CSE 587: Data Intensive Computing

Project Phase #1

Heart Stroke Prediction

Team Members details

Name: Anurag Lingabathina
UB Name: anuragli
UB Number: 50442601
UB Mail: anuragli@buffalo.edu

Name: Chandra Kiran Alla
UB Name: calla2
UB Number: 50468307
UB Mail: calla2@buffalo.edu

Problem Statement:

1. Background of the problem leading to your objectives. Why is it a significant problem?

As per the World Health Organization (WHO), stroke is the second most common cause of death worldwide, accounting for roughly 11% of all fatalities. Only in the United States, for every 40 seconds, a person is affected with Heart Stroke, and for every 3 min 30 sec a person dies, and 874,613 deaths occurred in the United States in 2019 due to heart-related diseases. If the person is during the initial stage Diagnosed, then he can change his/her lifestyle like food, exercise, etc to avoid severe consequences. This is a significant problem across the globe, based on the information collected, we can estimate the current condition of the person. People who are in the nitty-gritty stage can go through early check-ups to avoid such health conditions.

2. Explain the potential of your project to contribute to your problem domain. Discuss why this contribution is crucial?

Saving a life is the most important thing in the world, even doctors sometimes can't help due to a lack of patient information. If the doctor has all the related information about the patient from time to time, he can give proper medication and guide the patient to not enter the risk of getting a heart stroke.
Based on input characteristics like gender, age, numerous diseases, and smoking status, these details are helpful in categorizing the problem based on their overall information. This will definitely help doctors in examining the basis of stroke.
This dataset is used to determine whether a patient is likely to get a stroke or not.

2. Data Source

Collected Data from Kaggle. The data set consists of 5110 rows and 12 columns.

0. **Unnamed** - It contains integers
1. **id** - Id is used for identification.
2. **gender**- To categorize between Male, Female, and Other.
3. **age** - patient's age
4. test coll_date - states the date on which data is collected
5. **hypertension** - categorizing patient who has hypertension as 1 and who doesn't have it as 0
6. **heart_disease** - 1 for the patient who already has heart disease, 0 for the patient who doesn't have it.
7. **ever_married** - Categories between 'Yes' or 'No.'
8. **work_type** - Categories between children, Govt_jov, Never_worked, Private, Self-employed
9. **residence_type** - Categories on residential type of patient between 'Urban' or 'Rural'
10. **avg_glucose_level**- a patient's average blood sugar level
11. bmi - It indicates Body Mass Index value of a patient
12. **Smoking_status** - a variable that has been recorded depending on queries regarding smoking cigarettes. formerly smoked, never smoked, smokes, not known
13. **stroke** - Possibility of a stroke, 1 for yes, 0 for No

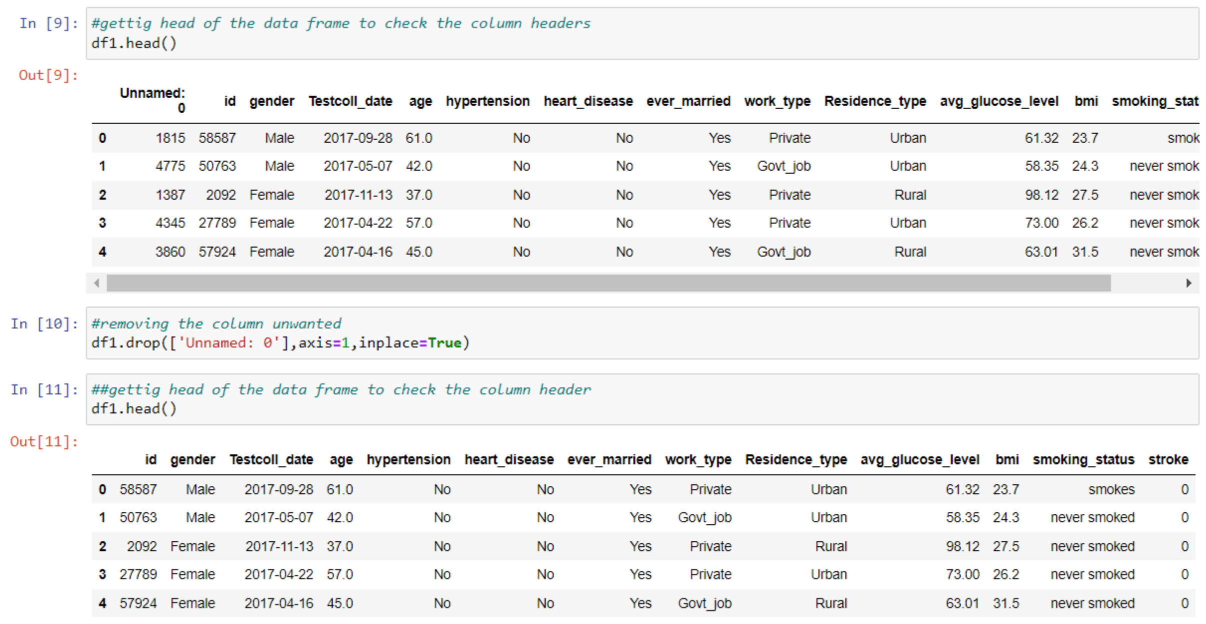
3. Data Cleaning/Processing :

Real time Data is not clean. We need to perform pre-processing and data cleaning before sending data to train or test the models. It helps us to identify the errors , missing data and helps to increase the quality of data. It is useful to get better accuracy from machine learning models.

We have performed 11 types of data cleaning and processing to improve data quality

Step 1: Removing Junk Columns

Data set has few columns without names and as no information is present for these columns. The models really can't work with this type of data and give result. As there is no need to store this data we have dropped unnamed columns.



In [9]:
```
#gettig head of the data frame to check the column headers
df1.head()
```

Out[9]:

	Unnamed: 0	id	gender	Testcoll_date	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_stat
0	1815	58587	Male	2017-09-28	61.0	No	No	Yes	Private	Urban	61.32	23.7	smok
1	4775	50763	Male	2017-05-07	42.0	No	No	Yes	Govt_job	Urban	58.35	24.3	never smok
2	1387	2092	Female	2017-11-13	37.0	No	No	Yes	Private	Rural	98.12	27.5	never smok
3	4345	27789	Female	2017-04-22	57.0	No	No	Yes	Private	Urban	73.00	26.2	never smok
4	3860	57924	Female	2017-04-16	45.0	No	No	Yes	Govt_job	Rural	63.01	31.5	never smok

In [10]:
```
#removing the column unwanted
df1.drop(['Unnamed: 0'],axis=1,inplace=True)
```

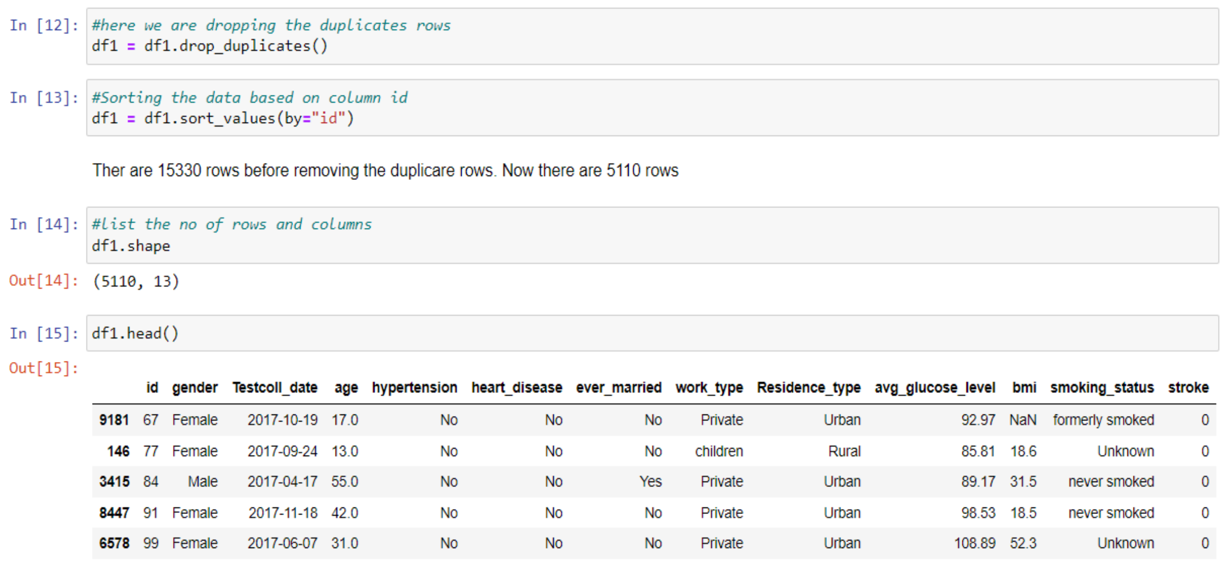
In [11]:
```
##gettig head of the data frame to check the column header
df1.head()
```

Out[11]:

	id	gender	Testcoll_date	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	58587	Male	2017-09-28	61.0	No	No	Yes	Private	Urban	61.32	23.7	smokes	0
1	50763	Male	2017-05-07	42.0	No	No	Yes	Govt_job	Urban	58.35	24.3	never smoked	0
2	2092	Female	2017-11-13	37.0	No	No	Yes	Private	Rural	98.12	27.5	never smoked	0
3	27789	Female	2017-04-22	57.0	No	No	Yes	Private	Urban	73.00	26.2	never smoked	0
4	57924	Female	2017-04-16	45.0	No	No	Yes	Govt_job	Rural	63.01	31.5	never smoked	0

Step 2 : Removing Duplicate values in the rows

The data has Duplicate rows i.e same information is entered multiple times . we can't train the models with the same data as there will be no learning for models with the same data . Always we need to provide new and different types of inputs so that models can learn a wide range of things and give better results.
We dropped rows and sorted the data using id column '



```
In [12]: #here we are dropping the duplicates rows
         df1 = df1.drop_duplicates()

In [13]: #Sorting the data based on column id
         df1 = df1.sort_values(by="id")
```

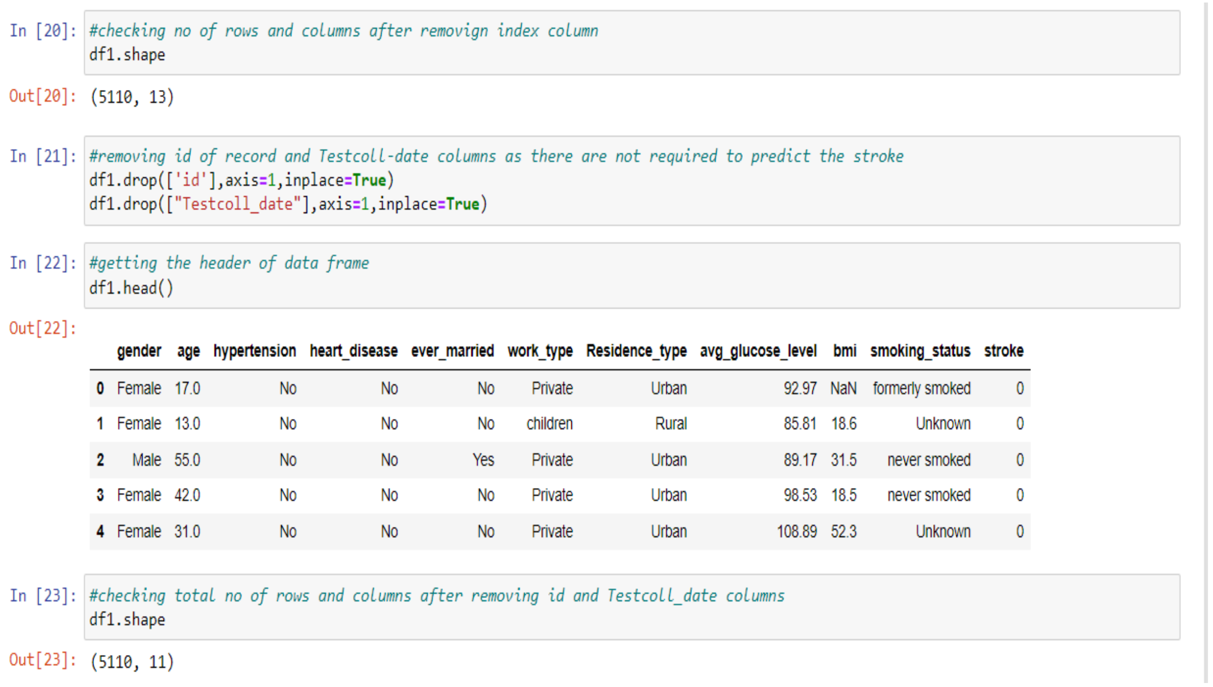
Ther are 15330 rows before removing the duplicare rows. Now there are 5110 rows

```
In [14]: #list the no of rows and columns
         df1.shape

Out[14]: (5110, 13)

In [15]: df1.head()

Out[15]:
```

	id	gender	Testcoll_date	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9181	67	Female	2017-10-19	17.0	No	No	No	Private	Urban	92.97	NaN	formerly smoked	0
146	77	Female	2017-09-24	13.0	No	No	No	children	Rural	85.81	18.6	Unknown	0
3415	84	Male	2017-04-17	55.0	No	No	Yes	Private	Urban	89.17	31.5	never smoked	0
8447	91	Female	2017-11-18	42.0	No	No	No	Private	Urban	98.53	18.5	never smoked	0
6578	99	Female	2017-06-07	31.0	No	No	No	Private	Urban	108.89	52.3	Unknown	0

Step 3 : Removing Unique Columns which has no Relation to Target

There might be extra data present which is not useful like serial no , mobile number etc. to predict the Stroke . These can be collected due to many reasons during the data collection phase but not needed to train and test the models .
In this data set we are having id and testcoll-date column which has no relation to predict the stroke , therefore we are dropping these 2 columns



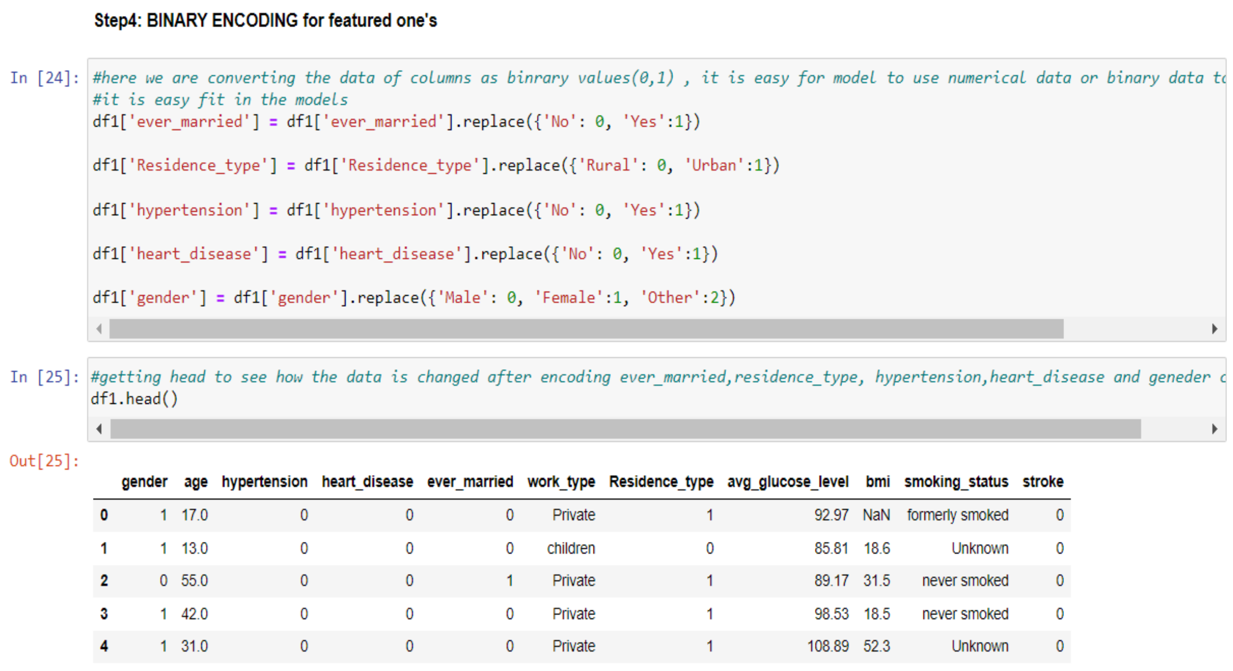
```
In [20]: #checking no of rows and columns after removign index column
         df1.shape

Out[20]: (5110, 13)

In [21]: #removing id of record and Testcoll-date columns as there are not required to predict the stroke
         df1.drop(['id'],axis=1,inplace=True)
         df1.drop(["Testcoll_date"],axis=1,inplace=True)

In [22]: #getting the header of data frame
         df1.head()

Out[22]:
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Female	17.0	No	No	No	Private	Urban	92.97	NaN	formerly smoked	0
1	Female	13.0	No	No	No	children	Rural	85.81	18.6	Unknown	0
2	Male	55.0	No	No	Yes	Private	Urban	89.17	31.5	never smoked	0
3	Female	42.0	No	No	No	Private	Urban	98.53	18.5	never smoked	0
4	Female	31.0	No	No	No	Private	Urban	108.89	52.3	Unknown	0

```
In [23]: #checking total no of rows and columns after removing id and Testcoll_date columns
         df1.shape

Out[23]: (5110, 11)
```

Step 4: BINARY ENCODING for featured one's

Binary Encoding is nothing but converting categorical data to binary digits (0,1,2 etc..). Generally 1st we convert into numerical data and later to binary digits.
After converting data to binary , it is easy to fit in machine learning models
In this data we are converting ever_married, Residence_type , hyperion, heart_disease, gender column values to binary digits(0 and 1)
For ever_married - No is replaced with 0 and Yes with 1
For Residence_type - Rural is replaced with 0 and Urban with 1
For hypertension - No is replaced with 0 and Yes with 1
For heart_disease - No is replaced with 0 and Yes with 1
For gender - Male is replaced with 0 , Female with 1 and Other 2



Step4: BINARY ENCODING for featured one's

```
In [24]: #here we are converting the data of columns as binrary values(0,1) , it is easy for model to use numerical data or binary data tc
         #it is easy fit in the models
         df1['ever_married'] = df1['ever_married'].replace({'No': 0, 'Yes':1})

         df1['Residence_type'] = df1['Residence_type'].replace({'Rural': 0, 'Urban':1})

         df1['hypertension'] = df1['hypertension'].replace({'No': 0, 'Yes':1})

         df1['heart_disease'] = df1['heart_disease'].replace({'No': 0, 'Yes':1})

         df1['gender'] = df1['gender'].replace({'Male': 0, 'Female':1, 'Other':2})
```

```
In [25]: #getting head to see how the data is changed after encoding ever_married,residence_type, hypertension,heart_disease and geneder c
         df1.head()
```

Out[25]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	17.0	0	0	0	Private	1	92.97	NaN	formerly smoked	0
1	1	13.0	0	0	0	children	0	85.81	18.6	Unknown	0
2	0	55.0	0	0	1	Private	1	89.17	31.5	never smoked	0
3	1	42.0	0	0	0	Private	1	98.53	18.5	never smoked	0
4	1	31.0	0	0	0	Private	1	108.89	52.3	Unknown	0

Step 5: One Hot Encoding

Here we are hot encoding the work_type and smoking_status. It created new columns for every possible category present in the column and assigned 0 and 1 based on its presence. For example if a person does a gov job the value is 1 and if not 0 is assigned.
It is easy to rescale the data after hot encoding . Hot encoding is useful to increase the prediction probability.
In this data we are converting work_type and smoking_status to binary using hot encoding.

Step5: One Hot Encoding

```
In [26]: #here we are hot encoding the work_type and smoking_status. it created new columns for every possbile category present in column
         #and assign 0 and 1 based on its presence for example if person do gov job the value is 1 and if not 0 is assigned
         for column in ['work_type','smoking_status']:
             dummies = pd.get_dummies(df1[column],prefix=column)
             df1 = pd.concat([df1, dummies], axis =1)
             df1 = df1.drop(column, axis =1)
```

```
In [27]: #checking how the encoding divided work_type and smoking_status columns
         df1.head()
```

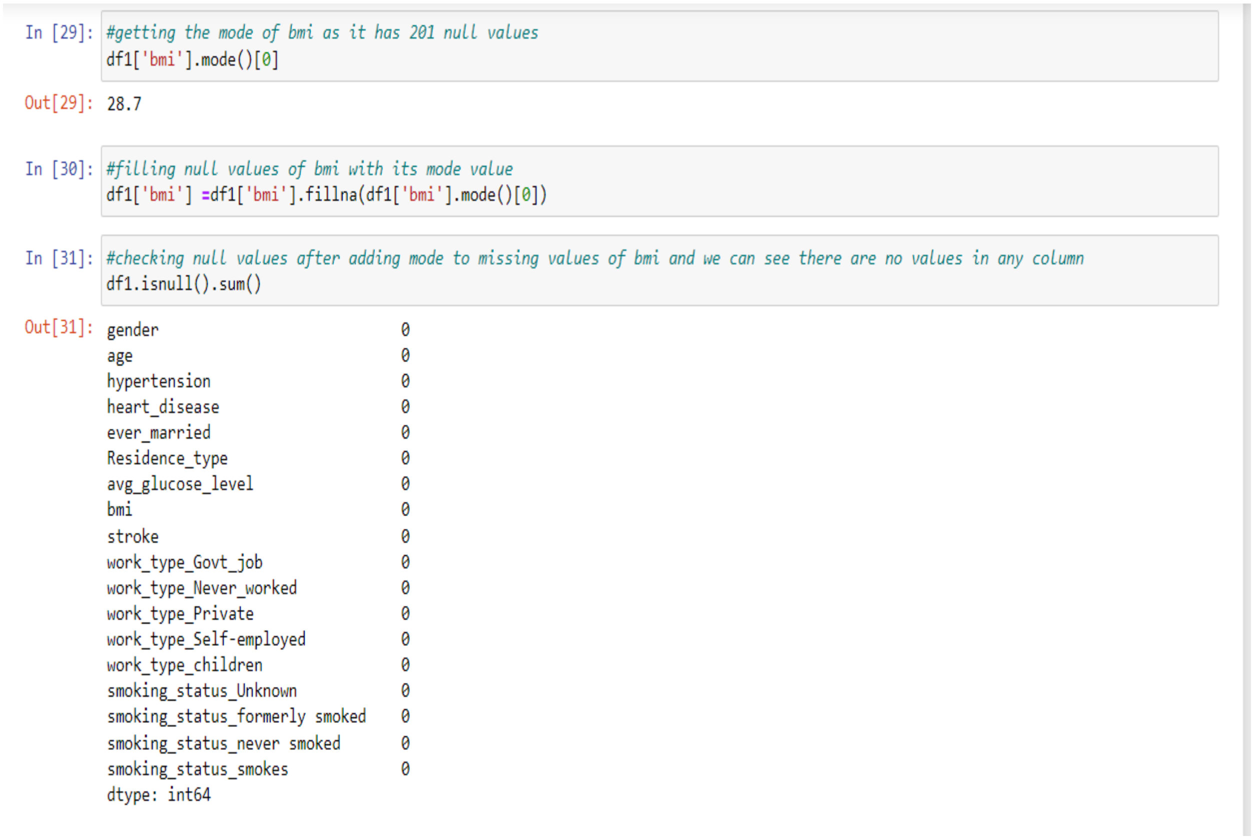
Out[27]:

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_job	work_type_Never_worked	wc
0	1	17.0	0	0	0	1	92.97	NaN	0	0	0	
1	1	13.0	0	0	0	0	85.81	18.6	0	0	0	
2	0	55.0	0	0	1	1	89.17	31.5	0	0	0	
3	1	42.0	0	0	0	1	98.53	18.5	0	0	0	
4	1	31.0	0	0	0	1	108.89	52.3	0	0	0	

Step 6 : checking for any missing Values and fill with Mode imputation

Real time data sets are impure. Some data might be missing due to various reasons like data is not available during collection time, or lost etc.. we need to be careful while cleaning the data and training the models because if we train models without missing values , then if the column which data is missed is important to predict the model gives wrong outputs
In this data set we have identified bmi columns as missing values , so filled with mode of the bmi column . as mean is nothing the average of all data present , it is best to fill with mean for missing values

```
In [28]: #Checking no of null values present in each row
         df1.isnull().sum()

Out[28]: gender                              0
         age                                 0
         hypertension                        0
         heart_disease                       0
         ever_married                        0
         Residence_type                      0
         avg_glucose_level                   0
         bmi                               201
         stroke                              0
         work_type_Govt_job                  0
         work_type_Never_worked              0
         work_type_Private                   0
         work_type_Self-employed             0
         work_type_children                  0
         smoking_status_Unknown              0
         smoking_status_formerly smoked      0
         smoking_status_never smoked         0
         smoking_status_smokes               0
         dtype: int64
```



```
In [29]: #getting the mode of bmi as it has 201 null values
         df1['bmi'].mode()[0]

Out[29]: 28.7

In [30]: #filling null values of bmi with its mode value
         df1['bmi'] =df1['bmi'].fillna(df1['bmi'].mode()[0])

In [31]: #checking null values after adding mode to missing values of bmi and we can see there are no values in any column
         df1.isnull().sum()

Out[31]: gender                            0
         age                               0
         hypertension                      0
         heart_disease                     0
         ever_married                      0
         Residence_type                    0
         avg_glucose_level                 0
         bmi                               0
         stroke                            0
         work_type_Govt_job                0
         work_type_Never_worked            0
         work_type_Private                 0
         work_type_Self-employed           0
         work_type_children                0
         smoking_status_Unknown            0
         smoking_status_formerly smoked    0
         smoking_status_never smoked       0
         smoking_status_smokes             0
         dtype: int64
```

Step 7: Adding New features and categorizing Age

Some diseases occur in certain age groups . It's better we categorize age into groups rather than having a range. We have divided age into 7 Groups as follows
Age between 0 to 2.5 categorized as infant
Age between 2.5 and 4 categorized as toddler
Age between 4 and 12 categorized as child
Age between 12 and 19 categorized as Teen
Age between 19 and 30 categorized as Youth
Age between 30 and 50 categorized as Middle Age
Age between 30 and 99 categorized as Old Age

```
def age_category(val):
    if val >= 0 and val < 2.5:
        return "infant"
    elif val >= 2.5 and val < 4.0:
        return "toddler"
    elif val >=4.0 and val < 12.0:
        return "child"
    elif val >=12.0 and val <19.0:
        return "Teen"
    elif val >=19.0 and val <30.0:
        return "Youth"
    elif val >=30.0 and val <50.0:
        return "Middle Age"
    elif val >=50.0 and val <99.0:
        return "Old Age"
```

```
In [33]: #calling age_category by passing age of the person and assigning the category to each person
         df1['age_category'] = df1['age']
         df1['age_category'] = df1['age_category'].apply(age_category)
```

```
In [34]: df1.head()
```

Out[34]:

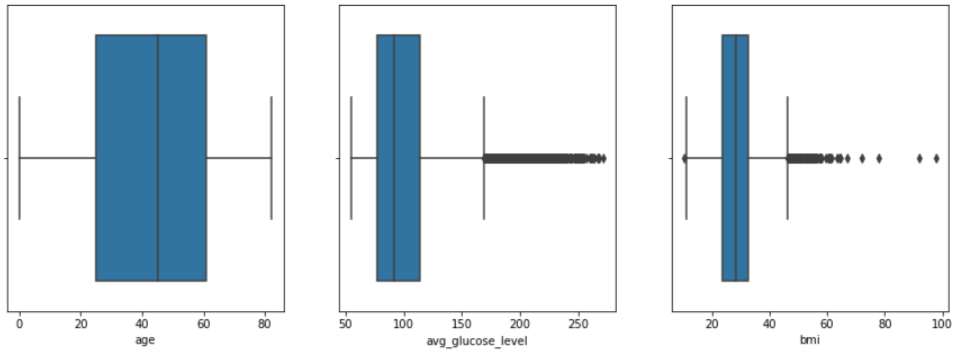
	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_job	work_type_Never_worked	wo
0	1	17.0	0	0	0	1	92.97	28.7	0	0	0	
1	1	13.0	0	0	0	0	85.81	18.6	0	0	0	
2	0	55.0	0	0	1	1	89.17	31.5	0	0	0	
3	1	42.0	0	0	0	1	98.53	18.5	0	0	0	
4	1	31.0	0	0	0	1	108.89	52.3	0	0	0	

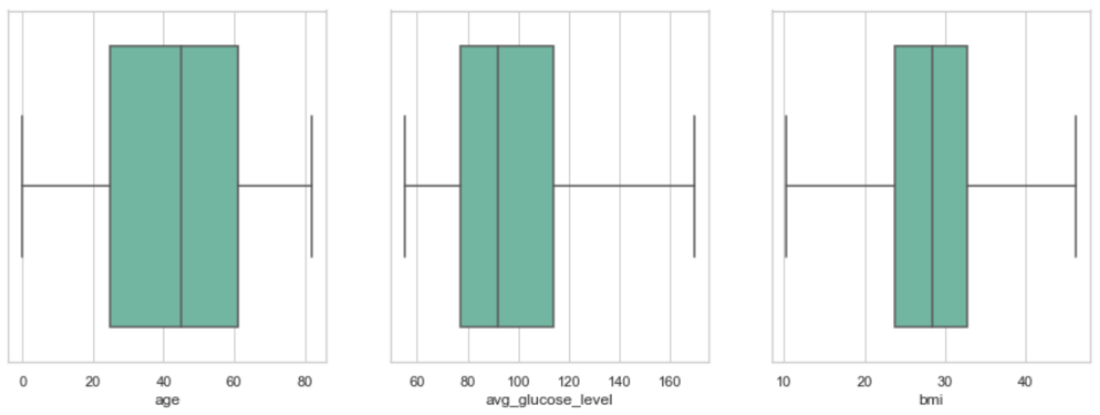
Step8: Detection of Outlier and Removal

Outliers are nothing but the points which are far or extremely small or big in the group of data. Outliers can decrease the mean and increase the variance resulting in decrease in model accuracy.Thus we should always remove the outliers from data

In this data we have identified by plotting boxplot avg_glucose_level', 'bmi have few outliers. Later we removed the outliers from data.

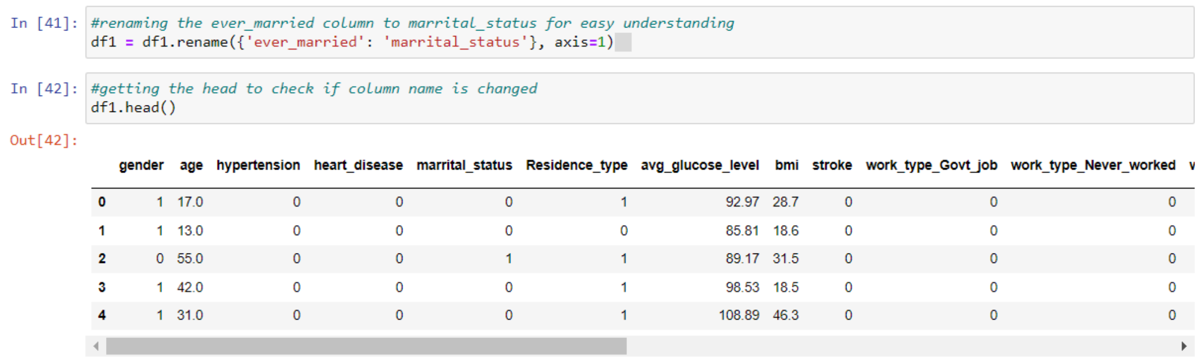
```
In [35]: #we are plotting boxplot for age , avg_glucose_level and bmi and finding the outlier in these columns
         plt.figure(figsize =(15,5))
         j=1
         for i in ['age', 'avg_glucose_level', 'bmi']:
             plt.subplot(1,3,j)
             sns.boxplot(data = df1 ,x=i)
             j=j+1
```





Step9: Renaming Column Names for easy understanding

Sometimes the column names might not be clear to understand , we need to keep end user in mind and rename the column names so that the user can understand easily
We have renamed the ever_married column name to 'marrital_status

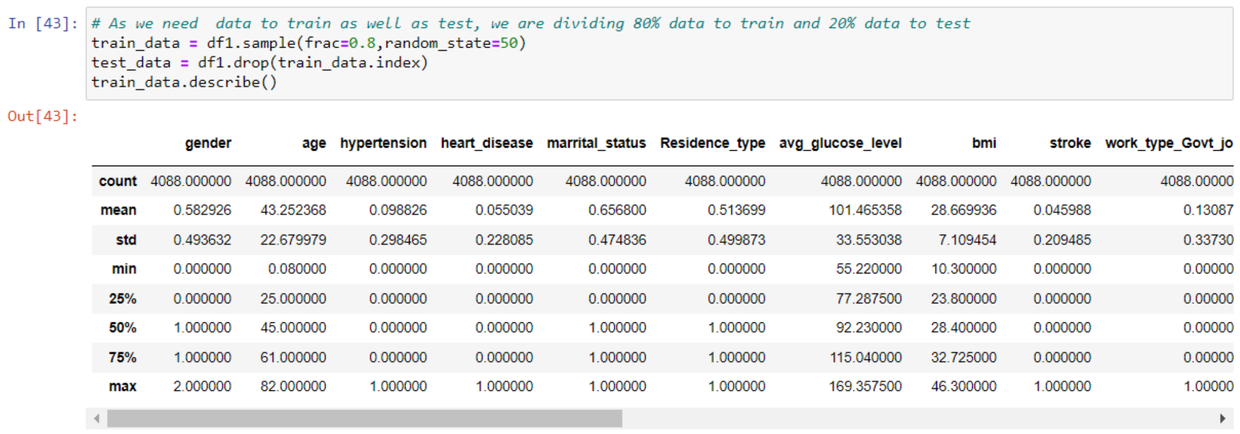


```
In [41]: #renaming the ever_married column to marrital_status for easy understanding
         df1 = df1.rename({'ever_married': 'marrital_status'}, axis=1)

In [42]: #getting the head to check if column name is changed
         df1.head()
Out[42]:
```

	gender	age	hypertension	heart_disease	marrital_status	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_job	work_type_Never_worked	v
0	1	17.0	0	0	0	1	92.97	28.7	0	0	0	
1	1	13.0	0	0	0	0	85.81	18.6	0	0	0	
2	0	55.0	0	0	1	1	89.17	31.5	0	0	0	
3	1	42.0	0	0	0	1	98.53	18.5	0	0	0	
4	1	31.0	0	0	0	1	108.89	46.3	0	0	0	

Step 10 : Splitting Data into 80% Training ang 20% Testing

We need data to train and to test. So we have divided 80% of data for training and the rest 20% of data for testing the model accuracy. We have divide the data randomly



```
In [43]: # As we need  data to train as well as test, we are dividing 80% data to train and 20% data to test
         train_data = df1.sample(frac=0.8,random_state=50)
         test_data = df1.drop(train_data.index)
         train_data.describe()
Out[43]:
```

	gender	age	hypertension	heart_disease	marrital_status	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_jo
count	4088.000000	4088.000000	4088.000000	4088.000000	4088.000000	4088.000000	4088.000000	4088.000000	4088.000000	4088.00000
mean	0.582926	43.252368	0.098826	0.055039	0.656800	0.513699	101.465358	28.669936	0.045988	0.13087
std	0.493632	22.679979	0.298465	0.228085	0.474836	0.499873	33.553038	7.109454	0.209485	0.33730
min	0.000000	0.080000	0.000000	0.000000	0.000000	0.000000	55.220000	10.300000	0.000000	0.00000
25%	0.000000	25.000000	0.000000	0.000000	0.000000	0.000000	77.287500	23.800000	0.000000	0.00000
50%	1.000000	45.000000	0.000000	0.000000	1.000000	1.000000	92.230000	28.400000	0.000000	0.00000
75%	1.000000	61.000000	0.000000	0.000000	1.000000	1.000000	115.040000	32.725000	0.000000	0.00000
max	2.000000	82.000000	1.000000	1.000000	1.000000	1.000000	169.357500	46.300000	1.000000	1.00000

we have 4088 rows to train and 1022 rows to test the model

In [45]:
```python
#showing test data
test_data.describe()
```

Out[45]:

	gender	age	hypertension	heart_disease	marrital_status	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_jo
count	1022.000000	1022.000000	1022.000000	1022.000000	1022.000000	1022.000000	1022.000000	1022.000000	1022.000000	1022.00000
mean	0.599804	43.123601	0.091977	0.049902	0.653620	0.485323	99.119587	28.890313	0.059687	0.11937
std	0.490178	22.351996	0.289134	0.217849	0.476049	0.500029	31.772256	7.162096	0.237022	0.32438
min	0.000000	0.080000	0.000000	0.000000	0.000000	0.000000	55.120000	14.100000	0.000000	0.00000
25%	0.000000	25.000000	0.000000	0.000000	0.000000	0.000000	76.830000	23.800000	0.000000	0.00000
50%	1.000000	45.000000	0.000000	0.000000	1.000000	0.000000	90.660000	28.400000	0.000000	0.00000
75%	1.000000	59.000000	0.000000	0.000000	1.000000	1.000000	111.005000	33.075000	0.000000	0.00000
max	1.000000	82.000000	1.000000	1.000000	1.000000	1.000000	169.357500	46.300000	1.000000	1.00000

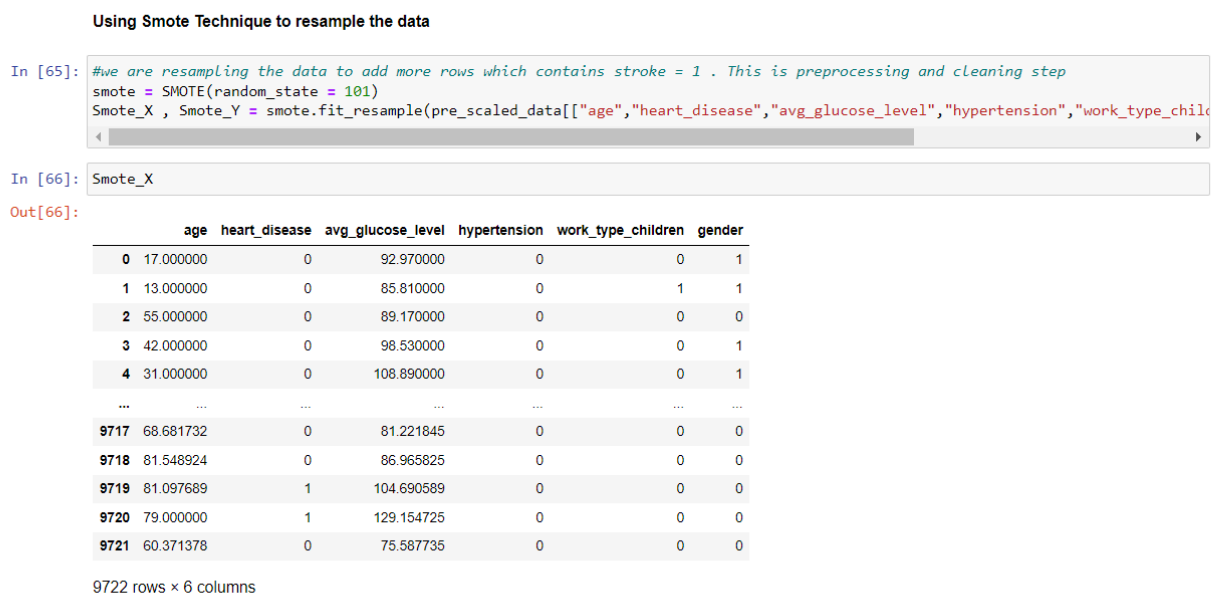
Step 11 : Using Smote Technique to resample the data

We have data where the cases with no stroke are 95 % and only 5% rows are with stroke. So we have resampled using a smote technique to add more rows so that we can get more rows with stroke. Before doing this we have ~5000 rows , later we have ~9700 rows having more rows with stroke as result.

In [64]:
```python
#here we are seeing the most the data is where stroke = 0 or no stroke
features = ["age","heart_disease","avg_glucose_level","hypertension","work_type_children","gender","stroke"]
pre_scaled_data =df1[features]
pre_scaled_data.head(10)
```

Out[64]:

	age	heart_disease	avg_glucose_level	hypertension	work_type_children	gender	stroke
0	17.0	0	92.97	0	0	1	0
1	13.0	0	85.81	0	1	1	0
2	55.0	0	89.17	0	0	0	0
3	42.0	0	98.53	0	0	1	0
4	31.0	0	108.89	0	0	1	0
5	38.0	0	91.44	0	0	1	0
6	24.0	0	97.55	0	0	1	0
7	80.0	0	84.86	0	0	1	0
8	33.0	0	86.97	0	0	1	0
9	20.0	0	94.67	0	0	1	0



Using Smote Technique to resample the data

```
In [65]: #we are resampling the data to add more rows which contains stroke = 1 . This is preprocessing and cleaning step
         smote = SMOTE(random_state = 101)
         Smote_X , Smote_Y = smote.fit_resample(pre_scaled_data[["age","heart_disease","avg_glucose_level","hypertension","work_type_chil
```

```
In [66]: Smote_X
Out[66]:
```

	age	heart_disease	avg_glucose_level	hypertension	work_type_children	gender
0	17.000000	0	92.970000	0	0	1
1	13.000000	0	85.810000	0	1	1
2	55.000000	0	89.170000	0	0	0
3	42.000000	0	98.530000	0	0	1
4	31.000000	0	108.890000	0	0	1
...	...	...	...	...	...	...
9717	68.681732	0	81.221845	0	0	0
9718	81.548924	0	86.965825	0	0	0
9719	81.097689	1	104.690589	0	0	0
9720	79.000000	1	129.154725	0	0	0
9721	60.371378	0	75.587735	0	0	0

9722 rows × 6 columns

4. Exploratory Data Analysis (EDA):

Exploratory data analysis (EDA) is used to analyze data, and visualize to explore more on datasets. Below is the list of EDA processes used with respect to our dataset.

1. Getting the Profile Report for the entire Dataset

For the dataset, it allows report generation with a variety of features and customizations.

Let's examine each section of the created report individually now that it has been generated.

Overview

The Overview shows the overall statistics of all the data. It contains a number of variables, Number of observations, Missing cells, missing cells in terms of percentage, Total size in memory, and average record size in memory.

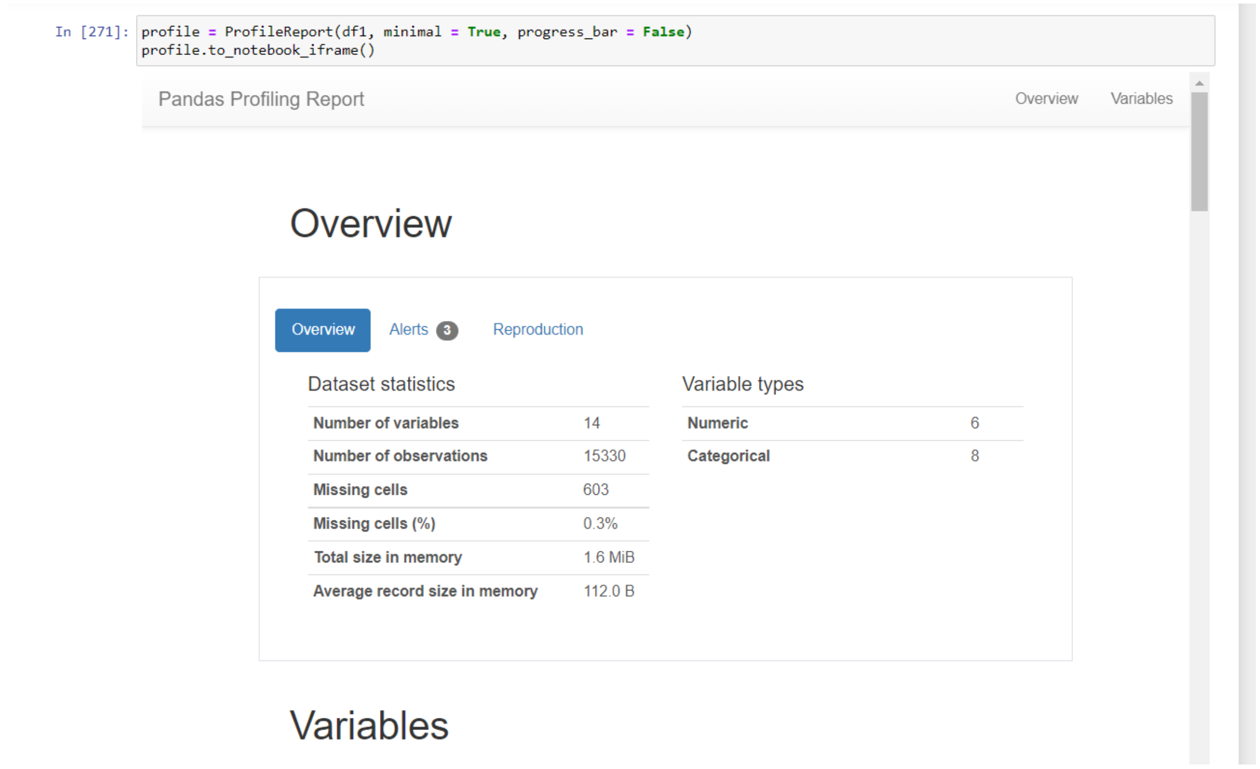
Alerts contains all the warnings

Reproduction

Information on the generation of reports is simply displayed on the reproduction tab. It displays the analysis's beginning and end times, the length of time it took to produce the report, the Pandas profiling software version, and a download option for the configuration.

The variables section gives all the information regarding the columns . It contains information such as No of distinct values , missing values , Minimum , Mean and Maximum values of the Column

Profile reports can be considered as a summary of a data set. We are checking this before pre-processing or cleaning and after cleaning. Before cleaning we can see that they are missing values and the difference between mean , min and max values . Later after cleaning from profiling we can check that there are no missing values



```
In [271]: profile = ProfileReport(df1, minimal = True, progress_bar = False)
          profile.to_notebook_iframe()
```

Dataset statistics

Number of variables	14
Number of observations	15330
Missing cells	603
Missing cells (%)	0.3%
Total size in memory	1.6 MiB
Average record size in memory	112.0 B

Variable types

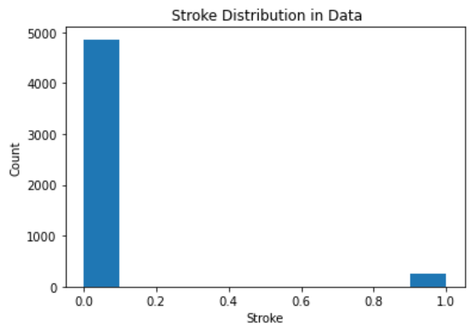
Numeric	6
Categorical	8

2. Histogram for Stroke Column

A graphic representation called a histogram is frequently used to show how numerical data are distributed.

In the below Histogram, we can check the number of people who got a stroke vs people with no stroke.

```
In [47]: #Plotting Histogram to check no of people got stroke vs no Stroke
         plt.hist(df1['stroke'])
         plt.title(" Stroke Distribution in Data")
         plt.ylabel("Count")
         plt.xlabel("Stroke")

Out[47]: Text(0.5, 0, 'Stroke')
```

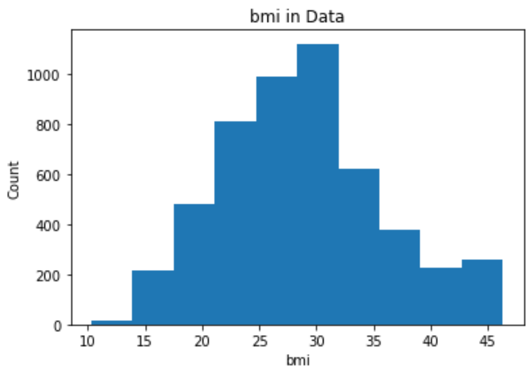


From this histogream we can understand we have 95% data with no stroke and 5% with stroke

From this, we can say that the majority of data is supporting no stroke, so here we can’t really say who gets the stroke but we need to say the probability of getting the stroke.

From the below histogram, we examine the count of people with respect to their values. We can say that, count of the section of people and their bmi values. We can see that, nearly 100 people have a BMI value between 25 to 35.

```
In [48]: #Plotting Histogram to check distribution of BMI
         plt.hist(df1['bmi'])
         plt.title("bmi in Data")
         plt.ylabel("Count")
         plt.xlabel("bmi")

Out[48]: Text(0.5, 0, 'bmi')
```

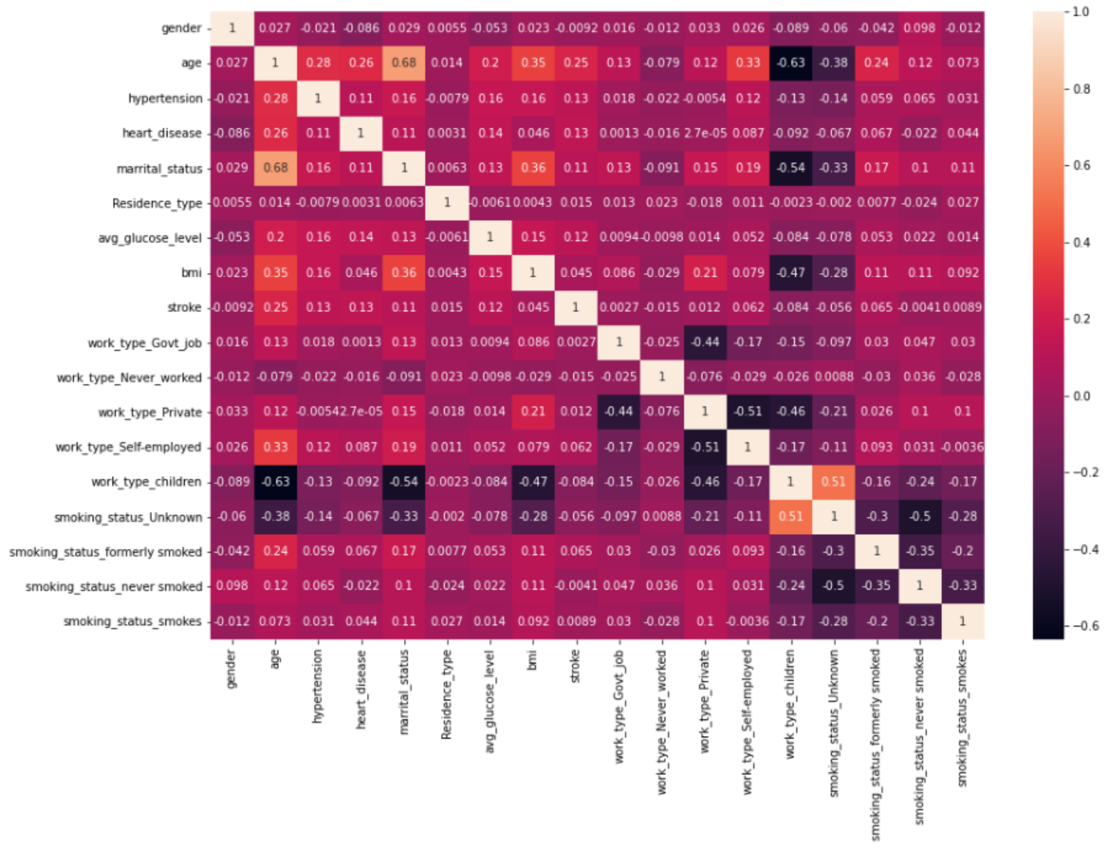


3. Heat Map for Cleaned Data

A heatmap is a 2-dimensional image in the style of a matrix that displays numerical data as cells. The heatmap's cells are all colored, and the variations in color show how the value and data frame are related in some way.

We are using a heat map to plot the correlation between each variable with every variable. This correlation value states the relation between two variables. From the below heat map, we can find the correlation values of pair of variables.

```
In [49]: #plotting the correlation matrix/heat map
         plt.figure(figsize = (15,10))
         sns.heatmap(df1.corr(),annot = True)

Out[49]: <AxesSubplot:>
```

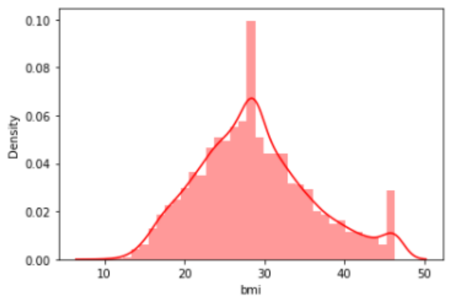


```
In [112]: df1.corr()['stroke'].sort_values(ascending=False)[1:]

Out[112]: age                               0.245257
          heart_disease                     0.134914
          avg_glucose_level                 0.131945
          hypertension                      0.127904
          ever_married                      0.108340
          smoking_status_formerly smoked    0.064556
          work_type_Self-employed           0.062168
          bmi                               0.038257
          Residence_type                    0.015458
          work_type_Private                 0.011888
          smoking_status_smokes             0.008939
          work_type_Govt_job                0.002677
          smoking_status_never smoked      -0.004129
          gender                           -0.009200
          work_type_Never_worked           -0.014882
          smoking_status_Unknown           -0.055892
          work_type_children               -0.083869
          Name: stroke, dtype: float64
```

4. Displot

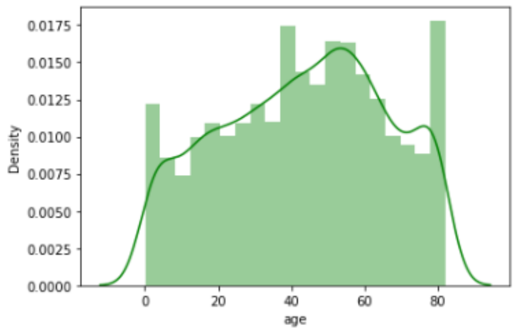
The distplot displays a variable's data distribution against a density distribution, which is known as a univariate distribution of data. From the below distplot, we can examine based on bmi. We can see that, between the bmi of 20 and 40, we could see that density is high. We can find how bmi is distributed in our data.

```
In [52]: #plotting displot to see distribution of bmi
         sns.distplot(df1.bmi, color="red")

Out[52]: <AxesSubplot:xlabel='bmi', ylabel='Density'>
```



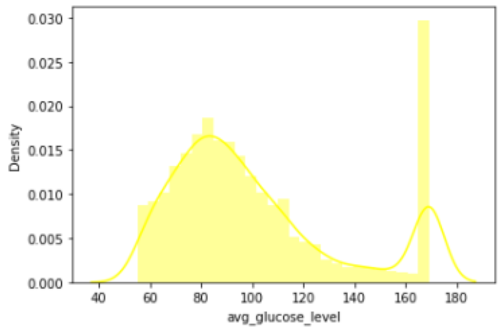
Similarly, when we use age, we see the density of ages in our data. We can see that data contains people of age groups from 0 to 80.

```
In [53]: # To find out how the age distribution in our dataset is present.
         sns.distplot(df1.age, color = "green")

Out[53]: <AxesSubplot:xlabel='age', ylabel='Density'>
```



Similarly, we can use the average glucose value, we could the density of the respective values. There is a peak in density between 160 and 180. We can state that our data contains avg_glucose_level between 160 and 180.

