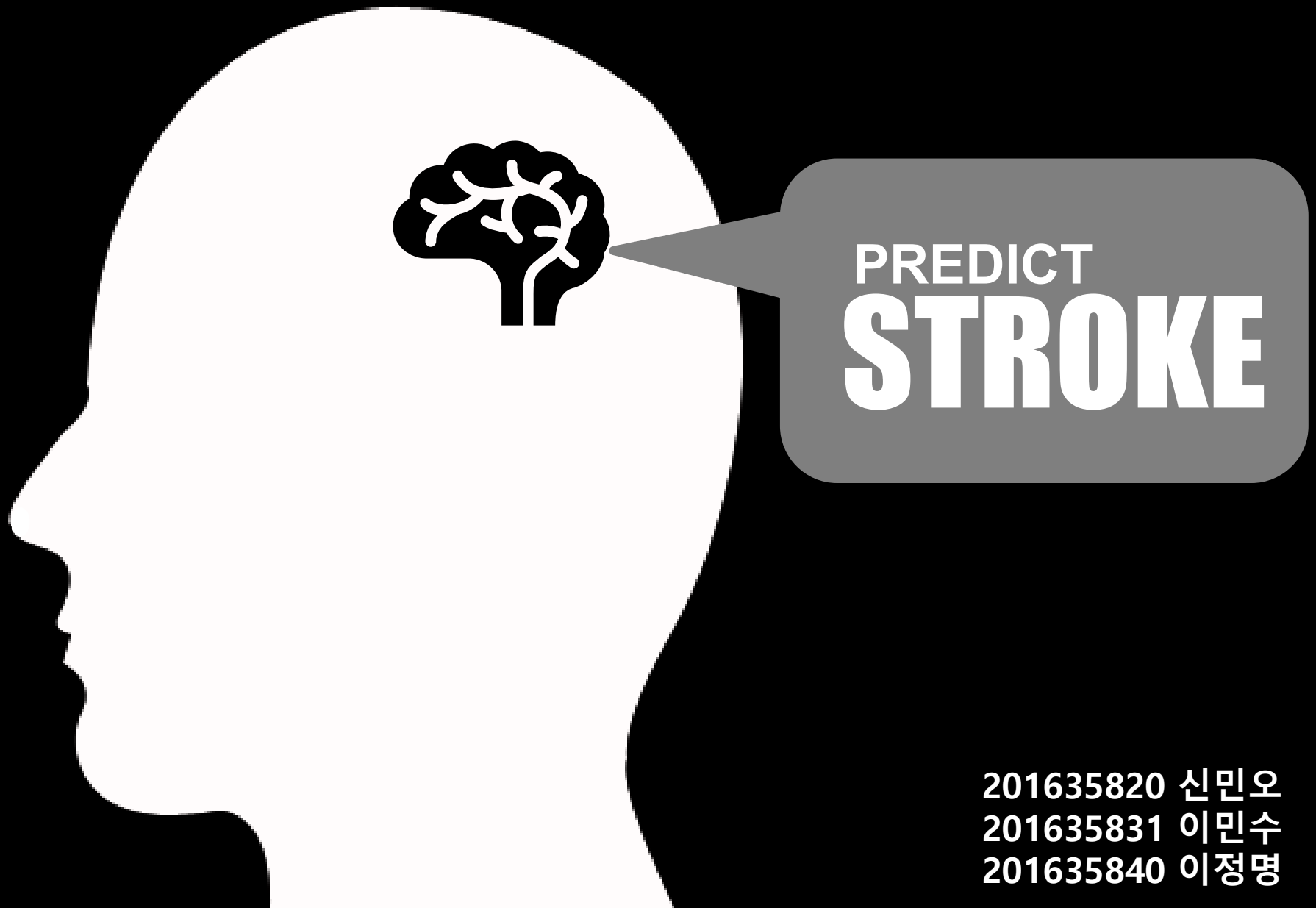
Data Science Term Project



201635820 신민오
201635831 이민수
201635840 이정명

Objective
-Brief description

Data
- Curation
- Inspection
- Preprocesssing
- Analysis
- Evaluation

Role

Objective

- Brief description

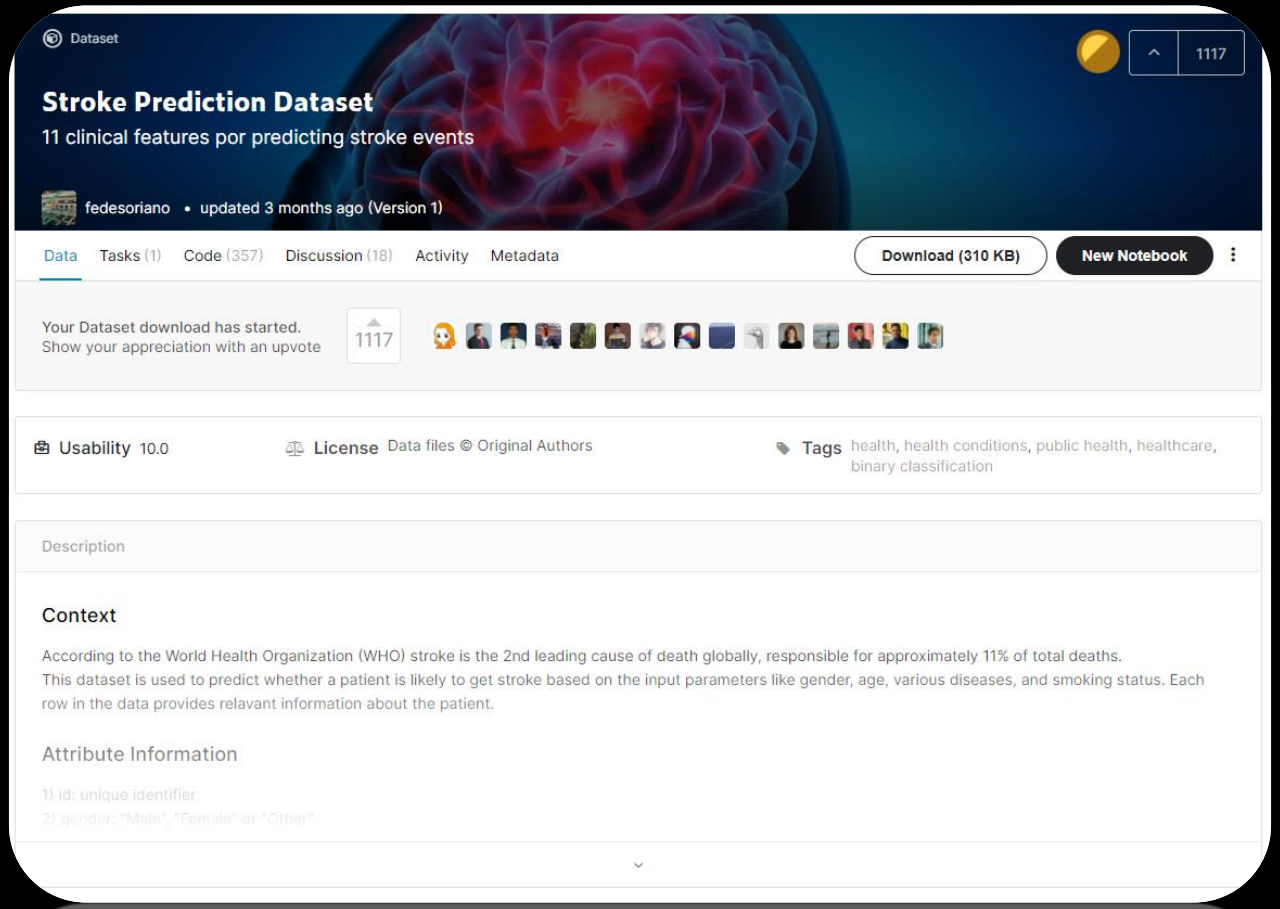
Among the mortality rates by cause in Korea, stroke is second only to cancer, and it is 70.3 per 100,000 population. This accounts for 13.9% of all deaths.

We try to identify the cause and prevent stroke through data analysis.

Data
- Curation

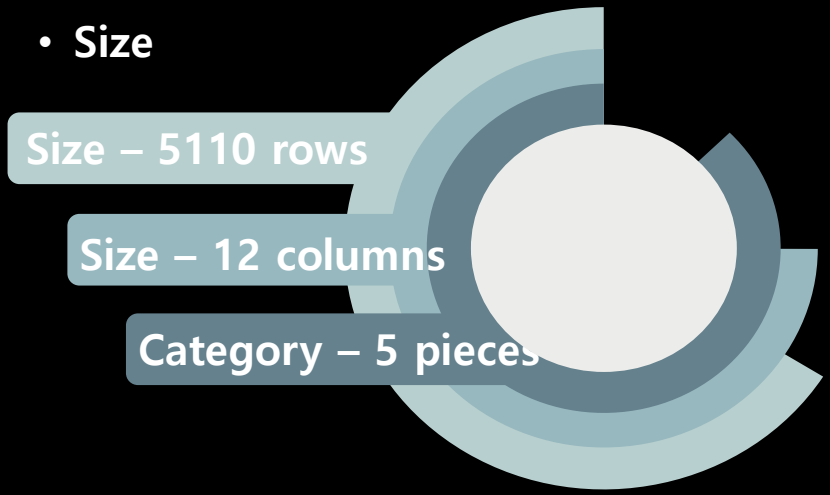
Source of data citation

https://www.kaggle.com/fedesoriano/stroke-prediction-dataset



Data
- Inspection

- Size



- Data format

	id	gender	age	...	bmi	smoking_status	stroke
0	9046	Male	67.0	...	36.6	formerly smoked	1
1	51676	Female	61.0	...	NaN	never smoked	1
2	31112	Male	80.0	...	32.5	never smoked	1
3	60182	Female	49.0	...	34.4	smokes	1
4	1665	Female	79.0	...	24.0	never smoked	1
...	...	...	...	...	...	...	...
5105	18234	Female	80.0	...	NaN	never smoked	0
5106	44873	Female	81.0	...	40.0	never smoked	0
5107	19723	Female	35.0	...	30.6	never smoked	0
5108	37544	Male	51.0	...	25.6	formerly smoked	0
5109	44679	Female	44.0	...	26.2	Unknown	0

