DAFTAR PUSTAKA

[1] Pusdatin.kemkes.go.id,"Kesehatan Gigi Nasional", September 2019. <https://www.kemkes.go.id/resources/download/pusdatin/infodatin/infodatin%20gigi.pdf />[Diakses, 8 November 2020).

[2] Lais de Fatima. M., et.al, 2018, *Tongue Pressure and Endurance in Patients with Class II and Class II Malocclusion*, Rev CEFAC Vol. 20 No. 2.

[3] Cristina Bezerra dos Santos, Elaine., et.al, 2019, *Quantitative evaluation of tongue pressure in children with oral breathing*, , Rev CEFAC Vol. 21 No.2.

[4] Jong-Hoon Moon, et. al, 2018, *The Effect of Tongue Pressure Strength and Accuracy Training on Tongue Pressure Strength, Swallowing Function and Quality of Life in Subacture Stroke Patients with Dysphagia: A Preliminary Randomized Clinical Trial*, International Journal of Rehabilitation Research 41(3).

[5] Interlink Electronics. 2010. FSR 204. www.interlinkelectronics.com (accessed 26 Maret 2021).

[6] Proffit, W.R. & Fields, H.W. 2000. *Contemporary Orthodontics. 4th Edition.* Mosby Inc., St. Louis. h. 151-158, 218 – 220, 282 – 283).

[7] Perkins, R. E. , Blanton, P. L. , & Biggs, N. L. 1977. *Electromyographic analysis of the "buccinator mechanism" in human beings*. Journal of Dental Research, 56(7), 783–794.

[8] Chigira, A. , Omoto, K. , Mukai, Y. , & Kaneko, Y. 1994. *SummerLip closing pressure in disabled children: A comparison with normal children*. *Dysphagia*, 9(3), 193–198.

[9] Yoneyama, T. , Yoshida, M. , Ohrui, T. , Mukaiyama, H. , Okamoto, H. , Hoshiba, K. ,Sasaki, H. 2002. *Oral care reduces pneumonia in older patients in nursing home*s. Journal of the American Geriatrics Society, 50(3), 430–433.

[10] Gabre, P. , Norrman, C. , & Birkhed, D. 2005. *Oral sugar clearance in individuals with oral motor dysfunctions.* Caries Research, 39(5), 357–362.

[11] Axelsson, K. , Norberg, A. , & Asplund, K. 1984. *Eating after a stroke—towards an integrated view*. *International Journal of Nursing Studies*, 21(2), 93–99.

[12] Millwood, J. , & Fiske, J. 2001. *Lip-biting in patients with profound neuro-disability*. Dental Update, 28(2), 105–108.

[13] Barlow, S. M. , & Rath, E. M. 1985. *Maximum voluntary closing forces in the upper and lower lips of humans.* Journal of Speech and Hearing Research, 28(3), 373–376.

[14] Herlin, Rendi, 2017, Rancang bangun *Prototipe* pengurang kadar air pada ampas tahu berbasis Microcontroller Arduino Mega 2560 . Skripsi thesis, Universitas Islam Negeri Sultan Syarif Kasim Riau.

[15] Nuryanto. R, 2015, *Pengukur Berat dan Tinggi Badan Ideal Berbasis Arduino. Karya Ilmiah Program Sarjana*. Surakarta: Universitas Muhammadiyah Surakarta.

[16] Bambang. Y., Pulong Nugroho. S., Herianto, 2015, *Pengembangan Model Public Monitoring System Menggunakan Raspberry Pi*, Universitas Pembangunan Nasional "Veteran" Yogyakarta Vol. 12, No. 02, Juli, 2015, Pp. 123 – 133.

[17] Ashari, Eko Yasin, 2018, *Perancangan Pintu Otomatis Menggunakan Pola Ketukan Berbasis Arduino.* Undergraduate Thesis, Universitas 17 Agustus 1945.

[18] Pazriyah, Depi. 2017. *Penggunaan raspberry pi dalam mendeteksi warna melalui webcam.* Other thesis, Politeknik Negeri Sriwijaya.

[19] https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf[Diakses 3 juli 2021, 10:18:00 WITA]

[20] Yuwono, Mochammad Wisang. 2019. *Rancang bangun system pengukuran regangan pada alat percobaan bejana tekan menggunakan mikrokontroller Arduino uno*. Undergraduate (S1) thesis, University of Muhammadiyah Malang.

[21] Putra, Indra Yusrianto. 2020. *Klasifikasi tumor otak miningioma, glioma dan pituitari dengan menggunakan convolutional neural network*. Undergraduate (S1) thesis, Universitas Muhammadiyah Malang.

[22] Harun, Achmad Muh. 2007. *Koreksi Protrusif Dengan Oral Screen Pada Anak Sebagai Tahap Terapi Awal Maloklusi Klas II Divisi 1*. Fakultas Kedokteran Gigi, Universitas Padjajaran Bandung.

LAMPIRAN

1. Surat validasi Dokter

Surat Keterangan

Yang bertanda tangan di bawah ini:

Nama	: Prof. Dr. drg. Muh. Harun Achmad, M.Kes., Sp.KGA(K)
NIP	197105232002121002
Jabatan	: Guru Besar
Instansi	: Fakultas Kedokteran Gigi, Universitas Hasanuddin

Menerangkan bahwa saudara berikut:

Nama	: Afif Fahmi Misbahuddin
NIM	: D041171504
Program Studi	: S1 Teknik Elektro, Universitas Hasanuddin

Telah melakukan penelitian dengan judul **RANCANG BANGUN ALAT *LIP AND TONGUE PREASSURE* UNTUK TEKNOLOGI KESEHATAN GIGI** dan melakukan simulasi prototype alat tersebut pada Panti Asuhan yang berada di Perumahan Murina Regency, Jalan Nipa-Nipa 1, Kecamatan Manggala.

Dengan demikian data penelitian yang bersangkutan telah saya validasi dan prototype alat dapat dipergunakan dengan baik.

Demikian pernyataan ini saya buat untuk dapat dipergunakan dengan baik.

Makassar, 09 Juli 2021

Prof. Dr. drg. Muh. Harun Achmad, M.Kes., Sp.KGA(K)
NIP. 197105232002121002

Surat Validasi Data

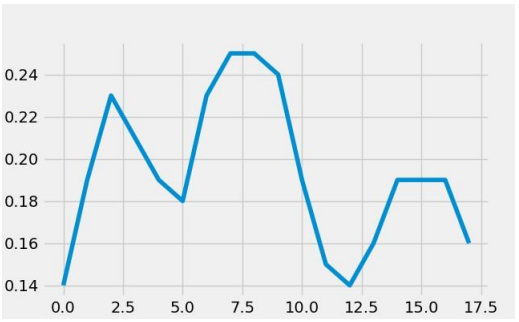
2. Data Pasien sebelum diolah

Data Pasien

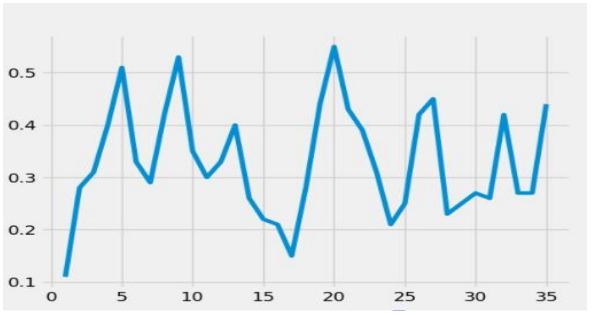
Pengkuran Tekanan Bibir

Pasien Terindikasi Maloklusi	Pasien Normal

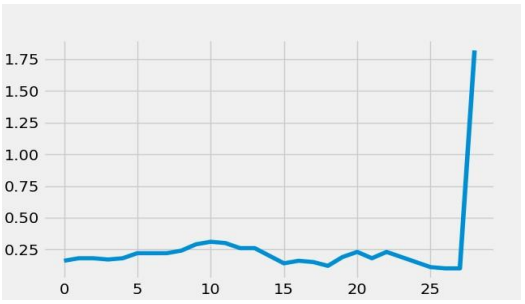
Time(S)	AS	IK	N A	Ir	Az	Ar	Nab	Faw
1	0.14	0.11	0.11	0.16	0.15	0.56	0.25	0.11
2	0.19	0.28	0.14	0.18	0.57	0.52	0.39	0.45
3	0.23	0.31	0.12	0.18	0.51	0.64	0.18	0.34
4	0.21	0.4	0.12	0.17	0.64	0.61	0.41	0.44
5	0.19	0.51	0.24	0.18	0.61	0.6	0.11	0.41
6	0.18	0.33	0.19	0.22	0.63	0.32	0.22	0.39
7	0.23	0.29	0.25	0.22	0.6	0.44	0.33	0.56
8	0.25	0.42	0.28	0.22	0.35	0.52	0.2	0.64
9	0.25	0.53	0.31	0.24	0.46	0.35	0.13	0.7
10	0.24	0.35	0.44	0.29	0.52	0.52	0.15	0.7
11	0.19	0.3	0.45	0.31	0.36	0.3	0.24	0.7
12	0.15	0.33	0.46	0.3	0.51		0.34	0.73
13	0.14	0.4	0.51	0.26	0.31		0.16	0.77
14	0.16	0.26	0.53	0.26			0.12	0.84
15	0.19	0.22	0.48	0.2			0.23	0.77
16	0.19	0.21	0.44	0.14			0.34	0.71
17	0.19	0.15	0.43	0.16			0.16	0.67
18	0.16	0.28	0.48	0.15			0.14	0.68
19		0.44	0.47	0.12			0.26	0.77
20		0.55	0.34	0.19			0.2	0.83
21		0.43	0.74	0.23			0.25	
22		0.39	0.66	0.18			0.13	
23		0.31	0.16	0.23			0.15	
24		0.21	0.33	0.19			0.18	
25		0.25	0.23	0.15				
26		0.42	0.15	0.11				
27		0.45	0.17	0.1				
28		0.23	0.42	0.1				
29		0.25	0.38	1.82				
30		0.27	0.39	1.82				
31		0.26	0.36	1.6				
32		0.42	0.39	1.37				
33		0.27	0.44					
34		0.27						
35		0.44						



