In [1]:
```
import pandas as pd
import numpy as np
import sklearn as sk
```

In [2]:
```
dataset=pd.read_csv(r'C:\Users\HP\Downloads\Brain Stroke.csv')
```

In [3]:
```
dataset
```

Out[3]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gl
0	Male	67.0	0	1	Yes	Private	Urban	
1	Male	80.0	0	1	Yes	Private	Rural	
2	Female	49.0	0	0	Yes	Private	Urban	
3	Female	79.0	1	0	Yes	Self-employed	Rural	
4	Male	81.0	0	0	Yes	Private	Urban	
...	...	...	...	...	...	...	...	
4976	Male	41.0	0	0	No	Private	Rural	
4977	Male	40.0	0	0	Yes	Private	Urban	
4978	Female	45.0	1	0	Yes	Govt_job	Rural	
4979	Male	40.0	0	0	Yes	Private	Rural	
4980	Female	80.0	1	0	Yes	Private	Urban	

4981 rows × 11 columns

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
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 [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4981 entries, 0 to 4980
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             4981 non-null   object 
 1   age                4981 non-null   float64
 2   hypertension       4981 non-null   int64  
 3   heart_disease      4981 non-null   int64  
 4   ever_married       4981 non-null   object 
 5   work_type          4981 non-null   object 
 6   Residence_type     4981 non-null   object 
 7   avg_glucose_level  4981 non-null   float64
 8   bmi                4981 non-null   float64
 9   smoking_status     4981 non-null   object 
 10  stroke             4981 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
```

In [6]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset["gender"]=le.fit_transform(dataset["gender"])
dataset["ever_married"]=le.fit_transform(dataset["ever_married"])
dataset["work_type"]=le.fit_transform(dataset["work_type"])
dataset["Residence_type"]=le.fit_transform(dataset["Residence_type"])
dataset["smoking_status"]=le.fit_transform(dataset["smoking_status"])
```

In [7]:
```
dataset
```

Out[7]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gl
0	1	67.0	0	1	1	1	1	
1	1	80.0	0	1	1	1	0	
2	0	49.0	0	0	1	1	1	
3	0	79.0	1	0	1	2	0	
4	1	81.0	0	0	1	1	1	
...	...	...	...	...	...	...	...	
4976	1	41.0	0	0	0	1	0	
4977	1	40.0	0	0	1	1	1	
4978	0	45.0	1	0	1	0	0	
4979	1	40.0	0	0	1	1	0	
4980	0	80.0	1	0	1	1	1	

4981 rows × 11 columns

In [8]:
```
x=dataset.iloc[:,:-1].values  #slicing independent and depedent variables.
y=dataset.iloc[:,-1].values
x,y
```

Out[8]:
```
(array([[  1.  ,  67.  ,   0.  , ..., 228.69,  36.6 ,   1.  ],
        [  1.  ,  80.  ,   0.  , ..., 105.92,  32.5 ,   2.  ],
        [  0.  ,  49.  ,   0.  , ..., 171.23,  34.4 ,   3.  ],
        ...,
        [  0.  ,  45.  ,   1.  , ...,  95.02,  31.8 ,   3.  ],
        [  1.  ,  40.  ,   0.  , ...,  83.94,  30.  ,   3.  ],
        [  0.  ,  80.  ,   1.  , ...,  83.75,  29.1 ,   2.  ]]),
 array([1, 1, 1, ..., 0, 0, 0], dtype=int64))
```

In [9]:
```
dataset["stroke"].value_counts()  #Checking Balancing data or not of the depender
```

Out[9]:
```
0    4733
1     248
Name: stroke, dtype: int64
```

In [10]:
```
#balanceing the data
from imblearn.over_sampling import RandomOverSampler
ab=RandomOverSampler()
x_data, y_data =ab.fit_resample(x,y)
```

In [11]:
```
!pip install -U imbalanced-learn
```

```
Requirement already up-to-date: imbalanced-learn in c:\users\hp\anaconda3\lib\s
ite-packages (0.9.1)
Requirement already satisfied, skipping upgrade: numpy>=1.17.3 in c:\users\hp\a
naconda3\lib\site-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied, skipping upgrade: joblib>=1.0.0 in c:\users\hp\a
naconda3\lib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied, skipping upgrade: threadpoolctl>=2.0.0 in c:\use
rs\hp\anaconda3\lib\site-packages (from imbalanced-learn) (2.1.0)
Requirement already satisfied, skipping upgrade: scikit-learn>=1.1.0 in c:\user
s\hp\anaconda3\lib\site-packages (from imbalanced-learn) (1.1.3)
Requirement already satisfied, skipping upgrade: scipy>=1.3.2 in c:\users\hp\an
aconda3\lib\site-packages (from imbalanced-learn) (1.5.2)
```

In [12]:
```
from collections import Counter
print(Counter(y_data))
```

```
Counter({1: 4733, 0: 4733})
```

In [13]:
```
dataset.corr()
```

Out[13]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Res
gender	1.000000	-0.026538	0.021485	0.086476	-0.028971	0.065784	
age	-0.026538	1.000000	0.278120	0.264852	0.677137	-0.415935	
hypertension	0.021485	0.278120	1.000000	0.111974	0.164534	-0.061618	
heart_disease	0.086476	0.264852	0.111974	1.000000	0.114765	-0.036943	
ever_married	-0.028971	0.677137	0.164534	0.114765	1.000000	-0.406439	
work_type	0.065784	-0.415935	-0.061618	-0.036943	-0.406439	1.000000	
Residence_type	-0.004301	0.017155	-0.004755	0.002125	0.008191	-0.003524	
avg_glucose_level	0.055796	0.236763	0.170028	0.166847	0.150724	-0.059658	
bmi	-0.012093	0.373703	0.158762	0.060926	0.371690	-0.382418	
smoking_status	-0.062666	0.265623	0.110045	0.048093	0.262384	-0.356738	
stroke	0.008870	0.246478	0.131965	0.134610	0.108398	-0.041835	

In [14]:
```
# Normalization
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
x_data1=scaler.fit_transform(x_data)
x_data1
```

Out[14]:
```
array([[1.        , 0.81689453, 0.        , ..., 0.80126489, 0.64756447,
        0.33333333],
       [1.        , 0.97558594, 0.        , ..., 0.23451205, 0.53008596,
        0.66666667],
       [0.        , 0.59716797, 0.        , ..., 0.53600776, 0.58452722,
        1.        ],
       ...,
       [0.        , 0.82910156, 0.        , ..., 0.1245499 , 0.37535817,
        0.        ],
       [0.        , 0.93896484, 0.        , ..., 0.8145139 , 0.65616046,
        0.66666667],
       [1.        , 0.97558594, 0.        , ..., 0.06190564, 0.29226361,
        1.        ]])
```

In [15]:
```
x_data1.max()    # Max value after normalization
```

Out[15]: 1.0

In [16]:
```
x_data1.min()    # Min value after normalization
```

Out[16]: 0.0

In [17]:
```
 #Standardization
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_data1=ss.fit_transform(x)
x_data1
```

Out[17]:
```
array([[ 1.18390850e+00,  1.04058433e+00, -3.26185770e-01, ...,
         2.72341090e+00,  1.19323816e+00, -3.53933192e-01],
       [ 1.18390850e+00,  1.61427033e+00, -3.26185770e-01, ...,
        -5.22766599e-04,  5.89389611e-01,  5.78839946e-01],
       [-8.44659868e-01,  2.46249882e-01, -3.26185770e-01, ...,
         1.44852918e+00,  8.69221866e-01,  1.51161308e+00],
       ...,
       [-8.44659868e-01,  6.97311148e-02,  3.06573766e+00, ...,
        -2.42364234e-01,  4.86293516e-01,  1.51161308e+00],
       [ 1.18390850e+00, -1.50917344e-01, -3.26185770e-01, ...,
        -4.88199415e-01,  2.21189274e-01,  1.51161308e+00],
       [-8.44659868e-01,  1.61427033e+00,  3.06573766e+00, ...,
        -4.92415000e-01,  8.86371531e-02,  5.78839946e-01]])
```

In [18]:
```
x_data1.max()    # Max value after Standardization
```

Out[18]:
```
4.136753228405977
```

In [19]:
```
x_data1.min()    # Min value after Standardization
```

Out[19]:
```
-2.1352928787181042
```

In [20]:
```
# Spliting data into training testing
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data,y_data,test_size=0.30,
```

In [21]:
```
print(Counter(y_train))
```

```
Counter({1: 3332, 0: 3294})
```

In [22]:
```
print(Counter(y_test))
```

```
Counter({0: 1439, 1: 1401})
```

In [23]:
```
#model fitting algorithum
# 1. Logistic Regression
from sklearn.linear_model import LogisticRegression
l1=LogisticRegression()
lr=l1.fit(x_train,y_train)
lr
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:444:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

Out[23]:
```
▾ LogisticRegression
LogisticRegression()
```

In [24]:
```
y_pred=l1.predict(x_test)
y_pred
```

Out[24]:
```
array([0, 1, 1, ..., 0, 1, 0], dtype=int64)
```

In [25]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_pred,y_test))
```

```
[[1061  262]
 [ 378 1139]]
```

In [26]:
```
from sklearn.metrics import accuracy_score
```

In [27]:
```
print(accuracy_score(y_pred,y_test)*100)
```

```
77.46478873239437
```

In [28]:
```
# 2. support vector machine
from sklearn.svm import SVC
s1=SVC(kernel='linear',random_state=5)
an=s1.fit(x_train,y_train)
an
```

Out[28]:
```
▾ SVC
SVC(kernel='linear', random_state=5)
```

In [29]:
```
y_pred1=s1.predict(x_test)
y_pred1
```

Out[29]:
```
array([0, 1, 1, ..., 0, 1, 0], dtype=int64)
```

In [30]:
```
an1=accuracy_score(y_test,y_pred)*100
an1
```

Out[30]:
```
77.46478873239437
```

In [31]:
```
an2=confusion_matrix(y_test,y_pred1)
an2
```

Out[31]:
```
array([[1030,  409],
       [ 240, 1161]], dtype=int64)
```

In [32]:
```
#3. K-nearrest neighbors
from sklearn.neighbors import KNeighborsClassifier
k1=KNeighborsClassifier(n_neighbors=5,metric="euclidean",p=1)
ks=k1.fit(x_train,y_train)
ks
```

Out[32]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='euclidean', p=1)
```

In [33]:
```
y_pred2=k1.predict(x_test)
y_pred2
```

Out[33]:
```
array([1, 1, 1, ..., 0, 0, 1], dtype=int64)
```

In [34]:
```
ks1=accuracy_score(y_test,y_pred)*100
ks1
```

Out[34]:
```
77.46478873239437
```

In [35]:
```
ks2=confusion_matrix(y_test,y_pred1)
ks2
```

Out[35]:
```
array([[1030,  409],
       [ 240, 1161]], dtype=int64)
```

In [36]:
```
#4.Random forest
from sklearn.ensemble import RandomForestClassifier
r1=RandomForestClassifier(n_estimators=5,criterion="entropy",max_depth=3,random_s
rs=r1.fit(x_train,y_train)
rs
```

Out[36]:
```
RandomForestClassifier

RandomForestClassifier(criterion='entropy', max_depth=3, n_estimators=5,
                       random_state=10)
```

In [37]:
```
y_pred3=r1.predict(x_test)
y_pred3
```

Out[37]:
```
array([1, 1, 1, ..., 0, 1, 0], dtype=int64)
```

In [38]:
```
rs1=accuracy_score(y_test,y_pred)*100
rs1
```

Out[38]:
```
77.46478873239437
```

In [39]:
```
rs2=confusion_matrix(y_test,y_pred1)
rs2
```

Out[39]:
```
array([[1030,  409],
       [ 240, 1161]], dtype=int64)
```

In [40]:
```
#5.Navie baysien
from sklearn.naive_bayes import GaussianNB
n1=GaussianNB()
nb=n1.fit(x_train,y_train)
nb
```

Out[40]:
```
GaussianNB

GaussianNB()
```

In [41]:
```
y_pred4=n1.predict(x_test)
y_pred4
```

Out[41]:
```
array([0, 1, 1, ..., 0, 1, 0], dtype=int64)
```

In [42]:
```
nb1=accuracy_score(y_test,y_pred)*100
nb1
```

Out[42]:
```
77.46478873239437
```

In [43]:
```
nb2=confusion_matrix(y_test,y_pred1)
nb2
```

Out[43]:
```
array([[1030,  409],
       [ 240, 1161]], dtype=int64)
```

In [44]:
```
#6. Decision tree
from sklearn.tree import DecisionTreeClassifier
d1=DecisionTreeClassifier(random_state=5,max_depth=3)
dt=d1.fit(x_train,y_train)
dt
```

Out[44]:
```
DecisionTreeClassifier

DecisionTreeClassifier(max_depth=3, random_state=5)
```

In [45]:
```
y_pred5=d1.predict(x_test)
y_pred5
```

Out[45]:
```
array([1, 1, 1, ..., 0, 1, 0], dtype=int64)
```

In [46]:
```
dt1=accuracy_score(y_test,y_pred)*100
dt1
```

Out[46]:
```
77.46478873239437
```

In [47]:
```
dt2=confusion_matrix(y_test,y_pred1)
dt2
```

Out[47]:
```
array([[1030,  409],
       [ 240, 1161]], dtype=int64)
```

In [48]:
```
#    Cross Validation Technique
# 1) Leave one out cross validation technique
# 2) K Fold Cross Validation
# 3) Stratified K Fold Cross Validation
```

In [49]:
```
#2) K Fold Cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
kf= KFold(n_splits=5, random_state=13, shuffle=True)
kf.get_n_splits(x_data1,y_data)
print(kf)
```

```
KFold(n_splits=5, random_state=13, shuffle=True)
```

In [50]:
```
for train_data,test_data in kf.split(x_data):
    x_train,x_test=x_data[train_data],x_data[test_data]
    y_train,y_test=y_data[train_data],y_data[test_data]

    print("check y_test balance",Counter(y_test))

    scores=cross_val_score(dt,x_train,y_train,cv=kf)
    y_pred6=cross_val_predict(dt,x_test,y_test)

    print(y_pred6)
    print(scores)

    k1=accuracy_score(y_test,y_pred6)*100
    print(k1,"Accuracy score on testing set")
```

```
check y_test balance Counter({1: 970, 0: 924})
[1 1 1 ... 0 1 1]
[0.78679868 0.78547855 0.78467635 0.75825627 0.77344782]
77.61351636747624 Accuracy score on testing set
check y_test balance Counter({0: 998, 1: 895})
[1 1 1 ... 1 1 0]
[0.79207921 0.77227723 0.78283828 0.79260238 0.7681638 ]
77.33755942947703 Accuracy score on testing set
check y_test balance Counter({0: 951, 1: 942})
[1 1 1 ... 1 1 0]
[0.76435644 0.76633663 0.79207921 0.7661823  0.77410832]
77.54886423666139 Accuracy score on testing set
check y_test balance Counter({1: 974, 0: 919})
[1 1 1 ... 1 1 1]
[0.77953795 0.79207921 0.78481848 0.77741083 0.74900925]
79.55625990491284 Accuracy score on testing set
check y_test balance Counter({1: 952, 0: 941})
[1 1 1 ... 1 1 1]
[0.75709571 0.77689769 0.77425743 0.78071334 0.75759577]
77.17908082408876 Accuracy score on testing set
```

In [51]:
```
#3). stratified k-fold
```

In [52]:
```
from sklearn.model_selection import StratifiedKFold
skf=StratifiedKFold(n_splits=5, shuffle=True, random_state=10)
skf.get_n_splits(x_data,y_data)
print(skf)
```

```
StratifiedKFold(n_splits=5, random_state=10, shuffle=True)
```

In [53]:
```
for train_data,test_data in kf.split(x_data,y_data):
    x_train1,x_test1=x_data[train_data],x_data[test_data]
    y_train1,y_test1=y_data[train_data],y_data[test_data]

    print("check y_test balance",Counter(y_test1))

    scores1=cross_val_score(dt,x_train1,y_train1,cv=kf)*100
    y_pred7=cross_val_predict(dt,x_test1,y_test1)

    print(y_pred7)
    print(scores1)
    print(np.mean(scores1))
    cb=accuracy_score(y_test,y_pred6)*100
    print(cb,"Accuracy score on testing set")
```

```
check y_test balance Counter({1: 970, 0: 924})
[1 1 1 ... 0 1 1]
[78.67986799 78.54785479 78.4676354  75.82562748 77.34478203]
77.7731535372824
77.17908082408876 Accuracy score on testing set
check y_test balance Counter({0: 998, 1: 895})
[1 1 1 ... 1 1 0]
[79.20792079 77.22772277 78.28382838 79.26023778 76.81638045]
78.15921803540989
77.17908082408876 Accuracy score on testing set
check y_test balance Counter({0: 951, 1: 942})
[1 1 1 ... 1 1 0]
[76.43564356 76.63366337 79.20792079 76.61822985 77.41083223]
77.26125796199172
77.17908082408876 Accuracy score on testing set
check y_test balance Counter({1: 974, 0: 919})
[1 1 1 ... 1 1 1]
[77.95379538 79.20792079 78.48184818 77.74108322 74.9009247 ]
77.65711445649188
77.17908082408876 Accuracy score on testing set
check y_test balance Counter({1: 952, 0: 941})
[1 1 1 ... 1 1 1]
[75.70957096 77.68976898 77.42574257 78.07133421 75.75957728]
76.93119880019707
77.17908082408876 Accuracy score on testing set
```

In [ ]:

In [ ]:

In [ ]: