BRAIN STROKE PREDICTION USING MACHINE LEARNING

Anoushka Dighe
Computer Engineering
Shah and Anchor Kutchhi Engineering college
BE3-16
anoushka.dighe15528@sakec.ac.in

Ashpak Sheik
Computer Engineering
Shah and Anchor Kutchhi Engineering college
BE4-27
ashpak.shaik15951@sakec.ac.in

Umang Bid
Computer Engineering
Shah and Anchor Kutchhi Engineering college
BE3-7
umangbid28@gmail.com

***Abstract* — Stroke is one of the most serious diseases worldwide, directly or indirectly responsible for a significant number of deaths. Various data mining techniques are used in the healthcare industry to aid in the diagnosis and early detection of diseases. Current research considers several elements that lead to stroke. First, we examine the characteristics of those who suffer a stroke more often than others.The Dataset is from a freely available source and various classification algorithms are used to predict the onset of a stroke shortly. Using the Naïve Bays and Decision Tree, it was possible to achieve an accurate percent. Using various statistical techniques and principal component analysis, we identify the most important factors for stroke prediction. We conclude that age, heart disease, average glucose level, and hypertension are the most important factors for detecting stroke in patients.**

Key Words: Machine learning, Stroke, Classification, Supervised Learning, Data Mining

Outcomes: Our project titled "BRAIN STROKE PREDICTION USING MACHINE LEARNING" is mapped with the following outcomes

I. Introduction

Stroke is an ailment that impacts vessels that supply blood to the thoughts. A mind stroke takes a region which lists blood glides to the mind and is each reduced or interrupted. whilst this occurs, the mind no longer gets sufficient oxygen or other crucial components, and the brain cells start to die. A stroke affects an important lengthy-time period of incapacity or demise. Mind stroke is one of the leading causes of death all around the world. There are 3 kinds of brain strokes: ischemic strokes, hemorrhagic strokes, and transient ischemic assault (TIA), which is also referred to as a caution or mini-stroke . Ischemic strokes arise due to loss of blood supply, and hemorrhagic strokes occur because of ruptured blood vessels. The most typical kind of ISCHEMIC STROKE stroke is this one. It occurs when the blood arteries in the brain narrow or block, significantly reducing the amount of blood flow (ischemia). Fat Deposits that accumulate in blood vessels or blood clots or other debris that move through the bloodstream, typically from the heart, and lodge in the blood vessels in the brain cause blocked or restricted blood arteries. Brain bleeding results in a hemorrhagic stroke. This may occur when a brain blood artery rupture or when bleeding occurs in the brain tissue. Pressure brought on by bleeding, edema, or a lack of blood flow can all contribute to hemorrhagic stroke damage. An ischemic stroke, which is a stroke brought on by a stopped for the year 2002 was estimated to be as high as $49.4 billion in the United States ofAmerica (USA), while costs Blood supply can result in bleeding in the brain tissue. As a result, the brain's tissue is harmed. Transient ischemic attack, or TIA for short, is a dangerous repercussion. A TIA causes a temporary interruption in the blood supply to a portion of the brain. Another name for it is a "ministroke," but don't be deceived by the diminutive. A TIA may be a precursor to a full blown stroke.The most common cause of TIAs is a blood clot that becomes stuck in an artery that carries blood to the brain. Your brain is oxygen-starved and unable to function normally if there isn't regular blood flow

II Project Overview :

Stroke is one of the most serious diseases worldwide, directly or indirectly responsible for a significant number of deaths. Various data mining techniques are used in the healthcare industry to aid in the diagnosis and early detection of diseases. Current research considers several elements that lead to stroke. First, we examine the characteristics of those who suffer a stroke more often than others. The dataset is from a freely available source and various classification algorithms are used to predict the onset of a stroke shortly. Using various statistical techniques and principal component analysis, we identify the most important factors for stroke prediction. We conclude that age, heart disease, average glucose level, and hypertension are the most important factors for detecting stroke in patients. Furthermore, to provide the highest accuracy rate and lowest miss rate compared to using all available input features and other benchmarking algorithms.

III Research

Burden of Stroke in the World

Stroke is the second leading cause of death and leading cause of adult disability worldwide with 400-800 strokes per 100,000, 15 million new acute strokes every year, 28,500,000 disability adjusted life-years and 28-30-day case fatality ranging from 17% to 35%. The burden of stroke will likely worsen with stroke and heart disease related deaths projected to increase to five million in 2020, compared to three million in 1998. This will be a result of continuing health and demographic transition resulting in increase in vascular disease risk factors and population of the elderly. Developing countries account for 85% of the global deaths from stroke. The social and economic consequences of stroke are substantial. The cost of stroke after discharge was estimated to amount to 2.9 billion Euros in France.

Burden of Stroke in Uganda

The actual burden of stroke in Uganda is not known. According to WHO estimates for heart disease and stroke 2002, stroke was responsible for 11 per 1000 population (25,004,000) 4 disability adjusted life years and mortality of 11,043. Stroke is one of the common neurological diseases among patients admitted to the neurology ward at Mulago, Uganda's national referral hospital accounting for 21% of all neurological admissions. Unpublished research done at Mulago hospital, showed a 30-day case fatality of 43.8% among 133 patients admitted with stroke. The economic burden caused by stroke has not been explored in Uganda but given the very high dependent population (53%), high prevalence of HIV/AIDS, drug resistant TB and Malaria, the impact of stroke and other emerging non-communicable diseases on the resource limited economy is astronomical.

Stroke Prediction Using SVM

In this paper we were using a Support Vector Machine for stroke prediction. This research work investigates the various physiological parameters that are used as risk factors for the prediction of stroke.Data was collected from the International Stroke Trial database and was successfully trained and tested using Support Vector Machine (SVM). Machine learning algorithms have been proposed as important tools indecision making in the medical field. The objective of this work is to develop a machine learning based approach to predict the possibility of stroke in people having the symptoms or risk factors of stroke. In this work, we have implemented SVM with different kernel functions and found that the linear kernel gavean accuracy of 90 %. The results were evaluated on a spectrum of patients of different age groups.

Stroke Prediction using Artificial Intelligence

The stroke deprives a person's brain of oxygen and nutrients, which can cause brain cells to die. Numerous works have been carried out for predicting various diseases by comparing the performance of predictive data mining technologies. In this work, we compare different methods with our approach for stroke prediction on the Cardiovascular Health Study (CHS) dataset. Here, decision tree algorithm is used for feature selection process, principal component analysis algorithm is used for reducing the dimension and adopted back propagation neural network classification algorithm, to construct a classification model.The proposed method use Decision Tree algorithm for feature selection method, PCA for dimension reduction and ANN for the classification. The experimental results show that the proposed method has higher performance than other related well-known methods.

IV Problem Statement

Damage to the brain from interruption of its blood supply. A stroke is a medical emergency. Symptoms of stroke include trouble walking, speaking and understanding, as well as paralysis or numbness of the face, arm or leg. Early treatment with medication like tPA (clot buster) can minimize brain damage. Other treatments focus on limiting complications and preventing additional strokes.

V Layout :

Layout of the model website:

```
### Layout

```
├──images
├──Tabs
│   └──__pycache__
│   └── home.py
│   └── data.py
│   └── predict.py
│   └── visualize.py
└──__pycache__
└── main.py
└── BS.csv
└── web_functions.py
└── requirements.txt
└── Procfile
└── setup.sh
```
```

VI Data information

Brain stroke detection:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	stroke
0	1	3	0	0	0	0	0	95.12	18	0
1	1	58	1	0	1	1	1	87.96	39.2	0
2	0	8	0	0	0	1	1	110.89	17.6	0
3	0	70	0	0	1	1	0	69.04	35.9	0
4	1	14	0	0	0	4	0	161.28	19.1	0
5	0	47	0	0	1	1	1	210.95	50.1	0
6	0	52	0	0	1	1	1	77.59	17.7	0
7	0	75	0	1	1	3	0	243.53	27	0
8	0	32	0	0	1	1	0	77.67	32.3	0
9	0	74	1	0	1	3	1	205.84	54.6	0

Column Names

	0
0	gender
1	age
2	hypertension
3	heart_disease
4	ever_married
5	work_type
6	Residence_type
7	avg_glucose_level
8	bmi
9	stroke

Columns data types

	0
gender	int64
age	float64
hypertension	int64
heart_disease	int64
ever_married	int64
work_type	int64
Residence_type	int64
avg_glucose_level	float64
bmi	float64
stroke	int64

VII : Analysis of Data :

```
import pandas as pd
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
%matplotlib inline
stroke =
pd.read_csv(r'healthcare-dataset-stroke-data.csv')
stroke_rep =
stroke['smoking_status'].replace('Unknown',np.nan)
```

This line of code creates a new Series named 'stroke_rep.' It selects the 'smoking_status' column from the 'stroke' DataFrame and then uses the .replace() method to replace any occurrences of the string 'Unknown' with np.nan in this specific column. The resulting Series is stored in the 'stroke_rep' variable.

```
stroke.isnull().sum()
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status    1544
stroke               0
dtype: int64
```

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

```
0    4861
1     249
Name: stroke, dtype: int64
```

```
stroke_neg = stroke[stroke['stroke'] == 0]
stroke_neg.isnull().sum()
```

0: There are 4,861 occurrences of the value 0 in the 'stroke' column.

1: There are 249 occurrences of the value 1 in the 'stroke' column

```
stroke_pos = stroke[stroke['stroke'] ==1]
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
..	..	..	..	..	..	..	..	..	..	..	..	..
244	17739	Male	57.0	0	0	Yes	Private	Rural	84.96	36.7	NaN	1
245	49669	Female	14.0	0	0	No	children	Rural	57.93	30.9	NaN	1
246	27153	Female	75.0	0	0	Yes	Self-employed	Rural	78.80	29.3	formerly smoked	1
247	34060	Male	71.0	1	0	Yes	Self-employed	Rural	87.80	NaN	NaN	1
248	43424	Female	78.0	0	0	Yes	Private	Rural	78.81	19.6	NaN	1

249 rows × 12 columns

```
stroke_pos.isnull().sum()
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                 40
smoking_status      47
stroke               0
dtype: int64
```

id, gender, age, hypertension, heart_disease, ever_married, work_type, Residence_type, avg_glucose_level, and stroke columns: There are no missing values in these columns because they all have a count of 0.

bmi column: There are 161 missing values in the 'bmi' column. This means that there are 161 rows in the DataFrame where the 'bmi' data is missing or undefined.

smoking_status column: There are 1,497 missing values in the 'smoking_status' column. This indicates that there are 1,497 rows in the DataFrame where the 'smoking_status' data is missing or undefined.

```
id                    0
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
bmi                 161
smoking_status     1497
stroke                0
dtype: int64
```

stroke_neg.isnull().sum(): This code is used to count the number of missing (null) values in each column of the 'stroke_neg' DataFrame. It displays the count of missing values for each column in the 'stroke_neg' DataFrame. The output will show how many missing values exist in each column.

stroke_neg = stroke[stroke['stroke'] == 0]: This code is used to filter the original DataFrame 'stroke' and create a new DataFrame 'stroke_neg' containing only the rows where the 'stroke' column has a value of 0, meaning the individuals in this DataFrame did not have a stroke.

```
test_df = stroke[stroke['smoking_status'].isnull() & stroke['bmi'].isnull()]
test_df
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
8	27419	Female	59.00	0	0	Yes	Private	Rural	76.15	NaN	NaN	1
13	8213	Male	78.00	0	1	Yes	Private	Urban	219.84	NaN	NaN	1
19	25226	Male	57.00	0	1	No	Govt_job	Urban	217.08	NaN	NaN	1
27	61843	Male	58.00	0	0	Yes	Private	Rural	189.84	NaN	NaN	1
46	37937	Female	75.00	0	1	No	Self-employed	Urban	109.78	NaN	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...	...
4202	23194	Male	32.00	1	0	No	Private	Rural	74.43	NaN	NaN	0
4286	23339	Male	3.00	0	0	No	children	Rural	194.75	NaN	NaN	0
4616	5774	Male	59.00	0	0	Yes	Private	Urban	223.16	NaN	NaN	0
4713	2044	Female	70.00	0	1	Yes	Self-employed	Rural	65.68	NaN	NaN	0
4934	32147	Male	1.32	0	0	No	children	Rural	107.02	NaN	NaN	0

61 rows × 12 columns

stroke['smoking_status'].isnull(): This condition checks for rows where the 'smoking_status' column is null (missing values).

stroke['bmi'].isnull(): This condition checks for rows where the 'bmi' column is null (missing values).

stroke['smoking_status'].isnull() & stroke['bmi'].isnull():

```
stroke_neg_both_null = stroke_neg[stroke_neg['smoking_status'
```

The & operator combines the two conditions using a logical AND operation. It selects rows where both conditions are true, meaning the 'smoking_status' and 'bmi' columns are both missing.

test_df = stroke[...]: This line filters the 'stroke' DataFrame based on the combined condition and creates the 'test_df' DataFrame, which includes only the rows where both 'smoking_status' and 'bmi' have missing values.

```
stroke_pos_both_null = stroke_pos[stroke_pos['smoking_status'].isnull() & stroke_pos['bmi'].isnull()]
stroke_pos_both_null
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
8	27419	Female	59.00	0	0	Yes	Private	Rural	76.15	NaN	NaN	1
13	8213	Male	78.00	0	1	Yes	Private	Urban	219.84	NaN	NaN	1
19	25226	Male	57.00	0	1	No	Govt_job	Urban	217.08	NaN	NaN	1
27	61843	Male	58.00	0	0	Yes	Private	Rural	189.84	NaN	NaN	1
46	37937	Female	75.00	0	1	No	Self-employed	Urban	109.78	NaN	NaN	1
50	18587	Female	76.00	0	0	No	Private	Urban	89.96	NaN	NaN	1
64	7356	Male	75.00	0	0	Yes	Private	Urban	104.72	NaN	NaN	1
81	26015	Female	66.00	0	0	Yes	Self-employed	Urban	101.45	NaN	NaN	1
84	70042	Male	58.00	0	0	Yes	Private	Urban	71.20	NaN	NaN	1
124	14164	Female	72.00	0	0	Yes	Private	Urban	219.91	NaN	NaN	1
129	48796	Female	75.00	0	0	Yes	Govt_job	Urban	62.48	NaN	NaN	1
150	11933	Female	79.00	0	0	Yes	Private	Rural	169.67	NaN	NaN	1
162	69768	Female	1.32	0	0	No	children	Urban	70.37	NaN	NaN	1
178	33486	Female	80.00	0	0	Yes	Govt_job	Urban	110.66	NaN	NaN	1
183	8003	Female	77.00	0	0	No	Private	Urban	81.32	NaN	NaN	1
189	66955	Male	61.00	0	1	Yes	Private	Urban	209.86	NaN	NaN	1
200	54695	Male	74.00	0	0	Yes	Private	Urban	167.13	NaN	NaN	1
247	34060	Male	71.00	1	0	Yes	Self-employed	Rural	87.80	NaN	NaN	1

stroke_pos['smoking_status'].isnull(): This condition checks for rows in the 'smoking_status' column of the 'stroke_pos' DataFrame where the values are null (missing).

stroke_pos['bmi'].isnull(): This condition checks for rows in the 'bmi' column of the 'stroke_pos' DataFrame where the values are null (missing).

stroke_pos['smoking_status'].isnull() & stroke_pos['bmi'].isnull(): The & operator combines the two conditions using a logical AND operation. It selects rows where both conditions are true, meaning the 'smoking_status' and 'bmi' columns are both missing in the 'stroke_pos' DataFrame.

stroke_pos_both_null = stroke_pos[...]: This line filters the 'stroke_pos' DataFrame based on the combined condition and creates the 'stroke_pos_both_null' DataFrame, which contains only the rows where both 'smoking_status' and 'bmi' have missing values among the cases in the 'stroke_pos' DataFrame.

```
len(stroke_pos_both_null) : 18
```

```
].isnull()                                    &
stroke_neg['bmi'].isnull()]
stroke_neg_both_null
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
477	22470	Male	61.00	0	0	Yes	Govt_job	Urban	184.15	NaN	NaN	0
742	33723	Female	9.00	0	0	No	children	Urban	95.81	NaN	NaN	0
1115	809	Male	13.00	0	0	No	children	Urban	71.73	NaN	NaN	0
1183	13602	Male	73.00	1	0	Yes	Self-employed	Rural	102.06	NaN	NaN	0
1194	542	Female	3.00	0	0	No	children	Urban	79.63	NaN	NaN	0
1277	33187	Female	6.00	0	0	No	children	Urban	201.25	NaN	NaN	0
1300	37327	Female	71.00	0	0	Yes	Private	Urban	214.77	NaN	NaN	0
1324	132	Female	80.00	0	0	Yes	Govt_job	Urban	84.86	NaN	NaN	0
1471	937	Male	7.00	0	0	No	children	Urban	87.94	NaN	NaN	0
1596	16593	Male	47.00	0	0	No	Private	Rural	237.17	NaN	NaN	0
1640	17752	Male	76.00	0	1	Yes	Private	Urban	79.05	NaN	NaN	0
1669	1842	Male	58.00	0	0	Yes	Private	Urban	94.00	NaN	NaN	0
1670	34720	Male	45.00	0	1	Yes	Private	Rural	93.77	NaN	NaN	0
1816	9170	Male	60.00	0	0	Yes	Self-employed	Urban	185.71	NaN	NaN	0
1894	2019	Male	20.00	0	0	No	Private	Rural	70.96	NaN	NaN	0
1993	27163	Female	60.00	1	0	Yes	Private	Urban	109.00	NaN	NaN	0
2030	38920	Male	0.48	0	0	No	children	Urban	73.02	NaN	NaN	0
2215	19439	Male	67.00	0	1	Yes	Private	Rural	97.24	NaN	NaN	0
2321	1077	Male	77.00	0	1	Yes	Govt_job	Rural	106.03	NaN	NaN	0
2477	24096	Female	34.00	1	0	Yes	Self-employed	Urban	100.61	NaN	NaN	0
2532	20112	Male	79.00	0	1	Yes	Private	Urban	213.38	NaN	NaN	0
2739	2538	Female	5.00	0	0	No	children	Rural	105.18	NaN	NaN	0
2768	2885	Male	72.00	1	0	Yes	Private	Rural	231.71	NaN	NaN	0
2867	37526	Female	68.00	1	1	Yes	Private	Rural	233.30	NaN	NaN	0
3059	121	Female	38.00	0	0	Yes	Private	Urban	91.44	NaN	NaN	0
3135	33704	Male	44.00	1	0	Yes	Private	Rural	84.10	NaN	NaN	0
3161	51554	Male	42.00	0	0	Yes	Private	Urban	177.91	NaN	NaN	0
3162	2296	Male	78.00	1	0	Yes	Self-employed	Urban	90.19	NaN	NaN	0
3197	37213	Male	60.00	0	0	Yes	Self-employed	Rural	212.02	NaN	NaN	0

stroke_neg['smoking_status'].isnull(): This condition checks for rows in the 'smoking_status' column of the 'stroke_neg' DataFrame where the values are null (missing).

stroke_neg['bmi'].isnull(): This condition checks for rows in the 'bmi' column of the 'stroke_neg' DataFrame where the values are null (missing).

stroke_neg['smoking_status'].isnull() & stroke_neg['bmi'].isnull(): The & operator combines the two conditions using a logical AND operation. It selects rows where both conditions are true, meaning the 'smoking_status' and 'bmi' columns are both missing in the 'stroke_neg' DataFrame.

stroke_neg_both_null = stroke_neg[...]: This line filters the 'stroke_neg' DataFrame based on the combined condition and creates the 'stroke_neg_both_null' DataFrame, which contains only the rows where both 'smoking_status' and 'bmi' have missing values among the cases in the 'stroke_neg' DataFrame.

```
len(stroke_neg_both_null): 43
```

```
stroke['work_type'].unique()
```

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

```
stroke_male  =  stroke[stroke['gender']
== 'Male']
stroke_male
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
6	53882	Male	74.0	1	1	Yes	Private	Rural	70.09	27.4	never smoked	1
13	8213	Male	78.0	0	1	Yes	Private	Urban	219.84	NaN	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5097	64520	Male	68.0	0	0	Yes	Self-employed	Urban	91.68	40.8	NaN	0
5098	579	Male	9.0	0	0	No	children	Urban	71.88	17.5	NaN	0
5099	7293	Male	40.0	0	0	Yes	Private	Rural	83.94	NaN	smokes	0
5100	68398	Male	82.0	1	0	Yes	Self-employed	Rural	71.97	28.3	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0

2115 rows × 12 columns

```
stroke_female                                 =
stroke[stroke['gender'] == 'Female']
stroke_female
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
7	10434	Female	69.0	0	0	No	Private	Urban	94.39	22.8	never smoked	1
8	27419	Female	59.0	0	0	Yes	Private	Rural	76.15	NaN	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5104	14180	Female	13.0	0	0	No	children	Rural	103.08	18.6	NaN	0
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	NaN	0

