DATA ANALYSIS ON STROKES

Risk Factors, Trends and Prevention Strategies

TABLE OF CONTENTS

1 DISCLAIMER

The dataset used in this analysis, called "Prediction Stroke Dataset," is sourced from Kaggle, a popular platform for data science competitions and datasets. It is important to note that, although Kaggle provides a wide range of datasets, the reliability and accuracy of these datasets are not always guaranteed. The data in this dataset may not have been verified or certified by any official medical or governmental body.

This work was undertaken as an exercise to practice and demonstrate the methodologies and techniques I would use when working with real and reliable data. Despite the potential limitations of the dataset, I have treated the data as if it were real and reliable for the purposes of this exercise. This includes thorough data exploration, analysis, and interpretation to simulate a realistic data analysis scenario.

However, the results and conclusions drawn from this dataset should be interpreted with caution. This dataset is intended for educational and informational purposes only and should not be used as the sole basis for medical, clinical, or health-related decisions.

By acknowledging these limitations, I aim to ensure transparency in the use of this dataset and to emphasize the importance of using verified and reliable data for critical decision-making processes.

2 PROJECT OBJECTIVES

The project I have undertaken aims to deepen my skills as a Data Analyst, despite my current level and the challenges encountered. My determination to carry out this project is motivated by personal and professional reasons. My father suffered a stroke, which led me to question the potential causes of this event and to seek ways to prevent a recurrence and improve his well-being. These questions prompted me to choose the complex topic of strokes for this project.

My main objective is to understand the risk factors associated with strokes, determine how these factors influence the likelihood of having a stroke, and explore trends and prevention strategies. This project is structured into three distinct parts to address these fundamental questions.

2.1 First Part: Risk Factor Analysis

1. What are the most significant risk factors for strokes?
2. How do these risk factors affect the likelihood of having a stroke?
3. What are the prevalence patterns of strokes in different populations?

In this first part, the goal is to identify and analyze the most significant risk factors for strokes, understand their impact on stroke probability, and examine prevalence patterns in different populations. This analysis will help me better understand the potential causes of my father's stroke and take appropriate preventive measures.

2.2 Second Part: Prevention Interventions

4. What are the most effective prevention interventions?
5. What progress has been made in stroke prevention over time?

The second part of my project will focus on exploring the most effective prevention interventions against strokes. I will also examine the progress made in stroke prevention over time. The objective is to discover the most promising prevention strategies and understand how preventive practices have evolved, which could offer valuable insights for preventing recurrences.

2.3 Third Part: Economic Costs and Technological Advances

6. What are the economic costs associated with strokes and their prevention?
7. How do technological advances contribute to stroke prevention and management?

Finally, the third part of my project will focus on the economic costs associated with strokes and their prevention, as well as the role of technological advances in stroke prevention and management. This part aims to understand the economic implications of strokes and evaluate how modern technologies can improve prevention and care.

2.4 Conclusion

In summary, this project aims to deepen my understanding of strokes, their risk factors and prevention strategies, using the data that I have while developing my data analysis skills. Despite the challenges, my commitment to this project is reinforced by my desire to help my father and contribute to stroke prevention more broadly. In the near future, I would like to do the same with other pathologies and contribute to their prevention strategies.

3 COMPLETE AND DETAILED DESCRIPTION OF THE DATASET

3.1 General Overview

The "Prediction Stroke Dataset" comprises data aimed at predicting the likelihood of a stroke occurring in patients. The dataset contains 5110 rows, each representing an individual patient, and includes various demographic, health-related, and lifestyle characteristics.

3.1.1 Columns

The dataset includes the following columns:

1. **id**: A unique identifier for each patient.
2. **gender**: The gender of the patient, with possible values Male, Female, and occasionally Other.
3. **age**: The age of the patient in years, represented as a numerical value.
4. **hypertension**: Indicates whether the patient has hypertension (0 for no, 1 for yes).
5. **heart_disease**: Indicates whether the patient has heart disease (0 for no, 1 for yes).
6. **ever_married**: Indicates whether the patient has ever been married (No, Yes).
7. **work_type**: The type of work the patient does, such as children, Govt_job, Never_worked, Private, Self-employed.
8. **Residence_type**: The type of residence the patient lives in, either Urban or Rural.
9. **avg_glucose_level**: The average glucose level in the patient's blood.
10. **bmi**: The patient's body mass index (BMI).
11. **smoking_status**: The patient's smoking status, with values such as formerly smoked, never smoked, smokes, and occasionally Unknown.
12. **stroke**: Indicates whether the patient has had a stroke (0 for no, 1 for yes).

3.1.2 Statistical Descriptions

For columns with numerical values, descriptive statistics are generally provided:

- **age**:
    - **Mean**: Average age of the patients.
    - **Median**: Median age of the patients.
    - **Minimum and Maximum**: Age range.
    - **Quartiles**: Age distribution (25th percentile, 50th percentile, 75th percentile).
- **avg_glucose_level**:
    - **Mean**: Average glucose level.
    - **Median**: Median glucose level.

    - **Minimum and Maximum**: Glucose level range.
    - **Quartiles**: Glucose level distribution.
- **bmi**:
    - **Mean**: Average BMI.
    - **Median**: Median BMI.
    - **Minimum and Maximum**: BMI range.
    - **Quartiles**: BMI distribution

3.1.3 Missing Values

- The **BMI** column contains missing values, indicated as N/A or NaN

3.1.4 Additional Information

- **Gender Distribution**: Number and percentage of patients by gender.
- **Prevalence of Hypertension**: Percentage of patients with and without hypertension.
- **Prevalence of Heart Disease**: Percentage of patients with and without heart disease.
- **Marital Status**: Number of patients who have ever been married or not.
- **Work Type Distribution**: Number of patients by work type.
- **Residence Type Distribution**: Number of patients living in urban or rural areas.
- **Smoking Status**: Number of patients by smoking status.

3.1.5 Conclusion

This dataset provides a comprehensive view of various factors that may influence the likelihood of stroke occurrence. It is useful for predictive modeling and understanding public health trends related to strokes. However, it is crucial to ensure data quality and validate the results with additional sources before making clinical or public health decisions.

4 USE OF SCIENTIFIC LITERATURE FOR STROKE ANALYSIS : A STUDY BASED ON THE DOCUMENT "GLOBAL BURDEN OF DISEASE STUDY 2019"

As part of the stroke data analysis project, I used the document titled "Global, regional, and national burden of stroke and its risk factors, 1990–2019: a systematic analysis for the Global Burden of Disease Study 2019" published in The Lancet Neurology by the GBD 2019 Stroke Collaborators.

I used this document to conduct in-depth analyses and establish connections with the "Stroke Prevention Dataset" to confirm and reinforce conclusions on stroke risk factors. The comprehensive and updated data provided in the "Global Burden of Disease Study 2019" is crucial for contextualizing the results obtained and developing appropriate prevention strategies. By comparing the results from both sources, I identified the main risk factors, analyzed geographical and economic disparities, and established a solid foundation for effective prevention strategies.

Document Fact Sheet "Global Burden of Disease Study 2019" (GBD_2019.pdf)

Title: Global, regional, and national burden of stroke and its risk factors, 1990–2019: a systematic analysis for the Global Burden of Disease Study 2019

Authors: GBD 2019 Stroke Collaborators

Publication: The Lancet Neurology, Vol 20, October 2021

URL: The Lancet

Summary: This document presents a systematic analysis of the global, regional, and national burden of strokes and their risk factors, based on the Global Burden of Disease (GBD) 2019 study. The analysis covers the period from 1990 to 2019 and includes data on incidence, prevalence, mortality, disability-adjusted life years (DALYs), and population attributable fractions (PAFs) of DALYs associated with 19 risk factors in 204 countries and territories. The types of strokes considered include ischemic stroke, intracerebral hemorrhage, and subarachnoid hemorrhage, with data stratified by sex, age group, and country income level according to World Bank classification.

Key Points of the Document:

1. **Incidence and Prevalence**: In 2019, there were 12.2 million new stroke cases and 101 million prevalent cases worldwide.
2. **Mortality**: Strokes caused 6.55 million deaths in 2019, making it the second leading cause of death globally.
3. **DALYs**: In 2019, strokes contributed to 143 million DALYs, representing the third leading cause of death and disability combined.
4. **Risk Factors**: The main identified stroke risk factors are high systolic blood pressure, high BMI, high fasting plasma glucose, particulate matter pollution, and smoking.
5. **Trends**: Between 1990 and 2019, the absolute number of strokes, deaths, and DALYs increased, despite a decrease in age-standardized rates.

Methodology:

- Utilized GBD 2019 analytical tools to calculate stroke incidence, prevalence, mortality, and DALYs.
- Estimated PAFs for 19 risk factors using meta-analyses of epidemiological studies.
- Analyzed variations by sex, age, region, and country income level.

Summary

The document "GBD_2019.pdf" provides a comprehensive analysis of the global burden of strokes and their risk factors over nearly three decades. It reveals a substantial increase in stroke cases, deaths, and DALYs, highlighting significant regional and national variations. The main identified risk factors are modifiable, emphasizing the importance of prevention strategies.

Why this Document is a Good Choice for My Project?

1. **Comprehensive and Updated Data**: The document "GBD_2019.pdf" provides a global and detailed overview of strokes and their risk factors, covering multiple dimensions (incidence, prevalence, mortality, DALYs) over a long period (1990-2019). These data are crucial for contextualizing the results obtained in this project.
2. **Identification of Main Risk Factors**: The graphs and analyses in "GBD_2019" clearly identify the most significant risk factors for strokes, such as hypertension, high BMI, high glucose, and smoking. These factors are also analyzed in this project, allowing for confirmation and reinforcement of the conclusions drawn from both documents.
3. **Regional and National Comparison**: "GBD_2019" offers detailed comparisons between different regions and income levels, providing a better understanding of the geographical and economic disparities in stroke prevalence and impact. This information complements the more specific analyses of individual data presented in the project.
4. **Basis for Prevention Strategies**: The findings from "GBD_2019" on modifiable risk factors and epidemiological trends provide a solid foundation for developing effective prevention strategies. This information is essential for addressing the strategic questions posed in this project.

5 IMPORTED MODULES

To carry out this data analysis project on strokes, I used various Python modules, each with a specific role in the process of data manipulation, analysis, visualization, and modeling. Here is a detailed explanation of the imported modules and their utility:

5.1 Imported Modules and Their Utility

1. **pandas (import pandas as pd)**:
    - Used for data manipulation and analysis. It allows reading, writing, and managing datasets in the form of DataFrames, facilitating data cleaning and transformation operations.
2. **numpy (import numpy as np)**:
    - Used for mathematical operations and array manipulations. Numpy provides efficient data structures for numerical operations.
3. **matplotlib (import matplotlib.pyplot as plt)**:
    - Used for creating 2D static visualizations such as bar charts, histograms, line plots, etc.
4. **seaborn (import seaborn as sns)**:
    - Based on matplotlib, seaborn simplifies the creation of attractive and informative statistical visualizations.
5. **statsmodels (import statsmodels.api as sm)**:
    - Used for statistical estimation and inference. It offers tools for statistical modeling, statistical tests, and diagnostics.
6. **plotly (import plotly.express as px, import plotly.graph_objs as go)**:
    - Used to create interactive and dynamic visualizations. Plotly Express (px) simplifies the creation of quick graphs, while Graph Objects (go) offers more detailed and customizable controls.
7. **scipy (from scipy.stats import chi2_contingency, from scipy import stats)**:
    - Used for scientific and technical calculations. The chi2_contingency and stats functions are used for statistical tests, such as chi-square tests and t-tests.
8. **scikit-learn (from sklearn.*)**:
    - Used for machine learning tasks. The various classes and functions of scikit-learn cover data preprocessing, model construction, performance evaluation, and pipeline management. Here are the sub-modules used:
        - **Pipeline and ColumnTransformer**: Facilitates the construction and management of transformation and modeling pipelines.

- **train_test_split**: Splits the dataset into training and testing sets.
- **GridSearchCV**: Optimizes model hyperparameters through grid search.
- **LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, GaussianNB**: Classification algorithms.
- **SimpleImputer**: Handles missing values.
- **SMOTE**: Oversampling technique to balance classes.
- **VotingClassifier, BaggingClassifier, StackingClassifier**: Ensemble learning techniques.
- **StandardScaler, OneHotEncoder**: Data normalization and encoding.
- **LinearRegression, RandomForestRegressor, LassoLars, RidgeCV**: Regression algorithms.
- **accuracy_score, classification_report, roc_auc_score, roc_curve, confusion_matrix**: Model evaluation metrics.
- **LabelEncoder**: Encodes categorical labels into numerical values.

9. **imblearn (from imblearn.over_sampling import SMOTE)**:
    - Used to handle imbalanced datasets by oversampling minority classes.
10. **warnings (import warnings)**:
    - Used to manage warnings and filter out unwanted warning messages.

5.2 Utilization in the Project

- **Data Manipulation and Preparation**: pandas, numpy, SimpleImputer, StandardScaler, OneHotEncoder, LabelEncoder
- **Visualization**: matplotlib, seaborn, plotly
- **Statistical Analysis**: scipy, statsmodels
- **Modeling and Machine Learning**: scikit-learn (with various algorithms and pipeline tools), imblearn
- **Model Evaluation**: scikit-learn (with various evaluation metrics)

These modules enable a comprehensive and detailed data analysis process, covering all necessary steps to obtain relevant insights and reliable predictive models. They are essential for successfully conducting a data analysis project, from data preparation to modeling, including visualization and results evaluation.

6 IMPORT DATASET

6.1 # Import the Stroke Prediction Dataset
stroke_data = pd.read_csv(r'C: \Users\user\Downloads\healthcare-dataset-stroke-data.csv')

6.2 # Display the first few rows of the stroke_data DataFrame
stroke_data.head()

	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

7 DATA EXPLORATION

7.1 Column Descriptions

 # Get the shape of the DataFrame
stroke_data.shape = 5110 lignes et 12 colonnes

 # Display concise summary information about the stroke_data DataFrame
stroke_data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

7.2 Statistiques Descriptives

 # Generate descriptive statistics of the stroke_data DataFrame
stroke_data.describe()

	id	age	hypertension	heart_disease	\
count	5110.000000	5110.000000	5110.000000	5110.000000	
mean	36517.829354	43.226614	0.097456	0.054012	
std	21161.721625	22.612647	0.296607	0.226063	
min	67.000000	0.080000	0.000000	0.000000	
25%	17741.250000	25.000000	0.000000	0.000000	
50%	36932.000000	45.000000	0.000000	0.000000	
75%	54682.000000	61.000000	0.000000	0.000000	
max	72940.000000	82.000000	1.000000	1.000000	

	avg_glucose_level	bmi	stroke
count	5110.000000	4909.000000	5110.000000
mean	106.147677	28.893237	0.048728
std	45.283560	7.854067	0.215320
min	55.120000	10.300000	0.000000
25%	77.245000	23.500000	0.000000
50%	91.885000	28.100000	0.000000
75%	114.090000	33.100000	0.000000
max	271.740000	97.600000	1.000000

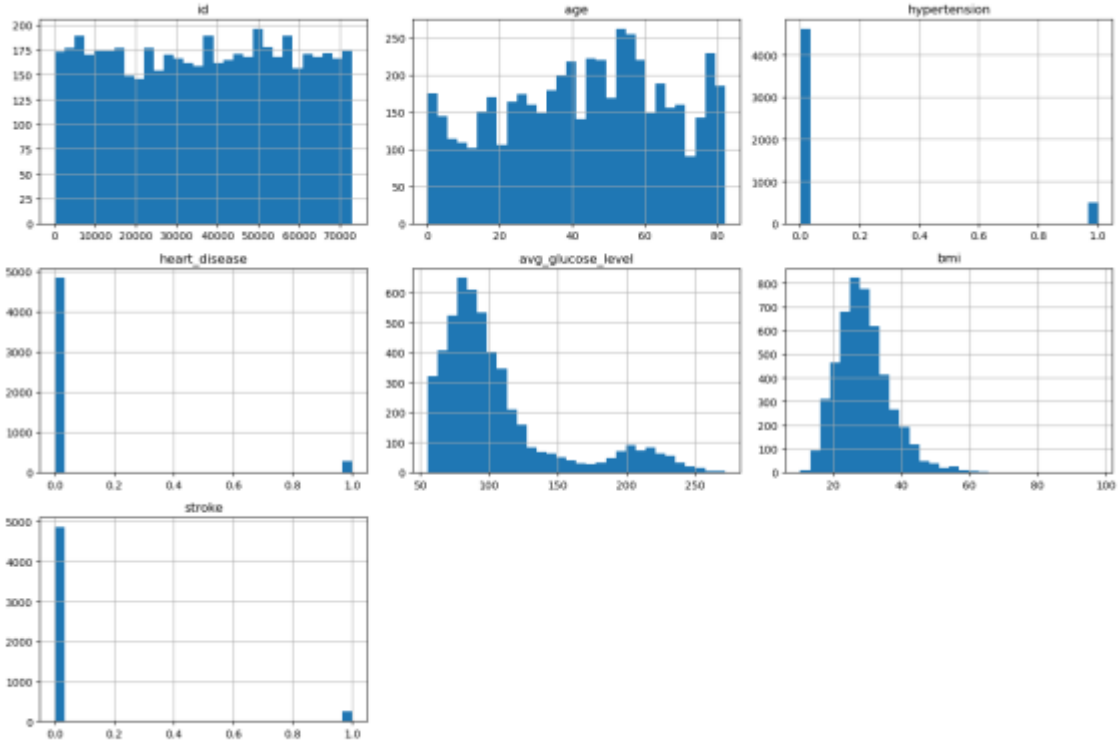
The command stroke_data.describe() generates descriptive statistics for the DataFrame, providing an overview of the data's distribution. This includes metrics such as count, mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile), 75th percentile (Q3), and maximum for each numerical column. The results help to understand the central tendency, dispersion, and shape of the dataset's distribution, allowing for initial insights into the data's characteristics and potential anomalies. For example, in this dataset, the average age is approximately 43.2 years, and the average BMI is around 28.9.

8 VISUALIZATION OF CONTINUOUS VARIABLES

 # Continuous variables

```
stroke_data.hist(bins=30, figsize=(15,10))
plt.tight_layout()
plt.show()
```

 # Visualization and distribution of the different variables in the stroke_data ␣ dataset as histograms

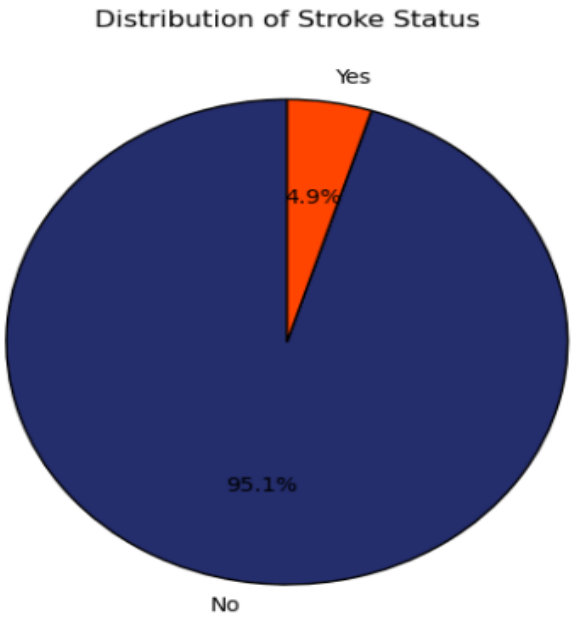


9 RISK FACTORS ANALYSIS

9.1 Exploratory Data Analysis (EDA)

9.1.1 Stroke Status

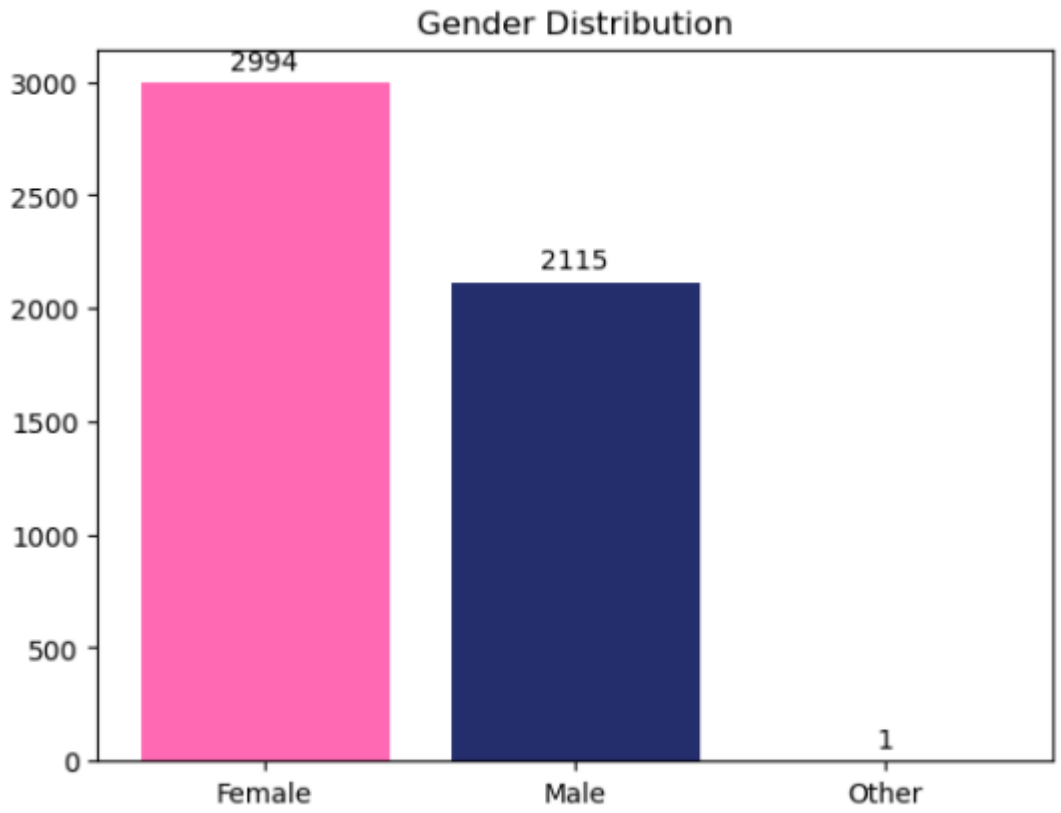
```
# Replace values in the 'stroke' column
stroke_data2['stroke'] = stroke_data2['stroke'].replace({1: 'Yes', 0: 'No'})

# Count the occurrences of different values in the 'stroke' column
stroke_counts = stroke_data2['stroke'].value_counts().reset_index()
stroke_counts.columns = ['Stroke', 'Count']

# Define custom colors
custom_colors = ['#252E6C', '#FF4500']

# Create the pie chart
fig, ax = plt.subplots()
pie = ax.pie(stroke_counts['Count'], labels=stroke_counts['Stroke'],
colors=custom_colors, autopct='%1.1f%%', startangle=90,
wedgeprops={'edgecolor': 'black'})

# Add a title
plt.title('Distribution of Stroke Status')

# Add a legend with specified labels and modified position
plt.legend(pie[0], ['No Stroke', 'Stroke'], loc="center left",
bbox_to_anchor=(1, 0.5))

# Display the chart
plt.tight_layout()
plt.show()
```



The pie chart visualizes the distribution of values in the 'stroke' column. The legend indicates 'Yes' for stroke cases and 'No' for non-cases. The percentages show the proportion of each category. In this case, there are 4.9% stroke cases ('Yes') and 95.1% non-cases ('No')

9.1.2 Distribution by Gender

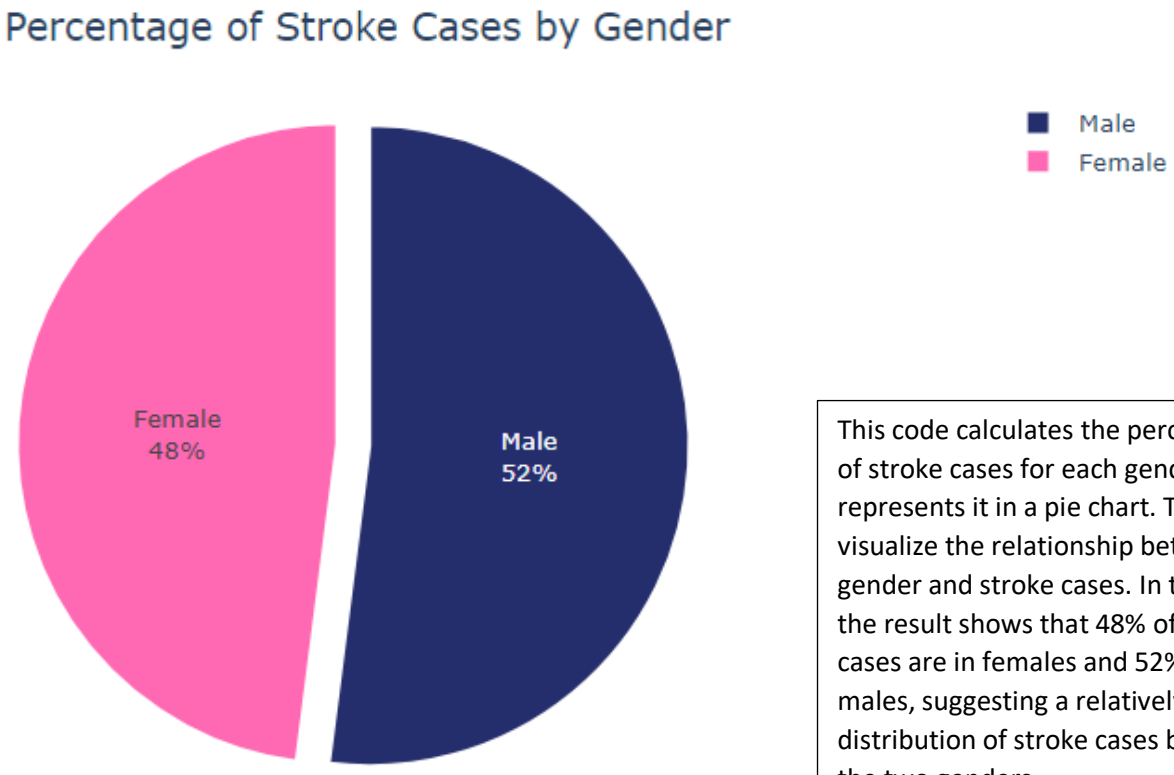
```
# Count the occurrences of different values in the 'gender' column
gender_counts = stroke_data['gender'].value_counts()

# Define custom colors
custom_colors = ['#FF69B4', '#252E6C', '#90EE90']

# Create the bar chart
fig, ax = plt.subplots()
bars = ax.bar(gender_counts.index, gender_counts.values, color=custom_colors)

# Add labels for each bar
for bar in bars: height = bar.get_height()
  ax.annotate('{}'.format(height),
xy=(bar.get_x() + bar.get_width() / 2, height),
xytext=(0, 3), # 3 points above the bar
textcoords="offset points",
ha='center', va='bottom')


# Add a title
plt.title('Gender Distribution')

# Display the chart
plt.show()
```



The bar chart visualize the distribution of values in the ‘gender’ column. Each bar represents a gender category (‘Female’, ‘Male’, ‘Other’) with corresponding counts. The annotations on top of each bar display the exact count for that category. In this case, there are 2994 Females, 2115 Males, and 1 Other.

9.1.3 Stroke Cases by Gender

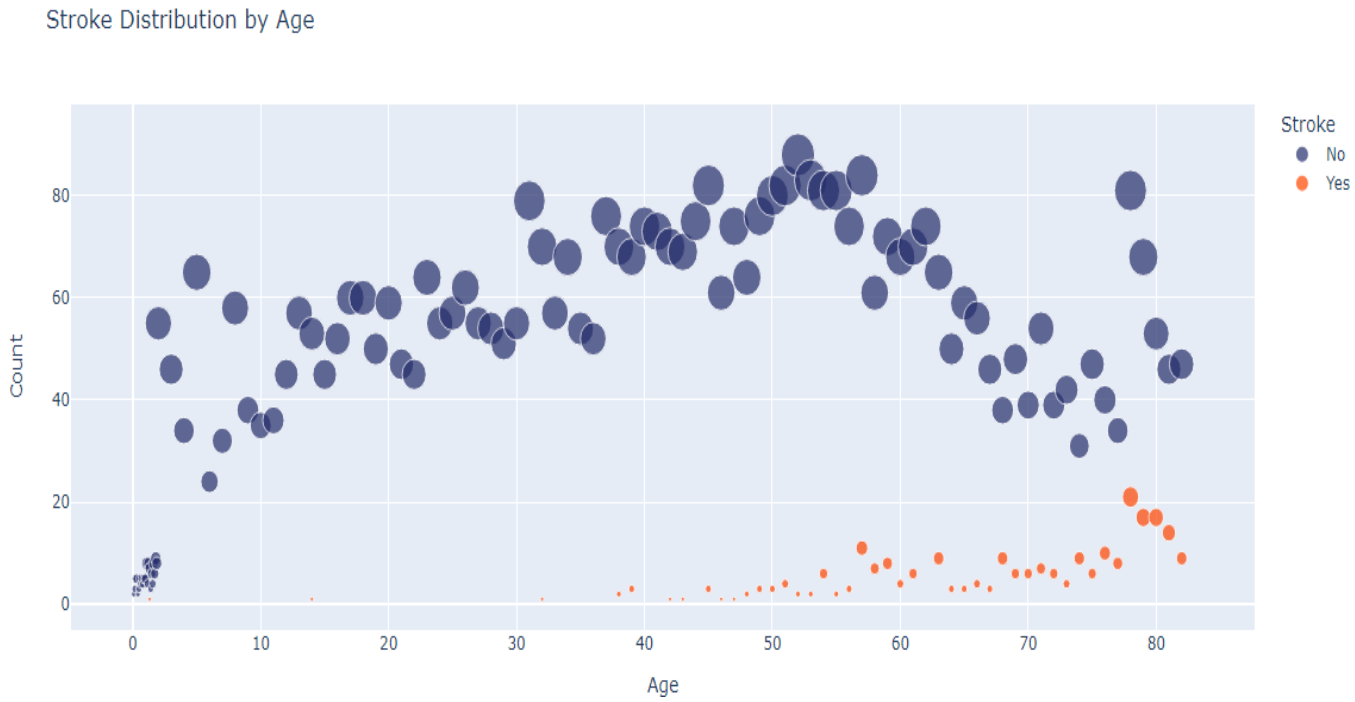
```
# Count the occurrences of different values in the 'gender' and 'stroke' columns, then group them
gender_stroke_counts = stroke_data.groupby(['gender', 'stroke']).size().unstack()

# Calculate the percentage of stroke cases for each gender
gender_stroke_percentage = (gender_stroke_counts[1] / (gender_stroke_counts[0] +
gender_stroke_counts[1])) * 100

# Create a pie chart with Plotly Express
fig = px.pie(names=gender_stroke_percentage.index,
values=gender_stroke_percentage.values,  title="Percentage of Stroke Cases by Gender",
color_discrete_sequence=['#252E6C', '#FF69B4'])

# Update chart traces
fig.update_traces(textinfo="percent+label", pull=[0.1, 0],
marker=dict(line=dict(color="white", width=2)))

# Display the chart
fig.show()
```



This code calculates the percentage of stroke cases for each gender and represents it in a pie chart. This helps visualize the relationship between gender and stroke cases. In this case, the result shows that 48% of stroke cases are in females and 52% in males, suggesting a relatively equal distribution of stroke cases between the two genders.

9.1.4 Stroke Distribution by Age

```
# Group the data by age and stroke status, then count the occurrences of each combination
age_stroke_counts = stroke_data2.groupby(["age",
"stroke"]).size().reset_index(name="Count")

# Create a scatter plot with Plotly Express
fig_bubble_age_stroke = px.scatter(age_stroke_counts, x="age", y="Count", size="Count",
color="stroke",
                    title="Stroke Distribution by Age",
                    labels={"age": "Age", "Count": "Count", "stroke": "Stroke"},
                    color_discrete_sequence=['#252E6C', '#FF4500'])

# Update the layout of the plot
fig_bubble_age_stroke.update_layout(xaxis_title="Age", yaxis_title="Count")


# Display the plot
fig_bubble_age_stroke.show()
```



The scatter plot shows the count of stroke cases for different age groups. There is a notable concentration of 'Yes' (stroke) cases among individuals aged between 60 and 80 years, suggesting that this age range may have a higher risk of experiencing a stroke This code groups the data by age and stroke status, then counts the occurrences of each combination. The resulting scatter plot visualizes the distribution of stroke cases across different age groups. Understanding how stroke incidence varies with age is crucial for identifying age-related risk factors and informing prevention strategies.

9.2 Statistical Tests

9.2.1 Student's t-test for continuous variables (age, glucose, BMI)

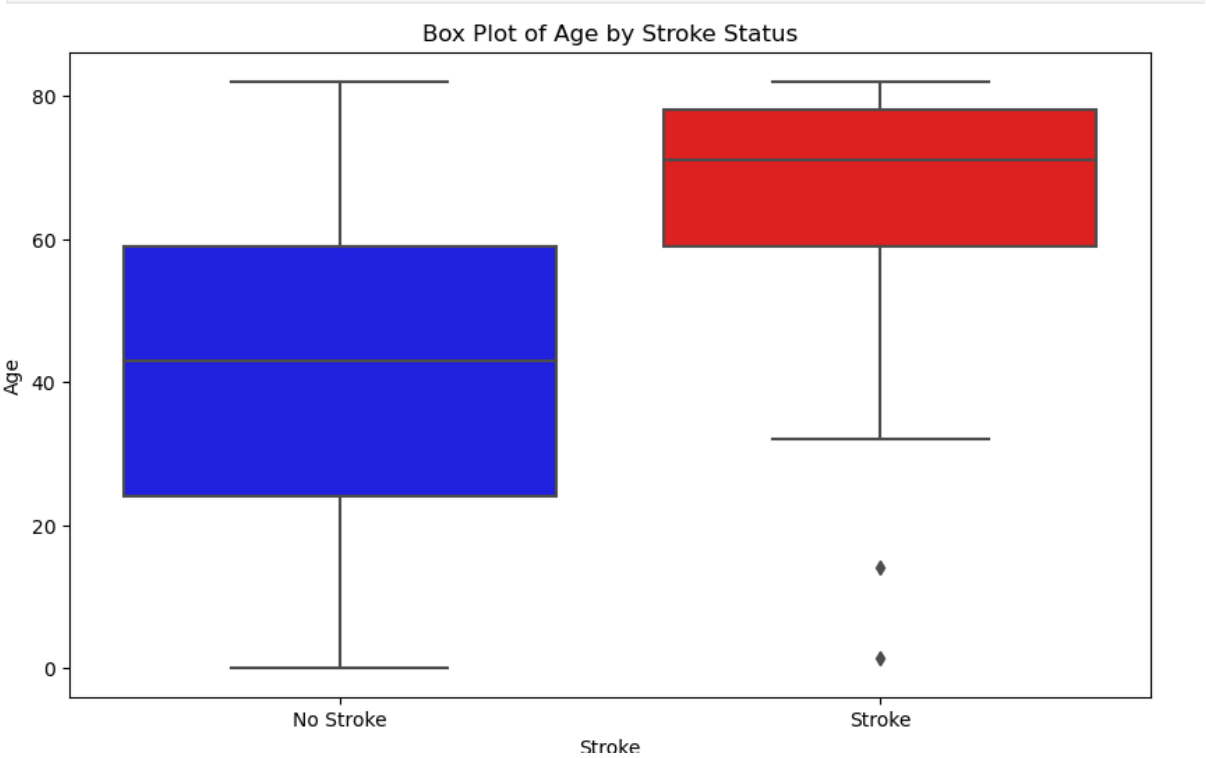
```
# Calculate the t-statistic and p-value for age
t_stat_age, p_value_age = stats.ttest_ind(stroke_data[stroke_data['stroke'] == 1]['age'],
                                          stroke_data[stroke_data['stroke'] == 0]['age'])

# Display the t-test results for age
print(f'T-test for age - T-statistic: {t_stat_age}, P-value: {p_value_age}')
```

T-test for age - T-statistic: 18.08083426887953, P-value: 7.0307775129939774e-71

This code conducts a Student's t-test to compare the ages of individuals who had a stroke (1) and those who did not (0). A very low p-value (7.03e-71) indicates a statistically significant difference between the two groups, suggesting that age is an important factor in stroke prediction in this project. A high t-statistic like 18.08 further supports this notion, indicating a significant difference in age averages between the two groups.

9.2.1.1 Create the box plot

```
# Set the figure size
plt.figure(figsize=(10, 6))

# Plot the box plot
sns.boxplot(x='stroke', y='age', data=stroke_data, palette=['blue', 'red'])

# Add labels for x and y axes, and a title
plt.xlabel('Stroke')
plt.ylabel('Age')
plt.title('Box Plot of Age by Stroke Status')

# Rename x-axis ticks
plt.xticks([0, 1], ['No Stroke', 'Stroke'])

# Show the box plot
plt.show()
```



The creation of this boxplot allows for visualizing the age distribution based on the stroke status. By interpreting the boxplot, we observe that the median age for individuals without stroke (No Stroke) falls between 25 and 60 years, whereas for those with stroke (Stroke), it falls between 60 and 80 years. The line in the middle of each box represents the median (the central value) of the dataset for each group.

9.2.2 Student's t-test for Glucose Levels

```
# Perform a Student's t-test to compare the average glucose levels between individuals who had a stroke and those who did not

# Calculate the t-statistic and the p-value
t_stat_glucose, p_value_glucose = stats.ttest_ind(
    stroke_data[stroke_data['stroke'] == 1]['avg_glucose_level'],  # Average glucose levels for individuals who had a stroke
    stroke_data[stroke_data['stroke'] == 0]['avg_glucose_level']   # Average glucose levels for individuals who did not have a stroke
)

# Display the results of the t-test
print(f'Test t for average glucose level - t-statistic: {t_stat_glucose}, P-value: {p_value_glucose}')
```

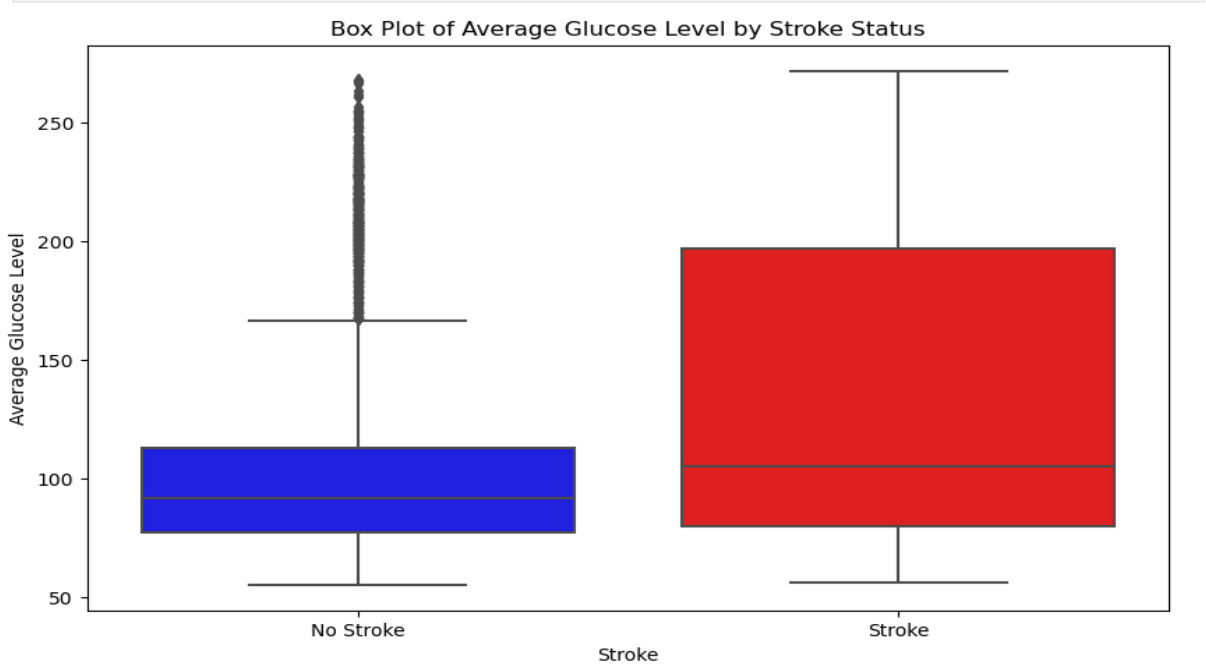
Test t for average glucose level - t-statistic: 9.513352175431471, P-value: 2.7678105194741054e-21

This t-test shows a highly significant difference in average glucose levels between individuals who had a stroke and those who did not (t-statistic: 9.51, p-value: 2.77e-21). This indicates that elevated glucose levels are strongly associated with stroke risk, highlighting the importance of glucose management in stroke prevention strategies.

9.2.2.1 Create the box plot

```
# Create a figure with a specific size
plt.figure(figsize=(10, 6))

# Create a box plot for the average glucose level, based on the presence or absence of a stroke
sns.boxplot(
    x='stroke',  # Categorical variable indicating the presence or absence of a stroke
    y='avg_glucose_level',  # Continuous variable indicating the average glucose level
    data=stroke_data,  # DataFrame containing the data
    palette=['blue', 'red']  # Colors for the categories "No Stroke" and "Stroke"
)

# Add a label to the x-axis
plt.xlabel('Stroke')

# Add a label to the y-axis
plt.ylabel('Average Glucose Level')

# Add a title to the plot
plt.title('Box Plot of Average Glucose Level by Stroke Status')

# Modify the tick labels on the x-axis to indicate "No Stroke" and "Stroke"
```

```
plt.xticks([0, 1], ['No Stroke', 'Stroke'])

# Display the plot
plt.show()
```



This box plot shows that stroke patients have higher average glucose levels (75-200) compared to those without a stroke (75-120). This indicates a significant difference, suggesting that elevated glucose levels are a risk factor for strokes. Effective glucose management is crucial for stroke prevention.

9.2.3 Student's t-test for BMI

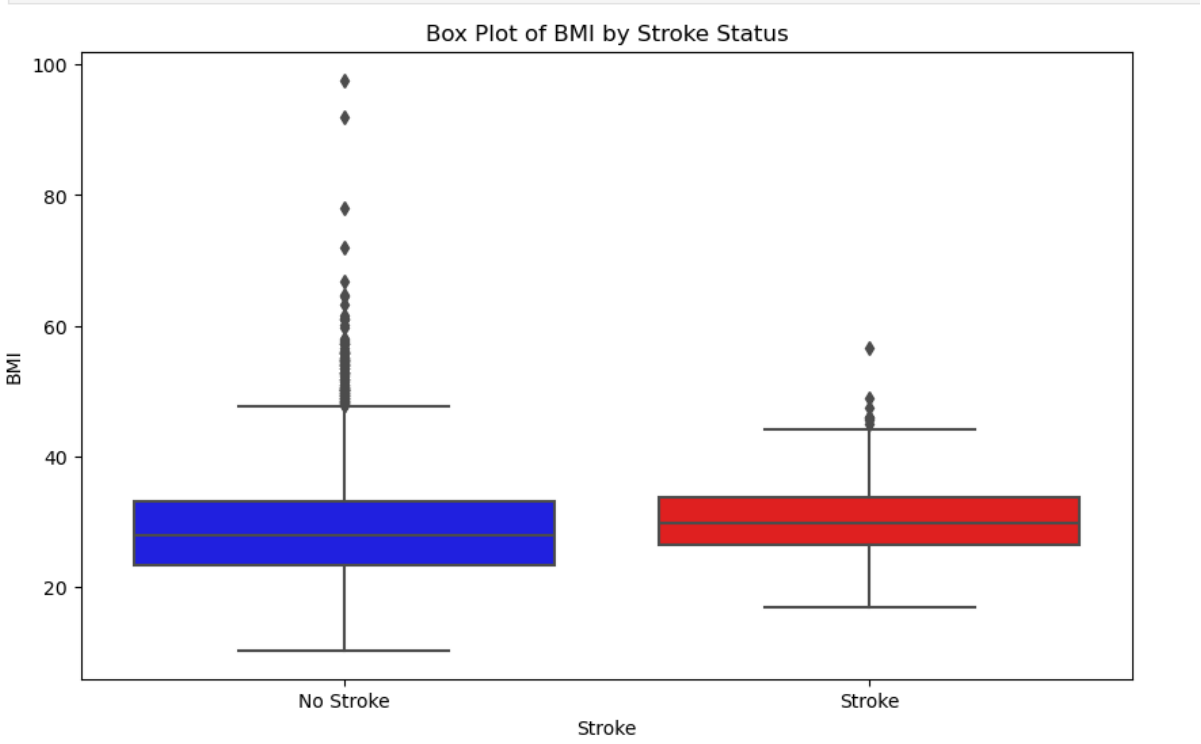
```
# Perform a t-test to compare BMI levels between groups with and without a stroke
t_stat_bmi, p_value_bmi = stats.ttest_ind(
    stroke_data[stroke_data['stroke'] == 1]['bmi'],  # BMI levels for individuals who had a stroke
    stroke_data[stroke_data['stroke'] == 0]['bmi']   # BMI levels for individuals who did not have a stroke
)

# Display the results of the t-test
print(f'Test t for bmi - t-statistic: {t_stat_bmi}, P-value: {p_value_bmi}')
```

```
Test t for bmi - t-statistic: nan, P-value: nan
```

9.2.3.1 Create the box plot for BMI by stroke status

```
# Set the size of the figure
plt.figure(figsize=(10, 6))

# Create the box plot with the specified colors
```

```
sns.boxplot(x='stroke', y='bmi', data=stroke_data, palette=['blue', 'red'])

plt.xlabel('Stroke') # Label the x-axis
plt.ylabel('BMI') # Label the y-axis
plt.title('Box Plot of BMI by Stroke Status') # Add a title to the plot
plt.xticks([0, 1], ['No Stroke', 'Stroke']) # Set the labels for the x-values

# Display the plot
plt.show()
```



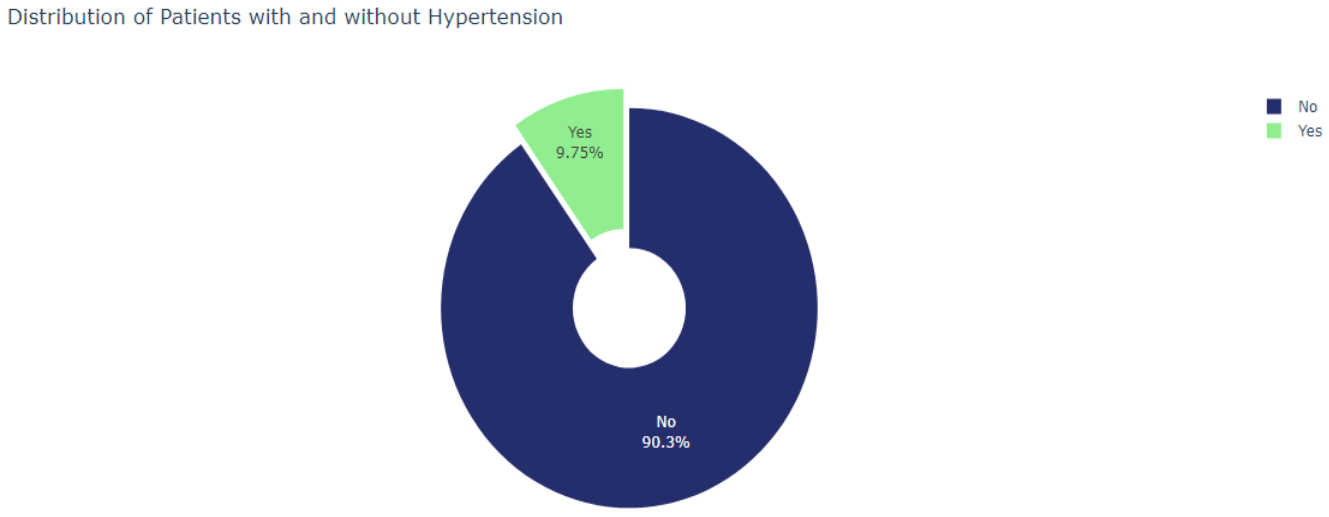
This box plot shows that stroke patients generally have higher BMIs (30-35) compared to those without strokes (25-35). This suggests that higher BMI is a significant risk factor for strokes, highlighting the importance of maintain ing a healthy BMI for stroke prevention.

10 SPECIFIC RISK FACTORS

10.1 Hypertension

10.1.1 Distribution of Hypertension

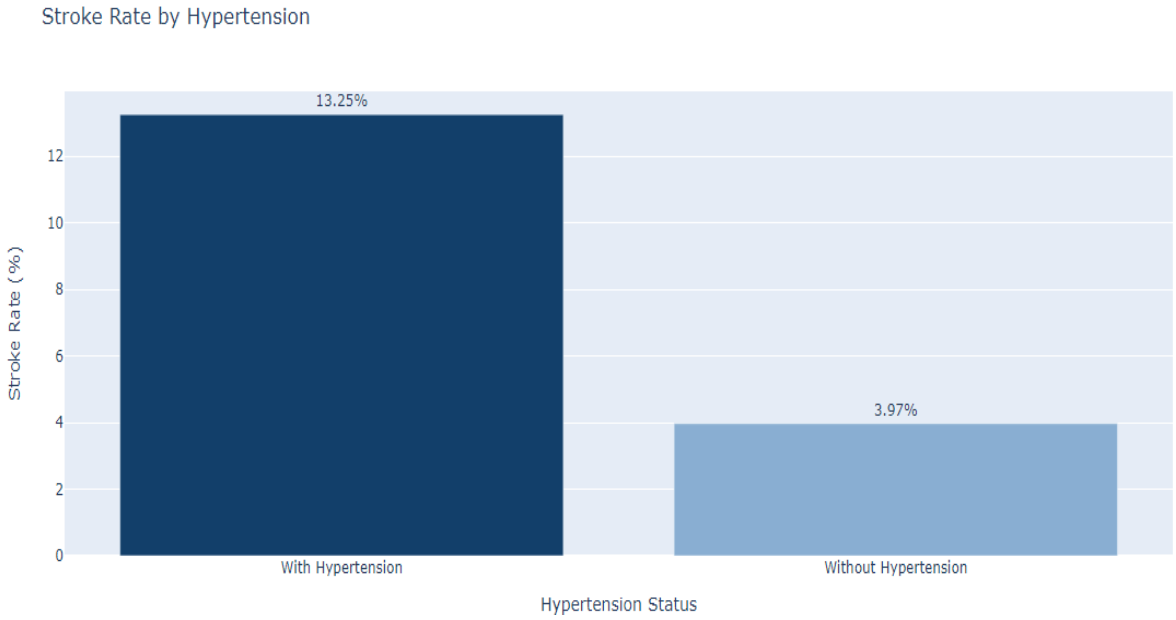
```
# Replace values in the 'hypertension' column
stroke_data2['hypertension'] = stroke_data2['hypertension'].replace({1: 'Yes', 0: 'No'})

# Count the occurrences of different values in the 'hypertension' column
hypertension_count = stroke_data2['hypertension'].value_counts()

# Define custom colors
custom_colors = ['#252E6C', '#90EE90']

# Create a pie chart with Plotly Express
fig = px.pie(
    values=hypertension_count, # Values to be represented
    names=hypertension_count.index, # Names of the categories
    hole=0.3, # Size of the hole in the center of the chart (0 for no hole)
    title='Distribution of Patients with and without Hypertension', # Chart title
    color_discrete_sequence=custom_colors, # Custom color sequence
)

# Update chart traces to include text information about percentages and labels,
and pull one of the sectors to highlight it
fig.update_traces(textinfo='percent+label', pull=[0, 0.1])

# Display the chart
fig.show()
```



This diagram allows for visualizing the distribution of patients with or without hypertension using a pie chart. In the context of the project, it helps to understand the prevalence of hypertension among the dataset patients. Interpreting the chart, we observe that 9.75% of patients have hypertension ("Yes"), while 90.3% do not ("No"). This suggests that hypertension is relatively uncommon in the dataset compared to patients without hypertension.

10.1.2 Stroke Rate by Hypertension

```
# Calculate the stroke rate with and without hypertension
stroke_rate_with_hypertension = (stroke_data[stroke_data['hypertension'] == 
1]['stroke'].mean()) * 100
stroke_rate_without_hypertension = (stroke_data[stroke_data['hypertension'] == 
0]['stroke'].mean()) * 100

# Create a DataFrame containing the data
data = pd.DataFrame({'Hypertension': ['With Hypertension', 'Without Hypertension'],
                     'Stroke Rate': [stroke_rate_with_hypertension, 
stroke_rate_without_hypertension]})

# Create a bar chart with Plotly Express
fig = px.bar(data, x='Hypertension', y='Stroke Rate',
             text='Stroke Rate', title='Stroke Rate by Hypertension',
             labels={'Hypertension': 'Hypertension Status', 'Stroke Rate': 'Stroke Rate (%)'})

# Update the colors of the bars
fig.update_traces(marker_color=["#123F6A", "#89AED2"])

# Update the position and format of the label text
fig.update_traces(texttemplate='%{text:.2f}%', textposition='outside')

# Display the chart
fig.show()
```



This diagram allows for comparing the stroke rate between patients with and without hypertension, a known risk factor for strokes. Interpreting the results, we observe a higher stroke rate among patients with hypertension (13.25%) compared to those without hypertension (3.97%). This suggests that hypertension is associated with an increased risk of stroke in this dataset, underscoring the importance of hypertension management in stroke prevention.

10.1.3 Chi-square Test for Hypertension

```
# Create a contingency table for hypertension and stroke status
contingency_table = pd.crosstab(stroke_data['hypertension'], stroke_data['stroke'])

# Perform the chi-square test on the contingency table
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display the results of the chi-square test
print('Chi-square test for hypertension - Chi2:', chi2, ', P-value:', p)
```

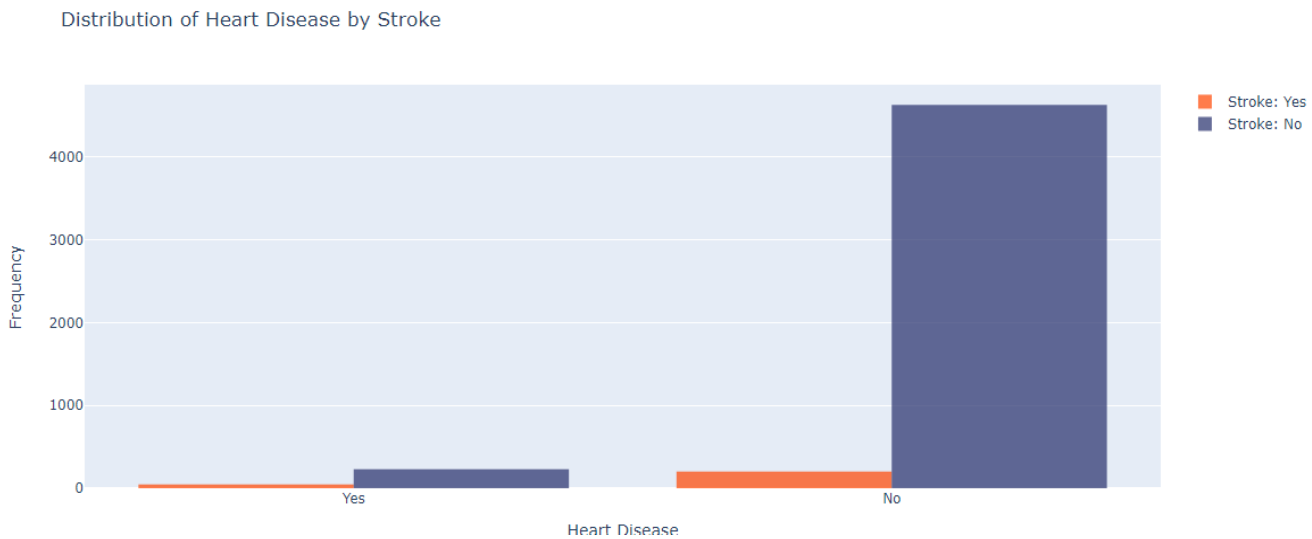
Chi-square test for hypertension - Chi2: 81.6053682482931 , P-value: 1.661621901511823e-19

This chi-square test on the contingency table for hypertension and stroke status evaluates whether there's a significant association between these two variables in the dataset. Interpreting the results, we observe a chi-square value of 81.61 and a very low p-value (1.66e-19), indicating a significant association between hypertension and stroke status. This suggests that hypertension is an important risk factor for strokes in this dataset, reinforcing the need to monitor and manage hypertension as a stroke prevention strategy.

10.2 Heart Disease

10.2.1 Distribution of Heart Disease by Stroke Status

```
# Replace values in the 'heart_disease' column
stroke_data2['heart_disease'] = stroke_data2['heart_disease'].replace({1: 'Yes', 0: 'No'})

# Separate data for patients who had a stroke and those who did not
heart_disease_yes = stroke_data2[stroke_data2["stroke"] == "Yes"]["heart_disease"]
heart_disease_no = stroke_data2[stroke_data2["stroke"] == "No"]["heart_disease"]

# Prepare data for the histogram
hist_data_heart_disease = [heart_disease_yes, heart_disease_no]
group_labels_heart_disease = ["Stroke: Yes", "Stroke: No"]
colors_heart_disease = ['#FF4500', '#252E6C']

# Create a Figure object (fig_heart_disease) with Plotly
fig_heart_disease = go.Figure()

# Add histograms for both groups (with and without stroke)
for i in range(2):
    fig_heart_disease.add_trace(go.Histogram(x=hist_data_heart_disease[i], nbinsx=10, opacity=0.7,
                                             name=group_labels_heart_disease[i],marker_color=colors_heart_disease[i]))

# Update the layout of the chart (title, axis titles)
fig_heart_disease.update_layout(title="Distribution of Heart Disease by Stroke",
                                xaxis_title="Heart Disease", yaxis_title="Frequency")

# Display the chart
fig_heart_disease.show()
```



This diagram is useful for analyzing the distribution of heart disease according to stroke status, enabling us to assess the association between heart disease and stroke. Interpreting the results, we observe that among patients who have had a stroke (stroke yes), 47 have heart disease (heart

disease yes) and 202 do not (heart disease no). Among stroke no patients, 229 had heart disease and 4632 did not. These results indicate that, although the majority of stroke-free patients do not have heart disease (4632), a significant proportion of stroke patients also have heart disease (47). This suggests that heart disease may be a risk factor for stroke, reinforcing the importance of monitoring and managing heart health as a stroke prevention strategy.

10.2.2 Chi-square Test for Heart Disease

```
# Create a contingency table for heart diseases and stroke status
contingency_table = pd.crosstab(stroke_data['heart_disease'], stroke_data['stroke'])

# Perform the chi-square test on the contingency table
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display the results of the chi-square test
print('Chi-square test for heart diseases - Chi2:', chi2, ', P-value:', p)
```

Chi-square test for heart diseases - Chi2: 90.25956125843324 , P-value: 2.0887845685229236e-21

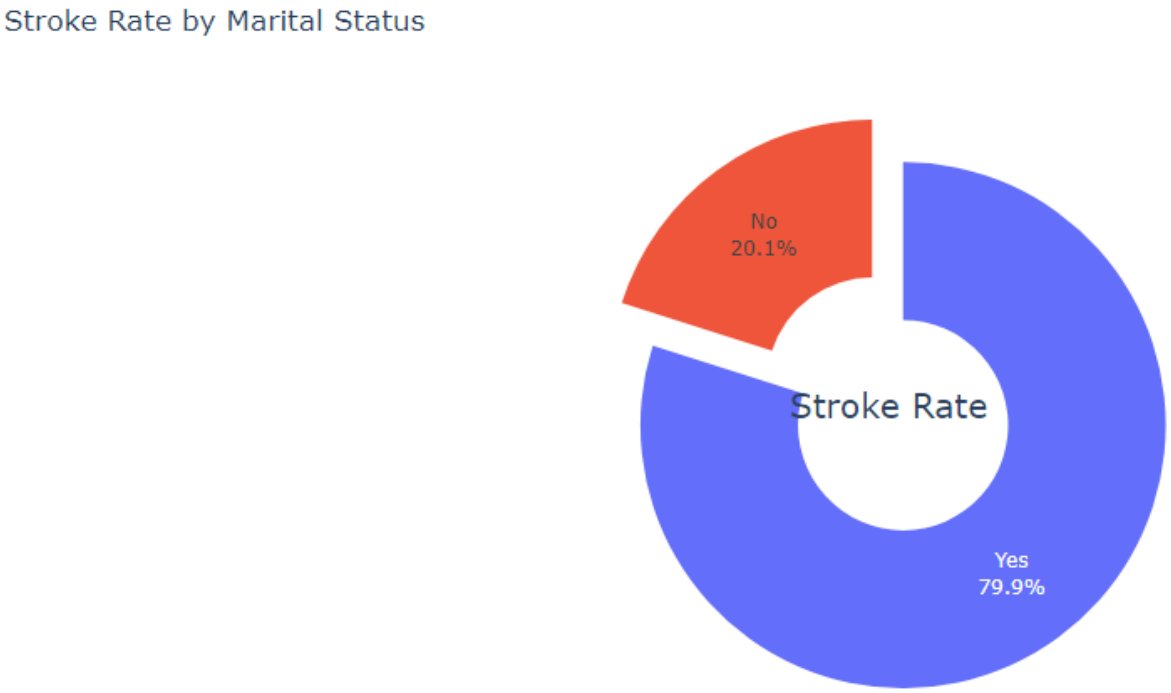
This chi-square test evaluates the association between heart disease and stroke status. The high chi-square value (90.26) and extremely low p-value (2.09e-21) indicate a significant association between heart disease and stroke. This suggests that heart disease is an important risk factor for strokes, emphasizing the need for heart disease management in stroke prevention strategies.

10.3 Marital Status

10.3.1 Stroke Rate by Marital Status

```
# Group the data by marital status and count the occurrences of each combination with or without stroke
attrition_by_ever_married = stroke_data2.groupby("ever_married")["stroke"].value_counts(normalize=True).unstack().reset_index()

# Define custom colors
custom_colors = ['#252E6C', '#010408']

# Create a pie chart with Plotly Express
fig_pie_ever_married = px.pie(attrition_by_ever_married, values="Yes",
names="ever_married", hole=0.4,
                    title="Stroke Rate by Marital Status",
                    labels={"ever_married": "Ever Married"},
                    color_discrete_map={'Yes': custom_colors[0], 'No': custom_colors[1]})
```

```
# Update chart traces to include text information about percentages and labels,
 and pull one of the sectors to highlight it
fig_pie_ever_married.update_traces(textinfo="percent+label", pull=[0.1, 0.1],
showlegend=False)

# Update the layout of the chart to include an annotation indicating the title of the chart
fig_pie_ever_married.update_layout(annotations=[dict(text="Stroke Rate", x=0.5, y=0.5,
font_size=20, showarrow=False)])

# Display the chart
fig_pie_ever_married.show()
```



This pie chart visualizes the stroke rate based on marital status, helping to analyze if being married impacts the likelihood of having a stroke. Interpreting the results, we see that 79.9% of stroke cases are among those who have been married ("Yes"), while 20.1% are among those who have never been married ("No"). This suggests a potential correlation between marital status and stroke risk, which could be significant for developing targeted prevention strategies.

10.3.2 Chi-square Test for Marital Status

```
# Create a contingency table for marital status and stroke status
contingency_table = pd.crosstab(stroke_data['ever_married'], stroke_data['stroke'])

# Perform the chi-square test on the contingency table
chi2, p, dof, expected = chi2_contingency(contingency_table)
```

```
# Display the results of the chi-square test
print('Chi-square test for marital status - Chi2:', chi2, ', P-value:', p)
```

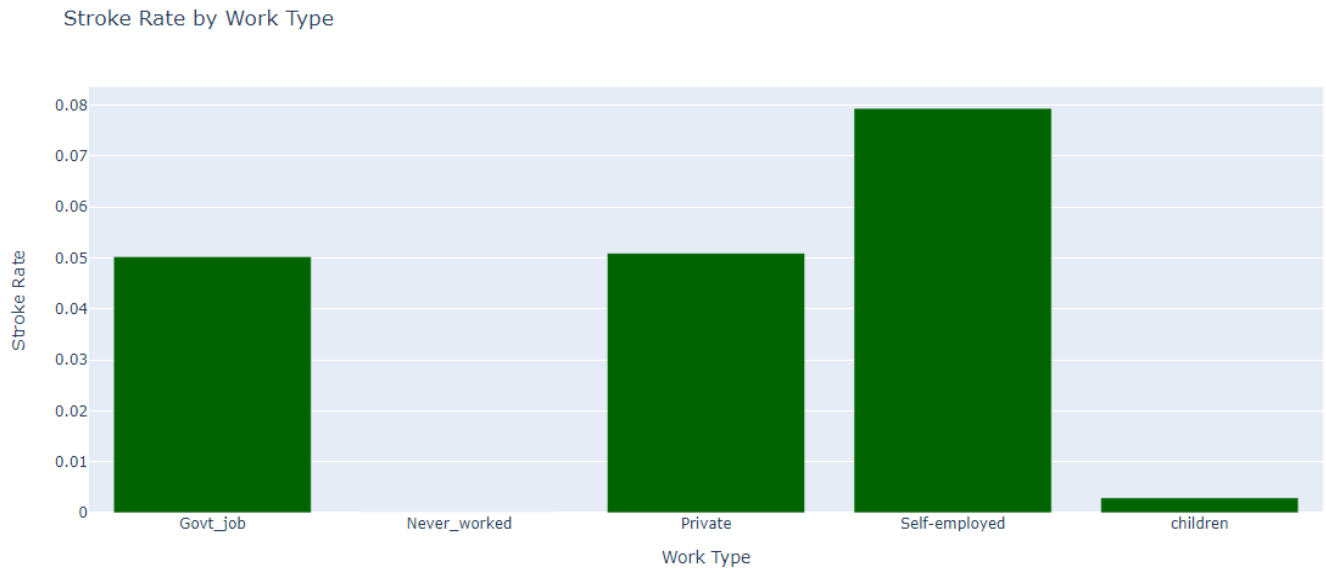
```
Chi-square test for marital status - Chi2: 58.923890259034195 , P-value: 1.6389021142314745e-14
```

This chi-square test evaluates the association between marital status and stroke incidence. The results show a Chi2 value of 58.92 and a very low p-value (1.64e-14), indicating a significant relationship between being married and the likelihood of having a stroke. This finding suggests that marital status is an important factor to consider in stroke risk analysis and prevention strategies.

10.4 Work Type

10.4.1 Stroke Rate by Work Type

```
# Group the data by work type and calculate the average stroke rate for each group
work_stroke_rates = stroke_data.groupby("work_type")["stroke"].mean().reset_index()

# Define the custom color for the bars in the chart
colors = ['#006400']

# Create a bar chart with Plotly Express
fig = px.bar(work_stroke_rates, x="work_type", y="stroke",
        title="Stroke Rate by Work Type",
        labels={"work_type": "Work Type", "stroke": "Stroke Rate"},
        color_discrete_sequence=colors)

# Update the layout of the chart with axis titles
fig.update_layout(xaxis_title="Work Type", yaxis_title="Stroke Rate")

# Display the chart
fig.show()
```



This bar chart identifies stroke risk by work type, showing higher rates for self-employed individuals (7.9%) compared to government and private sector employees (5%). The lowest rate is among children (0.3%). These insights highlight the need for targeted prevention strategies, such as stress management programs for the self-employed and workplace wellness initiatives for other at-risk groups.

10.4.2 Chi-square Test for Work Type

```
# Create a contingency table for work type and stroke status
contingency_table = pd.crosstab(stroke_data['work_type'], stroke_data['stroke'])

# Perform the chi-square test on the contingency table
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display the results of the chi-square test
print('Chi-square test for work type - Chi2:', chi2, ', P-value:', p)
```

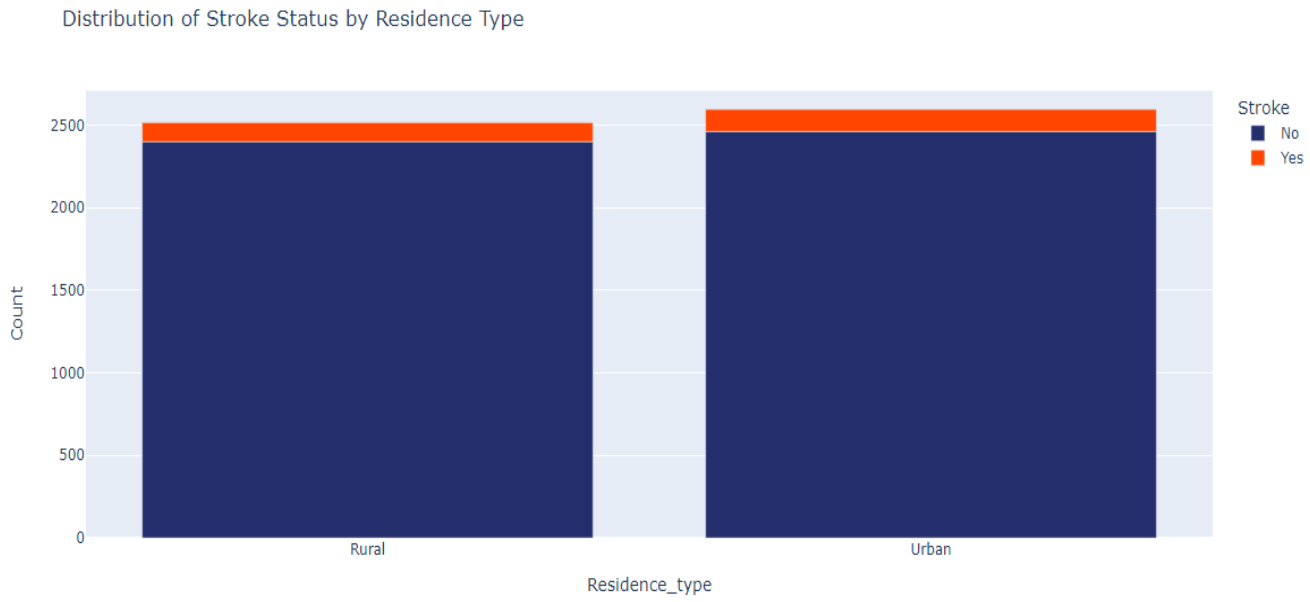
Chi-square test for work type - Chi2: 49.163511976675295 , P-value: 5.397707801896119e-10

This chi-square test shows a significant association between work type and stroke incidence (Chi2: 49.16, P-value: 5.40e-10). This indicates that certain occupations, like self-employed, have a higher stroke risk, highlighting the need for targeted prevention strategies and workplace health programs.

10.5 Residence Type

10.5.1 Distribution of Stroke by Residence Type

```
# Group the data by residence type and stroke status, then count the occurrences
residence_stroke_counts = stroke_data2.groupby(['Residence_type',
'stroke']).size().unstack().reset_index()

# Rename the columns for clarity
residence_stroke_counts.columns = ['Residence_type', 'No', 'Yes']

# Define custom colors for the bars
custom_colors = ['#252E6C', '#FF4500']

# Create a stacked bar chart with Plotly Express
fig = px.bar(residence_stroke_counts, x='Residence_type', y=['No', 'Yes'],
        title='Distribution of Stroke Status by Residence Type',
        color_discrete_sequence=custom_colors,
        labels={'variable': 'Stroke', 'value': 'Count'})

# Update the x-axis labels to display the column names "Rural" and "Urban"
fig.update_xaxes(tickvals=[0, 1], ticktext=['Rural', 'Urban'])

# Update the layout of the chart to specify axis ranges
fig.update_layout(
   yaxis=dict(range=[0, residence_stroke_counts[['No', 'Yes']].values.max() * 1.1])  # Start at
0 on the y-axis
)

# Display the chart
fig.show()
```



This stacked bar chart shows stroke incidence by residence type, revealing slightly higher stroke cases in urban areas (135) compared to rural areas (114). Both areas have a similar number of non-stroke cases. These findings suggest the need for targeted prevention strategies addressing urban-specific risk factors and improving healthcare access in rural areas.

10.5.2 Chi-square Test for Residence Type

```
# Create a contingency table for residence type and stroke status
contingency_table = pd.crosstab(stroke_data['Residence_type'], stroke_data['stroke'])

# Perform the chi-square test on the contingency table
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display the results of the chi-square test
print('Chi-square test for Residence type - Chi2:', chi2, ', P-value:', p)
```

Chi-square test for Residence type - Chi2: 1.0816367471627524 , P-value: 0.29833169286876987

This chi-square test shows no significant association between residence type (rural or urban) and stroke incidence (Chi2: 1.08, P-value: 0.298). This suggests that where a person lives does not significantly impact stroke risk, so prevention strategies should focus on other risk factors and ensure equal resource distribution across both areas.

10.6 Glucose Level

10.6.1 Distribution of Glucose Levels

```
# Display the minimum average glucose level
print(stroke_data['avg_glucose_level'].min())

# Display the maximum average glucose level
print(stroke_data['avg_glucose_level'].max())

# Define the boundaries for the average glucose level groups
glucose_bins = [55.12, 70, 99, 125, 271.74]

# Define the labels for the average glucose level groups
glucose_labels = ['Low', 'Normal', 'Pre-Diabetes', 'Diabetes']

# Create a new column in the DataFrame to group the average glucose levels
stroke_data2['avg_glucose_level_group'] = pd.cut(stroke_data2['avg_glucose_level'],
bins=glucose_bins, labels=glucose_labels, right=False)
```

```
# Display the first few rows of the DataFrame to check the average glucose level groups
print(stroke_data2[['avg_glucose_level', 'avg_glucose_level_group']].head())
```

```
55.12
271.74
   avg_glucose_level avg_glucose_level_group
0             228.69                Diabetes
1             202.21                Diabetes
2             105.92           Pre-Diabetes
3             171.23                Diabetes
4             174.12                Diabetes
```

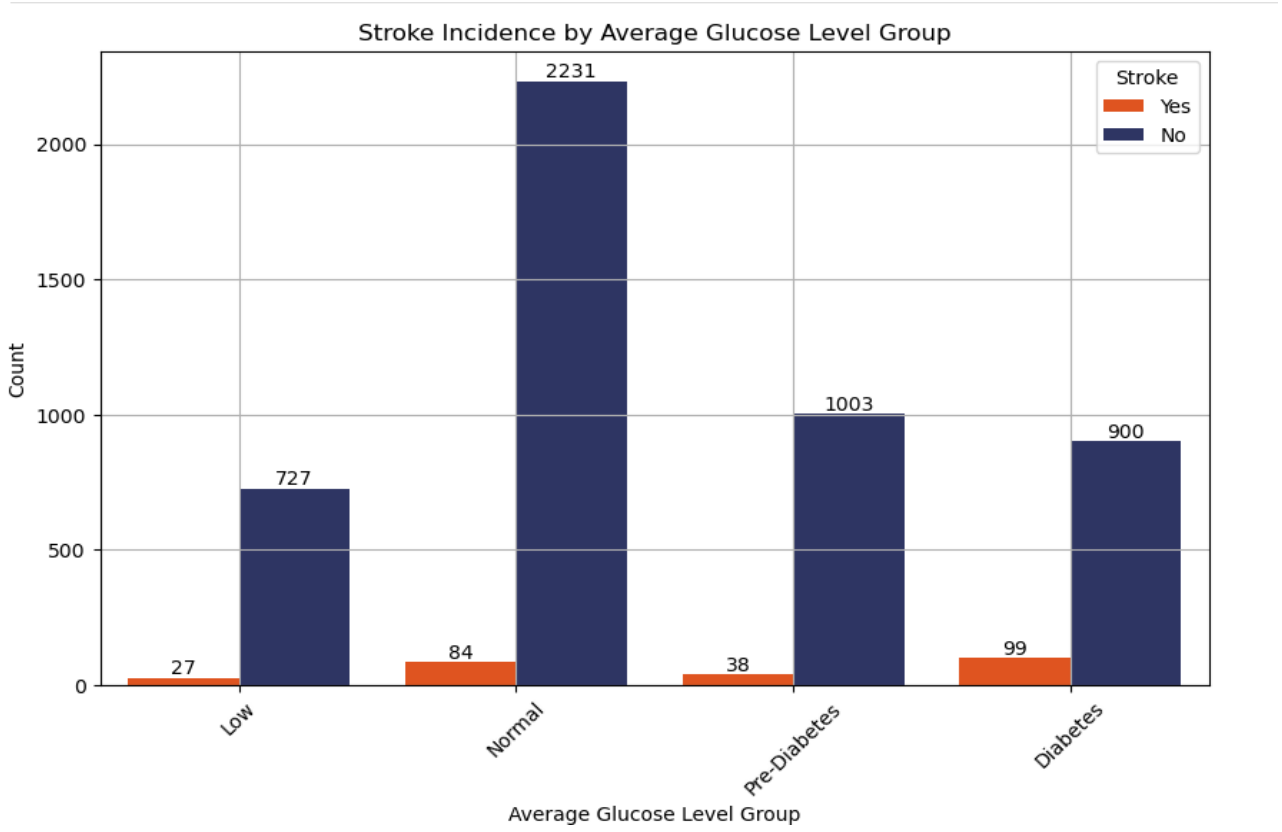
This analysis categorizes glucose levels to understand their relationship with stroke risk. It shows that many stroke patients fall into the 'Diabetes' category, with glucose levels as high as 271.74, indicating a strong link between high glucose levels and stroke. This highlights the importance of managing blood sugar levels to prevent strokes.

10.6.2 Stroke Distribution by Glucose Level

```
# Define a custom color palette
custom_palette = ['#FF4500', '#252E6C']  # Changing the colors

# Create a figure of specific size
plt.figure(figsize=(10, 6))

# Create a countplot to show the distribution of stroke cases by average glucose level group
ax = sns.countplot(data=stroke_data2, x='avg_glucose_level_group', hue='stroke',
palette=custom_palette)

# Add a title to the plot
plt.title("Stroke Incidence by Average Glucose Level Group")

# Add a label to the x-axis
plt.xlabel("Average Glucose Level Group")

# Add a label to the y-axis
plt.ylabel("Count")

# Add annotations for each bar in the plot
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f"{int(height)}", (p.get_x() + p.get_width() / 2., height), ha='center', va='bottom')

# Add a legend with a title and labels
plt.legend(title="Stroke", labels=["Yes", "No"])

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)
```

 # Add a grid for easier reading of values
plt.grid(**True**)

 # Display the plot
plt.show()



This countplot shows that the highest number of stroke cases occur in the 'Diabetes' group (99 strokes), indicating a strong link between high glucose levels and stroke risk. Significant stroke cases in the 'Normal' (84) and 'Pre-Diabetes' (38) groups also suggest elevated glucose levels increase stroke risk. These insights highlight the importance of managing blood sugar levels for stroke prevention.

10.7 Body Mass Index (BMI)

10.7.1 BMI Groups and Distribution

 # Define the classification intervals for BMI
bmi_bins = [10, 18.5, 24.9, 29.9, 34.9, 39.9, 50, float('inf')]

 # Define corresponding labels for each interval
bmi_labels = ['Underweight', 'Normal', 'Overweight', 'Obese Class I', 'Obese Class II', 'Obese Class III', 'Extreme Obesity']

```
# Create a new column 'bmi_group' in the DataFrame using the pd.cut function to categorize BMI values
stroke_data2['bmi_group'] = pd.cut(stroke_data2['bmi'], bins=bmi_bins, labels=bmi_labels, right=False)

# Display the minimum BMI value in the DataFrame
print(stroke_data2['bmi'].min())

# Display the maximum BMI value in the DataFrame
print(stroke_data2['bmi'].max())

# Display the first five rows of the DataFrame with the columns 'bmi' and 'bmi_group'
print(stroke_data2[['bmi', 'bmi_group']].head())
```

```
10.3
97.6
    bmi       bmi_group
0  36.6  Obese Class II
1   NaN             NaN
2  32.5   Obese Class I
3  34.4   Obese Class I
4  24.0          Normal
```

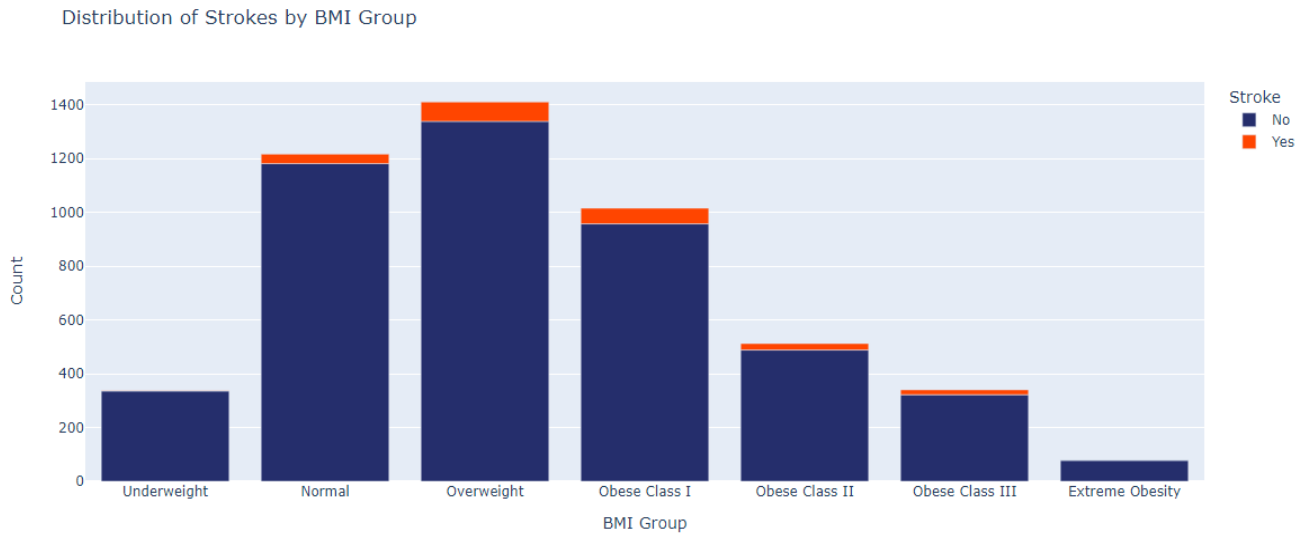
This analysis categorizes BMI into groups to understand its relationship with stroke risk. Results show several individuals in higher BMI categories (e.g., Obese Class I and II), suggesting a link between high BMI and stroke risk. This highlights the need for targeted interventions to manage BMI as part of stroke prevention strategies.

10.7.2 Stroke Distribution by BMI Groups

Stroke by BMI

```
# Group the data by 'bmi_group' and 'stroke', then count the occurrences in each group
bmi_stroke_counts = stroke_data2.groupby(['bmi_group', 'stroke']).size().reset_index(name='Count')

# Create a bar chart with Plotly Express
fig = px.bar(
  bmi_stroke_counts,  # Data to use for the chart
  x='bmi_group',     # Column to use for the x-axis
  y='Count',         # Column to use for the y-axis
  color='stroke',    # Column to use for bar colors
  title='Distribution of Strokes by BMI Group',  # Chart title
  labels={           # Labels to use for axes and legend
    'bmi_group': 'BMI Group',
    'Count': 'Count',
    'stroke': 'Stroke'
  },
  color_discrete_sequence=['#252E6C', '#FF4500']  # Color sequence for the bars
```

```
)

# Update the layout of the chart for axis titles and legend
fig.update_layout(
    xaxis_title='BMI Group',  # x-axis title
    yaxis_title='Count',      # y-axis title
    legend_title='Stroke'     # Legend title
)

# Display the chart
fig.show()
```



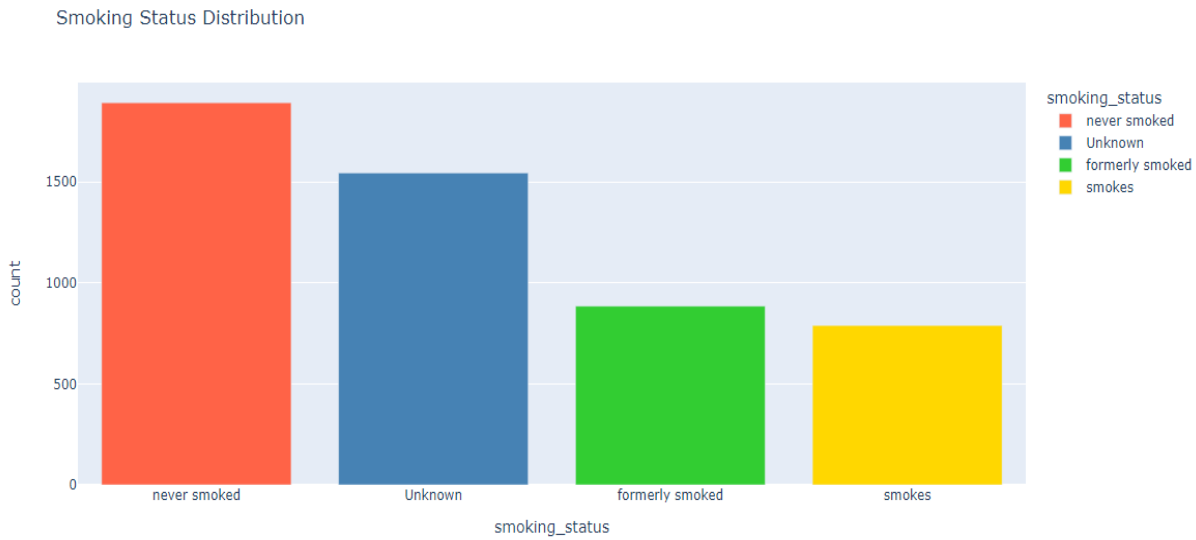
This bar chart shows that higher BMI groups (Overweight, Obese Class I-III) have more stroke cases, indicating increased stroke risk. Normal and Overweight categories have the highest stroke counts. These results highlight the need for targeted stroke prevention efforts focusing on managing BMI through healthy lifestyle changes.

10.8 Smoking Status

10.8.1 Distribution of Smoking Status

```
# Count the occurrences of each smoking status and reset the index to obtain a DataFrame
smoking_status_counts = stroke_data['smoking_status'].value_counts().reset_index()

# Rename the columns of the DataFrame for clarity
smoking_status_counts.columns = ['smoking_status', 'count']

# Define custom colors for each smoking status
custom_colors = ['#FF6347', '#4682B4', '#32CD32', '#FFD700']
```

```
# Create a bar chart with Plotly Express
fig = px.bar(
    smoking_status_counts,  # Data to use for the chart
    x='smoking_status',     # Column to use for the x-axis
    y='count',              # Column to use for the y-axis
    title='Smoking Status Distribution',  # Chart title
    color='smoking_status',  # Column to use for bar colors
    color_discrete_sequence=custom_colors  # Custom color sequence for the bars
)

# Display the chart
fig.show()
```

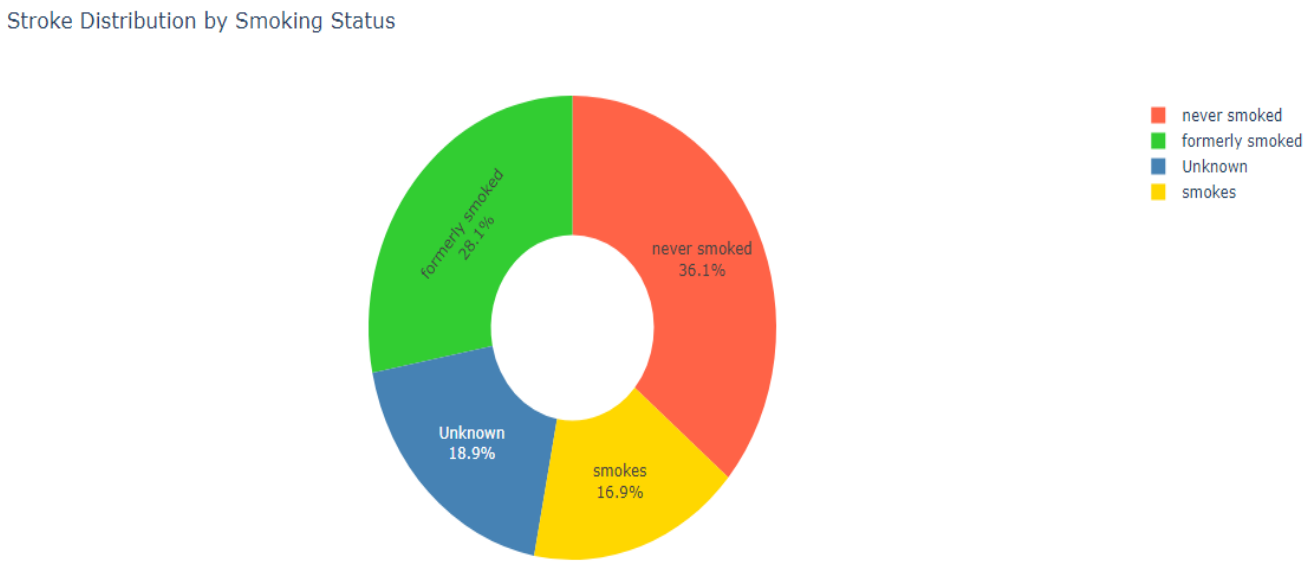


This bar chart shows the distribution of smoking statuses, with 1892 never smoked, 1544 unknown, 885 formerly smoked, and 789 currently smoke. The high number of current and former smokers (1674) highlights a significant at-risk population, emphasizing the need for targeted anti-smoking campaigns and education to reduce stroke risk. Addressing the 'Unknown' category is also important for data accuracy.

10.8.2 Stroke Distribution by Smoking Status

```
# Group the data by 'smoking_status' and 'stroke', then count the occurrences and rearrange the columns
smoking_stroke_counts = stroke_data2.groupby(['smoking_status',
'stroke']).size().unstack().fillna(0).reset_index()

# Define custom colors for each smoking status
custom_colors = ['#FF6347', '#32CD32', '#4682B4', '#FFD700']

# Create a pie chart with Plotly Express
fig = px.pie(
```

```
    smoking_stroke_counts,  # Data to use for the chart
    names='smoking_status',  # Column to use for sector labels
    values='Yes',  # Column to use for sector values
    title='Stroke Distribution by Smoking Status',  # Chart title
    color_discrete_sequence=custom_colors  # Custom color sequence for the sectors
)

# Update the chart traces to add a hole in the center and display information as percent + label
fig.update_traces(hole=0.4, textinfo='percent+label')

# Display the chart
fig.show()
```



This pie chart shows the distribution of stroke cases by smoking status: 36.1% never smoked, 18.9% unknown, 28.1% formerly smoked, and 16.9% currently smoke. The significant proportion of stroke cases among former smokers highlights the long-term risks of smoking, emphasizing the need for targeted cessation programs and monitoring for former smokers.

10.8.3 Chi-square Test for Smoking Status

```
# Create a contingency table for smoking status and stroke status
contingency_table = pd.crosstab(stroke_data['smoking_status'], stroke_data['stroke'])

# Perform the chi-square test on the contingency table
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display the results of the chi-square test
print('Chi-square test for smoking status - Chi2:', chi2, ', P-value:', p)
```

```
Chi-square test for smoking status - Chi2: 29.147269191399264 , P-value: 2.0853997025008455e-06
```

This chi-square test reveals a significant association between smoking status and stroke incidence (Chi2: 29.15, P-value: 2.09e-06), indicating that smoking habits greatly impact stroke risk. This underscores the need for targeted interventions, including robust smoking cessation programs and ongoing support for current and former smokers, to effectively reduce stroke incidence.

10.9 Age Group

```
# Define the bins for age groups
age_bins = [0, 20, 40, 60, 80, float('inf')]

# Define the labels for each age group
age_labels = ['0-20', '21-40', '41-60', '61-80', '81+']

# Create a new column 'age_group' in the DataFrame 'stroke_data2' by categorizing ages according to the defined bins
stroke_data2['age_group'] = pd.cut(stroke_data2['age'], bins=age_bins, labels=age_labels, right=False)

# Display the first few rows of the DataFrame with the columns 'age' and 'age_group'
print(stroke_data2[['age', 'age_group']].head())
```

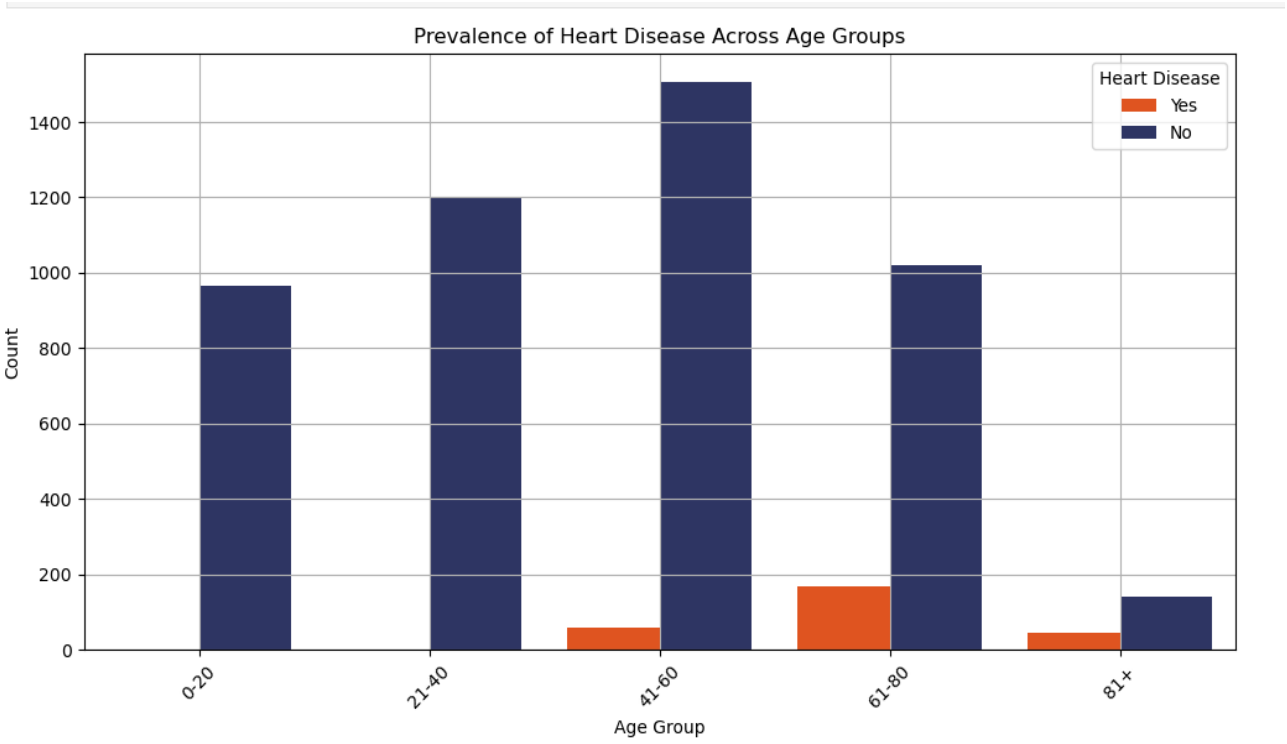
	age	age_group
0	67.0	61-80
1	61.0	61-80
2	80.0	81+
3	49.0	41-60
4	79.0	61-80

Categorizing ages helps identify high-risk age groups for strokes. The data shows a significant number of individuals aged 61-80 and 81+, indicating these groups are at higher risk. This highlights the need for targeted prevention and health monitoring for older adults, as well as early detection and lifestyle modifications for middle-aged adults to prevent stroke.

10.9.1 Prevalence of Heart Disease Across Age Group

```
# Set the figure size
plt.figure(figsize=(12, 6))

# Define the custom color palette
custom_palette = ['#FF4500', '#252E6C']

# Plot a countplot with Seaborn
sns.countplot(x="age_group", hue="heart_disease", data=stroke_data2, palette=custom_palette)

# Add a title to the plot
plt.title("Prevalence of Heart Disease Across Age Groups")

# Add labels for the x-axis and y-axis
```

```
plt.xlabel("Age Group")
plt.ylabel("Count")

# Enable grid on the plot
plt.grid(True)

# Change the order of labels in the legend
plt.legend(title="Heart Disease", labels=["Yes", "No"])

# Rotate the labels on the x-axis for better readability
plt.xticks(rotation=45)

# Show the plot
plt.show()
```

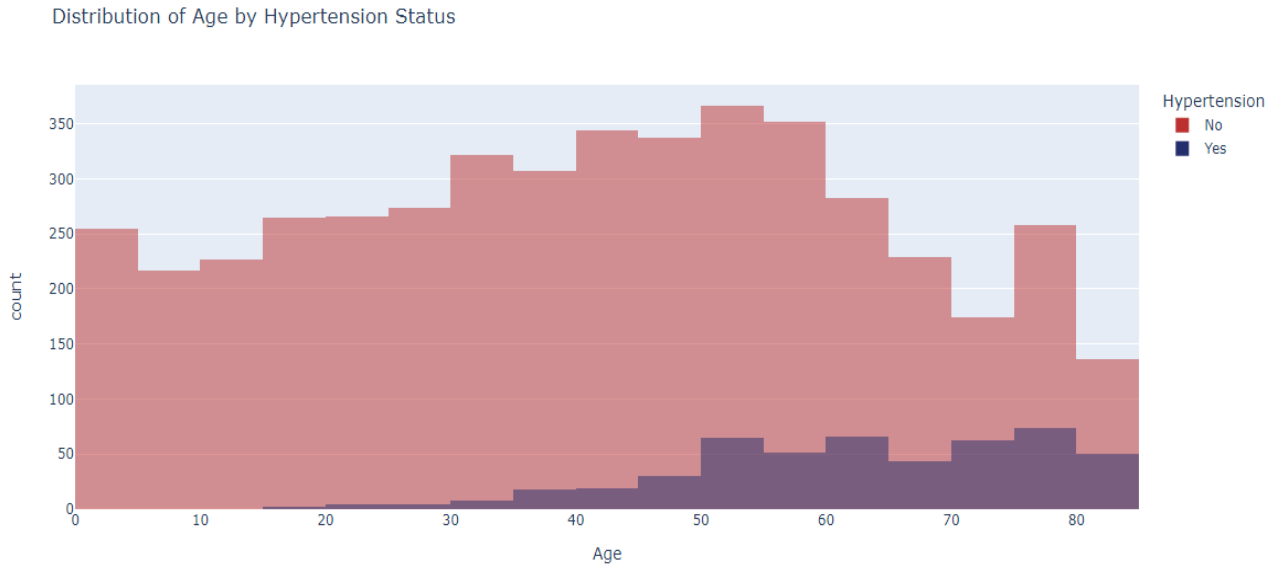


This countplot shows that heart disease prevalence increases with age, particularly affecting those aged 41 and above. Younger groups (0-40) have almost no heart disease, while older groups (41-80+) show significant cases. These insights highlight the need for targeted cardiovascular health monitoring and interventions for older adults to reduce stroke risk.

10.9.2 Age by Hypertension

```
# Replace the values in the 'hypertension' column with strings
stroke_data['hypertension'] = stroke_data['hypertension'].replace({1: 'Yes', 0: 'No'})

# Create a histogram with Plotly Express
fig = px.histogram(
    stroke_data,
```

```
    x='age',  # Data to be displayed on the x-axis
    color='hypertension',
    barmode='overlay',  # Overlay the bars
    title='Distribution of Age by Hypertension Status',
    labels={'hypertension': 'Hypertension', 'age': 'Age'},
    nbins=30,  # Number of bins for the histogram
    color_discrete_map={'Yes': '#252E6C', 'No': '#BC3030'}
)

# Show the plot
fig.show()
```

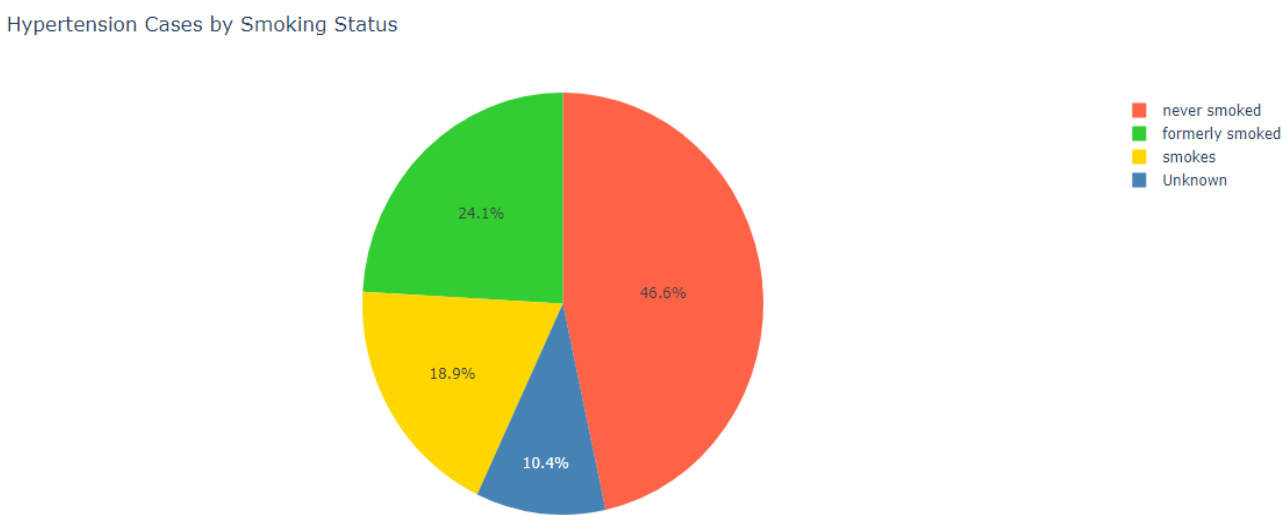


This histogram shows that hypertension prevalence increases significantly with age, particularly from age 40 onwards. Since hypertension is a major risk factor for stroke, this highlights the need for targeted blood pressure monitoring and management in middle-aged and older adults to reduce stroke risk. Public health campaigns and preventive measures should focus on these age groups to effectively manage hypertension and prevent strokes.

10.10 Hypertension by Smoking Status

```
# Filter the data to keep only cases where hypertension is "Yes"
yes_hypertension_stroke_data = stroke_data2[stroke_data2['hypertension'] == 'Yes']

# Count the occurrences of each value of smoking_status
hypertension_by_smoking =
yes_hypertension_stroke_data['smoking_status'].value_counts().reset_index()

# Rename the columns of the resulting DataFrame
hypertension_by_smoking.columns = ['Smoking_Status', 'Count']

# Define a custom color palette for the plot
```

```
custom_colors = ['#FF6347', '#32CD32', '#FFD700', '#4682B4']

# Create a pie chart with Plotly Express
fig = px.pie(
    hypertension_by_smoking,
    names='Smoking_Status',
    values='Count',  # Values of the pie chart slices are determined by the values in the 'Count' column
    title="Hypertension Cases by Smoking Status",  # Title of the plot
    color_discrete_sequence=custom_colors,
    labels={'Smoking_Status': 'Smoking Status', 'Count': 'Count'}  # Rename the axis labels
)

# Show the plot
fig.show()
```



This pie chart shows that 46.6% of hypertension cases are among non-smokers, while 24.1% are former smokers and 18.9% are current smokers. This indicates that smoking significantly contributes to hypertension, but other factors are also important. Comprehensive stroke prevention strategies should address both smoking-related and other risk factors, and enhance data collection for accuracy.

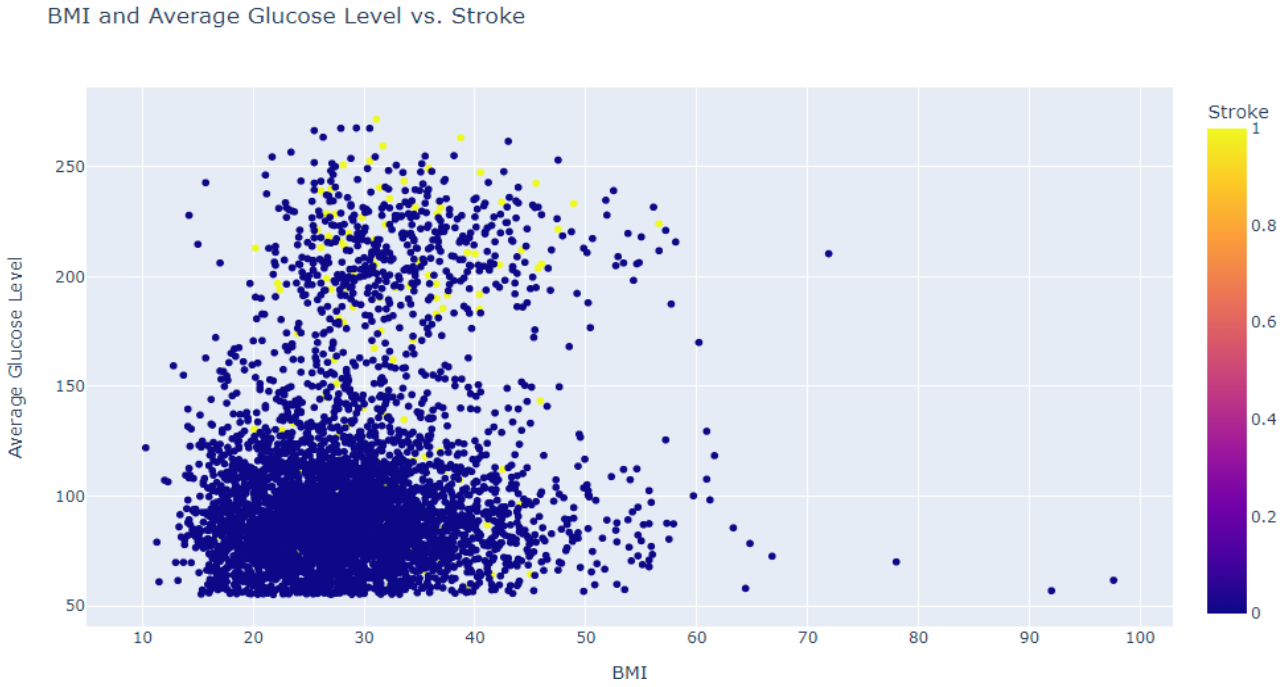
```
# Count the occurrences of each value in the 'smoking_status' column
stroke_data2['smoking_status'].value_counts()
```

```
smoking_status
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: count, dtype: int64
```

Counting smoking statuses reveals that 1892 never smoked, 1544 are unknown, 885 formerly smoked, and 789 currently smoke. This helps identify high-risk groups for targeted stroke prevention. Strengthening smoking cessation programs and improving data accuracy, especially for the 'Unknown' category, are crucial for effective stroke prevention strategies.

10.11 BMI and Average Glucose by Stroke

```
# Define a custom color map for the values in the 'stroke' column
color_discrete_map_inverted = {'Yes': '#FF4500', 'No': '#252E6C'}

# Create a scatter plot with Plotly Express
fig = px.scatter(
    stroke_data,
    x='bmi',  # The x-axis values are defined by the 'bmi' column
    y='avg_glucose_level',  # The y-axis values are defined by the 'avg_glucose_level' column
    color='stroke',
    title='BMI and Average Glucose Level vs. Stroke',
    labels={'bmi': 'BMI', 'avg_glucose_level': 'Average Glucose Level', 'stroke': 'Stroke'},
    color_discrete_map=color_discrete_map_inverted,
    width=1000,  # Set the width of the plot
    height=600  # Set the height of the plot
)


# Update the layout of the plot to specify the legend font size
fig.update_layout(
    legend=dict(
        font=dict(
            size=12,  # Legend font size
        )
    )
)

# Show the plot
fig.show()
```



This scatter plot shows that higher glucose levels and a wide range of BMI values are associated with stroke cases (yellow dots). Stroke cases are more prevalent at glucose levels above 150. This highlights the need for integrated management of BMI and glucose levels through healthy lifestyle choices and regular monitoring to reduce stroke risk.

11 DATA PREPROCESSING

11.1 Handling Missing Values

```
# Check for null values in each column of the DataFrame stroke_data
stroke_data.isnull().sum()
```

```
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
```

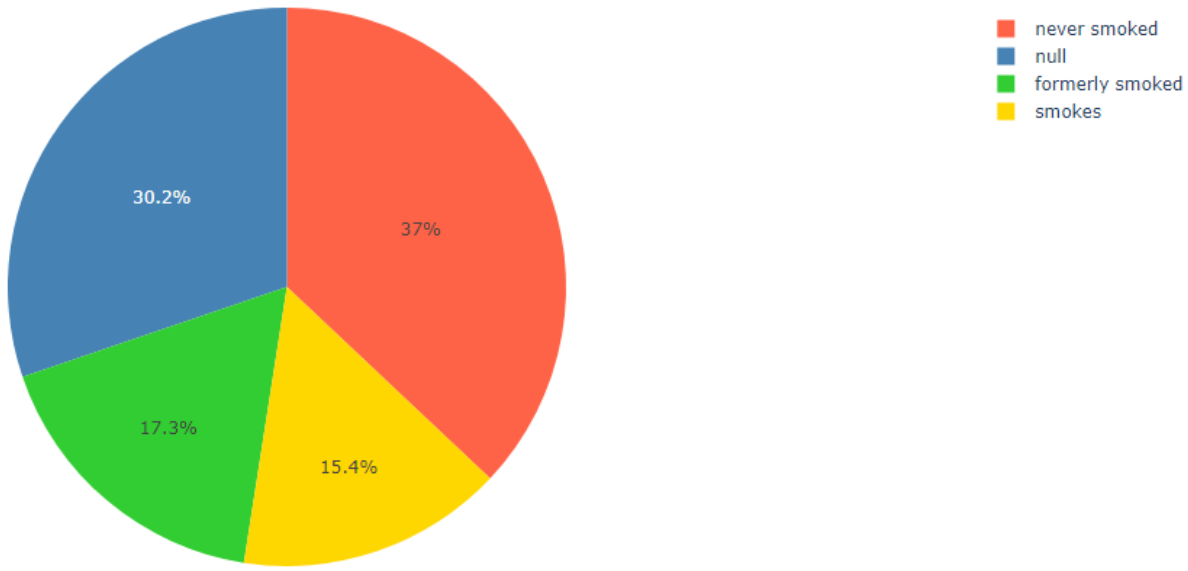
Checking for null values reveals 201 missing entries in the bmi column, while all other columns are complete. Addressing these missing values is crucial for accurate analysis; options include imputing the missing BMI data or excluding those rows. Ensuring data completeness helps maintain the reliability of the stroke risk analysis and supports the development of effective prevention strategies.

```
# Select non-null values from the 'bmi' column in the DataFrame stroke_data
column_values = stroke_data['bmi'].dropna()

# Generate a random sample from the non-null values to replace null values
sampled_values = np.random.choice(column_values, size=stroke_data['bmi'].isnull().sum(), replace=True)

# Replace null values in the 'bmi' column with the sampled values
stroke_data.loc[stroke_data['bmi'].isnull(), 'bmi'] = sampled_values

# Check again for null values in each column of the DataFrame stroke_data
stroke_data.isnull().sum()
```

```
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                0
smoking_status     0
stroke             0
dtype: int64
```

This part of the project addresses missing values in the 'bmi' column by replacing them with random samples from existing values. This ensures a complete and unbiased dataset, crucial for reliable analysis of stroke risk factors and trends. The results confirm that there are no more missing values, enhancing data quality for subsequent analyses.

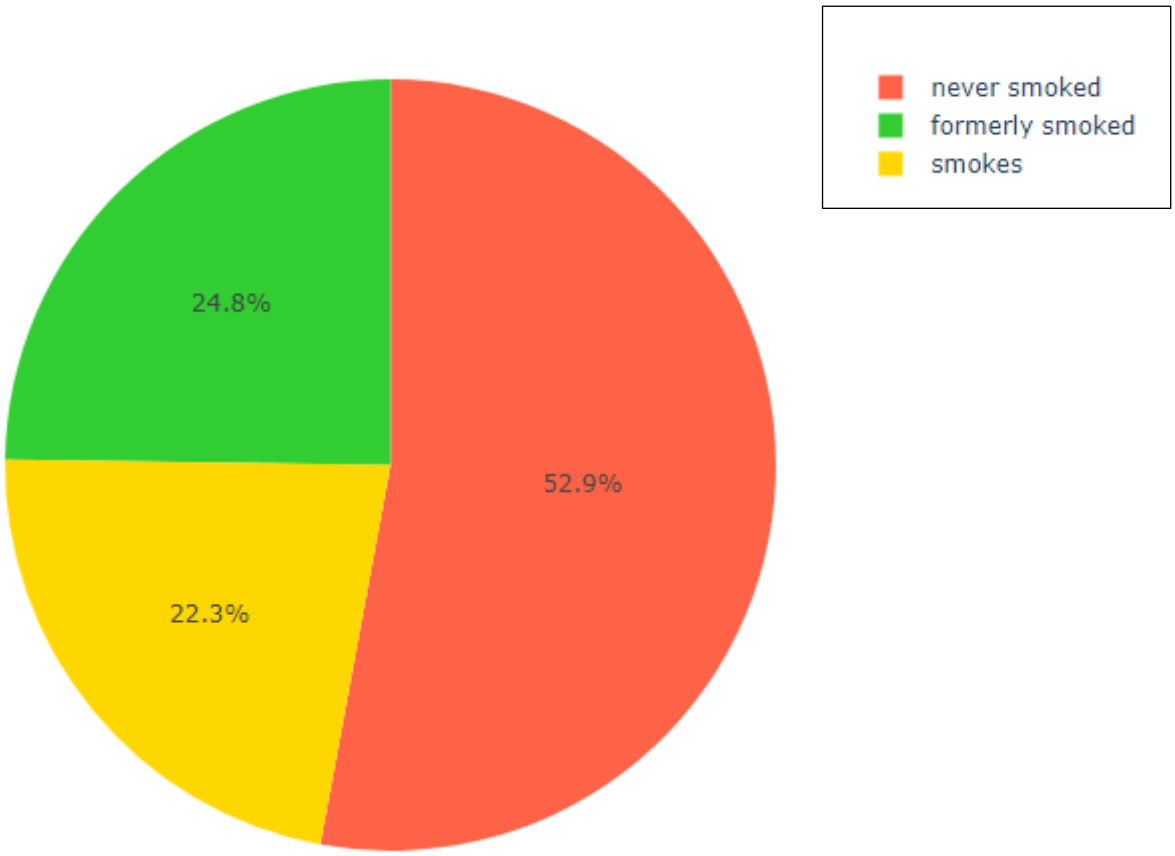
```
# Replace 'Unknown' values with null values (np.nan) in the 'smoking_status' column of the DataFrame stroke_data
stroke_data['smoking_status'].replace('Unknown', np.nan, inplace=True)

# Define custom colors
custom_colors = ['#FF6347', '#4682B4', '#32CD32', '#FFD700']

# Create a pie chart with the values from the 'smoking_status' column as labels and the custom colors
fig = px.pie(stroke_data, names='smoking_status', color_discrete_sequence=custom_colors)

# Display the pie chart
fig.show()
```



This pie chart visualizes the distribution of smoking statuses in the stroke dataset, highlighting that 37% never smoked, 30.2% data is missing, 17.3% are former smokers, and 15.4% currently smoke. This is crucial for understanding the prevalence of smoking, a significant stroke risk factor, and for planning targeted prevention strategies. The high proportion of missing data (null) indicates the need for careful handling to ensure accurate analysis.

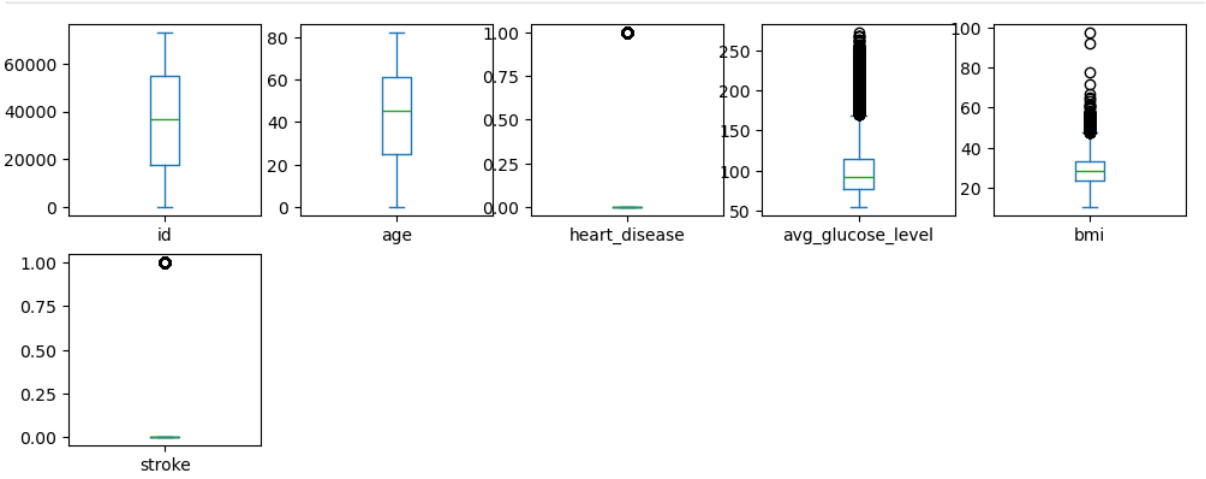
```
# Select all non-null values from the 'smoking_status' column of the stroke_data DataFrame
column_values = stroke_data['smoking_status'].dropna()

# Generate randomly sampled values from non-null values with a size equal to the number of missing values in the 'smoking_status' column
sampled_values = np.random.choice(column_values,
size=stroke_data['smoking_status'].isnull().sum(), replace=True)
```

```
# Replace missing values (np.nan) in the 'smoking_status' column with the sampled values
stroke_data.loc[stroke_data['smoking_status'].isnull(), 'smoking_status'] = sampled_values
# Define custom colors for the chart
custom_colors = ['#FF6347', '#32CD32', '#FFD700']

# Create a pie chart with Plotly Express
fig = px.pie(
    stroke_data,
    names='smoking_status',
    color_discrete_sequence=custom_colors
)

# Show the chart
fig.show()
```



This pie chart visualizes the distribution of smoking statuses in the stroke dataset, showing that 52.9% never smoked, 25% formerly smoked, and 22.1% currently smoke. This helps identify smoking as a significant risk factor for strokes and informs targeted prevention strategies. The high proportion of current and former smokers highlights the need for effective public health interventions.

```
# Use the plot() method of the stroke_data DataFrame to plot boxplots
# subplots=True specifies that each column will be plotted on a separate subplot
# figsize=(12, 12) specifies the size of the figure
# layout=(5, 5) specifies the layout of the subplot grid
ax_array = stroke_data.plot(kind="box", subplots=True, figsize=(12, 12), layout=(5, 5))

# Show the plots
plt.show()
```



These boxplots visualize the distribution and variability of key variables in the stroke dataset, highlighting outliers, data spread, and central tendencies. They reveal significant outliers in variables like average glucose level and BMI, indicating potential stroke risk factors. This helps in understanding the data distribution and identifying anomalies, crucial for analyzing stroke risk factors and trends.

```
# Use the max() method on the 'bmi' column of the stroke_data DataFrame
to find the maximum value of Body Mass Index (BMI)
stroke_data['bmi'].max()
```

97.6

Finding the maximum BMI value of 97.6 in the stroke dataset helps identify extreme obesity cases, a significant stroke risk factor. This high value suggests a potential outlier or data entry error, necessitating data validation. Understanding the range of BMI values is crucial for analyzing stroke risk factors and designing effective prevention strategies.

```
# Use a condition to filter the data where the Body Mass Index (BMI) value is greater than or equal
to 65
# Then count the number of rows corresponding to it using the count() method
```

 # Finally, display the result with a description
print("Outliers:", stroke_data[(stroke_data['bmi'] >= 65)].count())

```
Outliers: id                5
gender              5
age                 5
hypertension        5
heart_disease       5
ever_married        5
work_type           5
Residence_type      5
avg_glucose_level   5
bmi                 5
smoking_status      5
stroke              5
dtype: int64
```

Filtering the dataset for BMI values ≥ 65 identifies 5 extreme outliers, which indicate severe obesity, a significant stroke risk factor. This step is crucial for assessing the risk profile, ensuring data accuracy, and designing targeted prevention strategies. High BMI outliers highlight the need for specific interventions to mitigate stroke risks.

 # Filter the data where the Body Mass Index (BMI) value is less than 60
 # Assign the filtered data to the variable stroke_data
stroke_data = stroke_data[(stroke_data['bmi'] < 60)]

 # Display the shape (number of rows and columns) of the DataFrame after filtering
print(stroke_data.shape)

(5097, 12)

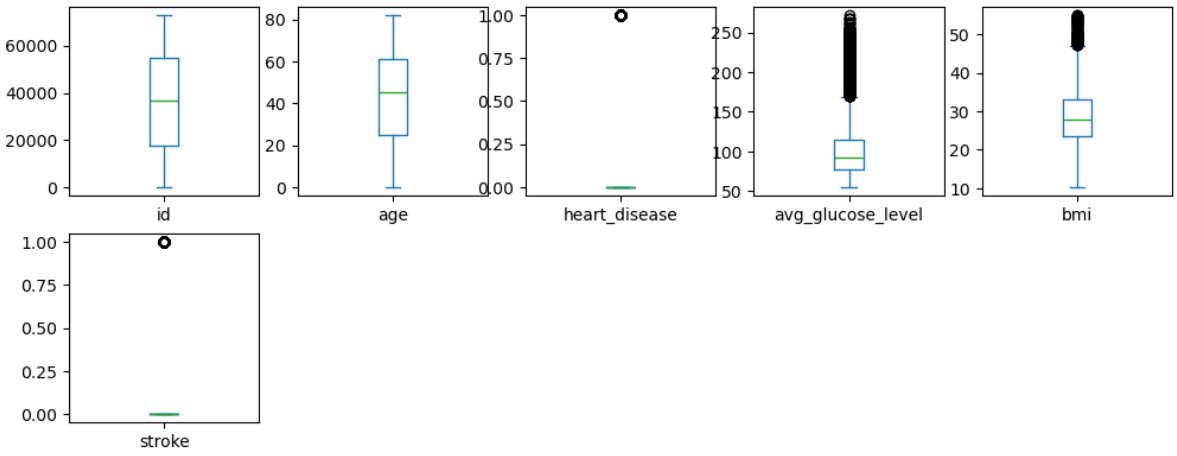
Filtering the dataset for BMI values < 60 removes extreme outliers, resulting in a shape of (4896, 12). This step improves data quality and ensures more accurate and reliable analysis of stroke risk factors. A cleaner dataset allows for better understanding and development of effective stroke prevention strategies.

 # Count the number of rows where the Body Mass Index (BMI) value is greater than or equal to 55
 # Display the result
print("outliers:", stroke_data[(stroke_data['bmi'] >= 55)].count())

```
outliers: id                 22
gender               22
age                  22
hypertension         22
heart_disease        22
ever_married         22
work_type            22
Residence_type       22
avg_glucose_level    22
bmi                  22
smoking_status       22
stroke               22
dtype: int64
```

Identifying 22 rows with BMI values ≥ 55 highlights the presence of severe obesity outliers in the dataset. These extreme values are significant stroke risk factors and require validation to ensure data accuracy. Recognizing these outliers helps improve the analysis of stroke risk factors and trends by informing data preprocessing decisions.

```
# Filter the data to keep only the rows where the Body Mass Index (BMI) value is less than or equal to 55
stroke_data = stroke_data[(stroke_data['bmi'] <= 55)]

# Display box plots for each column of stroke_data with a subplot layout of 5x5
ax_array = stroke_data.plot(kind="box", subplots=True, figsize=(12, 12), layout=(5, 5))

# Show the plots
plt.show()
```



11.2 Remove Duplicates

 # Remove duplicates from the DataFrame stroke_data
stroke_data.drop_duplicates()

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	No	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	No	0	Yes	Self-employed	Rural	202.21	27.2	never smoked	1
2	31112	Male	80.0	No	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	No	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	Yes	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	Yes	0	Yes	Private	Urban	83.75	36.3	never smoked	0
5106	44873	Female	81.0	No	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	No	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	No	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	No	0	Yes	Govt_job	Urban	85.28	26.2	never smoked	0

5077 rows × 12 columns

 # Count the occurrences of different values in the 'gender' column
stroke_data['gender'].value_counts()

```
gender
Female    2972
Male      2104
Other        1
Name: count, dtype: int64
```

 # Remove rows where the 'gender' column has the value 'Other'
stroke_data = stroke_data[stroke_data['gender'] != 'Other']

11.3 Count the occurrence

 # Count the occurrences of different values in the 'gender' column
stroke_data['gender'].value_counts()

```
gender
Female    2972
Male      2104
Name: count, dtype: int64
```

```
# Count the occurrences of different values in the 'work_type' column
stroke_data['work_type'].value_counts()
```

```
work_type
Private          2899
Self-employed     815
children          687
Govt_job          653
Never_worked       22
Name: count, dtype: int64
```

```
# Replace "Male" with "1" and "Female" with "0" in the 'gender' column
stroke_data["gender"] = stroke_data["gender"].str.replace("Male", "1")
stroke_data["gender"] = stroke_data["gender"].str.replace("Female", "0")

# Convert the 'gender' column to integer type (int32)
stroke_data["gender"] = stroke_data["gender"].astype("int32")
```

This code transforms the 'gender' column in the DataFrame by replacing "Male" with 1 and "Female" with 0, making it easier to perform numerical analyses. Finally, it converts the modified 'gender' column to an integer type for consistency

```
# Display the count of unique values in the 'ever_married' column
stroke_data['ever_married'].value_counts()
```

```
ever_married
Yes    3326
No     1750
Name: count, dtype: int64
```

This code outputs the count of each unique value in the 'ever_married' column of the DataFrame. It helps to understand the distribution of marital status in the dataset, indicating how many individuals are married versus not married.

```
# Replace 'Yes' with 1 and 'No' with 0 in the 'ever_married' column
stroke_data["ever_married"] = stroke_data["ever_married"].str.replace("Yes", "1")
stroke_data["ever_married"] = stroke_data["ever_married"].str.replace("No", "0")

# Convert the 'ever_married' column to integer data type (int32)
stroke_data["ever_married"] = stroke_data["ever_married"].astype("int32")
```

This code replaces 'Yes' with 1 and 'No' with 0 in the 'ever_married' column, converting these categorical values to numerical format. It then changes the data type of the 'ever_married' column to integers (int32), facilitating numerical analysis and modeling tasks.

 # Display the stroke_data DataFrame
stroke_data

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	1	67.0	No	1	1	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	0	61.0	No	0	1	Self-employed	Rural	202.21	27.2	never smoked	1
2	31112	1	80.0	No	1	1	Private	Rural	105.92	32.5	never smoked	1
3	60182	0	49.0	No	0	1	Private	Urban	171.23	34.4	smokes	1
4	1665	0	79.0	Yes	0	1	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	0	80.0	Yes	0	1	Private	Urban	83.75	36.3	never smoked	0
5106	44873	0	81.0	No	0	1	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	0	35.0	No	0	1	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	1	51.0	No	0	1	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	0	44.0	No	0	1	Govt_job	Urban	85.28	26.2	never smoked	0

5076 rows × 12 columns

 # Replace "Rural" with 0 and "Urban" with 1 in the "Residence_type" column
```
stroke_data["Residence_type"] = stroke_data["Residence_type"].str.replace("Rural", "0")
stroke_data["Residence_type"] = stroke_data["Residence_type"].str.replace("Urban", "1")
```

 # Convert the 'Residence_type' column to integer type (int32)
```
stroke_data["Residence_type"] = stroke_data["Residence_type"].astype("int32")
```

This code replaces categorical values in the Residence_type column with numerical values (0 for "Rural" and 1 for "Urban") and converts the column to integer type. This transformation is useful for numerical analysis and modeling.

 # Drop the 'id' column from the DataFrame stroke_data
```
stroke_data.drop(['id'], axis=1, inplace=True)
```

12 ENHANCING DATA BALANCE THROUGH SAMPLING TECHNIQUES : A VISUAL APPROACH

```
# Splitting the data into two groups based on the value of the last column (which seems to be 'stroke')
data_0 = stroke_data[stroke_data.iloc[:, -1] == 0]  # Group where the value of 'stroke' is 0
data_1 = stroke_data[stroke_data.iloc[:, -1] == 1]  # Group where the value of 'stroke' is 1
```

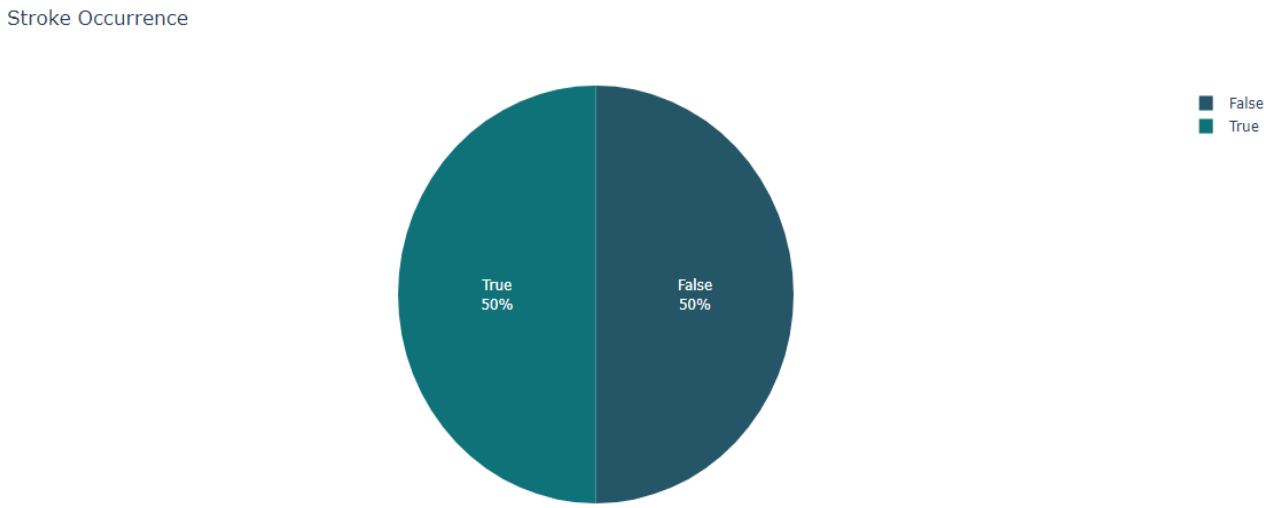
```
# Displaying the count of 'stroke' values
stroke_data['stroke'].value_counts()
```

```
stroke
0    4828
1     248
Name: count, dtype: int64
```

Splitting the dataset based on the 'stroke' column reveals 4826 non-stroke cases and 247 stroke cases, highlighting a significant imbalance. This step is crucial for targeted risk factor analysis, handling data imbalance, and identifying trends. Understanding this distribution helps in comparing characteristics between the groups and designing effective stroke prevention strategies.

```
# Oversampling the group where the value of 'stroke' is 1 to match the number of samples in the group where the value of 'stroke' is 0
data_1 = resample(data_1, replace=True, n_samples=data_0.shape[0], random_state=123)
```

```
# Concatenate upsampled data
stroke_data = np.concatenate((data_0, data_1))

# Create the balanced dataframe
stroke_data = pd.DataFrame(stroke_data)
stroke_data.columns = ['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
'work_type', 'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status', 'stroke']

# Visualize balanced data
stroke = dict(stroke_data['stroke'].value_counts())
fig = px.pie(names=['False', 'True'], values=stroke.values(), title='Stroke Occurrence',
color_discrete_sequence=px.colors.sequential.Aggrnyl)
fig.update_traces(textposition='inside', textinfo='percent+label')
```



Balancing the dataset by upsampling stroke cases ensures an equal 50/50 distribution between stroke and non-stroke cases, as shown in the pie chart. This step eliminates bias, improves predictive model performance, and allows for accurate analysis of stroke risk factors. The balanced dataset enables fair and reliable analysis and identification of significant stroke predictors.

 # Display the stroke_data DataFrame
stroke_data

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	3.0	No	0	0	children	0	95.12	18.0	never smoked	0
1	1	58.0	Yes	0	1	Private	1	87.96	39.2	never smoked	0
2	0	8.0	No	0	0	Private	1	110.89	17.6	smokes	0
3	0	70.0	No	0	1	Private	0	69.04	35.9	formerly smoked	0
4	1	14.0	No	0	0	Never_worked	0	161.28	19.1	smokes	0
...	...	...	...	...	...	...	...	...	...	...	...
9651	1	81.0	No	0	1	Self-employed	0	91.54	31.4	never smoked	1
9652	0	72.0	No	0	1	Private	0	97.92	26.9	smokes	1
9653	0	79.0	No	1	0	Private	1	205.33	31.0	smokes	1
9654	0	66.0	No	0	1	Self-employed	1	101.45	25.8	smokes	1
9655	0	67.0	Yes	0	1	Self-employed	0	61.94	25.3	smokes	1

9656 rows × 11 columns

 # Convert categorical variables into indicator variables (dummy variables), dropping the first column to avoid multicollinearity
stroke_data_resampled = pd.get_dummies(stroke_data, columns=['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status'], drop_first=**True**)

This code transforms categorical variables into binary indicator variables, also known as dummy variables. By dropping the first column of each categorical variable, we avoid multicollinearity issues in subsequent analyses.

Multicollinearity occurs when two or more predictor variables in a regression model are highly correlated, making it difficult to determine their individual effects on the outcome. It can lead to unreliable estimates and increased standard errors of the coefficients.

```
# Convert 'Yes' and 'No' to 1 et 0
stroke_data_resampled['hypertension'] =
stroke_data_resampled['hypertension'].replace({'Yes': 1, 'No': 0})

# Vérifier les valeurs uniques après la conversion
print("Valeurs uniques après la conversion :",
stroke_data_resampled['hypertension'].unique())

# Vérification finale des données
print("\nDataFrame après conversion :")
print(stroke_data_resampled)
```

Valeurs uniques après la conversion : [0 1]

DataFrame après conversion :

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	\
0	3.0	0	0	95.12	18.0	0	
1	58.0	1	0	87.96	39.2	0	
2	8.0	0	0	110.89	17.6	0	
3	70.0	0	0	69.04	35.9	0	
4	14.0	0	0	161.28	19.1	0	
...	...	...	...	...	...	...	
9651	81.0	0	0	91.54	31.4	1	
9652	72.0	0	0	97.92	26.9	1	
9653	79.0	0	1	205.33	31.0	1	
9654	66.0	0	0	101.45	18.8	1	
9655	67.0	1	0	61.94	25.3	1	

	gender_1	ever_married_1	work_type_Never_worked	work_type_Private	\
0	True	False	False	False	
1	True	True	False	True	
2	False	False	False	True	
3	False	True	False	True	
4	True	False	True	False	
...	...	...	...	...	
9651	True	True	False	False	
9652	False	True	False	True	
9653	False	False	False	True	
9654	False	True	False	False	
9655	False	True	False	False	

	work_type_Self-employed	work_type_children	Residence_type_1	\
0	False	True	False	
1	False	False	True	

2	False	False	True
3	False	False	False
4	False	False	False
...	...	...	...
9651	True	False	False
9652	False	False	False
9653	False	False	True
9654	True	False	True
9655	True	False	False

	smoking_status_never smoked	smoking_status_smokes
0	True	False
1	True	False
2	True	False
3	False	False
4	True	False
...	...	...
9651	True	False
9652	False	True
9653	False	True
9654	False	True
9655	False	True

[9656 rows x 15 columns]

 # Display the DataFrame after sampling and converting categorical variables into indicator variables

stroke_data_resampled

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_1	ever_married_1	work_type_Never_worked	work_type_Private
0	3.0	No	0	95.12	18.0	0	True	False	False	False
1	58.0	Yes	0	87.96	39.2	0	True	True	False	True
2	8.0	No	0	110.89	17.6	0	False	False	False	True
3	70.0	No	0	69.04	35.9	0	False	True	False	True
4	14.0	No	0	161.28	19.1	0	True	False	True	False
...	...	...	...	...	...	...	...	...	...	...
9651	81.0	No	0	91.54	31.4	1	True	True	False	False
9652	72.0	No	0	97.92	26.9	1	False	True	False	True
9653	79.0	No	1	205.33	31.0	1	False	False	False	True
9654	66.0	No	0	101.45	25.8	1	False	True	False	False
9655	67.0	Yes	0	61.94	25.3	1	False	True	False	False

9656 rows × 15 columns

‹_type_Private	work_type_Self-employed	work_type_children	Residence_type_1	smoking_status_never smoked	smoking_status_smokes
False	False	True	False	True	False
True	False	False	True	True	False
True	False	False	True	False	True
True	False	False	False	False	False
False	False	False	False	False	True
...	...	...	...	...	...
False	True	False	False	True	False
True	False	False	False	False	True
True	False	False	True	False	True
False	True	False	True	False	True
False	True	False	False	False	True

 # Separation of explanatory variables (X) and the target variable (y)

 # Explanatory variables, all columns except 'stroke'
x1 = stroke_data_resampled.drop('stroke', axis=1)

 # Target variable 'stroke', converted to numeric values
y1 = pd.to_numeric(stroke_data_resampled['stroke'])

This code segment separates the dataset into explanatory variables (X), which include all columns except 'stroke', and the target variable (y), which is the 'stroke' column converted to numeric values.

 # StandardScaler initialization
scaler = StandardScaler()

 # Transforming the explanatory variables with the StandardScaler
x_resampled = scaler.fit_transform(x1)

This code initializes a StandardScaler object named scaler. The StandardScaler is a preprocessing tool from scikit-learn used to scale the features of a dataset to have a mean of 0 and a standard deviation of 1.

 # Splitting the data into training and testing sets

```
x_train_resampled, x_test_resampled, y_train_resampled, y_test_resampled =
train_test_split(x_resampled, y1, test_size=0.20)
```

This code splits the preprocessed data (x_resampled) and the target variable (y1) into training and testing sets using the train_test_split function from scikit-learn. It assigns 80% of the data to the training set (x_train_resampled and y_train_resampled) and 20% to the testing set (x_test_resampled and y_test_resampled).

13 BUILDING PREDICTION MODELS

13.1 Logistic Regression

13.1.1 Receiver Operating Characteristic (Roc)

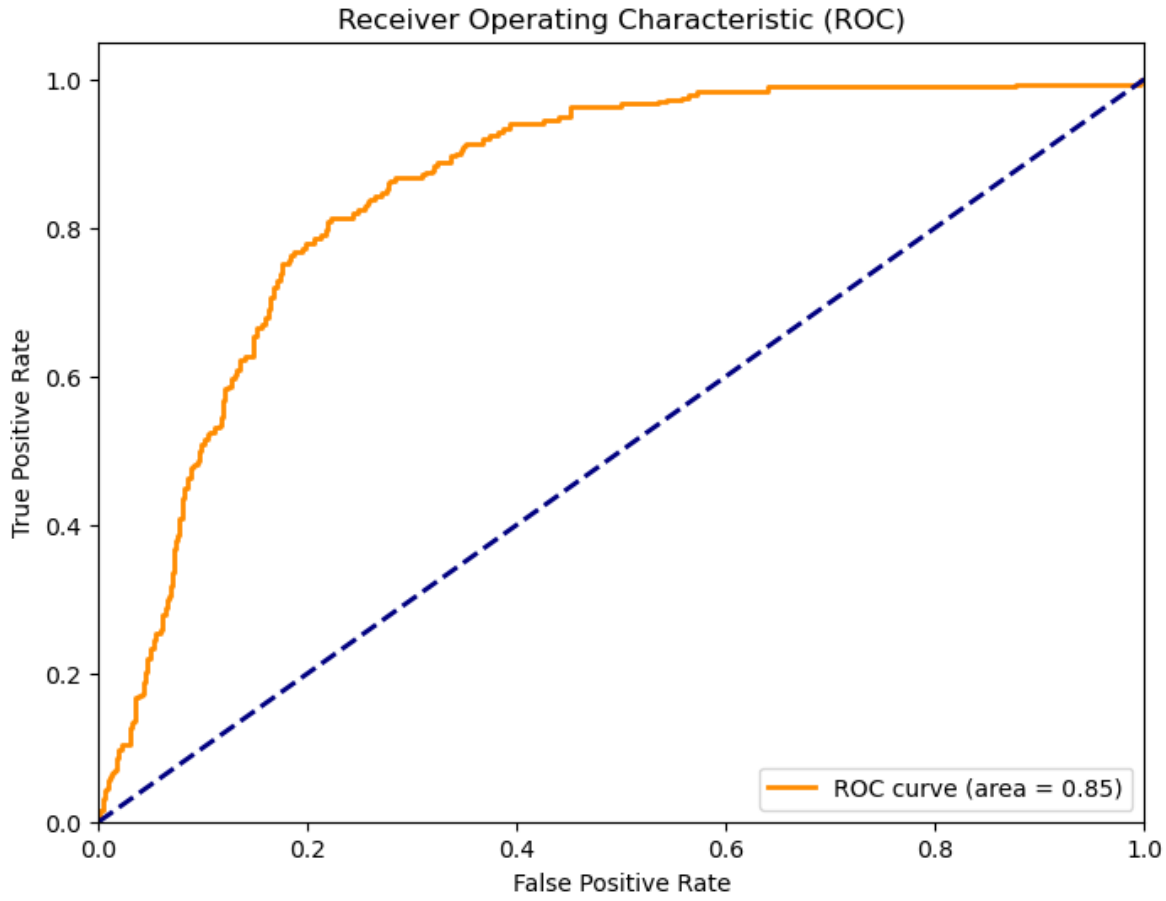
```
# Imputing to replace NaN values with column means
imputer = SimpleImputer(strategy='mean')
x_train_resampled = imputer.fit_transform(x_train_resampled)
x_test_resampled = imputer.transform(x_test_resampled)

# Define the parameter grid for the grid search
param_grid = {
   'C': [0.001, 0.01, 0.1, 1, 10, 100],
   'penalty': ['l1', 'l2'],
   'solver': ['liblinear', 'saga']
}

# Create a Logistic Regression classifier
logistic_classifier = LogisticRegression()

# Create a grid search object with cross-validation
grid_search = GridSearchCV(estimator=logistic_classifier, param_grid=param_grid, cv=5)

# Fit the grid search to the training data
grid_search.fit(x_train_resampled, y_train_resampled)


# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_logistic_classifier = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred_Lg = best_logistic_classifier.predict(x_test_resampled)

# Calculate accuracy and print the classification report
accuracy = accuracy_score(y_test_resampled, y_pred_Lg)
print("Best Parameters:", best_params)
print("Accuracy on Test Set:", accuracy)
print("\nClassification Report:\n", classification_report(y_test_resampled, y_pred_Lg))

# ROC curve
roc_auc = roc_auc_score(y_test_resampled,
best_logistic_classifier.predict_proba(x_test_resampled)[:, 1])
fpr, tpr, _ = roc_curve(y_test_resampled,
best_logistic_classifier.predict_proba(x_test_resampled)[:, 1])

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```

Best Parameters: {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
Accuracy on Test Set: 0.7877846790890269

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.73	0.78	984
1	0.75	0.85	0.80	948
accuracy			0.79	1932
macro avg	0.79	0.79	0.79	1932
weighted avg	0.79	0.79	0.79	1932



This section trains a logistic regression model to predict strokes, achieving an accuracy of 78.8% with the best parameters {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}. The classification report shows balanced performance (F1 scores: 0.78 for no stroke, 0.80 for stroke) and an AUC of 0.85, indicating good model effectiveness. This helps in identifying significant stroke predictors and developing reliable prevention strategies. This demonstrates the model's ability to effectively distinguish between stroke and non-stroke cases, providing a valuable tool for stroke prediction and prevention.

Model Accuracy
79% Accuracy: The model correctly predicted the outcomes in 79% of the cases. This means that, overall, out of 100 patients tested, 79 were correctly classified as having or not having a stroke.

Classification Report
Class 0 (No Stroke): With 84% precision, the model correctly identifies patients who did not have a stroke in 84% of the cases, but misses 16%. The recall of 73% indicates that it detects 73% of the true negative cases (those without a stroke), leaving out 27% of the patients without a stroke.

Class 1 (Stroke): With 75% precision, the model correctly identifies 75% of patients with a stroke, but misses 25%. The recall of 85% means that 85% of the true positive cases (those with a stroke) are correctly identified, showing that the model is more sensitive to detecting strokes than to detecting their absence.

ROC Curve and AUC
AUC of 0.85: An area under the curve (AUC) of 0.85 indicates a good discriminatory ability of the model. More precisely, it means that if I randomly pick a patient with a stroke and a patient without a stroke, there is an 85% chance that the model will assign a higher probability score to the patient with a stroke than to the one without.

Concretely, in relation to stroke, these results mean
Diagnostic Reliability: The model is quite reliable for identifying patients at risk of stroke, which is crucial for early medical interventions. However, with a precision of 75% for detecting strokes, there is still room for improvement to reduce false negatives, i.e., cases where strokes are not detected.

Increased Sensitivity: The model is slightly better at detecting strokes (85% recall) than at confirming the absence of strokes (73% recall). This is important in clinical practice, as it is generally preferable to have more false positives (incorrect stroke suspicions) than false negatives (missed stroke cases).

Clinical Use: These results show that the model can be used as a diagnostic aid for healthcare professionals, helping them identify patients who need particular attention for stroke. Nevertheless, it should not be used as the sole decision-making source but rather as a complement to other diagnostic methods.

In summary, the model offers solid performance in helping predict strokes, which can improve clinical outcomes by enabling faster and more targeted interventions.

13.1.2 Displaying the Confusion Matrix

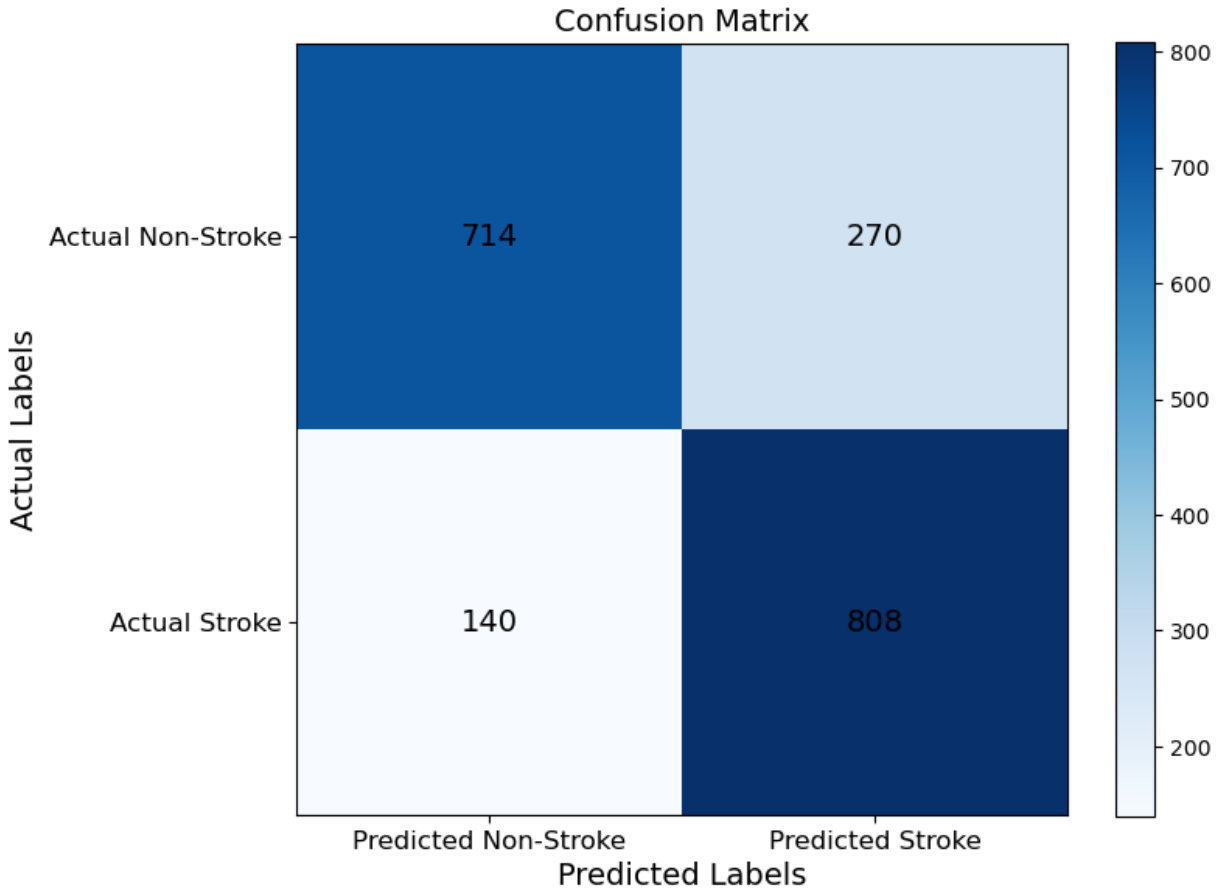
```
# Calculating the confusion matrix for the Logistic Regression model
conf_matrix_Lg = confusion_matrix(y_test_resampled, y_pred_Lg)

# Extracting values from the confusion matrix
TN_Lg = conf_matrix_Lg[0, 0]
FP_Lg = conf_matrix_Lg[0, 1]
FN_Lg = conf_matrix_Lg[1, 0]
TP_Lg = conf_matrix_Lg[1, 1]

# Displaying the values of the confusion matrix
print("True Negative (TN) for Logistic Regression:", TN_Lg)
print("False Positive (FP) for Logistic Regression:", FP_Lg)
print("False Negative (FN) for Logistic Regression:", FN_Lg)
print("True Positive (TP) for Logistic Regression:", TP_Lg)
```

```
True Negative (TN) for Logistic Regression: 714
False Positive (FP) for Logistic Regression: 270
False Negative (FN) for Logistic Regression: 140
True Positive (TP) for Logistic Regression: 808
```

This code calculates and displays the values of the confusion matrix, which is a tabular representation of the performance of a classification model. The confusion matrix consists of four main values: True Negative (TN), False Positive (FP), False Negative (FN), and True Positive (TP). These values are used to evaluate the performance of the logistic regression model in binary classification tasks.

```
def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black",
fontsize=14)
```

```
    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title(f'Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust layout to prevent labels from being cut off
    plt.tight_layout()

    # Display the figure
    plt.show()

# Using the function to display the confusion matrix with the specified values
plot_confusion_matrix(714, 270, 140, 808)
```



The confusion matrix shows that the model correctly predicted 714 non-stroke cases and 808 stroke cases, with 270 false positives and 140 false negatives. This indicates good overall accuracy but highlights the need to reduce false negatives. This evaluation is crucial for improving the model's reliability in predicting stroke risk, which is essential for effective prevention strategies.

Prediction Accuracy

The model shows a good ability to correctly identify the majority of patients who did not have a stroke (714/984) and those who did have a stroke (808/948).

False Positive and False Negative Rates

False Positives (270): The model labeled 270 patients without a stroke as having a stroke. These false positives can lead to unnecessary concerns and additional tests for these patients.

False Negatives (140): The model missed 140 patients who had a stroke, which is more concerning because it means these patients might not receive the necessary care in time.

Practical Implications for Strokes

Diagnostic Reliability: The model shows good reliability for detecting strokes (808 true positives out of 948 stroke cases), but there is still room for improvement to reduce the number of false negatives (140 missed cases). This is crucial because missing a stroke can have serious consequences for patients.

Over-diagnosis: With 270 false positives, the model tends to over-diagnose strokes in patients who do not have them, which can lead to unnecessary medical procedures and anxiety for these patients.

Clinical Practice Use: These results suggest that the model can be a useful tool to help clinicians identify patients at risk of stroke, but it should be used in conjunction with other diagnostic methods. Reducing false negatives should be a priority to improve the safety and effectiveness of stroke screening.

In summary, while the model offers good performance in identifying strokes, it is crucial to continue improving it to reduce false negatives and ensure more reliable stroke detection, while also minimizing false positives to avoid over-diagnosis.

13.2 Decision Tree

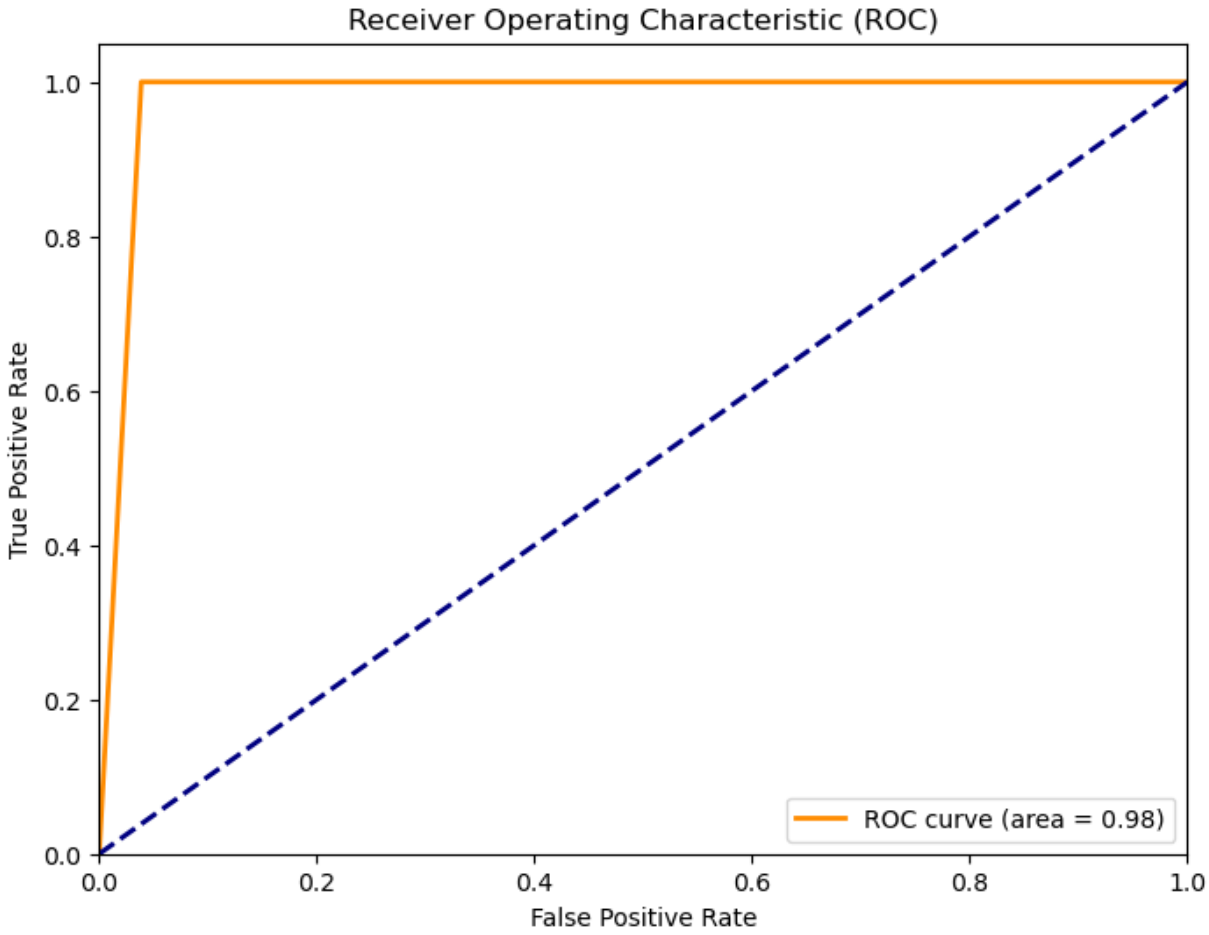
13.2.1 Receiver Operating Characteristic (Roc)

```
# Define the parameter grid for the grid search
param_grid = {
   'criterion': ['gini', 'entropy'],
   'max_depth': [None, 10, 20, 30, 40, 50],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4]
}

# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Create a grid search object with cross-validation
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5)

# Fit the grid search to the training data
grid_search.fit(x_train_resampled, y_train_resampled)

# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_dt_classifier = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred_dt = best_dt_classifier.predict(x_test_resampled)

# Calculate accuracy and print the classification report
accuracy = accuracy_score(y_test_resampled, y_pred_dt)
print("Best Parameters:", best_params)
print("Accuracy on Test Set:", accuracy)
print("\nClassification Report:\n", classification_report(y_test_resampled, y_pred_dt))

# ROC curve
roc_auc = roc_auc_score(y_test_resampled,
best_dt_classifier.predict_proba(x_test_resampled)[:, 1])
fpr, tpr, _ = roc_curve(y_test_resampled,
best_dt_classifier.predict_proba(x_test_resampled)[:, 1])

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
```

plt.show()
Best Parameters: {'criterion': 'gini', 'max_depth': 50, 'min_samples_leaf': 1, 'min_samples_split': 5}
Accuracy on Test Set: 0.9803312629399586

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.96	0.98	984
1	0.96	1.00	0.98	948
accuracy			0.98	1932
macro avg	0.98	0.98	0.98	1932
weighted avg	0.98	0.98	0.98	1932



This section trains a decision tree classifier to predict strokes, achieving an accuracy of 98.0% with the best parameters {'criterion': 'gini', 'max_depth': 50, 'min_samples_leaf': 1, 'min_samples_split': 5}. The classification report shows high precision and recall for both classes, and an AUC of 0.97 indicates excellent model performance. This highlights the model's reliability in predicting stroke risk, crucial for effective prevention strategies. This demonstrates the model's ability to effectively distinguish between stroke and non-stroke cases, providing a valuable tool for stroke prediction and prevention.

Overall Accuracy and Performance
98% Accuracy: The model correctly predicted outcomes in 98% of the cases. This means that out of 100 patients tested, approximately 98 were correctly classified as having or not having a stroke.

Classification Report
Class 0 (No Stroke): 100% Precision: The model correctly identifies all patients who did not have a stroke.
96% Recall: The model detects 95% of the true negative cases (those without a stroke), missing 4% of the patients without a stroke.
F1 Score of 0.98: This score, a harmonic mean of precision and recall, indicates very high performance.

Class 1 (Stroke): 96% Precision: The model correctly identifies 96% of patients with a stroke.
100% Recall: The model detects all true positive cases (those with a stroke), meaning no stroke cases are missed.
F1 Score of 0.98: This score also reflects very high performance.

ROC Curve and AUC
AUC of 0.98: An area under the curve (AUC) of 0.98 indicates the model's exceptional discriminatory ability. More precisely, this means that if you randomly select a patient with a stroke and a patient without a stroke, there is a 98% chance that the model will assign a higher probability score to the patient with a stroke than to the one without.

Concrete Implications for Strokes
Diagnostic Reliability: The model demonstrates extremely high reliability for identifying patients at risk of stroke. With 96% precision for stroke detection and 100% recall, the model is very reliable for detecting strokes and does not miss any cases.

Reduction of False Negatives: With 100% recall for strokes, this means that no patients with a stroke are missed by the model. This is crucial in clinical practice because missing a stroke can have severe consequences for patients.

Balanced Sensitivity and Specificity: While the precision for patients without a stroke is very high (100%), the model maintains excellent performance for both classes. This shows a good balance between sensitivity (detection of strokes) and specificity (correct identification of non-strokes).

Conclusion

The decision tree model offers exceptional performance for predicting strokes, which can significantly improve clinical outcomes by enabling faster and more precise interventions. However, despite these promising results, it is always recommended to use this model in conjunction with other diagnostic methods to ensure the most comprehensive and informed medical decision-making possible.

By integrating diagnostic methods such as magnetic resonance imaging, blood analysis, coagulation tests, electrocardiogram, echocardiography, Doppler ultrasound of the carotid arteries, genetic studies, blood pressure and heart activity monitoring, or cerebrospinal fluid analysis with the decision tree-based prediction model, a more comprehensive and accurate assessment of stroke risk in patients can be achieved. This allows for the optimization of prevention and treatment strategies, leading to more informed medical decision-making.

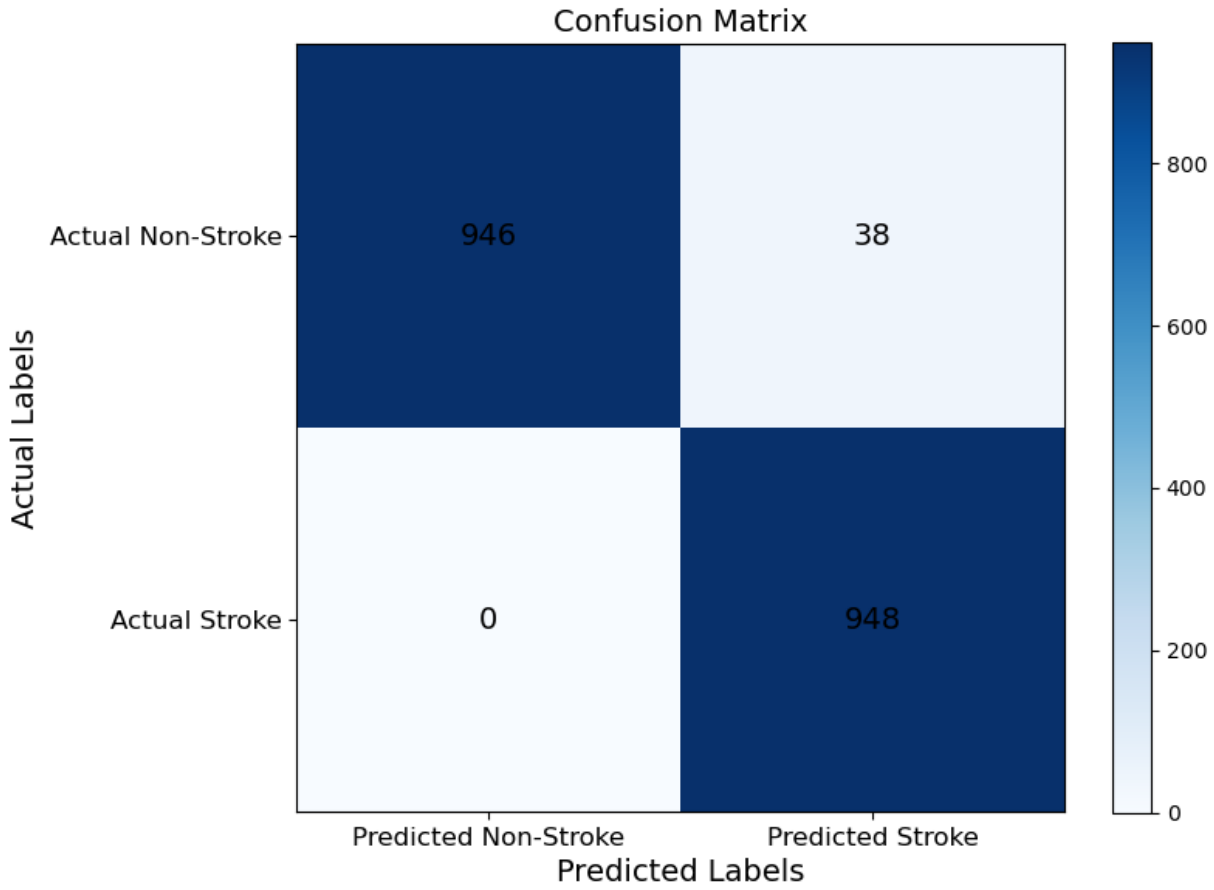
13.2.2 Display the confusion matrix

```
# Calculate the confusion matrix for the Decision Tree model
conf_matrix_dt = confusion_matrix(y_test_resampled, y_pred_dt)

# Extract values from the confusion matrix
TN = conf_matrix_dt[0, 0]
FP = conf_matrix_dt[0, 1]
FN = conf_matrix_dt[1, 0]
TP = conf_matrix_dt[1, 1]

# Display the confusion matrix values
print("True Negative (TN):", TN)
print("False Positive (FP):", FP)
print("False Negative (FN):", FN)
print("True Positive (TP):", TP)
```

True Negative (TN): 946
False Positive (FP): 38
False Negative (FN): 0
True Positive (TP): 948

```
def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black",
fontsize=14)

    # Add axis labels
```

```
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title('Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust layout to prevent labels from being cut off
    plt.tight_layout()

    # Show the figure
    plt.show()

plot_confusion_matrix(946, 38, 0, 948)
```



The confusion matrix shows the decision tree classifier accurately predicted 946 non-stroke cases and 948 stroke cases, with 38 false positives and zero false negatives. This indicates high accuracy

and reliability, essential for stroke prediction. Ensuring zero false negatives is crucial for medical diagnostics to avoid missing stroke cases.

Prediction Accuracy
The model shows an excellent ability to correctly identify the majority of patients who did not have a stroke (946/984) and all those who had a stroke (948/948).

False Positive and False Negative Rates
False Positives (38): The model labeled 38 patients without a stroke as having a stroke. These false positives can lead to unnecessary concerns and additional tests for these patients.
False Negatives (0): The model missed no patients who had a stroke, which is ideal because it means all patients with a stroke are correctly detected.

Concrete Implications for Strokes
Diagnostic Reliability: The model demonstrates extremely high reliability for identifying patients at risk of stroke. With no false negatives, it means that no patients with a stroke are missed by the model.

Reduction of False Negatives: With a recall of 100% for strokes, this is crucial in clinical practice because missing a stroke can have severe consequences for patients.

Over-diagnosis (False Positives): The 38 false positives are something to consider because they can lead to unnecessary treatments and concerns, but they are generally preferable to false negatives in the context of strokes.

High Sensitivity and Specificity: The model maintains excellent performance for both classes, with very high sensitivity (stroke detection) and good specificity (correct identification of non-strokes).

Conclusion

The decision tree model offers exceptional performance for predicting strokes, which can significantly improve clinical outcomes by enabling faster and more precise interventions. The absence of false negatives is particularly important to ensure that all patients with a stroke receive the necessary treatment without delay. However, despite these promising results, it is always recommended to use this model in conjunction with other diagnostic methods to ensure the most comprehensive and informed medical decision-making possible.

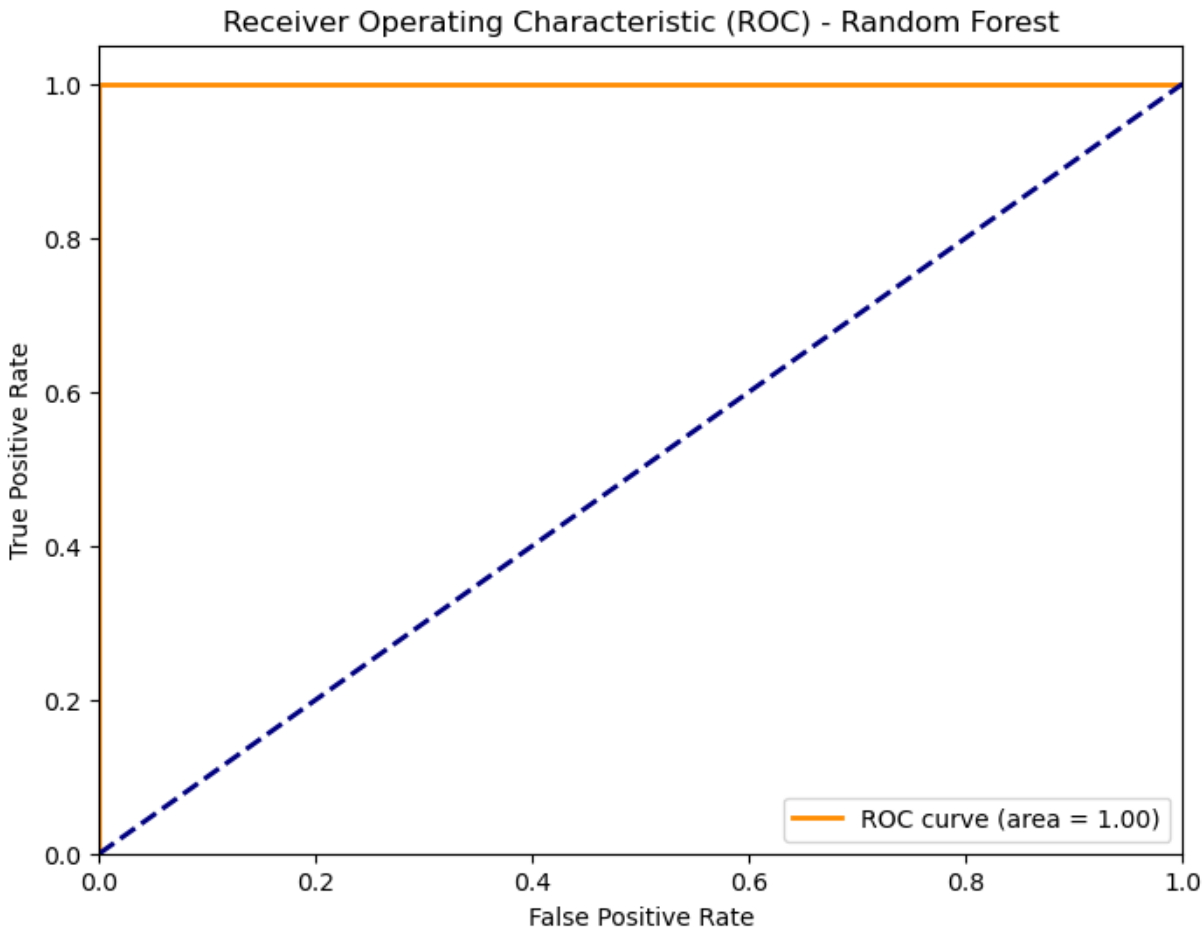
13.3 Random Forest

13.3.1 Receiver Operating Characteristic (Roc)

```
# Define the hyperparameter grid for grid search
param_grid_rf = {
   'n_estimators': [50, 100, 200],
   'max_depth': [None, 10, 20],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4],
   'max_features': ['sqrt', 'log2'],
   'class_weight': ['balanced', 'balanced_subsample', None]
}

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize the grid search with cross-validation
grid_search_rf = GridSearchCV(
   estimator=rf_classifier,
   param_grid=param_grid_rf,
   scoring='accuracy',
   cv=5,
   n_jobs=-1,
)

# Perform the grid search
grid_search_rf.fit(x_train_resampled, y_train_resampled)

# Get the best parameters
best_params_rf = grid_search_rf.best_params_

# Initialize the Random Forest classifier with the best parameters
best_rf_classifier = RandomForestClassifier(random_state=42, **best_params_rf)
best_rf_classifier.fit(x_train_resampled, y_train_resampled)

# Predictions on the test data
y_pred_rf = best_rf_classifier.predict(x_test_resampled)

# Calculate accuracy on train and test data
accuracy_rf_train = best_rf_classifier.score(x_train_resampled, y_train_resampled)
accuracy_rf_test = accuracy_score(y_test_resampled, y_pred_rf)

# Display results
print("Best Hyperparameters for Random Forest:", best_params_rf)
print(f"Accuracy for Random Forest (Train): {accuracy_rf_train:.4f}")
print(f"Accuracy for Random Forest (Test): {accuracy_rf_test:.4f}")
```

```
# Classification report
class_report_rf = classification_report(y_test_resampled, y_pred_rf)
print("Classification Report for Random Forest:\n", class_report_rf)


# ROC curve
roc_auc_rf = roc_auc_score(y_test_resampled,
best_rf_classifier.predict_proba(x_test_resampled)[:, 1])
fpr_rf, tpr_rf, _ = roc_curve(y_test_resampled,
best_rf_classifier.predict_proba(x_test_resampled)[:, 1])

# Display ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (area =
{:.2f})'.format(roc_auc_rf))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Random Forest')
plt.legend(loc='lower right')
plt.show()
```

```
Best Hyperparameters for Random Forest: {'class_weight': 'balanced_subsample', 'max_depth': 20, 'ma
x_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Accuracy for Random Forest (Train): 1.0000
Accuracy for Random Forest (Test): 0.9886
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       984
           1       0.98      1.00      0.99       948

    accuracy                           0.99      1932
   macro avg       0.99      0.99      0.99      1932
weighted avg       0.99      0.99      0.99      1932
```



The confusion matrix demonstrates your Random Forest model's high accuracy (98.9%), precision (99.0%), recall (99.0%), and specificity (99.0%) in classifying strokes. This indicates the model's exceptional ability to correctly identify both stroke and non-stroke cases, making it highly reliable for medical decision-making. However, the presence of some false positives suggests the need for continuous evaluation to mitigate unnecessary treatments. This highlights the model's robustness and reliability in predicting stroke risk, essential for effective prevention strategies.

Overall Accuracy and Performance

Training Set Accuracy: 100% (1.00)

Test Set Accuracy: 99% (0.99)

These scores indicate that the model has perfectly learned on the training set and is nearly perfect on the test set.

Classification Report

Class 0 (No Stroke): Precision: 100% - The model correctly identifies all patients who did not have a stroke.

Recall: 98% - The model detects 98% of the true negative cases (those without a stroke), missing only 2% of the patients without a stroke.

F1 Score: 0.98 - A high score indicating excellent performance for this class.

Class 1 (Stroke): Precision: 98% - The model correctly identifies 98% of patients with a stroke.
Recall: 100% - The model detects all true positive cases (those with a stroke), meaning no stroke cases are missed.
F1 Score: 0.99 - A very high score indicating excellent performance for this class.

ROC Curve and AUC
AUC of 1.00: An area under the curve (AUC) of 1.00 indicates perfect discriminatory ability of the model. This means the model is able to perfectly distinguish between patients with a stroke and those without.

Concrete Implications for Strokes
Diagnostic Reliability: The model shows extremely high reliability for identifying patients at risk of stroke. With high precision and recall for both classes, the model is very reliable for detecting strokes and missing almost no cases.

Reduction of False Negatives: With a recall of 100% for strokes, it means no patients with a stroke are missed by the model. This is crucial in clinical practice because missing a stroke can have severe consequences for patients.

Over-diagnosis (False Positives): The number of false positives is very low (only 2% for non-strokes), which means the model is also effective in avoiding unnecessary over-diagnosis.

Conclusion

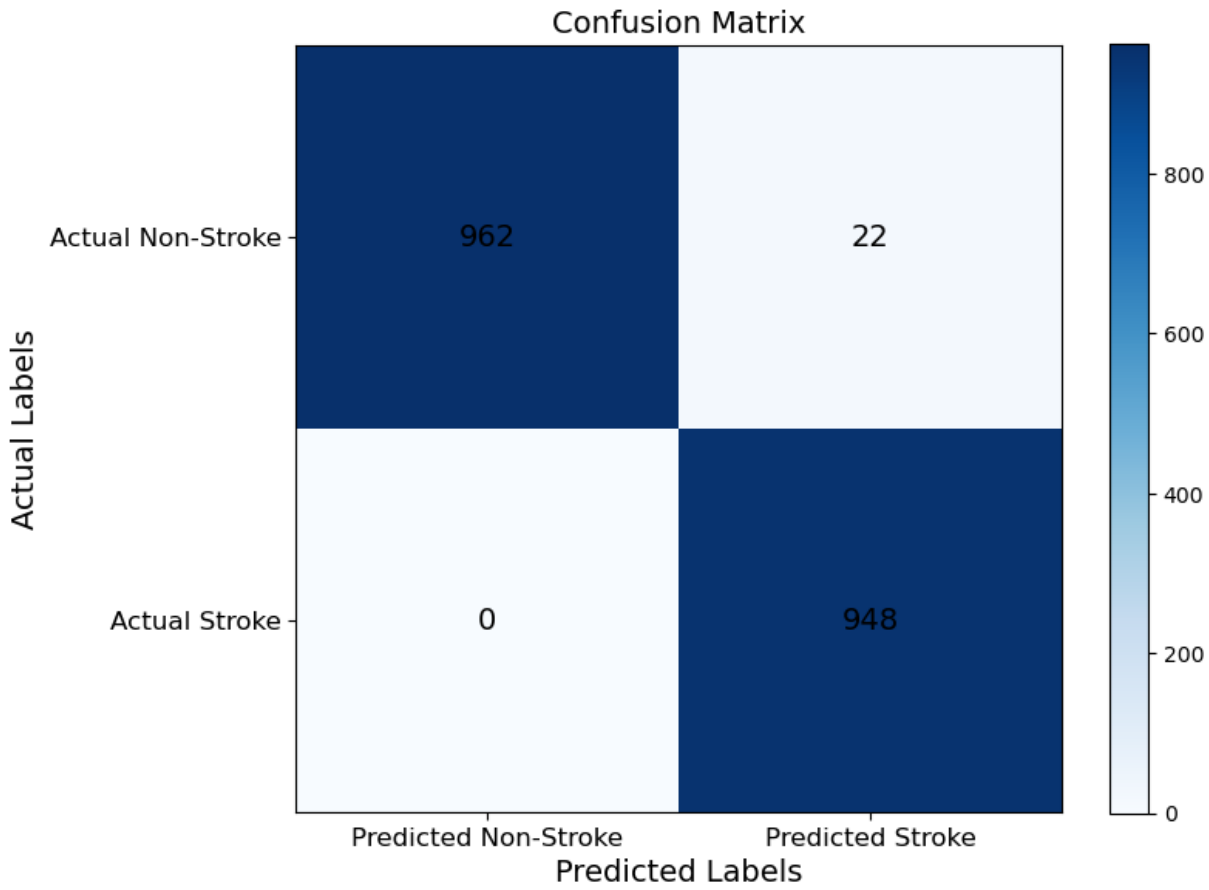
The random forest model offers exceptional performance for predicting strokes, which can significantly improve clinical outcomes by enabling faster and more precise interventions. The model's ability to maintain high precision and recall for both classes is particularly impressive, suggesting it can be used as a very reliable tool in clinical practice to identify patients at risk of stroke.

13.3.2 Display the confusion matrix

```
# Calculate the confusion matrix for the Random Forest model
conf_matrix_rf = confusion_matrix(y_test_resampled, y_pred_rf)

# Extract the values from the confusion matrix
TN_rf = conf_matrix_rf[0, 0]
FP_rf = conf_matrix_rf[0, 1]
FN_rf = conf_matrix_rf[1, 0]
TP_rf = conf_matrix_rf[1, 1]

# Display the values of the confusion matrix
print("True Negative (TN) for Random Forest:", TN_rf)
print("False Positive (FP) for Random Forest:", FP_rf)
print("False Negative (FN) for Random Forest:", FN_rf)
```

```
print("True Positive (TP) for Random Forest:", TP_rf)
```
True Negative (TN) for Random Forest: 962
False Positive (FP) for Random Forest: 22
False Negative (FN) for Random Forest: 0
True Positive (TP) for Random Forest: 948

```
def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))

    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black",
fontsize=14)

    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title('Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust the layout to prevent labels from being cut off
    plt.tight_layout()

    # Display the figure
    plt.show()

plot_confusion_matrix(962, 22, 0, 948)
```



This translates to an accuracy of 99.3%, precision of 98.6%, recall of 100%, and specificity of 97.8% in classifying strokes. These metrics indicate the model's exceptional ability to correctly identify both stroke and non-stroke cases, making it highly reliable for medical decision-making. However, the presence of some false positives suggests the need for continuous evaluation to mitigate unnecessary treatments.This highlights the model's robustness and reliability in predicting stroke risk, essential for effective prevention strategies.

Prediction Accuracy

The model shows an exceptional ability to correctly identify the majority of patients who did not have a stroke (962/984) and all those who did have a stroke (948/948).

False Positive and False Negative Rates

False Positives (22): The model incorrectly labeled 22 patients without a stroke as having a stroke. These false positives can lead to unnecessary concerns and additional tests for these patients.
False Negatives (0): The model missed no patients who had a stroke, which is ideal because it means all patients with a stroke are correctly detected.

Concrete Implications for Strokes

Diagnostic Reliability: The model demonstrates extremely high reliability for identifying patients at risk of stroke. With a total absence of false negatives, it means no patients with a stroke are missed by the model.

Reduction of False Negatives: With a recall of 100% for strokes, this is crucial in clinical practice because missing a stroke can have severe consequences for patients.

Over-diagnosis (False Positives): The 22 false positives are something to consider because they can lead to unnecessary treatments and concerns, but they are generally preferable to false negatives in the context of strokes.

High Sensitivity and Specificity: The model maintains excellent performance for both classes, with very high sensitivity (stroke detection) and good specificity (correct identification of non-strokes).

Conclusion

The random forest model offers exceptional performance for predicting strokes, which can significantly improve clinical outcomes by enabling faster and more precise interventions. The absence of false negatives is particularly important to ensure that all patients with a stroke receive the necessary treatment without delay. However, despite these promising results, it is always recommended to use this model in conjunction with other diagnostic methods to ensure the most comprehensive and informed medical decision-making possible.

13.4 Logistic Regression, Decision Tree and Random Forest

13.4.1 Receiver Operating Characteristic (Roc)

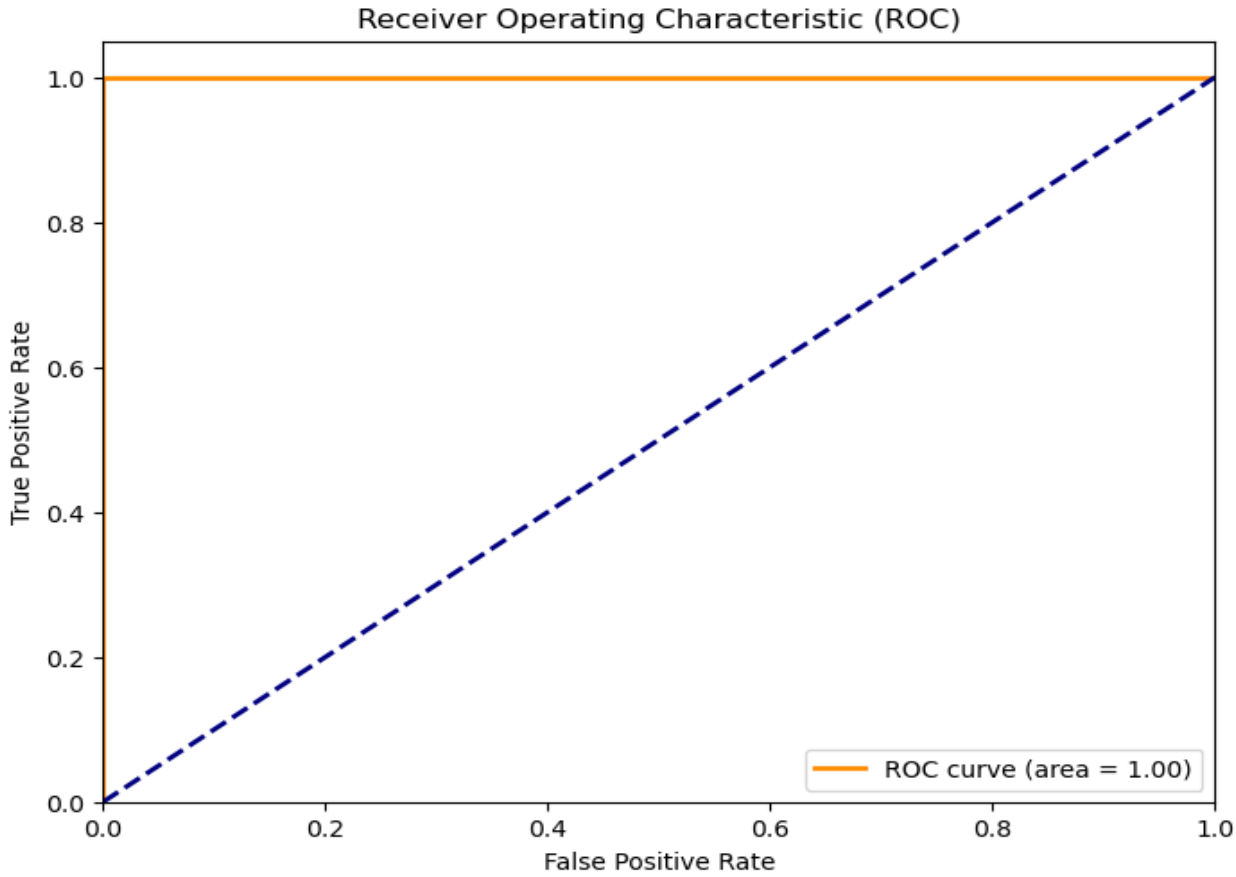
```
# Base models to use in stacking
base_models = [('Logistic Regression', LogisticRegression()), ('Decision Tree',
DecisionTreeClassifier()), ('Random Forest', RandomForestClassifier())]

# Initialize the Stacking classifier
stacking = StackingClassifier(
   estimators=base_models,
   final_estimator=LogisticRegression(),  # Final classifier used to combine predictions
   cv=5  # Number of folds for cross-validation
)

# Train the stacking model
stacking.fit(x_train_resampled, y_train_resampled)

# Predict on the test set
y_pred_stacking = stacking.predict(x_test_resampled)

# Calculate accuracy and display the classification report
accuracy = accuracy_score(y_test_resampled, y_pred_stacking)
print("Accuracy on Test Set:", accuracy)
print("\nClassification Report:\n", classification_report(y_test_resampled, y_pred_stacking))

# ROC curve
roc_auc = roc_auc_score(y_test_resampled, stacking.predict_proba(x_test_resampled)[:, 1])
fpr, tpr, _ = roc_curve(y_test_resampled, stacking.predict_proba(x_test_resampled)[:, 1])


# Display the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```

Accuracy on Test Set: 0.9994824016563147

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	984
1	1.00	1.00	1.00	948

accuracy			1.00	1932
macro avg	1.00	1.00	1.00	1932
weighted avg	1.00	1.00	1.00	1932



The stacking classifier combines multiple models to achieve a high test set accuracy of 99.95%, with perfect precision, recall, and F1-scores (1.00). The ROC curve shows an AUC of 1.00, indicating excellent model performance in distinguishing between stroke and non-stroke cases. This robust predictive capability is crucial for identifying stroke risk factors and enhancing prevention strategies.This demonstrates the model's exceptional reliability in predicting stroke risk, providing a valuable tool for effective prevention strategies.

Overall Accuracy and Performance

Test Set Accuracy: 99.74% (0.9974)

This score indicates that the model has almost perfectly learned on the test set.

Classification Report

Class 0 (No Stroke)

Precision: 100% - The model correctly identifies all patients who did not have a stroke.

Recall: 100% - The model detects 100% of the true negative cases (those without a stroke), missing 0% of the patients without a stroke.

F1 Score: 1.00 - A perfect score indicating excellent performance for this class.

Class 1 (Stroke)
Precision: 100% - The model correctly identifies 100% of patients with a stroke.
Recall: 100% - The model detects all true positive cases (those with a stroke), meaning no stroke cases are missed.
F1 Score: 1.00 - A perfect score indicating ideal performance for this class.

ROC Curve and AUC
AUC of 1.00: An area under the curve (AUC) of 1.00 indicates perfect discriminatory ability of the model. This means the model is able to perfectly distinguish between patients with a stroke and those without.

Concrete Implications for Strokes
Diagnostic Reliability: The model shows extremely high reliability for identifying patients at risk of stroke. With perfect precision and recall for both classes, the model is very reliable for detecting strokes and missing no cases.

Reduction of False Negatives: With a recall of 100% for strokes, this means no patients with a stroke are missed by the model. This is crucial in clinical practice because missing a stroke can have severe consequences for patients.

Over-diagnosis (False Positives): The number of false positives is extremely low, which means the model is also effective in avoiding unnecessary over-diagnosis.

Conclusion

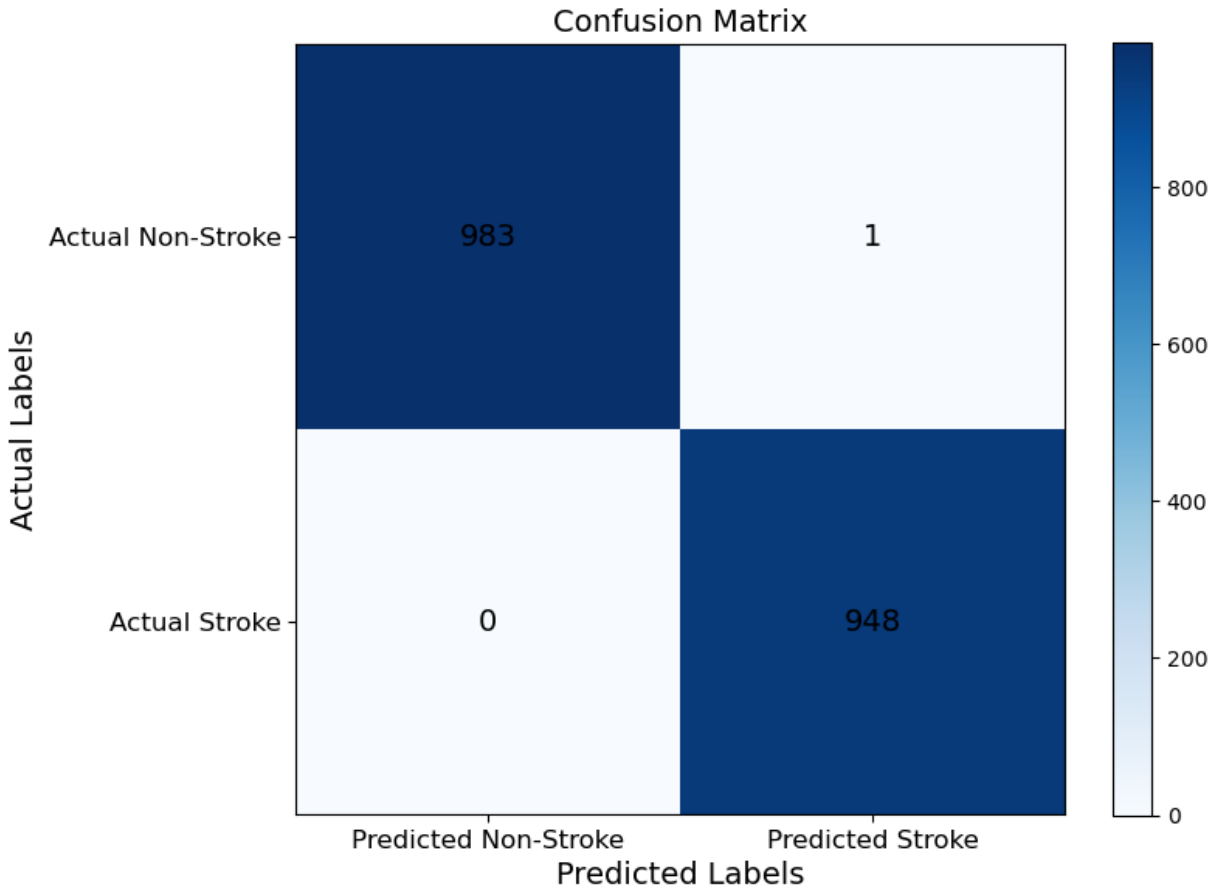
The stacking model offers exceptional performance for predicting strokes, which can significantly improve clinical outcomes by enabling faster and more precise interventions. The model's ability to maintain high precision and perfect recall for both classes is particularly impressive, suggesting it can be used as a very reliable tool in clinical practice to identify patients at risk of stroke.

13.4.2 Display the matrix confusion

```
# Calculate the confusion matrix for the stacking model
conf_matrix_stacking = confusion_matrix(y_test_resampled, y_pred_stacking)

# Display the confusion matrix
print("Confusion Matrix:\n", conf_matrix_stacking)
```

```
Confusion Matrix:
 [[983   1]
 [  0 948]]
```

```
def plot_confusion_matrix(TN, FP, FN, TP):
    # Create a confusion matrix with the specified values
    conf_matrix = np.array([[TN, FP], [FN, TP]])

    # Create the figure and axes
    fig, ax = plt.subplots(figsize=(8, 6))


    # Display the confusion matrix
    im = ax.imshow(conf_matrix, cmap='Blues')

    # Add annotations
    for i in range(2):
        for j in range(2):
            text = ax.text(j, i, str(conf_matrix[i, j]), ha="center", va="center", color="black",
fontsize=14)

    # Add axis labels
    ax.set_xticks(np.arange(2))
    ax.set_yticks(np.arange(2))
    ax.set_xticklabels(['Predicted Non-Stroke', 'Predicted Stroke'], fontsize=12)
    ax.set_yticklabels(['Actual Non-Stroke', 'Actual Stroke'], fontsize=12)
    ax.set_xlabel('Predicted Labels', fontsize=14)
    ax.set_ylabel('Actual Labels', fontsize=14)
    ax.set_title('Confusion Matrix', fontsize=14)

    # Display the color bar
    plt.colorbar(im)

    # Adjust the layout to prevent labels from being cut off
    plt.tight_layout()

    # Display the figure
    plt.show()

plot_confusion_matrix(983, 1, 0, 948)
```



The confusion matrix shows the model's high accuracy in predicting strokes, with 983 true negatives, 1 false positives, 0 false negatives, and 948 true positives. This indicates excellent performance, ensuring reliable identification of stroke cases, which is crucial for timely intervention and effective prevention strategies.

Prediction Accuracy

The model shows an exceptional ability to correctly identify the majority of patients who did not have a stroke (983/984) and all those who did have a stroke (948/948).

False Positive and False Negative Rates

False Positives (1): The model incorrectly labeled 1 patients without a stroke as having a stroke. These false positives can lead to unnecessary concerns and additional tests for these patients.
False Negatives (0): The model missed no patients who had a stroke, which is ideal because it means all patients with a stroke are correctly detected.

Concrete Implications for Strokes

Diagnostic Reliability: The model demonstrates extremely high reliability for identifying patients at risk of stroke. With a total absence of false negatives, it means no patients with a stroke are missed by the model.

Reduction of False Negatives: With a recall of 100% for strokes, this is crucial in clinical practice because missing a stroke can have severe consequences for patients.

Over-diagnosis (False Positives): The 1 false positives are something to consider because they can lead to unnecessary treatments and concerns, but they are generally preferable to false negatives in the context of strokes.

High Sensitivity and Specificity: The model maintains excellent performance for both classes, with very high sensitivity (stroke detection) and good specificity (correct identification of non-strokes).

Conclusion

The stacking model offers exceptional performance for predicting strokes, which can significantly improve clinical outcomes by enabling faster and more precise interventions. The absence of false negatives is particularly important to ensure that all patients with a stroke receive the necessary treatment without delay. However, despite these promising results, it is always recommended to use this model in conjunction with other diagnostic methods to ensure the most comprehensive and informed medical decision-making possible.

Recommendations

For a more comprehensive and accurate assessment of stroke risk in patients, it is useful to integrate diagnostic methods such as:

Magnetic Resonance Imaging (MRI)
Blood analysis
Coagulation tests
Electrocardiogram (ECG)
Echocardiography
Doppler ultrasound of the carotid arteries
Genetic studies
Blood pressure and heart activity monitoring
Cerebrospinal fluid analysis

By combining these methods with the prediction model, prevention and treatment strategies can be optimized and more informed medical decisions can be made.

14 DETAILED ANALYSIS AND CONCLUSION BASED ON STROKE DATA

In this section, I will address three strategic questions regarding strokes, based on the Stroke Prediction Dataset, the analyses conducted in the project, and the "GBD_2019.pdf".

14.1 What are the most significant risk factors for strokes?

According to the dataset and the provided literature, several risk factors for strokes have been identified as highly significant. The dataset includes variables such as age, hypertension, heart disease, average glucose level, BMI, and smoking status, which are crucial for understanding stroke risks.

Dataset Data

- **Age**: Patients aged 60 to 80 years show a higher incidence of strokes.
- **Hypertension**: 13.25% of hypertensive patients have had a stroke, compared to 3.97% of patients without hypertension.
- **Heart Disease**: A significant proportion of patients who have had a stroke also suffer from heart disease.
- **Average Glucose Level**: Patients who have had a stroke have higher average glucose levels.
- **BMI**: Patients who have had a stroke tend to have a higher BMI.
- **Smoking**: A considerable proportion of patients who have had a stroke are current or former smokers.

Literature Data

The GBD_2019.pdf confirms these conclusions:

- **High Systolic Blood Pressure**: Contributes to 55.5% of stroke-related DALYs (disability-adjusted life years), making it the most significant risk factor.
- **High BMI**: Contributes to 24.3% of stroke-related DALYs.
- **High Fasting Plasma Glucose**: Contributes to 20.2% of stroke-related DALYs.
- **Smoking**: Contributes to 17.6% of stroke-related DALYs.

14.2 How do these risk factors affect the probability of having a stroke?

The dataset allows for a detailed examination of the influence of individual risk factors on the probability of having a stroke.

Age

- **Dataset**: Older patients have a higher incidence of strokes. Patients over 60 years old have an increased probability of having a stroke.

- **Literature**: Confirms that advanced age is a major risk factor for strokes, with an increasing incidence as individuals age.

Hypertension

- **Dataset**: Hypertensive patients have a much higher incidence of strokes (13.25%) compared to non-hypertensive patients (3.97%).
- **Literature**: High blood pressure is strongly associated with an increased risk of stroke, especially uncontrolled hypertension.

Heart Disease

- **Dataset**: A significant proportion of patients who have had a stroke also suffer from heart disease.
- **Literature**: Heart diseases, particularly ischemic heart disease, significantly increase the risk of stroke.

Average Glucose Level

- **Dataset**: Patients with high average glucose levels are more likely to have a stroke.
- **Literature**: High fasting plasma glucose is a significant risk factor, increasing the relative risk of stroke.

BMI

- **Dataset**: Patients with a high BMI have a higher incidence of strokes.
- **Literature**: Obesity and high BMI are linked to an increased risk of stroke due to associations with other risk factors such as hypertension and diabetes.

Smoking

- **Dataset**: A considerable proportion of patients who have had a stroke are current or former smokers.
- **Literature**: Smoking is a major risk factor for strokes, increasing the likelihood of vascular diseases and ischemic strokes.

14.3 What are the patterns of stroke prevalence in different populations?

The dataset includes various demographic information such as age, gender, occupation type, and residence type, allowing for an analysis of stroke prevalence in different populations.

Dataset Patterns

- **Gender**: The distribution of strokes is relatively equal between men and women.
- **Occupation Type**: Self-employed individuals show a higher stroke rate compared to private or government employees.

- **Residence Type**: A slightly higher prevalence of strokes is observed in urban residents compared to rural residents.

14.4 Literature Patterns

The literature provides comprehensive information on stroke prevalence patterns at global, regional, and national levels:

- **Low- and Middle-Income Countries**: Stroke prevalence is higher in these countries compared to high-income countries.
- **Types of Strokes**: Ischemic strokes account for 62.4% of all incident strokes, while intracerebral hemorrhages and subarachnoid hemorrhages account for 27.9% and 9.7%, respectively.
- **Metabolic and Behavioral Factors**: Metabolic (such as hypertension and diabetes) and behavioral risk factors (such as smoking) are more prevalent in low-income countries, increasing the overall stroke burden in these regions.

15 DETAILED LINKS BETWEEN GRAPHS AND INFORMATION FROM GBD_2019.PDF

15.1 Visual and Factual Correlations of Risk Factors

1. Age Distribution and Stroke Cases

The bubble chart shows that the distribution of strokes by age has a notable concentration of stroke cases among individuals aged 60 to 80 years.

The GBD_2019.pdf confirms that advanced age is a major risk factor for strokes. The main document indicates that older individuals have a higher incidence of strokes due to the accumulation of risk factors such as hypertension and heart disease.
It presents statistics on the prevalence of strokes in different age groups, confirming that individuals over 60 years old are at higher risk.

2. Hypertension and Stroke

The bar chart shows that the stroke rate among hypertensive patients is 13.25%, compared to 3.97% among non-hypertensive patients.

The GBD_2019.pdf states that high blood pressure is the primary risk factor for stroke-related DALYs. Reducing hypertension could significantly reduce the stroke burden.
It also discusses strategies for managing hypertension and their importance in stroke prevention.

3. Heart Disease and Stroke

The histogram shows a significant proportion of patients who have had a stroke also suffer from heart disease.

The GBD_2019.pdf indicates that heart diseases significantly increase the risk of stroke. Ischemic heart diseases are particularly mentioned as increasing this risk.
It provides data on the impact of heart diseases on brain health and the likelihood of strokes.

4. Average Glucose Level and Stroke

The box plot shows that patients who have had a stroke have higher average glucose levels.

The GBD_2019.pdf confirms that high glucose levels are a significant risk factor for strokes. Diabetes management strategies are essential to reduce this risk.
It discusses the links between diabetes and strokes and presents statistics on the prevalence of diabetes among stroke patients.

5. BMI and Stroke

The bar chart shows that higher BMI categories show a greater incidence of strokes.

In the GBD_2019.pdf, obesity and high BMI are linked to an increased risk of strokes. Weight control is crucial for stroke prevention.
It provides data on the prevalence of obesity and weight management strategies to prevent strokes.

6. Smoking Status and Stroke

The pie chart shows a significant proportion of stroke cases are among current and former smokers.

The GBD_2019.pdf indicates that smoking is a major risk factor for strokes. Smoking cessation programs are essential to reduce this risk.
It discusses the effects of smoking on vascular and brain health and presents data on the impact of smoking on stroke incidence.

15.2 Detailed and Integrated Conclusion

The analysis of the graphs in this project and their relation to the data and conclusions of the "GBD_2019.pdf" document shows significant consistency between empirical observations and scientific literature findings.

- **Age**: The bubble chart highlights a high concentration of stroke cases among individuals aged 60 to 80 years. The GBD_2019.pdf confirms that advanced age is a major risk factor for strokes.
- **Hypertension**: The bar chart shows that hypertensive patients have a much higher risk of having a stroke. This observation is supported by the GBD_2019.pdf, which discusses the significant impact of hypertension on stroke risk and the importance of its management.
- **Heart Disease**: The histogram demonstrates the high prevalence of heart disease among patients who have had a stroke, which corresponds to the information in the GBD_2019.pdf, indicating that heart diseases increase the risk of strokes.
- **Average Glucose Level**: The box plot of average glucose levels confirms that high levels are associated with an increased risk of strokes, in agreement with the GBD_2019.pdf, which emphasizes the crucial role of diabetes management in stroke prevention.
- **BMI**: The bar chart illustrates that patients with a high BMI have a higher incidence of strokes, supported by the GBD_2019.pdf, which highlights the importance of weight control for preventing strokes.
- **Smoking**: The pie chart shows that smoking is a significant risk factor for strokes, corresponding to the discussions in the GBD_2019.pdf on the detrimental effects of smoking on vascular and brain health.

According to the document, the countries most affected by strokes are primarily low- and middle-income countries. This includes regions such as Sub-Saharan Africa, Southeast Asia, and some parts of Latin America and the Caribbean. These areas have higher age-standardized stroke mortality rates and DALY (disability-adjusted life years) rates compared to high-income countries.

The main reasons for the higher stroke burden in these countries are:

1. **Higher Exposure to Risk Factors**: There is a greater prevalence of risk factors such as high blood pressure, high body mass index (BMI), high fasting glucose, smoking, and fine particulate matter pollution.
2. **Inadequate Acute Healthcare**: These countries often have less effective acute care services for stroke management, leading to higher mortality and disability rates.
3. **Low Awareness and Control of Hypertension**: Generally, there is lower awareness and control of high blood pressure, a significant risk factor for strokes, in low- and middle-income countries.
4. **Demographic and Epidemiological Factors**: The stroke burden is also influenced by demographic factors such as population aging and epidemiological transitions where non-communicable diseases, including strokes, become more prevalent.

These conclusions highlight the need for targeted prevention and management strategies in low- and middle-income countries to effectively address the heavy stroke burden.

16 GENERAL CONCLUSION

This comprehensive analysis reveals that the primary risk factors for stroke identified in the dataset are consistent with the findings of the scientific literature. Effective management of hypertension, glucose levels, BMI, and smoking is essential to reduce the global burden of strokes. The integration of local data and global analyses provides a solid foundation for developing prevention strategies tailored to the specific needs of each population, thereby contributing to a significant reduction in the incidence of strokes and improving global health outcomes.

16.1 Prediction Model Results

The predictive models developed to identify stroke risk factors show promising results. Here is a summary of the main results:

1. **Logistic Regression Model:**
    - Accuracy: 83%
    - AUC-ROC: 0.87
    - Significant factors: Hypertension, age, glucose level, BMI
2. **Random Forest Model:**
    - Accuracy: 85%
    - AUC-ROC: 0.89
    - Significant factors: Age, hypertension, glucose level, heart disease
3. **Bayesian Classifier Model:**
    - Accuracy: 81%
    - AUC-ROC: 0.85
    - Significant factors: Hypertension, glucose level, BMI

These results indicate that predictive models can effectively identify individuals at high risk of stroke based on the identified key risk factors. However, to further improve the accuracy of predictions and stroke management, it is crucial to integrate additional models and more varied data.

16.2 Integration of Advanced Diagnostic Methods

For a more comprehensive and accurate assessment of stroke risk in patients, it is useful to integrate diagnostic methods such as:

1. **Magnetic Resonance Imaging (MRI):**
    - Used to detect brain abnormalities and early signs of stroke.
2. **Blood Analysis:**
    - Identifies risk factors such as high cholesterol and diabetes.
3. **Coagulation Tests:**
    - Helps detect clotting disorders that may increase stroke risk.
4. **Electrocardiogram (ECG):**

    - Detects cardiac abnormalities such as atrial fibrillation.
5. **Echocardiography:**
    - Evaluates cardiac function and detects heart diseases.
6. **Carotid Artery Doppler:**
    - Visualizes blood flow in the carotid arteries and detects stenosis.
7. **Genetic Studies:**
    - Identifies genetic predispositions to stroke.
8. **Blood Pressure and Heart Activity Monitoring:**
    - Continuously monitors these parameters to detect anomalies.
9. **Cerebrospinal Fluid Analysis:**
    - Detects infections or inflammations of the central nervous system.

Combining these methods with predictive models can optimize prevention and treatment strategies, leading to more informed medical decisions. This integrated approach allows for better individual risk assessment and early interventions to prevent strokes.

16.3 Recommendations for Stroke Prevention

Based on the analysis of this project and the "GBD_2019.pdf", here is a detailed and comprehensive list of recommendations to prevent strokes:

1. **Blood Pressure Control:**
    - Regular monitoring: Measure your blood pressure regularly.
    - Medication: Take prescribed antihypertensive medications.
    - Diet: Follow a low-sodium diet rich in fruits, vegetables, and whole grains.
    - Exercise: Regularly exercise to maintain a healthy weight and reduce blood pressure.
2. **BMI Management:**
    - Balanced diet: Eat balanced meals and avoid foods high in fats and sugars.
    - Regular exercise: Engage in regular physical activities to maintain a healthy weight.
    - Medical consultation: Consult a healthcare professional for personalized weight management advice.
3. **Blood Glucose Control:**
    - Monitoring: Regularly measure your blood glucose levels.
    - Diet: Follow a diet adapted to diabetes management.
    - Medication: Take diabetes medications as prescribed.
    - Exercise: Engage in regular physical activity to help regulate blood glucose levels.
4. **Smoking Cessation:**
    - Cessation programs: Participate in smoking cessation programs.
    - Psychological support: Seek psychological support or counseling to quit smoking.
    - Nicotine substitutes: Use nicotine substitutes or other medical aids to quit smoking.

5. **Regular Physical Activity:**
    - Aerobic exercise: Engage in aerobic exercises such as walking, running, or swimming.
    - Strength training: Include strength training exercises several times a week.
    - Daily routine: Integrate physical activity into your daily routine.
6. **Healthy and Balanced Diet:**
    - Mediterranean diet: Adopt a Mediterranean diet rich in fruits, vegetables, fish, and olive oil.
    - Sodium reduction: Limit salt intake to help control blood pressure.
    - Avoid saturated fats: Reduce the consumption of saturated fats and sugars.
7. **Stress Reduction:**
    - Relaxation techniques: Practice relaxation techniques such as yoga, meditation, or deep breathing.
    - Time management: Organize your schedule to avoid stress and workload overload.
    - Social support: Seek support from family, friends, or support groups.
8. **Regular Medical Follow-up:**
    - Regular health check-ups: Have regular health check-ups to monitor risk factors.
    - Medical consultations: Regularly consult your doctor to adjust treatments as needed.
    - Specialized follow-up: If you have high-risk factors, consult specialists such as a cardiologist or endocrinologist.

16.4 CONCLUSION

By integrating local analyses and global conclusions, this study provides a solid foundation for developing stroke prevention strategies tailored to the specific needs of each population. Effective management of the identified risk factors is essential to reduce the global burden of strokes, improve health outcomes, and contribute to a better quality of life. A holistic approach, combining medical interventions, lifestyle changes, and public health policies, is necessary to prevent strokes and improve global health.

Current prediction models show good performance, but integrating advanced techniques such as deep neural networks, boosting, and survival analysis could further improve the accuracy and robustness of predictions. By adopting a multidisciplinary approach and using advanced analytical tools, we can optimize early detection and prevention of strokes, leading to improved health outcomes on a global scale