FEDESORIANO

STROKE PREDICTION DATASET

www.kaggle.com

ที่มาและความสำคัญ

Stroke Prediction Dataset

องค์การอนามัยโลก (WHO) โรคหลอดเลือดสมองเป็นสาเหตุการตายอันดับสองทั่วโลกในปี 2562 มีผู้เสียชีวิต 5.9 ล้านคน หรือ 10.5% ของการเสียชีวิตทั้งหมด

About this data

The data contains 5110 observations with 12 attributes.

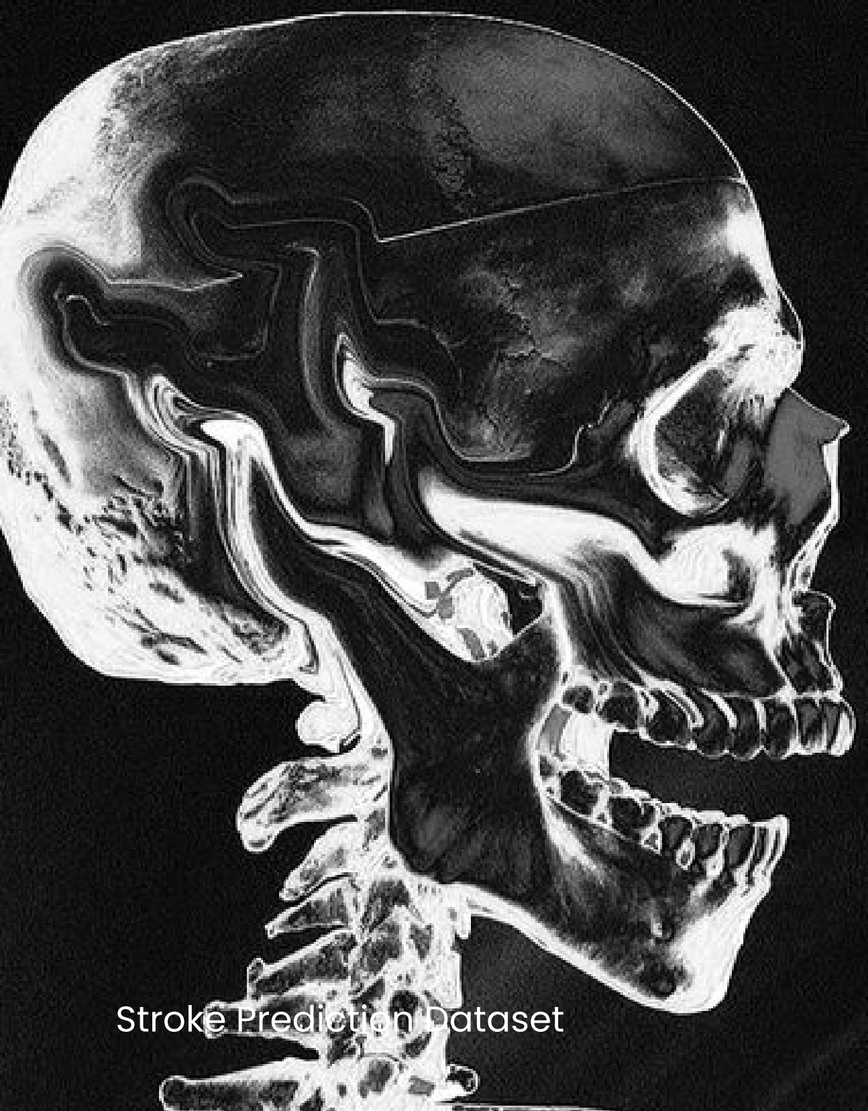
DATA

(5110, 12)

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
28]   1 len(set(stroke_data['id']))

      5110
```

DATA CLEANSING



Stroke Prediction Dataset

```
1 stroke_data.info()
```

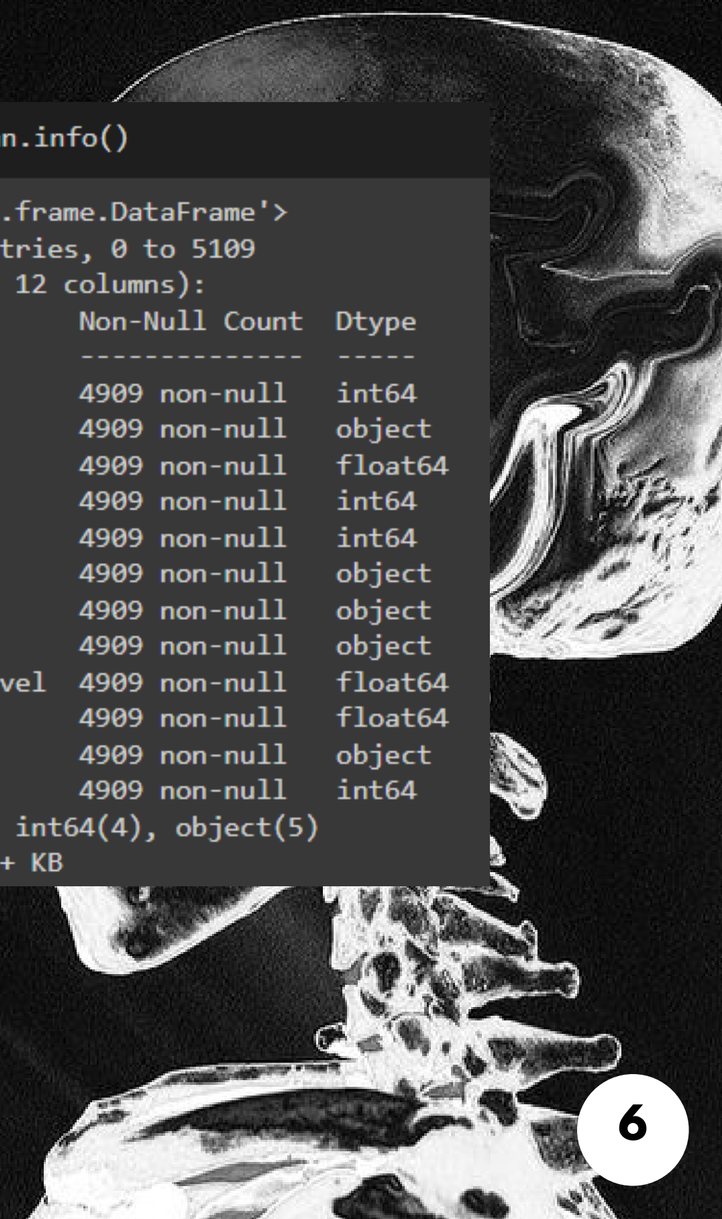
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 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
```

DATA CLEANSING

```
1 stroke_data_clean = stroke_data.dropna(subset =['bmi'])#ดรอป missing ของคอลัมน์ bmi
2 stroke_data_clean.head(3)
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_typ
0	9046	Male	67.0	0	1	Yes	Private	Urba
2	31112	Male	80.0	0	1	Yes	Private	Rur
3	60182	Female	49.0	0	0	Yes	Private	Urba

```
]   1 stroke_data_clean.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4909 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 4909 non-null   int64
 1   gender             4909 non-null   object
 2   age                4909 non-null   float64
 3   hypertension       4909 non-null   int64
 4   heart_disease      4909 non-null   int64
 5   ever_married       4909 non-null   object
 6   work_type          4909 non-null   object
 7   Residence_type     4909 non-null   object
 8   avg_glucose_level  4909 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     4909 non-null   object
 11  stroke             4909 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 498.6+ KB
```

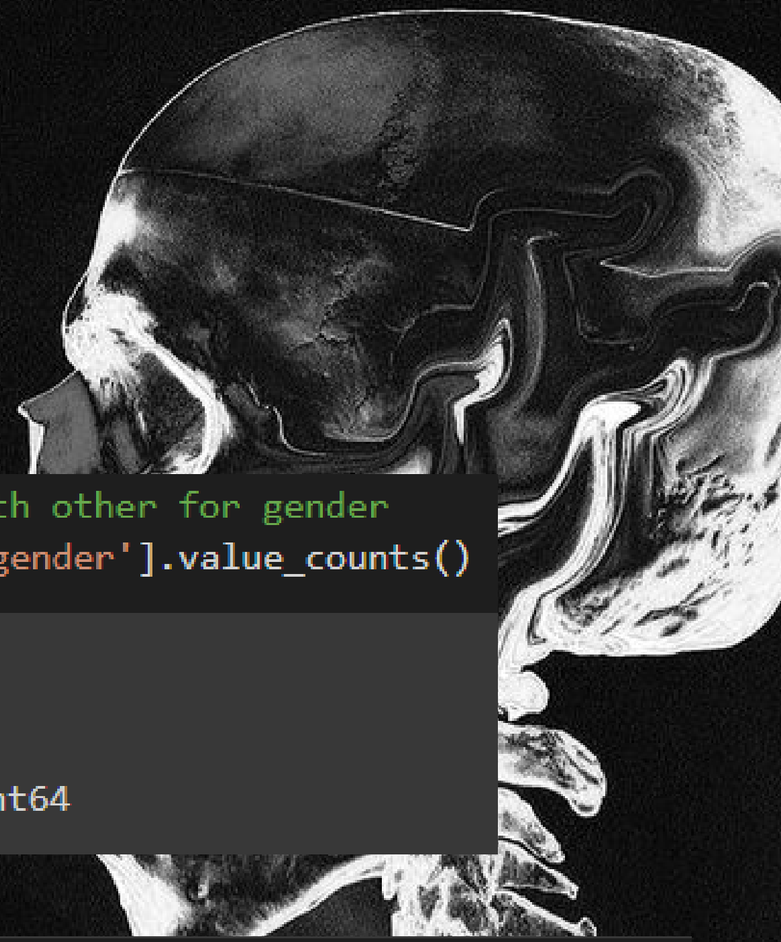


Stroke Prediction Dataset

DATA CLEANSING

```
]   1 stroke_data.nunique() #จำนวนข้อมูลที่ไม่ซ้ำกัน
```

```
id                   5110
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
```



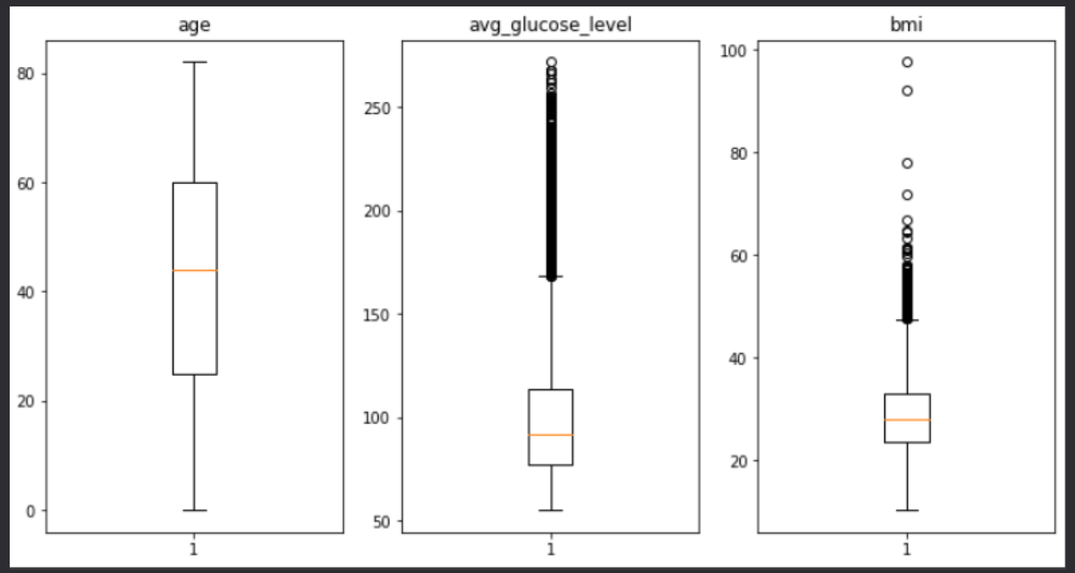
```
1 # drop 1 patient with other for gender
2 stroke_data_clean['gender'].value_counts()
```

```
Female    2897
Male      2011
Other        1
Name: gender, dtype: int64
```

```
1 stroke_data_clean = stroke_data_clean[stroke_data_clean.eq('Other').any(1)==False]
```

OUTLIER

```
1 import matplotlib.pyplot as plt
2 numerical_features = ["age", "avg_glucose_level", "bmi"]
3 # Create boxplots for each numerical feature
4 fig, axes = plt.subplots(nrows=1, ncols=len(numerical_features), figsize=(12, 6))
5 for i, feature in enumerate(numerical_features):
6     axes[i].boxplot(stroke_data_clean[feature])
7     axes[i].set_title(feature)
8 plt.show()
```




```
1 stroke_data_clean[['avg_glucose_level','bmi','age']].describe()
```

	avg_glucose_level	bmi	age
count	4908.000000	4908.00000	4908.000000
mean	105.297402	28.89456	42.868810
std	44.425550	7.85432	22.556128
min	55.120000	10.30000	0.080000
25%	77.067500	23.50000	25.000000
50%	91.680000	28.10000	44.000000
75%	113.495000	33.10000	60.000000
max	271.740000	97.60000	82.000000

www.kaggle.com

ตารางการแสดงผล BMI

การแปรผล	ค่า BMI	ภาวะแทรกซ้อน
น้ำหนักต่ำกว่าเกณฑ์	น้อยกว่า 18.5	เสี่ยงโรคขาดสารอาหาร
น้ำหนักสมส่วน	18.5 - 22.9	โอกาสมีโรคแทรกซ้อนน้อย
น้ำหนักเกินมาตรฐาน	23.0 - 24.9	ภาวะน้ำหนักเกินระยะเริ่มต้น
น้ำหนักอยู่ในเกณฑ์อ้วน	25.0 - 29.9	ภาวะน้ำหนักเกินมาก ระยะอ้วนเริ่มต้น
น้ำหนักอยู่ในเกณฑ์อ้วนมาก	มากกว่า 30	ภาวะน้ำหนักเกินมาก โรคอ้วน

ตารางแสดงระดับค่าน้ำตาลในเลือด (mg/dL)

ตารางแสดงระดับน้ำตาลในเลือด

ก่อนทานอาหาร	หลังทานอาหาร
70-99 ปกติ	< 140 ปกติ
100-125 เสี่ยงเป็นโรคเบาหวาน	140-199 เสี่ยงเป็นโรคเบาหวาน
>126 เป็นโรคเบาหวาน	>200 เป็นโรคเบาหวาน

Stroke Prediction Dataset

```
1 stroke_data_clean = stroke_data_clean[stroke_data_clean['bmi'] < 50]
2 stroke_data_clean.shape
```

```
(4829, 12)
```

```
1 stroke_data_clean = stroke_data_clean[stroke_data_clean['age'] > 1]
2 stroke_data_clean.shape
```

```
(4782, 12)
```

```
1 stroke_data_clean = stroke_data_clean[stroke_data_clean['avg_glucose_level'] > 70 ]
2 stroke_data_clean.shape
```

```
(4062, 12)
```

```
1 stroke_data_cleaned = stroke_data_clean.drop("id", axis=1) #ดรอปคอลัมน์ id
```

DATA

	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
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
5	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
5104	Female	13.0	0	0	No	children	Rural	103.08	18.6	Unknown	0
5106	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

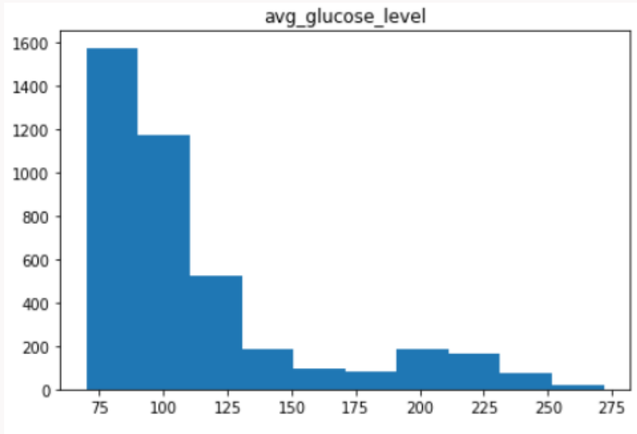
4062 rows × 11 columns

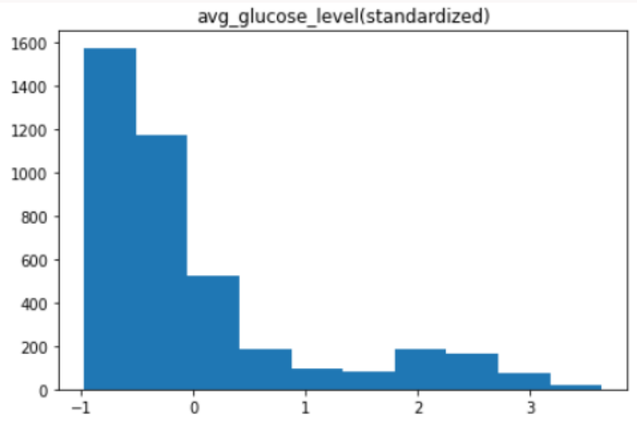
STROKE_DATA_CLEANED

PRE-PROCESSING DATA

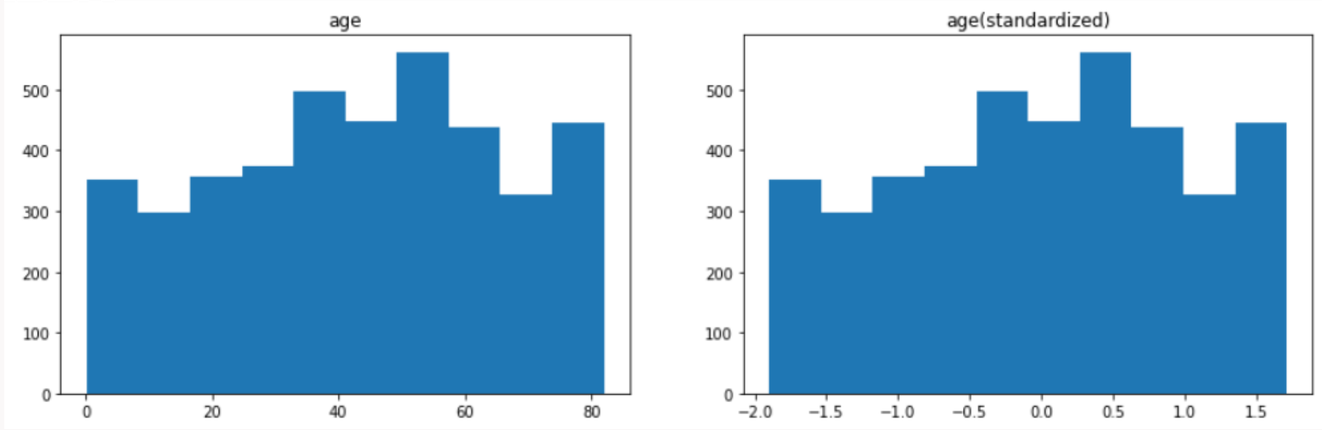
STANDARDSCALER

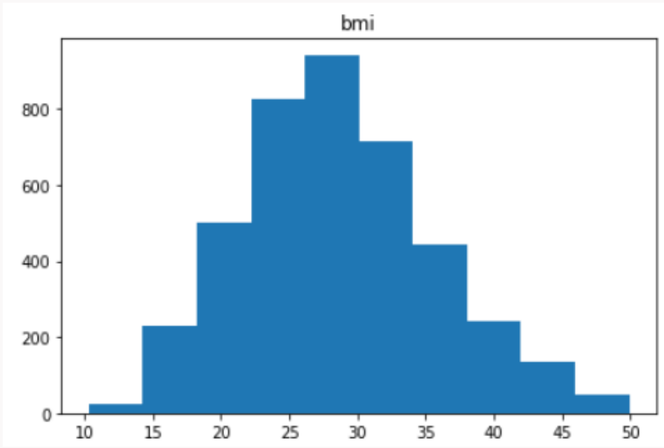
```python
from sklearn import preprocessing

numeric_features =  stroke_data_cleaned[['avg_glucose_level','age','bmi']]

#define บอกว่าวิธีการที่เราจะทำคืออะไร () เป็นพารามิเตอร์ที่เราใส่ได้
scaler = preprocessing.StandardScaler()
#train สร้างโมเดลจาก data
scaler.fit_transform(numeric_features)
#predict-transform
numeric_features_s = scaler.transform(numeric_features)
```

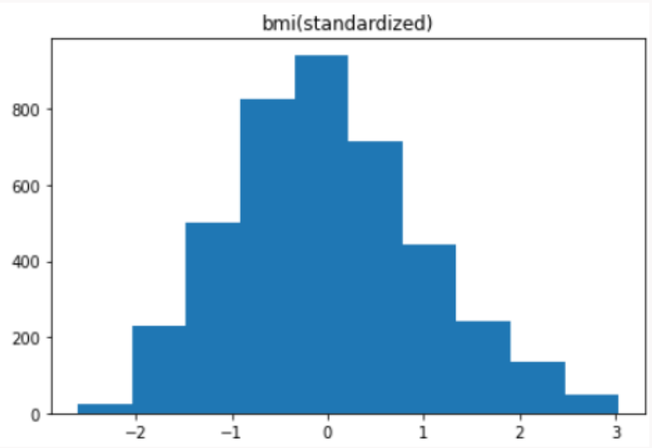




STANDARDSCALER







ONE-HOT-ENCODER

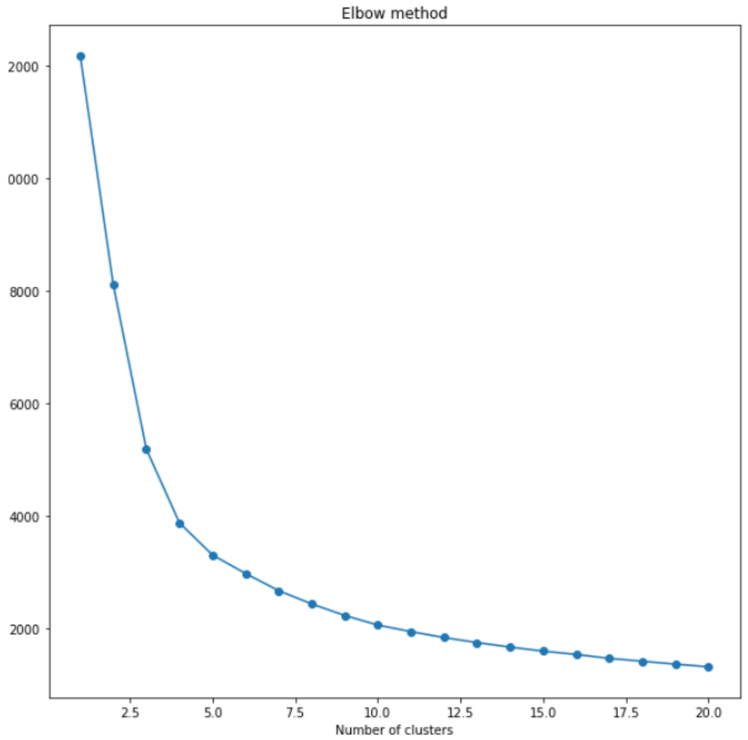
```
1 stroke_data_cleaned_dummies = pd.get_dummies(stroke_data_cleaned,columns=['gender','ever_married','work_type','smoking_status','Residence_type'])
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Female	gender_Male	ever_married_No	ever_married_Yes	...	work_type_Never_worked	w
0	67.0	0	1	228.69	36.6	1	0	1	0	1	...	0	
2	80.0	0	1	105.92	32.5	1	0	1	0	1	...	0	
3	49.0	0	0	171.23	34.4	1	1	0	0	1	...	0	
4	79.0	1	0	174.12	24.0	1	1	0	0	1	...	0	
5	81.0	0	0	186.21	29.0	1	0	1	0	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
5104	13.0	0	0	103.08	18.6	0	1	0	1	0	...	0	
5106	81.0	0	0	125.20	40.0	0	1	0	0	1	...	0	
5107	35.0	0	0	82.99	30.6	0	1	0	0	1	...	0	
5108	51.0	0	0	166.29	25.6	0	0	1	0	1	...	0	
5109	44.0	0	0	85.28	26.2	0	1	0	0	1	...	0	

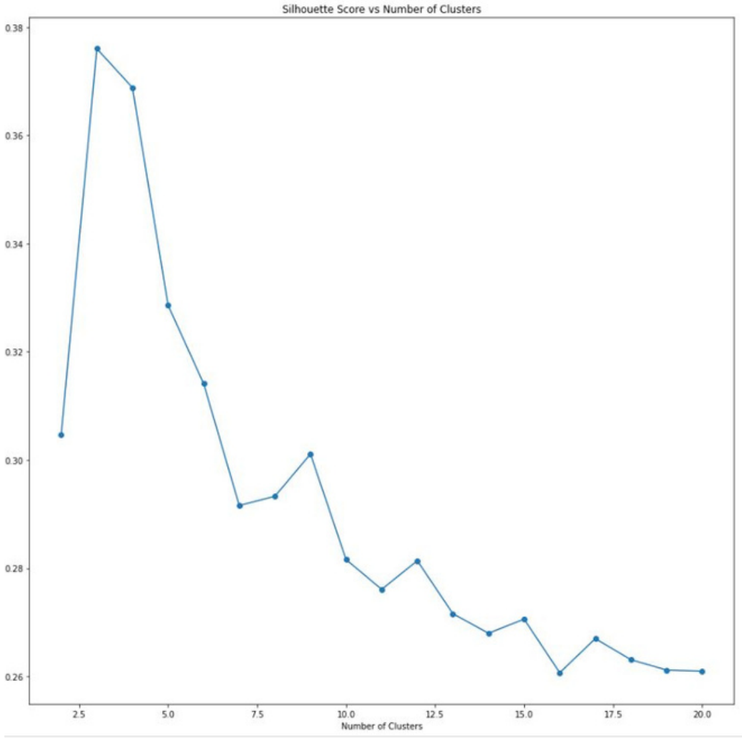
4062 rows × 21 columns

STROKE_DATA_CLEANED_DUMMIES

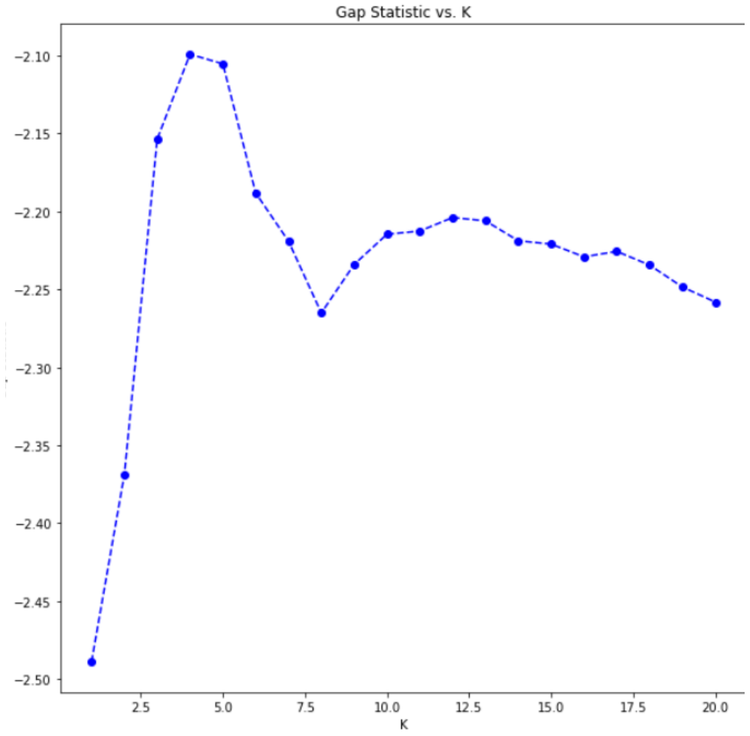
k-means clustering



OPTIMUM NUMBER OF CLUSTER BY ELBOW METHOD: 4



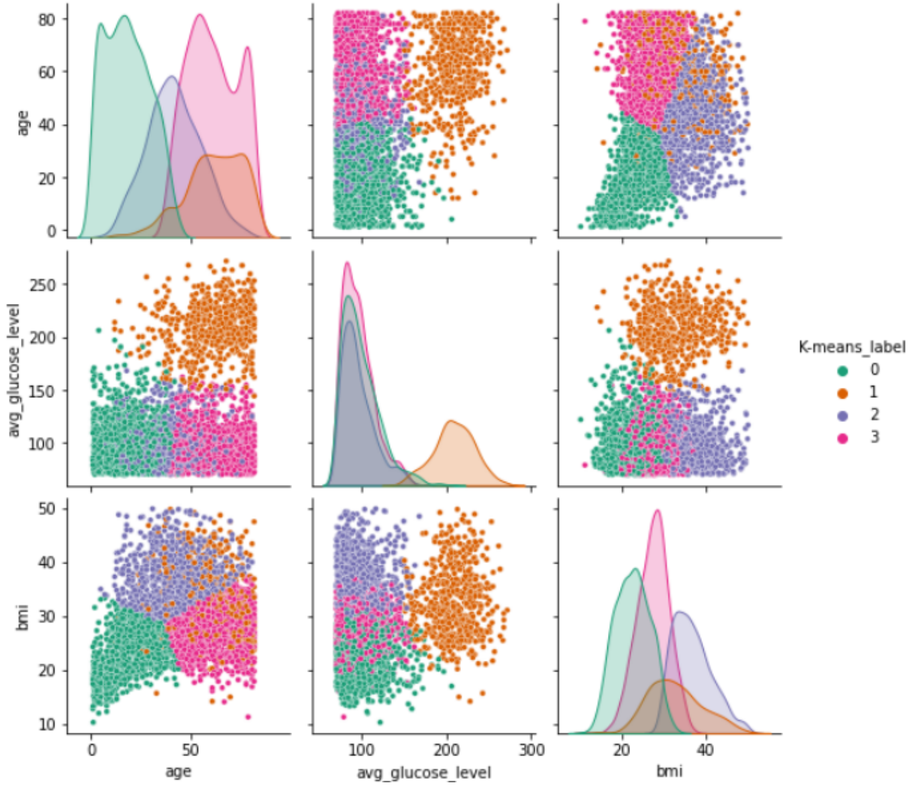
OPTIMUM NUMBER OF CLUSTER BY SILHOUETTE: 3



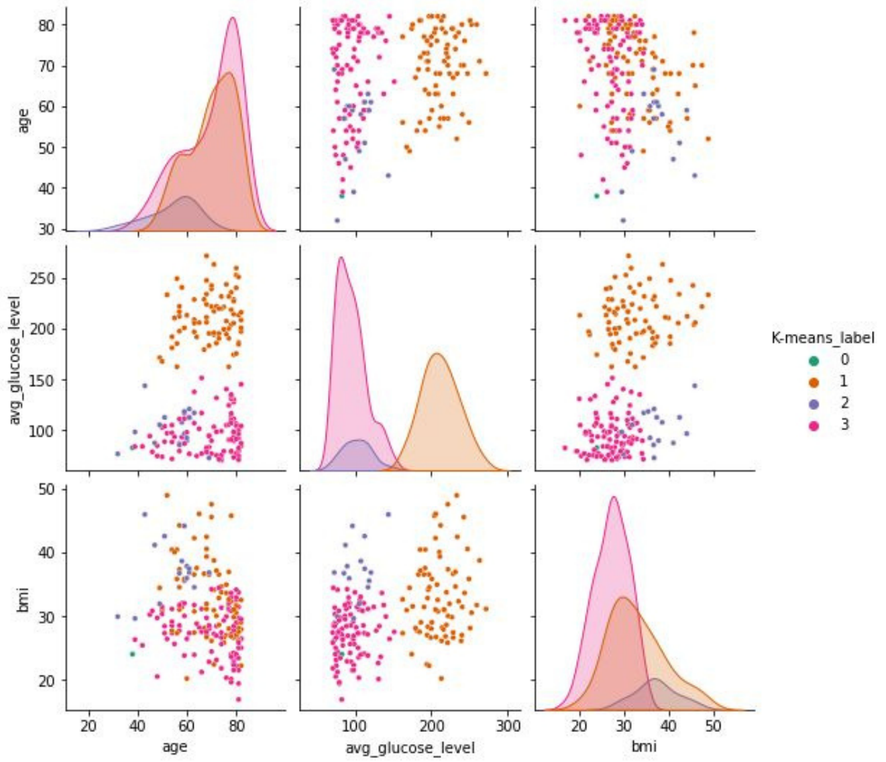
OPTIMUM NUMBER OF CLUSTER BY GAP STATISTIC METHOD: 4

```python
#Import
from sklearn.cluster import KMeans
#Define
kmeans = KMeans(n_clusters=4, random_state=1)
#Train
kmeans.fit(numeric_features_s)
```

K-MEANS



PATIENT



PATIENT HAD A STROKE

```python
numeric_features['K-means_label'].value_counts()#จำนวนสมาชิกในแต่ละกลุ่ม
```

```
3    1327
0    1244
2     922
1     569
Name: K-means_label, dtype: int64
```

```python
stroke3['K-means_label'].value_counts()
```

```
3    96
1    75
2    16
0     1
Name: K-means_label, dtype: int64
```

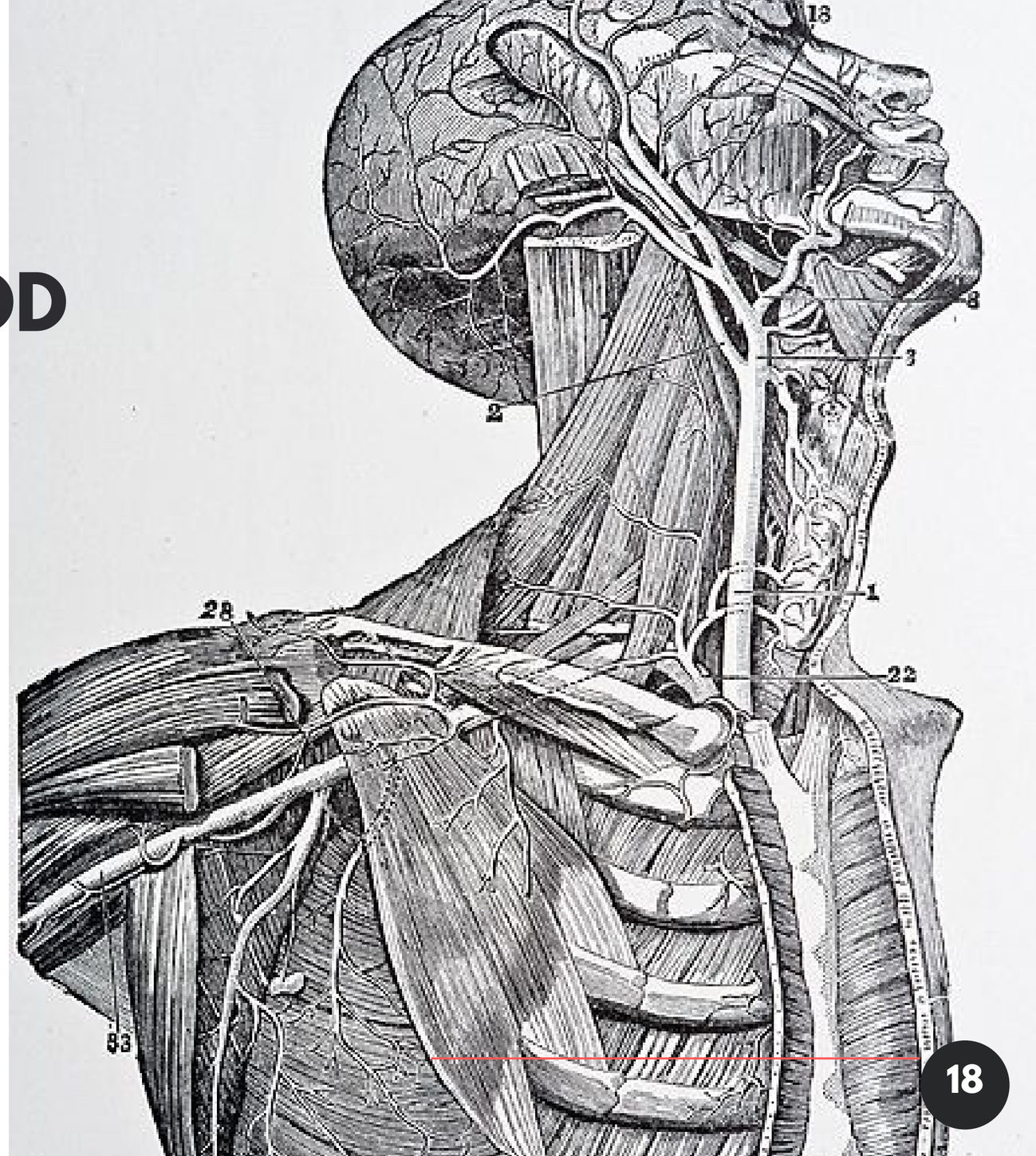
CLASSIFICATION METHOD

Decision Tree Classifier

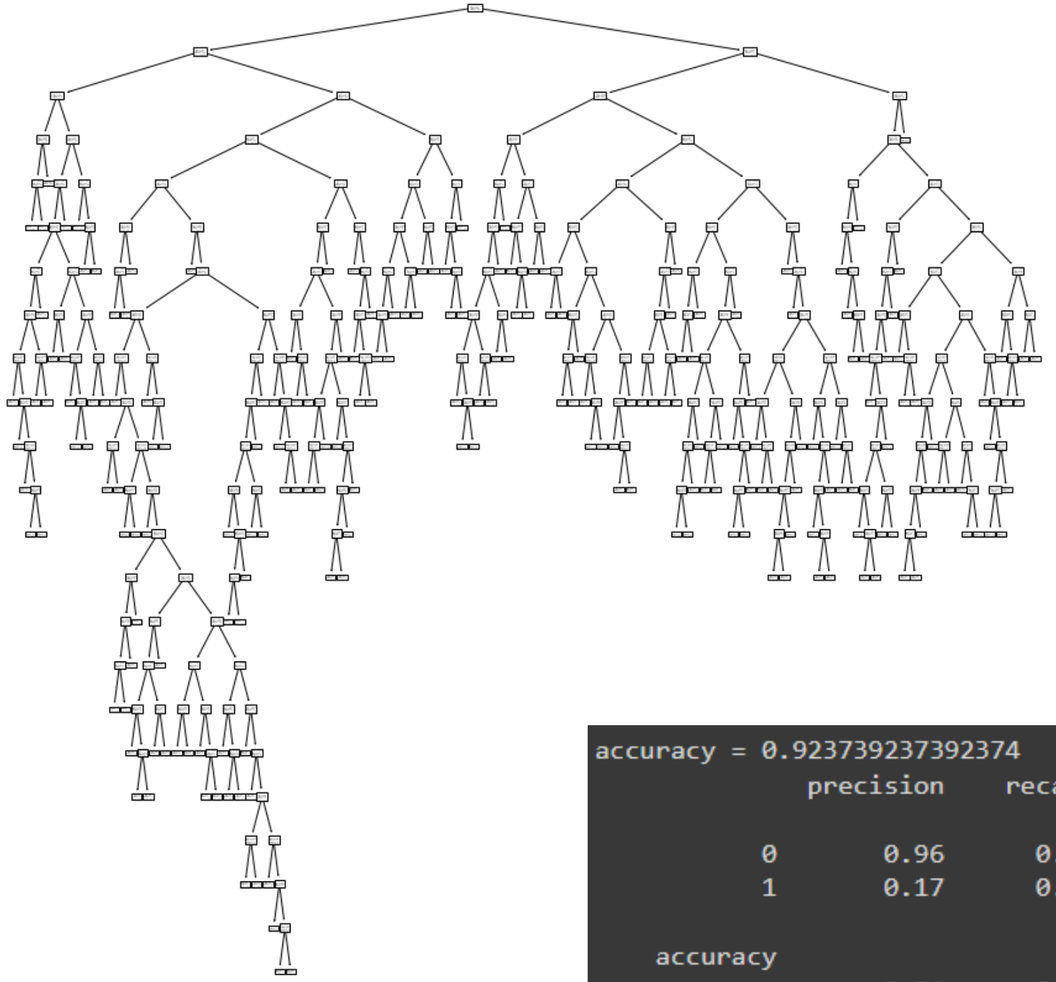
Naïve Bayes classifier

K-Nearest Neighbors

80%	20%
TRAINING	*TEST*



Decision Tree Classifier



accuracy = 0.923739237392374

	precision	recall	f1-score	support
0	0.96	0.96	0.96	774
1	0.17	0.15	0.16	39
accuracy			0.92	813
macro avg	0.56	0.56	0.56	813
weighted avg	0.92	0.92	0.92	813

Decision Tree Classifier

10-FOLD CROSS VALIDATION AND FIND THE BEST PARAMETERS

```
# Define the parameter grid to search over
parameters ={
        'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
        'criterion' : ['gini', 'entropy'],
        'min_samples_leaf': [2,3,4,5,7,10],
        'min_samples_split': [2,3,4,5,7,10]}

# Define
clf1 = DecisionTreeClassifier(random_state=1)

grid_dt =  GridSearchCV(estimator=clf1,
                        param_grid=parameters,
                        cv=10, n_jobs=-1, verbose=3,scoring = "accuracy")
# Fit the grid search object to the data // หาพารามิเตอร์ที่ดีที่สุด แล้วใช้  cross-va
```

```
grid_dt.fit(X_train, y_train)
print("Best hyperparameters: ", grid_dt.best_params_)
```

```
Fitting 10 folds for each of 648 candidates, totalling 6480 fits
Best hyperparameters:  {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 2, 'min_samples_split': 2}
```

Decision Tree Classifier

10-FOLD CROSS VALIDATION AND FIND THE BEST PARAMETERS

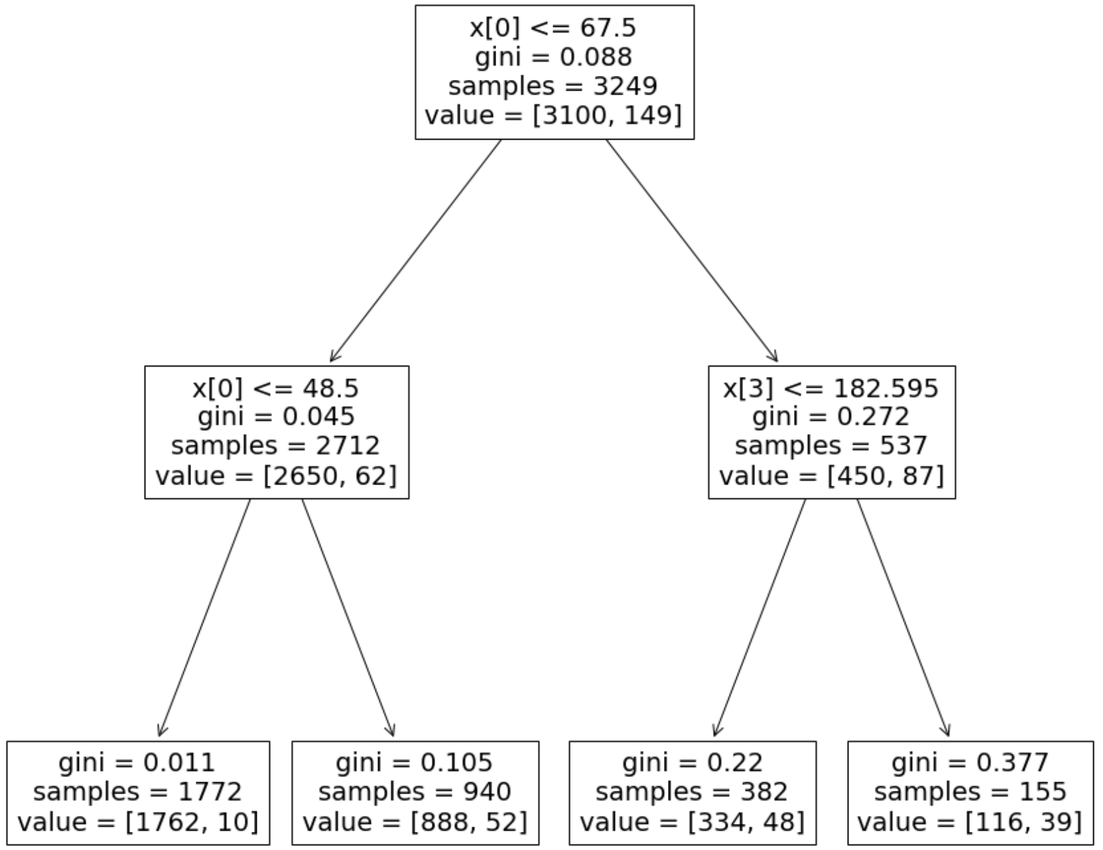
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import accuracy_score,f1_score
```

```
#Define
dt_clf = DecisionTreeClassifier(criterion = 'gini',
                                max_depth=2,
                                min_samples_leaf=5,
                                min_samples_split=2,
                                random_state=1)
#Train
dt_clf.fit(X_train, y_train)

#Predict
y_pred = dt_clf.predict(X_test)

#Evaluation
from sklearn.metrics import accuracy_score, classification_report

print(f'accuracy = {accuracy_score(y_test, y_pred)}')
print(classification_report(y_test, y_pred) )
```

```
accuracy = 0.9520295202952029
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       774
           1       0.00      0.00      0.00        39

    accuracy                           0.95       813
   macro avg       0.48      0.50      0.49       813
weighted avg       0.91      0.95      0.93       813
```

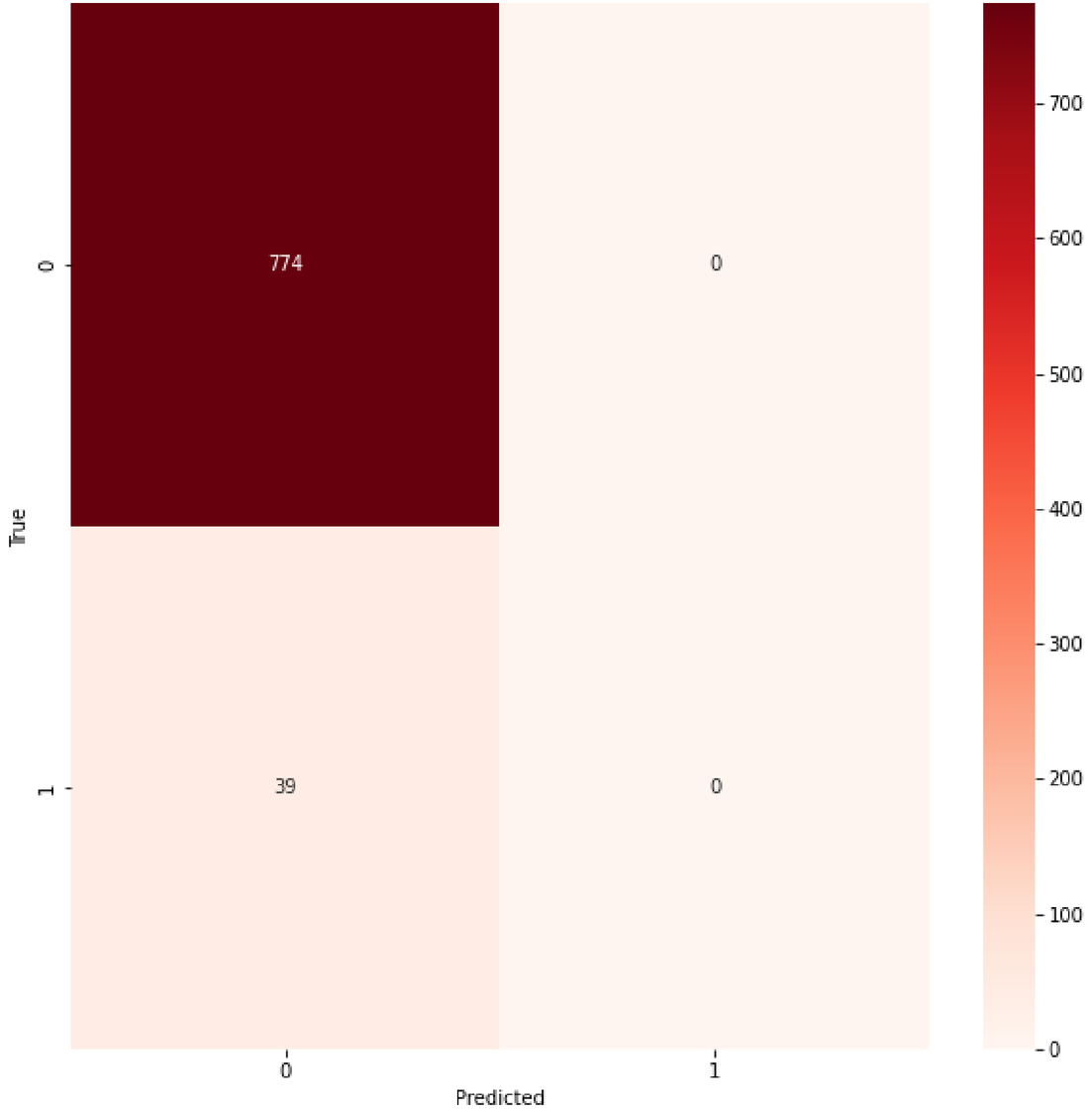
Decision Tree Classifier

10-FOLD CROSS VALIDATION
AND FIND THE BEST PARAMETERS



Decision Tree Classifier

CONFUSION_MATRIX



Decision Tree Classifier

10-FOLD CROSS VALIDATION AND FIND THE BEST PARAMETERS (RECALL)

```
parameters_dt ={
        'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
        'criterion' : ['gini', 'entropy'],
        'min_samples_leaf': [2,3,4,5,7,10],
        'min_samples_split': [2,3,4,5,7,10]}

#Define
clf2 = DecisionTreeClassifier(random_state=1)

grid_dt2 =  GridSearchCV(estimator=clf2,
                         param_grid=parameters_dt,
                         cv=10, n_jobs=-1, verbose=3, scoring = "recall")
# Fit the grid search object to the data
```

```
# Fit the grid search object to the data
%%time
grid_dt2.fit(X_train, y_train)
print("Best hyperparameters: ", grid_dt2.best_params_)
```

```
Fitting 10 folds for each of 648 candidates, totalling 6480 fits
Best hyperparameters:  {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 3, 'min_samples_split': 2}
CPU times: user 4.83 s, sys: 299 ms, total: 5.13 s
Wall time: 1min 18s
```

Decision Tree Classifier

TRAIN DECISIONTREE MODEL WITH RECALL

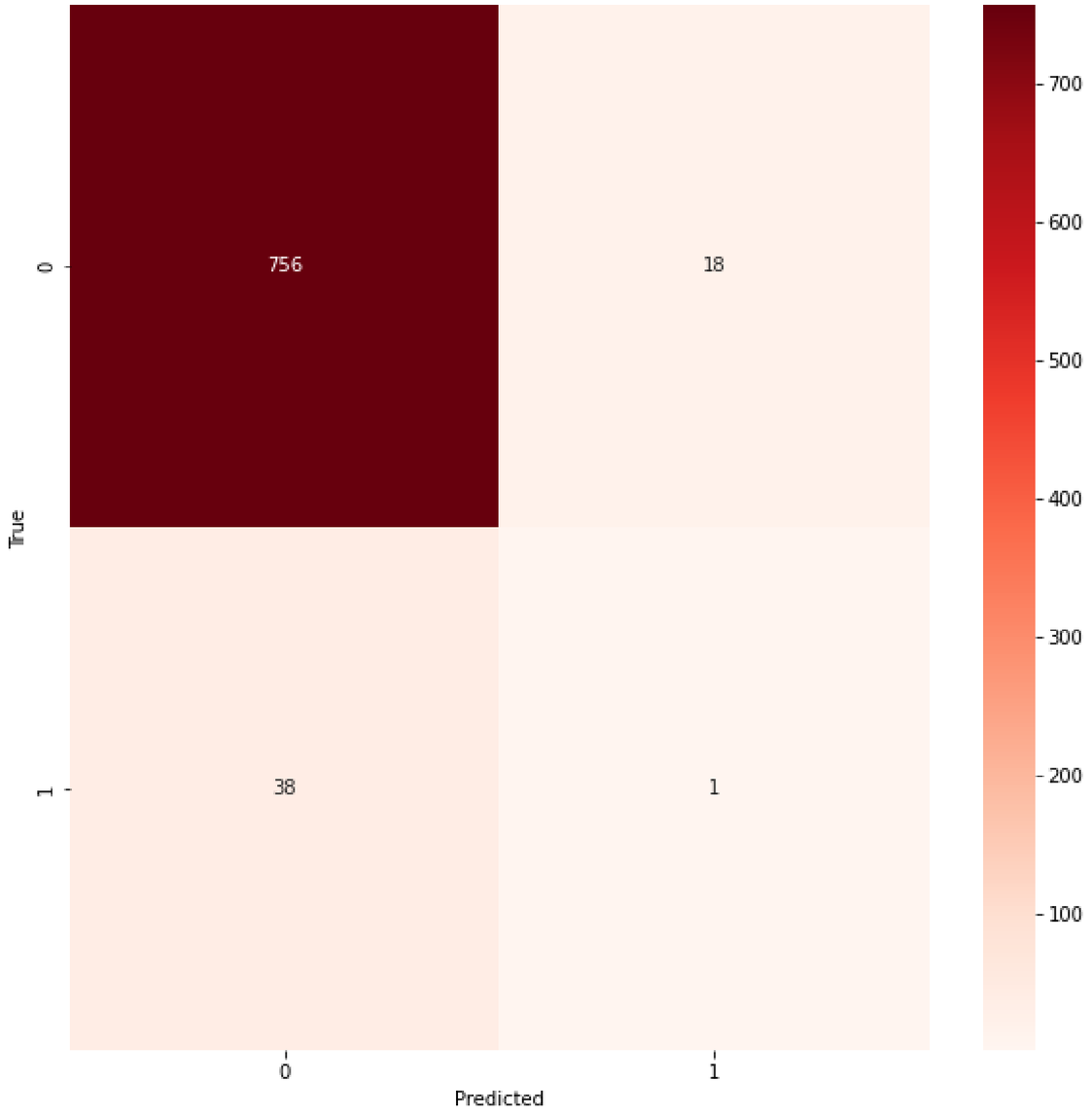
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import accuracy_score,recall_score
```

```
# Create a decision tree classifier
dt_clf2 = DecisionTreeClassifier(criterion = 'entropy',
                                 max_depth=10,
                                 min_samples_leaf=3,
                                 min_samples_split=2,
                                 random_state=1)
#Train
dt_clf2.fit(X_train, y_train)

# Make predictions on the test set
y_pred = dt_clf2.predict(X_test)

# Evaluate the recall of the classifier
print(f'recall = {recall_score(y_test, y_pred)}')
print(classification_report(y_test,y_pred))
```

```
recall = 0.02564102564102564
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       774
           1       0.05      0.03      0.03        39

    accuracy                           0.93       813
   macro avg       0.50      0.50      0.50       813
weighted avg       0.91      0.93      0.92       813
```

Decision Tree Classifier

CONFUSION_MATRIX



Naïve Bayes classifier

10-FOLD CROSS VALIDATION AND FIND THE BEST PARAMETERS (ACCURACY)

```
parameters_nb = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}

# Create a Naive Bayes classifier object
nb = GaussianNB()


# Create a GridSearchCV object with 10-fold cross-validation
grid_nb = GridSearchCV(estimator=nb,
                       param_grid=parameters_nb,
                       cv=10, n_jobs=-1, verbose=3,scoring = "accuracy")
```

```
1 # Fit the GridSearchCV object to the data
2 grid_nb.fit(X_train, y_train)
3 print("Best hyperparameters: ", grid_nb.best_params_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
Best hyperparameters:  {'var_smoothing': 1e-05}
```

www.kaggle.com

Naïve Bayes classifier

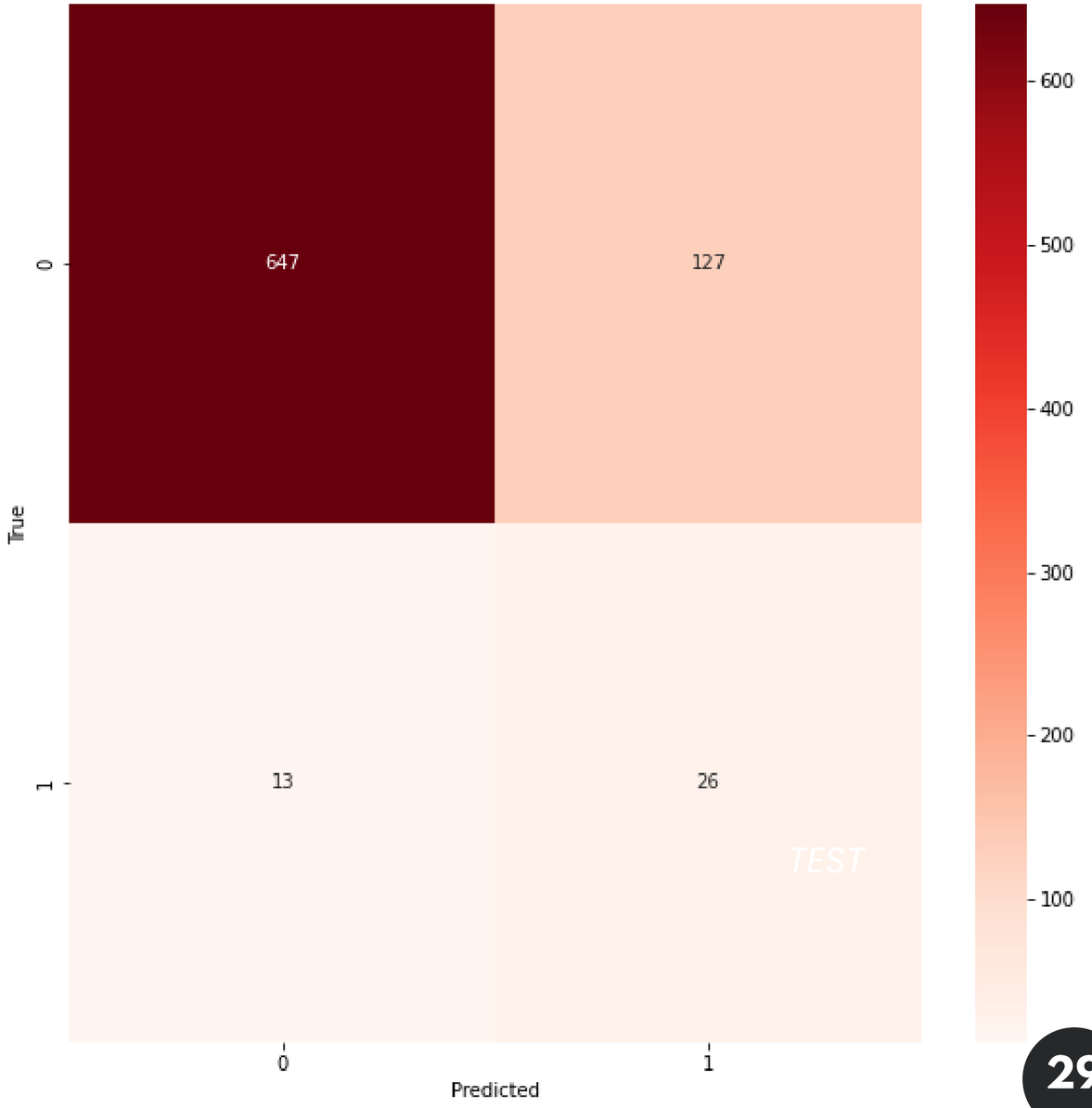
Train NaiveBayes model

```python
# Create a Gaussian Naive Bayes classifier
model_nb = GaussianNB(var_smoothing = 1e-05)

# Train the model on the training set
model_nb.fit(X_train, y_train)

y_pred = model_nb.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.84      0.90       774
           1       0.17      0.67      0.27        39

    accuracy                           0.83       813
   macro avg       0.58      0.75      0.59       813
weighted avg       0.94      0.83      0.87       813
```

Naïve Bayes classifier

confusion_matrix



www.kaggle.com

Naïve Bayes classifier

10-FOLD CROSS VALIDATION AND
FIND THE BEST PARAMETERS
(RECALL)

```python
parameters_nb = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}

# Create a Naive Bayes classifier object
nb2 = GaussianNB()


# Create a GridSearchCV object with 10-fold cross-validation
grid_nb2 = GridSearchCV(estimator=nb2,
                        param_grid=parameters_nb,
                        cv=10, n_jobs=-1, verbose=3,scoring = "recall")
```

```python
# Fit the GridSearchCV object to the data
%%time
grid_nb2.fit(X_train, y_train)
print("Best hyperparameters: ", grid_nb2.best_params_)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
Best hyperparameters:  {'var_smoothing': 1e-09}
```

www.kaggle.com

Stroke Prediction Dataset

Naïve Bayes classifier

Train NaiveBayes model

```python
# Create a Gaussian Naive Bayes classifier
model_nb2 = GaussianNB(var_smoothing = 1e-09)

# Train the model on the training set
model_nb2.fit(X_train, y_train)

#predict
y_pred = model_nb2.predict(X_test)

#Evaluation
from sklearn.metrics import recall_score, classification_report
print(f'recall = {recall_score(y_test, y_pred)}')
print(classification_report(y_test, y_pred))
```

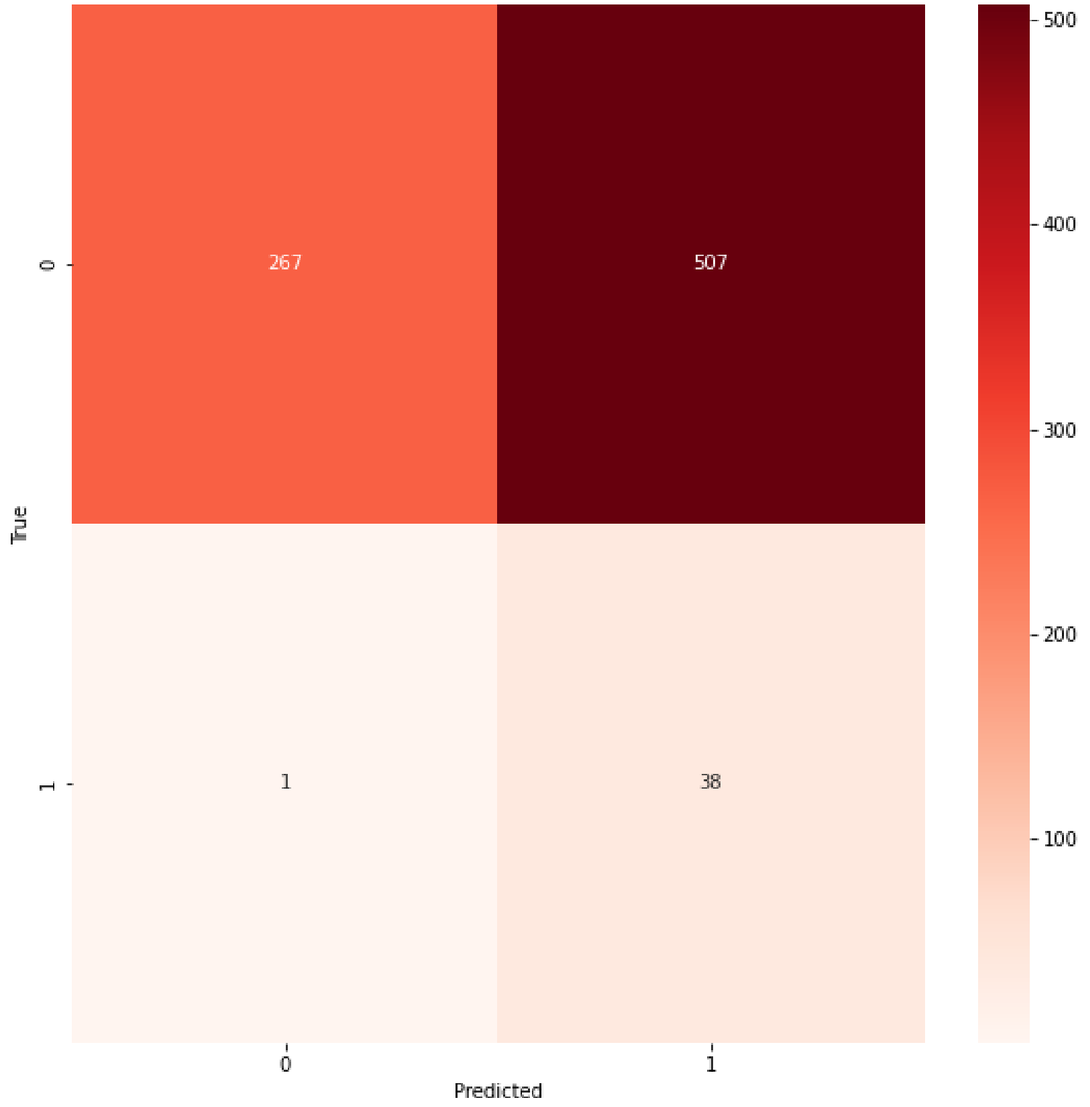
```
recall = 0.9743589743589743
              precision    recall  f1-score   support

           0       1.00      0.34      0.51       774
           1       0.07      0.97      0.13        39

    accuracy                           0.38       813
   macro avg       0.53      0.66      0.32       813
weighted avg       0.95      0.38      0.49       813
```

www.kaggle.com

Stroke Prediction Dataset

Naïve Bayes classifier

confusion_matrix



www.kaggle.com

K-Nearest Neighbors

10-FOLD CROSS VALIDATION AND FIND THE BEST PARAMETERS (ACC)

```python
param_grid = {'n_neighbors': [3, 5, 7, 9, 11],
              'weights': ['uniform', 'distance'],
              'algorithm': ['ball_tree', 'kd_tree', 'brute']}

# Create the KNN classifier
knn = KNeighborsClassifier()

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=10, n_jobs=-1,scoring = "accuracy")

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)
```

```python
# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'algorithm': 'ball_tree', 'n_neighbors': 11, 'weights': 'uniform'}
```

K-Nearest Neighbors

TRAIN KNN MODEL (ACC)

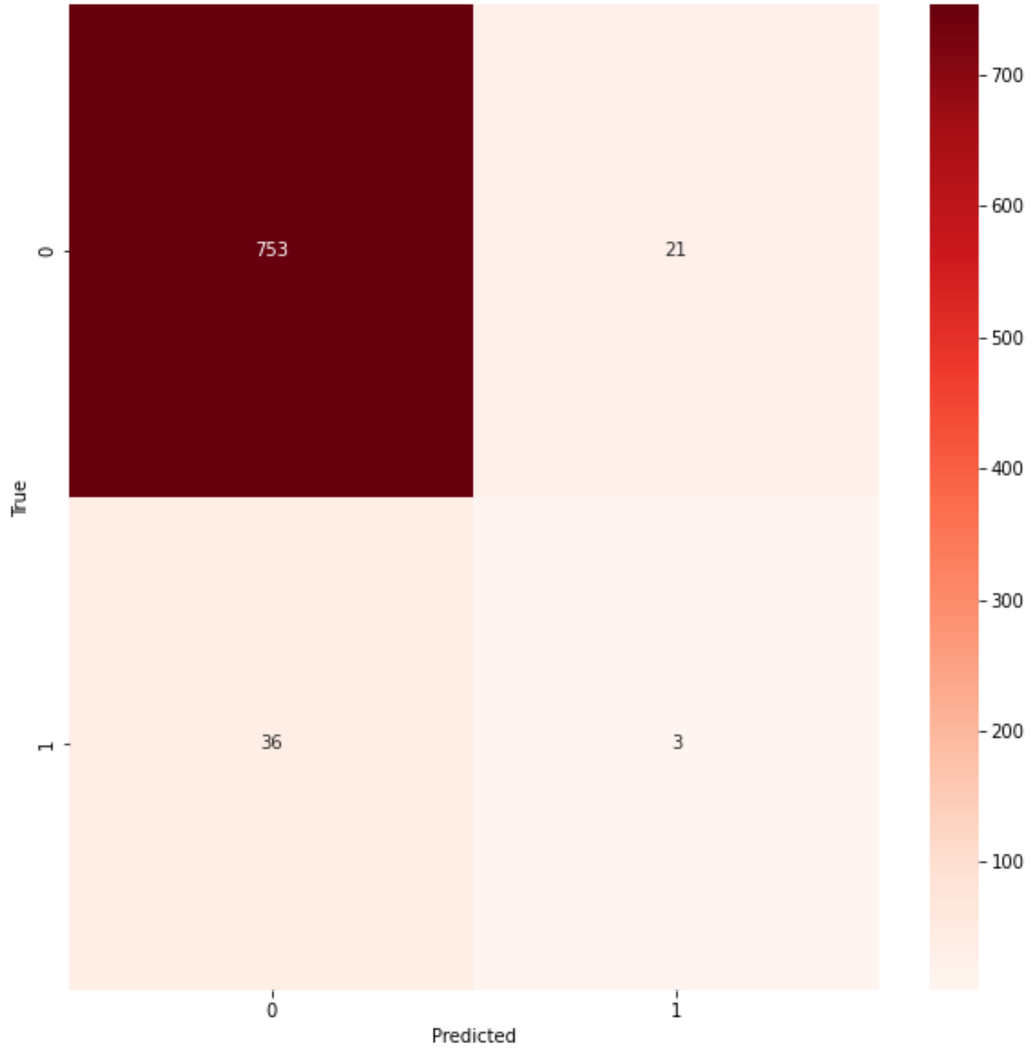
```
from sklearn.neighbors import KNeighborsClassifier
```

```
# Instantiate KNN model
knn = KNeighborsClassifier(n_neighbors=3, weights='distance', algorithm='ball_tree')

# Fit the model to the training data
knn.fit(X_train, y_train)

# Predict the test data
y_pred = knn.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.97	0.96	774
1	0.12	0.08	0.10	39
accuracy			0.93	813
macro avg	0.54	0.52	0.53	813
weighted avg	0.91	0.93	0.92	813

K-Nearest Neighbors

CONFUSION_MATRIX



K-Nearest Neighbors

10-FOLD CROSS VALIDATION AND FIND THE BEST PARAMETERS (RECALL)

```
parameters_knn = {'n_neighbors': [3, 5, 7, 9, 11],
                  'weights': ['uniform', 'distance'],
                  'algorithm': ['ball_tree', 'kd_tree', 'brute']}

# Create the KNN classifier
knn2 = KNeighborsClassifier()

# Create the GridSearchCV object
grid_knn2 = GridSearchCV(estimator=knn2, param_grid=parameters_knn, verbose=3, cv=10,scoring='recall',n_jobs=-1)
```

```
%%time
# Fit the GridSearchCV object to the training data
grid_knn2.fit(X_train, y_train)
# Print the best hyperparameters
print("Best hyperparameters: ", grid_knn2.best_params_)
```

```
Fitting 10 folds for each of 30 candidates, totalling 300 fits
Best hyperparameters:  {'algorithm': 'ball_tree', 'n_neighbors': 3, 'weights': 'uniform'}
CPU times: user 267 ms, sys: 22.5 ms, total: 290 ms
Wall time: 7.74 s
```

K-Nearest Neighbors

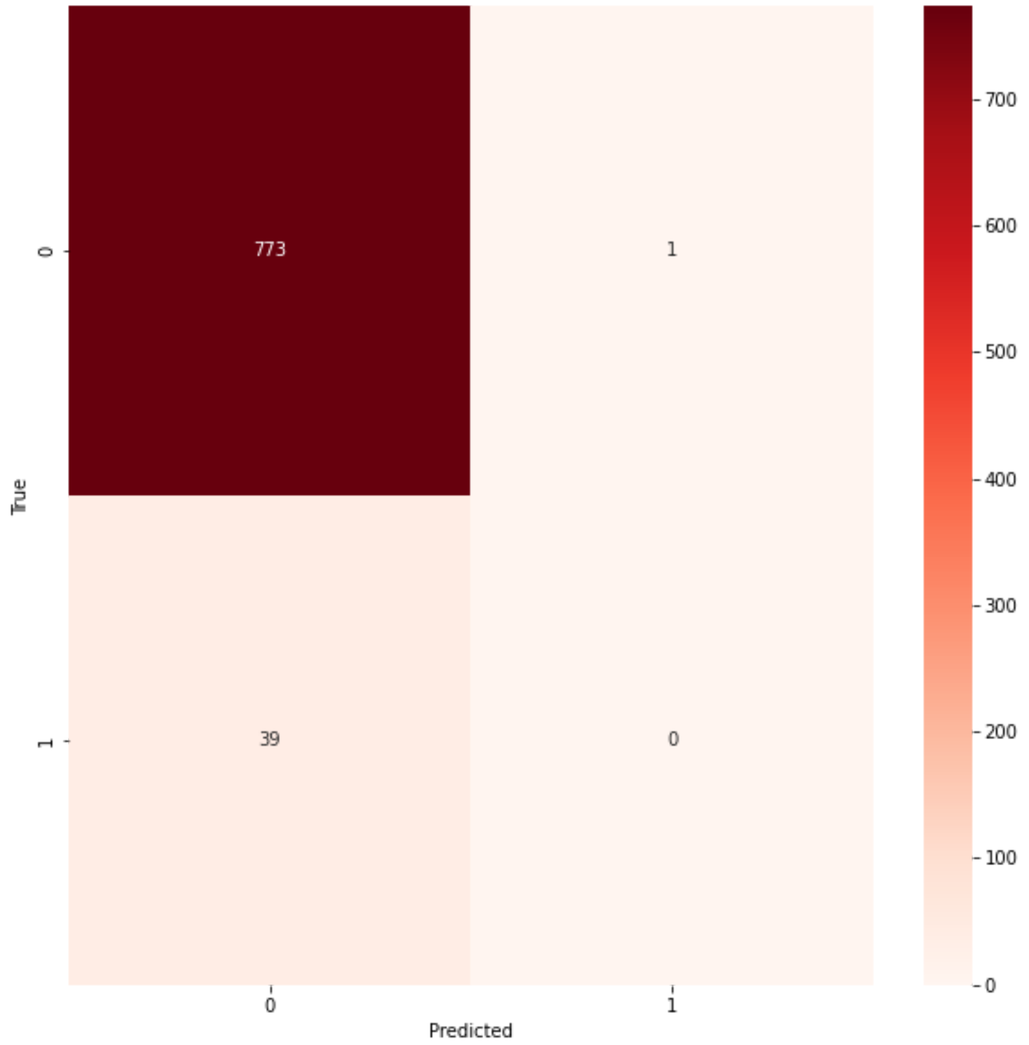
TRAIN KNN MODEL (RECALL)

```python
# Instantiate KNN model
knn2 = KNeighborsClassifier(n_neighbors=3, weights='uniform', algorithm='ball_tree')

# Fit the model to the training data
knn2.fit(X_train, y_train)

# Predict the test data
y_pred = knn2.predict(X_test)

#Evaluation
from sklearn.metrics import recall_score, classification_report
print(f'recall = {recall_score(y_test, y_pred)}')
print(classification_report(y_test, y_pred))
```

```
recall = 0.07692307692307693
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       774
           1       0.13      0.08      0.10        39

    accuracy                           0.93       813
   macro avg       0.54      0.53      0.53       813
weighted avg       0.91      0.93      0.92       813
```

K-Nearest Neighbors

CONFUSION_MATRIX



Decision Tree
VS
Naïve Bayes
VS
K-Nearest Neighbors

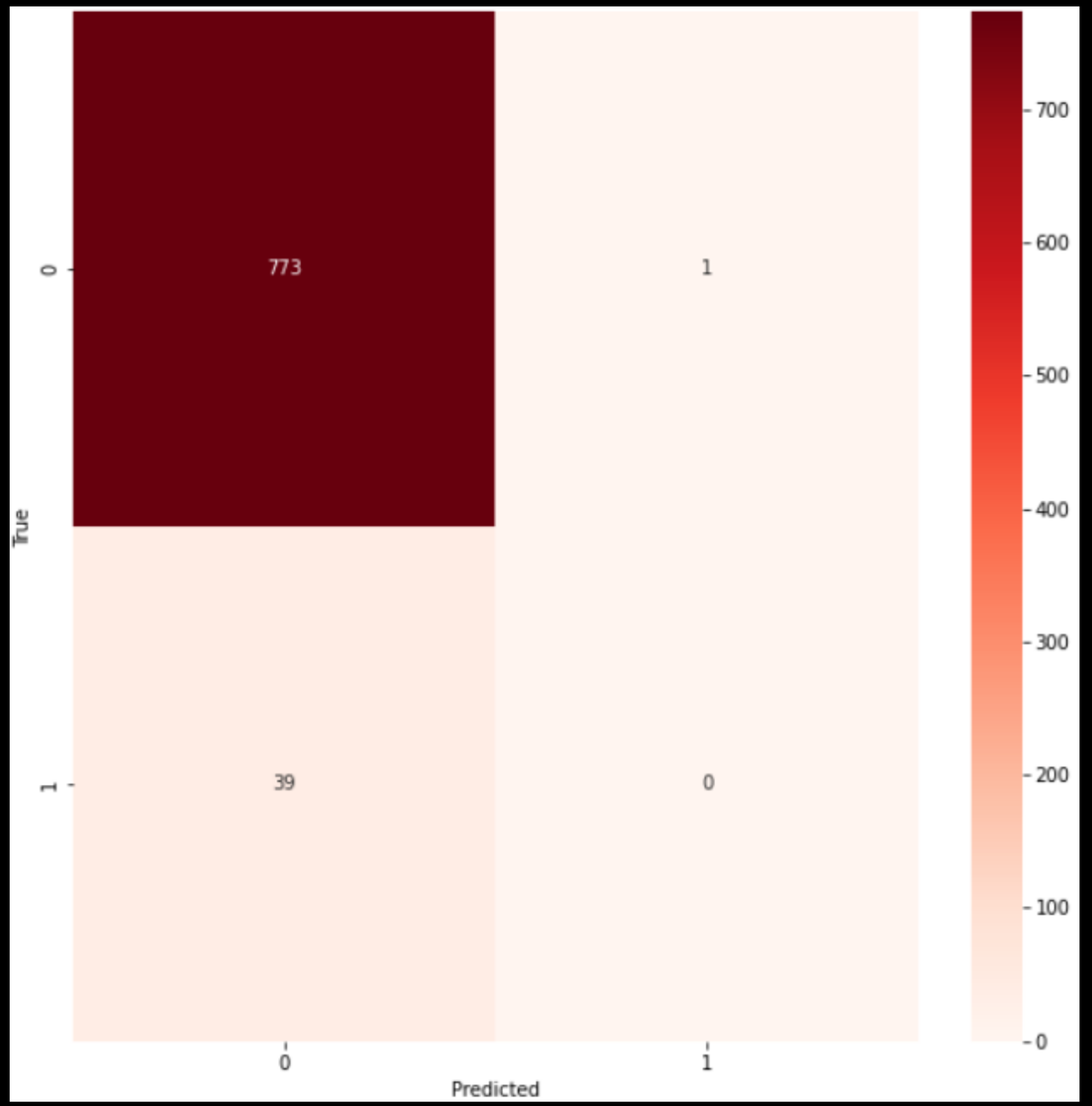
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

```
# 1
knn = KNeighborsClassifier(n_neighbors=3, weights='distance', algorithm='ball_tree')
scores = cross_val_score(knn, X_train, y_train, cv=10)
print('Mean cross-validation score: K3weighted', scores.mean())
#2
knn2 = KNeighborsClassifier(n_neighbors=11, weights='uniform', algorithm='ball_tree')
scores = cross_val_score(knn2, X_train, y_train, cv=10)
print('Mean cross-validation score: K11no_weighted', scores.mean())
#3
dt_clf = DecisionTreeClassifier(criterion = 'gini',max_depth=2,min_samples_leaf=5,min_samples_split=2,random_state=1)
scores = cross_val_score(dt_clf, X_train, y_train, cv=10)
print('Mean cross-validation score: DecisionTree_max_depth=2', scores.mean())
#4
dt_clf_F1 = DecisionTreeClassifier(criterion = 'gini',max_depth=10,min_samples_leaf=3,min_samples_split=7,max_features=0.3,random_state=1)
scores = cross_val_score(dt_clf, X_train, y_train, cv=10)
print('Mean cross-validation score: DecisionTree_max_depth=10', scores.mean())
#5
model_nb = GaussianNB(var_smoothing = 1e-05)
scores = cross_val_score(model_nb, X_train, y_train, cv=10)
print('Mean cross-validation score: var_smoothing1e-05', scores.mean())
```

```
Mean cross-validation score: K3weighted 0.9430550807217475
Mean cross-validation score: K11no_weighted 0.9532174738841406
Mean cross-validation score: DecisionTree_max_depth=2 0.9526020892687559
Mean cross-validation score: DecisionTree_max_depth=10 0.9526020892687559
Mean cross-validation score: var_smoothing1e-05 0.8125546058879394
```

```python
knn2 = KNeighborsClassifier(n_neighbors=11, weights='uniform', algorithm='ball_tree')
knn2.fit(X_train,y_train)
y_pred = knn2.predict(X_test)

from sklearn.metrics import accuracy_score
# Assume y_true and y_pred are the true and predicted labels, respectively
accuracy = accuracy_score(y_test, y_pred)
# Print the accuracy score
print("Accuracy score: ", accuracy)
```

```
Accuracy score:  0.95079950799508
```



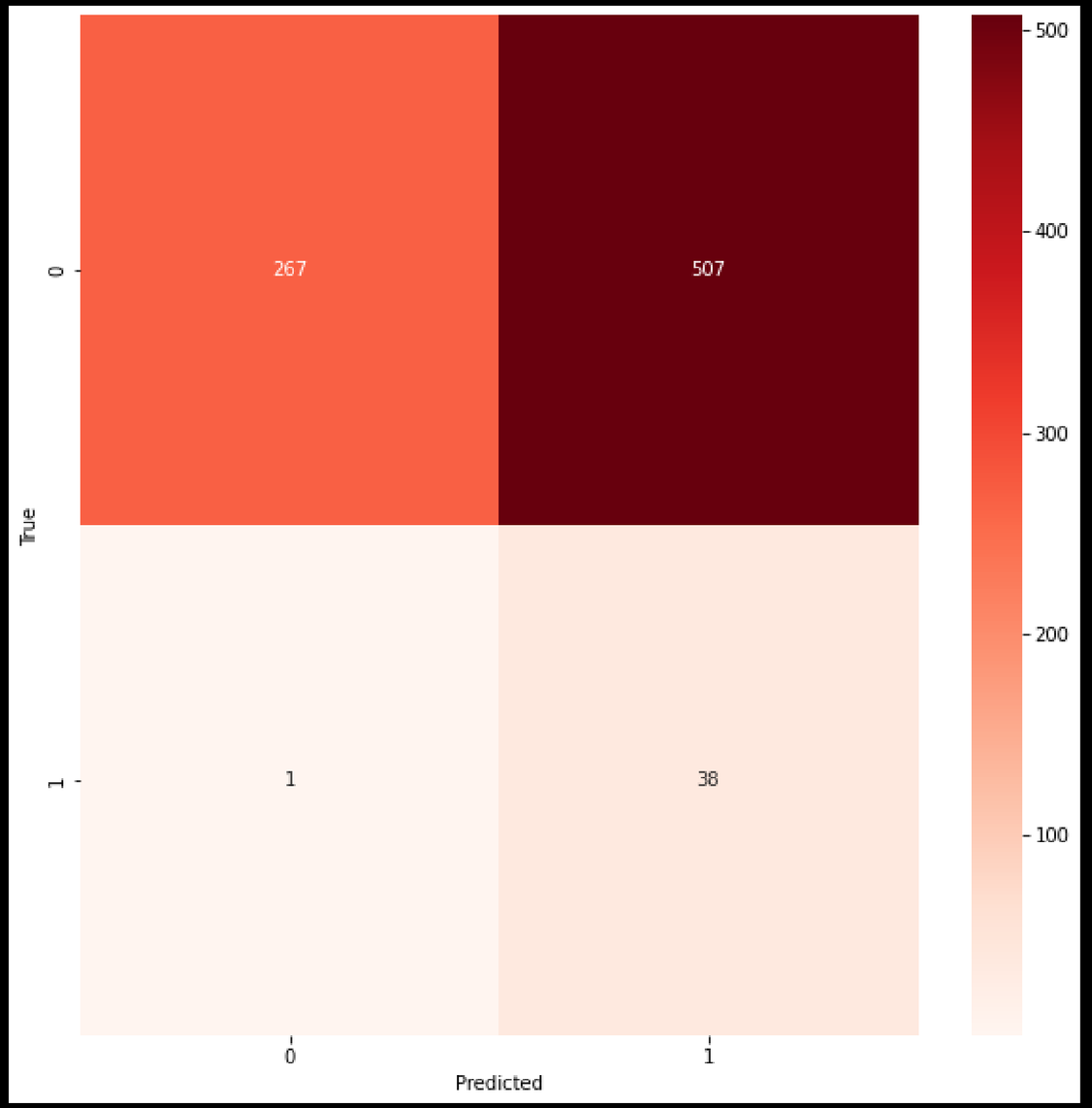
Stroke Prediction Dataset

```python
# 1
knn1 = KNeighborsClassifier(n_neighbors=11, weights='uniform', algorithm='ball_tree')
#2
knn2 = KNeighborsClassifier(n_neighbors=3, weights='distance', algorithm='ball_tree')
#3
dt_clf1 = DecisionTreeClassifier(criterion = 'gini',max_depth=2,min_samples_leaf=5,min_samples_split=2,random_state=1)
#4
dt_clf2 = DecisionTreeClassifier(criterion = 'gini',max_depth=10,min_samples_leaf=3,min_samples_split=2,random_state=1)
#5
model_nb1 = GaussianNB(var_smoothing = 1e-05)
#6
model_nb2 = GaussianNB(var_smoothing = 1e-09)

#ใช้ recall  เนื่องจากในทางการแพทย์เราต้องการทราบแค่ว่าโมเดลสามารถระบุได้ว่าเป็นแล้วมันเป็นเท่าไหร่
# Compute the recall_scores using cross-validation
recall_knn1 = cross_val_score(knn1, X_train, y_train, cv=10, scoring='recall')
recall_knn2 = cross_val_score(knn2, X_train, y_train, cv=10, scoring='recall')
recall_dt1 = cross_val_score(dt_clf1, X_train, y_train, cv=10, scoring='recall')
recall_dt2 = cross_val_score(dt_clf2, X_train, y_train, cv=10, scoring='recall')
recall_nb1 = cross_val_score(model_nb1, X_train, y_train, cv=10, scoring='recall')
recall_nb2 = cross_val_score(model_nb2, X_train, y_train, cv=10, scoring='recall')


# Print the mean recall scores for each model
print("Decision Tree1 recall score:", recall_dt1.mean())
print("Decision Tree2 recall score:", recall_dt2.mean())
print("Naive Bayes1 recall score:", recall_nb1.mean())
print("Naive Bayes2 recall score:", recall_nb2.mean())
print("K-Nearest Neighbors1 recall score:", recall_knn1.mean())
print("K-Nearest Neighbors2 recall score:", recall_knn2.mean())
```

```
Decision Tree1 recall score: 0.0133333333333333332
Decision Tree2 recall score: 0.10095238095238095
Naive Bayes1 recall score: 0.5571428571428572
Naive Bayes2 recall score: 0.9528571428571428
K-Nearest Neighbors1 recall score: 0.0
K-Nearest Neighbors2 recall score: 0.05333333333333333
```

```python
model_nb2 = GaussianNB(var_smoothing = 1e-09)
model_nb2.fit(X_train,y_train)
y_pred = model_nb2.predict(X_test)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
# Print f1 score
print("Recall score: ", recall)
# Print the accuracy score
print("Accuracy score: ", accuracy)
```

```
Recall score:  0.9743589743589743
Accuracy score:  0.37515375153751537
```



Association Rule

PREPARE DATA

	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
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
5	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
5104	Female	13.0	0	0	No	children	Rural	103.08	18.6	Unknown	0
5106	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

4062 rows × 11 columns

STROKE_DATA_CLEANED

PREPARE DATA

(จัดกลุ่ม)

AGE

```python
# Define age groups
bins = [0, 20 , 59, 100]
labels = ['<19','20-59','60+']

# Group ages into age groups
df1['age_group'] = pd.cut(df1['age'], bins=bins, labels=labels)

df1
```

BMI

```python
# Define BMI categories or groups
bins = [0, 18.5, 22.9, 24.9, 29.9, 50]
labels = ['Underweight', 'Normal', 'Overweight', 'Obese Class I', 'Obese Class II']

# Group BMI values into categories or groups
df1['bmi_category'] = pd.cut(df1['bmi'], bins=bins, labels=labels)

df1
```

AVG_GLUCOSE_LEVEL

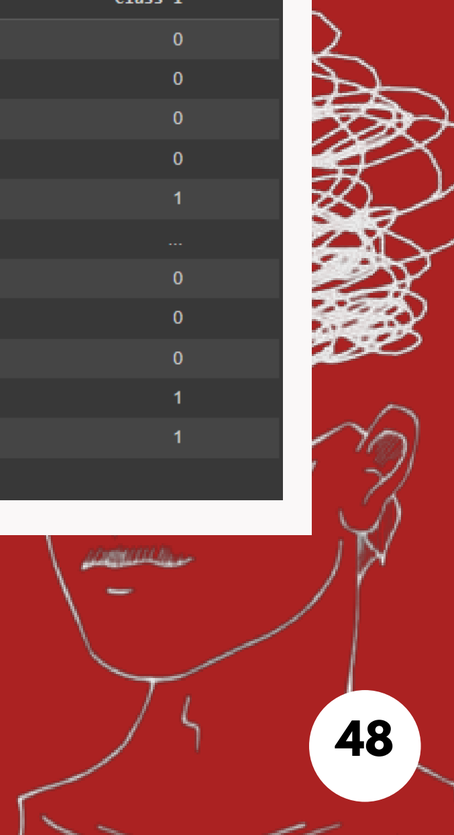
```python
# Define glucose level categories or bins
bins = [0, 70, 100, 125, 300]
labels = ['Low', 'Normal', 'High', 'Very high']

# Group average glucose level into categories or bins
df1['glucose_category'] = pd.cut(df1['avg_glucose_level'], bins=bins, labels=labels)

df1
```

ONE-HOT-ENCODER

```
# convert categorical variables to numerical using one-hot encoding
cat_vars = ['gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'smoking_status','age_group', 'bmi_category', 'glucose_category','stroke']
data_encoded = pd.get_dummies(df1, columns=cat_vars)
```

	age	avg_glucose_level	bmi	gender_Female	gender_Male	hypertension_0	hypertension_1	heart_disease_0	heart_disease_1	ever_married_No	...	bmi_category_Normal	bmi_category_Overweight	bmi_category_Obese Class I	bmi_cate
0	67.0	228.69	36.6	0	1	1	0	0	1	0	...	0	0	0	
2	80.0	105.92	32.5	0	1	1	0	0	1	0	...	0	0	0	
3	49.0	171.23	34.4	1	0	1	0	1	0	0	...	0	0	0	
4	79.0	174.12	24.0	1	0	0	1	1	0	0	...	0	1	0	
5	81.0	186.21	29.0	0	1	1	0	1	0	0	...	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5104	13.0	103.08	18.6	1	0	1	0	1	0	1	...	1	0	0	
5106	81.0	125.20	40.0	1	0	1	0	1	0	0	...	0	0	0	
5107	35.0	82.99	30.6	1	0	1	0	1	0	0	...	0	0	0	
5108	51.0	166.29	25.6	0	1	1	0	1	0	0	...	0	0	1	
5109	44.0	85.28	26.2	1	0	1	0	1	0	0	...	0	0	1	

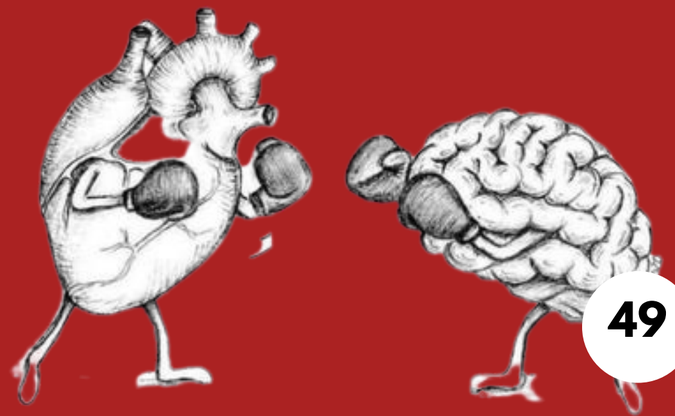
4062 rows × 36 columns



DROP COLUMNS

	gender_Female	gender_Male	hypertension_0	hypertension_1	heart_disease_0	heart_disease_1	ever_married_No	ever_married_Yes	work_type_Govt_job	work_type_Never_worked	...	bmi_category_Underweight	bmi_category_Nor
0	0	1	1	0	0	1	0	1	0	0	...	0	
2	0	1	1	0	0	1	0	1	0	0	...	0	
3	1	0	1	0	1	0	0	1	0	0	...	0	
4	1	0	0	1	1	0	0	1	0	0	...	0	
5	0	1	1	0	1	0	0	1	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
5104	1	0	1	0	1	0	1	0	0	0	...	0	
5106	1	0	1	0	1	0	0	1	0	0	...	0	
5107	1	0	1	0	1	0	0	1	0	0	...	0	
5108	0	1	1	0	1	0	0	1	0	0	...	0	
5109	1	0	1	0	1	0	0	1	1	0	...	0	

4062 rows × 32 columns



Association Rule

```
1 # Build association rules
2 frequent_itemsets = apriori(tdata, min_support=0.01, use_colnames=True)
3 rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

```
1 rules_stroke1.sort_values('lift', ascending=False)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
188354	(age_group_60+, glucose_category_Very high, wo...	(ever_married_Yes, stroke_1)	0.051699	0.041359	0.010094	0.195238	4.720578	0.007955	1.191211
48628	(age_group_60+, glucose_category_Very high, wo...	(stroke_1)	0.051699	0.046283	0.011078	0.214286	4.629939	0.008686	1.213822
188347	(glucose_category_Very high, age_group_60+, ev...	(stroke_1)	0.047760	0.046283	0.010094	0.211340	4.566297	0.007883	1.209289
188362	(age_group_60+, glucose_category_Very high)	(ever_married_Yes, work_type_Private, stroke_1)	0.096750	0.025357	0.010094	0.104326	4.114281	0.007640	1.088167
48632	(age_group_60+, glucose_category_Very high)	(work_type_Private, stroke_1)	0.096750	0.028065	0.011078	0.114504	4.079952	0.008363	1.097616
...	...	...	...	...	...	...	...	...	...
37428	(bmi_category_Obese Class II, heart_disease_0)	(work_type_Private, stroke_1)	0.367307	0.028065	0.010340	0.028150	1.003034	0.000031	1.000088
28257	(hypertension_0, bmi_category_Obese Class II)	(ever_married_Yes, stroke_1)	0.338503	0.041359	0.014032	0.041455	1.002312	0.000032	1.000100
43876	(work_type_Private)	(age_group_60+, ever_married_Yes, stroke_1)	0.576809	0.030281	0.017479	0.030303	1.000739	0.000013	1.000023
27685	(Residence_type_Urban)	(hypertension_0, ever_married_Yes, stroke_1)	0.507878	0.030527	0.015510	0.030538	1.000367	0.000006	1.000012
25771	(heart_disease_0)	(hypertension_0, glucose_category_Normal, stro...	0.946332	0.013786	0.013048	0.013788	1.000102	0.000001	1.000001

845 rows × 9 columns

Association Rule

```
1 rules_stroke1.sort_values('lift', ascending=False)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
188354	(age_group_60+, glucose_category_Very high, wo...	(ever_married_Yes, stroke_1)	0.051699	0.041359	0.010094	0.195238	4.720578	0.007955	1.191211
48628	(age_group_60+, glucose_category_Very high, wo...	(stroke_1)	0.051699	0.046283	0.011078	0.214286	4.629939	0.008686	1.213822
188347	(glucose_category_Very high, age_group_60+, ev...	(stroke_1)	0.047760	0.046283	0.010094	0.211340	4.566297	0.007883	1.209289
188362	(age_group_60+, glucose_category_Very high)	(ever_married_Yes, work_type_Private, stroke_1)	0.096750	0.025357	0.010094	0.104326	4.114281	0.007640	1.088167
48632	(age_group_60+, glucose_category_Very high)	(work_type_Private, stroke_1)	0.096750	0.028065	0.011078	0.114504	4.079952	0.008363	1.097616

แปลผล

- ได้ค่า lift ที่มากที่สุด เท่ากับ 4.720578 คือ {'age_group_60+', 'glucose_category_Very high', 'work_type_Private'} และ {'ever_married_Yes', 'stroke_1'} หมายความว่า คนในช่วงอายุ 60 ปีขึ้นไป ที่มีน้ำตาลในเลือดสูงมากจนเสี่ยงเป็นโรคเบาหวาน และมีธุรกิจส่วนตัว มีแนวโน้มที่จะแต่งงานแล้วและเป็น Stroke 4.720578 เท่า เมื่อเทียบกับโอกาสที่จะเกิดขึ้นทั้งหมด
- {'age_group_60+', 'glucose_category_Very high', 'work_type_Private'} และ {'ever_married_Yes', 'stroke_1'} ได้ค่า support เท่ากับ 0.0100935 หมายความว่า มี 1% ของชุดข้อมูลนี้ที่มีทั้ง {'age_group_60+', 'glucose_category_Very high', 'work_type_Private'} และ {'ever_married_Yes', 'stroke_1'} ปรากฏอยู่
- {'age_group_60+', 'glucose_category_Very high', 'work_type_Private'} และ {'ever_married_Yes', 'stroke_1'} ได้ค่า confidence เท่ากับ 0.195238 หมายความว่า 19.52% ของชุดข้อมูลนี้ คนในช่วงอายุ 60 ปีขึ้นไป ที่มีน้ำตาลในเลือดสูงมากจนเสี่ยงเป็นโรคเบาหวาน และมีธุรกิจส่วนตัว จะแต่งงานแล้วและเป็น Stroke

THANK YOU