# STRATEGI ALGORITMA

# CONVEX HULL DENGAN ALGORITMA DIVIDE AND CONQUER

## LAPORAN TUGAS KECIL 2

**Diajukan sebagai Salah Satu Tugas Kecil Mata Kuliah Strategi Algoritma pada Semester 4**

**Tahun Akademik 2021-2022**


**Oleh**

**Azka Syauqy Irsyad 13520107**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

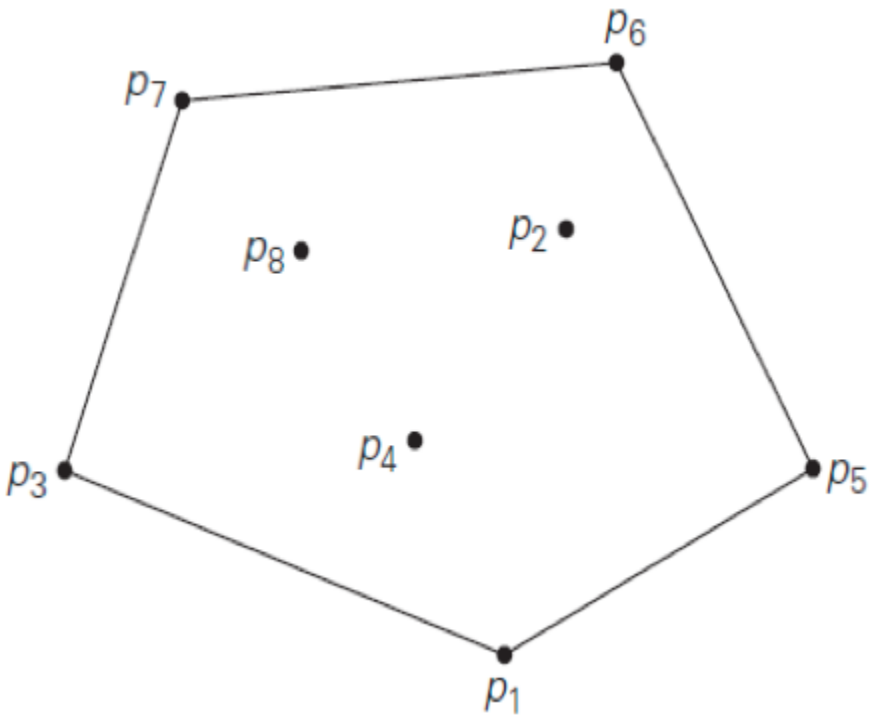
**BANDUNG**

**2022**

## I. Algoritma *Divide and Conquer* pada Pengaplikasian *Convex Hull*

Ada banyak sekali persoalan-persoalan yang berhubungan dengan komputasi, misalnya adalah persoalan untuk mencari *linear separability* dari data-data yang menyusun suatu dataset. Persoalan ini mengharuskan kita untuk mencari tahu apakah suatu atribut yang berada pada dataset tersebut *dependet* atau *independent* terhadap suatu atribut lain yang berada juga di dataset tersebut. Ada beberapa cara untuk menyelesaikan persoalan tersebut. Namun, kali ini akan dijelaskan metode penyelesaian dengan menggunakan *convex hull*.

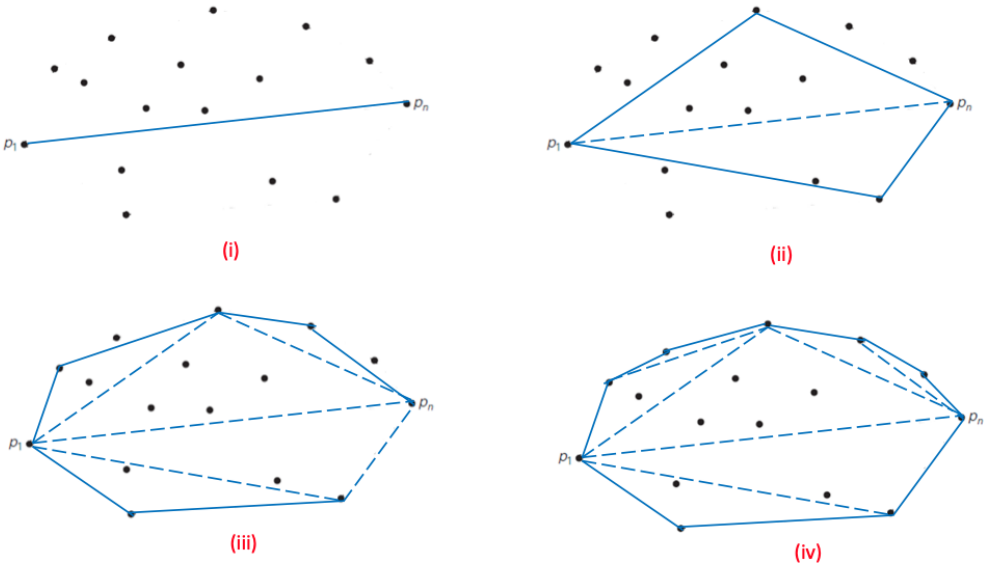
*Convex hull* merupakan salah satu hal yang penting dalam persoalan-persoalan komputasi geometri. Himpunan titik yang ada akan dikatakan *convex* jika untuk sembarang dua titik pada bidang tersebut, seluruh segemn garis yang berakhir di dua titik itu ada pada himpunan titiknya. Salah satu contoh dari *convex hull* adalah seperti berikut.



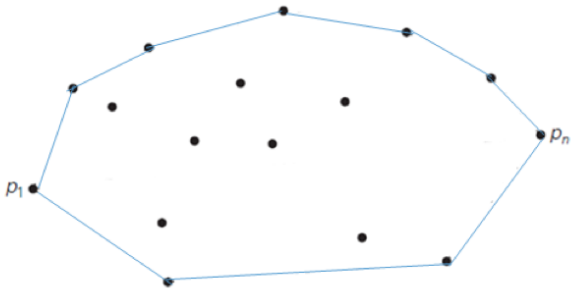

Gambar 1 *Convex Hull* untuk 8 titik

Dalam mengaplikasikan metode *convex hull* ini, kita dapat memanfaatkan algoritma *divide and conquer*. Algoritma membuat persoalan dibagi ke dalam bagian-bagian tertentu, dilakukan komputasi, dan pada solusi akhir dari bagian-bagian

tersebut akan digabungkan sehingga menghasilkan solusi final. Untuk proses dari pencarian titik-titik *convex hull* dapat diilustrasikan sebagai berikut.




Hasil akhir:




Gambar 2 Proses Pembentukan *Convex Hull*

Pada pengimplementasian algoritma ini, dari pengambilan data titik-titik dua atribut dari suatu dataset (anggap sebagai (x, y)), titik-titik tersebut pertama dilakukan proses *sorting* berdasarkan nilai x secara menaik, yang kemudian jika ada nilai x yang sama maka akan diurutkan berdasarkan nilai y secara menaik juga. Setelah itu, akan diambil nilai ekstrim paling kiri (elemen *array* pertama hasil *sorting*) dan nilai ekstrim paling kanan (elemen *array* terakhir hasil *sorting*). Dari kedua titik tersebut, akan dibuat suatu garis khayal yang berguna untuk memisahkan himpunan titik di atas garis dan

dibawah garis. Setelah itu, akan mulai dicari titik-titik pembentuk *convex hull* selain kedua titik ekstrim tadi, dimana pemanfaatan strategi *divide and conquer* dijalankan..

Pada algoritma ini, pencarian titik dibagi menjadi dua sisi, yaitu himpunan titik pembentuk atas garis dan pembentuk bawah garis. Kedua proses tersebut berjalan dengan proses yang sama. Pertama, dicari titik yang menghasilkan jarak paling jauh terhadap garis khayal yang sudah ditentukan sebelumnya. Setelah di dapat, titik tersebut akan masuk ke dalam himpunan solusi titik pembentuk *convex hull* sesuai bagian sisi yang sedang dicari (atas atau bawah). Lalu, garis ektrim kiri tadi akan dipasangkan dengan titik terjauh yang baru saja ditemukan untuk menjadi garis khayal baru, begitu juga dengan titik ekstrim kanan tadi juga dipasangkan dengan titik terjauh yang baru ditemukan. Garis-garis khayal baru ini akan kembali mencari titik-titik terjauh di luar garis khayal tersebut, terus-menerus secara rekursif hingga tidak ada lagi titik pada sisi terluar dari garis. Semua titik-titik terjauh yang ditemukan juga tidak lupa untuk dimasukkan ke dalam himpunan solusi titik sesuai sisi yang tadi sedang dicari. Untuk sisi bawah juga kronologi pencarian sama dengan pencarian pada sisi atas.

Setelah proses pencarian (rekursif) berhenti, himpunan solusi titik pada bagian atas dilakukan *sorting* menaik terhadap x, sedangkan untuk himpunan solusi titik pada bagian bawah dilakukan secara menurun terhadap x. Setelah proses itu selesai, akan dilakukan *merge* terhadap kedua himpunan solusi tersebut sehingga menghasilkan himpunan solusi secara utuh. Dari himpunan tersebut, akan dilakukan *plotting* dimana proses visualisasi dilakukan sehingga hasil *convex hull* dapat terlihat dengan jelas.

## II. Kode Program

Program dibuat dalam ekstensi .ipynb untuk mempermudah proses visualisasi dari berbagai dataset *scikit-learn* yang digunakan. Berikut merupakan *screenshot source code* yang telah diimplementasikan pada file hull.ipynb.

a. Proses Pembuatan Algoritma *Convex Hull*

```python
solutionPointUp = [] # deklarasi array global
solutionPointDown = [] # deklarasi array global

# fungsi mencari sisi titik dari garis acuan
def getSide(point1, point2, checkPoint):
    x1, y1 = point1 # alokasi koordinat x dan y pada point1
    x2, y2 = point2 # alokasi koordinat x dan y pada point2
    x3, y3 = checkPoint # alokasi koordinat x dan y pada point yang akan dicek
    a = x1 * y2
    b = x2 * y1
    c = x2 * y3
    d = x3 * y2
    e = x3 * y1
    f = x1 * y3
    det = a + e + c - d - b - f # rumus determinan untuk penentuan sisi
    if det < 0:
        return 'right'
    elif det > 0:
        return 'left'
```

Gambar 3 Fungsi Mencari Letak Titik terhadap Garis Khayal/Acuan

```python
# fungsi untuk mencari titik paling jauh dari garis acuan
def getFarthestPoint(point1, point2, partition):
    x1, y1 = point1 # alokasi koordinat x dan y pada point1
    x2, y2 = point2 # alokasi koordinat x dan y pada point2
    a = y2 - y1
    b = x1 - x2
    c = (x2*y1) - (x1*y2)
    farthest = -99
    getPointMax = None
    for coordinate in partition: # loop mencari titik terjauh
        x, y = coordinate # alokasi koordinat x dan y pada koordinat yang dicek
        funcDist = abs(a*x + b*y + c)/math.sqrt(a**2 + b**2) # rumus distance
        if farthest < funcDist:
            farthest = funcDist # pembaruan value terjauh
            getPointMax = coordinate # memasukkan koordinat titik terjauh
    return getPointMax
```

Gambar 4 Fungsi Mencari Titik dengan Jarak Terjauh dari Garis Khayal/Acuan

```python
# fungsi mencari titik-titik untuk hull
def searchHullPoint(partition, point1, point2, condition):
    # selama banyak elemen partisi tidak 0 maka akan mencari titik hull
    if len(partition) != 0:
        getPointMax = getFarthestPoint(point1, point2, partition)
        x, y = getPointMax # alokasi koordinat x dan y terhadap koordinat titik terjauh
        x1, y1 = point1 # alokasi koordinat x dan y terhadap koordinat point1
        x2, y2 = point2 # alokasi koordinat x dan y terhadap koordinat point2

        if condition == 'up': # pengecekan dilakukan pada bagian atas garis acuan
            solutionPointUp.append(getPointMax) # masukkan koordinat pada solusi atas
            partition1 = [] # inisialisasi partition1
            partition2 = [] # inisialisasi partition2

            # cek titik-titik untuk partisi berikutnya
            for coordinate in partition:
                # tidak memasukkan titik point1, point2, dan titik terjauh ke dalam partisi
                if (coordinate[0] == x and coordinate[1] == y) or (coordinate[0] == x1 and coordinate[1] == y1) or (coordinate[0] == x2 and
                coordinate[1] == y2):
                    pass
                else:
                    side1 = getSide(point1, getPointMax, coordinate) # cek sisi terhadap garis acuan
                    if side1 == 'right':
                        partition1.append(coordinate) # masukkan pada partition1
```

```python
            for coordinate in partition:
                # tidak memasukkan titik point1, point2, dan titik terjauh ke dalam partisi
                if (coordinate[0] == x and coordinate[1] == y) or (coordinate[0] == x1 and coordinate[1] == y1) or (coordinate[0] == x2 and
                coordinate[1] == y2):
                    pass
                else:
                    side2 = getSide(getPointMax, point2, coordinate) # cek sisi terhadap garis acuan
                    if side2 == 'right':
                        partition2.append(coordinate) # masukkan pada partition2
        else: # pengecekan dilakukan pada sisi bawah garis acuan
            solutionPointDown.append(getPointMax) #masukkan koordinat pada solusi bawah
            partition1 = [] # inisialisasi partition1
            partition2 = [] # inisialisasi partition2

            # cek titik-titik untuk partisi berikutnya
            for coordinate in partition:
                # tidak memasukkan titik point1, point2, dan titik terjauh ke dalam partisi
                if (coordinate[0] == x and coordinate[1] == y) or (coordinate[0] == x1 and coordinate[1] == y1) or (coordinate[0] == x2 and
                coordinate[1] == y2):
                    pass
                else:
                    side1 = getSide(point1, getPointMax, coordinate) # cek sisi terhadap garis acuan
                    if side1 == 'right':
                        partition1.append(coordinate) # masukkan pada partition1
```

```python
            for coordinate in partition:
                # tidak memasukkan titik point1, point2, dan titik terjauh ke dalam partisi
                if (coordinate[0] == x and coordinate[1] == y) or (coordinate[0] == x1 and coordinate[1] == y1) or (coordinate[0] == x2 and
                coordinate[1] == y2):
                    pass
                else:
                    side2 = getSide(getPointMax, point2, coordinate) # cek sisi terhadap garis acuan
                    if side2 == 'right':
                        partition2.append(coordinate) # masukkan pada partition2

        # rekursif pencarian titik hull dengan partisi baru dan titik-titik yang baru berdasarkan kondisi up/down yang sudah ditentukan
        searchHullPoint(partition1, point1, getPointMax, condition)
        searchHullPoint(partition2, getPointMax, point2, condition)
```

Gambar 5, 6, 7 Fungsi Mencari Titik-Titik Pembentuk *Convex Hull*

```python
def myConvexHull(listPoint):
    global solutionPointUp, solutionPointDown # variabel global

    sortingPoint = sorted(listPoint, key=lambda x:(x[0], x[1])) # sort data berdasarkan nilai x, lalu y
    sortingPoint = np.array(sortingPoint) # membuat array numpy

    extremeLeft = sortingPoint[0] # mengambil nilai extreme x paling rendah
    extremeRight = sortingPoint[-1] # mengambil nilai extreme x paling jauh

    solutionPointUp = [] # selalu set ini menjadi kosong agar solusi selalu direset tiap kali dipanggil
    solutionPointDown = [] # selalu set ini menjadi kosong agar solusi selalu direset tiap kali dipanggil
    onUp = [] # inisialisasi partisi awal untuk bagian atas
    onDown = [] # inisialisasi partisi awal untuk bagian bawah

    # pencarian titik untuk memisahkan bagian atas dan bawah terhadap garis acuan
    for coordinate in sortingPoint:
        coordinateSide = getSide(extremeLeft, extremeRight, coordinate) # cek sisi terhadap garis acuan
        if coordinateSide == 'left':
            onUp.append(coordinate) # masukkan pada onUp
        elif coordinateSide == 'right':
            onDown.append(coordinate) # masukkan pada onDown
```

```python
    solutionPointUp.append(extremeLeft) # masukkan koordinat extremeLeft pada solusi bagian atas
    solutionPointDown.append(extremeRight) # masukkan koordinat extremeRight pada solusi bagian bawah
    solutionPointDown.append(extremeLeft) # masukkan koordinat extremeLeft pada solusi bagian bawah agar di akhir dapat menyambung garis

    searchHullPoint(onDown, extremeLeft, extremeRight, "down") # cari titik hull di bagian bawah
    searchHullPoint(onUp, extremeRight, extremeLeft, "up") # cari titik hull di bagian atas

    solutionPointUp = sorted(solutionPointUp, key=lambda x: x[0]) # sort solusi menaik berdasarkan nilai x
    solutionPointDown = sorted(solutionPointDown, key=lambda x: x[0], reverse=True) # sort solusi menurun berdasarkan nilai x

    solutionFix = [] # inisialisasi solutionFix
    solutionFix = solutionPointUp # masukkan solusi atas
    solutionFix.extend(solutionPointDown) # extend dengan solusi bawah
    # solutionFix.append(extremeLeft) # tambahkan nilai titik extremeLeft agar dapat disambung

    return solutionFix
```

Gambar 8, 9 Fungsi Utama Pembentuk *Convex Hull*

b. Proses Visualisasi Program

Untuk bagian ini, hanya akan ditunjukan salah satu kode dalam proses visualisasi, karena modifikasi yang terjadi hanya pada penentuan atribut data.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Sepal Width vs Sepal Length')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,1]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 10 Proses Visualisai Dataset *load_iris* dengan Atribut *Sepal Widht vs Sepal Length*

## III. Uji Coba Program

1. Dataset *load_iris()*

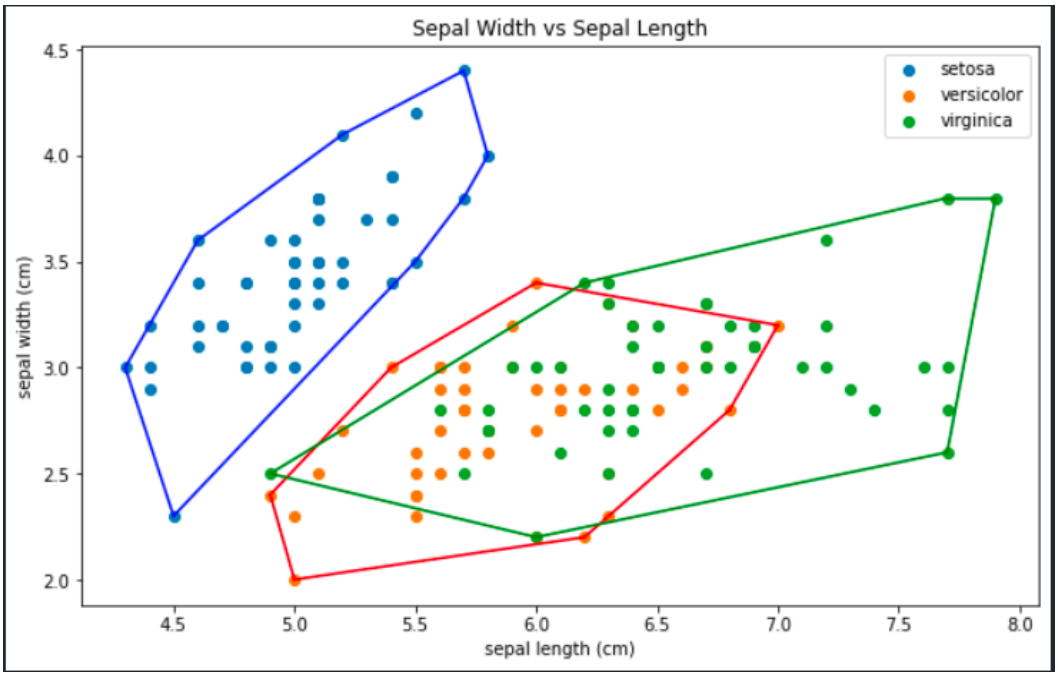
   *a. Sepal Widht vs Sepal Lenght*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Sepal Width vs Sepal Length')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,1]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 11 Kode Visualisasi *Sepal Widht vs Sepal Lenght*




Gambar 12 Hasil Visualisasi *Convex Hull Sepal Widht vs Sepal Lenght*

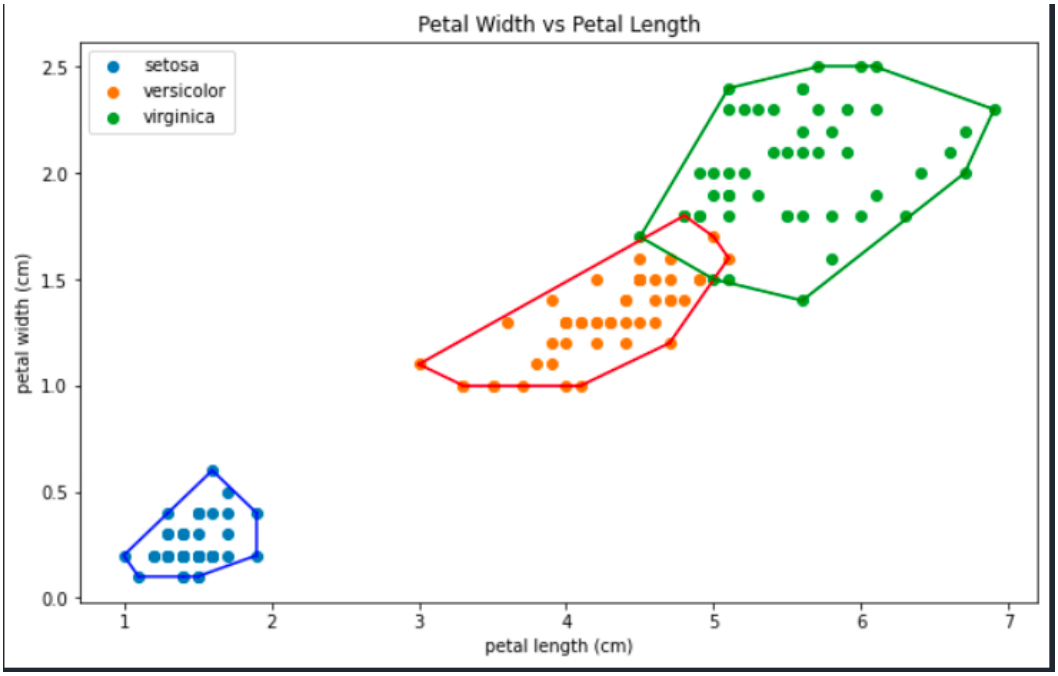
*b. Petal Widht vs Petal Lenght*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Petal Width vs Petal Length')
plt.xlabel(data.feature_names[2])
plt.ylabel(data.feature_names[3])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[2,3]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 13 Kode Visualisasi *Petal Widht vs Petal Lenght*




Gambar 14 Hasil Visualisasi *Convex Hull Petal Widht vs Petal Lenght*

2. Dataset *load_breast_cancer()*

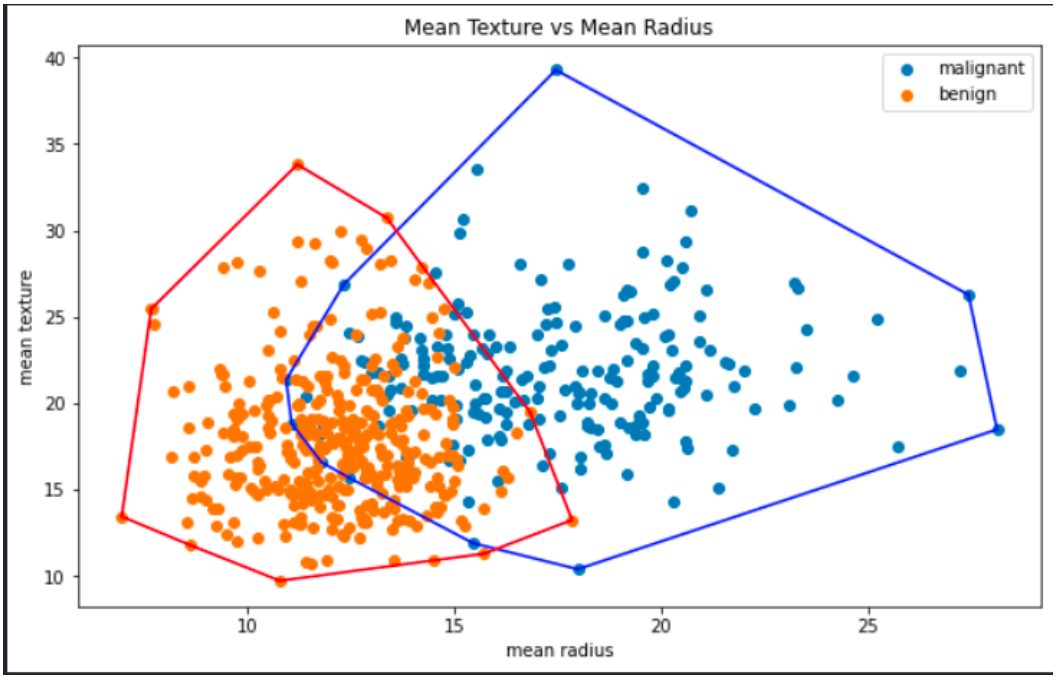
*a. Mean Texture vs Mean Radius*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Mean Texture vs Mean Radius')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,1]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 15 Kode Visualisasi *Mean Texture vs Mean Radius*




Gambar 16 Hasil Visualisasi *Convex Hull Mean Texture vs Mean Radius*

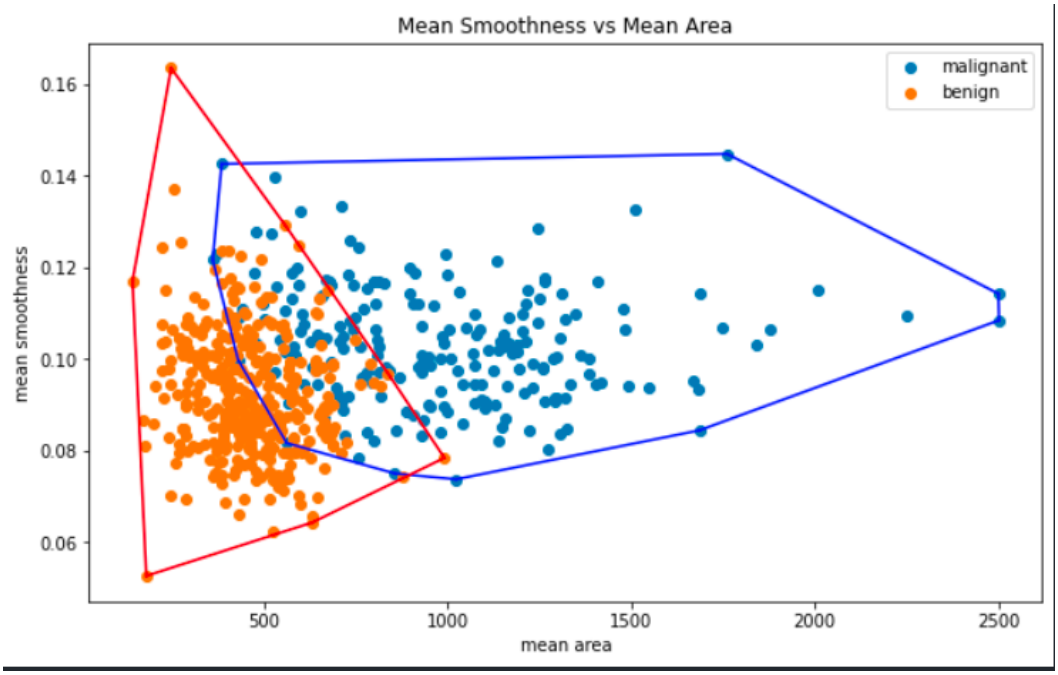
*b. Mean Smoothness vs Mean Area*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Mean Smoothness vs Mean Area')
plt.xlabel(data.feature_names[3])
plt.ylabel(data.feature_names[4])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[3,4]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 17 Kode Visualisasi *Mean Smoothness vs Mean Area*




Gambar 18 Hasil Visualisasi *Convex Hull Mean Smoothness vs Mean Area*

3. Dataset *load_wine()*

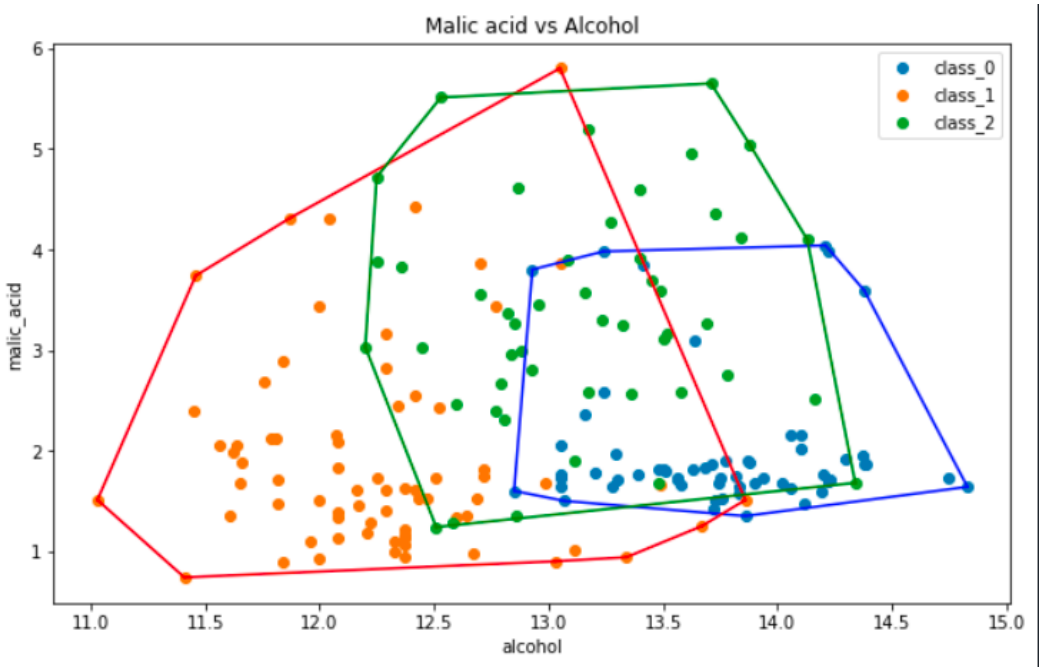
   *a. Malic Acid vs Alcohol*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Malic acid vs Alcohol')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[0,1]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 19 Kode Visualisasi *Malic Acid vs Alcohol*




Gambar 20 Hasil Visualisasi *Convex Hull Malic Acid vs Alcohol*

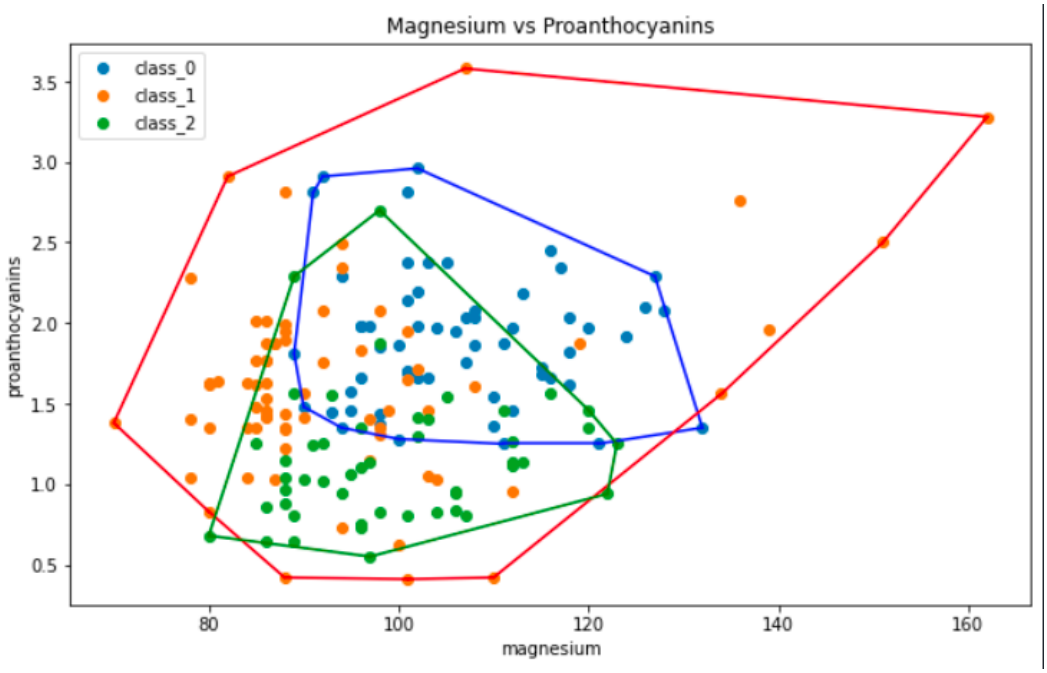
*b. Magnesium vs Proanthocyanins*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import datasets

data = datasets.load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = pd.DataFrame(data.target)
print(df.shape)
df.head()

#visualisasi hasil myConvexHull
plt.figure(figsize = (10, 6))
colors = ['b','r','g']
plt.title('Magnesium vs Proanthocyanins')
plt.xlabel(data.feature_names[4])
plt.ylabel(data.feature_names[8])
for i in range(len(data.target_names)):
    bucket = df[df['Target'] == i]
    bucket = bucket.iloc[:,[4,8]].values
    hull = myConvexHull(bucket) #pemanggilan convex hull
    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target_names[i])
    # plotting hasil dalam plot garis
    for j in range(len(hull)-1):
        plt.plot((hull[j][0], hull[j+1][0]), (hull[j][1], hull[j+1][1]), colors[i])
plt.legend()
```

Gambar 21 Kode Visualisasi *Magnesium vs Proanthocyanins*




Gambar 22 Hasil Visualisasi *Convex Hull Magnesium vs Proanthocyanins*

## IV. *Link* GitHub Program

https://github.com/irsyadazka/dncConvexHull

## V. Cek Keberhasilan Program

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