

Tugas Kecil 2 IF2211 Strategi Algoritma

Semester II tahun 2021/2022

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

Disusun oleh:

Rayhan Kinan Muhannad

13520065



**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2022**

## 1. Penjelasan Algoritma *Divide and Conquer*

Strategi *divide and conquer* merupakan suatu metode penyelesaian masalah dengan cara membagi masalah menjadi beberapa submasalah yang lebih kecil dan sederhana, kemudian menyelesaikan masing-masing submasalah tersebut, lalu menggabungkannya kembali menjadi algoritma utuh. Umumnya, strategi *divide and conquer* digunakan untuk menyelesaikan permasalahan yang dapat dibagi menjadi beberapa submasalah serupa yang lebih kecil dari permasalahan awalnya dan independen satu sama lain, kemudian solusi-solusi yang dihasilkan dari subpermasalahan tersebut ketika digabungkan dapat membuat solusi permasalahan yang lebih besar. Strategi *divide and conquer* biasanya digunakan pada permasalahan penggunaan struktur data *collection* yang memiliki properti rekursif, seperti *dynamic list* dan *tree*. Selain struktur data rekursif, strategi *divide and conquer* juga sering digunakan pada permasalahan geometri komputasional, seperti *quickhull* dan *closest pair of points*.

Di dalam pengimplementasiannya, strategi *divide and conquer* berkaitan erat dengan implementasi algoritma rekursif. Algoritma rekursif adalah algoritma yang terdiri atas dua bagian, yaitu *base case* dan *recurrence*. *Base case* dari suatu algoritma rekursif merupakan kondisi dimana rekursivitas algoritma tersebut berhenti. Biasanya, *base case* dari algoritma rekursif merupakan *initial value* atau kasus terkecil yang tidak dapat dibagi lebih lanjut. Kemudian *recurrence* dari algoritma rekursif merupakan kondisi dimana terdapat pemanggilan kembali algoritma rekursif tersebut (pemanggilan dalam kasus ini dapat diartikan sebagai pemanggilan fungsi, prosedur, *method*, ataupun ADT rekursif) dengan nilai masukan yang mendekati *base case*. Jika nilai masukan tidak berubah atau bahkan menjauhi *base case*, maka akan terjadi *infinite recursion* yang akan menyebabkan permasalahan tidak dapat diselesaikan.

Dalam pengimplementasiannya strategi *divide and conquer* dapat dibagi menjadi 3 bagian, yaitu *divide*, *conquer*, serta *merge*. Semua langkah tersebut akan dijelaskan lebih mendetil di bawah ini:

1. *Divide*: Membagi permasalahan menjadi beberapa subpermasalahan yang identik, berukuran hampir sama dengan satu sama lain, independen dengan subpersoalan lainnya, serta pembagiannya partisi subpermasalahannya konsisten. Pada pembagian permasalahan sederhana seperti algoritma *insertion sort* maupun *merge sort*, algoritma partisi yang dilakukan pada masukan tidaklah terlalu kompleks. Algoritma partisi tersebut dinamakan sebagai *easy split/hard join*. Tetapi, untuk

permasalahan pembagian lebih kompleks, seperti *selection sort*, *quicksort*, *quicksort*, maupun *closest pair of points*, algoritma partisi yang dilakukan untuk memecah permasalahan akan menjadi jauh lebih kompleks. Pada permasalahan tersebut, dibutuhkan algoritma yang telah terbukti paling efektif secara matematis untuk membagi permasalahan tersebut (sebagai contoh penggunaan algoritma partisi Hoare untuk menyelesaikan permasalahan *quick sort*).

2. *Conquer*: Menyelesaikan masing-masing subpersoalan dengan langsung melakukan perhitungan solusi bila subpersoalan tersebut termasuk *base case* dari algoritma rekursif atau membaginya kembali menjadi subpersoalan-subpersoalan baru yang diselesaikan secara rekursif bila subpersoalan masih terlalu besar atau termasuk sebagai *recurrence* dari algoritma rekursif. Pemrogram dapat menambahkan beberapa manipulasi algoritma untuk mengefisienkan rekursivitas algoritma. Umumnya, pemrogram dapat melakukan memoisasi atau pencatatan hasil dari hasil algoritma rekursif yang telah dilakukan terlebih dahulu oleh pemrogram. Contoh algoritma yang menggunakan strategi *conquer* tersebut adalah algoritma *binary exponentiation* dimana pemrogram hanya melakukan satu kali *recurrence* untuk dua subpersoalan yang berbeda. Selain itu, pemrogram juga dapat mengurangi jumlah subpersoalan yang diteruskan ke dalam *recurrence* dengan melakukan manipulasi aljabar. Contoh algoritma yang menggunakan strategi *conquer* tersebut adalah *Strassen's matrix multiplication* serta *Karatsuba algorithm*.
3. *Merge*: Menggabungkan seluruh solusi yang didapatkan dari algoritma rekursif dengan masukkan subpersoalan sehingga dapat membentuk solusi untuk menyelesaikan permasalahan. Seperti pada tahap *divide*, terdapat beberapa algoritma penggabungan yang tingkat kompleksitasnya berbeda-beda bergantung dari jenis algoritmanya itu sendiri. Terdapat pola menarik yang terlihat pada tahap ini, ketika suatu permasalahan memiliki strategi *divide and conquer* dengan algoritma partisi yang cukup sederhana, maka akan dipastikan bahwa algoritma penggabungan yang dimiliki oleh strategi tersebut akan cukup kompleks. Hal ini benar untuk algoritma-algoritma sederhana, seperti *merge sort* dan *insertion sort* dimana algoritma penggabungan solusi dari subpersoalannya tidaklah sederhana. Sebaliknya, ketika suatu permasalahan memiliki strategi *divide and conquer* dengan algoritma partisi yang kompleks, maka akan dipastikan bahwa algoritma penggabungan yang dimiliki oleh strategi tersebut akan cukup mudah. Hal ini benar

untuk algoritma-algoritma yang cukup kompleks, seperti *selection sort*, *quicksort*, *quicksort*, dan *closest pair of points*.

Seperti yang telah dijelaskan sebelumnya, strategi *divide and conquer* mengimplementasikan algoritma rekursif untuk menyelesaikan permasalahannya. Seperti yang telah dijelaskan sebelumnya, algoritma rekursif akan memanggil dirinya sendiri sebagai solusi dari masukkan yang diberikan. Oleh karena itu, muncul suatu permasalahan baru ketika penulis ingin melakukan analisis terhadap strategi *divide and conquer*. Dikarenakan rekursivitas algoritmanya, maka untuk kompleksitas waktu dari algoritma tersebut juga memiliki bentuk rekursif. Hal tersebut menyulitkan penulis untuk memformulasikan notasi *Big O* untuk menentukan seberapa efektif strategi *divide and conquer* yang telah dibuat. Oleh karena itu, penulis akan menggunakan Teorema Master untuk membantu penulis mencari kompleksitas waktu dari algoritma rekursif. Berikut ini merupakan penjelasan mengenai Teorema Master.

Misalkan  $T(n)$  adalah fungsi monoton menaik yang memenuhi relasi rekurens:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

Dengan asumsi bahwa  $n = b^k, k \in \{1, 2, 3, \dots\}$  serta  $a \geq 1, b \geq 2, c \geq 0$ , dan  $d \geq 0$ .

