



## Intro to classification - kNN - 2

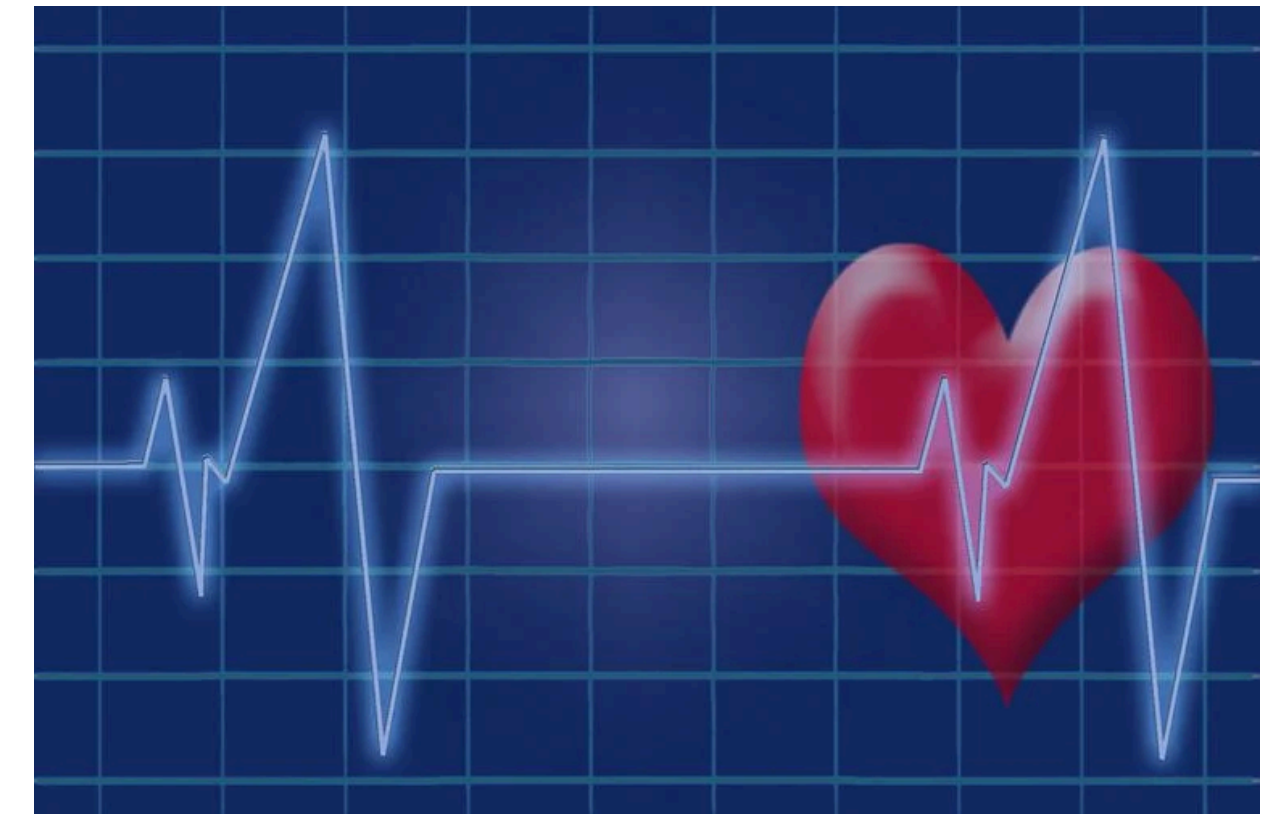
*One should look for what is and not what he thinks should be. (Albert Einstein)*

# Module completion checklist

Objective	Complete
Clean and transform data to run kNN	
Define cross-validation and discuss how to use it	

# Stroke Prediction survey: case study

- According to the World Health Organization (WHO), stroke is the 2nd leading cause of death globally
- **Click here** to see a dataset showing the results of a clinical trial of a heart-disease drug survey on a sample of US adults
- Each row in the data provides relevant information about the adult, including whether they had a stroke or not
- We would like to use this data to predict whether a patient is likely to have a stroke based on their demographic information and medical history



# Dataset

- In order to implement what you learn in this course, we will be using the `healthcare-dataset-stroke-data` dataset
- We will be working with columns from the dataset such as:
  - `stroke`
  - `gender`
  - `age`
  - `hypertension`
  - `heart_disease`
  - `ever_married`
- We will be using different columns of the dataset to predict `stroke` as the target variable

# Loading packages

Let's load the packages we will be using:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

# We will introduce it when we use it
import pickle
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import scale, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import metrics
```

# Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into variables
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your course folder
- Let `data_dir` be the variable corresponding to your data folder

