Theoretical Machine Learning Project

Heart Stroke Prediction

By -
Thejaswin S(RA1911026010029)
Sarthak Gupta(RA1911026010041)
Shravya Sharan(RA1911026010055)
K1 SECTION

1. ABSTRACT

In recent times, Heart Stroke prediction has been one of the most complicated tasks in the medical field. In the modern era, approximately one person dies per minute due to a heart Stroke. Data science plays a crucial role in processing a massive amount of data in the field of healthcare. As heart stroke prediction is a complex task, there is a need to automate the prediction process to avoid associated risks and alert the patient well in advance.

Heart disease and strokes have rapidly increased globally, even at young ages. Stroke prediction is a complex task requiring a tremendous amount of data pre-processing. There is a need to automate the prediction process for the early detection of symptoms related to stroke so that it can be prevented at an early stage. In the proposed model, heart stroke prediction is performed on a dataset. We used machine learning algorithms like Gradient Boosting Classifier and Random Forest Classifier to train two different models for accurate prediction. The algorithm that best performed this task is Random Forest Classifier that gave an accuracy of approximately 90%.

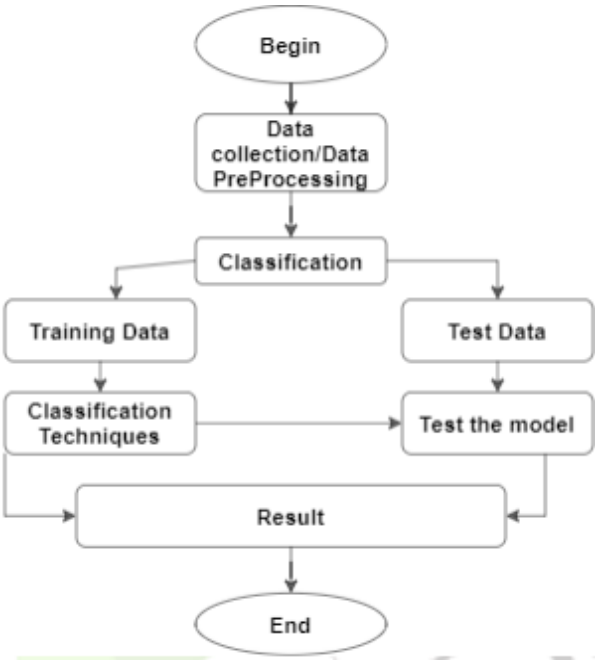
2. INTRODUCTION

Cardiovascular Diseases (CVDs) are the most common cause of death globally, representing 32% of all global deaths, with about 17.9 million people being affected. Out of these, the two most common CVDs are heart attack and heart strokes,e accounting for 85% of the total people. Heart attack is caused due to blockage of oxygen or blood supply to the heart muscle, while heart stroke is caused when there is blockage of the vessel feeding the brain. Although both diseases are different from each other, the risk factors contributing to them are pretty similar. The risk factors include unhealthy diet, tobacco use, diabetes, sedentary lifestyle, harmful alcohol use, high blood pressure and family history. Detecting heart stroke and taking medical action immediately can prolong life and help prevent heart disease in the future.

Machine learning has become one of the most demanding fields in modern technology. It is a form of artificial intelligence where the model can analyse the data, identify patterns and predict the outcome with minimal human intervention. Various machine learning algorithms can make heart stroke predictions in adults. It has become an intriguing research problem as multiple factors or parameters can influence the outcome. The factors include work type, gender, residence type, age, average glucose level, body mass index, smoking status of the individual and any previous heart disease.

3. PROPOSED SYSTEM

The proposed work predicts heart stroke by exploring the above mentioned four classification algorithms and does performance analysis. The objective of this study is to predict if the patient suffers from heart stroke effectively. The health professional enters the input values from the patient's health report. The data is fed into the model, which predicts the probability of having a heart stroke. Fig. shows the entire process involved.



4. METHODOLOGY

This section is divided into two parts; these are Data description and Machine learning classifiers. These two processes are described below:

a. *Data Description*: We used the heart stroke dataset available on the Kaggle website for our analysis. This dataset consists of a total of 12 attributes. The complete description of the attributes used in the proposed work is given below:
   - id: This attribute means a person's id. It's numerical data.
   - Age: This attribute means a person's age. It's numerical data.
   - Gender: This attribute means a person's gender. It's categorical data.
   - Hypertension: This attribute means that this person is hypertensive or not. It's numerical data.
   - Work type: This attribute represents the person work scenario. It's categorical data.
   - Residence type: This attribute represents the person living scenario. It's categorical data.
   - Heart disease: This attribute means whether this person has a heart disease person or not. It's numerical data.
   - Avg glucose level: This attribute means what was the level of a person's glucose condition. It's numerical data.
   - Bmi: This attribute means the body mass index of a person. It's numerical data.
   - Ever married: This attribute represents a person's married status. It's categorical data.
   - Smoking Status: This attribute means a person's smoking condition. It's categorical data.
   - Stroke: This attribute means a person previously had a stroke or not. It's numerical data. Attribute stroke is the decision class, and the rest of the attribute is the response class.

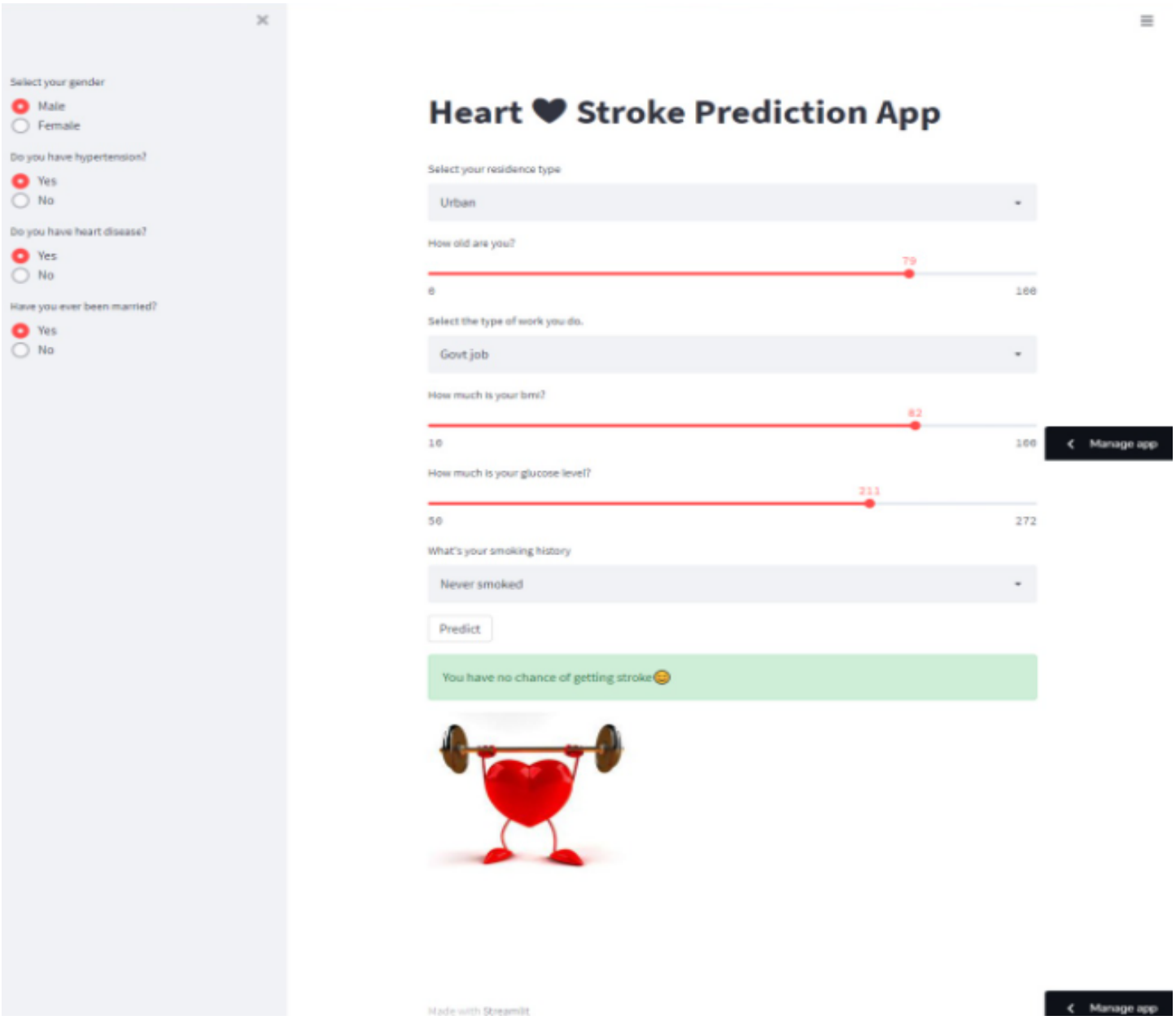
b. Machine Learning Classifiers:
   - Random Forest classifier:
     Random Forest algorithms are used for classification as well as regression. It creates a tree for the data and makes a prediction based on that. Random Forest algorithm can be used on large datasets and produce the same result even when large sets of record values are missing. The generated samples from the decision tree can be saved to be used on other data. There are two

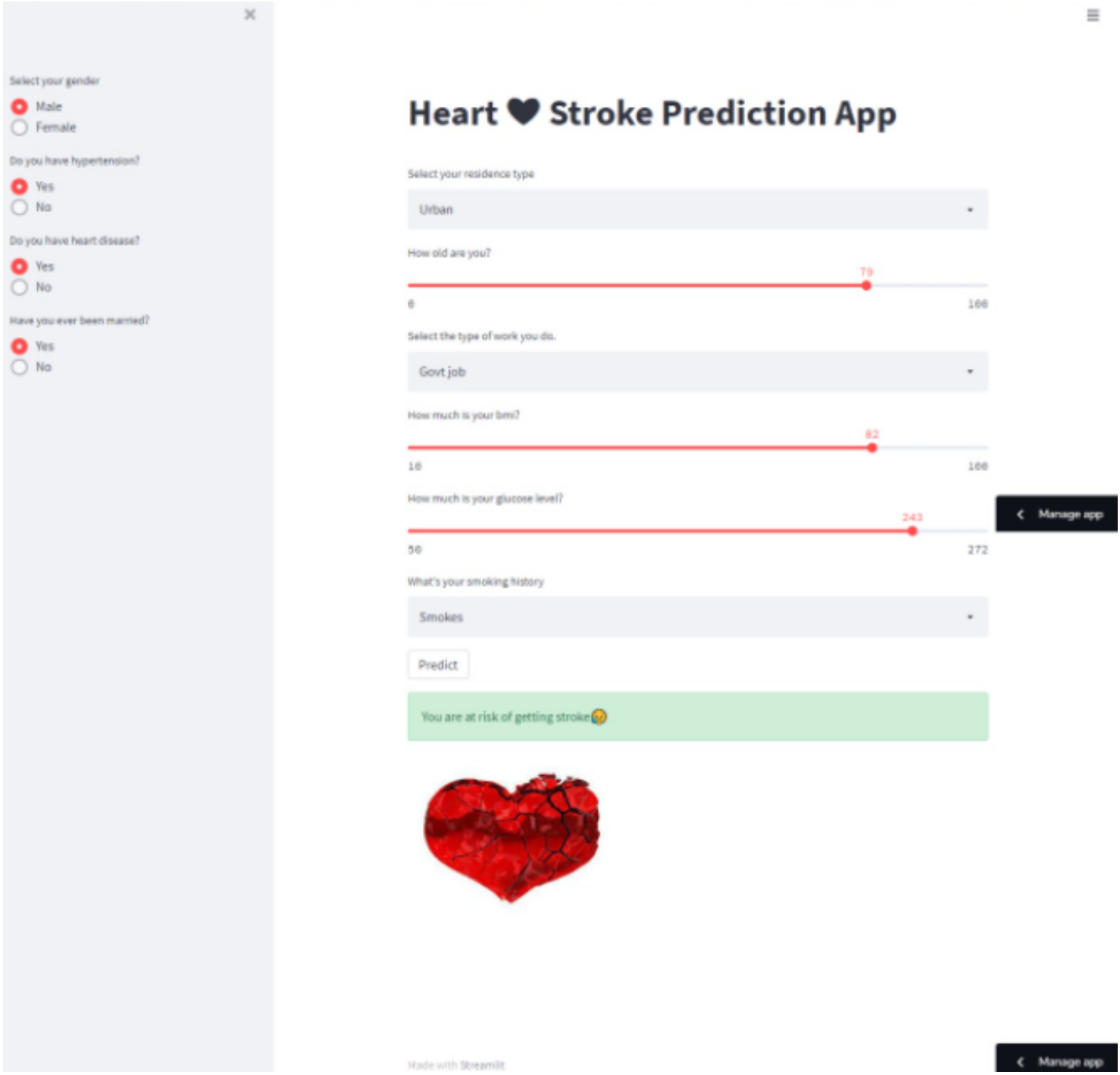
stages in a random forest: firstly, create a random forest, then make a prediction using a random forest classifier built in the first stage.

- Gradient Boosting classifier:
  Gradient boosting algorithm can be used for predicting continuous target variable (as a Regressor) and categorical target variable (as a Classifier). When used as a regressor, the cost function is Mean Square Error (MSE) and when it is used as a classifier, the cost function is Log loss.

5. IMPLEMENTATION

(code attached at the end of the report)





6. CONCLUSION

As heart diseases and strokes are increasing rapidly worldwide and causing deaths, it becomes necessary to develop an efficient system that would predict the heart stroke effectively beforehand so that immediate medical attention can be given. In the proposed method, the most effective algorithm for stroke prediction was obtained after a comparative analysis of the accuracy scores of various models. The most effective one was Random Forest, with an accuracy score of 90%.

Heart Stroke Prediction

- Random Forest Classifier : Accuracy - 0.90
- Gradient Boosting Classifier : Accuracy - 0.89

Context

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. This dataset is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relavant information about the patient.

Attribute Information

- 1) id: unique identifier
- 2) gender: "Male", "Female" or "Other"
- 3) age: age of the patient
- 4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
- 5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
- 6) ever_married: "No" or "Yes"
- 7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
- 8) Residence_type: "Rural" or "Urban"
- 9) avg_glucose_level: average glucose level in blood
- 10) bmi: body mass index
- 11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
- 12) stroke: 1 if the patient had a stroke or 0 if not

*Note: "Unknown" in smoking_status means that the information is unavailable for this patient

---

Contents(index)

Step 1. Data Load & EDA

In [192...
```python
import pandas as pd
```

In [ ]:

In [194...
```python
df = pd.read_csv('healthcare-dataset-stroke-data.csv')
```

In [195...
```python
df.head()
```

Out[195...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_lev
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.6
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.2
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.9
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.2
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.1

In [196...
```python
df.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
```

In [197...
```python
df.describe().T
```

Out[197...

	count	mean	std	min	25%	50%	75%	max
id	5110.0	36517.829354	21161.721625	67.00	17741.250	36932.000	54682.00	72940.00
age	5110.0	43.226614	22.612647	0.08	25.000	45.000	61.00	82.00
hypertension	5110.0	0.097456	0.296607	0.00	0.000	0.000	0.00	1.00
heart_disease	5110.0	0.054012	0.226063	0.00	0.000	0.000	0.00	1.00
avg_glucose_level	5110.0	106.147677	45.283560	55.12	77.245	91.885	114.09	271.74
bmi	4909.0	28.893237	7.854067	10.30	23.500	28.100	33.10	97.60
stroke	5110.0	0.048728	0.215320	0.00	0.000	0.000	0.00	1.00

In [198...
```python
df.nunique()
```

Out[198...
```
id                   5110
gender                  3
age                   104
hypertension            2
heart_disease           2
ever_married            2
work_type               5
Residence_type          2
avg_glucose_level    3979
bmi                   418
smoking_status          4
stroke                  2
dtype: int64
```

In [199...
```python
df.gender.unique()
```

Out[199...
```
array(['Male', 'Female', 'Other'], dtype=object)
```

In [200...
```python
df.work_type.unique()
```

Out[200...
```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```

In [201...
```python
df.smoking_status.unique()
```

Out[201...
```
array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

In [202...
```python
df.bmi
```

Out[202...
```
0       36.6
1        NaN
2       32.5
3       34.4
4       24.0
        ...
5105     NaN
5106    40.0
5107    30.6
5108    25.6
5109    26.2
Name: bmi, Length: 5110, dtype: float64
```

In [203...
```python
df.avg_glucose_level
```

Out[203...
```
0       228.69
1       202.21
2       105.92
3       171.23
4       174.12
         ...
5105     83.75
5106    125.20
5107     82.99
5108    166.29
5109     85.28
Name: avg_glucose_level, Length: 5110, dtype: float64
```

- ID : delete
- Age : transform to Category (20s, 30s, 40s ...)
- Gender : transfrom by One-hot Encoding
- work_type : transfrom by One-hot Encoding
- smoking_status : transfrom by One-hot Encoding
- bmi : transform to Category (20-29, 30-39 ...)
- avg_glucose_level : transform to Category (0-50, 51-100 ...)
- Others : binary

It's sparse table, so we use RandomForest and GradientBoosting Algorithm instead of KNN

In [204...
```python
df
```

Out[204...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose
0	9046	Male	67.0	0	1	Yes	Private	Urban	2
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	2
2	31112	Male	80.0	0	1	Yes	Private	Rural	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	1
...	...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	Urban	
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	1
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	
5108	37544	Male	51.0	0	0	Yes	Private	Rural	1
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	

5110 rows × 12 columns

Step 2. Feature Engineering

In [205...
```python
df.gender.value_counts()
```

Out[205...
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [206...
```python
df[df['gender'] == 'Other']
```

Out[206...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose
3116	56156	Other	26.0	0	0	No	Private	Rural	1

In [207...
```python
df[df['gender'] == 'Other'].index
```

Out[207...
```
Int64Index([3116], dtype='int64')
```

In [208...
```python
df.drop(df[df['gender'] == 'Other'].index, inplace=True)
```

In [209...
```python
df
```

Out[209...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose
0	9046	Male	67.0	0	1	Yes	Private	Urban	2
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	2
2	31112	Male	80.0	0	1	Yes	Private	Rural	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	1
...	...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	Urban	
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	1
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	
5108	37544	Male	51.0	0	0	Yes	Private	Rural	1
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	

5109 rows × 12 columns

2-a. Feature Engineering - Binary Features

1. gender : 0 or 1

In [210...
```python
df['gender'] = df['gender'].apply(lambda x : 0 if x == 'Female' else 1)
```

1. ever_married : 0 or 1

In [211...
```python
df['ever_married'].apply(lambda x : 0 if x == 'No' else 1).unique()
```

Out[211...
```
array([1, 0], dtype=int64)
```

In [212...
```python
df['ever_married'] = df['ever_married'].apply(lambda x : 0 if x == 'No' else 1)
```

1. Residence_type

In [213...
```python
df['Residence_type'].value_counts()
```

Out[213...
```
Urban    2596
Rural    2513
Name: Residence_type, dtype: int64
```

In [214...
```python
df['Residence_type'] = df['Residence_type'].apply(lambda x : 0 if x == 'Rural' else 1
```

In [215...
```python
df
```

Out[215...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose
0	9046	1	67.0	0	1	1	Private	1	2
1	51676	0	61.0	0	0	1	Self-employed	0	2
2	31112	1	80.0	0	1	1	Private	0	1
3	60182	0	49.0	0	0	1	Private	1	1
4	1665	0	79.0	1	0	1	Self-employed	0	1
...	...	...	...	...	...	...	...	...	
5105	18234	0	80.0	1	0	1	Private	1	
5106	44873	0	81.0	0	0	1	Self-employed	1	1
5107	19723	0	35.0	0	0	1	Self-employed	0	
5108	37544	1	51.0	0	0	1	Private	0	1
5109	44679	0	44.0	0	0	1	Govt_job	1	

5109 rows × 12 columns

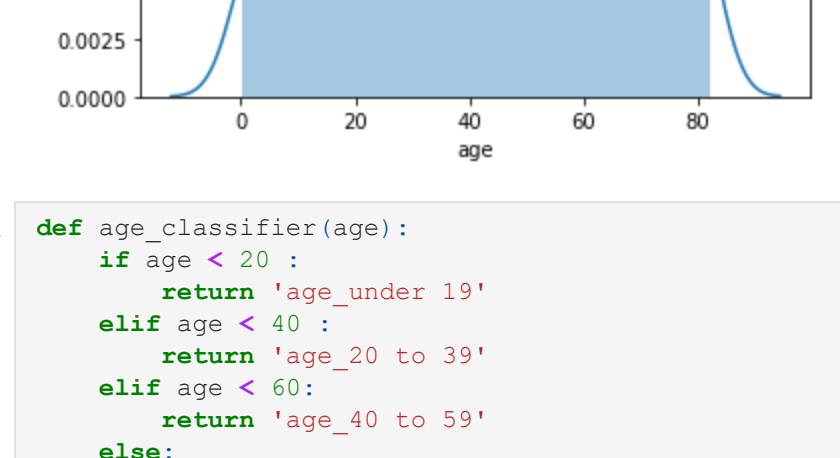
2-b. Feature Engineering - Continuous Features

