brainstrokeprediction

April 18, 2024

[ ]:
```python
# The libraries used in processing the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import imblearn as ib
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

[ ]:
```python
# The dataframe is read from the csv file - healthcare-dataset-stroke-data.csv␣
↪- taken from kaggle
df = pd.read_csv("healthcare-dataset-stroke-data.csv")
```

[ ]:
```python
# The first 5 instances of the dataframe
df.head()
```

[ ]:
```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes

       work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0        Private          Urban             228.69  36.6  formerly smoked
1  Self-employed          Rural             202.21   NaN     never smoked
2        Private          Rural             105.92  32.5     never smoked
3        Private          Urban             171.23  34.4           smokes
4  Self-employed          Rural             174.12  24.0     never smoked

   stroke
0       1
1       1
2       1
3       1
4       1
```

[ ]:
```
df.value_counts()
```

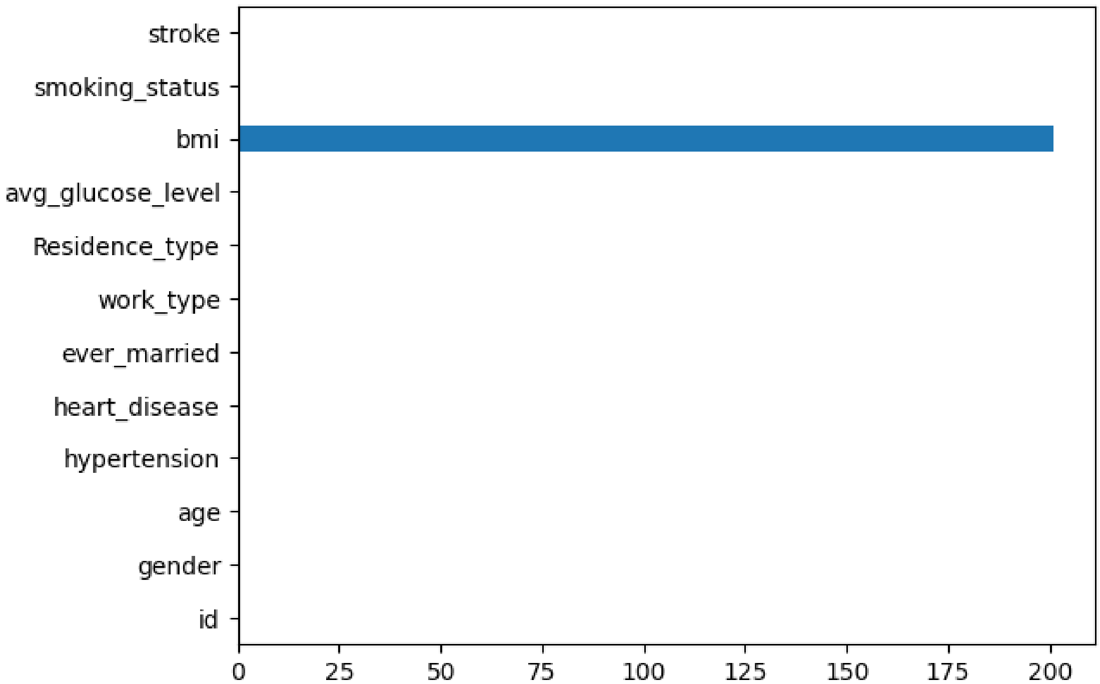
[ ]:
```
id     gender  age   hypertension  heart_disease  ever_married  work_type
Residence_type  avg_glucose_level  bmi   smoking_status   stroke
77     Female  13.0  0             0              No            children
Rural           85.81              18.6  Unknown          0           1
49605  Male    63.0  0             0              Yes           Private
Urban           74.39              31.0  formerly smoked  0           1
49661  Male    53.0  0             0              Yes           Govt_job
Urban           85.17              29.2  never smoked     0           1
49646  Male    72.0  0             1              Yes           Self-employed
Rural           113.63             26.5  Unknown          0           1
49645  Male    58.0  0             0              No            Private
Rural           76.22              22.2  formerly smoked  0           1
                                                                     ..
25138  Female  78.0  1             0              Yes           Private
Rural           91.63              33.5  smokes           0           1
25130  Female  27.0  0             0              Yes           Private
Urban           79.21              19.5  Unknown          0           1
25107  Female  47.0  0             0              Yes           Private
Urban           65.04              30.9  never smoked     0           1
25102  Female  51.0  0             0              Yes           Govt_job
Urban           95.16              42.7  formerly smoked  0           1
72940  Female  2.0   0             0              No            children
Urban           102.92             17.6  Unknown          0           1
Name: count, Length: 4909, dtype: int64
```

0.0.1 Find the number of NULL values in each column

[ ]:
```
# Printing the number of N/A values in eacg column
print(df.isna().sum())
# Graphical representation of the na values present in the attribute - bar graph
df.isna().sum().plot.barh()
```

```
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
```

[ ]:
```
<Axes: >
```



- Found 201 NULL values in bmi column

[ ]:
```python
# To check the  statistical analysis of all numerical type attributes  (count,␣
↪mean, standaard deviation, minimum values, all quartiles, maximum values)
df.describe()
```

[ ]:
```
                 id          age  hypertension  heart_disease  \
count   5110.000000  5110.000000   5110.000000    5110.000000
mean   36517.829354    43.226614      0.097456       0.054012
std    21161.721625    22.612647      0.296607       0.226063
min       67.000000     0.080000      0.000000       0.000000
25%    17741.250000    25.000000      0.000000       0.000000
50%    36932.000000    45.000000      0.000000       0.000000
75%    54682.000000    61.000000      0.000000       0.000000
max    72940.000000    82.000000      1.000000       1.000000

       avg_glucose_level          bmi       stroke
count        5110.000000  4909.000000  5110.000000
mean          106.147677    28.893237     0.048728
std            45.283560     7.854067     0.215320
min            55.120000    10.300000     0.000000
25%            77.245000    23.500000     0.000000
50%            91.885000    28.100000     0.000000
```

```
75%          114.090000    33.100000     0.000000
max          271.740000    97.600000     1.000000
```

[ ]:
```
# Provides the data type of all attributes and the number of NOT NULL values␣
↪count is obtained
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

0.1 PRE PROCESSING + EDA

[ ]:
```
# The 'id' column is dropped since the attribute holds no significant␣
↪importance to the problem at hand
df = df.drop(['id'],axis=1)
```

0.1.1 Gender analysis

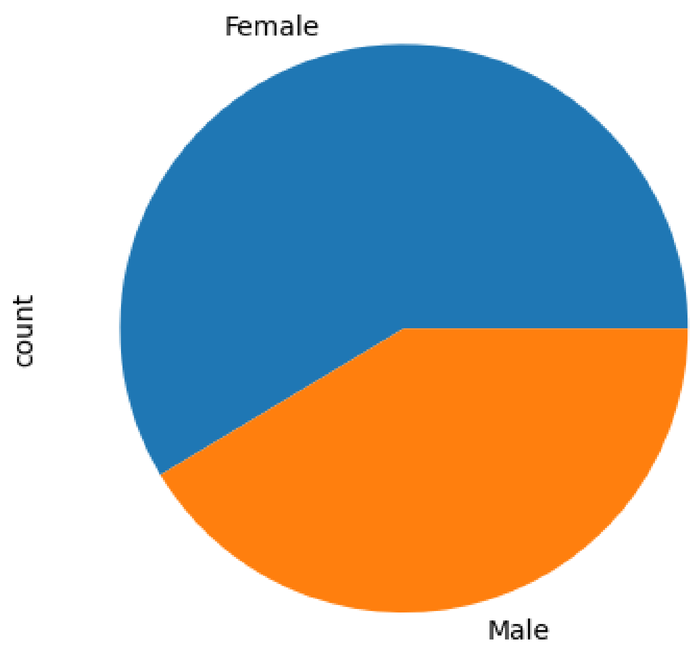
[ ]:
```
# Checking the values in the gender column
df['gender'].value_counts()
```

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

- We have a ‘other’ gender and since there is only 1 instance we will remove it as to reduce the dimension

```
[ ]: # Removing the 'other' gender instance inorder to reduce the dimension
df['gender'] = df['gender'].replace('Other','Female')
# plotting a pie chart to see the gender count distribution
df['gender'].value_counts().plot(kind="pie")
```

```
[ ]: <Axes: ylabel='count'>
```



- There are more females as compared to males

0.2 Target feature - Stroke

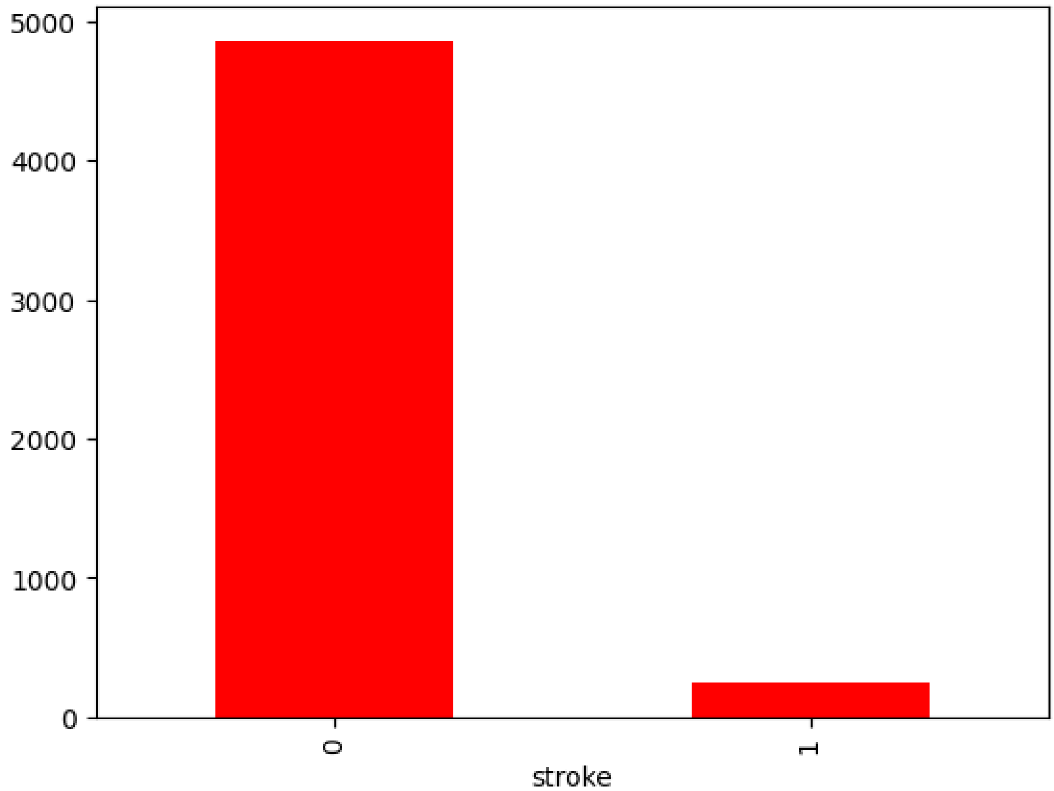
- Stroke analysis

```
[ ]: # Value count in the stroke attribute
df['stroke'].value_counts()
```

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

```
[ ]: # Graphical representation of the value count distribution of the target␣
 ↪attribute
df['stroke'].value_counts().plot(kind="bar",color = "red")
```

[ ]: <Axes: xlabel='stroke'>




```
[ ]: print("% of people who actualy got a stroke : ",(df['stroke'].value_counts()[1]/
      ↪df['stroke'].value_counts().sum()).round(3)*100)
```

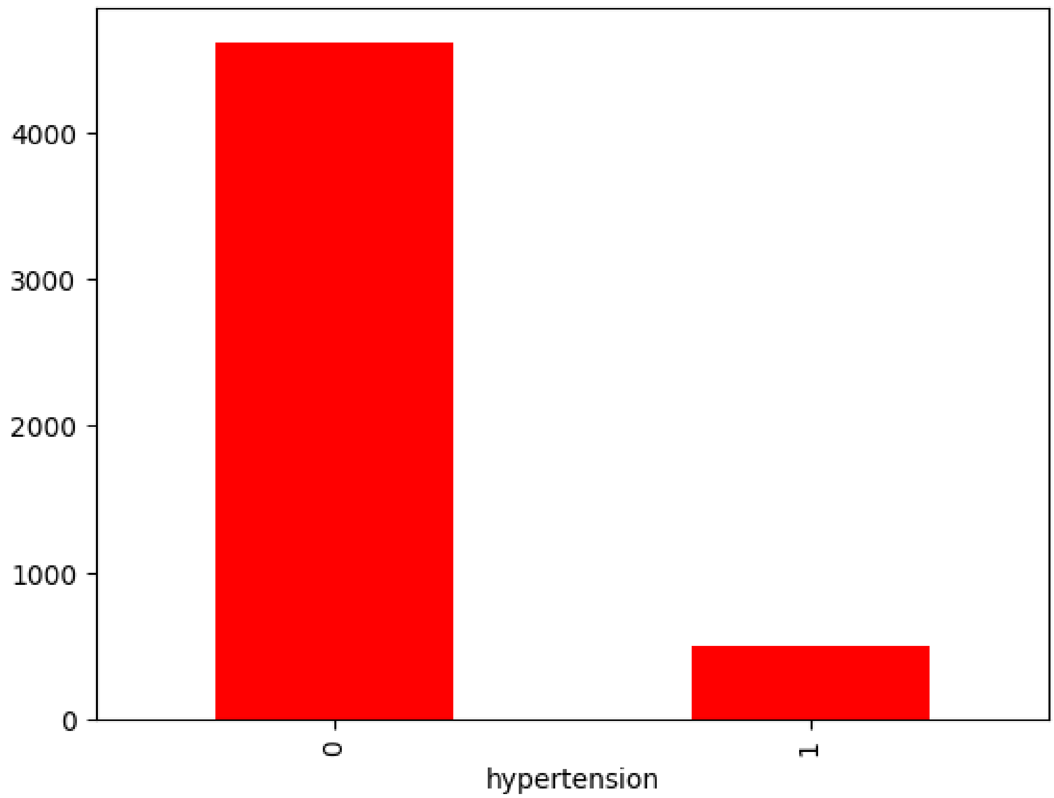
```
% of people who actualy got a stroke :  4.9
```

- Our dataset is highly skewed since only around 5% of the instances got stroke
- We will be needing to perform necessary transformations to improve samples of minority class

0.2.1 Hyper-tension Analysis

```
[ ]: # Graphical representation of the value counts of the hypertension attribute
     df['hypertension'].value_counts().plot(kind="bar",color = "red")
```

[ ]: <Axes: xlabel='hypertension'>



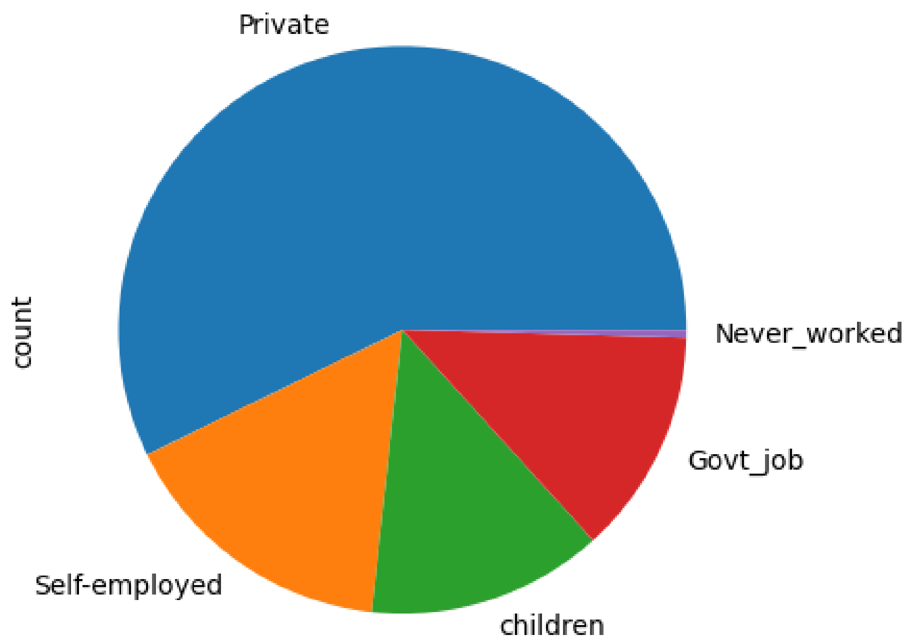
0.2.2 Work type Analysis

```
[ ]: # Value of count of work-type attribute
     df['work_type'].value_counts()
```

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

```
[ ]: # Graphical representation of the value counts of the work-type attribute
     df['work_type'].value_counts().plot(kind="pie")
```

```
[ ]: <Axes: ylabel='count'>
```



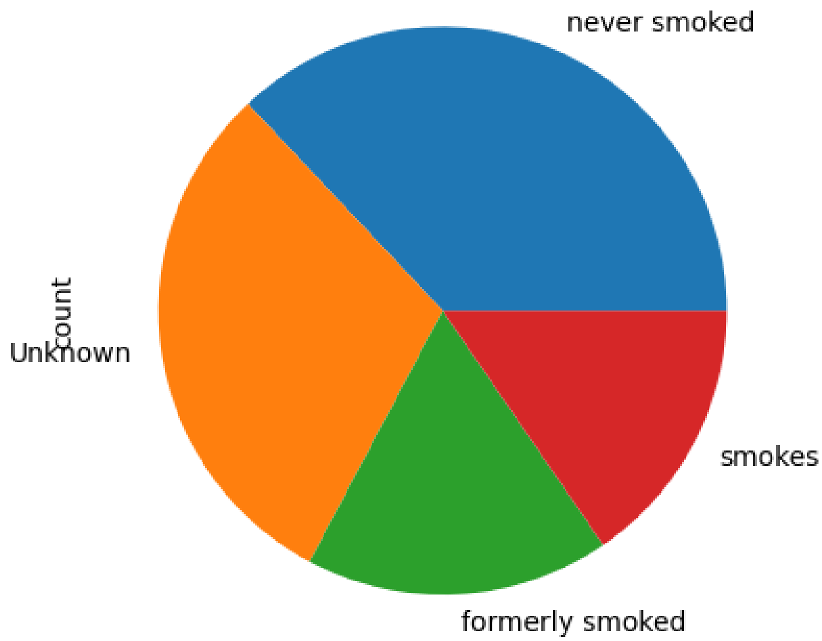
0.2.3 Smoking status Analysis

[ ]:
```
# Value of count of somoking status attribute
df['smoking_status'].value_counts()
```

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

[ ]:
```
# Graphical representation of the value counts of the smoking staus attribute
df['smoking_status'].value_counts().plot(kind="pie")
```

[ ]:
```
<Axes: ylabel='count'>
```



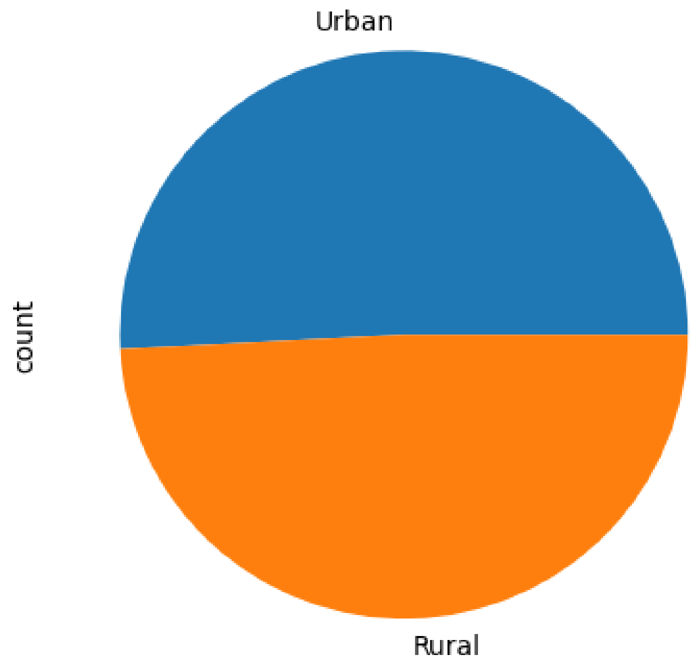
0.2.4 Residence type Analysis

[ ]:
```
# Value of count of residence attribute
df['Residence_type'].value_counts()
```

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

[ ]:
```
# Graphical representation of the value counts of the residence attribute
df['Residence_type'].value_counts().plot(kind="pie")
```

[ ]:
```
<Axes: ylabel='count'>
```



- We have an equal percentage of population who are from Urban and rural areas

0.2.5 BMI analysis

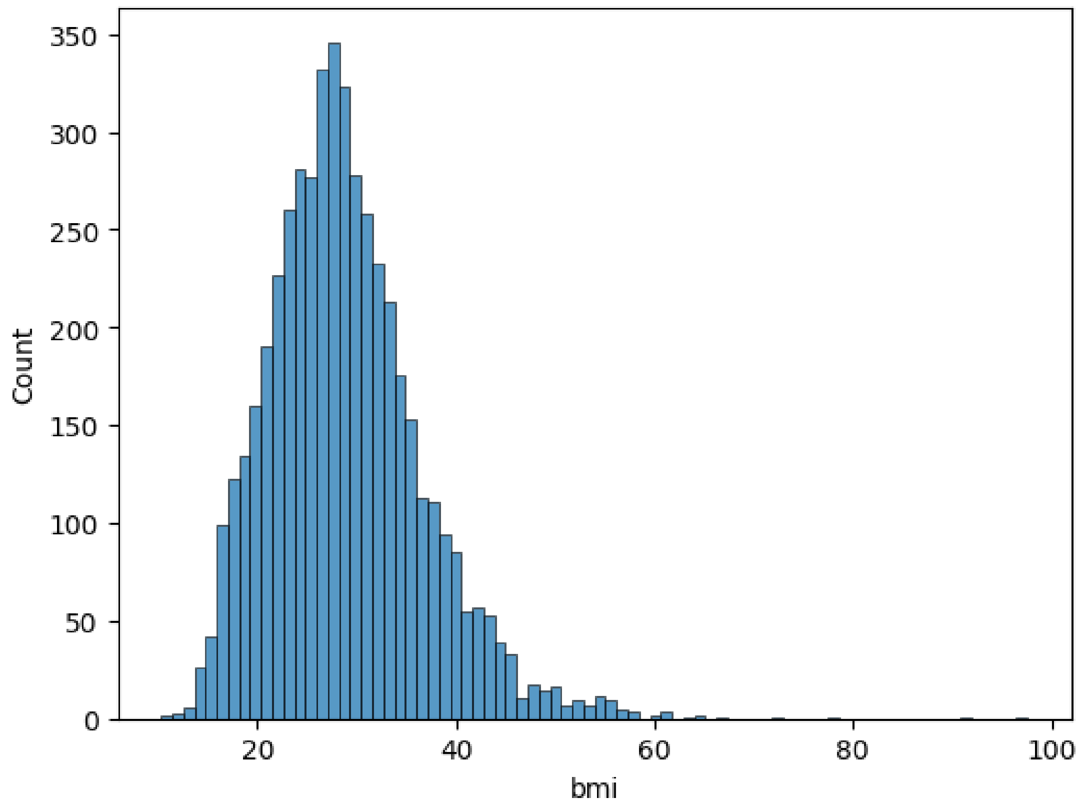
[ ]:
```
# Number of BMI - NULL values
df['bmi'].isnull().sum()
```

[ ]: 201

- We only have N/A values in bmi column - 201 Null values

[ ]:
```
# Graphical representation of bmi attribute
sns.histplot(data=df['bmi'])
```

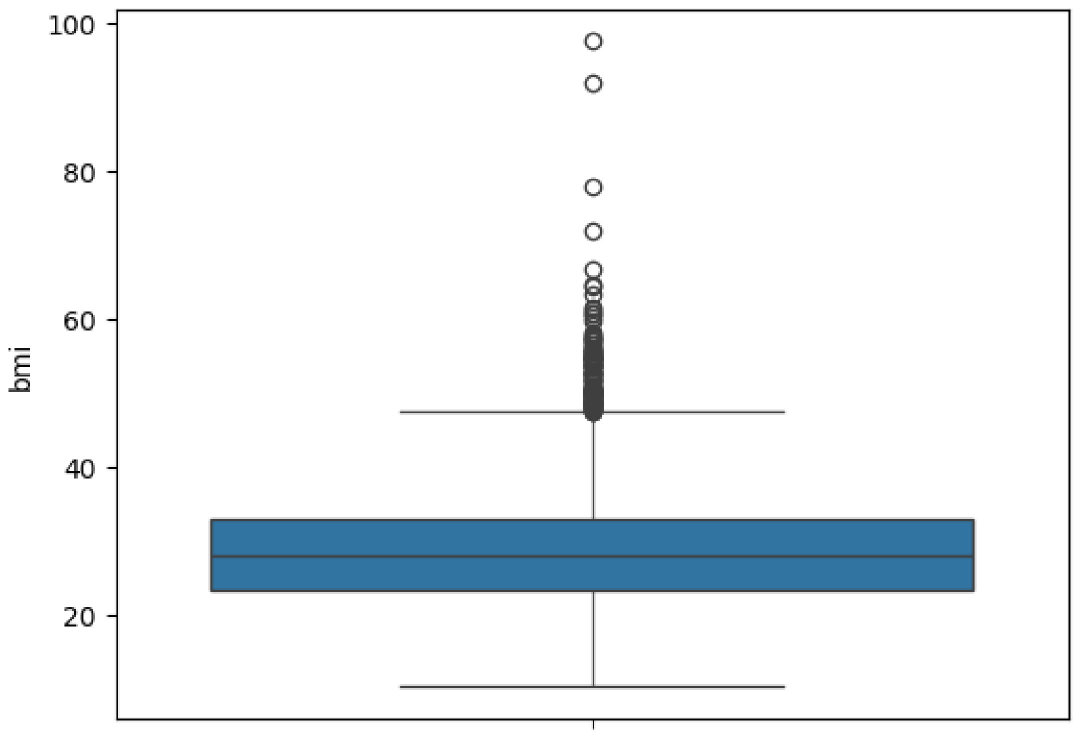
[ ]: <Axes: xlabel='bmi', ylabel='Count'>



- Bmi is rightly skewed

[ ]:
```
sns.boxplot(data=df['bmi'])
```

[ ]: <Axes: ylabel='bmi'>



- Based on the histogram and boxplot we see that there are many outliers in bmi

[ ]:
```
# Finding the count of outliers based on those instances which are out of iqr
Q1 = df['bmi'].quantile(0.25)
Q3 = df['bmi'].quantile(0.75)
# Finding IQR
IQR = Q3 - Q1
da=(df['bmi'] < (Q1 - 1.5 * IQR)) | (df['bmi'] > (Q3 + 1.5 * IQR))
da.value_counts()
```

[ ]:
```
bmi
False    5000
True      110
Name: count, dtype: int64
```

- Total outliers in bmi:110
- Total non-outliers in bmi:5000

[ ]:
```
# Percentage of NULL values in bmi
df['bmi'].isna().sum()/len(df['bmi'])*100
```

[ ]:
```
3.9334637964774952
```

- NULL values hold 3.93 % of the instances in the dataframe

[ ]:
```
df_na=df.loc[df['bmi'].isnull()]
g=df_na['stroke'].sum()
print("People who got stroke and their BMI is NA:",g)
h=df['stroke'].sum()
print("People who got stroke and their BMI is given:",h)
print("percentage of people with stroke in Nan values to the overall dataset:
↪",g/h*100)
```

```
People who got stroke and their BMI is NA: 40
People who got stroke and their BMI is given: 249
percentage of people with stroke in Nan values to the overall dataset:
16.06425702811245
```

[ ]:
```
# Percentage of instances who got stroke
df['stroke'].sum()/len(df)*100
```

[ ]:
```
4.87279843444227
```

- Our main target function is stroke And the instances who got a stroke is in the minority - 249 Which is only 4.9 % of the instances

[ ]:
```
# Analysing whether to drop NA values in Bmi column
df_na=df.loc[df['bmi'].isnull()]
print("Nan BMI values where people have stroke:",df_na['stroke'].sum())
print("overall BMI values where people have stroke:",df['stroke'].sum())
```

```
Nan BMI values where people have stroke: 40
overall BMI values where people have stroke: 249
```

- Among the 201 bmi NULL values 40 values in them got stroke
- Thus we cant drop NULL values
- Since there are outliers present we can’t perform mean imputation as mean is affected by the outliers
- Hence we impute it with median values

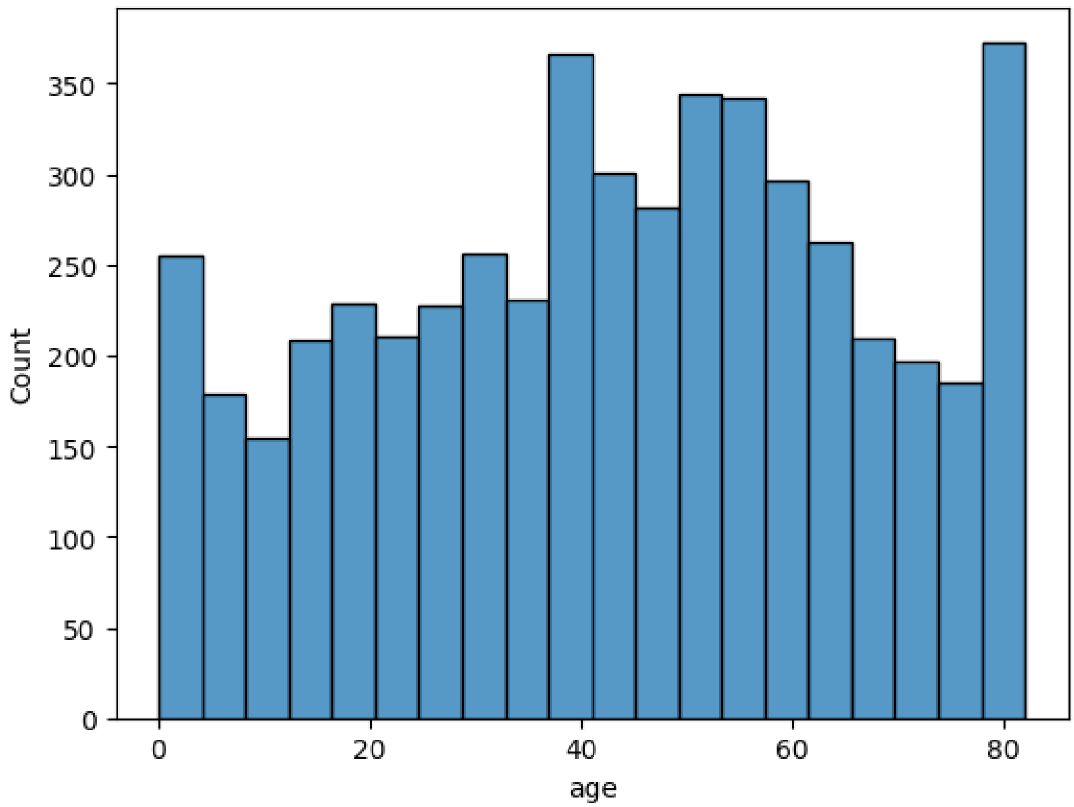
[ ]:
```
# Imputing the missing N/A values using the median of bmi column
print("median of bmi",df['bmi'].median())
df['bmi']=df['bmi'].fillna(df['bmi'].median())
```

```
median of bmi 28.1
```

0.2.6 AGE analysis

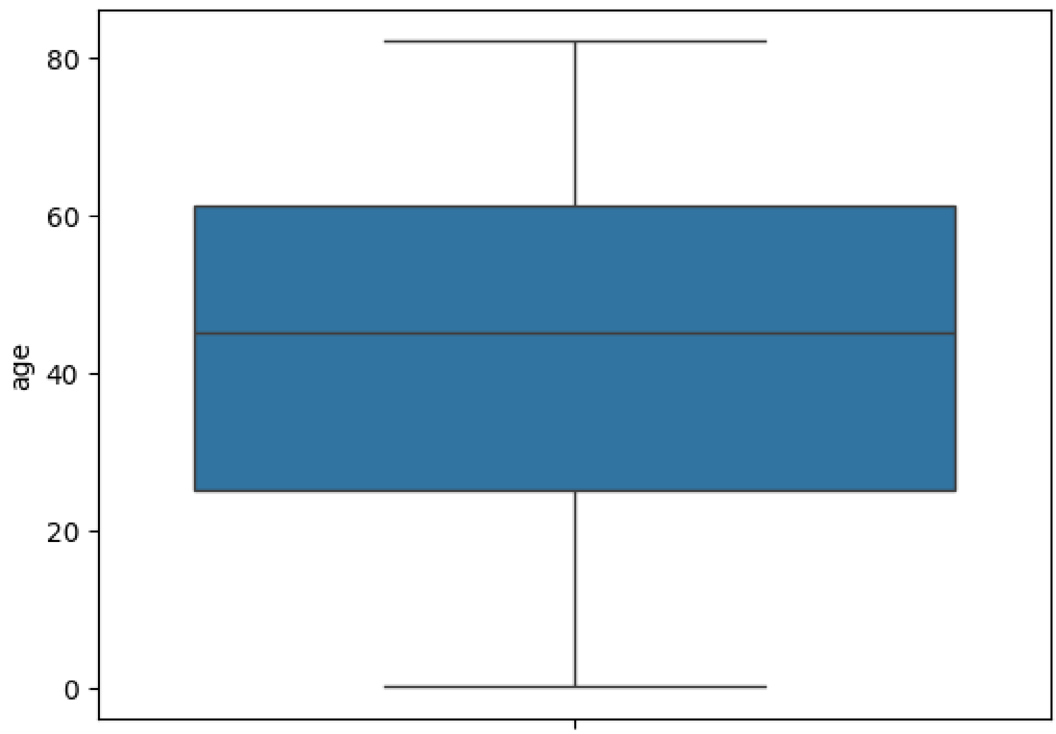
[ ]:
```
# Graphical representation fo the data in age column
# histogram
sns.histplot(data=df['age'])
```

[ ]:
```
<Axes: xlabel='age', ylabel='Count'>
```



[ ]:
```
# boxplot
sns.boxplot(data=df['age'])
```

[ ]: <Axes: ylabel='age'>

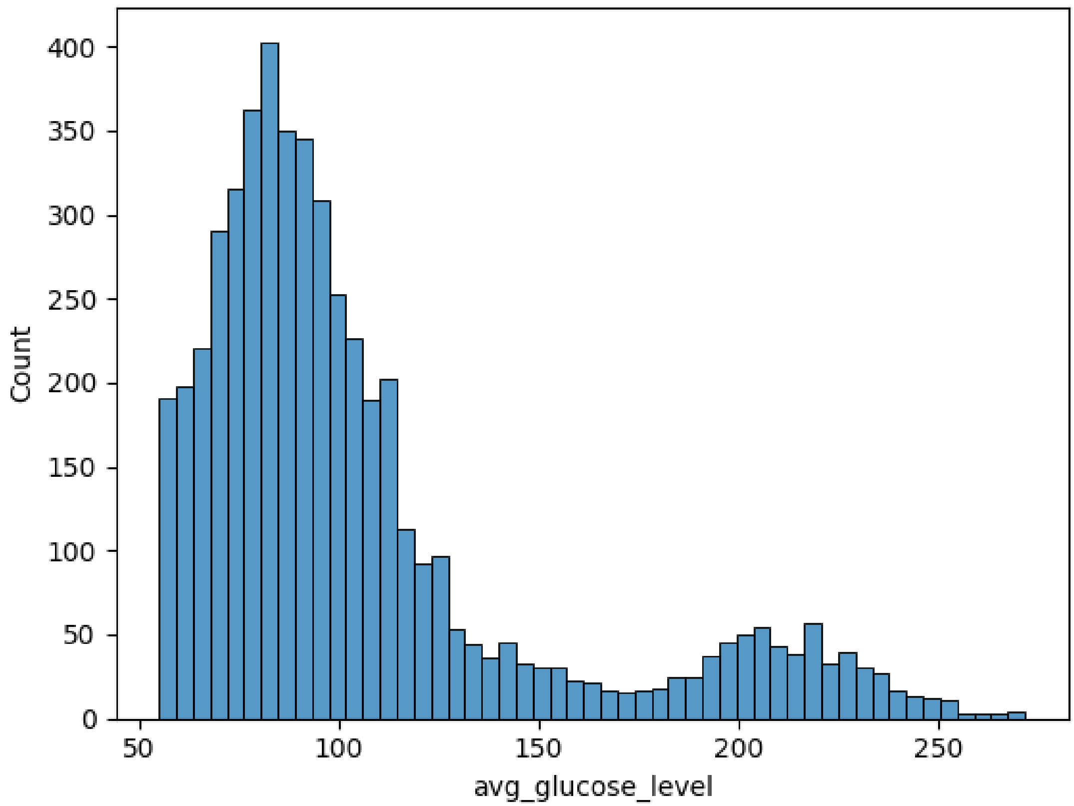


- The age parameter values does not have any outliers
- And has a normal distribution

0.2.7 AVERAGE GLUCOSE LEVEL ANALYSIS

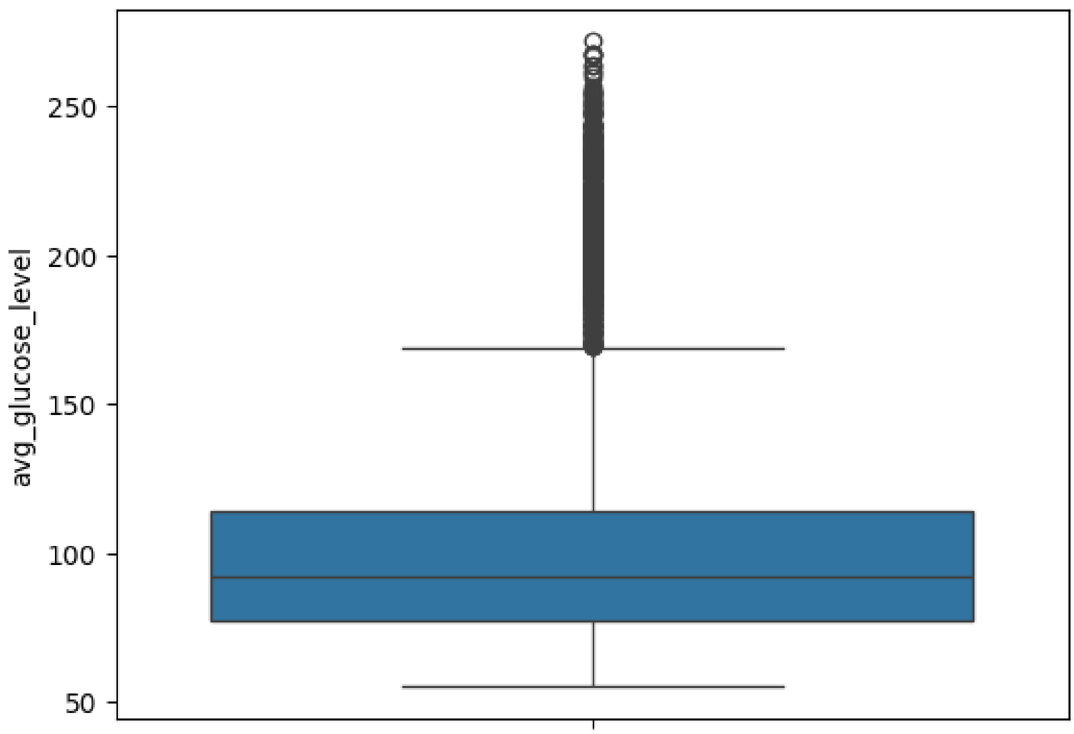
[ ]:
```
# Graphical representation fo the data in glucose level column
# histogram
sns.histplot(data=df['avg_glucose_level'])
```

[ ]:
```
<Axes: xlabel='avg_glucose_level', ylabel='Count'>
```



[ ]:
```
# Boxplot
sns.boxplot(data=df['avg_glucose_level'])
```

[ ]: <Axes: ylabel='avg_glucose_level'>



- There are many outliers present based on the boxplot and histogram
- The data is positively skewed

[ ]:
```
# Finding the count of outliers based on those instances which are out of iqr
Q1 = df['avg_glucose_level'].quantile(0.25)
Q3 = df['avg_glucose_level'].quantile(0.75)
IQR = Q3 - Q1
da=(df['avg_glucose_level'] < (Q1 - 1.5 * IQR)) | (df['avg_glucose_level'] >
  ↪(Q3 + 1.5 * IQR))
da.value_counts()
```

[ ]:
```
avg_glucose_level
False    4483
True      627
Name: count, dtype: int64
```

- Total outliers in avg_glucose_level : 627
- Total non-outliers in avg_glucose_level : 4483

0.2.8 Heart_disease analysis

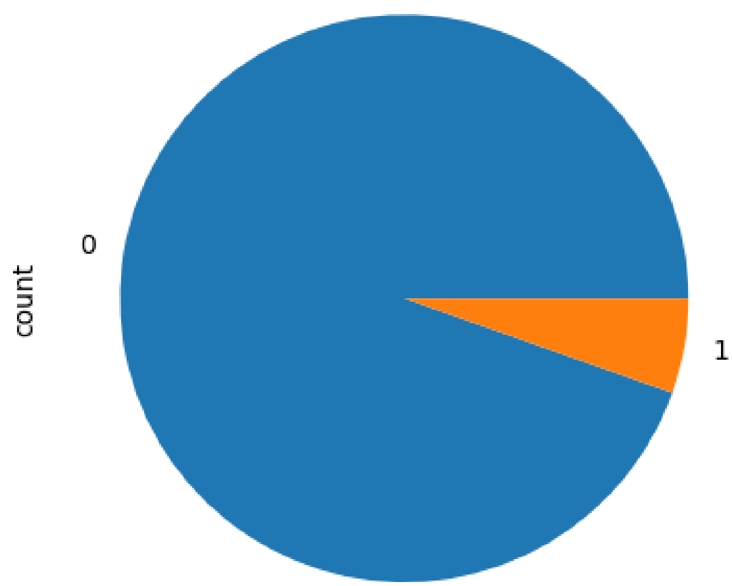
[ ]:
```
# Value count of heart disease attribute
df['heart_disease'].value_counts()
```

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

- This data reflects that around 94.5 % of the total population or list of people are free from Heart_disease and only 6.5 % are having heart_disease.

[ ]:
```
df['heart_disease'].value_counts().plot(kind="pie")
```

[ ]:
```
<Axes: ylabel='count'>
```



0.2.9 Ever_married analysis with Values

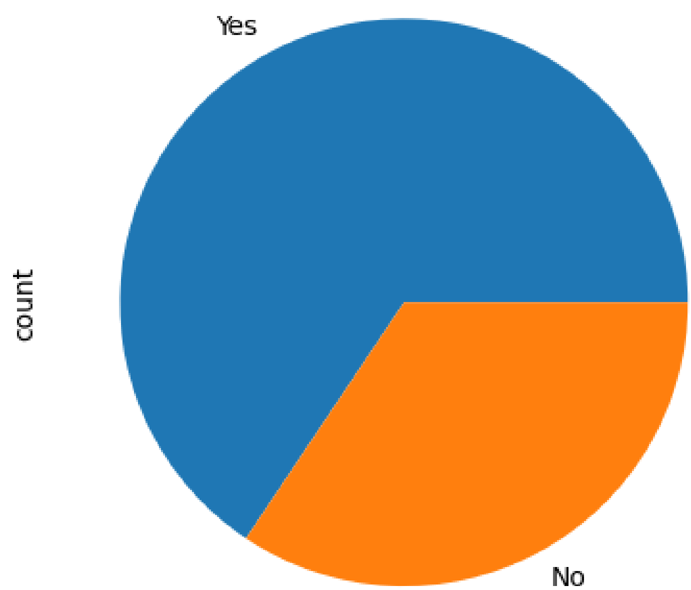
[ ]:
```
# Value count of evver married attribute
df['ever_married'].value_counts()
```

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

- This result shows that 65.6 % of people from the list are married and 34.4 % are unmarried.

[ ]:
```
# Graphical representation
df['ever_married'].value_counts().plot(kind="pie")
```

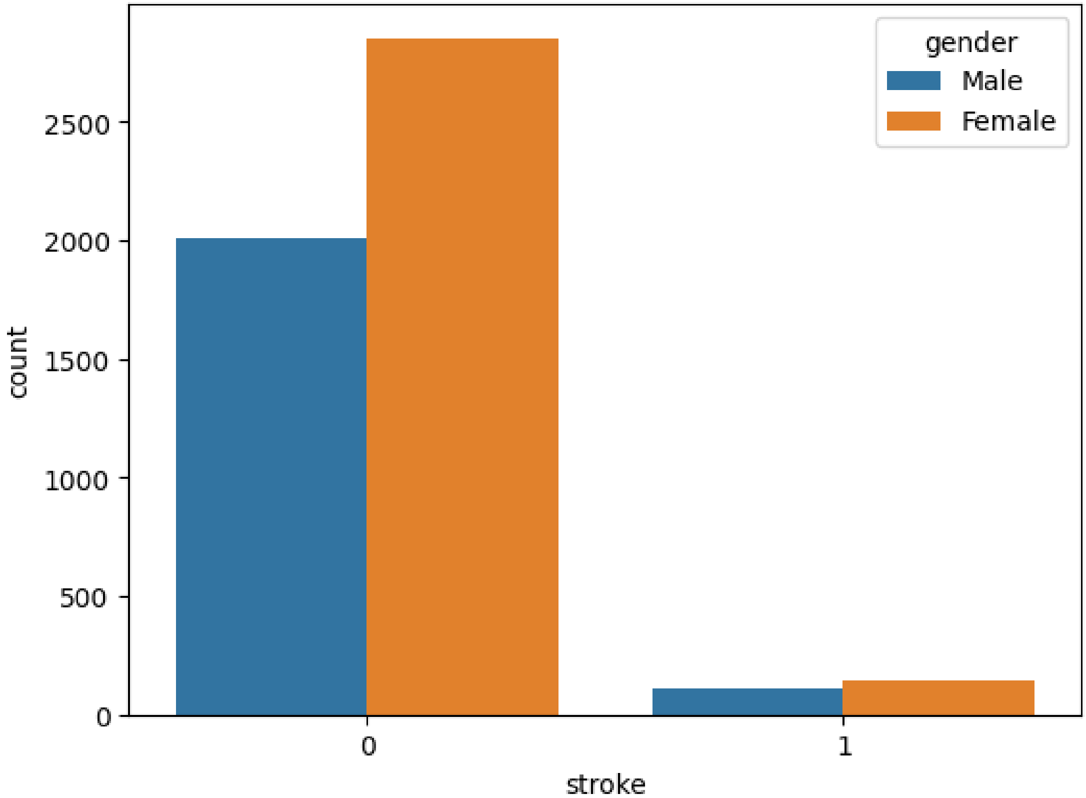
[ ]:
```
<Axes: ylabel='count'>
```



0.3 Cross analysis - all the attribute compared with target attibute

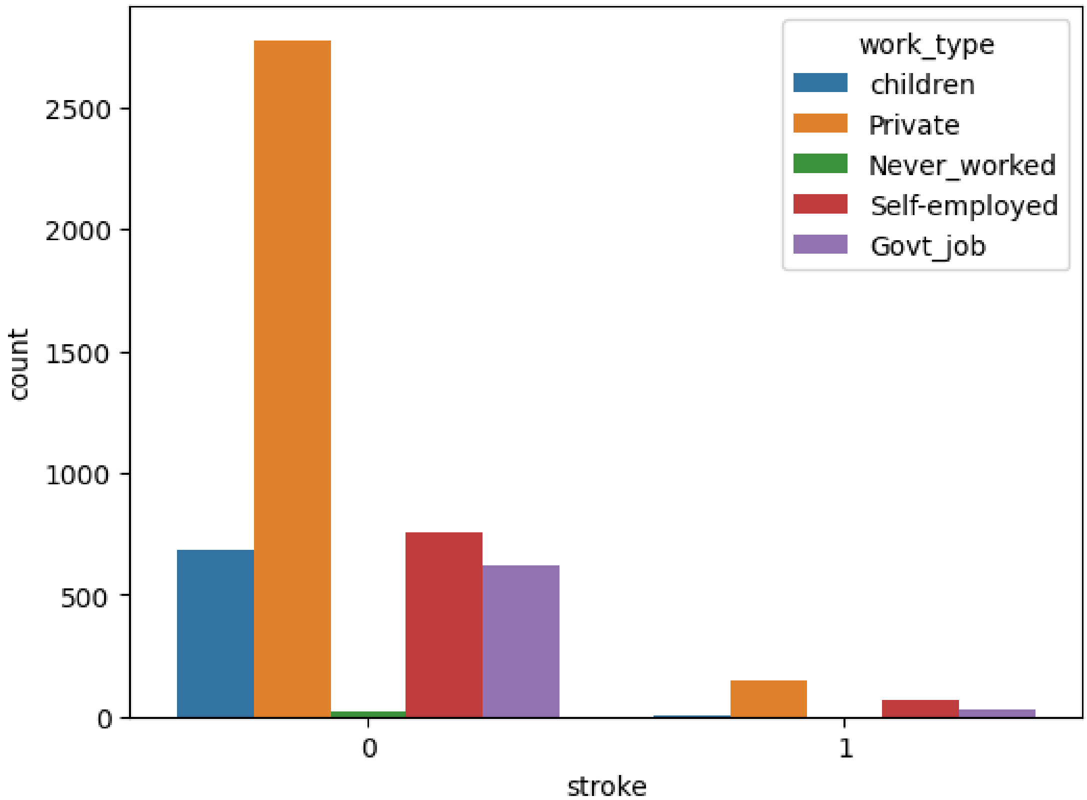
[ ]:
```
# Comparing stroke with gender
sns.countplot(x='stroke', hue='gender', data=df)
```

[ ]:
```
<Axes: xlabel='stroke', ylabel='count'>
```




```
[ ]: # Comparing stroke with work-type
     sns.countplot(x='stroke', hue='work_type', data=df)
```

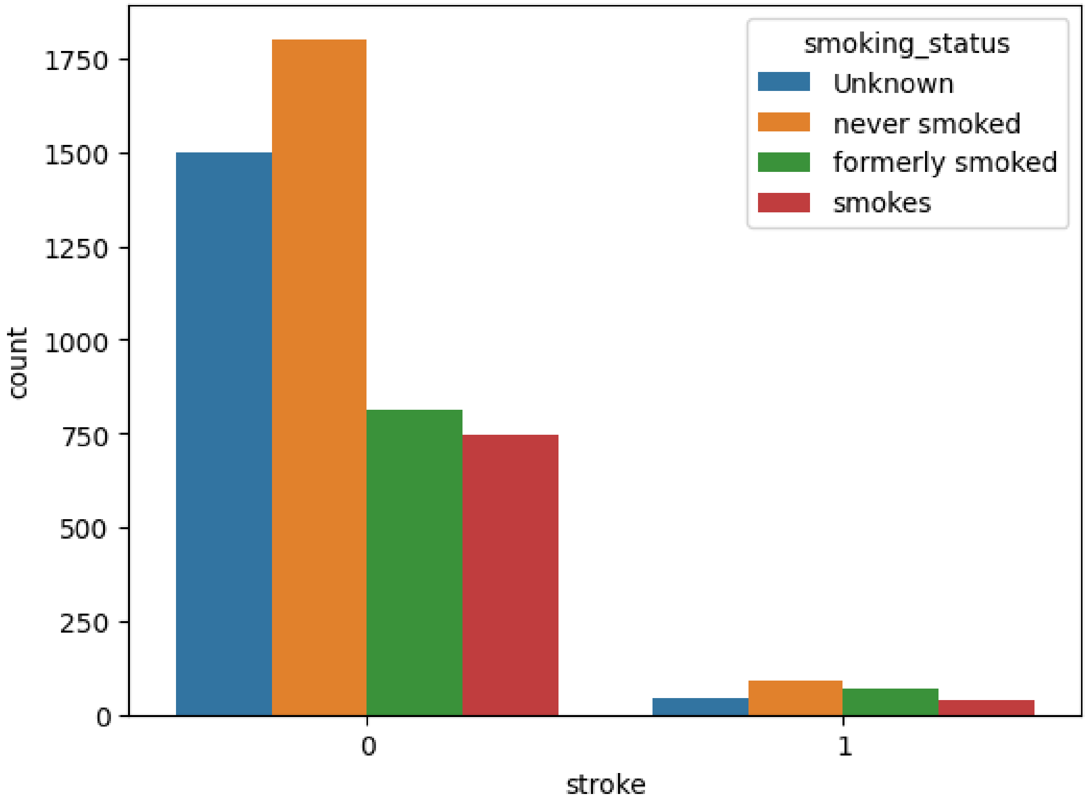
```
[ ]: <Axes: xlabel='stroke', ylabel='count'>
```



- Based on this comparison we see in the provided dataset that people who never worked never got a heart attack and the people who are privetly employed got more strokes

[ ]:
```
# Comparing stroke with somking_status
sns.countplot(x='stroke', hue='smoking_status', data=df)
```

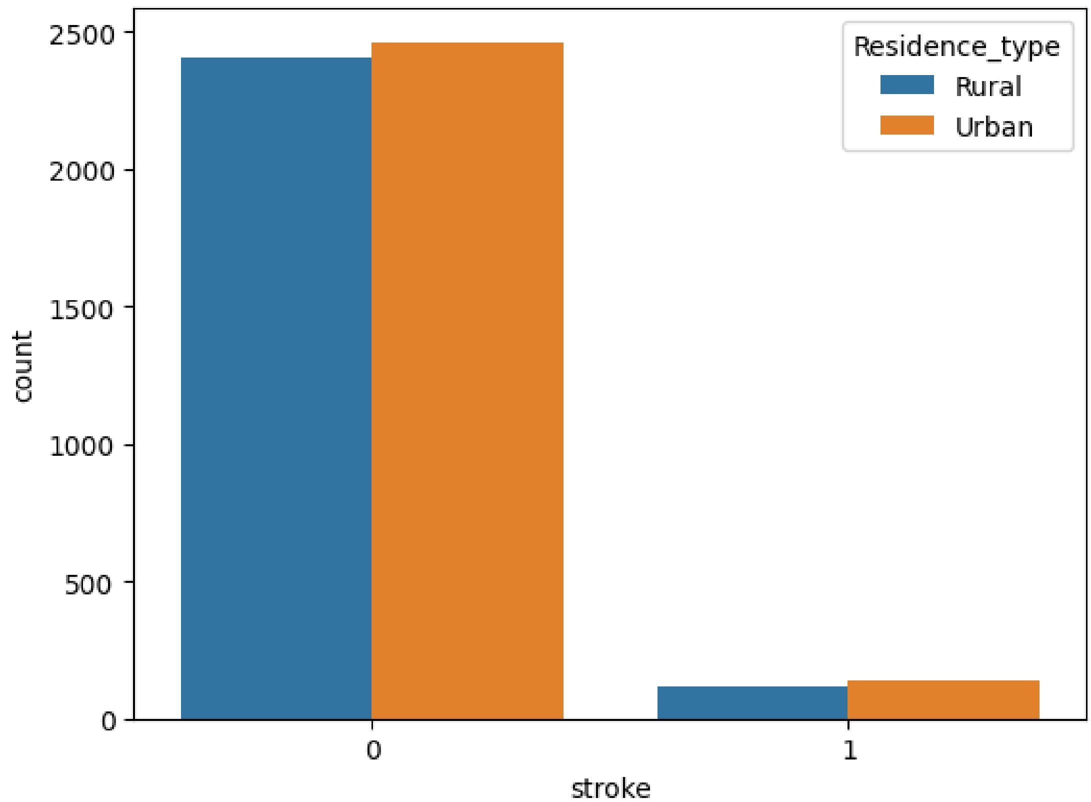
[ ]:
```
<Axes: xlabel='stroke', ylabel='count'>
```



- Based on the plot we can that those who formerly smoked got more strokes The people who smoked and never smoked has a somewhat same probability of getting stroke

[ ]:
```
# Comparing stroke with residence type
sns.countplot(x='stroke', hue='Residence_type', data=df)
```

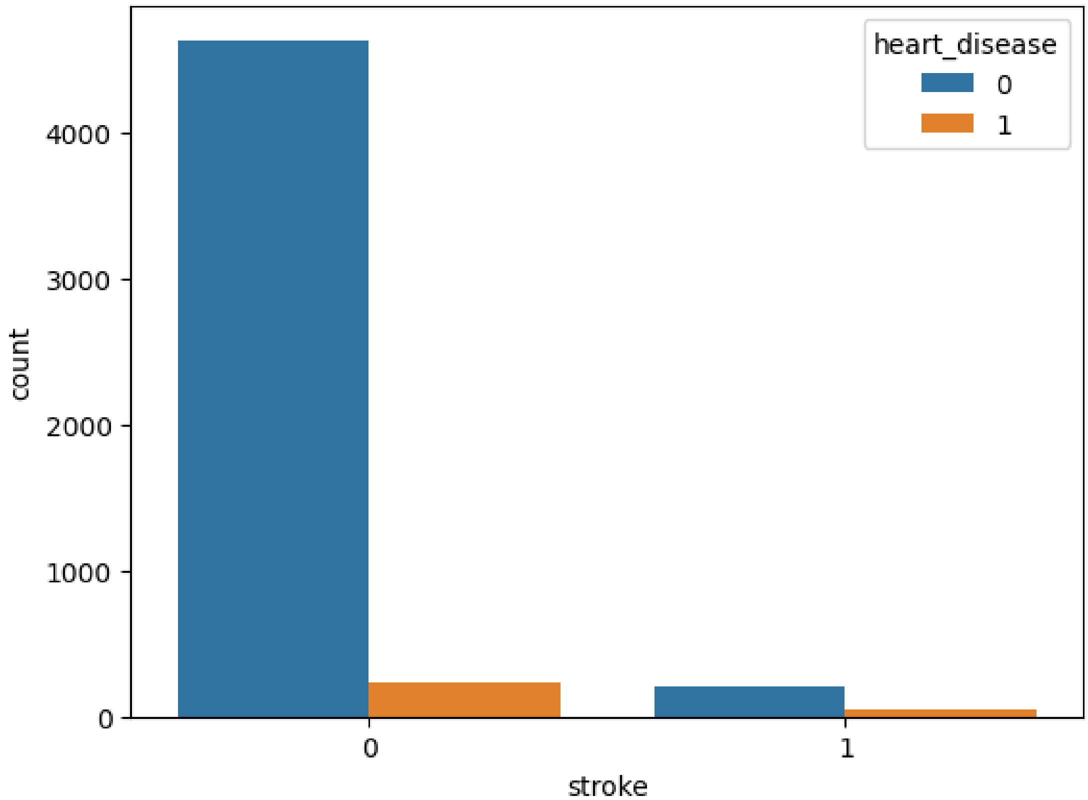
[ ]:
```
<Axes: xlabel='stroke', ylabel='count'>
```



- Based on the analysis the people who live in Urban areas were reported with more strokes

[ ]:
```
# Comparing stroke with heart disease
sns.countplot(x='stroke', hue='heart_disease', data=df)
```

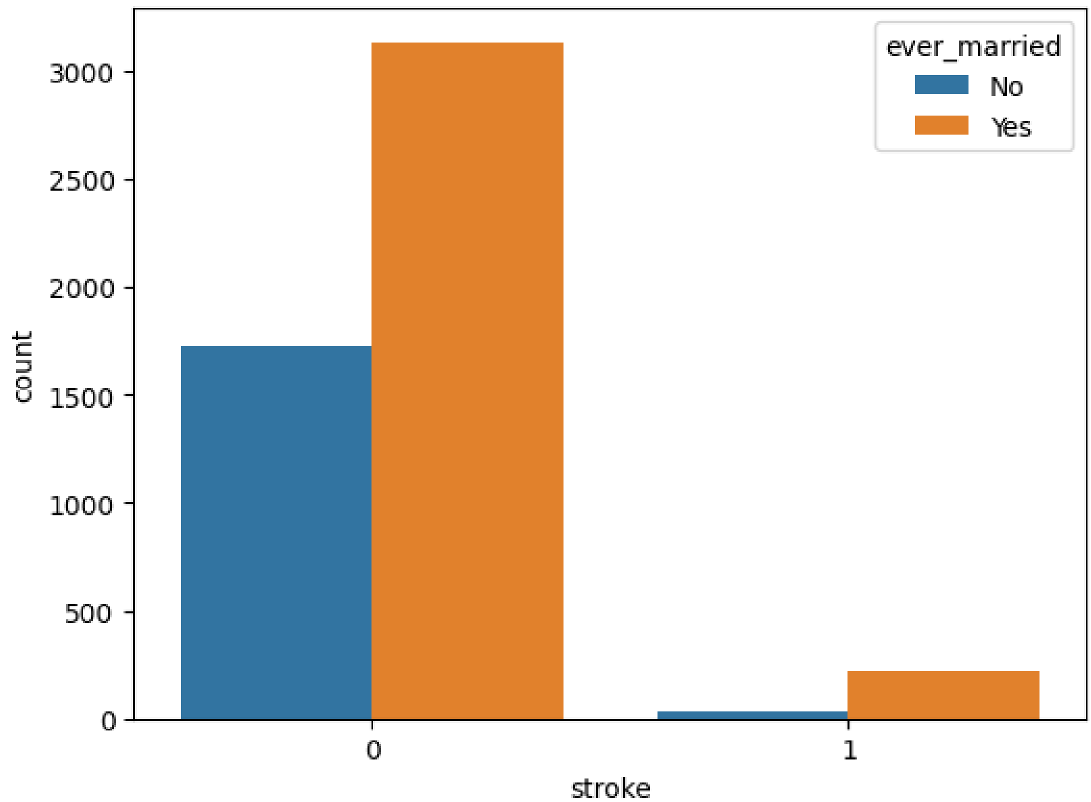
[ ]:
```
<Axes: xlabel='stroke', ylabel='count'>
```



- This plotting shows that the number of “people with Strokes but no heart disease” is approximately 6 to 8 times the number of “people with Strokes and also heart disease”. This shows most of the people with no heart disease are suffering with Strokes compared to the once who have Heart Disease.

[ ]:
```
# Comparing stroke with married status
sns.countplot(x='stroke', hue='ever_married', data=df)
```

[ ]: `<Axes: xlabel='stroke', ylabel='count'>`



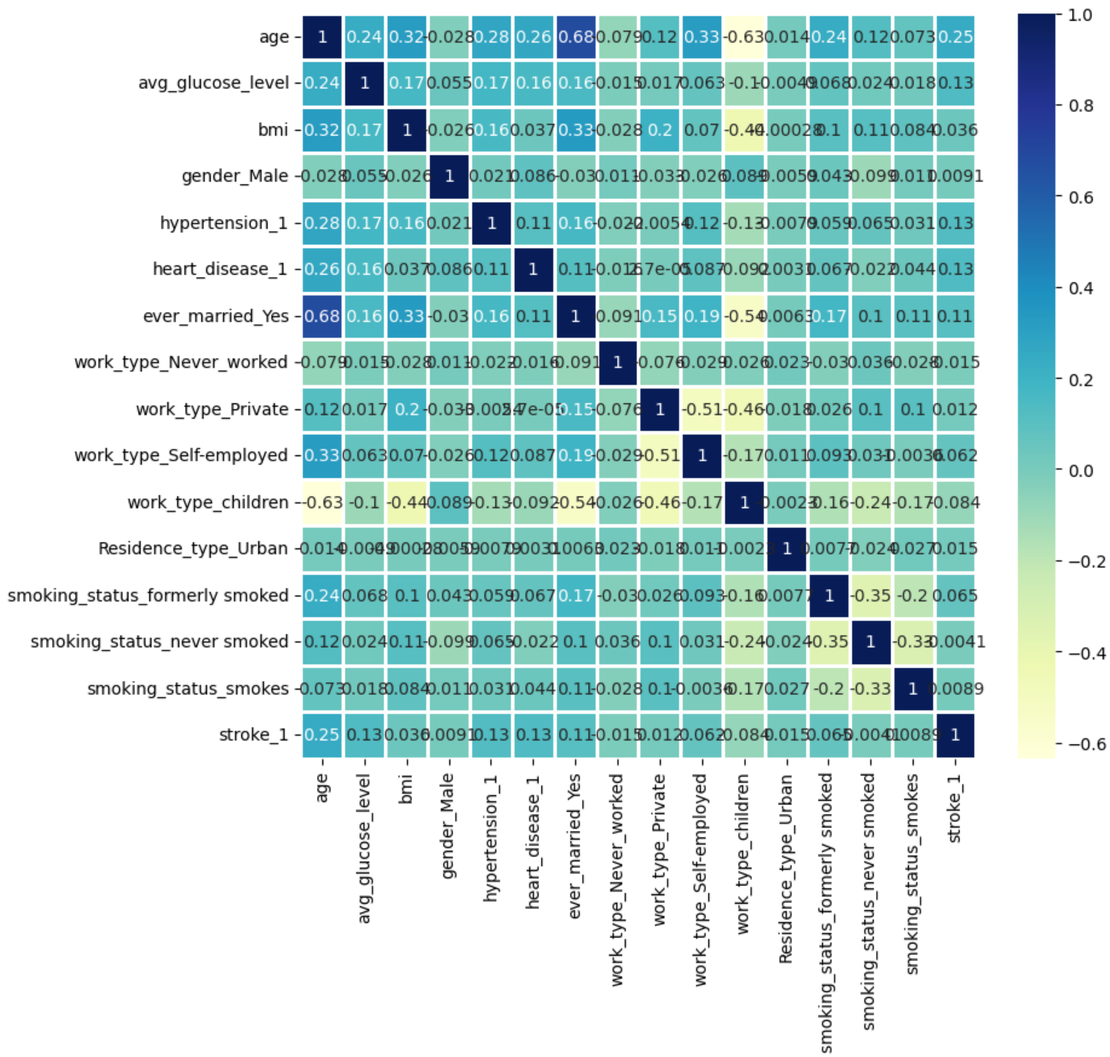
- This plotting shows that the number of “Married people with Strokes” is approximately 10 to 12 times the no. “Unmarried people with Strokes”.
- This shows most of the Married people got Strokes compared to Unmarried people.

0.3.1 Creating dummy variables for numeric-binary attributes

[ ]:
```
# Converting numeric-binary value attributes to string
df[['hypertension', 'heart_disease', 'stroke']] = df[['hypertension',␣
↪'heart_disease', 'stroke']].astype(str)
# Generating dummy attributes - one hot encoding format
df = pd.get_dummies(df, drop_first= True)
```

[ ]:
```
# Correlation matrix between the attributes in the dataset to find if any␣
↪attributes are correlated
corrmat=df.corr()
f,ax=plt.subplots(figsize=(9,8))
sns.heatmap(corrmat,ax=ax,cmap="YlGnBu",linewidth=0.8,annot=True)
```

[ ]: <Axes: >



[ ]:
```
# The data frame after performing dummy attributes
df.head()
```

[ ]:
```
    age  avg_glucose_level   bmi  gender_Male  hypertension_1  \
0  67.0             228.69  36.6         True           False
1  61.0             202.21  28.1        False           False
2  80.0             105.92  32.5         True           False
3  49.0             171.23  34.4        False           False
4  79.0             174.12  24.0        False            True

   heart_disease_1  ever_married_Yes  work_type_Never_worked  \
0             True              True                   False
1            False              True                   False
2             True              True                   False
```

```
3           False              True                   False
4           False              True                   False

   work_type_Private  work_type_Self-employed  work_type_children  \
0               True                    False               False
1              False                     True               False
2               True                    False               False
3               True                    False               False
4              False                     True               False

   Residence_type_Urban  smoking_status_formerly smoked  \
0                  True                            True
1                 False                           False
2                 False                           False
3                  True                           False
4                 False                           False

   smoking_status_never smoked  smoking_status_smokes  stroke_1
0                        False                  False      True
1                         True                  False      True
2                         True                  False      True
3                        False                   True      True
4                         True                  False      True
```

[ ]:
```
# Since our Dataset is highly undersampled (based on target instances) we are␣
↪going to perform a over sampling method to have equal representation of both␣
↪the target classes
# Using random oversampling - importing the library
from imblearn.over_sampling import RandomOverSampler

# Performing a minority oversampling
oversample = RandomOverSampler(sampling_strategy='minority')
X=df.drop(['stroke_1'],axis=1)
y=df['stroke_1']

# Obtaining the oversampled dataframes - testing and training
X_over, y_over = oversample.fit_resample(X, y)
```

[ ]:
```
# importing a scaling modeule
from sklearn.preprocessing import StandardScaler

# Since the numeric attributes in the dataset is in different ranges and three␣
↪are outliers persent we are usign a scaler to get all the values into the␣
↪same range.
s = StandardScaler()
# Scaling the numeric attributes
```

```
df[['bmi', 'avg_glucose_level', 'age']] = s.fit_transform(df[['bmi',␣
↪'avg_glucose_level', 'age']])
```

- Scaling the numeric values for bringing them all to the same scale

0.3.2 Creating test-train split (80-20 split)

[ ]:
```
# creating dataset split for training and testing the model
from sklearn.model_selection import train_test_split
# Performing a 80-20 test-train split
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, test_size=␣
↪0.20, random_state= 42)
```

[ ]:
```
# Checking the size of the splits
print('X_train:', X_train.shape)
print('y_train:', y_train.shape)
print('X_test:', X_test.shape)
print('y_test:', y_test.shape)
```

```
X_train: (7777, 15)
y_train: (7777,)
X_test: (1945, 15)
y_test: (1945,)
```

0.4 Training Model

0.4.1 Decision Tree

[ ]:
```
#importing the Decision Tree Classifier module
from sklearn.tree import DecisionTreeClassifier
# Libraries for calculating performance metrics
from sklearn import metrics
from sklearn.metrics import␣
↪auc,roc_auc_score,roc_curve,precision_score,recall_score,f1_score

# Create the classifier object
clf = DecisionTreeClassifier()

# Training the classifier
clf = clf.fit(X_train,y_train)

#predicting result using the test dataset
y_pred = clf.predict(X_test)

# Printing the accuracyof the model
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9778920308483291
```

0.4.2 KNN

[ ]:
```
#importing the KNN Classifier module
from sklearn.neighbors import KNeighborsClassifier
# Libraries for calculating performance metrics
from sklearn.metrics import␣
 ↪classification_report,accuracy_score,confusion_matrix
from sklearn.metrics import␣
 ↪auc,roc_auc_score,roc_curve,precision_score,recall_score,f1_score

# Create the classifier object
# 2 neighbours because of the 2 classes
knn = KNeighborsClassifier(n_neighbors = 2)
# Training the classifier
knn.fit(X_train,y_train)
#predicting result using the test dataset
y_pred_knn = knn.predict(X_test)
y_pred_prob_knn = knn.predict_proba(X_test)[:, 1]

# Printing the accuracy and roc-auc score of the model
confusion_matrix(y_test, y_pred_knn)
print('Accuracy:',accuracy_score(y_test, y_pred_knn))
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_knn))
```

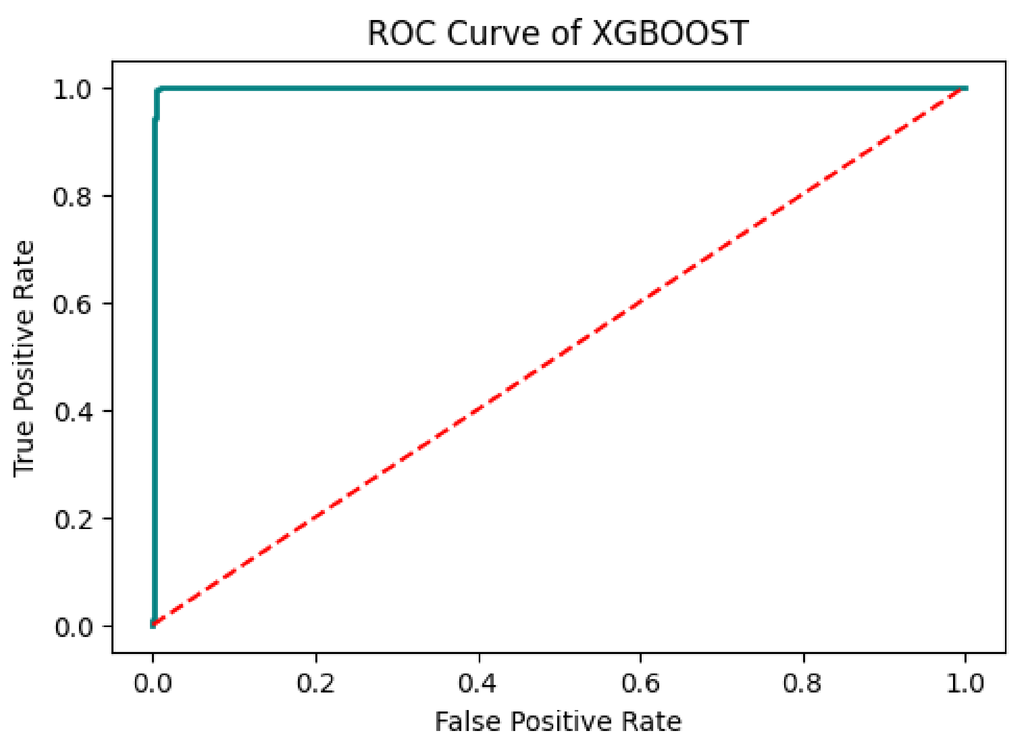
```
Accuracy: 0.9722365038560411
ROC AUC Score: 0.9723076923076923
```

0.4.3 XGBoost

[ ]:
```
#importing the XGBoost Classifier module
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score,␣
 ↪roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier

# Create the classifier object
xgb = XGBClassifier()

# Training the classifier
xgb.fit(X_train, y_train)

# Predicting result using the test dataset
y_pred_xgb = xgb.predict(X_test)
y_pred_prob_xgb = xgb.predict_proba(X_test)[:, 1]

# Printing the accuracy and roc-auc score of the model
print('Accuracy:', accuracy_score(y_test, y_pred_xgb))
```

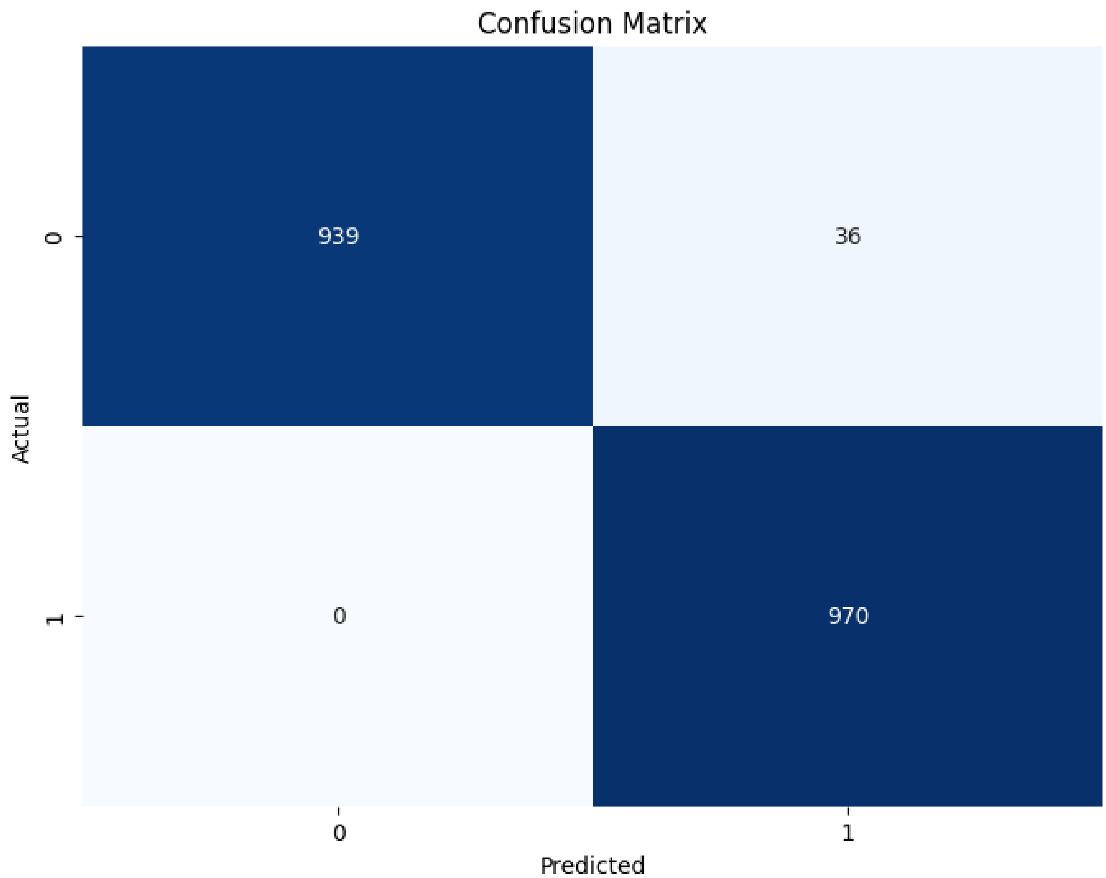
```
print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_xgb))

# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_xgb)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, linewidth=2, color='teal')
plt.plot([0, 1], [0, 1], 'r--')
plt.title('ROC Curve of XGBOOST')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
Accuracy: 0.9814910025706941
ROC AUC Score: 0.9988146973301613
```



[ ]:
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_xgb)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
[ ]: # Printing the precision,recall,f1score and support values of the model based␣
     ↪on the confusion matrix
     from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
     print("Accuracy_score:",accuracy_score(y_test,y_pred_xgb))
     print("Precision_score:",precision_score(y_test,y_pred_xgb))
     print("Recall_score:",recall_score(y_test,y_pred_xgb))
     print("f1_score:",f1_score(y_test,y_pred_xgb))
     print('ROC AUC Score:', roc_auc_score(y_test, y_pred_prob_xgb))
```

```
Accuracy_score: 0.9814910025706941
Precision_score: 0.9642147117296223
Recall_score: 1.0
f1_score: 0.9817813765182186
ROC AUC Score: 0.9988146973301613
```

0.4.4 Random Forest

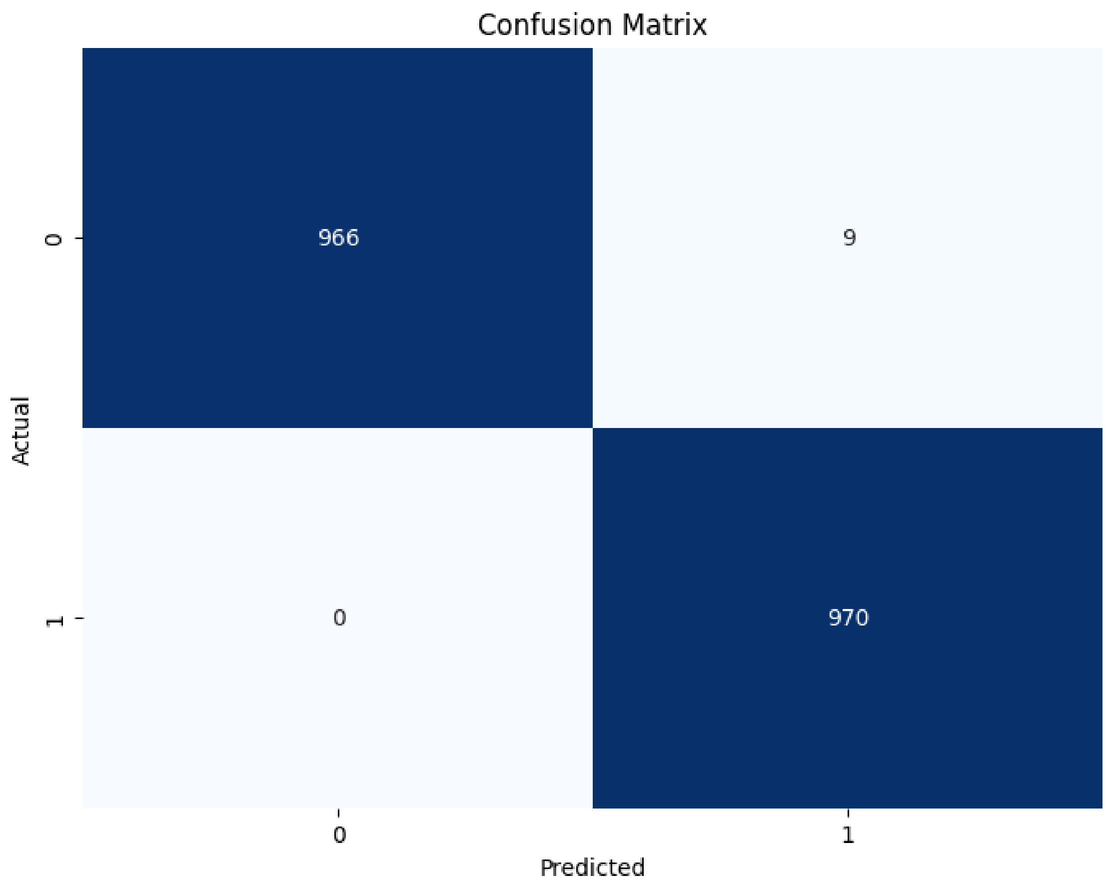
[ ]:
```
# importing random forest classifier module for training
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Create the classifier object
rf_clf = RandomForestClassifier(n_estimators = 100)

# Train the model using the training sets
rf_clf.fit(X_train, y_train)

# performing predictions on the test dataset
y_pred_rf = rf_clf.predict(X_test)

# Printing accuracy of the model
print('Accuracy:', accuracy_score(y_test, y_pred_rf))
```

```
Accuracy: 0.9953727506426735
```

[ ]:
```
# Importing module for kfold cross validation
from sklearn import model_selection
from sklearn.model_selection import KFold

# Performing k fold cross validation using 20 splits
kfold_kridge = model_selection.KFold(n_splits=20, shuffle=True)
results_kfold = model_selection.cross_val_score(rf_clf, X_over, y_over,
  ↪cv=kfold_kridge)
print("Accuracy: ", results_kfold.mean()*100)
print(results_kfold)
```

```
Accuracy:  99.35206310577064
[0.98973306 0.98973306 0.99176955 0.99382716 0.99176955 0.99588477
 0.99176955 0.99382716 0.99382716 0.98765432 0.99382716 0.99794239
 0.99588477 0.99382716 0.99382716 0.99176955 0.99176955 0.99794239
 0.99588477 0.99794239]
```

[ ]:
```
# Plotting the confusion matrix
from sklearn.metrics import confusion_matrix,precision_recall_fscore_support
cm2 = confusion_matrix(y_test,y_pred_rf)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm2, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



0.4.5 Logistic regression

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

classifier = LogisticRegression(random_state = 0, max_iter=1000)
classifier.fit(X_train, y_train)

y_pred_lr = classifier.predict(X_test)

confusion_matrix(y_test, y_pred_lr)
print('Accuracy:', accuracy_score(y_test, y_pred_lr))
```

```
Accuracy: 0.7717223650385604
```

[ ]:
```
# Making sample predictions based on manual value entry
age=75
avg_glucose_level=300
bmi=36.6
gender_Male=1
```

```
ever_married_Yes=1
work_type_Never_worked=0
work_type_Private=1
work_type_Self_employed=0
work_type_children=0
Residence_type_Urban=1
smoking_status_formerly_smoked=1
smoking_status_never_smoked=0
smoking_status_smokes=0
hypertension_1=1
heart_disease_1=1
input_features =␣
↪[age        ,avg_glucose_level,        bmi        ,gender_Male,hypertension_1,        heart

features_value = [np.array(input_features)]
features_name =␣
↪['age'        ,'avg_glucose_level',        'bmi'        ,'gender_Male'        ,'hypertensio
↪smoked','smoking_status_never smoked'        ,'smoking_status_smokes']

df = pd.DataFrame(features_value, columns=features_name)
prediction = rf_clf.predict(df)[0]
print(prediction)
```

```
True
```

[ ]:
```
# For the front end
import pickle

with open('model.pickle','wb') as f:
  pickle.dump(rf_clf,f)
```