```
In [54]: ## To find out how the glucose_level distribution in our dataset is present.
         sns.distplot(df1.avg_glucose_level, color = "yellow")

Out[54]: <AxesSubplot:xlabel='avg_glucose_level', ylabel='Density'>
```

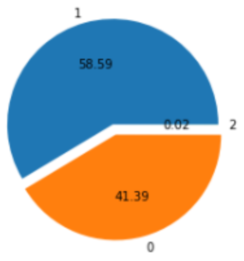


5.Pie Chart

The pie chart is used for the segregation of data. When using a pie chart for the segregation of gender from

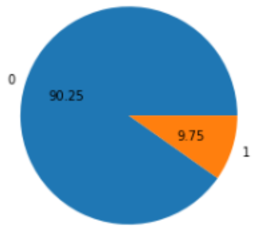
The gender segregation in our data is visualized using a pie chart. From the below pie chart, demonstrates that there are more women than men and other groups, with 58.59% of the population female, 41.39% male, and 0.02% other.

```
In [55]: #ploting a piechart to understand the gender segregration in our Data
         plt.pie(df1['gender'].value_counts(), labels = df1['gender'].value_counts().index,autopct ='%.2f',explode =[0,0.1,0])

Out[55]: ([<matplotlib.patches.Wedge at 0x17dcec74df0>,
           <matplotlib.patches.Wedge at 0x17dceaf8430>,
           <matplotlib.patches.Wedge at 0x17dceaf8100>],
          [Text(-0.2932923170393164, 1.0601790493901062, '1'),
           Text(0.3192440878104747, -1.1567554678487835, '0'),
           Text(1.099999792043413, -0.0006763907511957427, '2')],
          [Text(-0.15997762747599073, 0.5782794814855124, '58.59'),
           Text(0.18622571788944353, -0.6747740229117903, '41.39'),
           Text(0.5999998865691343, -0.00036894040974313233, '0.02')])
```



Similarly, we are also using a pie chart for the segregation of hypertension. We could see that 9.75% of people have hypertension. 90.25% of people who don't smoke.

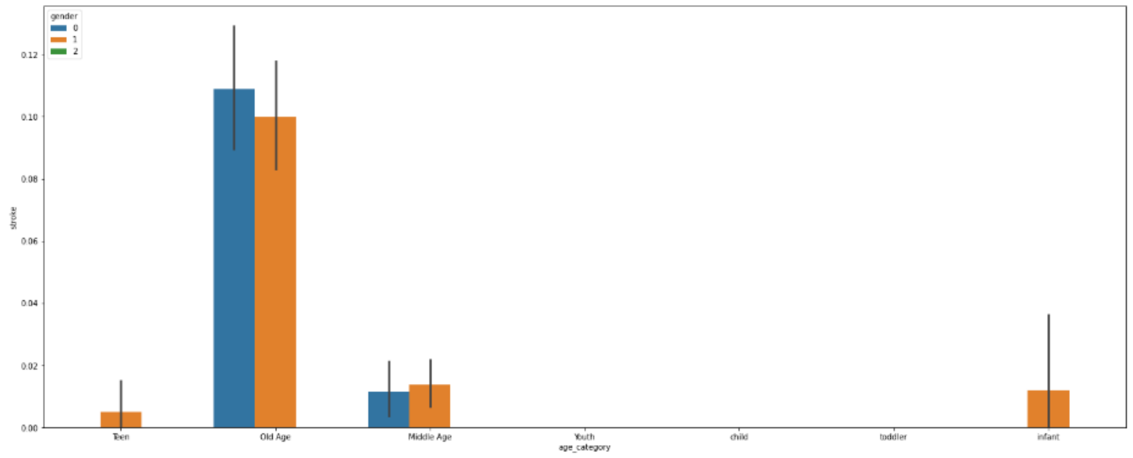
```
In [56]: #ploting a piechart to understand the hypertension in our Data
         plt.pie(df1['hypertension'].value_counts(), labels = df1['hypertension'].value_counts().index,autopct ='%.2f')

Out[56]: ([<matplotlib.patches.Wedge at 0x17dceb88d00>,
           <matplotlib.patches.Wedge at 0x17dcec3da30>],
          [Text(-1.0488454518125865, 0.3315467059285184, '0'),
           Text(1.0488454440521755, -0.3315467304785183, '1')],
          [Text(-0.5720975191705017, 0.18084365777919187, '90.25'),
           Text(0.5720975149375501, -0.1808436711701009, '9.75')])
```



6. Bar Plot

Barplot is used for the representation of data. Here we are using this barplot to examine the relation between age and stroke. This will provide us with the details of the possibility of stroke with respect to age.
In the below barplot, we are examining age and stroke. We can say from the barplot that, people who are of old age have a possibility of stroke when compared to other ages.

```
In [57]: #Here we plotted a bar plot to use relation between age and storke. This tells us the stroke is likely to occured to old age
         # and less in child and toddler
         plt.figure(figsize = (25,10))
         sns.barplot('age_category','stroke',hue='gender',data=df1)

Out[57]: <AxesSubplot:xlabel='age_category', ylabel='stroke'>
```

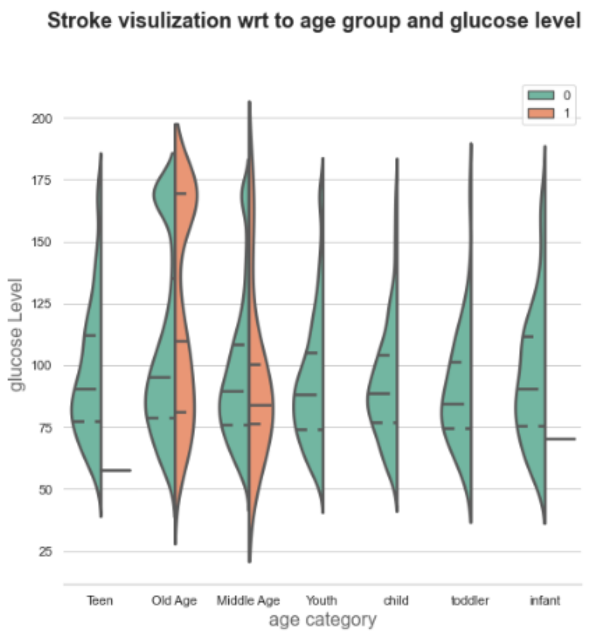


7. Violin plot

In order to allow for comparison, it displays the distribution of quantitative data across a number of levels of one (or more) categorical variables.

We are using the Violin plot for Stroke visualization with respect to age group and glucose level. From the below violin plot, we can see that the people who are of middle age and old age have glucose value which shows the possibility of stroke. Based on this plot, we can examine that old age and middle age people with high glucose value have a chance of stroke.

```
In [58]: #Here we are using violin plot between age , glucose_level and stroke to see the probability density

         sns.set(style="whitegrid",palette = "Set2", color_codes=True)

         f, ax = plt.subplots(figsize=(8, 8))

         sns.violinplot(x="age_category", y="avg_glucose_level", hue="stroke", data=df1, split=True, linewidth=2.5,
                        inner="quart")
         sns.despine(left=True)

         f.suptitle('Stroke visulization wrt to age group and glucose level', fontsize=18, fontweight='bold')
         ax.set_xlabel("age category",size = 16,alpha=0.7)
         ax.set_ylabel("glucose Level ",size = 16,alpha=0.7)
         plt.legend(loc='upper right')

Out[58]: <matplotlib.legend.Legend at 0x17dcec4edf0>
```

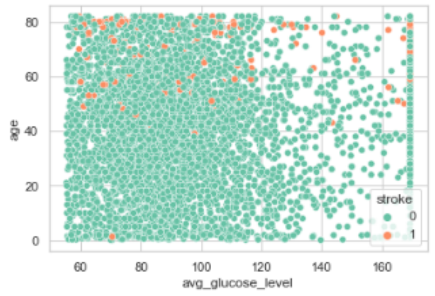


8. Scatterplot

The association between variables is displayed using a data visualization technique called a scatter plot. From the below scatterplot, we can examine the association between the average glucose value and age of a person and the possibility of a stroke.

```
In [59]: #we are doing a scatterplot between age , glucose level and stroke . to check the stroke occurance
         sns.scatterplot(data = df1, x ='avg_glucose_level', y = 'age', hue = 'stroke')
```

```
Out[59]: <AxesSubplot:xlabel='avg_glucose_level', ylabel='age'>
```



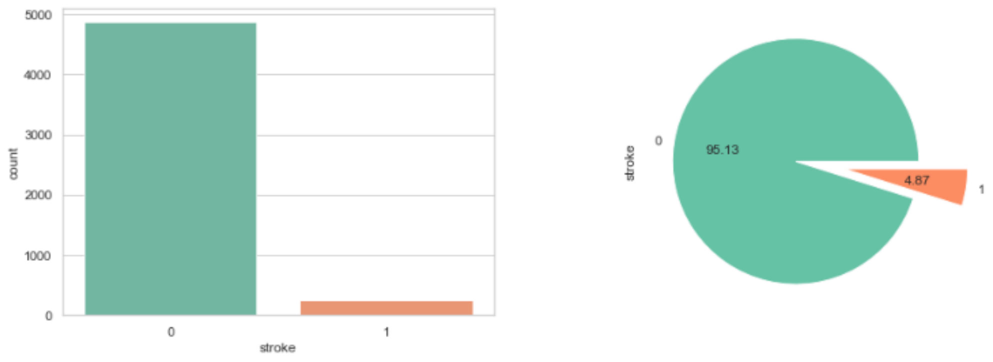
> From this figure we can understand that Stroke chance are very less populated

9.Count-plot

Count plot is used for showing the count of the number of people categorically using bars
This is used for showcasing the possibility of stroke for the pope in our dataset. From the below count plot, we can say that the possibility of a stroke is less. Added pie chart as well for the same for getting a percentage of stroke from our datasets.

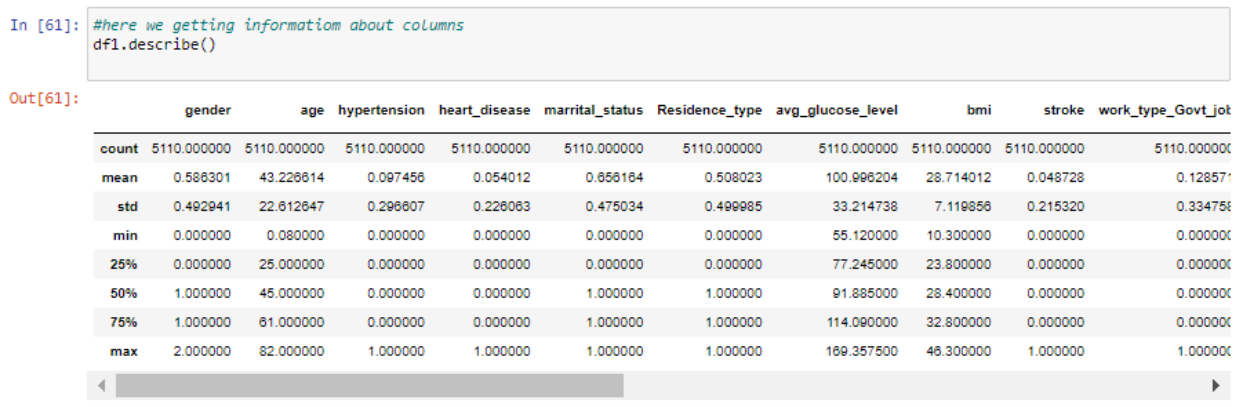
```
In [60]: #here are checking the count of rows with stroke and rows with no stroke .from below we an say 95% is without stroke and only 5%
         plt.figure(figsize = (15,5))
         plt.subplot(121)
         sns.countplot(data = df1, x ='stroke')
         plt.subplot(122)
         df1['stroke'].value_counts().plot(kind = 'pie', autopct ='%.2f',explode =[0,0.4])
```

```
Out[60]: <AxesSubplot:ylabel='stroke'>
```



10. Getting Description about Columns

This step is used to describe the overall of each column like count, mean, std, etc. Here we get information about each column.



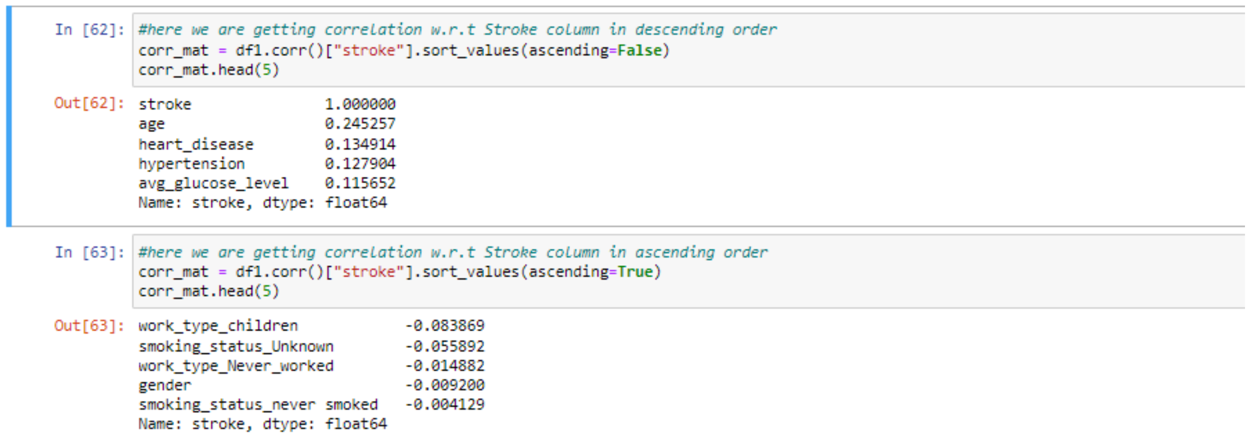
```
In [61]: #here we getting informatiom about columns
         df1.describe()
```

Out[61]:

	gender	age	hypertension	heart_disease	marrital_status	Residence_type	avg_glucose_level	bmi	stroke	work_type_Govt_jo
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.00000
mean	0.586301	43.226614	0.097456	0.054012	0.656164	0.508023	105.996204	28.714012	0.048728	0.12857
std	0.492941	22.612647	0.296607	0.226063	0.475034	0.499985	33.214738	7.119856	0.215320	0.33475
min	0.000000	0.080000	0.000000	0.000000	0.000000	0.000000	55.120000	10.300000	0.000000	0.00000
25%	0.000000	25.000000	0.000000	0.000000	0.000000	0.000000	77.245000	23.800000	0.000000	0.00000
50%	1.000000	45.000000	0.000000	0.000000	1.000000	1.000000	91.885000	28.400000	0.000000	0.00000
75%	1.000000	61.000000	0.000000	0.000000	1.000000	1.000000	114.090000	32.800000	0.000000	0.00000
max	2.000000	82.000000	1.000000	1.000000	1.000000	1.000000	169.357500	46.300000	1.000000	1.00000

11.Feature selection

In this step, we are getting correlation w.r.t Stroke column in descending order and ascending order. Below code shows the correlation with respect to stroke.



```
In [62]: #here we are getting correlation w.r.t Stroke column in descending order
         corr_mat = df1.corr()["stroke"].sort_values(ascending=False)
         corr_mat.head(5)

Out[62]: stroke               1.000000
         age                  0.245257
         heart_disease        0.134914
         hypertension         0.127904
         avg_glucose_level    0.115652
         Name: stroke, dtype: float64

In [63]: #here we are getting correlation w.r.t Stroke column in ascending order
         corr_mat = df1.corr()["stroke"].sort_values(ascending=True)
         corr_mat.head(5)

Out[63]: work_type_children            -0.083869
         smoking_status_Unknown        -0.055892
         work_type_Never_worked        -0.014882
         gender                        -0.009200
         smoking_status_never smoked   -0.004129
         Name: stroke, dtype: float64
```

With the help of a smote technique, we are resampling the data in order the balance them. After using a smote technique, from the below scatterplot, we can see the increase in stroke using resampled data.

Using Smote Technique to resample the data

In [65]:
```
#we are resampling the data to add more rows which contains stroke = 1 . This is preprocessing and cleaning step
smote = SMOTE(random_state = 101)
Smote_X , Smote_Y = smote.fit_resample(pre_scaled_data[["age","heart_disease","avg_glucose_level","hypertension","work_type_chil
```

In [66]:
```
Smote_X
```

Out[66]:

	age	heart_disease	avg_glucose_level	hypertension	work_type_children	gender
0	17.000000	0	92.970000	0	0	1
1	13.000000	0	85.810000	0	1	1
2	55.000000	0	89.170000	0	0	0
3	42.000000	0	98.530000	0	0	1
4	31.000000	0	108.890000	0	0	1
...	...	...	...	...	...	...
9717	68.681732	0	81.221845	0	0	0
9718	81.548924	0	86.965825	0	0	0
9719	81.097689	1	104.690589	0	0	0
9720	79.000000	1	129.154725	0	0	0
9721	60.371378	0	75.587735	0	0	0

9722 rows × 6 columns

In [69]:
```
#here we ar merging the stroke column and other columns
merge = [Smote_X, gk]
Sampled_data = pd.concat(merge, axis = 1)
```

In [70]:
```
#printing the data we collected by resampling
Sampled_data
```

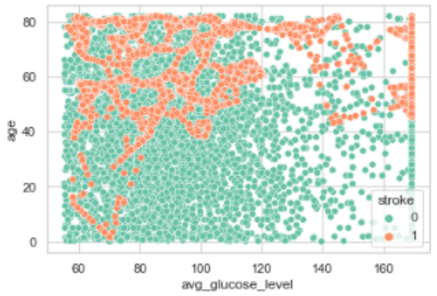
Out[70]:

	age	heart_disease	avg_glucose_level	hypertension	work_type_children	gender	stroke
0	17.000000	0	92.970000	0	0	1	0
1	13.000000	0	85.810000	0	1	1	0
2	55.000000	0	89.170000	0	0	0	0
3	42.000000	0	98.530000	0	0	1	0
4	31.000000	0	108.890000	0	0	1	0
...	...	...	...	...	...	...	...
9717	68.681732	0	81.221845	0	0	0	1
9718	81.548924	0	86.965825	0	0	0	1
9719	81.097689	1	104.690589	0	0	0	1
9720	79.000000	1	129.154725	0	0	0	1
9721	60.371378	0	75.587735	0	0	0	1

9722 rows × 7 columns

In [71]:
```
#scatterplot to check the stroke occurance after resampling the data
sns.scatterplot(data = Sampled_data, x ='avg_glucose_level', y = 'age', hue = 'stroke')
```

Out[71]:
```
<AxesSubplot:xlabel='avg_glucose_level', ylabel='age'>
```



After Using SMOTE Technique you can see Data has been resampled and Stroke ->1 has been increased

References:

https://www.kaggle.com/code/mohamedibrahim206/stroke-prediction-eda-modeling/data
https://app.mode.com/modeanalytics/reports/f2a442bd3b9a/details/notebook
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3825015/