MA336: Artificial intelligence and machine learning with applications

STROKE PREDICTION

Registration ID: 2111901

1.Introduction:

Machine Learning(ML) is a branch of computer science that is simply based on training a computer by leveraging a set of data to improve the performance of some tasks.Machine Learning algorithms are used to train the computer, based on that training data or algorithm provided it should perform the tasks given to it without being explicitily programmed to do so.By implementing various Machine Learning algorithms, we have tried to predict the factors which are main causes of stroke in an individual from the given sample dataset. In the following dataset we have features like gender,age, hypertension,heart_disease, smoking_status which can help us identify the individual with what kind of features is more likely to receive a stroke. I have decided to go for Stroke prediction as stroke in individuals has been an alarming issue in the world and according to WHO data, annually, 15 million people suffer from a stroke out of which 5 million die from stroke and 5 million are left disabled which puts a drastic burden on the family. With the help of this project I would like to point out the factors responsible for stroke which will help to identify the underlying problem in the Healthcare Sector and providing insights on it using Machine Learning algorithms like Logistic Regression, Random Forest Classifiers (RFC), Decision Tree Classifiers(CART).

Data Loading and Pre-Processing

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plot
%matplotlib inline
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
```

In [3]:

```
##To read the heart stroke data.
new_data = pd.read_csv("healthcare-dataset-stroke-data (1).csv")
```

In [4]:

```
## To see the first five observations of the dataframe.
```

```
new_data.head(10)
```

Out[4]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_ty
0	9046	Male	67.0	0	1	Yes	Private	Urb
1	51676	Female	61.0	0	0	Yes	Self-employed	Ru
2	31112	Male	80.0	0	1	Yes	Private	Ru
3	60182	Female	49.0	0	0	Yes	Private	Urb
4	1665	Female	79.0	1	0	Yes	Self-employed	Ru
5	56669	Male	81.0	0	0	Yes	Private	Urb
6	53882	Male	74.0	1	1	Yes	Private	Ru
7	10434	Female	69.0	0	0	No	Private	Urb
8	27419	Female	59.0	0	0	Yes	Private	Ru
9	60491	Female	78.0	0	0	Yes	Private	Urb

In [5]:

```
## To see the total number of rows and columns.
new_data.shape
```

Out[5]:

```
(5110, 12)
```

In [6]:

```
## The function info() represents the data types of every variable.
new_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 [7]:

```
## To check the null values present in our data frame.
new_data.isnull().sum()
```

Out[7]:

```
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
```

Data Cleaning

In [8]:
```
## Imputation of null values that is present in 'bmi' with median.
new_data['bmi']=new_data['bmi'].fillna(new_data['bmi'].median())
```

In [9]:
```
## To check again whether our above method removed the null values or not.
new_data.isnull().sum()
```

Out[9]:
```
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 [10]:
```
## To present the statistical summary of the dataframe.
new_data.describe()
```

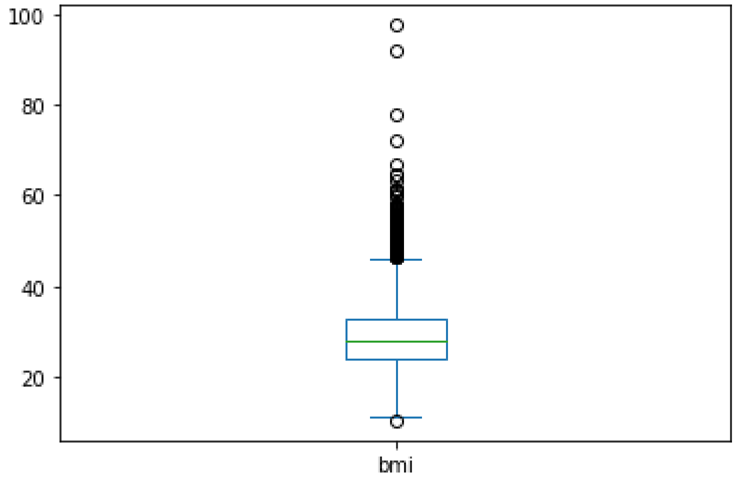
Out[10]:

	id	age	hypertension	heart_disease	avg_glucose_level	bı
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.00000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.8620
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.6995
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.3000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.8000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.1000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	32.8000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.6000

Exploratory data analysis (EDA)

In [11]:
```
## BMI Boxplot to identify Outliers
new_data['bmi'].plot.box()
```

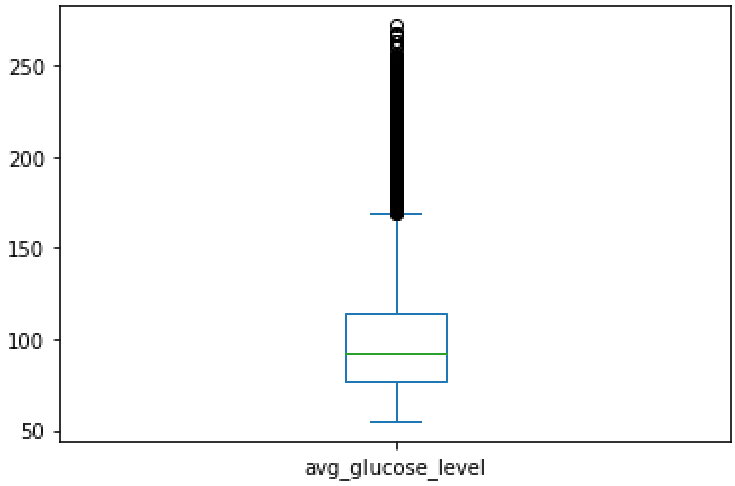
Out[11]: <AxesSubplot:>



In [12]:

```
## Average Glucose Level Boxplot to show Outliers
new_data['avg_glucose_level'].plot.box()
```

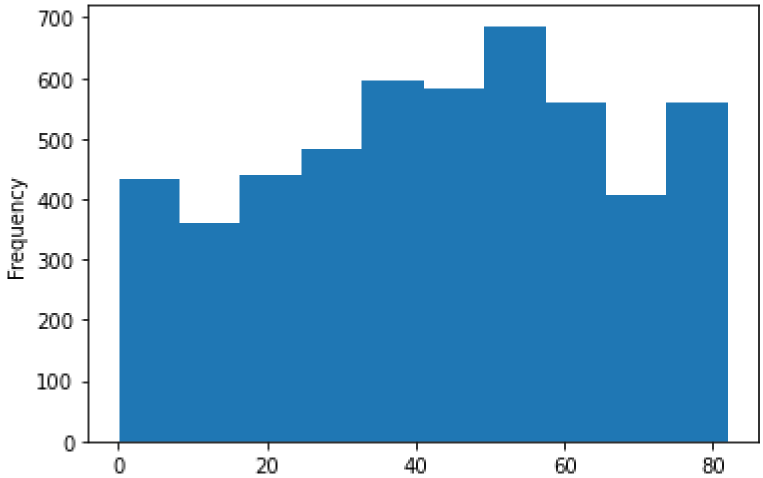
Out[12]: <AxesSubplot:>



In [13]:

```
## To check the frequency of the 'age' with the help of histogram.
new_data['age'].plot.hist()
```

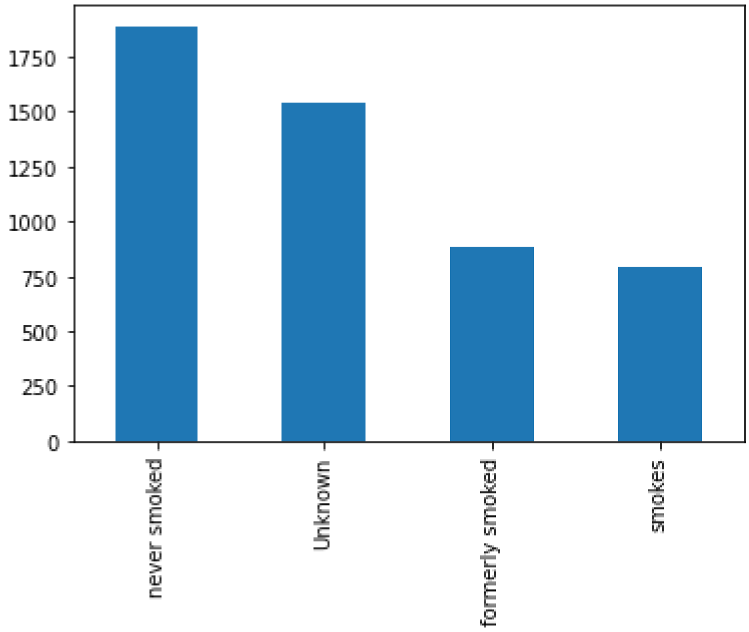
Out[13]: <AxesSubplot:ylabel='Frequency'>



In [14]:

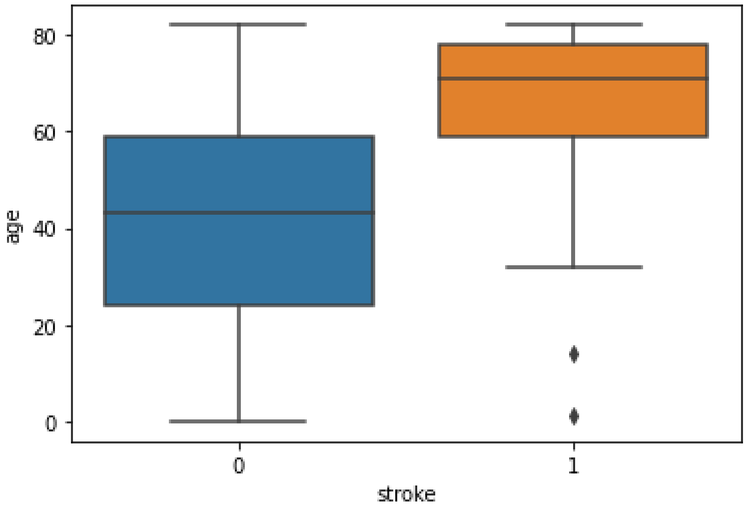
```
## To count different levels of smoking status and plotting a bar chart.
new_data['smoking_status'].value_counts().plot.bar()
```

Out[14]: <AxesSubplot:>



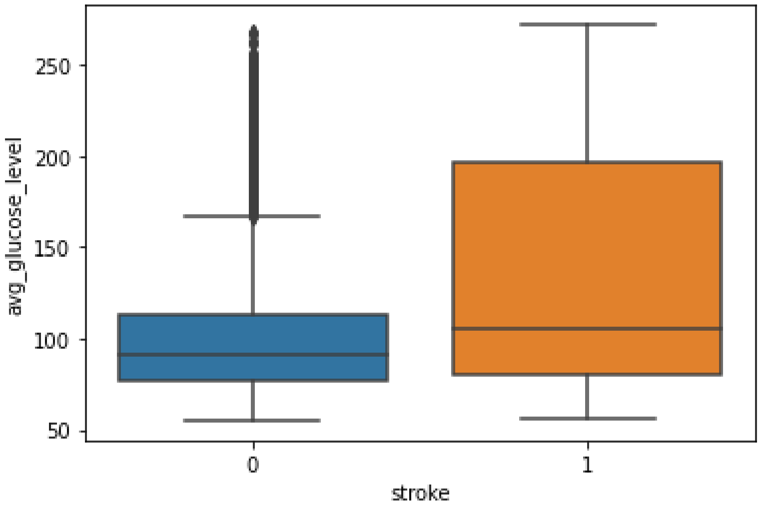
In [15]:
```
## To see the rate of stroke amoung different age groupes.
sns.boxplot(data=new_data,y=new_data['age'],x=new_data['stroke'])
```

Out[15]: <AxesSubplot:xlabel='stroke', ylabel='age'>



In [16]:
```
## To check whether glucose level has any affect on stroke
sns.boxplot(data=new_data,y=new_data['avg_glucose_level'],x=new_data['stroke'
```

Out[16]: <AxesSubplot:xlabel='stroke', ylabel='avg_glucose_level'>



In [17]:

```
# Cross-Tabulation of Gender and Stroke variables for plotting 2 categorical
gender_stroke=pd.crosstab(new_data['gender'],new_data['stroke'])
gender_stroke
```

Out[17]:

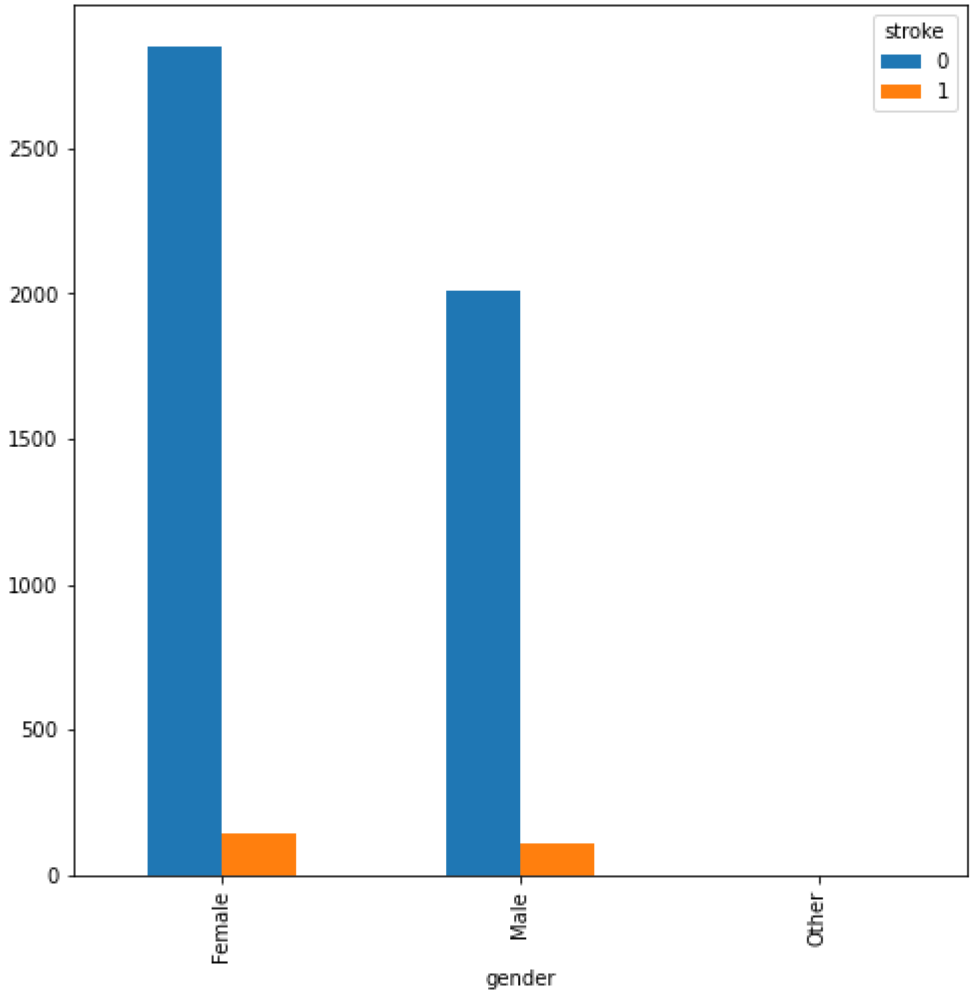
stroke	0	1
gender		
Female	2853	141
Male	2007	108
Other	1	0

In [18]:

```
# Gender-Stroke Plot
gender_stroke.plot(kind='bar',figsize=(8,8))
```

Out[18]:

```
<AxesSubplot:xlabel='gender'>
```



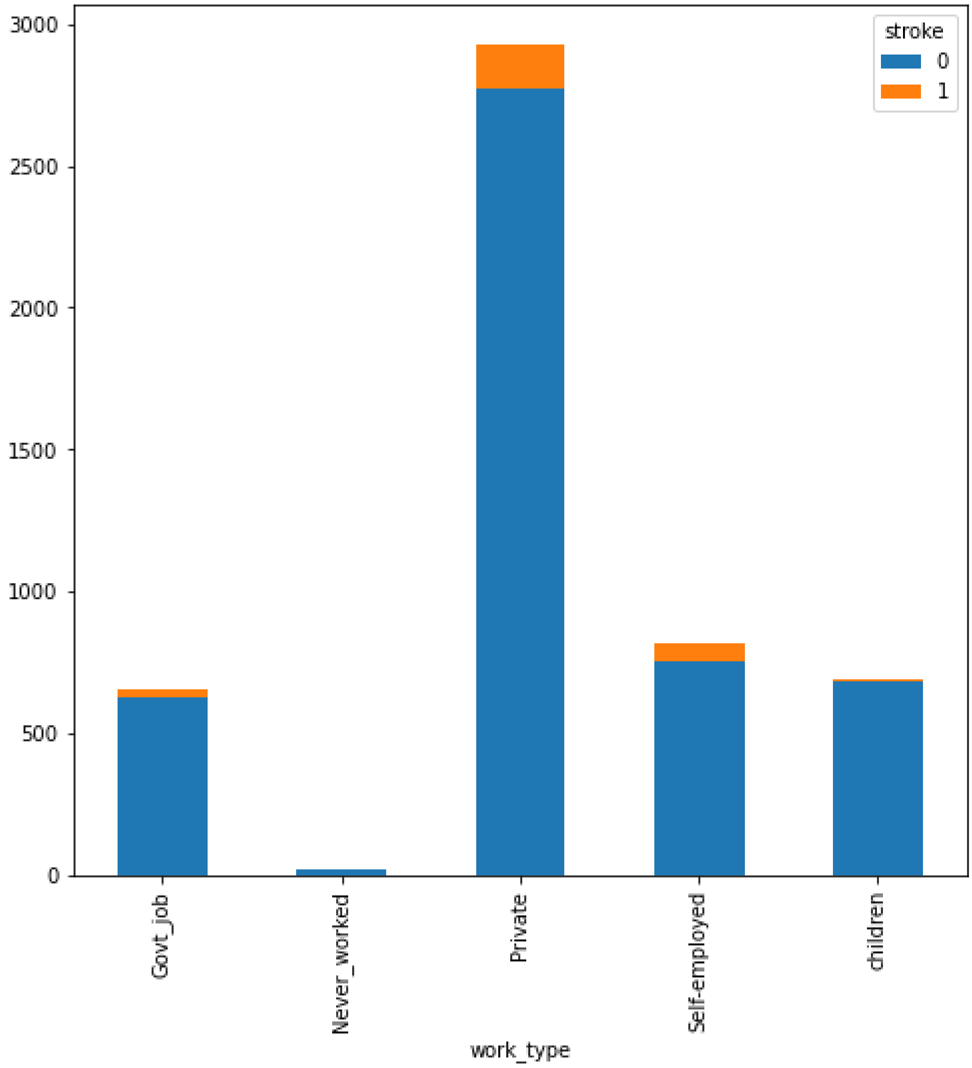
In [19]:
```
worktype_stroke=pd.crosstab(new_data['work_type'],new_data['stroke'])
worktype_stroke
```

Out[19]:

stroke	0	1
work_type		
Govt_job	624	33
Never_worked	22	0
Private	2776	149
Self-employed	754	65
children	685	2

In [20]:
```
worktype_stroke.plot(kind='bar',figsize=(8,8), stacked=True)
```

Out[20]: <AxesSubplot:xlabel='work_type'>



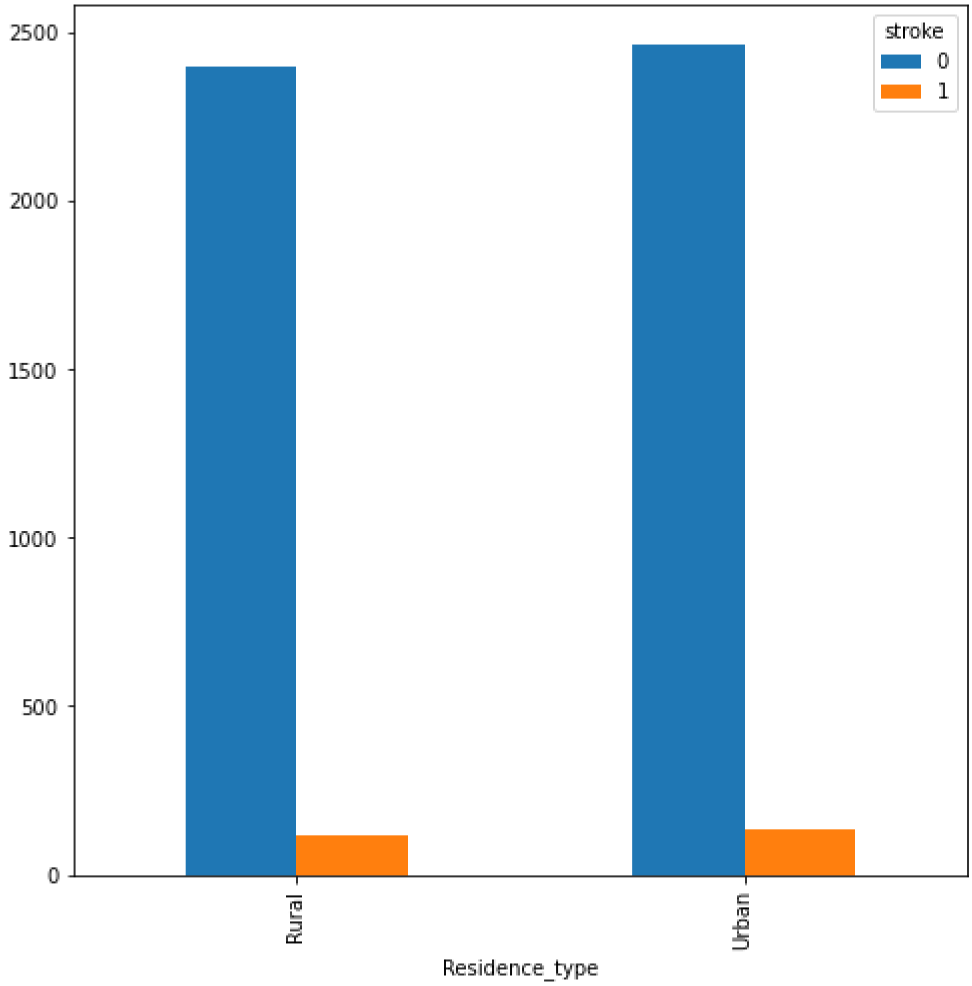
In [21]:
```
residence_stroke=pd.crosstab(new_data['Residence_type'],new_data['stroke'])
residence_stroke
```

Out[21]:

stroke	0	1
Residence_type		
Rural	2400	114
Urban	2461	135

In [22]:
```
residence_stroke.plot(kind='bar',figsize=(8,8))
```

Out[22]: <AxesSubplot:xlabel='Residence_type'>

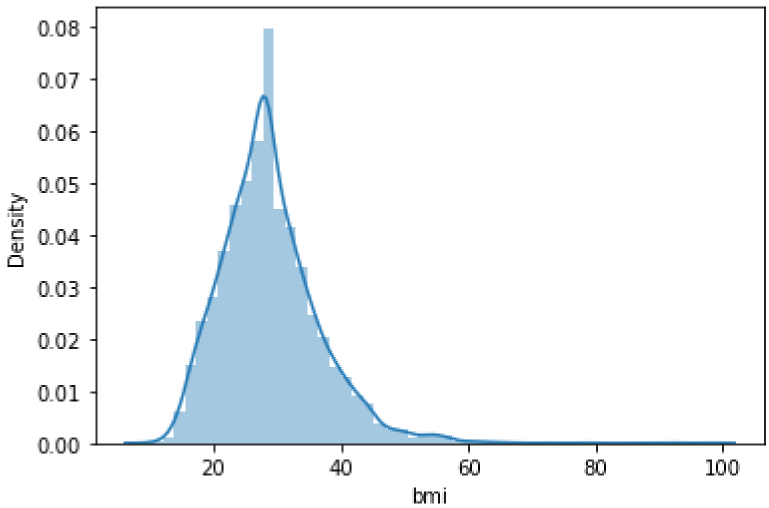


In [ ]:

In [23]:
```
sns.distplot(new_data['bmi'])
```

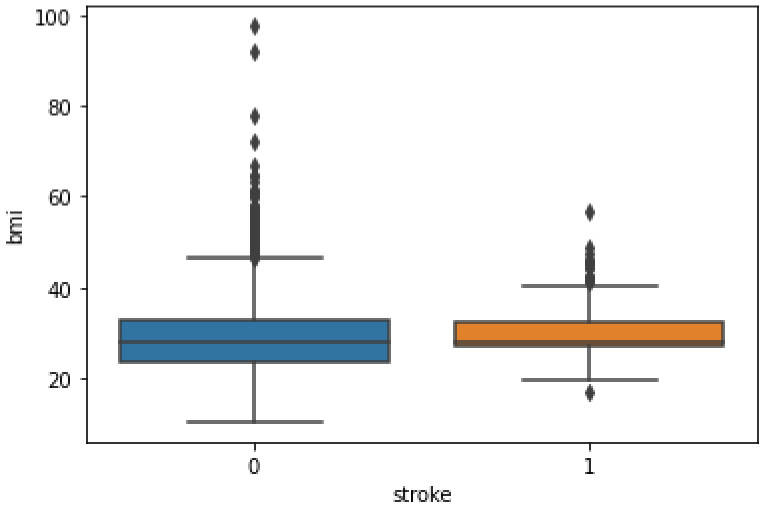
```
/Users/rudramm0205/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[23]: <AxesSubplot:xlabel='bmi', ylabel='Density'>



In [24]:
```
sns.boxplot(data=new_data,y=new_data['bmi'],x=new_data['stroke'])
```

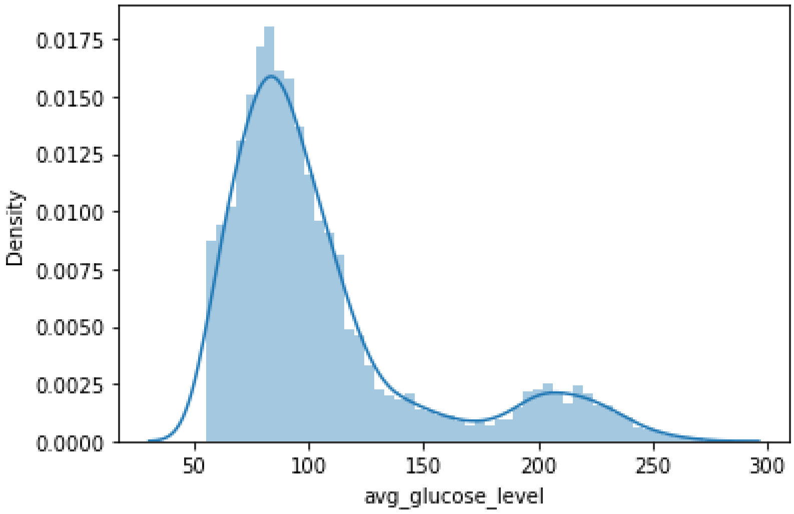
Out[24]:
```
<AxesSubplot:xlabel='stroke', ylabel='bmi'>
```



In [25]:
```
sns.distplot(new_data['avg_glucose_level'])
```

```
/Users/rudramm0205/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

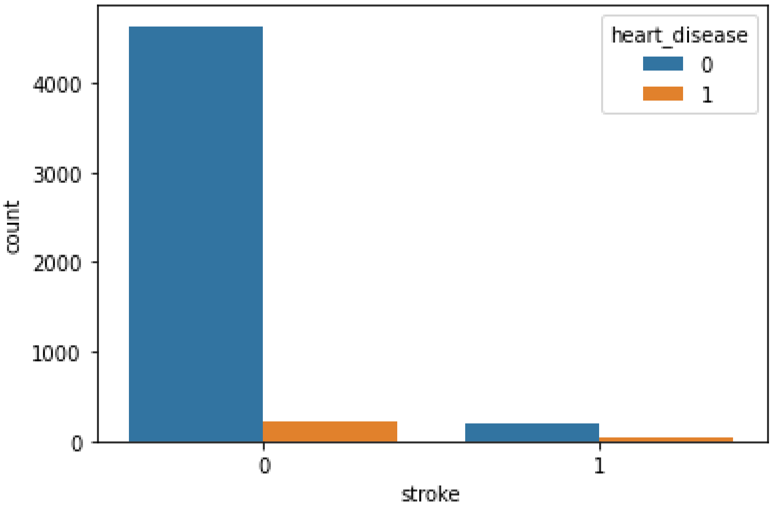
Out[25]:
```
<AxesSubplot:xlabel='avg_glucose_level', ylabel='Density'>
```



In [26]:
```
sns.countplot(new_data['stroke'],hue=new_data['heart_disease'])
```

```
/Users/rudramm0205/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

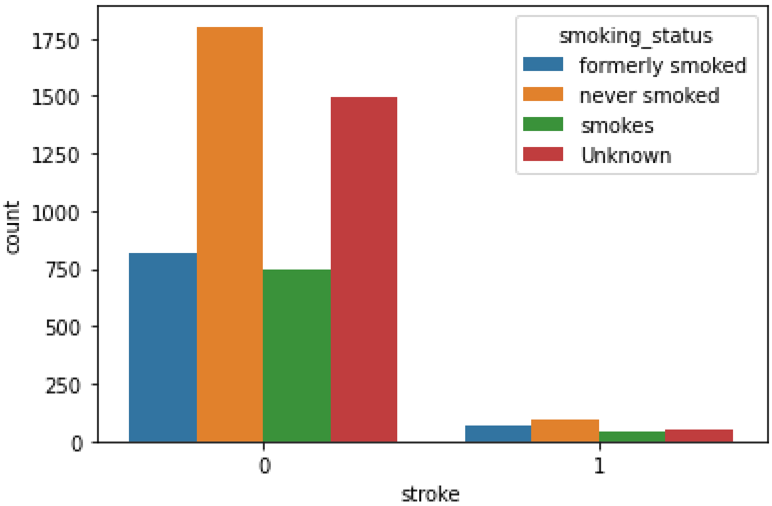
Out[26]:
```
<AxesSubplot:xlabel='stroke', ylabel='count'>
```



In [27]:
```
sns.countplot(new_data['stroke'],hue=new_data['smoking_status'])
```

```
/Users/rudramm0205/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorato
rs.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From
version 0.12, the only valid positional argument will be `data`, and passing o
ther arguments without an explicit keyword will result in an error or misinter
pretation.
  warnings.warn(
```

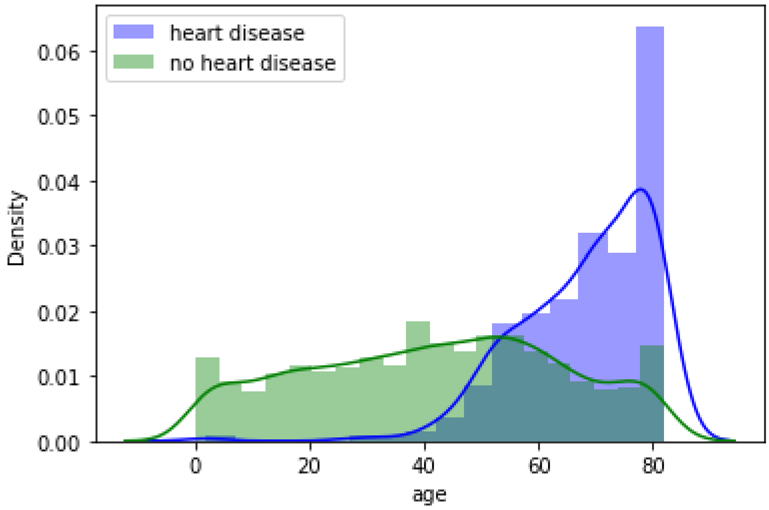
Out[27]:
```
<AxesSubplot:xlabel='stroke', ylabel='count'>
```



In [28]:
```
sns.distplot(new_data['age'][new_data['heart_disease']==1], kde=True, color='
sns.distplot(new_data['age'][new_data['heart_disease']==0], kde=True, color='
plot.legend()
```

```
/Users/rudramm0205/opt/anaconda3/lib/python3.9/site-packages/seaborn/distribut
ions.py:2619: FutureWarning: `distplot` is a deprecated function and will be r
emoved in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
/Users/rudramm0205/opt/anaconda3/lib/python3.9/site-packages/seaborn/distribut
ions.py:2619: FutureWarning: `distplot` is a deprecated function and will be r
emoved in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[28]:
```
<matplotlib.legend.Legend at 0x7fe048aeac70>
```



In [29]:

```
## To count the total number of males and females in our data frame.
new_data["gender"].value_counts()
```

Out[29]:

```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [30]:

```
## To count the total number of married and unmarried people in our data fram
new_data['ever_married'].value_counts()
```

Out[30]:

```
Yes    3353
No     1757
Name: ever_married, dtype: int64
```

In [31]:

```
## To count the different types of work present in our data frame.
new_data['work_type'].value_counts()
```

Out[31]:

```
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

In [32]:

```
## To count the types of residence found in our data frame.
new_data['Residence_type'].value_counts()
```

Out[32]:

```
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
```

In [33]:

```
## To check the correlation between all the continuous variables present in o
new_data.corr()
```

Out[33]:

	id	age	hypertension	heart_disease	avg_glucose_level	
id	1.000000	0.003538	0.003550	-0.001296	0.001092	0.005
age	0.003538	1.000000	0.276398	0.263796	0.238171	0.324
hypertension	0.003550	0.276398	1.000000	0.108306	0.174474	0.158
heart_disease	-0.001296	0.263796	0.108306	1.000000	0.161857	0.036

	id	age	hypertension	heart_disease	avg_glucose_level	
avg_glucose_level	0.001092	0.238171	0.174474	0.161857	1.000000	0.166
bmi	0.005555	0.324296	0.158293	0.036916	0.166876	1.000
stroke	0.006388	0.245257	0.127904	0.134914	0.131945	0.03

In [34]:

```
## To show the columns names present in our data frame.
new_data.columns
```

Out[34]:

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

In [35]:

```
## Used Label Encoder for converting the categorical variables into integer t
new_col = ["gender", "ever_married" ,"Residence_type","smoking_status","work_
new_encoder = preprocessing.LabelEncoder()
for col in new_col:
    new_data[col]=  new_encoder.fit_transform(new_data[col])
```

In [36]:

```
## To check whether it has changed the variables properly or not...
new_data.head()
```

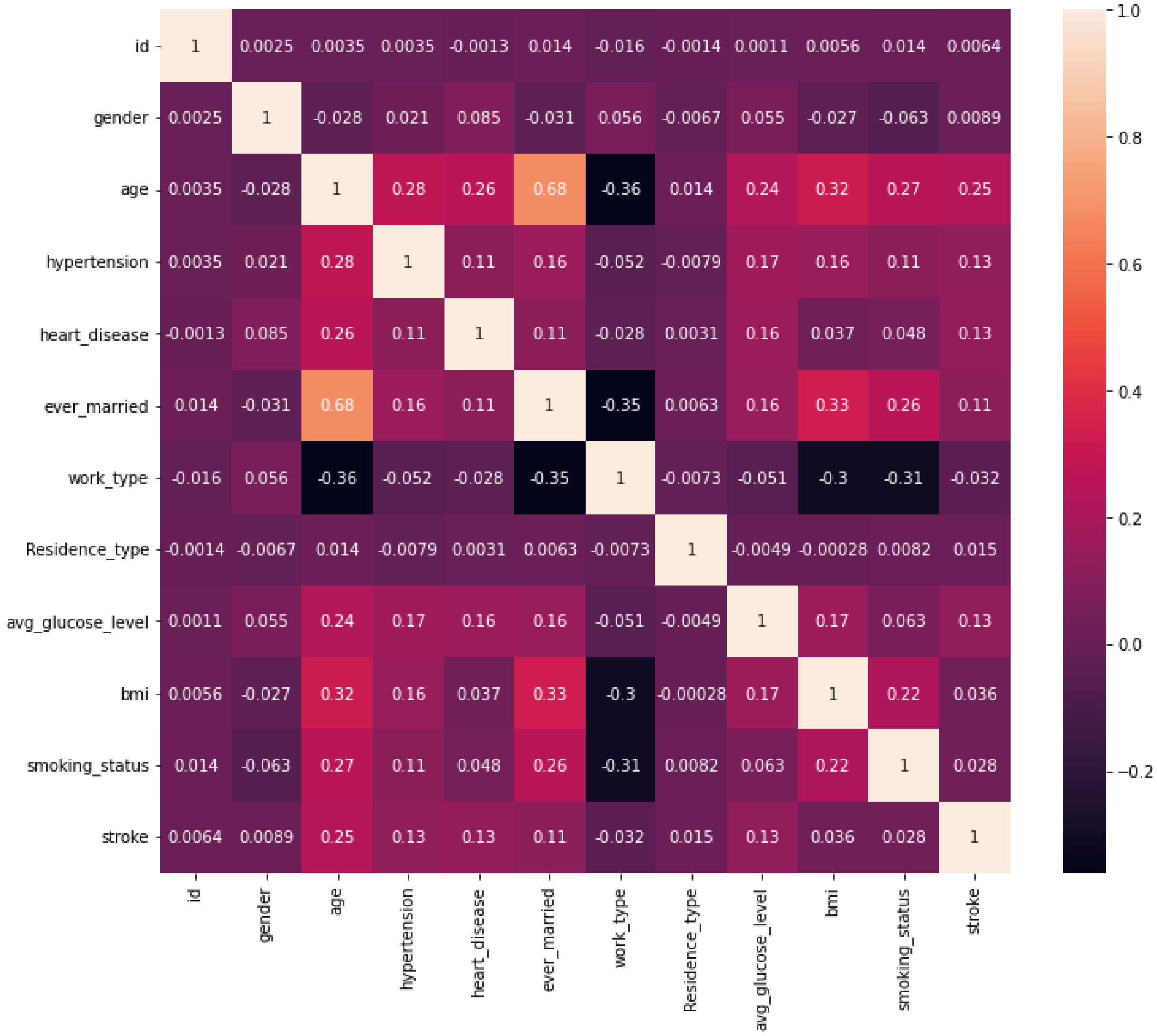
Out[36]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_typ
0	9046	1	67.0	0	1	1	2	
1	51676	0	61.0	0	0	1	3	
2	31112	1	80.0	0	1	1	2	
3	60182	0	49.0	0	0	1	2	
4	1665	0	79.0	1	0	1	3	

In [37]:

```
## This correlation heatmap takes all varibles into account and it shows that
c, axes = plot.subplots(figsize = (12,10))
sns.heatmap(new_data.corr(), annot=True, ax=axes)
```

Out[37]: <AxesSubplot:>



2.Methods

- In this Dataframe, we have total 12 Variables and 5 Categorical Variables.
- We have used Label Encoder to convert these 5 categorical Variables into Integer type.
- In BMI variable, we found there are some Missing(NA) Values, we Imputed the variable with Median.
- We Used Standard Scaler in SkLearn Package to remove outliers and to normalise the data.
- We have Splitted the data into 60:40 Train Test Ratio.
- Performed Oversampling using SMOTE function to Balance our Dataset which was rather imbalanced.

2.1. Machine learning Algorithms used:

1. Logistic Regression - We chose this method because our predictor variable stroke is a bi-variate variable which has values stroke(1) or no stroke(0).We are also going to compare it with other Models based on the accurancy, F1 accuracy score [F1 = 2 / (1/Precision + 1/Recall)] and Feature Importance of the Logistic Regression Model.
2. Random Forest Classifier - As it is a classification problem, Random Forest Classifier will be able to provide good accuracy and it helps to detect a class which is more infrequent

then other classes.

3. Decision Tree Classifier - This Classifier Algorithm is used to detect the most important features and also the relations between them. A Classification tree is used as a target to classify whether the patient has a stroke or no stroke.

In [38]:
```
## To drop the predictor variable 'stroke' from the independent variables for
x=new_data.drop(['stroke'],axis=1)
y=new_data['stroke']
```

In [39]:
```
x.shape
```

Out[39]:
```
(5110, 11)
```

In [40]:
```
y.shape
```

Out[40]:
```
(5110,)
```

In [41]:
```
## Performing Oversampling to Balance the imbalanced dataset
oversampling = SMOTE(random_state=123)
x_oversampling , y_oversampling = oversampling.fit_resample(x,y)

print(f'''Before SMOTE:{x.shape}
After SMOTE:{x_oversampling.shape}''',"\n")

print(f'''Distribution before SMOTE:\n{y.value_counts(normalize=True)}
Distribution after SMOTE :\n{y_oversampling.value_counts(normalize=True)}''')
```

```
Before SMOTE:(5110, 11)
After SMOTE:(9722, 11)

Distribution before SMOTE:
0    0.951272
1    0.048728
Name: stroke, dtype: float64
Distribution after SMOTE :
1    0.5
0    0.5
Name: stroke, dtype: float64
```

In [42]:
```
## To split the data into train and test for model building. We seperated int
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_oversampling,y_oversampling,
```

In [43]:
```
## To represent the number of observations after splitting.
x_train.shape,x_test.shape
```

Out[43]:
```
((5833, 11), (3889, 11))
```

In [44]:
```
## To count the total number of strokes present in our data frame.
new_data['stroke'].value_counts()
```

Out[44]:
```
0    4861
1     249
Name: stroke, dtype: int64
```

In [45]:
```
## Standard scaler is used for normalization of our training and testing data
sca=StandardScaler()
x_train=sca.fit_transform(x_train)
x_test=sca.fit_transform(x_test)
```

In [46]:
```
## For importing Logistic regression and its following metrices...
from sklearn.linear_model import LogisticRegression as LgRg
from sklearn.metrics import f1_score,accuracy_score,confusion_matrix,precisio
```

In [47]:
```
## To fit the Logistic Regression model into the training set and then predic
lg= LgRg()
lg.fit(x_train,y_train)
y_lg_pred = lg.predict(x_test)
score_lg=accuracy_score(y_test,y_lg_pred)*100
print("training accuracy score: ",accuracy_score(y_train,lg.predict(x_train))
print("testing accuracy score: ",score_lg)
print("F1 score", f1_score(y_train,lg.predict(x_train)))
```

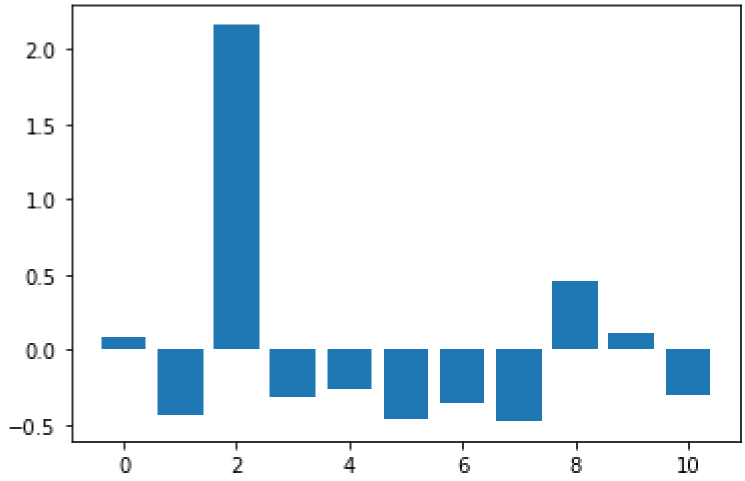
```
training accuracy score:  82.22184124807131
testing accuracy score:  82.18050912831062
F1 score 0.825919086788652
```

In [48]:
```
## Predicting Feature Importance of Logistic Regression Model
imp = lg.coef_[0]
```

In [49]:
```
for i,v in enumerate(imp):
        print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.08066
Feature: 1, Score: -0.43212
Feature: 2, Score: 2.16344
Feature: 3, Score: -0.31242
Feature: 4, Score: -0.25542
Feature: 5, Score: -0.45998
Feature: 6, Score: -0.35048
Feature: 7, Score: -0.47268
Feature: 8, Score: 0.46232
Feature: 9, Score: 0.11943
Feature: 10, Score: -0.30245
```

In [50]:
```
# Plotting Feature Importance of Each Variable
plot.bar([x for x in range(len(imp))], imp)
plot.show()
```



In [51]:

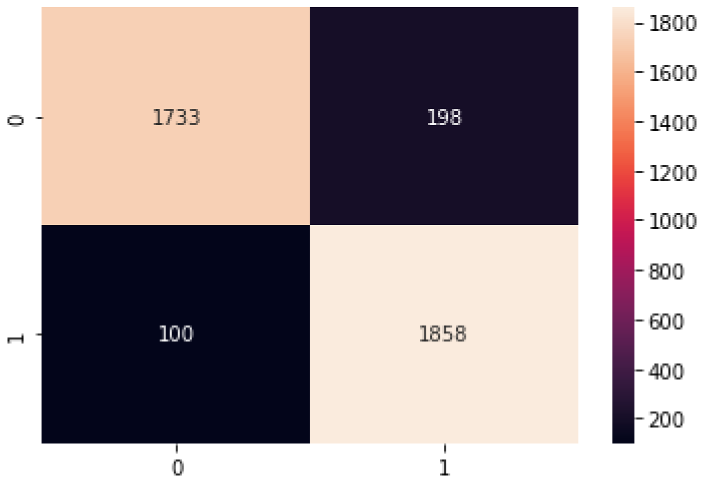
```
# Performing Random Forest Classifier on Training and Test Data Set
from sklearn.ensemble import RandomForestClassifier


rfc = RandomForestClassifier(random_state=200)
rfc = rfc.fit(x_train,y_train)
y_pred_rfc = rfc.predict(x_test)
ac = accuracy_score(y_test, y_pred_rfc)
print('Testing Accuracy score is:', ac)
print('Training Accuracy score is:',accuracy_score(y_train,rfc.predict(x_trai
cm = confusion_matrix(y_test, y_pred_rfc)
sns.heatmap(cm, annot = True, fmt = "d")
```

```
Testing Accuracy score is: 0.9233736178966315
Training Accuracy score is: 1.0
```

Out[51]:

```
<AxesSubplot:>
```



In [52]:

```
## Predicting Feature Importance of Random Forest Classifier
imp2 = rfc.feature_importances_
```

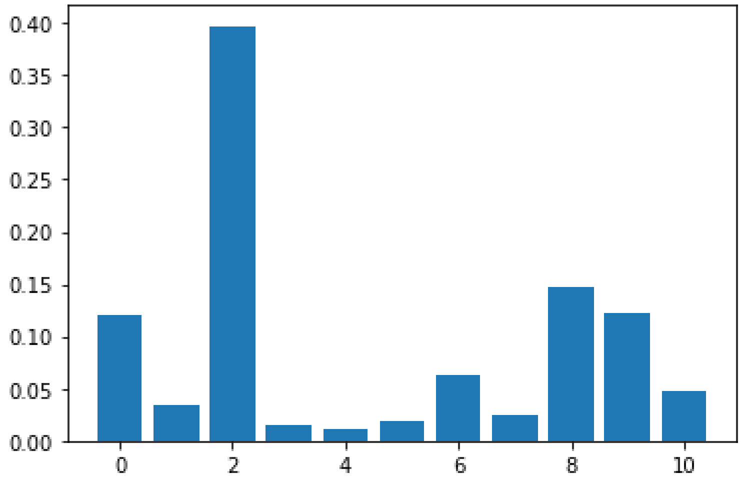
In [53]:

```
for i,v in enumerate(imp2):
        print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.11993
Feature: 1, Score: 0.03319
Feature: 2, Score: 0.39717
Feature: 3, Score: 0.01413
Feature: 4, Score: 0.01019
Feature: 5, Score: 0.01897
```

```
Feature: 6, Score: 0.06387
Feature: 7, Score: 0.02521
Feature: 8, Score: 0.14716
Feature: 9, Score: 0.12238
Feature: 10, Score: 0.04782
```

In [54]:
```
## Plotting Feature importance of Random Forest Classifier
plot.bar([x for x in range(len(imp2))], imp2)
plot.show()
```



In [55]:
```
## Performing Decision Tree Classifier with Random State = 20
from sklearn.tree import DecisionTreeClassifier
dt_model=DecisionTreeClassifier(random_state=20)
```

In [56]:
```
## Fitting the Decision Tree Classifier into our training dataframe
dt_model.fit(x_train,y_train)
```

Out[56]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=20)
```

In [57]:
```
## Predicting the score of our Decision Tree Classifier on Training DataFrame
dt_model.score(x_train,y_train)
```

Out[57]:
```
1.0
```

In [58]:
```
## Predicting the score of our Decision tree Classifier on Test DataFrame
dt_model.score(x_test,y_test)
```

Out[58]:
```
0.8732321933659039
```

In [59]:
```
dt_model.predict(x_train)
```

Out[59]:
```
array([0, 1, 0, ..., 0, 0, 0])
```

In [60]:
```
dt_model.predict_proba(x_test)
```

Out[60]:
```
array([[1., 0.],
       [0., 1.],
```

```
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

In [61]:

```
y_pred=dt_model.predict_proba(x_test)[:,1]
```

In [62]:

```
y_new=[]
for i in range(len(y_pred)):
    if y_pred[i]<=0.7:
        y_new.append(0)
    else:
        y_new.append(1)
```

In [63]:

```
accuracy_score(y_test,y_new)
```

Out[63]:

```
0.8732321933659039
```

In [64]:

```
train_accuracy=[]
test_accuracy=[]
for depth in range(1,10):
    dt_model=DecisionTreeClassifier(max_depth=depth,random_state=20)
    dt_model.fit(x_train,y_train)
    train_accuracy.append(dt_model.score(x_train,y_train))
    test_accuracy.append(dt_model.score(x_test,y_test))
```

In [65]:

```
frame=pd.DataFrame({'max_depth':range(1,10),'train_acc':train_accuracy,'test_
frame.head()
```

Out[65]:

	max_depth	train_acc	test_cc
0	1	0.793245	0.787863
1	2	0.803360	0.796349
2	3	0.805075	0.796092
3	4	0.825647	0.814605
4	5	0.838676	0.826434

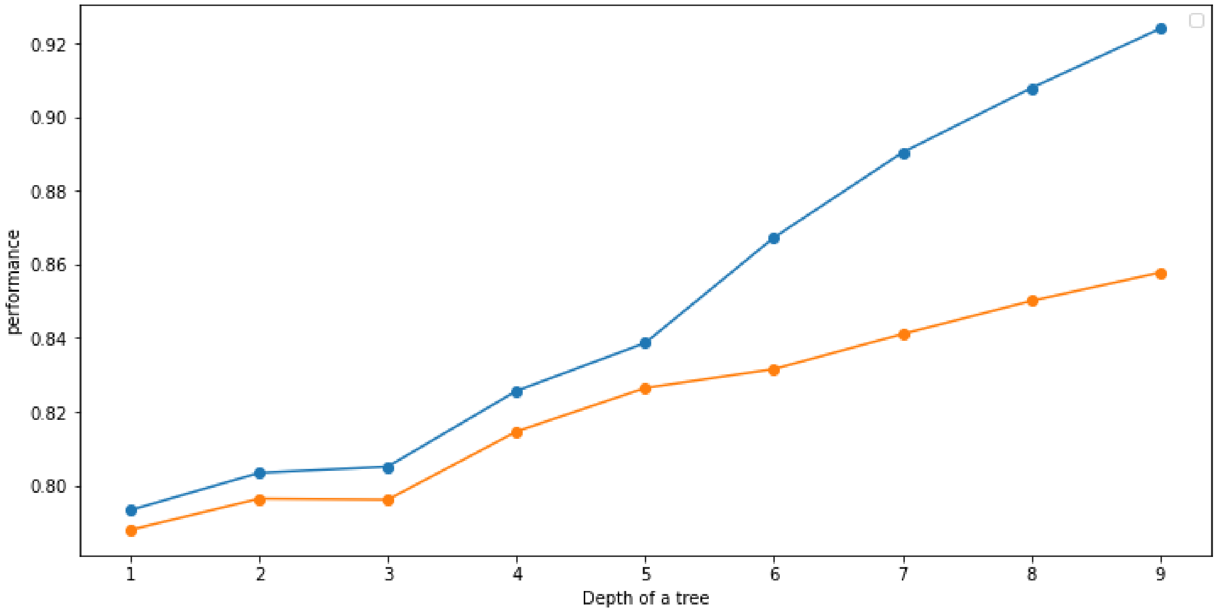
In [66]:

```
## Finding the Depth Of Tree to optimize the Decision Tree Classifier
plot.figure(figsize=(12,6))
plot.plot(frame['max_depth'],frame['train_acc'],marker='o')
plot.plot(frame['max_depth'],frame['test_cc'],marker='o')
plot.xlabel('Depth of a tree')
plot.ylabel('performance')
plot.legend()
```

```
No handles with labels found to put in legend.
```

Out[66]:

```
<matplotlib.legend.Legend at 0x7fe03cad6fd0>
```

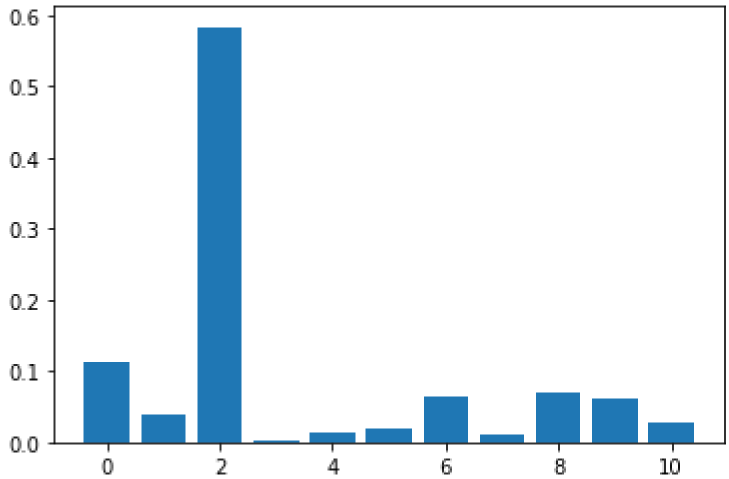


In [67]:
```
## Predicting Feature Importance of Decision Tree Classifier
imp3 = dt_model.feature_importances_
```

In [68]:
```
for i,v in enumerate(imp3):
        print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.11080
Feature: 1, Score: 0.03949
Feature: 2, Score: 0.58372
Feature: 3, Score: 0.00199
Feature: 4, Score: 0.01248
Feature: 5, Score: 0.01871
Feature: 6, Score: 0.06547
Feature: 7, Score: 0.00934
Feature: 8, Score: 0.06921
Feature: 9, Score: 0.06026
Feature: 10, Score: 0.02853
```

In [69]:
```
## Plotting Feature Importance of Decision Tree Classifier
plot.bar([x for x in range(len(imp3))], imp3)
plot.show()
```



3. Results

3.1 Exploratory Data Analysis(EDA) Results

‣ More Observations are seen between the age of 45 and 60.
‣ Maximum Number of people in our Dataset are non-smokers followed by Unknown smoking status, formerly smoked, Smokes Respectively.
‣ Observations with Age>60 have the highest possibility of having a stroke.
‣ 80% of Observations in the Data who have stroke have Average Glucose Levels over 115 which is over the threshold Glucose level.
‣ Number of Strokes in Male and Female are almost Identical.
‣ According to the stacked Bar Plot, observations having a private job are having more stroke cases then other types of jobs or those who haven't worked.
‣ Residence Type (Urban or Rural) of observations has no effect on Stroke.
‣ Number of Smokers and Non Smokers in theDataset has no Drastic Effect on our number of Stroke observation.
‣ Most of Patients having a stroke are in the Age between 60-80 and having a Higher Glucose Level.

3.2 Machine Learning Results

3.2.1 Logistic Regression Results

‣ Our training and testing Accuracy Score is 82.22% and 82.18% respectively is almost similar so there is no overfitting in this model.
‣ We Calculate the F1 Score with the formula which is close to 1 it means it has compared our 2 Classifiers (Stroke or no stroke) accurately.
‣ In Feature importance bar plot, age is the most Important Feature according to Logistic Regression Model.

3.2.2 Random Forest Classifier Results

‣ Testing Accuracy score is around 82% which means it has measured correctly the mean of the subsets.
‣ In the confusion matrix, We have the Highest true negative value which is a good sign for detection of a stroke in a healthcare dataset as we have predicted Actual Value and Prediction outcome are more accurate.
‣ In Feature importance bar plot, age,work_type, glucose Levels and bmi is the most Important feature according to Random Forest Classifier Model.

3.2.3 Decision Tree Classifier Results

‣ We have chosen the Hyper-parameter which is Random State = 20 that is why we will get different train and test sets with different integer values and we received an accuracy score of 87%.

▸ In the Decision Tree Classifier, we have found that the depth of the Tree will be 4.

▸In Feature importance bar plot, age is shown is the highest and most important feature according to Decision Tree Classifier.

4. Conclusion

As we can analyse from the Results, Random Forest Classifier algorithm is the best fitted model for this dataset, so we can say that patients having older age, High Glucose Level and High bmi values have the highest probability of getting a Stroke. The dataset was imbalanced and to balance the dataset we had to use the SMOTE technique which equally splits the predictor variable into 2 halves.As we have seen from this data, older-aged patients with high glucose levels and high bmi have high probability of getting a stroke. There is no plausible evidence that patients having a heart disease will have a high probability of having a Stroke.There is also some evidence that work lifestyle also has a high probability when it comes to stroke in a patient.

5. References

https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset
https://en.wikipedia.org/wiki/Machine_learning