STROKE PREDICTION

Small Introduction:

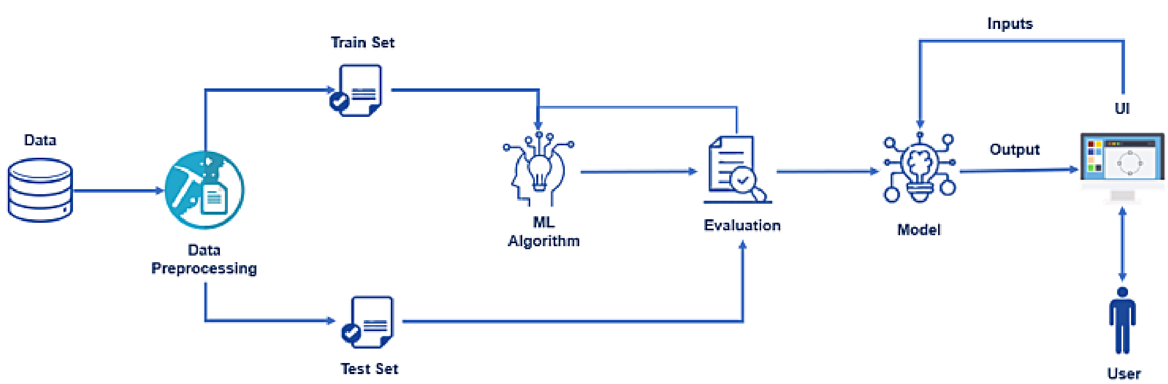
The project aims to predict heart stroke prediction using machine learning with the help of some health vitals of patients.

Introduction: 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 a stroke based on the input parameters like gender, age, various diseases, and smoking status.

Each row in the data provides relevant information about the patient. Here we will be building a flask application that uses a machine learning model to get the prediction of heart stroke.

Architecture:



Project Objectives:
By the end of this project:
● You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.

● You will be able to know how to pre-process/clean the data using different data pre-processing techniques.
● How to perform oversampling when the dataset is imbalanced.
● Applying different algorithms according to the dataset
● You will be able to know how to find the accuracy of the model.
● You will be able to build web applications using the Flask framework.

Project Flow:

● Download the dataset.
● Preprocess or clean the data.
● Identify Outliers and remove Outliers
● Analyze the pre-processed data.
● Perform Oversampling
● Train the machine with preprocessed data using an appropriate machine learning algorithm.
● Save the model and its dependencies.
● Build a Web application using a flask that integrates with the model built.

Pre-requisites:

1. In order to develop this project we need to install the following software/packages:

Anaconda Navigator :
Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

Python packages:
NumPy: NumPy is a Python package that stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object.
Pandas: pandas is a fast, powerful, flexible, and easy to use open-source data analysis

and manipulation tool, built on top of the Python programming language. Matplotlib: It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits Scikit-learn:

It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports python numerical and scientific libraries like NumPy and SciPy.

Flask: Web framework used for building Web applications

Project Structure :

Model Training

model		15 days ago	1.02 MB
model.pkl		3 hours ago	374 kB
Stroke prediction.ipynb	Running	3 hours ago	199 kB
column		2 days ago	1.88 kB
modelor.pkl		3 hours ago	788 B
transform		3 hours ago	534 B

Dataset

healthcare-dataset-stroke-data.csv	15 days ago	317 kB

Flask

app.py	2 days ago	1.22 kB
modelor.pkl	2 days ago	788 B
transform	2 days ago	534 B
templates	2 days ago	

Outputs

output (1).png	15 days ago	49.9 kB
output (3).png	15 days ago	39.4 kB
output (2).png	15 days ago	39.3 kB

● Stroke Prediction.ipynb is the jupyter notebook file where the model is built.
● Dataset.zip is the dataset file used in this project.
● model is the model file that generates when the notebook file is executed.
● Column,mar_transform,res_transform are the transformation and encoding files that were generated when u run the main program.
● Flask folder is the application folder where the web application and server-side program are present.
● healthcare-dataset-stroke-data.csv is the dataset file

Milestone 1:

Data Collection For any Machine learning project data is the primary source

Attribute Information
1) id: a unique identifier
2) gender: "Male", "Female" or "Other"
3) age: age of the patient
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6) ever_married: "No" or "Yes"
7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8) Residence_type: "Rural" or "Urban"
9) avg_glucose_level: average glucose level in the blood
10) BMI: body mass index
11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12) 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

Milestone 2:

Pre-process the data In this milestone, we will be preprocessing the dataset that is collected. Preprocessing includes:
1. Handling the null values.

2. Handling the categorical values if any.
3. Removing Outliers
4. Oversampling to balance the data.
5. Identify the dependent and independent variables.
6. Split the dataset into train and test sets.

Activity 1:

Import required libraries Go to the project folder which you have created copy the project path and open anaconda prompt from the menu and go to the location of your project folder in anaconda prompt and type jupyter notebook. Now Jupyter notebook will be opened and create a python file and start the programming.

```
# Ignore  the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
#import missingno as msno
#import pandas_profiling as pdp

#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#import the necessary modelling algos.

#classifiaction.
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC,SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
#model selection
from sklearn.model_selection import train_test_split,cross_validate
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

#preprocessing
from sklearn.preprocessing import MinMaxScaler,StandardScaler,LabelEncoder

#evaluation metrics
from sklearn.metrics import mean_squared_log_error,mean_squared_error, r2_score,mean_absolute_error # for regression
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score  # for classification
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

Data loading and overview

```
In [15]: df = pd.read_csv(r'C:\Users\KAUSHIK P\Downloads\archive\healthcare-dataset-stroke-data.csv')
         df.head()
```

Out[15]:

	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

Read the datasets The dataset is read as a data frame (df in our program) using the pandas library (pd is the alias name given to the pandas package).

```
In [16]: df.shape
Out[16]: (5110, 12)

In [17]: df.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
```

There are total of 5110 records in the dataset with a total of 12 features.

Activity 3:

Check Null values Here we check the presence of Null values in the dataset and dropping the null values.

```
In [18]: df.isnull().sum()
```

```
Out[18]: id                     0
         gender                 0
         age                    0
         hypertension           0
         heart_disease          0
         ever_married           0
         work_type              0
         Residence_type         0
         avg_glucose_level      0
         bmi                  201
         smoking_status         0
         stroke                 0
         dtype: int64
```

```
In [19]: df.dropna(inplace = True)
```

```
In [20]: df.isnull().sum()
```

```
Out[20]: 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 [21]: for i in ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']:
             print(df[i].unique())

         ['Male' 'Female' 'Other']
         ['Yes' 'No']
         ['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
         ['Urban' 'Rural']
         ['formerly smoked' 'never smoked' 'smokes' 'Unknown']
```

Milestone 3:

Exploratory Data Analysis Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

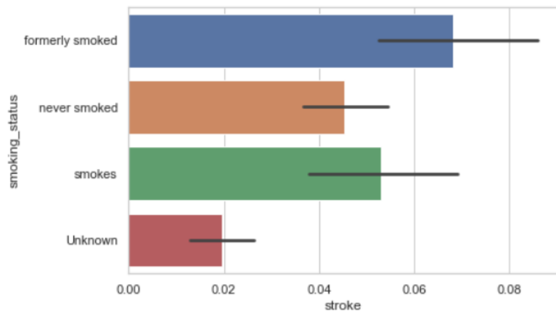
Activity1: Plotting Boxplot

EDA

BOXPLOT

In [22]:
```
sns.barplot(x = 'stroke', y = 'smoking_status', data = df)
```

Out[22]: <AxesSubplot:xlabel='stroke', ylabel='smoking_status'>



'Unknown' and 'never smoker' has a low percentage of strokes in the sample. Let's combine them into one group: 'never smoker'.

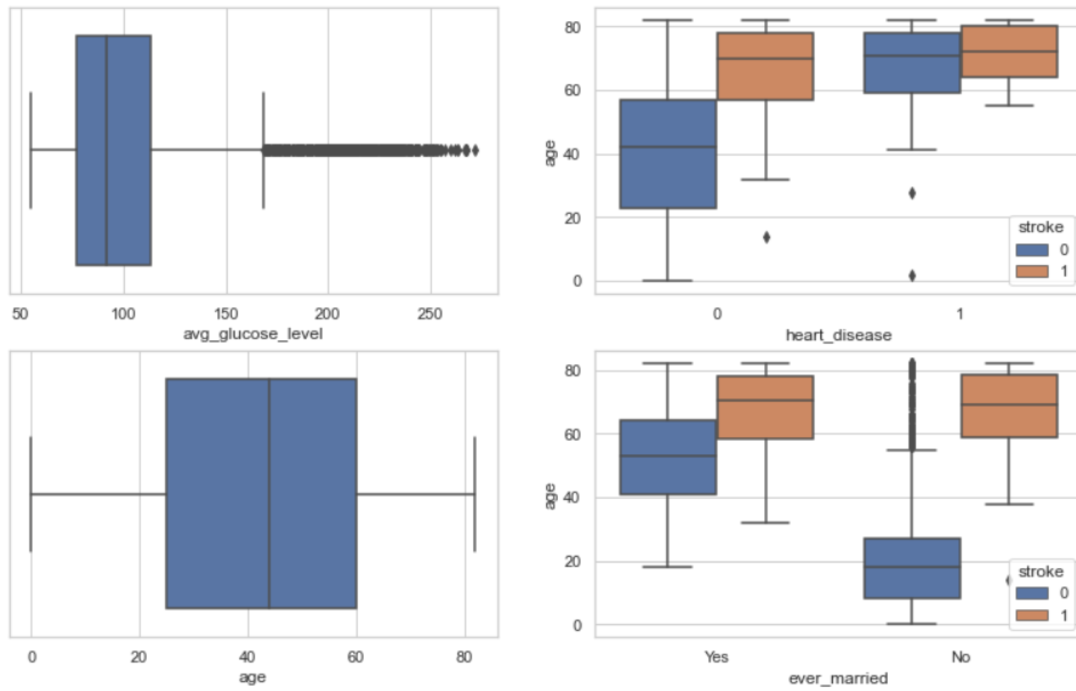
In [23]:
```
replace_values = {'Unknown': 'never smoked'}

