American International University-Bangladesh

CSC4232- MACHINE LEARNING

PROJECT TITLE:

Comparative Analysis of Classification Models for Stroke Prediction

Group-2

Group Members	
Name	**Student ID**
ANIK DAS	18-38080-2
A.S.M. FAZLE RABBI	19-39714-1
SALAH-UD-DIN ELIAS KHAN	20-44139-2

SUPERVISED BY: DR. MD. ASRAF ALI

Submission Deadline: 08/05/2023

Executive Summary:

Stroke is a major cause of disability and death globally, with significant economic and social costs. Accurately predicting stroke outcomes can aid in early detection and treatment, leading to better patient outcomes and reduced healthcare costs. Machine learning models can improve stroke prediction accuracy and improve patient care.

In this project, we developed and compared five classification-based models using Logistic Regression, Decision Tree, Naive Bayes, K-Nearest Neighbors, and Support Vector Machine. We collected a dataset from Kaggle, preprocessed it, conducted exploratory data analysis, and selected appropriate model training features.

The accuracy of the five models was evaluated using a confusion matrix. The Support Vector Machine and Logistic Regression models achieved the highest accuracy, with an accuracy of 0.9586. The K-Nearest Neighbors model also performed well, with an accuracy of 0.9532. The Decision Tree and Naive Bayes models had lower accuracy, indicating they were less effective in predicting stroke outcomes.

Our findings demonstrate the potential of machine learning models for stroke prediction, which can aid in the early detection and treatment of stroke. However, a limitation of our study is that the models could have been better trained with more features and samples. Despite the promising results, there are still opportunities to improve the accuracy and reliability of the models for stroke prediction by incorporating additional features or developing more advanced algorithms.

In conclusion, our study highlights the potential of machine learning models for improving stroke prediction accuracy. Future research can focus on improving the models' accuracy and reliability by incorporating additional features, developing more advanced algorithms, and exploring the relationship between features and stroke outcomes. Accurate stroke prediction can help healthcare professionals provide early diagnosis and treatment for stroke, improving patient outcomes and reducing healthcare costs.

1. Project Objective

The objectives of the project are:

- To develop five different classification-based models for stroke prediction using Logistic Regression, Decision Tree, Naive Bayes, K-Nearest Neighbors and Support Vector Machine.
- To compare the model’s accuracy in stroke prediction
- To identify the best model for stroke prediction based on accuracy

2. Project Methodology

The project involved the collection of a dataset from Kaggle and its subsequent preprocessing. The initial preprocessing stage entailed the removal of redundant columns, duplicate rows, and rows containing null values, followed by removing outliers. Subsequently, exploratory data analysis was conducted, which involved the use of relevant plots. String values were replaced with numeric numbers to facilitate computation. Next, a correlation matrix and parallel

coordinates were created to identify the appropriate model training features. The feature matrix and target variable were selected, and the dataset was partitioned into a 70:30 ratio for training and testing. Subsequently, the models were trained, and their accuracy was tested. The accuracy of each model was determined by creating a confusion matrix. Finally, the accuracy of the five models was compared, and the model with the highest accuracy was identified.

2.1 Data Collection Procedure

The data collection procedure for this dataset involved accessing the Kaggle website https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset and searching for the "stroke prediction dataset" created by user "FEDESORIANO". After locating the dataset, it was downloaded to a local computer for further analysis. The dataset contained 12 columns and 5110 unique rows or pieces of information. Each row included information on an individual's age, gender, medical history, lifestyle factors, and stroke status.

The information was collected through medical checkups and tests, including blood glucose level tests, BMI measurements, and smoking status assessments. The dataset was anonymized to ensure patient privacy and confidentiality.

Once the dataset was downloaded, it was stored in a suitable format for analysis, and data preprocessing techniques were applied to prepare it for statistical analysis. This included removing unnecessary columns, checking for duplicates and missing values, and removing any complex data to ensure the integrity of the dataset.

2.2. Data Validation Procedure

To validate the data, we followed some procedure. They are:

1. *Define data validation rules:*

- Age: must be a positive integer between greater than 0 and less than 100.
- Gender: must be either "Male" or "Female".
- Hypertension: must be a binary value (0 or 1).
- Heart disease: must be a binary value (0 or 1).
- Ever married: must be either "Yes" or "No".
- Work type: must be one of the following values: "Private", "Self-employed", "Govt_job", "children", "Never_worked".
- Residence type: must be either "Urban" or "Rural".
- Average glucose level: must be a positive float.
- BMI: must be a positive float.
- Smoking status: must be one of the following values: "formerly smoked", "never smoked", "smokes", "Unknown".
- Stroke: must be a binary value (0 or 1).

2. *Check data accuracy:*

    - Check age values to ensure that they are accurate and there is no outlier.
    - Check average glucose level and BMI values against medical records to ensure that they are accurate.

3. *Check data completeness:*
    - Check for missing values in all fields and handle them appropriately (removal).
    - Ensure that all required fields are present and contain valid values.

4. *Check data consistency:*
    - Check for inconsistencies between fields (e.g., someone being classified as a "child" in the work type field but having a high age).
    - Check for inconsistencies between multiple records for the same person (e.g., different values for age or gender).

2.3 Data Preprocessing and Normalization

In the data preprocessing step, the first stage involved the removal of the unnecessary column containing the patient ID. The dataset was then checked for any duplicate values, and it was observed that there were no duplicates. Next, the dataset was checked for missing values, and the rows containing such values were removed. Additionally, a row containing an outlier was also removed. We chose to remove problematic data rather than replace them because they were very small in number. String values were replaced with numeric numbers to facilitate computation.

2.4 Feature Extraction Procedure

For feature extraction, we used a correlation matrix and parallel coordinates. In the parallel coordinate plot, most feature lines were overlapping, so selecting the appropriate feature for training the models was impossible. While in the correlation matrix, we identified highly positive and negative correlations between features and tried to avoid highly correlated features since they have high chances of generating biased results based on another feature which may result in overfitting of the model. In the end, with a thorough analysis of the feature with sufficient domain knowledge, we could select features that will provide the best possible result. The selected features are gender, hypertension, heart disease, average glucose level, BMI and smoking status.

2.5 Classification Algorithms

A brief explanation of the classification algorithms used in this project is given below:

1. Logistic Regression (LR): Logistic Regression is a type of linear regression used for classification tasks. It predicts the probability of an input belonging to a particular class. Logistic regression is simple to implement and interpret, making it a popular choice for many applications. It is often used in marketing, finance, and healthcare.

2. Decision Tree (DT): A decision Tree is a supervised learning algorithm mostly used in classification problems. It works by creating a tree-like model of decisions and their possible

consequences. Decision Tree is easy to understand and interpret, making it a popular choice in various fields such as finance, healthcare, and customer service.

3. Naive Bayes (NB): Naive Bayes is a probabilistic algorithm used for classification problems. It works on the principle of Bayes' Theorem and assumes that features are independent of each other. Naive Bayes is simple and fast, making it an excellent choice for many text classification tasks, such as spam detection and sentiment analysis.

4. K-Nearest Neighbors (KNN): K-Nearest Neighbors is a supervised learning algorithm used for classification and regression problems. It works by finding the k-nearest neighbours of an input data point and predicting its class based on the majority class of its neighbours. KNN is easy to implement and interpret, making it a popular choice in various fields such as image recognition, customer service, and finance.

5. Support Vector Machine (SVM): SVM is a supervised learning algorithm that can be used for classification and regression problems. It works by finding the hyperplane that best separates the data points into different classes. SVM is known for its ability to handle high-dimensional data and its robustness to noise. It is often used in bioinformatics, finance, and image recognition.

2.6 Data Analysis Techniques

After performing data preprocessing, the dataset was analyzed to gain insights into the relationship between various features and the likelihood of stroke.

Firstly, the count of unique values was determined for each column. The gender column had three possible values, with females being the majority (2994), followed by males (2115) and others (1). The hypertension column had two values, with the majority (4612) not having hypertension and only 498 having hypertension. Similarly, the heart disease column had two values, with the majority (4834) not having a heart disease and only 276 having a heart disease. The ever_married column had two values, with the majority (3353) being married and only 1757 not being married. The work_type column had five possible values, with private jobs being the most common (2925), followed by government jobs (657), self-employed (819), children (687), and never worked (22). The Residence_type column had two values, with the majority (2596) living in urban areas and 2514 living in rural areas. The smoking_status column had four possible values, with the majority (1544) being unknown, followed by never smoked (1892), formerly smoked (885), and smokes (789). Lastly, the stroke column had two values, with the majority (4861) not having a stroke and only 249 having a stroke.

Next, basic statistical analysis was performed to gain insights into the central tendency and dispersion of the data. The mean, standard deviation, minimum, 25th percentile, 50th percentile, 75th percentile, and maximum values were calculated for each numerical column (age, avg_glucose_level, bmi).

Lastly, graphs were plotted to visualize the relationship between the likelihood of stroke and various features. The relationship between BMI and heart disease was plotted, and the graph showed a positive correlation. The relationship between BMI and stroke was plotted, and the graph showed that higher BMI values were associated with a higher likelihood of stroke. Lastly, the relationship between smoking status and stroke was plotted. The graph showed that individuals who currently smoked or formerly smoked had a higher likelihood of stroke than

those who never smoked. These findings suggest that hypertension, heart disease, BMI, and smoking status may be risk factors for stroke, and further investigation is needed.

2.7 Block Diagram and Workflow Diagram of Proposed Model

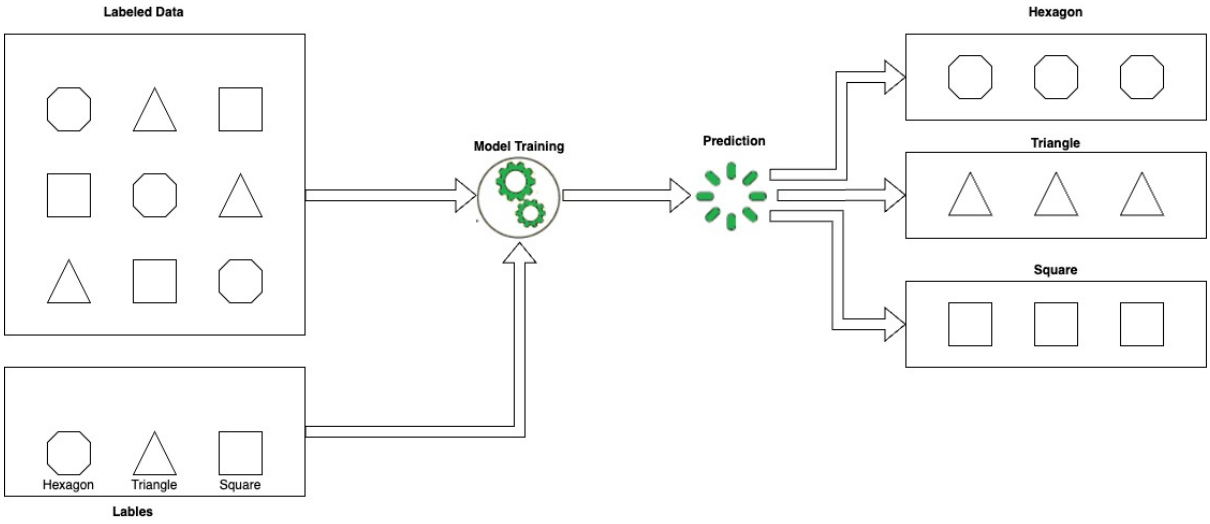

Figure: Block Diagram

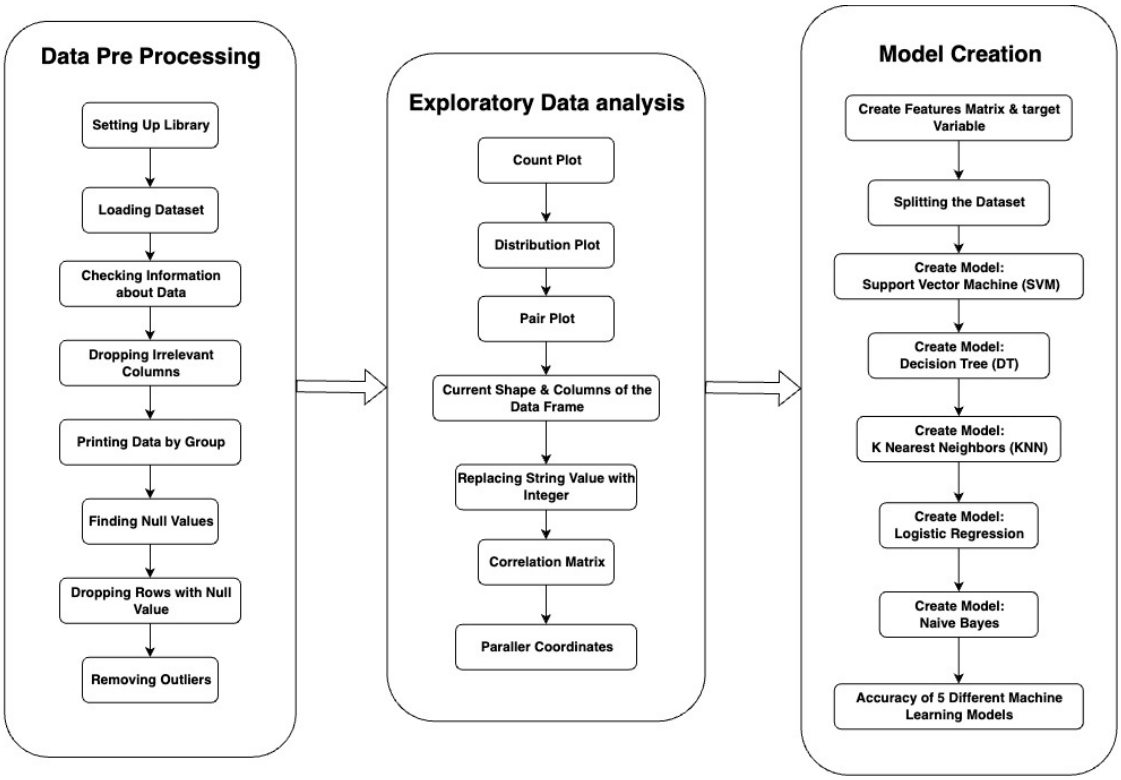

Figure: Workflow Diagram

2.8 Experimental Setup and Implementations

For this research, stroke data was collected from Kaggle. The implementation was done using Python programming language. Python libraries such as NumPy and pandas were used for data processing, and matplotlib and seaborn were used for data visualization. The scikit-learn library was used for creating and building the models.

The first step in the implementation was loading the dataset, followed by data preprocessing using basic Python techniques. Various plots from the matplotlib and Seaborn libraries were used for data visualization and exploratory data analysis. For feature extraction, a correlation matrix and parallel coordinates were used. Most feature lines overlapped in the parallel coordinate plot, making it difficult to select appropriate features for training the models. However, in the correlation matrix, highly positive and negative correlations between features were identified, and highly correlated features were avoided to prevent biased results and overfitting.

After thoroughly analyzing the features with sufficient domain knowledge, the most appropriate features were selected for training the models. The train-test split function from the model selection module of sci-kit-learn was used to split the data for training and testing the models. Scikit-learn was used to create different models and find their accuracy using the metrics module. Finally, the confusion matrix was created using the same module for different models, which provided an overall evaluation of the model's performance.

3. Results and Discussion

The accuracy of the five models trained for stroke prediction was:

Support Vector Machine (SVM): 0.9586 ~ 96%
Decision Tree (DT): 0.924 ~ 92%
K-Nearest Neighbors (KNN): 0.9532 ~ 95%
Logistic Regression (LR): 0.9586 ~ 96%
Naive Bayes (NB): 0.8968 ~ 90%

The results demonstrate that machine learning models can effectively predict stroke outcomes. In particular, the SVM and LR models showed excellent predictive performance, while the KNN and DT models also showed high accuracy. These results are promising, as accurate prediction models can aid in the early detection and treatment of stroke.

Overall, the findings of this project demonstrate the potential of machine learning models for stroke prediction. Further research could focus on improving the accuracy and reliability of these models by incorporating additional features or developing more advanced algorithms. These models could ultimately contribute to improving the diagnosis and treatment of stroke, a critical public health issue.

3.1 Results Comparison

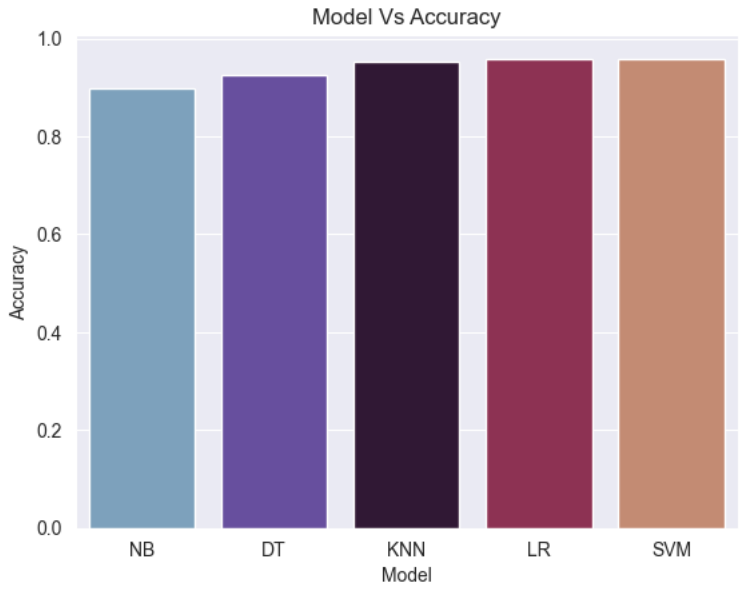

Figure: Model Vs Accuracy

Based on the results of the five models trained for stroke prediction, the Support Vector Machine (SVM) and Logistic Regression (LR) models achieved the highest accuracy, with an accuracy of 0.9586. The K-Nearest Neighbors (KNN) model achieved an accuracy of 0.9532, which was also quite high. The Decision Tree (DT) model had an accuracy of 0.924, while the Naive Bayes (NB) model had the lowest accuracy of 0.8968.

In terms of comparing the models, the SVM and LR models achieved the highest accuracy, indicating they were the most effective at predicting stroke outcomes. The KNN model also performed well but had slightly lower accuracy than the SVM and LR models. The DT and NB models had lower accuracy than the others, indicating they were less effective in predicting stroke outcomes.

3.2 Confusion Matrix Analysis

In this confusion matrix of SVM, we can see that the model predicted 1412 true positives (TP), which means the model correctly identified 1412 instances of a stroke. On the other hand, the model has predicted 0 true negatives (TN) and 0 false positives (FP), which means that the model did not correctly identify any instance where a stroke did not occur. Additionally, the model has predicted 61 false negatives (FN), which means that the model incorrectly identified 61 instances where a stroke occurred as not having a stroke.

Based on the confusion matrix of DT, we can see that the model has predicted 1357 cases of stroke correctly as true positives (TP). The model failed to identify 57 actual cases of stroke, which are false negatives (FN). On the other hand, the model predicted 55 cases as strokes, which were not, resulting in false positives (FP). Finally, there were only 4 cases correctly predicted as no stroke, which are the true negatives (TN).

In the confusion matrix of KNN, we can observe that the model has predicted 1402 true positives (TP), indicating that the model has correctly identified 1402 instances where a stroke

has occurred. The model has also identified 2 true negatives (TN), which means that the model has correctly identified 2 instances where no stroke occurred. However, the model had also predicted 10 false positives (FP), which means that the model incorrectly identified 10 instances as strokes when they were not. Moreover, the model had predicted 59 false negatives (FN), indicating that it had incorrectly identified 59 instances as not having a stroke when they were strokes.

For the LR confusion matrix, we can see that the model has predicted 1412 true positives (TP), which means that the model has correctly identified 1412 instances where a stroke occurred. On the other hand, the model has predicted 0 true negatives (TN) and 0 false positives (FP), which means that the model did not correctly identify any instance where a stroke did not occur. However, the model has predicted 61 false negatives (FN), which means that the model incorrectly identified 61 instances where a stroke occurred as not having a stroke.

In the confusion matrix for NB, we can see that the model has predicted 1301 true positives (TP), which means the model correctly identified 1301 instances where a stroke occurred. The model has predicted 20 true negatives (TN), correctly identifying instances where a stroke did not occur. However, the model has predicted 111 false positives (FP), which means that the model incorrectly identified 111 instances where there was no stroke as having a stroke. Additionally, the model has predicted 41 false negatives (FN), which means that the model incorrectly identified 41 instances where a stroke occurred as not having a stroke.

3.3 Graphical Representation of Results

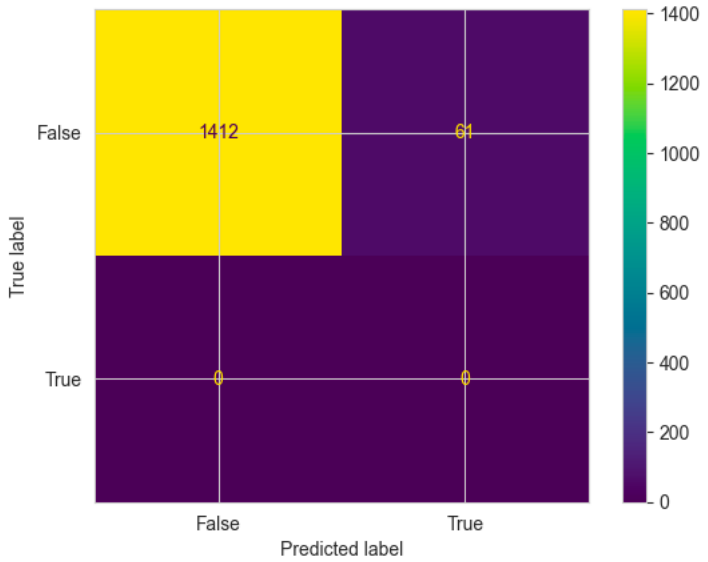

Figure: Confusion Matrix for SVM

In this case, the confusion matrix indicates that the model correctly identified 1412 instances as stroke (TP) and correctly identified 0 instances as no stroke (TN). However, the model incorrectly classified 61 instances as no stroke (FN), and there were no false positive (FP) cases.

Based on this confusion matrix, we can calculate various performance metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of the classification model in predicting strokes.
Accuracy = (TP + TN) / (TP + TN + FP + FN) = (1412 + 0) / (1412 + 0 + 0 + 61) ≈ 0.959 or 95.9%
The accuracy of the model is high, indicating that the model was able to predict strokes correctly for the majority of the cases.

Precision = TP / (TP + FP) = 1412 / (1412 + 0) = 1.0 or 100%
The precision of the model is perfect, which means that all the instances predicted as strokes were actually strokes.

Recall (Sensitivity) = TP / (TP + FN) = 1412 / (1412 + 61) ≈ 0.96 or 96%
The recall score of the model is high, indicating that it was able to identify most of the actual strokes.

F1 score = 2 * (precision * recall) / (precision + recall) = 2 * (1.0 * 0.96) / (1.0 + 0.96) ≈ 0.98 or 98%
The F1 score of the model is also high, indicating that the model has good overall performance in predicting strokes.

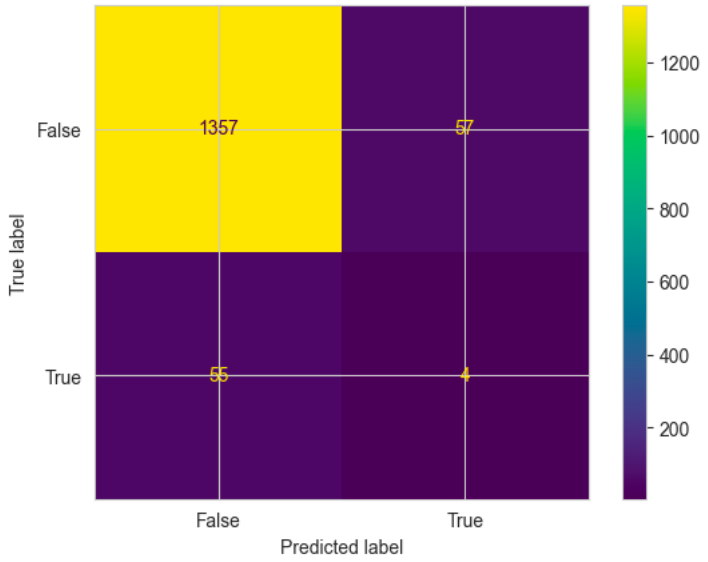

Figure: Confusion Matrix for DT

In this case, the confusion matrix indicates that the model correctly identified 1357 instances as stroke (TP) and correctly identified 4 instances as no stroke (TN). However, the model incorrectly classified 55 instances as stroke when they were actually no stroke (FP), and there were 57 false negative (FN) cases.

Based on this confusion matrix, we can calculate various performance metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of the classification model in predicting strokes.

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (1357 + 4) / (1357 + 4 + 55 + 57) ≈ 0.924 or 92.4%
The accuracy of the model is high, indicating that the model was able to predict strokes correctly for the majority of the cases.
Precision = TP / (TP + FP) = 1357 / (1357 + 55) ≈ 0.961 or 96.1%
The precision of the model is high, indicating that most of the instances predicted as strokes were actually strokes.

Recall (Sensitivity) = TP / (TP + FN) = 1357 / (1357 + 57) ≈ 0.959 or 95.9%
The recall score of the model is also high, indicating that it was able to identify most of the actual strokes.

F1 score = 2 * (precision * recall) / (precision + recall) = 2 * (0.961 * 0.959) / (0.961 + 0.959) ≈ 0.96 or 96%
The F1 score of the model is high, indicating that the model has good overall performance in predicting strokes.

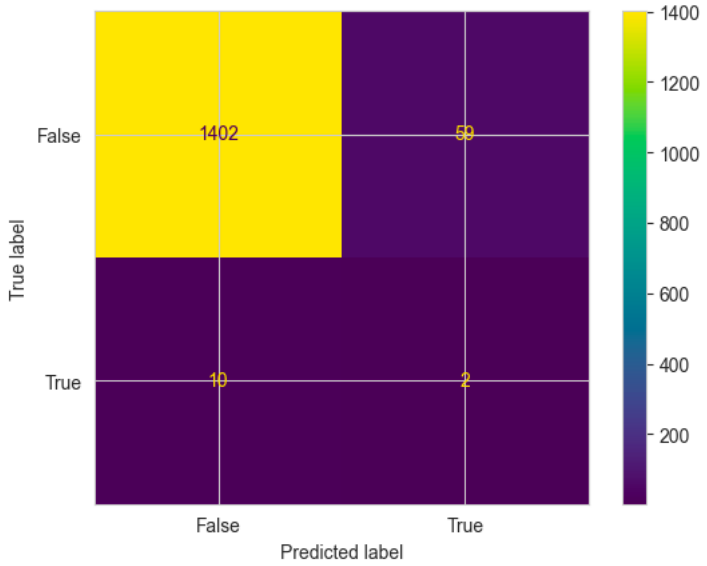

Figure: Confusion Matrix for KNN

In this case, the confusion matrix indicates that the model correctly identified 1402 instances as stroke (TP), and 2 instances as no stroke (TN). However, the model incorrectly classified 10 instances as stroke when they were actually no stroke (FP), and 59 instances as no stroke when they were actually stroke (FN).

Based on this confusion matrix, we can calculate various performance metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of the classification model in predicting strokes.

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (1402 + 2) / (1402 + 2 + 10 + 59) ≈ 0.953 or 95.3%
The accuracy of the model is high, indicating that the model was able to predict strokes correctly for the majority of the cases.

Precision = TP / (TP + FP) = 1402 / (1402 + 10) ≈ 0.993 or 99.3%
The precision of the model is high, indicating that almost all instances predicted as strokes were actually strokes.
Recall (Sensitivity) = TP / (TP + FN) = 1402 / (1402 + 59) ≈ 0.959 or 95.9%
The recall score of the model is also high, indicating that it was able to identify most of the actual strokes.

F1 score = 2 * (precision * recall) / (precision + recall) = 2 * (0.993 * 0.959) / (0.993 + 0.959) ≈ 0.976 or 97.6%
The F1 score of the model is high, indicating that the model has good overall performance in predicting strokes.

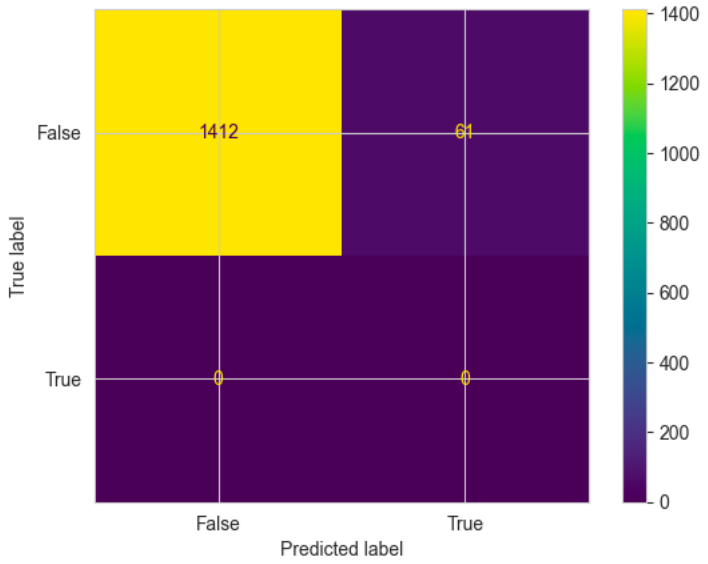

Figure: Confusion Matrix for LR

In this case, the confusion matrix indicates that the model correctly identified 1412 instances as stroke (TP), but incorrectly classified all other instances as no stroke. Specifically, the model did not classify any instance as no stroke (TN = 0) and did not make any false positive predictions (FP = 0). However, there were 61 false negative (FN) cases, where the model predicted that a patient did not have a stroke, but the patient actually had a stroke.

Based on this confusion matrix, we can calculate various performance metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of the classification model in predicting strokes.

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (1412 + 0) / (1412 + 0 + 0 + 61) ≈ 0.959 or 95.9%
The accuracy of the model is high, indicating that the model was able to predict strokes correctly for the majority of the cases.

Precision = TP / (TP + FP) = 1412 / (1412 + 0) = 1.0 or 100%

The precision of the model is perfect, indicating that all instances predicted as strokes were actually strokes.

Recall (Sensitivity) = TP / (TP + FN) = 1412 / (1412 + 61) ≈ 0.959 or 95.9%
The recall score of the model is also high, indicating that it was able to identify most of the actual strokes.

F1 score = 2 * (precision * recall) / (precision + recall) = 2 * (1.0 * 0.959) / (1.0 + 0.959) ≈ 0.979 or 97.9%
The F1 score of the model is high, indicating that the model has good overall performance in predicting strokes.

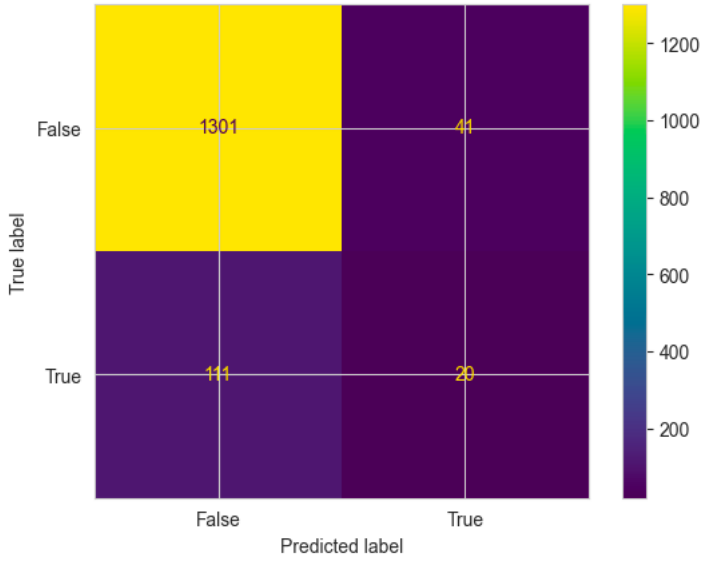

Figure: Confusion Matrix for NB

In this case, the confusion matrix indicates that the model correctly identified 1301 instances as stroke (TP), and 20 instances as no stroke (TN). However, the model incorrectly classified 111 instances as stroke when they were actually no stroke (FP), and 41 instances as no stroke when they were actually stroke (FN).

Based on this confusion matrix, we can calculate various performance metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of the classification model in predicting strokes.

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (1301 + 20) / (1301 + 20 + 111 + 41) ≈ 0.896 or 89.6%
The accuracy of the model is relatively high, indicating that the model was able to predict strokes correctly for the majority of the cases.

Precision = TP / (TP + FP) = 1301 / (1301 + 111) ≈ 0.921 or 92.1%
The precision of the model is moderate, indicating that a large number of instances predicted as strokes were actually strokes, but some instances were incorrectly classified as strokes.

Recall (Sensitivity) = TP / (TP + FN) = 1301 / (1301 + 41) ≈ 0.969 or 96.9%
The recall score of the model is high, indicating that it was able to identify most of the actual strokes.

F1 score = 2 * (precision * recall) / (precision + recall) = 2 * (0.921 * 0.969) / (0.921 + 0.969) ≈ 0.944 or 94.4%
The F1 score of the model is relatively high, indicating that the model has good overall performance in predicting strokes.

In summary, based on the confusion matrix and performance metrics, we can conclude that the classification model performs well in predicting strokes, but there is room for improvement, particularly in reducing the false positive rate. It is important to note that a single confusion matrix may not provide a complete understanding of the model's performance, and further analysis and evaluation may be required to validate the model's effectiveness.

4. Conclusion and Future Recommendations

In conclusion, this study aimed to predict stroke outcomes using machine learning models. The dataset was collected from Kaggle and preprocessed to remove the duplicate, null values, and outliers. The appropriate features for training the models were selected using a correlation matrix and parallel coordinates, and the dataset was split into 70:30 for training and testing the models. The accuracy of five models, SVM, DT, KNN, LR, and NB, was evaluated, and the SVM and LR models achieved the highest accuracy. The findings demonstrate the potential of machine learning models for stroke prediction, which can contribute to improving the diagnosis and treatment of stroke.

However, a limitation of this study is that the machine learning models could have been better trained if more features and samples were available. Despite the promising results, there are still opportunities to improve the accuracy and reliability of the models for stroke prediction by incorporating additional features or developing more advanced algorithms. Cross-validation and standard error could also be used to increase the acceptance of the model. Furthermore, further exploration of the relationship between features and stroke outcomes can provide valuable insights into stroke development and prevention mechanisms. Ultimately, improving the accuracy of the models can help healthcare professionals provide early diagnosis and treatment for stroke, improving patient outcomes and reducing healthcare costs.

Appendixes:

Setting up library:

```
import numpy as np
import pandas as pd

# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# For checking the model accuracy
from sklearn import metrics
```

```
#To plot the graph embedded in the notebook
%matplotlib inline
```

Loading Dataset:

```
#  Loading CSV file into the dataframe
Stroke_df = pd.read_csv('Stroke_Data.csv')
Stroke_df
```

Checking information about Data:

```
Stroke_df.info()

Stroke_df.describe()

#Printing some important group size information
print(Stroke_df.groupby('gender').size(),"\n")
print(Stroke_df.groupby('hypertension').size(),"\n")
print(Stroke_df.groupby('heart_disease').size(),"\n")
print(Stroke_df.groupby('ever_married').size(),"\n")
print(Stroke_df.groupby('work_type').size(),"\n")
print(Stroke_df.groupby('Residence_type').size(),"\n")
print(Stroke_df.groupby('smoking_status').size(),"\n")
print(Stroke_df.groupby('stroke').size(),"\n")

#Printing Shape and Column of Data Frame
print('Shape: ',Stroke_df.shape)
print('Columns: ',Stroke_df.columns)
```

Data Preprocessing:

```
# Droping id Column
Stroke_df = Stroke_df.drop(Stroke_df.iloc[:,[0]], axis = 1)

# Droping duplicate rows and printing updated Data Frame
Stroke_df.drop_duplicates(inplace= True)

# Finding the sum of null value in each column
Stroke_df.isnull().sum()

# Finding the percentage of null value in each column
Stroke_df.isnull().sum()/len(Stroke_df)

# Droping Null Value and loading Updated Dataset
Stroke_df = Stroke_df.dropna()

# Removing one row that has Other gender
for i in Stroke_df.index:
    if Stroke_df.loc[i,'gender']=='Other':
```

```
    Stroke_df.drop(i,inplace=True)

# Replacing String Value with integer and printing updated Dataset ever_married
Stroke_df['gender'] = Stroke_df['gender'].replace(['Female','Male'], [0,1])
Stroke_df['ever_married'] = Stroke_df['ever_married'].replace(['No','Yes'], [0,1])
Stroke_df['work_type'] =
Stroke_df['work_type'].replace(['Govt_job','Never_worked','Private','Self-
employed','children'], [0,1,2,3,4])
Stroke_df['Residence_type'] = Stroke_df['Residence_type'].replace(['Rural','Urban'], [0,1])
Stroke_df['smoking_status'] = Stroke_df['smoking_status'].replace(['formerly smoked','never
smoked','smokes','Unknown'], [0,1,2,3])
```

Data Visualization:

```
# Count Plot
plt.figure(figsize=(20,12))

plt.subplot(3, 3, 1)
sns.countplot(data=Stroke_df,x='gender',palette='twilight')

plt.subplot(3, 3, 2)
sns.countplot(data=Stroke_df,x='hypertension',palette='twilight')

plt.subplot(3, 3, 3)
sns.countplot(data=Stroke_df,x='heart_disease',palette='twilight')

plt.subplot(3, 3, 4)
sns.countplot(data=Stroke_df,x='ever_married',palette='twilight')

plt.subplot(3, 3, 5)
sns.countplot(data=Stroke_df,x='work_type',palette='twilight')

plt.subplot(3, 3, 6)
sns.countplot(data=Stroke_df,x='Residence_type',palette='twilight')

plt.subplot(3, 3, 7)
sns.countplot(data=Stroke_df,x='smoking_status',palette='twilight')

plt.subplot(3, 3, 8)
sns.countplot(data=Stroke_df,x='stroke',palette='twilight')

# Distribution Plot of Age and Flight Distance
sns.set_style ('whitegrid')
plt.figure(figsize=(17,8))

sns.displot(Stroke_df['age'], bins=30, color='black')
plt.title("Data distribution of Age");

sns.displot(Stroke_df['avg_glucose_level'], bins=30, color='black')
plt.title("Data distribution of Average glucose Level");
```

```
sns.displot(Stroke_df['bmi'], bins=30, color='black')
plt.title("Data distribution of BMI");

# Pair Plot of the colums
sns.pairplot(data=Stroke_df)

# Age vs stroke
sns.lineplot(x="age", y="stroke",hue='gender' ,data=Stroke_df)

# Setting title of the plot
plt.title("Age vs stroke");

# Bar plot for Gender vs stroke
sns.barplot(y=Stroke_df['gender'], x=Stroke_df['stroke'])

# Setting title of the plot
plt.title("Gender vs Stroke");

# Violin plot for Smoking vs Stroke
sns.violinplot(x="smoking_status",y="stroke",data=Stroke_df)

# Setting title of the plot
plt.title("Smoking vs Stroke");

# Between feature variables and target varable
plt.figure(figsize=(14,10))

# Relatiosnship in bmi vs heart_disease
plt.subplot(2, 3, 1)
sns.scatterplot(x=Stroke_df['bmi'], y=Stroke_df['heart_disease'])
plt.title("bmi vs heart_disease");

# Relatiosnship in bmi vs stroke
plt.subplot(2, 3, 2)
sns.scatterplot(x=Stroke_df['bmi'], y=Stroke_df['stroke'])
plt.title("bmi vs stroke", color='green');

# Ploting Parallel cordinates
from pandas.plotting import parallel_coordinates
plt.figure(figsize = (45, 8))
parallel_coordinates(Stroke_df, "stroke", color = ['blue', 'green']);

# corr() to calculate the correlation between variables
correlation_matrix = Stroke_df.corr().round(2)
# changing the figure size
plt.figure(figsize = (8, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True);
# corr() to calculate the correlation between variables
correlation_matrix = Stroke_df.corr().round(2)
```

```
# Steps to remove redundant values
# Return a array filled with zeros
mask = np.zeros_like(correlation_matrix)
# Return the indices for the upper-triangle of array
mask[np.triu_indices_from(mask)] = True
# Changing the figure size
plt.figure(figsize = (8, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True, mask=mask);
```

Features Matrix & Target Variable:

```
# Feature matrix
X =
Stroke_df[['gender','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]

# Target Variable
y= Stroke_df['stroke']
```

Splitting Data for Training and Testing:

```
# Import the `train_test_split` function from the `model_selection` module of the `sklearn` library
from sklearn.model_selection import train_test_split

# Split the feature set `X` and the target set `y` into random train and test subsets
# `test_size=0.3` means that 30% of the data is used for testing and 70% is used for training
# `random_state=10` sets the random seed to 10, so that the split is reproducible
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

# Print the shape of the training feature set
print("Feature train shape: ", X_train.shape)

# Print the shape of the test feature set
print("Feature test shape: ", X_test.shape)

# Print the shape of the training target set
print("Target train shape: ", y_train.shape)

# Print the shape of the test target set
print("Target test shape: ", y_test.shape)
```

Model Creation and accuracy:

```
# importing the necessary package to use the classification algorithm
from sklearn import svm #for Support Vector Machine (SVM) Algorithm

model_svm = svm.SVC() #select the algorithm
model_svm.fit(X_train, y_train) #train the model with the training dataset
```

```
y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print("---------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("---------------------------------")

# save the accuracy score
score = set()
score.add(('SVM', score_svm))

# Importing the necessary package to use the classification algorithm
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
model_dt = DecisionTreeClassifier(random_state=4)
model_dt.fit(X_train, y_train) #train the model with the training dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained model
# Checking the accuracy of the algorithm.
# By comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
# Save the accuracy score
score.add(('DT', score_dt))

# Importing the necessary package to use the classification algorithm
from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
# From sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for putting the new data
into a class
model_knn.fit(X_train, y_train) #train the model with the training dataset
y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the trained model
# Checking the accuracy of the algorithm.
# By comparing predicted output by the model and the actual output
score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("---------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("---------------------------------")
# Save the accuracy score
score.add(('KNN', score_knn))

# Importing the necessary package to use the classification algorithm
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train) #train the model with the training dataset
y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the trained model
# Checking the accuracy of the algorithm.
# By comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
```

```
print("---------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("---------------------------------")
# Save the accuracy score
score.add(('LR', score_lr))

# Importing the necessary package to use the classification algorithm
from sklearn.naive_bayes import GaussianNB
model_nb = GaussianNB()
model_nb.fit(X_train, y_train) #train the model with the training dataset
y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the trained model
#  Checking the accuracy of the algorithm.
# By comparing predicted output by the model and the actual output
score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
print("---------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("---------------------------------")
# Save the accuracy score
score.add(('NB', score_nb))

# Printing the accurecy of 5 different models from score set using for loop
print("The accuracy scores of different Models:")
print("----------------------------------------")

for s in score:
 print(s)

# Inserting the score into a dataset and arrenging it in assending order
accuracy_df = pd.DataFrame(score)
accuracy_df.rename(columns = {0:'Model',1:'Accuracy'}, inplace = True)
accuracy_df = accuracy_df.sort_values('Accuracy')
accuracy_df

# Ploting model vs accuracy barplot
sns.set_style ('darkgrid')
sns.barplot(data= accuracy_df,x='Model',y='Accuracy',palette='twilight')
plt.title('Model Vs Accuracy')
```

Confusion Matrix:

```
# Confusion Matrix for Support Vector Machine
confusion_matrix = metrics.confusion_matrix(y_prediction_svm, y_test)
confusion_matrix
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
cm_display.plot()

# Confusion Matrix for Decision Tree
confusion_matrix = metrics.confusion_matrix(y_prediction_dt, y_test)
confusion_matrix
```

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
cm_display.plot()

# Confusion Matrix for K Nearest Neighbour
confusion_matrix = metrics.confusion_matrix(y_prediction_knn, y_test)
confusion_matrix
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
cm_display.plot()

# Confusion Matrix for Logistic Regression
confusion_matrix = metrics.confusion_matrix(y_prediction_lr, y_test)
confusion_matrix
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
cm_display.plot()

# Confusion Matrix for Naive Bayes Model
confusion_matrix = metrics.confusion_matrix(y_prediction_nb, y_test)
confusion_matrix
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
cm_display.plot()
```