CSE 422:Artificial Intelligence

Inspiring Excellence

Project

Brain Stroke Analysis And Classification

Group No:03

Student Id	Student Name
24141175	Minhaj Uddin
24141174	Mirza Bushra Tabassum

1.Introduction: The research project utilizes machine learning to identify and categorize characteristics related with brain strokes, such as gender, age, hypertension, heart disease, marital status, habitation type, and average glucose level,BMI, employment type, and smoking status. Computer algorithms identify patterns that may indicate a stroke risk.

Several models were evaluated to see which had the greatest accuracy in identifying and predicting brain strokes.

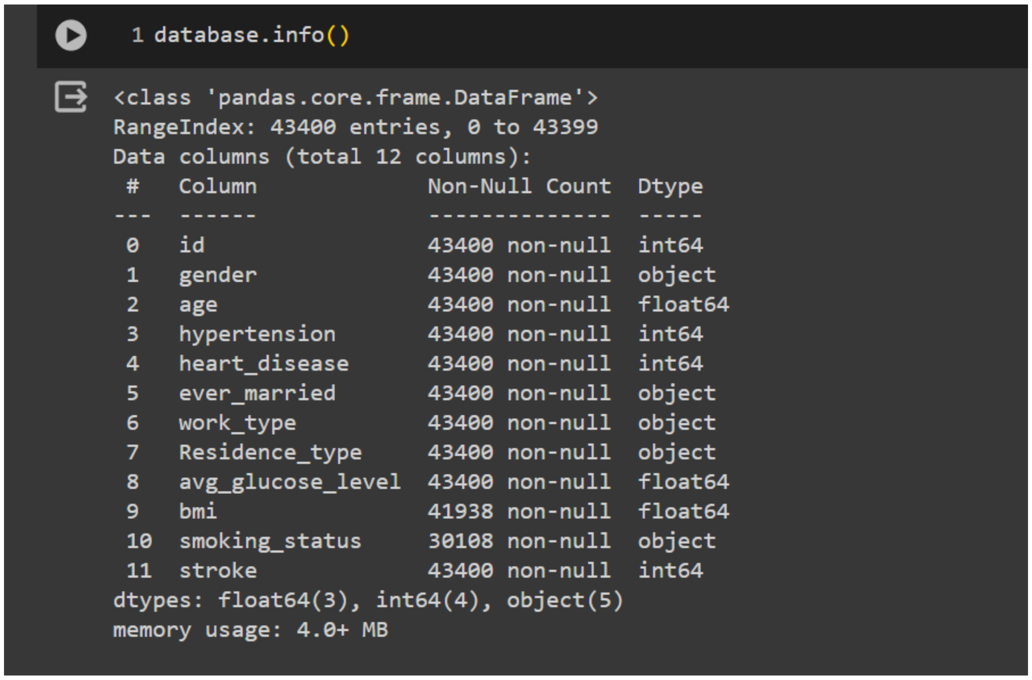
This effort is motivated by the mysterious nature of brain strokes, sometimes known as "silent killers," due to their faint early signs. Late diagnosis can cause lasting neurological damage, reducing quality of life or even leading to death. The growing patient-to-doctor ratio poses problems for traditional diagnostic procedures to diagnose diseases accurately and on time.

The digital era and the quantity of health data, ranging from electronic health records to wearable-generated biometric data, provide an untapped resource.Machine learning has emerged as a transformational solution, capable of recognising

Intricate patterns and relationships exist within large datasets. This goes beyond enhancing human observation; it enhances it to degrees above intrinsic talents. Machine learning algorithms can predict stroke start before symptoms occur, leading to quicker treatments and reduced healthcare expenses.
Integrating machine learning into stroke diagnosis is seen as a humanitarian endeavor with the potential to transform stroke care. This project attempts to change the way stroke is predicted and treated by giving patients a fighting chance against one of the most deadly medical conditions.

2.Dataset Description:

Link:
Number of Features: 12
Type of class/label: Categorical and Continuous
Number of data points: 43400

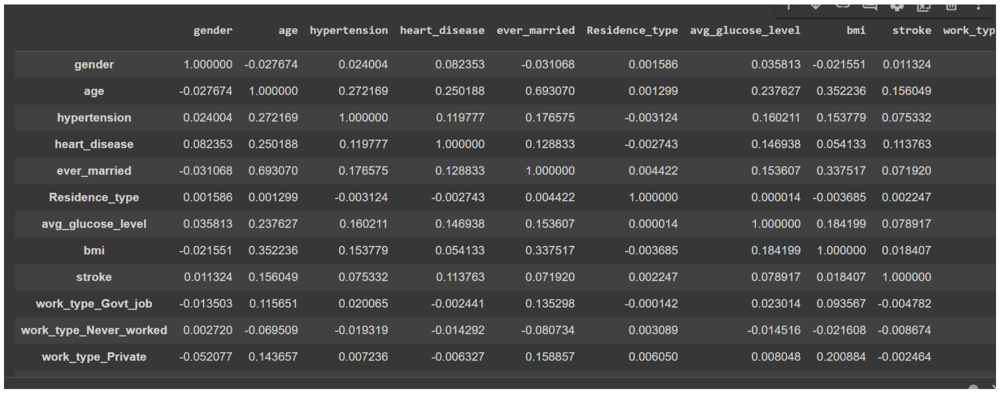


```
database.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43400 entries, 0 to 43399
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 43400 non-null  int64  
 1   gender             43400 non-null  object 
 2   age                43400 non-null  float64
 3   hypertension       43400 non-null  int64  
 4   heart_disease      43400 non-null  int64  
 5   ever_married       43400 non-null  object 
 6   work_type          43400 non-null  object 
 7   Residence_type     43400 non-null  object 
 8   avg_glucose_level  43400 non-null  float64
 9   bmi                41938 non-null  float64
 10  smoking_status     30108 non-null  object 
 11  stroke             43400 non-null  int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 4.0+ MB
```

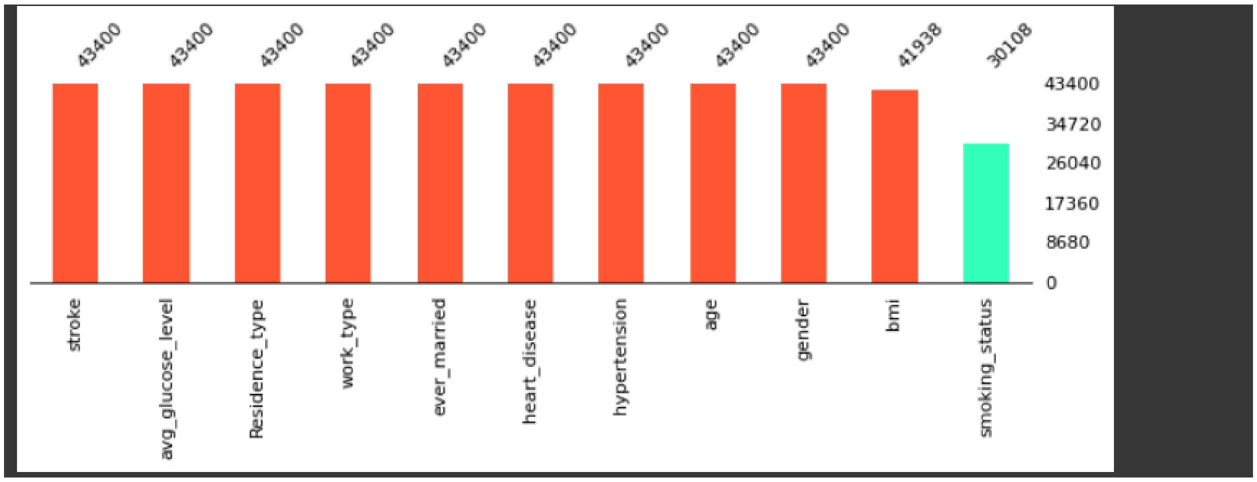
Categorical Features:
Gender
Ever_married
Work_type
Residence_type
Smoking_status (though it contains NaN values, which might need handling)
Continuous Features: Age
Hypertension
Heart_disease
Avg_glucose_level
BMI
Label (Target):
Stroke (Binary: 0 or 1),categorical

Correlation of all features:
Correlation between features where close to 1 means two features have strong positive linear relationship means if one increases,the other will increase as well.If the value is close to -1 that means two features have strong negative linear relationship means if one increases,the other will decrease.If value is 0 or close to 0 means no linear correlation between them



	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	work_typ
gender	1.000000	-0.027674	0.024004	0.082353	-0.031068	0.001586	0.035813	-0.021551	0.011324	
age	-0.027674	1.000000	0.272169	0.250188	0.693070	0.001299	0.237627	0.352236	0.156049	
hypertension	0.024004	0.272169	1.000000	0.119777	0.176575	-0.003124	0.160211	0.153779	0.075332	
heart_disease	0.082353	0.250188	0.119777	1.000000	0.128833	-0.002743	0.146938	0.054133	0.113763	
ever_married	-0.031068	0.693070	0.176575	0.128833	1.000000	0.004422	0.153607	0.337517	0.071920	
Residence_type	0.001586	0.001299	-0.003124	-0.002743	0.004422	1.000000	0.000014	-0.003685	0.002247	
avg_glucose_level	0.035813	0.237627	0.160211	0.146938	0.153607	0.000014	1.000000	0.184199	0.078917	
bmi	-0.021551	0.352236	0.153779	0.054133	0.337517	-0.003685	0.184199	1.000000	0.018407	
stroke	0.011324	0.156049	0.075332	0.113763	0.071920	0.002247	0.078917	0.018407	1.000000	
work_type_Govt_job	-0.013503	0.115651	0.020065	-0.002441	0.135298	-0.000142	0.023014	0.093567	-0.004782	
work_type_Never_worked	0.002720	-0.069509	-0.019319	-0.014292	-0.080734	0.003089	-0.014516	-0.021608	-0.008674	
work_type_Private	-0.052077	0.143657	0.007236	-0.006327	0.158857	0.006050	0.008048	0.200884	-0.002464	

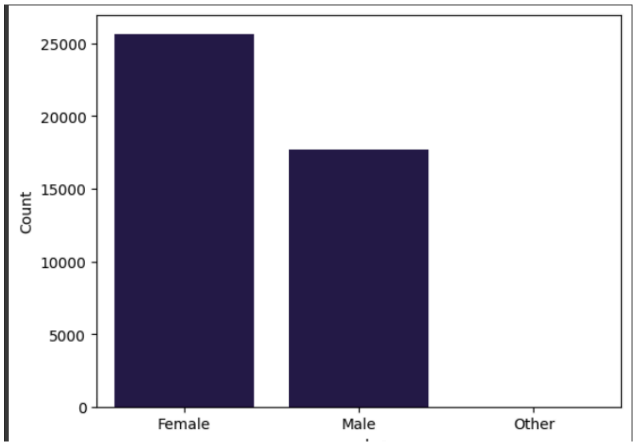
Imbalance:

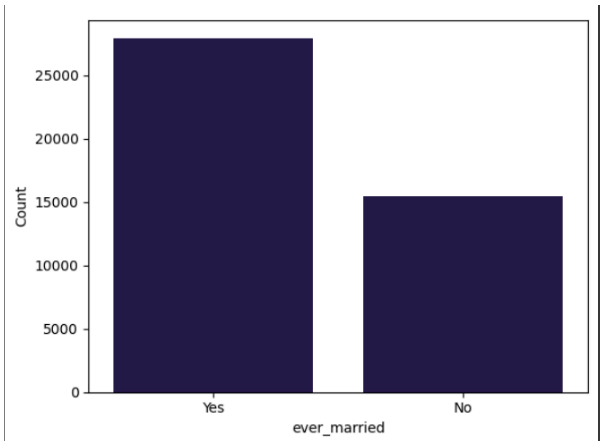


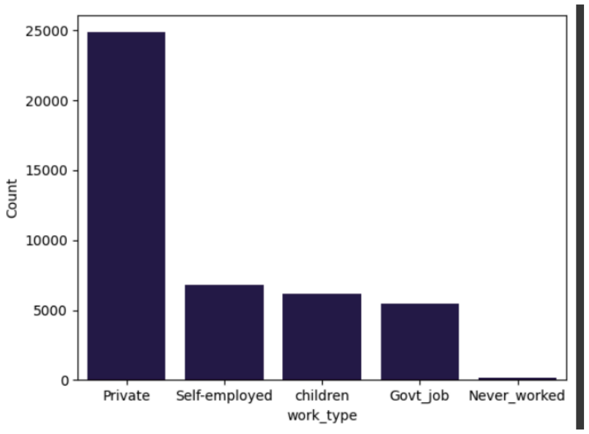
BMI and Smoking_status don't have the same number of datapoints as other features

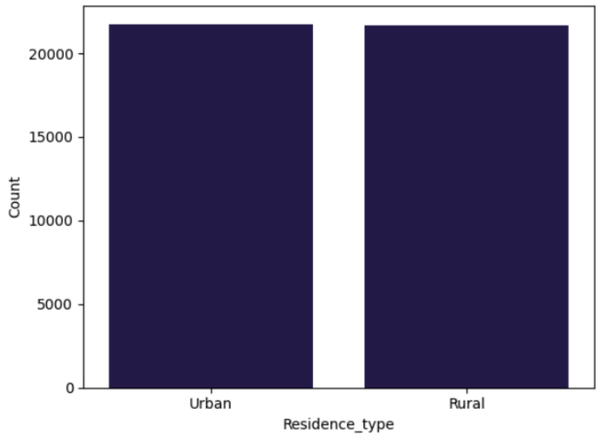
3.Dataset Pre-processing:

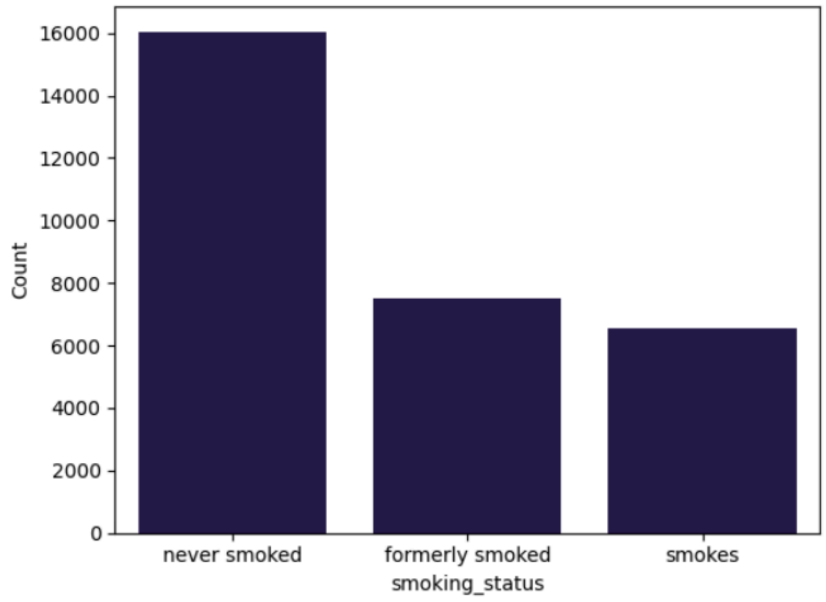
Categorical features:





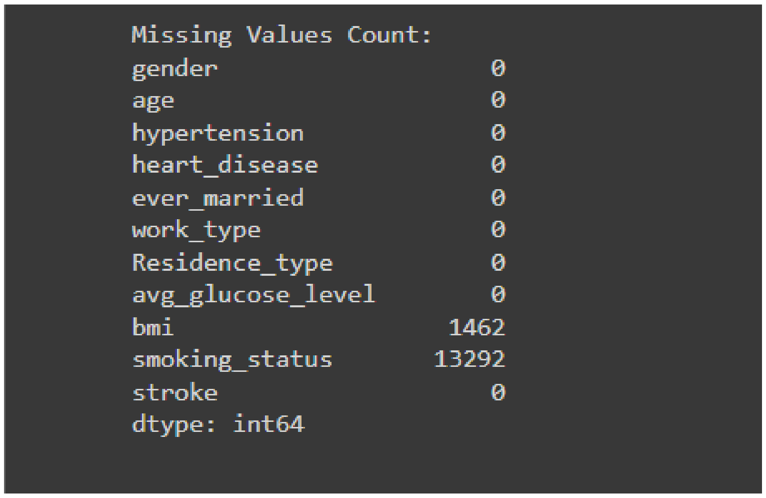






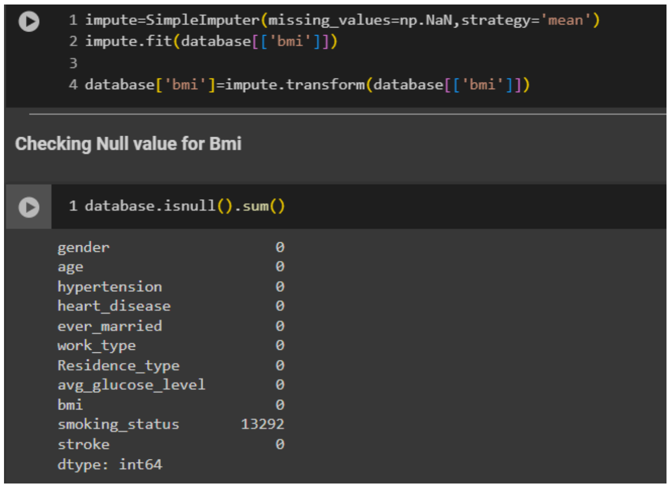
Null values:

5 features contained a total of 868 null values in BMI and 13292 null values in smoking status



```
Missing Values Count:
gender                   0
age                      0
hypertension             0
heart_disease            0
ever_married             0
work_type                0
Residence_type           0
avg_glucose_level        0
bmi                   1462
smoking_status       13292
stroke                   0
dtype: int64
```

Imputing BMI:



```
impute=SimpleImputer(missing_values=np.NaN,strategy='mean')
impute.fit(database[['bmi']])

database['bmi']=impute.transform(database[['bmi']])
```

Checking Null value for Bmi

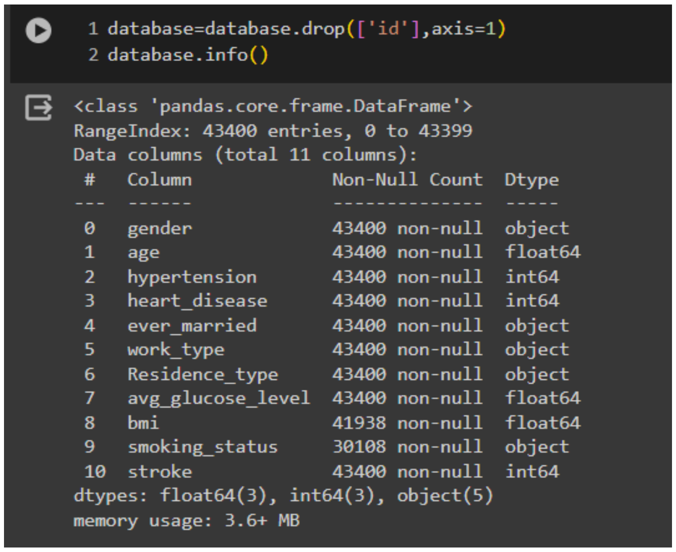
```
database.isnull().sum()
```

```
gender                   0
age                      0
hypertension             0
heart_disease            0
ever_married             0
work_type                0
Residence_type           0
avg_glucose_level        0
bmi                      0
smoking_status       13292
stroke                   0
dtype: int64
```

We are basically imputing missing values with the mean of non-missing values.With the help of that there are no missing values for BMI.

Removing ID Column:

As the id column is not needed so we are removing it.

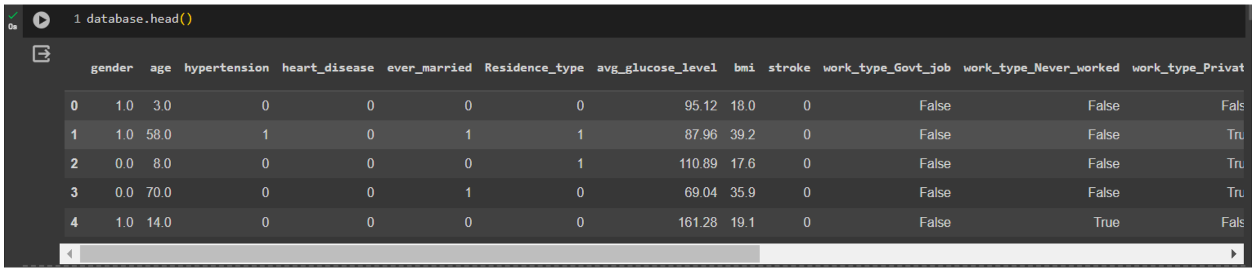


```
database=database.drop(['id'],axis=1)
database.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43400 entries, 0 to 43399
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             43400 non-null  object 
 1   age                43400 non-null  float64
 2   hypertension       43400 non-null  int64  
 3   heart_disease      43400 non-null  int64  
 4   ever_married       43400 non-null  object 
 5   work_type          43400 non-null  object 
 6   Residence_type     43400 non-null  object 
 7   avg_glucose_level  43400 non-null  float64
 8   bmi                41938 non-null  float64
 9   smoking_status     30108 non-null  object 
 10  stroke             43400 non-null  int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 3.6+ MB
```

Handling Categorical Values:

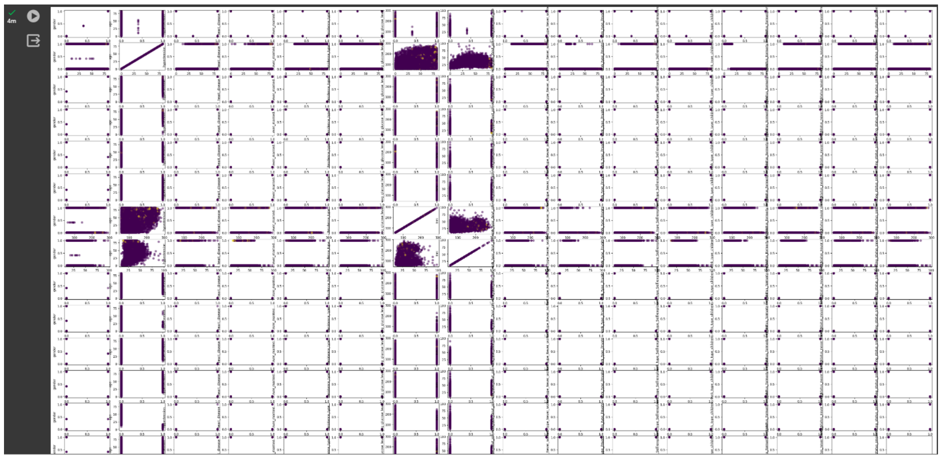
Mapped ‘ever_married’ Yes into ‘1’ and No into ‘0’.Mapped ‘Residence_type’ ‘Urban’ into 1, ’Rural’ into 0. Using ‘one-hot encoding’ to convert categorical (non-numeric) variables into a numerical format.



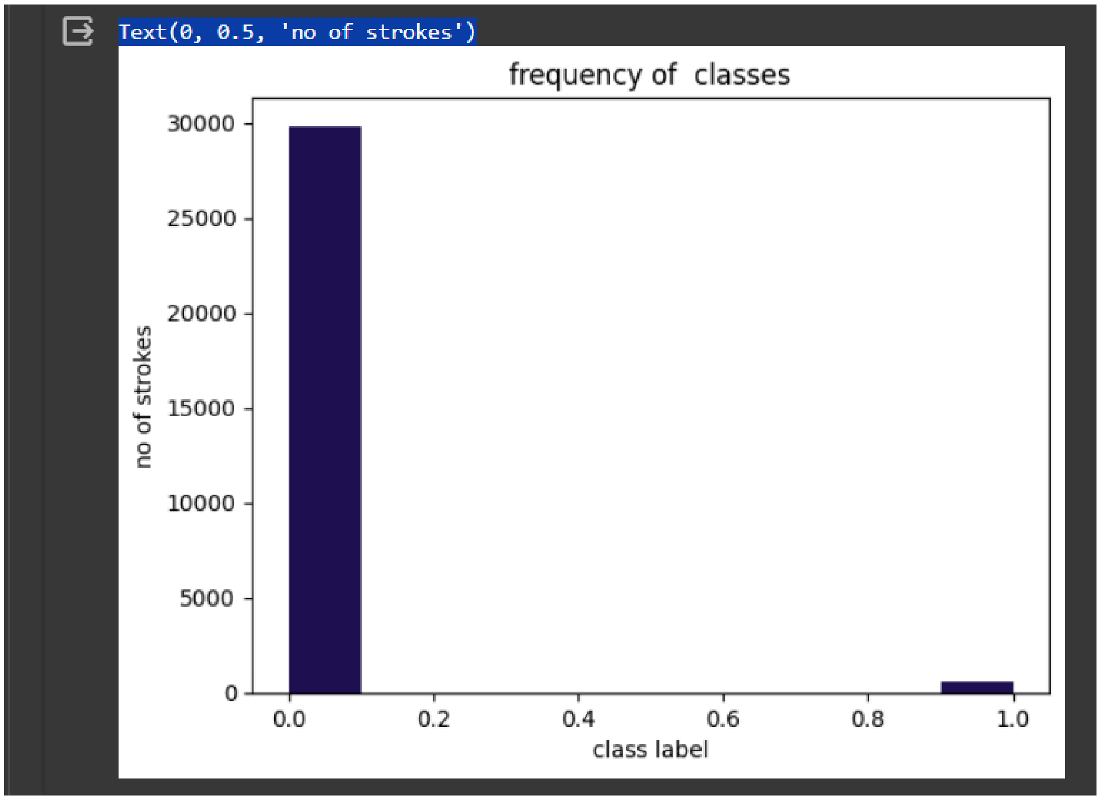
```
database.head()
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_job	work_type_Never_worked	work_type_Privat
0	1.0	3.0	0	0	0	0	95.12	18.0	0	False	False	Fals
1	1.0	58.0	1	0	1	1	87.96	39.2	0	False	False	Tru
2	0.0	8.0	0	0	0	1	110.89	17.6	0	False	False	Tru
3	0.0	70.0	0	0	1	0	69.04	35.9	0	False	False	Tru
4	1.0	14.0	0	0	0	0	161.28	19.1	0	False	True	Fals

<u>4.Dataset Splitting:</u>A classic 70-30 split was maintained - 30380 for training and 13020 for testing. After that we have shown the scatter matrix for all features.

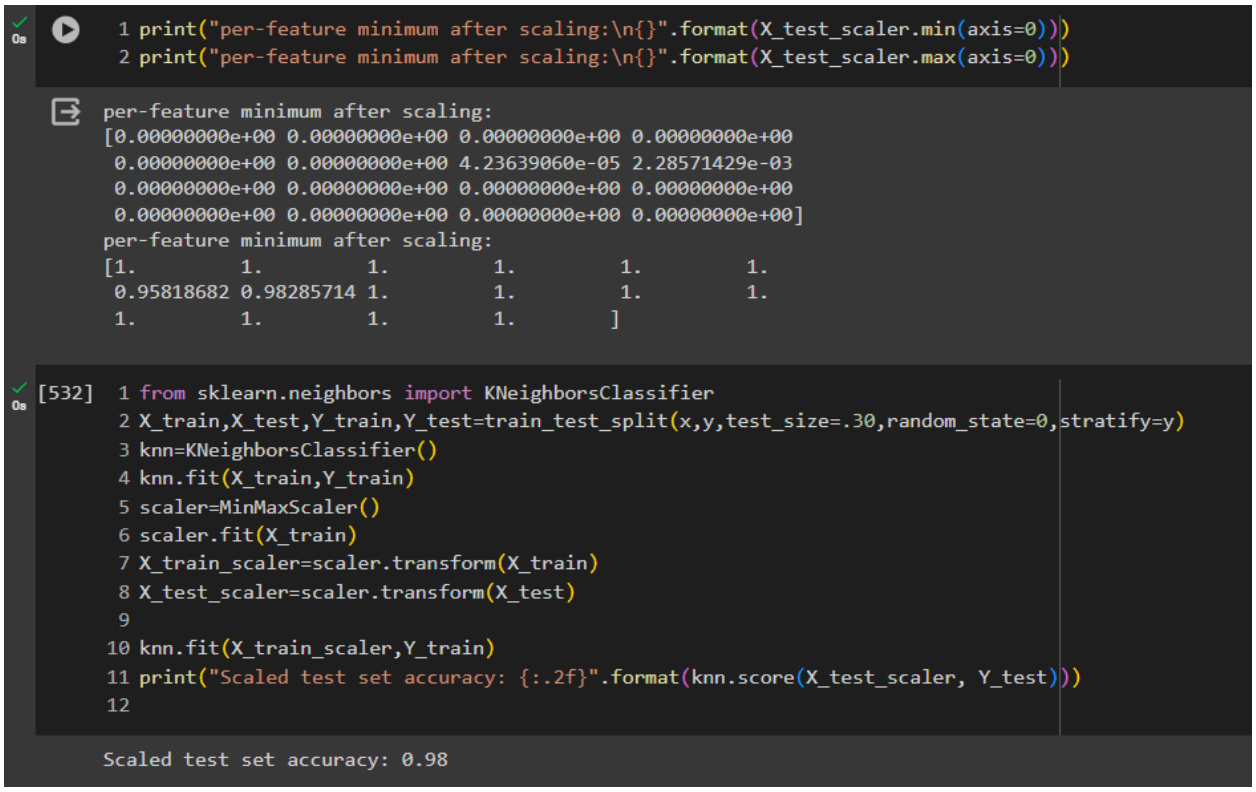


Also , we generated a histogram to visualize the distribution of stroke occurrences represented by the target variable "Y_train" which helps us in understanding the frequency or distribution of stroke cases within the dataset.



```
Text(0, 0.5, 'no of strokes')
```

5.Scaling: We used MinMaxScaler from scikit-learn to scale our datasets. Also, created an instance of it as ‘scaler’ and calculated the min and max values with our training data (X_train). Then, we transformed both our training and testing data with this scaling. At last, verified and displayed the min and max values for each feature before and after scaling.And after scaling the scaled test set accuracy was almost 98 percent.



```
print("per-feature minimum after scaling:\n{}".format(X_test_scaler.min(axis=0)))
print("per-feature minimum after scaling:\n{}".format(X_test_scaler.max(axis=0)))
```

```
per-feature minimum after scaling:
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 4.23639060e-05 2.28571429e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
per-feature minimum after scaling:
[1.         1.         1.         1.         1.         1.
 0.95818682 0.98285714 1.         1.         1.         1.
 1.         1.         1.         1.        ]
```

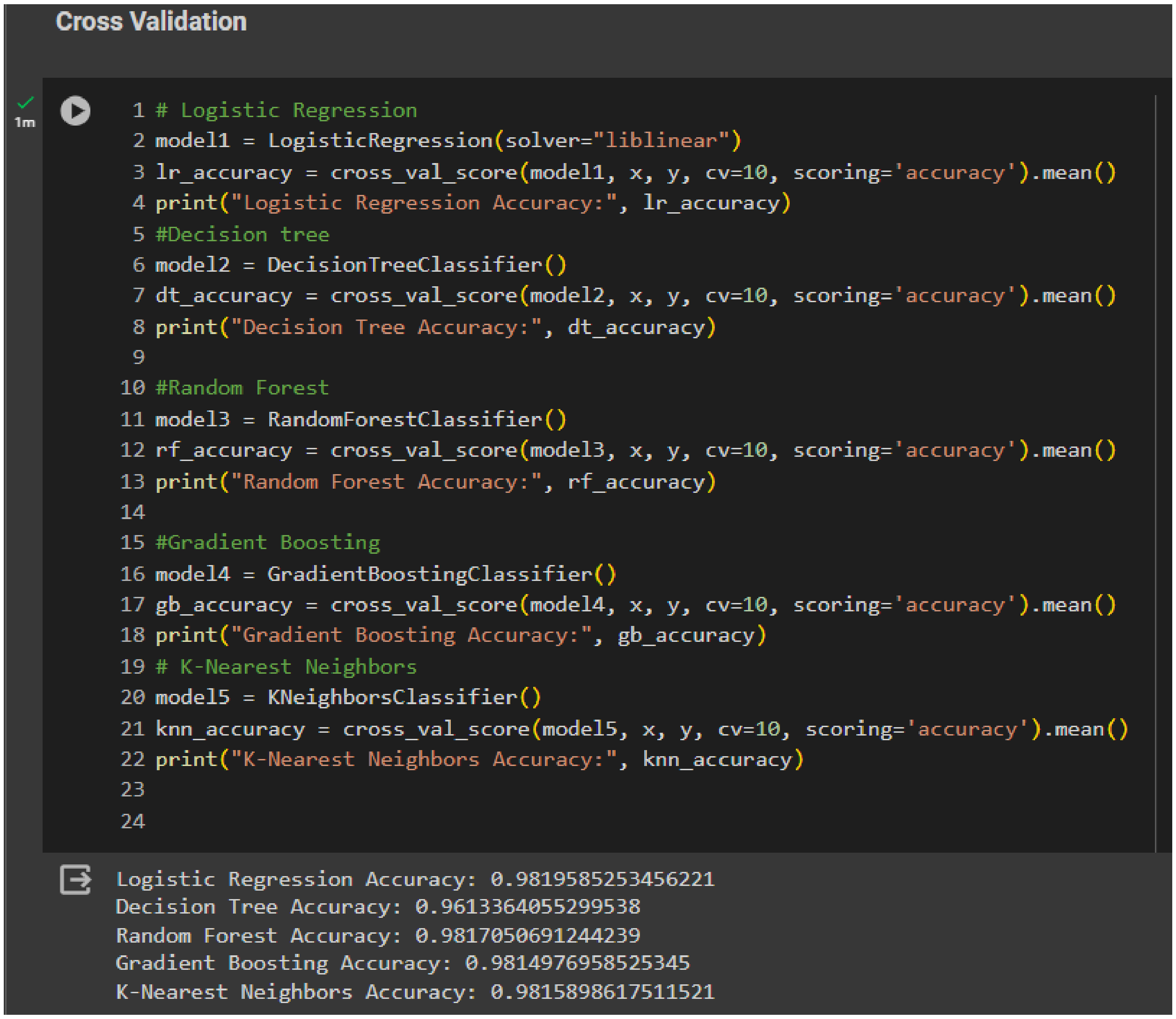
```
[532]
from sklearn.neighbors import KNeighborsClassifier
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=.30,random_state=0,stratify=y)
knn=KNeighborsClassifier()
knn.fit(X_train,Y_train)
scaler=MinMaxScaler()
scaler.fit(X_train)
X_train_scaler=scaler.transform(X_train)
X_test_scaler=scaler.transform(X_test)

knn.fit(X_train_scaler,Y_train)
print("Scaled test set accuracy: {:.2f}".format(knn.score(X_test_scaler, Y_test)))

```

```
Scaled test set accuracy: 0.98
```

6.Cross Validation and Smoting:

Cross validation:

We used 10-fold cross-validation with cross_val_score to measure the performance of the models. Then the approach calculated and displayed the accuracy of each model, such as Decision Tree, Random Forest, Gradient Boosting, K-Nearest Neighbors, and Logistic Regression. Then, a Voting Classifier that combined the predictions of multiple models using a soft voting method was introduced . We also used cross_val_score with 10-fold cross-validation to estimate the accuracy of the Voting Classifier and assess our ensemble model.



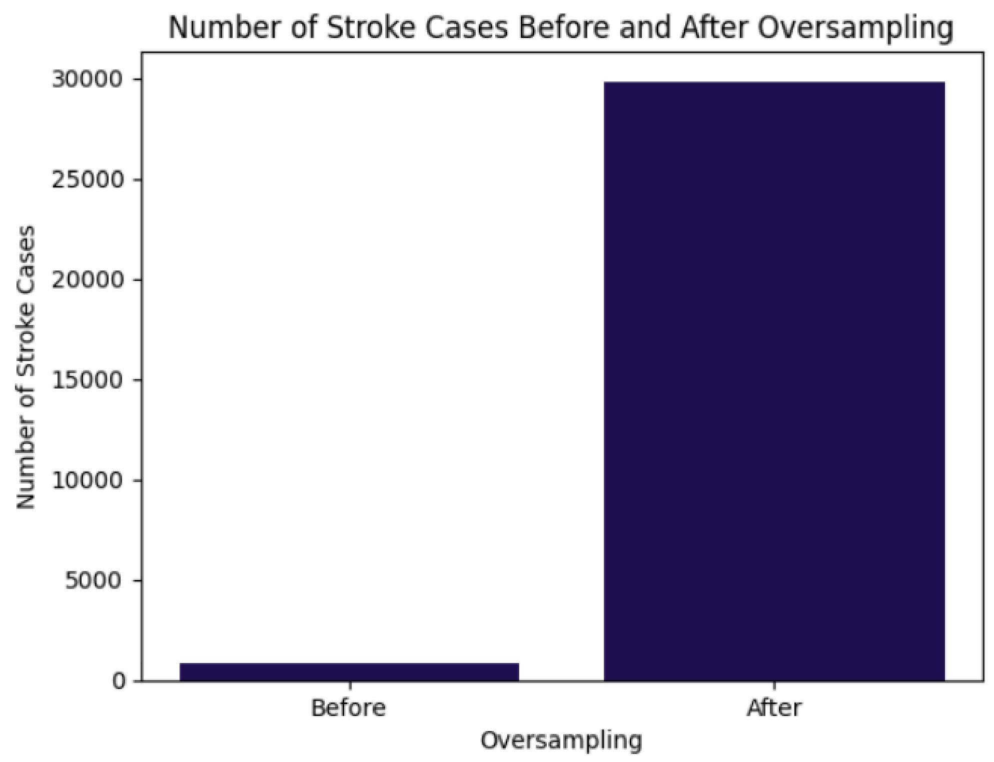
Cross Validation

```python
# Logistic Regression
model1 = LogisticRegression(solver="liblinear")
lr_accuracy = cross_val_score(model1, x, y, cv=10, scoring='accuracy').mean()
print("Logistic Regression Accuracy:", lr_accuracy)
#Decision tree
model2 = DecisionTreeClassifier()
dt_accuracy = cross_val_score(model2, x, y, cv=10, scoring='accuracy').mean()
print("Decision Tree Accuracy:", dt_accuracy)

#Random Forest
model3 = RandomForestClassifier()
rf_accuracy = cross_val_score(model3, x, y, cv=10, scoring='accuracy').mean()
print("Random Forest Accuracy:", rf_accuracy)

#Gradient Boosting
model4 = GradientBoostingClassifier()
gb_accuracy = cross_val_score(model4, x, y, cv=10, scoring='accuracy').mean()
print("Gradient Boosting Accuracy:", gb_accuracy)
# K-Nearest Neighbors
model5 = KNeighborsClassifier()
knn_accuracy = cross_val_score(model5, x, y, cv=10, scoring='accuracy').mean()
print("K-Nearest Neighbors Accuracy:", knn_accuracy)


```

```
Logistic Regression Accuracy: 0.9819585253456221
Decision Tree Accuracy: 0.9613364055299538
Random Forest Accuracy: 0.9817050691244239
Gradient Boosting Accuracy: 0.9814976958525345
K-Nearest Neighbors Accuracy: 0.9815898617511521
```

Smoting:

To reduce the difference between the minority and majority class smoting has been introduced. It works by oversampling the minority region , resulting into better distribution of two different classes. The number of stroke cases before and after oversampling is visualized here using a bar chart.This technique is used to balance an imbalanced dataset of stroke cases and increase its accuracy in predicting them.



7.Model Training:

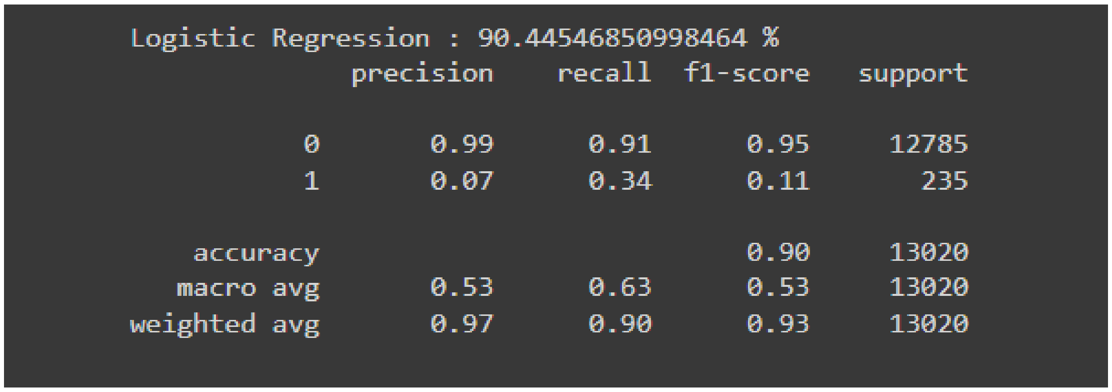
The main idea behind the proposed system architecture was to create a brain stroke prediction system based on the inputs datasets. For predicting the accuracy of brain stroke, this study analyzed the classification algorithms namely KNN, Decision Tree, Random forest,Gradient Boosting and Linear Regression. This technique helps the model to capture the important features of the stroke cases and increase its accuracy in predicting them.

1)Logistic Regression:

Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set.

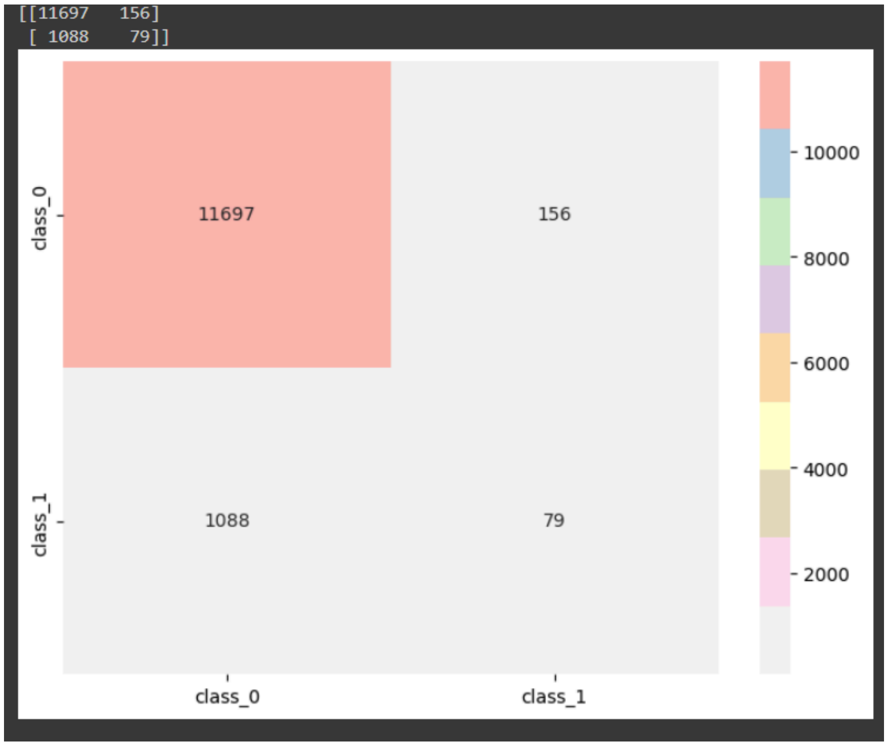
● A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. For example, a logistic regression could be used to predict whether a political candidate will win or lose an election or whether a high school student will be admitted or not to a particular college. These binary outcomes allow straightforward decisions between two alternatives.

● A logistic regression model can take into consideration multiple input criteria. In the case of college acceptance, the logistic function could consider factors such as the student's grade point average, SAT score and number of extracurricular activities. Based on historical data about earlier outcomes involving the same input criteria, it then scores new cases on their probability of falling into one of two outcome categories.



```
Logistic Regression : 90.44546850998464 %
              precision    recall  f1-score   support

           0       0.99      0.91      0.95     12785
           1       0.07      0.34      0.11       235

    accuracy                           0.90     13020
   macro avg       0.53      0.63      0.53     13020
weighted avg       0.97      0.90      0.93     13020
```

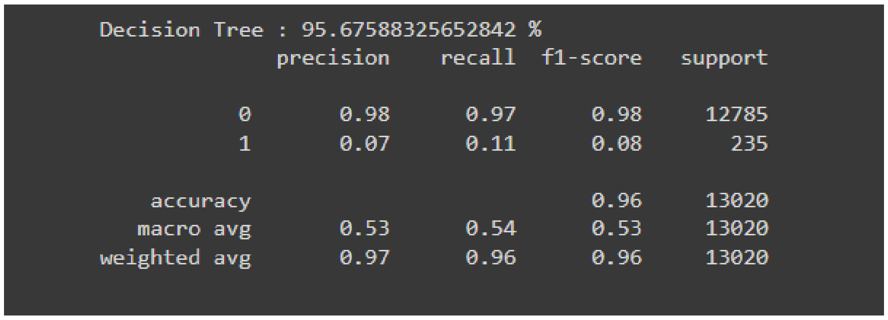
Confusion Matrix:



```
[[11697   156]
 [ 1088    79]]
```

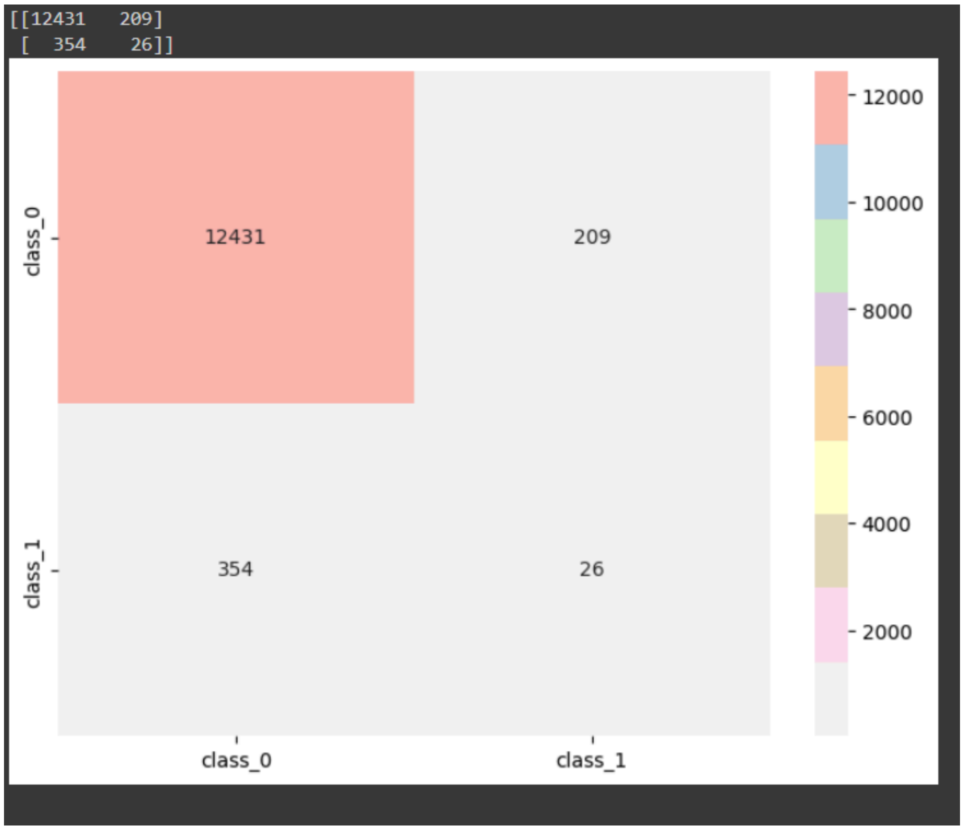
2)Decision Tree:

Decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.



```
Decision Tree : 95.67588325652842 %
              precision    recall  f1-score   support

           0       0.98      0.97      0.98     12785
           1       0.07      0.11      0.08       235

    accuracy                           0.96     13020
   macro avg       0.53      0.54      0.53     13020
weighted avg       0.97      0.96      0.96     13020
```

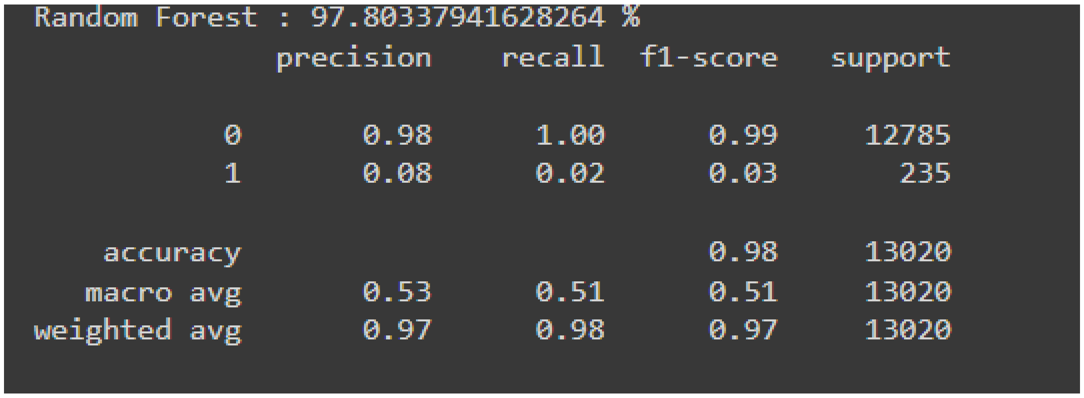
Confusion matrix:



```
[[12431   209]
 [  354    26]]
```

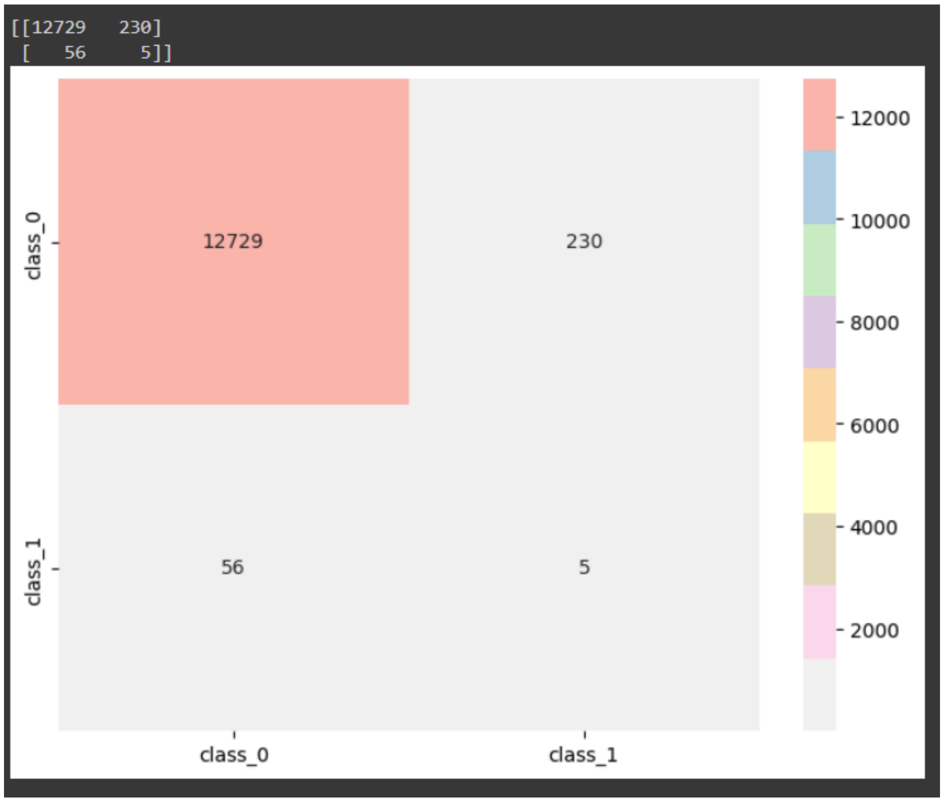
3)Random Forest:

Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve regression or classification problems. The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble consists of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e., the most frequent categorical variable—will yield the predicted class. Finally, the sample is then used for cross-validation, finalizing that prediction.



```
Random Forest : 97.80337941628264 %
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     12785
           1       0.08      0.02      0.03       235

    accuracy                           0.98     13020
   macro avg       0.53      0.51      0.51     13020
weighted avg       0.97      0.98      0.97     13020
```

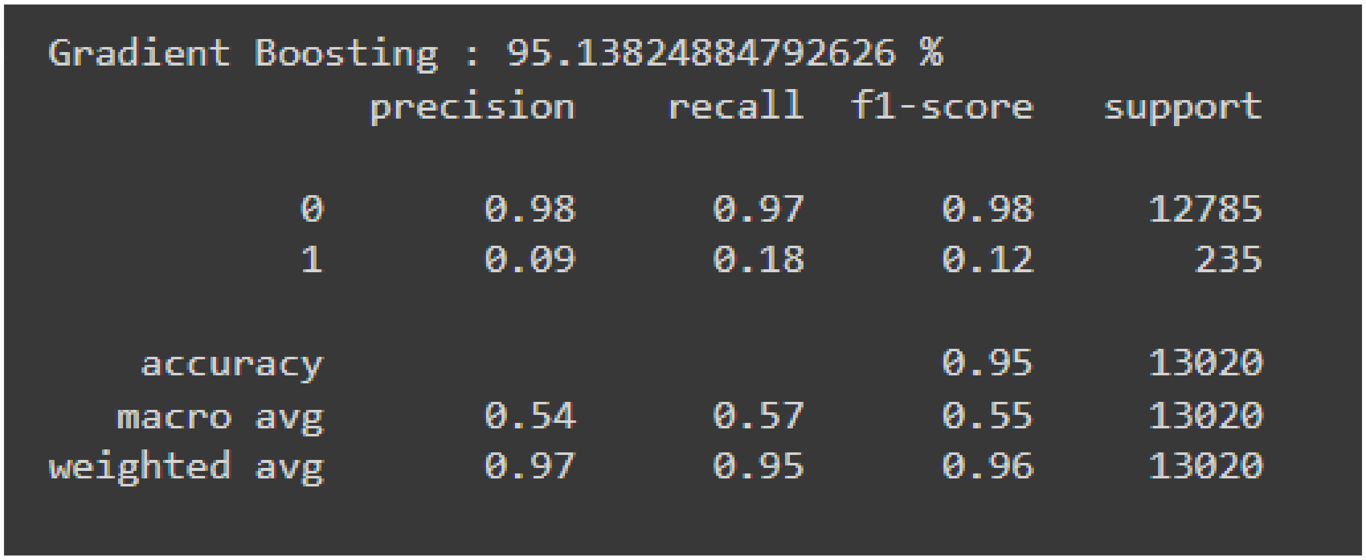
Confusion matrix:



```
[[12729   230]
 [   56     5]]
```

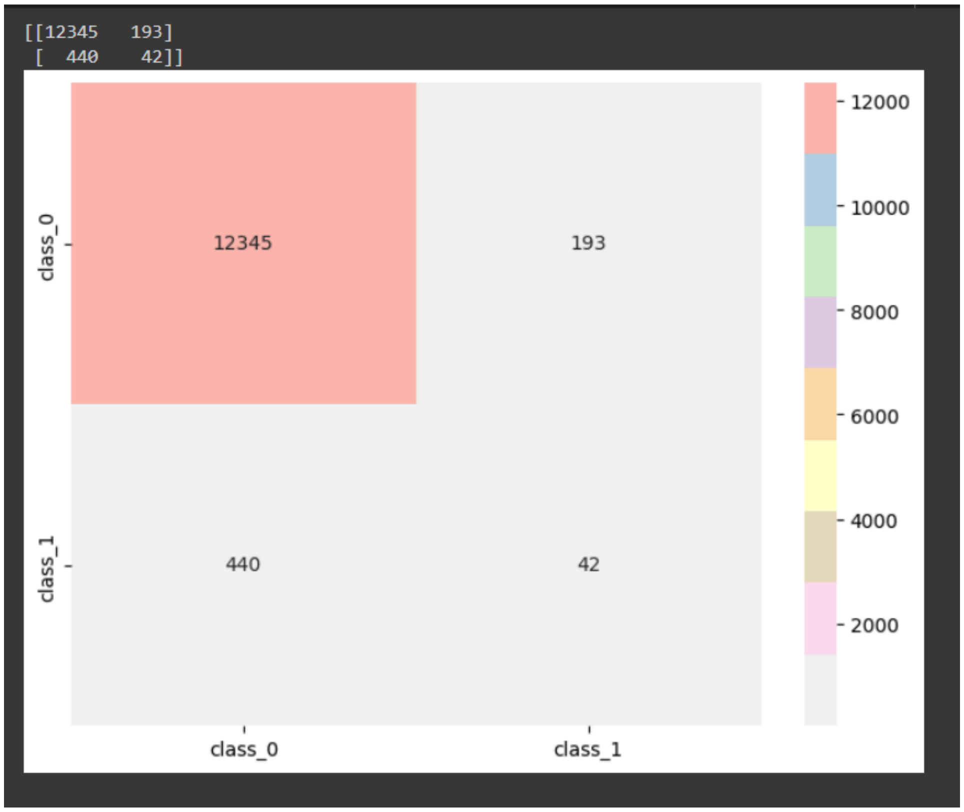
4)Gradient Boosting:

Gradient Boosting is an ensemble learning method that builds a powerful predictive model by iteratively improving upon the weaknesses of simpler models, typically decision trees. It starts with an initial model, makes predictions, calculates the errors (residuals), and then trains a new model to correct these errors. This process is repeated, gradually refining the model's predictions. The term "gradient" refers to the use of gradient descent to minimize the prediction errors. This technique, known for its high predictive accuracy, has various implementations like Gradient Boosting Machines (GBM), XGBoost, LightGBM, and CatBoost. It excels in handling complex relationships in data and is widely applied in tasks such as regression, classification, and ranking. The sequential nature of gradient boosting allows it to capture intricate patterns in the data, making it a popular choice in machine learning applications.



```
Gradient Boosting : 95.13824884792626 %
              precision    recall  f1-score   support

           0       0.98      0.97      0.98     12785
           1       0.09      0.18      0.12       235

    accuracy                           0.95     13020
   macro avg       0.54      0.57      0.55     13020
weighted avg       0.97      0.95      0.96     13020
```

Confusion matrix:



```
[[12345   193]
 [  440    42]]
```

5)KNN Classifier:

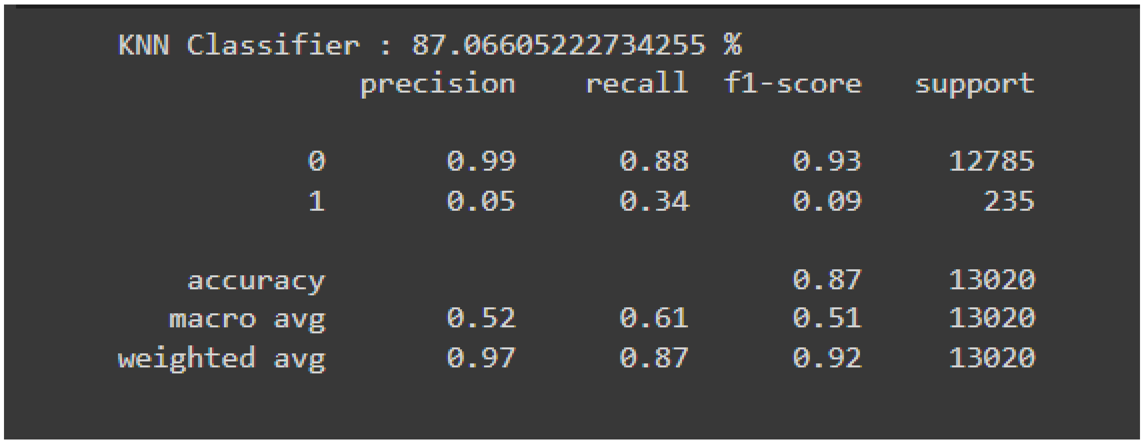
KNN Model:

● K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

● K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories.

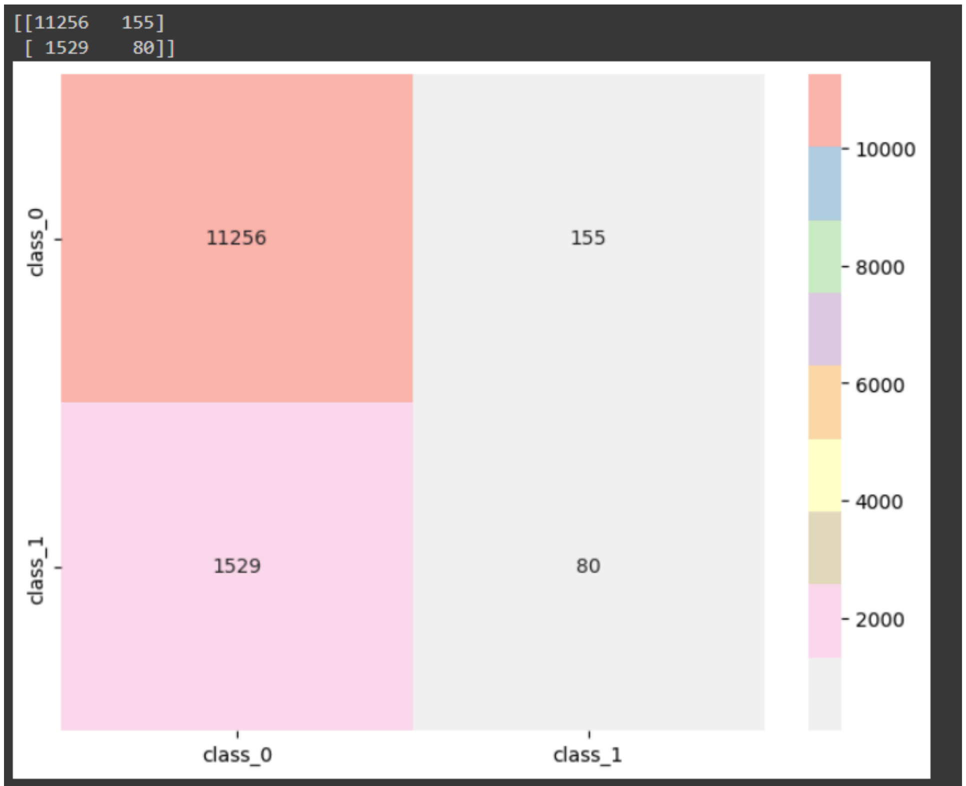
● K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm.

● K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
● K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
● 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.



```
KNN Classifier : 87.06605222734255 %
              precision    recall  f1-score   support

           0       0.99      0.88      0.93     12785
           1       0.05      0.34      0.09       235

    accuracy                           0.87     13020
   macro avg       0.52      0.61      0.51     13020
weighted avg       0.97      0.87      0.92     13020
```

Confusion matrix:



```
[[11256   155]
 [ 1529    80]]
```

8)Model Testing:

Logistic Regression: 90% Accuracy

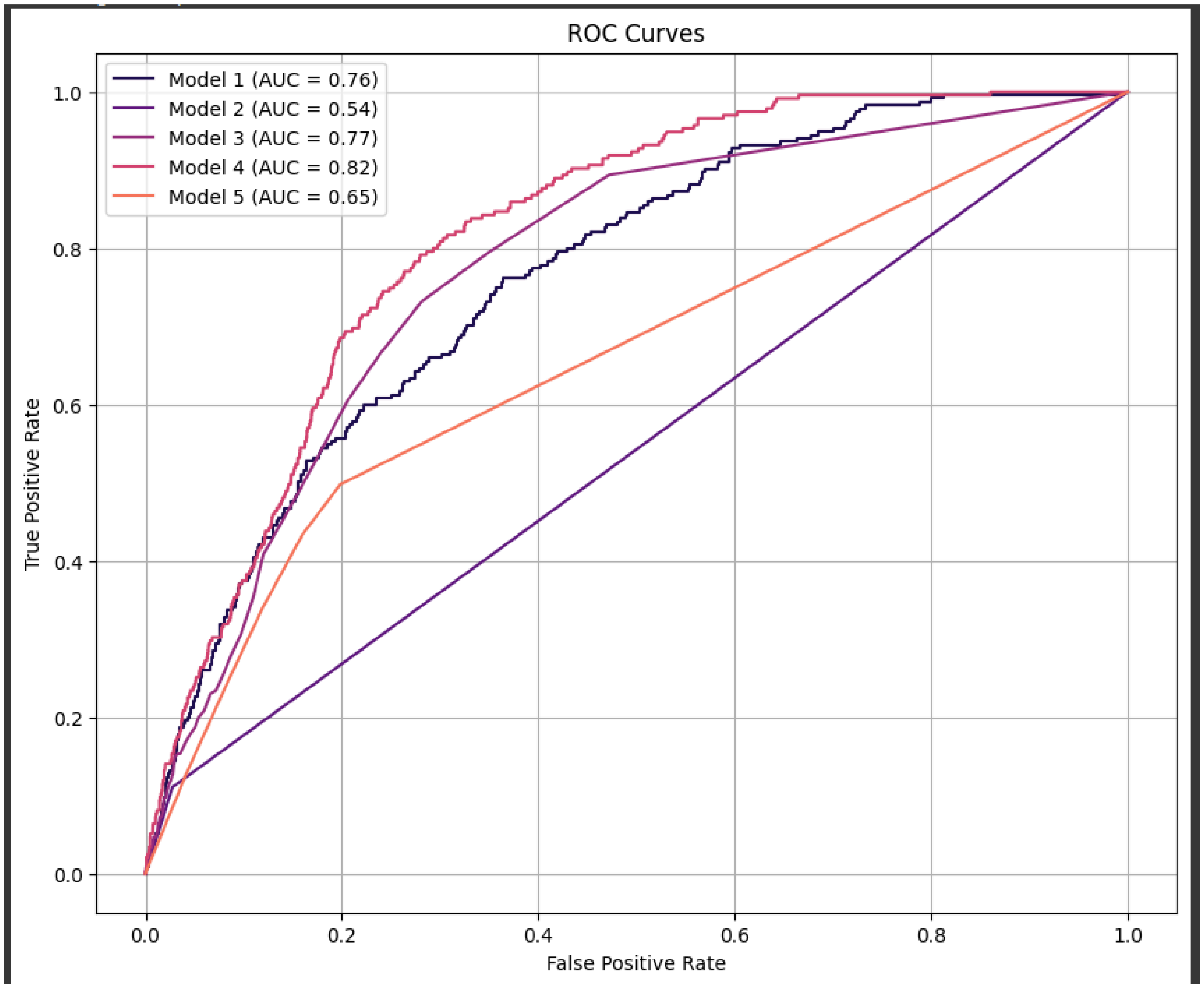
Decision Tree: 95% Accuracy

KnnClassifier: 87% Accuracy

Gradient boosting: 95% Accuracy

Random forest: 97% Accuracy

Roc Curve graph for all model with AUC score:

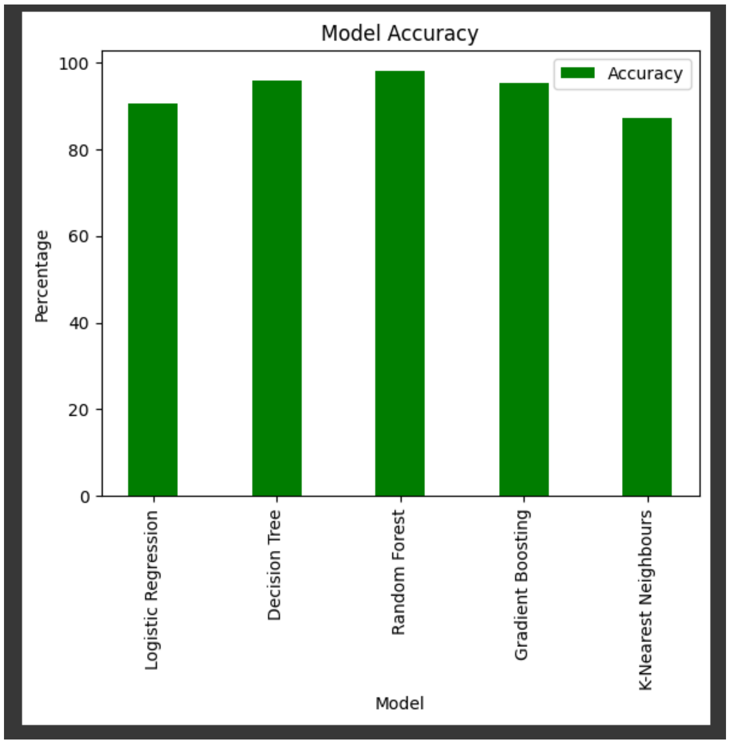


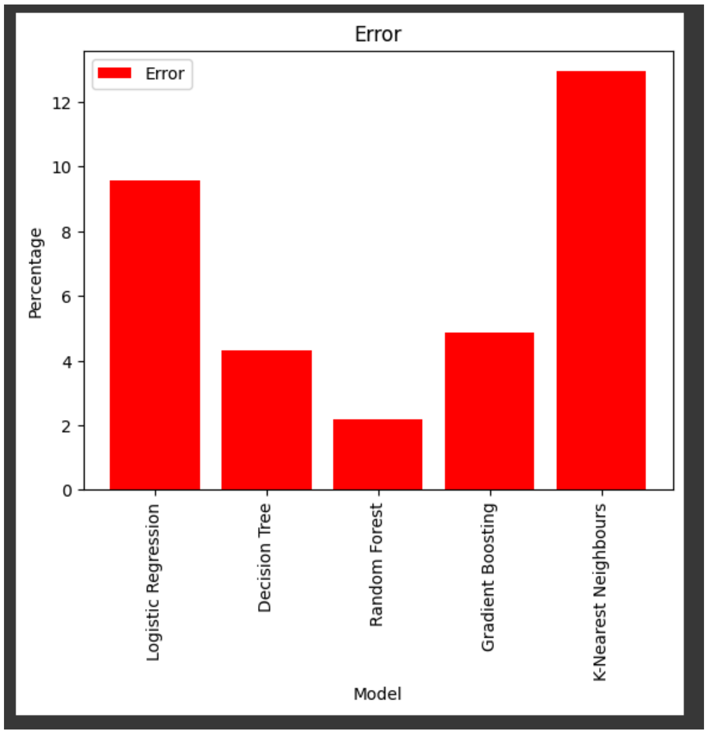
Comparison Analysis:

Model Name	**Accuracy(%)**	**Error(%)**
Logistic Regression	**90%**	**10%**
Decision Tree	**95%**	**5%**
Random Forest	**97%**	**3%**
Gradient Boosting	**95%**	**5%**
KNN Classifier	**87%**	**13%**

From the table, we can see that the Random forest model showed the best performance with 97% accuracy and only 3% error. On the other hand, the worst performance was given by Knn classifier 87% where error is 13% and Logistic regression which is 90% accuracy and the error was 10%. After that, Decision tree and Gradient boosting was mostly satisfying with 95% accuracy where error is 5%.

Here is the graph for all models with accuracy and error.





Conclusion:

The findings from the application of various machine learning algorithms, including Logistic Regression, Decision Tree, K-Nearest Neighbor, and Random Forest Classifier, for predicting brain strokes based on personal indicators show promising results. The accuracy of these models ranges from 87% to 97%, indicating good performance in stroke prediction. Among the tested models, Random Forest stands out with an accuracy rate of 97%, while Decision Tree also demonstrates promising results with 95% accuracy. Although Logistic Regression achieved 90% and K-NN Classifier achieved 87% accuracy, there is room for further refinement to enhance their predictive performance. Looking ahead, the machine learning-driven approach to predicting brain strokes presents numerous opportunities for exploration. Key areas for future development include in-depth feature analysis, exploring advanced model training using deep learning techniques, conducting real-world clinical trials, enhancing model Model Name Accuracy (%) Error(%) Logistic Regression 90 10 Decision Tree 95 5 KnnClassifier 87 13 Gradient Boosting 95 5 Random forest 97 3 interpretability, and ensuring continuous data integration to keep models updated with the latest medical knowledge. Deeper Feature Analysis involves prioritizing variables that significantly influence stroke predictions using techniques like Recursive Feature Elimination or Feature Importance Ranking. Additionally, incorporating emerging biometric data sources, such as wearables and smart devices, could provide more granular data points. Advanced Model Training suggests exploring neural networks, particularly convolutional and recurrent types, for recognizing intricate patterns in large datasets, potentially improving prediction accuracies. Real-world Clinical Trials involve collaborating with medical institutions to deploy models in actual clinical settings, gathering feedback, and making necessary adjustments for diverse environments. Continuous Data Integration emphasizes the need for mechanisms that allow continuous updates to models with new data, research findings, and patient histories to ensure relevance and alignment with current medical knowledge. Ethical Considerations highlight the importance of robust data handling and privacy protocols in the integration of AI and machine learning into healthcare to safeguard patient rights and security. In conclusion, the path forward presents challenges and excitement at the intersection of technology and medicine, promising a future where predictive accuracy becomes a continually refined tool for ensuring optimal healthcare outcomes globally.