About The Dataset

The dataset contains information about the persons who suffering with stroke disease.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

DATASET

```
df=pd.read_csv('/content/drive/MyDrive/Datasets/stroke.csv')
df
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	neve
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	neve
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	neve
...	...	...	...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	neve
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	neve
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	neve
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	

5110 rows × 12 columns

Next steps: Generate code with df View recommended plots

DATA PREPROCESSING AND CLEANING

```
df.columns
```

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

```
df.dtypes
```

```
id                   int64
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
```

```
df.isna().sum()
```

```
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
```

```
df.describe()
```

	id	age	hypertension	heart_disease	avg_glucose_level	
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.00
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.89
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.85
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.30
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.50
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.10
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.10
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.60

```
df.bmi.isna().sum()
```

```
201
```

```
df.bmi.fillna(df['bmi'].mean(),inplace=True)
```

```
df.isna().sum()
```

```
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
```

```
df.drop_duplicates(inplace=True)
```

```
df.drop(columns=['id'],inplace=True)
```

```
df
```

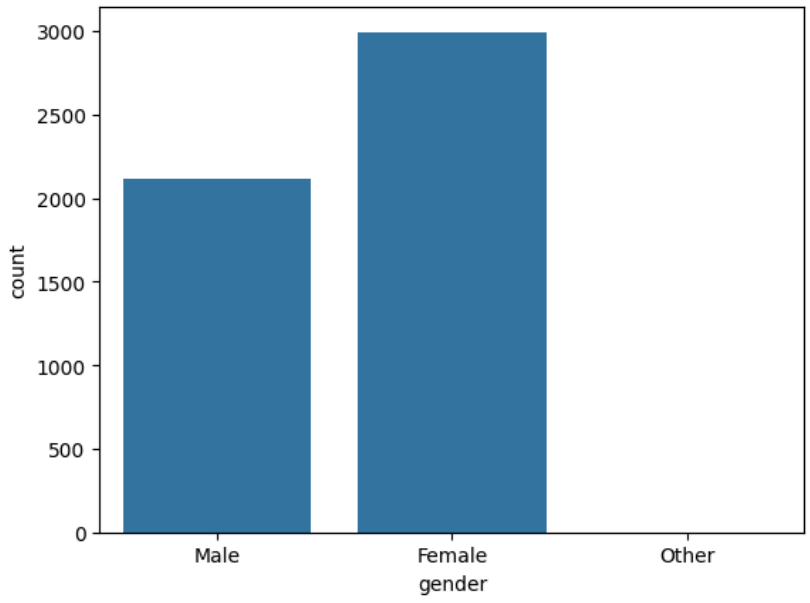
	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
0	Male	67.0	0	1	Yes	Private	Urban
1	Female	61.0	0	0	Yes	Self-employed	Rural
2	Male	80.0	0	1	Yes	Private	Rural
3	Female	49.0	0	0	Yes	Private	Urban
4	Female	79.0	1	0	Yes	Self-employed	Rural
...	...	...	...	...	...	...	...
5105	Female	80.0	1	0	Yes	Private	Urban
5106	Female	81.0	0	0	Yes	Self-employed	Urban
5107	Female	35.0	0	0	Yes	Self-employed	Rural
5108	Male	51.0	0	0	Yes	Private	Rural
5109	Female	44.0	0	0	Yes	Govt_job	Urban

5110 rows × 11 columns

Next steps: Generate code with df View recommended plots

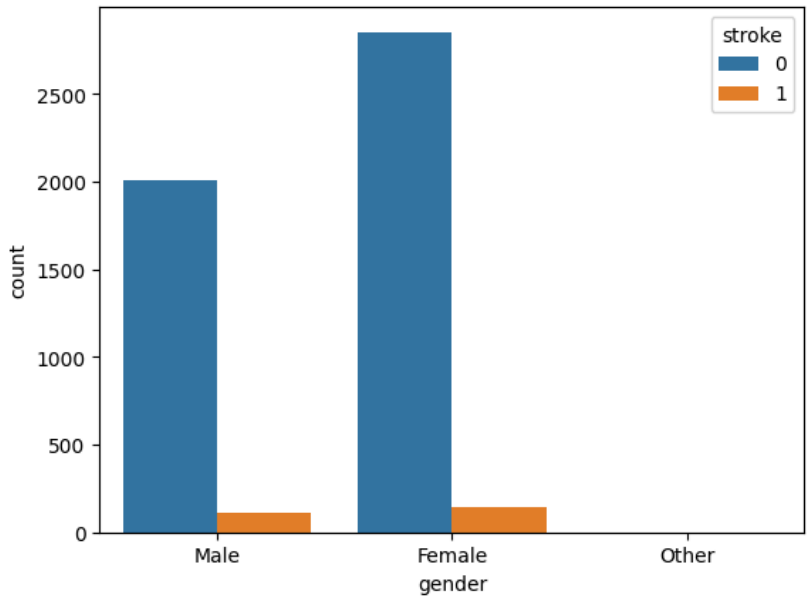
DATA VISUALIZATION