2994 rows × 12 columns

```
stroke_female.isnull().sum()
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                 97
smoking_status     836
stroke               0
dtype: int64
```

```
stroke_female.bmi.mean():
29.065757680358992
```

```
stroke_female['bmi'].fillna(stroke_fem
ale['bmi'].mean(),inplace=True)
stroke_female.isnull().sum()
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status     836
stroke               0
dtype: int64
```

stroke_female['bmi'].fillna(stroke_female['bmi'].mean(), inplace=True): This line is filling missing (null) values in the 'bmi' column of the 'stroke_female' DataFrame with the mean value of the 'bmi' column. The fillna() method is used for this purpose. The inplace=True argument means that the operation is performed directly on the DataFrame 'stroke_female' without the need to assign the result to a new variable.

stroke_female.isnull().sum(): After filling the missing values, this line is used to check and count the remaining missing values in each column of the 'stroke_female' DataFrame.

```
stroke                    =
stroke.drop(stroke[stroke['gender']  ==
'male'],axis=1)
stroke                    =
pd.concat([stroke_female,stroke_male])
stroke
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	29.065758	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.400000	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.000000	never smoked	1
7	10434	Female	69.0	0	0	No	Private	Urban	94.39	22.800000	never smoked	1
8	27419	Female	59.0	0	0	Yes	Private	Rural	76.15	29.065758	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5097	64520	Male	68.0	0	0	Yes	Self-employed	Urban	91.68	40.800000	NaN	0
5098	579	Male	9.0	0	0	No	children	Urban	71.88	17.500000	NaN	0
5099	7293	Male	40.0	0	0	Yes	Private	Rural	83.94	28.647936	smokes	0
5100	68398	Male	82.0	1	0	Yes	Self-employed	Rural	71.97	28.300000	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.600000	formerly smoked	0

5109 rows × 12 columns

stroke = stroke.drop(stroke[stroke['gender'] == 'male'], axis=1): This line attempts to drop columns (axis=1) where the 'gender' column has the value 'male.' However, there is an issue in the code, and it is trying to drop instead of rows. To drop rows based on a condition, you should specify axis=0 rather than axis=1.

stroke = pd.concat([stroke_female, stroke_male]): This line attempts to concatenate two DataFrames, 'stroke_female' and 'stroke_male, and assigns the result to 'stroke'. This operation combines the two DataFrames vertically (along rows) to create a new DataFrame.

Finally, you set 'stroke' to be the result of concatenating 'stroke_female' and 'stroke_male', so it will contain the combined data from both DataFrames.

```
storke                    =
stroke.drop('smoking_status',axis=1,in
place =True)
stroke.columns
```

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

```
stroke                    =
pd.concat([stroke_male,stroke_female])
stroke
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.600000	formerly smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.500000	never smoked	1
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.000000	formerly smoked	1
6	53882	Male	74.0	1	1	Yes	Private	Rural	70.09	27.400000	never smoked	1
13	8213	Male	78.0	0	1	Yes	Private	Urban	219.84	28.647936	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5104	14180	Female	13.0	0	0	No	children	Rural	103.08	18.600000	NaN	0
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	29.065758	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.000000	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.600000	never smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.200000	NaN	0

5109 rows × 12 columns

```
stroke['age']                    =
stroke['age'].astype(int)
stroke.dtypes
```

columns

```
stroke.drop('smoking_status',axis=1,inplace=True)
stroke.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	stroke
0	9046	Male	67	0	1	Yes	Private	Urban	228.69	36.600000	1
2	31112	Male	80	0	1	Yes	Private	Rural	105.92	32.500000	1
5	56669	Male	81	0	0	Yes	Private	Urban	186.21	29.000000	1
6	53882	Male	74	1	1	Yes	Private	Rural	70.09	27.400000	1
13	8213	Male	78	0	1	Yes	Private	Urban	219.84	28.647936	1

```
stroke['gender'] = stroke['gender'].map({'Male':0,'Female':1})
stroke
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	stroke
0	9046	0	67	0	1	Yes	Private	Urban	228.69	36.600000	1
2	31112	0	80	0	1	Yes	Private	Rural	105.92	32.500000	1
5	56669	0	81	0	0	Yes	Private	Urban	186.21	29.000000	1
6	53882	0	74	1	1	Yes	Private	Rural	70.09	27.400000	1
13	8213	0	78	0	1	Yes	Private	Urban	219.84	28.647936	1
...	...	...	...	...	...	...	...	...	...	...	...
5104	14180	1	13	0	0	No	children	Rural	103.08	18.600000	0
5105	18234	1	80	1	0	Yes	Private	Urban	83.75	29.065758	0
5106	44873	1	81	0	0	Yes	Self-employed	Urban	125.20	40.000000	0
5107	19723	1	35	0	0	Yes	Self-employed	Rural	82.99	30.600000	0
5109	44679	1	44	0	0	Yes	Govt_job	Urban	85.28	26.200000	0

5109 rows × 11 columns

stroke['gender'].map({'Male': 0, 'Female': 1}): This code takes the 'gender' column and uses the .map() function to replace values according to the provided mapping dictionary. In this case, it replaces 'Male' with 0 and 'Female' with 1.

The result of this operation is that the 'gender' column in the 'stroke' DataFrame will now contain numerical values (0 for 'Male' and 1 for 'Female'), making it more suitable for certain types of analysis or machine learning algorithms that require numerical input.

VIII Feature Engineering:

```
one_hot = OneHotEncoder()
work_type = one_hot.fit_transform(stroke.work_type.values.reshape(-1,1)).toarray()
```

```
id                   int64
gender              object
age                  int32
hypertension         int64
heart_disease        int64
ever_married        object
work_type           object
Residence_type      object
avg_glucose_level  float64
bmi                float64
smoking_status      object
stroke               int64
dtype: object
```

One-hot encoding is a technique used to convert categorical variables into binary vectors to be used in machine learning models. Here's what this code does:

OneHotEncoder(): This line creates an instance of the OneHotEncoder class.

stroke.work_type.values.reshape(-1, 1): This part extracts the 'work_type' column from the 'stroke' DataFrame and reshapes it into a two-dimensional array with a single column. The reshape(-1, 1) operation ensures that the data is in the right shape for one-hot encoding.

one_hot.fit_transform(...): This line fits the one-hot encoder to the reshaped 'work_type' data and transforms it into a one-hot encoded array.

.toarray(): The one-hot encoded result is converted to a NumPy array for further use.

```
work_type = stroke['work_type']
work_type
work_type_df = pd.DataFrame(work_type,columns=['work_type'])
work_type_df['work_type'].unique()
```

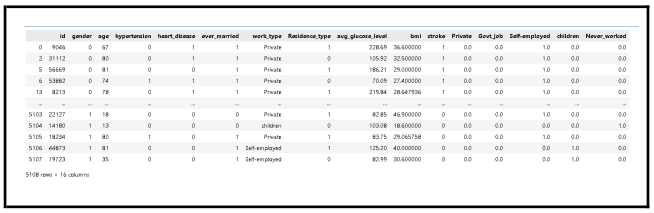
```
array([[0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       ...,
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.]])
```

```
encoded = pd.DataFrame(encoded,columns = ['Private', 'Govt_job', 'Self-employed', 'children', 'Never_worked'])
encoded
```

	Private	Govt_job	Self-employed	children	Never_worked
0	0.0	0.0	1.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0
...	...	...	...	...	...
5104	0.0	0.0	0.0	0.0	1.0
5105	0.0	0.0	1.0	0.0	0.0
5106	0.0	0.0	0.0	1.0	0.0
5107	0.0	0.0	0.0	1.0	0.0
5108	1.0	0.0	0.0	0.0	0.0

5109 rows × 5 columns

```
merged_stroke = pd.merge(left = stroke, right = encoded,left_index=True,right_index = True)
merged_stroke
```



IX EDA : Exploratory Data Analysis

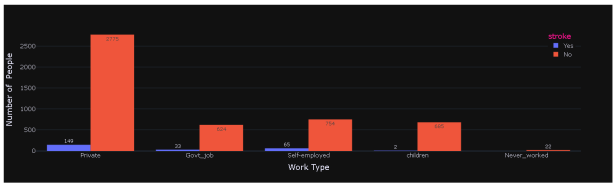
```
import plotly.express as px

fig = px.histogram(stroke,
                   x='work_type',
```

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


0             Private
2             Private
5             Private
6             Private
13            Private
             ...
5104         children
5105          Private
5106    Self-employed
5107    Self-employed
5109         Govt_job
Name: work_type, Length: 5109, dtype: object
```

```
template='plotly_dark')

fig.update_layout(yaxis_title="Number of  People")

fig.update_layout(legend=dict(yanchor="top",     y=0.99,     xanchor='right', x=0.99))

#
fig.update_layout(xaxis=dict(showgrid=False),
#
yaxis=dict(showgrid=False)
# )
fig.update_layout(font_color='lavender',

legend_title_font_color="deeppink",

yaxis=dict(tickfont=dict(size=15), titlefont=dict(size=20)),

xaxis=dict(tickfont=dict(size=15), titlefont=dict(size=20)),
```

```
                    barmode='group',
                    color='stroke',
                    text_auto=True,
                    labels={
                           'work_type':
'Work Type',
                          'count': 'No.
of People'
                    },
```

From the Below plot it can be interpreted that the people working in the private sector are having higher chances of getting a stroke, followed by people who are self-employed, as it clearly seen the number of people in the private sector having stroke is higher than the others compared to the people working in the different sectors.




```
import plotly.express as px

fig = px.box(stroke,
             x='stroke',
             y='age',
             color='gender',
             template='plotly_dark',
                labels={'age': 'Age of
the people'},

color_discrete_sequence=px.colors.qual
itative.Alphabet)

fig.update_layout(legend=dict(yanchor=
"top",     y=0.25,     xanchor='right',
x=0.55))
                                    legend_font_size=15)

# Changing the label names in the
legend -
newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda           t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.show()




yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),

xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
                    legend_font_size=14)

# Changing the label names in the
legend
newnames    =    {'0':    'Male',    '1':
'Female'}
fig.for_each_trace(lambda           t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.update_layout(legend=dict(font=dic
t(size=10), orientation="v"))
```

```
#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False))

fig.update_layout(font_color='lavender
',

legend_title_font_color="deeppink",

legend_title_font_size=15,
```


```
fig = px.box(stroke,
             x='stroke',
             y='bmi',
             color='gender',

color_discrete_sequence=px.colors.qual
itative.Alphabet,
             labels={'bmi': 'Body Mass
Index'},
             template='plotly_dark')

#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False))

fig.update_layout(font_color='white',

legend_title_font_color="deeppink",

legend_title_font_size=15,

yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
#
fig.update_layout(legend=dict(legend_t
itle_font_size=dict(size=15)))

fig.show()
```

From the below plot it can be interpreted that in both the gender chances of getting stroke increases with age and if looked specifically then in males aged above 70 years have higher chances of getting stroke, on the other hand females aged above 72 years have higher chances of getting stroke. And It can also be seen in the female category there are some extreme outliers where children aged 1 and 14 also getting stroke. For this outlier detection was carried out and the findings were that in both the cases BMI was way above the normal range at that age.

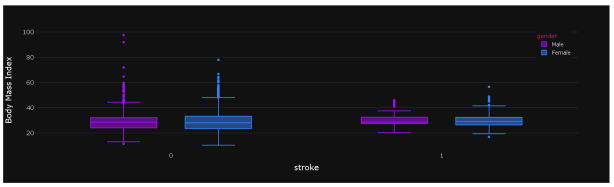
```
hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.show()
```

Facts:
1. As the BMI gets higher, the risk of high blood pressure, high cholesterol, and high blood sugar also rises. Each of these conditions puts you at a greater risk of stroke.
2.mild strokes occurred more often in men with BMIs of 30 or greater, while fatal strokes occurred more often in men with BMIs lower than 23.

```
xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
                  legend_font_size=14)

fig.update_layout(legend=dict(yanchor=
"top",     y=0.99,     xanchor='right',
x=0.99))

newnames    =    {'0':    'Male',    '1':
'Female'}
fig.for_each_trace(lambda            t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],
fig = px.box(stroke,
             x='stroke',
             y='avg_glucose_level',
             color='gender',

color_discrete_sequence=px.colors.qual
itative.Alphabet,

labels={'avg_glucose_level':  'Average
Glucose Level'},
             template='plotly_dark')

#   fig.for_each_annotation(lambda   a:
a.update(text=a.text.split('=')[-1]))
#   fig.for_each_annotation(lambda   a:
a.update(text=a.text.replace('0','Male
')))
#   fig.for_each_annotation(lambda   a:
a.update(text=a.text.replace('1','Fema
le')))

fig.update_layout(
    font_color='lavender',
    yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
    xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
)
```

3.In females the risk of having a stroke noticeably increases when BMI values exceed 27.

Insight -
From the above graph we can see that in males having stroke the median bmi is touching almost 29 which also justifies the scientific data that in men having bmi greater than or equal to 30 have a higher chance of getting stroke. And in females having stroke the median bmi is also touching 30 which is high as compared to normal range and which induces the chances of getting a stroke.

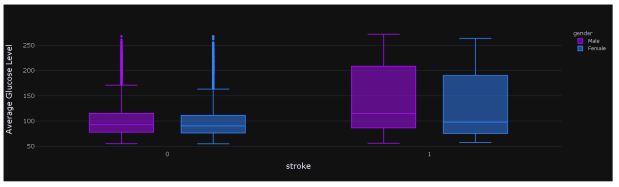



```
fig.show()
```

According to research elevated blood glucose level is an early sign of a possible stroke and going by numbers a person having average glucose level above 115mg/dL could be at a potential risk and the same can be seen from the figure , in males having stroke the median average glucose level goes above 115 and in females almost touching 100. It can also be seen that that there are a number of outliers where the glucose

```
newnames = {'0': 'Male', '1': 'Female'}
fig.for_each_trace(lambda t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False)
# )
stroke['stroke'] =
stroke['stroke'].astype('category')
fig = px.scatter(stroke,
                 x='age',

y='avg_glucose_level',
                 color='stroke',
                 text='age',
                 labels={
                         'age': 'Age of
the person',

'avg_glucose_level': 'Average Glucose
Level',
                              'gender=0':
'Male',
                              'gender=1':
'Female'
                 },

template='plotly_dark',
                 facet_col='gender')

fig.update_layout(
    font_color='lavender',
    yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
```

levels are extremely high but the people don't have stroke.




```
test = stroke[(stroke['avg_glucose_level'] > 165) & (stroke['bmi'] > 45) & (stroke['stroke'] == 0)]
test
```

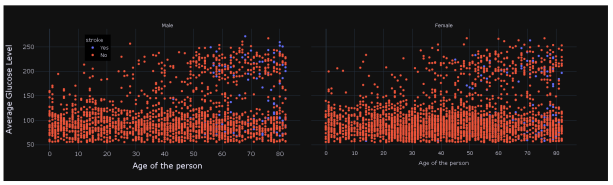
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	stroke
544	545	0	42	0	0	1	Private	0	210.48	71.9	0
1898	25405	0	62	0	0	1	Govt_job	1	187.52	57.7	0
2081	5355	0	63	0	0	1	Govt_job	0	231.69	56.1	0
2284	63990	0	52	1	0	1	Self-employed	0	192.37	49.2	0
2893	52164	0	29	0	0	1	Private	1	193.81	46.8	0
3060	32604	0	49	0	0	1	Self-employed	0	215.81	58.1	0
3606	14872	0	45	1	0	1	Self-employed	0	239.19	52.5	0
3688	38575	0	58	1	0	1	Self-employed	0	209.15	52.9	0
3780	29878	0	49	0	0	1	Private	1	175.74	45.4	0
3909	4077	0	49	0	0	1	Private	1	219.70	53.8	0
4176	10870	0	51	0	0	1	Private	0	232.64	45.2	0
4952	16245	0	51	1	0	1	Self-employed	0	211.83	56.6	0
254	32257	1	47	0	0	1	Private	1	210.95	50.1	0
258	28674	1	74	1	0	1	Self-employed	1	205.84	54.6	0
417	22320	1	37	0	0	1	Private	1	203.81	46.6	0
466	1307	1	61	1	0	1	Private	0	170.05	60.2	0
595	39639	1	46	0	0	1	Private	0	188.11	50.2	0
1061	8332	1	50	0	0	1	Private	0	206.25	53.4	0
1131	22363	1	47	0	0	1	Private	0	195.04	45.5	0
1257	5821	1	50	0	0	1	Private	0	217.39	50.6	0
1322	35913	1	55	1	0	1	Private	1	206.40	54.8	0
1529	34496	1	82	0	0	1	Private	1	253.16	47.5	0
1575	49495	1	18	0	0	0	Private	0	168.15	48.5	0
1782	11412	1	59	0	0	1	Private	0	234.82	51.8	0
2067	16938	1	40	0	0	1	Self-employed	0	212.97	49.8	0
2282	55566	1	34	0	0	1	Private	0	231.50	45.4	0
2290	61895	1	65	0	0	1	Private	0	220.47	48.7	0
2629	72915	1	45	0	0	1	Private	1	172.33	45.3	0

```
xanchor='right',
                                x=0.14,

bgcolor='black'))

fig.for_each_annotation(lambda a:
a.update(text=a.text.split('=')[-1]))
fig.for_each_annotation(lambda a:
a.update(text=a.text.replace('0',
'Male')))
fig.for_each_annotation(lambda a:
a.update(text=a.text.replace('1',
'Female')))
```

```
    xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
)

newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda            t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.update_layout(legend=dict(font=dic
t(size=12),

orientation="v",

yanchor="top",
                              y=0.95,
fig = px.scatter(stroke,
                 x='age',
                 y='bmi',
                 color='stroke',
                 text='age',
                 labels={
                         'age': 'Age of
the person',
                     'bmi': 'Body Mass
Index',
                             'gender=0':
'Male',
                             'gender=1':
'Female'
                 },

template='plotly_dark',
                 facet_col='gender',
                )

fig.update_layout(
    font_color='lavender',
    yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
```

```
#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False))

fig.show()
```

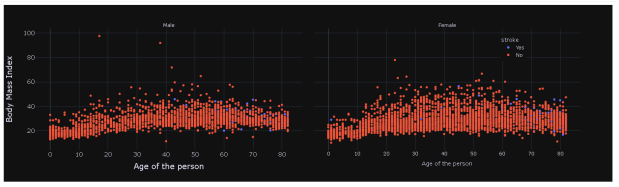
From the above figure it can be interpreted that in both males and females mostly the people who have got stroke are in the age group of 60-80 years. In both the genders there are certain people who have a normal average glucose level but still have a stroke. Going further into the possible reason of it then it can be seen that in the plot representing BMI,both in males and females those who have normal average glucose level but still have stroke thier BMI values are greater than the median BMI above which the chances of stroke increases.




```
fig.update_layout(legend=dict(
    font=dict(size=12),
    orientation="v",
    yanchor="top",
    y=0.95,
    xanchor='right',
    x=0.9,
))

#
fig.update_layout(xaxis=dict(showgrid=
False),
```

```
    xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
)

newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.for_each_annotation(lambda a:
a.update(text=a.text.split('=')[-1]))
fig.for_each_annotation(lambda a:
a.update(text=a.text.replace('0',
'Male')))
fig.for_each_annotation(lambda a:
a.update(text=a.text.replace('1',
'Female')))

fig = px.histogram(stroke,
                   x='ever_married',
                   barmode='group',
                   color='stroke',
                   text_auto=True,

labels={'ever_married': 'Marraige
Status'},

template='plotly_dark',
                   )

fig.update_layout(yaxis_title="Number
of  People")

#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False))

fig.update_layout(legend=dict(
#
yaxis=dict(showgrid=False))

fig.show()
```

From the above figure it can be observed that in males the people who have got a stroke almost everyone has their Body Mass Index above 27 to 27.5 but in females who have got a stroke there is variation in the values of Body Mass Index ranging all over and as seen from the box plot of body mass index there were outliers in the plot where there were extremely high bmi but those people were not having stroke and it can be seen here that mostly the people who had those extreme bmi values are in the age group of 20-40 in which the chances of getting a stroke is not that much.

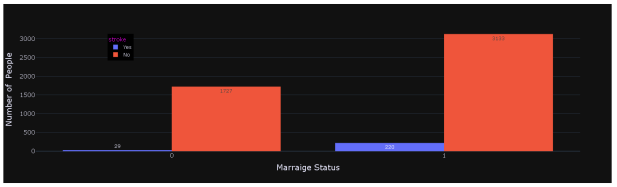



```
t.name, newnames[t.name])))

fig.show()
```

From above plot it can be inferred that the number of people who are married have greater chances of getting stroke as compared to

```
                yanchor="top",    y=0.95,
xanchor='right',                  x=0.18,
bgcolor='black'))

fig.update_layout(font_color='lavender
',

legend_title_font_color="magenta",

yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),

xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
                    legend_font_size=12)

newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda            t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(



newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda            t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.update_layout(font_color='lavender
',

legend_title_font_color="magenta",

yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),

xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
```

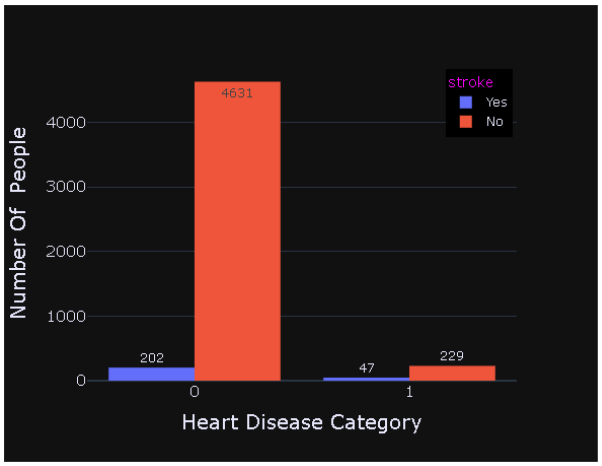
unmarried people as the number of people who are married and have stroke are greater than the number of people who are unmarried and have stroke.




```
fig = px.histogram(stroke,
                   x='heart_disease',
                   color='stroke',
                   barmode='group',

labels={'heart_disease':        'Heart
Disease Category'},
                   text_auto=True,

template='plotly_dark')

#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False)
# )

fig.update_layout(yaxis_title="Number
Of  People")
fig.update_layout(legend=dict(
                yanchor="top",    y=0.99,
xanchor='right',                  x=0.99,
bgcolor='black'))


fig = px.histogram(stroke,
                   x='hypertension',
                   color='stroke',
                   barmode='group',

labels={'hypertension':  'Hypertension
Category'},
                   text_auto=True,
```

```
                    legend_font_size=12)

fig.show()
```

From the above figure it can be interpreted that there are more people who do not have any sort of heart problem or disease but have a stroke and less number of people who have got some heart disease and have a stroke.



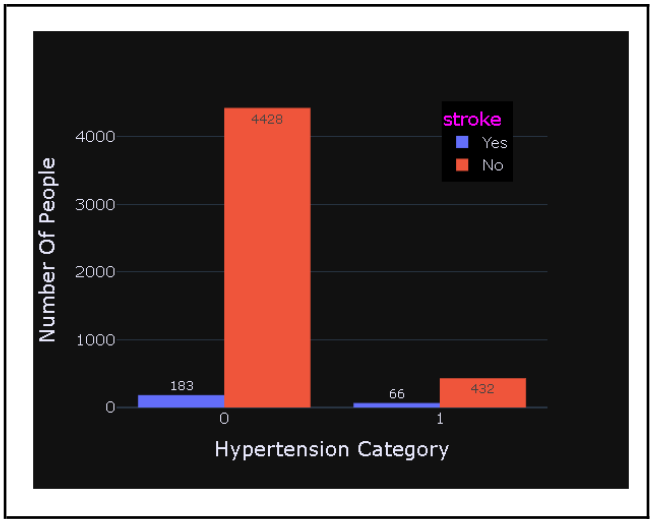

```
fig.show()
```

From the above figure it can been interpreted that there are more number of people who do not have hypertension but have a stroke and less number of people who have hypertension and have a stroke.

```
template='plotly_dark')

#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False)
# )
fig.update_layout(yaxis_title="Number
Of People")
fig.update_layout(legend=dict(
                yanchor="top",    y=0.97,
xanchor='right',                  x=0.92,
bgcolor='black'))

newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda            t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.update_layout(font_color='lavender
',

legend_title_font_color="magenta",

yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),

xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
                    legend_font_size=15)
```

For each row (i-th row), the code checks the value in the 'bmi' column at index 9 (column index 9) for that row.

Depending on the value of 'bmi,' it categorizes individuals into different BMI classes by updating the 'Bmi_classification' column:




```
stroke['Bmi_classification'] = 0
for i in range(len(stroke.index)):
    if stroke.iloc[i, 9] < 18.5:
                stroke.iloc[i,  11]  =
'Underweight'
     elif stroke.iloc[i, 9] < 25.0 and
stroke.iloc[i, 9] >= 18.5:
          stroke.iloc[i, 11] = 'Normal
weight'
     elif stroke.iloc[i, 9] < 30.0 and
stroke.iloc[i, 9] >= 25.0:
                stroke.iloc[i,  11]  =
'Overweight'
    else:
        stroke.iloc[i, 11] = 'Obese'
```

stroke['Bmi_classification'] = 0: This line initializes a new column 'Bmi_classification' with all values set to 0.

The for loop iterates through each row in the 'stroke' DataFrame using the index, from 0 to the length of the DataFrame.

If 'bmi' is less than 18.5, the 'Bmi_classification' is set to 'Underweight.'

If 'bmi' is between 18.5 and 25.0, the 'Bmi_classification' is set to 'Normal weight.'

If 'bmi' is between 25.0 and 30.0, the 'Bmi_classification' is set to 'Overweight.'

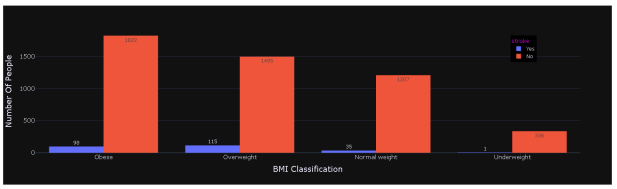
If 'bmi' is greater than or equal to 30.0, the 'Bmi_classification' is set to 'Obese.'

```
fig = px.histogram(stroke,

x='Bmi_classification',
                    color='stroke',
                    barmode='group',

labels={'hypertension':  'Hypertension
Category'},
                    text_auto=True,

template='plotly_dark')

fig.update_layout(yaxis_title="Number
Of People")
fig.update_layout(xaxis_title="BMI
Classification")
fig.update_layout(legend=dict(
              yanchor="top",    y=0.95,
xanchor='right',                x=0.92,
bgcolor='black'))

newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda           t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

                  'gender=0': 'Male',
                          'gender=1':
'Female'
                ),

template='plotly_dark',
```

```
t.name, newnames[t.name])))

fig.update_layout(font_color='lavender
',

legend_title_font_color="magenta",
```

```
yaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),

xaxis=dict(tickfont=dict(size=15),
titlefont=dict(size=20)),
                    legend_font_size=12)


fig.show()
```

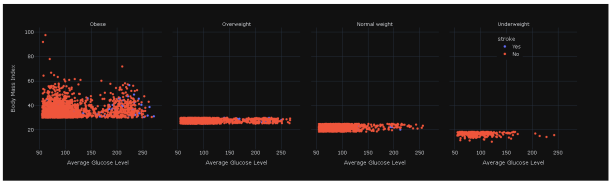
From the above figure it can be interpreted that there are the highest number of people in the overweight and obese category who have got stroke, which is also well justified by research that the people who are obese and overweight have higher chances of getting stroke as triglycerides in the blood rises and due to which there are chances of blood clot formation and eventually resulting to stroke




```
fig = px.scatter(stroke,

x='avg_glucose_level',
                    y='bmi',
                    color='stroke',
                    text='age',
                    labels={

'avg_glucose_level': 'Average Glucose
Level',
                        'bmi': 'Body Mass
Index',
    x=0.9,
))
facet_col='Bmi_classification',

hover_name='work_type')

# fig.update_layout(
#                           font_color =
'lavender',
#
yaxis=dict(tickfont=dict(size=15),titl
efont=dict(size=20)),
#
xaxis=dict(tickfont=dict(size=15),titl
efont=dict(size=20)),
#                   )

newnames = {'0': 'No', '1': 'Yes'}
fig.for_each_trace(lambda            t:
t.update(name=newnames[t.name],

legendgroup=newnames[t.name],

hovertemplate=t.hovertemplate.replace(

t.name, newnames[t.name])))

fig.for_each_annotation(lambda       a:
a.update(text=a.text.split('=')[-1]))
fig.for_each_annotation(lambda       a:
a.update(text=a.text.replace('0',
'Male')))
fig.for_each_annotation(lambda       a:
a.update(text=a.text.replace('1',
'Female')))

fig.update_layout(legend=dict(
    font=dict(size=12),
    orientation="v",
    yanchor="top",
    y=0.95,
    xanchor='right',
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)
```

```
#
fig.update_layout(xaxis=dict(showgrid=
False),
#
yaxis=dict(showgrid=False))

fig.show()
```

From the above plot it can be interpreted that most of the people who have got stroke are under the obese and overweight category. Further in these categories some have normal glucose levels but thier body mass index values are a bit on the higher side as compared to the normal levels which could potenially be the reason of them getting stroke.



X . Modelling :

Random Forest Classifier :

```
# On merged_stroke dataframe
from sklearn.ensemble import
RandomForestClassifier
from sklearn.model_selection import
train_test_split

np.random.seed(42)

X = merged_stroke.drop('stroke',
axis=1)
y = merged_stroke['stroke']

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2)

rfc = RandomForestClassifier()
rfc = RandomForestClassifier()
```

1. Import necessary libraries for building a Random Forest Classifier and data splitting.
2. Set a random seed for reproducibility.
3. Prepare the data by separating input features and the target variable.
4. Split the data into training and testing sets, with the test set representing 20% of the data.
5. Create a Random Forest Classifier model with default hyperparameters.
6. Train the model using the training data.
7. Evaluate the model's performance on the test set by computing its accuracy.

```
from sklearn.metrics import
classification_report,
confusion_matrix, accuracy_score

print(classification_report(y_test,
y_preds))
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	963
1	0.00	0.00	0.00	59
accuracy			0.94	1022
macro avg	0.47	0.50	0.49	1022
weighted avg	0.89	0.94	0.91	1022

```
# On test_df dataframe
from sklearn.ensemble import
RandomForestClassifier
from sklearn.model_selection import
train_test_split

np.random.seed(42)

X = test_df.drop('stroke', axis=1)
y = test_df['stroke']

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2)

test_df2 =
merged_stroke.drop(['heart_disease','h
ypertension'],axis=1)
```

```
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)
```

```
0.9703947368421053
```

Import the necessary libraries for building a Random Forest Classifier and data splitting.

Set a random seed to ensure reproducibility of random elements in the code.

Prepare the data by separating input features (X) and the target variable (y) from the test_df dataframe.

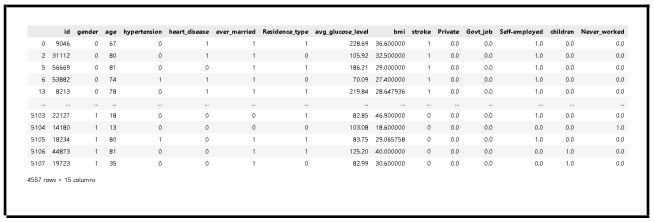
Split the data into training and testing sets, with the test set representing 20% of the data.

Create a Random Forest Classifier model with default hyperparameters.

Train the model using the training data.

Evaluate the model's performance on the test set by computing its accuracy. The code calculates and returns the mean accuracy of the model on the given test data.

```
test_df
```



	id	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	stroke	Private	Govt_job	Self-employed	children	Never_worked
0	9046	0	67	0	1	1	1	228.69	36.600000	1	0.0	0.0	1.0	0.0	0.0
2	31112	0	80	0	1	1	0	105.92	32.500000	1	0.0	0.0	1.0	0.0	0.0
5	56669	0	81	0	0	1	1	186.21	29.000000	1	0.0	0.0	1.0	0.0	0.0
6	53882	0	74	1	1	1	0	70.09	27.400000	1	0.0	0.0	1.0	0.0	0.0
13	8213	0	78	0	1	1	1	219.84	28.647536	1	0.0	0.0	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5103	22127	1	18	0	0	0	1	82.85	46.900000	0	0.0	0.0	1.0	0.0	0.0
5104	14180	1	13	0	0	0	0	103.08	18.600000	0	0.0	0.0	0.0	0.0	1.0
5105	18234	1	80	1	0	1	1	83.75	29.065758	0	0.0	0.0	1.0	0.0	0.0
5106	44873	1	81	0	0	1	1	125.20	40.000000	0	0.0	0.0	0.0	1.0	0.0
5107	19723	1	35	0	0	1	0	82.99	30.600000	0	0.0	0.0	0.0	1.0	0.0

4557 rows × 15 columns

```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score
print(classification_report(y_test, y_preds))
```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	870
1	0.94	0.38	0.54	42
accuracy			0.97	912
macro avg	0.96	0.69	0.76	912
weighted avg	0.97	0.97	0.96	912

Stratified K-fold Cross Validation:

```
# On test_df2 dataframe
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

np.random.seed(42)

X = test_df2.drop('stroke', axis=1)
y = test_df2['stroke']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
rfc.score(X_test, y_test)

y_preds = rfc.predict(X_test)
y_preds
```

```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score
print(classification_report(y_test, y_preds))
```

	precision	recall	f1-score	support
0	0.94	1.00	0.97	963
1	0.00	0.00	0.00	59
accuracy			0.94	1022
macro avg	0.47	0.50	0.49	1022
weighted avg	0.89	0.94	0.91	1022

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

XI. Algorithm and Metrics :

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
X = merged_stroke.drop('stroke', axis=1)
y = merged_stroke['stroke']
skf = StratifiedKFold()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
skf = StratifiedKFold()
rfc = RandomForestClassifier()

results = cross_val_score(rfc, X, y, cv=skf)
results
print(np.mean(results))
```

1. Import the necessary libraries for cross-validation and other operations.
2. Prepare the data by separating input features (X) and the target variable (y) from the merged_stroke dataframe.
3. Initialize a StratifiedKFold object skf. This is a cross-validation technique that maintains the class distribution in each fold.
4. Split the data into training and testing sets using train_test_split, but it appears that the skf object is created twice, which is redundant.
5. Create a Random Forest Classifier model rfc.
6. Use cross_val_score to perform k-fold cross-validation. The cross_val_score function fits the model (rfc) on the data (X and y) using stratified k-fold cross-validation (the number of folds is determined by the skf object), and it returns an array of accuracy scores for each fold.
7. Calculate the mean of the accuracy scores obtained from cross-validation and print it.

```
0.950078872062423
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve

skf = StratifiedKFold(n_splits=5)

X = test_df.drop('stroke', axis=1)
y = test_df['stroke']

import imblearn
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_smote, y_smote = smote.fit_resample(X, y)


X_train, X_test, y_train, y_test = train_test_split(X_smote,

y_smote,

test_size=0.2,

random_state=42)

models = []
models.append(('RandomForestClassifier', RandomForestClassifier()))
models.append(('KNeighborsClassifier', KNeighborsClassifier()))
```

```
models.append(('GaussianNB',
GaussianNB()))
models.append(('SupportVectorMachine',
SVC()))

#evaluate each model in turn
results = []
names = []
results_mean = []
pscore = []
rscore = []
scoring = 'accuracy'
#              roc_curve              =
plot_roc_curve(RandomForestClassifier,
X_test,y_test)

for name, model in models:
    model.fit(X_train, y_train)
    y_preds = model.predict(X_test)
    skf = StratifiedKFold(n_splits=5)
                      cv_results      =
cross_val_score(model,  X,  y,  cv=skf,
scoring=scoring)
    results.append(cv_results)

results_mean.append(cv_results.mean())
    names.append(name)
          #              precision    =
precision_score(y_test,y_preds)
           #                recall    =
recall_score(y_test,y_preds)
    msg = "%s accuracy : %f " % (name,
cv_results.mean())
    #     msg1 = "%s precision_score :
%f" % (name, precision)
     #      msg2 = "%s recall_score :
%f" % (name, recall)
    print(msg)
    #     print(msg1)
    #     print(msg2)
    print("Confusion Matrix for %s : "
% name)
        print(confusion_matrix(y_test,
y_preds))
print(classification_report(y_test,
y_preds))
```

```
RandomForestClassifier accuracy : 0.963572
Confusion Matrix for RandomForestClassifier :
[[831  40]
 [ 44 809]]
Classification report for RandomForestClassifier :
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       871
           1       0.95      0.95      0.95       853

    accuracy                           0.95      1724
   macro avg       0.95      0.95      0.95      1724
weighted avg       0.95      0.95      0.95      1724

KNeighborsClassifier accuracy : 0.945798
Confusion Matrix for KNeighborsClassifier :
[[673 198]
 [128 725]]
Classification report for KNeighborsClassifier :
              precision    recall  f1-score   support

           0       0.84      0.77      0.81       871
           1       0.79      0.85      0.82       853

    accuracy                           0.81      1724
...
    accuracy                           0.53      1724
   macro avg       0.54      0.53      0.51      1724
weighted avg       0.54      0.53      0.51      1724
```

1. Imports necessary libraries for machine learning models, cross-validation, and evaluation metrics.
2. Initializes a StratifiedKFold object for stratified k-fold cross-validation with 5 splits.
3. Prepares the data by separating input features (X) and the target variable (y) from the test_df dataframe.
4. Uses the Synthetic Minority Over-sampling Technique (SMOTE) to oversample the minority class in the dataset, resulting in X_smote and y_smote.
5. Splits the oversampled data into training and testing sets using train_test_split. The test size is set to 20%, and the random seed is fixed for reproducibility.
6. Defines a list of machine learning models, including Random Forest Classifier, K-Nearest Neighbors Classifier, Gaussian Naive Bayes, and Support Vector Machine.
7. Iterates through each model and performs the following steps for each:

a. Fits the model to the training data.

b. Makes predictions on the test data.

c. Uses stratified k-fold cross-validation with 5 splits to evaluate the model's accuracy.

d. Collects the cross-validation results, model names, and their respective accuracy means.

e. Prints the name of the model and its mean accuracy.

```
    print("Classification report for
%s : " % name)
```

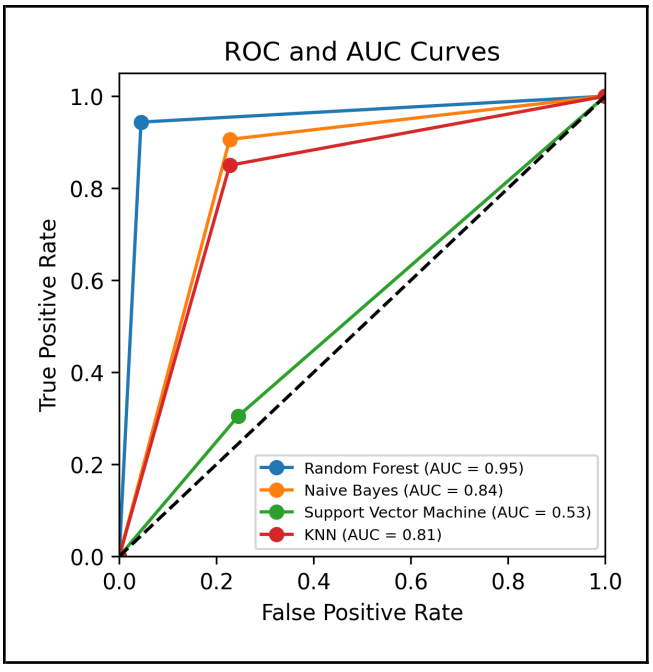
f. Prints the confusion matrix for the model's predictions on the test data.

g. Prints the classification report for the model's predictions on the test data, which includes precision, recall, and other classification metrics.

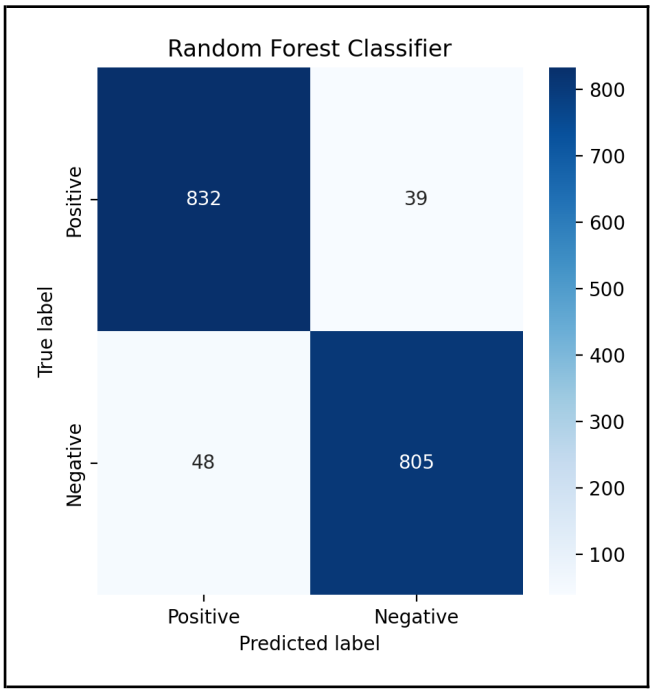
The code evaluates multiple machine learning models on the oversampled data and reports their accuracy, confusion matrices, and classification reports. It provides insights into how well each model performs on the task of predicting the target variable 'stroke' based on the input features in test_df.

```
from sklearn.metrics import roc_curve,
auc
rfc = RandomForestClassifier()
svc = SVC()
nb = GaussianNB()
kn = KNeighborsClassifier()

rfc.fit(X_train,y_train)
svc.fit(X_train,y_train)
nb.fit(X_train,y_train)
kn.fit(X_train,y_train)

y_pred_rfc = rfc.predict(X_test)
y_pred_svc = svc.predict(X_test)
y_pred_nb = nb.predict(X_test)
y_pred_kn = kn.predict(X_test)

rfc_fpr,       rfc_tpr,       _       =
roc_curve(y_test, rfc.predict(X_test))
rfc_auc = auc(rfc_fpr, rfc_tpr)



nb_fpr, nb_tpr, _ = roc_curve(y_test,
nb.predict(X_test))
nb_auc = auc(nb_fpr, nb_tpr)
print(_)
svc_fpr,       svc_tpr,       _       =
roc_curve(y_test, svc.predict(X_test))
svc_auc = auc(svc_fpr, svc_tpr)

kn_fpr, kn_tpr, _ = roc_curve(y_test,
kn.predict(X_test))
kn_auc = auc(kn_fpr, kn_tpr)
```

```
plt.plot(rfc_fpr, rfc_tpr, marker='o'
,label='Random Forest (AUC = %0.2f)' %
rfc_auc)
plt.plot(nb_fpr,               nb_tpr,
marker='o',label='Naive Bayes (AUC =
%0.2f)' % nb_auc)
plt.plot(svc_fpr,              svc_tpr,
marker='o',label='Support       Vector
Machine (AUC = %0.2f)' % svc_auc)
plt.plot(kn_fpr,               kn_tpr,
marker='o',label='KNN (AUC = %0.2f)' %
kn_auc)

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC and AUC Curves')
plt.legend(loc="lower
right",fontsize=7)
plt.show()
```



```
plt.figure(figsize=(4,4),dpi=300)

y_pred_rfc = rfc.predict(X_test)
y_pred_svc = svc.predict(X_test)
y_pred_nb = nb.predict(X_test)
y_pred_kn = kn.predict(X_test)

# Confusion matrix for rfc

plt.figure(figsize=(5,5),dpi=200)
confusion_matrix_rf =
confusion_matrix(y_test,y_pred_rfc)
sns.heatmap(confusion_matrix_rf,annot=
True,xticklabels=['Positive','Negative
'], yticklabels=[
'Positive','Negative'],
            cmap='Blues',fmt='d')

plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title("Random Forest Classifier")
plt.show()
```



1. Make predictions using the Random Forest Classifier (rfc) and the test data (X_test) for each of the models: Random Forest (rfc), Support Vector Machine (svc), Naive Bayes (nb), and K-Nearest Neighbors (kn).

2. Calculate the confusion matrix for the predictions made by the Random Forest Classifier (rfc) on the test data

3.Create a matplotlib figure with a specific size and resolution for the heatmap.

4.Display the confusion matrix using a heatmap. This heat map includes annotations (the actual numerical values of the confusion matrix) and labels for the x-axis and y-axis, indicating "Positive" and "Negative" for both true and predicted labels.

5.Set the colormap (cmap) to 'Blues' for the heatmap, which results in different shades of blue representing different values in the confusion matrix.

6.Set the format of the annotations in the heatmap to display as integers (fmt='d').

7.Add labels to the x-axis, y-axis, and a title to the heatmap to provide context for the visualization.

8.Finally, display the heatmap using plt.show().

The code's output will be a visual representation of the confusion matrix for the Random Forest Classifier, showing how well the model performed in correctly classifying positive and negative instances in the test data.

```
# Confusion matrix for SVC

plt.figure(figsize=(5,5),dpi=200)
confusion_matrix_svc =
confusion_matrix(y_test,y_pred_svc)
sns.heatmap(confusion_matrix_svc,annot
=True,xticklabels=['Positive','Negativ
e'], yticklabels=[
'Positive','Negative'],
            cmap='Blues',fmt='d')


plt.title("Support Vector Machine")
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```

1. Calculate the confusion matrix for the predictions made by the Support Vector Machine (svc) on the test data (X_test) using the confusion_matrix function. The confusion matrix provides information about the true positives, true negatives, false positives, and false negatives.
2. Create a matplotlib figure with a specific size and resolution for the heatmap.
3. Display the confusion matrix using a heatmap. This heat map includes annotations (the actual numerical values of the confusion matrix) and labels for the x-axis and y-axis, indicating "Positive" and "Negative" for both true and predicted labels.

(X_test) using the confusion_matrix function. The confusion matrix provides information about the true positives, true negatives, false positives, and false negatives.

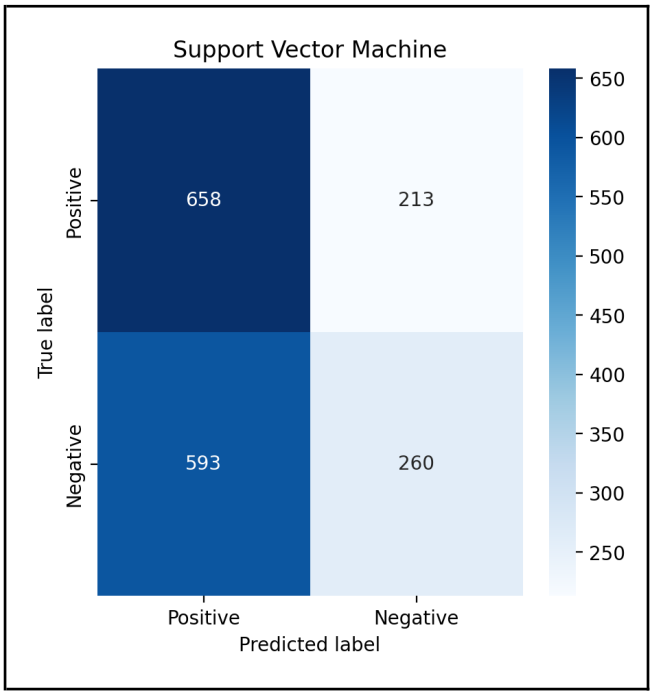
4.Set the colormap (cmap) to 'Blues' for the heatmap, resulting in different shades of blue representing different values in the confusion matrix.

5.Set the format of the annotations in the heatmap to display as integers (fmt='d').

6.Add a title to the heatmap to indicate that it represents the Support Vector Machine.

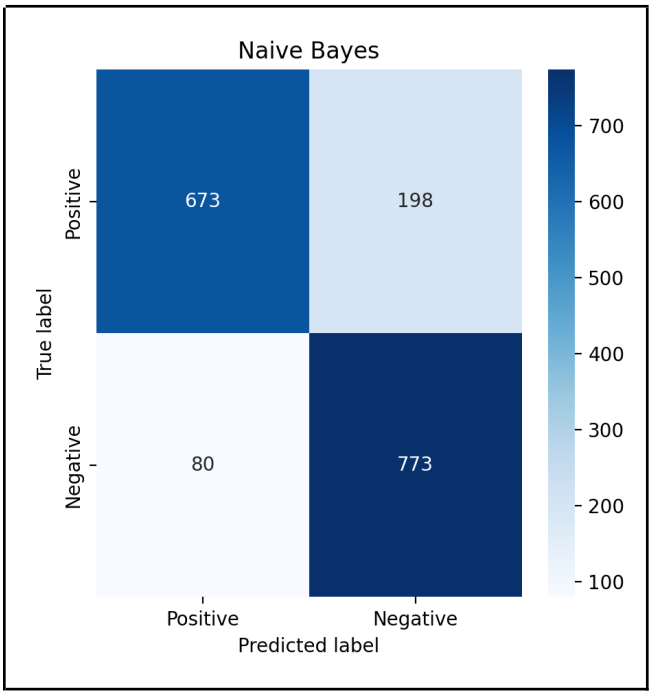
7.Add labels to the x-axis and y-axis and specify the order of label categories.

8.Finally, display the heatmap using plt.show().




```
# Confusion matrix for NB

plt.figure(figsize=(5,5),dpi=200)
confusion_matrix_nb                    =
confusion_matrix(y_test,y_pred_nb)
sns.heatmap(confusion_matrix_nb,annot=
True,xticklabels=['Positive','Negative
'],                          yticklabels=[
'Positive','Negative'],
            cmap='Blues',fmt='d')


plt.title("Naive Bayes")
```

1. Calculate the confusion matrix for the predictions made by the Naive Bayes model (nb) on the test data (X_test) using the confusion_matrix function. The confusion matrix provides information about the true positives, true negatives, false positives, and false negatives.
2. Create a matplotlib figure with a specific size and resolution for the heatmap.
3. Display the confusion matrix using a heatmap. This heat map includes annotations (the actual numerical values of the confusion matrix) and labels for the x-axis and y-axis, indicating "Positive" and "Negative" for both true and predicted labels.
4. Set the colormap (cmap) to 'Blues' for the heatmap, which results in different shades of blue representing different values in the confusion matrix.
5. Set the format of the annotations in the heatmap to display as integers (fmt='d').
6. Add a title to the heatmap to indicate that it represents the Naive Bayes (NB) model.
7. Add labels to the x-axis and y-axis and specify the order of label categories.
8. Finally, display the heatmap using plt.show().




```
plt.figure(figsize=(5,5),dpi=200)
confusion_matrix_kn                    =
confusion_matrix(y_test,y_pred_kn)
sns.heatmap(confusion_matrix_kn,annot=
True,xticklabels=['Positive','Negative
'],                          yticklabels=[
'Positive','Negative'],
            cmap='Blues',fmt='d')


plt.xlabel('Predicted label')
```

```
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

plt.title("KNN")
plt.show()
```

1. Calculate the confusion matrix for the predictions made by the K-Nearest Neighbors model (kn) on the test data (X_test) using the confusion_matrix function. The confusion matrix provides information about the true positives, true negatives, false positives, and false negatives.
2. Create a matplotlib figure with a specific size and resolution for the heatmap.
3. Display the confusion matrix using a heatmap. This heatmap includes annotations (the actual numerical values of the confusion matrix) and labels for the x-axis and y-axis, indicating "Positive" and "Negative" for both true and predicted labels.
4. Set the colormap (cmap) to 'Blues' for the heatmap, which results in different shades of blue representing different values in the confusion matrix.
5. Set the format of the annotations in the heatmap to display as integers (fmt='d').
6. Add labels to the x-axis and y-axis and specify the order of label categories.
7. Add a title to the heatmap to indicate that it represents the K-Nearest Neighbors (KNN) model.
8. Finally, display the heatmap using plt.show()

```
df                                    =
pd.DataFrame(data=results_mean,index=n
ames)
df.rename(columns={'0':'Mean_accuracy'
},inplace=True)
df['Mean_accuracy'] = results_mean
df.drop(0,inplace=True,axis=1)
df
```

```
plt.ylabel('True label')
```

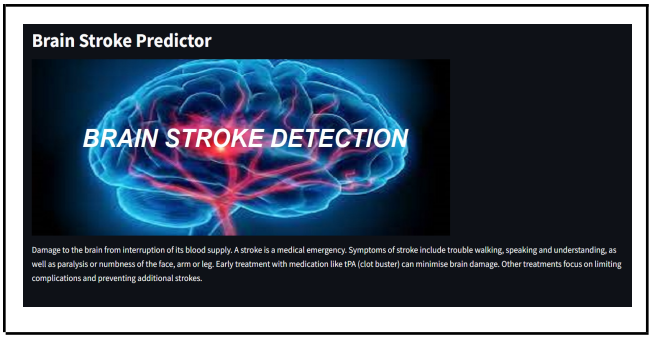
XI Pages :

a. Home :

```
"""This modules contains data about
home page"""

# Import necessary modules
import streamlit as st

def app():
    """This function create the home
page"""

    # Add title to the home page
    st.title("Brain Stroke Predictor")

    # Add image to the home page
    st.image("./images/home.jpeg")

     # Add brief describtion of your
web app
    st.markdown(
    """<p style="font-size:20px;">
            Damage to the brain from
interruption of its blood supply.
A stroke is a medical emergency.
Symptoms of stroke include trouble
walking, speaking and understanding,
as well as paralysis or numbness of
the face, arm or leg. Early treatment
with medication like tPA (clot buster)
can minimise brain damage. Other
treatments focus on limiting
complications and preventing
additional strokes.</p>
    """, unsafe_allow_html=True)
```

Streamlit Projection :

	Mean_accuracy
RandomForestClassifier	0.963572
KNeighborsClassifier	0.945798
GaussianNB	0.954136
SupportVectorMachine	0.945359

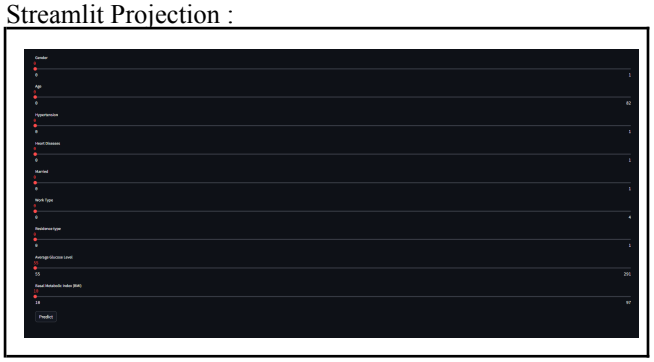


b. Predict :

```
"""This modules contains data about prediction page"""

# Import necessary modules
import streamlit as st

# Import necessary functions from web_functions
from web_functions import predict


def app(df, X, y):
        """This function create the prediction page"""

    # Add title to the page
    st.title("Prediction Page")

    # Add a brief description
    st.markdown(
        """
            <p style="font-size:25px">
                    This app uses <b style="color:green">Decision Tree Classifier</b> for the Brain Stroke Detection.
            </p>
        """, unsafe_allow_html=True)

     with st.expander("View attribute details"):
                st.markdown("""General Convension:
                1 -> Urban
                   0 -> Rural""")

    # Take feature input from the user
    # Add a subheader
    st.subheader("Select Values:")

     # Take input of features from the user.
            A = st.slider("Gender", int(df["gender"].min()), int(df["gender"].max()))
             B = st.slider("Age", int(df["age"].min()), int(df["age"].max()))
         C= st.slider("Hypertension", int(df["hypertension"].min()), int(df["hypertension"].max()))
      D = st.slider("Heart Diseases", int(df["heart_disease"].min()), int(df["heart_disease"].max()))
           E = st.slider("Married", int(df["ever_married"].min()), int(df["ever_married"].max()))
         F = st.slider("Work Type", int(df["work_type"].min()), int(df["work_type"].max()))
      G = st.slider("Residence type", int(df["Residence_type"].min()), int(df["Residence_type"].max()))
       H = st.slider("Average Glucose Level", int(df["avg_glucose_level"].min()), int(df["avg_glucose_level"].max()))
```

```
        \n 0 -> Absent or False
        \n 1 -> Present or True\n

        \nWork types:\n
        0 -> children
        1 -> Government Job
        2 -> Private Job
        3 -> Self-employed
        4 -> Unemployed

        \nResidence Type:\n

        prediction, score = predict(X, y, features)
        score = score
                    st.info("Predicted Sucessfully...")

          # Print the output according to the prediction
        if (prediction == 1):
            st.warning("The person is prone to experience a Brain Stroke!!")
        else:
            st.success("The person has relatively less probability of Brain Stroke")

        # Print teh score of the model
          st.write("The model used is trusted by doctor and has an accuracy of ", (score*100),"%")
```

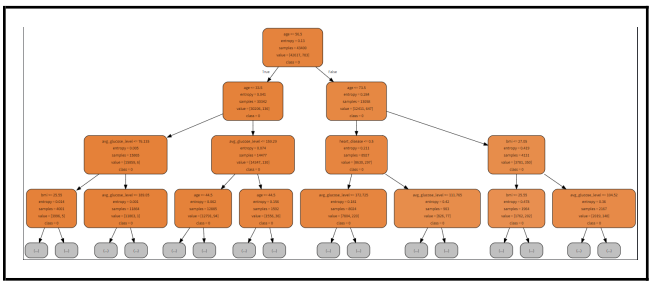
Streamlit Projection :



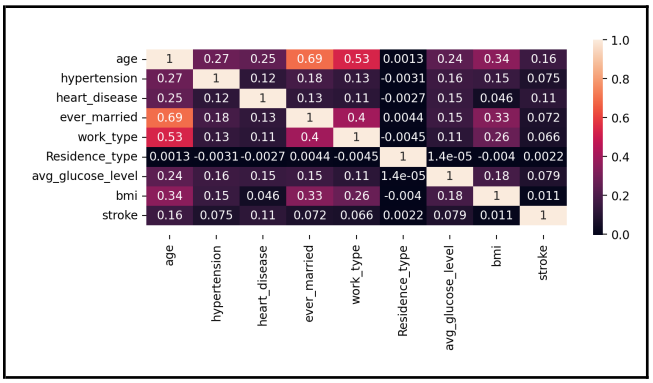
b. Sample Result :

```
        I = st.slider("Basal Metabolic Index (BMI)", int(df["bmi"].min()), int(df["bmi"].max()))

     # Create a list to store all the features
    features = [A,B,C,D,E,F,G,H,I]

    # Create a button to predict
    if st.button("Predict"):
            # Get prediction and model score
```

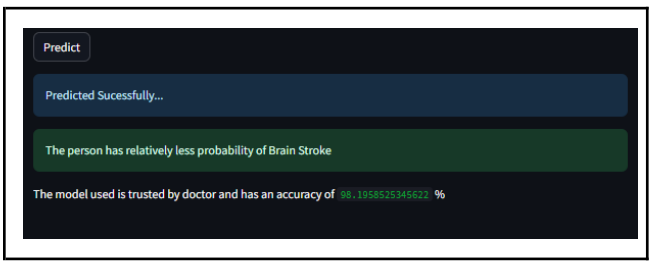
c.Plot decision Tree :

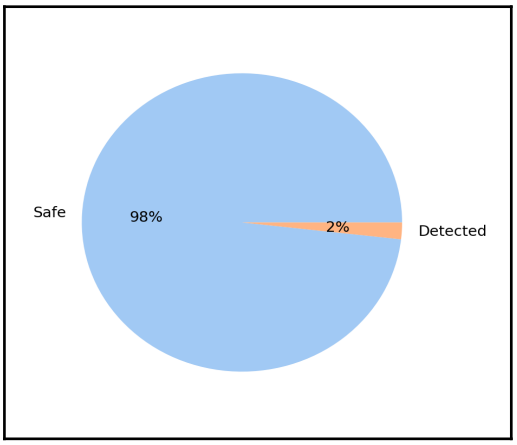


d. .Correlation Heatmap :



e.Results:





Software requirements
• Python
• Jupyter Notebook
• Windows 8,10,11

Hardware requirements
• Processor : i5 (or Above)
• RAM : 32 GB (or 16 GB of 1600 MHz DDR3 RAM)
• Storage : 300 GB
• Key Board : Standard Windows Keyboard
• Mouse : Two or Three Button Mouse.
• Internet access to download the files from Anaconda Cloud or a USB drive containing
all of the files you need with alternate instructions for air gapped installations.

XIII Conclusion :

This project demonstrates a programme that allows hand gestures to be used as a practical and simple method of software control. A gesture-based presentation controller requires no special markers and can be used in real life on simple PCs with low-cost cameras because it does not require particularly high-quality cameras to recognise or record hand movements. The method keeps track of the index finger and countertop locations on each hand. The primary goal of this typeof system is to essentially automate system components so that they are easy to control. As a result,we have used this method to make the system easier to control with the help of these applications.

XII System Requirements:

System requirements specification is a detailed statement of the effects that a system is
required to achieve. A good specification gives a complete statement of what the systems is to
do, without making any commitment as to how the system is to do it.
A system requirements specification is normally produced in response to a user requirement
specifications or other expression of requirements, and is then used as the basis for system design.
The system requirement specification typically differs from expression of requirements in both scope and precision; the latter may cover both the envisaged system and the environment in which it will operate, but may leave many broad concepts unrefined.

In order to get required knowledge about various concepts related to the present analysis, existing literature were studied. Some of the important conclusions were made through those are listed below.

[1]. JEENA R S, Dr. Sukesh Kumar , " Stroke Prediction Using SVM" , 2016 International Conferecnce on Control Instrumentation, Communication and Computational Technologies (ICCICCT),978-1-5240- 076/$31.00@2016IEEE, https://ieeexplore.ieee.org/document/8079581

[2]. M. S. Singh and P. Choudhary, "Stroke prediction using artificial intelligence," 2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON), 2017, pp. 158- 161,doi: 10.1109/IEMECON.2017.8079581. https://ieeexplore.ieee.org/document/8079581

[3]. JAEHAK YU1 , SEJIN PARK 2 , SOON-HYUN KWON1 , KANG-HEE CHO3 , AND HANSUNG LEE 4 , " AI-Based Stroke Disease Prediction System Using Real-Time ElectromyographySignals",IEEEAccess,volume10,2022. . https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9761215

[4]. .H. Mcheick, H. Nasser, M. Dbouk and A. Nasser, "Stroke Prediction Context-Aware Health Care System," 2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016, pp. 30-35, doi: 10.1109/CHASE.2016.49. https://ieeexplore.ieee.org/document/7545809

[5]. .P. A, N. G, V. K. R, P. P and S. R. R.V.T, "Stroke Prediction System Using Artificial Neural Network," 2021 6th International Conference on Communication and Electronics Systems (ICCES), 2021, pp. 1898- 1902, doi: 10.1109/ICCES51350.2021.9489055. https://ieeexplore.ieee.org/document/9489055