Objective : build a machine learning model to predict whether someone has high risk of getting stroke or not

```
Background: A stroke occurs when the blood supply to part of your brain is interrupted or reduced,
preventing brain tissue from getting oxygen and nutrients. Brain cells begin to die in minutes. A stroke
is a medical emergency, and prompt treatment is crucial. Early action can reduce brain damage and other
complications. The good news is that many fewer Americans die of stroke now than in the past. Effective
treatments can also help prevent disability from stroke.

Motivation : Our objective is to understand what are the reasons that cause stroke to peoeple and see if
we can succefully detect stroke on some features using ML technics
```

In [1]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler,OneHotEncoder,OrdinalEncoder,LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier,VotingClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import f1_score,confusion_matrix,plot_confusion_matrix,classification_report,accuracy_scor
from imblearn.over_sampling import RandomOverSampler
```

In [3]:
```
stroke_data=pd.read_csv('data/healthcare-dataset-stroke-data.csv')
```

In [4]:
```
stroke_data.head()
```

Out[4]:

	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 [5]:
```
stroke_data.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 [6]:
```
stroke_data_replaced=stroke_data.copy()
```

In [7]:
```
stroke_data_replaced.fillna(value=stroke_data_replaced.mean(axis=0),axis=0,inplace=True)
```

In [8]:
```
for col in ['hypertension','heart_disease','stroke']:
    stroke_data_replaced[col]=stroke_data_replaced[col].replace([0,1],['No','Yes'])
```

In [9]:
```
stroke_data_replaced.describe()
```

Out[9]:

	id	age	avg_glucose_level	bmi
count	5110.000000	5110.000000	5110.000000	5110.000000
mean	36517.829354	43.226614	106.147677	28.893237
std	21161.721625	22.612647	45.283560	7.698018
min	67.000000	0.080000	55.120000	10.300000
25%	17741.250000	25.000000	77.245000	23.800000
50%	36932.000000	45.000000	91.885000	28.400000
75%	54682.000000	61.000000	114.090000	32.800000
max	72940.000000	82.000000	271.740000	97.600000

In [10]:
```
stroke_data_replaced.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   object
 4   heart_disease      5110 non-null   object
 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   object
dtypes: float64(3), int64(1), object(8)
memory usage: 479.2+ KB
```

In [11]:
```
stroke_data_replaced.head()
```

Out[11]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	9046	Male	67.0	No	Yes	Yes	Private	Urban	228.69	36.600000	formerly smoked
1	51676	Female	61.0	No	No	Yes	Self-employed	Rural	202.21	28.893237	never smoked
2	31112	Male	80.0	No	Yes	Yes	Private	Rural	105.92	32.500000	never smoked
3	60182	Female	49.0	No	No	Yes	Private	Urban	171.23	34.400000	smokes
4	1665	Female	79.0	Yes	No	Yes	Self-employed	Rural	174.12	24.000000	never smoked

In [12]:
```
categorical_col=[]
numerical_col=[]
for col in stroke_data_replaced.dtypes.index:
    if stroke_data_replaced[col].dtype == 'object':
        categorical_col.append(col)
    else:
        numerical_col.append(col)
```

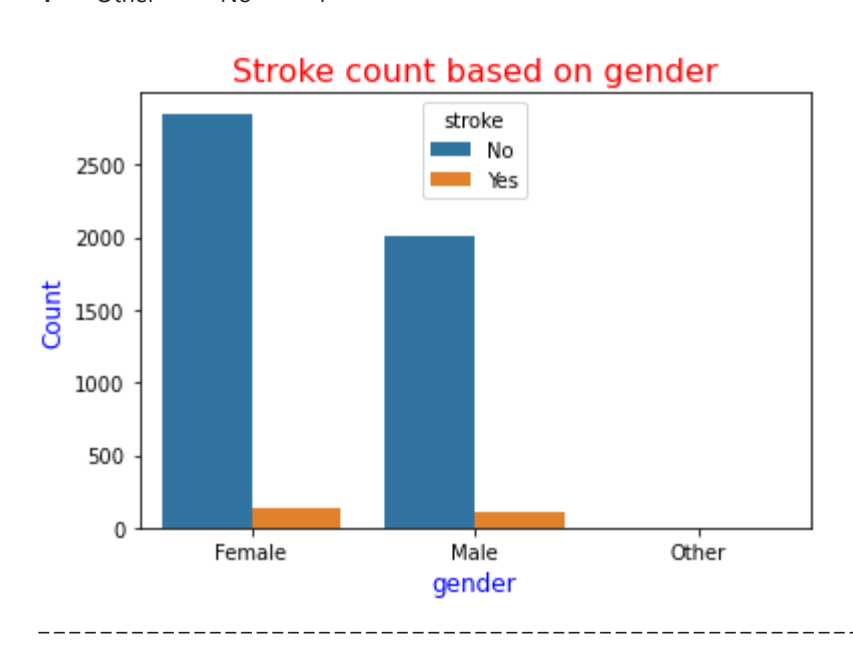
In [13]:
```
print("Numerical Columns in DataFrame are :",numerical_col, '\n')
print("Categorical Columns in DataFrame are :",categorical_col)
```

```
Numerical Columns in DataFrame are : ['id', 'age', 'avg_glucose_level', 'bmi']

Categorical Columns in DataFrame are : ['gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type',
'Residence_type', 'smoking_status', 'stroke']
```

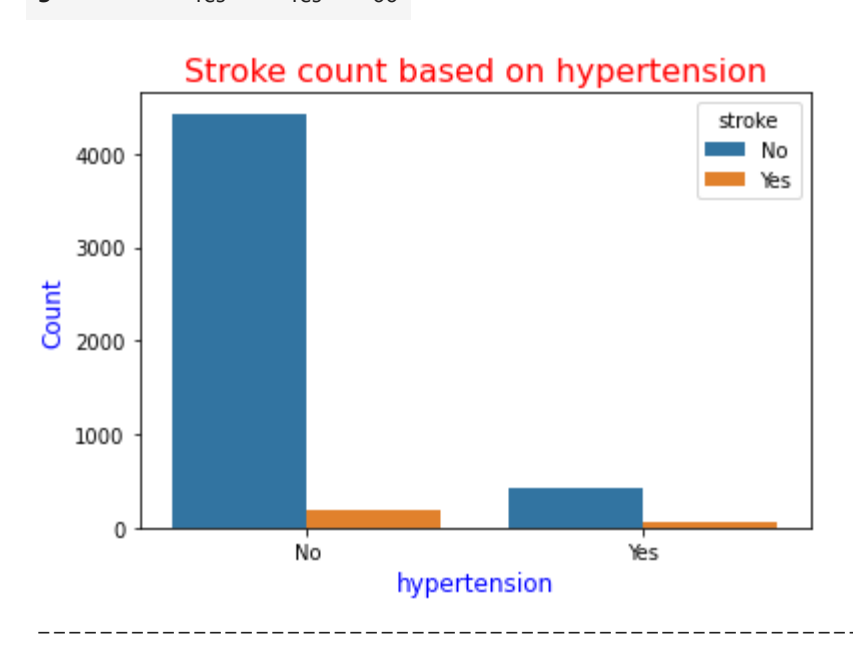
In [14]:
```
def bar_plot(data,x,y='id',hue='stroke',group='stroke' ,title=None):
    """
        function use to plot barplot by grouping the data

    """
    try:
        group=data.groupby([x,group],as_index=False)['id'].count()
        sns.barplot(data=group,x=x,y=y,hue=hue,)
        plt.title("Stroke count based on "+ x , fontdict={'size' :16,'color':'red'})
        plt.ylabel('Count',fontdict={'size':12,'color':'blue'})
        plt.xlabel(x,fontdict={'size':12,'color':'blue'})
        print('-'*100)
        return display(group) ,plt.show()
    except:
        pass
```

In [15]:
```
for col in categorical_col:
    bar_plot(stroke_data_replaced,col)
```

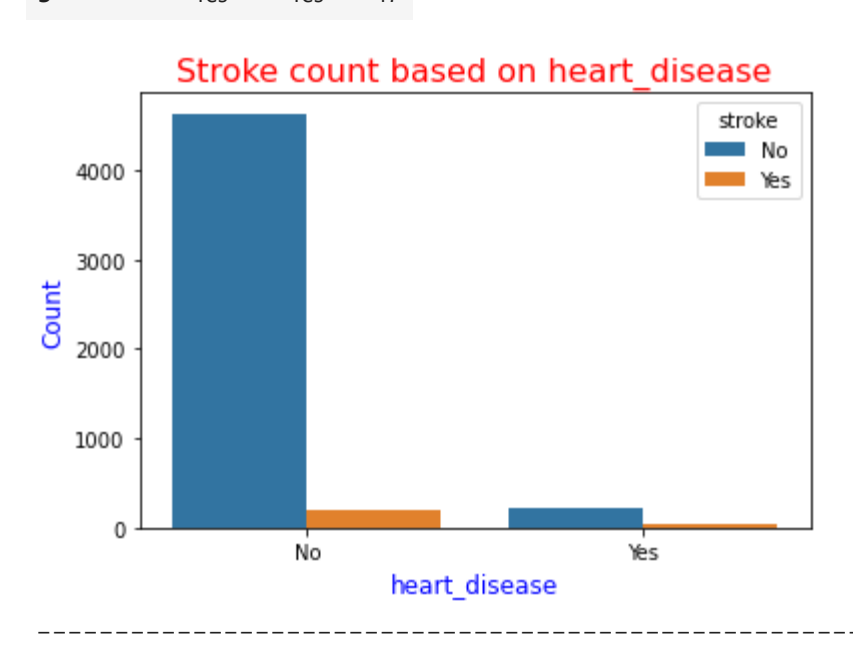
	gender	stroke	id
0	Female	No	2853
1	Female	Yes	141
2	Male	No	2007
3	Male	Yes	108
4	Other	No	1



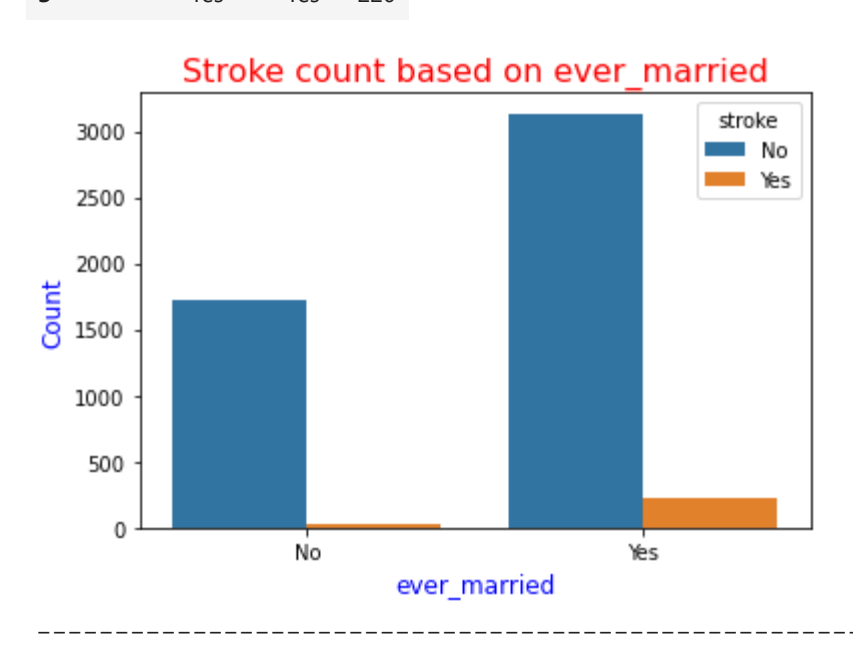
	hypertension	stroke	id
0	No	No	4429
1	No	Yes	183
2	Yes	No	432
3	Yes	Yes	66



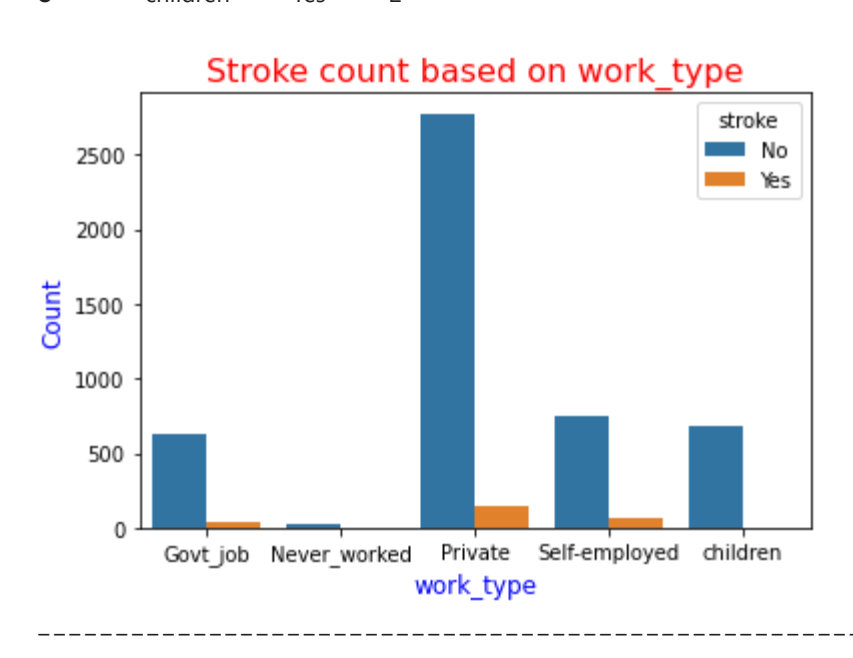
	heart_disease	stroke	id
0	No	No	4632
1	No	Yes	202
2	Yes	No	229
3	Yes	Yes	47



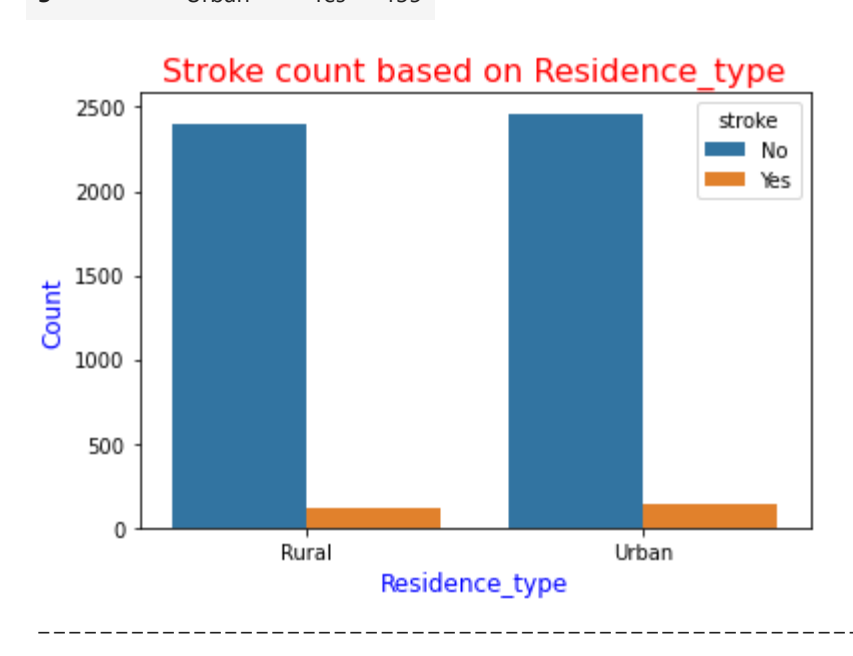
	ever_married	stroke	id
0	No	No	1728
1	No	Yes	29
2	Yes	No	3133
3	Yes	Yes	220



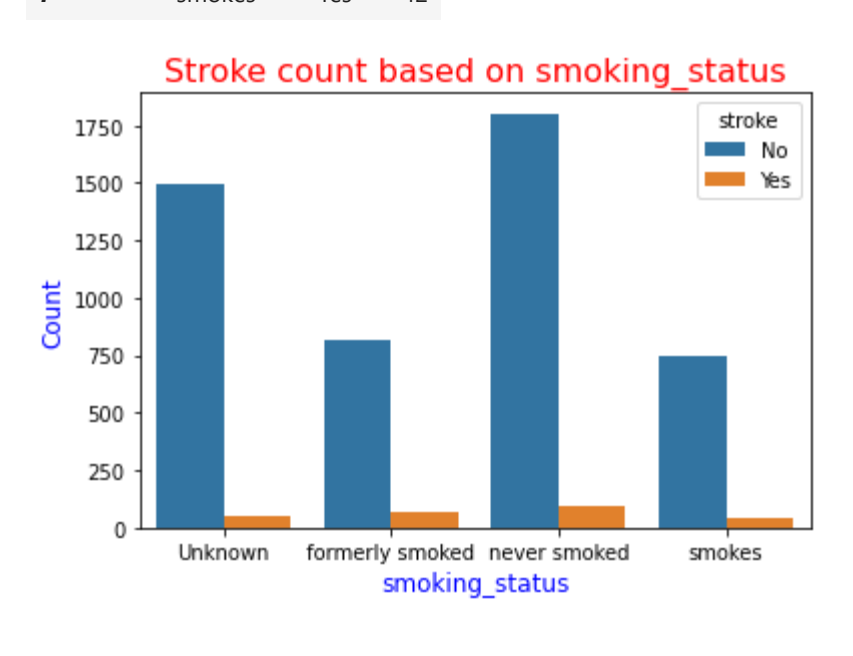
	work_type	stroke	id
0	Govt_job	No	624
1	Govt_job	Yes	33
2	Never_worked	No	22
3	Private	No	2776
4	Private	Yes	149
5	Self-employed	No	754
6	Self-employed	Yes	65
7	children	No	685
8	children	Yes	2



	Residence_type	stroke	id
0	Rural	No	2400
1	Rural	Yes	114
2	Urban	No	2461
3	Urban	Yes	135



	smoking_status	stroke	id
0	Unknown	No	1497
1	Unknown	Yes	47
2	formerly smoked	No	815
3	formerly smoked	Yes	70
4	never smoked	No	1802
5	never smoked	Yes	90
6	smokes	No	747
7	smokes	Yes	42



Insights and key findings

Things that can be assumed on the basis of above plots:

- Stroke Count based on gender
  - it is around 5% for males and 4% for females those have been struggling with stroke
  - this can be the important feature as 1% is a significant drop
- Stroke Count based on hypertension
  - It is found to be person tested with hypertension is more likely to be stroked (~ 13%)
  - and this is also accepted by having some prior domain knowledge
- Stroke Count based on heart_disease
  - It is found to be person with heart disease are at high risk of stroke (~ 17%)
  - This is also accepted by having some prior domain knowlegde
- Stroke Count Based on maritial Status
  - 6% of married people have faced stroke
  - also is it to be notice that we have more baised data around the married people nearly 2000 more entries married people
  - This need to be investigated further
- Stroke count Based on Work Type
  - Based on the data it is seen that there are more number of records for private sector jobs
  - It can be possible that people having govt job are more relived than people with private jobs but on other have people having no job don't have stroke who should probaly be more worried about getting a job
  - And children are being tested positive for stroke
  - Conclusively Stroke doesn't depends on Work Type
- Stroke Count Based on Smoking Status
  - Similary for people who formerly smoked or smokes have more chances to get Stroke than people who doesn't
  - It is un predictable for the people with Unknown Status

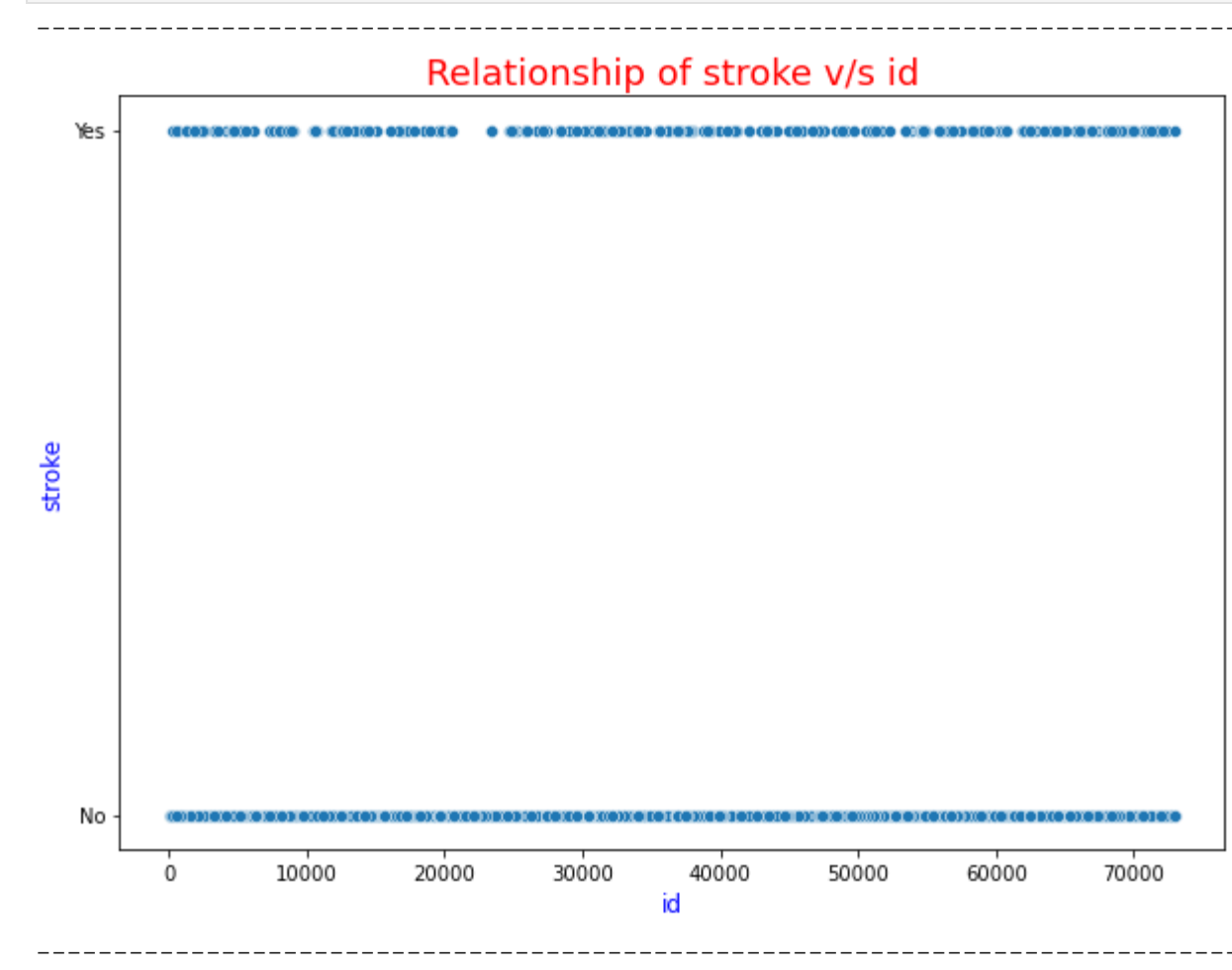
By above Study it is clear that we are going to drop the featue Work Type as it doesn't contribute to prediction

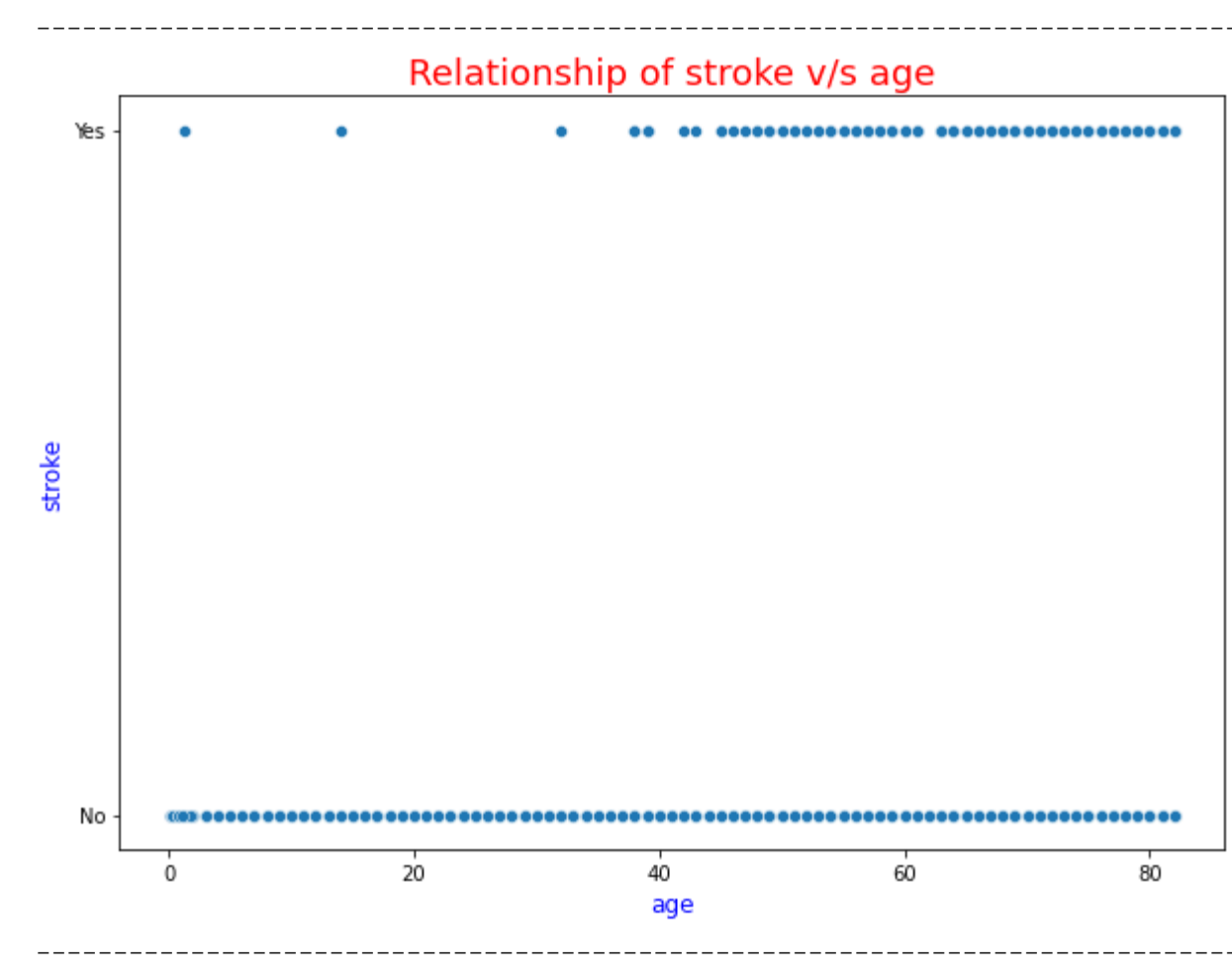
And The Featue Maritial Status need to be further investigated

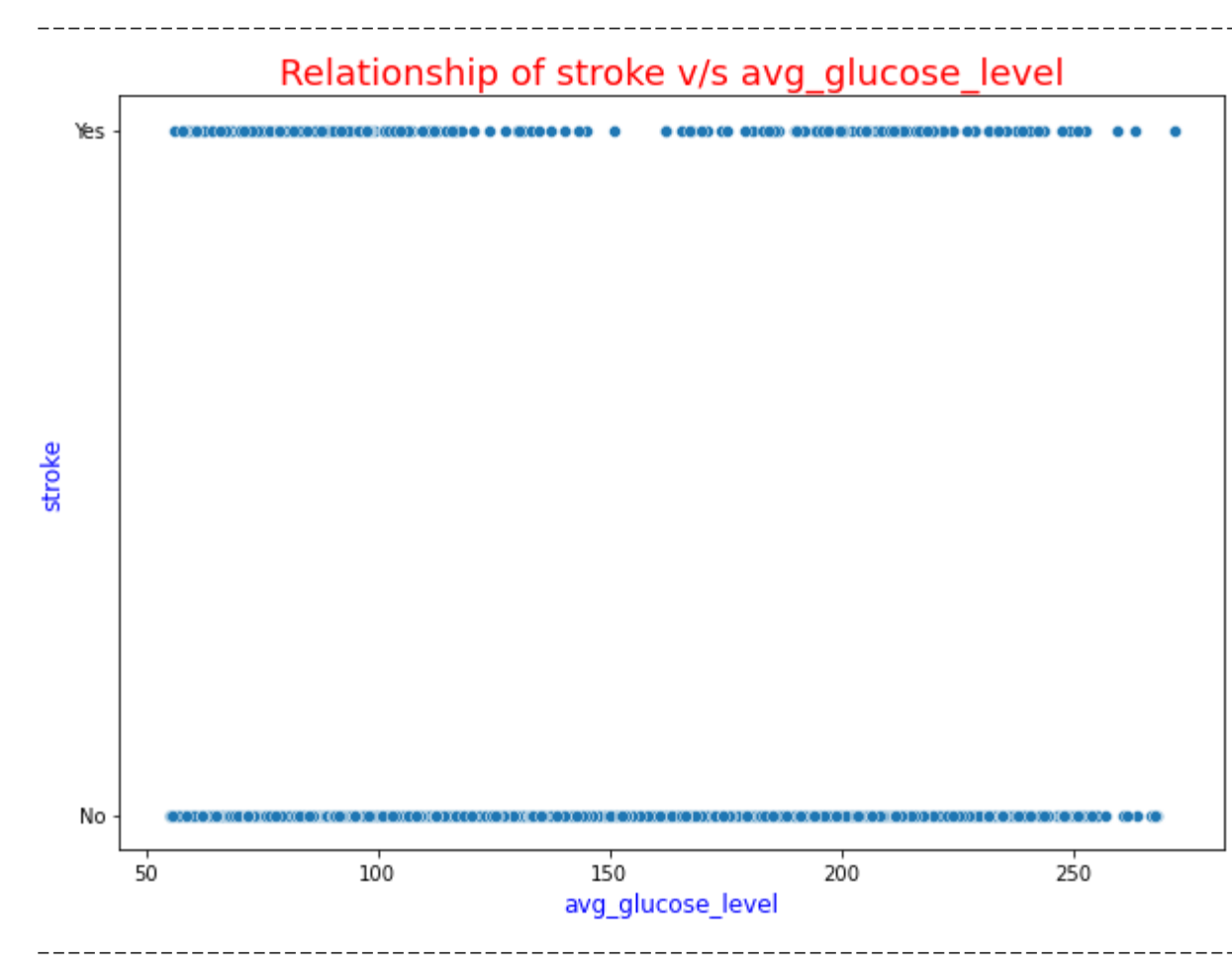
In [16]:
```
Feature_to_drop=['work_type']
```

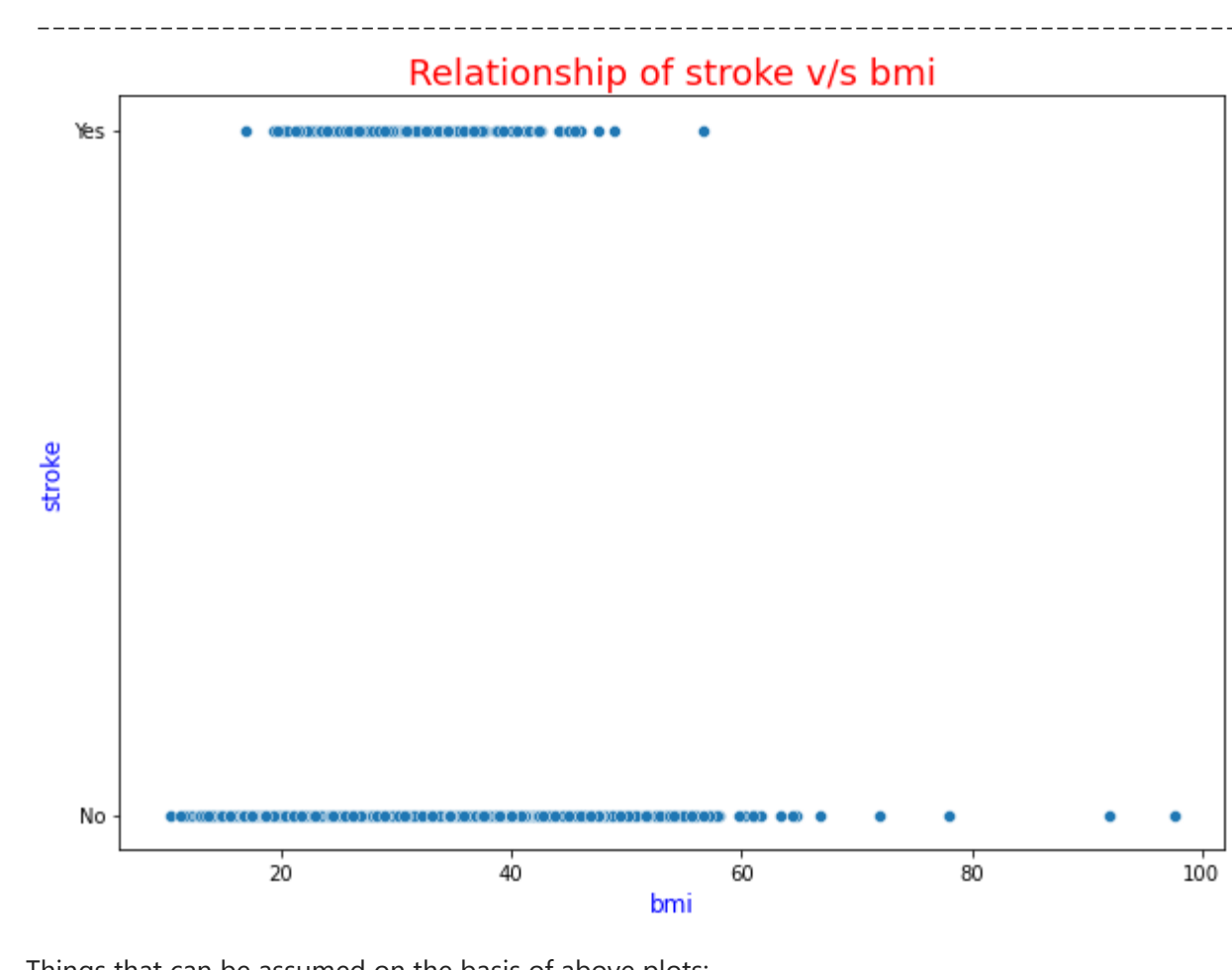
In [17]:
```
def scatter_plot(data,x,y='stroke',title=None):
    plt.figure(figsize=(10,7))
    plt.title("Relationship of "+ y + " v/s "+ x , fontdict={'size':18,'color':'red'})
    sns.scatterplot(data=data,x=x,y='stroke')
    plt.xlabel(x,fontdict={'size':12,"color":'blue'})
    plt.ylabel(y,fontdict={'size':12,"color":'blue'})
    print('-'*100)
    return plt.show()
```

In [18]:
```
for col in numerical_col:
    scatter_plot(stroke_data_replaced,col)
```









Things that can be assumed on the basis of above plots:

- Relationship of Stroke and id
  - this feature id is totally irrelavent for prediction purpose as it add no information to the data
  - every person will have different id and no pattern will be formed
- Relationship of Stroke and age
  - According to the trend elder people will have high risk of getting stroked
- Relationship of Stroke and average glucose level
  - People having glucose level around 150 mg/dl are less prone to get stroke
  - while people having more or less than this are more prone for stroke
- Relationship of Stroke and average BMI
  - we can't really tell if this is a good predictor or not
  - further investgation is needed

In [19]:
```
stroke_data_replaced
```

Out[19]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
0	9046	Male	67.0	No	Yes	Yes	Private	Urban	228.69	36.600000	formerly smoked
1	51676	Female	61.0	No	No	Yes	Self-employed	Rural	202.21	28.893237	never smoked
2	31112	Male	80.0	No	Yes	Yes	Private	Rural	105.92	32.500000	never smoked
3	60182	Female	49.0	No	No	Yes	Private	Urban	171.23	34.400000	smokes
4	1665	Female	79.0	Yes	No	Yes	Self-employed	Rural	174.12	24.000000	never smoked
...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	Yes	No	Yes	Private	Urban	83.75	28.893237	never smoked
5106	44873	Female	81.0	No	No	Yes	Self-employed	Urban	125.20	40.000000	never smoked
5107	19723	Female	35.0	No	No	Yes	Self-employed	Rural	82.99	30.600000	never smoked
5108	37544	Male	51.0	No	No	Yes	Private	Rural	166.29	25.600000	formerly smoked
5109	44679	Female	44.0	No	No	Yes	Govt_job	Urban	85.28	26.200000	Unknown

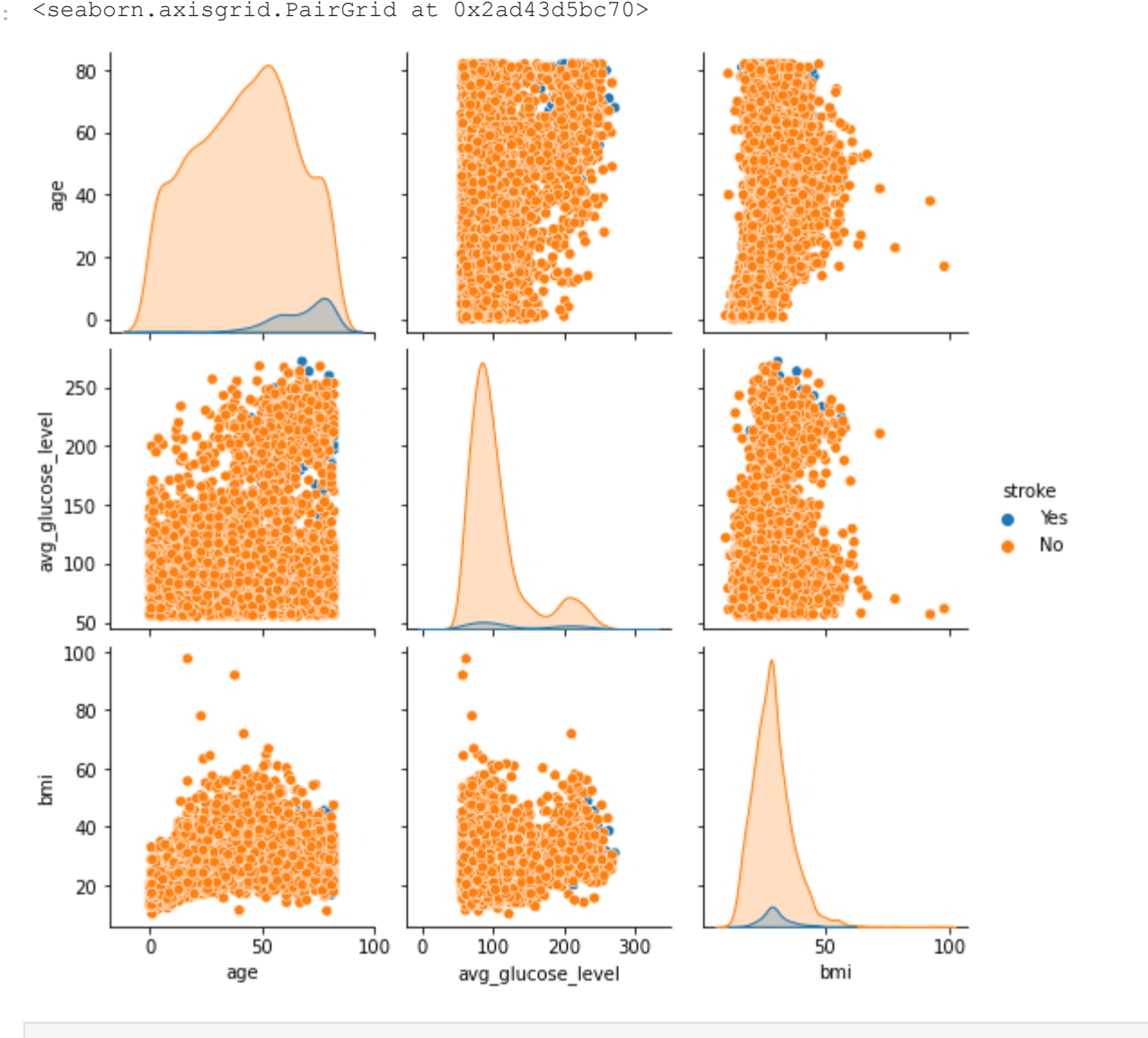
5110 rows × 12 columns

In [20]:
```
Feature_to_drop.append('id')
```

In [21]:
```
stroke_data_replaced.drop(Feature_to_drop,axis=1,inplace=True)
```

In [22]:
```
sns.pairplot(stroke_data_replaced,hue='stroke')
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x2ad43d5bc70>



In [23]:
```
sample=stroke_data_replaced.sample(n=100,random_state=72018,)
sample
```

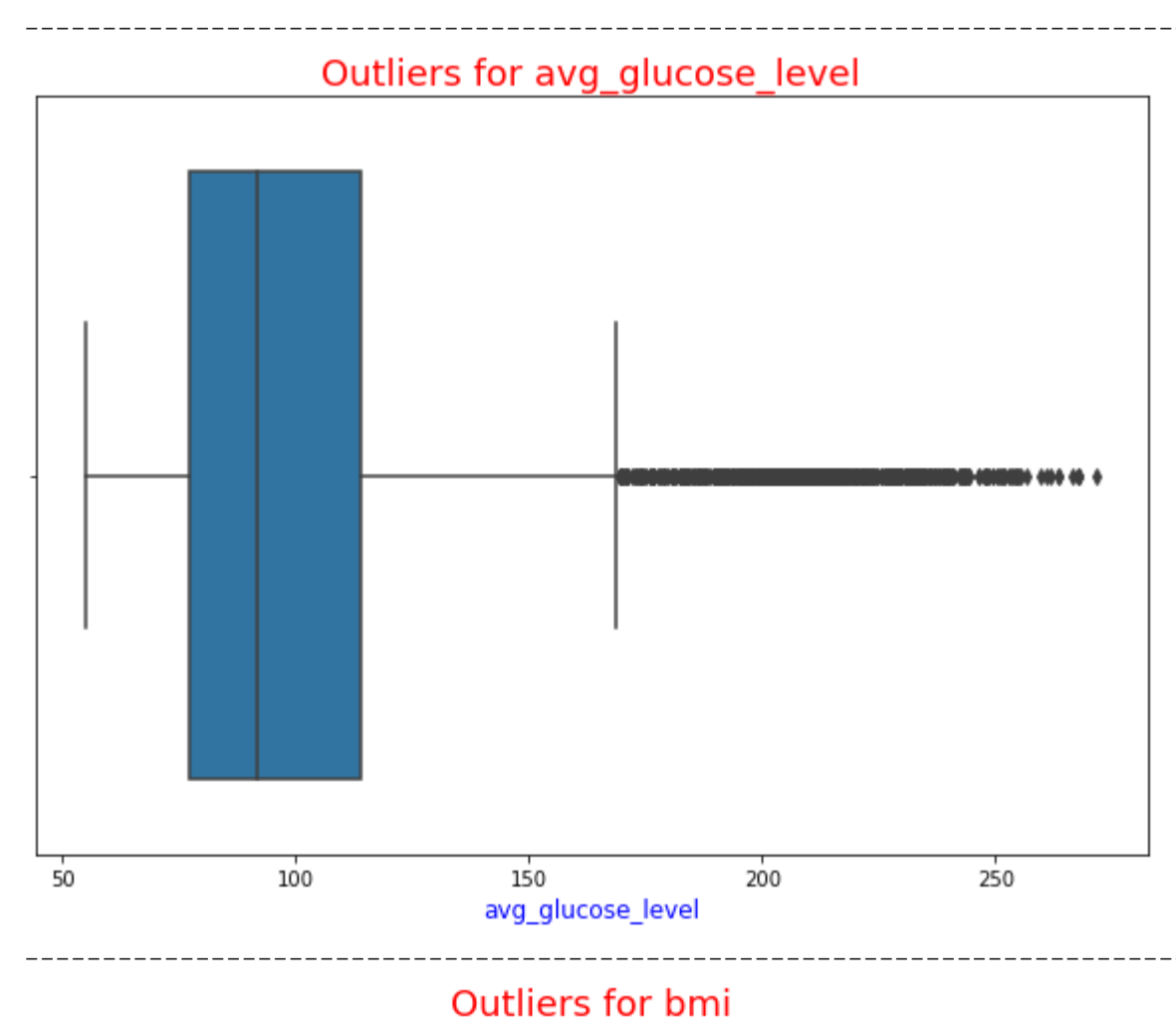
Out[23]:

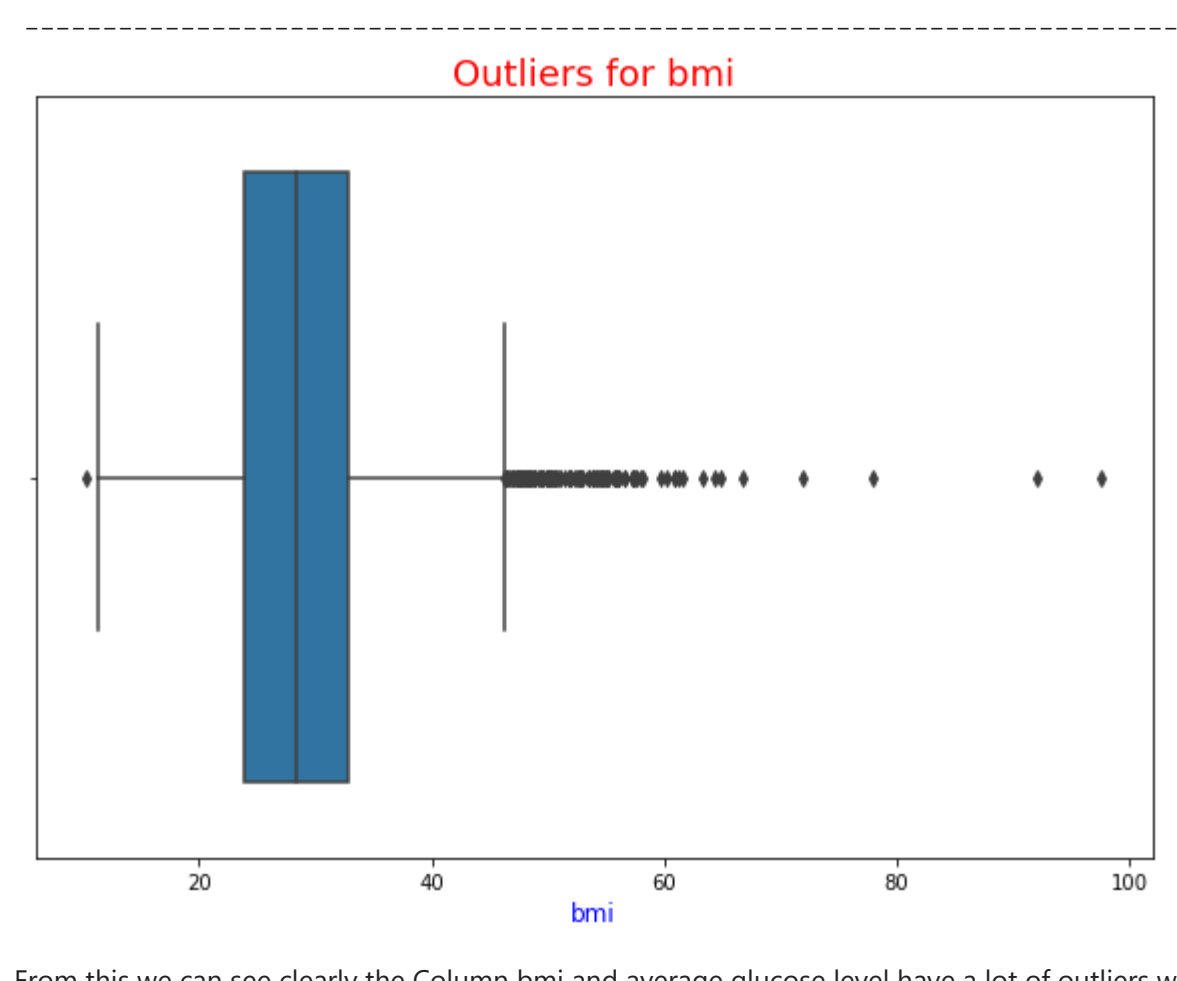
	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
485	Female	31.0	No	No	Yes	Urban	98.99	31.200000	never smoked	No
871	Female	62.0	Yes	No	Yes	Rural	77.04	34.700000	never smoked	No
1125	Female	59.0	Yes	No	No	Rural	57.26	23.500000	never smoked	No
3347	Male	80.0	No	No	Yes	Urban	148.72	28.700000	never smoked	No
2427	Female	37.0	No	No	No	Rural	77.37	21.400000	never smoked	No
...	...	...	...	...	...	...	...	...	...	...
3197	Male	60.0	No	No	Yes	Rural	212.02	28.893237	Unknown	No
4747	Female	12.0	No	No	No	Rural	80.89	20.100000	Unknown	No
3359	Female	23.0	No	No	No	Urban	59.07	21.600000	never smoked	No
2483	Female	39.0	No	No	Yes	Urban	107.47	21.300000	Unknown	No
2583	Female	75.0	No	Yes	Yes	Urban	206.15	25.400000	never smoked	No

100 rows × 10 columns

In [24]:
```
def outlier_detection(data,x):
    plt.figure(figsize=(10,7))
    plt.title("Outliers for " + x , fontdict={'size':18,'color':'red'})
    sns.boxplot(data=data,x=x)
    plt.xlabel(x,fontdict={'size':12,"color":'blue'})
    print('-'*100)
    return plt.show()
```

In [25]:
```
for i in numerical_col:
    if i!='id':
        outlier_detection(stroke_data_replaced,i)
```





From this we can see clearly the Column bmi and average glucose level have a lot of outliers we can handle this with suitable techniques like replacing it with mean or median or dropping in in this case we are leaving these outliers as it is.

In hope that an extreme behaviour target variable are highly dependent on the pattern of these features

In [26]:
```
#this will mask the categorical columns
mask = stroke_data_replaced.dtypes == np.object_
```

In [27]:
```
#here we separate the numerical and categorical columns
categorical_col = stroke_data_replaced.columns[mask.values]
numerical_col = stroke_data_replaced.columns[~mask.values]
```

In [28]:
```
#now separate ordinal and onehot columns
#here by the vizulization we know our ordinal column have >2 variables

onehot_categorical_columns=[]
ordinal_categorical_columns=[]
for i in categorical_col:
    if stroke_data_replaced[i].nunique()<3:
        onehot_categorical_columns.append(i)
    else:
        ordinal_categorical_columns.append(i)
```

In [29]:
```
#create our transformers that will transform the columns
#ordinal transformer
OD=OrdinalEncoder(handle_unknown='use_encoded_value',unknown_value=5)

#min max scaler
MM=MinMaxScaler()

#label encoder for target variable
LE=LabelEncoder()

#one hot encoder column
OHE=OneHotEncoder(handle_unknown='ignore')
```

In [30]:
```
#target variable
y=stroke_data_replaced['stroke']

#remove from onehot columns
onehot_categorical_columns.remove('stroke')
```

In [31]:
```
#create column transformer that will transform the columns respective to the criteria
transformer=ColumnTransformer(transformers=[('numerical',MM,numerical_col),
                                            ('ordinal',OD,ordinal_categorical_columns),
                                            ('onehot',OHE,onehot_categorical_columns)])
```

In [32]:
```
#feature columns
X=stroke_data_replaced.drop('stroke',axis=1)
```

In [33]:
```
#used for oversampling the data as it is highly imbalanced
ROS=RandomOverSampler(random_state=42)
```

In [34]:

```python
def transform_and_resample_data(X,y):
    """
    It accepts feature column and target column.
    split the data int train and test set in stratify manner.
    transform the target with label encoder
    over sample the target set.
    return train set and test set
    """

    train_X,test_X,train_y,test_y = train_test_split(X,y,test_size=0.2,random_state=42,stratify=y)
    train_y = LE.fit_transform(train_y)
    test_y = LE.transform(test_y)
    train_X_resampled , train_y_resampled = ROS.fit_resample(train_X,train_y)


    return train_X_resampled,train_y_resampled,test_X,test_y
```

In [35]:

```python
def prune_tree(train_X,train_y,transformer):
    """
    accepts the model and train data.
    fit to Decision tree return the required parameter.

    It is used to prune the tree to generate range of the parameters.
    """
    model = DecisionTreeClassifier(criterion='gini',random_state=42)

    pipeline=Pipeline(steps=[('transform',transformer),
                             ('model',model)])

    pipeline.fit(train_X,train_y)
    return model.tree_.max_depth,model.tree_.n_features
```

In [36]:

```python
def model_selection(train_X,train_y,transformer,current=None):
    """
    It is used to select different models for traning purpose based in the value of current.

    """
    if current == 0:

        label='Logistic Regression'
        model = LogisticRegression(solver='liblinear')
        params = {'model__C':[0.001,0.01,0.1,1,10],
               'model__penalty':['l1','l2']}

    elif current == 1:

        label = 'KNeighborsClassifier'
        model = KNeighborsClassifier(weights='distance')
        params = {'model__n_neighbors':list(range(2,20))}

    elif current == 2:

        label = 'SVC'
        model = SVC(kernel='rbf')
        params = {'model__C':[0.001,0.01,0.1,1,10]}

    elif current == 3:

        label = 'DecisionTreeClassifier'
        model = DecisionTreeClassifier(criterion='entropy',random_state=42)
        max_depth, max_features = prune_tree(train_X,train_y,transformer)
        params = {'model__max_depth':list(range(1,max_depth+1,2)),
                 'model__max_features':list(range(1,max_features))}

    else:
        raise ValueError('No value passed for current')

    return model, params, label
```

In [37]:

```python
def scoring(model , params, true, pred):
    """
    It is used to score the model based on its preformance
    Accepts model , parameter, original labels and predicted labels

    """
    re_Y = recall_score(true, pred, pos_label=1)
    re_N = recall_score(true, pred, pos_label=0)
    pr_Y = precision_score(true, pred, pos_label=1)
    pr_N = precision_score(true, pred, pos_label=0)
    f1_Y = f1_score(true, pred, pos_label=1)
    f1_N = f1_score(true, pred, pos_label=0)
    acc = accuracy_score(true, pred)
    data = pd.Series({'model':model,
                     'params': params,
                     'recall_Yes': re_Y,
                     'recall_No': re_N,
                     'precision_Yes':pr_Y,
                     'precision_No':pr_N,
                     'f1_Yes':f1_Y,
                     'f1_No':f1_N,
                     'accuracy':acc})

    return data
```

In [38]:

```python
#final dataframe which will store the performance of the model
performance =  pd.DataFrame(columns=['model','params','recall_Yes','recall_No','precision_Yes','precision_No','

#get train and test set
train_X,train_y,test_X,test_y = transform_and_resample_data(X,y)

#call different model fir training and testing
for i in range(4):

    model, param, label = model_selection(train_X, train_y, transformer, current=i)
    pipeline = Pipeline(steps = [('transform',transformer),
                                ('model',model)])

    #score metric
    score = {'r':'recall','p':'precision'}
    grid = GridSearchCV(pipeline,param,cv=4,scoring=score, refit='r')

    grid.fit(train_X,train_y)

    pred = grid.predict(test_X)

    para = grid.best_estimator_.named_steps['model']

    performance = performance.append(scoring(label, para, test_y, pred),ignore_index=True)

    string = 'This results are for '+label
    print(19*'* '+string+19*' *')
    print('----- Classification report -----')
    print(classification_report(test_y,pred))
    print('----- Confusion matrix -----')
    print(confusion_matrix(test_y,pred))
```

```
* * * * * * * * * * * * * * * * * * * This results are for Logistic Regression * * * * * * * * * * * * * * * *
* * *
----- Classification report -----
              precision    recall  f1-score   support

           0       0.98      0.67      0.80       972
           1       0.11      0.78      0.19        50

    accuracy                           0.68      1022
   macro avg       0.55      0.73      0.50      1022
weighted avg       0.94      0.68      0.77      1022

----- Confusion matrix -----
[[654 318]
 [ 11  39]]
* * * * * * * * * * * * * * * * * * * This results are for KNeighborsClassifier * * * * * * * * * * * * * * * *
* * *
----- Classification report -----
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       972
           1       0.20      0.18      0.19        50

    accuracy                           0.92      1022
   macro avg       0.58      0.57      0.57      1022
weighted avg       0.92      0.92      0.92      1022

----- Confusion matrix -----
[[935  37]
 [ 41   9]]
* * * * * * * * * * * * * * * * * * * This results are for SVC * * * * * * * * * * * * * * * * * * *
----- Classification report -----
              precision    recall  f1-score   support

           0       0.98      0.76      0.86       972
           1       0.12      0.66      0.21        50

    accuracy                           0.76      1022
   macro avg       0.55      0.71      0.53      1022
weighted avg       0.94      0.76      0.82      1022

----- Confusion matrix -----
[[740 232]
 [ 17  33]]
* * * * * * * * * * * * * * * * * * * This results are for DecisionTreeClassifier * * * * * * * * * * * * * * * *
* * * *
----- Classification report -----
              precision    recall  f1-score   support

           0       0.97      0.92      0.94       972
           1       0.20      0.38      0.26        50

    accuracy                           0.90      1022
   macro avg       0.58      0.65      0.60      1022
weighted avg       0.93      0.90      0.91      1022

----- Confusion matrix -----
[[897  75]
 [ 31  19]]
```

In [39]:

```python
performance
```

Out[39]:

	model	params	recall_Yes	recall_No	precision_Yes	precision_No	f1_Yes	f1_No	accuracy
0	Logistic Regression	LogisticRegression(C=0.001, solver='liblinear')	0.78	0.672840	0.109244	0.983459	0.191646	0.799023	0.678082
1	KNeighborsClassifier	KNeighborsClassifier(n_neighbors=2, weights='d...	0.18	0.961934	0.195652	0.957992	0.187500	0.959959	0.923679
2	SVC	SVC(C=10)	0.66	0.761317	0.124528	0.977543	0.209524	0.855986	0.756360
3	DecisionTreeClassifier	DecisionTreeClassifier(criterion='entropy', ma...	0.38	0.922840	0.202128	0.966595	0.263889	0.944211	0.896282

observing recall, precision and accuracy of SVC is higher than everyone else.

while KNeighborsClassifier perform the worst.

Can use voting ensemble to increase the recall, precision and accuray