INFOSYS 722 Data Mining and Big Data

Iteration 4

Semester 2, 2022

Qiong Zhou / 365217677 / qzho906

GitHub Link:
https://github.com/Anbyya/Infosys722_iteration4

Table of Contents

1. Business / Situation Understanding

1.1 Business understanding Overview

Ischemic heart disease, stroke, and chronic obstructive pulmonary disease are the top three causes of death from disease, accounting for 16%, 11%, and 6 % of deaths worldwide respectively (WHO, 2020).

Stroke is a condition in which the blood supply to the brain is interrupted, resulting in a lack of oxygen, brain damage, and loss of function. Stroke can lead to permanent damage, including partial paralysis and impairments in speech, understanding, and memory, all of which affect the type and severity of disability depending on the part of the brain affected and the length of time the blood supply is stopped (World Stroke Organization, 2022). The prevalence of stroke has reached epidemic proportions beyond what is thought possible. Globally, one in four adults over the age of 25 will have a stroke in their lifetime (World Stroke Organization, 2022). This year 12.2 million people worldwide will have their first stroke and 6.5 million will die as a result. Worldwide, more than 110 million people have experienced a stroke (World Stroke Organization, 2022). The incidence of stroke increases significantly with age, but over 60% of strokes occur in people under the age of 70 and 16% in people under the age of 50 (World Stroke Organization, 2022).

Although stroke is an acute cerebrovascular disease with high morbidity, mortality, and disability rate. Many factors contribute to stroke, including age, race, gender, geography, and environment, which are not controllable. However, according to a review article published in the Journal of the American College of Cardiology, 90% of strokes are preventable and the key is to manage and treat controllable risk factors (Kleindorfer DO, Towfighi A, Chaturvedi S, et al., 2021). The controllable risk factors are blood pressure, blood lipids, blood sugar, and lifestyle (smoking, alcohol, etc.).

1.2 Determining Business Objectives

Although the disease itself can develop in a very short period of time and without any precursors, it is still possible to classify and predict patients based on their early personal history and to offer solutions and advice to patients while reducing the chances of a stroke. One of the main clinical risk factors for stroke is atherosclerosis-induced hypertension, also, there are many other risk factors including smoking, physical inactivity, unhealthy diet, harmful use of alcohol, atrial fibrillation, elevated blood lipid levels, obesity, genetic predisposition, stress, and depression (World Stroke Organization, 2022).

In this case, the study was commissioned with the following data business objectives:

- Be able to predict, prevent and reduce the occurrence of stroke disease based on the patient's current status.

The expected outcome of the study:

- Be able to provide current patients with detailed information on the causes of their current stroke in terms of current health indicators, age, and life status.
- Reduces the likelihood of a potential patient having a stroke at an early stage.

1.3 Accessing the Situation

In order to achieve the goal of being able to analyze in detail the causes of stroke in stroke patients and to detect early manifestations of signs of stroke. This study should review a large number of existing patient cases and engage them as a dataset to find models, as well as relationships between signs, data mining specialists should be applied.

Personnel. It is clear and confident that it is important to consult doctors and specialists in the field of stroke research, as well as those working in the field of healthcare and rehabilitation for stroke patients, and to obtain clear and detailed information about the disease itself, including the primary, secondary and direct causes of stroke. Regarding the data and information collected from these stroke specialists, a database specialist should be consulted. Since these specialists hope the results of the study will become part of a continuing studying and research process, data warehousing and data cleaning for analysis are required. Also, the information about the patients involves their privacy, the data management must also be considered.

Data. The data required for the study analysis will come primarily from the records of doctors and healthcare professionals, where detailed desensitized data on patients is recorded. For initial studies, the data can be easily found on the internet, furthermore, the study data can be expanded as the study is conducted.

Risks. As the project is primarily concerned with healthcare, the accuracy of the model depends to a large extent on the quality and quantity of the dataset used for the study. The model production process should also be concerned with the risk of privacy breaches to users. Once generated, the models should be further validated by a team of experts who specialize in the treatment and research of stroke patients.

1.4 Data Mining Objectives

Experts in data mining contributed by translating the project's business objectives into data mining objectives. All data mining objectives studied in the initial phase are listed below.
In this case, the study was commissioned with the following data mining objectives:

- Identify a clear relationship between the patient's current health Index (current illness and BMI) and the stroke.
- Find out the relationship between the patient's age and the chances of stroke.
- Find the relationship between the patient's current life status (smoking status, marital status, work type, living environment, etc.) and the stroke.
- Reduce the chances of stroke by providing predictions based on the patient's current illness and age.
- Use this data to train and fit the model to make it suitable for use in data prediction.
  The validation phase uses a selection of healthcare records as a validation process to see the prediction results of a given model, and if the model successfully passes a pre-determined threshold, validation by healthcare professionals is continued.
  The initial use of the model phase allows the healthcare professional to enter a new patient's medical history, view the patient's stroke risk rating, and have the doctor diagnose to see if the system passes the threshold.
  The enhancement and optimization phase should continue to collect the required data and retrain and improve the model to increase accuracy.

The expected outcome of the study:

- Provide detailed information on the causes of the stroke to current patients.
- Provide useful information and advice to people who are at risk of stroke.
- Reduce the likelihood of stroke in potential patients at an early stage.

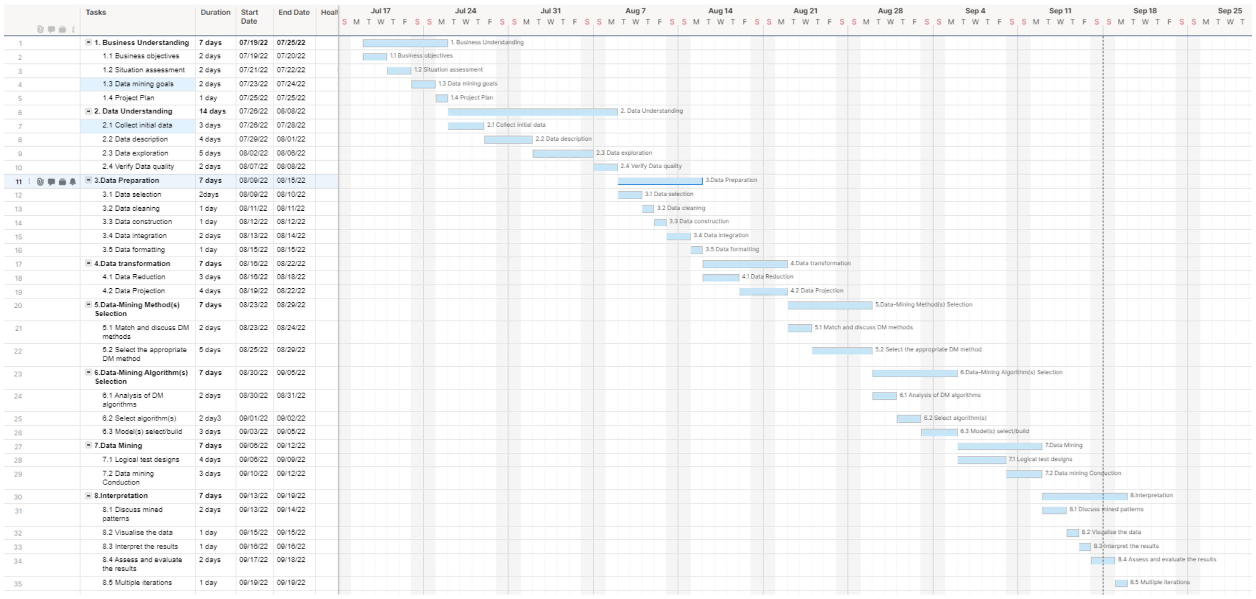
If the above objectives are achieved, the system should provide the ability to diagnose the patient's risk level.

1.5 Project Plan

The entire project is planned to take ten weeks of work and is planned as follows, also the detailed project schedule is shown in Gantt Chart in Figure 1.

Phase	**Time**	**Resources**	**Risks**
Business Understanding - Business background research - Business case study	1 week	Analysts	- Project change - Find the unsuitable case to study
Data Understanding - Dataset selection - Data exploration	2 weeks	Analysts	- Data problems - Attribute selections in the dataset
Data preparation - Data selection - Data cleaning - Data construction - Data integration - Data formatting	2 weeks	Data mining experts	- Technical problems
Modeling - Train the model - Evaluation of the performance - improve the performance	2 weeks	Database analyst time	- Model fitting problems - Technical problems
Validation - Validate the model by test sets	1 week	Analysts and tests	- Incompatible model - Incompatible test sets - Technical problems
Evaluation	1 week	Analysts	- Technical problems
Deployment	1 week	- Data mining consultant - Database analyst time	- Inability to deploy the whole project

Table 1. Project Plan



	Tasks	Duration	Start Date	End Date	Heal
1	**1. Business Understanding**	**7 days**	**07/19/22**	**07/25/22**	
2	1.1 Business objectives	2 days	07/19/22	07/20/22	
3	1.2 Situation assessment	2 days	07/21/22	07/22/22	
4	1.3 Data mining goals	2 days	07/23/22	07/24/22	
5	1.4 Project Plan	1 day	07/25/22	07/25/22	
6	**2. Data Understanding**	**14 days**	07/26/22	08/08/22	
7	2.1 Collect initial data	3 days	07/26/22	07/28/22	
8	2.2 Data description	4 days	07/29/22	08/01/22	
9	2.3 Data exploration	5 days	08/02/22	08/06/22	
10	2.4 Verify Data quality	2 days	08/07/22	08/08/22	
11	**3.Data Preparation**	**7 days**	08/09/22	08/15/22	
12	3.1 Data selection	2days	08/09/22	08/10/22	
13	3.2 Data cleaning	1 day	08/11/22	08/11/22	
14	3.3 Data construction	1 day	08/12/22	08/12/22	
15	3.4 Data integration	2 days	08/13/22	08/14/22	
16	3.5 Data formatting	1 day	08/15/22	08/15/22	
17	**4.Data transformation**	**7 days**	08/16/22	08/22/22	
18	4.1 Data Reduction	3 days	08/16/22	08/18/22	
19	4.2 Data Projection	4 days	08/19/22	08/22/22	
20	**5.Data-Mining Method(s) Selection**	**7 days**	08/23/22	08/29/22	
21	5.1 Match and discuss DM methods	2 days	08/23/22	08/24/22	
22	5.2 Select the appropriate DM method	5 days	08/25/22	08/29/22	
23	**6.Data-Mining Algorithm(s) Selection**	**7 days**	08/30/22	09/05/22	
24	6.1 Analysis of DM algorithms	2 days	08/30/22	08/31/22	
25	6.2 Select algorithm(s)	2 day3	09/01/22	09/02/22	
26	6.3 Model(s) select/build	3 days	09/03/22	09/05/22	
27	**7.Data Mining**	**7 days**	09/06/22	09/12/22	
28	7.1 Logical test designs	4 days	09/06/22	09/09/22	
29	7.2 Data mining Conduction	3 days	09/10/22	09/12/22	
30	**8.Interpretation**	**7 days**	09/13/22	09/19/22	
31	8.1 Discuss mined patterns	2 days	09/13/22	09/14/22	
32	8.2 Visualise the data	1 day	09/15/22	09/15/22	
33	8.3 Interpret the results	1 day	09/16/22	09/16/22	
34	8.4 Assess and evaluate the results	2 days	09/17/22	09/18/22	
35	8.5 Multiple iterations	1 day	09/19/22	09/19/22	

Figure 1. Gantt Chart of project schedule

2. Data Understanding

2.1 Data Understanding Overview

The healthcare information document is collected from the Internet, Kaggle. The data used will be accessed from a website link https://www.kaggle.com/datasets/lirilkumaramal/heart-stroke. The API command is kaggle datasets download -d lirilkumaramal/heart-stroke. It was last updated on 14 August 2021 (Amal, 2020).

2.2 Collecting Initial Data

The detailed information covered in the dataset is as below:

- **Personal information.** The raw data contain basic information about individuals but does not relate to the disease itself, which includes age and gender.
- **Health Index.** The raw data contain a survey of diseases and basic physical indicators. Diseases investigated include the presence of hypertension and heart disease. The underlying physical indicators include mean glucose levels and BMI.
- **Life status.** The raw data contains living status and habits, whether married or not, type of work, type of residence, and smoking status.

In terms of data quality assumptions, firstly, the dataset is a desensitization dataset, as obviously the dataset did not include sensitive data, for example, name, address, postal code, and so on. Secondly, the dataset is rich enough for training a model for stroke prediction. Thirdly, the dataset is correctly collected from stroke patients or potential patients.

The main problem with this dataset is that it provides limited information about the origin of the dataset itself and only states that this dataset is widely used. This makes the source of this dataset, and in which region it was collected and generated, ambiguous in the study. Once the dataset has been fabricated, all of the findings and predictions in this article will be meaningless. After searching through the discussion section of this dataset, it was found that the data was actually collected by the clinic and the source of the data was deliberately removed due to privacy concerns. After confirming the authenticity of the data sources, the reliability of this dataset was verified and approved for data mining.

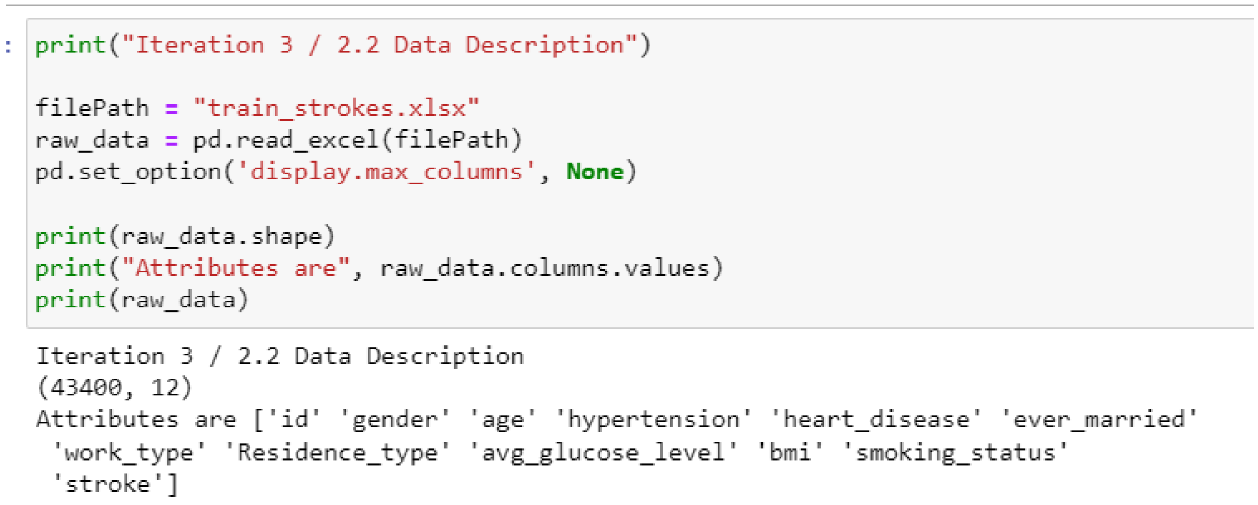
2.3 Describing Data

Data Quantity

The format of the dataset has been packed as a .csv file, it can be easily accessed and converted to the desired format and used, for example, as a .xlsx file. The data mining process can be done using SPSS Modeler, and it is readable for an application like python, spark, and Weka. The dataset contains 12 attributes and 43,400 records, in which, each record contains the information of individuals according to these 12 attributes, as shown below in Figure 2.

In this section, we will use Python Pandas to look at and uncover the overall quality and characteristics of the entire dataset and to visualize and analyze the raw data.
In the next section, we will use Pyspark to perform data cleaning and model building.

We use the Python Pandas framework and Jupyter Notebook to analyze data. Pandas is a Python library for working with datasets (W3Schools, 2022). It has functions for analyzing, cleaning, exploring, and manipulating data. Pandas enable us to analyze big data and make conclusions based on statistical theory. Pandas can clean up messy datasets and make them readable and relevant. And in data science, relevant data is very important. Jupyter Notebook is the original web application for creating and sharing computing documents (Jupyter, 2022). It provides a simple, streamlined, document-centric experience. In which, we can better express the code in a documentation style.

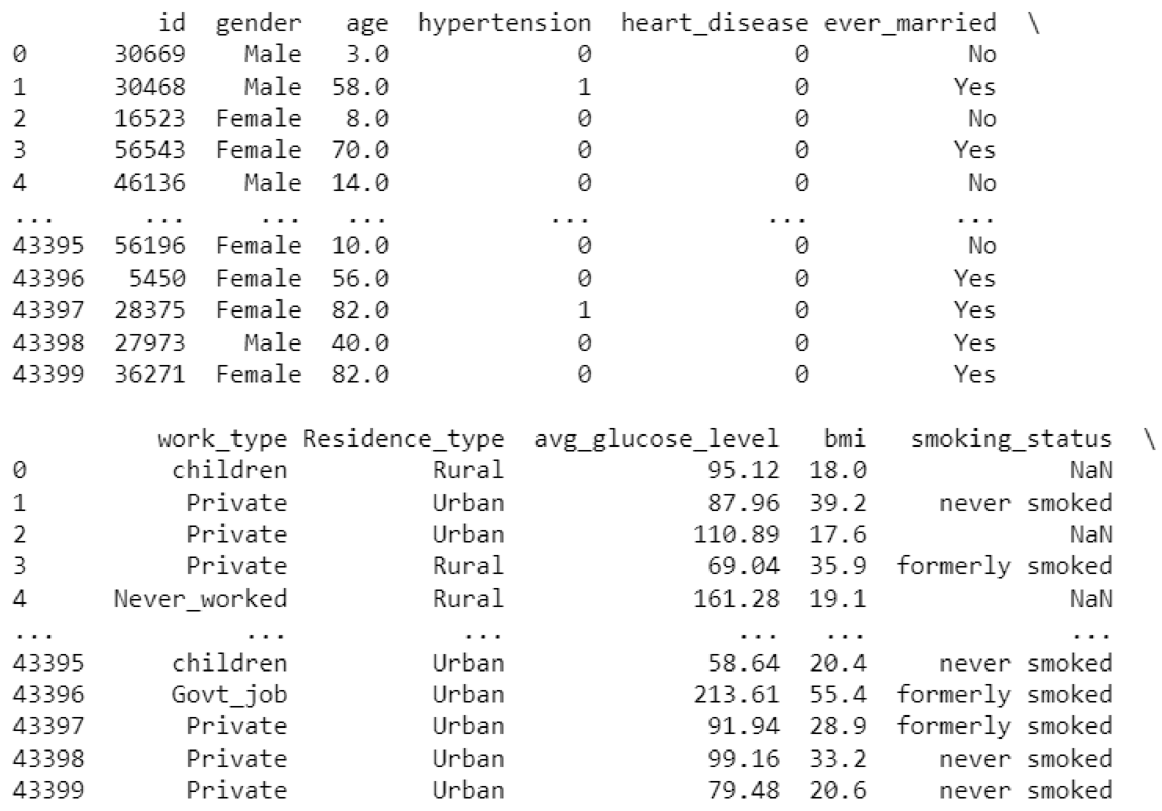


```
: print("Iteration 3 / 2.2 Data Description")

  filePath = "train_strokes.xlsx"
  raw_data = pd.read_excel(filePath)
  pd.set_option('display.max_columns', None)

  print(raw_data.shape)
  print("Attributes are", raw_data.columns.values)
  print(raw_data)
```

```
Iteration 3 / 2.2 Data Description
(43400, 12)
Attributes are ['id' 'gender' 'age' 'hypertension' 'heart_disease' 'ever_married'
 'work_type' 'Residence_type' 'avg_glucose_level' 'bmi' 'smoking_status'
 'stroke']
```

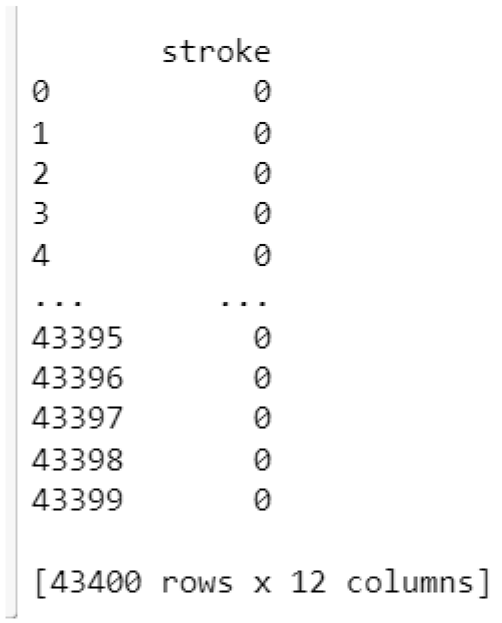
Figure 2. overview of the dataset using Python Pandas

The raw data is shown in Figure 3 & Figure 4.



```
          id  gender   age  hypertension  heart_disease ever_married  \
0      30669    Male   3.0             0              0           No
1      30468    Male  58.0             1              0          Yes
2      16523  Female   8.0             0              0           No
3      56543  Female  70.0             0              0          Yes
4      46136    Male  14.0             0              0           No
...      ...     ...   ...           ...            ...          ...
43395  56196  Female  10.0             0              0           No
43396   5450  Female  56.0             0              0          Yes
43397  28375  Female  82.0             1              0          Yes
43398  27973    Male  40.0             0              0          Yes
43399  36271  Female  82.0             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0          children          Rural              95.12  18.0              NaN
1           Private          Urban              87.96  39.2     never smoked
2           Private          Urban             110.89  17.6              NaN
3           Private          Rural              69.04  35.9  formerly smoked
4      Never_worked          Rural             161.28  19.1              NaN
...             ...            ...                ...   ...              ...
43395      children          Urban              58.64  20.4     never smoked
43396      Govt_job          Urban             213.61  55.4  formerly smoked
43397       Private          Urban              91.94  28.9  formerly smoked
43398       Private          Urban              99.16  33.2     never smoked
43399       Private          Urban              79.48  20.6     never smoked
```

Figure 3. Initial Data Overview 1



```
       stroke
0           0
1           0
2           0
3           0
4           0
...       ...
43395       0
43396       0
43397       0
43398       0
43399       0

[43400 rows x 12 columns]
```

Figure 4. Initial Data Overview 2

Data Features

The objective of this project is to find the relationship between given information and the chance of a stroke occurring. The information collected in the dataset, such as age, BMI, glucose, hypertension, history of heart disease, etc., were categorized early on as risk factors for stroke. The dataset is combining multiple formats of coding, the table 2 is showing the data type as well as the note of each attribute. Among these 12 attributes, 4 are processed as numeric data, 4 are processed as flags, and the rest 4 are processed as categories. These attributes provide sufficient information related to stroke. Therefore, based on medical research, it can be said that the information contained in the dataset has a strong relationship with stroke (Lim, 2021).

The dataset provides a realistic picture of the individual, as each person is flagged stroke status and other related attributes as shown below.

Attribute	Value Type	Usage	Example
ID	Numeric	Unique Identifier	30468
Gender	Categorical (string)	Male, Female, Other	Male
Age	Numeric	Continues number	58
Hypertension	Boolean (Flag)	0 = No hypertension before; 1 = Yes	1 (Yes)
Heart_disease	Boolean (Flag)	0 = No heart disease before; 1 = Yes	0 (No)
Ever_married	Boolean (Flag)	0 = Never married; 1 = Married	Yes
Work_type	Categorical (string)	Type of work the individual has	Private
Residence_type	Categorical (string)	Rural/Urban	Urban
Avg_glucose_level	Numeric (Decimal)	The average glucose level of an individual	87.96
BMI	Numeric (Decimal)	The body mass index of an individual	39.2
Smoking_status	Categorical (string)	Formerly smoked/ smokes/never smoke	never smoked
Stroke	Boolean (Flag)	0 = No stoke happened before; 1 = Yes	0

Table 2. Attribute and Value types of Analysis

Inside this dataset, 41% of samples were recorded from male patients, and 59% of samples were from females. The average age of all patients was 42.2±22.5 with an average BMI of 28.6-±7.77, 64% of patients were ever married. 57% of patients worked for private companies, and

16% were self-employed. 50% of them lived in a rural place while others were urban. 90.64% of patients had a record of hypertension, and 95.25% of all patients were diagnosed with heart diseases and stroked happened in 98.20% of patients. The average glucose level of all patients was 104±43.1. In the records of smoking status for patients, 37% of them never smoked, the rest of them either did provide any records or were grouped into others.
The business case for this project is to find the relationship between given information and the chance of having a stroke. The information collected in the dataset, such as age, BMI, glucose, hypertension, and history of heart disease, was categorized in advance as risk factors for stroke. Therefore, based on medical research, it can be said that the information contained in the dataset has a strong relationship with stroke.
The data most relevant to the final outcome could be prioritized, with the medical information component supposed to be the most relevant and lifestyle habits coming second. Personal information is listed last, which has a relatively small and negligible impact on stroke.
In summary, the information contained in the dataset is useful and appears to be relevant to stroke, although it contains multiple errors, and the number of labels is uneven. However, these issues can be easily addressed in the subsequent data cleaning process prior to application to the data mining process.
At the current stage, the preference attributes associated with stroke are not known. Further exploration of the dataset is required.

2.4 Exploring Data

After observing the relationship between different attributes with target attribute stroke by pairplot using the python seaborn package, as shown in Figure 5. With these attributes, we do not consider with the relationship between id and stroke, as it is obvious that ID is not an influential attribute toward stroke.

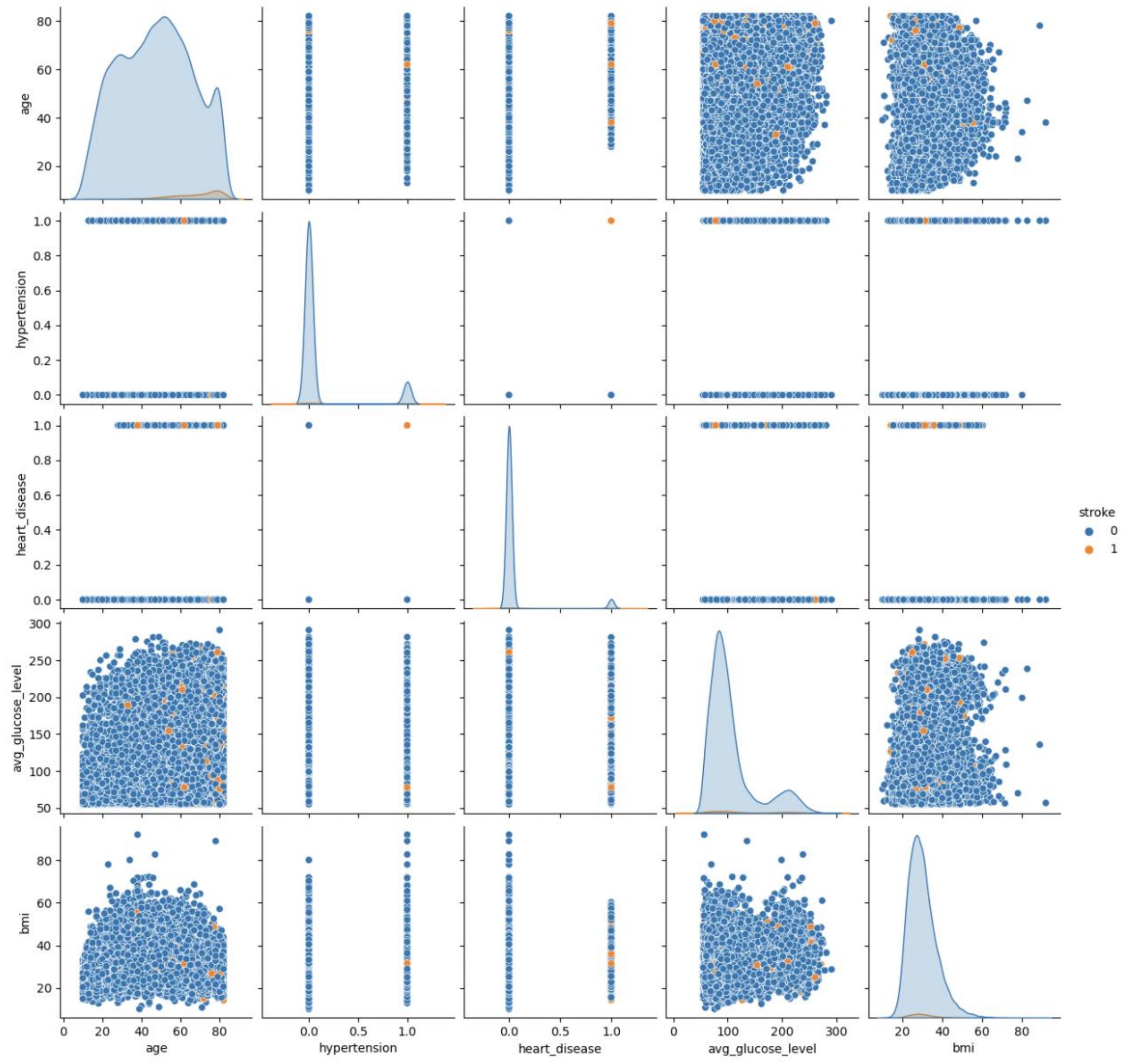


Figure 5. The pairplot between attributes

Then we take the first glance of the relationship between age, hypertension, heart disease, average glucose and BMI with target attribute stroke respectively.
From Figure 5, we can observe that

1. The intuitive hypothesis is that strokes are more likely to occur in older adults. This hypothesis was verified by the present dataset. The number of stroke cases among all different age patients were presented. A larger count circle indicates a higher records of stroke cases. Most of stroke cases were happened when age (>40) years old. And there are most stroke cases happened around 80 years old patients.
2. Patients without underlying conditions of hypertension are less likely to suffer a stroke.
3. Patients without underlying heart disease are less likely to have a stroke, and patients are more likely to have a stroke underlying heart disease.

4. The number of stroke cases among all different glucose level (56 - 260) patients were presented, but glucose level from 55 to 116 has most patients who did not get stroke before. And the patients who has glucose level higher than 271 did not get stroke in this dataset.
5. The number of stroke cases were analyzed in patients with different BMI. From the statistical result, patients with from 14.3 to 56.6 showed more cases of a stroke happening, while patients with a lower BMI (<15) or higher BMI (>55) showed less cases of stroke.

Figure 5 shows the predictors that are important for the wind in the target attributes and also shows the relationship between each attribute. The figure shows that age, hypertension, heart disease, blood glucose, and BMI all have a direct and observable effect on stroke. Of these, age is the single most important risk factor for stroke, and as stated in the figure, older people have a greater chance of having a stroke. Hypertension and heart disease and other such underlying conditions are also important predictors for stroke, while a clear relationship between blood glucose and BMI for stroke is difficult to observe at this stage and further exploration is needed.

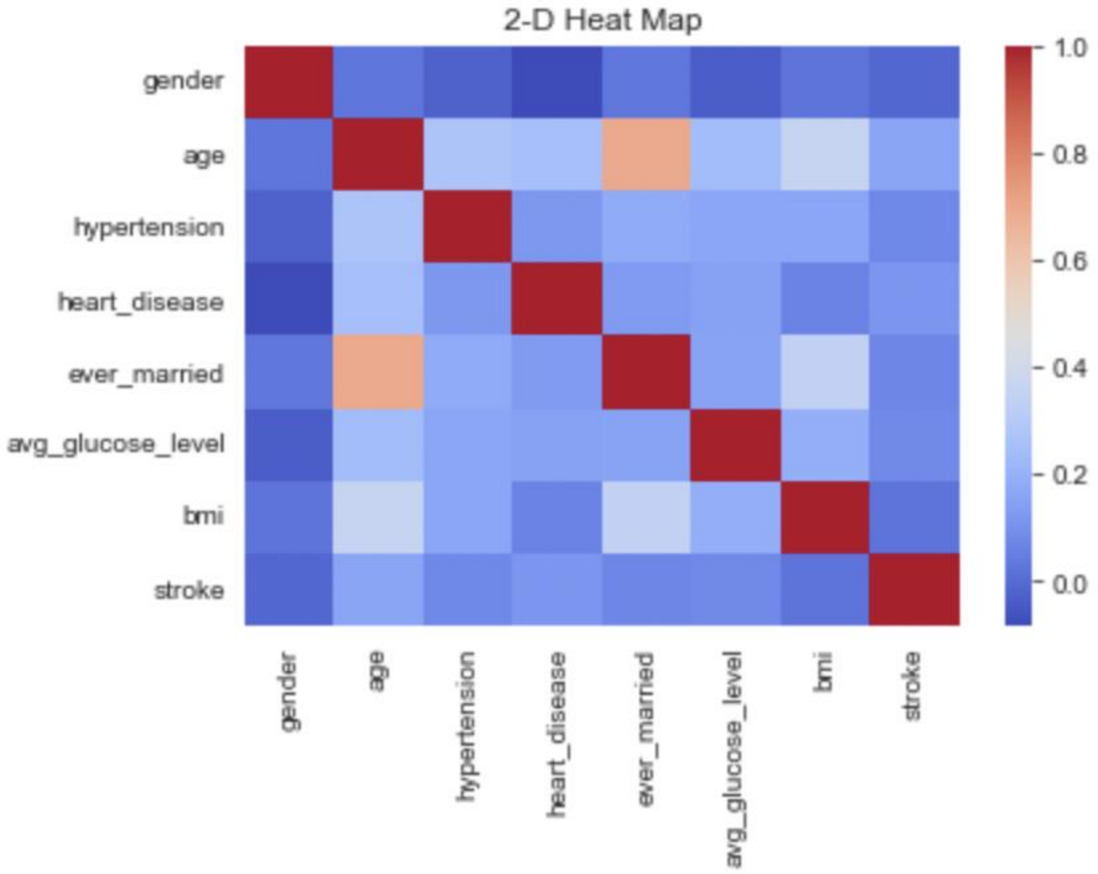


Figure 6: The correlation heatmap between attributes

Figure 6 shows the heat map of correlations between the attributes of the dataset, the most influential attribute is the age of the individual as shown in Figure 7 below, followed by history of heart disease, average blood glucose level and hypertension status, the least relevant attribute is the type of residence, which can be considered as the least important attribute.

```
: # Draw age and stroke boxplot
  seaborn.boxplot(x=raw_data['stroke'],y=raw_data['age'])

: <AxesSubplot: xlabel='stroke', ylabel='age'>
```

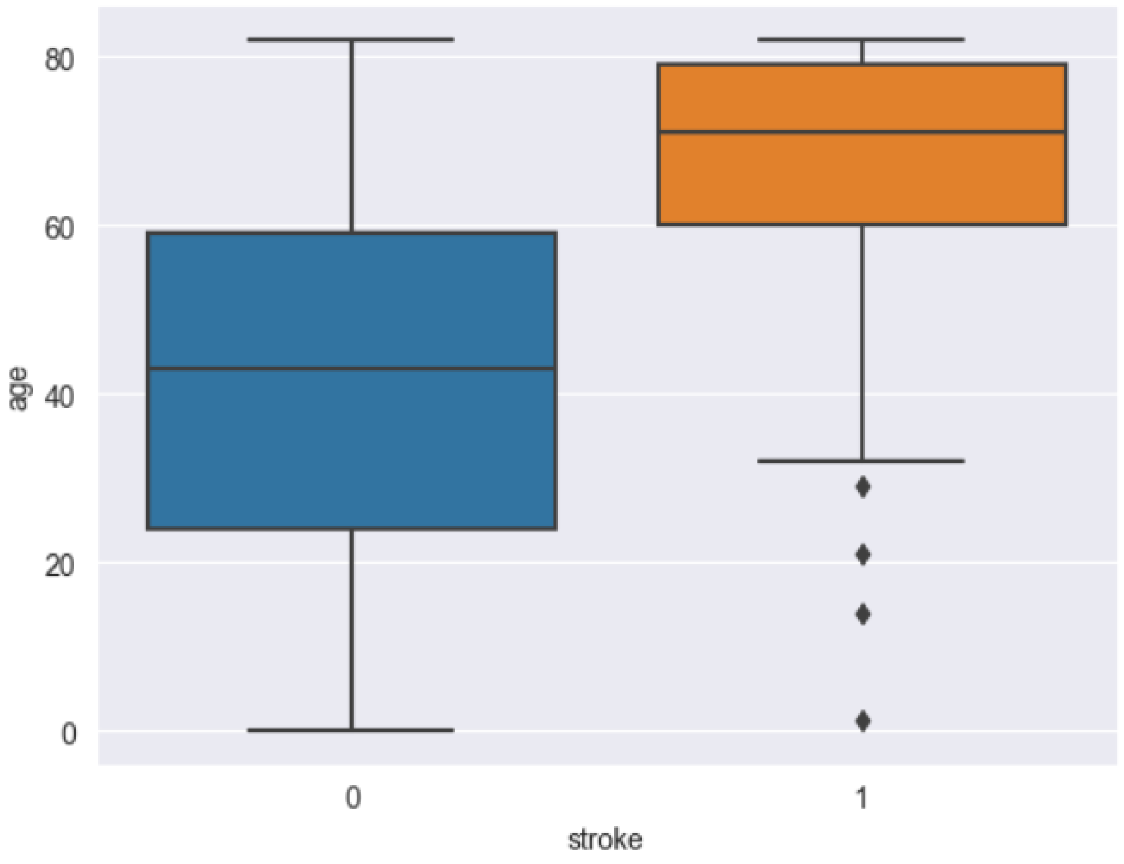


Figure 7. Correlation of age and stroke

In the next data cleaning process, we will filter and remove some less relevant attributes, such as the type of work mentioned earlier. And keeping the goal of data mining will remain the same, which is to discover the relationship between the attributes and the final result. In addition, sensitive data will be processed, such as extreme values. another aspect, it is necessary to use a subset of instances. Too much difference between the two labels can affect the final accuracy. Therefore, certain instances and attributes need to be selected to eliminate potential waste of time and processing costs. And for this amount of data, we need to perform further filtering.

Furthermore, we can see from Figure 8, the stroke cases are extremely imbalanced, only 783 (1.8%) patients got a stroke, and the other 42,617 (98.2%) patients did not get stroke. Though the relationship between the different attributes is not yet clear, the attributes gender, age, health index (current illness: hypertension, heart disease, average glucose level; BMI) and work type need more investigation as they may not be removed due to the imbalance in stroke cases.

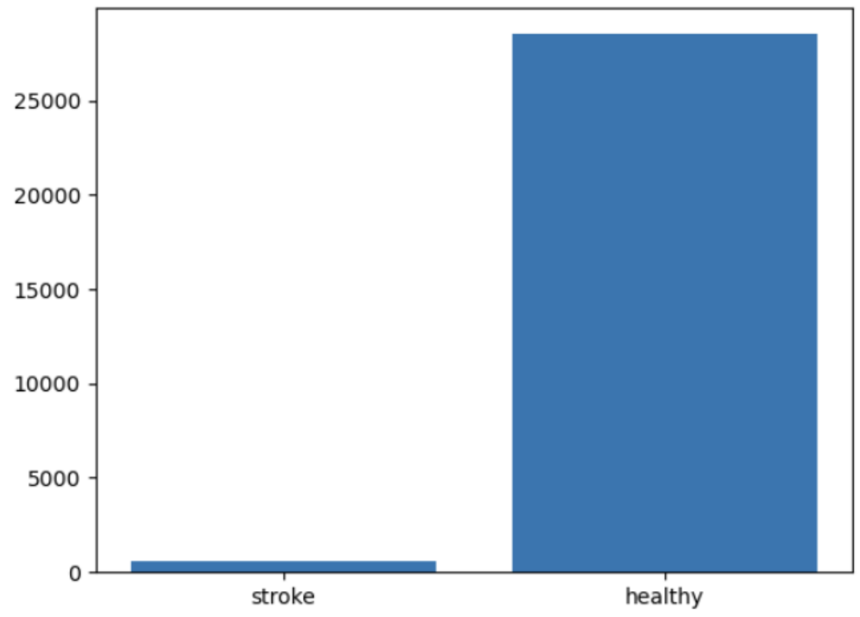
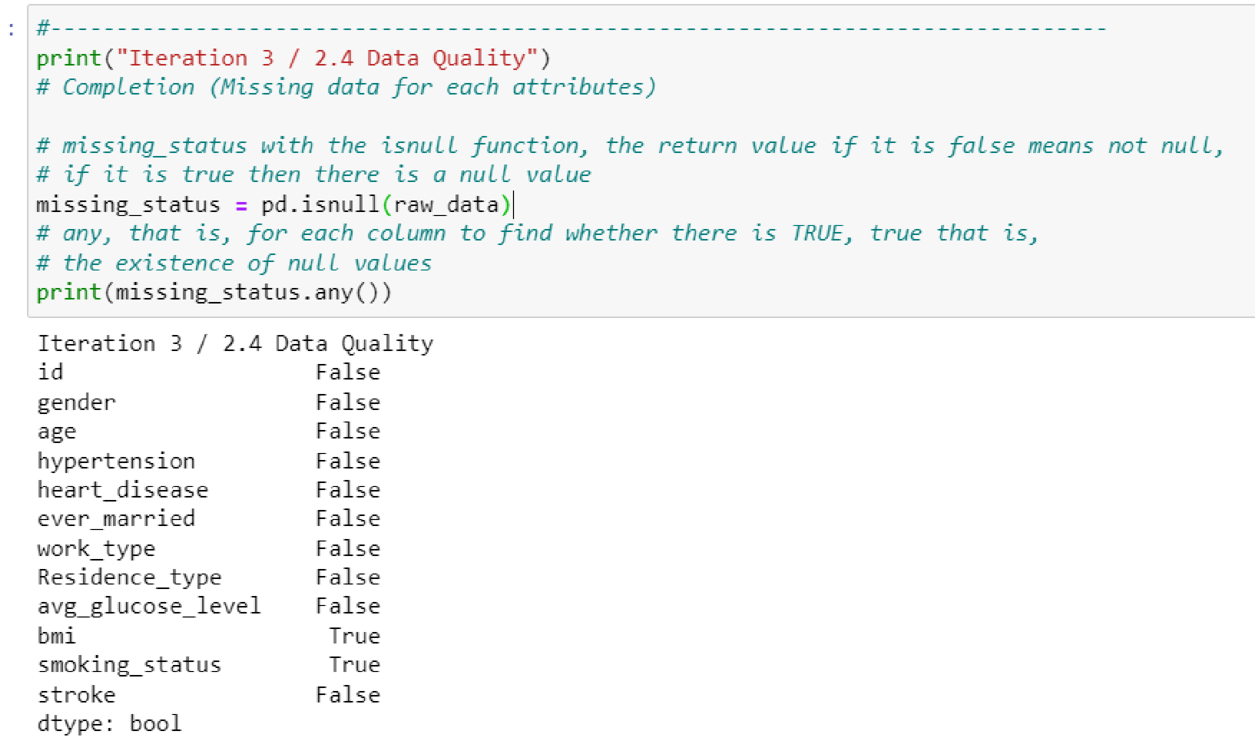


Figure 8. Distribution of stroke cases

2.5 Verifying Data Quality

Identifying the quality of the entire dataset, the percentage of attribute completion is 83.33%, and the percentage of records completion is 66.99%. The main reason for lowering the records completion rate is the incompletion of attribute smoking status of individuals, since only 69% of the patients provided their status of smoking. Also, 96.631% completion rate of BMI for individuals also lowers the total completion rate compared to the 100% completion rate for other attributes. As shown in Figure 9, we can see there are missing values in attributes BMI and smoking status. Because this data set is a median size dataset, for later identifying the relationship between the attribute of smoking status and the possibility of stroke, we could only use these 66.99 % datasets (with smoking status and BMI completed) to train the related prediction model.

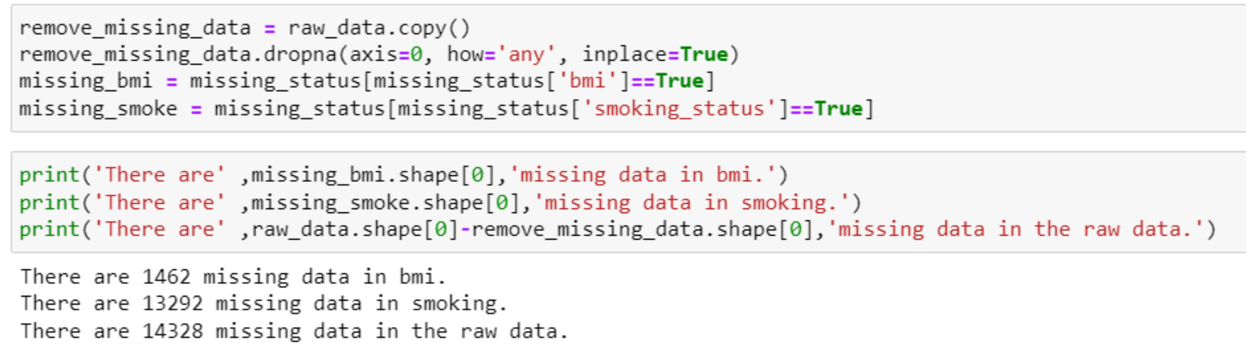
Furthermore, there are 1,462 and 13,292 records are missing/null values in BMI and smoking status respectively. Totally there are 14,326 records contain missing values in the raw dataset, as shown in Figure 10.



```
#---------------------------------------------------------------------------------
print("Iteration 3 / 2.4 Data Quality")
# Completion (Missing data for each attributes)

# missing_status with the isnull function, the return value if it is false means not null,
# if it is true then there is a null value
missing_status = pd.isnull(raw_data)
# any, that is, for each column to find whether there is TRUE, true that is,
# the existence of null values
print(missing_status.any())
```

```
Iteration 3 / 2.4 Data Quality
id                     False
gender                 False
age                    False
hypertension           False
heart_disease          False
ever_married           False
work_type              False
Residence_type         False
avg_glucose_level      False
bmi                     True
smoking_status          True
stroke                 False
dtype: bool
```

Figure 9. Show whether have missing values for each attribute



```
remove_missing_data = raw_data.copy()
remove_missing_data.dropna(axis=0, how='any', inplace=True)
missing_bmi = missing_status[missing_status['bmi']==True]
missing_smoke = missing_status[missing_status['smoking_status']==True]
```

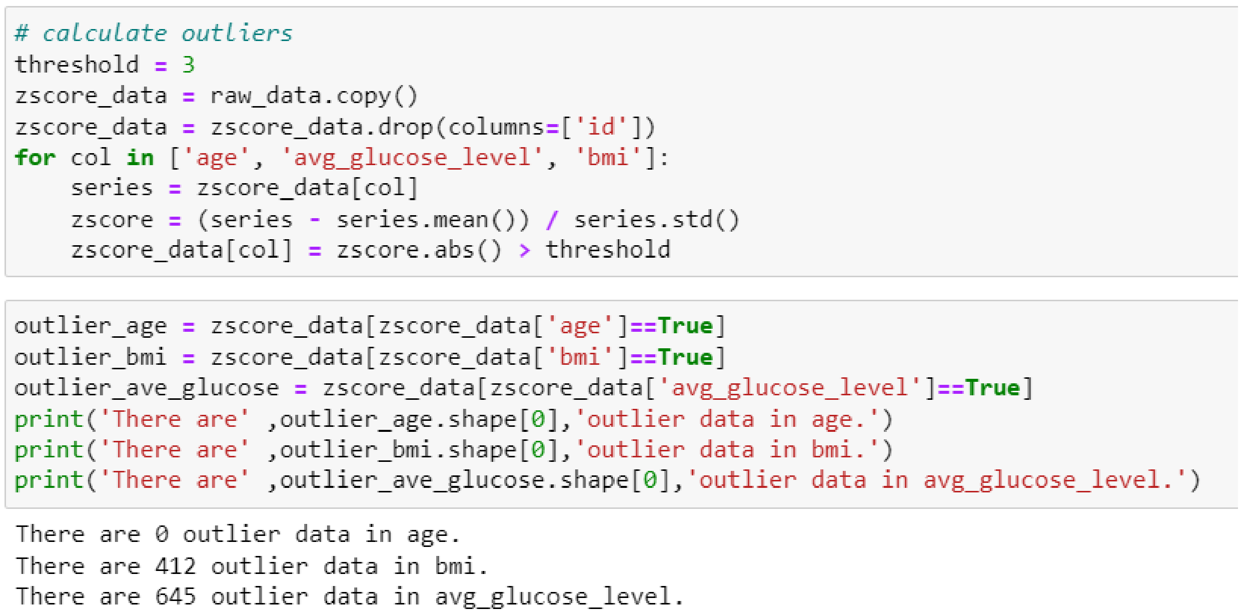
```
print('There are' ,missing_bmi.shape[0],'missing data in bmi.')
print('There are' ,missing_smoke.shape[0],'missing data in smoking.')
print('There are' ,raw_data.shape[0]-remove_missing_data.shape[0],'missing data in the raw data.')
```

```
There are 1462 missing data in bmi.
There are 13292 missing data in smoking.
There are 14328 missing data in the raw data.
```

Figure 10. Show the number of missing values

The outliers/extremes are calculated and identified in python as shown in Figure 11. We can see there are 0 outliers in age, 412 outlier data in BMI, and 645 outliers in average glucose level.
We can only identify outliers for attribute age, BMI, and average glucose level, because there is no means to identify outliers for id, it is a unique identifier, we will not analyze the relationship between id and stroke; for attribute hypertension, heart disease, ever married, and stroke, they are Boolean values, there are no outlier values; for attribute gender, work type, residence type, smoking status, their values are recorded as string, which the outlier value cannot be identified using a computer program. Because the way the computer calculated outliers for attributes is that the computer finds the mean value of each attribute and finds the range where most of the values lie according to the normal distribution function and calculates the deviation for each value in the attribute, and values with a large deviation (those that differ too much from

the group characteristics are considered outliers). The presence of outliers can have a significant impact on data analysis.



```
# calculate outliers
threshold = 3
zscore_data = raw_data.copy()
zscore_data = zscore_data.drop(columns=['id'])
for col in ['age', 'avg_glucose_level', 'bmi']:
    series = zscore_data[col]
    zscore = (series - series.mean()) / series.std()
    zscore_data[col] = zscore.abs() > threshold
```

```
outlier_age = zscore_data[zscore_data['age']==True]
outlier_bmi = zscore_data[zscore_data['bmi']==True]
outlier_ave_glucose = zscore_data[zscore_data['avg_glucose_level']==True]
print('There are' ,outlier_age.shape[0],'outlier data in age.')
print('There are' ,outlier_bmi.shape[0],'outlier data in bmi.')
print('There are' ,outlier_ave_glucose.shape[0],'outlier data in avg_glucose_level.')
```

```
There are 0 outlier data in age.
There are 412 outlier data in bmi.
There are 645 outlier data in avg_glucose_level.
```

Figure 11. Show the number of outliers

Based on the observation of data quality inside python, we found missing data and extreme data (most likely data errors) in this database. It is not difficult to understand that in the cycle of information collection it is unlikely that perfect data will be available, due to privacy or sensitivity issues, etc. And problems with the data will often be discovered when the data is analysed as the project progresses. Here are some of the potential issues that can arise during the execution of a project.

- **Missing data.** As can be seen from this database, the missing data is obvious and expected. In the questionnaires used to collect data, privacy concerns and other concerns caused patients to fill in questionnaires without filling in detailed personal information such as BMI, marital status, and place of work, among other things.
- **Data errors.** There are two possibilities for causing data errors and incorrect data entry at the time of recording. In the medical dataset used in this study, each record was manually recorded into the system by the healthcare professionals and a small amount of incorrect data entry was inevitable. In addition, machine misinterpretation, results in incorrect data being read in, as different machines use different ways of reading data. And most of these small amounts of incorrect data can be eliminated in most cases by filtering/processing outliers.
- **Measurement errors.** When data is measured without strict adherence to the measurement rules, small errors do occur from time to time or there is also a risk that units are reported incorrectly.
- **Imbalance numbers of labels.**
  As mentioned earlier, the distribution of labels was inconsistent. Only 783 instances were labelled as strokes, while the remaining 42,617 samples were labelled as non-

strokes. The large discrepancy is likely to make the model fit less well and insensitive to unseen values.

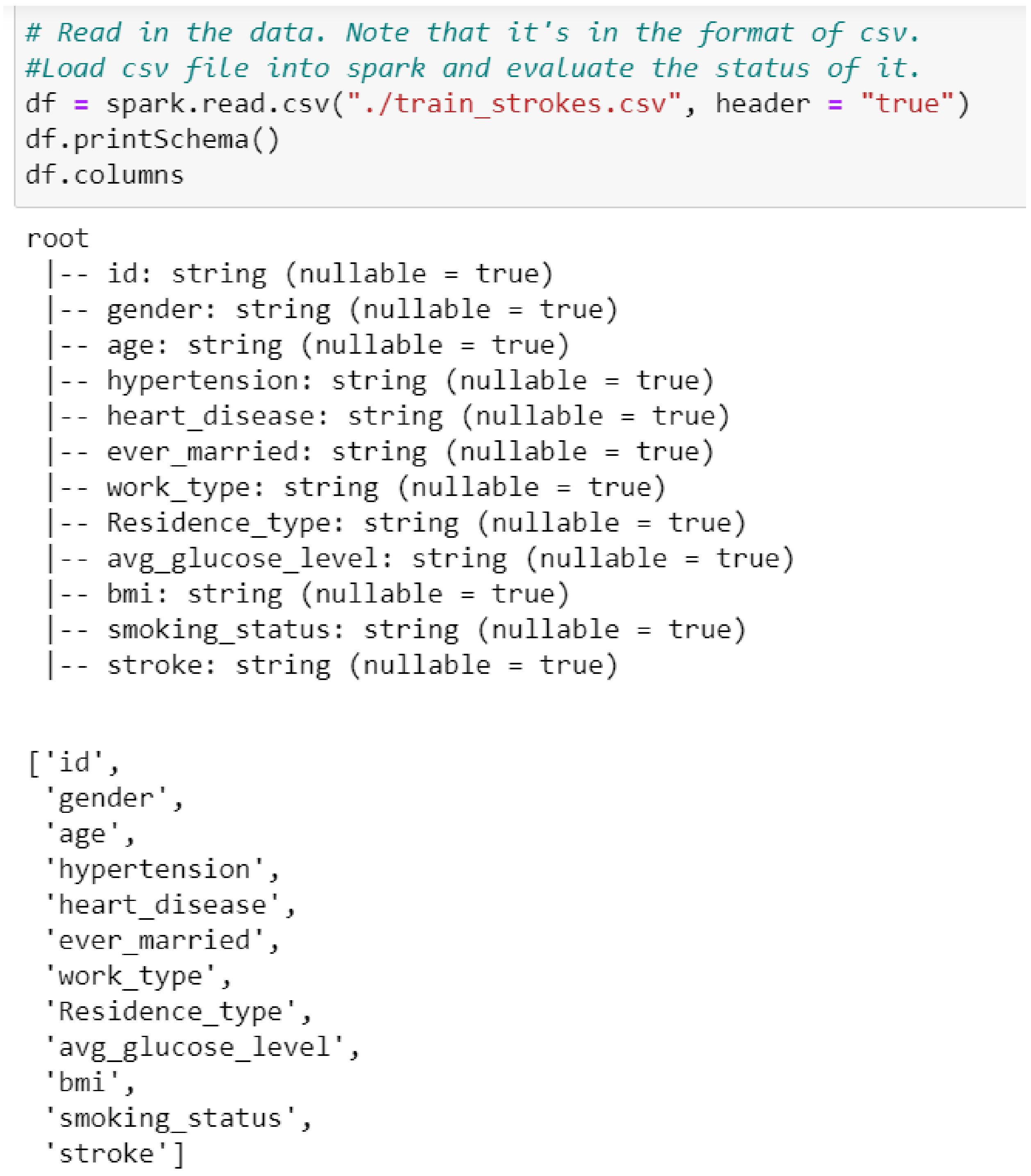
Attribute	Value Type	Completeness	Extreme / Outlier
ID	Numeric	N/A	N/A
Gender	Categorical (string)	100%	N/A
Age	Numeric	100%	0
Hypertension	Boolean (Flag)	100%	N/A
Heart_disease	Boolean (Flag)	100%	N/A
Ever_married	Boolean (Flag)	100%	N/A
Work_type	Categorical (string)	100%	N/A
Residence_type	Categorical (string)	100%	N/A
Avg_glucose_level	Numeric (Decimal)	100%	645
BMI	Numeric (Decimal)	96.631% （1462/43400）	412
Smoking_status	Categorical (string)	69.373% （13292/43400）	N/A
Stroke	Boolean (Flag)	100%	N/A

Table 3. Analysis of raw dataset quality

3. Data Preparation

3.1 Data Selection

Data selection, as the initial step in the data cleaning process, aims to remove some unnecessary attributes and eliminate the number of instances. It can be divided into two main parts, selecting items (row selection) and selecting attributes (column selection).
Items (row) selection: All instances of the initial dataset will be used to have a taste of how biased the dataset is as below in Figure 12 using Spark frame. The results are categorized as 0 and 1, with 0 indicating non-stroke and 1 indicating the occurrence of a stroke. However, there is a significant imbalance between the two categories, as only 783 instances were marked as stoke and the remaining 42,617 records were cited as no stoke as below in Figure 13.
For the row selection, we tried to keep all the original data as much as possible to reduce the impact of noise and deletion of real data on the later model building. Because more real data, build a better model, which will predict more accurately target attributes for long-term and future use purposes. We try to keep the data of each entry in this session, and then reduce the data rows as needed when doing data cleaning, handling empty data, data errors, and noise in the following sessions.

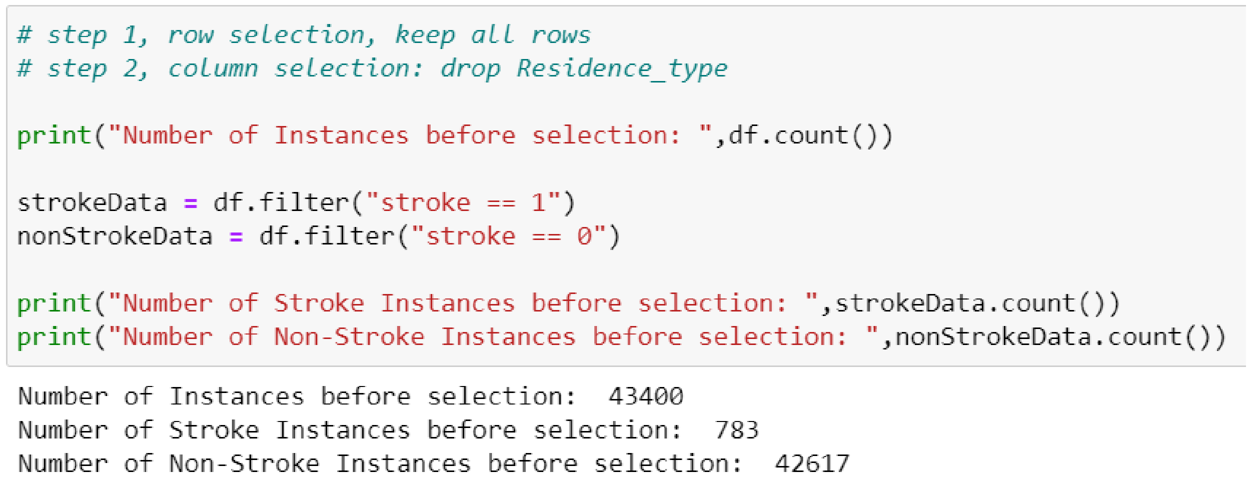


```python
# Read in the data. Note that it's in the format of csv.
#Load csv file into spark and evaluate the status of it.
df = spark.read.csv("./train_strokes.csv", header = "true")
df.printSchema()
df.columns
```

```
root
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: string (nullable = true)
 |-- hypertension: string (nullable = true)
 |-- heart_disease: string (nullable = true)
 |-- ever_married: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- Residence_type: string (nullable = true)
 |-- avg_glucose_level: string (nullable = true)
 |-- bmi: string (nullable = true)
 |-- smoking_status: string (nullable = true)
 |-- stroke: string (nullable = true)


['id',
 'gender',
 'age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'work_type',
 'Residence_type',
 'avg_glucose_level',
 'bmi',
 'smoking_status',
 'stroke']
```

Figure 12. Items (row) selection of the dataset

Iteration 4 / 3.1 Data Selection

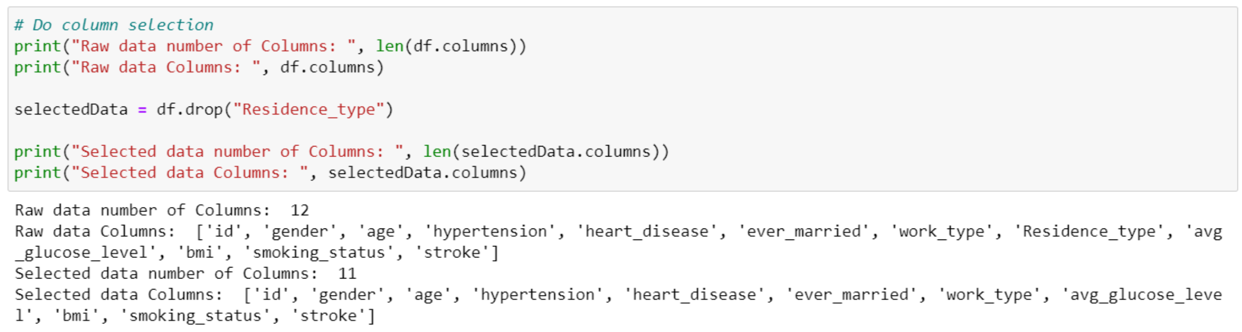


```
# step 1, row selection, keep all rows
# step 2, column selection: drop Residence_type

print("Number of Instances before selection: ",df.count())

strokeData = df.filter("stroke == 1")
nonStrokeData = df.filter("stroke == 0")

print("Number of Stroke Instances before selection: ",strokeData.count())
print("Number of Non-Stroke Instances before selection: ",nonStrokeData.count())
```

```
Number of Instances before selection:  43400
Number of Stroke Instances before selection:  783
Number of Non-Stroke Instances before selection:  42617
```

Figure 13. Analyze the target attribute based on the Items (row) selection

Attribute (column) selection: According to (Barnett & H, 2005; Chong, 2020; Wajngarten & Silva, 2019), risk factors for stroke include heart disease, hypertension, smoking habits, diabetes, and obesity, but of these risk factors, the type of dwelling is not one of them. Furthermore, according to section 2.4, some attributes contribute less to stroke, which means that they are less critical attributes. The presence of these values has the potential to influence the judgements and patterns of the algorithm. Once these tributes have been removed, the algorithm will start to move along the right track towards the correct and robust pattern.
As we did data exploration in section 2.4, residence type is not a risk, which has no effect on the outcome that ultimately leads to stroke. Therefore, it can be removed from further studies. Also, the attribute id is not needed for further studies, as id is an unique identifier, we do not need it to build the model, and it has no relationship with the target attribute stroke.
Figure 14 below illustrates the attribute selection process. We have deleted the attribute of Residence type and can see the data frame after column selection, there are 11 attributes left. It also, illustrates the dataset after column selection.



```
# Do column selection
print("Raw data number of Columns: ", len(df.columns))
print("Raw data Columns: ", df.columns)

selectedData = df.drop("Residence_type")

print("Selected data number of Columns: ", len(selectedData.columns))
print("Selected data Columns: ", selectedData.columns)
```

```
Raw data number of Columns:  12
Raw data Columns:  ['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'avg
_glucose_level', 'bmi', 'smoking_status', 'stroke']
Selected data number of Columns:  11
Selected data Columns:  ['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'avg_glucose_leve
l', 'bmi', 'smoking_status', 'stroke']
```

Figure 14. Attribute (column) selection of the dataset

3.2 Cleaning Data

The original files of the dataset had gaps and unfinished areas and these errors were often missed or deliberately ignored by those who did not record them in the dataset due to privacy concerns or other reasons of consideration. Two types of errors were found in the processed data, missing values and data errors. These values will be removed or replaced in preparation for further use of the data.

Missing values:

A total of two attributes were found to contain missing values, BMI with an integrity value of 96.631% and smoking status with 69.373%. These missing values can reduce the efficiency of the module's execution. Therefore, they need to be removed or averaged to fill them in before being applied to any model. This step eliminates the problem when applying data to certain algorithms and makes the algorithm less skewed in the wrong direction. Therefore, they need to be removed or filled by the mean value of the attribute before being applied to any model. This step eliminates the problem when applying data to certain algorithms, making the algorithm less skewed in the wrong direction.

A null or empty value is not very compatible with some algorithms. It is usually represented in many different forms, such as empty cells as well as N/A markers. In this dataset, both smoking status and BMI contain these missing values. There are three ways to remove them: directly from the record or populated according to some pattern (usually copied from the last populated instance) or populated with a fixed number derived from the averaging algorithm.

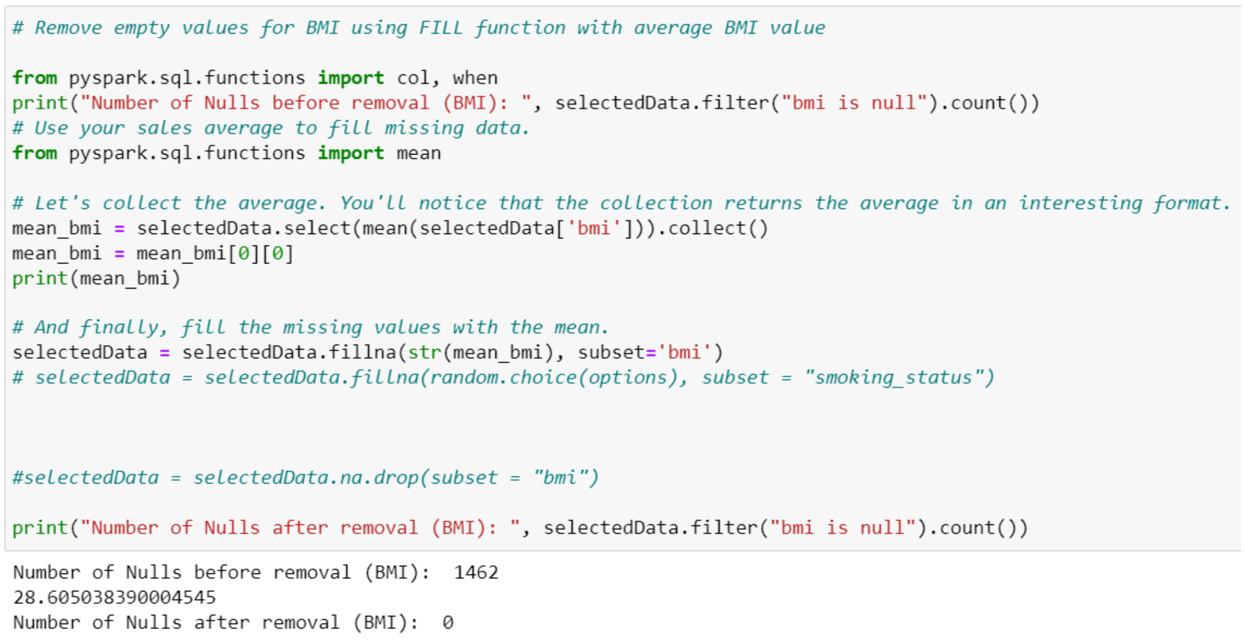
For the missing BMI and smoking status values in this project, we took 2 approaches to deal with the missing BMI and smoking status values separately.

For the smoking status values, the best approach is to simply delete these instances as there are three smoking statuses and the missing values take up 31% of the existing dataset, which is too large a proportion and randomly populating these values or replicating them from later instances is likely to result in the dataset losing its original characteristics, resulting in much less accurate predictions.

The BMI values could instead be populated using a fixed number derived from the averaging algorithm. As the BMI is only 1,462, roughly 4% of the missing values, using the mean padding would not affect the overall dataset too much and would preserve as many instances as possible to mitigate the problem of shortage of stroke cases.

Therefore, we decided to remove those instances with missing values in the smoking status and use the mean to populate those instances with missing values in the BMI.

Figure 15 shows the process of removing null/missing values for BMI in the dataset and filling in the mean value for BMI, Figure 16 shows the results of removing the null values from the smoking status attribute.

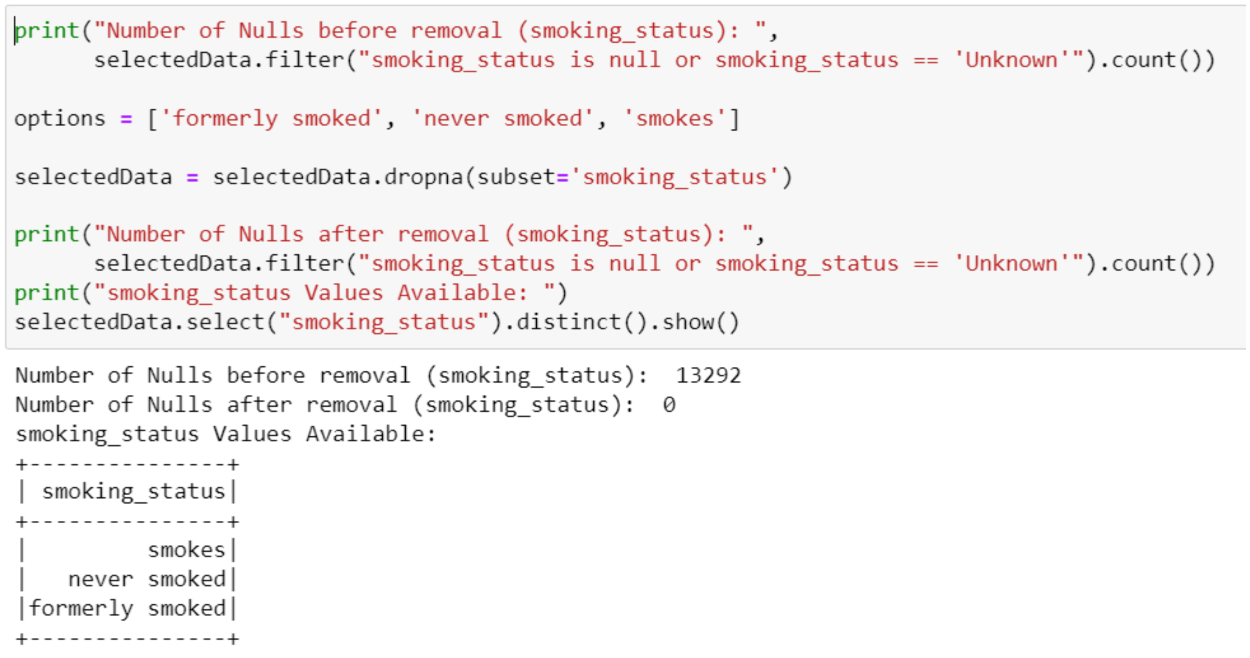


```
# Remove empty values for BMI using FILL function with average BMI value

from pyspark.sql.functions import col, when
print("Number of Nulls before removal (BMI): ", selectedData.filter("bmi is null").count())
# Use your sales average to fill missing data.
from pyspark.sql.functions import mean

# Let's collect the average. You'll notice that the collection returns the average in an interesting format.
mean_bmi = selectedData.select(mean(selectedData['bmi'])).collect()
mean_bmi = mean_bmi[0][0]
print(mean_bmi)

# And finally, fill the missing values with the mean.
selectedData = selectedData.fillna(str(mean_bmi), subset='bmi')
# selectedData = selectedData.fillna(random.choice(options), subset = "smoking_status")



#selectedData = selectedData.na.drop(subset = "bmi")

print("Number of Nulls after removal (BMI): ", selectedData.filter("bmi is null").count())
```

```
Number of Nulls before removal (BMI):  1462
28.605038390004545
Number of Nulls after removal (BMI):  0
```

Figure 15. Cleaning missing values in BMI

The method used to find missing values in the BMI attribute is filter(), which selects all null values(rows). And calculate the mean value in this attribute, then, use fillna() method to fill all missing values by mean value of BMI (28.61).



```
print("Number of Nulls before removal (smoking_status): ",
      selectedData.filter("smoking_status is null or smoking_status == 'Unknown'").count())

options = ['formerly smoked', 'never smoked', 'smokes']

selectedData = selectedData.dropna(subset='smoking_status')

print("Number of Nulls after removal (smoking_status): ",
      selectedData.filter("smoking_status is null or smoking_status == 'Unknown'").count())
print("smoking_status Values Available: ")
selectedData.select("smoking_status").distinct().show()
```

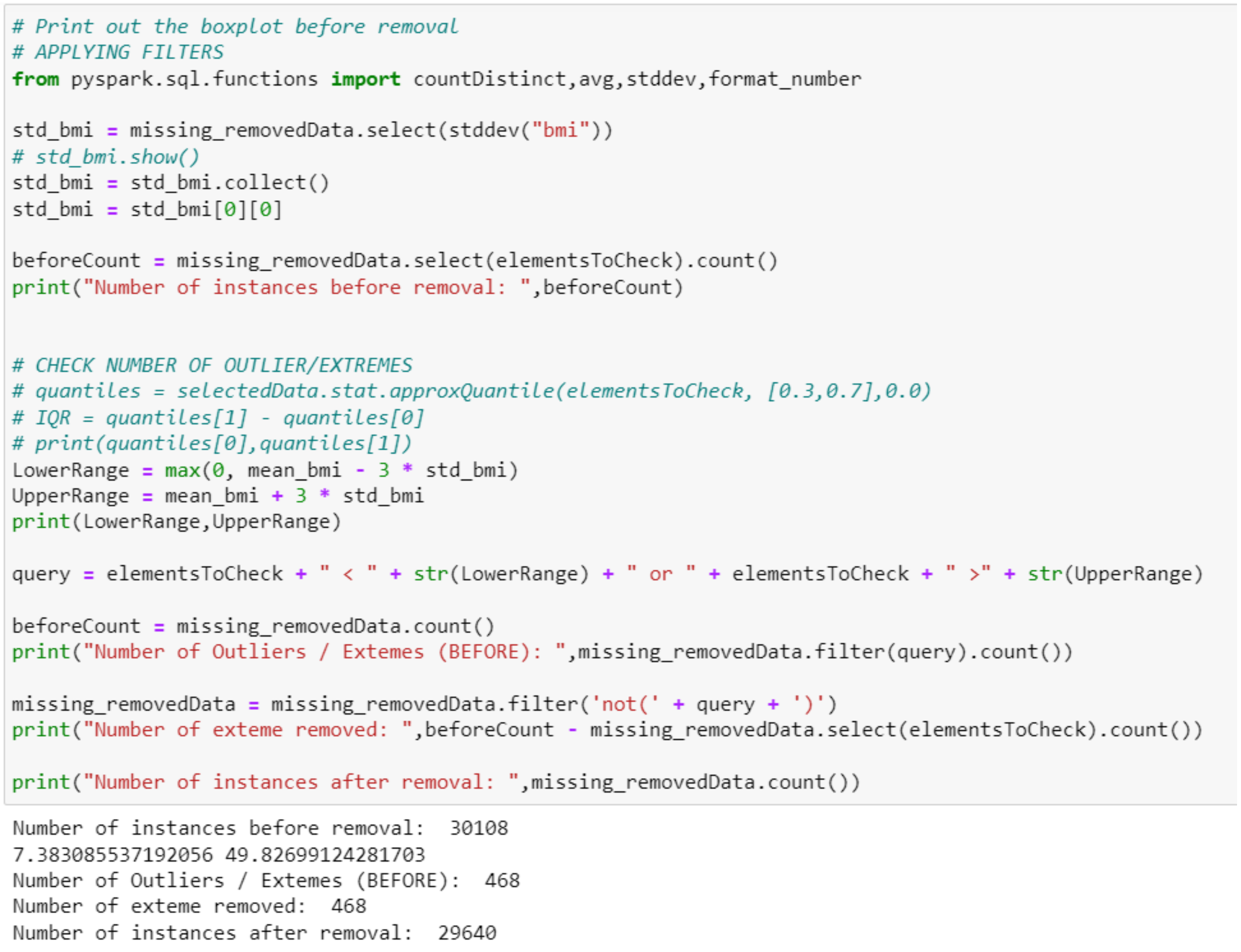
```
Number of Nulls before removal (smoking_status):  13292
Number of Nulls after removal (smoking_status):  0
smoking_status Values Available: 
+---------------+
| smoking_status|
+---------------+
|         smokes|
|   never smoked|
|formerly smoked|
+---------------+
```

Figure 16. Cleaning missing values in smoking_status

The method used to remove the missing values is to use the dropna() function in python, which selects all valid values and filters out those uncompleted values in smoking status values(rows). After this step, there are 30,108 records left, which means 13,292 records are reduced. When this algorithm is applied, the completeness of all attributes is increased to 100%.

Data Errors:
The only data errors contained in the file are extreme/outlier values, which can affect the module's predictions as the module may be biased towards outliers as it tries to cover many different possible values.
From Figure 11, we know that there are 0 outliers in age, 412 outlier data in BMI, and 645 outliers in average glucose level. we used the Zscore algorithm to calculate outliers in age, and BMI, and average glucose level. The method used to remove the outliers is to use the zscore method to calculate the deviation of the value with normal distribution, which selects all values in the range of normal distribution and filters out those outlier values in age, BMI, and average glucose(rows).
However, there is no zscore method/package in Spark, so we need to calculate the deviation of the value with normal distribution in BMI and average glucose level ourselves. We have identified that the values are out of 3 times the standard deviation as outliers.
From the calculation in Spark, we find that there are 468 outliers in BMI, and 241 outliers in average glucose level. These extreme values are removed using forced deletion. This method will attempt to drag the values from the extreme range to a reasonable range within the range.

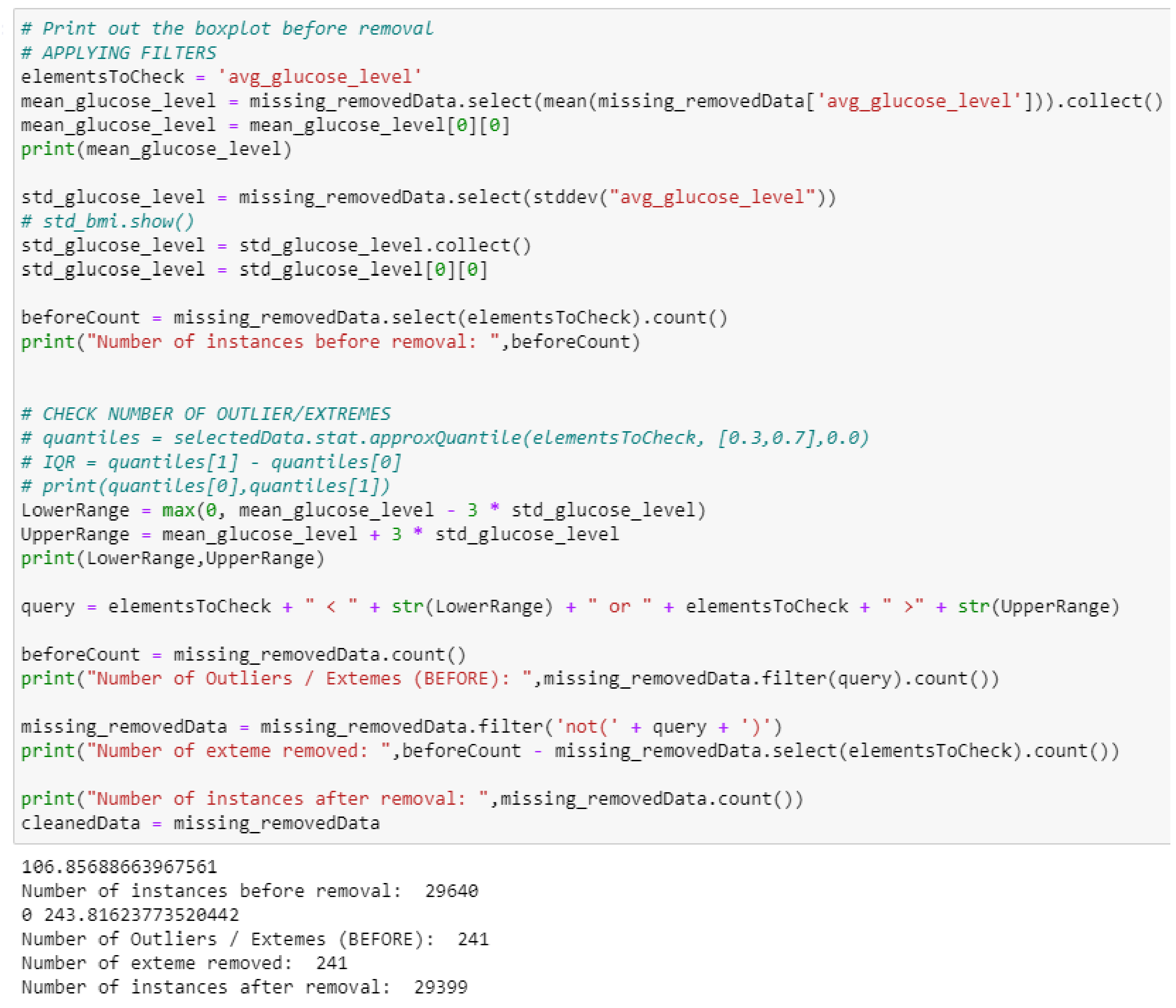
Figure 17 shows the process of removing outlier values in BMI, Figure 18 shows the process of removing outlier values in average glucose level.
Figure 18 also shows the results of removing the outliers from the BMI, and average glucose level attribute.



```python
# Print out the boxplot before removal
# APPLYING FILTERS
from pyspark.sql.functions import countDistinct,avg,stddev,format_number

std_bmi = missing_removedData.select(stddev("bmi"))
# std_bmi.show()
std_bmi = std_bmi.collect()
std_bmi = std_bmi[0][0]

beforeCount = missing_removedData.select(elementsToCheck).count()
print("Number of instances before removal: ",beforeCount)


# CHECK NUMBER OF OUTLIER/EXTREMES
# quantiles = selectedData.stat.approxQuantile(elementsToCheck, [0.3,0.7],0.0)
# IQR = quantiles[1] - quantiles[0]
# print(quantiles[0],quantiles[1])
LowerRange = max(0, mean_bmi - 3 * std_bmi)
UpperRange = mean_bmi + 3 * std_bmi
print(LowerRange,UpperRange)

query = elementsToCheck + " < " + str(LowerRange) + " or " + elementsToCheck + " >" + str(UpperRange)

beforeCount = missing_removedData.count()
print("Number of Outliers / Extemes (BEFORE): ",missing_removedData.filter(query).count())

missing_removedData = missing_removedData.filter('not(' + query + ')')
print("Number of exteme removed: ",beforeCount - missing_removedData.select(elementsToCheck).count())

print("Number of instances after removal: ",missing_removedData.count())
```

```
Number of instances before removal:  30108
7.383085537192056 49.82699124281703
Number of Outliers / Extemes (BEFORE):  468
Number of exteme removed:  468
Number of instances after removal:  29640
```

Figure 17. Cleaning BMI outliers process



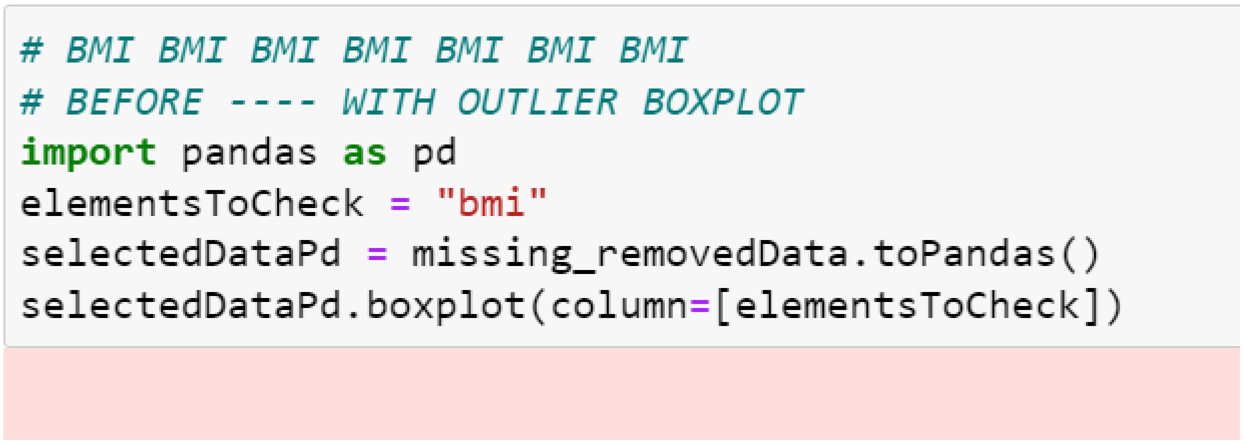
```
# Print out the boxplot before removal
# APPLYING FILTERS
elementsToCheck = 'avg_glucose_level'
mean_glucose_level = missing_removedData.select(mean(missing_removedData['avg_glucose_level'])).collect()
mean_glucose_level = mean_glucose_level[0][0]
print(mean_glucose_level)

std_glucose_level = missing_removedData.select(stddev("avg_glucose_level"))
# std_bmi.show()
std_glucose_level = std_glucose_level.collect()
std_glucose_level = std_glucose_level[0][0]

beforeCount = missing_removedData.select(elementsToCheck).count()
print("Number of instances before removal: ",beforeCount)


# CHECK NUMBER OF OUTLIER/EXTREMES
# quantiles = selectedData.stat.approxQuantile(elementsToCheck, [0.3,0.7],0.0)
# IQR = quantiles[1] - quantiles[0]
# print(quantiles[0],quantiles[1])
LowerRange = max(0, mean_glucose_level - 3 * std_glucose_level)
UpperRange = mean_glucose_level + 3 * std_glucose_level
print(LowerRange,UpperRange)

query = elementsToCheck + " < " + str(LowerRange) + " or " + elementsToCheck + " >" + str(UpperRange)

beforeCount = missing_removedData.count()
print("Number of Outliers / Extemes (BEFORE): ",missing_removedData.filter(query).count())

missing_removedData = missing_removedData.filter('not(' + query + ')')
print("Number of exteme removed: ",beforeCount - missing_removedData.select(elementsToCheck).count())

print("Number of instances after removal: ",missing_removedData.count())
cleanedData = missing_removedData
```

```
106.85688663967561
Number of instances before removal:  29640
0 243.81623773520442
Number of Outliers / Extemes (BEFORE):  241
Number of exteme removed:  241
Number of instances after removal:  29399
```

Figure 18. Cleaning average glucose level outliers process

The method used to remove the outliers is to use the manual calculation of the standard deviation of the value with normal distribution, which selects all values in the range of normal distribution and filters out those outlier values in age, BMI, and average glucose(rows). After this step, there are 29,399 records left, which means 709 records are reduced.