```
# Helps to plot a count plot which will help us to see count of values in each unique category.
sns.countplot(x=df['gender'])
plt.show()
```




```
# This plot will help to analyze how gender will affect chances of stroke.
sns.countplot(data=df,x='gender',hue='stroke')
```

```
<Axes: xlabel='gender', ylabel='count'>
```



Observation:

Dataset is imbalanced.There is much difference between stroke rate concerning gender

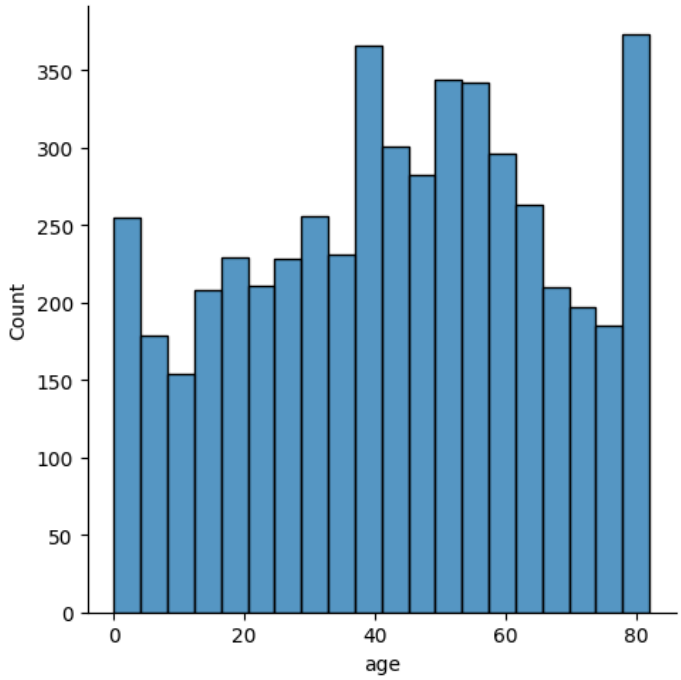
AGE

```
df['age'].nunique()
```

```
104
```

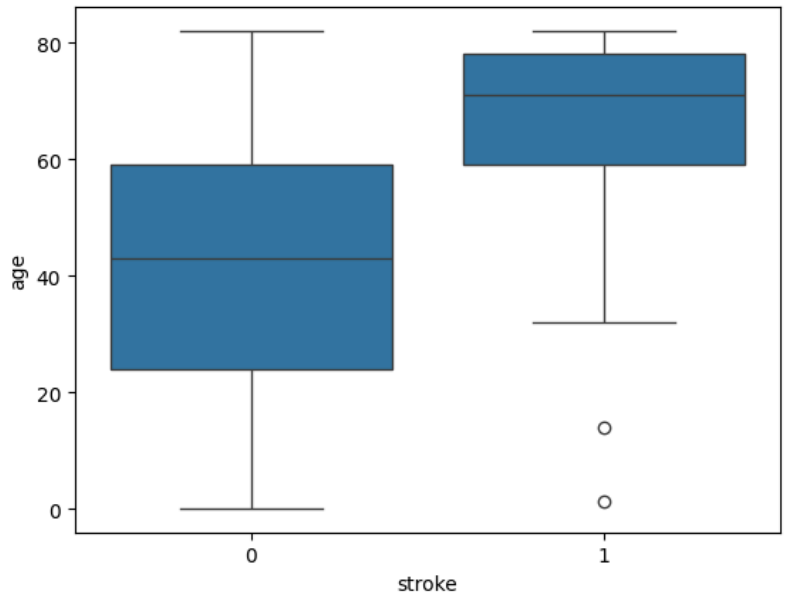
```
sns.displot(df['age'])
# This will plot a distribution plot of variable age
```

```
<seaborn.axisgrid.FacetGrid at 0x7ae3e7d6a2c0>
```



```
sns.boxplot(data=df,x='stroke',y='age')
# Above code will plot a boxplot of variable age with respect of target attribute stroke
```

```
<Axes: xlabel='stroke', ylabel='age'>
```



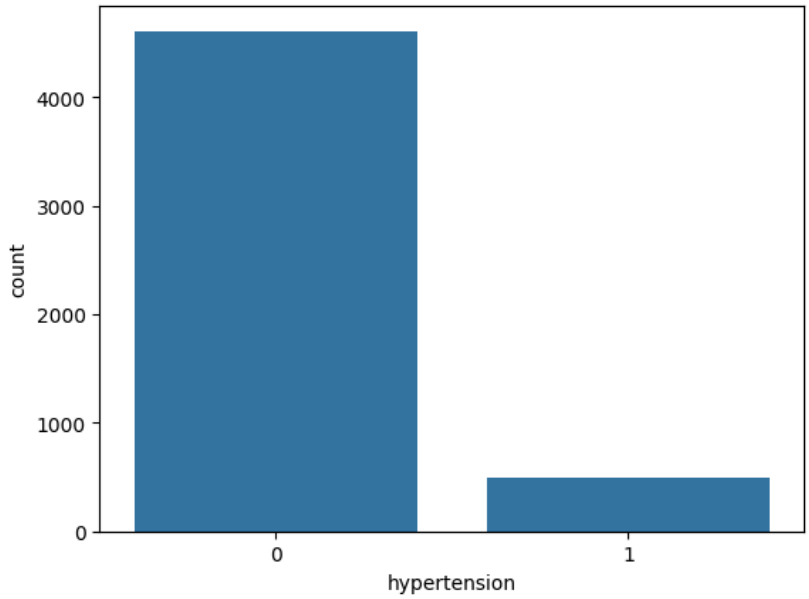
Observation:

People aged more than 60 years tend to have a stroke. Some outliers can be seen as people below age 20 are having a stroke it might be possible that it's valid data as stroke also depends on our eating and living habits. Another observation is people not having strokes also consist of people age > 60 years.

Hypertension

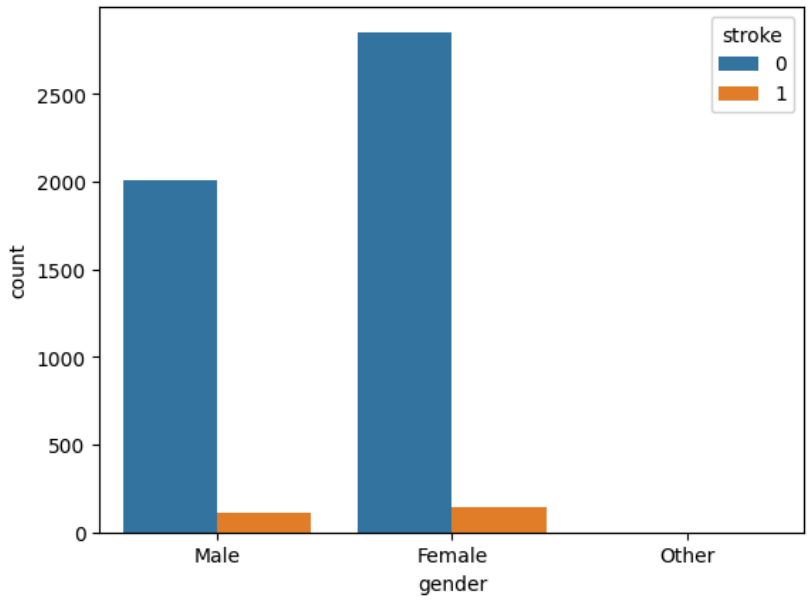
Hypertension is a condition when a person has high blood pressure. Hypertension might result in a stroke.

```
sns.countplot(x=df['hypertension'])
plt.show()
```




```
sns.countplot(data=df,x='gender',hue='stroke')
# Above code will plot a boxplot of variable hypertension with respect of stroke
```

```
<Axes: xlabel='gender', ylabel='count'>
```



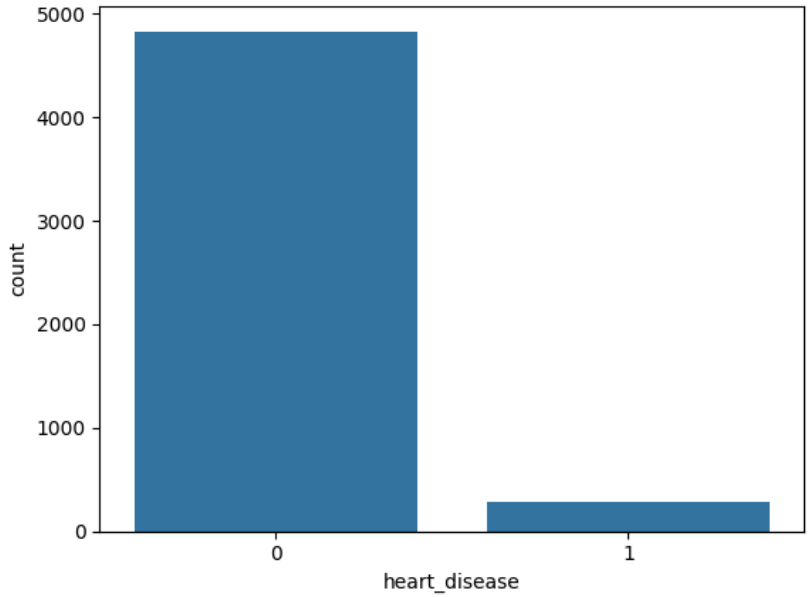
Observation:

Well, hypertension is rare in young people and common in aged people. Hypertension can cause a stroke. Based on our data picture is not that clear for hypertension. It has quite little data on patients having hypertension.

Heart Disease

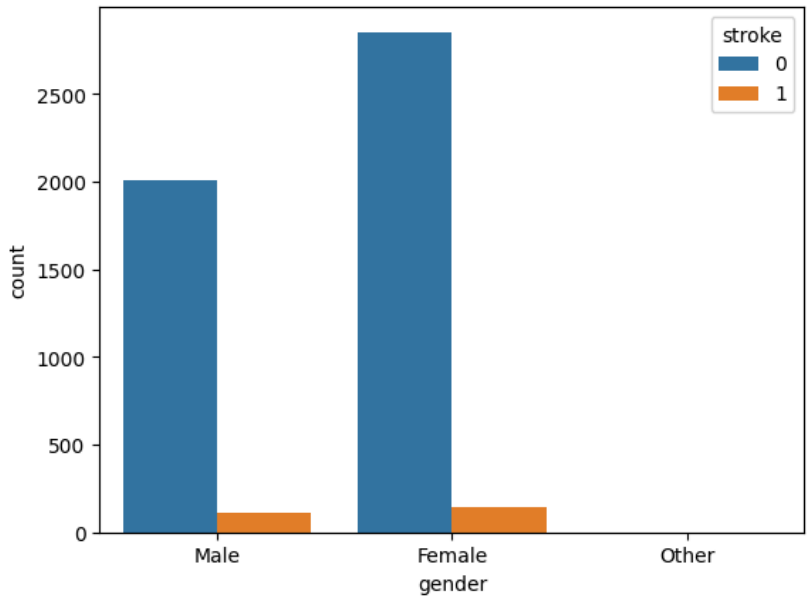
```
sns.countplot(data=df,x='heart_disease')
#  plot a counter plot of variable heart diseases
```

```
<Axes: xlabel='heart_disease', ylabel='count'>
```




```
sns.countplot(data=df,x='gender',hue='stroke')
```

```
<Axes: xlabel='gender', ylabel='count'>
```



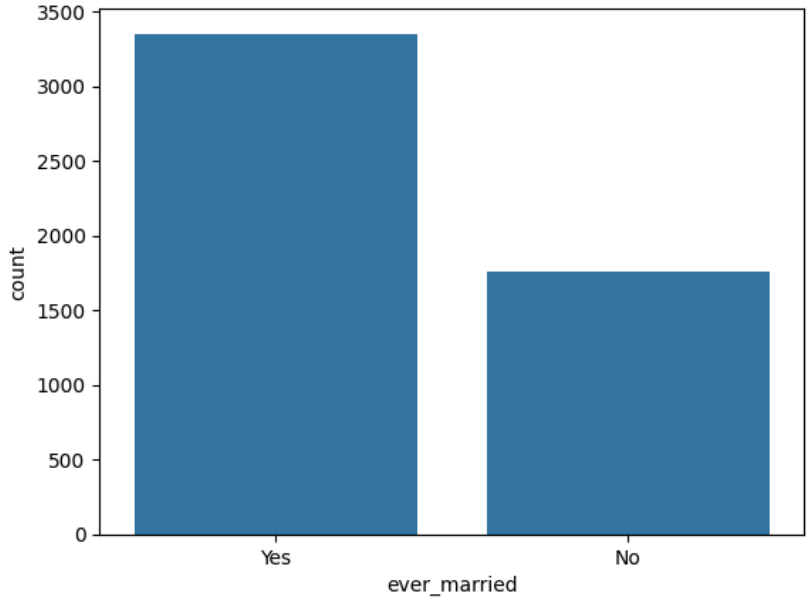
Observation:

Because of the imbalanced dataset, it's a little bit difficult to get an idea. But as per this plot, we can say that heart disease is not affecting Stroke.

Ever Married

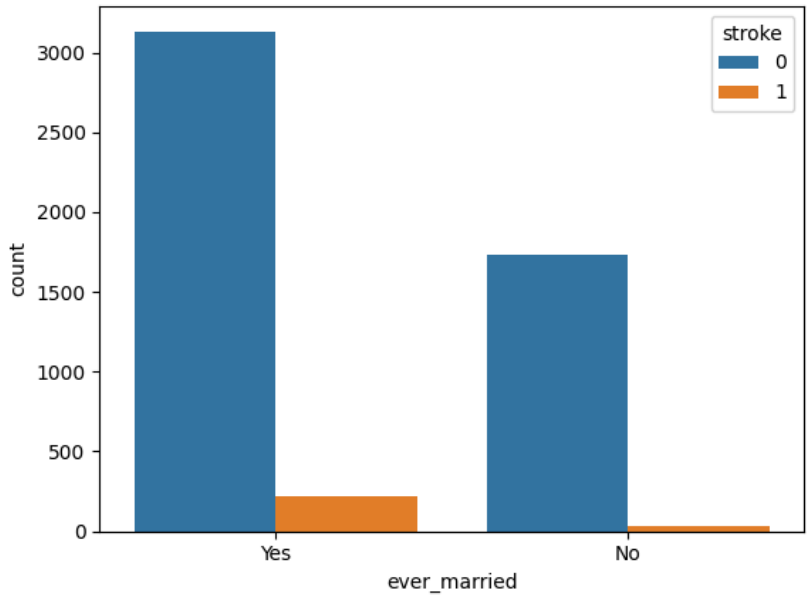
```
sns.countplot(data=df,x='ever_married')
# Counter plot of ever married attribute
```

```
<Axes: xlabel='ever_married', ylabel='count'>
```




```
sns.countplot(data=df,x='ever_married',hue='stroke')
# Ever married with respect of stroke
```

```
<Axes: xlabel='ever_married', ylabel='count'>
```



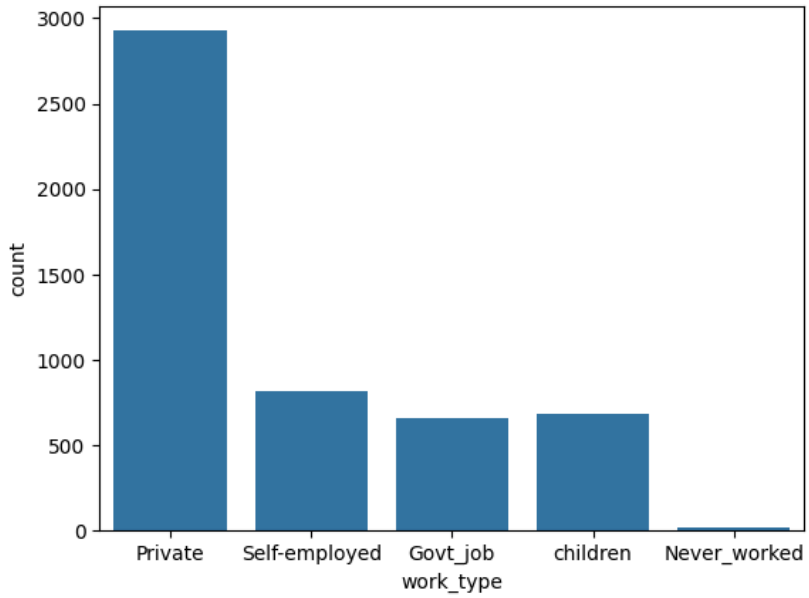
Observation:

People who are married have a higher stroke rate.

Work Type

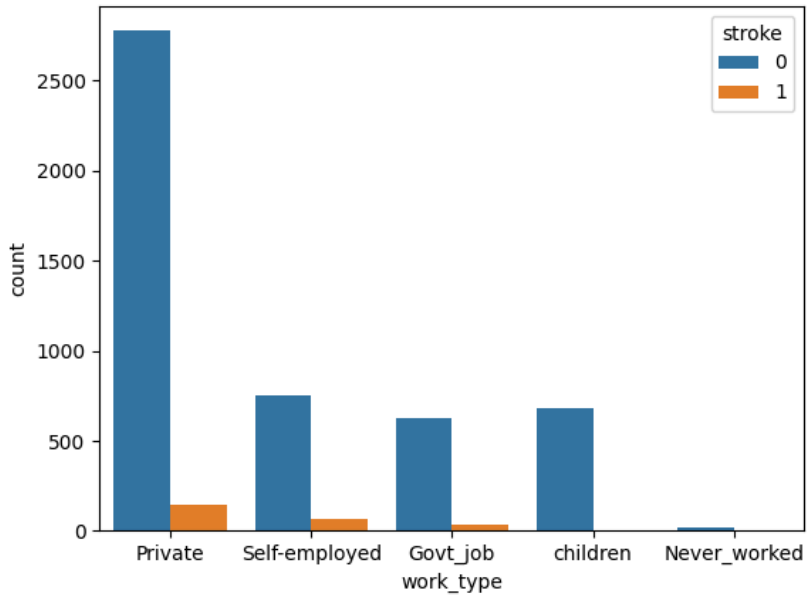
```
sns.countplot(data=df,x='work_type')
# Above code will create a count plot
```

```
<Axes: xlabel='work_type', ylabel='count'>
```




```
sns.countplot(data=df,x='work_type',hue='stroke')
# Above code will create a count plot with respect to stroke
```

```
<Axes: xlabel='work_type', ylabel='count'>
```



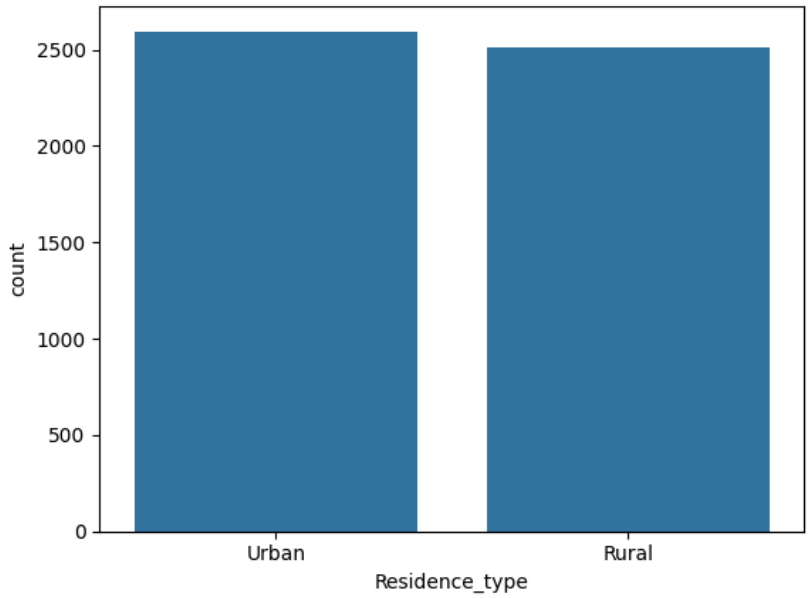
Observation:

People working in the Private sector have a higher risk of getting a stroke. And people who have never worked have a very less stroke rate.

Residence Type

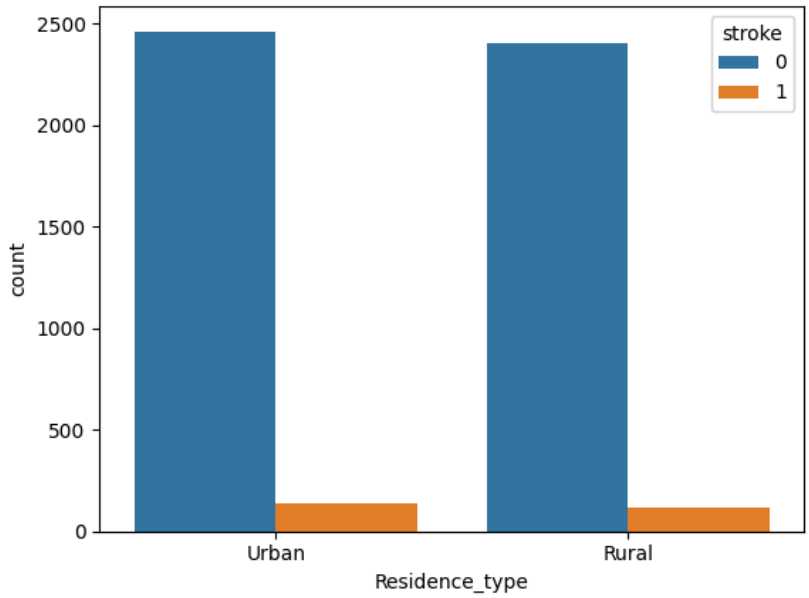
```
sns.countplot(data=df,x='Residence_type')
# This will create a counter plot
```

```
<Axes: xlabel='Residence_type', ylabel='count'>
```




```
sns.countplot(data=df,x='Residence_type',hue='stroke')
# Residence Type with respect to stroke
```

```
<Axes: xlabel='Residence_type', ylabel='count'>
```



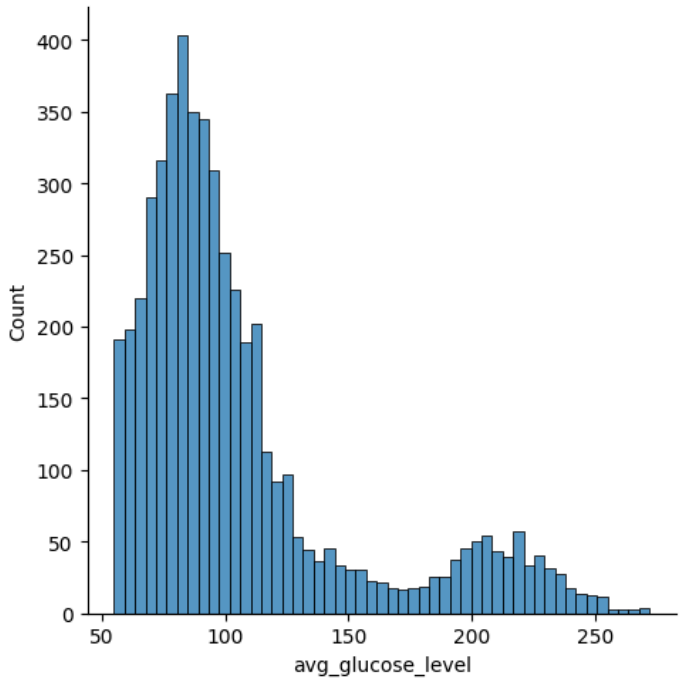
Observation:

This attribute is of no use. As we can see there not much difference in both attribute values. Maybe we have to discard it.

Average Glucose Level

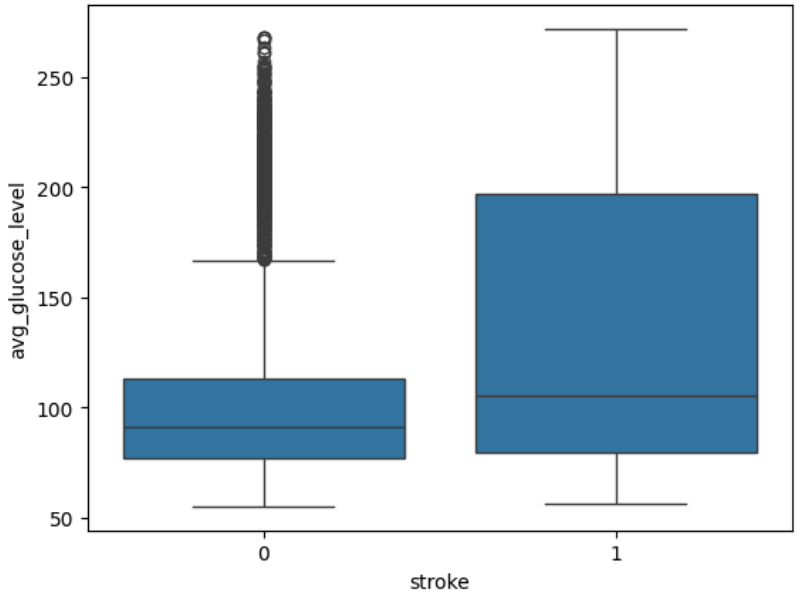
```
sns.displot(df['avg_glucose_level'])
# Distribution of avg_glucose_level
```

```
<seaborn.axisgrid.FacetGrid at 0x7ae3e79e3820>
```




```
sns.boxplot(data=df,x='stroke',y='avg_glucose_level')
# Avg_glucose_level and Stroke
```

```
<Axes: xlabel='stroke', ylabel='avg_glucose_level'>
```



Observation:

From this above graph, we can see that people having stroke have an average glucose level of more than 100. There are some obvious outliers in patients who have no stroke but there are some chances of this being genuine records.

BMI

```
sns.displot(df['bmi'])
# Distribution of bmi
```

```
<seaborn.axisgrid.FacetGrid at 0x7ae3e769b3a0>
```

```
sns.boxplot(data=df,x='stroke',y='bmi')
# BMI with respect to Stroke
```

Observation:

There is as such no prominent observation of how does BMI affects the chances of having a stroke.

Smoking Status

```
sns.countplot(data=df,x='smoking_status')
# Count plot of smoking status
```

```
sns.countplot(data=df,x='smoking_status',hue='stroke')
# Smoking Status with respect to Stroke
```

Observation:

As per these plots, we can see there is not much difference in the chances of stroke irrespective of smoking status.

Stroke

```
sns.countplot(data=df,x='stroke')
# Count Plot of Stroke
```

Smoking Status pie chart

```
plt.pie(df['smoking_status'].value_counts(),autopct='%1.1f%%')
plt.legend(df['smoking_status'].value_counts().index,loc=(1,0.5))
plt.title('smoking_status',color='red')
```

LabelEncoding

LabelEncoding Machine learning models work only with numerical values.so categorical columns are converted in to numerical values.

```
from sklearn.preprocessing import LabelEncoder
list=['gender','ever_married','work_type','Residence_type','smoking_status']
dict1={}
for columns in list:
   dict1[columns]=LabelEncoder()
   df[columns]=dict1[columns].fit_transform(df[columns])
dict1
```

BALANCING THE DATA,SPLITTING THE DATA AND SCALING THE FEATURES

BALANCING THE DATA

A balanced dataset is a dataset where each output class (or target class) is represented by the same number of input samples. Balancing can be performed by exploiting one of the following techniques: oversampling,undersampling

TRAIN TEST SPLIT

A train test split function is used to split data in to training set and testing set.Training set are used to train the model and test their accuracy on unseen data(testing data).

SCALING USING STANDARDSCALER

Data standardization involves scaling data values so that they have a mean of 0 and standard deviation of 1, and is best for normal distributions. Normalization in machine learning is the process of translating data into the range[0,1] or simply transforming data onto the unit sphere.

```
df.stroke.value_counts()
```

BALANCING THE DATA

```
from imblearn.over_sampling import SMOTE
oversampler=SMOTE()
```

```
x=df.drop(columns='stroke')
y=df.stroke
```

TRAIN TEST SPLIT AND SCALING

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_scaled=sc.fit_transform(x_train)
x_test_scaled=sc.transform(x_test)
x_train_scaled_sample,y_train_sample=oversampler.fit_resample(x_train_scaled,y_train)
```

```
y_train_sample.value_counts()
```

MODEL CREATION

K-Nearest Neighbors algorithm(KNN) The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today.

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=21)
knn.fit(x_train_scaled,y_train)
y_pred_knn=knn.predict(x_test_scaled)
y_pred_knn
```

```
np.array(y_test)
```

NAIVE_BAYES

Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem.

```
from sklearn.naive_bayes import BernoulliNB
naive=BernoulliNB()
naive.fit(x_train_scaled,y_train)
y_pred_naive=naive.predict(x_test_scaled)
y_pred_naive
```

SUPPORT VECTOR MACHINE

A support vector machine (SVM) is a type of supervised learning algorithm used in machine learning to solve classification and regression tasks; SVMs are particularly good at solving binary classification problems, which require classifying the elements of a data set into two groups.

```
from sklearn.svm import SVC
sv=SVC()
sv.fit(x_train_scaled,y_train)
y_pred_sv=sv.predict(x_test_scaled)
y_pred_sv
```

DECISION TREE

A decision tree is a non-parametric supervised learning algorithm for classification and regression tasks. It has a hierarchical tree structure consisting of a root node, branches, internal nodes, and leaf nodes. Decision trees are used for classification and regression tasks, providing easy-to-understand models.

```
from sklearn.tree import DecisionTreeClassifier
tree=DecisionTreeClassifier(max_depth=20,criterion='gini')
tree.fit(x_train_scaled,y_train)
y_pred_tree=tree.predict(x_test_scaled)
y_pred_tree
```

RANDOM FOREST

Random forest is a commonly-used machine learning algorithm that combines the output of multiple decision trees to reach a single result.

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100,max_depth=100,min_samples_split=5,criterion='gini',bootstrap=False)
rf.fit(x_train_scaled,y_train)
y_pred_rf=rf.predict(x_test_scaled)
y_pred_rf
```

XGBOOST

XGBoost is a boosting algorithm that uses bagging, which trains multiple decision trees and then combines the results. It allows XGBoost to learn more quickly than other algorithms but also gives it an advantage in situations with many features to consider.

```
import xgboost
xb=xgboost.XGBClassifier()
xb.fit(x_train_scaled,y_train)
y_pred_xb=xb.predict(x_test_scaled)
y_pred_xb
```

CLASSIFICATION REPORT AND CONFUSION MATRIX

A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score,accuracy score and support of your trained classification model.

Confusion Matrix - an overview | ScienceDirect Topics A confusion matrix is a table that is used to define the performance of a classification algorithm. A confusion matrix visualizes and summarizes the performance of a classification algorithm

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,ConfusionMatrixDisplay
print(classification_report(y_test,y_pred_knn))
```

```
print(accuracy_score(y_test,y_pred_knn))
```

```
print(classification_report(y_test,y_pred_naive))
```

```
print(accuracy_score(y_test,y_pred_naive))
```

```
print(classification_report(y_test,y_pred_sv))
```

```
print(accuracy_score(y_test,y_pred_sv))
```

```
print(classification_report(y_test,y_pred_tree))
```

```
print(accuracy_score(y_test,y_pred_tree))
```

```
print(classification_report(y_test,y_pred_rf))
```

```
print(accuracy_score(y_test,y_pred_rf))
```

```
print(classification_report(y_test,y_pred_xb))
```

```
print(accuracy_score(y_test,y_pred_xb))
```

CONFUSION MATRIX DISPLAY

```
label=[0,1]
matx_knn=confusion_matrix(y_pred_knn,y_test)
print(matx_knn)
cmd=ConfusionMatrixDisplay(matx_knn,display_labels=label)
cmd.plot()
```

```
label=[0,1]
matx_naive=confusion_matrix(y_pred_naive,y_test)
print(matx_naive)
cmd=ConfusionMatrixDisplay(matx_naive,display_labels=label)
cmd.plot()
```

```
label=[0,1]
matx_sv=confusion_matrix(y_pred_sv,y_test)
print(matx_sv)
cmd=ConfusionMatrixDisplay(matx_sv,display_labels=label)
cmd.plot()
```

```
label=[0,1]
matx_tree=confusion_matrix(y_pred_tree,y_test)
print(matx_tree)
cmd=ConfusionMatrixDisplay(matx_tree,display_labels=label)
cmd.plot()
```

```
label=[0,1]
matx_rf=confusion_matrix(y_pred_rf,y_test)
print(matx_rf)
cmd=ConfusionMatrixDisplay(matx_rf,display_labels=label)
cmd.plot()
```