STROKE PREDICTION USING MACHINE LEARNING ALGORITHM

Rohan Sunil Sapkal (Registration No.: 202007778)

SYMBIOSIS CENTER FOR DISTANCE LEARNING (JULY 2020 – JULY 2022)

Table of Contents

Introduction

Approximately 11% of total deaths worldwide are caused by stroke, which is ranked number 2 by the World Health Organization (WHO). This use case targets to predict stroke risk based on input parameters such as gender, age, various diseases, and smoking status. The information in each row is relevant to the patient. This can help the patient improve health and prevent strokes by taking preventative measures. In the medical industry, this can be extremely useful.

Problem Statement

A stroke prediction dataset is used to predict whether or not a patient will have a stroke. It contains one target feature that is “stroke”. The target feature contains two instances, "0" and "1". "0" indicates that there is no chance of stroke, and "1" indicates that there is a high chance of stroke. In this problem, we are dealing with a classification problem, and we can solve it by using classification models.

Data Information

The stroke prediction dataset has 11 independent features and one target/dependent feature in total there are 12 features in a dataset with 5110 instances.

```
dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Independent features:

1. id
   Datatype: int64
   Numeric

2. Gender
   Datatype: object
   Categorical: “Male”, “Female” or “Other”

3. Age
   Datatype: float64
   Decimal

4. Hypertension
   Datatype: int64
   Binary: “0”, “1”

5. Heart_disease
   Datatype: int64
   Binary: “0”, “1”

6. Ever_married
   Datatype: object
   Categorical: “No” or “Yes”

7. Work_type
   Datatype: object
   Categorical: “children", “Govt_jov", “Never_worked", “Private" or “Self-employed"

8. Residence_type
   Datatype: object
   Categorical: “Rural” or “Urban”

9. Avg_glucose_level
   Datatype: flaot64
   Decimal

10. Bmi
    Datatype: float64
    Decimal

11. Smoking_status
    Datatype: object
    Categorical: “formerly smoked”, “never smoked”, “smokes” or “Unknown”

Dependent/target feature:

12. Stroke
    Datatype: int64
    Binary: “0” or “1”

Data Cleaning

The first thing we will do is to locate the null values in the dataset. To do this, we have used the following code:

```
dataset.isnull().sum()

id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

As we can observe there are 201 null values present in the bmi feature. Null values can be "N/A" or blank. Now, these 201 null values need to be handled. There are multiple ways to handle the null value we can drop the null values or we can do imputation means replacing the null value with the central tendency that is mean, median, or mode.

```
dataset1 = dataset.fillna(dataset.median())

dataset1.isnull().sum()

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

Here, null values present in bmi feature are replaced with the median. The reason behind replacing the null value with the median is that no records are dropped and the median works well with the outliers. If there are any outliers in the dataset will not affect the value that is been filled in the place of null values.

In the dataset, there are some error values that are "unknown". This value is not relevant and doesn't provide any information about the feature.

```
dataset1['smoking_status'].unique()

array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

So, the feature "somking_status" contains the value "unknown". To handle such irrelevant values, we can directly drop them from the dataset or impute/replace them by using mode. Mode replaces such values with the most frequent value of the entire feature.

In this case, we can directly drop the row that contains the "unknown" value.

```
dataset2 = dataset1[dataset1['smoking_status'] == 'Unknown'].index

dataset1.drop(dataset2 , inplace=True)
```

```
dataset1['smoking_status'].unique()
```

```
array(['formerly smoked', 'never smoked', 'smokes'], dtype=object)
```

```
dataset1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3566 entries, 0 to 5108
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 3566 non-null   int64
 1   gender             3566 non-null   object
 2   age                3566 non-null   float64
 3   hypertension       3566 non-null   int64
 4   heart_disease      3566 non-null   int64
 5   ever_married       3566 non-null   object
 6   work_type          3566 non-null   object
 7   Residence_type     3566 non-null   object
 8   avg_glucose_level  3566 non-null   float64
 9   bmi                3566 non-null   float64
 10  smoking_status     3566 non-null   object
 11  stroke             3566 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 362.2+ KB
```

After dropping the rows which contain the “unknown” values only 3566 instances are left. Now the data has been cleaned from null and error values.

After cleaning data it’s time to check if there are any duplicate rows in the dataset. To check duplication of rows we have used below code:

```
dataset1.duplicated().sum()
```

```
0
```

The result for the duplication is showing zero means there are no duplicate entries in the dataset.

EDA (Exploratory Data Analysis)

Sea-born is the best open-source library for statistical data visualization. (https://seaborn.pydata.org) Using the above programming language and open-source libraries, the following graphs can be generated.

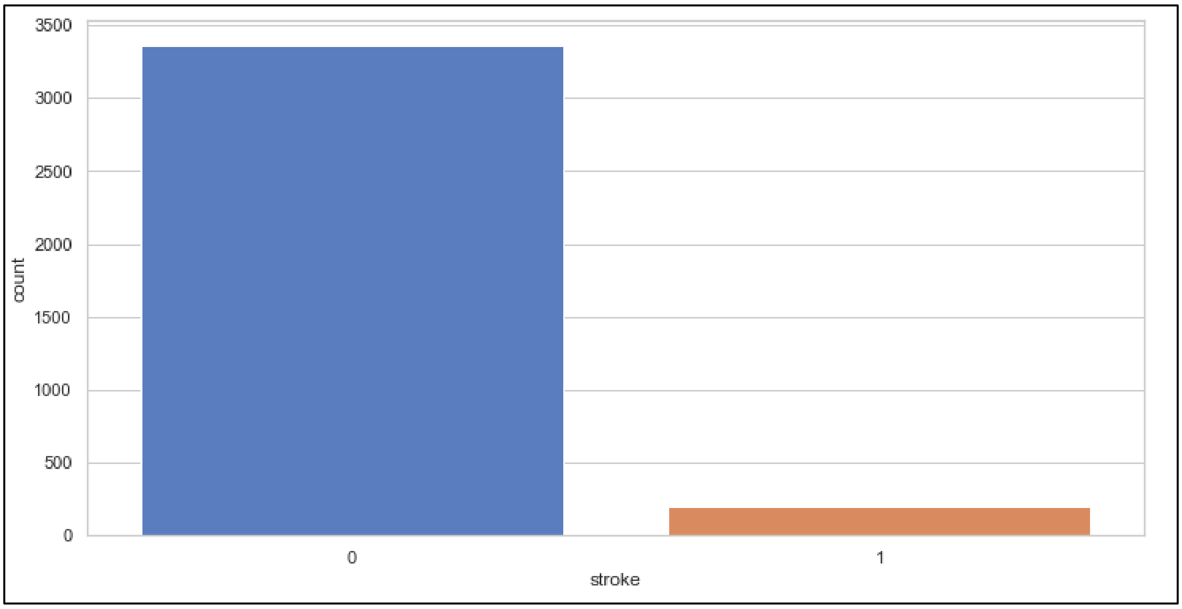
Univariate Analysis

Using a univariate analysis is the easiest way to analyze data. It is based on one variable, so your data only has one variable. It doesn't work in describing things, but merely summarizing them.

```
dataset1['stroke'].value_counts()
```

```
0    3364
1     202
Name: stroke, dtype: int64
```

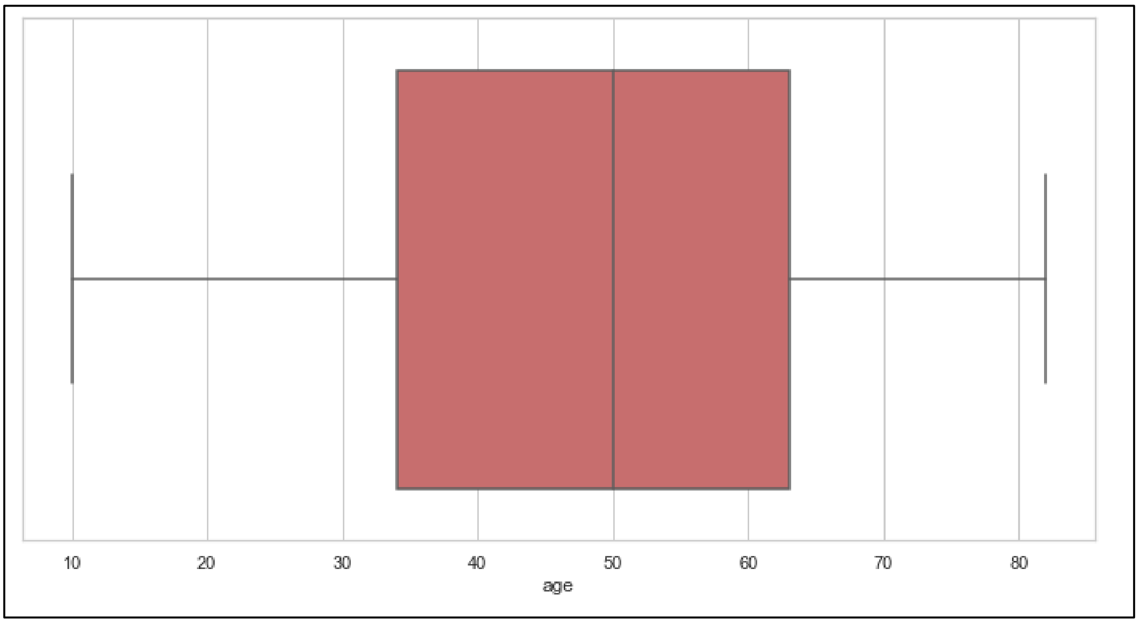
The dataset contains a greater number of records for no chance of stroke “0” and a smaller number of records for high chances of getting stroke “1”. “0” has 3364 records and “1” has only 202 records it needed to balance while data pre-processing so a model can train properly and can give high accuracy. The graphical visualization of the stokes count data is as follow:



The above graph clearly shows us that the dataset is imbalance.

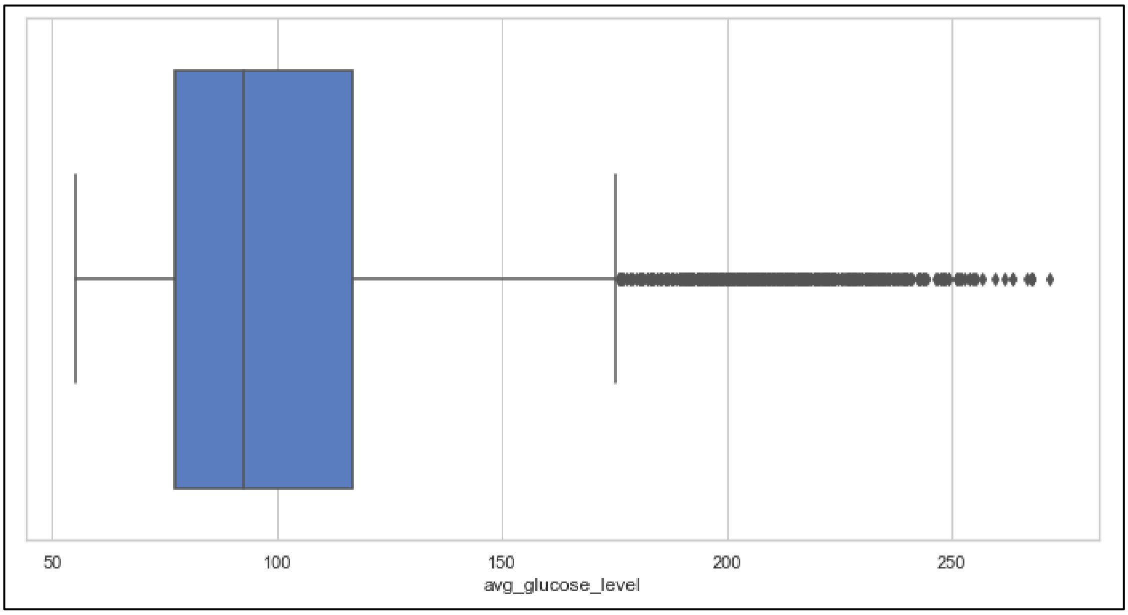
The next graphs are to show the outliers in the dataset. To show outlier the most recommend graph is box plot. So below are the graphs for each numerical feature for recognizing the outliers.

1)Age:



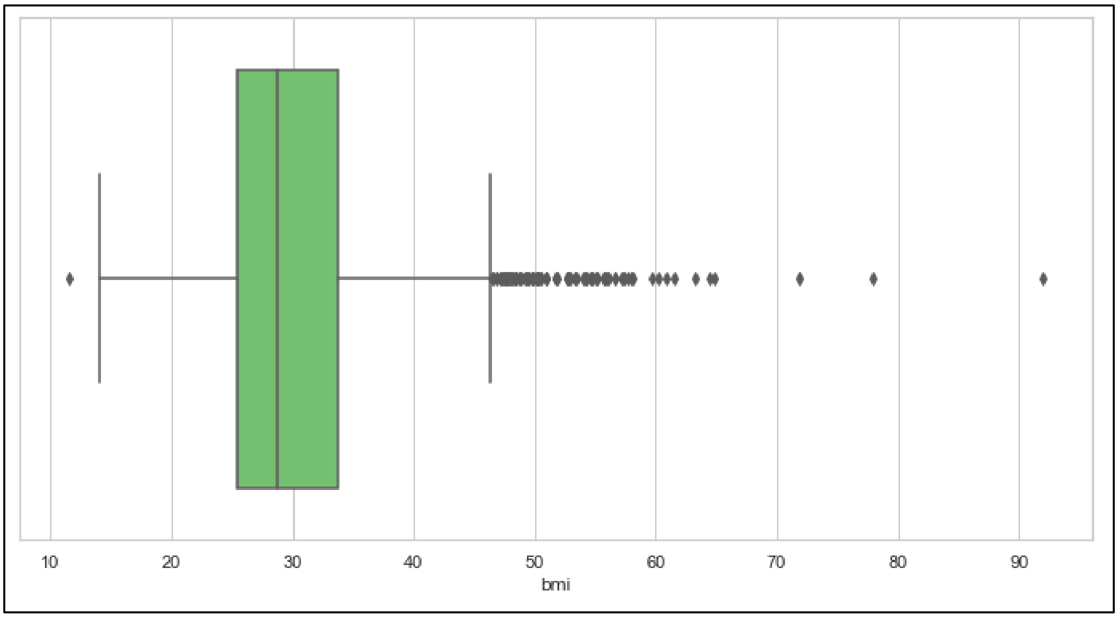
As in age we can’t observe any outlier. In the above graph, we can see that the minimum value is 10, the first quartile is somewhere between 30 to 40, the median is at 50, the third quartile is somewhere between 60 to 70 and the maximum value is above 80. Box plot is divided into above mention five terms and if the point is outside the lower and higher fence it is an outlier.

2) Avg_glucose_level:



For "avg_glucose_level" feature there are outliers. As we can see the point outside the higher fence. To handle this outlier, we can make use of Standard scaler function which shrinks the values between -1 to 1 which increase the speed of model training.

3)BMI:

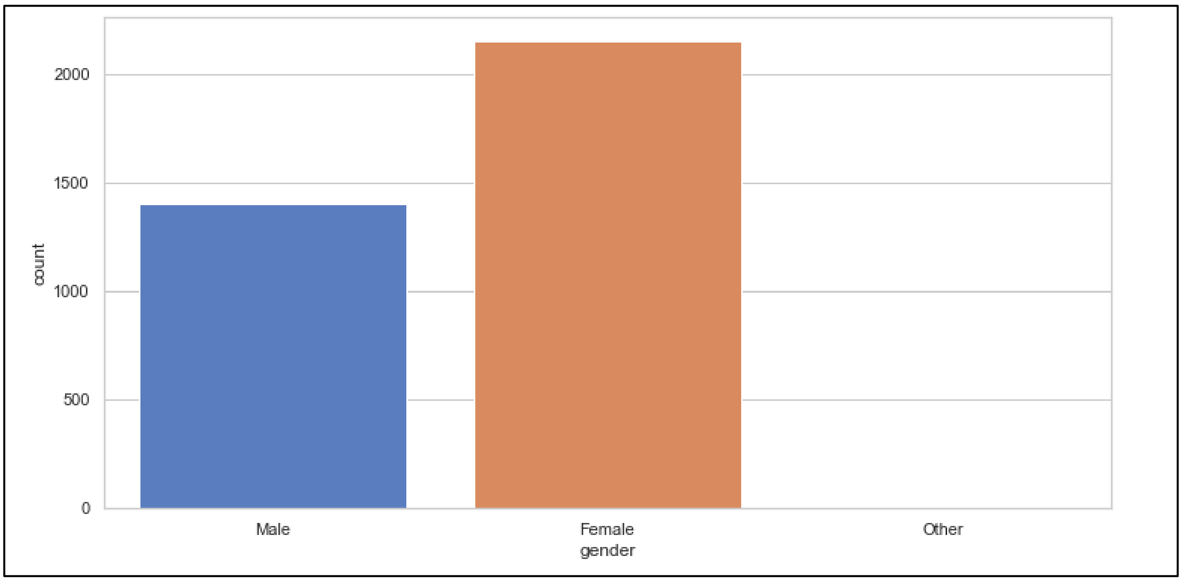


For "bmi" feature there are outliers. Outliers can be handled by the Standard scaler function to increase the speed of model training.

The next graphs will represent the distribution of values in each feature we will see each feature.

1)Gender:

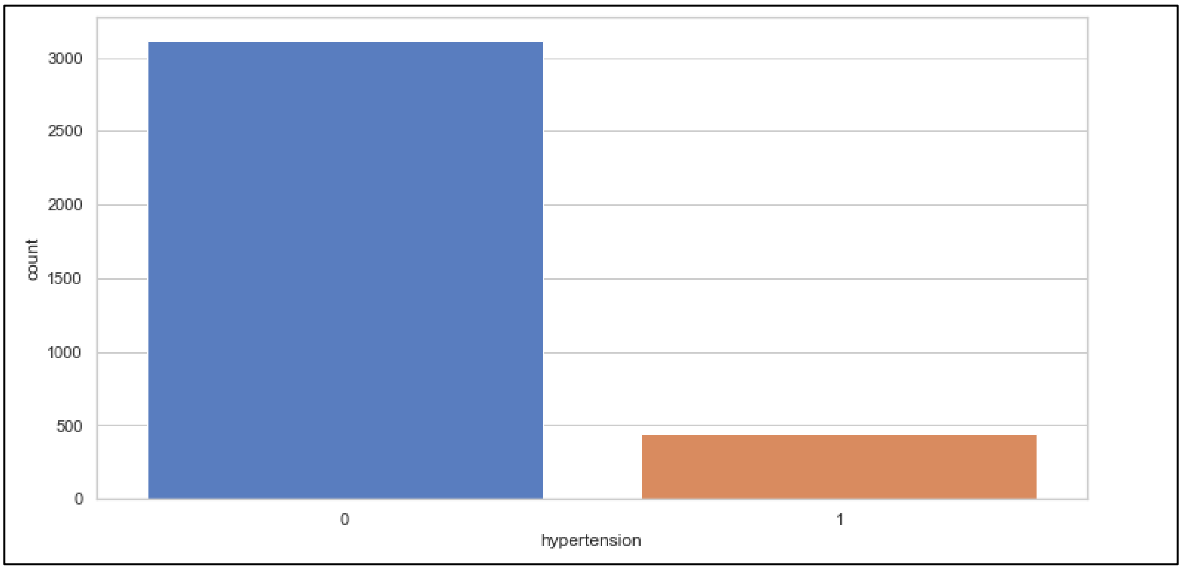
```
dataset1['gender'].value_counts()

Female    2158
Male      1407
Other        1
Name: gender, dtype: int64
```



For “gender” feature we can observe that number of female records is more than number of male record but the other has very low number of records. Female has 2158 records; male has 1407 records and other has 1 record.

2)Hypertension:

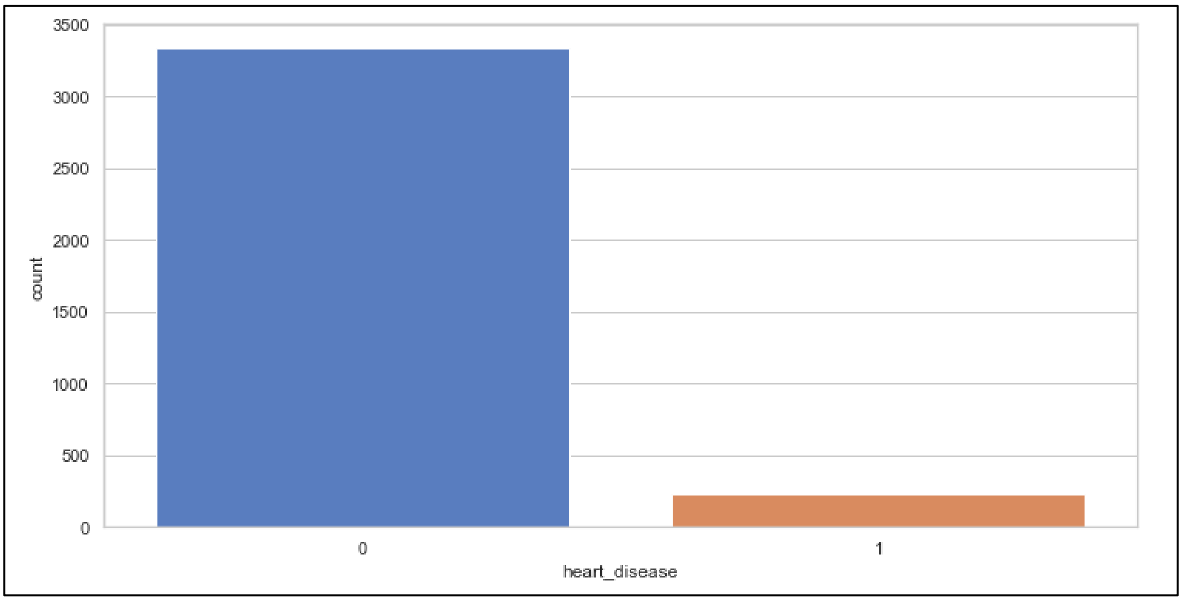
```
dataset1['hypertension'].value_counts()

0    3120
1     446
Name: hypertension, dtype: int64
```



For “hypertension” feature we can observe that maximum number of patients doesn’t have hypertension problem. 3120 patients don’t have hypertension problem but 446 patients have hypertension problem. Hypertension means having high blood pressure.

3)Heart_disease:

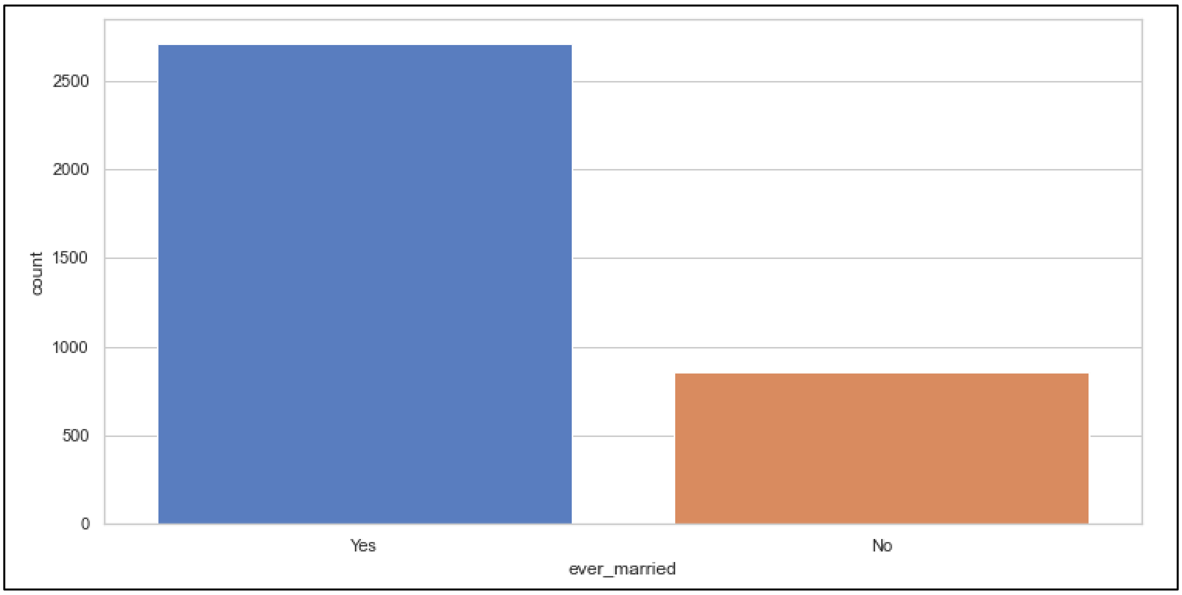
```
dataset1['heart_disease'].value_counts()

0    3338
1     228
Name: heart_disease, dtype: int64
```



For “heart_disease” feature we can observe that a maximum number of patients doesn’t have a heart disease problem. 3338 patients don’t have heart disease problems but 228 patients have a heart disease problem.

4)Ever_married:

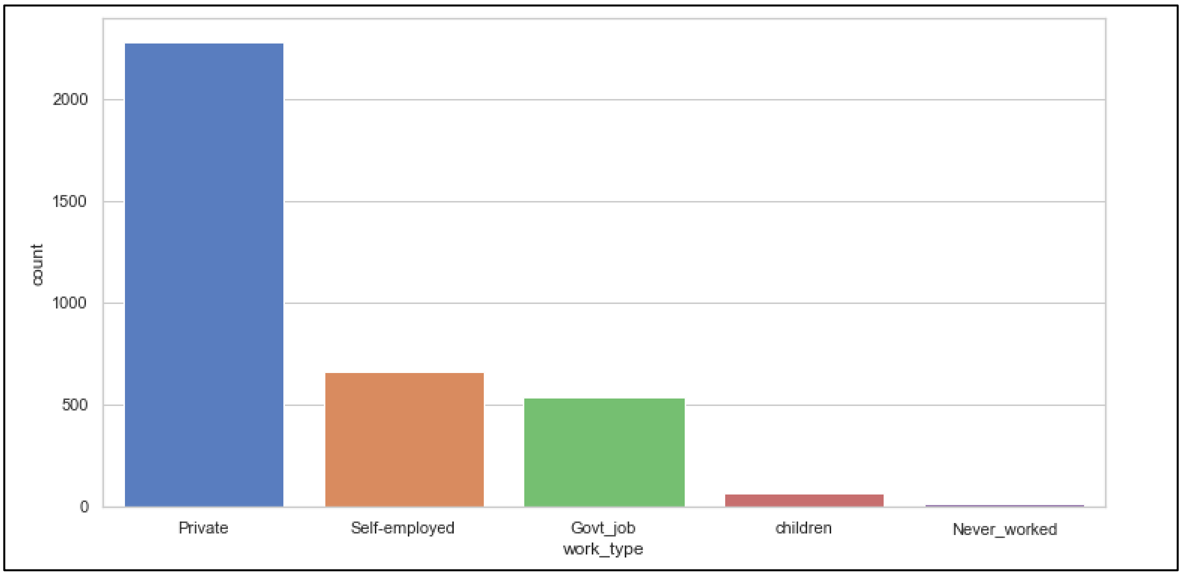
```
dataset1['ever_married'].value_counts()

Yes    2710
No      856
Name: ever_married, dtype: int64
```



In “ever_married” feature the count of married people is greater than the unmarried people. The number of married people is 2710 and number of unmarried people is 856.

5)Work_type:

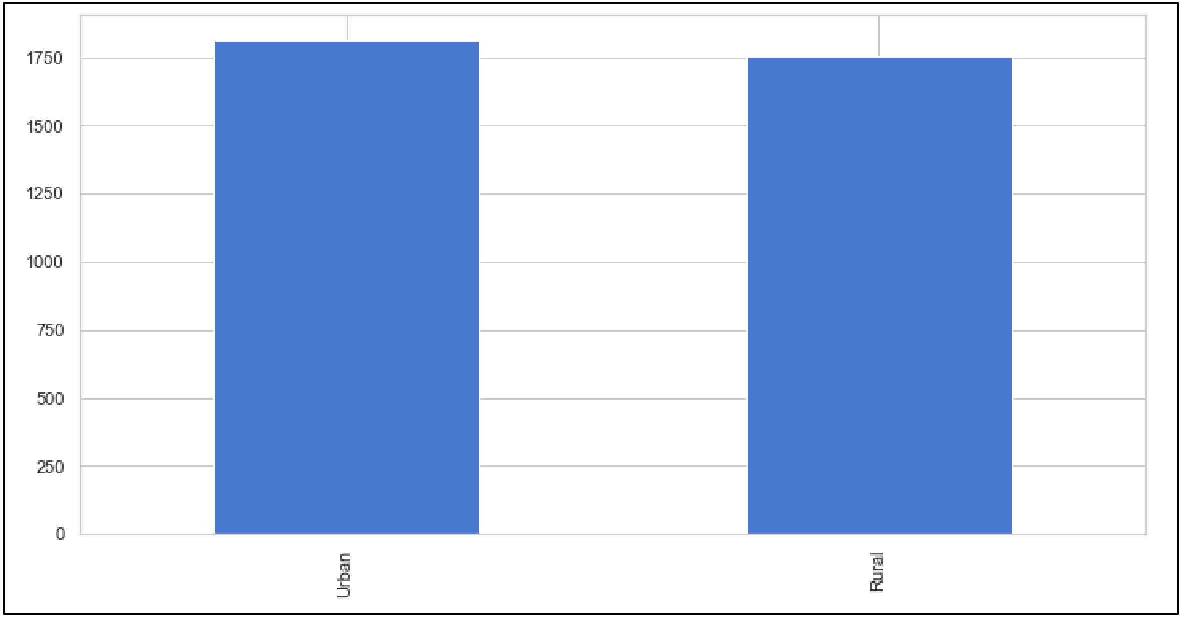
```
dataset1['work_type'].value_counts()

Private          2285
Self-employed     663
Govt_job          535
children           69
Never_worked       14
Name: work_type, dtype: int64
```



In “work_type” feature we can see that people from private job has large number of people and lower count is for never worked. Private has 2285 number of people, self-employed has 663 number of people, government job has 535 number of people and there are 69 number of children and 14 number of people who never worked.

6)Residence_type:

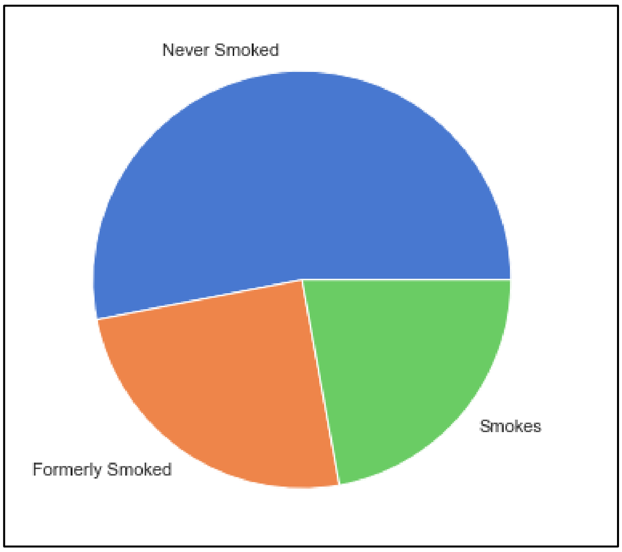
```
dataset1['Residence_type'].value_counts()

Urban    1814
Rural    1752
Name: Residence_type, dtype: int64
```



In “Residence_type” featureshows people living in urban and rural are almost same. The number of people living in urban areas are 1814 and the number of people living in rural areas are 1752.

7)Smoking_status:

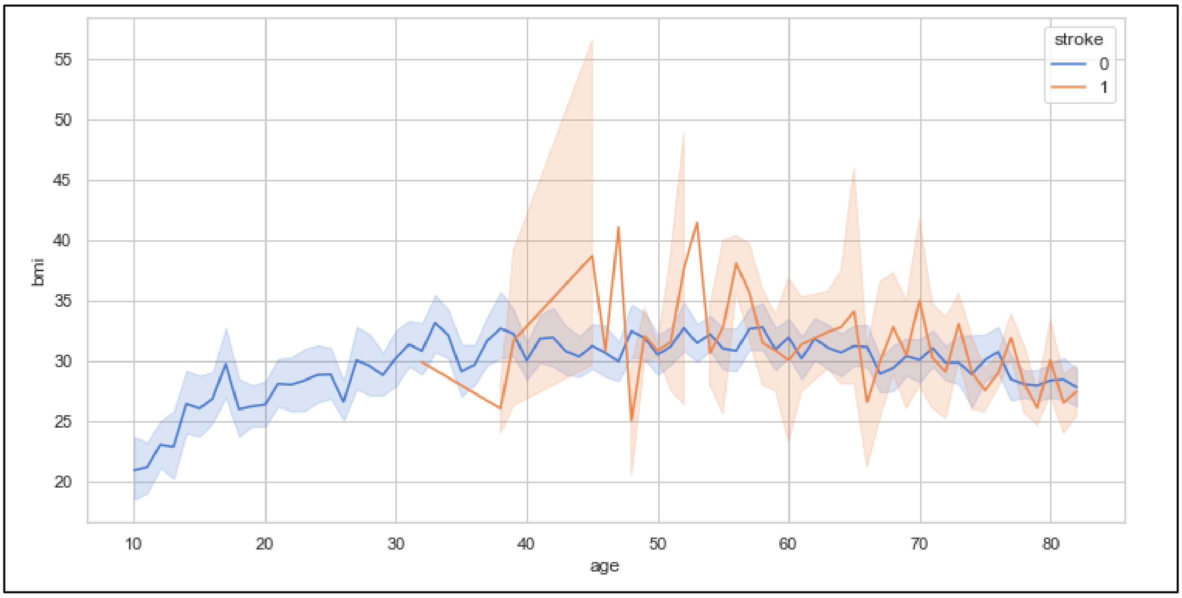
```
dataset1['smoking_status'].value_counts()

never smoked       1892
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```



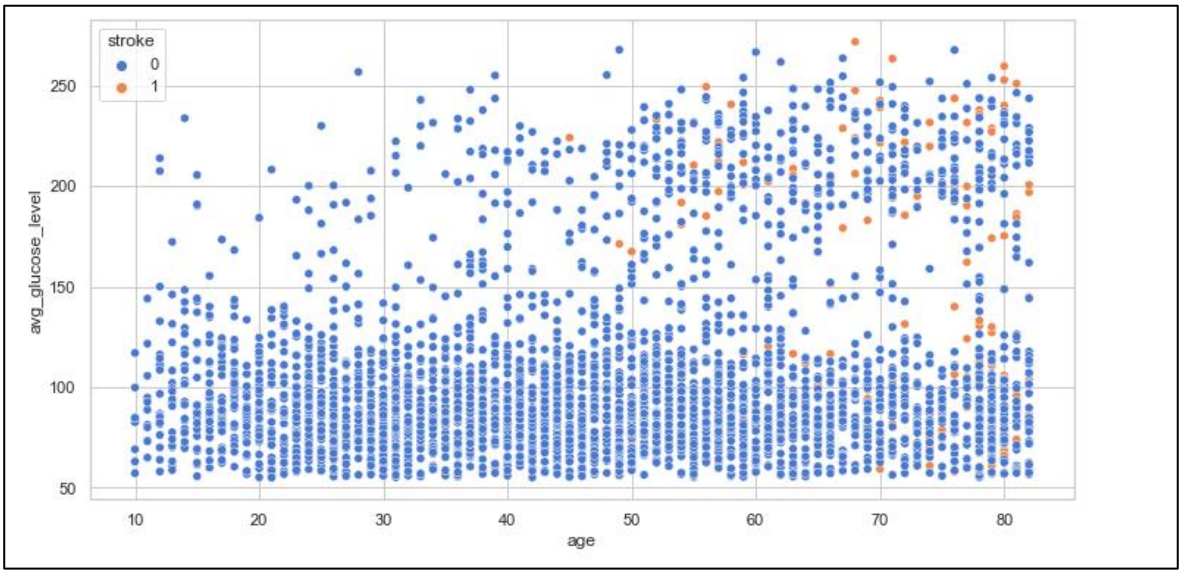
In “smoking_status” feature show that large area is acquire by never smoked and less area is acquired by smokes. The number of people that never smoked are 1892 and the number of people who smokes formerly are 885 and the number of people who always smokes are 789.

Bivariate Analysis

As one of the simplest forms of statistical analysis, the bivariate analysis consists of looking at two factors for the purpose of identifying a relationship between them. Bivariate analysis can be useful for testing the simple hypothesis of association.



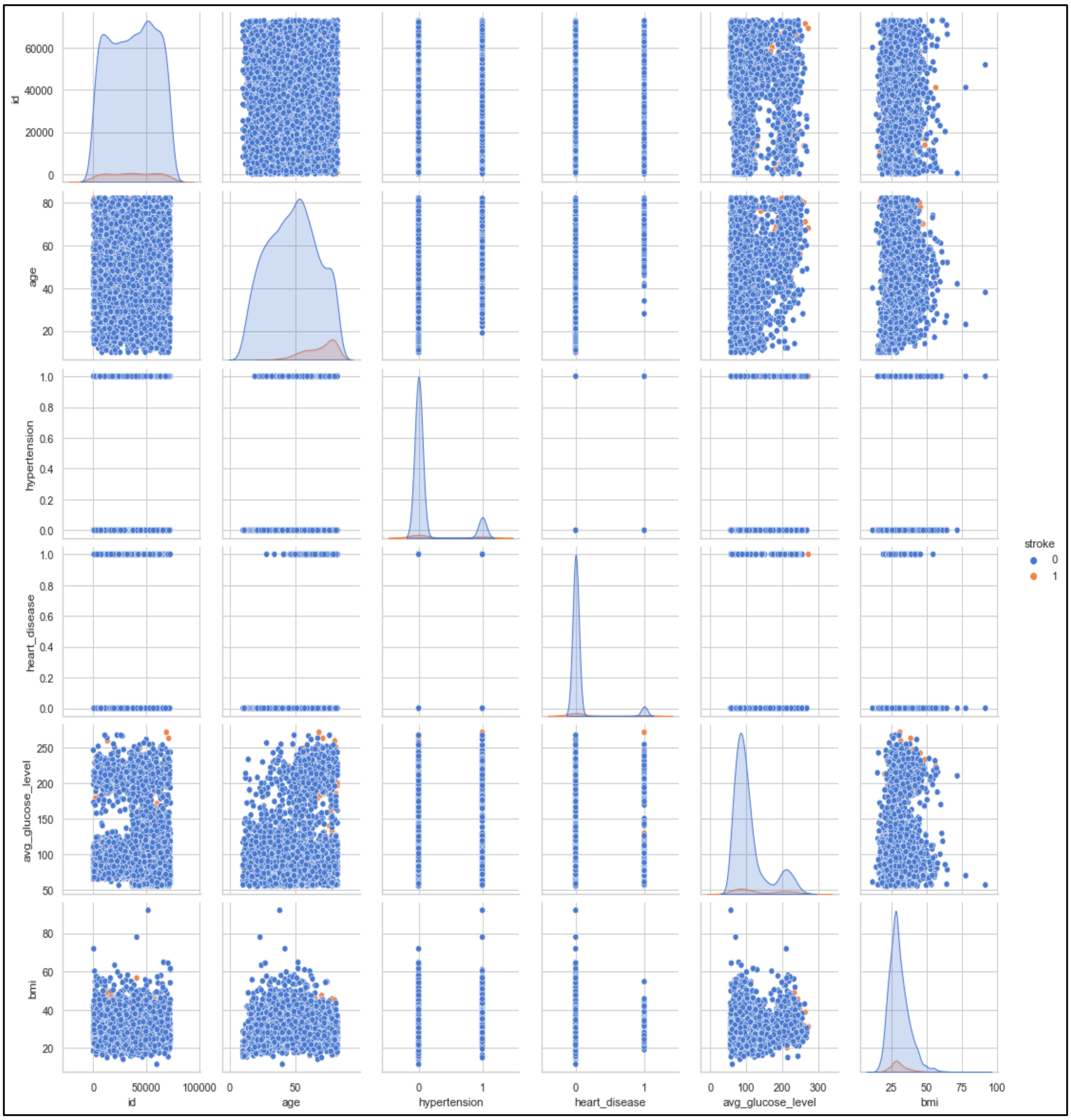
In the above graph, we can observe that if the age of the person is above 30 and the BMI of the person is less than 30 or greater than 35 can cause a stroke.



In the above graph it is been observed that if the age of the person is grater than 50 and glucose level is higher than 150 then there are chances to get a stroke.

Multivariate Analysis

The traditional definition of multivariate analysis is the statistical analysis of experiments in which multiple measurements on each experimental unit are made, and the relationship between multivariate measurements and their structure is critical.



In multivariate all features are compared with each other and every possible graph has been plotted.

Data Pre-processing

As this is classification problem all the features that are having object datatype needs to be converted into integer datatype.

By using below code, we can observe that there five object feature that needs to be converted into integer. The five features that are of object datatype are “gender”, “ever_married”, “work_type”, “Residence_type”, “smoking_status”.

```
dataset1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3566 entries, 0 to 5108
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 3566 non-null   int64
 1   gender             3566 non-null   object
 2   age                3566 non-null   float64
 3   hypertension       3566 non-null   int64
 4   heart_disease      3566 non-null   int64
 5   ever_married       3566 non-null   object
 6   work_type          3566 non-null   object
 7   Residence_type     3566 non-null   object
 8   avg_glucose_level  3566 non-null   float64
 9   bmi                3566 non-null   float64
 10  smoking_status     3566 non-null   object
 11  stroke             3566 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 491.2+ KB
```

Now to convert object datatype to integer datatype there are two types of method and that are "map()" and other one is "get_dummies()".

In this dataset we will be using "get_dummies()" method as the feature's values are not ordinal so we can't map values to the number. So now data will be in 0's and 1's. "Map()" is not used as the reason is clear that the dataset is not ordinal means if data would have values as name of months or weeks or education from school to graduation in such cases we can use map() method.

After using the "get_dummies" method below is a table containing all integer datatype features.

```
categorical = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
final = pd.get_dummies(dataset1, columns=categorical)
```

```
final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3566 entries, 0 to 5108
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              3566 non-null   int64
 1   age                             3566 non-null   float64
 2   hypertension                    3566 non-null   int64
 3   heart_disease                   3566 non-null   int64
 4   avg_glucose_level               3566 non-null   float64
 5   bmi                             3566 non-null   float64
 6   stroke                          3566 non-null   int64
 7   gender_Female                   3566 non-null   uint8
 8   gender_Male                     3566 non-null   uint8
 9   gender_Other                    3566 non-null   uint8
 10  ever_married_No                 3566 non-null   uint8
 11  ever_married_Yes                3566 non-null   uint8
 12  work_type_Govt_job              3566 non-null   uint8
 13  work_type_Never_worked          3566 non-null   uint8
 14  work_type_Private               3566 non-null   uint8
 15  work_type_Self-employed         3566 non-null   uint8
 16  work_type_children              3566 non-null   uint8
 17  Residence_type_Rural            3566 non-null   uint8
 18  Residence_type_Urban            3566 non-null   uint8
 19  smoking_status_formerly smoked  3566 non-null   uint8
 20  smoking_status_never smoked     3566 non-null   uint8
 21  smoking_status_smokes           3566 non-null   uint8
dtypes: float64(3), int64(4), uint8(15)
memory usage: 404.2 KB
```

Now dataset is ready, we need to determine X and Y:

X: X variable contains of all independent variable and for this dataset we can drop "id" as it is not relevant to dataset. The target feature "stroke" is also dropped.

Y: Y variable contains of only dependent/target feature that is "stroke".

```
X = final.drop(['stroke','id'],axis=1)
Y = final['stroke']
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3566 entries, 0 to 5108
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   age                             3566 non-null   float64
 1   hypertension                    3566 non-null   int64  
 2   heart_disease                   3566 non-null   int64  
 3   avg_glucose_level               3566 non-null   float64
 4   bmi                             3566 non-null   float64
 5   gender_Female                   3566 non-null   uint8  
 6   gender_Male                     3566 non-null   uint8  
 7   gender_Other                    3566 non-null   uint8  
 8   ever_married_No                 3566 non-null   uint8  
 9   ever_married_Yes                3566 non-null   uint8  
 10  work_type_Govt_job              3566 non-null   uint8  
 11  work_type_Never_worked          3566 non-null   uint8  
 12  work_type_Private               3566 non-null   uint8  
 13  work_type_Self-employed         3566 non-null   uint8  
 14  work_type_children              3566 non-null   uint8  
 15  Residence_type_Rural            3566 non-null   uint8  
 16  Residence_type_Urban            3566 non-null   uint8  
 17  smoking_status_formerly smoked  3566 non-null   uint8  
 18  smoking_status_never smoked     3566 non-null   uint8  
 19  smoking_status_smokes           3566 non-null   uint8  
dtypes: float64(3), int64(2), uint8(15)
memory usage: 348.4 KB
```

```
print(Y)
```

```
0       1
1       1
2       1
3       1
4       1
       ..
5102    0
5105    0
5106    0
5107    0
5108    0
Name: stroke, Length: 3566, dtype: int64
```

As mention above we have dropped the column, in X we have used axis=1 it means that drop the whole column that is mentioned in code.

In visualization we have found the outliers in the many features so to overcome from the outliers we will be using Standard scaler method.

StandardScalar() method imported from sklearn to normalize our numerical features. It normalizes features by first subtracting mean from each observation, and then dividing each observation by standard deviation. Each feature column has a mean of 0 and a variance of 1 after normalization. This means that most values will be between -1 and 1.

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```python
print(X_scaled)
```

```
[[ 0.96154954 -0.3780856   3.82627115 ...  1.740511   -1.06312135
  -0.53302866]
 [ 0.64361365 -0.3780856  -0.26135105 ... -0.57454391  0.94062639
  -0.53302866]
 [ 1.65041066 -0.3780856   3.82627115 ... -0.57454391  0.94062639
  -0.53302866]
 ...
 [ 1.70339997 -0.3780856  -0.26135105 ... -0.57454391  0.94062639
  -0.53302866]
 [-0.73410858 -0.3780856  -0.26135105 ... -0.57454391  0.94062639
  -0.53302866]
 [ 0.11372048 -0.3780856  -0.26135105 ...  1.740511   -1.06312135
  -0.53302866]]
```

As you can observe that after normalizing the X by using standard scaler method all the values are between -1 to 1. This can help to train model faster.

Conclusion

In stroke prediction dataset we followed each and every process from data cleaning to data pre-processing. In data cleaning we have replaced null values with median. Then after that we dropped the row which contains error value. Then in EDA we visualize the dataset and represented the meaningful information in the form of a graph. In Data pre-processing we have converted object datatype to integer datatype and also determine X and Y variables. Then normalize the dataset to avoid outliers.

The dataset is now ready to be analyzed by the machine learning model, and this process will be discussed in submission 2.