```
# Set 'main_dir' to location of the project folder
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

# Load data into Python

- Let's load the entire dataset
- We are now going to use the function `read_csv` to read in our `healthcare-dataset-stroke-data.csv` dataset

```
df = pd.read_csv(str(data_dir)+"/" + 'healthcare-dataset-stroke-data.csv')  
print(df.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

[5 rows x 12 columns]

# Subset data

- Remove any columns from the dataframe that are not numeric or categorical as we will not be using them in our models

```
df = df[['age', 'avg_glucose_level', 'heart_disease', 'ever_married', 'hypertension',  
'Residence_type', 'gender', 'smoking_status', 'work_type', 'stroke', 'id']]  
print(df.head())
```

	age	avg_glucose_level	heart_disease	...	work_type	stroke	id
0	67.0	228.69	1	...	Private	1	9046
1	61.0	202.21	0	...	Self-employed	1	51676
2	80.0	105.92	1	...	Private	1	31112
3	49.0	171.23	0	...	Private	1	60182
4	79.0	174.12	0	...	Self-employed	1	1665

[5 rows x 11 columns]



# Convert target to binary

- Let's check if the target is binary, and convert it to binary if it is not already

```
# Target not binary - calculate the mean and assign the above mean to 1 and below to 0
threshold = np.mean(df['stroke'])
df['stroke'] = np.where(df['stroke'] > threshold, 1, 0)
```

```
# Target is binary
print(df['stroke'])
```

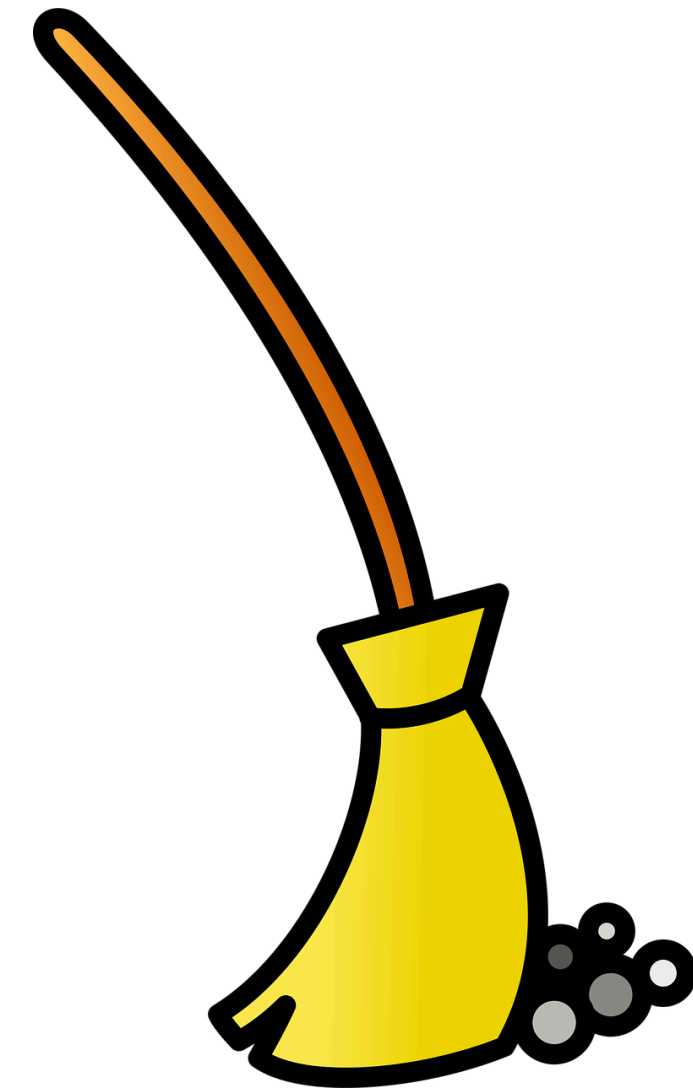
```
0      1
1      1
2      1
3      1
4      1
..
5105    0
5106    0
5107    0
5108    0
5109    0
Name: stroke, Length: 5110, dtype: int64
```

# ID variables

- We will not use certain columns such as ID variables or variables with more than 50% NAs
  - **id**
- We want to see if the independent variables would predict `stroke` well

# Data cleaning steps for kNN

- There are a few steps to remember to take before jumping into splitting the data and training the model
- Let's look at what it means to scale our predictors, and why it's necessary with kNN
- We will also talk through why we need to make sure the target variable is labeled
  - i. Make sure the target is labeled
  - ii. Check for NAs (null values)
  - iii. Encode categorical data into numerical
  - iv. Split into train and test
  - v. Scale the predictors



# The data at first glance

- Look at the first 3 rows and the data types

