Description:

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

Table of Contents:

- Step 1 | Python Libraries
  - 1.1 | Import Libraries
  - 1.2 | Library configurations
- Step 2 | Data
  - 2.1 | Collecting Data
  - 2.2 | Data Information
  - 2.3 | Attribute Information
- Step 3 | Data Preprocessning
  - 3.1 | Missing Values Handling
  - 3.2 | Visualization and Plots
  - 3.3 | Plots Analysis
  - 3.4 | Unique Values
  - 3.5 | Normalization
- Step 4 | Modeling
  - 4.1 | Initialization
  - 4.2 | RandomForestClassifier
  - 4.3 | LogisticRegression
  - 4.4 | SVC
  - 4.5 | DecisionTreeClassifier
  - 4.6 | KNeighborsClassifier
  - 4.7 | result

- 4.8 | Final Modeling

Step1 | Python Libraries

Step 1.1 | Import Libraries

In [2]:
```
!pip install --upgrade imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in d:\ana\lib\site-packages (0.8.0)
Collecting imbalanced-learn
  Obtaining dependency information for imbalanced-learn from https://files.pythonhosted.org/packages/a3/9e/fbe60a768502
af54563dcb59ca7856f5a8833b3ad5ada658922e1ab09b7f/imbalanced_learn-0.11.0-py3-none-any.whl.metadata
  Using cached imbalanced_learn-0.11.0-py3-none-any.whl.metadata (8.3 kB)
Requirement already satisfied: numpy>=1.17.3 in d:\ana\lib\site-packages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in d:\ana\lib\site-packages (from imbalanced-learn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in d:\ana\lib\site-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in d:\ana\lib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\ana\lib\site-packages (from imbalanced-learn) (2.2.0)
Using cached imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.8.0
    Uninstalling imbalanced-learn-0.8.0:
      Successfully uninstalled imbalanced-learn-0.8.0
Successfully installed imbalanced-learn-0.11.0
```

In [20]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from datetime import datetime

################### Sklearn ####################################
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
```

Step 1.2 | Library configurations

In [21]:
```
pd.options.mode.copy_on_write = True # Allow re-write on variable
#sns.set_style('darkgrid') # Seaborn style
warnings.filterwarnings('ignore') # Ignore warnings
pd.set_option('display.max_columns', None) # Setting this option will print all collumns of a dataframe
pd.set_option('display.max_colwidth', None) # Setting this option will print all of the data in a feature
```

Step 2 | Data

Step 2.1 | Collecting Data

At first, import dataset in csv format by pandas library and read_csv method.

In [22]:
```
data = pd.read_csv(r"C:\Users\hp\Downloads\archive (8)\healthcare-dataset-stroke-data.csv")
data.head()
```

Out[22]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	strok
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	

Step 2.2 | Data Informations

We drop id columns, because its a unique identifier number.

In [23]:
```
# Drop column = 'id'
data.drop(columns='id', inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             5110 non-null   object 
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64  
 3   heart_disease      5110 non-null   int64  
 4   ever_married       5110 non-null   object 
 5   work_type          5110 non-null   object 
 6   Residence_type     5110 non-null   object 
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     5110 non-null   object 
 10  stroke             5110 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 439.3+ KB
```

In [24]:
```
round(data.describe(include='all'), 2)
```

Out[24]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	5
count	5110	5110.00	5110.0	5110.00	5110	5110	5110	5110.00	4909.00	5110	
unique	3	NaN	NaN	NaN	2	5	2	NaN	NaN	4	
top	Female	NaN	NaN	NaN	Yes	Private	Urban	NaN	NaN	never smoked	
freq	2994	NaN	NaN	NaN	3353	2925	2596	NaN	NaN	1892	
mean	NaN	43.23	0.1	0.05	NaN	NaN	NaN	106.15	28.89	NaN	
std	NaN	22.61	0.3	0.23	NaN	NaN	NaN	45.28	7.85	NaN	
min	NaN	0.08	0.0	0.00	NaN	NaN	NaN	55.12	10.30	NaN	
25%	NaN	25.00	0.0	0.00	NaN	NaN	NaN	77.24	23.50	NaN	
50%	NaN	45.00	0.0	0.00	NaN	NaN	NaN	91.88	28.10	NaN	
75%	NaN	61.00	0.0	0.00	NaN	NaN	NaN	114.09	33.10	NaN	
max	NaN	82.00	1.0	1.00	NaN	NaN	NaN	271.74	97.60	NaN	

We have 5110 samples , with no null values

Step 2.3 | Attribute Information

Attribute Information

|Feature|Describtion| |:-----:|:---------:| |id| unique identifier| |gender| "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 diseas| |ever_married| "No" or "Yes"| |work_type| "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"| |Residence_type| "Rural" or "Urban"| |avg_glucose_level| average glucose level in blood| |bmi| body mass index| |smoking_status| "formerly smoked", "never smoked", "smokes" or "Unknown"*| |stroke| 1 if the patient had a stroke or 0 if not|

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

Step 3 | Data Preprocessning

Step 3.1 | Missing Values Handling

In [25]:
```
data.isna().sum()
```

Out[25]:
```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

In [26]:
```
print((data.isna().sum()/len(data))*100)
```

```
gender               0.000000
age                  0.000000
hypertension         0.000000
heart_disease        0.000000
ever_married         0.000000
work_type            0.000000
Residence_type       0.000000
avg_glucose_level    0.000000
bmi                  3.933464
smoking_status       0.000000
stroke               0.000000
dtype: float64
```

There is 201 samples with no values in bmi column , its about 4% of data. For better result we drop them.

In [27]:
```
data.dropna(how='any', inplace=True)
```

Step 3.2 | Visualization and Plots

In [28]:
```
cols = data.columns[:-1]
cols
```

Out[28]:
```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status'],
      dtype='object')
```

In [29]:
```
data
```

Out[29]:

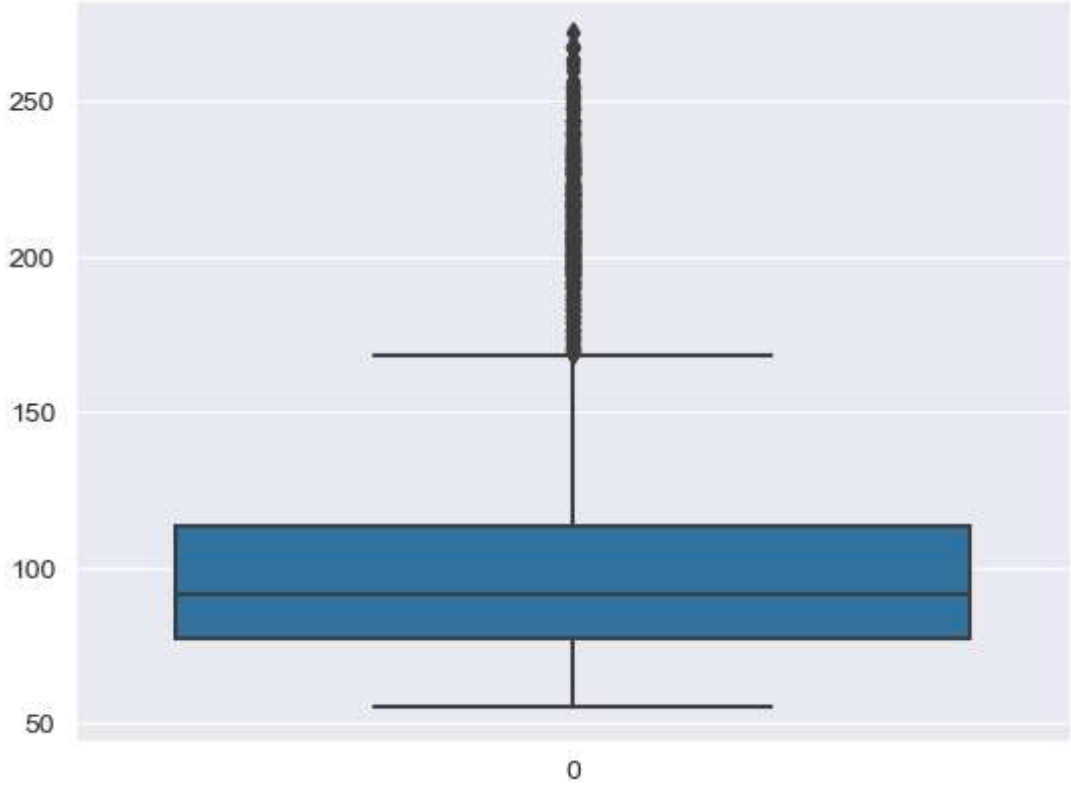
	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
5	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
5104	Female	13.0	0	0	No	children	Rural	103.08	18.6	Unknown	0
5106	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

4909 rows × 11 columns

In [30]:
```
numeric_columns = ['age', 'bmi', 'avg_glucose_level']
categorical_columns = ['gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'smoki
```

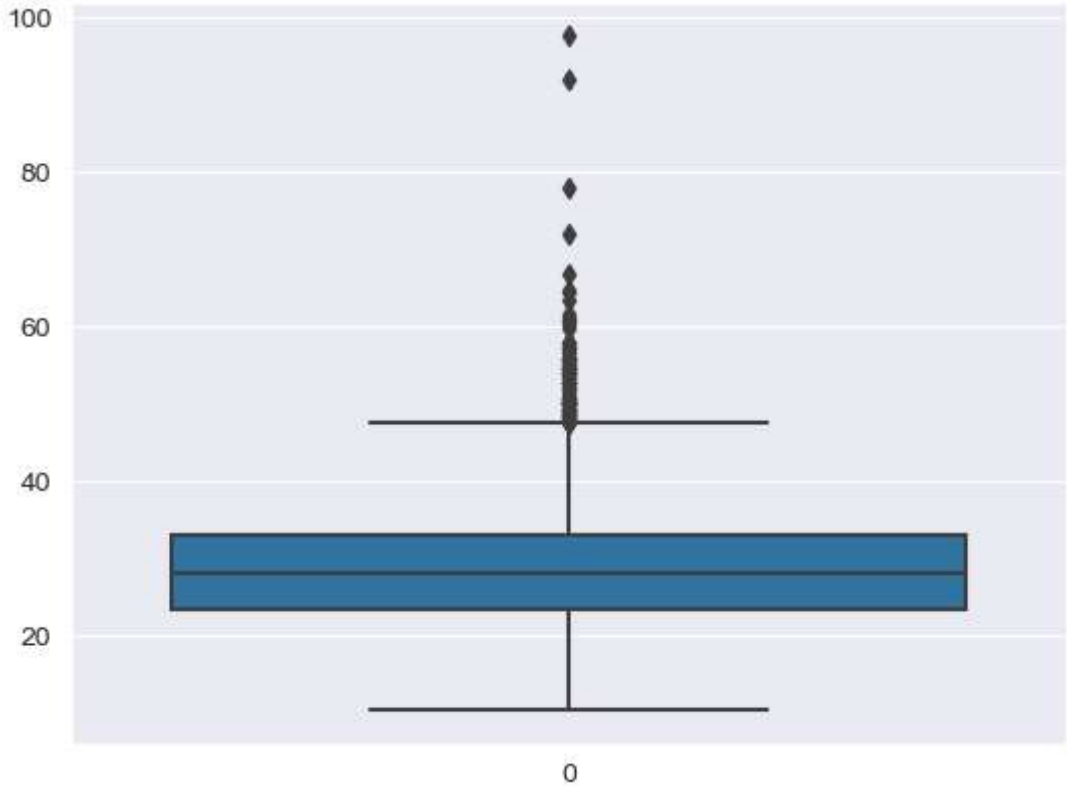
In [31]:
```
#Box plot presentation for Average Glucose Level
sns.boxplot(data['avg_glucose_level'])
```

Out[31]:
```
<Axes: >
```



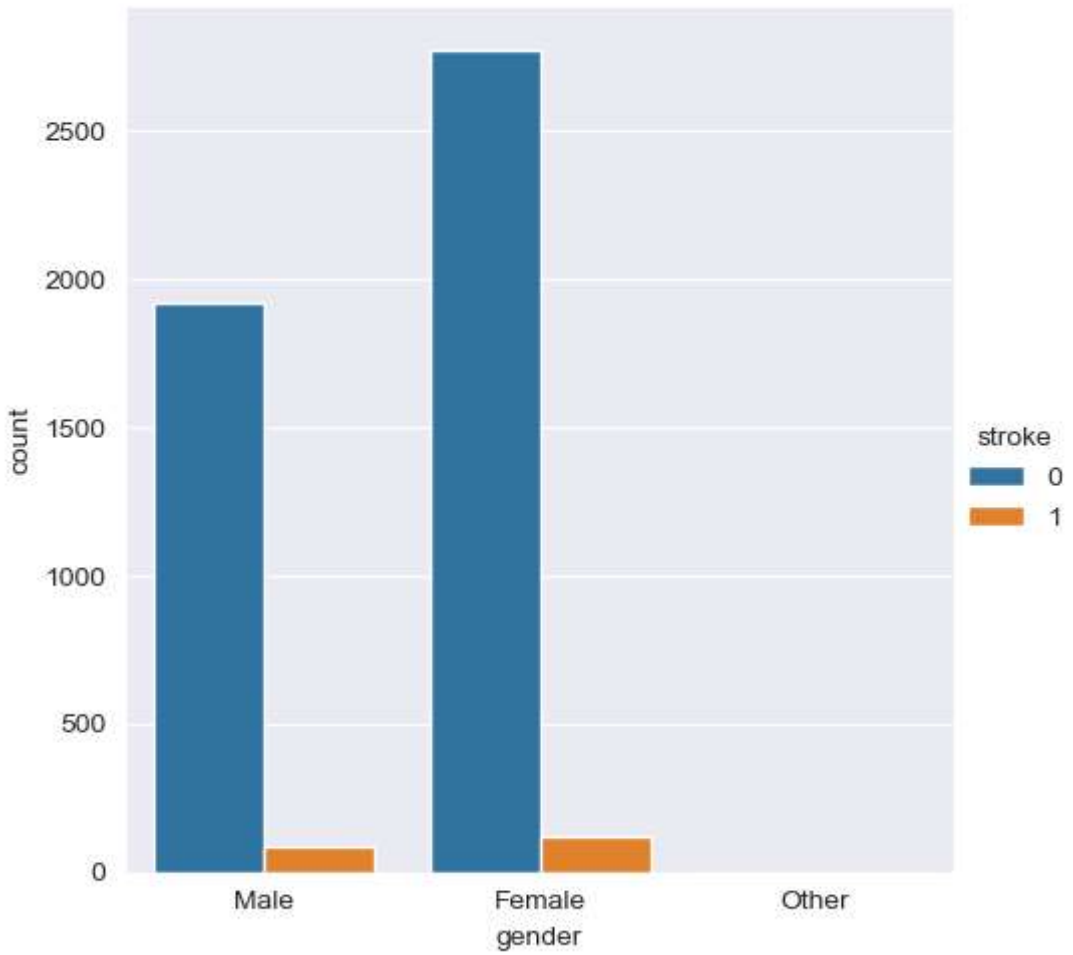
In [32]:
```
#Box plot presentation for BMI
sns.boxplot(data['bmi'])
```

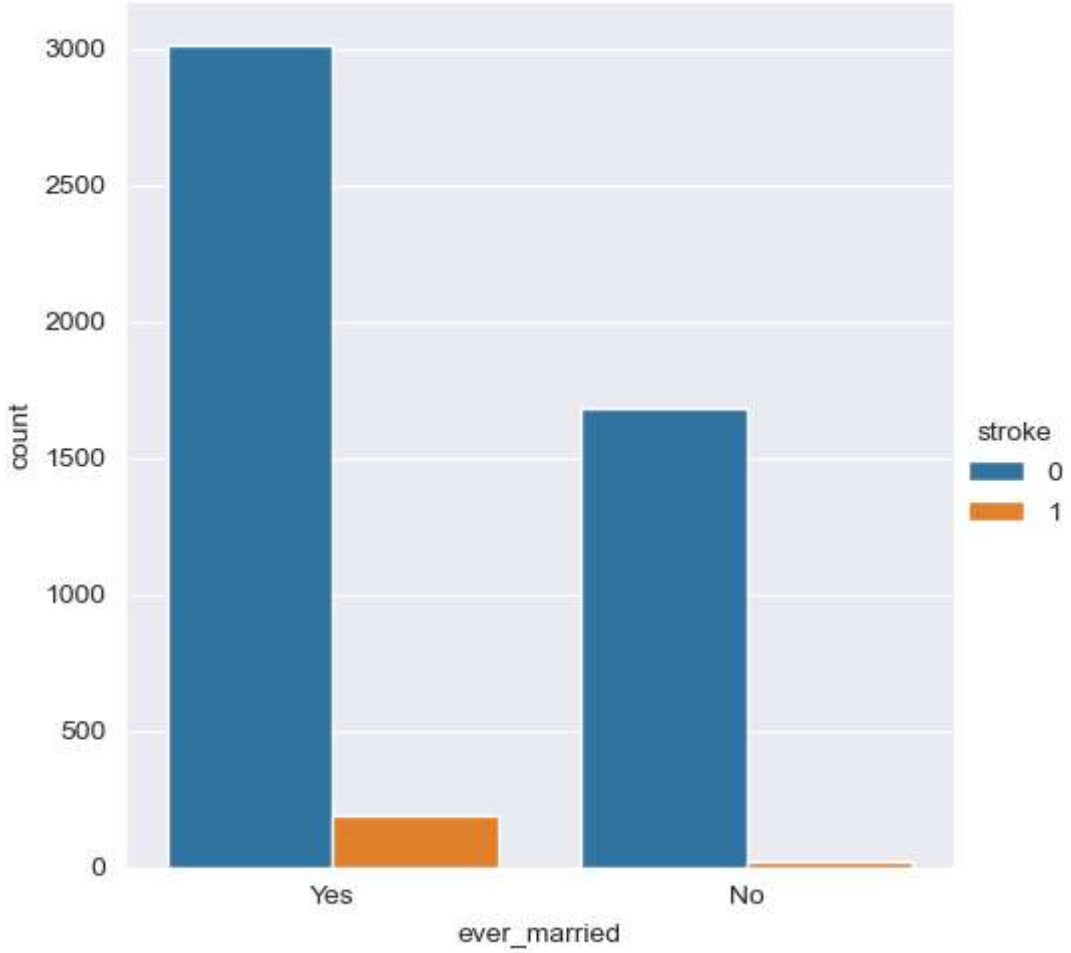
Out[32]:
```
<Axes: >
```

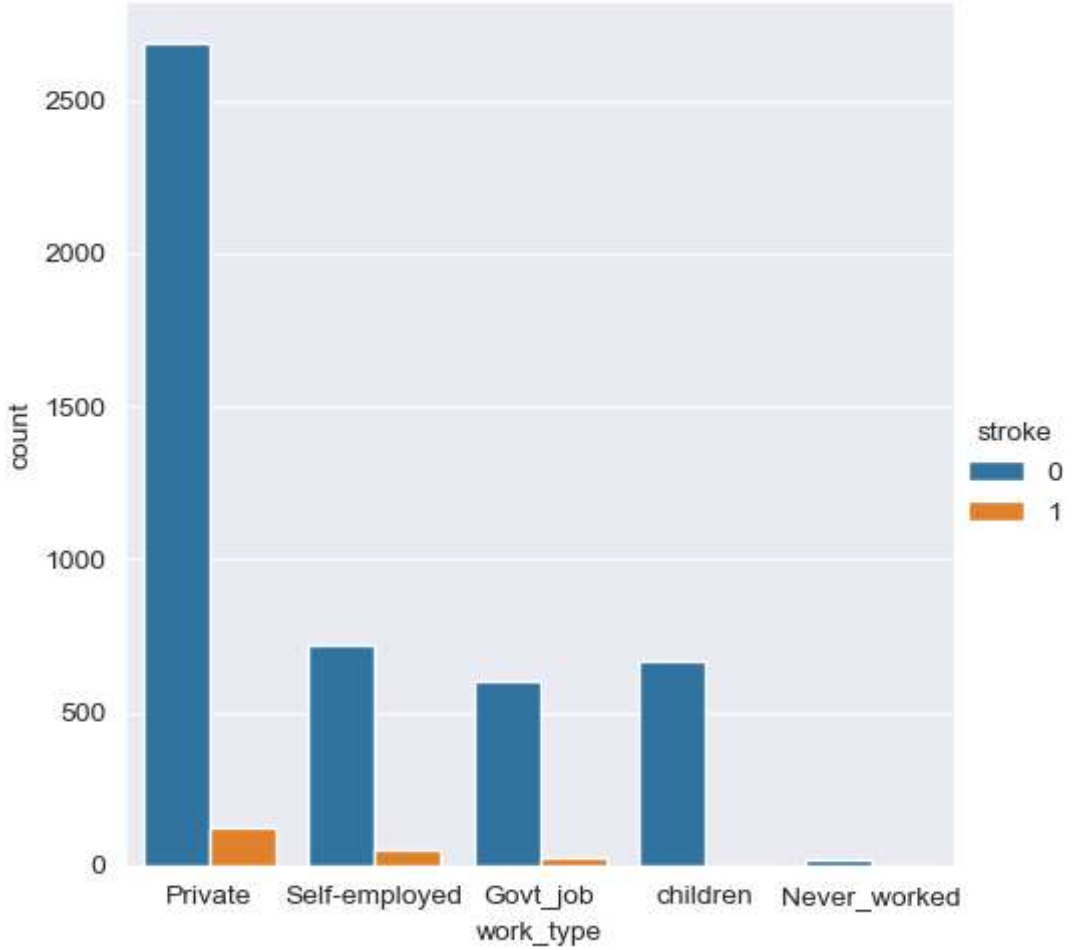


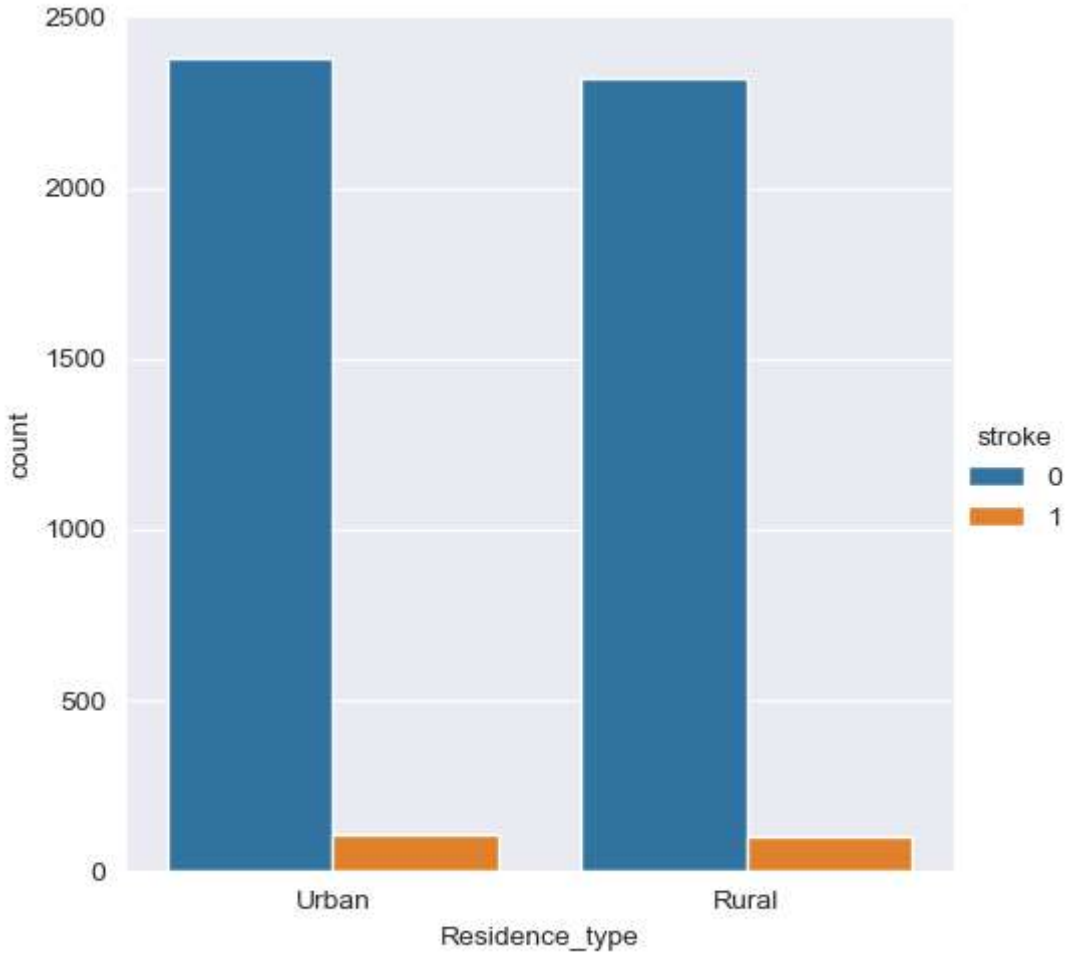
In [33]:
```
#Categorical plot presentation
sns.catplot(x="gender", hue="stroke", data=data, kind="count")
sns.catplot(x="ever_married", hue="stroke", data=data, kind="count")
sns.catplot(x="work_type", hue="stroke", data=data, kind="count")
sns.catplot(x="Residence_type", hue="stroke", data=data, kind="count")
sns.catplot(x="smoking_status", hue="stroke", data=data, kind="count")
```

