BIRMINGHAM CITY University

Predictive Modelling of Stroke using Decision Tree Classifier
Individual Report
Helly Thakar

Student ID Number: 19155625

AI and Machine Learning Project 2022-23: CMP6202

BSc (Hons) Computer and Data Science

Birmingham City University

Faculty of Computing Engineering & the Built Environment School of Computing & Digital Technology

Table of Contents

1 Report Introduction 2

1.1 Dataset identification 3

1.2 Supervised learning task identification 4

2 Exploratory Data Analysis 4

2.1 Questions identification 5

2.2 Splitting the dataset 6

2.3 Exploratory Data Analysis process and results 8

2.4 EDA conclusions 24

3 Experimental Design 24

3.1 Identification of your chosen supervised learning algorithm(s) 25

3.2 Identification of appropriate evaluation techniques 25

3.3 Data cleaning and Pre-processing transformations 26

3.4 Limitations and Options 30

4 Predictive Modelling / Model Development 31

4.1 The predictive modelling process 31

4.2 Evaluation results on “seen” data 32

5 Evaluation and further modelling improvements 34

5.1 Initial evaluation comparison 34

6 Conclusion 38

6.2 Suggested further improvements to the model development process 38

6.3 Reflection on Research Team 39

6.4 Reflection on Individual Learning 39

7 References and Bibliography 39

1 Report Introduction

A non-communicable illness called stroke is to blame for about 11% of all fatalities. Machine learning can be used to forecast the onset of a stroke thanks to advancements in medical technology. The algorithms used in machine learning are beneficial in supplying accurate analysis and producing accurate predictions. The majority of the prior efforts on stroke concern heart stroke prediction [1]. The research on brain stroke is rather limited. This report uses machine learning to forecast the likelihood of a brain stroke. The classification algorithm utilised is crucial to the methods adopted and outcomes realised. This model has a drawback because it was trained on tabular data rather than actual brain imagery through which we can understand the relationships between specific areas of the brain and how they are functioning and locate the areas of the brain that are affected by neurological disorders, as we can see here stroke. The decision tree machine learning classification technique is demonstrated in the study. It is possible to evaluate a few more machine learning algorithms using the findings of this research.

To continue with this challenge, a dataset from Kaggle is picked that has a variety of physiological characteristics as its properties. These characteristics are analysed subsequently and utilised in the final forecast. Then supervised learning task is identified and further data split is done into train, test and validate and then Exploratory Data Analysis is done on the training set and further the report specifies about the chosen supervised learning algorithm and then it talks about the evaluation techniques used. Afterwards, data cleaning and pre-processing is done to get more accurate result and then predictive modelling process is explained with results on train data(seen data).

Further, it explains the evaluation results on the validate data (initially seen data) of the model and comparison with the other team member's results. Following it explains the improvements attempted on model further and then it discusses about the final model evaluation results.

At end the conclusion summarises existing results from the model development process and provides further enhancement recommendations and finally reflection on research team and individual learning.

1.1 Dataset identification

Stroke, which occurs when an artery becomes clogged or a blood vessel breaks, causes the blood flow to a part of the brain to stop, is a potentially deadly medical condition (haemorrhagic stroke). This harm may affect not just how your body functions, but also how you think, feel, and communicate. The blood's delivery of nutrients and oxygen is essential for the brain's proper operation, just like it is for all other organs. If the blood flow is restricted or stopped, brain cells begin to deteriorate. A brain damage, a disability, or even death might occur from this. A balanced, healthy lifestyle that abstains from harmful practises like drinking and smoking and manages body mass index can reduce the risk of stroke. The dataset that was utilised in this study was discovered on Kaggle and was initially acquired from different medical centres in Bangladesh [2]. It was collected by researchers from Jahangir University of Bangladesh for research on stroke. The patient details are collected based on their various health conditions, which is in the occurrence of stroke disease. The dataset contains data on 5110 individuals, which are of following types:

1) Categorical Features: gender, ever married, work type, Residence type, smoking status

2) Binary Numerical Features: hypertension, heart disease, stroke

3) Continuous Numerical Features: age, avg glucose level, bmi

Each of their attributes are listed below with its description:

Attributes	Description	Values
id	Represents special identification number which is a unique number assigned to the patient and used for identification purpose	5-digit number
gender	Represents the gender of the patient like where its female, male or other	Male, Female, Other
age	Represents the patient's age	Number between 1 - 150
hypertension	0 if the patient doesn't have hypertension, 1 if the patient has hypertension	1 or 0
heart_disease	If the patient has heart disease, then 1 or else 0	1 or 0
ever_married	Represents whether the patient is married or not? Yes, if patient is married and no patient is not married	Yes or No
work_type	Represents the work scenario of the patient	"children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
residence_type	Represents the living scenario of patients	"Rural" or "Urban"
avg_glucose_level	Patient's average glucose level in blood	Number between 0 – 30
bmi	Patient's body mass index (bmi)	Number between 0 - 200
smoking_status	Patient's status of smoking	"formerly smoked", "never smoked", "smokes" or "Unknown"
stroke	Denotes whether patient previously had a stroke or not, if had a stroke then 1 otherwise 0	1 or 0

Table 1: Description of attributes and its values

So here our target variable is stroke, and all other attributes are our feature variable.

1.2 Supervised learning task identification

A prediction or inference created in response to a problem or question and the present state of the data is known as a machine learning task. The classification task, for instance, classifies data. supervised machine learning is used to decide which of two classes (categories) a given item of data belongs to.

Here we are trying to build a predictive model which is capable of automatically predicting whether a given patient should be diagnosed with stroke. The selected target ground truth value is "stroke" and stroke is a nominal variable. Classification techniques are used to predict nominal values (categories).

Based on problem mentioned above we get the collection of following factors:

A) The selected target ground truth value is "stroke"

B) "stroke" is a nominal variable

C) Classification techniques are used to predict nominal values (categories)

Because A, B, C are all true, therefore, due to our target ground truth value "stroke" being a nominal variable, we believe that classification would be the appropriate supervised learning task to build this predictive model.

2 Exploratory Data Analysis

As now we have identified our dataset and supervised learning task which are stroke and classification technique. Now, we will try to get some initial pattern from our data to further help to choose an algorithm and create our model and evaluate our result. So this section we will go through the Exploratory Data Analysis process by starting with asking questions to our data then splitting the dataset to get a train setting on which we can perform our exploratory data analysis.

2.1 Questions identification

Having a well stated research question allows investigators to focus their study and work toward supporting or rejecting a certain hypothesis. Once the research question has been defined, the remaining components of the study can be identified logically. [3]. And it allows us to get clear answer. So by general observation of the stroke dataset and literature search about stroke we will investigate on the following questions and assumptions to allow us to specifically confirm / reject our assumed answer at the conclusion of the EDA:

Sr. No.	Questions	Assumptions
1.	Do older patients have an increased likelihood of being diagnosed with a stroke?	The risk increases with age, the incidence doubling with each decade after the age of 45 years and over 70% of all strokes occur above the age of 65 [4]. That means the older you become, the more probable it is that you will have a stroke [5]. As cardiovascular and metabolic disease incidence rises with age, older people are more likely to experience strokes. [6]
2.	Does have hypertension or heart disease increase a patient's likelihood of being diagnosed with a stroke?	Blockages and decreased blood flow are brought on by hypertension. A stroke is caused by a blockage close to the brain. As a result, patients who have hypertension are more prone to suffer a stroke. But because a blockage close to the heart causes a number of cardiac disorders, hypertension is also linked to heart disease. So, stroke has a direct causal relationship with both hypertension and heart disease. A patient's likelihood of having heart disease increases if they have hypertension. As a result, the patient has an increased chance of experiencing a stroke.
3.	Does a specific BMI, along with high glucose levels, increase a patient's likelihood of being diagnosed with a stroke?	Too much sugar in your blood makes the glucose level high. Too much intake of sugar can increase weight as well so there are more chances of getting overweight which can make the BMI level above the threshold and as blood sugar levels increases, it damages the blood vessels and in result increases the risk of Stroke. [7]

4.	Does a patient's employment type increase their likelihood of being diagnosed with a stroke?	Work pressure can sometime become stressful which can cause hypertension and hypertension is caused due to blood clots formed in the arteries leading to the brain, blocking blood flow which can potentially cause stroke. [8] Private sector employees may be more susceptible to stroke due to workload, long working hours and stress. [9]
5.	Does being retired increase a patient's likelihood of being diagnosed with a stroke?	In retirement people feel bored, aimless, isolated and also affects the mental health make people depressed, sometime retired people may grieve of loss of old life which sometime become stressful, in result it can become hypertension which may lead them to chances of getting stroke [10].
6.	Does a high workload increase hypertension, which in turn increases a patient's likelihood of being diagnosed with a stroke?	High workload can make people work longer hours and can become stressful which may lead to hypertension and blood blockages may occur due to hypertension and can possibly cause stroke.
7.	Does being unemployed increase BMI levels, which in turn increase a patient's likelihood of being diagnosed with a stroke?	Unemployed tend to have less healthy behaviours including smoking, alcohol use and physical inactivity which makes their body overweight which increases BMI levels. Inflammation is probable as a result of excess fat in the body, resulting in impaired blood flow and possible blockages which can increase the chances of stroke [11]. And unemployment also leads to mental stress, sleep loss which can lead to hypertension so which can may be increase the risk of stroke.
8.	Do female patients have an increased likelihood of being diagnosed with a stroke due to balancing home and work lives?	The typical lifespan is greater for female [12], and as people age increase, their chance of having a stroke rises. In addition, female's hormonal imbalances, pregnancy, delivery, reproductive health, and work-life balance, all can sometime become stressful which can increase the risk of stroke.

Table 3: Questions with assumptions on the dataset

2.2 Splitting the dataset

The goal of dividing our data into training and validating and testing subsets is to be able to represent previously "seen", and currently "initially unseen" and "unseen" data throughout the model building process. As a result, almost all of your pre-processing (and even EDA) should be done on a sizable sample of previously "seen" data. This is to prevent any patterns / insights from accidentally leaking into the model and interfering with / biassing the evaluation results obtained on this "initially unseen" and "unseen" data at the end of our model development, thereby preventing data leakage [13]. And the initially unseen evaluation can be used to improve model and get better evaluation results on "unseen" data.

So here we are splitting the data into three parts: train ("seen"), validate("initially unseen") and test ("unseen").

To split the data we used the **model_selection.train_test_split** method from the Scikit Learn package to get two subsets of data, one for intermediate and one for testing and dividing into the features (the X values) and the targets (the Y values). In the method, we have the **random_state** value. The **random_state** can be any positive number,

(except 0). It is the parameter used to control the random number generator used [14]. We need to pick a number to be our **random_state**. There are a few ways to split the dataset into such as splitting the data into 70:30 ratio or 80:20.

```
stroke_X_intermediate, stroke_X_test, stroke_Y_intermediate, stroke_Y_test =  \
  model_selection.train_test_split(df_stroke.drop(labels=["stroke"], axis='columns'),
                                   df_stroke["stroke"],test_size=0.10, random_state=42)

stroke_X_train, stroke_X_validate, stroke_Y_train, stroke_Y_validate =  \
  model_selection.train_test_split(stroke_X_intermediate, stroke_Y_intermediate, test_size=0.1, random_state=42)
```

Code-snippet 1:

Code-snippet 1 : Train, Validate and Test Split

A straightforward way to do this is to use the **model_selection.train_test_split** method which will first split the dataset into intermediate (90%), testing (10%). Then, use the **model_selection.train_test_split** method again on the intermediate data only. This will split the intermediate dataset into training (80%) and validation (10%). So we are keeping the text size as 0.1 and here, we are defining the random state in our code because, if we don't, a new random value would be generated each time we run (execute) our code, changing the values of the train, test, and validation datasets.

```
df_stroke_train = deepcopy(stroke_X_train)
df_stroke_train["stroke"] = stroke_Y_train

df_stroke_validate = deepcopy(stroke_X_validate)
df_stroke_validate["stroke"] = stroke_Y_validate

df_stroke_test = deepcopy(stroke_X_test)
df_stroke_test["stroke"] = stroke_Y_test
```

Code-snippet 2: Generating deep copies of data frame

As well as we are saving our test data with the data frame for later and data frame with all our training data which are done as deep copy, which we will be used for EDA and model building. Then checking the shape of the train, test and validate data to ensure its properly divided:

```
[ ]  stroke_X_train.shape
     (4139, 11)
[ ]  df_stroke_train.shape
     (4139, 12)
[ ]  stroke_X_test.shape
     (511, 11)
[ ]  df_stroke_test.shape
     (511, 12)
[ ]  stroke_X_validate.shape
     (460, 11)
[ ]  df_stroke_validate.shape
     (460, 12)
```

Code-snippet 3: Checking the shape of the data frames after split

2.3 Exploratory Data Analysis process and results

To prevent data leakage as discussed in 2.2 and avoid overfitting which is not getting good predictions on unseen data rather than seen, we will be doing our EDA solely on seen data which is training set while minimising the development of too many insights on analysing unseen data beforehand. We will start by visualising the potential relationships between feature and target variable using visualising libraries such as seaborn and matplotlib. We will performing univariate, bivariate, and multivariate analysis on our feature and target variables using count plot, histogram, and boxplot.

2.3.1 Univariate Analysis

```
sns.set_style("darkgrid")
stroke = sns.countplot(df_stroke_train['stroke'])
for p in stroke.patches:
    height = p.get_height()
    stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 4: Code to visualise count plot of target variable - Stroke

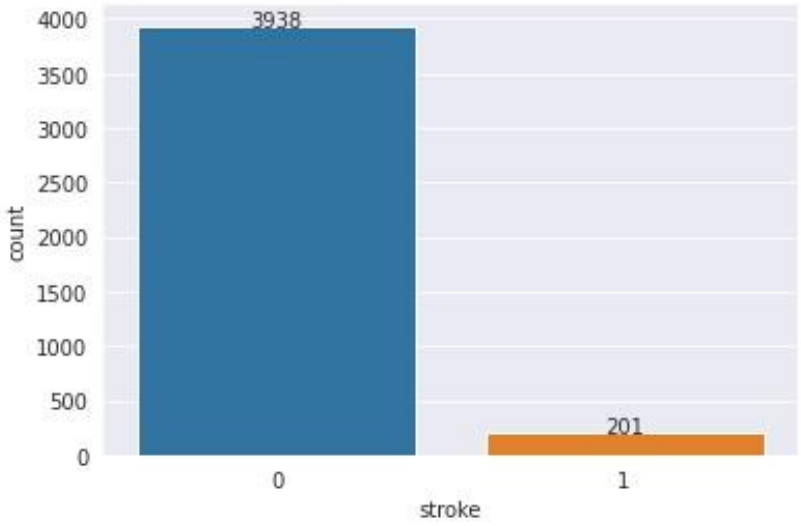


Figure 1: Count of patients with and without stroke in the training data set

In Figure 1, less than 500 patients have experienced stroke in our training set, while more than 3500 have not. So 201 patients out of 4139 have a stroke, whereas 3938 do not. This means highly unbalanced data distribution.

```
sns.set_style("darkgrid")
gender = sns.countplot(df_stroke_train['gender'])
for p in gender.patches:
    height = p.get_height()
    gender.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 5: Code to visualise count plot of feature variable - Gender

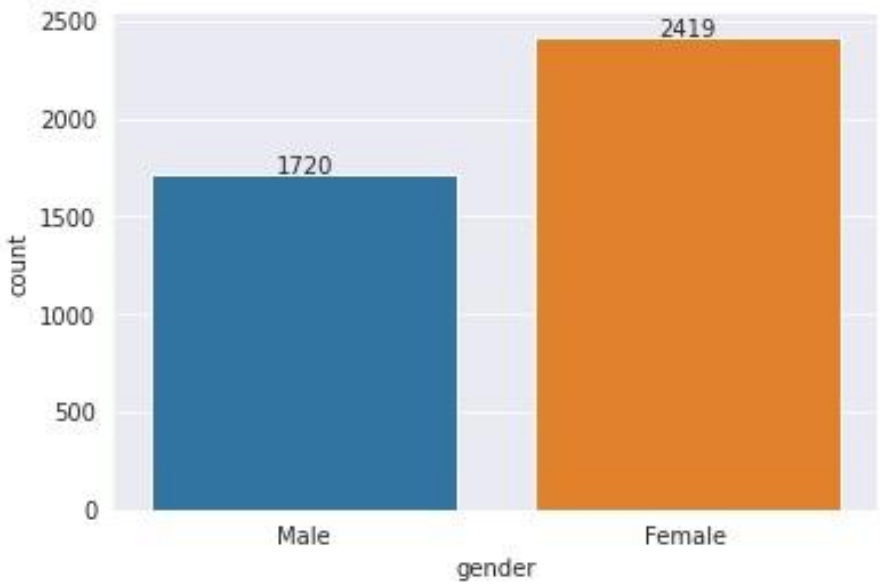


Figure 2: Count of Female and Male patients in the training data set

Females outweigh males in our training data as we can see in Figure 2. Out of a total of 4139 patients, there are 2419 women and 1720 males.

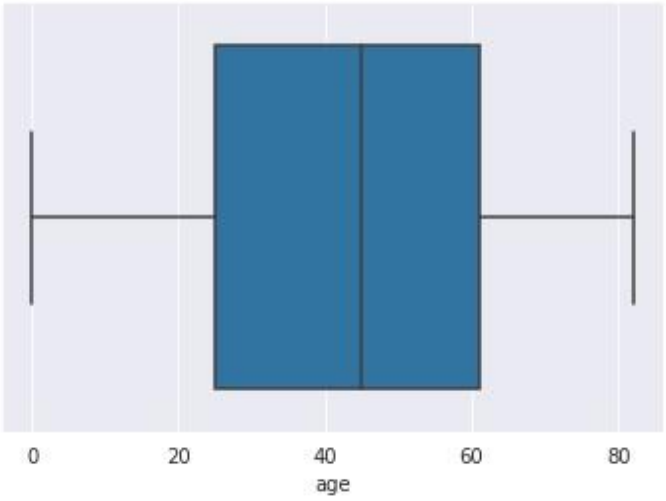
```
sns.histplot(df_stroke_train['age'],  kde=True)
```

```
sns.boxplot(df_stroke_train['age'])
df_stroke_train['age'].describe()
```

Code-snippet 6 & 7: Code to visualise distribution of age variable Output

of code:

```
count    4139.000000
mean       43.328002
std        22.733361
min         0.080000
25%        25.000000
50%        45.000000
75%        61.000000
max        82.000000
```



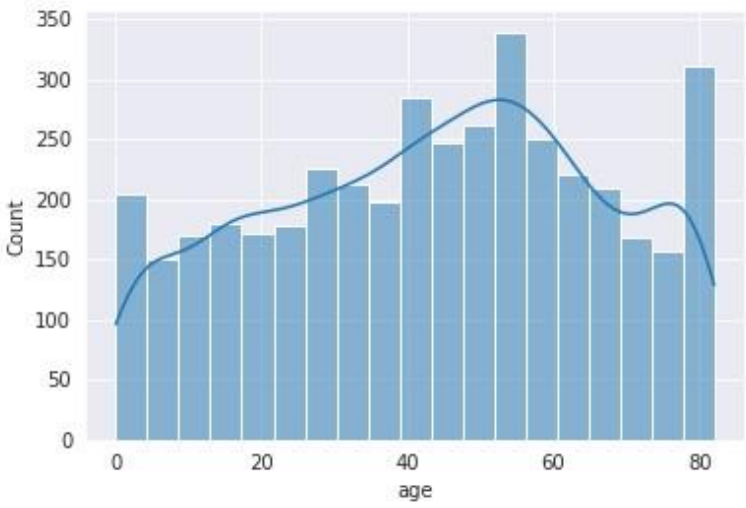


Figure 3 & 4: Age variable distribution

We can see for the Stroke dataset's Age attribute:

- Q1 (the 25th percentile) is about 25 years. So, 25% of all patients are less than or equal to 25 years
- Q2 (the median and 50th percentile) is about 45 years. So, 50% of all patients are less than or equal to 45 years
- Q3 (the 75th percentile) is about 61 years. So, 75% of all patients are less than or equal to 45 years

There for most of the patients falls in between the age of 25 years to 61 years in our training data. By keep in my mind above all point and the distribution is somewhat negatively skewed which slightly supports our first assumption that as age, the chances of getting a stroke increase because most of the patients are mid- adults or older and the skew also shows that.

```
hypertension = sns.countplot(df_stroke_train['hypertension'])
for p in hypertension.patches:
    height = p.get_height()
    hypertension.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
hypertension.set_xticklabels(labels=['No', 'Yes'])
```

Code-snippet 8: Code to visualise count plot of feature variable - Hypertension

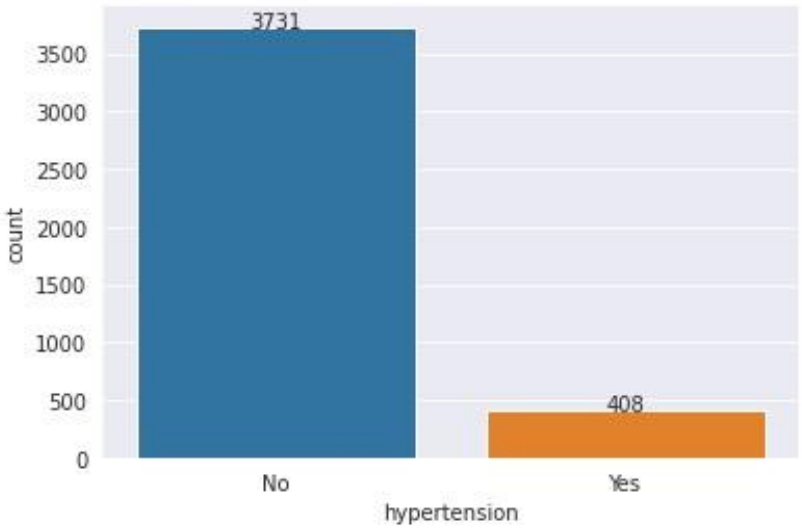


Figure 5: Count of patients with hypertension and without in the training data set

There are much fewer people who do not have hypertension than those who do. Out of 4139 people in our training set, 3731 do not have hypertension and 408 do.

```
heart_disease = sns.countplot(df_stroke_train['heart_disease'])
for p in heart_disease.patches:
    height = p.get_height()
    heart_disease.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
heart_disease.set_xticklabels(labels=['No', 'Yes'])
```

Code-snippet 9: Code to visualise count plot of feature variable – Heart disease

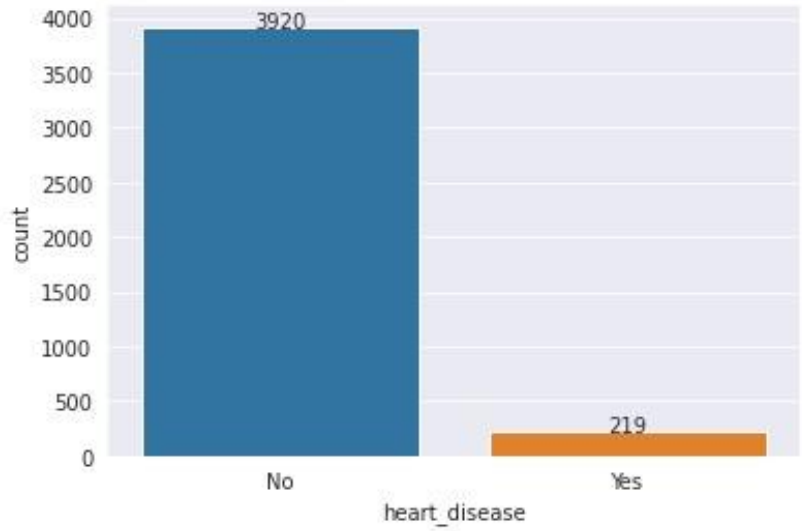


Figure 6: Count of patients with heart disease and without in the training data set

Less patients have heart disease than who don't. 219 patients have heart disease out of 4139 patients and 3920 patients have no heart disease.

```
ever_married = sns.countplot(df_stroke_train['ever_married'])
for p in ever_married.patches:
    height = p.get_height()
    ever_married.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 10: Code to visualise count plot of feature variable – Ever married

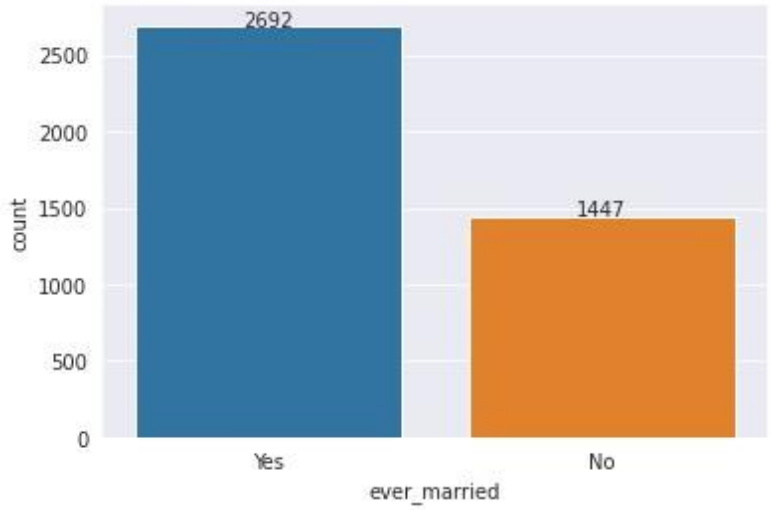


Figure 7: Count of patients who are married and not married in the training data set

The number of married people outnumbers the number of unmarried people (which makes sense given the distribution of age 25 to 65 in the Figure 3 & 4).

```
work_type = sns.countplot(df_stroke_train['work_type'])
for p in work_type.patches:
    height = p.get_height()
    work_type.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 11: Code to visualise count plot of feature variable – Work type

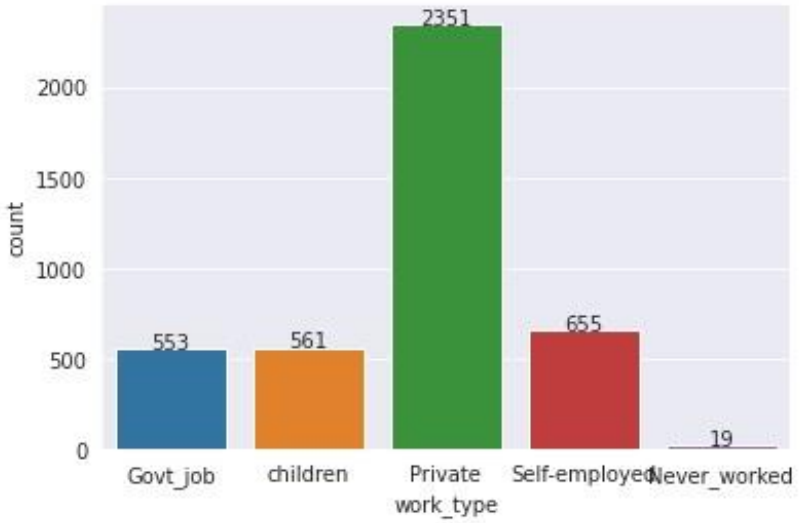


Figure 8: Count of patients with scenario they work in, in the training data set

According to our training data most of our patients working in private sectors and the second most are selfemployed and the patients who have children and have government job are of similar amount. There are least people who never worked in our training data.

In reference to figure 7, married people have responsibility in real world which shows they might be working in private sector so here we can see most of the patients are working in private.

```
Residence_type = sns.countplot(df_stroke_train['Residence_type'])
for p in Residence_type.patches:
    height = p.get_height()
    Residence_type.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 12: Code to visualise count plot of feature variable – Residence type

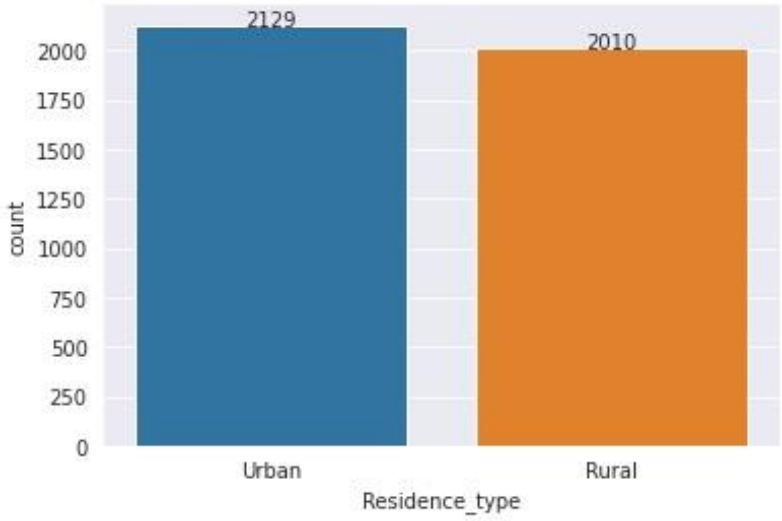


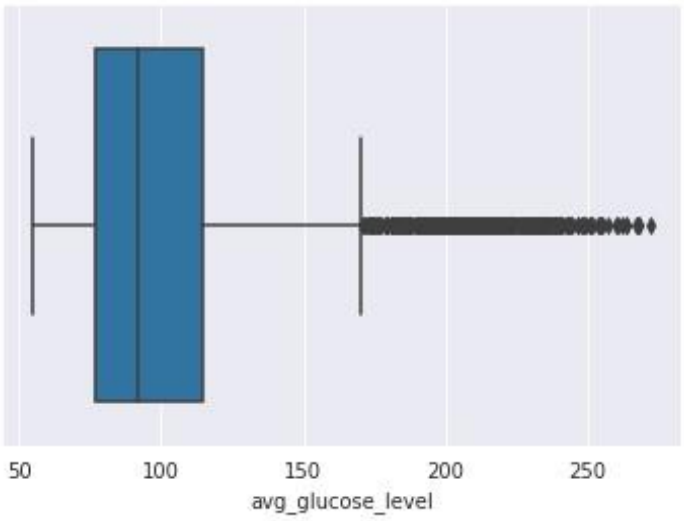
Figure 9: Count of patients living in Rural or Urban Area

Most of the patients are living in urban which are 2129 patients compared to rural which are 2010 patients in our training dataset. As from figure 8, most of the people are working in private sector that show here in figure 9 that they might be living in urban area as most jobs are in city than villages in real world.

```
sns.histplot(df_stroke_train['avg_glucose_level'],  kde=True)
sns.boxplot(df_stroke_train['avg_glucose_level'])
df_stroke_train["avg_glucose_level"].describe()
```

Code-snippet 13 & 14: Code to visualise distribution of average glucose level variable Output of code:

```
count    4139.000000
mean      106.273283
std        45.457825
min        55.120000
25%        77.080000
50%        91.890000
75%       114.465000
max       271.740000
```



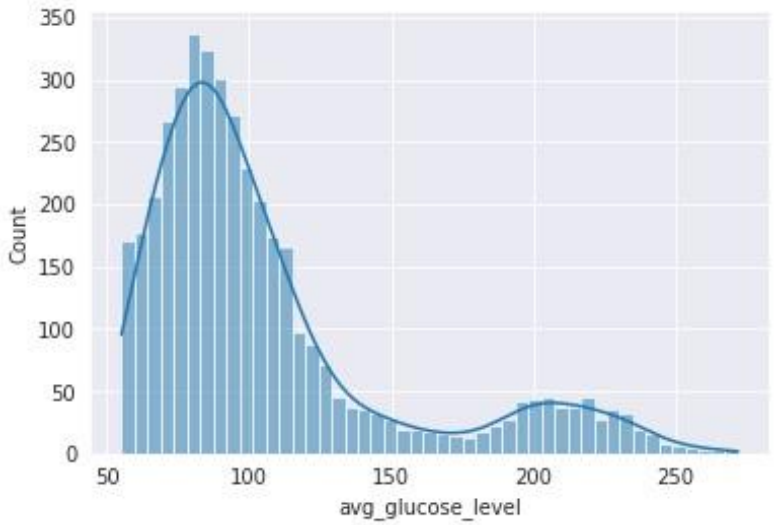


Figure 10 & 11: Distribution of average glucose level in blood of patients

We can see for the Stroke training dataset's average glucose level attribute:

- Q1 (the 25th percentile) is about 77 mg/dl. So, 25% of all patients have less than or equal to 77 mg/dl which is good.
- Q2 (the median and 50th percentile) is about 92 mg/dl. So, 50% of all patients are less than or equal to 92 mg/dl which is good.
- Q3 (the 75th percentile) is about 115 mg/dl. So, 75% of all patients are less than or equal to 115 mg/dl which is also good.

So most of the patients have glucose level between 77 to 115 mg/dl in our training data and the distribution is positive skewed. That indicates most of the people are not diabetic patients. And the outliers shows that there are few diabetic patients which are outside the "minimum" and "maximum" expected range of values which is more than 115 mg/dl. And the distribution is positive skewed.

```
sns.histplot(df_stroke_train['bmi'], kde=True)
sns.boxplot(df_stroke_train['bmi'])
df_stroke_train["bmi"].describe()
```

Code-snippet 15 & 16: Code to visualise distribution of bmi (body mass index) variable Output of code:

```
count    4139.000000
mean       28.845736
std         7.661763
min        10.300000
25%        23.800000
50%        28.300000
75%        32.700000
max        97.600000
```

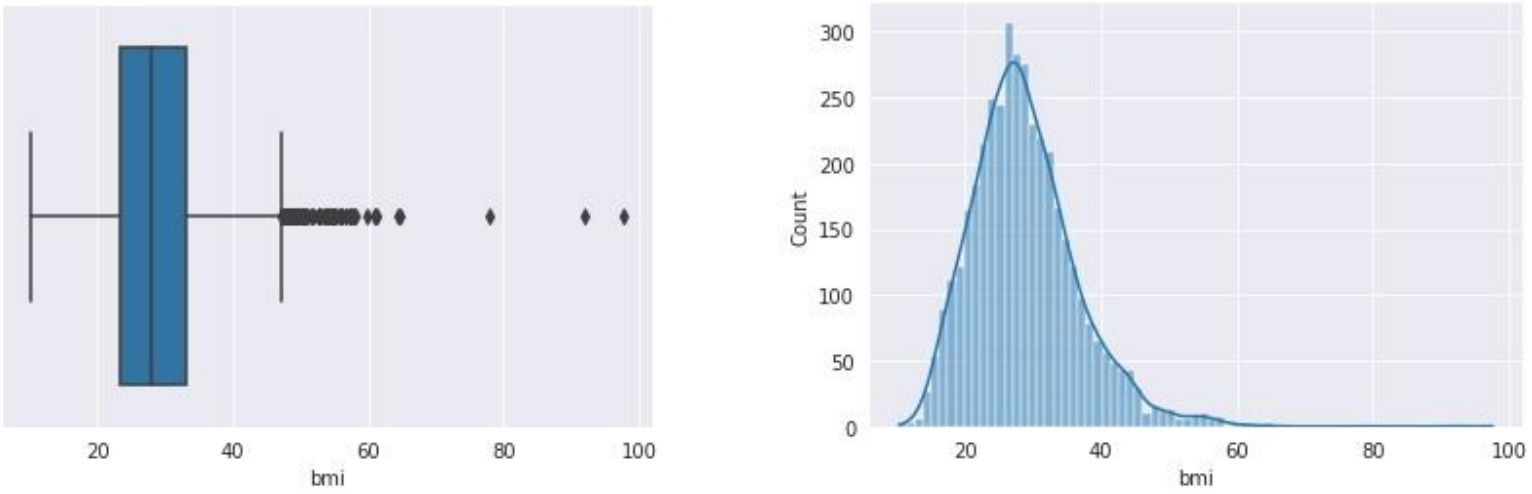


Figure 11 & 12: Distribution of BMI (Body Mass Index) of patients

We can see for the Stroke training dataset's average glucose level attribute:

- Q1 (the 25th percentile) is about 24. So, 25% of all patients have less than or equal to 24 which are good.
- Q2 (the median and 50th percentile) is about 28.30. So, 50% of all patients are less than or equal to 28.30 which are overweight.
- Q3 (the 75th percentile) is about 33. So, 75% of all patients are less than or equal to 33 which are also overweight.

So most of the patients have BMI between 24 to 33 in our training data and the normal BMI range is **18.5 to 24.9**. So it shows most of the patients are overweight in our training data. And the outliers shows that there are patients who are overweight which are outside the "minimum" and "maximum" expected range of values, who have more than 33 BMI which slightly support our 3rd assumption in 2.1 so they can be prone to get a stroke.

```
smoking_status = sns.countplot(df_stroke_train['smoking_status'])
for p in smoking_status.patches:
    height = p.get_height()
    smoking_status.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 17: Code to visualise count plot of feature variable – Smoking status

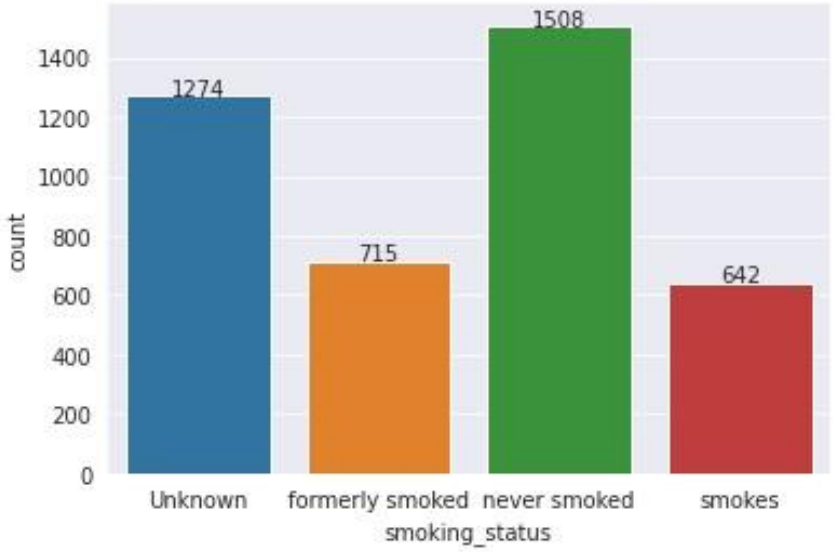


Figure 13: Count of patient's smoking status

In our data most of the stroke patients have Never Smoked which are 1508 out of 4139 and second highest count is of unknown so other than never smoked there are more patients whose smoking status is unknown in our training data. There might be patients who don't who to share their smoking status or the value might be not filled or missed during the data collection.

2.3.2 Bivariate Analysis

```
gender_vs_stroke = sns.countplot(df_stroke_train["gender"], hue=df_stroke_train["stroke"])
for p in gender_vs_stroke.patches:
    height = p.get_height()
    gender_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 18: Code to visualise the count plot of feature - gender with target - Stroke

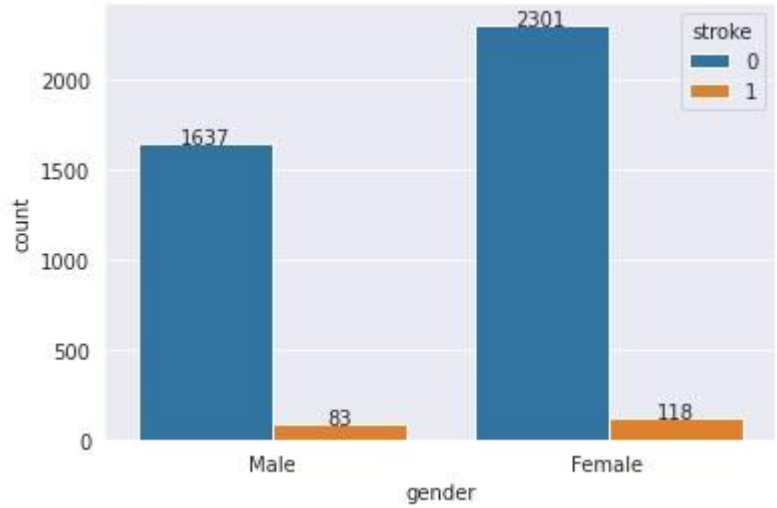


Figure 14: Count of Female and Male patients with and without Stroke

The overall number of female participants in this dataset is higher than the total number of male patients. There are more 35 females than the count of males who had stroke. Currently we are unable to find compelling evidence in our dataset that supports our 8th assumption in 2.1 fully and answers 8th question as well.

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

Code-snippet 19: Code to visualise the boxplot for distribution of target - stroke among feature - age

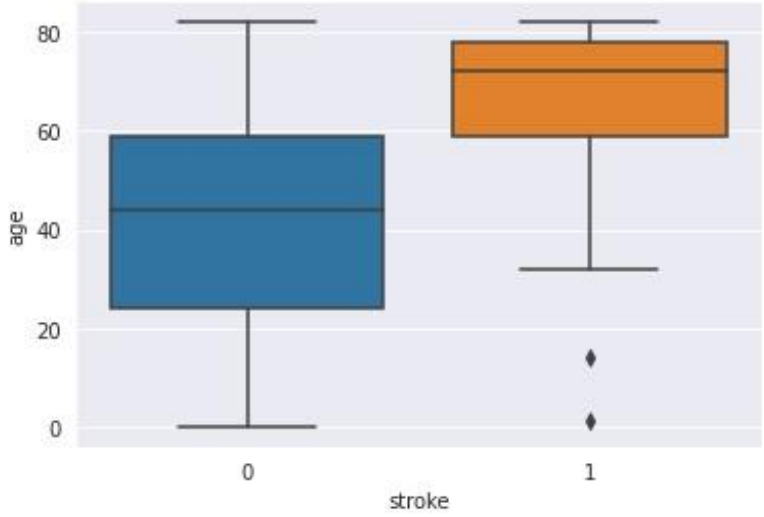


Figure 15: Distribution of patients having stroke among age

Despite the fact that the majority of patients did not have a stroke, it can occur at any age. The age range of participants who have had a stroke is 60 to 80 as we can see in figure 15. So this support our 1st assumption which we did in 2.1 and answers our 1st question that older patients have an increased likelihood of being diagnosed with a stroke. And as we can see, there are two outliers that indicate that two individuals between the age of 0 and 20 had a stroke, which may be accurate given that although many people think of stroke as an old disease, there is an incredibly significant chance that it will happen to a child during the perinatal era [15].

```
hypertension_vs_stroke = sns.countplot(df_stroke_train["hypertension"], hue=df_stroke_train["stroke"])
for p in hypertension_vs_stroke.patches:
    height = p.get_height()
    hypertension_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 20: Code to visualise the count plot of feature - hypertension with target - stroke

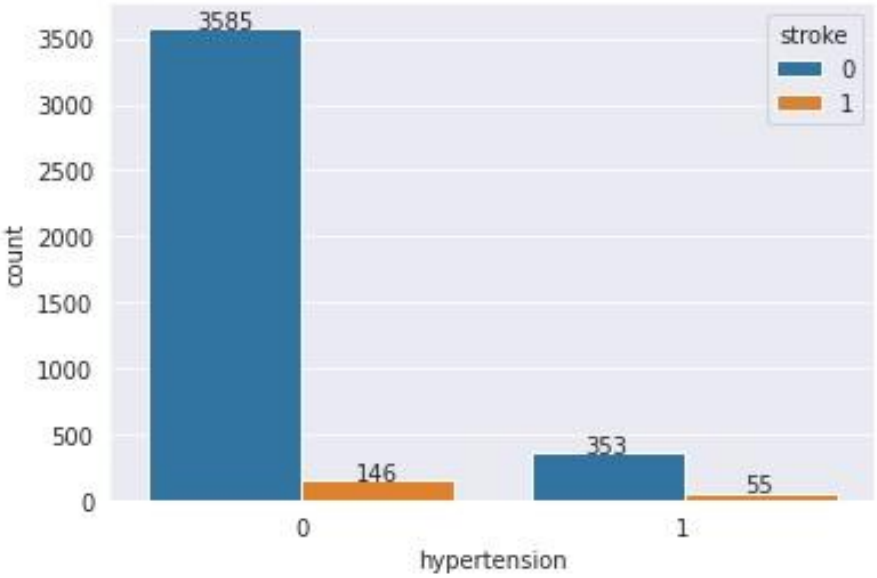


Figure 16: Count of patients with and without hypertension vs with and without stroke

In this training dataset, there are significantly more people without hypertension than those who do. The patients who had hypertension also had stroke are less than patient who had stroke but did not had hypertension, so, here it answers our 2nd question in 2.1 and also does not support our 2nd assumption that patients who have hypertension likelihood of being diagnosed with a stroke because here the number of patients who had hypertension but does not had stroke are more than who had stroke had hypertension.

```
heart_disease_vs_stroke = sns.countplot(df_stroke_train["heart_disease"], hue=df_stroke_train["stroke"])
for p in heart_disease_vs_stroke.patches:
    height = p.get_height()
    heart_disease_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 21: Code to visualise the count plot of feature – heart disease with target - stroke

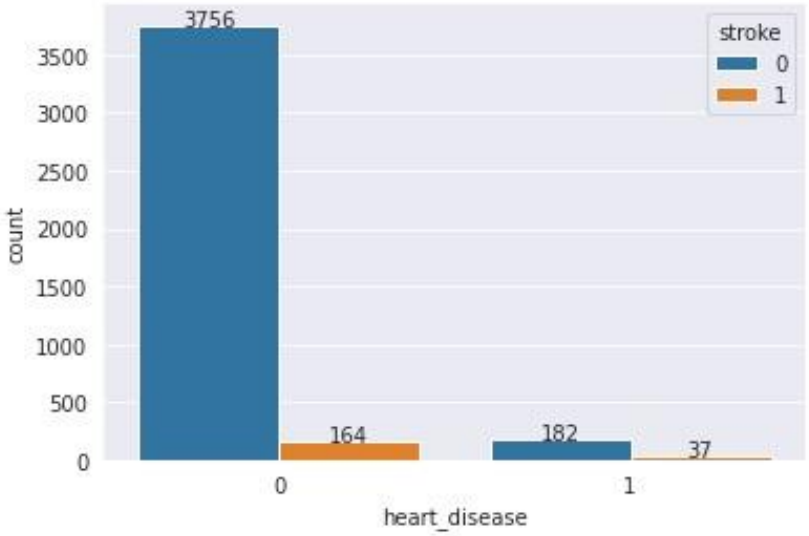


Figure 17: Count of patients with and without heart disease vs with and without stroke

There are very few people who had heart disease than people who don't. The patients who had heart disease also had stroke are less than patients who had stroke but who did not had heart disease, so, here it answers our 2nd question in 2.1 and also does not support our 2nd assumption that patients who have heart disease likelihood of being diagnosed with a stroke because here the number of patients who had heart disease but does not had stroke are more than who had stroke had heart disease.

```
martial_vs_stroke = sns.countplot(df_stroke_train["ever_married"], hue=df_stroke_train["stroke"])
for p in martial_vs_stroke.patches:
    height = p.get_height()
    martial_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 22: Code to visualise the count plot of feature – ever married with target - stroke

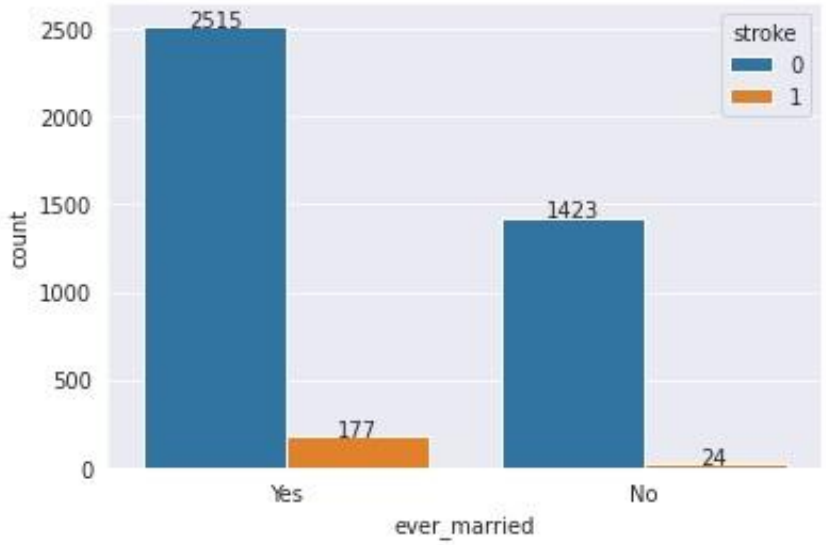


Figure 18: Count of patients married and not married vs with and without stroke

Based on the above figure, it seems patient who are married had stroke are more than patients who are not married and had not had stroke. Married people have more responsibility and have to manage work-life-balance so they might have more stress which can be a reason for stroke.

```
work_vs_stroke = sns.countplot(df_stroke_train["work_type"], hue=df_stroke_train["stroke"])
for p in work_vs_stroke.patches:
    height = p.get_height()
    work_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 23: Code to visualise the count plot of feature – work type with target - stroke

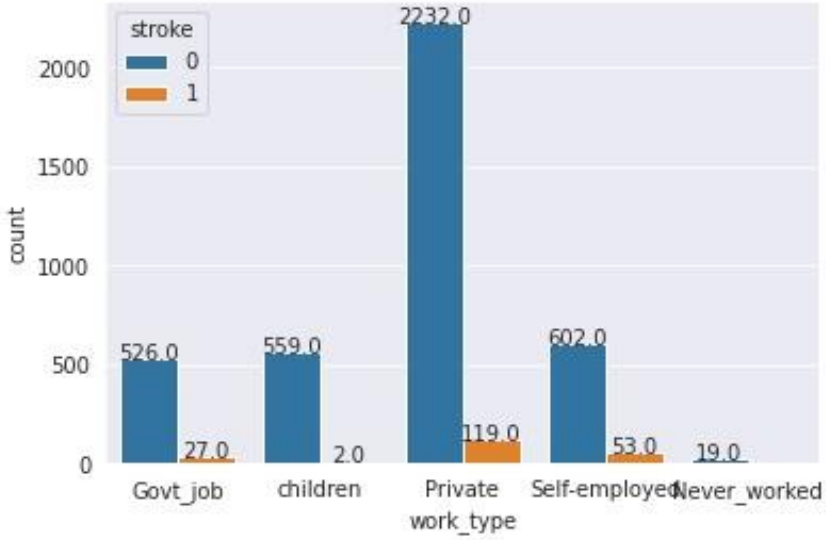


Figure 19: Count of patient's working scenario vs with and without stroke

Most stroke patients are either self-employed or employed by the government in addition to the private sector. The majority of stroke patients among them all work in the private sector which supports our 4th assumption in 2.1 that given their workload and potential for stress, private sector employees may be more susceptible to stroke and also answer the 4th question that patients employment type does increase their likelihood of getting stroke and here we can see patients who are working in private sector are more prone to get a stroke.

```
residence_vs_stroke = sns.countplot(df_stroke_train["Residence_type"], hue=df_stroke_train["stroke"])
for p in residence_vs_stroke.patches:
    height = p.get_height()
    residence_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 24: Code to visualise the count plot of feature – residence type with target - stroke

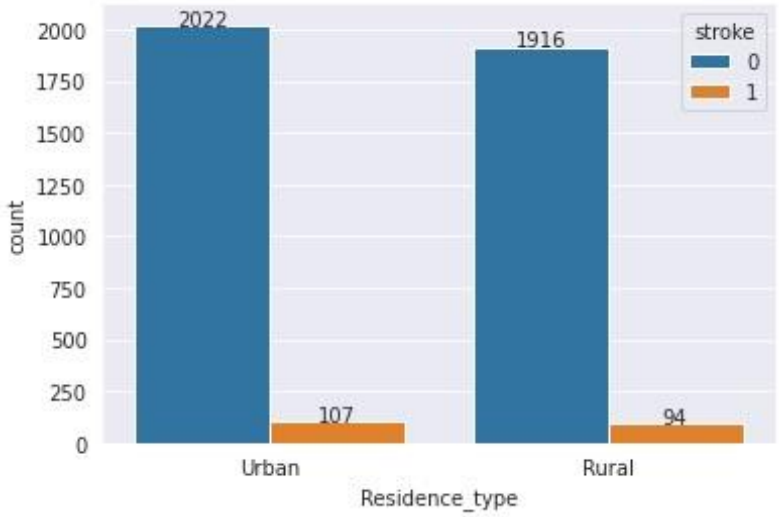


Figure 20: Count of patient's dwelling type vs with and without stroke

From the above figure, we can see patients who are living in urban had stroke outnumbers patients who are living in rural had stroke. May be patients who are living in urban might have more stressful life, might be working in private sectors(refer. Fig. 19), might be married (refer. Fig. 18) and moved from rural to urban area due to job (refer. Fig. 19), which can be reasons for likelihood of getting a stroke.

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

Code-snippet 25: Code to visualise the boxplot for distribution of target - stroke among feature – avg. glucose level

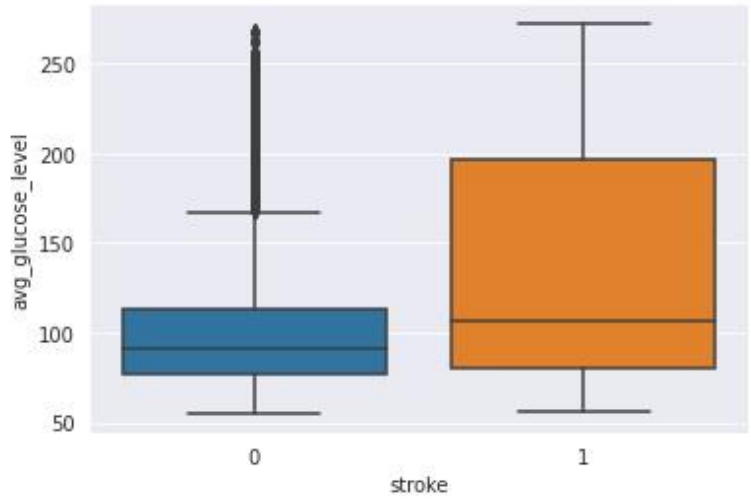


Figure 21: Distribution of patient's average glucose level among stroke and no stroke

The boxplot above show that patients who have had a stroke have had higher average blood glucose level than patients who have not. 77 mg/ dL to 199 mg/dL is the range of average blood glucose level in the patients who had stroke and in that the patient who had more than 125 mg/ dL or higher average blood glucose level indicates that the patients had diabetes. So as blood sugar levels increases, it damages the blood vessels and in result increases the risk of Stroke which supports our 3rd assumption in 2.1.

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

Code-snippet 26: Code to visualise the boxplot for distribution of target - stroke among feature – bmi

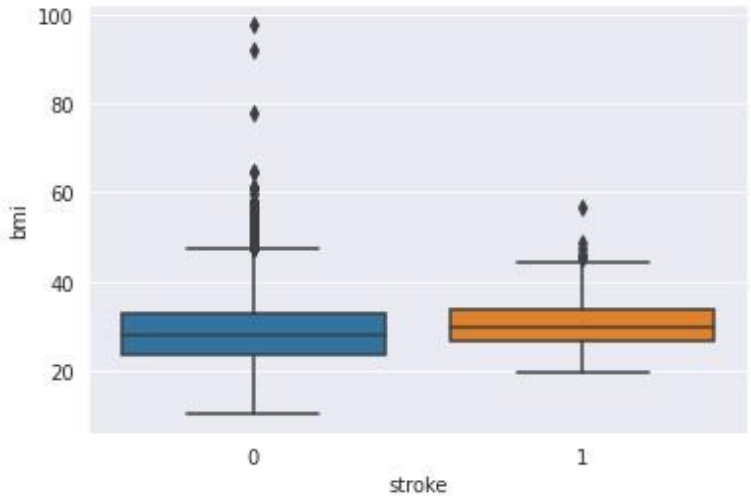


Figure 22: Distribution of patient's BMI (Body Mass Index) among stroke and no stroke

According to the boxplot above, patients who have had a stroke had higher BMI which is between 29 to 33 which is higher than normal BMI range is **18.5 to 24.9**. This slightly support our 3rd assumption in 2.1 that higher BMI, increases the chances of getting a stroke. The outliers here tells that few people have higher than 33 BMI so their chances of getting stroke is greater than people who have BMI between 29 to 33.

```
smoke_vs_stroke = sns.countplot(df_stroke_train["smoking_status"], hue=df_stroke_train["stroke"])
for p in smoke_vs_stroke.patches:
    height = p.get_height()
    smoke_vs_stroke.text(p.get_x()+p.get_width()/2., height + 0.1,height ,ha="center")
```

Code-snippet 27: Code to visualise the count plot of feature – smoking status with target - stroke

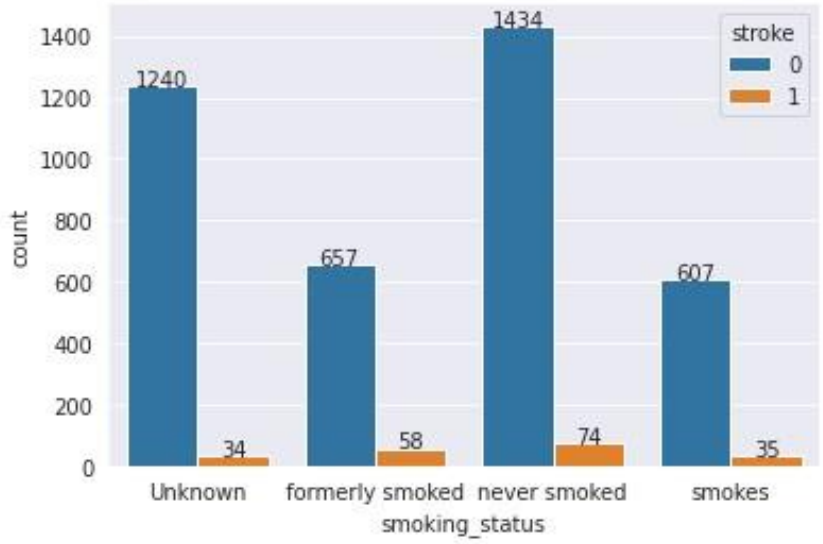


Figure 23: Count of patient's smoking status vs with and without stroke

We can observe that patients who never smoked are higher in terms of have had a stroke then patients who smokes and formerly smokes have had a stroke which clears that smoking doesn't have effect on getting a stroke but their might be slight chance if we get to know the unknown values, which might be patient who don't want to share their smoking status and they might be formerly smoking or smokes then it will increase the chance of getting a stroke.

2.3.3 Multivariate Analysis

And then we performed multivariate analysis of relationship between the features and features variable by

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

Code-snippet 28: Code to visualise pair plot

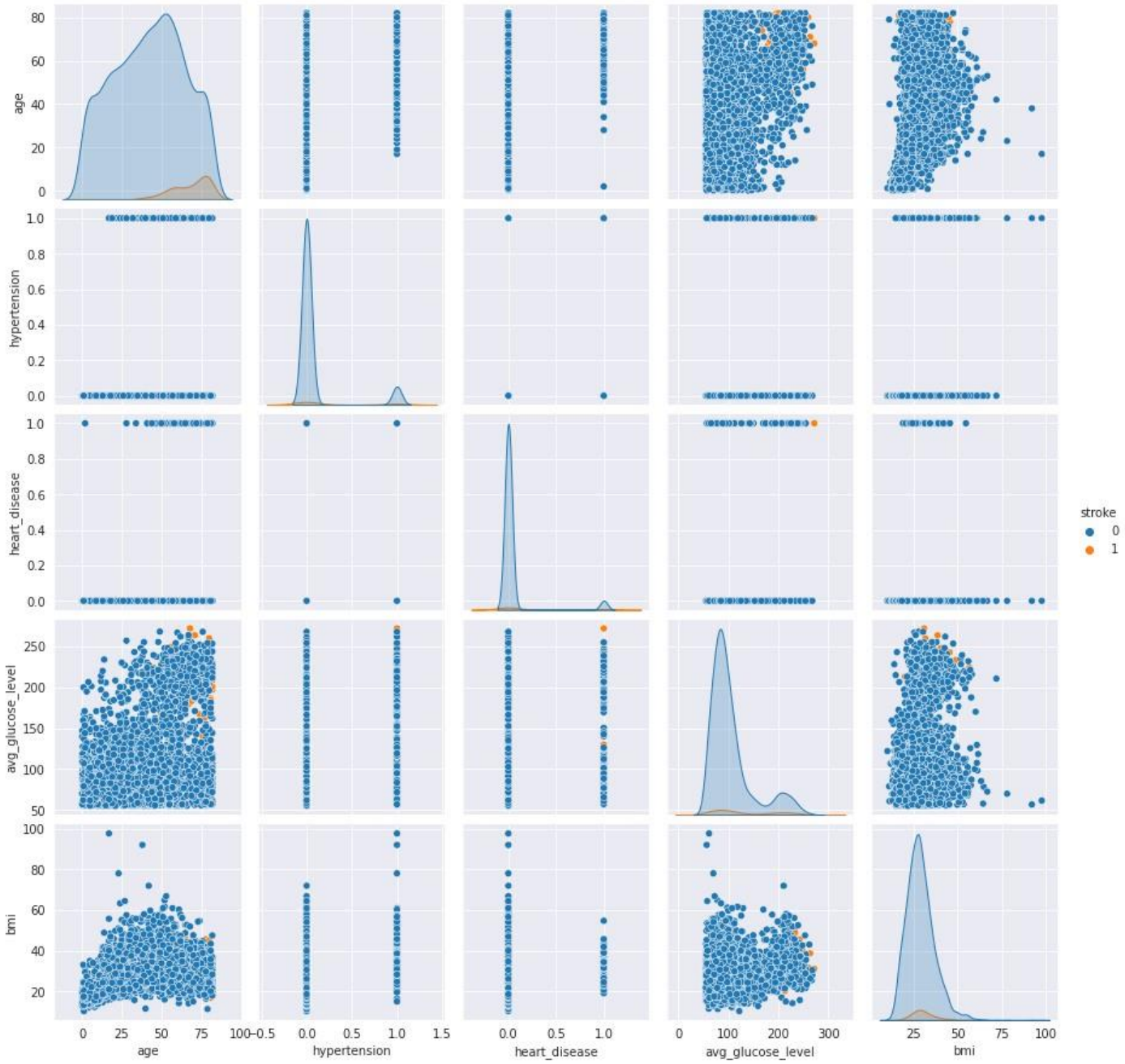


Figure 24: Pair plot of features vs features

We can observe the following from the pair plot:

1. Patients whose age was between 60 to 80, had average glucose level between 150 to 290 mg/dL had stroke.
2. Patients whose age was more than 75, had BMI between 25 to 50 had stroke.
3. Patient who had hypertension and average glucose level more that 250 mg/dL had stroke.
4. Patients who had average glucose level between 125 to 275 mg/dL and age between 45 to 80 had stroke.
5. Patient who had heart disease and had average glucose level 295 mg/dl had stroke.
6. Patient who had heart disease and had average glucose level 145 mg/dl had stroke.
7. Patients who had average glucose level between 200 to 295 mg/dL and BMI between 20 to 60 had stroke which answers our 3rd question in 2.1.
8. Here it does not support our 4th assumption in 2.1 when we see the plot age vs hypertension because most of the patients are without stroke.

To see the correlation between target and numerical features we will be using Pearson method from pandas.

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

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

Code snippet 29 & 30: Visualise Pearson's correlation between target and numerical features

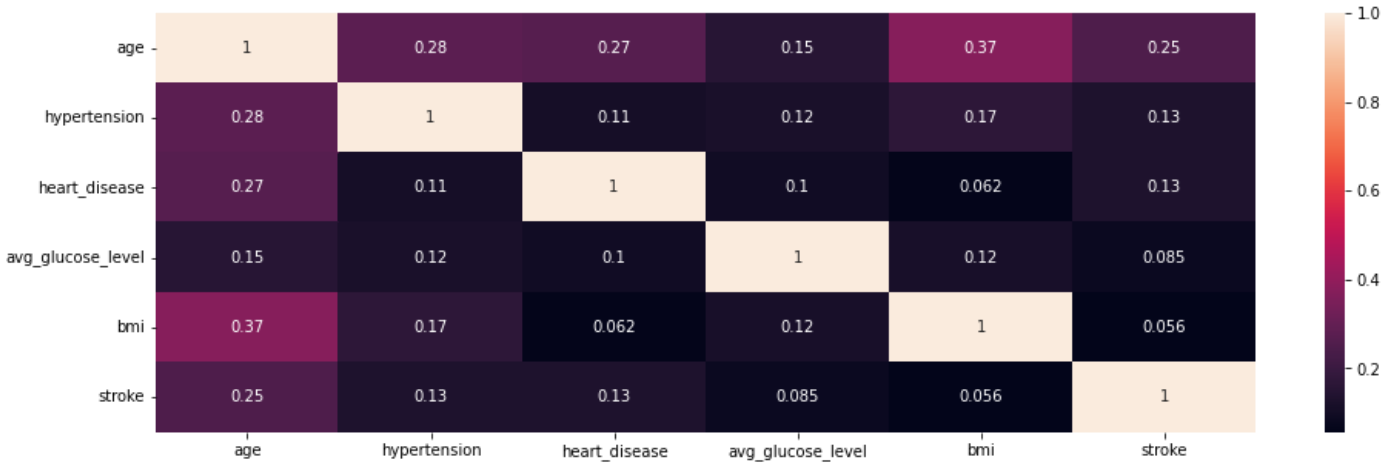


Figure 25: Correlation between target and numerical features

The features may be found to have little correlation with one another. Like positive correlation between age and BMI, hypertension, heart disease and target - stroke.

2.4 EDA conclusions

The following conclusion may be derived from this dataset's exploratory analysis:

1. As patients/people age increases, their chance of getting a stroke increases.

2. The risk of having a stroke is increased in patients/people with higher average glucose levels.

3. We don't see more patients/people had heart disease had stroke or had hypertension had stroke.

4. Patient/ people who are married, working in private sector, and living in urban area have likelihood of getting a stroke so variable like marital status, type of employment and dwelling are statistically significant.

5. Patient/People who has higher glucose level and higher BMI are prone to get a stroke.

6. Smoking status does not affect getting a stroke.

3 Experimental Design

3.1 Identification of your chosen supervised learning algorithm(s)

As mentioned in 1.1, the target variable is stroke, and all other attributes are feature, keeping in mind the factors mentioned in 1.2 and after EDA, Decision Tree Algorithm is chosen. The decision tree will help to get the decision in sequence of steps and in logical way that makes us understand how it reached to the decision.

A Decision tree is a tree structure that looks like a flowchart, with each internal node representing a test on an attribute, each branch representing a test outcome, and each leaf node (terminal node) holding a class label which is show in figure 26 [16].

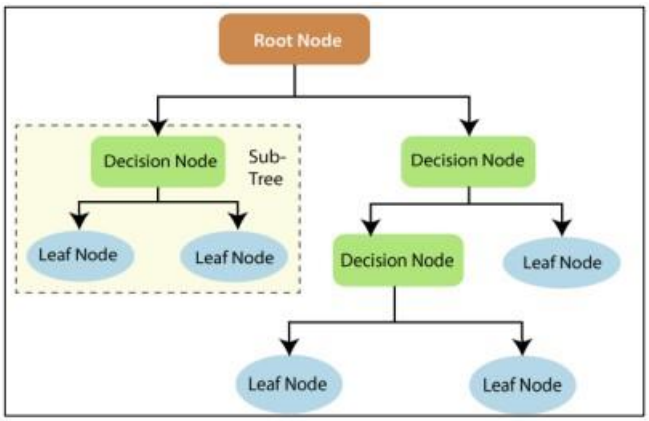


Figure 26: How decision tree works [17]

Decision trees categorise instances by sorting them along the tree from the root to some leaf node that offers the instance's categorization. As indicated in the diagram above, an instance is categorised by starting at the root node of the tree, checking the attribute given by this node, and then going down the tree branch according to the value of the attribute. This method is then repeated for the new node-rooted subtree [16].

The primary goal of the decision trees algorithm, which is a member of the family of supervised learning algorithms, is to build a training model that can be used to predict the class or value of target variables which is stroke here by learning decision rules inferred from the training data.

Its advantages make it appropriate to choose:

- Simple to understand as we can see from the figure 26
- Give a clear indication of which fields (feature variables from our stroke data) are most relevant for prediction or categorization (target variable i.e. stroke)
- Capable of handling numerical as well as categorical data - which are the only two type of data in our dataset
- It can be used for classification (refer 1.2)

Because the advantages are matched with what we want. Therefore we believe that decision tree algorithm would be the appropriate for our classification supervised learning task to build the predictive model.

3.2 Identification of appropriate evaluation techniques

We are dealing with classification and our classifier includes determining whether a patient has stroke or not as we identified in 1.2. So we will use classification report which can provide us with a bunch of information about a number of different evaluation metrics. As accuracy is only one of the metrics, we use to assess the performance of our models, and it might provide an inadequate view of your model's performance on its own as well as the data is highly imbalanced so it

might not be the only one to rely on. So with accuracy we will evaluating our model's performance using precision, recall, F1-score and support.

- The proportion of correctly predicted positive classes to all items with positive predictions is called precision.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

Figure 27: How precision is calculated

- Recall is defined as the proportion of accurately predicted positive classes to all positively categorised items:

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Figure 28: How recall is calculated

- The F1-score is a single performance statistic that considers both recall and accuracy. It is also frequently referred to as the F-Measure. It is calculated by averaging the two measures:

$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Figure 29: How F1-score is calculated

- The support is the number of occurrences of each class in y_true which is true value.

As we want to see the positive predictions, to known occasions to forecast the good outcomes, rank models and get true values, we will be choosing precision, recall, F1-score and support as our appropriate evaluation techniques.

3.3 Data cleaning and Pre-processing transformations

Data cleaning is the process of removing erroneous, damaged, badly formatted, duplicate, or missing data from a dataset. When combining several data sources, there are many possibilities for data to be duplicated or improperly categorised. We will do our data cleaning and pre-processing in three steps which can be seen in the sub-sections.

3.3.1 Dropping the unwanted column

```
df_stroke_train = df_stroke_train.drop("id",axis=1)
```

Code snippet 31: Dropping the id column from the training set

So first we will drop the unwanted id column from our training data as it not required and don't identify the sample class in any way.

Training set before dropping the id column:

```
df_stroke_train
```

Code snippet 32: Checking the training set before dropping the id column

Output:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
08	56734	Male	33.0	0	0	Yes	Govt_job	Urban	82.83	25.4	Unknown
84	25676	Female	7.0	0	0	No	children	Rural	89.38	19.0	Unknown
19	39017	Female	72.0	0	0	Yes	Govt_job	Rural	118.22	21.9	formerly smoked
43	2730	Male	58.0	0	0	Yes	Private	Urban	94.53	36.1	never smoked
1	33879	Male	42.0	0	0	Yes	Private	Rural	83.41	25.4	Unknown
.	...	...	...	...	...	...	...	...	...	...	...
77	15533	Male	46.0	0	0	No	Private	Urban	107.59	26.2	formerly smoked
92	65324	Female	48.0	0	0	Yes	Govt_job	Rural	75.91	27.8	Unknown
25	40210	Male	78.0	0	1	Yes	Self-employed	Rural	206.62	28.0	formerly smoked
36	59745	Female	27.0	0	0	Yes	Private	Urban	76.74	53.9	Unknown
95	4449	Male	48.0	0	0	Yes	Govt_job	Rural	124.64	26.4	smokes

9 rows × 12 columns

Training set after dropping the id column:

```
df_stroke_train
```

Code snippet 33: Checking the training set after dropping the id column

Output:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	Female	Male	Govt_job	Never_worked	...	Self-employed	children	Rural	Ur
1108	33.0	0	0	82.83	25.4	0	0	1	1	0	...	0	0	0	
3684	7.0	0	0	89.38	19.0	0	1	0	0	0	...	0	1	1	
3419	72.0	0	0	118.22	21.9	0	1	0	1	0	...	0	0	1	
3443	58.0	0	0	94.53	36.1	0	0	1	0	0	...	0	0	0	
31	42.0	0	0	83.41	25.4	1	0	1	0	0	...	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2777	46.0	0	0	107.59	26.2	0	0	1	0	0	...	0	0	0	
2492	48.0	0	0	75.91	27.8	0	1	0	1	0	...	0	0	1	
3625	78.0	0	1	206.62	28.0	0	0	1	0	0	...	1	0	1	
2136	27.0	0	0	76.74	53.9	0	1	0	0	0	...	0	0	0	
3595	48.0	0	0	124.64	26.4	0	0	1	1	0	...	0	0	1	

3.3.2 Encoding of nominal / categorical features

When providing data to our Machine Learning algorithms (in order to create a model), we must examine how we encode our features and targets based on the statistical data type / degree of measurement that supports it.

To that purpose, depending on whether we are dealing with "categorical data" (nominal, ordinal) or "numerical data," pandas and Scikit Learn give different ways to encode nominal, ordinal, interval, and ratio values (interval, ratio)

We will be doing dummy variable encoding. We'll take our data frame and get a binary variable encoding of a given attribute/column that allows us to represent the various categories / values for that attribute / column, replace the original column name in the dataset with our binary encoded variables, and then wrap this functionality inside a function so we can perform it on multiple columns if needed.

