LAPORAN
PROJECT TEAM BASED

Diajukan untuk memenuhi tugas besar
mata kuliah CII2M3 Pengantar Kecerdasan Buatan

Disusun oleh:

Annisa Izzatul Latifa	1301213328
Firman Hoerulloh	1301213392
Reza Mu'ammar Widyanto	1301210513
Renaldhy Vebryan Hermanto	1301213335

IF -45-08

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM

2022/2023

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas berkat rahmat dan karunia-Nya Laporan project team based “Machine Learning : Seleksi Model, Decision Tree, kNN, Naïve Bayes” ini dapat tersusun tepat pada waktunya sebagaimana yang direncanakan. Laporan ini disusun untuk memenuhi tugas besar mata kuliah Pengantar Kecerdasan Buatan di Telkom University.

Pada kesempatan ini, kami mengucapkan terima kasih kepada Ibu Izzatul Ummah selaku dosen pengampu mata kuliah Pengantar Kecerdasan Buatan kelas IF-45-08 dan pihak-pihak terkait yang telah membantu dalam menyelesaikan Laporan ini. Penulis menyadari Laporan ini masih jauh dari kata sempurna. Hal ini tidak terlepas dari keterbatasan penulis dalam berbagai hal. Oleh karena itu, penulis mengharapkan saran konstruktif dari pembaca agar Laporan ini nantinya dapat menjadi yang lebih baik lagi.

Akhir kata, penulis berharap Laporan ini dapat memberikan manfaat dan wawasan yang berguna bagi semua orang, terutama bagi para pembaca.

Bandung, 03 Juni 2023

Penulis

PERNYATAAN

“Tim Kami mengerjakan tugas ini dengan cara yang tidak melanggar aturan perkuliahan dan kode etik akademisi. Jika melakukan plagiarism atau jenis pelanggaran lainnya, maka Tim kami bersedia diberi nilai E untuk Mata Kuliah ini”.

BAB I
PENDAHULUAN

A. Latar Belakang

Banyak metode yang bisa digunakan dalam klasifikasi teks diantaranya yaitu metode naive bayes, *k-nearest neighbor* (kNN), support vector machine (SVM) decision tree, neural network dan sebagainya. Masing – masing dari metode tersebut mempunyai kelebihan dan kekurangan masing – masing dan tingkat akurasi yang berbeda-beda. dalam pengujian akurasi suatu data kita dapat menggunakan perbandingan antara algoritma diatas. Membandingkan dua algoritma ini dalam klasifikasi data medis dapat membantu diagnosis penyakit, hasil perawatan, dan pengambilan keputusan yang lebih baik.

Algoritma *k-Nearest Neighbor* (kNN) adalah machine learning sederhana yang dapat digunakan untuk dataset dengan keberagaman fitur dan mudah diterapkan untuk menyelesaikan masalah klasifikasi dan regresi dengan mengidentifikasi tetangga terdekat dari titik kueri yang diberikan, sehingga kita dapat menetapkan label kelas ke titik tersebut. disini algoritma *k-Nearest Neighbor* (kNN) dapat bekerja dengan baik karena dataset arrhythmia memiliki banyak fitur seperti informasi tentang jantung, data numerik seperti tekanan darah, denyut nadi, dan kategorikal seperti jenis aritmia. Algoritma *k-Nearest Neighbor* (kNN) menjadi pilihan karena tidak memiliki asumsi distribusi yang kuat dimana kita melihat dalam *Arrhythmia dataset* yang kompleks, seperti distribusi kelas yang tidak teratur.

Algoritma *Naive Bayes* adalah metode klasifikasi data berdasarkan faktor-faktor probabilitas dengan menghitung probabilitas kelas berdasarkan fitur-fitur yang ada dan memilih kelas dengan probabilitas tertinggi. Metode yang cocok untuk klasifikasi biner dan multiclass. Algoritma *Naive Bayes* memiliki keuntungan dalam efisiensi komputasi dan kecepatan secara kesederhanaan dalam implementasi. *Naive bayes* dapat memperoleh informasi dari atribut kategorikal dan menghasilkan prediksi

yang akurat dimana *Arrhythmia dataset* mencakup atribut kategorikal seperti jenis *aritmia*. Saat kita membandingkan kedua algoritma tersebut dapat membantu dalam memahami kekuatan dan kelemahan masing-masing metode, dan menurut kami dalam kasus Arrhythmia Dataset algoritma *k-Nearest Neighbor* (kNN) dan *Naive Bayes* dapat menjadi pilihan yang tepat untuk mengevaluasi performa dan akurasi prediksi.

B. Informasi Arrhythmia Dataset

Judul	*Arrhythmia Dataset*
Sumber	UCI *Machine Learning Repository* (https://archive.ics.uci.edu/ml/datasets/Arrhythmia)
Deskripsi	*Arrhythmia Dataset* ini berisi informasi medis mengenai pasien yang didiagnosa memiliki penyakit a*rrhythmia* seperti informasi klinis dan hasil tes yang dilakukan pada organ jantung(elektrokardiogram) dari pasien yang terdiagnosa. *Arrhythmia* adalah gangguan pada detak jantung atau irama jantung yang ditandai dengan detak jantung yang tidak teratur, bisa terlalu cepat atau terlalu lambat yang dapat menyebabkan komplikasi penyakit pada organ lain, bahkan kematian mendadak.
Tujuan	Tujuan dari dibuatnya *dataset* ini adalah untuk membedakan antara ada dan tidak adanya aritmia jantung serta mengklasifikasikannya ke dalam salah satu dari enam belas kelas. Menurut pemilik asli dari *dataset* ini, pada tahun 1998 sudah ada program komputer yang membuat klasifikasi aritmia. Namun, terdapat perbedaan hasil antara klasifikasi oleh kardiolog dan oleh program komputer. Oleh karena itu, *dataset* ini bertujuan untuk membantu meminimalkan perbedaan tersebut melalui alat pembelajaran mesin.
Sumber data	Rumah sakit dan lembaga kesehatan.
Tahun data	1998-01-01
Jumlah sampel	452 sampel
Jumlah atribut	280 atribut
Tipe data	Kategorikal dan numerik (integer dan real)

Kolom target	kelas adalah yang menunjukkan klasifikasi ada tidaknya aritmia jantung. terdapat beberapa kelas 01 : mengacu pada EKG 'normal' 02 : Ischemic changes (Coronary Artery Disease) 03 : Old Anterior Myocardial Infarction 04 : Old Inferior Myocardial Infarction 05 : Sinus tachycardy 06 : Sinus bradycardy 07 : Ventricular Premature Contraction (PVC) 08 : Supraventricular Premature Contraction 09 : Left bundle branch block 10 : Right bundle branch block 11 : 1. degree AtrioVentricular block 12 : 2. degree AV block 13 : 3. degree AV block 14 : Left ventricule hypertrophy 15 : Atrial Fibrillation or Flutter 16 : Others

C. Pra-pemrosesan data

tahap pra-pemrosesan data adalah tahap untuk menyiapkan data.

a) Library

- *k-Nearest Neighbor*(kNN)

```
import urllib.request
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

- *Naive Bayes*

```
import urllib.request
import pandas as pd
import numpy as np
import math
```

b) Membaca *dataset* dan menambah kolom

```
# melakukan import dataset dari URL
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/arrhythmia/arrhythmia.data'
filename = 'arrhythmia.data'

urllib.request.urlretrieve(url, filename)
```

```
# Membaca dataset
df = pd.read_csv('arrhythmia.data', header=None)

column_names = [
    'age','gender','height','weight','qrs_duration','p-r_Interval','q-t_interval','t_interval','p_interval',
            'QRS','T','P','QRST','J','heart_rate','Q_D1','R_D1','S_D1','R\'_D1','S\'_D1','NOD_D1','EOR_R_D1','EOD_R_D1',
            'EOR_P_D1','EOD_P_D1','EOR_T_D1','EOD_T_D1','Q_D2','R_D2','S_D2','R\'_D2','S\'_D2','NOD_D2','EOR_R_D2','EOD_R_D2',
            'EOR_P_D2','EOD_P_D2','EOR_T_D2','EOD_T_D2','Q_D3','R_D3','S_D3','R\'_D3','S\'_D3','NOD_D3','EOR_R_D3','EOD_R_D3',
            'EOR_P_D3','EOD_P_D3','EOR_T_D3','EOD_T_D3','Q_AVR','R_AVR','S_AVR','R\'_AVR','S\'_AVR','NOD_AVR','EOR_R_AVR',
            'EOD_R_AVR','EOR_P_AVR','EOD_P_AVR','EOR_T_AVR','EOD_T_AVR','Q_AVL','R_AVL','S_AVL','R\'_AVL','S\'_AVL','NOD_AVL',
            'EOR_R_AVL','EOD_R_AVL','EOR_P_AVL','EOD_P_AVL','EOR_T_AVL','EOD_T_AVL','Q_AVF','R_AVF','S_AVF','R\'_AVF','S\'_AVF',
            'NOD_AVF','EOR_R_AVF','EOD_R_AVF','EOR_P_AVF','EOD_P_AVF','EOR_T_AVF','EOD_T_AVF','Q_V1','R_V1','S_V1','R\'_V1',
            'S\'_V1','NOD_V1','EOR_R_V1','EOD_R_V1','EOR_P_V1','EOD_P_V1','EOR_T_V1','EOD_T_V1','Q_V2','R_V2','S_V2','R\'_V2',
            'S\'_V2','NOD_V2','EOR_R_V2','EOD_R_V2','EOR_P_V2','EOD_P_V2','EOR_T_V2','EOD_T_V2','Q_V3','R_V3','S_V3','R\'_V3',
            'S\'_V3','NOD_V3','EOR_R_V3','EOD_R_V3','EOR_P_V3','EOD_P_V3','EOR_T_V3','EOD_T_V3','Q_V4','R_V4','S_V4','R\'_V4',
            'S\'_V4','NOD_V4','EOR_R_V4','EOD_R_V4','EOR_P_V4','EOD_P_V4','EOR_T_V4','EOD_T_V4','Q_V5','R_V5','S_V5','R\'_V5',
            'S\'_V5','NOD_V5','EOR_R_V5','EOD_R_V5','EOR_P_V5','EOD_P_V5','EOR_T_V5','EOD_T_V5','Q_V6','R_V6','S_V6','R\'_V6',
            'S\'_V6','NOD_V6','EOR_R_V6','EOD_R_V6','EOR_P_V6','EOD_P_V6','EOR_T_V6','EOD_T_V6','A_JJ_D1','A_Q_D1','A_R_D1',
            'A_S_D1','A_R\'_D1','A_S\'_D1','A_P_D1','A_T_D1','QRSA_D1','QRSTA_D1','A_JJ_D2','A_Q_D2','A_R_D2','A_S_D2',
            'A_R\'_D2','A_S\'_D2','A_P_D2','A_T_D2','QRSA_D2','QRSTA_D2','A_JJ_D3','A_Q_D3','A_R_D3','A_S_D3','A_R\'_D3',
            'A_S\'_D3','A_P_D3','A_T_D3','QRSA_D3','QRSTA_D3','A_JJ_AVR','A_Q_AVR','A_R_AVR','A_S_AVR','A_R\'_AVR','A_S\'_AVR',
            'A_P_AVR','A_T_AVR','QRSA_AVR','QRSTA_AVR','A_JJ_AVL','A_Q_AVL','A_R_AVL','A_S_AVL','A_R\'_AVL','A_S\'_AVL','A_P_AVL',
            'A_T_AVL','QRSA_AVL','QRSTA_AVL','A_JJ_AVF','A_Q_AVF','A_R_AVF','A_S_AVF','A_R\'_AVF','A_S\'_AVF','A_P_AVF','A_T_AVF',
            'QRSA_AVF','QRSTA_AVF','A_JJ_V1','A_Q_V1','A_R_V1','A_S_V1','A_R\'_V1','A_S\'_V1','A_P_V1','A_T_V1','QRSA_V1','QRSTA_V1',
            'A_JJ_V2','A_Q_V2','A_R_V2','A_S_V2','A_R\'_V2','A_S\'_V2','A_P_V2','A_T_V2','QRSA_V2','QRSTA_V2','A_JJ_V3','A_Q_V3',
            'A_R_V3','A_S_V3','A_R\'_V3','A_S\'_V3','A_P_V3','A_T_V3','QRSA_V1','QRSTA_V3','A_JJ_V4','A_Q_V4','A_R_V4','A_S_V4',
            'A_R\'_V4','A_S\'_V4','A_P_V4','A_T_V4','QRSA_V4','QRSTA_V4','A_JJ_V5','A_Q_V5','A_R_V5','A_S_V5','A_R\'_V5','A_S\'_V5',
            'A_P_V5','A_T_V5','QRSA_V5','QRSTA_V5','A_JJ_V6','A_Q_V6','A_R_V6','A_S_V6','A_R\'_V6','A_S\'_V6','A_P_V6','A_T_V6',
            'QRSA_V6','QRSTA_V6','Class'
]

df.columns = column_names
```

c) pada penelitian ini kami menggunakan semua atribut

```
df.head()  # Menampilkan beberapa baris pertama dataset
```

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_V6	Class
0	75	0	190	80	91	193	371	174	121	-16	...	0.0	9.0	-0.9	0.0	0.0	0.9	2.9	23.3	49.4	8
1	56	1	165	64	81	174	401	149	39	25	...	0.0	8.5	0.0	0.0	0.0	0.2	2.1	20.4	38.8	6
2	54	0	172	95	138	163	386	185	102	96	...	0.0	9.5	-2.4	0.0	0.0	0.3	3.4	12.3	49.0	10
3	55	0	175	94	100	202	380	179	143	28	...	0.0	12.2	-2.2	0.0	0.0	0.4	2.6	34.6	61.6	1
4	75	0	190	80	88	181	360	177	103	-16	...	0.0	13.1	-3.6	0.0	0.0	-0.1	3.9	25.4	62.8	7

d) memeriksa *missing value*

```
# melakukan pemeriksaan data yang tidak lengkap
missing_values = dict()
for column in df:
    for data in df[column]:
        if data == '?':
            missing_values[column] = missing_values.get(column, 0) + 1

print("Jumlah data yang hilang:")
for keys, values in missing_values.items():
  print(keys, ":", values)
```

```
Jumlah data yang hilang:
T : 8
P : 22
QRST : 1
J : 376
heart_rate : 1
```

e) mengisi *missing value* dengan rata-rata dari data yang tidak hilang

```
# menghapus baris dengan data yang hilang
def data_clean(df):
  # dari pemeriksaan data di atas, didapat bahwa kolom 'heart_rate', 'T', 'P', 'QRST', dan 'J' memiliki data yang hilang
  for i in ['heart_rate', 'T', 'P', 'QRST', 'J']:
    df[i] = df[i].replace('?','-1').astype(int)
    mean = df[i].mean(axis=0)
    df[i] = df[i].replace(-1,mean)
  return df
```

f) visualisasi data

- *dataset* lengkap yang berisikan jumlah data, rata-rata, standar deviasi minimal, nilai kuartil 1, kuartil 2, kuartil 3, dan maksimum

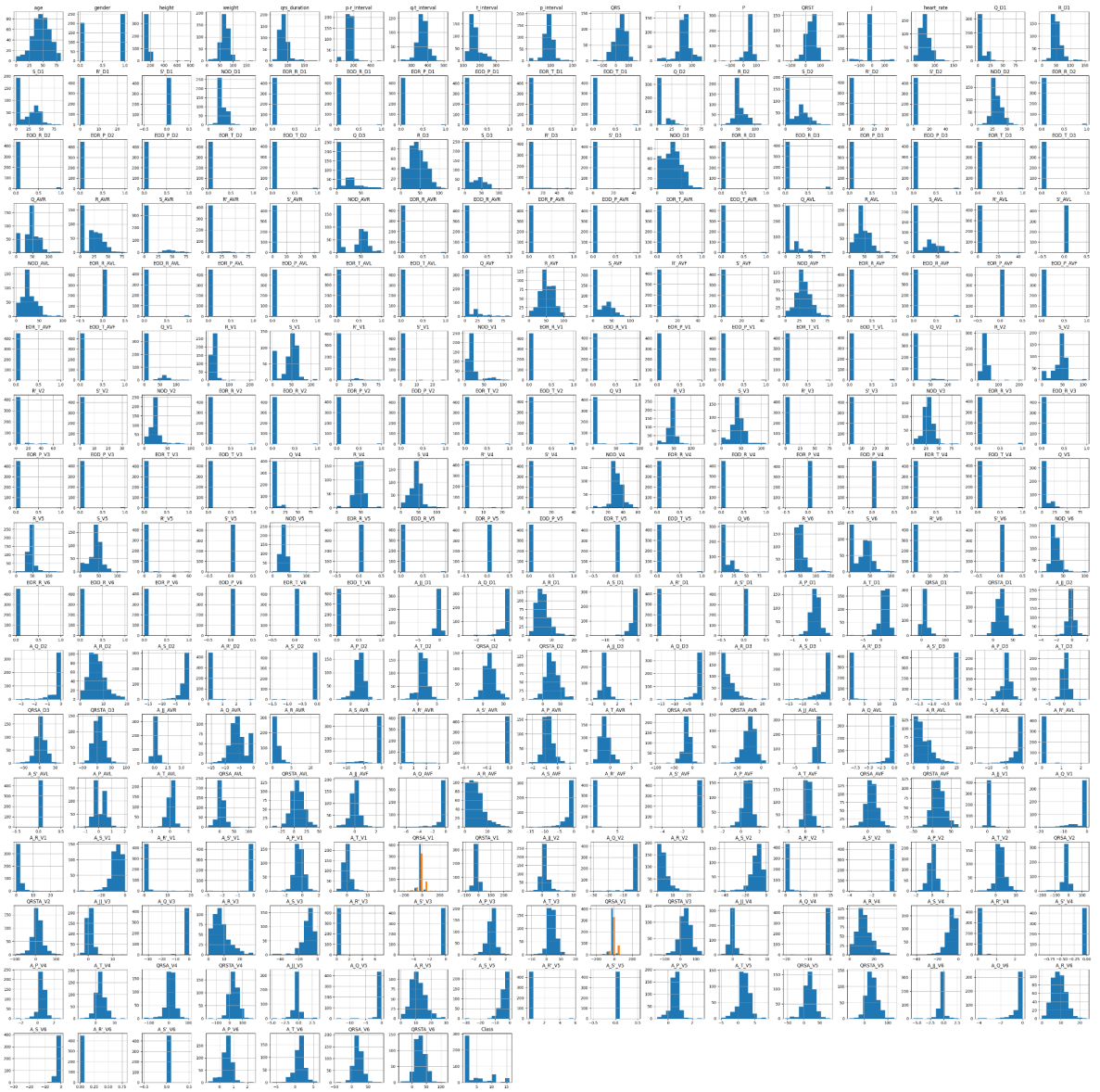
```
df.describe() # Menampilkan ringkasan statistik deskriptif untuk setiap kolom numerik
```

	age	gender	height	weight	qrs_duration	p-r_interval	q-t_interval	t_interval	p_interval	QRS	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_V6	Class
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	...	452.000000	452.000000	452.000000	452.000000	452.0	452.000000	452.000000	452.000000	452.000000	452.000000
mean	46.471239	0.550885	166.188053	68.170354	88.920354	155.152655	367.207965	169.949115	90.004425	33.676991	...	-0.278982	9.048009	-1.457301	0.003982	0.0	0.514823	1.222345	19.326106	29.473230	3.880531
std	16.466631	0.497955	37.170340	16.590803	15.364394	44.842283	33.385421	35.633072	25.826643	45.431434	...	0.548876	3.472862	2.002430	0.050118	0.0	0.347531	1.426052	13.503922	18.493927	4.407097
min	0.000000	0.000000	105.000000	6.000000	55.000000	0.000000	232.000000	108.000000	0.000000	-172.000000	...	-4.100000	0.000000	-28.600000	0.000000	0.0	-0.800000	-6.000000	-44.200000	-38.600000	1.000000
25%	36.000000	0.000000	160.000000	59.000000	80.000000	142.000000	350.000000	148.000000	79.000000	3.750000	...	-0.425000	6.600000	-2.100000	0.000000	0.0	0.400000	0.500000	11.450000	17.550000	1.000000
50%	47.000000	1.000000	164.000000	68.000000	86.000000	157.000000	367.000000	162.000000	91.000000	40.000000	...	0.000000	8.800000	-1.100000	0.000000	0.0	0.500000	1.350000	18.100000	27.900000	1.000000
75%	58.000000	1.000000	170.000000	79.000000	94.000000	175.000000	384.000000	179.000000	102.000000	66.000000	...	0.000000	11.200000	0.000000	0.000000	0.0	0.700000	2.100000	25.825000	41.125000	6.000000
max	83.000000	1.000000	780.000000	176.000000	188.000000	524.000000	509.000000	381.000000	205.000000	169.000000	...	0.000000	23.600000	0.000000	0.800000	0.0	2.400000	6.000000	88.800000	115.900000	16.000000

8 rows × 280 columns

- visualisasi seluruh atribut

```
# Menggambar histogram untuk setiap kolom numerik
df.hist(figsize=(50, 50))
plt.show()
```



link gambar :

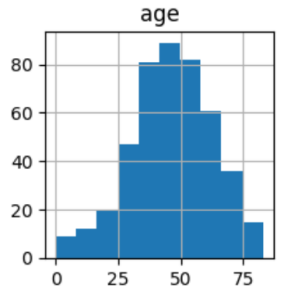
https://drive.google.com/file/d/17qbPXNg1ApAtROnDoAA6NvCb6VDX75AM/view?usp=share_link

D. Plot kumpulan data

beberapa diagram yang mewakili kumpulan data

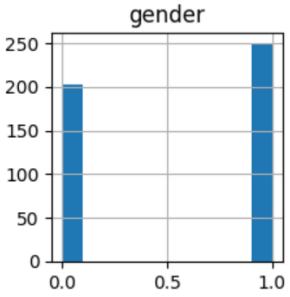
a) Age

dari 451 data terlihat umur pasien paling banyak umur 40-50 dengan jumlah diatas 80 orang



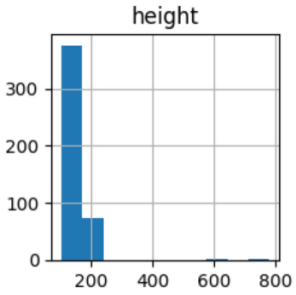
b) *Gender*

dari 451 data terlihat paling banyak pasien perempuan dengan jumlah 250 orang dan untuk pasien laki-laki berjumlah 200 orang



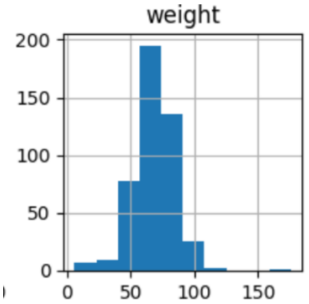
c) *Height*

dari 451 data terlihat tinggi badan paling banyak pada >100 dan <200 dengan jumlah diatas 300 orang



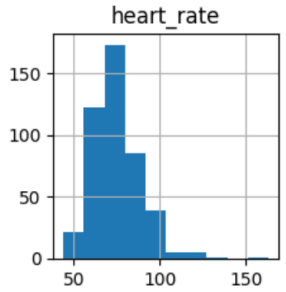
d) *Weight*

dari 451 data terlihat berat badan paling banyak pada rentang >50 dan <75 dengan jumlah diatas 150 orang



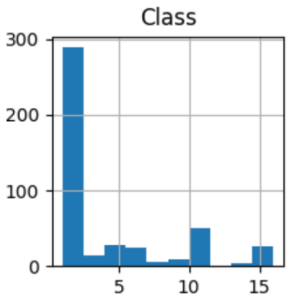
e) *Heart rate*

dari 451 data terlihat *heart rate* paling banyak pada >60 dan < 80 bpm dengan jumlah lebih dari 150 orang



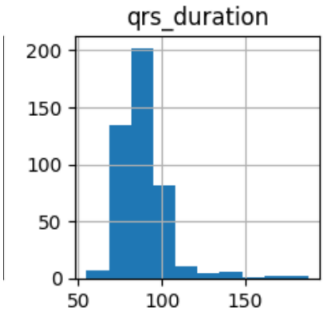
f) Class

dari 451 data terlihat *Arrhthmia Class* paling banyak berada pada kelas normal karena mendekati 0 dengan jumlah diatas dari 200



g) qrs_duration

dari 451 data terlihat durasi qrs paling banyak antara 80-100 ms dengan jumlah diatas 200 orang



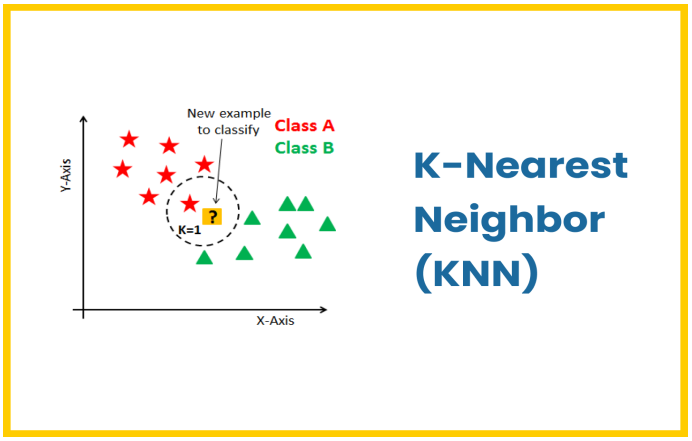
BAB II
DASAR TEORI

A. Algoritma Learning kNN (k-Nearest Neighbors)

Algoritma learning kNN *(k-Nearest Neighbors)* adalah algoritma paling sederhana yang merupakan sebuah metode untuk melakukan klasifikasi terhadap objek berdasarkan data pembelajaran *(train datasets)* yang jaraknya paling dekat dengan objek tersebut. *Nearest Neighbor* adalah sebuah metode untuk mencari kasus dengan menghitung kedekatan antara kasus baru dan kasus lama berdasarkan kecocokan bobot dari sejumlah fitur yang ada dan bersifat *non-parametric* dan *lazy learning.*

Teknik pencarian tetangga terdekat yang umum dilakukan dengan menggunakan formula jarak euclidean

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



a) Cara kerja algoritma *k-Nearest Neighbor*(kNN)

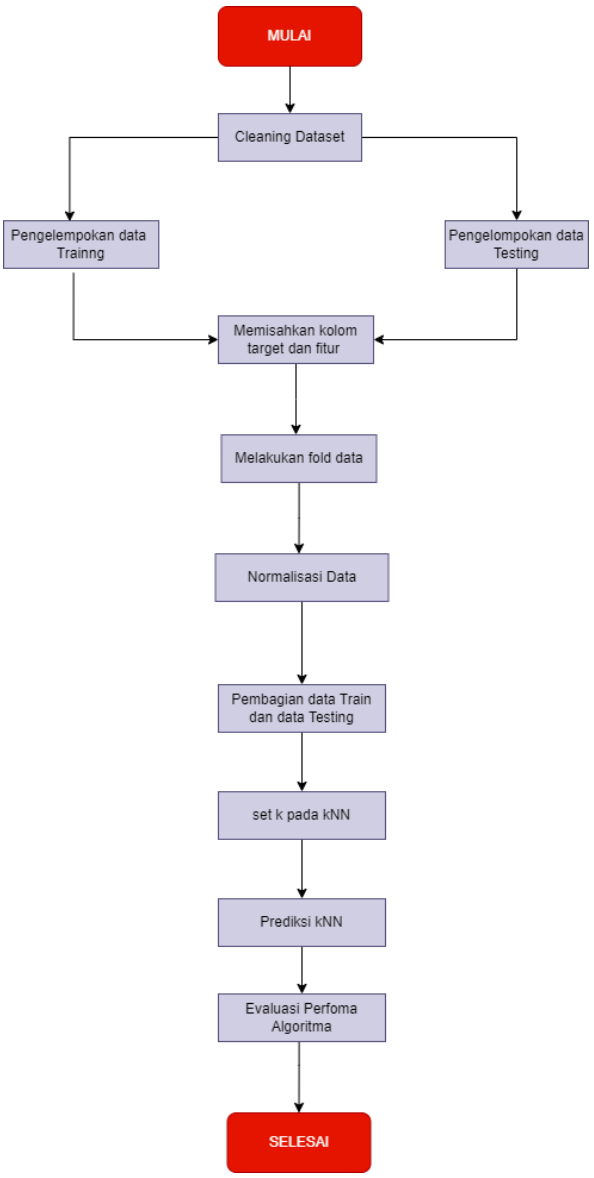
1. menentukan parameter k yang akan digunakan untuk pertimbangan penentuan kelas. jika kelas berjumlah genap maka sebaiknya nilai K-nya ganjil, sebaliknya jika kelas berjumlah ganjil maka sebaiknya nilai K-nya genap.
2. Menghitung jarak menggunakan metode euclidean distance
3. menentukan tetangga terdekat yaitu memilih k sesuai jarak yang dihitung
4. menentukan mayoritas kelas dengan menghitung titik data di setiap kategori

5. menentukan titik data baru ke kategori yang jumlah tetangganya banyak

b) Implementasi algoritma *k-Nearest Neighbor*(kNN) dalam dataset arrhythmia

1. Load dataset Arrhythmia ke dalam Python
2. Pisahkan fitur-fitur dan label dari dataset.
3. Lakukan fold pada data
4. Lakukan normalisasi atau standarisasi data jika diperlukan.
5. Bagi dataset menjadi data pelatihan dan data pengujian.
6. Hitung jarak antara data uji dengan setiap data pelatihan menggunakan matrik jarak seperti *Euclidean distance.*
7. Pilih k tetangga terdekat.
8. Gunakan mayoritas kelas dari tetangga terdekat sebagai prediksi kelas untuk data uji.
9. Evaluasi performa algoritma menggunakan matrik evaluasi seperti akurasi.

FlowChart Implementasi Algoritma :



Hasil atau output yang diharapkan dari penggunaan algoritma kNN terhadap dataset Arrhythmia diantaranya yaitu prediksi kelas untuk setiap data uji berdasarkan mayoritas kelas dari tetangga terdekat, matriks evaluasi seperti akurasi, recall, dan F1-score untuk mengukur performa kNN dalam mengklasifikasikan data.

Pemilihan hyperparameter yang bervariasi dari algoritma kNN terhadap dataset Arrhythmia dapat melibatkan beberapa analisis seperti pemilihan jumlah tetangga (nilai k), dimana k merupakan hyperparameter yang menentukan jumlah tetangga yang digunakan untuk melakukan prediksi. Selain itu, analisis jarak atau matriks juga diperlukan, dimana dalam kNN jarak antara data point dan tetangga harus dihitung. Beberapa matrik yang umum digunakan adalah jarak Euclidean, Manhattan, atau Minkowski.

Berikut tangkapan layar implementasi algoritma kNN*(k-Nearest Neighbor)* menggunakan bahasa pemrograman python:

- **Import *library* dan pembacaan *dataset***

Import

```
import urllib.request
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
[2] # melakukan import dataset dari URL
    url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/arrhythmia/arrhythmia.data'
    filename = 'arrhythmia.data'

    urllib.request.urlretrieve(url, filename)
```

1. Melakukan import *library* dan *dataset*

READ CSV

```
# Membaca dataset
df = pd.read_csv('arrhythmia.data', header=None)
column_names = [
    'age','gender','height','weight','qrs_duration','p-r_Interval','q-t_interval','t_interval','p_interval',
            'QRS','T','P','QRST','J','heart_rate','Q_D1','R_D1','S_D1','R\'_D1','S\'_D1','NOD_D1','EOR_R_D1','EOD_R_D1',
            'EOR_P_D1','EOD_P_D1','EOR_T_D1','EOD_T_D1','Q_D2','R_D2','S_D2','R\'_D2','S\'_D2','NOD_D2','EOR_R_D2','EOD_R_D2',
            'EOR_P_D2','EOD_P_D2','EOR_T_D2','EOD_T_D2','Q_D3','R_D3','S_D3','R\'_D3','S\'_D3','NOD_D3','EOR_R_D3','EOD_R_D3',
            'EOR_P_D3','EOD_P_D3','EOR_T_D3','EOD_T_D3','Q_AVR','R_AVR','S_AVR','R\'_AVR','S\'_AVR','NOD_AVR','EOR_R_AVR',
            'EOD_R_AVR','EOR_P_AVR','EOD_P_AVR','EOR_T_AVR','EOD_T_AVR','Q_AVL','R_AVL','S_AVL','R\'_AVL','S\'_AVL','NOD_AVL',
            'EOR_R_AVL','EOD_R_AVL','EOR_P_AVL','EOD_P_AVL','EOR_T_AVL','EOD_T_AVL','Q_AVF','R_AVF','S_AVF','R\'_AVF','S\'_AVF',
            'NOD_AVF','EOR_R_AVF','EOD_R_AVF','EOR_P_AVF','EOD_P_AVF','EOR_T_AVF','EOD_T_AVF','Q_V1','R_V1','S_V1','R\'_V1',
            'S\'_V1','NOD_V1','EOR_R_V1','EOD_R_V1','EOR_P_V1','EOD_P_V1','EOR_T_V1','EOD_T_V1','Q_V2','R_V2','S_V2','R\'_V2',
            'S\'_V2','NOD_V2','EOR_R_V2','EOD_R_V2','EOR_P_V2','EOD_P_V2','EOR_T_V2','EOD_T_V2','Q_V3','R_V3','S_V3','R\'_V3',
            'S\'_V3','NOD_V3','EOR_R_V3','EOD_R_V3','EOR_P_V3','EOD_P_V3','EOR_T_V3','EOD_T_V3','Q_V4','R_V4','S_V4','R\'_V4',
            'S\'_V4','NOD_V4','EOR_R_V4','EOD_R_V4','EOR_P_V4','EOD_P_V4','EOR_T_V4','EOD_T_V4','Q_V5','R_V5','S_V5','R\'_V5',
            'S\'_V5','NOD_V5','EOR_R_V5','EOD_R_V5','EOR_P_V5','EOD_P_V5','EOR_T_V5','EOD_T_V5','Q_V6','R_V6','S_V6','R\'_V6',
            'S\'_V6','NOD_V6','EOR_R_V6','EOD_R_V6','EOR_P_V6','EOD_P_V6','EOR_T_V6','EOD_T_V6','A_JJ_D1','A_Q_D1','A_R_D1',
            'A_S_D1','A_R\'_D1','A_S\'_D1','A_P_D1','A_T_D1','QRSA_D1','QRSTA_D1','A_JJ_D2','A_Q_D2','A_R_D2','A_S_D2',
            'A_R\'_D2','A_S\'_D2','A_P_D2','A_T_D2','QRSA_D2','QRSTA_D2','A_JJ_D3','A_Q_D3','A_R_D3','A_S_D3','A_R\'_D3',
            'A_S\'_D3','A_P_D3','A_T_D3','QRSA_D3','QRSTA_D3','A_JJ_AVR','A_Q_AVR','A_R_AVR','A_S_AVR','A_R\'_AVR','A_S\'_AVR',
            'A_P_AVR','A_T_AVR','QRSA_AVR','QRSTA_AVR','A_JJ_AVL','A_Q_AVL','A_R_AVL','A_S_AVL','A_R\'_AVL','A_S\'_AVL','A_P_AVL',
            'A_T_AVL','QRSA_AVL','QRSTA_AVL','A_JJ_AVF','A_Q_AVF','A_R_AVF','A_S_AVF','A_R\'_AVF','A_S\'_AVF','A_P_AVF','A_T_AVF',
            'QRSA_AVF','QRSTA_AVF','A_JJ_V1','A_Q_V1','A_R_V1','A_S_V1','A_R\'_V1','A_S\'_V1','A_P_V1','A_T_V1','QRSA_V1','QRSTA_V1',
            'A_JJ_V2','A_Q_V2','A_R_V2','A_S_V2','A_R\'_V2','A_S\'_V2','A_P_V2','A_T_V2','QRSA_V2','QRSTA_V2','A_JJ_V3','A_Q_V3',
            'A_R_V3','A_S_V3','A_R\'_V3','A_S\'_V3','A_P_V3','A_T_V3','QRSA_V1','QRSTA_V3','A_JJ_V4','A_Q_V4','A_R_V4','A_S_V4',
            'A_R\'_V4','A_S\'_V4','A_P_V4','A_T_V4','QRSA_V4','QRSTA_V4','A_JJ_V5','A_Q_V5','A_R_V5','A_S_V5','A_R\'_V5','A_S\'_V5',
            'A_P_V5','A_T_V5','QRSA_V5','QRSTA_V5','A_JJ_V6','A_Q_V6','A_R_V6','A_S_V6','A_R\'_V6','A_S\'_V6','A_P_V6','A_T_V6',
            'QRSA_V6','QRSTA_V6','Class'
]
```

2. Membaca dataset dari arrhytmia.data

- **Pemeriksaan Data**

```
[7]  def loss_data(df):
       null = df.isnull().sum()
       for i in range(len(df.columns)):
         if null[i] > 0:
           print(f"{df.columns[i]}: {null[i]} ({(null[i]/len(df))*100}%)")
       total_cells = np.product(df.shape)
       total_missing = null.sum()
       print(f"\nTotal data yang hilang: {total_missing} ({(total_missing/total_cells) * 100}%)\n")
```

1. Fungsi *loss data* digunakan untuk mencari data-data yang hilang pada *dataset*

```
def data_clean(df):
  for i in ['heart_rate', 'T', 'P', 'QRST', 'J']:
    df[i] = df[i].replace('?','-1').astype(int)
    mean = df[i].mean(axis=0)
    df[i] = df[i].replace(-1,mean)
  return df
```

2. Fungsi data_clean digunakan untuk membersihkan data dalam suatu DataFrame, Dengan menggunakan fungsi data_clean, kita dapat membersihkan data pada kolom-kolom yang disebutkan, mengganti nilai yang tidak valid atau hilang dengan nilai rata-rata dari kolom tersebut.

```
# melakukan pemeriksaan ulang data yang tidak lengkap
df = data_clean(df)
loss_data(df)
```

```
Total data yang hilang: 0 (0.0%)
```

3. Menggunakan fungsi data_clean untuk membersihkan dataset arrhythmia dan menggunakan fungsi loss_data untuk melakukan pengecekan lagi. Dari output tersebut didapatkan bahwa sudah tidak ada data yang hilang pada *dataset.*

```
[10] df.info() # Menampilkan informasi tentang dataset, seperti jumlah baris, kolom, tipe data, dll.

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 452 entries, 0 to 451
     Columns: 280 entries, age to Class
     dtypes: float64(125), int64(155)
     memory usage: 988.9 KB
```

4. Menampilkan informasi dari dataset. Dalam dataset arrhythmia terdapat 452 baris dan 280 kolom serta 2 tipe data yaitu float dan int64.

```
df.describe() # Menampilkan ringkasan statistik deskriptif untuk setiap kolom numerik
```

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	...
mean	46.471239	0.550885	166.188053	68.170354	88.920354	155.152655	367.207965	169.949115	90.004425	33.676991	...
std	16.466631	0.497955	37.170340	16.590803	15.364394	44.842283	33.385421	35.633072	25.826643	45.431434	...
min	0.000000	0.000000	105.000000	6.000000	55.000000	0.000000	232.000000	108.000000	0.000000	-172.000000	...
25%	36.000000	0.000000	160.000000	59.000000	80.000000	142.000000	350.000000	148.000000	79.000000	3.750000	...
50%	47.000000	1.000000	164.000000	68.000000	86.000000	157.000000	367.000000	162.000000	91.000000	40.000000	...
75%	58.000000	1.000000	170.000000	79.000000	94.000000	175.000000	384.000000	179.000000	102.000000	66.000000	...
max	83.000000	1.000000	780.000000	176.000000	188.000000	524.000000	509.000000	381.000000	205.000000	169.000000	...

8 rows × 280 columns

5. Menampilkan statistik deskriptif dari *dataset arrhythmia*. dimana didalamnya terdapat count, mean, std, min, max, dan kuartil.

```
# replace missing values
df['height'].sort_values()

df.loc[df["height"] == 608, "height"] = 61
df.loc[df["height"] == 780, "height"] = 78

df.describe()
```

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	...
mean	46.471239	0.550885	163.424779	68.170354	88.920354	155.152655	367.207965	169.949115	90.004425	33.676991	...
std	16.466631	0.497955	12.146961	16.590803	15.364394	44.842283	33.385421	35.633072	25.826643	45.431434	...
min	0.000000	0.000000	61.000000	6.000000	55.000000	0.000000	232.000000	108.000000	0.000000	-172.000000	...
25%	36.000000	0.000000	160.000000	59.000000	80.000000	142.000000	350.000000	148.000000	79.000000	3.750000	...
50%	47.000000	1.000000	164.000000	68.000000	86.000000	157.000000	367.000000	162.000000	91.000000	40.000000	...
75%	58.000000	1.000000	170.000000	79.000000	94.000000	175.000000	384.000000	179.000000	102.000000	66.000000	...
max	83.000000	1.000000	190.000000	176.000000	188.000000	524.000000	509.000000	381.000000	205.000000	169.000000	...

8 rows × 280 columns

6. Melakukan *replace* nilai yang tidak masuk akal, dimana sebelumnya terdapat atribut *height* dengan nilai 608 dan 780. Maka nilai tersebut kami *replace* dengan 61 dan 78. Sehingga dapat dilihat pada atribut height nilai max menjadi 190.

\

- **Melakukan *Fold***

```
# melakukan fold pada data
fold1 = (df.iloc[0:150].reset_index(drop=True), df.iloc[300:452].reset_index(drop=True))
fold2 = (df.iloc[150:300].reset_index(drop=True), pd.concat([df.iloc[0:150], df.iloc[300:452]]).reset_index(drop=True))
fold3 = (df.iloc[300:452].reset_index(drop=True), df.iloc[0:150].reset_index(drop=True))

test, train = fold2
train
```

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_V6	Cla
0	66	1	155	62	80	188	418	178	104	19	...	0.0	13.9	-1.2	0.0	0.0	0.3	0.8	30.7	37.4	
1	54	0	172	95	138	163	386	185	102	96	...	0.0	9.5	-2.4	0.0	0.0	0.3	3.4	12.3	49.0	
2	57	0	175	80	98	157	304	130	78	43	...	-0.7	6.3	0.0	0.0	0.0	0.7	-1.0	18.1	11.7	
3	57	0	175	70	94	148	382	147	100	46	...	-1.2	5.7	0.0	0.0	0.0	0.4	0.7	17.7	21.9	
4	38	1	160	63	79	0	376	165	0	34	...	0.0	10.1	0.0	0.0	0.0	0.0	1.5	26.2	37.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
297	50	1	160	73	75	125	353	183	63	38	...	-0.5	8.1	-0.8	0.0	0.0	0.7	2.5	14.5	39.5	
298	49	1	170	79	85	128	377	151	76	65	...	-1.3	16.1	0.0	0.0	0.0	0.3	2.1	37.3	55.3	
299	50	1	174	90	81	105	362	197	70	-31	...	0.0	5.3	-2.9	0.0	0.0	-0.3	1.6	3.5	18.5	
300	58	0	175	78	95	145	376	202	92	-5	...	-2.4	9.5	0.0	0.0	0.0	0.7	0.2	17.6	18.8	
301	46	0	165	55	82	0	415	108	0	-1	...	0.0	13.1	0.0	0.0	0.0	-0.7	0.4	44.5	48.2	

302 rows × 280 columns

1. Program di atas bertujuan untuk melakukan *fold* atau pembagian data menjadi beberapa subset yang saling tumpang tindih untuk keperluan validasi silang (cross-validation).Setelah melakukan pembagian *fold*, program memilih salah satu *fold* sebagai data train dan menyimpannya dalam variabel *train*. Dalam contoh tersebut, subset *train* dari fold2 dipilih dan disimpan dalam variabel *train*. Output yang dihasilkan adalah DataFrame *train*, yang berisi data dari subset *train* yang telah dipilih.

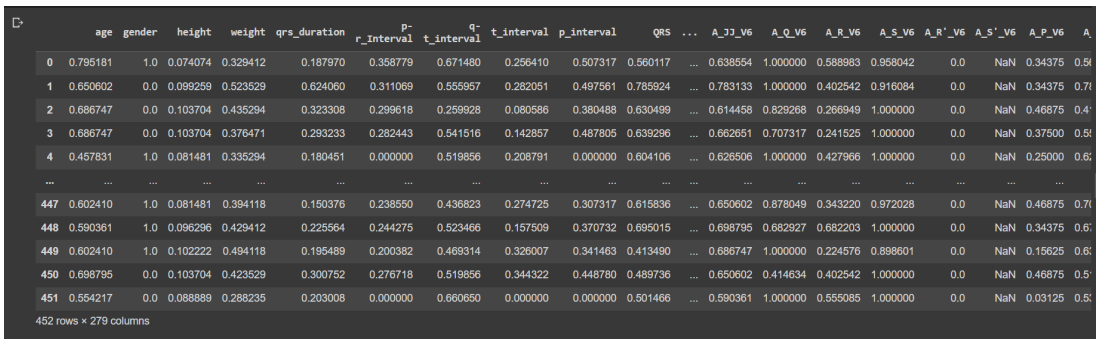
- **Normalisasi**

NORMALISASI

```
# normalisasi dengan min max norm
def norm(df):
  df = (df - df.min()) / (df.max() - df.min())
  return df

# memisahkan kolom target dan fitur
x = df.drop('Class', axis=1) # kolom fitur
y = df.Class #kolom target

# melakukan normalisasi untuk x (fitur-fitur)
x = norm(x)
x
```

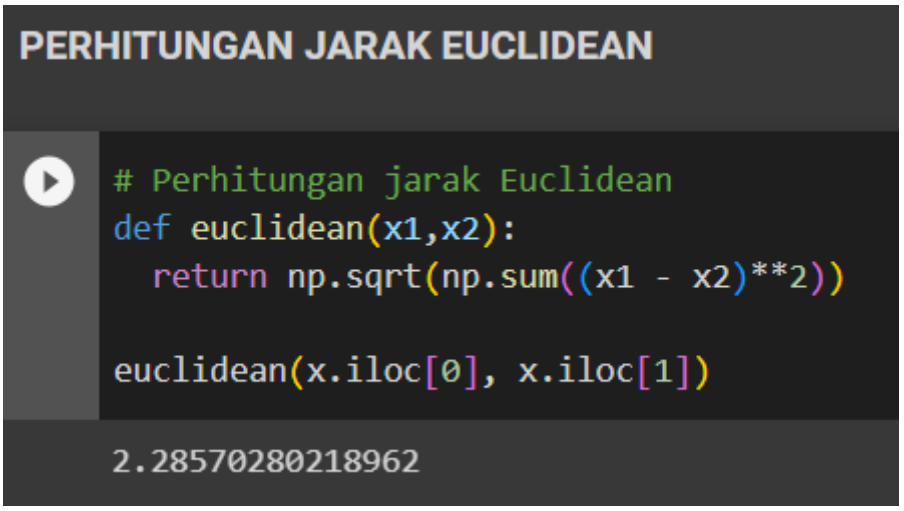


	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...	A_JJ_V6	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A
0	0.795181	1.0	0.074074	0.329412	0.187970	0.358779	0.671480	0.256410	0.507317	0.560117	...	0.638554	1.000000	0.588983	0.958042	0.0	NaN	0.34375	0.5
1	0.650602	0.0	0.099259	0.523529	0.624060	0.311069	0.555957	0.282051	0.497561	0.785924	...	0.783133	1.000000	0.402542	0.916084	0.0	NaN	0.34375	0.7
2	0.686747	0.0	0.103704	0.435294	0.323308	0.299618	0.259928	0.080586	0.380488	0.630499	...	0.614458	0.829268	0.266949	1.000000	0.0	NaN	0.46875	0.4
3	0.686747	0.0	0.103704	0.376471	0.293233	0.282443	0.541516	0.142857	0.487805	0.639296	...	0.662651	0.707317	0.241525	1.000000	0.0	NaN	0.37500	0.5
4	0.457831	1.0	0.081481	0.335294	0.180451	0.000000	0.519856	0.208791	0.000000	0.604106	...	0.626506	1.000000	0.427966	1.000000	0.0	NaN	0.25000	0.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
447	0.602410	1.0	0.081481	0.394118	0.150376	0.238550	0.436823	0.274725	0.307317	0.615836	...	0.650602	0.878049	0.343220	0.972028	0.0	NaN	0.46875	0.7
448	0.590361	1.0	0.096296	0.429412	0.225564	0.244275	0.523466	0.157509	0.370732	0.695015	...	0.698795	0.682927	0.682203	1.000000	0.0	NaN	0.34375	0.6
449	0.602410	1.0	0.102222	0.494118	0.195489	0.200382	0.469314	0.326007	0.341463	0.413490	...	0.686747	1.000000	0.224576	0.898601	0.0	NaN	0.15625	0.6
450	0.698795	0.0	0.103704	0.423529	0.300752	0.276718	0.519856	0.344322	0.448780	0.489736	...	0.650602	0.414634	0.402542	1.000000	0.0	NaN	0.46875	0.5
451	0.554217	0.0	0.088889	0.288235	0.203008	0.000000	0.660650	0.000000	0.000000	0.501466	...	0.590361	1.000000	0.555085	1.000000	0.0	NaN	0.03125	0.5

452 rows × 279 columns

1. Program di atas memiliki beberapa tujuan terkait normalisasi data menggunakan metode Min-Max *normalization*. Fungsi norm(df) Fungsi ini bertujuan untuk melakukan normalisasi data dalam Data Frame df *(Arrhythmia)* menggunakan metode Min-Max *normalization*. Selanjutnya dilakukan pemisahan kolom target dan fitur. Lalu dilakukan normalisasi untuk x (fitur-fitur), output dari program ini adalah DataFrame x yang sudah mengalami normalisasi dengan metode Min-Max *normalization*.

- **Perhitungan Jarak**



1. Fungsi euclidean digunakan untuk menghitung jarak euclidean antara dua vektor. Setelah itu, fungsi ini dipanggil untuk menghitung dua vektor data yang diambil dari data frame. Output dari program ini adalah nilai jarak Euclidean antara x.iloc[0] dan x.iloc[1], yaitu 2.28570280218962. Nilai tersebut merupakan hasil perhitungan jarak Euclidean antara dua vektor berdasarkan formula yang diterapkan dalam fungsi euclidean.

- **Training kNN*(k-Nearest Neighbors)***

TRAINING KNN

```
[22] #Training KNN
     def KNN(x_train, y_train, x_test, k): # -K adalah banyaknya neighbors
       dist = []
       # menghitung distance dari data training dan data testing
       for row in range(x_train.shape[0]):
         dist.append(euclidean(x_train.iloc[row], x_test))

       data = x_train.copy()
       data['Dist'] = dist #menambahkan data distance pada data
       data['Class'] = y_train #menambahkan class pada data
       data = data.sort_values(by = 'Dist').reset_index(drop = True) #mengurutkan data

       #mengambil label kelas yg sering muncul
       y_pred = data.iloc[:k].Class.mode()
       return y_pred[0]
```

Fungsi kNN *(k-Nearest Neighbors)* bertujuan untuk melatih model kNN *(k-Nearest Neighbors)* menggunakan data training dan melakukan prediksi label kelas untuk data testing berdasarkan k-neighbors terdekat. Dengan menggunakan fungsi kNN *(k-Nearest Neighbors)*, kita dapat melatih model kNN*(k-Nearest Neighbors)* dan melakukan prediksi label kelas untuk data testing berdasarkan jumlah neighbors (k) yang ditentukan.

- **Akurasi**

AKURASI

```
[23] # menghitung akurasi dari output berdasar kelas
     def acc(y_pred, y_true):
       true = 0
       for i in range(len(y_pred)):
         if y_pred[i]== y_true[i]:
           true += 1
       return true/len(y_pred)
```

1. Fungsi acc(y_pred, y_true) memiliki makna untuk menghitung akurasi prediksi berdasarkan hasil prediksi (y_pred) dan nilai sebenarnya (y_true). Output dari fungsi ini adalah nilai akurasi, yaitu proporsi prediksi yang benar

dari keseluruhan prediksi yang diberikan. Dengan menggunakan fungsi acc, kita dapat menghitung akurasi dari hasil prediksi dan membandingkannya dengan nilai sebenarnya untuk mengevaluasi performa *model*.

- **Evaluasi**

EVALUASI

```
[24] #evaluasi model dengan data fold
     def evaluate(fold, k):
       test, train = fold
       x_train, y_train = train.drop('Class',axis=1), train.Class
       x_test, y_test = test.drop('Class', axis=1), test.Class
       x_train = norm(x_train)
       x_test = norm(x_test)
       y_preds = []
       for row in range(x_test.shape[0]):
         y_preds.append(KNN(x_train, y_train, x_test.iloc[row], k))

       return (acc(y_preds, y_test))
```

1. Fungsi evaluate(fold, k) memiliki makna untuk mengevaluasi performa model kNN *(k-Nearest Neighbors)* dengan menggunakan data fold yang telah dibagi sebelumnya. Fungsi ini mengembalikan nilai akurasi evaluasi *model*, yaitu proporsi prediksi yang benar dari label kelas sebenarnya pada data test.

```
[77] # Evaluasi kinerja model menggunakan akurasi
     k = 10
     accs = []
     folds = [fold1, fold2, fold1]
     for i in range(len(folds)):
       accs.append(evaluate(folds[i],k))
     print(f'Menggunakan k : {k}, dengan rata-rata akurasi : {sum(accs)/3}')

     Menggunakan k : 10, dengan rata-rata akurasi : 0.5666666666666668
```

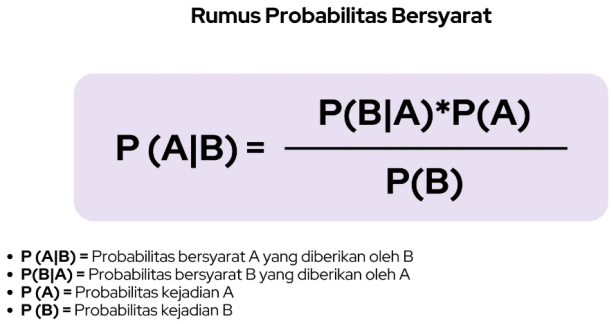
2. Program di atas memiliki makna untuk mengukur kinerja model menggunakan metrik akurasi dengan menggunakan nilai k yang ditentukan pada model kNN *(k-Nearest Neighbors)*. Dimana k diinisialisasi dengan nilai 10. kemudian dilakukan evaluasi model kNN *(k-Nearest Neighbors)* untuk setiap fold. Setelah selesai melakukan evaluasi untuk semua fold, program menghitung rata-rata akurasi dengan menjumlahkan semua nilai akurasi dalam accs dan

membaginya dengan jumlah fold. Output dari program ini adalah rata-rata akurasi evaluasi model kNN *(k-Nearest Neighbors)* dengan menggunakan nilai k yang ditentukan. Dalam kasus ini, outputnya adalah "Menggunakan k: 10, dengan rata-rata akurasi : 0.566666666666666". Ini menunjukkan rata-rata akurasi dari evaluasi model kNN *(k-Nearest Neighbors)* pada data fold dengan menggunakan nilai k = 10.

Link Algoritma KNN :
https://github.com/muammarrz/tubes-ai/blob/3f21f217ff0c6e7291915875ab4852ac6e3b9949/arrhythmia_kNN.ipynb

B. Algoritma Learning Naive Bayes

Algoritma *Naive Bayes* merupakan metode yang cocok untuk klasifikasi biner dan *multi class* dengan variabel input kategori daripada numerik yang menggunakan probabilitas bersyarat, Probabilitas bersyarat adalah ukuran peluang suatu peristiwa yang terjadi berdasarkan peristiwa lain yang telah terjadi.



Naive Bayes memiliki keuntungan dalam efisiensi komputasi, kecepatan serta sederhana dalam implementasi, kemampuan untuk mengatasi dataset dengan jumlah fitur yang besar. algoritma ini rentan terhadap masalah nilai nol dan nilai yang tidak pernah ditemukan dalam probabilitas kondisional. *Naive bayes* perlu memperhatikan pemilihan metode *smoothing,* atribut numerik atau kategorikal, dan data hilang untuk memperoleh hasil prediksi yang lebih baik.

Cara kerja *Naive bayes* :

1. menyiapkan dataset
2. menghitung *probabilitas prior* untuk setiap kelas dengan menghitung jumlah sampel dalam kelas dibagi dengan total jumlah sampel

3. menghitung probabilitas kondisional untuk setiap fitur dalam kondisional dengan menghitung jumlah sampel dalam kelas tertentu yang memiliki fitur tersebut dibagi total jumlah sampel dalam kelas.
4. menggunakan teorema bayes untuk menghitung probabilitas kelas tertentu berdasarkan fitur-fitur yang ada dengan

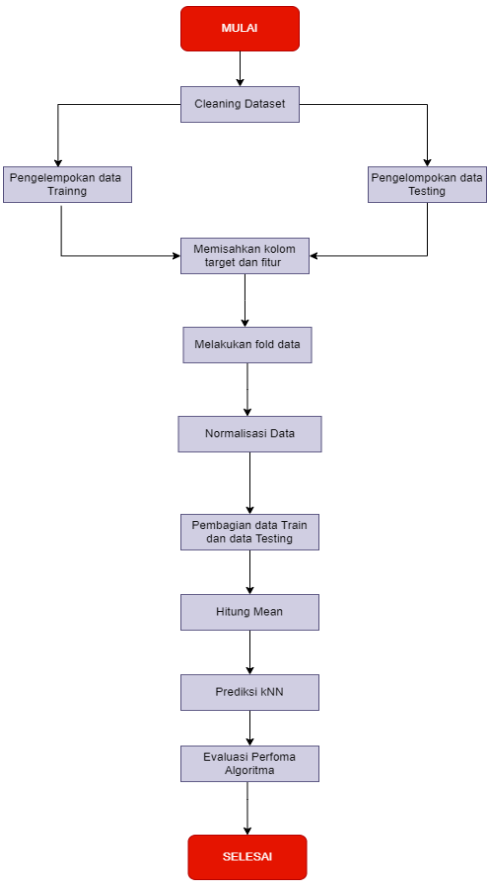
$$\frac{probabilitas\ kondisional\ x\ probabilitas\ prior}{faktor\ normalisasi}$$

5. menentukan kelas dengan probabilitas yang tinggi

Untuk pengimplementasian algoritma *Naive Bayes* dalam *dataset arrhythmia* secara garis besar adalah sebagai berikut:

1. *Load dataset Arrhythmia* ke dalam Python
2. Pisahkan fitur-fitur dan label dari dataset.
3. Lakukan *preprocessing* data seperti normalisasi jika diperlukan.
4. Bagi *dataset* menjadi data pelatihan dan data pengujian.
5. Hitung *Mean* untuk setiap kelas dalam data pelatihan
6. Hitung probabilitas untuk setiap kelas dalam data pelatihan.
7. Evaluasi performa algoritma menggunakan matrik evaluasi seperti akurasi.

flowchart implementasi algoritma *Naive Bayes* :



Hasil atau output yang diharapkan dari penggunaan algoritma *Naive Bayes* terhadap *dataset Arrhythmia* diantaranya yaitu prediksi kelas untuk setiap data uji berdasarkan probabilitas posterior tertinggi, confusion matriks yang menunjukkan sejauh mana klasifikasi *Naive Bayes* sesuai dengan kelas yang sebenarnya, matriks evaluasi seperti akurasi, presisi, recall, dan F1-score untuk mengukur performa *Naive Bayes* dalam mengklasifikasikan data.

Pemilihan *hyperparameter* yang bervariasi dari algoritma Naive Bayes terhadap dataset Arrhythmia dapat melibatkan beberapa analisis seperti pemilihan tipe distribusi probabilitas, dimana dalam *Naive Bayes* diasumsikan bahwa setiap fitur dalam dataset terdistribusi secara independen. Oleh karena itu, pemilihan tipe distribusi probabilitas yang tepat untuk setiap fitur sangat penting. Misalnya, jika fitur-fitur dalam dataset *Arrhythmia* memiliki distribusi normal, maka *Naive Bayes* dengan asumsi Gaussian dapat digunakan. Selain itu, *Naive Bayes* juga melibatkan teknik penghalusan untuk menghindari probabilitas nol atau probabilitas yang sangat kecil.

Berikut tangkapan layar implementasi algoritma *Naive Bayes* menggunakan bahasa pemrograman python:

- **Import *library* dan pembacaan *dataset***

IMPORT

```
[1] import urllib.request
    import pandas as pd
    import numpy as np
    import math
```

```
# melakukan import dataset dari URL
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/arrhythmia/arrhythmia.data'
filename = 'arrhythmia.data'

urllib.request.urlretrieve(url, filename)
```

1. Melakukan import *library* dan *dataset*

READ CSV

```
# Membaca dataset
df = pd.read_csv('arrhythmia.data', header=None)
column_names = [
    'age','gender','height','weight','qrs_duration','p-r_Interval','q-t_interval','t_interval','p_interval',
            'QRS','T','P','QRST','J','heart_rate','Q_D1','R_D1','S_D1','R\'_D1','S\'_D1','NOD_D1','EOR_R_D1','EOD_R_D1',
            'EOR_P_D1','EOD_P_D1','EOR_T_D1','EOD_T_D1','Q_D2','R_D2','S_D2','R\'_D2','S\'_D2','NOD_D2','EOR_R_D2','EOD_R_D2',
            'EOR_P_D2','EOD_P_D2','EOR_T_D2','EOD_T_D2','Q_D3','R_D3','S_D3','R\'_D3','S\'_D3','NOD_D3','EOR_R_D3','EOD_R_D3',
            'EOR_P_D3','EOD_P_D3','EOR_T_D3','EOD_T_D3','Q_AVR','R_AVR','S_AVR','R\'_AVR','S\'_AVR','NOD_AVR','EOR_R_AVR',
            'EOD_R_AVR','EOR_P_AVR','EOD_P_AVR','EOR_T_AVR','EOD_T_AVR','Q_AVL','R_AVL','S_AVL','R\'_AVL','S\'_AVL','NOD_AVL',
            'EOR_R_AVL','EOD_R_AVL','EOR_P_AVL','EOD_P_AVL','EOR_T_AVL','EOD_T_AVL','Q_AVF','R_AVF','S_AVF','R\'_AVF','S\'_AVF',
            'NOD_AVF','EOR_R_AVF','EOD_R_AVF','EOR_P_AVF','EOD_P_AVF','EOR_T_AVF','EOD_T_AVF','Q_V1','R_V1','S_V1','R\'_V1',
            'S\'_V1','NOD_V1','EOR_R_V1','EOD_R_V1','EOR_P_V1','EOD_P_V1','EOR_T_V1','EOD_T_V1','Q_V2','R_V2','S_V2','R\'_V2',
            'S\'_V2','NOD_V2','EOR_R_V2','EOD_R_V2','EOR_P_V2','EOD_P_V2','EOR_T_V2','EOD_T_V2','Q_V3','R_V3','S_V3','R\'_V3',
            'S\'_V3','NOD_V3','EOR_R_V3','EOD_R_V3','EOR_P_V3','EOD_P_V3','EOR_T_V3','EOD_T_V3','Q_V4','R_V4','S_V4','R\'_V4',
            'S\'_V4','NOD_V4','EOR_R_V4','EOD_R_V4','EOR_P_V4','EOD_P_V4','EOR_T_V4','EOD_T_V4','Q_V5','R_V5','S_V5','R\'_V5',
            'S\'_V5','NOD_V5','EOR_R_V5','EOD_R_V5','EOR_P_V5','EOD_P_V5','EOR_T_V5','EOD_T_V5','Q_V6','R_V6','S_V6','R\'_V6',
            'S\'_V6','NOD_V6','EOR_R_V6','EOD_R_V6','EOR_P_V6','EOD_P_V6','EOR_T_V6','EOD_T_V6','A_JJ_D1','A_Q_D1','A_R_D1',
            'A_S_D1','A_R\'_D1','A_S\'_D1','A_P_D1','A_T_D1','QRSA_D1','QRSTA_D1','A_JJ_D2','A_Q_D2','A_R_D2','A_S_D2',
            'A_R\'_D2','A_S\'_D2','A_P_D2','A_T_D2','QRSA_D2','QRSTA_D2','A_JJ_D3','A_Q_D3','A_R_D3','A_S_D3','A_R\'_D3',
            'A_S\'_D3','A_P_D3','A_T_D3','QRSA_D3','QRSTA_D3','A_JJ_AVR','A_Q_AVR','A_R_AVR','A_S_AVR','A_R\'_AVR','A_S\'_AVR',
            'A_P_AVR','A_T_AVR','QRSA_AVR','QRSTA_AVR','A_JJ_AVL','A_Q_AVL','A_R_AVL','A_S_AVL','A_R\'_AVL','A_S\'_AVL','A_P_AVL',
            'A_T_AVL','QRSA_AVL','QRSTA_AVL','A_JJ_AVF','A_Q_AVF','A_R_AVF','A_S_AVF','A_R\'_AVF','A_S\'_AVF','A_P_AVF','A_T_AVF',
            'QRSA_AVF','QRSTA_AVF','A_JJ_V1','A_Q_V1','A_R_V1','A_S_V1','A_R\'_V1','A_S\'_V1','A_P_V1','A_T_V1','QRSA_V1','QRSTA_V1',
            'A_JJ_V2','A_Q_V2','A_R_V2','A_S_V2','A_R\'_V2','A_S\'_V2','A_P_V2','A_T_V2','QRSA_V2','QRSTA_V2','A_JJ_V3','A_Q_V3',
            'A_R_V3','A_S_V3','A_R\'_V3','A_S\'_V3','A_P_V3','A_T_V3','QRSA_V1','QRSTA_V3','A_JJ_V4','A_Q_V4','A_R_V4','A_S_V4',
            'A_R\'_V4','A_S\'_V4','A_P_V4','A_T_V4','QRSA_V4','QRSTA_V4','A_JJ_V5','A_Q_V5','A_R_V5','A_S_V5','A_R\'_V5','A_S\'_V5',
            'A_P_V5','A_T_V5','QRSA_V5','QRSTA_V5','A_JJ_V6','A_Q_V6','A_R_V6','A_S_V6','A_R\'_V6','A_S\'_V6','A_P_V6','A_T_V6',
            'QRSA_V6','QRSTA_V6','Class'
]
```

2. Membaca *dataset* dari arrhytmia.data

- **Pemeriksaan Data**

```
[7]  def loss_data(df):
       null = df.isnull().sum()
       for i in range(len(df.columns)):
         if null[i] > 0:
           print(f"{df.columns[i]}: {null[i]} ({((null[i]/len(df))*100}%)")
       total_cells = np.product(df.shape)
       total_missing = null.sum()
       print(f"\nTotal data yang hilang: {total_missing} ({(total_missing/total_cells) * 100}%)\n")
```

1. Fungsi loss_data digunakan untuk mencari data-data yang hilang pada *dataset*

```
def data_clean(df):
  for i in ['heart_rate', 'T', 'P', 'QRST', 'J']:
    df[i] = df[i].replace('?','-1').astype(int)
    mean = df[i].mean(axis=0)
    df[i] = df[i].replace(-1,mean)
  return df
```

2. Fungsi data_clean digunakan untuk membersihkan data dalam suatu DataFrame, Dengan menggunakan fungsi data_clean, kita dapat membersihkan data pada kolom-kolom yang disebutkan, mengganti nilai yang tidak valid atau hilang dengan nilai rata-rata dari kolom tersebut.

```
# melakukan pemeriksaan ulang data yang tidak lengkap
df = data_clean(df)
loss_data(df)
```

```
Total data yang hilang: 0 (0.0%)
```

3. Menggunakan fungsi data_clean untuk membersihkan dataset arrhythmia dan menggunakan fungsi loss_data untuk melakukan pengecekan lagi. Dari output tersebut didapatkan bahwa sudah tidak ada data yang hilang pada *dataset.*

```
[10] df.info() # Menampilkan informasi tentang dataset, seperti jumlah baris, kolom, tipe data, dll.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Columns: 280 entries, age to Class
dtypes: float64(125), int64(155)
memory usage: 988.9 KB
```

4. Menampilkan informasi dari dataset. Dalam dataset arrhythmia terdapat 452 baris dan 280 kolom serta 2 tipe data yaitu float dan int64.

```
df.describe() # Menampilkan ringkasan statistik deskriptif untuk setiap kolom numerik
```

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	...
mean	46.471239	0.550885	166.188053	68.170354	88.920354	155.152655	367.207965	169.949115	90.004425	33.676991	...
std	16.466631	0.497955	37.170340	16.590803	15.364394	44.842283	33.385421	35.633072	25.826643	45.431434	...
min	0.000000	0.000000	105.000000	6.000000	55.000000	0.000000	232.000000	108.000000	0.000000	-172.000000	...
25%	36.000000	0.000000	160.000000	59.000000	80.000000	142.000000	350.000000	148.000000	79.000000	3.750000	...
50%	47.000000	1.000000	164.000000	68.000000	86.000000	157.000000	367.000000	162.000000	91.000000	40.000000	...
75%	58.000000	1.000000	170.000000	79.000000	94.000000	175.000000	384.000000	179.000000	102.000000	66.000000	...
max	83.000000	1.000000	780.000000	176.000000	188.000000	524.000000	509.000000	381.000000	205.000000	169.000000	...

8 rows × 280 columns

5. Menampilkan statistik deskriptif dari dataset arrhythmia. dimana didalamnya terdapat count, mean, std, min, max, dan kuartil.

- **Normalisasi Data**

```
NORMALISASI DATA

[13] # melakukan normalisasi
     def normalization(allData, ColumnTarget):
       for column in ColumnTarget:
         allData[column] = (allData[column] - allData[column].min()) / (allData[column].max() - allData[column].min())

       return allData

     testStartID = df.index.stop
     allData = pd.concat([df])
     allData = normalization(allData, ['age','height','weight'])
     normNaive= allData.iloc[:testStartID].drop('gender', axis=1)
```

1. Fungsi normalization memiliki makna untuk melakukan normalisasi pada kolom-kolom tertentu dari suatu dataset. Program di atas menggunakan fungsi normalization untuk melakukan normalisasi pada kolom-kolom tertentu dalam dataset allData. Kolom-kolom yang akan dinormalisasi adalah 'age', 'height', dan 'weight'. Kemudian, dataset yang sudah dinormalisasi (tanpa kolom 'gender') disimpan dalam variabel normNaive. Dengan menggunakan fungsi dan program di atas, dataset dapat dinormalisasi dengan memilih kolom-kolom yang ingin dinormalisasi.

```
normNaive
```

	age	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	T	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_V6
0	0.903614	0.125926	0.435294	91	193	371	174	121	-16	13.0	...	0.0	9.0	-0.9	0.0	0.0	0.9	2.9	23.3	49.4
1	0.674699	0.088889	0.341176	81	174	401	149	39	25	37.0	...	0.0	8.5	0.0	0.0	0.0	0.2	2.1	20.4	38.8
2	0.650602	0.099259	0.523529	138	163	386	185	102	96	34.0	...	0.0	9.5	-2.4	0.0	0.0	0.3	3.4	12.3	49.0
3	0.662651	0.103704	0.517647	100	202	380	179	143	28	11.0	...	0.0	12.2	-2.2	0.0	0.0	0.4	2.6	34.6	61.8
4	0.903614	0.125926	0.435294	88	181	360	177	103	-16	13.0	...	0.0	13.1	-3.6	0.0	0.0	-0.1	3.9	25.4	62.8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
447	0.638554	0.081481	0.376471	80	199	382	154	117	-37	4.0	...	0.0	4.3	-5.0	0.0	0.0	0.7	0.6	-4.4	-0.8
448	0.445783	0.125926	0.464706	100	137	361	201	73	86	66.0	...	0.0	15.6	-1.6	0.0	0.0	0.4	2.4	38.0	62.4
449	0.433735	0.090370	0.364706	108	176	365	194	116	-85	-19.0	...	0.0	16.3	-28.6	0.0	0.0	1.5	1.0	-44.2	-33.7
450	0.385542	0.074074	0.288235	93	106	386	218	63	54	29.0	...	-0.4	12.0	-0.7	0.0	0.0	0.5	2.4	25.0	46.6
451	0.939759	0.081481	0.376471	79	127	364	138	78	28	79.0	...	0.0	10.4	-1.8	0.0	0.0	0.5	1.6	21.3	32.8

452 rows × 279 columns

2. Gambar diatas adalah hasil dari normalisasi data yang disimpan pada variabel normNaive.

- **Menentukan Fold**

```
[16] fold1 = (df.iloc[0:150].reset_index(drop=True), df.iloc[300:452].reset_index(drop=True))
     fold2 = (df.iloc[150:300].reset_index(drop=True), pd.concat([df.iloc[0:150], df.iloc[300:452]]).reset_index(drop=True))
     fold3 = (df.iloc[300:452].reset_index(drop=True), df.iloc[0:150].reset_index(drop=True))
```

1. Program di atas bertujuan untuk melakukan fold atau pembagian data menjadi beberapa subset yang saling tumpang tindih untuk keperluan validasi

- **Standarisasi Data**

STANDARISASI DATA

```
[17] # melakukan standarisasi
     def standarization(allData, ColumnTarget):
       for column in ColumnTarget:
         allData[column] = (allData[column] - allData[column].mean()) / (allData[column].std())

       return allData
```

```
[18] testStartID = df.index.stop
     allData = pd.concat([df])
     allData = standarization(allData, ['age','height','weight'])
     stdNaive= allData.iloc[:testStartID].drop('gender', axis=1)
```

1. Fungsi Standarization memiliki makna untuk melakukan standarisasi pada kolom-kolom tertentu dari suatu dataset. Program di atas menggunakan fungsi standarization untuk melakukan standarisasi pada kolom-kolom tertentu dalam dataset allData. Kolom-kolom yang akan di standarisasi adalah 'age', 'height', dan 'weight'. Kemudian, dataset yang sudah di standarisasi (tanpa kolom 'gender') disimpan dalam variabel stdNaive. Dengan menggunakan fungsi dan program di atas, dataset dapat di standarisasi dengan memilih kolom-kolom yang ingin di standarisasi.

stdNaive

	age	height	weight	qrs_duration	P-r_Interval	q-t_interval	t_interval	p_interval	QRS	T	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_
0	1.732520	0.640617	0.713024	91	193	371	174	121	-16	13.0	...	0.0	9.0	-0.9	0.0	0.0	0.9	2.9	23.3	4
1	0.578671	-0.031962	-0.251365	81	174	401	149	39	25	37.0	...	0.0	8.5	0.0	0.0	0.0	0.2	2.1	20.4	3
2	0.457213	0.156360	1.617140	138	163	386	185	102	96	34.0	...	0.0	9.5	-2.4	0.0	0.0	0.3	3.4	12.3	4
3	0.517942	0.237069	1.556865	100	202	380	179	143	28	11.0	...	0.0	12.2	-2.2	0.0	0.0	0.4	2.6	34.6	6
4	1.732520	0.640617	0.713024	88	181	360	177	103	-16	13.0	...	0.0	13.1	-3.6	0.0	0.0	-0.1	3.9	25.4	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
447	0.396484	-0.166478	0.110281	80	199	382	154	117	-37	4.0	...	0.0	4.3	-5.0	0.0	0.0	0.7	0.6	-4.4	
448	-0.575178	0.640617	1.014396	100	137	361	201	73	86	66.0	...	0.0	15.6	-1.6	0.0	0.0	0.4	2.4	38.0	6
449	-0.635907	-0.005059	-0.010268	108	176	365	194	116	-85	-19.0	...	0.0	16.3	-28.6	0.0	0.0	1.5	1.0	-44.2	-3
450	-0.878822	-0.300994	-0.793835	93	106	386	218	63	54	29.0	...	-0.4	12.0	-0.7	0.0	0.0	0.5	2.4	25.0	4
451	1.914706	-0.166478	0.110281	79	127	364	138	78	28	79.0	...	0.0	10.4	-1.8	0.0	0.0	0.5	1.6	21.3	3

452 rows × 279 columns

2. Gambar diatas adalah hasil dari standarisasi data yang disimpan pada variabel stdNaive.

- **Split Truth**

SPLIT TRUTH

```
[20] # melakukan split Truth
     def splitTruth(df, columnTarget):
       truthData = []
       for truth in df[columnTarget].unique() :
         truthData.append(df.where(df[columnTarget] == truth).dropna())
       return truthData
     yesData, noData = splitTruth(df, columnTarget='gender')
```

1. Fungsi splithTruth memiliki makna untuk membagi dataset menjadi dua subset berdasarkan nilai unik dari kolom target yang ditentukan. Dimana Setelah selesai membagi dataset berdasarkan nilai unik kolom target, fungsi mengembalikan list truthData yang berisi subset dataset dengan masing-masing subset berisi baris-baris yang memiliki nilai target yang sama. Dengan menggunakan fungsi dan program di atas, dataset dapat dibagi berdasarkan nilai unik kolom target yang ditentukan. Hal ini berguna untuk memisahkan data berdasarkan kategori atau kelas tertentu dalam proses analisis atau pemodelan data.

- **Mencari rata-rata**

MENCARI RATA-RATA

```
[21] # mencari mean
     def findMean(yesData, noData, columnTarget):
       yesMean = dict()
       noMean = dict()
       for column in columnTarget :
         yesMean[column] = yesData[column].mean()
         noMean[column] = noData[column].mean()
       return yesMean, noMean
     yesMean, noMean = findMean(yesData, noData, columnTarget = ['Class'])
```

```
[22] print(f"Mean Result\n1 : {yesMean}\n0 : {noMean}")
```

```
Mean Result
1 : {'Class': 4.748768472906404}
0 : {'Class': 3.1726907630522088}
```

1. Fungsi findMean memiliki makna untuk mencari nilai rata-rata (mean) dari kolom target yang ditentukan untuk dua subset dataset yang diberikan. Dimana Setelah selesai mencari nilai rata-rata untuk semua kolom target yang ditentukan, fungsi mengembalikan dua dictionary yesMean dan noMean yang berisi nilai rata-rata dari masing-masing subset dataset. Program di atas menggunakan fungsi findMean untuk mencari nilai rata-rata dari kolom target 'Class' pada dua subset dataset yesData dan noData. Nilai rata-rata dari 'Class' pada subset yesData disimpan dalam variabel yesMean, sedangkan nilai rata-rata dari 'Class' pada subset noData disimpan dalam variabel noMean. Outputnya menunjukkan nilai rata-rata dari kolom target 'Class' pada subset 'yesData' adalah 4.748768472906404, sedangkan nilai rata-rata pada subset 'noData' adalah 3.1726907630522088.

- **Mencari Standar Deviasi**

MENCARI STD

```
[23] # Mencari std
     def findStd(yesData, noData, columnTarget):
       yesStd = dict()
       noStd = dict()
       for column in columnTarget :
         yesStd[column] = yesData[column].std()
         noStd[column] = noData[column].std()

       return yesStd, noStd

     yesStd, noStd = findStd(yesData, noData, columnTarget = ['Class'])
```

1. Program di atas menggunakan fungsi findStd untuk mencari nilai standar deviasi (std) dari kolom target yang ditentukan untuk masing-masing subset data 'yesData' dan 'noData'. Dalam program diatas, fungsi findStd digunakan untuk mencari nilai standar deviasi dari kolom target 'Class' pada subset data 'yesData' dan 'noData'. Hasil nilai standar deviasi untuk subset 'yesData' disimpan dalam variabel yesStd, sedangkan hasil nilai standar deviasi untuk subset 'noData' disimpan dalam variabel noStd.

- **Probabilitas**

PROBABILITAS

```
[24] # melakukan kalkulasi probabilitas
     def calc_probality(mean, std, x):
       exponent = math.exp(-(x-mean)**2 / (2*(std**2)))
       return (1 / (math.sqrt(2*math.pi)*std)) * exponent
```

1. Fungsi calc_probability memiliki makna untuk menghitung probabilitas suatu nilai x berdasarkan distribusi normal dengan mean mean dan standar deviasi std. Fungsi ini mengimplementasikan rumus untuk menghitung probabilitas dari fungsi kepadatan probabilitas (probability density function) dari distribusi normal.

- **Confusion Matriks**

CONFUSION MATRIKS

```
# melakukan confusion matriks
def confussionMatrix(result):

    TP = 0
    FP = 0
    TN = 0
    FN = 0
    x = True

    for i in result:
        if(i['Ground Truth'] == '?'):
            x = False
            break
        elif((i['Prediction Result'] == 1)and(i['Prediction Result'] == i['Ground Truth'])):
            TP += 1
        elif((i['Prediction Result'] == 0)and(i['Prediction Result'] == i['Ground Truth'])):
            TN += 1
        elif((i['Prediction Result'] == 1)and(i['Prediction Result'] != i['Ground Truth'])):
            FP += 1
        elif((i['Prediction Result'] == 0)and(i['Prediction Result'] != i['Ground Truth'])):
            FN += 1
    if(x):
        print(f"\nTP : {TP} FN : {FP}\nTN : {TN} FN : {FN}")
        print(f"Accuracy : {((TP+TN)/(TP+TN+FP+FN))*100}%")
        print(f"Precission : {((TP)/(TP+FP))*100}%")
        print(f"Recall : {((TP)/(TP+FN))*100}%")
    else:
        print("\nCannot process the confussion matrix with unknown Ground Truth!")
```

1. Fungsi confusionMatrix memiliki makna untuk menghitung dan menampilkan confusion matrix berdasarkan hasil prediksi yang diberikan. Confusion matrix

adalah alat evaluasi kinerja yang umum digunakan dalam pemodelan klasifikasi untuk memahami sejauh mana model berhasil memprediksi kelas target. Dengan menggunakan fungsi confusionMatrix, kita dapat menghitung dan menampilkan matriks kebingungan berdasarkan hasil prediksi. Ini membantu dalam mengevaluasi kinerja model klasifikasi dengan membandingkan prediksi dengan *Ground Truth* untuk mengukur akurasi, presisi, dan recall dari model tersebut.

- **Prediksi**

PREDIKSI

```
# melakukan prediksi
def prediction(yesMean, yesStd, noMean, noStd, target, columnTarget, truthColumn):
    result = []
    for i in range(len(target)):
      yesResult = 1
      noResult = 1
      for column in columnTarget:
        yesResult *= calc_probality(yesMean[column], yesStd[column], target[column].iloc[i])
        noResult *= calc_probality(noMean[column], noStd[column], target[column].iloc[i])

      result.append({'CLASS' : target['Class'].iloc[i], 'Yes Probability' : "{}".format(yesResult), 'No Probability' : "{}".format(noResult),
      'Prediction Result' : int(yesResult > noResult),
      'Ground Truth' : target[truthColumn].iloc[i]})

    return result
```

```
[27] result = []
     target = df

     result = prediction(yesMean, yesStd, noMean, noStd, target, columnTarget = ['Class'], truthColumn='gender')

     for p in result:
       print(p)
```

```
{'CLASS': 8, 'Yes Probability': '0.06649253613061346', 'No Probability': '0.048058024650189844', 'Prediction Result': 1, 'Ground Truth': 0}
{'CLASS': 6, 'Yes Probability': '0.0812188136216501', 'No Probability': '0.07784848938913368', 'Prediction Result': 1, 'Ground Truth': 1}
{'CLASS': 10, 'Yes Probability': '0.045572348380052564', 'No Probability': '0.023057566479196264', 'Prediction Result': 1, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 0}
{'CLASS': 7, 'Yes Probability': '0.07513862404239731', 'No Probability': '0.06312357313980496', 'Prediction Result': 1, 'Ground Truth': 0}
{'CLASS': 14, 'Yes Probability': '0.01256016060798973', 'No Probability': '0.002491747660695327', 'Prediction Result': 1, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 0}
{'CLASS': 10, 'Yes Probability': '0.045572348380052564', 'No Probability': '0.023057566479196264', 'Prediction Result': 1, 'Ground Truth': 1}
{'CLASS': 3, 'Yes Probability': '0.07856972501036705', 'No Probability': '0.10005153985912607', 'Prediction Result': 0, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 10, 'Yes Probability': '0.045572348380052564', 'No Probability': '0.023057566479196264', 'Prediction Result': 1, 'Ground Truth': 1}
{'CLASS': 6, 'Yes Probability': '0.0812188136216501', 'No Probability': '0.07784848938913368', 'Prediction Result': 1, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 10, 'Yes Probability': '0.045572348380052564', 'No Probability': '0.023057566479196264', 'Prediction Result': 1, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 0}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 1, 'Yes Probability': '0.06154171746165373', 'No Probability': '0.08630533044815372', 'Prediction Result': 0, 'Ground Truth': 1}
{'CLASS': 16, 'Yes Probability': '0.005050803423189807', 'No Probability': '0.000561238034897 4867', 'Prediction Result': 1, 'Ground Truth': 0}
```

1. Fungsi program di atas adalah sebuah fungsi prediksi yang menggunakan naive Bayes untuk memprediksi kelas target berdasarkan fitur-fitur yang diberikan. Fungsi ini menerima parameter berupa statistik probabilitas dari kelas "Yes" dan "No" (yesMean, yesStd, noMean, noStd), data target yang akan diprediksi (target), kolom target (columnTarget), dan kolom yang berisi kebenaran (truthColumn). Hasil prediksi untuk setiap baris data kemudian

disimpan dalam bentuk dictionary yang berisi informasi tentang kelas target, probabilitas "Yes" dan "No", hasil prediksi (1 jika probabilitas "Yes" lebih besar dari "No", 0 jika sebaliknya), dan kebenaran yang terdapat dalam kolom truthColumn. Output yang ditampilkan adalah hasil prediksi untuk setiap baris data target dalam bentuk dictionary. Setiap dictionary menunjukkan kelas target, probabilitas "Yes" dan "No", hasil prediksi, dan kebenaran yang sesuai dengan kolom truthColumn.

- **Folding**

FOLDING

```
[28] # melakukan folding / membagi dataset menjadi beberapa subset untuk melatih dan menguji model secara berulang
     def folding(dataset, trainingPercentage, location, shuffle:bool):
         lengthTraining = int(len(dataset)*trainingPercentage/100)
         # randomize the the data position
         if(shuffle):
             dataset = dataset.sample(frac=1).reset_index(drop=True)
         naive = []
         validation = []
         if(location == 'left'):
             naive, validation = dataset.iloc[:lengthTraining].reset_index(drop=True), dataset.iloc[lengthTraining:].reset_index(drop=True)
         elif(location == 'right'):
             validation,naive = dataset.iloc[:abs(lengthTraining-len(dataset))].reset_index(drop=True), dataset.iloc[abs(lengthTraining-len(dataset)):].reset_index(drop=True)
         elif(location == 'middle'):
             naive = dataset.iloc[int(abs(lengthTraining-len(dataset))/2):len(dataset)-int(abs(lengthTraining-len(dataset))/2)]
             validation = pd.concat([dataset.iloc[:int(abs(lengthTraining-len(dataset))/2)],dataset.iloc[len(dataset)-int(abs(lengthTraining-len(dataset))/2):]])
         return naive,validation

[29] dataNaive, validationData = folding(df.copy(), 70, 'left', shuffle=True)
```

1. Fungsi program di atas adalah fungsi untuk membagi *dataset* menjadi subset untuk pelatihan dan pengujian model menggunakan pendekatan k-fold cross-validation. Fungsi ini menerima parameter dataset (data yang akan dibagi), trainingPercentage (persentase data yang akan digunakan untuk pelatihan), location (lokasi subset yang akan digunakan untuk pelatihan), dan shuffle (flag untuk mengacak urutan data sebelum pembagian). Hasilnya, fungsi ini akan mengembalikan subset pelatihan (naive) dan subset validasi (validation) dalam bentuk DataFrame. Pemanggilan fungsi di atas akan menghasilkan dua subset: dataNaive (subset pelatihan) dan validationData (subset validasi). Subset pelatihan akan terdiri dari 70% data awal yang diacak, sedangkan subset validasi akan terdiri dari 30% data yang tersisa setelah pembagian subset pelatihan.

dataNaive

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_V6	Cla
0	74	0	170	67	84	175	444	182	42	-27	...	0.0	7.6	0.0	0.0	0.0	0.5	1.2	19.7	29.5	
1	54	1	160	63	82	158	410	141	87	25	...	0.0	7.3	-2.1	0.0	0.0	0.6	-0.3	11.0	8.3	
2	8	0	120	28	118	126	303	164	80	120	...	-0.6	12.5	-3.6	0.0	0.0	0.5	2.3	9.2	32.2	
3	48	0	162	84	80	154	354	171	85	39	...	0.0	7.8	-1.2	0.0	0.0	0.8	1.1	14.0	23.6	
4	59	1	155	72	83	209	344	297	105	69	...	-0.5	8.8	-1.8	0.0	0.0	0.4	-1.2	14.0	0.4	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
311	57	1	166	72	82	181	399	158	79	-12	...	0.0	7.7	-0.9	0.0	0.0	0.5	1.8	25.2	38.5	
312	64	1	155	88	82	194	342	138	126	-4	...	0.0	4.9	-0.9	0.5	0.0	0.3	0.4	9.6	12.8	
313	69	1	154	59	74	159	380	154	96	15	...	0.0	10.3	0.0	0.0	0.0	0.7	1.8	28.8	41.0	
314	34	1	167	60	63	164	396	139	84	49	...	0.0	8.7	0.0	0.0	0.0	0.4	1.5	22.6	33.4	
315	25	1	162	78	77	141	378	206	84	56	...	-0.4	9.7	-0.7	0.0	0.0	0.5	2.6	22.2	49.2	

316 rows × 280 columns

2. Gambar diatas adalah hasil yang disimpan pada variabel dataNaive

validationData

	age	gender	height	weight	qrs_duration	p-r_Interval	q-t_interval	t_interval	p_interval	QRS	...	A_Q_V6	A_R_V6	A_S_V6	A_R'_V6	A_S'_V6	A_P_V6	A_T_V6	QRSA_V6	QRSTA_V6	Clas
0	39	1	160	45	75	163	418	155	90	12	...	-0.4	5.3	-1.3	0.0	0.0	0.6	1.9	8.2	22.2	
1	75	1	156	55	73	159	350	138	99	-18	...	0.0	2.9	-1.6	0.0	0.0	0.7	1.5	3.0	12.0	
2	43	1	157	71	80	162	383	141	84	53	...	0.0	10.8	-1.5	0.0	0.0	0.4	1.9	23.5	36.4	
3	52	1	155	104	84	188	450	193	89	22	...	0.0	7.6	-3.7	0.0	0.0	0.4	0.7	8.6	15.7	
4	48	1	170	74	74	160	381	142	87	40	...	0.0	8.4	-1.1	0.0	0.0	0.5	0.9	16.5	21.1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
131	54	1	172	58	78	155	382	163	81	-24	...	0.0	6.3	-2.1	0.0	0.0	0.8	0.5	8.8	12.1	1
132	45	0	177	72	94	164	431	158	81	116	...	-1.9	0.0	0.0	0.0	0.0	0.6	-2.2	-8.3	-38.6	
133	19	1	156	47	80	137	342	243	92	-7	...	-0.4	9.4	-1.4	0.0	0.0	1.3	4.3	16.1	71.1	
134	53	1	160	60	86	133	338	159	82	74	...	-0.4	11.6	-5.3	0.0	0.0	1.0	2.1	0.5	17.3	
135	62	1	165	70	72	169	328	135	85	-13	...	0.0	4.2	-1.2	0.0	0.0	0.5	0.7	6.0	10.7	

136 rows × 280 columns

3. Gambar diatas adalah hasil yang disimpan pada variabel validationData.

- **Evaluasi**

Evaluasi Prediksi

```
[32] nodatanaive, yesdatanaive = splitTruth(df, columnTarget='gender')
```

```
[33] # Split yes and no
     yesDatanaive, noDatanaive = splitTruth(df, columnTarget='gender')
     # Find mean
     yesMeanNaive, noMeanNaive = findMean(yesDatanaive, noDatanaive, columnTarget=['Class'])
     # Find standard deviation
     yesStdnaive, noStdnaive = findStd(yesDatanaive, noDatanaive, ['Class'])

     result = []

     target = validationData # your ground truth data

     result = prediction(yesMean, yesStd, noMean, noStd, target, columnTarget = ['Class'], truthColumn='gender')

     for p in result:
       print(p)

     confussionMatrix(result)
```

1. Program di atas memiliki beberapa langkah yang melibatkan pemrosesan data dan evaluasi prediksi. Dimana pada tahap ini terdapat pemanggilan fungsi splitTruth, findMean, findStd, prediction, dan confussionMatriks. Output yang ditampilkan adalah hasil prediksi untuk setiap baris data target dalam bentuk dictionary. Setiap dictionary menunjukkan kelas target, probabilitas "Yes" dan "No", hasil prediksi (1 jika probabilitas "Yes" lebih besar dari "No", 0 jika sebaliknya), dan kebenaran yang sesuai dengan kolom 'gender'. Selain itu, output juga mencakup hasil dari confusion matrix dengan informasi True Positive (TP), False Negative (FN), True Negative (TN), False Positive (FP), akurasi (accuracy), presisi (precision), dan recall (sensitivity).

Link Algoritma Naive Bayes:
https://github.com/muammarrz/tubes-ai/blob/3f21f217ff0c6e7291915875ab4852ac6e3b9949/arrhythmia_Naive_Bayes.ipynb

BAB III
EVALUASI HASIL DAN DISKUSI

a. Rumus Performansi

Berdasarkan hasil yang ditemukan, rumus pengukuran performansi yang kami dapat adalah sebagai berikut :

```
if(x):
    print(f"\nTP : {TP} FN : {FP}\nTN : {TN} FN : {FN}")
    print(f"Accuracy : {((TP+TN)/(TP+TN+FP+FN))*100}%")
    print(f"Precission : {((TP)/(TP+FP))*100}%")
    print(f"Recall : {((TP)/(TP+FN))*100}%")
```

1. Akurasi : (jumlah prediksi benar) / (jumlah total prediksi) * 100%
2. Presisi : (Jumlah True Positive) / (Jumlah True Positive + Jumlah False Positive) * 100%
3. Recall : (Jumlah True Positive) / (Jumlah True Positive + Jumlah False Negative) * 100%
4. F1 Score (penggabungan presisi dan recall) : 2 * ((Presisi * Recall) / (Presisi + Recall))

b. Hasil yang Ditemukan

Algoritma kNN

Nilai Euclidean = 2.28570280218962	**PERHITUNGAN JARAK EUCLIDEAN**
	`# Perhitungan jarak Euclidean`
	`def euclidean(x1,x2):`
	`  return np.sqrt(np.sum((x1 - x2)**2))`
	`euclidean(x.iloc[0], x.iloc[1])`
	`2.28570280218962`

Nilai Akurasi = 0.5666666666666668 = 56.66666666666668%	`# Evaluasi kinerja model menggunakan akurasi`
	`k = 10`
	`accs = []`
	`folds = [fold1, fold2, fold1]`
	`for i in range(len(folds)):`
	`  accs.append(evaluate(folds[i],k))`
	`print(f'Menggunakan k : {k}, dengan rata-rata akurasi : {sum(accs)/3}')`
	`Menggunakan k : 10, dengan rata-rata akurasi : 0.5666666666666668`

Nilai Euclidean pada algoritma kNN berfungsi untuk mengukur jarak atau kesamaan antara dua data dalam ruang fitur. sedangkan nilai akurasi pada algoritma kNN digunakan untuk mengukur sejauh mana algoritma ini dapat melakukan klasifikasi dengan benar.

Algoritma Naive Bayes

Nilai Akurasi : 39.705882352941174%	TP : 17 FN : 20 TN : 37 FN : 62 Accuracy : 39.705882352941174% Precission : 45.94594594594595% Recall : 21.518987341772153%
Nilai Presisi : 45.94594594594595%	
Nilai Recall : 21.518987341772153%	

Nilai Akurasi pada algoritma Naive Bayes digunakan untuk mengukur ketepatan akurasi algoritma ini berjalan. Nilai Presisi digunakan untuk mengukur sejauh mana algoritma ini dapat mengklasifikasikan dengan benar instansi yang sebenarnya positif. Sedangkan nilai Recall berfungsi untuk mengukur seberapa banyak prediksi positif yang benar dari semua data yang sebenarnya positif.

c. Analisis

Algoritma kNN

- Nilai Euclidean merupakan jarak Euclidean antara data yang diuji dengan data latihan terdekat. Semakin kecil nilai Euclidean, semakin mirip data yang diuji dengan data latihan terdekat.
- Dalam konteks kNN, nilai Euclidean digunakan untuk menentukan tetangga terdekat dari suatu data. Nilai Euclidean yang diperoleh, yaitu 2.28570280218962, menunjukkan adanya kemiripan antara data yang diuji dengan data latihan terdekat.
- Nilai akurasi sebesar 56.66666666666668% menunjukkan bahwa algoritma kNN dapat melakukan klasifikasi dengan tingkat keakuratan sebesar 56.66%. Akurasi ini menggambarkan seberapa baik algoritma dapat memprediksi kelas yang benar.

Algoritma Naive Bayes

- Nilai akurasi sebesar 39.705882352941174% menunjukkan bahwa algoritma Naive Bayes memiliki tingkat keakuratan sebesar 40.44% dalam melakukan klasifikasi pada dataset tersebut.
- Nilai presisi sebesar 45.94594594594595% mengindikasikan seberapa baik algoritma dapat mengidentifikasi kelas positif dengan benar dari keseluruhan prediksi yang dilakukan.
- Nilai recall sebesar 21.518987341772153% menggambarkan kemampuan algoritma dalam mengidentifikasi dan mendeteksi data yang benar dari kelas positif.

d. Perbandingan

Berdasarkan nilai akurasi yang diperoleh, algoritma kNN memiliki akurasi yang lebih baik dibandingkan dengan *Naive Bayes*. Algoritma kNN mencapai akurasi sebesar 56.66%, sedangkan *Naive Bayes* mencapai akurasi sebesar 39.70%. Oleh karena itu, jika tujuan utama adalah mencapai akurasi yang lebih tinggi, algoritma kNN dapat dianggap lebih baik dalam konteks ini.

Berdasarkan Confusion Matrix pada algoritma Naive Bayes terdapat Confusion Matrix yaitu

```
TP : 17 FN : 20
TN : 37 FN : 62
Accuracy : 39.705882352941174%
Precission : 45.94594594594595%
Recall : 21.518987341772153%
```

tetapi karena pada algoritma kNN tidak terdapat Confusion Matrix, tidak perlu adanya perbandingan.

e. Lampiran

Untuk melihat semua hasil output yang lebih jelas dan lengkap, bisa diakses pada link GitHub kami :

https://github.com/muammarrz/tubes-ai

Link Slide Presentasi:

https://www.canva.com/design/DAFkx_QmUew/9iCfJYOxzeykqD8qRN_p0Q/edit?utm_content=DAFkx_QmUew&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

BAB IV
KESIMPULAN DAN SARAN

A. Kesimpulan

Dalam penelitian yang kami lakukan berhasil mencapai tujuan untuk mengidentifikasi hubungan antar atribut sesuai kondisi pasien. Penelitian ini menunjukan hasil bahwa kNN *(k-Nearest Neighbors)* memiliki akurasi yang lebih tinggi daripada *Naive Bayes,* jadi algoritma yang lebih baik dalam penelitian kami untuk *Arrhythmia Dataset* adalah algoritma kNN *(k-Nearest Neighbors)*. Kedua algoritma tersebut pasti memiliki kelebihan dan kekurangan dalam pengujian suatu penelitian.

Algoritma kNN *(k-Nearest Neighbors)* dan algoritma *Naive Bayes* memiliki fungsi dalam klasifikasi data, kinerja antara dua algoritma tersebut dapat berbeda-beda sesuai dengan kondisi algoritma dan data. kami harus mempertimbangkan karakteristik dataset terlebih dahulu dalam pemilihan algoritma. pada dataset Arrhtythmia algoritma kNN *(k-Nearest Neighbors)* dan algoritma *Naive Bayes* menjadi pilihan yang baik dalam melakukan prediksi kelas dengan akurasi yang memadai.

B. Saran

Untuk penelitian selanjutnya ada beberapa hal yang perlu diperhatikan yang berkaitan dengan masalah dalam dataset dimana akan mempengaruhi kinerja algoritma, memberikan penanganan yang lebih efisien dalam permasalahan dalam dataset untuk dapat meningkatkan kinerja algoritma, dapat melakukan eksperimen pada berbagai dataset dengan karakteristik yang berbeda untuk mendapatkan pemahaman dan solusi dari setiap masalah dalam dataset.

BAB V
DAFTAR PUSTAKA

[1] (2021) java T point - K-Nearest Neighbor (KNN) Algorithm for Machine Learning Toward Data. Available: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

[2] (2018) Science - Machine Learning Basics with the K-Nearest Neighbors Algorithm. Available : https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[3] (2022) Analytics Vidhya - A Quick Introduction to K – Nearest Neighbor (KNN) . Available : https://www.analyticsvidhya.com/blog/2022/01/introduction-to-knn-algorithms/

[4] (2022) Classification Using Python kdnuggets– Naïve Bayes Algorithm: Everything you need to know. Available : https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html

[5] (2021) dataversity – What Is Naïve Bayes Classification and How Is It Used for Enterprise Analysis? . Available : https://www.dataversity.net/what-is-naive-bayes-classification-and-how-is-it-used-for-enterprise-analysis/

[6] (2018) towardsdatascience – NB Classifier. available : https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

[7] (2023) geeksforgeeks – NB Classifiers. Available : https://www.geeksforgeeks.org/naive-bayes-classifiers/

[8] Y.V. Saragih , A.W. Widodo , M.A. Rahman, “Pemilihan Fitur Berbasis Wavelet untuk Klasifikasi Denyut Jantung dari Rekaman Elektrokardiogram”, Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 3.4,p.3140-3147