Maka, notasi *Big O* dari fungsi monoton  $T(n)$  adalah:

$$T(n) = \begin{cases} O(n^d), & a < b^d \\ O(n^d \cdot \log n), & a = b^d \\ O(n^{\log_b a}), & a > b^d \end{cases}$$

Permasalahan yang diangkat pada tugas kecil ini adalah memvisualisasikan data yang terdapat pada suatu dataset *classification* dan menganalisis apakah dataset tersebut dapat dipisahkan secara linear (*linearly separable*) atau tidak dengan menggunakan *convex hull*. Dataset yang dapat dianalisis menggunakan metode ini hanyalah dataset yang terdiri atas data-data numerik. Hal tersebut dikarenakan dibutuhkan teknik-teknik pengolahan data lanjut untuk menganalisis dataset yang mempunyai atribut kategorikal.

*Convex hull* merupakan himpunan titik pada suatu bidang planar (*convex*) yang dimana jika sembarang dua titik pada himpunan tersebut dihubungkan oleh suatu garis, maka seluruh segmen garis tersebut terdapat di dalam himpunan tersebut. Dengan kata lain jika terdapat himpunan titik pada suatu bidang planar, maka *convex hull* merupakan poligon planar terkecil yang mencakup seluruh himpunan titik tersebut. Terdapat banyak aplikasi *convex hull*, seperti dalam kasus ini menjadi suatu pengujian apakah

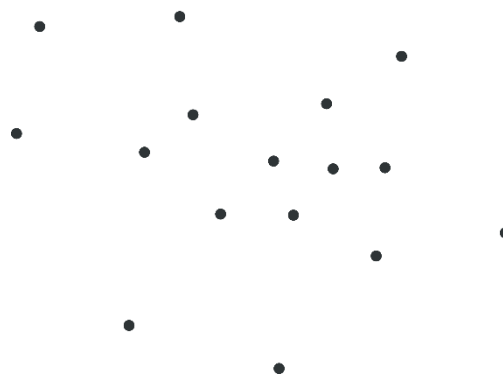
suatu dataset dapat dipisahkan secara linear atau tidak. Jika suatu dataset dapat dipisahkan secara linear, maka pemrogram dapat menggunakan model *machine learning* dengan tipe *linear classification* untuk memodelkan data tersebut. Untuk menentukan apakah suatu dataset *linearly separable* menggunakan visualisasi *convex hull* adalah dengan melakukan visualisasi untuk setiap pasangan atribut pada dataset tersebut. Ketika *convex hull* yang terbentuk untuk kategori atribut target yang berbeda terjadi *overlap* antara satu sama lain, maka dataset tersebut tidak bisa dipisahkan secara linear. Dataset dikatakan *linearly separable* ketika semua *convex hull* yang terbentuk dari hasil kombinasi dua atribut datasetnya tidak terdapat *overlap* sama sekali.

<Penjelasan solusi algoritma menggunakan divide and conquer (gunakan gambar)>

Terdapat beberapa algoritma yang dapat digunakan untuk mencari *convex hull*, diantaranya adalah *Jarvis's Algorithm* (menggunakan strategi *brute force*), *Graham Scan* (menggunakan strategi *sort and select*), serta *quickhull* (menggunakan strategi *divide and conquer*). Pada tugas kecil ini, penulis mengambil algoritma *quickhull* dikarenakan tingkat kompleksitasnya yang tidak terlalu rumit serta berkorelasi dengan materi kelas.

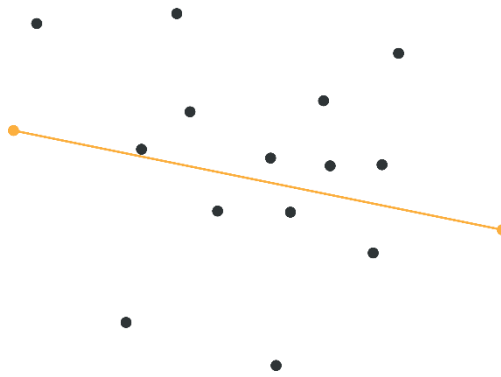
Langkah-langkah yang dilakukan pada algoritma *quickhull* tidak terlalu jauh perbedaannya dengan algoritma *quicksort*. Berikut ini merupakan langkah-langkah dari algoritma *quickhull* untuk mencari *convex hull* pada bidang planar 2-D:

1. Carilah dua titik pada bidang dengan koordinat sumbu x terkecil dan koordinat x terbesar (jika terdapat dua titik dengan koordinat x yang sama, maka akan dibandingkan menggunakan koordinat sumbu y).

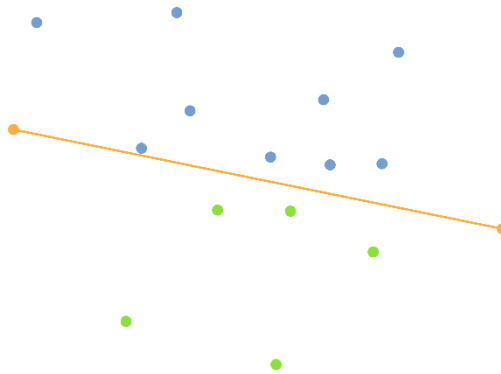


Gambar 1.1 Himpunan titik pada bidang planar

2. Hubungkan kedua titik ekstremum tersebut menjadi sebuah garis. Hal tersebut akan membuat dua partisi pada himpunan titik, yaitu himpunan titik diatas garis dan himpunan titik dibawah garis.

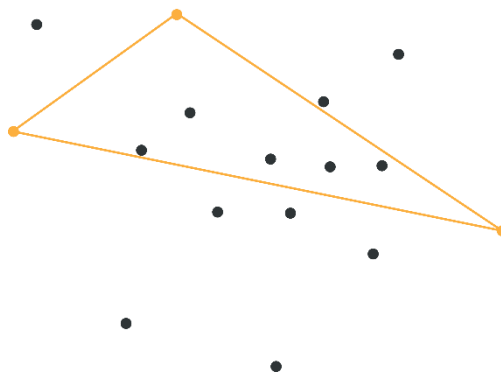


Gambar 1.2 Garis partisi



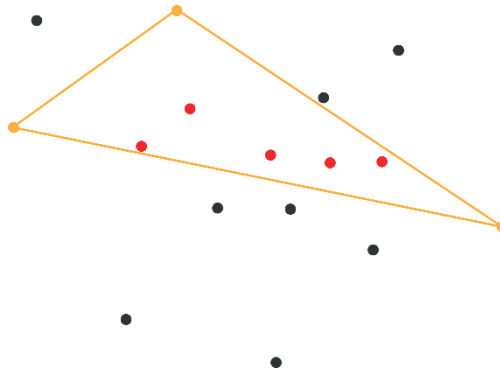
Gambar 1.3 Partisi himpunan titik

3. Pada setiap himpunan partisi, carilah sebuah titik ekstrem dimana titik tersebut memiliki jarak euclidean terjauh dengan garis partisi yang terbentuk. Tariklah garis dari kedua ujung garis partisi menuju titik ekstrem tersebut sehingga terbentuklah sebuah area partisi segitiga.



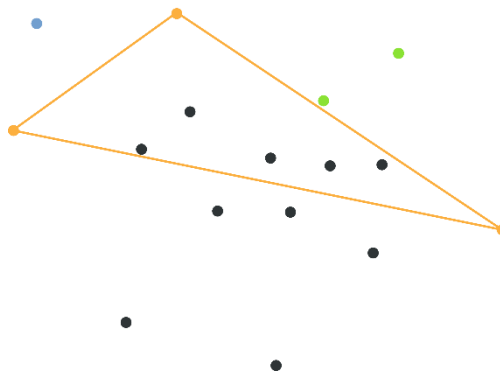
Gambar 1.4 Area segitiga yang terbentuk

4. Himpunan titik yang terdapat di dalam area partisi segitiga dapat dihiraukan. Himpunan titik tersebut diasumsikan sudah termasuk di dalam *convex hull* yang terbentuk.



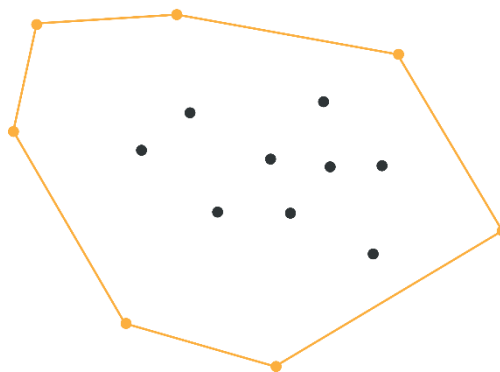
Gambar 1.5 Himpunan titik di dalam segitiga dihiraukan

5. Ulangi langkah 3 dan 4 dengan menggunakan titik pada luar segitiga sebagai himpunan titik serta garis segitiga yang baru terbentuk sebagai garis partisi.



Gambar 1.6 Langkah 3 dan 4 dilakukan kembali

6. Rekursi langkah 3 dan 4 secara terus menerus hingga himpunan titik yang tidak terdapat di dalam *convex hull* merupakan himpunan kosong. Garis-garis yang terbentuk oleh algoritma rekursif tersebut merupakan sisi-sisi dari *convex hull*.



Gambar 1.7 *Convex hull* yang terbentuk

Dikarenakan struktur algoritma *quickhull* memiliki bentuk dan langkah-langkah yang serupa dengan algoritma *quicksort*, maka kompleksitas waktu *quickhull* juga akan serupa dengan algoritma *quicksort*. Untuk kasus terbaik, diasumsikan untuk setiap partisi yang dilakukan oleh algoritma, himpunan titik tersebut terbagi secara merata (besar himpunan partisi atas sama dengan himpunan partisi bawah). Pada setiap partisi

tersebut, dilakukan pengecekan jarak euclidean setiap titik ke garis partisi untuk mencari titik dengan jarak euclidean terbesar. Oleh karena itu untuk kasus terbaik, kompleksitas waktunya adalah sebagai berikut.

$$T(n) = \begin{cases} a, & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n, & n > 1 \end{cases}$$

Dengan menggunakan Teorema Master, didapatkan bahwa kompleksitas waktu dari kasus terbaik algoritma *quickhull* adalah sebagai berikut.

$$T(n) = O(n \cdot \log n)$$

Untuk kasus terburuk, diasumsikan untuk setiap partisi yang dilakukan oleh algoritma, hanya terdapat pengurangan dua titik pada setiap partisinya (jumlah minimum titik yang tidak masuk partisi adalah dua titik, yaitu titik yang menjadi titik awal dan titik akhir dari garis partisi). Oleh karena itu untuk kasus terburuk, kompleksitas waktunya adalah sebagai berikut.

$$T(n) = \begin{cases} a, & n = 1 \\ T(n - 2) + c \cdot n, & n > 1 \end{cases}$$

Dengan menggunakan Teorema Master, didapatkan bahwa kompleksitas waktu dari kasus terburuk algoritma *quickhull* adalah sebagai berikut.

$$T(n) = O(n^2)$$

Untuk kasus rata-rata, diasumsikan bahwa partisi yang dilakukan oleh algoritma, terjadi pembagian yang cukup merata. Meskipun tidak terbagi menjadi dua bagian secara merata, tetapi setidaknya pembagian tersebut membentuk bagian yang cukup serupa. Oleh karena itu, dengan menggunakan Teorema Master, didapatkan bahwa kompleksitas waktu dari kasus rata-rata algoritma *quickhull* adalah sebagai berikut.

$$T(n) = O(n \cdot \log n)$$



## 2. Source Code dengan Bahasa Python

### a. visualize.py

```
import pandas as pd
from argparse import ArgumentParser
from sklearn import datasets
from myConvexHull import plot_convex_hull

# BATASAN: HANYA TERDEFINISI PADA DATASET CLASSIFICATION
# BATASAN: SEMUA DATA ATRIBUT ARGUMEN HARUS DALAM BENTUK NUMERIK

# MAIN PROGRAM
if __name__ == "__main__":
    my_parser = ArgumentParser(description="Masukkan nama database serta atribut yang ingin divisualisasi")

    my_parser.add_argument("path", metavar="path", type=str, help="relative path menuju file database")
    my_parser.add_argument("first_argument", metavar="first_argument", type=str, help="atribut pertama yang ingin dievaluasi")
    my_parser.add_argument("second_argument", metavar="second_argument", type=str, help="atribut kedua yang ingin dievaluasi")
    my_parser.add_argument("-t", "--target", type=str, help="atribut yang dijadikan target", default="")
    my_parser.add_argument("-c", "--class-name", type=str, nargs="+", help="nama kelas pada atribut target", default=[])

    args = my_parser.parse_args()

    if args.path == "iris":
        data = datasets.load_iris()
        df = pd.DataFrame(data.data, columns=data.feature_names)
        df["Target"] = pd.DataFrame(data.target)

        try:
            plot_convex_hull(df=df, first_argument=args.first_argument, second_argument=args.second_argument, target="Target", class_name=["Iris-Setosa", "Iris-Versicolour", "Iris-Virginica"])
```

```

except:
    print("Argumen tidak valid!")

elif args.path == "wine":
    data = datasets.load_wine()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df["Target"] = pd.DataFrame(data.target)

    try:
        plot_convex_hull(df=df, first_argument=args.first_argument,
second_argument=args.second_argument, target="Target", class_name=["class_0",
"class_1", "class_2"])
    except:
        print("Argumen tidak valid!")

elif args.path == "breast_cancer":
    data = datasets.load_breast_cancer()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df["Target"] = pd.DataFrame(data.target)

    try:
        plot_convex_hull(df=df, first_argument=args.first_argument,
second_argument=args.second_argument, target="Target", class_name=["WDBC-
Malignant", "WDBC-Benign"])
    except:
        print("Argumen tidak valid!")

else:
    try:
        df = pd.read_csv(args.path)

    except:
        print(f"Database dengan relative path {args.path} tidak ada!")

    try:
        plot_convex_hull(df=df, first_argument=args.first_argument,
second_argument=args.second_argument, target=args.target,
class_name=args.class_name)

```

```
except:  
    print("Argumen tidak valid!")
```

## b. myConvexHull.py

```
import numpy as np
import matplotlib.pyplot as plt

# CLASS
class Color():
    COLORS = ["blue", "orange", "green", "red", "purple", "brown", "pink",
"gray", "olive", "cyan"]
    i = 0

    def get_color():
        new_color = Color.COLORS[Color.i % len(Color.COLORS)]
        Color.i += 1

        return new_color

class ConvexHull():
    data = []

    def __init__(self, *args, **kwargs):
        if "points" in kwargs.keys():
            ConvexHull.data = kwargs["points"].tolist()

            self.indexData = [i for i in range(len(ConvexHull.data))]
            self.line = []

        elif len(args) > 0:
            ConvexHull.data = args[0].tolist()

            self.indexData = [i for i in range(len(ConvexHull.data))]
            self.line = []

        else:
            self.indexData = []
            self.line = []

    def distance(self, point):
        p1 = np.array(ConvexHull.data[self.line[0][0]])
```

```

p2 = np.array(ConvexHull.data[self.line[0][1]])
p3 = np.array(ConvexHull.data[point])

return np.cross(p2 - p1, p3 - p1) / np.linalg.norm(p2 - p1)

def find_extremum(self):
    minimum = self.indexData[0]
    maximum = self.indexData[0]

    if self.line == []:
        for i in self.indexData:
            if ConvexHull.data[i][0] < ConvexHull.data[minimum][0]:
                minimum = i
            if ConvexHull.data[i][0] == ConvexHull.data[minimum][0] and
ConvexHull.data[i][1] < ConvexHull.data[minimum][1]:
                minimum = i
            if ConvexHull.data[i][0] > ConvexHull.data[maximum][0]:
                maximum = i
            if ConvexHull.data[i][0] == ConvexHull.data[maximum][0] and
ConvexHull.data[i][1] > ConvexHull.data[maximum][1]:
                maximum = i

        else:
            for i in self.indexData:
                if self.distance(i) < self.distance(minimum):
                    minimum = i
                if self.distance(i) > self.distance(maximum):
                    maximum = i

    return (minimum, maximum)

def partition(self):
    left = ConvexHull()
    right = ConvexHull()

    if self.line == []:
        minimum, maximum = self.find_extremum()
        self.line = [[minimum, maximum]]

```

```

        for i in self.indexData:
            if self.distance(i) < 0:
                left.indexData.append(i)
            if self.distance(i) > 0:
                right.indexData.append(i)

        left.line = [[minimum, maximum]]
        right.line = [[minimum, maximum]]

    else:
        pMin, pMax = self.find_extremum()

        if self.distance(pMin) > 0: # partisi atas
            left.line = [[self.line[0][0], pMax]]
            right.line = [[pMax, self.line[0][1]]]

            for i in self.indexData:
                if left.distance(i) > 0:
                    left.indexData.append(i)
                if right.distance(i) > 0:
                    right.indexData.append(i)

            if self.distance(pMax) < 0: # partisi bawah
                left.line = [[self.line[0][0], pMin]]
                right.line = [[pMin, self.line[0][1]]]

                for i in self.indexData:
                    if left.distance(i) < 0:
                        left.indexData.append(i)
                    if right.distance(i) < 0:
                        right.indexData.append(i)

        return (left, right)

def merge(self, left, right):
    self.line = left.line + right.line
    self.indexData = list(set(left.indexData) | set(right.indexData))

```

```

def divide_and_conquer(self):
    if self.indexData != []:
        left, right = self.partition()

        left.divide_and_conquer()
        right.divide_and_conquer()

        self.merge(left, right)

def create_convex(self):
    self.divide_and_conquer()

    return np.array(self.line)

# FUNCTION
def plot_convex_hull(df, first_argument, second_argument, target, class_name):
    temp_df = df.copy()

    if target != "" and class_name != []:
        # Mengubah data atribut target menjadi numerik
        temp_df[target].replace(class_name, [i for i in
range(len(class_name))], inplace=True)

        plt.figure(figsize=(10, 6), num="Convex Hull")
        plt.title(f"{first_argument} vs {second_argument}")

        plt.xlabel(first_argument)
        plt.ylabel(second_argument)

        for target_value in temp_df[target].unique():
            bucket = temp_df[temp_df[target] == target_value]

            hull = ConvexHull(bucket[[first_argument,
second_argument]].values)

            plt.scatter(bucket[first_argument].values,
bucket[second_argument].values, label=class_name[target_value])

```

```

        color = Color.get_color()

        for simplex in hull.create_convex(): # hull.simplices adalah
numpy.ndarray of numpy.ndarray yang berisi dua index dari titik-titik terluar
data yang jika dihubungkan membentuk sisi pada convex hull
            # plt.plot([x1, x2], [y1, y2]) akan membentuk garis dari (x1,
y1) ke (x2, y2)

            plt.plot(bucket[[first_argument,
second_argument]].values[simplex, 0], bucket[[first_argument,
second_argument]].values[simplex, 1], color)

plt.legend()
plt.show()

elif target != "" and class_name == []:
    # Mengubah data atribut target menjadi numerik
    temp_df[target].replace(class_name, [i for i in
range(len(class_name))], inplace=True)

plt.figure(figsize=(10, 6), num="Convex Hull")
plt.title(f"{first_argument} vs {second_argument}")

plt.xlabel(first_argument)
plt.ylabel(second_argument)

for target_value in temp_df[target].unique():
    bucket = temp_df[temp_df[target] == target_value]

    hull = ConvexHull(bucket[[first_argument,
second_argument]].values)

    plt.scatter(bucket[first_argument].values,
bucket[second_argument].values, label=target_value)
    color = Color.get_color()

```



```

        for simplex in hull.create_convex(): # hull.simplices adalah
numpy.ndarray of numpy.ndarray yang berisi dua index dari titik-titik terluar
data yang jika dihubungkan membentuk sisi pada convex hull
            # plt.plot([x1, x2], [y1, y2]) akan membentuk garis dari (x1,
y1) ke (x2, y2)

            plt.plot(bucket[[first_argument,
second_argument]].values[simplex, 0], bucket[[first_argument,
second_argument]].values[simplex, 1], color)

plt.legend()
plt.show()

else:
    plt.figure(figsize=(10, 6), num="Convex Hull")
    plt.title(f"{first_argument} vs {second_argument}")

    plt.xlabel(first_argument)
    plt.ylabel(second_argument)

    hull = ConvexHull(temp_df[[first_argument, second_argument]].values)

    plt.scatter(temp_df[first_argument].values,
temp_df[second_argument].values)
    color = Color.get_color()

    for simplex in hull.create_convex(): # hull.simplices adalah
numpy.ndarray of numpy.ndarray yang berisi dua index dari titik-titik terluar
data yang jika dihubungkan membentuk sisi pada convex hull
        # plt.plot([x1, x2], [y1, y2]) akan membentuk garis dari (x1, y1)
ke (x2, y2)

        plt.plot(temp_df[[first_argument,
second_argument]].values[simplex, 0], temp_df[[first_argument,
second_argument]].values[simplex, 1], color)

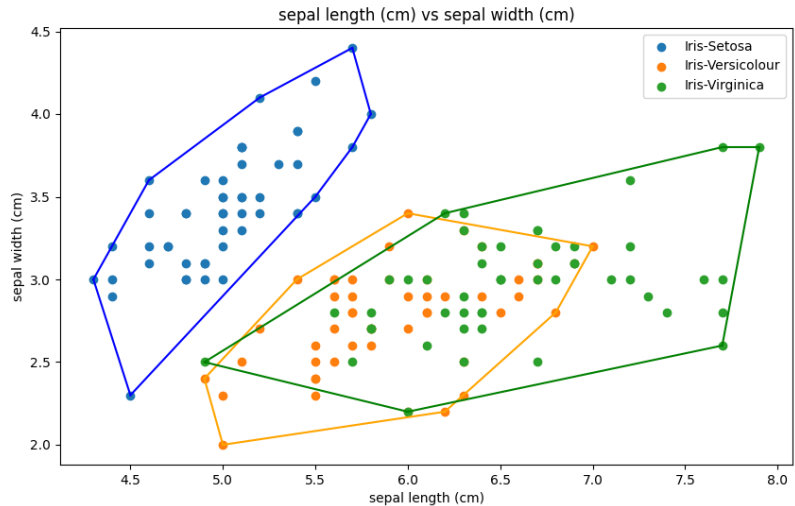
plt.show()

```

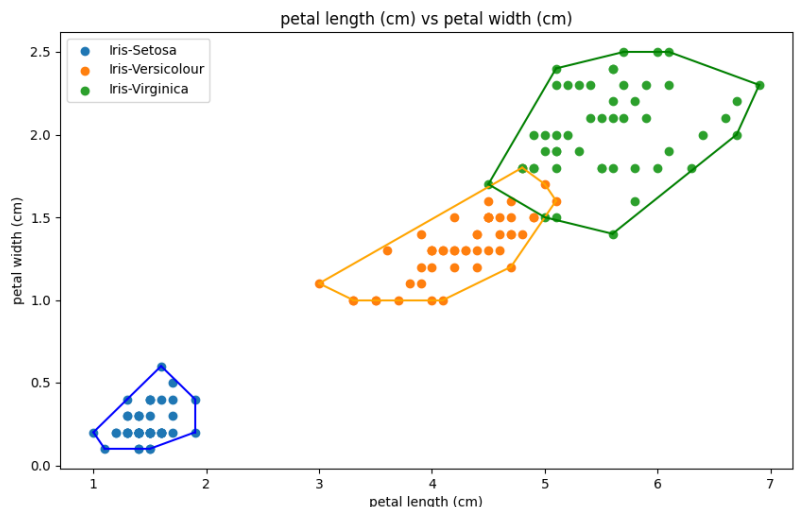
### 3. Screenshot dari Input dan Output Program dengan menggunakan Dataset

#### a. Iris Dataset

##### i. Sepal Length (cm) vs Sepal Width (cm)

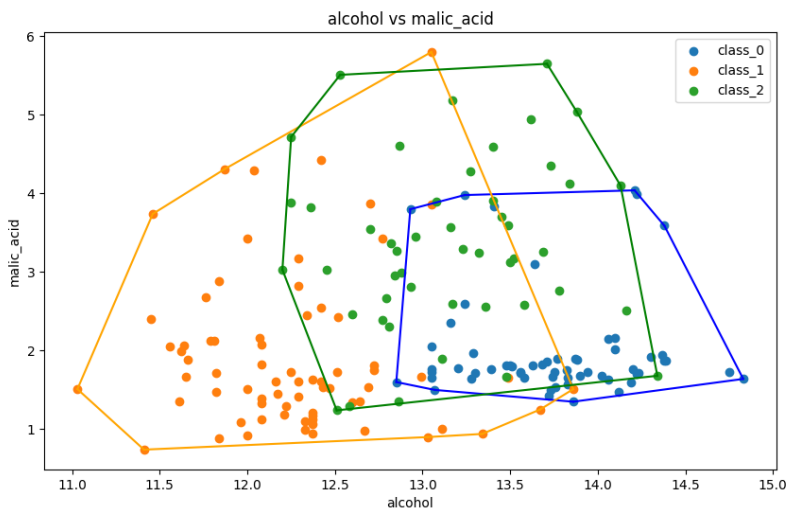
	Screenshot
Input	<pre>PS C:\Users\rayha\Downloads\App\Stima\convex-hull-visualizer\src&gt; py visualize.py "iris" "sepal length (cm)" "sepal width (cm)"</pre>
Output	

##### ii. Petal Length (cm) vs Petal Width (cm)

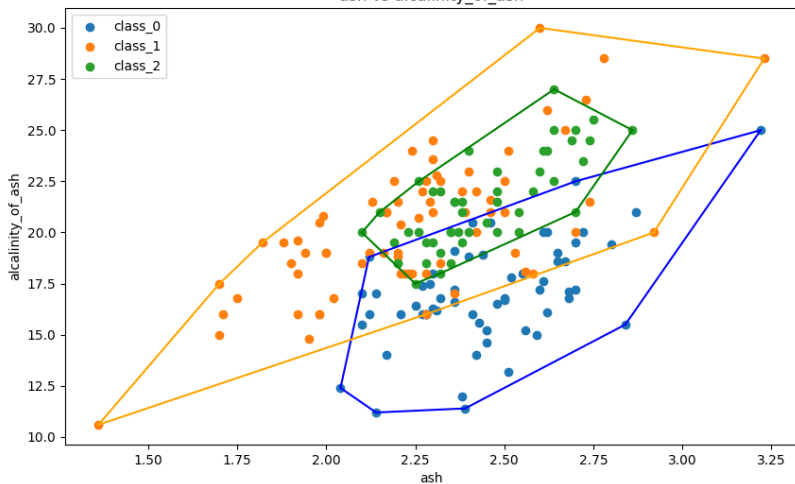
	Screenshot
Input	<pre>PS C:\Users\rayha\Downloads\App\Stima\convex-hull-visualizer\src&gt; py visualize.py "iris" "petal length (cm)" "petal width (cm)"</pre>
Output	

#### b. Wine Dataset

##### i. Alcohol vs Malic Acid

	<i>Screenshot</i>
Input	PS C:\Users\rayha\Downloads\App\Stima\convex-hull-visualizer\src> py visualize.py "wine" "alcohol" "malic_acid"
Output	

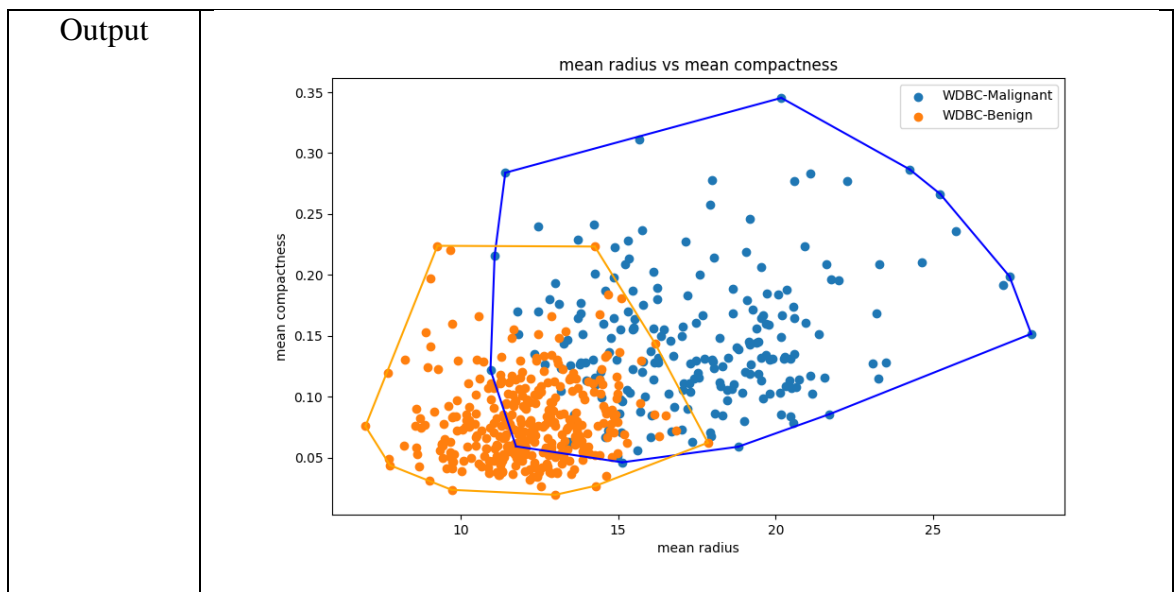
## ii. Ash vs Alcalinity of Ash

	<i>Screenshot</i>
Input	PS C:\Users\rayha\Downloads\App\Stima\convex-hull-visualizer\src> py visualize.py "wine" "ash" "alcalinity_of_ash"
Output	

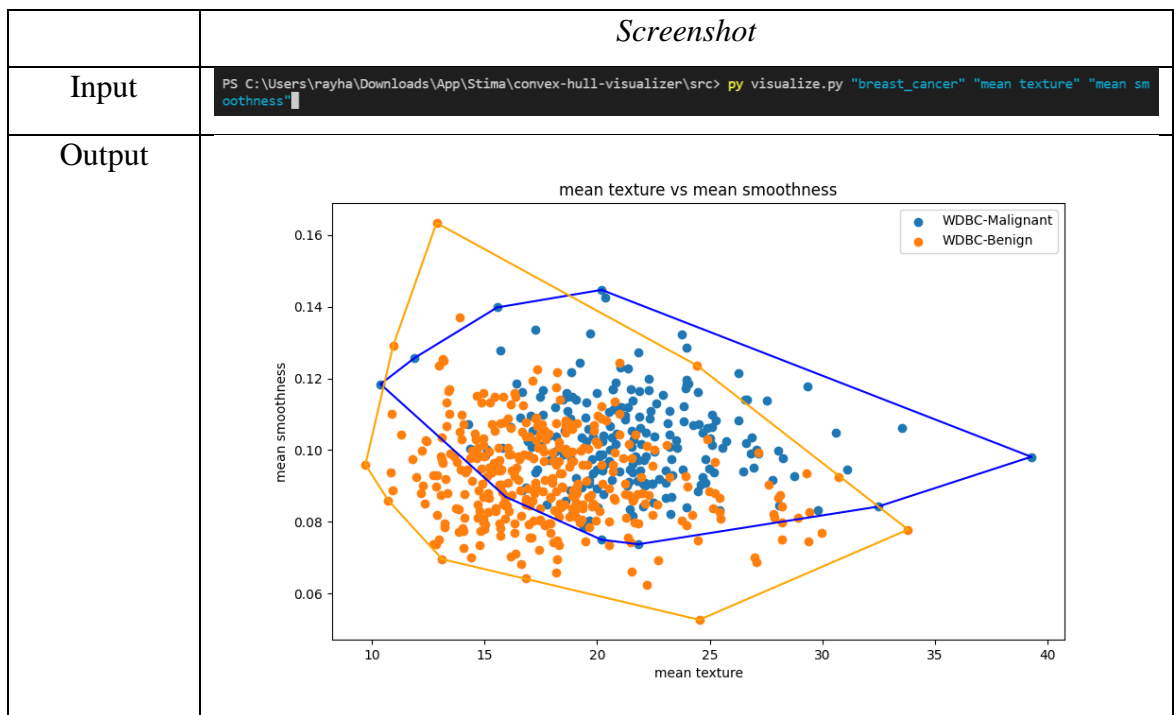
## c. Breast Cancer Dataset

### i. Mean Radius vs Mean Compactness

	<i>Screenshot</i>
Input	PS C:\Users\rayha\Downloads\App\Stima\convex-hull-visualizer\src> py visualize.py "breast_cancer" "mean radius" "mean compactness"



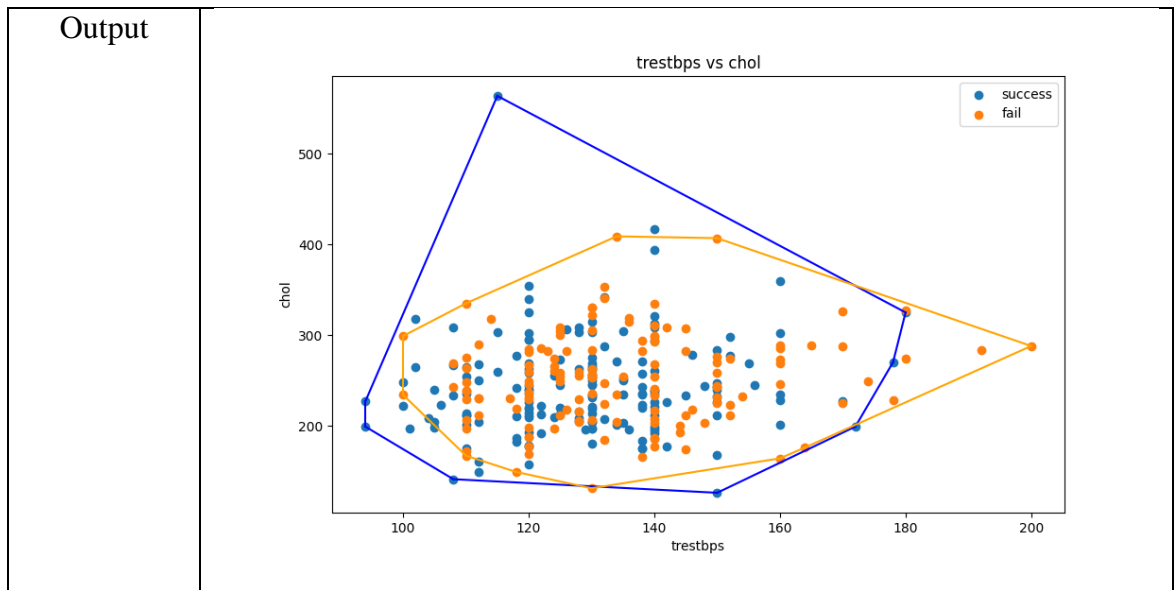
## ii. Mean Texture vs Mean Smoothness



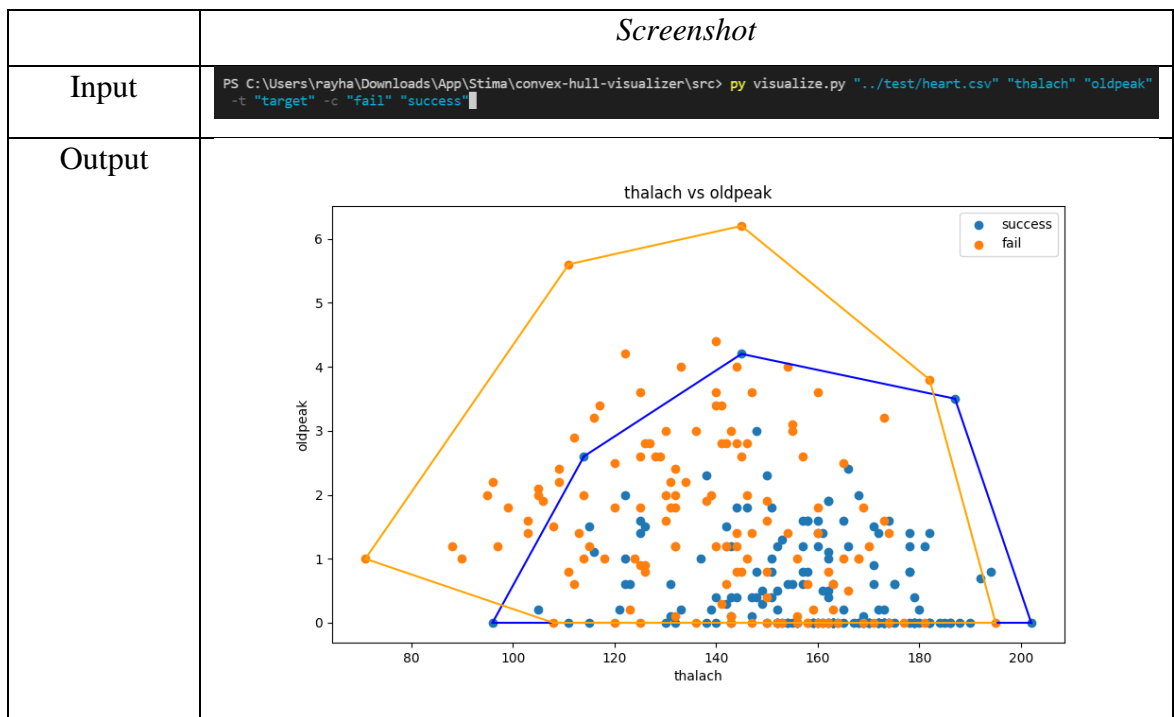
## d. Heart Disease Dataset

### i. Resting Blood Pressure (mmHg) vs Serum Cholesterol (mg/dl)





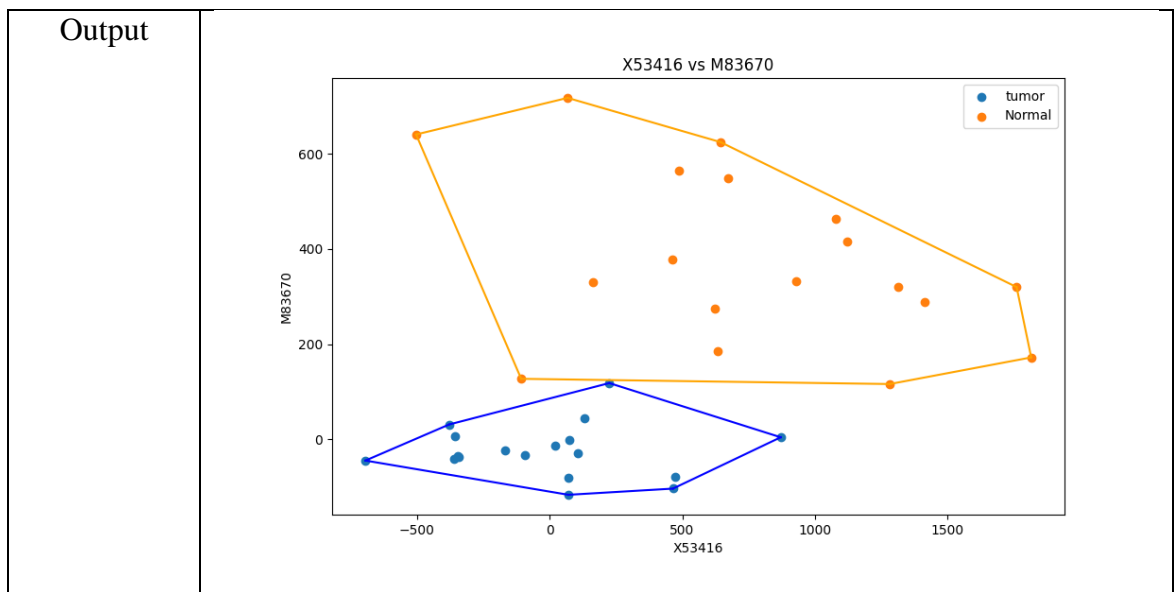
**ii. Maximum Heart Rate (BPM) vs ST Depression Induced By Exercise Relative to Rest**



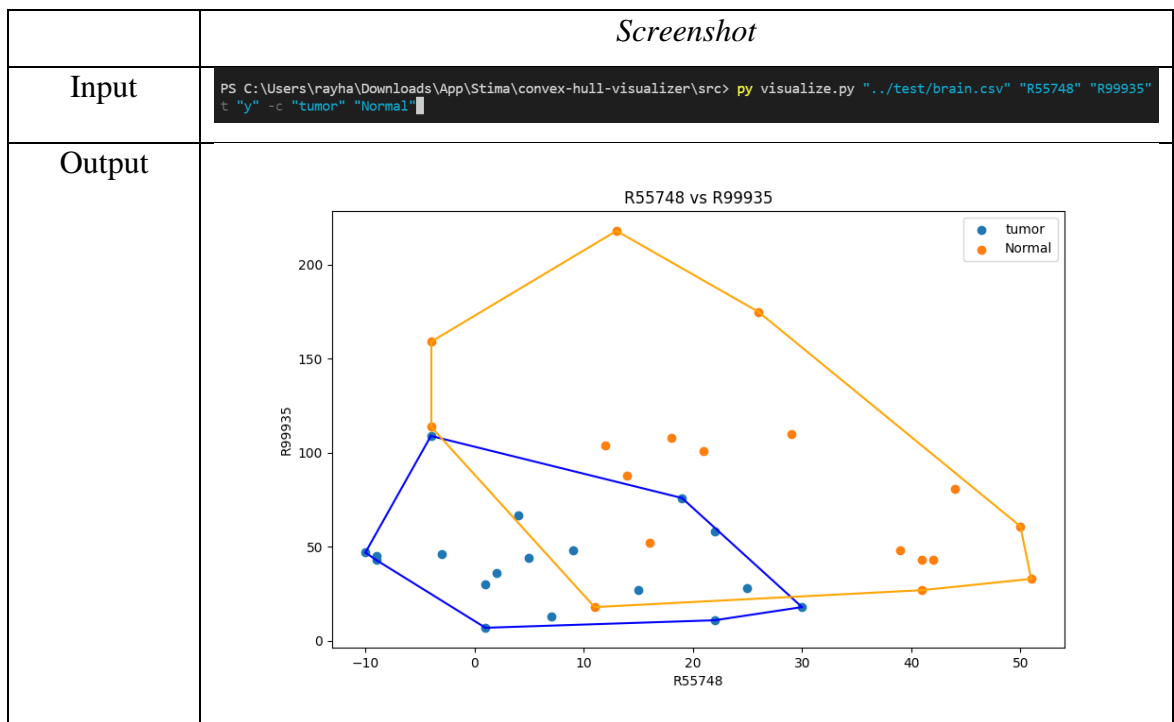
**e. Brain Tumor Dataset**

**i. X53416 vs M83670**





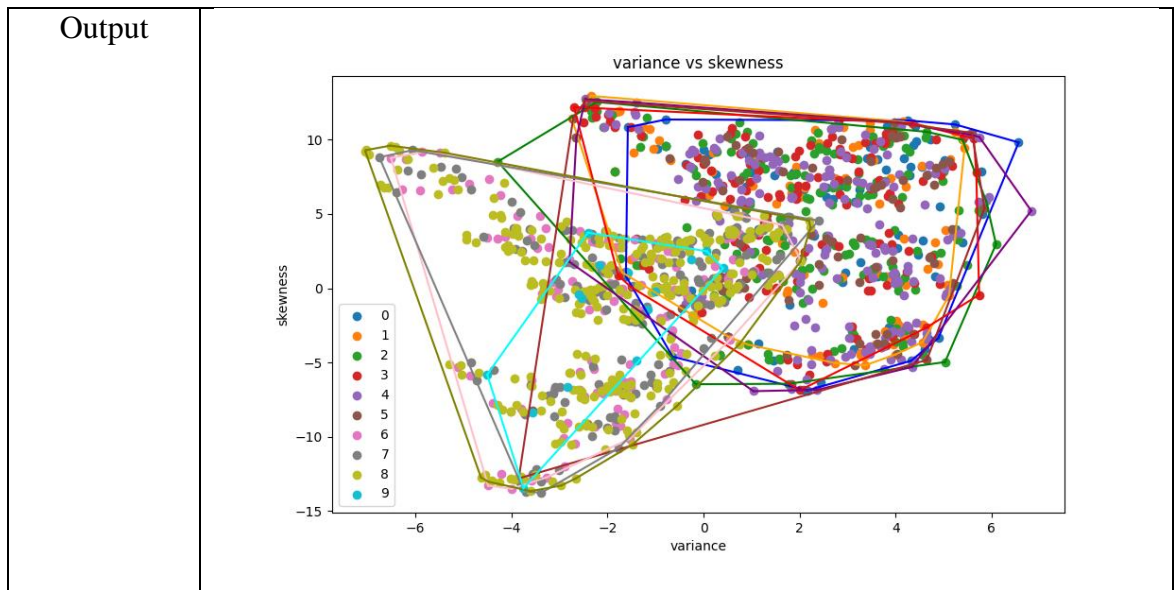
## ii. R55748 vs R99935



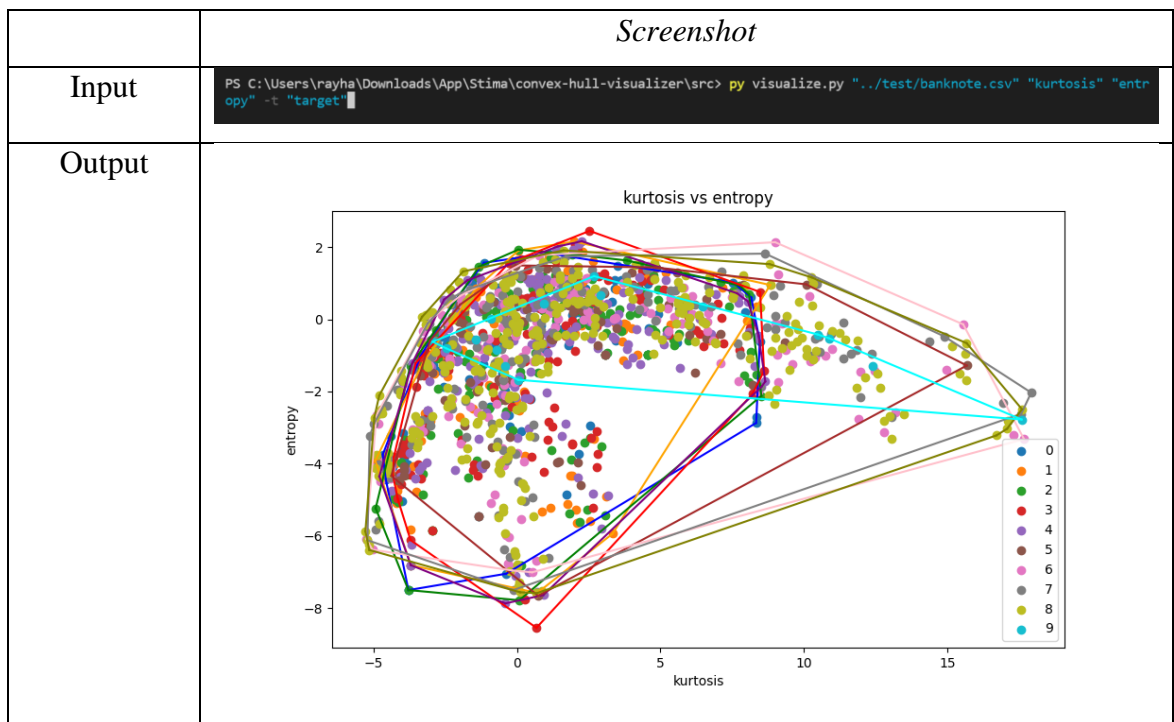
## f. Banknote Authentication Dataset

### i. Variance vs Skewness





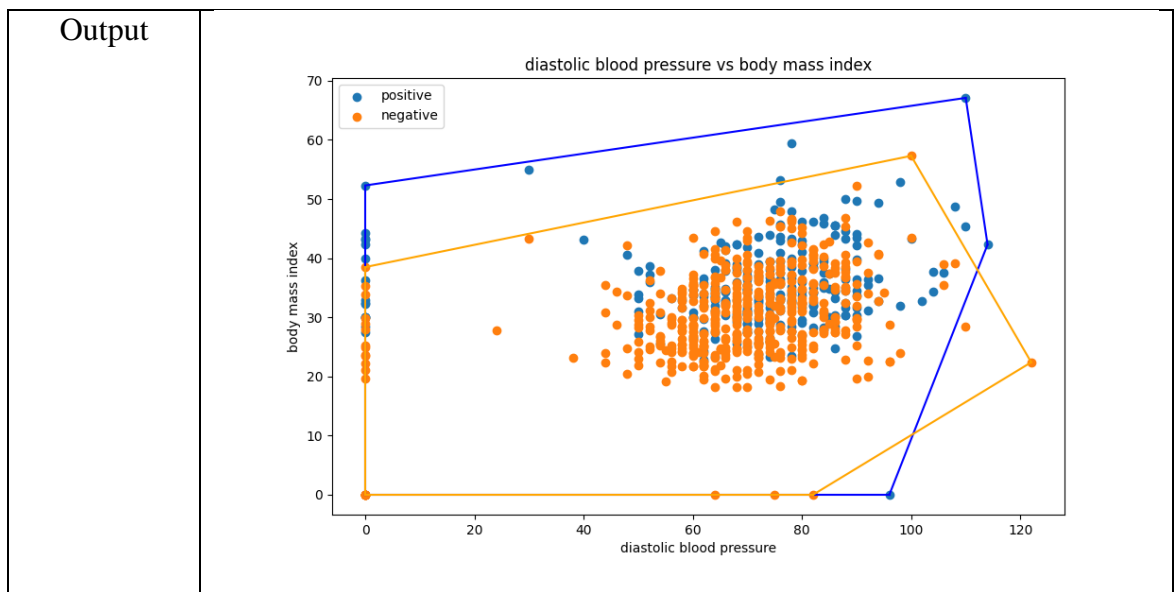
## ii. Kurtosis vs Entropy



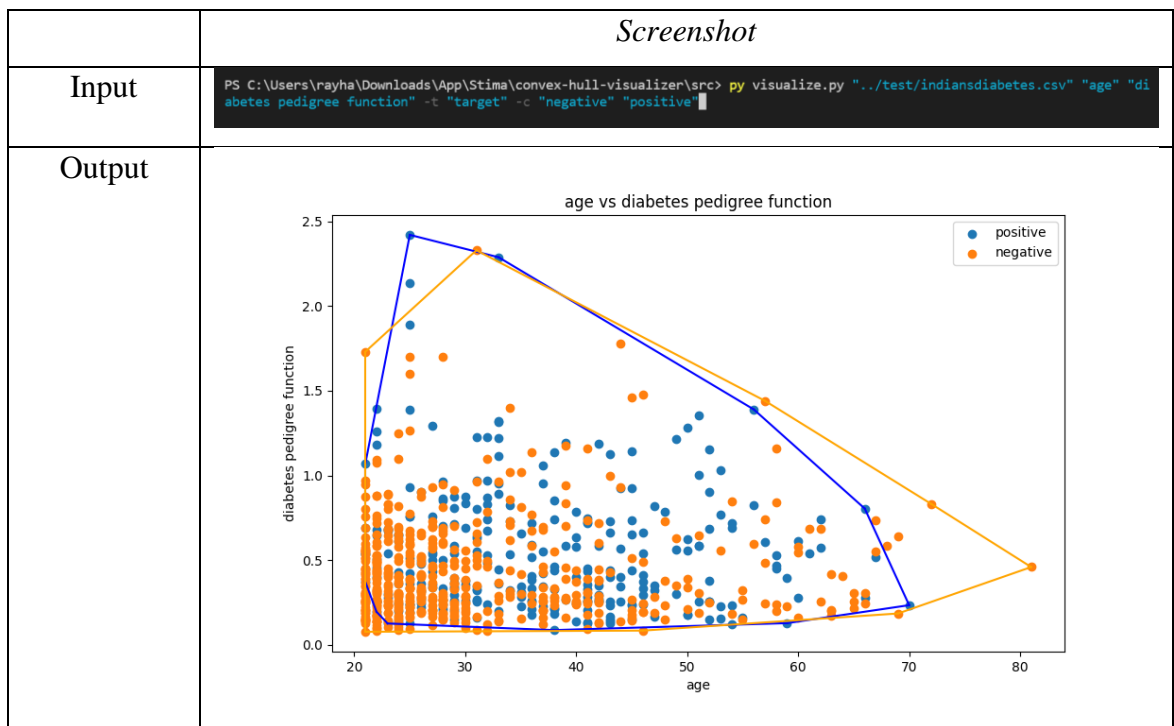
## g. Diabetes Database

### i. Diastolic Blood Pressure vs Body Mass Index





## ii. Age vs Diabetes Pedigree Function





#### 4. Link Google Drive dan Repository GitHub

##### a. Google Drive

<https://drive.google.com/drive/folders/1yGzasUzqF-kC2fjzoLW7yJIx6RZ8flm1?usp=sharing>

##### b. Repository GitHub

<https://github.com/rayhankinan/convex-hull-visualizer>

#### 5. Checklist

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