df = df.replace({'smoking_status': replace_values})
df.head()
```

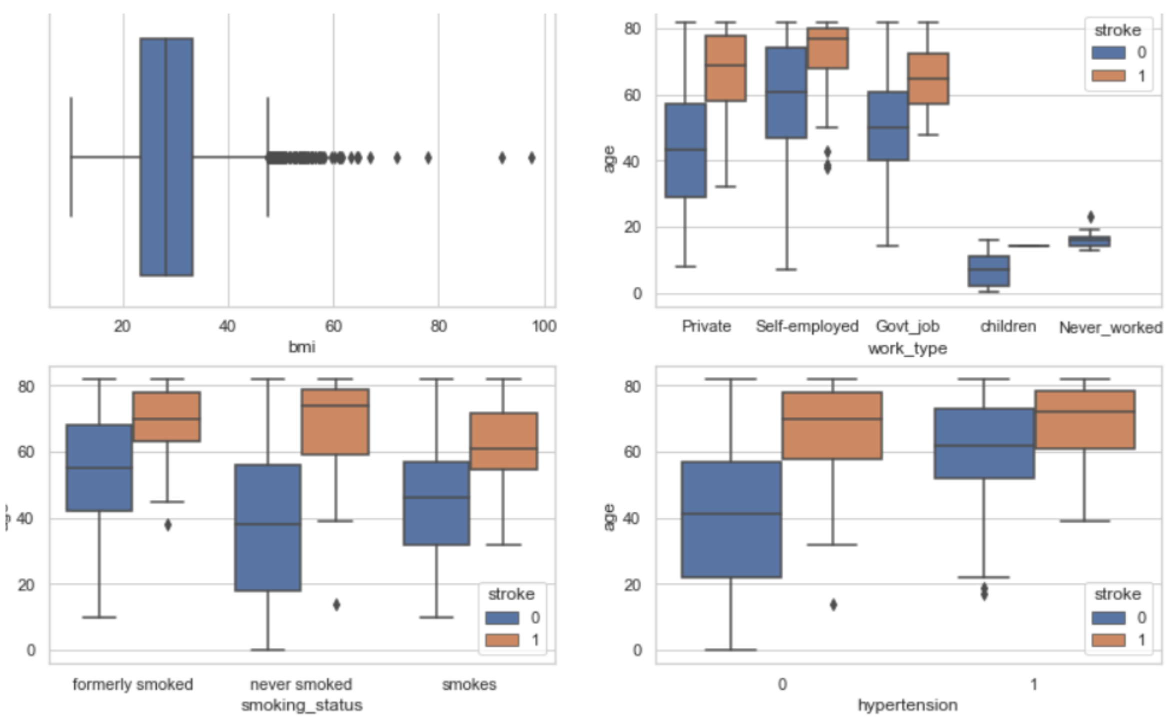
Out[23]:

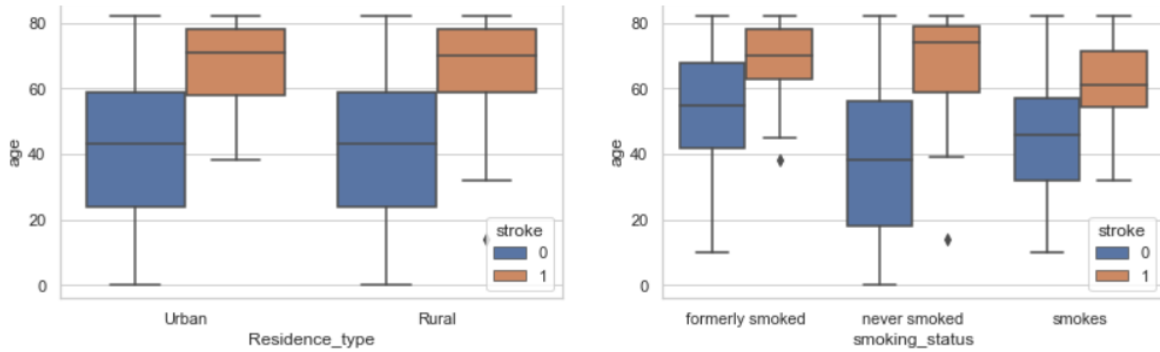
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	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

In [24]:
```
fig, axes = plt.subplots(nrows = 5, ncols = 2, figsize = (12, 20))
sns.boxplot(x = 'avg_glucose_level', data = df, ax=axes[0][0])
sns.boxplot(x = 'age', data = df, ax=axes[1][0])
sns.boxplot(x = 'bmi', data = df, ax=axes[2][0])
sns.boxplot(x = 'smoking_status', y = 'age', hue = 'stroke', data = df, ax=axes[3][0])
sns.boxplot(x = 'hypertension',y = 'age', hue = 'stroke', data = df, ax=axes[3][1])
sns.boxplot(x = 'heart_disease', y= 'age', hue = 'stroke', data = df, ax=axes[0][1])
sns.boxplot(x = 'ever_married',y = 'age', hue = 'stroke', data = df, ax = axes[1][1])
sns.boxplot(x = 'work_type',y = 'age',hue = 'stroke', data = df, ax = axes[2][1])
sns.boxplot(x = 'Residence_type',y = 'age',hue = 'stroke', data = df, ax = axes[4][0])
sns.boxplot(x = 'smoking_status',y = 'age', hue = 'stroke', data = df, ax = axes[4][1])
```

Out[24]: <AxesSubplot:xlabel='smoking_status', ylabel='age'>


50
100
150
200
250
avg_glucose_level
80
60
40
20
0
age
0
1
heart_disease
stroke
0
1
0
20
40
60
80
age
Yes
No
ever_married


20
40
60
80
100
bmi
age
Private
Self-employed
Govt_job
children
Never_worked
work_type
stroke
0
1
formerly smoked
never smoked
smokes
smoking_status
0
1
hypertension



We have outliers in avg_glucose_level and BMI.

Activity3:
Removing Outliers

Remove outliers

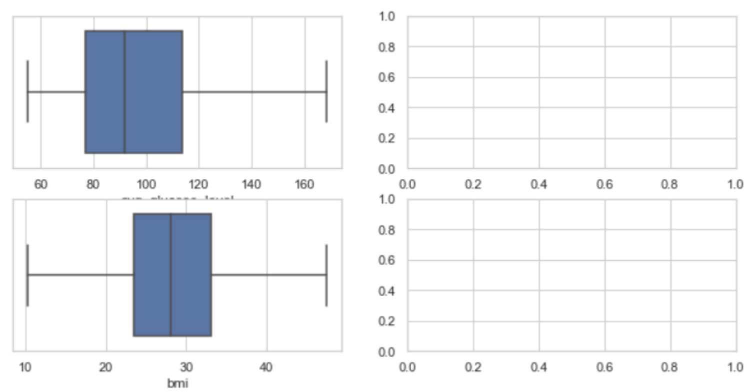
```
In [25]: ### Thanks for the function https://www.kaggle.com/ankitak46

def remove_outliers(data):
    arr=[]
    #print(max(list(data)))
    q1=np.percentile(data,25)
    q3=np.percentile(data,75)
    iqr=q3-q1
    mi=q1-(1.5*iqr)
    ma=q3+(1.5*iqr)
    #print(mi,ma)
    for i in list(data):
        if i<mi:
            i=mi
            arr.append(i)
        elif i>ma:
            i=ma
            arr.append(i)
        else:
            arr.append(i)
    #print(max(arr))
    return arr
```

```
In [26]: df['bmi'] = remove_outliers(df['bmi'])
df['avg_glucose_level'] = remove_outliers(df['avg_glucose_level'])
print('Outliers successfully removed')

Outliers successfully removed
```

Checking whether the outliers are removed or not and dropping ID column from the dataset

```
In [27]: fig, axes = plt.subplots(nrows = 2, ncols = 2, figsize = (10, 5))
sns.boxplot(x = 'bmi', data = df, ax=axes[1][0])
sns.boxplot(x = 'avg_glucose_level', data = df, ax=axes[0][0])

Out[27]: <AxesSubplot:xlabel='avg_glucose_level'>
```

Out[27]: <AxesSubplot:xlabel='avg_glucose_level'>



In [28]:
```
df = df.drop('id', axis = 1)
```

In [29]:
```
df.shape
```

Out[29]: (4909, 11)

Milestone4:

Processing Categorical Data In machine learning, we usually deal with datasets that contain multiple labels in one or more than one columns. These labels can be in the form of words or numbers. To make the data understandable or in human-readable form, the training data is often labelled in words.

Activity1:Label Encoding on Categorical Variables Label Encoding refers to converting the labels into the numeric form so as to convert them into the machine-readable f

Preprocessing

In [30]:
```
from sklearn.preprocessing import LabelEncoder
le1 = LabelEncoder()
df['Residence_type'] = le1.fit_transform(df['Residence_type'])
df['ever_married'] = le1.fit_transform(df['ever_married'])
df['gender'] = le1.fit_transform(df['gender'])
df['work_type'] = le1.fit_transform(df['work_type'])
df['smoking_status'] = le1.fit_transform(df['smoking_status'])
```

In [31]:
```
print(df['smoking_status'].unique())
```

```
[0 1 2]
```

In [32]:
```
df.head()
```

Out[32]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.0	0	1	1	2	1	168.32	36.6	0	1
2	1	80.0	0	1	1	2	0	105.92	32.5	1	1
3	0	49.0	0	0	1	2	1	168.32	34.4	2	1
4	0	79.0	1	0	1	3	0	168.32	24.0	1	1
5	1	81.0	0	0	1	2	1	168.32	29.0	0	1

In [33]:
```
df[200:220]
```

Out[33]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
239	1	60.0	0	1	1	2	1	91.92	35.9	2	1
240	1	66.0	0	0	1	2	0	76.46	21.2	0	1
241	1	57.0	0	0	1	2	0	168.32	34.5	0	1
242	1	68.0	0	0	1	2	0	168.32	42.4	1	1
243	0	68.0	1	1	1	2	1	168.32	40.5	0	1
244	1	57.0	0	0	1	2	0	84.96	36.7	1	1
245	0	14.0	0	0	0	4	0	57.93	30.9	1	1
246	0	75.0	0	0	1	3	0	78.80	29.3	0	1
248	0	78.0	0	0	1	2	0	78.81	19.6	1	1
249	1	3.0	0	0	0	4	0	95.12	18.0	1	0
250	1	58.0	1	0	1	2	1	87.96	39.2	1	0
251	0	8.0	0	0	0	2	1	110.89	17.6	1	0
252	0	70.0	0	0	1	2	0	69.04	35.9	0	0
253	1	14.0	0	0	0	1	0	161.28	19.1	1	0
254	0	47.0	0	0	1	2	1	168.32	47.5	1	0
255	0	52.0	0	0	1	2	1	77.59	17.7	0	0
256	0	75.0	0	1	1	3	0	168.32	27.0	1	0
257	0	32.0	0	0	1	2	0	77.67	32.3	2	0
258	0	74.0	1	0	1	3	1	168.32	47.5	1	0
259	0	79.0	0	0	1	0	1	77.08	35.0	1	0

In [34]:
```
import joblib
joblib.dump(le1,"transform")
```

Out[34]:
```
['transform']
```

In [35]:
```
df.shape
```

Out[35]:
```
(4909, 11)
```

In [36]:
```
df.iloc[0,:]
```

Out[36]:
```
gender                1.00
age                  67.00
hypertension          0.00
heart_disease         1.00
ever_married          1.00
work_type             2.00
Residence_type        1.00
avg_glucose_level   168.32
bmi                  36.60
smoking_status        0.00
stroke                1.00
Name: 0, dtype: float64
```

In [37]:
```
df.head()
```

Out[37]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.0	0	1	1	2	1	168.32	36.6	0	1
2	1	80.0	0	1	1	2	0	105.92	32.5	1	1
3	0	49.0	0	0	1	2	1	168.32	34.4	2	1
4	0	79.0	1	0	1	3	0	168.32	24.0	1	1
5	1	81.0	0	0	1	2	1	168.32	29.0	0	1

In [38]:
```
df.shape
```

Out[38]:
```
(4909, 11)
```

In [39]:
```
X = df.iloc[:,0:10].values
y = df.iloc[:,10].values
```

Milestone 5:

Model Building We will be using the features to build the model by splitting them into dependent and independent variables and balance the dataset using oversampling

Activity1: Perform Oversampling Imbalanced datasets are those where there is a severe skew in the class distribution, such as 1:100 or 1:1000 examples in the minority class to the majority class. This bias in the training dataset can influence many machine learning algorithms, leading some to ignore the minority class entirely. This is a problem as it is typically the minority class on which predictions are most important. One approach to addressing the problem of class imbalance is to randomly resample the training dataset. The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling.

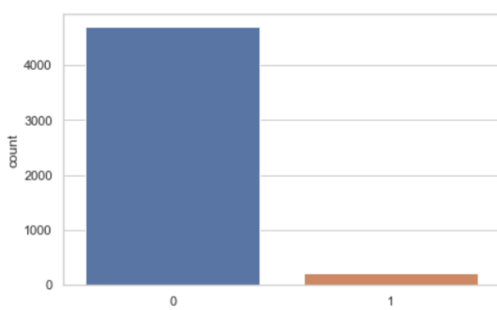
Model

In [40]:
```
X.shape
```
Out[40]: (4909, 10)

In [41]:
```
y.shape
```
Out[41]: (4909,)

We use sampling because there is an imbalance of the target feature

In [42]:
```
### before oversampling
sns.countplot(y)
```
Out[42]: <AxesSubplot:ylabel='count'>



In [43]:
```
from imblearn.over_sampling import SMOTE