Figures 19 and 20 are the distribution boxplot (before and after) outlier removal status for the attribute BMI. It totally removed 468 extreme values/outliers.
Figures 21 and 22 are the distribution boxplot (before and after) outlier removal status of the attribute of Average glucose level. The boxplot is still containing some outliers that are being removed.
We can see that after removing outliers from BMI and average glucose attribute, the distribution of values in these 2 attributes is more normally distributed, and we reduced the potential noise for future model building and training.



```
# BMI BMI BMI BMI BMI BMI BMI
# BEFORE ---- WITH OUTLIER BOXPLOT
import pandas as pd
elementsToCheck = "bmi"
selectedDataPd = missing_removedData.toPandas()
selectedDataPd.boxplot(column=[elementsToCheck])
```

```
<AxesSubplot:>
```

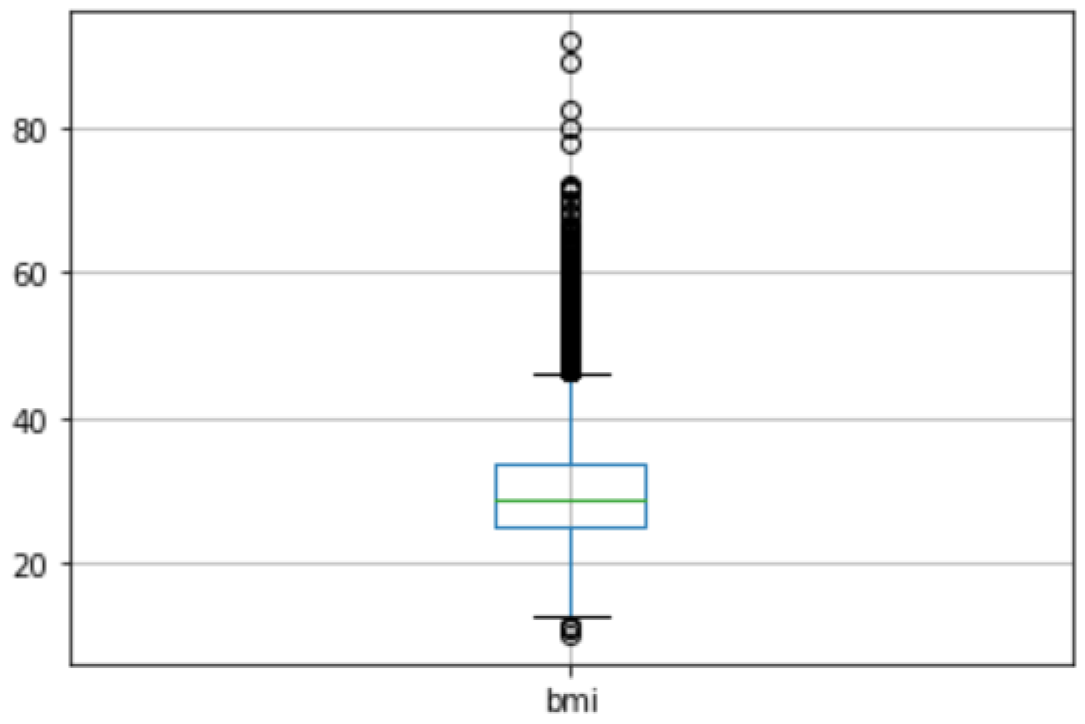


Figure 19: The boxplot of BMI (before clean outliers)

```
# AFTER ---- BMI WITHOUTLIER OUTLIER BOXPLOT
missing_removedData = missing_removedData.toPandas()
missing_removedData.boxplot(column=["bmi"])
```

```
<AxesSubplot:>
```

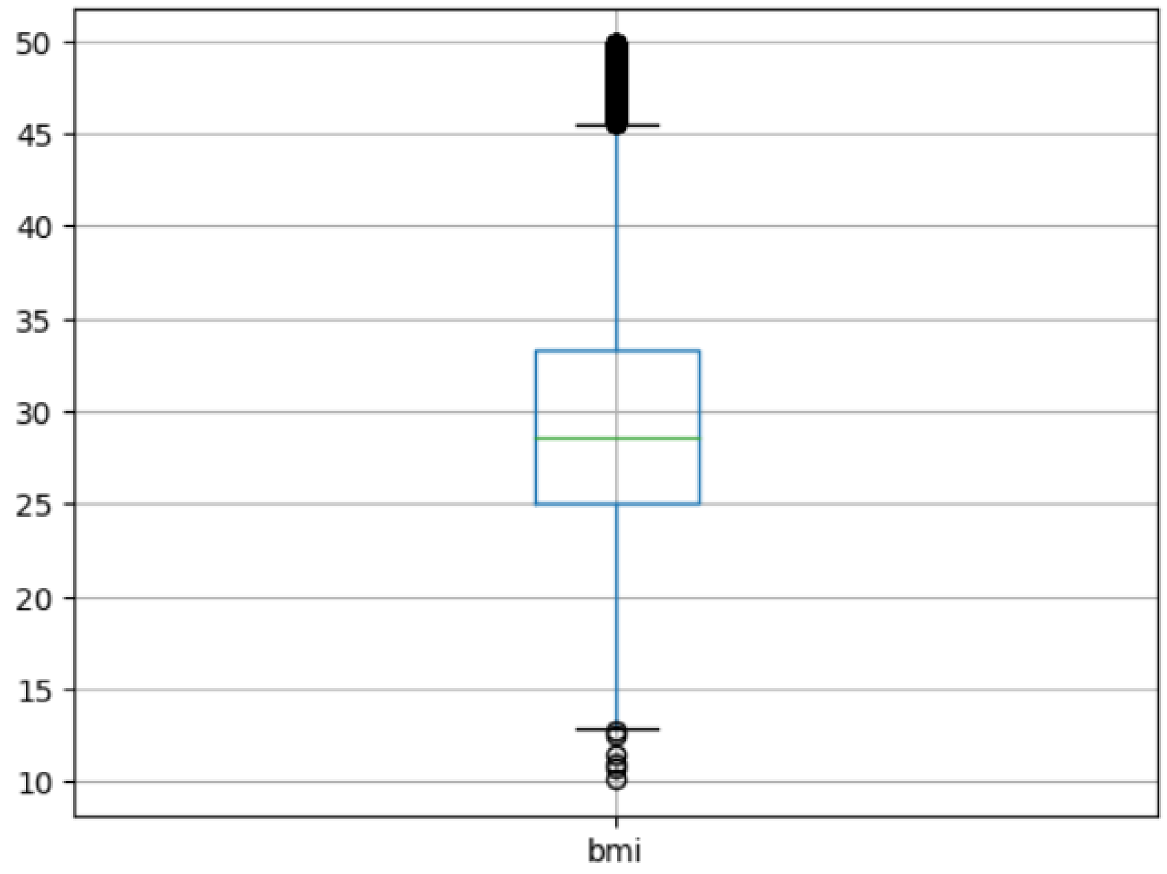


Figure 20: The boxplot of BMI (after clean outliers)

```
: # avg_glucose_level
  # BEFORE ---- WITH OUTLIER BOXPLOT
  elementsToCheck = "avg_glucose_level"
  selectedDataPd = missing_removedData.toPandas()
  selectedDataPd.boxplot(column=[elementsToCheck])
```

```
: <AxesSubplot:>
```

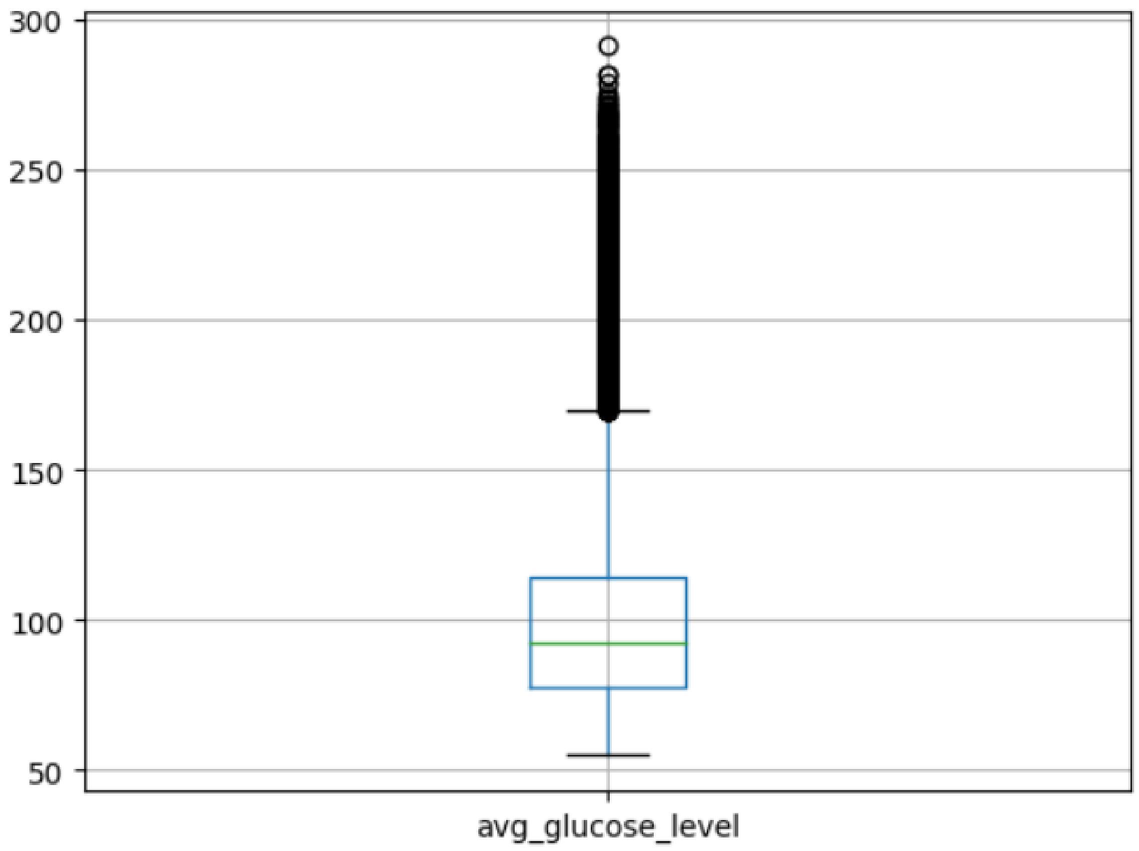


Figure 21: The boxplot of Average glucose level (before clean outliers)

```python
# AFTER ---- average glucose level WITHOUTLIER OUTLIER BOXPLOT
import pandas as pd
missing_removedData = missing_removedData.toPandas()
missing_removedData.boxplot(column=["avg_glucose_level"])
```

```
<AxesSubplot:>
```

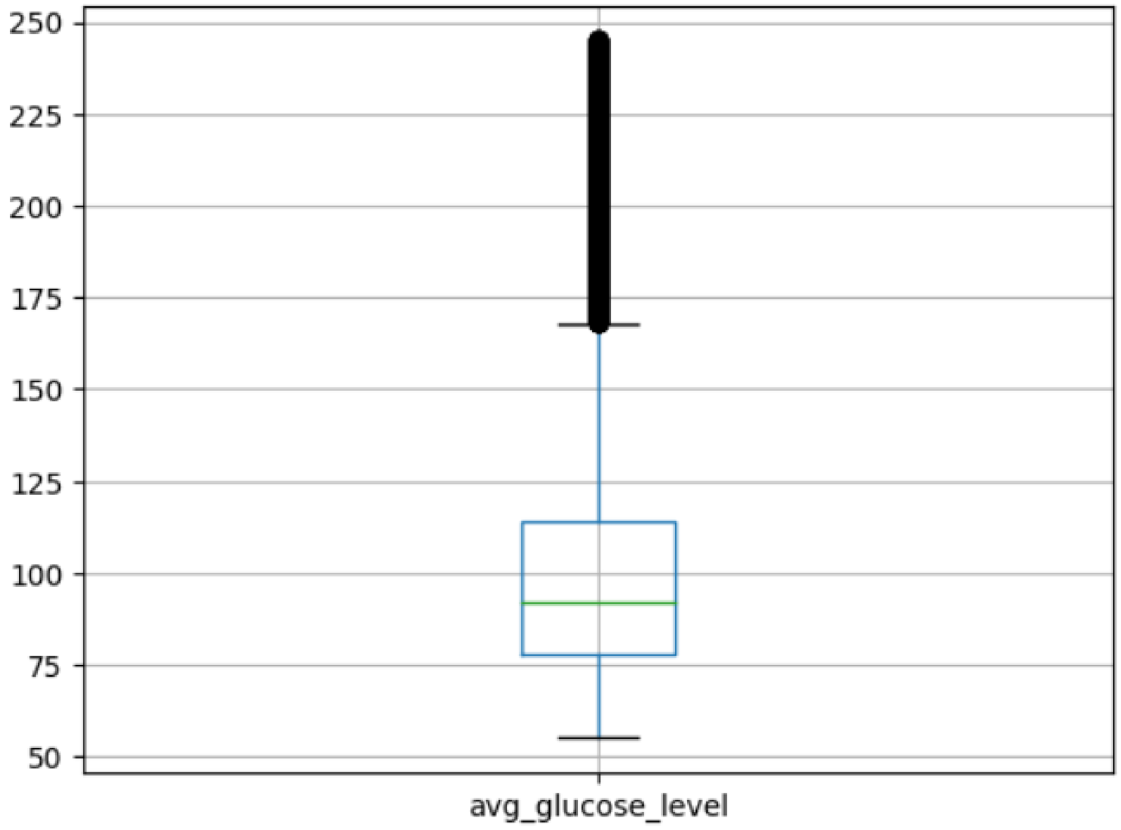


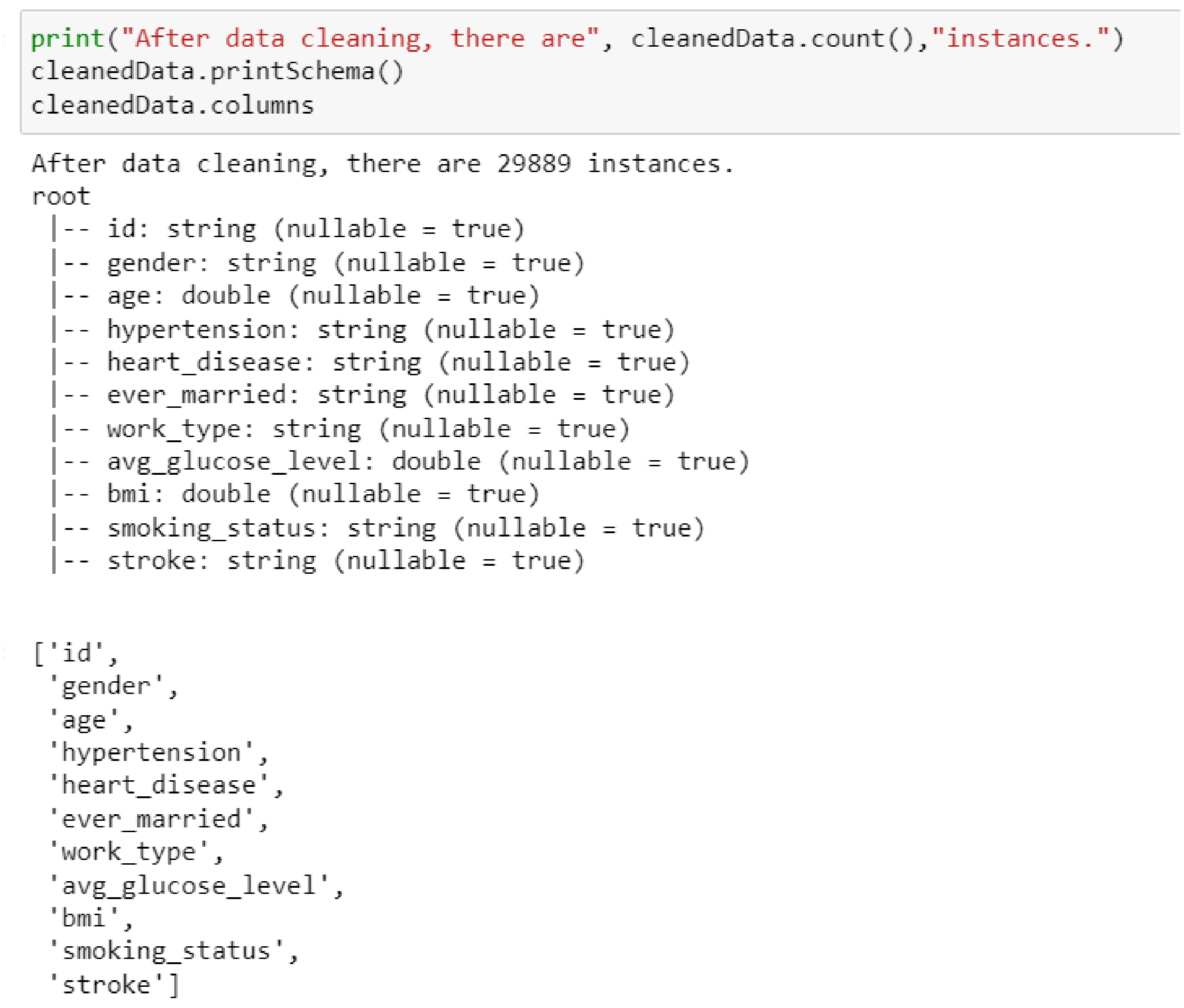
Figure 22: The boxplot of Average glucose level (after clean outliers)

From Figure 23 and Figure 24, we can see the difference between before cleaning the dataset and after cleaning the dataset.
Figure 23 is showing the dataset before data cleaning.
Figure 24 is showing the dataset after data cleaning.

```python
# Read in the data. Note that it's in the format of csv.
#Load csv file into spark and evaluate the status of it.
df = spark.read.csv("./train_strokes.csv", header = "true")
print("Raw data, there are", df.count(),"instances.")
df.printSchema()
df.columns
```

```
Raw data, there are 43400 instances.
root
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: string (nullable = true)
 |-- hypertension: string (nullable = true)
 |-- heart_disease: string (nullable = true)
 |-- ever_married: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- Residence_type: string (nullable = true)
 |-- avg_glucose_level: string (nullable = true)
 |-- bmi: string (nullable = true)
 |-- smoking_status: string (nullable = true)
 |-- stroke: string (nullable = true)


['id',
 'gender',
 'age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'work_type',
 'Residence_type',
 'avg_glucose_level',
 'bmi',
 'smoking_status',
 'stroke']
```

Figure 23. Dataset before data cleaning



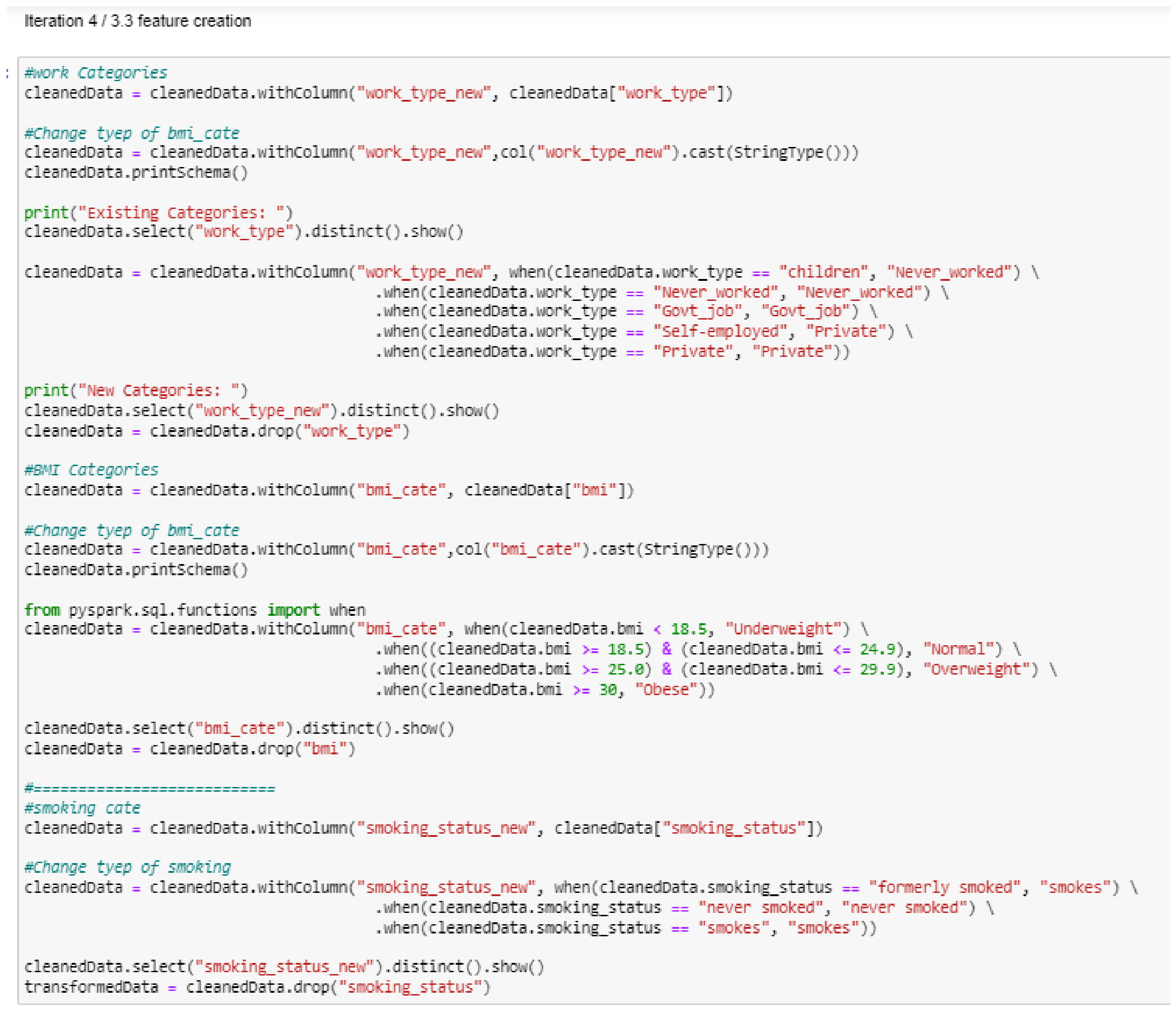
```
print("After data cleaning, there are", cleanedData.count(),"instances.")
cleanedData.printSchema()
cleanedData.columns
```

```
After data cleaning, there are 29889 instances.
root
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: double (nullable = true)
 |-- hypertension: string (nullable = true)
 |-- heart_disease: string (nullable = true)
 |-- ever_married: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- avg_glucose_level: double (nullable = true)
 |-- bmi: double (nullable = true)
 |-- smoking_status: string (nullable = true)
 |-- stroke: string (nullable = true)


['id',
 'gender',
 'age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'work_type',
 'avg_glucose_level',
 'bmi',
 'smoking_status',
 'stroke']
```

Figure 24. Dataset after data cleaning

3.3 Constructing New Data

To construct new data by creating new features, here three new features are created, new_smoking_status, new_bmi, new_work_type. New smoking status is set as a flag type from the nominal type of smoking status, the New BMI is re-grouped by a certain range of values and the new work type is re-classified by general knowledge base respectively.
Figure 25 shows the process of constructing new data as below.

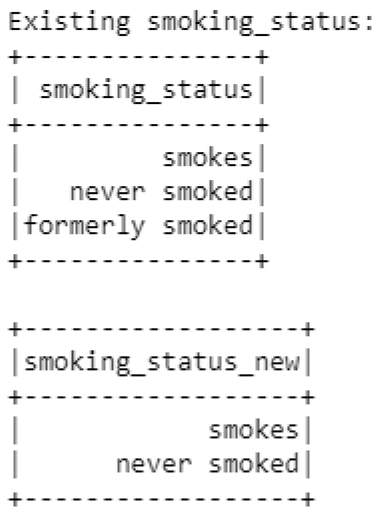


Iteration 4 / 3.3 feature creation

```
#work Categories
cleanedData = cleanedData.withColumn("work_type_new", cleanedData["work_type"])

#Change tyep of bmi_cate
cleanedData = cleanedData.withColumn("work_type_new",col("work_type_new").cast(StringType()))
cleanedData.printSchema()

print("Existing Categories: ")
cleanedData.select("work_type").distinct().show()

cleanedData = cleanedData.withColumn("work_type_new", when(cleanedData.work_type == "children", "Never_worked") \
                                       .when(cleanedData.work_type == "Never_worked", "Never_worked") \
                                       .when(cleanedData.work_type == "Govt_job", "Govt_job") \
                                       .when(cleanedData.work_type == "Self-employed", "Private") \
                                       .when(cleanedData.work_type == "Private", "Private"))

print("New Categories: ")
cleanedData.select("work_type_new").distinct().show()
cleanedData = cleanedData.drop("work_type")

#BMI Categories
cleanedData = cleanedData.withColumn("bmi_cate", cleanedData["bmi"])

#Change tyep of bmi_cate
cleanedData = cleanedData.withColumn("bmi_cate",col("bmi_cate").cast(StringType()))
cleanedData.printSchema()

from pyspark.sql.functions import when
cleanedData = cleanedData.withColumn("bmi_cate", when(cleanedData.bmi < 18.5, "Underweight") \
                                       .when((cleanedData.bmi >= 18.5) & (cleanedData.bmi <= 24.9), "Normal") \
                                       .when((cleanedData.bmi >= 25.0) & (cleanedData.bmi <= 29.9), "Overweight") \
                                       .when(cleanedData.bmi >= 30, "Obese"))

cleanedData.select("bmi_cate").distinct().show()
cleanedData = cleanedData.drop("bmi")

#==========================
#smoking cate
cleanedData = cleanedData.withColumn("smoking_status_new", cleanedData["smoking_status"])

#Change tyep of smoking
cleanedData = cleanedData.withColumn("smoking_status_new", when(cleanedData.smoking_status == "formerly smoked", "smokes") \
                                       .when(cleanedData.smoking_status == "never smoked", "never smoked") \
                                       .when(cleanedData.smoking_status == "smokes", "smokes"))

cleanedData.select("smoking_status_new").distinct().show()
transformedData = cleanedData.drop("smoking_status")
```

Figure 25. Constructing new data

1. **The new smoking status**
   The initial smoking status includes formerly smoked, smoking, and never smoking. Studies show whether smoke or not is a key predictor of our target attribute stroke. Therefore, smoking status is flagged as smoking and never smoking, as Figure 26 shows.



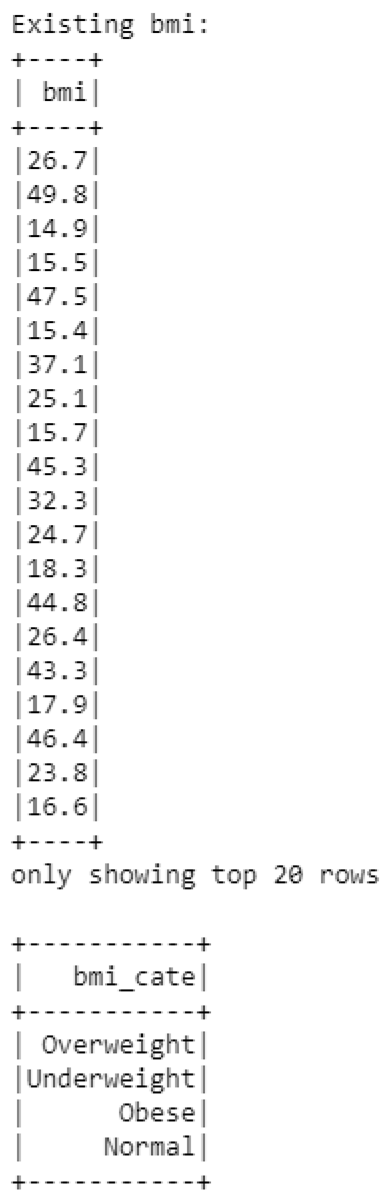
```
Existing smoking_status:
+---------------+
| smoking_status|
+---------------+
|         smokes|
|   never smoked|
|formerly smoked|
+---------------+

+------------------+
|smoking_status_new|
+------------------+
|            smokes|
|      never smoked|
+------------------+
```

Figure 26. New Smoking status after set a flag

2. **BMI groups:**
BMI stands for body mass index and is a simple calculation using a person's height and weight. The formula is BMI = kg/m^2, where kg is a person's weight in kilograms and m^2 is the square of their height in meters. A BMI of 25.0 or higher is considered overweight, while a healthy range is 18.5 to 24.9. And one of the risk factors for stroke, the target attribute of this study, is BMI. for these reasons, I created the characteristics of the weight categories based on the BMI results as Figure 27 shows.

BMI ranges	**Category**
–18.5	Underweight
18.5 - 24.9	Normal
25.0 - 29.9	Overweight
30 –	Obese

Table 4. new BMI category



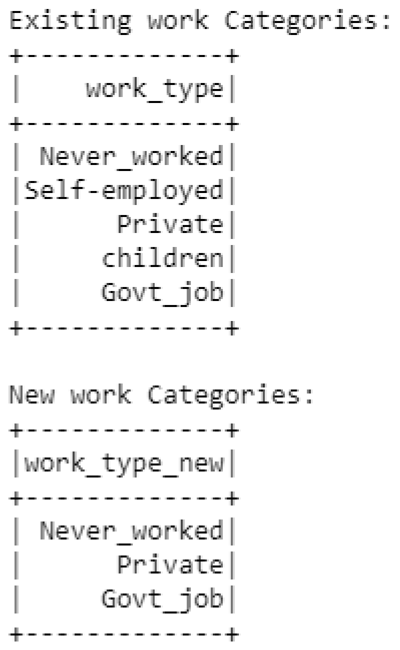
```
Existing bmi:
+----+
| bmi|
+----+
|26.7|
|49.8|
|14.9|
|15.5|
|47.5|
|15.4|
|37.1|
|25.1|
|15.7|
|45.3|
|32.3|
|24.7|
|18.3|
|44.8|
|26.4|
|43.3|
|17.9|
|46.4|
|23.8|
|16.6|
+----+
only showing top 20 rows

+-----------+
|   bmi_cate|
+-----------+
| Overweight|
|Underweight|
|      Obese|
|     Normal|
+-----------+
```

Figure 27. New BMI groups

3. **New Work type:**
   The work type initially includes five categories: children, never worked, self-employed, government jobs, and private. Some of these types should be grouped together because these job categories have a high degree of similarity to each other. Also, by merging these similar job types, the complexity of the module is reduced at the same time. It was decided to combine the categories of Child and Never Worked into Never Worked and Self-Employed and Private Company Work into self-employed, as shown in Figure 28.

Existing categories	**Merge to**
Children	Never Worked
Never Worked	Never Worked(remain)
Government Job	Government Job(remain)
Self-employment Job	Private Job
Private Job	Private job (remain)

Table 5. new work type



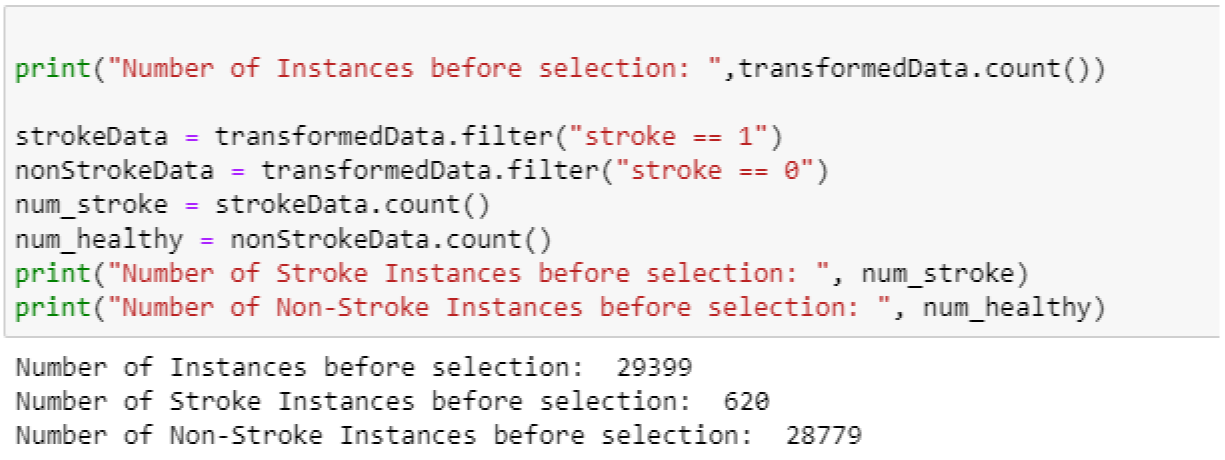
```
Existing work Categories:
+-------------+
|    work_type|
+-------------+
| Never_worked|
|Self-employed|
|      Private|
|     children|
|     Govt_job|
+-------------+

New work Categories:
+-------------+
|work_type_new|
+-------------+
| Never_worked|
|      Private|
|     Govt_job|
+-------------+
```

Figure 28. New Work categories

3.4 Integrating Data

Following the data cleaning process and constructing new features, the data cleaning process succeeded in removing the existing extreme/outlier and missing values. However, the imbalanced label problem is not solved, and this problem needs to be handled before building a model and doing data mining.

After the data cleaning process and the construction of new features, the data cleaning process successfully eliminated the existing extreme/outliers and missing values. However, the imbalanced labelling problem was not solved, and this issue needs to be addressed before building the model and performing data mining. Unbalanced labels, with too large a difference in the number of stroke patients and non-stroke patients, can lead to the prior probability of the data itself, which can have a bias towards negative samples of stroke. And according to Bayesian theory, the prior probability is the probability that can be obtained before the model experiment or sampling based on previous experience and analysis. To put it in layman's terms, there are now 620 samples labelled as stroke and 29,399 samples labelled as not stroke in the dataset as shown in Figure 29. Using this dataset to train the model, the model will most likely predict the patients who do not have a stroke accurately, while the patients with high risk of stroke are predicted to have a high probability of not having a stroke. This model, however, has a high risk of not being able to accurately predict the likelihood of stroke for patients at an early stage in practical use, which is contrary to our goal of doing this project and modelling each other. To address the potential dangers of such label imbalances, we need to rebalance the data set between stroke and healthy (non-stroke) patients.

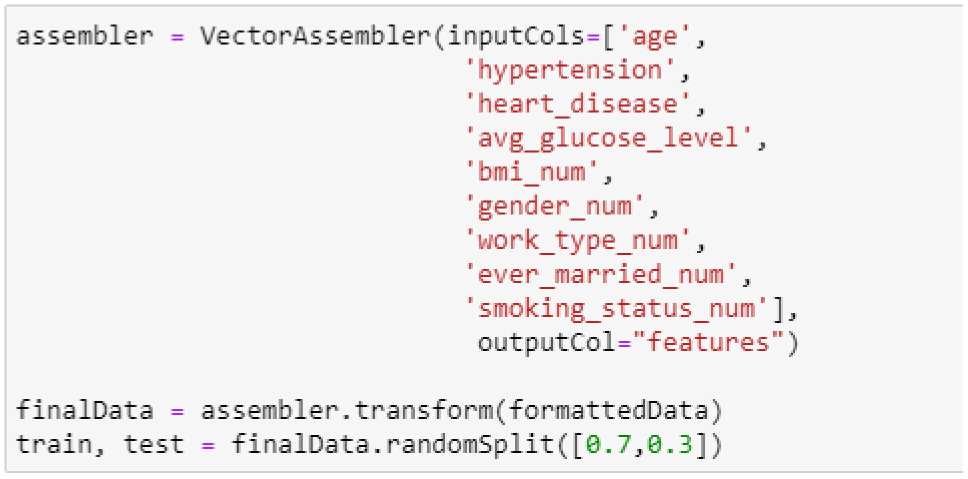


```
print("Number of Instances before selection: ",transformedData.count())

strokeData = transformedData.filter("stroke == 1")
nonStrokeData = transformedData.filter("stroke == 0")
num_stroke = strokeData.count()
num_healthy = nonStrokeData.count()
print("Number of Stroke Instances before selection: ", num_stroke)
print("Number of Non-Stroke Instances before selection: ", num_healthy)
```

```
Number of Instances before selection:  29399
Number of Stroke Instances before selection:  620
Number of Non-Stroke Instances before selection:  28779
```

Figure 29. Current stroke distribution

Before balancing the stroke and non-stroke samples, we need to perform data partitioning first, because we only want to balance the data inside the training set so that the model can have enough balanced data for training and can predict the results of the real data more accurately. The data in the test set maintains the characteristics of the original real data set as much as possible, which can also better test the results of the model training. Therefore, we split the dataset to train and test datasets based on the 7:3 ratio.
The main idea of partitioning the dataset into validation sets is to prevent our model from overfitting, i.e., the model becomes very good at classifying samples in the training set but cannot generalize and accurately classify data that it has not seen before.



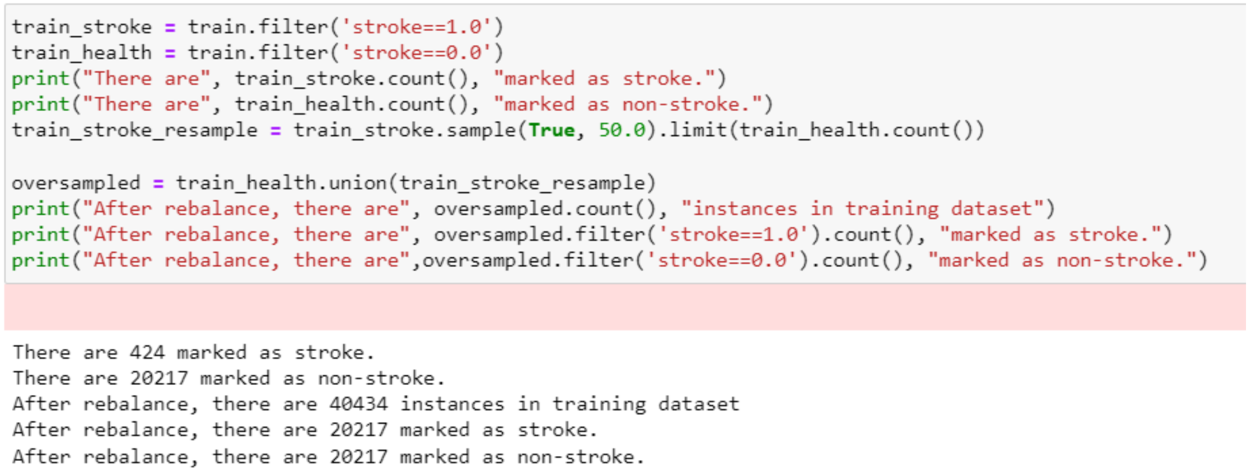
```
assembler = VectorAssembler(inputCols=['age',
                             'hypertension',
                             'heart_disease',
                             'avg_glucose_level',
                             'bmi_num',
                             'gender_num',
                             'work_type_num',
                             'ever_married_num',
                             'smoking_status_num'],
                              outputCol="features")

finalData = assembler.transform(formattedData)
train, test = finalData.randomSplit([0.7,0.3])
```

Figure 30. Train, test split to 7:3

It is assumed that these data sets are applied directly to the algorithm. In this case, the algorithm would obtain a low recall, i.e. the algorithm would be good at detecting those non-stroke values, but very insensitive to those stroke data, as the sample size for stroke is much smaller and would not allow the algorithm to study its patterns.
Based on this concern, further data prediction of the dataset was required to fill the gap between the stroke and non-stroke data. In total, 620 cases were classified as instances of stroke and 28,779 cases were classified as non-stroke. On this basis, the oversampling method is more applicable to this case than the under sampling. It would increase the number of stroke instances without decreasing the number of non-stroke instances. By doing so, the algorithm itself will have a large enough record to find and refine patterns.

After splitting the dataset into the training dataset and test dataset. Repeated sampling will be used in this data prediction process. The algorithm will calculate the distance between the existing minority and the other records in the dataset, select a certain number of nearest neighbors with similar characteristics to the existing records, generate a new record based on the combination, and randomly select elements from its nearest neighbors.
We use the algorithm of repeating to solve the imbalance label problem in the training dataset, and generate more data for stroke, which makes the stroke patients and non-stroke patients' ratio around 1: 1. We repeat stroke instances 50 times to make stroke patients and non-stroke patients' ratio around 1: 1, which solves the imbalance label problem. And we are integrating newly generated stroke instances into the dataset.