In [216...
```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [217...
```python
import warnings
warnings.filterwarnings("ignore")
```

1. age

In [218...
```python
sns.distplot(df['age']);
```



In [219...
```python
def age_classifier(age):
    if age < 20 :
        return 'age_under 19'
    elif age < 40 :
        return 'age_20 to 39'
    elif age < 60:
        return 'age_40 to 59'
    else:
        return 'age_over 60'
```

In [220...
```python
df['age'].apply(lambda x : age_classifier(x))
```

Out[220...
```
0        age_over 60
1        age_over 60
2        age_over 60
3       age_40 to 59
4        age_over 60
            ...
5105     age_over 60
5106     age_over 60
5107    age_20 to 39
5108    age_40 to 59
5109    age_40 to 59
Name: age, Length: 5109, dtype: object
```

In [221...
```python
df['age'] = df['age'].apply(lambda x : age_classifier(x))
```

In [222...
```python
df['age'].unique()
```

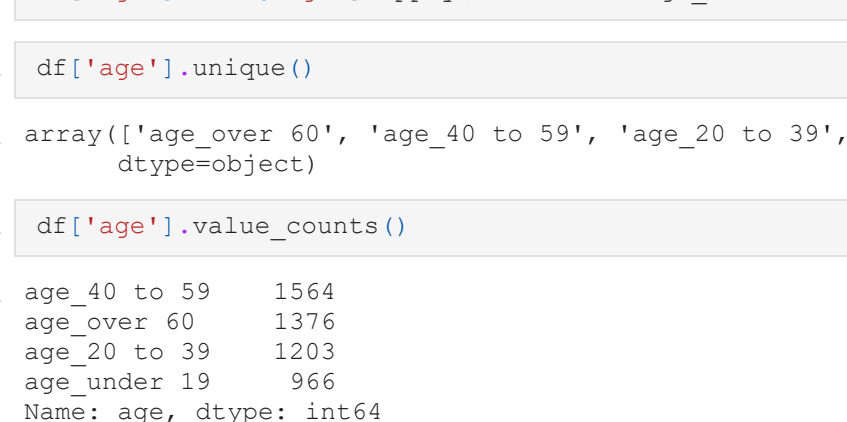
Out[222...
```
array(['age_over 60', 'age_40 to 59', 'age_20 to 39', 'age_under 19'],
      dtype=object)
```

In [223...
```python
df['age'].value_counts()
```

Out[223...
```
age_40 to 59    1564
age_over 60     1376
age_20 to 39    1203
age_under 19     966
Name: age, dtype: int64
```

1. avg_glucose_level

In [224...
```python
sns.distplot(df['avg_glucose_level']);
```



In [225...
```python
df['avg_glucose_level'].describe()
```

Out[225...
```
count    5109.000000
mean      106.140399
std        45.285004
min        55.120000
25%        77.240000
50%        91.880000
75%       114.090000
max       271.740000
Name: avg_glucose_level, dtype: float64
```

---

let's split by 25%, 50%, 75% line

In [226...
```python
def glucose_level_classifier(level):
    if level < 77:
        return 'gl_under 25%'
    elif level < 91:
        return 'gl_26% to 50%'
    elif level < 114:
        return 'gl_50% to 75%'
    else:
        return 'gl_over 75%'
```

In [227...
```python
df['avg_glucose_level'].apply(lambda x : glucose_level_classifier(x))
```

Out[227...
```
0         gl_over 75%
1         gl_over 75%
2       gl_50% to 75%
3         gl_over 75%
4         gl_over 75%
            ...
5105    gl_26% to 50%
5106      gl_over 75%
5107    gl_26% to 50%
5108      gl_over 75%
5109    gl_26% to 50%
Name: avg_glucose_level, Length: 5109, dtype: object
```

In [228...
```python
# let's check 'is properly distributed?'
df['avg_glucose_level'].apply(lambda x : glucose_level_classifier(x)).value_counts()
```

Out[228...
```
gl_50% to 75%    1340
gl_over 75%      1281
gl_under 25%     1260
gl_26% to 50%    1228
Name: avg_glucose_level, dtype: int64
```

In [229...
```python
df['avg_glucose_level'] = df['avg_glucose_level'].apply(lambda x : glucose_level_class
```

1. bmi

In [230...
```python
df['bmi'].describe()
```

Out[230...
```
count    4908.00000
mean       28.89456
std         7.85432
min        10.30000
25%        23.50000
50%        28.10000
75%        33.10000
max        97.60000
Name: bmi, dtype: float64
```

Let's classify 'bmi' the same as 'avg_glucose_level' (distribution %)

In [231...
```python
def bmi_classifier(figure):
    if figure < 23:
        return 'bmi_under 25%'
    elif figure < 28:
        return 'bmi_26% to 50%'
    elif figure < 33:
        return 'bmi_50% to 75%'
    else:
        return 'bmi_over 75%'
```

!! But 'bmi' has 101 null values

In [232...
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5109 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5109 non-null   int64
 1   gender             5109 non-null   int64
 2   age                5109 non-null   object
 3   hypertension       5109 non-null   int64
 4   heart_disease      5109 non-null   int64
 5   ever_married       5109 non-null   int64
 6   work_type          5109 non-null   object
 7   Residence_type     5109 non-null   int64
 8   avg_glucose_level  5109 non-null   object
 9   bmi                4908 non-null   float64
 10  smoking_status     5109 non-null   object
 11  stroke             5109 non-null   int64
dtypes: float64(1), int64(7), object(4)
memory usage: 518.9+ KB
```

In [233...
```python
df[df.bmi.isnull()]
```

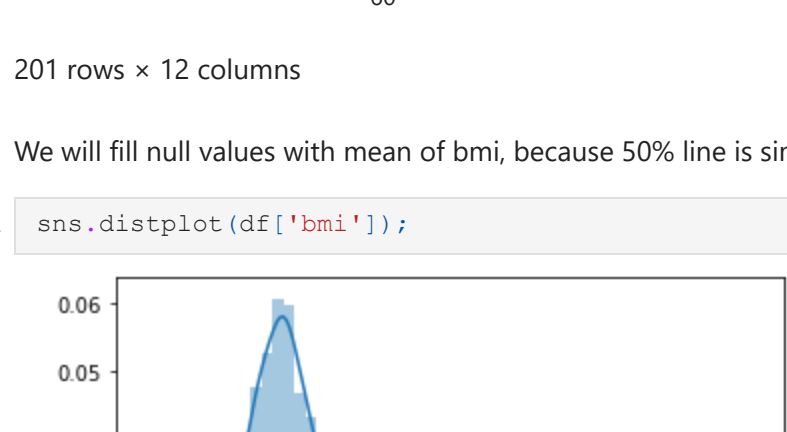
Out[233...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
1	51676	0	age_over 60	0	0	1	Self-employed	0	g
8	27419	0	age_40 to 59	0	0	1	Private	0	gl_
13	8213	1	age_over 60	0	1	1	Private	1	g
19	25226	1	age_40 to 59	0	1	0	Govt_job	1	g
27	61843	1	age_40 to 59	0	0	1	Private	0	g
...	...	...	...	...	...	...	...	...	
5039	42007	1	age_40 to 59	0	0	0	Private	0	gl_
5048	28788	1	age_40 to 59	0	0	1	Private	1	g
5093	32235	0	age_40 to 59	1	0	1	Govt_job	0	gl_5
5099	7293	1	age_40 to 59	0	0	1	Private	0	gl_2
5105	18234	0	age_over 60	1	0	1	Private	1	gl_2

201 rows × 12 columns

We will fill null values with mean of bmi, because 50% line is similar with mean.

In [234...
```python
sns.distplot(df['bmi']);
```



In [235...
```python
df['bmi'].fillna(28, inplace=True)
```

In [236...
```python
df.bmi.isnull().sum()
```

Out[236...
```
0
```

Filled well

In [237...
```python
df['bmi'].apply(lambda x : bmi_classifier(x)).value_counts()
```

Out[237...
```
bmi_50% to 75%     1434
bmi_26% to 50%     1348
bmi_over 75%       1250
bmi_under 25%      1077
Name: bmi, dtype: int64
```

In [238...
```python
df['bmi'] = df['bmi'].apply(lambda x : bmi_classifier(x))
```

In [239...
```python
df
```

Out[239...

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	9046	1	age_over 60	0	1	1	Private	1	g
1	51676	0	age_over 60	0	0	1	Self-employed	0	g
2	31112	1	age_over 60	0	1	1	Private	0	gl_5
3	60182	0	age_40 to 59	0	0	1	Private	1	g
4	1665	0	age_over 60	1	0	1	Self-employed	0	g
...	...	...	...	...	...	...	...	...	
5105	18234	0	age_over 60	1	0	1	Private	1	gl_2
5106	44873	0	age_over 60	0	0	1	Self-employed	1	g
5107	19723	0	age_20 to 39	0	0	1	Self-employed	0	gl_2
5108	37544	1	age_40 to 59	0	0	1	Private	0	g
5109	44679	0	age_40 to 59	0	0	1	Govt_job	1	gl_2

5109 rows × 12 columns

2-c. Feature Engineering - Categorical Features

In This part, We try One-Hot Encoding to All Categorical Features including we tranformed before(age, avg_glucose_level, bmi)

In [240...
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5109 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5109 non-null   int64
 1   gender             5109 non-null   int64
 2   age                5109 non-null   object
 3   hypertension       5109 non-null   int64
 4   heart_disease      5109 non-null   int64
 5   ever_married       5109 non-null   int64
 6   work_type          5109 non-null   object
 7   Residence_type     5109 non-null   int64
 8   avg_glucose_level  5109 non-null   object
 9   bmi                5109 non-null   object
 10  smoking_status     5109 non-null   object
 11  stroke             5109 non-null   int64
dtypes: int64(7), object(5)
memory usage: 518.9+ KB
```

In [241...
```python
columns = df.columns
```

In [242...
```python
columns
```

Out[242...
```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [243...
```python
df[columns[0]].dtype
```

Out[243...
```
dtype('int64')
```

In [254...
```python
num_cols = []
cat_cols = []
for col in columns:
    if df[col].dtype == 'int64':
        num_cols.append(col)
    else:
        cat_cols.append(col)
```

In [255...
```python
print('numeric columns : {}'.format(num_cols))
print('categorical columns : {}'.format(cat_cols))
```

```
numeric columns : ['id', 'gender', 'hypertension', 'heart_disease', 'ever_married', 'R
esidence_type', 'stroke']
categorical columns : ['age', 'work_type', 'avg_glucose_level', 'bmi', 'smoking_statu
s']
```