```
# The first 3 rows.  
print(df.head(3))
```

```
   age  avg_glucose_level  heart_disease  ...  
work_type  stroke      id  
0  67.0          1    228.69          1  ...  
Private  
1  61.0          1    202.21          0  ...  Self-  
employed  
2  80.0          1    105.92          1  ...  
Private  
[3 rows x 11 columns]
```

```
# The data types.  
print(df.dtypes)
```

```
age          float64  
avg_glucose_level  float64  
heart_disease    int64  
ever_married    object  
hypertension    int64  
Residence_type  object  
gender          object  
smoking_status  object  
work_type       object  
stroke          int64  
id              int64  
dtype: object
```

- Frequency table of the target variable

```
print(df['stroke'].value_counts())
```

```
0    4861  
1     249  
Name: stroke, dtype: int64
```

# Data prep: check for NAs

- We now check for NAs and there are multiple methods to deal with them

```
# Check for NAs.  
print(df.isnull().sum())
```

```
age                0  
avg_glucose_level  0  
heart_disease      0  
ever_married       0  
hypertension       0  
Residence_type     0  
gender             0  
smoking_status     1544  
work_type          0  
stroke             0  
id                 0  
dtype: int64
```

# Data prep: check for NAs

- If we do have NA, we could replace them with a mean or 0

```
percent_missing = df.isnull().sum() * 100 / len(df)
print(percent_missing)
```

```
age                0.000000
avg_glucose_level  0.000000
heart_disease      0.000000
ever_married       0.000000
hypertension       0.000000
Residence_type     0.000000
gender             0.000000
smoking_status     30.215264
work_type          0.000000
stroke             0.000000
id                0.000000
dtype: float64
```

# Data prep: check for NAs

```
# Delete columns containing either 50% or more than 50% NaN Values
perc = 50.0
min_count = int(((100-perc)/100)*df.shape[0] + 1)
df = df.dropna(axis=1,
               thresh=min_count)
print(df.shape)
```

```
(5110, 11)
```

```
# Function to impute NA in both numeric and categorical columns
def fillna(df):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    df_numeric = df.select_dtypes(include=numerics)
    df_non_numeric = df.select_dtypes(exclude=numerics)

    df_numeric = df_numeric.fillna(df_numeric.mean())

    for column in df_non_numeric.columns:
        df_non_numeric[column].fillna(df_non_numeric[column].mode()[0], inplace=True)

    df = pd.concat([df_numeric, df_non_numeric], axis=1)
    return df
df = fillna(df)
```

```
/opt/conda/envs/sdaia-python-classification/lib/python3.7/site-
packages/pandas/core/generic.py:6383: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

# Data prep: ready for kNN

- The next step of our data cleanup is to ensure the target variable is binary and has a label
- Let's look at the `dtype` of `stroke`

```
print(df['stroke'].dtypes)
```

```
int64
```

- We want to convert this to `bool` so that is a binary class

```
# Identify the the two unique classes
unique_values = sorted(df['stroke'].unique())
df['stroke'] = np.where(df['stroke'] == unique_values[0], False, True)
```

```
# Split the data into X and y
columns_to_drop_from_X = ['stroke'] + ['id']
X = df.drop(columns_to_drop_from_X, axis = 1)
y = np.array(df['stroke'])
```



# Data prep: numeric variables

- In kNN, we use **numeric data** as predictors
- In some cases, we can **convert categorical data to integer values**

```
print(X.dtypes)
```

```
age                float64
avg_glucose_level  float64
heart_disease      int64
hypertension       int64
ever_married       object
Residence_type     object
gender             object
smoking_status     object
work_type          object
dtype: object
```

```
X = pd.get_dummies(X, columns = ['heart_disease',
                                  'ever_married', 'hypertension', 'Residence_type',
                                  'gender', 'smoking_status', 'work_type'],
                    dtype=float, drop_first=True)
print(X.dtypes)
```

```
age                float64
avg_glucose_level  float64
heart_disease_1    float64
ever_married_Yes   float64
hypertension_1     float64
Residence_type_Urban float64
gender_Male        float64
gender_Other       float64
smoking_status_never smoked float64
smoking_status_smokes float64
work_type_Never_worked float64
work_type_Private  float64
work_type_Self-employed float64
work_type_children float64
dtype: object
```

# Module completion checklist

Objective	Complete
Clean and transform data to run kNN	✓
Define cross-validation and discuss how to use it	

# Introducing cross-validation

- Before applying any machine learning algorithms on the data, we usually need to split the data into a **train set** and a **test set**
- But now, we are doing this **multiple times**
- We have a new **test set** for each fold  $n$
- The rest of the data is the **train set**



# Why do we use cross-validation?

- Cross-validation is helpful in multiple ways:
  - It tunes our model better by running it multiple times on our data (instead of just once on the train set and once on the test set)
  - You get assurance that your model has most of the patterns from the data correct and it's not picking up too much on the noise
  - It finds optimal parameters for your model because it runs multiple times

# Cross-validation: train and test

## Train

- This is the data that you train your model on
- Use a larger portion of the data to train so that the model gets a large enough sample of the population
- Usually about **70%** of your dataset
- **When there is not a large population to begin with, cross-validation techniques can be implemented**

## Test

- This is the data that you test your model on
- Use a smaller portion to test your trained model on
- Usually about **30%** of your dataset
- **When cross-validation is implemented, small test sets will be held out multiple times**

# Cross-validation: n-fold

Here is how cross-validation works:

1. Split the dataset into several subsets (“n” number of subsets) of equal size
2. **Use each subset as the test dataset** and **use the rest of the data as the training dataset**
3. Repeat the process for every subset you create

Test	Data	x	y	z
	1	...	...	...
Train	2	...	...	...
	3	...	...	...
	4	...	...	...
	5	...	...	...
	6	...	...	...

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Data	x	y	z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

# Train and test: small scale before n-fold

- Before we actually use n-fold cross-validation:
  - We split our data into a train and test set
  - We run kNN initially on the training data

```
# Set the seed.  
np.random.seed(1)  
  
# Split into train and test.  
X_train, X_test, y_train, y_test = train_test_split(X,  
                                                    y,  
                                                    test_size = 0.3)
```

- Success! Now let's scale our predictors

# Data prep: scaling variables

- Once the data is converted to `numeric` (if necessary), we **scale** the dataset to make sure that we can properly calculate the relationship between variables
- There are a few methods to scale data and we will use the `scale` function from `sklearn.preprocessing`
- A few things to remember about `scale`:
  - It is a generic function whose default method **centers** and/or scales the columns of a numeric matrix
  - It will convert your dataset to have a mean of 0 and a standard deviation of 1

## `sklearn.preprocessing.scale`

```
sklearn.preprocessing.scale(X, axis=0, with_mean=True, with_std=True, copy=True) \[source\]
```

Standardize a dataset along any axis

Center to the mean and component wise scale to unit variance.

Read more in the [User Guide](#).

**Parameters:**

- X : {array-like, sparse matrix}**  
The data to center and scale.
- axis : int (0 by default)**  
axis used to compute the means and standard deviations along. If 0, independently standardize each feature, otherwise (if 1) standardize each sample.
- with\_mean : boolean, True by default**  
If True, center the data before scaling.
- with\_std : boolean, True by default**  
If True, scale the data to unit variance (or equivalently, unit standard deviation).
- copy : boolean, optional, default True**  
set to False to perform inplace row normalization and avoid a copy (if the input is already a numpy array or a scipy.sparse CSC matrix and if axis is 1).



# Data prep: scaling variables

- We scale only our predictors
- We scale our X and y predictors separately

```
# Scale X.  
X_train = scale(X_train)  
X_test = scale(X_test)
```

```
print(X_train[0:2])
```

```
[[ 0.60884288  0.63825752 -0.23816135  0.71900176 -0.33037446  0.99414629  
 1.17402064  0.        0.69941096 -0.42534432 -0.06910341  0.86404979  
-0.43810503 -0.3922639 ]  
 [-1.88698872 -0.38666334 -0.23816135 -1.39081718 -0.33037446 -1.00588818  
-0.85177378  0.        0.69941096 -0.42534432 -0.06910341 -1.15734072  
-0.43810503  2.54930418]]
```

```
print(X_test[0:2])
```

```
[[ -0.04508466  0.12994683 -0.24077171  0.73532545 -0.32444284 -1.04061541  
 -0.81405762 -0.02554881 -1.43949446 -0.43189409 -0.0572036  -1.1562397  
 -0.43401854 -0.39840954]  
 [ 1.52709552 -0.64601991 -0.24077171  0.73532545 -0.32444284  0.96096982  
 -0.81405762 -0.02554881 -1.43949446 -0.43189409 -0.0572036  0.86487257  
 -0.43401854 -0.39840954]]
```

# Knowledge check



# Module completion checklist

Objective	Complete
Clean and transform data to run kNN	✓
Define cross-validation and discuss how to use it	✓

# Congratulations on completing this module!

You are now ready to try Tasks 1-8 in the Exercise for this topic

