BIRMINGHAM CITY University

CMP7161
Advanced Data Science Project
2022–2023

Individual Report

Predicting Event of Stroke with K-Nearest Neighbors Algorithm

Student: B
Student Number: -

Table of Contents

1 Report Introduction

Reported by the World Stroke Organisation, stroke is the second leading cause of death, with numerous global burdens occurring in Lower/Lower-Middle Income Countries (LMIC), (Feigin *et al.*, 2022). As concluded by Wiens and Shenoy (2017), combining machine learning with electronic data will allow early identification of ‘underlying risks’ to ‘develop targeted interventions,’ leading to a ‘major shift in healthcare.’

Therefore, this report explores how machine learning can be applied to electronic data and its ability to predict the event of a stroke occurring. This report incorporates the stages of dataset identification, Exploratory Data Analysis (EDA), model identification, data pre-processing, model training, model testing and evaluation. These stages have been developed in Google's Colab notebook using Python, Pandas and Scikit Learn.

1.1 Dataset identification

Before building a predictive machine learning model, a dataset must be identified. To do this, the research team considered various fields; a collective interest was the medical field. This was due to a curiosity in how medical datasets can be leveraged to develop meaningful, and potentially lifechanging, models.

The research team chose the 'Stroke Prediction Dataset' which represents the event of a stroke occurring accounting 10 clinical features (Fedesoriano, 2021). Table 1 outlines the attributes found in the dataset.

Table 1 Table of the attributes identified in the stroke prediction dataset with a description for each.

Attribute	Description
id	A unique id to identify each patient.
gender	The patient's gender.
age	The patient's age.
hypertension	Whether the patient has hypertension.
heart_disease	Whether the patient has heart disease.
ever_married	Whether the patient has been ever married.
work_type	The patient's work type.
Residence_type	The patient's residence type.
avg_glucose_level	The average glucose level in the patient's blood.
bmi	The patient's Body Mass Index (BMI).
smoking_status	The patient's smoking status.
stroke	Whether the patient has been diagnosed with having had a stroke.

The dataset was found on Kaggle, and was chosen due to its relatively large size, the varied data types of the attributes, and its high usability ranking calculated by Kaggle.

The dataset has been cited in publications which revealed information about its source. The data of this dataset was collected from 5110 patients from medical clinics/hospitals across Bangladesh. These records were collected for research as they demonstrate the occurrence of certain health conditions which can lead to a stroke (Emon *et al.*, 2020).

1.2 Supervised learning task identification

Newman-Toker *et al.* (2014) found misdiagnosis of stroke accounting for at least 40,000 preventable deaths across the US. This figure may be higher for Bangladesh, a LMIC, where there is poor access to healthcare with only 25% of medical professionals serving the rural areas where 70% of the population reside (OCHA, 2018).

The research team aims to predict the event of a stroke occurring by using machine learning techniques on the dataset chosen in section 1.1. To solve this predictive problem, the attributes in Table 1 must be considered. As all attributes are labelled, this is a supervised machine learning task.

Extracting the dataset into the development environment provided at insight to the data (Figure 1). The final attribute of the dataset records the event of a stroke. Based on the predictive problem, the stroke attribute will be the target variable as this is what is being predicted. Consequently, the remaining attributes will be input variables, known as features.

The target variable only has two possible values, 0 or 1. As these are distinct, categorical values, a classification algorithm will be chosen (Sarangam, 2021). Furthermore, as there only two possible classes, this is a binary classification (Sarker, 2021). Although medically, there are two types of strokes, haemorrhagic and ischaemic, they have presumably been generalised to stroke in this dataset.

```
df_stroke = pd.read_csv('/content/drive/MyDrive/data_sets/healthcare-dataset-stroke-data.csv')
df_stroke
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

5110 rows × 12 columns

Figure 1 Extracting the chosen dataset into the development environment to gain an insight to the data. This was done using Pandas' read_csv function the read the CSV file of the dataset into a Pandas' dataframe.

Before a model is applied, the ground truth should be established. As stated by Kozyrkov (2020), ground truth is an 'ideal expected result.' In machine learning, the ground truth is what is ideally expected of a model's results, but it is highly dependent on the dataset. Ground truth can be established by the 'human decision-maker' who initially labelled the data, the period the data was collected from, or from its location; this makes ground truth subjective to a particular dataset (Kozyrkov, 2020).

For the chosen dataset, the ground truth is established by the target variable where patients have been labelled as 1 or 0 for stroke or no stroke. Other empirical evidence from the real world is also subjective to this dataset including when the data was collected (2020) and its location (Bangladesh).

To conclude, with supervised machine learning techniques, a classification model can be developed to predict the event of a stroke occurring. The ground truth, established by the target variable, will form a baseline for future predictions the model makes.

1.3 Team Identification

Table 2 A table identifying each team member and the classification model they developed for the first iteration of the supervised machine learning task.

Forename	Surname	Student ID	Model(s) developed
B			K-Nearest Neighbour
A			Complement Naïve Bayes
C			Decision Tree Classifier
D			Support Vector Machine

2 Exploratory Data Analysis

The EDA maximises understanding of the dataset through data visualisations. Findings from EDA show how features of the dataset may be distributed and/or correlated, allowing assumptions to be derived about how the model should perform within a problem domain. Furthermore, understanding the features allows appropriate data pre-processing techniques to be chosen.

2.1 Questions identification

Factors that increase risk of stroke include gender, age, hypertension, heart disease, high glucose levels, Body Mass Index (BMI), the level of physical activity and smoking (NHS, 2022). Most factors are included in the dataset as features; this will allow the model's predictions to be compared to current medical research.

Each of these features, the medical research on them and the assumptions made by the research team are detailed in Table 3.

Table 3 Table showing the features of the dataset, with research on them and the assumptions made by the research team.

Feature	Research and Assumption
gender	Females are more likely to experience a stroke due to the use of birth control methods and pregnancies (Centers for Disease Control and Prevention (CDC), 2022). Hence, it is assumed more females in the dataset will have had a stroke.
age	As 'the older you are, the more likely you are to have a stroke,' it is assumed that the age feature will have a positive linear relationship to likelihood of stroke (Centers for Disease Control and Prevention (CDC) - Stroke, 2022).
hypertension	Research shows hypertension increasing an individual's likelihood of a stroke as it damages arteries, leading to reduced blood flow and blockages. Therefore, it is assumed that the likelihood of a patient having hypertension will have a positive linear relationship with likelihood of stroke.
heart_disease	Heart disease also increases an individual's likelihood of a stroke as a blockage near the heart can lead to various heart diseases (Fletcher, 2022). Hence, a positive linear relationship is assumed between the presence of heart disease and an individual's likelihood of having a stroke.
ever_married	There is little research on whether a patient's status affects their likelihood of experiencing a stroke, however, a partial relationship is assumed. This is as older patients are more likely to have been married at some point, and older patients have an increased risk of stroke.

	There is no feature for physical activity in the dataset, but other features can be used to draw an assumption such as work type and residence type. These patients are more likely to get physical activity than those living and working in urban areas as rural areas provide direct access to nature and agricultural work (Surbhi, 2017). Hence, it can be assumed that work type and residence type are partially related, and they both will have a relationship with the likelihood of having a stroke.
work_type	Bangladesh has a rural population of 61.05% (Kameke, 2022). From this it is assumed that many of the patients will work in rural areas, such as agriculture. It is assumed patients working in rural areas will be involved in more physical tasks. Hence, these patients are less likely to have a stroke.
Residence_type	As 61.05% of Bangladesh's population is rural, it is assumed that more patients will belong to rural areas (Kameke, 2022). As rural areas provide direct access to nature and agricultural work, it is likely for patients from rural areas to do more physical activities (Surbhi, 2017). Hence, it is assumed patients from urban areas have a greater likelihood having a stroke than those from rural areas.
avg_glucose_level	High glucose levels are caused by a build-up of glucose in an individual's blood, damaging blood vessels. As high glucose levels contribute to the likelihood of having a stroke, a positive linear relationship is assumed between glucose levels and likelihood of a stroke.
bmi	High BMI triggers other health conditions such as hypertension and high glucose levels. Therefore, a positive linear relationship is assumed between a high BMI and the likelihood of having a stroke.
smoking_status	Research shows that tobacco use increases the likelihood of experiencing a stroke, (*Centers for Disease Control and Prevention (CDC) - Stroke*, 2022). Therefore, it is assumed that patients that have either formerly smoked or currently smoke will be at a greater risk of experiencing a stroke.

Based on medical research and assumptions, the research team derived research questions to explore in the EDA process in section 2.3.

Table 4 Research questions derived by the research team, which will be explored in the EDA process.

Research Question ID	Research Question
1	Do older patients have an increased likelihood of being diagnosed with a stroke?
2	Does have hypertension or heart disease increase a patient's likelihood of being diagnosed with a stroke?
3	Does a specific BMI, along with high glucose levels, increase a patient's likelihood of being diagnosed with a stroke?
4	Does a patient's employment type increase their likelihood of being diagnosed with a stroke?
5	Does being retired increase a patient's likelihood of being diagnosed with a stroke?
6	Does a high workload increase hypertension, which in turn increases a patient's likelihood of being diagnosed with a stroke?

7	Does being unemployed increase BMI levels, which in turn increase a patient's likelihood of being diagnosed with a stroke?
8	Do female patients have an increased likelihood of being diagnosed with a stroke due to balancing home and work lives?

2.2 Splitting the dataset

A discussion on the concept of splitting the dataset can be found in Appendix A – Data Splitting.

2.2.1 Implementation

The research team had to split the dataset whilst minimising risks of data leakage, overfitting or underfitting. These terms have been explained in section 9.1.1 of Appendix A – Data Splitting. Furthermore, it was critical that all members use the same method of data splitting to ensure model development and evaluation is done on the exact same subset of data; allowing us to later compare model performances.

As data leakage could negatively affect each model's performance, it was decided that data splitting will be the first stage after data extraction.

To split the dataset into the known information and what is being predicted, it was split into X and Y, representing the feature and target variables respectively (Figure 2). Keeping what is being predicted aside, minimised data leakage.

```
X = df_stroke.drop(labels=["stroke"], axis='columns')
Y = df_stroke["stroke"]
```

Figure 2 Splitting the dataset into X and Y, which represent the feature and target variables respectively.

From X and Y, training and testing subsets are created. The research team decided that a validation set will be used, allowing model evaluation for potential overfitting or underfitting ahead of model testing.

To create the subsets, Scikit Learn's train_test_split() function was used. As this function only creates four subsets of data, it had to be considered how the function could be re-used to create six subsets.

Figure 3 shows how the function was used to create intermediate and testing subsets from the defined X and Y. The intermediate subset is a temporary subset and will be later split to provide training and validation subsets.

The other parameters used by this function are test size and random state. The test size parameter defines the percentage of the dataset that will be in the test subset. The research team defined the test size as 0.10, meaning 10% of the whole dataset will be the testing subset, and the remaining 90% will be the intermediate subset.

The random state parameter controls the data shuffling. Using the same random state for each team member's model, guarantees a deterministic split meaning the data is randomised in the same manner for each.

```
#using the train_test_split() method to split X and Y into four further subsets of data
#test_size is set to 0.10, to create a split proportional to 90:10
df_stroke_intermediate_X, df_stroke_test_X, df_stroke_intermediate_Y, df_stroke_test_Y = \
  model_selection.train_test_split(X, Y, test_size=0.10, random_state=42)
```

Figure 3 Using Scikit-Learn's model_selection module, and train_test_split() method, to further split X and Y into intermediate and testing subsets at a 90:10 ratio.

The intermediate subset can be split further to create training and validation subsets, again using the train_test_split() function with the same parameters (Figure 4). The 90% intermediate subset was reduced further to create training and validation subsets of 80% and 10% of the whole dataset, respectively.

```
#using the train_test_split() method to split the intermediate subsets into four further subsets of data
#test_size is set to 0.10, to create a split proportional to 90:10
df_stroke_train_X, df_stroke_validate_X, df_stroke_train_Y, df_stroke_validate_Y = \
  model_selection.train_test_split(df_stroke_intermediate_X, df_stroke_intermediate_Y, test_size=0.10, random_state=42)
```

Figure 4 Using the train_test_split() method to split the intermediate set into training and validation subsets at a 80:10 ratio.

```
df_stroke_train = deepcopy(df_stroke_train_X)
df_stroke_train["stroke"] = df_stroke_train_Y

df_stroke_validate = deepcopy(df_stroke_validate_X)
df_stroke_validate["stroke"] = df_stroke_validate_Y

df_stroke_test = deepcopy(df_stroke_test_X)
df_stroke_test["stroke"] = df_stroke_test_Y
```

Figure 5 Using Python's deep copy operation to create new objects containing copies of the objects from the original.

```
df_stroke_train.shape
```

```
(4139, 12)
```

Figure 6 Using Python's shape() method to return the dimensions of the training subset. This shows that the training subset has 4139 records and 12 attributes.

```
(4139/5110) * 100
```

```
80.99804305283757
```

Figure 7 Using division and multiplication operations to find out the proportion of the training subset in relation to the whole dataset. This calculation shows that the training set is proportional to 80% of the whole dataset.

```
df_stroke_validate.shape
```

```
(460, 12)
```

Figure 8 Using Python's shape() method to return the dimensions of the validation subset. This shows that the validation subset has 460 records and 12 attributes.

```
(460/5110) * 100
```

```
9.001956947162427
```

Figure 9 Using division and multiplication operations to find out the proportion of the validation subset in relation to the whole dataset. This calculation shows that the validation set is proportional to 9% of the whole dataset, which is relatively close to being 10%.

```
1  df_stroke_test.shape
```

```
(511, 12)
```

Figure 10 Python's shape() method to return the dimensions of the testing subset. This shows that the testing subset has 511 records and 12 attributes.

```
1  (511/5110) * 100
```

```
10.0
```

Figure 11 Using division and multiplication operations to find out the proportion of the testing subset in relation to the whole dataset. This calculation shows that the testing set is proportional to 10% of the whole dataset.

To conclude, data splitting is a crucial stage as it can affect a model's performance. How well the model was trained affects the model's performance in making a prediction for the unseen data.

2.3 Exploratory Data Analysis process and results

The EDA process provides an understanding of the dataset. The research team decided that first the dataset would be split, and then EDA will be performed on the training set only to prevent data leakage. The visualisations of the EDA process utilise Seaborn, a Python library for statistical data visualisations.

The process and results of the EDA can be found in Appendix B – EDA.

2.4 EDA Findings

The EDA showed how features may be distributed individually and in relation to the target variable. It was found that some patients are more likely to experience a stroke due to the presence of certain factors. These factors include hypertension, heart disease, high BMI level, high glucose levels and use of tobacco. This is based on research showing these factors posing a higher risk.

Additionally, age and gender influence a patient's likelihood of experiencing a stroke. As older patients are more likely to have a stroke, older female patients are presumably at a greater risk.

Other factors have been more difficult to assume, such work type and residence type. Both factors must be considered in relation to the geographical source of the dataset which is Bangladesh. Research shows most of Bangladesh's population lives in the rural. Therefore, it was assumed that a high number of patients will be involved in agricultural work, belonging to the private sector. The EDA on the work type and residence type features confirmed this, but their relation to the target variable of stroke is weak.

Overall, the EDA drew useful insights into the dataset being used as it demonstrated how features may affect the model, and whether this represents the problem and location domain.

3 Experimental Design

3.1 Identification of your chosen supervised learning algorithm(s)

The chosen supervised machine learning algorithm is the K-Nearest Neighbours (KNN) algorithm. How the KNN algorithm works, its benefits and limitations are discussed in Appendix C – KNN.

3.2 How the Chosen Algorithm Can Be Applied

Based on the EDA, KNN was chosen for the predictive problem of whether a patient has had a stroke.

The KNN algorithm is suitable for this machine learning task as it predicts class labels on the assumption of similar points being near each other. This assumption is applicable to the problem, because as found in the EDA, patients that have had a stroke inherit similar characteristics (excluding anomalies). This is supported by research of certain factors more considerably increasing a patient's risk of a stroke. When a patient displays certain characteristics, their risk of stroke can be predicted by comparing to other patients; the KNN algorithm will measure how many *k* neighbours surround this data point from either class.

A limitation of KNN is the curse of dimensionality; the curse of dimensionality is explained in section 9.3.2 of Appendix C – KNN. However, as the chosen dataset only has ten features, this machine learning task is unlikely to be affected by the curse of dimensionality. Additionally, even though KNN is a lazy learner, this is unlikely to affect the performance of this machine learning task as the chosen dataset only has 5110 records.

As this machine learning model would be deployed for healthcare purposes, it is critical that the model makes correct predictions as whether a patient is diagnosed with having a stroke can have life-changing consequences. To ensure this, the model's hyperparameters can be tuned for optimal performance.

3.3 Identification of appropriate evaluation techniques

To evaluate the model's performance, appropriate evaluation techniques must be used. All research team members must use the same evaluation techniques to be able to compare model performances.

First, a baseline score will be derived to compare the model's performance against. This is calculated by dividing the most abundant class by the total number of data values (Dang, 2022). The initial baseline score is calculated as:

Baseline Score = Most Abundant Class / Total Number of Data Values

Baseline Score = 4861 / 5110 = 0.95, giving a baseline score as 0.95 or 95%.

The classification problem has four possible outcomes as outlined in Table 5.

Table 5 A table to show the four possible outcomes of the classification problem as identified as: true positive, false positive, false negative and true negative.

	Actual Class		
		Positive (1 – stroke)	**Negative (0 – no stroke)**

Predicted Class	**Positive (1 – stroke)**	When the machine learning model correctly predicts the patient has had a stroke. This is a true positive, as the model has correctly predicted the positive class.	When the machine learning model incorrectly predicts the patient has had a stroke, where in fact, the patient has not. This is a false positive, as the model has incorrectly predicted the positive class.
	Negative (0 – no stroke)	When the machine learning model incorrectly predicts the patient has not had a stroke, when in fact, the patient has. This is a false negative, as the model has incorrectly predicted the negative class.	When the machine learning model correctly predicts the patient has not had a stroke. This is a true negative, as the model has correctly predicted the negative class.

Then, appropriate evaluation techniques can be chosen to reflect the possible outcomes.

The chosen evaluation techniques are accuracy, confusion matrix, precision, recall and F1 score. Each of these are imported from Scikit Learn's library and have been discussed in Appendix D – Evaluation Metrics.

3.4 Data cleaning and Pre-processing transformations

Data pre-processing transforms raw data into useable data. Raw datasets contain 'redundant information, noisy, and unreliable data,' all which effect model performance (Acheme and Vincent, 2021). Therefore, it is important to transform the data.

The research team had to apply the same data pre-processing transformations to ensure the same, 'transformed' data is used for each model. This ensured that each model is trained on the same data, allowing for comparison.

3.4.1 Dropping Unrequired Attributes

The research team decided to drop the id attribute as there is no reason to identify individual patients. However, in a healthcare setting, the patient records would require a form of identification in the case an anomaly is found and needs to be attended to.

```
1  df_stroke_train
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1108	56734	Male	33.0	0	0	Yes	Govt_job	Urban	82.83	25.4	Unknown	0
3684	25676	Female	7.0	0	0	No	children	Rural	89.38	19.0	Unknown	0
3419	39017	Female	72.0	0	0	Yes	Govt_job	Rural	118.22	21.9	formerly smoked	0
3443	2730	Male	58.0	0	0	Yes	Private	Urban	94.53	36.1	never smoked	0
31	33879	Male	42.0	0	0	Yes	Private	Rural	83.41	25.4	Unknown	1
...	...	...	...	...	...	...	...	...	...	...	...	...
2777	15533	Male	46.0	0	0	No	Private	Urban	107.59	26.2	formerly smoked	0
2492	65324	Female	48.0	0	0	Yes	Govt_job	Rural	75.91	27.8	Unknown	0
3625	40210	Male	78.0	0	1	Yes	Self-employed	Rural	206.62	28.0	formerly smoked	0
2136	59745	Female	27.0	0	0	Yes	Private	Urban	76.74	53.9	Unknown	0
3595	4449	Male	48.0	0	0	Yes	Govt_job	Rural	124.64	26.4	smokes	0

4139 rows × 12 columns

Figure 12 Viewing the training set as it appears with the id attribute. As seen the id attribute holds a unique id value for each patient. The Pandas data frame also assigns each record a unique id value.

```
df_stroke_train = df_stroke_train.drop(columns = 'id')
```

```
df_stroke_train
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1108	Male	33.0	0	0	Yes	Govt_job	Urban	82.83	25.4	Unknown	0
3684	Female	7.0	0	0	No	children	Rural	89.38	19.0	Unknown	0
3419	Female	72.0	0	0	Yes	Govt_job	Rural	118.22	21.9	formerly smoked	0
3443	Male	58.0	0	0	Yes	Private	Urban	94.53	36.1	never smoked	0
31	Male	42.0	0	0	Yes	Private	Rural	83.41	25.4	Unknown	1
...	...	...	...	...	...	...	...	...	...	...	...
2777	Male	46.0	0	0	No	Private	Urban	107.59	26.2	formerly smoked	0
2492	Female	48.0	0	0	Yes	Govt_job	Rural	75.91	27.8	Unknown	0
3625	Male	78.0	0	1	Yes	Self-employed	Rural	206.62	28.0	formerly smoked	0
2136	Female	27.0	0	0	Yes	Private	Urban	76.74	53.9	Unknown	0
3595	Male	48.0	0	0	Yes	Govt_job	Rural	124.64	26.4	smokes	0

4139 rows × 11 columns

Figure 13 By using Pandas' drop() method, the id attribute has been dropped from the training data frame. The 'transformed' training data frame can then be viewed to see that the change has took effect.

3.4.2 Imputation of Missing Values

Missing values refer to any values that have not been stored in the observation. Missing values affect conclusions drawn and reduce the representativeness of the existing data (Kang, 2013). Furthermore, numerous missing values can skew the distribution of data.

As a third-party dataset is being used, it is important to check for missing values so that they can be dealt with appropriately.

The research team decided on the technique of imputation. Imputation is when a missing value is replaced with another value calculated from the available data. The team had to decide which descriptive statistic would be most meaningful to use; an overview is presented in Table 6.

Table 6 An overview of each descriptive statistic that can be used for the SimpleImputer() method. To do this, the SimpleImputer() method provided by Scikit Learn was used. The SimpleImputer() method replaces missing values with a descriptive statistic.

Descriptive Statistic	Description	Reasoning
Mean	An average of all the present data values.	The mean is easily defined and is representative of all the values in the dataset. But the mean cannot be used for categorical data.
Median	The middle value from all the present data values.	As the median is merely the value that is at the middle point of the dataset, it is not affected by outliers. However, this also means that the median is not representative of all the possible values of the dataset.
Mode	The data value most commonly present.	The mode is straightforward, simple to calculate and works with all data types. Yet, in a large dataset, more than one value may appear as the most common. This raises confusion as to what should be defined as the mode.

The mean was identified as the most valuable descriptive statistic to use. Then missing values could be identified and imputed.

```
1  df_stroke_train.isna().sum()
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  164
smoking_status         0
stroke                 0
dtype: int64
```

Figure 14 Using Pandas' isna() and sum() methods to return the sum of the missing values in the training set. This shows that there are 164 missing values for the bmi attribute. Due to the large number of missing values, it is important that the missing values are handled to reduce the chances of the machine learning model being biased.

```
1  msno.matrix(df_stroke_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb335e65e50>
```

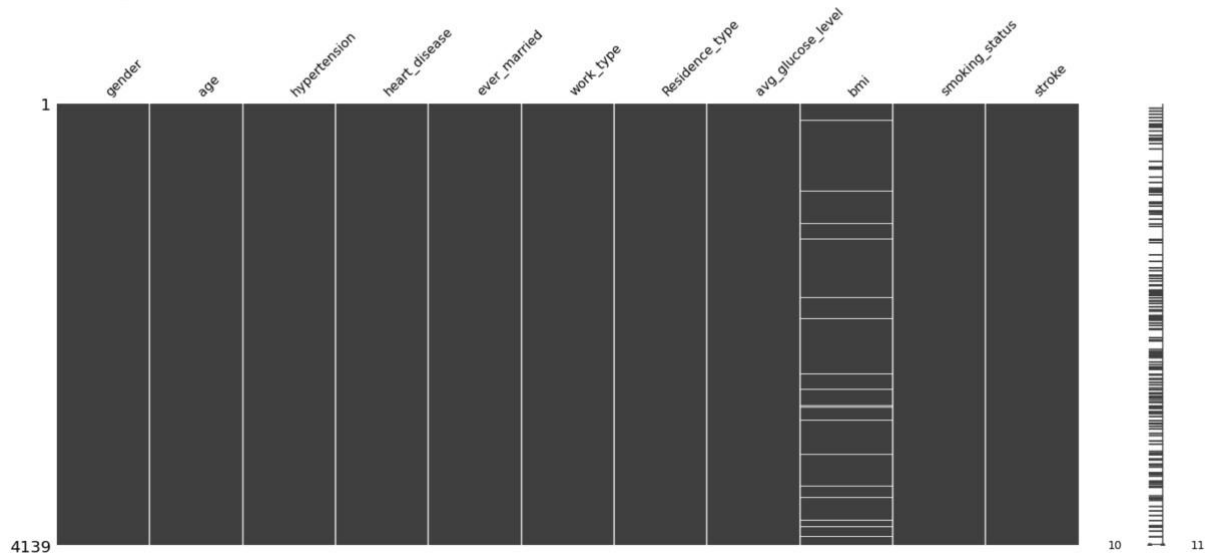


Figure 15 Using the Missingno library to visualise the missing data of the bmi attribute from the training set.

The SimpleImputer() method was used as in Figure 16.

```
1  #impute_attribute function imputes missing values in a specified dataframe and column
2  #impute_attribute function takes parameters: df (the dataframe), column (the column), imputer=imputer (imputer object created from SimpleImputer() met
3  #impute_attribute function used to impute missing values by using the mean of the specified column as the mean takes into consideration all values
4
5  imputer = impute.SimpleImputer(missing_values=np.nan, strategy='mean')
6  def impute_attribute(df, column, imputer=imputer):
7    imputer = imputer.fit(df[[column]])
8    return imputer.transform(df[[column]]), imputer
```

Figure 16 Defining the impute_attribute function which imputes missing values for a specified data frame and column. First, an instance of the SimpleImputer() method is created in line 5. The two parameters for the method are: missing_values and strategy. The missing_values parameter is the placeholder for missing values whilst the strategy parameter is the imputation strategy of descriptive statistic chosen. Following on from this, in line 7, the fit() method is used to store the calculated mean and fit the imputer on the specified column of the specified data frame. Finally, the transform() method is used to transform the specified data frame and column by applying the calculated mean.

```
df_stroke_train['bmi'], bmi_imputer = impute_attribute(df_stroke_train, 'bmi')
df_stroke_train
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1108	Male	33.0	0	0	Yes	Govt_job	Urban	82.83	25.4	Unknown	0
3684	Female	7.0	0	0	No	children	Rural	89.38	19.0	Unknown	0
3419	Female	72.0	0	0	Yes	Govt_job	Rural	118.22	21.9	formerly smoked	0
3443	Male	58.0	0	0	Yes	Private	Urban	94.53	36.1	never smoked	0
31	Male	42.0	0	0	Yes	Private	Rural	83.41	25.4	Unknown	1
...	...	...	...	...	...	...	...	...	...	...	...
2777	Male	46.0	0	0	No	Private	Urban	107.59	26.2	formerly smoked	0
2492	Female	48.0	0	0	Yes	Govt_job	Rural	75.91	27.8	Unknown	0
3625	Male	78.0	0	1	Yes	Self-employed	Rural	206.62	28.0	formerly smoked	0
2136	Female	27.0	0	0	Yes	Private	Urban	76.74	53.9	Unknown	0
3595	Male	48.0	0	0	Yes	Govt_job	Rural	124.64	26.4	smokes	0

4139 rows × 11 columns

Figure 17 Calling the impute_attribute function on the data frame and attribute where the missing values need to imputed; in this case, the df_stroke_train data frame and the bmi attribute. The result of calling the function is held as bmi_imputer.

```
df_stroke_train.isna().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
```

Figure 18 Using Pandas' isna() and sum() methods after calling the impute_attribute function to return the sum of the missing values in the training set. As seen here, there are no longer any missing values in the training set.

3.4.3 Feature Encoding

Machine learning models can only perform with numerical data. Data of other types must be converted into a numerical form before it can be used for machine learning.

```
df_stroke_train.dtypes
```

```
gender               object
age                 float64
hypertension          int64
heart_disease         int64
ever_married         object
work_type            object
Residence_type       object
avg_glucose_level   float64
bmi                 float64
smoking_status       object
stroke                int64
dtype: object
```

Figure 19 Returning the data types of each attribute within the training set.

Table 7 A table of the data types of the attributes in the training set with their descriptions, which attribute uses each data type (with an example), and how the data type converts to a statistical data type.

Data Type	Description	Attributes from the training dataset that use this data type	Example from the training dataset	As a statistical data type

Float	A floating-point number.	age	33.0	Numerical (Continuous - Ratio)
		avg_glucose_level	82.83	Numerical (Continuous - Ratio)
		bmi	25.4	Numerical (Continuous - Ratio)
Int	An integer value.	hypertension	0	Numerical (Discrete)
		heart_disease	0	Numerical (Discrete)
		stroke	0	Numerical (Discrete)
Object	A string value.	gender	Male	Categorical (Nominal)
		ever_married	Yes	Categorical (Nominal)
		work_type	Govt_job	Categorical (Nominal)
		Residence_type	Urban	Categorical (Nominal)
		smoking_status	Unknown	Categorical (Nominal)

The research team decided to convert categorical data into numerical data using dummy encoding, rather than another common approach of one-hot encoding.

Both, dummy encoding and one-hot encoding, create a set of dummy variables. However, one-hot encoding creates *k* number of dummy variables, where *k* is equivalent to the number of possible categories a variable has, Pramoditha (2021).

An example is converting the categorical gender attribute into numerical data.

```
1  df_stroke_train
```

	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	Female	Male
1108	33.0	0	0	Yes	Govt_job	Urban	82.83	25.4	Unknown	0	0	1
3684	7.0	0	0	No	children	Rural	89.38	19.0	Unknown	0	1	0
3419	72.0	0	0	Yes	Govt_job	Rural	118.22	21.9	formerly smoked	0	1	0
3443	58.0	0	0	Yes	Private	Urban	94.53	36.1	never smoked	0	0	1
31	42.0	0	0	Yes	Private	Rural	83.41	25.4	Unknown	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 20 Using one-hot encoding to encode the gender attribute. One-hot encoding creates two dummy variables for the gender attribute as there are two possible categories. If one-hot encoding was used for the gender attribute, it would create two dummy variables: male and female. Figure 20 shows how this would appear. This increases the dimensionality of the training set by two. Also, one attribute remains somewhat redundant, as in this example, if the patient is not a female, they are obviously a male. Therefore, two dummy variables are not entirely required, and a duplicate category is created.

When a duplicate category is created, the dummy variable trap is introduced. This results in multicollinearity, which is when independent variables have a high correlation; making it difficult to interpret a variable individually, hence reducing the reliability of the inferences made (Karabiber, 2022).

Dummy encoding avoids the dummy variable trap as it creates *k-1* number of dummy variables. This means redundant variables are simply not created. Additionally, as dummy encoding always creates one less dummy variable than one-hot encoding, it always results in a smaller dimensionality of the training set.

To use dummy encoding, a create_dummies function has been created as in Figure 21.

```
#create_dummies function converts categorical values into dummy variables for a specified dataframe and column
#create_dummies function takes parameters: df (the dataframe), column (the column), drop_first=True (whether the first column created should be dropped
#create_dummies function uses Pandas' get_dummies which removes duplicate category columns.

def create_dummies(df, column_name, drop_first=True):
  temp_dummies = pd.get_dummies(df[column_name], drop_first=drop_first)
  temp_df = df.drop(labels=[column_name], axis='columns')
  return temp_df.join(temp_dummies)
```

Figure 21 Defining a create_dummies function which converts categorical values into dummy variables. The create_dummies function takes in the following parameters: df (the data frame), column (the column) and drop_first. Fundamentally, the parameter of drop_first differentiates whether dummy encoding or one-hot encoding is used. If the value of drop_first is true, the first dummy variable level is removed, which is dummy encoding. Finally, the function returns a temporary data frame (temp_df) with the new dummy variables.

Once the function has been defined, it can be called on each categorical feature.

```
[59]  df_stroke_train = create_dummies(df_stroke_train, 'gender')
```

Displaying the training set to see how the 'gender' feature has been converted.

```
[60]  df_stroke_train
```

	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	Male
1108	33.0	0	0	Yes	Govt_job	Urban	82.83	25.4	Unknown	0	1
3684	7.0	0	0	No	children	Rural	89.38	19.0	Unknown	0	0
3419	72.0	0	0	Yes	Govt_job	Rural	118.22	21.9	formerly smoked	0	0
3443	58.0	0	0	Yes	Private	Urban	94.53	36.1	never smoked	0	1
31	42.0	0	0	Yes	Private	Rural	83.41	25.4	Unknown	1	1
...	...	...	...	...	...	...	...	...	...	...	...
2777	46.0	0	0	No	Private	Urban	107.59	26.2	formerly smoked	0	1
2492	48.0	0	0	Yes	Govt_job	Rural	75.91	27.8	Unknown	0	0
3625	78.0	0	1	Yes	Self-employed	Rural	206.62	28.0	formerly smoked	0	1
2136	27.0	0	0	Yes	Private	Urban	76.74	53.9	Unknown	0	0
3595	48.0	0	0	Yes	Govt_job	Rural	124.64	26.4	smokes	0	1

4139 rows × 11 columns

Figure 22 Calling the create_dummies function with the parameters of the training set and the column to be converted, in this case gender. As seen, there is no longer a gender attribute. Instead, there is an attribute called 'Male.' This shows that the gender attribute has been converted into a discrete attribute, where 1 represents male and 0 represents not male.

```
[62]  1  df_stroke_train = create_dummies(df_stroke_train, 'ever_married')
```

Displaying the training set to see how the 'ever_married' feature has been converted.

```
[63]  1  df_stroke_train
```

	age	hypertension	heart_disease	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	Male	Yes
1108	33.0	0	0	Govt_job	Urban	82.83	25.4	Unknown	0	1	1
3684	7.0	0	0	children	Rural	89.38	19.0	Unknown	0	0	0
3419	72.0	0	0	Govt_job	Rural	118.22	21.9	formerly smoked	0	0	1
3443	58.0	0	0	Private	Urban	94.53	36.1	never smoked	0	1	1
31	42.0	0	0	Private	Rural	83.41	25.4	Unknown	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
2777	46.0	0	0	Private	Urban	107.59	26.2	formerly smoked	0	1	0
2492	48.0	0	0	Govt_job	Rural	75.91	27.8	Unknown	0	0	1
3625	78.0	0	1	Self-employed	Rural	206.62	28.0	formerly smoked	0	1	1
2136	27.0	0	0	Private	Urban	76.74	53.9	Unknown	0	0	1
3595	48.0	0	0	Govt_job	Rural	124.64	26.4	smokes	0	1	1

4139 rows × 11 columns

Figure 23 Calling the create_dummies function with the parameters of the training set and the column to be converted, in this case ever_married. As seen, there is no longer an ever_married attribute. Instead, there is an attribute called 'Yes.' This shows that the ever_married attribute has been converted into a discrete attribute, where 1 represents the patient has been married and 0 represents the patient has not been married.

```
[65]  1  df_stroke_train = create_dummies(df_stroke_train, 'work_type')
```

Displaying the training set to see how the 'work_type' feature has been converted into multiple columns.

```
[66]  1  df_stroke_train
```

	age	hypertension	heart_disease	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	Male	Yes	Never_worked	Private	Self-employed	children
1108	33.0	0	0	Urban	82.83	25.4	Unknown	0	1	1	0	0	0	0
3684	7.0	0	0	Rural	89.38	19.0	Unknown	0	0	0	0	0	0	1
3419	72.0	0	0	Rural	118.22	21.9	formerly smoked	0	0	1	0	0	0	0
3443	58.0	0	0	Urban	94.53	36.1	never smoked	0	1	1	0	1	0	0
31	42.0	0	0	Rural	83.41	25.4	Unknown	1	1	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2777	46.0	0	0	Urban	107.59	26.2	formerly smoked	0	1	0	0	1	0	0
2492	48.0	0	0	Rural	75.91	27.8	Unknown	0	0	1	0	0	0	0
3625	78.0	0	1	Rural	206.62	28.0	formerly smoked	0	1	1	0	0	1	0
2136	27.0	0	0	Urban	76.74	53.9	Unknown	0	0	1	0	1	0	0
3595	48.0	0	0	Rural	124.64	26.4	smokes	0	1	1	0	0	0	0

4139 rows × 14 columns

Figure 24 Calling the create_dummies function with the parameters of the training set and the column to be converted, in this case work_type. As seen, there is no longer a work_type attribute. Instead, the following attributes are added: 'Never_worked', 'Private', 'Self-employed' and 'children.' This shows that the work_type attribute has been converted into discrete attributes. A 1 in any attribute represents the work type of the patient being that attribute. Likewise, a 0 represents that the patient does not have that work type.

```
[68]  1  df_stroke_train = create_dummies(df_stroke_train, 'Residence_type')
```

Displaying the training set to see how the 'Residence_type' feature has been converted.

```
[69]  1  df_stroke_train
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	smoking_status	stroke	Male	Yes	Never_worked	Private	Self-employed	children	Urban
1108	33.0	0	0	82.83	25.4	Unknown	0	1	1	0	0	0	0	1
3684	7.0	0	0	89.38	19.0	Unknown	0	0	0	0	0	0	1	0
3419	72.0	0	0	118.22	21.9	formerly smoked	0	0	1	0	0	0	0	0
3443	58.0	0	0	94.53	36.1	never smoked	0	1	1	0	1	0	0	1
31	42.0	0	0	83.41	25.4	Unknown	1	1	1	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2777	46.0	0	0	107.59	26.2	formerly smoked	0	1	0	0	1	0	0	1
2492	48.0	0	0	75.91	27.8	Unknown	0	0	1	0	0	0	0	0
3625	78.0	0	1	206.62	28.0	formerly smoked	0	1	1	0	0	1	0	0
2136	27.0	0	0	76.74	53.9	Unknown	0	0	1	0	1	0	0	1
3595	48.0	0	0	124.64	26.4	smokes	0	1	1	0	0	0	0	0

4139 rows × 14 columns

Figure 25 Calling the create_dummies function with the parameters of the training set and the column to be converted, in this case Residence_type. As seen, there is no longer a Residence_type attribute. Instead, there is an attribute called 'Urban.' This shows that the Residence_type attribute has been converted into a discrete attribute, where 1 represents an urban residence type and 0 can be assumed to represent a rural residence type.

```
[71]  1  df_stroke_train = create_dummies(df_stroke_train, 'smoking_status')
```

Displaying the training set to see how the 'smoking_status' feature has been converted.

```
[72]  1  df_stroke_train
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	Male	Yes	Never_worked	Private	Self-employed	children	Urban	formerly smoked	never smoked	smokes
1108	33.0	0	0	82.83	25.4	0	1	1	0	0	0	0	1	0	0	0
3684	7.0	0	0	89.38	19.0	0	0	0	0	0	0	1	0	0	0	0
3419	72.0	0	0	118.22	21.9	0	0	1	0	0	0	0	0	1	0	0
3443	58.0	0	0	94.53	36.1	0	1	1	0	1	0	0	1	0	1	0
31	42.0	0	0	83.41	25.4	1	1	1	0	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2777	46.0	0	0	107.59	26.2	0	1	0	0	1	0	0	1	1	0	0
2492	48.0	0	0	75.91	27.8	0	0	1	0	0	0	0	0	0	0	0
3625	78.0	0	1	206.62	28.0	0	1	1	0	0	1	0	0	1	0	0
2136	27.0	0	0	76.74	53.9	0	0	1	0	1	0	0	1	0	0	0
3595	48.0	0	0	124.64	26.4	0	1	1	0	0	0	0	0	0	0	1

4139 rows × 16 columns

Figure 26 Calling the create_dummies function with the parameters of the training set and the column to be converted, in this case smoking_status. As seen, there is no longer a smoking_status attribute. Instead, the following attributes are added: formerly smoked', 'never smoked', and 'smokes.' This shows that the smoking_status attribute has been converted into discrete attributes. A 1 in any attribute represents the smoking status of the patient being that attribute. Likewise, a 0 represents that the patient does not have that smoking status.

```
1  df_stroke_train.rename(columns={"Yes" : "Married"})
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	Male	Married	Never_worked	Private	Self-employed	children	Urban	formerly smoked	never smoked	smokes
1108	33.0	0	0	82.83	25.4	0	1	1	0	0	0	0	1	0	0	0
3684	7.0	0	0	89.38	19.0	0	0	0	0	0	0	1	0	0	0	0
3419	72.0	0	0	118.22	21.9	0	0	1	0	0	0	0	0	1	0	0
3443	58.0	0	0	94.53	36.1	0	1	1	0	1	0	0	1	0	1	0
31	42.0	0	0	83.41	25.4	1	1	1	0	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2777	46.0	0	0	107.59	26.2	0	1	0	0	1	0	0	1	1	0	0
2492	48.0	0	0	75.91	27.8	0	0	1	0	0	0	0	0	0	0	0
3625	78.0	0	1	206.62	28.0	0	1	1	0	0	1	0	0	1	0	0
2136	27.0	0	0	76.74	53.9	0	0	1	0	1	0	0	1	0	0	0
3595	48.0	0	0	124.64	26.4	0	1	1	0	0	0	0	0	0	0	1

4139 rows × 16 columns

Figure 27 After all categorical attributes have been converted to discrete attributes, the names of the attributes can be changed for better understanding. The 'Yes' attribute, which is meant to represent whether a patient has been ever married, is changed to 'Married' so that, it is clearer as to what this attribute represents.

3.5 Limitations and Options

A limitation in terms of the dataset, was its size compared to other medical datasets. A considerable amount of time was spent finding a dataset that would satisfy the constraints of the research domain. The limited number of datasets led to choosing an imbalanced dataset. In the future we would use a larger dataset, also allowing us to demonstrate more preprocessing techniques.

4 Predictive Modelling / Model Development

4.1 The predictive modelling process

The model development process involved training a machine learning model on the transformed training set.

```
X = df_stroke_train.drop(labels=["stroke"], axis='columns')
Y = df_stroke_train["stroke"]
```

Figure 28 Defining X, the feature variables, and Y, the target variable. X is defined by assigning it to the training set without the target variable which is dropped. Y is defined by assigning it as the target variable attribute only.

Scikit Learn's KNeighborsClassifier is instantiated as KNN was chosen. This is shown in Figure 29. For the KNeighborsClassifier, which implements the KNN algorithm, two parameters have been used: n_neighbors and metric.

The n_neighbors parameter defines the number of neighbours to consider for the majority vote: previously, the number of neighbours has been referred to as *k.* As there is no ideal value of *k*, this has been decided based on the dataset. For this model, *k* has been defined as 101 by considering the size of the training set. The value of *k* must be balanced as a value too small means noise will influence the result, but a value too large leads to 'lower variance but higher bias,' (Navlani, 2018). Also, when the number of classes is even, the value of *k* should be odd.

The metric parameter defines the distance metric to use when calculating the distance between the *k* data points. The distance metric has been defined as Euclidean which is also the default value. The Euclidean distance between two data points is measured by a true straight line between the two points.

```
stroke_clfr_knn = neighbors.KNeighborsClassifier(n_neighbors=101, metric='euclidean')
stroke_clfr_knn = stroke_clfr_knn.fit(X, Y)
```

Figure 29 Using the KNeighborsClassifier with the parameters of n_neighbors, as 101, and metric as Euclidean. The classifier is then fitted onto X and Y, the feature and target variables.

Next, Scikit Learn's predict() method is used as shown in Figure 30.

```
stroke_Y_train_knn = stroke_clfr_knn.predict(X)
```

Figure 30 The predict() method uses the fitted parameters to perform a prediction on new data points (Myrianthous, 2021).

After model training, the model can be evaluated on the seen, training data.

4.2 Evaluation results on “seen” data

Evaluation of the model’s performance on the seen, training data involved using the evaluation metrics decided in section 3.3.

The initial baseline score is calculated as:

Baseline Score = Most Abundant Class / Total Number of Data Values Baseline

Score = 4861 / 5110 = 0.95.

4.2.1 Accuracy

The first metric to evaluate the model’s performance on the seen, training data is accuracy. The accuracy score is retrieved using Scikit Learn’s accuracy_score function; this is shown in Figure 31.

```
1  stroke_acc_train_knn = metrics.accuracy_score(df_stroke_train_Y, stroke_Y_train_knn)
2  accuracy_score = stroke_acc_train_knn * 100
```

Figure 31 Calculating the accuracy score for the seen, training data by using the accuracy_score from Scikit Learn’s metrics library. This function takes in two parameters: df_stroke_train_Y which is the ground truth labels for what is being predicted, and stroke_Y_train_knn which is the predictions made by the KNN classifier. The accuracy score that is retrieved from this is stored as a variable and multiplied by 100 to represent the score as a percentage.

The accuracy is denoted as how often the model correctly predicts either class, I.e. whether a patient has had a stroke. As seen in Figure 32, this model has an accuracy of 95%.

```
1  print(accuracy_score)
```

```
95.14375453007973
```

Figure 32 Using Python’s print statement to return the calculated accuracy score.

The accuracy score for the model aligns with the baseline score but is questionably high. This raises concerns as to whether the accuracy paradox is in play. The accuracy paradox is caused by class imbalance, which as seen in section 9.2.1, does exist. An accuracy of 95% shows that the model can make a correct prediction 95% of the time, regardless of class. Therefore, the majority of the 95% correct predictions may belong to the negative class due to it being the majority class. Even though the accuracy score is high, it is likely to be highly skewed towards the majority class which is where patients are not diagnosed with having had a stroke.

Furthermore, the accuracy of model testing on unseen data will need to be compared to this accuracy score. As a low accuracy in model testing will indicate overfitting as the model has been unable to generalise its learning.

4.2.2 Confusion Matrix

The following metric to evaluate the model’s performance on the seen, training data is the confusion matrix.

The function for a confusion matrix can be retrieved from Scikit Learn’s metric library as shown in Figure 33.

```
1  stroke_cm_train_knn = metrics.confusion_matrix(df_stroke_train_Y, stroke_Y_train_knn)
```

Figure 33 Calculating the confusion matrix for the seen, training data by using the confusion_matrix from Scikit

Learn's metrics library. This function takes in two parameters: df_stroke_train_Y which is the ground truth labels for what is being predicted, and stroke_Y_train_knn which is the predictions made by the KNN classifier.

Figure 34 shows how the confusion matrix is displayed.

```
2  print(stroke_cm_train_knn)
```

```
[[3938    0]
 [ 201    0]]
```

Figure 34 Using Python's print statement to return the calculated confusion matrix. Each row of the confusion matrix corresponds to the predicted class, and each column corresponds to the actual class (Brownlee, 2016c) .

The confusion matrix shows the four possible outcomes from this machine learning task as explained in Table 5 of section 3.3. From the confusion matrix it can be understood that there were 3938 true negative predictions, 0 true positive predictions, 201 false negative predictions and 0 false positive predictions. This shows the model was able to correctly make 3938 predictions where the patient has not had a stroke. For another 201 predictions, the model predicted that the patient has not had a stroke when they have. As discussed, these results are likely down to the class imbalance. The majority class is where the patients have not had a stroke, which is what the model is correctly predicting for a high number of predictions.

To aid the interpretation of the confusion matrix, it can be transformed into a colour-coded heatmap. This is shown in Figure 35.

```
1  sns.heatmap(stroke_cm_train_knn, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa12174610>
```

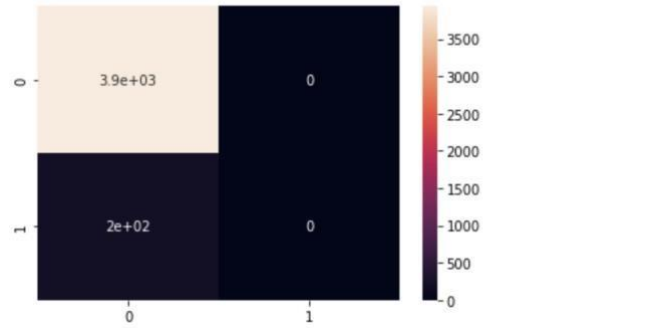


Figure 35 A heatmap from the Seaborn library to aid the interpretation of the confusion matrix. The heatmap, created from the Seaborn library, is passed in two parameters. These are the confusion matrix that has been created already, and a value of true for the heatmap to be annotated with the data values for each cell.

4.2.3 KNN Classification Report

The metric of precision can be calculated as such:

Precision = True Positives / (True Positives + False Positives)

Precision = 0 / (0 + 0) = 0

This precision is as the model was not able to make any predictions where the patient had had a stroke, which means the model was not precise.

Similarly, recall can be calculated as such:

Recall = True Positives / (True Positives + False Negatives)

Recall = 0 / (0 + 201) = 0

The recall value is 0, which represents a poor recall. This means all the patients from the training set that have had a stroke, the model was unable to correctly retrieve any of these.

As a result of the calculated precision and recall scores, the calculated F1 score of 0 is also poor. The F1 score has not surpassed the calculated baseline score of 0.95.

```
1  print("KNN Classification Report: \n%s" % metrics.classification_report(df_stroke_train_Y, stroke_Y_train_knn))
```

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

           0       0.95      1.00      0.98      3938
           1       0.00      0.00      0.00       201

    accuracy                           0.95      4139
   macro avg       0.48      0.50      0.49      4139
weighted avg       0.91      0.95      0.93      4139
```

Figure 36 Using the KNN classification report from Scikit Learn's metrics library. The classification library shows the precision, recall, f1 scores and support for each class. Additionally, it shows the accuracy, macro average and weighted averages.

5 Evaluation and further modelling improvements

5.1 Initial evaluation comparison

To evaluate the model's performance on unseen data, as would happen in a real-world problem, the validation set has been used. The validation set has been transformed using the pre-processing techniques that were used for the training set. However, the validation set has been kept as unseen data as no EDA has been performed on it.

Similarly, the model's performance on the unseen, validation set has been evaluated against the same metrics used for evaluating the model's performance on seen data as in section 4.2. The values obtained for each metric are shown in Table 8.

The code to obtain the results of the unseen, validation set have been omitted from this section but can be found in Appendix E - Validation Set Evaluation.

Table 8 A table summarising the values obtained for the chosen evaluation metrics for the seen, training set and the unseen, validation set. This shows that the model performed equally poorly on seen and unseen data. Although the accuracy score is high, this is likely to be due to class imbalance, which essentially invalidates such a high accuracy.

Evaluation Metric	**Seen, training set**	**Unseen, validation set**
Baseline Score	0.95	Not calculated as the distribution of the classes for the unseen data is unknown.
Accuracy	95%	96%
Confusion Matrix	[[3938 0] [201 0]]	[[440 0] [20 0]]
Precision	0	0
Recall	0	0
F1 Score	0	0

5.1.1 Research Team Comparison

As part of this evaluation on unseen data, Table 9 has been created to show how each team member's model performs on unseen data.

Table 9 A table summarising the values obtained for the chosen evaluation metrics for the unseen, validation set by each team member's classification model.

	Accuracy	**Confusion Matrix**	**Precision**	**Recall**	**F1 Score**
KNN	96%	[[440 0] [20 0]]	0	0	0

Complement Naïve Bayes	69%	[[304 136] [5 15]]	0.1	0.75	0.18
Decision Tree Classifier	95%	[[439 1] [20 0]]	0	0	0
Support Vector Machine	96%	[[440 0] [20 0]]	0	0	0

As seen in Table 9, the KNN, decision tree classifier and support vector machine have all performed terribly in terms of precision and recall. Similarly, these three classifiers have the highest accuracy scores, which as discussed are due to a class imbalance in the dataset.

The complement naïve bayes has a more reasonable accuracy, despite the class imbalance. This is because, as described by (Rennie *et al.*, 2003), the complement naïve bayes classifier was developed to correct the 'severe assumptions' of the standard naïve bayes classifier and hence is appropriate for imbalanced datasets. Even though the accuracy score for the complement naïve bayes classifier does not surpass the baseline score, the accuracy is a truer representation of the model's performance, unlike the other classifiers. This classifier may have low precision but does have a fair recall score. This shows this model will be able to predict a great number of positive classes (i.e patients that have had a stroke), but some negative classes may be identified as positive. For a medical dataset this is acceptable, as it would be safer to mistakenly monitor more patients rather than miss out on high-risk patients.

5.2 Final Evaluation results

The model was iteratively improved for the final test; this can be found in Appendix F - Further Modelling Improvements. After model improvements, it is evident that the third model, using the stratified data split and SMOTE, has had the truest performance. This is as the initial and second model were highly influenced by class imbalance, leading to exceedingly high accuracy scores on unseen data.

Therefore, the final model was used for the final evaluation on the unseen, testing data. Ideally, the testing data replicates real-world data as it has been kept as unseen by the model. Additionally, this is the final evaluation, meaning regardless how the model performs, no improvements will be made. Table 10 summarises the results.

Table 10 A table summarising the values obtained for the chosen evaluation metrics on the unseen, testing data using the final, improved model.

Evaluation Metric	**Unseen, testing set**
Baseline Score	Not calculated as the distribution of the classes for the unseen data is unknown.
Accuracy	69%
Confusion Matrix	[[334 152] [7 18]]
Precision	0.1
Recall	0.7
F1 Score	0.2

Although the accuracy score does not exceed the baseline score, it is justified for the model's performance on unseen data.

The confusion matrix shows that the model made 18 true positive and 334 true negative predictions. Even though the values from the confusion matrix are not optimal, they are

improved from the initial model evaluation. This is due to the data being trained on an imbalanced dataset.

The recall, which is more important in a medical setting, is closer to a score of perfect recall than previous attempts. The recall shows that from all patients that have had a stroke, the model has retrieved a proportion of 0.7. Therefore, the model has only missed on a small number of positive predictions, which could be retrieved with further improvements to the model.

The weighted average of the F1 score has decreased further to 0.69, confirming the absence of class imbalance.

```
1  print("KNN Classification Report: \n%s" % metrics.classification_report(Y, stroke_Y_test_knn))
```

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

           0       0.98      0.69      0.81       486
           1       0.11      0.72      0.18        25

    accuracy                           0.69       511
   macro avg       0.54      0.70      0.50       511
weighted avg       0.94      0.69      0.78       511
```

Figure 37 KNN Classification report for the final model's (with stratified split and SMOTE) performance on the unseen, testing data set.

6 Conclusion

6.1 Summary of results

The final model development evaluation shows that the model had a truer performance as it has not been skewed by class imbalance. The model had a relative accuracy that is acceptable for unseen data. More importantly, the model had a good recall value of 0.7. This demonstrates the model only missed on a small number of positive predictions. Furthermore, the absence of class imbalance is confirmed by the F1 score.

6.2 Suggested further improvements to the model development process

To reflect on the model development process, a few things went well. This is considering that the chosen dataset had a crucial class imbalance at a proportion of 95% to 5% between the no stroke and stroke class respectively. Firstly, focusing on two main parameters of KNN allowed me to thoroughly understand what they represent and how they could be optimised. Secondly, I was able to identify and improve the model for the class imbalance. These two experiences were a learning curve but provided the opportunity to further research KNN and the dataset's intricacies. Nevertheless, I believe there still are improvements that could be implemented in my model development process.

With the experience I've gained through this project, I believe the following improvements can be implemented:

1. The first improvement would be to understand the data initially better, without regards to the task. Understanding the data more thoroughly, would've allowed me to better see the story the data tells. From this I would have identified the class imbalance earlier on and implemented SMOTE as a stage of pre-processing for initial model development. This experience has shown me the importance of using data visualisations to better understand the whole dataset early on.

2. The second improvement would be to experiment with my model's parameters. As I understand the parameters KNN uses, next time I would experiment with them further to refine the model.

6.3 Reflection on Research Team

Working within a research team allowed us to clarify our understanding with one another. From this we were able to individually draw assumptions about our dataset and compare how they vary from one another. This provided a holistic understanding of the dataset. As each member of the team used a different algorithm, there were some confusions about which preprocessing techniques and evaluation metrics should be used. To mitigate an issue arising, we chose the fundamental pre-processing techniques and evaluation metrics to be used by all. This provided each of us time to experiment with other measures for model improvement.

6.4 Reflection on Individual Learning

Initially, I approached this coursework with naivety, as I believed it would be straightforward. However, my experience was quite contradictory.

The first challenge I faced was deciding which pre-processing techniques and evaluation metrics should be used by the research team. This was mitigated by choosing the fundamental pre-processing techniques and evaluation metrics to be used by all.

Once challenges within the team were solved, I moved on to writing the pipeline of the model. The challenges I faced here came after creating and executing the model on the training data. This resulted in a macro average of 0.49 and an accuracy of 95%. After further investigation into the shape of the data, I realised that the data was skewed towards the no stroke class. Then, I had to learn the importance of class imbalance and how it negatively affected results. Also, through model improvement, I learnt that the best solution for class imbalance is resampling the minority class in the form of SMOTE.

Despite the challenges, I learnt many important things. The first being the importance of thoroughly examining the data after finding out that the shape of the data could heavily affect my model. This helped me to improve my model for consistent predictions.

Also, I learnt the importance of teamwork. Taking the initiative to research and help lead the group was a big learning opportunity. I believe stepping out of my comfort zone, both in terms of academic and social skills, was demonstrated in this project.

I believe the best aspects of this project came from the long hours of research, as it motivated me to explore new concepts in data science. Learning about the mathematics of the relevancy of different scores such as F1 macro average, precision, and recall was one such example. From this learning I understood the performance of the model and how my improvements to the data would affect these scores in much more depth. It showed me that with the right time, thought, and work, I could see a career for myself in this field.

In the future I will focus on learning more about data science and the intricacies of AI. This project, I believe, was the first time I encountered the concept of class imbalance. This concept is an example of things I still don't know about machine learning. I will improve my knowledge on such concepts and their remedies and/or processes by working on AI-based personal projects and taking part in Kaggle ML competitions. These would give me the space and time to learn these concepts without an impending deadline.

7 References

Acheme, I.D. and Vincent, O.R. (2021) 'Machine-learning models for predicting survivability in COVID-19 patients', in *Data Science for COVID-19*. Elsevier, pp. 317–336. Available at: https://doi.org/10.1016/B978-0-12-824536-1.00011-3.

Afonja, T. (2017) *Accuracy Paradox*. Available at: https://towardsdatascience.com/accuracyparadox-897a69e2dd9b (Accessed: 30 December 2022).

Altman, N. and Krzywinski, M. (2015) 'Association, correlation and causation', *Nature Methods*, 12(10), pp. 899–900. Available at: https://doi.org/10.1038/nmeth.3587.

Baheti, P. (2023) *Train Test Validation Split: How To & Best Practices [2023]*. Available at: https://www.v7labs.com/blog/train-validation-test-set (Accessed: 14 January 2023).

Brownlee, J. (2016a) *Data Leakage in Machine Learning*. Available at: https://machinelearningmastery.com/data-leakage-machine-learning/ (Accessed: 9 January 2023).

Brownlee, J. (2016b) *Overfitting and Underfitting With Machine Learning Algorithms*. Available at: https://machinelearningmastery.com/overfitting-and-underfitting-with-machinelearningalgorithms/#:~:text=Overfitting%3A%20Good%20performance%20on%20the,poor%20gener alization%20to%20other%20data (Accessed: 7 January 2023).

Brownlee, J. (2016c) *What is a Confusion Matrix in Machine Learning*. Available at: https://machinelearningmastery.com/confusion-matrix-machine-learning/ (Accessed: 11 January 2023).

Brownlee, J. (2020a) *How to Avoid Data Leakage When Performing Data Preparation*. Available at: https://machinelearningmastery.com/data-preparation-without-data-leakage/ (Accessed: 9 January 2023).

Brownlee, J. (2020b) *Train-Test Split for Evaluating Machine Learning Algorithms*. Available at: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learningalgorithms/ (Accessed: 9 January 2023).

Centers for Disease Control and Prevention (CDC) - Stroke (2022). Available at: https://www.cdc.gov/stroke/index.htm (Accessed: 23 November 2022).

Chawla, N. v. *et al.* (2002) 'SMOTE: Synthetic Minority Over-sampling Technique', *Journal of Artificial Intelligence Research*, 16, pp. 321–357. Available at: https://doi.org/10.1613/jair.953.

Cook, A. and B, D. (no date) *Data Leakage*. Available at: https://www.kaggle.com/code/alexisbcook/data-leakage (Accessed: 9 January 2023).

Dang, T. (2022) *Guide to accuracy, precision, and recall*. Available at: https://www.mage.ai/blog/definitive-guide-to-accuracy-precision-recall-for-productdevelopers (Accessed: 12 January 2023).

Donges, N. (2018) *Data Types in Statistics*. Available at: https://towardsdatascience.com/data-types-in-statistics-347e152e8bee (Accessed: 9 January 2023).

Emon, M.U. *et al.* (2020) 'Performance Analysis of Machine Learning Approaches in Stroke Prediction', in *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, pp. 1464–1469. Available at: https://doi.org/10.1109/ICECA49313.2020.9297525.

IBM (2022) *What is the k-nearest neighbors algorithm?* Available at: https://www.ibm.com/uk-en/topics/knn (Accessed: 30 December 2022).

Johnson, J.M. and Khoshgoftaar, T.M. (2019) 'Survey on deep learning with class imbalance', *Journal of Big Data*, 6(1), p. 27. Available at: https://doi.org/10.1186/s40537019-0192-5.

Kameke, L. von (2022) *Share of rural population in Bangladesh from 2012 to 2021*. Available at: https://www.statista.com/statistics/760934/bangladesh-share-of-rural-population/ (Accessed: 28 December 2022).

Kang, H. (2013) 'The prevention and handling of the missing data', *Korean Journal of Anesthesiology*, 64(5), p. 402. Available at: https://doi.org/10.4097/kjae.2013.64.5.402.

Karabiber, F. (2022) *Dummy Variable Trap*. Available at: https://www.learndatasci.com/glossary/dummy-variable-trap/ (Accessed: 2 January 2023).

Korstanje, J. (2021) *The F1 score*. Available at: https://towardsdatascience.com/the-f1score-bec2bbc38aa6 (Accessed: 12 January 2023).

Kozyrkov, C. (2020) *What is "Ground Truth" in AI? (A warning.)*. Available at: https://towardsdatascience.com/in-ai-the-objective-is-subjective-4614795d179b (Accessed: 23 November 2022).

Kozyrkov, C. (2022) *MFML 044 - Precision vs recall*. Available at: https://www.youtube.com/watch?v=BYQQlCVt4aE&t=7s (Accessed: 12 January 2023).

Leung, K. (2022) *Micro, Macro & Weighted Averages of F1 Score, Clearly Explained*. Available at: https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-scoreclearly-explained-b603420b292f (Accessed: 16 January 2023).

Navlani, A. (2018) *KNN Classification Tutorial using Scikit-learn*. Available at: https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn (Accessed: 25 November 2022).

Pramoditha, R. (2021) *Encoding Categorical Variables: One-hot vs Dummy Encoding*. Available at: https://towardsdatascience.com/encoding-categorical-variables-one-hot-vsdummy-encoding-6d5b9c46e2db (Accessed: 12 January 2023).

Raschka, S. (2018) 'STAT479: Machine Learning Lecture Notes'. Available at: chromeextension://efaidnbmnnnibpcajpcglclefindmkaj/https://sebastianraschka.com/pdf/lecturenotes/stat479fs18/02_knn_notes.pdf (Accessed: 30 December 2022).

Rennie, J.D.M. *et al.* (2003) 'Tackling the Poor Assumptions of Naive Bayes Text Classifiers', *Proceedings of the 20th international conference on machine learning*, pp. 616–623. Available at: chromeextension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.aaai.org/Papers/ICML/2003/ICML 03-081.pdf?ref=https://githubhelp.com (Accessed: 16 January 2023).

Sandilands, D. (2014) 'Bivariate Analysis', in *Encyclopedia of Quality of Life and Well-Being Research*. Dordrecht: Springer Netherlands, pp. 416–418. Available at: https://doi.org/10.1007/978-94-007-0753-5_222.

Sarangam, A. (2021) *Classification vs Regression: An Easy Guide in 6 Points*. Available at: https://www.jigsawacademy.com/blogs/ai-ml/classification-vsregression#:~:text=The%20key%20distinction%20between%20Classification,Spam%20or% 20Not%20Spam%2C%20etc. (Accessed: 15 January 2023).

Sarker, I.H. (2021) 'Machine Learning: Algorithms, Real-World Applications and Research Directions', *SN Computer Science*, 2(3), p. 160. Available at: https://doi.org/10.1007/s42979021-00592-x.

T, D. (2019) *Confusion Matrix Visualization*. Available at: https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea (Accessed: 11 January 2023).

Wood, T. (2022) *What is the F-score?* Available at: https://deepai.org/machine-learningglossary-and-terms/f-score (Accessed: 12 January 2023).

8 Bibliography

Altman, N. and Krzywinski, M. (2015) 'Association, correlation and causation', *Nature Methods*, 12(10), pp. 899–900. Available at: https://doi.org/10.1038/nmeth.3587.

Chawla, N. v. *et al.* (2002) 'SMOTE: Synthetic Minority Over-sampling Technique', *Journal of Artificial Intelligence Research*, 16, pp. 321–357. Available at: https://doi.org/10.1613/jair.953.

Kozyrkov, C. (2022) *MFML 044 - Precision vs recall*. Available at: https://www.youtube.com/watch?v=BYQQlCVt4aE&t=7s (Accessed: 12 January 2023).

T, D. (2019) *Confusion Matrix Visualization*. Available at: https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea (Accessed: 11 January 2023).

Wood, T. (2022) *What is the F-score?* Available at: https://deepai.org/machine-learningglossary-and-terms/f-score (Accessed: 12 January 2023).

9 Appendices

9.1 Appendix A – Data Splitting

To build a machine learning model that makes reliable predictions, it is essential that the model is trained and tested on different sets of data. This allows comparisons to be made in terms of

how well a model was trained and then, how well this model performs on new data. The stage of creating these subsets of data is known as data splitting. Data splitting involves splitting the initial dataset into subsets which remain representative of the problem domain.

A dataset can be split using Scikit Learn's train_test_split() function. This function takes a series of parameters to randomly split the dataset into two subsets: a training set and a testing set. Arguably, the most important parameter to configure is the size of the training or testing set, which is represented by train_size or test_size, respectively. The size of the training and testing sets represent what percentage will be designated to each from the complete dataset. As there is 'no optimal split percentage,' other factors are considered such as the dataset's size and computational costs (Brownlee, 2020b). A common split percentage is the 80:20 ratio to the training and testing set respectively.

The stage of data splitting should occur immediately after a chosen dataset has been extracted into the development environment. This is because each subset of data serves a different purpose and so should be handled differently.

The training set is used in model training. The model learns the patterns within the training set and hence this data becomes known as 'seen' data. Before the model is trained, EDA is performed on the training set only, as this is the only subset the model will see to learn from. The testing set is kept as 'unseen' data to provide an unbiased evaluation of the model's performance. Therefore, the testing set mimics real-world data as the model has no idea about the patterns the data may, or may not, contain. The model's performance in the model testing stage reveals how well the model was trained.

If the model performs similarly well in both stages, model training and model testing, it demonstrates that the data within the training set was varied. A varied training set allows the model to take what has been learnt and readily apply it to new, unseen data. But this is not always the case.

9.1.1 Issues That May Arise

9.1.1.1 Data Leakage

When a model performs exceedingly well in the model testing stage, compared to its model training stage, this raises concerns about data leakage. Data leakage is when the training set has information about what is being predicted. This enables the model to learn something it should not know, leading to an exceeding performance in model testing (Brownlee, 2016a). A model that performs exceedingly well in the stage of model testing, essentially, invalidates its own performance.

Primarily, there are two types of data leakage. The first is target leakage. Target leakage is when the training set contains data that will not be available when the predictions need to be made. In target leakage, certain data may not be accessible due to the timing of when it is made available (Cook and B, no date).

The second type of data leakage is the train-test contamination and is considered a naïve approach to data preparation (Brownlee, 2020a). This is when the training set contains information from what should be kept as unseen data, i.e. the validation or testing set. The train-test contamination occurs when data pre-processing techniques are applied holistically to the whole dataset.

9.1.1.2 Overfitting and Underfitting

Conversely, if the model performs well in the model training stage but not in the model testing stage, this indicates overfitting. Overfitting is when the model learns patterns, noise, and fluctuations from the training set as concepts. When the model is tested, the same concepts

cannot be applied as there are different patterns in the unseen data, resulting to poor performance,(Brownlee, 2016b).

Fortunately, certain techniques can limit model overfitting. One technique is introducing a validation set. A validation set is treated as unseen data, and the stage of model validation sits between model training and model testing. Using a validation set allows the model's hyperparameters to be tuned and only the most optimal parameter value is used for model testing.

Another technique is introducing cross-validation. As with a validation set, cross-validation also allows a model's hyperparameters to be repeatedly tuned until the optimal parameter value is found. An advantage of cross-validation, over a single validation set, is that crossvalidation provides several validation sets for iterative tuning of hyperparameters.

Finally, underfitting is when a model performs poorly in both stages of model training and model testing (Brownlee, 2016b). Underfitting demonstrates that the model is not suited to the data it is being used for and the best solution is to use a different machine learning algorithm.

9.2 Appendix B – EDA

9.2.1 Analysing the Target Variable

As discussed in section 1.2, the stroke attribute is the target variable as this is what is being predicted. Categorical attributes can be visualised using a count plot from the Seaborn library. This is shown in Figure 38, where 1 and 0 represent whether a patient has had a stroke.

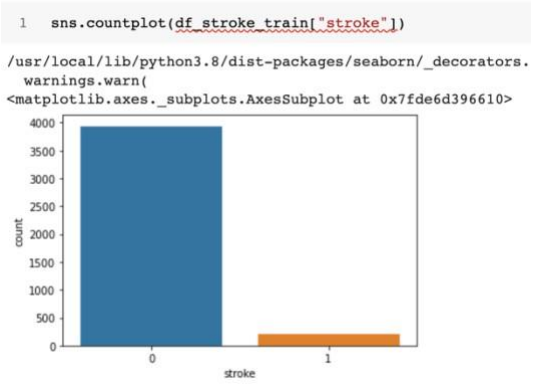


Figure 38 A count plot to represent the number of patients from the training set who have (1) and who have not (0) been diagnosed as having had a stroke, demonstrating that this a categorical data type. This shows that 3900 (approx.) patients have not been diagnosed as having had a stroke, and 200 patients (approx.) have been.

As the background research found that a vast majority of stroke-related burdens occur in LMICs, such as Bangladesh, it was assumed that many patients in the dataset will have had a stroke. However, the count plot contradicts this assumption as the count plot shows a far smaller number of patients than expected that have had a stroke. This raises concern for the potential presence of class imbalance within the training set.

Class imbalance is when one class contains 'significantly fewer samples than the other class,' thus being referred to as the minority and majority class (Johnson and Khoshgoftaar, 2019). Work by Johnson and Khoshgoftaar (2019), states that class imbalance and skewed data 'exists in many real-world applications,' with medical diagnosis being a common scenario for this. This shows that the potential class imbalance within the training set is likely to be representative of the problem domain.

9.2.2 Analysing the Feature Variables

9.2.2.1 Analysing 'gender'

Research shows that females are more likely to experience a stroke. Hence, it is assumed that from the patients of the training set that have been diagnosed with having had a stroke, there will be more female patients.

```
1  sns.countplot(df_stroke_train["gender"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fde6d373580>
```

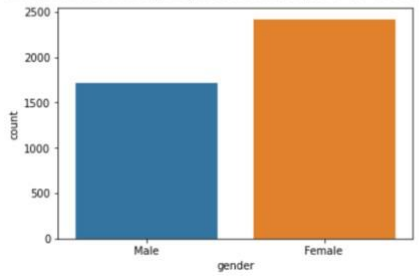


Figure 39 The gender attribute represents the genders of the patients in the training set. A count plot can be used represent the number of males and females present in the training set. It is evident that in the training set there are only two genders, which can be represented using a count plot. Clearly, there are more females (2400 approx.) than males (1700 approx.) in the training set.

9.2.2.2 Analysing 'age'

As research shows the likelihood of having a stroke increases with age, it is assumed that older patients in the training set are more likely to have had a stroke.

```
1  sns.boxplot(df_stroke_train["age"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e7da15370>
```

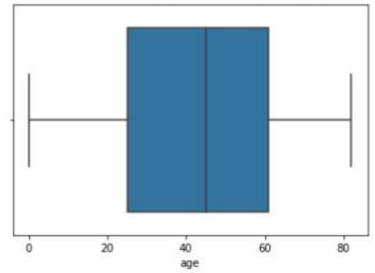


*Figure 40 The age attribute is a ratio data type and can be represented as a box plot*Click or tap here to enter text.*. The box plot shows that the ages of the patients in the training set range for 0 to 85 years. The lower quartile of this box plot shows that 25% of patients in the training set are equal to or below the age of 25 years (approx.). Likewise, the upper quartile of the box plot shows that 75% of the patients in the training set are equal to or below the age of 60 years. Hence, only 25% of the patients from the training set are aged between 60 and 80 years. The median of this box plot shows that the middle age value of all the patients from the training set is 45 years (approx.).*

Figure 41 is a histogram to understand how many patients are within each age range. The kde parameter plots the Kernel Density Estimate to show the shape of the data's distribution. The somewhat negative skew of the distribution shows that as the age increases, so does the number of patients in the training set. Additionally, the histogram shows which age range of patients the data was collected from.

```
1  sns.histplot(df_stroke_train["age"], bins=10, kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb44cfbe400>
```

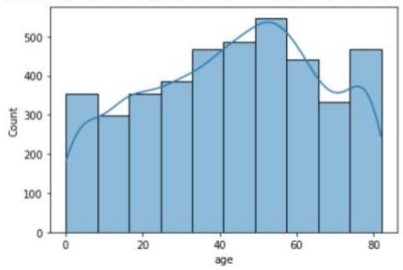


Figure 41 A histogram representing the ages of the patients within the training set. The bins have the default value of 10 to group the age ranges so that they are easier to read. This shows that the most common age range for patients in the training set is 50 to 55 years.

9.2.2.3 Analysing 'hypertension'

```
1  sns.countplot(df_stroke_train["hypertension"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fde67b30580>
```

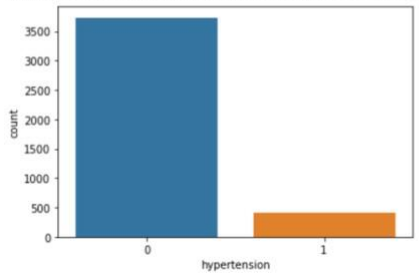


Figure 42 A count plot to represent the number of patients from the training set that do (1), and do not (0), have hypertension. Hypertension is a categorical data type as there are two categories where the numbers don't hold mathematical significance. This shows that in the training set 3700 (approx.) patients do not have hypertension and 400 (approx.) patients do.

Supported by research, patients with hypertension are more likely to experience a stroke. From this an initial assumption is that the hypertension attribute will reflect the target variable where patients in the positive class for either will be proportional. This is also supported by Figure 42.

9.2.2.4 Analysing 'heart_disease'

```
1  sns.countplot(df_stroke_train["heart_disease"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fde6d1a4a90>
```

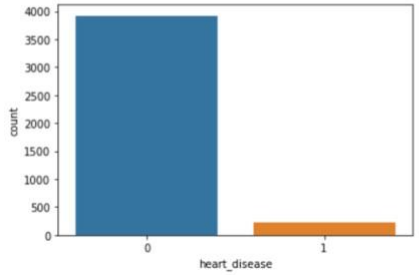


Figure 43 A count plot to represent the number of patients from the training set who do (1) and who do not (0) have heart disease. Heart disease is another categorical data type. This shows that 3900 (approx.) patients don't have heart disease but 250 (approx.) patients do. Even though only a small number of patients have heart disease, this is proportional to the relatively low number of patients in the training set that have hypertension and/or have had a stroke

It is assumed that the number of patients with heart disease will reflect the number of patients that have hypertension and/or have had a stroke. This assumption is represented in the count plot (Figure 43).

9.2.2.5 Analysing 'ever_married'

```
1  sns.countplot(df_stroke_train["ever_married"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fde6d0f2b20>
```

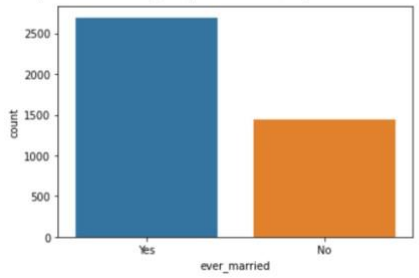


Figure 44 A count plot representing the number of patients from the training set that have been ever married. This attribute is another categorical data type. This shows that from the training set 2800 (approx.) patients have been married at some point of their life and 1400 (approx.) have not been. The number of patients that have been married is almost double the number of patients that have not. This supports the assumption of the age range within the training set being higher.

Based on research, it is assumed that many patients in the training set will be married. This is because stroke is more prevalent in older ages, with its likelihood increasing as a patient ages. The older a patient is, the more likely it is for them to have been married at some point.

9.2.2.6 Analysing 'work_type'

```
1  sns.countplot(df_stroke_train["work_type"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fde6d0b6c40>
```

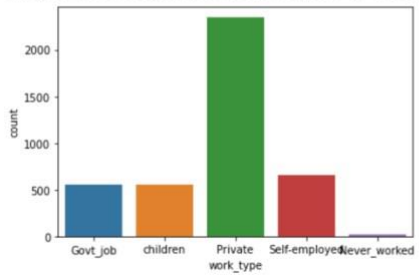


Figure 45 A count plot representing the work type carried out by the patients of the training set. This shows there are five work type categories in the training set: government job (approx. 550 patients), children (approx. 550 patients), private (approx. 2800 patients), self-employed (approx. 600 patients) and never worked (approx. 50 patients). This demonstrates that work type is a categorical data type. The count plot shows that the assumption of many patients working in the private sector is correct as this is the most popular category. The noticeable difference between the children and private categories can be accounted to the training set containing more data on older patients.

Based on background research of Bangladesh, it is assumed that most patients are working within the private sector. This is as research shows that 61.05% of Bangladesh's population is rural (Kameke, 2022). Therefore, it is likely that most of the population is working in rural areas, such as agriculture, which fall in the private sector.

Assumptions made on the children category can be misleading as there is no context as to what this category represents. Initially, it was assumed the children category represents patients in the training that have recently had children and so are out of work. After further consideration, the children category may represent the number of children in the training set.

As they are children, they are unlikely to have a work type. Yet, it can be argued that children patients would have been recorded within the never worked category. *9.2.2.7 Analysing 'Residence_type'*

```
1  sns.countplot(df_stroke_train["Residence_type"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e7d7825b0>
```

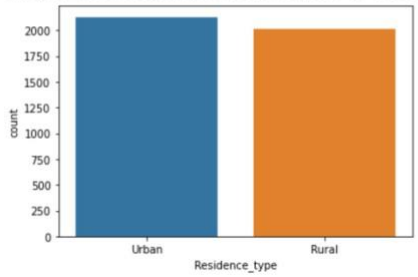


Figure 46 The residence types of the patients from the training set, which are: rural and urban. This demonstrates that residence type is a categorical data type. This shows that 2300 (approx.) patients live in an urban area and 2000 (approx.) patients live in a rural area.

Based on the research findings, it is assumed that most of the patients from the training set would reside in rural areas. However, this contradicts the findings in the count plot where a slightly higher number of patients reside in urban areas. This draws attention to where the dataset was collected. Research showed that the dataset was collected from clinics and hospitals across Bangladesh. But as a Bangladesh is a LMIC, it is worth thinking about which types of patients are more likely to have access to healthcare. A general assumption for a LMIC would be that wealthier patients would have better access to healthcare. Wealthier patients are also more likely to move from rural to urbans areas in search for better living and working conditions.

9.2.2.8 Analysing 'avg_glucose_level'

It is assumed that patients in the training set will have low average glucose levels to mirror findings of the analysis of the target variable (section 9.2.1).

```
1  sns.boxplot(df_stroke_train["avg_glucose_level"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa07caa8af0>
```

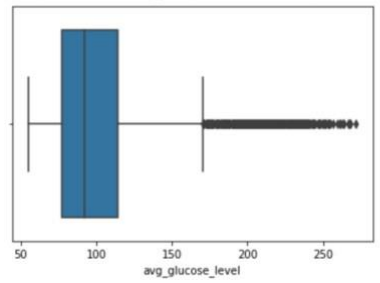


Figure 47 The avg_glucose_level attribute is a ratio data as the value is a measurement with an absolute zero and so can be represented with a box plot. The boxplot shows that the average glucose levels for patients in the training set range from 55 to 170. The lower quartile shows that's the average glucose level for 25% of patients is equal to or below 75, whilst the upper quartile shows that the average glucose level for 75% of patients is equal to or below 120. The median value for the average glucose levels of patients in the training set is 90. Furthermore, this boxplot also has a noteworthy number of outliers that are beyond the maximum value of 170. As a high glucose level is associated with the possibility of having a stroke, it can be assumed that glucose levels above the maximum value of 170 belong to those patients that have been diagnosed with having had a stroke. As analysis of the target variable has shown, the training set contains fewer patients that have been diagnosed with having had a stroke. Therefore, their average glucose levels are represented as outliers with the greater number of values being the boxplot.

```
1  sns.histplot(df_stroke_train["avg_glucose_level"], bins=10, kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa07ca074c0>
```

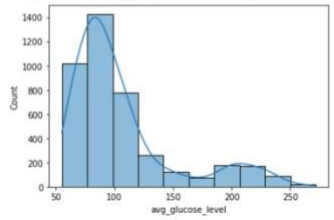


Figure 48 A histogram representing the range average glucose levels of the patients within the training set. Again, 10 bins have been created and the kde parameter is used to understand the shape of the data's distribution. This shows that the most common average glucose level for the patients in the training set is between 75 and 100. The kde shows that the data follows a negative skew. This echoes the pattern of the outliers in the boxplot. More patients have a lower average glucose level, reflecting the number of patients in the training set that have not had a stroke.

9.2.2.9 Analysing 'bmi'

Patients with a higher BMI are more likely to be diagnosed with a stroke. It is assumed only a small number of patients in the training set will have a high BMI to reflect the low number of patients that have had a stroke.

```
1  sns.boxplot(df_stroke_train["bmi"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa07c9808b0>
```

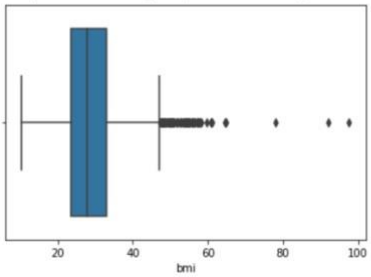


Figure 49 A box plot representing the bmi of the patients within the training set. The boxplot shows that the minimum value of a patient's BMI from the training set is just below 10 (approx.) and the maximum value is 45. The lower quartile shows that 25% of patients in the training set have a BMI equal to or below 23. Similarly, the upper quartile shows that 75% of patients in the training set have a BMI equal to or below the value of 35. The median value of the box plot is 27. As the box plot is quite narrow, it shows that many patients in the training set have a similar BMI. However, the box plot has numerous outliers that are beyond the maximum value of 45. As a higher BMI increases the likelihood of having a stroke, it is assumed that the outliers represent patients in the training set that have been diagnosed with having a stroke.

```
1  sns.histplot(df_stroke_train["bmi"], bins=10, kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa07c9594c0>
```

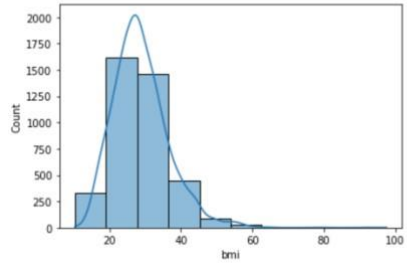


Figure 50 A histogram representing the bmi of the patients within the training set. The histogram has been created with 10 bins and the kde parameter. The bins show that the range in which most BMI values of the patients from the training set fall in is 19 to 27 (approx.). Again, the negative skew of the data replicates the pattern of the outliers in the box plot. As fewer patients in the training set have been diagnosed with having had a stroke, a smaller number of patients have a high BMI, hence the negative skew.

9.2.2.10 Analysing 'smoking_status'

```
1  sns.countplot(df_stroke_train["smoking_status"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e7d469760>
```

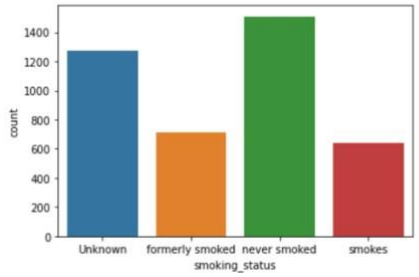


Figure 51 A count plot representing the smoking statuses of the patients from the training set. This shows four discrete categories exist within the training set with 1250 (approx.) patients where the smoking status is unknown, 700 (approx.) patients having formerly smoked, 1550 (approx.) have never smoked and 600 (approx.) currently smoke. This shows that the smoking status attribute is a categorical data type.

Research shows that tobacco use increases the likelihood of experiencing a stroke. Therefore, patients that have either formerly smoked or currently smoke are at a greater risk of experiencing a stroke. Both these categories, formerly smoked and smokes, are identical to one another. The never smoked category is clearly the largest category, containing most of the patients from the training set. This mirrors the target variable of stroke where many patients were not diagnosed as having had a stroke, which is common for this problem domain. The unknown category is a noteworthy category, but no context is provided as to what it represents. As this is a medical dataset, the unknown category may be of patients who did not want to disclose this information; equally, it could a category for when the smoking statuses of patients was not collected for various reasons.

9.2.3 Analysing the Relationships between Features

9.2.3.1 Pairwise Relationships

To understand relationships between the features of the training set, a bivariate analysis is done to compare features. To plot these pairwise relationships, Seaborn's Pairplot can be used.

```
1  sns.pairplot(df_stroke_train, hue='stroke', height=2)
```

Figure 52 The function used to plot a Pairplot using the Seaborn library. The parameters that are passed in are the training set data frame, the hue and the height.

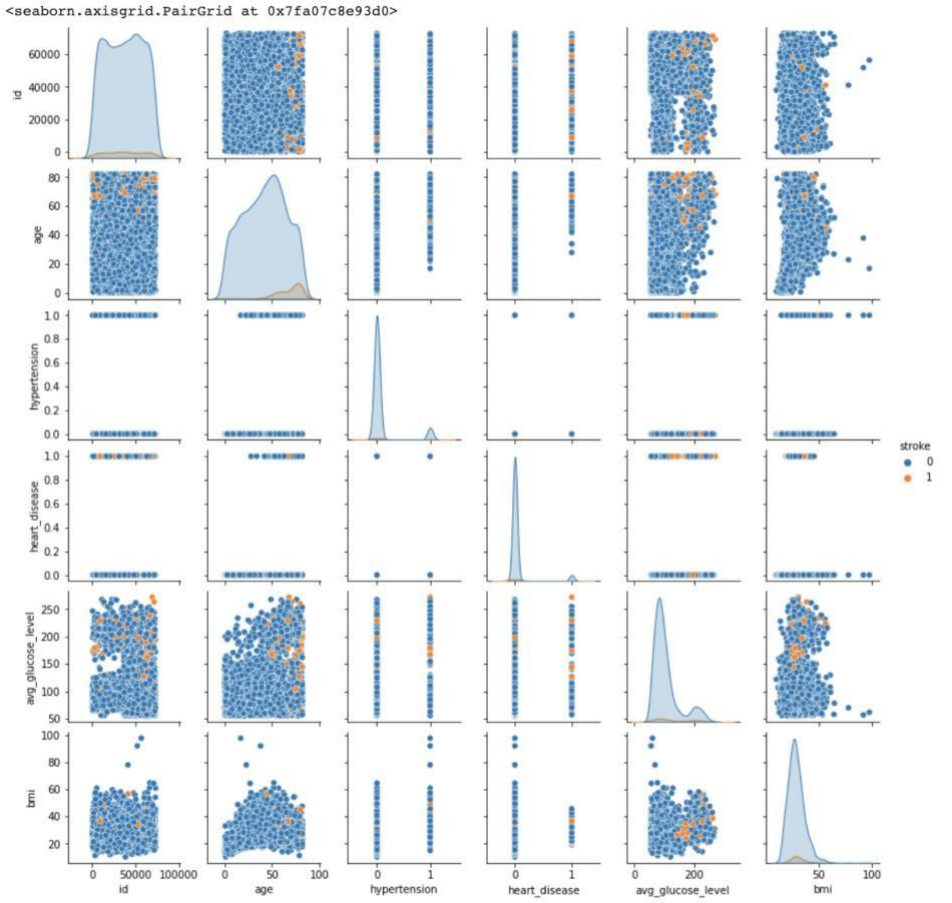


Figure 53 A Pairplot from the Seaborn library to understand the relationships between the features of the training set. Numerical features are plotted against the x-axis and the y-axis. Each plot shows how the data points are plotted when taking account into two features. The diagonal plots are histograms representing a univariate analysis to show how each feature is distributed.

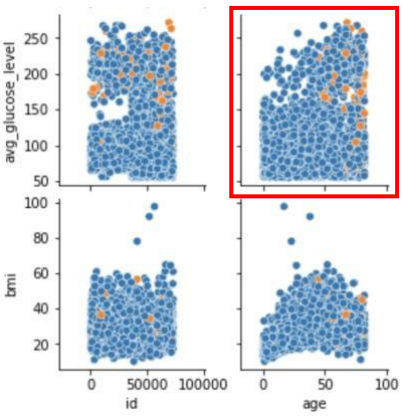


Figure 54 An extraction from the Pairplot specifically highlighting the plotting of data points against age and average glucose levels. This allows us to visually understand the relationship, if any, between the age of a patient and their average glucose levels. From this it can be interpreted that the patients that have been diagnosed with having a stroke from the training set are those towards the higher end of the age range and those with mid to high average glucose levels. This supports the findings of the univariate analysis on the age and average glucose level features.

9.2.3.2 Correlations

Correlation is the strength of the relationship between two features and is determined by an 'increasing or decreasing trend,' (Altman and Krzywinski, 2015).

As shown in Figure 55, the corr() method computes the correlations between the features of the training set. The correlation coefficient is measured from +1 to -1, with complete correlation being at either end. A positive correlation is denoted by +1, and similarly, a negative correlation

is denoted by -1. The closer the correlation coefficient is to either of these values indicates whether the correlation is positive or negative.

```
df_stroke_train_corr = df_stroke_train.corr(method='pearson')
```

Figure 55 Using the corr() method to compute correlations between the features of the training set.

Figure 56 shows the result of the corr() method. For example, the correlation coefficient between age and average glucose levels is 0.24. The value of 0.24 is closest to +1, rather than -1. This implies a positive correlation between the features of age and average glucose level, yet it may not be a strong correlation. This supports the background research and EDA findings.

```
df_stroke_train_corr
```

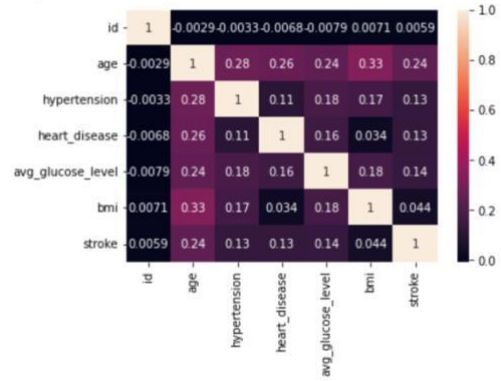
	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
id	1.000000	-0.002866	-0.003324	-0.006820	-0.007898	0.007094	0.005913
age	-0.002866	1.000000	0.275350	0.264527	0.241258	0.329668	0.244147
hypertension	-0.003324	0.275350	1.000000	0.110112	0.182588	0.172786	0.132676
heart_disease	-0.006820	0.264527	0.110112	1.000000	0.160357	0.034347	0.132379
avg_glucose_level	-0.007898	0.241258	0.182588	0.160357	1.000000	0.176462	0.135428
bmi	0.007094	0.329668	0.172786	0.034347	0.176462	1.000000	0.044284
stroke	0.005913	0.244147	0.132676	0.132379	0.135428	0.044284	1.000000

Figure 56 The correlation results returned from the corr() method which demonstrate the extent to which one feature correlates with another.

The correlation between features can be plotted in a heatmap (Figure 57), which uses colour to visually signify the strength of the relationships.

```
sns.heatmap(df_stroke_train_corr, annot=True)
plt.figure(figsize = (16,5))
```

```
<Figure size 1152x360 with 0 Axes>
```




```
<Figure size 1152x360 with 0 Axes>
```

Figure 57 A heatmap to demonstrate the correlations between each feature of the training set.

9.2.4 Analysing the Relationships between the Target and Feature Variables

Bivariate analysis involves analysing two features to understand the relationship between them and whether one feature can predict the other (Sandilands, 2014). This will be done by analysing the relationship between the target variable of stroke and each feature variable of the training set.

9.2.4.1 Analysing the Relationship between 'stroke' and 'gender'

As discussed in section 9.2.2.1, medical research shows that females are more likely to experience a stroke compared to males. From this it is assumed that patients in the training set that have had a stroke, more of these patients are likely to be female.

As stroke and gender are both categorical data types, a count plot can be plotted as in Figure 58.

```
1  sns.countplot(df_stroke_train["gender"], hue=df_stroke_train["stroke"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa077bb3ac0>
```

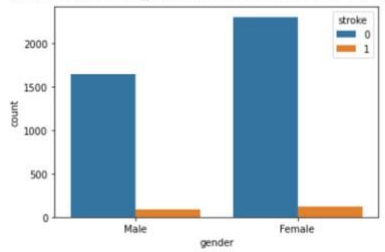


Figure 58 A count plot to demonstrate the number of male and female patients from the training set that have (1) and have not (0) been diagnosed with having a stroke.

Evidently, the number of patients overall that have not been diagnosed with stroke outweigh the patients that have. The number of females that have had a stroke is slightly greater than the number of males; this supports the medical findings.

9.2.4.2 Analysing the Relationship between 'stroke' and 'age'

As medical research shows that the likelihood of having a stroke increases with age, it assumed that the patients from the training set that have had a stroke will be of a higher age range.

Age is a ratio data type, hence a box plot (Figure 59) has been used to analyse the relationship between stroke and age.

```
1  sns.boxplot(data=df_stroke_train, x="stroke", y="age")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa077bb2880>
```

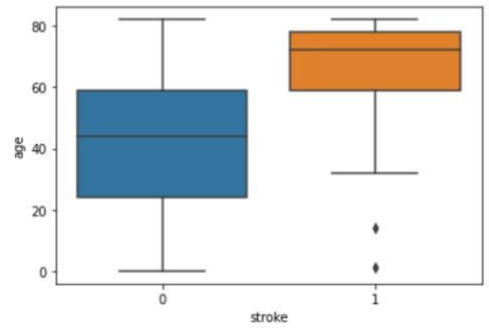


Figure 59 A box plot to show the range of ages, including outliers, for patients that have (1) and have not (0) been diagnosed with having a stroke from the training set.

The box plot shows that the ages of the patients from the training set that have not been diagnosed with a stroke range from 0 to 80. Comparable to this is the age ranges of the patients who have been diagnosed with a stroke, which range from 30 to 80. This supports medical research of the likelihood of having a stroke increasing with age. Additionally, the median value for the patients that have not been diagnosed with a stroke is 45 years. This is far greater for patients that have been diagnosed with a stroke where the median values lies at 74 years. The box plot does contain outliers for patients that have been diagnosed with having a stroke but at relatively young ages, which would have been either rare cases or misdiagnosis.

9.2.4.3 Analysing the Relationship between 'stroke' and 'hypertension'

Hypertension is proven to increase a patient's risk of experiencing a stroke and so it is assumed that patients in the training set that have had a stroke, will most probably also have hypertension.

Stroke and hypertension are both categorical data types and so a count plot has been used to compare their relationship (Figure 60).

```
1  sns.countplot(df_stroke_train["hypertension"], hue=df_stroke_train["stroke"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: P
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa077b0ed30>
```

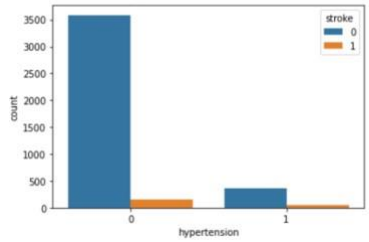


Figure 60 A count plot to demonstrate the number of patients with (1) or without (0) hypertension from the training set that have (1) or have not (0) been diagnosed with having a stroke.

The number of patients that have not had stroke or hypertension is relatively high, as expected. This is explained with the analysis of the target variable in section 9.2.1 and as patients without hypertension have a reduced risk of experiencing a stroke.

The count plot shows a small number of patients that don't have hypertension but have been diagnosed with having a stroke. This is understandable as hypertension is a contributing factor to increase a patient's risk of experiencing a stroke, but it is not the only factor. For these patients, other contributing factors have played a greater influence.

More importantly, the count plot shows a smaller number of patients that have had a stroke and have hypertension. This is proportional to the smaller number of patients in the training set that have been diagnosed with having a stroke.

9.2.4.4 Analysing the Relationship between 'stroke' and 'heart_disease'

Based on the medical research, it is assumed that patients in the training set with heart disease are also more likely to have a stroke.

As stroke and heart disease are both categorical data types, a count plot can be used to compare their relationship (Figure 61). The count plot shows a high number of patients that don't have stroke and don't have heart disease. This reflects the findings of analysis of the target variable (section 9.2.1).

```
1  sns.countplot(df_stroke_train["heart_disease"], hue=df_stroke_train["stroke"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pa
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa077ba31c0>
```

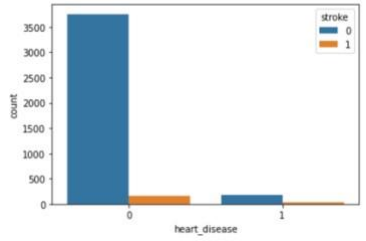


Figure 61 A count plot to demonstrate the number of patients with (1) or without (0) heart disease from the training set that have (1) or have not (0) been diagnosed with having a stroke.

The count plot shows a small number of patients in the training set that don't have heart disease but have had a stroke. The number of patients in this category is equivalent to the number of patients in Figure 60 that don't have hypertension but have had a stroke. Again, this shows that heart disease is another contributing factor, of many, to increase a patient's risk of experiencing a stroke. Comparably, the count plot shows a number of patients that have heart disease but have not had a stroke. Likewise, this demonstrates that heart disease is a single factor that can increase a patient's likelihood of experiencing a stroke, but it is not the determining factor.

Moreover, the count plot shows a small number of patients that do have heart disease and have been diagnosed with having a stroke. This is relational to the smaller number of patients in the training set that have been diagnosed with having a stroke, and the number of patients that have been diagnosed with hypertension.

9.2.4.5 Analysing the Relationship between 'stroke' and 'ever_married'

A patient's status is not a medical feature, but it can have a relationship with the likelihood of having a stroke. It is assumed that patients in the training set that have been married, are more likely to also have been diagnosed with having had a stroke. This assumption is based on the medical research that a patient's likelihood of experiencing a stroke increases as they age, and older patients are more likely to have been married at some point.

As stroke and the ever_married attribute are both categorical data types, the assumption can be validated with a count plot (Figure 62).

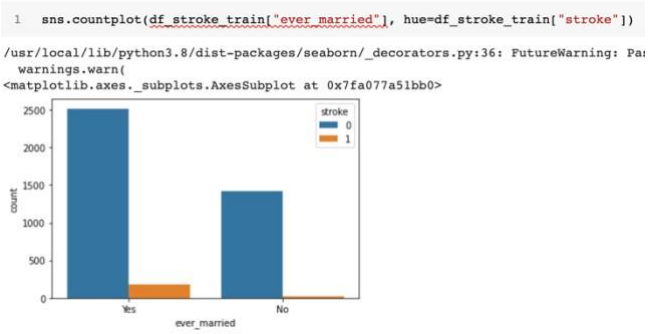


Figure 62 A count plot to demonstrate the number of patients that have (Yes) or have not (No) been married from the training set that have (1) or have not (0) been diagnosed with having a stroke.

The count plot shows that amongst the number of patients that have not had a stroke, a high number are married. However, a significant number of these patients are also not married. As a greater number of patients are married, even though they have not had a stroke, this raises the assumption that more patients within this training set are towards the higher age ranges. This is because older patients are more likely to have been married at some point in their lives.

A small number of patients in the training set have been married and have had a stroke. This finding aligns with the initial assumption that married patients more likely to have had a stroke due to age. Yet, there is a smaller number of patients that have not been married but have been diagnosed with having a stroke. As it is a very small number within this category, it may be representing the outliers that were identified in section 9.2.4.2.

9.2.4.6 Analysing the Relationship between 'stroke' and 'work_type'

Analysis of the work type attribute (section 9.2.2.6) supported the background research on Bangladesh, where the most common work type is the private sector. This is because 61.05% of Bangladesh's population is rural, and hence most people are involved in agricultural work, which falls within the private sector.

The relationship between stroke and work type can be visualised in a count plot as they are both categorical values; this is shown in Figure 63.

```
1  sns.countplot(df_stroke_train["work_type"], hue=df_stroke_train["stroke"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning:
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa077a22370>
```

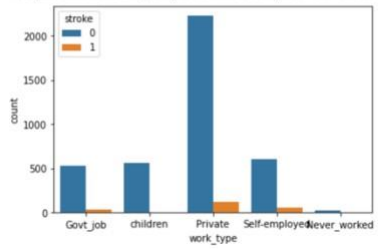


Figure 63 A count plot to demonstrate the number of patients within each work type attribute from the training set that have (1) or have not (0) been diagnosed with having a stroke.

The count plot shows that the private work type is the most common work type amongst patients that have and have not had a stroke. This supports the findings about Bangladesh's rural population.

There are patients from the training set that have not had a stroke in each work type category. This contrasts to the category of patients that have had a stroke, from which there are none in the category of children and never having worked. This shows that the patients in the training set are likely to be of working-class age. Having no patients that have been diagnosed having had a stroke in the children and never worked category can be hard to interpret. This is due to ambiguity on what these categories represents.

9.2.4.7 Analysing the Relationship between 'stroke' and 'Residence_type'

Research shows that most patients from the training set would reside in rural areas. It is assumed that patients residing in rural areas would have a more active lifestyle, than those in urban areas, due to there being less transport links and a closer relation to nature. Therefore, patients in rural areas can be assumed to be at a lower risk of experiencing a stroke. It is important to consider the lifestyle of the patients living in rural and urban areas. Patients living and working in urban areas likely have more knowledge on medical diseases and better access to healthcare. This makes it questionable if patients living in rural areas are less likely to experience a stroke, or they are simply less likely to be diagnosed of their medical conditions.

As stroke and residence type are categorical data types, they can be plotted using a count plot as shown in Figure 64.

```
1  sns.countplot(df_stroke_train["Residence_type"], hue=df_stroke_train["stroke"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pas
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fa07798d4f0>
```

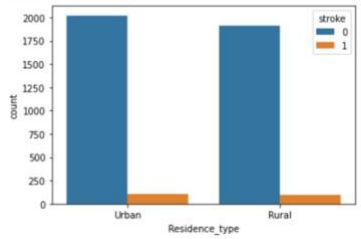


Figure 64 A count plot to demonstrate the number of patients that reside in urban or rural areas from the training set that have (1) or have not (0) been diagnosed with having a stroke.

The count plot shows a similar number of patients that have not had a stroke reside in both rural and urban areas. The number of patients that have had a stroke residing is rural and

urban areas is also similar. The little variation makes it difficult to confirm our assumption and draw a relationship, if any, between the features of stroke and residence type.

9.2.4.8 Analysing the Relationship between 'stroke' and 'avg_glucose_level'

Analysis of the target variable in section 9.2.1 found that in the training set, there is a greater number of patients that have not been diagnosed with having a stroke compared to those who have. Therefore, it can be assumed that the average glucose level for the patients in the training set will also be low to mirror the number of patients having been diagnosed as having a stroke.

The average glucose level feature is a ratio data type; hence, a boxplot can be used to compare the relationship between average glucose levels and stroke. This is shown in Figure 65.

```
1  sns.boxplot(data=df_stroke_train, x="stroke", y="avg_glucose_level")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0779e32e0>
```

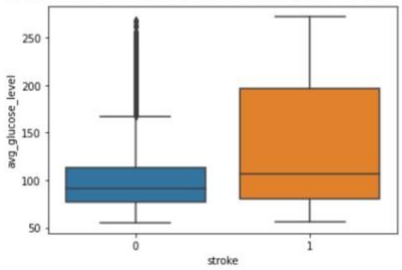


Figure 65 A box plot to show the range of average glucose levels, including outliers, for patients that have (1) and have not (0) been diagnosed with having a stroke from the training set.

For patients that have not had a stroke, the average glucose levels range from a minimum value of 55 to a maximum value of 175. This is considerably a small range compared to the range of average glucose levels for the patients that have had a stroke which range from a minimum value of 60 to a maximum value of 280. This aligns with the assumption that the average glucose levels for patients that haven't had a stroke will be low.

9.2.4.9 Analysing the Relationship between 'stroke' and 'bmi'

Based on medical research, it is assumed that the BMI values for patients that have had a stroke will be higher than patients that have not. To validate this assumption, a box plot can be used as BMI is of a ratio data type (Figure 66).

```
1  sns.boxplot(data=df_stroke_train, x="stroke", y="bmi")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa077c6b2e0>
```

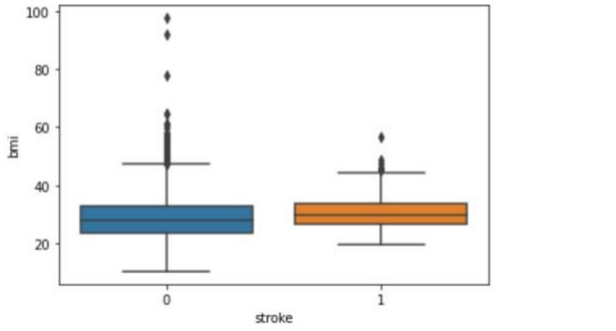


Figure 66 A box plot to show the range of BMI values, including outliers, for patients that have (1) and have not (0) been diagnosed with having a stroke from the training set.

From the box plot, for patients that have not had a stroke, the minimum BMI value is 5 and the maximum value is 45. However, the box plot for these patients has a significant number of

outliers beyond the maximum value. This shows several patients do have a BMI higher than 45 but these patients have not had a stroke.

For patients that have had a stroke, the minimum BMI value is 20 and the maximum value is 43. This box plot has considerably less outliers. It can be interpreted that patients with a mid to high BMI will have had a stroke.

9.2.4.10 Analysing the Relationship between 'stroke' and 'smoking_status' Research shows that tobacco use increases a patient's likelihood of experiencing a stroke. Therefore, it is assumed that patients that currently smoke or have formerly smoked are at a greater likelihood of having a stroke.

As stroke and smoking status are categorical data types, a count plot can be used to show their relation (Figure 67).

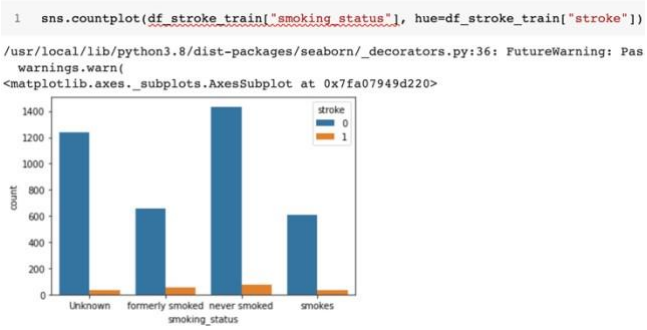


Figure 67 A count plot to demonstrate the number of patients within each smoking status category from the training set that have (1) or have not (0) been diagnosed with having a stroke.

The count plot shows that the highest number of patients that have not had a stroke are in the categories of never smoked and unknown. As in the count plot, and supported by medical research, patients that do not smoke have reduced risks of having a stroke. Furthermore, no context has been provided on what the unknown category represents. But, with consideration to the possibility of children being included in this training set (as discussed in section 9.2.2.6), the unknown category may represent children patients. This is because it is unlikely that young children will be questioned on tobacco use, making it an unknown piece of data until a later date.

When analysing the patients that have had a stroke, a number of patients fall within each smoking status category. The category with the greatest number of patients that have had a stroke is the never smoked category. This is unclear as it does not reflect the medical research. The next largest category is of formerly smoked; this does reflect the medical research. The penultimate category is the smokes category, representing patients that currently smoke. This is not the largest category, but it is unknown how long each patient has been smoking for. Therefore, even though these patients have not had a stroke yet, they have increased their risk for the future.

9.3 Appendix C – KNN

9.3.1 How the KNN Algorithm Works

The KNN algorithm is defined as a 'lazy learner' as all computation does not occur until a prediction needs to be made. Additionally, KNN is an 'instance-based' algorithm, as each prediction is its own instance, as the distance between each existing data point and new data point must be computed individually (Raschka, 2018).

The KNN algorithm assumes that 'similar points can be found near one another,' (IBM, 2022). The algorithm does this by plotting all data points and conducting a majority vote. The majority vote considers the classes of the data points surrounding a new data point and assigns the class label that appears more frequently.

The data points neighbours, which are considered for the majority vote, are defined by *k*. *k* is a hyperparameter for which there is no optimal value but can be tuned to improve a model's performance. To define *k,* the size of the dataset is considered; a value too high will lead to low variance yet a high bias, but a value too small will lead to a higher variance but a low bias.

Furthermore, to decide which *k* neighbours are closest to a new data point, a distance metric is defined. Again, a distance metric is chosen based on the dataset. The most common distance metric is the Euclidean distance which measures a true straight line between the points. Other distance metrics include Hamming, Manhattan, and Minkowski.

9.3.2 Benefits and Limitations of the Chosen Algorithm

A limitation of the KNN algorithm is that it doesn't perform as well for a higher number of dimensions. This is known as the curse of dimensionality. The curse of dimensionality occurs as the KNN algorithm individually calculates the distance for *k* neighbours in each dimension. This can be computationally expensive and time consuming for a large dataset. Nevertheless, techniques such as dimensionality reduction can overcome this limitation.

Furthermore, KNN is an instance-based algorithm. As the distance between data points must be computed individually, this is time consuming for a large dataset. This is another limitation of the KNN algorithm as it makes the algorithm less desirable to use, especially when predictions need to be made urgently.

Another limitation of the KNN algorithm is that it is a lazy learner and hence all computations are not made until a prediction is required. A benefit of this is that there is no separate training stage. This makes the algorithm faster and easier to add new data to. However, this requires a large amount of memory as the training data and testing data are stored at the same time.

Despite these limitations, the KNN algorithm is easy to implement as it only has a few hyperparameters. This makes it easy experiment with the algorithm to tune the hyperparameters (IBM, 2022). Additionally, the KNN algorithm can be tailored for classification or regression machine learning tasks, making it adaptable to use and adopt.

9.4 Appendix D – Evaluation Metrics

9.4.1 Accuracy

Accuracy is the 'correctness' of a model, as it measures the correct predictions a model makes for either class, (Afonja, 2017). For this classification problem, the accuracy will be denoted by the model correctly predicting whether a patient has had a stroke.

Naively, accuracy may be seen as the most critical evaluation metric. But the accuracy paradox must be considered. This is because metrics, such as accuracy, can be affected by an imbalance within the dataset. As seen in section 9.2.1, where the target variable of the training set is explored, there is an imbalance between the positive and negative classes. A class imbalance leads to the model more correctly predicting the majority class as there is more of this data, resulting in the accuracy paradox. As stated by Afonja (2017), a model with a certain accuracy may have 'greater predictive power than models with higher accuracy.' This means a model with a moderate accuracy score may have higher values in other metrics, such as precision or recall, demonstrating the importance of evaluating metrics beyond accuracy.

Despite the accuracy paradox, the research team still chose the accuracy metric as it will assess the correctness of the models, i.e. how often the models correctly predict either class. As a class imbalance has been identified, other metrics will also be used to possibly demonstrate the accuracy paradox in play.

9.4.2 Confusion Matrix

A confusion matrix summarises the results of a classification algorithm by the number of correct and incorrect predictions made (Brownlee, 2016c). Unlike accuracy, a confusion matrix is not affected by class imbalance. Instead, it shows which classes the model was or was not able to correctly predict. The meanings of the values represented by a confusion matrix are outlined in Table 5.

The research team decided to use a confusion matrix to understand how the models are performing and where errors are occurring. By understanding the confusion matrix, it can be assessed whether the accuracy paradox has affected the models. Figure 68 shows an example confusion matrix, and Figure 69 represents the confusion matrix as a heatmap.

```
array([[23,  5],
       [ 3, 30]])
```

Figure 68 An example confusion matrix to show how it would be displayed (T, 2019).

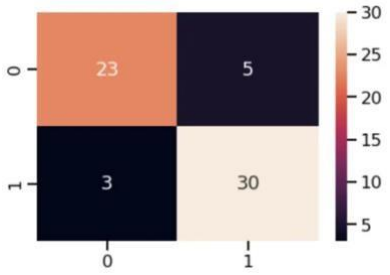


Figure 69 An example heatmap to show how the confusion matrix can be represented in a colour coded matrix (T, 2019).

9.4.3 Precision

Precision is the proportion of retrieved items that are relevant (Kozyrkov, 2022). Precision is calculated as:

Precision = True Positives / (True Positives + False Positives)

A precision value of 0.0 represents no precision, and likewise, a value of 1.0 represents a perfect precision. A model with a precision value very close to 1.0 represents a very precise model where all predictions that have been classed as positive are likely to be correct (Korstanje, 2021).

For this machine learning task, precision will be the proportion of patients that have actually had a stroke from all the patients the model said to have been diagnosed.

9.4.4 Recall

Recall is the proportion of relevant items from everything that is retrieved (Kozyrkov, 2022). Recall is calculated as:

Recall = True Positives / (True Positives + False Negatives)

Like precision, a recall value of 0.0 represents no recall, and a value of 1.0 represents a perfect recall. A model with a recall value very close to 1.0 demonstrates that model was able to predict all positive classes, even though some negative classes may have been identified as positive (Korstanje, 2021).

For this machine learning task, recall will be the proportion from all patients that actually have had a stroke, how many of these were retrieved correctly. Therefore, recall indicates the positive class predictions the model missed on.

9.4.5 F1 Score

The F1 score is the harmonic mean of precision and recall, and can be calculated as:

*F1 Score = (2 * Precision * Recall) / (Precision + Recall)*

As with precision and recall, the calculated F1 score will be between 0.0 and 1.0, for a low score or a perfect score, respectively.

The harmonic mean is an average when numbers represent a ratio in information retrieval (Wood, 2022). For a F1 score, an equal weight is given to both precision and recall.
Therefore, if both, precision and recall values are low, the F1 score will also be low. The F1 score allows precision and recall to be combined into a single metric. This is particularly useful for imbalanced datasets where it is difficult to balance the values of precision and recall.

The F1 score combines precision and recall into a single metric, which is useful for imbalanced datasets where it is difficult to balance the values.

9.5 Appendix E - Validation Set Evaluation

9.5.1 Accuracy Result of Unseen, Validation Set

The accuracy of the model's performance on the unseen, validation set is retrieved by using Scikit Learn's accuracy_score function. This is shown in Figure 70. The function takes in two parameters: df_stroke_validate_Y which is the ground truth labels for what is being predicted, and stroke_Y_validate_knn which is the predictions made by the KNN classifier. The accuracy score retrieved is stored as a variable and multiplied by 100 to represent it as a percentage.

```
1  stroke_acc_validate_knn = metrics.accuracy_score(df_stroke_validate_Y, stroke_Y_validate_knn)
2  accuracy_score = stroke_acc_validate_knn * 100
```

Figure 70 Calculating the accuracy score for the unseen, validation data by using the accuracy_score from Scikit Learn's metrics library.

```
1  print(accuracy_score)
```

```
95.65217391304348
```

Figure 71 Using Python's print statement to return the accuracy score calculated.

9.5.2 Confusion Matrix of Unseen, Validation Set

The confusion matrix for the model's performance on the unseen, validation set can be retrieved from Scikit Learn's metric library as shown in Figure 72. This function takes in two parameters: df_stroke_validate_Y which is the ground truth labels for what is being predicted, and stroke_Y_validate_knn which is the predictions made by the KNN classifier.

```
1  stroke_cm_validate_knn = metrics.confusion_matrix(df_stroke_validate_Y, stroke_Y_validate_knn)
```

Figure 72 Calculating the confusion matrix for the unseen, validation data by using the confusion_matrix from Scikit Learn's metrics library.

```
2  print(stroke_cm_validate_knn)
```

```
[[440   0]
 [ 20   0]]
```

Figure 73 Using Python's print statement to return the confusion matrix.

9.5.3 Precision of Unseen, Validation Set

The precision value of the unseen, validation set can be calculated using the values from the confusion matrix as such:

Precision = True Positives / (True Positives + False Positives)

Precision = 0 / (0 + 0) = 0

9.5.4 Recall of Unseen, Validation Set

The recall value of the unseen, validation set can be calculated using the values from the confusion matrix as such:

Recall = True Positives / (True Positives + False Negatives)

Recall = 0 / (0 + 20) = 0

9.5.5 F1 Score of Unseen, Validation Set

The F1 score, the harmonic mean of precision and recall, can be calculated as:

*F1 Score = (2 * Precision * Recall) / (Precision + Recall)*

*F1 Score = (2 * 0 * 0) / (0 + 0) = 0*

9.6 Appendix F - Further Modelling Improvements

9.6.1 Stratified Data Splitting

A problem that has been identified throughout section 4.2, and is evident in each team member's evaluation results on the unseen data is the presence of class imbalance. The presence of class imbalance in both the training and validation stages has affected the model's performance. This is because as there is a majority class present, the model is more skewed towards making predictions for this class.

Removing class imbalance will allow us to truly evaluate our models' performances on seen and unseen data. To overcome class imbalance, the way the dataset is split can be changed. The initial method of data splitting used in section 2.2.1 is a random split. As explained by Baheti (2023), a random split is when records are randomly assigned to the train, validation and test sets based on the test size defined. This works well for a balanced dataset but can cause a significant bias in an imbalanced dataset.

For an imbalanced dataset, stratified data splitting is a better approach. In stratified data splitting, the proportion of overall records that belong to each class is maintained in each subset. This reduces class imbalance as each subset of data will contain a similar proportion of each class (Baheti, 2023). Even though random splitting is meant to be representative of the whole dataset, stratified sampling ensures an equal proportion of each class in present in each subset.

9.6.1.1 Implementation

Before applying the stratified data split, it is worth visualising the class imbalance present in the whole dataset. The count plot shows that there are considerably more patients in the negative class of not having had a stroke compared to those in the positive class that have had a stroke. The exact numbers can be retrieved using Pandas' value_counts() method as shown in Figure 75, and the proportion these values are equivalent to is shown in Figure 76.

```
sns.countplot(df_stroke['stroke'])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f5488cd4610>
```

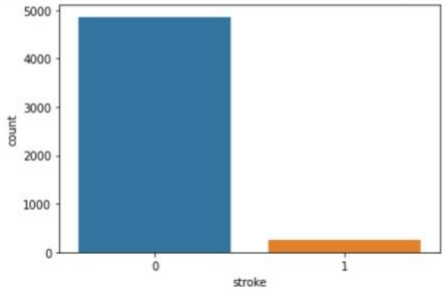


Figure 74 A count plot from the Seaborn library to visualise the number of patients within each class from the whole dataset.

```
df_stroke['stroke'].value_counts()
```

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

Figure 75 Using Pandas' value_counts() method to retrieve the exact number of patients within each class. This shows us that 4861 patients from the whole dataset belong to the negative class and 249 patients belong to the positive class.

```
df_stroke['stroke'].value_counts(normalize=True)*100
```

```
0    95.127202
1     4.872798
Name: stroke, dtype: float64
```

Figure 76 Using Pandas' value_counts method with the parameter of normalize to understand what proportion of patients from the whole dataset belong to each class. This shows us that 95% of the patients belong to the negative class of not having been diagnosed a with a stroke, and 5% of patients belong to the positive class of having been diagnosed with having had a stroke.

Then, the dataset can be split into subsets. As done previously, the dataset is split into X and Y, the feature and target variables, respectively.

```
X = df_stroke.drop(labels=["stroke"], axis='columns')
Y = df_stroke["stroke"]
```

Figure 77 Defining X, the feature variables, and Y, the target variable. X is defined by assigning it to the training set without the target variable which is dropped. Y is defined by assigning it as the target variable attribute only.

Then, as done in section 2.2.1, the dataset can be split into intermediate and testing subsets as shown in Figure 78. The intermediate set is then split further to create training and validation subsets a shown in Figure 79. The only difference in this data splitting to the initial method, is that an additional parameter of 'stratify' has been used. In both, the parameter of stratify is passed in the target variable through Y and df_stroke_intermediate_Y. This tells the train_test_split to stratify how the data is split based on the target variable.

```
#using the train_test_split() method to split X and Y into four further subsets of data
#test_size is set to 0.10, to create a split proportional to 90:10
df_stroke_intermediate_X, df_stroke_test_X, df_stroke_intermediate_Y, df_stroke_test_Y = \
  model_selection.train_test_split(X, Y, test_size=0.10, random_state=42, stratify=Y)
```

Figure 78 Using Scikit-Learn's model_selection module, and train_test_split() method, to perform a stratified split on X and Y into intermediate and testing subsets at a 90:10 ratio, with a random state of 42.

```
#using the train_test_split() method to split the intermediate subsets into four further subsets of data
#test_size is set to 0.10, to create a split proportional to 90:10
df_stroke_train_X, df_stroke_validate_X, df_stroke_train_Y, df_stroke_validate_Y = \
  model_selection.train_test_split(df_stroke_intermediate_X, df_stroke_intermediate_Y, test_size=0.10, random_state=42, stratify=df_stroke_intermediate_Y)
```

Figure 79 Using Scikit-Learn's model_selection module, and train_test_split() method, to perform a stratified split on the intermediate set into training and validation subsets at a 80:10 ratio, with a random state of 42.

The number of patient records from each class that have been assigned to each subset, and more importantly the proportion of each, can be checked to see how the data has been distributed.

```
1  df_stroke_train_Y.value_counts()
```

```
0    3937
1     202
Name: stroke, dtype: int64
```

```
1  df_stroke_train_Y.value_counts(normalize=True)*100
```

```
0    95.119594
1     4.880406
Name: stroke, dtype: float64
```

Figure 80 Using Pandas' value_counts() method to count the number of values in the training subset that belong to each class. Then using the value_counts() method again, but with the normalize parameter, to return the proportion of each class in the training subset. This shows that the training subset has 95% records that belong to the negative class where patients have not been diagnosed with having had a stroke, and 5% belong to the positive class where patients have been diagnosed with having had a stroke.

```
1  df_stroke_validate_Y.value_counts()
```

```
0    438
1     22
Name: stroke, dtype: int64
```

```
1  df_stroke_validate_Y.value_counts(normalize=True)*100
```

```
0    95.217391
1     4.782609
Name: stroke, dtype: float64
```

Figure 81 Using Pandas' value_counts() method to count the number of values in the validation subset that belong to each class. Then using the value_counts() method again, but with the normalize parameter, to return the proportion of each class in the validation subset. This shows that the validation subset has 95% records that belong to the negative class where patients have not been diagnosed with having had a stroke, and 5% belong to the positive class where patients have been diagnosed with having had a stroke.

```
1  df_stroke_test_Y.value_counts()
```

```
0    486
1     25
Name: stroke, dtype: int64
```

```
1  df_stroke_test_Y.value_counts(normalize=True)*100
```

```
0    95.107632
1     4.892368
Name: stroke, dtype: float64
```

Figure 82 Using Pandas' value_counts() method to count the number of values in the testing subset that belong to each class. Then using the value_counts() method again, but with the normalize parameter, to return the proportion of each class in the testing subset. This shows that the testing subset has 95% records that belong to the negative class where patients have not been diagnosed with having had a stroke, and 5% belong to the positive class where patients have been diagnosed with having had a stroke.

As seen in Figure 80, Figure 81, Figure 82, the number of records from the negative and positive class are identical across the training, validation and testing subsets at 95% and 5%, respectively. This demonstrates that the dataset has been successfully split using a stratified split as the proportions of each class in each subset are balanced.

After the dataset has been split, EDA on the training set, as done in section 2.3, can be performed. When this EDA was performed on the training set, a certain visualisation stood out that had not occurred in the initial iteration of data splitting. This was the count plot of the genders present in the training set as shown in Figure 83. This count plot shows that the training set contains the genders of male and female, and even though the gender of Other is present in the dataset, the training set contains no occurrences of it.

```
1  sns.countplot(df_stroke_train["gender"])
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7ff6f7872460>
```

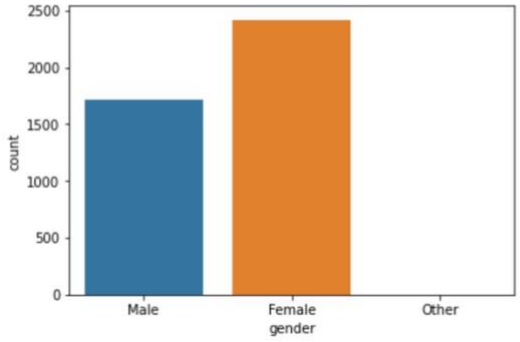


Figure 83 A count plot of the genders in the training set. This shows that the training set has the genders of male and female present. The Other gender does exist but there are no occurrences of this in the training set.

Based on this, it can be considered whether it may be worth dropping the attribute of other. As there are no occurrences of the other attribute, the model will not be trained on this attribute anyways and hence it will be treated as an outlier. Furthermore, in a medical setting, where this machine learning model would be applied, medical professionals will be more concerned with the biological gender of patients at birth. This is as when providing medical interventions, the biological gender of a patient will be more important than their preferred pronoun as certain medical interventions must be adapted for males and females. Therefore, it was decided to drop this as an attribute.

After the EDA, the initial pre-processing techniques were performed on the training set as in section 3.4, where the other attribute was also dropped. Next, the model was trained and evaluated on this transformed, training set. The same pre-processing techniques were then applied on the validation set. Finally, the model was tested on this transformed validation set and again evaluated using the same evaluation metrics as in section 5.1.

The evaluation results gained at this stage have been compared to the evaluation results of section 5.1, which were obtained using a random data split rather than a stratified data split. Table 11 shows this comparison. Both these evaluation results are gained from testing the model on the unseen, validation sets.

Table 11 A comparison of the results retrieved from testing the model on unseen, validation sets when using a random data split versus a stratified data split.

Evaluation Metric	Random Data Split	Stratified Data Split
Accuracy	96%	95%
Confusion Matrix	[[440 0] [20 0]]	[[438 0] [22 0]]
Precision	0	0
Recall	0	0
F1 Score	0	0

As Table 11 demonstrates, the improved model performed slightly lower in accuracy, with equivalent values in precision, recall and F1 score. This shows even though each subset has an equivalent proportion of each class, there is still a class imbalance in the dataset resulting in an exceedingly high accuracy on unseen data. Therefore, it is worth considering how else the class imbalance could be minimised.

9.6.2 SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is another technique to remove class balance within a dataset. SMOTE involves creating synthetic examples for the minority class, which is an over-sampling approach (Chawla *et al.*, 2002).

9.6.2.1 Implementation

The implementation of SMOTE involves using the techniques from the initial iteration and the stratified data split from section 9.6.1, with the addition of the following.

After the pre-processing has been performed on the training set, SMOTE can be performed.

```
9  from imblearn.over_sampling import SMOTE
```

```
1  pip install imbalanced-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.8/dist-packages (0.8.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.8/dist-packages (from imbalanced-learn) (1.7.3)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.8/dist-packages (from imbalanced-learn) (1.21.6)
Requirement already satisfied: scikit-learn>=0.24 in /usr/local/lib/python3.8/dist-packages (from imbalanced-learn) (1.0.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.24->imbalanced-learn) (3.1.0)
```

Figure 84 Installing the libraries and packages required for SMOTE.

After the required libraries are installed, a SMOTE instance must be defined as shown in Figure 85**Error! Reference source not found.**. This takes the parameters of a random state which controls how the data is shuffled, and a sampling strategy which defines the ratio of samples of the minority class over the majority class.

```
1  smote = SMOTE(random_state=23, sampling_strategy=1.0)
```

Figure 85 Creating a SMOTE instance which takes the parameters of a random state and a sampling strategy.

Then, the SMOTE instance can be fitted onto the training subsets to create resampled training subsets (Figure 86).

```
1  df_stroke_train_X, df_stroke_train_Y = smote.fit_resample(df_stroke_train_X, df_stroke_train_Y)
```

Figure 86 Fitting the SMOTE instance onto the training subsets to create resampled training subsets.

```
1  df_stroke_train = deepcopy(df_stroke_train_X)
2  df_stroke_train["stroke"] = df_stroke_train_Y
```

Figure 87 Using Python's deep copy function to create new objects.

The model can then be trained using the resampled training subsets. Then, the chosen preprocessing techniques are used on the validation set which is then tested and evaluated. Testing the model is done as in the initial iteration with the addition of using the SMOTE instance to fit and resample the validation subsets.

```
1  X = df_stroke_validate_X
2  Y = df_stroke_validate_Y
```

```
1  X, Y = smote.fit_resample(X, Y)
```

Figure 88 Using the SMOTE instance to fit and resample the validation subsets before they are used to make the prediction.

Finally, the performance of the model on the unseen, validation set can be evaluated. The evaluation results gained have been compared to the evaluation results on the validation set of section 5.1 (initial iteration) and section 9.6; Table 12 shows this comparison.

Table 12 A comparison of the results retrieved from testing the model on unseen, validation sets when using a random data split versus a stratified data split.

Evaluation Metric	Initial Iteration (Random data split)	Stratified Data Split	SMOTE
Accuracy	96%	95%	81%
Confusion Matrix	[[440 0] [20 0]]	[[438 0] [22 0]]	[[304 134] [29 409]]
Precision	0	0	0.75
Recall	0	0	0.93
F1 Score	0	0	0.83

When comparing the improved model with SMOTE, it evidently has a lower accuracy than the previous two models. However, this is reasonable model performance on unseen data as it still surpasses the baseline score. Furthermore, comparing the three models' accuracy confirms that SMOTE has removed class imbalance within the dataset. This is also further confirmed with the confusion matrix.

As the initial dataset had a class imbalance towards the negative class, it is worth noting how often the model correctly predicted each class. The number of true negatives show that the model correctly predicted the negative class for 304 predictions, i.e. predicting a patient has not had a stroke and they have not. Similarly, the model correctly predicted the positive class for 409 predictions, i.e. predicting a patient has had a stroke and they have. This balance in correct predictions, for the positive and negative class, show that the model is not leaning towards a particular class as there is no longer a majority class.

This leads to the precision and recall results; the model achieved a good precision score and a better recall score. As this is a medical dataset, the recall value is of slightly higher importance. This is as recall is the proportion from all patients that have had a stroke, how many of these were retrieved by the model. This allows us to pick on how many positive class predictions the model missed.

When comparing the evaluation metrics retrieved for each model, the macro and weighted averages on the classification reports stood out. These are shown in Figure 89, Figure 90 and Figure 91.

```
1  print("KNN Classification Report: \n%s" % metrics.classification_report(df_stroke_validate_Y, stroke_Y_validate_knn))
```

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

           0       0.96      1.00      0.98       440
           1       0.00      0.00      0.00        20

    accuracy                           0.96       460
   macro avg       0.48      0.50      0.49       460
weighted avg       0.91      0.96      0.94       460
```

Figure 89 KNN Classification report for the initial model's performance on the unseen, validation data set.

```
1  print("KNN Classification Report: \n%s" % metrics.classification_report(df_stroke_validate_Y, stroke_Y_validate_knn))
```

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

           0       0.95      1.00      0.98       438
           1       0.00      0.00      0.00        22

    accuracy                           0.95       460
   macro avg       0.48      0.50      0.49       460
weighted avg       0.91      0.95      0.93       460
```

Figure 90 KNN Classification report for the first improved model's (with stratified split) performance on the unseen, validation data set.

```
1  print("KNN Classification Report: \n%s" % metrics.classification_report(Y, stroke_Y_validate_knn))
```

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

           0       0.91      0.69      0.79       438
           1       0.75      0.93      0.83       438

    accuracy                           0.81       876
   macro avg       0.83      0.81      0.81       876
weighted avg       0.83      0.81      0.81       876
```

Figure 91 KNN Classification report for the second improved model's (with stratified split and SMOTE) performance on the unseen, validation data set.

As stated by Leung (2022), the macro average is an 'unweighted mean,' and the weighted average is the mean of each class's F1 scores considering their support. Support is the 'number of actual occurrences of the class in the dataset, (Leung, 2022). But both, macro average and weighted average, are highly influenced by the F1 score. Based on the dataset, it can be decided which metric is a more important measure. Leung (2022), explains that the weighted average is preferred for imbalanced datasets as this allows a 'greater contribution' to be assigned to the majority class. Therefore, I will focus on the weighted averages of the F1 score which have decreased from 0.95 (initial) to 0.93 (second) to 0.81 (final). This demonstrates how the class imbalance of the majority class has decreased with each iteration.