Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Implementasi Convex Hull untuk Visualisasi Tes *Linear Separability Dataset* dengan Algoritma *Divide and Conquer*

Marchotridyo/13520119/K-02

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.		

Algoritma *divide and conquer*

Algoritma *divide and conquer* dilakukan pada tahap *sorting* dan pembuatan *convex hull*. Untuk itu, algoritma *divide and conquer* pada laporan ini akan dibagi menjadi dua, yaitu algoritma *divide and conquer* saat pengurutan dan *divide and conquer* saat membuat *convex hull*.

Divide and conquer pada pengurutan

Algoritma pengurutan untuk mengurutkan titik-titik yang terletak pada *bucket* (sebuah kumpulan titik) yang digunakan adalah algoritma quicksort versi Hoare.

Didefinisikan suatu titik P_1 dan titik P_2. P_1 dikatakan lebih kecil dari P_2 jika absis dari P_1 lebih kecil dari P_2 atau jika absisnya sama, tetapi ordinat P_1 lebih kecil dari ordinat P_2.

Secara garis besar, algoritma sorting dimulai dengan cara membuat partisi pada *bucket* dimulai dari awal sampai akhir. Partisi dibuat sedemikian sehingga seluruh anggota di partisi kiri lebih kecil dari seluruh anggota di partisi kanan. Sorting dilakukan lagi pada masing-masing partisi kiri dan partisi kanan.

Partisi dilakukan mengikuti langkah-langkah berikut:

(1) Memilih nilai tengah larik sebagai pivot, yaitu nilai yang terdapat pada (index awal + index akhir) *div* 2.
(2) Menginisialisasi p sebagai indeks pertama larik dan q sebagai indeks terakhir larik.
(3) p dimajukan sedemikian sehingga titik pada indeks p tidak lagi lebih kecil dari pivot.
(4) q dimundurkan sedemikian sehingga titik pada indeks q tidak lagi lebih besar dari pivot.
(5) Apabila p < q, pertukarkan titik pada indeks p dengan indeks q, majukan p sebanyak 1, mundurkan q sebanyak 1, lalu ulangi langkah (3) dan (4).
(6) Apabila p >= q, proses partisi selesai.
(7) Indeks akhir dari partisi kiri adalah nilai dari q.

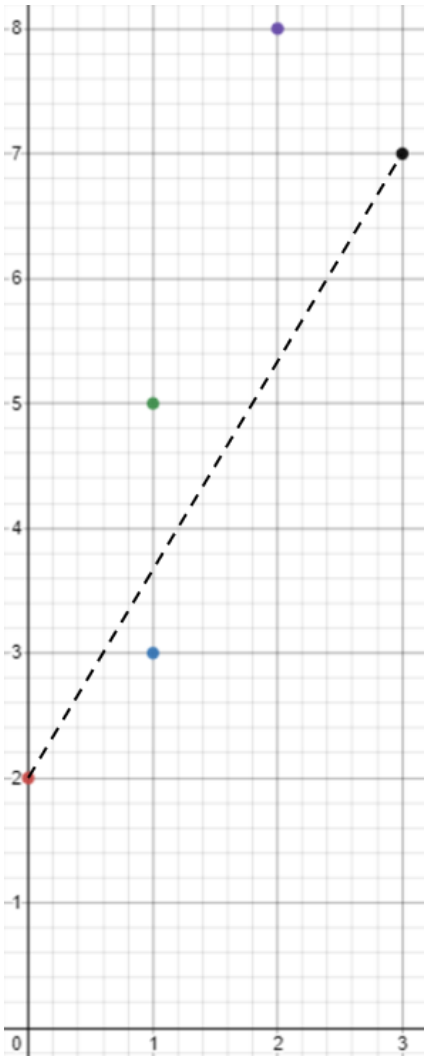
Setelah partisi selesai, dilakukan sorting pada upalarik dimulai dari indeks awal sampai q dan dari q + 1 sampai indeks akhir.

Divide and conquer pada pembuatan *convex hull*

Algoritma pembuatan *convex hull* dibuat berdasarkan algoritma yang ada di *slide* perkuliahan *Divide and Conquer* bagian 4 dengan sedikit modifikasi, yaitu mengecekan keberadaan titik terhadap garis (di atas atau di bawah) tidak menggunakan determinan, tetapi memanfaatkan persamaan garis y = mx + c.

Algoritma dimulai dengan melakukan pengurutan secara menaik terhadap input *bucket* yaitu kumpulan titik, yaitu [x,y]. Misalnya, apabila awalnya input berisikan kumpulan titik [[1, 5], [1, 3], [0, 2], [3, 7], [2, 8]], setelah dilakukan pengurutan didapat kumpulan titik [[0, 2], [1, 3], [1, 5], [2, 8], [3, 7]].

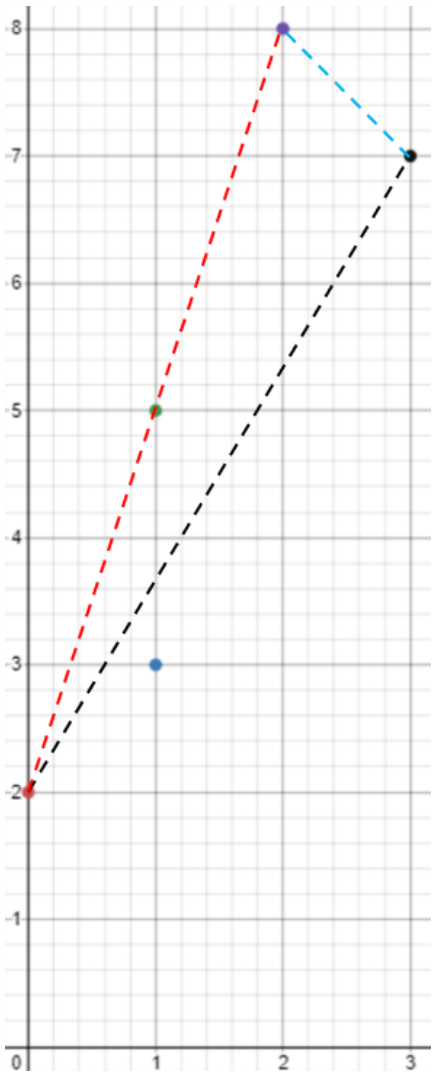
Setelah dilakukan pengurutan terhadap titik-titik tersebut, dibuat garis bayangan yang menghubungkan titik pertama (paling kiri bawah) dan titik terakhir (paling kanan atas). Menggunakan titik-titik dari contoh yang diberikan, kurang lebih akan tercipta garis seperti ini:



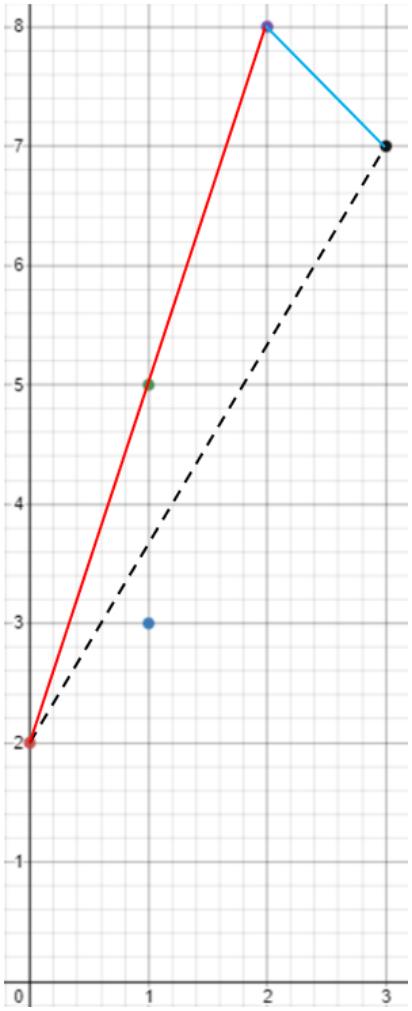
Setelah itu, akan terbentuk dua daerah: S_1 yang berisikan titik-titik yang terletak di atas garis, dan S_2 yang berisikan titik-titik yang terletak di bawah garis. Proses ini adalah proses *divide* dengan detail pencarian titik-titik yang berada di atas garis atau di bawah garis dilakukan secara *brute force*. Persamaan garis dalam bentuk $y = mx + c$ dibuat dari titik pertama dan titik terakhir untuk mendapatkan nilai m dan c. Apabila titik (x_0, y_0) memenuhi $y_0 > mx_0 + c$, titik berada di atas garis. Sebaliknya, apabila titik (x_0, y_0) memenuhi $y_0 < mx_0 + c$, titik berada di bawah garis.

Proses *conquer* dilakukan dengan cara memanggil algoritma *convex hull* untuk bagian atas terhadap S_1 dan memanggil algoritma *convex hull* untuk bagian bawah terhadap S_2.

Algoritma *convex hull* untuk bagian atas dilakukan dengan cara mencari titik yang memiliki jarak terjauh ke garis yang membaginya. Jarak dihitung menggunakan formula $d = \frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}$ dengan keterangan bahwa Ax + By +C adalah persamaan garis yang membaginya dan (x_0, y_0) adalah titik yang ingin dihitung jaraknya ke garis. Pencarian ini dilakukan menggunakan metode *brute force*. Setelah titik tersebut ditemukan, dibuat garis bayangan yang menghubungkan titik garis yang membaginya sebelumnya ke titik tersebut. Melanjutkan contoh yang tadi, akan tercipta garis merah dan biru pada gambar berikut:

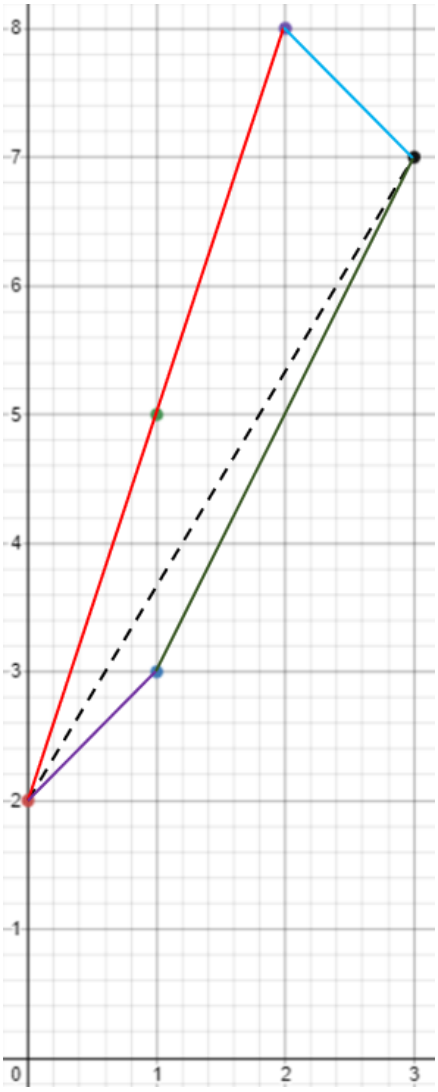


Algoritma *convex hull* untuk bagian atas akan dipanggil untuk daerah yang berada di atas garis merah dan daerah yang berada di atas garis biru. Apabila tidak ada titik yang berada di atas garis merah, maka garis merah diubah menjadi garis tebal, begitu pula untuk garis biru. Pada kasus ini, karena tidak lagi ada titik yang berada di atas garis merah dan garis biru, kedua garis tersebut diubah menjadi garis tebal, ditunjukkan pada gambar di bawah:

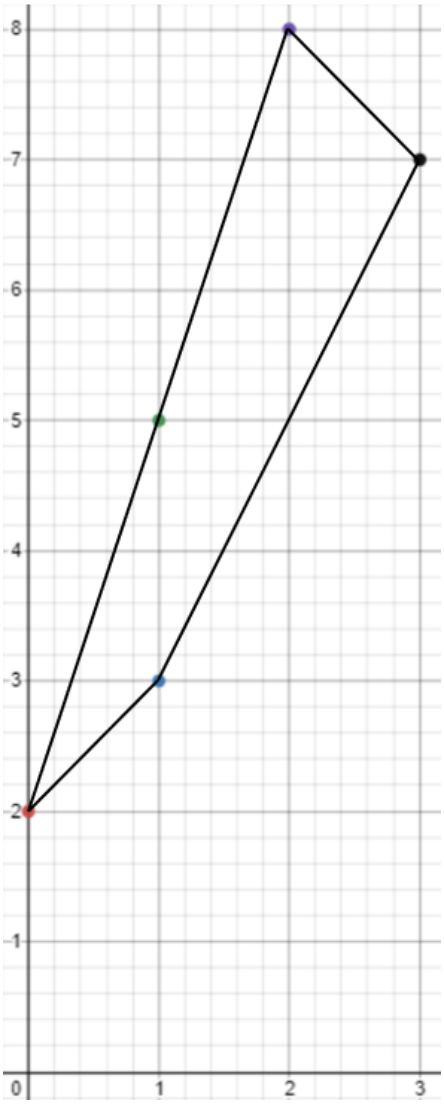


Hal yang sama dilakukan untuk algoritma *convex hull* untuk bagian bawah, dengan satu perbedaan bahwa pencarian terus dilakukan untuk titik-titik yang berada di bawah garis, bukan di atas garis seperti algoritma *convex hull* untuk bagian atas.

Setelah *convex hull* untuk bagian bawah selesai dibuat, akan terbentuk gambar seperti ini:



Kumpulan garis-garis yang dibuat tebal adalah *convex hull* dari kumpulan titik yang diberikan. Didapat hasil akhir seperti gambar di bawah ini:



Source code program

Program ditulis menggunakan bahasa Python dan *source code*-nya ditunjukkan oleh kode yang dilampirkan berikut. Ada dua *file* .py yang dibuat, yaitu satu pustaka implementasi *convex hull* dan satu program utama untuk menjalankan visualisasi dataset.

Source code pustaka implementasi *convex hull*

```
import numpy as np

def isPointLEThanPoint(p1, p2):
  '''
    p1 dan p2 adalah list [x, y]
    p1 dikatakan <= p2 jika
    p1.x < p2.x atau p1.x = p2.x dan p1.y <= p2.y
  '''
  if (p1[0] < p2[0]):
    return True
  elif (p1[0] > p2[0]):
    return False
  else:
    return p1[1] <= p2[1]

def isPointGEThanPoint(p1, p2):
  '''
    p1 dan p2 adalah list [x, y]
    p1 dikatakan >= p2 jika
    p1.x > p2.x atau p1.x = p2.x dan p1.y >= p2.y
  '''
  if (p1[0] > p2[0]):
    return True
  elif (p1[0] < p2[0]):
    return False
  else:
    return p1[1] >= p2[1]

def partition(bucket, start, end):
  '''
    Implementasi partisi titik-titik pada bucket
    menggunakan metode quicksort.
    Mengembalikan index akhir dari partisi pertama.
  '''
  pivot = bucket[(start + end) // 2]
  p = start
  q = end
  while (True):
    # Cari lokasi yang benar untuk p
    while (not isPointGEThanPoint(bucket[p], pivot)):
      p += 1
    # Cari lokasi yang benar untuk q
    while (not isPointLEThanPoint(bucket[q], pivot)):
      q -= 1
    # Apabila p < q, lakukan swap
```

```
    if (p < q):
      tmp = bucket[p]
      bucket[p] = bucket[q]
      bucket[q] = tmp
      # Lanjutkan iterasi untuk p += 1 dan q -= 1
      p += 1
      q -= 1
    else:
      # Looping berakhir
      break
  return q

def sortBucket(bucket, start, end):
  '''
    Implementasi quicksort pada bucket.
  '''
  if (start < end):
    # Sorting hanya dilakukan jika start < end
    k = partition(bucket, start, end)
    sortBucket(bucket, start, k)
    sortBucket(bucket, k + 1, end)

def isPointAboveLine(p1, p2, p):
  '''
    Mengecek apakah p ada di atas p1p2
    Dengan dasar persamaan
    y = mx + c => c = y - mx
    apakah y0 > m*x0 + c?
  '''
  if (p2[0] != p1[0]):
    m = (p2[1] - p1[1]) / (p2[0] - p1[0])
    c = p1[1] - m * p1[0]
    return p[1] > m*p[0] + c
  else:
    return False

def isPointBelowLine(p1, p2, p):
  '''
    Mengecek apakah p ada di atas p1p2
    Dengan dasar persamaan
    y = mx + c => c = y - mx
    apakah y0 < m*x0 + c?
  '''
  if (p2[0] != p1[0]):
    m = (p2[1] - p1[1]) / (p2[0] - p1[0])
    c = p1[1] - m * p1[0]
    return p[1] < m*p[0] + c
  else:
    return False

def dividePointsByLine(p1, p2, bucket):
  '''
    Mengembalikan tupel (s1, s2)
    di mana s1 adalah titik-titik pada bucket yang berada di atas garis p1p2
    dan s2 adalah titik-titik pada bucket yang berada di bawah garis p1p2
  '''
```

```
  s1 = []
  s2 = []
  for p in bucket:
    if isPointAboveLine(p1, p2, p):
      s1.append(p)
    elif isPointBelowLine(p1, p2, p):
      s2.append(p)
  return (s1, s2)

def getDistanceToLine(p1, p2, p):
  '''
    Mencari jarak dari titik p ke garis p1p2
  '''
  # Bentuk persamaan garis Ax + By + C = 0
  A = p1[1] - p2[1]
  B = p2[0] - p1[0]
  C = p1[0]*p2[1] - p2[0]*p1[1]
  return (abs(A*p[0] + B*p[1] + C))/((A ** 2 + B ** 2) ** (1/2))

def convexHullAbove(p1, p2, s, hullPairs):
  '''
    Algoritma convex hull untuk daerah atas
  '''
  if (len(s) > 0):
    # Cari titik terjauh terhadap garis p1p2 dari s
    maxDist = None
    maxP = None
    maxPIdx = None
    for i, p in enumerate(s):
      if maxDist == None:
        maxDist = getDistanceToLine(p1, p2, p)
        maxP = p
        maxPIdx = i
      else:
        if (getDistanceToLine(p1, p2, p) > maxDist):
          maxDist = getDistanceToLine(p1, p2, p)
          maxP = p
          maxPIdx = i
    # Hapus dari s
    s.pop(maxPIdx)
    # Bagi daerah untuk diconquer secara rekursif
    above1, below1 = dividePointsByLine(p1, maxP, s)
    above2, below2 = dividePointsByLine(p2, maxP, s)
    # Penambahan ke hullPairs jika ada daerah yang kosong
    if len(above1) == 0:
      hullPairs.append([p1, maxP])
    if len(above2) == 0:
      hullPairs.append([p2, maxP])
    # Conquer
    convexHullAbove(p1, maxP, above1, hullPairs)
    convexHullAbove(p2, maxP, above2, hullPairs)

def convexHullBelow(p1, p2, s, hullPairs):
  '''
    Algoritma convex hull untuk daerah bawah
  '''
```

```
  if (len(s) > 0):
    # Cari titik terjauh terhadap garis p1p2 dari s
    maxDist = None
    maxP = None
    maxPIdx = None
    for i, p in enumerate(s):
      if maxDist == None:
        maxDist = getDistanceToLine(p1, p2, p)
        maxP = p
        maxPIdx = i
      else:
        if (getDistanceToLine(p1, p2, p) > maxDist):
          maxDist = getDistanceToLine(p1, p2, p)
          maxP = p
          maxPIdx = i
    # Hapus dari s
    s.pop(maxPIdx)
    # Bagi daerah untuk diconquer secara rekursif
    above1, below1 = dividePointsByLine(p1, maxP, s)
    above2, below2 = dividePointsByLine(p2, maxP, s)
    # Penambahan ke hullPairs jika ada daerah yang kosong
    if len(below1) == 0:
      hullPairs.append([p1, maxP])
    if len(below2) == 0:
      hullPairs.append([p2, maxP])
    # Conquer
    convexHullBelow(p1, maxP, below1, hullPairs)
    convexHullBelow(p2, maxP, below2, hullPairs)

def ConvexHull(bucket):
  '''
    Mengembalikan hullPairs, yaitu list of [point1, point2]
    di mana point1 dan point2 sendiri merupakan list [x, y]
    untuk nantinya diplot.
  '''
  myBucket = bucket.tolist()
  hullPairs = []
  # Kasus 'sederhana' yaitu di bucket hanya ada satu atau dua titik.
  # Dalam kasus ini, cukup hubungkan dua garis tersebut.
  if len(myBucket) == 1:
    return [[myBucket[0], myBucket[0]]]
  if len(bucket) == 2:
    return [[myBucket[0], myBucket[1]]]
  # Dalam realisasinya, perlu melakukan sorting dahulu terhadap bucket.
  # Bucket diurutkan berdasarkan absis menaik, jika absis sama, berdasarkan
ordinat menaik.
  # Implementasinya menggunakan quickSort sendiri.
  sortBucket(myBucket, 0, len(myBucket) - 1)
  # Masukkan titik pertama dan titik terakhir ke dalam hullPairs
  p1 = myBucket[0]; pn = myBucket[-1]
  # Hapus dari titik-titik yang ada
  myBucket.pop(0); myBucket.pop(-1)
  # Bagi menjadi dua daerah: s1 (titik-titik di atas garis) dan s2
(titik-titik di bawah garis)
  s1, s2 = dividePointsByLine(p1, pn, myBucket)
  # Bila ada satu daerah yang kosong, maka masukkan pair ke dalam hullPairs
```

```
  if len(s1) == 0 or len(s2) == 0:
    hullPairs.append([p1, pn])
  # Kerjakan dengan divide and conquer
  convexHullAbove(p1, pn, s1, hullPairs)
  convexHullBelow(p1, pn, s2, hullPairs)
  return hullPairs
```

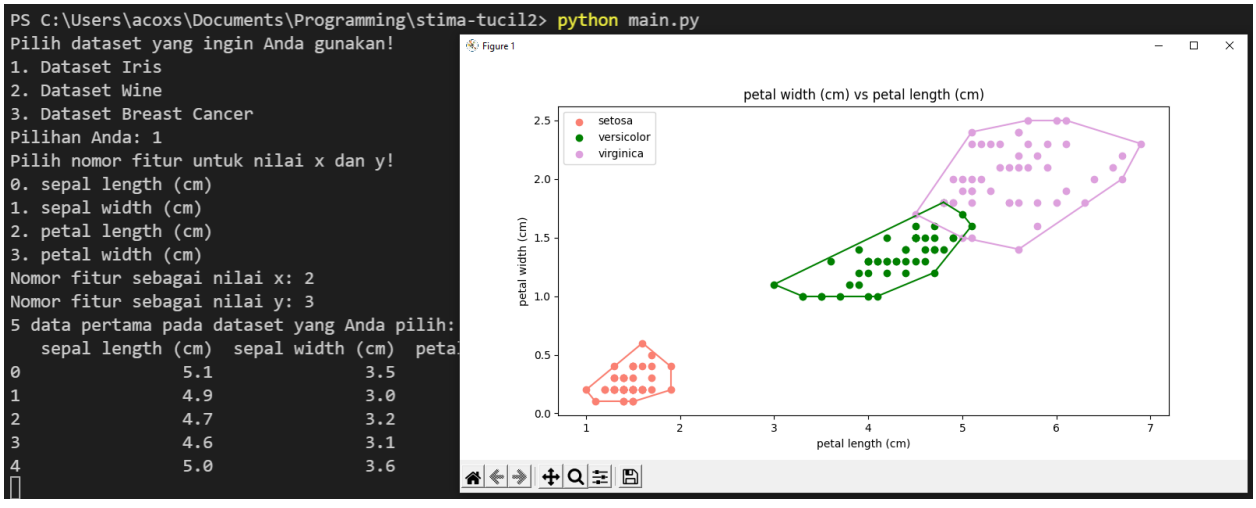
Source code program utama untuk menjalankan visualisasi dataset

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
irisData = datasets.load_iris()
wineData = datasets.load_wine()
breastCancerData = datasets.load_breast_cancer()
import matplotlib.pyplot as plt
from myConvexHull import ConvexHull

# Minta input dataset
print("Pilih dataset yang ingin Anda gunakan!")
print("1. Dataset Iris")
print("2. Dataset Wine")
print("3. Dataset Breast Cancer")
while (True):
  n = int(input("Pilihan Anda: "))
  if (n >= 1 and n <= 3):
    break

# Set data sesuai input
data = None
if n == 1:
  data = irisData
elif n == 2:
  data = wineData
elif n == 3:
  data = breastCancerData

# Minta input x dan y
x = None; y = None
print("Pilih nomor fitur untuk nilai x dan y!")
for i in range(len(data.feature_names)):
  print(f'{i}. {data.feature_names[i]}')
while (True):
  x = int(input("Nomor fitur sebagai nilai x: "))
  if (x >= 0 and x < len(data.feature_names)):
    break
while (True):
  y = int(input("Nomor fitur sebagai nilai y: "))
  if (y >= 0 and y < len(data.feature_names)):
    break
```

```
# Tunjukkan head dari dataset
print("5 data pertama pada dataset yang Anda pilih:")
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.head())

# Cetak convex hull
plt.figure(figsize = (10, 6))
colors = ['salmon', 'green', 'plum', 'aquamarine', 'grey', 'purple', 'azure',
'indigo', 'red', 'beige', 'ivory', 'aqua']
plt.title(f'{data.feature_names[y]} vs {data.feature_names[x]}')
plt.xlabel(data.feature_names[x])
plt.ylabel(data.feature_names[y])
for i in range(len(data.target_names)):
  bucket = df[df['Target'] == i]
  bucket = bucket.iloc[:,[x,y]].values
  hull = ConvexHull(bucket)
  plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i],
c=colors[i])
  for pair in hull:
    plt.plot([pair[0][0], pair[1][0]], [pair[0][1], pair[1][1]], colors[i])
plt.legend()
plt.show()
```

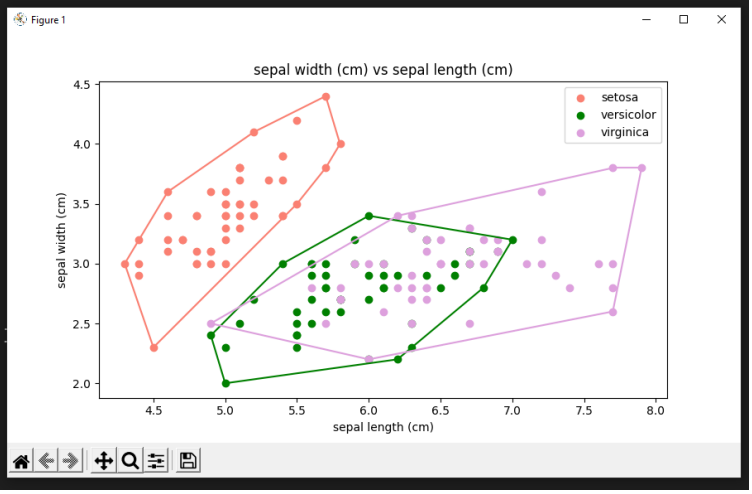
Screenshot input-output dari program

Dataset iris: petal width (cm) vs petal length (cm)



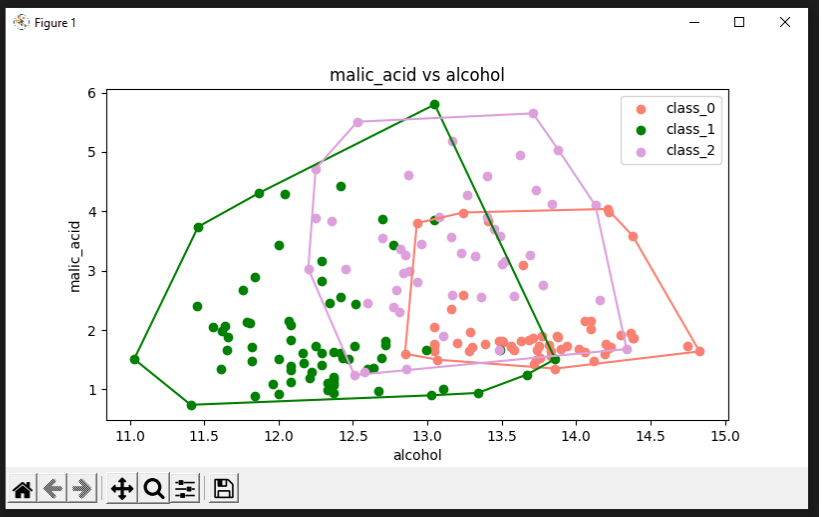
Dataset iris: sepal width (cm) vs sepal length (cm)

```
PS C:\Users\acoxs\Documents\Programming\stima-tucil2> python main.py
Pilih dataset yang ingin Anda gunakan!
1. Dataset Iris
2. Dataset Wine
3. Dataset Breast Cancer
Pilihan Anda: 1
Pilih nomor fitur untuk nilai x dan y!
0. sepal length (cm)
1. sepal width (cm)
2. petal length (cm)
3. petal width (cm)
Nomor fitur sebagai nilai x: 0
Nomor fitur sebagai nilai y: 1
5 data pertama pada dataset yang Anda pilih:
   sepal length (cm)  sepal width (cm)  petal
0                5.1               3.5
1                4.9               3.0
2                4.7               3.2
3                4.6               3.1
4                5.0               3.6
```



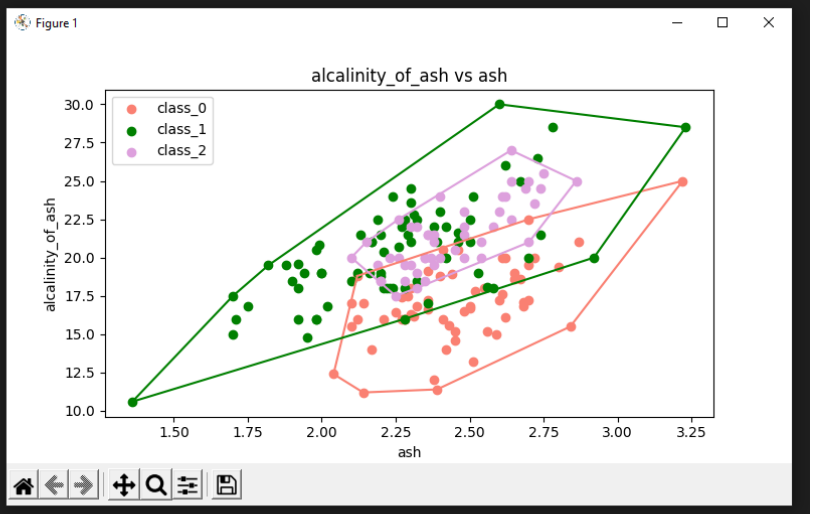
Dataset wine: malic_acid vs alcohol

```
PS C:\Users\acoxs\Documents\Programming\stima-tucil2> python main.py
Pilih dataset yang ingin Anda gunakan!
1. Dataset Iris
2. Dataset Wine
3. Dataset Breast Cancer
Pilihan Anda: 2
Pilih nomor fitur untuk nilai x dan y!
0. alcohol
1. malic_acid
2. ash
3. alcalinity_of_ash
4. magnesium
5. total_phenols
6. flavanoids
7. nonflavanoid_phenols
8. proanthocyanins
9. color_intensity
10. hue
11. od280/od315_of_diluted_wines
12. proline
Nomor fitur sebagai nilai x: 0
Nomor fitur sebagai nilai y: 1
5 data pertama pada dataset yang Anda pilih:
   alcohol  malic_acid   ash  alcalinity_of_ash  ...   hue  od280/od315_of_diluted_wines  proline  Target
0    14.23        1.71  2.43               15.6  ...  1.04                          3.92   1065.0       0
```



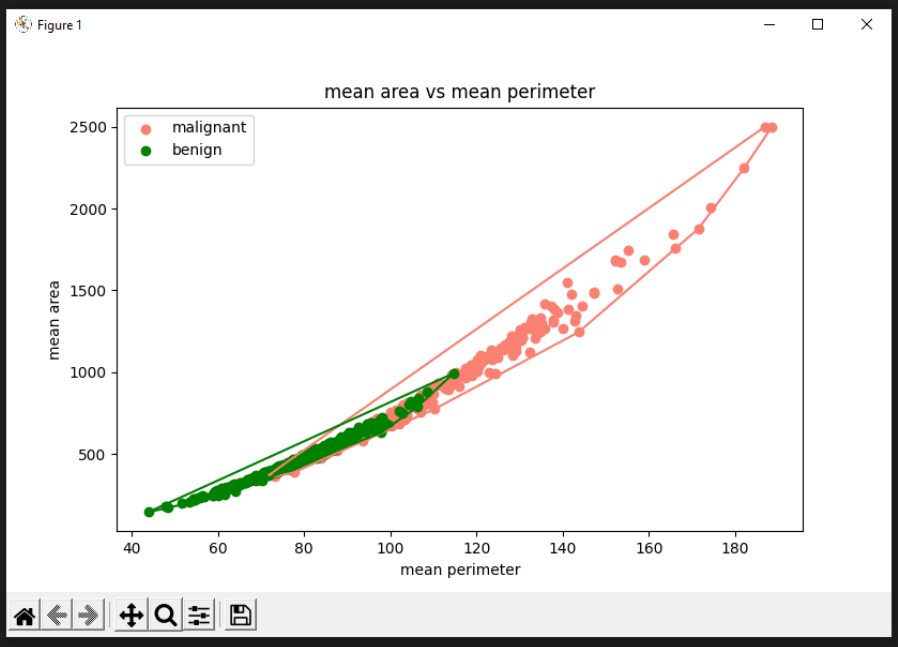
Dataset wine: alcalinity_of_ash vs ash

```
PS C:\Users\acoxs\Documents\Programming\stima-tucil2> python main.py
Pilih dataset yang ingin Anda gunakan!
1. Dataset Iris
2. Dataset Wine
3. Dataset Breast Cancer
Pilihan Anda: 2
Pilih nomor fitur untuk nilai x dan y!
0. alcohol
1. malic_acid
2. ash
3. alcalinity_of_ash
4. magnesium
5. total_phenols
6. flavanoids
7. nonflavanoid_phenols
8. proanthocyanins
9. color_intensity
10. hue
11. od280/od315_of_diluted_wines
12. proline
Nomor fitur sebagai nilai x: 2
Nomor fitur sebagai nilai y: 3
5 data pertama pada dataset yang Anda pilih:
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  ...  color_intensity   hue  od280/od315_of_diluted_w
```

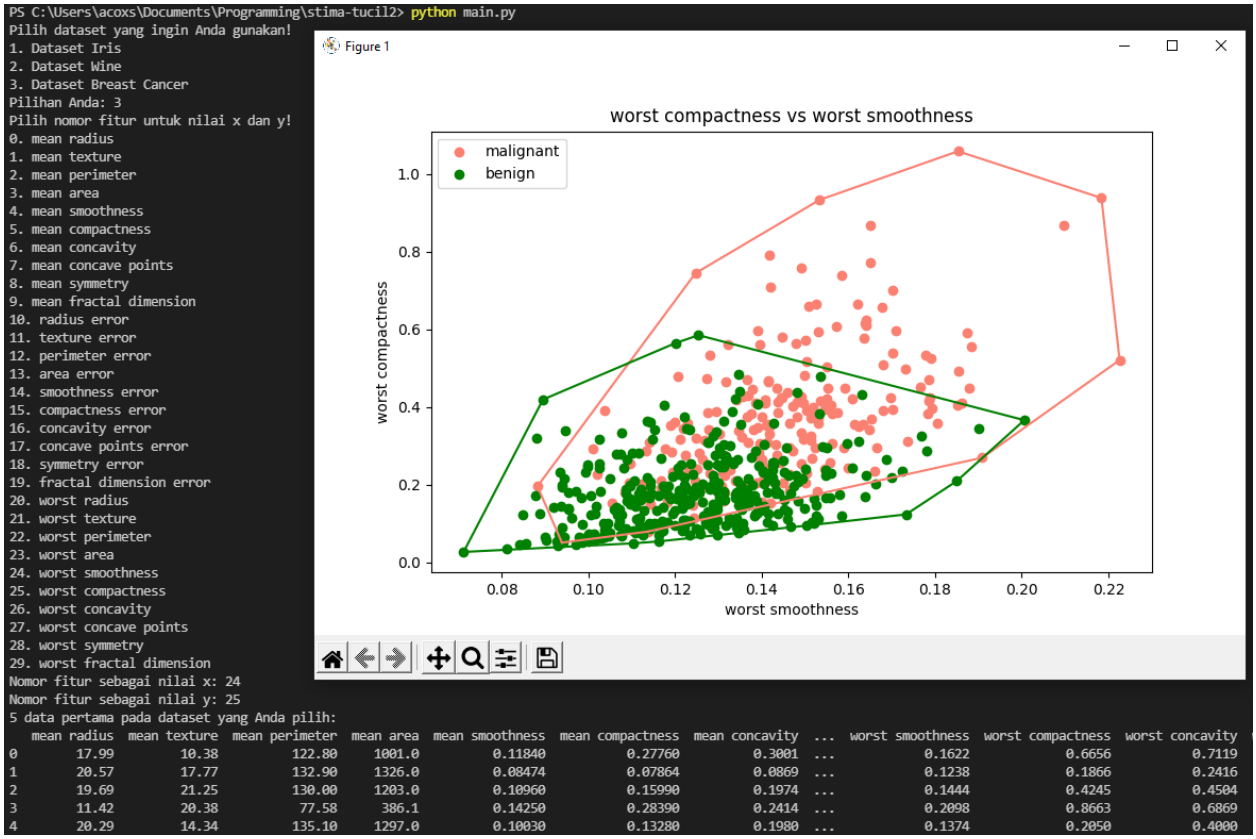


Dataset breast cancer: mean area vs mean perimeter

```
PS C:\Users\acoxs\Documents\Programming\stima-tucil2> python main.py
Pilih dataset yang ingin Anda gunakan!
1. Dataset Iris
2. Dataset Wine
3. Dataset Breast Cancer
Pilihan Anda: 3
Pilih nomor fitur untuk nilai x dan y!
0. mean radius
1. mean texture
2. mean perimeter
3. mean area
4. mean smoothness
5. mean compactness
6. mean concavity
7. mean concave points
8. mean symmetry
9. mean fractal dimension
10. radius error
11. texture error
12. perimeter error
13. area error
14. smoothness error
15. compactness error
16. concavity error
17. concave points error
18. symmetry error
19. fractal dimension error
20. worst radius
21. worst texture
22. worst perimeter
23. worst area
24. worst smoothness
25. worst compactness
26. worst concavity
27. worst concave points
28. worst symmetry
29. worst fractal dimension
Nomor fitur sebagai nilai x: 2
Nomor fitur sebagai nilai y: 3
5 data pertama pada dataset yang Anda pilih:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  mean compactness  mean concavity  ...  worst smoothness  worst compactness  worst concavity
0        17.99         10.38          122.80     1001.0          0.11840           0.27760          0.3001  ...            0.1622             0.6656           0.7119
1        20.57         17.77          132.90     1326.0          0.08474           0.07864          0.0869  ...            0.1238             0.1866           0.2416
2        19.69         21.25          130.00     1203.0          0.10960           0.15990          0.1974  ...            0.1444             0.4245           0.4504
3        11.42         20.38           77.58      386.1          0.14250           0.28390          0.2414  ...            0.2098             0.8663           0.6869
4        20.29         14.34          135.10     1297.0          0.10030           0.13280          0.1980  ...            0.1374             0.2050           0.4000
```



Dataset breast cancer: worst compactness vs worth smoothness



Alamat drive yang berisi kode program

https://drive.google.com/drive/folders/1J0eb69jcox9OD_zWpQpXoRu2JQxKbTcf?usp=sharing