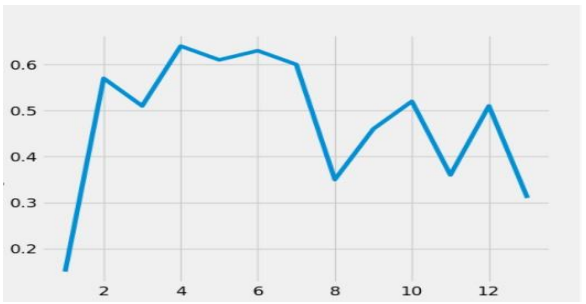
A/n As



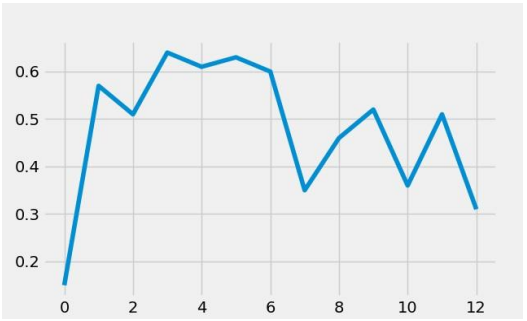
A/n Ik



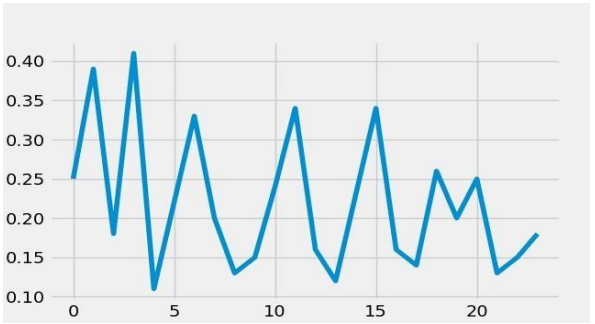
A/n Ir



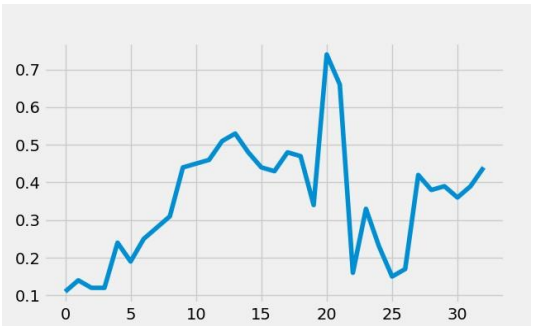
A/n Az



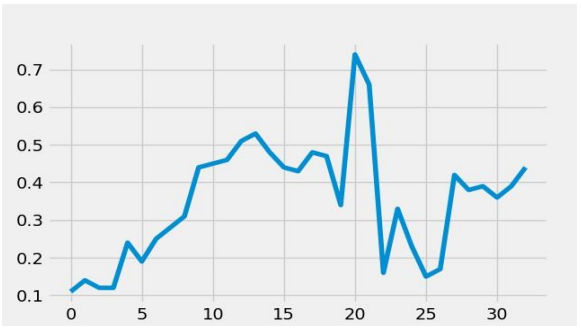
A/n Ar



A/n Nab



A/n NA

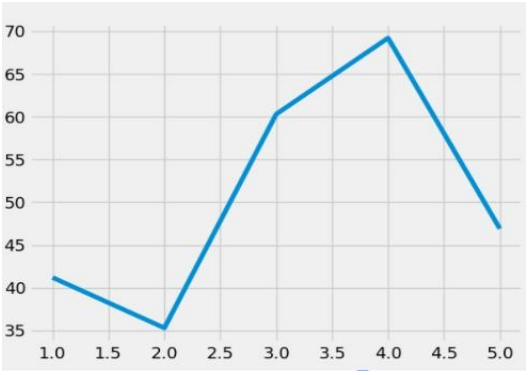


A/N Faw

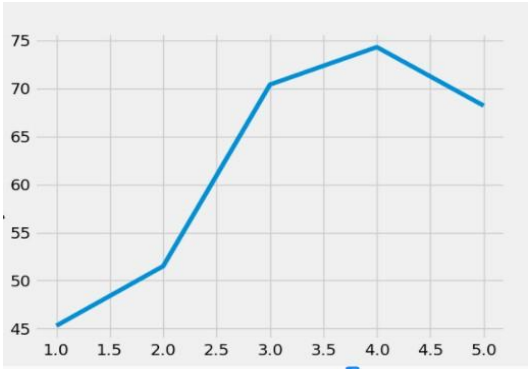
Pasien Untuk Tongue Pressure

Titik 5mm posterior papilla insisivus

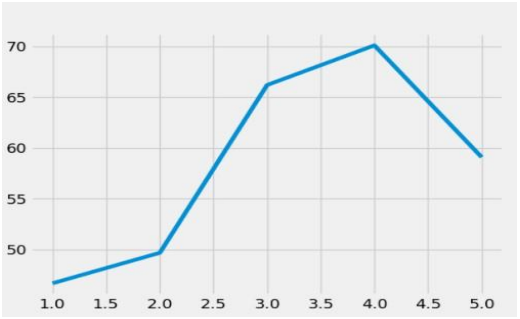
time(s)	IK	NA	AS	AR
1	41.2	45.3	46.7	20.3
2	35.3	51.5	49.7	32.2
3	60.3	70.4	66.2	31.7
4	69.2	74.3	70.1	40
5	46.9	68.2	59.1	35.3
Mean	50.58	61.94	58.36	31.9



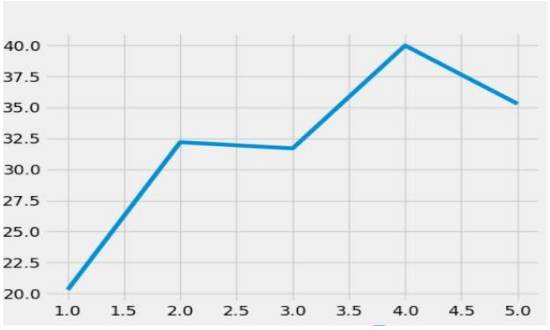
A/n IK



A/n Na



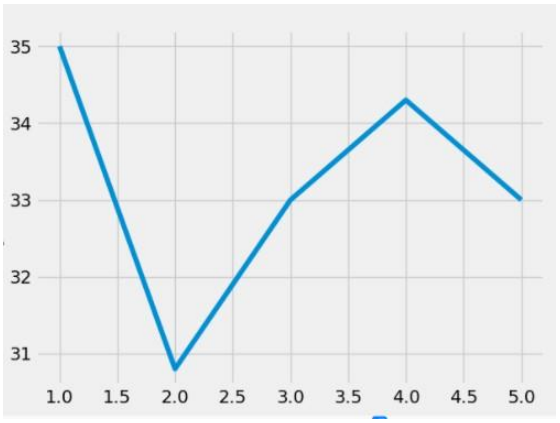
A/n As



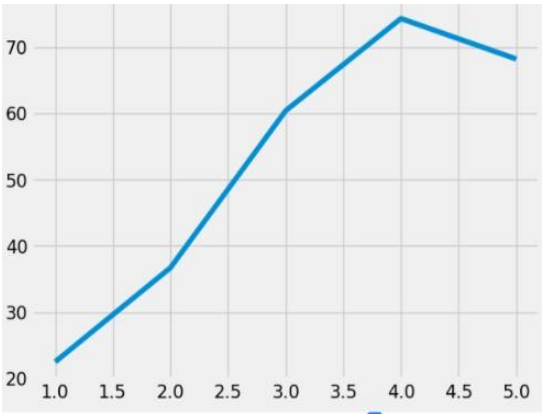
A/n Ar

terletak pada sepertiga posterior antara incisive papilla
dan lekukan hamuller disisi kanan

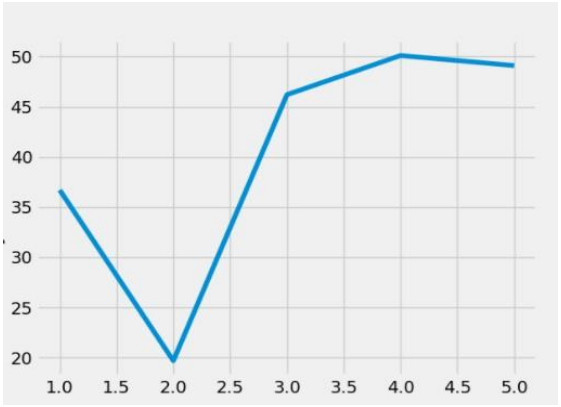
time(s)	IK	NA	AS	AR
1	35	22.5	36.7	30.3
2	30.8	36.7	19.7	22.2
3	33	60.4	46.2	28,1
4	34.3	74.3	50.1	15
5	33	68.2	49.1	33,2
Mean	33.22	52.42	42.36	22.5



