ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Khoa Khoa học và Kỹ thuật máy tính

Học máy (CO3094)

CHỦ ĐỀ

Huấn luyện mô hình dự đoán bệnh tiểu đường

Giảng viên: Vương Bá Thịnh
Group: Nhóm 2 – CN01
Student: Nguyễn Minh Hiếu – 2153343 (Nhóm trưởng)
Bùi Hoàng Quang Huy – 2153372
Lê Hoàng Phúc – 2153685
Hồng Anh Quân – 2152916

Thành phố Hồ Chí Minh, Tháng 5 năm 2024

Table of contents

Danh sách thành viên & Phân công nhiệm vụ

STT	Họ và tên	ID	Công việc	Đóng góp
1	Nguyễn Minh Hiếu	2153343	Cơ sở lý thuyết, xử lý ngoại lai và dữ liệu mất cân bằng, so sánh hiệu quả, mô hình Naive Bayes, KNN, Ensemble	25%
2	Bùi Hoàng Quang Huy	2153372	Cơ sở lý thuyết, tổng quan dữ liệu và tiền xử lý, mô hình SVM, Decision Tree, kết luận	25%
3	Lê Hoàng Phúc	2152239	Cơ sở lý thuyết, mô hình Logistic Regression, ANN, làm slide thuyết trình	25%
4	Hồng Anh Quân	2152916	Cơ sở lý thuyết, mô hình Random Forest, XGBoost, làm slide thuyết trình	25%

1 CƠ SỞ LÝ THUYẾT

1.1 Kiểm định VIF (Variance Inflation Factor)

Kiểm tra VIF (Variance Inflation Factor) được sử dụng để đánh giá độ tương quan giữa các biến độc lập trong một mô hình hồi quy tuyến tính. Nó đo lường mức độ tăng lên trong phương sai của hệ số hồi quy do sự tương quan giữa biến độc lập và các biến còn lại trong mô hình. Giá trị VIF cao hơn 10 thường được coi là biểu hiện của đa cộng tuyến nghiêm trọng, làm suy giảm độ chính xác của các ước lượng hồi quy và làm tăng phương sai của các ước lượng đó.

Công thức tính VIF cho biến độc lập X_i là:

$$VIF(X_i) = \frac{1}{1 - R^2_{X_i}}$$

Trong đó $R^2_{X_i}$ là hệ số xác định của mô hình hồi quy tuyến tính có X_i là biến phụ thuộc và tất cả các biến độc lập còn lại.

Nếu giá trị VIF lớn hơn 10 hoặc 5, đây có thể là dấu hiệu của vấn đề đa cộng tuyến, và cần phải xem xét lại việc bao gồm biến đó trong mô hình.

1.2 Kiểm định ANOVA (Analysis of Variance)

Kiểm định ANOVA (Analysis of Variance) được sử dụng để so sánh trung bình của ba hoặc nhiều nhóm khác nhau. Nó kiểm tra xem có sự khác biệt ý nghĩa nào đó giữa các trung bình của các nhóm hay không.

Trong kiểm định ANOVA, giả thuyết không có ý nghĩa là tất cả các nhóm có cùng trung bình. Giả thuyết thay thế là ít nhất một cặp trung bình khác nhau.

Kiểm định ANOVA tính toán một giá trị thống kê được gọi là giá trị F, dựa trên phương sai giữa các nhóm và phương sai trong các nhóm. Nếu giá trị p nhỏ hơn một mức ý nghĩa đã chọn (thường là 0.05), chúng ta bác bỏ giả thuyết không có ý nghĩa và kết luận rằng có sự khác biệt ý nghĩa giữa ít nhất một cặp trung bình.

1.3 Sai số bình phương trung bình (MSE) và Sai số bình phương trung bình của căn bậc hai (RMSE)

Sai số bình phương trung bình (MSE) và sai số bình phương trung bình của căn bậc hai (RMSE) là hai phép đo thường được sử dụng để đánh giá hiệu suất của một mô hình dự đoán so với dữ liệu thực tế.

MSE được tính bằng cách lấy tổng bình phương của sai số (khác biệt giữa giá trị dự đoán và giá trị thực tế) và chia cho số lượng quan sát. Nó thể hiện trung bình của bình phương của sai số.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Trong đó n là số lượng quan sát, y_i là giá trị thực tế và $\hat{y}_i$ là giá trị dự đoán tương ứng.

RMSE là căn bậc hai của MSE và thường được sử dụng để đo lường sự sai lệch trung bình giữa các dự đoán và giá trị thực tế trong cùng đơn vị đo lường.

$$RMSE = \sqrt{MSE}$$

Những giá trị MSE và RMSE càng thấp thì mô hình dự đoán càng tốt.

1.4 Hệ số kappa của Cohen

Hệ số kappa của Cohen Hệ số kappa của Cohen là một chỉ số thống kê được sử dụng để đánh giá độ tin cậy giữa các nhà đánh giá cho các mục định tính (phân loại) . Đây là một thước đo mạnh mẽ hơn so với việc chỉ tính phần trăm thỏa thuận, vì nó xem xét khả năng xảy ra thỏa thuận do ngẫu nhiên . Hệ số này thường được sử dụng để đo lường sự thỏa thuận giữa hai nhà đánh giá, mỗi người phân loại các mục vào các danh mục khác nhau mà không chồng chéo . Hệ số được định nghĩa là tỷ lệ của thỏa thuận quan sát được so với thỏa thuận tối đa có thể, có tính đến thỏa thuận có thể xảy ra do ngẫu nhiên . Hệ số này dao động từ -1 đến 1, trong đó 1 biểu thị thỏa thuận hoàn hảo, 0 biểu thị thỏa thuận không tốt hơn ngẫu nhiên, và -1 biểu thị bất đồng hoàn toàn.

Với p_0 là độ chính xác tổng thể của mô hình, p_e là phép đo thỏa thuận giữa các dự đoán của mô hình và các giá trị thực tế như thể xảy ra do ngẫu nhiên, hệ số kappa của Cohen được tính bằng công thức sau:

$$\kappa = \frac{p_0 - p_e}{1 - p_e}$$

1.5 Kỹ thuật quantile flooring và capping

Quantile flooring và capping là hai phương pháp được sử dụng để xử lý các giá trị ngoại lai trong một tập dữ liệu. Các giá trị ngoại lai là những điểm dữ liệu nằm xa so với phần còn lại của các quan sát và có thể ảnh hưởng tiêu cực đến phân tích thống kê cũng như quá trình huấn luyện của thuật toán máy học, dẫn đến độ chính xác thấp hơn.

Trong kỹ thuật quantile flooring, giá trị ngoại lai được đặt ở mức thấp hơn một hệ số dưới giá trị phân vị thứ 10. Ví dụ, nếu giá trị phân vị thứ 10 là 10, thì bất kỳ giá trị nào dưới 0.9 lần 10 (tức là, 9) sẽ được đặt là 9.

Trong kỹ thuật quantile capping, giá trị ngoại lai được đặt ở mức cao hơn một giá trị nhất định trên giá trị phân vị thứ 90. Ví dụ, nếu giá trị phân vị thứ 90 là 100, thì bất kỳ giá trị nào trên 1.1 lần 100 (tức là, 110) sẽ được đặt là 110.

Những kỹ thuật này rất hữu ích khi tập dữ liệu chứa các giá trị cực đoan không đại diện cho phần lớn dữ liệu. Bằng cách sử dụng những kỹ thuật này, chúng ta có thể thay thế các giá trị cực đoan bằng các giá trị cố định, trong trường hợp này là các phân vị, mà không mất quá nhiều thông tin.

Tóm lại, quantile flooring và capping là hai phương pháp được sử dụng để xử lý các giá trị ngoại lai trong một tập dữ liệu. Chúng rất hữu ích khi tập dữ liệu có chứa các giá trị cực đoan không đại diện cho phần lớn dữ liệu. Bằng cách sử dụng các kỹ thuật này, chúng ta có thể thay thế các giá trị cực đoan bằng các giá trị cố định, trong trường hợp này là các phân vị, mà không mất quá nhiều thông tin.

1.6 Mô hình Naive Bayes

1. **Giới thiệu:** Mô hình Naive Bayes là một mô hình học máy phổ biến và đơn giản, thường được sử dụng cho các bài toán phân loại và dự đoán. Nó dựa trên nguyên lý của định lý Bayes và giả định "ngây thơ"(naive) rằng các đặc trưng là độc lập có điều kiện đối với lớp.

2. **Nguyên lý hoạt động:** Giả định "ngây thơ"trong mô hình Naive Bayes cho rằng các đặc trưng đầu vào là độc lập có điều kiện đối với lớp. Điều này có nghĩa là giá trị của mỗi đặc trưng được xem xét riêng lẻ và không phụ thuộc vào các đặc trưng khác khi đã biết lớp của mẫu dữ liệu.

   Dựa trên nguyên lý này, mô hình Naive Bayes tính xác suất của mỗi lớp cho một mẫu dữ liệu mới bằng cách sử dụng định lý Bayes:

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y) \times P(x_1, x_2, ..., x_n|y)}{P(x_1, x_2, ..., x_n)}$$

Trong đó: - $P(y|x_1, x_2, ..., x_n)$ là xác suất có điều kiện của lớp y cho một mẫu dữ liệu với các đặc trưng $x_1, x_2, ..., x_n$. - $P(y)$ là xác suất tiên nghiệm của lớp y. - $P(x_1, x_2, ..., x_n|y)$ là xác suất của các đặc trưng $x_1, x_2, ..., x_n$ trong lớp y. - $P(x_1, x_2, ..., x_n)$ là xác suất của các đặc trưng $x_1, x_2, ..., x_n$ trong tất cả các lớp.

3. **Ưu điểm và nhược điểm:**

- **Ưu điểm**:
  - Dễ hiểu và triển khai.
  - Hiệu quả trong nhiều tình huống, đặc biệt là khi số lượng đặc trưng lớn.
  - Hoạt động tốt với dữ liệu có nhiễu.
- **Nhược điểm**:
  - Giả định về sự độc lập giữa các đặc trưng không phù hợp với mọi loại dữ liệu thực tế.
  - Dễ bị ảnh hưởng bởi các đặc trưng không quan trọng.

1.7 Mô hình Logistic Regression (Hồi quy Logistic)

Hồi quy Logistic là một mô hình thống kê được sử dụng để phân loại nhị phân, tức dự đoán một đối tượng thuộc vào một trong hai nhóm. Hồi quy Logistic làm việc dựa trên nguyên tắc của hàm sigmoid – một hàm phi tuyến tự chuyển đầu vào của nó thành xác suất thuộc về một trong hai lớp nhị phân.

Hồi quy Logistic hoạt động dựa trên hàm Sigmoid, được biểu diễn như sau:

$$S = \frac{1}{1 + \mathrm{e}^{-z}}$$

Hàm Sigmoid nhận đầu vào là một giá trị z bất kỳ, và trả về đầu ra là một giá trị xác suất nằm trong khoảng [0,1]. Khi áp dụng vào mô hình Hồi quy Logistic với đầu vào là ma trận dữ liệu X và trọng số ω, ta có $z = X\omega$.

Việc huấn luyện của mô hình là tìm ra bộ trọng số ω sao cho đầu ra dự đoán của hàm Sigmoid gần với kết quả thực tế nhất. Để làm được điều này, ta sử dụng hàm mất mát (Loss Function) để đánh giá hiệu năng của mô hình. Mô hình càng tốt khi hàm mất mát càng nhỏ.

Hàm mất mát (Loss Function) là một hàm số được sử dụng để đo lường mức độ lỗi mà mô hình của chúng ta tạo ra khi dự đoán các kết quả từ dữ liệu đầu vào. Trong bài toán Hồi quy Logistic, chúng ta sử dụng hàm mất mát Cross-Entropy (còn gọi là Log Loss) để đánh giá hiệu năng của mô hình.

Hàm mất mát Cross-Entropy được định nghĩa như sau:

$$L(\omega) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(p_i) + (1 - y_i) log(1 - p_i)]$$

Trong đó:

- n: số lượng mẫu dữ liệu trong tập huấn luyện.
- y_i: giá trị thực tế của đầu ra thứ i.
- p_i: xác suất dự đoán thuộc lớp 1 của mô hình cho đầu vào thứ i.

Hàm Cross-Entropy đo lường khoảng cách giữa hai phân phối xác suất y_i và p_i. Khi mô hình dự đoán chính xác, tức là nếu $y_i = 1$ thì p_i càng gần 1, và nếu $y_i = 0$ thì p_i càng gần 0, sau đó hàm mất mát sẽ tiến gần về 0.

Trong quá trình huấn luyện, chúng ta tìm cách cập nhật bộ trọng số ω sao cho giá trị hàm mất mát Cross-Entropy đạt giá trị nhỏ nhất, dẫn đến một mô hình dự đoán tốt nhất.

Để tìm giá trị tối ưu cho bộ trọng số ω, chúng ta có thể sử dụng kỹ thuật Gradient Descent. Tại mỗi bước lặp, chúng ta cập nhật ω theo phương từm ứng với đạo hàm của hàm mất mát $L(\omega)$ theo ω.

1.8 Mô hình Decision Tree

1. **Giới thiệu:** Mô hình Decision Tree là một mô hình học máy phổ biến được sử dụng cho các bài toán phân loại và dự đoán. Decision Tree hoạt động bằng cách tạo ra một cây quyết định dựa trên các quy tắc if-else để dự đoán lớp hoặc giá trị mục tiêu cho dữ liệu mới.

2. **Nguyên lý hoạt động:** Mô hình Decision Tree chia dữ liệu dựa trên các đặc trưng thành các "nốt"(nodes) và áp đặt các quy tắc quyết định cho mỗi nốt. Khi dự đoán cho một mẫu dữ liệu mới, mô hình sẽ đi theo các quy tắc từ nốt gốc (root node) đến các nốt lá (leaf nodes) để quyết định lớp hoặc giá trị mục tiêu của mẫu đó.

3. **Ưu điểm và nhược điểm:**
   - **Ưu điểm:**
     - Dễ hiểu và diễn giải.
     - Có khả năng xử lý cả dữ liệu số liệu và dữ liệu phân loại.
     - Không cần nhiều tiền xử lý dữ liệu.
     - Cho phép dễ dàng biểu diễn quyết định và giải thích kết quả.
   - **Nhược điểm:**
     - Có thể dễ dàng bị overfitting nếu không được cắt tỉa (pruned) đúng cách.
     - Khá nhạy cảm với nhiễu và biến động trong dữ liệu.
     - Không thể đại diện cho mối quan hệ phức tạp giữa các đặc trưng.

1.9 Mô hình Support Vector Machine (SVM)

1. **Giới thiệu:** Mô hình Support Vector Machine (SVM) là một trong những phương pháp phân loại mạnh mẽ trong học máy. SVM hoạt động bằng cách tìm ra ranh giới quyết định tốt nhất giữa các lớp trong không gian đặc trưng.

2. **Nguyên lý hoạt động:** SVM tìm ra một ranh giới phân chia (hyperplane) sao cho khoảng cách từ các điểm dữ liệu gần nhất của mỗi lớp đến ranh giới đó là lớn nhất (rãnh hai bên). Điểm dữ liệu gần nhất này được gọi là các vector hỗ trợ (support vectors), từ đó xuất phát tên gọi của mô hình.

   Trong trường hợp dữ liệu không thể phân chia tuyến tính, SVM sử dụng một hàm biến đổi (kernel function) để ánh xạ dữ liệu từ không gian ban đầu sang một không gian chiều cao hơn, nơi một ranh giới phân chia tuyến tính có thể được tìm thấy.

3. **Hàm mất mát (Loss function):** Trong SVM, mục tiêu là tối đa hóa rãnh giữa các lớp, điều này tương đương với việc tối thiểu hóa hàm mất mát. Hàm mất mát thường được định nghĩa dựa trên việc đo khoảng cách từ các điểm dữ liệu đến ranh giới phân chia.

   Đối với bài toán phân loại nhị phân, hàm mất mát thường được biểu diễn như sau:

$$L(y, f(x)) = \max(0, 1 - y \cdot f(x))$$

   trong đó:

- y là nhãn lớp ($y \in \{-1, 1\}$).
- $f(x)$ là hàm quyết định, thường là một hàm tuyến tính.

4. **Ưu điểm và nhược điểm:**
   - **Ưu điểm:**
     - Hiệu suất cao trong các bài toán có dữ liệu chiều cao.
     - Hiệu quả khi có một số lượng lớn các đặc trưng.
     - Khả năng làm việc tốt với dữ liệu có tính chất phi tuyến tính bằng cách sử dụng các hàm kernel.
   - **Nhược điểm:**
     - Khó thực hiện và tối ưu đối với các tập dữ liệu lớn.
     - Đòi hỏi lựa chọn tham số tốt và xử lý dữ liệu một cách cẩn thận để tránh overfitting.
     - Không tốt khi đối mặt với dữ liệu có nhiễu lớn và lớp mất cân bằng.

1.10 Mô hình Random Forest

1. **Giới thiệu:** Mô hình Random Forest là một phương pháp học máy mạnh mẽ được sử dụng cho các bài toán phân loại và hồi quy. Nó là một mô hình dựa trên cây quyết định, được xây dựng từ nhiều cây quyết định độc lập, được huấn luyện trên các mẫu dữ liệu con của tập huấn luyện.

2. **Nguyên lý hoạt động:** Random Forest tạo ra một tập hợp các cây quyết định ngẫu nhiên từ dữ liệu huấn luyện bằng cách sử dụng phương pháp bagging (Bootstrap Aggregating). Mỗi cây trong Random Forest được xây dựng dựa trên một tập dữ liệu con được lấy mẫu từ tập huấn luyện theo nguyên tắc lấy mẫu tái chọn có thay thế.

   Khi dự đoán, Random Forest sử dụng kết quả từ tất cả các cây con và chọn lớp hoặc giá trị mục tiêu được biểu đồ dựa trên đa số phiếu bầu (voting) hoặc trung bình (for regression).

3. **Ưu điểm và nhược điểm:**
   - **Ưu điểm:**
     - Hiệu suất cao và khả năng chống overfitting tốt.
     - Có khả năng làm việc tốt với dữ liệu có nhiễu và dữ liệu có số lượng lớn các đặc trưng.
     - Cho phép đánh giá tầm quan trọng của các đặc trưng.
     - Dễ dàng sử dụng và không đòi hỏi nhiều tham số tinh chỉnh.
   - **Nhược điểm:**
     - Tốn nhiều tài nguyên tính toán so với một số mô hình khác, đặc biệt là khi số lượng cây lớn.
     - Không thể hiệu quả khi làm việc với dữ liệu có cấu trúc phức tạp và không tuyến tính.
     - Khó hiểu và giải thích kết quả so với một số mô hình khác như cây quyết định đơn lẻ.

1.11 Mô hình Extreme Gradient Boosting (XGBoost)

1. **Giới thiệu:** Extreme Gradient Boosting (XGBoost) là một trong những thuật toán phổ biến và mạnh mẽ nhất trong học máy, đặc biệt là trong các cuộc thi trên các nền tảng như Kaggle. XGBoost là một phiên bản cải tiến của thuật toán Gradient Boosting, được tối ưu hóa để đạt được hiệu suất tốt hơn và thời gian huấn luyện nhanh hơn.

2. **Nguyên lý hoạt động:** XGBoost xây dựng một tập hợp các cây quyết định tương tự như các thuật toán Gradient Boosting khác, nhưng điểm khác biệt chính là XGBoost sử dụng một hàm mất mát (loss function) có đạo hàm tính toán được, giúp tối ưu hóa quá trình huấn luyện một cách hiệu quả hơn.

   XGBoost sử dụng một số kỹ thuật tinh chỉnh để giảm overfitting và tăng độ chính xác, bao gồm regularization, subsampling và pruning.

3. **Hàm mất mát (Loss function):** Trong XGBoost, hàm mất mát được xây dựng dựa trên hàm mất mát của Gradient Boosting và thêm các thành phần regularization để tránh overfitting. Hàm mất mát chính thường được biểu diễn như sau:

$$\text{Loss} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

   Trong đó:

   - $L(y_i, \hat{y}_i)$ là hàm mất mát cho mẫu dữ liệu thứ i.
   - T là số lượng nốt của cây.
   - γ và λ là các tham số regularization.
   - w_j là trọng số của nốt thứ j.

4. **Ưu điểm và nhược điểm:**

   - **Ưu điểm:**
     - Hiệu suất cao và thời gian huấn luyện nhanh.
     - Tích hợp các kỹ thuật tinh chỉnh để giảm overfitting.
     - Có khả năng làm việc với dữ liệu lớn, có số lượng lớn các đặc trưng.
     - Tự động xử lý dữ liệu bị thiếu và dữ liệu không chuẩn.
   - **Nhược điểm:**
     - Cần phải lựa chọn các tham số tối ưu để tránh overfitting.
     - Khó hiểu và giải thích kết quả so với một số mô hình khác.
     - Đòi hỏi một lượng dữ liệu đủ lớn để hiệu quả.

1.12 Mô hình K-Nearest Neighbors (KNN)

1. **Giới thiệu:** Mô hình K-Nearest Neighbors (KNN) là một trong những thuật toán phân loại và hồi quy đơn giản nhất trong học máy. Nó dựa trên nguyên lý rằng các điểm dữ liệu cùng lớp thường có các đặc trưng gần nhau trong không gian đặc trưng.

2. **Nguyên lý hoạt động:** KNN phân loại một điểm dữ liệu mới bằng cách sử dụng đa số phiếu bầu (voting) của các điểm dữ liệu gần nhất trong tập huấn luyện. KNN đo khoảng cách từ điểm dữ liệu mới đến tất cả các điểm trong tập huấn luyện, sau đó chọn ra k điểm gần nhất và gán lớp cho điểm mới dựa trên đa số phiếu bầu của các điểm gần nhất đó.

3. **Tham số K:** Tham số K trong KNN là số lượng điểm gần nhất mà thuật toán sử dụng để dự đoán lớp của một điểm dữ liệu mới. Lựa chọn tham số K quan trọng và có thể ảnh hưởng đến hiệu suất của mô hình.

4. **Hàm đo khoảng cách:** Để xác định các điểm gần nhất, KNN sử dụng các hàm đo khoảng cách như Euclidean distance, Manhattan distance, hoặc các phương pháp đo khoảng cách khác tùy thuộc vào loại dữ liệu.

5. **Ưu điểm và nhược điểm:**

   - **Ưu điểm:**
     - Đơn giản và dễ triển khai.
     - Không cần huấn luyện trước.
     - Hoạt động tốt cho dữ liệu có biến động cao hoặc không gian đặc trưng phức tạp.
   - **Nhược điểm:**
     - Yêu cầu lưu toàn bộ tập dữ liệu trong bộ nhớ.
     - Đòi hỏi tính toán khoảng cách đến tất cả các điểm dữ liệu trong tập huấn luyện.
     - Nhạy cảm với dữ liệu nhiễu và lớp mất cân bằng.

1.13 Mô hình Mạng Neuron nhân tạo (Artificial Neural Network – ANN)

- **Định nghĩa ANN**

Mạng neuron nhân tạo (Artificial Neuron Network) là một mô hình tính toán bắt chước cách thức hoạt động của các tế bào thần kinh trong não người. Mạng neuron nhân tạo (ANN) sử dụng các giải thuật learning có thể thực hiện các điều chỉnh một cách độc lập – hoặc học theo một nghĩa nào đó – khi chúng nhận được giá trị input mới.

- **Cấu trúc cơ bản của Mạng neuron nhân tạo (ANN)**

Cấu trúc cơ bản của mạng neuron nhân tạo (ANN) được thiết kế để mô phỏng cách thức hoạt động của bộ não sinh học, nơi thông tin được xử lý và truyền đi qua một mạng lưới phức tạp của neuron. Trong ANN, đơn vị cơ bản nhất là neuron nhân tạo, hay còn gọi là nút, được lập trình để nhận đầu vào, xử lý chúng qua một hàm toán học, và sinh ra đầu ra. Mỗi neuron có thể kết nối với nhiều neuron khác, và thông qua mạng lưới các kết nối này, ANN có thể thực hiện các tác vụ phức tạp như học và ra quyết định.

Kiến trúc của ANN thường bao gồm ba loại lớp chính:

- Lớp Đầu Vào (Input Layer): Là cổng đầu vào của mạng, nơi dữ liệu được cung cấp vào mô hình. Mỗi neuron trong lớp này tương ứng với một thuộc tính/đặc trưng của dữ liệu đầu vào.
- Lớp Ẩn (Hidden Layers): Nằm giữa lớp đầu vào và lớp đầu ra, lớp ẩn có thể có một hoặc nhiều lớp. Các neuron trong các lớp ẩn thực hiện phần lớn xử lý thông qua việc kết hợp và biến đổi dữ liệu đầu vào, thực hiện các phép tính toán học để học các đặc điểm và mối quan hệ phức tạp từ dữ liệu.
- Lớp Đầu Ra (Output Layer): Chứa thông tin đầu ra của mạng, dựa trên việc học từ dữ liệu đầu vào và các lớp ẩn. Số lượng neuron trong lớp này tương ứng với số lượng đầu ra mong muốn, ví dụ như các lớp trong bài toán phân loại.

Thông tin trong ANN được truyền từ lớp đầu vào qua các lớp ẩn và cuối cùng đến lớp đầu ra thông qua một quá trình được gọi là lan truyền tiến. Tại mỗi neuron, đầu vào từ các neuron khác được tổng hợp và xử lý bằng một hàm kích hoạt, sau đó đầu ra của neuron được gửi đến các neuron tiếp theo mà nó kết nối. Trong suốt quá trình này, trọng số của mỗi kết nối – thể hiện mức độ quan trọng của kết nối đó trong việc xác định đầu ra của mạng – được điều chỉnh thông qua quá trình học để mô hình có thể tối ưu hóa việc dự đoán hoặc phân loại dữ liệu.

