Prediction Of Stroke Based on Real-Time Patient Data

DIC Project Phase 1

Team:

Name	UB ID
Venkata Rohil Wardhan Kancharla	vkanchar
Mahendra Nallabothu	mnallabo

1. Problem Statement:

We try to address the problem of predicting a stroke in any individual based on various parameters obtained from their medical data.

a. According to the World Health Organization (WHO), stroke is the second most common cause of death worldwide. Stroke can also lead to organ damage or long-term disability, leaving the patient completely handicapped. This is a significant problem.

b. Predicting stroke is important as it can save countless lives outside the hospital and for patients who have been admitted for some other reason but are likely to have a stroke. Although many machine learning methods are already present to predict stroke, the accuracy of such models is a concerning issue that results in getting ignored and the patient having a stroke.

2. Data Sources:

The dataset has been taken from Kaggle, a real-time dataset of several patient medical attributes.

Columns	Description
ID	Unique identifier
Gender	Gender of patients ”Male," "Female" or "Other"

Age	Age of the patient
Hypertension	0 if the patient doesn't have hypertension, 1 if the patient has hypertension
Heart disease	0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
Ever married	Patient has ever married or not "No" or "Yes"
Work_type	This field represents the work type of patient.
Residence type	This field represents the "Rural" or "Urban" residence type
Avg glucose level	Average glucose level in the blood for the patient.
BMI	Body mass index of the patient.
smoking	"formerly smoked", "never smoked", "smokes," or "Unknown"*.
Stroke	1 if the patient had a stroke or 0 if not

https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

3. Data Cleaning/Processing:

a. Renaming Columns -

Some header names for the columns in the dataset are renamed like (from work.type to work_type). This is done to avoid the unnecessary error that will be faced in the upcoming coding like (work.type.value_counts()), which shows an error as '.' is read twice.

```
stroke = Stroke.rename(columns = {'heart.disease':'heart_disease', 'ever.married':'marital_status', 'work.type':'work_type', 'Res
stroke
```

	id	gender	age	hypertension	heart_disease	marital_status	work_type	residence_type	avgglucose_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
...	...	...	...	...	...	...	...	...	...	...	...	...
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
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

5110 rows × 12 columns

b. Number of NotNNull and Null Values -

Displaying the number of not-null and null values shows the count of null values present in the whole dataset and are displayed separately for every column.

```
stroke.notna().sum()
```

```
id                  5110
gender              5110
age                 5110
hypertension        5110
heart_disease       5110
marital_status      5110
work_type           5110
residence_type      5110
avgglucose_level    5110
bmi                 4909
smoking_status      5110
stroke              5110
dtype: int64
```

```
stroke.isna().sum()
```

```
id                    0
gender                0
age                   0
hypertension          0
heart_disease         0
marital_status        0
work_type             0
residence_type        0
avgglucose_level      0
bmi                 201
smoking_status        0
stroke                0
dtype: int64
```

c. **Dropping of Null Values -**

The Null values can either be dropped or filled with random values based on their preference. Maintaining Null values in a dataset changes the accuracy of the dataset when using regression and future steps.

```
stroke = stroke.dropna(axis=0)
stroke
```

	id	gender	age	hypertension	heart_disease	marital_status	work_type	residence_type	avgglucose_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
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
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5104	14180	Female	13.0	0	0	No	children	Rural	103.08	18.6	Unknown	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
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

d. **Unique Values of Every Column-**

The Unique values of every column are taken and displayed, but only the unique value names are displayed. This shows us the number of categories present for every dataset column.

```
s1 = pd.unique(stroke['id'])
s2 = pd.unique(stroke['gender'])
s3 = pd.unique(stroke['age'])
s4 = pd.unique(stroke['hypertension'])
s5 = pd.unique(stroke['heart_disease'])
s6 = pd.unique(stroke['marital_status'])
s7 = pd.unique(stroke['work_type'])
s8 = pd.unique(stroke['residence_type'])
s9 = pd.unique(stroke['avgglucose_level'])
s10 = pd.unique(stroke['bmi'])
s11 = pd.unique(stroke['smoking_status'])
s12 = pd.unique(stroke['stroke'])
s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12
```

```
(array([ 9046, 31112, 60182, ..., 19723, 37544, 44679], dtype=int64),
 array(['Male', 'Female', 'Other'], dtype=object),
 array([6.70e+01, 8.00e+01, 4.90e+01, 7.90e+01, 8.10e+01, 7.40e+01,
        6.90e+01, 7.80e+01, 6.10e+01, 5.40e+01, 5.00e+01, 6.40e+01,
        7.50e+01, 6.00e+01, 7.10e+01, 5.20e+01, 8.20e+01, 6.50e+01,
        5.70e+01, 4.20e+01, 4.80e+01, 7.20e+01, 5.80e+01, 7.60e+01,
        3.90e+01, 7.70e+01, 6.30e+01, 7.30e+01, 5.60e+01, 4.50e+01,
        7.00e+01, 5.90e+01, 6.60e+01, 4.30e+01, 6.80e+01, 4.70e+01,
        5.30e+01, 3.80e+01, 5.50e+01, 4.60e+01, 3.20e+01, 5.10e+01,
        1.40e+01, 3.00e+00, 8.00e+00, 3.70e+01, 4.00e+01, 3.50e+01,
        2.00e+01, 4.40e+01, 2.50e+01, 2.70e+01, 2.30e+01, 1.70e+01,
        1.30e+01, 4.00e+00, 1.60e+01, 2.20e+01, 3.00e+01, 2.90e+01,
        1.10e+01, 2.10e+01, 1.80e+01, 3.30e+01, 2.40e+01, 3.60e+01,
        6.40e-01, 3.40e+01, 4.10e+01, 8.80e-01, 5.00e+00, 2.60e+01,
        3.10e+01, 7.00e+00, 1.20e+01, 6.20e+01, 2.00e+00, 9.00e+00,
        1.50e+01, 2.80e+01, 1.00e+01, 1.80e+00, 3.20e-01, 1.08e+00,
        1.90e+01, 6.00e+00, 1.16e+00, 1.00e+00, 1.40e+00, 1.72e+00,
        2.40e-01, 1.64e+00, 1.56e+00, 7.20e-01, 1.88e+00, 1.24e+00,
        8.00e-01, 4.00e-01, 8.00e-02, 1.48e+00, 5.60e-01, 1.32e+00,
        1.60e-01, 4.80e-01]),
 array([0, 1], dtype=int64),
 array([1, 0], dtype=int64),
 array(['Yes', 'No'], dtype=object),
 array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
```

e. **Count of Duplicates for a Single Column -**
The value count is the number of duplicates present for every unique value in a column. In gender, only one value of 'other' is present. But for other columns, many duplicates are present for every unique value.

```
dup1 = stroke.pivot_table(columns=['gender'], aggfunc='size')
dup1
```

```
gender
Female    2897
Male      2011
Other        1
dtype: int64
```

```
stroke.pivot_table(columns=['smoking_status'], aggfunc='size')
```

```
smoking_status
Unknown            1483
formerly smoked     837
never smoked       1852
smokes              737
dtype: int64
```

f. **Cleaning of Column in Dataframe -**
As mentioned above, the 'gender' column has three unique values, but there is no duplicate value for one of the unique values. This can be removed by cleaning the data and dropping the row where the 'other' gender is present. Since the data is large, one dropping off one value will not disrupt the data. The value is dropped for easy processing of data.

```
stroke.drop(stroke[stroke['gender'] == 'Other'].index, inplace = True)
stroke['gender'].unique()
```

```
array(['Male', 'Female'], dtype=object)
```

g. **Correlation of Data -**
The correlation of columns in the dataset is taken to understand the different variables and attributes of the dataset. It can also be used to check the relationship between two or more variables.

```
stroke.corr()
```

	id	age	hypertension	heart_disease	avgglucose_level	bmi	stroke
id	1.000000	0.009124	0.001206	0.004058	0.006252	0.003238	0.004878
age	0.009124	1.000000	0.274395	0.257104	0.236000	0.333314	0.232313
hypertension	0.001206	0.274395	1.000000	0.115978	0.180614	0.167770	0.142503
heart_disease	0.004058	0.257104	0.115978	1.000000	0.154577	0.041322	0.137929
avgglucose_level	0.006252	0.236000	0.180614	0.154577	1.000000	0.175672	0.138984
bmi	0.003238	0.333314	0.167770	0.041322	0.175672	1.000000	0.042341
stroke	0.004878	0.232313	0.142503	0.137929	0.138984	0.042341	1.000000

h. Label Encoding of Dataset -

Label encoding refers to assigning integer values to columns of the dataset with object datatype. Label encoding starts with 0 assigned to the highest number of duplicates present for that column and then goes on assigning in descending order. Label encoding avoids errors while performing regression, convolution, and machine learning techniques, as object datatypes cannot perform such techniques.\

```
stroke['gender'] = LabelEncoder().fit_transform(stroke[['gender']])
stroke['marital_status'] = LabelEncoder().fit_transform(stroke[['marital_status']])
stroke['work_type'] = LabelEncoder().fit_transform(stroke[['work_type']])
stroke['residence_type'] = LabelEncoder().fit_transform(stroke[['residence_type']])
stroke['smoking_status'] = LabelEncoder().fit_transform(stroke[['smoking_status']])
stroke
```

	id	gender	age	hypertension	heart_disease	marital_status	work_type	residence_type	avgglucose_level	bmi	smoking_status	stroke
0	9046	1	67.0	0	1	1	2	1	228.69	36.6	1	1
2	31112	1	80.0	0	1	1	2	0	105.92	32.5	2	1
3	60182	0	49.0	0	0	1	2	1	171.23	34.4	3	1
4	1665	0	79.0	1	0	1	3	0	174.12	24.0	2	1
5	56669	1	81.0	0	0	1	2	1	186.21	29.0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5104	14180	0	13.0	0	0	0	4	0	103.08	18.6	0	0
5106	44873	0	81.0	0	0	1	3	1	125.20	40.0	2	0
5107	19723	0	35.0	0	0	1	3	0	82.99	30.6	2	0
5108	37544	1	51.0	0	0	1	2	0	166.29	25.6	1	0
5109	44679	0	44.0	0	0	1	0	1	85.28	26.2	0	0

4908 rows × 12 columns

i. Knowing the Datatypes -

It is important to know the data types present in the dataset to check its feasibility and smooth functioning while performing additional actions, i.e. Additional data cannot be added if its datatype differs from the present dataset.

```
stroke.dtypes
```

```
id                     int64
gender                 int32
age                  float64
hypertension           int64
heart_disease          int64
marital_status         int32
work_type              int32
residence_type         int32
avgglucose_level     float64
bmi                  float64
smoking_status         int32
stroke                 int64
dtype: object
```

j. Change of Datatypes -

Datatypes for every attribute of the dataset should be the same as equivalent datatypes (int' and 'float'). This helps in the normalization of data.

```
stroke['gender'] = stroke['gender'].astype(np.int64)
stroke['age'] = stroke['age'].astype(np.int64)
stroke['marital_status'] = stroke['marital_status'].astype(np.int64)
stroke['work_type'] = stroke['work_type'].astype(np.int64)
stroke['residence_type'] = stroke['residence_type'].astype(np.int64)
stroke['smoking_status'] = stroke['smoking_status'].astype(np.int64)
stroke
```

	id	gender	age	hypertension	heart_disease	marital_status	work_type	residence_type	avgglucose_level	bmi	smoking_status	stroke
0	9046	1	67	0	1	1	2	1	228.69	36.6	1	1
2	31112	1	80	0	1	1	2	0	105.92	32.5	2	1
3	60182	0	49	0	0	1	2	1	171.23	34.4	3	1
4	1665	0	79	1	0	1	3	0	174.12	24.0	2	1
5	56669	1	81	0	0	1	2	1	186.21	29.0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5104	14180	0	13	0	0	0	4	0	103.08	18.6	0	0
5106	44873	0	81	0	0	1	3	1	125.20	40.0	2	0
5107	19723	0	35	0	0	1	3	0	82.99	30.6	2	0
5108	37544	1	51	0	0	1	2	0	166.29	25.6	1	0
5109	44679	0	44	0	0	1	0	1	85.28	26.2	0	0

4908 rows × 12 columns

k. The number of Unique Values of the Dataset -

The number of unique values of a dataset is taken to find the total number of unique values present for every dataset column. This is helpful when preprocessing large amounts of data.

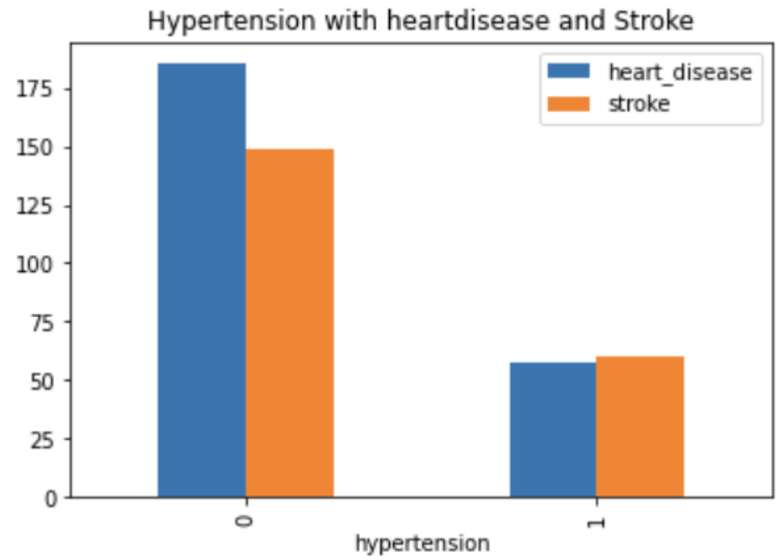
```
stroke.nunique(axis=0)
```

```
id                  4908
gender                 2
age                   83
hypertension           2
heart_disease          2
marital_status         2
work_type              5
residence_type         2
avgglucose_level    3851
bmi                  418
smoking_status         4
stroke                 2
dtype: int64
```

4. Exploratory Data Analysis (EDA):

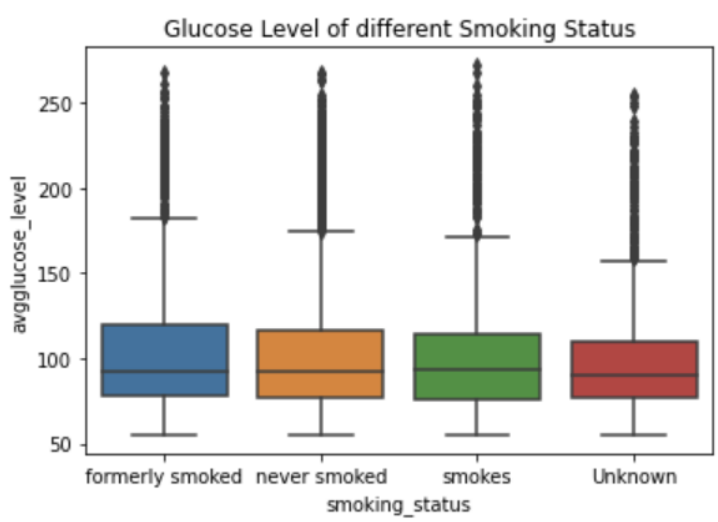
a. Count of Heart Disease and Stroke for Hypertension -

The number of people with heart disease and if the patient had a stroke based on their hypertension is plotted by a bar plot. We can observe that more people without hypertension have heart disease and have had a stroke than those with hypertension.



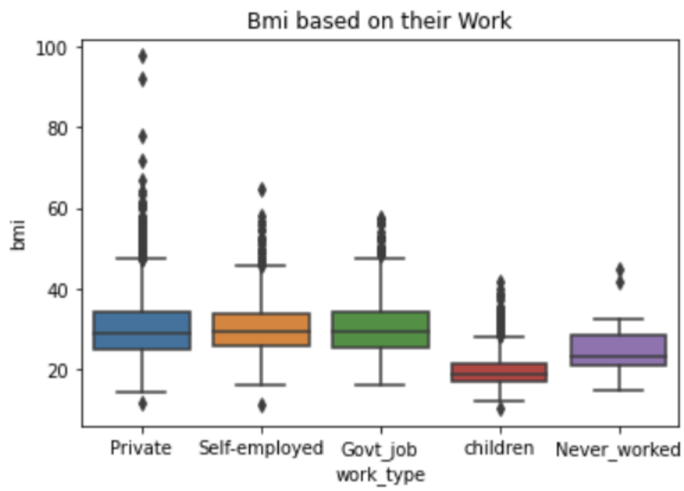
b. Smoking Status vs Glucose Level -

A box plot of smoking status and glucose level is taken to show how a person's glucose level might get affected if they smoke. In the plot, we can observe that a person who formerly smoked has a higher range of average glucose levels, but the highest glucose level is present for a smoker. This can be used to predict the likeliness of a stroke.



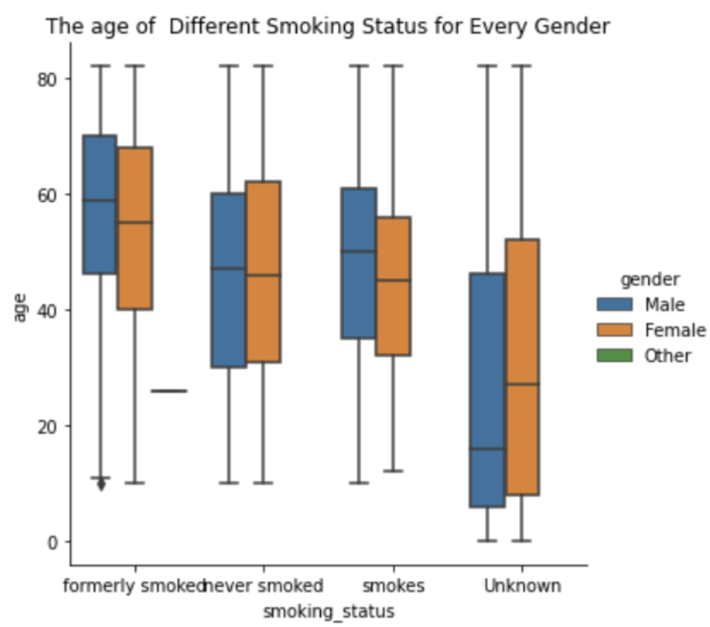
c. **WorkType vs BMI -**

The box plot maps the BMI of the entries based on their work type. Stressful work often leads to increased eating as they are always hungry, which results in an increase in their BMI. This can also lead to the probability of a stroke. As we can observe from the plot, people working in the private sector have a higher BMI than other attributes. But, the BMI for Private, Self-Employed, and Govt-Job has almost the same range.



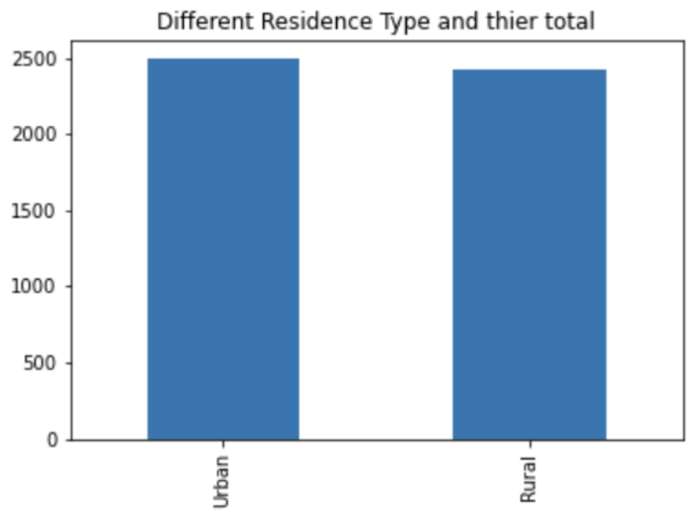
d. **Smoking Status vs Age for every Gender -**

This plot shows the age of people correlated with their smoking status for every gender. We can observe that the range of age is the same irrespective of their smoking status for both males and females.



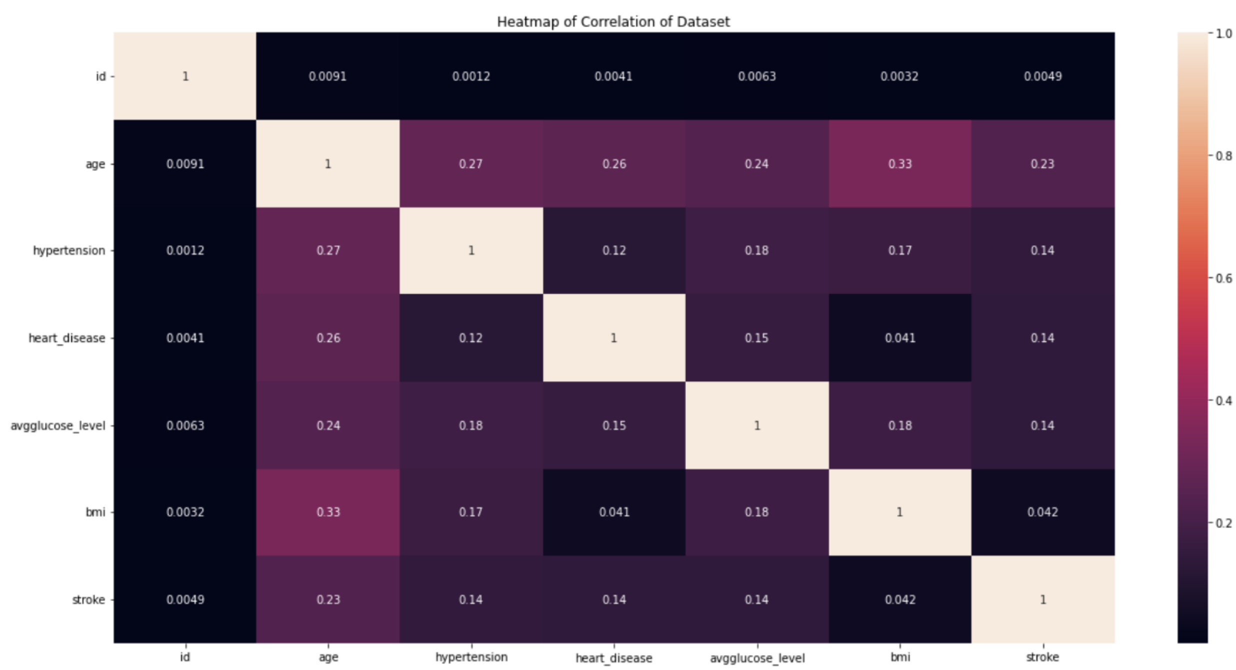
e. **Count of Residence Type -**

A bar plot of the total number of rural vs urban residents has been shown. Since the difference between the people living in rural and urban is negligible, we cannot take this data to predict the probability of a stroke based on just the residence type data without using other attributes.



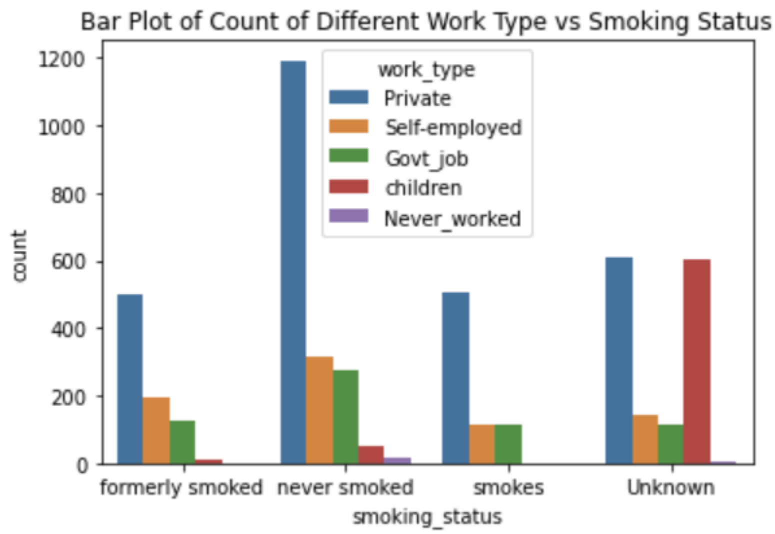
f. **Correlation of Dataset -**

A heatmap of the correlation of the dataset is shown. Only the int and float values are considered in the heatmap since object datatypes cannot be mapped in a heatmap. The heatmap can be used for the visual display and easy understanding of the relationship between variables and attributes.



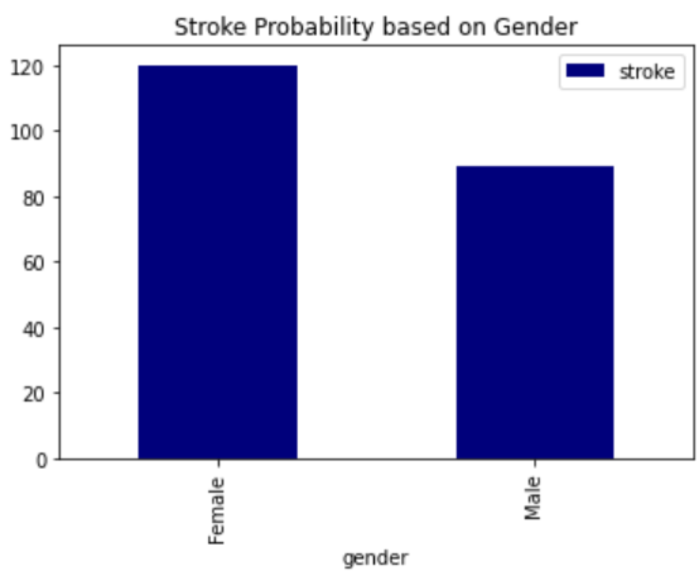
g. Count of Smoking Status vs Work Type -

The total people count is shown in a bar plot based on their smoking status and work type. The number of people working in the private sector is high irrespective of their smoking status. Self-employed and Govt-Job are close to each other for all smoking statuses.



h. Stroke based on Gender -

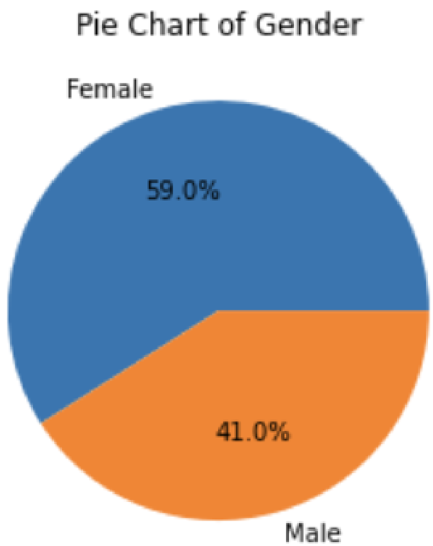
A bar plot of stroke based on gender is displayed. The plot shows that more females have had stroke than males. But, other factors of the data need to be taken into consideration before declaring this result.



i. **Pie Plot of Gender -**

The pie plot of the number of males vs females in the dataset is shown. This is taken to find and map the label encoder value with the names of the values.

```
gender
0    2897
1    2011
dtype: int64
```



j. **Heart Disease vs Stroke -**

A bar plot of people with and without heart disease who had a stroke or not is taken. It can be seen that people with heart disease have had a stroke previously. The observation can be taken as a reference, but other variables need to be added to know the result of the actual probability of a stroke.