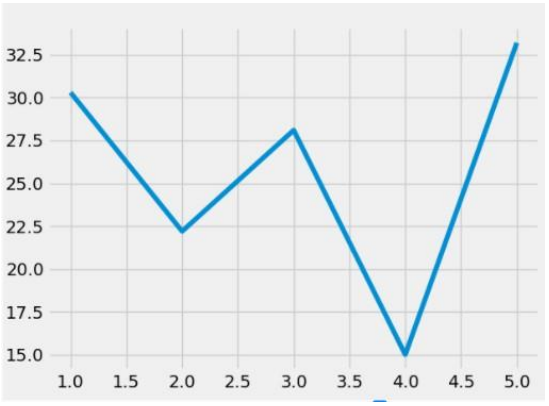
A/n IK



A/n Na



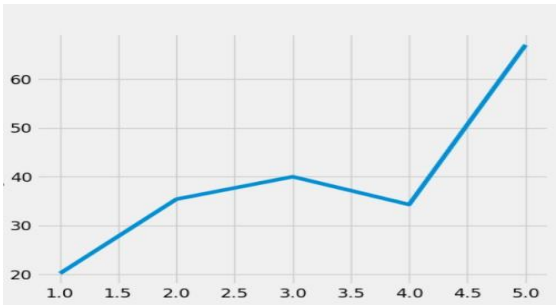
A/n As



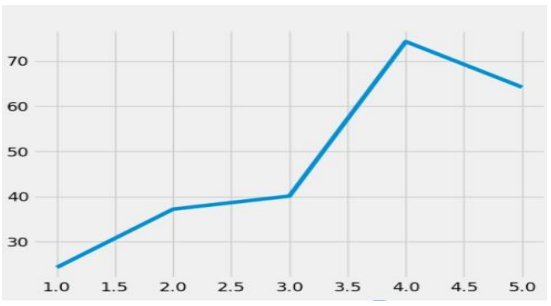
A/n Ar

sepertiga posterior antara papilla tajam dan takik hamuller disisi kiri

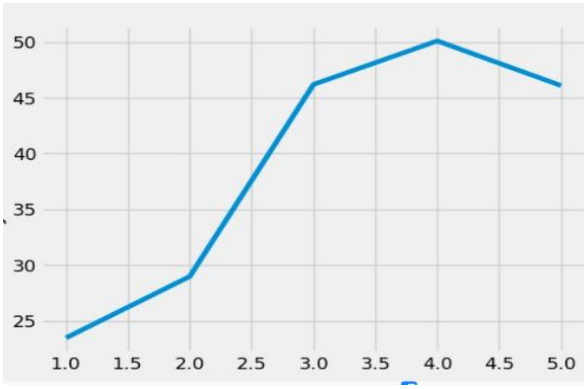
time(s)	IK	NA	AS	AR
1	20.2	24.3	23.5	33.1
2	35.4	37.2	29	22
3	40	40.1	46.2	31.8
4	34.3	74.3	50.1	20
5	67	64.2	46.1	21,2
Mean	39.38	48.02	38.98	26.725



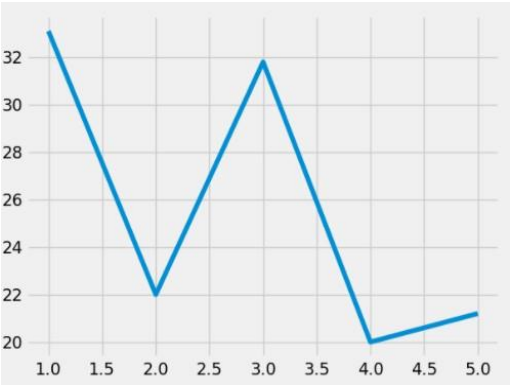
A/n IK



A/n Na


50
45
40
35
30
25
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

A/n As


32
30
28
26
24
22
20
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

A/n Ar

3. Program Untuk Tongue Pressure

```
#!/usr/bin/env python3
import tkinter
from matplotlib.backends.backend_tkagg
import (
    FigureCanvasTkAgg,
NavigationToolbar2Tk)
# Implement the default Matplotlib key
bindings.
from matplotlib.backend_bases import
key_press_handler
from matplotlib.figure import Figure
import matplotlib.animation as animation
from matplotlib import style
import csv
import pandas as pd
from gpiozero import MCP3008
import time

import numpy as np

style.use("fivethirtyeight")

sensor = MCP3008(3)
running = False
i = 1

x_value = 0
fieldnames = ["x_value", "sensor"]

with open ('data.csv', 'w') as csv_file:
    csv_writer = csv.DictWriter(csv_file,
fieldnames=fieldnames)
    csv_writer.writeheader()
    csv_writer.writerow({'x_value': 0.0,
'sensor': 0.0})

def scan():
    global x_value, running, i
    if running:
        fix_data = sensor.value
        fix_data = 1.00000 -
float(sensor.value)
        ebit_fix_data = fix_data * 21.42000
        ebit_fix_data = float(ebit_fix_data)

        if i == 1 :
            df = pd.read_csv('data.csv')
            data = 100
            df.at[0, 'sensor'] = ebit_fix_data
            df.at[0, 'x_value'] = 0
            df.to_csv('data.csv', index=False)
            x_value +=1
            i +=1
            print (ebit_fix_data)
        else :
            with open('data.csv', 'a') as
csv_file:

csv_writer=csv.DictWriter(csv_file,
fieldnames=fieldnames)
            info={
                "x_value": x_value,
                "sensor": ebit_fix_data
            }

            csv_writer.writerow(info)
            print(x_value, ebit_fix_data)

            x_value +=1
    root.after(1000, scan)

def start():
    """Enable scanning by setting the
global flag to True."""
    global running
    running = True

def stop():
    """Stop scanning by setting the global
flag to False."""
    global running
    running = False

fig = Figure(figsize=(5, 4), dpi=100)
a = fig.add_subplot(111)
pullData = pd.read_csv('data.csv')

def animate(i):
    pullData = pd.read_csv('data.csv')
    x  = pullData['x_value']
    y1 = pullData['sensor']
    a.clear()
    a.plot(x,y1)

root = tkinter.Tk()
root.wm_title("Embedding in Tk")

canvas =  FigureCanvasTkAgg(fig,
master=root)  # A tk.DrawingArea.
canvas.draw()

# pack_toolbar=False will make it easier
to use a layout manager later on.
toolbar = NavigationToolbar2Tk(canvas,
root)
toolbar.update()
```

```
#canvas.mpl_connect("key_press_event",
lambda event: print(f"you pressed
{event.key}"))
#canvas.mpl_connect("key_press_event",
key_press_handler)

button = tkinter.Button(master=root,
text="Quit", command=root.quit)
button.pack(side=tkinter.BOTTOM)

button2 = tkinter.Button(master=root,
text="Scan", command=start)
button2.pack(side=tkinter.BOTTOM)


button3 = tkinter.Button(master=root,
text="Stop", command=stop)
button3.pack(side=tkinter.BOTTOM)

# Packing order is important. Widgets are
processed sequentially and if there
# is no space left, because the window is
too small, they are not displayed.
# The canvas is rather flexible in its size,
so we pack it last which makes
# sure the UI controls are displayed as
long as possible.

toolbar.pack(side=tkinter.BOTTOM,
fill=tkinter.X)
canvas.get_tk_widget().pack(side=tkinter.
TOP, fill=tkinter.BOTH, expand=1)

ani = animation.FuncAnimation(fig,
animate, interval=100)
root.after(1000, scan)
tkinter.mainloop()
```

4. Program untuk lip

```
import os
import time
import Adafruit_GPIO.SPI as SPI
import Adafruit_MCP3008
import matplotlib.pyplot as plt
import matplotlib.animation as
animation
from matplotlib import style
import threading
style.use("ggplot")

os.remove('./data/tekanan.txt')
open('./data/tekanan.txt', 'a').close()

CLK  = 11
MISO = 9
MOSI = 10
CS   = 8

mcp =
Adafruit_MCP3008.MCP3008(clk=C
LK, cs=CS, miso=MISO,
mosi=MOSI)

fig, ax = plt.subplots()
style.use('fivethirtyeight')
def animate(i):
   with open("./data/tekanan.txt", "r")
as f:

      d = f.readlines()
   if(len(d) > 0):
      y = [int(i) for i in range(len(d))]

      s1 = []
      s2 = []
      s3 = []
      s4 = []

      for data in d:
         tmp = data.split(",")
         s1.append(str(tmp[0]))
         s2.append(str(tmp[1]))
         s3.append(str(tmp[2]))
         s4.append(str(tmp[3]))

      ax.plot(y, s1, color="red")
      ax.plot(y, s2, color="green")
      ax.plot(y, s3, color="blue")
      ax.plot(y, s4, color="yellow")

def sensor():
   while True:
      values = [0]*8
      for i in range(8):
         values[i] = mcp.read_adc(i)

      sens_1 = values[0];
      sens_2 = values[1];
      sens_3 = values[2];
      sens_4 = values[3];

      if sens_1 != "" and sens_2 != ""
and sens_3 != "" and sens_4 != "":
         data = str(sens_1) + "," +
str(sens_2) + "," + str(sens_3) + "," +
str(sens_4) + "\n"

         with open("./data/tekanan.txt",
"a") as f:
            f.write(data)
```

```
    time.sleep(0.1)

t = threading.Thread(target=sensor)
t.start()
ani = animation.FuncAnimation(fig,

animate, interval=50)
plt.show()
```