- **Nguyên lý hoạt động của ANN**

ANN hoạt động dựa trên nguyên lý mô phỏng quá trình xử lý thông tin của bộ não sinh học, qua đó ANN nhận dữ liệu đầu vào, xử lý và cuối cùng đưa ra dự đoán hoặc phân loại. Cơ chế cơ bản của ANN bao gồm hai quá trình chính: lan truyền tiến (forward propagation) và tính toán lỗi, cùng với sự hỗ trợ của hàm kích hoạt.

- **Lan Truyền Tiến (Forward Propagation)**: Quá trình này bắt đầu khi dữ liệu đầu vào được cung cấp cho lớp đầu vào của ANN. Mỗi neuron trong lớp này sẽ tiếp nhận một giá trị đầu vào cụ thể. Sau đó, dữ liệu được truyền qua mạng từ lớp này sang các lớp ẩn, và cuối cùng đến lớp đầu ra. Tại mỗi neuron, dữ liệu đầu vào sẽ được nhân với trọng số của kết nối – giá trị này thể hiện mức độ quan trọng của kết nối đối với việc xác định đầu ra của neuron. Kết quả của phép nhân này, cùng với một giá trị bias, sẽ được tổng hợp và xử lý qua một hàm kích hoạt để tạo ra đầu ra của neuron.

- **Tính Toán Lỗi**: Sau khi dữ liệu được lan truyền qua tất cả các lớp và đến lớp đầu ra, ANN sẽ so sánh kết quả đầu ra với giá trị đích thực tế để xác định lỗi. Lỗi này thường được tính bằng cách sử dụng một hàm lỗi, như Mean Squared Error (MSE) hoặc Cross-Entropy. Việc tính toán lỗi giúp mạng xác định mức độ chênh lệch giữa kết quả dự đoán và thực tế, từ đó hướng dẫn quá trình điều chỉnh trọng số trong quá trình học.

- **Hàm Kích Hoạt (Activation Functions)**: Hàm kích hoạt đóng vai trò quan trọng trong ANN bằng cách quyết định liệu một neuron có nên được "kích hoạt" hay không, tức là liệu nó có nên truyền thông tin của mình đến các neuron tiếp theo trong mạng. Các hàm kích hoạt phổ biến bao gồm ReLU (Rectified Linear Unit), Sigmoid, và Tanh. Mỗi hàm có những đặc điểm riêng biệt phù hợp với các loại bài toán khác nhau. Hàm kích hoạt giúp thêm tính phi tuyến vào mô hình, cho phép ANN học được các mối quan hệ phức tạp và phi tuyến tính trong dữ liệu.

  Qua những bước này, ANN xử lý dữ liệu đầu vào, tính toán và đưa ra dự đoán, đồng thời thông qua quá trình học, mạng liên tục điều chỉnh trọng số dựa trên lỗi tính được, nhằm giảm thiểu sai số và cải thiện khả năng dự đoán trong tương lai.

2 Tổng quan về tập dữ liệu

2.1 Vấn đề

Tiểu đường là một trong những bệnh mãn tính phổ biến nhất tại Hoa Kỳ, ảnh hưởng đến hàng triệu người Mỹ mỗi năm và gây ra một gánh nặng tài chính đáng kể cho nền kinh tế. Tiểu đường là một căn bệnh mãn tính nghiêm trọng, trong đó người bị mất khả năng điều chỉnh mức đường glucose trong máu một cách hiệu quả, có thể dẫn đến giảm chất lượng cuộc sống và tuổi thọ. Sau khi các loại thức ăn được phân hủy thành đường trong quá trình tiêu hóa, các đường đó sau đó được giải phóng vào hệ tuần hoàn máu. Điều này kích thích tụy tiết ra insulin. Insulin giúp các tế bào trong cơ thể sử dụng các đường đó trong tuần hoàn máu để tạo năng lượng. Tiểu đường thường được đặc trưng bởi việc cơ thể không tạo ra đủ insulin hoặc không sử dụng insulin được tạo ra một cách hiệu quả như cần thiết.

Các biến chứng như bệnh tim, mất thị lực, cắt cụt chi dưới, và bệnh thận liên quan đến việc duy trì mức đường huyết cao mãi mãi trong hệ tuần hoàn máu của những người mắc tiểu đường. Mặc dù không có phương pháp chữa trị cho tiểu đường, các chiến lược như giảm cân, ăn uống lành mạnh, vận động, và nhận điều trị y tế có thể làm giảm tổn thất từ căn bệnh này ở nhiều bệnh nhân. Việc chẩn đoán sớm có thể dẫn đến thay đổi lối sống và điều trị hiệu quả hơn, làm cho các mô hình dự đoán về rủi ro tiểu đường trở thành công cụ quan trọng cho các cơ quan y tế và sức khỏe cộng đồng.

Quy mô của vấn đề này cũng quan trọng phải nhận thức. Trung tâm Kiểm soát và Phòng ngừa Dịch bệnh đã chỉ ra rằng tính đến năm 2018, có 34,2 triệu người Mỹ mắc tiểu đường và 88 triệu người có tiền tiểu đường. Hơn nữa, CDC ước tính rằng 1 trên 5 người mắc tiểu đường và khoảng 8 trên 10 người có tiền tiểu đường không nhận thức được rủi ro của mình. Mặc dù có nhiều loại tiểu đường khác nhau, tiểu đường loại II là hình thức phổ biến nhất và tỷ lệ phổ biến của nó thay đổi theo tuổi tác, giáo dục, thu nhập, vị trí, sắc tộc, và các nhân tố xác định xã hội khác về sức khỏe. Phần lớn gánh nặng của căn bệnh đè nặng lên những người thu nhập thấp. Tiểu đường cũng gây gánh nặng lớn cho nền kinh tế, với chi phí của tiểu đường đã được chẩn đoán khoảng 327 tỷ đô la và tổng chi phí với tiểu đường không được chẩn đoán và tiền tiểu đường tiếp cận 400 tỷ đô la hàng năm.

2.2 Mô tả tập dữ liệu

Hệ thống Ghi nhận Yếu tố Rủi ro Hành vi (BRFSS) là một cuộc điều tra qua điện thoại liên quan đến sức khỏe được thu thập hàng năm bởi CDC. Mỗi năm, cuộc điều tra thu thập phản ứng từ hơn 400.000 người Mỹ về các hành vi liên quan đến sức khỏe, các điều kiện sức khỏe mãn tính và việc sử dụng dịch vụ phòng ngừa. Nó đã được tiến hành mỗi năm kể từ năm 1984.

Đối với dự án này, một tệp `csv` của tập dữ liệu có sẵn trên Kaggle cho năm 2015 đã được sử dụng. Tập dữ liệu gốc này chứa các phản ứng từ 441.455 cá nhân và có 330 thuộc tính. Những thuộc tính này hoặc là các câu hỏi được đặt trực tiếp cho người tham gia, hoặc là các biến được tính toán dựa trên các phản ứng của từng người tham gia.

Tập dữ liệu này chứa 3 tệp:

- `diabetes_012_health_indicators_BRFSS2015.csv` là một tập dữ liệu sạch với 253.680 phản ứng cuộc điều tra đến BRFSS2015 của CDC. Biến mục tiêu `Diabetes_012` có 3 lớp. 0 là không có tiểu đường hoặc chỉ trong thời kỳ mang thai, 1 là tiền tiểu đường và 2 là tiểu đường. Có mất cân bằng lớp trong tập dữ liệu này. Tập dữ liệu này có 21 biến thuộc tính.

- `diabetes_binary_5050split_health_indicators_BRFSS2015.csv` là một tập dữ liệu sạch với 70.692 phản ứng cuộc điều tra đến BRFSS2015 của CDC. Nó có sự chia sẻ 50-50 bằng nhau của người phản hồi không mắc tiểu đường và có tiền tiểu đường hoặc tiểu đường. Biến mục tiêu `Diabetes_binary` có 2 lớp. 0 là không mắc tiểu đường, và 1 là có tiền tiểu đường hoặc tiểu đường. Tập dữ liệu này có 21 biến thuộc tính và được cân bằng.

- `diabetes_binary_health_indicators_BRFSS2015.csv` là một tập dữ liệu sạch với 253.680 phản ứng cuộc điều tra đến BRFSS2015 của CDC. Biến mục tiêu `Diabetes_binary` có 2 lớp. 0 là không

mắc tiểu đường, và 1 là có tiền tiểu đường hoặc tiểu đường. Tập dữ liệu này có 21 biến thuộc tính và không được cân bằng.

Khám phá một số câu hỏi nghiên cứu sau:

1. Các câu hỏi của cuộc điều tra BRFSS có thể cung cấp dự đoán chính xác về việc một cá nhân có tiểu đường không?
2. Những yếu tố rủi ro nào có khả năng dự đoán cao nhất về nguy cơ tiểu đường?
3. Chúng ta có thể sử dụng một phần của các yếu tố rủi ro để dự đoán chính xác liệu một cá nhân có tiểu đường không?
4. Chúng ta có thể tạo ra một biểu mẫu ngắn gọn của các câu hỏi từ BRFSS bằng cách lựa chọn thuộc tính để dự đoán chính xác liệu ai đó có thể mắc tiểu đường hoặc có nguy cơ cao mắc tiểu đường không?

2.3 Mô tả các biến trong tập dữ liệu

Biến	Loại dữ liệu	Đơn vị	Mô tả
Diabetes_binary	cate	None	Indicate if the patient has diabetes (1 = Yes, 0 = No)
HighBP	cate	None	Indicate if the patient has high blood pressure (1 = Yes, 0 = No)
HighChol	cate	None	Indicate if the patient has high cholesterol (1 = Yes, 0 = No)
CholCheck	cate	None	Indicate if the patient checks cholesterol (1 = Yes, 0 = No)
BMI	cont	None	Body Mass Index of the patient
Smoker	cate	None	Indicate if the patient smokes (1 = Yes, 0 = No)
Stroke	cate	None	Indicate if the patient has had a stroke (1 = Yes, 0 = No)
HeartDiseaseorAttack	cate	None	Indicate if the patient has heart disease or has had a heart attack (1 = Yes, 0 = No)
PhysActivity	cate	None	Indicate the level of physical activity of the patient
Fruits	cate	None	Indicate if the patient consumes fruits regularly (1 = Yes, 0 = No)
AnyHealthcare	cate	None	Indicate if the patient has access to any form of healthcare (1 = Yes, 0 = No)
NoDocbcCost	cate	None	Indicate if the patient has no out-of-pocket healthcare costs (1 = Yes, 0 = No)
GenHlth	cont	None	General health rating of the patient
MentHlth	cont	None	Mental health rating of the patient
PhysHlth	cont	None	Physical health rating of the patient
DiffWalk	cate	None	Indicate if the patient has difficulty walking (1 = Yes, 0 = No)
Sex	cate	None	Gender of the patient (1 = Male, 0 = Female)
Age	cont	None	Age of the patient
Education	cont	None	Education level of the patient
Income	cont	None	Income level of the patient

3 Tiền xử lý dữ liệu

3.1 Tổng quan

Đọc dữ liệu từ một URL trực tuyến, sau đó đổi tên cột 'Diabetes_012' thành 'Diabetes_binary'

```
url = 'https://raw.githubusercontent.com/PhucLe03/Datasets-for-Diabetes-Prediction-using-Classification-Models/main/diabetes_012_health_indicators_BRFSS2015.csv'
df = pd.read_csv(url)
df.rename(columns={'Diabetes_012': 'Diabetes_binary'}, inplace=True)
```

Kết quả:

Kết quả 5 hàng đầu tiên của dữ liệu

```
   Diabetes_binary  HighBP  HighChol  CholCheck   BMI  Smoker  Stroke  \
0              0.0     1.0       1.0        1.0  40.0     1.0     0.0
1              0.0     0.0       0.0        0.0  25.0     1.0     0.0
2              0.0     1.0       1.0        1.0  28.0     0.0     0.0
3              0.0     1.0       0.0        1.0  27.0     0.0     0.0
4              0.0     1.0       1.0        1.0  24.0     0.0     0.0

   HeartDiseaseorAttack  PhysActivity  Fruits  ...  AnyHealthcare  \
0                   0.0           0.0     0.0  ...            1.0
1                   0.0           1.0     0.0  ...            0.0
2                   0.0           0.0     1.0  ...            1.0
3                   0.0           1.0     1.0  ...            1.0
4                   0.0           1.0     1.0  ...            1.0

   NoDocbcCost  GenHlth  MentHlth  PhysHlth  DiffWalk  Sex   Age  Education  \
0          0.0      5.0      18.0      15.0       1.0  0.0   9.0        4.0
1          1.0      3.0       0.0       0.0       0.0  0.0   7.0        6.0
2          1.0      5.0      30.0      30.0       1.0  0.0   9.0        4.0
3          0.0      2.0       0.0       0.0       0.0  0.0  11.0        3.0
4          0.0      2.0       3.0       0.0       0.0  0.0  11.0        5.0

   Income
0     3.0
1     1.0
2     8.0
3     6.0
4     4.0

[5 rows x 22 columns]
```

Bản tóm tắt thông tin về Diabetes

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Diabetes_binary       253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

Chuyển các giá trị khác 0 trong cột 'Diabetes_012' thành 1 và kiểm tra lại các giá trị của cột 'Diabetes_012'

```
df['Diabetes_binary'] = df['Diabetes_binary'].replace([1.0, 2.0], 1.0)
print(df.groupby('Diabetes_binary').size())
```

Bản tóm tắt thống kê cho các số liệu thống kê trong dữ liệu Diabetes

```
Diabetes_binary
0.0    213703
1.0     39977
dtype: int64
```

Kiểm tra giá trị NULL

Kết quả kiểm tra giá trị NULL

```
#Check null values
df.isnull().sum()
Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

Kiểm tra và đếm số lượng hàng trùng lặp và xóa các hàng trùng lặp khỏi tập dữ liệu

```
#Check duplicate
df.duplicated().sum()
#Drop duplicate rows
df.drop_duplicates(inplace=True)
df.shape
```

Chuyển đổi cột 'Diabetes_binary' thành 'Diabetes', loại bỏ cột cũ và hiển thị tần suất của mỗi giá trị trong các cột còn lại.

Kết quả tần suất của mỗi giá trị

```
    HighBP              HighChol            CholCheck
0.0 125336        0.0 128249          1.0 220414
1.0 104376        1.0 101463          0.0   9298

BMI
27.0    21543
26.0    17803
24.0    16528
28.0    14928
25.0    14804
...

Smoker              Stroke              HeartDiseaseorAttack
0.0 122738        0.0 219429                0.0 206002
1.0 106974        1.0 10283                 1.0  23710

PhysActivity      Fruits                Veggies
1.0 168444        1.0 140792        1.0 182566
0.0  61268         0.0  88920         0.0  47146

HvyAlcoholConsump     AnyHealthcare       NoDocbcCost
0.0 215762            1.0 217321        0.0 208386
1.0  13950            0.0  12391        1.0  21326

GenHlth             MentHlth            PhysHlth
2.0 77502         0.0 152554            0.0 136811
3.0 73684         2.0  12697           30.0  19385
1.0 34903         30.0 12080            2.0  14494

DiffWalk              Sex               Age
0.0 187087        0.0 128836         9.0  29715
1.0  42625        1.0 100876        10.0  29147

Education             Income            Diabetes
6.0 88396           8.0 71774           0.0 190055
5.0 66484           7.0 40178           1.0  39657
4.0 61151           6.0 34995
3.0  9467           5.0 25341
2.0  4040           4.0 19955
1.0   174           3.0 15920
                    2.0 11757
                    1.0  9792
```

Chuyển đổi kiểu dữ liệu của tất cả các cột trong DataFrame df thành kiểu số nguyên 32-bit

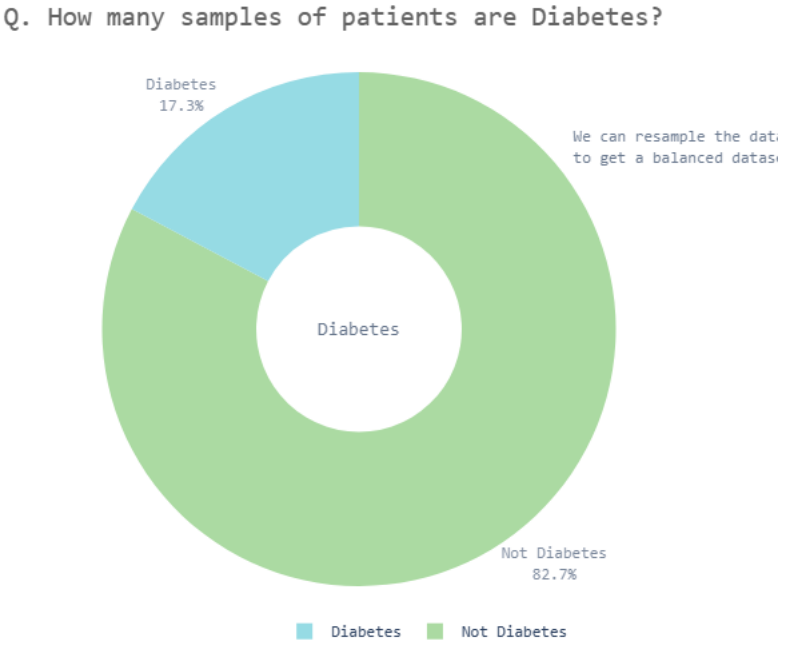
```
1  df = df.astype('int32')
```

3.2 EDA

Tạo một biểu đồ tròn sử dụng thư viện Plotly Express để thể hiện phân phối của các mẫu trong cột 'Diabetes'. Biểu đồ tròn này hiển thị tỉ lệ giữa số lượng mẫu không có tiểu đường ('Not Diabetes') và số lượng mẫu có tiểu đường ('Diabetes'). Các mẫu không có tiểu đường được đại diện bằng màu xanh lá cây, trong khi các mẫu có tiểu đường được đại diện bằng màu xanh dương.

Ngoài ra, biểu đồ cũng có các chú thích và điều chỉnh về văn bản và kiểu dáng để tạo ra một giao diện trực quan và dễ hiểu.

```
d= pd.DataFrame(df['Diabetes'].value_counts())
fig = px.pie(values=d['count'], names=['Not Diabetes','Diabetes'], hole=0.4, opacity=0.6,
             color_discrete_sequence=[colors_green[3], colors_blue[3]],
             labels={'label': 'Diabetes', 'Diabetes': 'No. Of Samples'})

fig.add_annotation(text='We can resample the data<br> to get a balanced dataset',
                   x=1.2,y=0.9,showarrow=False,font_size=12,opacity=0.7,font_family='monospace')
fig.add_annotation(text='Diabetes',
                   x=0.5,y=0.5,showarrow=False,font_size=14,opacity=0.7,font_family='monospace')

fig.update_layout(
    font_family='monospace',
    title=dict(text='Q. How many samples of patients are Diabetes?',x=0.47,y=0.98,
               font=dict(color=colors_dark[2],size=20)),
    legend=dict(x=0.37,y=-0.05,orientation='h',traceorder='reversed'),
    hoverlabel=dict(bgcolor='white'))

fig.update_traces(textposition='outside', textinfo='percent+label')

fig.show()
```

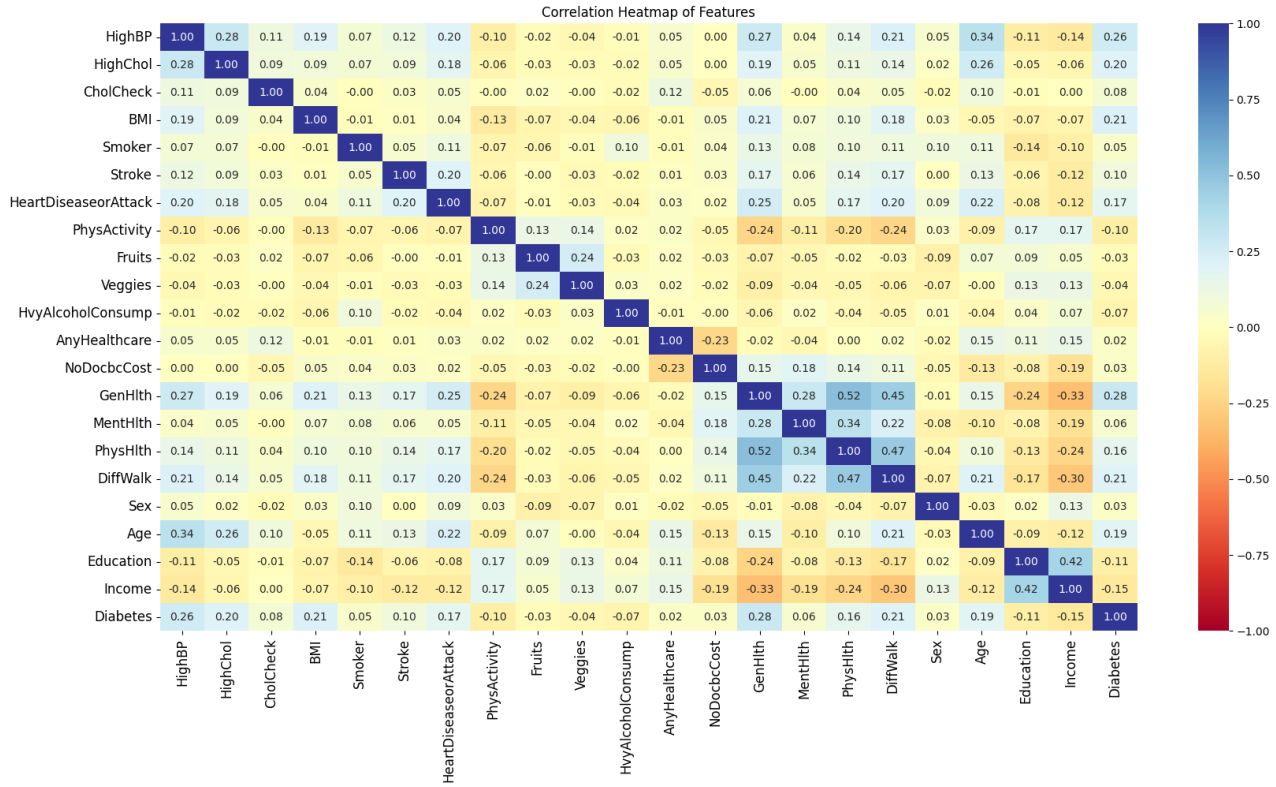


Hình 1: Phân phối các mẫu trong cột 'Diabetes'

Tạo ra một biểu đồ heatmap để thể hiện ma trận tương quan giữa các biến trong DataFrame df bằng thư viện seaborn. Mỗi ô trong heatmap sẽ hiển thị giá trị tương quan giữa hai biến, được biểu diễn bằng

màu sắc, trong đó màu đậm tương ứng với giá trị tương quan cao hơn và màu nhạt tương ứng với giá trị tương quan thấp hơn. Các giá trị tương quan được hiển thị trong từng ô.

```
plt.figure(figsize=(20, 10))

sns.heatmap(df.corr(), annot=True, cmap='RdYlBu', vmin=-1, vmax=1, fmt=".2f")
plt.title('Correlation Heatmap of Features')

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```



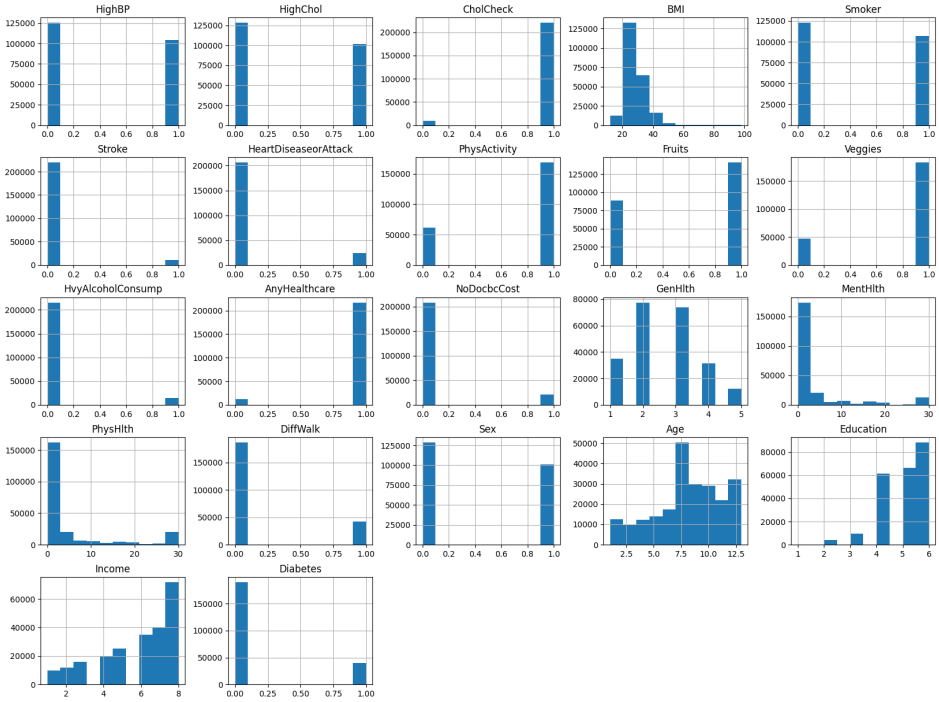
Hình 2: Heatmap thể hiện ma trận tương quan giữa các biến

Tạo ra một tập hợp các biểu đồ histogram cho từng biến trong DataFrame df sử dụng matplotlib. Mỗi histogram biểu diễn phân phối của một biến, giúp bạn hiểu rõ hơn về phân phối của dữ liệu.

```
    df.hist(figsize=(20,15))
```

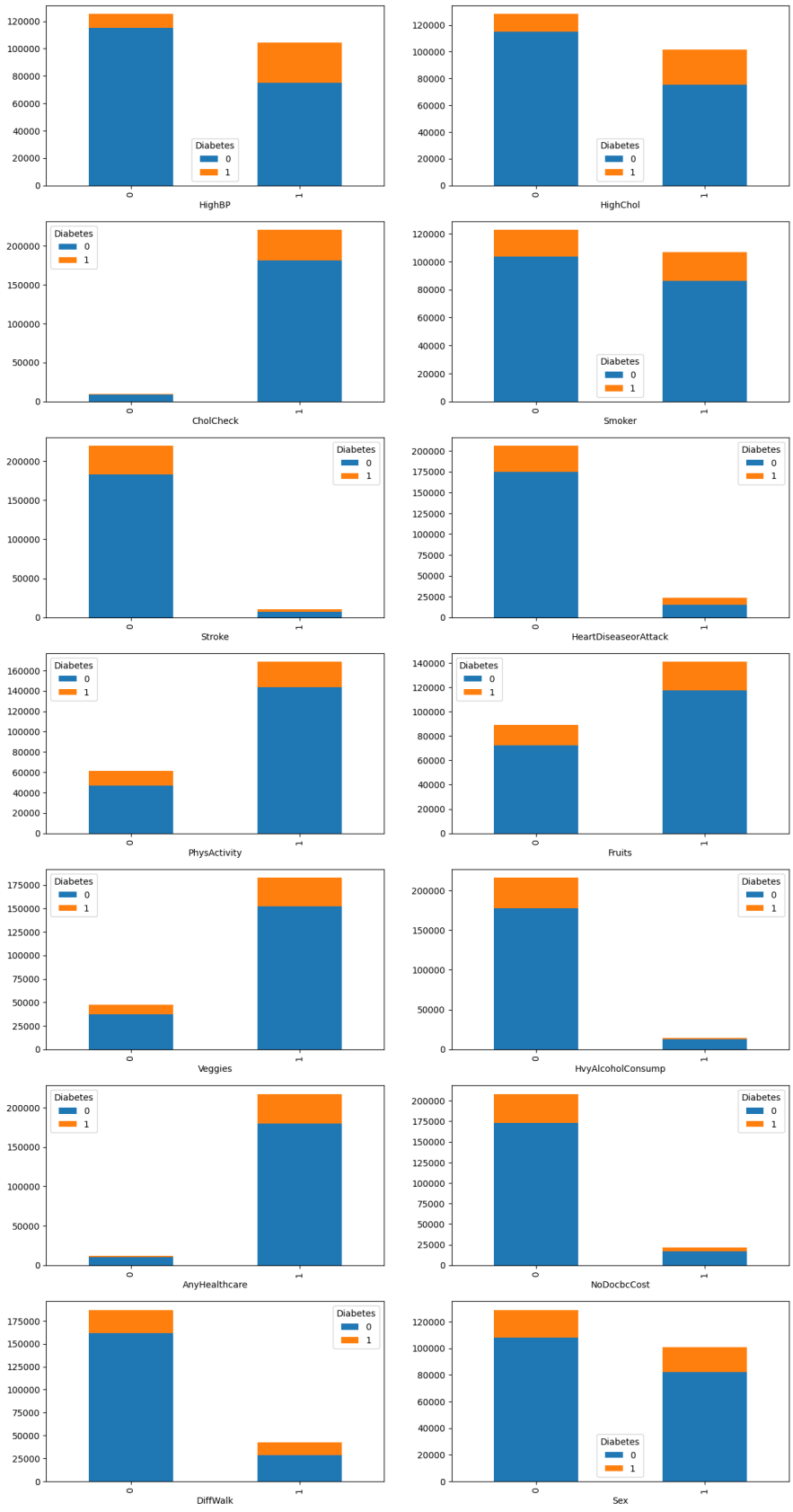
3.2.1 Biểu diễn trực quan các biến nhị phân

Tạo ra một loạt các biểu đồ cột xếp chồng cho từng biến trong danh sách cols, hiển thị số lượng mẫu của mỗi giá trị của biến đó, được phân tách bởi trạng thái tiểu đường (Diabetes). Mỗi biểu đồ cột biểu diễn phân phối của biến tương ứng, với các cột được xếp chồng lên nhau để thể hiện phân phối của mỗi giá trị trong biến theo trạng thái tiểu đường.



Hình 3: Phân phối các mẫu trong cột 'Diabetes'

```
cols = ['HighBP', 'HighChol', 'CholCheck','Smoker',
      'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
      'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'DiffWalk', 'Sex']
def create_plot(df, columnx):
  dfplot = df.groupby([columnx, 'Diabetes']).size() \
  .reset_index().pivot(columns='Diabetes', index=columnx, values=0)
  return dfplot

fig, ax=plt.subplots(7, 2, figsize=(15,30))
ax=ax.ravel()
c=len(cols)
for i in range(c):
  create_plot(df,cols[i]).plot(kind='bar',stacked=True,ax=ax[i])

fig.show()
```



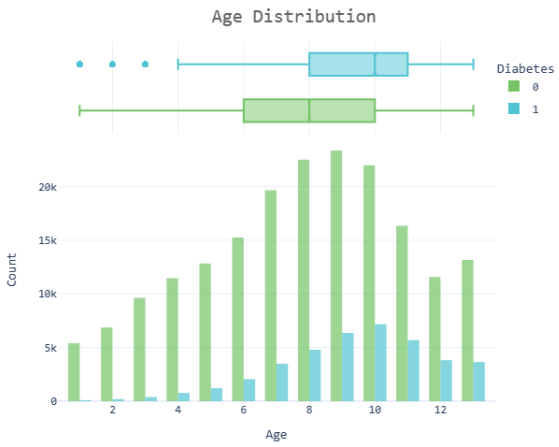
Hình 4: Biểu đồ cột xếp chồng hiển thị số lượng mẫu của mỗi giá trị của biến

3.2.2 Biểu diễn trực quan các biến liên tục

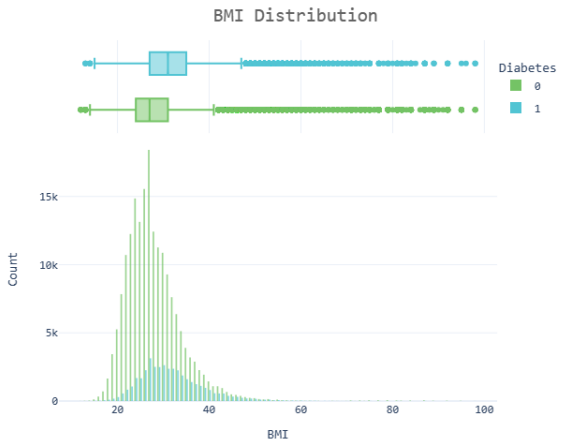
Sử dụng Plotly Express để tạo biểu đồ histogram thể hiện phân phối của chỉ số BMI trong DataFrame df, với mỗi cột biểu diễn một phân bin và màu sắc phân biệt giữa các mẫu có tiểu đường và không có tiểu đường. Biểu đồ bao gồm hộp để thể hiện phân phối chi tiết hơn của dữ liệu và các thiết lập khác nhau như số lượng bins, màu sắc, và tiêu đề.

```
    fig = px.histogram(df, x='BMI', color='Diabetes', template='plotly_white',
                       marginal='box', opacity=0.7, nbins=100,
                       color_discrete_sequence=[colors_green[3], colors_blue[3]],
                       barmode='group', histfunc='count')

    fig.update_layout(
        font_family='monospace',
        title=dict(text='BMI Distribution', x=0.53, y=0.95,
                   font=dict(color=colors_dark[2], size=20)),
        xaxis_title_text='BMI',
        yaxis_title_text='Count',
        legend=dict(x=1, y=0.96, bordercolor=colors_dark[4], borderwidth=0, tracegroupgap=5),
        bargap=0.3,
    )
    fig.show()
```

(a) Income

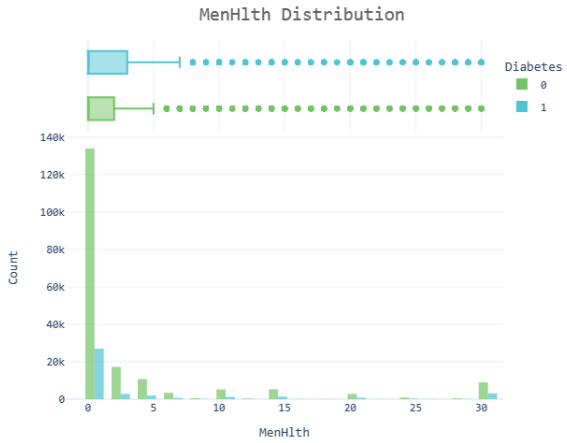


(b) Age

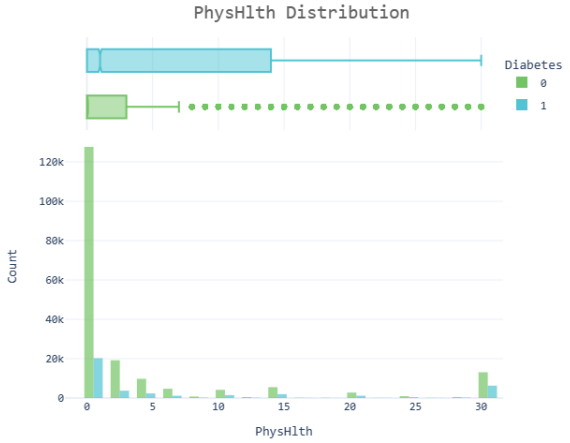


(c) BMI

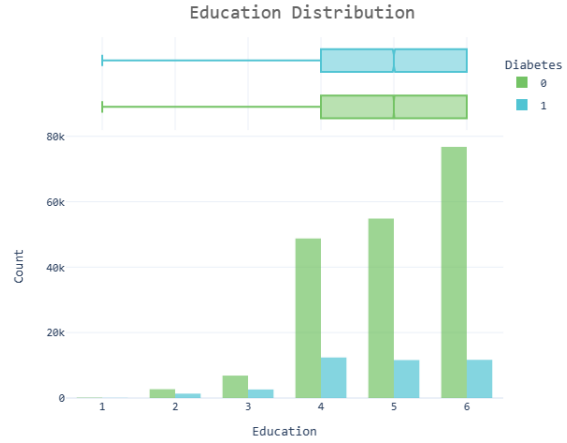
(a) GenHlth



(b) MenHlth



(c) PhysHlth

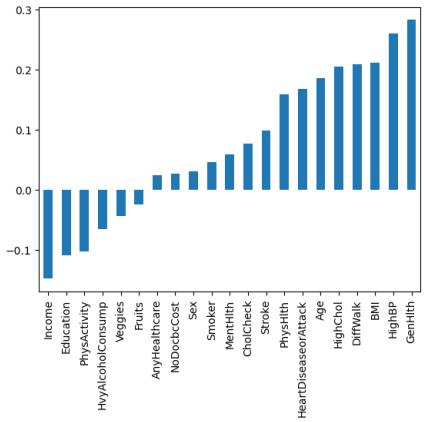


(d) Education

3.3 Chọn lựa biến độc lập

Tạo ra một biểu đồ cột để thể hiện mức độ tương quan giữa các biến độc lập và biến mục tiêu Diabetes_binary trong DataFrame df. Các biến độc lập được sắp xếp theo mức độ tương quan tăng dần với biến mục tiêu.

```
df.corr()['Diabetes'][:-1].sort_values().plot(kind='bar')
```



Hình 7: Biểu đồ cột mức độ tương quan tăng dần

Tính toán chỉ số VIF cho mỗi biến trong DataFrame x, đánh giá mức độ đa cộng tuyến giữa các biến. Kết quả được trả về dưới dạng DataFrame với hai cột: 'variables' chứa tên của các biến và 'VIF' chứa giá trị VIF tương ứng.

```
def calc_VIF(x):
    vif= pd.DataFrame()
    vif['variables']=x.columns
    vif["VIF"]=[variance_inflation_factor(x.values,i) for i in range(x.shape[1])]

    return(vif)


X = add_constant(df)
ds=pd.Series([variance_inflation_factor(X.values, i) for i in
    range(X.shape[1])],index=X.columns)
print(ds)
```

Kết quả 5 hàng đầu tiên của dữ liệu

```
const                   109.500762
HighBP                    1.315411
HighChol                  1.167976
CholCheck                 1.036130
BMI                       1.143750
Smoker                    1.076205
Stroke                    1.077837
HeartDiseaseorAttack      1.169848
PhysActivity              1.130727
Fruits                    1.098085
Veggies                   1.098279
HvyAlcoholConsump         1.027728
AnyHealthcare             1.110008
NoDocbcCost               1.135781
GenHlth                   1.742215
MentHlth                  1.221841
PhysHlth                  1.594588
DiffWalk                  1.514007
Sex                       1.076672
Age                       1.360478
Education                 1.272545
Income                    1.432718
Diabetes                  1.193754
dtype: float64
```

Chọn ra 10 đặc trưng tốt nhất từ tập dữ liệu ban đầu để sử dụng trong việc xây dựng mô hình dự đoán tiểu đường, dựa trên phương pháp F-score của phân tích ANOVA

```
X = df.iloc[:,1:]
Y = df.iloc[:,0]
# define feature selection
fs = SelectKBest(score_func=f_classif, k=10)
# apply feature selection
X_selected = fs.fit_transform(X, Y)
print(X_selected.shape)
# Result: (229712,10)
```

Kết luận: các biến được chọn bao gồm 'HighBP', 'HighChol', 'BMI', 'Smoker', 'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'HvyAlcoholConsump', 'NoDocbcCost', 'GenHlth', 'MentHlth', 'PhysHlth', 'DiffWalk', 'Age', 'Education', và 'Income'.

3.4 Xử lý ngoại lai

Trước khi xây dựng mô hình để dự đoán khả năng mắc bệnh tiểu đường, chúng ta cần xử lý các phần tử ngoại lai. Đây là khâu quan trọng để làm giảm thiểu việc dữ liệu có thể bị nhiễu, từ đó khiến độ chính xác của việc dự đoán bị giảm xuống.

Đoạn mã dưới đây để vừa thống kê tỉ lệ ngoại lai của các biến độc lập đã được chọn trong tập dữ liệu và cũng đồng thời xử lý dữ liệu ngoại lai của chúng:

```
# List of variables are needed to be check
variables = ['BMI', 'GenHlth', 'MentHlth', 'PhysHlth', 'Age', 'Education', 'Income']
# Check rate of outlier
for var in variables:
    # Calculate IQR
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    # Count number of outlier
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    num_outliers = df[(df[var] < lower_bound) | (df[var] > upper_bound)].shape[0]

    # Calculate rate of outlier
    percent_outliers = (num_outliers / df.shape[0]) * 100

    # Process outlier
    if percent_outliers < 5:
        # Replace by mean values
        mean_value = df[var].mean()
        df[var] = df[var].apply(lambda x: mean_value if (x < lower_bound or x > upper_bound)
            else x)
    else:
        # Quantile Based Flooring and Capping
        df[var] = np.where(df[var] < lower_bound, lower_bound, df[var])
        df[var] = np.where(df[var] > upper_bound, upper_bound, df[var])

    print(f'Variable {var} has {percent_outliers:.2f}% outlier.')
```

Listing 1: Phần xử lý outlier

Kết quả:

Kết quả phân tích outlier

```
Biến BMI có 2.45% outlier.
Biến GenHlth có 5.26% outlier.
Biến MentHlth có 15.74% outlier.
Biến PhysHlth có 14.95% outlier.
Biến Age có 0.00% outlier.
Biến Education có 0.00% outlier.
Biến Income có 0.00% outlier.
```

Từ kết quả, chúng ta có thể chia tỉ lệ ngoại lai của các biến độc lập thành 2 nhóm: **lớn hơn hoặc bằng 5 %** và **nhỏ hơn 5 %**.

Do đó, chúng ta sẽ tập trung xử lý cho các biến độc lập sau: BMI (**2.45%**), Stroke (**4.48%**), GenHlth (**5.26%**), MenHlth (**15.74%**), và PhyHlth (**14.95%**).

Trong đoạn mã in ra thống kê tỉ lệ ngoại lai của các biến độc lập, chúng ta cũng đã đồng thời xử lý ngoại lai bằng cách:

- Nếu tỉ lệ ngoại lai của biến độc lập lớn hơn hoặc bằng 5% thì sử dụng phương pháp **Quantile method** để đặt giới hạn giá trị tại tứ phân vị thứ nhất và thứ bốn. Với phương pháp này thì các giá trị ngoại lai vượt ngoài khoảng tứ phân vị sẽ được thay bằng giá trị của tứ phân vị gần ngoại lai đó nhất. Điều này giúp các giá trị đặc trưng của tập dữ liệu như **mean**, **các giá trị tứ phân vị**, **độ lệch chuẩn** sẽ không bị thay đổi quá nhiều và giúp giảm sai sót trong dữ liệu.
- Nếu tỉ lệ ngoại lai của biến độc lập nhỏ hơn 5% thì chúng ta chỉ đơn giản là thay thế các giá trị ngoại lai thành giá trị **mean** để bảo toàn các giá trị đặc trưng trong tập dữ liệu.

Sau đây là đoạn mã in ra kết quả:

```
# Check outlier ratio of each variable
for var in variables:
    # Calculate IQR
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    # Count outlier
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    num_outliers = df[(df[var] < lower_bound) | (df[var] > upper_bound)].shape[0]

    # Calculate rate of outlier
    percent_outliers = (num_outliers / df.shape[0]) * 100

    print(f'Variable {var} has {percent_outliers:.2f}% outlier.')
```

Listing 2: Kiểm tra

Kết quả:

Kết quả phân tích outlier

```
Biến BMI có 1.83% outlier.
Biến GenHlth có 0.00% outlier.
Biến MentHlth có 0.00% outlier.
Biến PhysHlth có 0.00% outlier.
Biến Age có 0.00% outlier.
Biến Education có 0.00% outlier.
Biến Income có 0.00% outlier.
```

4 Xây dựng mô hình và dự đoán

4.1 Xử lý dữ liệu mất cân bằng

Trong phần **EDA**, chúng ta thấy rằng tập dữ liệu đang bị mất cân bằng nghiêm trọng - tỉ lệ người được chẩn đoán không bị mắc bệnh tiểu đường chiếm tới hơn **80%**. Do đó, nếu ta phân chia tập dữ liệu ngay để tiến hành huấn luyện mô hình thì có thể dự đoán được rằng, phần lớn kết quả sẽ luôn nằm trong trường hợp **không mắc bệnh tiểu đường**. Điều này gây ảnh hưởng nghiêm trọng đến khả năng học và dự đoán của mô hình. Do đó, chúng ta cần có chiến lược khởi tạo tập dữ liệu. Có nhiều cách để khởi tạo nhưng trong bài báo cáo này, chúng ta sẽ sử dụng 4 phương pháp thường dùng nhất:

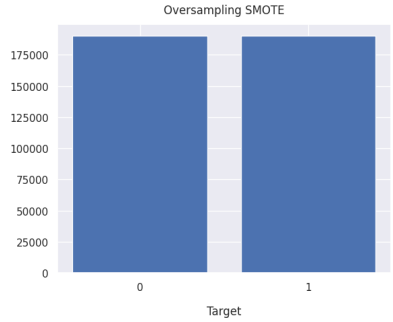
1. Giữ nguyên tập dữ liệu thô và xáo trộn ngẫu nhiên.
2. Sử dụng phương pháp **RUS**, trong đó, chúng ta lọc ra phần dữ liệu đang là thiểu số rồi lấy ngẫu nhiên dữ liệu trong tập dữ liệu chiếm đa số để được tập dữ liệu huấn luyện mới có lượng dữ liệu là bằng nhau cho mỗi loại kết quả của dữ liệu.
3. Sử dụng phương pháp **SMOTE**, trong đó, chúng ta nhân bản số lượng mẫu của tập dữ liệu chiếm thiểu số để có số lượng đúng bằng lượng dữ liệu của tập chiếm đa số.
4. Sử dụng phương pháp **SMOTEENN**, là sự kết hợp của cả 2 kỹ thuật **EEN** và **SMOTE**.

- Phương pháp xử lý **SMOTE**:

```
        # Declare feature vector and target variable
        X = df.drop(['Diabetes'], axis=1).values
        y = df['Diabetes']

        # SMOTE
        smote = SMOTE(random_state=42)
        X_resampled_smote, y_resampled_smote = SMOTE().fit_resample(X, y)

        # Oversampling SMOTE
        sns.set(font_scale=1.0)
        smote_counts = y_resampled_smote.value_counts()

        plt.bar(smote_counts.index, smote_counts.values)
        plt.xlabel("Target", labelpad=14)
        plt.xticks([0, 1])
        plt.title("Oversampling SMOTE", y=1.02)
        plt.show()

        smote_counts
```

Kết quả phân tích outlier

```
Diabetes
0    190055
1    190055
Name: count, dtype: int64
```



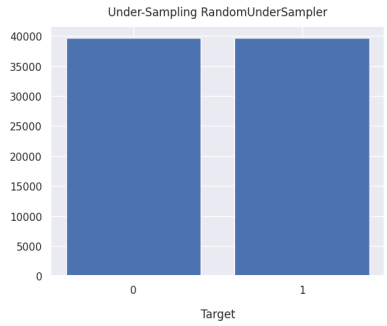
Hình 1: SMOTE

- Phương pháp xử lý **RUS**:

```
        # RandomUnderSampler
        rus = RandomUnderSampler(random_state=42)
        X_resampled_rus, y_resampled_rus = rus.fit_resample(X, y)

        # Under-Sampling RandomUnderSampler
        sns.set(font_scale=1.0)
        rus_counts = y_resampled_rus.value_counts()

        plt.bar(rus_counts.index, rus_counts.values)
        plt.xlabel("Target", labelpad=14)
        plt.xticks([0, 1])
        plt.title("Under-Sampling RandomUnderSampler", y=1.02)
        plt.show()

        rus_counts
```

Kết quả phân tích outlier

```
Diabetes
0    39657
1    39657
Name: count, dtype: int64
```



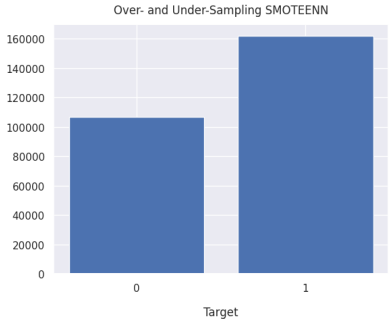
Hình 1: RUS

- Phương pháp xử lý **SMOTEENN**:

```
        # SMOTEENN
        smoteenn = SMOTEENN(random_state=42, n_jobs=-1)
        X_resampled_smoteenn, y_resampled_smoteenn = smoteenn.fit_resample(X, y)
        # Over- and Under-Sampling SMOTEENN
        sns.set(font_scale=1.0)
        smoteenn_counts = y_resampled_smoteenn.value_counts()

        plt.bar(smoteenn_counts.index, smoteenn_counts.values)
        plt.xlabel("Target", labelpad=14)
        plt.xticks([0, 1])
        plt.title("Over- and Under-Sampling SMOTEENN", y=1.02)
        plt.show()

        smoteenn_counts
```

Kết quả phân tích outlier

```
Diabetes
1    161765
0    106658
Name: count, dtype: int64
```



Hình 1: SMOTEEN

Tạo tập huấn luyện và kiểm tra, chúng tôi thống nhất tỉ lệ tập dữ liệu huấn luyện và kiểm tra là **70%** và **30%**:

```
def create_train_test_data(X_resampled, y_resampled, sampling_method):
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
        test_size=0.3, random_state=42)
    print("\nData", sampling_method)
    print("Train data size:", len(X_train))
    print("Test data size:", len(X_test))
    return X_train, X_test, y_train, y_test

# Raw Data
X_train, X_test, y_train, y_test = create_train_test_data(X, y, "Raw")

# Oversampling SMOTE
X_train_smote, X_test_smote, y_train_smote, y_test_smote =
    create_train_test_data(X_resampled_smote, y_resampled_smote, "Oversampling SMOTE")

# Under-Sampling RandomUnderSampler
X_train_rus, X_test_rus, y_train_rus, y_test_rus = create_train_test_data(X_resampled_rus,
    y_resampled_rus, "Under-Sampling RandomUnderSampler")

# Over- and Under-Sampling SMOTEENN
X_train_smoteenn, X_test_smoteenn, y_train_smoteenn, y_test_smoteenn =
    create_train_test_data(X_resampled_smoteenn, y_resampled_smoteenn, "Over- and
    Under-Sampling SMOTEENN")
```

Kết quả khởi tạo tập dữ liệu

```
    Data Raw
Train data size: 160798
Test data size: 68914

Data Oversampling SMOTE
Train data size: 266077
Test data size: 114033

Data Under-Sampling RandomUnderSampler
Train data size: 55519
Test data size: 23795

Data Over- and Under-Sampling SMOTEENN
Train data size: 187491
Test data size: 80354
```

