A PROJECT REPORT

On

Brain Stroke Prediction

Submitted by

Shilpi Kumari 10871021024
Ratna Kushwaha 10871021040
Moumita Goswami 10871021016
Farah Ghazala 10871021041
Ayan Jawaid 10871021048
Vaishali Kumari 10871021028

Under the Guidance of

Dr. Pintu Pal
Head Of Department

Computer Applications

Asansol Engineering College
Asansol

Affiliated to

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

May 2023

Contents

Project Objective

The objective of a brain stroke prediction project is to develop a machine learning model that can accurately predict the likelihood of a person having a stroke based on various risk factors. The model should be able to analyze a set of features such as age, gender, hypertension, heart disease, marital status, work type, average glucose level, smoking status, and stroke to make predictions. The goal is to provide early warning signs to patients so that they can take appropriate action to prevent or minimize the damage caused by a stroke.

In this project, we have a shortened 'Brain Stroke Prediction' Dataset from Kaggle.

Our objective in this project is to study the given Brain Stroke Prediction dataset.We might need to pre-process the given dataset if we need to. Then, we would train 5 modelsviz. 'KNN classifier model', 'XGBoost classifier model ', 'Decision Tree classifier model' and'Logistic Regression classifier model' ,'Rado m Forest classifier model'. After training the models, we will need to find out the accuracy of each model. After finding the accuracy of each model then we need to compare them.

Our methodology for solving the problems in the given project is described below:

- Load the required dataset.
- Study the dataset.
- Describe the dataset.
- Visualize the dataset.
- Find out if the dataset needs to be pre-processed.
    - It will be determined based on whether the dataset has null values or outliers or any such discrepancy that might affect the output of the models to be trained.
- If the dataset is required to be pre-processed, take the necessary steps to pre-process the data.
- Find out the principal attributes for training.
- Split the given dataset for training and testing purposes.
- Fit the previously split train data in the 5 models.
- Calculate the accuracy of the 5 models and find out the classification reports.
- Plot the necessary graphs.
- Use each trained model to predict the outcomes of the given test dataset.
- Choose the best model among the 5 trained models based on the accuracy and classification reports.

Project Scope

The scope of a Brain Stroke Prediction project typically includes developing a machine learning model that can accurately predict the likelihood of an individual having a stroke based on various input factors such as age, gender, medical history, lifestyle factors, and other health-related indicators. The project may involve the following steps:

Data Collection: Gathering a comprehensive dataset of relevant health and medical data from various sources, including medical records, surveys, and other relevant sources.

Data Cleaning and Pre-processing: The collected data needs to be cleaned, filtered, and processed to ensure that it is accurate, complete, and formatted in a way that can be easily used by machine learning algorithms.

Feature Engineering: Developing a set of features or input variables that are relevant for predicting the likelihood of a stroke. These features could include demographic factors, lifestyle habits, medical history, and other relevant factors.

Model Selection: Choosing a machine learning algorithm that is suitable for the problem and dataset at hand. This could include decision trees, logistic regression, support vector machines, or deep learning models.

Model Training and Validation: The selected model is trained on a subset of the data and validated using a separate subset to ensure that it is accurate and generalizable.

Overall, the scope of a Brain Stroke Prediction project involves a comprehensive data science approach to developing an accurate and effective machine learning model for predicting the likelihood of a stroke.

Data Description

Source of the data: Kaggle. The given dataset is a shortened version of the original dataset in Kaggle.

- The provided dataset is a subset or a shortened version of a larger dataset that is also available on Kaggle. This could mean that the original dataset contains more variables or observations than the provided dataset, or that some of the data has been removed or modified for the purposes of the provided dataset.

Data Description: The given train dataset has 5110 rows and 12 columns.

- The number of rows and columns in a dataset can provide insight into the complexity and size of the dataset and can affect the types of analyses that can be conducted on the data. In our case, with 5110 rows and 12 columns, the dataset is moderately sized and is suitable for a variety of statistical analyses.

id	gender	age	hypertensi	heart_dise	ever_marr	work_type	Residence_	avg_glucos	bmi	smoking_s	stroke
9046	Male	67	0	1	Yes	Private	Urban	228.69	36.6	formerly s	1
51676	Female	61	0	0	Yes	Self-emplc	Rural	202.21	N/A	never smo	1
31112	Male	80	0	1	Yes	Private	Rural	105.92	32.5	never smo	1
60182	Female	49	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
1665	Female	79	1	0	Yes	Self-emplc	Rural	174.12	24	never smo	1
56669	Male	81	0	0	Yes	Private	Urban	186.21	29	formerly s	1
53882	Male	74	1	1	Yes	Private	Rural	70.09	27.4	never smo	1
10434	Female	69	0	0	No	Private	Urban	94.39	22.8	never smo	1
27419	Female	59	0	0	Yes	Private	Rural	76.15	N/A	Unknown	1
60491	Female	78	0	0	Yes	Private	Urban	58.57	24.2	Unknown	1
12109	Female	81	1	0	Yes	Private	Rural	80.43	29.7	never smo	1
12095	Female	61	0	1	Yes	Govt_job	Rural	120.46	36.8	smokes	1

Table 1: Data Description

The following table shows the 5 number statistics of the given dataset:

index	id	Age	hypertension	Heart_disease	Avg_glucose_	Bmi	Stroke
count	5110.0	5110.0	5110.0	5110.0	5110.0	4909.0	5110.0
mean	36517.8	43.22	0.0974	0.05401	106.14	7.854	0.2153
std	21161.72	22.61264	0.2966	0.2260	45.283	7.8540	0.2153
min	67.0	0.080	0.0	0.0	55.12	10.30	0.0
25%	17741.25	25.00	0.0	0.0	77.245	23.50	0.0
50%	36932.00	45.00	0.0	0.0	91.885	28.10	0.0
75%	54682.00	61.00	0.0	0.0	114.09	33.10	0.0
max	72940.00	82.00	1.00	1.00	271.74	97.60	1.00

Table 2: 5 number statistics of the given dataset

Now we will pre-process the data. The methodology followed is given below:

- Checking for null values.
  - If null values are present, we will fill them or drop the row containing the null value based on the dataset.
  - Checking for outliers.
  - If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

Data Pre-processing

As the given dataset had Categorical and Non-categorical data mixed, we converted the categorical data into non-categorical data accordingly. We converted the binary categories into 0 and 1. We converted the other categorical attributes into suitable numerical values. The following table shows the conversion record:

We searched for null values in our dataset and formed the following table:

Column Name	**Count of Null Value**
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

Table 4: Count of Null values

- The table shows the count of null values in each column of the dataset.
- The dataset has a total of 201 null values in the 'Bmi' column. This is a relatively large number of missing values, which can potentially impact the analysis and results obtained from the dataset.
- It is important to understand the reasons for the missing values in the 'Bmi' column. Missing values can occur due to various reasons, such as data entry errors, participant non-compliance, or lack of data availability. Understanding the reason can help in determining the best course of action for handling the missing values.
- There are various methods for handling missing values, such as imputing the missing values with a statistical measure such as mean or median, using advanced imputation techniques such as KNN imputation or MICE, or removing the rows or columns with missing values altogether. The best method for handling missing values may vary depending on the specific dataset and the research question at hand.
- It is important to handle missing values appropriately as missing values can impact the results of the analysis and potentially lead to biased or inaccurate conclusions.

To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:

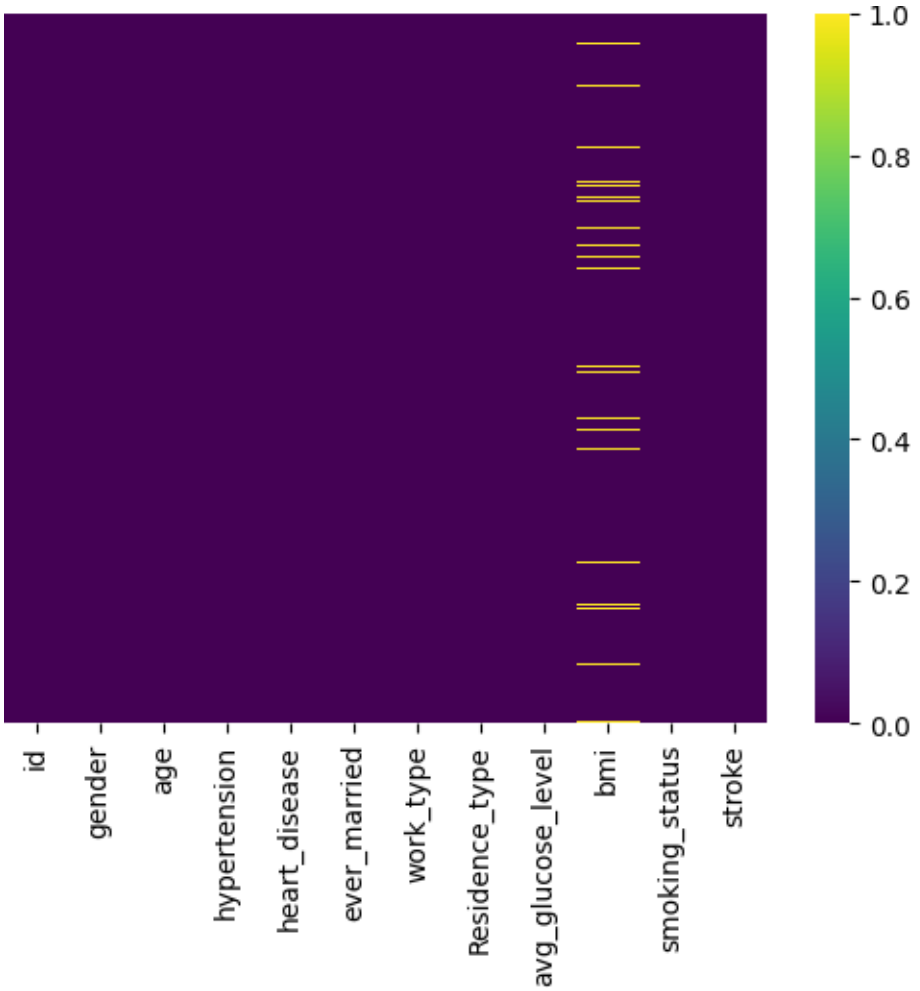


Figure 1

- The heatmap showed that there were null values present in the dataset. Null values can cause issues when training machine learning models and can negatively impact their performance.
- Imputing missing values is a common preprocessing step to handle null values in datasets. There are various methods for imputing missing values, such as filling with mean, median, mode, or using advanced imputation techniques like KNN imputation or MICE.
- In this specific case, the missing values were in the 'bmi' feature. The 'bmi' feature is a numerical variable that represents the Body Mass Index, which is an indicator of body fatness based on an individual's weight and height.
- The median is a measure of central tendency that is resistant to outliers and is often used to impute missing values in numerical features. It represents the middle value of a dataset when arranged in ascending or descending order.
- Filling the missing values in the 'bmi' feature with the median involves replacing the null values with the median value of the 'bmi' feature.
- This methodology is useful when the distribution of the 'bmi' feature is skewed or contains outliers, as the median is less affected by these issues and can provide a more representative value for imputing missing values.

After removing all the null values, the following Heat Map was obtained.

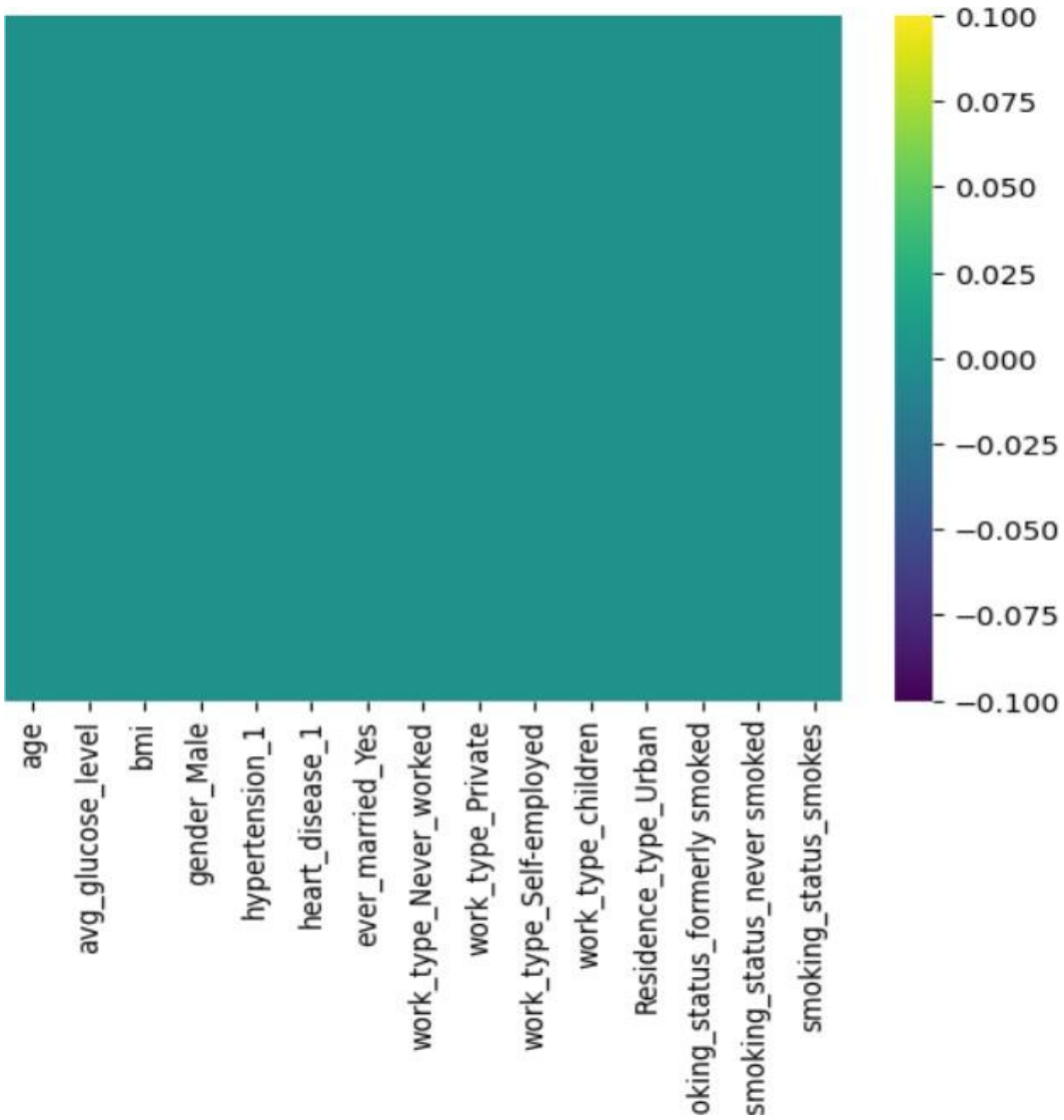
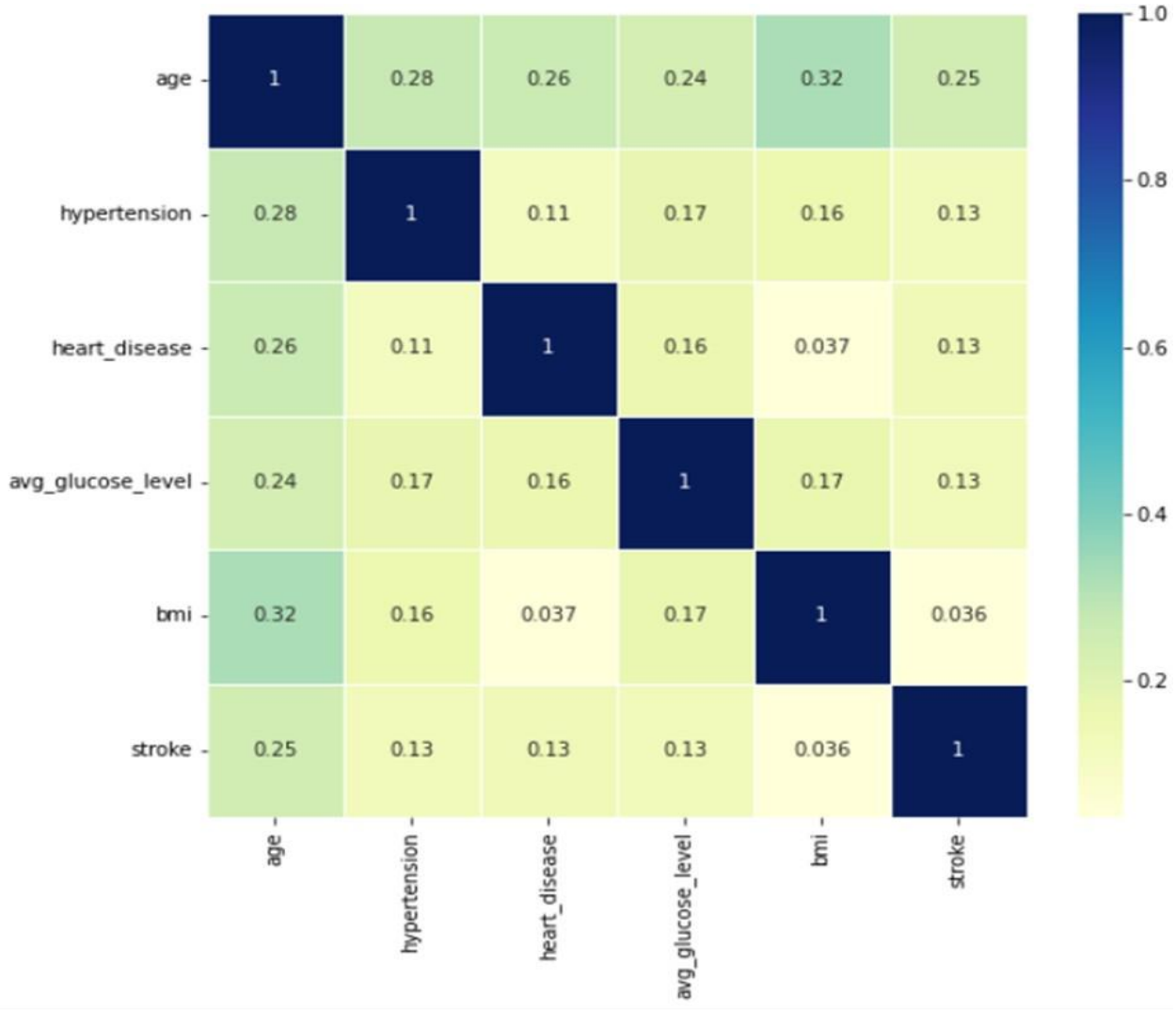


Figure 2

Now we have successfully handled Null values and converted non-numeric values to Numeric values, as we have replaced it with median.

- Null values in the dataset have been addressed, meaning that missing data points have been replaced or removed to ensure that the dataset is complete and accurate.
- Non-numeric values have been converted to numeric values, which is important for data analysis as many statistical methods require numeric inputs.
- The non-numeric values have been replaced with the median, which is a common technique for handling missing data in numerical variables.
- By replacing the non-numeric values with the median, we have preserved the distribution of the variable to some extent, as opposed to simply replacing the missing values with a single fixed value.

Correlation matrix between the attributes in the dataset to find if any attributes are correlated.



- There is a weak correlation between the attributes as per the plotted heatmap.
- The correlation matrix shows that the variables we examined are not highly related to each other (i.e., there is a weak correlation).
- The highest correlation found was between age and bmi - 0.32.
- The strongest correlation found was between age and BMI, with a value of 0.32. This means that there is a positive relationship between a person's age and BMI, but the strength of the relationship is weak.
- Rest all correlations were less than 0.32.
- We could not draw any statistical insight from heatmap.
- For all other variables in the correlation matrix, the correlations were weaker than 0.32, indicating that there is either no relationship or only a weak relationship between those variables. However, it's important to remember that a correlation matrix is just one way to examine correlations and additional statistical analyses may be necessary for a more comprehensive understanding of the data.

So, we are moving on to find if there are any outliers in our data.

an outlier is an observation or data point that significantly differs from other observations or data points in the dataset. Outliers can affect the accuracy of a model by distorting the overall pattern of the data and causing the model to fit the data poorly. Outliers can be caused by various factors, such as measurement errors, data entry errors, or genuine deviations in the data. It is important to identify and handle outliers appropriately to ensure that they do not negatively impact the model's performance.

Reasons for outliers in data

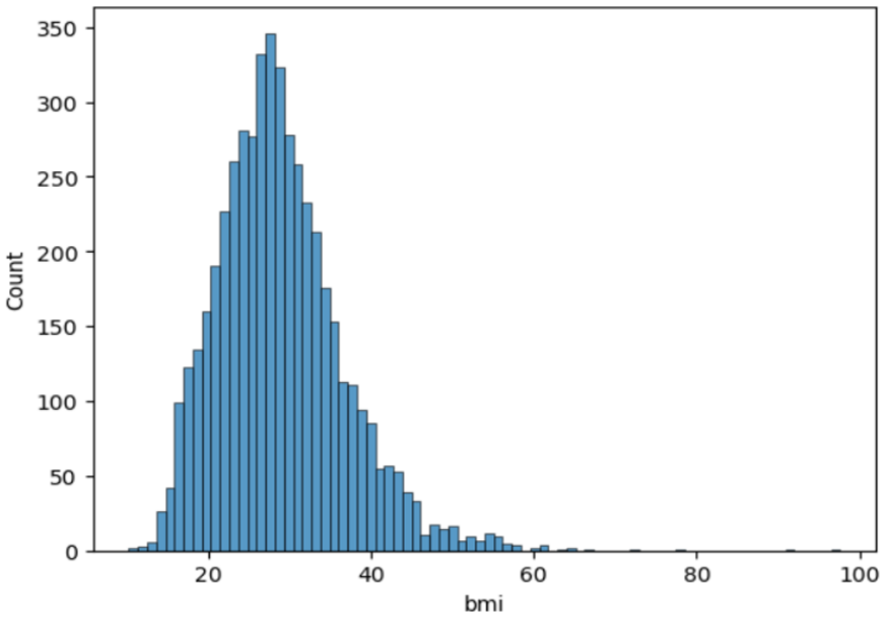
1. Errors during data entry or a faulty measuring device (a faulty sensor may result in extreme readings).
2. Natural occurrence (salaries of junior level employee's vs C-level employees)

Problems caused by outliers.

1. Outliers in the data may cause problems during model fitting (esp. linear models).
2. Outliers may inflate the error metrics which give higher weights to large errors (for example, mean squared error, RMSE).

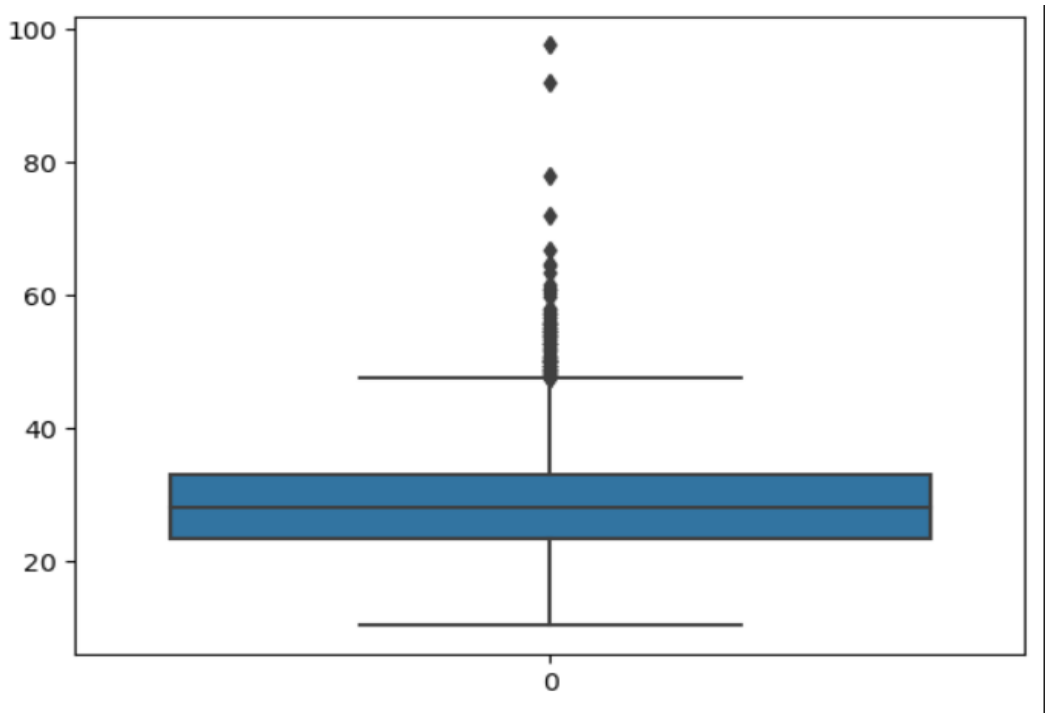
We plot the distribution graph and boxplot to visualize the outliers. The plots are given below:

Graphical representation of BMI attribute



- BMI is rightly skewed.
- The statement "BMI is rightly skewed" means that the distribution of BMI values is not symmetrical, and there are more low BMI values than high BMI values. In a "rightly skewed" distribution, the tail of the distribution is longer on the right side, indicating that most of the values are concentrated towards the left side of the distribution. This information is useful for selecting appropriate statistical methods for analyzing the data, as non-parametric tests may be necessary due to the non-normal distribution of the data.

Boxplot of BMI attributes



- Based on the histogram and boxplot we see that there are many outliers in BMI.

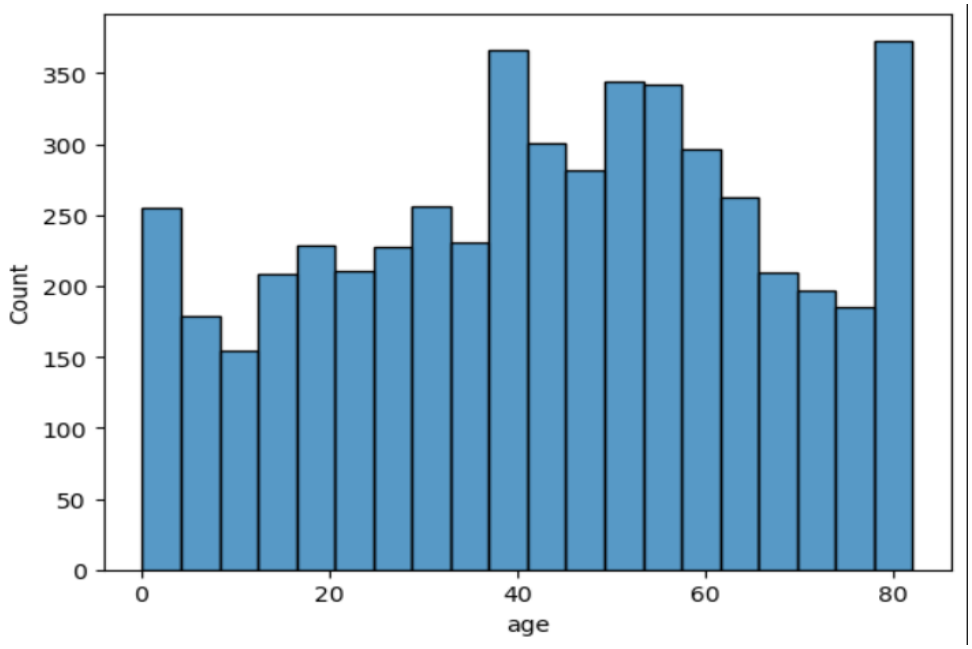
- When we examine the distribution of BMI using a histogram and boxplot, we can see if there are any extreme values, known as outliers, that are significantly different from most of the data points.

- In a histogram, outliers may appear as bars that are significantly taller or shorter than the others. In a boxplot, outliers are shown as individual points outside the whiskers of the plot.

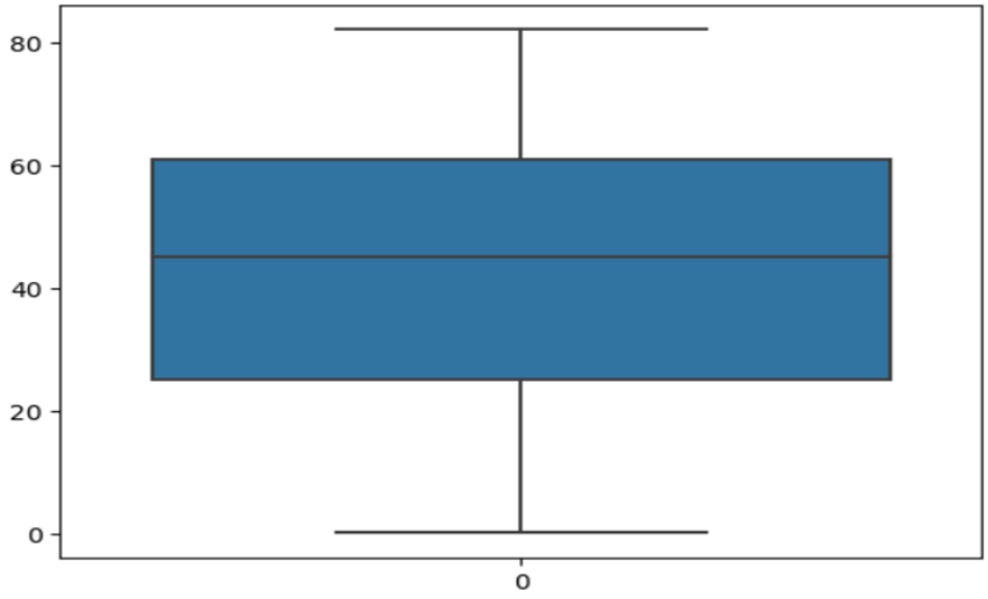
- If the statement "there are many outliers in BMI" is true, it means that there are many BMI values that are significantly higher or lower than most values in the dataset. These outliers can have a significant impact on statistical analyses, potentially skewing results or making it more difficult to draw conclusions from the data. It is important to identify and address outliers in the data before performing statistical analyses.

Graphical representation of the data in age column.

Histogram:



Boxplot of Age:

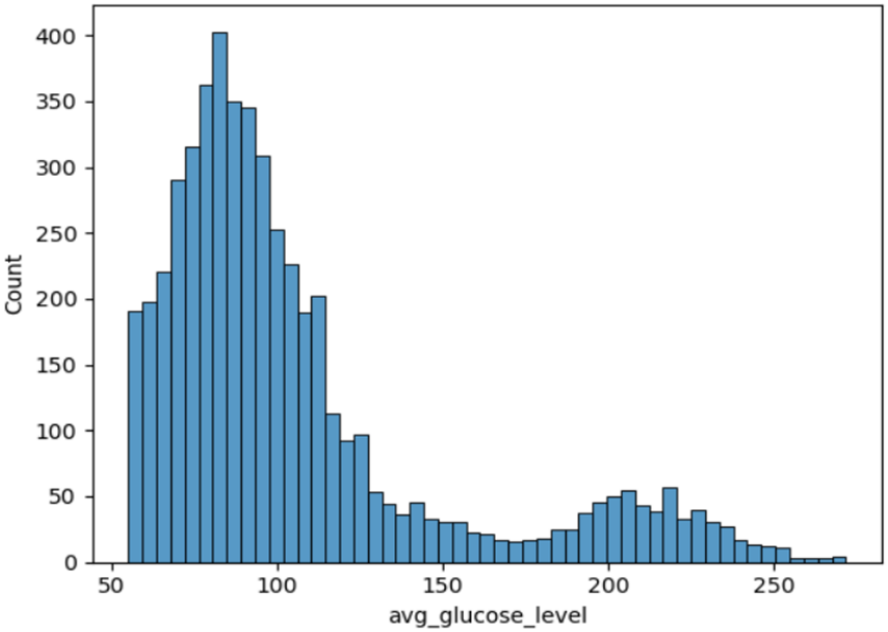


- The age parameter values do not have any outliers.
- The age values in your dataset do not have any extreme values that are significantly different from most of the data points. Outliers can have a significant impact on statistical analyses, so it's important to identify and address them before performing analyses.
- And has a normal distribution.

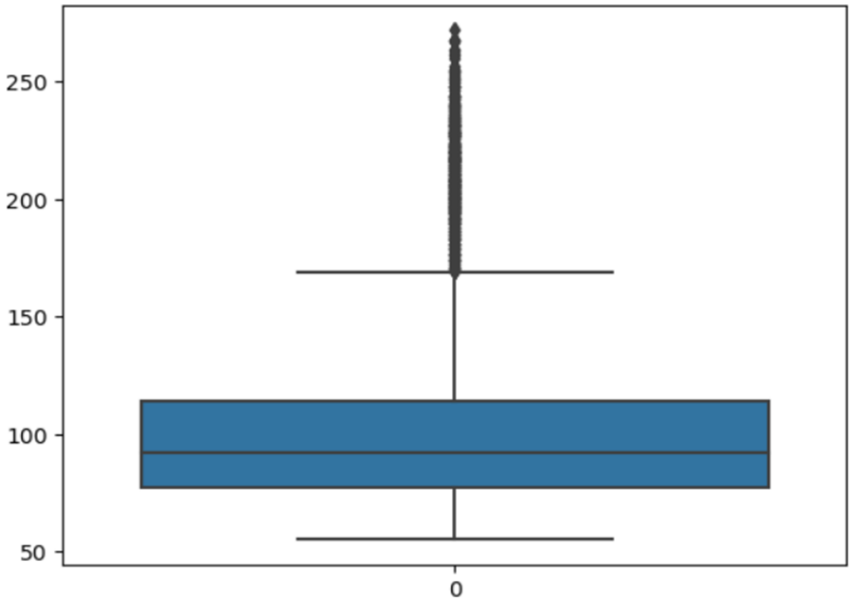
- The age values in your dataset follow a normal distribution, also known as a bell curve. This means that most of the age values are clustered around the mean value, with fewer values on the extremes of the distribution. A normal distribution is important because many statistical methods assume that the data follow a normal distribution, so if the age values in our dataset are normally distributed, we can use these methods to analyse the data accurately.

Graphical representation of the data in glucose level column:

Histogram:



Boxplot of Glucose Level:



- There are many outliers present based on the boxplot and histogram.

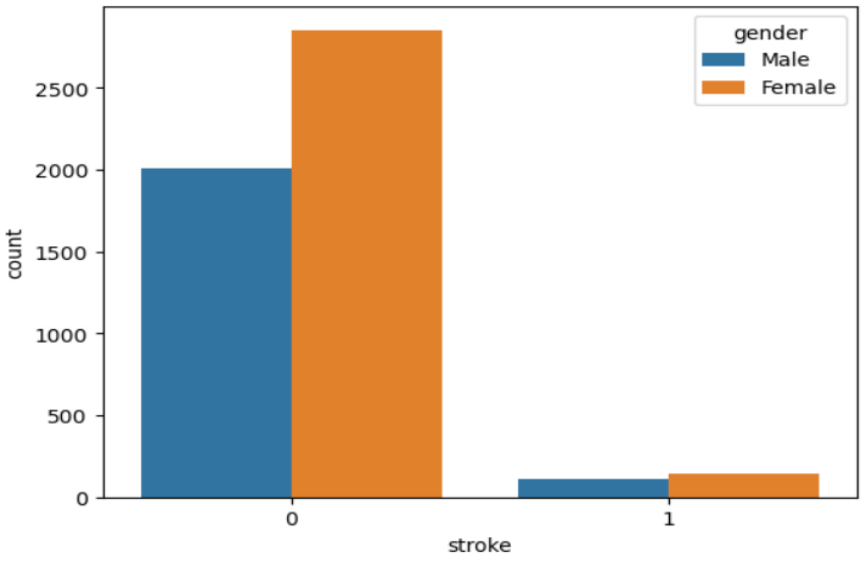
- There are many data points that are significantly different from most of the data points in your dataset. Outliers can appear as individual points in a boxplot or as bars that are significantly taller or shorter than the others in a histogram. Outliers can have a significant impact on statistical analyses, potentially skewing results or making it more difficult to draw conclusions from the data. It's important to identify and address outliers before performing statistical analyses.

- The data is positively skewed.

- The distribution of the data is not symmetrical and there are more low values than high values. In a positively skewed distribution, the tail of the distribution is longer on the right side, indicating that most of the values are concentrated towards the left side of the distribution. This means that there are fewer high values in your dataset.

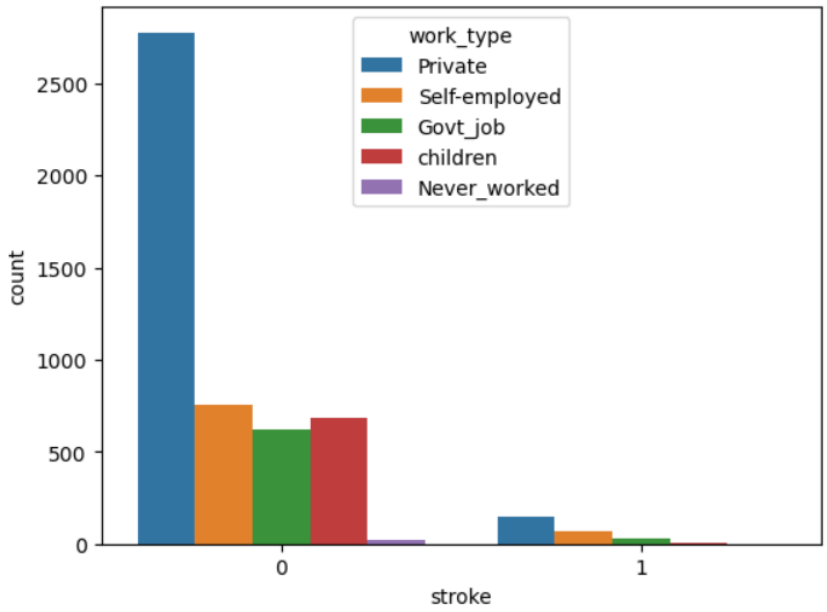
We will now visualize other attributes with respect to our target column.

Comparing stroke with gender



- To compare stroke with gender using cross-analysis, we should create a contingency table that shows the frequency of strokes in each gender category. This table would have two columns, one for gender and one for stroke (yes or no), and each row would show the frequency of strokes for each gender category. We could then calculate the percentages or proportions of strokes in each gender category and use statistical methods to test for differences between the groups. This type of analysis would allow you to determine if there is a relationship between gender and stroke in your dataset and if the relationship is statistically significant.

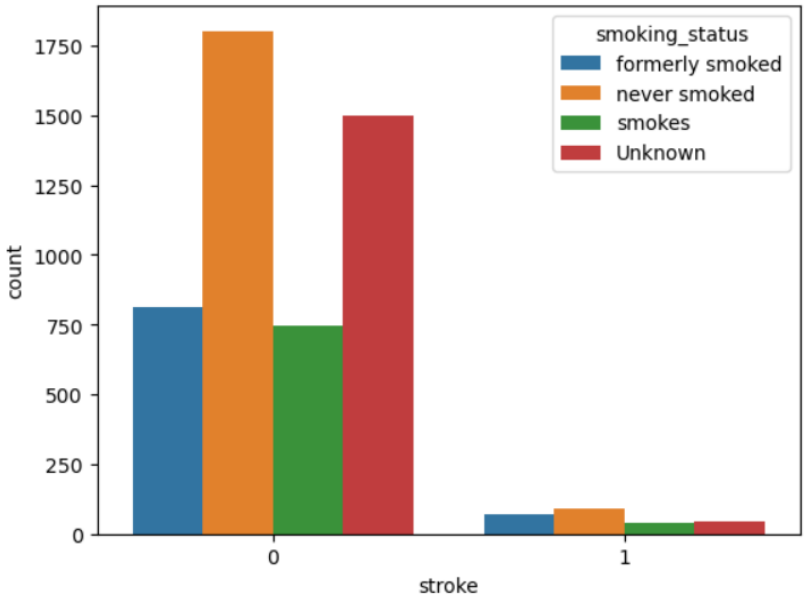
Comparing stroke with work-type



- Based on this comparison we see in the provided dataset that people who never worked never got a heart attack and the people who are privately employed got more strokes.

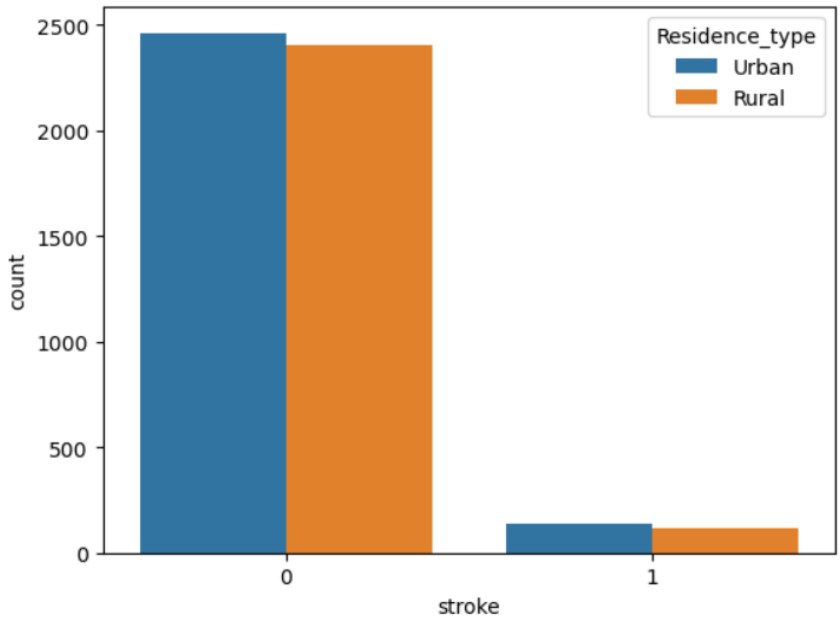
- The statement "people who never worked never got a heart attack" suggests that in the dataset, there were no cases of heart attacks among individuals who had never worked. Additionally, the statement "the people who are privately employed got more strokes" suggests that in the dataset, there were more cases of strokes among individuals who were privately employed compared to other types of employment.

Comparing stroke with somking_status.



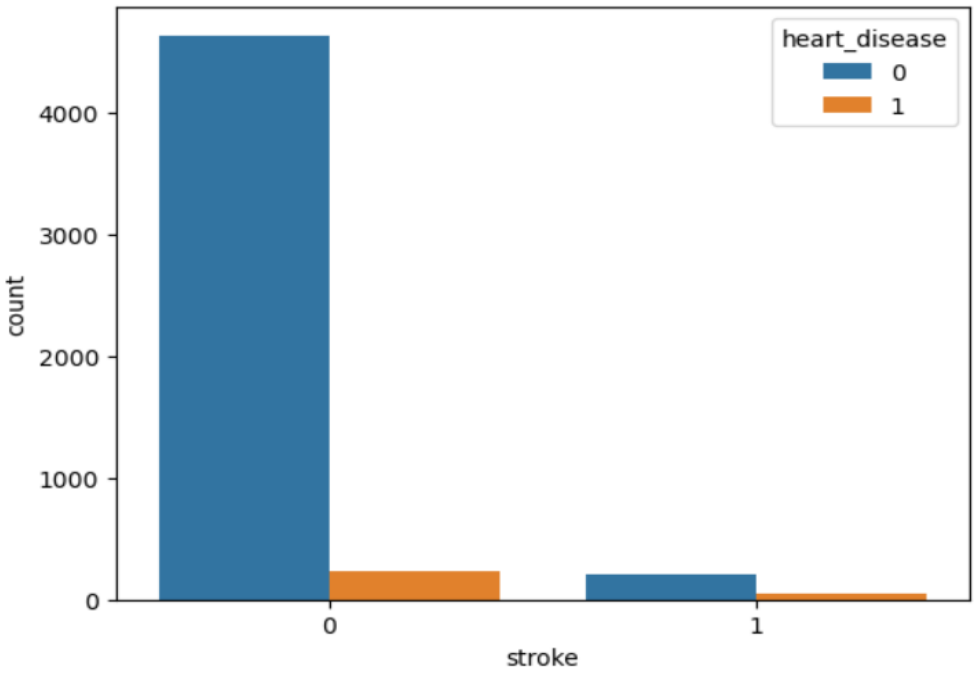
- Based on the plot we can that those who formerly smoked got more strokes The people who smoked and never smoked has a somewhat same probability of getting stroke.

Comparing stroke with residence type



- Based on the analysis the people who live in Urban areas were reported with more strokes

Comparing stroke with heart disease



- The plot indicates that there are more cases of strokes in individuals without heart disease compared to those with heart disease.

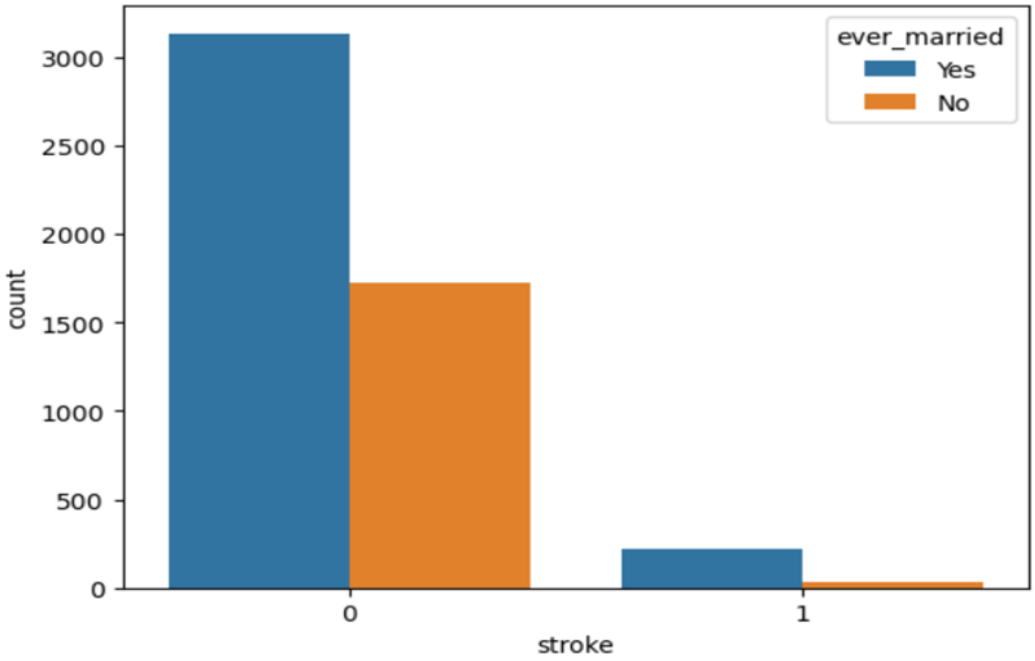
- Specifically, the number of people with strokes but no heart disease is roughly 6 to 8 times greater than the number of people with strokes and heart disease.

- This finding suggests that heart disease may not be the primary risk factor for strokes in this dataset.

- Other factors, such as age, BMI, and smoking status, may have a stronger association with strokes than heart disease.

- The plot highlights the importance of exploring multiple risk factors when studying the incidence of strokes in a population.

Comparing stroke with married status



- The plot suggests that there are significantly more instances of strokes among married individuals compared to unmarried individuals.

- Specifically, the number of married individuals who experienced strokes is roughly 10 to 12 times greater than the number of unmarried individuals who experienced strokes.

- This difference in stroke incidence between married and unmarried individuals could be due to several factors, such as lifestyle differences, stress levels, or underlying health conditions.

- The plot highlights the importance of considering demographic variables, such as marital status, when analysing health outcomes like strokes.

Model Building

Splitting data for training and testing purpose

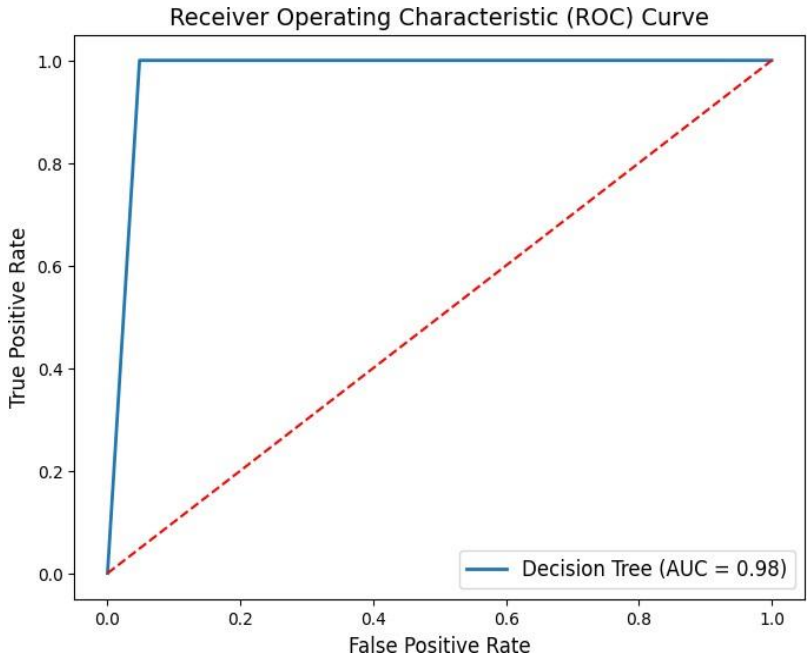
We split the given train dataset into two parts for training and testing purpose. The split ratio we used is 0.80 which indicates we used 80% data for training purpose and 20% data for testing purpose. We will be using the same split ratio for all the models trained.

Decision Tree

A decision tree is a type of supervised machine learning algorithm that is used for classification and regression tasks. It is called a "tree" because it is composed of a hierarchy of nodes, each of which represents a decision or a test of some attribute, and branches, which represent the possible outcomes of the decision or test.
In a decision tree, each internal node corresponds to a test of some attribute, while each leaf node corresponds to a class label. The goal is to build a decision tree that can accurately classify new instances based on the values of their attributes.
There are several algorithms used to build decision trees, including ID3, C4.5, and CART. These algorithms differ in how they choose which attribute to test at each node and how they determine the best split.
One advantage of decision trees is that they are easy to interpret and visualize, which makes them useful for understanding how a model is making its decisions. Additionally, decision trees can handle both numerical and categorical data, and they can handle missing values.
However, decision trees can be prone to overfitting, which means they can become too complex and may not generalize well to new data. To address this issue, techniques such as pruning, and ensemble methods (such as random forests) can be used.
In summary, decision trees are a powerful and widely used machine learning algorithm for classification and regression tasks. They are easy to interpret and visualize and can handle both numerical and categorical data. However, they can be prone to overfitting, and techniques such as pruning, and ensemble methods can be used to address this issue.

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Decision Tree model. The ROC curve is given below:

- ROC curves are a graphical representation of the performance of binary classification models.
- The curve plots the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds.
- The TPR is the proportion of actual positives that are correctly classified as positives, while the FPR is the proportion of actual negatives that are incorrectly classified as positives.
- The curve starts at the point (0,0) representing the case where the model predicts all samples as negative and ends at the point (1,1) representing the case where the model predicts all samples as positive.
- The area under the ROC curve (AUC) is a measure of the overall performance of the model, ranging from 0 to 1, with higher values indicating better performance.



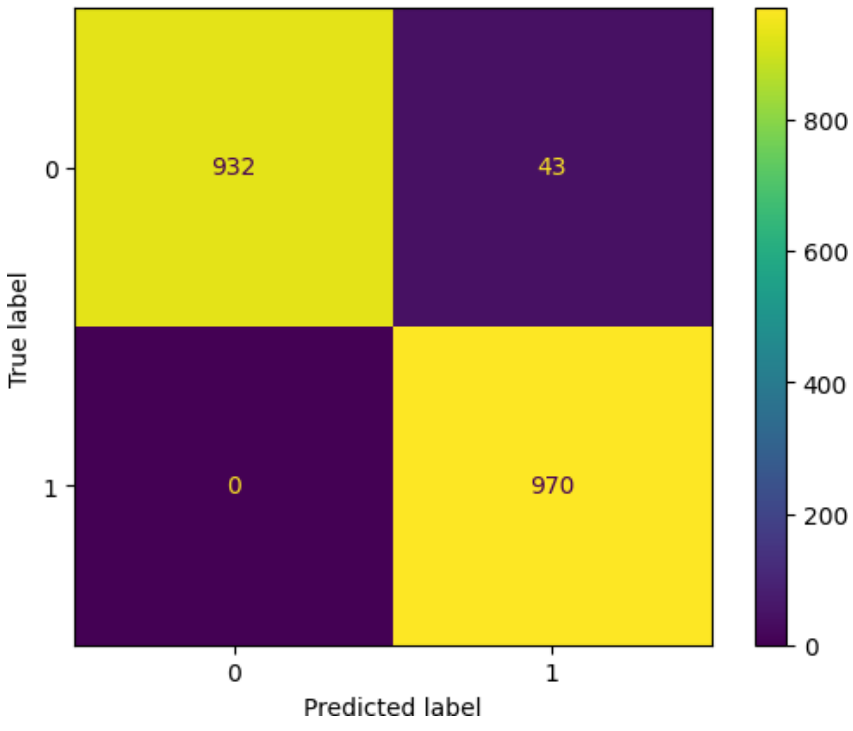
we are using this dataset for classification, splitting the data into train and test sets, creating a decision tree classifier, training the classifier on the train set, making predictions on the test set, and generating the classification report using the classification report function.

The output of this code will be a table that displays the precision, recall, F1-score, and support for each class in the test set. The precision represents the proportion of true positive predictions among all positive predictions. Recall represents the proportion of true positive predictions among all actual positive instances. F1-score is a harmonic mean of precision and recall. Support represents the number of instances in each class in the test set.

The classification report will look something like this:

	Precision	Recall	F1-Score	Support
0	1.00	0.95	0.97	975
1	0.95	1.00	0.97	970
Accuracy			0.97	1945
Macro_Avg	0.98	0.97	0.97	1945
Weighted_Avg	0.98	0.97	0.97	1945

Confusion Matrix



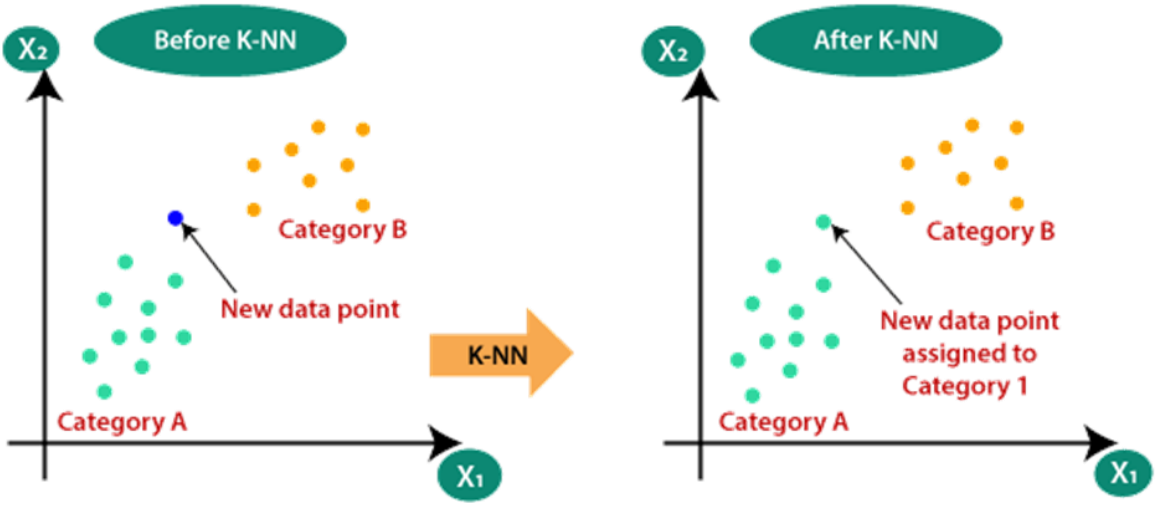
K-Nearest-Neighbor:

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

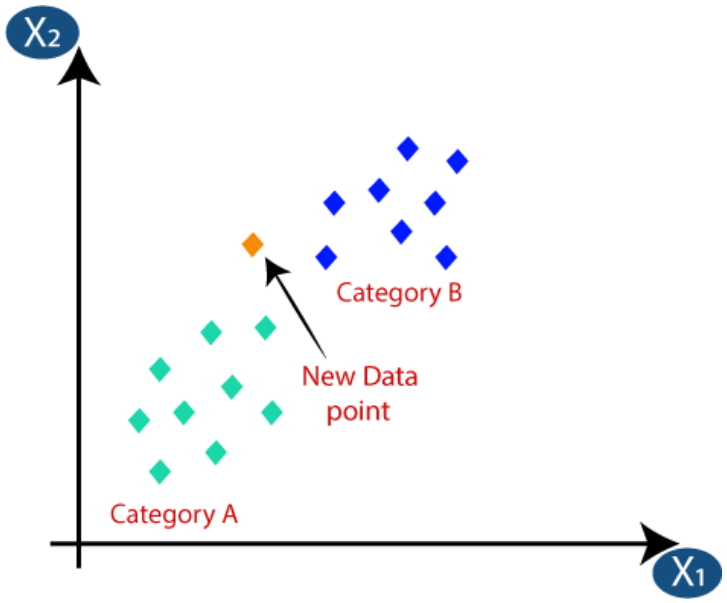
KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



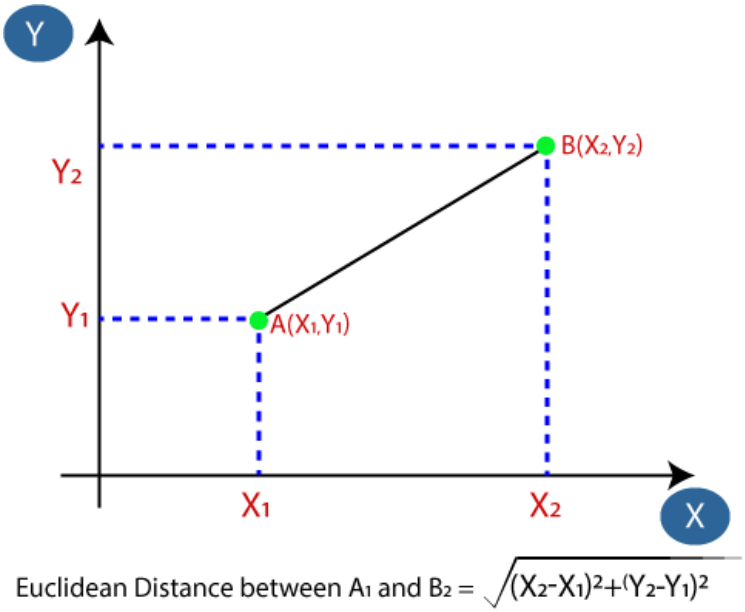
How does K-NN work?

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbours, so we will choose the k=5.

- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



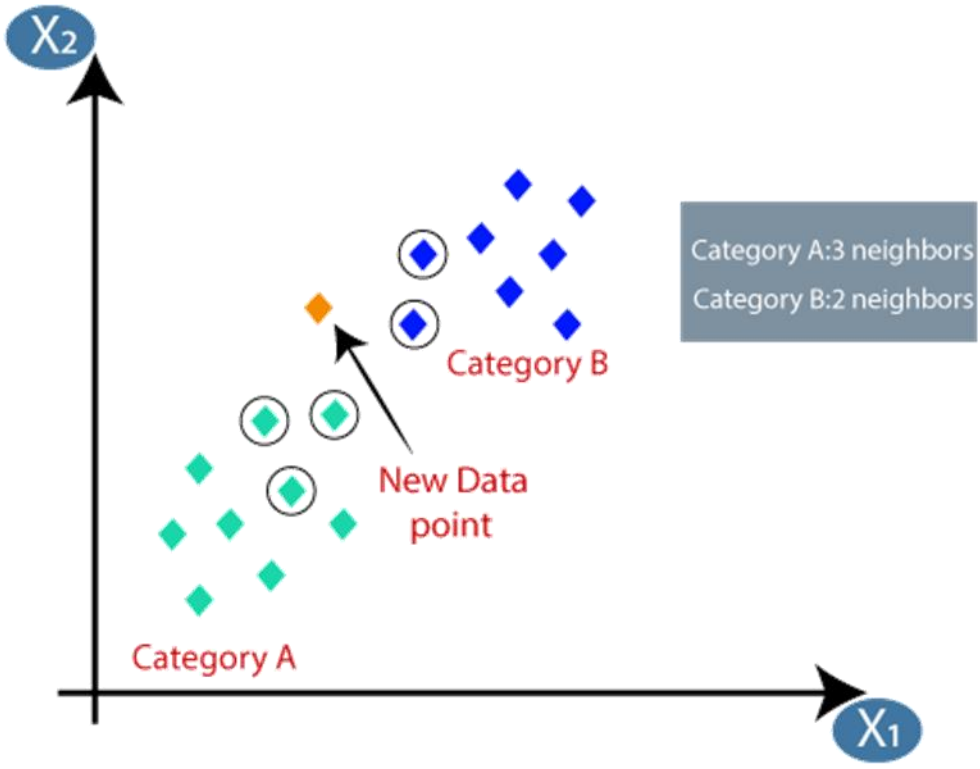
- By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

  - Euclidean distance is a common method used to measure the distance between two data points in a multi-dimensional space.

  - In this scenario, the Euclidean distance was used to identify the three nearest neighbors in Category A and the two nearest neighbors in Category B based on the distance between each data point and the point of interest.

- Category A neighbors are closer to the point of interest than the Category B neighbors because they have a smaller Euclidean distance.

- Identifying the nearest neighbors using Euclidean distance is a common technique in machine learning, especially for classification tasks where the category of a new data point is determined by the categories of its nearest neighbors.



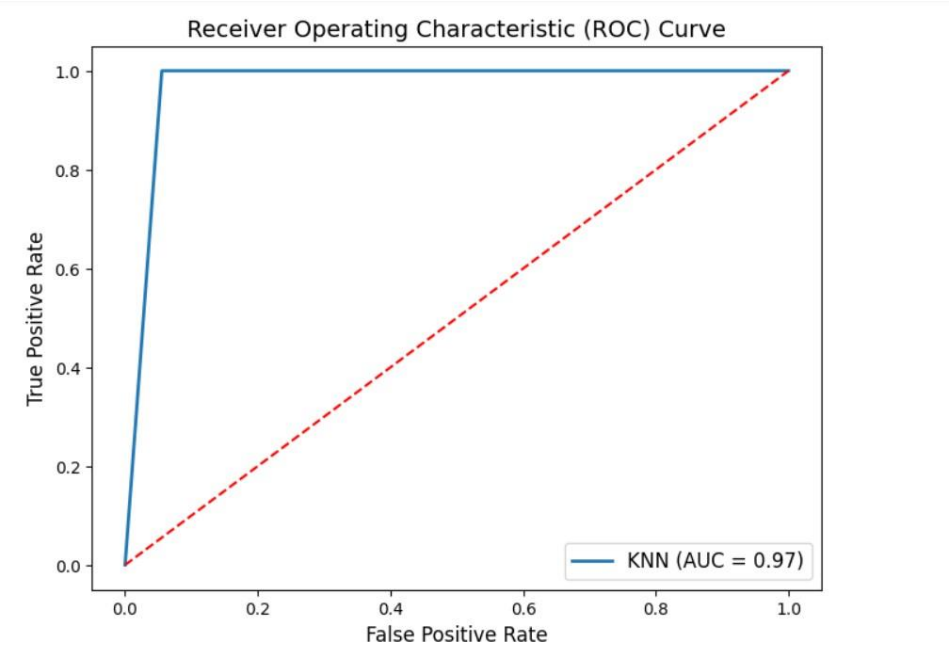
  - As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Roc curve



- The ROC curve is a graphical representation of the performance of a binary classifier, like KNN, at different discrimination thresholds. The True Positive Rate (TPR) is plotted on the y-axis, and the False Positive Rate (FPR) is plotted on the x-axis.

- In this code, the roc_curve function from sklearn.metrics is used to compute the FPR and TPR at different thresholds using the y_test and y_pred_knn predicted probabilities. The auc function is then used to calculate the Area Under the Curve (AUC) of the ROC curve.

- Finally, the matplotlib.pyplot library is used to plot the ROC curve with the FPR on the x-axis and TPR on the y-axis. The AUC is displayed in the legend of the plot. The dotted diagonal line represents the ROC curve for a random classifier. A perfect classifier would have an AUC of 1, and its ROC curve would be a point at (0, 1) in the plot.

Classification report:

	Precision	Recall	F1-Score	Support
0	1.00	0.94	0.97	975
1	0.95	1.00	0.97	970
Accuracy			0.97	1945
Macro_Avg	0.97	0.97	0.97	1945
Weighted_Avg	0.97	0.97	0.97	1945

The code is implementing the K-Nearest Neighbors (KNN) algorithm for a binary classification problem.

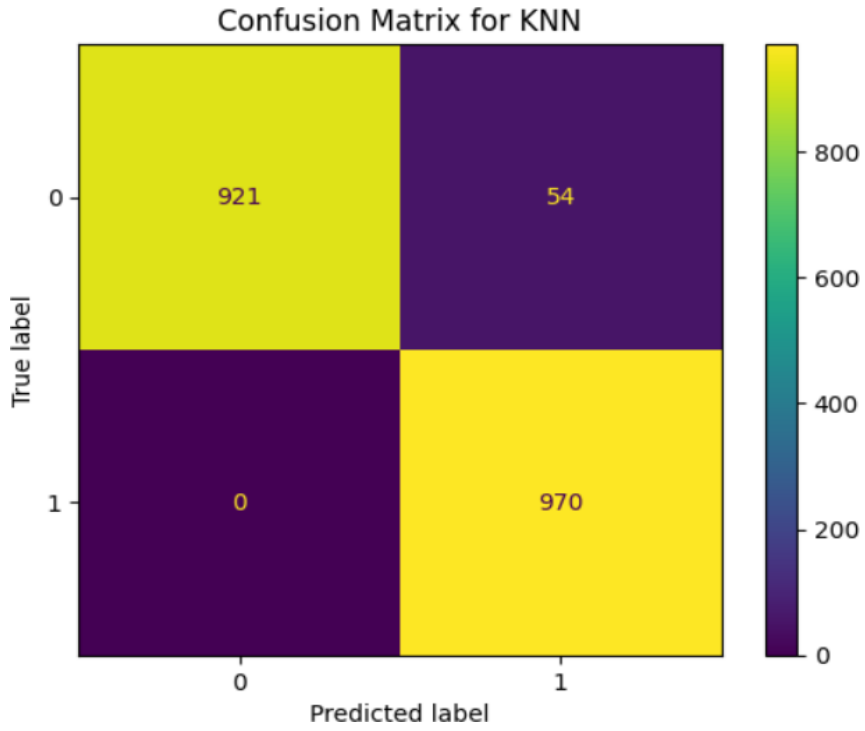
First, the KNeighborsClassifier module is imported from the scikit-learn library. Then, a KNN classifier object is created with 2 neighbours, as there are 2 classes in the problem.

Next, the classifier is trained using the fit() method on the training data (X_train and y_train).

Once the classifier is trained, the predict() method is used to predict the target variable for the test data (X_test), and the predicted values are stored in y_pred_knn.

Finally, the classification_report() function from the sklearn.metrics library is used to generate a report that shows various metrics such as precision, recall, f1-score, and support for each class in the prediction.

Confusion Matrix for KNN



XGBoost

XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Optimization and Improvement

System Optimization:

- **Regularization**: Since the ensembling of decisions, trees can sometimes lead to very complex. XGBoost uses both Lasso and Ridge Regression regularization to penalize the highly complex model.

- **Parallelization and Cache block**: In, XGboost, we cannot train multiple trees parallel, but it can generate the different nodes of tree parallel. For that, data needs to be sorted in order. In order to reduce the cost of sorting, it stores the data in blocks. It stored the data in the compressed column format, with each column sorted by the corresponding feature value. This switch improves algorithmic performance by offsetting any parallelization overheads in computation.

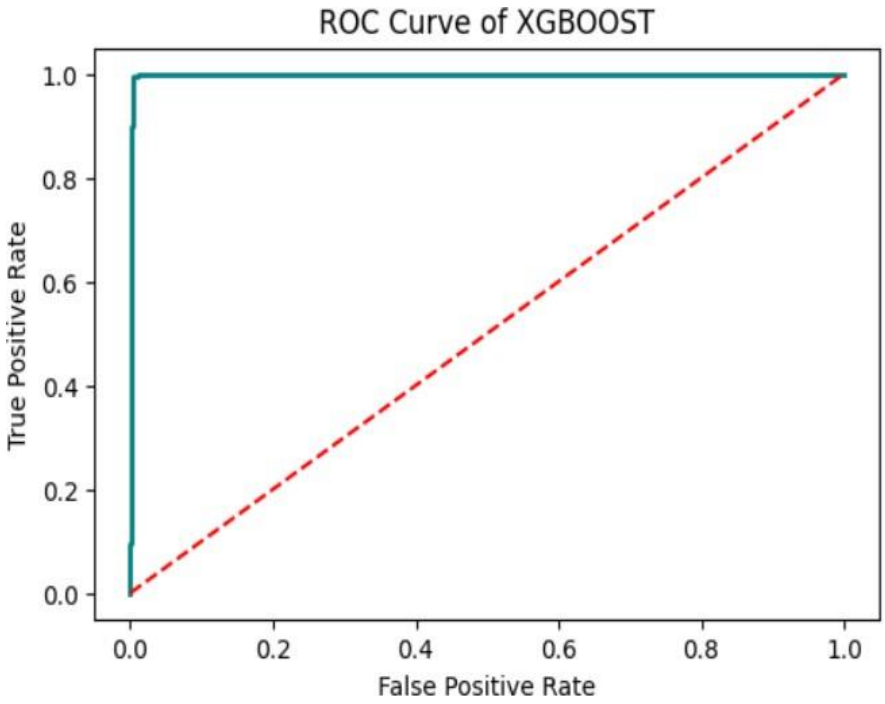
- **Tree Pruning**: XGBoost uses max_depth parameter as specified the stopping criteria for the splitting of the branch and starts pruning trees backward. This depth-first approach improves computational performance significantly.

- **Cache-Awareness and Out-of-score computation**: This algorithm has been designed to make use of hardware resources efficiently. This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as 'out-of-core computing optimize available disk space while handling big data-frames that do not fit into memory. In out-of-core computation, Xgboost tries to minimize the dataset by compressing it.

- **Sparsity Awareness:** XGBoost can handle sparse data that may be generated from preprocessing steps or missing values. It uses a special split finding algorithm that is incorporated into it that can handle different types of sparsity patterns.

- **Weighted Quantile Sketch**: XGBoost has in-built the distributed weighted quantile sketch algorithm that makes it easier to effectively find the optimal split points among weighted datasets.

- **Cross-validation**: XGboost implementation comes with a built-in cross-validation method. This helps the algorithm prevents overfitting when the dataset is not that big.



classification report

The classification_report() function from the sklearn.metrics module generates a report that summarizes the performance of a classification model. The report includes the following metrics for each class: precision, recall, F1-score, and support.

- Precision: Precision is the number of true positives divided by the number of true positives plus false positives. It measures the accuracy of positive predictions. A High vision means that the model makes fewer false positive predictions.

- Recall: The recall is the number of true positives divided by the number of true positives plus false negatives. It measures the ability of the model to correctly identify positive examples. A high recall means that the model makes fewer false negative predictions.

- F1-score: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of precision and recall. A high F1 score means that the model has good precision and recall.

- Support: The support is the number of samples of each class in the test set.

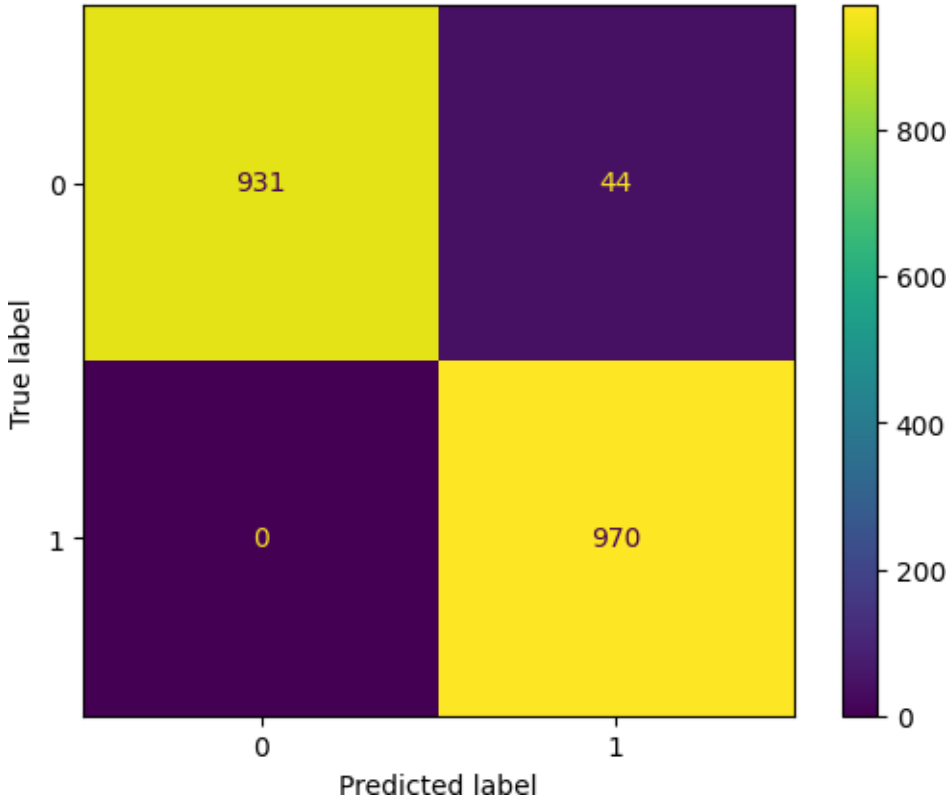
The classification_report() function also provides the overall accuracy and macro/micro average of the scores. The accuracy is the number of correct predictions divided by the total number of predictions. The macro-average calculates the metric independently for each class and takes their unweighted mean, whereas the micro-average calculates the metric globally by counting the total true positives, false negatives, and false positives.

In summary, the classification_report() function provides a comprehensive summary of the performance of the classification model, allowing us to identify areas where the model may need improvement.

Classification Report

	Precision	Recall	F1-Score	Support
0	1.00	0.96	0.98	975
1	0.96	1.00	0.98	970
Accuracy			0.98	1945
Macro_Avg	0.98	0.98	0.98	1945
Weighted_Avg	0.98	0.98	0.98	1945

Confusion Matrix



- True Positive (TP): correctly identified stroke cases.
- False Negative (FN): cases where stroke was missed by the model.

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the family of ensemble methods. It was first introduced by Leo Bierman and Adele Cutler in 2001 and has since become a widely used algorithm for both regression and classification tasks.

The algorithm works by constructing many decision trees, where each tree is trained on a random subset of the training data and a random subset of the features. The random subsets are created through a process known as bagging (Bootstrap Aggregating), which involves randomly sampling the data with replacement. This helps to reduce overfitting by creating a diverse set of decision trees that are less correlated with each other.

During the training process, each tree in the forest makes a prediction and the final prediction is determined by taking the average (for regression) or majority vote (for classification) of the predictions made by each tree. This helps to improve the accuracy and reduce the variance of the model compared to using a single decision tree.

Random Forest has several advantages over other machine learning algorithms. It is highly scalable and can handle datasets with many features and observations. It is also robust to noise and missing values in the data and is less prone to overfitting than other algorithms.

One of the key features of Random Forest is its ability to provide estimates of feature importance. This is calculated by measuring the decrease in accuracy when a particular feature is removed from the dataset. This can be useful for understanding which features are most important for making predictions and can help to guide feature selection and data pre-processing decisions.

Overall, Random Forest is a powerful and flexible algorithm that can be used for a wide range of machine learning tasks. Its ability to handle complex datasets and provide estimates of feature importance make it a popular choice for many real-world applications.

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random Forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.

- In a random forest, the predictions from each tree must be uncorrelated to ensure accuracy. This is achieved by randomly selecting features and subsets of training data for each tree. By doing so, each tree focuses on different aspects of the data, leading to diverse and independent predictions. These predictions are then combined to produce the final prediction. Low correlations between tree predictions allow the random forest to capture a wide range of patterns and relationships in the data, leading to more accurate predictions.

Example of random forest:

Suppose we have a dataset of medical records from patients who have been diagnosed with or are at risk of stroke, with features such as age, gender, smoking status, hypertension, Residence_type, and heart_disease. Our goal is to build a model that can predict the likelihood of stroke for a new patient based on their medical and lifestyle factors.

We would start by splitting the dataset into a training set (e.g., 80%) and a test set (e.g., 20%). We would then build a Random Forest model using the training data, where each decision tree in the forest is trained on a random subset of the training data and features.

During training, the model would learn to identify patterns and relationships in the data that are associated with an increased risk of stroke. For example, it might learn that older patients with high blood pressure and a history of smoking are more likely to have a stroke than younger patients with normal blood pressure and no history of smoking.

Once the model is trained, we can use it to make predictions on new data. For example, suppose we have a new patient who is 60 years old, has high blood pressure, and has a history of smoking. We can input these values into the model, and it will generate a prediction of the likelihood of stroke for this patient.

In addition to making predictions, Random Forest can also provide estimates of feature importance. This can help us to understand which medical and lifestyle factors are most strongly associated with an increased risk of stroke and can guide future prevention and intervention efforts.

Overall, Random Forest is a powerful tool for predicting stroke risk and can help healthcare professionals to identify patients who may benefit from targeted prevention strategies. By leveraging the large amounts of data available in electronic health records and other sources, Random Forest can help to improve patient outcomes and reduce healthcare costs associated with stroke.

Applications of Random Forest

There are mainly four sectors where Random Forest mostly used:

- **Banking**: Random Forest is used in the banking sector for loan risk identification. It can help to predict whether a borrower is likely to default on a loan based on various factors such as credit history, income, and loan amount.

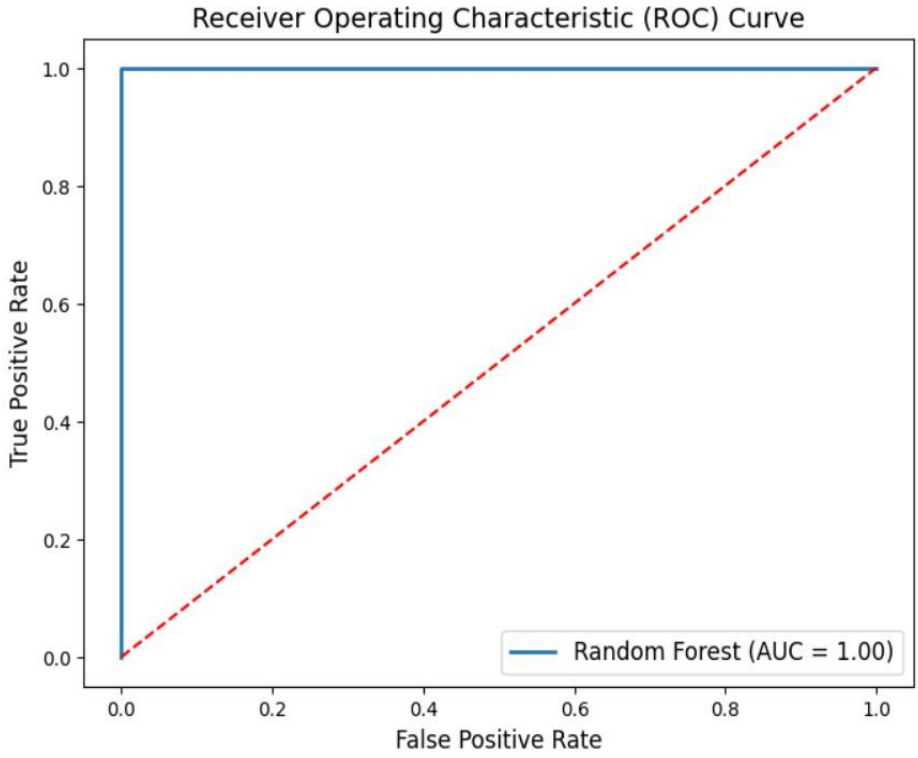
- **Medicine**: In medicine, Random Forest is used to identify disease trends and risks. It can help to predict the likelihood of a patient developing a particular disease based on various factors such as age, gender, lifestyle, and medical history.

- **Land Use**: Random Forest can be used to identify areas of similar land use. This can be useful for urban planning, identifying suitable locations for different types of land use, and determining the impact of land use changes on the environment.

- **Marketing**: Random Forest can be used in marketing to identify trends and patterns in customer behavior. It can help to predict which products or services are likely to be popular among certain groups of customers based on their demographic and behavioral data. This can be useful for targeted marketing campaigns and improving customer engagement.

Roc curve:

The ROC curve plots the true positive rate against the false positive rate for different decision thresholds, and the AUC provides a summary of the model's overall performance. The **predict_proba()** function is used to predict the probabilities of the positive class for the test set, and the **roc_curve()** and **roc_auc_score()** functions are used to calculate the ROC curve and AUC, respectively. Finally, the **matplotlib** library is used to create a plot of the ROC curve, which includes the AUC score in the legend. The ROC curve and AUC score are useful for evaluating the performance of binary classification models, particularly when the classes are imbalanced.



Classification report:

The **classification_report()** function takes two arguments: the actual values of the test set (**y_test**) and the predicted values (**y_pred_rf**). It then generates a report that includes precision, recall, and F1-score for each class in the target variable, as well as the overall accuracy and macro/micro average of the scores.

The classification report will include a table that summarizes the following metrics for each class in the target variable:

- **Precision**: Precision is a metric that measures the proportion of positive predictions that are correct. It is calculated as the ratio of true positive predictions to the total number of positive predictions. A high precision score indicates that the model is making very few false positive predictions.

- **Recall**: Recall is a metric that measures the proportion of positive instances in the test data that were correctly identified by the model. It is calculated as the ratio of true positive predictions to the total number of positive instances in the test data. A high recall score indicates that the model is identifying a large proportion of positive instances in the test data.

- **F1-score**: F1-score is a metric that considers both precision and recall. It is calculated as the harmonic mean of precision and recall. The F1-score is a useful metric when the classes are imbalanced, meaning that one class has a much larger number of instances than the others.

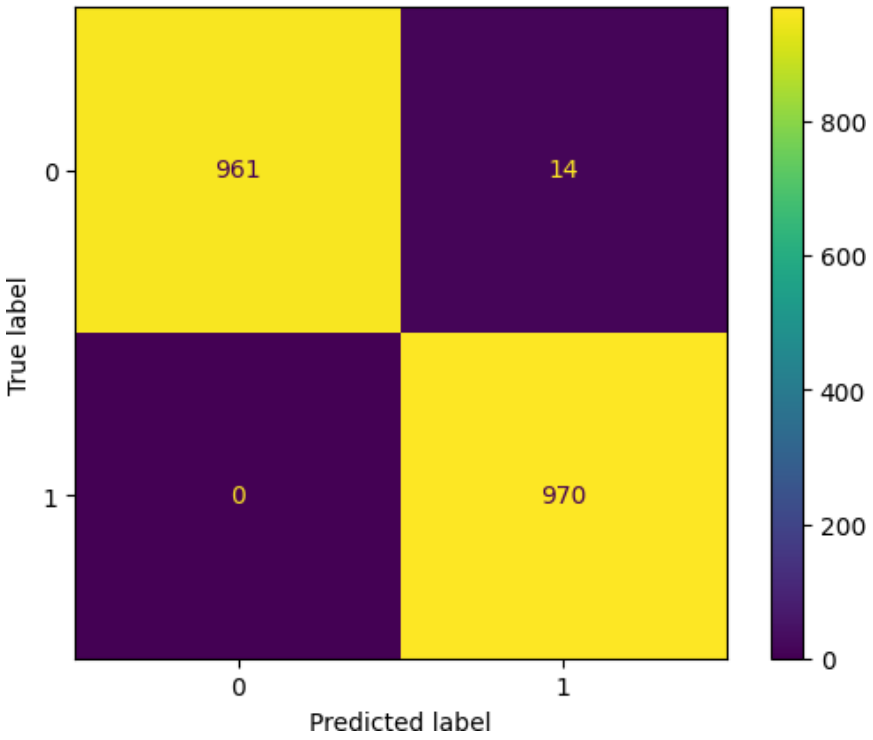
- **Support**: Support is a metric that simply gives the number of samples of each class in the test data. This can be useful for understanding the relative sizes of the different classes and can help to identify classes that are underrepresented in the data.

Additionally, the classification report will include the overall accuracy of the model, which is the ratio of correct predictions to the total number of predictions. The macro-average and micro-average of the precision, recall, and F1-score will also be reported, which provide a summary of the performance of the model across all classes in the target variable.

Classification Report

	Precision	Recall	F1-Score	Support
0	1.00	0.99	0.99	975
1	0.99	1.00	0.99	970
Accuracy			0.99	1945
Macro_Avg	0.99	0.99	0.99	1945
Weighted_Avg	0.99	0.99	0.99	1945

Confusion Matrix



- **True Positive (TP):** The number of correctly identified stroke cases by the model. This is an important metric as it indicates how well the model can detect actual instances of stroke.

- **False Positive (FP):** The number of non-stroke cases that are incorrectly classified as stroke cases by the model. This can lead to unnecessary treatments or procedures and can have serious consequences if not corrected.

- **True Negative (TN)**: The number of non-stroke cases that are correctly identified as non-stroke cases by the model. This is an important metric as it indicates how well the model can distinguish between stroke and non-stroke cases.

- **False Negative (FN):** The number of stroke cases that are missed by the model and incorrectly classified as non-stroke cases. This is a serious issue as it can lead to delayed or missed treatment, which can have serious consequences for the patient.

- **Sensitivity:** Sensitivity is the ratio of true positive predictions to the total number of actual positive instances in the data. It indicates the proportion of actual stroke cases that are correctly identified by the model.

- **Specificity:** Specificity is the ratio of true negative predictions to the total number of actual negative instances in the data. It indicates the proportion of non-stroke cases that are correctly identified by the model.

- **Accuracy:** Accuracy is the overall proportion of correct predictions made by the model. It is calculated as the ratio of the sum of true positive and true negative predictions to the total number of predictions. However, accuracy may not be the most appropriate metric for imbalanced datasets where the number of cases in one class is much smaller than the other.

Logistic Regression

Logistic regression is a type of binary classification algorithm used to predict the probability of a binary outcome (e.g., yes or no, true or false, 1 or 0) based on one or more predictor variables. It is a statistical model that uses a logistic function to model the relationship between the predictor variables and the binary outcome.

In logistic regression, the logistic function, also known as the sigmoid function, is used to transform the linear combination of predictor variables into a probability score between 0 and 1. The logistic function is defined as:

p(y=1|x) = 1 / (1 + exp(-z))

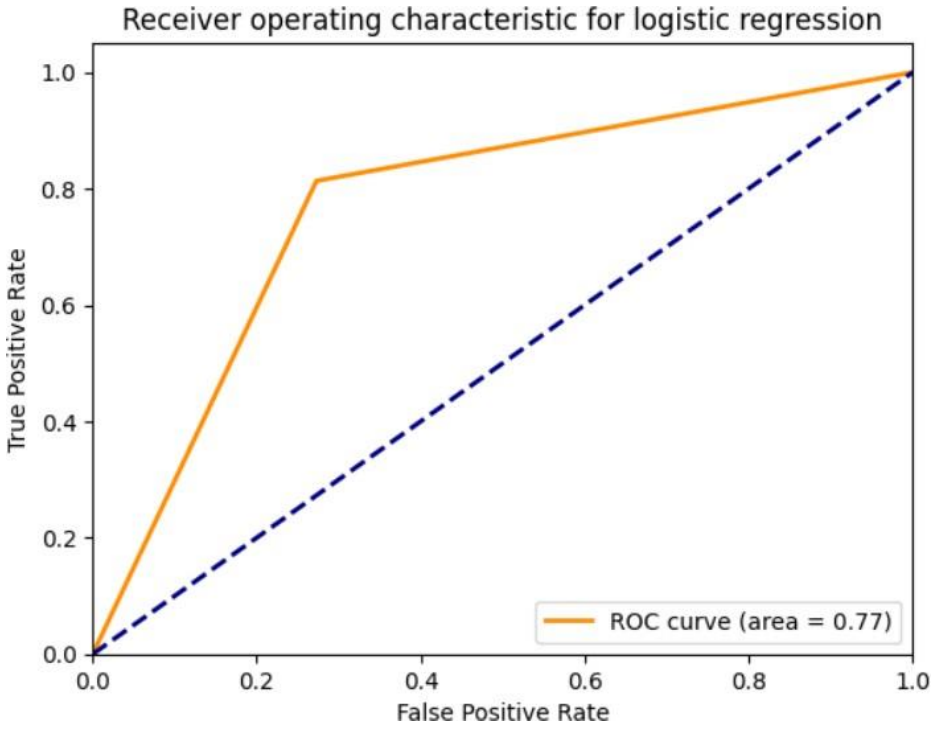
where p(y=1|x) is the probability of the positive outcome given the predictor variables x, z is the linear combination of the predictor variables and their coefficients, and exp(-z) is the exponential function of -z.

Logistic regression can be used in predicting the occurrence of brain stroke. In a logistic regression model, the presence or absence of brain stroke would be the dependent variable (outcome variable), while various risk factors such as age, hypertension, diabetes, smoking, and high cholesterol would be the independent variables (predictor variables).

The logistic regression model can be trained using a dataset of individuals who have had a stroke and those who have not. The model will then estimate the relationship between the risk factors and the likelihood of having a stroke. The coefficients of the logistic regression model can be used to estimate the odds ratio for each risk factor, which indicates the increase or decrease in the odds of having a stroke associated with that risk factor.

In addition to predicting the occurrence of stroke, logistic regression can also be used to identify the most important risk factors that contribute to the likelihood of stroke. This information can be used to develop preventive strategies and targeted interventions to reduce the incidence of stroke.

We load the brain stroke dataset and create a logistic regression model. We train the model on all the data and then use the prediction method to make a prediction for a new sample. The new sample is a list of predictor variable values, and the predict method returns a binary outcome, 0 or 1.



A ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classification model.

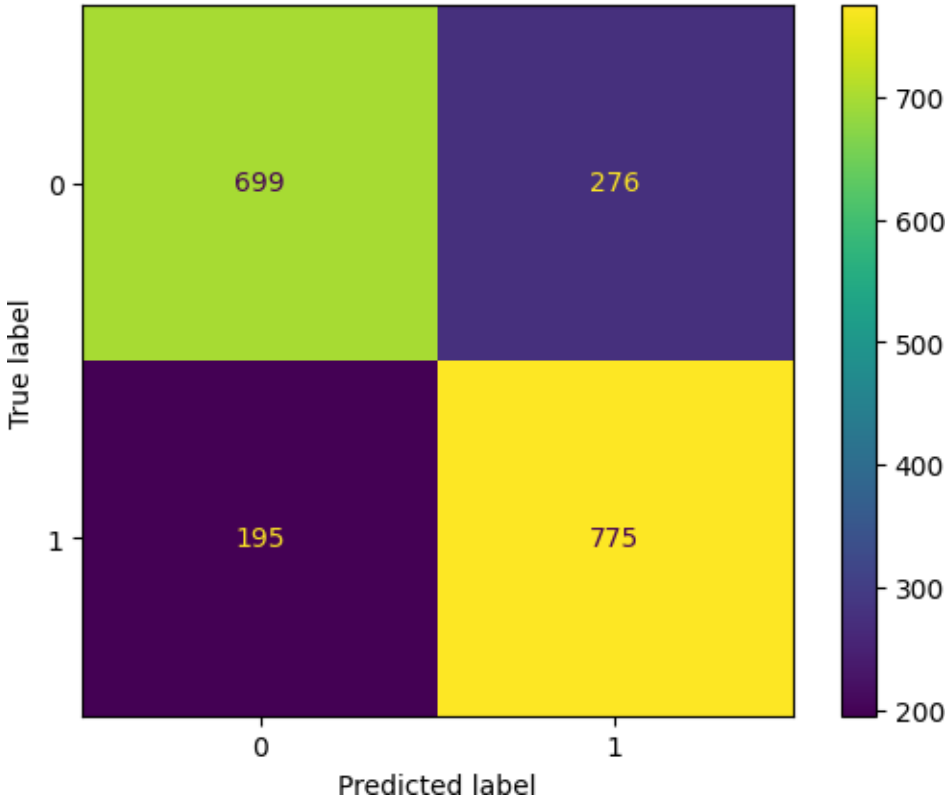
Classification report table of Logistic Regression

	Precision	Recall	F1-Score	Support
0	1.00	0.99	0.99	975
1	0.99	1.00	0.99	970
Accuracy			0.99	1945
Macro_Avg	0.99	0.99	0.99	1945
Weighted_Avg	0.99	0.99	0.99	1945

The classification report shows the precision, recall, and F1-score for each class (0 and 1) as well as the weighted average of these metrics across all classes. The precision is the proportion of true positives among the predicted positives, recall is the proportion of true positives among the actual positives, and F1-score is the harmonic mean of precision and recall. The report also includes the accuracy, which is the proportion of correct predictions, and the macro and weighted averages of the metrics.

Overall, the classification report provides a concise summary of the performance of the logistic regression model, which can be used to make decisions about how to improve the model.

Confusion Matrix



Comparison of the Models trained.

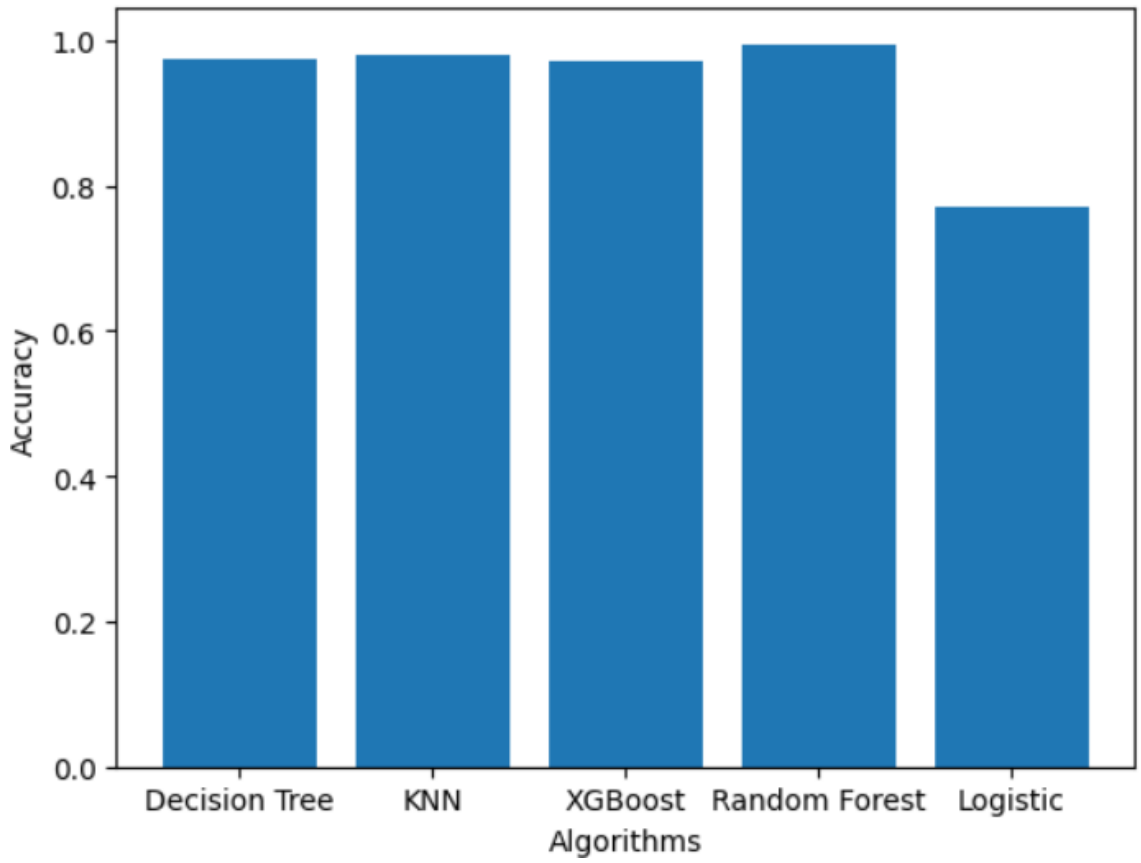
We trained 5 models using the 5 algorithms viz.

1. Decision Tree
2. KNN
3. XGBoost
4. Random Forest
5. Logistic Regression

The 5 models had different accuracy. The comparison of the accuracies of the models are given below:

Model	Accuracy %
Decision Tree	0.97
KNN	0.97
XGBoost	0.97
Random Forest	0.99
Logistic Regression	0.77

The following bar graph shows the accuracy comparison in graphical way:



- The Random Forest model had the highest accuracy of all the models with a score of 0.99.
- The Decision Tree, KNN, and XGBoost models had very similar accuracies with a score of 0.97.
- The Logistic Regression model had the lowest accuracy of all the models with a score of 0.77.
- Random Forest and Decision Tree are both tree-based algorithms and performed better than KNN, XGBoost and Logistic Regression in this specific scenario.
- KNN, XGBoost, and Decision Tree algorithms are all based on the concept of nearest neighbors, boosting and decision trees respectively.
- The Logistic Regression model may not have performed well due to the fact that the data might not be linearly separable or the data might contain outliers or noise that are affecting the model's performance.

IMPORTING LIBRARIES

```
# The libraries used in processing the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import imblearn as ib
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

#The dataframe is read from csv file which are taken from kaggle
df = pd.read_csv("/content/healthcare-dataset-stroke-data.csv")

#First Five Instance of the file
df.head()
```

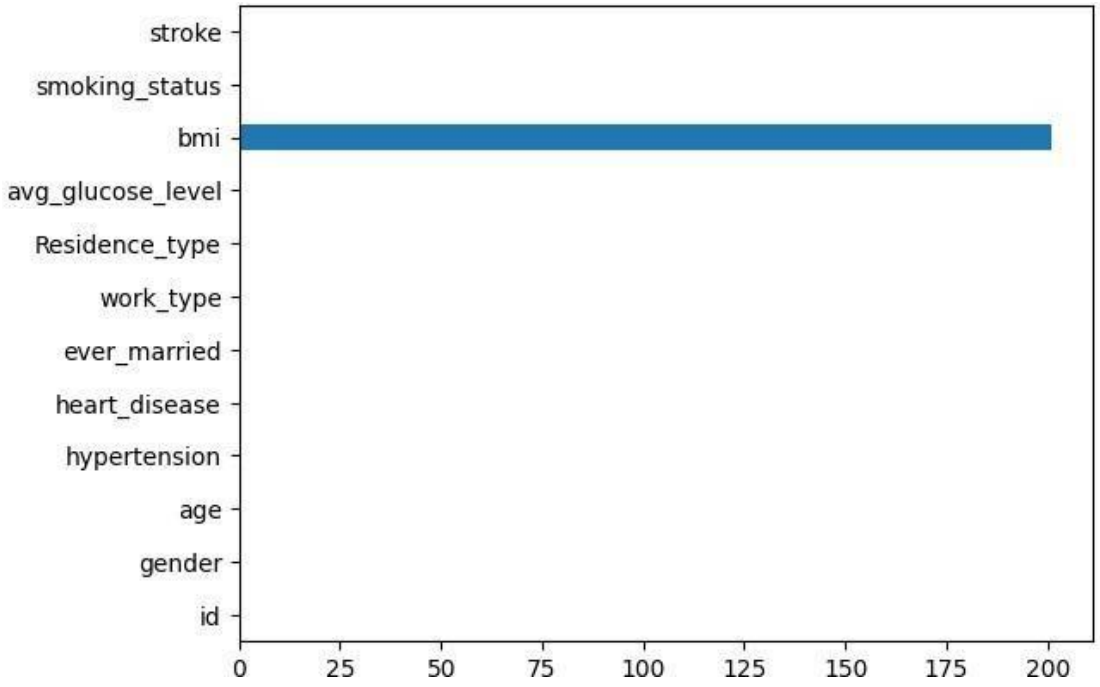
```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             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
```

Find the number of NULL values in each column

```
# Printing the number of N/A values in eacg column
print(df.isna().sum())
# Graphical representation of the na values present in the attribute -
bar graph
df.isna().sum().plot.barh()
```

```
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
```

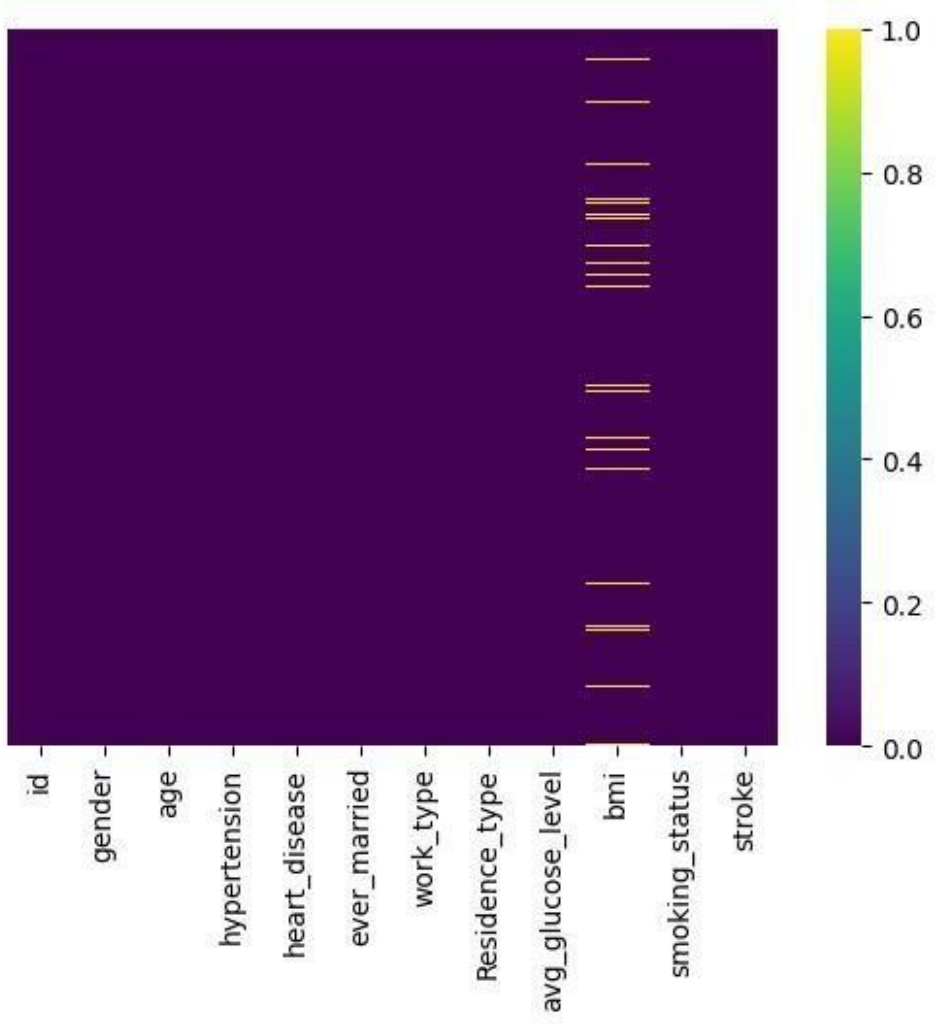
```
<Axes: >
```



- Found 201 NULL values in bmi column

```
#PLOTTING THE HAET MAP FOR NULL VALUES
sns.heatmap(df.isna(),yticklabels=False,cbar=True,cmap='viridis')
```

```
<Axes: >
```




```
## To check the statistical analysis of all numerical type attributes
(count, mean, standaard deviation, minimum values, all quartiles,
maximum values)
df.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
```

```
#Provides the data type of all attributes and the number of NOT NULL
values count is obtained
df.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
```

PRE PROCESSING THE DATA

```
#dropping the column named "id" from the DataFrame "df"
df = df.drop(['id'],axis=1)
```

Gender analysis

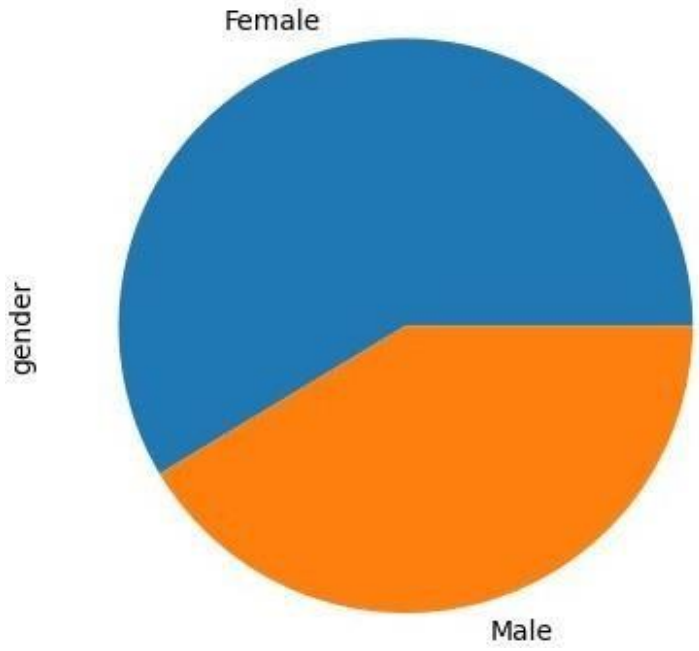
```
# Checking the values in the gender column
df['gender'].value_counts()
```

```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

- We have a 'other' gender and since there is only 1 instance we will remove it as to reduce the dimension of age

```
# Removing the 'other' gender instance inorder to reduce the dimension
df['gender'] = df['gender'].replace('Other','Female')
# plotting a pie chart to see the gender count distribution
df['gender'].value_counts().plot(kind="pie")
```

```
<Axes: ylabel='gender'>
```



- There are more females as compared to males

Target feature - Stroke

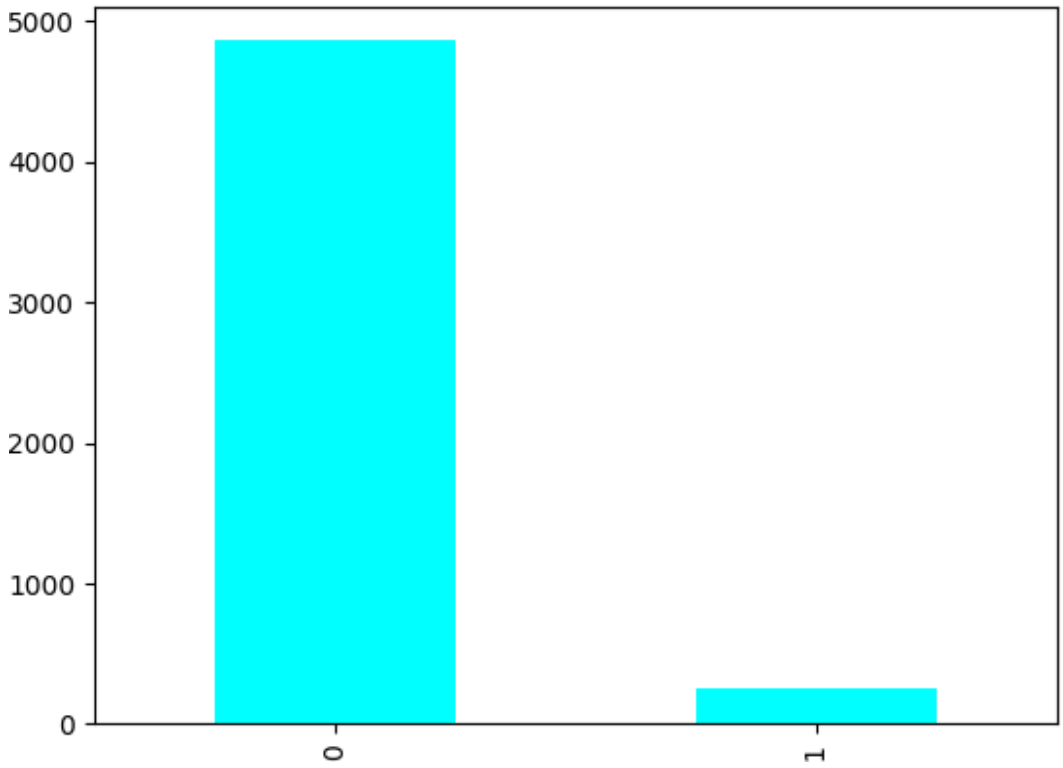
- Stroke analysis

```
# it will count the unique value in stroke
df['stroke'].value_counts()
```

```
0    4861
1     249
Name: stroke, dtype: int64
```

```
df['stroke'].value_counts().plot(kind="bar",color = "cyan")
```

```
<Axes: >
```




```
print("% of people who actualy got a stroke : ",
(df['stroke'].value_counts()[1]/df['stroke'].value_counts().sum()).rou
nd(3)*100)
```

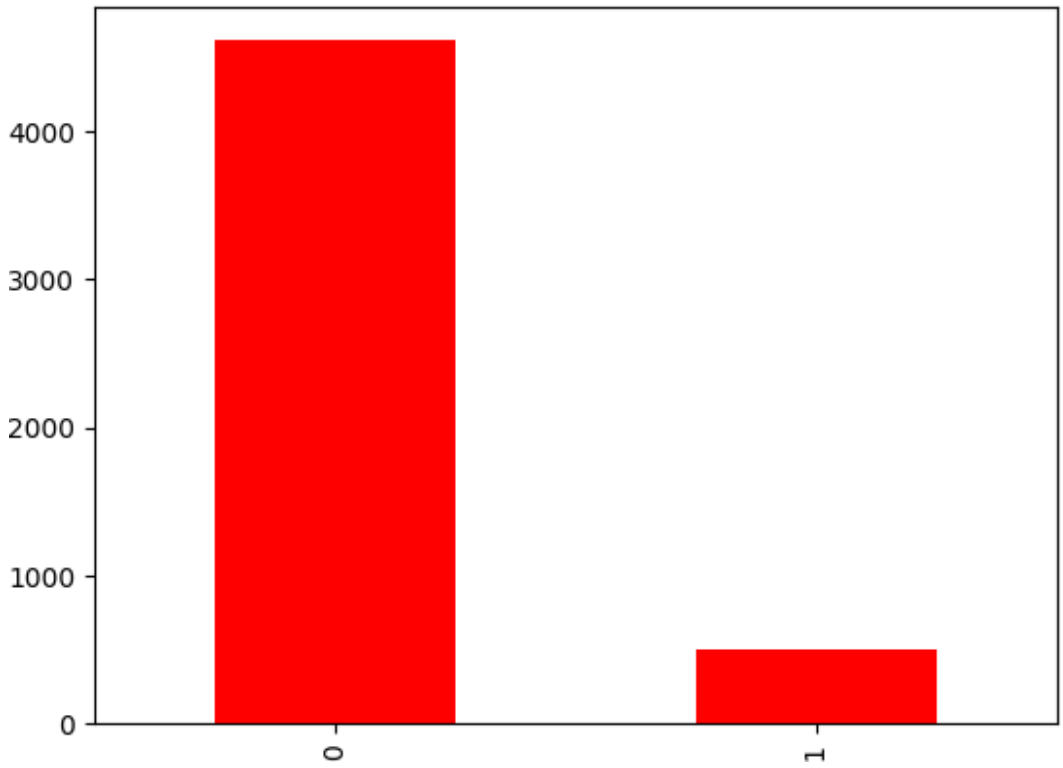
```
% of people who actualy got a stroke :  4.9
```

- Our dataset is highly skewed since only around 5% of the instances got stroke
- We will be needing to perform necessary transformations to improve samples of minority class

Hyper-tension Analysis

```
# Graphical representation of the value counts of the hypertension
attribute
df['hypertension'].value_counts().plot(kind="bar",color = "red")
```

```
<Axes: >
```



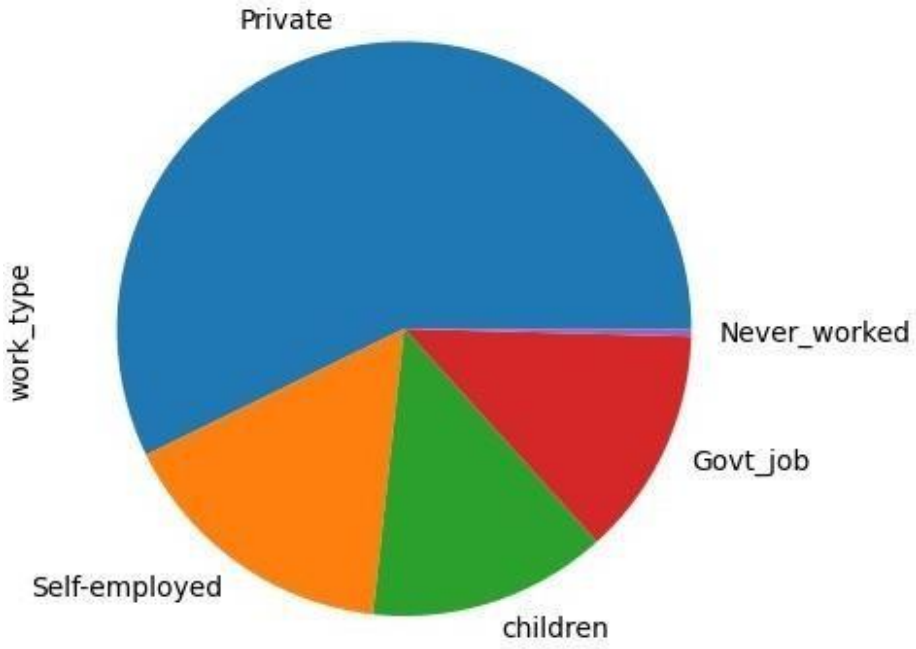
Work type Analysis

```
# Value of count of work-type attribute
df['work_type'].value_counts()
```

```
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

```
# Graphical representation of the value counts of the work-type attribute
df['work_type'].value_counts().plot(kind="pie")
```

```
<Axes: ylabel='work_type'>
```



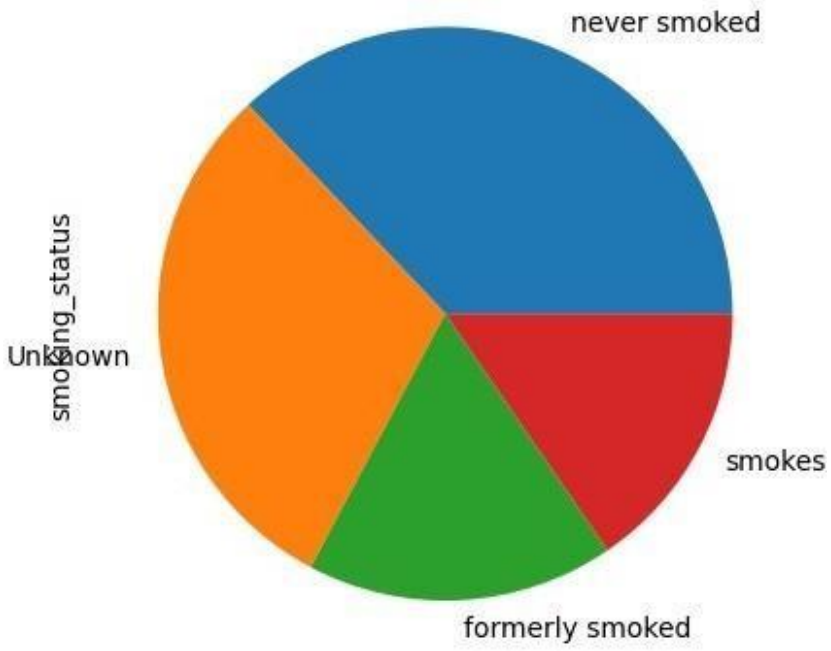
Smoking status Analysis

```
# Value of count of smoking status attribute
df['smoking_status'].value_counts()
```

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

```
# Graphical representation of the value counts of the smoking staus attribute
df['smoking_status'].value_counts().plot(kind="pie")
```

```
<Axes: ylabel='smoking_status'>
```



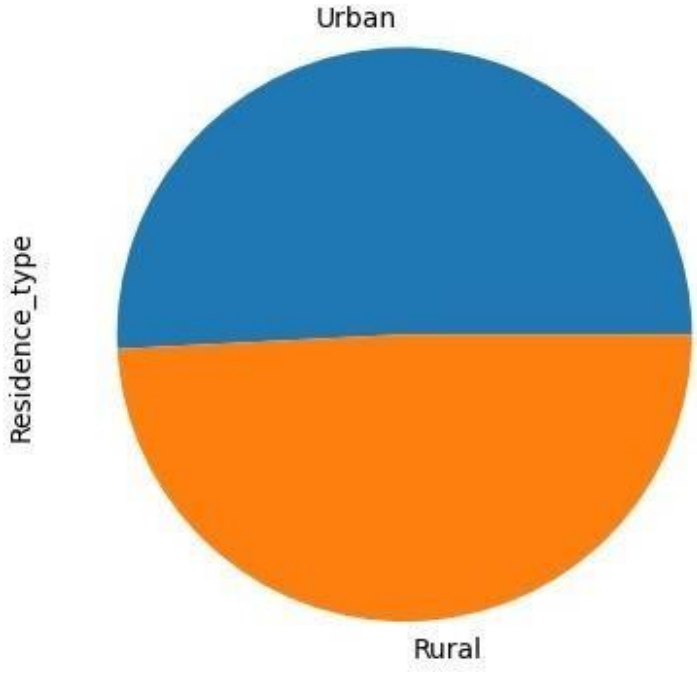
Residence type Analysis

```
# Value of count of residence attribute
df['Residence_type'].value_counts()
```

```
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
```

```
# Graphical representation of the value counts of the residence attribute
df['Residence_type'].value_counts().plot(kind="pie")
```

```
<Axes: ylabel='Residence_type'>
```



- We have an equal percentage of population who are from Urban and rural areas

BMI analysis

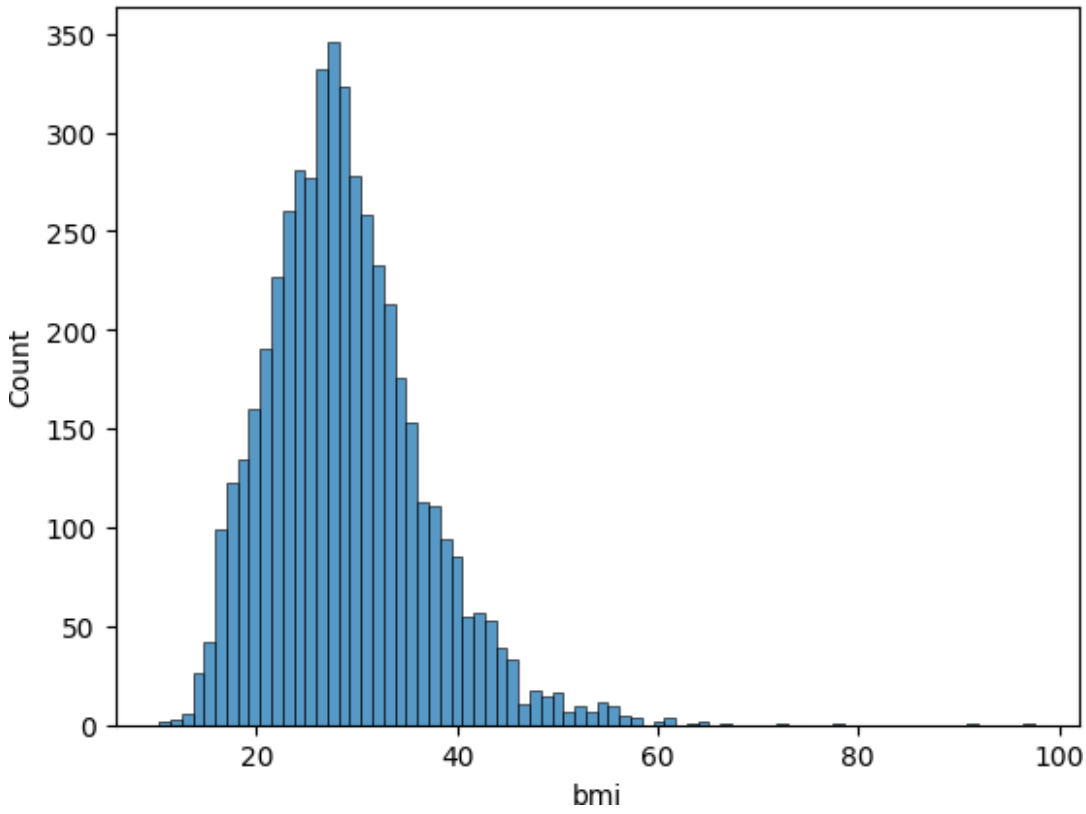
```
# Number of BMI - NULL values
df['bmi'].isnull().sum()
```

```
201
```

- We only have N/A values in bmi column - 201 Null values

```
# Graphical representation of bmi attribute
sns.histplot(data=df['bmi'])
```

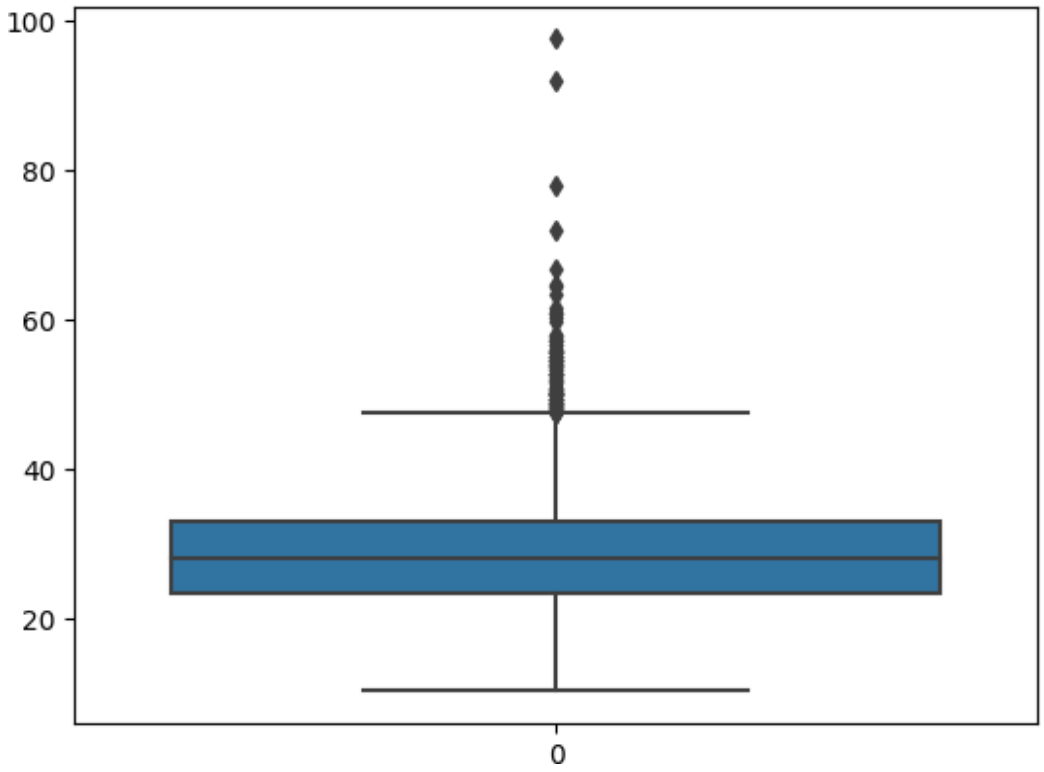
```
<Axes: xlabel='bmi', ylabel='Count'>
```



- Bmi is rightly skewed

```
sns.boxplot(data=df['bmi'])
```

```
<Axes: >
```



- Based on the histogram and boxplot we see that there are many outliers in bmi

```
# Finding the count of outliers based on those instances which are out
of iqr
Q1 = df['bmi'].quantile(0.25)
Q3 = df['bmi'].quantile(0.75)
# Finding IQR
IQR = Q3 - Q1
da=(df['bmi'] < (Q1 - 1.5 * IQR)) | (df['bmi'] > (Q3 + 1.5 * IQR))
da.value_counts()
```

```
False    5000
True      110
Name: bmi, dtype: int64
```

- Total outliers in bmi:110
- Total non-outliers in bmi:5000

```
# Percentage of NULL values in bmi
df['bmi'].isna().sum()/len(df['bmi'])*100
```

```
3.9334637964774952
```

- NULL values hold 3.93 % of the instances in the dataframe

```
df_na=df.loc[df['bmi'].isnull()]
g=df_na['stroke'].sum()
print("People who got stroke and their BMI is NA:",g)
h=df['stroke'].sum()
```

```
print("People who got stroke and their BMI is given:",h)
print("percentage of people with stroke in Nan values to the overall
dataset:",g/h*100)
```

```
People who got stroke and their BMI is NA: 40
People who got stroke and their BMI is given: 249
percentage of people with stroke in Nan values to the overall dataset:
16.06425702811245
```

```
# Percentage of instances who got stroke
df['stroke'].sum()/len(df)*100
```

```
4.87279843444227
```

- Our main target function is stroke And the instances who got a stroke is in the minority - 249 Which is only 4.9 % of the instances

```
# Analysing whether to drop NA values in Bmi column
df_na=df.loc[df['bmi'].isnull()]
print("Nan BMI values where people have
stroke:",df_na['stroke'].sum())
print("overall BMI values where people have
stroke:",df['stroke'].sum())
```

```
Nan BMI values where people have stroke: 40
overall BMI values where people have stroke: 249
```

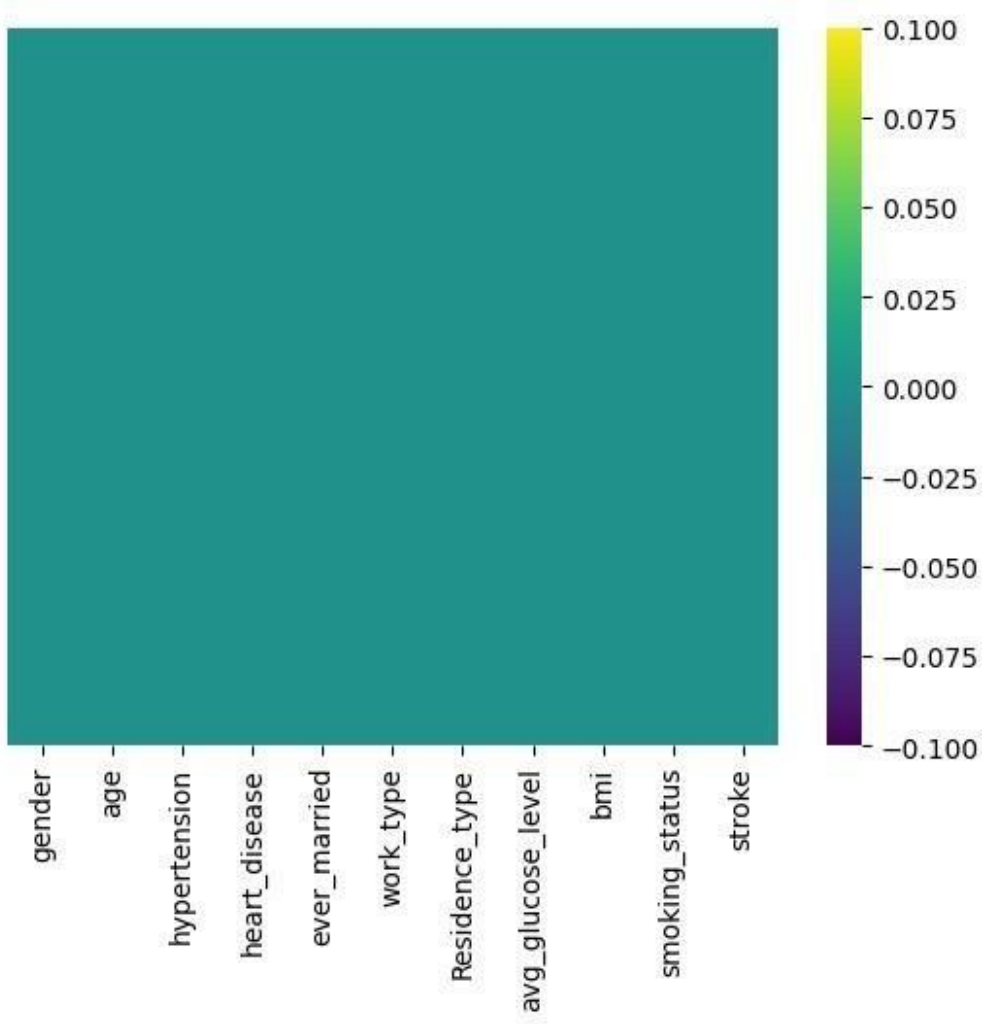
- Among the 201 bmi NULL values 40 values in them got stroke
- Thus we cant drop NULL values
- Since there are outliers present we can't perform mean imputation as mean is affected by the outliers
- Hence we impute it with median values

```
# Imputing the missing N/A values using the median of bmi column
print("median of bmi",df['bmi'].median())
df['bmi']=df['bmi'].fillna(df['bmi'].median())
```

```
median of bmi 28.1
```

```
#PLOTTING THE HEAT MAP AFTER REMOVING NULLL VALUES
sns.heatmap(df.isna(),yticklabels=False,cbar=True,cmap='viridis')
```

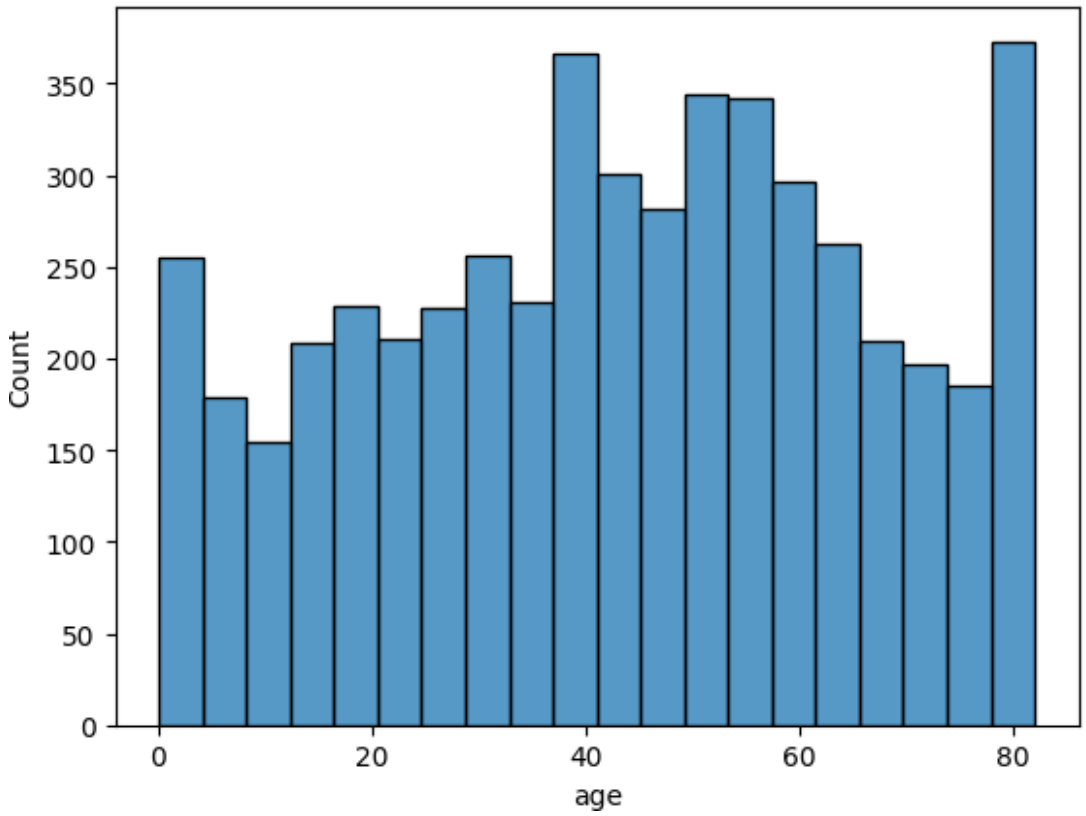
```
<Axes: >
```



AGE analysis

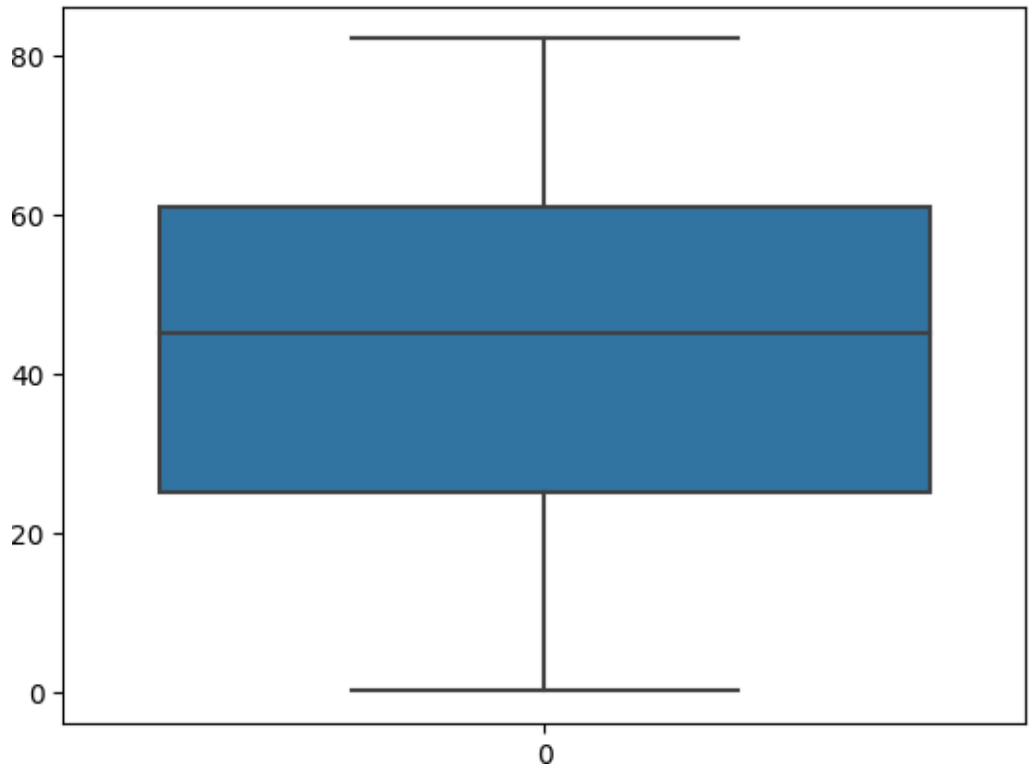
```
# Graphical representation fo the data in age column
# histogram
sns.histplot(data=df['age'])
```

```
<Axes: xlabel='age', ylabel='Count'>
```




```
# boxplot
sns.boxplot(data=df['age'])
```

```
<Axes: >
```

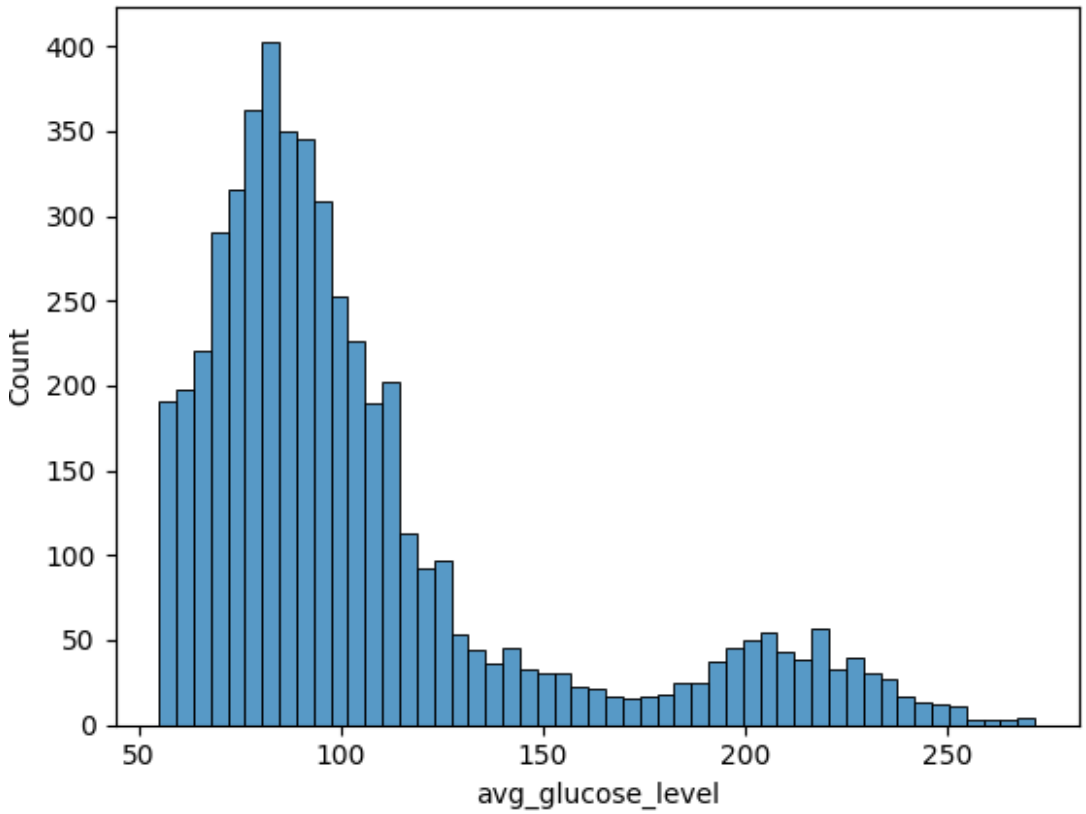


- The age parameter values does not have any outliers
- And has a normal distribution

AVERAGE GLUCOSE LEVEL ANALYSIS

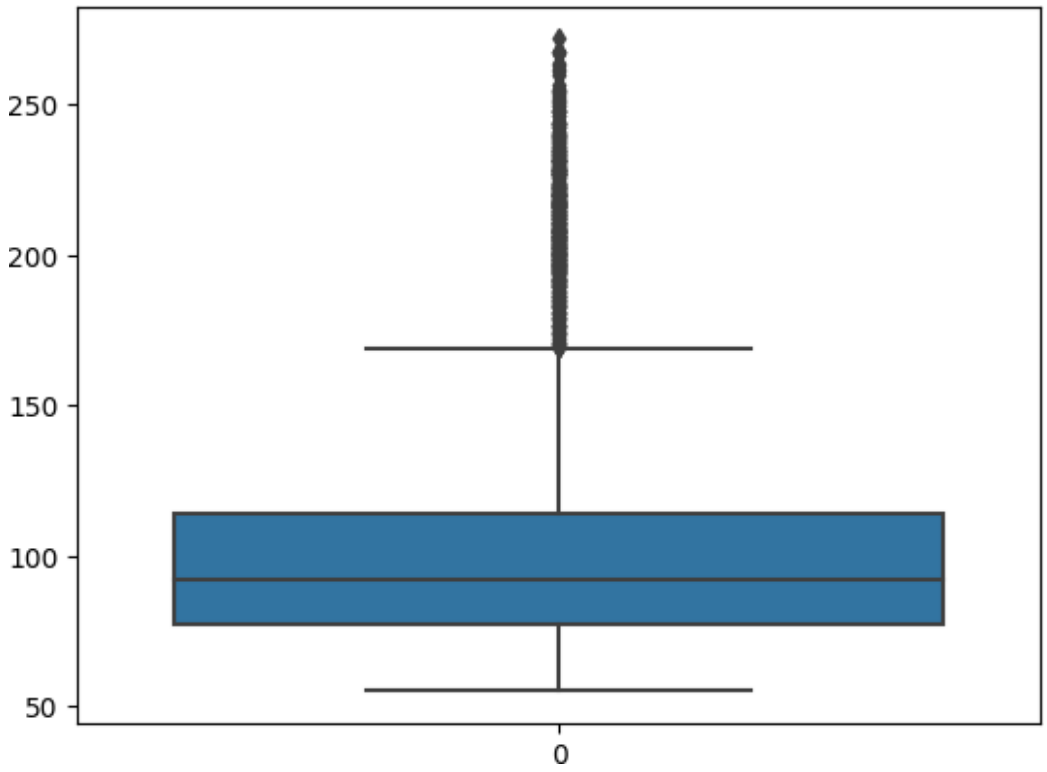
```
# Graphical representation fo the data in glucose level column
# histogram
sns.histplot(data=df['avg_glucose_level'])
```

```
<Axes: xlabel='avg_glucose_level', ylabel='Count'>
```




```
sns.boxplot(data=df['avg_glucose_level'])
```

```
<Axes: >
```



- There are many outliers present based on the boxplot and histogram
- The data is positively skewed

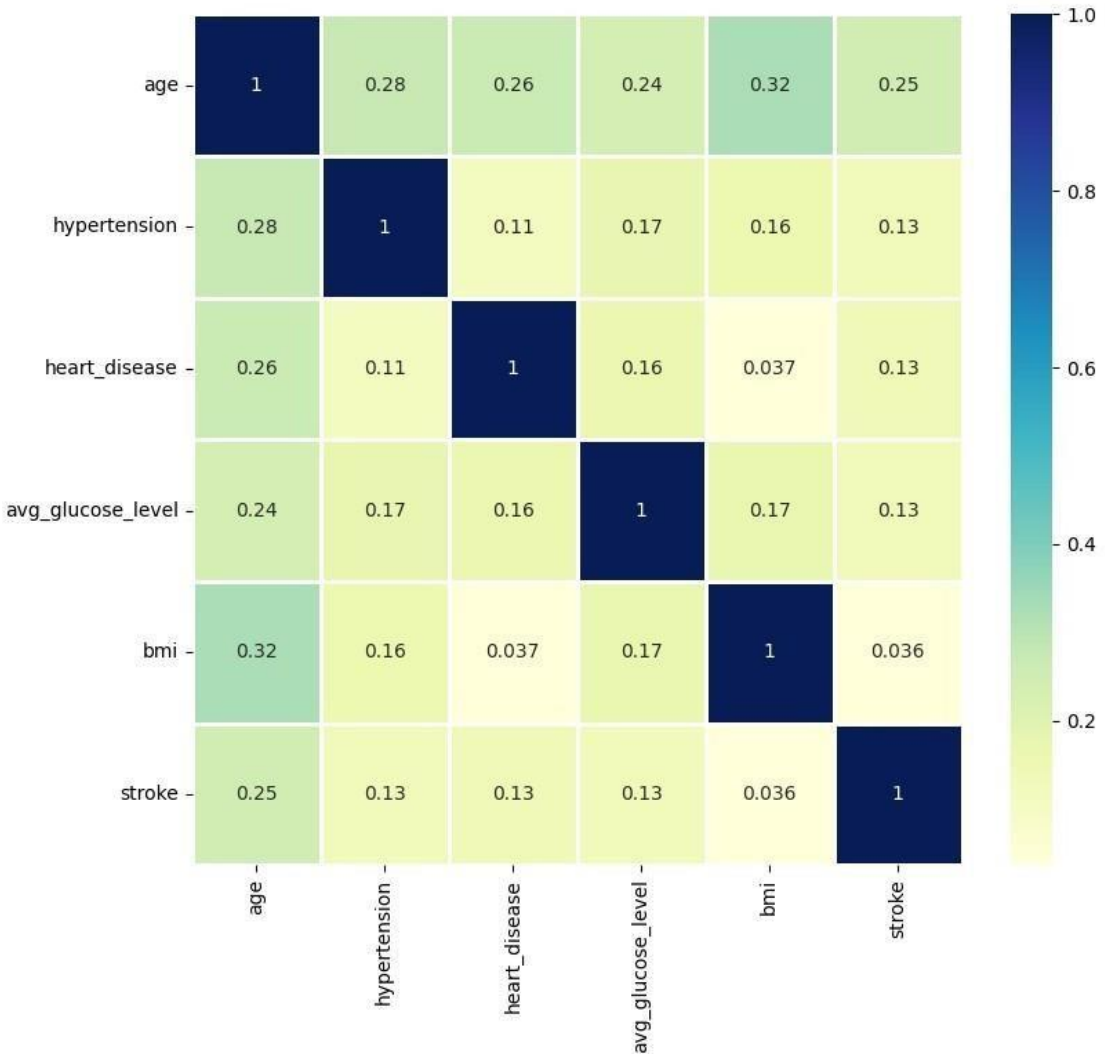
```
# Finding the count of outliers based on those instances which are out
of iqr
Q1 = df['avg_glucose_level'].quantile(0.25)
Q3 = df['avg_glucose_level'].quantile(0.75)
IQR = Q3 - Q1
da=(df['avg_glucose_level'] < (Q1 - 1.5 * IQR)) |
(df['avg_glucose_level'] > (Q3 + 1.5 * IQR))
da.value_counts()
```

```
False    4483
True      627
Name: avg_glucose_level, dtype: int64
```

- Total outliers in avg_glucose_level : 627
- Total non-outliers in avg_glucose_level : 4483

```
# Correlation matrix between the attributes in the dataset to find if
any attributes are correlated
corrmat=df.corr()
f,ax=plt.subplots(figsize=(9,8))
sns.heatmap(corrmat,ax=ax,cmap="YlGnBu",linewidth=0.8,annot=True)
```

```
<Axes: >
```



- There is a weak correlation between the attributes as per the plotted heatmap
- The highest correlation found was between age and bmi - 0.32
- Rest all correlations were less than 0.32
- We could not drw any statistical insight from heatmap

Heart_disease analysis

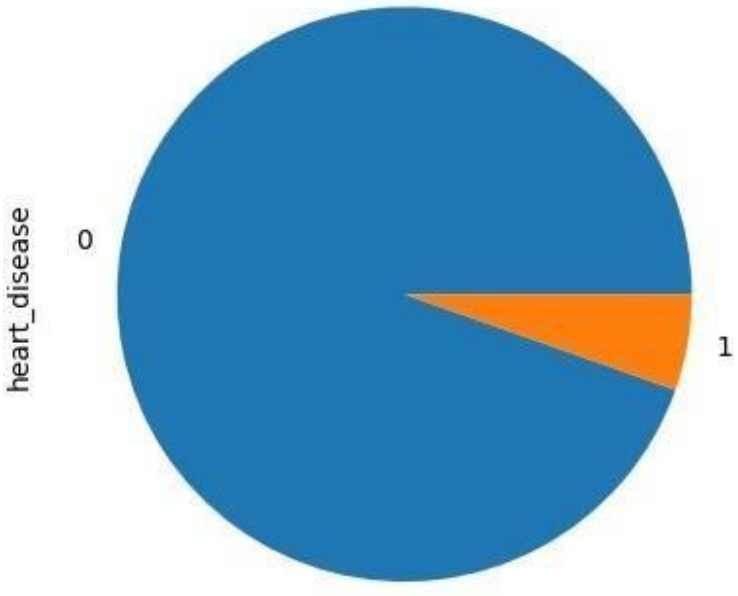
```
# Value count of heart disease attribute
df['heart_disease'].value_counts()
```

```
0    4834
1     276
Name: heart_disease, dtype: int64
```

- This data reflects that around 94.5 % of the total population or list of people are free from Heart_disease and only 6.5 % are having heart_disease.

```
df['heart_disease'].value_counts().plot(kind="pie")
```

```
<Axes: ylabel='heart_disease'>
```



Ever_married analysis with Values

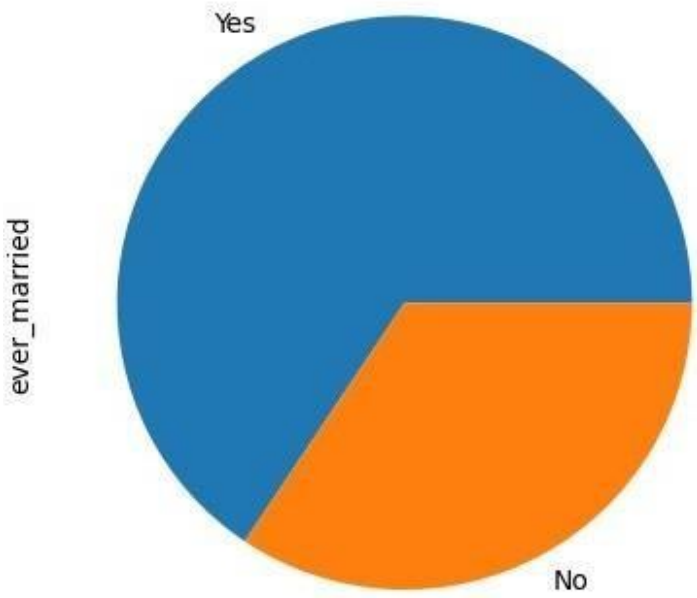
```
# Value count of evver married attribute
df['ever_married'].value_counts()
```

```
Yes     3353
No      1757
Name: ever_married, dtype: int64
```

- This result shows that 65.6 % of people from the list are married and 34.4 % are unmarried.

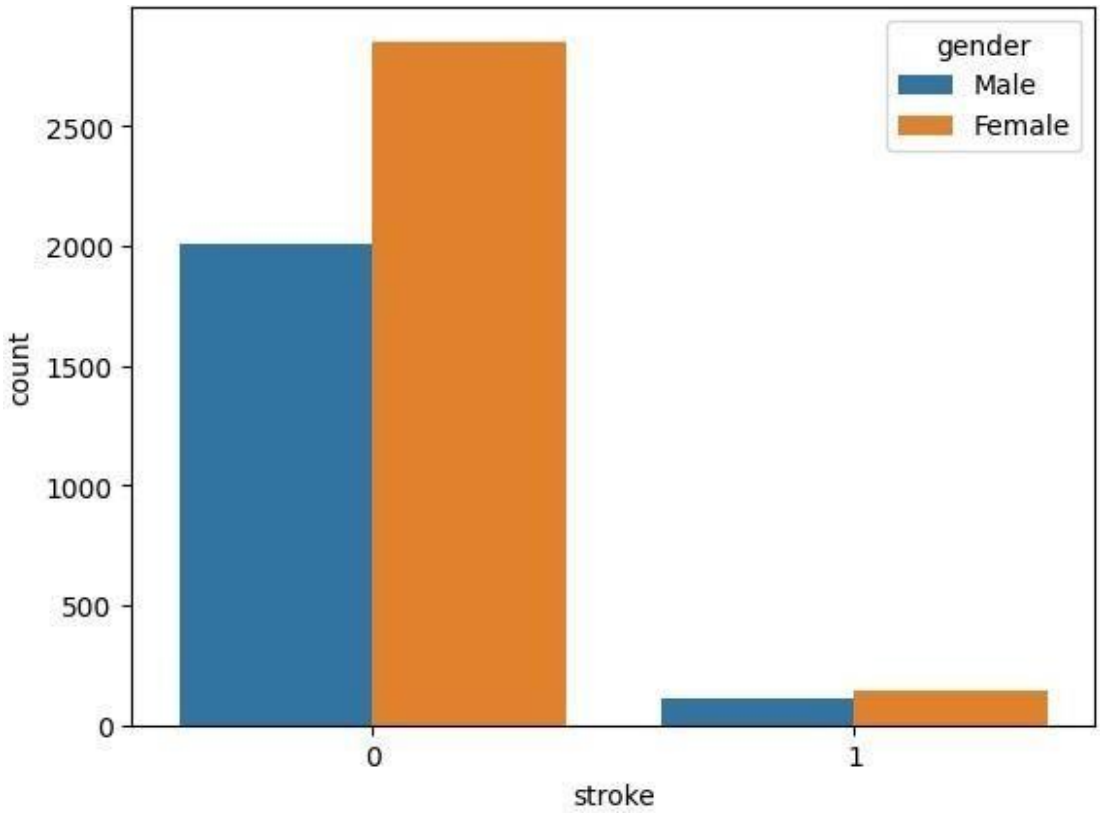
```
#GRAPHICAL REPRESENTATION
df['ever_married'].value_counts().plot(kind="pie")
```

```
<Axes: ylabel='ever_married'>
```




```
# Comparing stroke with gender
sns.countplot(x='stroke', hue='gender', data=df)
```

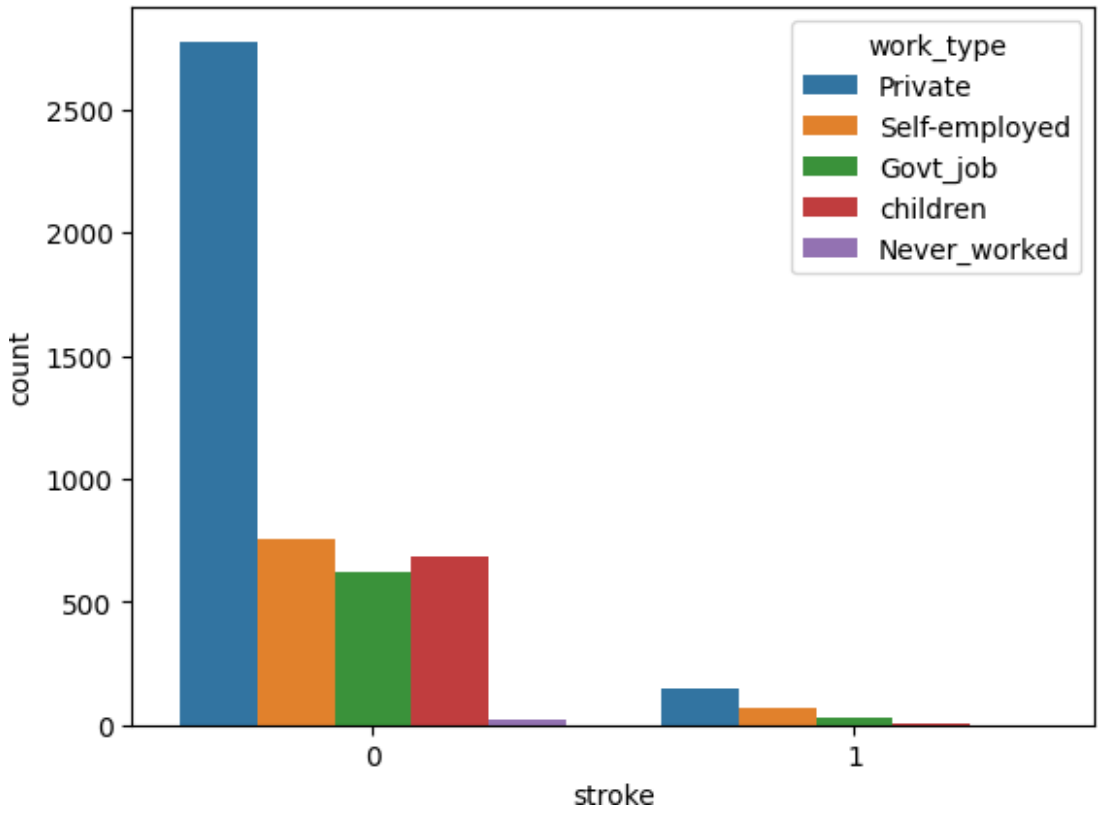
```
<Axes: xlabel='stroke', ylabel='count'>
```



Cross analysis - all the attribute compared with target attibute

```
# Comparing stroke with work-type
sns.countplot(x='stroke', hue='work_type', data=df)
```

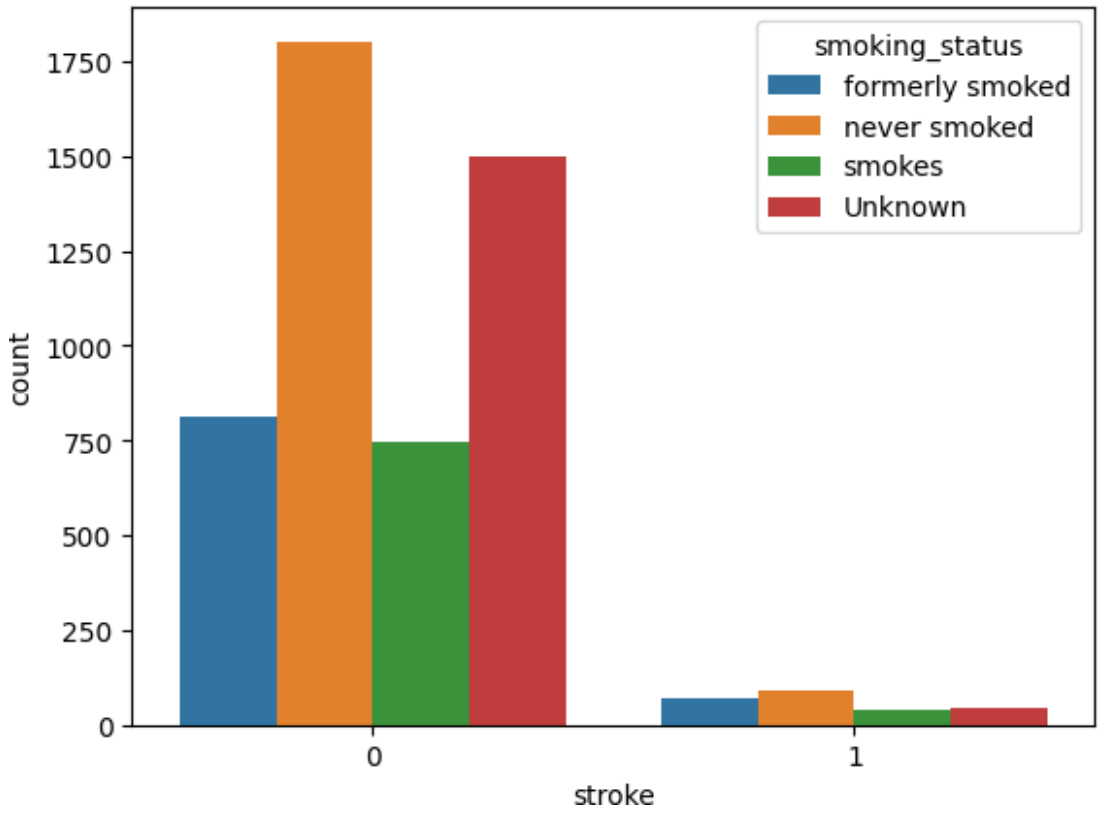
```
<Axes: xlabel='stroke', ylabel='count'>
```



- Based on this comparison we see in the provided dataset that people who never worked never got a heart attack and the people who are privetly employed got more strokes

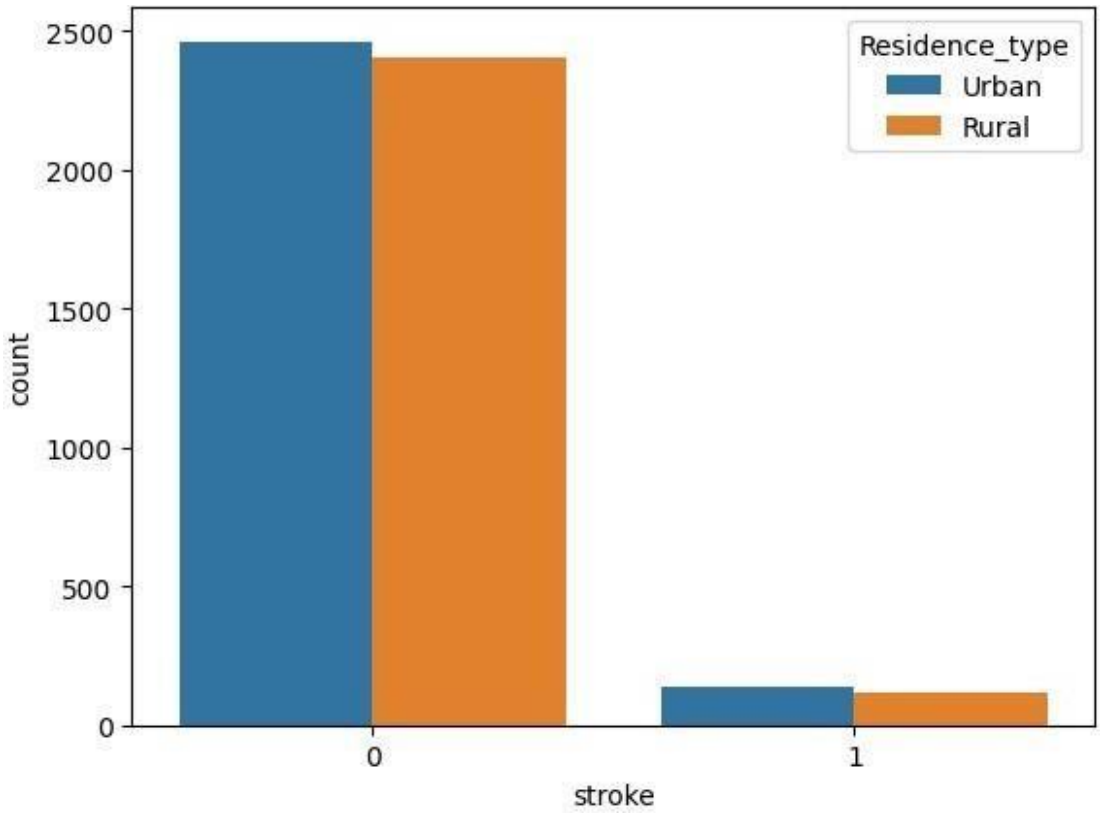
```
# Comparing stroke with somking_status
sns.countplot(x='stroke', hue='smoking_status', data=df)
```

```
<Axes: xlabel='stroke', ylabel='count'>
```



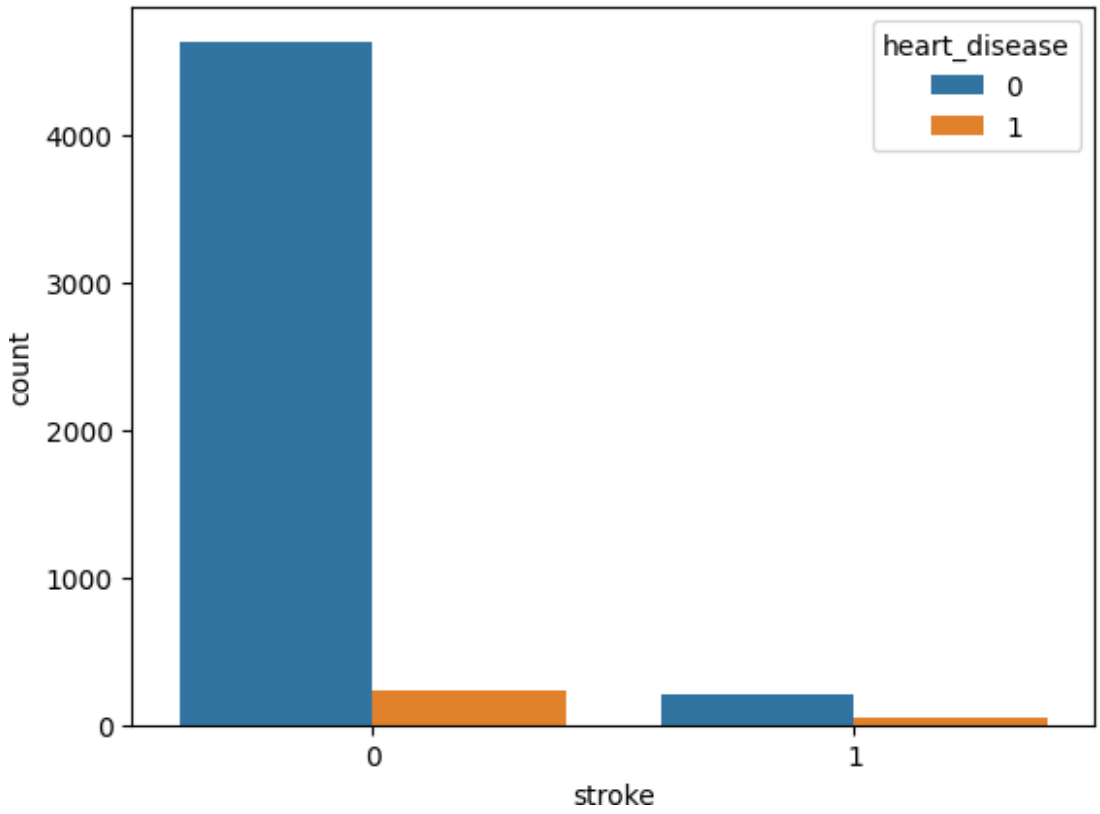

```
# Comparing stroke with residence type
sns.countplot(x='stroke', hue='Residence_type', data=df)
```

```
<Axes: xlabel='stroke', ylabel='count'>
```



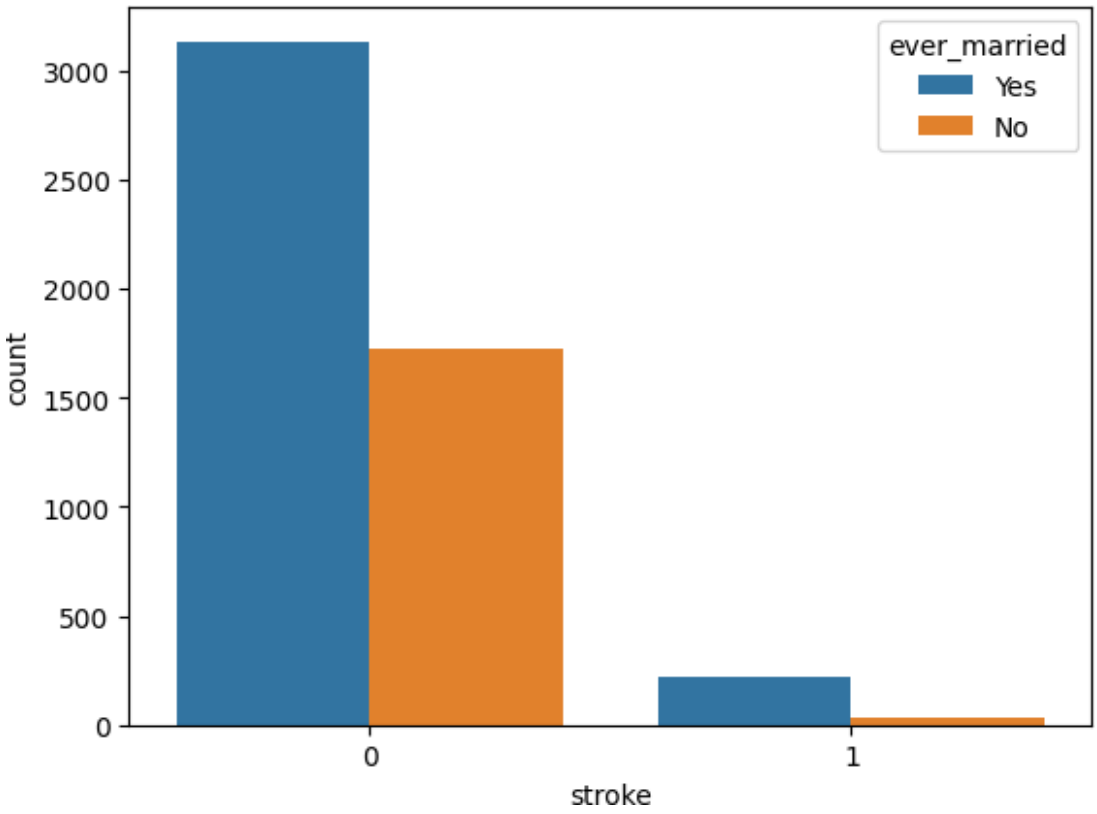

```
# Comparing stroke with heart disease
sns.countplot(x='stroke', hue='heart_disease', data=df)
```

```
<Axes: xlabel='stroke', ylabel='count'>
```




```
# Comparing stroke with married status
sns.countplot(x='stroke', hue='ever_married', data=df)
```

```
<Axes: xlabel='stroke', ylabel='count'>
```



Creating dummy variables for numeric-binary attributes

```
# Converting numeric-binary value attributes to string
df[['hypertension', 'heart_disease', 'stroke']] = df[['hypertension',
'heart_disease', 'stroke']].astype(str)
# Generating dummy attributes - one hot encoding format
df = pd.get_dummies(df, drop_first= True)

# The data frame after performing dummy attributes
df.head()
```

```
    age  avg_glucose_level    bmi  gender_Male  hypertension_1  \
0  67.0             228.69  36.6            1               0
1  61.0             202.21  28.1            0               0
2  80.0             105.92  32.5            1               0
3  49.0             171.23  34.4            0               0
4  79.0             174.12  24.0            0               1

   heart_disease_1  ever_married_Yes  work_type_Never_worked  \
0                1                 1                       0
1                0                 1                       0
2                1                 1                       0
3                0                 1                       0
4                0                 1                       0
```

```
   work_type_Private  work_type_Self-employed  work_type_children  \
0                  1                        0                   0
1                  0                        1                   0
2                  1                        0                   0
3                  1                        0                   0
4                  0                        1                   0

   Residence_type_Urban  smoking_status_formerly smoked  \
0                     1                               1
1                     0                               0
2                     0                               0
3                     1                               0
4                     0                               0

   smoking_status_never smoked  smoking_status_smokes  stroke_1
0                            0                      0         1
1                            1                      0         1
2                            1                      0         1
3                            0                      1         1
4                            1                      0         1
```

```
# Since our Dataset is highly undersampled (based on target instances)
we are going to perform a over sampling method to have equal
representation of both the target classes
# Using random oversampling - importing the library
from imblearn.over_sampling import RandomOverSampler

# Performing a minority oversampling
oversample = RandomOverSampler(sampling_strategy='minority')
X=df.drop(['stroke_1'],axis=1)
y=df['stroke_1']

# Obtaining the oversampled dataframes - testing and training
X_over, y_over = oversample.fit_resample(X, y)
```

```
import pandas as pd
from scipy.stats import zscore

# Select the numerical columns
num_cols = df.select_dtypes(include='number').columns.tolist()

# Calculate the z-scores for the numerical columns
df[num_cols] = df[num_cols].apply(zscore)

# Set the threshold for the z-scores
threshold = 3

# Remove the rows containing outliers
df = df[(df[num_cols] < threshold).all(axis=1)]
```

Creating test-train split (80-20 split)

```
# creating dataset split for training and testing the model
from sklearn.model_selection import train_test_split
# Performing a 80-20 test-train split
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over,
test_size= 0.20, random_state= 42)

# Checking the size of the splits
print('X_train:', X_train.shape)
print('y_train:', y_train.shape)
print('X_test:', X_test.shape)
print('y_test:', y_test.shape)
```

```
X_train: (7777, 15)
y_train: (7777,)
X_test: (1945, 15)
y_test: (1945,)
```

Training Model

Decision Tree

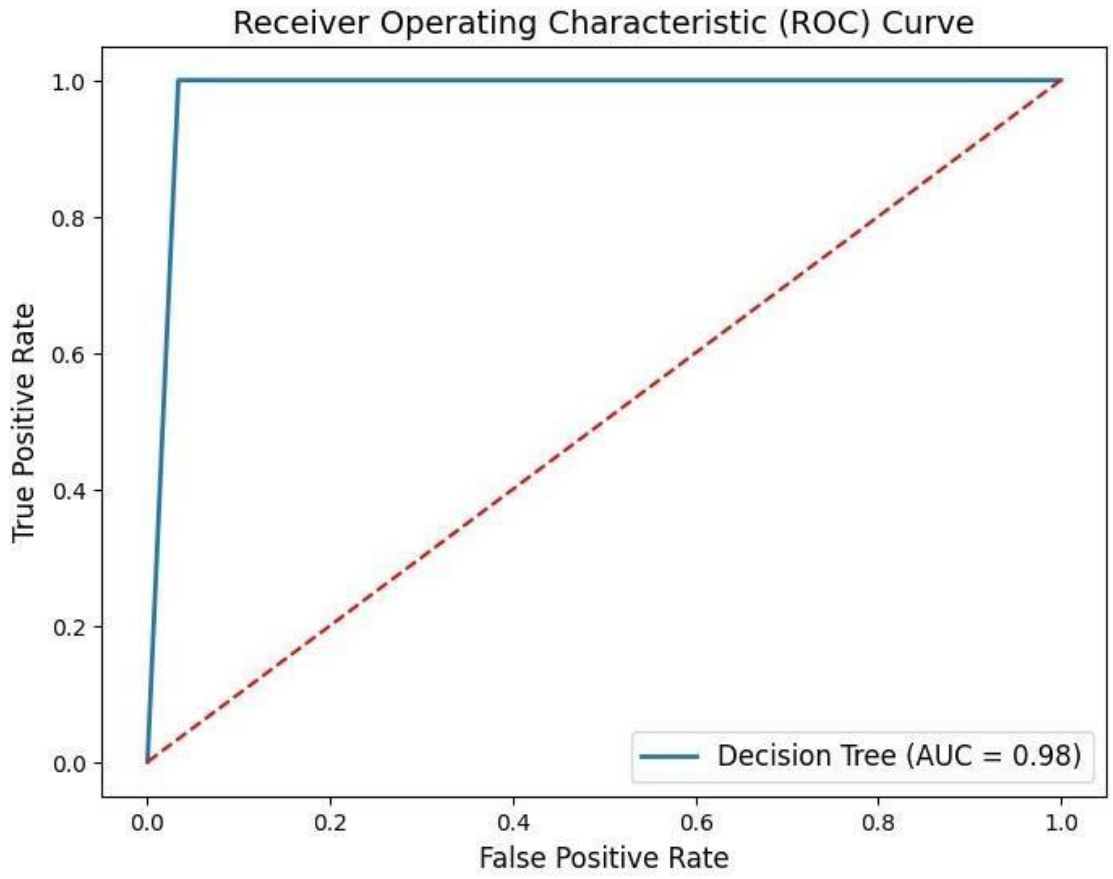
```
#importing the Decision Tree Classifier module
from sklearn.tree import DecisionTreeClassifier
# Libraries for calculating performance metrics
from sklearn import metrics
from sklearn.metrics import
auc,roc_auc_score,roc_curve,precision_score,recall_score,f1_score

# Create the classifier object
clf = DecisionTreeClassifier()

# Training the classifier
clf = clf.fit(X_train,y_train)

#predicting result using the test dataset
y_pred = clf.predict(X_test)

# Printing the accuracyof the model
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9830334190231362
```

```
#CLASSIFICATION REPORT OF DECISION TREE
from sklearn.metrics import
classification_report,accuracy_score,confusion_matrix
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

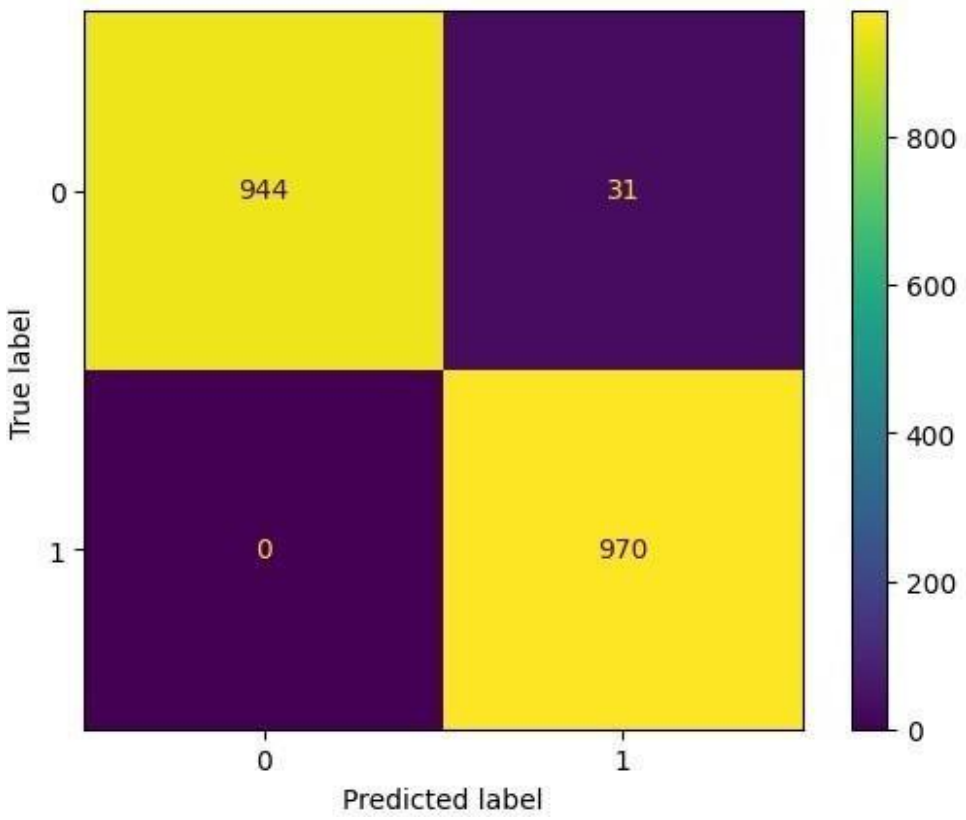
```
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       975
           1       0.97      1.00      0.98       970

    accuracy                           0.98      1945
   macro avg       0.98      0.98      0.98      1945
weighted avg       0.98      0.98      0.98      1945
```

```
#PLOTTING THE ROC CURVE
# Calculate the probabilities of the positive class
y_pred_prob_dt = clf.predict_proba(X_test)[:, 1]

# Calculate the false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_dt)

# Calculate the area under the ROC curve
auc_score = roc_auc_score(y_test, y_pred_prob_dt)

# Plot the ROC curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, linewidth=2, label=f'Decision Tree (AUC =
{auc_score:.2f})')
plt.plot([0, 1], [0, 1], 'r--')
plt.title('Receiver Operating Characteristic (ROC) Curve',
fontsize=14)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.legend(loc='lower right', fontsize=12)
plt.show()
```




```
#CONFUSION MATRIX FOR DECISION TREE
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Create the classifier object
clf = DecisionTreeClassifier()

# Training the classifier
clf = clf.fit(X_train, y_train)

#predicting result using the test dataset
y_pred = clf.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fa2867aea00>
```




```
ac_dt = metrics.accuracy_score(y_test, y_pred)

ac_dt

0.9840616966580977
```

KNN

```
#importing the KNN Classifier module
from sklearn.neighbors import KNeighborsClassifier
# Libraries for calculating performance metrics
from sklearn.metrics import
classification_report,accuracy_score,confusion_matrix
from sklearn.metrics import
auc,roc_auc_score,roc_curve,precision_score,recall_score,f1_score

# Create the classifier object
# 2 neighbours because of the 2 classes
knn = KNeighborsClassifier(n_neighbors = 2)
# Training the classifier
knn.fit(X_train,y_train)
#predicting result using the test dataset
y_pred_knn = knn.predict(X_test)
y_pred_prob_knn = knn.predict_proba(X_test)[:, 1]

# Printing the accuracy and roc-auc score of the model
```

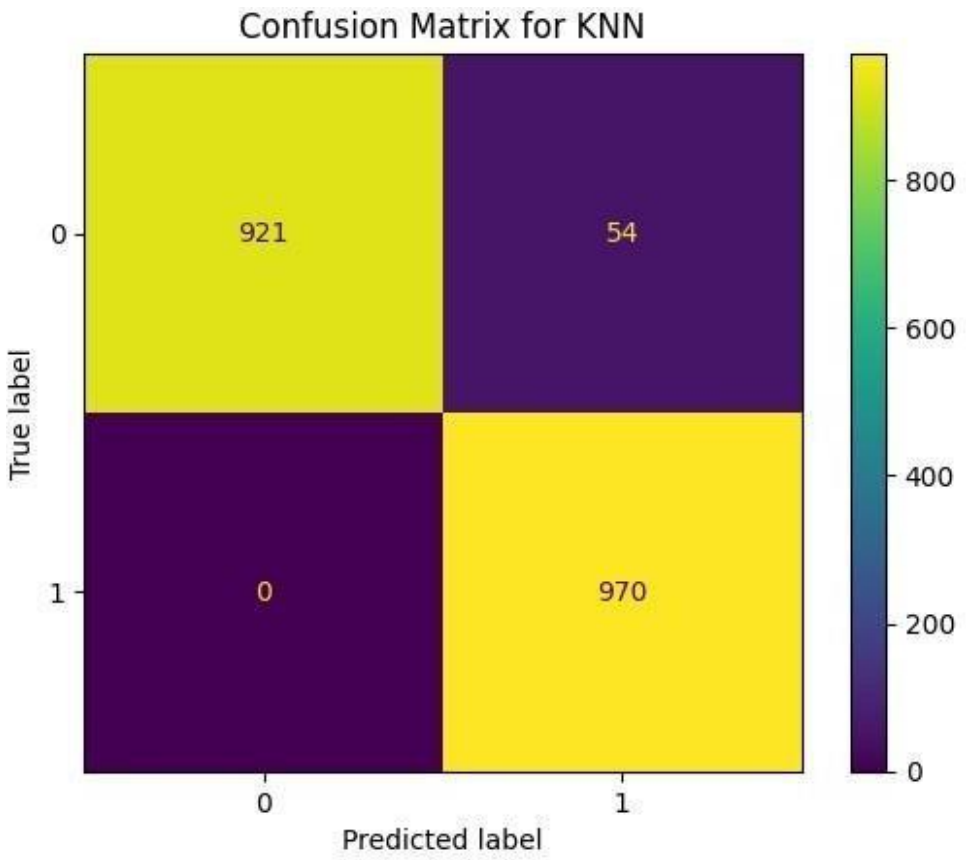
```
confusion_matrix(y_test, y_pred_knn)
print('Accuracy:',accuracy_score(y_test, y_pred_knn))
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_knn))
```

```
Accuracy: 0.9722365038560411
ROC AUC Score: 0.9723076923076923
```

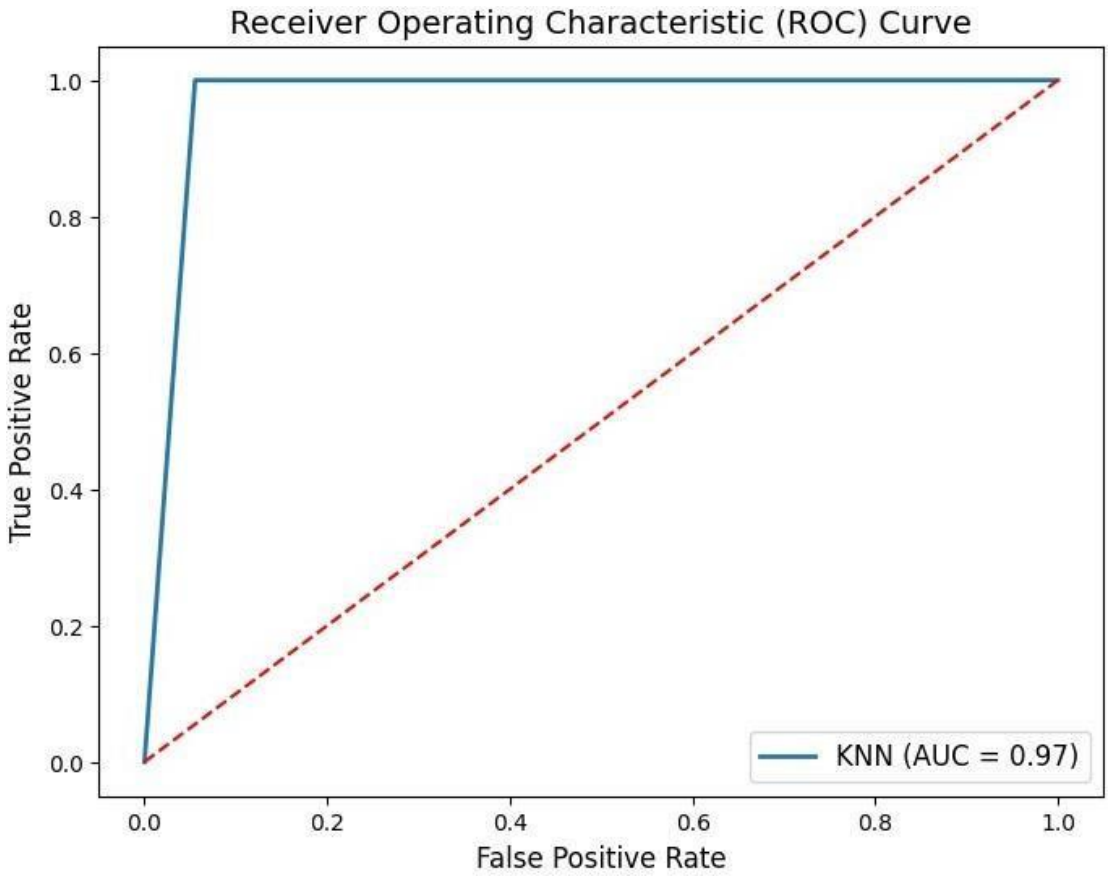
```
#CLASSIFICATION REPORT FOR KNN
y_pred_knn = knn.predict(X_test)
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       975
           1       0.95      1.00      0.97       970

    accuracy                           0.97      1945
   macro avg       0.97      0.97      0.97      1945
weighted avg       0.97      0.97      0.97      1945
```

```
#CONFUSION MATRIX FOR KNN
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred_knn)
cmd = ConfusionMatrixDisplay(cm, display_labels=['0', '1'])
cmd.plot()
plt.title("Confusion Matrix for KNN")
plt.show()
```




```
#PLOTTING THE ROC CURVE
# Calculate the probabilities of the positive class
y_pred_prob_knn = knn.predict_proba(X_test)[:, 1]

# Calculate the false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_knn)

# Calculate the area under the ROC curve
auc_score = roc_auc_score(y_test, y_pred_prob_knn)

# Plot the ROC curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, linewidth=2, label=f'KNN (AUC = {auc_score:.2f})')
plt.plot([0, 1], [0, 1], 'r--')
plt.title('Receiver Operating Characteristic (ROC) Curve',
fontsize=14)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.legend(loc='lower right', fontsize=12)
plt.show()
```




```
ac_knn=accuracy_score(y_test, y_pred_knn)
```

```
ac_knn
```

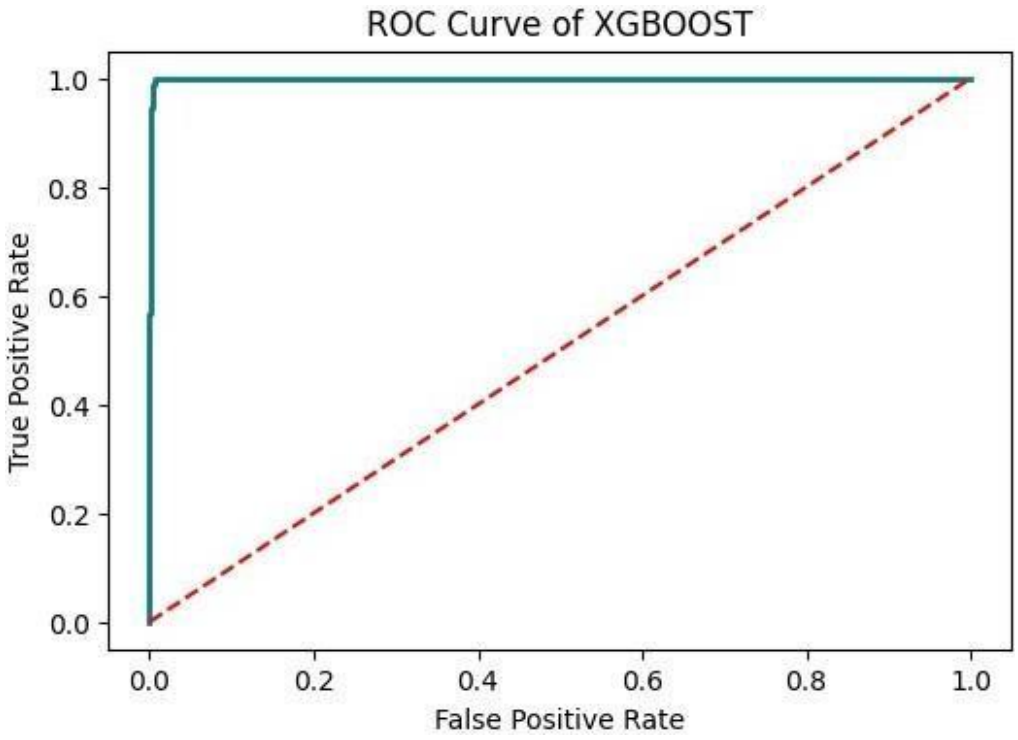
```
0.9722365038560411
```

XGBOOST

```
#importing the XGBoost Classifier module and plotting roc curve
from xgboost  import XGBClassifier

# Create the classifier object
xgb = XGBClassifier()
# Training the classifier
xgb.fit(X_train,y_train)
#predicting result using the test dataset
y_pred_xgb = xgb.predict(X_test)
y_pred_prob_xgb = xgb.predict_proba(X_test)[:, 1]

# Printing the accuracy and roc-auc score of the model
print('Accuracy:', accuracy_score(y_test, y_pred_xgb))
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_xgb))

# plots of roc_auc
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_xgb)

plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, linewidth=2, color= 'teal')
plt.plot([0,1], [0,1], 'r--' )
plt.title('ROC Curve of XGBOOST')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```

```
Accuracy: 0.9773778920308483
ROC AUC Score: 0.9993782712133227
```

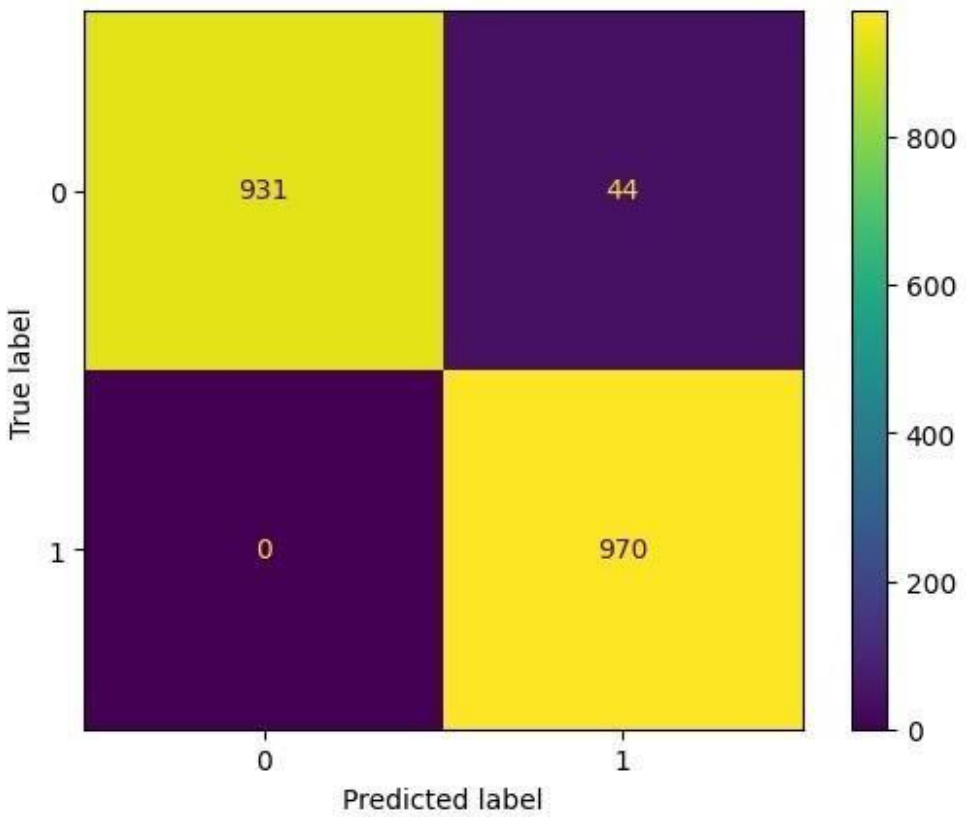



```
ac_xgboost = accuracy_score(y_test, y_pred_xgb)

ac_xgboost
```

```
0.9773778920308483
```

```
# Plotting the confusion matrix for xgboost
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(xgb,X_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fa287204dc0>
```




```
#CLASSIFICATION REPORT FOR XGBOOST
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.98       975
           1       0.96      1.00      0.98       970

    accuracy                           0.98      1945
   macro avg       0.98      0.98      0.98      1945
weighted avg       0.98      0.98      0.98      1945
```

RANDOM FOREST

```
# importing random forest classifier module for training
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Create the classifier object
rf_clf = RandomForestClassifier(n_estimators = 100)

# Train the model using the training sets
```

```
rf_clf.fit(X_train, y_train)

# performing predictions on the test dataset
y_pred_rf = rf_clf.predict(X_test)

# Printing accuracy of the model
print('Accuracy:', accuracy_score(y_test, y_pred_rf))
```

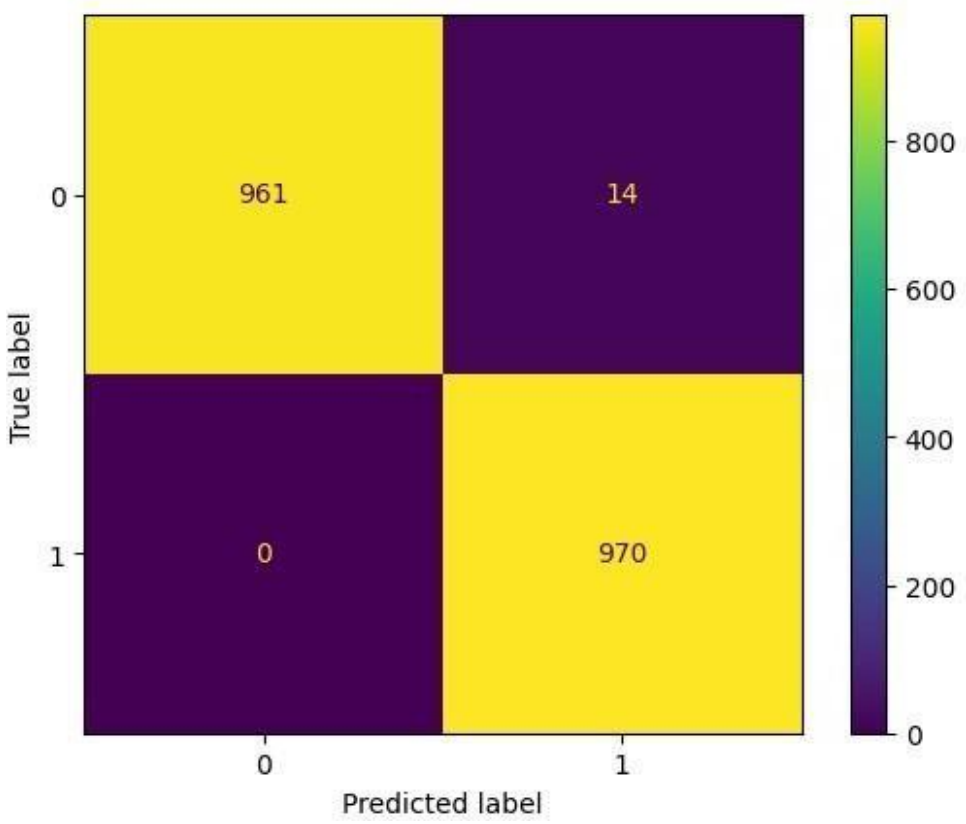
```
Accuracy: 0.9943444730077121
```

```
#CLASSIFICATION REPORT FOR RANDOM FOREST
from sklearn.metrics import classification_report

# Print classification report
print("Classification Report:\n", classification_report(y_test,
y_pred_rf))
```

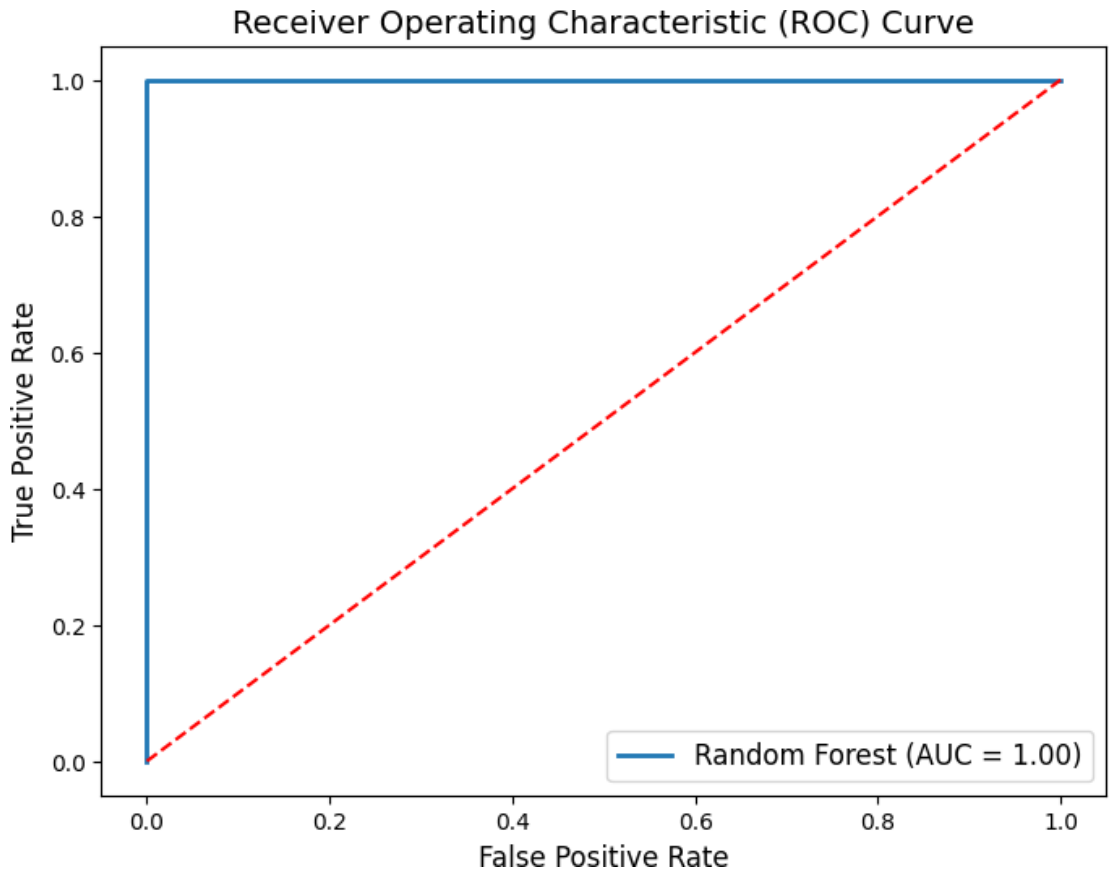
```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99       975
           1       0.99      1.00      0.99       970

    accuracy                           0.99      1945
   macro avg       0.99      0.99      0.99      1945
weighted avg       0.99      0.99      0.99      1945
```

```
#CONFUSION MATRIX FOR RANDOM FOREST

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Create the classifier object
rf_clf = RandomForestClassifier(n_estimators=100)

# Train the model using the training sets
rf_clf.fit(X_train, y_train)

# performing predictions on the test dataset
y_pred_rf = rf_clf.predict(X_test)

cm = confusion_matrix(y_test, y_pred_rf)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fa285d93a00>
```




```
# import necessary libraries
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# predict probabilities for the test dataset
y_pred_prob_rf = rf_clf.predict_proba(X_test)[:, 1]

# calculate the ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_rf)
auc_score = roc_auc_score(y_test, y_pred_prob_rf)

# plot the ROC curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, linewidth=2, label=f'Random Forest (AUC =
{auc_score:.2f})')
plt.plot([0, 1], [0, 1], 'r--')
plt.title('Receiver Operating Characteristic (ROC) Curve',
fontsize=14)
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.legend(loc='lower right', fontsize=12)
plt.show()
```




```
ac_rf=accuracy_score(y_test, y_pred_rf)

ac_rf

0.9928020565552699
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

y_pred_lr = classifier.predict(X_test)

confusion_matrix(y_test, y_pred_lr)
print('Accuracy:', accuracy_score(y_test, y_pred_lr))

Accuracy: 0.7737789203084833

/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

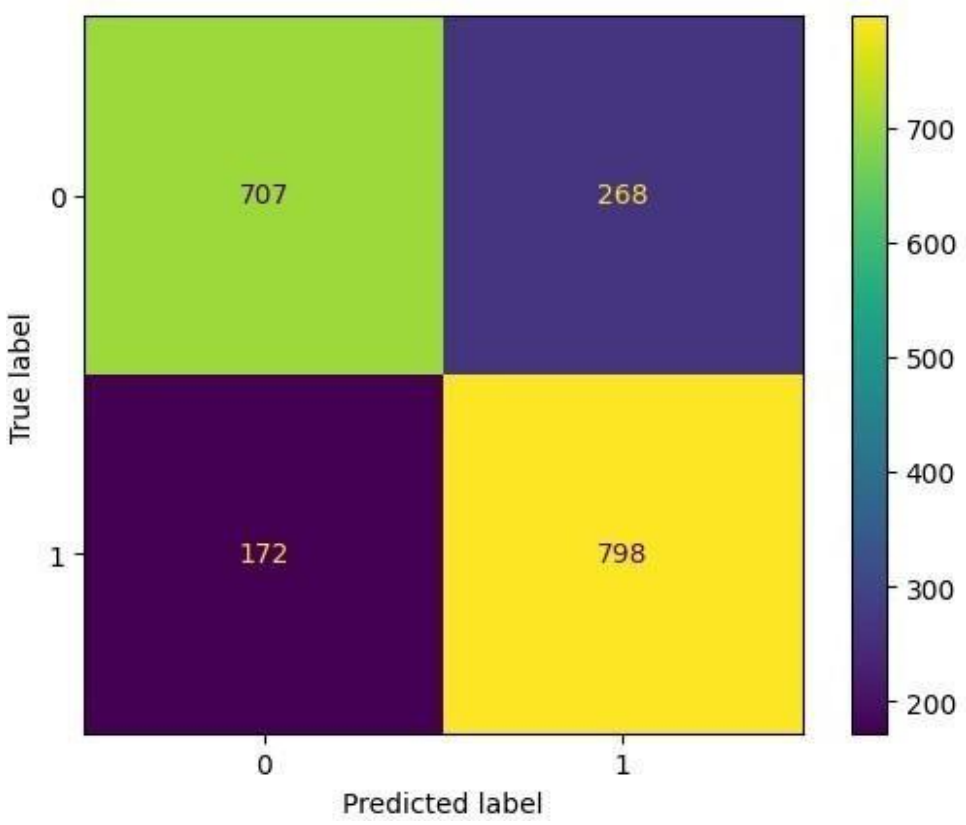
```
#CLASSIFICATION REPORT OF LOGISTIC
from sklearn.metrics import classification_report

# Print classification report
print("Classification Report:\n", classification_report(y_test,
y_pred_rf))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       975
           1       0.99      1.00      0.99       970

    accuracy                           0.99      1945
   macro avg       0.99      0.99      0.99      1945
weighted avg       0.99      0.99      0.99      1945
```

```
#CONFUSION MATRIX FOR LOGISTIC
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

y_pred_lr = classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred_lr)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

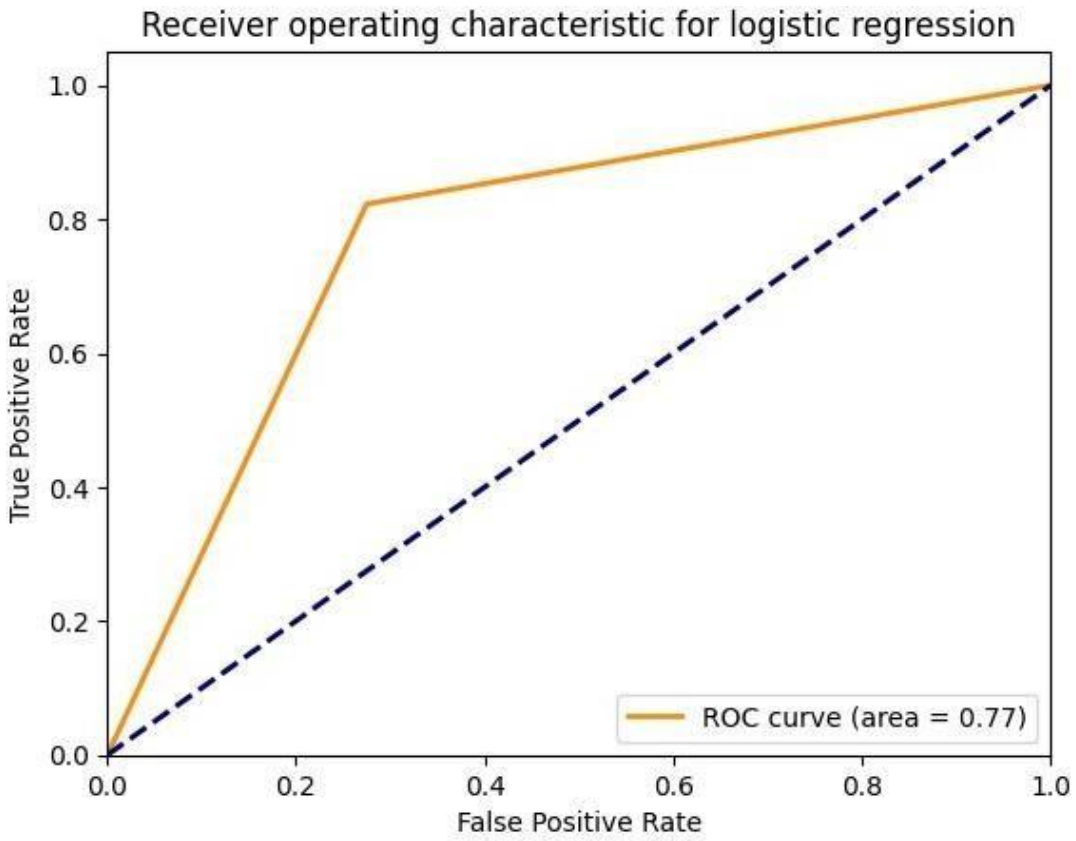
```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fa286c5c940>
```



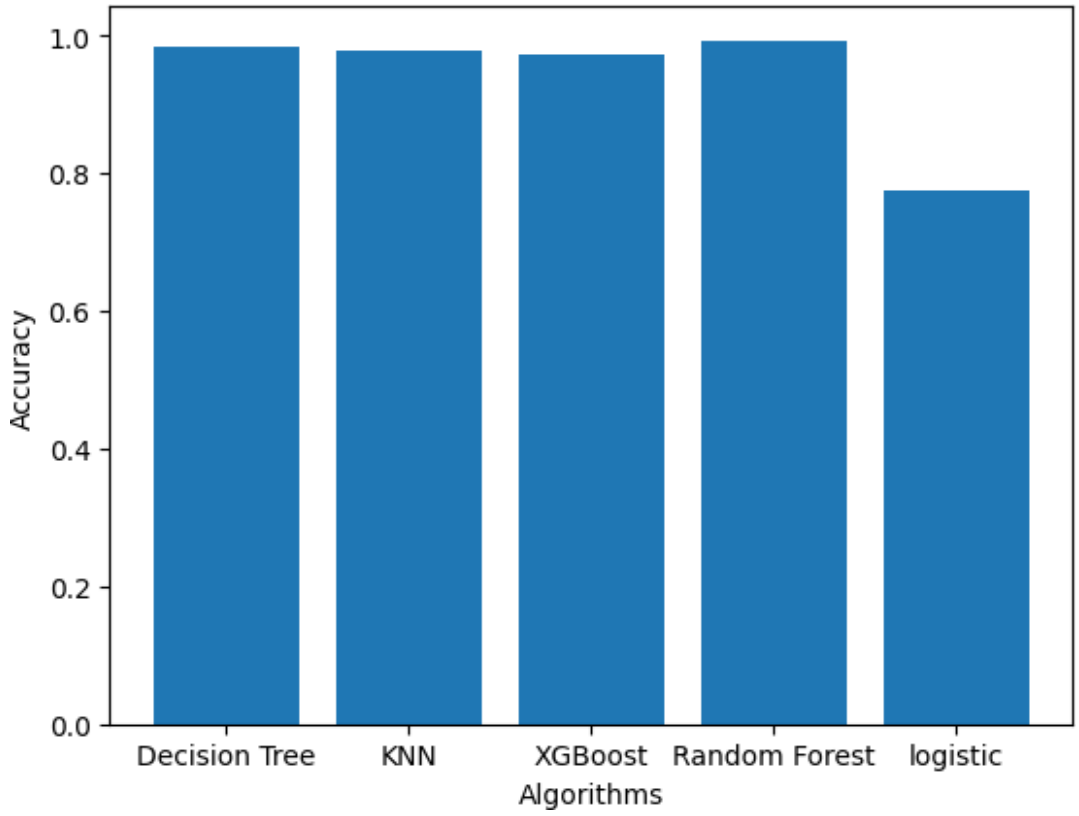

```
from sklearn.metrics import roc_curve, auc

# Train the logistic regression model
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lr = classifier.predict(X_test)

# Calculate the fpr and tpr for all thresholds of the classification
fpr, tpr, thresholds = roc_curve(y_test, y_pred_lr)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
```

```
%0.2f)' % 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 for logistic regression')
plt.legend(loc="lower right")
plt.show()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```



```
ac_1r = accuracy_score(y_test, y_pred_lr)

ac_1r

0.7737789203084833

#graph accuracy
plt.bar(['Decision Tree','KNN', 'XGBoost', 'Random
Forest','logistic'],[ac_dt,ac_xgboost,ac_knn,ac_rf,ac_1r])
plt.xlabel("Algorithms")
plt.ylabel("Accuracy")
plt.show

<function matplotlib.pyplot.show(close=None, block=None)>
```




```
# Making sample predictions based on manual value entry
age=75
avg_glucose_level=300
bmi=36.6
gender_Male=1
ever_married_Yes=1
work_type_Never_worked=0
work_type_Private=1
work_type_Self_employed=0
work_type_children=0
Residence_type_Urban=1
```

```
smoking_status_formerly_smoked=1
smoking_status_never_smoked=0
smoking_status_smokes=0
hypertension_1=1
heart_disease_1=1
input_features = [age ,avg_glucose_level,    bmi
      ,gender_Male,hypertension_1,
      heart_disease_1,ever_married_Yes,      work_type_Never_worked,
      work_type_Private,    work_type_Self_employed,
      work_type_children    ,Residence_type_Urban,
      smoking_status_formerly_smoked,smoking_status_never_smoked
      ,smoking_status_smokes]

features_value = [np.array(input_features)]
features_name = ['age'      ,'avg_glucose_level', 'bmi' ,'gender_Male'
      ,'hypertension_1',    'heart_disease_1','ever_married_Yes', 'work
_type_Never_worked', 'work_type_Private', 'work_type_Self-employed',
      'work_type_children'  ,'Residence_type_Urban',    'smoking_status_
formerly smoked','smoking_status_never smoked'
      ,'smoking_status_smokes']

df = pd.DataFrame(features_value, columns=features_name)
prediction = rf_clf.predict(df)[0]
print(prediction)
```

```
1
```

TESTING

```
import pandas as pd
Data = pd.read_csv("/content/2testhealthcare-dataset-stroke-data
test.csv")

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

Data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1023 entries, 0 to 1022
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 1023 non-null   int64
 1   gender             1023 non-null   object
 2   age                1023 non-null   float64
 3   hypertension       1023 non-null   int64
 4   heart_disease      1023 non-null   int64
 5   ever_married       1023 non-null   object
 6   work_type          1023 non-null   object
 7   Residence_type     1023 non-null   object
 8   avg_glucose_level  1023 non-null   float64
 9   bmi                963 non-null    float64
 10  smoking_status     1023 non-null   object
 11  stroke             1023 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 96.0+ KB

Data.describe()

                 id          age hypertension   heart_disease  \
count   1023.000000  1023.000000   1023.000000    1023.000000
mean   35791.105572    44.746276      0.117302       0.066471
std    20867.793866    22.803867      0.321937       0.249226
min      132.000000     0.080000      0.000000       0.000000
25%    17078.500000    27.000000      0.000000       0.000000
50%    36226.000000    47.000000      0.000000       0.000000
75%    52759.500000    62.000000      0.000000       0.000000
max    72861.000000    82.000000      1.000000       1.000000

       avg_glucose_level         bmi       stroke
count        1023.000000  963.000000  1023.000000
mean          109.166207   28.980789     0.099707
std            48.806266    7.673518     0.299755
min            55.220000   10.300000     0.000000
25%            78.050000   23.600000     0.000000
```

```
50%            92.620000   28.300000     0.000000
75%           117.540000   33.100000     0.000000
max           267.760000   66.800000     1.000000
```

```
# load the data into a pandas DataFrame
Data = pd.read_csv("/content/2testhealthcare-dataset-stroke-data
test.csv")
Data = Data.drop(['id'],axis=1)
# replace missing values with NaN
Data.replace(" ", np.nan, inplace=True)

# fill missing values in numerical columns with mean
Data.fillna(Data.select_dtypes(include=np.number).mean(),
inplace=True)

# fill missing values in categorical columns with mode
Data.fillna(Data.select_dtypes(include=object).mode().iloc[0],
inplace=True)

# save the cleaned dataset to a new file
Data.to_csv("cleaned_dataset.csv", index=False)
```

decision tree

```
# Load the dataset
data = pd.read_csv('/content/cleaned_dataset.csv')

# Preprocessing the data
# Convert categorical variables to numerical
data = pd.get_dummies(data, columns=['gender', 'ever_married',
'work_type', 'Residence_type', 'smoking_status'])

# Split the data into features and target
X = data.drop(['stroke'], axis=1)
y = data['stroke']

# Create a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Predict the target values for the testing data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
```

```
print('Accuracy:', accuracy)
print('Number of predictions:', len(y_pred))
```

```
Accuracy: 0.9825192802056555
Number of predictions: 1945
```

KNN

```
# Import the necessary libraries
# Load the dataset
Data = pd.read_csv('/content/cleaned_dataset.csv')

# Preprocessing the data
# Convert categorical variables to numerical
Data = pd.get_dummies(Data, columns=['gender', 'ever_married',
'work_type', 'Residence_type', 'smoking_status'])

# Split the data into features and target
X = Data.drop(['stroke'], axis=1)
y = Data['stroke']

# Create a k-NN classifknn
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Predict the target values for the testing data
y_predi = knn.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predi)

print('Accuracy:', accuracy)
print('Number of predictions:', len(y_predi))
```

```
Accuracy: 0.9311053984575836
Number of predictions: 1945
```

XGBOOST

```
# Import the necessary libraries
import xgboost as xgb


# Load the dataset
df = pd.read_csv('/content/cleaned_dataset.csv')
```

```
# Preprocessing the data
# Convert categorical variables to numerical
df = pd.get_dummies(df, columns=['gender', 'ever_married',
'work_type', 'Residence_type', 'smoking_status'])

# Split the data into features and target
X = df.drop(['stroke'], axis=1)
y = df['stroke']

# Create an XGBoost classifier
clf = xgb.XGBClassifier(random_state=42)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Predict the target values for the testing data
y_predic = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predic)

print('Accuracy:', accuracy)
print('Number of predictions:', len(y_predic))
```

```
Accuracy: 0.9773778920308483
Number of predictions: 1945
```

logistic

```
# Load the dataset
Data = pd.read_csv('/content/cleaned_dataset.csv')

# Preprocessing the data
# Convert categorical variables to numerical
Data = pd.get_dummies(Data, columns=['gender', 'ever_married',
'work_type', 'Residence_type', 'smoking_status'])

# Split the data into features and target
X = Data.drop(['stroke'], axis=1)
y = Data['stroke']

# Create a logistic regression classifier
logreg = LogisticRegression()

# Fit the classifier to the training data
logreg.fit(X_train, y_train)

# Predict the target values for the testing data
y_predicts = logreg.predict(X_test)
```

```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_predicts)

print('Accuracy:', accuracy)
print('Number of predictions:', len(y_predicts))
```

```
Accuracy: 0.7737789203084833
Number of predictions: 1945

/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

RANDOM FOREST CLASSIFIER

```
# Load the dataset
Data = pd.read_csv('/content/cleaned_dataset.csv')

# Preprocessing the data
# Convert categorical variables to numerical
Data = pd.get_dummies(Data, columns=['gender', 'ever_married',
'work_type', 'Residence_type', 'smoking_status'])

# Split the data into features and target
X = Data.drop(['stroke'], axis=1)
y = Data['stroke']

# Create a random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to the training data
rf.fit(X_train, y_train)

# Predict the target values for the testing data
y_preds = rf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_preds)
```

```
print('Accuracy:', accuracy)
print('Number of predictions:', len(y_preds))
```

```
Accuracy: 0.9943444730077121
Number of predictions: 1945
```

```
print(sum(1 for i,j,k,l,m in zip(y_pred, y_predi, y_predic,
y_predicts, y_preds) if i==j==k==l==m))
```

```
1455
```

```
unique,counts=np.unique(y_pred,return_counts=True)
dict(zip(unique,counts))
```

```
{0: 941, 1: 1004}
```

```
unique,counts=np.unique(y_predi,return_counts=True)
dict(zip(unique,counts))
```

```
{0: 841, 1: 1104}
```

```
unique,counts=np.unique(y_predic,return_counts=True)
dict(zip(unique,counts))
```

```
{0: 931, 1: 1014}
```

```
unique,counts=np.unique(y_predicts,return_counts=True)
dict(zip(unique,counts))
```

```
{0: 879, 1: 1066}
```

```
unique,counts=np.unique(y_preds,return_counts=True)
dict(zip(unique,counts))
```

```
{0: 964, 1: 981}
```

Future Scope of Improvement

Brain stroke prediction is an important area of research with significant potential for improving pa tient outcomes and reducing healthcare costs. With advances in machine learning and wearable t echnology, there are many potential directions for future research in this area. One promising app roach is the development of mobile apps that incorporate brain stroke prediction models, allowing users to monitor their risk for brain stroke and receive personalized recommendations for reducing their risk. Another approach is the integration of wearable devices and electronic health records to provide real-time monitoring and risk assessments for patients at high risk for brain stroke. Natural language processing and chatbots could also be used to ask patients about their risk factors and provide personalized recommendations. Deep learning algorithms, such as CNNs and RNNs, co uld be used to develop more accurate prediction models. Finally, web-based platforms could be d eveloped to enable healthcare providers to input patient data and receive real-time risk assessme nts for brain stroke, enabling early detection and intervention. Overall, there is significant potenti al for future research in brain stroke prediction to improve patient outcomes and reduce healthca re costs.

The future scope of brain stroke prediction is promising, as advancements in technology and research continue to improve our ability to identify individuals at high risk of stroke. Some of the key ar eas of development in stroke prediction include:

Big Data and Artificial Intelligence (AI) - With the availability of large datasets and advanced AI algorithms, we can now identify patterns and risk factors associated with strokes more accurately. For example, AI can analyze medical records and lifestyle data to predict an individual's likelihood of experiencing a stroke in the future.

Genetic Testing - Genetic testing can identify gene variants associated with an increased risk of stroke. This information can be used to tailor prevention and treatment strategies to an individual's specific needs.

Biomarkers - Biomarkers are measurable indicators that can be used to predict the risk of stroke. For example, high levels of certain proteins in the blood have been linked to an increased risk of st roke.

Finally, web-based platforms could be developed to enable healthcare providers to input patient data and receive real-time risk assessments for brain stroke. By integrating these platforms with EHRs, providers could receive real-time alerts when a patient is at high risk for brain stroke, enabling early detection and intervention. This could potentially reduce hospitalizations and emergency room visits, improving patient outcomes and reducing healthcare costs.

REFERENCES

- **Javatpoint: -** Machine Learning: What It is, Tutorial, Definition, Types - Javatpoint

- **GeeksforGeeks: -** What is Machine Learning? - GeeksforGeeks

- **Kaggle: -** https://www.kaggle.com/