sm = SMOTE()
X_res, y_res = sm.fit_resample(X, y)

print("Before OverSampling, counts of label '1': {}".format(sum(y==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y==0)))

print('After OverSampling, the shape of train_X: {}'.format(X_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_res==0)))
```

Activity2:

Splitting Data to train and test Here we split the dataset into train and test data so that we can use train data to build the model.

Modeling

In [45]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

Activity3:

Building models using algorithms

Decision Tree

In [46]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
dtc_pred = dtc.predict(X_test)
print(confusion_matrix(dtc_pred, y_test))
print('-----')
print(classification_report(dtc_pred, y_test))
```

Random Forest

In [47]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
print(confusion_matrix(rf_pred, y_test))
print('-----')
print(classification_report(rf_pred, y_test))
```

Logistic Regression

```
In [48]: lr = LogisticRegression()
         lr.fit(X_train, y_train)
         lr_pred = lr.predict(X_test)
         print(confusion_matrix(lr_pred, y_test))
         print('-----')
         print(classification_report(lr_pred, y_test))
```

SVC

```
In [49]: svc = SVC()
         svc.fit(X_train, y_train)
         svc_pred = svc.predict(X_test)
         print(confusion_matrix(svc_pred, y_test))
         print('-----')
         print(classification_report(svc_pred, y_test))
```

KNN

```
In [50]: knn = KNeighborsClassifier()
         knn.fit(X_train, y_train)
         knn_pred = knn.predict(X_test)
         print(confusion_matrix(knn_pred, y_test))
         print('-----')
         print(classification_report(knn_pred, y_test))
```

Activity4:

Appying Grid SearchCV Here we apply the grid search to get best hyper tuning parameters

Grid Search

```
In [51]: cross_valid_scores = {}
```

Decision Tree

```
In [52]: %%time
parameters = {
    "max_depth": [3, 5, 7, 9, 11, 13],
}

model_dtc = DecisionTreeClassifier(
    random_state=42,
    class_weight='balanced',
)

model_dtc = GridSearchCV(
    model_dtc,
    parameters,
    cv=5,
)

model_dtc.fit(X_train, y_train)
model_dtc_pred = model_dtc.predict(X_test)
print(classification_report(model_dtc_pred, y_test))

print('-----')
print(f'Best parameters {model_dtc.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: ' + \
    f'{model_dtc.best_score_:.3f}'
)
cross_valid_scores['desicion_tree'] = model_dtc.best_score_
print('-----')
```

Random Forest

```
In [53]: %%time
parameters = {
    "n_estimators": [5, 10, 15, 20, 25],
    "max_depth": [3, 5, 7, 9, 11, 13],
}

model_rf = RandomForestClassifier(
    random_state=42,
    class_weight='balanced',
)

model_rf = GridSearchCV(
    model_rf,
    parameters,
    cv=5,
)

model_rf.fit(X_train, y_train)
model_rf_pred = model_rf.predict(X_test)
print(classification_report(model_rf_pred, y_test))

print('-----')
print(f'Best parameters {model_rf.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: '+ \
    f'{model_rf.best_score_:.3f}'
)
cross_valid_scores['random_forest'] = model_rf.best_score_
print('-----')
```