Xây dựng hàm cho việc in ra các thông số và ma trận nhầm lẫn (**Confusion Matrix**):

```
    def evaluate_model(model, X_train, y_train, X_test, y_test, title):
    print("\n" + title + " Model:")

    # Predictions on Training Set
    y_train_pred = model.predict(X_train)

    # Classification Report for Training Set
    print("\nTraining Set:")
```

```
    print(classification_report(y_train, y_train_pred))

    # Accuracy for Training Set
    accuracy_train = accuracy_score(y_train, y_train_pred)
    print("\nTraining Set Accuracy:", accuracy_train)

    # Predictions on Test Set
    y_test_pred = model.predict(X_test)

    # Classification Report for Test Set
    print("\nTest Set:")
    print(classification_report(y_test, y_test_pred))

    # Accuracy for Test Set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    print("\nTest Set Accuracy:", accuracy_test)

    # Cohen's Kappa
    kappa = cohen_kappa_score(y_test, y_test_pred)
    print("\nCohen's Kappa:", kappa)

    # MSE & RMSE
    mse = mean_squared_error(y_test, y_test_pred)
    print('Mean Squared Error : {:.4f}'.format(mse))
    rmse = math.sqrt(mse)
    print('Root Mean Squared Error : {:.4f}'.format(rmse))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', cbar=False)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title('Confusion Matrix - ' + title)
    plt.show()
```

4.2 Naive Bayes

- Dựa vào dữ liệu sau khi được xử lý, ta khởi tạo các mô hình Naive Bayes tương ứng:

```
from sklearn.naive_bayes import GaussianNB
# Train Naive Bayes model on raw data
nb_model = GaussianNB()
nb_model.fit(X_train_nb, y_train_nb)
```

```
# Train Naive Bayes model on oversampled SMOTE data
nb_model_smote = GaussianNB()
nb_model_smote.fit(X_train_smote_nb, y_train_smote_nb)
```

```
# Train Naive Bayes model on undersampled RandomUnderSampler data
nb_model_rus = GaussianNB()
nb_model_rus.fit(X_train_rus_nb, y_train_rus_nb)
```

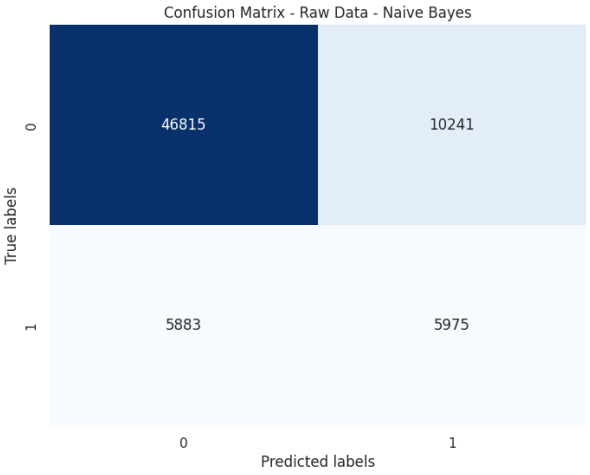
```
# Train Naive Bayes model on over and under-sampled SMOTEENN data
nb_model_smoteenn = GaussianNB()
nb_model_smoteenn.fit(X_train_smoteenn_nb, y_train_smoteenn_nb)
```

- Ở 4 đoạn mã trên, mỗi đoạn có 2 dòng:
  - Dòng 1: Khởi tạo mô hình Gauss Naive Bayes.
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.
- 4 đoạn mã tạo và huấn luyện 4 mô hình Gauss Naive Bayes tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMOTEEN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```
# Evaluate Naive Bayes models
evaluate_model(nb_model, X_train_nb, y_train_nb, X_test_nb, y_test_nb, 'Raw Data - Naive
    Bayes')
evaluate_model(nb_model_smote, X_train_smote_nb, y_train_smote_nb, X_test_smote_nb,
    y_test_smote_nb, 'Over-Sampling SMOTE - Naive Bayes')
evaluate_model(nb_model_rus, X_train_rus_nb, y_train_rus_nb, X_test_rus_nb, y_test_rus_nb,
    'Under-Sampling RandomUnderSampler - Naive Bayes')
evaluate_model(nb_model_smoteenn, X_train_smoteenn_nb, y_train_smoteenn_nb,
    X_test_smoteenn_nb, y_test_smoteenn_nb, 'Over and Under-Sampling SMOTEENN - Naive Bayes')
```

Raw data

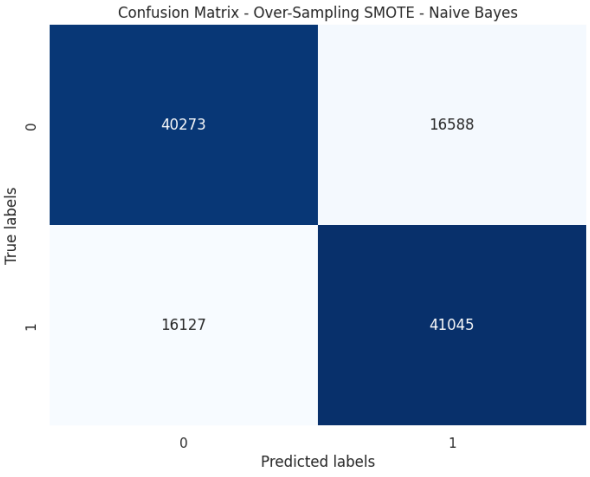
```
           Raw Data - Naive Bayes Model:
Training Set:
              precision    recall  f1-score   support
           0       0.89      0.82      0.85    132999
           1       0.38      0.52      0.44     27799
    accuracy                           0.77    160798
   macro avg       0.63      0.67      0.65    160798
weighted avg       0.80      0.77      0.78    160798

Training Set Accuracy: 0.7687222477891517

Test Set:
              precision    recall  f1-score   support
           0       0.89      0.82      0.85     57056
           1       0.37      0.50      0.43     11858
    accuracy                           0.77     68914
   macro avg       0.63      0.66      0.64     68914
weighted avg       0.80      0.77      0.78     68914

Test Set Accuracy: 0.7660272223350844

Cohen's Kappa: 0.28316880747621376
Mean Squared Error : 0.2340
Root Mean Squared Error : 0.4837
```



Hình 1: Raw Data

SMOTE

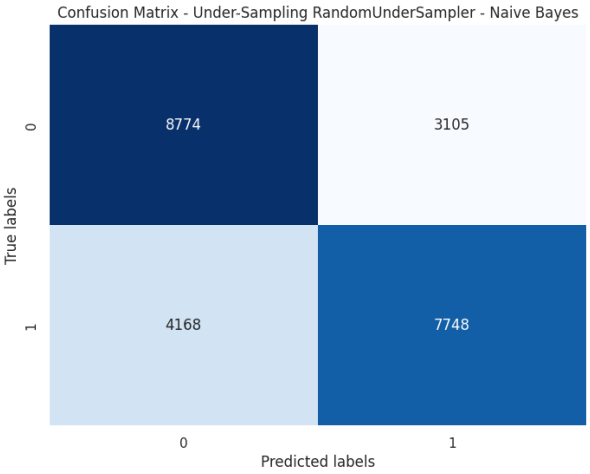
```
           Over-Sampling SMOTE - Naive Bayes Model:
Training Set:
              precision    recall  f1-score   support
           0       0.72      0.71      0.71    133194
           1       0.71      0.72      0.71    132883
    accuracy                           0.71    266077
   macro avg       0.71      0.71      0.71    266077
weighted avg       0.71      0.71      0.71    266077

Training Set Accuracy: 0.7135904268313308

Test Set:
              precision    recall  f1-score   support
           0       0.71      0.71      0.71     56861
           1       0.71      0.72      0.72     57172
    accuracy                           0.71    114033
   macro avg       0.71      0.71      0.71    114033
weighted avg       0.71      0.71      0.71    114033

Test Set Accuracy: 0.7131093630791087

Cohen's Kappa: 0.42620180533285534
Mean Squared Error : 0.2869
Root Mean Squared Error : 0.5356
```



Hình 1: SMOTE

RUS

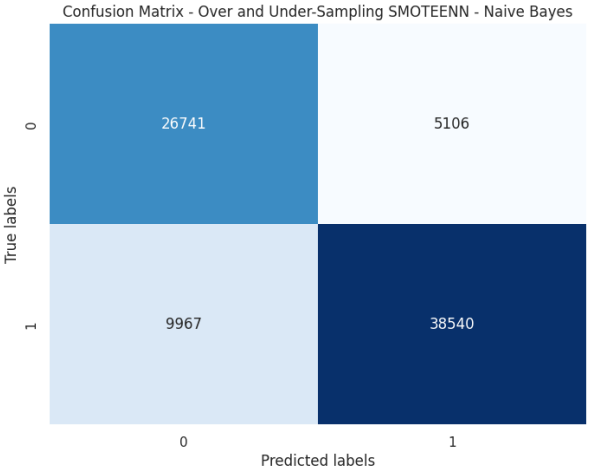
```
           Under-Sampling RandomUnderSampler
           Naive Bayes Model:
Training Set:
              precision    recall  f1-score   support

           0       0.68      0.74      0.71     27778
           1       0.72      0.65      0.68     27741

    accuracy                           0.70     55519
   macro avg       0.70      0.70      0.70     55519
weighted avg       0.70      0.70      0.70     55519

Training Set Accuracy: 0.6963922260847638

Test Set:
              precision    recall  f1-score   support

           0       0.68      0.74      0.71     11879
           1       0.71      0.65      0.68     11916

    accuracy                           0.69     23795
   macro avg       0.70      0.69      0.69     23795
weighted avg       0.70      0.69      0.69     23795

Test Set Accuracy: 0.6943475520067242

Cohen's Kappa: 0.38877854267670775
Mean Squared Error : 0.3057
Root Mean Squared Error : 0.5529
```



Hình 1: RUS

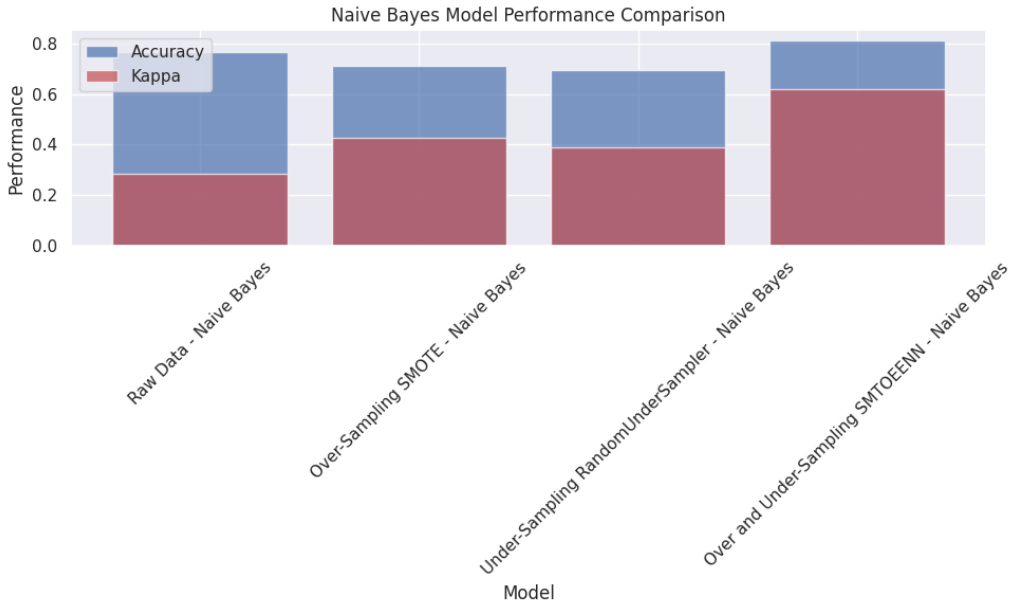
SMOTEEN

```
           Over and Under-Sampling
           SMOTEENN - Naive Bayes Model:
Training Set:
              precision    recall  f1-score   support

           0       0.73      0.84      0.78     74992
           1       0.88      0.79      0.83    112499

    accuracy                           0.81    187491
   macro avg       0.81      0.82      0.81    187491
weighted avg       0.82      0.81      0.81    187491

Training Set Accuracy: 0.8111269340928364

Test Set:
              precision    recall  f1-score   support

           0       0.73      0.84      0.78     31847
           1       0.88      0.79      0.84     48507

    accuracy                           0.81     80354
   macro avg       0.81      0.82      0.81     80354
weighted avg       0.82      0.81      0.81     80354

Test Set Accuracy: 0.8124175523309356

Cohen's Kappa: 0.6179965940323398
Mean Squared Error : 0.1876
Root Mean Squared Error : 0.4331
```

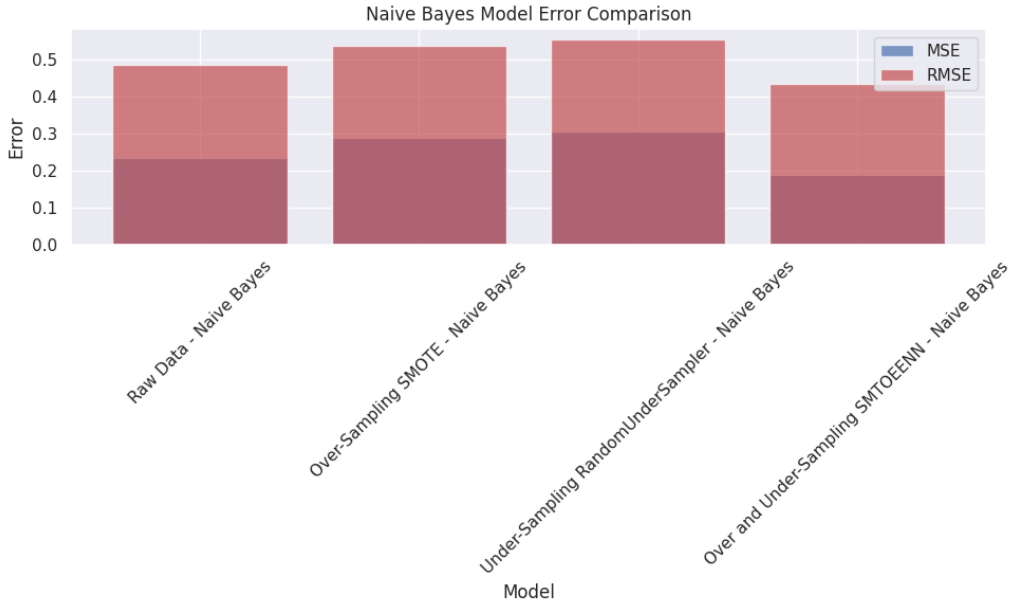


Hình 1: SMOTEENN

Biểu đồ kết quả huấn luyện của các tập dữ liệu:



Hình 8: So sánh hiệu suất mô hình trên các tập dữ liệu



Hình 9: So sánh hiệu suất mô hình trên các tập dữ liệu

Comparison

```
                                                Model  Accuracy     Kappa
0                             Raw Data - Naive Bayes  0.766027  0.283169
1                  Over-Sampling SMOTE - Naive Bayes  0.713109  0.426202
2  Under-Sampling RandomUnderSampler - Naive Bayes  0.694348  0.388779
3   Over and Under-Sampling SMOTEENN - Naive Bayes  0.812418  0.617997

        MSE      RMSE
0  0.233973  0.483707
1  0.286891  0.535622
2  0.305652  0.552858
3  0.187582  0.433108
```

4.3 Logistic Regression

- Dựa vào dữ liệu sau khi được xử lý, ta khởi tạo những mô hình Hồi quy Logistic tương ứng:

```
1 lg = LogisticRegression(max_iter = 1500)
2 lg.fit(X_train, y_train)
```

```
1 lg_smote = LogisticRegression(max_iter = 1500)
2 lg_smote.fit(X_train_smote, y_train_smote)
```

```
1 lg_rus = LogisticRegression(max_iter = 1500)
2 lg_rus.fit(X_train_rus, y_train_rus)
```

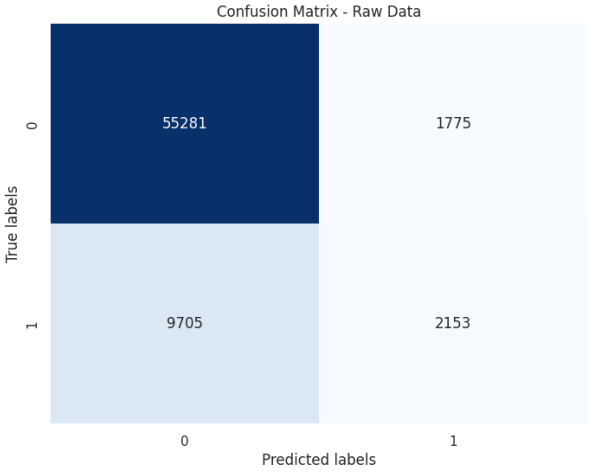
```
1 lg_smoteenn = LogisticRegression(max_iter = 1500)
2 lg_smoteenn.fit(X_train_smoteenn, y_train_smoteenn)
```

- Ở 4 khung code trên, mỗi khung có 2 dòng:
  - Dòng 1: Khởi tạo mô hình Hồi quy Logistic với max_iter được đặt là 1500. max_iter là số lần lặp tối đa mà giải thuật sẽ thực hiện để tối ưu hóa mô hình.
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.
- 4 khung code tạo và huấn luyện 4 mô hình Hồi quy Logistic tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMOTEENN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```python
# Raw Data
evaluate_model(lg, X_train, y_train, X_test, y_test, 'Raw Data')

# Over-Sampling SMOTE
evaluate_model(lg_smote, X_train_smote, y_train_smote, X_test_smote, y_test_smote,
    'Over-Sampling SMOTE')

# Under-Sampling RandomUnderSampler
evaluate_model(lg_rus, X_train_rus, y_train_rus, X_test_rus, y_test_rus, 'Under-Sampling
    RandomUnderSampler')

# Over and Under-Sampling SMTOEENN
evaluate_model(lg_smoteenn, X_train_smoteenn, y_train_smoteenn, X_test_smoteenn,
    y_test_smoteenn, 'Over and Under-Sampling SMTOEENN')
```

Raw data

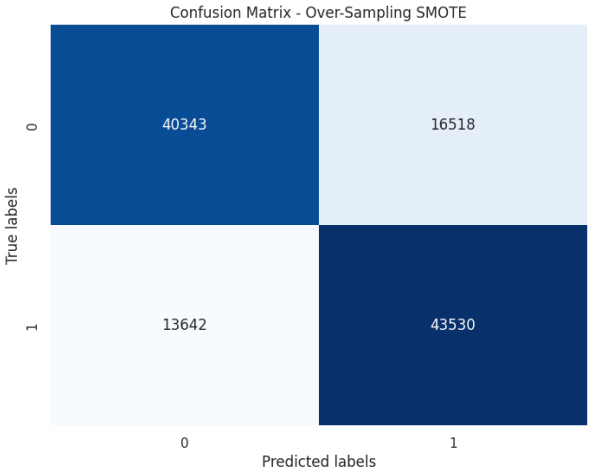
```
            Raw Data Model:

Training Set:
              precision    recall  f1-score   support

           0       0.85      0.97      0.91    132999
           1       0.55      0.19      0.28     27799

    accuracy                           0.83    160798
   macro avg       0.70      0.58      0.59    160798
weighted avg       0.80      0.83      0.80    160798

Training Set Accuracy: 0.8335302677894003

Test Set:
              precision    recall  f1-score   support

           0       0.85      0.97      0.91     57056
           1       0.55      0.18      0.27     11858

    accuracy                           0.83     68914
   macro avg       0.70      0.58      0.59     68914
weighted avg       0.80      0.83      0.80     68914

Test Set Accuracy: 0.833415561424384

Cohen's Kappa: 0.2046678914723613
Mean Squared Error : 0.1666
Root Mean Squared Error : 0.4081
```



Hình 1: Raw data

Over-Sampling SMOTE

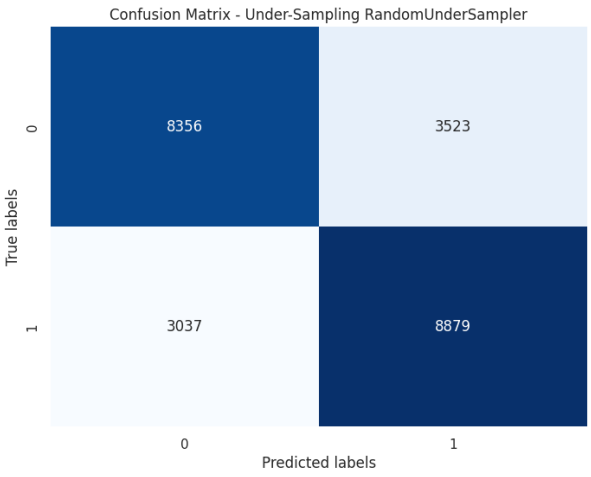
```
            Over-Sampling SMOTE Model:

Training Set:
              precision    recall  f1-score   support

           0       0.75      0.71      0.73    133194
           1       0.72      0.76      0.74    132883

    accuracy                           0.74    266077
   macro avg       0.74      0.74      0.74    266077
weighted avg       0.74      0.74      0.74    266077

Training Set Accuracy: 0.7357569425391898

Test Set:
              precision    recall  f1-score   support

           0       0.75      0.71      0.73     56861
           1       0.72      0.76      0.74     57172

    accuracy                           0.74    114033
   macro avg       0.74      0.74      0.74    114033
weighted avg       0.74      0.74      0.74    114033

Test Set Accuracy: 0.7355151578928907

Cohen's Kappa: 0.47095360072734016
Mean Squared Error : 0.2645
Root Mean Squared Error : 0.5143
```



Hình 1: Over-Sampling SMOTE

Under-Sampling RandomUnderSampler

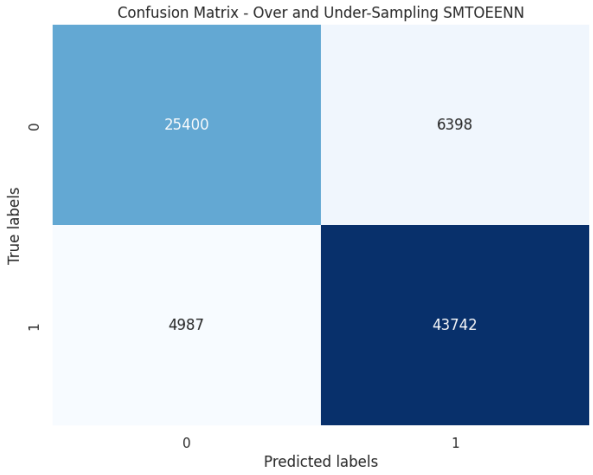
```
            Under-Sampling RandomUnderSampler Model:

Training Set:
              precision    recall  f1-score   support

           0       0.74      0.71      0.72     27778
           1       0.72      0.75      0.73     27741

    accuracy                           0.73     55519
   macro avg       0.73      0.73      0.73     55519
weighted avg       0.73      0.73      0.73     55519

Training Set Accuracy: 0.7284713341378627

Test Set:
              precision    recall  f1-score   support

           0       0.73      0.70      0.72     11879
           1       0.72      0.75      0.73     11916

    accuracy                           0.72     23795
   macro avg       0.72      0.72      0.72     23795
weighted avg       0.72      0.72      0.72     23795

Test Set Accuracy: 0.724311830216432

Cohen's Kappa: 0.448587302579441
Mean Squared Error : 0.2757
Root Mean Squared Error : 0.5251
```



Hình 1: Under-Sampling RandomUnderSampler

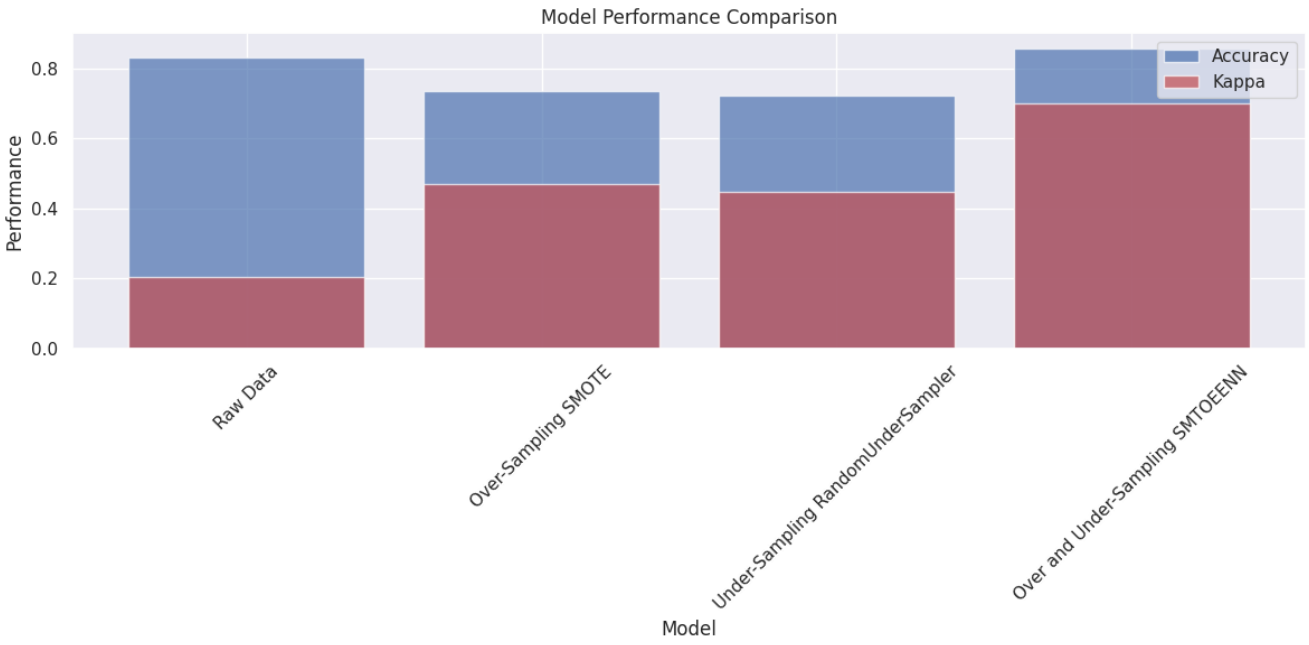
Over and Under-Sampling SMTOEENN

```
            Over and Under-Sampling SMTOEENN Model:

Training Set:
              precision    recall  f1-score   support

           0       0.84      0.80      0.82     74860
           1       0.87      0.90      0.88    113036

    accuracy                           0.86    187896
   macro avg       0.85      0.85      0.85    187896
weighted avg       0.86      0.86      0.86    187896

Training Set Accuracy: 0.8588793800825989

Test Set:
              precision    recall  f1-score   support

           0       0.84      0.80      0.82     31798
           1       0.87      0.90      0.88     48729

    accuracy                           0.86     80527
   macro avg       0.85      0.85      0.85     80527
weighted avg       0.86      0.86      0.86     80527

Test Set Accuracy: 0.8586188483365828

Cohen's Kappa: 0.7018614339113927
Mean Squared Error : 0.1414
Root Mean Squared Error : 0.3760
```

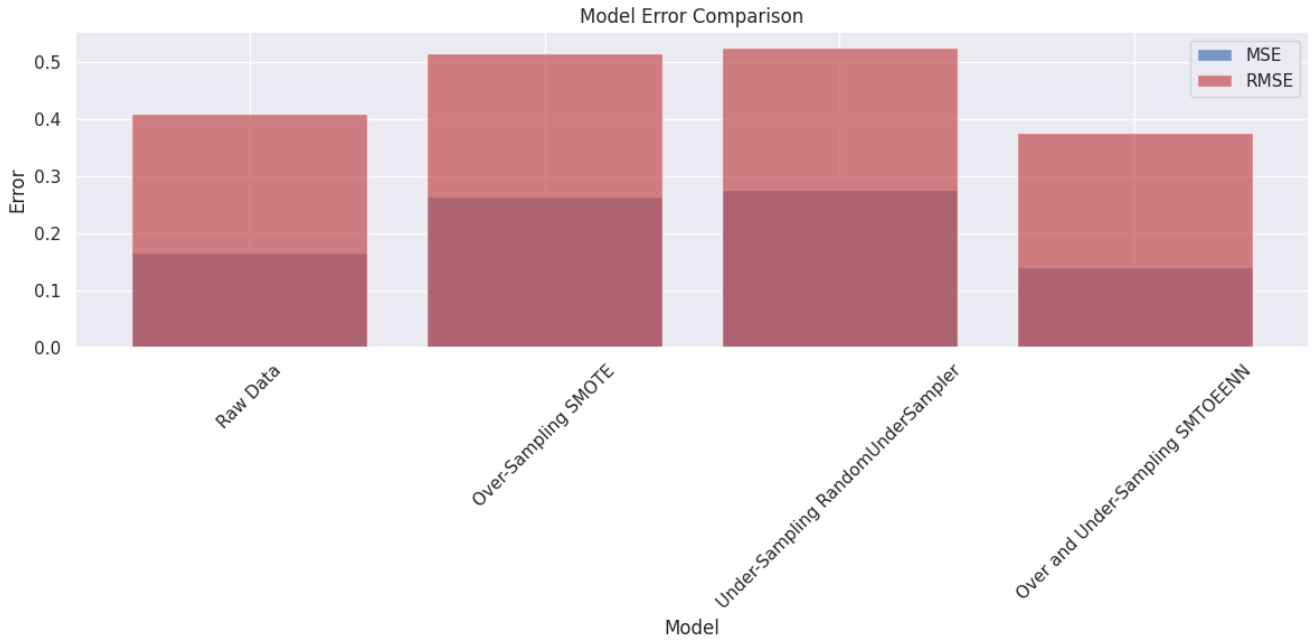


Hình 1: Over and Under-Sampling SMTOEENN

Biểu đồ trực quan cho kết quả huấn luyện:



Hình 10: So sánh hiệu suất mô hình trên các tập dữ liệu



Hình 11: So sánh hiệu suất mô hình trên các tập dữ liệu

Comparison

Các chỉ số hiệu suất cho các mô hình khác nhau:

	Model	Accuracy	Kappa	MSE	RMSE
0	Raw Data	0.833416	0.204668	0.166584	0.408148
1	Over-Sampling SMOTE	0.735515	0.470954	0.264485	0.514281
2	Under-Sampling RandomUnderSampler	0.724312	0.448587	0.275688	0.525060
3	Over and Under-Sampling SMTOEENN	0.858619	0.701861	0.141381	0.376007

4.4 SVM

- Dựa vào dữ liệu sau khi được xử lý, ta khởi tạo những mô hình SVM tương ứng:

```
svm = SGDClassifier(loss='hinge', alpha=0.001, max_iter=7000, tol=1e-5)
svm.fit(X_train, y_train)
```

```
svm_smote = SGDClassifier(loss='hinge', alpha=0.001, max_iter=7000, tol=1e-5)
svm_smote.fit(X_train_smote, y_train_smote)
```

```
svm_rus = SGDClassifier(loss='hinge', alpha=0.001, max_iter=7000, tol=1e-5)
svm_rus.fit(X_train_rus, y_train_rus)
```

```
svm_smoteenn = SGDClassifier(loss='hinge', alpha=0.001, max_iter=7000, tol=1e-5)
svm_smoteenn.fit(X_train_smoteenn, y_train_smoteenn)
```

- Ở 4 khung code trên, mỗi khung có 2 dòng:
  - Dòng 1: Khởi tạo mô hình SVM với tham số loss xác định hàm mất mát được sử dụng trong quá trình huấn luyện, alpha là hệ số điều chuẩn để kiểm soát overfitting, max_iter là số lượng vòng lặp tối đa mà thuật toán sẽ thực hiện, tol là ngưỡng tiêu chuẩn để xác định khi nào thuật toán được coi là hội tụ.

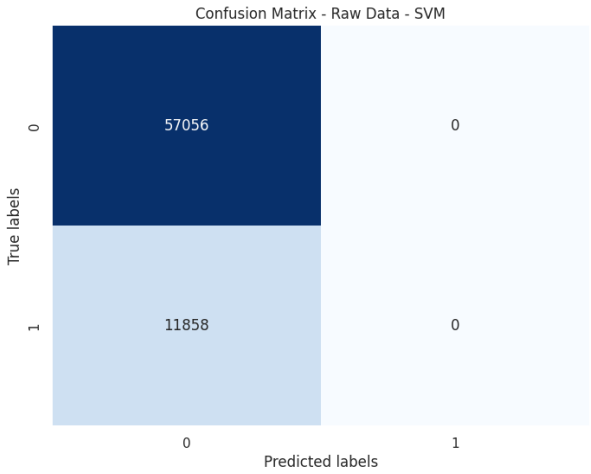
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.

- 4 khung code tạo và huấn luyện 4 mô hình SVM tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMTOEENN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```
# Raw Data
evaluate_model(svm, X_train, y_train, X_test, y_test, 'Raw Data - SVM')
# Over-Sampling SMOTE
evaluate_model(svm_smote, X_train_smote, y_train_smote, X_test_smote, y_test_smote,
    'Over-Sampling SMOTE - SVM')
# Under-Sampling RandomUnderSampler
evaluate_model(svm_rus, X_train_rus, y_train_rus, X_test_rus, y_test_rus, 'Under-Sampling
    RandomUnderSampler - SVM')
# Over and Under-Sampling SMTOEENN
evaluate_model(svm_smoteenn, X_train_smoteenn, y_train_smoteenn, X_test_smoteenn,
    y_test_smoteenn, 'Over and Under-Sampling SMOTEENN - SVM')
```

Raw data

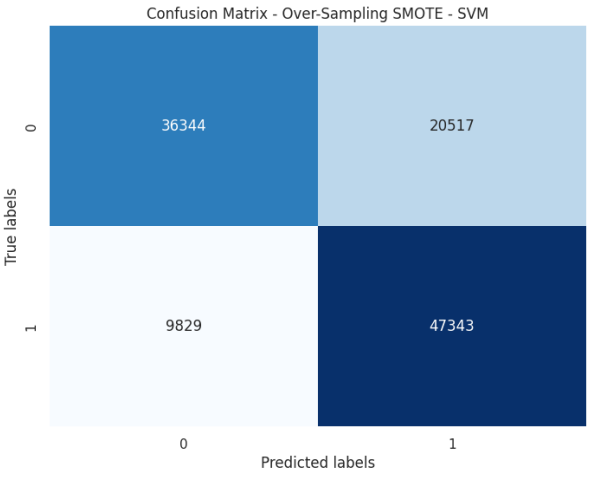
```
            Raw Data - SVM Model:

Training Set:
              precision    recall  f1-score   support

           0       0.83      1.00      0.91    132999
           1       0.00      0.00      0.00     27799

    accuracy                           0.83    160798
   macro avg       0.41      0.50      0.45    160798
weighted avg       0.68      0.83      0.75    160798


Training Set Accuracy: 0.8271184964987127

Test Set:
              precision    recall  f1-score   support

           0       0.83      1.00      0.91     57056
           1       0.00      0.00      0.00     11858

    accuracy                           0.83     68914
   macro avg       0.41      0.50      0.45     68914
weighted avg       0.69      0.83      0.75     68914


Test Set Accuracy: 0.8279304640566503

Cohen's Kappa: 0.0
Mean Squared Error : 0.1721
Root Mean Squared Error : 0.4148
```



Hình 1: Raw data

SMOTE

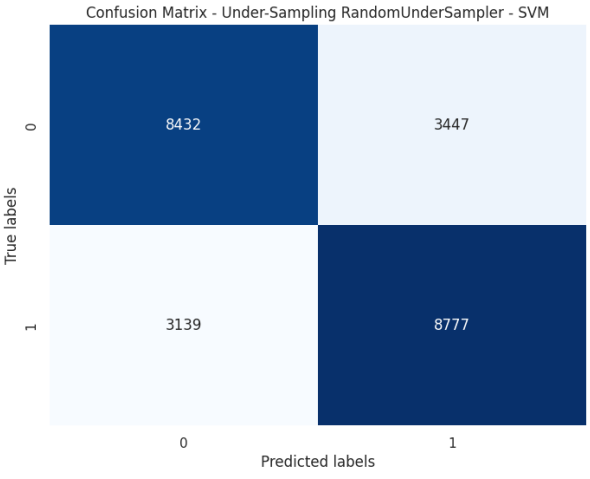
```
            Over-Sampling SMOTE - SVM Model:

Training Set:
              precision    recall  f1-score   support

           0       0.72      0.75      0.74    133194
           1       0.74      0.71      0.72    132883

    accuracy                           0.73    266077
   macro avg       0.73      0.73      0.73    266077
weighted avg       0.73      0.73      0.73    266077


Training Set Accuracy: 0.730901205290198

Test Set:
              precision    recall  f1-score   support

           0       0.72      0.75      0.74     56861
           1       0.74      0.71      0.72     57172

    accuracy                           0.73    114033
   macro avg       0.73      0.73      0.73    114033
weighted avg       0.73      0.73      0.73    114033


Test Set Accuracy: 0.7298501311024002

Cohen's Kappa: 0.45976854702127434
Mean Squared Error : 0.2701
Root Mean Squared Error : 0.5198
```



Hình 1: SMOTE

RUS

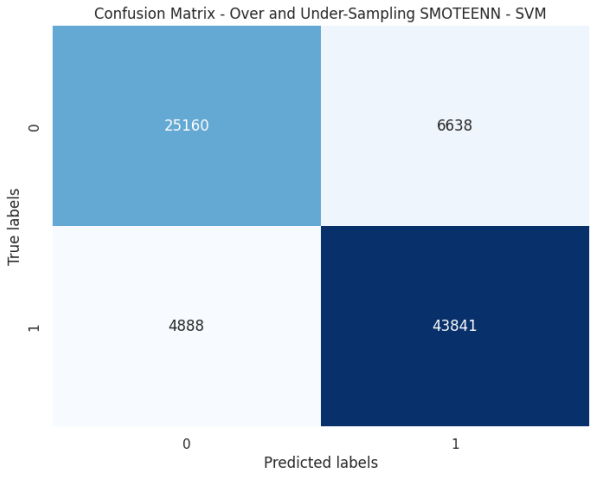
```
Under-Sampling RandomUnderSampler - SVM Model:

Training Set:
              precision    recall  f1-score   support

           0       0.74      0.70      0.72     27778
           1       0.72      0.75      0.73     27741

    accuracy                           0.73     55519
   macro avg       0.73      0.73      0.73     55519
weighted avg       0.73      0.73      0.73     55519


Training Set Accuracy: 0.7271744808083719

Test Set:
              precision    recall  f1-score   support

           0       0.73      0.70      0.72     11879
           1       0.71      0.75      0.73     11916

    accuracy                           0.72     23795
   macro avg       0.72      0.72      0.72     23795
weighted avg       0.72      0.72      0.72     23795


Test Set Accuracy: 0.7231351124185753

Cohen's Kappa: 0.446228355664531
Mean Squared Error : 0.2769
Root Mean Squared Error : 0.5262
```



Hình 1: RUS

SMOTEENN

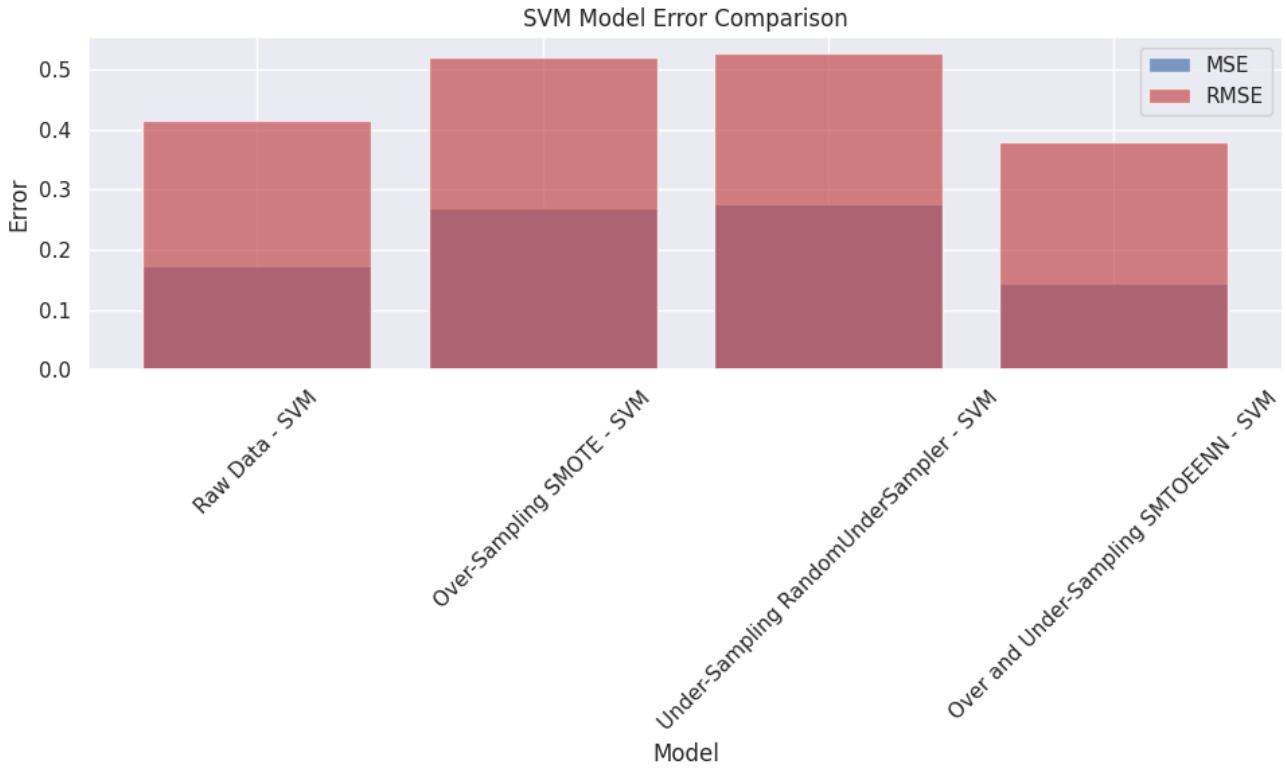
```
Over and Under-Sampling SMOTEENN - SVM Model:

Training Set:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82     74860
           1       0.88      0.88      0.88    113036

    accuracy                           0.86    187896
   macro avg       0.85      0.85      0.85    187896
weighted avg       0.86      0.86      0.86    187896


Training Set Accuracy: 0.8564418614552731

Test Set:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82     31798
           1       0.88      0.88      0.88     48729

    accuracy                           0.86     80527
   macro avg       0.85      0.85      0.85     80527
weighted avg       0.86      0.86      0.86     80527


Test Set Accuracy: 0.856731282675376

Cohen's Kappa: 0.6998705949089551
Mean Squared Error : 0.1433
Root Mean Squared Error : 0.3785
```



Hình 1: SMOTEENN

Biểu đồ trực quan cho kết quả huấn luyện:

Hình 12: So sánh hiệu suất mô hình trên các tập dữ liệu



Hình 13: So sánh hiệu suất mô hình trên các tập dữ liệu

Comparison

	Model	Accuracy	Kappa	MSE	RMSE
0	Raw Data - SVM	0.827930	0.000000	0.172070	0.414813
1	Over-Sampling SMOTE - SVM	0.729850	0.459769	0.270150	0.519759
2	Under-Sampling RandomUnderSampler - SVM	0.723135	0.446228	0.276865	0.526180
3	Over and Under-Sampling SMTOEENN - SVM	0.856731	0.699871	0.143269	0.378509

4.5 Decision Tree

- Dựa vào dữ liệu sau khi được xử lý, ta khởi tạo những mô hình Decision Tree tương ứng:

```
dt = DecisionTreeClassifier(max_depth=18)
dt.fit(X_train, y_train)
```

```
dt_smote = DecisionTreeClassifier(max_depth=18)
dt_smote.fit(X_train_smote, y_train_smote)
```

```
dt_rus = DecisionTreeClassifier(max_depth=18)
dt_rus.fit(X_train_rus, y_train_rus)
```

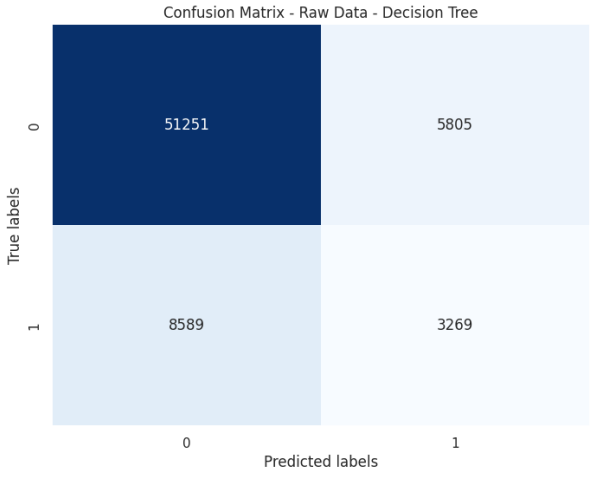
```
dt_smoteenn = DecisionTreeClassifier(max_depth=18)
dt_smoteenn.fit(X_train_smoteenn, y_train_smoteenn)
```

- Ở 4 khung code trên, mỗi khung có 2 dòng:
  - Dòng 1: Khởi tạo mô hình Decision Tree với max_depth được đặt là 18. max_depth quy định số lượng tối đa các nút được phép có trên cây.
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.
- 4 khung code tạo và huấn luyện 4 mô hình Decision Tree tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMOTEENN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```
# Raw Data
evaluate_model(dt, X_train, y_train, X_test, y_test, 'Raw Data - Decision Tree')
# Over-Sampling SMOTE
evaluate_model(dt_smote, X_train_smote, y_train_smote, X_test_smote, y_test_smote,
    'Over-Sampling SMOTE - Decision Tree')
# Under-Sampling RandomUnderSampler
evaluate_model(dt_rus, X_train_rus, y_train_rus, X_test_rus, y_test_rus, 'Under-Sampling
    RandomUnderSampler - Decision Tree')
# Over and Under-Sampling SMTOEENN
evaluate_model(dt_smoteenn, X_train_smoteenn, y_train_smoteenn, X_test_smoteenn,
    y_test_smoteenn, 'Over and Under-Sampling SMOTEENN - Decision Tree')
```

Raw data

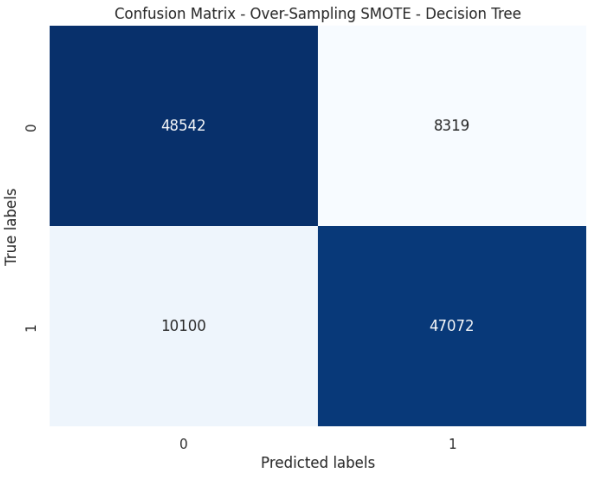
```
            Raw Data - Decision Tree Model:
Training Set:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95    132999
           1       0.87      0.60      0.71     27799
    accuracy                           0.92    160798
   macro avg       0.89      0.79      0.83    160798
weighted avg       0.91      0.92      0.91    160798

Training Set Accuracy: 0.9155959651239444

Test Set:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88     57056
           1       0.36      0.28      0.32     11858
    accuracy                           0.79     68914
   macro avg       0.61      0.59      0.60     68914
weighted avg       0.77      0.79      0.78     68914


Test Set Accuracy: 0.7920451577328265

Cohen's Kappa: 0.19527762341495092
Mean Squared Error : 0.2080
Root Mean Squared Error : 0.4560
```



Hình 1: Raw data

SMOTE

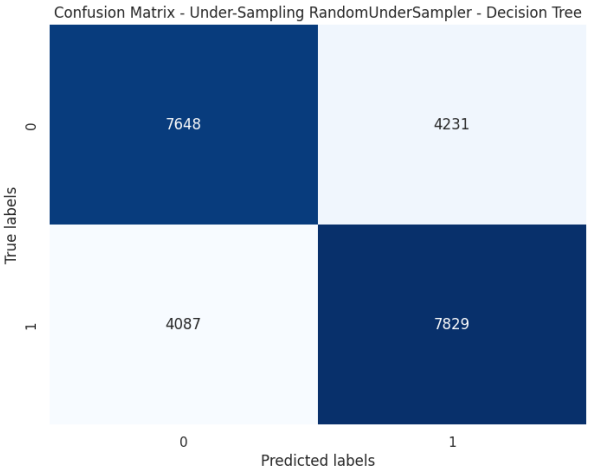
```
            Over-Sampling SMOTE - Decision Tree Model:

Training Set:
              precision    recall  f1-score   support

           0       0.86      0.88      0.87    133194
           1       0.88      0.86      0.87    132883

    accuracy                           0.87    266077
   macro avg       0.87      0.87      0.87    266077
weighted avg       0.87      0.87      0.87    266077


Training Set Accuracy: 0.8718378514490166

Test Set:
              precision    recall  f1-score   support

           0       0.83      0.85      0.84     56861
           1       0.85      0.83      0.84     57172

    accuracy                           0.84    114033
   macro avg       0.84      0.84      0.84    114033
weighted avg       0.84      0.84      0.84    114033


Test Set Accuracy: 0.838537967079705

Cohen's Kappa: 0.6770950326098252
Mean Squared Error : 0.1615
Root Mean Squared Error : 0.4018
```



Hình 1: SMOTE

RUS

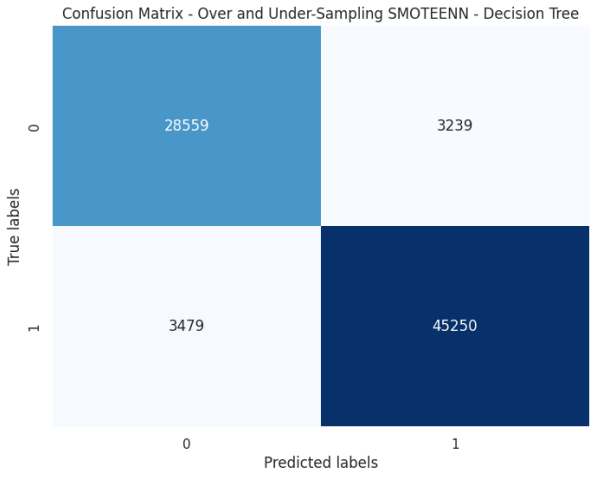
```
Under-Sampling RandomUnderSampler - Decision Tree Model:

Training Set:
              precision    recall  f1-score   support

           0       0.90      0.90      0.90     27778
           1       0.90      0.91      0.90     27741

    accuracy                           0.90     55519
   macro avg       0.90      0.90      0.90     55519
weighted avg       0.90      0.90      0.90     55519


Training Set Accuracy: 0.9019614906608548

Test Set:
              precision    recall  f1-score   support

           0       0.65      0.65      0.65     11879
           1       0.65      0.66      0.65     11916

    accuracy                           0.65     23795
   macro avg       0.65      0.65      0.65     23795
weighted avg       0.65      0.65      0.65     23795


Test Set Accuracy: 0.6533725572599286

Cohen's Kappa: 0.3067394516470757
Mean Squared Error : 0.3466
Root Mean Squared Error : 0.5888
```



Hình 1: RUS

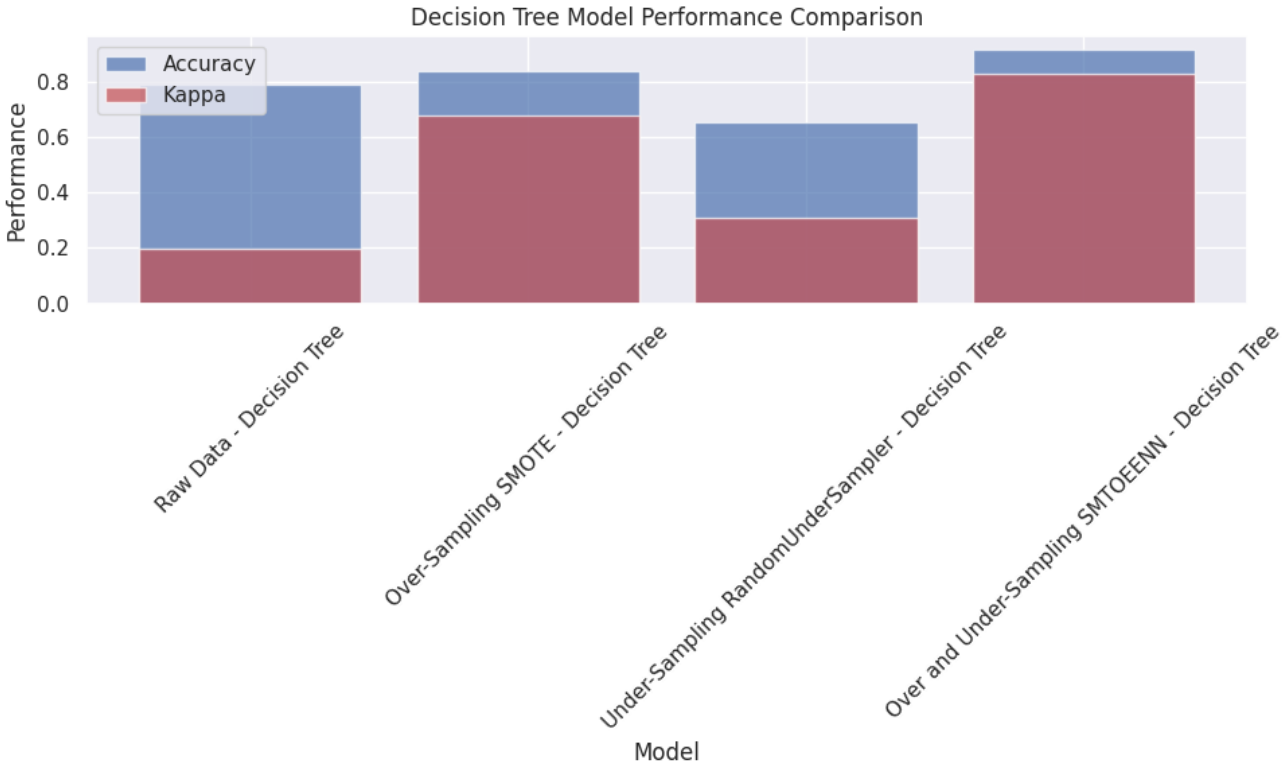
SMOTEENN

```
Over and Under-Sampling SMOTEENN - Decision Tree Model:

Training Set:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     74860
           1       0.97      0.96      0.97    113036

    accuracy                           0.96    187896
   macro avg       0.96      0.96      0.96    187896
weighted avg       0.96      0.96      0.96    187896


Training Set Accuracy: 0.9602918635841103

Test Set:
              precision    recall  f1-score   support

           0       0.89      0.90      0.90     31798
           1       0.93      0.93      0.93     48729

    accuracy                           0.92     80527
   macro avg       0.91      0.91      0.91     80527
weighted avg       0.92      0.92      0.92     80527


Test Set Accuracy: 0.9170837110534354

Cohen's Kappa: 0.8267427200399029
Mean Squared Error : 0.0829
Root Mean Squared Error : 0.2880
```

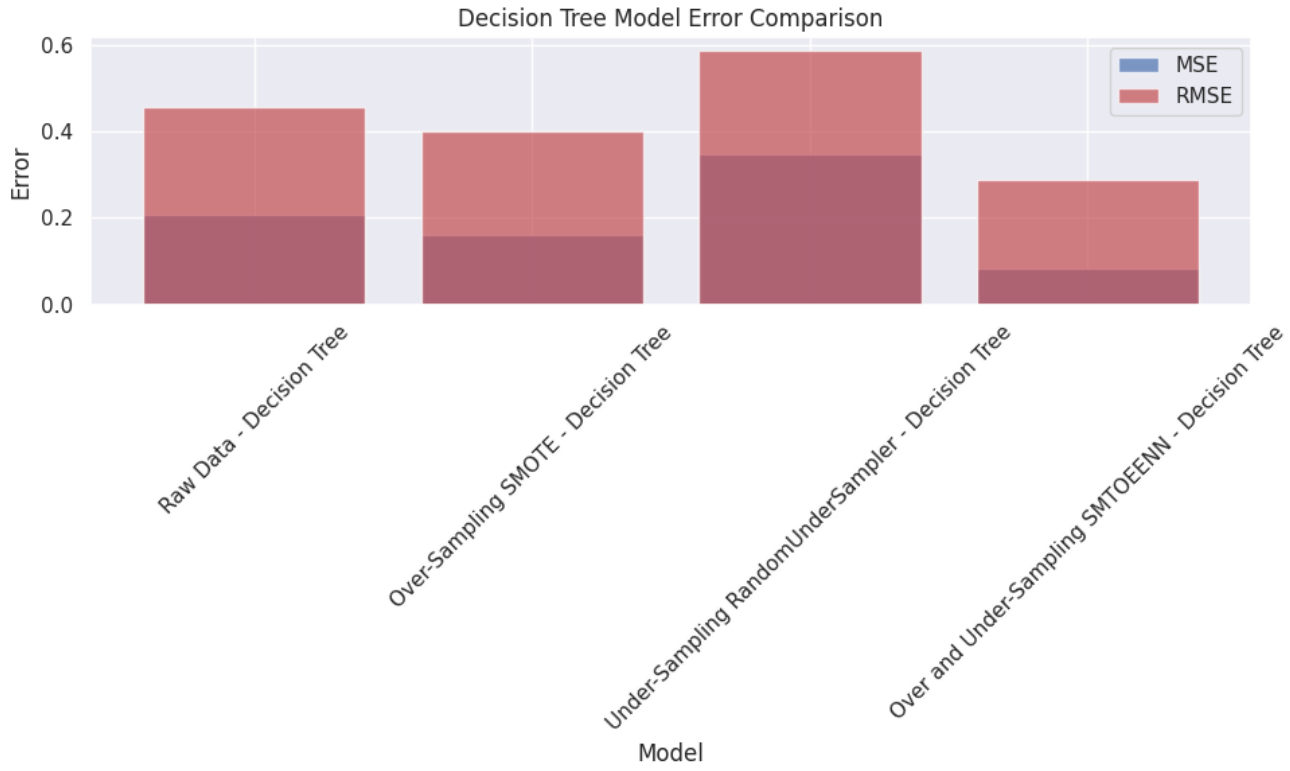


Hình 1: SMOTEENN

Biểu đồ trực quan cho kết quả huấn luyện:



Hình 14: So sánh hiệu suất mô hình trên các tập dữ liệu



Hình 15: So sánh hiệu suất mô hình trên các tập dữ liệu

Comparison

	Model	Accuracy	Kappa	MSE	RMSE
0	Raw Data - Decision Tree	0.792045	0.195278	0.207955	0.456021
1	Over-Sampling SMOTE - Decision Tree	0.838538	0.677095	0.161462	0.401823
2	Under-Sampling RandomUnderSampler - Decision Tree	0.653373	0.306739	0.346627	0.588751
3	Over and Under-Sampling SMTOEENN - Decision Tree	0.917084	0.826743	0.082916	0.287952

4.6 Random Forest

- Dựa vào dữ liệu sau khi được xử lý, ta khởi tạo những mô hình Random Forest tương ứng:

```
rf = RandomForestClassifier(max_depth=12, n_estimators=10, random_state=42)
rf.fit(X_train, y_train)
```

```
rf_smote = RandomForestClassifier(max_depth=12, n_estimators=10, random_state=42)
rf_smote.fit(X_train_smote, y_train_smote)
```

```
rf_rus = RandomForestClassifier(max_depth=12, n_estimators=10, random_state=42)
rf_rus.fit(X_train_rus, y_train_rus)
```

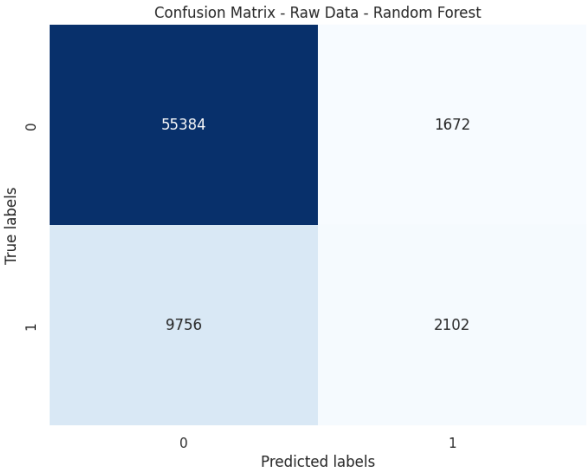
```
rf_smoteenn = RandomForestClassifier(max_depth=12, n_estimators=10, random_state=42)
rf_smoteenn.fit(X_train_smoteenn, y_train_smoteenn)
```

- Ở 4 khung code trên, mỗi khung có 2 dòng:
  - Dòng 1: Khởi tạo mô hình Random Forest với max_depth=12, n_estimators=10, random_state=42. Trong đó max_depth kiểm soát độ sâu tối đa của cây sẽ được tạo, n_estimators xác định số lượng cây được tạo ra trong mô hình, random_state kiểm soát tính ngẫu nhiên của mô hình.
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.
- 4 khung code tạo và huấn luyện 4 mô hình Random Forest tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMTOEENN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```
# Evaluate Random Forest models
evaluate_model(rf, X_train, y_train, X_test, y_test, 'Raw Data - Random Forest')
evaluate_model(rf_smote, X_train_smote, y_train_smote, X_test_smote, y_test_smote,
    'Over-Sampling SMOTE - Random Forest')
evaluate_model(rf_rus, X_train_rus, y_train_rus, X_test_rus, y_test_rus, 'Under-Sampling
    RandomUnderSampler - Random Forest')
evaluate_model(rf_smoteenn, X_train_smoteenn, y_train_smoteenn, X_test_smoteenn,
    y_test_smoteenn, 'Over and Under-Sampling SMOTEENN - Random Forest')
```

Raw data

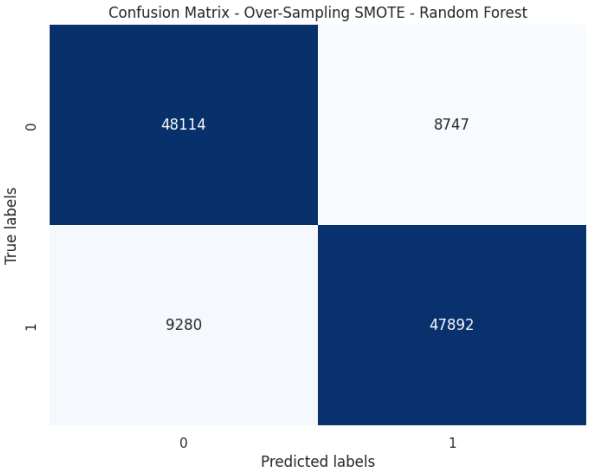
```
            Raw Data - Random Forest Model:

Training Set:
              precision    recall  f1-score   support
           0       0.86      0.98      0.92    132999
           1       0.71      0.24      0.36     27799
    accuracy                           0.85    160798
   macro avg       0.78      0.61      0.64    160798
weighted avg       0.83      0.85      0.82    160798

Training Set Accuracy: 0.8514036244231894

Test Set:
              precision    recall  f1-score   support
           0       0.85      0.97      0.91     57056
           1       0.56      0.18      0.27     11858
    accuracy                           0.83     68914
   macro avg       0.70      0.57      0.59     68914
weighted avg       0.80      0.83      0.80     68914

Test Set Accuracy: 0.8341701250834374

Cohen's Kappa: 0.20269132242908072
Mean Squared Error : 0.1658
Root Mean Squared Error : 0.4072
```



Hình 1: Raw data

SMOTE

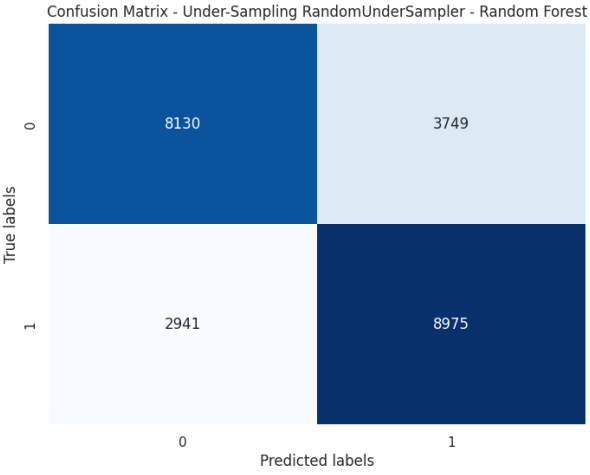
```
            Over-Sampling SMOTE - Random Forest Model:

Training Set:
              precision    recall  f1-score   support
           0       0.85      0.85      0.85    133194
           1       0.85      0.85      0.85    132883
    accuracy                           0.85    266077
   macro avg       0.85      0.85      0.85    266077
weighted avg       0.85      0.85      0.85    266077

Training Set Accuracy: 0.8486340420254287

Test Set:
              precision    recall  f1-score   support
           0       0.84      0.85      0.84     56861
           1       0.85      0.84      0.84     57172
    accuracy                           0.84    114033
   macro avg       0.84      0.84      0.84    114033
weighted avg       0.84      0.84      0.84    114033

Test Set Accuracy: 0.8419141827365763

Cohen's Kappa: 0.6838340744897178
Mean Squared Error : 0.1581
Root Mean Squared Error : 0.3976
```



Hình 1: SMOTE

RUS

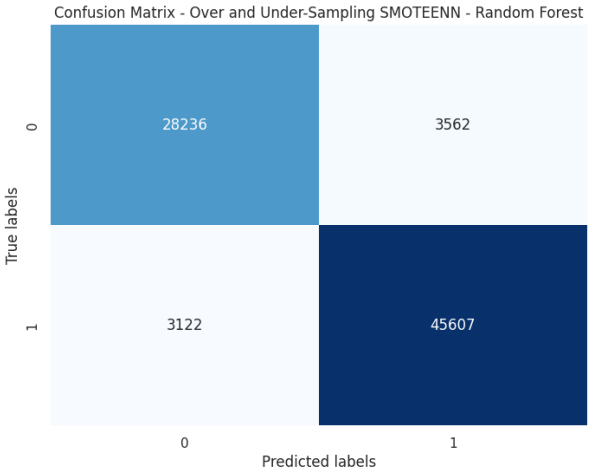
```
Under-Sampling RandomUnderSampler - Random Forest Model:

Training Set:
              precision    recall  f1-score   support
           0       0.80      0.75      0.77     27778
           1       0.76      0.81      0.79     27741
    accuracy                           0.78     55519
   macro avg       0.78      0.78      0.78     55519
weighted avg       0.78      0.78      0.78     55519

Training Set Accuracy: 0.7795709576901602

Test Set:
              precision    recall  f1-score   support
           0       0.73      0.68      0.71     11879
           1       0.71      0.75      0.73     11916
    accuracy                           0.72     23795
   macro avg       0.72      0.72      0.72     23795
weighted avg       0.72      0.72      0.72     23795

Test Set Accuracy: 0.718848497583526

Cohen's Kappa: 0.4376362487764176
Mean Squared Error : 0.2812
Root Mean Squared Error : 0.5302
```



Hình 1: RUS

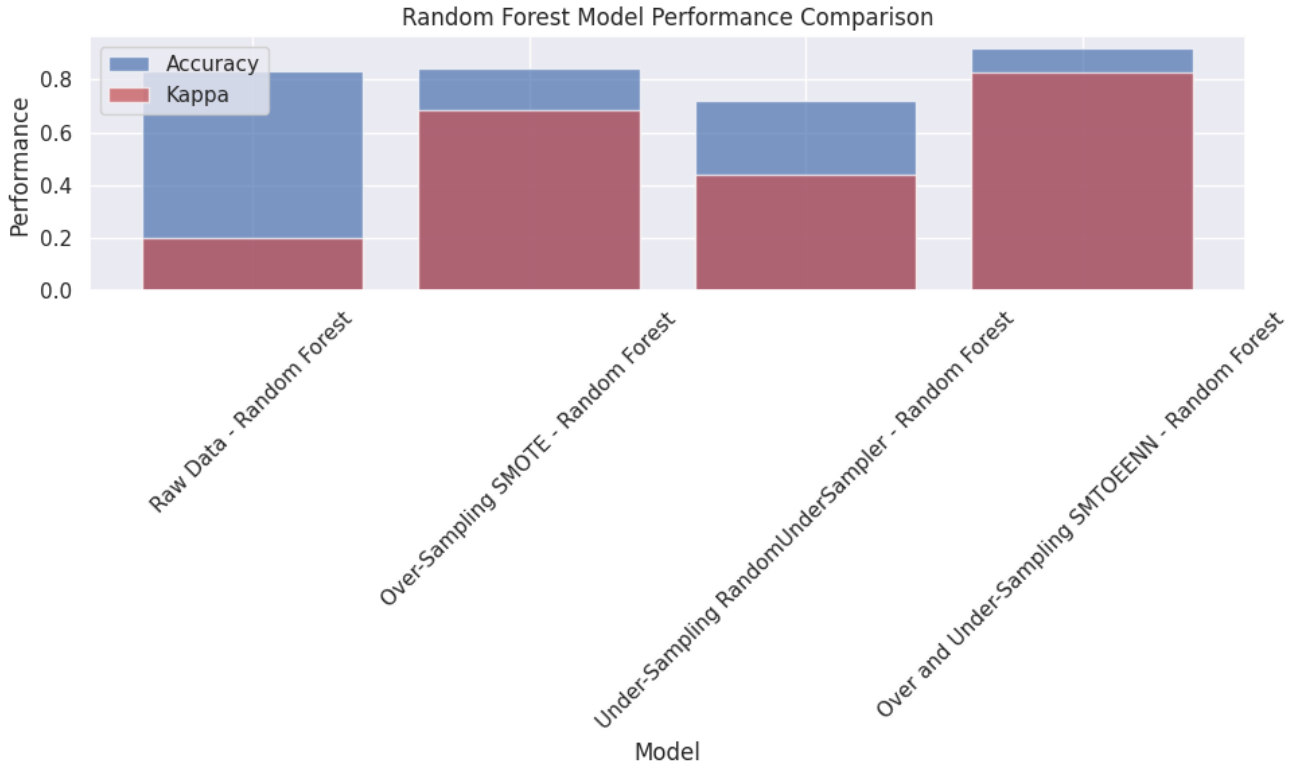
SMOTEENN

```
Over and Under-Sampling SMOTEENN - Random Forest Model:

Training Set:
              precision    recall  f1-score   support
           0       0.91      0.90      0.90     74860
           1       0.93      0.94      0.94    113036
    accuracy                           0.92    187896
   macro avg       0.92      0.92      0.92    187896
weighted avg       0.92      0.92      0.92    187896

Training Set Accuracy: 0.9238940690594797

Test Set:
              precision    recall  f1-score   support
           0       0.90      0.89      0.89     31798
           1       0.93      0.94      0.93     48729
    accuracy                           0.92     80527
   macro avg       0.91      0.91      0.91     80527
weighted avg       0.92      0.92      0.92     80527

Test Set Accuracy: 0.9169967836874588

Cohen's Kappa: 0.8258971330736831
Mean Squared Error : 0.0830
Root Mean Squared Error : 0.2881
```

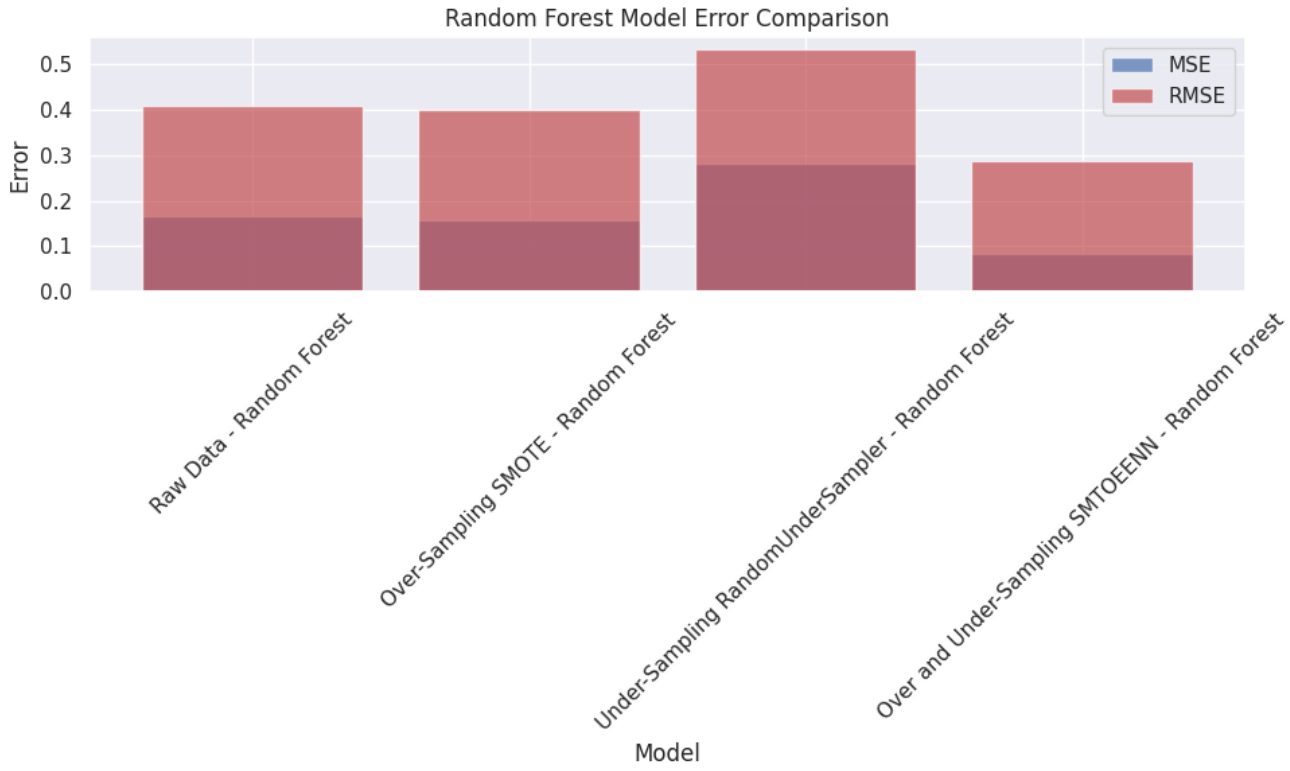


Hình 1: SMOTEENN

Biểu đồ trực quan cho kết quả huấn luyện:



Hình 16: So sánh hiệu suất mô hình trên các tập dữ liệu



Hình 17: So sánh hiệu suất mô hình trên các tập dữ liệu

Comparison

	Model	Accuracy	Kappa	MSE	RMSE
0	Raw Data - Random Forest	0.834170	0.202691	0.165830	0.407222
1	Over-Sampling SMOTE - Random Forest	0.841809	0.683614	0.158191	0.397732
2	Under-Sampling RandomUnderSampler - Random Forest	0.718848	0.437636	0.281152	0.530237
3	Over and Under-Sampling SMTOEENN - Random Forest	0.916997	0.825897	0.083003	0.288103

4.7 XGBoost

- Dựa vào dữ liệu sau khi được xử lý, ta khởi tạo những mô hình XGBoost tương ứng:

```
xgb = xgboost.XGBClassifier()
xgb.fit(X_train, y_train)
```

```
xgb_smote = xgboost.XGBClassifier()
xgb_smote.fit(X_train_smote, y_train_smote)
```

```
xgb_rus = xgboost.XGBClassifier()
xgb_rus.fit(X_train_rus, y_train_rus)
```

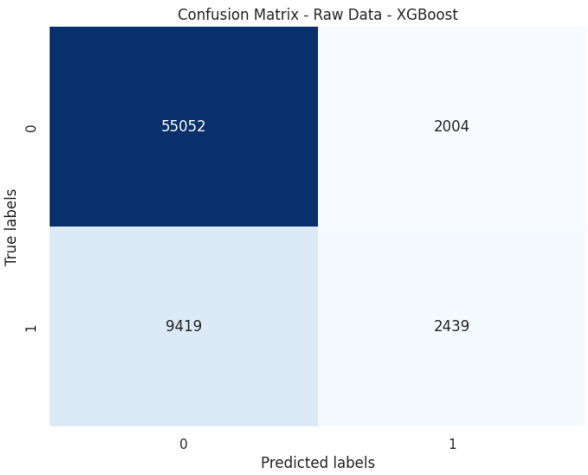
```
xgb_smoteenn = xgboost.XGBClassifier()
xgb_smoteenn.fit(X_train_smoteenn, y_train_smoteenn)
```

- Ở 4 khung code trên, mỗi khung có 2 dòng:
  - Dòng 1: Khởi tạo mô hình XGBoost.
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.
- 4 khung code tạo và huấn luyện 4 mô hình XGBoost tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMOTEENN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```
# Evaluate XGBoost models
evaluate_model(xgb, X_train, y_train, X_test, y_test, 'Raw Data - XGBoost')
evaluate_model(xgb_smote, X_train_smote, y_train_smote, X_test_smote, y_test_smote,
    'Over-Sampling SMOTE - XGBoost')
evaluate_model(xgb_rus, X_train_rus, y_train_rus, X_test_rus, y_test_rus, 'Under-Sampling
    RandomUnderSampler - XGBoost')
evaluate_model(xgb_smoteenn, X_train_smoteenn, y_train_smoteenn, X_test_smoteenn,
    y_test_smoteenn, 'Over and Under-Sampling SMOTEENN - XGBoost')
```

Raw data

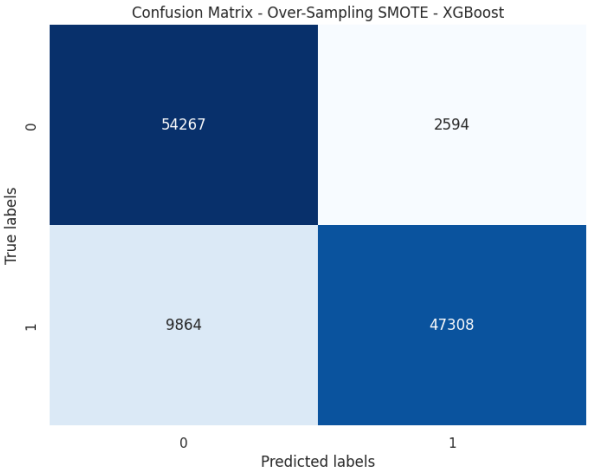
```
            Raw Data - XGBoost Model:

Training Set:
              precision    recall  f1-score   support
           0       0.86      0.98      0.92    132999
           1       0.71      0.24      0.36     27799
    accuracy                           0.85    160798
   macro avg       0.78      0.61      0.64    160798
weighted avg       0.83      0.85      0.82    160798

Training Set Accuracy: 0.8514036244231894

Test Set:
              precision    recall  f1-score   support
           0       0.85      0.97      0.91     57056
           1       0.56      0.18      0.27     11858
    accuracy                           0.83     68914
   macro avg       0.70      0.57      0.59     68914
weighted avg       0.80      0.83      0.80     68914

Test Set Accuracy: 0.8341701250834374

Cohen's Kappa: 0.20269132242908072
Mean Squared Error : 0.1658
Root Mean Squared Error : 0.4072
```



Hình 1: Raw data

SMOTE

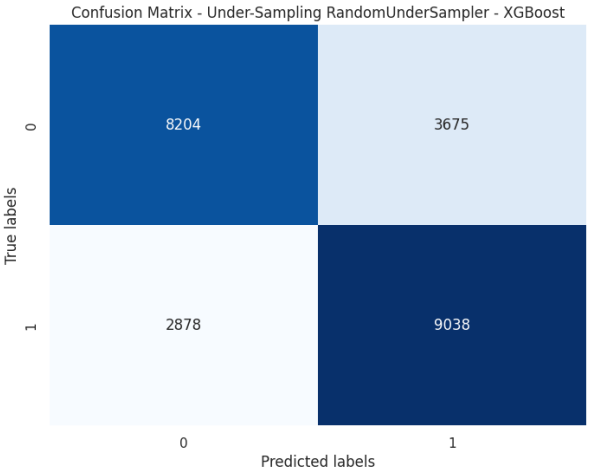
```
            Over-Sampling SMOTE - XGBoost Model:

Training Set:
              precision    recall  f1-score   support
           0       0.85      0.85      0.85    133194
           1       0.85      0.85      0.85    132883
    accuracy                           0.85    266077
   macro avg       0.85      0.85      0.85    266077
weighted avg       0.85      0.85      0.85    266077

Training Set Accuracy: 0.8486340420254287

Test Set:
              precision    recall  f1-score   support
           0       0.84      0.85      0.84     56861
           1       0.85      0.84      0.84     57172
    accuracy                           0.84    114033
   macro avg       0.84      0.84      0.84    114033
weighted avg       0.84      0.84      0.84    114033

Test Set Accuracy: 0.8419141827365763

Cohen's Kappa: 0.6838340744897178
Mean Squared Error : 0.1581
Root Mean Squared Error : 0.3976
```



Hình 1: SMOTE

RUS

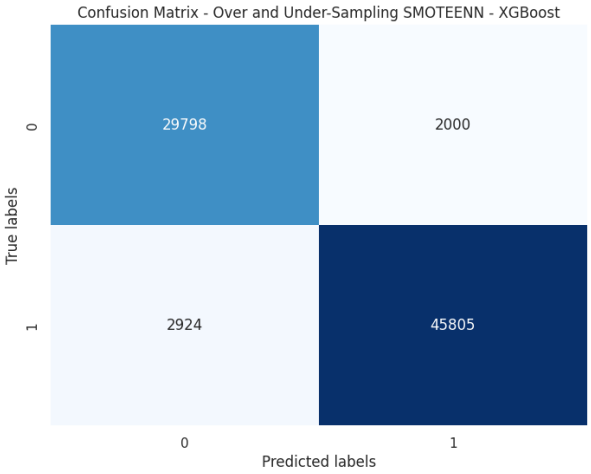
```
Under-Sampling RandomUnderSampler - XGBoost Model:

Training Set:
              precision    recall  f1-score   support
           0       0.80      0.75      0.77     27778
           1       0.76      0.81      0.79     27741
    accuracy                           0.78     55519
   macro avg       0.78      0.78      0.78     55519
weighted avg       0.78      0.78      0.78     55519

Training Set Accuracy: 0.7795709576901602

Test Set:
              precision    recall  f1-score   support
           0       0.73      0.68      0.71     11879
           1       0.71      0.75      0.73     11916
    accuracy                           0.72     23795
   macro avg       0.72      0.72      0.72     23795
weighted avg       0.72      0.72      0.72     23795

Test Set Accuracy: 0.718848497583526

Cohen's Kappa: 0.4376362487764176
Mean Squared Error : 0.2812
Root Mean Squared Error : 0.5302
```



Hình 1: RUS

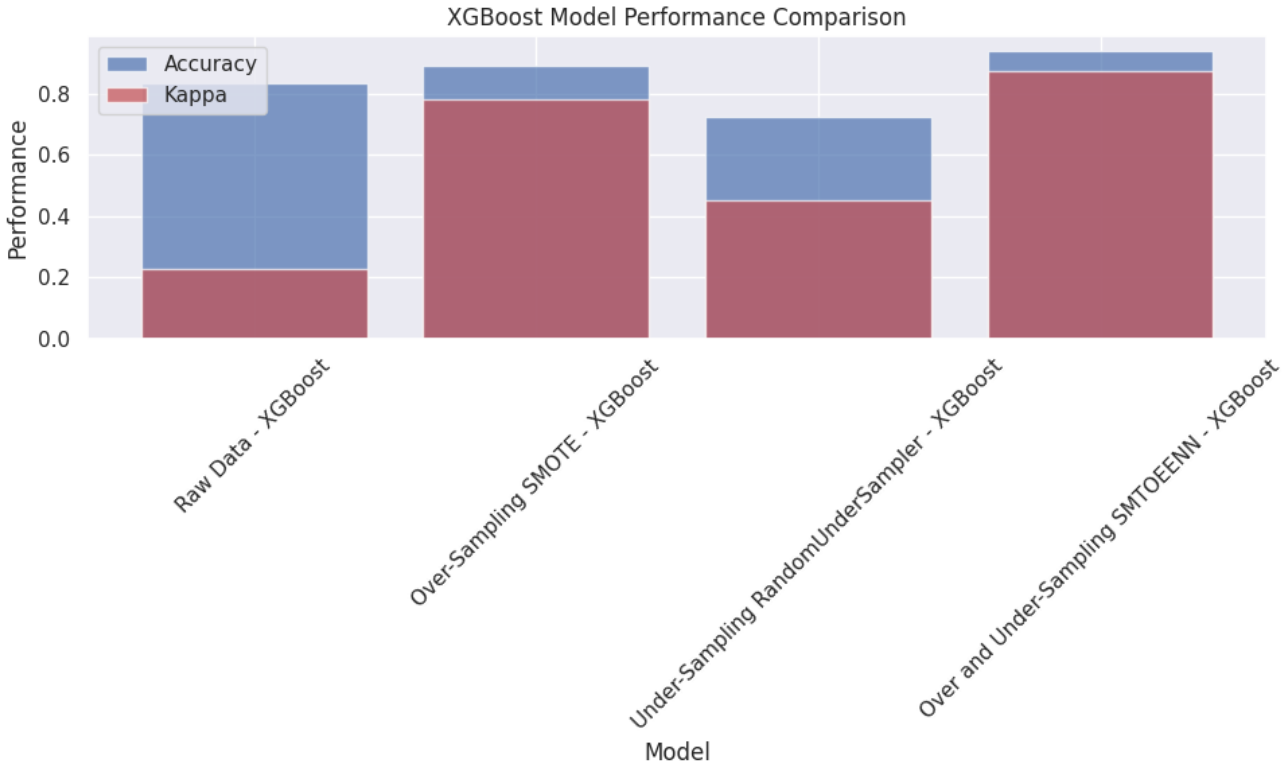
SMOTEENN

```
Over and Under-Sampling SMOTEENN - XGBoost Model:

Training Set:
              precision    recall  f1-score   support
           0       0.91      0.90      0.90     74860
           1       0.93      0.94      0.94    113036
    accuracy                           0.92    187896
   macro avg       0.92      0.92      0.92    187896
weighted avg       0.92      0.92      0.92    187896

Training Set Accuracy: 0.9238940690594797

Test Set:
              precision    recall  f1-score   support
           0       0.90      0.89      0.89     31798
           1       0.93      0.94      0.93     48729
    accuracy                           0.92     80527
   macro avg       0.91      0.91      0.91     80527
weighted avg       0.92      0.92      0.92     80527

Test Set Accuracy: 0.9169967836874588

Cohen's Kappa: 0.8258971330736831
Mean Squared Error : 0.0830
Root Mean Squared Error : 0.2881
```

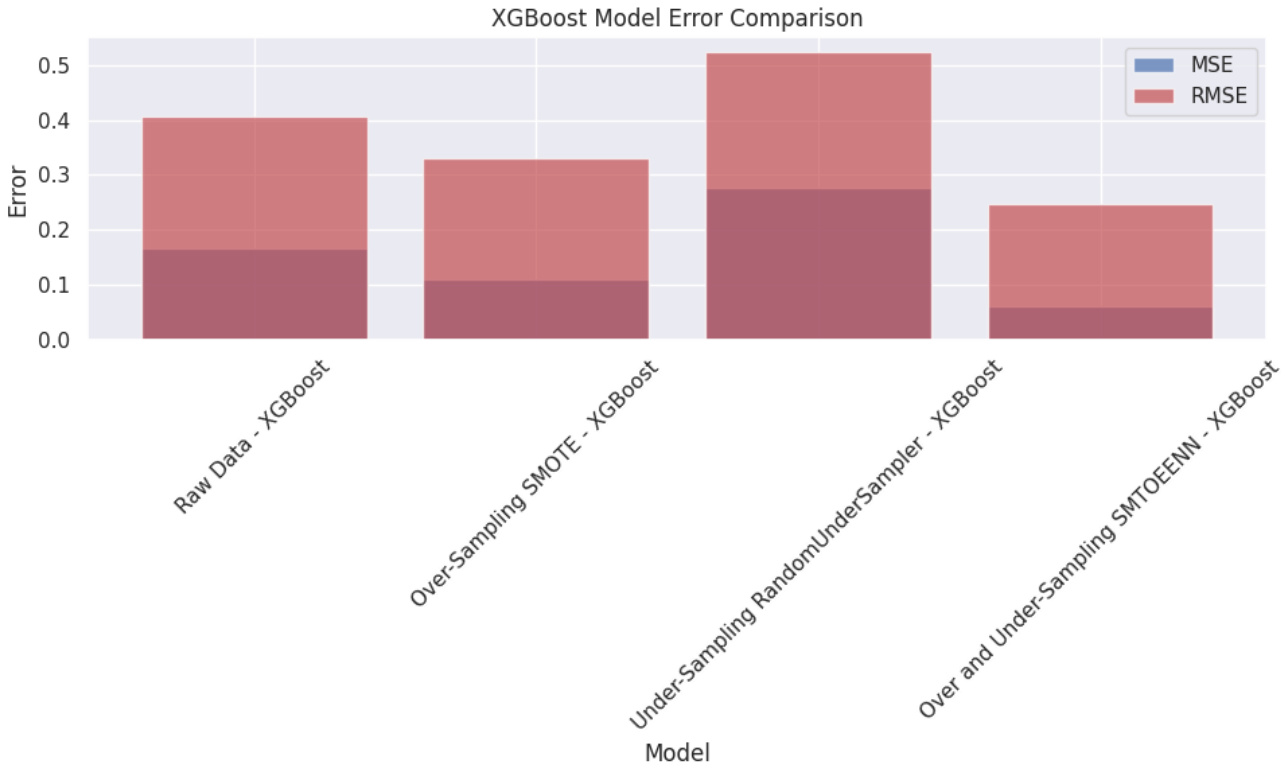


Hình 1: SMOTEENN

Biểu đồ trực quan cho kết quả huấn luyện:



Hình 18: So sánh hiệu suất mô hình trên các tập dữ liệu



Hình 19: So sánh hiệu suất mô hình trên các tập dữ liệu

Comparison

	Model	Accuracy	Kappa	MSE	RMSE
0	Raw Data - XGBoost	0.834243	0.226712	0.165757	0.407133
1	Over-Sampling SMOTE - XGBoost	0.891084	0.782242	0.108916	0.330024
2	Under-Sampling RandomUnderSampler - XGBoost	0.724606	0.449153	0.275394	0.524780
3	Over and Under-Sampling SMTOEENN - XGBoost	0.938853	0.872692	0.061147	0.247280

4.8 KNN

- Để tiến hành huấn luyện mô hình KNN, ta cần sử dụng class KNeighborsClassifier trong thư viện sklearn.neighbors

```
from sklearn.neighbors import KNeighborsClassifier
```

- Và dựa vào dữ liệu sau khi được xử lý, ta khởi tạo những mô hình KNN tương ứng:

```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

```
knn_model_smote = KNeighborsClassifier(n_neighbors=5)
knn_model_smote.fit(X_train_smote, y_train_smote)
```

```
knn_model_rus = KNeighborsClassifier(n_neighbors=5)
knn_model_rus.fit(X_train_rus, y_train_rus)
```

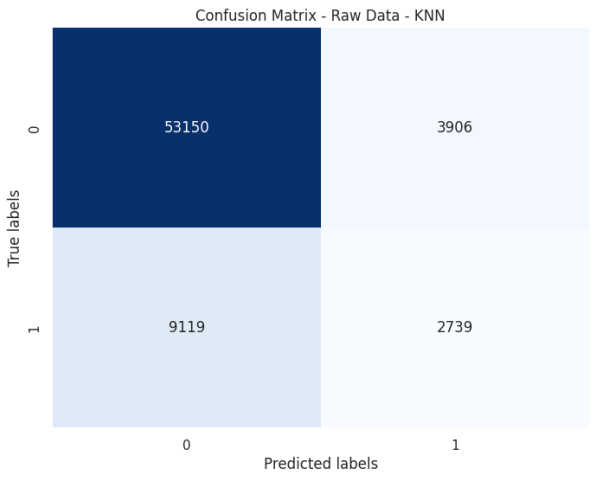
```
knn_model_smoteenn = KNeighborsClassifier(n_neighbors=5)
knn_model_smoteenn.fit(X_train_smoteenn, y_train_smoteenn)
```

- Ở 4 khung code trên, mỗi khung có 2 dòng:
  - Dòng 1: Khởi tạo mô hình KNN với n_neighbors được đặt là 5. n_neighbors là tham số chỉ số lượng điểm gần nhất mà giải thuật KNN sử dụng để dự đoán
  - Dòng 2: Sử dụng mô hình vừa tạo để huấn luyện trên dữ liệu huấn luyện X_train và nhãn tương ứng y_train.
- 4 khung code tạo và huấn luyện 4 mô hình KNN tương ứng lần lượt là mô hình cho dữ liệu thô (Raw Data), Over-Sampling SMOTE, Under-Sampling RandomUnderSampler, Over and Under-Sampling SMTOEENN
- Sau khi huấn luyện các mô hình, ta tiến hành đánh giá kết quả của từng mô hình bằng đoạn code sau:

```
# Evaluate KNN models
evaluate_model(knn_model, X_train, y_train, X_test, y_test, 'Raw Data - KNN')
evaluate_model(knn_model_smote, X_train_smote, y_train_smote, X_test_smote, y_test_smote,
    'Over-Sampling SMOTE - KNN')
evaluate_model(knn_model_rus, X_train_rus, y_train_rus , X_test_rus, y_test_rus,
    'Under-Sampling RandomUnderSampler - KNN')
evaluate_model(knn_model_smoteenn, X_train_smoteenn, y_train_smoteenn , X_test_smoteenn,
    y_test_smoteenn, 'Over and Under-Sampling SMOTEENN - KNN')
```

Raw Data

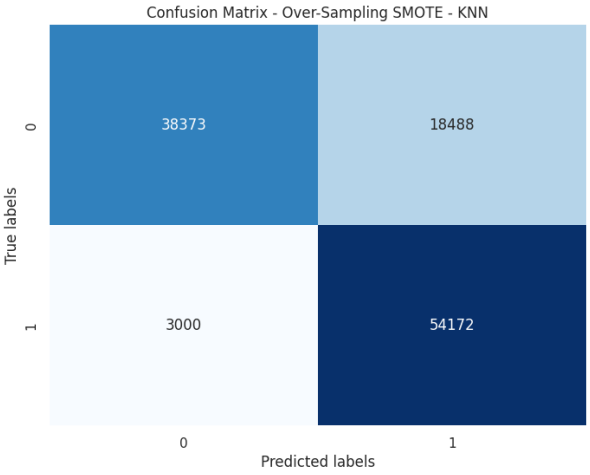
```
Raw Data - KNN Model:

Training Set:
              precision    recall  f1-score   support
           0       0.88      0.96      0.92    132999
           1       0.68      0.39      0.49     27799
    accuracy                           0.86    160798
   macro avg       0.78      0.67      0.71    160798
weighted avg       0.85      0.86      0.85    160798

Training Set Accuracy: 0.8625977935048943

Test Set:
              precision    recall  f1-score   support
           0       0.85      0.93      0.89     57056
           1       0.41      0.23      0.30     11858
    accuracy                           0.81     68914
   macro avg       0.63      0.58      0.59     68914
weighted avg       0.78      0.81      0.79     68914

Test Set Accuracy: 0.8109963142467423

Cohen's Kappa: 0.19679065876019675
Mean Squared Error : 0.1890
Root Mean Squared Error : 0.4347
```



Hình 1: Raw Data

Over-Sampling SMOTE

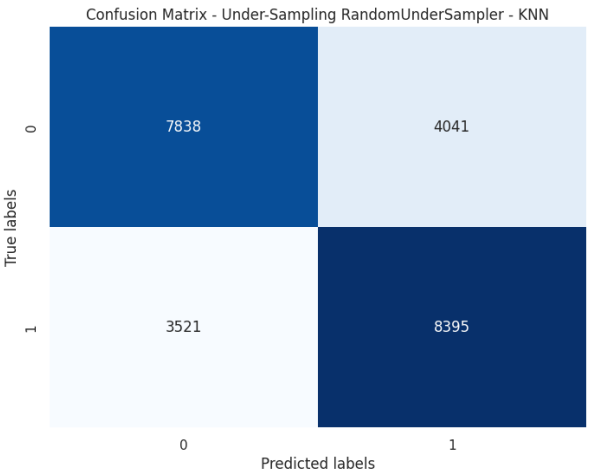
```
Over-Sampling SMOTE - KNN Model:

Training Set:
              precision    recall  f1-score   support
           0       0.97      0.75      0.85    133194
           1       0.80      0.97      0.88    132883
    accuracy                           0.86    266077
   macro avg       0.88      0.86      0.86    266077
weighted avg       0.88      0.86      0.86    266077

Training Set Accuracy: 0.8639453992641228

Test Set:
              precision    recall  f1-score   support
           0       0.93      0.67      0.78     56861
           1       0.75      0.95      0.83     57172
    accuracy                           0.81    114033
   macro avg       0.84      0.81      0.81    114033
weighted avg       0.84      0.81      0.81    114033

Test Set Accuracy: 0.8115633193900011

Cohen's Kappa: 0.622844421405591
Mean Squared Error : 0.1884
Root Mean Squared Error : 0.4341
```



Hình 1: Over-Sampling SMOTE

RUS

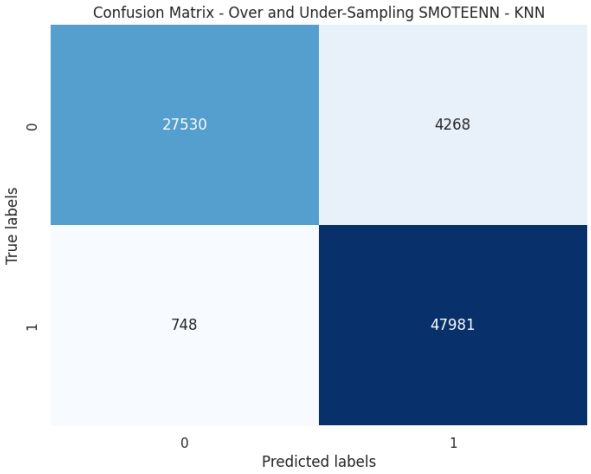
```
Under-Sampling RandomUnderSampler - KNN Model:

Training Set:
              precision    recall  f1-score   support
           0       0.80      0.77      0.78     27778
           1       0.77      0.80      0.79     27741
    accuracy                           0.78     55519
   macro avg       0.78      0.78      0.78     55519
weighted avg       0.78      0.78      0.78     55519

Training Set Accuracy: 0.784109944343378

Test Set:
              precision    recall  f1-score   support
           0       0.69      0.66      0.67     11879
           1       0.68      0.70      0.69     11916
    accuracy                           0.68     23795
   macro avg       0.68      0.68      0.68     23795
weighted avg       0.68      0.68      0.68     23795

Test Set Accuracy: 0.6822021433074176

Cohen's Kappa: 0.36435955057513414
Mean Squared Error : 0.3178
Root Mean Squared Error : 0.5637
```



Hình 1: RUS

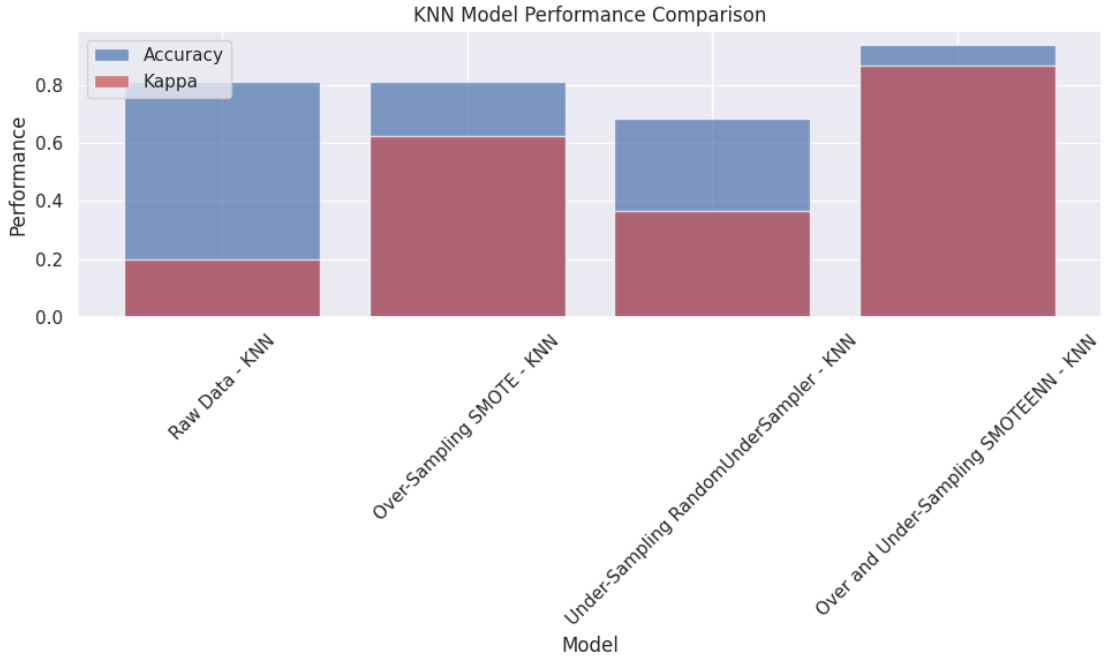
SMTOEENN

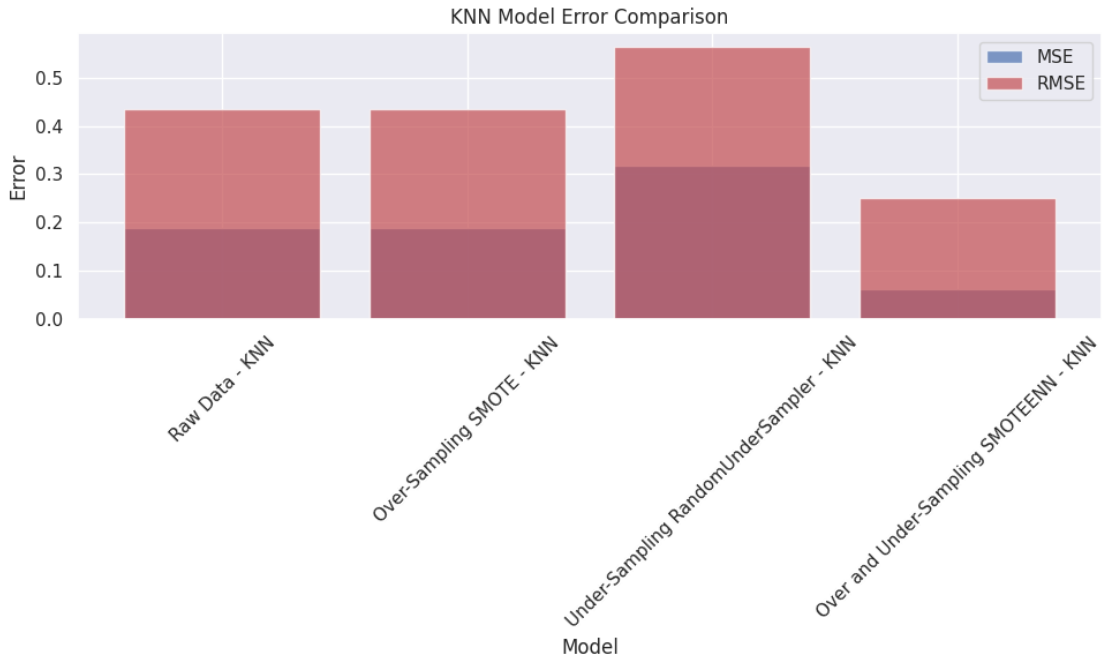
```
Over and Under-Sampling SMOTEENN - KNN Model:

Training Set:
              precision    recall  f1-score   support
           0       0.99      0.92      0.95     74860
           1       0.95      0.99      0.97    113036
    accuracy                           0.96    187896
   macro avg       0.97      0.96      0.96    187896
weighted avg       0.96      0.96      0.96    187896

Training Set Accuracy: 0.9629954868650743

Test Set:
              precision    recall  f1-score   support
           0       0.97      0.87      0.92     31798
           1       0.92      0.98      0.95     48729
    accuracy                           0.94     80527
   macro avg       0.95      0.93      0.93     80527
weighted avg       0.94      0.94      0.94     80527

Test Set Accuracy: 0.9377103331801756

Cohen's Kappa: 0.867103012192816
Mean Squared Error : 0.0623
Root Mean Squared Error : 0.2496
```



Hình 1: SMOTEENN

Biểu đồ trực quan cho kết quả huấn luyện:





Comparison

	Model	Accuracy	Kappa	MSE	RMSE
0	Raw Data	0.810996	0.196791	0.189004	0.434746
1	Over-Sampling SMOTE	0.811563	0.622844	0.188437	0.434093
2	Under-Sampling RandomUnderSampler	0.682202	0.364360	0.317798	0.563736
3	Over and Under-Sampling SMOTEENN	0.937710	0.867103	0.062290	0.249579

4.9 ANN

- Để xây dựng mô hình ANN, ta sẽ dùng thư viện sklearn.metrics và các thư viện của tensorflow

```
from sklearn.metrics import accuracy_score, cohen_kappa_score, mean_squared_error
import math
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the architecture of the ANN
model_ann = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

- Mô hình ANN (biến model_ann trong code) được được khởi tạo với kiểu Sequential. Mô hình này được xây dựng từ một chuỗi các lớp neuron. Trong đoạn code này, mô hình bao gồm ba lớp Dense.
- Sau khi khởi tạo mô hình, ta có thể tiến hành huấn luyện:

```
# Compile the model
model_ann.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```
# Train the model
history = model_ann.fit(X_train_smoteenn, y_train_smoteenn, epochs=200, batch_size=4096)
```

Kết quả:

```
Epoch 1/200
46/46 [==============================] - 2s 16ms/step - loss: 0.5296 - accuracy: 0.7296
Epoch 2/200
46/46 [==============================] - 0s 10ms/step - loss: 0.3864 - accuracy: 0.8283
Epoch 3/200
46/46 [==============================] - 0s 9ms/step - loss: 0.3537 - accuracy: 0.8454
```

.

.

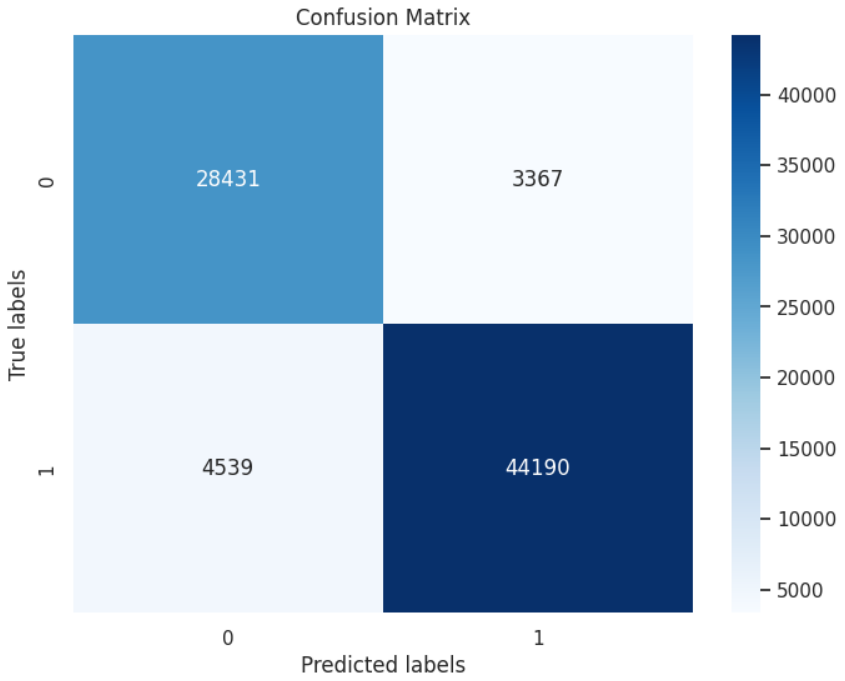
.

```
Epoch 198/200
46/46 [==============================] - 0s 10ms/step - loss: 0.2257 - accuracy: 0.9047
Epoch 199/200
46/46 [==============================] - 0s 10ms/step - loss: 0.2258 - accuracy: 0.9049
Epoch 200/200
46/46 [==============================] - 0s 10ms/step - loss: 0.2255 - accuracy: 0.9047
```

- Sau khi huấn luyện mô hình, ta tiến hành đánh giá kết quả thông qua đoạn code sau:

```
# Evaluate the model
y_pred_ann = model_ann.predict(X_test_smoteenn)
y_pred_ann = (y_pred_ann > 0.5).astype(int)

conf_matrix = confusion_matrix(y_test_smoteenn, y_pred_ann)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

accuracy_ann = accuracy_score(y_test_smoteenn, y_pred_ann)
kappa_ann = cohen_kappa_score(y_test_smoteenn, y_pred_ann)
mse_ann = mean_squared_error(y_test_smoteenn, y_pred_ann)
rmse_ann = math.sqrt(mse_ann)
f1_ann = f1_score(y_test_smoteenn, y_pred_ann, average='weighted'),

print("Accuracy:", accuracy_ann)
print("Kappa:", kappa_ann)
print("MSE:", mse_ann)
print("RMSE:", rmse_ann)
print("F1_Score:", f1_ann)
```

```
2517/2517 [==============================] - 5s 2ms/step
```




```
Accuracy: 0.9018217492269673
Kappa: 0.795868960278696
MSE: 0.09817825077303265
RMSE: 0.3133340881120863
F1_Score: (0.9021125181082698,)
```

4.10 Ensemble algorithm

Các mô hình đã xây dựng ở các phần trước đều cho dự đoán với độ chính cao. Do đó, chúng ta có thể kết hợp các mô hình này lại để cho ra một mô hình dự đoán mạnh hơn và tốt hơn. Đây chính là ý tưởng của ensemble algorithm: Từ các mô hình dự đoán đã xây dựng, chúng ta chọn lọc ra các mô hình có độ chính xác cao bao gồm 6 mô hình là **Random Forest, XGBoost, SVM, Decision Tree, ANN, KNN**:

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

model_rf_ens = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=15)
model_xgb_ens = XGBClassifier(random_state=42, n_jobs=-1, max_depth=15)
model_svc_ens = SVC(random_state=42, probability=True, max_iter=1000)
model_ann_ens = MLPClassifier(random_state=42, max_iter=200, hidden_layer_sizes=(50, 50))
model_dt_ens = DecisionTreeClassifier(random_state=42, max_depth=15)
model_knn_ens = KNeighborsClassifier(n_neighbors=3, n_jobs=-1)

ensemble_model = VotingClassifier(estimators=[
    ('rf', model_rf_ens),
    ('xgb', model_xgb_ens),
    ('svc', model_svc_ens),
```

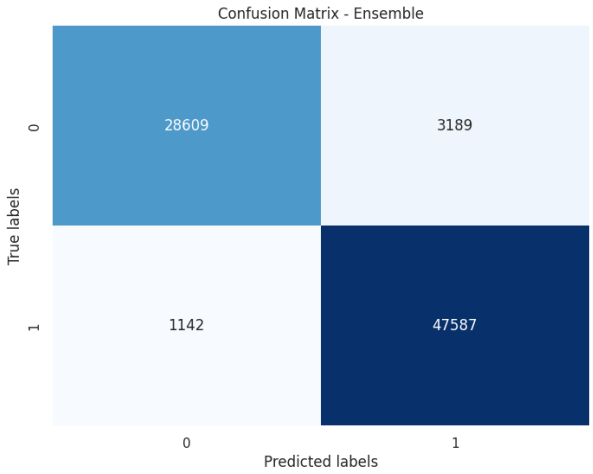
```
    ('ann', model_ann_ens),
    ('dt', model_dt_ens),
    ('knn', model_knn_ens)
], voting='soft')

ensemble_model.fit(X_train_smoteenn, y_train_smoteenn)
ensemble_predictions = ensemble_model.predict(X_test_smoteenn)

ensemble_accuracy = accuracy_score(y_test_smoteenn, ensemble_predictions)
ensemble_kappa = cohen_kappa_score(y_test_smoteenn, ensemble_predictions)
ensemble_mse = mean_squared_error(y_test_smoteenn, ensemble_predictions)
ensemble_rmse = math.sqrt(ensemble_mse)
ensemble_f1 = f1_score(y_test_smoteenn, ensemble_predictions, average='weighted')

print("Ensemble Accuracy:", ensemble_accuracy)
print("Ensemble Kappa:", ensemble_kappa)
print("Ensemble MSE:", ensemble_mse)
print("Ensemble RMSE:", ensemble_rmse)
print("Ensemble F1 Score:", ensemble_f1)
```

Kết quả huấn luyện mô hình được kết hợp từ 6 mô hình đã chọn:

```
import math
from sklearn.metrics import classification_report, accuracy_score, cohen_kappa_score,
    mean_squared_error, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

evaluate_model(ensemble_model, X_train_smoteenn, y_train_smoteenn, X_test_smoteenn,
    y_test_smoteenn, "Ensemble")
```

Over and Under-Sampling SMTOEENN

```
Ensemble Model:

Training Set:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98     85356
           1       0.98      1.00      0.99    129382

    accuracy                           0.99    214738
   macro avg       0.99      0.99      0.99    214738
weighted avg       0.99      0.99      0.99    214738


Training Set Accuracy: 0.987435852061582

Test Set:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94     21302
           1       0.95      0.97      0.96     32383

    accuracy                           0.95     53685
   macro avg       0.95      0.95      0.95     53685
weighted avg       0.95      0.95      0.95     53685


Test Set Accuracy: 0.9521095278010617

Cohen's Kappa: 0.899594710937901
Mean Squared Error : 0.0479
Root Mean Squared Error : 0.2188
```



Hình 1: Over and Under-Sampling SMTOEENN

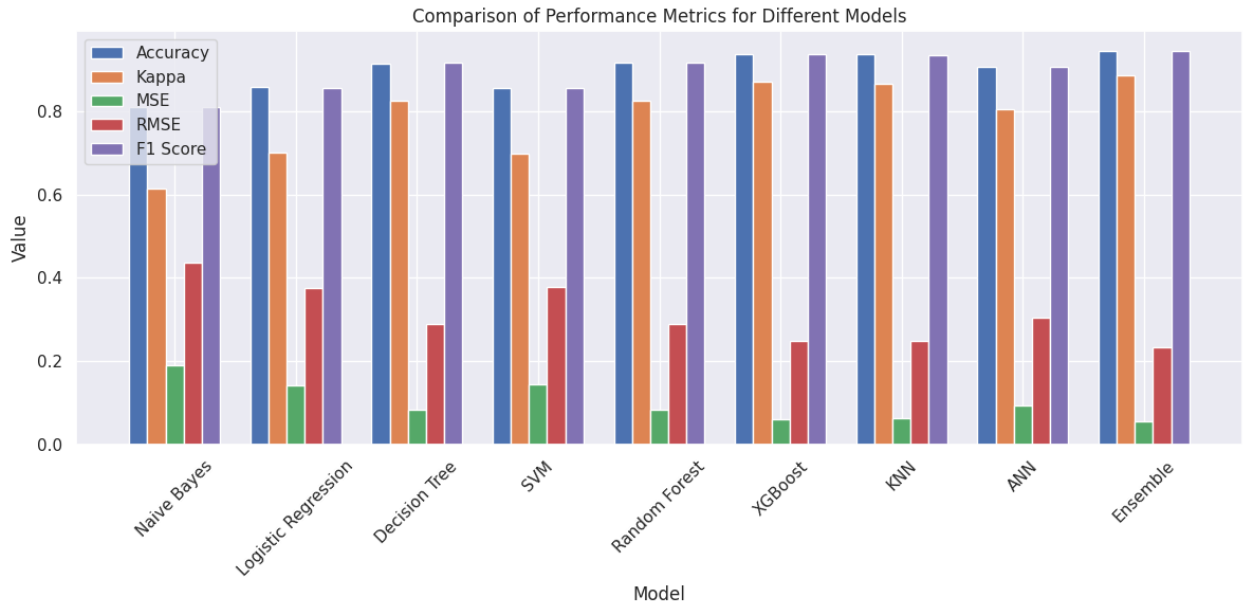
5 So sánh hiệu quả giữa các mô hình

Để chọn lọc ra mô hình tốt nhất thì chúng ta sẽ vẽ bảng biểu thị và hiển thị dữ liệu so sánh:

```python
import pandas as pd
import matplotlib.pyplot as plt
import math
import seaborn as sns
from sklearn.metrics import accuracy_score, cohen_kappa_score, mean_squared_error, f1_score

# Define the performance metrics for all models
models = ['Naive Bayes', 'Logistic Regression', 'Decision Tree', 'SVM', 'Random Forest',
    'XGBoost', 'KNN', 'ANN', 'Ensemble']

metrics = {
    'Naive Bayes': {'Accuracy': accuracy_nb_comp, 'Kappa': kappa_nb_comp, 'MSE': mse_nb_comp,
        'RMSE': rmse_nb_comp, 'F1 Score': f1_nb},
    'Logistic Regression': {'Accuracy': accuracy_lg_comp, 'Kappa': kappa_lg_comp, 'MSE':
        mse_lg_comp, 'RMSE': rmse_lg_comp, 'F1 Score': f1_lg},
    'Decision Tree': {'Accuracy': accuracy_dt_comp, 'Kappa': kappa_dt_comp, 'MSE':
        mse_dt_comp, 'RMSE': rmse_dt_comp, 'F1 Score': f1_dt},
    'SVM': {'Accuracy': accuracy_svm_comp, 'Kappa': kappa_svm_comp, 'MSE': mse_svm_comp,
        'RMSE': rmse_svm_comp, 'F1 Score': f1_svm},
    'Random Forest': {'Accuracy': accuracy_rf_comp, 'Kappa': kappa_rf_comp, 'MSE':
        mse_rf_comp, 'RMSE': rmse_rf_comp, 'F1 Score': f1_rf},
    'XGBoost': {'Accuracy': accuracy_xgb_comp, 'Kappa': kappa_xgb_comp, 'MSE': mse_xgb_comp,
        'RMSE': rmse_xgb_comp, 'F1 Score': f1_xgb},
```

```
    'KNN': {'Accuracy': accuracy_knn_comp, 'Kappa': kappa_knn_comp, 'MSE': mse_knn_comp,
        'RMSE': rmse_knn_comp, 'F1 Score': f1_knn},
    'ANN': {'Accuracy': accuracy_ann, 'Kappa': kappa_ann, 'MSE': mse_ann, 'RMSE': rmse_ann,
        'F1 Score': f1_ann},
    'Ensemble': {'Accuracy': ensemble_accuracy, 'Kappa': ensemble_kappa, 'MSE': ensemble_mse,
        'RMSE': ensemble_rmse, 'F1 Score': ensemble_f1},
}

# Ensure that all metrics are single values (not lists or tuples)
for model, data in metrics.items():
    for key in data.keys():
        if isinstance(data[key], (list, tuple)):
            data[key] = data[key][0]

# Create a DataFrame to store the performance metrics for all models
metrics_df = pd.DataFrame(metrics).T

# Check the dtype of all metrics to ensure they are numerical
for metric in metrics_df.columns:
    if not pd.api.types.is_numeric_dtype(metrics_df[metric]):
        metrics_df[metric] = metrics_df[metric].astype(float)

# Plot a grouped bar chart for multiple metrics
metric_names = ['Accuracy', 'Kappa', 'MSE', 'RMSE', 'F1 Score']
num_models = len(models)
num_metrics = len(metric_names)
bar_width = 0.15
bar_positions = list(range(num_models))
bar_shifts = [bar_width * i for i in range(num_metrics)]

plt.figure(figsize=(12, 6))
```

```python
for i, metric_name in enumerate(metric_names):
    metric_values = [metrics[model][metric_name] for model in models]
    plt.bar([x + bar_shifts[i] for x in bar_positions], metric_values, width=bar_width,
        label=metric_name)

plt.xlabel('Model')
plt.ylabel('Value')
plt.title('Comparison of Performance Metrics for Different Models')
plt.xticks([x + (bar_width * (num_metrics / 2 - 0.5)) for x in bar_positions], models,
    rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

print("Performance Metrics for Different Models:")
print(metrics_df)
```



Over and Under-Sampling SMTOEENN

```
Performance Metrics for Different Models:
                     Accuracy     Kappa       MSE      RMSE  F1 Score
Naive Bayes          0.810337  0.614001  0.189663  0.435503  0.812151
Logistic Regression  0.858619  0.701861  0.141381  0.376007  0.858024
Decision Tree        0.916674  0.825904  0.083326  0.288663  0.916736
SVM                  0.856607  0.698727  0.143393  0.378673  0.856302
Random Forest        0.916997  0.825897  0.083003  0.288103  0.916894
XGBoost              0.938853  0.872692  0.061147  0.247280  0.938998
KNN                  0.937710  0.867103  0.062290  0.249579  0.936971
ANN                  0.906715  0.804800  0.093285  0.305427  0.906715
Ensemble             0.946217  0.886186  0.053783  0.231912  0.945876
```

Từ bảng so sánh và bảng dữ liệu chi tiết, chúng ta chọn ra được mô hình tốt nhất là mô hình sử dụng **Ensemble Algorithm**. Mô hình cho kết quả đáng tin cậy khi có độ chính xác cao (gần 95%) với các giá trị sai số ở mức rất thấp (khoảng 0.05 với MSE và 0.23 với RMSE).

6 Kết luận

- Trong bài nghiên cứu này, chúng ta đã khám phá một loạt các mô hình học máy khác nhau từ Naive Bayes, Logistic Regression, Decision Trees, đến mạng Neuron nhân tạo, mỗi mô hình có đặc điểm và phương pháp tiếp cận riêng biệt để giải quyết các loại bài toán phân loại và hồi quy. Các kết quả thu được cho thấy sự đa dạng trong cách thức mỗi mô hình xử lý dữ liệu và đưa ra dự đoán.
- Mô hình Naive Bayes, với giả định về sự độc lập giữa các đặc trưng, phù hợp với dữ liệu có nhiều đặc trưng nhưng không yêu cầu mối quan hệ phức tạp giữa chúng . Ngược lại, Logistic Regression hiệu quả trong việc dự đoán các kết quả nhị phân thông qua hàm Sigmoid, mang lại cái nhìn sâu sắc hơn vào mối liên hệ giữa các biến .
- Mô hình Decision Tree và Random Forest cho phép hiểu sâu về cách dữ liệu được chia để đưa ra quyết định, với Random Forest cải thiện hiệu quả bằng cách giảm thiểu tình trạng overfitting thường thấy trong Decision Tree đơn lẻ . SVM, với khả năng xử lý hiệu quả dữ liệu phi tuyến tính bằng cách sử dụng hàm kernel, cũng là một công cụ mạnh mẽ cho các bài toán phân loại .
- Ngoài ra, các mô hình như K-Nearest Neighbors và mạng Neuron nhân tạo (ANN) cho thấy sự linh hoạt trong việc xử lý các loại dữ liệu và mô hình hóa các mối quan hệ phức tạp, với KNN dựa trên khoảng cách giữa các điểm và ANN sử dụng cấu trúc mạng phức tạp để học từ dữ liệu .

Về Ensemble Models:

- Nghiên cứu này cũng đã khám phá sâu vào tiềm năng của các mô hình ensemble trong việc cải thiện hiệu quả dự đoán so với việc sử dụng các mô hình đơn lẻ. Các kỹ thuật ensemble như Bagging và Boosting đã được áp dụng để kết hợp các dự đoán từ nhiều mô hình khác nhau, giúp giảm variance và bias, mang lại sự cân bằng giữa độ chính xác và tính tổng quát của mô hình.
- Bagging, với việc tạo ra nhiều phiên bản của một mô hình như Random Forest, đã cho thấy khả năng giảm overfitting bằng cách lấy trung bình hoặc bình chọn từ các mô hình được huấn luyện độc lập. Boosting, mặt khác, tập trung vào việc cải thiện các mô hình yếu bằng cách tuần tự điều chỉnh trọng số của các trường hợp, nâng cao hiệu suất của mô hình trên các trường hợp khó hơn.
- XGBoost, một dạng tiên tiến của Gradient Boosting, đã đặc biệt nổi bật trong nghiên cứu này nhờ khả năng xử lý dữ liệu lớn, tính toán nhanh và hiệu quả, cũng như khả năng cung cấp các giải pháp tối ưu qua từng lần lặp, làm giảm đáng kể sai số và thời gian huấn luyện.
- Kết quả cuối cùng từ các mô hình ensemble đã chứng minh rằng kết hợp nhiều mô hình mang lại sự ổn định và tin cậy hơn trong các dự đoán, đồng thời làm nổi bật tầm quan trọng của việc chọn lọc các mô hình phù hợp để tạo thành một bộ ensemble hiệu quả. Điều này không chỉ mở rộng khả năng ứng dụng của các mô hình học máy trong nghiên cứu khoa học dữ liệu mà còn trong thực tiễn ứng dụng thực tế.

Tóm lại, mặc dù mỗi mô hình có ưu và nhược điểm riêng, sự kết hợp và lựa chọn phù hợp các mô hình có thể dẫn đến việc cải thiện đáng kể hiệu suất trong các ứng dụng thực tế. Việc này đòi hỏi một sự hiểu biết sâu rộng về tính chất của dữ liệu và bài toán cụ thể để có thể tận dụng tối đa sức mạnh của học máy.

Tài liệu tham khảo

[1] *A Step-by-Step Explanation of Principal Component Analysis (PCA)*. https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

[2] *Comparing Decision Tree Algorithms: Random Forest vs. XGBoost*. https://www.activestate.com/blog/comparing-decision-tree-algorithms-random-forest-vs-xgboost/.

[3] Peter Dalgaard. *Introductory Statistics with R*. Springer, 2008.

[4] *DEFINITION histogram*. https://www.techtarget.com/searchsoftwarequality/definition/histogram.

[5] *Detecting and Treating Outliers | Treating the odd one out!* https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/.

[6] *Guide to AUC ROC Curve in Machine Learning*. https://www.geeksforgeeks.org/auc-roc-curve/.

[7] Chris C Wright Julius Sim. *The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements, Physical Therapy*. 2005.

[8] *Linear Regression on Student Grade Prediction*. https://rstudio-pubs-static.s3.amazonaws.com/716359_6902dfdd88684340a5f5e11038b9ac22.html.

[9] *Logistic Regression*. https://www.geeksforgeeks.org/understanding-logistic-regression/.

[10] *Support Vector Machine (SVM) Algorithm*. https://www.geeksforgeeks.org/support-vector-machine-algorithm/.

[11] *Understanding Boxplots*. https://builtin.com/data-science/boxplot.

[12] *Understanding Overfitting and How to Prevent It*. https://www.investopedia.com/terms/o/overfitting.asp.

[13] *What is SVM? Machine Learning Algorithm Explained*. https://www.springboard.com/blog/data-science/svm-algorithm/.