5. Program Hx711

```
    """
    This file holds HX711 class
    """
    #!/usr/bin/env python3

    import statistics as stat
    import time

    import RPi.GPIO as GPIO


    class HX711:
        """
        HX711 represents chip for
    reading load cells.
        """

        def __init__(self,
                     dout_pin,
                     pd_sck_pin,
                     gain_channel_A=128,
                     select_channel='A'):
            """
            Init a new instance of HX711

            Args:
                dout_pin(int): Raspberry
    Pi pin number where the Data pin
    of HX711 is connected.
                pd_sck_pin(int):
    Raspberry Pi pin number where
    the Clock pin of HX711 is
    connected.
                gain_channel_A(int):
    Optional, by default value 128.
    Options (128 || 64)
                select_channel(str):
    Optional, by default 'A'. Options
    ('A' || 'B')

            Raises:
                TypeError: if pd_sck_pin
    or dout_pin are not int type
            """
            if (isinstance(dout_pin, int)):
                if (isinstance(pd_sck_pin,
    int)):
                    self._pd_sck =

    pd_sck_pin
                    self._dout = dout_pin
                else:
                    raise
    TypeError('pd_sck_pin must be
    type int. '
                              'Received
    pd_sck_pin:
    {}'.format(pd_sck_pin))
            else:
                raise TypeError('dout_pin
    must be type int. '
                                'Received
    dout_pin: {}'.format(dout_pin))

            self._gain_channel_A = 0
            self._offset_A_128 = 0  #
    offset for channel A and gain 128
            self._offset_A_64 = 0  #
    offset for channel A and gain 64
            self._offset_B = 0  # offset
    for channel B
            self._last_raw_data_A_128 =
    0
            self._last_raw_data_A_64 =
    0
            self._last_raw_data_B = 0
            self._wanted_channel = ''
            self._current_channel = ''
            self._scale_ratio_A_128 = 1
    # scale ratio for channel A and
    gain 128
            self._scale_ratio_A_64 = 1  #
    scale ratio for channel A and gain
    64
            self._scale_ratio_B = 1  #
    scale ratio for channel B
            self._debug_mode = False
            self._data_filter =
    outliers_filter  # default it is used
    outliers_filter

            GPIO.setup(self._pd_sck,
    GPIO.OUT)  # pin _pd_sck is
    output only
            GPIO.setup(self._dout,
    GPIO.IN)  # pin _dout is input
    only
```

```
self.select_channel(select_channel
)

self.set_gain_A(gain_channel_A)

    def select_channel(self,
channel):

        """
        select_channel method
evaluates if the desired channel

        is valid and then sets the
_wanted_channel variable.

        Args:
            channel(str): the channel
to select. Options ('A' || 'B')
        Raises:
            ValueError: if channel is
not 'A' or 'B'
        """
        channel =
channel.capitalize()
        if (channel == 'A'):
            self._wanted_channel = 'A'
        elif (channel == 'B'):
            self._wanted_channel = 'B'
        else:
            raise
ValueError('Parameter "channel"
has to be "A" or "B". '
                           'Received:
{}'.format(channel))
        # after changing channel or
gain it has to wait 50 ms to allow
adjustment.
        # the data before is garbage
and cannot be used.
        self._read()
        time.sleep(0.5)

    def set_gain_A(self, gain):
        """
        set_gain_A method sets gain
for channel A.

        Args:
            gain(int): Gain for channel
A (128 || 64)

        Raises:
            ValueError: if gain is
different than 128 or 64

        """
            self._gain_channel_A =
gain
        elif gain == 64:
            self._gain_channel_A =
gain
        else:
            raise ValueError('gain has

to be 128 or 64. '
                             'Received:
{}'.format(gain))

        # after changing channel or
gain it has to wait 50 ms to allow
adjustment.
        # the data before is garbage
and cannot be used.
        self._read()
        time.sleep(0.5)

    def zero(self, readings=30):
        """
        zero is a method which sets
the current data as
        an offset for particulart
channel. It can be used for
        subtracting the weight of the
packaging. Also known as tare.

        Args:
            readings(int): Number of
readings for mean. Allowed
values 1..99

        Raises:
            ValueError: if readings are
not in range 1..99

        Returns: True if error
occured.
        """
        if readings > 0 and readings
< 100:
            result =
self.get_raw_data_mean(readings)
            if result != False:
                if (self._current_channel
== 'A' and

self._gain_channel_A == 128):
                    self._offset_A_128 =
result
                    return False
                elif

                if gain == 128:
```

```
(self._current_channel
== 'A' and
```

```
self._gain_channel_A == 64):
                self._offset_A_64 =
result
                return False

            elif
(self._current_channel == 'B'):
                self._offset_B =
result
                return False
            else:
                if self._debug_mode:
                    print('Cannot
zero() channel and gain
mismatch.\n'
                          'current
channel: {}\n'
                          'gain A:
{}\n'.format(self._current_channel
,

self._gain_channel_A))
                return True
        else:
            if self._debug_mode:
                print('From method
"zero()".\n'

'get_raw_data_mean(readings)
returned False.\n')
            return True
    else:
        raise
ValueError('Parameter "readings"
'
                         'can be in range
1 up to 99. '
                         'Received:
{}'.format(readings))

  def set_offset(self, offset,
channel='', gain_A=0):
    """

    set offset method sets desired
offset for specific
    channel and gain. Optional,
by default it sets offset for current
    channel and gain.

    Args:
        offset(int): specific offset
for channel
        channel(str): Optional, by
default it is the current channel.
            Or use these options ('A'
|| 'B')

    Raises:
        ValueError: if channel is

not ('A' || 'B' || '')
        TypeError: if offset is not
int type
    """
    channel =
channel.capitalize()
    if isinstance(offset, int):
        if channel == 'A' and
gain_A == 128:
            self._offset_A_128 =
offset
            return
        elif channel == 'A' and
gain_A == 64:
            self._offset_A_64 =
offset
            return
        elif channel == 'B':
            self._offset_B = offset
            return
        elif channel == '':
            if self._current_channel
== 'A' and self._gain_channel_A
== 128:
                self._offset_A_128 =
offset
                return
            elif
self._current_channel == 'A' and
self._gain_channel_A == 64:
                self._offset_A_64 =
offset
                return
            else:
                self._offset_B =
offset
                return
        else:

            raise
ValueError('Parameter "channel"
has to be "A" or "B". '
                       'Received:
{}'.format(channel))
    else:
        raise
TypeError('Parameter "offset" has
to be integer. '
                  'Received: ' +
str(offset) + '\n')
```

```
    def set_scale_ratio(self,
scale_ratio, channel='',
gain_A=0):
        """
        set_scale_ratio method sets
the ratio for calculating
        weight in desired units. In
order to find this ratio for
        example to grams or kg. You
must have known weight.

        Args:
            scale_ratio(float): number
> 0.0 that is used for
                conversion to weight
units
            channel(str): Optional, by
default it is the current channel.
                Or use these options
('a'|| 'A' || 'b' || 'B')
            gain_A(int): Optional, by
default it is the current channel.
                Or use these options
(128 || 64)
        Raises:
            ValueError: if channel is
not ('A' || 'B' || '')
            TypeError: if offset is not
int type """
        channel =
channel.capitalize()
        if isinstance(gain_A, int):
            if channel == 'A' and
gain_A == 128:
                self._scale_ratio_A_128
= scale_ratio
                return
            elif channel == 'A' and
gain_A == 64:
                self._scale_ratio_A_64
= scale_ratio
                return
            elif channel == 'B':
                self._scale_ratio_B =
scale_ratio
                return
            elif channel == '':
                if self._current_channel
== 'A' and self._gain_channel_A
== 128:

self._scale_ratio_A_128 =
scale_ratio
                    return
                elif
self._current_channel == 'A' and
self._gain_channel_A == 64:

self._scale_ratio_A_64 =
scale_ratio
                    return
                else:
                    self._scale_ratio_B =
scale_ratio
                    return
            else:
                raise
ValueError('Parameter "channel"
has to be "A" or "B". '
                           'received:
{}'.format(channel))
        else:
            raise
TypeError('Parameter "gain_A"
has to be integer. '
                      'Received: ' +
str(gain_A) + '\n')

    def set_data_filter(self,
data_filter):
        """
        set_data_filter method sets
data filter that is passed as an
argument.

        Args:
            data_filter(data_filter):
Data filter that takes list of int
numbers and
                returns a list of filtered
int numbers.

        Raises:
            TypeError: if filter is not a
function.
        """
        if callable(data_filter):
            self._data_filter =
data_filter
        else:
            raise
TypeError('Parameter
"data_filter" must be a function. '
                      'Received:
{}'.format(data_filter))

    def set_debug_mode(self,
```

```
flag=False):
        """
        set_debug_mode method is
for turning on and off
        debug mode.


        Args:
            flag(bool): True turns on
the debug mode. False turns it off.

        Raises:
            ValueError: if fag is not
bool type
        """
        if flag == False:
            self._debug_mode = False
            print('Debug mode
DISABLED')
            return
        elif flag == True:
            self._debug_mode = True
            print('Debug mode
ENABLED')
            return
        else:
            raise
ValueError('Parameter "flag" can
be only BOOL value. '
                       'Received:
{}'.format(flag))


    def _save_last_raw_data(self,
channel, gain_A, data):
        """
        _save_last_raw_data saves
the last raw data for specific
channel and gain.

        Args:
            channel(str):
            gain_A(int):
            data(int):
        Returns: False if error
occured
        """
        if channel == 'A' and gain_A
== 128:

self._last_raw_data_A_128 = data
        elif channel == 'A' and
gain_A == 64:
            self._last_raw_data_A_64
= data
        elif channel == 'B':
            self._last_raw_data_B =
data
        else:
            return False


    def _ready(self):
        """
        _ready method check if data
is prepared for reading from
HX711

        Returns: bool True if ready
else False when not ready
        """
        # if DOUT pin is low data is
ready for reading
        if GPIO.input(self._dout) ==
0:
            return True
        else:
            return False

    def _set_channel_gain(self,
num):
        """
        _set_channel_gain is called
only from _read method.
        It finishes the data
transmission for HX711 which
sets

        the next required gain and
channel.

        Args:
            num(int): how many ones
it sends to HX711
                options (1 || 2 || 3)

        Returns: bool True if HX711
is ready for the next reading
            False if HX711 is not
ready for the next reading
        """
        for _ in range(num):
            start_counter =
time.perf_counter()
            GPIO.output(self._pd_sck,
True)
            GPIO.output(self._pd_sck,
False)
            end_counter =
time.perf_counter()
            # check if hx 711 did not
turn off...
```

```
            if end_counter -
start_counter >= 0.00006:
                # if pd_sck pin is HIGH
for 60 us and more than the HX
711 enters power down mode.
                if self._debug_mode:
                    print('Not enough fast
while setting gain and channel')
                    print(
                        'Time elapsed:
{}'.format(end_counter -
start_counter))
                # hx711 has turned off.
First few readings are inaccurate.
                # Despite it, this reading
was ok and data can be used.
                result =
self.get_raw_data_mean(6)  # set
for the next reading.
                if result == False:
                    return False
        return True

    def _read(self):
        """
        _read method reads bits from
hx711, converts to INT
        and validate the data.

        Returns: (bool || int) if it
returns False then it is false
reading.
            if it returns int then the
reading was correct
        """
        GPIO.output(self._pd_sck,
False)  # start by setting the
pd_sck to 0
        ready_counter = 0
        while (not self._ready() and
ready_counter <= 40):
            time.sleep(0.01)  # sleep
for 10 ms because data is not
ready
            ready_counter += 1
            if ready_counter == 50:  #
if counter reached max value then
return False
                if self._debug_mode:
                    print('self._read() not
ready after 40 trials\n')
                return False

        # read first 24 bits of data
        data_in = 0  # 2's
complement data from hx 711
        for _ in range(24):
            start_counter =
time.perf_counter()
            # request next bit from hx
711
            GPIO.output(self._pd_sck,
True)
            GPIO.output(self._pd_sck,
False)
            end_counter =
time.perf_counter()
            if end_counter -
start_counter >= 0.00006:  #
check if the hx 711 did not turn
off...
                # if pd_sck pin is HIGH
for 60 us and more than the HX
711 enters power down mode.
                if self._debug_mode:
                    print('Not enough fast
while reading data')
                    print(
                        'Time elapsed:
{}'.format(end_counter -
start_counter))
                return False
            # Shift the bits as they
come to data_in variable.
            # Left shift by one bit then
bitwise OR with the new bit.
            data_in = (data_in << 1) |
GPIO.input(self._dout)

        if self._wanted_channel ==
'A' and self._gain_channel_A ==
128:
            if not
self._set_channel_gain(1):  # send
only one bit which is 1
                return False  # return
False because channel was not set
properly
            else:
                self._current_channel =
'A'  # else set current channel
variable
                self._gain_channel_A =
128  # and gain
        elif self._wanted_channel ==
'A' and self._gain_channel_A ==
64:
            if not
```

```
self._set_channel_gain(3):  # send three ones
                return False  # return False because channel was not set properly
            else:
                self._current_channel = 'A'  # else set current channel variable

                self._gain_channel_A = 64
        else:

            if not self._set_channel_gain(2):  # send two ones
                return False  # return False because channel was not set properly
            else:
                self._current_channel = 'B'  # else set current channel variable

        if self._debug_mode:  # print 2's complement value
            print('Binary value as received: {}\n'.format(bin(data_in)))

        #check if data is valid
        if (data_in == 0x7fffff
                or  # 0x7fffff is the highest possible value from hx711
                data_in == 0x800000
            ):  # 0x800000 is the lowest possible value from hx711
            if self._debug_mode:
                print('Invalid data detected: {}\n'.format(data_in))
            return False  # rturn false because the data is invalid

        # calculate int from 2's complement
        signed_data = 0
        # 0b1000 0000 0000 0000 0000 0000 check if the sign bit is 1. Negative number.
        if (data_in & 0x800000):
            signed_data = -(
                (data_in ^ 0xffffff) + 1)  # convert from 2's complement to int
        else:  # else do not do anything the value is positive number
            signed_data = data_in

        if self._debug_mode:
            print('Converted 2\'s complement value: {}\n'.format(signed_data))


        return signed_data

    def get_raw_data_mean(self, readings=30):
        """
        get_raw_data_mean returns mean value of readings.

        Args:
            readings(int): Number of readings for mean.

        Returns: (bool || int) if False then reading is invalid.
            if it returns int then reading is valid
        """
        # do backup of current channel befor reading for later use
        backup_channel = self._current_channel
        backup_gain = self._gain_channel_A
        data_list = []
        # do required number of readings
        for _ in range(readings):

            data_list.append(self._read())
        data_mean = False
        if readings > 2 and self._data_filter:
            filtered_data = self._data_filter(data_list)
            if self._debug_mode:
                print('data_list: {}'.format(data_list))
                print('filtered_data list: {}'.format(filtered_data))
                print('data_mean:', stat.mean(filtered_data))
            data_mean = stat.mean(filtered_data)
        else:
            data_mean =
```

```
stat.mean(data_list)

self._save_last_raw_data(backup_
channel, backup_gain,
data_mean)
        return int(data_mean)

    def get_data_mean(self,
readings=30):
        """
 get_data_mean returns average
      value of readings minus
       offset for the channel which
was read.

        Args:
            readings(int): Number of
readings for mean

        Returns: (bool || int) False if
reading was not ok.
            If it returns int then
reading was ok
        """
        result =
self.get_raw_data_mean(readings)
        if result != False:
            if self._current_channel
== 'A' and self._gain_channel_A
== 128:
                return result -
self._offset_A_128
            elif self._current_channel
== 'A' and self._gain_channel_A
== 64:
                return result -
self._offset_A_64
            else:
                return result -
self._offset_B
        else:
            return False

    def get_weight_mean(self,
readings=30):
        """
        get_weight_mean returns
average value of readings minus
        offset divided by scale ratio
for a specific channel
        and gain.

        Args:
            readings(int): Number of
readings for mean

        Returns: (bool || float) False
if reading was not ok.
            If it returns float then
reading was ok
        """
        result =
self.get_raw_data_mean(readings)
        if result != False:
            if self._current_channel
== 'A' and self._gain_channel_A
== 128:
                return float(
                    (result -
self._offset_A_128) /
self._scale_ratio_A_128)
            elif self._current_channel
== 'A' and self._gain_channel_A
== 64:
                return float(
                    (result -
self._offset_A_64) /
self._scale_ratio_A_64)
            else:
                return float((result -
self._offset_B) /
self._scale_ratio_B)
        else:
            return False

    def get_current_channel(self):
        """
        get current channel returns
the value of current channel.

        Returns: ('A' || 'B')
        """
        return self._current_channel

    def get_data_filter(self):
        """
        get data filter.

        Returns: self._data_filter
        """
        return self._data_filter

    def get_current_gain_A(self):
        """
        get current gain A returns the
value of current gain on channel
A
```

```
        Returns: (128 || 64) current
gain on channel A
        """
        return self._gain_channel_A

    def get_last_raw_data(self,
channel='', gain_A=0):
        """
        get last raw data returns the
last read data for a
        channel and gain. By default
for current one.

        Args:
            channel(str): select
channel ('A' || 'B'). If not then it
returns the current one.
            gain_A(int): select gain
(128 || 64). If not then it returns
the current one.

        Raises:
            ValueError: if channel is
not ('A' || 'B' || '') or gain_A is not
(128 || 64 || 0)
                '' and 0 is default value.

        Returns: int the last data that
was received for the chosen
channel and gain
        """
        channel =
channel.capitalize()
        if channel == 'A' and gain_A
== 128:
            return
self._last_raw_data_A_128
        elif channel == 'A' and
gain_A == 64:
            return
self._last_raw_data_A_64
        elif channel == 'B':
            return
self._last_raw_data_B
        elif channel == '':
            if self._current_channel
== 'A' and self._gain_channel_A
== 128:
                return
self._last_raw_data_A_128
            elif self._current_channel
== 'A' and self._gain_channel_A
== 64:
                return
self._last_raw_data_A_64
            else:
                return
self._last_raw_data_B
        else:
            raise ValueError(
                'Parameter "channel"
has to be "A" or "B". '
                'Received: {}
\nParameter "gain_A" has to be
128 or 64. Received {}'
                .format(channel,
gain_A))

    def get_current_offset(self,
channel='', gain_A=0):
        """
        get current offset returns the
current offset for
        a particular channel and gain.
By default the current one.

        Args:
            channel(str): select for
which channel ('A' || 'B')
            gain_A(int): select for
which gain (128 || 64)

        Raises:
            ValueError: if channel is
not ('A' || 'B' || '') or gain_A is not
(128 || 64 || 0)
                '' and 0 is default value.

        Returns: int the offset for the
chosen channel and gain
        """
        channel =
channel.capitalize()
        if channel == 'A' and gain_A
== 128:
            return self._offset_A_128
        elif channel == 'A' and
gain_A == 64:
            return self._offset_A_64
        elif channel == 'B':
            return self._offset_B
        elif channel == '':
            if self._current_channel
== 'A' and self._gain_channel_A
== 128:
                return
self._offset_A_128
            elif self._current_channel
```

```
== 'A' and self._gain_channel_A
== 64:

                return self._offset_A_64
            else:
                return self._offset_B
        else:
            raise ValueError(
                'Parameter "channel"
has to be "A" or "B". '
                'Received: {}
\nParameter "gain_A" has to be
128 or 64. Received {}'
                .format(channel,
gain_A))

    def
get_current_scale_ratio(self,
channel='', gain_A=0):
        """
        get current scale ratio returns
the current scale ratio
        for a particular channel and
gain. By default
        the current one.

        Args:
            channel(str): select for
which channel ('A' || 'B')
            gain_A(int): select for
which gain (128 || 64)

        Returns: int the scale ratio
for the chosen channel and gain
        """
        channel =
channel.capitalize()
        if channel == 'A' and gain_A
== 128:
            return
self._scale_ratio_A_128
        elif channel == 'A' and
gain_A == 64:
            return
self._scale_ratio_A_64
        elif channel == 'B':
            return self._scale_ratio_B
        elif channel == '':
            if self._current_channel
== 'A' and self._gain_channel_A
== 128:
                return
self._scale_ratio_A_128
            elif self._current_channel
== 'A' and self._gain_channel_A
== 64:
                return

self._scale_ratio_A_64
            else:
                return
self._scale_ratio_B
        else:
            raise ValueError(
                'Parameter "channel"
has to be "A" or "B". '
                'Received: {}
\nParameter "gain_A" has to be
128 or 64. Received {}'
                .format(channel,
gain_A))

    def power_down(self):
        """
        power down method turns
off the hx711. """
        GPIO.output(self._pd_sck,
False)
        GPIO.output(self._pd_sck,
True)
        time.sleep(0.01)

    def power_up(self):
        """
        power up function turns on
the hx711.
        """
        GPIO.output(self._pd_sck,
False)
        time.sleep(0.01)

    def reset(self):
        """
    reset method resets the hx711
      and prepare it for the next
reading.

        Returns: True if error
encountered
        """
        self.power_down()
        self.power_up()
        result =
self.get_raw_data_mean(6)
        if result:
            return False
        else:
            return True
```

```
def outliers_filter(data_list):
    """
    It filters out outliers from the
provided list of int.
    Median is used as an estimator
of outliers.

    Args:
        data_list([int]): List of int. It
can contain Bool False that is
removed.

    Returns: list of filtered data.
Excluding outliers.
    """
    data = []
    for num in data_list:
        if num:
            data.append(num)
    # set 'm' to lower value to
remove more outliers
    # set 'm' to higher value to keep
more data samples (also some
outliers)
    m = 2.0
    # It calculates the absolute
distance to the median.
    # Then it scales the distances
by their median value (again)
    # so they are on a relative scale
to 'm'.
    data_median =
stat.median(data)
    abs_distance = []
    for num in data:

abs_distance.append(abs(num -
data_median))
    mdev =
stat.median(abs_distance)
    s = []
    if mdev:
        for num in abs_distance:
            s.append(num / mdev)
    else:
        # mdev is 0. Therefore all
data samples in the list data have
the same value.
        return data
    filtered_data = []
    for i in range(len(data)):
        if s[i] < m:

filtered_data.append(data[i])
    return filtered_data
```

6. Dokumentasi Pengambilan data