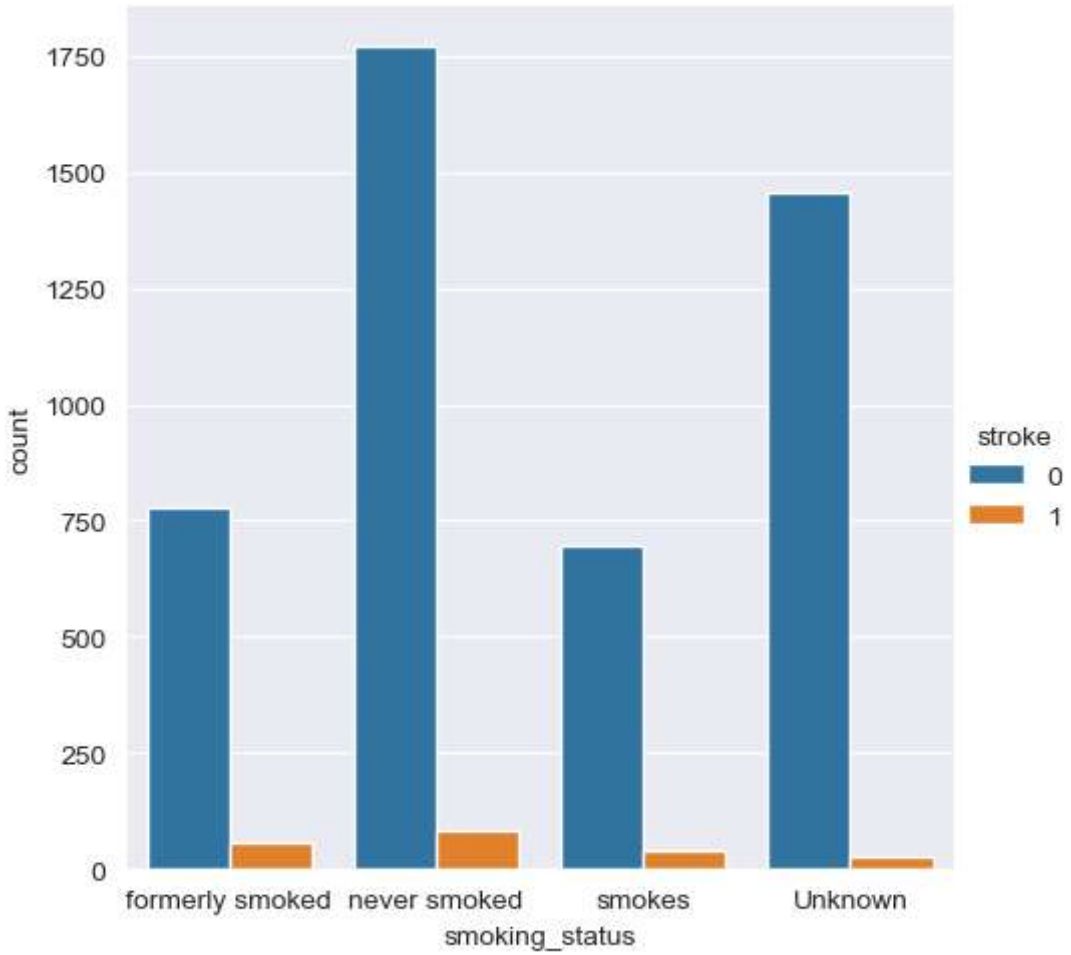
Out[33]:
```
<seaborn.axisgrid.FacetGrid at 0x232294e5b90>
```


2500
2000
1500
1000
500
0
count
Male
Female
Other
gender
stroke
0
1


3000
2500
2000
1500
1000
500
0
count
Yes
No
ever_married
stroke
0
1


2500
2000
1500
1000
500
0
count
Private
Self-employed
Govt_job
children
Never_worked
work_type
stroke
0
1


2500
2000
1500
1000
500
0
count
Urban
Rural
Residence_type
stroke
0
1



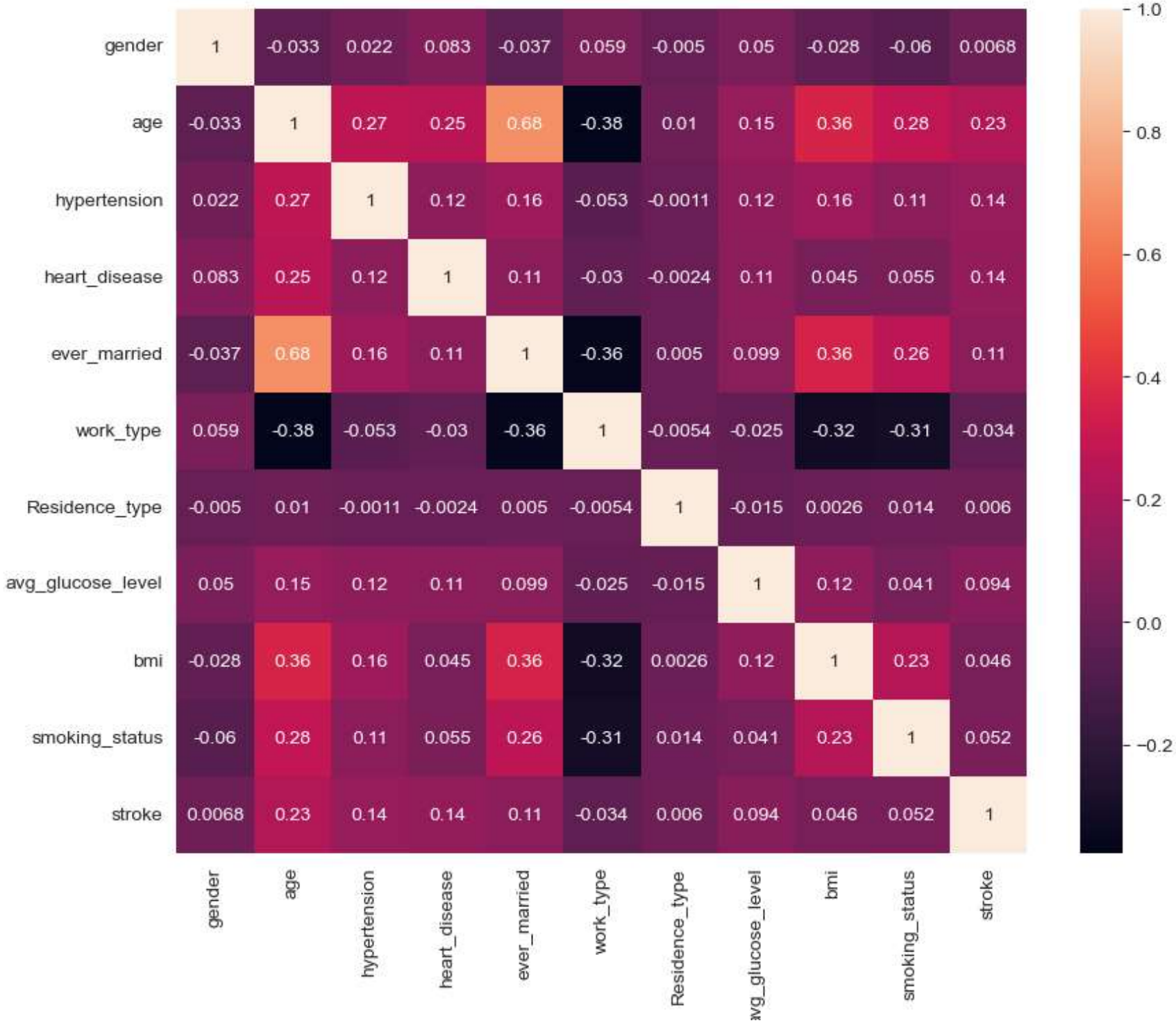
In [34]:
```
#Encoding of categorical variables
le = LabelEncoder()
data = data.apply(le.fit_transform)
data.head(3)
```

Out[34]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	88	0	1	1	2	1	3734	239	1	1
2	1	101	0	1	1	2	0	2429	198	2	1
3	0	70	0	0	1	2	1	3309	217	3	1

In [35]:
```
#Plotting the heat map for correlation
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(),annot=True)
```

