1- Problem

Stroke is a major global health concern, often resulting in death or long-term disability. It can occur without warning, and early detection is critical for improving patient outcomes. This project addresses the urgent need for predictive tools that can assess stroke risk by analyzing key health indicators, including age, hypertension, heart disease, glucose level, and body mass index (BMI). By leveraging data mining methods, we aim to assist healthcare providers with timely, data-driven insights that enable early intervention, reduce complications, and ultimately lower stroke-related mortality rates.

2- Data Mining Task

In this project, we will utilize two primary data mining techniques—classification and clustering—to analyze stroke risk factors and gain a deeper understanding of the data.

(Classification) involves building a predictive model that determines whether a patient is at risk of having a stroke based on various health and demographic features. The classification task focuses on the “stroke” attribute as the target variable, using inputs such as age, gender, hypertension, and glucose level to make accurate predictions.

(Clustering), on the other hand, groups patients based on similar characteristics without considering the “stroke” label. This technique helps uncover hidden patterns and relationships within the data, allowing for the identification of high-risk groups and providing new insights into the factors associated with stroke.

3-Data

Source of DataSet :

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

General information about the dataset:

Number of Attributes: 12

Number of Objects: 5110

Class lable : Stroke

```
import pandas as pd
from tabulate import tabulate

df = pd.read_csv('Stroke_Datas.csv')


data = [
    ["id", "Patient ID", "nominal"],
    ["gender", "Gender (Male or Female)", "binary"],
    ["age", "Age", "numric"],
    ["hypertension", "Hypertension (0 = No, 1 = Yes)", "binary"],
    ["heart_disease", "Heart disease (0 = No, 1 = Yes)", "binary"],
    ["ever_married", "Has the person ever been married?", "binary"],
    ["work_type", "Type of work (Private, Self-employed, Government, etc.)", "nominal"],
    ["Residence_type", "Type of residence (Rural or Urban)", "binary"],
    ["avg_glucose_level", "Average blood glucose level", "numric"],
    ["bmi", "Body Mass Index (with missing values)", "numric"],
    ["smoking_status", "Smoking status (Never smoked, Former smoker, Current smoker, Unknown)", "nominal"],
    ["stroke", "Has the person had a stroke? (0 = No, 1 = Yes)", "binary"]
]


print(tabulate(data, headers=["Attribute", "Description", "Type"]))
```

```
Attribute          Description                                                        Type
-----------------  -----------------------------------------------------------------  -------
id                 Patient ID                                                         nominal
gender             Gender (Male or Female)                                            binary
age                Age                                                                numric
hypertension       Hypertension (0 = No, 1 = Yes)                                     binary
```

```
heart_disease      Heart disease (0 = No, 1 = Yes)                                    binary
ever_married       Has the person ever been married?                                  binary
work_type          Type of work (Private, Self-employed, Government, etc.)            nominal
Residence_type     Type of residence (Rural or Urban)                                 binary
avg_glucose_level  Average blood glucose level                                        numric
bmi                Body Mass Index (with missing values)                              numric
smoking_status     Smoking status (Never smoked, Former smoker, Current smoker, Unknown)  nominal
stroke             Has the person had a stroke? (0 = No, 1 = Yes)                     binary
```

Show missing value :

```
missing_values = df.isna().sum()
print ('missing values in each column :')
print (missing_values)


print (" \n total number of missing value " ,missing_values.sum())
```

```
missing values in each column :
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64

 total number of missing value  201
```

we have missing value in column (bmi) = 201

statistical measures for numeric attributes :

```
numeric_columns = ["age", "avg_glucose_level", "bmi"]

summary = df[numeric_columns].describe().T

summary = summary.rename(columns={
    "min": "Min",
    "25%": "1st Qu",
    "50%": "Median",
    "mean": "Mean",
    "75%": "3rd Qu",
    "max": "Max",
    "std": "Standard Deviation"
})

print(summary)
```

```
                    count        Mean  Standard Deviation    Min  1st Qu  \
age                5110.0   43.226614           22.612647   0.08  25.000
avg_glucose_level  5110.0  106.147677           45.283560  55.12  77.245
bmi                4909.0   28.893237            7.854067  10.30  23.500

                   Median  3rd Qu     Max
age                45.000   61.00   82.00
avg_glucose_level  91.885  114.09  271.74
bmi                28.100   33.10   97.60
```

Age Patients’ ages range from 0.08 to 82 years, with a mean of 43.2 and a median of 45, indicating a sample skewed slightly toward older adults. Around 50% of patients are 45 years or older, while 25% are younger than 25 years, placing a significant portion of the population in the middle-aged to elderly range. The minimum value (0.08) is suspicious and may indicate a data entry error, which will be addressed during preprocessing.

Average Glucose Level Glucose levels range from 55.12 mg/dL to 271.74 mg/dL, with a mean of 106.15 and a median of 91.88 mg/dL.

IQR (Interquartile Range): 114.09 - 77.25 = 36.84 mg/dL This suggests most individuals have glucose levels within a moderate range, although some extreme values (above 200 mg/dL) indicate potential outliers, which may correspond to patients with uncontrolled diabetes or

metabolic disorders.

Body Mass Index (BMI) BMI values range from 10.3 to 97.6, with a mean of 28.89 and a median of 28.1.

IQR: 33.10 - 23.50 = 9.6 These values indicate that the majority of patients are either overweight or obese, though the presence of very high BMI values (e.g., above 60) and very low values (e.g., 10.3) point to possible data inconsistencies or rare medical cases. These extremes may need verification or treatment as outliers during data preprocessing.

📐 Calculation Reference Interquartile Range (IQR) = Q3 − Q1

Age IQR = 61.0 − 25.0 = 36.0

Glucose IQR = 114.09 − 77.25 = 36.84

BMI IQR = 33.10 − 23.50 = 9.6

IQR is useful for understanding where the central bulk of the data lies and identifying outliers.

```
numeric_columns = ["age", "avg_glucose_level", "bmi"]

variance_values = df[numeric_columns].var()

print("Variance for Numeric Variables:")
print(variance_values)
```

```
Variance for Numeric Variables:
age                  511.331792
avg_glucose_level   2050.600820
bmi                   61.686364
dtype: float64
```

Variance Analysis for Numeric Variables :

1- Age: Moderate variance of 511.33 indicates a moderate level of variability in patient ages. This aligns with the wide age range in the dataset, suggesting a diverse population that includes both younger and older individuals.

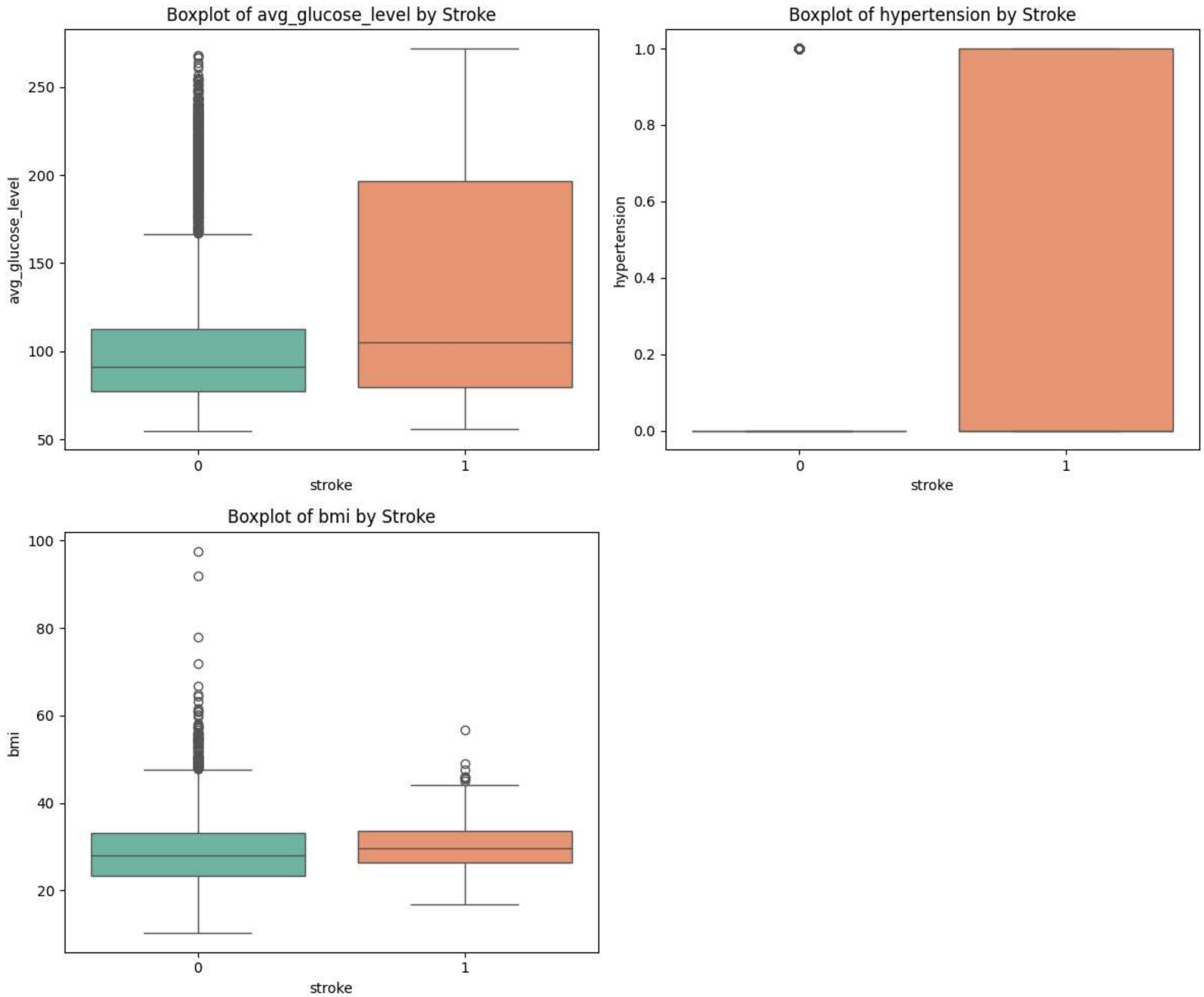
2- Average Glucose Level: A high variance of 2050.60 suggests significant differences in blood glucose levels among patients. This could be due to different dietary habits, underlying health conditions like diabetes, or the presence of outliers that require further investigation.

3- BMI (Body Mass Index): Variance of 61.69 indicates noticeable variation in BMI values across patients. This suggests that the dataset includes individuals with a wide range of body compositions, from underweight to obese. The presence of extreme values may contribute to this high variance.

boxplot

```
import matplotlib.pyplot as plt
import seaborn as sns

# List of columns to plot
cols_to_plot = ['avg_glucose_level', 'hypertension','bmi']

plt.figure(figsize=(12, 10))

# Creating boxplots for each selected column
for i, col in enumerate(cols_to_plot, 1):
    plt.subplot(2, 2, i)  # Adjusting layout (2 rows, 2 columns)
    sns.boxplot(x=data['stroke'], y=data[col], palette='Set2')
    plt.title(f"Boxplot of {col} by Stroke")

plt.tight_layout()
plt.show()
```

```
<ipython-input-12-09e3839096f7>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x=data['stroke'], y=data[col], palette='Set2')
<ipython-input-12-09e3839096f7>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x=data['stroke'], y=data[col], palette='Set2')
<ipython-input-12-09e3839096f7>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `leg

  sns.boxplot(x=data['stroke'], y=data[col], palette='Set2')
```



1. Boxplot of Hypertension by Stroke: Stroke cases (orange) predominantly have hypertension (value 1), indicating that hypertension is a strong contributing factor to stroke risk.

The absence of hypertension (value 0) aligns more frequently with non-stroke cases (green), highlighting the predictive significance of hypertension for stroke occurrences.

2. Boxplot of Avg Glucose Level by Stroke: Individuals with stroke (orange) tend to have significantly higher glucose levels compared to those without a stroke (green).

The median glucose level for stroke cases is elevated beyond 150 mg/dL, while it is closer to 100 mg/dL for non-stroke cases.

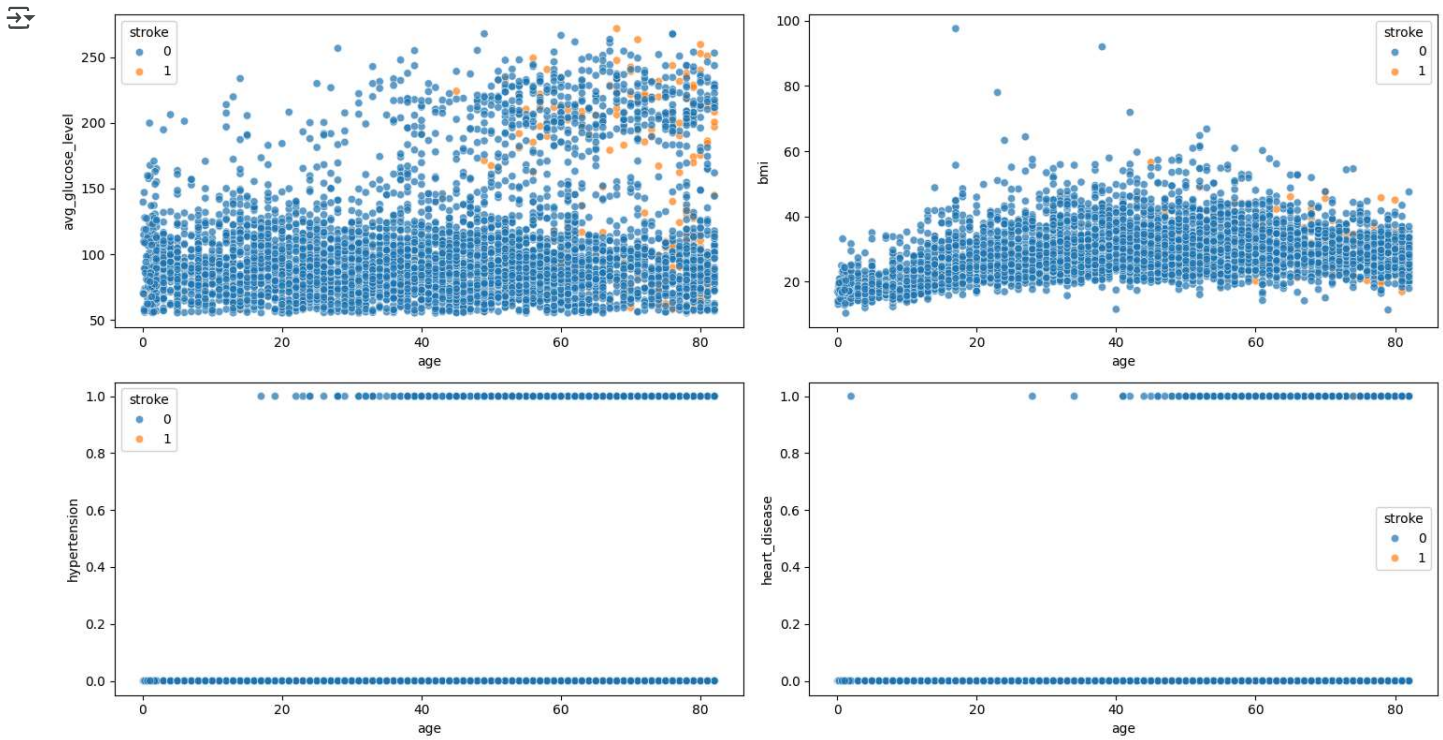
Outliers for non-stroke individuals show occasional spikes above 200 mg/dL, but are much more prevalent for stroke patients, suggesting a strong relationship between elevated glucose levels and stroke risk.

3. Boxplot of BMI by Stroke: BMI levels for individuals with and without strokes show a wide distribution, but the median BMI values for both groups are quite similar, around 28-30.

There are notable outliers in both groups, with BMI levels exceeding 60 in non-stroke individuals. This suggests that while BMI might play a role in overall health, it may not be as strong an independent predictor for stroke.

scatter plot

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


data = pd.read_csv("Stroke_Datas.csv")


fig, axes = plt.subplots(2, 2, figsize=(15, 8))
sns.scatterplot(data=data, x='age', y='avg_glucose_level', hue='stroke', alpha=0.7, ax=axes[0][0])
sns.scatterplot(data=data, x='age', y='bmi', hue='stroke', alpha=0.7, ax=axes[0][1])
sns.scatterplot(data=data, x='age', y='hypertension', hue='stroke', alpha=0.7, ax=axes[1][0])
sns.scatterplot(data=data, x='age', y='heart_disease', hue='stroke', alpha=0.7, ax=axes[1][1])


plt.tight_layout()
plt.show()
```



Age vs. Avg Glucose Level:

Stroke cases (orange points) are strongly associated with elevated glucose levels, particularly among older individuals. Higher glucose levels seem to play a key role in stroke prediction across all age groups but become especially prominent in individuals aged 50 and above.

Age vs. BMI:

BMI values are widely dispersed across all age groups. Stroke occurrences are seen both in individuals with normal and higher BMI levels, indicating BMI may not be a strong standalone predictor for strokes.

Age vs. Hypertension:

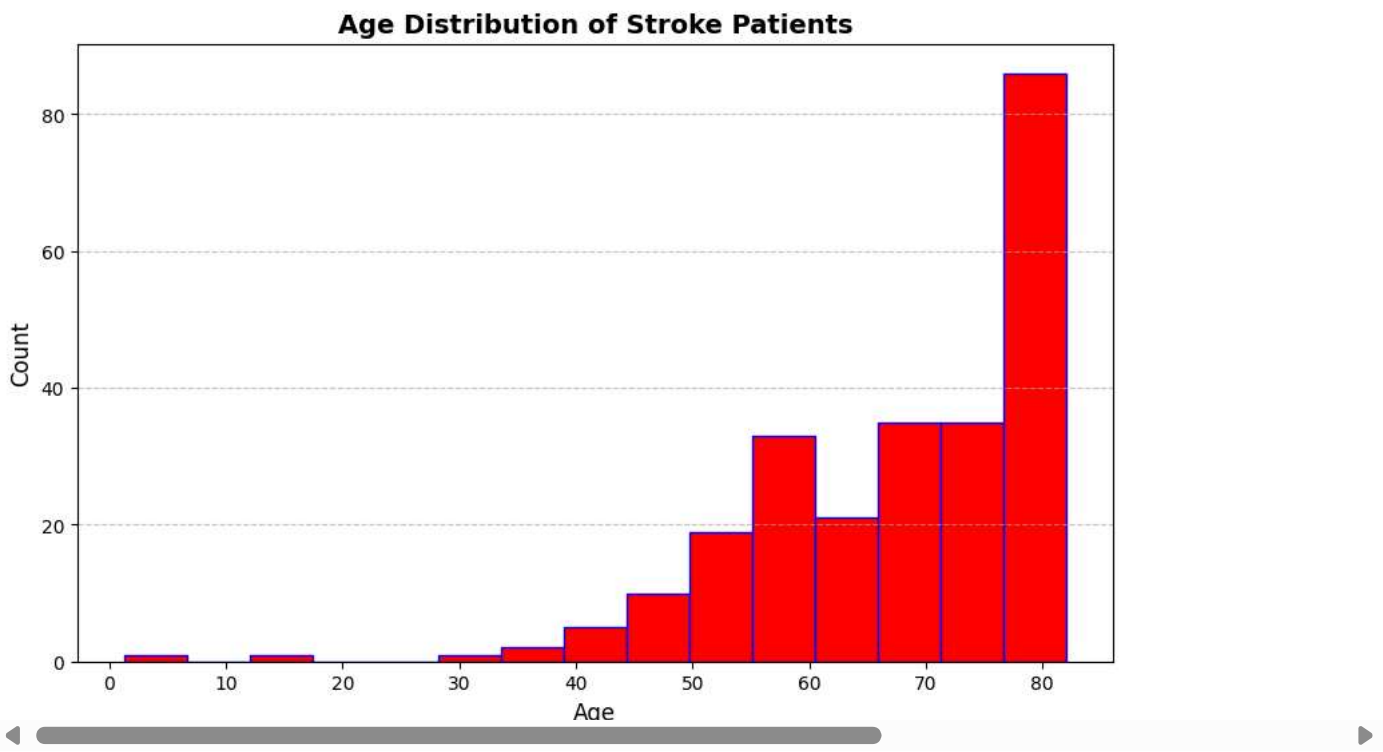
Hypertension (value = 1) is more frequently seen in older individuals. There is a noticeable cluster of stroke cases among hypertensive patients, suggesting hypertension as a contributing factor, particularly in older adults.

Age vs. Heart Disease:

While some stroke cases are associated with heart disease, many strokes occur even without it (heart disease = 0), indicating a less direct relationship. The presence of heart disease becomes a more critical factor for predicting strokes in the older population.

pie chart :

```
import pandas as pd
import matplotlib.pyplot as plt

stroke_ages = df[df['stroke'] == 1]['age']

plt.figure(figsize=(10, 6))
plt.hist(stroke_ages, bins=15, color='red', edgecolor='blue')

plt.title('Age Distribution of Stroke Patients', fontsize=14, fontweight='bold')
plt.xlabel('Age', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



The distribution is right-skewed (positively skewed).

The majority of individuals are elderly, especially in the 80+ age group, which has the highest count.

The lower age ranges (0–30) have very few entries.

The data suggests a strong imbalance in age representation, with a concentration among older adults.

Possible Preprocessing Techniques: Binning / Discretization You could categorize age into groups such as:

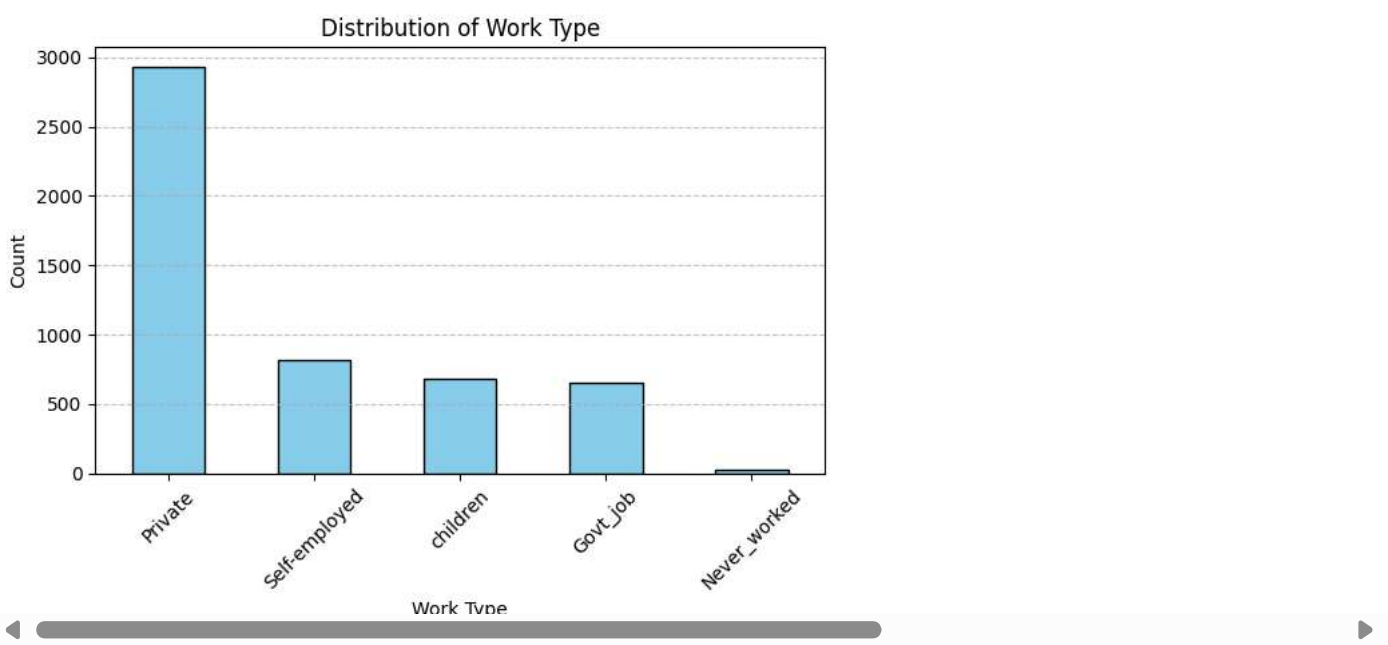
Children (0–18)

Adults (19–60)

Elderly (61+)

Bar Plot

```
df['work_type'].value_counts().plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Distribution of Work Type')
plt.xlabel('Work Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



The "Private" sector has the highest number of individuals, with nearly 3000 entries.

"Self-employed", "children", and "Govt_job" have relatively balanced distributions, each with around 600–800 entries.

The "Never_worked" category has a very small number of entries, almost negligible compared to the others.

This plot reveals a clear class imbalance in the "work_type" attribute.

The overrepresentation of "Private" sector and underrepresentation of "Never_worked" may affect the performance of any predictive models, especially if "work_type" is a significant feature.

These insights suggest the need to consider techniques such as data balancing .

4- Data preprocessing:

- Detecting the outliers:

```
import numpy as np
import pandas as pd

df = pd.read_csv("Stroke_Datas.csv")

num_colums = ["bmi", "avg_glucose_level","age"]

Q1 = df[num_colums].quantile(0.25)
Q3 = df[num_colums].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
outliers = ((df[num_colums] < lower_bound) | (df[num_colums] > upper_bound)).sum()
print("Number of outliers per column (IQR Method):")
print(outliers)

print("------")
df_cleaned = df[~((df[num_colums] < lower_bound) | (df[num_colums] > upper_bound)).any(axis=1)]
print(f"Number of rows before removing outliers: {df.shape[0]}")
```

```
Number of outliers per column (IQR Method):
bmi                 110
avg_glucose_level    627
age                    0
dtype: int64
------
Number of rows before removing outliers: 5110
```

The Interquartile Range (IQR) method is used to detect outliers in numerical data.

The IQR is calculated as the difference between the 75th percentile (Q3) and the 25th percentile (Q1): IQR = Q3 - Q1

The upper bound is defined as: Upper Bound = Q3 + (1.5 × IQR)

The lower bound is defined as: Lower Bound = Q1 - (1.5 × IQR)

Any data points that fall outside these bounds are considered outliers.

Using this method, the number of outliers detected in each column is:

bmi: 110 outliers

avg_glucose_level: 627 outliers

age: 0 outliers

- Data Transformation:

1. Encoding:

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

df = pd.read_csv("Stroke_Datas.csv")

print("\n Display a sample of the original data:")
print(df.head())
label_encoder = LabelEncoder()
df['gender'] = label_encoder.fit_transform(df['gender'])
df['ever_married'] = label_encoder.fit_transform(df['ever_married'])
df['smoking_status'] = label_encoder.fit_transform(df['smoking_status'])
df['work_type'] = label_encoder.fit_transform(df['work_type'])
df['Residence_type'] = label_encoder.fit_transform(df['Residence_type'])


print("\n Display a sample after encoding:")
print(df.head())
```

```
 Display a sample of the original data:
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

        work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0         Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2         Private          Rural             105.92  32.5     never smoked
3         Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked
```

```
   stroke
0       1
1       1
2       1
3       1
4       1

 Display a sample after encoding:
      id  gender   age  hypertension  heart_disease  ever_married  work_type  \
0   9046       1  67.0             0              1             1          2
1  51676       0  61.0             0              0             1          3
2  31112       1  80.0             0              1             1          2
3  60182       0  49.0             0              0             1          2
4   1665       0  79.0             1              0             1          3

   Residence_type  avg_glucose_level   bmi  smoking_status  stroke
0               1             228.69  36.6               1       1
1               0             202.21   NaN               2       1
2               0             105.92  32.5               2       1
3               1             171.23  34.4               3       1
4               0             174.12  24.0               2       1
```

2. Normalization:

Here in the Normalization method, we normalize the attributes and unify their scale since the range for each attribute is quite different, this method helps us to format all the values in the dataset and facilitates the analysis process.

```
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("Stroke_Datas.csv")

# Define the numerical columns that we will normalize
numeric_cols = ['avg_glucose_level', 'bmi']
print("\n Display a sample of the original data::")
print(df.head())
scaler = MinMaxScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
print("\n Show sample data after conversion:")

print(df[numeric_cols].describe())
```

```
 Display a sample of the original data::
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

      work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0       Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2       Private          Rural             105.92  32.5     never smoked
3       Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1

 Show sample data after conversion:
       avg_glucose_level          bmi
count        5110.000000  4909.000000
mean            0.235563     0.212981
std             0.209046     0.089966
min             0.000000     0.000000
25%             0.102137     0.151203
50%             0.169721     0.203895
75%             0.272228     0.261168
max             1.000000     1.000000
```

3. Discretization:

```
print("\n Display a sample of the original data::")
print(df.head())
df = pd.read_csv("Stroke_Datas.csv")

bins = [0, 30, 60, 100]
age_labels = ['Young', 'Middle-aged', 'Senior']

print("\n Show sample data after conversion:")


df['age_group'] = pd.cut(df['age'], bins=bins, labels=age_labels, right=False)

print(df['age_group'].value_counts())

print(df[['age', 'age_group']].head(10))
```

```
 Display a sample of the original data::
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

       work_type Residence_type  avg_glucose_level       bmi   smoking_status  \
0        Private          Urban           0.801265  0.301260  formerly smoked
1  Self-employed          Rural           0.679023       NaN     never smoked
2        Private          Rural           0.234512  0.254296     never smoked
3        Private          Urban           0.536008  0.276060           smokes
4  Self-employed          Rural           0.549349  0.156930     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1

 Show sample data after conversion:
age_group
Middle-aged    2219
Young          1515
Senior         1376
Name: count, dtype: int64
    age    age_group
0  67.0       Senior
1  61.0       Senior
2  80.0       Senior
3  49.0  Middle-aged
4  79.0       Senior
5  81.0       Senior
6  74.0       Senior
7  69.0       Senior
8  59.0  Middle-aged
9  78.0       Senior
```

In this discretization method, we transform continuous numerical values into categorical groups to enhance interpretability and simplify data analysis.

For the age column, individuals are grouped into three age categories:

Young (0–34 years)

Middle-aged (35–64 years)

Senior (65+ years)

5- Data Mining Techniques

In this project, we applied both supervised and unsupervised data mining techniques: classification and clustering, to analyze the dataset and uncover meaningful insights.

Classification

We used the DecisionTreeClassifier from the sklearn.tree module to predict the probability of stroke (stroke column) based on multiple demographic and medical features.

The dataset was divided into training and testing subsets using train_test_split(). The model was trained using the entropy criterion to determine information gain and generate decision splits accordingly. After training, we evaluated its performance using a confusion matrix, visualized with ConfusionMatrixDisplay.

This method was selected because it provides interpretable results and handles both numerical and categorical data without the need for feature scaling.

Clustering

We applied K-means clustering using the KMeans algorithm from sklearn.cluster to uncover hidden patterns within the dataset. Before clustering, we normalized the features using StandardScaler to ensure consistent scale across all variables.

To determine the optimal number of clusters (K), we used:

• **Elbow Method**: We calculated the Within-Cluster Sum of Squares (WSS) for K values from 2 to 10.

• **KneeLocator**: From the kneed library, it was used to automatically identify the elbow point on the WSS plot, indicating the optimal K.

• **Silhouette Score**: To validate the quality of the clusters, we calculated the silhouette score for each K, helping to confirm the selection made by the elbow method.

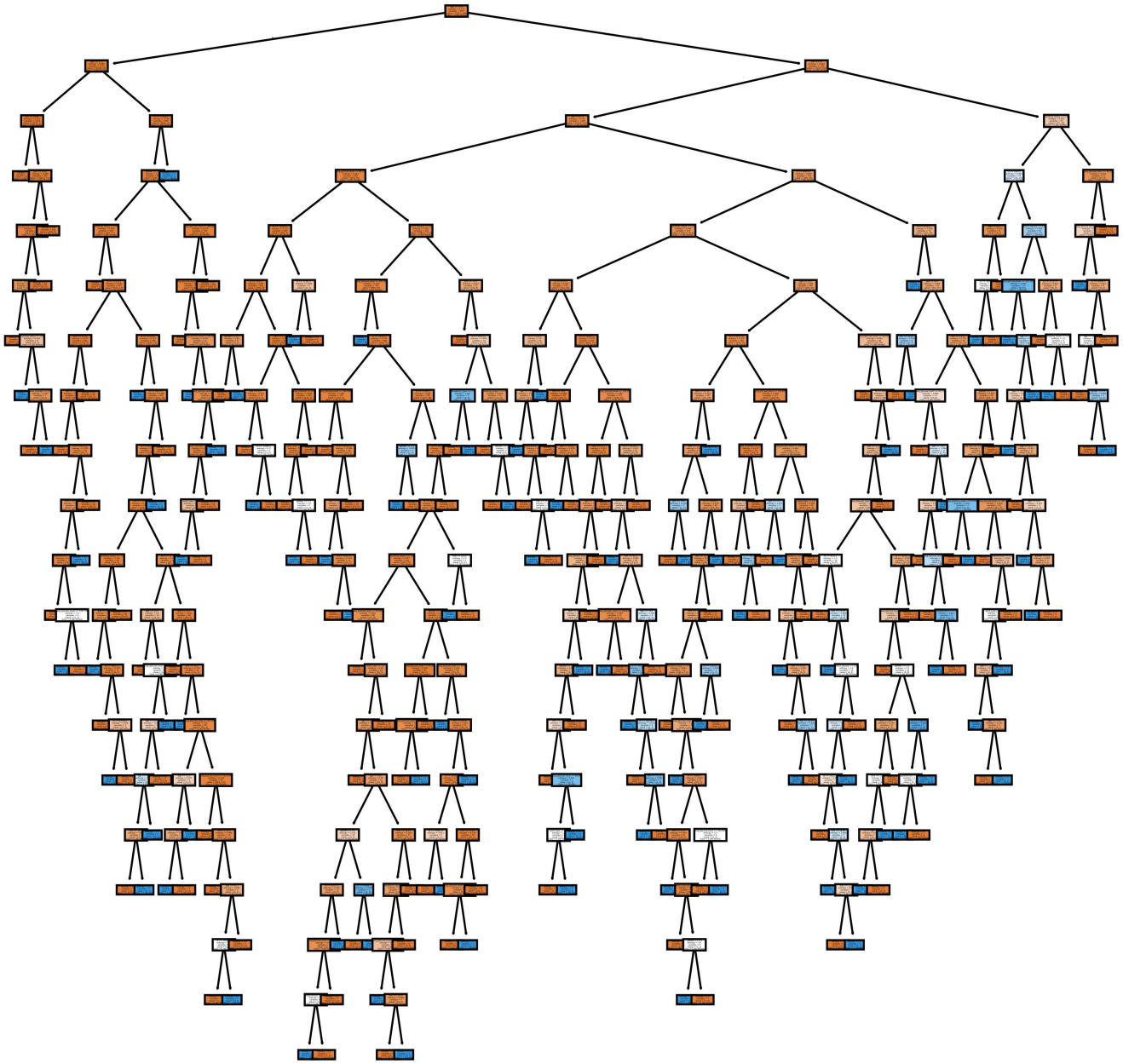
This combination of techniques allowed us to determine a meaningful number of clusters and better understand the internal structure of the data without relying on the target label.

6-Evaluation and Comparison

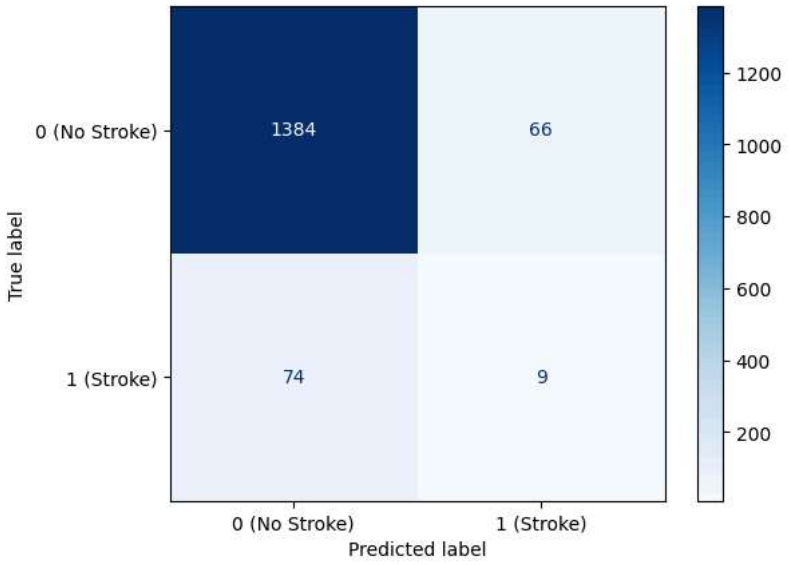
part 1: Classification

- Classification [70% training, 30% testing] Information Gain:

- **decision tree**

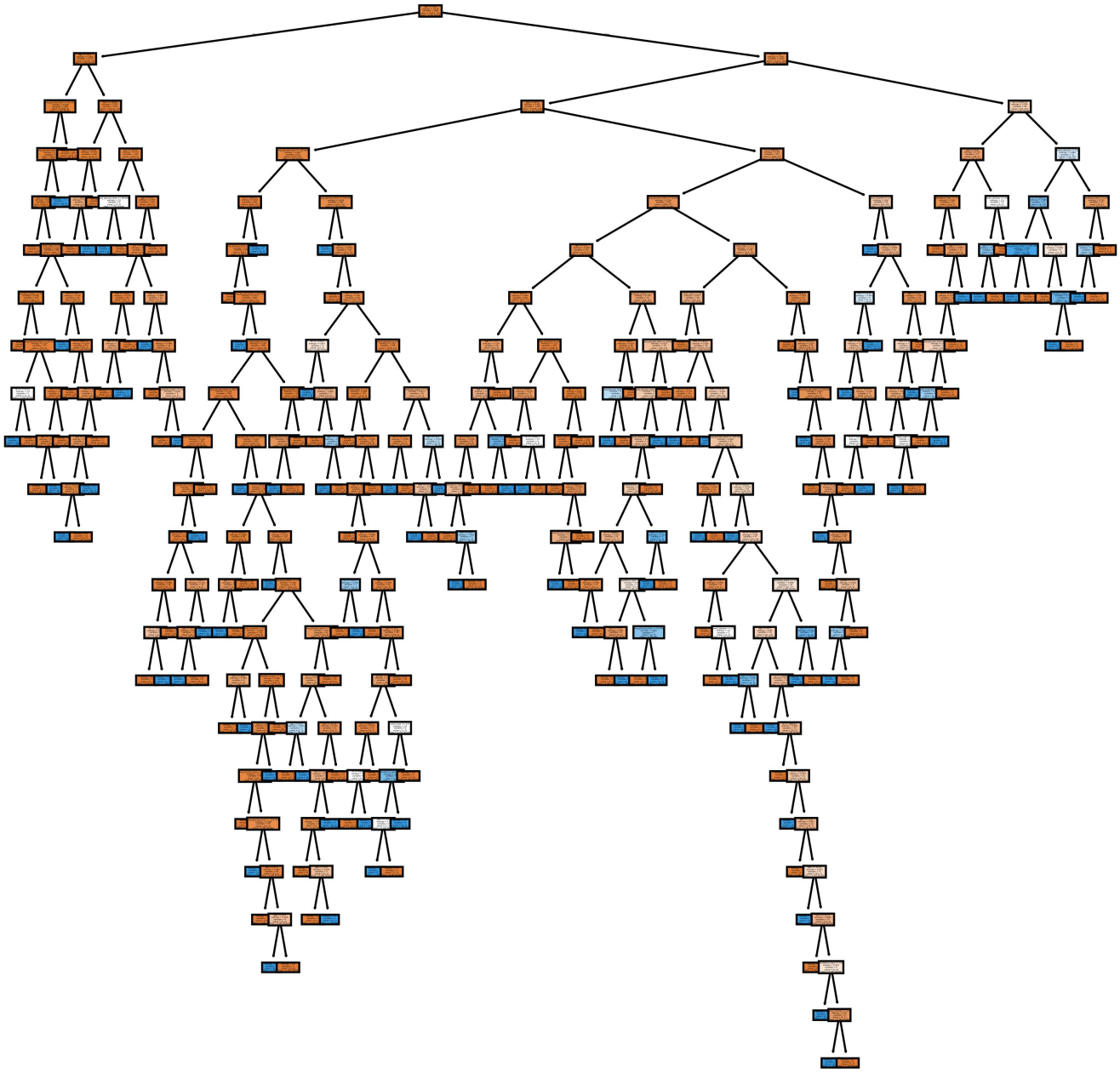


- **confusion matrix**

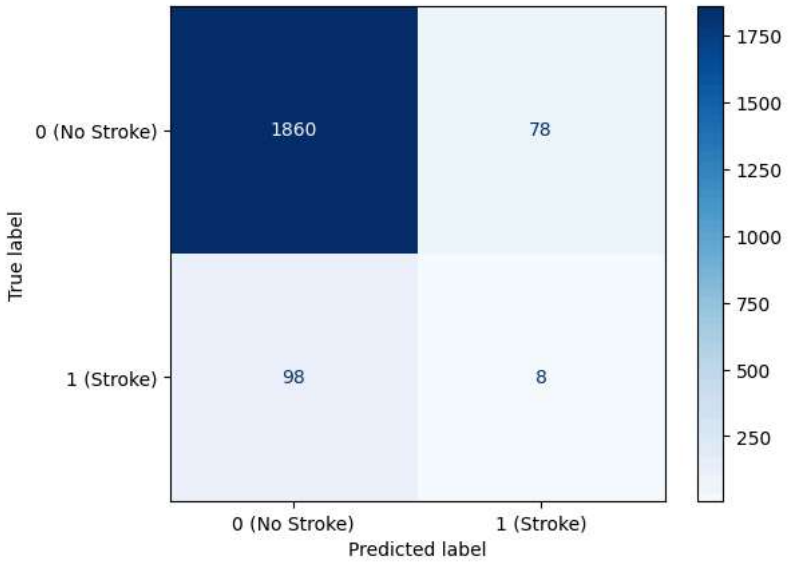


- Classification [60% training, 40% testing] Information Gain:

- **decision tree**

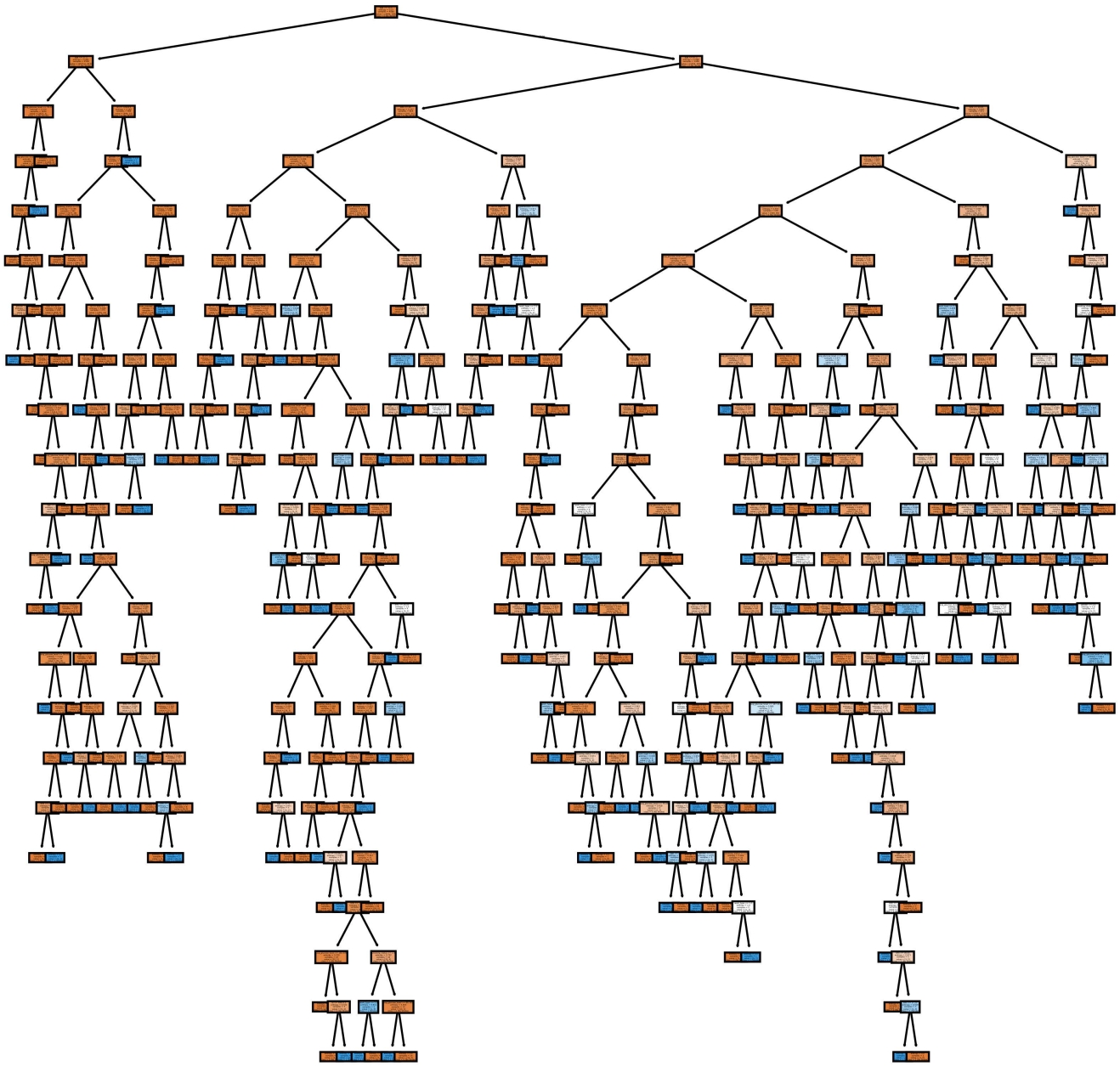


- **confusion matrix**

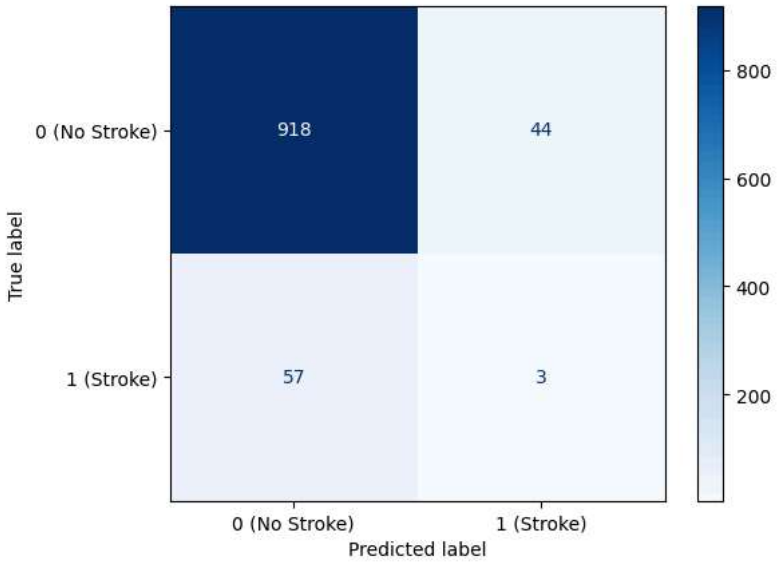


- Classification [80% training, 20% testing] Information Gain:

- **decision tree**



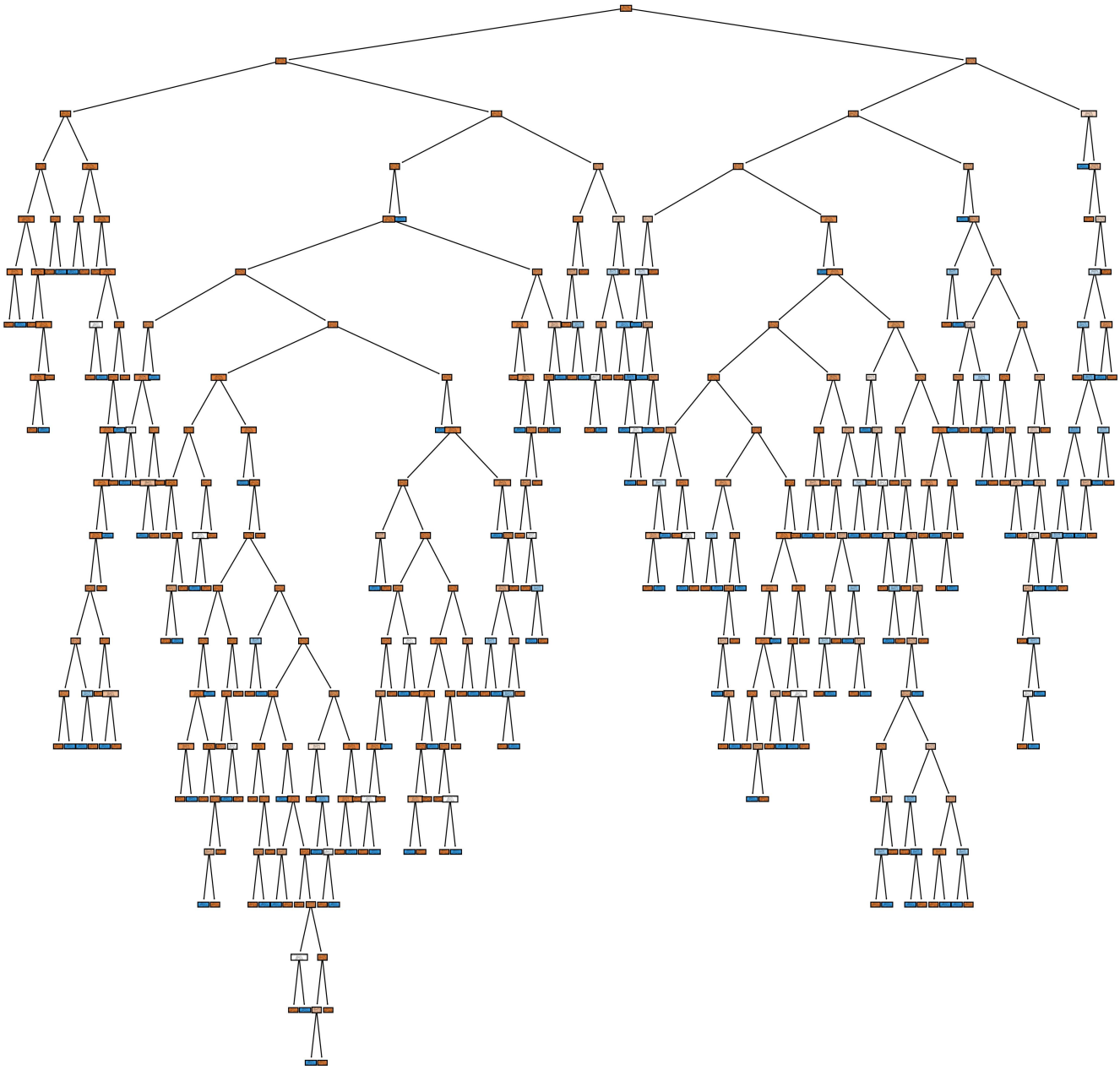
- **confusion matrix**



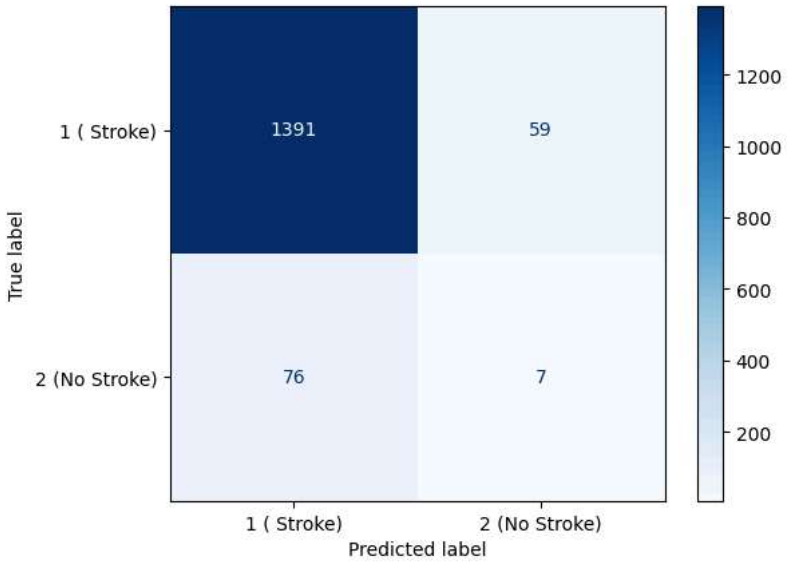
Mining Task	*Comparison Criteria*	*Value*
	70% Training 30% Test data	
	Accuracy:	90.86%
	Error Rate:	9.13%
	Sensitivity:	10.84%
	Specificity:	95.44%
	Precision:	12.00%
	60% Training 40% Test data	
	Accuracy:	91.38%
Classification for	*Error Rate:*	8.61%
Information Gain	*Sensitivity:*	7.54%
	Specificity:	95.97%
	Precision:	9.30%
	80% Training 20% Test data	
	Accuracy:	90.11%
	Error Rate:	9.88%
	Sensitivity:	5.00%
	Specificity:	95.42%
	Precision:	6.38%

- Classification [70% training, 30% testing] Gini Index:

- **decision tree**

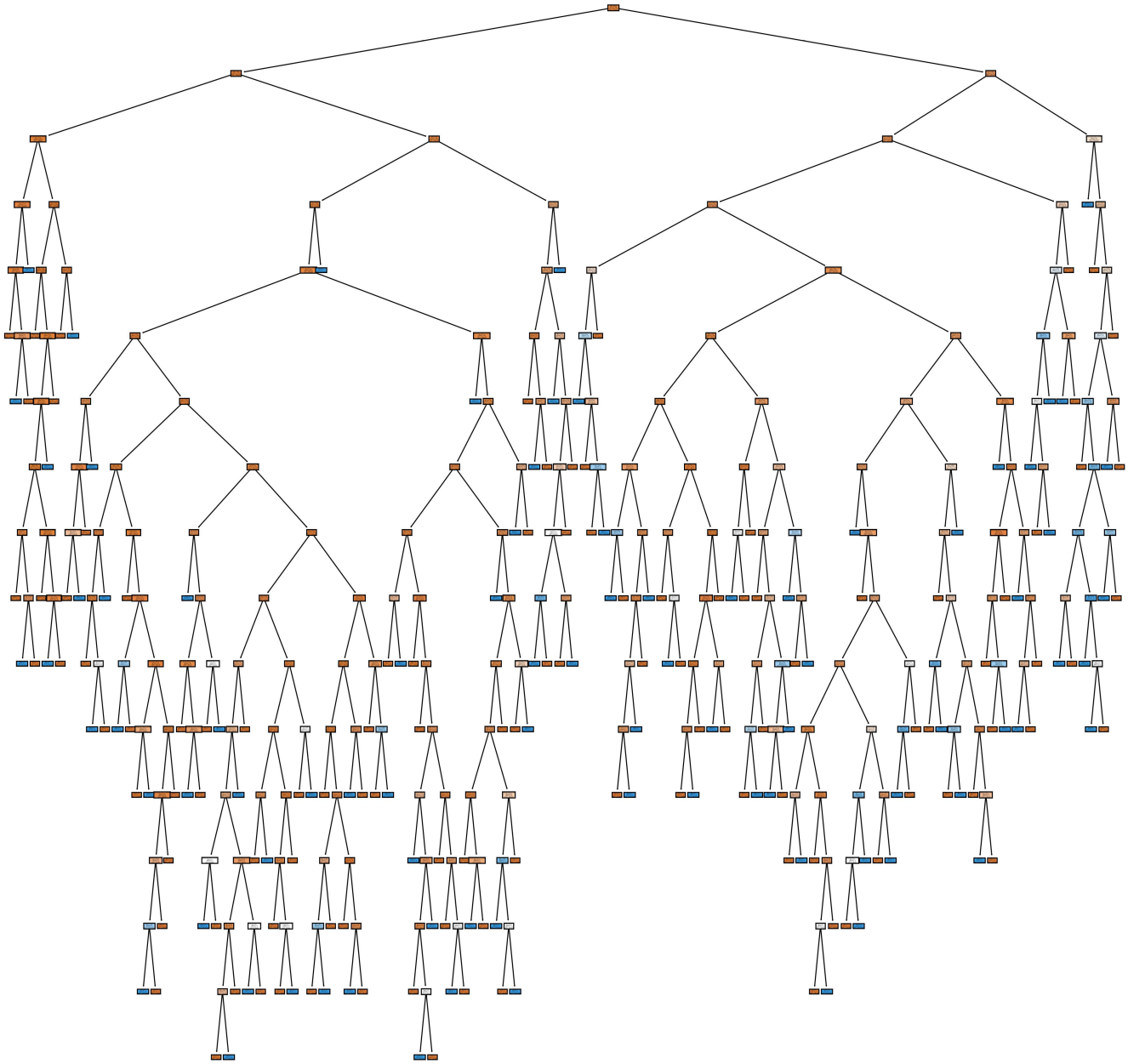


- **confusion matrix**

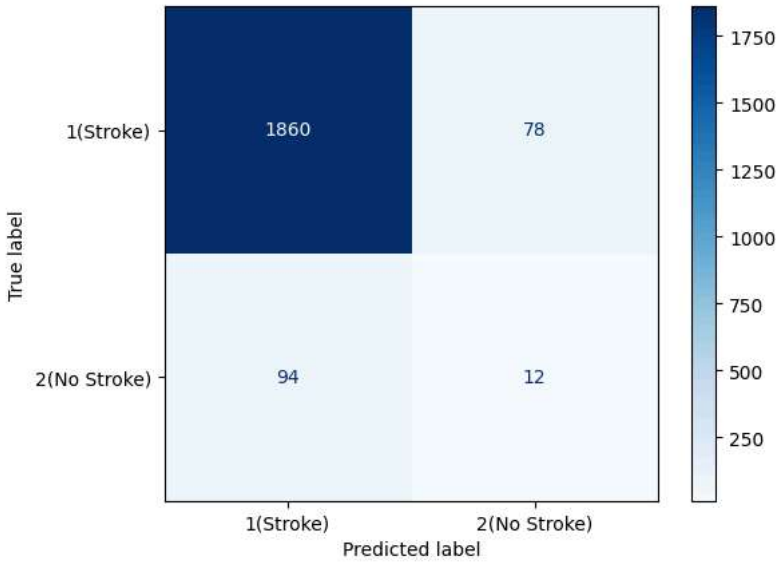


- Classification [60% training, 40% testing] Gini Index:

- **decision tree**

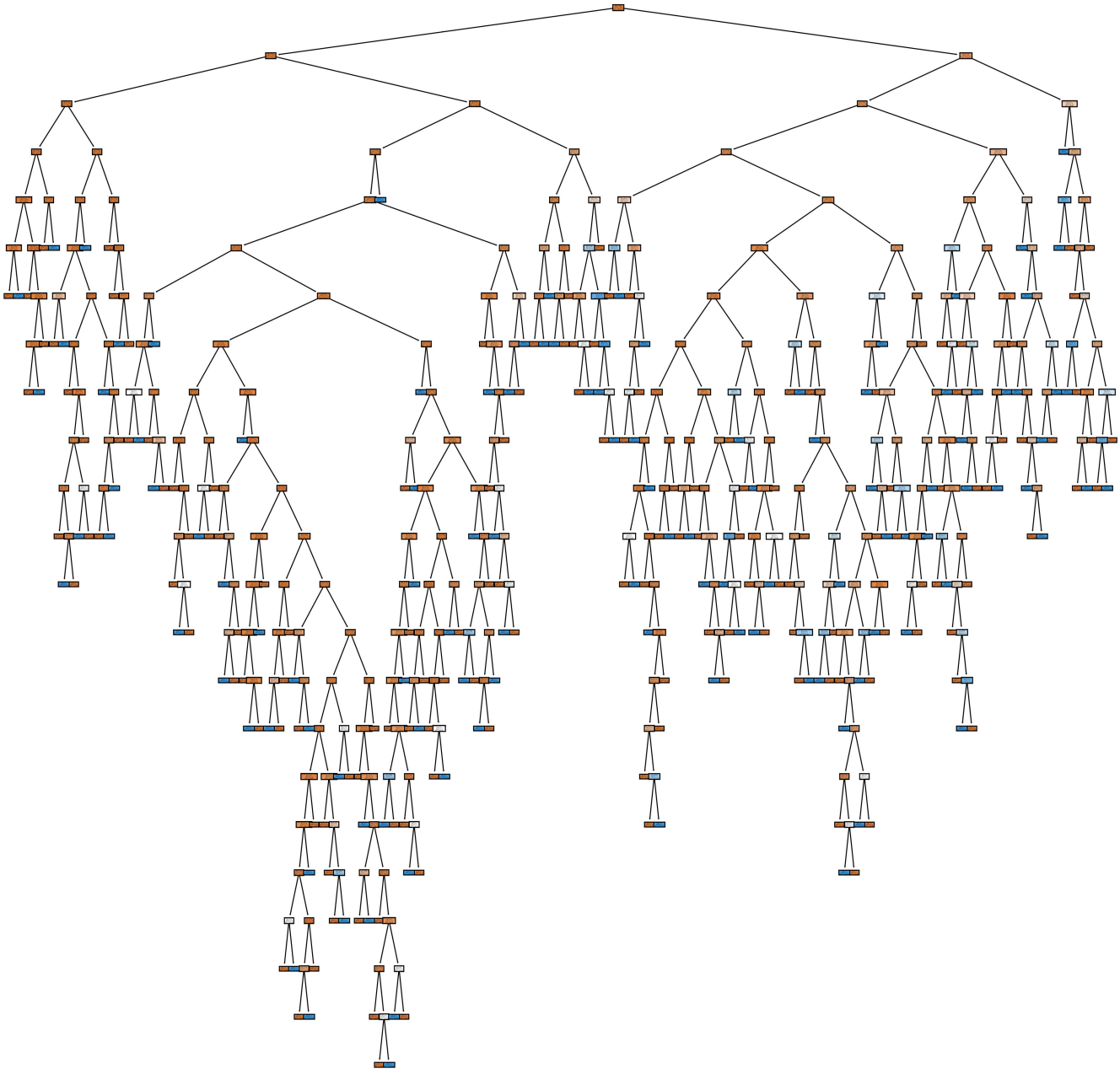


- **confusion matrix**

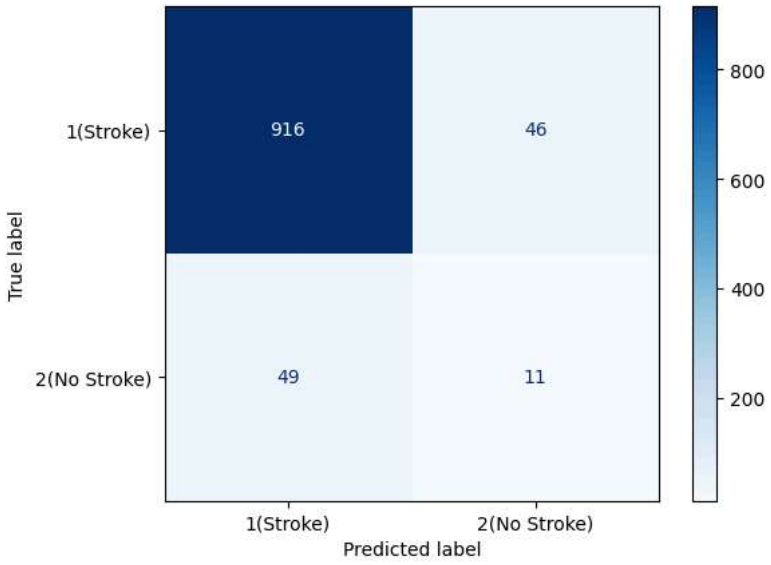


- Classification [80% training, 20% testing] Gini Index:

- **decision tree**



- **confusion matrix**



Mining Task	*Comparison Criteria*	*Value*
	70% Training 30% Test data	
	Accuracy:	90.90%
	Error Rate:	9.03%
	Sensitivity:	13.33%
	Specificity:	95.21%
	Precision:	19.29%
	60% Training 40% Test data	
	Accuracy:	91.49%
Classification for	*Error Rate:*	8.51%
Gini index	*Sensitivity:*	11.33%
	Specificity:	95.97%
	Precision:	13.33%
	80% Training 20% Test data	
	Accuracy:	91.49%
	Error Rate:	8.55%
	Sensitivity:	8.43%
	Specificity:	95.93%
	Precision:	10.60%

Analysis of Results

1. **Accuracy**:
   - Both models maintain high accuracy across all splits.
   - **Gini Index** achieves slightly better accuracy (91.49%) at the 60%-40% split compared to **Information Gain** (91.38%).
2. **Error Rate**:
   - **Gini Index** has a slightly lower error rate (8.51%) at the 60%-40% split versus Information Gain (8.61%).
   - The difference is small but in favor of Gini Index.
3. **Sensitivity**:
   - **Gini Index** performs better in sensitivity across all splits compared to Information Gain.
   - For example, at the 70%-30% split, Gini Index has 13.33% sensitivity compared to 10.84% for Information Gain.
4. **Specificity**:
   - **Information Gain** and **Gini Index** are comparable in specificity, both achieving around 95.9% at the 60%-40% split.
5. **Precision**:
   - **Gini Index** outperforms Information Gain in precision.
   - Precision values for Gini range from 10.60% to 19.29%, while for Information Gain it drops to 6.38% at the 80%-20% split.

Conclusion:

Both models demonstrate strong accuracy and specificity across all splits.
However, **Gini Index** is superior in terms of **sensitivity** and **precision**, which are critical for correctly identifying true stroke cases and reducing false positives.

- **Information Gain**:
  Offers slightly better specificity in some splits and a competitive error rate, but struggles significantly in sensitivity and precision, especially at the 80%-20% split.
- **Gini Index**:
  Provides a more balanced and reliable performance, particularly important in medical applications where identifying true positives is vital.

Final Analysis:
If the goal is to detect as many true positive stroke cases as possible, the **Gini Index** model is strongly preferred.
If minimizing false positives slightly more is critical, **Information Gain** could be considered, but it misses many real stroke cases.

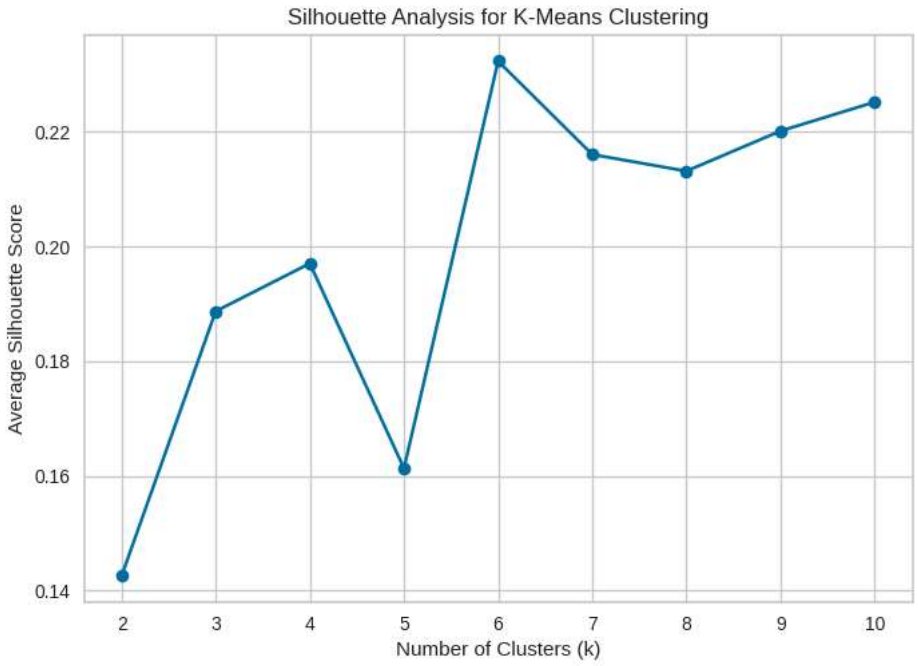
Thus, for stroke detection, **Gini Index** remains the better and safer choice.

part 2: Clustering

We selected three cluster sizes (3, 6, and 10) based on the outcomes of the validation methods we plan to apply. These sizes will then be used to execute the K-means clustering algorithm.

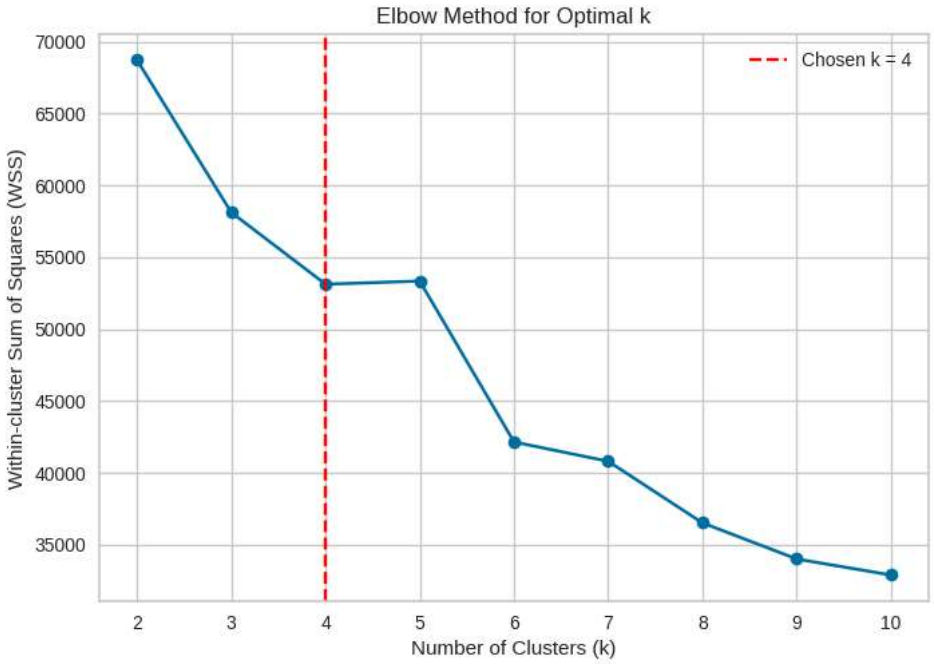
Silhouette method:

The Silhouette method evaluates clustering quality by measuring how well data points fit their cluster compared to others, with higher scores indicating better separation.



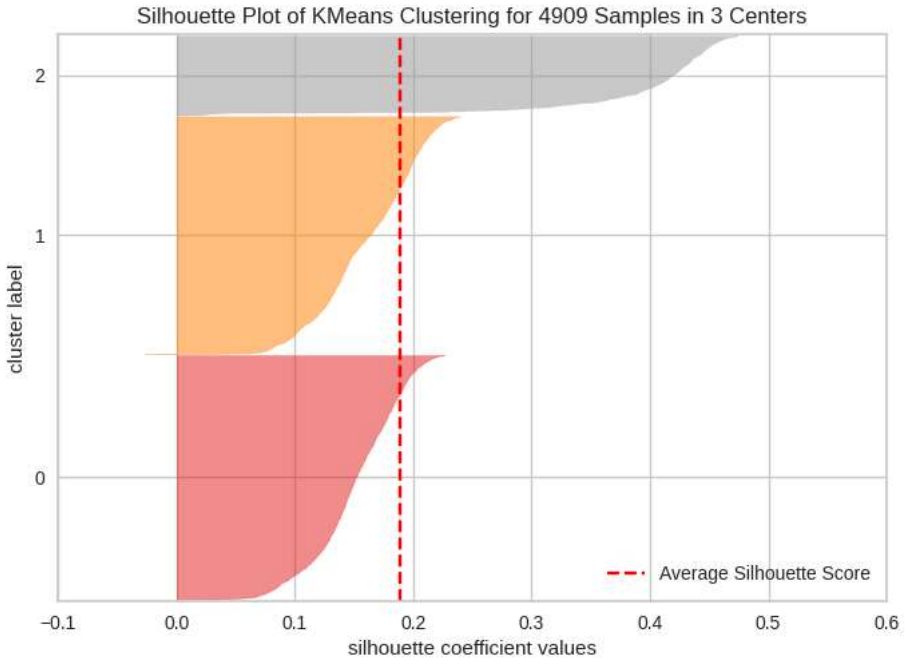
Elbow method:

The Elbow method identifies the optimal number of clusters for K-means by plotting the Within-Cluster Sum of Squares (WSS) and selecting the point where the decrease slows, forming an "elbow."

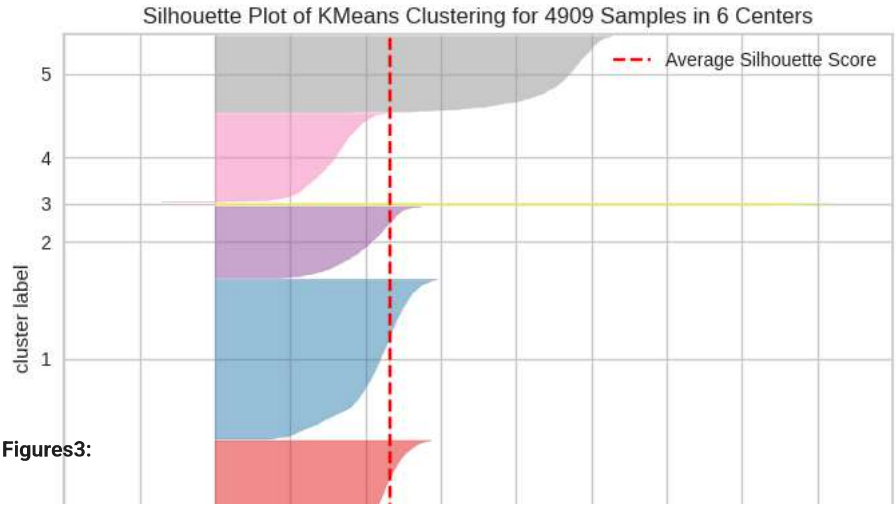


Figures:

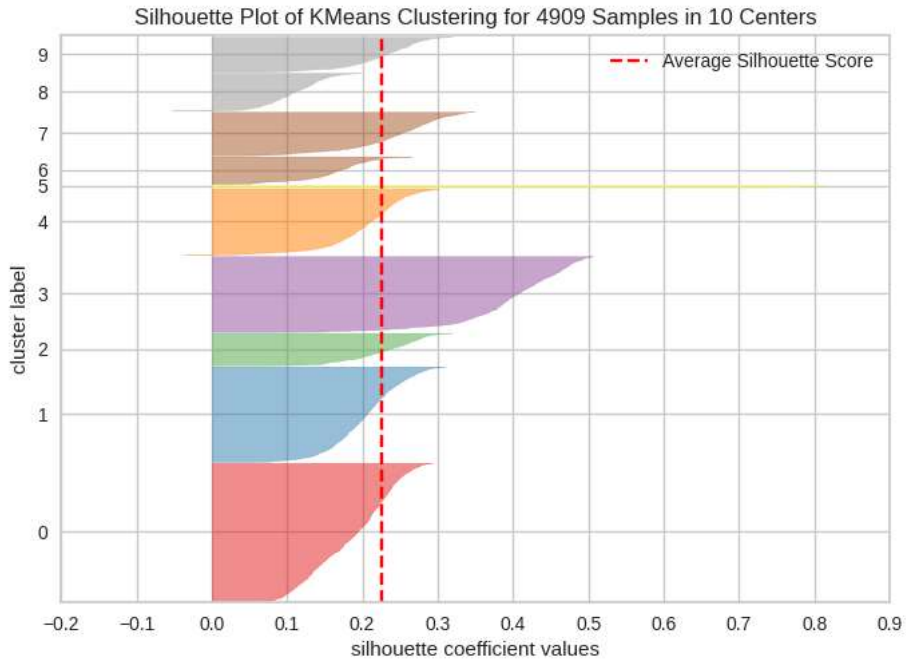
Figures1:



Figures2:



Figures3:



	K = 3	K = 6	K = 10
WSS	58121.34	42140.73	32880.61
Silhouette Score	0.188	0.233	0.225

Final decision:

Based on the evaluation metrics including WSS (Within-Cluster Sum of Squares) and the Average Silhouette Score, we have determined that K = 6 is the most suitable choice for our clustering model. This decision is supported by the fact that K = 6 achieved the highest Silhouette Score (0.233) among the tested values, indicating better-defined and well-separated clusters. Although K = 6 does not have the highest WSS value, the balance it provides between intra-cluster compactness and inter-cluster separation makes it the most optimal configuration for this dataset. This trade-off confirms that K = 6 offers superior clustering quality overall.

7-Findings