-Nearest Neighbors	x		x	
Trực quan hóa dữ liệu				x
Lý do lựa chọn độ đo	x	x		
Phát biểu kết quả	x	x		
Đánh giá	x	x		
Kết luận			x	
Bản tóm tắt nội dung đề tài	x		x	x
Làm báo cáo				x
Làm Slide	x	x		
Chỉnh sửa format các file báo cáo				x

Bảng 3. Phân công công việc

## TÀI LIỆU THAM KHẢO

- [1] “The top 10 causes of death.” Accessed: Nov. 25, 2024. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>
- [2] Trí D., “Thấy gì từ tỷ lệ ca bệnh đột quỵ của người Việt?,” Báo điện tử Dân Trí. Accessed: Nov. 25, 2024. [Online]. Available: <https://dantri.com.vn/tam-diem/thay-gi-tu-ty-le-ca-benh-dot-quy-cua-nguoi-viet-20240501192817129.htm>
- [3] “Spark Machine Learning MLlib: StringIndexer | by Big Data Landscape | Medium.” Accessed: Dec. 07, 2024. [Online]. Available: [https://medium.com/@big\\_data\\_landscape/spark-machine-learning-mllib-stringindexer-b91e3f6b8905](https://medium.com/@big_data_landscape/spark-machine-learning-mllib-stringindexer-b91e3f6b8905)
- [4] “PySpark StringIndexer - A Comprehensive Guide to master PySpark StringIndexer - Machine Learning Plus.” Accessed: Dec. 07, 2024. [Online]. Available: <https://www.machinelearningplus.com/pyspark/pyspark-stringindexer/>
- [5] “Tìm hiểu về Confusion matrix trong Machine Learning? - Viblo.” Accessed: Dec. 14, 2024. [Online]. Available: <https://viblo.asia/p/tim-hieu-ve-confusion-matrix-trong-machine-learning-Az45bRpo5xY>