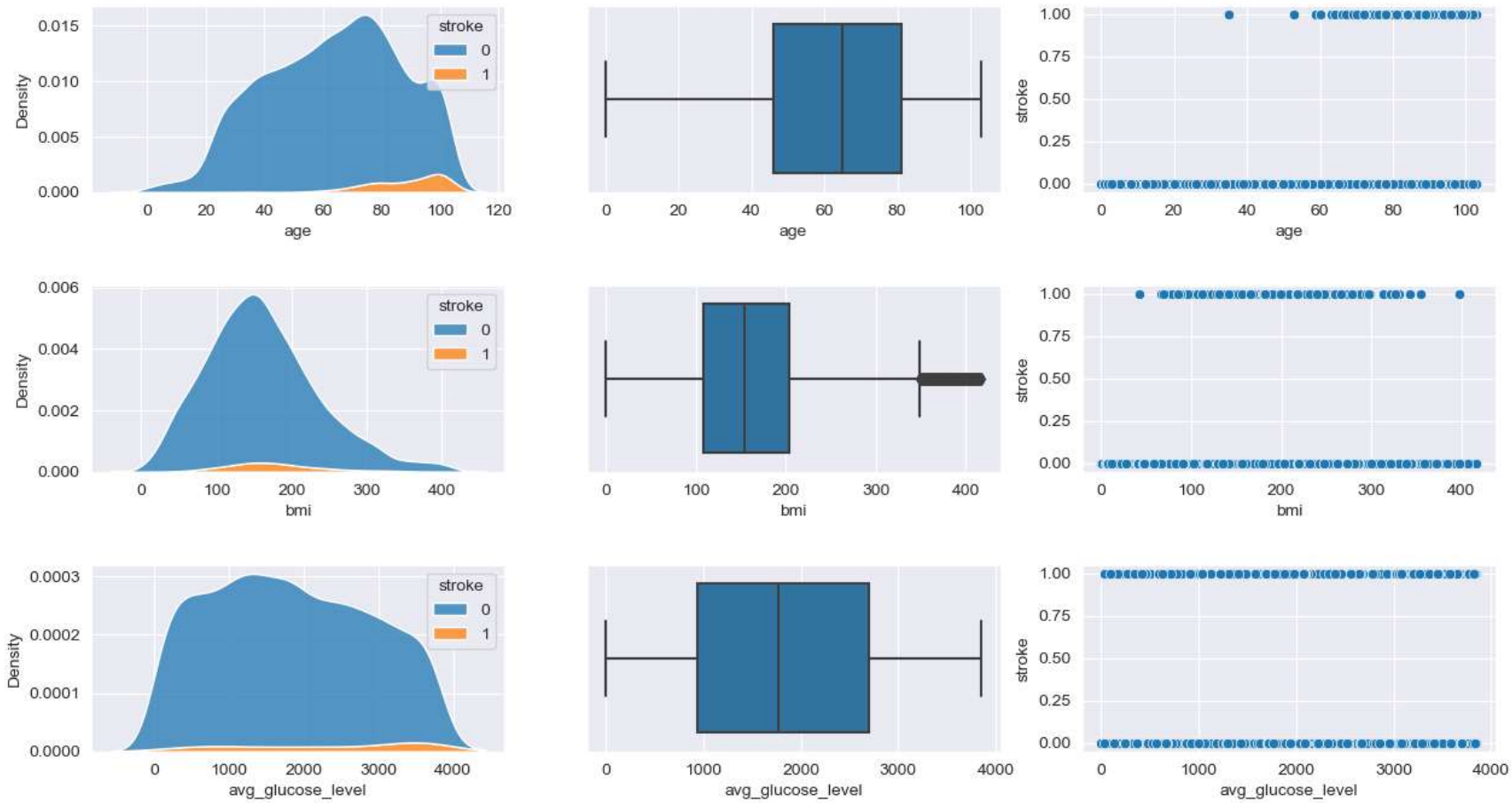
Out[35]:
```
<Axes: >
```


gender 1 -0.033 0.022 0.083 -0.037 0.059 -0.005 0.05 -0.028 -0.06 0.0068
age -0.033 1 0.27 0.25 0.68 -0.38 0.01 0.15 0.36 0.28 0.23
hypertension 0.022 0.27 1 0.12 0.16 -0.053 -0.0011 0.12 0.16 0.11 0.14
heart_disease 0.083 0.25 0.12 1 0.11 -0.03 -0.0024 0.11 0.045 0.055 0.14
ever_married -0.037 0.68 0.16 0.11 1 -0.36 0.005 0.099 0.36 0.26 0.11
work_type 0.059 -0.38 -0.053 -0.03 -0.36 1 -0.0054 -0.025 -0.32 -0.31 -0.034
Residence_type -0.005 0.01 -0.0011 -0.0024 0.005 -0.0054 1 -0.015 0.0026 0.014 0.006
avg_glucose_level 0.05 0.15 0.12 0.11 0.099 -0.025 -0.015 1 0.12 0.041 0.094
bmi -0.028 0.36 0.16 0.045 0.36 -0.32 0.0026 0.12 1 0.23 0.046
smoking_status -0.06 0.28 0.11 0.055 0.26 -0.31 0.014 0.041 0.23 1 0.052
stroke 0.0068 0.23 0.14 0.14 0.11 -0.034 0.006 0.094 0.046 0.052 1
gender
age
hypertension
heart_disease
ever_married
work_type
Residence_type
avg_glucose_level
bmi
smoking_status
stroke
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

- Age and Ever married columns has the highest correlation of 0.68
- bmi and Ever married columns has the highest correlation of 0.35

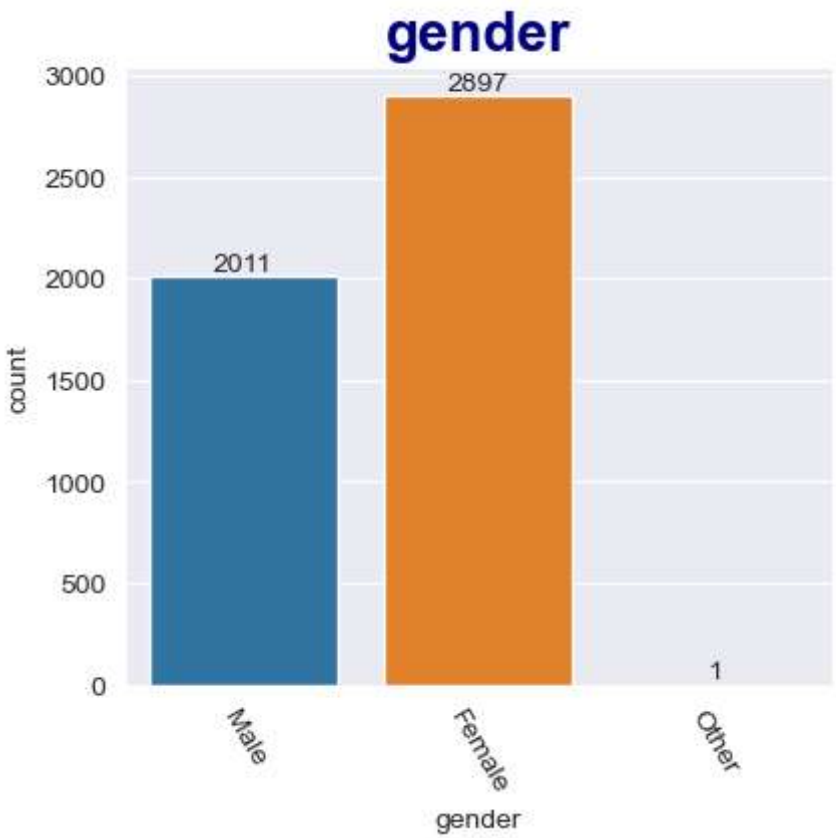
- BMI and avg_glucose_level attributes have the outliers from the boxplots

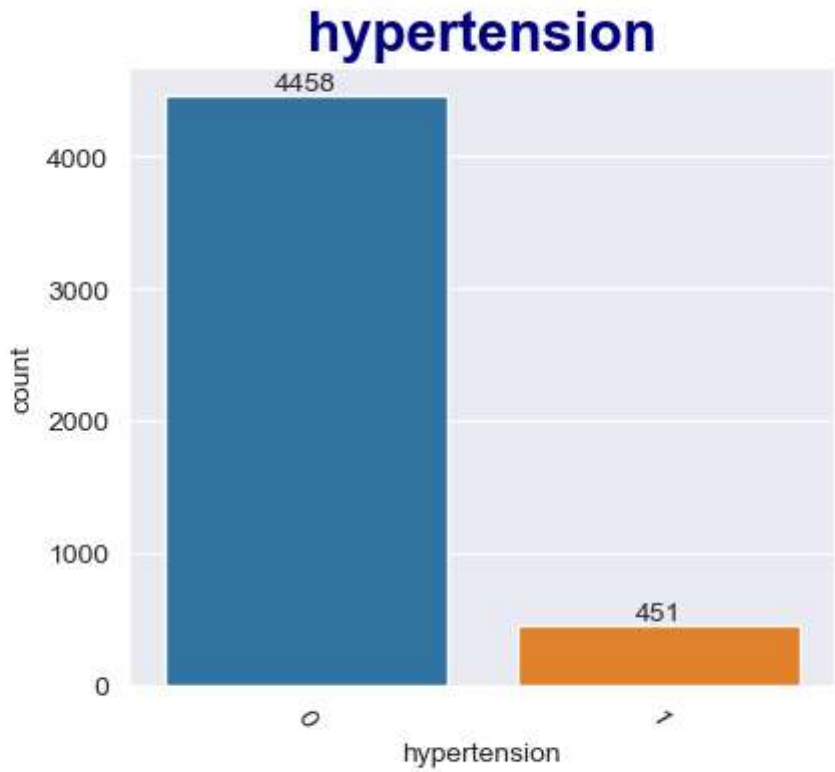
In [37]:
```
i = 0
fig, ax = plt.subplots(3, 3, figsize=(15, 8))
plt.subplots_adjust(hspace = 0.5)
for num_col in numeric_columns :
    sns.kdeplot(x=num_col, hue='stroke', data=data, multiple='stack', ax=ax[i,0])
    sns.boxplot(x=num_col, data=data, ax=ax[i, 1])
    sns.scatterplot(x=num_col, y='stroke', data=data, ax=ax[i, 2])
    i+=1
plt.show()
```

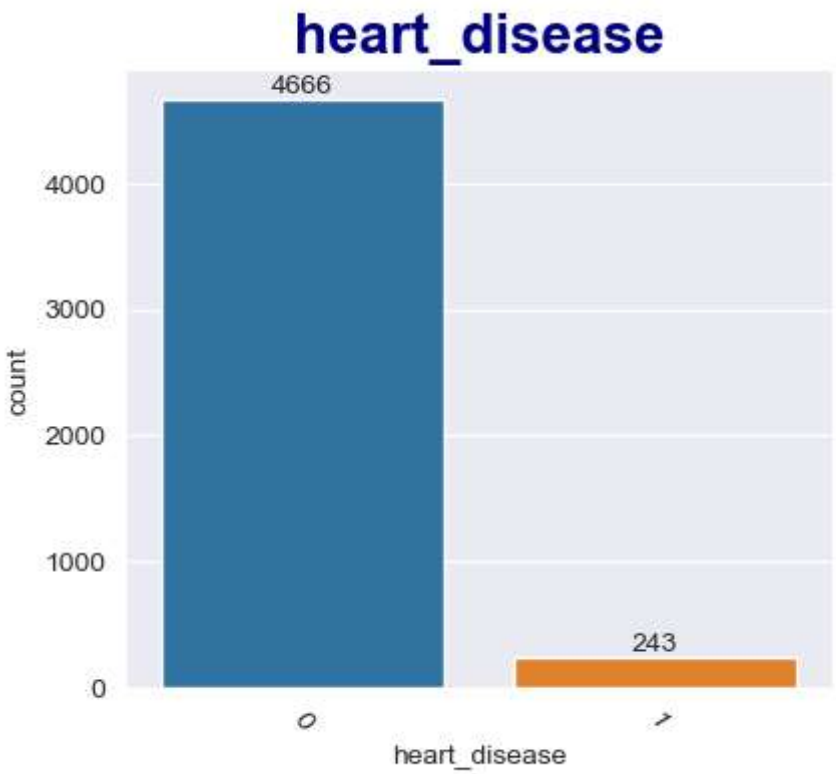


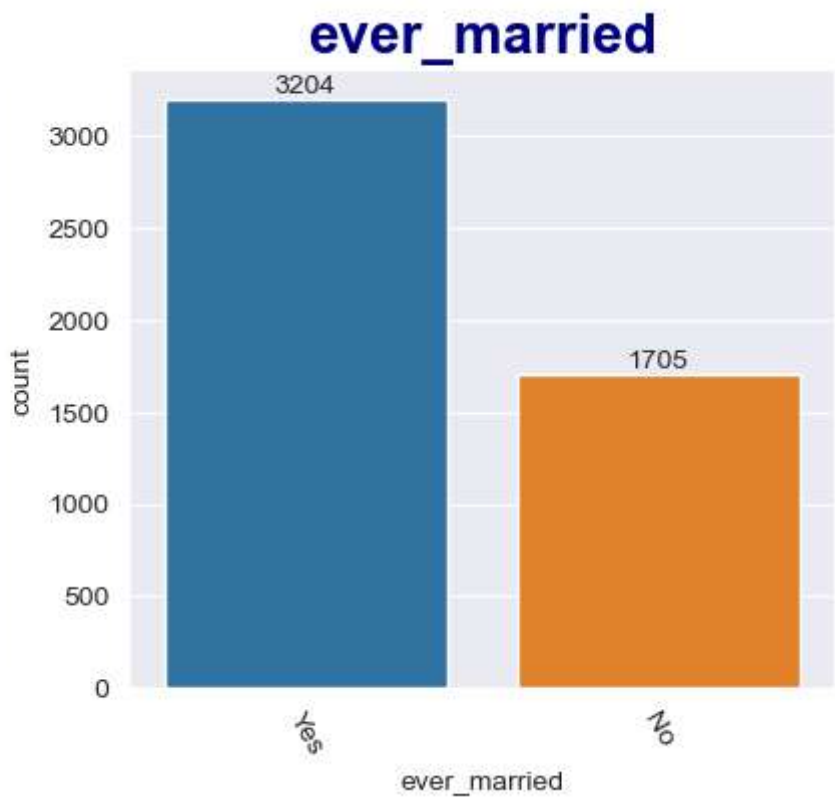
In [13]:
```
i=0
while i<8 :

    # Left AX
    fig = plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.title(categorical_columns[i], size=20, weight='bold', color='navy')
    ax = sns.countplot(x=categorical_columns[i], data=data)
    ax.bar_label(ax.containers[0])
    ax.tick_params(axis='x', rotation=300)
    i+=1

    # Right AX
    plt.subplot(1, 2, 2)
    plt.title(categorical_columns[i], size=20, weight='bold', color='navy')
    ax = sns.countplot(x=categorical_columns[i], data=data)
```