```
In [54]: import pickle
pickle.dump(lr,open("modelor.pkl","wb"))
```

XGBoost

In [55]:
```
%%time
parameters = {
    'max_depth': [3, 5, 7, 9],
    'n_estimators': [5, 10, 15, 20, 25, 50, 100],
    'learning_rate': [0.01, 0.05, 0.1]
}

model_xgb = xgb.XGBClassifier(
    random_state=42, verbosity = 0
)

model_xgb = GridSearchCV(
    model_xgb,
    parameters,
    cv=5,
    scoring='accuracy',
)

model_xgb.fit(X_train, y_train)
model_xgb_pred = model_xgb.predict(X_test)
print(classification_report(model_xgb_pred, y_test))


print('-----')
print(f'Best parameters {model_xgb.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: ' +
    f'{model_xgb.best_score_:.3f}'
)
cross_valid_scores['xgboost'] = model_xgb.best_score_
print('-----')
```

LightGBM

In [56]:
```
%%time
parameters = {
    'n_estimators': [5, 10, 15, 20, 25, 50, 100],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [7, 15, 31],
}

model_lgbm = lgbm.LGBMClassifier(
    random_state=42,
    class_weight='balanced',
)

model_lgbm = GridSearchCV(
    model_lgbm,
    parameters,
    cv=5)

model_lgbm.fit(X_train, y_train)
model_lgbm_pred = model_lgbm.predict(X_test)
print(classification_report(model_lgbm_pred, y_test))

print('-----')
print(f'Best parameters {model_lgbm.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: ' +
    f'{model_lgbm.best_score_:.3f}'
)
cross_valid_scores['lightgbm'] = model_lgbm.best_score_
print('-----')
```

Adaboost

In [57]:
```
%%time
parameters = {
    "n_estimators": [5, 10, 15, 20, 25, 50, 75, 100],
    "learning_rate": [0.001, 0.01, 0.1, 1.],
}

model_adaboost = AdaBoostClassifier(
    random_state=42,
)

model_adaboost = GridSearchCV(
    model_adaboost,
    parameters,
    cv=5,
    scoring='accuracy',
)

model_adaboost.fit(X_train, y_train)
model_adaboost_pred = model_adaboost.predict(X_test)
print(classification_report(model_adaboost_pred, y_test))

print('-----')
print(f'Best parameters {model_adaboost.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: '+ \
    f'{model_adaboost.best_score_:.3f}'
)
cross_valid_scores['ada_boost'] = model_adaboost.best_score_
print('-----')
```

Logistic Regression

In [58]:
```
%%time
parameters = {
    "C": [0.001, 0.01, 0.1, 1.],
    "penalty": ["l1", "l2"]
}

model_lr = LogisticRegression(
    random_state=42,
    class_weight="balanced",
    solver="liblinear",
)

model_lr = GridSearchCV(
    model_lr,
    parameters,
    cv=5,
    scoring='accuracy',
)

model_lr.fit(X_train, y_train)
model_lr_pred = model_lr.predict(X_test)
print(classification_report(model_lr_pred, y_test))

print('-----')
print(f'Best parameters {model_lr.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: ' +
    f'{model_lr.best_score_:.3f}'
)
cross_valid_scores['logistic_regression'] = model_lr.best_score_
print('-----')
```

KNN

In [59]:
```
%%time
parameters = {
    "weights": ["uniform", "distance"],
}

model_k_neighbors = KNeighborsClassifier(
)

model_k_neighbors = GridSearchCV(
    model_k_neighbors,
    parameters,
    cv=5,
    scoring='accuracy',
)

model_k_neighbors.fit(X_train, y_train)
model_k_neighbors_pred = model_k_neighbors.predict(X_test)
print(classification_report(model_k_neighbors_pred, y_test))

print('-----')
print(f'Best parameters {model_k_neighbors.best_params_}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: ' +
    f'{model_k_neighbors.best_score_:.3f}'
)
cross_valid_scores['k_neighbors'] = model_k_neighbors.best_score_
print('-----')
```

In [60]:
```
import pickle
pickle.dump(df,open("model.pkl","wb"))
```

Milestone 6:

Application Building After the model is built, we will be integrating it into a web application

Activity 1:
Build the python flask app In the flask application, the user values are taken from the HTML page.

```python
from flask import Flask, request,render_template
import joblib
import numpy as np
app = Flask(__name__)
model = joblib.load("model")
label1 = joblib.load("mar_transform")
label2 = joblib.load("res_transform")
column = joblib.load("column")

app = Flask(__name__)

```

Load the home page

```python
@app.route('/')
def predict():
    return render_template('Manual_predict.html')
```

Prediction function

```python
@app.route('/y_predict',methods=['POST'])
def y_predict():
    x_test = [[(x) for x in request.form.values()]]
    print('actual',x_test)
    x_test=np.array(x_test)
    x_test[:,4]=label1.transform(x_test[:,4])
    x_test[:,6]=label2.transform(x_test[:,6])
    x_test=column.transform(x_test)
    pred = model.predict(x_test)
    print(pred)
    if(pred[0]==0):
        result="no chances of stroke"
    else:result="chances of stroke"

    return render_template('Manual_predict.html', \
                           prediction_text=('There are \
                                 ',result))
```

Activity 2:

Build an HTML Page We Build an HTML page to take the values from the user in a form and upon clicking on the predict button we get the prediction

```
<html>
<head>
<title>
    Prediction
</title>
<link href='https://fonts.googleapis.com/css?family=Montserrat' rel='stylesheet'>
<style>
        * {
            box-sizing: border-box;
        }

        body {
            font-family: 'Montserrat' ;
        }

        .header {
            top:0;
            margin:0px;
            left: 0px;
            right: 0px;
            position: fixed;
            background-color: black;
            color: white;
            box-shadow: 0px 8px 4px grey;
            overflow: hidden;
            padding: 15px;
            font-size: 2vw;
            width: 100%;
            text-align: left;
            padding-left: 100px;
            opacity:0.9;
        }
        .header_text{
            font-size:40px;
            text-align:center;
        }
        .content{
        margin-top:100px;
        }
```

```
        .text{
            font-size:20px;
            margin-top:10px;
            text-align:center;
        }
        input[type=number], select {
  width: 50%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
        input[type=text], select {
  width: 50%;
  padding: 12px 20px;
  margin: 8px 0;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
input[type=submit] {
  width: 50%;
  background-color: #000000;
  color: white;
  padding: 14px 20px;
  margin: 8px 0;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
```

```
input[type=submit]:hover {
  background-color: #5d6568;
  color:#ffffff;
  border-color:black;
}
form{
margin-top:20px;
}
.result{
color:black;
margin-top:30px;
margin-bottom:20px;
font-size:25px;
color:red;
}
</style>
</head>
<body align=center>
<div class="header">
        <div>Stroke Prediction </div>
</div>
<div class="content">
<div class="header_text">Stroke Prediction</div>
<div class="text">Fill in and below details to predict whether a person might get a stroke.</div>
<div class="result">
  {{ prediction_text }}
  </div>
```

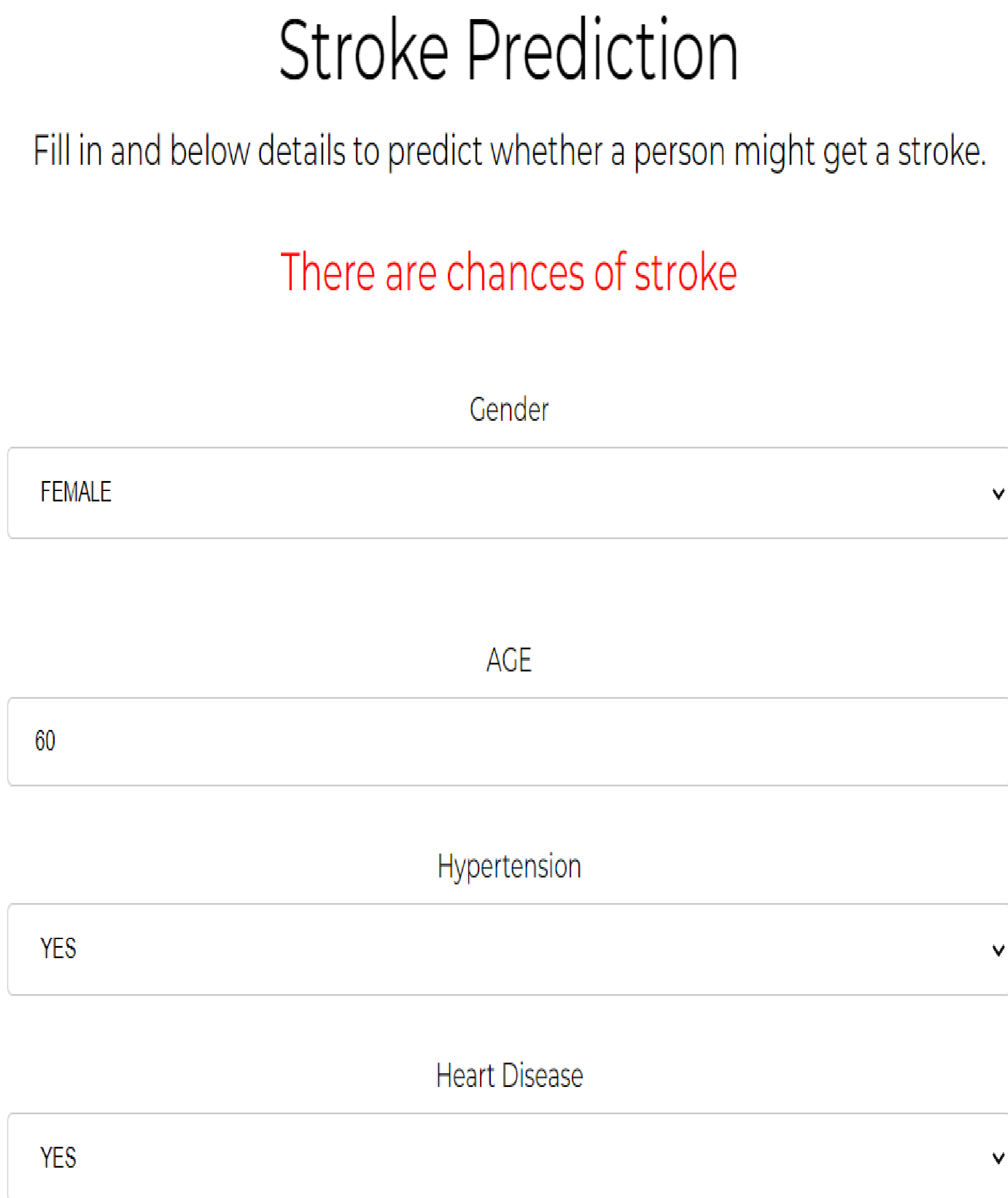
```
<form action="{{ url_for('y_predict') }}" method="POST">
    <input type="text" id="gender" name="Gender" placeholder="gender">
    <input type="number" id="age" name="Age" placeholder="age">
    <input type="number"  id="hypertension" name="Hypertension" placeholder="hypertension(0/1)">
    <input type="number"  id="heart_disease" name="Heart Disease" placeholder="heart_disease(0/1)">
    <input type="text"  id="ever_married" name="Ever Married" placeholder="ever_married">
    <input type="text"  id="work_type" name="Work Type" placeholder="work_type">
    <input type="text"  id="Residence_type" name="Residence Type" placeholder="Residence_type">
    <input type="number" step="any" id="avg_glucose_level" name="avg glucose level" placeholder="Avg Glucose Level">
    <input type="number" step="any" id="bmi" name="BMI" placeholder="BMI">
    <input type="text"  id="smoking_status" name="smoking_status" placeholder="smoking_status">

    <input type="submit" value="Submit">
  </form>

</div>
</body>
</html>
```

Activity 3:

Run the application Step 1: Open anaconda prompt go to the project folder and in that go to flask folder and run the python file by using the command “python app.py”

```
* Restarting with stat
* Debugger is active!
* Debugger PIN: 301-111-576
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Output


Stroke Prediction
Fill in and below details to predict whether a person might get a stroke.
There are chances of stroke
Gender
FEMALE
AGE
60
Hypertension
YES
Heart Disease
YES

Ever Married

YES

Work Type

Self-Employed

Residence Type

Urban

avg glucose level

230

BMI

160

smoking_status

Smokes

Submit

Stroke Prediction

Fill in and below details to predict whether a person might get a stroke.

There are no chances of stroke

Gender

MALE

AGE

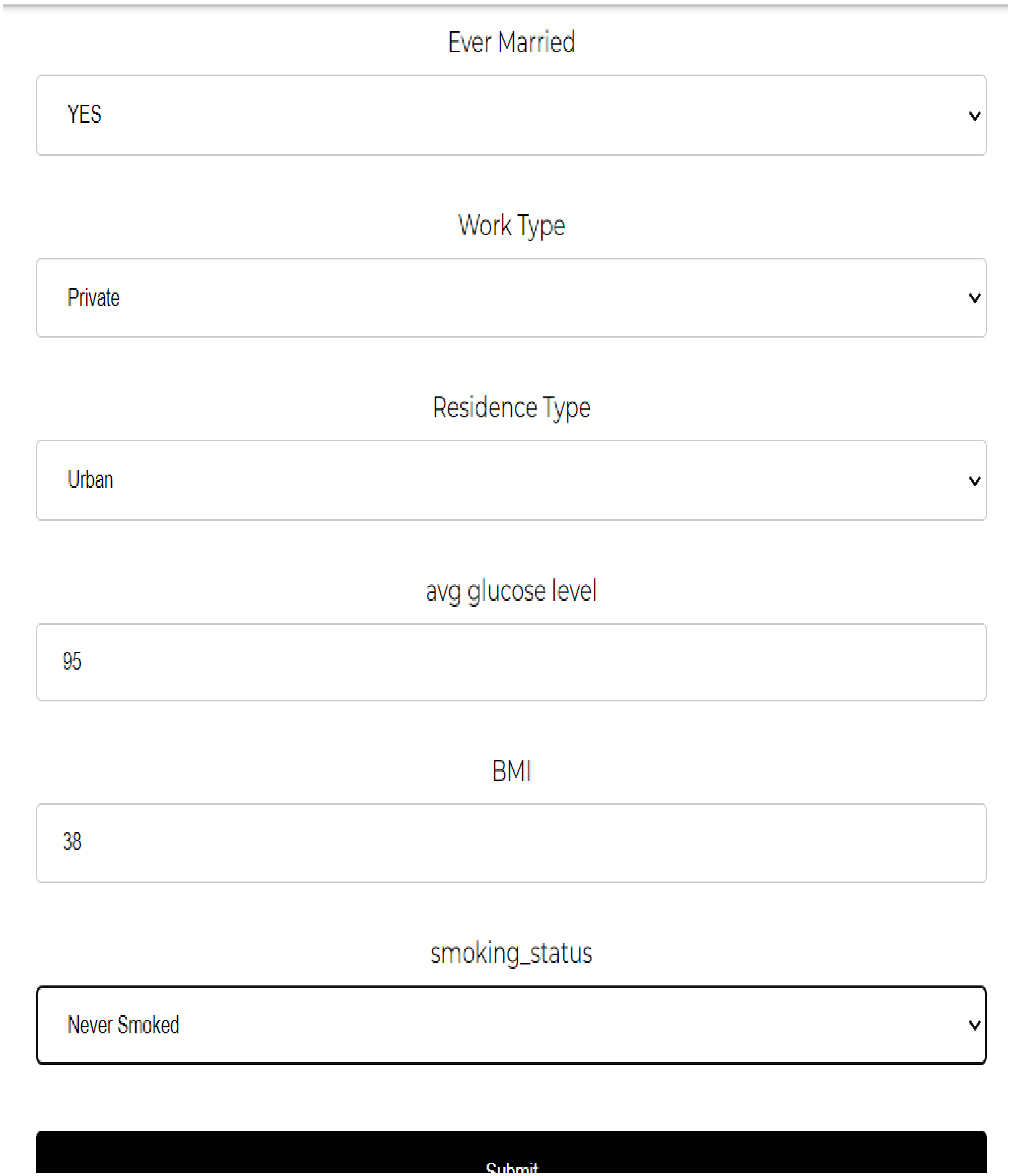
51

Hypertension

NO

Heart Disease

NO



Thank for watching!