So from pandas we are using the method get_dummies to create dummy variable from categorical variable and drop first is useful since it helps to reduce the excess column formed during the dummy variable generation process.

```
def create_dummies(df, column_name,drop_first=False):
  # Get the binary encodings for the columns
  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)
```

Code snippet 34:

We'll use k-1 dummy variables to encode our category columns, where k is the number of nominal values in the variable.

```
df_stroke_train = create_dummies(df_stroke_train, 'gender')
```

```
df_stroke_train = create_dummies(df_stroke_train, 'work_type')
```

```
df_stroke_train = create_dummies(df_stroke_train, 'Residence_type')
```

```
df_stroke_train = create_dummies(df_stroke_train, 'smoking_status')
```

```
df_stroke_train = create_dummies(df_stroke_train, 'ever_married')
df_stroke_train.rename(columns= {"Yes" : "married"})
```

```
df_stroke_train.rename(columns= {"No" : "Not married"})
```

Code snippet 34: Encoding categorical data

Training set after encoding:

```
df_stroke_train
```

Code snippet 35: Code to get training set after encoding

age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	Female	Male	Govt_job	Never_worked	...	Self-employed	children	Rural	U
68.00	0	0	116.23	26.1	0	0	1	0	0	...	0	0	1	
64.00	0	0	75.13	31.1	0	1	0	0	0	...	0	0	1	
0.08	0	0	139.67	14.1	0	1	0	0	0	...	0	1	0	
33.00	1	0	74.44	45.2	0	1	0	0	0	...	0	0	0	
0.80	0	0	75.22	33.1	0	0	1	0	0	...	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
43.00	0	0	88.00	30.6	0	0	1	1	0	...	0	0	1	
61.00	1	0	170.05	60.2	0	1	0	0	0	...	0	0	1	
1.16	0	0	97.28	17.8	0	1	0	0	0	...	0	1	0	
80.00	0	0	196.08	31.0	0	0	1	0	0	...	1	0	1	
46.00	0	0	100.15	50.3	0	1	0	0	0	...	0	0	1	

As you can see above all categorical variables are converted into dummy variables.

3.3.3 Dealing with missing values

Many Machine Learning algorithms demand no missing values in their input data; thus, we have two ways to deal when we get a record with some missing values for particular attributes / columns.

One way would be to simply remove all records with missing values from our dataset - but this has its own set of issues because we may be throwing away vital data. Another way is to replace the missing values with a suitable replacement based on some properties of the variable under consideration. This is known as imputation.

Missing values can be imputed using a constant value or the statistics (mean, median, or most common) of each column in which the missing values are found.

First of all we are checking whether there are missing values:

```
[ ] stroke_X_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                152
smoking_status       0
dtype: int64
```

Code snippet 36 and its output: Number of missing values in specific column

And as we can see 152 values are missing in bmi so now we have to fill that missing values so that we can train our model and get the accuracy. We will be using the mean of bmi values to impute because the data distribution of BMI is symmetric which can be seen in 2.3.1 above.

We will start by defining the function and passing in the dataframe and column then we're using Sklearn's SimpleImputer algorithm to 'fit' the imputer on the specified column then we're using Sklearn's transform method to 'transform' the specified column and are returning the imputer object for future use. This imputer object holds the initial mean from the training set. Therefore, this mean can be used for the testing set. We can't create a mean from the testing set for two reasons: 1. the testing data must be kept unseen, so we can't use these values 2. we don't know how much data we have. 3. To prevent data leakage (refer 2.2) In a real-world example, there will always be more data, hence a mean can't be devised at any one time. In the next code block, we're setting bmi_imputer as the result of calling the function impute_attribute with the parameters of dataframe and column - X_train and bmi.

```
imputer = impute.SimpleImputer(missing_values=np.nan, strategy='mean')
def impute_attribute(df, column, imputer=imputer):
  imputer = imputer.fit(df[[column]])
  return imputer.transform(df[[column]]), imputer

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

Code snippet 37: Imputation of missing values by mean

After imputing we are again checking the data:

```
[ ]  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
```

Code snippet 38 and its output: Number of missing values in specific column after imputation

And as we can see there are no missing values.

3.4 Limitations and Options

The data is very unbalanced, so, we've got class imbalance but so it limits how accurate our model could be and for that we could apply SMOTE or isolation forest or one of the other techniques to balance the data such as oversampling of the data.

4 Predictive Modelling / Model Development

4.1 The predictive modelling process

To increase the accuracy and efficiency of a machine learning model we used our pre-processed data. We started by training the model using features and target:

```
# Our features
stroke_X_train = df_stroke_train.drop(labels=["stroke"], axis=1)
# Our target
stroke_Y_train = df_stroke_train["stroke"]
stroke_X_train
```

Code snippet 39: Training feature and target of the model

Here we are dropping the stroke from features because it is our target variable .

DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset. As with other classifiers, DecisionTreeClassifier takes as input two samples: stroke_X_train holding the train samples, and stroke_Y_train of integer values, holding the class labels for the training samples then fit function is used to it function adjusts weights according to data values so that better accuracy can be achieved.

So we are creating and fitting the model with maximum depth = 5 and random state = 42:

```
# create the model
stroke_clf_tree = tree.DecisionTreeClassifier(max_depth=5, random_state=42)
# ValueError: could not convert string to float: 'nontypical'
# fit the model
stroke_clf_tree = stroke_clf_tree.fit(stroke_X_train, stroke_Y_train)
```

Code snippet 39: Fitting and creating the model

Then we created a visualisation of the tree using the plot_tree fuction from sklearn:

```
plt.figure(figsize=(90,90))
tree.plot_tree(stroke_clf_tree)
plt.show()
```

Code snippet 39: Visualising the tree

The following visualisation of decision tree we get:

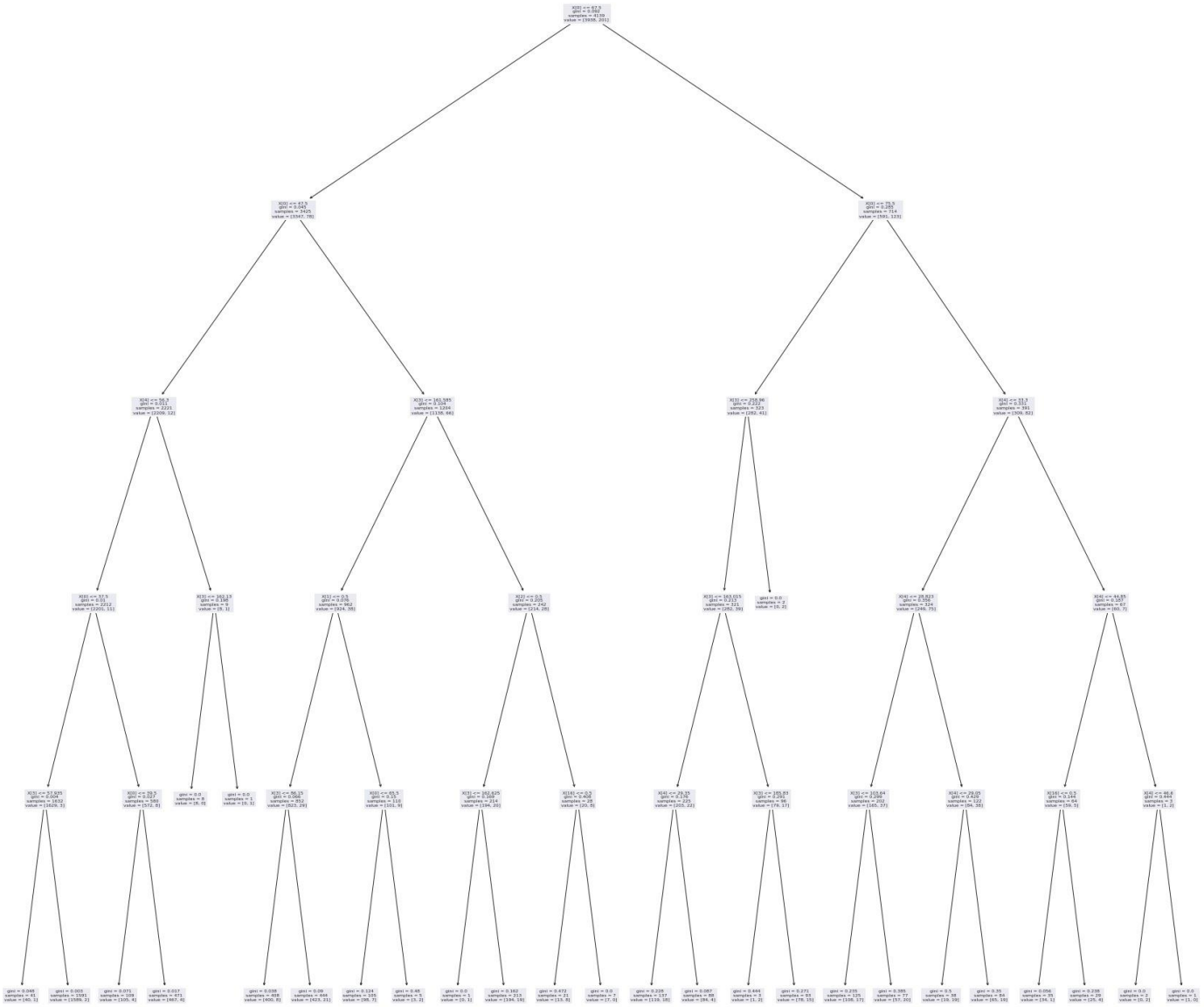


Figure 27: Decision Tree of Stroke

4.2 Evaluation results on “seen” data

After being fitted, the model can then be used to predict the class of samples:

```
[187] stroke_Y_train_tree = stroke_clf_tree.predict(stroke_X_train)
```

Confusion matrix

```
stroke_cm_train_tree = metrics.confusion_matrix(stroke_Y_train,stroke_Y_train_tree)
print(stroke_cm_train_tree)
sns.heatmap(stroke_cm_train_tree, annot=True)
```

```
[[3937    1]
 [ 193    8]]
```

Code Snippet 40 with output: Evaluation using confusion matrix of classification model

And then to get in dept evaluation we applied confusion_matirix function to evaluate classification accuracy by computing the confusion matrix with each row corresponding to the true class as we can see from the above screencapture. The confusion matrix of predictions on seen data is as follows:

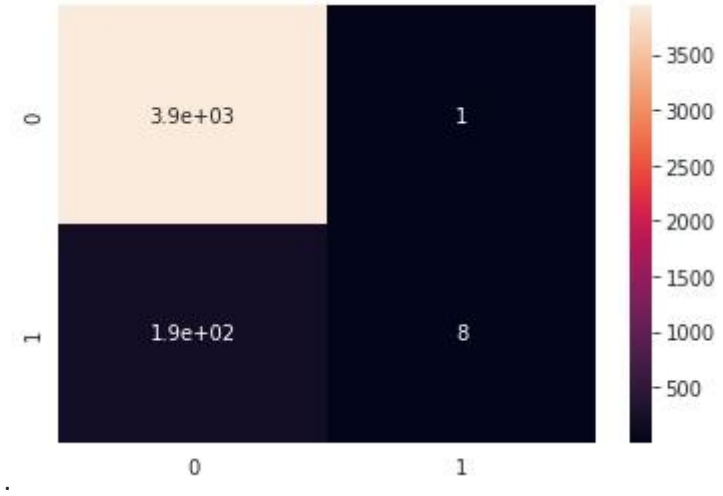


Figure 28: Confusion matrix of Stroke predictions

In the above example we can see that we appear to have 1 false positive and 193 false negatives when predicting our training set .

True Negative: 3937 out of 4139 so its good that model show people who don't have stroke and the model predicted they really don't have, that's good.

False Negative: 193 out of 4139, this show it did not perform well where 193 cases were positive and showed negative. In real world if the patient doesn't know that they are having stroke then patient might get permanent damage to their brain or even death

False Positive: 1 out of 4139, 1 patient doesn't had stroke, but the model predicted stroke which can in real world costs people money in large amount for getting a treatment for stroke and have high chance of getting permanent disability and sometime death as well.

True Positive: 8 out of 4139 , this shows 8 patients who have stroke but the model predicts the stroke so this people can minimize the long-term effect of stroke by taking immediate treatment and prevent death.

And then used the classification_report function from sklearn which builds a text report showing the main classification metrics which gave the precision recall and f1-score:

```
print("Tree classification:\n%s" % metrics.classification_report(stroke_Y_train, stroke_Y_train_tree))
```

```
Tree classification:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98      3938
           1       0.89      0.04      0.08       201

    accuracy                           0.95      4139
   macro avg       0.92      0.52      0.53      4139
weighted avg       0.95      0.95      0.93      4139
```

Code Snippet 41 with output: Evaluation using various metrics of classification model

Here we notice that our decision tree classifier seems to perform poor in terms of its training metric results in predicting stroke. It can be seen through training results, for class 1 it has an F1 score close to 0 Therefore the model has not been able to fit to the training data effectively (it didn't learn how to get a '1'). Though precision score is of 0.89 which is good.

5 Evaluation and further modelling improvements

5.1 Initial evaluation comparison

Model A: Decision Tree

```
stroke_acc_validate_tree = metrics.accuracy_score(stroke_Y_validate,stroke_Y_validate_tree)
stroke_acc_validate_tree
```

```
0.9543478260869566
```

```
print("Tree classification:\n%s" % metrics.classification_report(stroke_Y_validate, stroke_Y_validate_tree))
```

```
Tree classification:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       440
           1       0.00      0.00      0.00        20

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

Model A: Classification report of validation performed using Decision Tree Model

Model B: KNN

```
1  stroke_acc_validate_knn = metrics.accuracy_score(Y, Y_knn)
2  accuracy_score = stroke_acc_validate_knn * 100
3  print("KNN Classification Accuracy score: ", accuracy_score)
```

```
KNN Classification Accuracy score:  95.65217391304348
```

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

```
KNN Classification:
              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
```

Model B: Classification report of validation performed using KNN

KNN also performs poor in terms of its validation metrics results in predicting stroke. Model

C: Naïve Bayes

```
ComplimentNB Classifier validation report:

              precision    recall  f1-score   support

           0       0.98      0.69      0.81       440
           1       0.10      0.75      0.18        20

    accuracy                           0.69       460
   macro avg       0.54      0.72      0.49       460
weighted avg       0.95      0.69      0.78       460

Test Accuracy: 69.35%
```

Model C: Classification report of validation performed using Naïve Bayes

Model D: Support Vector Machine

```
1  svm_validate_clfrprt = classification_report(stroke_Y_validate, svm_validate_pred)
2  print(svm_validate_clfrprt)
```

```
              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

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWar
  _warn_prf(average, modifier, msg_start, len(result))
```

Model D: Classification report of validation performed using Support Vector Machine

Support Vector Machine also performs poor in terms of its validation metrics results in predicting stroke.

From the validation metrics results of Model A, B and D, we can observe that the class 1 which patient has stroke has an F1 score of 0 or close to 0 indicating that the model was unable to fit to the validation data well. And Model C performed the best with F1- score of 0.18 which is 18%.

5.2 Further modelling improvements attempted

Iteration 1: Changing Data Representation by applying SMOTE to deal with imbalanced data

```
from collections import Counter
print("Before OverSampling, counts of label '1': {}".format(sum(stroke_Y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(stroke_Y_train==0)))

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_res, Y_res = sm.fit_resample(stroke_X_train, stroke_Y_train)

print('After OverSampling, the shape of X_res: {}'.format(X_res.shape))
print('After OverSampling, the shape of Y_res: {} \n'.format(Y_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(Y_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(Y_res==0)))
print('Resampled dataset shape %s' % Counter(Y_res))
```

```
Before OverSampling, counts of label '1': 201
Before OverSampling, counts of label '0': 3938

After OverSampling, the shape of X_res: (7876, 20)
After OverSampling, the shape of Y_res: (7876,)

After OverSampling, counts of label '1': 3938
After OverSampling, counts of label '0': 3938
Resampled dataset shape Counter({0: 3938, 1: 3938})
```

Code Snippet 42: Application of SMOTE

- As we have mentioned before, we have imbalanced data and majority class (no stroke) oversamples the examples.
- So here we are applying SMOTE (Synthetic Minority Oversampling Technique) to balance the data and check the accuracy score again : generating the representative samples to overcome the overfitting problem posed by random oversampling.

Metrics classification report of seen data:

```
print("Tree classification:\n%s" % metrics.classification_report(Y_res,Y_res_tree))
```

```
Tree classification:
              precision    recall  f1-score   support

           0       0.93      0.76      0.84      3938
           1       0.80      0.94      0.86      3938

    accuracy                           0.85      7876
   macro avg       0.86      0.85      0.85      7876
weighted avg       0.86      0.85      0.85      7876
```

And we can see improvement from 0.08 to 0.86 of f1-score on predicting stroke values on training data than not applying SMOTE. Not only f1-score but we can see improvement prediction of stroke in other evaluation metrics as well such as precision and recall.

Metrics classification report of initially unseen data:

```
print("Tree classification:\n%s" % metrics.classification_report(stroke_Y_validate, stroke_Y_validate_tree))
```

```
Tree classification:
              precision    recall  f1-score   support

           0       0.99      0.78      0.87       440
           1       0.14      0.80      0.24        20

    accuracy                           0.78       460
   macro avg       0.56      0.79      0.55       460
weighted avg       0.95      0.78      0.84       460
```

And we can witness an increase in f1-score from 0.00 to 0.24 when predicting stroke values on training data compared to not using SMOTE. Not only does the f1-score increase stroke prediction, but other evaluation metrics such as precision and recall also improved.

Iteration 2: Changing Algorithm to Random Forest Classifier

```
# create the model
from sklearn.ensemble import RandomForestClassifier
stroke_clf_forest = RandomForestClassifier()
# ValueError: could not convert string to float: 'nontypical'
# fit the model
stroke_clf_forest = stroke_clf_forest.fit(stroke_X_train, stroke_Y_train)
```

Code Snippet 43: Creating and fitting random forest classifier model

Metrics classification report of seen data:

```
#We will use the classification_report function on the metrics
print("forest classification:\n%s" % metrics.classification_re
```

```
Tree classification:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3938
           1       1.00      1.00      1.00       201

    accuracy                           1.00      4139
   macro avg       1.00      1.00      1.00      4139
weighted avg       1.00      1.00      1.00      4139
```

Here we can see the model memorised that value, that's why it might be giving the accuracy 100% and the data is highly imbalance so we cannot believe that the accuracy can be 100%. Therefore random forest classifier did not improved our initial model.

Metrics classification report of initially unseen data:

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       440
           1       0.00      0.00      0.00        20

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

It appears to perform poorly in terms of verifying metric results in predicting stroke for previously unreported data. It is clear from confirming findings that class 1 has an F1 score of 0. As a result, the model was unable to fit the validation data properly and was no better than our initial model.

6 Conclusion

Initially the model had not had good result on seen data i.e. it did not predicted the stroke (refer Code Snippet 41) and because that happened on trained data it remained in evaluation of initially seen (refer model A of 5.1) and unseen data as well. But while doing further improvement by applying SMOTE (refer Iteration 1 in 5.2) its evaluation results improved on seen data and initially unseen data (refer Iteration 1 in 5.2) as well but it again dropped and gave more bad results that the first iteration on both seen and unseen data while applying new algorithm (Code Snippet 43). So the final model is the iteration 1 in 5.2 which was application of SMOTE and it improved the f1-score of 0.81 on seen data and 0.24 on initially unseen data.

6.2 Suggested further improvements to the model development process

Model Training seemed to go fairly smooth compared to other parts of model development.

If I go through similar process again, I will first try to get more insight by doing EDA and then try to collect more data and get more background information about the data set such as unknown values in smoking status of our stroke data set. Use stratified split while splitting when dealing with imbalanced classification problems. I will use oversampling before split to try to deal with never worked values in work type variable or put them from validation set into test set for class representation and to get more insight on that and whether it affects the stroke or not. I will add bins while creating plots to get more clear insights. And will create the source code on github. And as a group we could use gantchart in the future to do more insightful research in timely manner

6.3 Reflection on Research Team

Firstly by working in research team we got the opportunity to gain experience from each other. We descriptively discussed the process of doing the research step by step and asking for guidance from each other and by being in different course one in computer science and other three in computer and data science, we were able to get guidance from wider range of experienced professionals when taking feedback and tackling problems as we were immediately discussing either writing descriptive about what the tutors suggested about tackling a particular problem or arranging a quick meeting so we can see everyone's agreement. Overall there was effective communication, collaboration readiness and regular and open discussion as well as effective brainstorming.

6.4 Reflection on Individual Learning

We individually took advantage of support sessions available during the module to get feedback and support if we were stuck on any part of the coursework and to effectively draft the report from the feedback give to us after the session. Based on the support sessions we improved the code and removed unwanted stuff and did more EDA and tried to understand why we need certain methods to improve our result individually.

The insights gained through this module were how we can do better for world using data, creating model and save one life, one business.

The insights gained through the process of trying the course work were how I can create amazing model by refining the accuracy through number of iterations on data, algorithm, and parameters. And how model made through dataset of specific region can only predict things of that regions like here we are predicting stroke from the data collected from different medical centres in Bangladesh so it can predict stroke of that region only, it cannot predict stroke of China due to its climate, lifestyle etc.

7 References and Bibliography

1. Boehme, A.K., Esenwa, C. and Elkind, M.S.V. (2017). Stroke Risk Factors, Genetics, and Prevention. *Circulation research*, [online] 120(3), pp.472–495. doi:10.1161/CIRCRESAHA.116.308398.

2. Emon, M.U., Keya, M.S., Meghla, T.I., Rahman, Md.M., Mamun, M.S.A. and Kaiser, M.S. (2020). Performance Analysis of Machine Learning Approaches in Stroke Prediction. [online] IEEE Xplore. doi:10.1109/ICECA49313.2020.9297525.

3. Lane, S. (2018). A good study starts with a clearly defined question. BJOG: An International Journal of Obstetrics & Gynaecology, [online] 125(9), pp.1057–1057. doi:10.1111/1471-0528.15196.

4. Kelly-Hayes, M. (2010). Influence of Age and Health Behaviors on Stroke Risk: Lessons from Longitudinal Studies. Journal of the American Geriatrics Society, [online] 58(2), pp.S325–S328. doi:10.1111/j.1532-5415.2010.02915.x.
5. Centers for Disease Control and Prevention. (2019). Stroke Risk. [online] Available at: https://www.cdc.gov/stroke/risk_factors.htm.

6. Michael, K.M. and Shaughnessy, M. (2006). Stroke Prevention and Management in Older Adults. The Journal of Cardiovascular Nursing, 21, pp.S21–S26. doi:10.1097/00005082-200609001-00006.

7. Cleveland Clinic. (2021). Diabetes and Stroke: Causes, Symptoms, Treatment & Prevention. [online] Available at: https://my.clevelandclinic.org/health/diseases/9812-diabetes-and-stroke.

8. Mayo Clinic (2022). How high blood pressure can affect your body. [online] Mayo Clinic. Available at: https://www.mayoclinic.org/diseases-conditions/high-blood-pressure/in-depth/high-blood-pressure/art20045868.

9. Bhui, K., Dinos, S., Galant-Miecznikowska, M., de Jongh, B. and Stansfeld, S. (2016). Perceptions of Work Stress Causes and Effective Interventions in Employees Working in public, Private and non-governmental organisations: a Qualitative Study. BJPsych Bulletin, [online] 40(6), pp.318–325. doi:10.1192/pb.bp.115.050823.

10. https://www.helpguide.org. (n.d.). Adjusting to Retirement: Handling the Stress and Anxiety - HelpGuide.org. [online] Available at: https://www.helpguide.org/articles/aging-issues/adjusting-to-retirement.htm.

11. Communicating Diet and Activity Research. (2015). Job-loss associated with weight gain - but diet and physical activity may not be the reason. [online] Available at: https://www.cedar.iph.cam.ac.uk/2015/09/21/job-lossassociated-weight-gain/.

12. PRB (2001). Around the Globe, Women Outlive Men. [online] PRB. Available at: https://www.prb.org/resources/around-the-globe-women-outlive-men/.

13. Data Science Stack Exchange. (n.d.). classification - Is it right to impute Train and Test set? [online] Available at: https://datascience.stackexchange.com/questions/52282/is-it-right-to-impute-train-and-test-set [Accessed 16 Jan. 2023].

14. scikit-learn.org. (n.d.). Glossary of Common Terms and API Elements — scikit-learn 0.24.2 documentation. [online] Available at: https://scikit-learn.org/stable/glossary.html#term-random_state.

15. www.ninds.nih.gov. (n.d.). Brain Basics: Preventing Stroke | National Institute of Neurological Disorders and Stroke. [online] Available at: https://www.ninds.nih.gov/health-information/public-education/brainbasics/brain-basics-preventing-stroke#:~:text=Some%20ways%20that%20work%3A%20Maintain.

16. GeeksforGeeks. (2017). Decision Tree - GeeksforGeeks. [online] Available at:

https://www.geeksforgeeks.org/decision-tree/.

17. Charbuty, B. and Abdulazeez, A. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. Journal of Applied Science and Technology Trends, 2(01), pp.20–28. doi:10.38094/jastt20165.