Import libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as w
w.filterwarnings("ignore")
```

Import dataset

In [2]:
```
df=pd.read_csv("healthcare-dataset-stroke-data.csv")
df
```

Out[2]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_typ
0	9046	Male	67.0	0	1	Yes	Private	Urba
1	51676	Female	61.0	0	0	Yes	Self-employed	Rura
2	31112	Male	80.0	0	1	Yes	Private	Rura
3	60182	Female	49.0	0	0	Yes	Private	Urba
4	1665	Female	79.0	1	0	Yes	Self-employed	Rura
...	...	...	...	...	...	...	...	.
5105	18234	Female	80.0	1	0	Yes	Private	Urba
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urba
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rura
5108	37544	Male	51.0	0	0	Yes	Private	Rura
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urba

5110 rows × 12 columns

Summary of dataset

In [3]:
```
df.info()    # df.info() give us the summary of dataset
```

```
<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 [4]:
```
# In the above dataset there are 12 columns and 5110 entries
# In this dataset id, hypertension, heart_disease, stroke columns have integer
# And gender, ever_married, work_type, Residence_type, smoking_status have obj
# And rest of the columns have float datatype
# This dataset take 479.2 KB memory
```

Handling missing values (Null Values)

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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 [6]:
```
# Column bmi have 201 null values
```

In [7]:
```
(201/5110)*100     # Finding how many percent of null value present in column
```

Out[7]:
```
3.9334637964774952
```

In [8]:
```
# We are going to fill bmi null values with mean value as percent of null valu
```

In [9]:
```
bmimean=df["bmi"].mean()
df["bmi"].fillna(bmimean,inplace=True)
```

In [10]:
```
df.isnull().sum()     # df.isnull is boolean function which give us output
```

Out[10]:
```
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
```

Removing unwanted column

In [11]:
```
# In the above dataset id column is unwanted column
# As id column don't give us more or required information in the analysis or p
# So we are going to remove id column from this dataset
```

In [12]:
```
df.drop("id",inplace=True,axis=1)     # df.drop() is use to drop column as wel
```

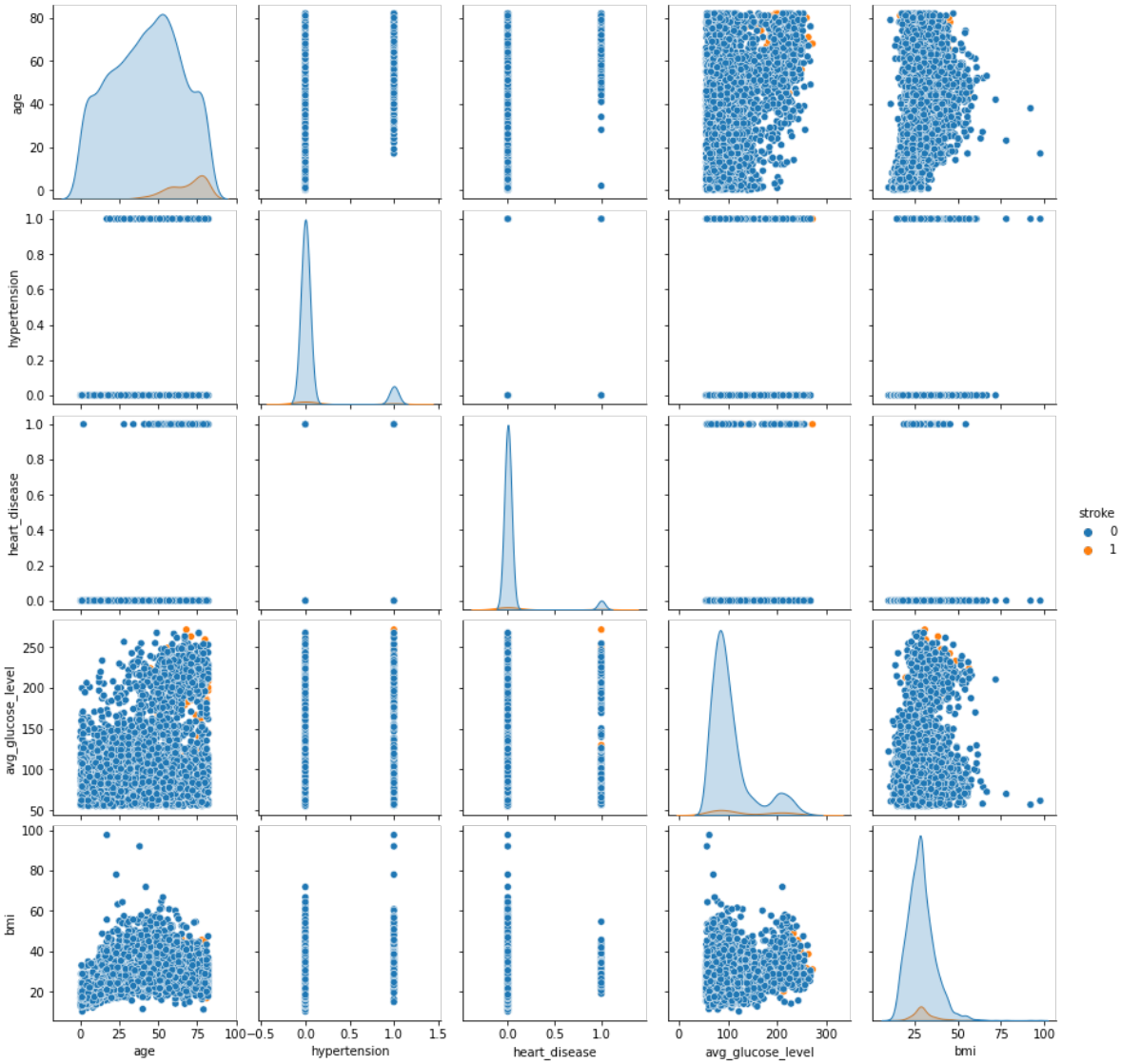
In [13]:
```
df.head()     # df.head() gives us top 5 records of dataset
```

Out[13]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	Male	67.0	0	1	Yes	Private	Urban	
1	Female	61.0	0	0	Yes	Self-employed	Rural	
2	Male	80.0	0	1	Yes	Private	Rural	
3	Female	49.0	0	0	Yes	Private	Urban	
4	Female	79.0	1	0	Yes	Self-employed	Rural	

Now let's plot some graph and try to find little bit more information about dataset

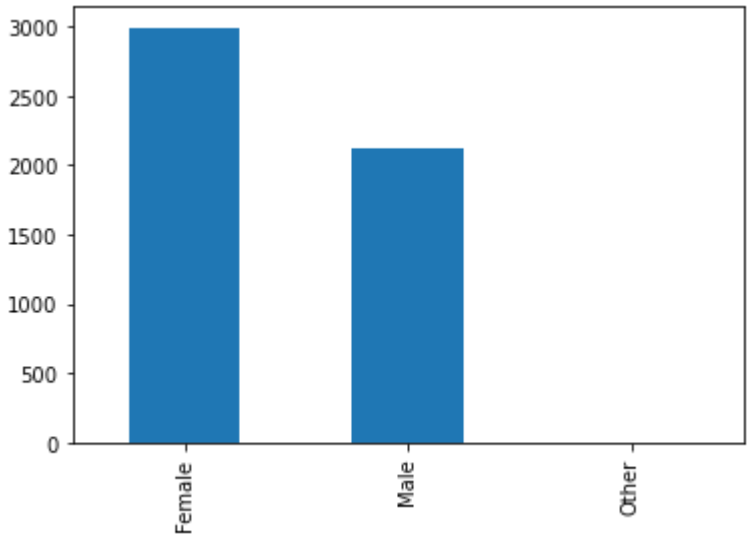
In [14]:
```
sns.pairplot(df,hue="stroke")     #is used to create a matrix of scatter plots
plt.show()     # use to display the plot
```



In [15]:
```
df["gender"].value_counts()
```

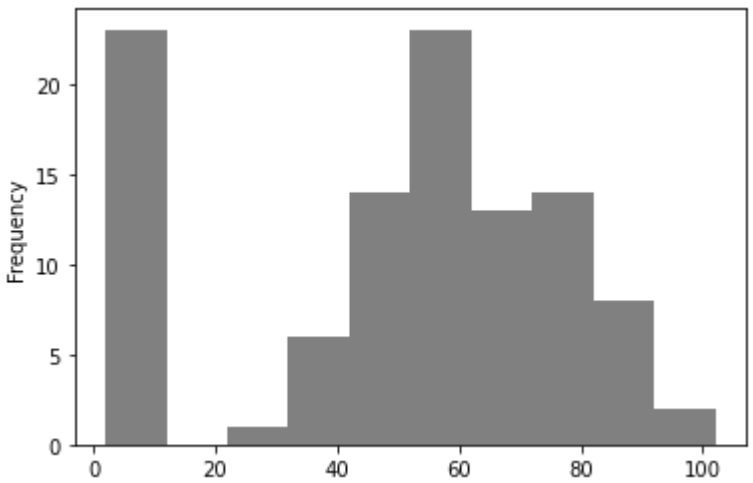
Out[15]:
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [16]:
```
df.gender.value_counts().plot(kind="bar")
plt.show()
```



In [17]:
```
# In the gender column there are 3 categories which is Female, Male and Others
# In the gender column Females are 2994, Males are 2115,Other is 1
```

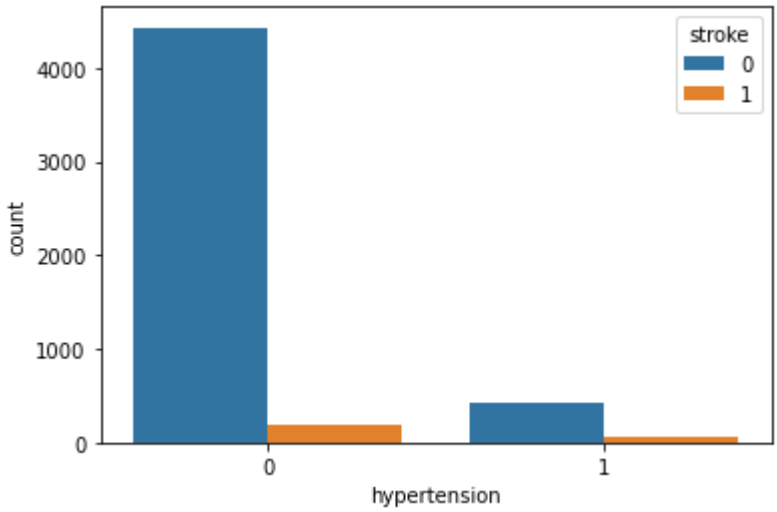
In [18]:
```
df["age"].value_counts().plot(kind="hist",color="gray")
plt.show()
```



In [19]:
```
df["hypertension"].value_counts()
```

Out[19]:
```
0    4612
1     498
Name: hypertension, dtype: int64
```

In [20]:
```
sns.countplot(data=df,x="hypertension",hue="stroke")
plt.show()
```

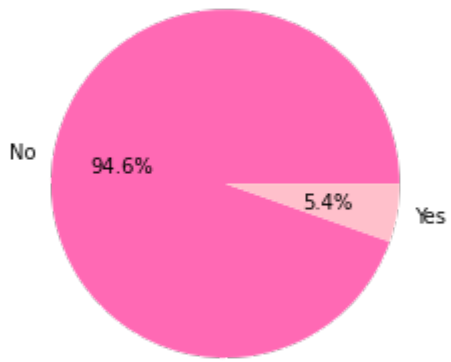


In [21]:
```
# From the above graph it shows that 4162 people don't have hypertendion while
```

In [22]:
```
df["heart_disease"].value_counts()
```

Out[22]:
```
0    4834
1     276
Name: heart_disease, dtype: int64
```

In [23]:
```
plt.pie(df["heart_disease"].value_counts(),labels=["No","Yes"],colors=["hotpin
plt.show()
```

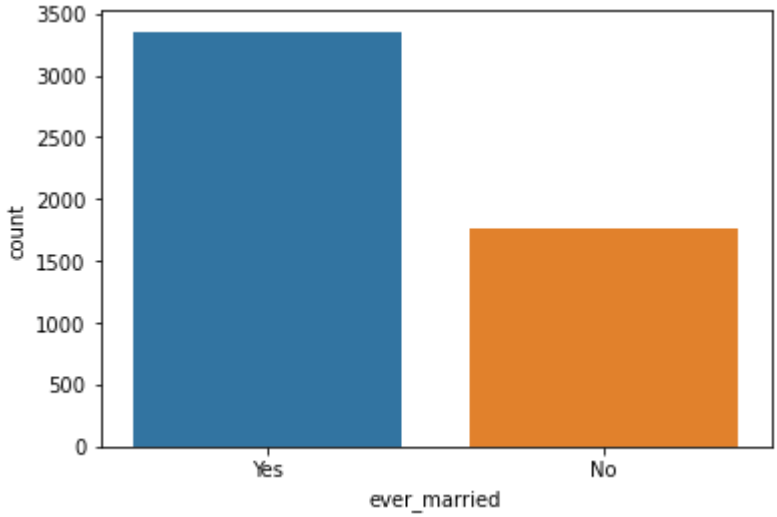


In [24]:
```
# From above graph it shows that 5.4 percent of people have heart dieses and r
```

In [25]:
```
df["ever_married"].value_counts()
```

Out[25]:
```
Yes    3353
No     1757
Name: ever_married, dtype: int64
```

In [26]:
```
sns.countplot(data=df,x="ever_married")
plt.show()
```

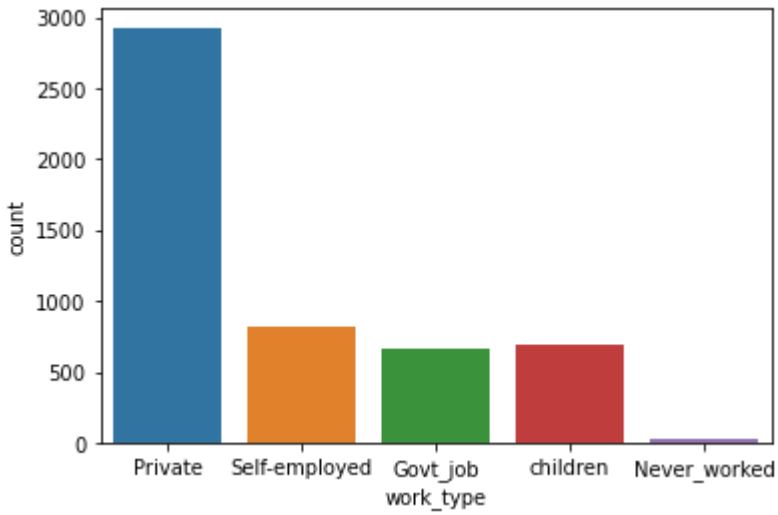


In [27]:
```
# From the above graph it shows that 3353 people married and 1757 people not m
```

In [28]:
```
df["work_type"].value_counts()
```

Out[28]:
```
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

In [29]:
```
sns.countplot(data=df,x="work_type")
plt.show()
```

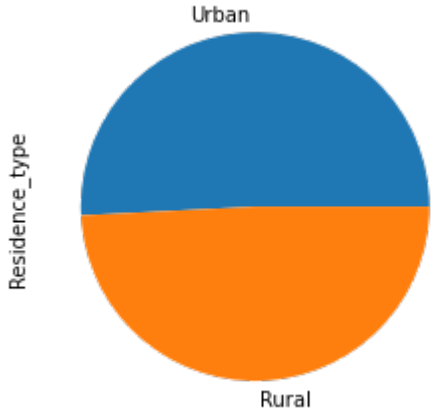


In [30]:
```
# from the above graph it shows that 2925 people are work at private company,
# 687 people are children and 657 people are work at goverment job and 22 peop
```

In [31]:
```
df["Residence_type"].value_counts()
```

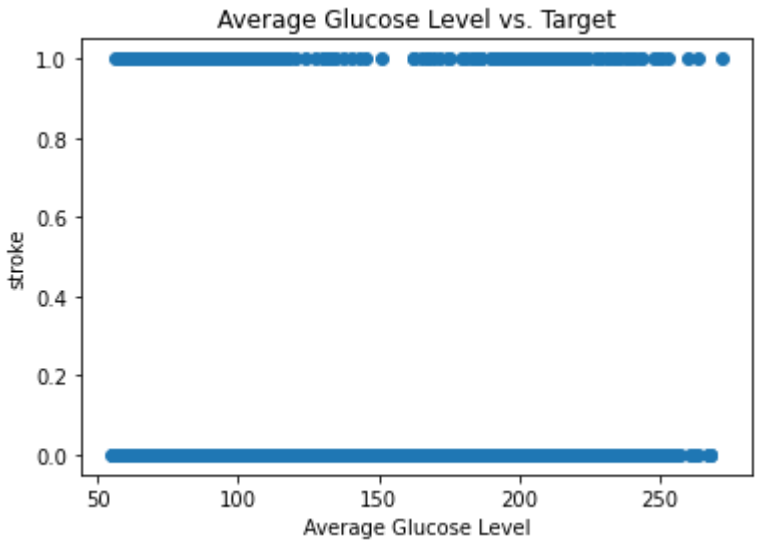
Out[31]:
```
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
```

In [32]:
```
df["Residence_type"].value_counts().plot(kind="pie")
plt.show()
```

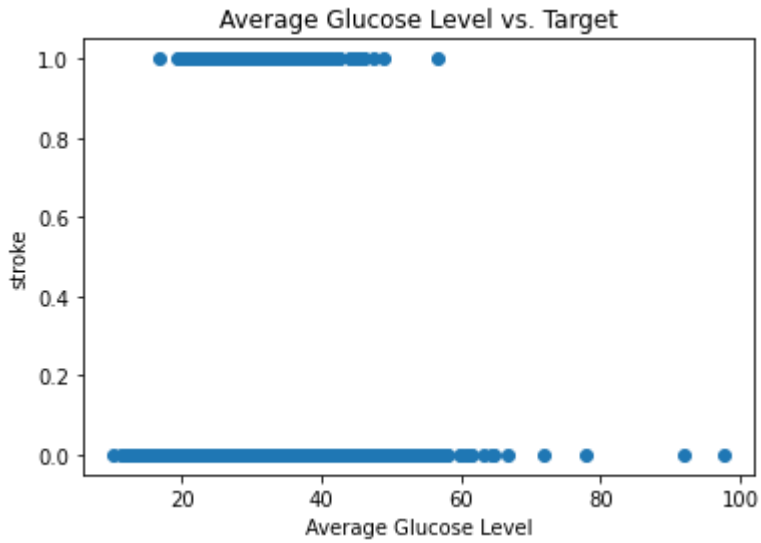


In [33]:
```
# From the above dataset it shows that almost 50% of people live at rural area
```

In [34]:
```
x = df['avg_glucose_level']
y = df['stroke']

# Create a scatter plot
plt.scatter(x, y)

# Set the title and axis labels
plt.title('Average Glucose Level vs. Target')
plt.xlabel('Average Glucose Level')
plt.ylabel('stroke')

# Display the plot
plt.show()
```

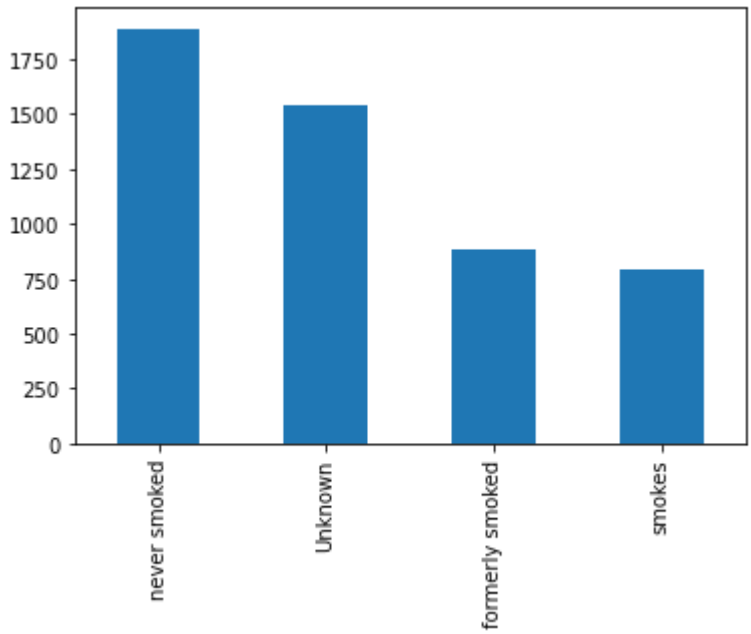


In [35]:
```
x = df['bmi']
y = df['stroke']

# Create a scatter plot
plt.scatter(x, y)

# Set the title and axis labels
plt.title('Average Glucose Level vs. Target')
plt.xlabel('Average Glucose Level')
plt.ylabel('stroke')

# Display the plot
plt.show()
```



In [36]:
```
df["smoking_status"].value_counts().plot(kind="bar")
```

Out[36]: <AxesSubplot:>



In [37]:
```
df["smoking_status"].value_counts()
```

Out[37]:
```
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

In [38]:
```
# From the above it shows that 1892 people never smoked and 1544 people are un
# 789 people smokes
```

Describe dataset

In [39]:
```
df.describe()
```

Out[39]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000
mean	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	22.612647	0.296607	0.226063	45.283560	7.698018	0.215320
min	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	25.000000	0.000000	0.000000	77.245000	23.800000	0.000000
50%	45.000000	0.000000	0.000000	91.885000	28.400000	0.000000
75%	61.000000	0.000000	0.000000	114.090000	32.800000	0.000000
max	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

In [40]:
```
# In the above data minimum age is 0.08 years & maximum age is 82 years and av
# maximum glucose level is 271.74 and minimum glucose level is 55.12 and avera
# minimum bmi is 10.30 and maximum bmi is 97.60 and average bmi is 28.89
```

Encode categorical column into numerical column

In [41]:
```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()

catcol=df.select_dtypes(object).columns
print(catcol)

df[catcol]=oe.fit_transform(df[catcol])
```

```
Index(['gender', 'ever_married', 'work_type', 'Residence_type',
       'smoking_status'],
      dtype='object')
```

In [42]:
```
df
```

Out[42]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
0	1.0	67.0	0	1	1.0	2.0	1.0	
1	0.0	61.0	0	0	1.0	3.0	0.0	
2	1.0	80.0	0	1	1.0	2.0	0.0	
3	0.0	49.0	0	0	1.0	2.0	1.0	
4	0.0	79.0	1	0	1.0	3.0	0.0	
...	...	...	...	...	...	...	...	
5105	0.0	80.0	1	0	1.0	2.0	1.0	
5106	0.0	81.0	0	0	1.0	3.0	1.0	
5107	0.0	35.0	0	0	1.0	3.0	0.0	
5108	1.0	51.0	0	0	1.0	2.0	0.0	
5109	0.0	44.0	0	0	1.0	0.0	1.0	

5110 rows × 11 columns

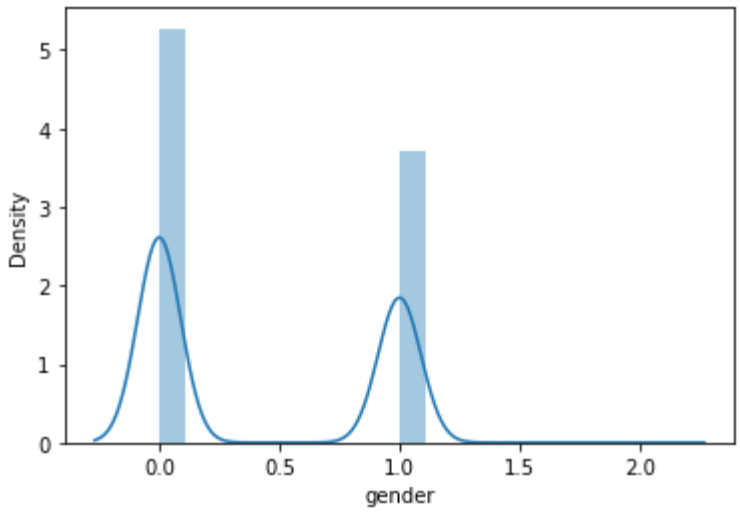
Skewness removing

In [43]:
```
from scipy.stats import skew

colname=df.select_dtypes(["int","float"]).columns
print(colname)
```

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

In [44]:
```
for col in df[colname]:
    print(col)
    print(skew(df[col]))

    plt.figure()
    sns.distplot(df[col])
    plt.show()
```

```
gender
0.35290826168415185
```




```
age
```

In [45]:
```
df.corr().style.background_gradient()
```

Out[45]:

	gender	age	hypertension	heart_disease	ever_married	work_type	R
gender	1.000000	-0.028202	0.020994	0.085447	-0.031005	0.056422	
age	-0.028202	1.000000	0.276398	0.263796	0.679125	-0.361642	
hypertension	0.020994	0.276398	1.000000	0.108306	0.164243	-0.051761	
heart_disease	0.085447	0.263796	0.108306	1.000000	0.114644	-0.028023	
ever_married	-0.031005	0.679125	0.164243	0.114644	1.000000	-0.352722	
work_type	0.056422	-0.361642	-0.051761	-0.028023	-0.352722	1.000000	
Residence_type	-0.006738	0.014180	-0.007913	0.003092	0.006261	-0.007316	
avg_glucose_level	0.055180	0.238171	0.174474	0.161857	0.155068	-0.050513	
bmi	-0.026109	0.325942	0.160189	0.038899	0.335705	-0.299448	
smoking_status	-0.062581	0.265199	0.111038	0.048460	0.259647	-0.305927	
stroke	0.008929	0.245257	0.127904	0.134914	0.108340	-0.032316	

In [46]:
```
# in the case of stroke dataset when i found out that co-relation is low and s
# in the hypertension and avg_glocuse_level and bmi column but i think it is n
# beacuse it show continuous value and on the top of the hand it gives us the
# and people may differ person to person in glocuse level and bmi and hyperten
```

Split data into X and Y

In [47]:
```
x=df.iloc[:,:-1]
x
```

Out[47]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
0	1.0	67.0	0	1	1.0	2.0	1.0	
1	0.0	61.0	0	0	1.0	3.0	0.0	
2	1.0	80.0	0	1	1.0	2.0	0.0	
3	0.0	49.0	0	0	1.0	2.0	1.0	
4	0.0	79.0	1	0	1.0	3.0	0.0	
...	...	...	...	...	...	...	...	
5105	0.0	80.0	1	0	1.0	2.0	1.0	
5106	0.0	81.0	0	0	1.0	3.0	1.0	
5107	0.0	35.0	0	0	1.0	3.0	0.0	
5108	1.0	51.0	0	0	1.0	2.0	0.0	
5109	0.0	44.0	0	0	1.0	0.0	1.0	

5110 rows × 10 columns

In [48]:
```
y=df.iloc[:,-1]
y
```

Out[48]:
```
0       1
1       1
2       1
3       1
4       1
       ..
5105    0
5106    0
5107    0
5108    0
5109    0
Name: stroke, Length: 5110, dtype: int64
```

Split data into training data and testing data

In [49]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=0,st
```

Apply different algorithm

In [50]:
```
def mymodel(model):

    #model creation
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    #cheaking bias and variance
    train=model.score(xtrain,ytrain)
    test=model.score(xtest,ytest)

    print(f"Training Accuracy= {train}")
    print(f"Testing Accuracy= {test}")

    #model evaluation

    print(classification_report(ytest,ypred))
    return model
```

Import algorithm which we need

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier


from sklearn.metrics import classification_report
```

In [52]:
```
knn=mymodel(KNeighborsClassifier())
```

```
Training Accuracy= 0.9516354487000279
Testing Accuracy= 0.9471624266144814
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.12      0.01      0.02        75

    accuracy                           0.95      1533
   macro avg       0.54      0.50      0.50      1533
weighted avg       0.91      0.95      0.93      1533
```

In [53]:
```
logreg=mymodel(LogisticRegression())
```

```
Training Accuracy= 0.9513558848196813
Testing Accuracy= 0.9510763209393346
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.50      0.01      0.03        75

    accuracy                           0.95      1533
   macro avg       0.73      0.51      0.50      1533
weighted avg       0.93      0.95      0.93      1533
```

In [54]:
```
svm=mymodel(SVC())
```

```
Training Accuracy= 0.9513558848196813
Testing Accuracy= 0.9510763209393346
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.00      0.00      0.00        75

    accuracy                           0.95      1533
   macro avg       0.48      0.50      0.49      1533
weighted avg       0.90      0.95      0.93      1533
```

In [55]:
```
dt=mymodel(DecisionTreeClassifier())
```

```
Training Accuracy= 1.0
Testing Accuracy= 0.9106327462491846
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      1458
           1       0.11      0.12      0.12        75

    accuracy                           0.91      1533
   macro avg       0.53      0.54      0.53      1533
weighted avg       0.91      0.91      0.91      1533
```

In [56]:
```
# After applying different classification model we get training and testing ac
# In case of KNN regression Training Accuracy= 0.9516354487000279 and Testing
# In case of logistic regession Training Accuracy= 0.9513558848196813 and Test
# In case of support vector machine Training Accuracy= 0.9513558848196813 and
# In case if Decision Tree Training Accuracy= 1.0 and Testing Accuracy= 0.9145
```

In [57]:
```
# From the above observation it shown that support vector machine and logistic
# Perfect training and testing accuracy so that i'll go for support vector mac
```

In [58]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

In [59]:
```
pipe=Pipeline(
    steps=[
        ("scaler",StandardScaler()),
        ("svm",SVC())
    ]
)
```

In [60]:
```
pipe.fit(xtrain,ytrain)
ypred=pipe.predict(xtest)
print(classification_report(ytest,ypred))

train = pipe.score(xtrain,ytrain)
test = pipe.score(xtest,ytest)

print(f"Training Accuracy:- {train}\n Testing Accuracy:- {test}")
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.00      0.00      0.00        75

    accuracy                           0.95      1533
   macro avg       0.48      0.50      0.49      1533
weighted avg       0.90      0.95      0.93      1533

Training Accuracy:- 0.9519150125803746
 Testing Accuracy:- 0.9510763209393346
```

In [61]:
```
from sklearn.model_selection import GridSearchCV
```

In [62]:
```
parameter = {
             "C":[0.1,1,10],
             "gamma":[0.1,1,10],
             "kernel":["rbf"]

}
```

In [63]:
```
grid = GridSearchCV(SVC(), parameter, verbose=2)
grid.fit(xtrain,ytrain)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
1.3s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
1.3s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
1.2s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
1.2s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
1.2s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
2.1s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
2.1s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
2.1s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
2.1s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
2.2s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
2.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
2.1s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
2.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
2.1s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
2.1s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
1.4s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
1.4s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
1.4s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
1.5s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
1.4s
[CV] END .........................C=1, gamma=1, kernel=rbf; total time=
2.1s
[CV] END .........................C=1, gamma=1, kernel=rbf; total time=
2.0s
[CV] END .........................C=1, gamma=1, kernel=rbf; total time=
2.1s
[CV] END .........................C=1, gamma=1, kernel=rbf; total time=
2.8s
[CV] END .........................C=1, gamma=1, kernel=rbf; total time=
2.8s
[CV] END ......................C=1, gamma=10, kernel=rbf; total time=
2.4s
```

```
[CV] END .....................C=1, gamma=10, kernel=rbf; total time=
2.5s
[CV] END .....................C=1, gamma=10, kernel=rbf; total time=
2.1s
[CV] END .....................C=1, gamma=10, kernel=rbf; total time=
2.2s
[CV] END .....................C=1, gamma=10, kernel=rbf; total time=
2.2s
[CV] END ....................C=10, gamma=0.1, kernel=rbf; total time=
1.5s
[CV] END ....................C=10, gamma=0.1, kernel=rbf; total time=
1.1s
[CV] END ....................C=10, gamma=0.1, kernel=rbf; total time=
1.5s
[CV] END ....................C=10, gamma=0.1, kernel=rbf; total time=
1.5s
[CV] END ....................C=10, gamma=0.1, kernel=rbf; total time=
1.5s
[CV] END .....................C=10, gamma=1, kernel=rbf; total time=
2.2s
[CV] END .....................C=10, gamma=1, kernel=rbf; total time=
2.0s
[CV] END .....................C=10, gamma=1, kernel=rbf; total time=
2.4s
[CV] END .....................C=10, gamma=1, kernel=rbf; total time=
2.3s
[CV] END .....................C=10, gamma=1, kernel=rbf; total time=
2.4s
[CV] END ....................C=10, gamma=10, kernel=rbf; total time=
2.4s
[CV] END ....................C=10, gamma=10, kernel=rbf; total time=
2.5s
[CV] END ....................C=10, gamma=10, kernel=rbf; total time=
2.3s
[CV] END ....................C=10, gamma=10, kernel=rbf; total time=
2.8s
[CV] END ....................C=10, gamma=10, kernel=rbf; total time=
2.8s
```

Out[63]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10],
                         'kernel': ['rbf']},
             verbose=2)
```

In [64]:
```
grid.best_params_
```

Out[64]:
```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

In [65]:
```
grid.best_score_
```

Out[65]:
```
0.9516357385631128
```

In [66]:
```
grid.best_estimator_
```

Out[66]:
```
SVC(C=1, gamma=0.1)
```

In [67]:
```
svm = grid.best_estimator_
svm.fit(xtrain,ytrain)
ypred = svm.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.00      0.00      0.00        75

    accuracy                           0.95      1533
   macro avg       0.48      0.50      0.49      1533
weighted avg       0.90      0.95      0.93      1533
```

Cross validation

In [68]:
```
from sklearn.model_selection import cross_val_score

# Create an SVM classifier with default hyperparameters
svm = SVC()

# Perform 10-fold cross-validation on the SVM model
scores = cross_val_score(svm, x, y, cv=10)

# Print the mean and standard deviation of the cross-validation scores
print("Cross-validation accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.
```

```
Cross-validation accuracy: 0.95 (+/- 0.00)
```

Ensemble learning

In [69]:
```
from sklearn.ensemble import BaggingClassifier

# Create a bagging classifier with 10 SVM base estimators
bagging = BaggingClassifier(base_estimator=svm, n_estimators=10)

# Train the bagging classifier on the training data
bagging.fit(xtrain, ytrain)

# Evaluate the bagging classifier on the testing data
score = bagging.score(xtest, ytest)
print(f"Bagging accuracy:  {score}")
```

```
Bagging accuracy:  0.9510763209393346
```

In [70]:
```
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
```

In [71]:
```
mymodel(AdaBoostClassifier())
```

```
Training Accuracy= 0.9513558848196813
Testing Accuracy= 0.9504240052185258
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.40      0.03      0.05        75

    accuracy                           0.95      1533
   macro avg       0.68      0.51      0.51      1533
weighted avg       0.93      0.95      0.93      1533
```

Out[71]:
```
AdaBoostClassifier()
```

In [72]:
```
mymodel(GradientBoostingClassifier())
```

```
Training Accuracy= 0.9611406206318144
Testing Accuracy= 0.9497716894977168
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1458
           1       0.25      0.01      0.03        75

    accuracy                           0.95      1533
   macro avg       0.60      0.51      0.50      1533
weighted avg       0.92      0.95      0.93      1533
```

Out[72]:
```
GradientBoostingClassifier()
```

In [73]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\omkar\anaconda3\lib\site-p
ackages (1.7.4)
Requirement already satisfied: numpy in c:\users\omkar\anaconda3\lib\site-pac
kages (from xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\omkar\anaconda3\lib\site-pac
kages (from xgboost) (1.7.3)
```

In [74]:
```
from xgboost import XGBClassifier
```

In [75]:
```
mymodel(XGBClassifier())
```

```
Training Accuracy= 0.9980430528375733
Testing Accuracy= 0.9399869536855838
              precision    recall  f1-score   support

           0       0.95      0.98      0.97      1458
           1       0.23      0.09      0.13        75

    accuracy                           0.94      1533
   macro avg       0.59      0.54      0.55      1533
weighted avg       0.92      0.94      0.93      1533
```

Out[75]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=Non
e,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

Hyper parameter tuning

In [76]:
```
# Train the final model on the entire dataset
final_model = SVC(C=1.0, kernel='rbf')
final_model.fit(x, y)
```

Out[76]:
```
SVC()
```

Predict new observation

In [81]:
```
from sklearn.svm import SVC
svm=SVC()
svm.fit(xtrain,ytrain)
svm.predict(xtrain)
```

Out[81]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [82]:
```
def new_observation():
    gender=input("Enter gender(Male/Female/Other)=")
    age=float(input("Enter age="))
    hypertension=int(input("Do you have hypertension (Yes=1/No=0)="))
    heart_disease=int(input("Do you have heart disease (Yes=1/No=0)="))
    ever_married=input("Are you married (Yes/No)=")
    work_type=input("Enter work type(Private/Self_employed/children/Govt_job/N
    residence_type=input("Enter residence type (Urban/Rural)=")
    avg_glucose_level=float(input("Enter average glucose level="))
    bmi=float(input("Enter BMI="))
    smoking_status=input("Enter smoking status (formerly smoked/never smoked/u

    newob=[gender,age,hypertension,heart_disease,ever_married,work_type,reside
    newob[0],newob[4],newob[5],newob[6],newob[-1]=oe.transform([[newob[0],newo

    y=svm.predict([newob])[0]

    if y==1:
        print("person have stroke")
    else:
        print("person don't have stroke")

    return y
```

In [83]:
```
new_observation()
```

```
Enter gender(Male/Female/Other)=Female
Enter age=34
Do you have hypertension (Yes=1/No=0)=0
Do you have heart disease (Yes=1/No=0)=0
Are you married (Yes/No)=Yes
Enter work type(Private/Self_employed/children/Govt_job/Never_worked)=Never_w
orked
Enter residence type (Urban/Rural)=Urban
Enter average glucose level=34
Enter BMI=23
Enter smoking status (formerly smoked/never smoked/unknown/smokes)=smokes
person don't have stroke
```

Out[83]: 0

In [ ]: