# Tugas Kecil 2 IF2211 Strategi Algoritma

# Semester II tahun 2021/2022

# Pencarian Convex Hull Menggunakan Algoritma Divide and Conquer


Dipersiapkan oleh:

Rozan Fadhil Al Hafidz

13520039

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

# Daftar Isi

# 1 Algoritma Divide and Conquer yang Digunakan

Algoritma divide and conqueror adalah algoritma yang memiliki tiga tahap, yaitu

1. Divide
   Tahap ini adalah membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil

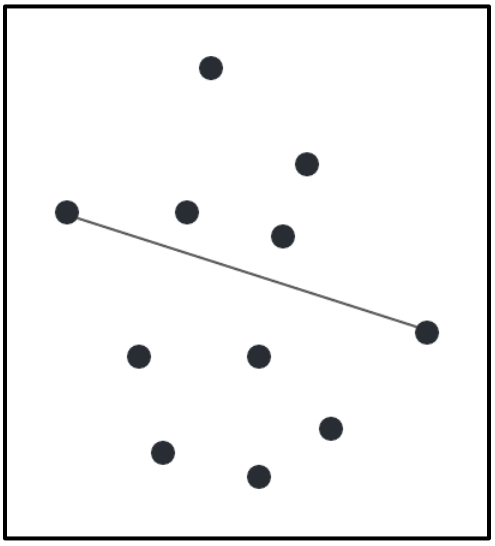
2. Conquer (solve)
   Tahap ini adalah menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran lebih kecil atau secara rekursif jika masih berukuran besar).

3. Combine
   Menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula

Berikut ini adalah penjelasan tahapan-tahapan algoritma divide and conquer yang digunakan pada program yang saya buat

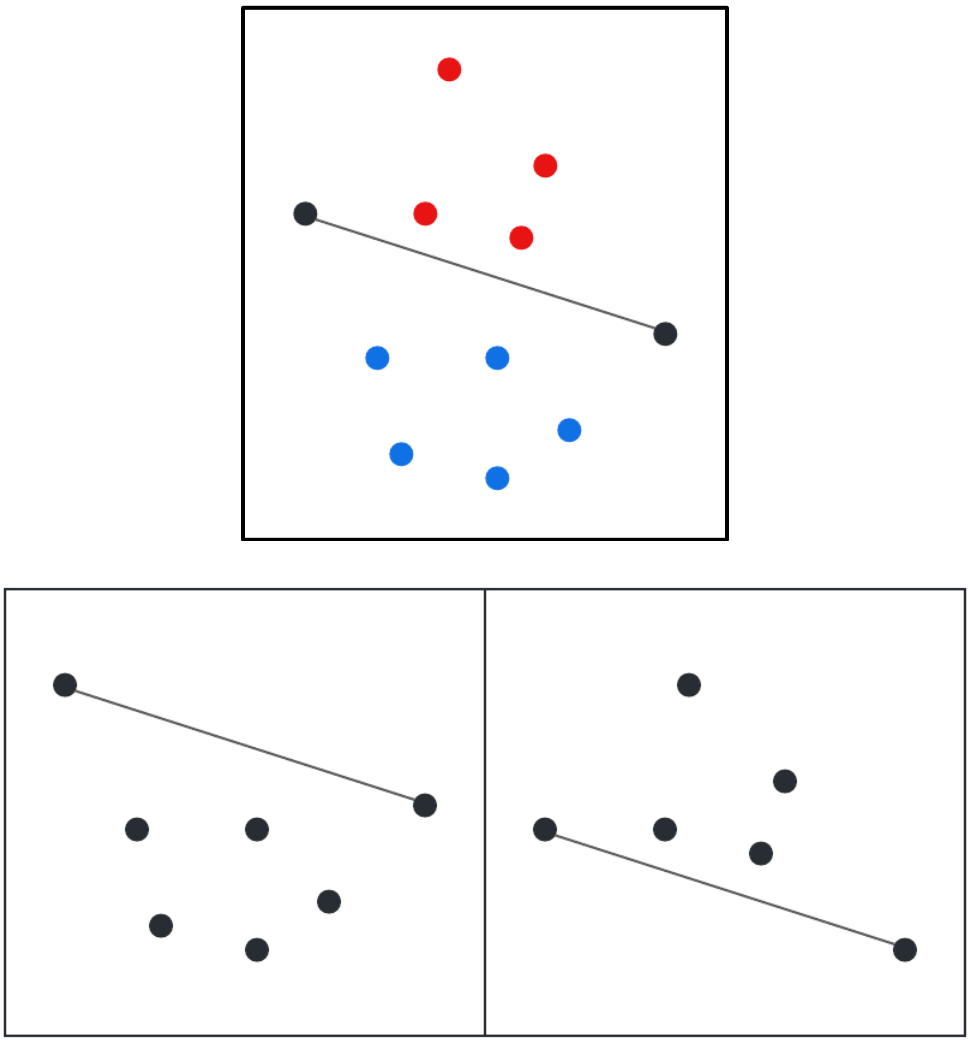
## 1.1 Tahap Divide

Pada tahap ini, persoalan yang akan dibagi adalah banyaknya titik yang akan dicek. Untuk pembagian pertama kali, program akan mencari titik terkiri dan titik terkanan, kemudian menghubungkannya menjadi sebuah garis.



Gambar 1.1. Ilustriasi pemilihan dua titik terjauh

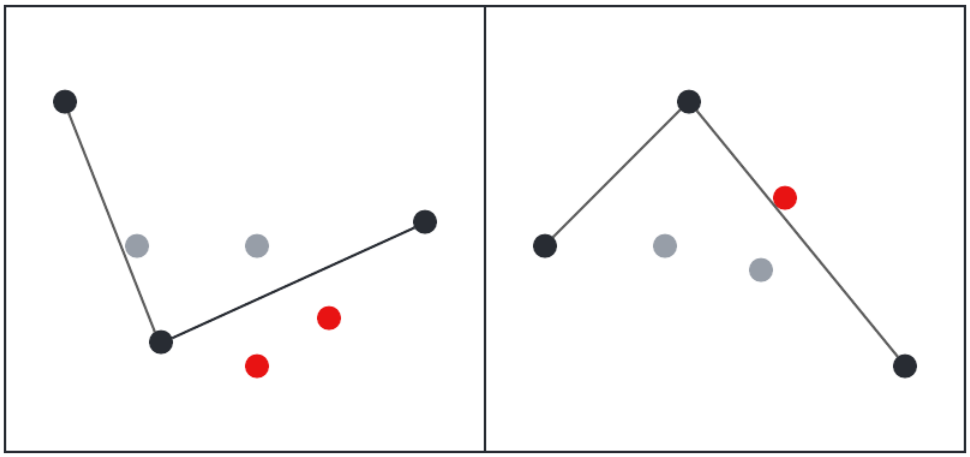
Garis tersebut digunakan untuk memisahkan titik menjadi dua bagian. Titik yang berada di atas garis akan masuk ke upa-persoalan atas, sedangkan titik yang berada di bawah garis akan masuk ke upa-persoalan bawah. Kemudian, panggil fungsi untuk menyelesaikan upa-persoalan atas dan bawah secara rekursif.



Gambar 1.2. Ilustriasi pembagian titik berdasarkan garis tengah

## 1.2 Tahap Conquer (Solve)

Pada tahap ini, persoalan yang telah dibagi akan dieksekusi. Jika setelah pembagian hanya terdapat dua titik, maka langsung hubungkan kedua titik tersebut. Jika terdapat lebih dari dua titik, cari titik yang berjarak terjauh dari garis kemudian hapus garis lama dan buat garis baru dari kedua titik awal ke titik baru tersebut.

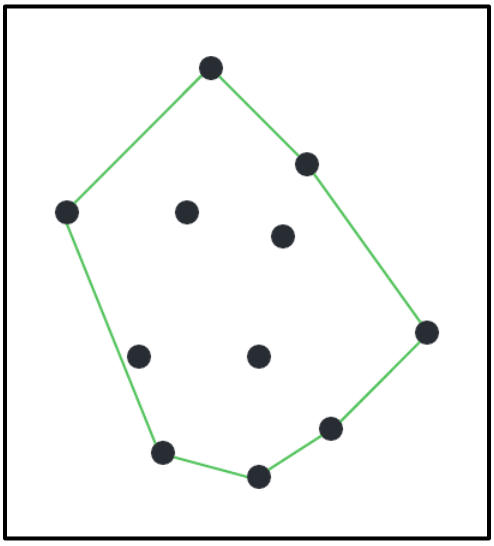


Gambar 1.3. Ilustriasi pembagian titik lagi berdasarkan jarak terjauh

Setelah itu, abaikan titik yang berada di dalam bidang dan lakukan tahap conquer lagi sampai tidak ada titik luar yang tersisa.

## 1.3 Tahap Combine

Tahap combine adalah tahap yang tidak diperlukan pada algoritma pencarian convex hull yang saya buat karena setelah semua titik luar selesai diproses, maka titik-titik tersebut sudah terhubung sehingga tidak perlu algoritma tambahan untuk menghubungkannya. Berikut ini adalah hasil akhirnya.



Gambar 1.4. Hasil dari pencarian convex hull

# 2 Source Program (Bahasa Python)

## 2.1 File main.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import random
import sys
import matplotlib.pyplot as plt
from myConvexHull import MyConvexHull

# Cek apakah terdapat argumen
if len(sys.argv) >= 2:
    if int(sys.argv[1]) == 1:
        print("Menggunakan dataset iris")
        data =datasets.load_iris()
    elif int(sys.argv[1]) == 2:
        print("Menggunakan dataset wine")
        data =datasets.load_wine()
    elif int(sys.argv[1]) == 3:
        print("Menggunakan dataset breast cancer")
        data = datasets.load_breast_cancer()
    else :
        print("Menggunakan dataset default : iris")
        data = datasets.load_iris()
    if len(sys.argv) >= 3:
        x = int(sys.argv[2])
        print(f"Menggunakan kolom ke-{x} sebagai sumbu x")
        if len(sys.argv) >= 4:
            y = int(sys.argv[3])
            print(f"Menggunakan kolom ke-{y} sebagai sumbu y")
        else :
            y = 1
            print("Menggunakan kolom ke-1 sebagai sumbu y (default)")
    else :
        x = 0
        print("Menggunakan kolom ke-0 sebagai sumbu x (default)")
        y = 1
        print("Menggunakan kolom ke-1 sebagai sumbu y (default)")

else :
    print("Menggunakan dataset default : iris")
    data = datasets.load_iris()
    x = 0
```

```
    print("Menggunakan kolom ke-0 sebagai sumbu x (default)")
    y = 1
    print("Menggunakan kolom ke-1 sebagai sumbu y (default)")
print()

# Buat DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)

# Visualisasi hasil ConvexHull
try :
    # Atur ukuran dan label grafik
    plt.figure(figsize = (10, 6))
    plt.title(data.feature_names[x] + ' vs ' + data.feature_names[y])
    plt.xlabel(data.feature_names[x])
    plt.ylabel(data.feature_names[y])

    # Cari convex hull untuk setiap target
    for i in range(len(data.target_names)):
        bucket = df[df['Target'] == i]
        bucket = bucket.iloc[:,[x,y]].values

        # Bagian ini diganti dengan hasil implementasi ConvexHull Divide &
Conquer
        hull = MyConvexHull(bucket)

        # Acak warna grafik
        color = '#' + str(hex(random.randint(48,160))[2::]) +
str(hex(random.randint(48,160))[2::]) + str(hex(random.randint(48,160))[2::])

        # Plot semua titik
        plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i],
color=color)

        # Plot garis yang membentuk convex hull
        for simplex in hull.simplices:
            plt.plot(bucket[simplex, 0], bucket[simplex, 1], color)

    # Tampilkan grafik
    plt.legend()
    plt.show()

except :
    print("Kolom dari dataset yang diberikan bukan merupakan bilangan")
    print("Tidak dapat membentuk convex hull")
```

## 2.2 File myConvexHull.py

```
import numpy as np
from utility import findDistance, splitDs


class MyConvexHull:
    def __init__(self, ds):
         # Ubah numpy list menjadi python list
        self.ds = ds.tolist()

        # Inisialisasi hasil
        self.chpoints = []
        self.topSimplices = []
        self.botSimplices = []

        # Cari convex hull
        self.findConvexHull()

        # Buat atribut points sebagai numpy array dari dataset
        self.points = np.array(self.ds)

        # Buat atribut simplices sebagai gabungan dari simplices bawah dan atas
        self.simplices = self.topSimplices + self.botSimplices


    # Mencari convex hull dari dataset ds.
    def findConvexHull(self):

        # Jika dataset hanya berisi 2 titik, langsung return
        if len(self.ds) <= 2:
            self.addToBotSimplices(self.ds[0], self.ds[1])
            return

        # Urutkan titik berdasarkan absis yang menaik
        sortedDs = sorted(self.ds, key=lambda x: x[0])
        p1 = sortedDs[0] # Titik minimum
        p2 = sortedDs[-1] # Titik maksimum

        # Masukkan ke dalam list hasil
        self.chpoints.append(p1)
        self.chpoints.append(p2)

        # Masukkan ke dalam simplices
        self.addToTopSimplices(p1, p2)
        self.addToBotSimplices(p1, p2)

        # Keluarkan p1 dan p2 dari sortedDs
        sortedDs.pop(0)
        sortedDs.pop(-1)

        # Pisahkan titik yang berada di atas dan di bawah gatis p1-p2
```

```
        dsAbove, dsBelow = splitDs(p1, p2, sortedDs)

        # Cek titik yang membentuk convex hull di atas dan bawah garis p1-p2
        self.findConvexHullPoint(p1, p2, dsAbove, True)
        self.findConvexHullPoint(p1, p2, dsBelow, False)


    # Mencari titik-titik yang membentuk convex hull secara rekursif
    # dengan menggunakan algoritma divide and conqueror
    def findConvexHullPoint(self, p1, p2, ds, isAbove):

        # Basis rekursi
        # Rekursi berhenti jika dataset ds sudah kosong atau ada titik yang
bernilai null
        if ds == [] or p1 is None or p2 is None:
            return []

        farthestDistance = -1   # Inisisasi jarak titik terjauh
        farthestPoint = None    # Inisiasi titik terjauh

        # Cari titik terjauh di datastore ds
        for point in ds:
            distance = findDistance(p1, p2, point)
            if distance > farthestDistance:
                farthestDistance = distance
                farthestPoint = point

        # Masukkan titik terjauh ke dalam list hasil
        #print(farthestPoint)
        self.chpoints.append(farthestPoint)

        # Masukkan ke dalam simplices
        if isAbove:
            self.addToTopSimplices(p1, farthestPoint)
            self.addToTopSimplices(farthestPoint, p2)
        else :
            self.addToBotSimplices(p1, farthestPoint)
            self.addToBotSimplices(farthestPoint, p2)

        # Hapus titik terjauh dari dataset
        ds.remove(farthestPoint)

        # Pisahkan titik titik berdasarkan lokasinya terhadap garis p1-p2
        ds1Above, ds1Below = splitDs(p1, farthestPoint, ds)
        ds2Above, ds2Below = splitDs(p2, farthestPoint, ds)

        if isAbove:
            # Jika is above bernilai true, cek titik bagian atas saja
            self.findConvexHullPoint(p1, farthestPoint, ds1Above, True)
            self.findConvexHullPoint(farthestPoint, p2, ds2Above, True)
        else:
            # Jika is above bernilai false, cek titik bagian bawah saja
```

```
        self.findConvexHullPoint(p1, farthestPoint, ds1Below, False)
        self.findConvexHullPoint(farthestPoint, p2, ds2Below, False)


# Menghubungkan dua titik ke dalam simplices bawah
def addToTopSimplices(self, p1, p2):

    # Cari index kedua titik
    i1 = self.ds.index(p1)
    i2 = self.ds.index(p2)

    simplicesSize = len(self.topSimplices)
    i = 0

    # Cek apakah titik p1 dan/atau p2 sudah ada
    # Jika ada, hapus terlebih dahulu
    while i < simplicesSize:
        if self.topSimplices[i][0] == i1:
            self.topSimplices.pop(i)
            simplicesSize -= 1
            continue
        if self.topSimplices[i][1] == i2:
            self.topSimplices.pop(i)
            simplicesSize -= 1
            continue
        i += 1

    # Masukkan index dari p1 dan p2 ke dalam simplices
    self.topSimplices.append([i1, i2])


# Menghubungkan dua titik ke dalam simplices atas
def addToBotSimplices(self, p1, p2):

    # Cari index kedua titik
    i1 = self.ds.index(p1)
    i2 = self.ds.index(p2)

    simplicesSize = len(self.botSimplices)
    i = 0

    # Cek apakah titik p1 dan/atau p2 sudah ada
    # Jika ada, hapus terlebih dahulu
    while i < simplicesSize:
        if self.botSimplices[i][0] == i1:
            self.botSimplices.pop(i)
            simplicesSize -= 1
            continue
        if self.botSimplices[i][1] == i2:
            self.botSimplices.pop(i)
            simplicesSize -= 1
            continue
```

```
            i += 1

        # Masukkan index dari p1 dan p2 ke dalam simplices
        self.botSimplices.append([i1, i2])
```

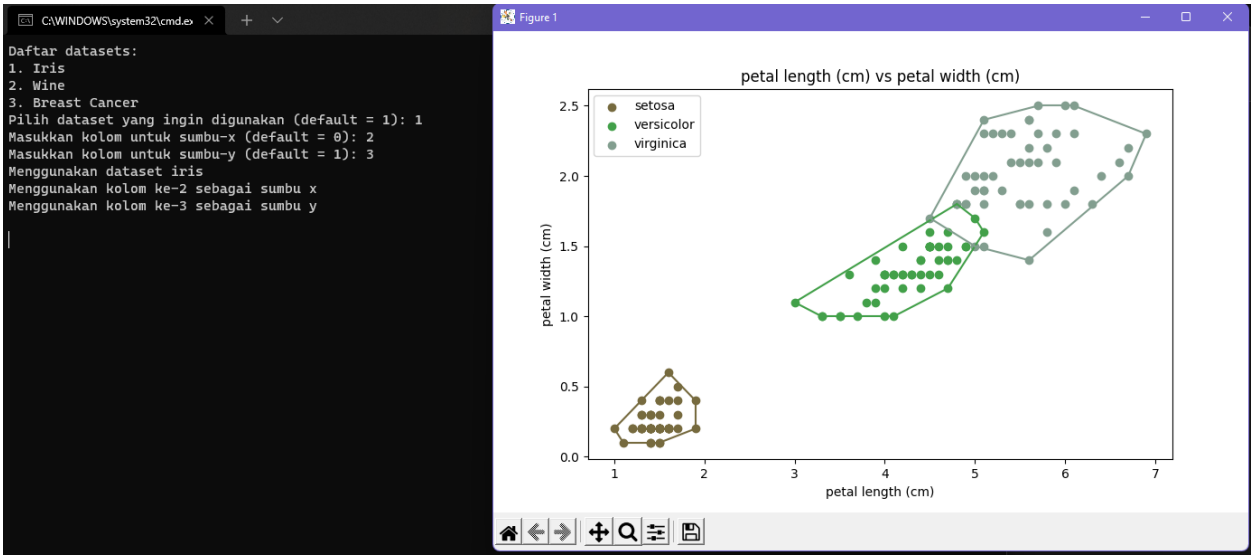
## 2.3 File utility.py

```
import math

# Mencari jarak antara garis p1-p2 dengan titik point
def findDistance(p1, p2, point):
    # Gunakan rumus ax + by + c = 0 untuk mengecek jarak titik terhadap garis
    a = p1[1] - p2[1]
    b = p2[0] - p1[0]
    c = p1[0]*p2[1] - p2[0]*p1[1]
    return abs(a*point[0] + b*point[1] + c)/math.sqrt(a*a + b*b)


# Membagi titik-titk yang ada di dataset ds berdasarkan lokasinya terhadap garis
p1-p2
def splitDs(p1, p2, ds):
    if p2[0] == p1[0]:
        return [], []

    # Inisiasi dataset titik atas dan bawah
    dsAbove = []
    dsBelow = []

    # Hitung gradien m dan konstanta c agar dapat membentuk persamaan garis y =
mx + c
    m = (p2[1] - p1[1]) / (p2[0] - p1[0])
    c = - m * p1[0] + p1[1]

    # Tentukan posisi setiap titik
    for point in ds:
        # Jika y > mx + c, letakkan di dataset titik atas
        if point[1] > m * point[0] + c:
            dsAbove.append(point)
        # Jika y < mx + c, letakkan di dataset titik bawah
        elif point[1] < m * point[0] + c:
            dsBelow.append(point)

    return dsAbove, dsBelow
```

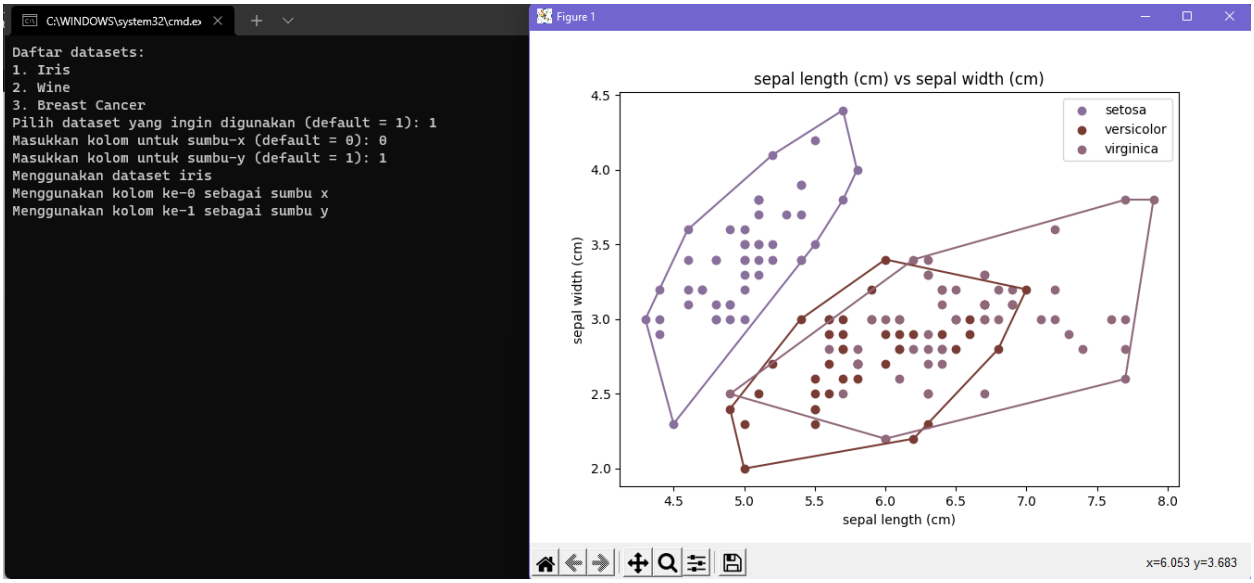
# 3 Skrinshut Program

## 3.1 Dataset Iris

### 3.1.1 Petal-Length vs Petal-Witdth

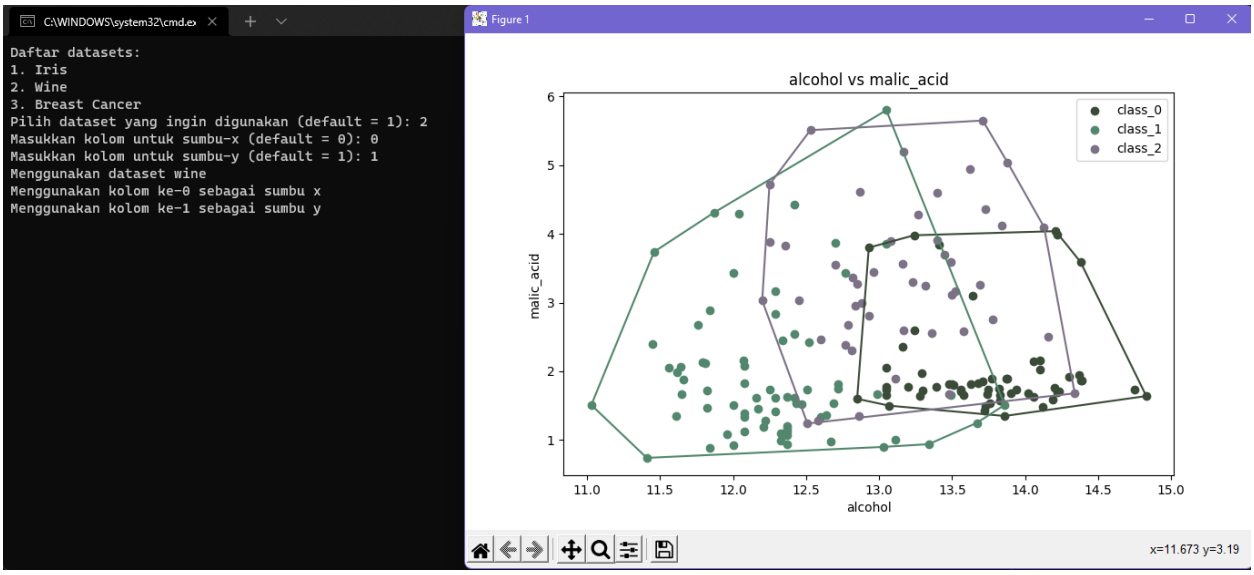



### 3.1.2 Sepal-Length vs Sepal-Witdth

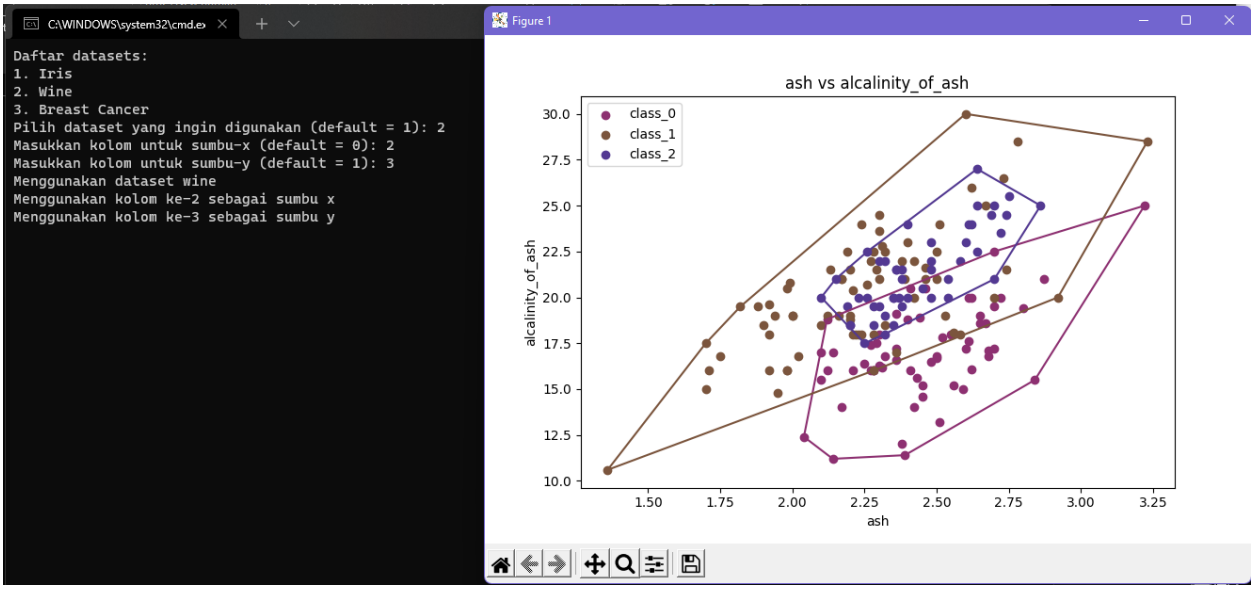


## 3.2 Dataset Wine

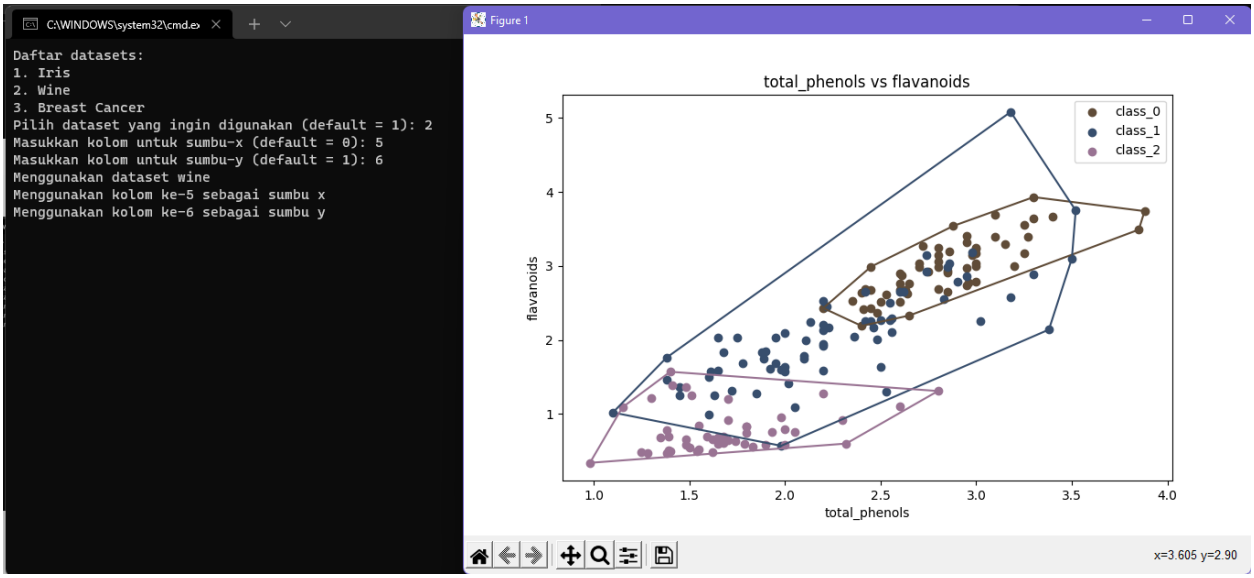
### 3.2.1 Alcohol vs Malic Acid



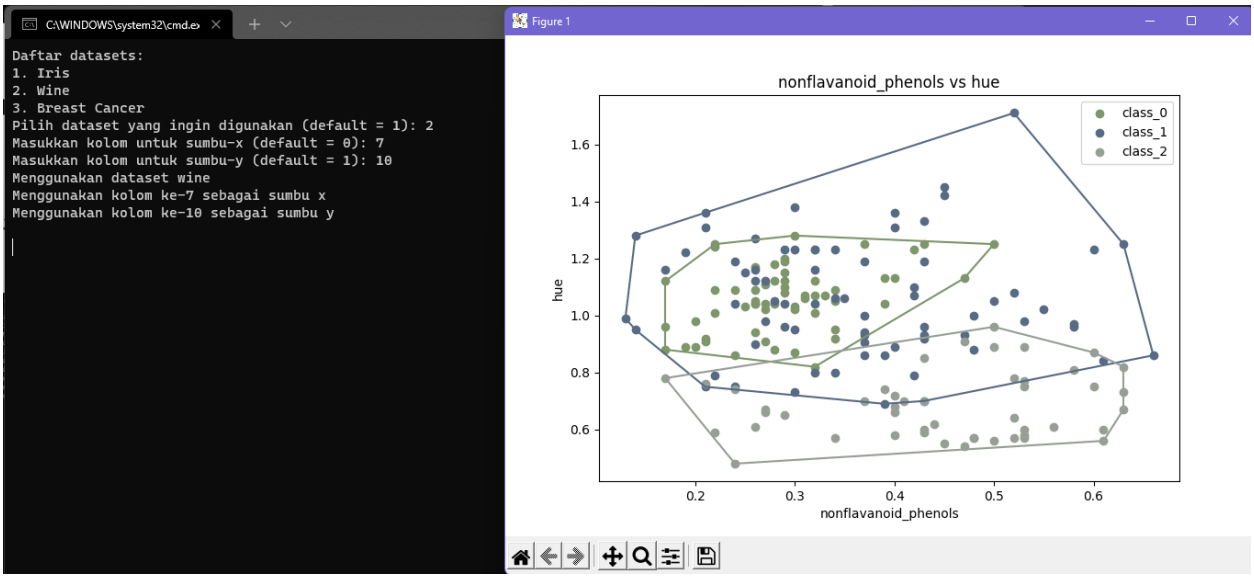

### 3.2.2 Ash vs Alcanity of Ash



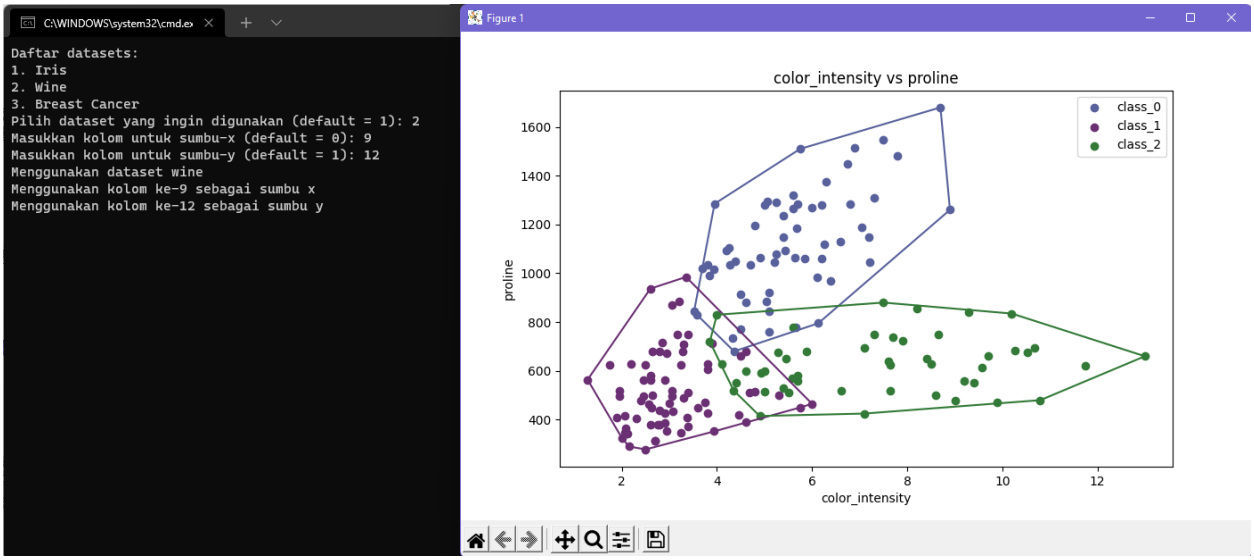
### 3.2.3 Total Phenols vs Flavanoids




### 3.2.4 Nonflavanoid Phenols vs Hue

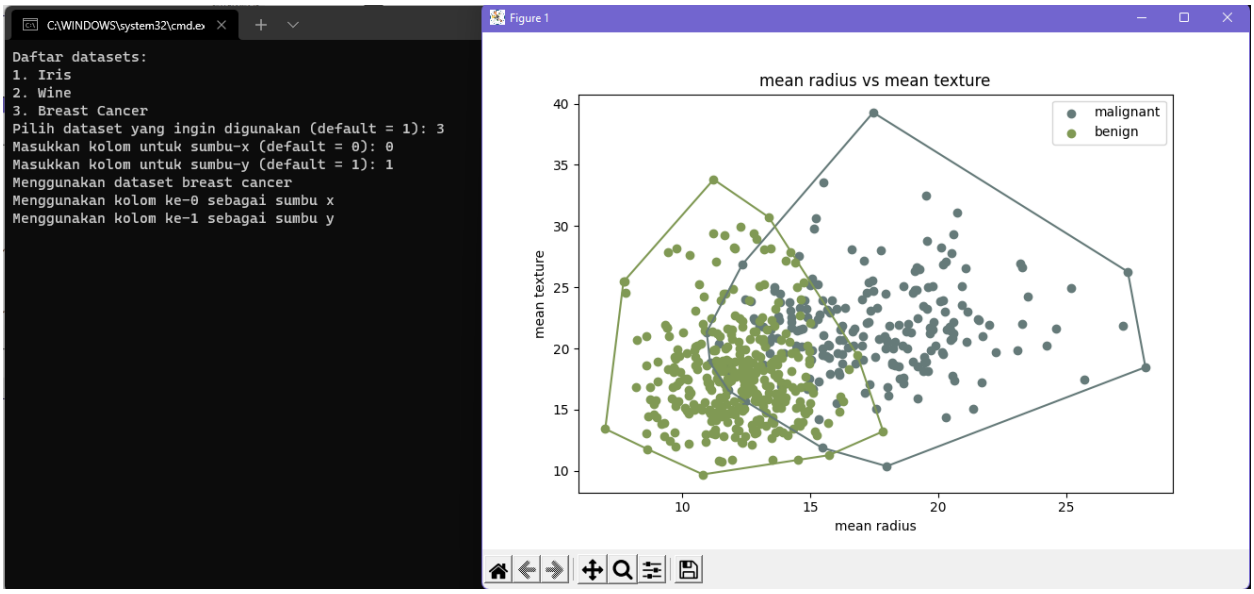


### 3.2.5 Color Intensity vs Proline

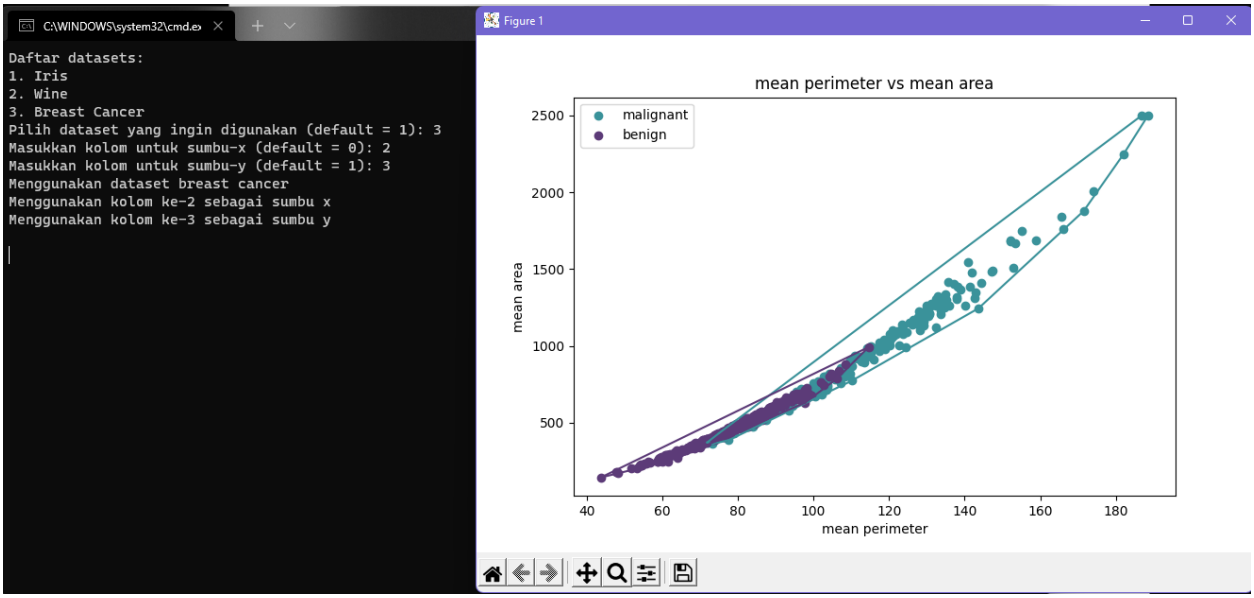



## 3.3 Dataset Breast Cancer

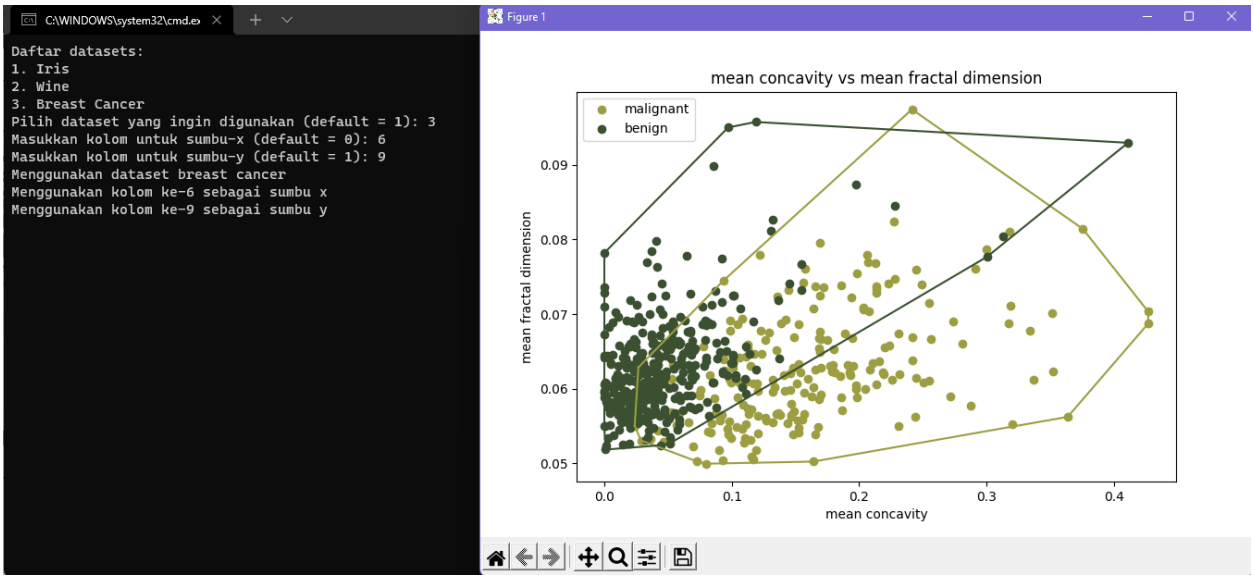
### 3.3.1 Mean Radius vs Mean Texture



### 3.3.2 Mean Perimeter vs Mean Area




### 3.3.3 Mean Concavity vs Mean Fractal Dimension



# 4 Alamat Github Kode Program

Kode program bisa diunduh di https://github.com/rozanfa/TucilStima2-DnC

# 5 Cek List Program

| Poin | Ya | Tidak |
|---|---|---|
| 1. Pustaka *myConvexHull* berhasil dibuat dan tidak ada kesalahan | √ | |
| 2. Convex hull *yang dihasilkan sudah benar* | √ | |
| 3. Pustaka *myConvexHull* dapat digunakan untuk menampilkan *convex hull* setiap label dengan warna yang berbeda | √ | |
| 4. Bonus: program dapat menerima input dan menuliskan output untuk dataset lainnya. | √ | |