1. age

In [256...
```python
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
X_encoded = lb.fit_transform(df['age'])
X_encoded
```

Out[256...
```
array([[0, 0, 1, 0],
       [0, 0, 1, 0],
       [0, 0, 1, 0],
       ...,
       [1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 1, 0, 0]])
```

In [257...
```python
df['age']
```

Out[257...
```
0        age_over 60
1        age_over 60
2        age_over 60
3       age_40 to 59
4        age_over 60
            ...
5105     age_over 60
5106     age_over 60
5107    age_20 to 39
5108    age_40 to 59
5109    age_40 to 59
Name: age, Length: 5109, dtype: object
```

- colums : [age_20 to 39 ,age_40 to 59, age_over 60, age_under 19]

In [258...
```python
df['age'].unique()
```

Out[258...
```
array(['age_over 60', 'age_40 to 59', 'age_20 to 39', 'age_under 19'],
      dtype=object)
```

In [259...
```python
pd.DataFrame(X_encoded, columns=['age_20 to 39' ,'age_40 to 59', 'age_over 60', 'age_u
```

Out[259...

	age_20 to 39	age_40 to 59	age_over 60	age_under 19
0	0	0	1	0
1	0	0	1	0
2	0	0	1	0
3	0	1	0	0
4	0	0	1	0
...	...	...	...	...
5104	0	0	1	0
5105	0	0	1	0
5106	1	0	0	0
5107	0	1	0	0
5108	0	1	0	0

5109 rows × 4 columns

In [260...
```python
age_df = pd.DataFrame(X_encoded, columns=['age_20 to 39' ,'age_40 to 59', 'age_over 60
```

In [261...
```python
cat_cols
```

Out[261...
```
['age', 'work_type', 'avg_glucose_level', 'bmi', 'smoking_status']
```

1. work_type

In [262...
```python
one_hot_encoded = lb.fit_transform(df['work_type'])
```

In [263...
```python
one_hot_encoded[-20:]
```

Out[263...
```
array([[1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0]])
```

In [264...
```python
df['work_type'].tail(20)
```

Out[264...
```
5090         Govt_job
5091          Private
5092         Govt_job
5093         Govt_job
5094         children
5095         children
5096         Govt_job
5097    Self-employed
5098         children
5099          Private
5100    Self-employed
5101          Private
5102          Private
5103          Private
5104         children
5105          Private
5106    Self-employed
5107    Self-employed
5108          Private
5109         Govt_job
Name: work_type, dtype: object
```

In [265...
```python
df['work_type'].unique()
```

Out[265...
```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```

---

mapping

- colums : ['Govt_job','Never_worked', 'Private', 'Self-employed', 'Children']

In [266...
```python
pd.DataFrame(one_hot_encoded, columns=['Govt_job','Never_worked', 'Private', 'Self-emp
```

Out[266...

	Govt_job	Never_worked	Private	Self-employed	Children
0	0	0	1	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	0	0
4	0	0	0	1	0
...	...	...	...	...	...
5104	0	0	1	0	0
5105	0	0	0	1	0
5106	0	0	0	1	0
5107	0	0	1	0	0
5108	1	0	0	0	0

5109 rows × 5 columns

In [267...
```python
work_type_df = pd.DataFrame(one_hot_encoded, columns=['Govt_job','Never_worked', 'Pri
```

1. avg_glucose_level

In [268...
```python
df['avg_glucose_level']
```

Out[268...
```
0         gl_over 75%
1         gl_over 75%
2       gl_50% to 75%
3         gl_over 75%
4         gl_over 75%
            ...
5105    gl_26% to 50%
5106      gl_over 75%
5107    gl_26% to 50%
5108      gl_over 75%
5109    gl_26% to 50%
Name: avg_glucose_level, Length: 5109, dtype: object
```

In [269...
```python
one_hot_encoded = lb.fit_transform(df['avg_glucose_level'])
```

In [270...
```python
one_hot_encoded
```

Out[270...
```
array([[0, 0, 1, 0],
       [0, 0, 1, 0],
       [0, 1, 0, 0],
       ...,
```

```
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [1, 0, 0, 0]])
```

In [271]:
```
df['avg_glucose_level'].unique()
```
Out[271]:
```
array(['gl_over 75%', 'gl_50% to 75%', 'gl_under 25%', 'gl_26% to 50%'],
      dtype=object)
```

mapping

- columns : ['gl_26% to 50%', 'gl_50% to 75%', 'gl_over 75%', 'gl_under 25%']

In [272]:
```
pd.DataFrame(one_hot_encoded, columns=['gl_26% to 50%', 'gl_50% to 75%', 'gl_over 75%
```
Out[272]:

	gl_26% to 50%	gl_50% to 75%	gl_over 75%	gl_under 25%
0	0	0	1	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
...	...	...	...	...
5104	1	0	0	0
5105	0	0	1	0
5106	1	0	0	0
5107	0	0	1	0
5108	1	0	0	0

5109 rows × 4 columns

In [273]:
```
agl_df = pd.DataFrame(one_hot_encoded, columns=['gl_26% to 50%', 'gl_50% to 75%', 'gl_
```

1. bmi

In [274]:
```
df['bmi']
```
Out[274]:
```
0         bmi_over 75%
1       bmi_50% to 75%
2       bmi_50% to 75%
3         bmi_over 75%
4       bmi_26% to 50%
             ...
5105    bmi_50% to 75%
5106      bmi_over 75%
5107    bmi_50% to 75%
5108    bmi_26% to 50%
5109    bmi_26% to 50%
Name: bmi, Length: 5109, dtype: object
```

In [275]:
```
one_hot_encoded = lb.fit_transform(df['bmi'])
one_hot_encoded
```
Out[275]:
```
array([[0, 0, 1, 0],
       [0, 1, 0, 0],
       [0, 1, 0, 0],
       ...,
       [0, 1, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0]])
```

In [276]:
```
df['bmi'].unique()
```
Out[276]:
```
array(['bmi_over 75%', 'bmi_50% to 75%', 'bmi_26% to 50%',
       'bmi_under 25%'], dtype=object)
```

---

mapping

- columns : ['bmi_26% to 50%', 'bmi_50% to 75%', 'bmi_over 75%', 'bmi_under 25%']

In [277]:
```
pd.DataFrame(one_hot_encoded, columns=['bmi_26% to 50%', 'bmi_50% to 75%', 'bmi_over 7
```
Out[277]:

	bmi_26% to 50%	bmi_50% to 75%	bmi_over 75%	bmi_under 25%
0	0	0	1	0
1	0	1	0	0
2	0	1	0	0
3	0	0	1	0
4	1	0	0	0
...	...	...	...	...
5104	0	1	0	0
5105	0	0	1	0
5106	0	1	0	0
5107	1	0	0	0
5108	1	0	0	0

5109 rows × 4 columns

In [278]:
```
bmi_df = pd.DataFrame(one_hot_encoded, columns=['bmi_26% to 50%', 'bmi_50% to 75%', 'b
```

In [279]:
```
df['smoking_status']
```
Out[279]:
```
0       formerly smoked
1          never smoked
2          never smoked
3                smokes
4          never smoked
             ...
5105       never smoked
5106       never smoked
5107       never smoked
5108    formerly smoked
5109            Unknown
Name: smoking_status, Length: 5109, dtype: object
```

In [280]:
```
df['smoking_status'].unique()
```
Out[280]:
```
array(['formerly smoked', 'never smoked', 'smokes', 'Unknown'],
      dtype=object)
```

In [281]:
```
lb.fit_transform(df['smoking_status'])
```
Out[281]:
```
array([[0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 1, 0],
       ...,
       [0, 0, 1, 0],
       [0, 1, 0, 0],
       [1, 0, 0, 0]])
```

In [282]:
```
one_hot_encoded = lb.fit_transform(df['smoking_status'])
```

mapping

- columns : ['Unknown','formerly smoked','never smoked','smokes']

In [283]:
```
pd.DataFrame(one_hot_encoded, columns=['Unknown','formerly smoked','never smoked','smo
```
Out[283]:

	Unknown	formerly smoked	never smoked	smokes
0	0	1	0	0
1	0	0	1	0
2	0	0	1	0
3	0	0	0	1
4	0	0	1	0
...	...	...	...	...
5104	0	0	1	0
5105	0	0	1	0
5106	0	0	1	0
5107	0	1	0	0
5108	1	0	0	0

5109 rows × 4 columns

In [284]:
```
smoked_df = pd.DataFrame(one_hot_encoded, columns=['Unknown','formerly smoked','never
```

In [285]:
```
cat_cols.append('age')
```

In [286]:
```
cat_cols
```
Out[286]:
```
['age', 'work_type', 'avg_glucose_level', 'bmi', 'smoking_status', 'age']
```

finally! merge!

In [287]:
```
df
```
Out[287]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	9046	1	age_over 60	0	1	1	Private	1	gl
1	51676	0	age_over 60	0	0	1	Self-employed	0	gl
2	31112	1	age_over 60	0	1	1	Private	0	gl_5
3	60182	0	age_40 to 59	0	0	1	Private	1	gl
4	1665	0	age_over 60	1	0	1	Self-employed	0	gl
...	...	...	...	...	...	...	...	...	
5105	18234	0	age_over 60	1	0	1	Private	1	gl_2
5106	44873	0	age_over 60	0	0	1	Self-employed	1	gl
5107	19723	0	age_20 to 39	0	0	1	Self-employed	0	gl_2
5108	37544	1	age_40 to 59	0	0	1	Private	0	gl
5109	44679	0	age_40 to 59	0	0	1	Govt_job	1	gl_2

5109 rows × 12 columns

In [288]:
```
df.drop(cat_cols,axis=1,inplace=True)
df
```
Out[288]:

	id	gender	hypertension	heart_disease	ever_married	Residence_type	stroke
0	9046	1	0	1	1	1	1
1	51676	0	0	0	1	0	1
2	31112	1	0	1	1	0	1
3	60182	0	0	0	1	1	1
4	1665	0	1	0	1	0	1
...	...	...	...	...	...	...	...
5105	18234	0	1	0	1	1	0
5106	44873	0	0	0	1	1	0
5107	19723	0	0	0	1	0	0
5108	37544	1	0	0	1	0	0
5109	44679	0	0	0	1	1	0

5109 rows × 7 columns

In [289]:
```
# delete meaningless column (for machie learning)
df.drop('id', axis=1, inplace=True)
```

In [290]:
```
df
```
Out[290]:

	gender	hypertension	heart_disease	ever_married	Residence_type	stroke
0	1	0	1	1	1	1
1	0	0	0	1	0	1
2	1	0	1	1	0	1
3	0	0	0	1	1	1
4	0	1	0	1	0	1
...	...	...	...	...	...	...
5105	0	1	0	1	1	0
5106	0	0	0	1	1	0
5107	0	0	0	1	0	0
5108	1	0	0	1	0	0
5109	0	0	0	1	1	0

5109 rows × 6 columns

In [291]:
```
cat_cols
```
Out[291]:
```
['age', 'work_type', 'avg_glucose_level', 'bmi', 'smoking_status', 'age']
```

In [292]:
```
df.reset_index().iloc[:,1:]
```
Out[292]:

	gender	hypertension	heart_disease	ever_married	Residence_type	stroke
0	1	0	1	1	1	1
1	0	0	0	1	0	1
2	1	0	1	1	0	1
3	0	0	0	1	1	1
4	0	1	0	1	0	1
...	...	...	...	...	...	...
5104	0	1	0	1	1	0
5105	0	0	0	1	1	0
5106	0	0	0	1	0	0
5107	1	0	0	1	0	0
5108	0	0	0	1	1	0

5109 rows × 6 columns

In [293]:
```
df = df.reset_index().iloc[:,1:]
```

In [294]:
```
df
```
Out[294]:

	gender	hypertension	heart_disease	ever_married	Residence_type	stroke
0	1	0	1	1	1	1
1	0	0	0	1	0	1
2	1	0	1	1	0	1
3	0	0	0	1	1	1
4	0	1	0	1	0	1
...	...	...	...	...	...	...
5104	0	1	0	1	1	0
5105	0	0	0	1	1	0
5106	0	0	0	1	0	0
5107	1	0	0	1	0	0
5108	0	0	0	1	1	0

5109 rows × 6 columns

In [295]:
```
pd.concat([df,age_df,agl_df,work_type_df,bmi_df,smoked_df], axis=1)
```
Out[295]:

	gender	hypertension	heart_disease	ever_married	Residence_type	stroke	age_20 to 39	age_40 to 59	age_over 60
0	1	0	1	1	1	1	0	0	1
1	0	0	0	1	0	1	0	0	1
2	1	0	1	1	0	1	0	0	1
3	0	0	0	1	1	1	0	1	0
4	0	1	0	1	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...
5104	0	1	0	1	1	0	0	0	1
5105	0	0	0	1	1	0	0	0	1
5106	0	0	0	1	0	0	1	0	0
5107	1	0	0	1	0	0	0	1	0
5108	0	0	0	1	1	0	0	1	0

5109 rows × 27 columns

In [296]:
```
pd.concat([df,age_df,agl_df,work_type_df,bmi_df,smoked_df], axis=1).isnull().sum()
```
Out[296]:
```
gender             0
hypertension       0
heart_disease      0
ever_married       0
Residence_type     0
stroke             0
age_20 to 39       0
age_40 to 59       0
age_over 60        0
age_under 19       0
gl_26% to 50%      0
gl_50% to 75%      0
gl_over 75%        0
gl_under 25%       0
Govt_job           0
Never_worked       0
Private            0
Self-employed      0
Children           0
bmi_26% to 50%     0
bmi_50% to 75%     0
bmi_over 75%       0
bmi_under 25%      0
Unknown            0
formerly smoked    0
never smoked       0
smokes             0
dtype: int64
```

In [297]:
```
final_df = pd.concat([df,age_df,agl_df,work_type_df,bmi_df,smoked_df], axis=1)
```

Step 3. Train, Test set split & Upsampling

In [298]:
```
from sklearn.model_selection import train_test_split
```

In [299]:
```
X = final_df.drop('stroke', axis=1)
y = final_df['stroke']
```

In [300]:
```
X.shape, y.shape
```
Out[300]:
```
((5109, 26), (5109,))
```

In [301]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=11
```

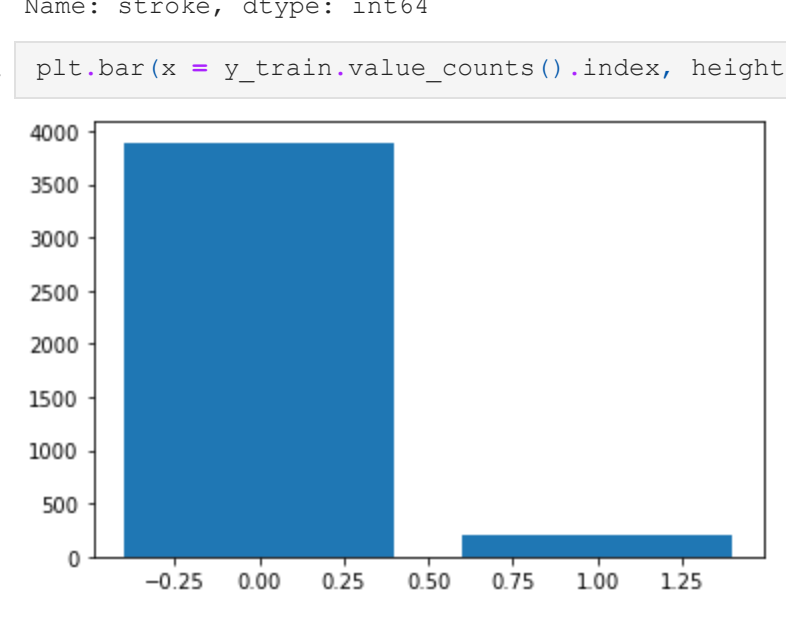
In [302]:
```
X_train.shape, y_train.shape
```
Out[302]:
```
((4087, 26), (4087,))
```

In [303]:
```
X_test.shape, y_test.shape
```
Out[303]:
```
((1022, 26), (1022,))
```

In [304]:
```
y_train.value_counts()
```
Out[304]:
```
0    3893
1     194
Name: stroke, dtype: int64
```

In [305]:
```
y_test.value_counts()
```
Out[305]:
```
0    967
1     55
Name: stroke, dtype: int64
```

In [306]:
```
plt.bar(x = y_train.value_counts().index, height = y_train.value_counts().values);
```



It's imbalanced data, we have to upsample '1' in **training set-y**

In [307]:
```
from sklearn.utils import resample
```

In [308]:
```
train_df = pd.concat([X_train,y_train], axis=1)
```

In [309]:
```
train_df
```
Out[309]:

	gender	hypertension	heart_disease	ever_married	Residence_type	age_20 to 39	age_40 to 59	age_over 60	age_unde
1977	1	0	0	1	1	1	0	0	
3378	0	0	0	1	1	0	1	0	
924	0	0	0	1	1	0	0	1	
4104	0	0	0	0	1	0	1	0	
3878	0	0	0	1	0	1	0	0	
...	...	...	...	...	...	...	...	...	
681	0	0	0	0	0	0	0	0	
4182	0	0	0	1	0	0	0	1	
4820	0	0	0	1	0	1	0	0	
2004	1	0	0	1	0	0	1	0	
3924	0	0	0	1	0	0	0	1	

4087 rows × 27 columns

In [310]:
```
train_0 = train_df[train_df['stroke']==0]
train_1 = train_df[train_df['stroke']==1]
```

In [311]:
```
train_0.shape, train_1.shape
```
Out[311]:
```
((3893, 27), (194, 27))
```

In [312]:
```
upsampled_train_1 = resample(train_1,
                             replace=True,
                             n_samples=3893,
                             random_state=123
                            )
upsampled_train_1.shape
```
Out[312]:
```
(3893, 27)
```

In [313]:
```
upsampled_train = pd.concat([train_0, upsampled_train_1])
```

In [314]:
```
upsampled_train['stroke'].value_counts()
```
Out[314]:
```
1    3893
0    3893
Name: stroke, dtype: int64
```

In [315]:
```
X_train = upsampled_train.drop('stroke',axis=1)
y_train = upsampled_train['stroke']
```

Step 4. Modeling & Prediction

1. RandomForestClassifier

In [316]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [323]:
```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
print('Train Accuracy : {:.2f}'.format(rfc.score(X_train, y_train)))
print('Test Accuracy : {:.2f}'.format(rfc.score(X_test, y_test)))
```
```
Train Accuracy : 0.97
Test Accuracy : 0.90
```

1. GradientBoostingClassifier

In [318]:
```
best_params = {}
score = 0
from sklearn.ensemble import GradientBoostingClassifier
for i in range(1,8):
    for j in [50, 100, 150, 200, 250, 300, 350, 400]:
        gbc = GradientBoostingClassifier(max_depth=i,
                                         n_estimators=j
                                        )
        gbc.fit(X_train, y_train)

        print('max_depth : {}'.format(i))
        print('n_estimators : {}'.format(j))
        print('Train Score : {}'.format(gbc.score(X_train,y_train)))
        print('Train Score : {}'.format(gbc.score(X_test,y_test)))
        print('----------------------------------------------------')
        if gbc.score(X_test,y_test) > score:
            score = gbc.score(X_test, y_test)
            best_params['max_depth'] = i
            best_params['n_estimators'] = j
```
```
max_depth : 1
n_estimators : 50
Train Score : 0.7499357821731313
Train Score : 0.7426614481409002
----------------------------------------------------
max_depth : 1
n_estimators : 100
Train Score : 0.7528898022090933
Train Score : 0.7172211350293543
----------------------------------------------------
max_depth : 1
n_estimators : 150
Train Score : 0.7549447726688929
Train Score : 0.7084148727984344
----------------------------------------------------
max_depth : 1
n_estimators : 200
Train Score : 0.7537888517852556
Train Score : 0.7074363992172211
----------------------------------------------------
max_depth : 1
n_estimators : 250
Train Score : 0.7549447726688929
Train Score : 0.7045009784735812
----------------------------------------------------
max_depth : 1
n_estimators : 300
Train Score : 0.7566144361674801
Train Score : 0.7035225048923679
----------------------------------------------------
max_depth : 1
n_estimators : 350
Train Score : 0.7563575648600052
Train Score : 0.7025440313111546
----------------------------------------------------
max_depth : 1
n_estimators : 400
Train Score : 0.7564860005137426
Train Score : 0.7025440313111546
----------------------------------------------------
max_depth : 2
n_estimators : 50
Train Score : 0.7634215258155664
Train Score : 0.7152641878669276
----------------------------------------------------
max_depth : 2
n_estimators : 100
Train Score : 0.7712561006935525
Train Score : 0.6966731898238747
----------------------------------------------------
max_depth : 2
n_estimators : 150
Train Score : 0.7833290521448755
Train Score : 0.6956947162426614
----------------------------------------------------
max_depth : 2
n_estimators : 200
Train Score : 0.7862830721808374
Train Score : 0.7025440313111546
----------------------------------------------------
max_depth : 2
n_estimators : 250
Train Score : 0.7968147957873105
Train Score : 0.7025440313111546
----------------------------------------------------
max_depth : 2
n_estimators : 300
Train Score : 0.7954020035961983
Train Score : 0.7084148727984344
----------------------------------------------------
max_depth : 2
n_estimators : 350
Train Score : 0.8006678653994349
Train Score : 0.7084148727984344
----------------------------------------------------
max_depth : 2
n_estimators : 400
Train Score : 0.8115848959671205
Train Score : 0.7093933463796478
----------------------------------------------------
max_depth : 3
n_estimators : 50
Train Score : 0.7819162599537631
Train Score : 0.7064579256360078
----------------------------------------------------
max_depth : 3
n_estimators : 100
Train Score : 0.8158232725404573
Train Score : 0.7084148727984344
----------------------------------------------------
max_depth : 3
n_estimators : 150
Train Score : 0.8291805805291549
Train Score : 0.7201565557729941
----------------------------------------------------
max_depth : 3
n_estimators : 200
Train Score : 0.8391985615206782
Train Score : 0.7348336594911937
----------------------------------------------------
max_depth : 3
n_estimators : 250
Train Score : 0.8493449781659389
Train Score : 0.7416829745596869
----------------------------------------------------
max_depth : 3
n_estimators : 300
Train Score : 0.851785255586951
Train Score : 0.7475538160469667
----------------------------------------------------
max_depth : 3
n_estimators : 350
Train Score : 0.8594913948111996
Train Score : 0.7534246575342466
----------------------------------------------------
max_depth : 3
n_estimators : 400
Train Score : 0.8688671975340354
Train Score : 0.7622309197651663
----------------------------------------------------
max_depth : 4
n_estimators : 50
Train Score : 0.8217313126123812
Train Score : 0.7328767123287672
----------------------------------------------------
max_depth : 4
n_estimators : 100
Train Score : 0.8556383251990752
Train Score : 0.7553816046966731
----------------------------------------------------
max_depth : 4
n_estimators : 150
Train Score : 0.8835088620601079
Train Score : 0.7661448140900196
----------------------------------------------------
max_depth : 4
n_estimators : 200
Train Score : 0.8937837143591061
Train Score : 0.7827788649706457
----------------------------------------------------
max_depth : 4
n_estimators : 250
Train Score : 0.9072694580015412
Train Score : 0.7945205479452054
----------------------------------------------------
max_depth : 4
n_estimators : 300
Train Score : 0.9170305676855895
Train Score : 0.7974559686888454
----------------------------------------------------
max_depth : 4
n_estimators : 350
Train Score : 0.9235807860262009
Train Score : 0.8072407045009785
----------------------------------------------------
max_depth : 4
n_estimators : 400
Train Score : 0.9312869252504495
Train Score : 0.812133072407045
----------------------------------------------------
max_depth : 5
n_estimators : 50
Train Score : 0.8751605445671718
Train Score : 0.7720156555772995
----------------------------------------------------
max_depth : 5
n_estimators : 100
Train Score : 0.9140765476496275
Train Score : 0.8043052837573386
----------------------------------------------------
max_depth : 5
n_estimators : 150
Train Score : 0.9287182121756999
Train Score : 0.8209393346379648
----------------------------------------------------
max_depth : 5
n_estimators : 200
Train Score : 0.9407911636270229
Train Score : 0.8228962818003914
----------------------------------------------------
max_depth : 5
n_estimators : 250
Train Score : 0.9501669663498588
Train Score : 0.8424657534246576
----------------------------------------------------
max_depth : 5
n_estimators : 300
Train Score : 0.9546622142306704
Train Score : 0.8512720156555773
----------------------------------------------------
max_depth : 5
n_estimators : 350
Train Score : 0.9609555612638069
Train Score : 0.8551859099804305
----------------------------------------------------
max_depth : 5
n_estimators : 400
Train Score : 0.9642948882609812
Train Score : 0.860078277886497
----------------------------------------------------
max_depth : 6
n_estimators : 50
Train Score : 0.9081685075777036
Train Score : 0.7974559686888454
----------------------------------------------------
max_depth : 6
n_estimators : 100
Train Score : 0.9424608271256101
Train Score : 0.837573385518591
----------------------------------------------------
max_depth : 6
n_estimators : 150
Train Score : 0.9605702543025944
Train Score : 0.8610567514677103
----------------------------------------------------
max_depth : 6
n_estimators : 200
Train Score : 0.9678910865656306
Train Score : 0.8767123287671232
----------------------------------------------------
max_depth : 6
n_estimators : 250
Train Score : 0.9698176213716928
Train Score : 0.8806262230919765
----------------------------------------------------
max_depth : 6
n_estimators : 300
Train Score : 0.9708451066015926
Train Score : 0.8864970645792564
----------------------------------------------------
max_depth : 6
n_estimators : 350
Train Score : 0.9722578987927049
Train Score : 0.8884540117416829
----------------------------------------------------
max_depth : 6
n_estimators : 400
Train Score : 0.9727716414076548
Train Score : 0.8913894324853229
----------------------------------------------------
max_depth : 7
n_estimators : 50
Train Score : 0.9378371435910609
Train Score : 0.8228962818003914
----------------------------------------------------
max_depth : 7
n_estimators : 100
Train Score : 0.9681479578731056
Train Score : 0.87279843444227
----------------------------------------------------
max_depth : 7
n_estimators : 150
Train Score : 0.9713588492165425
Train Score : 0.8825831702544031
----------------------------------------------------
max_depth : 7
n_estimators : 200
Train Score : 0.9727716414076548
Train Score : 0.8933463796477495
----------------------------------------------------
max_depth : 7
n_estimators : 250
Train Score : 0.9735422553300797
Train Score : 0.8943248532289628
----------------------------------------------------
max_depth : 7
n_estimators : 300
Train Score : 0.9735422553300797
Train Score : 0.8913894324853229
----------------------------------------------------
max_depth : 7
n_estimators : 350
Train Score : 0.9735422553300797
Train Score : 0.8923679060665362
----------------------------------------------------
max_depth : 7
n_estimators : 400
Train Score : 0.9735422553300797
Train Score : 0.8933463796477495
----------------------------------------------------
```

In [319]:
```
best_params
```
Out[319]:
```
{'max_depth': 7, 'n_estimators': 250}
```

In [320]:
```
score
```
Out[320]:
```
0.8943248532289628
```

In [321]:
```
import pickle
import os
```

In [322]:
```
scaler_path = os.path.join('' , 'classifier.pkl')
with open(scaler_path , 'wb') as sf:
    pickle.dump(rfc , sf)
```

In [ ]:

```
import pandas as pd
import numpy as np
import pickle
import streamlit as st
from PIL import Image

pickle_in = open('classifier.pkl', 'rb')
classifier = pickle.load(pickle_in)

def welcome():
    return 'welcome all'

def prediction(gender, age1, hypertension, heart_disease,married,work_type1,residence,avg_glucose1,bmi1,smoking_stat1):
    age = [0,0,0,0]
    work_type = [0,0,0,0,0]
    glucose_level = [0,0,0,0]
    bmi = [0,0,0,0]
    smoking = [0,0,0,0]
    if gender=="Male":
        gender=0
    else:
        gender=1

    if hypertension=="Yes":
        hypertension=1
    else:
        hypertension=0

    if heart_disease=="Yes":
        heart_disease=1
    else:
        heart_disease=0

    if married=="Yes":
        married=1
    else:
        married=0

    if residence=="Rural":
        residence=1
    else:
        residence=0

    if age1>=20 and age1<40:
        age[0]=1
    elif age1>=40 and age1<60:
        age[1]=1
    elif age1>=60:
        age[2]=1
    else:
        age[3]=1

    if avg_glucose1<77:
        glucose_level[3]=1
    elif avg_glucose1<91:
        glucose_level[0]=1
    elif avg_glucose1<114:
        glucose_level[1]=1
    else:
        glucose_level[2]=1

    if work_type1=="Govt job":
        work_type[0]=1
    elif work_type1=="Never worked":
        work_type[1]=1
    elif work_type1=="Private":
        work_type[2]=1
    elif work_type1=="Self-employed":
        work_type[3]=1
    else:
        work_type[4]=1

    if bmi1<23:
        bmi[3]=1
    elif bmi1<28:
        bmi[0]=1
    elif bmi1<33:
        bmi[1]=1
    else:
        bmi[2]=1

    if smoking_stat1=="Unknown":
        smoking[0]=1
    elif smoking_stat1=="Formerly smoked":
        smoking[1]=1
    elif smoking_stat1=="Never smoked":
        smoking[2]=1
    else:
```

```
        smoking[3]=1

    prediction = classifier.predict(

[[gender,hypertension,heart_disease,married,residence,age[0],age[1],age[2],age[3],glucose_level[0],glucose_level[1],glucose_level[2],glucose_level[3],

        work_type[0],work_type[1],work_type[2],work_type[3],work_type[4],bmi[0],bmi[1],bmi[2],bmi[3],smoking[0],smoking[1],smoking[2],smoking[3]]])
    #print(heart_disease)
    return prediction

def main():
        st.title("Heart ♥ Stroke Prediction App")
        html_temp=""
        ans=0
        st.markdown(html_temp,unsafe_allow_html = True)
        get_Gender = st.sidebar.radio("Select your gender",("Male","Female"))
        get_Hypertension = st.sidebar.radio("Do you have hypertension?",("Yes","No"))
        get_heartDisease = st.sidebar.radio("Do you have heart disease?",("Yes","No"))
        get_married = st.sidebar.radio("Have you ever been married?",("Yes","No"))
        get_residence = st.selectbox("Select your residence type",("Rural","Urban"))
        get_age = st.slider("How old are you?",value=25)
        get_workType = st.selectbox("Select the type of work you do.",("Govt job","Never worked","Private","Self-employed","Children"))
        get_bmi = st.slider("How much is your bmi?",min_value=10,max_value=100,value=65)
        get_glucose = st.slider("How much is your glucose level?",min_value=55,max_value=272,value=50)
        get_smoke = st.selectbox("What's your smoking history",("Unknown","Formerly smoked","Never smoked","Smokes"))

        if st.button("Predict"):

ans=prediction(get_Gender,get_age,get_Hypertension,get_heartDisease,get_married,get_workType,get_residence,get_glucose,get_bmi,get_smoke)[0]
                if ans==0:
                        st.success('You have no chance of getting stroke😊')
                        st.image('images/happy_heart.jfif')
                else:
                        st.success('You are at risk of getting stroke😢')
                        st.image('images/damaged_heart.jfif')

if __name__=='__main__':
    main()
```