# STROKE PREDICTION PROJECT

# CA2 ADVANCE PYTHON PROGRAMMING(CSC 580)

# UNIQUE ID:E7321004

## PROJECT PROPOSAL:

```
Heart Disease and Strokes have rapidly increased globally even at juvenile ages.Stroke prediction is a complex t
ask
requiring huge amount of Data pre-processing.In this proposed model heart stroke prdiction dataset has taken fro
m
kaggle.The model predicts the chances of a person will have stroke based on the symptoms.It classifies the perso
n's
risk level by implementing various machine learning algorithms.In this project we have used logistic Regression
 to
predict the model accuracy.
```

In [4]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [5]:
```
s=pd.read_csv("healthcare_stroke_dataset.csv")
```

In [6]:
```
s
```
Out[6]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **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

# NARRATIVE (PROBLEM STATEMENT)

The Following python code employs Logistic Regression Classifier by using the scikit-learn library APIs.In this project we utilize the healthcare_stroke Dataset which exist in kaggle.The Dataset is Divided into Training and test Dataset and then classified by using the logistic Regression classifier.The Classification Accuracy,precision,recall,F1 Scoreare Calculated. The Classification report and Confusion matrix are also given in the model.

## Dataset Description

The dataset has been taken from the Kaggle website. It has 5110 rows and 12 columns. The features or attributes include id, gender, age, hypertension, heart disease, ever married, work type, residence type, average glucose level, body mass index (BMI), smoking status. The label or the outcome is stroke. Excluding the ID, all the other features have been used for training the model. The independent attributes are stored in the X variable while the dependent attribute which the stroke attribute is stored in the y variable of the dataset.

In [7]:
```
s.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

In [8]:
```
s.describe()
```
Out[8]:

| | id | age | hypertension | heart_disease | avg_glucose_level | bmi | stroke |
|---|---|---|---|---|---|---|---|
| **count** | 5110.000000 | 5110.000000 | 5110.000000 | 5110.000000 | 5110.000000 | 4909.000000 | 5110.000000 |
| **mean** | 36517.829354 | 43.226614 | 0.097456 | 0.054012 | 106.147677 | 28.893237 | 0.048728 |
| **std** | 21161.721625 | 22.612647 | 0.296607 | 0.226063 | 45.283560 | 7.854067 | 0.215320 |
| **min** | 67.000000 | 0.080000 | 0.000000 | 0.000000 | 55.120000 | 10.300000 | 0.000000 |
| **25%** | 17741.250000 | 25.000000 | 0.000000 | 0.000000 | 77.245000 | 23.500000 | 0.000000 |
| **50%** | 36932.000000 | 45.000000 | 0.000000 | 0.000000 | 91.885000 | 28.100000 | 0.000000 |
| **75%** | 54682.000000 | 61.000000 | 0.000000 | 0.000000 | 114.090000 | 33.100000 | 0.000000 |
| **max** | 72940.000000 | 82.000000 | 1.000000 | 1.000000 | 271.740000 | 97.600000 | 1.000000 |

In [9]:
```
s.head()
```
Out[9]:

| | 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 |

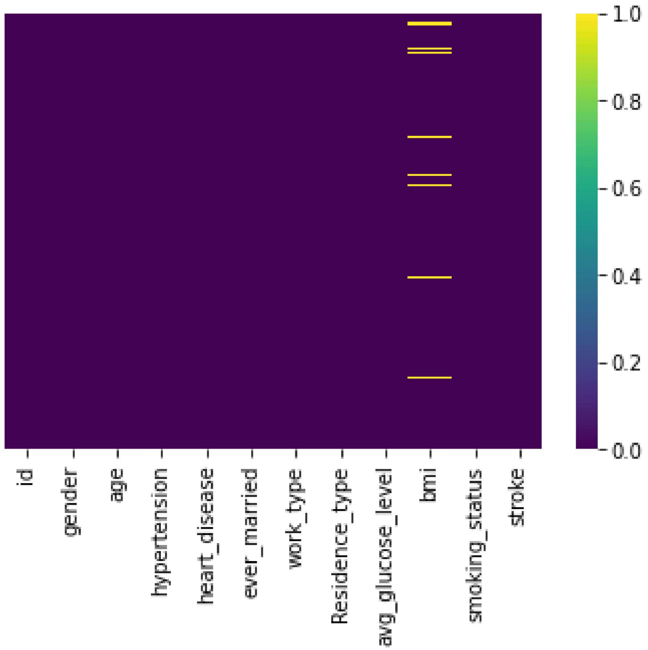
In [10]:
```
s["bmi"].isnull().sum()
```
Out[10]:
```
201
```

# Data Pre-processing

The dataset obtained contains 201 null values in the BMI attribute which needs to be removed.The cleaned pre-processed data.

In [11]:
```
import seaborn as sns
sns.heatmap(s.isnull(),yticklabels=False,cmap='viridis') #To Determine  the Null values
```

Out[11]: <AxesSubplot:>




In [13]:
```
s['bmi'].mean()
```

Out[13]: 28.893236911794673

In [14]:
```
p=s.fillna(s['bmi'].mean())
p
```

Out[14]:

| | id | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 9046 | Male | 67.0 | 0 | 1 | Yes | Private | Urban | 228.69 | 36.600000 | formerly smoked | |
| **1** | 51676 | Female | 61.0 | 0 | 0 | Yes | Self-employed | Rural | 202.21 | 28.893237 | never smoked | |
| **2** | 31112 | Male | 80.0 | 0 | 1 | Yes | Private | Rural | 105.92 | 32.500000 | never smoked | |
| **3** | 60182 | Female | 49.0 | 0 | 0 | Yes | Private | Urban | 171.23 | 34.400000 | smokes | |
| **4** | 1665 | Female | 79.0 | 1 | 0 | Yes | Self-employed | Rural | 174.12 | 24.000000 | never smoked | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **5105** | 18234 | Female | 80.0 | 1 | 0 | Yes | Private | Urban | 83.75 | 28.893237 | never smoked | |
| **5106** | 44873 | Female | 81.0 | 0 | 0 | Yes | Self-employed | Urban | 125.20 | 40.000000 | never smoked | |
| **5107** | 19723 | Female | 35.0 | 0 | 0 | Yes | Self-employed | Rural | 82.99 | 30.600000 | never smoked | |
| **5108** | 37544 | Male | 51.0 | 0 | 0 | Yes | Private | Rural | 166.29 | 25.600000 | formerly smoked | |
| **5109** | 44679 | Female | 44.0 | 0 | 0 | Yes | Govt_job | Urban | 85.28 | 26.200000 | Unknown | |

5110 rows × 12 columns

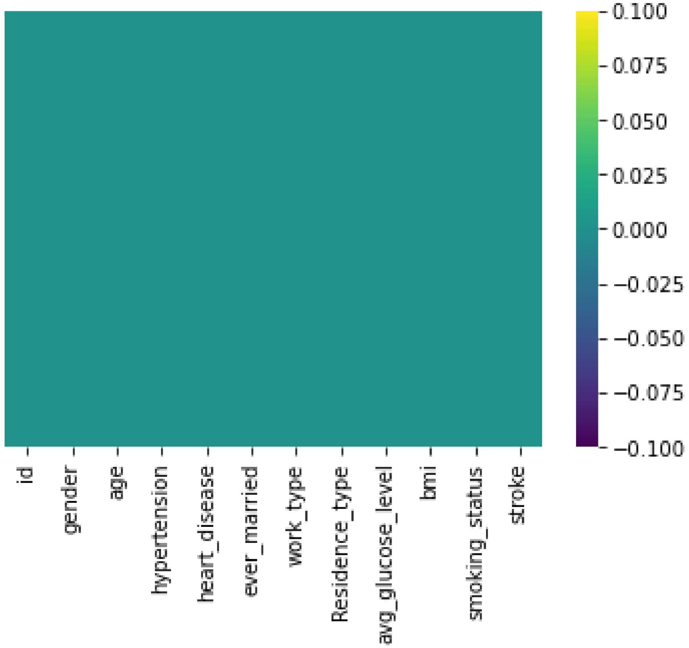
In [15]:
```
p.isnull().sum()
```

Out[15]:
```
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       0
stroke               0
dtype: int64
```

In [16]:
```
p.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                5110 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

In [17]:
```
sns.heatmap(p.isnull(),yticklabels=False,cmap='viridis')
```
Out[17]: <AxesSubplot:>




## Data Visualization

Data visualization helps to interpret the data easily through visual graphs or maps. Heatmaps are used to obtain the correlation between the attributes.Histogram plots are used to count the frequencies of smokers and non-smokers, number of females or males, the different work types of the people. Box plots have been used to indicate the relationship between two attributes and find out the outliers. All these plots give important insights about the data which can later be used for the modelling. It also shows which features are more important in making the most accurate prediction.

In [18]:
```
a=p.corr()
a
```

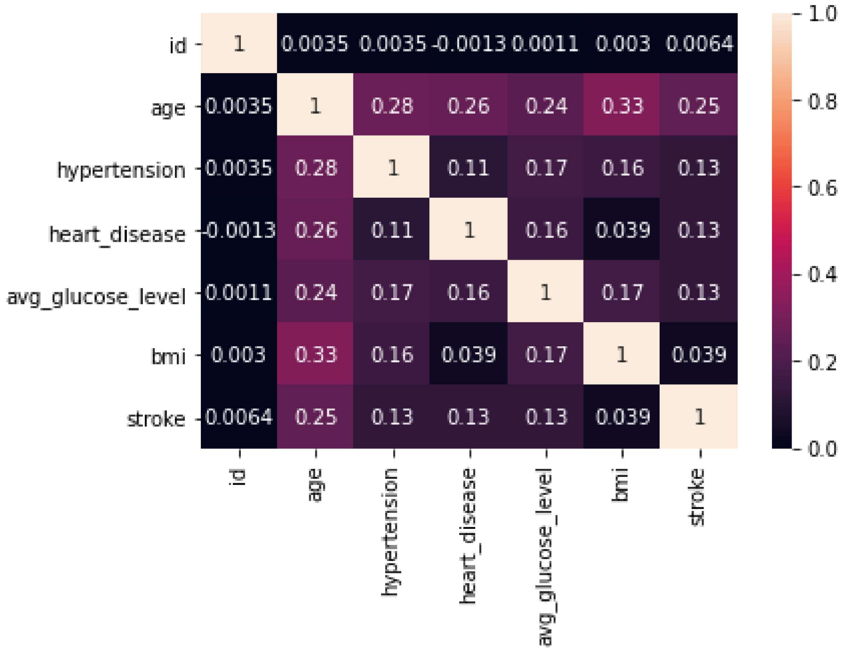
Out[18]:

| | id | age | hypertension | heart_disease | avg_glucose_level | bmi | stroke |
|---|---|---|---|---|---|---|---|
| **id** | 1.000000 | 0.003538 | 0.003550 | -0.001296 | 0.001092 | 0.002999 | 0.006388 |
| **age** | 0.003538 | 1.000000 | 0.276398 | 0.263796 | 0.238171 | 0.325942 | 0.245257 |
| **hypertension** | 0.003550 | 0.276398 | 1.000000 | 0.108306 | 0.174474 | 0.160189 | 0.127904 |
| **heart_disease** | -0.001296 | 0.263796 | 0.108306 | 1.000000 | 0.161857 | 0.038899 | 0.134914 |
| **avg_glucose_level** | 0.001092 | 0.238171 | 0.174474 | 0.161857 | 1.000000 | 0.168751 | 0.131945 |
| **bmi** | 0.002999 | 0.325942 | 0.160189 | 0.038899 | 0.168751 | 1.000000 | 0.038947 |
| **stroke** | 0.006388 | 0.245257 | 0.127904 | 0.134914 | 0.131945 | 0.038947 | 1.000000 |

# HEAT MAP

In [19]:
```
sns.heatmap(a,annot=True)
```

Out[19]: <AxesSubplot:>

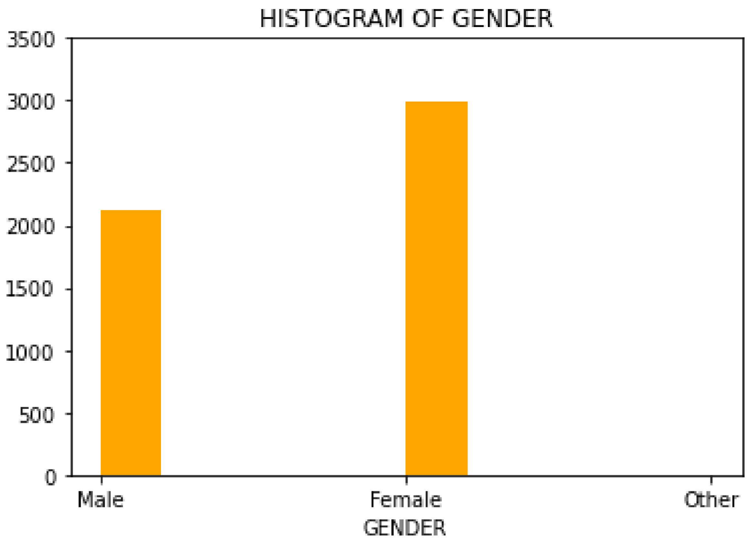


# HISTOGRAM

In [20]:
```
plt.hist(p.gender,color="orange")
plt.ylim(0,3500)
plt.xlabel("GENDER")
plt.title("HISTOGRAM OF GENDER")

#From the plot we analyze that Female count is more than the Male count.
```

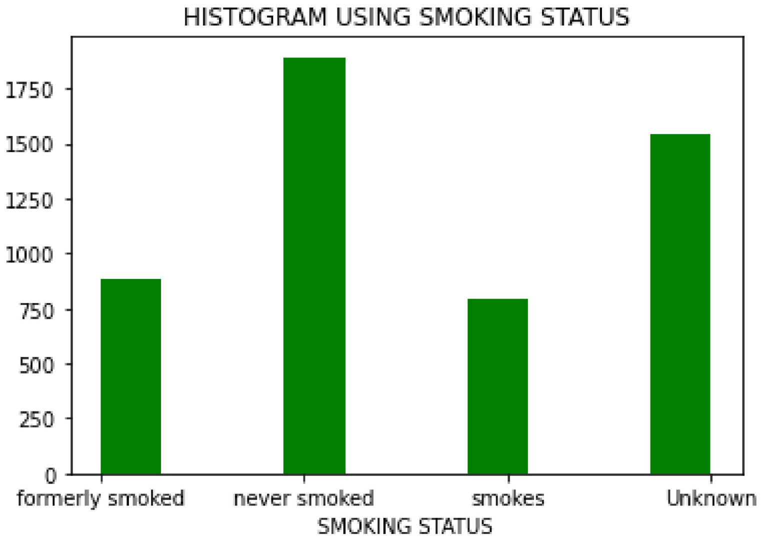
Out[20]:
```
Text(0.5, 1.0, 'HISTOGRAM OF GENDER')
```



In [22]:
```
plt.hist(p.smoking_status,color="green" )
plt.xlabel("SMOKING STATUS")
plt.title("HISTOGRAM USING SMOKING STATUS")

#From the analysis we determine that the 'never smoked' count is more comparing to 'smokes' and 'formerly smoked'
```

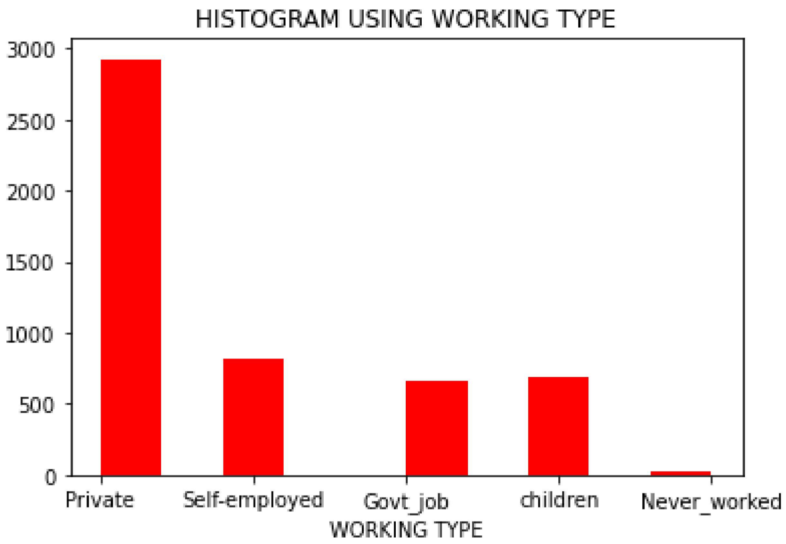
Out[22]:
```
Text(0.5, 1.0, 'HISTOGRAM USING SMOKING STATUS')
```



In [23]:
```
plt.hist(p.work_type,color="red" )
plt.xlabel("WORKING TYPE")
plt.title("HISTOGRAM USING WORKING TYPE")

#From the plot we analyze that private company employees are more than the government and self-Employees.
```

Out[23]:
```
Text(0.5, 1.0, 'HISTOGRAM USING WORKING TYPE')
```

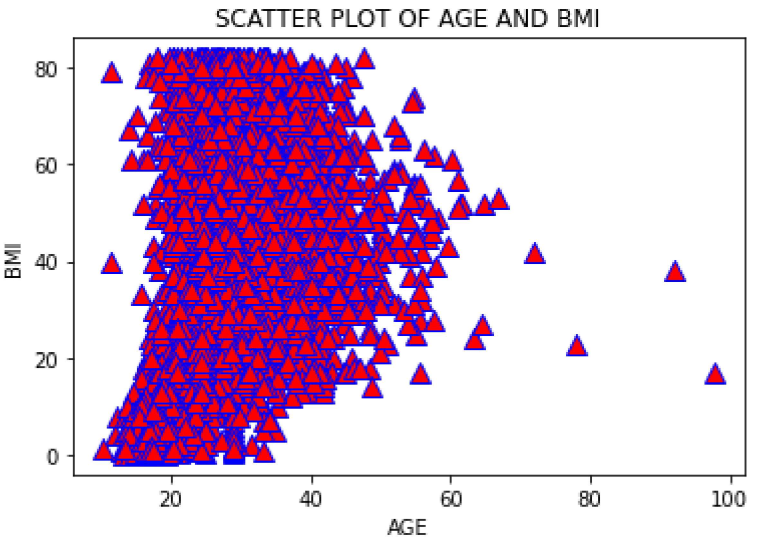



# SCATTER PLOT

In [24]:
```
plt.scatter(p.bmi,p.age,marker="^",color="red",s=100,edgecolor="blue")
plt.xlabel("AGE")
plt.ylabel("BMI")
plt.title("SCATTER PLOT OF AGE AND BMI")

#From the plot between BMI and age the BMI lies more between the age of 20 to 50.
```

Out[24]:
```
Text(0.5, 1.0, 'SCATTER PLOT OF AGE AND BMI')
```

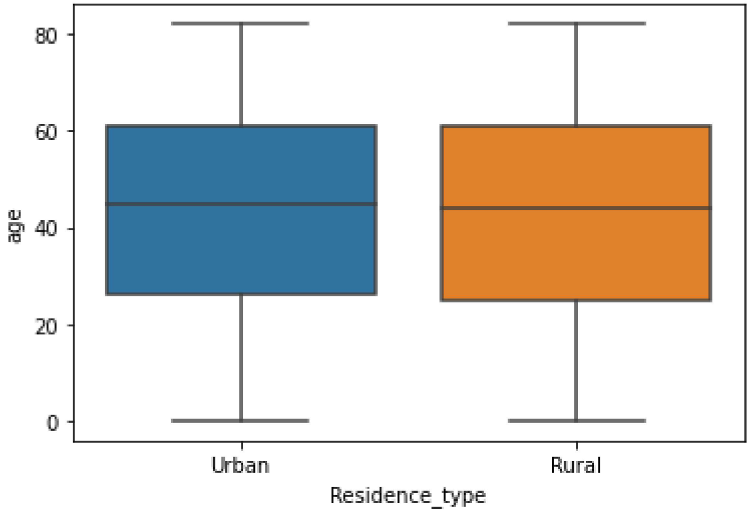



## BOX PLOT

In [49]:
```
sns.boxplot(x="Residence_type",y="age",data=p)

# From the plot we can analyze that for both rural and urban types the median of age lies between 40 to 60.
```

Out[49]: <AxesSubplot:xlabel='Residence_type', ylabel='age'>

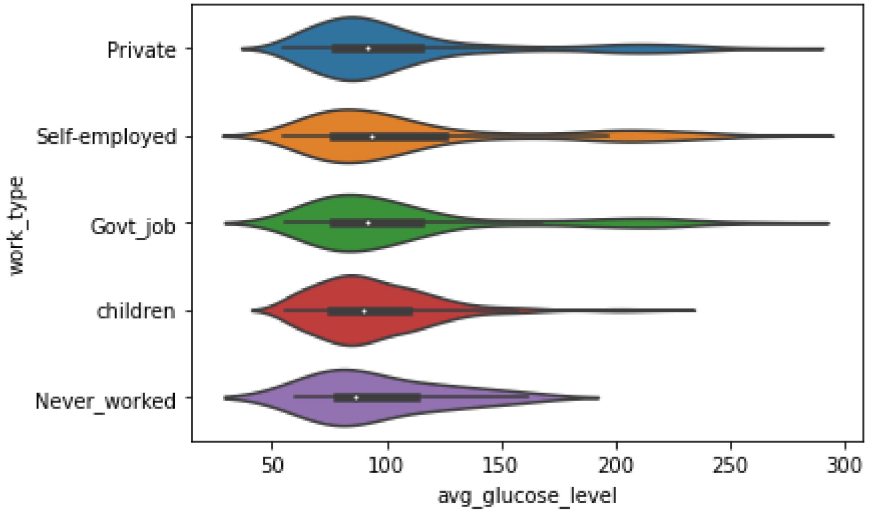



## VIOLIN PLOT

In [47]:
```
sns.violinplot(x="avg_glucose_level",y="work_type",data=p)

#From the plot we can analyze that for all the work types the avaerage Glucose level lies between 50 t0 100.
```

Out[47]: <AxesSubplot:xlabel='avg_glucose_level', ylabel='work_type'>




## Splitting into Training and Testing Data

The dataset is split into dependent and independent attributes using the train test split method of sklearn package in python. The dataset is split into 80% for training and 20% for testing. The independent features include all the input parameters like age, gender, work type, smoking status, etc while the dependent feature is stroke.

In [50]:
```
p.columns
```

Out[50]:
```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [51]:
```
p.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                5110 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

In [53]:
```
x=p[[ 'age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi' ]]
y=p['stroke']
```

In [54]:
```
from sklearn.model_selection import train_test_split
```

In [55]:
```
x_train,x_test,y_train,y_test=train_test=train_test_split(x,y,test_size=0.2,random_state=101)
```

In [56]:
```
from sklearn.linear_model import LogisticRegression
```

In [57]:
```
log=LogisticRegression()
log
```
Out[57]:
```
LogisticRegression()
```

## Fitting the Model

In [58]:
```
log.fit(x_train,y_train)
```
Out[58]:
```
LogisticRegression()
```

In [59]:
```
predict=log.predict(x_test)
predict
```
Out[59]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [60]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
```

## CONFUSION MATRIX

[TP TN]

[FP FN]

We are Getting the True Positive values are more,so our is Good.

In [63]:
```
confusion_matrix(y_test,predict)
```
Out[63]:
```
array([[968,   0],
       [ 54,   0]], dtype=int64)
```

In [64]:
```
accuracy_score(y_test,predict)
```
Out[64]:
```
0.9471624266144814
```

In [65]:
```
y_test
```

Out[65]:
```
5031    0
4017    0
744     0
1799    0
2314    0
       ..
4795    0
4641    0
1320    0
1098    0
4634    0
Name: stroke, Length: 1022, dtype: int64
```

In [66]:
```
from sklearn.metrics import classification_report
```

```
In [67]: print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       968
           1       0.00      0.00      0.00        54

    accuracy                           0.95      1022
   macro avg       0.47      0.50      0.49      1022
weighted avg       0.90      0.95      0.92      1022


C:\Users\K N Nithyanand\BG\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\K N Nithyanand\BG\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\K N Nithyanand\BG\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

# FINAL RESULT OF THE MODEL

The results obtained after applying Logistic Regression. The metrics used to carry out performance analysis of the algorithm are Accuracy score, Precision (P), Recall (R) and F-measure. Precision metric provides the measure of positive analysis that is correct. Recall defines the measure of actual positives that are correct. F-measure tests accuracy

From the model we can Analyze that using logistic Regression Algorithm and most efficient model is obtained. Logistic Regression was found out to be the most effective one with an Accuracy 95%.

A we have got the ACCURACY as 95 % ,The Model is Excellent.