In this way, we have generated new data on stroke patients to balance the unevenness of stroke and non-stroke distribution and integrate new data records into the dataset. We can see the new stroke distribution in the dataset is balanced as shown in Figure 31, and Figure 32. There are 20,217 records are marked as stroke, and there are 20,217 records are marked as non-stroke.



```
train_stroke = train.filter('stroke==1.0')
train_health = train.filter('stroke==0.0')
print("There are", train_stroke.count(), "marked as stroke.")
print("There are", train_health.count(), "marked as non-stroke.")
train_stroke_resample = train_stroke.sample(True, 50.0).limit(train_health.count())

oversampled = train_health.union(train_stroke_resample)
print("After rebalance, there are", oversampled.count(), "instances in training dataset")
print("After rebalance, there are", oversampled.filter('stroke==1.0').count(), "marked as stroke.")
print("After rebalance, there are",oversampled.filter('stroke==0.0').count(), "marked as non-stroke.")
```

```
There are 424 marked as stroke.
There are 20217 marked as non-stroke.
After rebalance, there are 40434 instances in training dataset
After rebalance, there are 20217 marked as stroke.
After rebalance, there are 20217 marked as non-stroke.
```

Figure 31. Apply repeat algorithm rebalances stroke attribute

```
import seaborn
strokeData = oversampled.filter('stroke==1.0')
nonStrokeData = oversampled.filter('stroke==0')
num_stroke = strokeData.count()
num_healthy = nonStrokeData.count()
print("There are", num_stroke, "records are marked stroke.")
print("There are", num_healthy, "records are marked as healthy (non-stroke).")
seaborn.barplot(y=['stroke', 'healthy'], x=[num_stroke, num_healthy], orient='horizon')
```

```
There are 20217 records are marked stroke.
There are 20217 records are marked as healthy (non-stroke).

<AxesSubplot:>
```

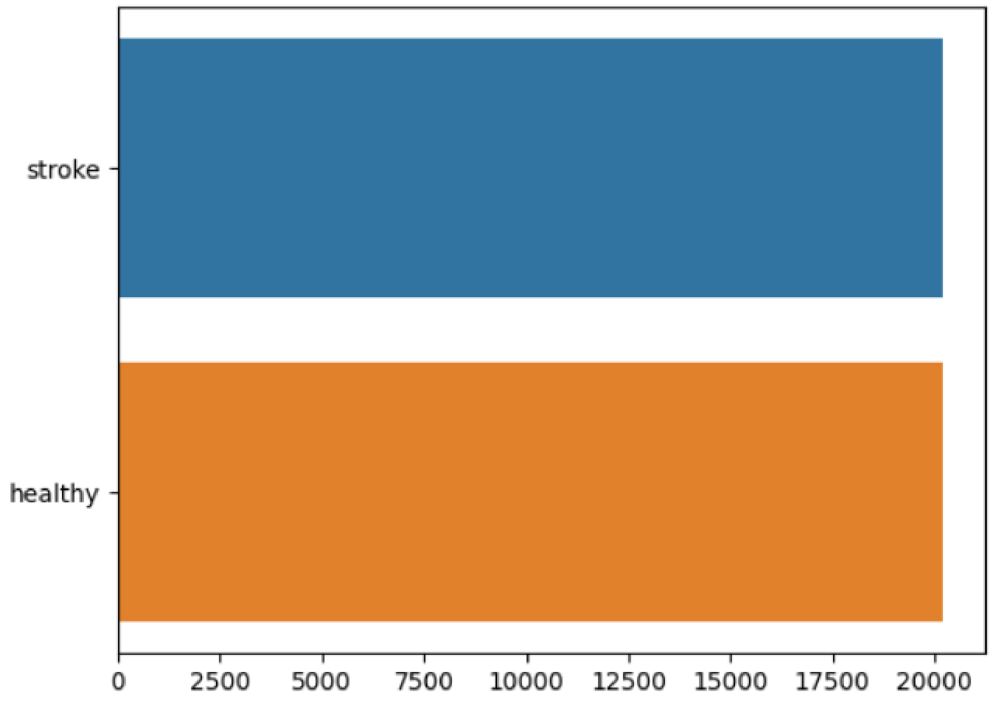


Figure 32. Stroke distribution after rebalances

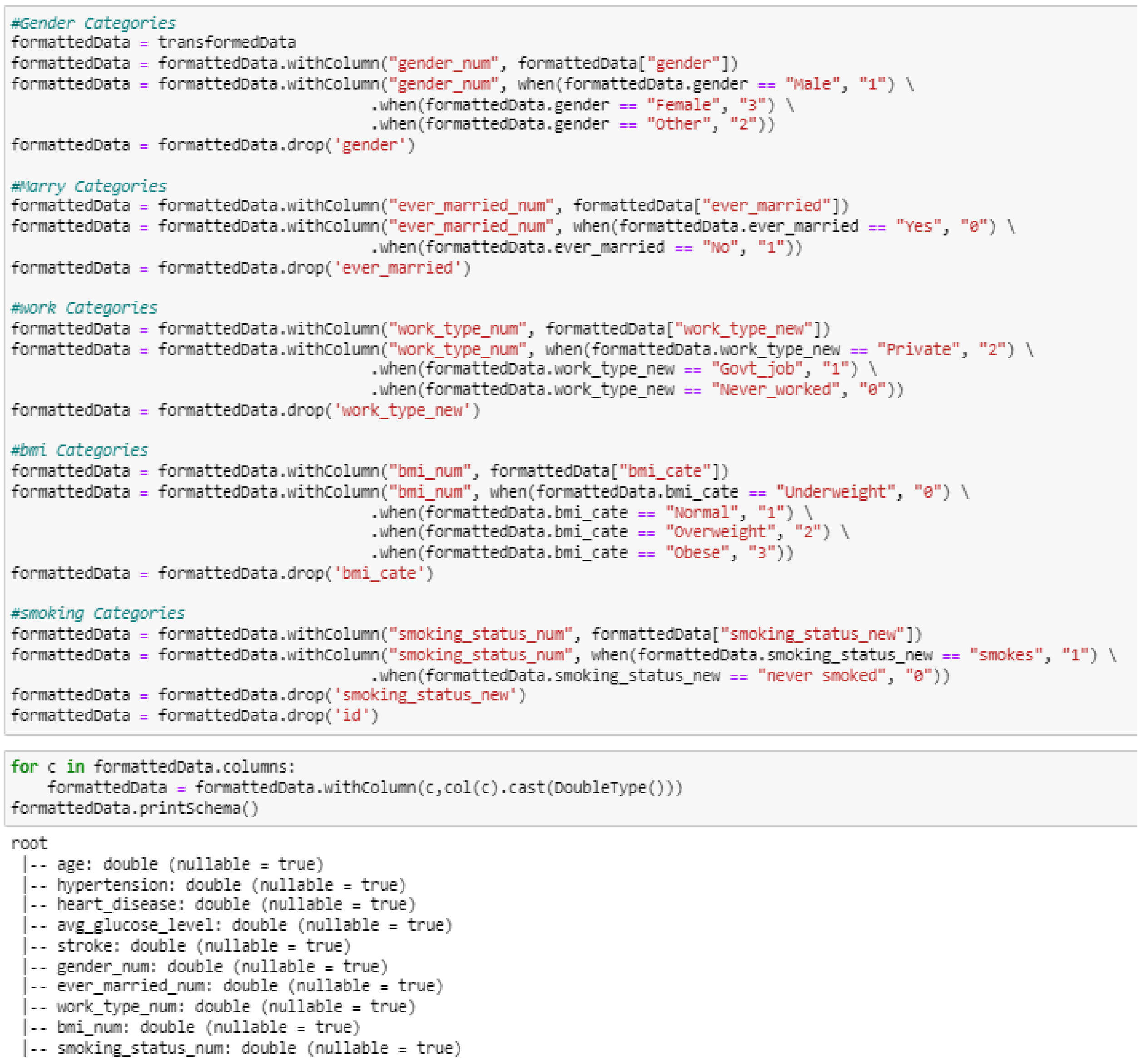
3.5 Formatting Data

Data sets have numeric and categorical features. Categorical features are string data types, which can be easily understood by humans. However, machines cannot interpret categorical data directly. Here, unlike the professional process application of SPSS modeler, categorical data needs to be converted to numerical data because many models/algorithms require that categories are represented numerically and can only be calculated and executed on a numerical basis. Based on this requirement, data formatting is applied to this dataset for further data mining. It is used the method of Label Encoding in python to convert string type to number.
In this dataset, there are 5 category or string attributes that need to be converted from strings to numbers, i.e., gender (male, female, other), ever married (yes, no), job type (private job,

never worked, government job), BMI (underweight, normal, overweight, obese) and smoking status (yes, no) need to be converted.
The issue to note here is that when transforming categorical features to numerical features, numbers are size differentiated, while gender and job type should not be size differentiated or unequal. So, we use the unique method to tag each value, where the number assigned to each value has no size or inequality but is just a tag for each value in the attribute.
Table 6 shows the reference table for these categories. Figure 33 shows the results of the implementation of the labelling process.

Gender			
Male	Female	Other	
1	3	2	
Work Type			
Private	Never Worked	Government Job	
2	2	1	
BMI			
Underweight	Normal	Overweight	Obese
0	1	2	3
Ever married			
Yes	No		
0	1		
Smoking status			
Smokes	Never smoke		
1	0		

Table 6. The data formatting referencing table



```
#Gender Categories
formattedData = transformedData
formattedData = formattedData.withColumn("gender_num", formattedData["gender"])
formattedData = formattedData.withColumn("gender_num", when(formattedData.gender == "Male", "1") \
                                .when(formattedData.gender == "Female", "3") \
                                .when(formattedData.gender == "Other", "2"))
formattedData = formattedData.drop('gender')

#Marry Categories
formattedData = formattedData.withColumn("ever_married_num", formattedData["ever_married"])
formattedData = formattedData.withColumn("ever_married_num", when(formattedData.ever_married == "Yes", "0") \
                                .when(formattedData.ever_married == "No", "1"))
formattedData = formattedData.drop('ever_married')

#work Categories
formattedData = formattedData.withColumn("work_type_num", formattedData["work_type_new"])
formattedData = formattedData.withColumn("work_type_num", when(formattedData.work_type_new == "Private", "2") \
                                .when(formattedData.work_type_new == "Govt_job", "1") \
                                .when(formattedData.work_type_new == "Never_worked", "0"))
formattedData = formattedData.drop('work_type_new')

#bmi Categories
formattedData = formattedData.withColumn("bmi_num", formattedData["bmi_cate"])
formattedData = formattedData.withColumn("bmi_num", when(formattedData.bmi_cate == "Underweight", "0") \
                                .when(formattedData.bmi_cate == "Normal", "1") \
                                .when(formattedData.bmi_cate == "Overweight", "2") \
                                .when(formattedData.bmi_cate == "Obese", "3"))
formattedData = formattedData.drop('bmi_cate')

#smoking Categories
formattedData = formattedData.withColumn("smoking_status_num", formattedData["smoking_status_new"])
formattedData = formattedData.withColumn("smoking_status_num", when(formattedData.smoking_status_new == "smokes", "1") \
                                .when(formattedData.smoking_status_new == "never smoked", "0"))
formattedData = formattedData.drop('smoking_status_new')
formattedData = formattedData.drop('id')
```

```
for c in formattedData.columns:
    formattedData = formattedData.withColumn(c,col(c).cast(DoubleType()))
formattedData.printSchema()
```

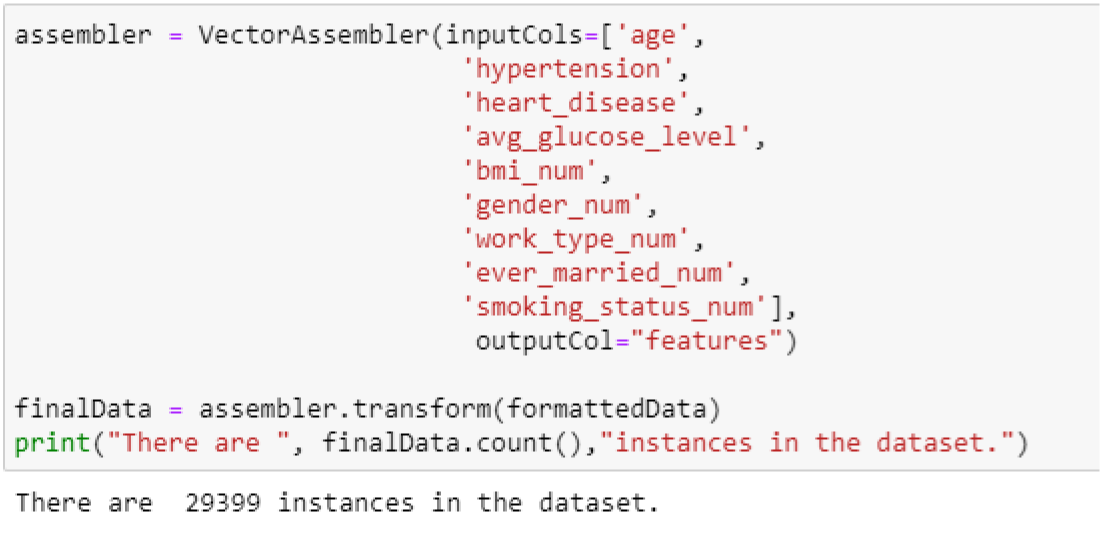
```
root
 |-- age: double (nullable = true)
 |-- hypertension: double (nullable = true)
 |-- heart_disease: double (nullable = true)
 |-- avg_glucose_level: double (nullable = true)
 |-- stroke: double (nullable = true)
 |-- gender_num: double (nullable = true)
 |-- ever_married_num: double (nullable = true)
 |-- work_type_num: double (nullable = true)
 |-- bmi_num: double (nullable = true)
 |-- smoking_status_num: double (nullable = true)
```

Figure 33. Python implementation of the labeling process

4. Data Transformation

4.1 Data Reduction

There are 14 properties in this dataset, including the three new ones created in 3.3. In this session, we will remove some attributes that are not used, such as BMI before update, smoking status before update, and job type before update; and id and target attribute stroke, which are not needed when training the model, because the association between id and stroke is 0, while the target attribute is the result.
After delete these 5 attributes, we get the dataset as below Figure 34. There are 9 attributes left, and 29,399 records.

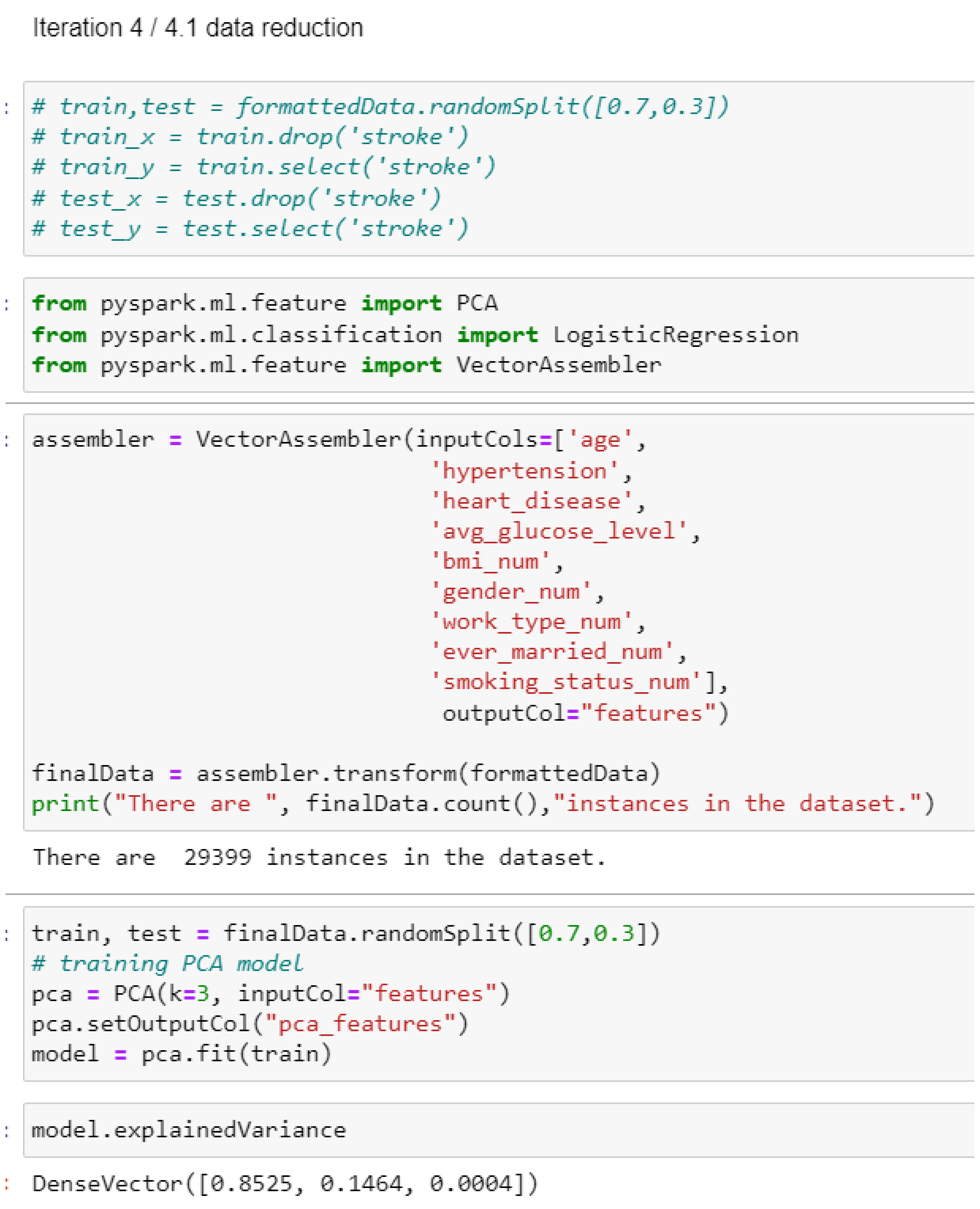


```
assembler = VectorAssembler(inputCols=['age',
                             'hypertension',
                             'heart_disease',
                             'avg_glucose_level',
                             'bmi_num',
                             'gender_num',
                             'work_type_num',
                             'ever_married_num',
                             'smoking_status_num'],
                              outputCol="features")

finalData = assembler.transform(formattedData)
print("There are ", finalData.count(),"instances in the dataset.")
```

```
There are  29399 instances in the dataset.
```

Figure 34. dataset after deleting unneeded attributes

Next, among these nine attributes, there are also attributes that need to be evaluated for some of the smaller contributions to the final results. Most of the time, these attributes can mislead the algorithm and ultimately reduce the overall accuracy of the model. Therefore, some amount of data reduction/dimensionality reduction is also required to eliminate the number of these attributes, using a method called PCA. principal component analysis (PCA) is a popular technique for analysing large datasets containing a large number of dimensions/features per observation, improving the interpretability of the data while retaining the maximum amount of information, and visualizing multidimensional data. Formally, PCA is a statistical technique that reduces the dimensionality of a data set. This is achieved by linearly transforming the data into a new coordinate system in which (most) changes in the data can be described in fewer dimensions than the initial data.
PCA is used for exploratory data analysis and building predictive models. It is typically used for dimensionality reduction, projecting each data point onto only the first few principal components to obtain low-dimensional data while preserving as much variation in the data as possible. The first principal component can be equivalently defined as the direction that maximizes the variance of the projected data. The i-th principal component the i-th principal component can be used as the direction orthogonal to the i-1st principal component that maximizes the variance of the projected data.
For either objective, it can be shown that the principal components are the eigenvectors of the data covariance matrix. Therefore, principal components are usually calculated by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix. PCA is the simplest of the true eigenvector-based multivariate analyses and is closely related to factor analysis. Factor analysis typically incorporates more domain-specific assumptions about the underlying structure and resolves the eigenvectors of a slightly different matrix.
We do the data reduction using PCA as shown in Figure 35. We project the 9 latitudes to a 3-dimensional data graph.



Iteration 4 / 4.1 data reduction

```
# train,test = formattedData.randomSplit([0.7,0.3])
# train_x = train.drop('stroke')
# train_y = train.select('stroke')
# test_x = test.drop('stroke')
# test_y = test.select('stroke')
```

```
from pyspark.ml.feature import PCA
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
```

```
assembler = VectorAssembler(inputCols=['age',
                            'hypertension',
                            'heart_disease',
                            'avg_glucose_level',
                            'bmi_num',
                            'gender_num',
                            'work_type_num',
                            'ever_married_num',
                            'smoking_status_num'],
                             outputCol="features")

finalData = assembler.transform(formattedData)
print("There are ", finalData.count(),"instances in the dataset.")
```

```
There are  29399 instances in the dataset.
```

```
train, test = finalData.randomSplit([0.7,0.3])
# training PCA model
pca = PCA(k=3, inputCol="features")
pca.setOutputCol("pca_features")
model = pca.fit(train)
```

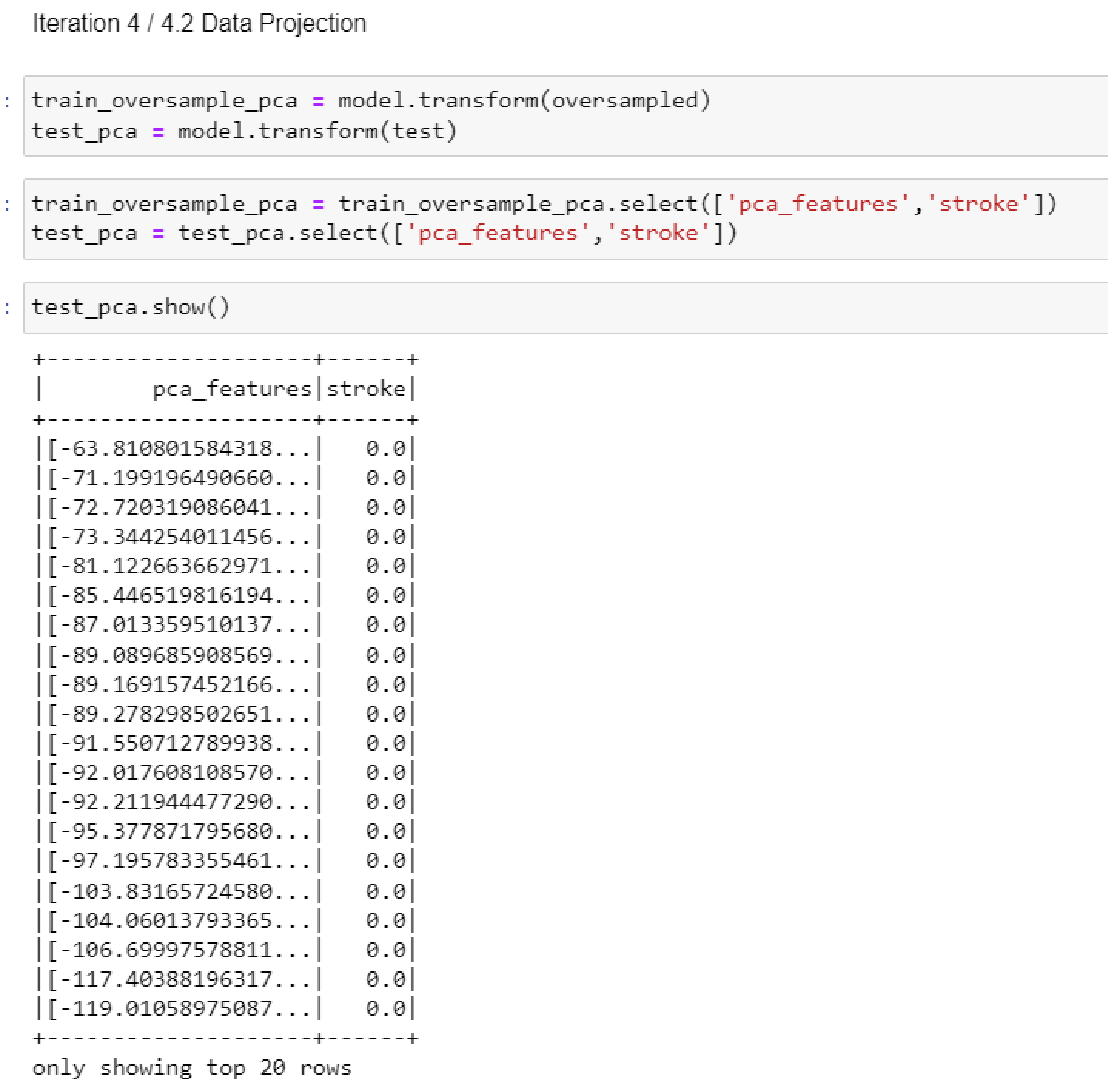
```
model.explainedVariance
```

```
DenseVector([0.8525, 0.1464, 0.0004])
```

Figure 35. Use PCA to do data reduction by reducing latitudes

4.2 Data Projection

As mentioned above, the data project after using PCA to do data reduction is as below in Figure 36. PCA evaluated the importance of each attribute for the target attribute and reduced the dataset from 9 latitudes to 3 latitudes.



Iteration 4 / 4.2 Data Projection

```
train_oversample_pca = model.transform(oversampled)
test_pca = model.transform(test)
```

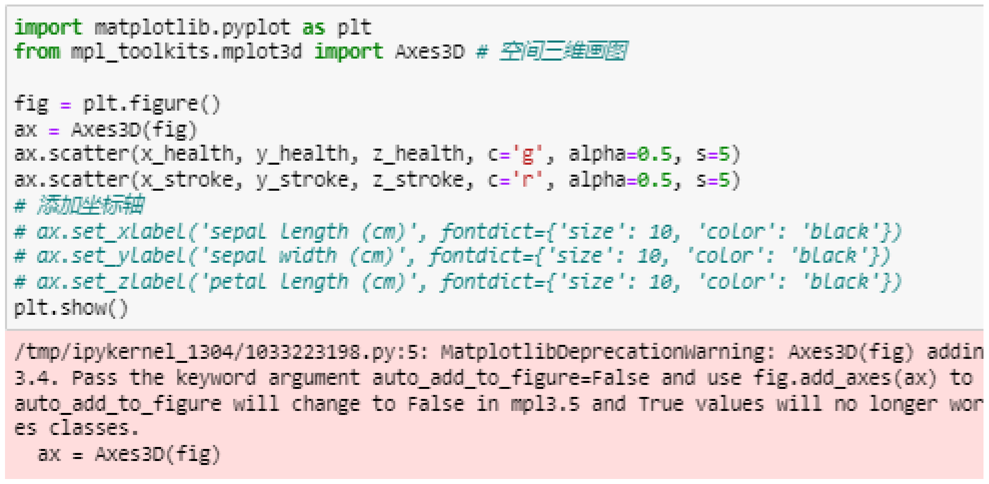
```
train_oversample_pca = train_oversample_pca.select(['pca_features','stroke'])
test_pca = test_pca.select(['pca_features','stroke'])
```

```
test_pca.show()
```

```
+--------------------+------+
|        pca_features|stroke|
+--------------------+------+
|[-63.810801584318...|   0.0|
|[-71.199196490660...|   0.0|
|[-72.720319086041...|   0.0|
|[-73.344254011456...|   0.0|
|[-81.122663662971...|   0.0|
|[-85.446519816194...|   0.0|
|[-87.013359510137...|   0.0|
|[-89.089685908569...|   0.0|
|[-89.169157452166...|   0.0|
|[-89.278298502651...|   0.0|
|[-91.550712789938...|   0.0|
|[-92.017608108570...|   0.0|
|[-92.211944477290...|   0.0|
|[-95.377871795680...|   0.0|
|[-97.195783355461...|   0.0|
|[-103.83165724580...|   0.0|
|[-104.06013793365...|   0.0|
|[-106.69997578811...|   0.0|
|[-117.40388196317...|   0.0|
|[-119.01058975087...|   0.0|
+--------------------+------+
only showing top 20 rows
```

Figure 36. Data visualization using PCA does data reduction

4.3 data visualization



```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D # 空间三维画图

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(x_health, y_health, z_health, c='g', alpha=0.5, s=5)
ax.scatter(x_stroke, y_stroke, z_stroke, c='r', alpha=0.5, s=5)
# 添加坐标轴
# ax.set_xlabel('sepal length (cm)', fontdict={'size': 10, 'color': 'black'})
# ax.set_ylabel('sepal width (cm)', fontdict={'size': 10, 'color': 'black'})
# ax.set_zlabel('petal length (cm)', fontdict={'size': 10, 'color': 'black'})
plt.show()
```

```
/tmp/ipykernel_1304/1033223198.py:5: MatplotlibDeprecationWarning: Axes3D(fig) addin
3.4. Pass the keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to
auto_add_to_figure will change to False in mpl3.5 and True values will no longer wor
es classes.
  ax = Axes3D(fig)
```

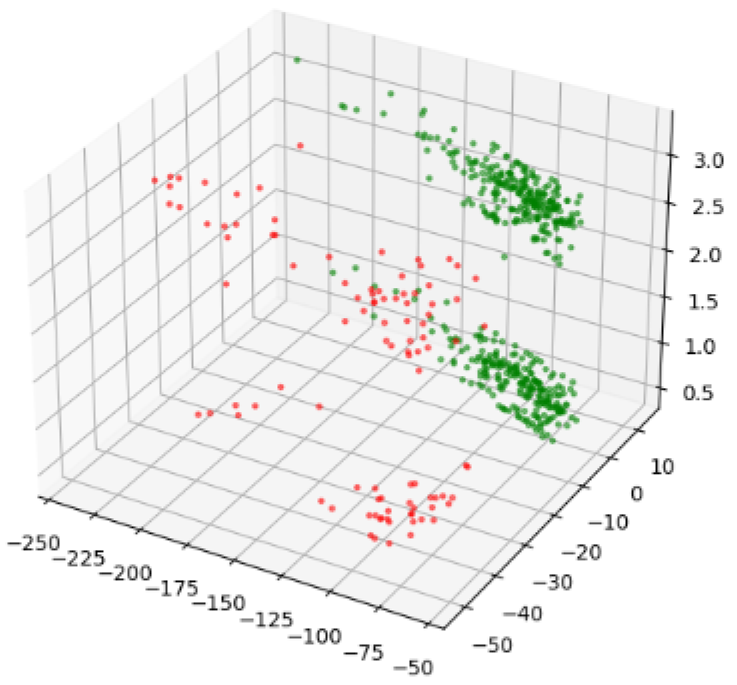


Figure 37. Result of using PCA in 3D-dimension

We can see a clearer relationship between reduced data dimensions in a 2D dimension, as below Figure 38, compared with 3D dimension Figure in Figure 37.

```
plt.scatter(x_health, y_health, c='g', alpha=0.5, s=5)
plt.scatter(x_stroke, y_stroke, c='r', alpha=0.5, s=5)
plt.show()
```

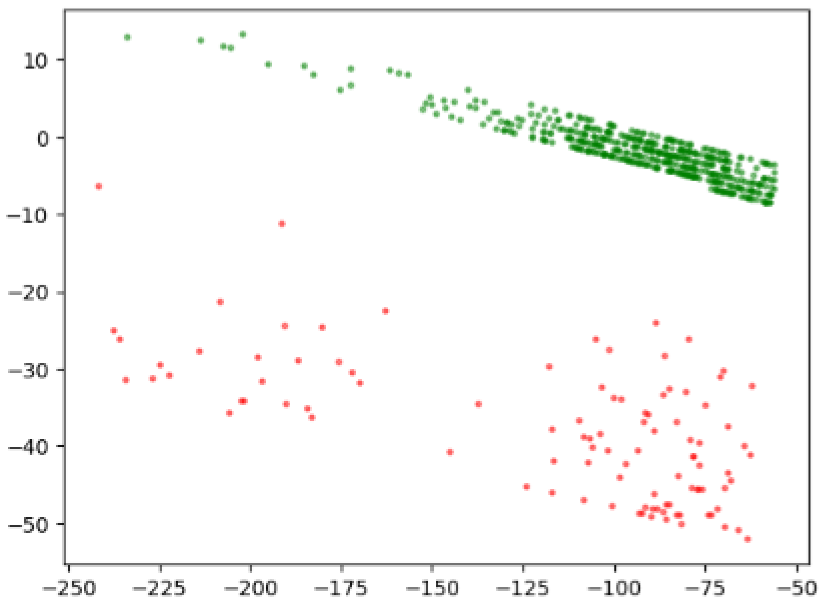


Figure 38. Result of using PCA in 2D-dimension

As described in previous chapters, the selected train dataset contains an unbalanced number of labels between the stroked and non-stroked instances. Figure 39 is a bar chart showing the difference between the two. Assume that these datasets are directly applied to the algorithm. In this case, the algorithm would obtain a low recall, i.e., the algorithm is good at detecting those non-stroke values, but very insensitive to those stroke data, since the stroke sample size is much too small to allow the algorithm to study these patterns.

```
There are 620 records are marked stroke.
There are 28779 records are marked as healthy (non-stroke).

<AxesSubplot:>
```

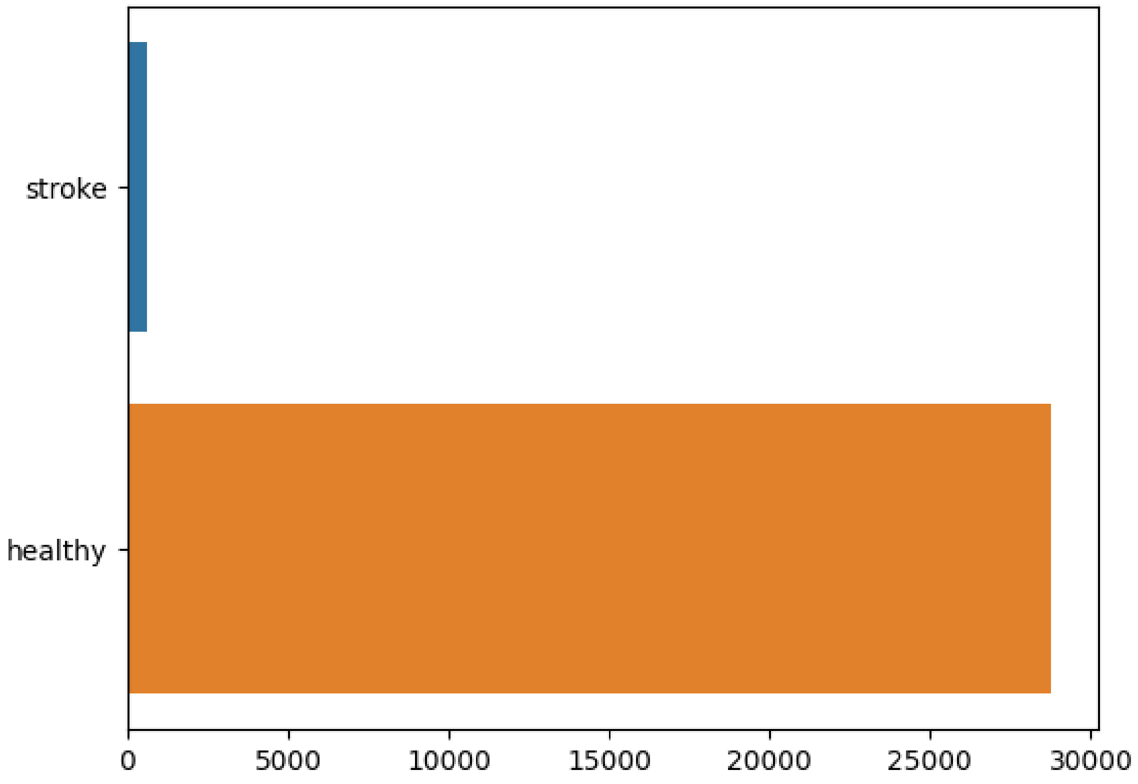


Figure 39. Stroke distribution in training dataset before re-balance

Based on this concern and considering trying to maintain the characteristics of the original data in the test set, we performed the rebalancing operation only on the training data. The dataset was further processed to fill in the gaps between the stroke and non-stroke data. In total, 620 cases were classified as stroke instances and 28, 779 cases were classified as non-stroke in the training set. On this basis, the oversampling method is more appropriate for this case compared to undersampling. It will increase the number of stroke instances without decreasing the number of non-stroke instances. Using this method, the algorithm itself will have a large enough record to find and refine the pattern.

As mentioned before, after rebalancing the stroke attributes inside the training set using the repeated algorithm, Figure 40 shows the histogram, which shows the number of stroke instances, which are balanced at this time. There are 20,217 stroke cases in the training dataset, and there are 20,217 non-stroke cases in the training dataset.

```
There are 20217 records are marked stroke.
There are 20217 records are marked as healthy (non-stroke).

<AxesSubplot:>
```

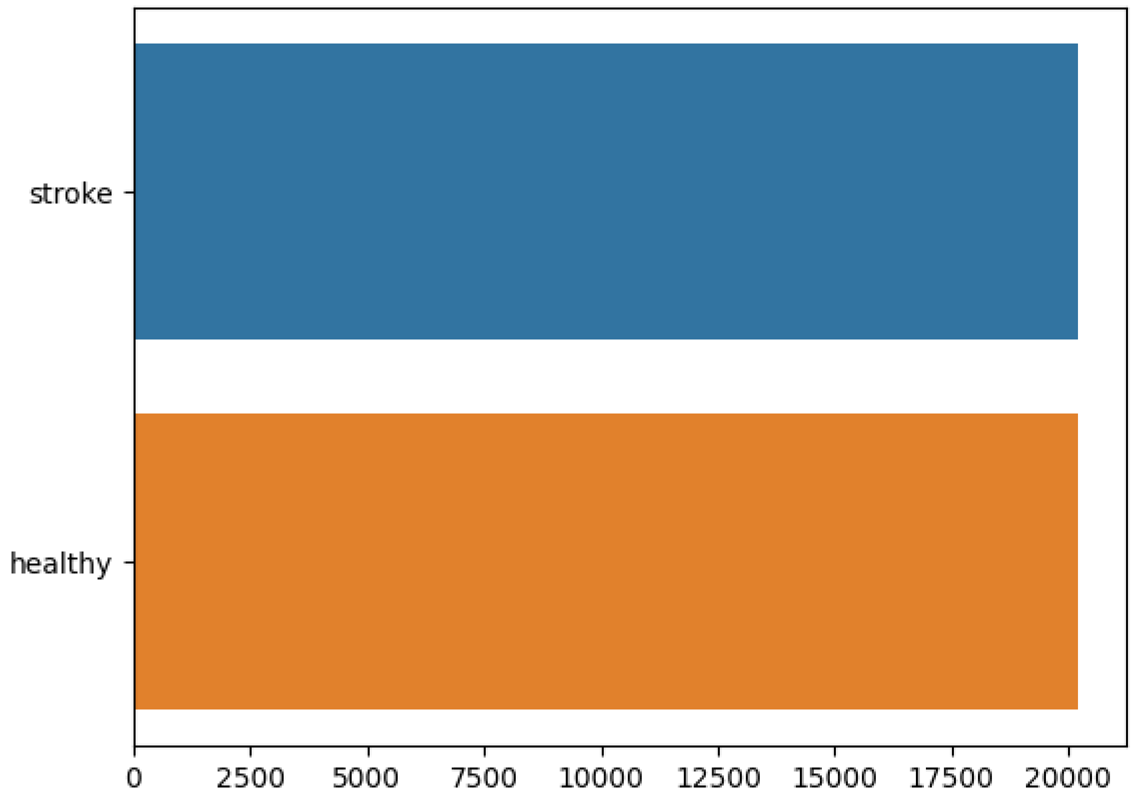


Figure 40. Stroke distribution in training dataset after re-balance

5. Data-Mining Method(s) Selection

5.1. Discussion of Data Mining Methods in Context of Data Mining Objectives

As mentioned earlier in the business objectives, we wanted to predict, prevent and reduce the occurrence of stroke disease based on the current status of the patient. In conjunction with the initial data mining goals, we want to clarify the relationship between a patient's current health index (current disease and BMI), age, and current life status (smoking status, marital status, type of work, living environment, etc.) and stroke. And by analyzing these relationships, the data is used to train and fit models suitable for data prediction, by providing these predictions to analyze the causes of current patient strokes and to reduce the chances of potential patient strokes at an early stage.

Before building the module, it is important to review our original intention of doing this study and choose to narrow or keep this goal for the module construction. The prediction system is based on a model that follows a number of patterns that are between the target and all relevant information relationships.

The assumptions of the modeling
In the preparation phase, nine attributes were selected: gender, age, average blood glucose level, hypertension, heart disease, whether married or not, whether smoking or not, type of work and weight type (BMI group), excluding the record identification number and the target attribute. We assume that these attributes are highly correlated with the target attributes, and the attributes that were removed and reconstructed for various reasons, the original smoking status of the dataset, job type and BMI, gender, etc., which will not affect the accuracy of the model's prediction of the target attributes. We keep as many attributes as possible and use pca's method to descend the latitude because having more information and keeping more attributes can provide more information for the model to train and understand the case more fully and build the model to predict more accurately.
Making predictions
The main function of this prediction system is to find the cause of stroke or to predict the level of stroke risk for an individual. In other words, it needs to collect information from individuals and use existing training models to assess the level of risk based on these collected data for consumption. The nine attributes to be predicted are to be collected as candidate factors or potential influences for stroke. The total number of instances prepared for the data mining process was 40,434 as shown in Figure 40.
And before we build a model for the prediction system, we need to create a test design for the validation of the prediction model. When creating the test design, the entire data will be used to divide the training set and the test set in the 7:3 ratio mentioned in section 3.4 and use the shuffle parameter inside python to make each test and training set division random as a way to better train the model.
Choosing the Right Modeling Techniques:

1. Association
   It is used to find correlations between two or more items by identifying hidden patterns in a dataset, hence the term relationship analysis. This method is used in shopping cart analysis to predict customer behavior. For example, the classic case, purchase (x, "beer") -> purchase (x, "diaper") [support = 1%, confidence = 50%], where x represents a customer who buys beer and diapers together. Confidence indicates that if a customer buys beer, there is a 50% chance that he/she will also buy diapers as well. Support implies that 1% of all transactions analyzed indicate that beer and diapers are purchased together.
   There are two types of association rules.
   Unidimensional association rules: These rules contain repeated individual attributes.
   Multidimensional association rules: These rules contain multiple attributes that are repeated.

2. Clustering
   A cluster is a collection of data objects; these objects are similar in the same cluster. This means that objects in the same group are similar to each other, and they are quite different, or they are different or unrelated to objects in other groups or other clusters. Cluster analysis is the process of discovering groups and clusters in the data, making the degree of association between two objects the highest if they

belong to the same group and the lowest between them otherwise. The results of this analysis can be used to create customer analysis.

3. Classification
   This data mining method is used to distinguish items in a dataset into classes or groups. It helps to accurately predict the behavior of entities within the group. It is a two-step process.
   Learning step (training phase): Here, the classification algorithm constructs the classifier by analyzing the training set.
   Classification step: Test data is used to estimate the accuracy or precision of the classification rules.

Conclusions are drawn based on the relationship between each field value and the final target. For example, a medical researcher analyzes cancer data to predict which drug to prescribe to a patient, groups patients according to their lives, and finds relationships or patterns between attributes.

5.2. Selecting the appropriate Data-Mining method(s)

According to the objectives of data mining mentioned in section 1.4, the three data mining methods mentioned in section 5.1, clustering, association and classification methods. Considering the suitability of this project, the classification method was selected as the data mining method.

Both classification methods are more suitable for this project than the other two methods, clustering and association. In terms of prediction of the target attributes, the association method analyses the relationship between two or more attributes and draws conclusions about association rules to predict the probability of one attribute occurring when another occurs. When there are multiple attributes, association methods are less sensitive to the data than classification methods. Classification methods, on the other hand, combine all the important attributes to make predictions about the future target data.

In terms of data grouping, association methods only absorb continuous values and make predictions. Classification methods also provide the ability to use non-continuous values as predictors. And a larger range of predictors can produce more accurate answers to this. However, cluster analysis, the process of discovering clusters and clustering in the data, is not compatible with our project objectives and is therefore not chosen.
In conclusion, the method of classification is the most suitable one for this case.

6. Data-Mining Algorithm(s) Selection

6.1 Algorithms analysis

According to the algorithm of Classification discussed in section 5.2, includes supervised learning and unsupervised learning algorithms. The difference between these two types is the target label. The information from supervised learning will be given a result label and it is

mainly used in terms of pattern search. Unsupervised learning will not be given any outcome labels and it focuses on the relationships in the dataset. It is mainly used in clustering methods (Mohamed, Yap, & Berry, 2019). In this data mining process, algorithms categorized as supervised learning will be used depending on the data mining objectives. Based on the limitations mentioned above, three data mining algorithms are being selected and discussed based on the data mining objectives.
Based on the above limitations, three data mining algorithms are being selected and discussed in accordance with the data mining objectives. The algorithms/models Random Tree, Random Forest, Logistic Regression, SVM and Neural network will be discussed.

1. **Random Tree**
   The random tree is a supervised classifier; it is an integrated learning algorithm that generates a large number of individual learners. It uses the idea of packing to construct a random set of data to build a decision tree. In a standard tree, each node uses the best split of all variables. In a random forest, each node is split using the best node in a randomly selected subset of predicates for that node.
   Leo Breiman and Adele Cutler (2001) introduced the random tree. This algorithm can handle classification and regression problems. The classification mechanism is as follows: the random tree classifier takes the input feature vector, classifies it with each tree in the forest, and outputs the class labels that receive the majority of "votes". In the case of regression, the classifier response is the average of the responses of all trees in the forest. Random trees are essentially a combination of two existing algorithms in machine learning: a single model tree merged with the idea of a random forest. Model trees are decision trees in which each leaf has a linear model that is optimized for the local subspace explained by that leaf.

2. **Random Forest**
   A random tree is a set (set) of tree predictors called a random forest. Random forests or stochastic decision forests are an integrated learning method for classification, regression and other tasks that operates by constructing multiple decision trees at training time. It also limits its drawbacks. Most importantly, different combinations of data are used to eliminate the uncertainty of a single decision tree (the order in which the data arrive makes the decision trees different). For classification tasks, the output of a random forest is the class of most tree choices. For regression tasks, the mean or average prediction of individual trees is returned. Random decision forests correct the habit of overfitting decision trees to their training set. Random forests usually outperform decision trees. However, data characteristics may affect its performance. Random forests have been shown to greatly improve the performance of individual decision trees: tree diversity is created by two types of randomizations. First, the training data is sampled, and each tree is replaced. Second, instead of always computing the best split for each node when planting a tree, a random subset of all attributes is considered at each node and the best split for that subset is computed. These trees combine for the first-time model trees and random forests for classifying stochastic model trees. Random trees use this product for split selection to induce reasonably

balanced trees where one global setting of ridge values applies to all leaves, thus simplifying the optimization process.

3. **Logistic Regression**
   This type of statistical model (also known as a logit model) is commonly used for classification and predictive analysis. Logistic regression estimates the probability of an event occurring, such as a vote or non-vote, based on a given data set of independent variables. As the outcome is probabilistic, the dependent variable ranges between 0 and 1. In logistic regression, a logit transformation is applied to the probabilities, i.e. the probability of success divided by the probability of failure. This is also commonly referred to as the logit odds ratio, or the natural logarithm of the odds ratio, and the logistic function is given by
   $Logit(pi) = 1/(1+ exp(-pi))$
   $ln(pi/(1-pi)) = Beta_0 + Beta_1*X_1 + ... + B_k*K_k$
   In this logistic regression equation, logit(pi) is the dependent or response variable and x is the independent variable. The beta parameter or coefficient in this model is usually estimated by maximum likelihood estimation (MLE). This method tests different beta values through several iterations to optimise the best fit to the log odds. All these iterations produce a log-likelihood function, which logistic regression attempts to maximise in order to find the best estimate of the parameters. Once the best coefficient (or coefficients, if there are multiple independent variables) is found, the conditional probabilities for each observation can be calculated, recorded and summed to produce predicted probabilities. For binary classification as our project, a probability of less than 0.5 will predict 0, while a probability greater than 0 will predict 1. After calculating the model, best practice is to assess how well the model predicts the dependent variable, which is called goodness of fit.

4. **SVM**
   SVC, or support vector classifier, is a supervised machine learning algorithm typically used for classification tasks. SVC works by mapping data points into a high-dimensional space and then finding the best hyperplane to classify the data into two classes. SVM, or support vector machine, is a powerful model that looks to separate data with invisible lines. It ensures that the distance between two types of data is uniform and away from each other. It is also able to do nonlinear separation by projecting the data into three dimensions and separating them. This model can be used because it meets the goals of data mining and is able to find patterns.
   SVM is particularly suitable for generalized datasets, that is, datasets with a large number of prediction domains. A classification algorithm is usually used to deal with two sets of classification problems. After giving SVM models a labelled training data set for each category, they are able to classify new text. They have two main advantages over newer algorithms such as neural networks: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithms very suitable for text classification problems, where datasets with up to a few thousand labelled samples are usually available.

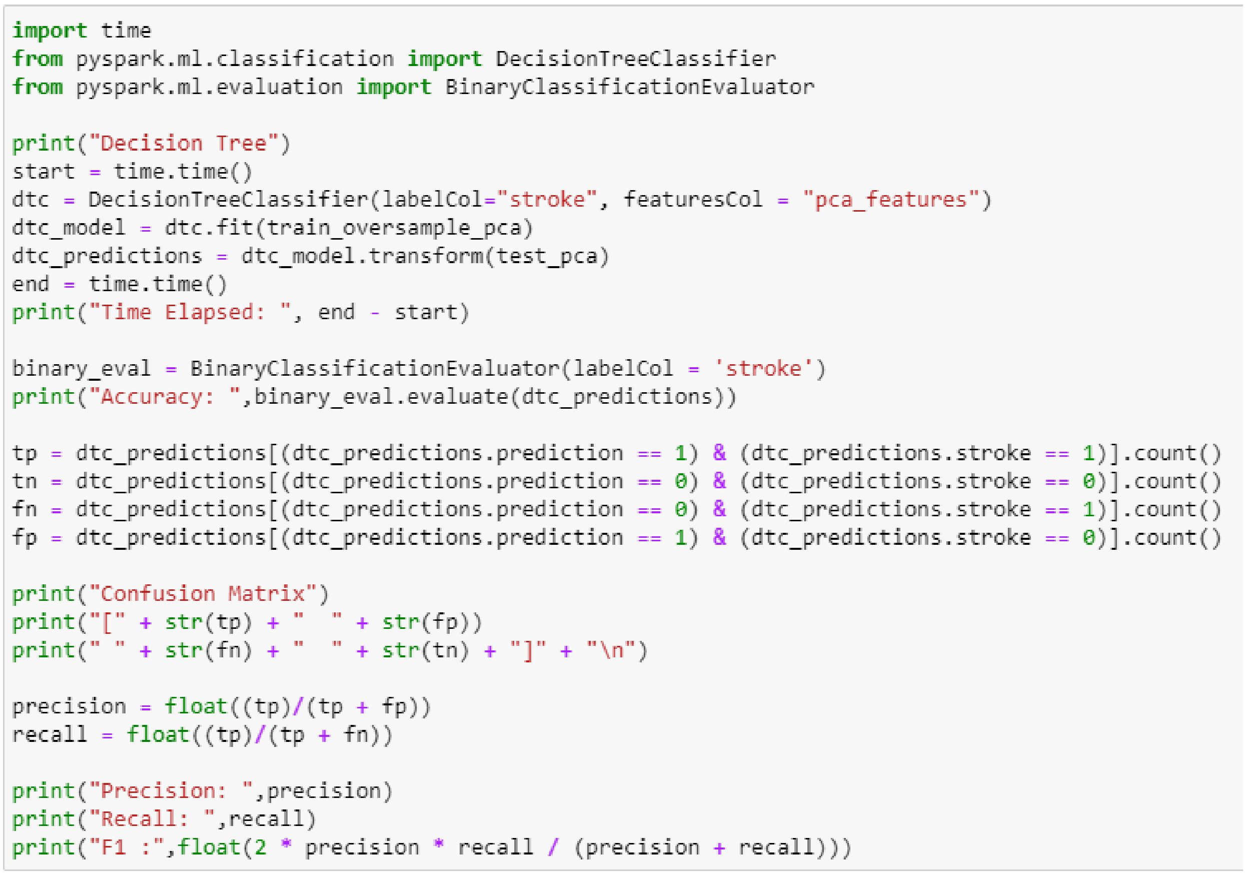
5. **Neural networks**,
   Also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain and mimic the way biological neurons send signals to each other.
   An artificial neural network (ANN) consists of a node layer containing an input layer, one or more hidden layers, and an output layer. Each node or artificial neuron is connected to another node and has an associated weight and threshold. If the output of any individual node is above a specified threshold, that node is activated, and the data is sent to the next layer of the network. Otherwise, no data is passed to the next layer of the network. Over time, neural networks rely on training data to learn and improve their accuracy. However, once these learning algorithms have been fine-tuned to improve accuracy, they become powerful tools in computer science and artificial intelligence, enabling us to classify and cluster data at high speed. Speech recognition or image recognition tasks can take minutes rather than hours compared to manual recognition by a human expert. Neural networks work by treating each individual node as its own linear regression model, consisting of input data, weights, deviations (or thresholds), and outputs.

All of those mentioned algorithms will be experienced in pySpark using models in the Spark, and the performance and accuracy of each algorithm will be evaluated in the following chapters.

6.2 Algorithms analysis

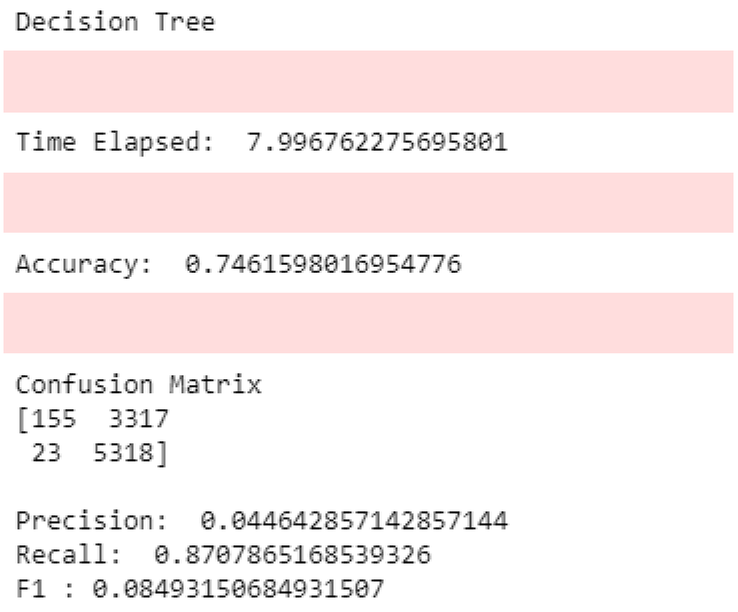
1. Random Tree
   According to DM objectives, the decision tree divides the importance level of attributes based on the attributes of this dataset, with the most important attribute as the first branch node condition, and then branches down in layers to partition the best nodes in the subset. Each leaf is a case. It is possible to clearly show what the most influential predictors of stroke are for the target attribute, stroke.

   Figure 41 and Figure 42 show the random tree model-building process and the overall accuracy, confusion matrix, recall score, and time consumption used.



```python
import time
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator

print("Decision Tree")
start = time.time()
dtc = DecisionTreeClassifier(labelCol="stroke", featuresCol = "pca_features")
dtc_model = dtc.fit(train_oversample_pca)
dtc_predictions = dtc_model.transform(test_pca)
end = time.time()
print("Time Elapsed: ", end - start)

binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
print("Accuracy: ",binary_eval.evaluate(dtc_predictions))

tp = dtc_predictions[(dtc_predictions.prediction == 1) & (dtc_predictions.stroke == 1)].count()
tn = dtc_predictions[(dtc_predictions.prediction == 0) & (dtc_predictions.stroke == 0)].count()
fn = dtc_predictions[(dtc_predictions.prediction == 0) & (dtc_predictions.stroke == 1)].count()
fp = dtc_predictions[(dtc_predictions.prediction == 1) & (dtc_predictions.stroke == 0)].count()

print("Confusion Matrix")
print("[" + str(tp) + "  " + str(fp))
print(" " + str(fn) + "  " + str(tn) + "]" + "\n")

precision = float((tp)/(tp + fp))
recall = float((tp)/(tp + fn))

print("Precision: ",precision)
print("Recall: ",recall)
print("F1 :",float(2 * precision * recall / (precision + recall)))
```

Figure 41. Random tree process

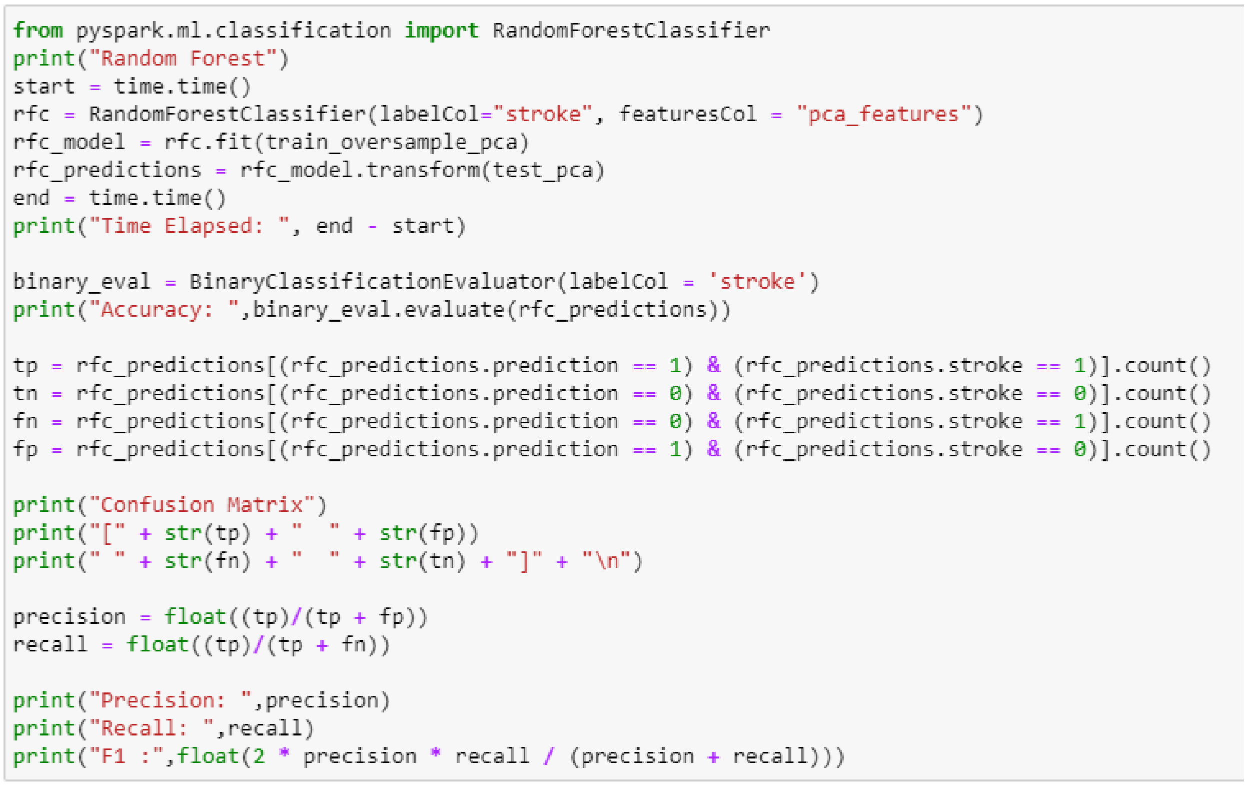


```
Decision Tree

Time Elapsed:  7.996762275695801

Accuracy:  0.7461598016954776

Confusion Matrix
[155  3317
 23  5318]

Precision:  0.044642857142857144
Recall:  0.8707865168539326
F1 : 0.08493150684931507
```

Figure 41. Random tree model result

We can see that the accuracy is around 75%, and the recall is 87.1%, which is not bad, the model is learning from the dataset, and make useful stroke prediction information. However, the precision and F1 score are very low, these 2 scores cannot present the quality of the model in this case. Because precision calculates the rate of true positive of stroke divided by the sum of true positive and false positive in the test dataset, but the test dataset still has the stroke label uneven problem, as we did not do rebalance process for the test dataset. And the F1 score calculates from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores). So, if the precision score is low will lower the F1 score.
In this case, the low precision score means the model predicts many people who do not have strokes as having strokes, a result that is better than predicting people who do have strokes as not having strokes. This suggests that many people who have a stroke have very similar characteristics to those who do not, or that those who are predicted to have a stroke but do not actually have a stroke are at high risk of stroke and need to be better protected against it.

2. Random Forest
   A decision forest is a collection of multiple decision trees and tends to perform better than a decision tree.
   Figure 42 and Figure 43 show the overall accuracy, confusion matrix, recall score, and time consumption used for the random forest.



```
from pyspark.ml.classification import RandomForestClassifier
print("Random Forest")
start = time.time()
rfc = RandomForestClassifier(labelCol="stroke", featuresCol = "pca_features")
rfc_model = rfc.fit(train_oversample_pca)
rfc_predictions = rfc_model.transform(test_pca)
end = time.time()
print("Time Elapsed: ", end - start)

binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
print("Accuracy: ",binary_eval.evaluate(rfc_predictions))

tp = rfc_predictions[(rfc_predictions.prediction == 1) & (rfc_predictions.stroke == 1)].count()
tn = rfc_predictions[(rfc_predictions.prediction == 0) & (rfc_predictions.stroke == 0)].count()
fn = rfc_predictions[(rfc_predictions.prediction == 0) & (rfc_predictions.stroke == 1)].count()
fp = rfc_predictions[(rfc_predictions.prediction == 1) & (rfc_predictions.stroke == 0)].count()

print("Confusion Matrix")
print("[" + str(tp) + "  " + str(fp))
print(" " + str(fn) + "  " + str(tn) + "]" + "\n")

precision = float((tp)/(tp + fp))
recall = float((tp)/(tp + fn))

print("Precision: ",precision)
print("Recall: ",recall)
print("F1 :",float(2 * precision * recall / (precision + recall)))
```

Figure 42. Random forest process

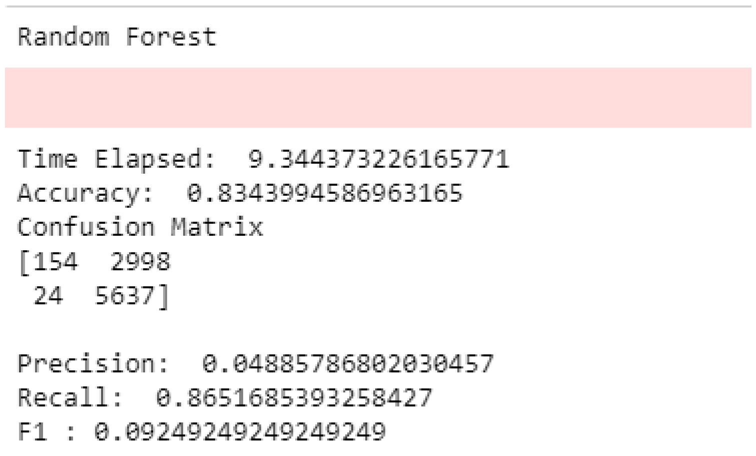

```
Random Forest

Time Elapsed:  9.344373226165771
Accuracy:  0.8343994586963165
Confusion Matrix
[154  2998
 24  5637]

Precision:  0.04885786802030457
Recall:  0.8651685393258427
F1 : 0.09249249249249249
```

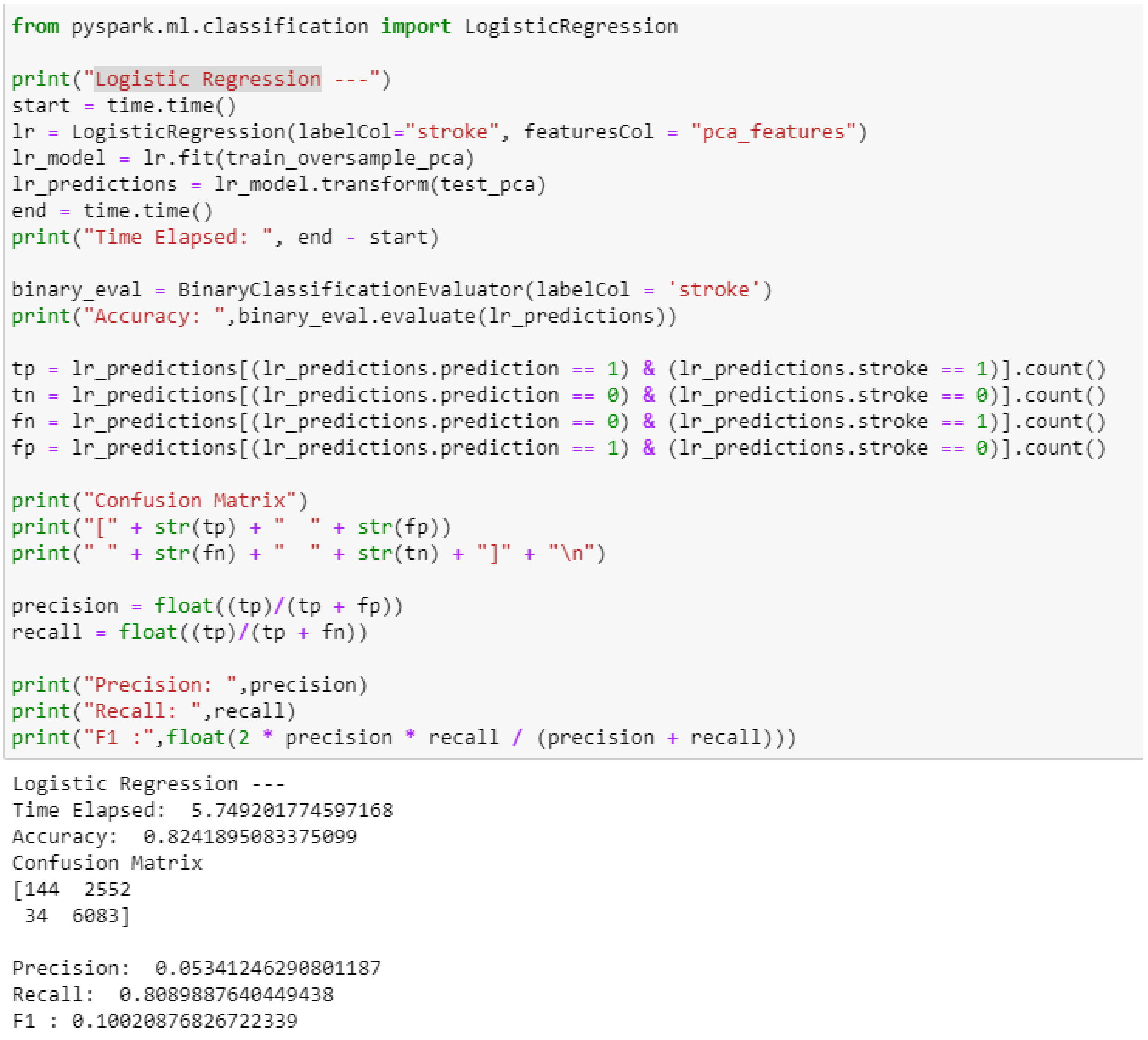
Figure 43. Random forest model result

We can see that the accuracy is 83.4%, the recall score is 86.52%. Although the random forest improves the performance compare with random tree, but the improvement is small, the model needs to be improved more.

The precision and F1 score are very low, these 2 scores cannot present the quality of the model in this case. Because precision calculates the rate of true positive of stroke divided by the sum of true positive and false positive in the test dataset, but the test dataset still has the stroke label uneven problem, as we did not do rebalance process for the test dataset. And the F1 score calculates from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores). So, if the precision score is low will lower the F1 score.

In this case, the low precision score means the model predicts many people who do not have strokes as having strokes, a result that is better than predicting people who do have strokes as not having strokes. This suggests that many people who have a stroke have very similar characteristics to those who do not, or that those who are predicted to have a stroke but do not actually have a stroke are at high risk of stroke and need to be better protected against it.

3. Logic regression
   Logic regression model analysis of all attributes and predict for the target attribute stroke, if the risk of stroke occurrence is over 0.5, the prediction is 1, if the risk of stroke occurrence is below 0.5, the prediction is 0.



```
from pyspark.ml.classification import LogisticRegression

print("Logistic Regression ---")
start = time.time()
lr = LogisticRegression(labelCol="stroke", featuresCol = "pca_features")
lr_model = lr.fit(train_oversample_pca)
lr_predictions = lr_model.transform(test_pca)
end = time.time()
print("Time Elapsed: ", end - start)

binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
print("Accuracy: ",binary_eval.evaluate(lr_predictions))

tp = lr_predictions[(lr_predictions.prediction == 1) & (lr_predictions.stroke == 1)].count()
tn = lr_predictions[(lr_predictions.prediction == 0) & (lr_predictions.stroke == 0)].count()
fn = lr_predictions[(lr_predictions.prediction == 0) & (lr_predictions.stroke == 1)].count()
fp = lr_predictions[(lr_predictions.prediction == 1) & (lr_predictions.stroke == 0)].count()

print("Confusion Matrix")
print("[" + str(tp) + "  " + str(fp))
print(" " + str(fn) + "  " + str(tn) + "]" + "\n")

precision = float((tp)/(tp + fp))
recall = float((tp)/(tp + fn))

print("Precision: ",precision)
print("Recall: ",recall)
print("F1 :",float(2 * precision * recall / (precision + recall)))
```

```
Logistic Regression ---
Time Elapsed:  5.749201774597168
Accuracy:  0.8241895083375099
Confusion Matrix
[144  2552
 34  6083]

Precision:  0.05341246290801187
Recall:  0.8089887640449438
F1 : 0.10020876826722339
```

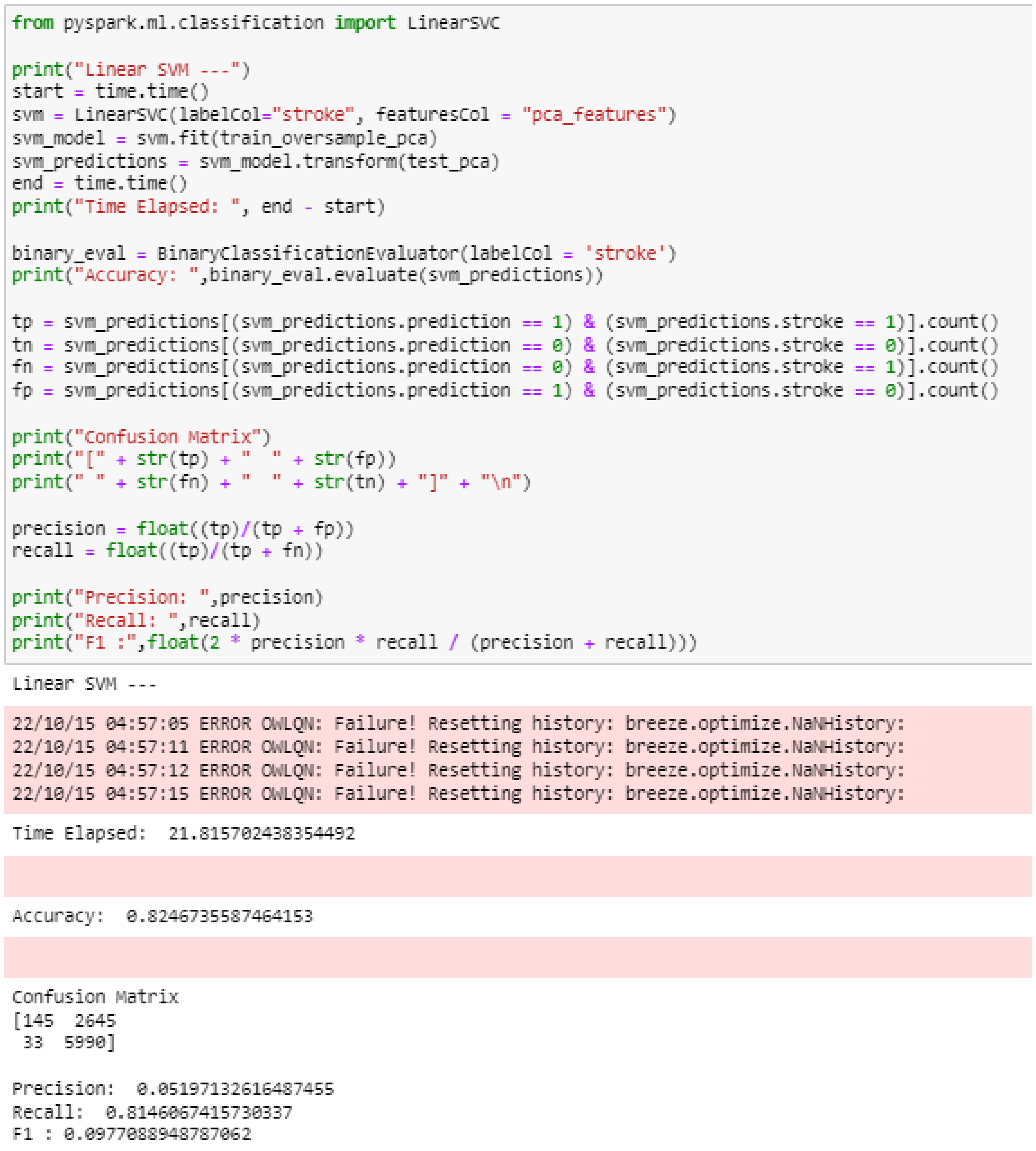
Figure 44. Logic regression model process and result

We can see that the accuracy is 82.4%, and the recall score is 80.9%, which performance worse compare with random forest.

The precision and F1 score are very low, these 2 scores cannot present the quality of the model in this case. Because precision calculates the rate of true positive of stroke divided by the sum of true positive and false positive in the test dataset, but the test dataset still has the stroke label uneven problem, as we did not do rebalance process for the test dataset. And the F1 score calculates from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores). So, if the precision score is low will lower the F1 score.

In this case, the low precision score means the model predicts many people who do not have strokes as having strokes, a result that is better than predicting people who do have strokes as not having strokes. This suggests that many people who have a stroke have very similar characteristics to those who do not, or that those who are predicted to have a stroke but do not actually have a stroke are at high risk of stroke and need to be better protected against it.

4. SVM
   According to DM objectives, SVM is a powerful model that wants to separate data with invisible lines. It ensures that the distance It ensures that the distance between the two types of data is uniform and away from each other. It is also able to do nonlinear separation by projecting the data into 3D space and separating them. It is a good choice for the database of this project, which is difficult to classify from a spatial perspective. Figure 45 shows the overall accuracy, confusion matrix, recall score, and time consumption used for the linear SVM model.



```
from pyspark.ml.classification import LinearSVC

print("Linear SVM ---")
start = time.time()
svm = LinearSVC(labelCol="stroke", featuresCol = "pca_features")
svm_model = svm.fit(train_oversample_pca)
svm_predictions = svm_model.transform(test_pca)
end = time.time()
print("Time Elapsed: ", end - start)

binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
print("Accuracy: ",binary_eval.evaluate(svm_predictions))

tp = svm_predictions[(svm_predictions.prediction == 1) & (svm_predictions.stroke == 1)].count()
tn = svm_predictions[(svm_predictions.prediction == 0) & (svm_predictions.stroke == 0)].count()
fn = svm_predictions[(svm_predictions.prediction == 0) & (svm_predictions.stroke == 1)].count()
fp = svm_predictions[(svm_predictions.prediction == 1) & (svm_predictions.stroke == 0)].count()

print("Confusion Matrix")
print("[" + str(tp) + "  " + str(fp))
print(" " + str(fn) + "  " + str(tn) + "]" + "\n")

precision = float((tp)/(tp + fp))
recall = float((tp)/(tp + fn))

print("Precision: ",precision)
print("Recall: ",recall)
print("F1 :",float(2 * precision * recall / (precision + recall)))
```

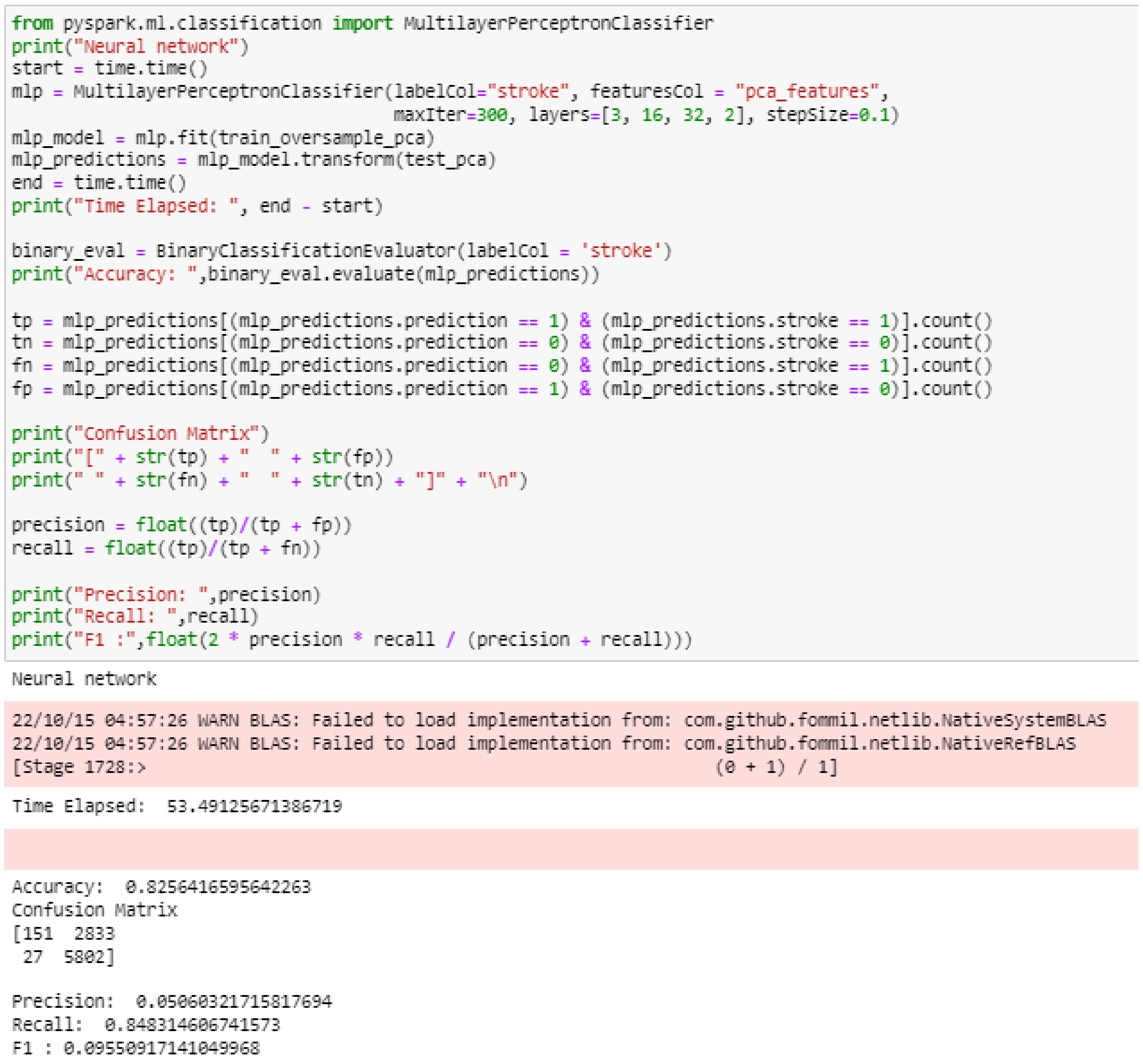
```
Linear SVM ---
22/10/15 04:57:05 ERROR OWLQN: Failure! Resetting history: breeze.optimize.NaNHistory:
22/10/15 04:57:11 ERROR OWLQN: Failure! Resetting history: breeze.optimize.NaNHistory:
22/10/15 04:57:12 ERROR OWLQN: Failure! Resetting history: breeze.optimize.NaNHistory:
22/10/15 04:57:15 ERROR OWLQN: Failure! Resetting history: breeze.optimize.NaNHistory:
Time Elapsed:  21.815702438354492

Accuracy:  0.8246735587464153

Confusion Matrix
[145  2645
 33  5990]

Precision:  0.05197132616487455
Recall:  0.8146067415730337
F1 : 0.0977088948787062
```

Figure 45. SVM model process and result

We can see that the accuracy is 82.5%, the recall score is 81.46%, which is not bad, the model is learning from the dataset, and make useful stroke prediction information. However, the precision and F1 score are very low, these 2 scores cannot present the quality of the model in this case. Because precision calculates the rate of true positive of stroke divided by the sum of true positive and false positive in the test dataset, but the test dataset still has the stroke label uneven problem, as we did not do rebalance process for the test dataset. And the F1 score calculates from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores). So, if the precision score is low will lower the F1 score.

In this case, the low precision score means the model predicts many people who do not have strokes as having strokes, a result that is better than predicting people who do have strokes as not having strokes. This suggests that many people who have a stroke have very similar characteristics to those who do not, or that those who are predicted to have a stroke but do not actually have a stroke are at high risk of stroke and need to be better protected against it.

5. Neural network
   In data mining, it is not enough to have a large amount of sample data, we also need an expression of a "universal function". In order to obtain a "universal function", a neural network model is designed based on our brain. The brain does not have a pre-stored model of functions for classifying various things, but rather 100 billion (11 times 10) neurons. The aggregation of a large number of units with a single function is capable of producing extremely complex functions. Neurons are to the human brain what transistors are to the CPU.
   In 1989, George Cybenko proved the universality theorem for neural networks: no matter how complex the function f(x), there is always a neural grid that can output f(x) or an approximation to it with sufficient accuracy for any possible input x.
   After a neural network has undergone a process called 'training', where the parameters of each neuron are gradually determined, meaning that f is determined and that for a known (x, y), $y \approx f(x)$ holds, we consider that the machine has automatically 'learned' "(find) the model. At this point, the nerve and the parameters determined for each neuron are now in the form of the model itself, and can be saved for use in prediction. Prediction here means that for a new sample x input, we do not know in advance what the corresponding outcome y will be, and can then use our trained f (neural network), to solve for y.



```
from pyspark.ml.classification import MultilayerPerceptronClassifier
print("Neural network")
start = time.time()
mlp = MultilayerPerceptronClassifier(labelCol="stroke", featuresCol = "pca_features",
                                     maxIter=300, layers=[3, 16, 32, 2], stepSize=0.1)
mlp_model = mlp.fit(train_oversample_pca)
mlp_predictions = mlp_model.transform(test_pca)
end = time.time()
print("Time Elapsed: ", end - start)

binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
print("Accuracy: ",binary_eval.evaluate(mlp_predictions))

tp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 1)].count()
tn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 0)].count()
fn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 1)].count()
fp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 0)].count()

print("Confusion Matrix")
print("[" + str(tp) + "  " + str(fp))
print(" " + str(fn) + "  " + str(tn) + "]" + "\n")

precision = float((tp)/(tp + fp))
recall = float((tp)/(tp + fn))

print("Precision: ",precision)
print("Recall: ",recall)
print("F1 :",float(2 * precision * recall / (precision + recall)))
```

```
Neural network
22/10/15 04:57:26 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
22/10/15 04:57:26 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
[Stage 1728:>                                                        (0 + 1) / 1]
Time Elapsed:  53.49125671386719

Accuracy:  0.8256416595642263
Confusion Matrix
[151  2833
 27  5802]

Precision:  0.050603217158176944
Recall:  0.848314606741573
F1 : 0.09550917141049968
```

Figure 46. Neural network model process and result

We can see that the accuracy is 82.56%, and the recall score is 84.83%, which is a very good performance, the model is learning from the dataset well, and make useful stroke prediction information. However, the precision and F1 score are very low, these 2 scores cannot present the quality of the model in this case. Because precision calculates the rate of true positive of stroke divided by the sum of true positive and false positive in the test dataset, but the test dataset still has the stroke label uneven problem, as we did not do rebalance process for the test dataset. And the F1 score calculates from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores). So, if the precision score is low will lower the F1 score.
In this case, the low precision score means the model predicts many people who do not have strokes as having strokes, a result that is better than predicting people who do have strokes as not having strokes. This suggests that many people who have a stroke have very similar

characteristics to those who do not, or that those who are predicted to have a stroke but do not actually have a stroke are at high risk of stroke and need to be better protected against it.

Compare with other models, we discussed before, Random Forest and Neural Network models perform well and have similar accuracy scores and recall scores. We would choose the Neural Network model as our decision for this project, as the deep learning model is more suitable when we do predictions for medical cases because it is more stable than other models.
We can improve the prediction accuracy score and recall score by adjusting the parameters of this algorithm in the following section.

6.3 Parameter Tuning

Most machine learning and deep learning algorithms have some parameters that can be tuned, called hyperparameters. Before training the model, we need to set the hyperparameters. Hyperparameters are crucial for building robust and accurate models. By helping us find a balance between bias and variance, they prevent the model from being overfitted or under fitted. In order to be able to tune hyperparameters, we need to understand what they mean and how they change the model.

We will choose the neural network model for our data mining objective, as it is the most suitable model for our project. It has the most satisfiable performance (accuracy score: 82.6%, recall score: 84.5%) compare with random tree (accuracy score: 74.6%, recall score: 87.1%), random forest (accuracy score: 83.4%, recall score: 86.52%), logic regression (accuracy score: 82.4%, recall score: 80.1%) and SVM (accuracy score: 82.5%, recall score: 81.4%).

The neural network was chosen as the algorithm for the following data mining process. Although the current accuracy is not high, there is still room for further improvement and parameter tuning. The algorithm has several parameters to control its performance of the algorithm.

In this section, Gradient Descent is a type of iterative method that can be used to solve least squares problems (both linear and non-linear). Gradient Descent is one of the most commonly used methods for solving model parameters of machine learning algorithms, i.e. unconstrained optimization problems, and another commonly used method is least squares. When solving for the minimum of the loss function, the gradient descent method can be used to solve the problem iteratively, step by step, to obtain the minimized loss function and model parameters. Conversely, if we need to solve for the maximum value of the loss function, it is then necessary to iterate with gradient ascent. In machine learning, two gradient descent methods have been developed based on the basic gradient descent method, namely stochastic gradient descent (SGD) and batch gradient descent.

We use stochastic gradient descent in this case, and there are 3 parameters that will be discussed, the reason to choose those three parameters is also discussed below.

The stochastic gradient descent algorithm updates the model parameters one sample at a time, so each learning is very fast and online updates can be performed. The biggest disadvantage is that each update may not go in the right direction, resulting in sharp fluctuations (perturbations) in the objective function, but on the other hand, the fluctuations caused by stochastic gradient descent have the advantage that for basin-like regions (i.e. many local minima) then the fluctuations may lead to a jump from the current local minima to another better local minima, eventually converging at a better local minima, or even a global minima. Due to the fluctuations, the number of iterations of learning will increase, i.e. convergence becomes slower.

The maximum number of runs of gradient descent：

The maximum number of runs of gradient descent stops when the maximum number of runs is reached, even if there is no convergence. We can do this by limiting the maximum number of runs of gradient descent so that the effect of a certain excessive error is limited, and this avoids excessive changes in the direction of the iteration.

step size：

the step size of each parameter updates for gradient descent. To minimize the fluctuations, we can adjust the step size. A larger step size avoids oscillations to some extent and allows walking over the local minima toward the larger extreme value points.

The structure of the neural network:

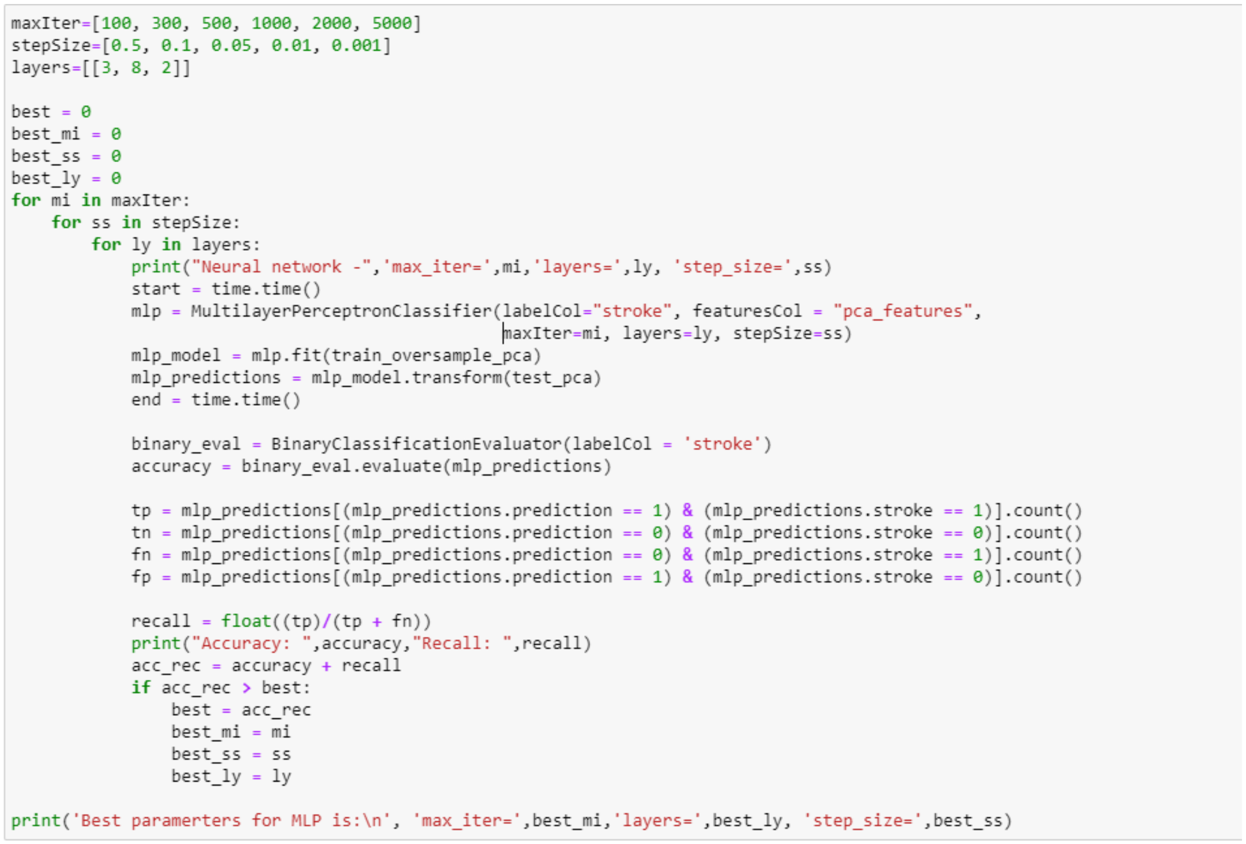
The structure of the neural network, how many layers there are, and how many neurons are in each layer. The length of the list means how many layers the neural network has, and the value in the list means how many neurons are in the layer. The structure of the neural network affects the capacity, the more complex the neural network, the more complex the function that can be fitted.

The parameters of the max number of runs, step size, and the structure of the neural network is being chosen, and set up a test combination to evaluate both the accuracy, recall score, and time efficiency of the algorithm.

Table 7 is showing the best result of choosing different structures of the neural network, and Figures 47 - 58 is showing the overall tuning process.

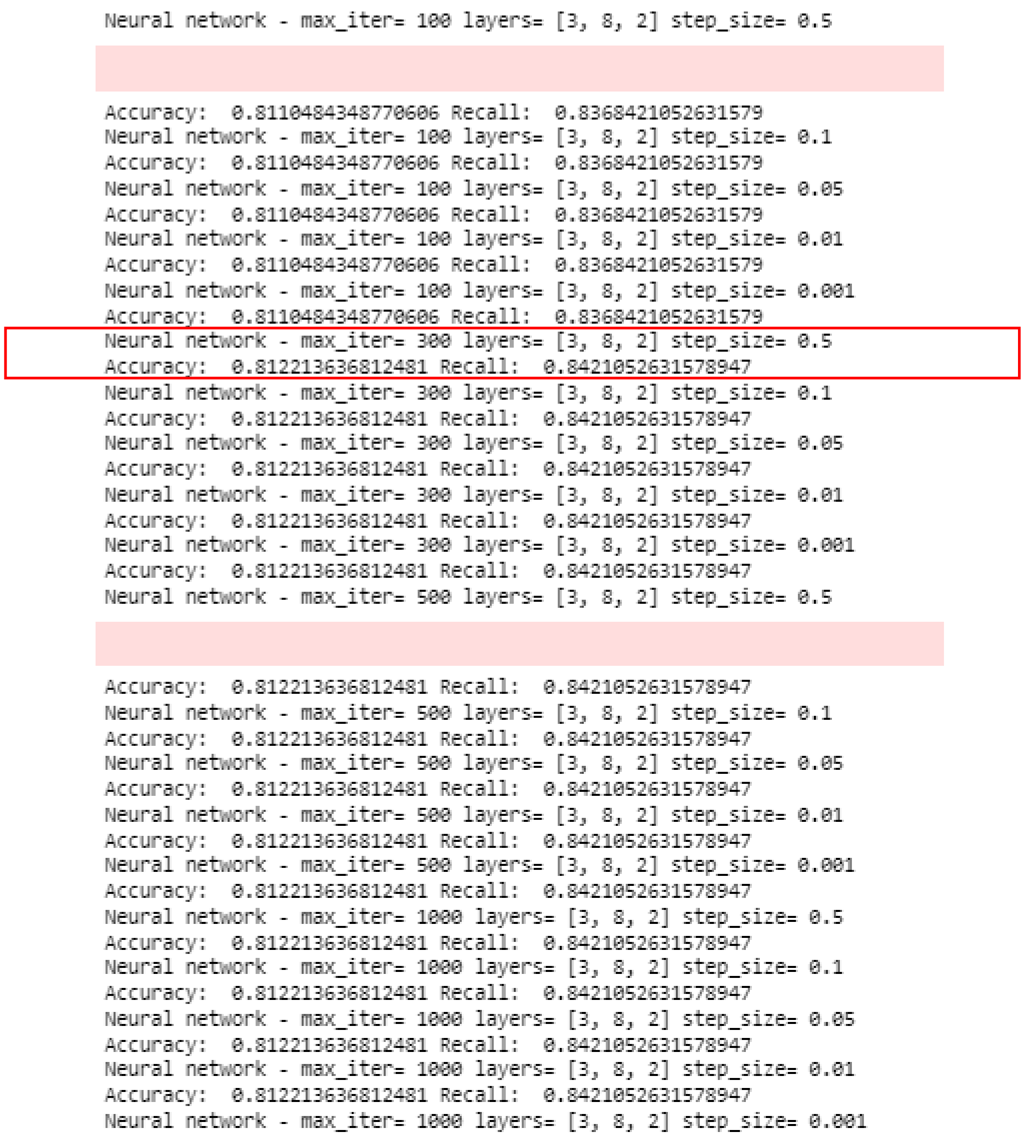
Max number of runs	**Step size**	**Structure of the neural network**	**Accuracy%**	**Recall%**
300	0.5	[3, 8, 2]	64.34	36.32
300	0.5	[3, 32, 2]	80.79	81.05
300	0.5	[3, 8, 32, 2]	81.5	83.68
100	0.5	[3, 16, 32, 8, 2]	81.57	82.63
100	0.5	[3, 6, 62, 2]	83.34	86.55

Table 7. Neural network model parameter pruning



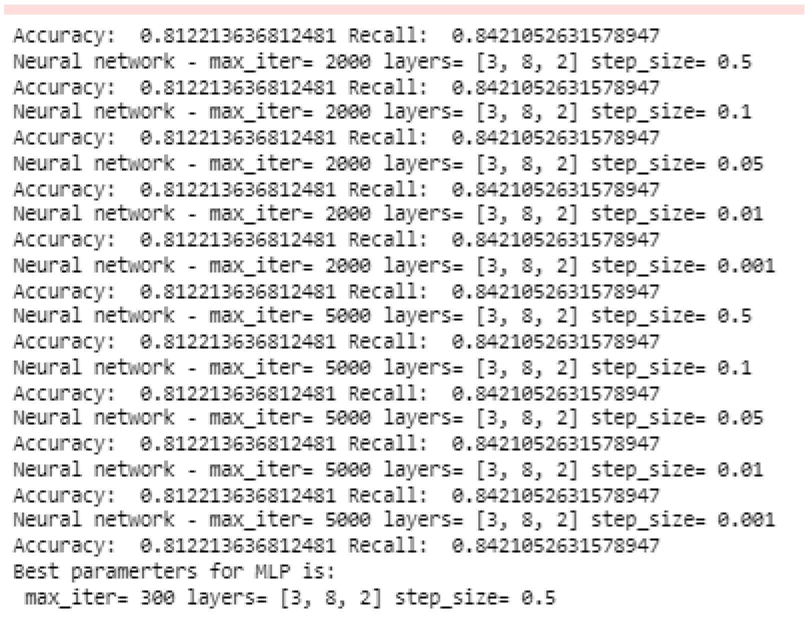
```
maxIter=[100, 300, 500, 1000, 2000, 5000]
stepSize=[0.5, 0.1, 0.05, 0.01, 0.001]
layers=[[3, 8, 2]]

best = 0
best_mi = 0
best_ss = 0
best_ly = 0
for mi in maxIter:
    for ss in stepSize:
        for ly in layers:
            print("Neural network -",'max_iter=',mi,'layers=',ly, 'step_size=',ss)
            start = time.time()
            mlp = MultilayerPerceptronClassifier(labelCol="stroke", featuresCol = "pca_features",
                                                maxIter=mi, layers=ly, stepSize=ss)
            mlp_model = mlp.fit(train_oversample_pca)
            mlp_predictions = mlp_model.transform(test_pca)
            end = time.time()

            binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
            accuracy = binary_eval.evaluate(mlp_predictions)

            tp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 1)].count()
            tn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 0)].count()
            fn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 1)].count()
            fp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 0)].count()

            recall = float((tp)/(tp + fn))
            print("Accuracy: ",accuracy,"Recall: ",recall)
            acc_rec = accuracy + recall
            if acc_rec > best:
                best = acc_rec
                best_mi = mi
                best_ss = ss
                best_ly = ly

print('Best paramerters for MLP is:\n', 'max_iter=',best_mi,'layers=',best_ly, 'step_size=',best_ss)
```

Figure 47. parameter pruning with layer [3, 8, 2]



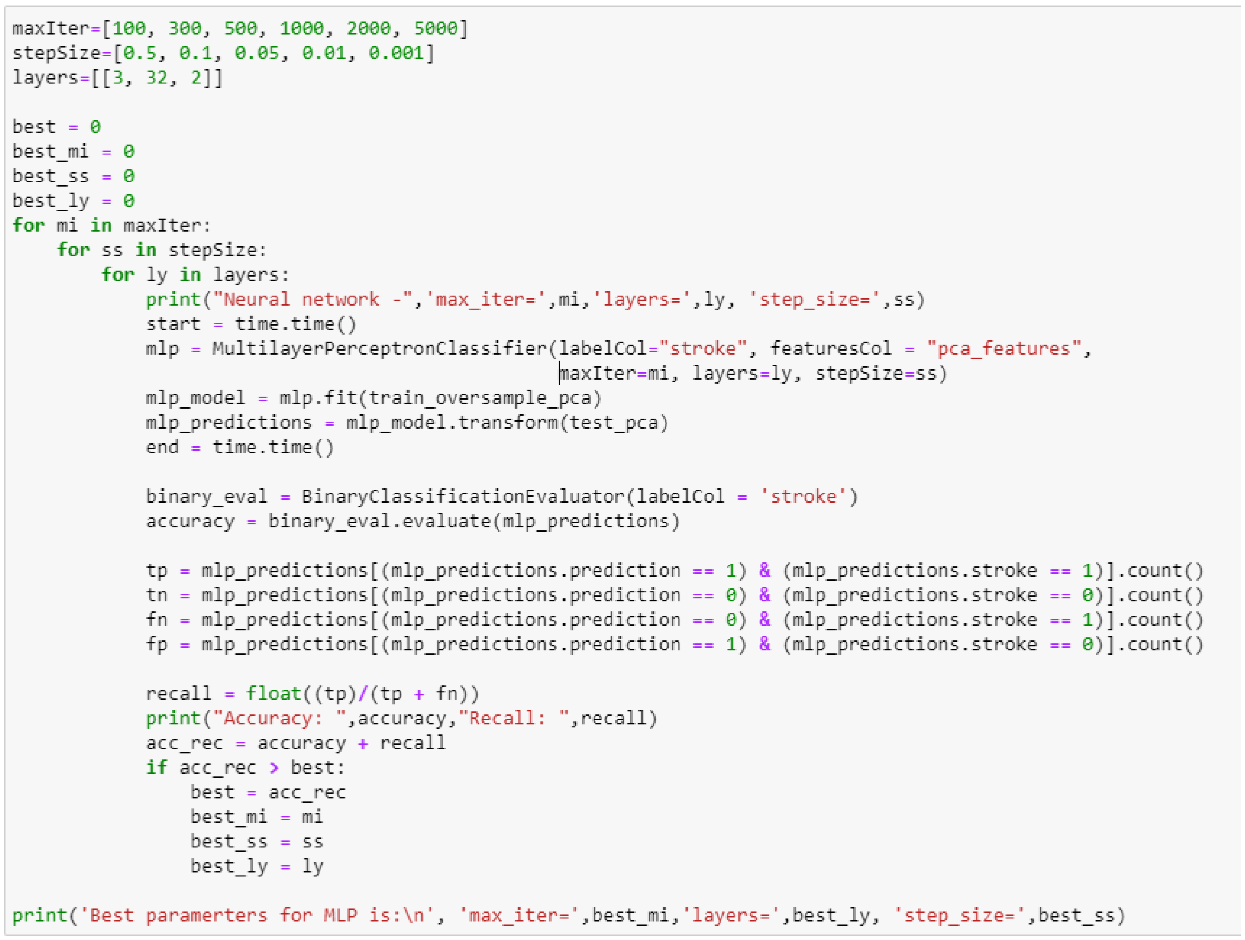
```
Neural network - max_iter= 100 layers= [3, 8, 2] step_size= 0.5

Accuracy:  0.8110484348770606 Recall:  0.8368421052631579
Neural network - max_iter= 100 layers= [3, 8, 2] step_size= 0.1
Accuracy:  0.8110484348770606 Recall:  0.8368421052631579
Neural network - max_iter= 100 layers= [3, 8, 2] step_size= 0.05
Accuracy:  0.8110484348770606 Recall:  0.8368421052631579
Neural network - max_iter= 100 layers= [3, 8, 2] step_size= 0.01
Accuracy:  0.8110484348770606 Recall:  0.8368421052631579
Neural network - max_iter= 100 layers= [3, 8, 2] step_size= 0.001
Accuracy:  0.8110484348770606 Recall:  0.8368421052631579
Neural network - max_iter= 300 layers= [3, 8, 2] step_size= 0.5
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 300 layers= [3, 8, 2] step_size= 0.1
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 300 layers= [3, 8, 2] step_size= 0.05
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 300 layers= [3, 8, 2] step_size= 0.01
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 300 layers= [3, 8, 2] step_size= 0.001
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 500 layers= [3, 8, 2] step_size= 0.5

Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 500 layers= [3, 8, 2] step_size= 0.1
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 500 layers= [3, 8, 2] step_size= 0.05
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 500 layers= [3, 8, 2] step_size= 0.01
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 500 layers= [3, 8, 2] step_size= 0.001
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 1000 layers= [3, 8, 2] step_size= 0.5
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 1000 layers= [3, 8, 2] step_size= 0.1
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 1000 layers= [3, 8, 2] step_size= 0.05
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 1000 layers= [3, 8, 2] step_size= 0.01
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 1000 layers= [3, 8, 2] step_size= 0.001
```

Figure 48. parameter pruning result 1 with layer [3, 8, 2]



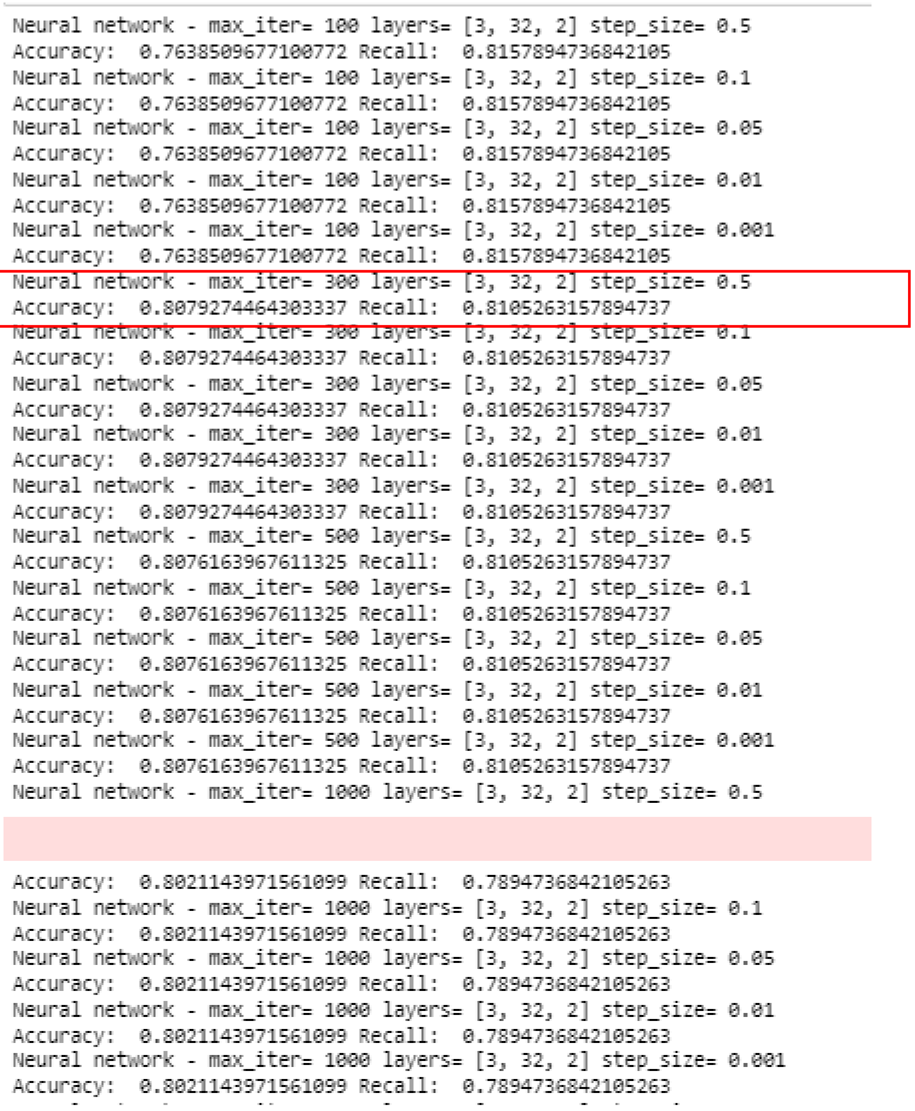
```
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 2000 layers= [3, 8, 2] step_size= 0.5
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 2000 layers= [3, 8, 2] step_size= 0.1
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 2000 layers= [3, 8, 2] step_size= 0.05
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 2000 layers= [3, 8, 2] step_size= 0.01
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 2000 layers= [3, 8, 2] step_size= 0.001
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 5000 layers= [3, 8, 2] step_size= 0.5
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 5000 layers= [3, 8, 2] step_size= 0.1
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 5000 layers= [3, 8, 2] step_size= 0.05
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 5000 layers= [3, 8, 2] step_size= 0.01
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Neural network - max_iter= 5000 layers= [3, 8, 2] step_size= 0.001
Accuracy:  0.812213636812481 Recall:  0.8421052631578947
Best paramerters for MLP is:
 max_iter= 300 layers= [3, 8, 2] step_size= 0.5
```

Figure 49. parameter pruning result 2 with layer [3, 8, 2]



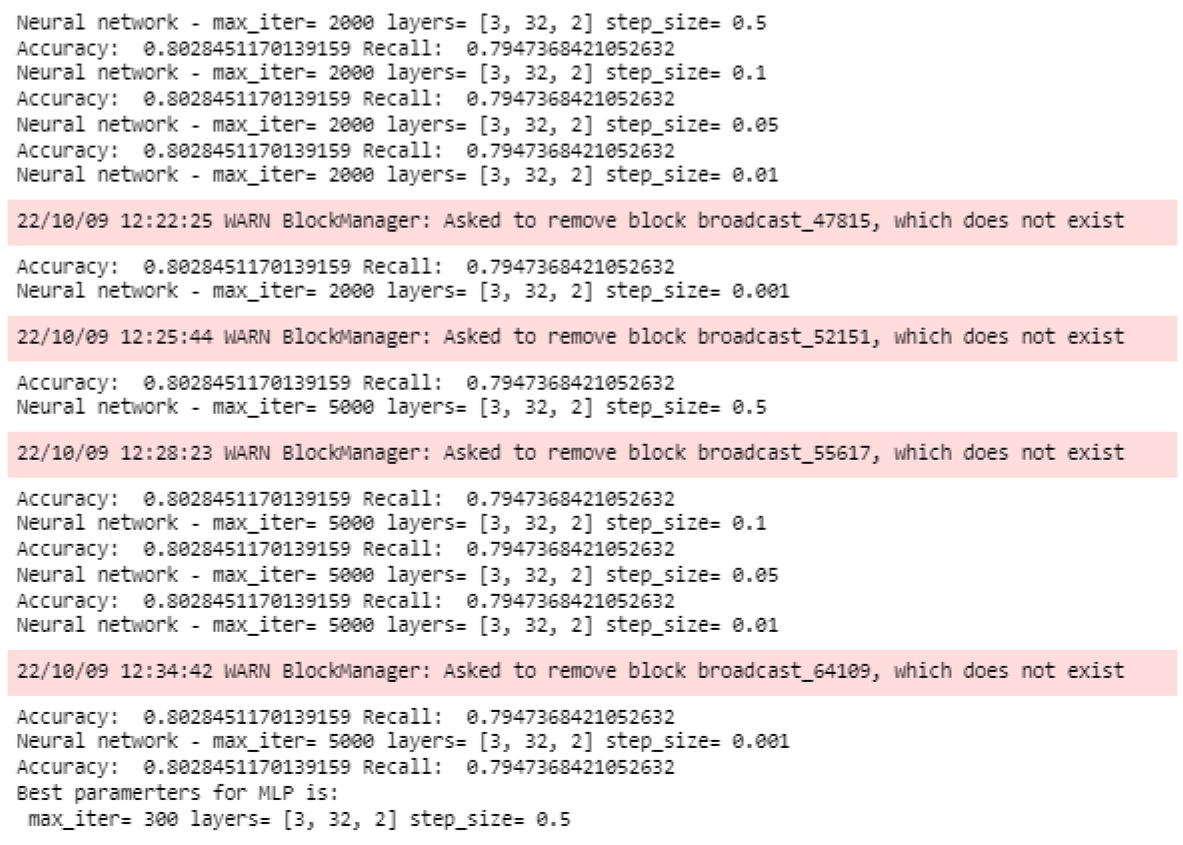
```
maxIter=[100, 300, 500, 1000, 2000, 5000]
stepSize=[0.5, 0.1, 0.05, 0.01, 0.001]
layers=[[3, 32, 2]]

best = 0
best_mi = 0
best_ss = 0
best_ly = 0
for mi in maxIter:
    for ss in stepSize:
        for ly in layers:
            print("Neural network -",'max_iter=',mi,'layers=',ly, 'step_size=',ss)
            start = time.time()
            mlp = MultilayerPerceptronClassifier(labelCol="stroke", featuresCol = "pca_features",
                                                 maxIter=mi, layers=ly, stepSize=ss)
            mlp_model = mlp.fit(train_oversample_pca)
            mlp_predictions = mlp_model.transform(test_pca)
            end = time.time()

            binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
            accuracy = binary_eval.evaluate(mlp_predictions)

            tp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 1)].count()
            tn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 0)].count()
            fn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 1)].count()
            fp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 0)].count()

            recall = float((tp)/(tp + fn))
            print("Accuracy: ",accuracy,"Recall: ",recall)
            acc_rec = accuracy + recall
            if acc_rec > best:
                best = acc_rec
                best_mi = mi
                best_ss = ss
                best_ly = ly

print('Best paramerters for MLP is:\n', 'max_iter=',best_mi,'layers=',best_ly, 'step_size=',best_ss)
```

Figure 50. parameter pruning process with layer [3, 32, 2]



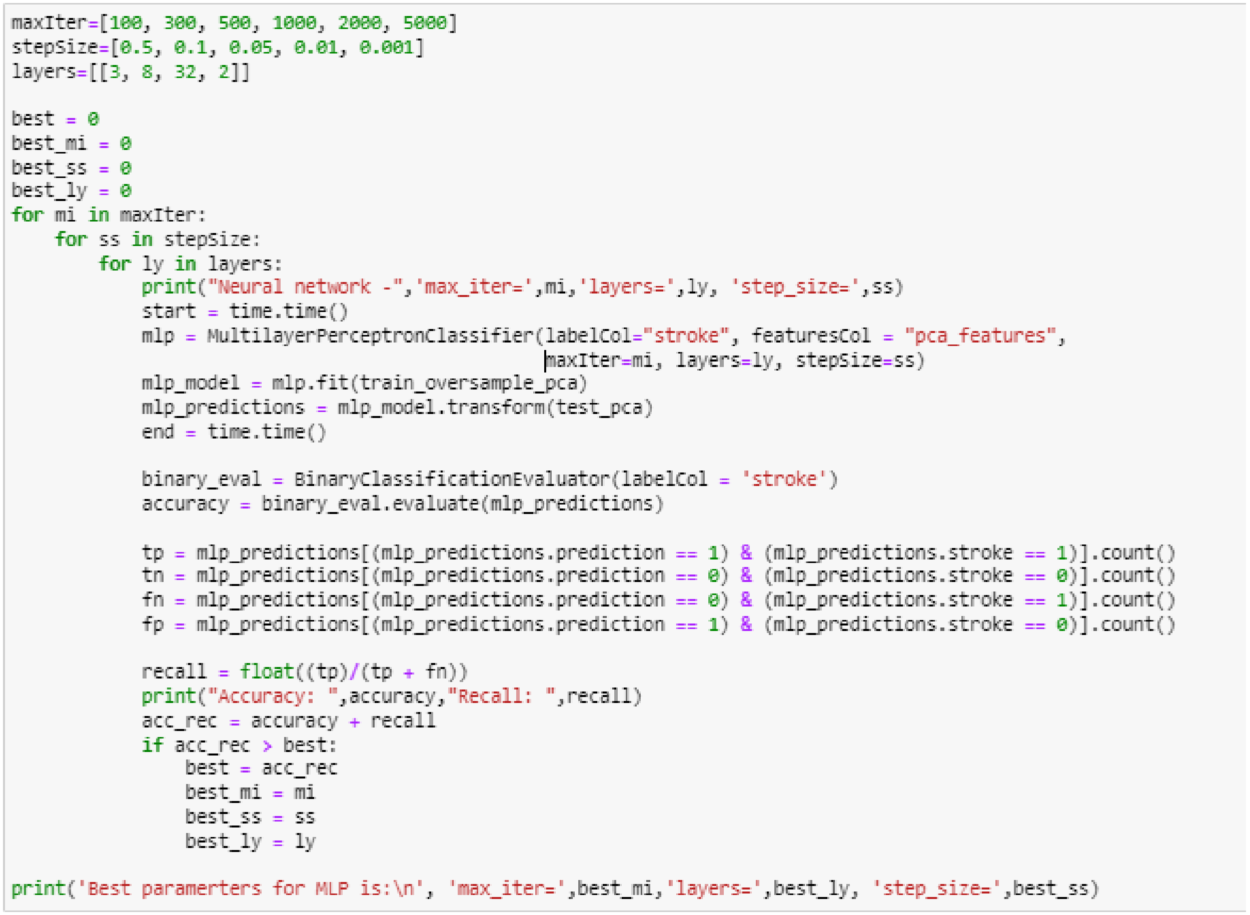
```
Neural network - max_iter= 100 layers= [3, 32, 2] step_size= 0.5
Accuracy:  0.7638509677100772 Recall:  0.8157894736842105
Neural network - max_iter= 100 layers= [3, 32, 2] step_size= 0.1
Accuracy:  0.7638509677100772 Recall:  0.8157894736842105
Neural network - max_iter= 100 layers= [3, 32, 2] step_size= 0.05
Accuracy:  0.7638509677100772 Recall:  0.8157894736842105
Neural network - max_iter= 100 layers= [3, 32, 2] step_size= 0.01
Accuracy:  0.7638509677100772 Recall:  0.8157894736842105
Neural network - max_iter= 100 layers= [3, 32, 2] step_size= 0.001
Accuracy:  0.7638509677100772 Recall:  0.8157894736842105
Neural network - max_iter= 300 layers= [3, 32, 2] step_size= 0.5
Accuracy:  0.8079274464303337 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 32, 2] step_size= 0.1
Accuracy:  0.8079274464303337 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 32, 2] step_size= 0.05
Accuracy:  0.8079274464303337 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 32, 2] step_size= 0.01
Accuracy:  0.8079274464303337 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 32, 2] step_size= 0.001
Accuracy:  0.8079274464303337 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 32, 2] step_size= 0.5
Accuracy:  0.8076163967611325 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 32, 2] step_size= 0.1
Accuracy:  0.8076163967611325 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 32, 2] step_size= 0.05
Accuracy:  0.8076163967611325 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 32, 2] step_size= 0.01
Accuracy:  0.8076163967611325 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 32, 2] step_size= 0.001
Accuracy:  0.8076163967611325 Recall:  0.8105263157894737
Neural network - max_iter= 1000 layers= [3, 32, 2] step_size= 0.5

Accuracy:  0.8021143971561099 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 32, 2] step_size= 0.1
Accuracy:  0.8021143971561099 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 32, 2] step_size= 0.05
Accuracy:  0.8021143971561099 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 32, 2] step_size= 0.01
Accuracy:  0.8021143971561099 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 32, 2] step_size= 0.001
Accuracy:  0.8021143971561099 Recall:  0.7894736842105263
```

Figure 51. parameter pruning result 1 with layer [3, 32, 2]



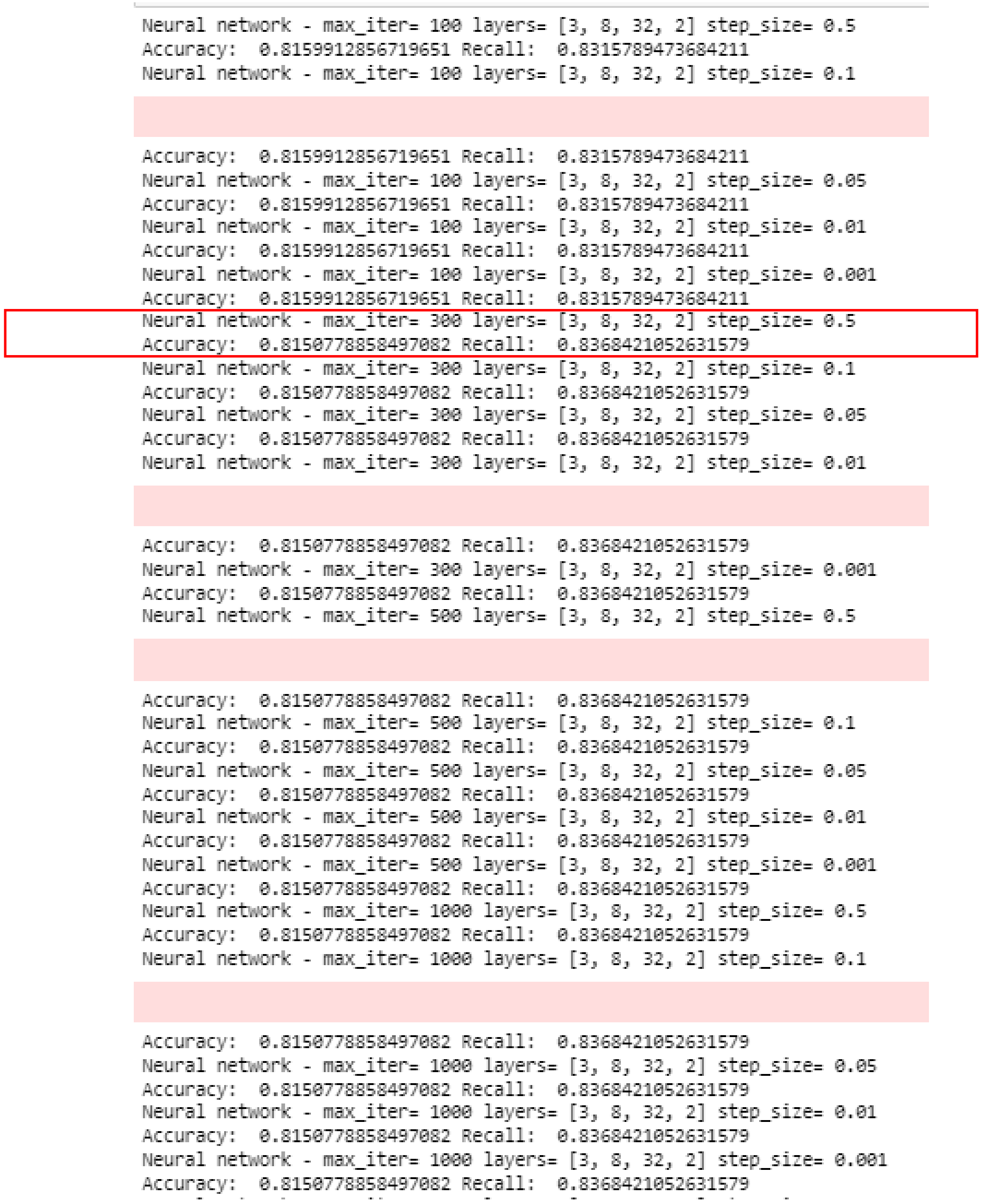
```
Neural network - max_iter= 2000 layers= [3, 32, 2] step_size= 0.5
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 2000 layers= [3, 32, 2] step_size= 0.1
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 2000 layers= [3, 32, 2] step_size= 0.05
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 2000 layers= [3, 32, 2] step_size= 0.01
22/10/09 12:22:25 WARN BlockManager: Asked to remove block broadcast_47815, which does not exist
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 2000 layers= [3, 32, 2] step_size= 0.001
22/10/09 12:25:44 WARN BlockManager: Asked to remove block broadcast_52151, which does not exist
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 5000 layers= [3, 32, 2] step_size= 0.5
22/10/09 12:28:23 WARN BlockManager: Asked to remove block broadcast_55617, which does not exist
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 5000 layers= [3, 32, 2] step_size= 0.1
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 5000 layers= [3, 32, 2] step_size= 0.05
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 5000 layers= [3, 32, 2] step_size= 0.01
22/10/09 12:34:42 WARN BlockManager: Asked to remove block broadcast_64109, which does not exist
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Neural network - max_iter= 5000 layers= [3, 32, 2] step_size= 0.001
Accuracy:  0.8028451170139159 Recall:  0.7947368421052632
Best paramerters for MLP is:
 max_iter= 300 layers= [3, 32, 2] step_size= 0.5
```

Figure 52. parameter pruning result 2 with layer [3, 32, 2]



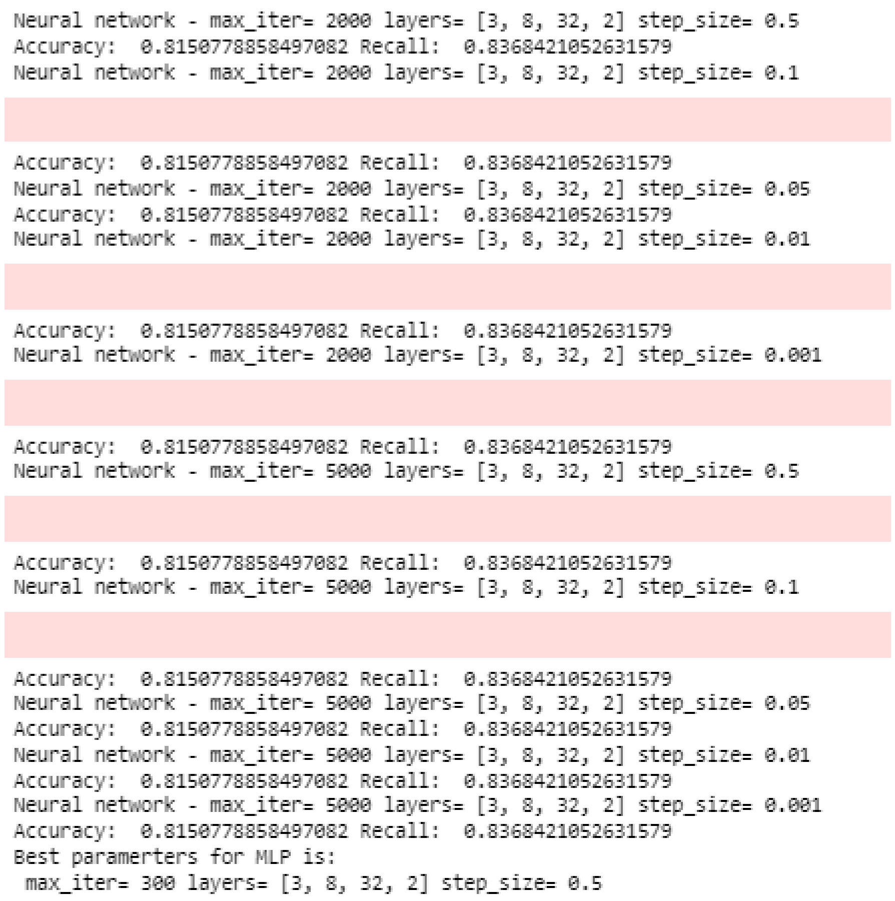
```
maxIter=[100, 300, 500, 1000, 2000, 5000]
stepSize=[0.5, 0.1, 0.05, 0.01, 0.001]
layers=[[3, 8, 32, 2]]

best = 0
best_mi = 0
best_ss = 0
best_ly = 0
for mi in maxIter:
    for ss in stepSize:
        for ly in layers:
            print("Neural network -",'max_iter=',mi,'layers=',ly, 'step_size=',ss)
            start = time.time()
            mlp = MultilayerPerceptronClassifier(labelCol="stroke", featuresCol = "pca_features",
                                                 maxIter=mi, layers=ly, stepSize=ss)
            mlp_model = mlp.fit(train_oversample_pca)
            mlp_predictions = mlp_model.transform(test_pca)
            end = time.time()

            binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
            accuracy = binary_eval.evaluate(mlp_predictions)

            tp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 1)].count()
            tn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 0)].count()
            fn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 1)].count()
            fp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 0)].count()

            recall = float((tp)/(tp + fn))
            print("Accuracy: ",accuracy,"Recall: ",recall)
            acc_rec = accuracy + recall
            if acc_rec > best:
                best = acc_rec
                best_mi = mi
                best_ss = ss
                best_ly = ly

print('Best paramerters for MLP is:\n', 'max_iter=',best_mi,'layers=',best_ly, 'step_size=',best_ss)
```

Figure 53. parameter pruning process with layer [3, 8, 32, 2]



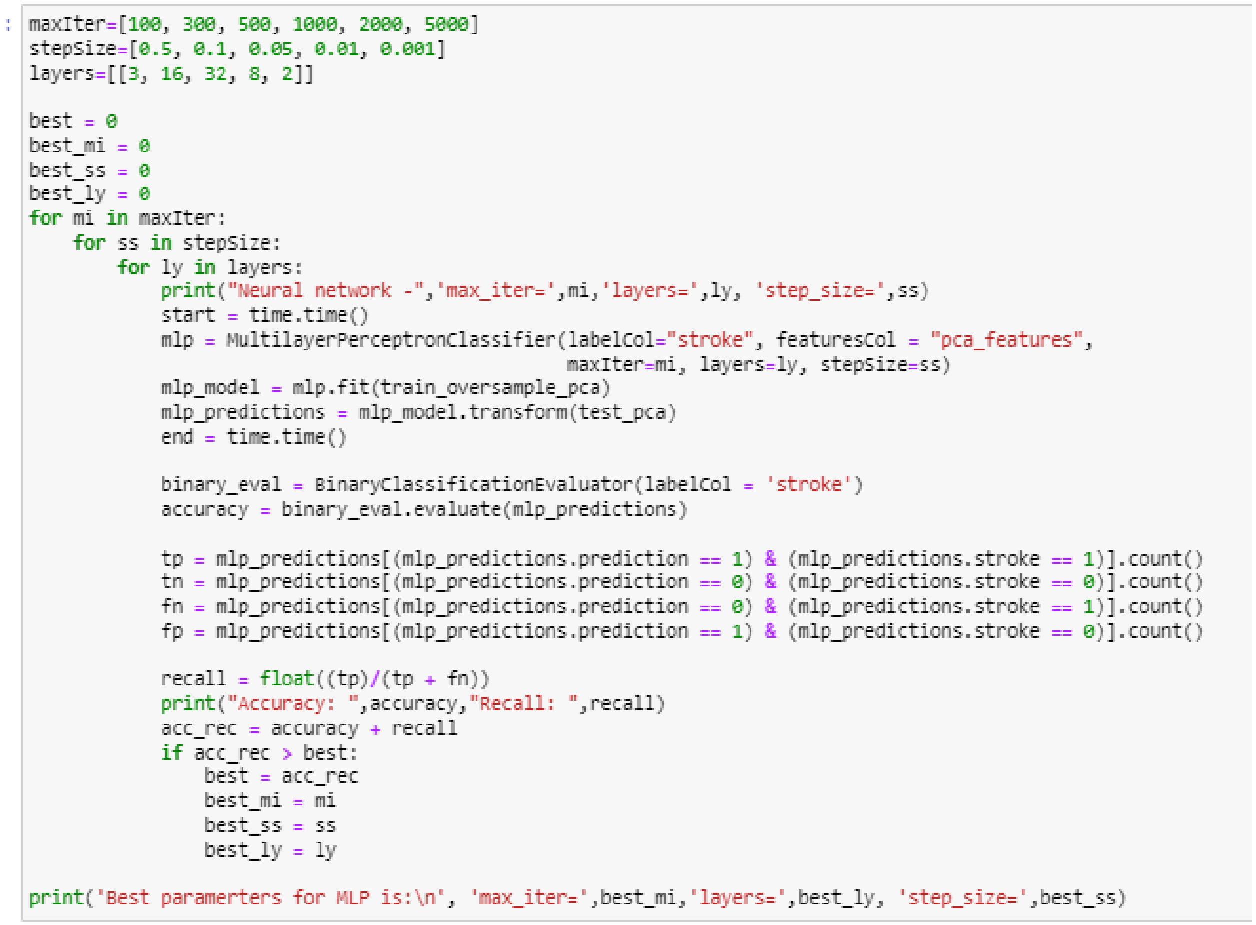
```
Neural network - max_iter= 100 layers= [3, 8, 32, 2] step_size= 0.5
Accuracy:  0.8159912856719651 Recall:  0.8315789473684211
Neural network - max_iter= 100 layers= [3, 8, 32, 2] step_size= 0.1

Accuracy:  0.8159912856719651 Recall:  0.8315789473684211
Neural network - max_iter= 100 layers= [3, 8, 32, 2] step_size= 0.05
Accuracy:  0.8159912856719651 Recall:  0.8315789473684211
Neural network - max_iter= 100 layers= [3, 8, 32, 2] step_size= 0.01
Accuracy:  0.8159912856719651 Recall:  0.8315789473684211
Neural network - max_iter= 100 layers= [3, 8, 32, 2] step_size= 0.001
Accuracy:  0.8159912856719651 Recall:  0.8315789473684211
Neural network - max_iter= 300 layers= [3, 8, 32, 2] step_size= 0.5
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 300 layers= [3, 8, 32, 2] step_size= 0.1
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 300 layers= [3, 8, 32, 2] step_size= 0.05
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 300 layers= [3, 8, 32, 2] step_size= 0.01

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 300 layers= [3, 8, 32, 2] step_size= 0.001
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 500 layers= [3, 8, 32, 2] step_size= 0.5

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 500 layers= [3, 8, 32, 2] step_size= 0.1
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 500 layers= [3, 8, 32, 2] step_size= 0.05
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 500 layers= [3, 8, 32, 2] step_size= 0.01
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 500 layers= [3, 8, 32, 2] step_size= 0.001
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 1000 layers= [3, 8, 32, 2] step_size= 0.5
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 1000 layers= [3, 8, 32, 2] step_size= 0.1

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 1000 layers= [3, 8, 32, 2] step_size= 0.05
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 1000 layers= [3, 8, 32, 2] step_size= 0.01
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 1000 layers= [3, 8, 32, 2] step_size= 0.001
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
```

Figure 54. parameter pruning result 1 with layer [3, 8, 32, 2]



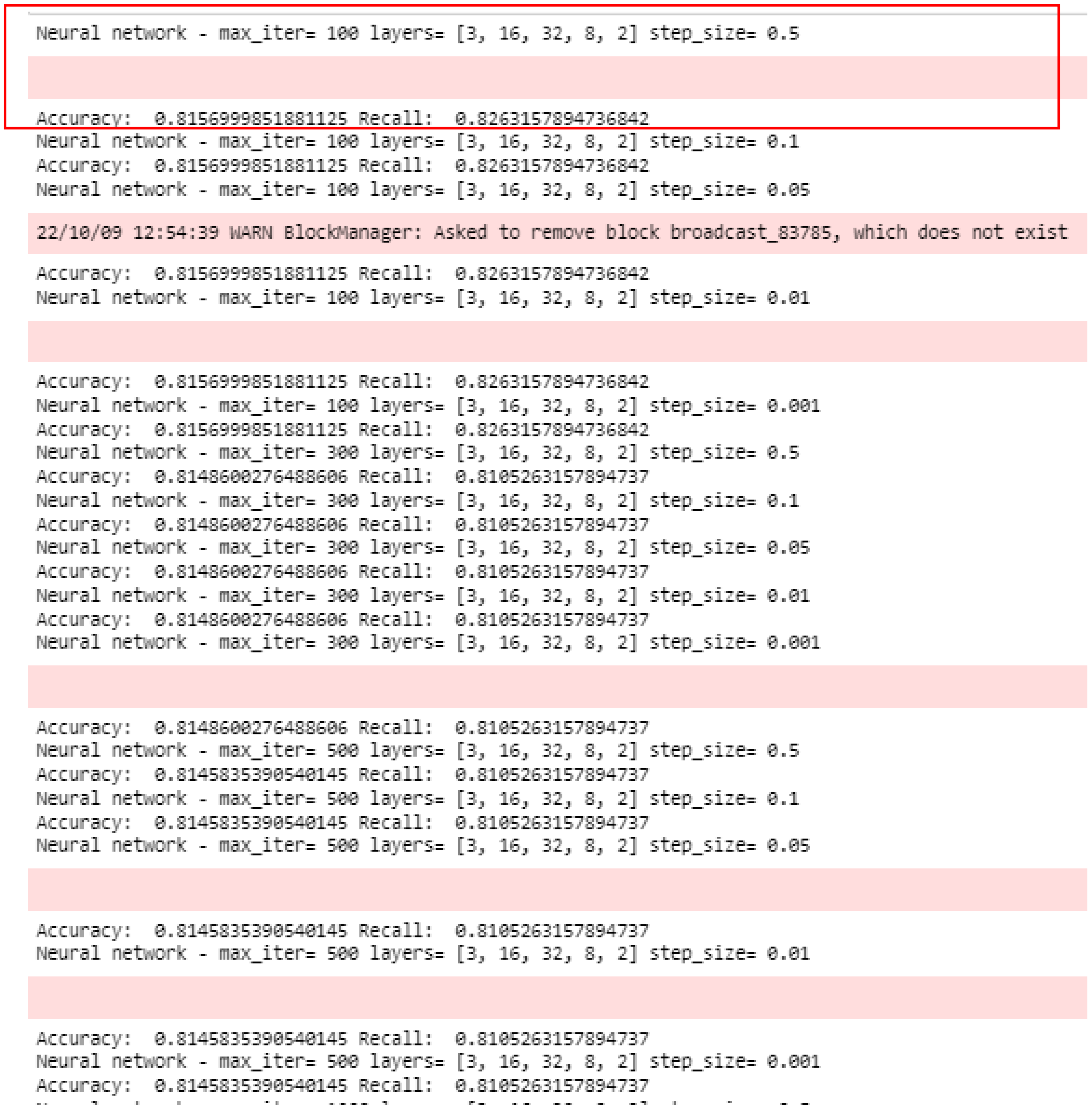
```
Neural network - max_iter= 2000 layers= [3, 8, 32, 2] step_size= 0.5
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 2000 layers= [3, 8, 32, 2] step_size= 0.1

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 2000 layers= [3, 8, 32, 2] step_size= 0.05
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 2000 layers= [3, 8, 32, 2] step_size= 0.01

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 2000 layers= [3, 8, 32, 2] step_size= 0.001

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 5000 layers= [3, 8, 32, 2] step_size= 0.5

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 5000 layers= [3, 8, 32, 2] step_size= 0.1

Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 5000 layers= [3, 8, 32, 2] step_size= 0.05
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 5000 layers= [3, 8, 32, 2] step_size= 0.01
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Neural network - max_iter= 5000 layers= [3, 8, 32, 2] step_size= 0.001
Accuracy:  0.8150778858497082 Recall:  0.8368421052631579
Best paramerters for MLP is:
 max_iter= 300 layers= [3, 8, 32, 2] step_size= 0.5
```

Figure 55. parameter pruning result 2 with layer [3, 8, 32, 2]



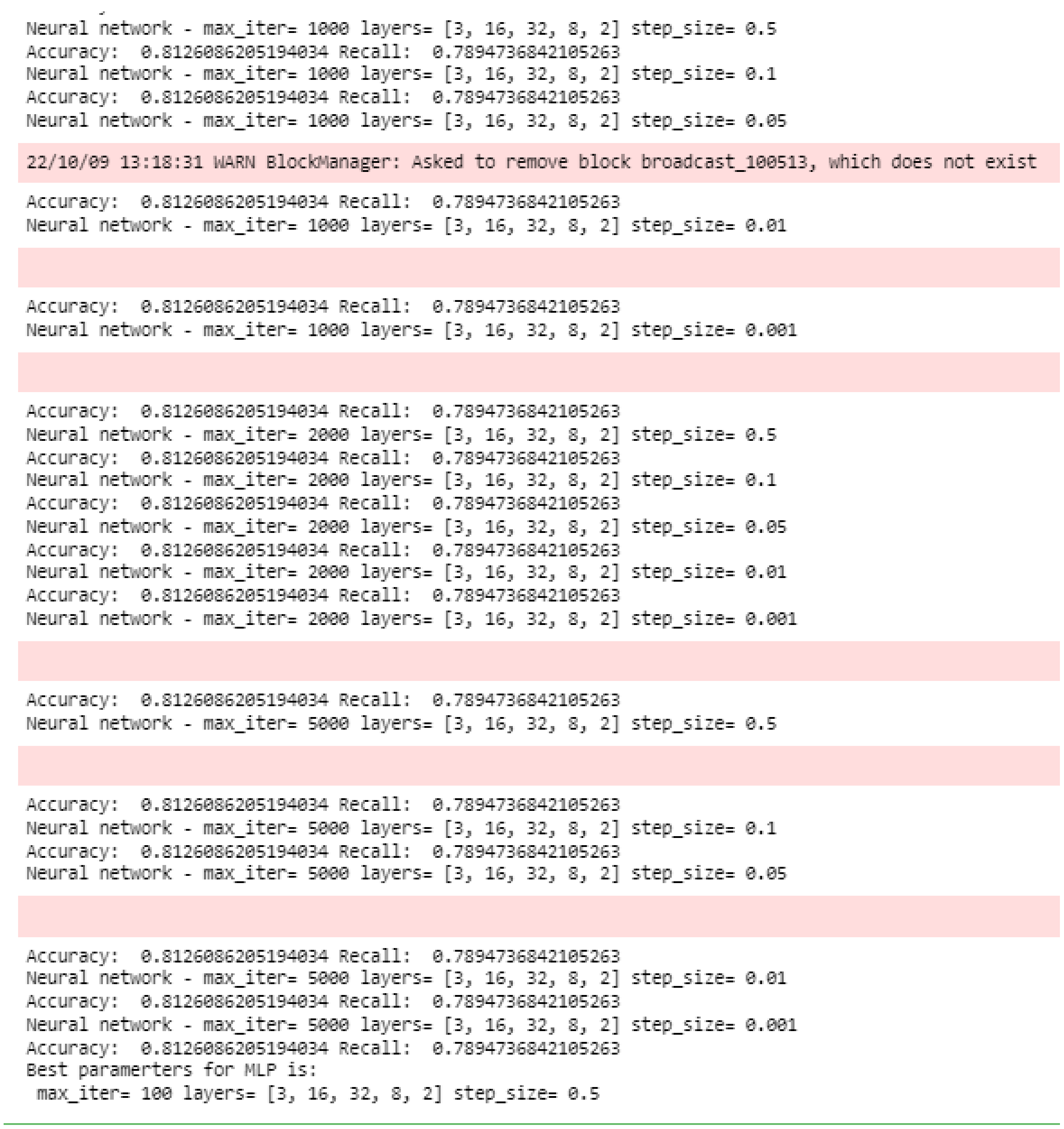
```
maxIter=[100, 300, 500, 1000, 2000, 5000]
stepSize=[0.5, 0.1, 0.05, 0.01, 0.001]
layers=[[3, 16, 32, 8, 2]]

best = 0
best_mi = 0
best_ss = 0
best_ly = 0
for mi in maxIter:
    for ss in stepSize:
        for ly in layers:
            print("Neural network -",'max_iter=',mi,'layers=',ly, 'step_size=',ss)
            start = time.time()
            mlp = MultilayerPerceptronClassifier(labelCol="stroke", featuresCol = "pca_features",
                                                 maxIter=mi, layers=ly, stepSize=ss)
            mlp_model = mlp.fit(train_oversample_pca)
            mlp_predictions = mlp_model.transform(test_pca)
            end = time.time()

            binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
            accuracy = binary_eval.evaluate(mlp_predictions)

            tp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 1)].count()
            tn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 0)].count()
            fn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 1)].count()
            fp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 0)].count()

            recall = float((tp)/(tp + fn))
            print("Accuracy: ",accuracy,"Recall: ",recall)
            acc_rec = accuracy + recall
            if acc_rec > best:
                best = acc_rec
                best_mi = mi
                best_ss = ss
                best_ly = ly

print('Best paramerters for MLP is:\n', 'max_iter=',best_mi,'layers=',best_ly, 'step_size=',best_ss)
```

Figure 56. parameter pruning process with layer [3, 16, 32, 8, 2]



```
Neural network - max_iter= 100 layers= [3, 16, 32, 8, 2] step_size= 0.5

Accuracy:  0.8156999851881125 Recall:  0.8263157894736842
Neural network - max_iter= 100 layers= [3, 16, 32, 8, 2] step_size= 0.1
Accuracy:  0.8156999851881125 Recall:  0.8263157894736842
Neural network - max_iter= 100 layers= [3, 16, 32, 8, 2] step_size= 0.05
22/10/09 12:54:39 WARN BlockManager: Asked to remove block broadcast_83785, which does not exist
Accuracy:  0.8156999851881125 Recall:  0.8263157894736842
Neural network - max_iter= 100 layers= [3, 16, 32, 8, 2] step_size= 0.01

Accuracy:  0.8156999851881125 Recall:  0.8263157894736842
Neural network - max_iter= 100 layers= [3, 16, 32, 8, 2] step_size= 0.001
Accuracy:  0.8156999851881125 Recall:  0.8263157894736842
Neural network - max_iter= 300 layers= [3, 16, 32, 8, 2] step_size= 0.5
Accuracy:  0.8148600276488606 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 16, 32, 8, 2] step_size= 0.1
Accuracy:  0.8148600276488606 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 16, 32, 8, 2] step_size= 0.05
Accuracy:  0.8148600276488606 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 16, 32, 8, 2] step_size= 0.01
Accuracy:  0.8148600276488606 Recall:  0.8105263157894737
Neural network - max_iter= 300 layers= [3, 16, 32, 8, 2] step_size= 0.001

Accuracy:  0.8148600276488606 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 16, 32, 8, 2] step_size= 0.5
Accuracy:  0.8145835390540145 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 16, 32, 8, 2] step_size= 0.1
Accuracy:  0.8145835390540145 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 16, 32, 8, 2] step_size= 0.05

Accuracy:  0.8145835390540145 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 16, 32, 8, 2] step_size= 0.01

Accuracy:  0.8145835390540145 Recall:  0.8105263157894737
Neural network - max_iter= 500 layers= [3, 16, 32, 8, 2] step_size= 0.001
Accuracy:  0.8145835390540145 Recall:  0.8105263157894737
```

Figure 57. parameter pruning result 1 with layer [3, 16, 32, 8, 2]



```
Neural network - max_iter= 1000 layers= [3, 16, 32, 8, 2] step_size= 0.5
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 16, 32, 8, 2] step_size= 0.1
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 16, 32, 8, 2] step_size= 0.05
22/10/09 13:18:31 WARN BlockManager: Asked to remove block broadcast_100513, which does not exist
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 16, 32, 8, 2] step_size= 0.01

Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 1000 layers= [3, 16, 32, 8, 2] step_size= 0.001

Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 2000 layers= [3, 16, 32, 8, 2] step_size= 0.5
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 2000 layers= [3, 16, 32, 8, 2] step_size= 0.1
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 2000 layers= [3, 16, 32, 8, 2] step_size= 0.05
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 2000 layers= [3, 16, 32, 8, 2] step_size= 0.01
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 2000 layers= [3, 16, 32, 8, 2] step_size= 0.001

Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 5000 layers= [3, 16, 32, 8, 2] step_size= 0.5

Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 5000 layers= [3, 16, 32, 8, 2] step_size= 0.1
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 5000 layers= [3, 16, 32, 8, 2] step_size= 0.05

Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 5000 layers= [3, 16, 32, 8, 2] step_size= 0.01
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Neural network - max_iter= 5000 layers= [3, 16, 32, 8, 2] step_size= 0.001
Accuracy:  0.8126086205194034 Recall:  0.7894736842105263
Best paramerters for MLP is:
 max_iter= 100 layers= [3, 16, 32, 8, 2] step_size= 0.5
```

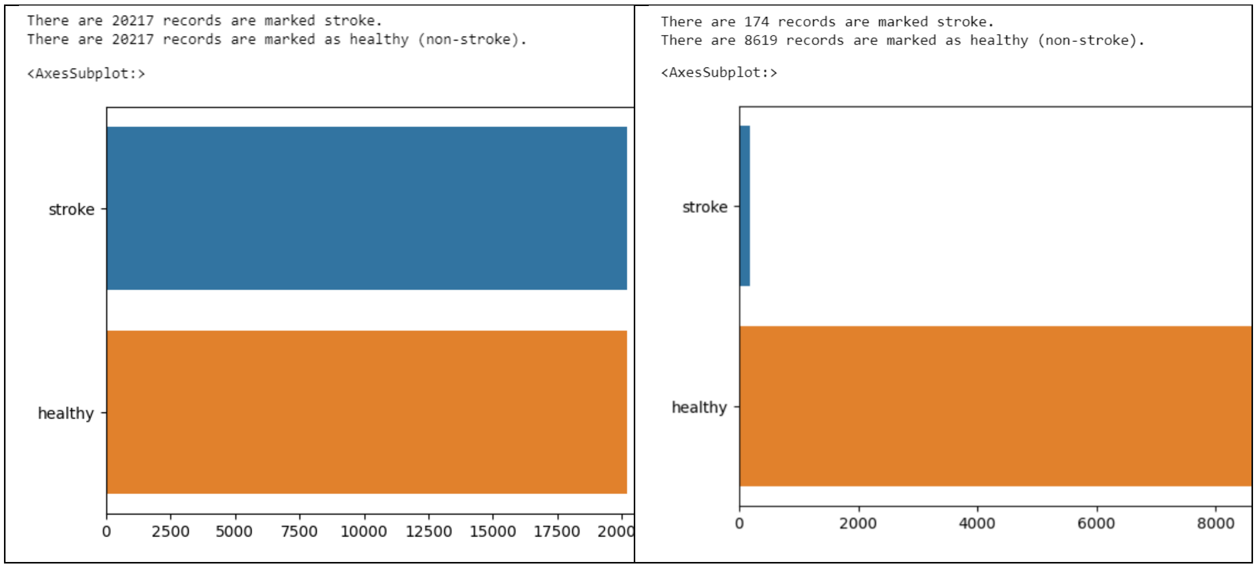
Figure 58. parameter pruning result 2 with layer [3, 16, 32, 8, 2]

7. Data Mining

7.1 Logical test designs

Before the data mining was conducted, we first spited the dataset into a training and testing dataset as shown in section 3.4. A method for assessing a machine learning algorithm's performance is the train-test split. It may be applied to issues involving classification or regression as well as any supervised learning technique. The process entails splitting the dataset into two subgroups. The training dataset is the initial subset, which is used to fit the

model. The model is not trained using the second subset; rather, it is given the input element of the dataset, and its predictions are then produced and contrasted with the expected values. The test dataset is the second dataset in question. For fitting the machine learning model, use the train dataset. Test dataset used to assess how well a machine learning model fits the data. The dataset is split into two parts, the training and test datasets. This operation simulates the actual environment faced by the model. Furthermore, by using the training/testing set, it is possible to assess the sensitivity of the model in the face of unseen data. The content inside the training dataset is the knowledge possessed before being labelled, while the content inside the test set is all the inputs, then the model needs to predict the answer based on the given information. Finally, the performance of this model can be evaluated based on various methods.
The split data ratio was set at 7:3, with the training data accounting for 70% of the total records and 30% reserved for the test data set. the 7:3 ratio allows the model to be adequately trained as it contains multiple cases where patterns can be found, and the 30% reset is large enough to cover sufficient instances (cases) for potential input and evaluation. Importantly, the data structure of test datasets should remain the same with raw dataset, especially for target attribute, so we did rebalance method on training dataset only. The target attribute distribution of training and testing dataset as shown in Figure 51.



```
There are 20217 records are marked stroke.
There are 20217 records are marked as healthy (non-stroke).

<AxesSubplot:>
```

```
There are 174 records are marked stroke.
There are 8619 records are marked as healthy (non-stroke).

<AxesSubplot:>
```

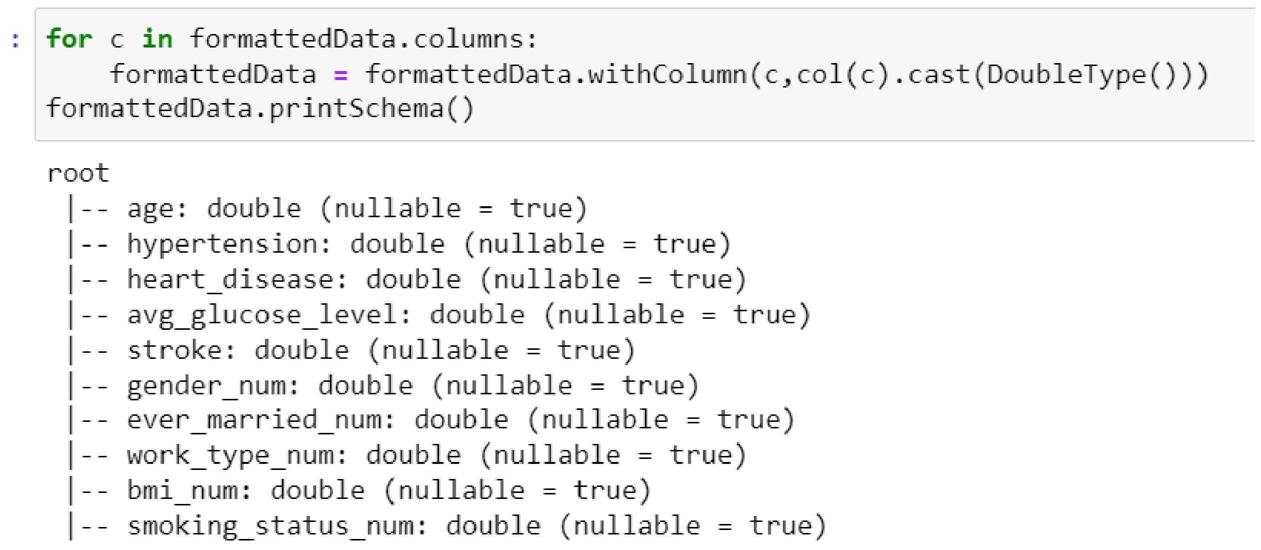
Figure 59. The data structure of stroke for the train & test set

As we can see, there are 20,217 records are marked as strokes, and 20,217 records marked as non-stroke in the training dataset.
There are 174 records are marked as stroke, and 8,619 records marked as non-stroke in the testing dataset.

7.2 Conduct Data mining

We run the partition that we have split in section 3.4, which split the train set and test set with a 7:3 ratio randomly. Then, we do data mining with a selected Neural Network algorithm.

The data mining model at this stage is being prepared to produce results. The input parameters will be gender, age, hypertension, heart disease, ever married, average blood glucose level, new BMI, new smoking status, and new job type, and the target is set to be stroke as shown in Figure 60. The total amount of data used was 49,227 instances, which were split into training/testing datasets. The split ratio was set to 7:3. Training dataset after using the rebalance method has 40,434 records as shown in Figure 59, and the testing dataset has 8,793 records as shown in Figure 59.



```
for c in formattedData.columns:
    formattedData = formattedData.withColumn(c,col(c).cast(DoubleType()))
formattedData.printSchema()
```

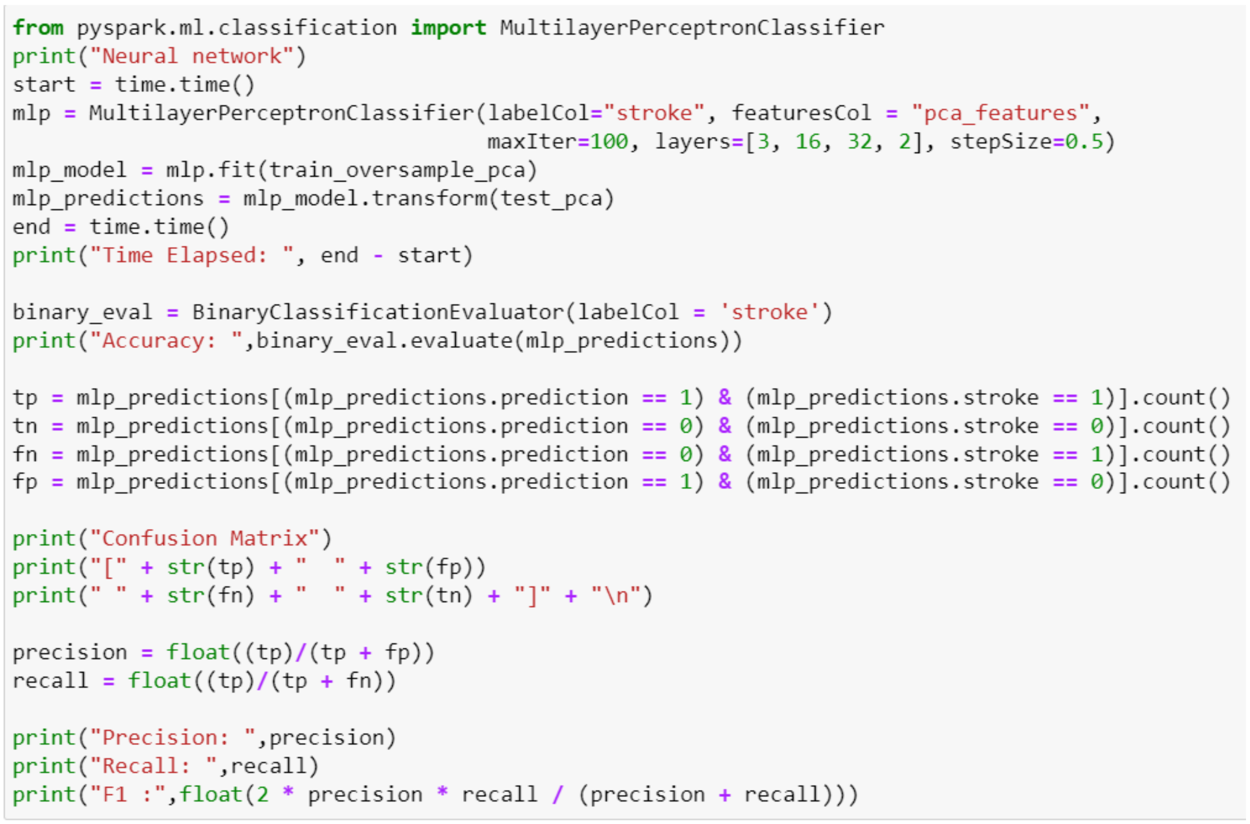
```
root
 |-- age: double (nullable = true)
 |-- hypertension: double (nullable = true)
 |-- heart_disease: double (nullable = true)
 |-- avg_glucose_level: double (nullable = true)
 |-- stroke: double (nullable = true)
 |-- gender_num: double (nullable = true)
 |-- ever_married_num: double (nullable = true)
 |-- work_type_num: double (nullable = true)
 |-- bmi_num: double (nullable = true)
 |-- smoking_status_num: double (nullable = true)
```

Figure 60. input parameters and target of the model

Figure 61 shows the entire data flow as well as the structure of the model. Also, the model with the predefined parameters and settings.

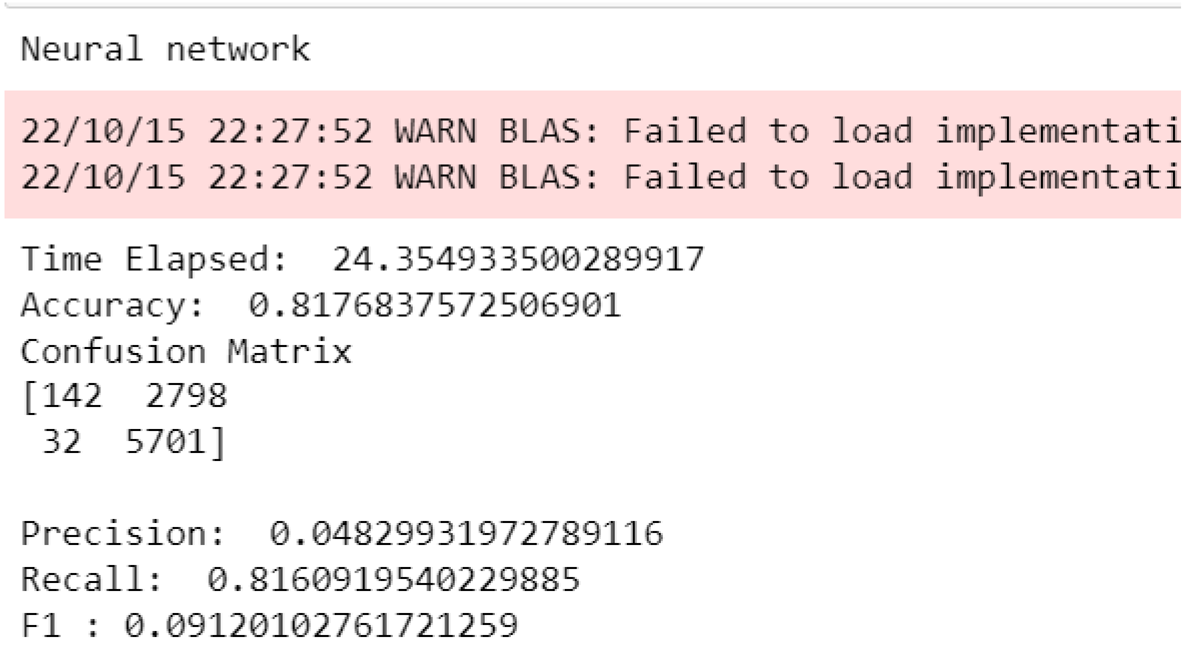
Figure 62 shows after the model ran, the results were successfully generated, which shows the results of the execution of the model. The figure includes the accuracy of the model, recall score, confusion matrix, and time measurements.

Figure 63 shows the importance of each attribute in the model. Age is the most important predictor, and heart disease is the second important predictor, and the new average glucose level is the third important predictor.



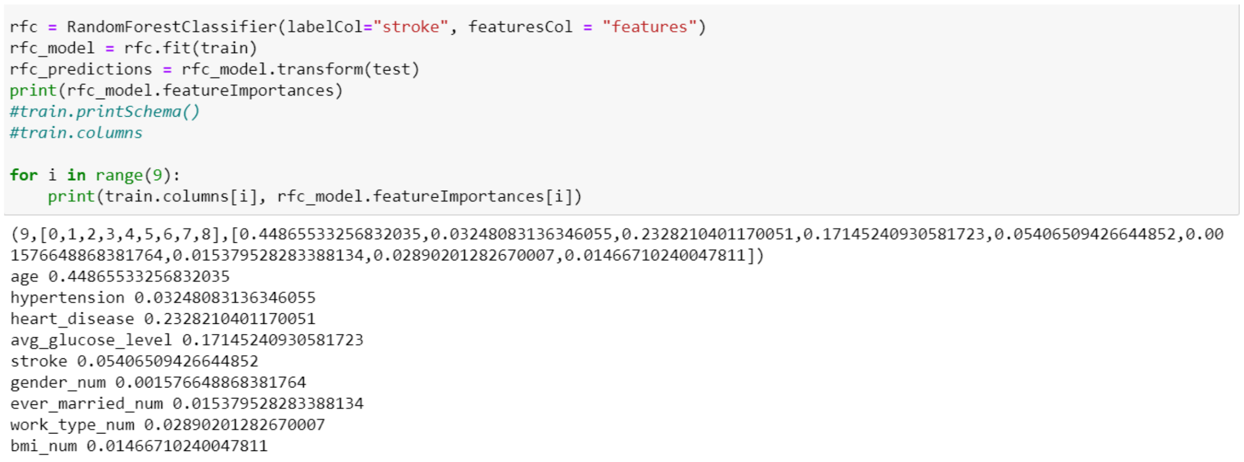
```
from pyspark.ml.classification import MultilayerPerceptronClassifier
print("Neural network")
start = time.time()
mlp = MultilayerPerceptronClassifier(labelCol="stroke", featuresCol = "pca_features",
                                     maxIter=100, layers=[3, 16, 32, 2], stepSize=0.5)
mlp_model = mlp.fit(train_oversample_pca)
mlp_predictions = mlp_model.transform(test_pca)
end = time.time()
print("Time Elapsed: ", end - start)

binary_eval = BinaryClassificationEvaluator(labelCol = 'stroke')
print("Accuracy: ",binary_eval.evaluate(mlp_predictions))

tp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 1)].count()
tn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 0)].count()
fn = mlp_predictions[(mlp_predictions.prediction == 0) & (mlp_predictions.stroke == 1)].count()
fp = mlp_predictions[(mlp_predictions.prediction == 1) & (mlp_predictions.stroke == 0)].count()

print("Confusion Matrix")
print("[" + str(tp) + "  " + str(fp))
print(" " + str(fn) + "  " + str(tn) + "]" + "\n")

precision = float((tp)/(tp + fp))
recall = float((tp)/(tp + fn))

print("Precision: ",precision)
print("Recall: ",recall)
print("F1 :",float(2 * precision * recall / (precision + recall)))
```

Figure 61. Selected Neural Network model with predefined parameters



```
Neural network

22/10/15 22:27:52 WARN BLAS: Failed to load implementati
22/10/15 22:27:52 WARN BLAS: Failed to load implementati

Time Elapsed:  24.354933500289917
Accuracy:  0.8176837572506901
Confusion Matrix
[142  2798
 32  5701]

Precision:  0.04829931972789116
Recall:  0.8160919540229885
F1 : 0.09120102761721259
```

Figure 62. Selected Neural Network model running result



```python
rfc = RandomForestClassifier(labelCol="stroke", featuresCol = "features")
rfc_model = rfc.fit(train)
rfc_predictions = rfc_model.transform(test)
print(rfc_model.featureImportances)
#train.printSchema()
#train.columns

for i in range(9):
    print(train.columns[i], rfc_model.featureImportances[i])
```

```
(9,[0,1,2,3,4,5,6,7,8],[0.44865533256832035,0.03248083136346055,0.2328210401170051,0.17145240930581723,0.05406509426644852,0.00
1576648868381764,0.015379528283388134,0.02890201282670007,0.01466710240047811])
age 0.44865533256832035
hypertension 0.03248083136346055
heart_disease 0.2328210401170051
avg_glucose_level 0.17145240930581723
stroke 0.05406509426644852
gender_num 0.001576648868381764
ever_married_num 0.015379528283388134
work_type_num 0.02890201282670007
bmi_num 0.01466710240047811
```

Figure 63. Feature Importance

7.3 The output of models (Search for patterns)

From the result, the algorithm accuracy was 81.77%, the recall score is 81.6%, which indicated a good performance of the used Neural Network algorithm for stroke prediction in the project. Recall that the business goal of this data mining process is to discover patterns among attributes and their impact on stroke. This process is also considered to assess the importance of the attributes and draw a conclusion about the risk factors for each feature.

The main findings are listed below:

1. The age factor is much more influential than other medical conditions (hypertension, heart disease). As seen in Figure 64, age is much more important than other medical-related information. Most patients get stroke within the age range of 60 – 80 years old.

```
seaborn.boxplot(x=featured_data['stroke'],y=featured_data['age'])
```

```
<AxesSubplot: xlabel='stroke', ylabel='age'>
```

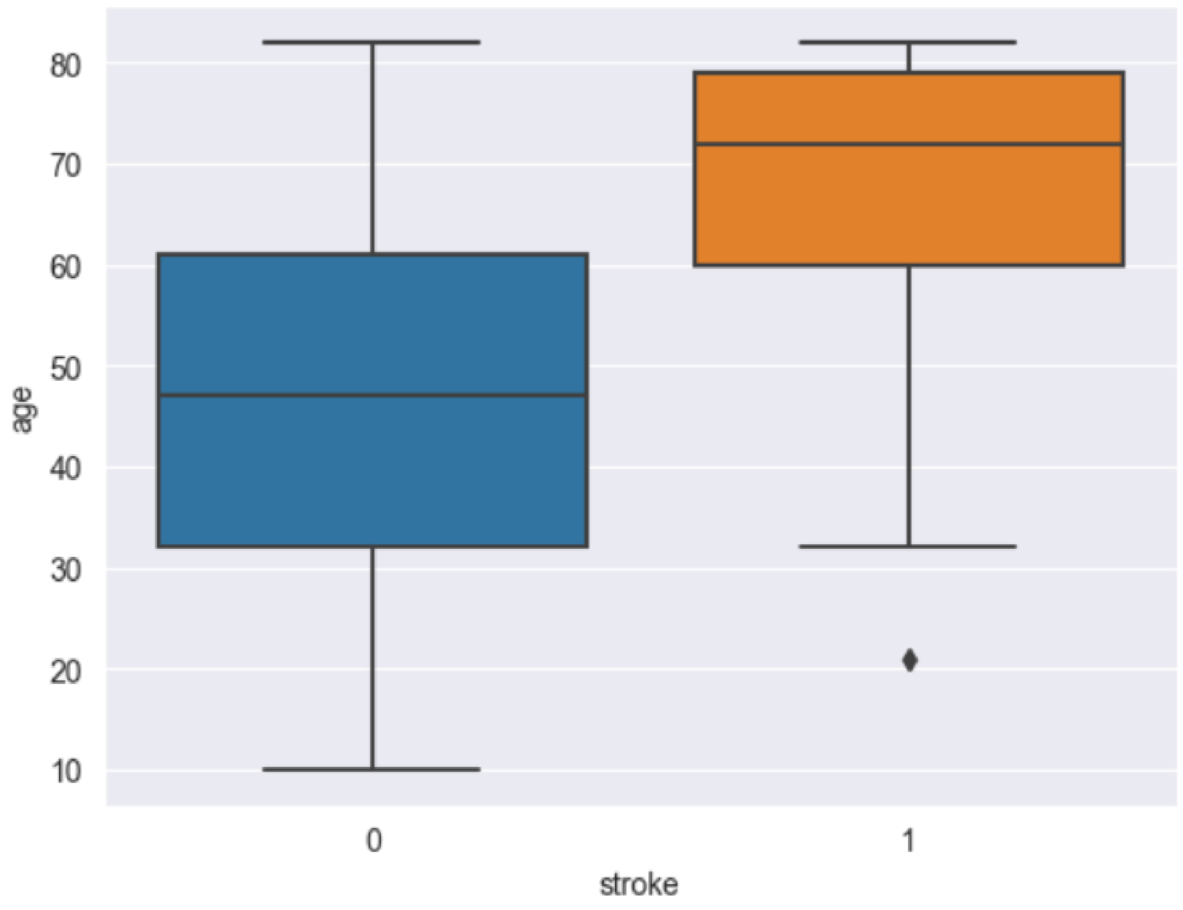


Figure 64. relationship of age and stroke

2. The average glucose is the second important predictor as the medical condition of stroke. Patients with higher average glucose level is more likely to get stroke. Most patients get stroke with the average glucose level in the range 113 – 180 mg/dL as shown in Figure 65. The normal blood glucose range is 70 to 99* mg/dL (Cleveland Clinic medical professional, 2018).

```
#plot relationship between attribute and stroke
seaborn.boxplot(x=featured_data['stroke'],y=featured_data['avg_glucose_level'])
```

```
<AxesSubplot: xlabel='stroke', ylabel='avg_glucose_level'>
```

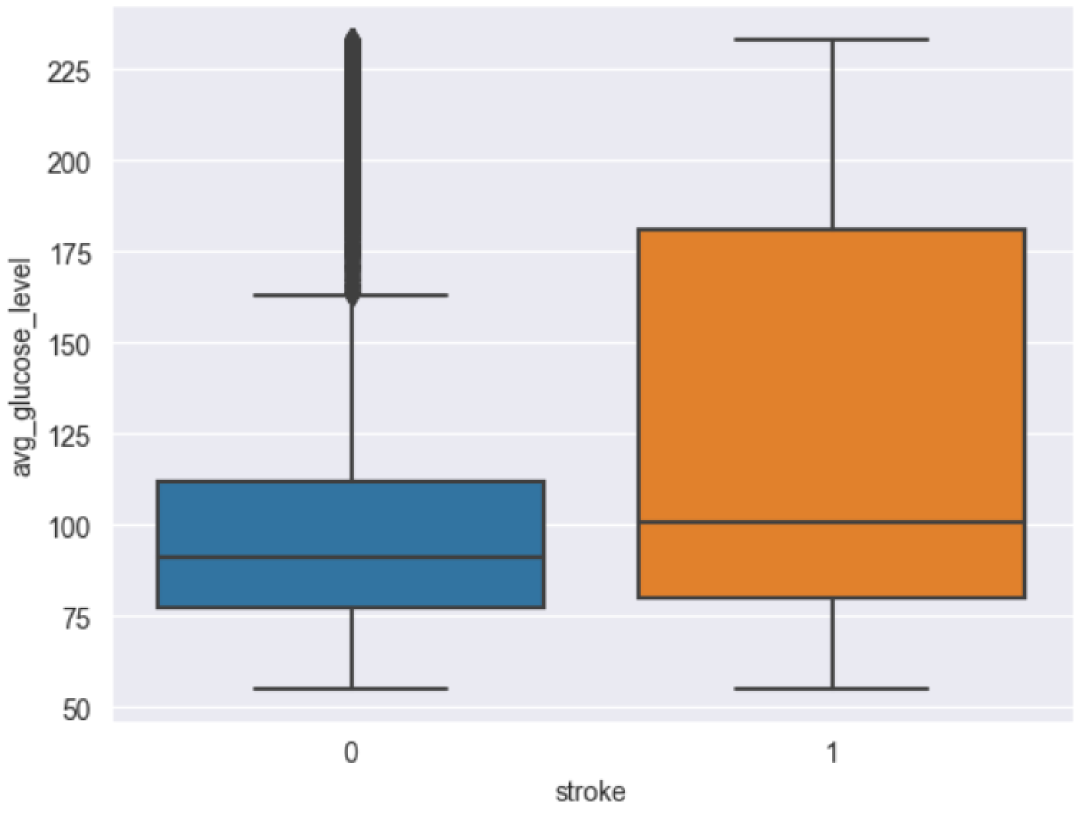


Figure 65. relationship of average glucose level and stroke

3. From the result in PCA, we did data reduction/dimensionality reduction is also required to avoid the mislead of algorithm/model. We can see that the distance between stroke patients and non-stroke patients is too close, which means most stroke the patients has very similar features as non-stroke patients, as shown in Figure 66. So, we can justify Neural Network model's parameters combination can do deep learning through max runs, layers, and structure when the dataset is complex or hard to prediction as in this case, which can identify stroke patients better. So that we can make better predictions.

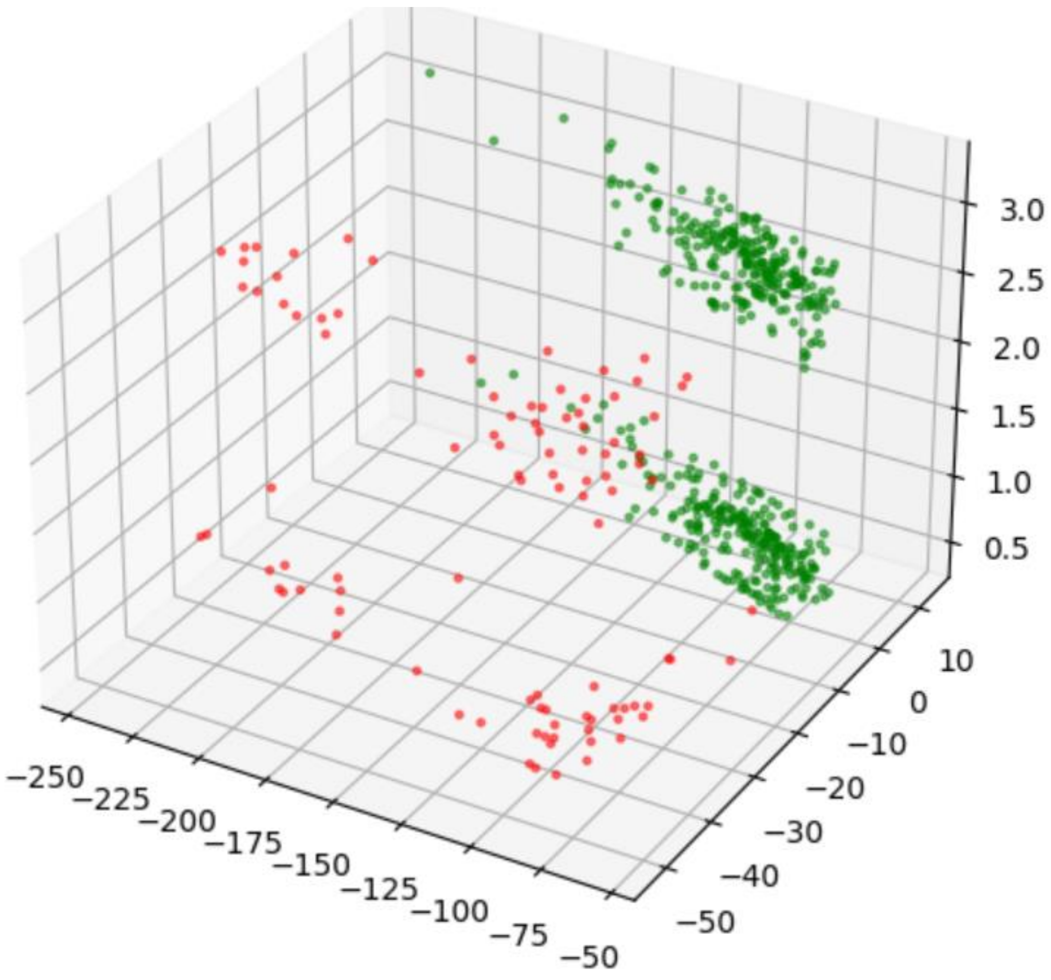


Figure 66. PCA data reduction

4. According to pattern 3, we get to know that stroke patients and non-stroke patients have similar features at some point. So, in addition, a combination of several factors will have an impact on the final stroke conditions. As the patients who has higher glucose level than normal range and elder age would have more chance to get stroke.

8. Interpretation

8.1 Data Mining Pattern

The quantity and quality of the dataset are good because it provides a sufficient number of cases for training and the test sample covers most of the cases. The use of the oversampling technique repeating algorithm balanced the difference between the two labels without causing overfitting problems.

The model used for this dataset is a neural network. It addresses the shortcomings of random trees and random forests for space-based classification, as the characteristics of stroke patients and non-stroke patients are similar. It also addresses the disadvantage of svm, which projects data to higher latitudes for classification and is not very efficient when there are many

observed samples. As well as the shortcomings of logistic regression, when the feature space is large, logistic regression does not perform very well and is prone to underfitting and generally less accurate.
Neural networks were able to predict outcomes more accurately. 82% recall and accuracy proved that the model was usable and that the model matched the dataset well. Therefore, it is a good model.

Pattern 1：
Age is a much more important factor than other medical conditions (hypertension, heart disease) As shown in Figure 60, age is much more important than other medically relevant information. Most patients have a stroke between the ages of 60 and 80. It is indisputable that, when combined with blood glucose, older people are more likely to have high blood glucose and therefore more likely to cause a stroke.

Pattern 2：
The age factor is much more influential than other medical conditions (hypertension, heart disease). As shown in Figure 60, age is much more important than other medically relevant information. Most patients had a stroke between the ages of 60 and 80 years. This is indisputable, and in combination with blood glucose, the probability of occurrence of high blood glucose is higher in the elderly and therefore causes a greater chance of stroke.

Pattern 3：
The analysis of the importance of different attributes concluded that the average blood glucose level and BMI had a greater degree of influence than other medically relevant factors. the BMI value as well as the average blood glucose level showed the lifestyle of a person. According to the research paper, a person who is obese has a higher chance of getting a stroke than others, and a person with a lower BMI and average blood sugar maintains a good lifestyle. Therefore, they are less likely to have a stroke.

Pattern 4：
Heart disease is also an important factor in influencing and predicting stroke, and patients with a history of heart disease or related disorders are more likely to have a stroke. It is therefore clear that the study of factors that impress stroke should also focus on the patient's own underlying disease.

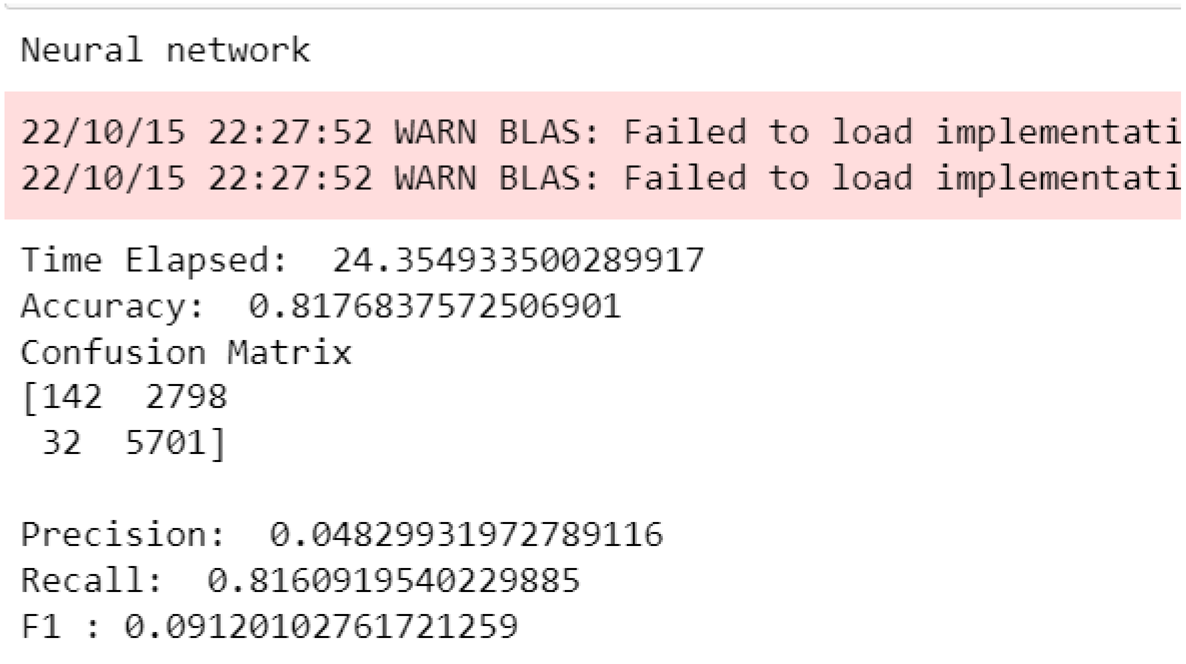
Pattern 5：
According to model 4, we can know that stroke patients and non-stroke patients have similar characteristics. So, in addition to that, the combination of several factors will have an impact on the final stroke situation. This is because patients with higher-than-normal blood glucose levels and older age will have a higher chance of having a stroke.

Pattern 6：

The type of work and smoking habits has a smaller impact on stroke. Figure 63 shows the importance of the attributes. It is clear from the figure that job category and smoking habits influence stroke, but the effect is limited. As a factor, these two attributes have no direct effect on stroke. However, these attributes as part of lifestyle, job type represents the stress and environment a person is exposed to, while smoking status influences the presence of hypertension and heart disease. Thus, the effects of job type and smoking habits on stroke are indirect.

8.2 Data Visualization

There are several visualisations being applied to the current model.
Figure 67 is showing the model execution result.
Figure 68 is showing the ROC curve of the model execution. A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. The curve plots two parameters: True Positive Rate and False positive rate.
Figure 69 is the bar plot showing the importance of each attribute.
According to the importance of attributes, the most 2 important predictors will show the relationship with stroke.
Figure 70 is showing the relationship between age and stroke.
Figure 71 is showing the relationship between average glucose level and stroke.



```
Neural network

22/10/15 22:27:52 WARN BLAS: Failed to load implementati
22/10/15 22:27:52 WARN BLAS: Failed to load implementati

Time Elapsed:  24.354933500289917
Accuracy:  0.8176837572506901
Confusion Matrix
[142  2798
 32  5701]

Precision:  0.04829931972789116
Recall:  0.8160919540229885
F1 : 0.09120102761721259
```

Figure 67. Selected SVM model running result

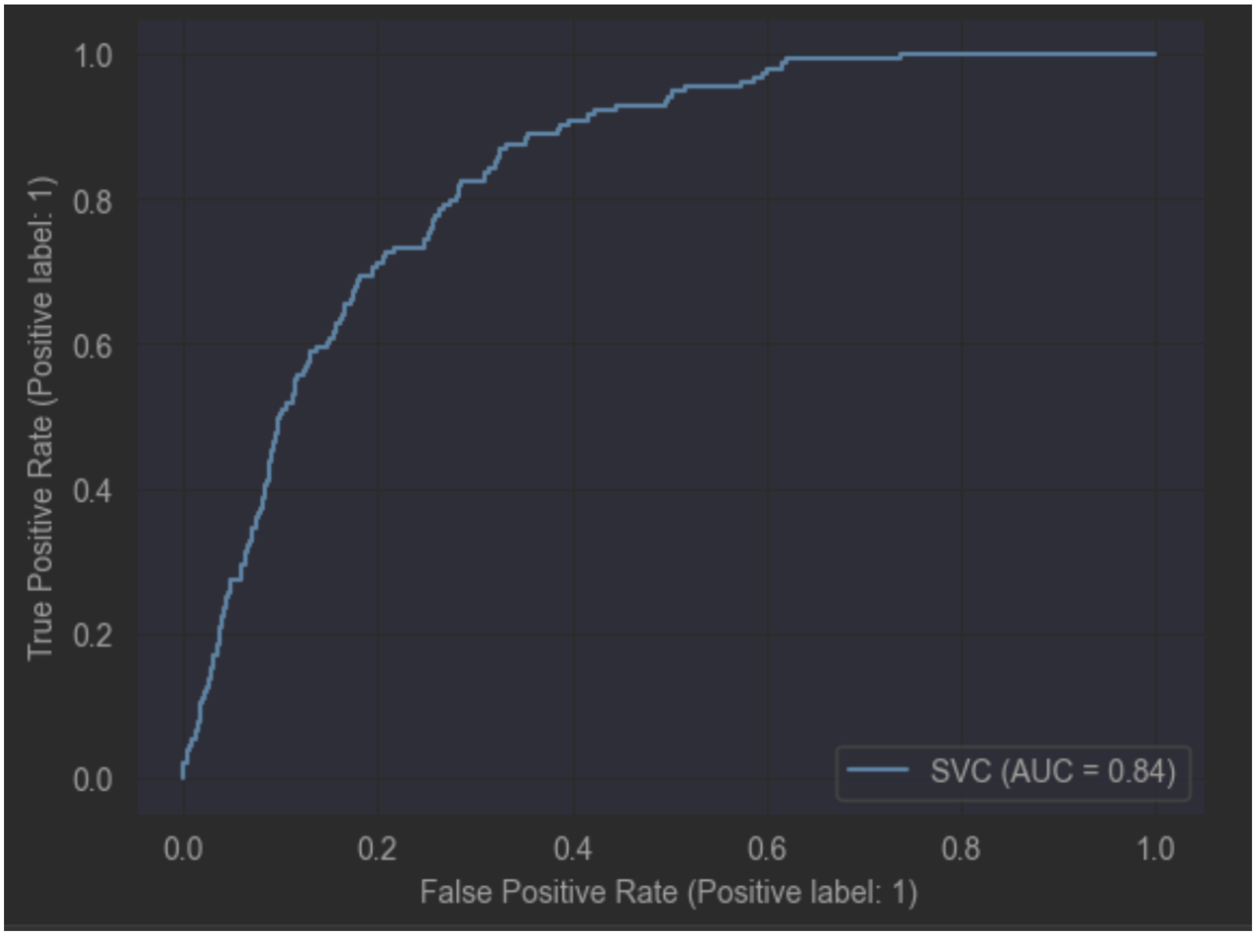


Figure 68. ROC curve

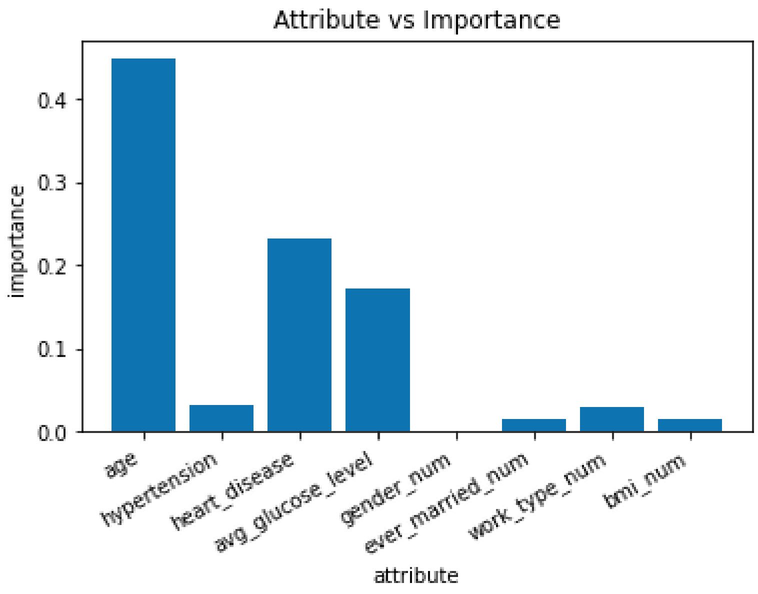


Figure 69. Attribute importance

```
seaborn.boxplot(x=featured_data['stroke'],y=featured_data['age'])
```

```
<AxesSubplot: xlabel='stroke', ylabel='age'>
```

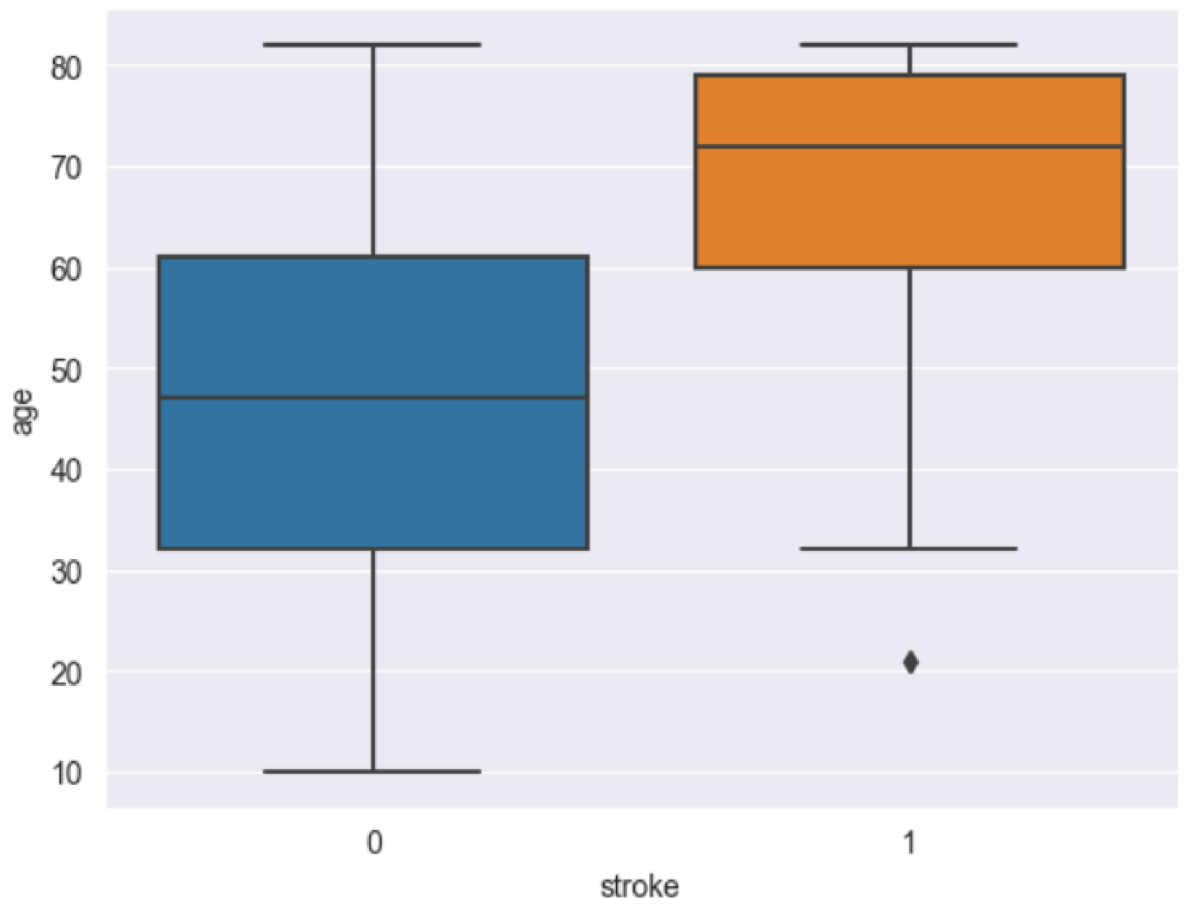


Figure 70. relationship of age and stroke

```
#plot relationship between attribute and stroke
seaborn.boxplot(x=featured_data['stroke'],y=featured_data['avg_glucose_level'])
```

```
<AxesSubplot: xlabel='stroke', ylabel='avg_glucose_level'>
```

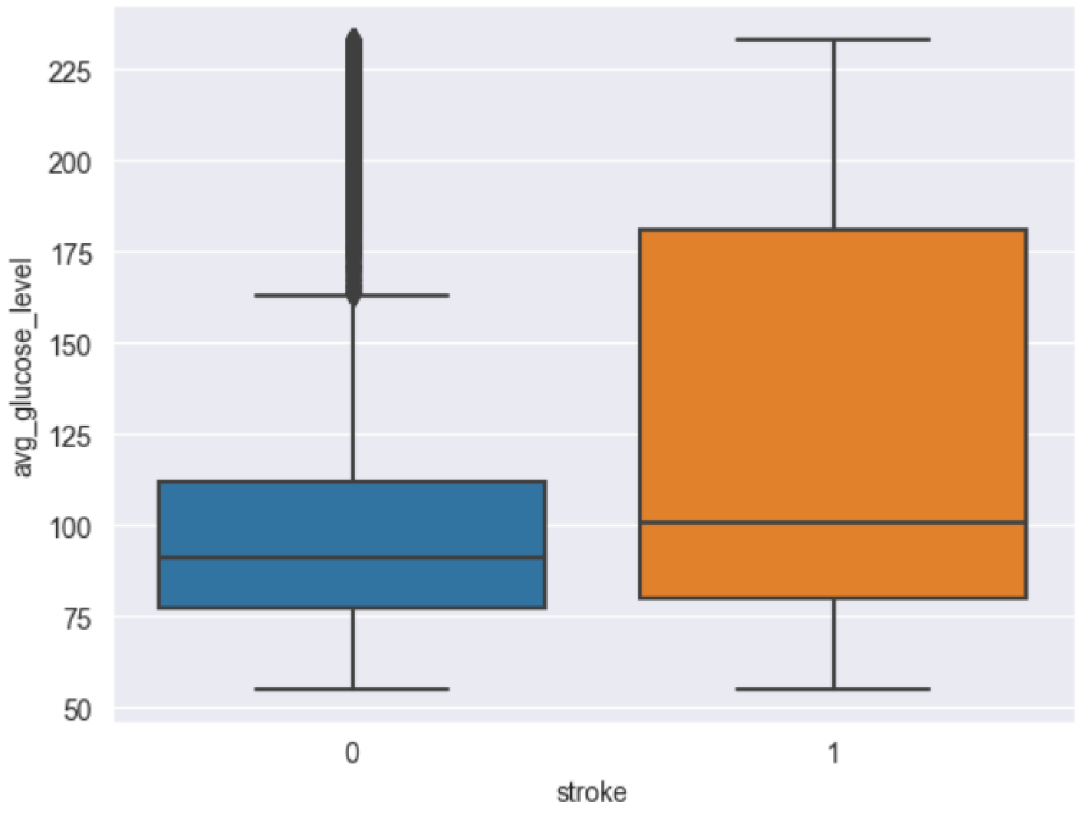
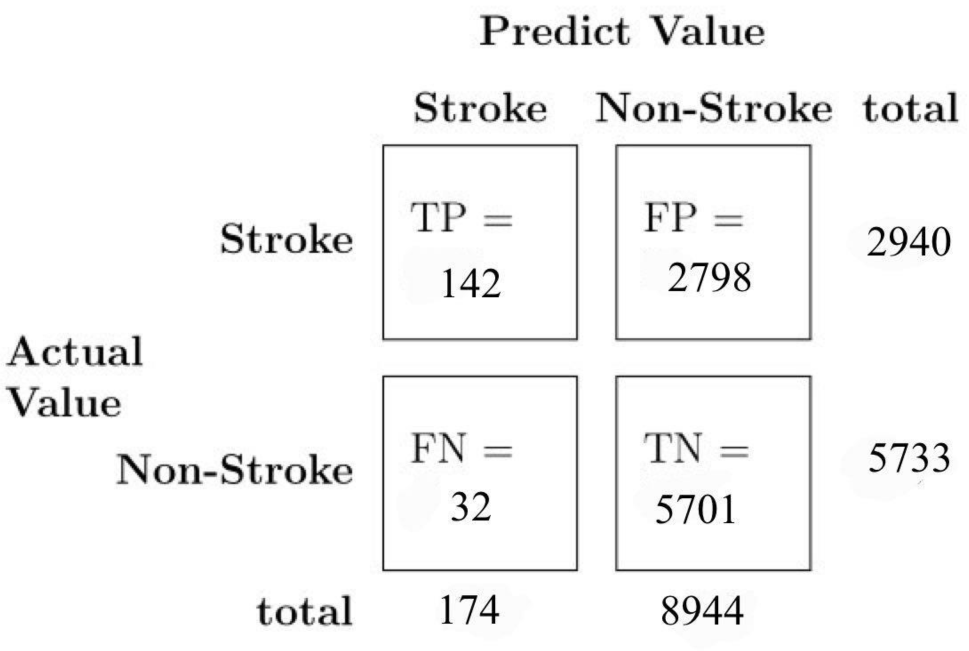


Figure 71. relationship of average glucose level and stroke

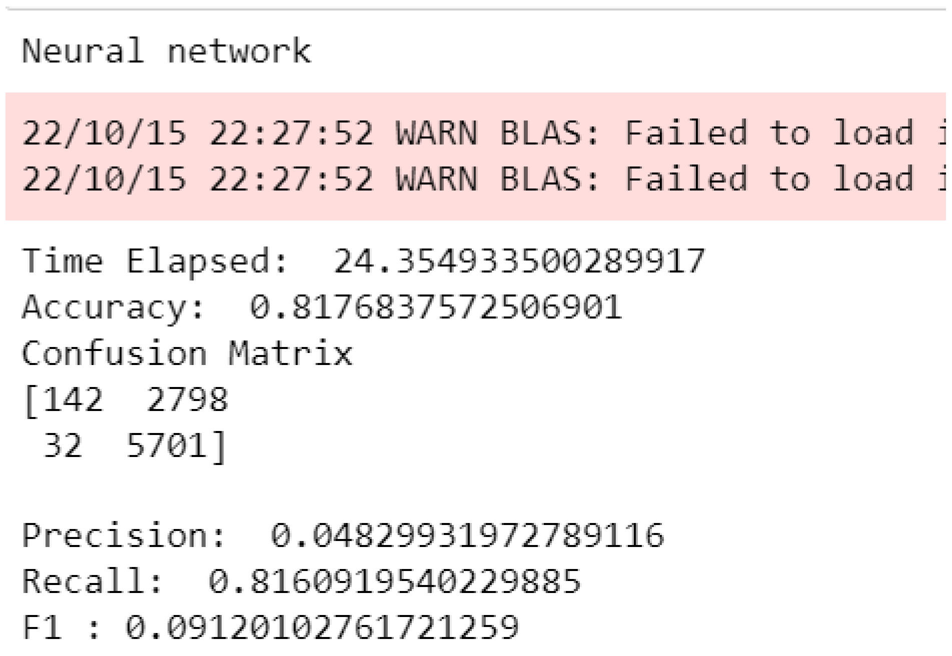
8.3 Interpretation of result, models and patterns

Patterns:
The patterns discussed above suggest that medical-related information is not the most risk factor for stroke. The original hypothesis focused on the influence of medical information, but the factors found to be more influential are now blood glucose and age. Based on the research, it is recommended that potential patients with blood glucose and age above a certain threshold undergo regular physical examinations to reduce the risk of stroke.
Model and result:



		Predict Value		
		Stroke	Non-Stroke	total
Actual Value	Stroke	TP = 142	FP = 2798	2940
	Non-Stroke	FN = 32	TN = 5701	5733
	total	174	8944	

Figure 72. Model running result confusion matrix

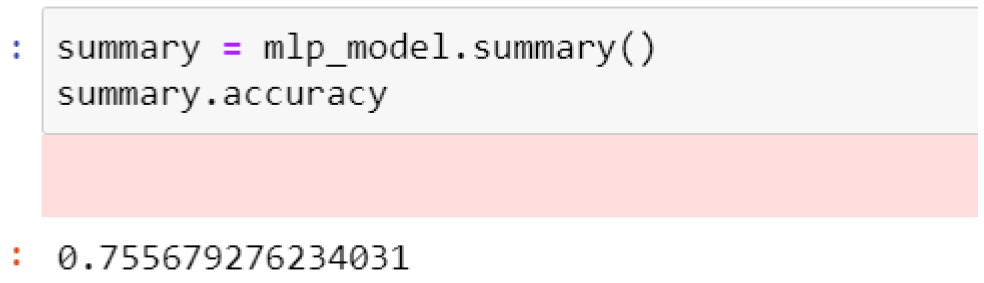


```
Neural network

22/10/15 22:27:52 WARN BLAS: Failed to load i
22/10/15 22:27:52 WARN BLAS: Failed to load i

Time Elapsed:  24.354933500289917
Accuracy:  0.8176837572506901
Confusion Matrix
[142  2798
 32  5701]

Precision:  0.04829931972789116
Recall:  0.8160919540229885
F1 : 0.09120102761721259
```

Figure 73. Model and result

The overall performance of the model was quite satisfactory, in dealing with an unbalanced database, we had to perform a rebalancing, meaning that noise was added to the dataset. However, the model still achieves an average accuracy of 82%. At the same time, the recall rate

(calculated by the confusion matrix above Figure 72) reached 82%, which is quite satisfactory. Figure 68 shows the ROC curve, indicating that the model performed well. The model is able to perform data prediction for both stroke and non-stroke, and the model itself is usable and valid.
However, overall, the accuracy and recall of this model are still not high enough to accurately achieve most predictions, but it is far from being ready for use for medical purposes, and more data needs to be collected for input and training, and more data is needed for testing. However, this model needs further improvement and involves more predictors and adjusting more parameters to make the model more accurate and usable for medical purposes in the future. From a data perspective, the model still sees relatively little information about the candidates for stroke. Even with the test dataset for detection, overfitting problems may still occur, resulting in an insensitive model.
The further evaluation of the mode, we get the accuracy when the model is training using a training dataset, the accuracy is 75.57% as shown in Figure 74, which is lower than the model running for the test dataset (82%), which means the model has a good generalization.



```
: summary = mlp_model.summary()
  summary.accuracy

: 0.755679276234031
```

Figure 74. Model training dataset accuracy

8.4 Assess and Evaluations of result, model and patterns

The assessment and evaluation process is similar to an after-action review, focusing on errors that occurred during the evaluation process and potential ways to improve them.

Evaluation of patterns:

The model finds patterns that provide relationships between attributes and targets, but the relationships themselves are not as clear. The model analysed the importance of other attributes on the target attribute and the relationship between other attributes and the target attribute. However, the effect of combining attributes with each other on the target attribute stroke is not clear; for example, the combination of heart disease and hypertension has not been assessed as a strong predictor of stroke, but there is now much literature strongly supporting these factors. Therefore, this will affect the accuracy of the results and conclusions. Potential solutions are to add additional measurements to the current algorithm or to use multiple algorithms to perform the task.
The patterns generated by this data model are less supported by medically relevant studies. These models found that hyperglycemia and age were the largest influencing factors for stroke. However, the importance of these attributes is generated by the data mining process, and limited medical studies are being reported to support these ideas. Further improvements to

this would be to add more attributes and large amounts of data, using mixed models of models to predict outcomes and gather opinions supported by medical studies.

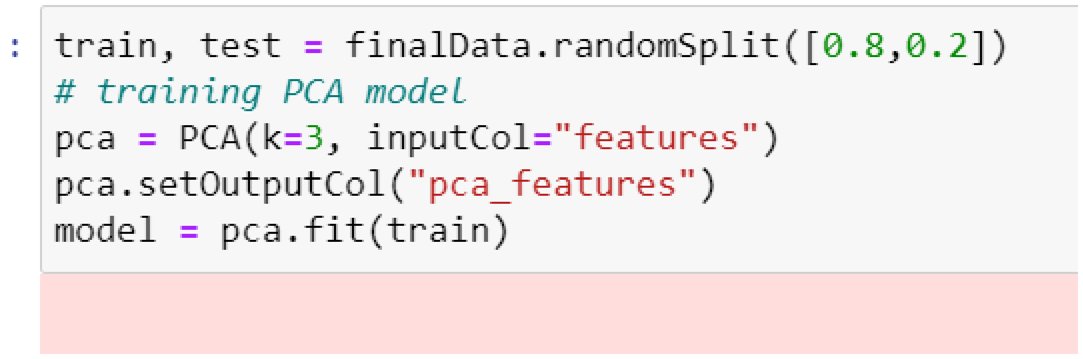
Performance evaluation of the model and results:
Current data models are good at providing good predictions from training datasets, but are not sufficient for use in medical purposes. A broader sample of data should be taken and the training success of the model should be examined further. Step-by-step adjustment of parameters to enhance the model can also introduce a mixture of multiple models. In addition, the data model should involve a large sample of many actual stroke data rather than predicted data. Predicted data are more likely to overfit the model and ultimately make the model insensitive to data that has not been seen before.

And for the results, all in all, the overall performance of the model is satisfactory. To improve the accuracy of the results, recall, roc, the accuracy of the algorithm can be further improved by adding more cases or using a larger database to do data mining, there or by doing a larger training data set. More, this result only predicts whether the patient will have a stroke or not, and for patients who have already had a stroke, this prediction is not very meaningful, so we need further improvement of the final result should be for the individual risk level of stroke. Because for patients who have already had a stroke, we are giving a prediction for what has already happened, rather than predicting the individual risk of stroke and giving warnings and recommendations.

8.5 Iterations and Improving Models

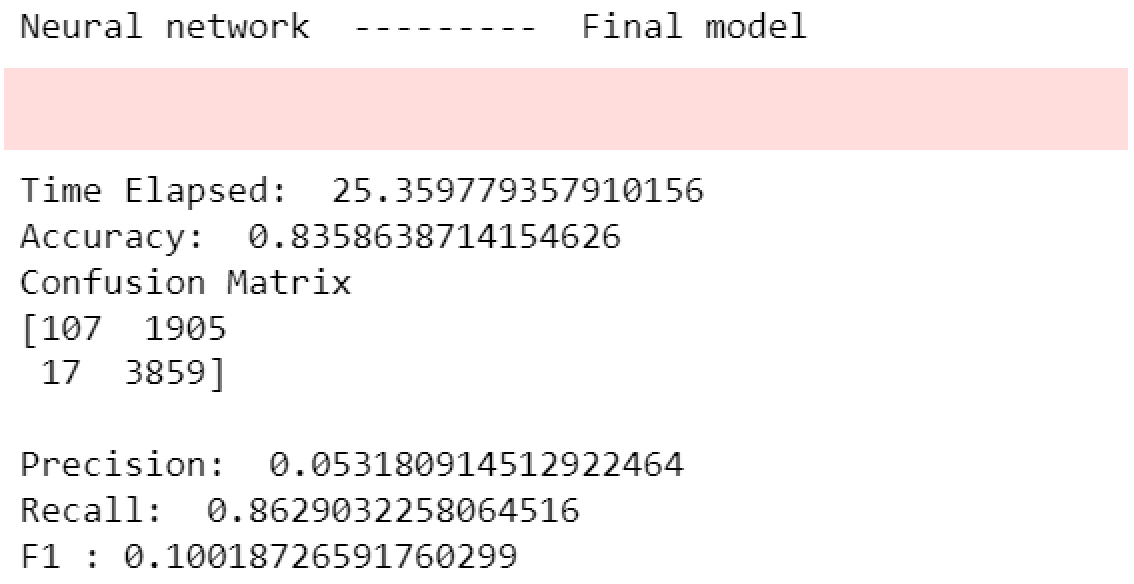
We could use the method of re-split training and test data set to improve the model, for example, re-split the training model and test model by 7.5:2.5 or 8: 2 ratios. Also, data pre-processing is quite important for Neural Netwrok model training. A suitable normalisation method could be used. In this dataset, a great number of smoking records are missing, a better way of dealing with the missing value should be explored, for example, we could set the smoke status of patients with an age smaller than 20 as non-smokers.
Iteration of the model will continue to improve the performance of the model and bring clarity to the data mining process. In order to improve the model, a review of the current steps is necessary.
Step 1: The first step in data mining is to understand the context of the dataset and to develop a business understanding of this dataset to have a grasp of the current situation. By understanding the dataset, the overall structure of this dataset will be clear, the business objectives will be clarified, and the expected results will be listed. In this step, the precise positioning of the business objectives is crucial.
Step 2: The second step is data mining. The initial data collection involves finding and identifying suitable datasets and importing them into Tableau and Python. The overall characteristics and quality of the data will then be assessed and analysed. In this step, an understanding of the structure and quality of the data set is an important step before data mining.
Step 3: The third step is data mining. This step involves data cleaning and improving the quality of the dataset using pySpark. In this step, all missing values will be removed, and extreme this

will also be processed to ensure the quality of the dataset. In addition, feature combination or integration will also take place in this step. The data cleaning and processing process will influence the final model selection and prediction discovery.
Step 4: The fourth step in data mining. In this step, imbalances in the data will be eliminated, especially in the target attributes. All unnecessary attributes with low relevance to the final result will be filtered out and removed and the unbalanced data will be rebalanced.
Step 5: The fifth step in data mining. In this step, we need to make a judgement based on the dataset and the expected results and determine the data mining method. The choice of data mining method will influence the choice of data mining algorithm. And the train and test set will be splinted before we do data mining.
Step 6: The sixth step in data mining. In this step, based on the training dataset and test dataset segmentation of the dataset, we perform model/algorithm selection and building. Data mining algorithms will be evaluated and selected based on business objectives and data mining goals.
Step 7: The seventh step in data mining. In this step, the data mining model/algorithm will be run, and the patterns are being identified.
Step 8: The eighth step of data mining. The results of the data mining, models and patterns are evaluated, and the results are analysed for validity and correctness. Attempts are made to improve the accuracy of the model, and the most convenient and easy updates can occur in the training/test data split. As the number of training sets increases, the efficiency of the algorithm increases. The new data splitting ratio was set to 8:2 as in Figure 75.



```
train, test = finalData.randomSplit([0.8,0.2])
# training PCA model
pca = PCA(k=3, inputCol="features")
pca.setOutputCol("pca_features")
model = pca.fit(train)
```

Figure 75. New train/test dataset split



```
Neural network  ---------  Final model

Time Elapsed:  25.359779357910156
Accuracy:  0.8358638714154626
Confusion Matrix
[107  1905
 17  3859]

Precision:  0.05318091451292246 4
Recall:  0.8629032258064516
F1 : 0.10018726591760299
```

Figure 76. The accuracy report after new data splitting

The application of the new data proportions resulted in an increase in overall training accuracy and in the accuracy of the test set. It can be seen that more training data will give the model higher accuracy and also validate the stability of the Neural Network model.

9. Reference

2021 Guideline for the Prevention of Stroke in Patients With Stroke and Transient Ischemic Attack: A Guideline From the American Heart Association/American Stroke Association. *Stroke* 2021; May 24: [Epub ahead of print].

Amal, L. (2020, October 26). *Heart stroke*. Kaggle. Retrieved September 18, 2022, from https://www.kaggle.com/datasets/lirilkumaramal/heart-stroke/discussion/225077

Barnett, H. J. (2005). Stroke by Cause. *Stroke, 36* (12), 2523-2525. doi: 10.1161/01.STR.0000194560.65809.47.

Berry, M. W., Mohamed, A., & Yap, B. W. (Eds.). (2019). *Supervised and unsupervised learning for data science*. Springer Nature.

Breiman Leo (2001). "Random Forests". Machine Learning 45 (1): 5–32.

Chong, J. Y. (2022, August 4). *Overview of stroke - brain, spinal cord, and nerve disorders.* MSD Manual Consumer Version. Retrieved August 15, 2022, from https://www.msdmanuals.com/home/brain,-spinal-cord,-and-nerve-disorders/stroke-cva/overview-of-stroke

Cleveland Clinic medical professional. (2018, February 21). *Blood glucose test: Levels & What They mean*. Cleveland Clinic. Retrieved September 19, 2022, from https://my.clevelandclinic.org/health/diagnostics/12363-blood-glucose-test#:~:text=A%20blood%20glucose%20test%20is,indicate%20pre%2Ddiabetes%20or%20diabetes.

Jupyter. (n.d.). *Project jupyter*. Project Jupyter. Retrieved October 11, 2022, from https://jupyter.org/

Lim, M. (2021, Jun 19). Effective ways to prevent a stroke: Take pre-emptive measures by understanding the common causes of stroke. *The Business Times* http://ezproxy.auckland.ac.nz/login?url=https://www.proquest.com/newspapers/effective-ways-prevent-stroke/docview/2542628647/se-2

W3Schools. (2022). *Pandas Tutorial*. Pandas tutorial. Retrieved September 18, 2022, from https://www.w3schools.com/python/pandas/default.asp

Wajngarten, M., & Silva, G. S. (2019). Hypertension and stroke: Update on treatment. *European Cardiology Review, 14(*2), 111–115. https://doi.org/10.15420/ecr.2019.11.1

WHO. (2020, December 9). *The top 10 causes of death*. World Health Organization. Retrieved July 29, 2022, from https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death

10. Disclaimer

"I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright.
(See: https://www.auckland.ac.nz/en/students/forms-policies-and-guidelines/student-policies-and-guidelines/academic-integrity-copyright.html, Links to an external site.).

I also acknowledge that I have appropriate permission to use the data that I have utilized in this project. (For example, if the data belongs to an organization and the data has not been published in the public domain then the data must be approved by the rights holder.) This includes permission to upload the data file to Canvas. The University of Auckland bears no responsibility for the student's misuse of data."