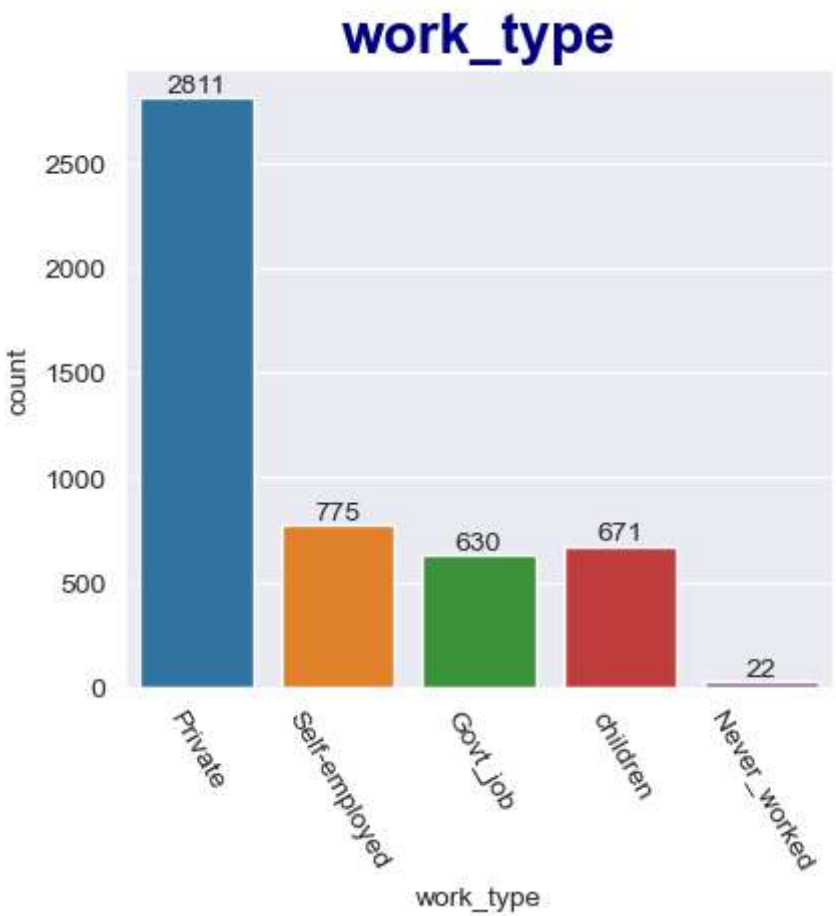
```
    ax.bar_label(ax.containers[0])
    ax.tick_params(axis='x', rotation=300)
    i+=1
    plt.show()
```

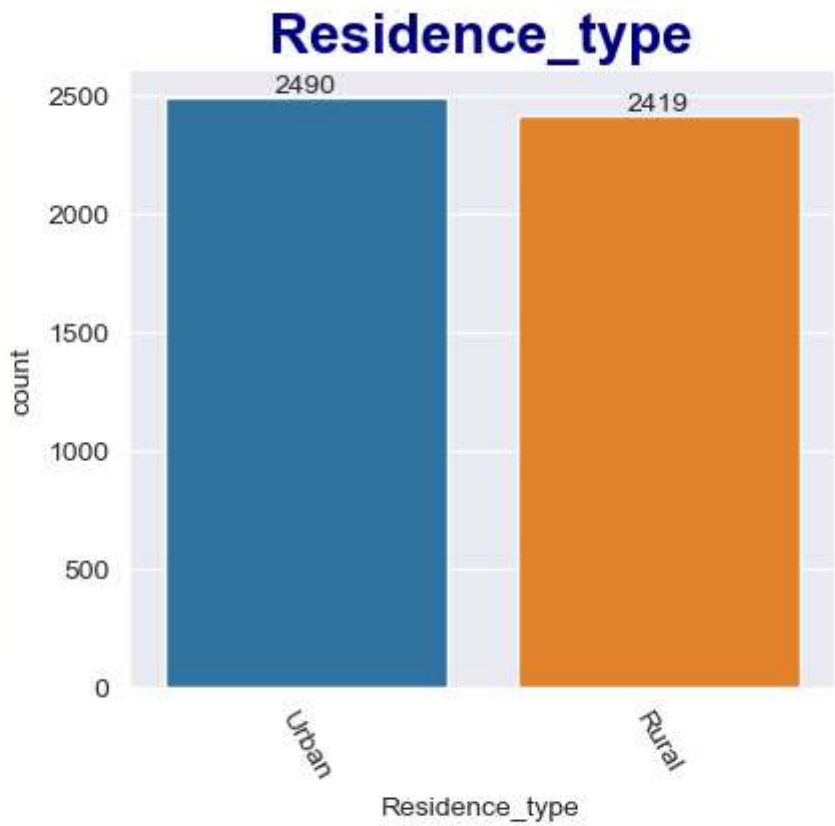


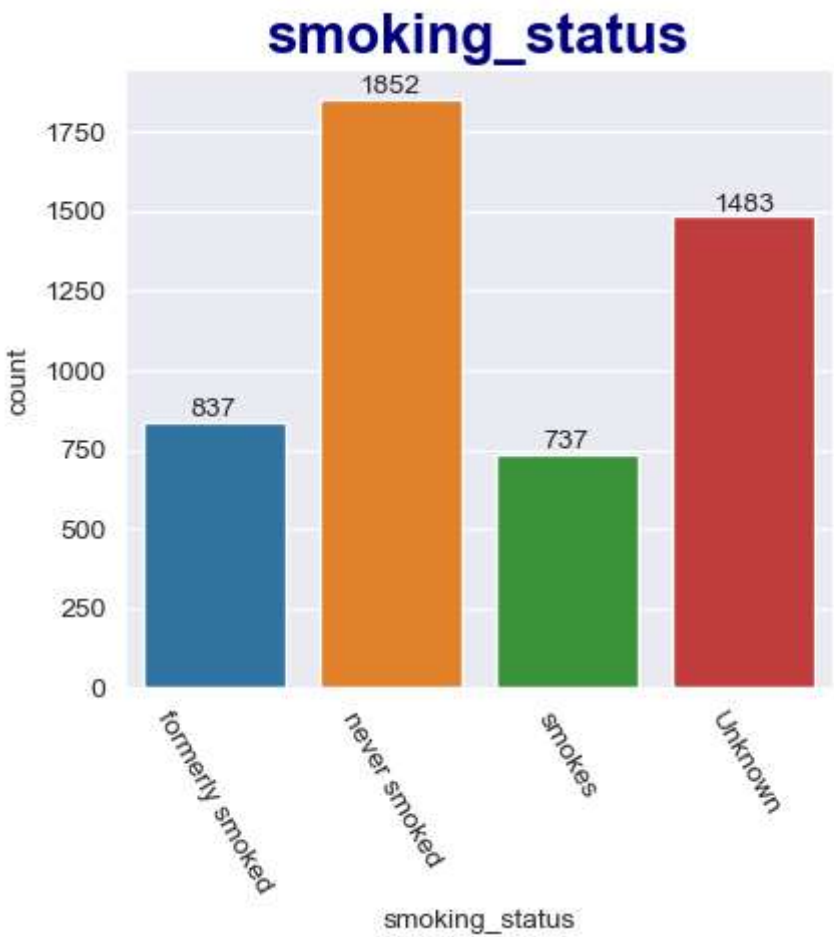


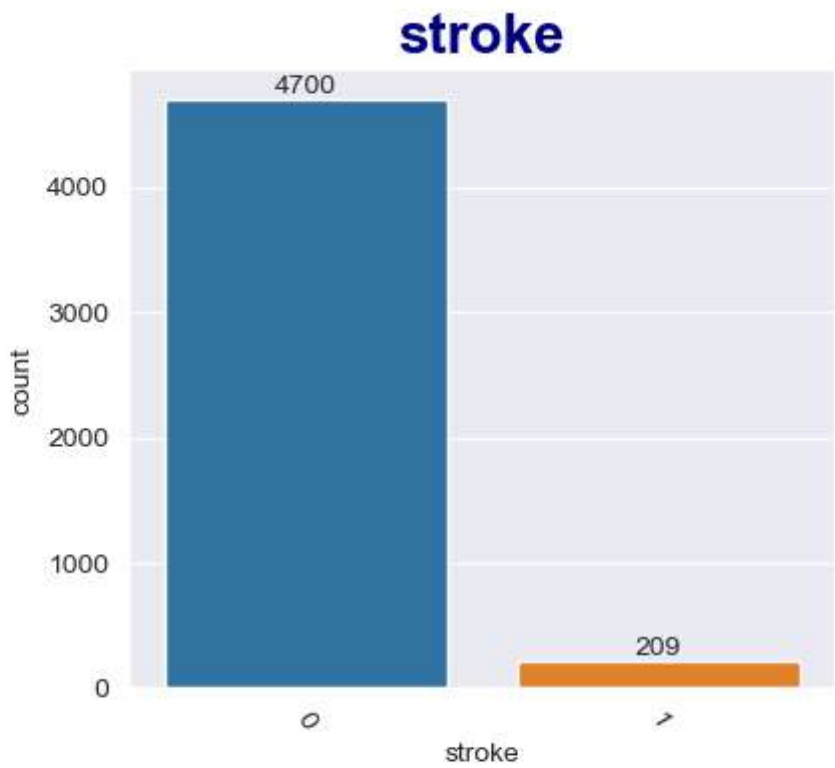

heart_disease
4666
243
4000
3000
2000
1000
0
count
0
1
heart_disease


ever_married
3204
1705
3000
2500
2000
1500
1000
500
0
count
Yes
No
ever_married


work_type
2811
775
630
671
22
2500
2000
1500
1000
500
0
count
Private
Self-employed
Govt_job
children
Never_worked
work_type

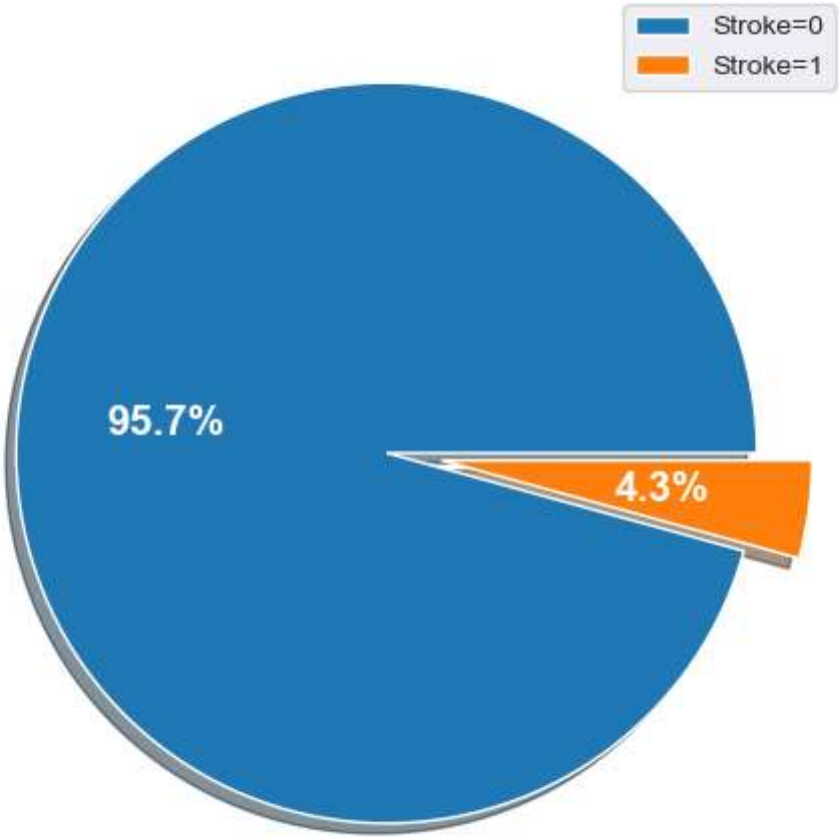

Residence_type
2490
2419
2500
2000
1500
1000
500
0
count
Urban
Rural
Residence_type





In [38]:

```
x = data['stroke'].value_counts()


explode = [0, 0.15]
labels = ['Stroke=0', 'Stroke=1']
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))

plt.pie(x, explode=explode, shadow=True, autopct='%1.1f%%', labels=labels, textprops=dict(color="w", weight='bold', siz
plt.legend()
plt.show()
```



Step 3.3 | Plots Analysis

Results :

1) About 96% of samples have not Stroke and 4% have stroke. 2) Distribution of samples is a Normal distribution. 3) Those who have had a stroke are in: age in range 40 to 85 bmi in range 20 to 40 glocuse level in range 50 to 130 4) About 60% of samples are female. 5) About 91% of samples dont have any hypertension. 6) About 95% of samples dont have any heart disease. 7) About 34% of samples never get married. 8) Most of samples worked in private. 9) We dont have any information in smoking field for 1483 of sapmples.

Step 3.4 | Unique Values

We count number of unique values in each categorical column, to change them with integer values. Here we use .unique() command.

In [39]:
```
columns_temp = ['gender', 'ever_married', 'work_type', 'smoking_status', 'Residence_type']

for col in columns_temp :
    print('column :', col)
    for index, unique in enumerate(data[col].unique()) :
        print(unique, ':', index)
    print('_'*45)
```

```
column : gender
1 : 0
0 : 1
2 : 2
_____________________________________________
column : ever_married
1 : 0
0 : 1
_____________________________________________
column : work_type
2 : 0
3 : 1
0 : 2
4 : 3
1 : 4
_____________________________________________
column : smoking_status
1 : 0
2 : 1
3 : 2
0 : 3
_____________________________________________
column : Residence_type
1 : 0
0 : 1
_____________________________________________
```

In [40]:
```
# gender
data_2 = data.replace(
```

```
    {'gender' : {'Male' : 0, 'Female' : 1, 'Other' : 2}}
)

# ever_married
data_2 =  data_2.replace(
    {'ever_married' : {'Yes' : 0, 'No' : 1}}
)

# work_type
data_2 =  data_2.replace(
    {'work_type' : {'Private' : 0, 'Self-employed' : 1, 'Govt_job' : 2, 'children' : 3, 'Never_worked' : 4}}
)

# smoking_status
data_2 =  data_2.replace(
    {'smoking_status' : {'formerly smoked' : 0, 'never smoked' : 1, 'smokes' : 2, 'Unknown' : 3}}
)

# Residence_type
data_2 =  data_2.replace(
    {'Residence_type' : {'Urban' : 0, 'Rural' : 1}}
)
```

In [41]:
```
data_2.head()
```

Out[41]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	88	0	1	1	2	1	3734	239	1	1
2	1	101	0	1	1	2	0	2429	198	2	1
3	0	70	0	0	1	2	1	3309	217	3	1
4	0	100	1	0	1	3	0	3318	113	2	1
5	1	102	0	0	1	2	1	3367	163	1	1

Step 3.5 | Normalization

➡️ Define X & y

In [42]:
```
X_temp = data_2.drop(columns='stroke')
y = data_2.stroke
```

➡️ To decrease effect of larg values, we use MinMaxScaler to normalize X.

In [43]:
```
scaler = MinMaxScaler().fit_transform(X_temp)
X = pd.DataFrame(scaler, columns=X_temp.columns)
X.describe()
```

Out[43]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smo
count	4909.000000	4909.000000	4909.000000	4909.000000	4909.000000	4909.000000	4909.000000	4909.000000	4909.000000	4
mean	0.205032	0.617912	0.091872	0.049501	0.652679	0.542524	0.507232	0.474533	0.386305	
std	0.246154	0.223542	0.288875	0.216934	0.476167	0.273148	0.499999	0.277768	0.180121	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.446602	0.000000	0.000000	0.000000	0.500000	0.000000	0.244612	0.258993	
50%	0.000000	0.631068	0.000000	0.000000	1.000000	0.500000	1.000000	0.459102	0.369305	
75%	0.500000	0.786408	0.000000	0.000000	1.000000	0.750000	1.000000	0.702155	0.489209	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Step 4 | Modeling

Step 4.1 | Initialization

In [44]:
```
# define a function to ploting Confusion matrix
def plot_confusion_matrix(y_test, y_prediction):
    cm = metrics.confusion_matrix(y_test, y_prediction)
    ax = plt.subplot()
    ax = sns.heatmap(cm, annot=True, fmt='', cmap="Greens")
    ax.set_xlabel('Prediced labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix')
    ax.xaxis.set_ticklabels(['Dont Had Stroke', 'Had Stroke'])
    ax.yaxis.set_ticklabels(['Dont Had Stroke', 'Had Stroke'])
    plt.show()
```

In [45]:
```
# Splite X, y to train & test dataset.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.25)
```

Step 4.2 | RandomForestClassifier

In [46]:
```
# a dictionary to define parameters to test in algorithm
parameters = {
    'n_estimators' : [50, 100, 250, 500],
    'criterion' : ['gini', 'entropy', 'log_loss'],
    'max_features' : ['sqrt', 'log2']
}

rf = RandomForestClassifier(n_jobs=-1)
rf_cv = GridSearchCV(estimator=rf, cv=10, param_grid=parameters).fit(X_train, y_train)

print('Tuned hyper parameters : ', rf_cv.best_params_)
print('accuracy : ', rf_cv.best_score_)
```

```
Tuned hyper parameters :  {'criterion': 'entropy', 'max_features': 'sqrt', 'n_estimators': 500}
accuracy :  0.955990780016496
```

In [47]:
```
# calculate time befor run algorithm
t1 = datetime.now()
# Model :
rf = RandomForestClassifier(**rf_cv.best_params_).fit(X_train, y_train)
# calculate time after run algorithm
t2 = datetime.now()
```

In [48]:
```
y_pred_rf = rf.predict(X_test)
```

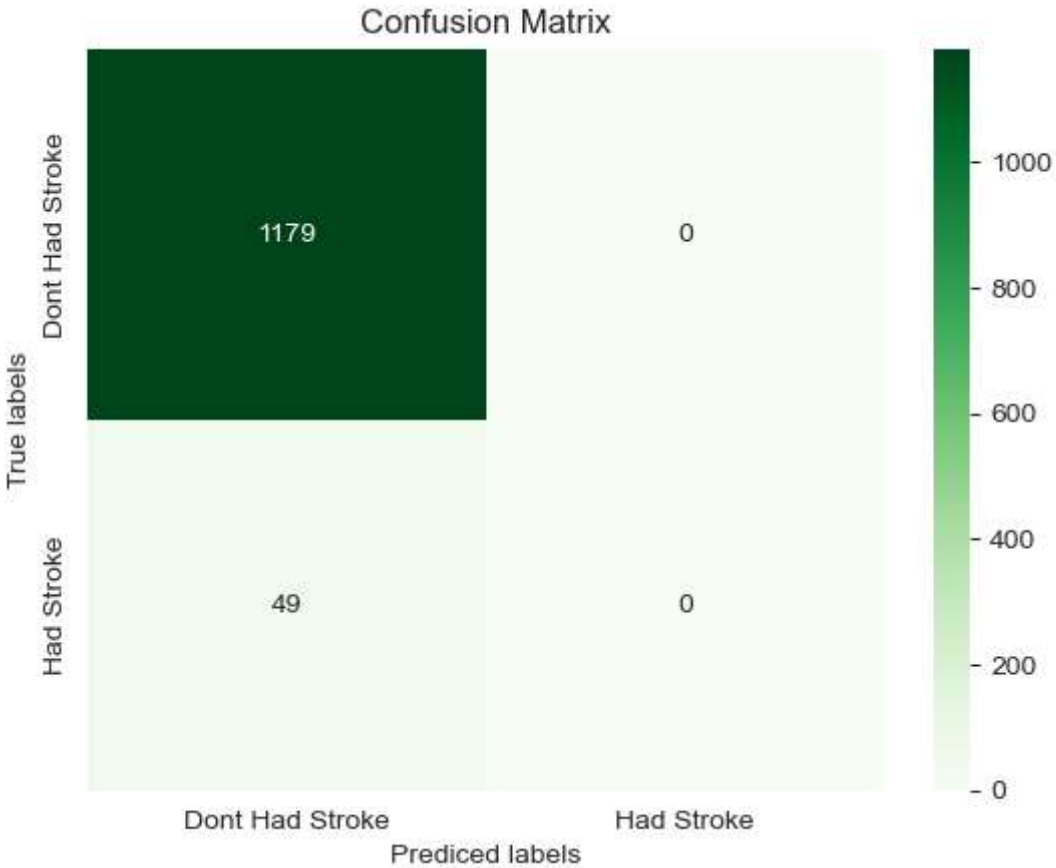
```
rf_score = round(rf.score(X_test, y_test), 3)
print('RandomForestClassifier score : ', rf_score)
```

```
RandomForestClassifier score :  0.96
```

In [49]:
```
delta = t2-t1
delta_rf = round(delta.total_seconds(), 3)
print('RandomForestClassifier takes : ', delta_rf, 'Seconds')
```

```
RandomForestClassifier takes :  2.97 Seconds
```

In [50]:
```
plot_confusion_matrix(y_test, y_pred_rf)
```



In [51]:
```
cr = metrics.classification_report(y_test, y_pred_rf)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1179
           1       0.00      0.00      0.00        49

    accuracy                           0.96      1228
   macro avg       0.48      0.50      0.49      1228
weighted avg       0.92      0.96      0.94      1228
```

Step 4.3 | LogisticRegression

In [52]:
```
# a dictionary to define parameters to test in algorithm
parameters = {
    'C' : [0.001, 0.01, 0.1, 1.0, 10, 100, 1000],
    'class_weight' : ['balanced'],
    'solver' : ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']
}

lr = LogisticRegression()
lr_cv = GridSearchCV(estimator=lr, param_grid=parameters, cv=10).fit(X_train, y_train)

print('Tuned hyper parameters : ', lr_cv.best_params_)
print('accuracy : ', lr_cv.best_score_)
```

```
Tuned hyper parameters :  {'C': 100, 'class_weight': 'balanced', 'solver': 'saga'}
accuracy :  0.7481589489807942
```

In [53]:
```
# Calculate time befor run algorithm
t1 = datetime.now()
# Model
lr = LogisticRegression(**lr_cv.best_params_).fit(X_train, y_train)
# Calculate time after run algorithm
t2 = datetime.now()
```

In [54]:
```
y_pred_lr = lr.predict(X_test)

lr_score = round(lr.score(X_test, y_test), 3)
print('LogisticRegression score : ', lr_score)
```

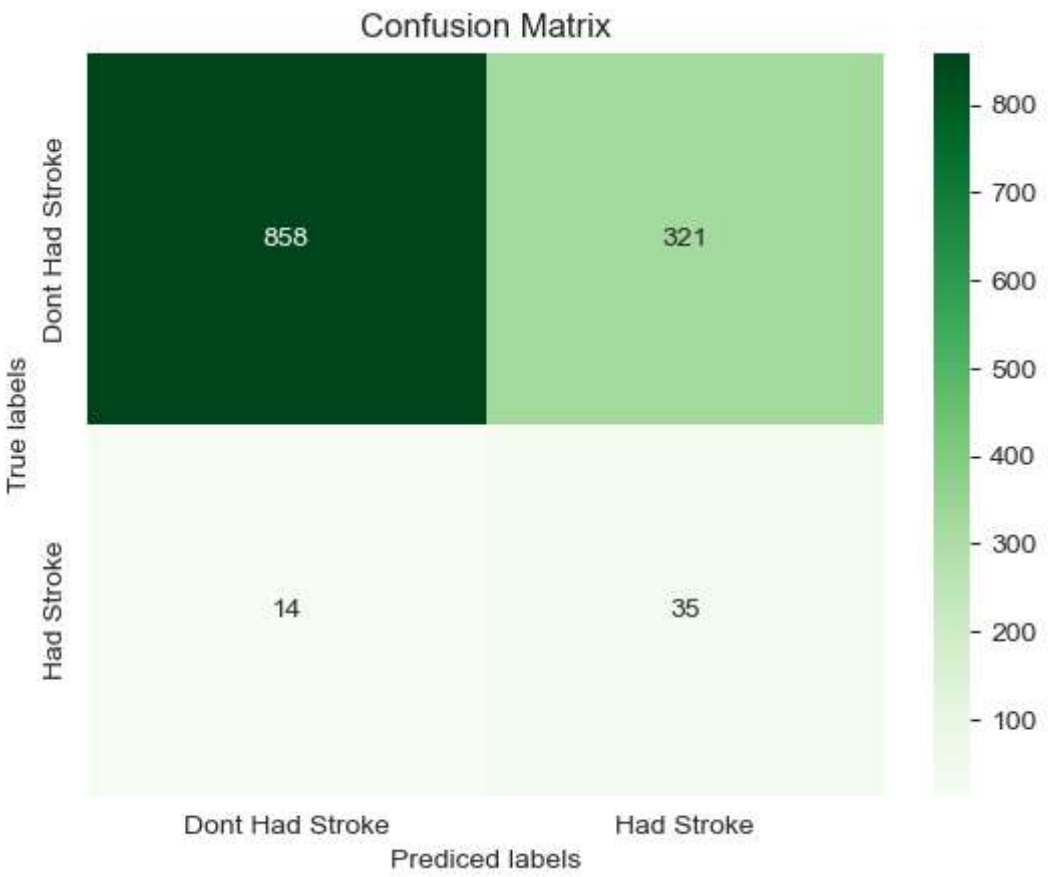
```
LogisticRegression score :  0.723
```

In [55]:
```
delta = t2-t1
delta_lr = round(delta.total_seconds(), 3)
```

```
print('LogisticRegression takes : ', delta_lr, 'Seconds')
```

```
LogisticRegression takes :  0.059 Seconds
```

In [32]:
```
plot_confusion_matrix(y_test, y_pred_lr)
```



In [56]:
```
cr = metrics.classification_report(y_test, y_pred_lr)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.99      0.72      0.83      1179
           1       0.11      0.80      0.19        49

    accuracy                           0.72      1228
   macro avg       0.55      0.76      0.51      1228
weighted avg       0.95      0.72      0.81      1228
```

Step 4.4 | SVC

In [57]:
```
# a dictionary to define parameters to test in algorithm
parameters = {
    'C' : [0.001, 0.01, 0.1, 1.0, 10, 100, 1000],
    'gamma' : [0.001, 0.01, 0.1, 1.0, 10, 100, 1000],
}



svc = SVC()
svc_cv = GridSearchCV(estimator=svc, param_grid=parameters, cv=10).fit(X_train, y_train)



print('Tuned hyper parameters : ', svc_cv.best_params_)
print('accuracy : ', svc_cv.best_score_)
```

```
Tuned hyper parameters :  {'C': 0.001, 'gamma': 0.001}
accuracy :  0.9565335218569577
```

In [58]:
```
# Calculate time befor run algorithm
t1 = datetime.now()
# Model
svc = SVC(**svc_cv.best_params_).fit(X_train, y_train)
# Calculate time after run algorithm
t2 = datetime.now()
```

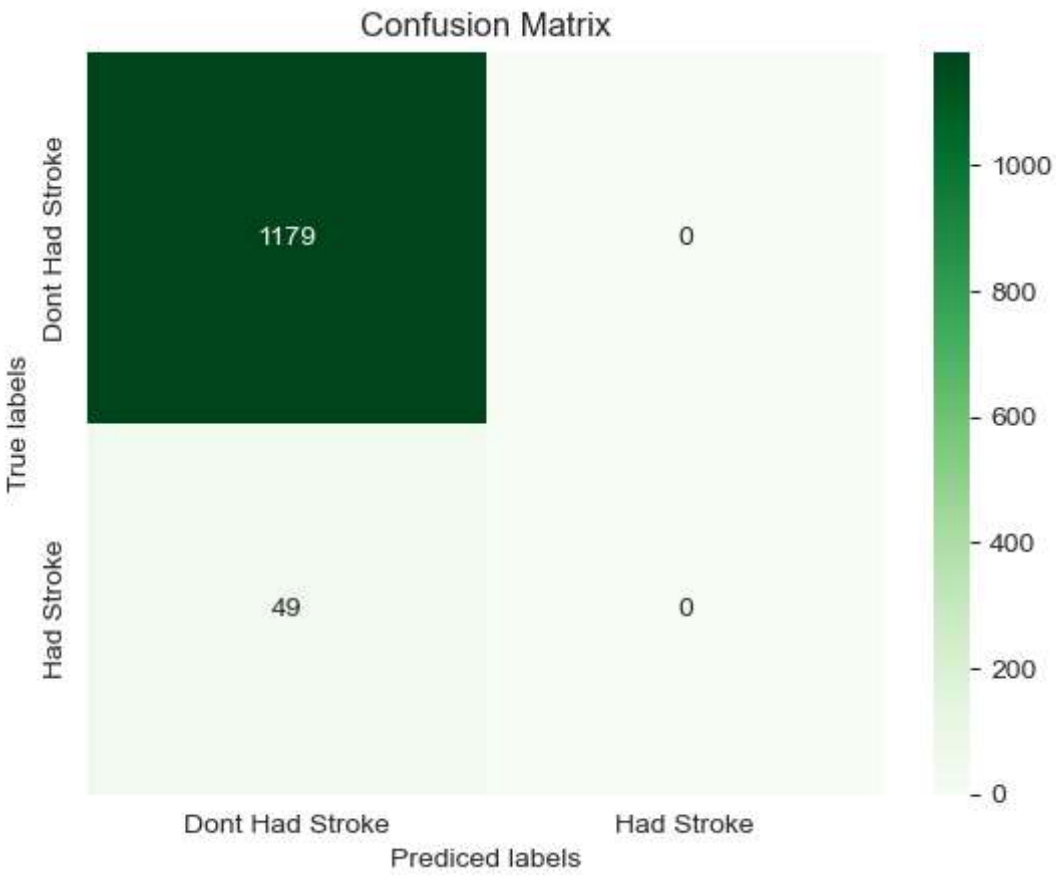
In [59]:
```
y_pred_svc = svc.predict(X_test)

svc_score = round(svc.score(X_test, y_test), 3)
print('SVC Score : ', svc_score)
```

```
SVC Score :  0.96
```

In [60]:
```
delta = t2-t1
delta_svc = round(delta.total_seconds(), 3)
print('SVC : ', delta_svc, 'Seconds')
```

```
SVC :  0.074 Seconds
```

In [61]:
```
plot_confusion_matrix(y_test, y_pred_svc)
```



In [62]:
```
cr = metrics.classification_report(y_test, y_pred_svc)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1179
           1       0.00      0.00      0.00        49

    accuracy                           0.96      1228
   macro avg       0.48      0.50      0.49      1228
weighted avg       0.92      0.96      0.94      1228
```

Step 4.5 | DecisionTreeClassifier

In [63]:
```
# a dictionary to define parameters to test in algorithm
parameters = {
    'criterion' : ['gini', 'entropy', 'log_loss'],
    'splitter' : ['best', 'random'],
    'max_depth' : list(np.arange(4, 30, 1))
        }



tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(estimator=tree, cv=10, param_grid=parameters).fit(X_train, y_train)



print('Tuned hyper parameters : ', tree_cv.best_params_)
print('accuracy : ', tree_cv.best_score_)
```

```
Tuned hyper parameters :  {'criterion': 'gini', 'max_depth': 4, 'splitter': 'random'}
accuracy :  0.9570762636974196
```

In [64]:
```
# Calculate time befor run algorithm :
t1 = datetime.now()
# Model :
tree = DecisionTreeClassifier(**tree_cv.best_params_).fit(X_train, y_train)
# Calculate time after run algorithm :
t2 = datetime.now()
```

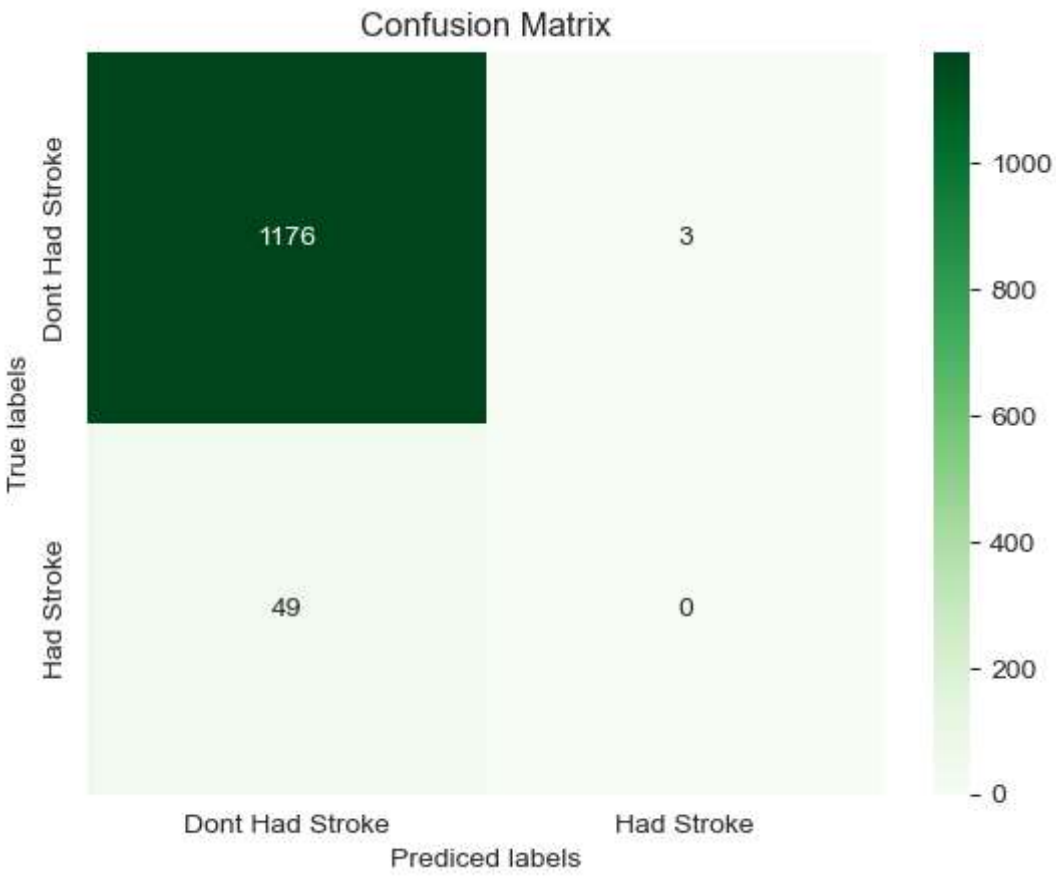
In [65]:
```
y_pred_tree = tree.predict(X_test)

tree_score = round(tree.score(X_test, y_test), 3)
print('DecisionTreeClassifier Score : ', tree_score)
```

```
DecisionTreeClassifier Score :  0.958
```

In [66]:
```
delta = t2-t1
delta_tree = round(delta.total_seconds(), 3)
print('DecisionTreeClassifier takes : ', delta_tree, 'Seconds')
```

```
DecisionTreeClassifier takes :  0.004 Seconds
```

In [67]:
```
plot_confusion_matrix(y_test, y_pred_tree)
```



In [68]:
```
cr = metrics.classification_report(y_test, y_pred_tree)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1179
           1       0.00      0.00      0.00        49

    accuracy                           0.96      1228
   macro avg       0.48      0.50      0.49      1228
weighted avg       0.92      0.96      0.94      1228
```

Step 4.6 | KNeighborsClassifier

In [69]:
```
# a dictionary to define parameters to test in algorithm
parameters = {
    'n_neighbors' : list(np.arange(3, 20, 2)),
    'p' : [1, 2, 3, 4]
}

# calculate time to run in second
t1 = datetime.now()

knn = KNeighborsClassifier()
knn_cv = GridSearchCV(estimator=knn, cv=10, param_grid=parameters).fit(X_train, y_train)

t2 = datetime.now()

print('Tuned hyper parameters : ', knn_cv.best_params_)
print('accuracy : ', knn_cv.best_score_)
```

```
Tuned hyper parameters :  {'n_neighbors': 11, 'p': 1}
accuracy :  0.9568045245669848
```

In [70]:
```
# Calculate time befor run algorithm :
t1 = datetime.now()
# Model :
knn = KNeighborsClassifier(**knn_cv.best_params_).fit(X_train, y_train)
# Calculate time after run algorithm :
t2 = datetime.now()
```

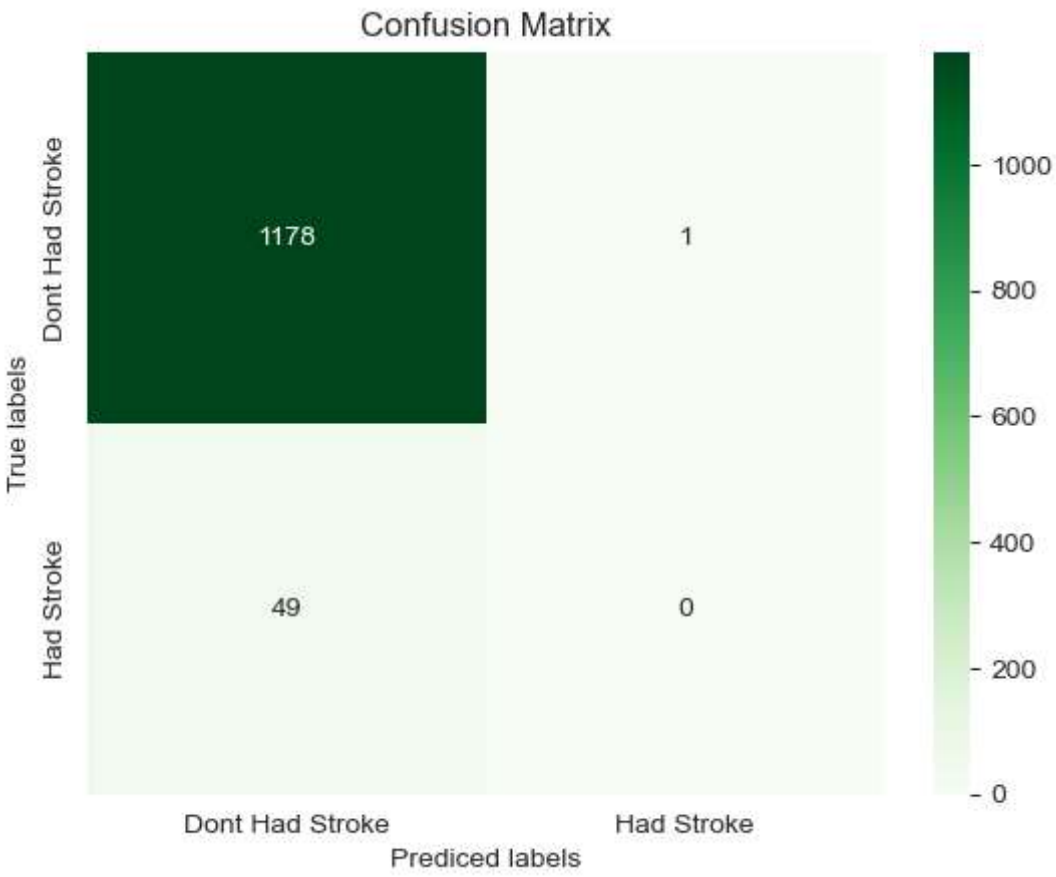
In [71]:
```
y_pred_knn = knn_cv.predict(X_test)

knn_score = round(knn.score(X_test, y_test), 3)
print('KNeighborsClassifier Score :', knn_score)
```

```
KNeighborsClassifier Score : 0.959
```

In [72]:
```
delta = t2-t1
delta_knn = round(delta.total_seconds(), 3)
print('KNeighborsClassifier takes : ', delta_knn, 'Seconds')
```

```
KNeighborsClassifier takes :  0.009 Seconds
```

In [73]:
```
plot_confusion_matrix(y_test, y_pred_knn)
```



In [74]:
```
cr = metrics.classification_report(y_test, y_pred_knn)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1179
           1       0.00      0.00      0.00        49

    accuracy                           0.96      1228
   macro avg       0.48      0.50      0.49      1228
weighted avg       0.92      0.96      0.94      1228
```

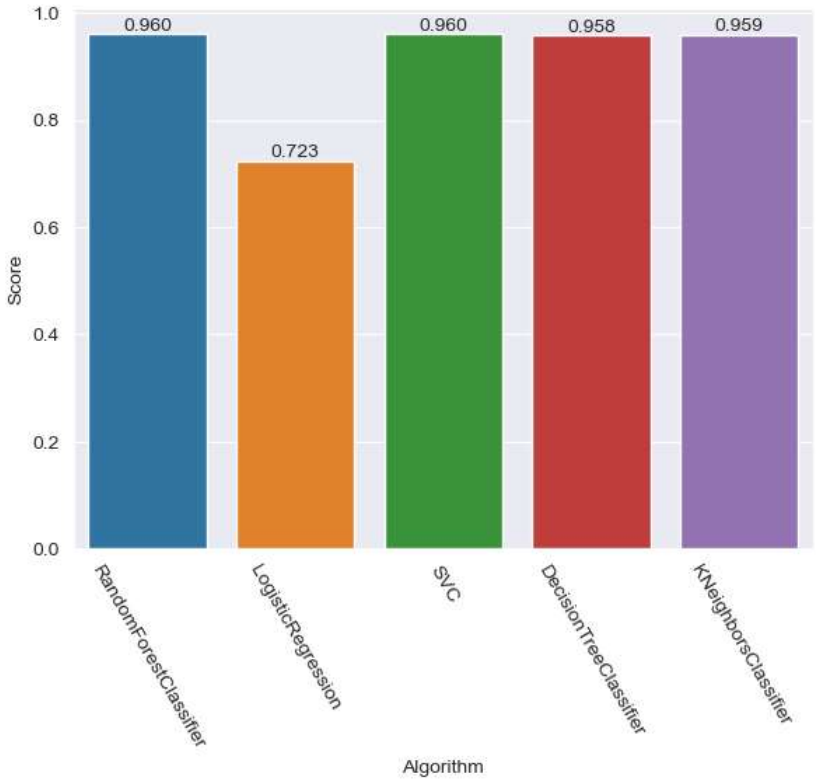
Step 4.7 | Result

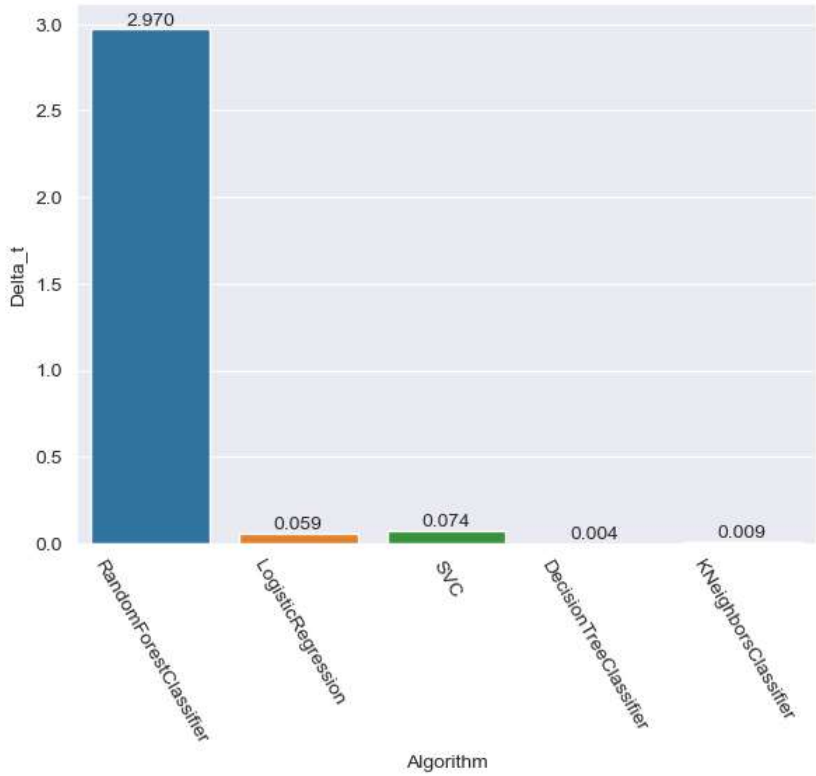
In [75]:
```
result = pd.DataFrame({
    'Algorithm' : ['RandomForestClassifier', 'LogisticRegression', 'SVC', 'DecisionTreeClassifier', 'KNeighborsClassifi
    'Score' : [rf_score, lr_score, svc_score, tree_score, knn_score],
    'Delta_t' : [delta_rf, delta_lr, delta_svc, delta_tree, delta_knn]
})

result
```

Out[75]:

	Algorithm	Score	Delta_t
0	RandomForestClassifier	0.960	2.970
1	LogisticRegression	0.723	0.059
2	SVC	0.960	0.074
3	DecisionTreeClassifier	0.958	0.004
4	KNeighborsClassifier	0.959	0.009

In [76]:
```
fig, ax = plt.subplots(1, 2, figsize=(15, 5))

sns.barplot(x='Algorithm', y='Score', data=result, ax=ax[0])
ax[0].bar_label(ax[0].containers[0], fmt='%.3f')
ax[0].set_xticklabels(labels=result.Algorithm, rotation=300)

sns.barplot(x='Algorithm', y='Delta_t', data=result, ax=ax[1])
ax[1].bar_label(ax[1].containers[0], fmt='%.3f')
ax[1].set_xticklabels(labels=result.Algorithm, rotation=300)
plt.show()
```





Results :

- Acording to the above plots, best algorithms base on Score are :
  - RandomForestClassifier
  - SVC
  - DecisionTreeClassifier
  - KNeighborsClassifier
- And best Algorithm base on runtime, are :
  - DecisionTreeClassifie
  - KNeighborsClassifier
- We choose KNeighborsClassifier

Step 4.8 | Final Modeling

In [77]:
```
knn = KNeighborsClassifier(**knn_cv.best_params_).fit(X, y)
knn
```

Out[77]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=11, p=1)
```

In [78]:
```
knn.score(X, y)
```

Out[78]:
```
0.9576288449786107
```

Conclusion

In these experiment, we find that the champion model is the KNeighborsClassifier, with the higher than 90% of accuracy.

In [ ]: