In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore' )
import plotly.express as px
```

In [2]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,roc_auc
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
import scikitplot as skplt
```

In [3]:
```
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree  import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostC
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

In [4]:
```
sns.set_style('darkgrid')
plt.rcParams['figure.figsize']=(15,8)
plt.rcParams['font.size']=18
```

In [5]:
```
# importing data frame
df = pd.read_csv("stroke.csv")
```

In [6]:
```
# top 5 row of df
df.head()
```

Out[6]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12

In [7]:
```
# shape of dataframe
df.shape
```

Out[7]:
```
(5110, 12)
```

In [8]:
```
# columns of df
df.columns
```

Out[8]:
```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
```

```
       'smoking_status', 'stroke'],
      dtype='object')
```

In [9]:
```
# basic information
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
```

In [10]:
```
#duplicated rows
df.duplicated().sum()
```

Out[10]:
```
0
```

observation

1. No duplicated row present in df

In [11]:
```
# null values:
df.isnull().sum()
```

Out[11]:
```
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
```

1. only BMI columns has null values

In [12]:
```
#describition of df
df.describe()
```

Out[12]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320

min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

In [13]:
```
# droping unwanted columns:
df.drop("id",1,inplace=True)
```

In [14]:
```
# no of unique values:
df.nunique()
```

Out[14]:
```
gender                  3
age                   104
hypertension            2
heart_disease           2
ever_married            2
work_type               5
Residence_type          2
avg_glucose_level    3979
bmi                   418
smoking_status          4
stroke                  2
dtype: int64
```

In [15]:
```
# printing unique values:
for i in df.columns:
        print(df[i].value_counts())
        print("------------------------------------------------------------------------")
```

```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
------------------------------------------------------------------------
78.00    102
57.00     95
52.00     90
54.00     87
51.00     86
        ...
1.40       3
0.48       3
0.16       3
0.40       2
0.08       2
Name: age, Length: 104, dtype: int64
------------------------------------------------------------------------
0    4612
1     498
Name: hypertension, dtype: int64
------------------------------------------------------------------------
0    4834
1     276
Name: heart_disease, dtype: int64
------------------------------------------------------------------------
Yes    3353
No     1757
Name: ever_married, dtype: int64
------------------------------------------------------------------------
Private          2925
```

```
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
-----------------------------------------------------------------------
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
-----------------------------------------------------------------------
93.88     6
91.68     5
91.85     5
83.16     5
73.00     5
         ..
111.93    1
94.40     1
95.57     1
66.29     1
85.28     1
Name: avg_glucose_level, Length: 3979, dtype: int64
-----------------------------------------------------------------------
28.7    41
28.4    38
26.7    37
27.6    37
26.1    37
        ..
48.7     1
49.2     1
51.0     1
49.4     1
14.9     1
Name: bmi, Length: 418, dtype: int64
-----------------------------------------------------------------------
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
-----------------------------------------------------------------------
0    4861
1     249
Name: stroke, dtype: int64
-----------------------------------------------------------------------
```

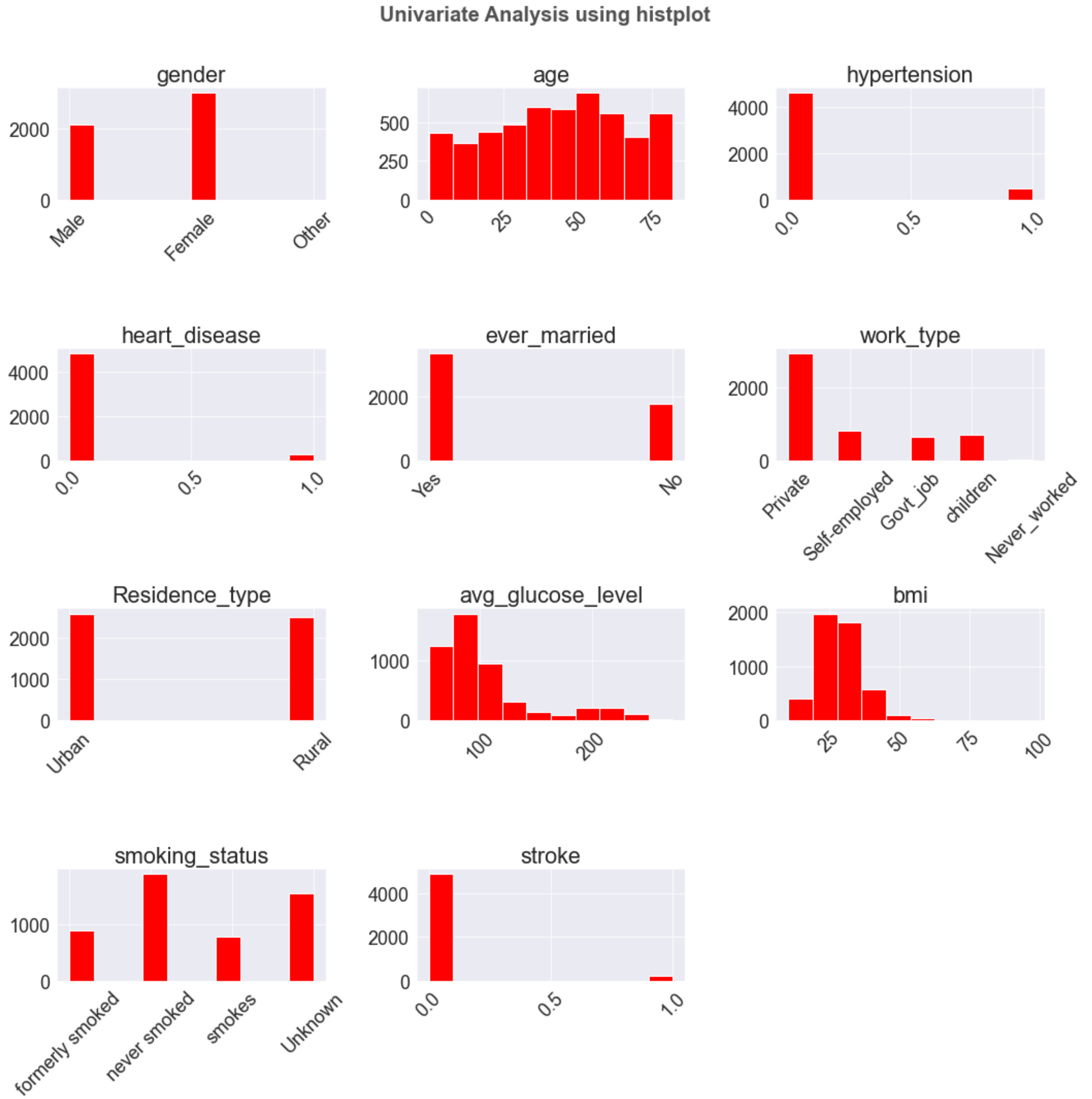
In [16]:
```python
# numerical columns
num_cols = df.select_dtypes(include=['int','float']).columns
# categorical columns:
cat_cols = df.select_dtypes(include=['object']).columns
```

In [17]:
```python
num_cols
```

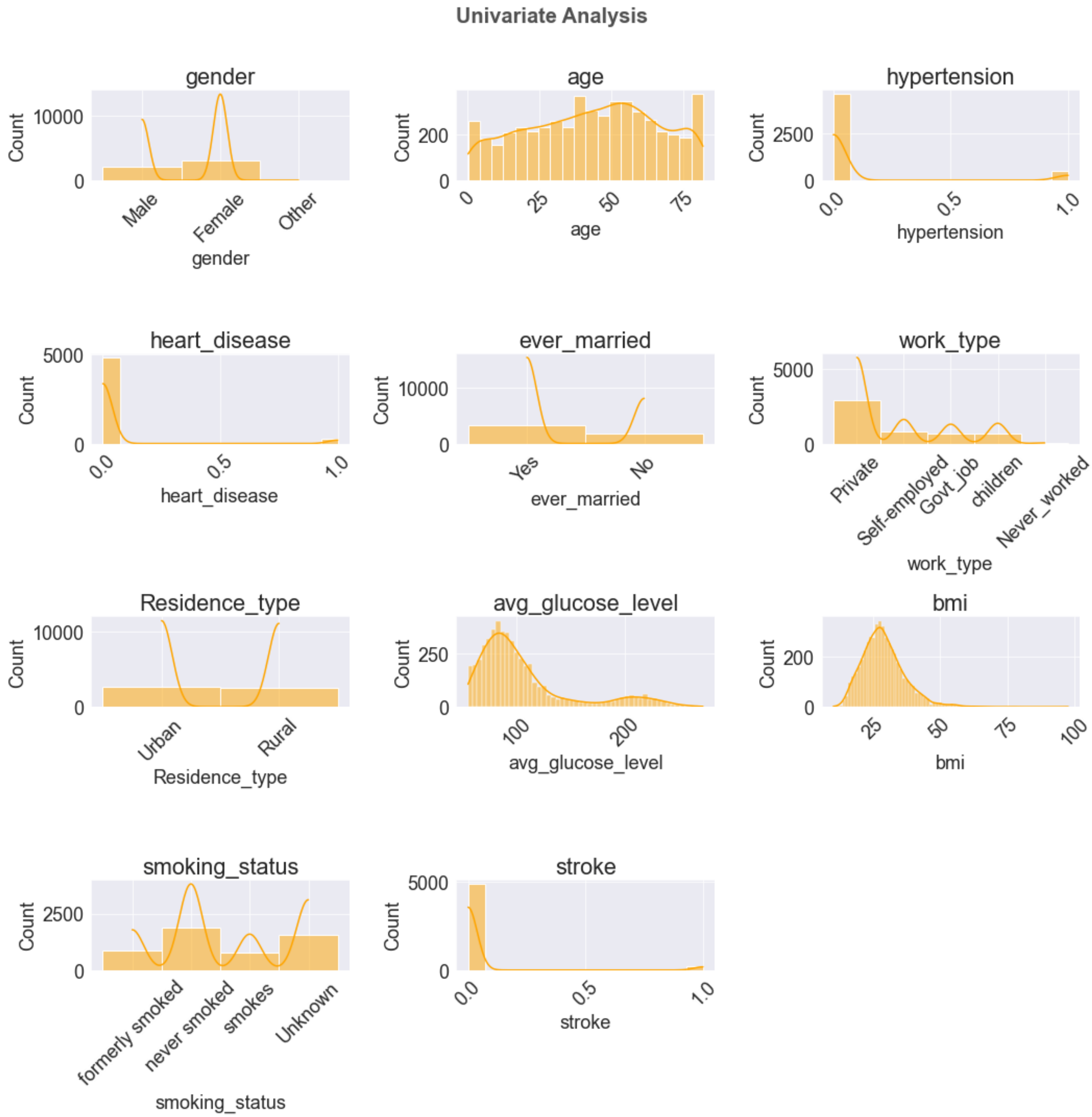
Out[17]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke'],
      dtype='object')
```

In [18]:
```python
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis using histplot', fontsize=20, fontweight='bold', alpha
for j,i in enumerate(df):
    plt.subplot(4,3,j+1)
    plt.title(i)
    plt.xticks(rotation=45)
    plt.hist(df[i],color='red')
```

```
plt.tight_layout()
plt.show()
```



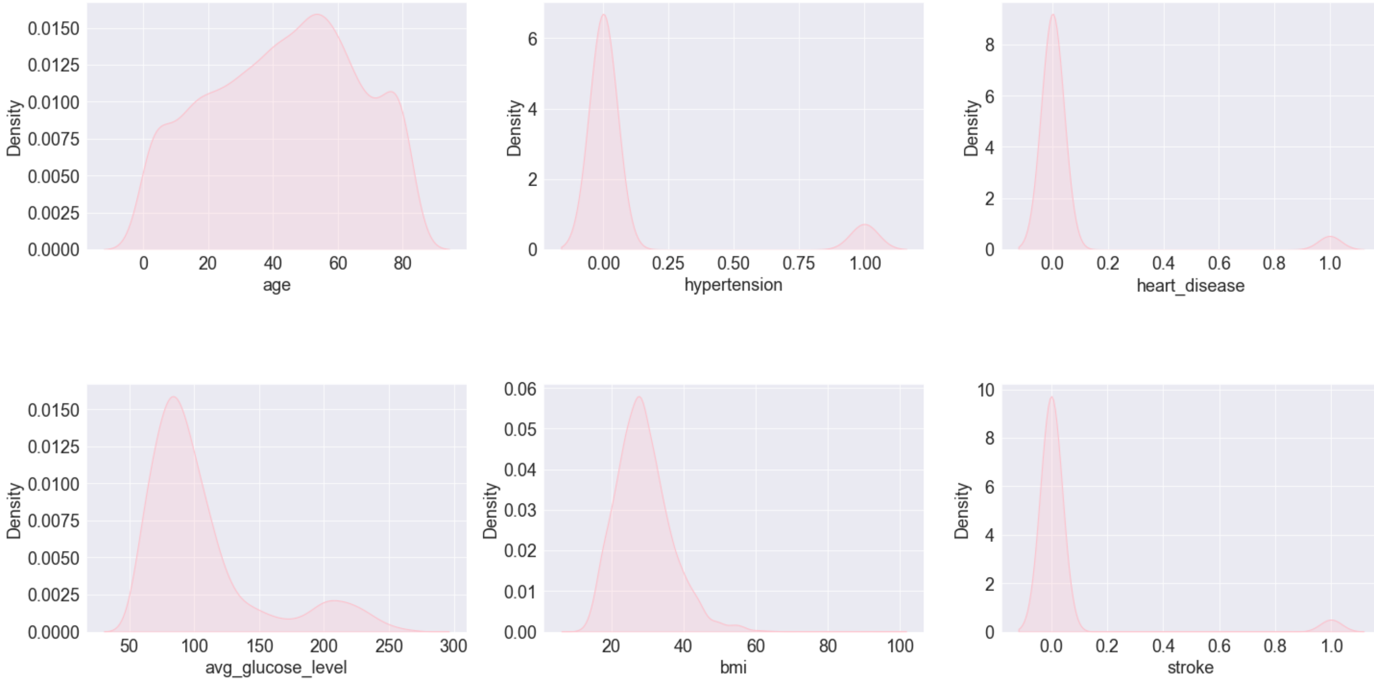
In [19]:
```
plt.figure(figsize=(15,15) )
plt.suptitle('Univariate Analysis', fontsize=20, fontweight='bold', alpha=0.8, y=1.)
for j,i in enumerate(df):
    plt.subplot(4,3,j+1)
    plt.title(i)
    plt.xticks(rotation=45)
    sns.histplot(df[i],color='orange',kde=True)

plt.tight_layout()
plt.show()
```



In [20]:

```
plt.suptitle('Univariate Analysis of Numerical Features using KDE Plot', fontsize=20, fo
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col in num_cols:
        sns.kdeplot(x=col, data=df, ax=ax[index],shade=True,color="Pink")
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

<Figure size 1080x576 with 0 Axes>



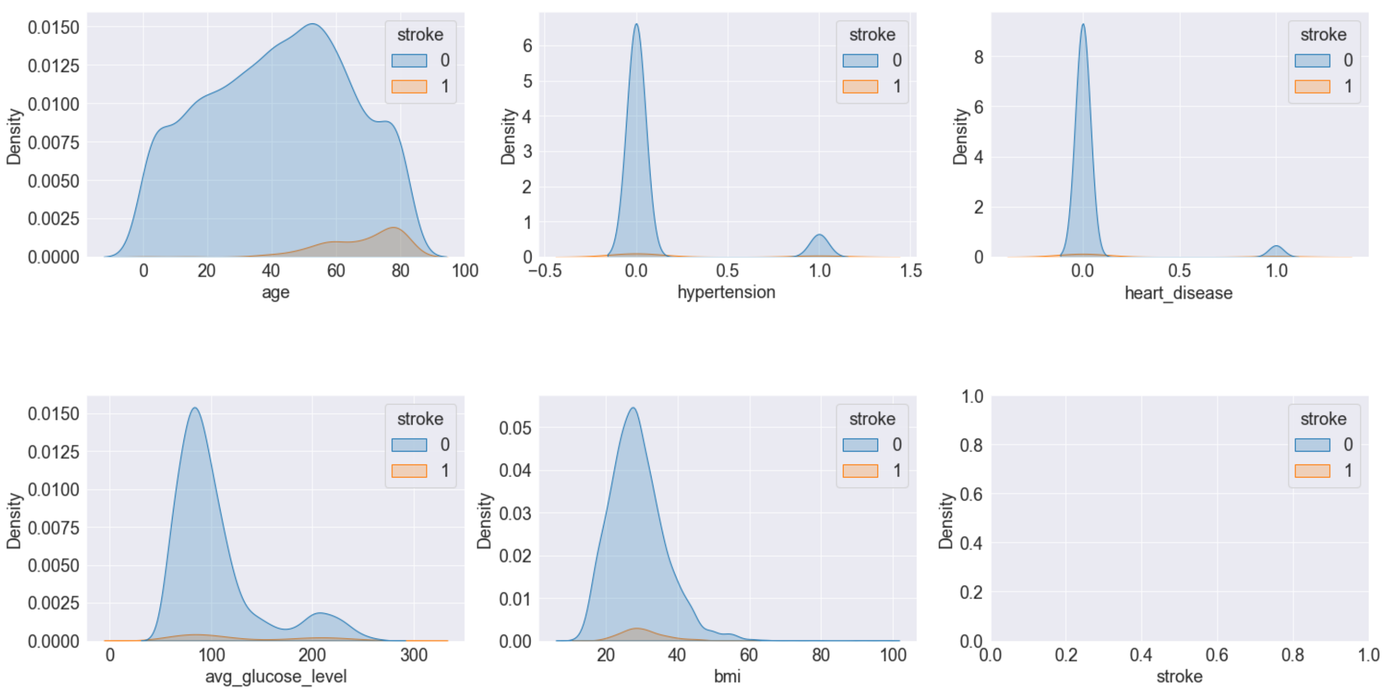

```
In [ ]:
```

observation

1. hypertension , heart_disease and stroke are left skewed
2. avg_glucose_level and BMI is normal distributed

In [21]:
```
plt.suptitle('Univariate Analysis of Numerical Features using KDE Plot', fontsize=20, fo
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col in num_cols:
        sns.kdeplot(x=col, data=df, ax=ax[index],shade=True,color="Pink",hue='stroke')
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

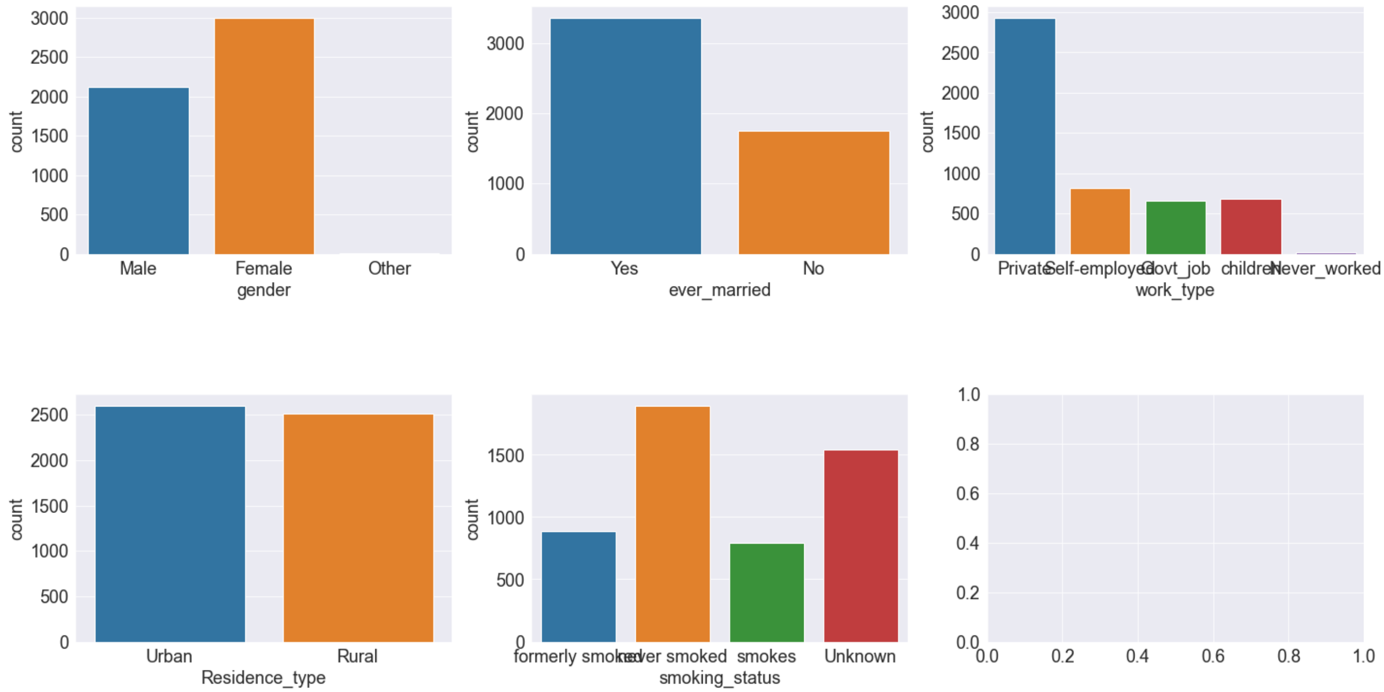
```
<Figure size 1080x576 with 0 Axes>
```



In [22]:
```
plt.suptitle('Univariate Analysis of categorical Features using count plot', fontsize=20
```

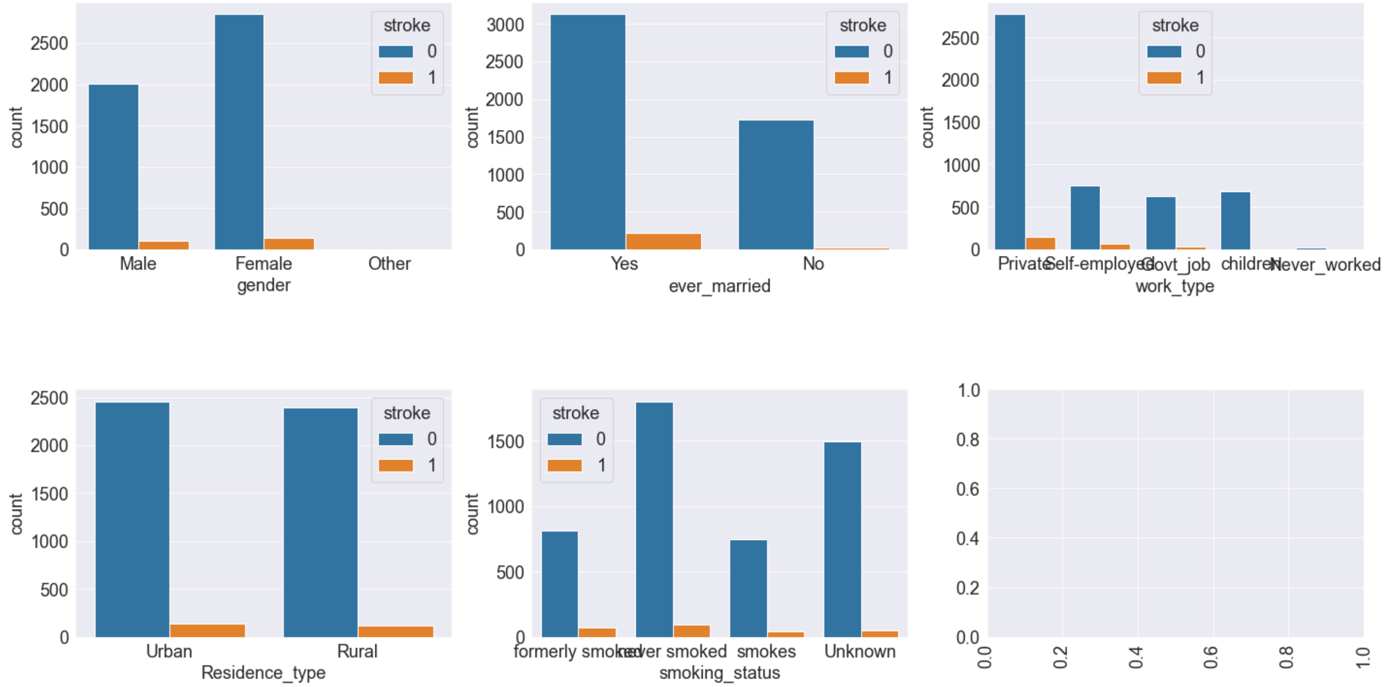
```
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col in cat_cols:
        sns.countplot(x=col, data=df, ax=ax[index])
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

```
<Figure size 1080x576 with 0 Axes>
```



In [23]:
```
plt.suptitle('Univariate Analysis of categorical Features using count plot', fontsize=20
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()
for col in cat_cols:
        sns.countplot(x=col, data=df, ax=ax[index],hue='stroke')
        index += 1
plt.xticks(rotation =90)
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
plt.show()
```

```
<Figure size 1080x576 with 0 Axes>
```



1. gender has 3 unique values male,female,and other

In [24]:
```
df['gender'].unique()
```
Out[24]:
```
array(['Male', 'Female', 'Other'], dtype=object)
```

In [25]:
```
df['gender'].value_counts()
```
Out[25]:
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [26]:
```
df[df['gender']=='Other']
```
Out[26]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bm
3116	Other	26.0	0	0	No	Private	Rural	143.33	22.

In [27]:
```
df = df.drop(df[df['gender']=='Other'].index)
```

In [28]:
```
df[df['gender']=='Other']
```
Out[28]:

gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	sm

In [29]:
```
# handling missing values
print('Mean of BMI = ',df['bmi'].mean())
print("Median of BMI= ",df['bmi'].median())
```
```
Mean of BMI =  28.894559902200502
Median of BMI=  28.1
```

In [30]:
```
# filling missing values with mean
bmi_mean=df['bmi'].mean()
df['bmi']=df['bmi'].fillna(bmi_mean)
```

In [31]:
```
df['bmi'].isnull().sum()
```

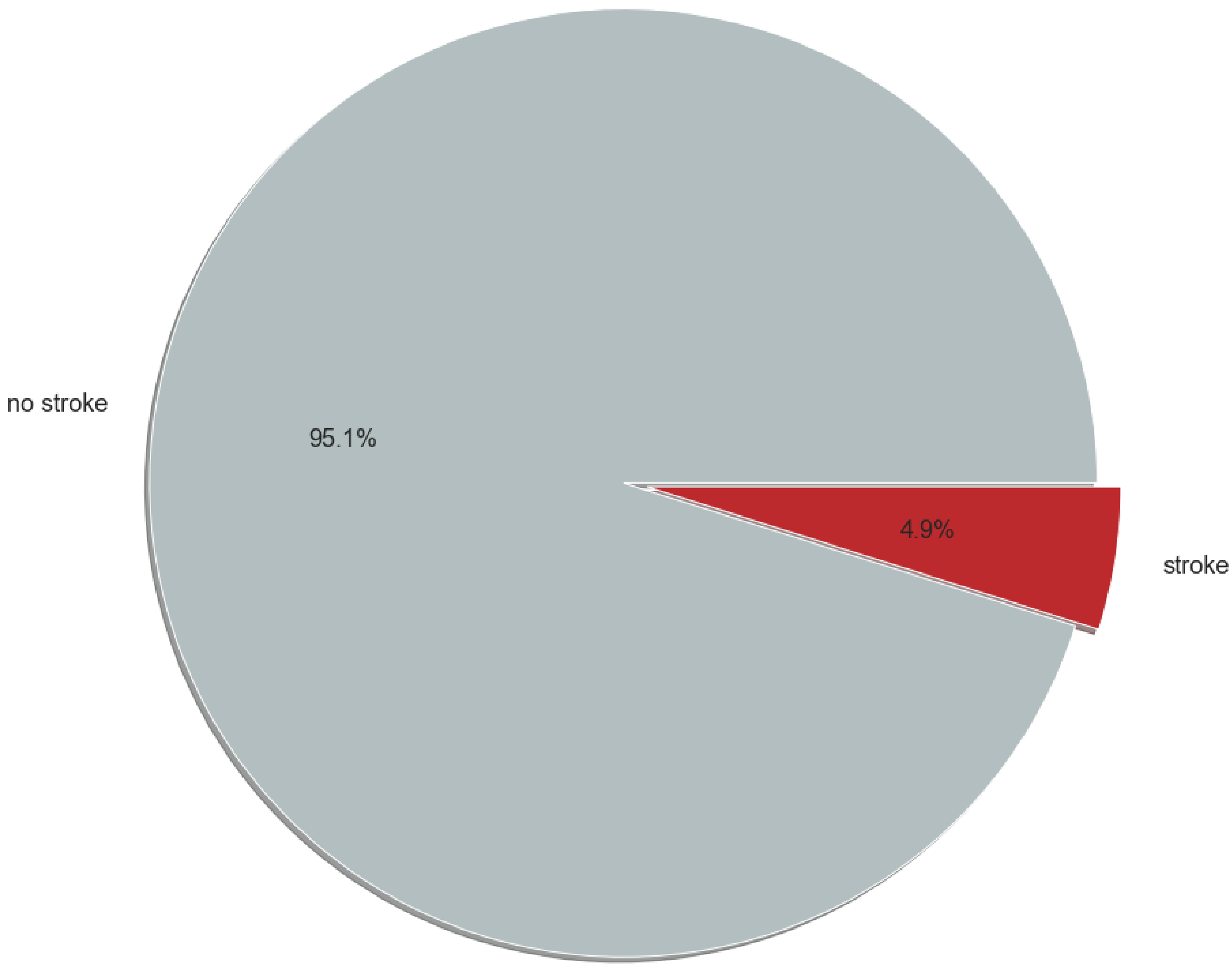
Out[31]: 0

In [32]:
```
# counting the number of passengers who are satisfied and who Dissatisfied
pie_df=pd.DataFrame(df.groupby('stroke')['stroke'].count())
pie_df
```

Out[32]:

	stroke
stroke	
0	4860
1	249

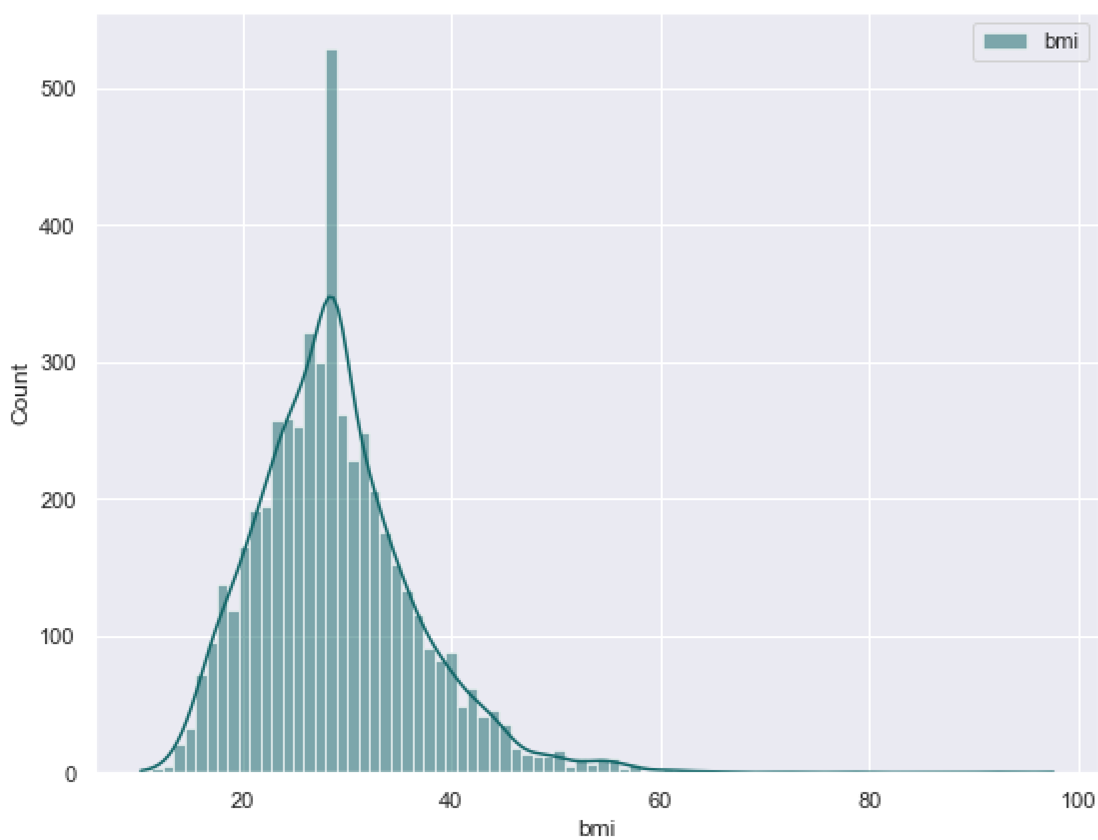
In [33]:
```
colors = ['#B2BEBF','#BD2A2E']
plt.pie(pie_df['stroke'],labels=['no stroke','stroke'],
        autopct='%.1f%%',colors=colors,radius=2,explode = (0, 0.1),shadow=True)
plt.show()
```



In [34]:
```
sns.set_theme(style="darkgrid")
fig = plt.figure(figsize=(9,7))
sns.histplot(df['avg_glucose_level'], color="#2C3532", label="avg_glucose_level", kde= T
plt.legend()
```

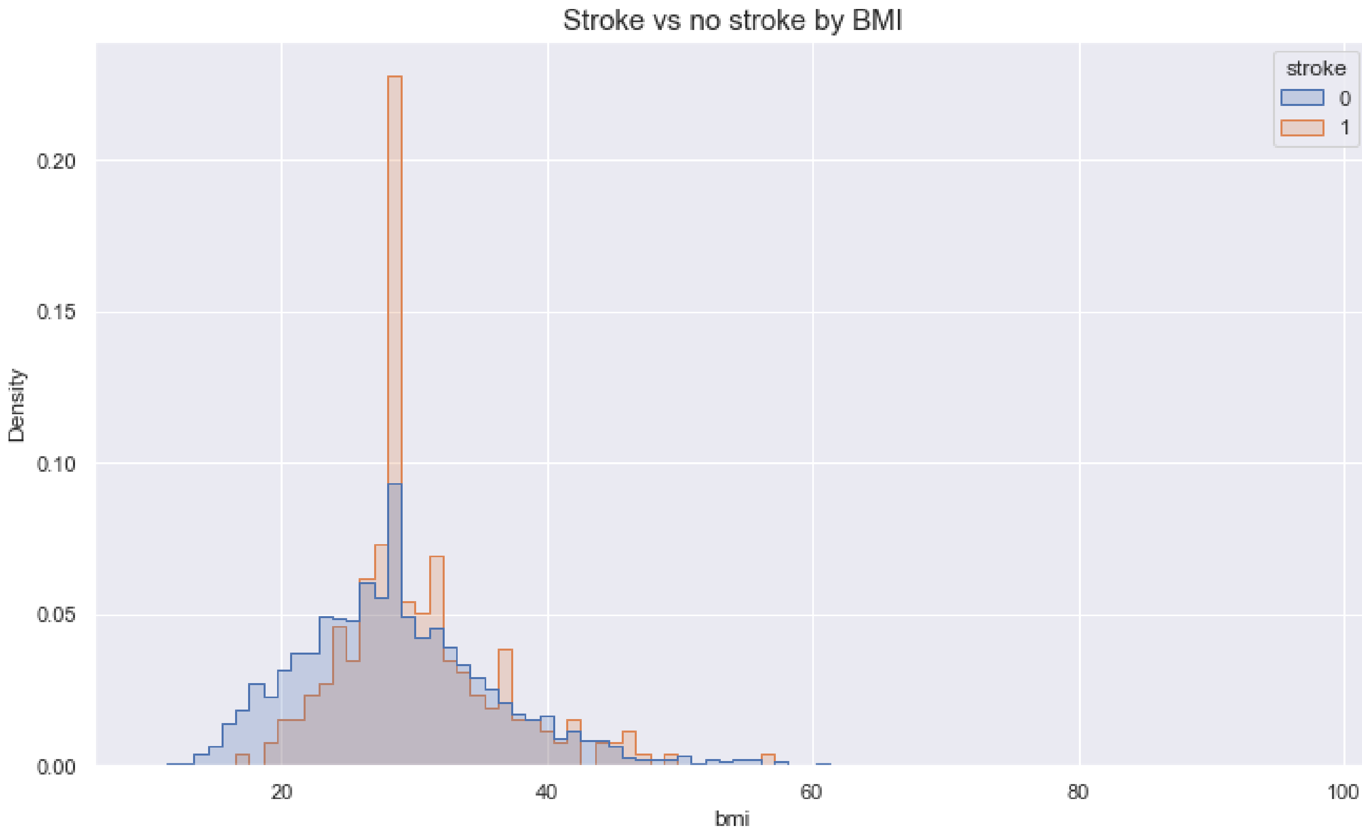
Out[34]: <matplotlib.legend.Legend at 0x28738359700>

In [35]:
```
fig = plt.figure(figsize=(9,7))
sns.histplot(df['bmi'], color="#0F6466", label="bmi", kde= True)
plt.legend()
plt.show()
```

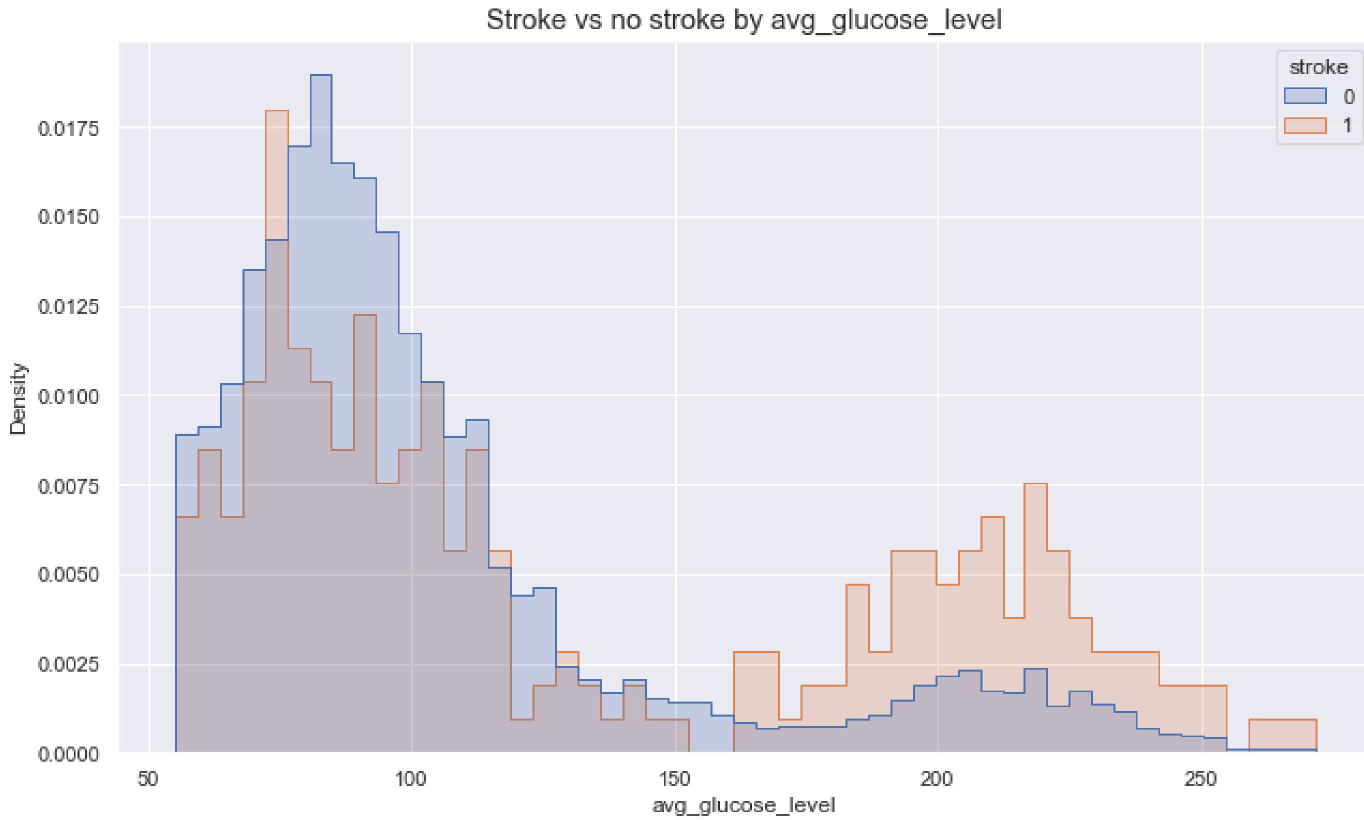


In [36]:
```
plt.figure(figsize=(12,7))
sns.histplot(
    df, x="bmi", hue="stroke",
    element="step",
    stat="density", common_norm=False,
```

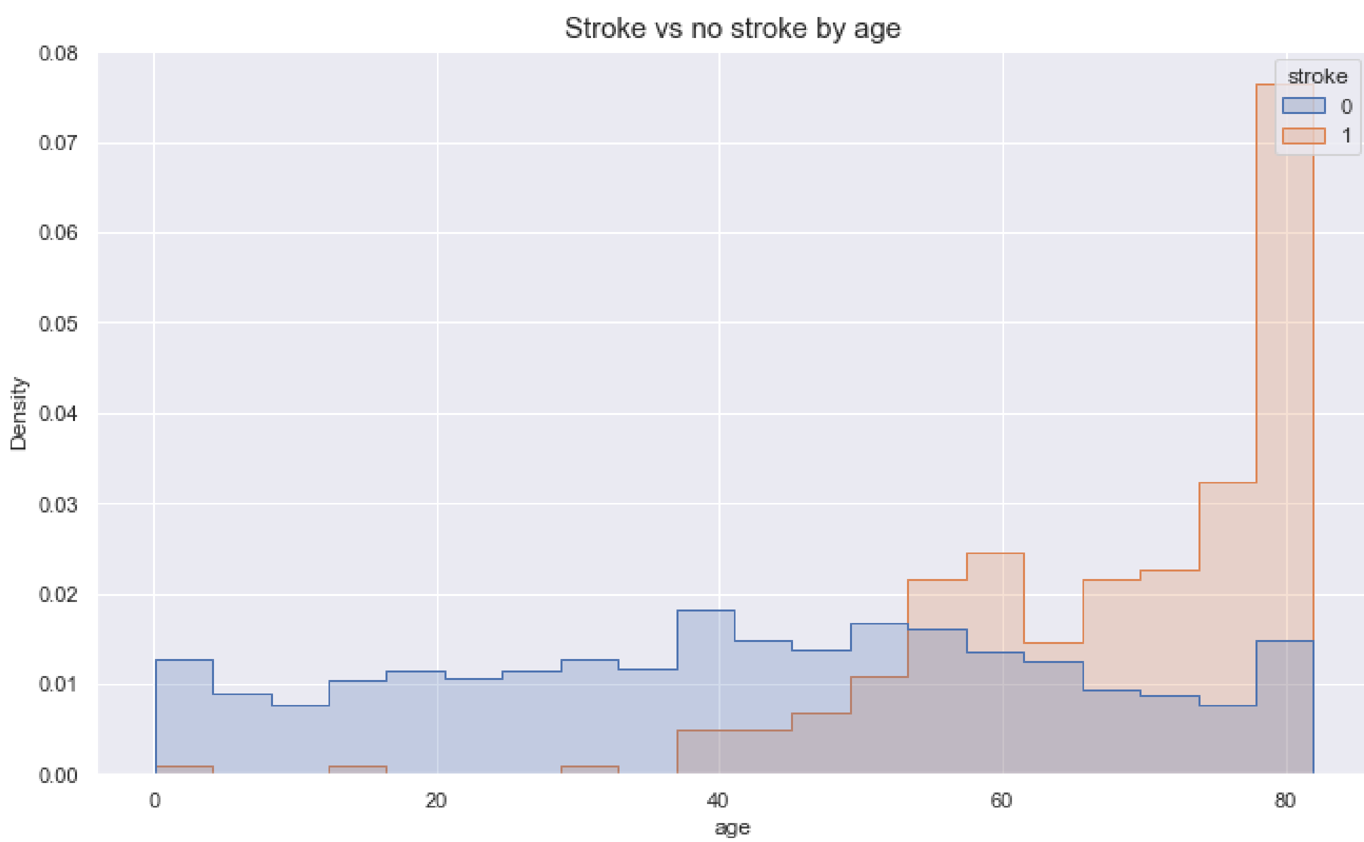
```
)
plt.title('Stroke vs no stroke by BMI', fontsize=15)
plt.show()
```



In [37]:
```
plt.figure(figsize=(12,7))
sns.histplot(
    df, x="avg_glucose_level", hue="stroke",
    element="step",
    stat="density", common_norm=False,
)
plt.title('Stroke vs no stroke by avg_glucose_level', fontsize=15)
plt.show()
```



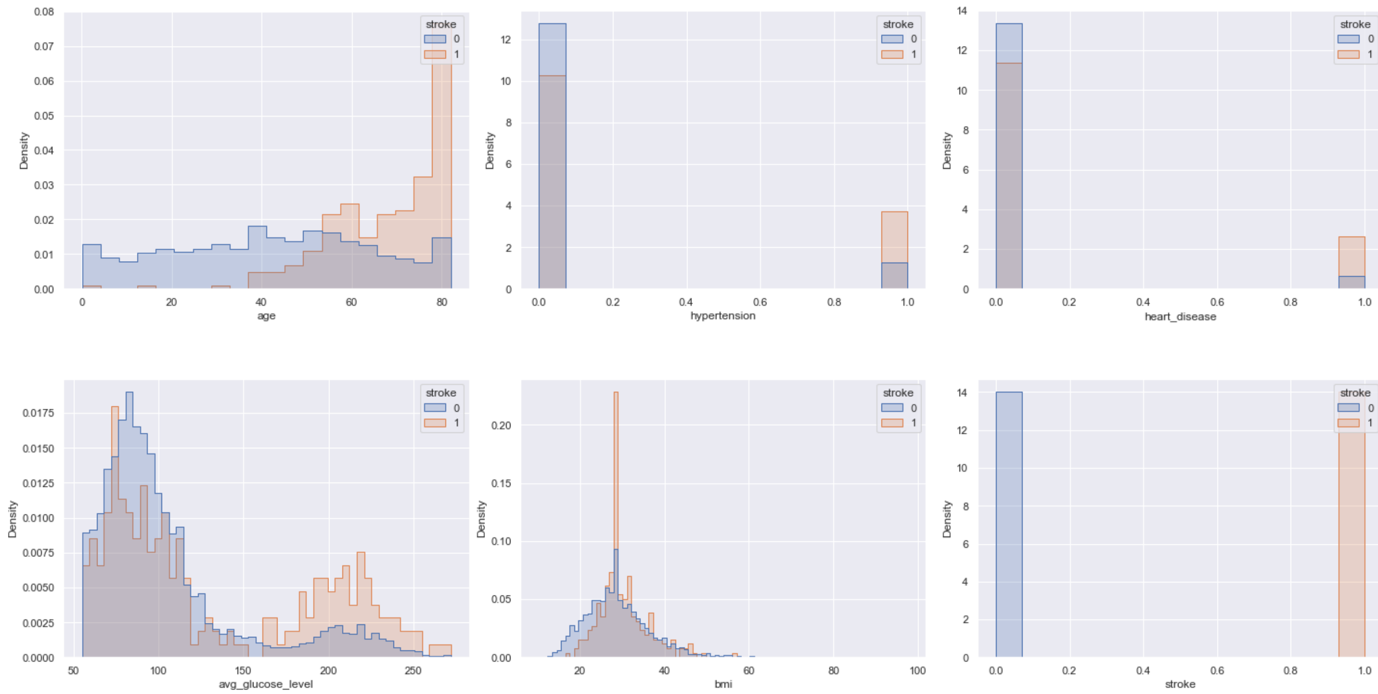
In [38]:
```
plt.figure(figsize=(12,7))
sns.histplot(
    df, x="age", hue="stroke",
    element="step",
    stat="density", common_norm=False,
)
plt.title('Stroke vs no stroke by age', fontsize=15)
plt.show()
```



In [39]:
```
plt.suptitle('Univariate Analysis of Numerical Features using KDE Plot', fontsize=20, fo
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
```

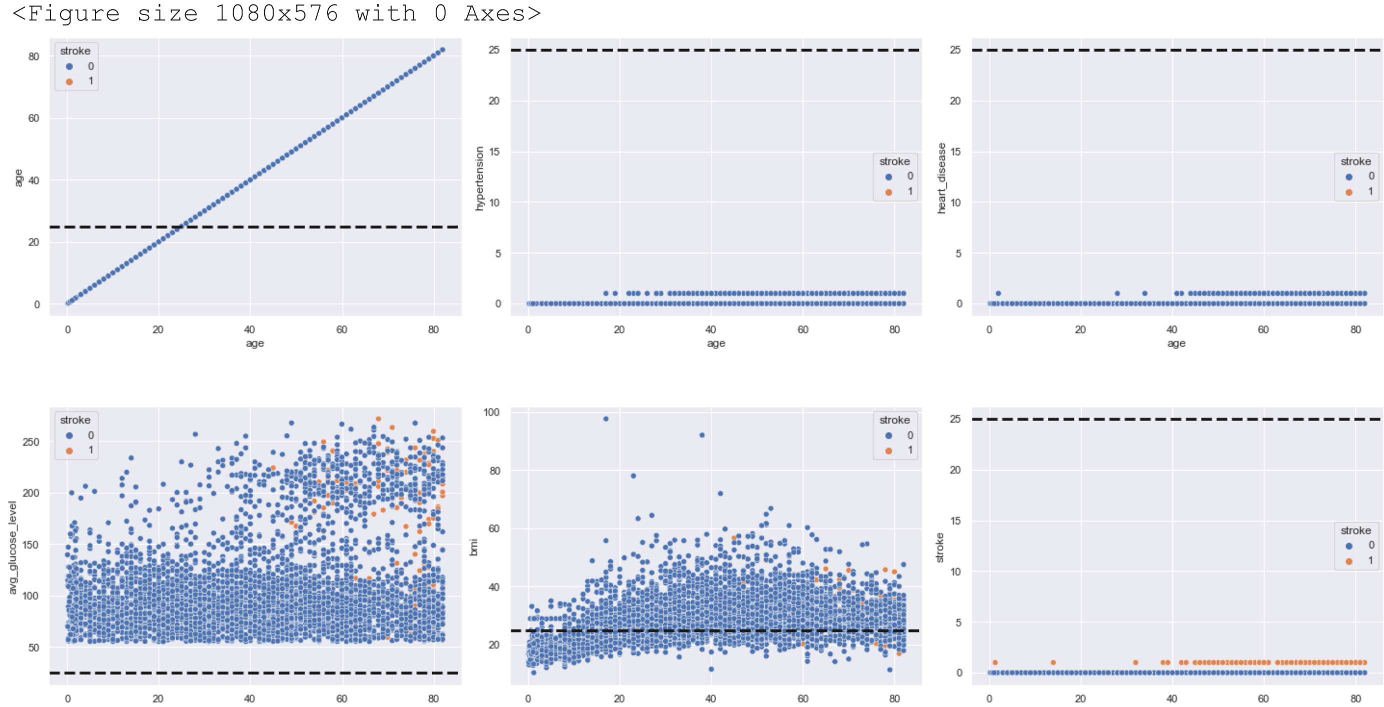
```
index = 0
ax = ax.flatten()

for col in num_cols:
        sns.histplot(x=col, data=df, ax=ax[index],hue='stroke',element='step',stat='dens
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

<Figure size 1080x576 with 0 Axes>



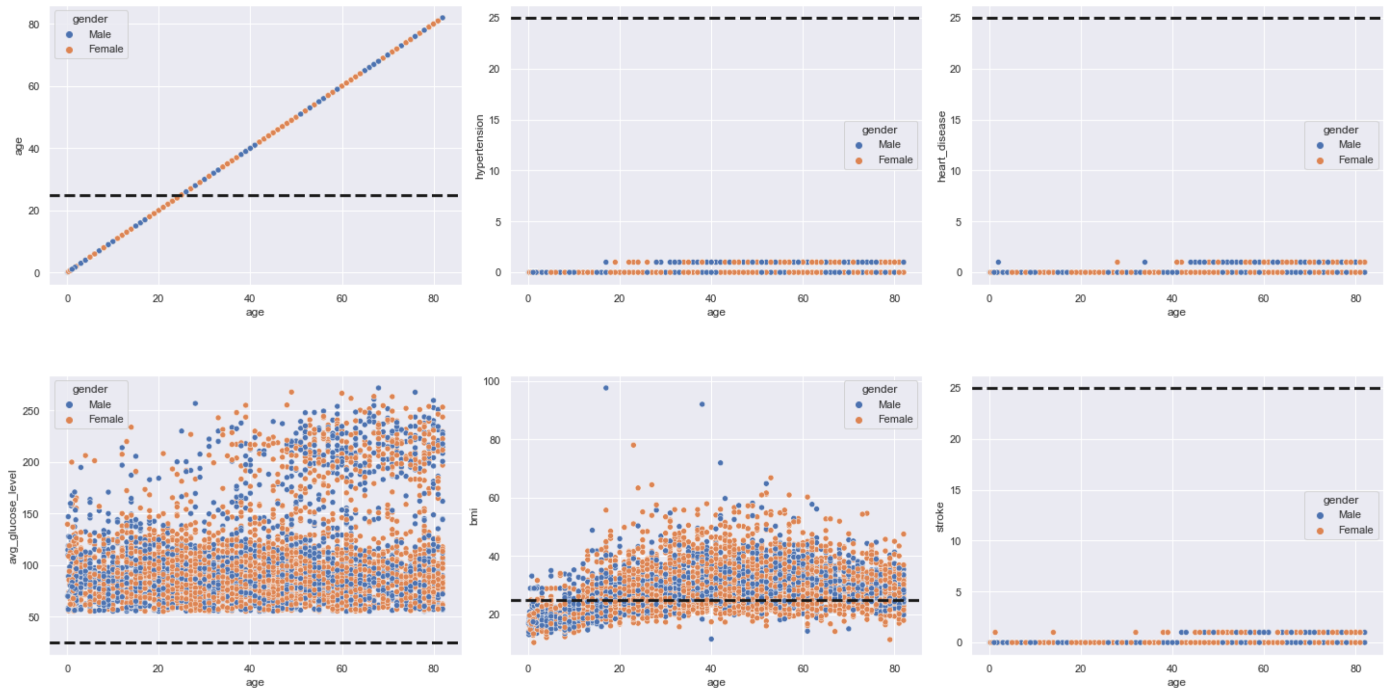
In [40]:
```
plt.suptitle('Univariate Analysis of Numerical Features using KDE Plot', fontsize=20, fo
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col in num_cols:
        sns.scatterplot(x='age',y = col, data=df, ax=ax[index],hue='stroke').axhline(y=
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

<Figure size 1080x576 with 0 Axes>



In [41]:
```python
plt.suptitle('Univariate Analysis of Numerical Features using KDE Plot', fontsize=20, fo
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col in num_cols:
        sns.scatterplot(x='age',y = col, data=df, ax=ax[index],hue='gender').axhline(y=
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
plt.show()
```

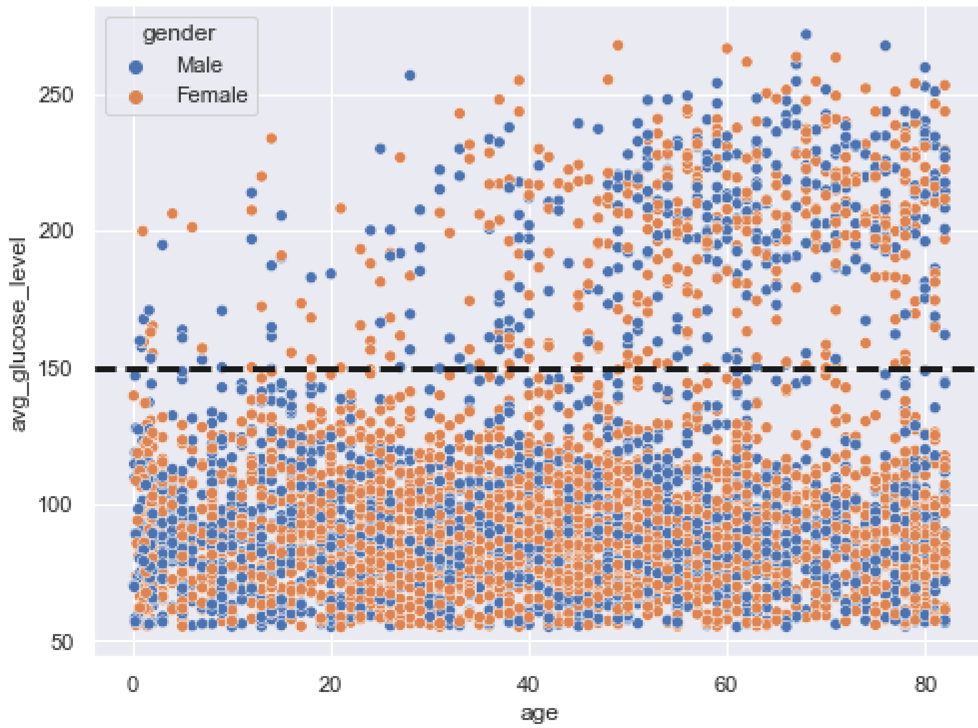
<Figure size 1080x576 with 0 Axes>



In [42]:
```python
plt.figure(figsize=(8,6))
fig = sns.scatterplot(data=df, x="age", y="bmi", hue='gender')
fig.axhline(y= 25, linewidth=3, color='k', linestyle= '--')
plt.show()
```

In [43]:
```python
plt.figure(figsize=(8,6))
fig = sns.scatterplot(data=df, x="age", y="avg_glucose_level", hue='gender')
```

```
fig.axhline(y= 150, linewidth=3, color='k', linestyle= '--')
plt.show()
```



In [44]:
```
# label encoding
le = LabelEncoder()
for i in cat_cols:
    df[i] = le.fit_transform(df[i])
```

In [45]:
```
df.head()
```

Out[45]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bn
0	1	67.0	0	1	1	2	1	228.69	36.6000
1	0	61.0	0	0	1	3	0	202.21	28.8945
2	1	80.0	0	1	1	2	0	105.92	32.5000
3	0	49.0	0	0	1	2	1	171.23	34.4000
4	0	79.0	1	0	1	3	0	174.12	24.0000

In [46]:
```
# saving cleaning df
df.to_csv("clean_heart_stroke.csv")
```

In [47]:
```
x = df.drop("stroke" , axis = 1).values
y = df["stroke"]
```

In [48]:
```
x_train, x_test, y_train , y_test = train_test_split(x,y, test_size=0.2 , random_state=4
```

In [49]:
```
sc=MinMaxScaler()
x_train = sc.fit_transform(x_train)
```

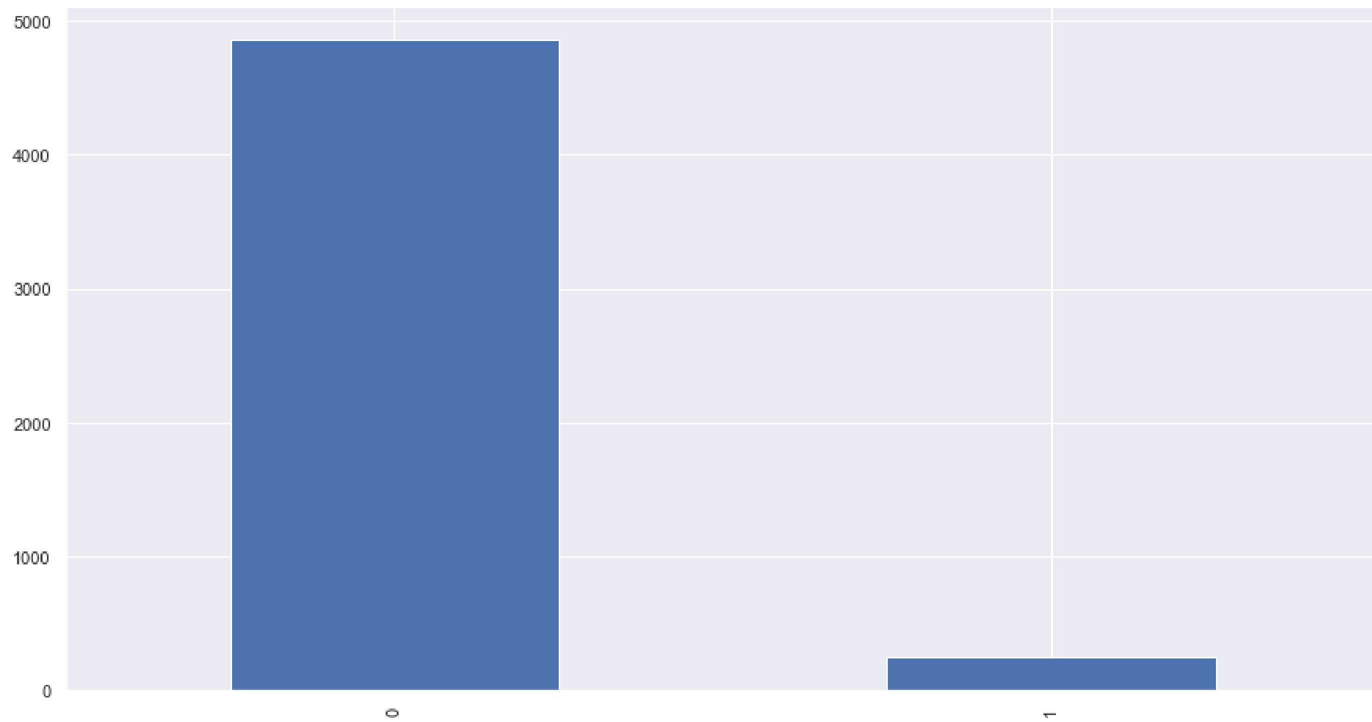
In [50]:
```
# show the value counts of the calsses in the target
#we can find data impalance
print(df['stroke'].value_counts())
df['stroke'].value_counts().sort_index().plot.bar()
```

```
0    4860
```

```
1      249
Name: stroke, dtype: int64
```

Out[50]:

```
<AxesSubplot:>
```



In [51]:

```
# Apply oversampling
oversample = SMOTE()
x_data_balanced, y_data_balanced = oversample.fit_resample(x_train, y_train.ravel())
```

Modeling:

Logistic regression:

In [52]:

```
lr = LogisticRegression()
lr.fit(x_data_balanced, y_data_balanced)
```

Out[52]:

```
▾ LogisticRegression
LogisticRegression()
```

In [53]:

```
y_pred_train_lr = lr.predict(x_data_balanced)
acc_train_lr = accuracy_score(y_data_balanced, y_pred_train_lr)

y_pred_test_lr = lr.predict(x_test)
acc_test_lr = accuracy_score(y_test, y_pred_test_lr)
print(acc_train_lr)
print(acc_test_lr)
```

```
0.7901282051282051
0.060665362035225046
```

In [54]:

```
print(classification_report(y_test, y_pred_test_lr))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       960
           1       0.06      1.00      0.11        62
```

```
    accuracy                           0.06      1022
   macro avg       0.03      0.50      0.06      1022
weighted avg       0.00      0.06      0.01      1022
```

In [55]:
```
lr_perc_score = precision_score(y_test, y_pred_test_lr)
lr_rec_score= recall_score(y_test, y_pred_test_lr)
lr_f1_score = f1_score(y_test, y_pred_test_lr)

print('Precision: %.3f' %lr_perc_score )
print('Recall: %.3f' % lr_rec_score)
print('F-measure: %.3f' % lr_f1_score)
```

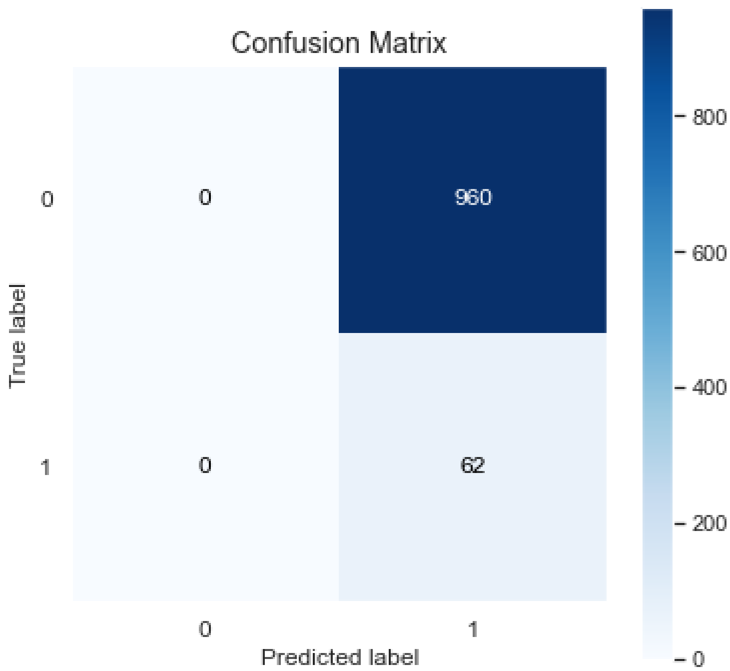
```
Precision: 0.061
Recall: 1.000
F-measure: 0.114
```

In [56]:
```
y_pred_prob_lr = lr.predict_proba(x_test)[:, 1]
lr_roc_auc_score = roc_auc_score(y_test, y_pred_prob_lr)
print('ROC AUC Score:', lr_roc_auc_score)
```

```
ROC AUC Score: 0.5
```

In [57]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_lr, figsize=(6,6), cmap= 'Blues'
```

Out[57]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True
label'>
```



Decision Tree

In [58]:
```
dt =DecisionTreeClassifier(max_features=14 , max_depth=12, criterion= 'gini')
dt.fit(x_data_balanced, y_data_balanced)
```

Out[58]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=12, max_features=14)
```

In [59]:
```
y_pred_train_dt = dt.predict(x_data_balanced)
acc_train_dt = accuracy_score(y_data_balanced, y_pred_train_dt)
```

```
y_pred_test_dt = dt.predict(x_test)
acc_test_dt = accuracy_score(y_test, y_pred_test_dt)
print(acc_train_dt)
print(acc_test_dt)
```

```
0.9401282051282052
0.901174168297456
```

In [60]:
```
print(classification_report(y_test, y_pred_test_dt))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       960
           1       0.05      0.03      0.04        62

    accuracy                           0.90      1022
   macro avg       0.49      0.49      0.49      1022
weighted avg       0.88      0.90      0.89      1022
```

In [61]:
```
dt_perc_score = precision_score(y_test, y_pred_test_dt)
dt_rec_score= recall_score(y_test, y_pred_test_dt)
dt_f1_score = f1_score(y_test, y_pred_test_dt)

print('Precision: %.3f' % dt_perc_score)
print('Recall: %.3f' % dt_rec_score)
print('F-measure: %.3f' % dt_f1_score)
```

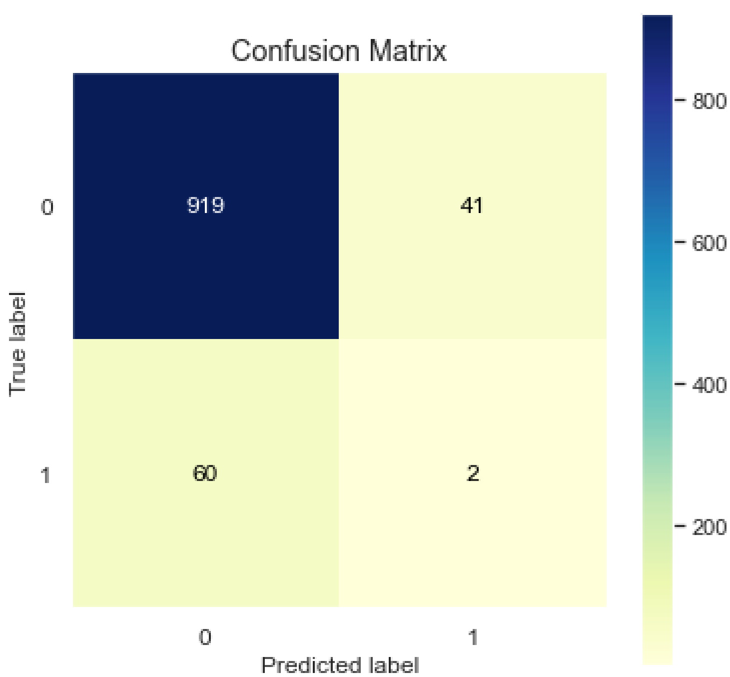
```
Precision: 0.047
Recall: 0.032
F-measure: 0.038
```

In [62]:
```
y_pred_prob_dt = dt.predict_proba(x_test)[:, 1]
dt_roc_auc_score = roc_auc_score(y_test, y_pred_prob_dt)
print('ROC AUC Score:', dt_roc_auc_score)
```

```
ROC AUC Score: 0.4947748655913978
```

In [63]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_dt, figsize=(6,6), cmap= 'YlGnBu
```

Out[63]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True
label'>
```



KNN

In [64]:
```
knn = KNeighborsClassifier()
knn.fit(x_data_balanced, y_data_balanced)
```

Out[64]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [65]:
```
y_pred_train_knn = knn.predict(x_data_balanced)
acc_train_knn = accuracy_score(y_data_balanced, y_pred_train_knn)

y_pred_test_knn = knn.predict(x_test)
acc_test_knn = accuracy_score(y_test, y_pred_test_knn)
print(acc_train_knn)
print(acc_test_knn)
```

```
0.9397435897435897
0.7906066536203522
```

In [66]:
```
print(classification_report(y_test, y_pred_test_knn))
```

```
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       960
           1       0.09      0.27      0.14        62

    accuracy                           0.79      1022
   macro avg       0.52      0.55      0.51      1022
weighted avg       0.89      0.79      0.84      1022
```

In [67]:
```
knn_perc_score = precision_score(y_test, y_pred_test_knn)
knn_rec_score= recall_score(y_test, y_pred_test_knn)
knn_f1_score = f1_score(y_test, y_pred_test_knn)

print('Precision: %.3f' % knn_perc_score)
print('Recall: %.3f' % knn_rec_score)
print('F-measure: %.3f' % knn_f1_score)
```

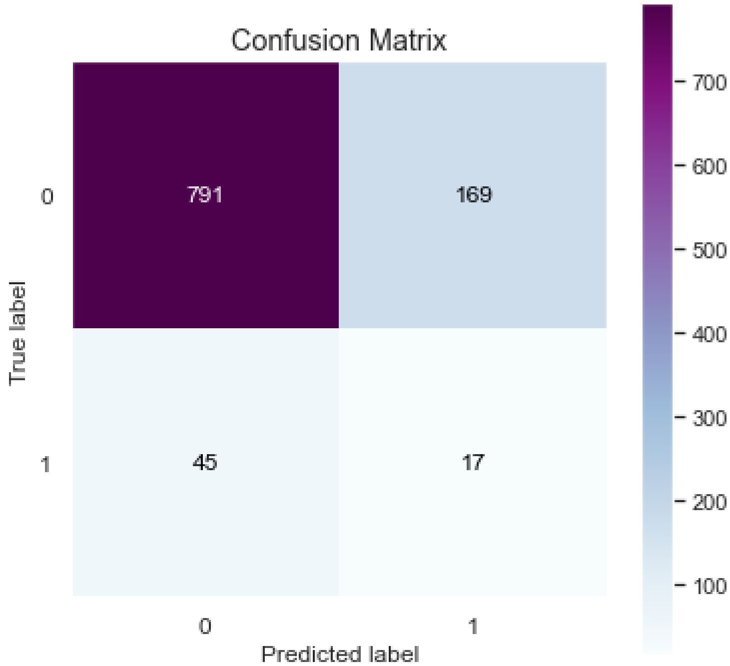
```
Precision: 0.091
Recall: 0.274
F-measure: 0.137
```

In [68]:
```
y_pred_prob_knn = knn.predict_proba(x_test)[:, 1]
knn_roc_auc_score = roc_auc_score(y_test, y_pred_prob_knn)
print('ROC AUC Score:', knn_roc_auc_score)
```

```
ROC AUC Score: 0.5656418010752688
```

In [69]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_knn, figsize=(6,6), cmap= 'BuPu'
```

Out[69]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True
label'>
```



SVC

In [70]:
```
svc = SVC(C=100, gamma=1000 ,probability= True)
svc.fit(x_data_balanced,y_data_balanced)
```

Out[70]:
```
▾ SVC
SVC(C=100, gamma=1000, probability=True)
```

In [71]:
```
y_pred_train_svc = svc.predict(x_data_balanced)
acc_train_svc = accuracy_score(y_data_balanced, y_pred_train_svc)

y_pred_test_svc = svc.predict(x_test)
acc_test_svc = accuracy_score(y_test, y_pred_test_svc)
print(acc_train_svc)
print(acc_test_svc)
```

```
1.0
0.9393346379647749
```

In [72]:
```
print(classification_report(y_test, y_pred_test_svc))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       960
           1       0.00      0.00      0.00        62

    accuracy                           0.94      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.88      0.94      0.91      1022
```

In [73]:
```
svc_perc_score = precision_score(y_test, y_pred_test_svc)
svc_rec_score= recall_score(y_test, y_pred_test_svc)
svc_f1_score = f1_score(y_test, y_pred_test_svc)

print('Precision: %.3f' % svc_perc_score)
print('Recall: %.3f' % svc_rec_score)
print('F-measure: %.3f' % svc_f1_score)
```

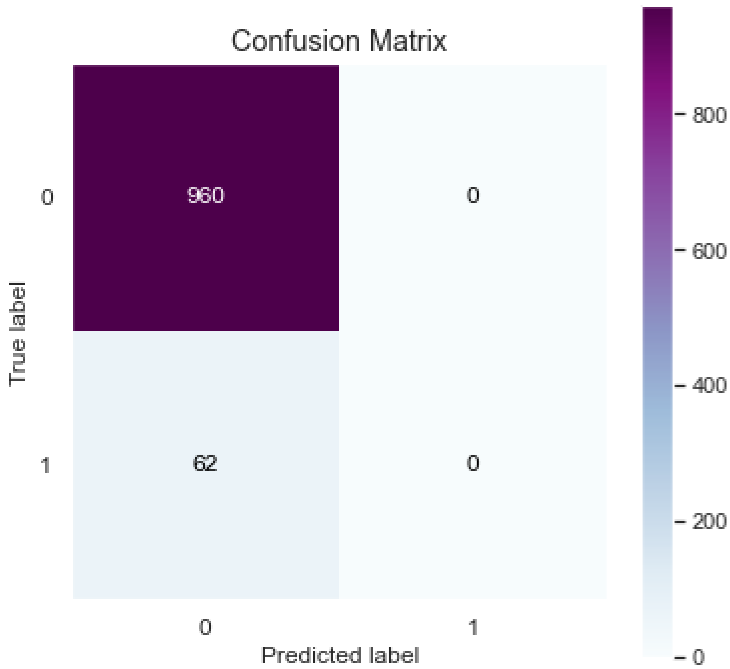
```
Precision: 0.000
Recall: 0.000
F-measure: 0.000
```

In [74]:
```
y_pred_prob_svc = svc.predict_proba(x_test)[:, 1]
svc_roc_auc_score = roc_auc_score(y_test, y_pred_prob_svc)
print('ROC AUC Score:', svc_roc_auc_score)
```

```
ROC AUC Score: 0.5
```

In [75]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_svc, figsize=(6,6), cmap= 'BuPu'
```

Out[75]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



Random Forest Classification:

In [76]:
```
rf = RandomForestClassifier(n_estimators = 100, criterion= 'entropy', random_state = 0)
rf.fit(x_data_balanced, y_data_balanced)
```

Out[76]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', random_state=0)
```

In [77]:
```
y_pred_train_rf = rf.predict(x_data_balanced)
acc_train_rf = accuracy_score(y_data_balanced, y_pred_train_rf)

y_pred_test_rf = rf.predict(x_test)
acc_test_rf = accuracy_score(y_test, y_pred_test_rf)
print(acc_train_rf)
print(acc_test_rf)
```

```
1.0
0.9383561643835616
```

In [78]:
```
print(classification_report(y_test, y_pred_test_rf))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       960
           1       0.00      0.00      0.00        62
```

```
    accuracy                           0.94      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.88      0.94      0.91      1022
```

In [79]:
```
rf_perc_score = precision_score(y_test, y_pred_test_rf)
rf_rec_score= recall_score(y_test, y_pred_test_rf)
rf_f1_score = f1_score(y_test, y_pred_test_rf)

print('Precision: %.3f' %rf_perc_score )
print('Recall: %.3f' % rf_rec_score)
print('F-measure: %.3f' % rf_f1_score)
```

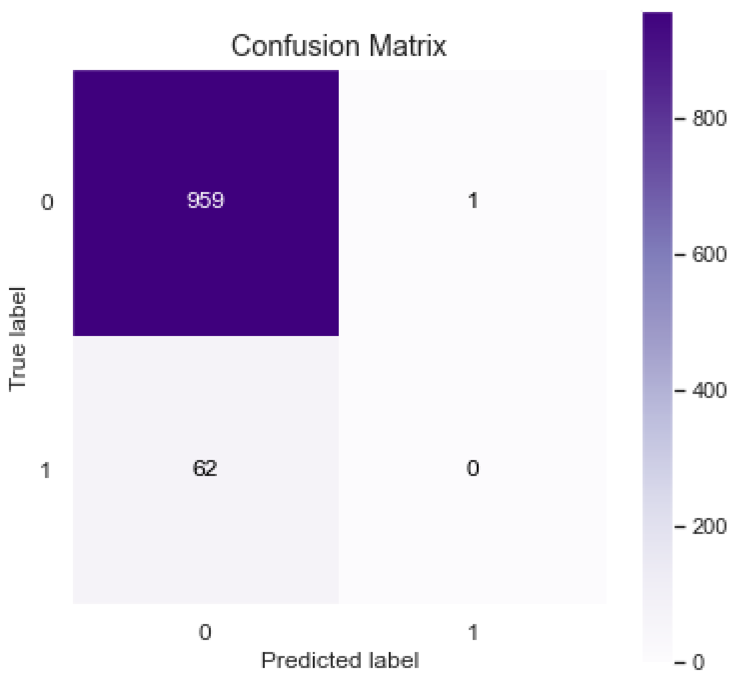
```
Precision: 0.000
Recall: 0.000
F-measure: 0.000
```

In [80]:
```
y_pred_prob_rf = rf.predict_proba(x_test)[:, 1]
rf_roc_auc_score = roc_auc_score(y_test, y_pred_prob_rf)
print('ROC AUC Score:', rf_roc_auc_score)
```

```
ROC AUC Score: 0.6210853494623655
```

In [81]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_rf, figsize=(6,6), cmap= 'Purple
```

Out[81]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



GradientBoostingClassifier

In [82]:
```
gbc = GradientBoostingClassifier()
gbc.fit(x_data_balanced,y_data_balanced)
```

Out[82]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [83]:
```
y_pred_train_gbc = gbc.predict(x_data_balanced)
acc_train_gbc = accuracy_score(y_data_balanced, y_pred_train_gbc)
```

```
y_pred_test_gbc = gbc.predict(x_test)
acc_test_gbc = accuracy_score(y_test, y_pred_test_gbc)
print(acc_train_gbc)
print(acc_test_gbc)
```

```
0.9215384615384615
0.9393346379647749
```

In [84]:
```
print(classification_report(y_test, y_pred_test_gbc))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       960
           1       0.00      0.00      0.00        62

    accuracy                           0.94      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.88      0.94      0.91      1022
```

In [85]:
```
gbc_perc_score = precision_score(y_test, y_pred_test_gbc)
gbc_rec_score= recall_score(y_test, y_pred_test_gbc)
gbc_f1_score = f1_score(y_test, y_pred_test_gbc)

print('Precision: %.3f' %gbc_perc_score )
print('Recall: %.3f' % gbc_rec_score)
print('F-measure: %.3f' % gbc_f1_score)
```

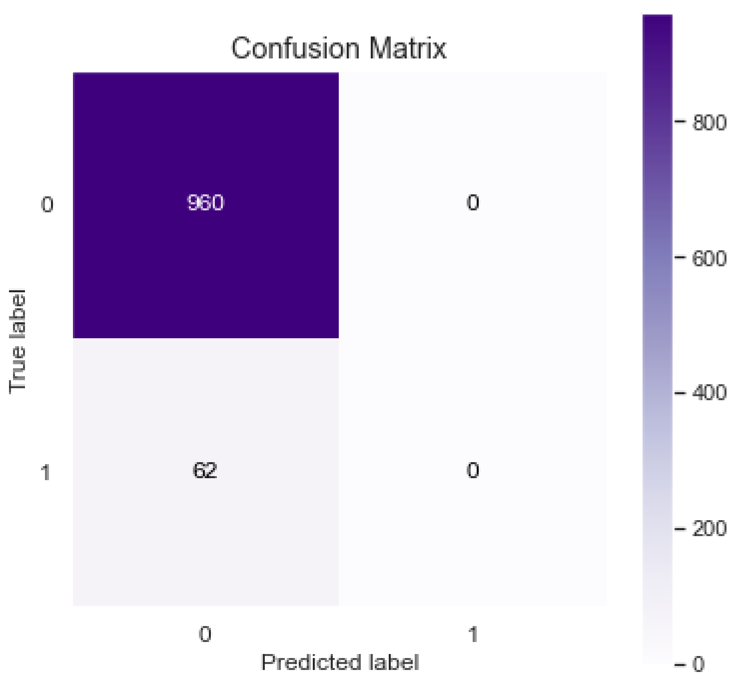
```
Precision: 0.000
Recall: 0.000
F-measure: 0.000
```

In [86]:
```
y_pred_prob_gbc = gbc.predict_proba(x_test)[:, 1]
gbc_roc_auc_score = roc_auc_score(y_test, y_pred_prob_gbc)
print('ROC AUC Score:', gbc_roc_auc_score)
```

```
ROC AUC Score: 0.5873487903225807
```

In [87]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_gbc, figsize=(6,6), cmap= 'Purpl
```

Out[87]:
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True
label'>
```



XGB CLASSIFIER

In [88]:
```
xgb = XGBClassifier(eval_metric= 'error', learning_rate= 0.05)
xgb.fit(x_data_balanced, y_data_balanced)
```

Out[88]:
```
XGBClassifier

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric='error', gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [89]:
```
y_pred_train_xgb = xgb.predict(x_data_balanced)
acc_train_xgb = accuracy_score(y_data_balanced, y_pred_train_xgb)

y_pred_test_xgb = xgb.predict(x_test)
acc_test_xgb = accuracy_score(y_test, y_pred_test_xgb)

print(acc_train_xgb)
print(acc_test_xgb)
```

```
0.9443589743589743
0.9393346379647749
```

In [90]:
```
print(classification_report(y_test, y_pred_test_xgb))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       960
           1       0.00      0.00      0.00        62

    accuracy                           0.94      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.88      0.94      0.91      1022
```

In [91]:
```
xgb_perc_score = precision_score(y_test, y_pred_test_xgb)
xgb_rec_score= recall_score(y_test, y_pred_test_xgb)
xgb_f1_score = f1_score(y_test, y_pred_test_xgb)

print('Precision: %.3f' %xgb_perc_score )
print('Recall: %.3f' % xgb_rec_score)
print('F-measure: %.3f' % xgb_f1_score)
```

```
Precision: 0.000
Recall: 0.000
F-measure: 0.000
```

In [93]:
```
y_pred_prob_xgb = xgb.predict_proba(x_test)[:, 1]
xgb_roc_auc_score = roc_auc_score(y_test, y_pred_prob_xgb)
print('ROC AUC Score:', xgb_roc_auc_score)
```

```
ROC AUC Score: 0.6870547715053764
```

hyperparmeters Tunning

In [94]:
```
grid_models = [(LogisticRegression(),[{"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}])
    (SVC(probability=True),[{'C':[10,100], 'gamma':[100,500,1000],'kernel':['linear', 'r
    (DecisionTreeClassifier(),[{'max_features':[5,6,10,12,14,18,20],'max_depth':[6,10,12
    (RandomForestClassifier(),[{'n_estimators':[100,150,200],'criterion':['gini','entrop
    (XGBClassifier(), [{'learning_rate': [0.01,0.05, 0.1, 0.5, 1], 'eval_metric': ['erro
```

In [95]:
```
for i,j in grid_models:
    grid = GridSearchCV(estimator=i,param_grid = j, scoring = 'accuracy',cv = 5)
    grid.fit(x_data_balanced, y_data_balanced)
    best_accuracy = grid.best_score_
    best_param = grid.best_params_
    print('{}:\nBest Accuracy : {:.2f}%'.format(i,best_accuracy*100))
    print('Best Parameters : ',best_param)
    print('')
    print('----------------')
    print('')
```

```
LogisticRegression():
Best Accuracy : 79.08%
Best Parameters :  {'C': 100.0, 'penalty': 'l2'}

----------------

SVC(probability=True):
Best Accuracy : 93.81%
Best Parameters :  {'C': 10, 'gamma': 100, 'kernel': 'rbf', 'random_state': 0}

----------------

DecisionTreeClassifier():
Best Accuracy : 91.59%
Best Parameters :  {'criterion': 'entropy', 'max_depth': 20, 'max_features': 10, 'random
_state': 0}

----------------

RandomForestClassifier():
Best Accuracy : 95.90%
Best Parameters :  {'criterion': 'entropy', 'n_estimators': 150, 'random_state': 0}

----------------

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, gamma=None,
              gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=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,
              reg_alpha=None, reg_lambda=None, ...):
Best Accuracy : 95.81%
Best Parameters :  {'eval_metric': 'error', 'learning_rate': 0.5}

----------------
```

In [96]:
```
# classification with random forest

# Change parameters
rf = RandomForestClassifier(n_estimators = 100, criterion = 'entropy', random_state = 0)
rf.fit(x_data_balanced, y_data_balanced)
```

```
y_pred_train_rf = rf.predict(x_data_balanced)
acc_train_rf = accuracy_score(y_data_balanced, y_pred_train_rf)

y_pred_test_rf = rf.predict(x_test)
acc_test_rf = accuracy_score(y_test, y_pred_test_rf)
print(acc_train_rf)
print(acc_test_rf)

rf_perc_score = precision_score(y_test, y_pred_test_rf)
rf_rec_score= recall_score(y_test, y_pred_test_rf)
rf_f1_score = f1_score(y_test, y_pred_test_rf)

print('Precision: %.3f' %rf_perc_score )
print('Recall: %.3f' % rf_rec_score)
print('F-measure: %.3f' % rf_f1_score)

y_pred_prob_rf = rf.predict_proba(x_test)[:, 1]
rf_roc_auc_score = roc_auc_score(y_test, y_pred_prob_rf)
print('ROC AUC Score:', rf_roc_auc_score)
```

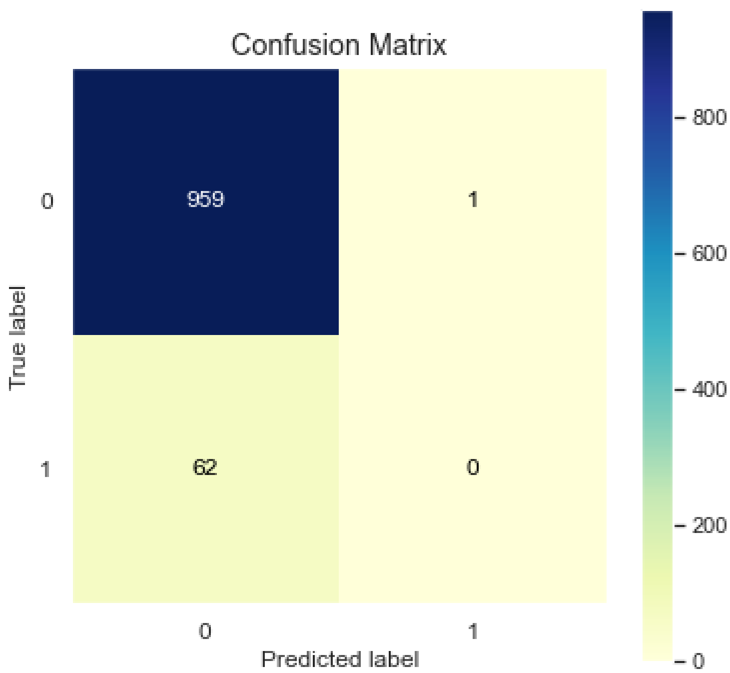
```
1.0
0.9383561643835616
Precision: 0.000
Recall: 0.000
F-measure: 0.000
ROC AUC Score: 0.6210853494623655
```

In [97]:

```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_rf, figsize=(6,6), cmap= 'YlGnBu
```

Out[97]:

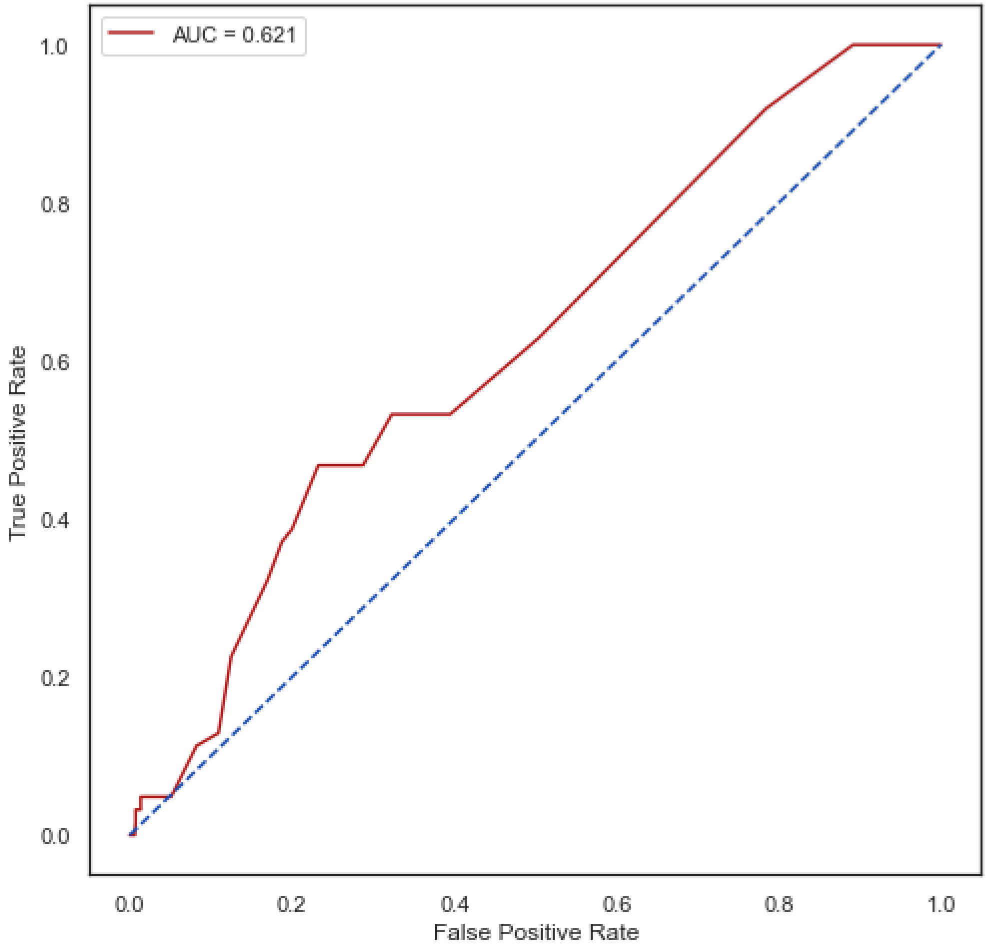
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True
label'>
```



In [98]:

```
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_prob_rf)
roc_auc = auc(false_positive_rate, true_positive_rate)

sns.set_theme(style = 'white')
plt.figure(figsize = (8, 8))
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
plt.legend()
plt.show()
```



In [99]:

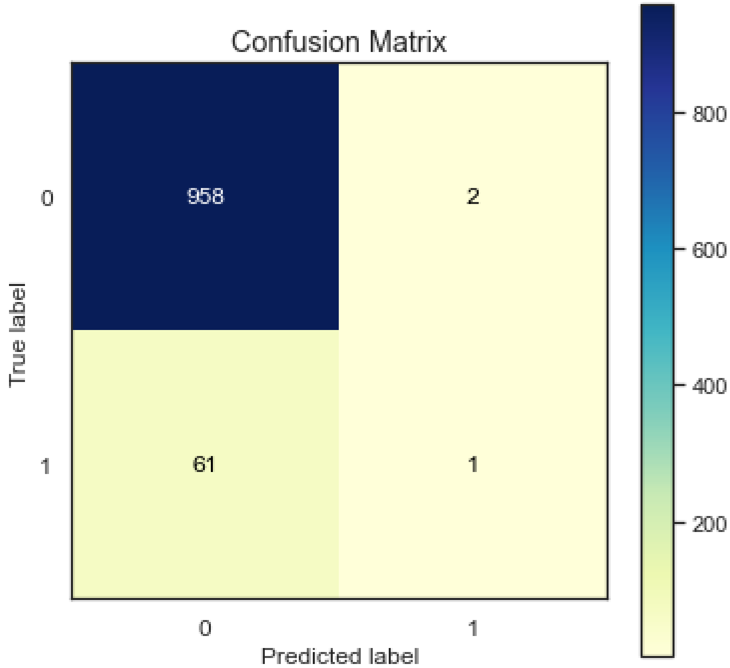
```
# Change parameters

dt =DecisionTreeClassifier(max_features=5 , max_depth=12,criterion = 'entropy', random_s
dt.fit(x_data_balanced, y_data_balanced)

y_pred_train_dt = dt.predict(x_data_balanced)
acc_train_dt = accuracy_score(y_data_balanced, y_pred_train_dt)

y_pred_test_dt = dt.predict(x_test)
acc_test_dt = accuracy_score(y_test, y_pred_test_dt)
print(acc_train_dt)
print(acc_test_dt)

dt_perc_score = precision_score(y_test, y_pred_test_dt)
dt_rec_score= recall_score(y_test, y_pred_test_dt)
dt_f1_score = f1_score(y_test, y_pred_test_dt)

print('Precision: %.3f' % dt_perc_score)
print('Recall: %.3f' % dt_rec_score)
print('F-measure: %.3f' % dt_f1_score)

y_pred_prob_dt = dt.predict_proba(x_test)[:, 1]
dt_roc_auc_score = roc_auc_score(y_test, y_pred_prob_dt)
print('ROC AUC Score:', dt_roc_auc_score)
```

```
0.9235897435897436
0.9383561643835616
Precision: 0.333
Recall: 0.016
F-measure: 0.031
ROC AUC Score: 0.5070228494623655
```

In [100…

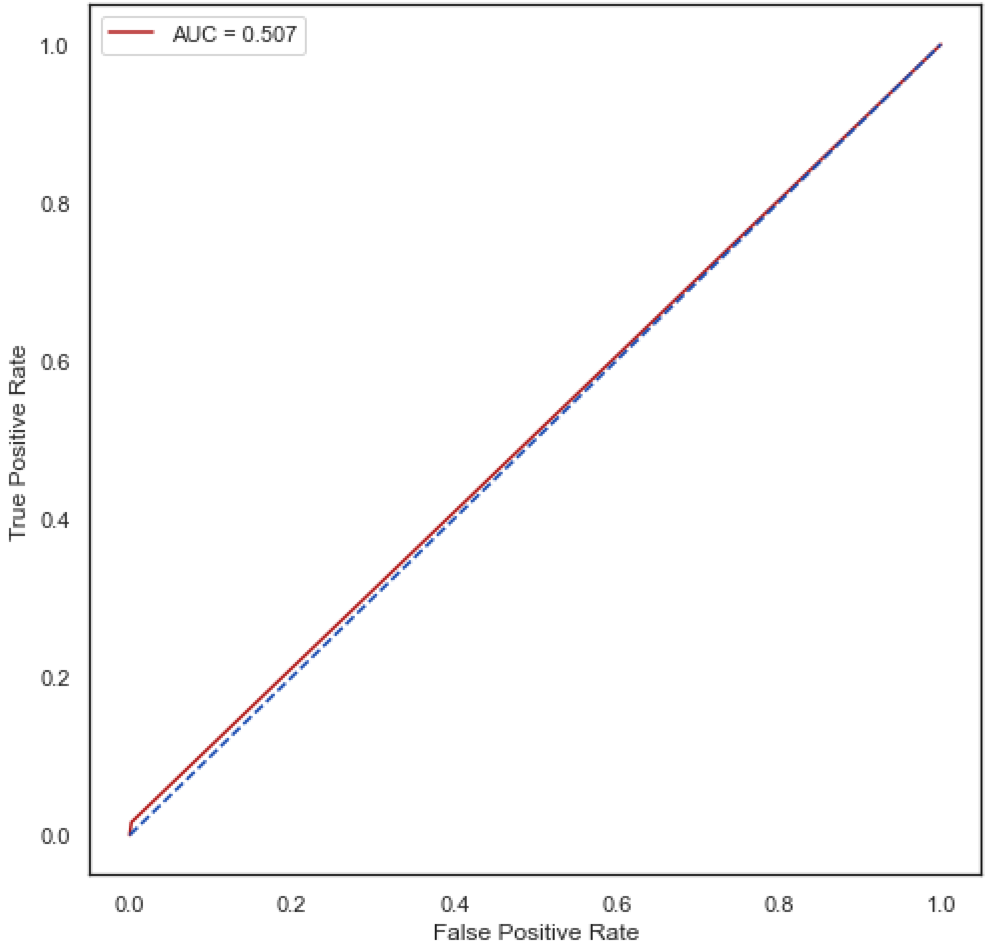
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_dt, figsize=(6,6), cmap= 'YlGnBu
```

Out[100]: <AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>



In [101…

```
# visualize Roc AUC Curve

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_prob_dt)
roc_auc = auc(false_positive_rate, true_positive_rate)

sns.set_theme(style = 'white')
plt.figure(figsize = (8, 8))
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



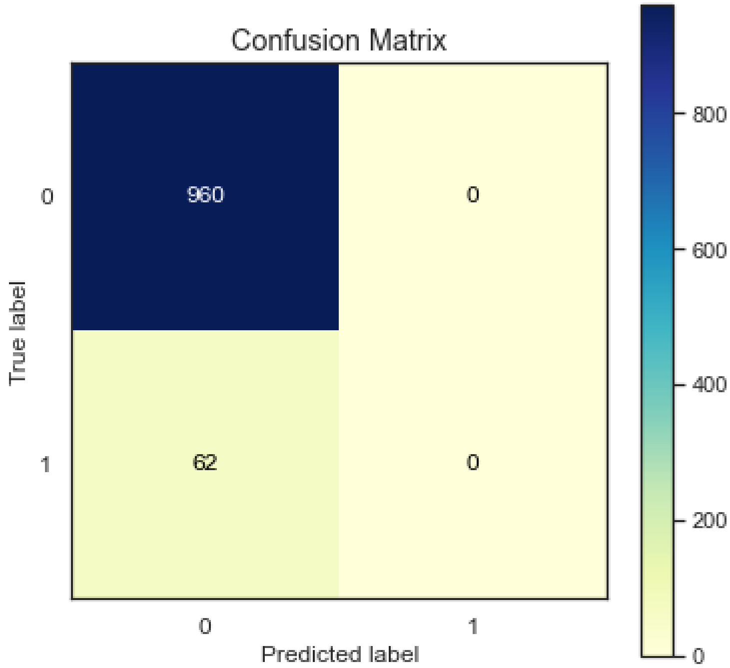
In [102…

```
# Change parameters

xgb = XGBClassifier(eval_metric= 'error', learning_rate= 0.1, random_state=0)
xgb.fit(x_data_balanced, y_data_balanced)

y_pred_train_xgb = xgb.predict(x_data_balanced)
acc_train_xgb = accuracy_score(y_data_balanced, y_pred_train_xgb)

y_pred_test_xgb = xgb.predict(x_test)
acc_test_xgb = accuracy_score(y_test, y_pred_test_xgb)

print(acc_train_xgb)
print(acc_test_xgb)

xgb_perc_score = precision_score(y_test, y_pred_test_xgb)
xgb_rec_score= recall_score(y_test, y_pred_test_xgb)
xgb_f1_score = f1_score(y_test, y_pred_test_xgb)

print('Precision: %.3f' %xgb_perc_score )
print('Recall: %.3f' % xgb_rec_score)
print('F-measure: %.3f' % xgb_f1_score)

y_pred_prob_xgb = xgb.predict_proba(x_test)[:, 1]
xgb_roc_auc_score = roc_auc_score(y_test, y_pred_prob_xgb)
print('ROC AUC Score:', xgb_roc_auc_score)
```

```
0.973974358974359
0.9393346379647749
Precision: 0.000
Recall: 0.000
F-measure: 0.000
ROC AUC Score: 0.6698840725806452
```

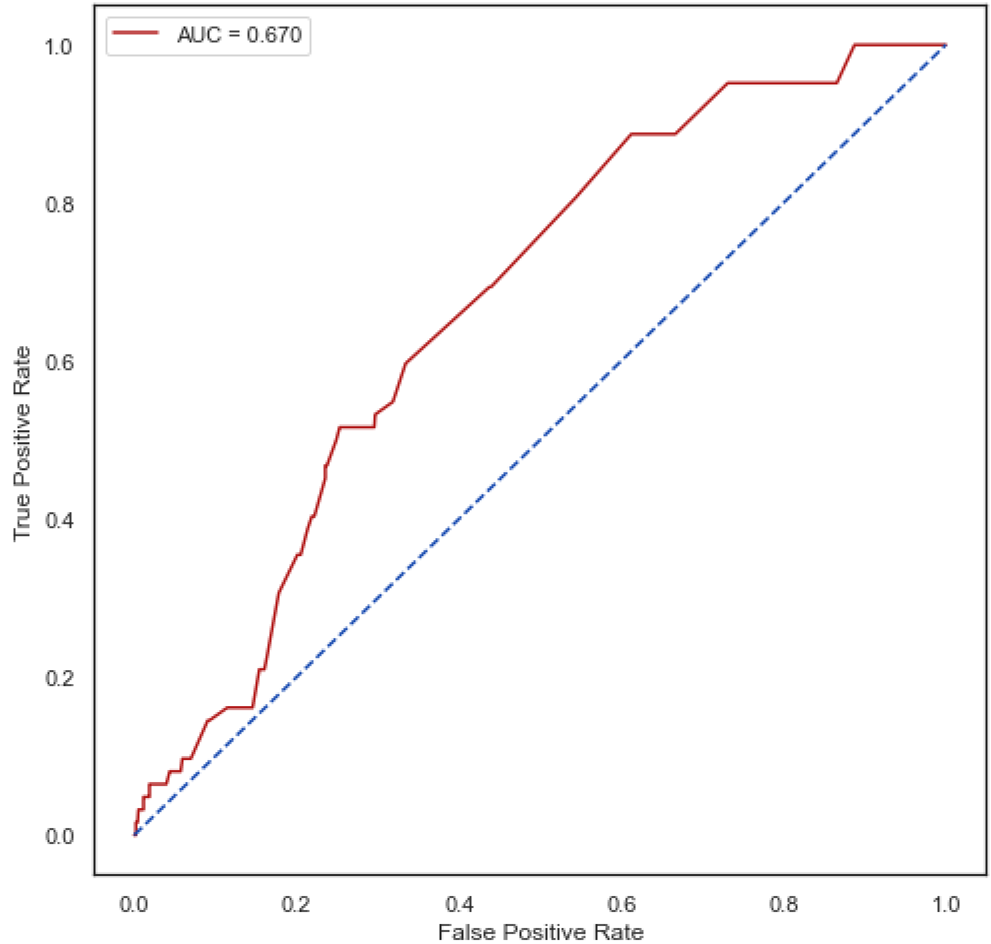
In [103…

```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_xgb, figsize=(6,6), cmap= 'YlGnB
```



In [104…

```
# visualize Roc AUC Curve

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_prob_xgb)
roc_auc = auc(false_positive_rate, true_positive_rate)

sns.set_theme(style = 'white')
plt.figure(figsize = (8, 8))
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [ ]:
```
svc = SVC(C=10, gamma=1000 ,probability= True)
svc.fit(x_data_balanced, y_data_balanced)

y_pred_train_svc = svc.predict(x_data_balanced)
acc_train_svc = accuracy_score(y_data_balanced, y_pred_train_svc)

y_pred_test_svc = svc.predict(x_test)
acc_test_svc = accuracy_score(y_test, y_pred_test_svc)

print(acc_train_svc)
print(acc_test_svc)

svc_perc_score = precision_score(y_test, y_pred_test_svc)
svc_rec_score= recall_score(y_test, y_pred_test_svc)
svc_f1_score = f1_score(y_test, y_pred_test_svc)

print('Precision: %.3f' % svc_perc_score)
print('Recall: %.3f' % svc_rec_score)
print('F-measure: %.3f' % svc_f1_score)

y_pred_prob_svc = svc.predict_proba(x_test)[:, 1]
svc_roc_auc_score=  roc_auc_score(y_test, y_pred_prob_svc)
print('ROC AUC Score:', svc_roc_auc_score)
```

In [ ]:
```
skplt.metrics.plot_confusion_matrix(y_test, y_pred_test_svc, figsize=(6,6), cmap= 'YlGnB
```

In [ ]:
```
# visualize Roc AUC Curve

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_prob_svc)
roc_auc = auc(false_positive_rate, true_positive_rate)

sns.set_theme(style = 'white')
plt.figure(figsize = (8, 8))
```

```python
plt.plot(false_positive_rate,true_positive_rate, color = '#b01717', label = 'AUC = %0.3f
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle = '--', color = '#174ab0')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```

In [ ]:

```python

```