**Data science Team\_8 Final Report**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Team\_8 members**   |  |  |  | | --- | --- | --- | | **Name** | **Major** | **student ID number** |  |  |  |  | | --- | --- | --- | | **ANH SUBIN** | **Ai/Software** | **(202334305)** | | **Seo Hyungyo** | **Ai/Software** | **(202334298)** | | **KIM WONJUN** | **Ai/Software** | **(202239866)** | | **CHAE HEEJAE** | **Ai/Software** | **(202135585)** | |

* ***GitHub URL***

[https://github.com/datascienceteam8/datascienceteam 8](https://github.com/datascienceteam8/datascienceteam8)

* *dataset URL*

<https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset>

**Team Member Reflections and Contributions \_ Heejae Chae:**

1. **Project Overview and Objectives**  
   This project addresses the rising number of stroke patients globally and the resulting social burden by aiming to develop early prediction models and personalized prevention strategies based on health data analysis. The main objectives are to analyze health data to predict stroke risk, cluster individuals with similar health profiles to classify risk groups, develop tailored prevention and management strategies for each cluster, and implement models and strategies applicable to real-world medical services. Stroke is a sudden and severe neurological condition where damaged brain cells are difficult to recover, making prevention and early diagnosis critical.
2. **Dataset Introduction and Exploration**  
   The study utilized the publicly available "Stroke Prediction Dataset" from Kaggle. The dataset consists of health records from 5,110 individuals, including numerical variables such as age, average glucose level, and BMI, and categorical variables like gender, smoking status, occupation, and residence type. The target variable is stroke occurrence (binary 0 or 1). The dataset is imbalanced, with only about 5% of samples indicating stroke cases. Exploratory analysis confirmed high correlations between stroke and age, hypertension, and heart disease. Missing BMI values and some inconsistencies in categorical variables were also identified.
3. **Data Preprocessing**  
   Comprehensive preprocessing steps ensured robust model training. The ID column was removed, missing BMI values were imputed with the median, and BMI outliers were clipped between the 1st and 99th percentiles. Categorical variables were trimmed and standardized. Age was grouped into ‘Child’, ‘Adult’, and ‘Senior’ categories. Variables such as ‘ever\_married’ and ‘Residence\_type’ were binary encoded. StandardScaler was applied to three numeric features and OneHotEncoder to 12 categorical features, while five binary variables were left unchanged. The final dataset contained 20 features and was split into 4,088 training and 1,022 testing samples. The processed data showed stable distributions and reproducibility through fixed feature ordering.
4. **Clustering Analysis (KMeans)**  
   KMeans clustering grouped individuals with similar health profiles based on age, average glucose level, and BMI. The optimal number of clusters was determined using the Elbow Method and Silhouette Score, both indicating k=4 as the best choice. Cluster distributions were stable across iterations, with max\_iter=300 used for the final analysis. The clusters were characterized as follows: Cluster 0 had a high proportion of elderly requiring regular 3-month health monitoring; Cluster 1 was a high-risk group with elevated glucose and BMI, recommending lifestyle changes and routine exams; Cluster 3 had relatively stable health but lifestyle improvements were advised; Cluster 4 consisted mainly of children and young adults, classified as low-risk with basic health maintenance recommended. These findings support targeted health policies and early high-risk group detection in clinical practice.
5. **Predictive Modeling**  
   ***5.1 Decision Tree***  
   The Decision Tree model classified stroke occurrence using entropy-based splits. Feature names were decoded and normalized values reversed for intuitive interpretation. The visualized tree displayed node splits, sample counts, and entropy values. Key predictors included age, average glucose level, and BMI. The optimal model achieved approximately 94.9% accuracy.

***5.2 XGBoost***  
XGBoost sequentially trains multiple decision trees, correcting errors from previous trees. It includes built-in regularization to prevent overfitting and exhibits superior prediction accuracy. It is widely used in practical settings due to its strong performance.

1. **Evaluation Strategy and Results**  
   To handle class imbalance (with only about 5% stroke cases), SMOTE was applied to generate synthetic minority samples. Repeated Stratified K-Fold Cross-Validation (10x10) was used for robust performance estimates. Evaluation metrics included Accuracy, Precision, Recall, F1 Score, and ROC AUC. XGBoost achieved an accuracy of 96.25%, precision of 96.59%, recall of 95.90%, F1 score of 96.24%, and ROC AUC of 99.32%, outperforming the Decision Tree, which scored 79.77%, 74.22%, 91.26%, 81.86%, and 85.54%, respectively. Confusion Matrix and ROC Curve analyses further confirmed XGBoost’s superior ability to accurately identify stroke patients compared to Decision Tree, which tended to over-classify non-stroke cases.
2. **Team Collaboration and Roles**  
   Seohyun Kyo handled data preprocessing. Subin An was responsible for clustering analysis. Wonjun Kim led classification modeling and evaluation. Heejae Chae managed overall workflow integration and presentation preparation. The team’s close collaboration enhanced project coherence and quality.
3. **Team Member Reflections**  
   Subin deepened understanding of supervised and unsupervised learning through hands-on experience. Seohyun recognized the importance of preprocessing and gained practical application skills. Wonjun learned the significance of model selection and evaluation criteria through multiple experiments. Heejae experienced the full data analysis process, consolidating analytical skills.
4. **Conclusions and Future Directions**  
   This project demonstrated the feasibility of predicting stroke risk and developing personalized prevention strategies using health data. The superior performance of XGBoost suggests strong potential for clinical application. Future research should focus on incorporating broader health data, applying additional machine learning models, and strengthening clinical validations to enhance model robustness and usability.

**Team Member Reflections and Contributions \_ Wonjun Kim:**

**Evaluation**

To quantitatively compare the performance of stroke prediction models, this study evaluated two classification algorithms: Decision Tree and XGBoost. Rather than relying solely on accuracy, the evaluation was based on multiple performance metrics and visualizations to ensure a comprehensive understanding of each model’s behavior.

**Strategy**

Healthcare data often contains significant class imbalance. In the case of the stroke data used in this study, the incidence of stroke is only about 5% of the total samples. This imbalance carries the risk that accuracy-based evaluation alone may distort the true performance of the model. For example, even if a model predicts all samples as 'non-stroke,' it could achieve a high accuracy of 95%, but in reality, it would commit a serious error by failing to identify any of the crucial minority class (stroke).

**SMOTE (Synthetic Minority Over-sampling Technique):** Class imbalance was alleviated by artificially generating data for the minority class (Stroke=1).

**Repeated Stratified K-Fold Cross-Validation (10x10):** To evaluate the model's stability and generalization performance, 10-fold cross-validation was repeated 10 times. This means that a total of 100 model training and evaluations were performed, allowing for the acquisition of average performance that considers both bias and variance.

**Evaluation Index**

The following classification indexes were used to measure the performance of the model

|  |  |
| --- | --- |
| Accuracy | Correct ratio among all predictions |
| Precision | Actual positive ratio among positive predictions |
| Recall | How well the model matched actual positives |
| F1 score | Harmonic mean of Precision and Recall |
| ROC AUC | Value according to changes in classification thresholds |

**Result**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1 Score | ROC AUC |
| XGBoost | 0.9625 | 0.9659 | 0.9590 | 0.9624 | 0.9932 |
| Decision tree | 0.7977 | 0.7422 | 0.9126 | 0.8186 | 0.8554 |

The XGBoost model outperformed the Decision Tree in all metrics. With its high recall, precision, and ROC AUC, it is considered highly suitable for practical use in medical applications where correctly identifying stroke patients is critical.

While the Decision Tree is simpler and more interpretable, it struggled to capture complex patterns in the data and exhibited lower overall performance, particularly in precision.

**Clustrering**

K-Means clustering was performed using three numerical variables: age, avg\_glucose\_level, and bmi. The Elbow Method and Silhouette Score were used to find the optimal value for the number of clusters (k).

**Elbow method**

In order to determine the optimal number of clusters (k) in clustering analysis, the Elbow Method was applied.

The KMeans algorithm was performed while changing the number of clusters k from 2 to 9, and the corresponding SSE values ​​were measured. SSE indicates the degree of dispersion within a cluster, and as k increases, data points are assigned to smaller clusters, so the SSE value tends to decrease.

On the graph, a point where the decrease in SSE is significantly reduced and a bend is clearly observed around k=4.

**Silhouette Score**

The average Silhouette Score was calculated by changing the k value from 2 to 9. The analysis results showed that the highest Silhouette Score was recorded at k=4, and then gradually decreased as k increased.

텍스트, 그래프, 라인, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**Clustering Conclusions**

Both the elbow method and the silhouette score indicated that k = 4 was the most appropriate clustering number for the data set

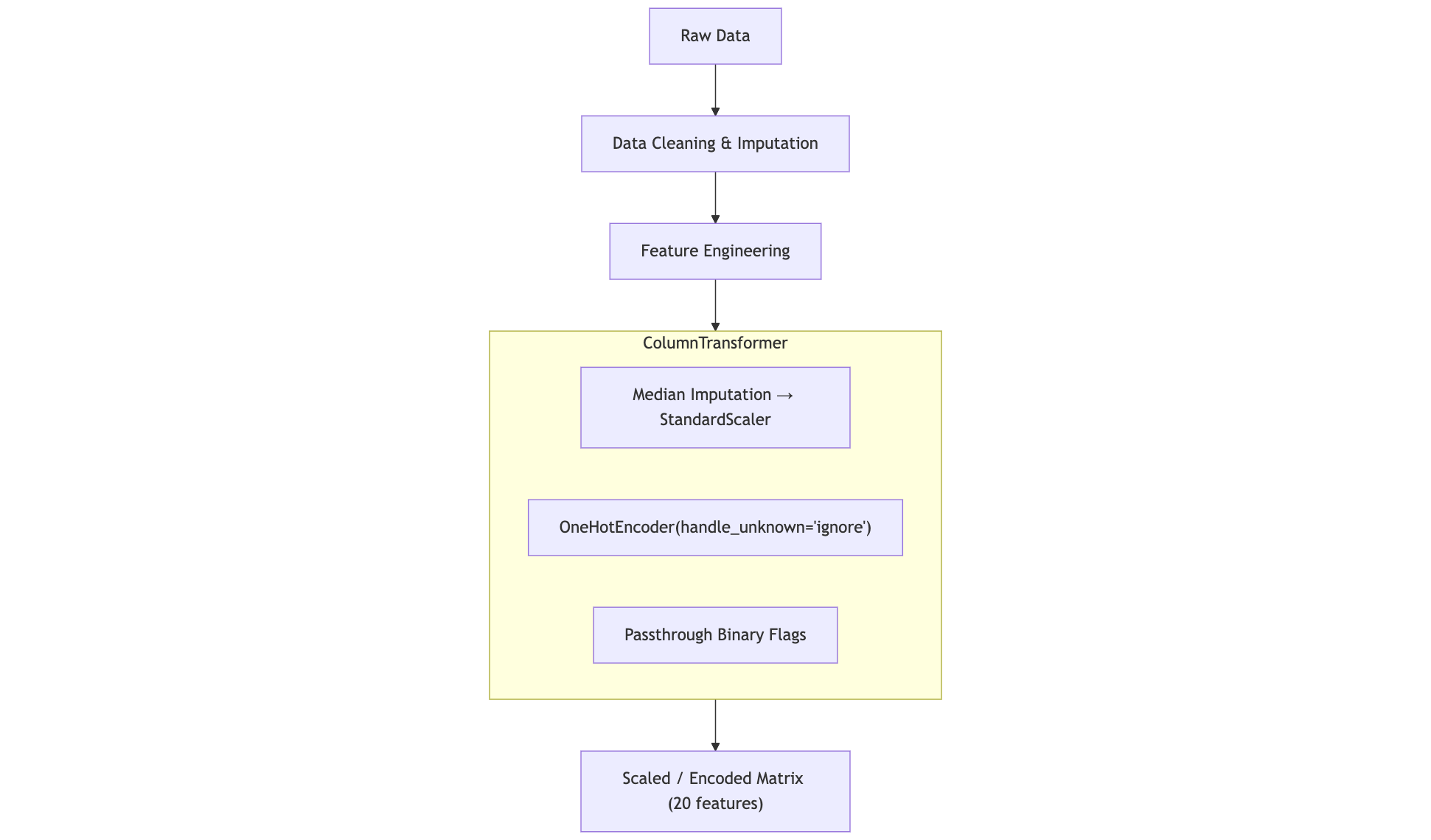
**Team Member Reflections and Contributions \_ Seohyun Kyo:**

**Preprocessing Flow**

**1. Data Overview**

* **Raw records**: 5,110 rows × 12 columns (original)
* **Number of features after preprocessing**: **20 columns** → 3 numeric, 12 one-hot, 5 binary
* **Train / Test split** (train\_test\_split(test\_size=0.20, stratify=y, random\_state=42)):
  + **Training matrix** X\_train shape → **(4088 × 20)**
  + **Test matrix** X\_test shape → **(1022 × 20)**  
    (exactly 80%:20% = 4088 / 5110)

**2. Preprocessing Pipeline**

****

**2-1 Cleaning & Imputation**

|  |  |  |
| --- | --- | --- |
| **Step** | **Detailed Actions** | **Rationale** |
| Remove ID column | Drop id | Prevents data leakage |
| String normalization | Trim whitespace, unify capitalization, map to "Unknown" | Reduces sparsity in one-hot encoding |
| BMI missing values | Median imputation | Median reduces the impact of outliers |

**2-2 Feature Engineering**

* **age\_group** (Child / Adult / Senior) — pd.cut buckets into categorical groups, helping the model interpret age non-linearly.

**2-3 Column Partition & Transformers**

|  |  |  |
| --- | --- | --- |
| **Block** | **Columns (post-processed names)** | **Transformer** |
| Numeric (3) | num\_\_age, num\_\_avg\_glucose\_level, num\_\_bmi | StandardScaler |
| One-hot (12) | cat\_\_work\_type\_\* (5), cat\_\_smoking\_status\_\* (4), cat\_\_age\_group\_\* (3) | OneHotEncoder |
| Binary passthrough (5) | remainder\_\_gender, remainder\_\_hypertension, remainder\_\_heart\_disease, remainder\_\_ever\_married, remainder\_\_Residence\_type | Passthrough |

Final count: 3 + 12 + 5 = **20** features, matching observed matrix shape.

**3. Final Feature List**

1. age (scaled)
2. avg\_glucose\_level (scaled)
3. bmi (scaled/imputed)
4. work\_type: Govt\_job, Never\_worked, Private, Self-employed, children
5. smoking\_status: Unknown, formerly smoked, never smoked, smokes
6. age\_group: Child, Adult, Senior
7. gender (0 Male / 1 Female / 2 Other)
8. hypertension (0 No / 1 Yes)
9. heart\_disease (0 No / 1 Yes)
10. ever\_married (0 No / 1 Yes)
11. Residence\_type (0 Rural / 1 Urban)

**4. Pipeline Reusability**

The preprocess\_stroke\_data() function returns (X\_train, X\_test, y\_train, y\_test, preprocess\_pipeline). The preprocessor step includes get\_feature\_names\_out() for feature order retrieval.

**5. Quality checks**

* Verified one-hot rows sum to 1 per feature family.
* Compared mean / std between train and test; differences < 0.5 σ for all numeric fields.
* Called get\_feature\_names\_out() to lock feature order for reproducibility.

**6. Summary & Implications**

* Data is cleaned, imputed, scaled, and encoded without loss, ready for model training.
* 20-dimensional design matrix ensures consistent input for diverse modeling techniques (Decision Trees, K-means, etc.).
* Train/test splitting guarantees no data leakage.

**Team Member Reflections and Contributions \_ Subin An**

1. **Classification (Data Analysis)**

Prediction of stroke occurrence based on personal health information (0: no stroke, 1: stroke). This allows medical professionals to quickly assess patients' risk of stroke and identify high-risk groups early to take effective precautions.

1. **Correlation Matrix**

Before learning the Decision Tree Classification Model, the correlation between health indicators is analyzed and visualized based on the pre-processed data. This identifies the association between each feature and the target variable, stroke, and provides insights on model learning.

Decision Tree Classification

**[Key libraries and modules]**

* matplotlib.pyplot(plt): a library for data visualization
* Seaborn (sns): Visualize color-based correlation matrix

**[Key Functions and Classes]**

* get\_feature\_names\_out(): return preprocessed feature name
* df.corr(): Calculate correlations between features using Pearson correlation coefficients

**[Key Parameters and Options]**

* columns = feature\_names: Output as preprocessed column name
* df["stroke"] = ytr: Add the target variable, stroke, to the data frame and analyze it together
* **Code**

# DS\_TermProject\_Correlation

from typing import Tuple

import numpy as np

import pandas as pd

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Correlation matrix of numerical feature

# Early detection of high-risk factors for stroke

def correlation\_matrix():

    # Preprocess data

    Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()

    # Transformed feature names from preprocessor pipeline

    feature\_names = pp.named\_steps["preprocessor"].get\_feature\_names\_out()

    # Training data in DataFrame with preprocessed column name

    df = pd.DataFrame(Xtr, columns = feature\_names)

    # Add stroke label

    df["stroke"] = ytr

    # Correlation coefficient for numerical features

    corrmat = df.corr()

    # Visualization heatmap

    plt.figure(figsize=(10, 8))

    sns.heatmap(

        corrmat,

        annot=False, # Number omitted

        cmap="coolwarm", # Red-Blue

        square=True, # Square cell

        linewidths=0.5) # Boundary between cells

    plt.title("Correlation Matrix (after Standardization & Encoding)")

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()

    correlation\_matrix()

**Result :**

텍스트, 스크린샷, 라인, 패턴이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* This confirms that 'cat\_\_age\_group\_senior', 'remainder\_\_hypertension', and 'remainder\_\_heart\_disase' are the top three features that correlate most with the occurrence of the stroke, meaning that if you are older or have high blood pressure and heart disease, you are more likely to have a stroke.

1. **Decision Tree model learning**

Based on the pre-processed data, the information gain (Entropy) was used as the segmentation criterion. This enables binary classification of stroke by learning the Decision Tree model. To enhance the interpretability of the model, the pre-processed feature names were decoded to make them easier for humans to understand when the tree structure was printed in text form, and the normalized feature values were also de-normalized to real values so that each branch condition could be intuitively interpreted.

**[Key libraries and modules]**

* matplotlib.pyplot(plt): Library for tree visualization
* sklearn.tree.\_tree: tree internal structure access object

**[Key Classes and Objects]**

* DecisionTreeClassifier: decision tree model generation class
* model.tree\_: structure and node information of the trained decision tree
* plot\_tree(): output visual structure of the tree
* export\_text(): text-based tree structure output

**[Key function]**

* get\_feature\_names\_out(): return encoded feature name
* model.fit(X, y): Decision Tree Model Training
* predict(X): performing prediction
* score(X, y): Accuracy assessment
* get\_depth(): return the maximum depth of the tree
* feature\_importances\_: Return the split importance of each feature
* tree\_to\_text (node, depth): recursively textualizing the branch structure of the tree
* inverse\_scale (feature, value): Restore normalized values to original values

**[Key parameters]**

* criteria="entropy": Set split criteria based on information gain (default 'gini')
* max\_depth=3: Limit the maximum depth of the tree to prevent overfitting
* random\_state=42: Reproduce results
* class\_names=["0", "1"]: Name class labels when visualizing (0: No stroke, 1: stroke)
* decodedDict: a dictionary that maps encoded feature names into a form that is easy for humans to understand
* **Code**

# DS\_TermProject\_DecisionTree

from typing import Tuple

import numpy as np

import pandas as pd

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.tree import DecisionTreeClassifier

from matplotlib import pyplot as plt

from sklearn.tree import plot\_tree

from sklearn.tree import export\_text

from sklearn.tree import \_tree

# Supervised learning

# Predict stroke using age, avg\_glucose\_level, bmi, hypertension, heart\_disease, ...

def decision\_tree\_classification():

    # Preprocess data

    Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()

    # Transformed feature names from preprocessor pipeline

    feature\_names = pp.named\_steps["preprocessor"].get\_feature\_names\_out()

    # Decision tree with entropy as criterion

    model = DecisionTreeClassifier(criterion = "entropy", # Criteria for division

                                   max\_depth = 3, # Maximum Depth of Tree

                                   random\_state = 42) # Reproduce Result

    model.fit(Xtr, ytr) # Model learning

    y\_predict = model.predict(Xte) # Test data prediction

    # Readable dictionary of decoded transformed feature names

    decodedDict = {

        'num\_\_age': 'age',

        'num\_\_avg\_glucose\_level': 'avg\_glucose\_level',

        'num\_\_bmi': 'bmi',

        'cat\_\_work\_type\_Govt\_job': 'work\_type: Govt\_job',

        'cat\_\_work\_type\_Never\_worked': 'work\_type: Never\_worked',

        'cat\_\_work\_type\_Private': 'work\_type: Private',

        'cat\_\_work\_type\_Self-employed': 'work\_type: Self-employed',

        'cat\_\_work\_type\_children': 'work\_type: children',

        'cat\_\_smoking\_status\_Unknown': 'smoking\_status: Unknown',

        'cat\_\_smoking\_status\_formerly smoked': 'smoking\_status: formerly smoked',

        'cat\_\_smoking\_status\_never smoked': 'smoking\_status: never smoked',

        'cat\_\_smoking\_status\_smokes': 'smoking\_status: smokes',

        'cat\_\_age\_group\_Child': 'age\_group: Child',

        'cat\_\_age\_group\_Adult': 'age\_group: Adult',

        'cat\_\_age\_group\_Senior': 'age\_group: Senior',

        'remainder\_\_gender': 'gender', # 0 = Male, 1 = Female, 2 = Other

        'remainder\_\_hypertension': 'hypertension', # 0 = No, 1 = Yes

        'remainder\_\_heart\_disease': 'heart\_disease', # 0 = No, 1 = Yes

        'remainder\_\_ever\_married': 'ever\_married', # 0 = No, 1 = Yes

        'remainder\_\_Residence\_type': 'Residence\_type' # 0 = Rural, 1 = Urban

    }

    # Decoded feature names for readability

    decoded\_names = []

    for i in feature\_names:

        if i in decodedDict:

            decoded\_names.append(decodedDict[i])

        else:

            decoded\_names.append(i)

    # Restore normalized numerical feature for reverse normalization

    scaler = pp.named\_steps["preprocessor"].named\_transformers\_["num"]

    num\_features = ["age", "avg\_glucose\_level", "bmi"]

    tree = model.tree\_ # Learned decision tree object

    # Convert scaled values back to original values

    def inverse\_scale(feature, value):

        if feature.startswith("num\_\_"):

            real\_feature = feature.split("\_\_")[1]

            idx = num\_features.index(real\_feature)

            return value \* scaler.scale\_[idx] + scaler.mean\_[idx] # Reverse normalization

        else:

            return value # Just use if not numerical feature

    # Recursive function to print decision tree

    def tree\_to\_text(node, depth):

        indent = "  " \* depth # Indent by current depth

        if tree.feature[node] != \_tree.TREE\_UNDEFINED: # Check current node is split node

            name = decoded\_names[tree.feature[node]] # Feature name for split

            threshold = tree.threshold[node] # Scaled value for split

            # Convert scaled values back to original values

            threshold\_value = inverse\_scale(feature\_names[tree.feature[node]], threshold)

            # Left child node when less than scaled value

            print(f"{indent} {name} <= {threshold\_value}:  # normalized: {threshold}")

            # Recursive call to left child node (depth = depth + 1)

            tree\_to\_text(tree.children\_left[node], depth + 1)

            # Right child node when more than scaled value

            print(f"{indent} {name} > {threshold\_value}")

            # Recursive call to right child node (depth = depth + 1)

            tree\_to\_text(tree.children\_right[node], depth + 1)

        else: # Current node is leaf node

            class\_id = tree.value[node].argmax() # Most class at current node

            print(f"{indent} class {class\_id}")

    # Split model based on feature in tree structure

    tree\_to\_text(0, 0) # Start at (node = 0, depth = 0)

    print()

    # Export tree in text format

    # Split model based on feature in tree structure

    normalized\_tree = export\_text(model, feature\_names = decoded\_names)

    print(normalized\_tree)

    print(f"Depth: {model.get\_depth()}")

    print(f"Accuracy: {model.score(Xte, yte)}")

    # Visualization Decision Tree

    # Split model based on feature in tree structure

    plt.figure(figsize  = (20, 10))

    plot\_tree(model,

              feature\_names = decoded\_names,

              class\_names = ["0", "1"], # 0: stroke X, 1: stroke O

              filled = True)

    plt.title("Decision Tree Classification (stroke prediction)")

    # Feature importance that which feature strongly influences stroke

    importances = model.feature\_importances\_

    plt.figure(figsize = (10, 6))

    plt.barh(decoded\_names, importances)

    plt.xlabel("Importance")

    plt.ylabel("Feature")

    plt.title("Feature Importance for Stroke Prediction")

    plt.show()

    return model

if \_\_name\_\_ == "\_\_main\_\_":

    Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()

    decision\_tree\_classification()

* **Result**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 도표, 라인, 평면도이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**2-1) accuracy with various combinations of parameters**

|  |
| --- |
| model = DecisionTreeClassifier(criterion = "entropy", # Criteria for division  max\_depth = 3, # Maximum Depth of Tree  min\_samples\_split = 10, # Minimum of sample to split node  min\_samples\_leaf = 5, # Minimum of sample of leaf node  class\_weight = "balanced", #Balanced between classes  random\_state = 42) # Reproduce Result |
| Accuracy: 0.5802348336594912 |
| model = DecisionTreeClassifier(criterion = "entropy", # Criteria for division                                      max\_depth = 3, # Maximum Depth of Tree                                      random\_state = 42) # Reproduce Result |
| Accuracy: 0.949119373776908 |

* min\_samples\_split=10: Set the number of samples needed to split a node to at least 10 to prevent reckless segmentation if there is insufficient data
* min\_samples\_leaf=5: Set leaf nodes to contain at least 5 samples to suppress the creation of overly small leaf nodes
* These settings rather inhibit tree segmentation, reducing the expressiveness of the model, resulting in performance degradation
* class\_weight="balanced": To correct class imbalances, adjust weights inversely proportional to the number of samples in each class
* However, this can result in trees that are overly sensitive to minority classes (stroke = 1) and reduce overall accuracy (Stroke = 1:249 sample, No Stroke = 0:4861 sample)

1. **Feature Importance**

From the trained models, we derive key characteristics that have a significant impact on the occurrence of strokes. This allows healthcare professionals to make decisions about which health indicators to focus on and diagnose more effectively.

텍스트, 스크린샷, 디스플레이, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

|  |  |  |
| --- | --- | --- |
|  | Correlation matrix | Feature importance |
| Age | Stroke higher with older age | Important as split basis |
| Avg\_glucose\_level | Low correlation | Used for frequent splitting |
| bmi | Little correlation | Used for some splitting |

* Correlation matrix shows linear relationships between variables and is used to identify simple associations between each feature and stroke
* Feature importance is calculated based on the conditional segmentation criteria of the decision tree and is evaluated based on the degree to which it contributes to the improvement of predictive performance

1. **contribution to open-source software**

<https://github.com/Aashvitt/Healtho-Healthcare_Chatbot/blob/main/HealthCare%20ChatBot%20AI%20PROJECT.py>

* In order to recursively output the decoded feature name and the inverse normalized value according to the tree structure, it was implemented by referring to the function 'tree\_to\_code'. In this code, it is defined as 'tree\_to\_text'.

1. **Clustering (Data Analysis & Evaluation of Results)**

By clustering people with similar health information, we analyze the characteristics of each group and design customized stroke prevention strategies based on them. This is an unsupervised learning method, which can discover natural patterns by analyzing data without stroke labels when training models. In addition, the KMeans algorithm gradually optimizes the inter-cluster boundaries by iteratively recalculating the center points. By analyzing the health information of each cluster in this way, high-risk, medium-risk, and low-risk groups can be distinguished, which can be effectively used to establish risk-based intervention strategies.

**1) Elbow Method (Quality Measure)**

As a representative way to determine the appropriate number of clusters (k), the KMeans algorithm is iterated over different numbers of clusters to measure the Within-Cluster Sum of Squares (WCSS). KMeans aims to divide the data in each cluster so that it is as close to the center point as possible, and a smaller WCSS value means higher clustering quality.

Since the id column was removed from the preprocessing process and the unstroke column was also excluded due to the unsupervised learning characteristics, we repeatedly performed it on a total of 10 clusters. Based on repeated WCSS values, KneeLocator is used to automatically detect the elbow point representing the most appropriate number of clusters.

* **Result**

폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

라인, 그래프, 텍스트, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**2) no PCA**

Instead of using PCA, which is a dimension reduction technique, KMeans is performed using the entire original feature. However, for intuitive visualization of the cluster distribution, two features, age and bmi, were selected and used for the 2D plane. The feature combination used at this time can change fluidly depending on user settings.

* **Result**

텍스트, 스크린샷, 다채로움, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 다채로움, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 다채로움, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* The visualized cluster distribution shows a vertically divided structure and is mainly divided around age. This makes it simple to see that the older the BMI, the higher the probability of stroke, but there is a limitation of information loss because the rest of the health indicators are not reflected.

**3) PCA (Improved performance)**

In order to improve the performance of the clustering model and increase the possibility of visual interpretation, PCA was used to minimize the loss of information in high-dimensional features and reduce it to two dimensions. This allows clustering to be performed in a state that reflects all the features in the dataset, and the boundaries between clusters can be clearly revealed to form a more effective cluster structure.

Also, like Decision Tree Classification, cluster-specific features were decoded and output in an easy-to-understand form, and normalized values were also de-normalized, allowing each cluster condition to be interpreted as an actual data criterion.

Numerical features (age, avg\_glucose\_level, bmi) summarize the average values of each cluster, and categorical features (gender, ever\_married, resistance\_type, hypertension, heart\_disase, work\_type, smoking\_status) summarize the typical characteristics of each cluster with the most frequent values. In addition, you can output the stroke incidence of each cluster to identify which cluster is at high risk and identify the group that needs preventive intervention.

**[Key libraries and modules]**

* matplotlib.pyplot(plt): Visualization Library
* kned.KneeLocator:elbowpoint automatic detection of WCSS graphs (pip install kned required)

**[Key Classes and Objects]**

* KMeans: Cluster formation by repeatedly updating central coordinates
* PCA: Compress entire data and reduce it to two of the most important key components

**[Key function]**

* model.fit(X): KMeans model learning
* model.labels\_: the cluster label to which each data belongs
* model.initia\_: return WCSS value
* model.cluster\_centers\_: center coordinates for each cluster
* pca.fit\_transform(X): Dimension to 2D with PCA
* scaler.inverse\_transform(X): restore normalized data to its original value
* df.groupby("cluster"): Group data by cluster
* mode(), mean(), value\_counts(): Statistics per cluster

**[Key parameters]**

* n\_clusters: Number of clusters to create
* random\_state=42: Reproduce results
* max\_iter=k: center coordinate readjustment iteration count
* curve='convex', direction='decreasing': shape the WCSS curve used in KneeLocator
* n\_components=2: When dimensionalized to PCA, dimensionalize to 2D
* hue="cluster": color different depending on cluster number
* decodedDict, label\_decoding\_map: a dictionary that maps preprocessed feature names into a human-readable form
* **Code**

# DS\_TermProject\_KMeans

from typing import Tuple

import numpy as np

import pandas as pd

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from kneed import KneeLocator # pip install kneed

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

* # Find best number of cluster
* def elbow\_method():
* # Preprocess data
* Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()
* # Data point in the cluster from centroid of the cluster
* wcss = [] # within-cluster sum of squares
* # Change number of clusters by number of features (except id, stroke)
* for i in range(1, 11):
* model = KMeans(n\_clusters = i, random\_state = 42) # KMeans
* model.fit(Xtr) # Model learning
* wcss.append(model.inertia\_) # inertia\_ = WCSS
* # Auto finding number of clusters at convex
* # Decreasing WCSS when increasing number of clusters
* clusterNum = KneeLocator(range(1, 11), wcss, curve = "convex", direction = "decreasing")
* print(f"number of clusters (Elbow point): {clusterNum.elbow}")
* # Visualization elbow method
* plt.figure(figsize = (8, 5))
* plt.plot(range(1, 11), wcss)
* plt.title("Elbow method")
* plt.xlabel("Number of clusters")
* plt.ylabel("WCSS")
* plt.xticks(range(1, 11))
* plt.grid(True)
* plt.show()
* return clusterNum
* # Unsupervised learning
* # Grouping based on similar health information without using stroke feature
* # Analysis of high-risk cluster for stroke feature
* def kmeans\_clustering():
* # Preprocess data
* Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()
* # Auto find best cluster number
* clusterNum = elbow\_method()
* # Transformed feature names from preprocessor pipeline
* feature\_names = pp.named\_steps["preprocessor"].get\_feature\_names\_out()
* # Readable dictionary of decoded transformed feature names
* decodedDict = {
* 'num\_\_age': 'age',
* 'num\_\_avg\_glucose\_level': 'avg\_glucose\_level',
* 'num\_\_bmi': 'bmi',
* 'cat\_\_work\_type\_Govt\_job': 'work\_type: Govt\_job',
* 'cat\_\_work\_type\_Never\_worked': 'work\_type: Never\_worked',
* 'cat\_\_work\_type\_Private': 'work\_type: Private',
* 'cat\_\_work\_type\_Self-employed': 'work\_type: Self-employed',
* 'cat\_\_work\_type\_children': 'work\_type: children',
* 'cat\_\_smoking\_status\_Unknown': 'smoking\_status: Unknown',
* 'cat\_\_smoking\_status\_formerly smoked': 'smoking\_status: formerly smoked',
* 'cat\_\_smoking\_status\_never smoked': 'smoking\_status: never smoked',
* 'cat\_\_smoking\_status\_smokes': 'smoking\_status: smokes',
* 'cat\_\_age\_group\_Child': 'age\_group: Child',
* 'cat\_\_age\_group\_Adult': 'age\_group: Adult',
* 'cat\_\_age\_group\_Senior': 'age\_group: Senior',
* 'remainder\_\_gender': 'gender', # 0 = Male, 1 = Female, 2 = Other
* 'remainder\_\_hypertension': 'hypertension', # 0 = No, 1 = Yes
* 'remainder\_\_heart\_disease': 'heart\_disease', # 0 = No, 1 = Yes
* 'remainder\_\_ever\_married': 'ever\_married', # 0 = No, 1 = Yes
* 'remainder\_\_Residence\_type': 'Residence\_type' # 0 = Rural, 1 = Urban
* }
* # Readable value of integer
* label\_decoding\_map = {
* 'gender': {0: 'Male', 1: 'Female', 2: 'Other'},
* 'ever\_married': {0: 'No', 1: 'Yes'},
* 'Residence\_type': {0: 'Rural', 1: 'Urban'},
* 'hypertension': {0: 'No', 1: 'Yes'},
* 'heart\_disease': {0: 'No', 1: 'Yes'},
* }
* # Decoded feature names for readability
* decoded\_names = []
* for i in feature\_names:
* if i in decodedDict:
* decoded\_names.append(decodedDict[i])
* else:
* decoded\_names.append(i)
* # Restore normalized numerical feature
* scaler = pp.named\_steps["preprocessor"].named\_transformers\_["num"]
* num\_features = ["age", "avg\_glucose\_level", "bmi"]
* # Convert scaled values back to original values
* denormalized\_num\_features = []
* for j in num\_features:
* name = "num\_\_" + j
* denormalized\_num\_features.append(name)
* # Optimizing cluster by iterative centroid recalculation
* for k in range(100, 301, 100):
* model = KMeans(n\_clusters = clusterNum.elbow, max\_iter = k) # KMeans
* model.fit(Xtr) # Model learning
* labels = model.labels\_ # Data point in which cluster
* # DataFrame of clustering result
* df\_clustering = pd.DataFrame(Xtr, columns = feature\_names)
* # Convert scaled values back to original values
* df\_clustering[denormalized\_num\_features] = scaler.inverse\_transform(df\_clustering[denormalized\_num\_features])
* # Decoded feature names for readability
* df\_clustering.columns = decoded\_names
* # Cluster number array
* df\_clustering["cluster"] = labels
* # Stroke or not by training data
* df\_clustering["stroke"] = ytr
* print(f"\n Iteration = {k}")
* print("\n < Number of data per cluster >\n", df\_clustering["cluster"].value\_counts())
* # Mean of numerical feature by cluster
* print("\n< Mean value of numerical feature per cluster >\n", df\_clustering.groupby("cluster")[["age", "avg\_glucose\_level", "bmi"]].mean())
* print("\n< Mode value of categorical feature per cluster >")
* binary\_col = ["gender", "ever\_married", "Residence\_type", "hypertension", "heart\_disease"]
* for col in binary\_col:
* if col in df\_clustering.columns:
* mode\_per\_cluster = []
* # Repeat label(cluster number) in df\_clustering
* for cluster in df\_clustering["cluster"].unique():
* # Data in cluster of current label
* cluster\_data = df\_clustering[df\_clustering["cluster"] == cluster]
* # Mode of feature in the cluster
* mode = cluster\_data[col].mode()
* if not mode.empty: # Mode O
* mode\_per\_cluster.append(mode.iloc[0]) # First mode
* else: # Mode X
* mode\_per\_cluster.append("N/A") # Can not compute
* if col in label\_decoding\_map:
* # Integer to readable value
* mode\_per\_cluster = pd.Series(mode\_per\_cluster).map(label\_decoding\_map[col])
* print(f"{col}:") # Current feature
* print(mode\_per\_cluster)
* print()
* onehot\_att = {
* "work\_type": [], "smoking\_status": [], "age\_group": []
* }
* # Check feature name in df\_clustering
* # Save feature in each list
* for att in df\_clustering.columns:
* if att.startswith("work\_type:"):
* onehot\_att["work\_type"].append(att)
* elif att.startswith("smoking\_status:"):
* onehot\_att["smoking\_status"].append(att)
* elif att.startswith("age\_group:"):
* onehot\_att["age\_group"].append(att)
* for group\_name, cols in onehot\_att.items():
* print(f"{group\_name}:") # work\_type, smoking\_status, age\_group
* mode\_labels = (
* df\_clustering[cols] # onehot\_att
* .groupby(df\_clustering["cluster"]).mean() # Mean of onehot feature by cluster
* .idxmax(axis=1) # Feature name in cluster with most mean
* )
* print(mode\_labels)
* print()
* # Mean of stroke by cluster \* 100
* print("< Stroke ratio per cluster (%) >\n", (df\_clustering.groupby("cluster")["stroke"].mean() \* 100))
* # PCA to 2D for clustering visualization
* pca = PCA(n\_components = 2)
* principalComponents = pca.fit\_transform(Xtr) # Model learning
* # PCA in DataFrame
* df\_clustering["PC1"] = principalComponents[:, 0] # First PC
* df\_clustering["PC2"] = principalComponents[:, 1] # Second PC
* # N-dimension centroid tok 2-d centroid
* centroids\_2d = pca.transform(model.cluster\_centers\_)
* print("\n< Centroid coordinate >\n", centroids\_2d)
* # Visualization KMeans
* plt.figure(figsize = (8, 6))
* color = ["red", "green", "blue", "yellow"]
* # hue = "cluster" : Color per cluster
* sns.scatterplot(x = "PC1", y = "PC2", hue = "cluster", data = df\_clustering, palette = color, s = 50)
* # Cluster centroids
* plt.scatter(centroids\_2d[:, 0], centroids\_2d[:, 1], s = 200, c = "black")
* plt.title(f"KMeans Clustering (k = {clusterNum.elbow}, max\_iter = {k})")
* plt.xlabel("Principal Component 1")
* plt.ylabel("Principal Component 2")
* plt.legend(title = "Cluster")
* plt.grid(True)
* plt.show()
* return model, df\_clustering
* if \_\_name\_\_ == "\_\_main\_\_":
* Xtr, Xte, ytr, yte, pp = preprocess\_stroke\_data()
* kmeans\_clustering()
* **Result**

텍스트, 도표, 다채로움, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 도표, 다채로움이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 도표, 다채로움이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **Changing the 'max\_iter' value only changes the cluster number assigned to each cluster, but the overall cluster distribution and characteristics are similar. Therefore, we interpreted the result based on 'max\_iter = 300'.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cluster 0 | Cluster 1 | Cluster 2 | Cluster 3 |
|  |  |  |  |  |
| Age (mean) | 71 | 61 | 8 | 39 |
| Age Group (mode) | Senior | Senior | Child | Adult |
| Glucose (mean) | 89 | 209 | 95 | 91 |
| BMI (mean) | 29 | 33 | 21 | 30 |
| Gender (Mode) | F | M | F | F |
| Ever Married (mode) | O | X | O | O |
| Residence Type (mode) | Urban | Rural | Urban | Urban |
| Hypertension (mode) | X | X | X | X |
| Heart Disease (mode) | X | X | X | X |
| Work Type (mode) | Private | Private | children | Private |
| Smoking Status (mode) | X | X | ? | X |
| Stroke Ratio | 11.3 | 13.7 | 0.2 | 1.7 |

* Cluster 1 is classified as a high-risk group because of its high blood sugar and high BMI. This group urgently needs lifestyle improvements such as diet control and regular exercise, and requires regular MRI/CT tests and blood sugar/blood pressure monitoring.
* Cluster 0 is a group with a high proportion of elderly people, although blood sugar levels are not high, and it is necessary to continuously track their health status through biometric measures every three months.
* Cluster 3's current status is stable, but lifestyle improvements are required to prevent elevated BMI and blood sugar levels.
* Cluster 4 consists mainly of children or younger age groups and is a low-risk group with a relatively low risk of stroke. Therefore, basic health care strategies are appropriate.

1. **short write-up by each team member on what they have learned**

Through this Term Project, we were able to experience the difference between supervised and unsupervised learning firsthand and gain a deeper understanding of each purpose and method of use. In addition, by performing encoding and normalization through preprocessing, as well as decoding and de-normalization for intuitive interpretation, we were able to systematically learn the overall flow of data analysis.

1. **demonstration of additional knowledge excluding learning model**

Stroke is a disease that occurs suddenly and causes serious neurological damage and disability. Prevention and early diagnosis are very important because brain cells that have been damaged once are difficult to recover from.

To this end, MRI and CT are essential imaging tests for stroke diagnosis. CT is the first test to be performed in an emergency because it is quick to check for cerebral hemorrhage and is quick. MRI is useful for diagnosing cerebrovascular blockage and can accurately check fine areas of the brain, helping to determine more accurate conditions and make treatment decisions.