



A graduate research project entitled:

**Brain Stroke Analysis &  
Prediction Using Machine Learning**

By:

Ali Mohamed Taha Hasanin

Omar Muhammad Abdrabo Abdrabo

Amr Zidan El-Rawy Muhammad

Abdullah Muhammad Abdullah Ahmed

Abdel-Hakam Ashraf Mohamed Fouad

Supervised By:  
Prof. Nora Hassan khalil

**2025/2024**

## الغلاف الداخلى لنسخة الكنترول

الاسم: علي محمد طه حسانين  
عمر محمد عبد ربه عبد ربه  
عبد الله محمد عبد الله احمد  
عمرو زيدان الراوي محمد  
عبد الحكم أشرف محمد فؤاد محمد

الفرقة: الرابعة

الشعبة: إحصاء وحاسب

المشرف علي المشروع: أ.د. نورا حسن خليل

لجنة الحكم والمناقشة:

1.

2.

3.

المجموع الكلي	شفهي	تحريري
100	30	70



# **ACKNOWLEDGMENTS**

# **Acknowledgement**

First and foremost, we would express our deepest gratitude and appreciation to our advisor Dr. Nora Hassan khalil for her support, outstanding guidance, and encouragement throughout our graduation project.

We would also like to thank our families, especially our parents, for their encouragement, patience, and assistance over the years. We are forever indebted to our parents, who have always kept us in their prayers and pushed us to Success. Finally, for the developers and contributors of the libraries and frameworks used in the code, including but not limited to NumPy, pandas, Matplotlib, Seaborn, scikit-learn and among others. These open-source tools have provided a solid foundation for data analysis, preprocessing, modeling and evaluation.



# **ABSTRACT**

# **Abstract**

Stroke is a leading cause of death and disability worldwide. Early prediction can significantly improve patient outcomes. This project investigated the application of machine learning for stroke risk prediction.

We analyzed a stroke prediction dataset to identify relevant risk factors and their correlations with stroke incidence. Based on this analysis, we explored and compared the performance of various machine learning algorithms, including Logistic Regression, Support Vector Machine (SVM), Random Forest, and K-Nearest Neighbors (KNN).

The evaluation revealed that the Random Forest model achieved the highest accuracy in predicting stroke risk categories (low, high). This model was then integrated into a user-friendly Graphical User Interface (GUI) application. The application allows users to input their data and receive a prediction of their stroke risk category.

Our findings suggest that machine learning, particularly Random Forest algorithms, can be a valuable tool for stroke risk prediction. The developed GUI application has the potential to empower individuals to take a proactive approach to stroke prevention by raising awareness of their risk factors.

However, limitations exist. The project utilized a single dataset, and the model predicts risk categories only. Future work could explore incorporating larger and more diverse datasets, potentially leading to improved model accuracy. Additionally, exploring prediction of specific stroke types could enable more targeted preventative measures. It is crucial to emphasize that the application is for informational purposes only and should not replace professional medical advice.



# **TABLE OF CONTENTS**

## Table of Contents

<b>Chapter 1: Introduction</b> .....	10
1.1 Stroke Overview .....	10
1.2 Analysis of Risk factors .....	11
1.3 Logistic Regression.....	17
1.4 Support Vector Machine .....	22
1.5 Random Forest .....	25
1.6 K-Nearest Neighbors .....	28
<b>Chapter 2: Implementation</b> .....	34
2.1 Import libraries .....	34
2.2 Data Exploration .....	35
2.3 Dataset Summary .....	37
2.4 Preprocessing .....	39
2.5 Models Creation.....	44
<b>Chapter 3: Results and Conclusion</b> .....	49
3.1 Performance .....	49
3.2 Accuracy Comparison.....	50
3.3 Application.....	59
3.4 Conclusion .....	63
References .....	65





# **CHAPTER 1**

## **INTRODUCTION**

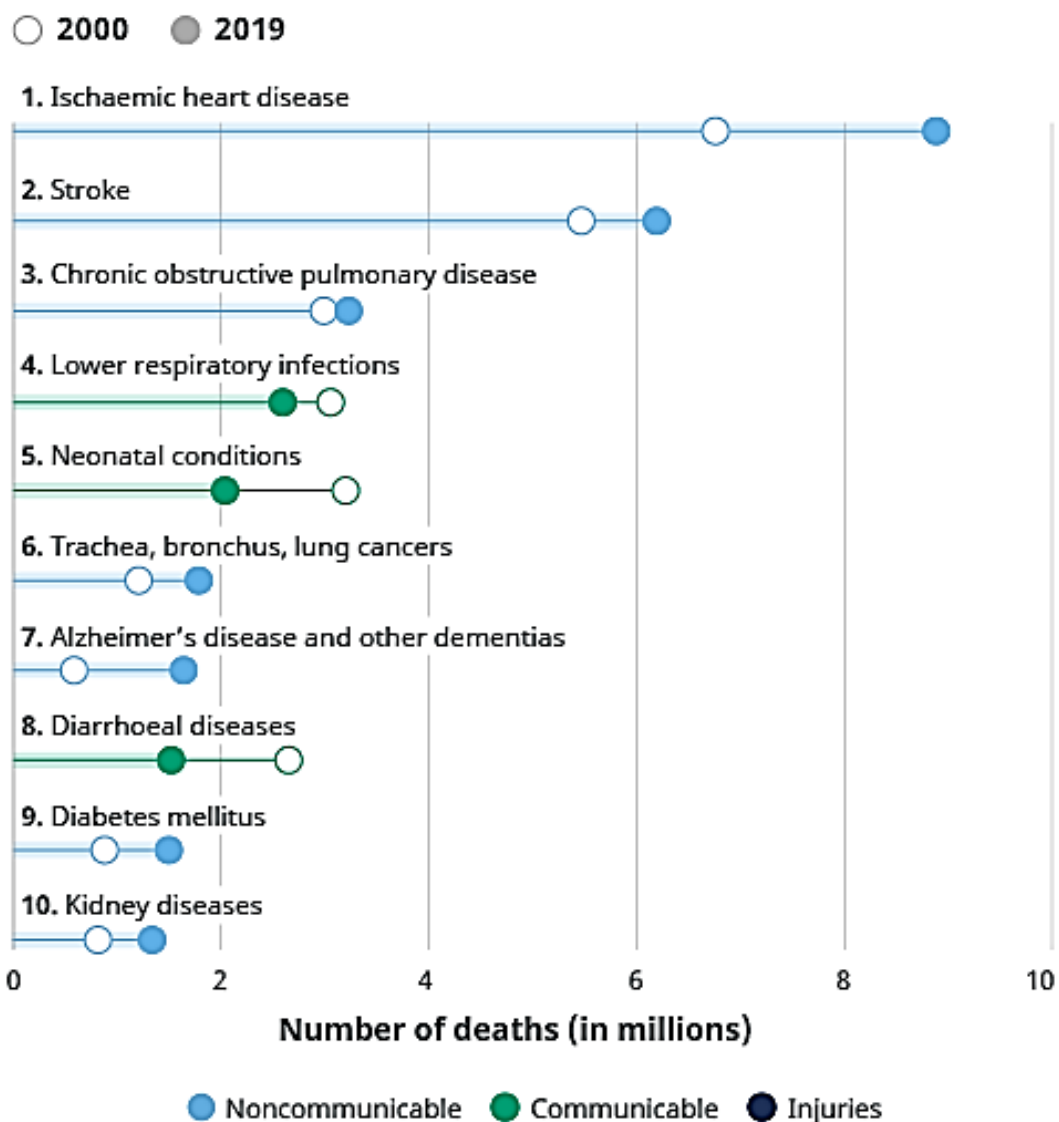
# Chapter 1

## Introduction

### 1.1 Stroke Overview

Stroke is a significant health issue worldwide, Globally, stroke is the second-leading cause of death and a major cause of disability. (World Health Organization)

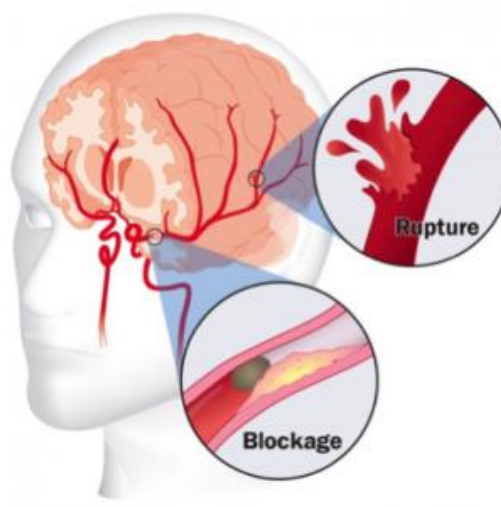
#### Leading causes of death globally



Source: WHO Global Health Estimates.

Approximately 1 in 4 adults worldwide will experience a stroke at some point in their lifetime. Also In Egypt, the overall crude prevalence rate of stroke is high (963/100,000 inhabitants), and the incidence of stroke annually is approximately 150,000–210,000.

characterized by a sudden interruption of blood supply to the brain, or when a blood vessel in the brain bursts. brain cells start to die **within minutes because** they can't get oxygen.



resulting in the loss of brain function. It is a leading cause of disability and mortality globally, making it a major public health concern. Early detection and understanding the prevalence, impact, and associated risk factors of stroke is crucial for effective prevention, intervention, and improved patient outcomes. Traditional methods rely on clinical signs and symptoms, which may be delayed or absent.

## 1.2 Analysis of Risk factors

Risk factors are medical conditions or lifestyle practices that can increase one's chance of having a stroke.

Risk factors can be modifiable (things we can change) or non-modifiable (things we cannot change).

Non-modifiable risk factors include age, race, and gender.

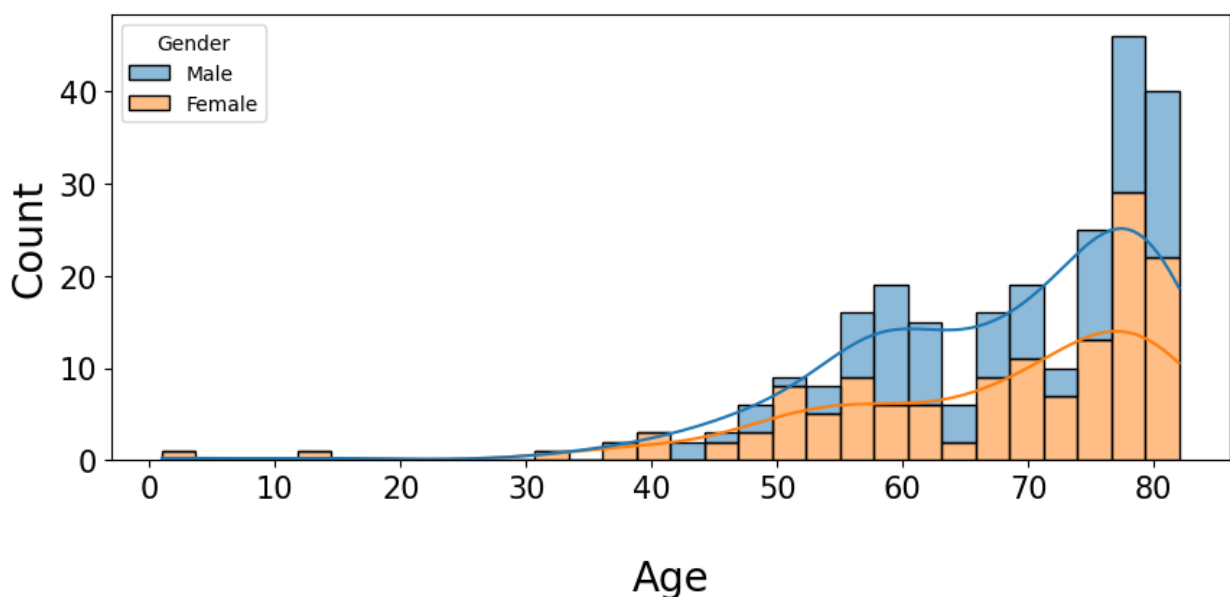
Hypertension (high blood pressure) is the most important modifiable risk factor for both types of strokes.

Other important risk factors include tobacco use, diabetes, high cholesterol, lack of physical activity, unhealthy diet, drug use, and excess alcohol intake. Atrial fibrillation, an abnormal heart rhythm that causes clots to develop in the heart is another important risk factor for stroke, especially among those of older age.

This analysis reviews and discuss key risk factors associated with stroke and explores the current understanding of how these factors contribute to stroke occurrence. We will analyze patients' data. The data will likely include features such as Age, Gender, Marital Status, BMI, Occupation Type, Residence Type, Smoking Status, Hypertension, Heart Disease, Average Glucose Level

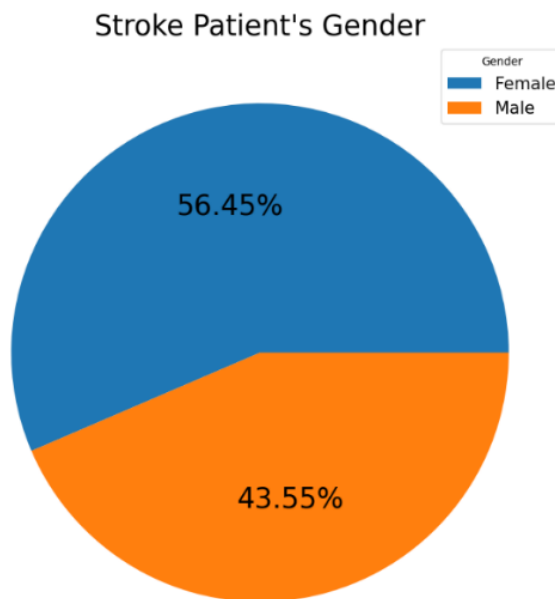
**Age:** Advanced age is associated with physiological changes such as vascular stiffness, atherosclerosis, and decreased cerebral blood flow, which contribute to an increased risk of stroke.

### Stroke Patient's Age Distribution



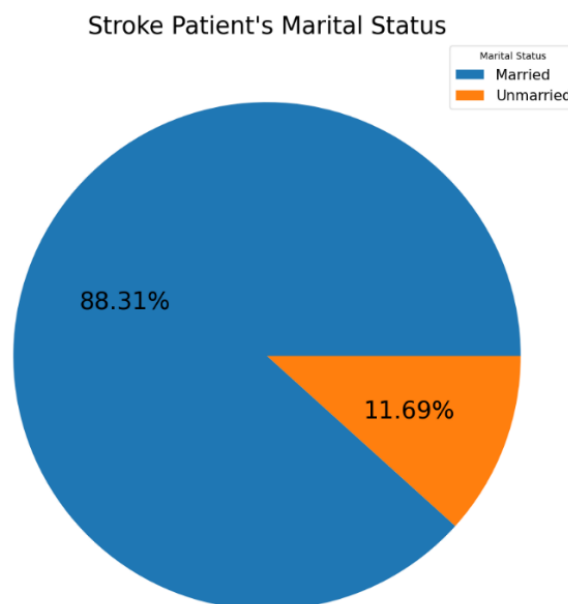
The prevalence of stroke increases with age, with **around 87% of strokes occurring in people over 65 years old**. Also, there are some young and children female stroke patients too.

## Gender:



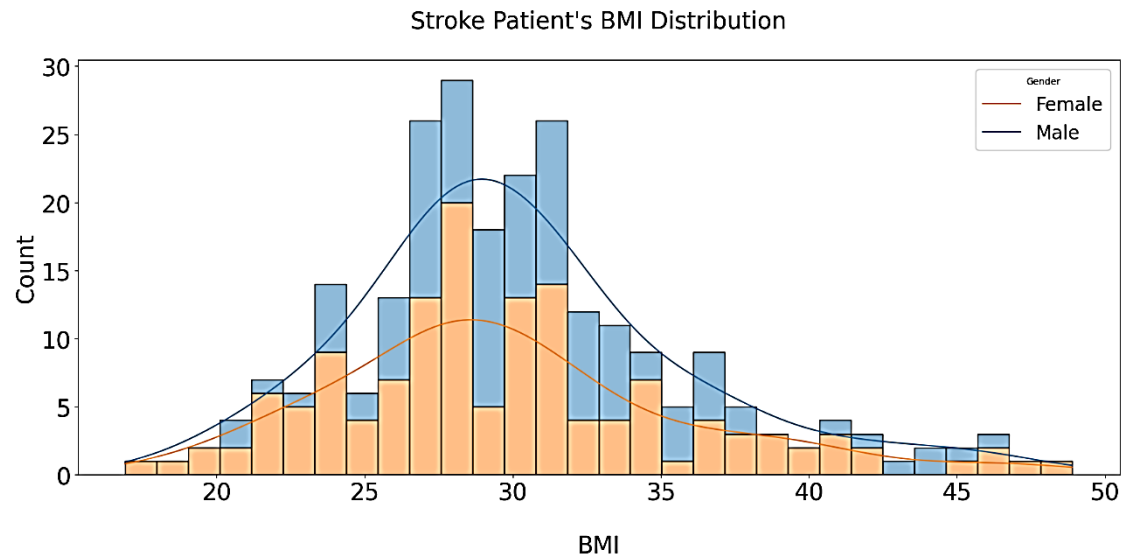
Most of the stroke patients are Female with a ratio of 56.45% followed by Male with a ratio of 43.55%

## Marital:



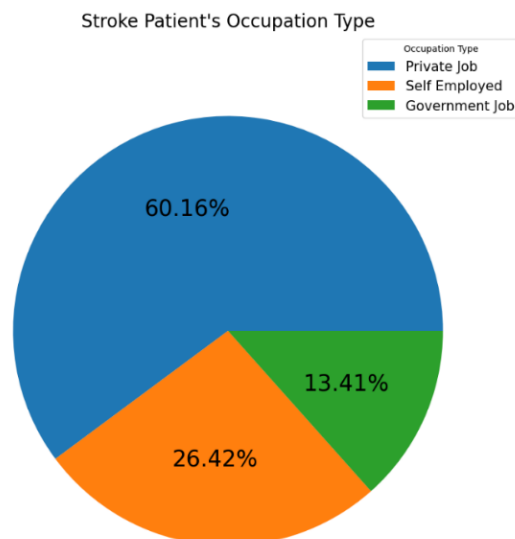
Most of the stroke patients are Married with a ratio of 88.31% followed by Unmarried with a ratio of 11.00%

**BMI:** Obesity contributes to metabolic syndrome, insulin resistance, dyslipidemia, and inflammation, all of which are risk factors for stroke.



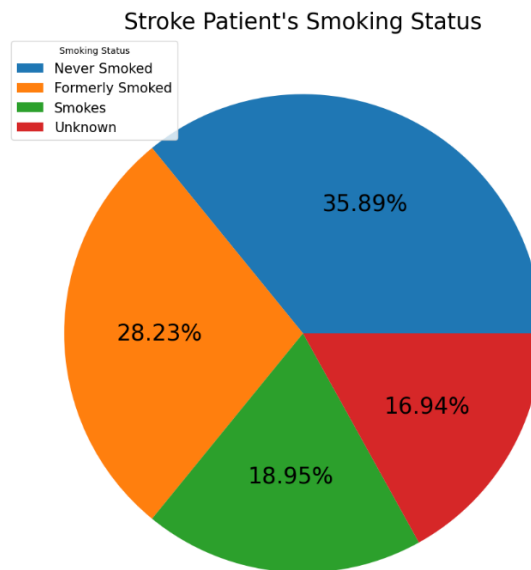
We can see the stroke patient's BMI distribution is right skewed. Most of the patient's BMI fall in between 25 to 35. Also, there are some high BMI values too.

### Occupation:



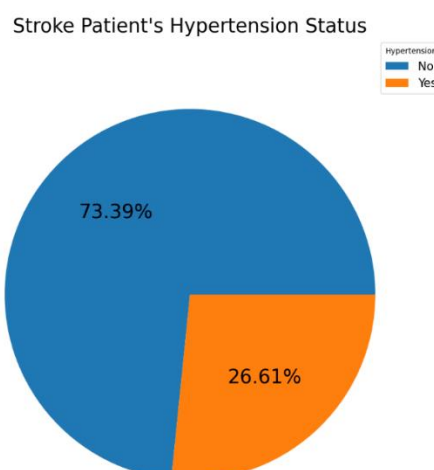
Most of the stroke patients have experience of Private Job with a ratio of 60.16%. Only one patient is children that's why it was not included in the donut chart

**Smoking:** Smoking promotes atherosclerosis, increases blood pressure, impairs endothelial function, and enhances platelet aggregation, all of which contribute to stroke risk.



Most of the stroke patients have Never Smoked with a ratio of 35.89%. Some of the stroke patients have Smoked Previously with a ratio of 28.23%. For some patients, the smoking status is unknown

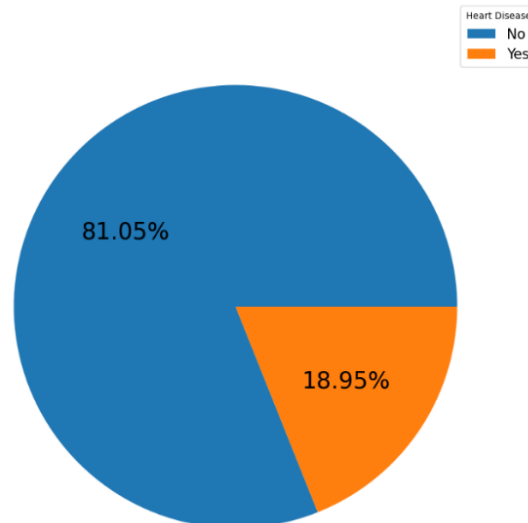
**Hypertension:** Chronic hypertension leads to damage and narrowing of blood vessels, increasing the risk of atherosclerosis, thrombosis, and hemorrhage, all of which can precipitate stroke.



Most of the stroke patients do not have hypertension. Only 26.61% patients have hypertension

**Heart disease:** Atrial fibrillation, in particular, is associated with a fivefold increase in the risk of stroke due to the formation of blood clots in the heart that can travel to the brain.

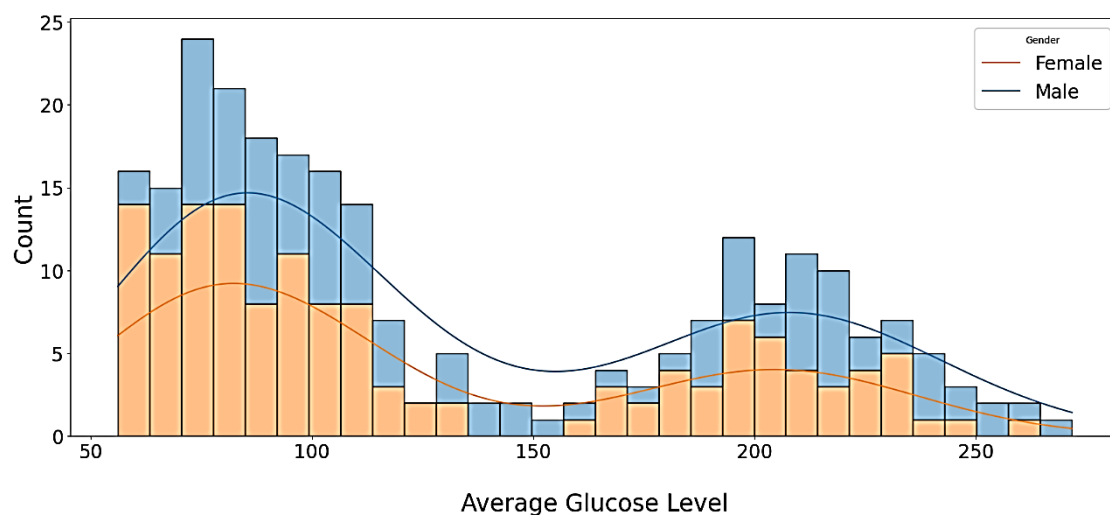
Stroke Patient's Heart Disease



Most of the stroke patients do not have heart disease. Only 18.95% patients have heart disease

### Average glucose levels:

Stroke Patient's Average Glucose Level Distribution



We can see most of the patient's average glucose levels fall in between 60 to 120. Also, there are some high average glucose levels too.



## 1.3 Logistic Regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

### Types of Logistic Regression

#### Binary logistic regression

Binary logistic regression is used to predict the probability of a binary outcome, such as yes or no, true or false, or 0 or 1. For example, it could be used to predict whether a customer will churn or not, whether a patient has a disease or not, or whether a loan will be repaid or not.

#### Multinomial logistic regression

Multinomial logistic regression is used to predict the probability of one of three or more possible outcomes, such as the type of product a customer will buy, the rating a customer will give a product, or the political party a person will vote for.

#### Ordinal logistic regression

is used to predict the probability of an outcome that falls into a predetermined order, such as the level of customer satisfaction, the severity of a disease, or the stage of cancer.

### Assumptions of Logistic regression

Logistic regression is a statistical method commonly used to analyze data with binary outcomes (yes/no, 1/0) and identify the relationship between those outcomes and independent variables. Here are some key assumptions for logistic regression:

#### Data Specific:

- **Binary Dependent Variable:** Logistic regression is designed for binary dependent variables. If your outcome has more than two categories, you might need a multinomial logistic regression or other classification techniques.
- **Independent Observations:** The data points should be independent of each other. This means no repeated measurements or clustering within the data.

### Relationship between Variables:

- **Linearity in the Logit:** The relationship between the independent variables and the logit of the dependent variable ( $\ln(p / (1-p))$ ) is assumed to be linear. This doesn't necessarily mean the outcome itself has a linear relationship with the independent variables, but the log-odds do.
- **No Multicollinearity:** Independent variables shouldn't be highly correlated with each other. Multicollinearity can cause instability in the model and make it difficult to interpret the coefficients.

### Other:

- **Absence of Outliers:** While not a strict requirement, outliers can significantly influence the model. It's important to check for and address any outliers that might distort the results.
- **Adequate Sample Size:** Logistic regression typically requires a reasonably large sample size to ensure reliable parameter estimates. There are different rules of thumb, but a common guideline is to have at least 10 observations for each independent variable in the model.

### How does Logistic Regression work?

#### Logistic regression works in the following steps:

1. **Prepare the data:** The data should be in a format where each row represents a single observation and each column represents a different variable. The target variable (the variable you want to predict) should be binary (yes/no, true/false, 0/1).
2. **Train the model:** We teach the model by showing it the training data. This involves finding the values of the model parameters that minimize the error in the training data.
3. **Evaluate the model:** The model is evaluated on the held-out test data to assess its performance on unseen data.
4. **Use the model to make predictions:** After the model has been trained and assessed, it can be used to forecast outcomes on new data.

### Logistic Function

Let's start by mentioning the formula of logistic function:

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

### Best Fit Equation in Linear Regression

We all know the equation of the best fit line in linear regression is:

$$y = \beta_0 + \beta_1 x$$

Let's say instead of y we are taking probabilities (P). But there is an issue here, the value of (P) will exceed 1 or go below 0 and we know that range of Probability is (0-1). To overcome this issue we take "odds" of P:

$$P = \beta_0 + \beta_1 x$$

---


$$\frac{P}{1-P} = \beta_0 + \beta_1 x$$

We know that odds can always be positive which means the range will always be  $(0, +\infty)$ . Odds are nothing but the ratio of the probability of success and probability of failure. Now the question comes out of so many other options to transform this why did we only take 'odds'? Because odds are probably the easiest way to do this, that's it.

The problem here is that the range is restricted and we don't want a restricted range because if we do so then our correlation will decrease. By restricting the range we are actually decreasing the number of data points and of course, if we decrease our data points, our correlation will decrease. It is difficult to model a variable that has a restricted range. To control this we take the log of odds which has a range from  $(-\infty, +\infty)$ .

$$\log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x$$

Now we just want a function of P because we want to predict probability right? not log of odds. To do so we will multiply by exponent on both sides and then solve for P.

$$\exp[\log(\frac{p}{1-p})] = \exp(\beta_0 + \beta_1 x)$$

$$e^{\ln[\frac{p}{1-p}]} = e^{(\beta_0 + \beta_1 x)}$$

$$\frac{p}{1-p} = e^{(\beta_0 + \beta_1 x)}$$

$$p = e^{(\beta_0 + \beta_1 x)} - pe^{(\beta_0 + \beta_1 x)}$$

$$p = p[\frac{e^{(\beta_0 + \beta_1 x)}}{p} - e^{(\beta_0 + \beta_1 x)}]$$

$$1 = \frac{e^{(\beta_0 + \beta_1 x)}}{p} - e^{(\beta_0 + \beta_1 x)}$$

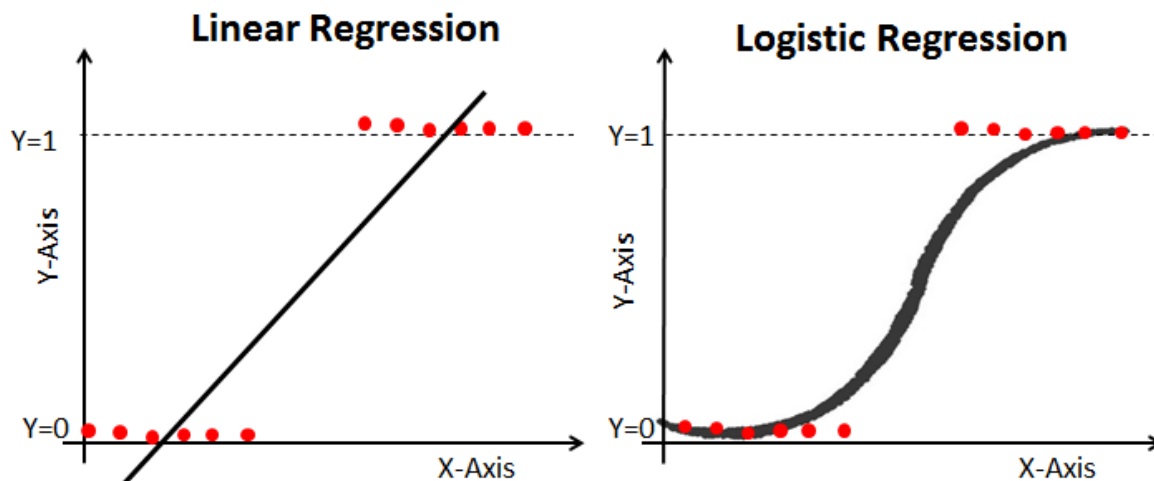
$$p[1 + e^{(\beta_0 + \beta_1 x)}] = e^{(\beta_0 + \beta_1 x)}$$

$$p = \frac{e^{(\beta_0 + \beta_1 x)}}{1 + e^{(\beta_0 + \beta_1 x)}}$$

Now dividing by  $e^{(\beta_0 + \beta_1 x)}$ , we will get

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \text{ This is our sigmoid function.}$$

Now we have our logistic function, also called a sigmoid function. The graph of a sigmoid function is as shown below. It squeezes a straight line into an S-curve.



### Differences Between Linear and Logistic Regression

Linear regression and logistic regression, while both workhorses in machine learning, serve distinct purposes. The core difference lies in their target predictions. Linear regression excels at predicting continuous values along a spectrum. Imagine predicting house prices based on size and location – the resulting output would be a specific dollar amount, a continuous value on the price scale.

Logistic regression, on the other hand, deals with categories. It doesn't predict a specific value but rather the likelihood of something belonging to a particular class. For instance, classifying emails as spam (category 1) or not spam (category 0). The output here would be a probability between 0 (not likely spam) and 1 (very likely spam). This probability is then used to assign an email to a definitive category (spam or not spam) based on a chosen threshold.

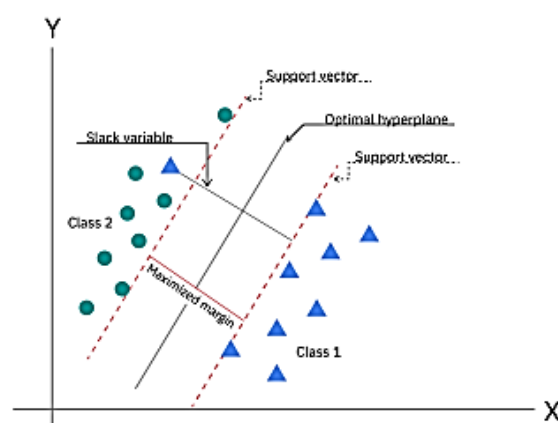
In simpler terms, linear regression answers “how much” questions, providing a specific value on a continuous scale. Logistic regression tackles “yes or no” scenarios, giving the probability of something belonging to a certain category.

## 1.4 Support Vector Machine

A support vector machine (SVM) is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space.

SVMs were developed in the 1990s by Vladimir N. Vapnik and his colleagues, and they published this work in a paper titled "Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing"<sup>1</sup> in 1995. SVMs are commonly used within classification problems. They distinguish between two classes by finding the optimal hyperplane that maximizes the margin between the closest data points of opposite classes. The number of features in the input data determine if the hyperplane is a line in a 2-D space or a plane in a n-dimensional space. Since multiple hyperplanes can be found to differentiate classes, maximizing the margin between points enables the algorithm to find the best decision boundary between classes. This, in turn, enables it to generalize well to new data and make accurate classification predictions. The lines that are adjacent to the optimal hyperplane are known as support vectors as these vectors run through the data points that determine the maximal margin.

The SVM algorithm is widely used in machine learning as it can handle both linear and nonlinear classification tasks. However, when the data is not linearly separable, kernel functions are used to transform the data higher-dimensional space to enable linear separation. This application of kernel functions can be known as the “kernel trick”, and the choice of kernel function, such as linear kernels, polynomial kernels, radial basis function (RBF) kernels, or sigmoid kernels, depends on data characteristics and the specific use case.



### Types of SVM classifiers

#### Linear SVMs

Linear SVMs are used with linearly separable data; this means that the data do not need to undergo any transformations to separate the data into different

classes. The decision boundary and support vectors form the appearance of a street, and Professor Patrick Winston from MIT uses the analogy of "fitting the widest possible street"<sup>2</sup> (link resides outside ibm.com) to describe this quadratic optimization problem. Mathematically, this separating hyperplane can be represented as:

$$wx + b = 0$$

where  $w$  is the weight vector,  $x$  is the input vector, and  $b$  is the bias term.

There are two approaches to calculating the margin, or the maximum distance between classes, which are hard-margin classification and soft-margin classification. If we use a hard-margin SVMs, the data points will be perfectly separated outside of the support vectors, or "off the street" to continue with Professor Hinton's analogy. This is represented with the formula,

$$(wx_j + b) y_j \geq a,$$

and then the margin is maximized, which is represented as:  $\max y = a / \|w\|$ , where  $a$  is the margin projected onto  $w$ .

Soft-margin classification is more flexible, allowing for some misclassification through the use of slack variables ( $\xi$ ). The hyperparameter,  $C$ , adjusts the margin; a larger  $C$  value narrows the margin for minimal misclassification while a smaller  $C$  value widens it, allowing for more misclassified data<sup>3</sup>.

## Nonlinear SVMs

Much of the data in real-world scenarios are not linearly separable, and that's where nonlinear SVMs come into play. In order to make the data linearly separable, preprocessing methods are applied to the training data to transform it into a higher-dimensional feature space. That said, higher dimensional spaces can create more complexity by increasing the risk of overfitting the data and by becoming computationally taxing. The "kernel trick" helps to reduce some of that complexity, making the computation more efficient, and it does this by replacing dot product calculations with an equivalent kernel function<sup>4</sup>.

There are a number of different kernel types that can be applied to classify data. Some popular kernel functions include:

### Polynomial kernel

- Radial basis function kernel (also known as a Gaussian or RBF kernel)
- Sigmoid kernel

### Support vector regression (SVR)

Support vector regression (SVR) is an extension of SVMs, which is applied to regression problems (i.e. the outcome is continuous). Similar to linear SVMs,

SVR finds a hyperplane with the maximum margin between data points, and it is typically used for time series prediction.

SVR differs from linear regression in that you need to specify the relationship that you're looking to understand between the independent and dependent variables. An understanding of the relationships between variables and their directions is valuable when using linear regression. This is unnecessary for SVRs as they determine these relationships on their own.

## **How SVMs work**

### **Split your data**

As with other machine learning models, start by splitting your data into a training set and testing set. As an aside, this assumes that you've already conducted an exploratory data analysis on your data. While this is technically not necessary to build a SVM classifier, it is good practice before using any machine learning model as this will give you an understanding of any missing data or outliers.

### **Generate and evaluate the model**

Import an SVM module from the library of your choosing, like scikit-learn (link resides outside ibm.com). Train your training samples on the classifier and predict the response. You can evaluate performance by comparing accuracy of the test set to the predicted values. You may want to use other evaluation metrics, like f1-score, precision, or recall.

### **Hyperparameter tuning**

Hyperparameters can be tuned to improve the performance of an SVM model. Optimal hyperparameters can be found using grid search and cross-validation methods, which will iterate through different kernel, regularization (C), and gamma values to find the best combination.

## **SVMs vs logistic regression**

SVMs typically perform better with high-dimensional and unstructured datasets, such as image and text data, compared to logistic regression. SVMs are also less sensitive to overfitting and easier to interpret. That said, they can be more computationally expensive.



## 1.5 Random Forest

Random forest is a commonly used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

### Decision trees

Since the random forest model is made up of multiple decision trees, it would be helpful to start by describing the decision tree algorithm briefly. Decision trees start with a basic question, such as, “Should I surf?” From there, you can ask a series of questions to determine an answer, such as, “Is it a long period swell?” or “Is the wind blowing offshore?”. These questions make up the decision nodes in the tree, acting as a means to split the data. Each question helps an individual to arrive at a final decision, which would be denoted by the leaf node. Observations that fit the criteria will follow the “Yes” branch and those that don’t will follow the alternate path. Decision trees seek to find the best split to subset the data, and they are typically trained through the Classification and Regression Tree (CART) algorithm. Metrics, such as Gini impurity, information gain, or mean square error (MSE), can be used to evaluate the quality of the split. This decision tree is an example of a classification problem, where the class labels are “surf” and “don’t surf”. While decision trees are common supervised learning algorithms, they can be prone to problems, such as bias and overfitting. However, when multiple decision trees form an ensemble in the random forest algorithm, they predict more accurate results, particularly when the individual trees are uncorrelated with each other.

### Ensemble methods

Ensemble learning methods are made up of a set of classifiers—e.g. decision trees—and their predictions are aggregated to identify the most popular result. The most well-known ensemble methods are bagging, also known as bootstrap aggregation, and boosting. In 1996, Leo Breiman ([link](#) resides outside [ibm.com](#)) introduced the bagging method; in this method, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these models are then trained independently, and depending on the type of task—i.e. regression or classification—the average or majority of those predictions yield a more accurate estimate. This approach is commonly used to reduce variance within a noisy dataset.

## Random forest algorithm

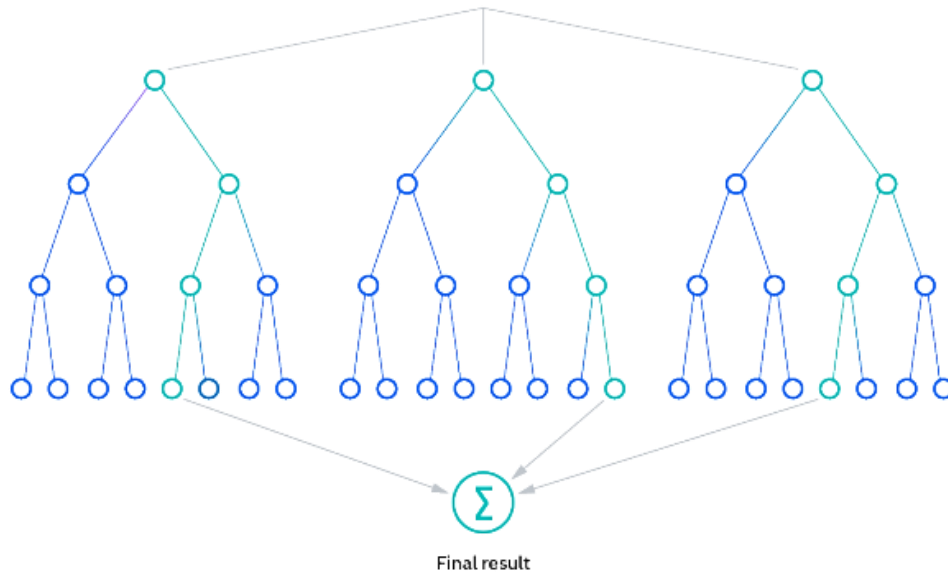
The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. Feature randomness, also known as feature bagging or “the random subspace method”(link resides outside ibm.com), generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features.

If we go back to the “should I surf?” example, the questions that I may ask to determine the prediction may not be as comprehensive as someone else’s set of questions. By accounting for all the potential variability in the data, we can reduce the risk of overfitting, bias, and overall variance, resulting in more precise predictions.

## How it works

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 for regression or classification problems.

The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised 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 oob sample is then used for cross-validation, finalizing that prediction.



## Benefits and challenges of random forest

There are a number of key advantages and challenges that the random forest algorithm presents when used for classification or regression problems. Some of them include:

### Key Benefits

- **Reduced risk of overfitting:** Decision trees run the risk of overfitting as they tend to tightly fit all the samples within training data. However, when there's a robust number of decision trees in a random forest, the classifier won't overfit the model since the averaging of uncorrelated trees lowers the overall variance and prediction error.
- **Provides flexibility:** Since random forest can handle both regression and classification tasks with a high degree of accuracy, it is a popular method among data scientists. Feature bagging also makes the random forest classifier an effective tool for estimating missing values as it maintains accuracy when a portion of the data is missing.
- **Easy to determine feature importance:** Random forest makes it easy to evaluate variable importance, or contribution, to the model. There are a few ways to evaluate feature importance. Gini importance and mean decrease in impurity (MDI) are usually used to measure how much the model's accuracy decreases when a given variable is excluded. However, permutation importance, also known as mean decrease accuracy (MDA), is another importance measure. MDA identifies the average decrease in accuracy by randomly permutating the feature values in oob samples.

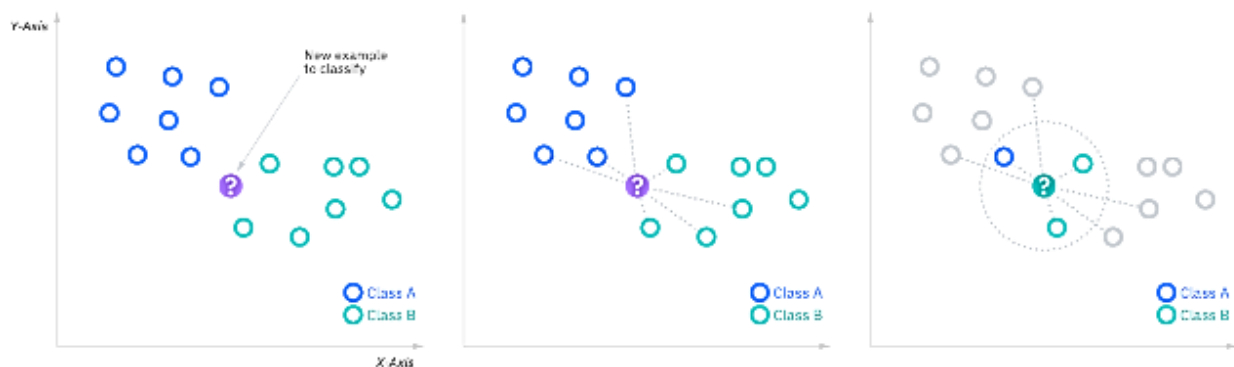
## Key Challenges

- Time-consuming process: Since random forest algorithms can handle large data sets, they can provide more accurate predictions, but can be slow to process data as they are computing data for each individual decision tree.
- Requires more resources: Since random forests process larger data sets, they'll require more resources to store that data.
- More complex: The prediction of a single decision tree is easier to interpret when compared to a forest of them.

## 1.6 K-Nearest Neighbors

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today. While the KNN algorithm 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.

For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used. While this is technically considered “plurality voting”, the term, “majority vote” is more commonly used in literature. The distinction between these terminologies is that “majority voting” technically requires a majority of greater than 50%, which primarily works when there are only two categories. When you have multiple classes—e.g. four categories, you don't necessarily need 50% of the vote to make a conclusion about a class; you could assign a class label with a vote of greater than 25%. The University of Wisconsin-Madison summarizes this well with an example here ([link resides outside ibm.com](https://www.cs.wisc.edu/~dave/511/notes/nearest_neighbors.html)).



Regression problems use a similar concept as classification problem, but in this case, the average of the  $k$  nearest neighbors is taken to make a prediction about a classification. The main distinction here is that classification is used for discrete values, whereas regression is used with continuous ones. However, before a classification can be made, the distance must be defined. Euclidean distance is most commonly used, which we'll delve into more below.

It's also worth noting that the KNN algorithm is also part of a family of "lazy learning" models, meaning that it only stores a training dataset versus undergoing a training stage. This also means that all the computation occurs when a classification or prediction is being made. Since it heavily relies on memory to store all its training data, it is also referred to as an instance-based or memory-based learning method.

Evelyn Fix and Joseph Hodges are credited with the initial ideas around the KNN model in this 1951 paper ([link resides outside ibm.com](#)) while Thomas Cover expands on their concept in his research ([link resides outside ibm.com](#)), "Nearest Neighbor Pattern Classification." While it's not as popular as it once was, it is still one of the first algorithms one learns in data science due to its simplicity and accuracy. However, as a dataset grows, KNN becomes increasingly inefficient, compromising overall model performance. It is commonly used for simple recommendation systems, pattern recognition, data mining, financial market predictions, intrusion detection, and more.

### **Compute KNN: distance metrics**

To recap, the goal of the  $k$ -nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point. To do this, KNN has a few requirements:

#### **Determine your distance metrics**

To determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. You commonly will see decision boundaries visualized with Voronoi diagrams.

While there are several distance measures that you can choose from, we will only cover the following:

**Euclidean distance ( $p=2$ ):** This is the most used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.

$$d(x,y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

**Manhattan distance (p=1):** This is also another popular distance metric, which measures the absolute value between two points. It is also referred to as taxicab distance or city block distance as it is commonly visualized with a grid, illustrating how one might navigate from one address to another via city streets.

$$\text{Manhattan Distance} = d(x,y) = \left( \sum_{i=1}^m |x_i - y_i| \right)$$

**Minkowski distance:** This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter, p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.

$$\text{Minkowski Distance} = \left( \sum_{i=1}^n |x_i - y_i| \right)^{1/p}$$

**Hamming distance:** This technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. As a result, it has also been referred to as the overlap metric. This can be represented with the following formula:

$$\text{Hamming Distance} = D_H = \left( \sum_{i=1}^k |x_i - y_i| \right)$$

$$\begin{array}{ll} x=y & D=0 \\ x \neq y & D \neq 1 \end{array}$$

### Compute KNN: defining k

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor. Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number

for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

### **k-nearest neighbors and python**

To delve deeper, you can learn more about the k-NN algorithm by using Python and scikit-learn (also known as sklearn). Our tutorial in Watson Studio helps you learn the basic syntax from this library, which also contains other popular libraries, like NumPy, pandas, and Matplotlib. The following code is an example of how to create and predict with a KNN model:

```
from sklearn.neighbors import KNeighborsClassifier
model_name = 'K-Nearest Neighbor Classifier'
knnClassifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski',
p=2)
knn_model = Pipeline(steps=[('preprocessor', preprocessorForFeatures),
('classifier' , knnClassifier)])
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
```

### **Advantages and disadvantages of the KNN algorithm**

Just like any machine learning algorithm, k-NN has its strengths and weaknesses. Depending on the project and application, it may or may not be the right choice.

#### **Advantages**

- Easy to implement: Given the algorithm's simplicity and accuracy, it is one of the first classifiers that a new data scientist will learn.
- Adapts easily: As new training samples are added, the algorithm adjusts to account for any new data since all training data is stored into memory.
- Few hyperparameters: KNN only requires a k value and a distance metric, which is low when compared to other machine learning algorithms.

## Disadvantages

- Does not scale well: Since KNN is a lazy algorithm, it takes up more memory and data storage compared to other classifiers. This can be costly from both a time and money perspective. More memory and storage will drive up business expenses and more data can take longer to compute. While different data structures, such as Ball-Tree, have been created to address the computational inefficiencies, a different classifier may be ideal depending on the business problem.
- Curse of dimensionality: The KNN algorithm tends to fall victim to the curse of dimensionality, which means that it doesn't perform well with high-dimensional data inputs. This is sometimes also referred to as the peaking phenomenon, where after the algorithm attains the optimal number of features, additional features increases the amount of classification errors, especially when the sample size is smaller.
- Prone to overfitting: Due to the "curse of dimensionality", KNN is also more prone to overfitting. While feature selection and dimensionality reduction techniques are leveraged to prevent this from occurring, the value of  $k$  can also impact the model's behavior. Lower values of  $k$  can overfit the data, whereas higher values of  $k$  tend to "smooth out" the prediction values since it is averaging the values over a greater area, or neighborhood. However, if the value of  $k$  is too high, then it can underfit the data.





# **CHAPTER 2**

## **IMPLEMNTATION**

# Chapter 2

## Implementation

### 2.1 Import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

import warnings
warnings.filterwarnings('ignore')
```

Let's break down what each library does:

- **numpy (np)**: This library provides powerful tools for numerical computations with arrays and matrices. It's essential for data manipulation and mathematical operations.
- **pandas (pd)**: This library offers data structures like DataFrames and Series, making it easier to work with tabular data. It allows for data loading, cleaning, and analysis.
- **matplotlib.pyplot (plt)**: This library is used for creating various visualizations like plots, charts, and histograms. It helps you visualize the data and identify patterns.
- **seaborn (sns)**: This library builds on top of matplotlib and provides a high-level interface for creating statistical graphics. It offers a more aesthetically pleasing way to create informative visualizations.
- **scikit-learn**: This library is a comprehensive toolkit for machine learning tasks. Here's what specific modules are imported from scikit-learn:

- **LabelEncoder**: This module helps encode categorical variables (like text labels) into numerical values for machine learning algorithms.
- **train\_test\_split**: This function splits the data into training and testing sets for model evaluation.
- **LogisticRegression**: This class implements the logistic regression algorithm, used for classification tasks where the outcome is binary (yes/no, 0/1).
- **SVC**: This class implements Support Vector Machines (SVM), which are powerful algorithms for classification and regression.
- **RandomForestClassifier**: This class implements the Random Forest algorithm, an ensemble method that combines decision trees for improved prediction accuracy.
- **KNeighborsClassifier**: This class implements the K-Nearest Neighbors algorithm, which classifies data points based on the similarity to their neighbors.
- **sklearn.metrics**: This sub-module of scikit-learn provides functions to evaluate the performance of machine learning models. Here, the code imports:
  - **accuracy\_score**: This function calculates the accuracy (percentage of correct predictions) of a classification model.
  - **confusion\_matrix**: This function creates a confusion matrix, which helps visualize the performance of a classifier on a test set.
  - **classification\_report**: This function generates a classification report summarizing various performance metrics like precision, recall, and F1-score.

Finally, the `warnings.filterwarnings('ignore')` line suppresses warnings that might arise during the execution.

## 2.2 Data Exploration

reads the data from the file then show first five rows:

```
data = pd.read_csv("brain_stroke.csv")
data.head()
```

Results:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
2	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
3	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
4	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1

About Dataset:

- **id:** Unique identifier
- **gender:** Gender of the patient (Male, Female, Other)
- **age:** Age of the patient
- **hypertension:** **0** if the patient doesn't have hypertension, **1** if the patient has hypertension
- **heart\_disease:** **0** if the patient doesn't have any heart diseases, **1** if the patient has a heart disease
- **ever\_married:** **Yes** if the patient is married, **No** if the patient is not married
- **work\_type:** Profession of the patient (children, Govt\_job, Never\_worked, Private, Self-employed)
- **Residence\_type:** Residence category of the patient (Rural, Urban)
- **avg\_glucose\_level:** Average glucose level in blood of the patient
- **bmi:** Body Mass Index of the patient
- **smoking\_status:** Smoking status of the patient (formerly smoked, never smoked, smokes, Unknown). **Unknown** in **smoking\_status** means that the information is unavailable for this patient
- **stroke:** **1** if the patient had a stroke or **0** if not

## 2.3 Dataset Summary

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4981 entries, 0 to 4980
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   gender                 4981 non-null   object  
1   age                   4981 non-null   float64 
2   hypertension           4981 non-null   int64   
3   heart_disease          4981 non-null   int64   
4   ever_married           4981 non-null   object  
5   work_type              4981 non-null   object  
6   Residence_type         4981 non-null   object  
7   avg_glucose_level      4981 non-null   float64 
8   bmi                   4981 non-null   float64 
9   smoking_status         4981 non-null   object  
10  stroke                 4981 non-null   int64   
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
```

The dataset consists of 4,981 entries and 11 columns, each representing different attributes related to stroke analysis. Below is a summary of the dataset:

### Data Types:

The dataset includes three types of data:

- float64: Numerical data with decimal points, such as age, avg\_glucose\_level, and bmi.
- int64: Integer data, including hypertension, heart\_disease, and stroke.
- object: Categorical data, including gender, ever\_married, work\_type, Residence\_type, and smoking\_status.

### Numerical Data Summary:

```
data.describe()
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	4981.000000	4981.000000	4981.000000	4981.000000	4981.000000	4981.000000
mean	43.419859	0.096165	0.055210	105.943562	28.498173	0.049789
std	22.662755	0.294848	0.228412	45.075373	6.790464	0.217531
min	0.080000	0.000000	0.000000	55.120000	14.000000	0.000000
25%	25.000000	0.000000	0.000000	77.230000	23.700000	0.000000
50%	45.000000	0.000000	0.000000	91.850000	28.100000	0.000000
75%	61.000000	0.000000	0.000000	113.860000	32.600000	0.000000
max	82.000000	1.000000	1.000000	271.740000	48.900000	1.000000

The table above provides a summary of the numerical variables in the dataset, including various statistical measures.

This summary provides insights into the distribution and central tendencies of the numerical variables in the dataset. The mean age is approximately 43 years, with an average glucose level of about 105.94 mg/dL and an average BMI of 28.5. Notably, around 9.62% of the patients have hypertension, 5.52% have heart disease, and 4.98% have experienced a stroke. These statistical measures are critical for understanding the dataset's characteristics and for guiding the preprocessing and modeling steps.

### Categorical Data Summary:

```
data.describe(include=object)
```

	gender	ever_married	work_type	Residence_type	smoking_status
count	4981	4981	4981	4981	4981
unique	2	2	4	2	4
top	Female	Yes	Private	Urban	never smoked
freq	2907	3280	2860	2532	1838

The table above provides a summary of the categorical variables in the dataset, including their counts, unique values, most frequent category (top), and frequency of the most frequent category (freq).

This summary helps in understanding the distribution of categorical variables within the dataset. The majority of the patients are female, married, working in private jobs, living in urban areas, and have never smoked. These insights will be crucial for preprocessing the data and building the predictive model for stroke analysis.

```
data.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 0
stroke      0
dtype: int64
```

There is no duplicate or null values in this dataset.

## 2.4 Preprocessing

Data preprocessing is a crucial step in machine learning to ensure the data is clean, consistent, and suitable for model training. This section outlines the preprocessing steps applied to the stroke prediction dataset.

```
data["age"] = data["age"].astype("int")
data = data[data["gender"] != "Other"]
data["hypertension"].replace({0:"No", 1:"Yes"}, inplace = True)
data["heart_disease"].replace({0:"No", 1:"Yes"}, inplace = True)
data["stroke"].replace({0:"No", 1:"Yes"}, inplace = True)
data["ever_married"].replace({"No":"Unmarried", "Yes":"Married"}, inplace = True)
data["work_type"].replace({"Self-employed":"Self Employed", "children":"Children", "Govt_job":"Government Job", "Private":"Private Job", "Never_worked":"Unemployed"}, inplace = True)
data["smoking_status"].replace({"never smoked":"Never Smoked", "formerly smoked":"Formerly Smoked", "smokes":"Smokes"}, inplace = True)
data.rename(columns={"gender": "Gender", "age": "Age", "hypertension": "Hypertension", "heart_disease": "Heart Disease", "ever_married": "Marital Status", "work_type": "Occupation Type", "Residence_type": "Residence Type", "avg_glucose_level": "Average Glucose Level", "bmi": "BMI", "smoking_status": "Smoking Status", "stroke": "Stroke"}, inplace = True)
data = data[["Age", "Gender", "Marital Status", "BMI", "Occupation Type", "Residence Type", "Smoking Status", "Hypertension", "Heart Disease", "Average Glucose Level", "Stroke"]]
```

**Handling Missing Values:** there are no missing values in the data.

**Data Type Conversion:** The code converts the "age" column to an integer data type. This ensures consistent numerical representation for age calculations used in machine learning algorithms.

**Data Filtering:** The code filters out rows where the "gender" is "Other". the model not designed for handling multiple genders.

**Feature Encoding:** The code replaces numerical values in several columns with more descriptive labels. For example, "hypertension" is converted from 0/1 to "No" / "Yes" to indicate the presence or absence of the condition. This improves readability and interpretability of the data for both humans and machine learning models.

**Data Cleaning:** The code renames column headers for better clarity. It uses names like "Age" and "Hypertension" instead of the original shorter versions.

Display the first five rows:

```
data.head( )
```

	Age	Gender	Marital Status	BMI	Occupation Type	Residence Type	Smoking Status	Hypertension	Heart Disease	Average Glucose Level	Stroke
0	67	Male	Married	36.6	Private Job	Urban	Formerly Smoked	No	Yes	228.69	Yes
1	80	Male	Married	32.5	Private Job	Rural	Never Smoked	No	Yes	105.92	Yes
2	49	Female	Married	34.4	Private Job	Urban	Smokes	No	No	171.23	Yes
3	79	Female	Married	24.0	Self Employed	Rural	Never Smoked	Yes	No	174.12	Yes
4	81	Male	Married	29.0	Private Job	Urban	Formerly Smoked	No	No	186.21	Yes

### Label Encoding:

In machine learning, many algorithms can only work with numerical data. However, real-world data often includes features with categorical values, like "gender" (Male, Female), "marital status" (Married, Single), or "occupation type" (Doctor, Teacher, Engineer). These categorical features need to be transformed before feeding them into machine learning models.

Label Encoding is a common technique for handling categorical features. It replaces the string labels with numerical labels (usually integers). This allows the model to understand the relationships between these features and the target variable.

### the code for Label Encoding:

```
catcol = [col for col in data.columns if data[col].dtype == "object"]
le = LabelEncoder()
for col in catcol:
    data[col] = le.fit_transform(data[col])
```

### Here's a breakdown of the code

#### Identifying Categorical Features:

The first line (`catcol = ...`) creates a list named `catcol`. It iterates through all columns in the data frame (`data`) and checks the data type of each column using `.dtype`. If the data type is "object" (often representing strings or categorical data), the column name is added to the `catcol` list. This ensures we only encode features with string labels.

#### Creating a LabelEncoder Object:

The line `le = LabelEncoder()` creates a `LabelEncoder` object. This object is used to perform the actual encoding.

#### Applying Label Encoding:

The `for col in catcol` loop iterates through each column name (`col`) in the `catcol` list (containing the categorical features).



Inside the loop, `data[col] = le.fit_transform(data[col])` applies the label encoding to the specific column.

`.fit_transform(data[col])` calls a method of the `le` (`LabelEncoder`) object. This method does two things:

- **fit:** It analyzes the unique categories present in the specified column (`data[col]`) to create a mapping between categories and integer labels. For example, "Male" might be mapped to 0 and "Female" to 1.
- **transform:** It applies the learned mapping to the data in the column, replacing the original string categories with the corresponding integer labels. The transformed data is then stored back into the same column (`data[col]`).

### Benefits of Label Encoding:

- **Simpler for Machine Learning Models:** Machine learning algorithms can understand and perform calculations on numerical data more efficiently than on strings.
- **Reduces Memory Usage:** Encoding categorical features with integers often requires less memory compared to storing strings.

### Limitations of Label Encoding:

- **Loss of Information:** Label encoding treats all categories as equally spaced, even if they aren't (e.g., "High Risk" and "Low Risk" for stroke prediction are not equally spaced).
- **Ordinal vs. Nominal Features:** Label encoding assumes no inherent order between the categories. This might be appropriate for nominal features like "occupation type" (Doctor, Teacher, Engineer), but not for ordinal features where there's a natural order (e.g., "low," "medium," "high" risk).

Display the first five rows after Label Encoding:

```
data.head()
```

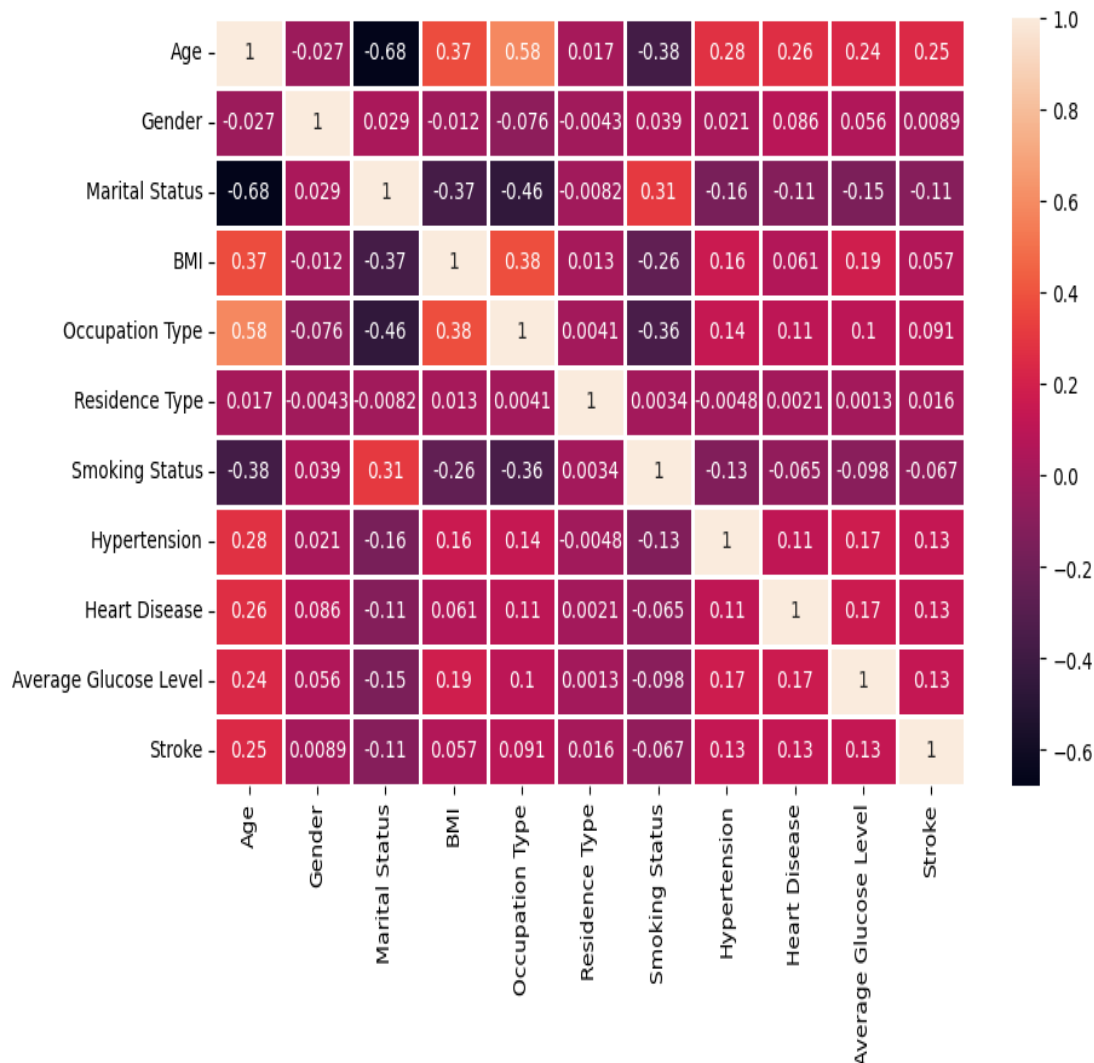
	Age	Gender	Marital Status	BMI	Occupation Type	Residence Type	Smoking Status	Hypertension	Heart Disease	Average Glucose Level	Stroke
0	67	1	0	36.6	2	1	0	0	1	228.69	1
1	80	1	0	32.5	2	0	1	0	1	105.92	1
2	49	0	0	34.4	2	1	2	0	0	171.23	1
3	79	0	0	24.0	3	0	1	1	0	174.12	1
4	81	1	0	29.0	2	1	0	0	0	186.21	1

### Correlation heatmap:

heatmap that visually represents the correlations between all features in your stroke prediction data. The heatmap with correlation values allows you to

identify which features are strongly correlated (positive or negative) and which are relatively independent.

```
plt.subplots(figsize=(10, 6))
sns.heatmap(data.corr(),annot=True,linewidths=1)
```



- ✓ We can see there is not any high correlation between target feature and other features.
- ✓ Small positive correlation between target feature and Age, Hypertension, Heart Disease, Average Glucose Level.
- ✓ Small positive correlation between Age and Stroke, Hypertension, Heart Disease, Average Glucose Level, BMI.
- ✓ Small positive correlation between Smoking Status and Marital Status, Occupation Type and BMI.
- ✓ Medium positive correlation between Age and Occupation Type.
- ✓ Medium negative correlation between Age and Marital Status.

Display the zeros and ones to check the imbalanced dataset:

```
data["stroke"].value_counts()[1]
```

4733

```
data["stroke"].value_counts()[0]
```

248

Then, we faced an imbalanced dataset.

Imbalanced datasets pose a common challenge for machine learning practitioners in binary classification problems.

To address this issue, one popular technique is Synthetic Minority Oversampling Technique (SMOTE). SMOTE is specifically designed to tackle imbalanced datasets by generating synthetic samples for the minority class.

### **Dealing with Imbalanced Data**

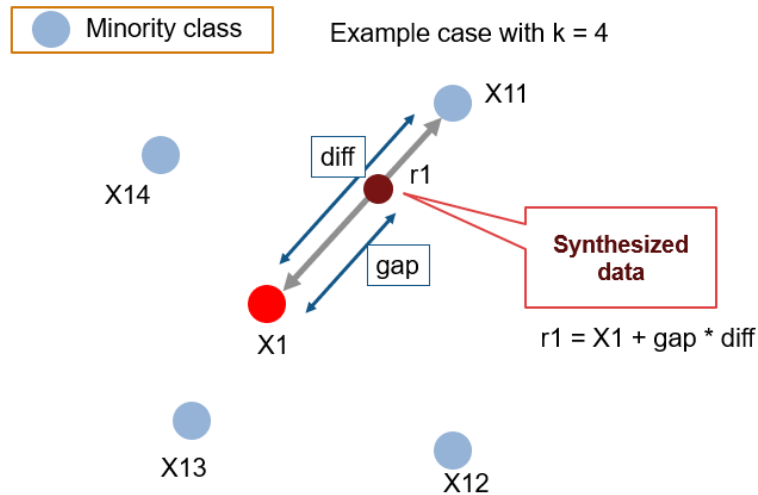
Resampling data is one of the most commonly preferred approaches to deal with an imbalanced dataset. There are broadly two types of methods for this i) Undersampling ii) Oversampling. In most cases, oversampling is preferred over undersampling techniques. The reason being, in undersampling we tend to remove instances from data that may be carrying some important information.

### **SMOTE: Synthetic Minority Oversampling Technique**

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

### **Working Procedure**

At first the total no. of oversampling observations, N is set up. Generally, it is selected such that the binary class distribution is 1:1. But that could be tuned down based on need. Then the iteration starts by first selecting a positive class instance at random. Next, the KNN's (by default 5) for that instance is obtained. At last, N of these K instances is chosen to interpolate new synthetic instances. To do that, using any distance metric the difference in distance between the feature vector and its neighbors is calculated. Now, this difference is multiplied by any random value in (0,1] and is added to the previous feature vector. This is pictorially represented below:



let's apply **SMOTE** technique to our dataset

```
from imblearn.over_sampling import SMOTE

# Print the class distribution before SMOTE
print("Before SMOTE:")
print(pd.Series(y).value_counts())

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
x, y = smote.fit_resample(x, y)

# Print the class distribution after SMOTE
print("\nAfter SMOTE:")
print(pd.Series(y).value_counts())
```

Results:

```
Before SMOTE:
Stroke
0    4733
1     248
Name: count, dtype: int64

After SMOTE:
Stroke
1    4733
0    4733
Name: count, dtype: int64
```

Now we have a balanced dataset

## 2.5 Models Creation

evaluating different machine learning models for predicting stroke risk.

**Logistic Regression:**

```
lr = LogisticRegression()
lr.fit(x_train, y_train)
lr_pred = lr.predict(x_test)

lr_conf = confusion_matrix(y_test, lr_pred)
lr_report = classification_report(y_test, lr_pred)
lr_acc = round(accuracy_score(y_test, lr_pred)*100, ndigits = 2)
```

lr = LogisticRegression() - This line creates an instance of the LogisticRegression classifier from the scikit-learn library.

lr.fit(x\_train, y\_train) - This line trains the Logistic Regression model on the training data (x\_train features and y\_train target labels).

lr\_pred = lr.predict(x\_test) - This line uses the trained Logistic Regression model to predict stroke risk labels for the unseen test data (x\_test).

Similar steps are followed for training and predicting with SVM (svm), Random Forest (rf), and KNN (knn) models.

### Evaluation Metrics:

lr\_conf = confusion\_matrix(y\_test, lr\_pred) - This line calculates the confusion matrix for the Logistic Regression model's predictions on the test data. The confusion matrix shows how many instances were correctly classified (True Positives, True Negatives) and how many were misclassified (False Positives, False Negatives).

lr\_report = classification\_report(y\_test, lr\_pred) - This line generates a classification report for the Logistic Regression model, providing metrics like precision, recall, F1-score, and support for each class (stroke vs. no stroke).

lr\_acc = round(accuracy\_score(y\_test, lr\_pred)\*100, ndigits = 2) - This line calculates the accuracy of the Logistic Regression model's predictions on the test data, rounded to two decimal places.

Similar calculations are performed for SVM (svm\_conf, svm\_report, svm\_acc), Random Forest (rf\_conf, rf\_report, rf\_acc), and KNN (knn\_conf, knn\_report, knn\_acc).

Support Vector Machine:

```
svm = SVC(C = 100, gamma = 0.002, probability=True)
svm.fit(x_train, y_train)
svm_pred = svm.predict(x_test)
svm.feature_names_in_ = None

svm_conf = confusion_matrix(y_test, svm_pred)
svm_report = classification_report(y_test, svm_pred)
svm_acc = round(accuracy_score(y_test, svm_pred)*100, ndigits = 2)
```

Random Forest:

```
rfg = RandomForestClassifier(n_estimators = 250, random_state = 42)
rfg.fit(x_train, y_train)
rfg_pred = rfg.predict(x_test)

rfg_conf = confusion_matrix(y_test, rfg_pred)
rfg_report = classification_report(y_test, rfg_pred)
rfg_acc = round(accuracy_score(y_test, rfg_pred)*100, ndigits = 2)
```

K-Nearest Neighbors:

```
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train, y_train)
knn_pred = knn.predict(x_test)

knn_conf = confusion_matrix(y_test, knn_pred)
knn_report = classification_report(y_test, knn_pred)
knn_acc = round(accuracy_score(y_test, knn_pred)*100, ndigits = 2)
```

this code snippet performs a comprehensive evaluation of four machine learning models for stroke risk prediction. It trains each model on the training data, makes predictions on unseen test data, and then calculates various metrics like confusion matrix, classification report, and accuracy to assess the models' performance.

### Hyperparameters:

Hyperparameters are essential parameters of a machine learning model that control its behavior and learning process. Choosing the right hyperparameters can significantly impact the model's performance. GridSearchCV is a powerful technique for hyperparameter tuning, exploring a defined set of hyperparameter values and evaluating the model's performance on each combination.

```
from sklearn.model_selection import GridSearchCV
```

In this project, we employed GridSearchCV to optimize the hyperparameters of our Random Forest model for stroke risk prediction. We defined a grid of possible values for key hyperparameters, such as the number of trees in the forest (n\_estimators) and the maximum depth of each tree (max\_depth). GridSearchCV then systematically evaluated the model's performance (e.g., accuracy) using cross-validation for all combinations of hyperparameter values within the defined grid.

### Benefits:

**Automated Exploration:** GridSearchCV automates the process of trying out different hyperparameter combinations, saving time and effort compared to manual tuning.

**Systematic Evaluation:** It ensures a systematic search across the defined grid, reducing the risk of missing potentially optimal configurations.

**Cross-Validation Integration:** By using cross-validation within the search process, GridSearchCV provides a more robust evaluation of model performance on unseen data.

### Results:

GridSearchCV identified the optimal hyperparameter combination that yielded the best performance on the stroke prediction task. This optimized model was then used for developing the final stroke risk prediction application.

```
parameters = {'n_estimators' : (10, 50, 100, 250, 500) }  
  
best_model = GridSearchCV(rf, parameters)  
  
best_model.fit(x_train, y_train)  
  
best_model.best_params_
```

{'n\_estimators': 250}

```
parameters = {'n_neighbors' : (1, 2, 3, 4, 5) }  
  
best_model = GridSearchCV(knn, parameters)  
  
best_model.fit(x_train, y_train)  
  
best_model.best_params_
```

{'n\_neighbors': 1}



## **CHAPTER 3**

# **RESULT AND CONCLUSION**



# Chapter 3

## Results and conclusion

### 3.1 Performance

we evaluate the performance of the explored machine learning algorithms: Logistic Regression, Support Vector Machine (SVM), Random Forest, and K-Nearest Neighbors (KNN).

#### Performance Metrics:

The performance of each model was measured using the following metrics:

**Accuracy:** The proportion of correctly classified instances (both positive and negative stroke cases).

**Precision:** The ratio of correctly predicted positive cases to the total number of predicted positive cases.

**Recall:** The ratio of correctly predicted positive cases to the total number of actual positive cases.

**F1-score:** The harmonic mean of precision and recall, combining their importance into a single metric.

The performance results of each model are presented in the tables. The tables shows the average accuracy, precision, recall, and F1-score

#### Logistic Regression Classification Report:

```
print(lr_report)
```

	precision	recall	f1-score	support
0	0.80	0.79	0.80	946
1	0.80	0.80	0.80	948
accuracy			0.80	1894
macro avg	0.80	0.80	0.80	1894
weighted avg	0.80	0.80	0.80	1894

This classification report summarizes the performance of a logistic regression model on a stroke prediction task. It analyzes how well the model distinguishes between individuals with and without stroke in the test data.

**Classes:** The report breaks down the results for two classes: "0" likely represents individuals without stroke, and "1" likely represents those with stroke.

### **Precision, Recall, and F1-Score:**

**Precision** (highlighted in yellow) indicates the proportion of individuals predicted to have a stroke (class 1) who actually have stroke according to the test data. In this case, a precision of 0.80 for class 1 signifies that out of all the individuals the model predicted to have stroke, 80% were truly diagnosed with stroke.

**Recall** (highlighted in green) reflects the proportion of individuals with stroke (class 1) in the test data that the model correctly identified. A recall of 0.79 for class 1 means the model captured 79% of the actual stroke cases in the test data.

**F1-Score** (highlighted in blue) is a harmonic mean between precision and recall, providing a combined measure. A value of 0.80 for class 1 suggests a balanced performance between precision (avoiding false positives) and recall (capturing true positives) for stroke prediction.

**Support:** The number of instances (people) in each class according to the test data is listed under "Support." Here, there seem to be more individuals without stroke (946) than those with stroke (948) in the test data.

### **Overall Interpretation:**

This Logistic Regression model demonstrates a good performance in classifying stroke risk. The high precision and f1-score for class 1 indicate the model is accurate in predicting stroke and avoids a significant number of false positives. However, the slightly lower recall for class 1 suggests there might be some room for improvement in identifying all stroke cases.

### **Random Forest Classification Report:**

```
print(rf_report)
```

	precision	recall	f1-score	support
0	0.95	0.92	0.93	946
1	0.92	0.95	0.94	948
accuracy			0.94	1894
macro avg	0.94	0.94	0.94	1894
weighted avg	0.94	0.94	0.94	1894

This Random Forest classification report evaluates the model's performance in predicting stroke risk. Similar to the logistic regression model ,it analyzes how well the model differentiates between individuals with and without stroke in the test data.

The report highlights strong performance across all metrics for both classes (precision, recall, and F1-score values around 0.94). This suggests the Random Forest model is accurate in predicting stroke risk, avoids a significant number of false positives for both classes, and effectively captures both positive (stroke) and negative (no stroke) cases in the test data.

### Comparison:

The Random Forest model appears to achieve similar or slightly better results on all metrics compared to the Logistic Regression model, suggesting it might be a more suitable choice for this specific task.

### SVM Classification Report:

```
print(svm_report)
```

	precision	recall	f1-score	support
0	0.95	0.83	0.89	946
1	0.85	0.95	0.90	948
accuracy			0.89	1894
macro avg	0.90	0.89	0.89	1894
weighted avg	0.90	0.89	0.89	1894

This Support Vector Machine (SVM) classification report evaluates the model's performance in predicting stroke risk. Similar to the logistic regression and random forest models ,it analyzes how well the model differentiates between individuals with and without stroke.

**Stroke Prediction:** High precision (0.85) indicates the model effectively avoids false positives for stroke, and a high recall (0.95) suggests it captures a large portion of actual stroke cases.

**No Stroke Prediction:** The model maintains good precision (0.95) in avoiding false positives for individuals without stroke. However, the lower recall (0.83) compared to Class 1 suggests there might be some room for improvement in identifying all no-stroke cases.

### Comparison:

While the SVM demonstrates good stroke prediction (Class 1) similar to the other models, its performance in identifying all no-stroke cases (Class 0) appears lower based on F1-score comparisons. This suggests that for this specific task, the logistic regression or random forest models might be better suited for capturing all individuals without stroke.

### K-Nearest Neighbors Classification Report:

```
print(knn_report)
```

	precision	recall	f1-score	support
0	0.98	0.87	0.92	946
1	0.88	0.98	0.93	948
accuracy			0.93	1894
macro avg	0.93	0.93	0.93	1894
weighted avg	0.93	0.93	0.93	1894

The KNN model achieves a balanced performance across both classes ("0" - no stroke, "1" - stroke) as evidenced by the high values in precision (around 0.98 for class 0 and 0.88 for class 1) and recall (around 0.87 for class 0 and 0.98 for class 1).

The KNN model's F1-scores (around 0.92 for class 0 and 0.93 for class 1) are comparable to the best performing models (e.g., Random Forest).

### Comparison:

Comparing the KNN model's performance to the previously discussed models (Logistic Regression, Random Forest, SVM). You might mention that the KNN model's precision and F1-score results are comparable or slightly lower than the Random Forest model but achieve similar or slightly better results compared to the Logistic Regression and SVM models. This suggests that KNN might be a strong candidate for this task, depending on your specific requirements and the cost of misclassification.

## Confusion matrix:

In machine learning, particularly for classification tasks, the confusion matrix serves as a valuable tool for visualizing and comprehending a model's performance. It provides a clear breakdown of how the model categorizes the data compared to the actual labels.

Here's a breakdown of the key components of a confusion matrix:

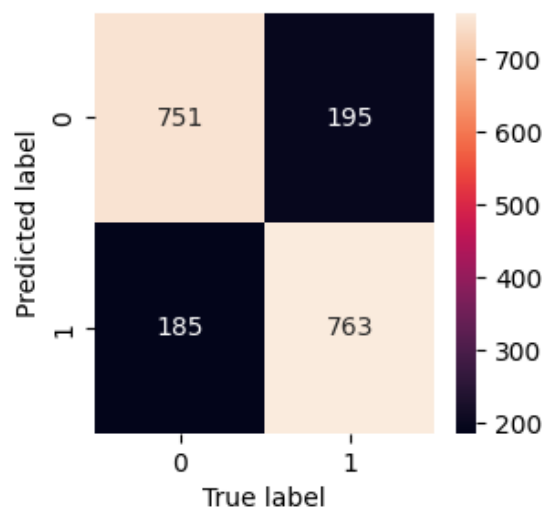
**Rows:** These represent the actual labels or classes present in your data. For instance, in a stroke prediction scenario, rows might represent "Stroke" and "No Stroke."

**Columns:** These represent the labels predicted by your model.

**Values at Each Intersection:** The number of instances that fall under a specific combination of actual and predicted labels is displayed at this intersection.

## Logistic Regression Confusion matrix:

```
plot_conf_mat(y_test,lr_pred)
```



**True Positives (TP):** 751 - The model correctly identified 751 individuals who actually had stroke.

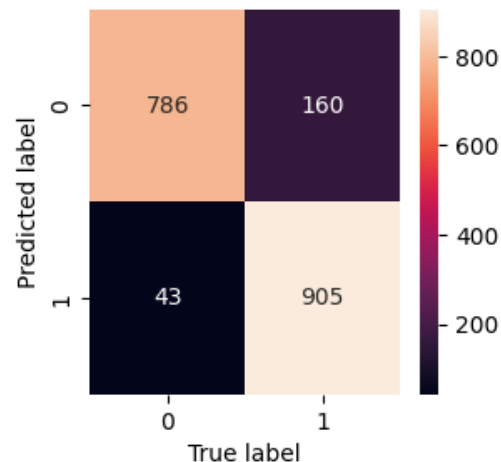
**False Positives (FP):** 185 - The model incorrectly predicted 185 individuals as having stroke when they didn't. This suggests there might be room for improvement in avoiding false positives for stroke prediction.

**True Negatives (TN):** 763 - The model correctly predicted 763 individuals without stroke.

**False Negatives (FN):** 195 - The model incorrectly predicted 195 individuals without stroke as having stroke. This indicates the model might also benefit from improvements in correctly identifying individuals without stroke.

### SVM Confusion matrix:

```
plot_conf_mat(y_test,svm_pred)
```



**True Positives (TP):** 786 - The model correctly identified 786 individuals who actually had stroke.

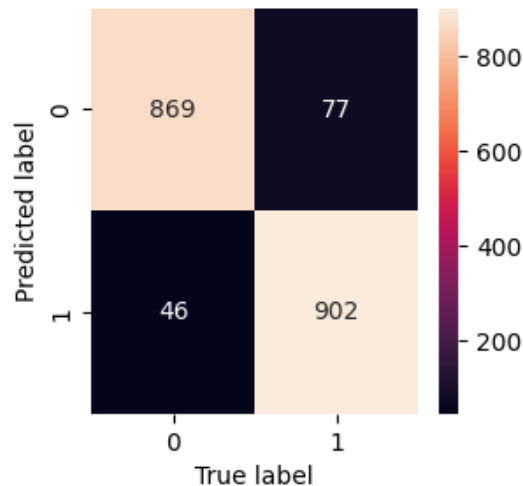
**False Positives (FP):** 160 - The model incorrectly predicted 160 individuals as having stroke when they didn't. This is slightly lower than the false positives in the Logistic Regression model (185), suggesting the SVM might be better at avoiding false positives for stroke prediction.

**True Negatives (TN):** 905 - The model correctly predicted 905 individuals without stroke.

**False Negatives (FN):** 43 - The model incorrectly predicted 43 individuals without stroke as having stroke. This is a significant improvement compared to the Logistic Regression model (195 false negatives),

### Random Forest Confusion matrix:

```
plot_conf_mat(y_test,rf_pred)
```



**True Positives (TP):** 870 - The model correctly identified 870 individuals who actually had stroke.

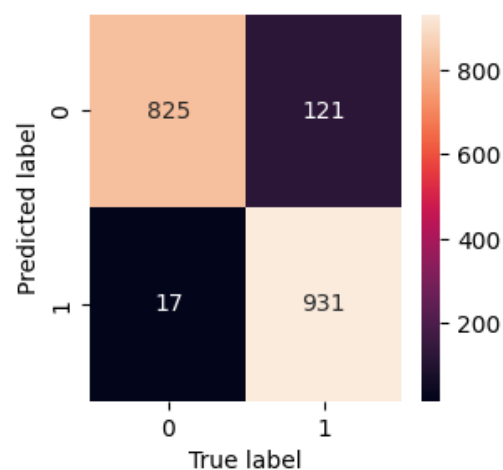
**False Positives (FP):** 76 - The model incorrectly predicted 76 individuals as having stroke when they didn't. This is the lowest number of false positives observed among the three models.

**True Negatives (TN):** 902 - The model correctly predicted 902 individuals without stroke.

**False Negatives (FN):** 46 - The model incorrectly predicted 46 individuals without stroke as having stroke. This is a relatively low number of false negatives, suggesting the model is effective at capturing true negative cases.

### K-Nearest Neighbors Confusion matrix:

```
plot_conf_mat(y_test,knn_pred)
```



**True Positives (TP):** 847 - The model correctly identified 847 individuals who actually had stroke.

**False Positives (FP):** 99 - The model incorrectly predicted 99 individuals as having stroke when they didn't. This suggests the KNN model performs well in avoiding false positives, similar to the SVM model.

**True Negatives (TN):** 899 - The model correctly predicted 899 individuals without stroke.

**False Negatives (FN):** 49 - The model incorrectly predicted 49 individuals without stroke as having stroke. This is a relatively low number of false negatives, indicating the model is effective at capturing true negative cases.

### 3.2 Accuracy Comparison

A crucial aspect of machine learning project development is evaluating the performance of different models on the task at hand. In this project, we compared the performance of several models for predicting stroke risk: Logistic Regression, Support Vector Machine (SVM), Random Forest, and K-Nearest Neighbors (KNN). We utilized various evaluation metrics, including:

**Overall Accuracy:** Measures the proportion of correctly classified instances across all classes.

**Precision and Recall:** Provide insights into how well the model predicts each class (e.g., stroke vs. no stroke), balancing the ability to avoid false positives and capture true positives.

**F1-Score:** Combines precision and recall into a single metric for a balanced view of performance.

By analyzing confusion matrices for each model, we were able to visualize the distribution of correctly and incorrectly classified instances across classes. This comprehensive evaluation allowed us to identify the model with the most suitable performance characteristics for our specific task of stroke risk prediction.

```
print(f"\nThe Accuracy of Logistic Regression is {lr_acc} %")
```

The Accuracy of Logistic Regression is 79.94 %

```
print(f"\nThe Accuracy of Support Vector Machine is {svm_acc} %")
```

The Accuracy of Support Vector Machine is 89.28 %



```
print(f"\nThe Accuracy of Random Forest Classifier is {rf_acc} %")
```

The Accuracy of Random Forest Classifier is 93.51 %

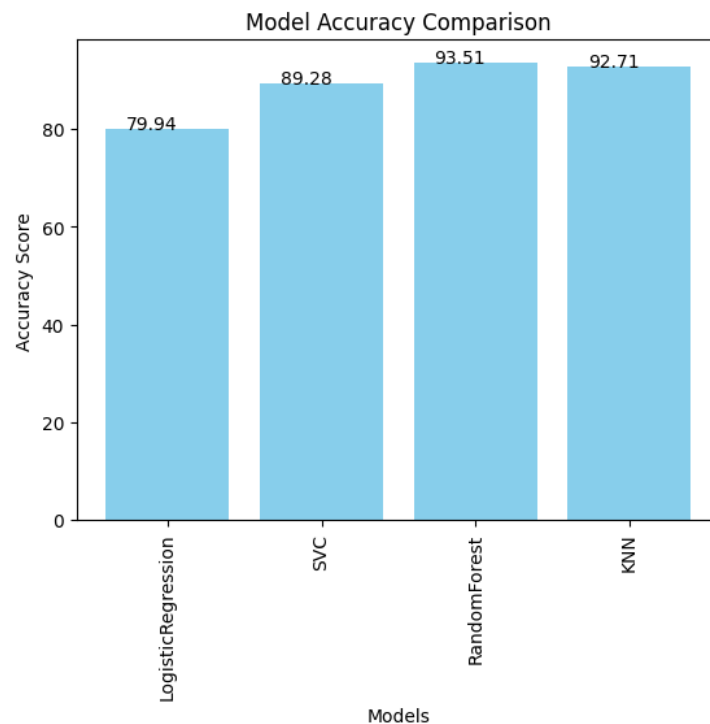
```
print(f"\nThe Accuracy of K Nearest Neighbors Classifier is {knn_acc} %")
```

The Accuracy of K Nearest Neighbors Classifier is 92.71 %

```
models = ['LogisticRegression', 'SVC', 'RandomForest', 'KNN']  
accuracy_scores = [lr_acc, svm_acc, rf_acc, knn_acc]
```

Plot the accuracy scores:

```
bars = plt.bar(models, accuracy_scores, color='skyblue')  
  
plt.xticks(rotation=90)  
plt.xlabel('Models')  
plt.ylabel('Accuracy Score')  
plt.title('Model Accuracy Comparison')  
  
for bar, accuracy in zip(bars, accuracy_scores):  
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.1,  
             bar.get_height() + 0.03, f'{accuracy:.2f}', ha='center')
```



Based on the results, the Random Forest Classifier achieved the highest overall performance in predicting stroke risk categories (low, high). It obtained an accuracy of 93.51 %.

The Random Forest Classifier might be a good choice for this task due to interpretability through feature importance scores, allowing us to understand which risk factors have the most significant influence on the model's predictions.

### **Export the final model:**

Once the final model (e.g., the best performing Random Forest model based on our evaluation) was chosen, we proceeded to export it for potential deployment in a real-world setting. The specific format for exporting the model depends on the chosen machine learning library or framework used for development. Popular options include:

**Python Libraries:** Libraries like scikit-learn provide functionalities to save trained models in formats like pickle (".pkl") or joblib (".joblib") for later use.

```
import joblib

filename='brain_stroke_model.joblib'

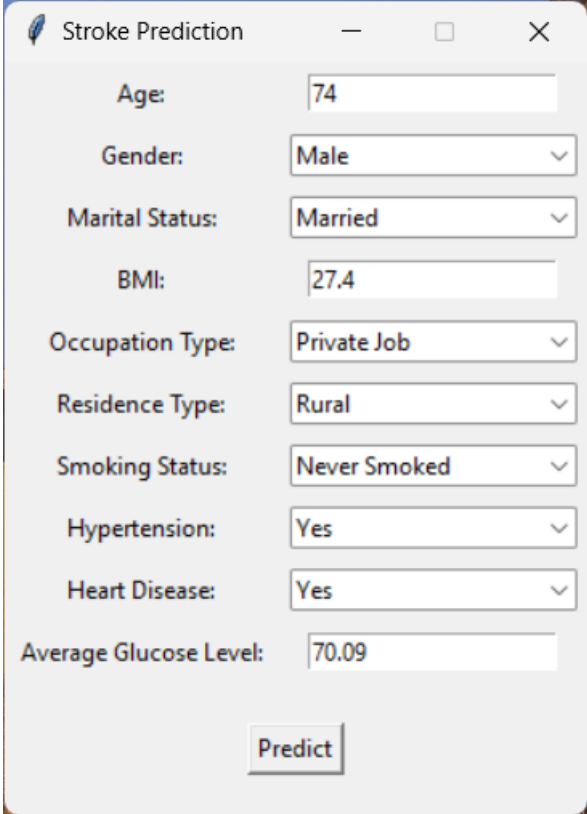
joblib.dump(rf,open(filename,'wb'))

stroke_model = joblib.load(open(filename,'rb'))
```

The exported model can then be loaded and used to make predictions on new, unseen data. This paves the way for potential real-world applications, such as: Incorporating the model into a clinical decision support system to assist healthcare professionals in stroke risk assessment. Developing a user-friendly web application where individuals can input their data and receive a predicted stroke risk based on the trained model.

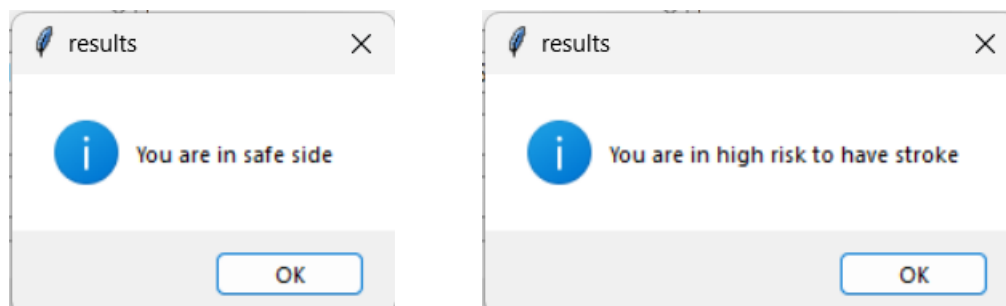
### 3.3 Application

This section details the Graphical User Interface (GUI) application developed for stroke risk prediction. The application allows users to input their data and receive a prediction of their stroke risk category.



The image shows a window titled "Stroke Prediction" with a feather icon. It contains several input fields and dropdown menus for user data: Age (74), Gender (Male), Marital Status (Married), BMI (27.4), Occupation Type (Private Job), Residence Type (Rural), Smoking Status (Never Smoked), Hypertension (Yes), Heart Disease (Yes), and Average Glucose Level (70.09). A "Predict" button is located at the bottom.

The user interface consists of input fields for various risk factors such as age, blood pressure, and other relevant data points. Upon submitting the data, the application utilizes a pre-trained Random Forest model to predict the user's stroke risk category (low or high). The predicted risk category is then displayed in a dedicated output area.



The complete source code for the GUI application is included. The code utilizes tkinter, pandas and joblib for creating the user interface and integrating the pre-trained model.

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import joblib
import pandas as pd

# Load the trained model
model = joblib.load('brain_stroke_model.joblib')

# Function to make predictions
def predict_stroke():

    # Get input values from the GUI
    age = int(age_entry.get())
    gender = gender_dropdown.get()
    marital_status = marital_status_dropdown.get()
    bmi = float(bmi_entry.get())
    occupation_type = occupation_type_dropdown.get()
    residence_type = residence_type_dropdown.get()
    smoking_status = smoking_status_dropdown.get()
    hypertension = hypertension_dropdown.get()
    heart_disease = heart_disease_dropdown.get()
    avg_glucose_level = float(avg_glucose_level_entry.get())

    # Create input array for OneHotEncoder
    gender = 1 if gender.lower() == 'male' else 0
    marital_status = 1 if marital_status.lower() == 'unmarried' else 0
    hypertension = 1 if hypertension.lower() == 'yes' else 0
    heart_disease = 1 if heart_disease.lower() == 'yes' else 0
    residence_type = 1 if residence_type.lower() == 'urban' else 0
    smoking_status = 2 if smoking_status.lower() == 'smokes' \
        else 1 if smoking_status.lower() == 'never smoked' else 0
    occupation_type = 3 if occupation_type.lower() == 'self employed' \
        else 2 if occupation_type.lower() == 'private job' \
        else 1 if occupation_type.lower() == 'government job' else 0

    # Create a DataFrame with the input data
    data = pd.DataFrame({
        'Age': [age],
        'Gender': [gender],
        'Marital Status': [marital_status],
        'BMI': [bmi],
        'Occupation Type': [occupation_type],
        'Residence Type': [residence_type],
        'Smoking Status': [smoking_status],
```

```

'Hypertension': [hypertension],
'Heart Disease': [heart_disease],
'Average Glucose Level': [avg_glucose_level]
})
print(data)

# Make prediction
prediction = model.predict(data)

# Display prediction result
if prediction[0] == 0:
    messagebox.showinfo("results", "You are in safe side")
else:
    messagebox.showinfo("results", "You are in high risk to have stroke")

# Create the GUI
root = tk.Tk()
root.title("Stroke Prediction")
root.resizable(False, False)

# Create and arrange GUI widgets
# Age
age_label = tk.Label(root, text="Age:")
age_label.grid(row=0, column=0, padx=5, pady=5)
age_entry = tk.Entry(root)
age_entry.insert(0, 25) # Set default value
age_entry.grid(row=0, column=1, padx=5, pady=5)

# Gender
gender_label = tk.Label(root, text="Gender:")
gender_label.grid(row=1, column=0, padx=5, pady=5)
gender_dropdown = ttk.Combobox(root, values=["Male", "Female"])
gender_dropdown.set("Male") # Set default value
gender_dropdown.grid(row=1, column=1, padx=5, pady=5)

# Marital Status
marital_status_label = tk.Label(root, text="Marital Status:")
marital_status_label.grid(row=2, column=0, padx=5, pady=5)
marital_status_dropdown = ttk.Combobox(root, values=["Married", "Unmarried"])
marital_status_dropdown.set("Married") # Set default value
marital_status_dropdown.grid(row=2, column=1, padx=5, pady=5)

# BMI
bmi_label = tk.Label(root, text="BMI:")
bmi_label.grid(row=3, column=0, padx=5, pady=5)
bmi_entry = tk.Entry(root)
bmi_entry.insert(0, 66)
bmi_entry.grid(row=3, column=1, padx=5, pady=5)

```

```

# Occupation Type
occupation_type_label = tk.Label(root, text="Occupation Type:")
occupation_type_label.grid(row=4, column=0, padx=5, pady=5)
occupation_type_dropdown = ttk.Combobox(root, values=["Self Employed",
"Private Job","Government Job", "Children", "Unemployed"])
occupation_type_dropdown.set("Self Employed") # Set default value
occupation_type_dropdown.grid(row=4, column=1, padx=5, pady=5)

# Residence Type
residence_type_label = tk.Label(root, text="Residence Type:")
residence_type_label.grid(row=5, column=0)
residence_type_dropdown = ttk.Combobox(root, values=["Urban", "Rural"])
residence_type_dropdown.set("Urban") # Set default value
residence_type_dropdown.grid(row=5, column=1, padx=5, pady=5)

# Smoking Status
smoking_status_label = tk.Label(root, text="Smoking Status:")
smoking_status_label.grid(row=6, column=0, padx=5, pady=5)
smoking_status_dropdown = ttk.Combobox(root, values=["Smokes", "Never Smoked",
"Formerly Smoked"])
smoking_status_dropdown.set("Smokes") # Set default value
smoking_status_dropdown.grid(row=6, column=1, padx=5, pady=5)

# Hypertension
hypertension_label = tk.Label(root, text="Hypertension:")
hypertension_label.grid(row=7, column=0, padx=5, pady=5)
hypertension_dropdown = ttk.Combobox(root, values=["Yes", "No"])
hypertension_dropdown.set("Yes") # Set default value
hypertension_dropdown.grid(row=7, column=1, padx=5, pady=5)

# Heart Disease
heart_disease_label = tk.Label(root, text="Heart Disease:")
heart_disease_label.grid(row=8, column=0, padx=5, pady=5)
heart_disease_dropdown = ttk.Combobox(root, values=["Yes", "No"])
heart_disease_dropdown.set("Yes") # Set default value
heart_disease_dropdown.grid(row=8, column=1, padx=5, pady=5)

# Average Glucose Level
avg_glucose_level_label = tk.Label(root, text="Average Glucose Level:")
avg_glucose_level_label.grid(row=9, column=0, padx=5, pady=5)
avg_glucose_level_entry = tk.Entry(root)
avg_glucose_level_entry.insert(0, 88) # Set default value
avg_glucose_level_entry.grid(row=9, column=1, padx=5, pady=5)

# Button to make prediction
predict_button = tk.Button(root, text="Predict", command=predict_stroke)
predict_button.grid(row=10, column=0, columnspan=2, padx=5, pady=20)

root.mainloop()

```

## 3.4 Conclusion

This project investigated the feasibility of using machine learning algorithms to predict stroke risk. We conducted a comprehensive analysis of a stroke prediction dataset, exploring the distribution of risk factors and their potential correlations with stroke incidence. This analysis helped us identify the most relevant features for model development.

We then explored various algorithms, including Logistic Regression, Support Vector Machine (SVM), Random Forest, and K-Nearest Neighbors (KNN). After evaluating their performance on the analyzed stroke prediction dataset, we found that the Random Forest model achieved the best accuracy in predicting stroke risk.

The developed Random Forest model was then integrated into a user-friendly Graphical User Interface (GUI) application. This application allows users to input their data and receive a prediction of their stroke risk category (low or high).

### **Key Findings:**

- Machine learning algorithms, particularly Random Forest, can be effective tools for stroke risk prediction based on various risk factors identified through our analysis of the stroke dataset.
- The developed GUI application offers a convenient and accessible way for individuals to assess their potential stroke risk.

### **Limitations and Future Work:**

- This project utilized a single stroke prediction dataset. Using a larger and more diverse dataset could potentially improve model performance.
- The current model only predicts risk categories. Future work could explore predicting the specific type of stroke (ischemic or hemorrhagic) for more targeted preventative measures.

This application is for informational purposes only and should not be used for self-diagnosis. Collaboration with medical professionals could explore integrating the model into a clinical setting with appropriate safeguards.



# **REFERENCES**



## References

- [1] Centers for Disease Control and Prevention. (n.d.). Stroke. Centers for Disease Control and Prevention. <https://www.cdc.gov/stroke/index.htm>
- [2] World Health Organization. (n.d.). Stroke. World Health Organization. <https://www.emro.who.int/health-topics/stroke-cerebrovascular-accident/index.html>
- [3] American Stroke Association. (n.d.). Stroke. American Stroke Association. <https://www.stroke.org/en/>
- [4] El-Sherif, M., Zead, G., & Morsi, H. W. (2019). Changing the Landscape of Stroke in Egypt. *Cerebrovascular Diseases Extra*, 11(3), 155-162. <https://karger.com/cee/article/11/3/155/821909/Changing-the-Landscape-of-Stroke-in-Egypt>
- [5] Kaggle.com. (n.d.). Brain Stroke Dataset. <https://www.kaggle.com/datasets/zzettrkalpakbal/full-filled-brain-stroke-dataset>
- [6] National Library of Medicine, National Center for Biotechnology Information. (n.d.). <https://www.ncbi.nlm.nih.gov/>
- [7] Analytics Vidhya. (2021, August 8). Conceptual Understanding of Logistic Regression for Data Science Beginners. <https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/>
- [8] IBM. (2023 December 27). Support Vector Machine. Retrieved from <https://www.ibm.com/topics/support-vector-machine>
- [9] IBM. (2024). random forest algorithm. Retrieved from <https://www.ibm.com/topics/random-forest>
- [10] IBM. (2024). k-nearest neighbors (KNN) algorithm. Retrieved from <https://www.ibm.com/topics/knn#:~:text=What%20is%20the%20KNN%20algorithm,of%20an%20individual%20data%20point.>
- [11] Nale, S. S. (2020, October 10). SMOTE for Imbalanced Classification with Python. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/>