Brain_Stroke_Prediction

A stroke occurs when a blood vessel in the brain ruptures and bleeds, or when there's a blockage in the blood supply to the brain. The rupture or blockage prevents blood and oxygen from reaching the brain's tissues. Without oxygen, brain cells and tissue become damaged and begin to die within minutes, and the abilities controlled by that area of the brain are lost.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.metrics import mean_absolute_error, accuracy_score, roc_curve, auc
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import collections
from collections import Counter
import plotly.express as px
import warnings; warnings.filterwarnings("ignore"); warnings.simplefilter('ignore')
```

In [2]:
```
df=pd.read_csv('E:\JUPYTER NOTE BOOK\Dataset/brain_stroke.csv')
df.head()
```

Out[2]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
2	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
3	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
4	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
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
```

In [4]:
```
df['smoking_status'].value_counts()
```

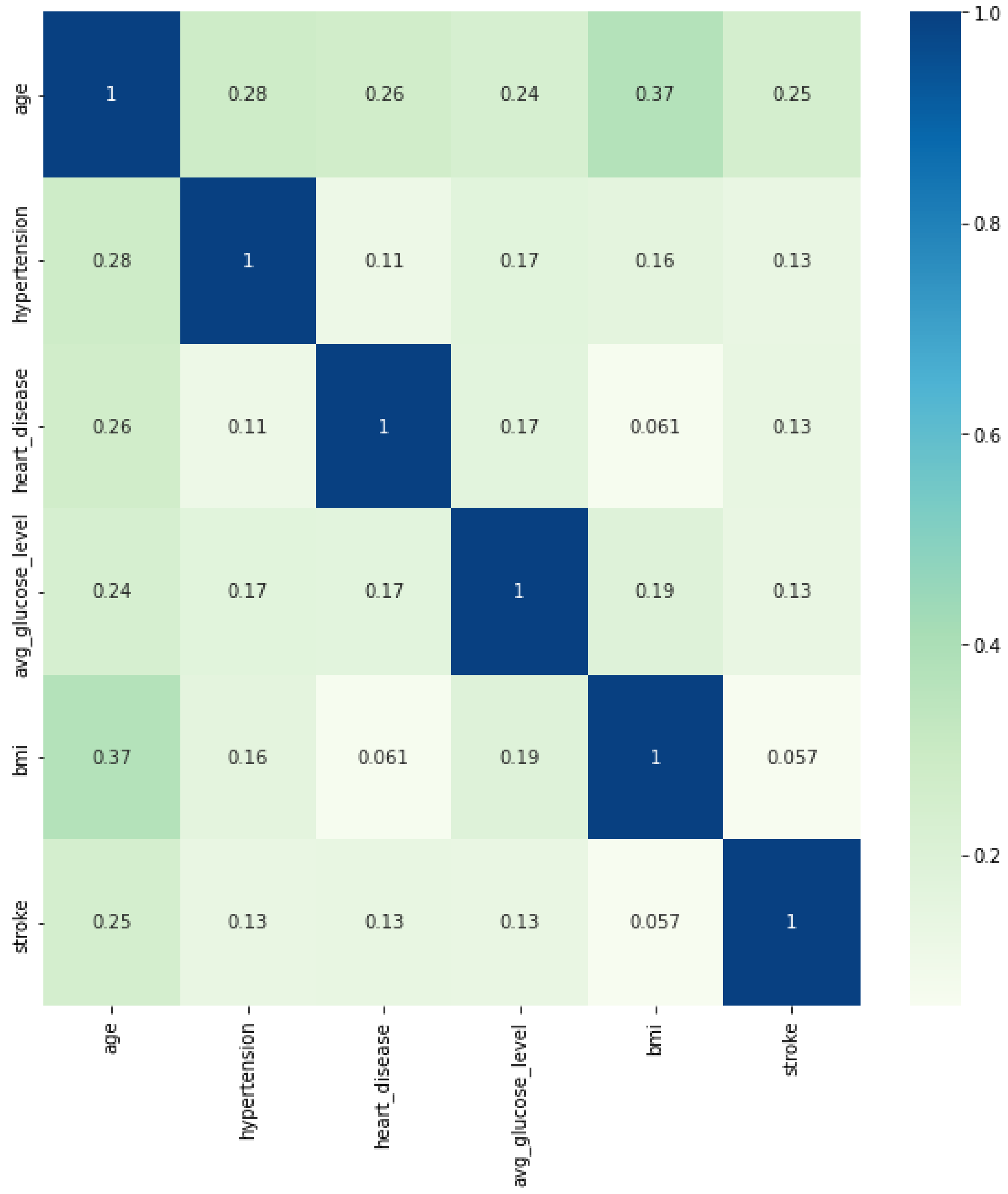
Out[4]:
```
never smoked       1838
Unknown            1500
formerly smoked     867
smokes              776
Name: smoking_status, dtype: int64
```

In [5]:
```
df.columns
```

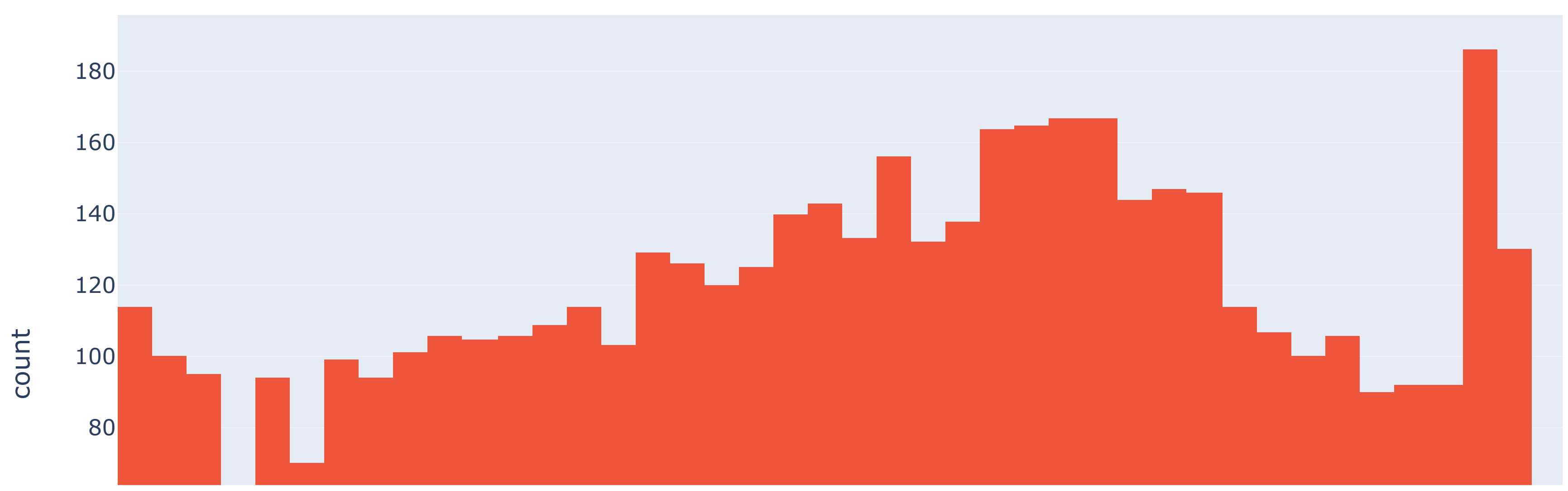
Out[5]:
```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [6]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot = True,cmap = 'GnBu')
```

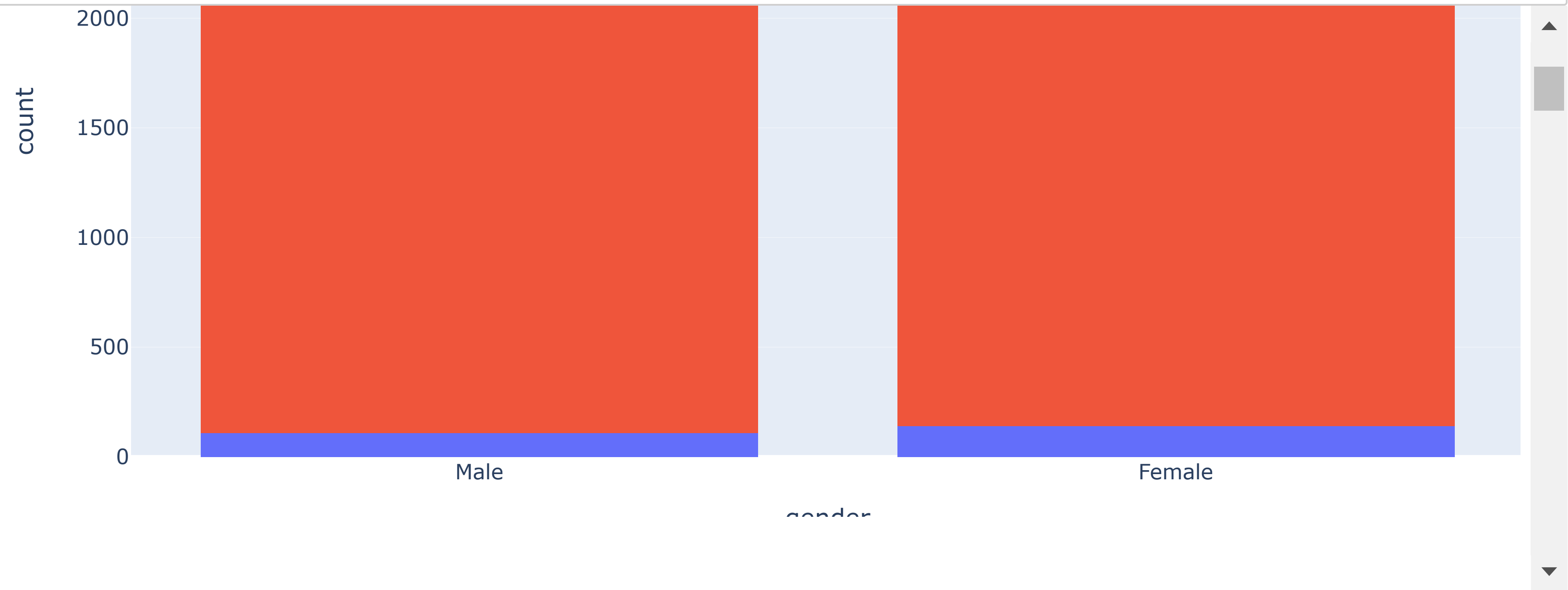
Out[6]: <AxesSubplot:>



In [7]:
```
fig = px.histogram(df, x="age", color="stroke")
fig.show()
```



In [8]:
```
features = ['gender','work_type','smoking_status','hypertension','heart_disease','ever_married',
            'Residence_type']
for feature in features:
    plt.figure(figsize=(3,3))
    fig = px.histogram(df, x=feature,color="stroke")
    fig.show()
```



Important Observations :

- There are total of 11 attributes and 5182 rows in data.
- Data contains no missing value.
- Stroke is the Target Variable.
- Age and BMI have the highest positive correlation.
- Percentage of Males experiencing Stroke is higher than Females.
- People who have had some kind of Heart Disease have higher risk of experiencing a Brain Stroke.
- Percentage of Males experiencing Stroke is higher than Females.
- Married Couples have higher risk of Brain Stroke.
- Those who live in Urban have a greater risk of Brain Stroke than those who live in Rural areas. Maybe because of hectic lifestyle in Urban Areas.
- Percentage of Males experiencing Stroke is higher than Females.
- If a person has condition of Hypertension then he/she has higher higher chance of getting Brain Stroke.
- Smokers have higher risk of having a Brain Stroke.

In [9]:
```
def married(x):
  if x=='Yes':
    return 1
  else:
    return 0
def residance(x):
  if x=='Urban':
    return 1
  else:
    return 0
def smoke(x):
  if x=='formerly smoked' or x=='smokes':
    return 1
  else:
    return 0
```

In [10]:
```
df['ever_married']=df['ever_married'].apply(lambda x:married(x))
df['smoking_status']=df['smoking_status'].apply(lambda x:smoke(x))
df['Residence_type'] = df['Residence_type'].apply(lambda x:residance(x))
```

In [11]:
```
df['gender'] = df['gender'].map({'Female':1,'Male':0})
df['work_type'] = df['work_type'].map({'Private': 0, 'Self-employed': 1, 'Govt_job':2, 'children':3})
```

In [12]:
```
df.head()
```

Out[12]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	67.0	0	1	1	0	1	228.69	36.6	1	1
1	0	80.0	0	1	1	0	0	105.92	32.5	0	1
2	1	49.0	0	0	1	0	1	171.23	34.4	1	1
3	1	79.0	1	0	1	1	0	174.12	24.0	0	1
4	0	81.0	0	0	1	0	1	186.21	29.0	1	1

In [13]:
```
X = df.iloc[:,:-1]
Y = df['stroke']
```

Oversampling for to handle imbalanced data set

In [14]:
```
smote=SMOTE()
os_X,os_Y=smote.fit_resample(X,Y)
print('Before sampling class distribution:',Counter(Y))
print('After sampling class distribution:',Counter(os_Y))
```

```
Before sampling class distribution: Counter({0: 4733, 1: 248})
After sampling class distribution: Counter({1: 4733, 0: 4733})
```

In [15]:
```
X_train, X_test, Y_train, Y_test = train_test_split(os_X,os_Y,test_size = 0.25, random_state=42,stratify=os_Y)
```

In [16]:
```
# Check the shape of train dataset
print(X_train.shape,Y_train.shape)

# Check the shape of test dataset
print(X_test.shape, Y_test.shape)
```

```
(7099, 10) (7099,)
(2367, 10) (2367,)
```

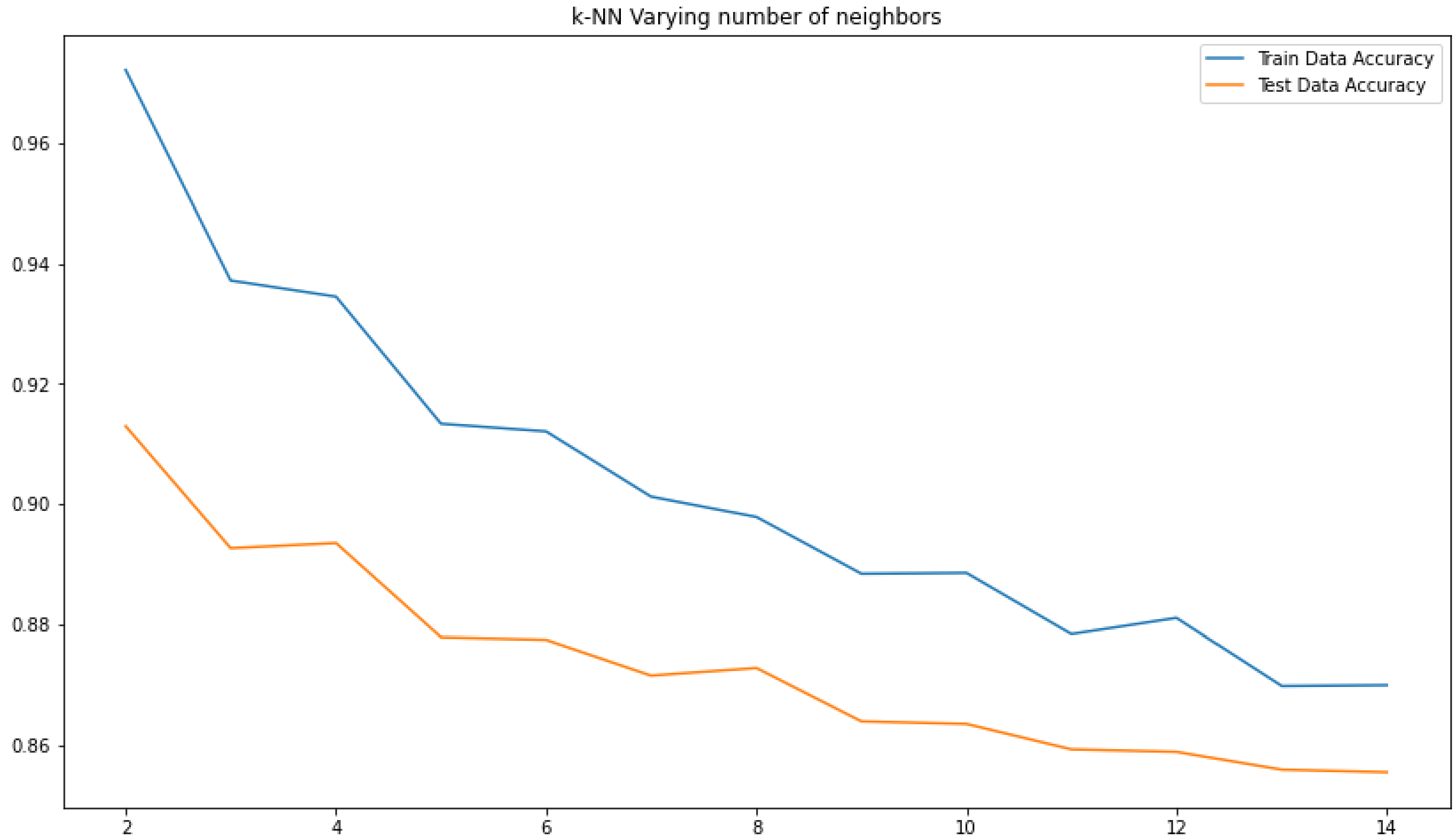
In [17]:
```
# minmax scalar gives less acuracy
from sklearn.preprocessing import StandardScaler
scale=StandardScaler()
X_train=scale.fit_transform(X_train)
X_test=scale.transform(X_test)
```

KNN model

In [18]:
```
test_acc=[]
train_acc=[]

for i in range(2,15):
  knn = KNeighborsClassifier(i) #setting up a knn classifier
  knn.fit(X_train,Y_train) #fitting the model
  # computing the accuracy for both the trainig and the test data
  train_acc.append(knn.score(X_train,Y_train))
  test_acc.append(knn.score(X_test,Y_test))
```

In [19]:
```
plt.figure(figsize=(14,8))
plt.title('k-NN Varying number of neighbors')
sns.lineplot(range(2,15),train_acc,label='Train Data Accuracy')
sns.lineplot(range(2,15),test_acc,label='Test Data Accuracy')
plt.show()
```



In [20]:
```
param_grid = {'n_neighbors':np.arange(2,7)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X_train,Y_train)
knn_cv.best_params_
```

Out[20]:
```
{'n_neighbors': 2}
```

In [21]:
```
test_class_preds=knn_cv.predict(X_test)
train_class_preds=knn_cv.predict(X_train)
```

In [22]:
```
# Calculating accuracy on train and test
train_accuracy = accuracy_score(Y_train,train_class_preds)
test_accuracy = accuracy_score(Y_test,test_class_preds)

#print("The accuracy on train dataset is", train_accuracy)
print("The accuracy on test dataset is", test_accuracy)
print("The accuracy on train dataset is", train_accuracy)
```

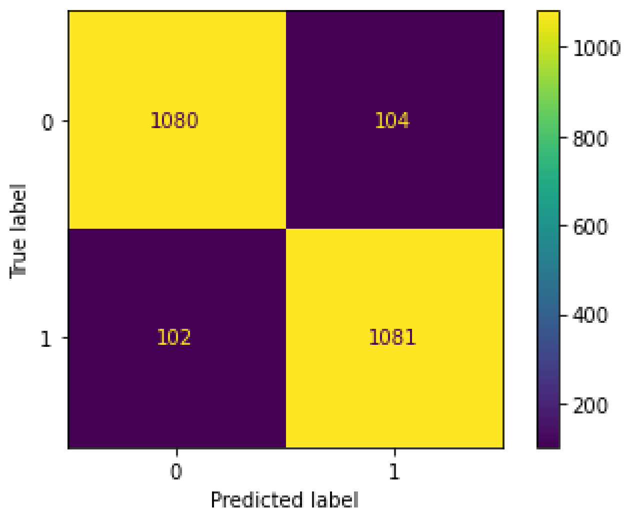
```
The accuracy on test dataset is 0.912970004224757
The accuracy on train dataset is 0.9721087477109452
```

In [23]:
```
cm1 = confusion_matrix(Y_test, test_class_preds)
print(cm1)
```

```
[[1080  104]
 [ 102 1081]]
```

In [24]:
```
cm = confusion_matrix(Y_test,test_class_preds, labels =knn_cv.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = cm1, display_labels =knn_cv.classes_)
disp.plot()
```

Out[24]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f497efa60>
```



In [25]:
```
clf_report = classification_report(Y_test, test_class_preds)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      1184
           1       0.91      0.91      0.91      1183

    accuracy                           0.91      2367
   macro avg       0.91      0.91      0.91      2367
weighted avg       0.91      0.91      0.91      2367
```

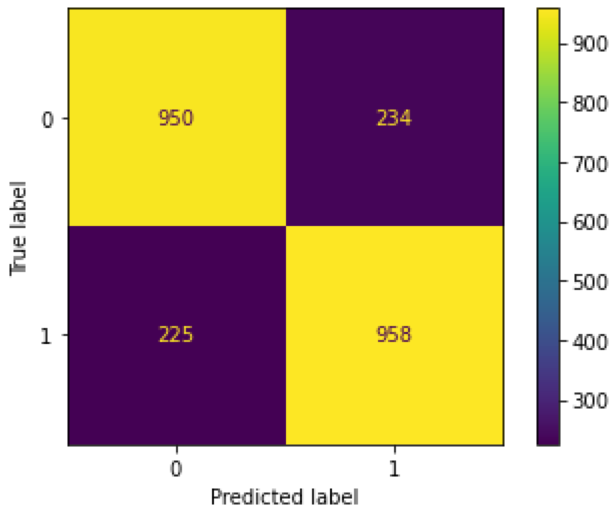
Logistic Regression

In [26]:
```
lr= LogisticRegression()

lr.fit(X_train, Y_train)
pred= lr.predict(X_test)
```

In [27]:
```
cm = confusion_matrix(Y_test, pred, labels = lr.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = lr.classes_)
disp.plot()
```

Out[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f4980fe20>
```



In [28]:
```
clf_report = classification_report(Y_test, pred)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.81      0.80      0.81      1184
           1       0.80      0.81      0.81      1183

    accuracy                           0.81      2367
   macro avg       0.81      0.81      0.81      2367
weighted avg       0.81      0.81      0.81      2367
```

Random forest

In [29]:
```
rfc= RandomForestClassifier()
param_grid={'n_estimators':[100,150,130]}
rf=GridSearchCV(rfc,param_grid,cv=5)
rf.fit(X_train,Y_train)
rf_pred= rf.predict(X_test)
rf.best_params_
```

Out[29]:
```
{'n_estimators': 130}
```

In [30]:
```
Y_test[:10]
```

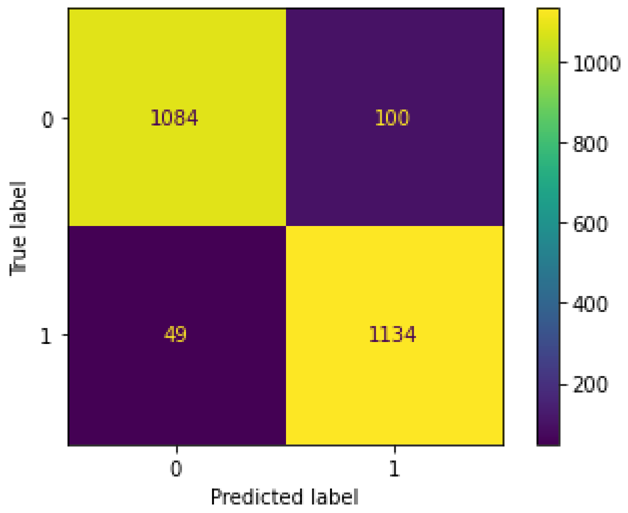
Out[30]:
```
1998    0
9169    1
1368    0
6640    1
6344    1
674     0
8867    1
7040    1
6675    1
7664    1
Name: stroke, dtype: int64
```

In [31]:
```
rf_pred[:10]
```

Out[31]:
```
array([0, 1, 0, 1, 0, 0, 1, 1, 1, 1], dtype=int64)
```

In [32]:
```
cm = confusion_matrix(Y_test, rf_pred, labels = rf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = rf.classes_)
disp.plot()
```

Out[32]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f4ab40b20>



In [33]:
```
clf_report = classification_report(Y_test, rf_pred)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      1184
           1       0.92      0.96      0.94      1183

    accuracy                           0.94      2367
   macro avg       0.94      0.94      0.94      2367
weighted avg       0.94      0.94      0.94      2367
```

SVM classifier

In [34]:
```
svc=SVC(kernel='rbf')
svc.fit(X_train,Y_train)
svc_pred=svc.predict(X_test)
```

In [35]:
```
clf_report = classification_report(Y_test, svc_pred)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.87      0.82      0.84      1184
           1       0.83      0.88      0.85      1183

    accuracy                           0.85      2367
   macro avg       0.85      0.85      0.85      2367
weighted avg       0.85      0.85      0.85      2367
```

Decision Tree classifier

In [36]:
```
dt= DecisionTreeClassifier()
dt.fit(X_train,Y_train)
dt_pred= dt.predict(X_test)
```

In [37]:
```
cm = confusion_matrix(Y_test,dt_pred)
print(cm)
```

```
[[1040  144]
 [  95 1088]]
```

In [38]:
```
clf_report = classification_report(Y_test,dt_pred)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90      1184
           1       0.88      0.92      0.90      1183

    accuracy                           0.90      2367
   macro avg       0.90      0.90      0.90      2367
weighted avg       0.90      0.90      0.90      2367
```

XGBClassifier

In [39]:
```
from xgboost import XGBClassifier
```

In [40]:
```
xgb= XGBClassifier()
# Fit
xgb.fit(X_train,Y_train)
xgb_pred= xgb.predict(X_test)
```

In [41]:
```
cm = confusion_matrix(Y_test,xgb_pred)
print(cm)
```

```
[[1135   49]
 [  54 1129]]
```

In [42]:
```
clf_report = classification_report(Y_test,xgb_pred)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      1184
           1       0.96      0.95      0.96      1183

    accuracy                           0.96      2367
   macro avg       0.96      0.96      0.96      2367
weighted avg       0.96      0.96      0.96      2367
```