In [1]:
```
pip install -U jupyter
```

```
ernel->jupyter) (0.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lenovo\appd
ata\local\programs\python\python310\lib\site-packages (from jupyter-client>=
6.1.12->ipykernel->jupyter) (2.8.2)
Requirement already satisfied: pywin32>=1.0 in c:\users\lenovo\appdata\local
\programs\python\python310\lib\site-packages (from jupyter-core>=4.7->nbconve
rt->jupyter) (304)
Requirement already satisfied: jsonschema>=2.6 in c:\users\lenovo\appdata\loc
al\programs\python\python310\lib\site-packages (from nbformat>=5.1->nbconvert
->jupyter) (4.16.0)
Requirement already satisfied: fastjsonschema in c:\users\lenovo\appdata\loca
l\programs\python\python310\lib\site-packages (from nbformat>=5.1->nbconvert-
>jupyter) (2.16.2)
Requirement already satisfied: wcwidth in c:\users\lenovo\appdata\local\progr
ams\python\python310\lib\site-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<
3.1.0,>=2.0.0->jupyter-console->jupyter) (0.2.5)
Requirement already satisfied: pywinpty>=1.1.0 in c:\users\lenovo\appdata\loc
al\programs\python\python310\lib\site-packages (from terminado>=0.8.3->notebo
ok->jupyter) (2.0.8)
Requirement already satisfied: argon2-cffi-bindings in c:\users\lenovo\appdat
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\lenovo\appdata\local\programs\py
thon\python310\lib\site-packages (22.3.1)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\lenovo\appdata\local\programs
\python\python310\lib\site-packages (1.24.1)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\lenovo\appdata\local\programs
\python\python310\lib\site-packages (1.5.2)
Requirement already satisfied: numpy>=1.21.0 in c:\users\lenovo\appdata\local\p
rograms\python\python310\lib\site-packages (from pandas) (1.24.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\lenovo\appdat
a\local\programs\python\python310\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\lenovo\appdata\local\pr
ograms\python\python310\lib\site-packages (from pandas) (2022.7)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\appdata\local\progra
ms\python\python310\lib\site-packages (from python-dateutil>=2.8.1->pandas) (1.
16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [6]:
```
pip install scipy
```

```
Requirement already satisfied: scipy in c:\users\lenovo\appdata\local\programs
\python\python310\lib\site-packages (1.10.0)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in c:\users\lenovo\appdata
\local\programs\python\python310\lib\site-packages (from scipy) (1.24.1)
Note: you may need to restart the kernel to use updated packages.
```

In [7]:
```
pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\lenovo\appdata\local\program
s\python\python310\lib\site-packages (0.0.post2)
Note: you may need to restart the kernel to use updated packages.
```

In [8]:
```
data = pd.read_csv('healthcare-dataset-stroke-data.csv')
data
```

Out[8]:

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

5110 rows × 12 columns

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

Out[9]:

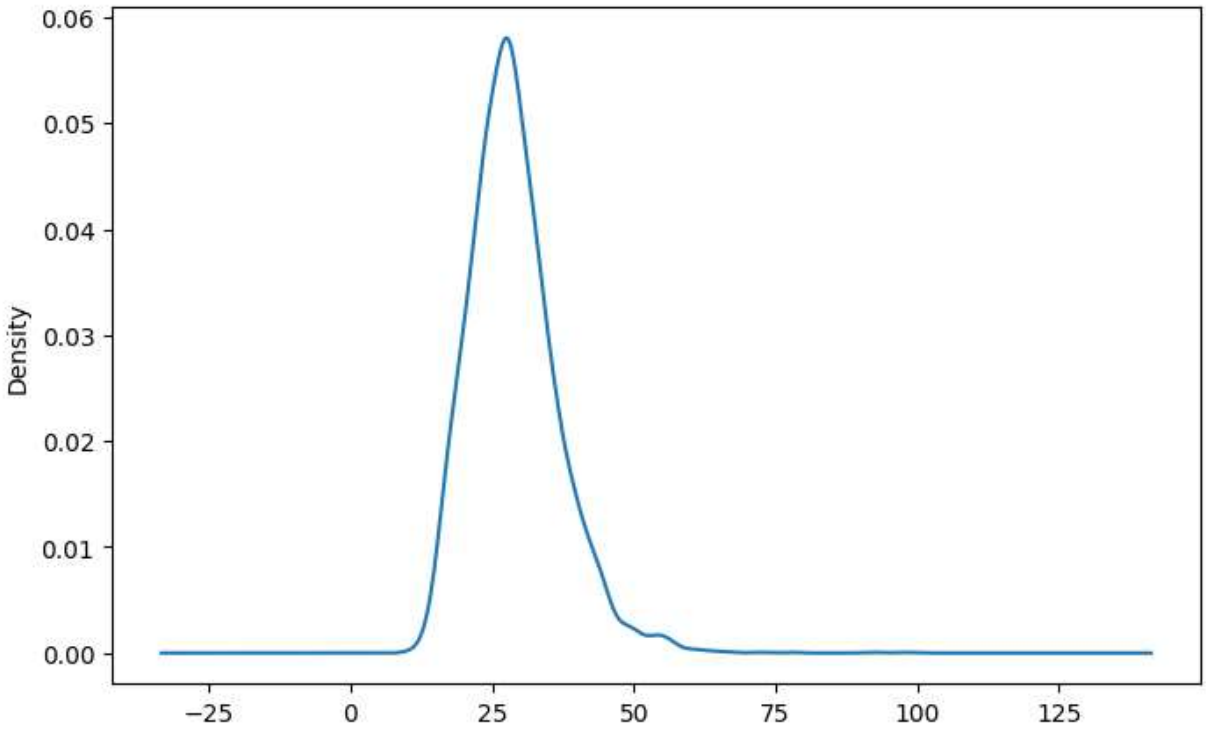
	id	age	hypertension	heart_disease	avg_glucose_level	bmi	
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	51
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	

In [10]:
```
data.isnull().sum()
```

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                201
smoking_status       0
stroke               0
dtype: int64
```

In [11]:
```
# Checking the distribution of the missing data column.

plt.figure(figsize=(8,5))
data['bmi'].plot(kind='kde')
plt.show()
```



Checking the distribution of the missing data column i.e bmi

Missing Value Treatment

In [12]:
```
data['bmi'].fillna(data['bmi'].mean(),inplace=True)
```

In [13]:
```
# re-checking missing value

data.isnull().sum()
```

Out[13]:
```
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
```

Droping unnecessary columns

In [14]:
```
data.drop(['id'],axis = 1, inplace=True)
```

In [15]:
```
data.head()
```

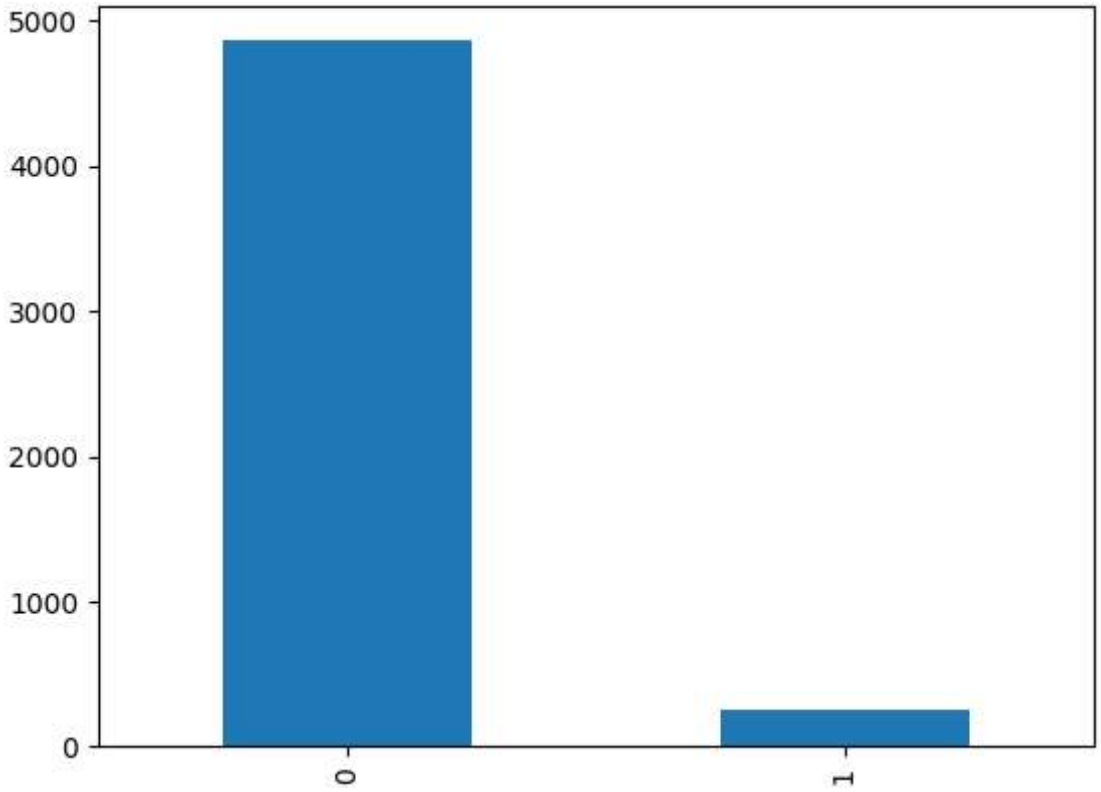
Out[15]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gluco
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	

EDA

Target Variable(Stroke)

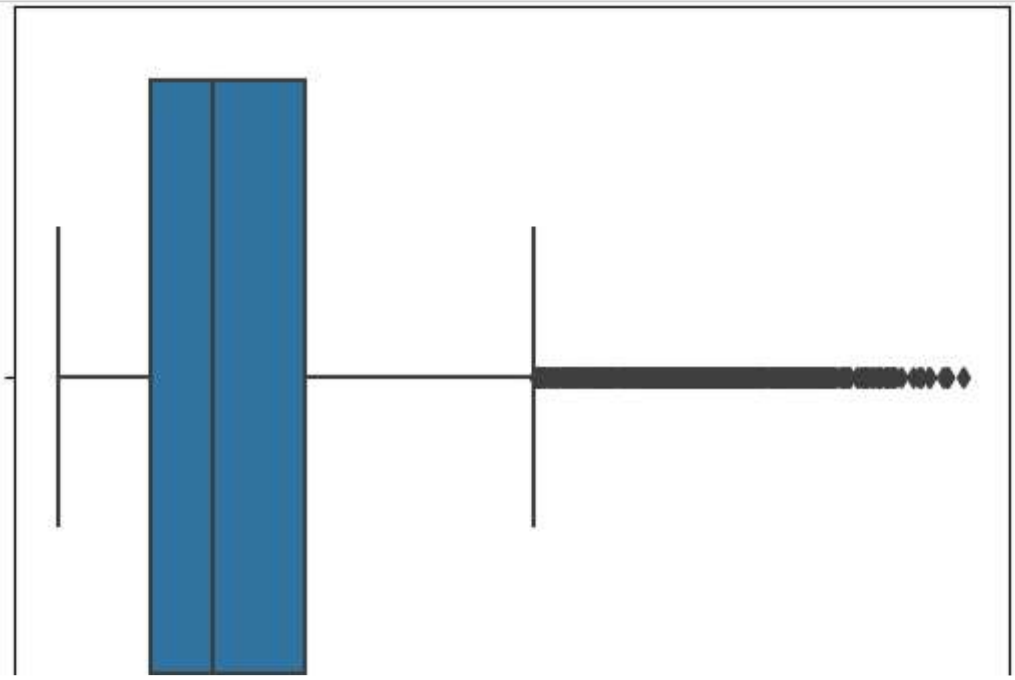
In [16]:
```
data['stroke'].value_counts().plot(kind='bar')
plt.show()
```



Checking outliers in our dataset(Categorical columns)

In [17]:
```
num=data.select_dtypes(exclude='object')
```

In [18]:
```
for i in num.columns:
    sns.boxplot(data=num,x=i)
    plt.show()
```

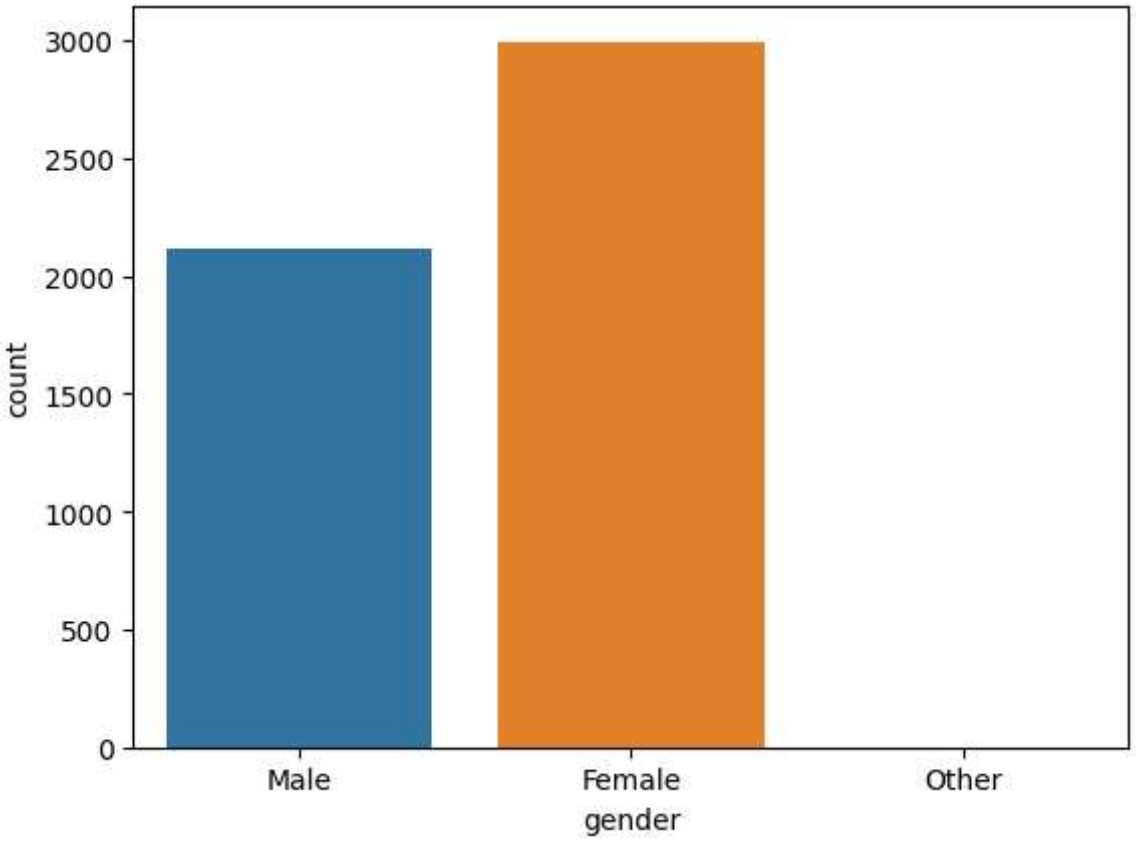


Gender

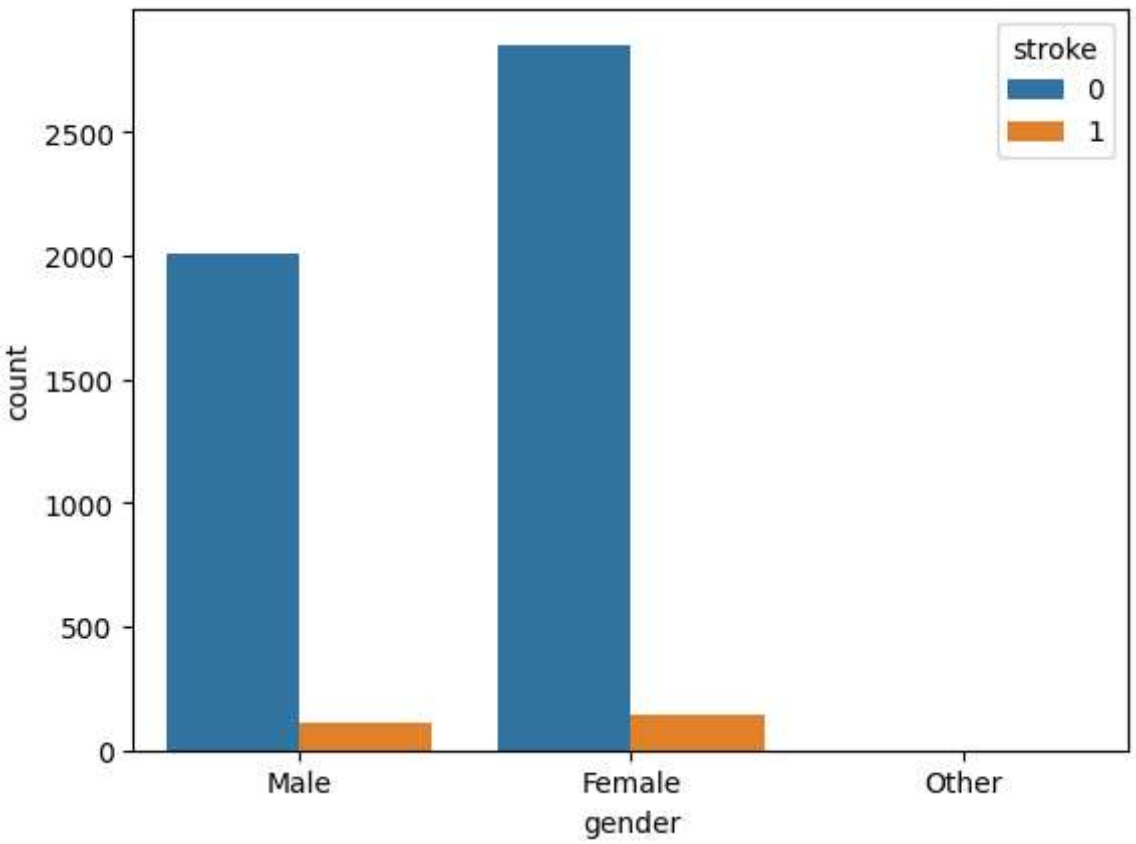
In [19]:
```
data['gender'].value_counts()
```

Out[19]:
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

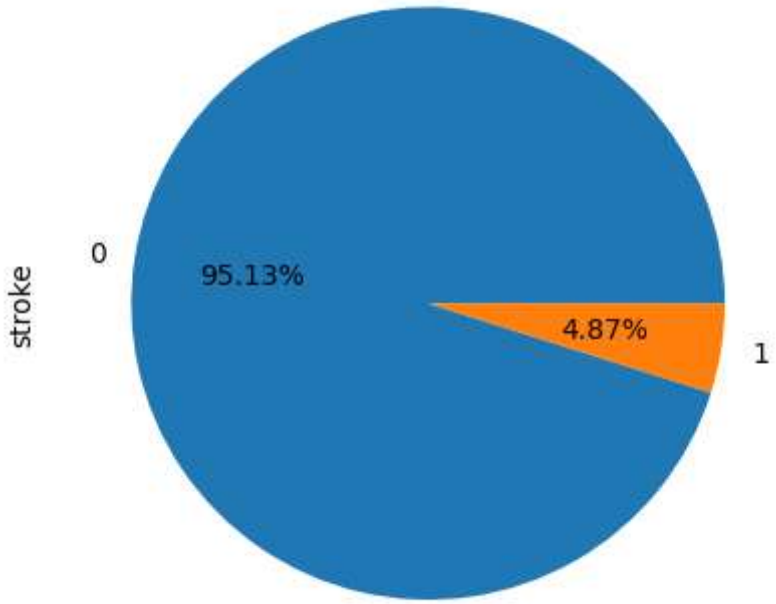
In [20]:
```
sns.countplot(data=data,x='gender')
plt.show()
```



In [21]:
```
sns.countplot(data=data,x='gender',hue='stroke')
plt.show()
```



In [22]:
```
data['stroke'].value_counts().plot(kind="pie",autopct='%0.2f%%')
plt.show()
```



Age

In [23]:
```
# more men than women had strokes.

data.groupby('gender').mean()[['age','stroke']]
```

Out[23]:

gender	age	stroke
Female	43.757395	0.047094
Male	42.483385	0.051064
Other	26.000000	0.000000

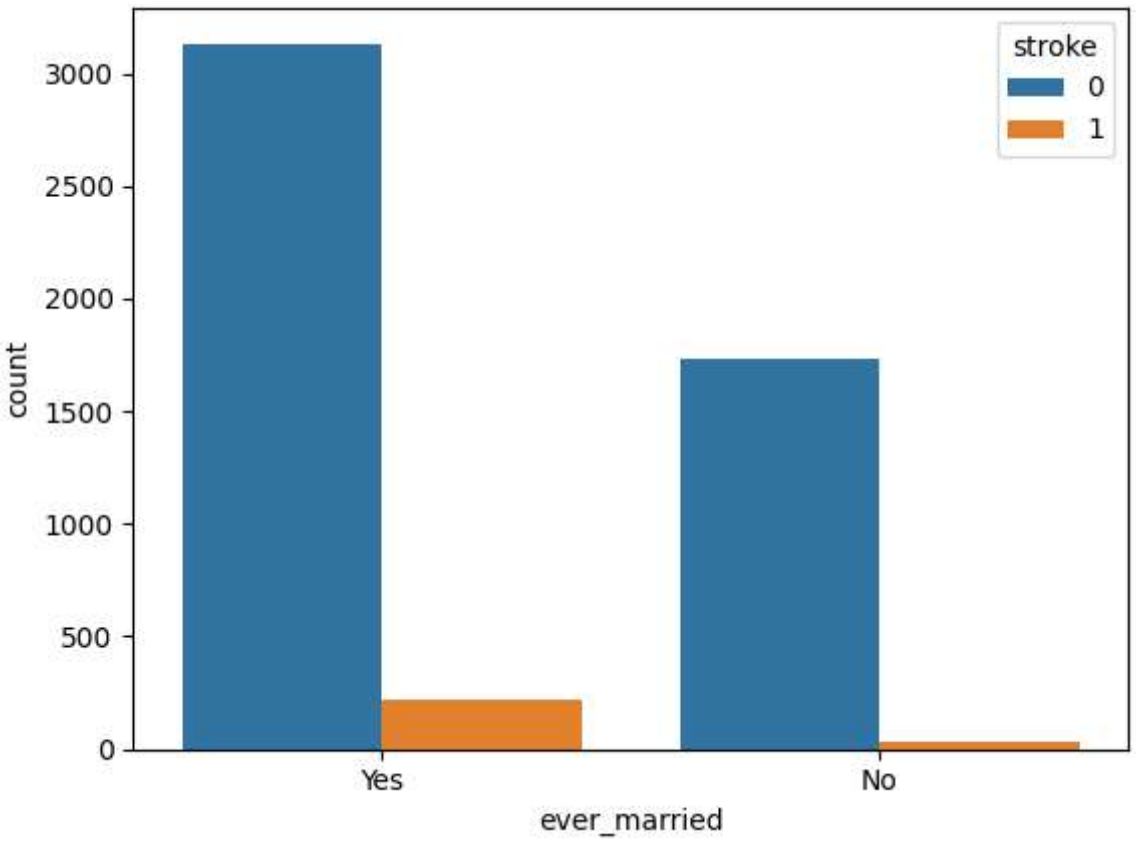
More men than women had stroke attack

Ever Married

In [24]:
```
data['ever_married'].value_counts()
```

Out[24]:
```
Yes    3353
No     1757
Name: ever_married, dtype: int64
```

In [25]:
```
sns.countplot(data=data,x="ever_married",hue='stroke')
plt.show()
```



Work Type

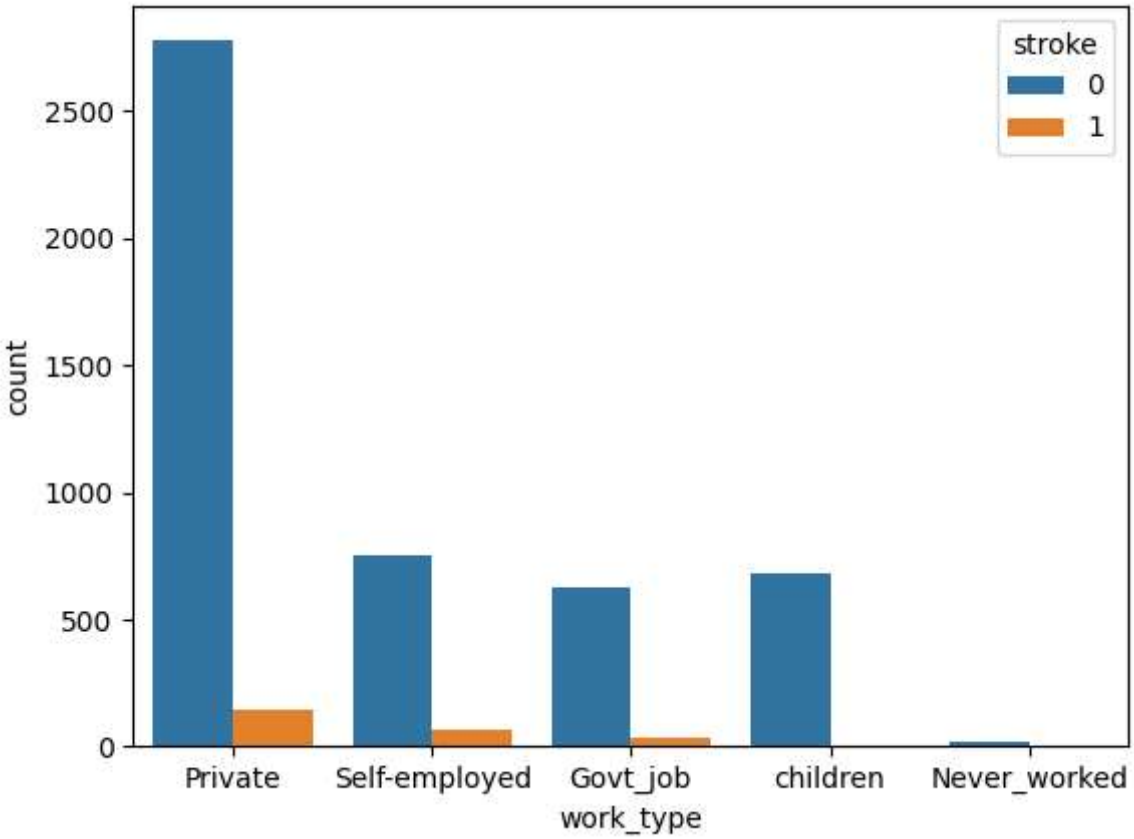
In [26]:
```
data['work_type'].unique()
```

Out[26]:
```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```

In [27]:
```
data['work_type'].value_counts()
```

Out[27]:
```
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

In [28]:
```
sns.countplot(data=data,x='work_type',hue='stroke')
plt.show()
```



Residence Type

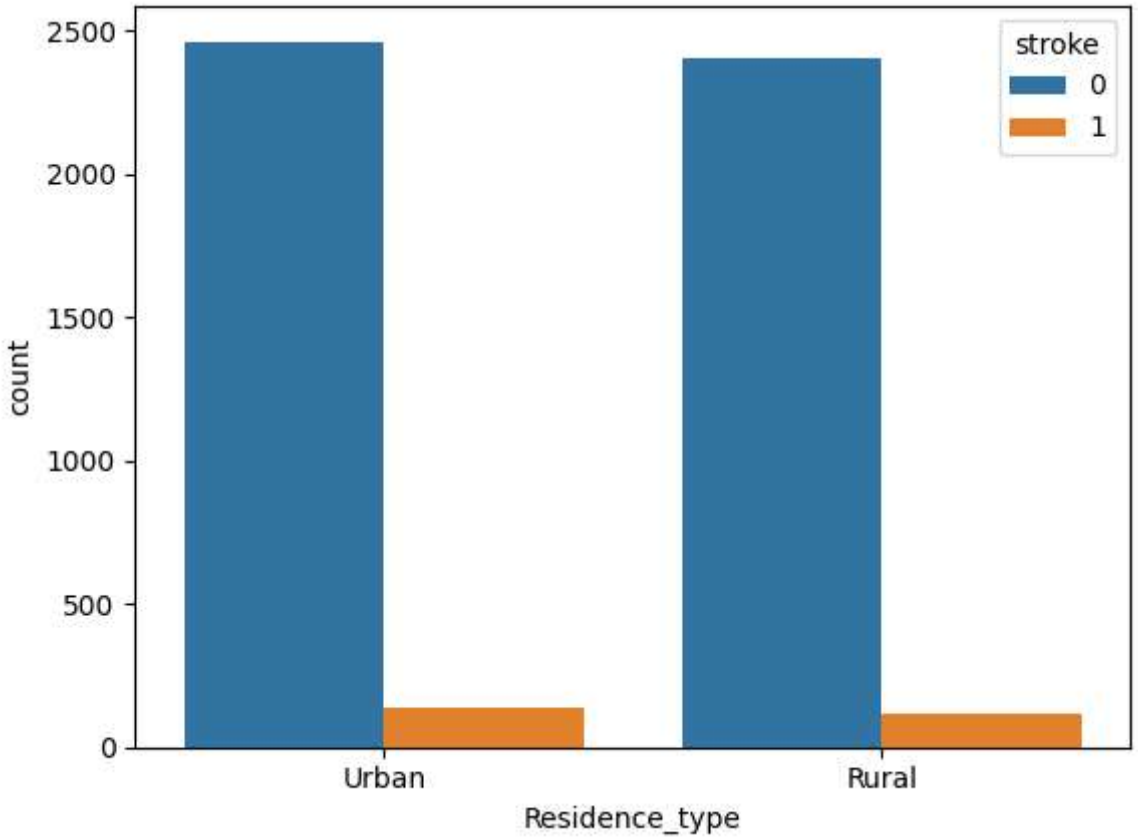
In [29]:
```
data['Residence_type'].unique()
```

Out[29]:
```
array(['Urban', 'Rural'], dtype=object)
```

In [30]:
```
data['Residence_type'].value_counts()
```

Out[30]:
```
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
```

In [31]:
```
sns.countplot(data=data,x='Residence_type',hue='stroke')
plt.show()
```

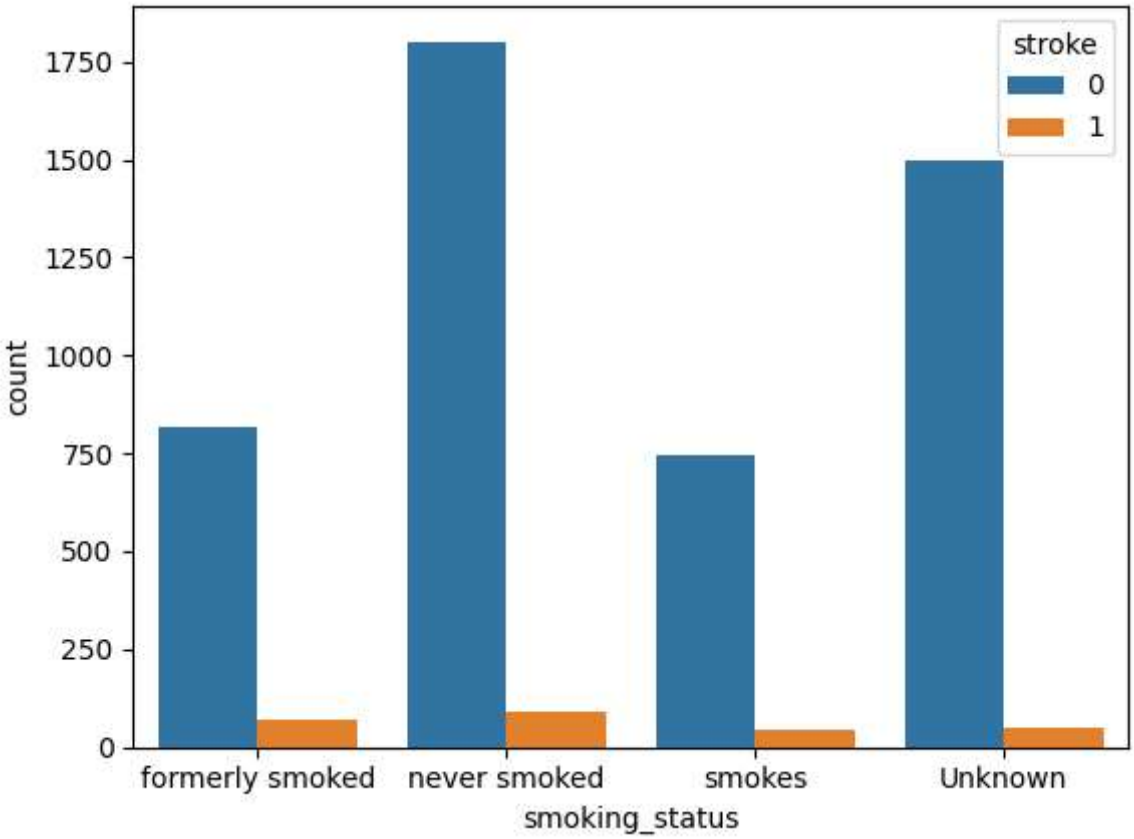


Smoking Features

In [32]:
```
data['smoking_status'].value_counts()
```

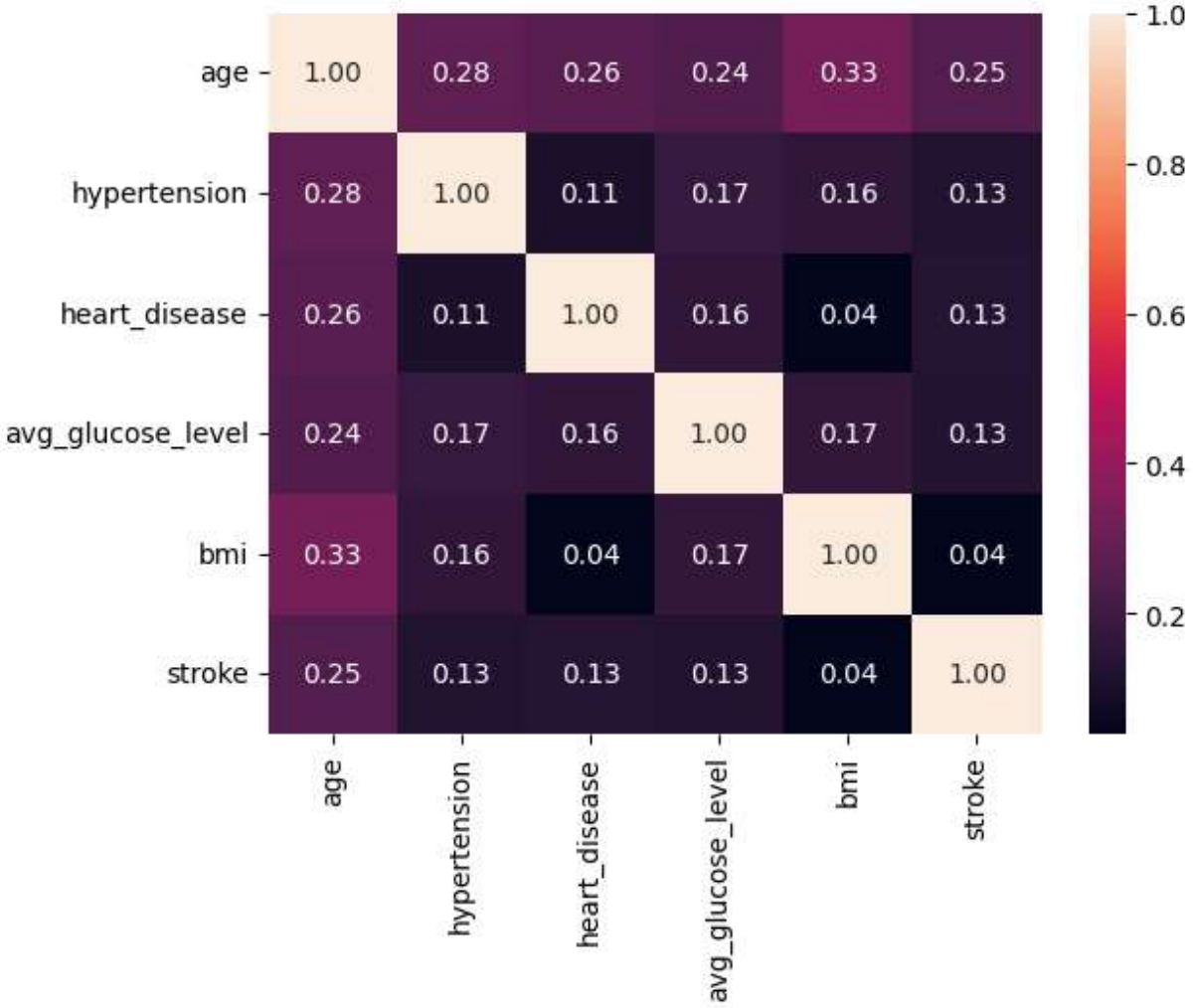
Out[32]:
```
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

In [33]:
```
sns.countplot(data=data,x='smoking_status',hue='stroke')
plt.show()
```



Heatmap

In [34]:
```
sns.heatmap(data.corr(),annot=True,fmt='.2f')
plt.show()
```



Encoding the categorical variables

In [35]:
```
data.dtypes
```

Out[35]:
```
gender               object
age                 float64
hypertension          int64
heart_disease         int64
ever_married         object
work_type            object
Residence_type       object
avg_glucose_level   float64
bmi                 float64
smoking_status       object
stroke                int64
dtype: object
```

In [54]:
```
from sklearn.preprocessing import LabelEncoder
lr = LabelEncoder()
```

In [55]:
```
data['gender'] = lr.fit_transform(data['gender'])
data['ever_married'] = lr.fit_transform(data['ever_married'])
data['work_type'] = lr.fit_transform(data['work_type'])
data['Residence_type'] = lr.fit_transform(data['Residence_type'])
data['smoking_status'] = lr.fit_transform(data['smoking_status'])
```

Splitting data into independent and dependent variables

In [56]:
```
X=data.drop('stroke',axis=1).values
X
```

Out[56]:
```
array([[  1.        ,  67.        ,   0.        , ..., 228.69      ,
          36.6       ,   1.        ],
       [  0.        ,  61.        ,   0.        , ..., 202.21      ,
          28.89323691,   2.        ],
       [  1.        ,  80.        ,   0.        , ..., 105.92      ,
          32.5       ,   2.        ],
       ...,
       [  0.        ,  35.        ,   0.        , ...,  82.99      ,
          30.6       ,   2.        ],
       [  1.        ,  51.        ,   0.        , ..., 166.29      ,
          25.6       ,   1.        ],
       [  0.        ,  44.        ,   0.        , ...,  85.28      ,
          26.2       ,   0.        ]])
```

In [57]:
```
Y=data['stroke'].values
Y
```

Out[57]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

Splitting

In [58]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_
```

In [52]:
```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\lenovo\appdata\local\pr
ograms\python\python310\lib\site-packages (1.2.1)Note: you may need to restart
the kernel to use updated packages.

Requirement already satisfied: numpy>=1.17.3 in c:\users\lenovo\appdata\local\p
rograms\python\python310\lib\site-packages (from scikit-learn) (1.24.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenovo\appdata
\local\programs\python\python310\lib\site-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\lenovo\appdata\local\pr
ograms\python\python310\lib\site-packages (from scikit-learn) (1.10.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\lenovo\appdata\local\p
rograms\python\python310\lib\site-packages (from scikit-learn) (1.2.0)
```

Logistic Regression

In [61]:
```
from sklearn.linear_model import LogisticRegression
```

In [62]:
```
classifier = LogisticRegression()
```

In [63]:
```
classifier.fit(X_train, Y_train)
```

Out[63]:
```
▾ LogisticRegression

LogisticRegression()
```

In [64]:
```
predict = classifier.predict(X_test)
predict
```

Out[64]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [65]:
```
Y_test
```

Out[65]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

Evaluation of Logistic Regression

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

In [67]:
```
print(confusion_matrix(Y_test, predict))
```

```
[[968   0]
 [ 54   0]]
```

In [68]:
```
print(classification_report(Y_test, predict))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       968
           1       0.00      0.00      0.00        54

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

In [70]:
```
print('Accuracy score :',accuracy_score(Y_test, predict))
```

```
Accuracy score : 0.9471624266144814
```

KNN Classifier

In [71]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [72]:
```
knn = KNeighborsClassifier()
```

In [73]:
```
knn.fit(X_train,Y_train)
```

Out[73]:
```
KNeighborsClassifier()
```

In [74]:
```
pred = knn.predict(X_test)
pred
```

Out[74]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [75]:
```
Y_test
```

Out[75]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

Evaluation of KNN Classifier

In [77]:
```
print('Accuracy score :',accuracy_score(Y_test, pred))
```

```
Accuracy score : 0.9422700587084148
```

Decision Tree Classifier

In [79]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [80]:
```
classifier = DecisionTreeClassifier(max_depth=3)
```

In [81]:
```
classifier.fit(X_train, Y_train)
```

Out[81]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

In [82]:
```
Y_pred = classifier.predict(X_test)
Y_pred
```

Out[82]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [83]:
```
Y_test
```

Out[83]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

Evaluation for Decision Tree Classifier

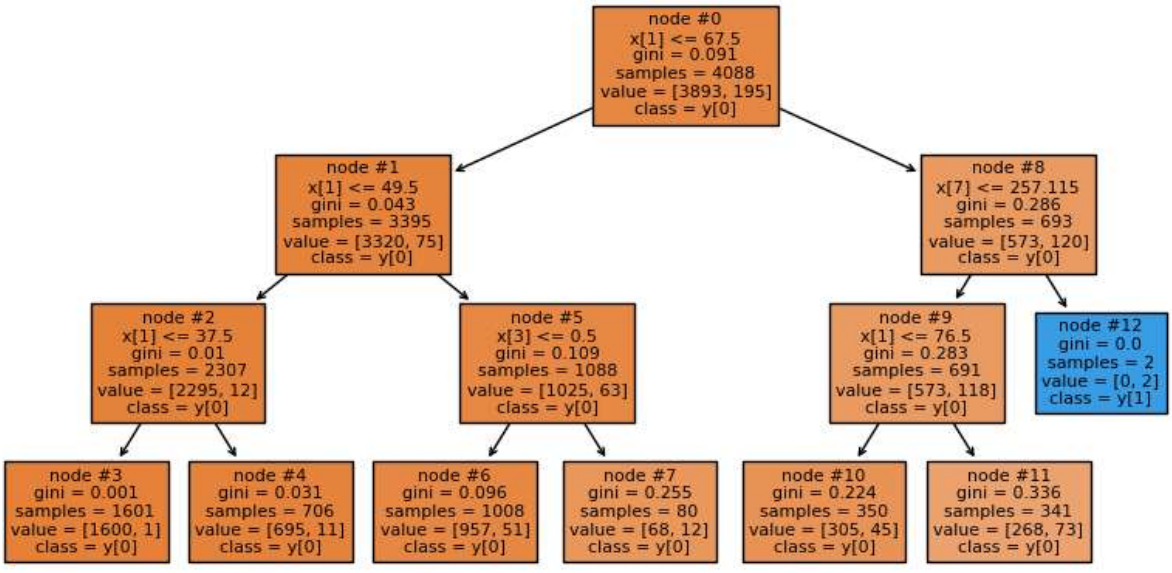
In [84]:
```
print('Accuracy:',accuracy_score(Y_test, Y_pred))
```

```
Accuracy: 0.9461839530332681
```

Ploting Tree with plot_tree

In [85]:
```
from sklearn import tree
```

In [87]:
```
fig = plt.figure(figsize=(10,5))
tree.plot_tree(classifier,filled=True,class_names=True,node_ids=True)
plt.show()
```



Random Forest Classifier

In [89]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [90]:
```
classifier = RandomForestClassifier()
```

In [91]:
```
classifier.fit(X_train,Y_train)
```

Out[91]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [92]:
```
Y_pred1 = classifier.predict(X_test)
Y_pred1
```

Out[92]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [93]:
```
Y_test
```

Out[93]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

Evaluation for Random Forest Classifier

In [94]:
```
print('Accuracy:', accuracy_score(Y_pred1, Y_test))
```

```
Accuracy: 0.9461839530332681
```

In [ ]:
```

```