[5110 rows x 12 columns]

- Category

id	gender	age	hypertens	heart_dis	ever_mar	work_type	Residence	avg_glucc	bmi	smoking_	stroke

Data
- Inspection

Data.head()

	id	gender	age	...	bmi	smoking_status	stroke
0	9046	Male	67.0	...	36.6	formerly smoked	1
1	51676	Female	61.0	...	NaN	never smoked	1
2	31112	Male	80.0	...	32.5	never smoked	1
3	60182	Female	49.0	...	34.4	smokes	1
4	1665	Female	79.0	...	24.0	never smoked	1

Data.describe()

	id	age	...	bmi	stroke
count	5110.000000	5110.000000	...	4909.000000	5110.000000
mean	36517.829354	43.226614	...	28.893237	0.048728
std	21161.721625	22.612647	...	7.854067	0.215320
min	67.000000	0.080000	...	10.300000	0.000000
25%	17741.250000	25.000000	...	23.500000	0.000000
50%	36932.000000	45.000000	...	28.100000	0.000000
75%	54682.000000	61.000000	...	33.100000	0.000000
max	72940.000000	82.000000	...	97.600000	1.000000

Data
- Inspection

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
```

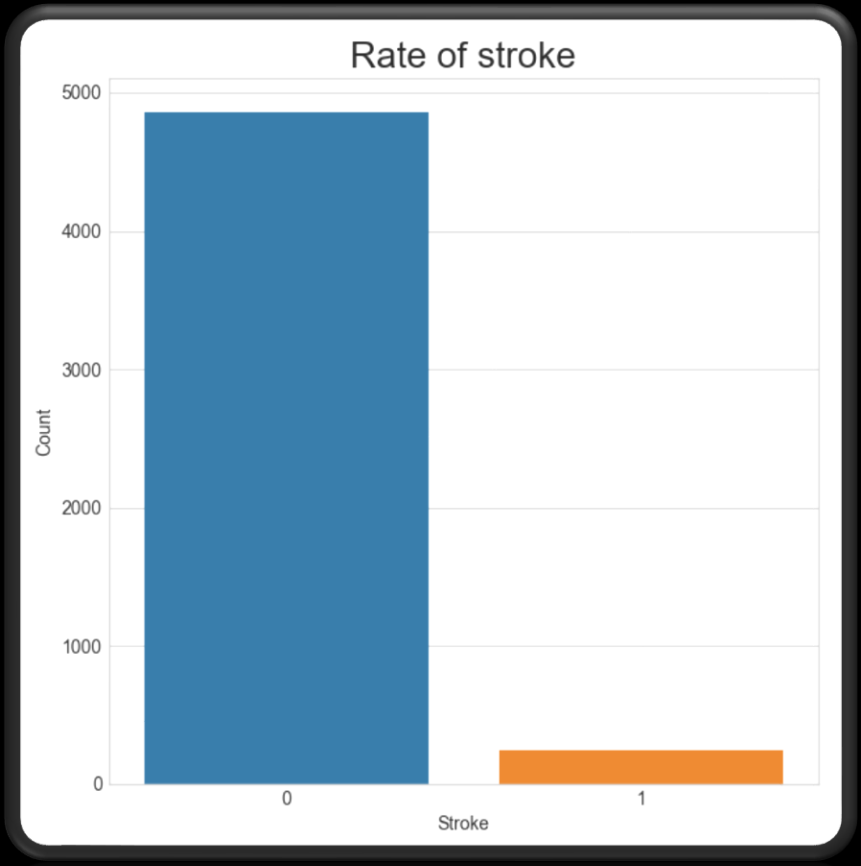
Data.isNull()

```
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
```

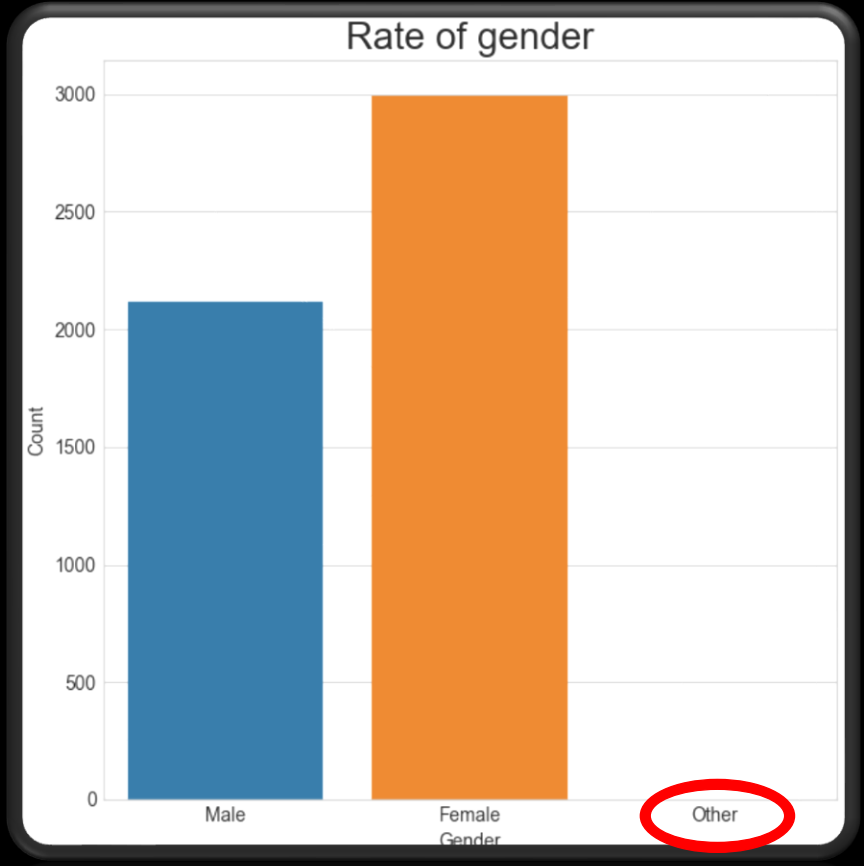
age	hypertens	heart_dise	ever_marr	work_type	Residence	avg_glucc	bmi	smoking_	stroke
67	0	1	Yes	Private	Urban	228.69	36.6	formerly s	1
61	0	0	Yes	Self-empl	Rural	202.2	N/A	never smc	1
80	0	1	Yes	Private	Rural	105.92	32.5	never smc	1
49	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
79	1	0	Yes	Self-empl	Rural	174.12	24	never smc	1
81	0	0	Yes	Private	Urban	186.21	29	formerly s	1
74	1	1	Yes	Private	Rural	70.09	27.4	never smc	1
69	0	0	No	Private	Urban	94.39	22.8	never smc	1
59	0	0	Yes	Private	Rural	76.15	N/A	Unknown	1
78	0	0	Yes	Private	Urban	58.57	24.2	Unknown	1
81	1	0	Yes	Private	Rural	80.43	29.7	never smc	1
61	0	1	Yes	Govt_job	Rural	120.46	36.8	smokes	1
54	0	0	Yes	Private	Urban	104.51	27.3	smokes	1
78	0	1	Yes	Private	Urban	219.84	N/A	Unknown	1

Data
- Inspection

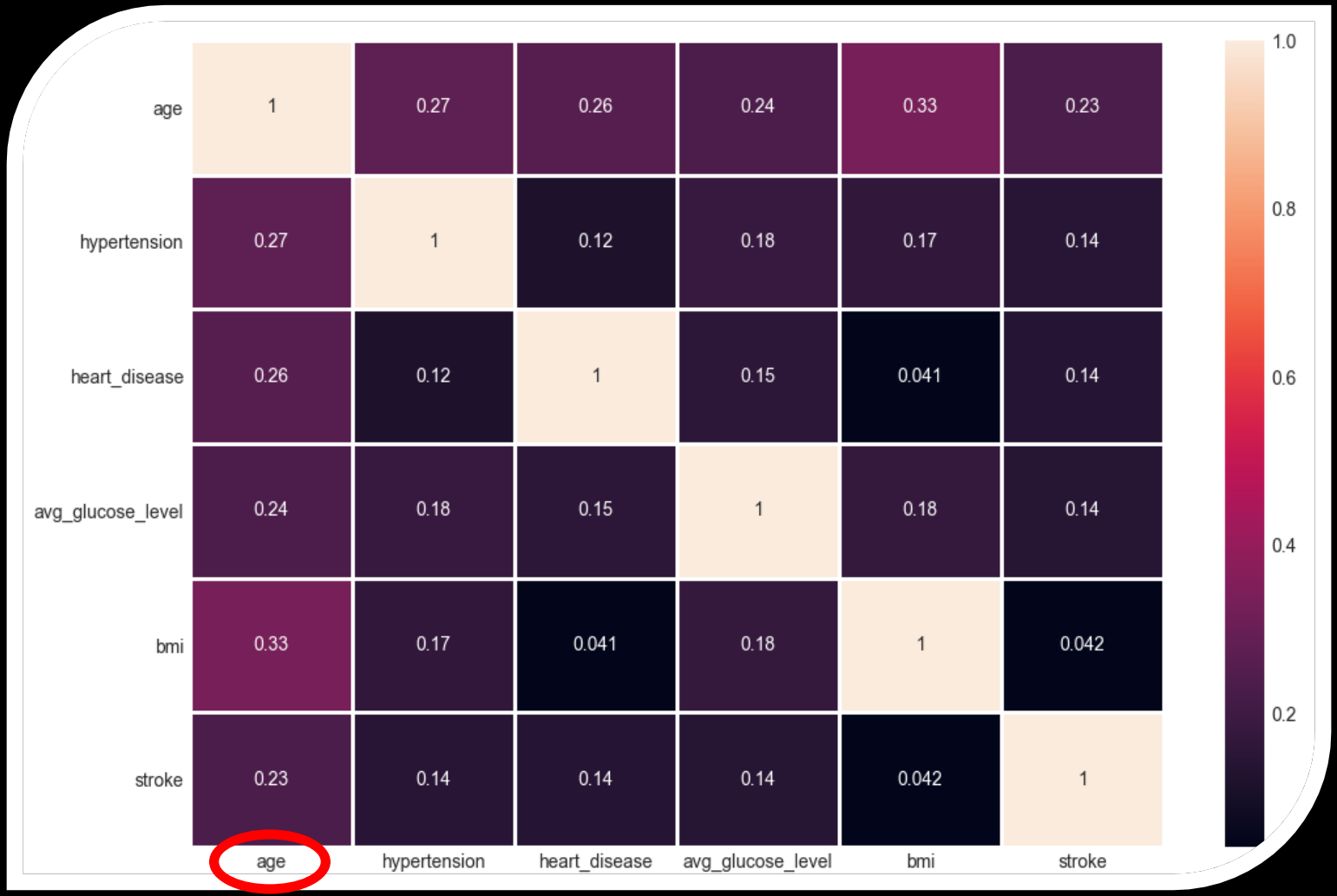
Rate of stroke



Rate of gender



Data
- Inspection



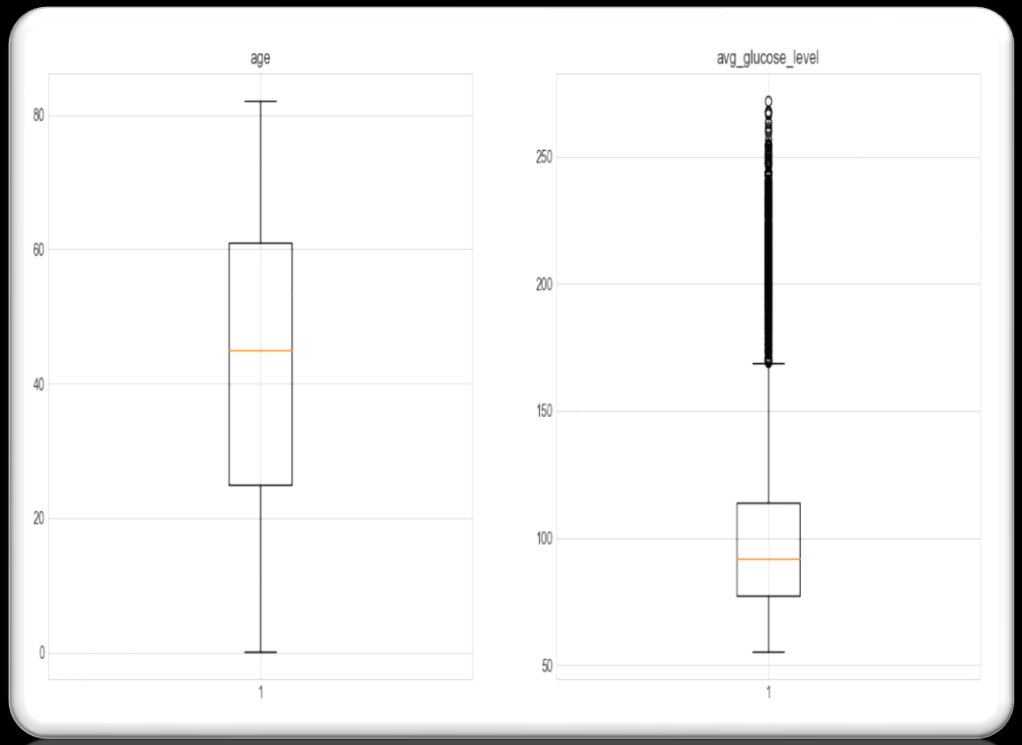
Data
- Inspection

```
####FeatureScore####
                            feature       Score
0                               age  279.980918
1                      hypertension  101.729361
3                 avg_glucose_level   96.585072
2                     heart_disease   95.175560
8                   ever_married_No   54.796734
9                  ever_married_Yes   54.796734
14               work_type_children   32.384214
17           smoking_status_Unknown   27.769605
18  smoking_status_formerly smoked   16.175335
13          work_type_Self-employed   15.082497
4                               bmi    8.826500
20            smoking_status_smokes    2.275740
12                work_type_Private    1.094600
11           work_type_Never_worked    0.982498
19      smoking_status_never smoked    0.564306
6                       gender_Male    0.236267
5                     gender_Female    0.230321
15             Residence_type_Rural    0.178514
16             Residence_type_Urban    0.178514
10               work_type_Govt_job    0.061951
7                     gender_Other    0.044459
```

```python
bestfeature = SelectKBest(f_classif, k='all')
fit = bestfeature.fit(x, y)

dfcolumns = pd.DataFrame(x.columns)
dfscores = pd.DataFrame(fit.scores_)

featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['feature', 'Score']

print("####FeatureScore####")
print(featureScores.nlargest(60, 'Score'))
```

Data
- Inspection

Find Outlier




```
# find outlier
fig, ax = plt.subplots(1, 2, figsize=(16, 4))
ax[0].boxplot(dataset['age'])
ax[0].set_title("age")
ax[1].boxplot(dataset['avg_glucose_level'])
ax[1].set_title("avg_glucose_level")
plt.show()
```

Data
- Preprocessing

Drop null data

```
dataset.dropna(inplace=True)
print("")
print("####After drop null####")
print(dataset.isnull().sum())
```

```
####After drop null####
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                  0
smoking_status       0
stroke               0
dtype: int64
```

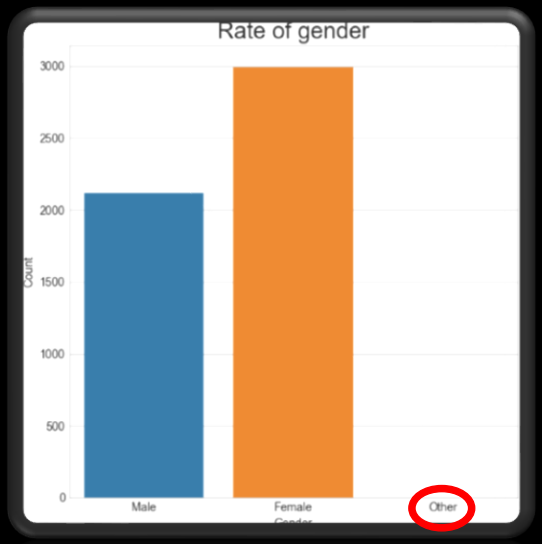
Drop unnecessary column(ID)

```
dataset = dataset.drop(['id'], axis=1)
print("")
print("####After drop unnecessary columns####")
print(dataset.head())
```

```
####After drop unnecessary columns####
   gender   age  hypertension  ...   bmi   smoking_status stroke
0    Male  67.0             0  ...  36.6  formerly smoked      1
2    Male  80.0             0  ...  32.5     never smoked      1
3  Female  49.0             0  ...  34.4           smokes      1
4  Female  79.0             1  ...  24.0     never smoked      1
5    Male  81.0             0  ...  29.0  formerly smoked      1
```

Data
- Preprocessing

Drop wrong value(gender)



Only one value of gender was in error.

Rows : 4909 -> 4908

```
indexNames = dataset[(dataset['gender'] != 'Male')
                & (dataset['gender'] != 'Female')].index
dataset.drop(indexNames, inplace=True)
print("####After drop invalid gender####")
print(dataset)
```

Data
- Data Analysis

Our team's function

-> A function that compares the accuracy to find the optimal encoding and scaling

3 Encoder

Label encoder
Ordinal encoder
OneHot encoder

✖

4 Scaler

Standard scaler
Robust scaler
MaxAbs scaler
MinMax scaler

= 12 Types

```
# function make preprocessing combination
# input dataframe
def makeCombination(x, y):
    listObj=['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
    encoder = [lblEncoding(listObj, x), ordEncoding(listObj, x), ohEncoding(x)]
    nameEnc=['Label encoder', 'Ordinal encoder', 'OneHot encoder']
    scaler = [preprocessing.StandardScaler(), preprocessing.RobustScaler(), preprocessing.MaxAbsScaler(), preprocessing.MinMaxScaler()]
    nameSc=['Standard scaler', 'Robust scaler', 'MaxAbs scaler', 'MinMax scaler']
    listDf=[]
    listBestDf=[]
    listClassifier=[DecisionTreeClassifier(), RandomForestClassifier(), KNeighborsClassifier()]
```

Data
- Data Analysis

Our team's function

-> A function that compares the accuracy to find the optimal encoding and scaling

```
for i in range(len(listClassifier)):
    classifer=listClassifier[i]
    scoreMax=0
    indexMax=0
    encBest=''
    scBest=''
    print(classifer)
    for j in range(len(listDf)):
        trainSetX, testSetX, trainSetY, testSetY = train_test_split(listDf[j], y, test_size=0.2, shuffle=False, random_state=1)
        classifer.fit(trainSetX, trainSetY)
        score=classifer.score(testSetX, testSetY)
        print(score)
        if(scoreMax<=score):
            scoreMax=score
            indexMax=j
    listBestDf.append(listDf[indexMax])
    encBest=nameEnc[(int)(indexMax/4)]
    scBest=nameSc[indexMax%4]
    print("####Function result####")
    print("Best accuracy :", scoreMax)
    print("Best combination : Encoding -> ", encBest, "  Scaling -> ", scBest)
```

```
####Function result####
Best accuracy : 0.9985974754558204
Best combination : Encoding ->  Label encoder    Scaling ->  Standard scaler
```

Data
- Preprocessing

Encoding(Label encoder)

```
# Function to encode column within the data frame
def OneHotEncoding(data_frame, column):
    label_encoder = OneHotEncoder()
    return label_encoder.fit_transform(data_frame.loc[:, column].values)

# Label encoding for Categorical values (Sex: 'M'= 1, 'F'= 0)
dataset['gender'] = OneHotEncoding(dataset, 'gender')

# Label encoding for Categorical values ('Unknow' : 0, 'formerly smoked' : 1, 'never smoke' : 2, 'smoke' : 3)
dataset['smoking_status'] = OneHotEncoding(dataset, 'smoking_status')
```

```
####After encoding
      gender   age  hypertension  ...   bmi  smoking_status  stroke
0          1  67.0             0  ...  36.6               1       1
2          1  80.0             0  ...  32.5               2       1
3          0  49.0             0  ...  34.4               3       1
4          0  79.0             1  ...  24.0               2       1
5          1  81.0             0  ...  29.0               1       1
...      ...   ...           ...  ...   ...             ...     ...
5104       0  13.0             0  ...  18.6               0       0
5106       0  81.0             0  ...  40.0               2       0
5107       0  35.0             0  ...  30.6               2       0
5108       1  51.0             0  ...  25.6               1       0
5109       0  44.0             0  ...  26.2               0       0
```

Data
- Preprocessing

Scaling(Standard scaling)

```
scaler = StandardScaler()
dataset = scaler.fit_transform(dataset)

print("####After scaling####")
print(dataset)
```

```
####After scaling####
[[ 1.20024032  1.06993757 -0.31810241 ...  0.98114481 -0.35182832
   4.74165093]
 [ 1.20024032  1.64633634 -0.31810241 ...  0.45908589  0.58510786
   4.74165093]
 [-0.83316648  0.27184695 -0.31810241 ...  0.70101563  1.52204404
   4.74165093]
 ...
 [-0.83316648 -0.34889019 -0.31810241 ...  0.21715615  0.58510786
  -0.21089701]
 [ 1.20024032  0.36052369 -0.31810241 ... -0.41950107 -0.35182832
  -0.21089701]
 [-0.83316648  0.05015511 -0.31810241 ... -0.3431022  -1.28876449
  -0.21089701]]
```

Data
- Data Analysis

Data split

```
X = dataset.drop(['stroke'], axis=1)
y = dataset.pop('stroke')

# data split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
print("####Check train, test data shape.####")
print('Number transations x_train df', X_train.shape)
print('Number transations x_test df', X_test.shape)
print('Number transations y_train df', y_train.shape)
print('Number transations y_test df', y_test.shape)
```

```
####Check train, test data shape.####
Number transations x_train df (3435, 10)
Number transations x_test df (1473, 10)
Number transations y_train df (3435,)
Number transations y_test df (1473,)
```

Data
- Data Analysis

Decision Tree

```
# grid rf
trainSetX, testSetX, trainSetY, testSetY = train_test_split(listBestDf[1], y2, test_size=0.2, shuffle=True, random_state=1)
param_grid = [{'max_features': np.arange(1, len(testSetX.columns)), 'max_depth': np.arange(1, 10)}]
rf_gscv = GridSearchCV(listClassifier[1], param_grid, cv=2, n_jobs=2)
rf_gscv.fit(trainSetX, trainSetY)
print(rf_gscv.best_params_)
print('Best score :', rf_gscv.best_score_)
```

```
dt = DecisionTreeClassifier(max_depth=3, max_features=8)
```

Accuracy

```
Accuracy on test set : 0.9032586558044806
```

Data

- Data Analysis

Random Forest

```python
# grid rf
trainSetX, testSetX, trainSetY, testSetY = train_test_split(listBestDf[1], y2, test_size=0.2, shuffle=True, random_state=1)
param_grid = [{'max_features': np.arange(1, len(testSetX.columns)), 'max_depth': np.arange(1, 10)}]
rf_gscv = GridSearchCV(listClassifier[1], param_grid, cv=2, n_jobs=2)
rf_gscv.fit(trainSetX, trainSetY)
print(rf_gscv.best_params_)
print('Best score :', rf_gscv.best_score_)
```

```python
rf = RandomForestClassifier(max_depth=9, max_features=8)
```

Accuracy

```
Accuracy on test set : 0.9470468431771895
```

Data
- Data Analysis

KNN

```
# grid rf
trainSetX, testSetX, trainSetY, testSetY = train_test_split(listBestDf[1], y2, test_size=0.2, shuffle=True, random_state=1)
param_grid = [{'max_features': np.arange(1, len(testSetX.columns)), 'max_depth': np.arange(1, 10)}]
rf_gscv = GridSearchCV(listClassifier[1], param_grid, cv=2, n_jobs=2)
rf_gscv.fit(trainSetX, trainSetY)
print(rf_gscv.best_params_)
print('Best score :', rf_gscv.best_score_)
```

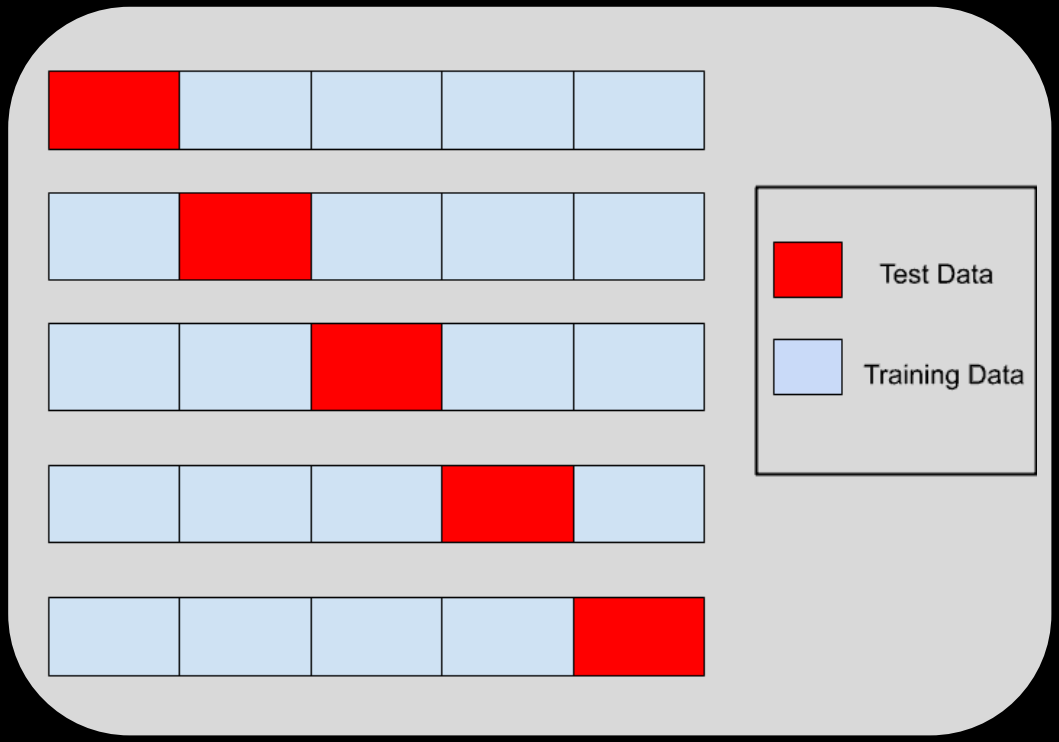
```
knn = KNeighborsClassifier(n_neighbors=6)
```

Accuracy

```
Accuracy on test set : 0.9490835030549898
```

Data
- Evaluation

K-fold cross validation



Decision Tree → `[0.90966921 0.93757962 0.92738854 0.92229299 0.92484076]`

Random Forest → `[0.95928753 0.96178344 0.96050955 0.95923567 0.9566879 ]`

KNN → `[0.95801527 0.9566879  0.95923567 0.96050955 0.9566879 ]`

Data
- Evaluation

Confusion Matrix & Report

Decision Tree

```
Confusion metrics
 [[883  48]
 [ 47   4]]
Precision : 0.9494623655913978
Recall : 0.9484425349087003
F1 score : 0.9489521762493284

Classification report
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       931
           1       0.08      0.08      0.08        51

    accuracy                           0.90       982
   macro avg       0.51      0.51      0.51       982
weighted avg       0.90      0.90      0.90       982
```

Random Forest

```
Confusion metrics
 [[930   1]
 [ 51   0]]
Precision : 0.9480122324159022
Recall : 0.9989258861439313
F1 score : 0.9728033472803348

Classification report
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       931
           1       0.00      0.00      0.00        51

    accuracy                           0.95       982
   macro avg       0.47      0.50      0.49       982
weighted avg       0.90      0.95      0.92       982
```

KNN

```
Confusion metrics
 [[931   0]
 [ 50   1]]
Precision : 0.9490316004077471
Recall : 1.0
F1 score : 0.9738493723849372

Classification report
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       931
           1       1.00      0.02      0.04        51

    accuracy                           0.95       982
   macro avg       0.97      0.51      0.51       982
weighted avg       0.95      0.95      0.93       982
```

Role



Curation
Inspection
Preprocesssing
Analysis
Evaluation

We are going to work on it **all together**.

Thank You